                     TCP Congestion Window Validation

Abstract

   TCP's congestion window controls the number of packets a TCP flow may
   have in the network at any time.  However, long periods when the
   sender is idle or application-limited can lead to the invalidation of
   the congestion window, in that the congestion window no longer
   reflects current information about the state of the network.  This
   document describes a simple modification to TCP's congestion control
   algorithms to decay the congestion window cwnd after the transition
   from a sufficiently-long application-limited period, while using the
   slow-start threshold ssthresh to save information about the previous
   value of the congestion window.

   An invalid congestion window also results when the congestion window
   is increased (i.e., in TCP's slow-start or congestion avoidance
   phases) during application-limited periods, when the previous value
   of the congestion window might never have been fully utilized.  We
   propose that the TCP sender should not increase the congestion window
   when the TCP sender has been application-limited (and therefore has
   not fully used the current congestion window).  We have explored
   these algorithms both with simulations and with experiments from an
   implementation in FreeBSD.

1.  Conventions and Acronyms

   The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD,
   SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this
   document, are to be interpreted as described in [B97].

2. Introduction

   TCP's congestion window controls the number of packets a TCP flow may
   have in the network at any time.  The congestion window is set using
   an Additive-Increase, Multiplicative-Decrease (AIMD) mechanism that
   probes for available bandwidth, dynamically adapting to changing
   network conditions.  This AIMD mechanism works well when the sender
   continually has data to send, as is typically the case for TCP used
   for bulk-data transfer.  In contrast, for TCP used with telnet
   applications, the data sender often has little or no data to send,
   and the sending rate is often determined by the rate at which data is
   generated by the user.  With the advent of the web, including
   developments such as TCP senders with dynamically-created data and
   HTTP 1.1 with persistent-connection TCP, the interaction between
   application-limited periods (when the sender sends less than is
   allowed by the congestion or receiver windows) and network-limited
   periods (when the sender is limited by the TCP window) becomes
   increasingly important.  More precisely, we define a network-limited
   period as any period when the sender is sending a full window of
   data.

   Long periods when the sender is application-limited can lead to the
   invalidation of the congestion window.  During periods when the TCP
   sender is network-limited, the value of the congestion window is
   repeatedly "revalidated" by the successful transmission of a window
   of data without loss.  When the TCP sender is network-limited, there
   is an incoming stream of acknowledgements that "clocks out" new data,
   giving concrete evidence of recent available bandwidth in the
   network.  In contrast, during periods when the TCP sender is
   application-limited, the estimate of available capacity represented
   by the congestion window may become steadily less accurate over time.
   In particular, capacity that had once been used by the network-
   limited connection might now be used by other traffic.

   Current TCP implementations have a range of behaviors for starting up
   after an idle period.  Some current TCP implementations slow-start
   after an idle period longer than the RTO estimate, as suggested in
   [RFC2581] and in the appendix of [VJ88], while other implementations
   don't reduce their congestion window after an idle period.  RFC 2581
   [RFC2581] recommends the following: "a TCP SHOULD set cwnd to no more
   than RW [the initial window] before beginning transmission if the TCP
   has not sent data in an interval exceeding the retransmission
   timeout."  A proposal for TCP's slow-start after idle has also been
   discussed in [HTH98].  The issue of validation of congestion
   information during idle periods has also been addressed in contexts
   other than TCP and IP, for example in "Use-it or Lose-it" mechanisms
   for ATM networks [J96,J95].

To address the revalidation of the congestion window after a
application-limited period, we propose a simple modification to TCP's
congestion control algorithms to decay the congestion window cwnd
after the transition from a sufficiently-long application-limited
period (i.e., at least one roundtrip time) to a network-limited
period.  In particular, we propose that after an idle period, the TCP
sender should reduce its congestion window by half for every RTT that
the flow has remained idle.

When the congestion window is reduced, the slow-start threshold
ssthresh remains as "memory" of the recent congestion window.
Specifically, ssthresh is never decreased when cwnd is reduced after
an application-limited period; before cwnd is reduced, ssthresh is
set to the maximum of its current value, and half-way between the old
and the new values of cwnd.  This use of ssthresh allows a TCP sender
increasing its sending rate after an application-limited period to
quickly slow-start to recover most of the previous value of the
congestion window.  To be more precise, if ssthresh is less than 3/4
cwnd when the congestion window is reduced after an application-
limited period, then ssthresh is increased to 3/4 cwnd before the
reduction of the congestion window.

An invalid congestion window also results when the congestion window
is increased (i.e., in TCP's slow-start or congestion avoidance
phases) during application-limited periods, when the previous value
of the congestion window might never have been fully utilized.  As
far as we know, all current TCP implementations increase the
congestion window when an acknowledgement arrives, if allowed by the
receiver's advertised window and the slow-start or congestion
avoidance window increase algorithm, without checking to see if the
previous value of the congestion window has in fact been used.  This
document proposes that the window increase algorithm not be invoked
during application-limited periods [MSML99].  In particular, the TCP
sender should not increase the congestion window when the TCP sender
has been application-limited (and therefore has not fully used the
current congestion window).  This restriction prevents the congestion
window from growing arbitrarily large, in the absence of evidence
that the congestion window can be supported by the network.  From
[MSML99, Section 5.2]: "This restriction assures that [cwnd] only
grows as long as TCP actually succeeds in injecting enough data into
the network to test the path."

A somewhat-orthogonal problem associated with maintaining a large
congestion window after an application-limited period is that the
sender, with a sudden large amount of data to send after a quiescent
period, might immediately send a full congestion window of back-to-
back packets.  This problem of sending large bursts of packets back-
to-back can be effectively handled using rate-based pacing (RBP,

[VH97]), or using a maximum burst size control [FF96].  We would
contend that, even with mechanisms for limiting the sending of back-
to-back packets or pacing packets out over the period of a roundtrip
time, an old congestion window that has not been fully used for some
time can not be trusted as an indication of the bandwidth currently
available for that flow.  We would contend that the mechanisms to
pace out packets allowed by the congestion window are largely
orthogonal to the algorithms used to determine the appropriate size
of the congestion window.

3. Description

When a TCP sender has sufficient data available to fill the available
network capacity for that flow, cwnd and ssthresh get set to
appropriate values for the network conditions.  When a TCP sender
stops sending, the flow stops sampling the network conditions, and so
the value of the congestion window may become inaccurate.  We believe
the correct conservative behavior under these circumstances is to
decay the congestion window by half for every RTT that the flow
remains inactive.  The value of half is a very conservative figure
based on how quickly multiplicative decrease would have decayed the
window in the presence of loss.

Another possibility is that the sender may not stop sending, but may
become application-limited rather than network-limited, and offer
less data to the network than the congestion window allows to be
sent.  In this case the TCP flow is still sampling network
conditions, but is not offering sufficient traffic to be sure that
there is still sufficient capacity in the network for that flow to
send a full congestion window.  Under these circumstances we believe
the correct conservative behavior is for the sender to keep track of
the maximum amount of the congestion window used during each RTT, and
to decay the congestion window each RTT to midway between the current
cwnd value and the maximum value used.

Before the congestion window is reduced, ssthresh is set to the
maximum of its current value and 3/4 cwnd.  If the sender then has
more data to send than the decayed cwnd allows, the TCP will slow-
start (perform exponential increase) at least half-way back up to the
old value of cwnd.

The justification for this value of "3/4 cwnd" is that 3/4 cwnd is a
conservative estimate of the recent average value of the congestion
window, and the TCP should safely be able to slow-start at least up
to this point.  For a TCP in steady-state that has been reducing its
congestion window each time the congestion window reached some
maximum value 'maxwin', the average congestion window has been 3/4
maxwin.  On average, when the connection becomes application-limited,

cwnd will be 3/4 maxwin, and in this case cwnd itself represents the
average value of the congestion window.  However, if the connection
happens to become application-limited when cwnd equals maxwin, then
the average value of the congestion window is given by 3/4 cwnd.

An alternate possibility would be to set ssthresh to the maximum of
the current value of ssthresh, and the old value of cwnd, allowing
TCP to slow-start all of the way back up to the old value of cwnd.
Further experimentation can be used to evaluate these two options for
setting ssthresh.

For the separate issue of the increase of the congestion window in
response to an acknowledgement, we believe the correct behavior is
for the sender to increase the congestion window only if the window
was full when the acknowledgment arrived.

We term this set of modifications to TCP Congestion Window Validation
(CWV) because they are related to ensuring the congestion window is
always a valid reflection of the current network state as probed by
the connection.

## 3.1. The basic algorithm for reducing the congestion window

A key issue in the CWV algorithm is to determine how to apply the
guideline of reducing the congestion window once for every roundtrip
time that the flow is application-limited.  We use TCP's
retransmission timer (RTO) as a reasonable upper bound on the
roundtrip time, and reduce the congestion window roughly once per
RTO.

This basic algorithm could be implemented in TCP as follows: When TCP
sends a new packet it checks to see if more than RTO seconds have
elapsed since the previous packet was sent.  If RTO has elapsed,
ssthresh is set to the maximum of 3/4 cwnd and the current value of
ssthresh, and then the congestion window is halved for every RTO that
elapsed since the previous packet was sent.  In addition, T_prev is
set to the current time, and W_used is reset to zero.  T_prev will be
used to determine the elapsed time since the sender last was network-
limited or had reduced cwnd after an idle period.  When the sender is
application-limited, W_used holds the maximum congestion window
actually used since the sender was last network-limited.

The mechanism for determining the number of RTOs in the most recent
idle period could also be implemented by using a timer that expires
every RTO after the last packet was sent instead of a check per
packet - efficiency constraints on different operating systems may
dictate which is more efficient to implement.

After TCP sends a packet, it also checks to see if that packet filled
the congestion window.  If so, the sender is network-limited, and
sets the variable T_prev to the current TCP clock time, and the
variable W_used to zero.

When TCP sends a packet that does not fill the congestion window, and
the TCP send queue is empty, then the sender is application-limited.
The sender checks to see if the amount of unacknowledged data is
greater than W_used; if so, W_used is set to the amount of
unacknowledged data.  In addition TCP checks to see if the elapsed
time since T_prev is greater than RTO.  If so, then the TCP has not
just reduced its congestion window following an idle period.  The TCP
has been application-limited rather than network-limited for at least
an entire RTO interval, but for less than two RTO intervals.  In this
case, TCP sets ssthresh to the maximum of 3/4 cwnd and the current
value of ssthresh, and reduces its congestion window to
(cwnd+W_used)/2.  W_used is then set to zero, and T_prev is set to
the current time, so a further reduction will not take place until at
least another RTO period has elapsed.  Thus, during an application-
limited period the CWV algorithm reduces the congestion window once
per RTO.

3.2.  Pseudo-code for reducing the congestion window

```
    Initially:
        T_last = tcpnow, T_prev = tcpnow, W_used = 0

    After sending a data segment:
        If tcpnow - T_last >= RTO
            (The sender has been idle.)
            ssthresh =  max(ssthresh, 3*cwnd/4)
            For i=1  To (tcpnow - T_last)/RTO
                win =  min(cwnd, receiver's declared max window)
                cwnd =  max(win/2, MSS)
            T_prev = tcpnow
            W_used = 0

        T_last = tcpnow

        If window is full
            T_prev = tcpnow
            W_used = 0
        Else
            If no more data is available to send
                W_used =  max(W_used, amount of unacknowledged data)
                If tcpnow - T_prev >= RTO
                    (The sender has been application-limited.)
                    ssthresh =  max(ssthresh, 3*cwnd/4)
```

```
               win =  min(cwnd, receiver's declared max window)
               cwnd = (win + W_used)/2
               T_prev = tcpnow
               W_used = 0
```

4. Simulations

   The CWV proposal has been implemented as an option in the network
   simulator NS [NS].  The simulations in the validation test suite for
   CWV can be run with the command "./test-all-tcp" in the directory
   "tcl/test".  The simulations show the use of CWV to reduce the
   congestion window after a period when the TCP connection was
   application-limited, and to limit the increase in the congestion
   window when a transfer is application-limited.  As the simulations
   illustrate, the use of ssthresh to maintain connection history is a
   critical part of the Congestion Window Validation algorithm.  [HPF99]
   discusses these simulations in more detail.

5. Experiments

   We have implemented the CWV mechanism in the TCP implementation in
   FreeBSD 3.2.  [HPF99] discusses these experiments in more detail.

   The first experiment examines the effects of the Congestion Window
   Validation mechanisms for limiting cwnd increases during
   application-limited periods.  The experiment used a real ssh
   connection through a modem link emulated using Dummynet [Dummynet].
   The link speed is 30Kb/s and the link has five packet buffers
   available.  Today most modem banks have more buffering available than
   this, but the more buffer-limited situation sometimes occurs with
   older modems.  In the first half of the transfer, the user is typing
   away over the connection.  About half way through the time, the user
   lists a moderately large file, which causes a large burst of traffic
   to be transmitted.

   For the unmodified TCP, every returning ACK during the first part of
   the transfer results in an increase in cwnd.  As a result, the large
   burst of data arriving from the application to the transport layer is
   sent as many back-to-back packets, most of which get lost and
   subsequently retransmitted.

   For the modified TCP with Congestion Window Validation, the
   congestion window is not increased when the window is not full, and
   has been decreased during application-limited periods closer to what
   the user actually used.  The burst of traffic is now constrained by
   the congestion window, resulting in a better-behaved flow with

minimal loss.  The end result is that the transfer happens
approximately 30% faster than the transfer without CWV, due to
avoiding retransmission timeouts.

The second experiment uses a real ssh connection over a real dialup
ppp connection, where the modem bank has much more buffering.  For
the unmodified TCP, the initial burst from the large file does not
cause loss, but does cause the RTT to increase to approximately 5
seconds, where the connection becomes bounded by the receiver's
window.

For the modified TCP with Congestion Window Validation, the flow is
much better behaved, and produces no large burst of traffic.  In this
case the linear increase for cwnd results in a slow increase in the
RTT as the buffer slowly fills.

For the second experiment, both the modified and the unmodified TCP
finish delivering the data at precisely the same time.  This is
because the link has been fully utilized in both cases due to the
modem buffer being larger than the receiver window.  Clearly a modem
buffer of this size is undesirable due to its effect on the RTT of
competing flows, but it is necessary with current TCP implementations
that produce bursts similar to those shown in the top graph.

6. Conclusions

This document has presented several TCP algorithms for Congestion
Window Validation, to be employed after an idle period or a period in
which the sender was application-limited, and before an increase of
the congestion window.  The goal of these algorithms is for TCP's
congestion window to reflect recent knowledge of the TCP connection
about the state of the network path, while at the same time keeping
some memory (i.e., in ssthresh) about the earlier state of the path.
We believe that these modifications will be of benefit to both the
network and to the TCP flows themselves, by preventing unnecessary
packet drops due to the TCP sender's failure to update its
information (or lack of information) about current network
conditions.  Future work will document and investigate the benefit
provided by these algorithms, using both simulations and experiments.
Additional future work will describe a more complex version of the
CWV algorithm for TCP implementations where the sender does not have
an accurate estimate of the TCP roundtrip time.

7. References

    [FF96]       Fall, K., and Floyd, S., Simulation-based Comparisons of
                 Tahoe, Reno, and SACK TCP, Computer Communication Review,
                 V. 26 N. 3, July 1996, pp. 5-21.  URL
                 "http://www.aciri.org/floyd/papers.html".

    [HPF99]      Mark Handley, Jitendra Padhye, Sally Floyd, TCP Congestion
                 Window Validation, UMass CMPSCI Technical Report 99-77,
                 September 1999.  URL "ftp://www-
                 net.cs.umass.edu/pub/Handley99-tcpq-tr-99-77.ps.gz".

    [HTH98]      Amy Hughes, Joe Touch, John Heidemann, "Issues in TCP
                 Slow-Start Restart After Idle", Work in Progress.

    [J88]        Jacobson, V., Congestion Avoidance and Control, Originally
                 from Proceedings of SIGCOMM '88 (Palo Alto, CA, Aug.
                 1988), and revised in 1992.  URL "http://www-
                 nrg.ee.lbl.gov/nrg-papers.html".

    [JKBFL96]    Raj Jain, Shiv Kalyanaraman, Rohit Goyal, Sonia Fahmy, and
                 Fang Lu, Comments on "Use-it or Lose-it", ATM Forum
                 Document Number:  ATM Forum/96-0178, URL
                 "http://www.netlab.ohio-
                 state.edu/~jain/atmf/af_rl5b2.htm".

    [JKGFL95]    R. Jain, S. Kalyanaraman, R. Goyal, S. Fahmy, and F. Lu, A
                 Fix for Source End System Rule 5, AF-TM 95-1660, December
                 1995, URL "http://www.netlab.ohio-
                 state.edu/~jain/atmf/af_rl52.htm".

    [MSML99]     Matt Mathis, Jeff Semke, Jamshid Mahdavi, and Kevin Lahey,
                 The Rate-Halving Algorithm for TCP Congestion Control,
                 June 1999.  URL
                 "http://www.psc.edu/networking/ftp/papers/draft-
                 ratehalving.txt".

    [NS]         NS, the UCB/LBNL/VINT Network Simulator.  URL
                 "http://www-mash.cs.berkeley.edu/ns/".

    [RFC2581]    Allman, M., Paxson, V. and W. Stevens, TCP Congestion
                 Control, RFC 2581, April 1999.

    [VH97]       Vikram Visweswaraiah and John Heidemann. Improving Restart
                 of Idle TCP Connections, Technical Report 97-661,
                 University of Southern California, November, 1997.

   [Dummynet] Luigi Rizzo, "Dummynet and Forward Error Correction",
             Freenix 98, June 1998, New Orleans.  URL
             "http://info.iet.unipi.it/~luigi/ip_dummynet/".

8. Security Considerations

   General security considerations concerning TCP congestion control are
   discussed in RFC 2581.  This document describes a algorithm for one
   aspect of those congestion control procedures, and so the
   considerations described in RFC 2581 apply to this algorithm also.
   There are no known additional security concerns for this specific
   algorithm.

9. Authors' Addresses

   Mark Handley
   AT&T Center for Internet Research at ICSI (ACIRI)

   Phone: +1 510 666 2946
   EMail: mjh@aciri.org
   URL: http://www.aciri.org/mjh/


   Jitendra Padhye
   AT&T Center for Internet Research at ICSI (ACIRI)

   Phone: +1 510 666 2887
   EMail: padhye@aciri.org
   URL: http://www-net.cs.umass.edu/~jitu/


   Sally Floyd
   AT&T Center for Internet Research at ICSI (ACIRI)

   Phone: +1 510 666 2989
   EMail: floyd@aciri.org
   URL:  http://www.aciri.org/floyd/

10. Full Copyright Statement

Acknowledgement