       TCP Congestion Control with Appropriate Byte Counting (ABC)

Status of this Memo

Copyright Notice

Abstract

   This document proposes a small modification to the way TCP increases
   its congestion window. Rather than the traditional method of
   increasing the congestion window by a constant amount for each
   arriving acknowledgment, the document suggests basing the increase on
   the number of previously unacknowledged bytes each ACK covers. This
   change improves the performance of TCP, as well as closes a security
   hole TCP receivers can use to induce the sender into increasing the
   sending rate too rapidly.

Terminology

   Much of the language in this document is taken from [RFC2581].

   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 [RFC2119].

1   Introduction

   This document proposes a modification to the algorithm for increasing
   TCP's congestion window (cwnd) that improves both performance and
   security. Rather than increasing a TCP's congestion window based on
   the number of acknowledgments (ACKs) that arrive at the data sender
   (per the current specification [RFC2581]), the congestion window is
   increased based on the number of bytes acknowledged by the arriving
   ACKs. The algorithm improves performance by mitigating the impact of
   delayed ACKs on the growth of cwnd. At the same time, the algorithm
   provides cwnd growth in direct relation to the probed capacity of a

network path, therefore providing a more measured response to ACKs
that cover only small amounts of data (less than a full segment size)
than ACK counting. This more appropriate cwnd growth can improve
both performance and can prevent inappropriate cwnd growth in
response to a misbehaving receiver. On the other hand, in some cases
the modified cwnd growth algorithm causes larger bursts of segments
to be sent into the network. In some cases this can lead to a non-
negligible increase in the drop rate and reduced performance (see
section 4 for a larger discussion of the issues).

This document is organized as follows. Section 2 outlines the
modified algorithm for increasing TCP's congestion window. Section 3
discusses the advantages of using the modified algorithm. Section 4
discusses the disadvantages of the approach outlined in this
document. Section 5 outlines some of the fairness issues that must
be considered for the modified algorithm. Section 6 discusses
security considerations.

Statement of Intent

   This specification contains an algorithm improving the performance
   of TCP which is understood to be effective and safe, but which has
   not been widely deployed. One goal of publication as an
   Experimental RFC is to be prudent, and encourage use and
   deployment prior to publication in the standards track. It is the
   intent of the Transport Area to re-submit this specification as an
   IETF Proposed Standard in the future, after more experience has
   been gained.

2   A Modified Algorithm for Increasing the Congestion Window

   As originally outlined in [Jac88] and specified in [RFC2581], TCP
   uses two algorithms for increasing the congestion window. During
   steady-state, TCP uses the Congestion Avoidance algorithm to linearly
   increase the value of cwnd. At the beginning of a transfer, after a
   retransmission timeout or after a long idle period (in some
   implementations), TCP uses the Slow Start algorithm to increase cwnd
   exponentially. According to RFC 2581, slow start bases the cwnd
   increase on the number of incoming acknowledgments. During
   congestion avoidance RFC 2581 allows more latitude in increasing
   cwnd, but traditionally implementations have based the increase on
   the number of arriving ACKs. In the following two subsections, we
   detail modifications to these algorithms to increase cwnd based on
   the number of bytes being acknowledged by each arriving ACK, rather
   than by the number of ACKs that arrive. We call these changes
   "Appropriate Byte Counting" (ABC) [All99].

2.1 Congestion Avoidance

   RFC 2581 specifies that cwnd should be increased by 1 segment per
   round-trip time (RTT) during the congestion avoidance phase of a
   transfer. Traditionally, TCPs have approximated this increase by
   increasing cwnd by 1/cwnd for each arriving ACK. This algorithm
   opens cwnd by roughly 1 segment per RTT if the receiver ACKs each
   incoming segment and no ACK loss occurs. However, if the receiver
   implements delayed ACKs [Bra89], the receiver returns roughly half as
   many ACKs, which causes the sender to open cwnd more conservatively
   (by approximately 1 segment every second RTT). The approach that
   this document suggests is to store the number of bytes that have been
   ACKed in a "bytes_acked" variable in the TCP control block. When
   bytes_acked becomes greater than or equal to the value of the
   congestion window, bytes_acked is reduced by the value of cwnd.
   Next, cwnd is incremented by a full-sized segment (SMSS). The
   algorithm suggested above is specifically allowed by RFC 2581 during
   congestion avoidance because it opens the window by at most 1 segment
   per RTT.

2.2 Slow Start

   RFC 2581 states that the sender increments the congestion window by
   at most, 1*SMSS bytes for each arriving acknowledgment during slow
   start. This document proposes that a TCP sender SHOULD increase cwnd
   by the number of previously unacknowledged bytes ACKed by each
   incoming acknowledgment, provided the increase is not more than L
   bytes. Choosing the limit on the increase, L, is discussed in the
   next subsection. When the number of previously unacknowledged bytes
   ACKed is less than or equal to 1*SMSS bytes, or L is less than or
   equal to 1*SMSS bytes, this proposal is no more aggressive (and
   possibly less aggressive) than allowed by RFC 2581. However,
   increasing cwnd by more than 1*SMSS bytes in response to a single ACK
   is more aggressive than allowed by RFC 2581. The more aggressive
   version of the slow start algorithm still falls within the spirit of
   the principles outlined in [Jac88] (i.e., of no more than doubling
   the cwnd per RTT), and this document proposes ABC for experimentation
   in shared networks, provided an appropriate limit is applied (see
   next section).

2.3 Choosing the Limit

   The limit, L, chosen for the cwnd increase during slow start,
   controls the aggressiveness of the algorithm. Choosing L=1*SMSS
   bytes provides behavior that is no more aggressive than allowed by
   RFC 2581. However, ABC with L=1*SMSS bytes is more conservative in a

number of key ways (as discussed in the next section) and therefore,
this document suggests that even though with L=1*SMSS bytes TCP
stacks will see little performance change, ABC SHOULD be used.

A very large L could potentially lead to large line-rate bursts of
traffic in the face of a large amount of ACK loss or in the case when
the receiver sends "stretch ACKs" (ACKs for more than the two full-
sized segments allowed by the delayed ACK algorithm) [Pax97].

This document specifies that TCP implementations MAY use L=2*SMSS
bytes and MUST NOT use L > 2*SMSS bytes. This choice balances
between being conservative (L=1*SMSS bytes) and being potentially
very aggressive. In addition, L=2*SMSS bytes exactly balances the
negative impact of the delayed ACK algorithm (as discussed in more
detail in section 3.2). Note that when L=2*SMSS bytes cwnd growth is
roughly the same as the case when the standard algorithms are used in
conjunction with a receiver that transmits an ACK for each incoming
segment [All98] (assuming no or small amounts of ACK loss in both
cases).

The exception to the above suggestion is during a slow start phase
that follows a retransmission timeout (RTO). In this situation, a
TCP MUST use L=1*SMSS as specified in RFC 2581 since ACKs for large
amounts of previously unacknowledged data are common during this
phase of a transfer. These ACKs do not necessarily indicate how much
data has left the network in the last RTT, and therefore ABC cannot
accurately determine how much to increase cwnd. As an example, say
segment N is dropped by the network, and segments N+1 and N+2 arrive
successfully at the receiver. The sender will receive only two
duplicate ACKs and therefore must rely on the retransmission timer
(RTO) to detect the loss. When the RTO expires, segment N is
retransmitted. The ACK sent in response to the retransmission will
be for segment N+2. However, this ACK does not indicate that three
segments have left the network in the last RTT, but rather only a
single segment left the network. Therefore, the appropriate cwnd
increment is at most 1*SMSS bytes.

2.4 RTO Implications

[Jac88] shows that increases in cwnd of more than a factor of two in
succeeding RTTs can cause spurious retransmissions on slow links
where the bandwidth dominates the RTT, assuming the RTO estimator
given in [Jac88] and [RFC2988]. ABC stays within this limit of no
more than doubling cwnd in successive RTTs by capping the increase
(no matter what L is employed) by the number of previously
unacknowledged bytes covered by each incoming ACK.

3   Advantages

   This section outlines several advantages of using the ABC algorithm
   to increase cwnd, rather than the standard ACK counting algorithm
   given in [RFC2581].

3.1 More Appropriate Congestion Window Increase

   The ABC algorithm outlined in section 2 increases TCP's cwnd in
   proportion to the amount of data actually sent into the network. ACK
   counting, on the other hand, increments cwnd by a constant upon the
   arrival of each ACK. For instance, consider an interactive telnet
   connection (e.g., ssh or telnet) in which ACKs generally cover only a
   few bytes of data, but cwnd is increased by 1*SMSS bytes for each ACK
   received. When a large amount of data needs to be transmitted (e.g.,
   displaying a large file) the data is sent in one large burst because
   the cwnd grows by 1*SMSS bytes per ACK rather than based on the
   actual amount of capacity used. Such a line-rate burst of data can
   potentially cause a large amount of segment loss.

   Congestion Window Validation (CWV) [RFC2861] addresses the above
   problem as well. CWV limits the amount of unused cwnd a TCP
   connection can accumulate. ABC can be used in conjunction with CWV
   to obtain an accurate measure of the network path.

3.2 Mitigate the Impact of Delayed ACKs and Lost ACKs

   Delayed ACKs [RFC1122,RFC2581] allow a TCP receiver to refrain from
   sending an ACK for each incoming segment. However, a receiver SHOULD
   send an ACK for every second full-sized segment that arrives.
   Furthermore, a receiver MUST NOT withhold an ACK for more than 500
   ms. By reducing the number of ACKs sent to the data originator the
   receiver is slowing the growth of the congestion window under an ACK
   counting system. Using ABC with L=2*SMSS bytes can roughly negate
   the negative impact imposed by delayed ACKs by allowing cwnd to be
   increased for ACKs that are withheld by the receiver. This allows
   the congestion window to grow in a manner similar to the case when
   the receiver ACKs each incoming segment, but without adding extra
   traffic to the network. Simulation studies have shown increased
   throughput when a TCP sender uses ABC when compared to the standard
   ACK counting algorithm [All99], especially for short transfers that
   never leave the initial slow start period.

   Note that delayed ACKs should not be an issue during slow start-based
   loss recovery, as RFC 2581 recommends that receivers should not delay
   ACKs that cover out-of-order segments. Therefore, as discussed
   above, ABC with L > 1*SMSS bytes is inappropriate for such slow start
   based loss recovery and MUST NOT be used.

   Note: In the case when an entire window of data is lost, a TCP
   receiver will likely generate delayed ACKs and an L > 1*SMSS bytes
   would be safe. However, detecting this scenario is difficult.
   Therefore to keep ABC conservative, this document mandates that L
   MUST NOT be > 1*SMSS bytes in any slow start-based loss recovery.

   ACK loss can also retard the growth of a congestion window that
   increases based on the number of ACKs that arrive. When counting
   ACKs, dropped ACKs represent forever-missed opportunities to increase
   cwnd. Using ABC with L > 1*SMSS bytes allows the sender to mitigate
   the effect of lost ACKs.

3.3 Prevents Attacks from Misbehaving Receivers

   [SCWA99] outlines several methods for a receiver to induce a TCP
   sender into violating congestion control and transmitting data at a
   potentially inappropriate rate. One of the outlined attacks is "ACK
   Division". This scheme involves the receiver sending multiple ACKs
   for each incoming data segment, each ACKing only a small portion of
   the original TCP data segment. Since TCP senders have traditionally
   used ACK counting to increase cwnd, ACK division causes
   inappropriately rapid cwnd growth and, in turn, a potentially
   inappropriate sending rate. A TCP sender that uses ABC can prevent
   this attack from being used to undermine standard congestion control
   because the cwnd increase is based on the number of bytes ACKed,
   rather than the number of ACKs received.

   To prevent misbehaving receivers from inducing inappropriate sender
   behavior, this document suggests TCP implementations use ABC, even if
   L=1*SMSS bytes (i.e., not allowing ABC to provide more aggressive
   cwnd growth than allowed by RFC 2581).

4   Disadvantages

   The main disadvantages of using ABC with L=2*SMSS bytes are an
   increase in the burstiness of TCP and a small increase in the overall
   loss rate. [All98] discusses the two ways that ABC increases the
   burstiness of the TCP sender. First, the "micro burstiness" of the
   connection is increased. In other words, the number of segments sent
   in response to each incoming ACK is increased by at most 1 segment
   when using ABC with L=2*SMSS bytes in conjunction with a receiver
   that is sending delayed ACKs. During slow start this translates into
   an increase from sending 2 back-to-back segments to sending 3 back-
   to-back packets in response to an ACK for a single packet. Or, an
   increase from 3 packets to 4 packets when receiving a delayed ACK for
   two outstanding packets. Note that ACK loss can cause larger bursts.
   However, ABC only increases the burst size by at most 1*SMSS bytes
   per ACK received when compared to the standard behavior. This slight

increase in the burstiness should only cause problems for devices
that have very small buffers. In addition, ABC increases the "macro
burstiness" of the TCP sender in response to delayed ACKs in slow
start. Rather than increasing cwnd by roughly 1.5 times per RTT, ABC
roughly doubles the congestion window every RTT. However, doubling
cwnd every RTT fits within the spirit of slow start, as originally
outlined [Jac88].

With the increased burstiness comes a modest increase in the loss
rate for a TCP connection employing ABC (see the next section for a
short discussion on the fairness of ABC to non-ABC flows). The
additional loss can be directly attributable to the increased
aggressiveness of ABC. During slow start cwnd is increased more
rapidly. Therefore when loss occurs cwnd is larger and more drops
are likely. Similarly, a congestion avoidance cycle takes roughly
half, as long when using ABC and delayed ACKs when compared to an ACK
counting implementation. In other words, a TCP sender reaches the
capacity of the network path, drops a packet and reduces the
congestion window by half roughly twice as often when using ABC.
However, as discussed above, in spite of the additional loss an ABC
TCP sender generally obtains better overall performance than a non-
ABC TCP [All99].

Due to the increase in the packet drop rate we suggest ABC be
implemented in conjunction with selective acknowledgments [RFC2018].

5   Fairness Considerations

[All99] presents several simple simulations conducted to measure the
impact of ABC on competing traffic (both ABC and non-ABC). The
experiments show that while ABC increases the drop rate for the
connection using ABC, competing traffic is not greatly effected. The
experiments show that standard TCP and ABC both obtain roughly the
same throughput, regardless of the variant of the competing traffic.
The simulations also reaffirm that ABC outperforms non-ABC TCP in an
environment with varying types of TCP connections. On the other
hand, the simulations presented in [All99] are not necessarily
realistic. Therefore we are encouraging more experimentation in the
Internet.

6   Security Considerations

As discussed in section 3.3, ABC protects a TCP sender from a
misbehaving receiver that induces the sender into transmitting at an
inappropriate rate with an "ACK division" attack. This, in turn,
protects the network from an overly aggressive sender.

7 Conclusions

   This document RECOMMENDS that all TCP stacks be modified to use ABC
   with L=1*SMSS bytes. This change does not increase the
   aggressiveness of TCP. Furthermore, simulations of ABC with L=2*SMSS
   bytes show a promising performance improvement that we encourage
   researchers to experiment with in the Internet.

Acknowledgments

   This document has benefited from discussions with and encouragement
   from Sally Floyd. Van Jacobson and Reiner Ludwig provided valuable
   input on the implications of byte counting on the RTO. Reiner Ludwig
   and Kostas Pentikousis provided valuable feedback on a draft of this
   document.

Normative References

   [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts --
             Communication Layers", STD 3, RFC 1122, October 1989.

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

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

Informative References

   [All98]   Mark Allman. On the Generation and Use of TCP
             Acknowledgments. ACM Computer Communication Review, 29(3),
             July 1998.

   [All99]   Mark Allman. TCP Byte Counting Refinements. ACM Computer
             Communication Review, 29(3), July 1999.

   [Jac88]   Van Jacobson. Congestion Avoidance and Control. ACM
             SIGCOMM 1988.

   [Pax97]   Vern Paxson. Automated Packet Trace Analysis of TCP
             Implementations. ACM SIGCOMM, September 1997.

   [RFC2018] Mathis, M., Mahdavi, J., Floyd, S. and A. Romanow, "TCP
             Selective Acknowledgment Options", RFC 2018, October 1996.

   [RFC2861] Handley, M., Padhye, J. and S. Floyd, "TCP Congestion
             Window Validation", RFC 2861, June 2000.

   [SCWA99] Stefan Savage, Neal Cardwell, David Wetherall, Tom
             Anderson. TCP Congestion Control with a Misbehaving
             Receiver. ACM Computer Communication Review, 29(5),
             October 1999.

Author's Address

   Mark Allman
   BBN Technologies/NASA Glenn Research Center
   Lewis Field
   21000 Brookpark Rd. MS 54-5
   Cleveland, OH 44135

   Fax: 216-433-8705
   Phone: 216-433-6586
   EMail: mallman@bbn.com
   http://roland.grc.nasa.gov/~mallman

Full Copyright Statement

Acknowledgement