                    TCP Slow Start, Congestion Avoidance,
                 Fast Retransmit, and Fast Recovery Algorithms


Status of this Memo

    This document is an Internet Draft.  Internet Drafts are working
    documents of the Internet Engineering Task Force (IETF), its Areas,
    and its Working Groups.  Note that other groups may also distribute
    working documents as Internet Drafts.

    Internet Drafts are draft documents valid for a maximum of six
    months.  Internet Drafts may be updated, replaced, or obsoleted by
    other documents at any time.  It is not appropriate to use Internet
    Drafts as reference material or to cite them other than as a
    "working draft" or "work in progress."

    To learn the current status of any Internet-Draft, please check the
    "1id-abstracts.txt" listing contained in the internet-drafts Shadow
    Directories on:

            ftp.is.co.za (Africa)
            nic.nordu.net (Europe)
            ds.internic.net (US East Coast)
            ftp.isi.edu (US West Coast)
            munnari.oz.au (Pacific Rim)

Abstract

    Modern implementations of TCP contain four intertwined algorithms
    that have never been fully documented as Internet standards:  slow
    start, congestion avoidance, fast retransmit, and fast recovery.
    [2] and [3] provide some details on these algorithms, [4] provides
    examples of the algorithms in action, and [5] provides the source
    code for the 4.4BSD implementation.  RFC 1122 requires that a TCP
    must implement slow start and congestion avoidance (Section 4.2.2.15
    of [1]), citing [2] as the reference, but fast retransmit and fast
    recovery were implemented after RFC 1122.  The purpose of this
    Internet Draft is to document these four algorithms for the
    Internet.

**1**.  **Slow Start**

    Old TCPs would start a connection with the sender injecting multiple
    segments into the network, up to the window size advertised by the
    receiver.  While this is OK when the two hosts are on the same LAN,
    if there are routers and slower links between the sender and the
    receiver, problems can arise.  Some intermediate router must queue
    the packets, and it's possible for that router to run out of space.
    [2] shows how this naive approach can reduce the throughput of a TCP
    connection drastically.

    The algorithm to avoid this is called slow start.  It operates by
    observing that the rate at which new packets should be injected into
    the network is the rate at which the acknowledgments are returned by
    the other end.

    Slow start adds another window to the sender's TCP:  the congestion
    window, called "cwnd".  When a new connection is established with a
    host on another network, the congestion window is initialized to one
    segment (i.e., the segment size announced by the other end, or the
    default, typically 536 or 512).  Each time an ACK is received, the
    congestion window is increased by one segment.  The sender can
    transmit up to the minimum of the congestion window and the
    advertised window.  The congestion window is flow control imposed by
    the sender, while the advertised window is flow control imposed by
    the receiver.  The former is based on the sender's assessment of
    perceived network congestion; the latter is related to the amount of
    available buffer space at the receiver for this connection.

    The sender starts by transmitting one segment and waiting for its
    ACK.  When that ACK is received, the congestion window is
    incremented from one to two, and two segments can be sent.  When
    each of those two segments is acknowledged, the congestion window is
    increased to four.  This provides an exponential increase, although
    it is not exactly exponential because the receiver may delay its
    ACKs, typically sending one ACK for every two segments that it
    receives.

    At some point the capacity of the internet can be reached, and an
    intermediate router will start discarding packets.  This tells the

sender that its congestion window has gotten too large.

Early implementations performed slow start only if the other end was on a different network.  Current implementations always perform slow start.

**[2].  Congestion Avoidance**

Congestion can occur when data arrives on a big pipe (a fast LAN) and gets sent out a smaller pipe (a slower WAN).  Congestion can also occur when multiple input streams arrive at a router whose output capacity is less than the sum of the inputs.  Congestion avoidance is a way to deal with lost packets.  It is described in [2].

The assumption of the algorithm is that packet loss caused by damage is very small (much less than 1%), therefore the loss of a packet signals congestion somewhere in the network between the source and destination.  There are two indications of packet loss:  a timeout occurring and the receipt of duplicate ACKs.

Congestion avoidance and slow start are independent algorithms with different objectives.  But when congestion occurs TCP must slow down its transmission rate of packets into the network, and then invoke slow start to get things going again.  In practice they are implemented together.

Congestion avoidance and slow start require that two variables be maintained for each connection: a congestion window, cwnd, and a slow start threshold size, ssthresh.  The combined algorithm operates as follows:

1.  Initialization for a given connection sets cwnd to one segment and ssthresh to 65535 bytes.

2.  The TCP output routine never sends more than the minimum of cwnd and the receiver's advertised window.

3.  When congestion occurs (indicated by a timeout or the reception of duplicate ACKs), one-half of the current window size (the minimum of cwnd and the receiver's advertised window, but at least two segments) is saved in ssthresh.  Additionally, if the congestion is indicated by a timeout, cwnd is set to one segment (i.e., slow start).

4.  When new data is acknowledged by the other end, increase cwnd, but the way it increases depends on whether TCP is performing slow start or congestion avoidance.

If cwnd is less than or equal to ssthresh, TCP is in slow start;
otherwise TCP is performing congestion avoidance.  Slow start
continues until TCP is halfway to where it was when congestion
occurred (since it recorded half of the window size that caused
the problem in step 2), and then congestion avoidance takes
over.

Slow start has cwnd begin at one segment, and be incremented by
one segment every time an ACK is received.  As mentioned
earlier, this opens the window exponentially:  send one segment,
then two, then four, and so on.  Congestion avoidance dictates
that cwnd be incremented by 1/cwnd each time an ACK is received.
This is an additive increase, compared to slow start's
exponential increase.  The increase in cwnd should be at most
one segment each round-trip time (regardless how many ACKs are
received in that RTT), whereas slow start increments cwnd by the
number of ACKs received in a round-trip time.

Many implementations incorrectly add a small fraction of the segment
size (typically the segment size divided by 8) during congestion
avoidance.  This is wrong and should not be emulated in future
releases.

**3.  Fast Retransmit**

Modifications to the congestion avoidance algorithm were proposed in
1990 [3].  Before describing the change, realize that TCP may
generate an immediate acknowledgment (a duplicate ACK) when an out-
of-order segment is received (Section 4.2.2.21 of [1], with a note
that one reason for doing so was for the experimental fast-
retransmit algorithm).  This duplicate ACK should not be delayed.
The purpose of this duplicate ACK is to let the other end know that
a segment was received out of order, and to tell it what sequence
number is expected.

Since TCP does not know whether a duplicate ACK is caused by a lost
segment or just a reordering of segments, it waits for a small
number of duplicate ACKs to be received.  It is assumed that if
there is just a reordering of the segments, there will be only one
or two duplicate ACKs before the reordered segment is processed,
which will then generate a new ACK.  If three or more duplicate ACKs
are received in a row, it is a strong indication that a segment has
been lost.  TCP then performs a retransmission of what appears to be
the missing segment, without waiting for a retransmission timer to
expire.  This is the fast retransmit algorithm.

**4.  Fast Recovery**

After fast retransmit sends what appears to be the missing segment, congestion avoidance, but not slow start is performed.  This is the fast recovery algorithm.  It is an improvement that allows high throughput under moderate congestion, especially for large windows.

The reason for not performing slow start in this case is that the receipt of the duplicate ACKs tells TCP more than just a packet has been lost.  Since the receiver can only generate the duplicate ACK when another segment is received, that segment has left the network and is in the receiver's buffer.  That is, there is still data flowing between the two ends, and TCP does not want to reduce the flow abruptly by going into slow start.

The fast retransmit and fast recovery algorithms are usually implemented together as follows.

1.  When the third duplicate ACK in a row is received, set ssthresh to one-half the current congestion window, cwnd, but no less than two segments.  Retransmit the missing segment.  Set cwnd to ssthresh plus 3 times the segment size.  This inflates the congestion window by the number of segments that have left the network and which the other end has cached (3).

2.  Each time another duplicate ACK arrives, increment cwnd by the segment size.  This inflates the congestion window for the additional segment that has left the network.  Transmit a packet, if allowed by the new value of cwnd.

3.  When the next ACK arrives that acknowledges new data, set cwnd to ssthresh (the value set in step 1).  This ACK should be the acknowledgment of the retransmission from step 1, one round-trip time after the retransmission.  Additionally, this ACK should acknowledge all the intermediate segments sent between the lost packet and the receipt of the first duplicate ACK.  This step is congestion avoidance, since TCP is down to one-half the rate it was at when the packet was lost.

The fast retransmit algorithm first appeared in the 4.3BSD Tahoe release, but it was incorrectly followed by slow start.  The fast recovery algorithm appeared in the 4.3BSD Reno release.

**[5]. Security Considerations**

Security considerations are not discussed in this memo.

**6**. **References**

[1]   B. Braden, ed., "Requirements for Internet Hosts --
      Communication Layers," RFC 1122, Oct. 1989.

[2]   V. Jacobson, "Congestion Avoidance and Control," Computer
      Communication Review, vol. 18, no. 4, pp. 314-329, Aug. 1988.
      ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z.

[3]   V. Jacobson, "Modified TCP Congestion Avoidance Algorithm,"
      end2end-interest mailing list, April 30, 1990.
      ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail.

[4]   W. R. Stevens, "TCP/IP Illustrated, Volume 1: The Protocols",
      Addison-Wesley, 1994.

[5]   G. R. Wright, W. R. Stevens, "TCP/IP Illustrated, Volume 2:
      The Implementation", Addison-Wesley, 1995.

Author's  Address:

    W. Richard Stevens
    1202 E. Paseo del Zorro
    Tucson, AZ  85718

    Phone: 520-297-9416

    EMail: rstevens@noao.edu


    Expires: August 26, 1996