TCP Variations
Tahoe, Reno, New-Reno

Internet Networking
recitation #10
Evolution of TCP

1975
Three-way handshake
Ray Tomlinson
In IEEE Trans Comm

1974
TCP described by
Vint Cerf, Bob Kahn
In IEEE Trans Comm

1975
TCP & IP
RFC 793 & 791

1981
TCP & IP
RFC 793 & 791

1983
BSD Unix 4.2
supports TCP/IP

1984
Nagel’s algorithm
to reduce overhead
of small packets;
predicts congestion
collapse

1981
TCP & IP
RFC 793 & 791

1985
Congestion
collapse
1st observed

1987
Karn’s algorithm
to better estimate
round-trip time

1986
Congestion
collapse
1st observed

1990
4.3BSD Reno
fast recovery
delayed ACK’s

1987
Karn’s algorithm
to better estimate
round-trip time

1987
Karn’s algorithm
to better estimate
round-trip time

1988
Van Jacobson’s
algorithms
slow start,
congestion
avoidance, fast
retransmit (all
implemented in
4.3BSD Tahoe)
SIGCOMM 88

1987
Karn’s algorithm
to better estimate
round-trip time
Evolution of TCP

1993
TCP Vegas (not implemented)
real congestion avoidance (Brakmo et al)

1994
ECN Explicit Congestion Notification (Floyd)

1996
NewReno modified fast recovery
SACK TCP Selective Ack (Floyd et al)

1996
Improving TCP startup (Hoe)

What do we use today?
Wait for next tutorial...

4.3.3 TCP loss recovery: the sliding window concept

- TCP uses the sliding window approach in order to guarantee:
  - reliable delivery of data
  - congestion control
  - flow control
- The sender’s window is divided into 4 parts:
  - Bytes 1-2 were sent and acknowledged
  - they can, therefore, be removed from the sender’s buffer.
  - Bytes 3-7 were sent but not yet acknowledged
  - Bytes 8-9 were not sent, but can be sent
  - Bytes 10-.... cannot be sent before more acknowledgments arrive.
- “Window size” is the number of bytes/segments that can be pending.

window size is 7
Reminder - TCP Algorithms

• Four intertwined algorithms used commonly in TCP implementations:
  • **Slow Start** - Every *ack* increases the sender’s window (*cwnd*) size by 1.
  • **Congestion Avoidance** - Reducing sender’s window size by half at experience of loss, and increase the sender’s window at the rate of about *one packet per RTT* (*NOTE: not per ack*).
  • **Fast Retransmit** - Don’t wait for retransmit timer to go off, retransmit packet if 3 *duplicate acks* received.
  • **Fast Recovery** - Since duplicate *ack* came through, one packet *has left the wire*. Perform *congestion avoidance*, don’t jump down to *slow start*. 
TCP Tahoe

• implements the slow start, congestion avoidance, and fast retransmit algorithms.
TCP Tahoe’s Fast Retransmit

1. Sender receives 3 dupACKS.
2. Sender infers that the segment is lost.
3. Sender re-sends the segment immediately!
4. Sender returns to slow-start.

**Fast Retransmit of Segment 3**
TCP Tahoe Trace (1 Lost Seg)

![Graph showing TCP Tahoe trace with lost segment, fast retransmit, begin slow-start, and begin congestion avoidance.]
Could Tahoe be Improved?

• Receipt of dupACKs tells the sender that the receiver is still getting new segments, i.e. there is still data flowing between sender and receiver.

• Why does sender go back to slow start after fast retransmit?
TCP Reno

• 2nd Improvement was TCP Reno (1990)
  • Nagle’s algorithm
  • Improved RTO calculation and back-off
  • AIMD congestion avoidance with slow-start
  • Fast retransmit & fast recovery
Fast Retransmit + Fast Recovery

• After receiving 3 dupACKS:
  1. Retransmit the lost segment.
  2. Set ssthresh = flight size/2.
  3. Set cwnd = ssthresh, and ndupacks = 3.
     (In Reno: send_win = min ( rwnd, cwnd + ndupacks ))

• If dupACK arrives:
  • ++ ndupacks
  • Transmit new segment, if allowed.

• If new ACK arrives:
  • ndupacks = 0
  • Exit fast recovery.

• If RTO expires:
  • ndupacks = 0
  • Perform slow-start - ( ssthresh = flight size/2, cwnd = 1 )
**TCP Reno**

**Initial state**
cwnd=7
ndupacks = 0
→ Slow start

**Fast Retransmit**
cwnd = 8/2 = 4
ssthresh = 8/2 = 4
ndupacks=3
→ Fast Recovery

**Exit Fast Recovery**
ndupacks = 0
→ Congestion Avoidance

**Fast Retransmit**
send_win=8

**Exit Fast Recovery**
send_win=9
send_win=10

**Congestion Avoidance**
send_win=8
send_win=7
send_win=8

**Initial state**
cwnd=7
ndupacks = 0
→ Slow start

**Fast Retransmit**
cwnd = 8/2 = 4
ssthresh = 8/2 = 4
ndupacks=3
→ Fast Recovery

**Exit Fast Recovery**
ndupacks = 0
→ Congestion Avoidance

**Fast Retransmit**
send_win=8

**Exit Fast Recovery**
send_win=9
send_win=10

**Congestion Avoidance**
send_win=8
send_win=7
send_win=8
TCP Reno Trace (1 Lost seg)
Multiple Losses

Initial state
\(cwnd=7\)
\(n_{\text{dupacks}}=0\) → Slow start

Fast Retransmit
\(ssthresh = 8/2 = 4\)
\(cwnd = ssthresh= 4\)
\(n_{\text{dupacks}}=3\) → Fast Recovery

Exit Fast Recovery
\(cwnd = ssthresh = 4\) → Congestion Avoidance

No. of unAcked Segs > cwnd → No New Segments

Fast Retransmit
\(cwnd = ssthresh = 4\) → Fast Recovery

Congestion Avoidance
\(cwnd = ssthresh = 4\) → Congestion Avoidance
What if There are Multiple Losses in a Window?

• With two losses in a window, Reno will occasionally timeout.

• With three losses in a window, Reno will usually timeout.

• With four losses in a window, Reno is guaranteed to timeout!

• With three or more losses in a window, Tahoe typically outperforms Reno!
Limitations of TCP Reno algorithm

• If cwnd size is too small (smaller than 4 packets) then it’s not possible to get 3 duplicate acks and run the algorithm.

• The algorithm can’t manage with lost of packets during the Fast Recovery stage.
  • Not a loss of the retransmitted packet.
  • There is no recursive run of Fast Retransmit.
TCP Reno Trace (2 lost segs)
TCP New Reno

- 3rd Improvement was TCP NewReno (1995)
  - Nagle’s algorithm
  - Improved RTO calculation and back-off
  - AIMD congestion avoidance with slow-start
  - Fast retransmit & modified fast recovery
Partial ACK

- **Partial ACKs**: An ACK that acknowledges some but not all the segments that were outstanding at the start of fast recovery. NewReno interprets this as an indication of multiple loss.
- How do we know it’s a partial ACK? We record to “Recovery” variable the highest sequence number transmitted.
Retransmission Process

• When a partial ACK is received:
  • retransmit the first unacknowledged segment.
  • set ndupacks=0
  • send a new segment if permitted by the new value of send_win.

• When an acknowledge of all of the data up to and including "recover“ arrives – Exit fast recovery.
New Reno

Initial state
$\text{cwnd} = 5$
$\rightarrow$ Slow start

Fast Retransmit
$ssthresh = 6/2 = 3$
$cwnd = ssthresh = 3$
$ndupacks = 3$
$send\_win = 6$

Recover = 6
$\rightarrow$ Fast Recovery
$\text{Recover} \geq \text{Ack}$
Partial Ack
$ndupacks = 0$
$send\_win = 3 + 0 = 3$
$\text{Recover < Ack}$
Exit Fast Recovery
$cwnd = ssthresh = 3$
$\rightarrow$ Congestion Avoidance

send\_win = 6

$\text{Seg}(0) \rightarrow \text{Ack}(1)$
$\text{Seg}(1) \rightarrow \text{Ack}(1)\times$
$\text{Seg}(2) \rightarrow \text{Ack}(1)$
$\text{Seg}(3) \rightarrow \text{Ack}(1)\times$
$\text{Seg}(4) \rightarrow \text{Ack}(1)$

send\_win = 7

$\text{Seg}(5) \rightarrow \text{Ack}(1)$
$\text{Seg}(6) \rightarrow \text{Ack}(1)$

$cwnd = 4$

$\text{Seg}(1) \rightarrow \text{Ack}(3)$
$\text{Seg}(7) \rightarrow \text{Ack}(3)$
$\text{Seg}(3) \rightarrow \text{Ack}(9)$
$\text{Seg}(8) \rightarrow \text{Ack}(9)$
$\text{Seg}(9) \rightarrow \text{Ack}(10)$

send\_win = 3

$\text{Seg}(1) \rightarrow \text{Ack}(3)$
$\text{Seg}(7) \rightarrow \text{Ack}(3)$
$\text{Seg}(3) \rightarrow \text{Ack}(9)$
$\text{Seg}(8) \rightarrow \text{Ack}(9)$
$\text{Seg}(9) \rightarrow \text{Ack}(10)$
TCP New Reno Trace
(2 lost segs)
Tahoe, Reno & NewReno Trace (2 lost segs)
Is There a Better Way?

• The only way Tahoe, Reno and NewReno can detect congestion is by creating congestion!
  • They carefully probe for congestion by slowly increasing their sending rate.
  • When they find (create), congestion, they cut sending rate at least in half!

• This slow advance and rapid retreat approach results in a saw-toothed sending rate, and highly erratic throughput.

• What if TCP could detect congestion without causing congestion?

• TCP VEGAS…
A TCP/IP packet goes into a bar. It says, "I'd like a beer". The barman asks, "A beer?" The packet responds, "Yes, a beer."