CIS 457 - Data Communications

Nathan Bowman

Images taken from Kurose and Ross book

TCP Congestion Control -- Policy

Final question for TCP congestion control is policy: how much do we lower or raise rate based on ACKs and losses?

Best way depends on network and traffic characteristics

Tricky balancing act, because too high of rate causes congestion, but too low of rate wastes bandwidth

Protocol must be distributed -- individual TCP senders control only their own behavior and know only what they directly observe from network

To further complicate matters, conditions in network may change over time

TCP constantly probing to attempt to determine highest rate it can send at right now

Any time ACK received, sender increases transmission rate

Keeps increasing until loss event, at which point it decreases

Cyclical process: after decrease, sender goes right back to increasing rate

By always "pushing the limit", sender can keep track of changes in network

Algorithm for determining rate split into three parts or phases

- slow start
- congestion avoidance
- fast recovery

Slow start and congestion avoidance are mandatory parts of protocol, but fast recovery is optional (recommended)

Slow start

Sender always begins in slow start

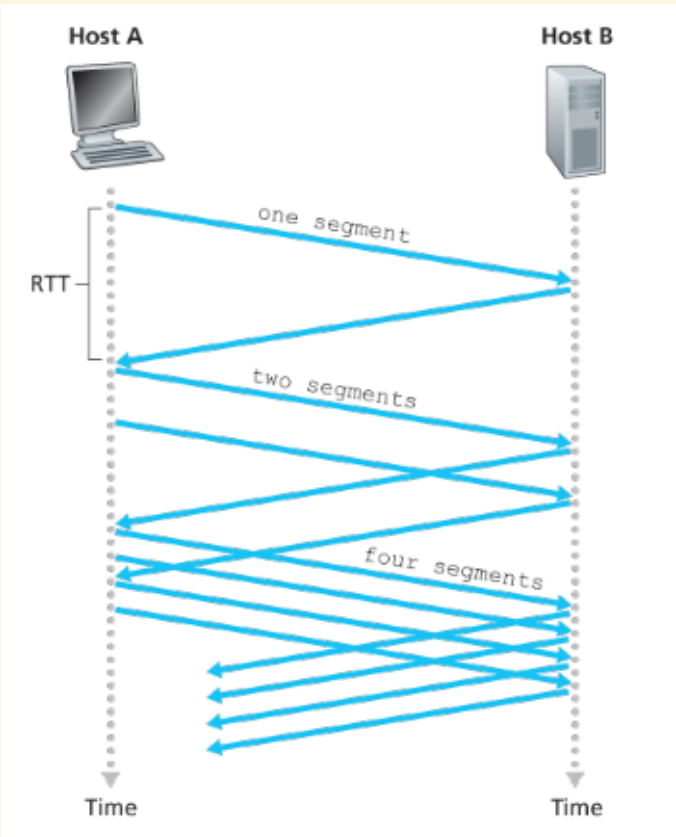
In slow start, cwnd initially set to just 1 MSS

Transmission rate therefore MSS/RTT

This lives up to the "slow" part -- sender wants to increase rate quickly

Sender increases window size by 1 MSS per new ACK

Does not sound fast, but consider that number of ACKs increases each time window size increases


Host A
Host B
one segment
RTT
two segments
four segments
Time
Time

- Send 1 MSS; wait RTT; ACK
- Send 2 MSS; wait RTT; 2 ACKs
- Send 4 MSS; wait RTT; 4 ACKs
- Send 8 MSS; wait RTT; 8 ACKs

Window size doubles each RTT -- this is exponential growth

Exponential growth can only continue so long

Three ways to leave slow start:

- packet loss (timeout)
- hit threshold based on most recent packet loss
- triple duplicate ACKs

When packet loss occurs, sender sets **slow start threshold** (`ssthresh`) to `cwnd/2`

Sender then resets `cwnd` to 1 and starts over, now with new stopping point `ssthresh`

If no loss events occur before `ssthresh` reached, sender still ceases to double rate after `ssthresh`

Doing so would cause rate to jump to where last packet loss occurred

Instead, sender leaves slow start stage and enters congestion avoidance

Third and final way to trigger end of slow start is receipt of triple duplicate ACKs

When this occurs, sender performs fast retransmit (as discussed in previous lecture) and enters fast recovery state

Congestion avoidance

In congestion avoidance stage, sender increases `cwnd` by 1 MSS per RTT (rather than per ACK)

How can sender know RTT without clocking it?

Still a matter of counting ACKs

At any time, sender has roughly cwnd/MSS packets in network

Should receive roughly cwnd/MSS ACKs per RTT

So, increase by MSS * (MSS/cwnd) bytes per new ACK

For example, if MSS is 1460 and cwnd is 14600, each ACK will increase cwnd by

1460 * (1460 / 14600)

= 1/10 * 1460

cwnd was 10 packets, and increases by 1/10 packet per ACK, which is roughly 1 packet per RTT

Congestion avoidance is slower, safer linear increase

No upper threshold -- continues increasing until loss event

When timeout occurs, do exactly same thing as slow start would:

Set `ssthresh` to `cwnd/2`

Set `cwnd` to 1

Restart in slow start

In case of triple duplicate ACK, also does same thing as slow start would, though we have not specified that yet

This scenario is different from timeout because it tells sender several packets have still made it through, so perhaps congestion not so bad

Sender performs fast retransmit

Sets `ssthresh` to `cwnd/2`

Rather than start over, set `cwnd` to `cwnd/2 + 3` (because of 3 ACKs)

Go to fast recovery state

Fast recovery

This is temporary state to deal with loss event in network that may not be especially congested

Increase cwnd by 1 MSS for each duplicate ACK of packet that put us into this state

If timeout occurs, behaves exactly the same as other two stages:

- set `ssthresh` appropriately,
- set `cwnd` to 1 MSS, and
- enter slow start

This state was based on lost packet, and ends when ACK for that packet arrives

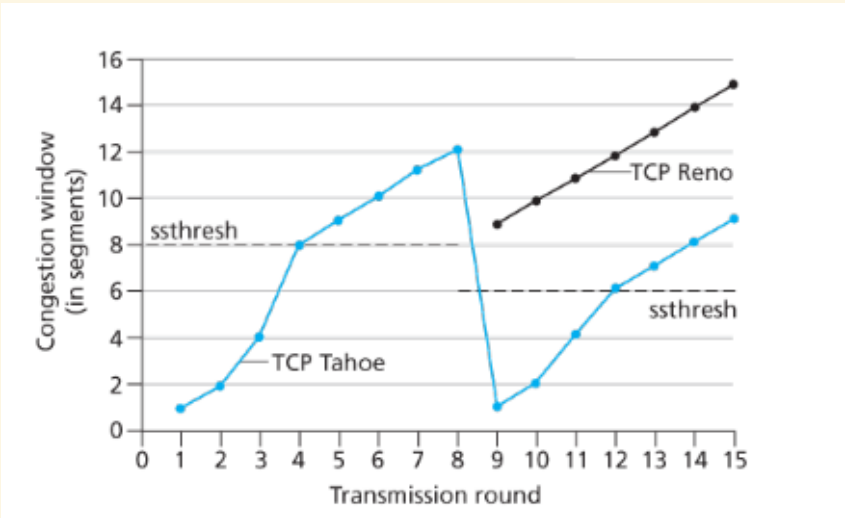
At that point, sender will:

- set `cwnd` to `ssthresh`
- enter congestion avoidance

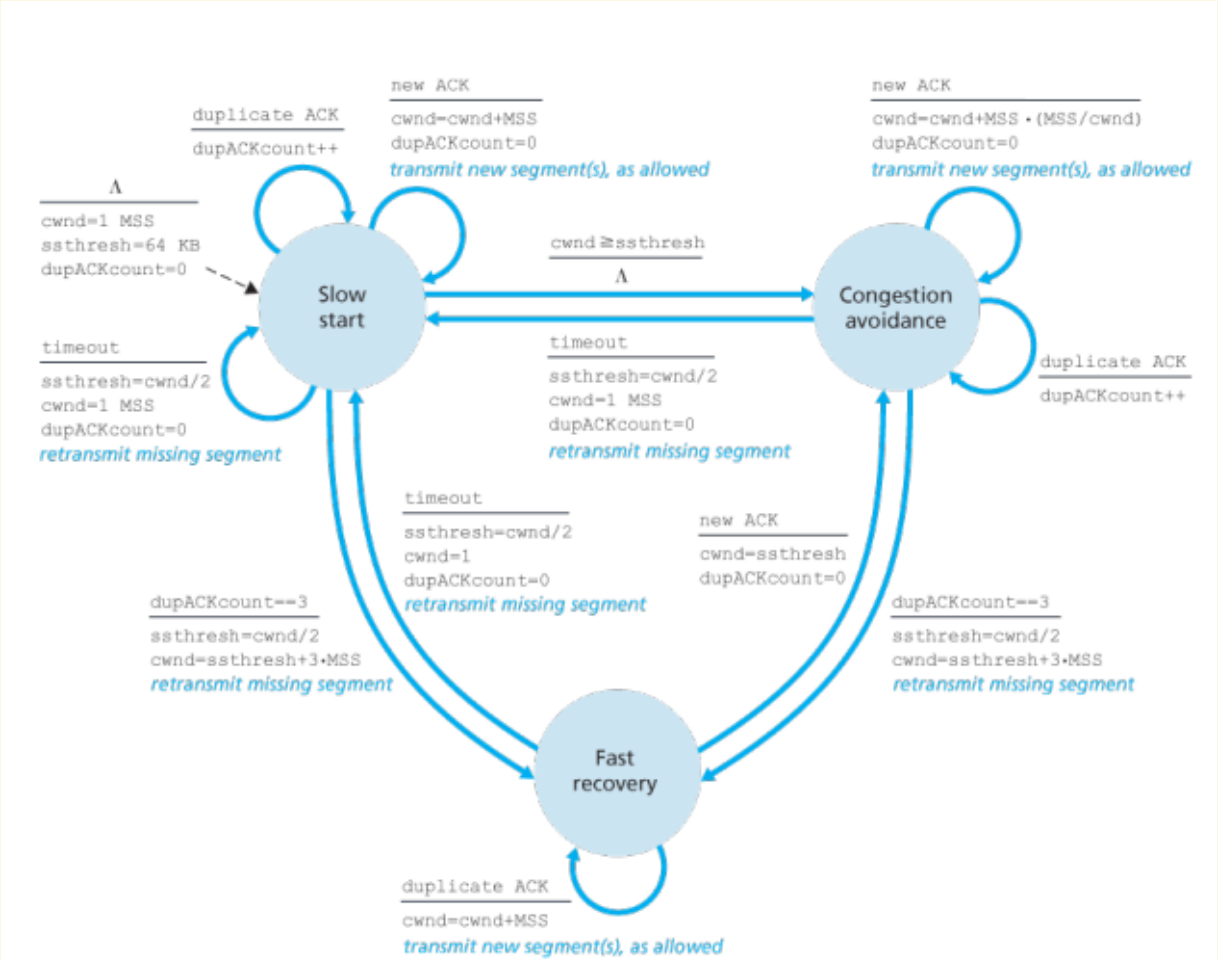
As mentioned, fast recovery is optional

Older TCP standard, TCP Tahoe, did not have fast recovery

Newer TCP Reno does


Congestion window (in segments)
16
14
12
10
8
6
4
2
0
ssthresh
TCP Tahoe
TCP Reno
ssthresh
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Transmission round

Like many protocols, congestion control in TCP can be described succinctly by FSM


duplicate ACK
dupACKcount++
new ACK
cwnd=cwnd+MSS
dupACKcount=0
transmit new segment(s), as allowed
new ACK
cwnd=cwnd+MSS • (MSS/cwnd)
dupACKcount=0
transmit new segment(s), as allowed
Λ
cwnd=1 MSS
ssthresh=64 KB
dupACKcount=0
Slow start
cwnd≥ssthresh
Λ
Congestion avoidance
timeout
ssthresh=cwnd/2
cwnd=1 MSS
dupACKcount=0
retransmit missing segment
timeout
ssthresh=cwnd/2
cwnd=1 MSS
dupACKcount=0
retransmit missing segment
duplicate ACK
dupACKcount++
timeout
ssthresh=cwnd/2
cwnd=1
dupACKcount=0
retransmit missing segment
new ACK
cwnd=ssthresh
dupACKcount=0
dupACKcount==3
ssthresh=cwnd/2
cwnd=ssthresh+3•MSS
retransmit missing segment
dupACKcount==3
ssthresh=cwnd/2
cwnd=ssthresh+3•MSS
retransmit missing segment
Fast recovery
duplicate ACK
cwnd=cwnd+MSS
transmit new segment(s), as allowed