CMSC417 Spring 2016 Lecture #16 4/4/2016

Agenda:

⇒ p3 due Friday April 8th at 11:59:59 pm
- p4 will still be assigned
- use extra time on p4 or studying for midterm 2

⇒ midterm 2 a week from today 4/11

⇒ TCP congestion control
- AIMD
- slow start
- fast recovery

CMSC417 Spring 2016 Lecture #16 4/4/2016

TCP congestion control at equilibrium RFC 2001

⇒ same as sliding window, but
- use min of cwnd and advertised window as the sender window
- also count retransmissions

⇒ in other words, don't put a new packet into the network unless
- window increased
- a packet left the network (I got a new ack)

⇒ also this only works if cwnd is a good window to avoid collapse

adjusting cwnd

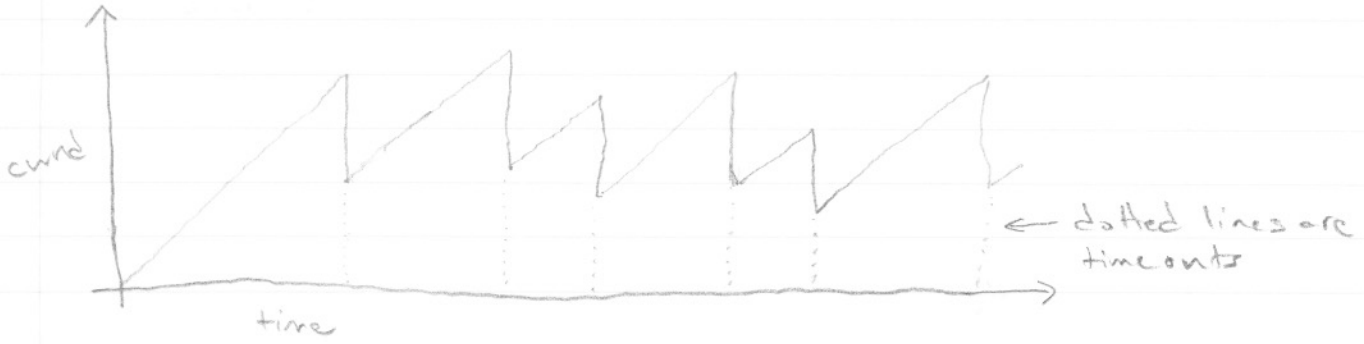
⇒ same as before w/ sliding window
- too big: crush the network
- too small: not filling the pipe

⇒ guessing wrong could be really bad
- risk of cwnd being too big is larger than too small
- start w/ a small cwnd (1-2 MSS)
  - in practice it's 4-10 MSS on real computers I've seen today
- increase cwnd slowly
- decrease cwnd quickly

CMSCH117 Spring 2016 Lecture #16 4/4/2016

AIMD (additive increase, multiplicative decrease)

⇒ in steady state TCP connections
- cwnd++ per RTT w/o loss (actually += MSS/cwnd per acked MSS)
- cwnd = cwnd/2 on a timeout (loss)



⇒ "sawtooth" from linear increase and exponential decrease

problem?

⇒ where do you start?
- MSS ~= 1500 bytes
- RTT ~= 50 ms
- starting at 1 MSS cwnd

} ⇒ bw-dely product = 625 KB to 1.25 MB ≈ 400-800 MSS

- 800 RTTs is too long to set up to speed

! 20-80 seconds to get cwnd to be large enough if starting from 1 MSS

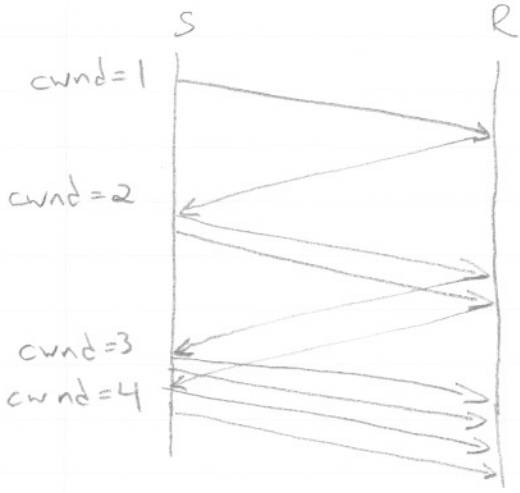
⇒ picking too big a starting value could cause congestion collapse while failing to the "right" value

⇒ solution: exponentially grow cwnd from ~1 MSS until there's a loss — called "Slow Start"

CMSC417 Spring 2016 Lecture #16 4/4/2016

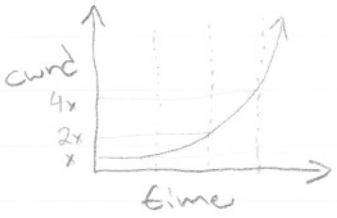
this is important it's what breaks you out of slow start

Slow start



⇒ every RTT, w/o loss, you get cwnd acks and thus cwnd doubles

⇒ exponential growth of cwnd



what about loss?

⇒ when you see loss, you need to go slower
- how much slower?
- a lot slower b/c you could be here

remember this?

⇒ original TCP w/ consestion control (TCP Tahoe)
- start from 1 MSS again
- use threshold = cwnd/2 to exit next time
  - intuitively, last RTT @ cwnd/2 biggest good one

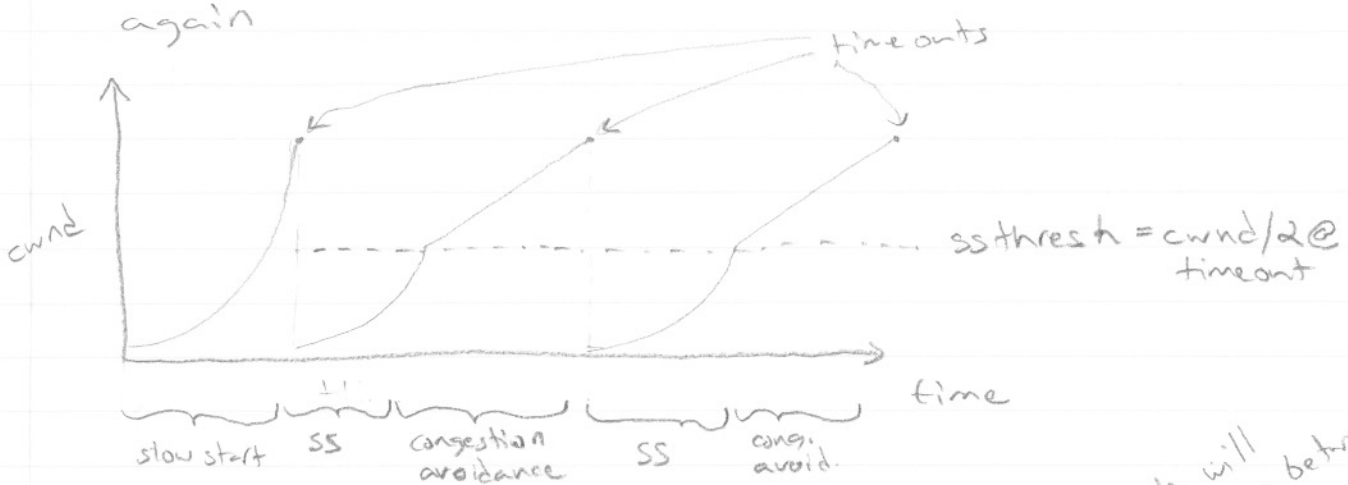
⇒ exiting slow start b/c of threshold
- after that add 1/cwnd to cwnd on every ack
- cwnd++ per RTT vs. cwnd*=2 per RTT
- slowly explore sending faster

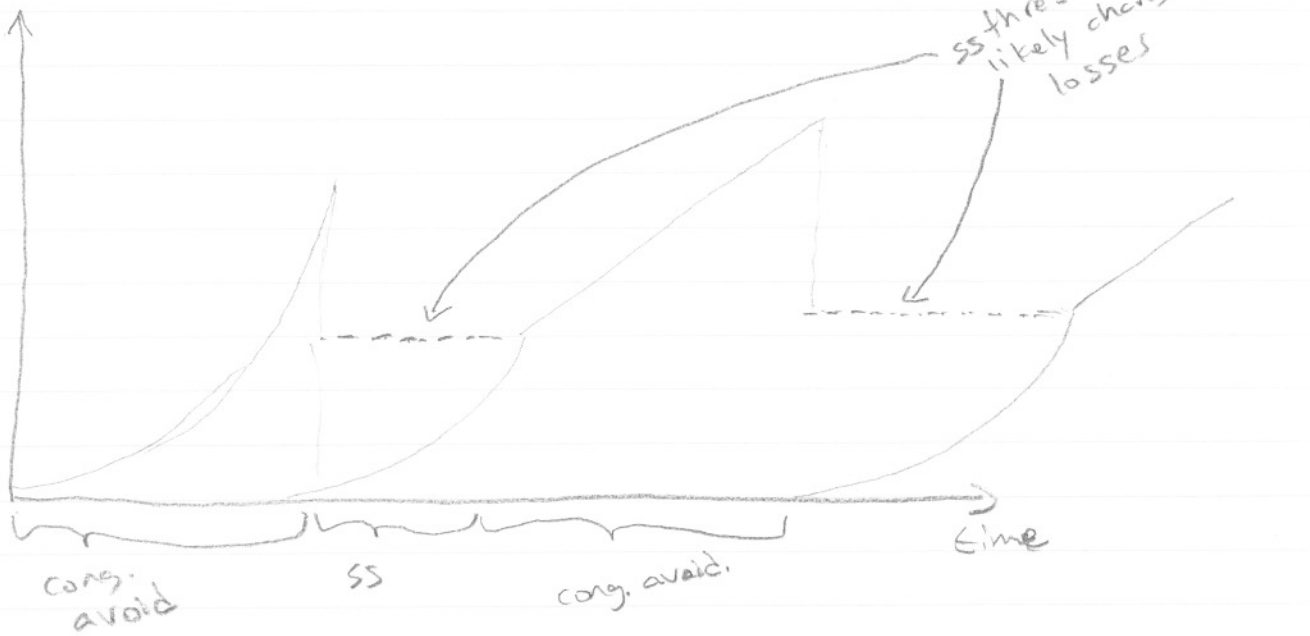
CMSC417 Spring 2016 Lecture#16 4/4/2016

slow start cont'd

⇒ also exit slow start on hitting ssthresh (or slow start threshold)
- ssthresh first set to 65,535 (largest advertisable window)
- ssthresh is set to cwnd/2 on loss

⇒ on a timeout (loss) start from 1 MSS again





CMSC4117 Spring 2016 Lecture#16 4/4/2016

Fast Recovery (TCP Reno)

⇒ remember fast retransmit?

- on 4 duplicate acks, treat it as a timeout and retransmit the segment
- should we treat it as a loss for TCP's cwnd?
  - conservative: yes
  - reality: maybe, we're still getting packets through, hopefully we're not off by much

⇒ **solution:** do some of the backoff, but not all of it

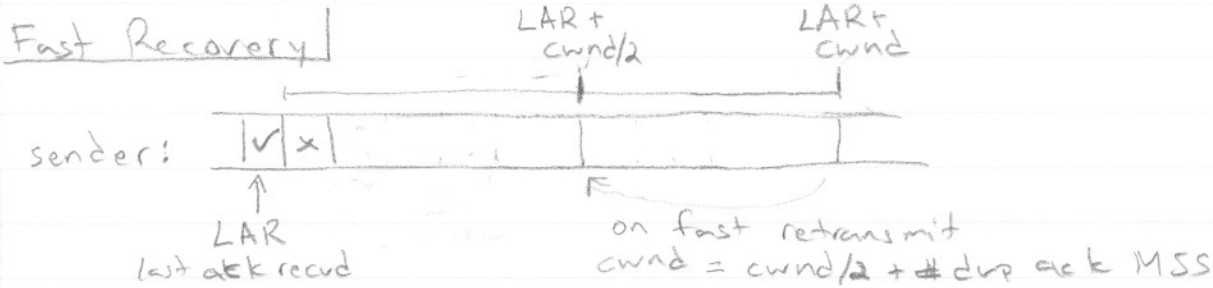
- set ssthresh to cwnd/2 as normal
- skip slow-start
- set cwnd to ssthresh + 3 * MSS for the packets leaving the network indicated by the duplicate acks (instead of 1 MSS)
- cwnd += 1 MSS for each subsequent duplicate ack

the growing by one MSS per ack is to allow cwnd to balloon to keep the stalled LAR sending packets in the face of the stalled LAR

- on the first new ack, cwnd = ssthresh (contract window to where it would have been after a loss)

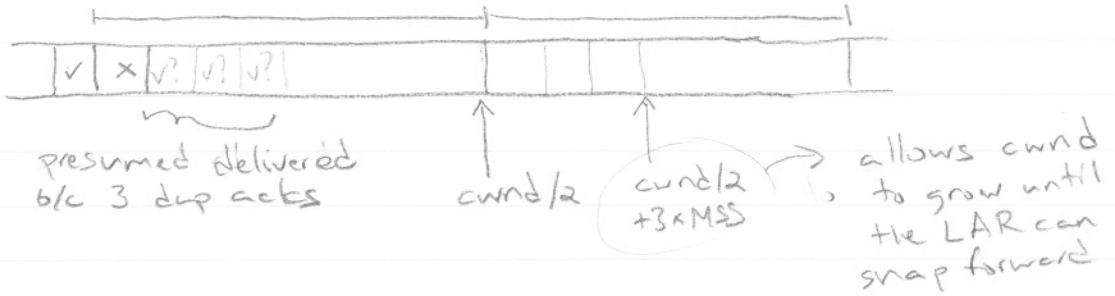
CMSC417 Spring 2016 Lecture #16 4/4/2016

Fast Recovery

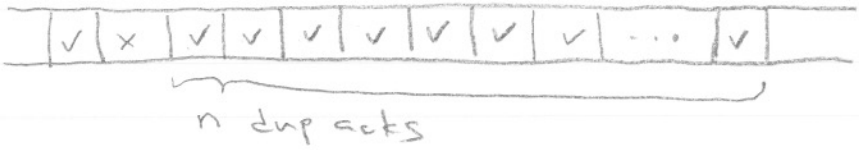


⇒ LAR is stuck for an RTT

⇒ we can still use acks to guess how many segments leave the network



eventually:



⇒ receiver gets retransmitted segment

- cwnd = old_cwnd/2 + n × mss
- LAR jumps by n+1
- cwnd = old_cwnd/2
  - snaps back the slack

CMSC417 Spring 2016 Lecture #16 4/4/2016

when do you actually use slowstart?

⇒ timeouts

⇒ connection beginnings

Other details of TCP

⇒ delayed ACK: can ack less than every packet, typically every other (sometimes less frequent than that)

⇒ cubic: used in Linux 2.6.19 – 3.1 (and more?)

instead of this: 

do this: 

~x³

idea: spend more time in the "good" range
ramp up fast, explore fast at the end

⇒ BIC

⇒ misbehaving receivers

⇒ TCP vegas (DCTCP)

⇒ where loss comes from