TCP: Congestion Avoidance and Adversaries

I. Congestion Avoidance

Key idea: conserve packets to avoid congestion (flow in == flow out)

1) slow start
   - add second window -- congestion window. Use the minimum of the two windows.
   - CW starts out a 1 packet, adds one with each (successful) ack
   - each ack thus causes two new packets to go out: one to replace the outstanding packet and one because the window size just got larger
   - after a loss, reset to 1 (cheaters reset to 3)

2) round-trip time estimator
   - need to estimate variance, since it goes up during congestion (a lot)
   - underestimate RTT => retransmit packets that are in transit (not dropped) -- this violates conservation
   - estimating (vs fixed) variance also helps high-latency links that have low variance (satellite)
   - need exponential backoff (just like ethernet) if you are retransmitting packets

3) assume packet loss means congestion
   - this only makes sense after you fix the RTT estimates
   - doesn’t work well for wireless (where losses have other causes)
   - alternative is to use signal in the ack to indicate congestion, but requires changing other parts of the system (see ELN -- explicit loss notification)
   - multiplicative decrease to quickly return to stability
   - additive increase to test for more bandwidth nicely
   - need to do additive increase anytime congestion window is less that receiver’s window, which is most of the time

II. Misbehaving Receivers

Key ideas:
   - TCP assumes cooperation: no longer valid
   - Users have motive and opportunity to cheat
   - Don’t attack implementations, instead attack the spec!!
TCP

- all valid implementations should be vulnerable
  o Introduce changes to prevent *motive* (benefit from cheating)

IPSEC and VPNs don’t help
  o possibly can help detect cheaters after the fact!

Selective acks (SACK):
  o extension to support ACKs for things received out of order
  o Without SACK, sender may retransmit packets that arrived OK (but out of order)
  o Does not really affect attacks

Attack 1: ack division
  o each ack increases window by a lot, even if you only ack a little (1 byte)
  o solution: open window incrementally, or just wait for whole segments to be acked before changing window
  o window completely open in two RTs (for typical web access)

Attack 2: Duplicate ack spoofing
  o repeat acks, causing fast retransmit and increased window
  o solution: nonce to prove that a packet was received. Only increment window for each proven received packet
  o (A nonce is a random key that you have to return to prove that you received it. You can guess it, so you have to actually get it....)

Attack 3: Optmistic acks
  o ack before you get the data
  o increases window
  o but... may lose data, and may ack something the sender hasn’t sent yet
  o for lost data, you can selectively retrieve it later for some higher level protocols (http in particular)
  o not as strong an attack as the first two
  o Use a cumulative nonce

Implementation: TCP Daytona  (joke related to other variants: TCP Reno, TCP Tahoe, ...)
  o very little code
TCP

- works on most OSs
- Linux solves partial problems (the 1st attack is nicely solved)
- NT solves one by not following the spec! (it’s broken)

Cumulative nonce is a nice general solution:

- lightweight
- idempotent
- cumulative

- See Abadi/Needham principles