CSE/EE 461

Retransmission and Timers

Last Time …

- More on the Transport Layer
- Focus
  - How do we manage connections?
- Topics
  - Three-Way Handshake
  - Close and TIME_WAIT

Application
Presentation
Session
Transport
Network
Data Link
Physical

This Lecture

- Focus
  - How do we decide <u>when to retransmit</u>?
- Topics
  - RTT estimation
  - Karn/Partridge algorithm
  - Jacobson/Karels algorithm

Application
Presentation
Session
Transport
Network
Data Link
Physical

Deciding When to Retransmit

- How do you know when a packet has been lost?

  again:

  ```
  Send(p);
  Wait(t);
  if (!p.acked)
      goto again;
  ```

- How long should the timer **t** be?
  - Too big: inefficient (large delays, poor use of bandwidth)
  - Too small: may retransmit unnecessarily (causing extra traffic)
  - A good retransmission timer is important for good performance
- Right timer is based on the round trip time (RTT)
  - Which varies greatly in the wide area (path length and queuing)

Congestion Collapse

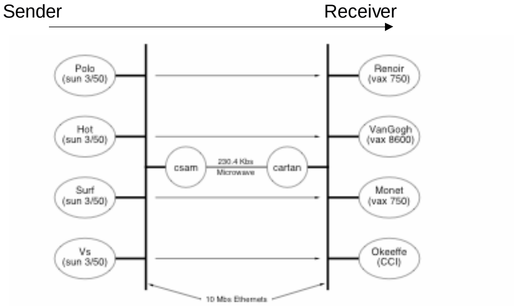
- In the limit, early retransmissions lead to congestion collapse
  - Sending more packets into the network when it is overloaded exacerbates the problem of congestion
  - Network stays busy but very little useful work is being done
- This happened in real life ~1987
  - Led to Van Jacobson's TCP algorithms, which form the basis of congestion control in the Internet today [See "Congestion Avoidance and Control", SIGCOMM'88]
  - Observed 1000x bandwidth reduction between two hosts separated by 400 yards.
  - Led to researchers asking two questions:
    - Was TCP/IP misbehaving?
    - Could TCP/IP be "trained" to work better under 'absymal network conditions'

5

A Scenario

Sender

Receiver



Test setup to examine the interaction of multiple, simultaneous TCP conversations sharing a bottleneck link. 1 MByte transfers (2048 512-data-byte packets) were initiated 3 seconds apart from four machines at LBL to four machines at UCB, one conversation per machine pair (the dotted lines above show the pairing). All traffic went via a 230.4 Kbps link connecting IP router **csam** at LBL to IP router **cartan** at UCB. The microwave link queue can hold up to 50 packets. Each connection was given a window of 16 KB (32 512-byte packets). Thus any two connections could overflow the available buffering and the four connections exceeded the queue capacity by 160%.

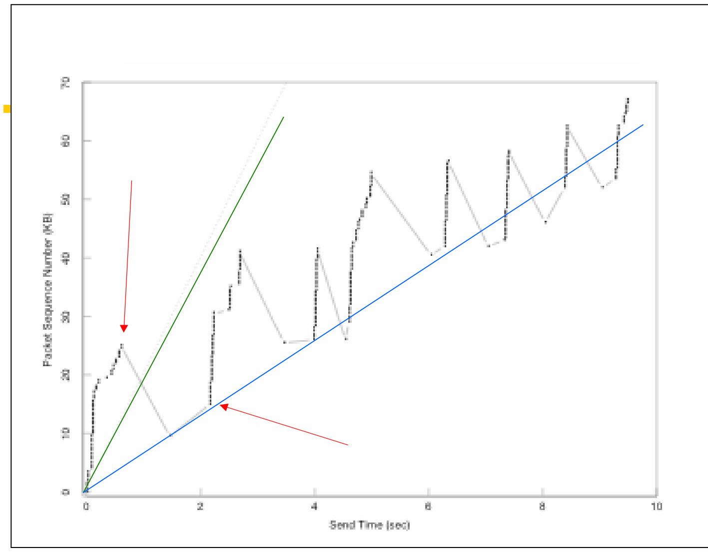
Receiver window size is 16KB.

Bottleneck router buffer size is 15 KB.

Data bandwidth is about 20KB/s

6

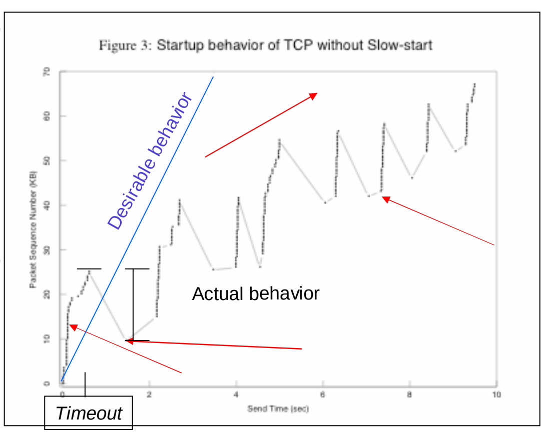
Sequence #



TIME (SEC)

Effects of Early Retransmissions

Packet Sequence Number (KB)



Send Time (sec)

Slope is bandwidth.

Steep smooth upward slope means good bandwidth.

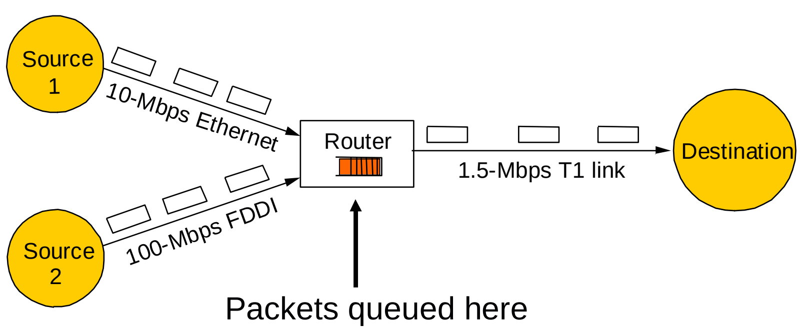
Downward slope means retransmissions (*bad).*

If only…

- We knew RTT and Current Router Queue Size,
  - Then we would send MIN(Router Queue Size, Effective Window Size)
  - And not resent a packet until it had been sent RTT ago.
- But we don't know these things, so we have to figure them out.
- And they may change dynamically due to other data sources

9

Congestion from Multiple Sources



- Buffers at routers used to absorb bursts when input rate > output
- Loss (drops) occur when sending rate is persistently > drain rate

Interpacket Spacing

Interpacket spacing mirrors that of slowest link

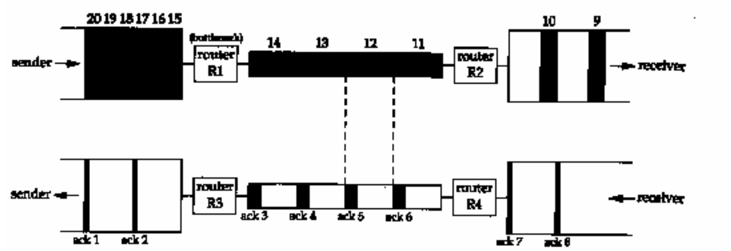


Figure 20.13 Congestion caused by a bigger pipe feeding a smaller pipe.

Inter-ACK spacing mirrors that of slowest downstream link

11

1988 Observations on Congestion Collapse

- Implementation, not the protocol, leads to collapse
- “Obvious” ways of doing things lead to non-obvious and undesirable results
  - “send eff-wind-size # packets, wait rtt, try again”
- Remedial algorithms achieve network stability by forcing the transport connection to obey a ‘packet conservation’ principle.
  - For a connection in ‘equilibrium, that is, running stably with a full window of data in transit, the packet flow is “conservative”: a new packet is not put into the network until an old packet leaves.

12

Resulting TCP/IP Improvements

- *Slow-start*
- *Round-trip time variance estimation*
- *Exponential retransmit timer backoff*
- *More aggressive receiver ack policy*
- *Dynamic window sizing on congestion*
- Clamped retransmit backoff (Karn)
- Fast Retransmit

Packet Conservation Principle

Congestion control means: "Finding places that violate the conservation of packets principle and then fixing them."

In order to conserve packets

1. The connection must reach equilibrium.
   – Hurry up and stabilize!
   – When things get wobbly, put on the brakes and reconsider
2. A sender must not inject a new packet before an old packet has exited.
   – A packet "exits" when the receiver picks it up.
   – Or it gets lost
     • Damaged in transit
     • Dropped at a congestion point
     • Fewer than 1% of packets get damaged
   – Ack or packet timeout signals that a packet has "exited."
   – Acks are easy to detect.
   – Good timeouts are harder…. All about estimating RTT.
3. Equilibrium is lost because of resource contention along the way.
   – New competing stream appears

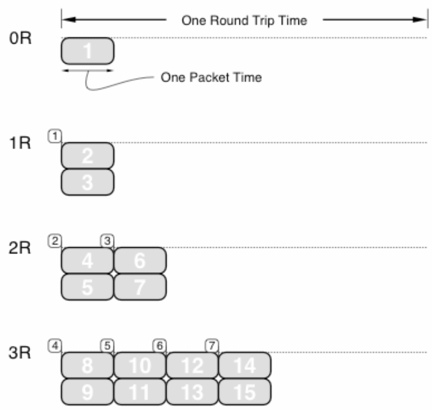
1. The connection must reach equilibrium.

1. Getting to Equilibrium -- Slow Start

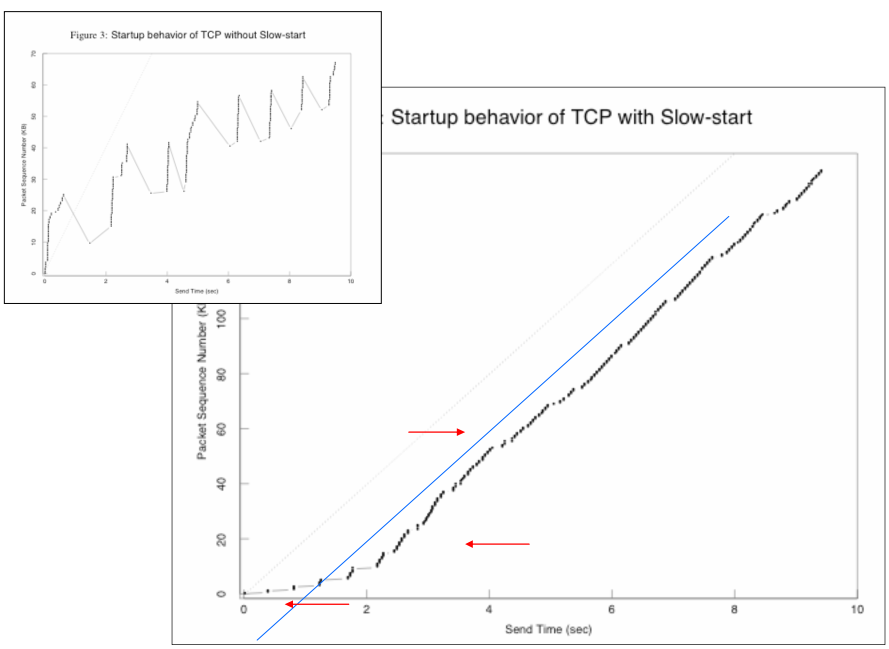
- Goal
  - Quickly determine the appropriate window size
- Strategy
  - Introduce *congestion_window (cwnd)*
  - When starting off, set cwnd to 1
  - For each ack received, add 1 to cwnd
  - When sending, send the minimum of receiver's advertised window and cwd
- Guaranteed to not transmit at more than twice the max bw, and for no more than RTT.
  - (bw delay product)

Cwnd doubles every RTT;
Opening the window of size
W takes time (RTT)$\log_2$W.

Figure 2: The Chronology of a Slow-start



The horizontal direction is time. The continuous time line has been chopped into one-round-trip-time pieces stacked vertically with increasing time going down the page. The grey, numbered boxes are packets. The white numbered boxes are the corresponding acks. As each ack arrives, two packets are generated: one for the ack (the ack says a packet has left the system so a new packet is added to take its place) and one because an ack opens the congestion window by one packet. It may be clear from the figure why an add-one-packet-to-window policy opens the window exponentially in time.



2. A sender must not inject a new packet before an old packet has exited.

19

2. Packet Injection. Estimating RTTs

- Do not inject a new packet until an old packet has left.
  - 1. Ack tells us that an old packet has left.
  - 2. Timeout expires tells us also.
    - *Gotta estimate RTT properly*.
- Strategy 1: Fixed RTT.
  - Simple, but probably wrong. (certainly not adaptive)
- Strategy 2: Estimate based on past behavior.

Tactic 0: Mean
Tactic 1: Mean with exponential decay
Tactic 2: Tactic 1 + *safety margin*
safety margin based on current estimate of error in Tactic 1

20

Simple Estimator (RFC793)

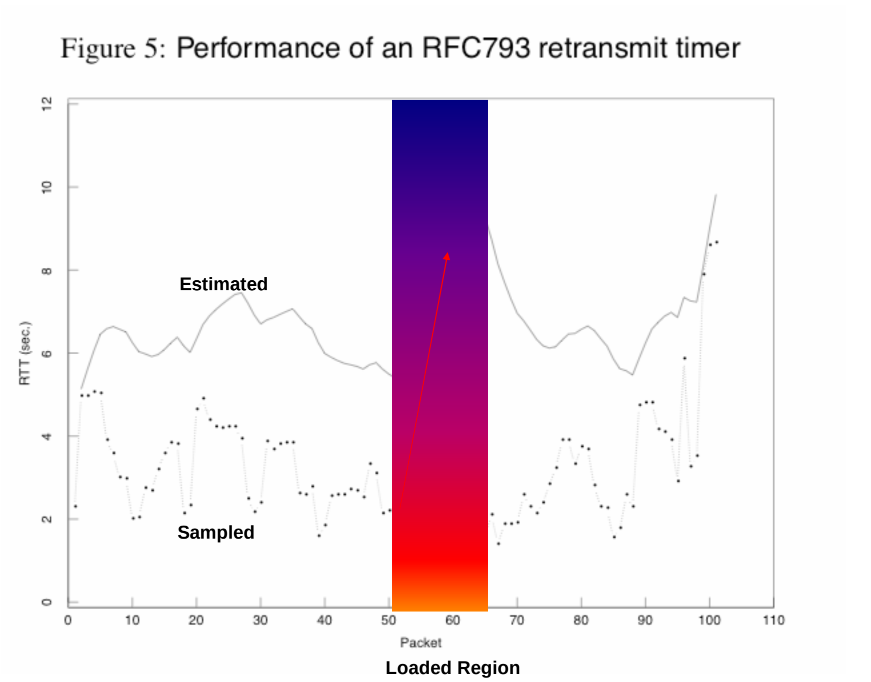
- Simple algorithm:
  - For each packet, note time sent and time ack received
  - Compute RTT samples and average recent samples for timeout

$$EstimatedRTT = (1\text{-}g)(EstimatedRTT) + g(SampleRTT)$$

  - This is an exponentially-weighted moving average (low pass filter) that smoothes the samples with a gain of *g*
    - Big *g* can be jerky (think static on a walkie talkie)
    - Small g can be soothing, but slow to respond (more stable)
    - Typically, g = .1 or .2, --> stable is better than precise
      - In other words, a lousy estimate of the RTT *right now* causes much more damage than an ok estimate right now followed by a better one a little later on.
        - » The Airplane Rule.
  - Conservatively set timeout to small multiple (2) of the estimate in order to accountr for variance

$$Timeout = 2(EstimatedRTT)$$

- Better to wait "too long" than not long enough. (Why?)



Bad Estimators and the Bad Things They Do

- Problem:
  - Variance in RTTs gets large as network gets loaded
  - So an average RTT isn't a good predictor when we need it most
    - Time out too soon, unnecessarily drop another packet onto the network.
    - Timing out too soon occurs during load increase
      - if we time out when load increases but packet not yet lost, then we'll inject another packet onto the network which will increase load, which will cause more timeouts, which will increase load, until we actually starting dropping packets!

23

Jacobson/Karels Algorithm

- EstimatedRTT + "safety margin"
  - large variation in EstimatedRTT --> larger safety margin
- First, estimate how much SampledRTT deviates from EstimatedRTT
  - DevRTT = (1-b) * DevRTT + b * |SampledRTT - EstimatedRTT|
    - typically, b = .25
- Then, set timeout interval as:
  - Timeout = EstimatedRTT + k * DevRTT
  - k is generally set to 4
- Thus,
  - Timeout is close to EstimatedRTT when the Estimate is good,
  - Timeout quickly moves away from EstimatedRTT (4x!) when the Estimate is bad.

24

Estimate with Mean + Variance

Figure 5: Performance of an RFC793 retransmit timer

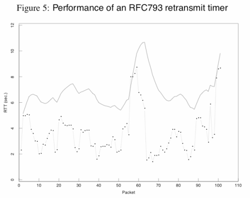
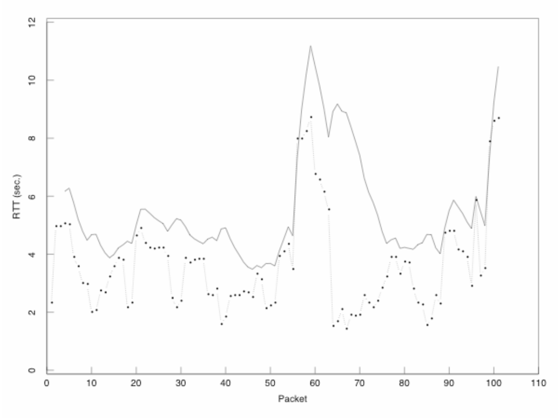
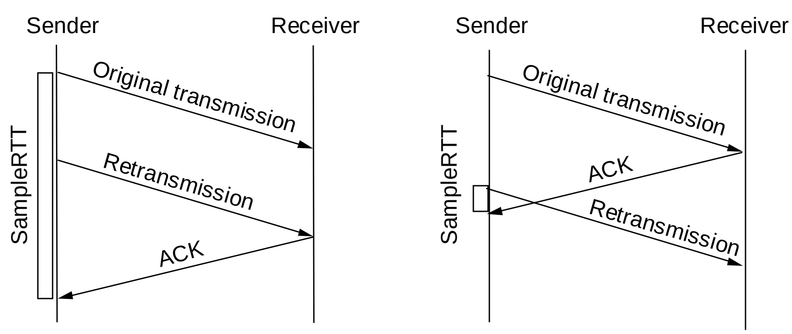


Figure 6: Performance of a Mean+Variance retransmit timer



Karn/Partridge Algorithm

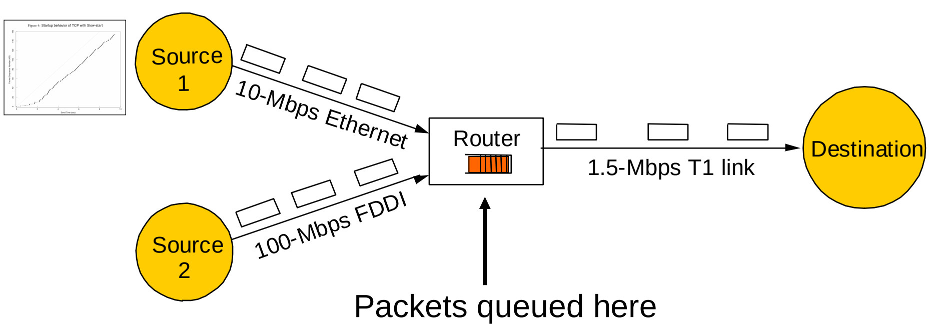
- Problem: RTT for retransmitted packets ambiguous



- Solution: Don’t measure RTT for retransmitted packets and do not relax backed off timeout until valid RTT measurements

3. Equilibrium is lost because of resource contention along the way.

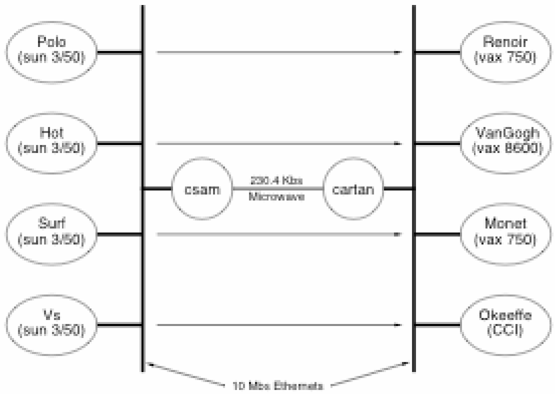
Congestion from Multiple Sources



Packets Lost Here

In Real Life

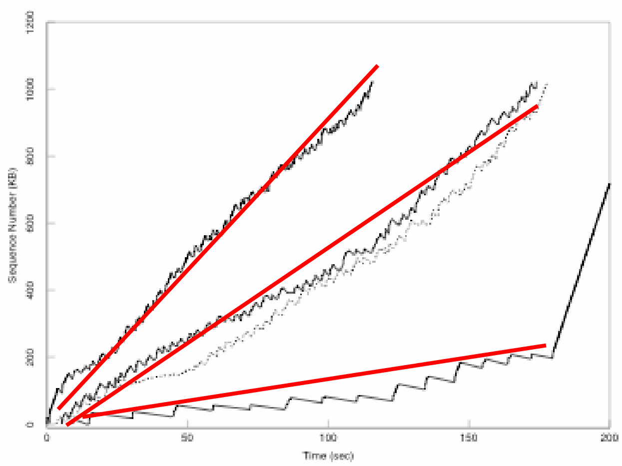
Figure 7: Multiple conversation test setup



29

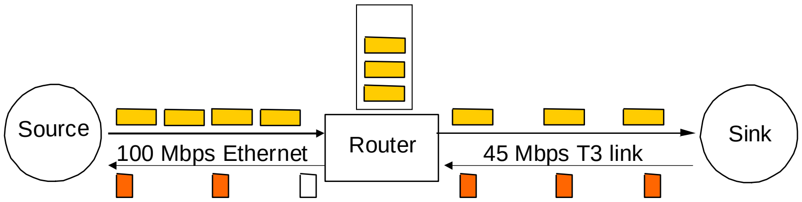
Four Simultaneous Streams

Figure 8: Multiple, simultaneous TCPs with no congestion avoidance



30

TCP is “Self-Clocking”



- Neat observation: acks pace transmissions at approximately the botteneck rate
- So just by sending packets we can discern the “right” sending rate (called the packet-pair technique)

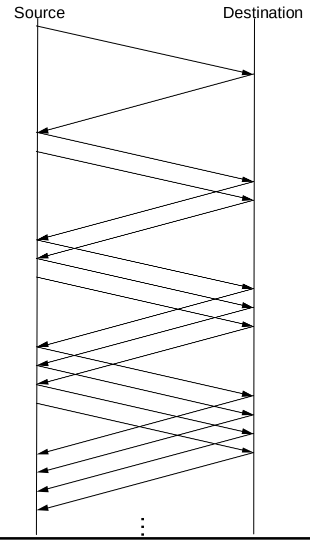
Congestion Control Relies on Signals from the Network

- The network is not saturated: *Send even more*
- The network is saturated: *Send less*

- *ACK* signals that the network is not saturated.
- A Lost packet (no ACK) signals that the network is saturated
  - Assumption here??
- Leads to a simple strategy:
  - On each ack, increase *congestion window* (***additive increase)***
  - On each lost packet, decrease congestion window (***multiplicative decrease)***
- Why increase slowly and decrease quickly?
  - *Respond to good news conservatively, but bad news aggressively*

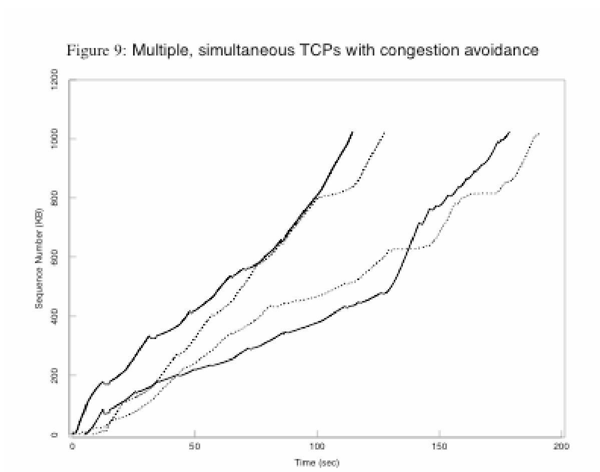
AIMD (Additive Increase/Multiplicative Decrease)

- How to adjust probe rate?
- Increase slowly while we believe there is bandwidth
  - Additive increase per RTT
  - Cwnd += 1 packet / RTT
- Decrease quickly when there is loss (went too far!)
  - Multiplicative decrease
  - Cwnd /= 2



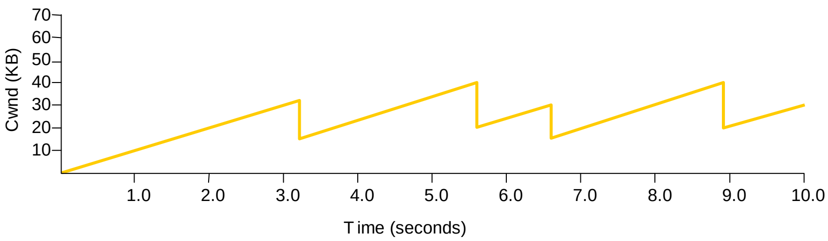
33

With Additive Increase/Multiplicative Decrease



34

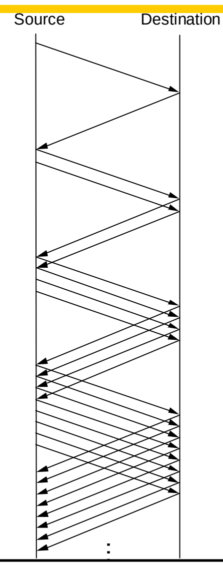
TCP Sawtooth Pattern



35

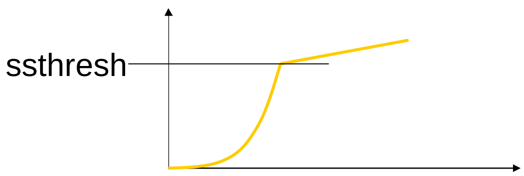
Comparing to “Slow Start”

- Q: What is the ideal value of cwnd? How long will AIMD take to get there?
- Use a different strategy to get close to ideal value
  - Double cwnd every RTT
    - Cwnd *= 2 / RTT
    - As opposed to Cwnd +=1 / packet received



36

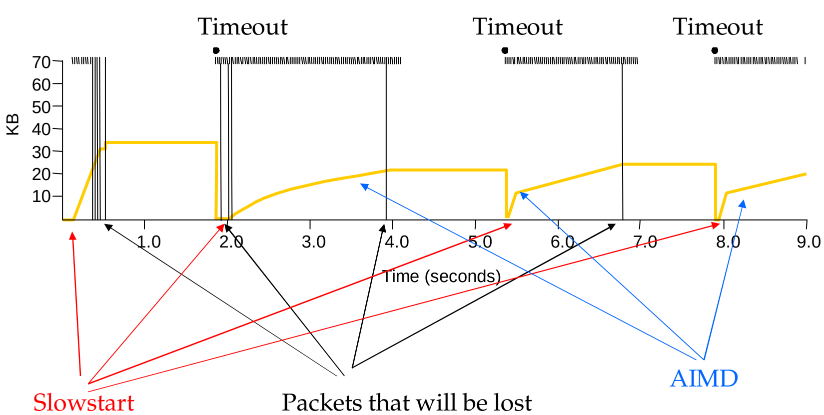
Combining Slow Start and AIMD



- Slow start is used whenever the connection is not running with packets outstanding
  - initially, and after timeouts indicating that there's no data on the wire
- But we don't want to overshoot our ideal cwnd on next slow start, so remember the last cwnd that worked with no loss
  - Ssthresh = cwnd after cwnd /= 2 on loss
  - Switch to AIMD once cwnd passes ssthresh

37

Example (Slow Start +AIMD)



38

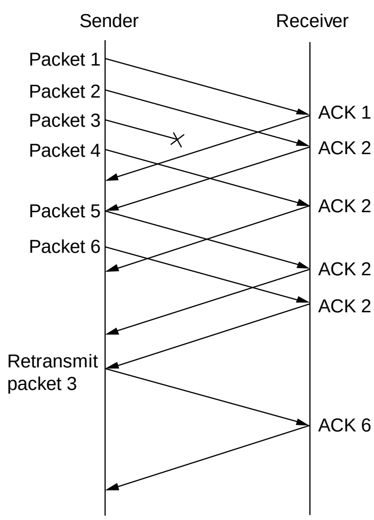
The Long Timeout Problem

- Would like to "signal" a lost packet earlier than timeout
  - enable retransmit sooner
- Can we infer that a packet has been lost?
  - Receiver receives an "out of order packet"
  - Good indicator that the one(s) before have been misplaced
- Receiver generates a duplicate ack on receipt of a misordered packet
- Sender interprets sequence of duplicate acks as a signal that the as-yet-unacked packet has not arrived

39

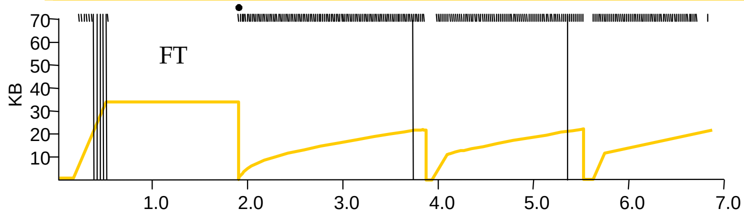
Fast Retransmit

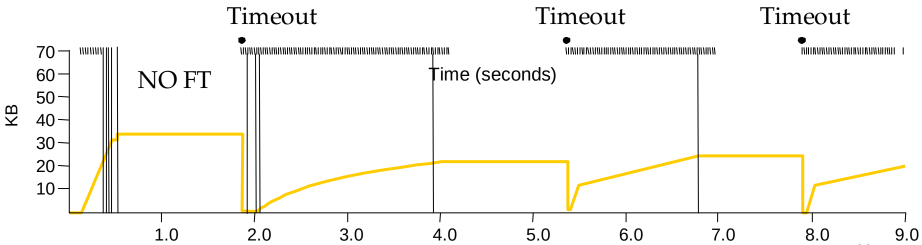
- TCP uses cumulative acks, so duplicate acks start arriving after a packet is lost.
- We can use this fact to infer which packet was lost, instead of waiting for a timeout.
- 3 duplicate acks are used in practice



40

Example (with Fast Retransmit)

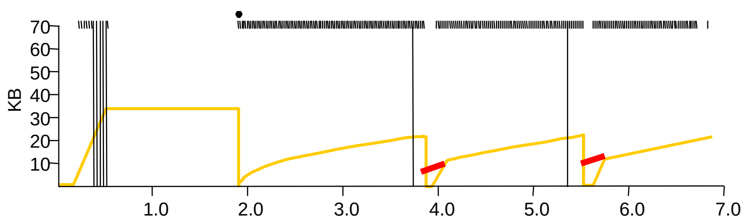




41

Fast Recovery

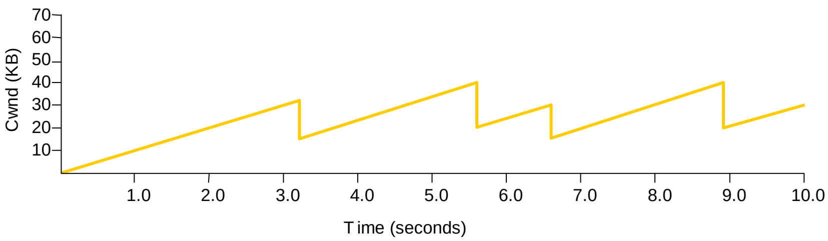
- After Fast Retransmit, use further duplicate acks to grow cwnd and clock out new packets, since these acks represent packets that have left the network.
- End result: Can achieve AIMD when there are single packet losses. Only slow start the first time and on a real timeout.



42

Example (with Fast Recovery)

(Not the same trace as before)



The Familiar Saw Tooth Pattern

Key Concepts

- Packet conservation is a fundamental concept in TCP's congestion management
  - Get to equilibrium
    - *Slow Start*
  - Do nothing to get out of equilibrium
    - *Good RTT Estimate*
  - Adapt when equilibrium has been lost due to other's attempts to get to/stay in equilibrium
    - *Additive Increase/Multiplicative Decrease*
- The Network Reveals Its Own Behavior

Key Concepts (next level down)

- TCP probes the network for bandwidth, assuming that loss signals congestion
- The congestion window is managed to be additive increase / multiplicative decrease
  - It took fast retransmit and fast recovery to get there
- Slow start is used to avoid lengthy initial delays
  - Ramp up to near target rate and then switch to AIMD

A Fast Algorithm for RTT Mean and Variation

- Let *a* = estimated round trip time, v = estimated error, *g* = gain *(0 <g < 1), m* = new sampled round trip time
- $a = (1-g)a + gm$ // compute new estimate using gain
- $a = a + g(m-a)$ // rearrange terms:
  - a is a prediction of next measurement, and (m-a) is the "error" in that prediction.
  - so, the new prediction is the old prediction plus some fraction of the prediction error.
  - The prediction error is the sum of two components:
    - E_r = noise (random unpredictable effects like fluctuations in competing traffic)
    - E_e = bad choice of *a*
    - $a = a + g\,E_r + g\,E_e$
      - The term g E_e kicks *a* in the right direction towards the real estimate
      - The term g E_r kicks it off in the random direction
      - Over many samples, the random errors cancel each other so we get closer and closer to the real estimate
      - **But, g represents a compromise.**
        - » **Big 'g' means that we get a lot of value out of a prediction error, but it also means that the random errors introduce a lot of noise.**
      - **Since g E_e moves *a* in the right direction regardless of *g*, we're better off using a small *g* and waiting a bit longer to get a better estimate than to very quickly get a lousy estimate**

- *Or,*
  - $Err = (m - a)$ *// Sampled Error*
  - $a = a + g\,(Err)$ *// Estimate of round trip time*
  - $v = v + g(|Err| - v)$ *// Estimate of error*
- Not necessary to use same gain; in general want to force timer to go up