CSE/EE 461 – Lecture 17

TCP Congestion Control

David Wetherall
djw@cs.washington.edu

Last Time …

- The Transport Layer

- Focus
  - How do we <u>allocate bandwidth</u>?

- Topics
  - Congestion
  - Fairness

| Application |
| Presentation |
| Session |
| Transport |
| Network |
| Data Link |
| Physical |

1

This Lecture

- The Transport Layer

- Focus
  - How does TCP <u>share bandwidth</u>?

- Topics
  - Additive Increase/Multiplicative Decrease
  - Slow Start
  - Fast Recovery

Application
Presentation
Session
Transport
Network
Data Link
Physical

TCP Before Congestion Control

- Just use a fixed size sliding window!
  - Will under-utilize the network or cause unnecessary loss

- Congestion control dynamically varies the size of the window to match sending and available bandwidth
  - Sliding window uses minimum of cwnd, the congestion window, and the advertised flow control window

- The big question: how do we decide what size the window should be?
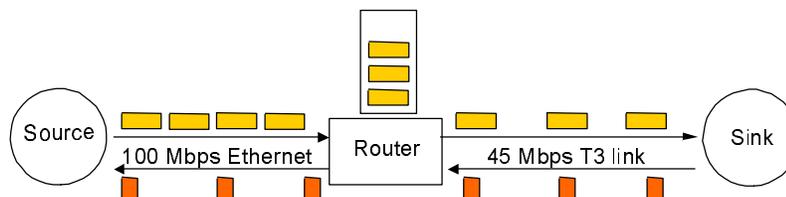
TCP Probes the Network
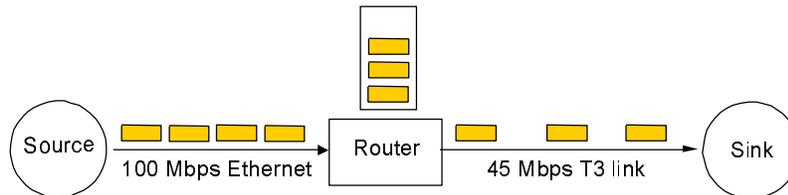
- Each source independently probes the network to determine how much bandwidth is available
  - Changes over time, since everyone does this
- Assume that packet loss implies congestion
  - Since errors are rare; also, requires no support from routers

TCP is "Self-Clocking"

- Neat observation: acks pace transmissions at approximately the botteneck rate
- So just be sending packets we can discern the "right" sending rate (called the packet-pair technique)
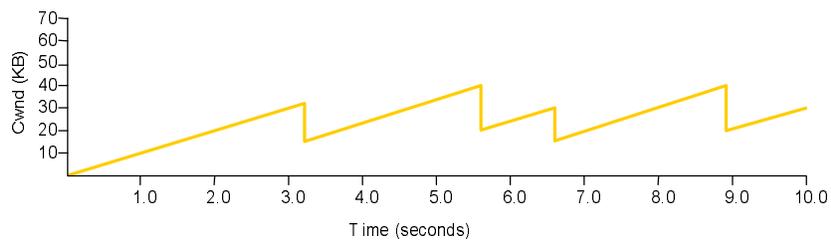
AIMD (Additive Increase/Multiplicative Decrease)

- 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

Source   Destination
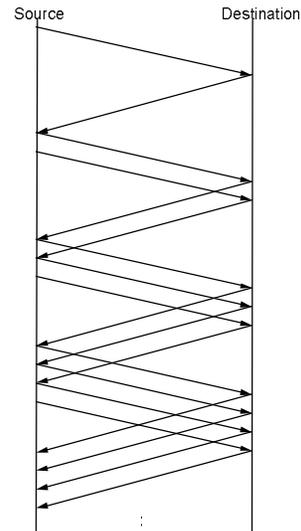
TCP Sawtooth Pattern

Cwnd (KB) vs Time (seconds)
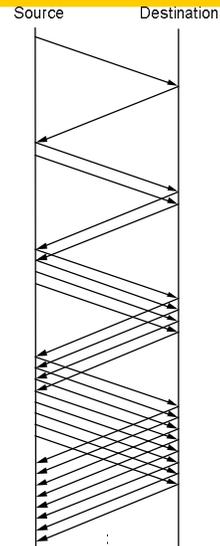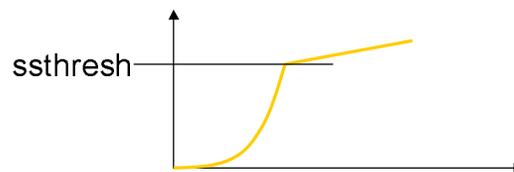
"Slow Start"

- 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
  - Cwnd +=1 / packet received

Source   Destination

L17.9
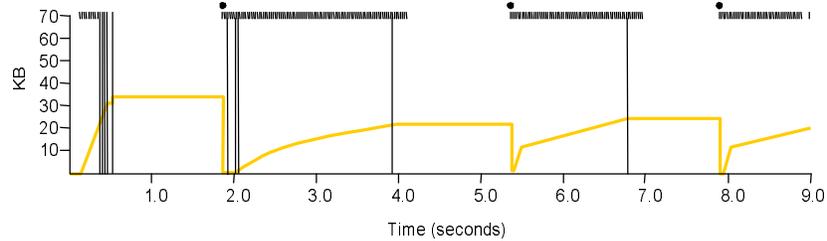

Combining Slow Start and AIMD

ssthresh ─

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

L17.10

Example (Slow Start +AIMD)
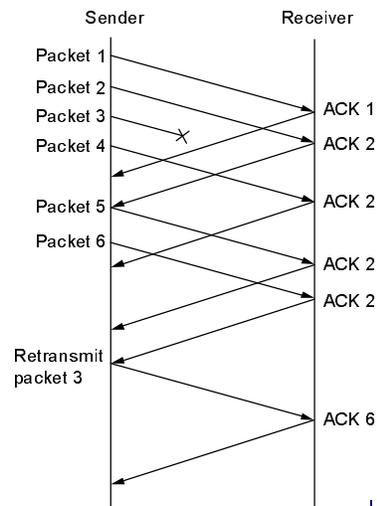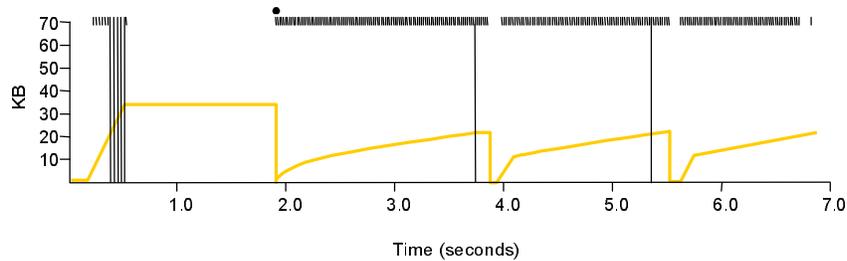
Fast Retransmit

- 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

Sender   Receiver

Packet 1
Packet 2   ACK 1
Packet 3   ACK 2
Packet 4   ACK 2
Packet 5   ACK 2
Packet 6   ACK 2
                  ACK 2
Retransmit
packet 3
                  ACK 6

6

Example (with Fast Retransmit)
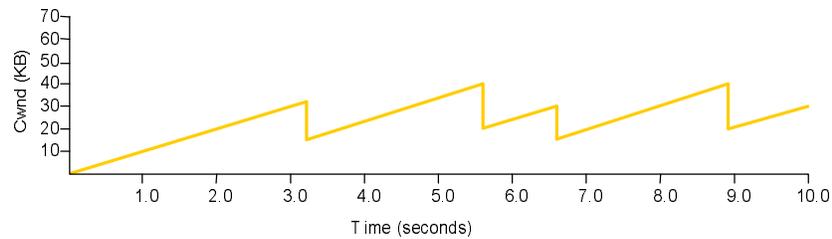

KB

Time (seconds)

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.

Example (with Fast Recovery)

Key Concepts

- 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