Introduction to Computer Networks

COSC 4377

Lecture 9

Spring 2012

February 15, 2012

Announcements

- HW4 due today
- Start working on HW5
- In-class student presentations
- TA office hours this week
  - TR 1030a – 100p

Today's Topics

- HW4 and HW5 discussions
- Transport Protocols
  - Flow Control
  - Congestion Control

HW4 and HW5

- Web Server Concurrency
  - Fork vs Thread vs Select
- Use large file for testing

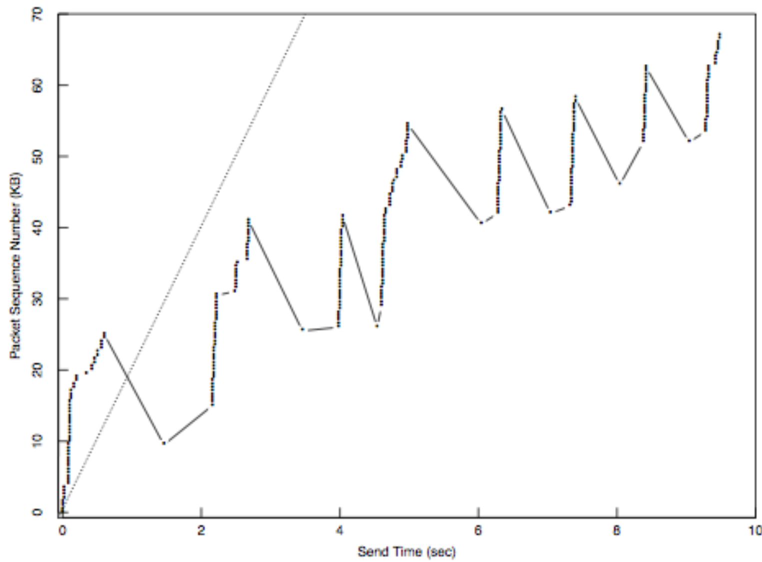
Flow Control

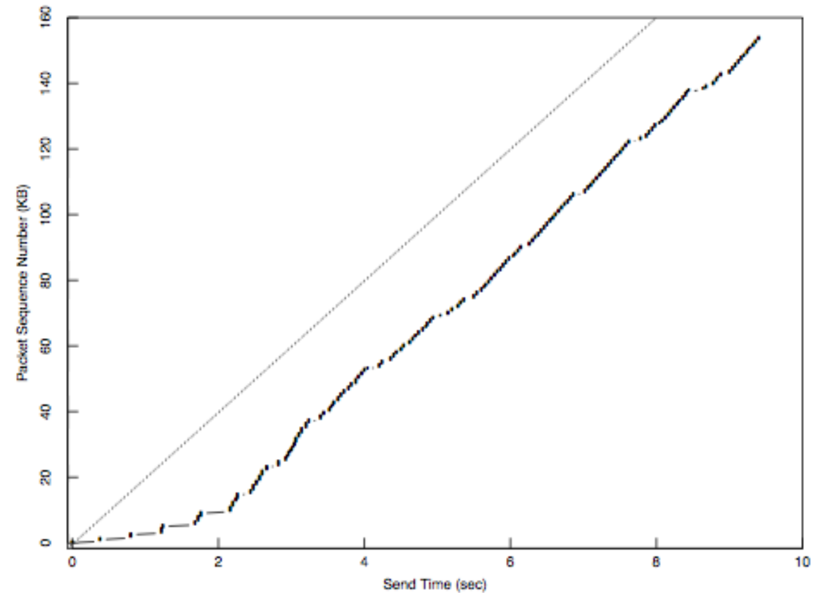
- Goal: not send more data than the receiver can handle
- Sliding window protocol
- Receiver uses window header field to tell sender how much space it has

Congestion Control

- Goal: do not send more data than the network can take
- 3 Key Challenges
  - Determining the available capacity
  - Adjusting to changes in the available capacity
  - Sharing capacity between flows

Slow Start





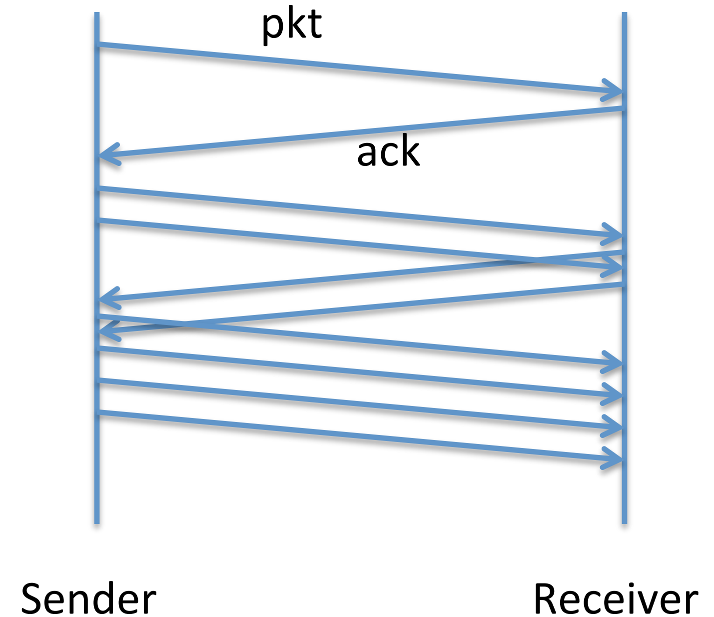
From [Jacobson88]

Slow start implementation

- Let w be the size of the window in *bytes*
  - We have w/MSS segments per RTT
- We are doubling w after each RTT
  - We receive w/MSS ACKs each RTT
  - So we can set w = w + MSS on every ack
- At some point we hit the network limit.
  - Experience loss
  - We are at most one window size above the limit
  - Remember this: ssthresh and reduce window

Slow Start

- We double cwnd every round trip
- We are still sending min (cwnd,rcvwnd) pkts
- Continue until ssthresh estimate or pkt drop



http://en.wikipedia.org/wiki/Slow-start

Dealing with Congestion

- Assume losses are due to congestion
- After a loss, reduce congestion window
  - How much to reduce?
- Idea: conservation of packets at equilibrium
  - Want to keep roughly the same number of packets network
  - Analogy with water in fixed-size pipe
  - Put new packet into network when one exits

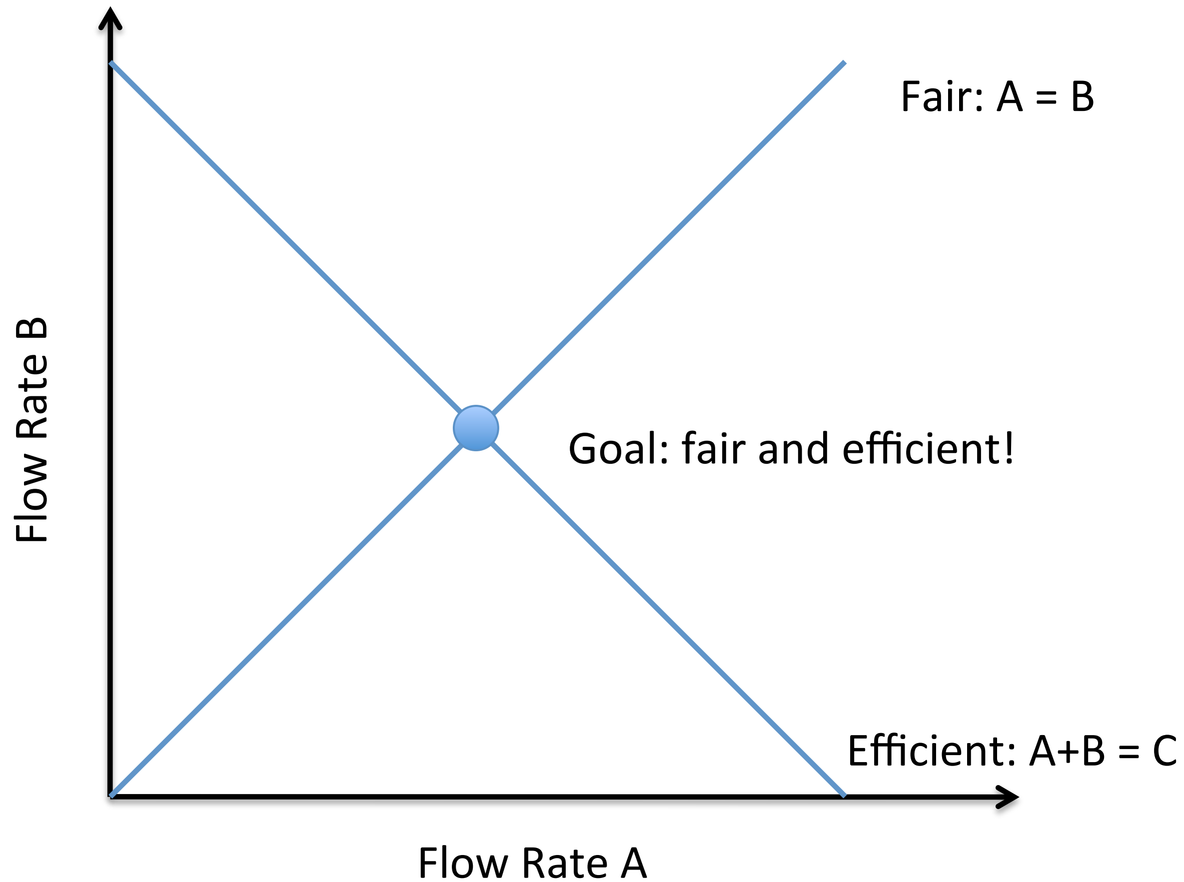
How much to reduce window?

- What happens under congestion?
  - Exponential increase in congestion
- Sources must decrease offered rate exponentially
  - i.e, multiplicative decrease in window size
  - TCP chooses to cut window in half

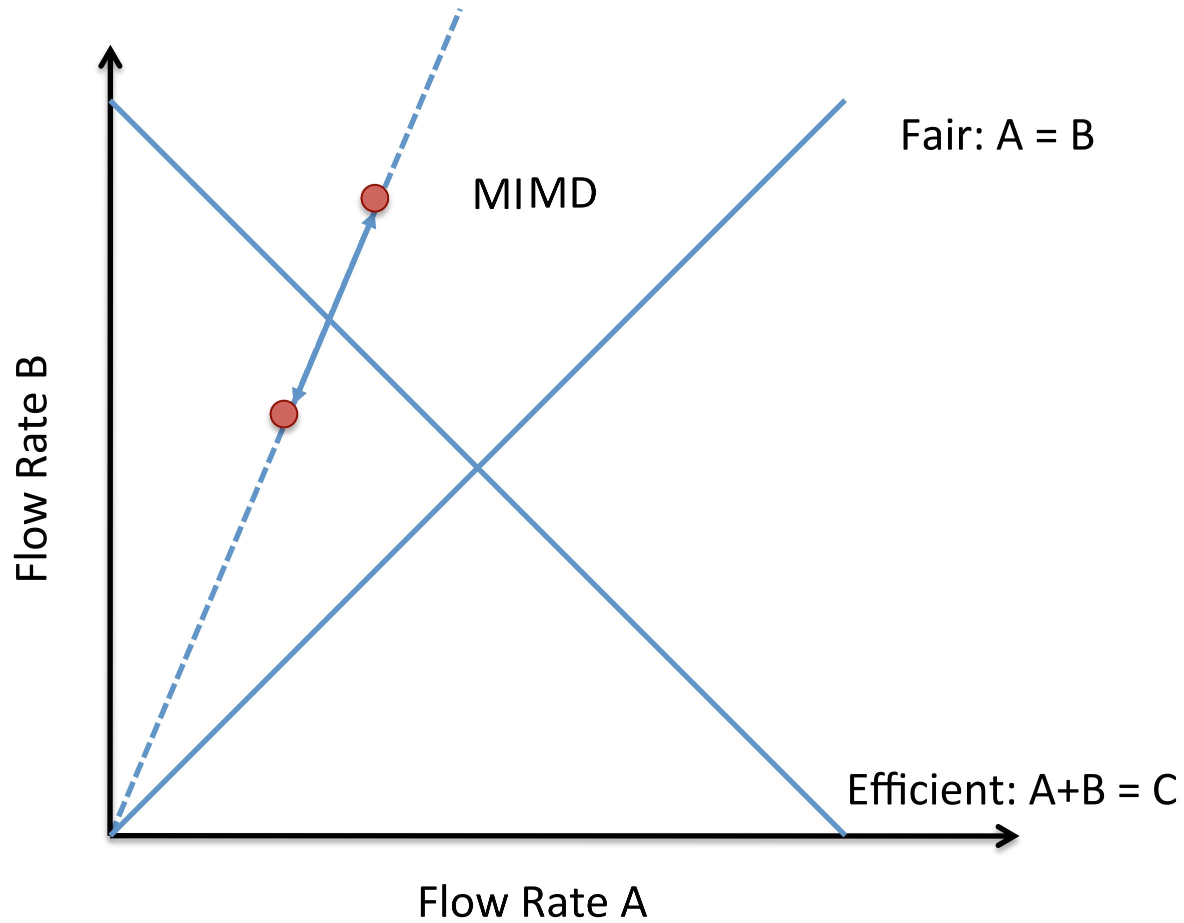
How to use extra capacity?

- Network signals congestion, but says nothing of underutilization
  - Senders constantly try to send faster, see if it works
  - So, increase window if no losses… By how much?
- Multiplicative increase?
  - Easier to saturate the network than to recover
  - Too fast, will lead to saturation, wild fluctuations
- Additive increase?
  - Won't saturate the network

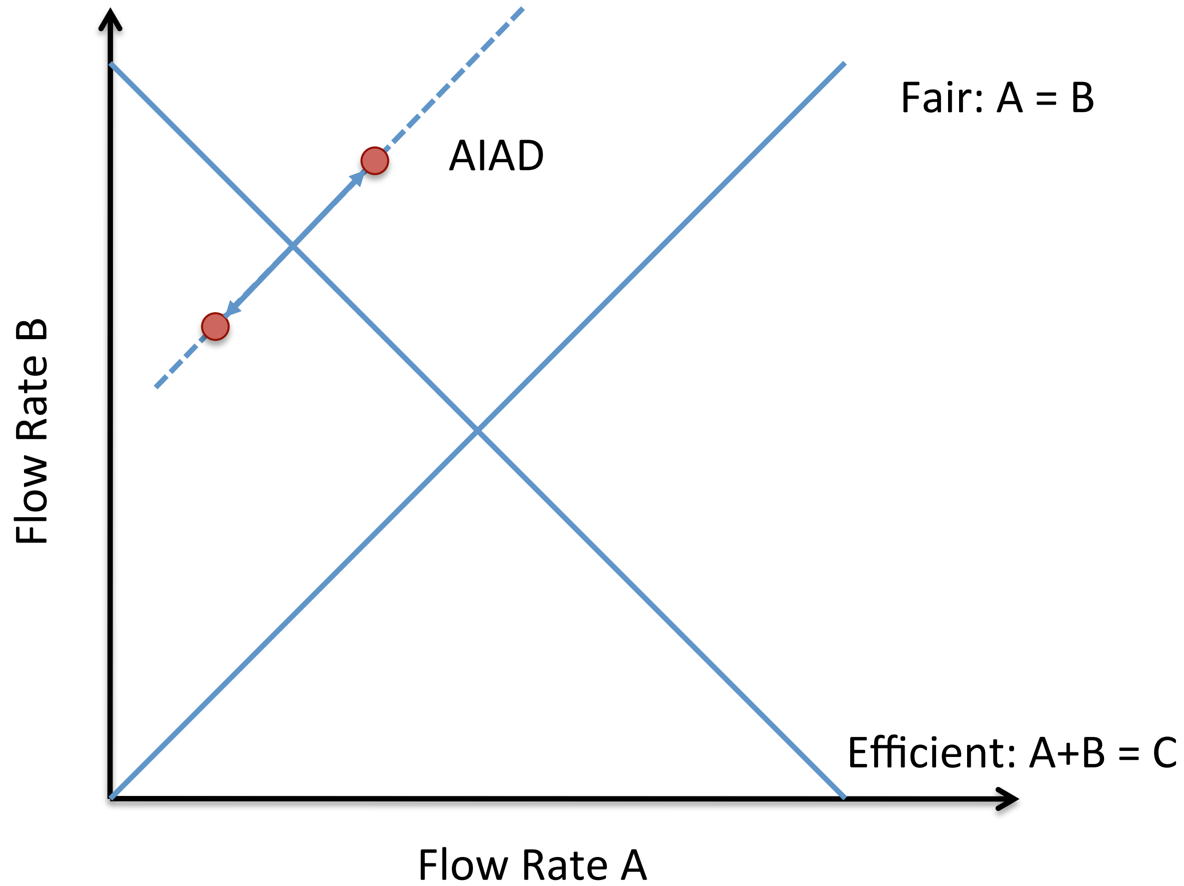
Chiu Jain Phase Plots



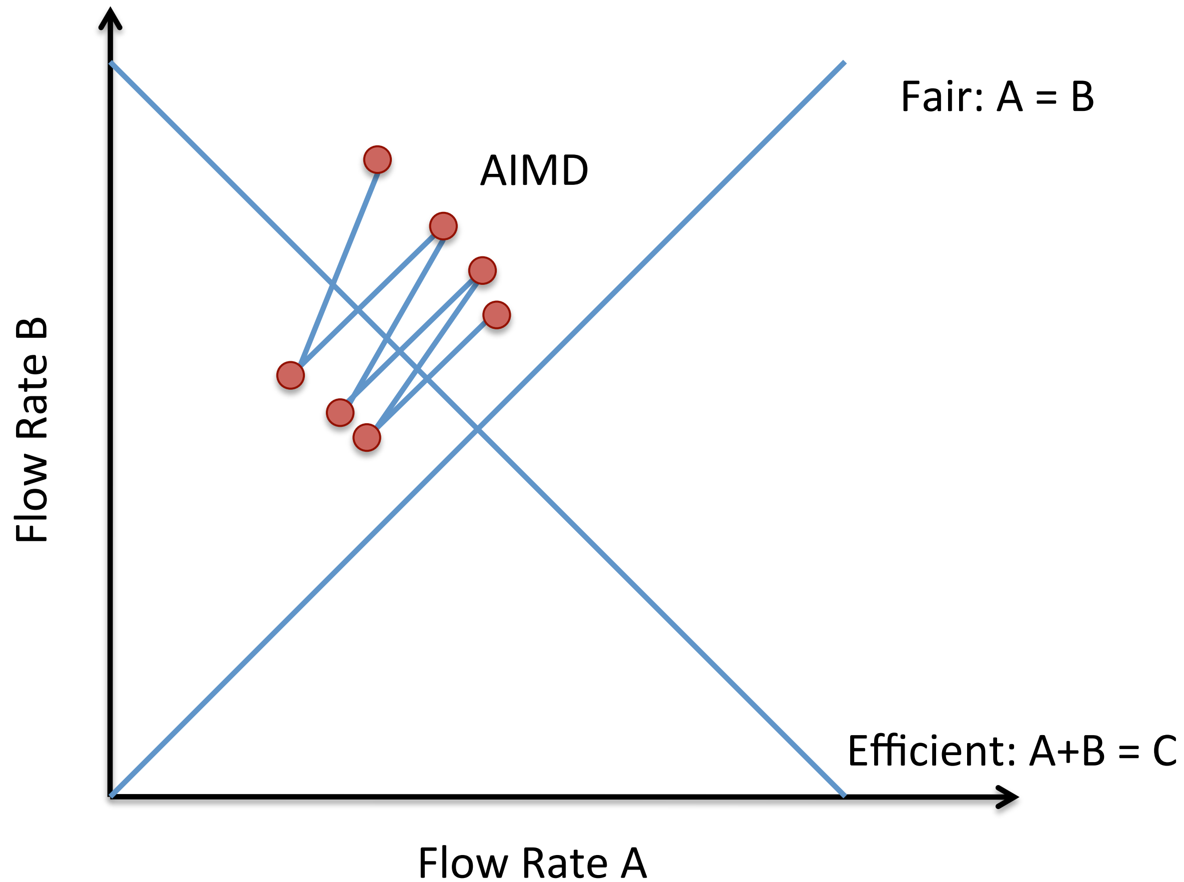
Chiu Jain Phase Plots



Chiu Jain Phase Plots



Chiu Jain Phase Plots

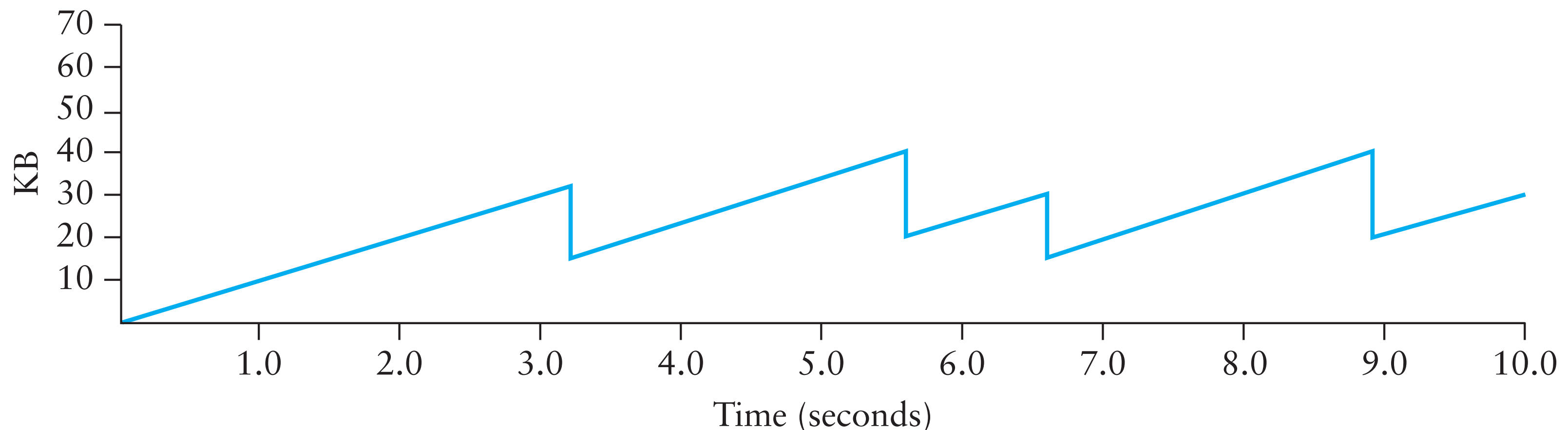


AIMD Implementation

- In practice, send MSS-sized segments
  - Let window size in bytes be w (a multiple of MSS)
- Increase:
  - After w bytes ACKed, could set w = w + MSS
  - Smoother to increment on each ACK
    - w = w + MSS * MSS/w
    - (receive w/MSS ACKs per RTT, increase by MSS/(w/MSS) for each)
- Decrease:
  - After a packet loss, w = w/2
  - But don't want w < MSS
  - So react differently to multiple consecutive losses
  - Back off exponentially (pause with no packets in flight)

AIMD Trace

- AIMD produces sawtooth pattern of window size
  - Always probing available bandwidth



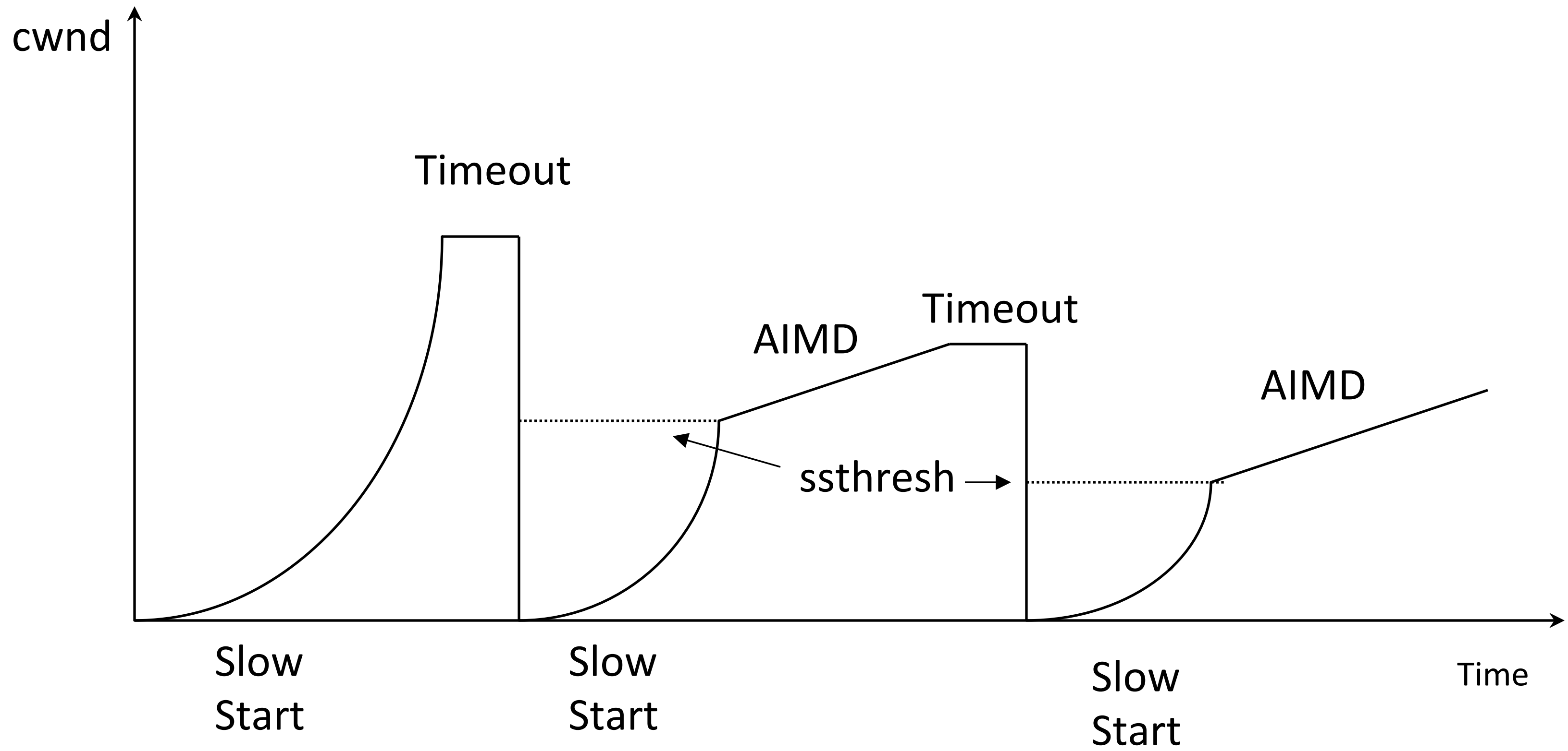
Putting it together

- TCP has two states: Slow Start (SS) and Congestion Avoidance (CA)
- A window size threshold governs the state transition
  - Window <= threshold: SS
  - Window > threshold: congestion avoidance
- States differ in how they respond to ACKs
  - Slow start: w = w + MSS
  - Congestion Avoidance: $w = w + MSS^2/w$ (1 MSS per RTT)
- On loss event: set w = 1, slow start

How to Detect Loss

- Timeout
- Any other way?
  - Gap in sequence numbers at receiver
  - Receiver uses cumulative ACKs: drops => duplicate ACKs
- 3 Duplicate ACKs considered loss

Putting it all together



RTT

- We want an estimate of RTT so we can know a packet was likely lost, and not just delayed
- Key for correct operation
- Challenge: RTT can be highly variable
  - Both at long and short time scales!
- Both average and variance increase a lot with load
- Solution
  - Use exponentially weighted moving average (EWMA)
  - Estimate deviation as well as expected value
  - Assume packet is lost when time is well beyond reasonable deviation

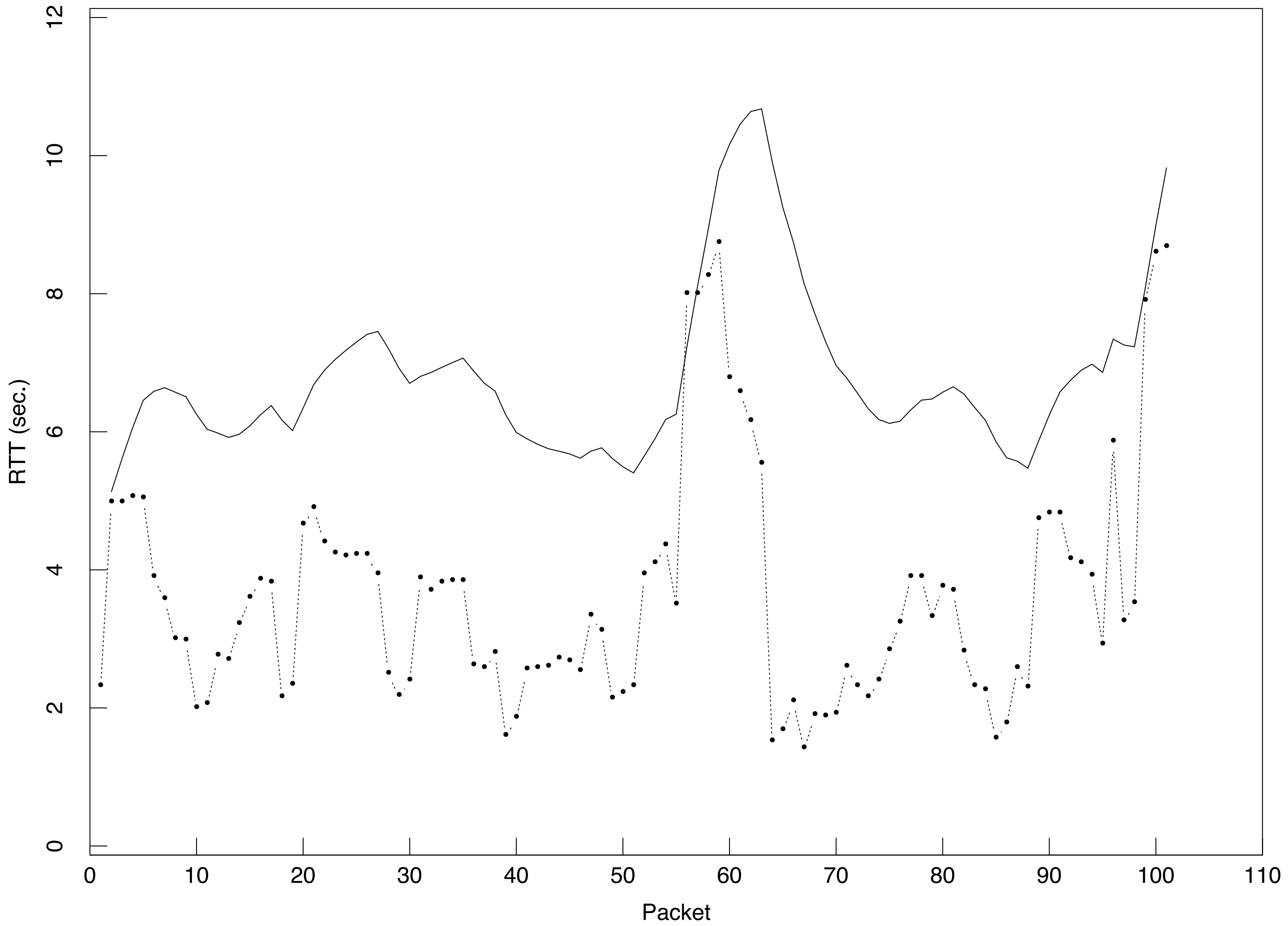
Originally

- $EstRTT = (1 - \alpha) \times EstRTT + \alpha \times SampleRTT$
- $Timeout = 2 \times EstRTT$
- Problem 1:
  - in case of retransmission, ack corresponds to which send?
  - Solution: only sample for segments with no retransmission
- Problem 2:
  - does not take variance into account: too aggressive when there is more load!

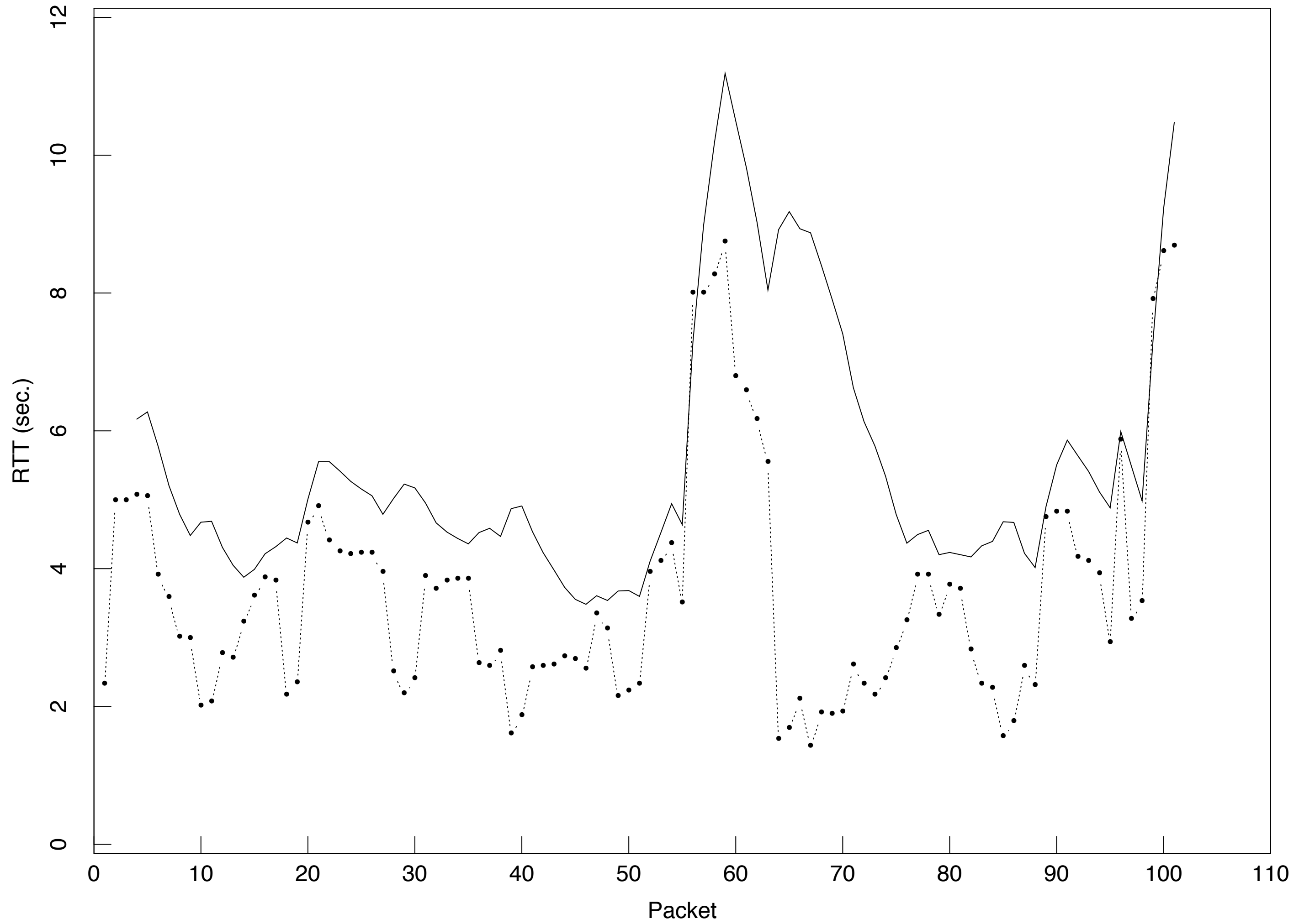
Jacobson/Karels Algorithm (Tahoe)

- $EstRTT = (1 - \alpha) \times EstRTT + \alpha \times SampleRTT$
  - Recommended α is 0.125
- $DevRTT = (1 - \beta) \times DevRTT + \beta \mid SampleRTT - EstRTT \mid$
  - Recommended β is 0.25
- Timeout = EstRTT + 4 DevRTT
- For successive retransmissions: use exponential backoff

Old RTT Estimation



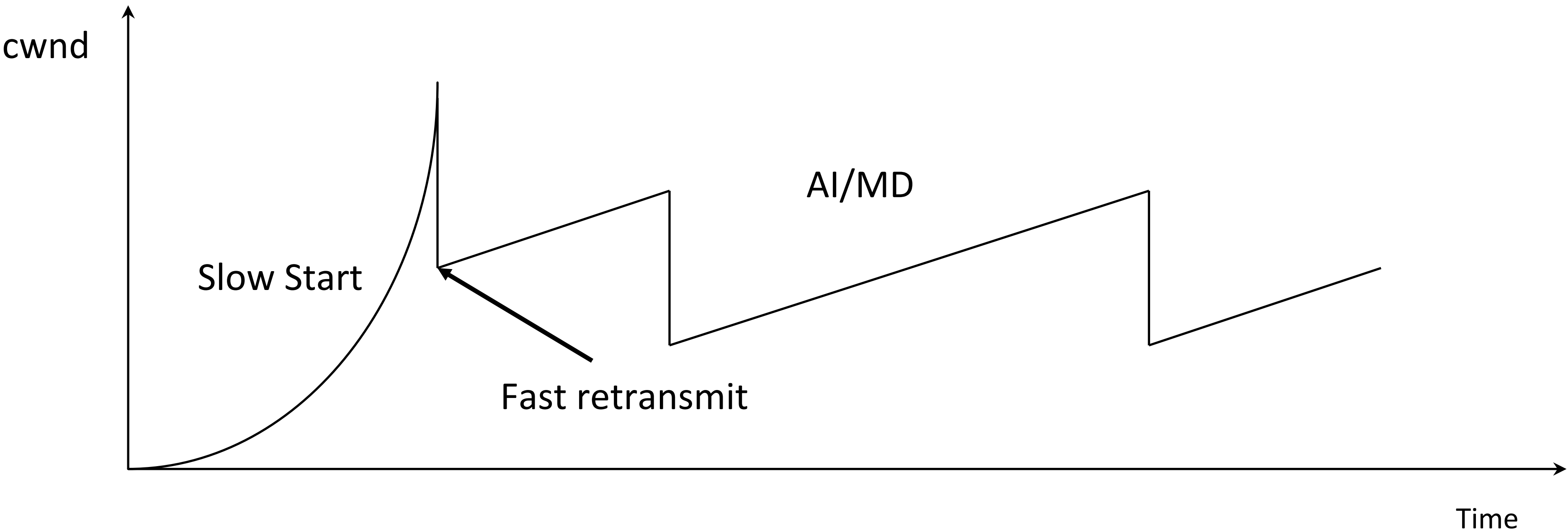
Tahoe RTT Estimation



Slow start every time?!

- Losses have large effect on throughput
- Fast Recovery (TCP Reno)
  - Same as TCP Tahoe on Timeout: w = 1, slow start
  - On triple duplicate ACKs: w = w/2
  - Retransmit missing segment (fast retransmit)
  - Stay in Congestion Avoidance mode

Fast Recovery and Fast Retransmit



3 Challenges Revisited

- Determining the available capacity in the first place
  - Exponential increase in congestion window
- Adjusting to changes in the available capacity
  - Slow probing, AIMD
- Sharing capacity between flows
  - AIMD
- Detecting Congestion
  - Timeout based on RTT
  - Triple duplicate acknowledgments
- Fast retransmit/Fast recovery
  - Reduces slow starts, timeouts