

# **Lecture 4: Congestion Control**

# Overview

- Internet is a network of networks
- Narrow waist of IP: unreliable, best-effort datagram delivery
- Packet forwarding: input port to output port
- Routing protocols: computing port mappings
- Transport: end-to-end, reliability, flow control, and congestion control

# Transport

- Provides end-to-end communication between applications
- UDP: unreliable datagram delivery (thin layer on top of IP)
- TCP: reliable stream delivery

# TCP

- **Connection establishment**
- **Connection teardown**
- **Retransmission timeout (RTT and variance)**
- **Flow control (receiver window)**
- **Congestion control (transmit window)**

# TCP

- Connection establishment
- Connection teardown
- Retransmission timeout (RTT and variance)
- Flow control (receiver window)
- Congestion control (transmit window)

## **A Bit of History**

- **1974: 3-way handshake**
- **1978: TCP and IP split into TCP/IP**
- **1983: January 1, ARPAnet switches to TCP/IP**
- **1986: Internet begins to suffer congestion collapses**
- **1987-8: Van Jacobson fixes TCP, publishes seminal paper (Tahoe)**
- **1990: Fast recovery and fast retransmit added (Reno)**

## Three questions

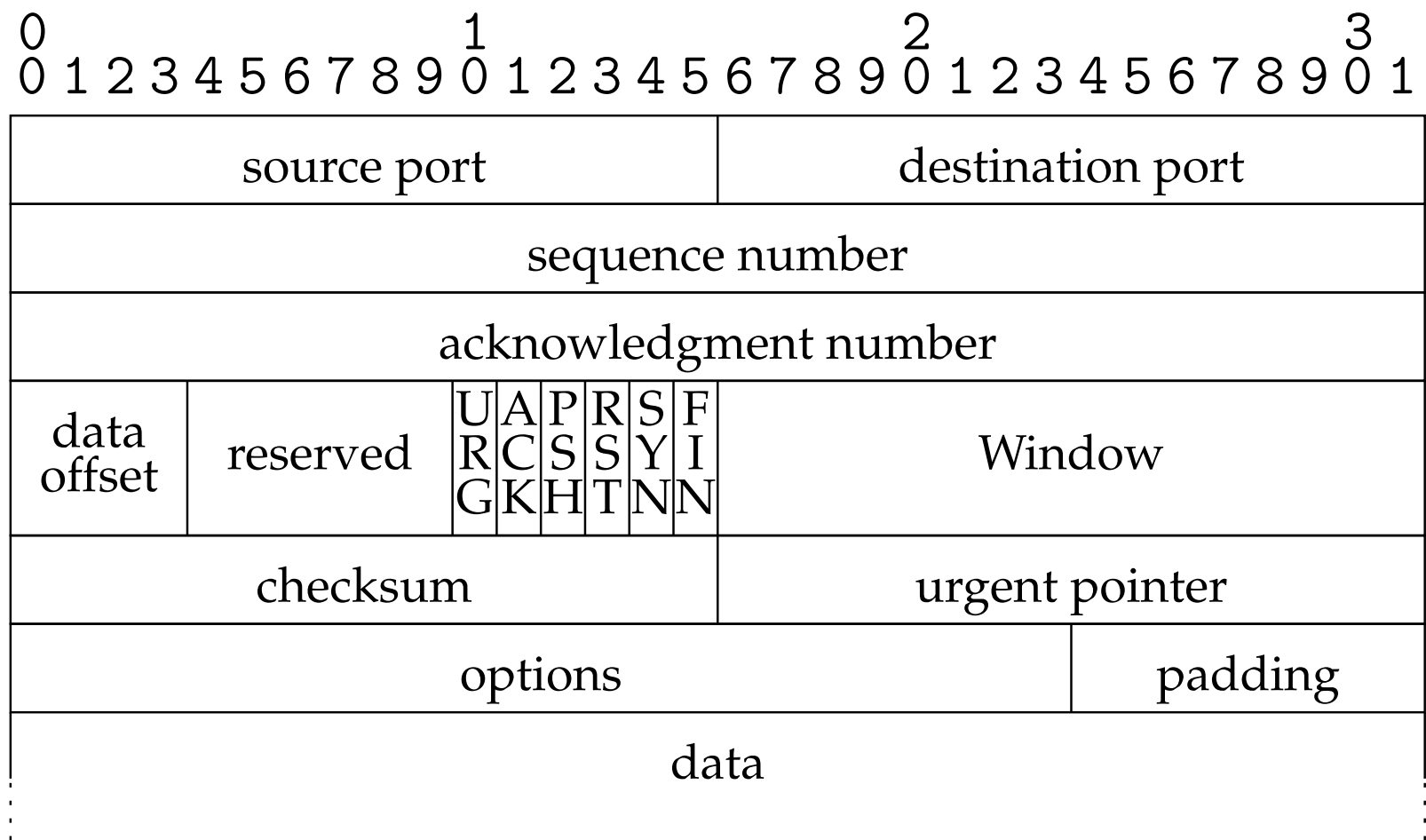
- Goal: maintain TCP goodput at equilibrium
- When does TCP retransmit packets?
- When does TCP transmit packets?
- When does TCP ack packets?

# Flow Control

- Part of TCP specification (pre-1988)
- Want to make sure we don't send more than what the receiver can handle
- Sliding window protocol as described in lectures 2 and 3
- Use `window` header field to tell other side how much space you have
- Rule: Sent and unacknowledged bytes  $\leq$  `window`



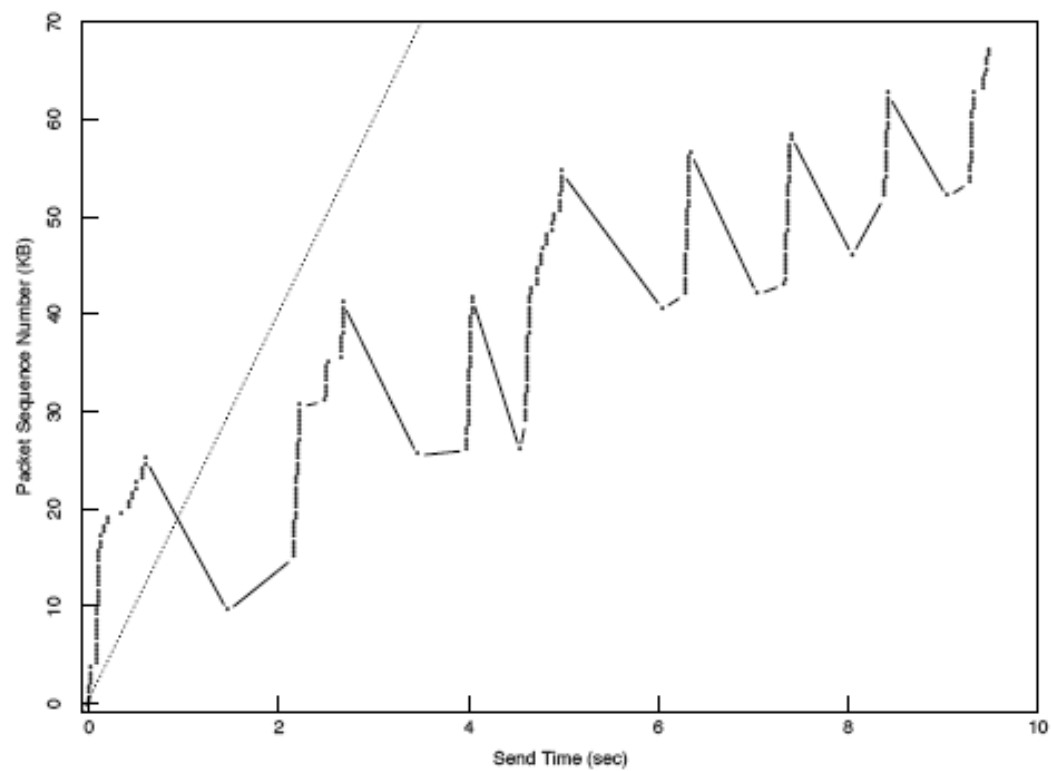
# TCP segment



# Send Timing

- **Before Tahoe**
  - On connection, nodes send full window of packets
  - Retransmit packet immediately after its timer expires
- **Result: window-sized bursts of packets in network**

# Bursts of packets



## Retransmission

- TCP dynamically estimates round trip time
- If segment goes unacknowledged, must retransmit
- Use exponential backoff (in case loss from congestion)
- After  $\sim 10$  minutes, give up and reset connection

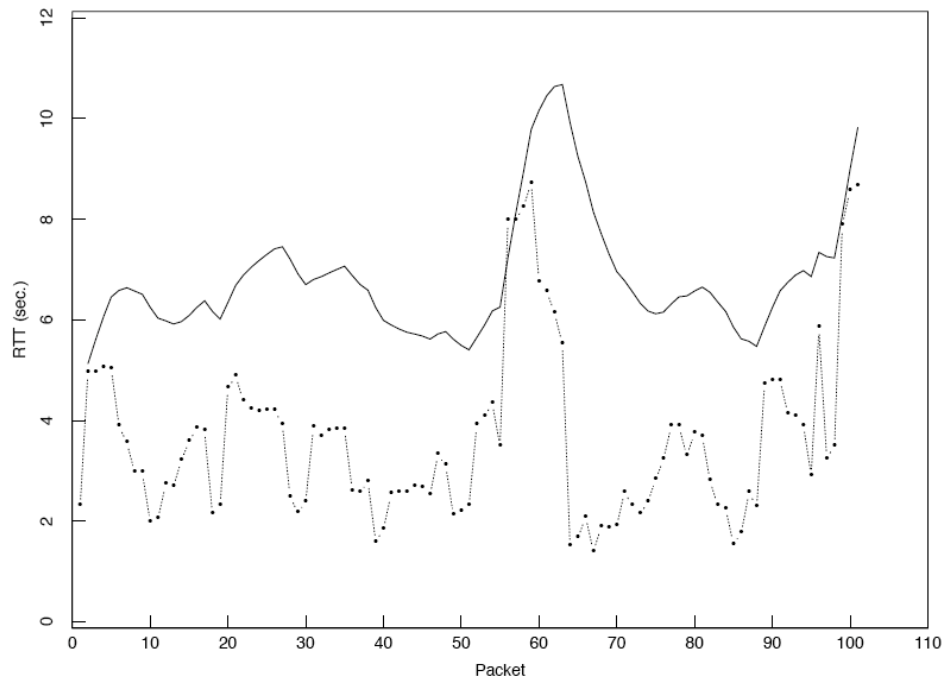
# Round Trip Time (RTT)

- We want to estimate RTT so we can know a packet was likely lost, not just delayed
- The challenge is that RTTs for individual packets can be highly variable, both on long-term and short-term time scales
- Can increase significantly 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

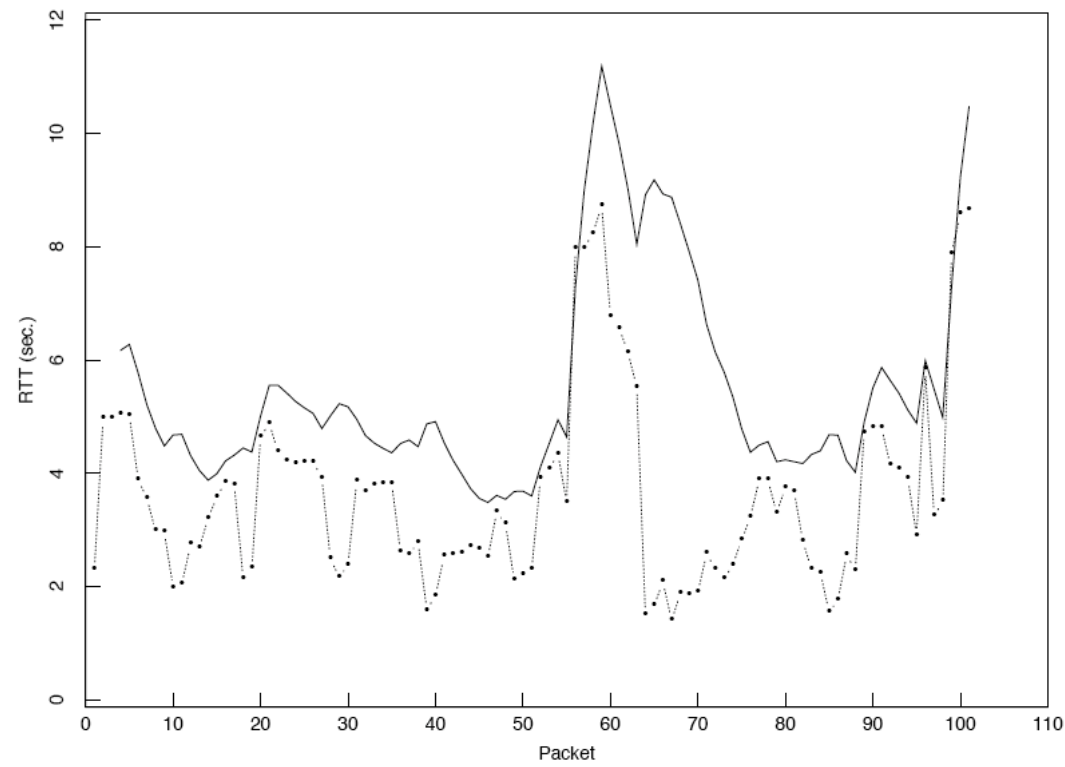
## RTT with EWMA

- $Est_{RTT} = (1 - \alpha) \cdot Est_{RTT} + \alpha \cdot Sample_{RTT}$ 
  - Recommended  $\alpha$  is 0.125
- $Dev_{RTT} = (1 - \beta) \cdot Dev_{RTT} + \beta \cdot |Sample_{RTT} - Est_{RTT}|$ 
  - Recommended  $\beta$  is 0.25
- **Timeout is**  $Est_{RTT} + 4 \cdot Dev_{RTT}$

# Old RTT estimation



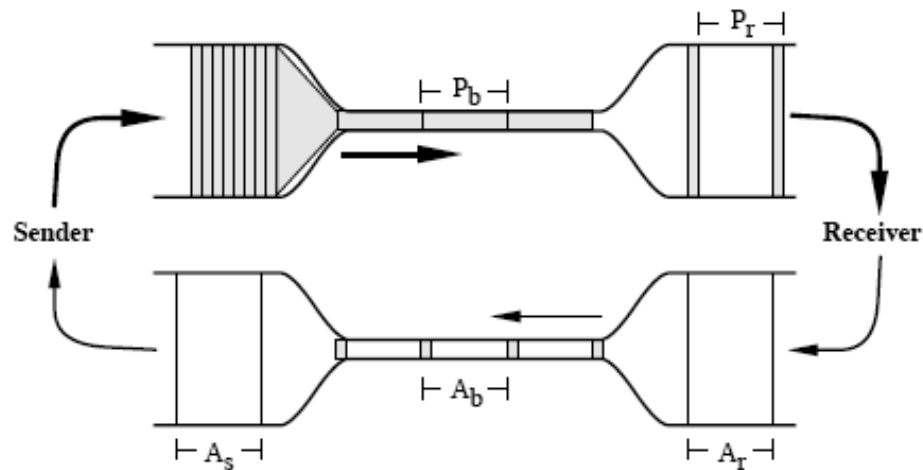
# Tahoe RTT estimation





# Self-Clocking

- Goal is *conservation of packets* in steady state
  - A new packet isn't put into the network until an old one leaves
  - Very effective in avoiding and controlling congestion
- Solution: send a data packet for each ack



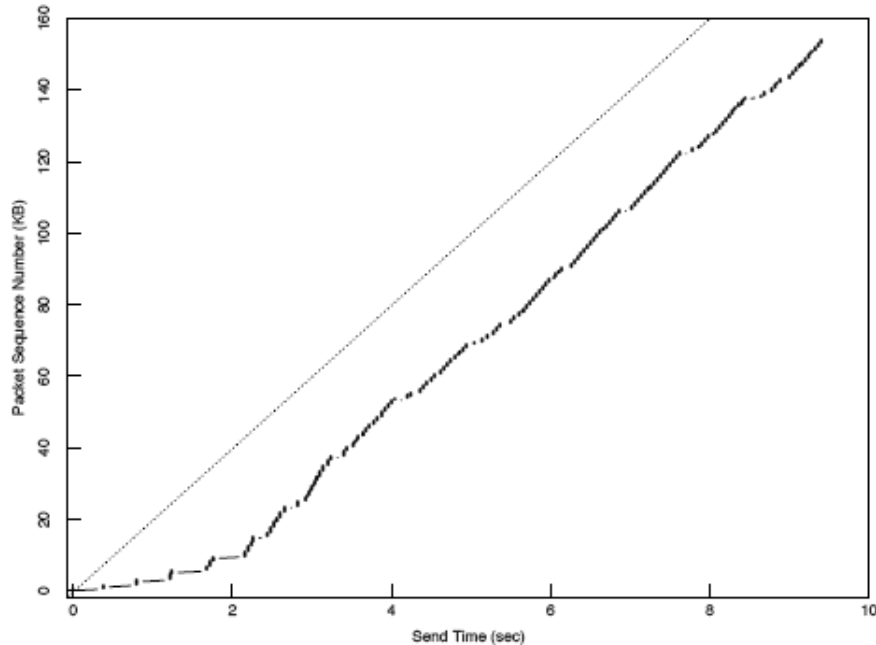
## Self-Clocking, Continued

- Assuming good RTT estimation, self-clocking prevents congestion by making `window` limit the number of packets in the network
- Very simple to implement
- But how big should the window be?

## Slow start

- But how do you start?
- Introduce a *congestion window* to connection state, `cwnd`
- Rule: Sent and unacknowledged bytes  $\leq$  `cwnd`
- On start or loss, set `cwnd` to one packet (or... more on this later)
- On each data ack, increase `cwnd` by one packet
- Prevents bursts of packets

# With Slow Start



- Slow start allows connection to quickly reach equilibrium
- How do we maintain it?

## Two States

- **TCP has two states: Slow Start (SS) and Congestion Avoidance (CA)**
- **A window size threshold governs the state transition**
  - Window  $\leq$  threshold: slow start
  - Window  $>$  threshold: collision avoidance
- **States differ in how they respond to new acks**
  - Slow start:  $cwnd += MSS$
  - Congestion avoidance:  $cwnd += \frac{MSS^2}{cwnd}$  (MSS every RTT)

# Congestion Control

- **Two conflicting goals**
  - Provider: high utilization
  - User: fairness among users
- **Want to converge to a state where everyone gets  $\frac{1}{N}$**
- **Avoid congestion collapse**

# Taming Congestion

- The optimal congestion window is the bandwidth/delay product
- Sender learns this by adapting window size
- Senders must slow down when there is congestion
- Absence of ACKs (timeout) implies serious congestion
- Duplicate ACKs imply some congestion

# Responding to Loss

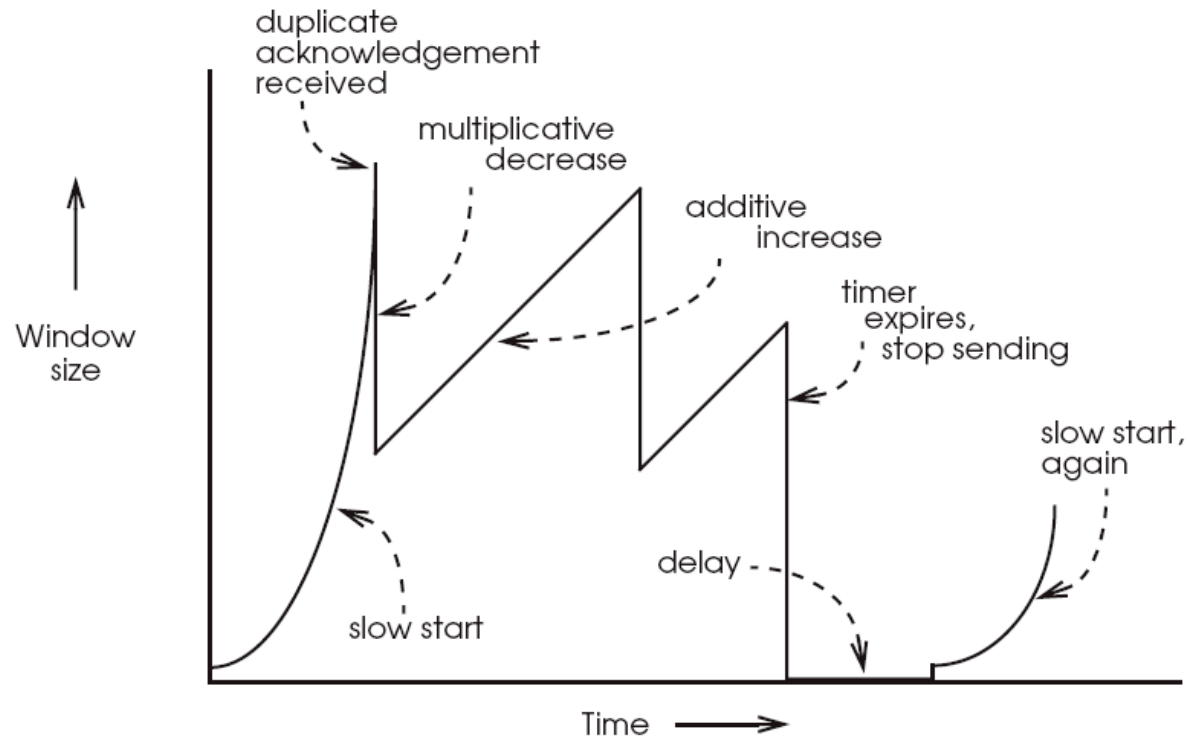
- Set threshold to  $\frac{cwnd}{2}$
- On timeout
  - Set cwnd to 1
  - Causes TCP to enter slow start
- On triple duplicate ACK (Reno)
  - Set cwnd to  $\frac{cwnd}{2}$
  - Retransmit missing segment
  - Causes TCP to stay in congestion avoidance



# AIMD

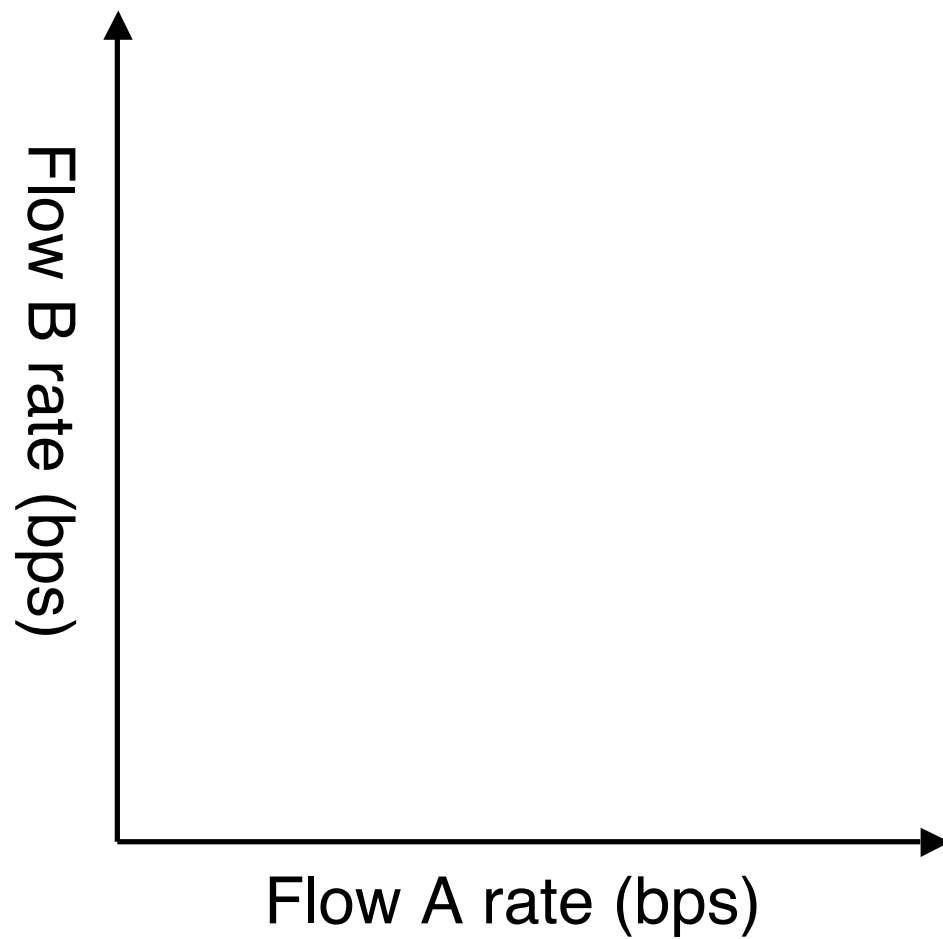
- Additive increase, multiplicative decrease
- Behavior in congestion avoidance mode
- Why AIMD? Remember goals:
  - Link utilization
  - Fairness

# AIMD in action

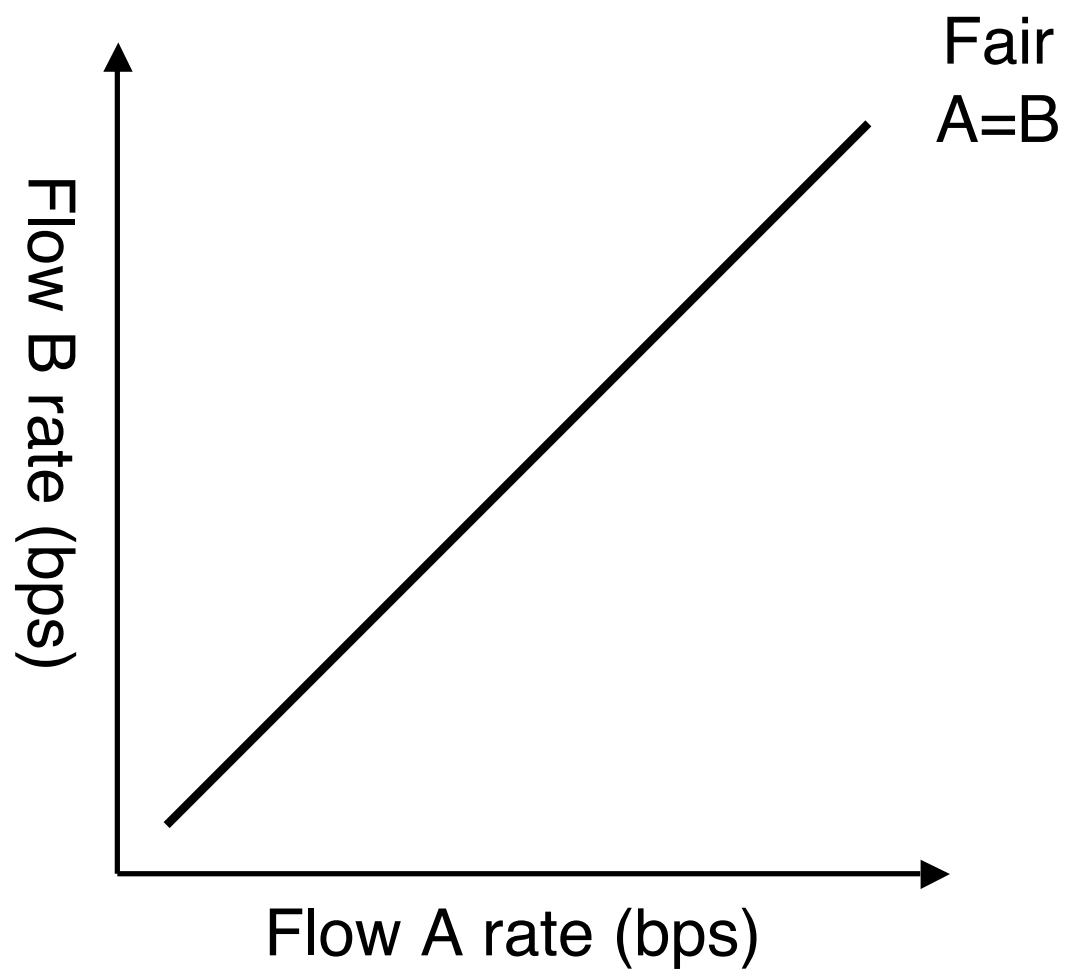


- Figure thanks to Brad Karp

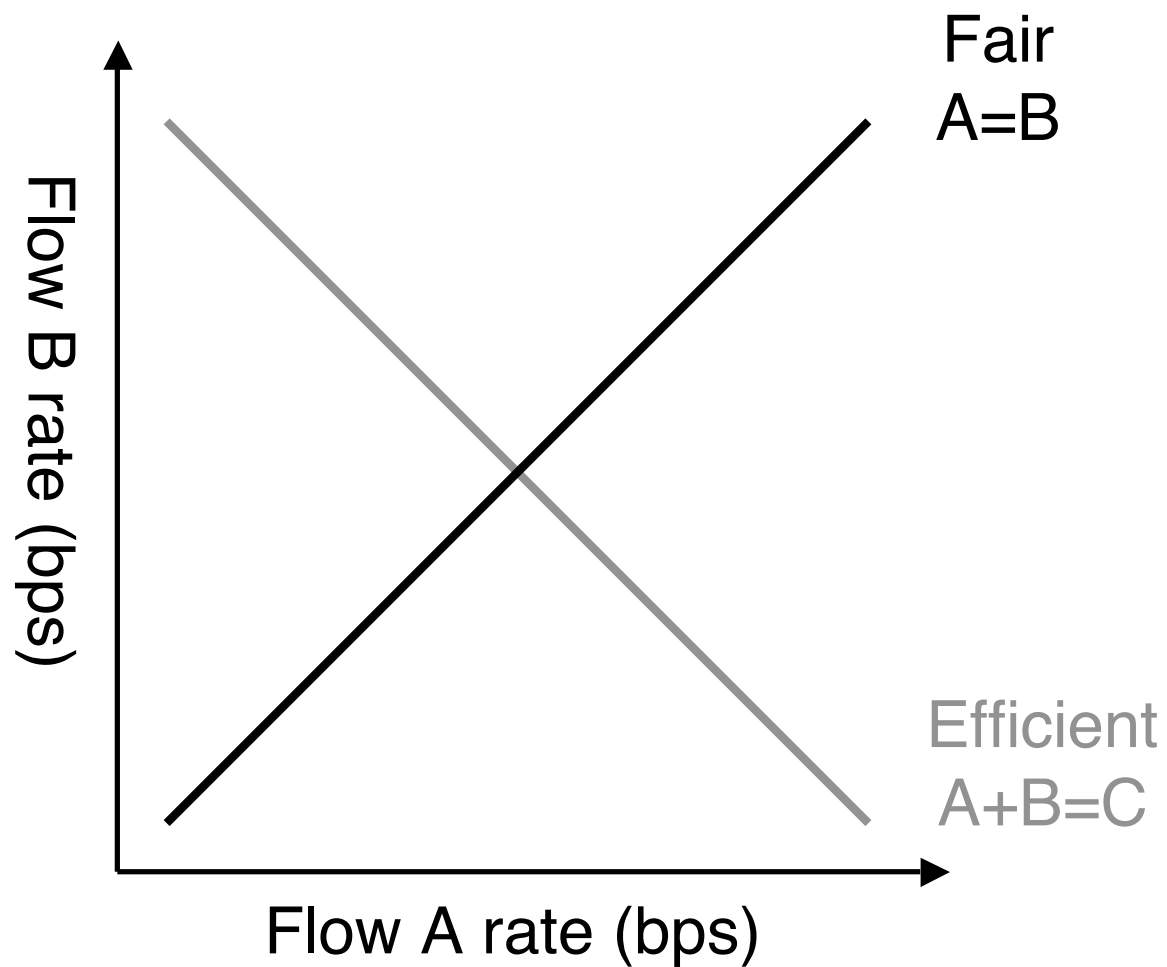
# Chiu Jain Phase Plots



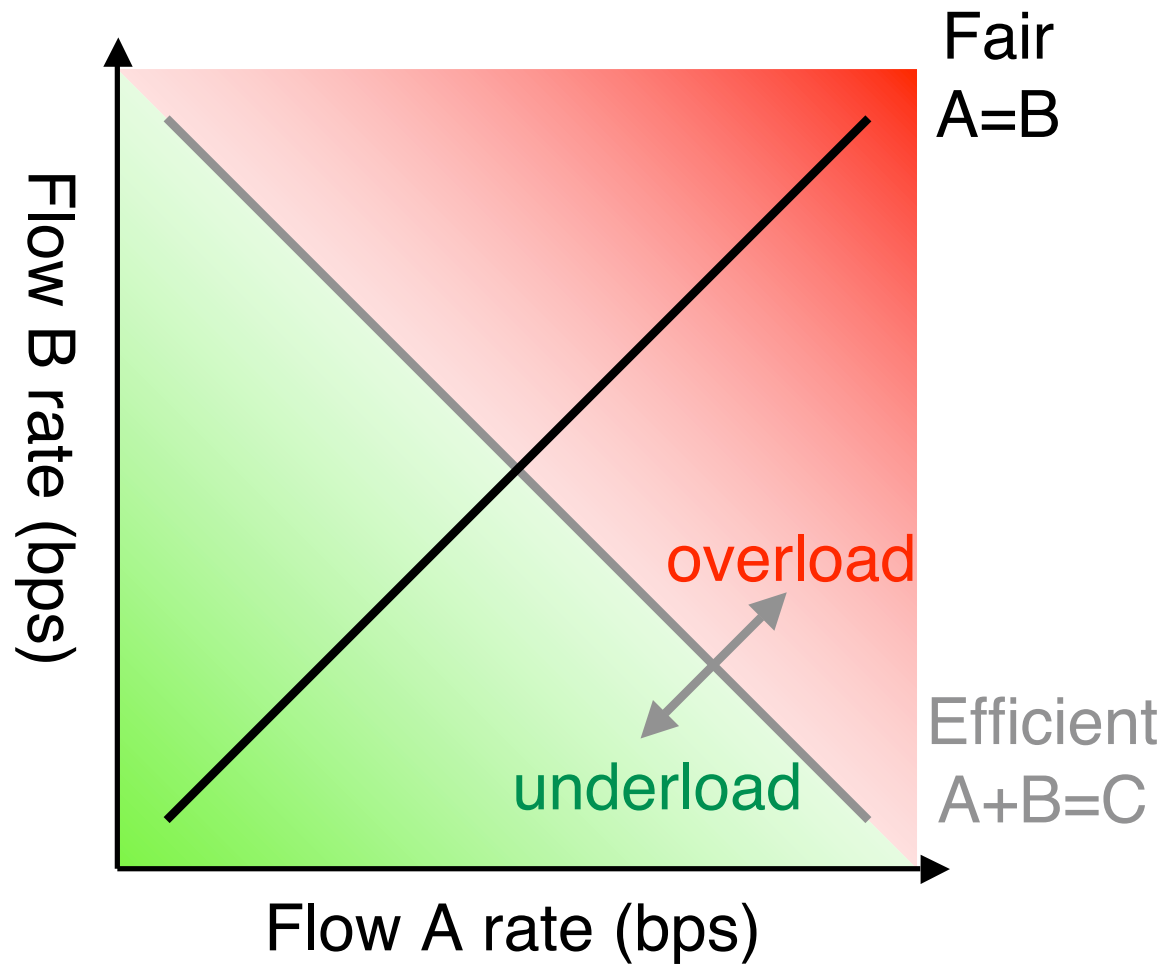
# Chiu Jain Phase Plots



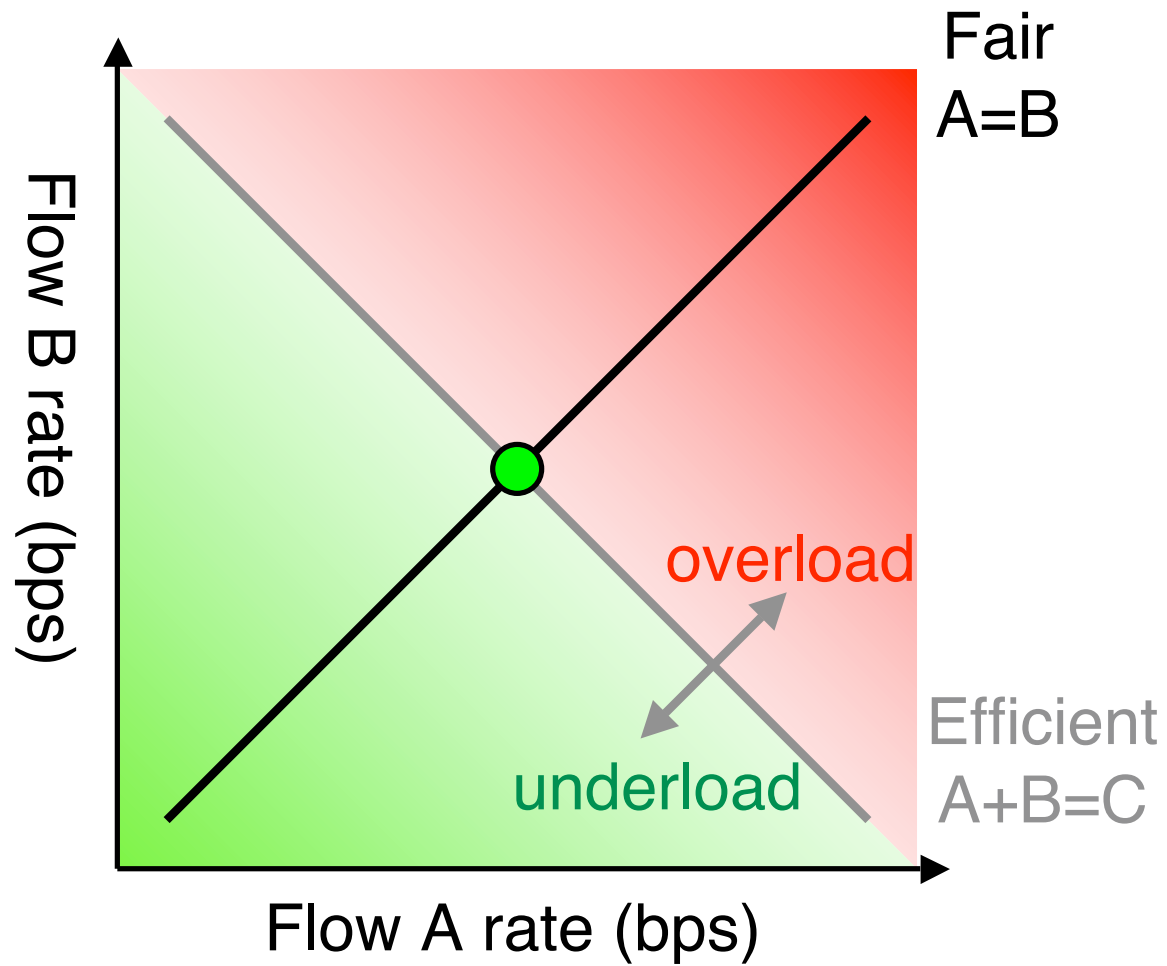
# Chiu Jain Phase Plots



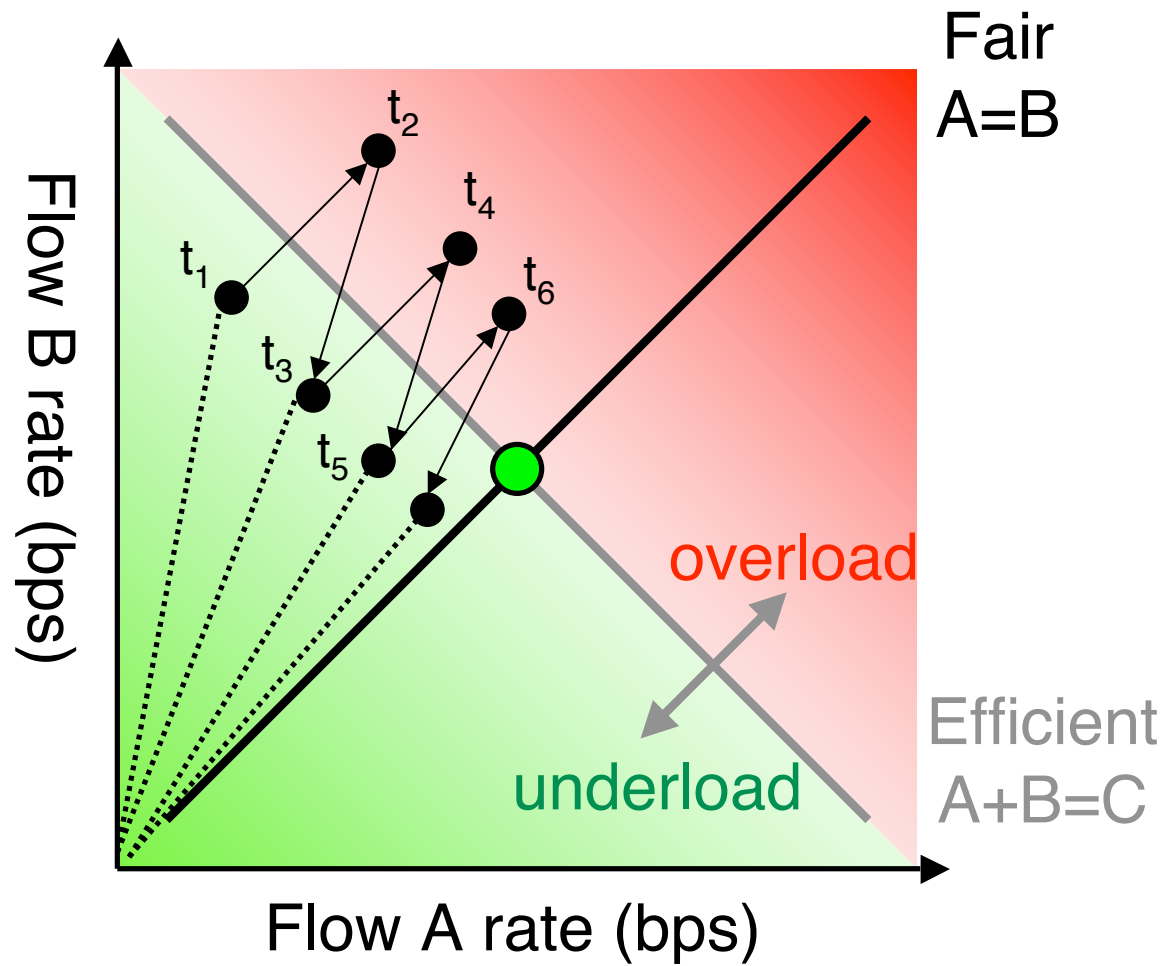
# Chiu Jain Phase Plots



# Chiu Jain Phase Plots

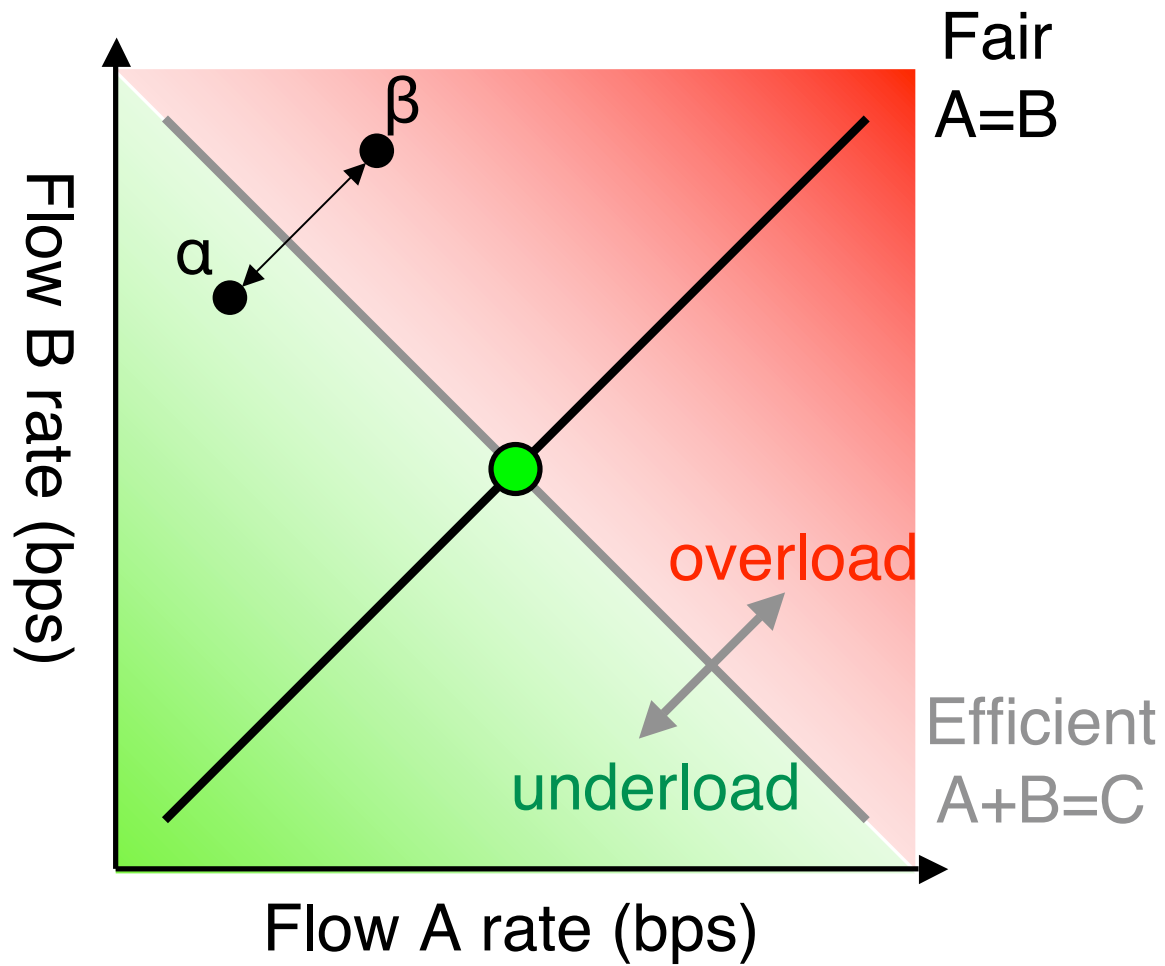


# Chiu Jain Phase Plots





# Chiu Jain Phase Plots



## Three questions, revisited

- Goal: maintain TCP goodput at equilibrium
- When does TCP retransmit packets?
- When does TCP transmit packets?
- When does TCP ack packets?

## **Sending acknowledgements**

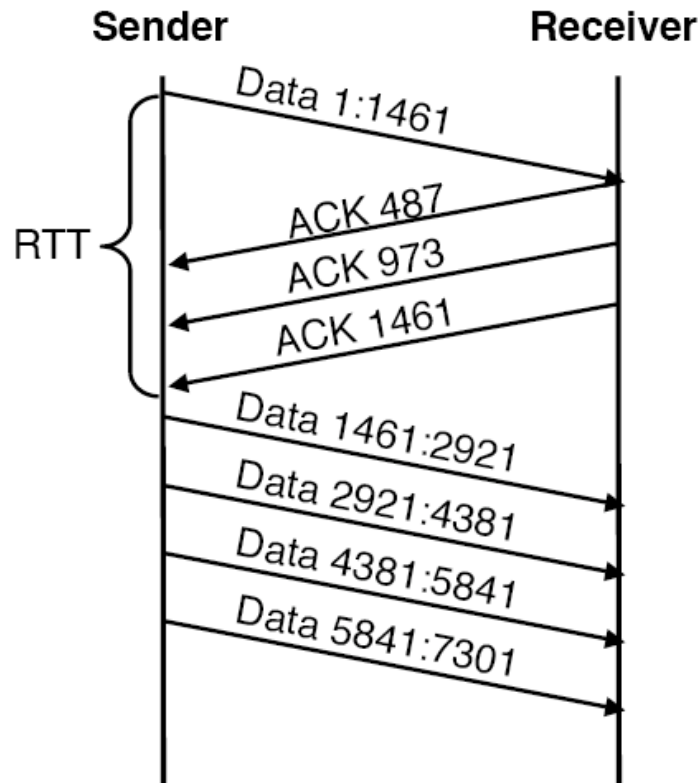
- **An ACK must be sent for every other segment (RFC 2581)**
- **An ACK must be sent within 500ms (RFC 2581)**
- **Send duplicate acks aggressively**

## Acknowledgement Issues

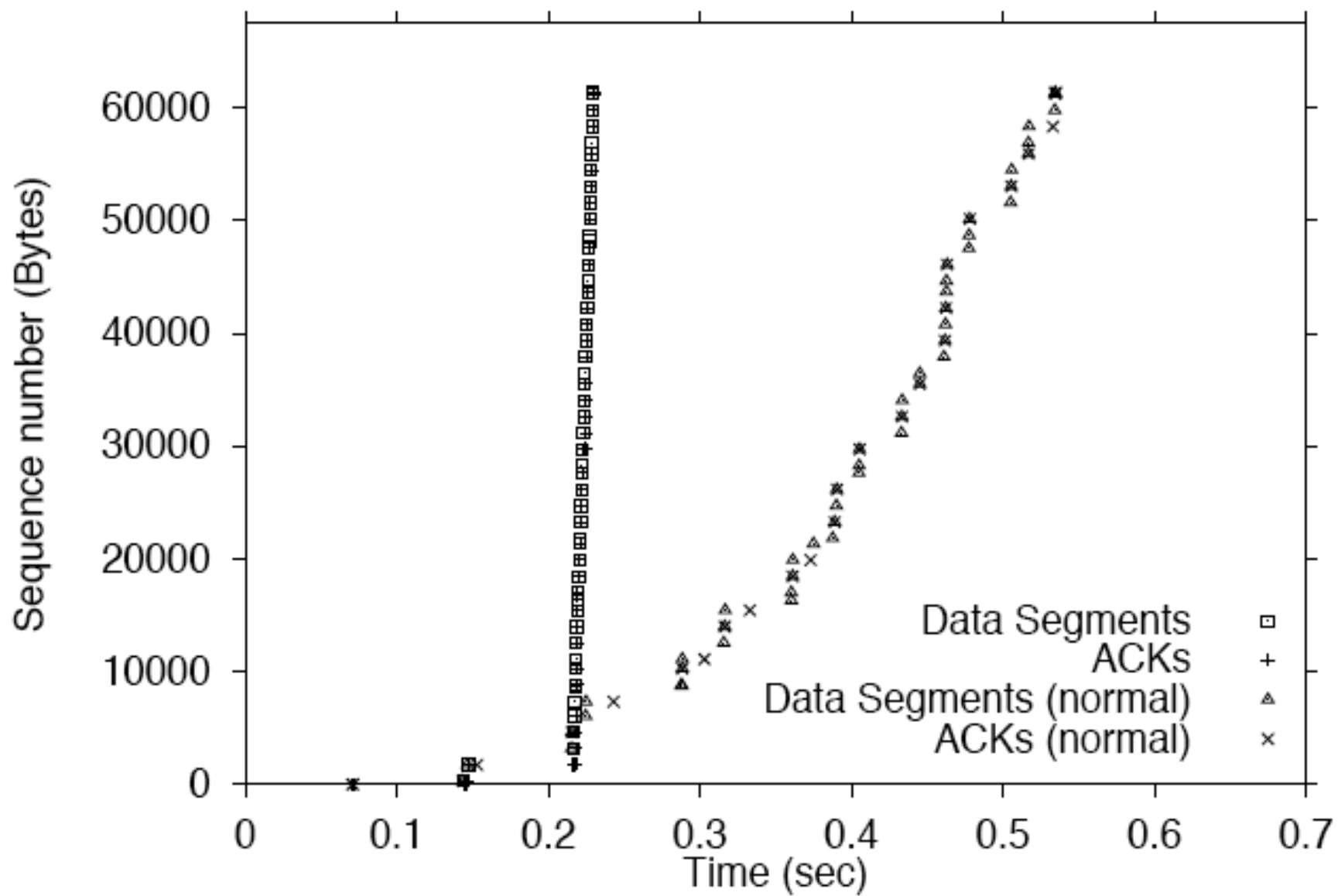
- Savage et al., CCR 1999
- ACKs are in terms of bytes
- Congestion control is in terms of segments
- What if you send multiple ACKs per segment?

# Multiplicative increase

- Receiver can force multiplicative increase, for  $M \leq$  segment length
- Can lead to 4GB cwnd in 4 RTTs!



**TCP Daytona!**



# TCP Daytona Lessons

- TCP implementations now symmetrically use bytes
- Protocol specification is difficult!
- Very subtle and simple design decisions can lead to undesired behavior
- IETF requires at least two interoperating implementations before moving to standards track
- Protocols evolve



# Future of TCP

- **BitTorrent can lead to other apps performing poorly**
  - BitTorrent (and other P2P applications) can open 10-100 connections
  - TCP provides per-flow, not per-application, fairness
- **Low Extra Delay Background Transport (ledbat)**
  - "LEDBAT is a transport-area WG that will focus on broadly applicable techniques that allow large amounts of data to be consistently transmitted without substantially affecting the delays experienced by other users and applications."

## Future of TCP, Continued

- **High speed links/high bandwidth-delay product**
  - Single packet loss cuts cwnd in half
  - Can take a long time to grow
  - Fast TCP, High-Speed TCP
- **Wireless**
  - Wireless exhibits many losses due to poor links, limits throughput
  - High-throughput TCP in wireless meshes does not yet exist
  - More in lecture 14