

CSE 123b

Communications Software

Spring 2002

Lecture 5: Congestion Control

Stefan Savage

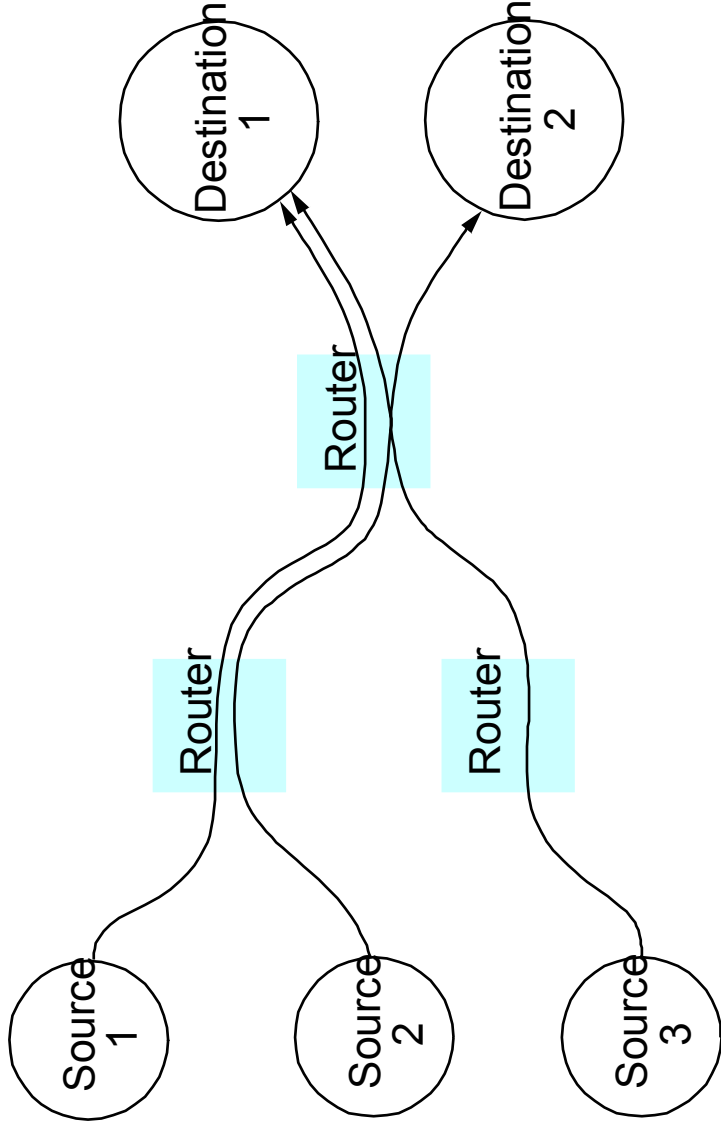
Last Class

- We talked about two things:
- Connections
 - ◆ Connection-oriented (TCP) vs connectionless (UDP) protocols
 - ◆ Connection establishment & termination
- Flow control
 - ◆ How to ensure that a host doesn't send more data than the receiver can buffer

Today's question

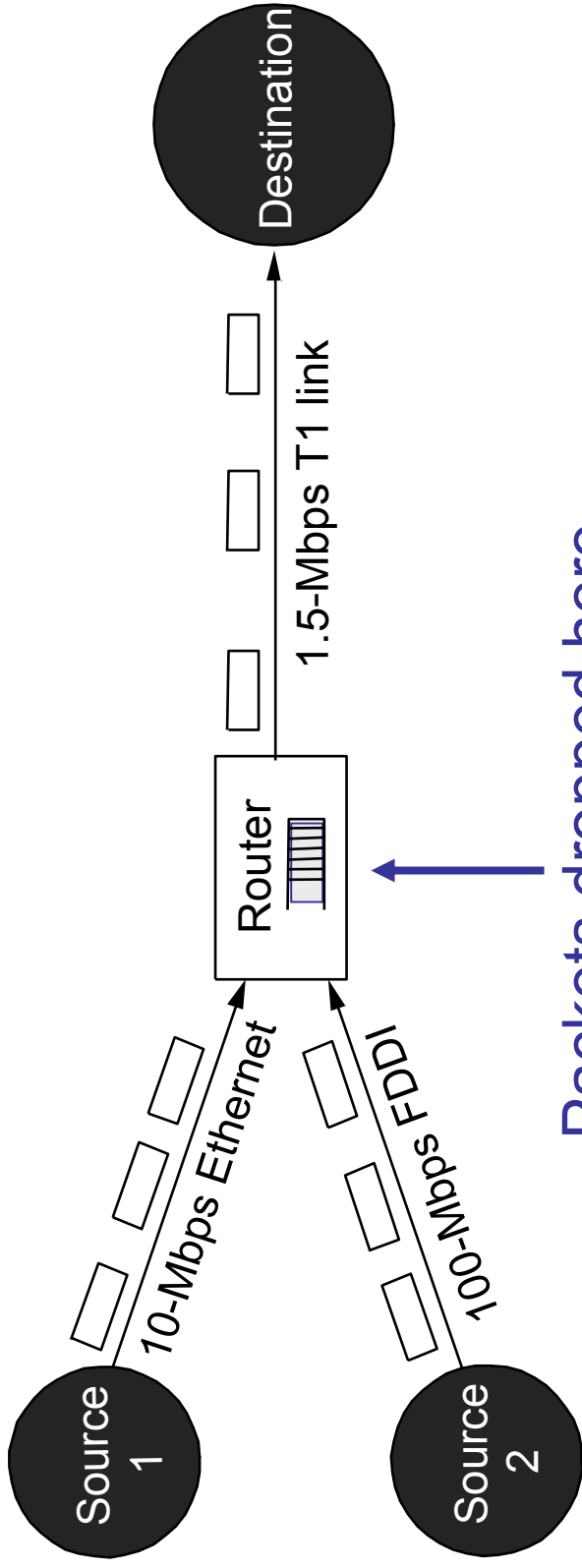
- How fast should the sender transmit data?
 - ♦ Not too slow, Not too fast, Just right...
- Should not be faster than the receiver can process
 - ♦ Flow control (last class)
- Should not be faster than the sender's "share"
 - ♦ **Bandwidth allocation**
- Should not be faster than the network can process
 - ♦ **Congestion control**
- Congestion control & bandwidth allocation are separate ideas, but frequently combined

What is bandwidth allocation?



- How much bandwidth should each flow receive when they compete for resources?

What is congestion?



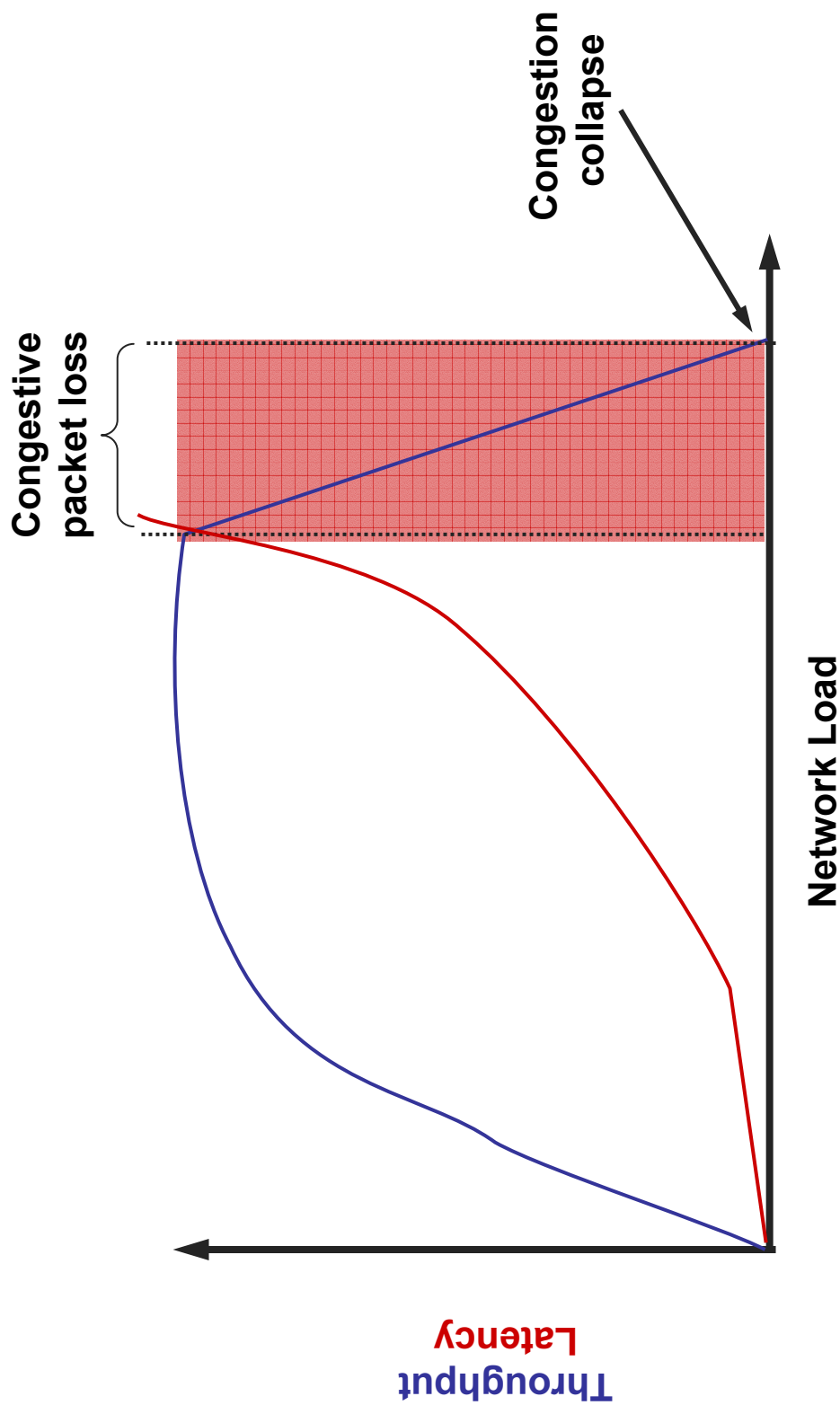
Packets dropped here

- Buffer intended to absorb bursts when input rate $>$ output
- But if sending rate is persistently $>$ drain rate, queue builds
- Dropped packets represent wasted work; $\text{goodput} < \text{throughput}$

Quick review: How queuing works

- Queues absorb **short-term** traffic bursts
- Long term overload will cause packets to be dropped
- Two components to a queuing mechanism
 - ◆ **Scheduling**: which packets are sent from queue?
 - ◆ **Buffer management**: what happens when queue is full?
- Most of the Internet is FIFO/Drop Tail
 - ◆ **First-In-First-Out**: Packets leave in same order they arrive
 - ◆ **Drop tail**: When queue is full, newly arriving packets dropped
 - ◆ Simple to implement at high speeds
 - ◆ There are other policies and we'll talk about some of them another time...

Impact of load on FIFO/Drop-Tail Queues



Congestion collapse

- Rough definition: “When an increase in network load produces a decrease in useful work”
- Why does it happen?
 - ♦ Sender sends faster than **bottleneck link** speed
 - » Whats a bottleneck link?
 - ♦ Packets queue until dropped
 - ♦ In response to packets being dropped, sender retransmits
 - ♦ Repeat in steady state
 - ♦ Everyone does the same thing...

What can be done?

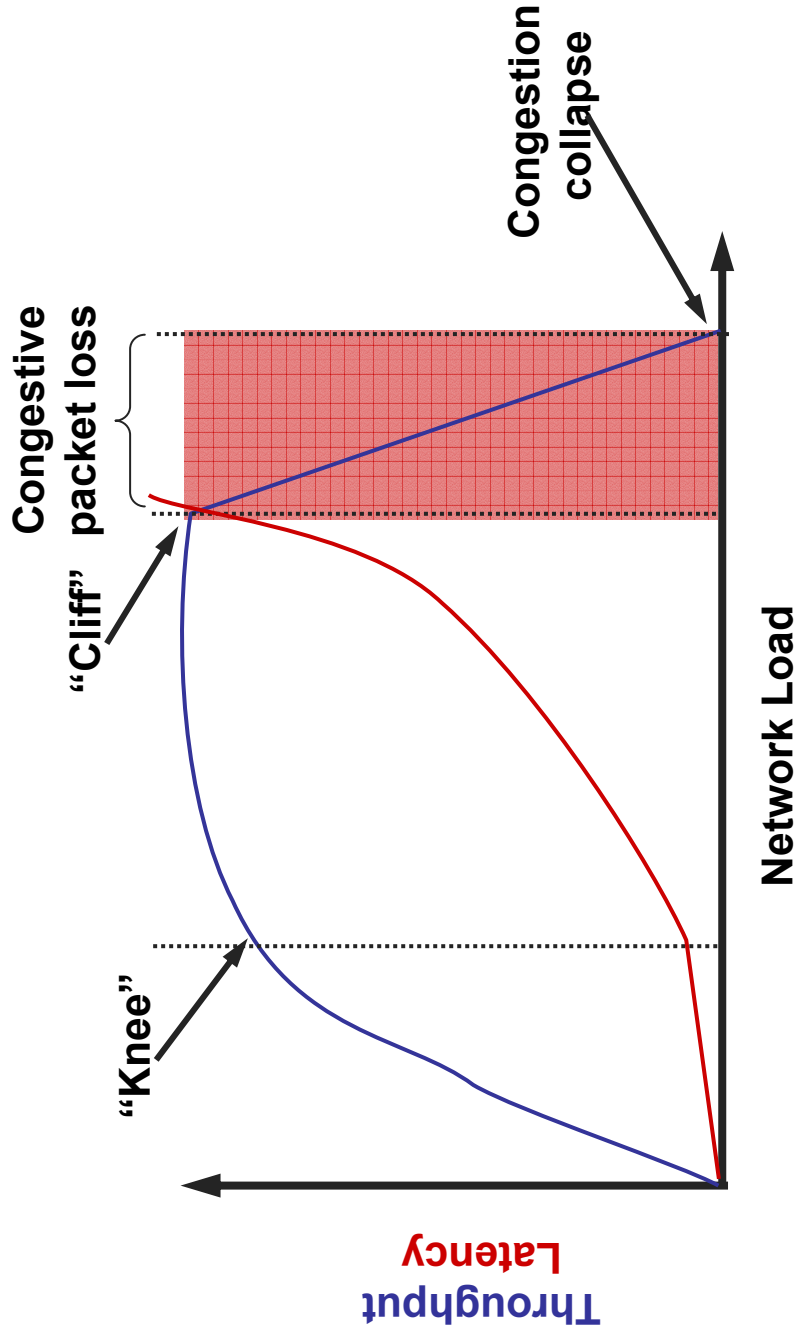
- **Increase network resources**
 - ◆ More buffers for queuing
 - ◆ Increase link speed
 - ◆ Pros/Cons of these approaches?
- **Reduce network load**
 - ◆ Send data more slowly
 - ◆ How much more slowly?
 - ◆ How much bandwidth does each host get?

High-level design choices

- Open loop
 - ♦ Explicitly reserve bandwidth in the network in advance of sending
- Closed loop
 - ♦ Respond to feedback and adjust bandwidth allocation
- Network-based
 - ♦ Network implements and enforces bandwidth allocation
- Host-based
 - ♦ Hosts are responsible for controlling their sending rate to be < their bandwidth share
- What is typically used on the Internet? Why?

Proactive vs reactive approaches

- Congestion avoidance: try to stay to the left of the knee
- Congestion control: try to stay to the left of the cliff



Key questions

- How to detect congestion?
- How to limit sending data rate?
- How fast to send?
- How to achieve stability?

How to detect congestion?

- Explicit congestion signaling
 - ♦ Source Quench: ICMP message from router to sender
 - ♦ DECBit / Explicit Congestion Notification (ECN):
 - » Router marks packet based on queue occupancy
 - » Receiver tells sender if queue is getting too full
 - ♦ Hop-by-hop backpressure
- Implicit congestion signaling
 - ♦ **Packet loss**
 - » Assume congestion is primary source of packet loss
 - » Lost packets (timeout, NAK) indicate congestion
 - ♦ Packet delay
 - » Round-trip time increases as packets queue
 - » Packet inter-arrival time is a function of bottleneck link
 - » Pros/Cons?

How to limit the sending rate?

- **Window-based (TCP)**
 - ◆ Artificially constrain number of outstanding packets allowed in network
 - ◆ Increase window to send faster; decrease to send slower
 - ◆ Pro: Cheap to implement; good failure properties
 - ◆ Con: creates bursty traffic
- **Rate-based (Many streaming media protocols)**
 - ◆ Two parameters (period, packets)
 - ◆ Allow sending of x packets in period y
 - ◆ Pro: smooth traffic
 - ◆ Con: per-connection timers; what if receiver fails?

How fast to send?

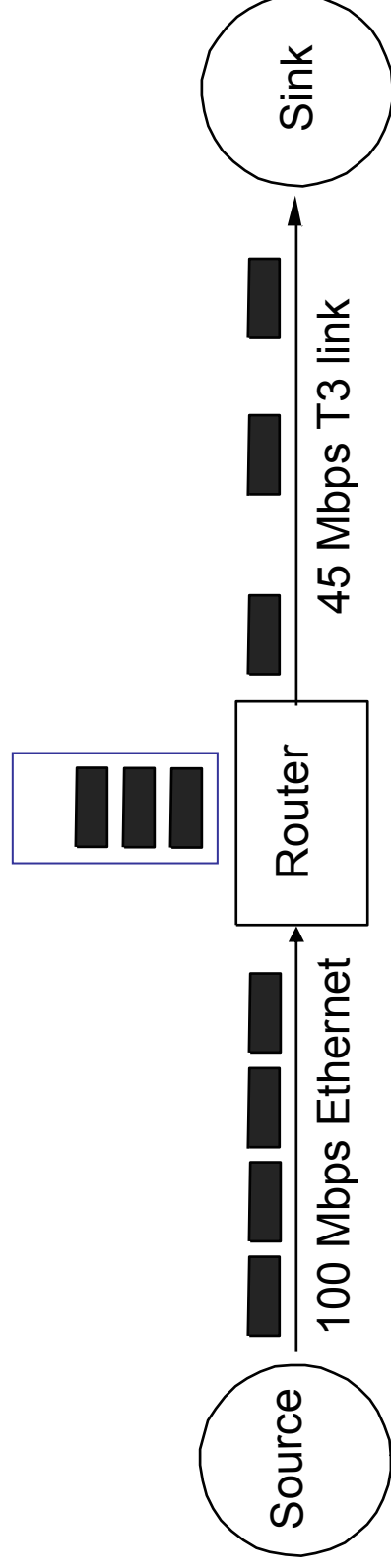
- Ideally: Keep equilibrium at “knee” of power curve
 - ♦ Find “knee” somehow
 - ♦ Keep number of packets “in flight” the same
 - ♦ Don’t send a new packet into the network until you know one has left (i.e. by receiving an ACK)
 - ♦ What if you guess wrong, or if bandwidth availability changes?
- Compromise: adaptive approximation
 - ♦ If congestion signaled, reduce sending rate by x
 - ♦ If data delivered successfully, increase sending rate by y
 - ♦ How to relate x and y ? Most choices don’t converge...

How does TCP do it?

Jacobson&Karels88

- Seminal paper in computer networking
 - ♦ 5th most cited paper in all computer science
- Context: 1986 brings huge congestion collapse
 - ♦ TCP just uses fixed-sized sliding window
 - ♦ LBL <-> Berkeley link throughput decreases by 1000x
 - ♦ Motivation for paper: Why? and how to fix it?
- Key algorithms:
 - ♦ Congestion avoidance (misnamed)
 - ♦ Slow start
 - ♦ Fast retransmit & fast recovery

TCP Probes the Network



- Increase sending rate to probe the network – and 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

Window-based congestion control

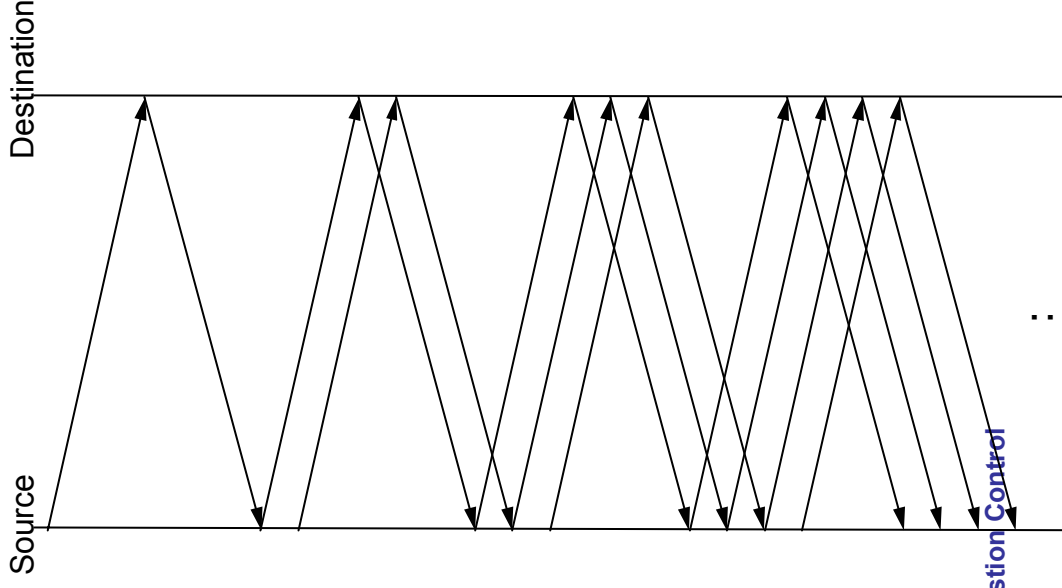
- Window-based
 - ♦ Makes sense make congestion control and flow control using same rate-limiting mechanism
 - ♦ *rwin*: advertised flow control window from receiver
 - ♦ *cwnd*: **congestion control window**
 - » Estimate of network limit on # of outstanding packets
 - ♦ Sender can only send $\text{MIN}(rwin, cwnd)$ at any time
- Idea: decrease *cwnd* when congestion is encountered; increase *cwnd* otherwise
- Question: how much to adjust?

Congestion avoidance algorithm

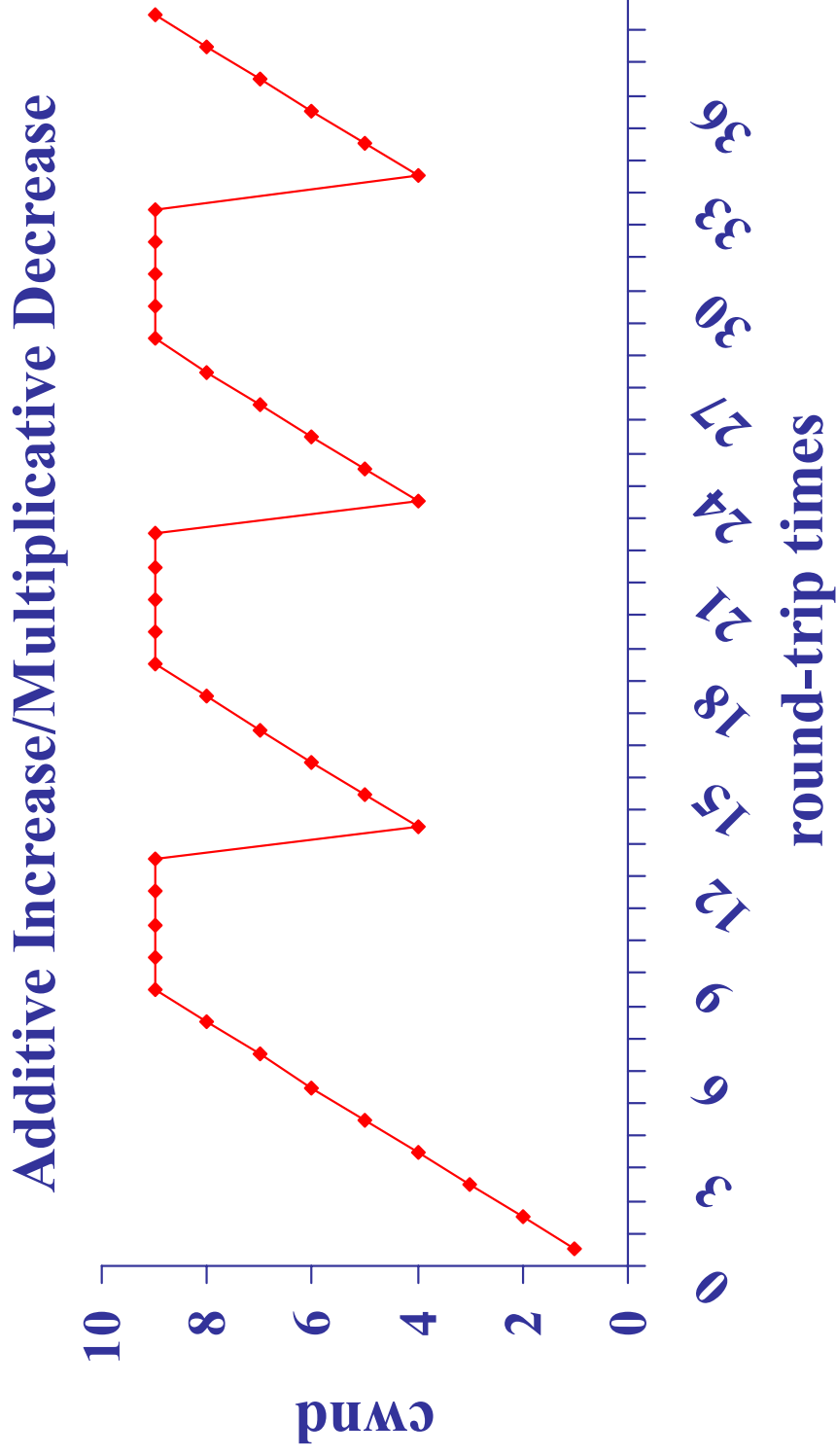
- Goal: Adapt to changes in available bandwidth
- Additive Increase, Multiplicative Decrease (AIMD)
 - ♦ Increase sending rate by a constant (e.g. by 1500 bytes)
 - ♦ Decrease sending rate by a linear factor (e.g. divide by 2)
- Rough intuition for why this works (from JK88)
 - ♦ Let L_i be queue length at time i
 - ♦ In steady state: $L_i = N$, where N is a constant
 - ♦ During congestion: $L_i = N + y L_{i-1}$, where $y > 0$
 - ♦ If y is large (close to 1), queue size increases exponentially
 - » Must reduce sending rate exponentially as well (multiplicative decrease)

AIMD (Additive Increase/Multiplicative Decrease)

- Increase slowly while we believe there is bandwidth
 - ◆ Additive increase per RTT
 - ◆ $Cwnd += 1 \text{ packet} / \text{RTT}$
- Decrease quickly when there is loss (went too far!)
 - ◆ Multiplicative decrease
 - ◆ $Cwnd /= 2$



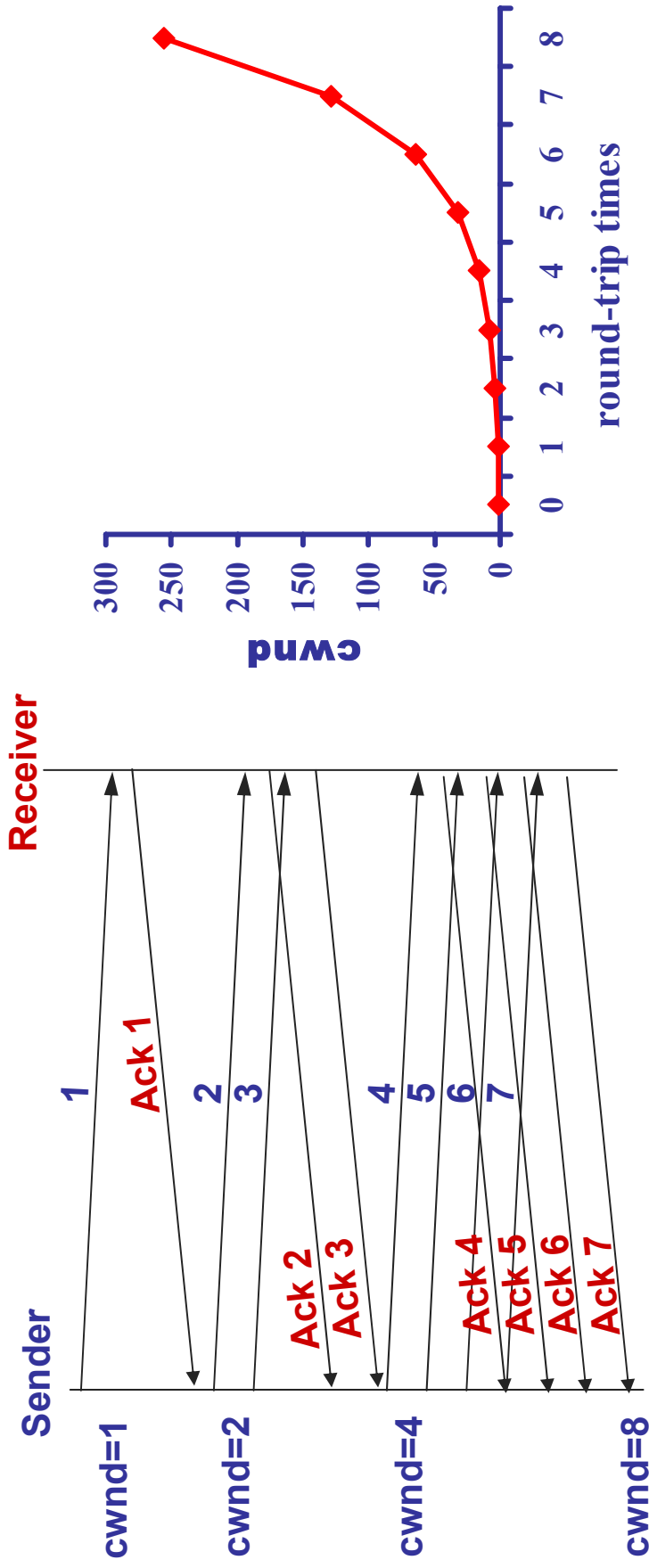
Congestion avoidance growth ("sawtooth" function)



Slow start

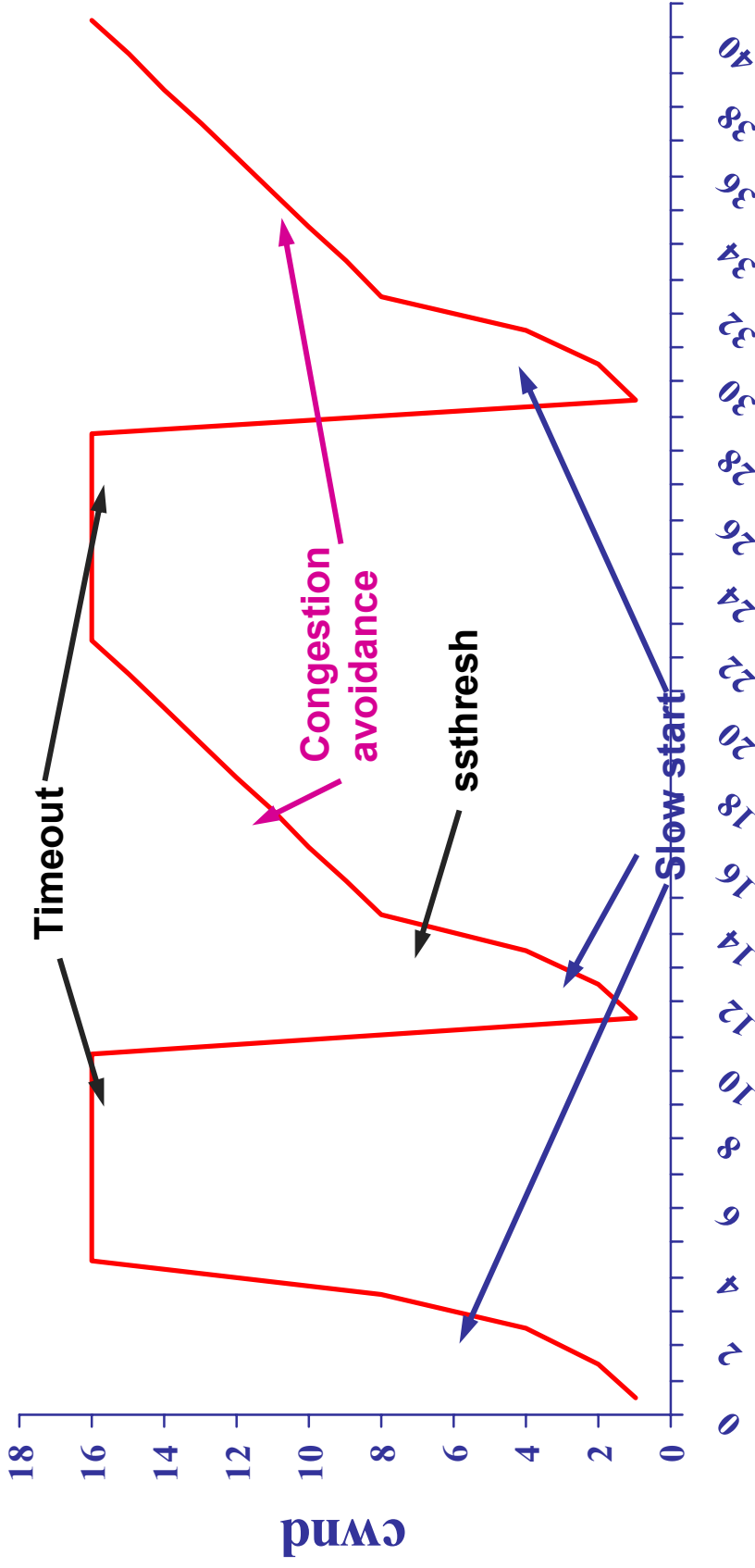
- Goal: find the equilibrium sending rate quickly
- Quickly increase sending rate until congestion detected
- Algorithm:
 - ♦ On new connection, or after timeout, set $cwnd=1$
 - ♦ For each segment acknowledged, increment $cwnd$ by 1
 - ♦ If timeout then divide $cwnd$ by 2, and set $ssthresh = cwnd$
 - ♦ If $cwnd \geq ssthresh$ then exit slow start
- Why called slow? Its exponential after all...

Slow start growth example



Putting it together

Slow Start + Congestion Avoidance

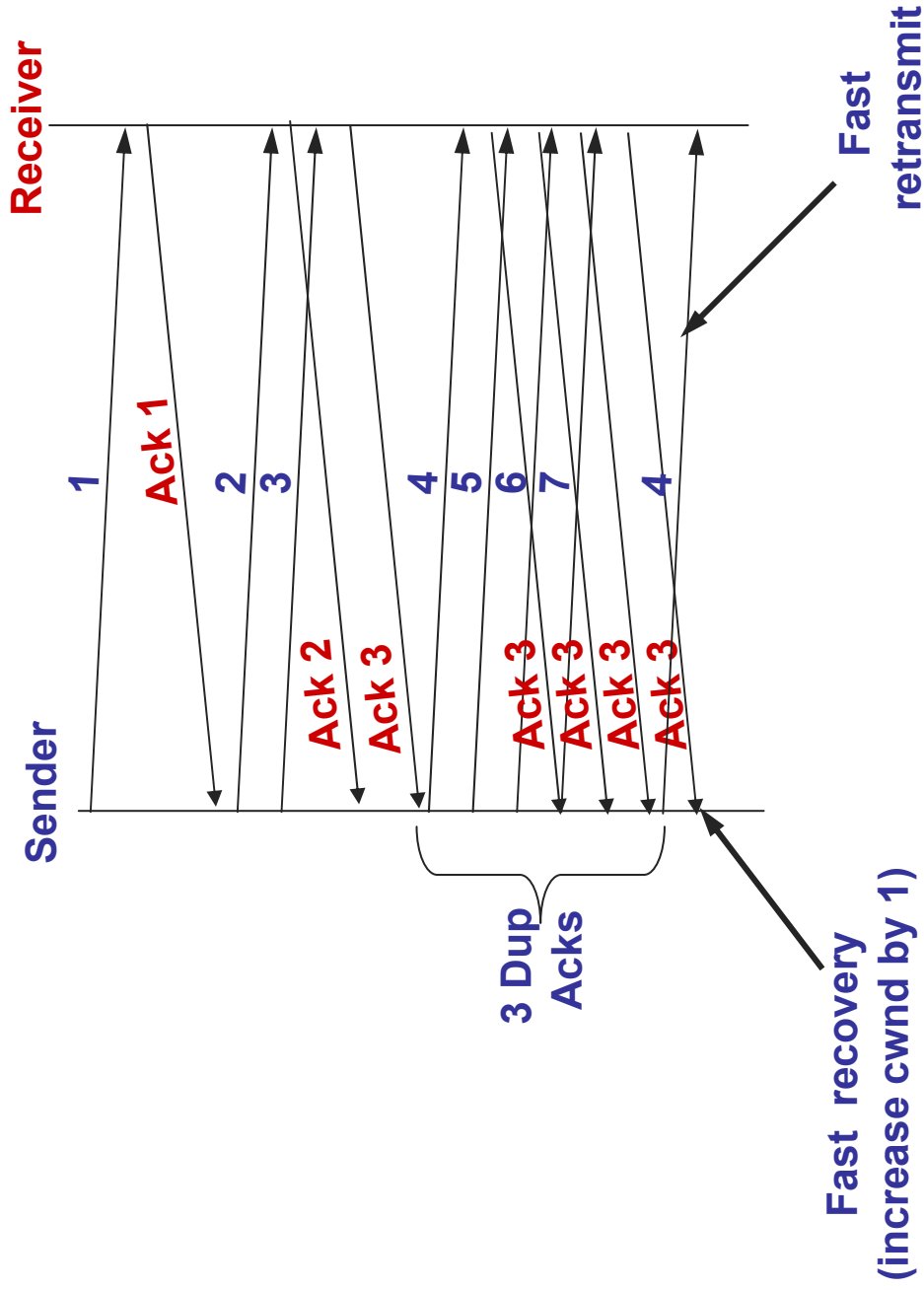


round-trip times

Fast retransmit & recovery

- **Review: Fast retransmit**
 - ♦ Timeouts are slow (1 second is fastest timeout on most TCPs)
 - ♦ When packet is lost, receiver still ACKs last in-order packet
 - ♦ Use 3 duplicate ACKs to indicate a loss
 - ♦ End result: can detect losses more quickly
- **Fast recovery**
 - ♦ If there are still ACKs coming in, then no need for slow start
 - ♦ Divide *cwnd* by 2 after fast retransmit
 - ♦ Increment *cwnd* by 1/*cwnd* for each additional duplicate ACK
 - ♦ End result: Can achieve AIMD when there are single packet losses. Only slow start the first time

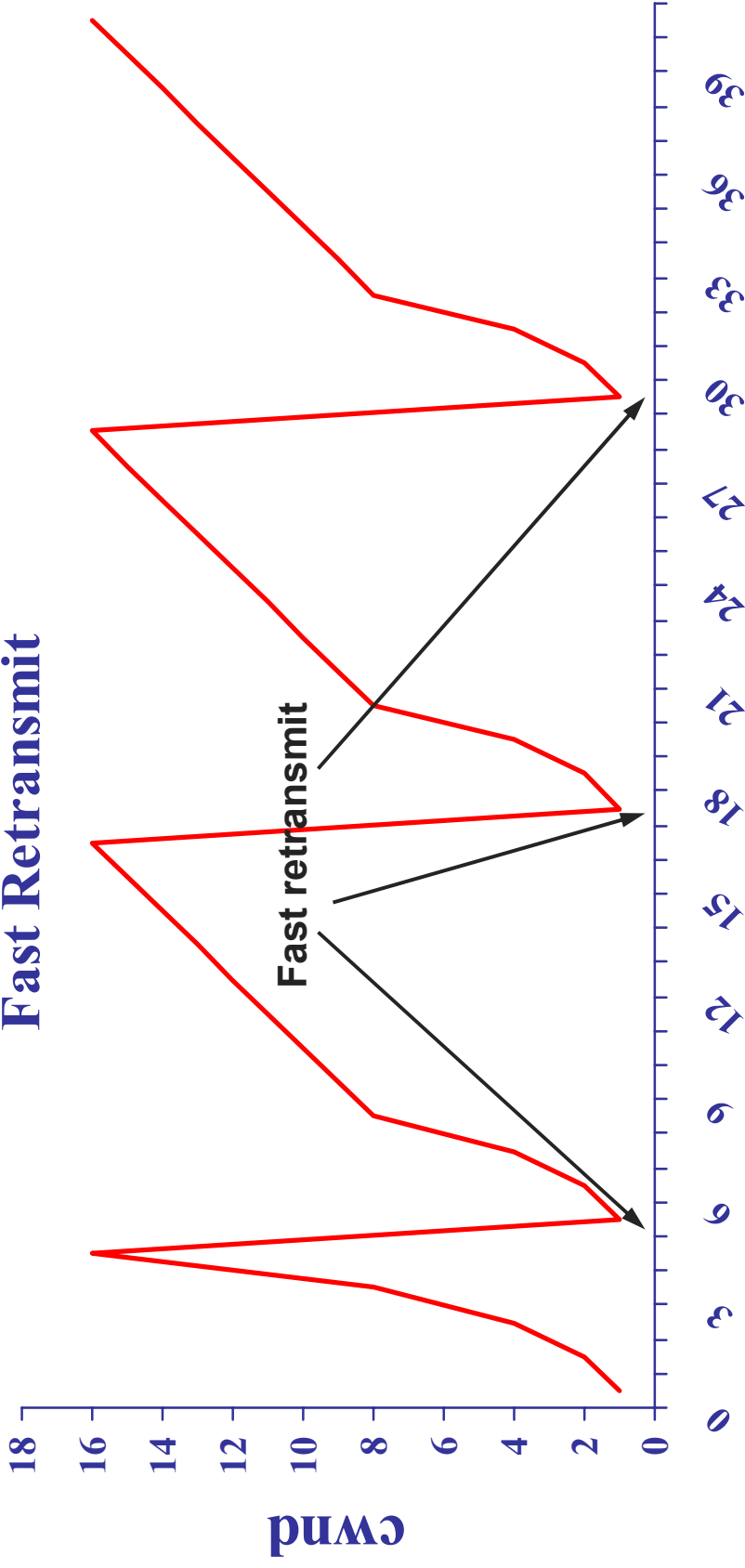
Fast retransmit&recovery



Fast retransmit in action

Slow Start + Congestion Avoidance +

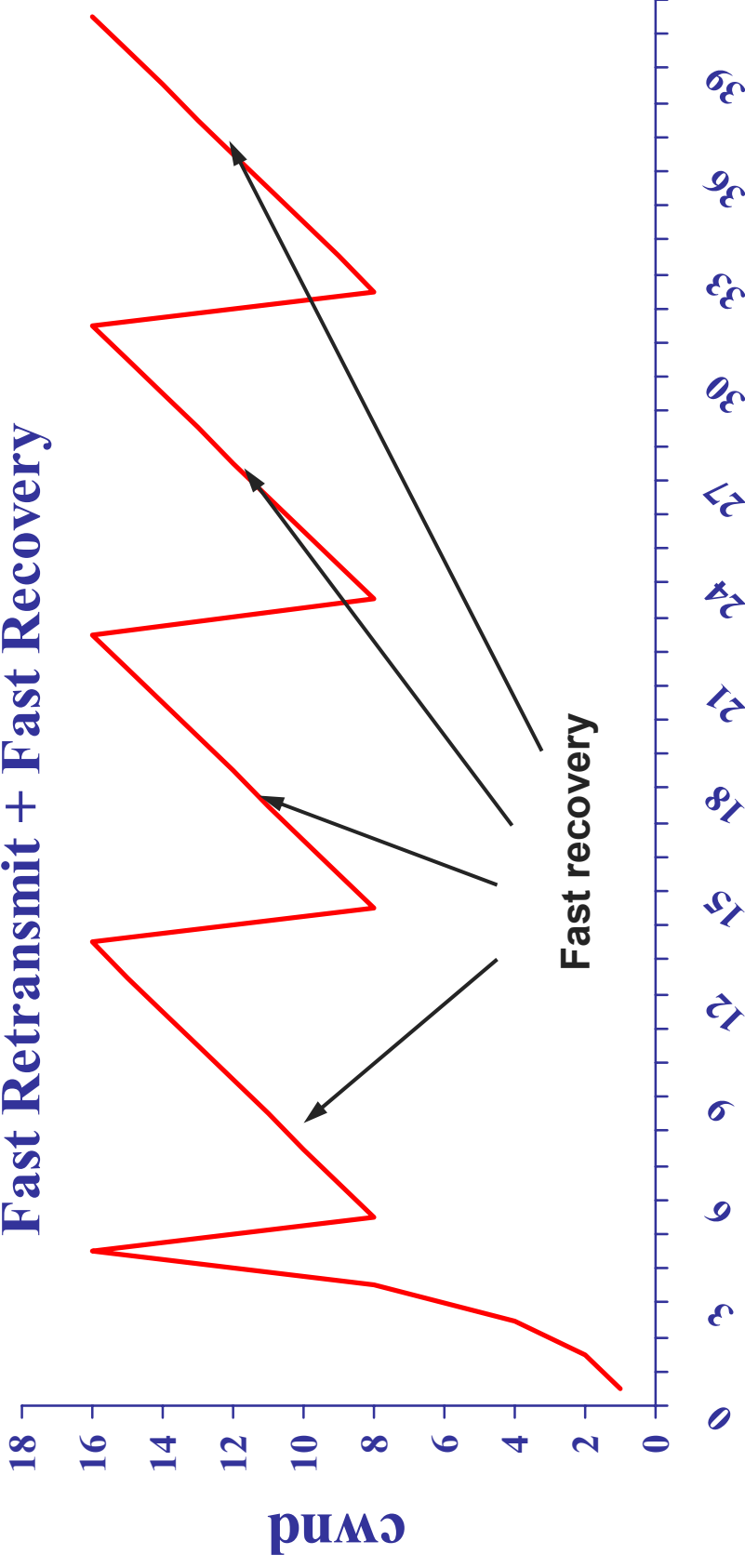
Fast Retransmit



round-trip times

Fast recovery in action

Slow Start + Congestion Avoidance +
Fast Retransmit + Fast Recovery



round-trip times

Delayed ACKs

- In request/response programs, want to combine an ACK to a request with a response in same packet
 - ♦ Wait 200ms before ACKing
 - ♦ Must ACK every other packet (or packet burst)
 - ♦ Impact on slow start?
- Must not delay duplicate ACKs
 - ♦ Why? What is the interaction with the congestion control algorithms?

Discussion:

Short Connections

- How do short connections and Slow-Start interact?
 - ♦ What happens when there is a drop in Slow-Start?
 - ♦ What happens when the SYN is dropped?
- Bottom line: Which packet gets dropped matters a lot
 - ♦ Syn
 - ♦ Slow-Start
 - ♦ Congestion avoidance
- Do you think most flows are short or long?
 - ♦ What's the current most popular application?
 - ♦ What were the most popular applications when Slow-Start was developed?

Stuff to think about

- TCP is designed around the premise of cooperation
 - ♦ What happens to TCP if it competes with a UDP flow?
 - ♦ What if divide cwnd by 3 instead of 2 after a loss?
 - ♦ What happens if receiver lies about receiving packets?
- There are a number of assumptions
 - ♦ Losses mean congestion, re-ordering is rare, etc...
- There are a bunch of magic numbers
 - ♦ Decrease by 2x, increase by $1/\text{cwnd}$, 3 duplicate acks, $g=1/8$, initial timeout = 3 seconds, etc
- But it works quite well!

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
- Fast recovery is used to keep network “full” while recovering from a loss

For next time...

- Make sure your computer accounts work
- Read: P&D 4.2