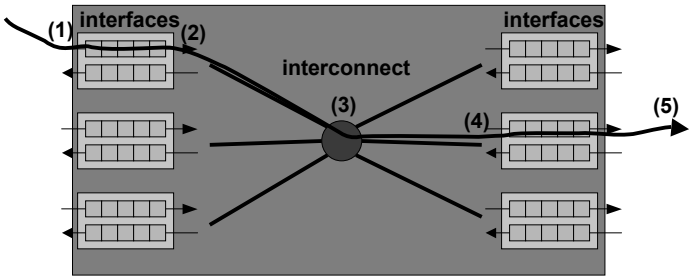
## Congestion Control – End Hosts

CSE 561 Lecture 7, Spring 2002.
David Wetherall

## 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 week)
- Should not be faster than the network can process
  - **Congestion control**

## Quick review: How routers/switches work




1. Packet arrives at input interface and is queued
2. Lookup address in forwarding table and find output interface
3. Switch packet to output interface
4. Packet is queued for transmission
5. Packet sent



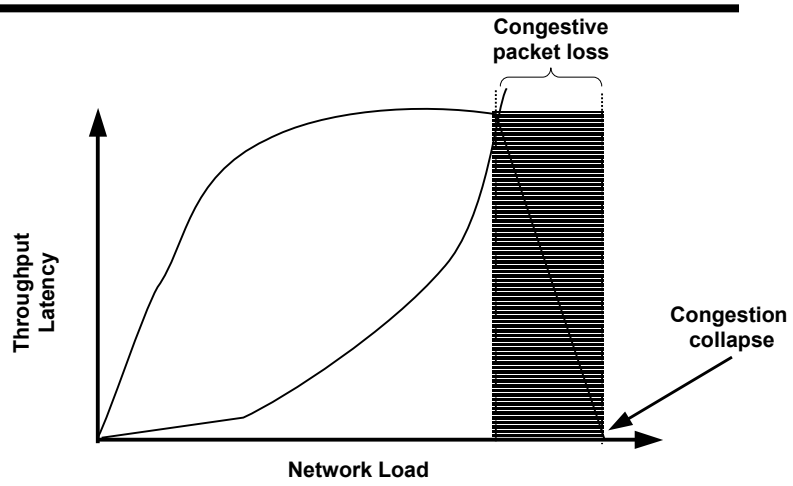
## Quick review: How Queuing works

- 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
  - There are other policies and we'll talk about some of them later…

## 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
  - 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?
  - When to slow down?

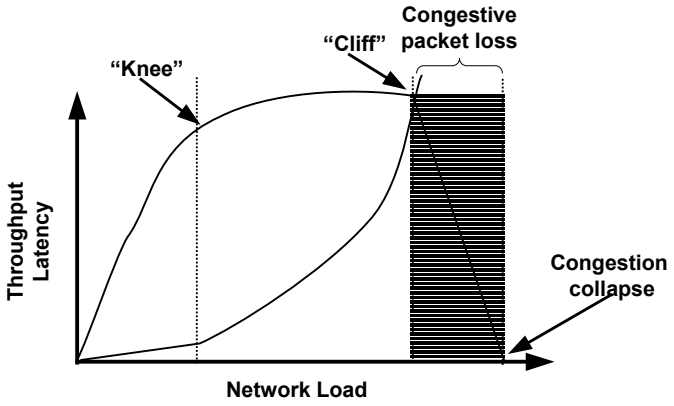


## Congestion management goals

- Efficiency
  - Utilize available bandwidth as much as possible
- Fairness
  - All hosts get equal access to bandwidth
- Distributed implementation
  - Only require state at endpoints
- Convergence
  - For constant load, arrive at single solution for using / sharing bandwidth

## 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?

- Implicit congestion signaling
  - **Packet loss (Today)**
    - 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?
- Explicit congestion signaling
  - Source Quench: ICMP message from router to sender
  - **Explicit Congestion Notification (ECN) (Next time)**
    - Router marks packet based on queue occupancy
    - Receiver tells sender if queue is getting too full
  - Hop-by-hop backpressure



## 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 to achieve stability?

- 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)

## Jacobson&Karels88

- Seminal paper in computer networking
  - 5th most cited paper in all computer science
- Context: 1986 brings huge congestion collapse
  - LBL<->Berkeley link throughput decreases by 1000x
  - Motivation for paper: Why? and how to fix it?
- Key principle: *packet conservation*
  - Once equilibrium is reached, only send new packet once old packet has been received or dropped (constant load)
  - Why is TCP not obeying this principle? Hmmm….



## Ways to violate packet conservation

- Connection never reaches equilibrium
  - **Slow start** (still a big problem)
- Premature sending of new packets
  - **Adaptive timeouts**
- Equilibrium can't be reached because of resource limits along path
  - **Congestion avoidance** (misnamed)

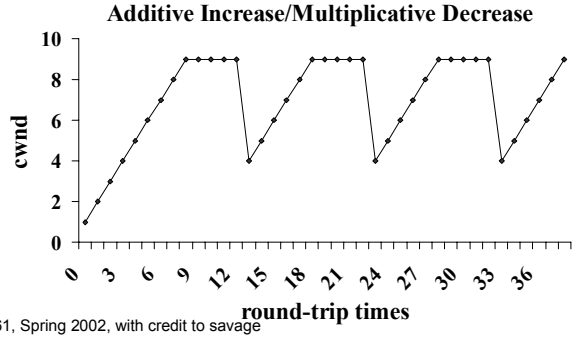
## Some TCP details

- Congestion control, not avoidance
- Implicit congestion detection
  - Packet losses
- 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 MIN(*rwin*,*cwnd*) at any time
- AIMD-based algorithm
  - Increase window by 1/cwnd, decrease by 2



## Congestion avoidance

- If packet is ACKd then increase *cwnd* by MSS*MSS/cwnd (fraction of *cwnd*)
  - Approximates 1/*cwnd* per window
- If timeout then divide *cwnd* by 2



## Adaptive timing

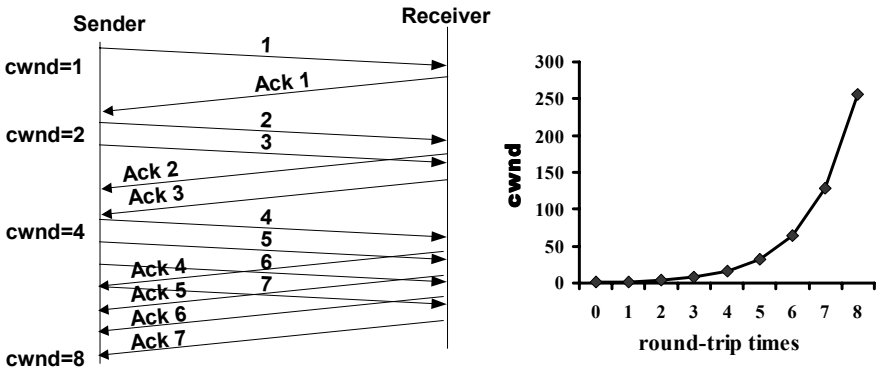
- How long to wait for a packet's acknowledgement?
  - Too short: spurious timeouts and retransmissions
  - Too long: wasted time
- Old TCP
  - Maintain weighted average of RTT samples: ***R***
  - Timeout set to ***R*B***, where ***B*** =2
  - Under high load, ***B*** doesn't reflect variation and spurious retransmissions occur
- Jacobson's contributions
  - Estimate variation, ***B*** based on same samples
  - After loss, increase timeout exponentially (by 2)



## Slow start

- Goal: find the equilibrium sending rate
- 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 >= ssthresh then exit slow start
- Why called slow? Its exponential after all…

## Slow start growth example

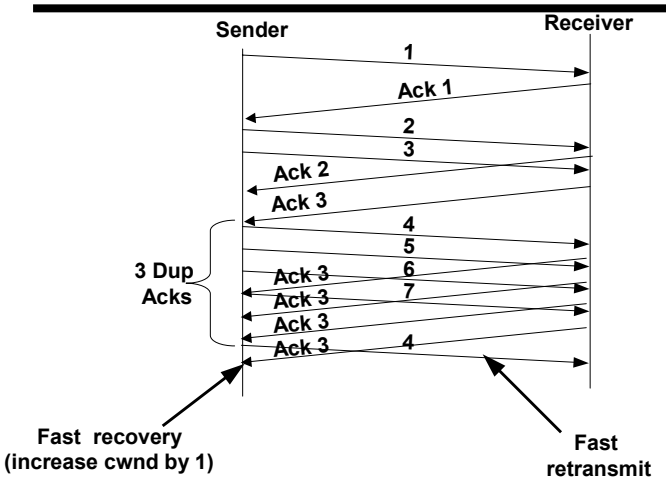





## Fast retransmit & recovery

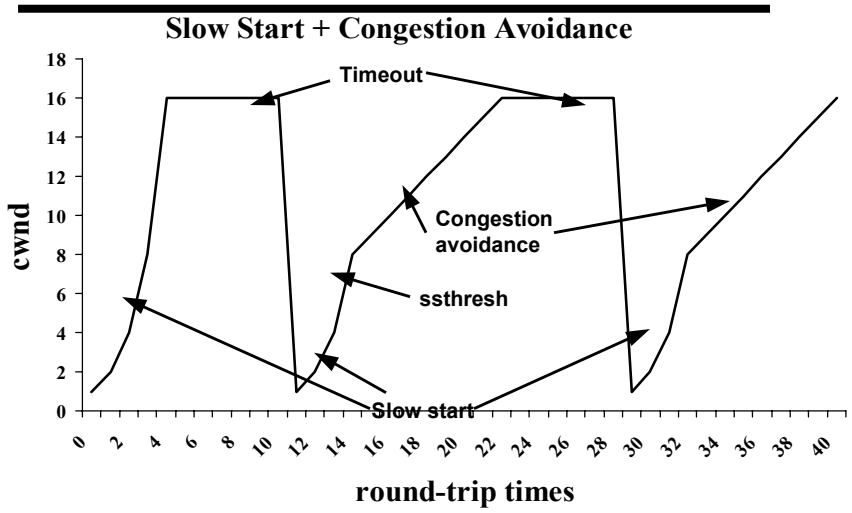
- **Fast retransmit**
  - Timeouts are slow (1 second is fastest timeout on most TCPs)
  - When packet is lost, receiver still ACKs last in-order packet
  - Use 3 duplicate ACKs to indicate a loss
    - Why 3? When wouldn't this work?
- **Fast recovery**
  - If there are still ACKs coming in, then no need for slow start
  - Divide *cwnd* by 2 after fast retransmit
  - Increment cwnd by 1/cwnd for each additional duplicate ACK

## Fast retransmit&recovery

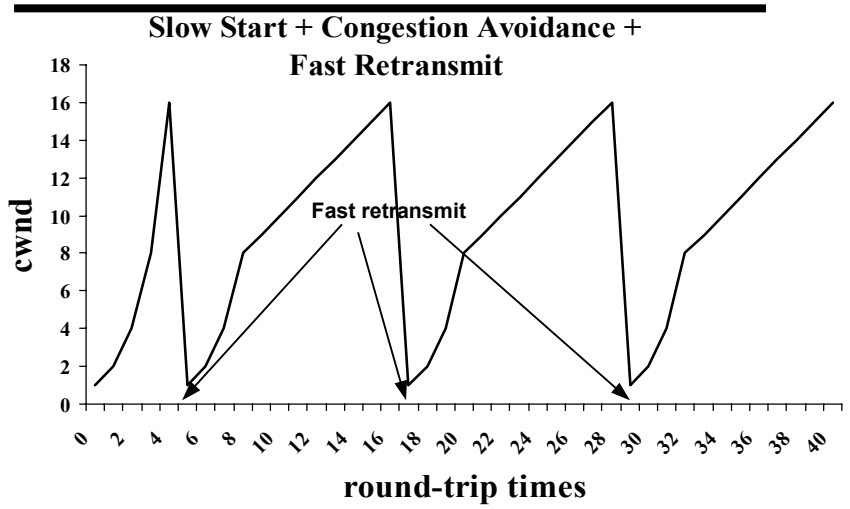





## Putting it together

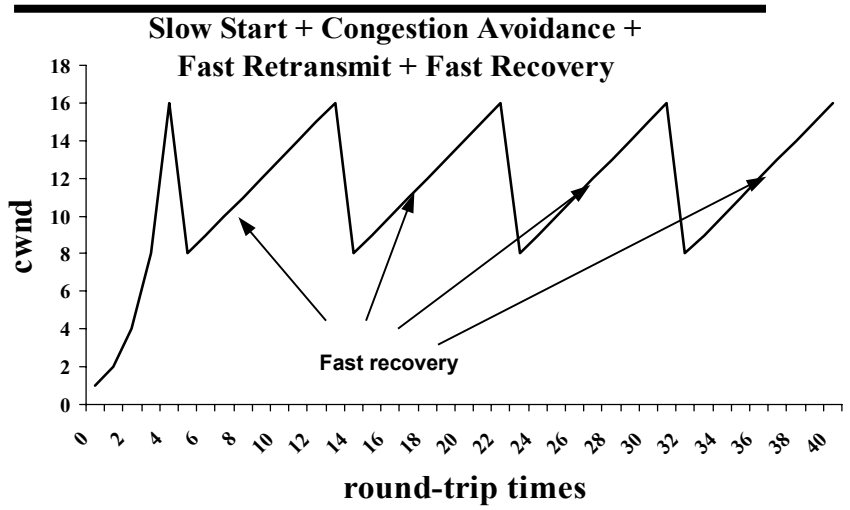


## Fast retransmit in action






## Fast recovery in action



## 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?



## Partial ACKs

- A partial ACK acknowledges only some of the packets outstanding at the start of Fast Recovery
  - Suggests that a second packet may have been lost
- Should you exit Fast Recovery and go into congestion avoidance, or continue in FR?
  - Hoe: Stay in Fast Recovery, retransmit packet after ACK
  - When all packets in window when FR was initiated have been ACKed, go into congestion avoidance

## A TCP Taxonomy

- TCP Tahoe (1988)
  - Slow-Start, Fast Retransmit, Congestion Avoidance
- TCP Reno (1990)
  - Tahoe + Fast Recovery
- TCP New-Reno (1996)
  - Reno + Hoe's partial ACK change that keeps TCP in Fast Recovery
- SACK TCP (1996)
  - Selective acknowledgements
- TCP Vegas (1993)
  - Contraversial attempts at real congestion avoidance



## 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/cwnd, 3 duplicate acks, g=1/8, initial timeout = 3 seconds, etc



## What's hot now?

- Equation-based congestion control
- Based on TCP behavior (*TCP-friendliness*)

$$BW \approx \frac{MSS}{RTT}\frac{0.7}{\sqrt{p}}$$

- Goal: tailored congestion control response that is still "fair" wrt current implementations
  - E.g. multimedia wants smoother response after loss, but can afford slower ramp up

## Summary

- Congestion avoidance
  - Slow down before congestion occurs
- Congestion control
  - Slow down when congestion occurs

- Can be implemented entirely end-to-end without network support using AIMD algorithms
- Benefit of router involvement?