Lecture 11:
Congestion Control

CS/ECE 438: Communication Networks
Prof. Matthew Caesar
April 9, 2010

Today’s Topic: Vacations

Congestion Control

reading: Peterson and Davie, Ch. 6

- Basics:
  - Problem, terminology, approaches, metrics
- Solutions
  - Router-based: queueing disciplines
  - Host-based: TCP congestion control
- Congestion avoidance
  - DECbit
  - RED gateways
- Quality of service

Congestion Control Basics

- Problem
  - Demand for network resources can grow beyond the resources available
  - Want to provide “fair” amount to each user
- Examples
  - Bandwidth between Chicago and San Francisco
  - Bandwidth in a network link
  - Buffers in a queue

Congestion Collapse

- Definition
  - Increase in network load results in decrease of useful work done
- Many possible causes
  - Spurious retransmissions of packets still in flight
    - Classical congestion collapse
    - Solution: better timers and TCP congestion control
  - Undelivered packets
    - Packets consume resources and are dropped elsewhere in network
    - Solution: congestion control for ALL traffic

Dealing with Congestion

- Range of solutions
  - Congestion control
    - Cure congestion when it happens
  - Resource allocation
    - Prevent congestion from occurring
- Model of network
  - Packet-switched internetwork (or network)
  - Connectionless flows (logical channels/connections) between hosts

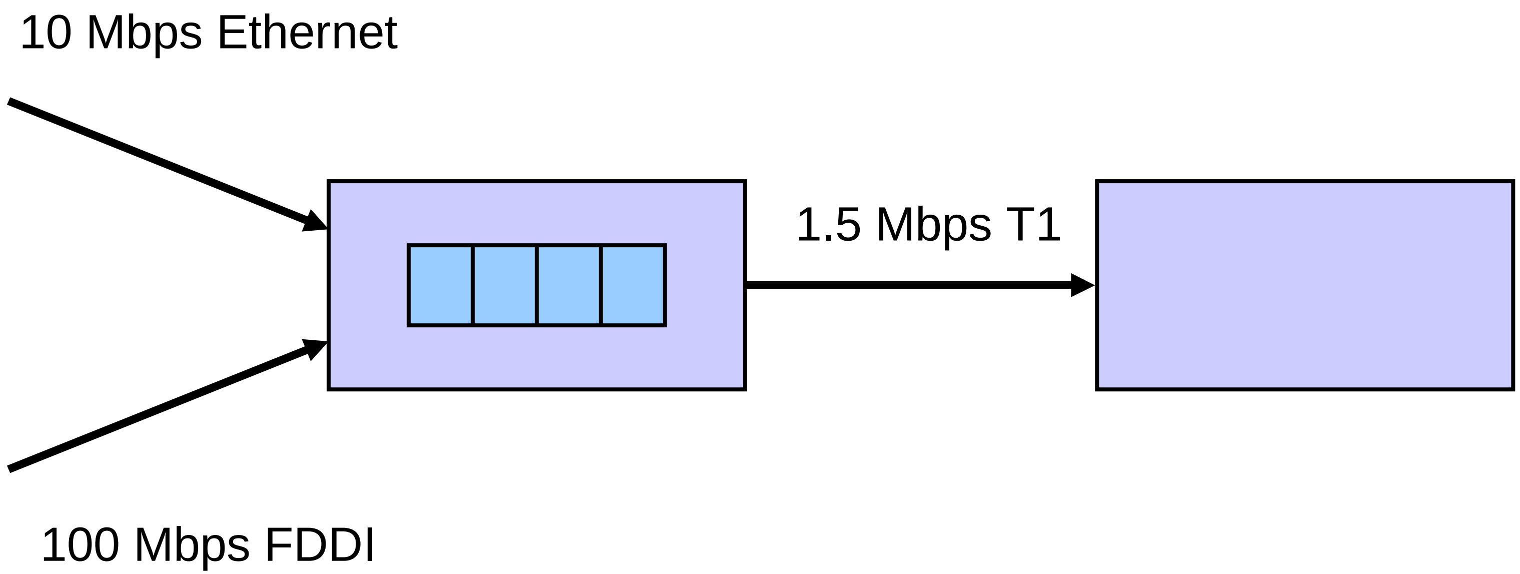
Congestion Control

- Goal
  - Effective and fair allocation of resources among a collection of competing users
  - Learning when to say no and to whom
- Resources
  - Bandwidth
  - Buffers
- Problem
  - Contention at routers causes packet loss

Flow Control vs. Congestion Control

- Flow control
  - Preventing senders from overrunning the capacity of the receivers
- Congestion control
  - Preventing too much data from being injected into the network, causing switches or links to become overloaded

Overview



Congestion cannot be controlled by routing alone

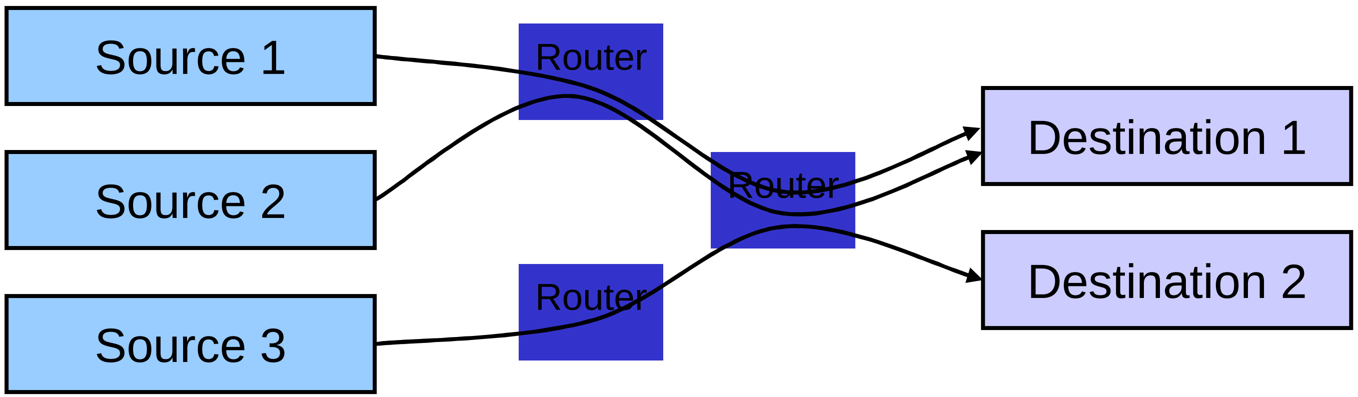
Need to limit traffic on bottleneck link

Basic Design Choices

- Prevention or Cure?
  - Pre-allocate resources to avoid congestion
  - Send data and control congestion if and when it occurs
- Possible implementation points
  - Hosts at the edge of the network
    - Transport protocol
  - Routers inside the network
    - Queueing disciplines
- Underlying service model
  - Best effort vs. quality of service (QoS)

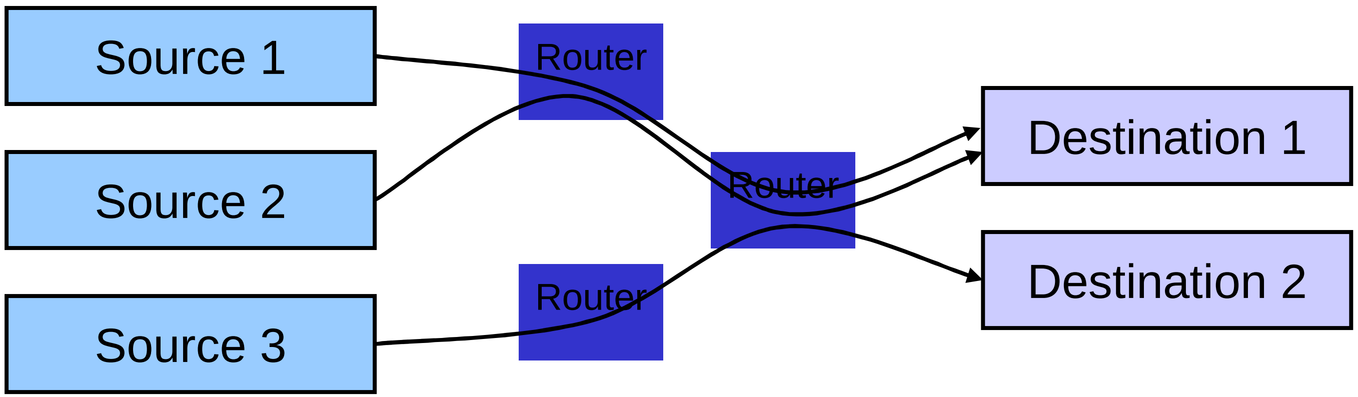
Flows

- Sequence of packets sent between source/destination pair
  – Similar to end-to-end abstraction of channel, but seen at routers
- Maintain per-flow soft state at the routers



Router State

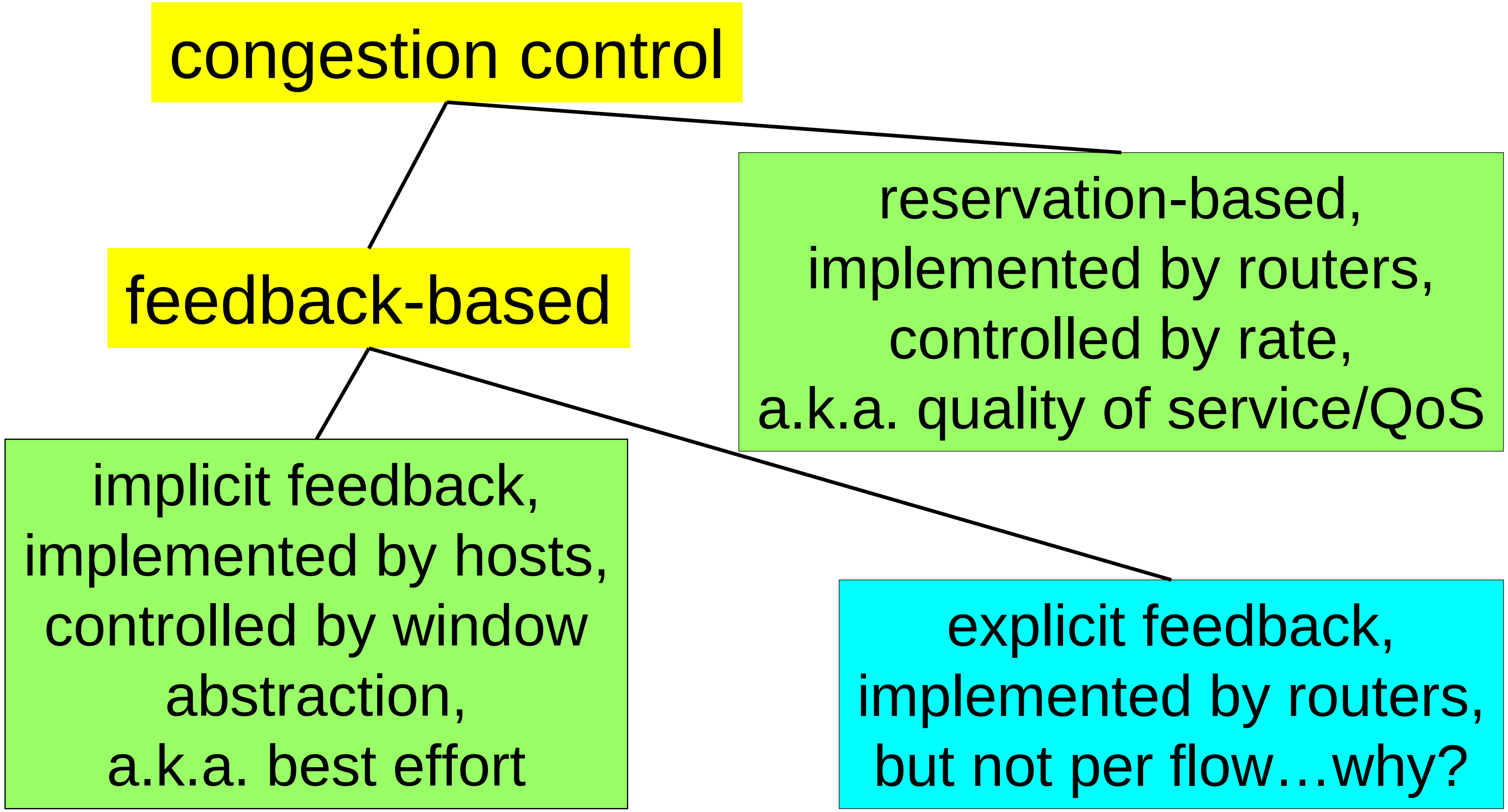
- Soft state:
  - Information about flows
  - Helps control congestion
  - Not necessary for correct routing
- Hard state:
  - state used to support routing



Congestion Control

- Router role
  - Controls forwarding and dropping policies
  - Can send feedback to source
- Host role
  - Monitors network conditions
  - Adjusts accordingly
- Routing vs. congestion
  - Effective adaptive routing schemes can sometimes help congestion
  - But not always

Congestion Control Taxonomy



Router-Centric vs. Host-Centric Flow Control

- Router-centric
  - Each router takes responsibility for deciding
    - When packets are forwarded
    - Which packets are to be dropped
    - Informing hosts of sending limitations
- Host-centric
  - Hosts observe network conditions and adjust their behavior accordingly

Reservation-Based vs. Feedback-Based Flow Control

- Reservation-based
  - End host asks network for capacity at flow establishment time
  - Routers along flow's route allocate appropriate resources
  - If resources are not available, flow is rejected
  - Implies the use of router-centric mechanisms
- Feedback-based
  - End host begins sending without asking for capacity
  - End host adjusts sending rate according to feedback
    - Explicit vs. implicit feedback mechanisms
  - May use router-centric (explicit) or host-centric (implicit) mechanisms

Per-flow Congestion Feedback

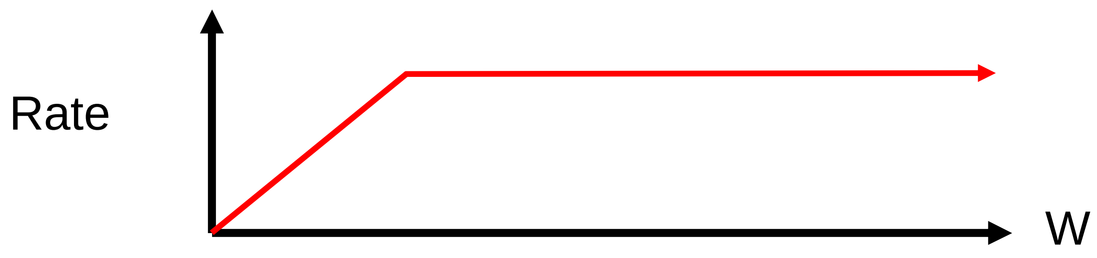
- Question
  - Why is explicit per-flow congestion feedback from routers rarely used in practice?

Per-flow Congestion Feedback

- Problem
  - Too many sources to track
    - Millions of flows may fan in to one router
    - Can’t send feedback to all of them
  - Adds complexity to router
    - Need to track more state
    - Certainly can’t track state for all sources
  - Wastes bandwidth: network already congested, not the time to generate more traffic
  - Can’t force the sources (hosts) to use feedback

Window-based vs. Rate-based Flow Control

- Remember
  - Given a RTT and window size W, long term throughput rate is
    - Rate = min(link speed, W/RTT)
- Since rate can be controlled by the window size, is there really any difference between controlling the window size and controlling the rate?

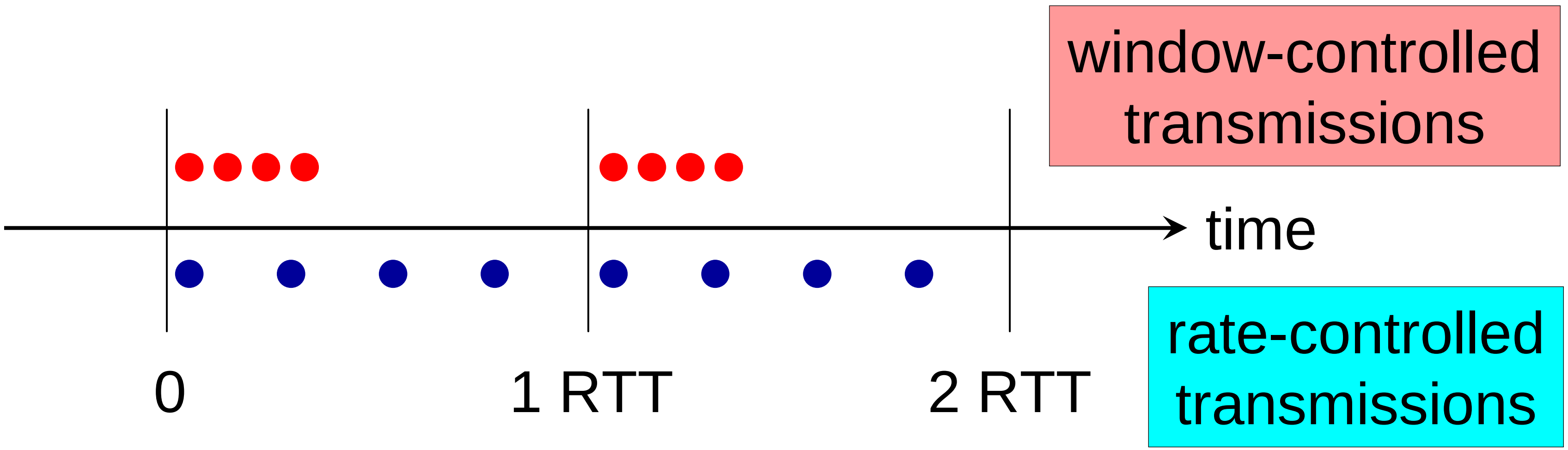


Rate Control

- Question
  - Why consider rate control?
- Problems
  - Buffer space (window size) is an intrinsic physical quantity
  - Can provide rate control with window control
  - Only need estimate of RTT

Answer

Want rate control when granularity of averaging must be smaller than RTT



Criticisms of Resource Allocation

- Example
  - Divide 10 Gbps bandwidth out of UIUC
- Case 1: reserve whatever you want
  - Users' line of thought
    - On average, I don't need much bandwidth, but when my personal Web crawler goes to work, I need at least 100 Mbps, so I'll reserve that much.
  - Result
    - 100 users consume all bandwidth, all others get 0

Criticisms of Resource Allocation

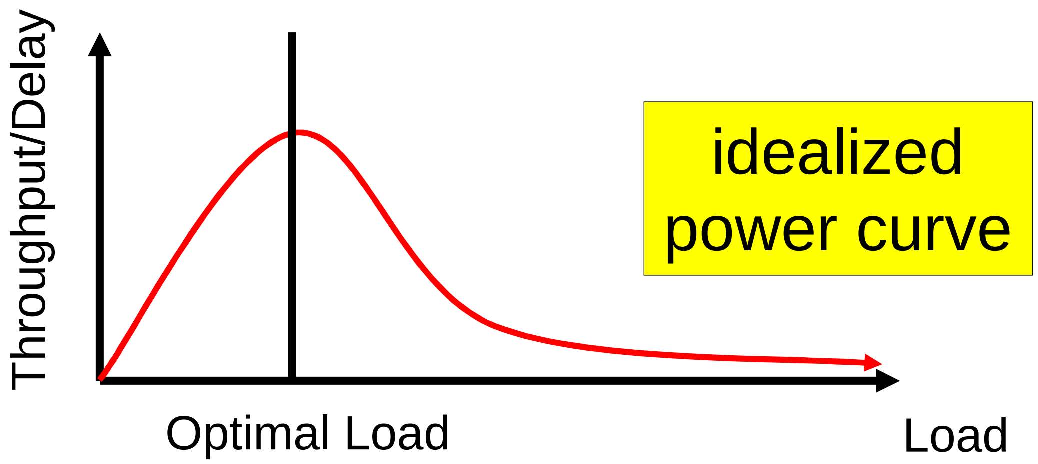
- Example
  - Divide 10 Gbps bandwidth out of UIUC
- Case 2: fair/equitable reservations
  - 35,000 students + 5,000 faculty and staff
  - Each user gets 250 kbps, almost 5x a modem!

Resource Allocation

- Back to the air travel analogy
  - Daily Chicago to San Francisco flight, 198 seats
  - Case 1: reserve whatever you want
    - 198 of us get seats. I'm Gold...are you?
  - Case 2: fair/equitable reservations
    - 2,000,000 possible customers
    - 0.000099 seats per customer per flight
    - Disclaimer:
      the passenger assumes all risks and damages
      related to unsuccessful reassembly in Chicago

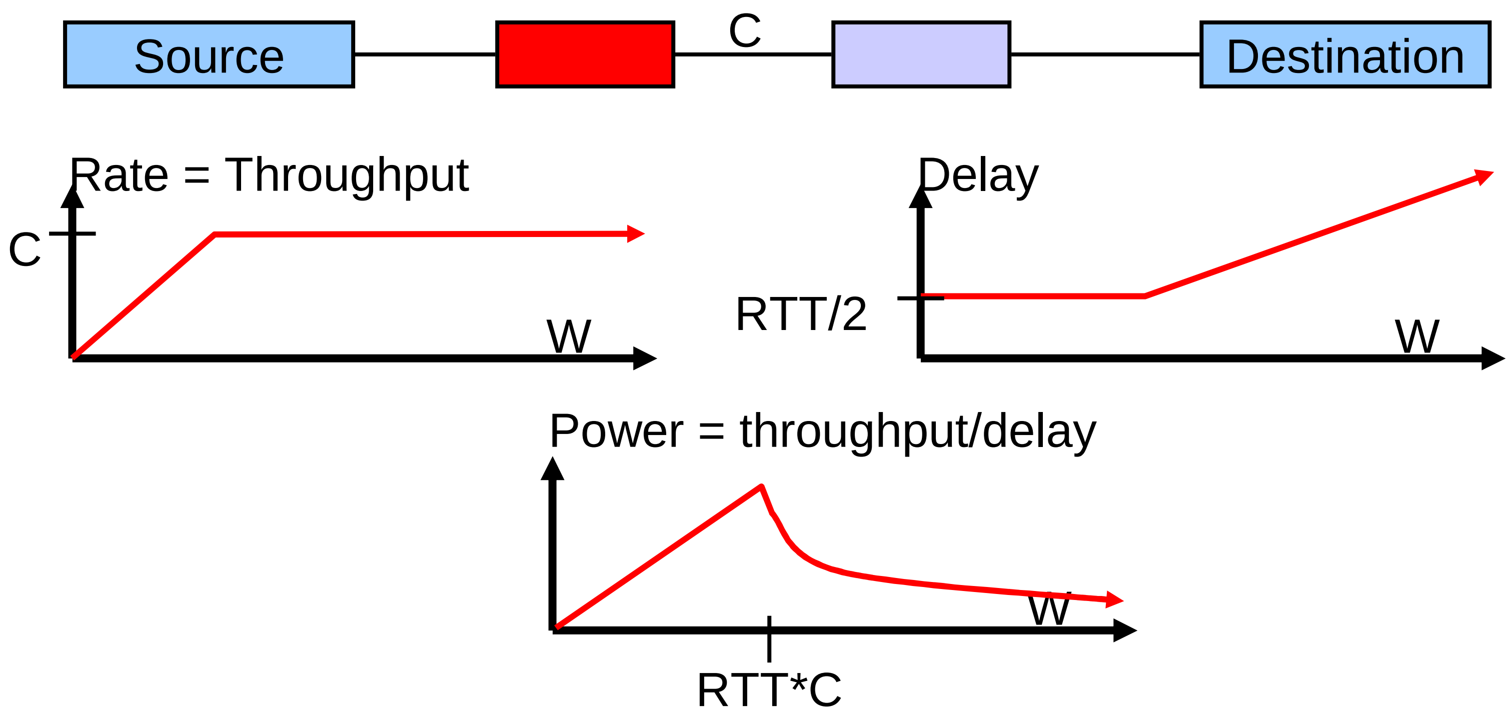
Evaluation

- Fairness
- Power
  - Ratio of throughput to delay
  - Function of load on network
  - Generally relative to a single flow



Window Size

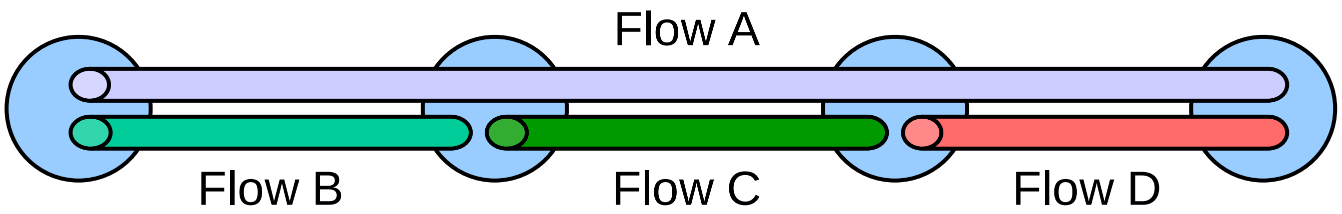
For non-random network with bottleneck capacity C:



Fairness

- Goals
  - Allocate resources "fairly"
  - Isolate ill-behaved users
  - Still achieve statistical multiplexing
    - One flow can fill entire pipe if no contenders
    - Work conserving → scheduler never idles link if it has a packet
- At what granularity?
  - Flows, connections, domains?

What's Fair?



Which is more fair:

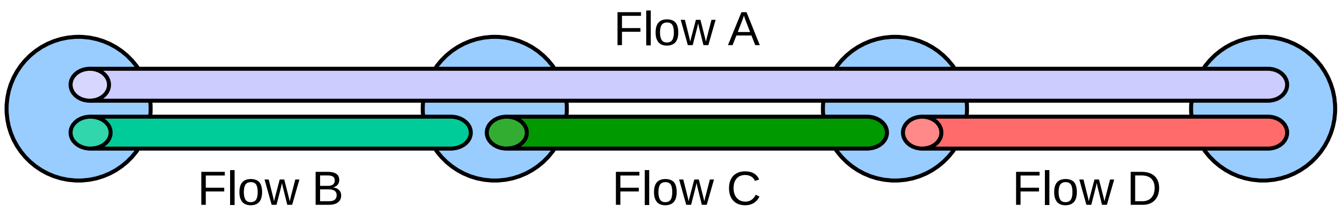
Globally Fair: Fa = Capacity/4, Fb = Fc = Fd = 3Capacity/4

or

Locally Fair: Fa = Fb = Fc = Fd = Capacity/2

This is the so-called "max-min fair" rate allocation. The minimum rate is maximized.

Max-Min Fairness



1. No user receives more than requested bandwidth
2. No other scheme with 1 has higher min bandwidth
3. 2 remains true recursively on removing minimal user $\mu_l = \mathrm{MIN}(\mu_{fair}, \rho_i)$

Queueing Disciplines

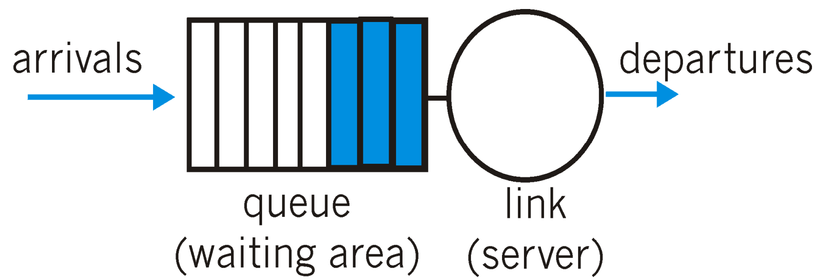
- Goal
  - Decide how packets are buffered while waiting to be transmitted
  - Provide protection from ill-behaved flows
  - Each router MUST implement some queuing discipline regardless of what the resource allocation mechanism is
- Impact
  - Directly impacts buffer space usage
  - Indirectly impacts flow control

Queueing Disciplines

- Allocate bandwidth
  - Which packets get transmitted
- Allocate buffer space
  - Which packets get discarded
- Affect packet latency
  - When packets get transmitted

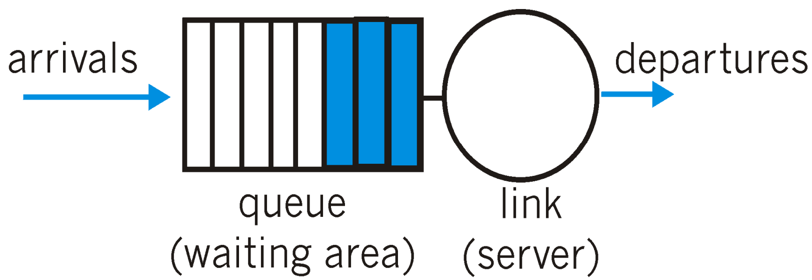
Scheduling Policies

- FIFO (First In First Out) a.k.a. FCFS (First Come First Serve)
  - Service
    - In order of arrival to the queue
  - Management
    - Packets that arrive to a full buffer are discarded
    - Another option: discard policy determines which packet to discard (new arrival or something already queued)



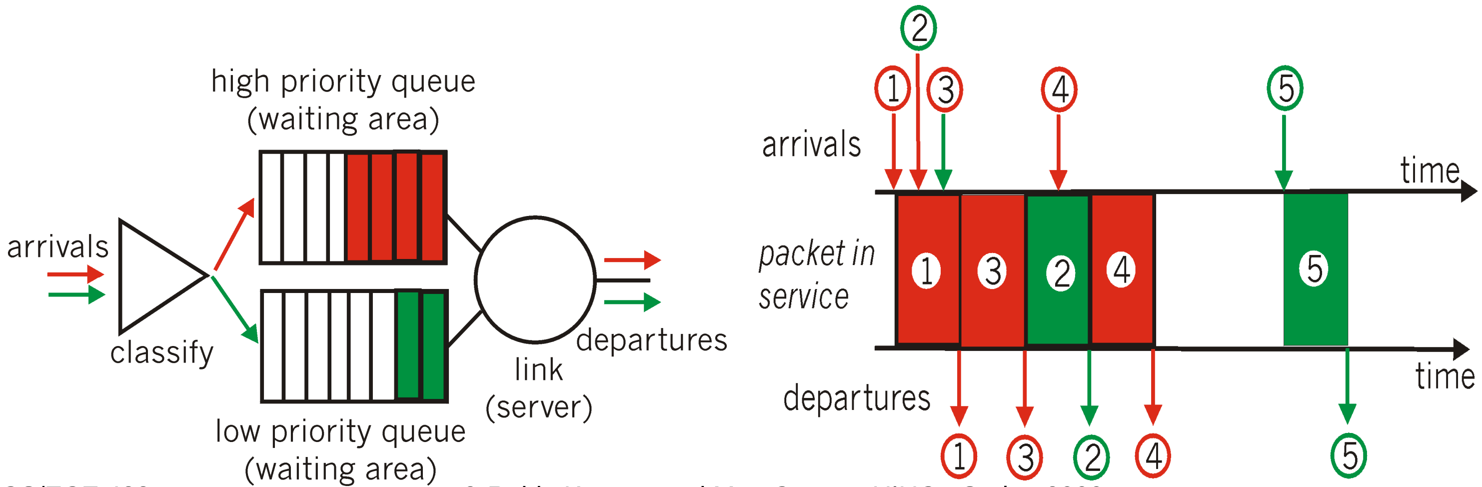
Scheduling Policies

- FIFO
  - Does not discriminate between traffic sources
  - Congestion control left to the sources
  - Tail drop dropping policy
  - Fairness for latency
  - Minimizes per-packet delay
  - Bandwidth not considered (not good for congestion)



Scheduling Policies

- Priority Queuing
  - Classes have different priorities
    - May depend on explicit marking or other header info
      - e.g., IP source or destination, TCP Port numbers, etc.
  - Service
    - Transmit packet from highest priority class with a non-empty queue

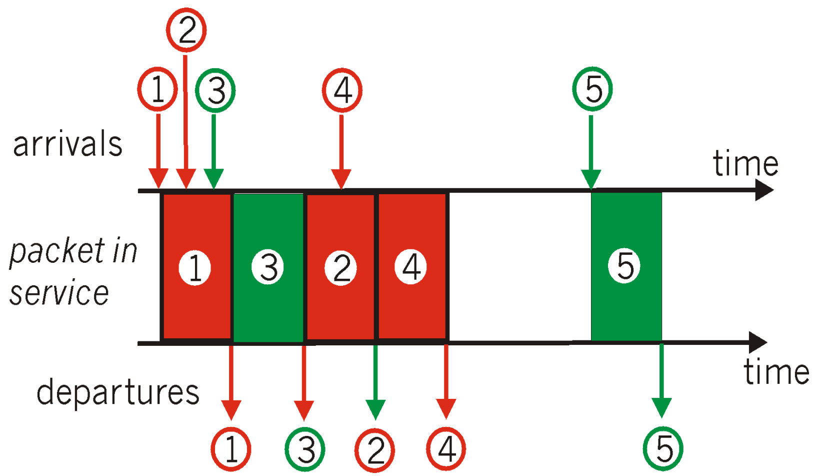


Scheduling Policies

- Priority Queueing Versions
  - Preemptive
    - Postpone low-priority processing if high-priority packet arrives
  - Non-preemptive
    - Any packet that starts getting processed finishes before moving on
- Limitation
  - May starve lower priority flows

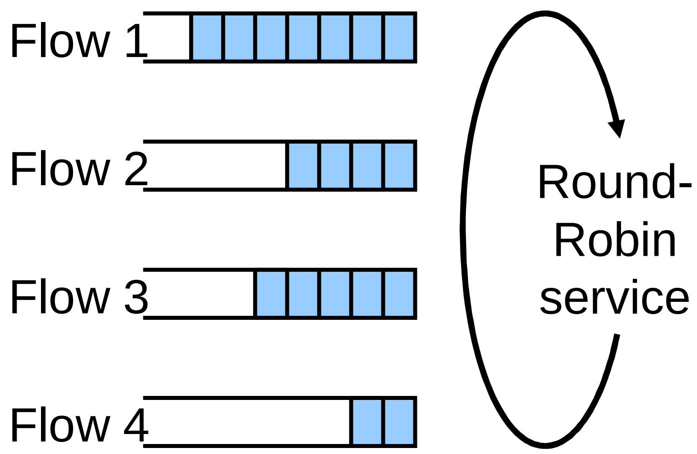
Scheduling Policies

- Round Robin
  - Each flow gets its own queue
  - Circulate through queues, process one packet (if queue non-empty), then move to next queue



Scheduling Policies

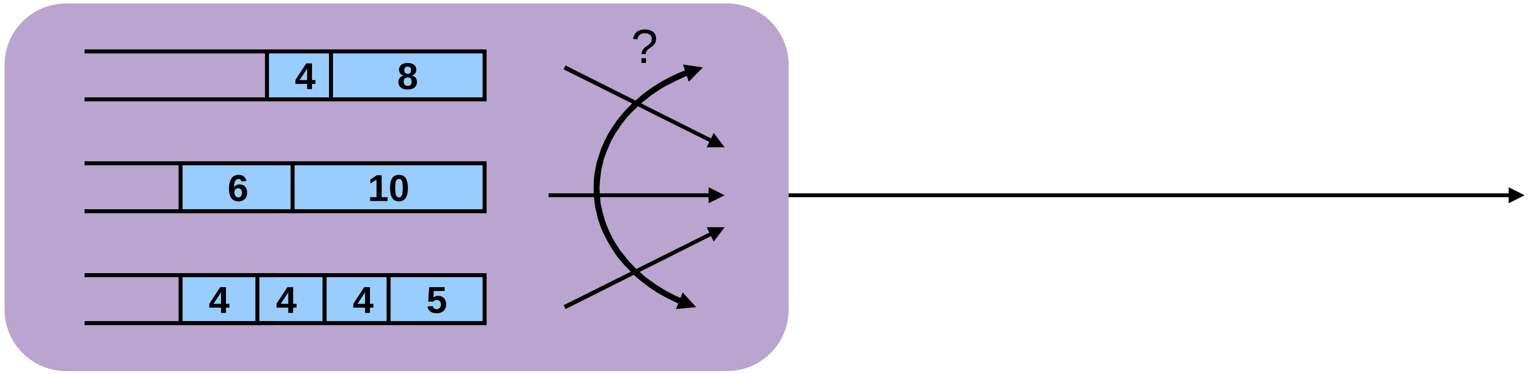
- Fair Queueing (FQ)
  - Explicitly segregates traffic based on flows
  - Ensures no flow captures more than its share of the capacity
  - Fairness for bandwidth
  - Delay not considered



Each flow is guaranteed ¼ of capacity

Fair Queueing with Variable Packet Length

- How should we implement FQ if packets are not all the same length?
  - Bit-by-bit round-robin
    - Not feasible to implement, must use packet scheduling
    - Solution: approximate



Fair Queueing with Variable Packet Length

- Idea
  - Let S_i = amount of service flow i has received so far
  - Always serve a flow with minimum value of Si
    - Can also use minimum (S_i + next packet length)
  - Upon serving a packet of length P from flow i, update:
    - $S_i = S_i + P$
- Never leave the link idle if there is a packet to send
  - Work conserving
    - A source will gets its fair share of the bandwidth
    - Unused bandwidth will be evenly divided between other sources

Fair Queueing with Variable Packet Length

- Problem
  - A flow resumes sending packets after being quite for a long time
- Effect
  - Such a flow could be considered to have “saved up credit”
  - Can lock out all other flows until credits are level again
- Solution
  - Enforce “use it or lose it policy”
    - Compute S_{min} = min(S_i such that queue i is not empty)
    - If queue j is empty, set $S_j = S_{min}$

Fair Queueing with Variable Packet Length

- Problem
  - A flow resumes sending packets after being quite for a long time
- Effect
  - Such a flow could be
  - Can lock out all othe
- Solution
  - Enforce "use it or los
    - Compute S_{min} = mi
    - If queue j is empty

~~Note:~~

~~The text book computes~~

$\cancel{F = MAX(F_{i-1}, A_i) = P_i}$

~~And then for multiple flows~~

- ~~Calculate F_i for each packet that arrives on each flow~~
- ~~Treat all F_i as timestamps~~
- ~~Next packet to transmit is one with lowest timestamp~~

Extension: Weighted Fair Queueing

- Extend fair queueing
  - Notion of importance for each flow
- Suppose flow i has weight w_i
  - Example: w_i could be the fraction of total service that flow i is targeted for
- Need only change basic update to
  - $S_i = S_i + P/w_i$

Fair Queuing Tradeoffs

- FQ can control congestion by monitoring flows
  - Non-adaptive flows can still be a problem – why?
- Complex state
  - Must keep queue per flow
    - Hard in routers with many flows (e.g., backbone routers)
    - Flow aggregation is a possibility (e.g. do fairness per domain)
- Complex computation
  - Classification into flows may be hard
  - Must keep queues sorted by finish times
  - Changes whenever the flow count changes

Fair Queueing

- Question
  - What makes up a flow for fair queueing in the Internet?
- Considerations
  - Too many resources to have separate queues/variables for host-to-host flows
  - Scale down number of flows
  - Typically just based on inputs
    - e.g., share outgoing STS-12 between incoming ISP's

TCP Congestion Control

Host Solutions

- Host has very little information
  - Assumes best-effort network
  - Acts independently of other hosts
- Host infers congestion
  - From synchronization feedback (e.g., dropped packet timeouts, duplicate ACK's)
  - Loss on wired lines rarely due to transmission error
- Host acts
  - Reduce transmission rate below congestion threshold
  - Continuously monitor network for signs of congestion

TCP Congestion Control

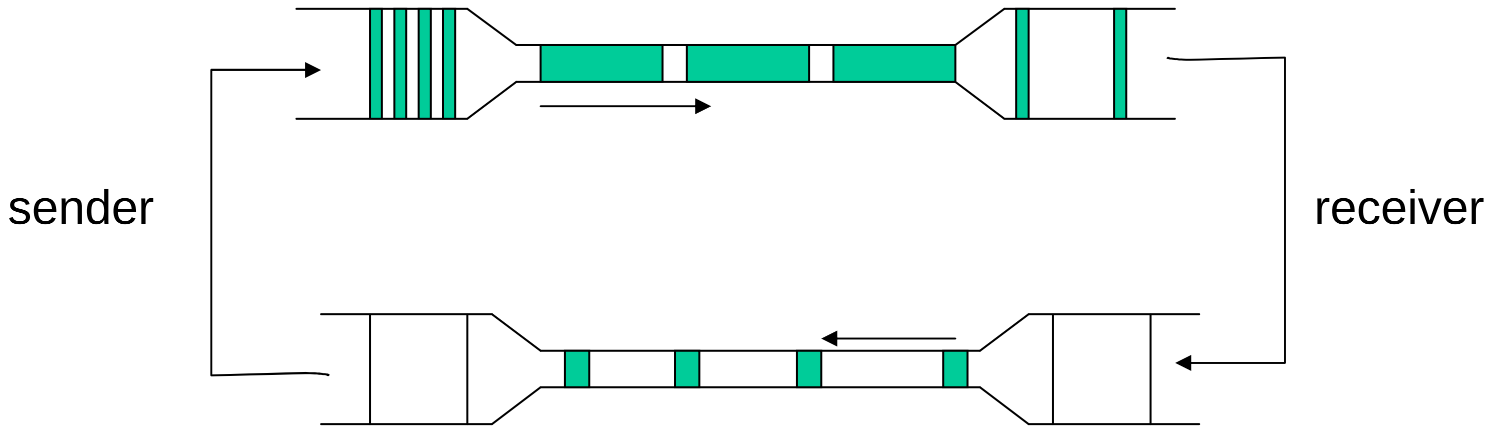
- Idea
  - Assumes best-effort network
    - FIFO or FQ
  - Each source determines network capacity for itself
  - Implicit feedback
  - ACKs pace transmission (self-clocking)
- Challenge
  - Determining initial available capacity
  - Adjusting to changes in capacity in a timely manner

TCP Congestion Control

- Basic idea
  - Add notion of congestion window
  - Effective window is smaller of
    - Advertised window (flow control)
    - Congestion window (congestion control)
  - Changes in congestion window size
    - Slow increases to absorb new bandwidth
    - Quick decreases to eliminate congestion

TCP Congestion Control

- Specific strategy
  - Self-clocking
    - Send data only when outstanding data ACK'd
    - Equivalent to send window limitation mentioned



TCP Congestion Control

- Specific strategy
  - Self-clocking
    - Send data only when outstanding data ACK'd
    - Equivalent to send window limitation mentioned
  - Growth
    - Add one maximum segment size (MSS) per congestion window of data ACK'd
    - It's really done this way, at least in Linux:
      - see tcp_cong_avoid in tcp_input.c.
      - Actually, every ack for new data is treated as an MSS ACK'd
    - Known as additive increase

TCP Congestion Control

- Specific strategy (continued)
  - Decrease
    - Cut window in half when timeout occurs
    - In practice, set window = window /2
    - Known as multiplicative decrease
  - Additive increase, multiplicative decrease (AIMD)

Additive Increase/ Multiplicative Decrease

- Objective
  - Adjust to changes in available capacity
- Tools
  - React to observance of congestion
  - Probe channel to detect more resources
- Observation
  - On notice of congestion
    - Decreasing too slowly will not be reactive enough
  - On probe of network
    - Increasing too quickly will overshoot limits

Additive Increase/ Multiplicative Decrease

- New TCP state variable
  - **`CongestionWindow`**
    - Similar to **`AdvertisedWindow`** for flow control
  - Limits how much data source can have in transit
    - **`MaxWin = MIN(CongestionWindow, AdvertisedWindow)`**
    - **`EffWin = MaxWin - (LastByteSent - LastByteAcked)`**
    - TCP can send no faster then the slowest component, network or destination
- Idea
  - Increase **`CongestionWindow`** when congestion goes down
  - Decrease **`CongestionWindow`** when congestion goes up

Additive Increase/ Multiplicative Decrease

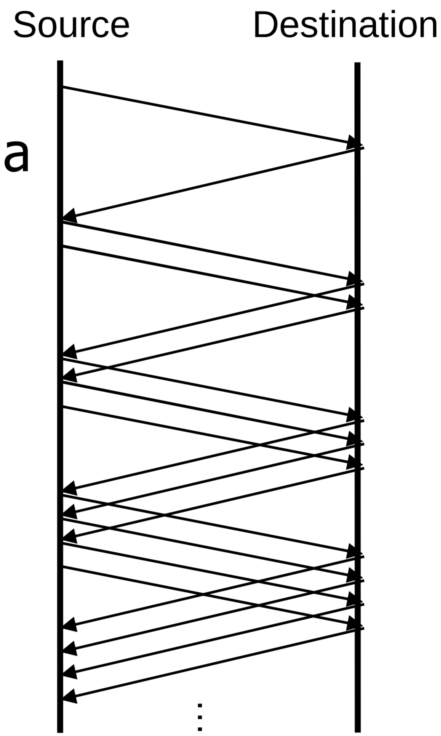
- Question
  - How does the source determine whether or not the network is congested?

- Answer
  - Timeout signals packet loss
  - Packet loss is rarely due to transmission error (on wired lines)
  - Lost packet implies congestion!

Additive Increase/ Multiplicative Decrease

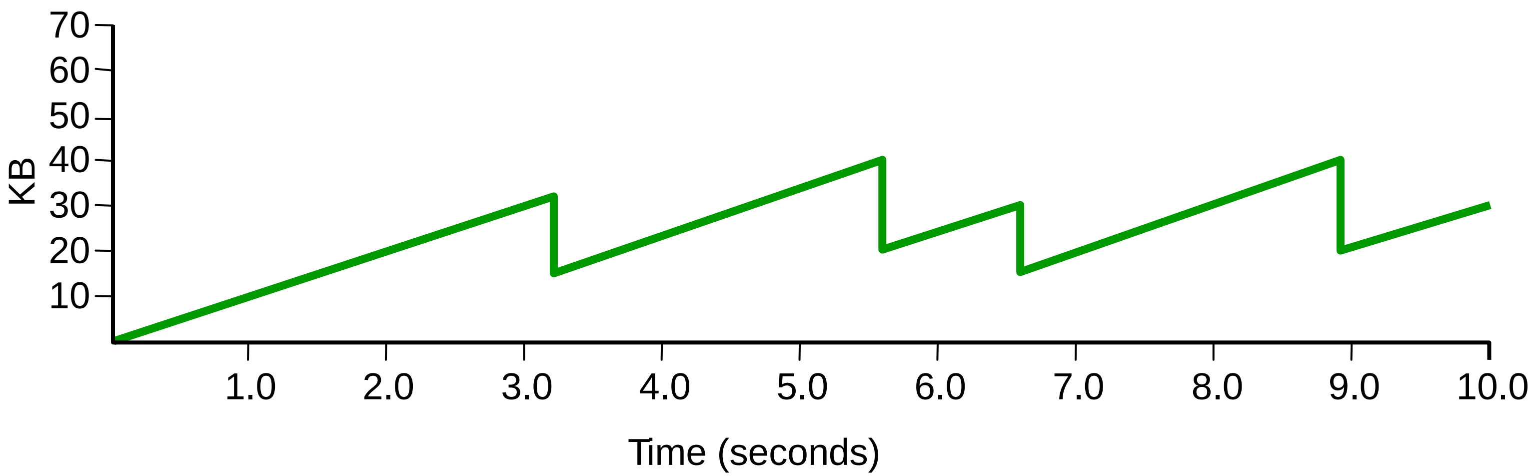
- Algorithm
  - Increment CongestionWindow by one packet per RTT
    - Linear increase
  - Divide CongestionWindow by two whenever a timeout occurs
    - Multiplicative decrease
- In practice
  - increment a little for each ACK

```
Inc = MSS * MSS/CongestionWindow
CongestionWindow += Inc
```



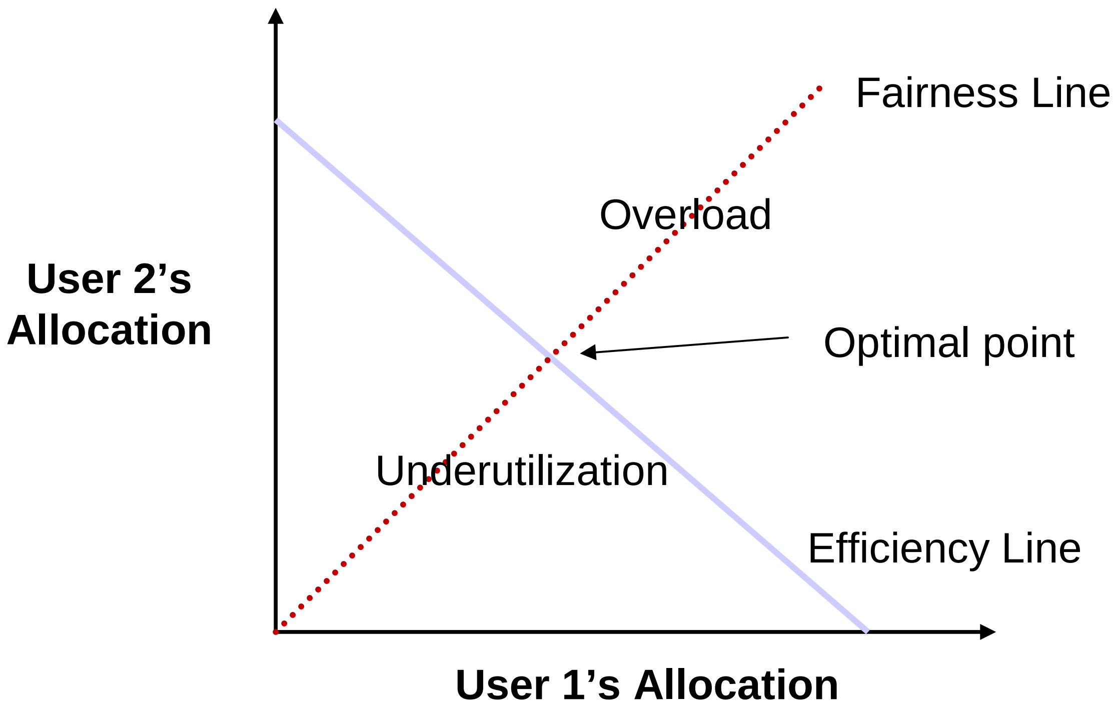
AIMD – Sawtooth Trace

- Packet loss is seen as sign of congestion and results in a multiplicative rate decrease
  - Factor of 2
- TCP periodically probes for available bandwidth by increasing its rate



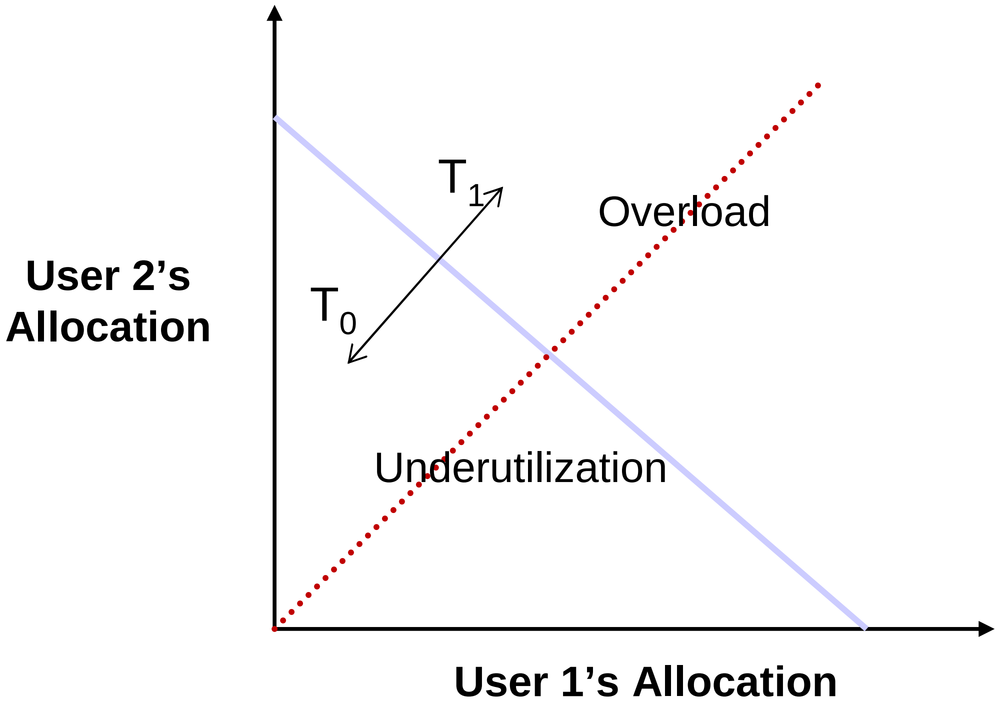
Why is AIMD Fair?

- Two competing sessions



Additive Increase/Decrease

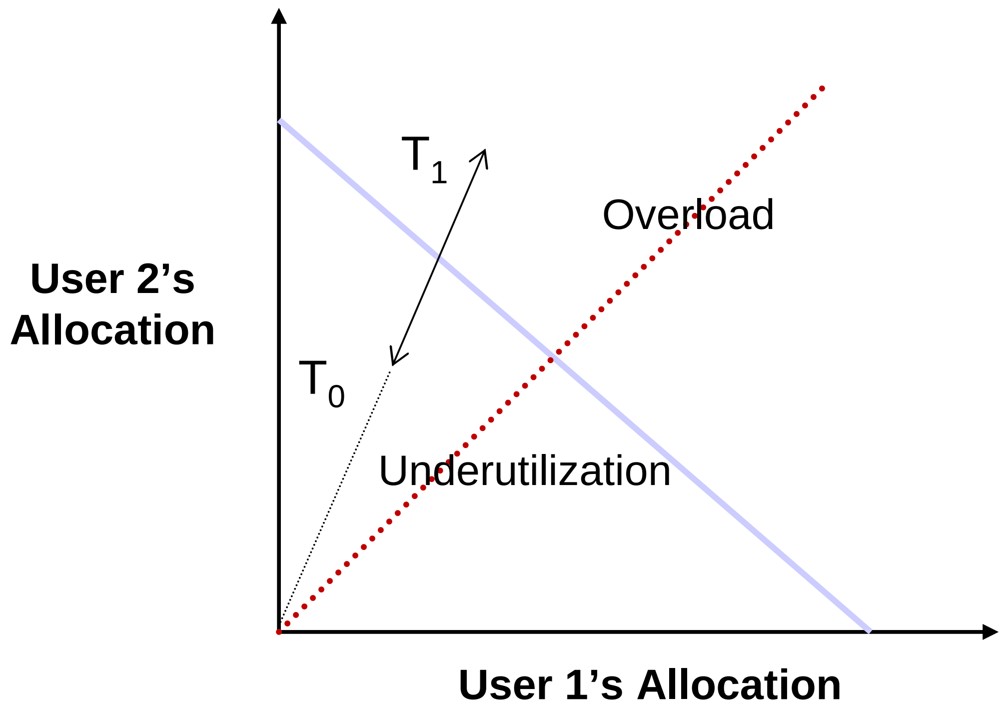
- Both increase/ decrease by the same amount



- Additive increase improves fairness
- Additive decrease reduces fairness

Muliplicative Increase/Decrease

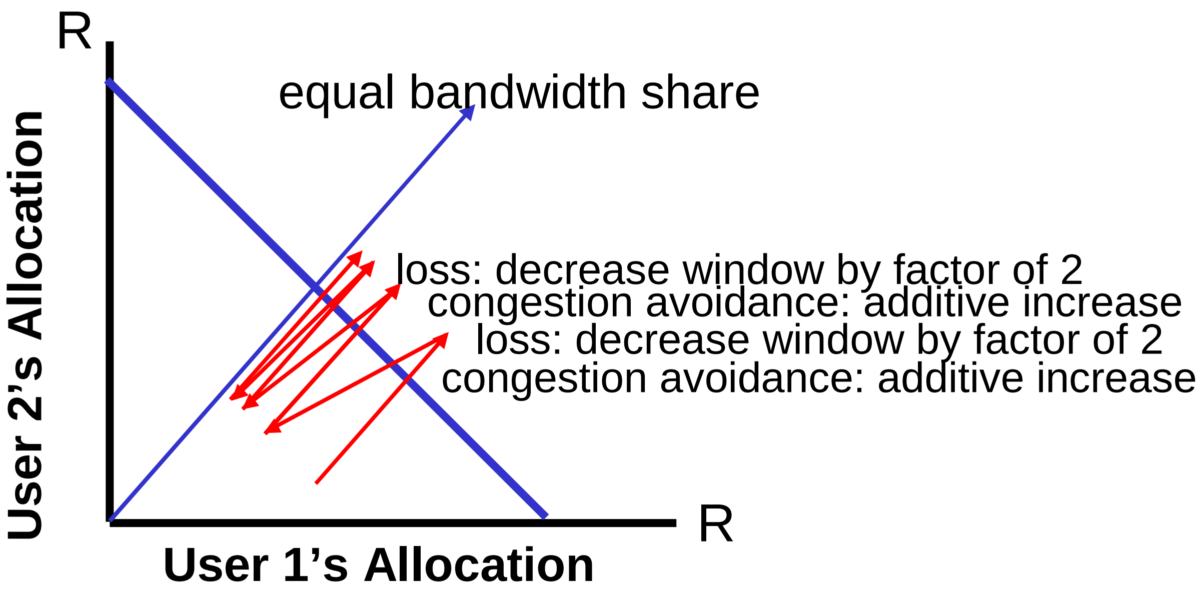
- Both increase/ decrease by the same amount



- Additive increase improves fairness
- Additive decrease reduces fairness

Why is AIMD Fair?

- Additive increase gives slope of 1, as throughout increases
- Multiplicative decrease decreases throughput proportionally



TCP Start Up Behavior

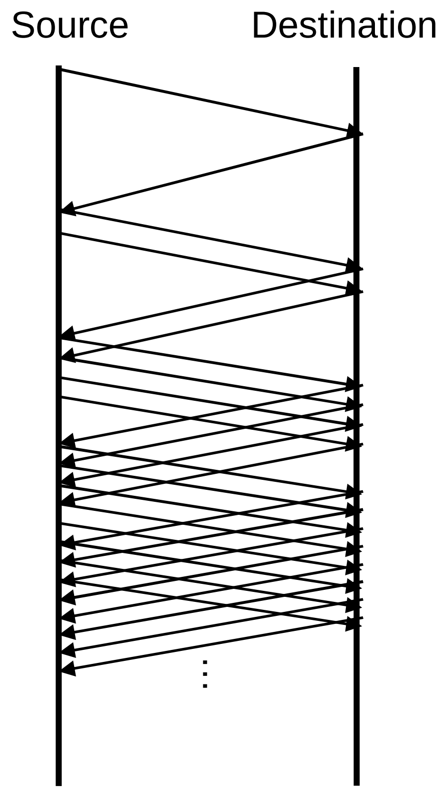
- How should TCP start sending data?
  - AIMD is good for channels operating at capacity
  - AIMD can take a long time to ramp up to full capacity from scratch
  - Use Slow Start to increase window rapidly from a cold start

TCP Start Up Behavior

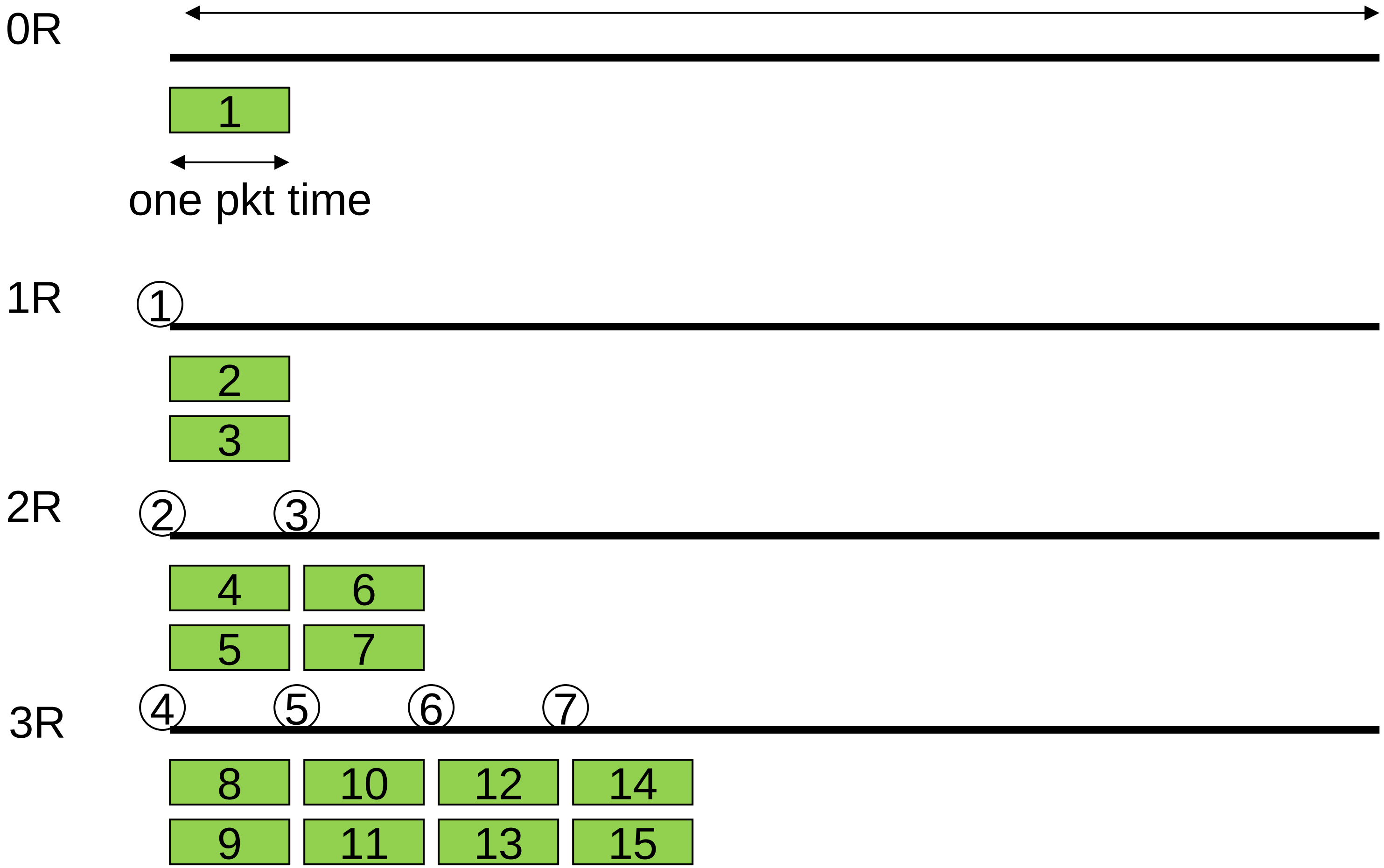
- Initialization of the congestion window
  - Congestion window should start small
  - Avoid congestion due to new connections
  - Start at 1 MSS, reset to 1 MSS with each timeout (note that timeouts are coarse-grained, ~1/2 sec)
  - Known as slow start

Slow Start

- Objective
  - Determine initial available capacity
- Idea
  - Begin with **`CongestionWindow`** = 1 packet
  - Double **`CongestionWindow`** each RTT
    - Increment by 1 packet for each ACK
  - Continue increasing until loss



Slow Start Example



Slow Start

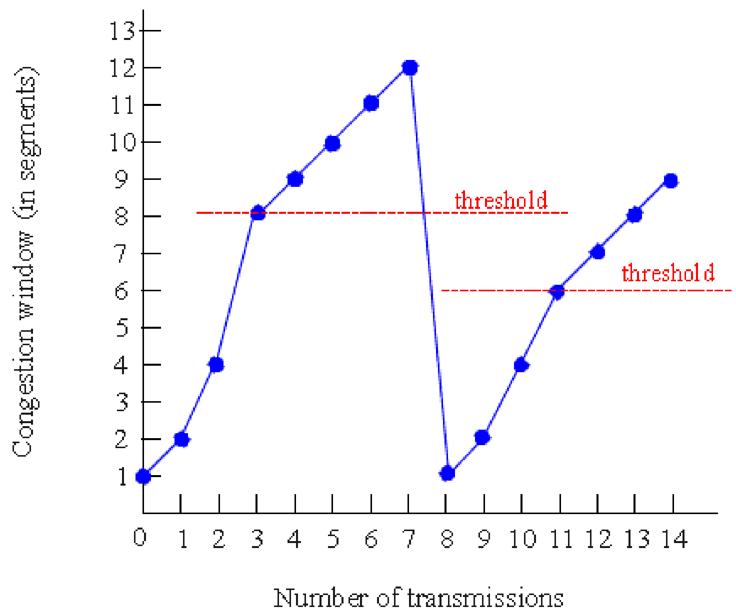
- Result
  - Exponential growth
  - Slower than all at once
- Used
  - When first starting connection
  - When connection times out

TCP Congestion Control

- Maintain threshold window size
  - Threshold value
    - Initially set to maximum window size
    - Set to 1/2 of current window on timeout
  - Use multiplicative increase
    - When congestion window when smaller than threshold
    - Double window for each window ACK'd
- In practice
  - Increase congestion window by one MSS for each ACK of new data (or N bytes for N bytes)

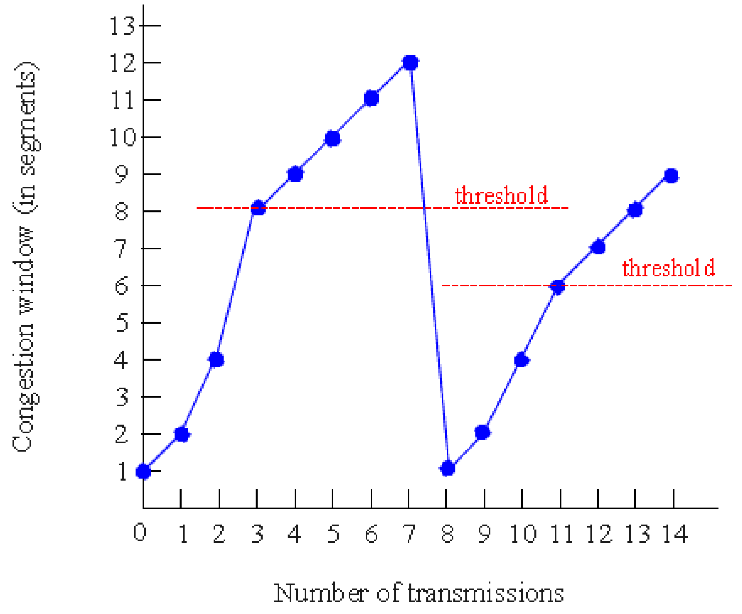
Slow Start

- How long should the exponential increase from slow start continue?
  - Use **`CongestionThreshold`** as target window size
  - Estimates network capacity
  - When **`CongestionWindow`** reaches **`CongestionThreshold`** switch to additive increase



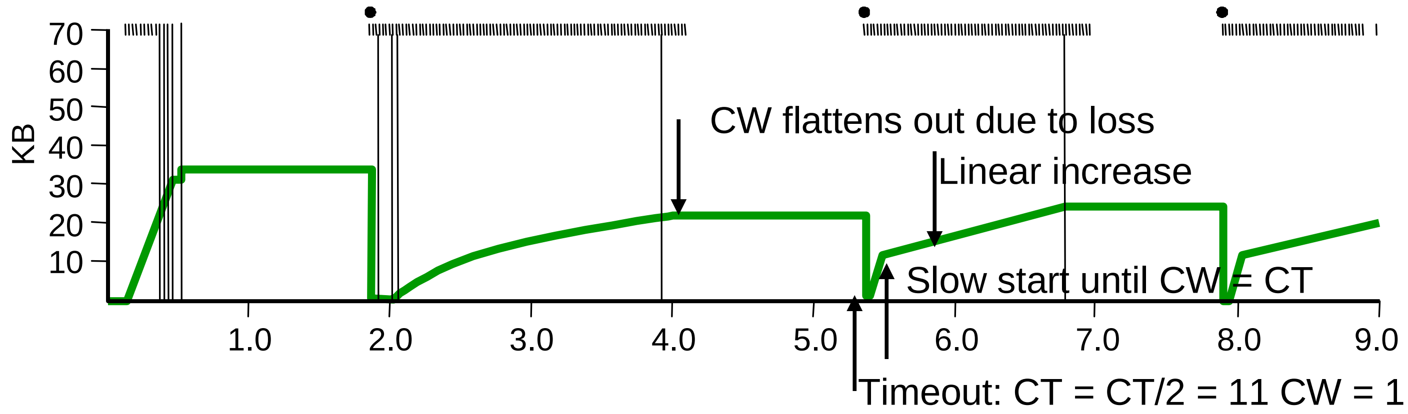
Slow Start

- Initial values
  - `CongestionThreshold = 8`
  - `CongestionWindow = 1`
- Loss after transmission 7
  - `CongestionWindow` currently 12
  - Set `Congestionthreshold = CongestionWindow/2`
  - Set `CongestionWindow = 1`



Slow Start

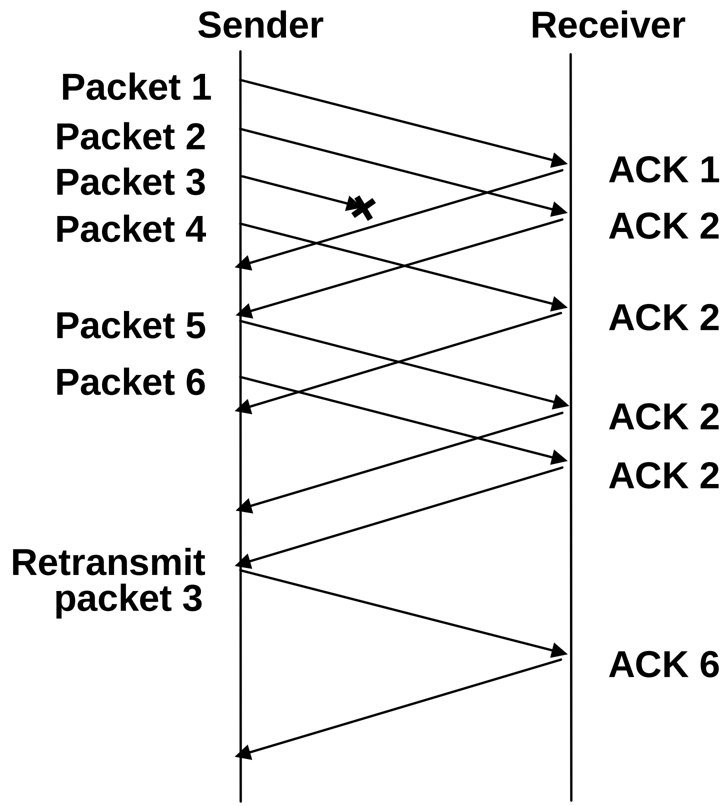
- Example trace of **`CongestionWindow`**



- Problem
  - Have to wait for timeout
  - Can lose half **`CongestionWindow`** of data

Fast Retransmit and Fast Recovery

- Problem
  - Coarse-grain TCP timeouts lead to idle periods
- Solution
  - Fast retransmit: use duplicate ACKs to trigger retransmission

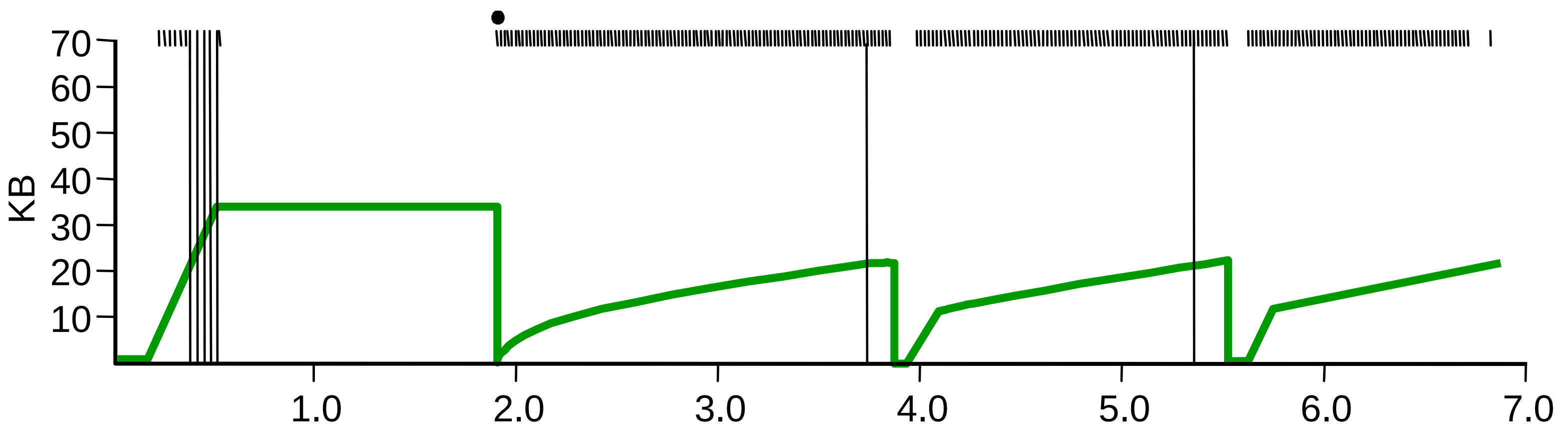


Fast Retransmit and Fast Recovery

- Send ACK for each segment received
- When duplicate ACK's received
  - Resend lost segment immediately
  - Do not wait for timeout
  - In practice, retransmit on 3rd duplicate
- Fast recovery
  - When fast retransmission occurs, skip slow start
  - Congestion window becomes 1/2 previous
  - Start additive increase immediately

Fast Retransmit and Fast Recovery

- Results



- Fast Recovery
  - Bypass slow start phase
  - Increase immediately to one half last successful **`CongestionWindow`** **(`ssthresh`)**

TCP Congestion Window Trace