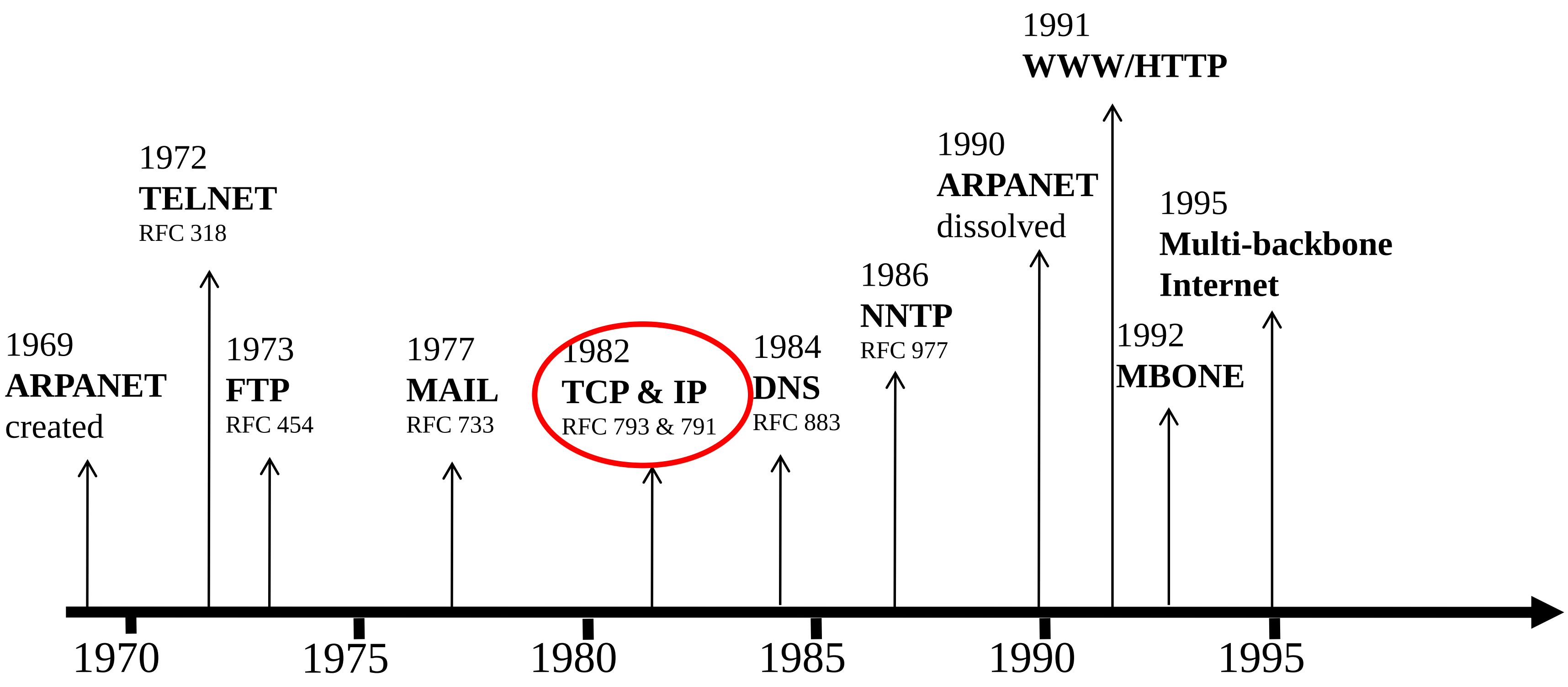
A Brief Primer on TCP/IP

Tom Anderson

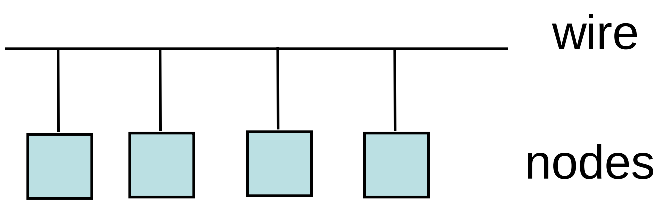
A brief Internet history...

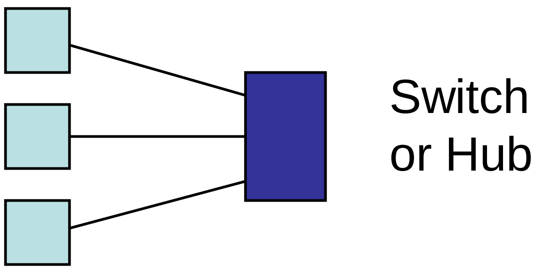


Limits of a single wire LAN

One wire can limit us in terms of:

- Distance
- Number of nodes
- Performance





How do we scale to a larger, faster network?

Scaling beyond one wire

Intra-network:

- Hubs, switches

Inter-network:

- Routers

Key tasks:

- Routing, forwarding, addressing

Key challenges:

- Scale, heterogeneity, robustness

Forwarding vs. routing

Forwarding: the process that each router goes through for every packet to send it on its way

- Involves local decisions

Routing: the process that all routers go through to calculate the routing tables

- Involves non-local decisions

Three ways to forward

Source routing

- The source embeds path information in packets
- E.g., Driving directions

Datagram forwarding

- The source embeds destination address in the packet
- E.g., Postal service

Virtual circuits

- Pre-computed connections: static or dynamic
- Embed connection IDs in packets
- E.g., Airline travel

Routing goals

Compute best path

- Defining "best" is slippery

Scale to billions of hosts

- Minimize control messages and routing table size

Quickly adapt to failures or changes

- Node and link failures, plus message loss

Routing alternatives

Spanning Tree (Ethernet)

- Convert graph into a tree; route only along tree

Distance vector (RIP)

- exchange routing tables with neighbors
- no one knows complete topology

Link state (OSPF, IS-IS)

- send everyone your neighbors
- everyone computes shortest path

Distance vector routing

Each router periodically exchanges messages with neighbors

- best known distance to each destination ("distance vector")

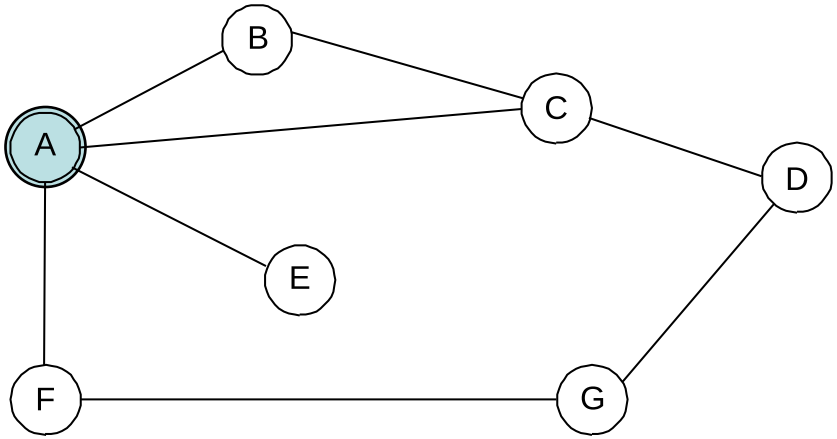
Initially, can get to self with zero cost

On receipt of update from neighbor, for each destination

- switch forwarding tables to neighbor if it has cheaper route
- update best known distance
- tell neighbors of any changes

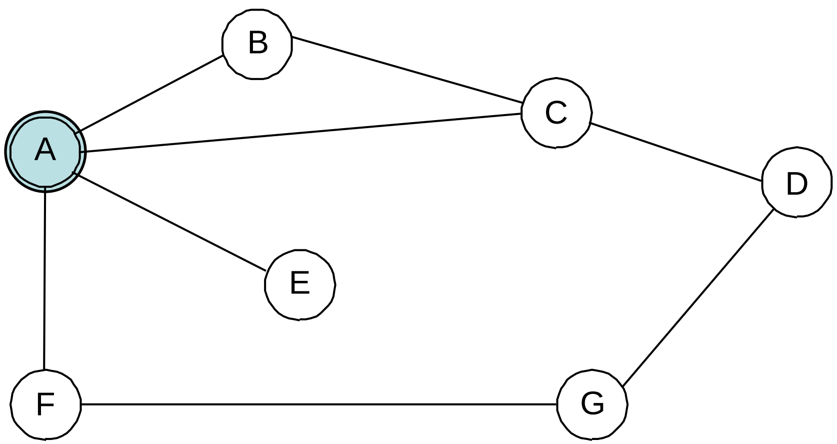
Absent topology changes, will converge to shortest path

DV Example: Initial Table at A



Dest	Cost	Next
A	0	here
B	∞	-
C	∞	-
D	∞	-
E	∞	-
F	∞	-
G	∞	-

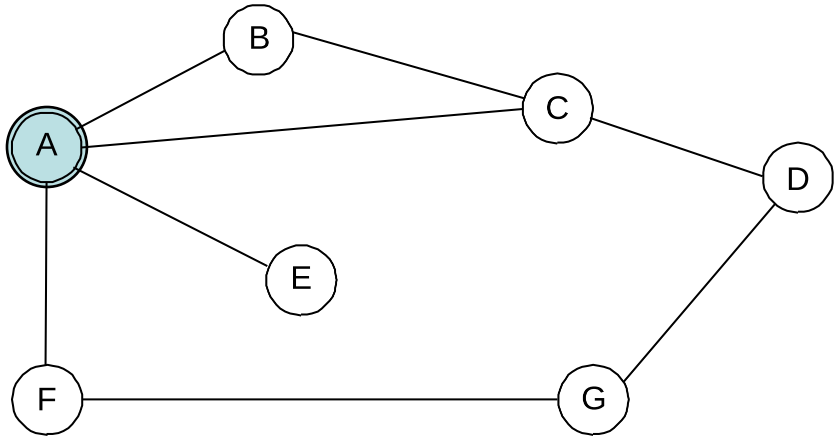
DV Example: Table at A, step 1



Dest	Cost	Next
A	0	here
B	1	B
C	1	C
D	∞	-
E	1	E
F	1	F
G	∞	-

DV Example: Final Table at A

Reached in two iterations
=> simple example

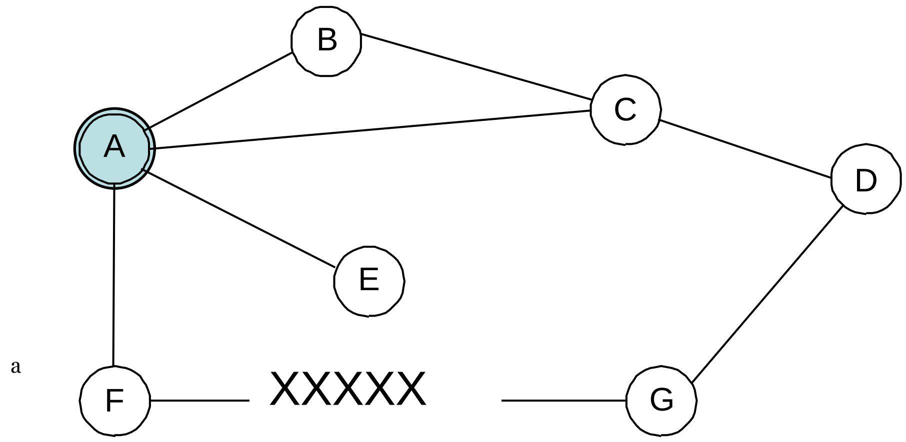


Dest	Cost	Next
A	0	here
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

What if there are changes?

Suppose link between F and G fails

1. F notices failure, sets its cost to G to infinity and tells A
2. A sets its cost to G to infinity too, since it can't use F
3. A learns route from C with cost 2 and adopts it

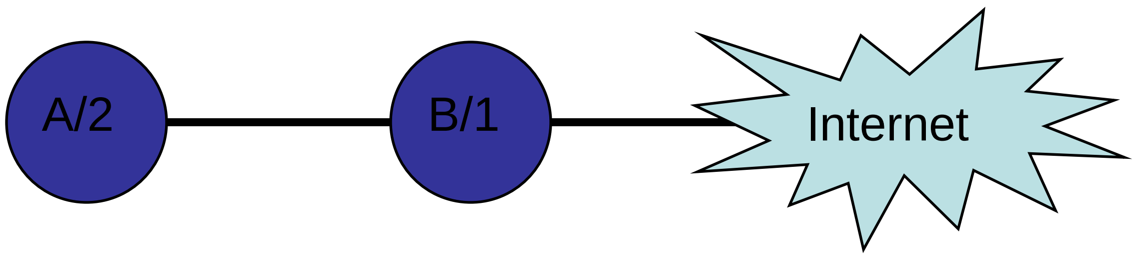


Dest	Cost	Next
A	0	here
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	3	F

Count To Infinity Problem

Simple example

- Costs in nodes are to reach Internet

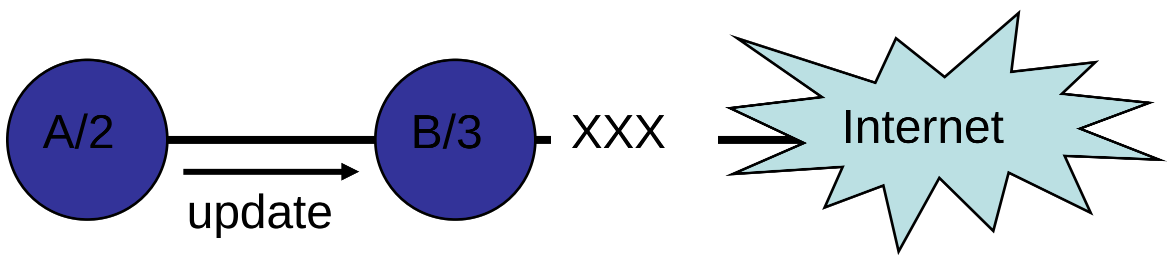


Now link between B and Internet fails …

Count To Infinity Problem

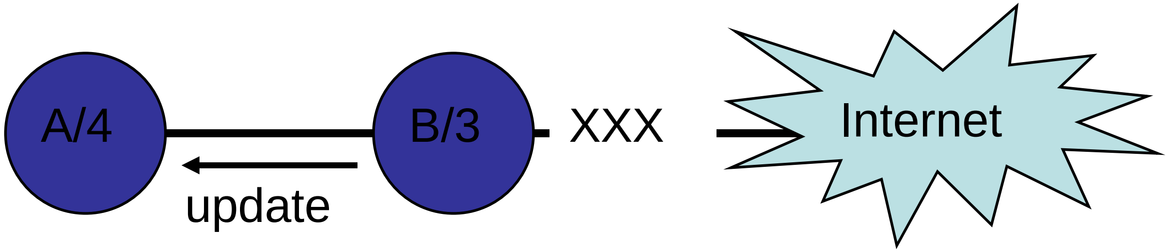
B hears of a route to the Internet via A with cost 2

So B switches to the “better” (but wrong!) route



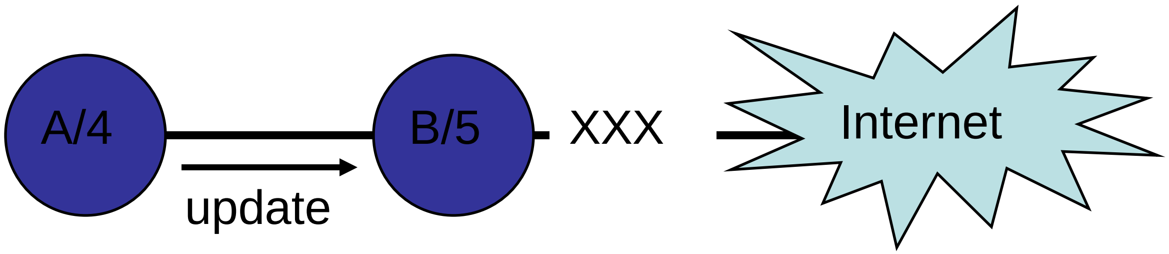
Count To Infinity Problem

A hears from B and increases its cost



Count To Infinity Problem

B hears from A and (surprise) increases its cost
Cycle continues and we "count to infinity"



Packets caught in a loop between A and B

Solutions to count to infinity

Lower infinity ☺

Split horizon

- Do not advertises the destination back to its next hop – that's where it learned it from!
- Solves trivial count-to-infinity problem

Poisoned reverse (RIP)

- Go farther: advertise infinity back to next hop

Link state routing

Every router learns complete topology and then runs shortest-path

Two phases:

- Topology dissemination -- each node gets complete topology via reliable flooding
- Shortest-path calculation (Dijkstra's algorithm)

As long as every router uses the same information, will reach consistent tables

Topology flooding

Each router identifies direct neighbors; put in numbered link state packets (LSPs) and periodically send to neighbors

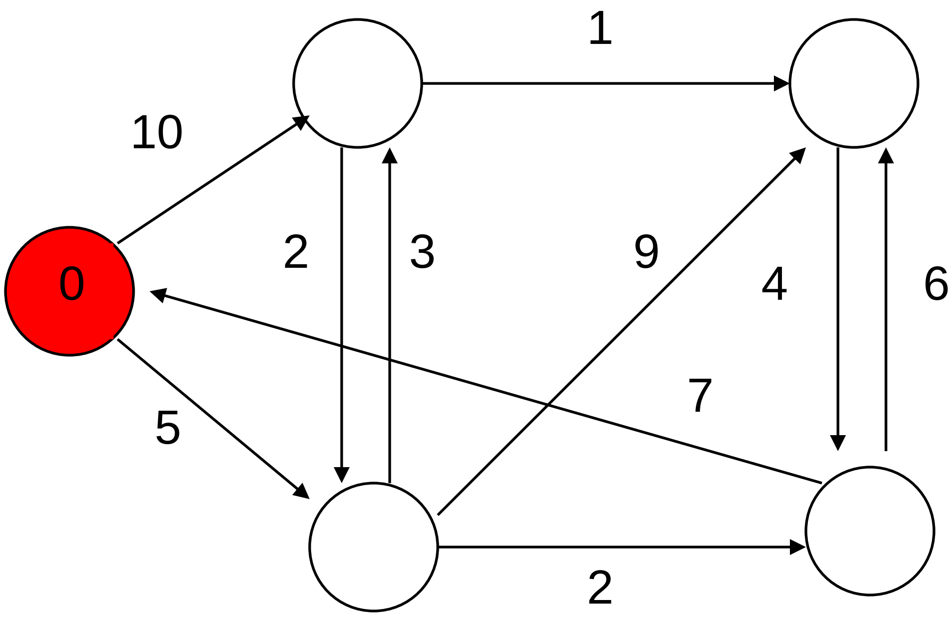
- LSPs contain [router, neighbors, costs]

If get a link state packet from neighbor Q

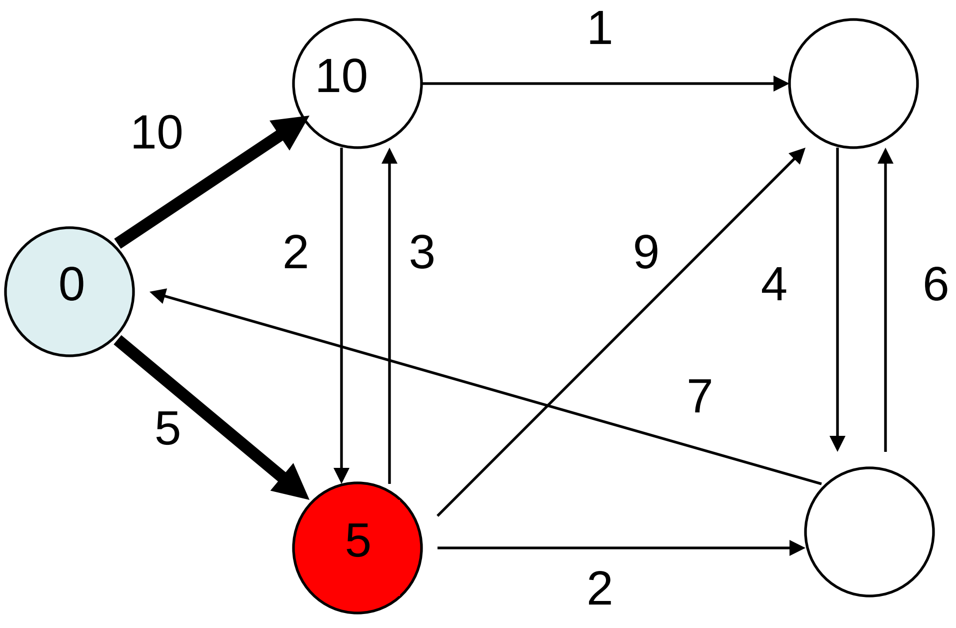
- drop if seen before
- else add to database and forward everywhere but Q

Each LSP will travel over the same link at most once in each direction

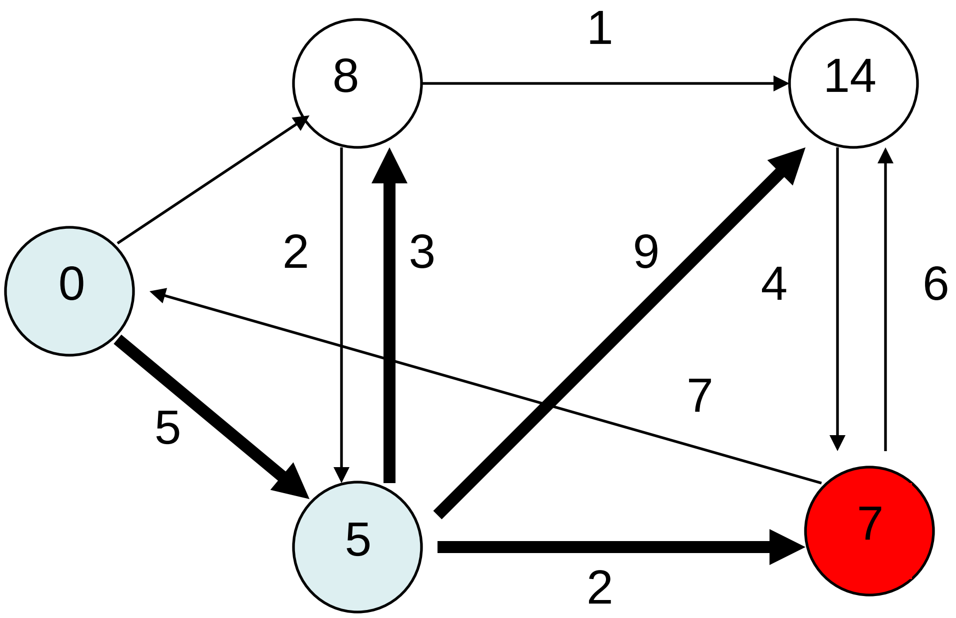
Dijkstra Example – Step 1



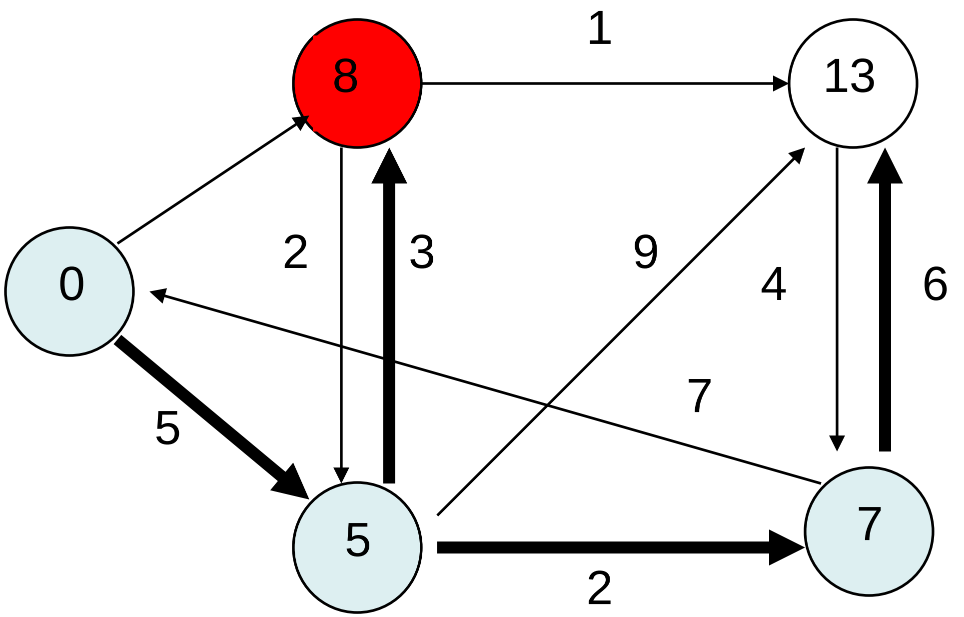
Dijkstra Example – Step 2



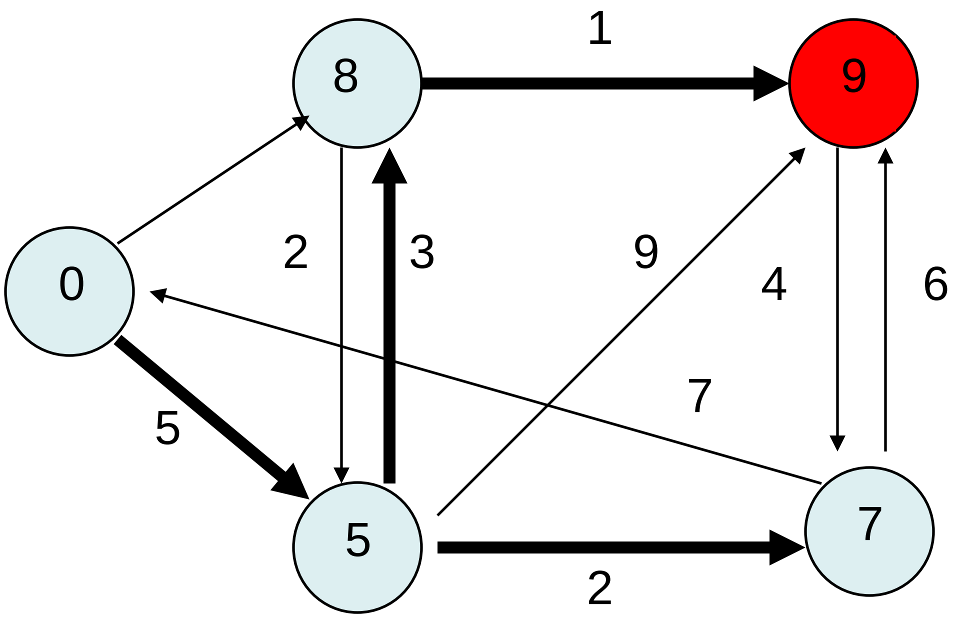
Dijkstra Example – Step 3



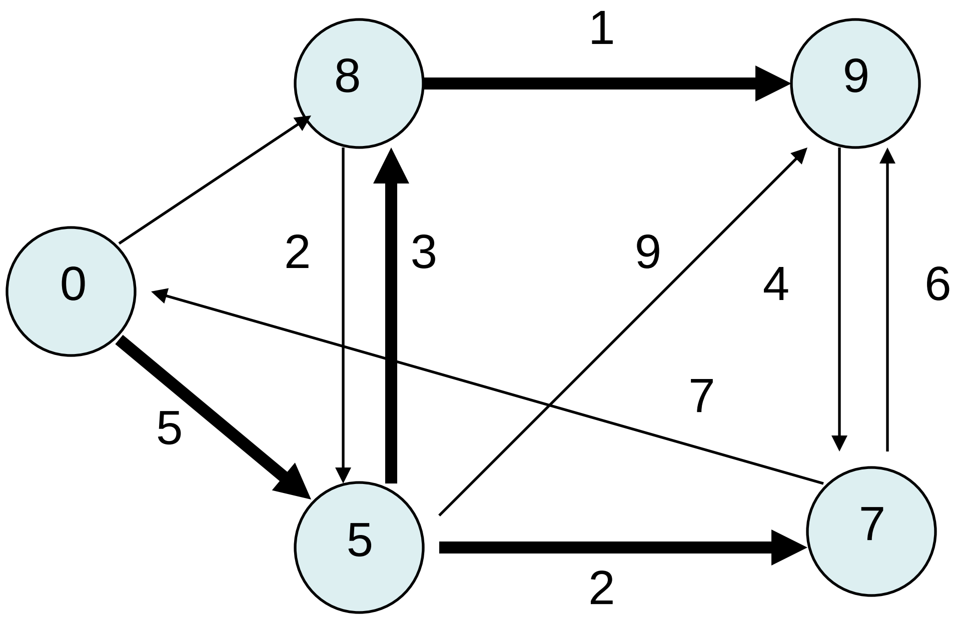
Dijkstra Example – Step 4



Dijkstra Example – Step 5



Dijkstra Example – Done

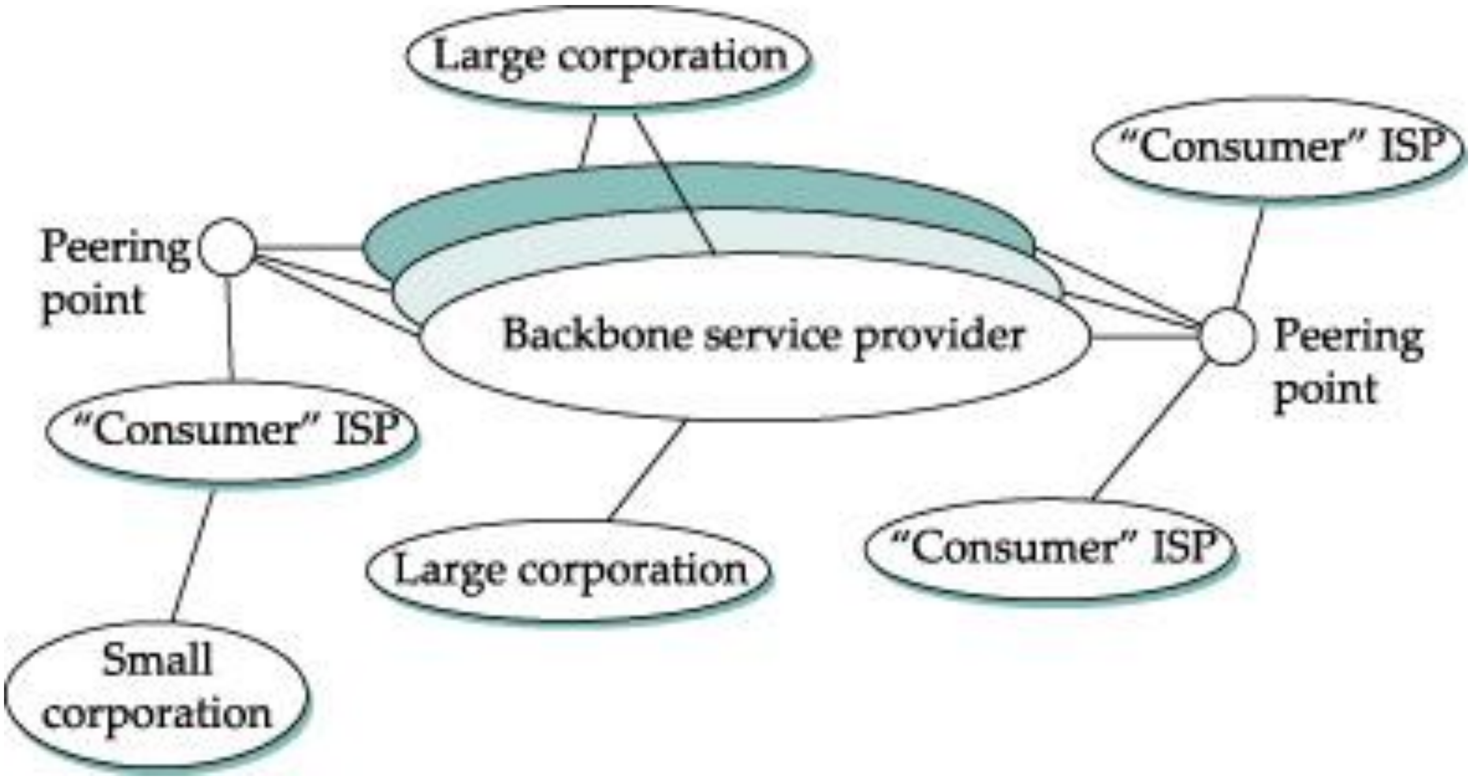


Question

Does link state algorithm guarantee routing tables are loop free?

One proposed solution: failure carrying packets

Internet today



Key goals for Internet routing

Scalability

Support arbitrary policies

- Finding “optimal” paths was less important

(Supporting arbitrary topologies)

Internet routing overview

Two-level hierarchy for scalability

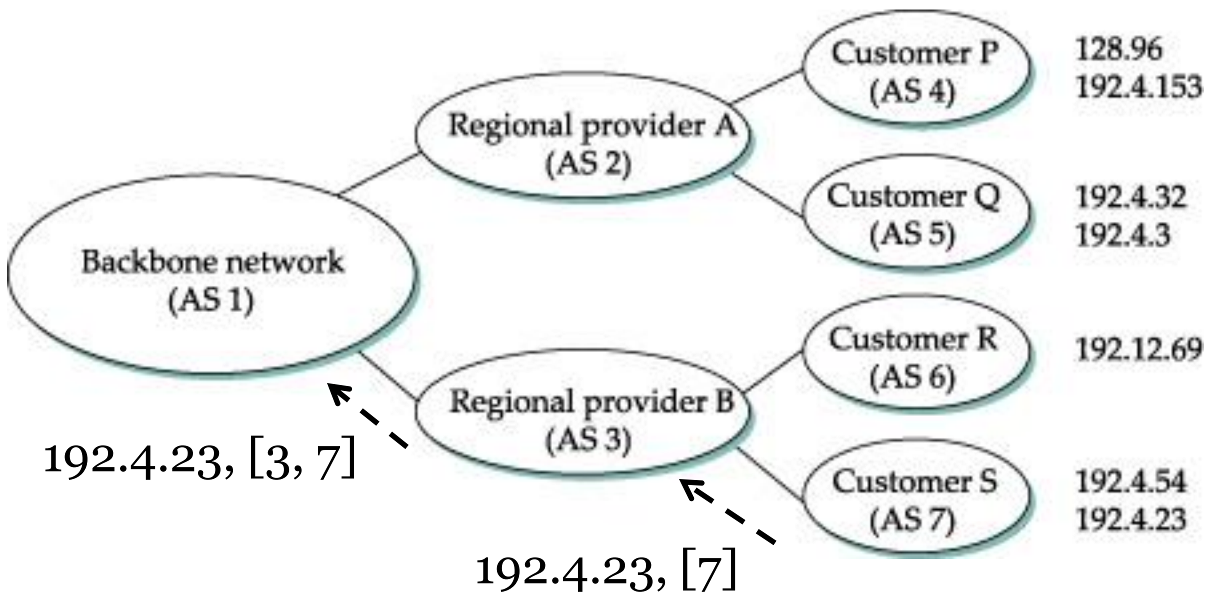
- Intra-domain: within an ISP (OSPF, MPLS)
- Inter-domain: across ISPs (BGP)

Path vector protocol between ASes

- Can support many policies
- Fewer messages in response to small changes
  - Only impacted routers are informed

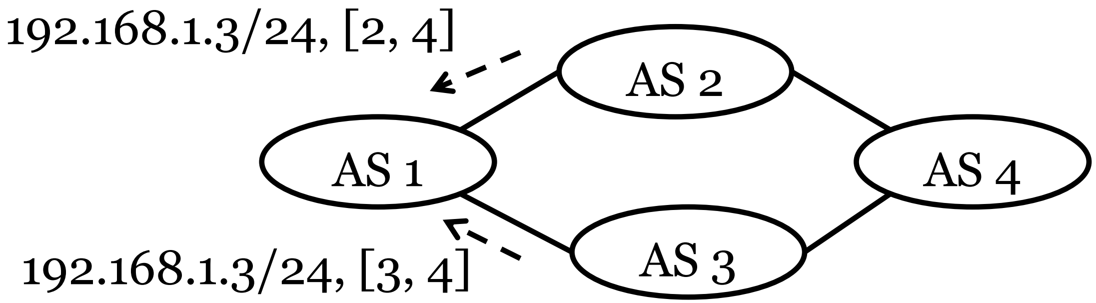
Path vector routing

Similar to distance vector routing info includes entire paths

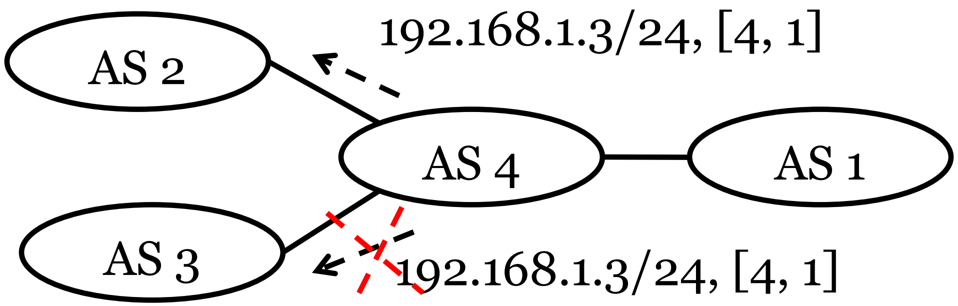


Policy knobs

1. Selecting one of the multiple offered paths



2. Deciding who to offer paths



Typical routing policies

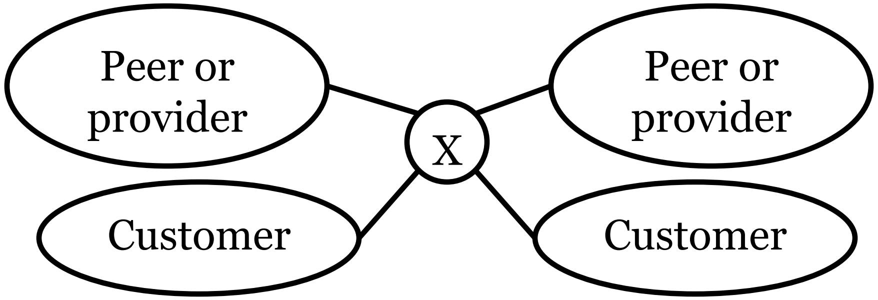
Driven by business considerations

Two common types of relationships between ASes

- Customer-provider: customer pays provider
- Peering: no monetary exchange

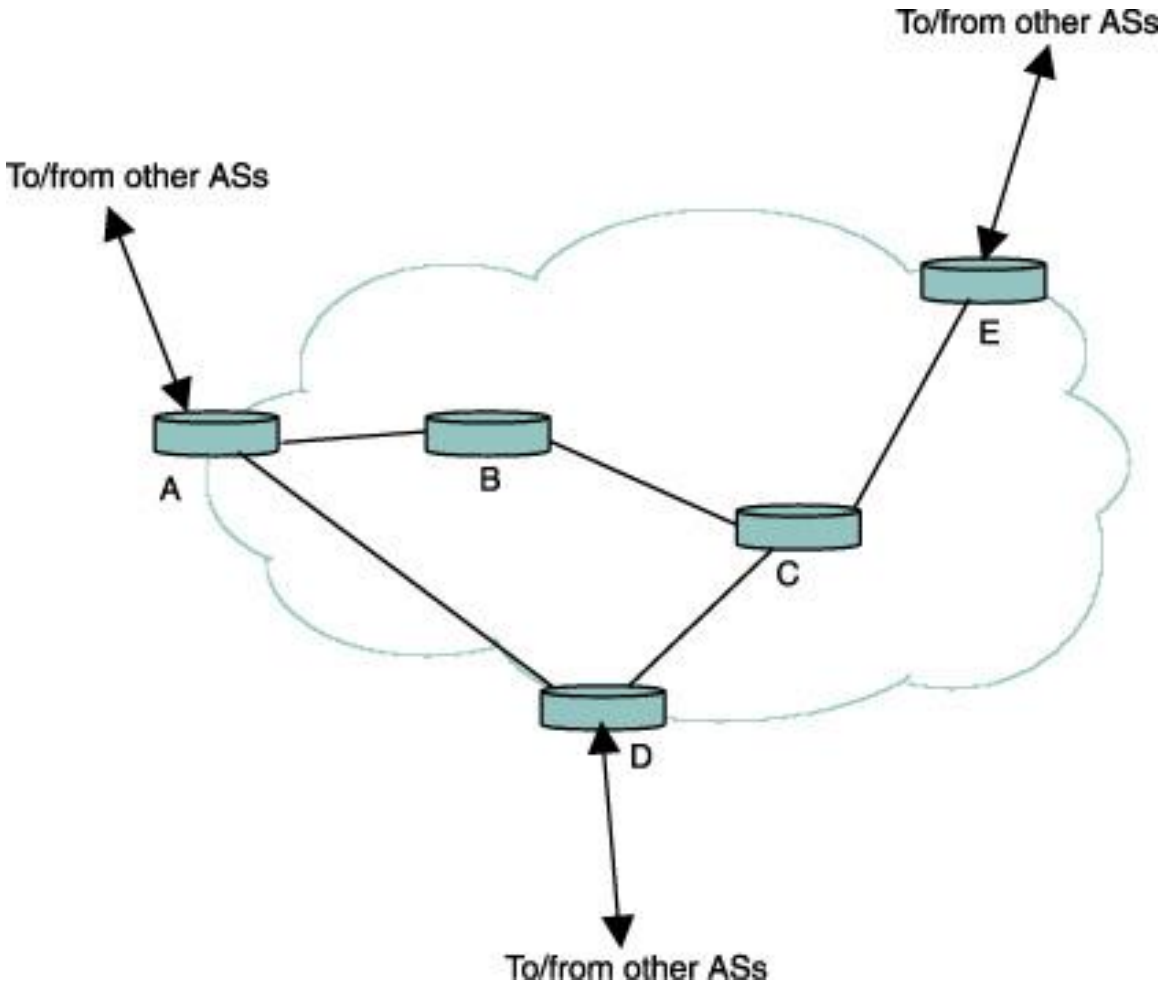
When selecting routes: customer > peer > provider

When exporting routes: do not export provider or peer routes to other providers and peers



Prefer routes with shorter AS paths

BGP at router level

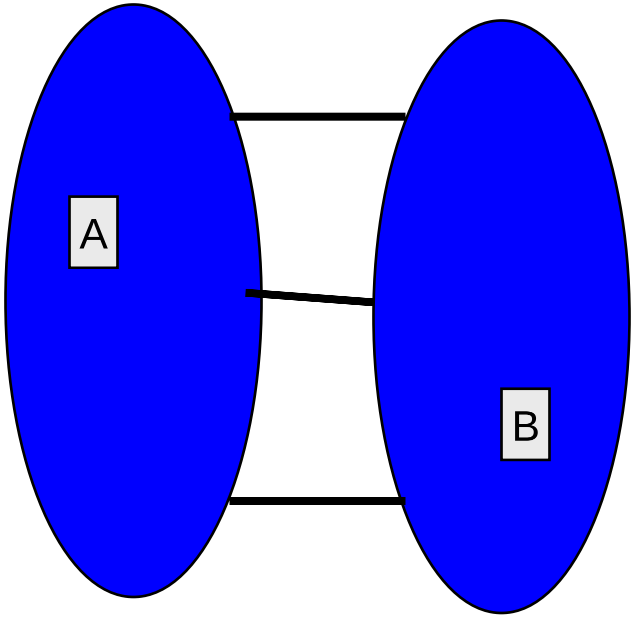


Path quality with BGP

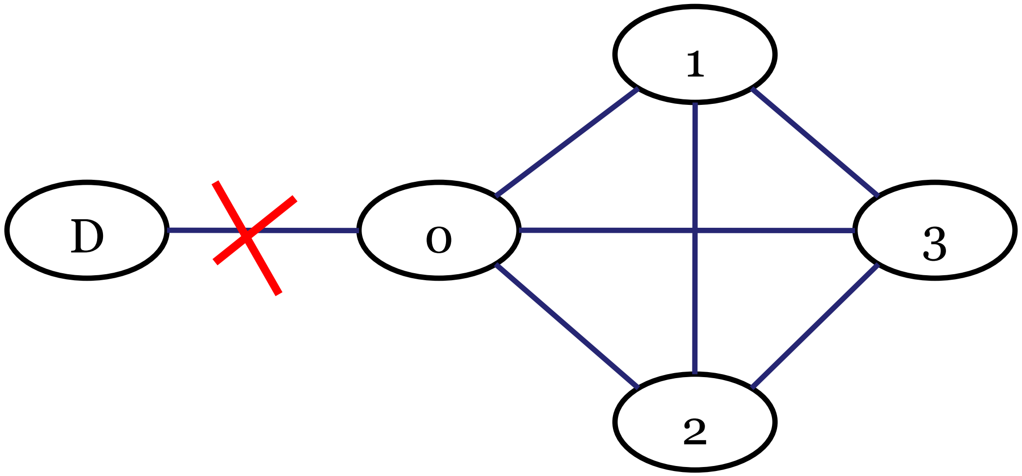
Combination of local policies may not be globally good

- Longer paths, asymmetric paths
- Shorter "detours" are often available

Example:
hot potato routing



BGP convergence



Temporary loops during path exploration

Can occur after failures, or after policy changes

BGP Convergence

Why not link state routing in BGP?

Proposed solution: consensus routing

BGP security

Extreme vulnerability to attacks and misconfigurations

- An AS can announce reachability to any prefix
- An AS can announce connectivity to other Ases

Many known incidents

- AS7007 brought down the whole internet in 1997
- 75% of new route adverts are due to misconfigs [SIGCOMM 2002]
- Commonly used for spamming

Technical solutions exist but none even close to deployment

- Incentives and deployability

Transport Challenge

IP: routers can be arbitrarily bad

- packets can be lost, reordered, duplicated, have limited size & can be fragmented

TCP: applications need something better

- reliable delivery, in order delivery, no duplicates, arbitrarily long streams of data, match sender/receiver speed, process-to-process

Reliable Transmission

How do we send packets reliably?

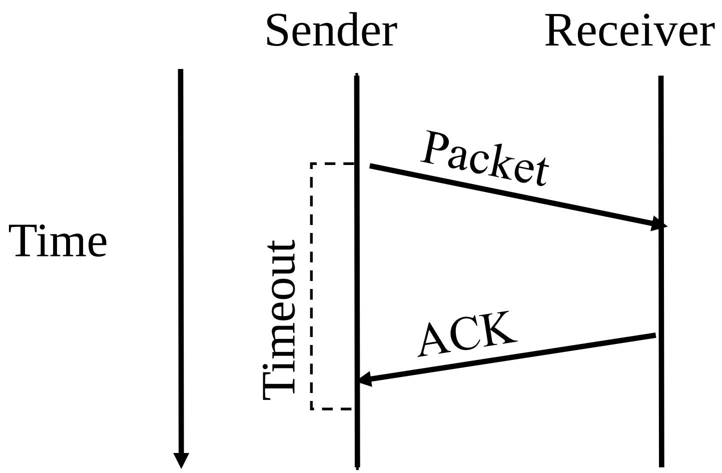
Two mechanisms

- Acknowledgements
- Timeouts

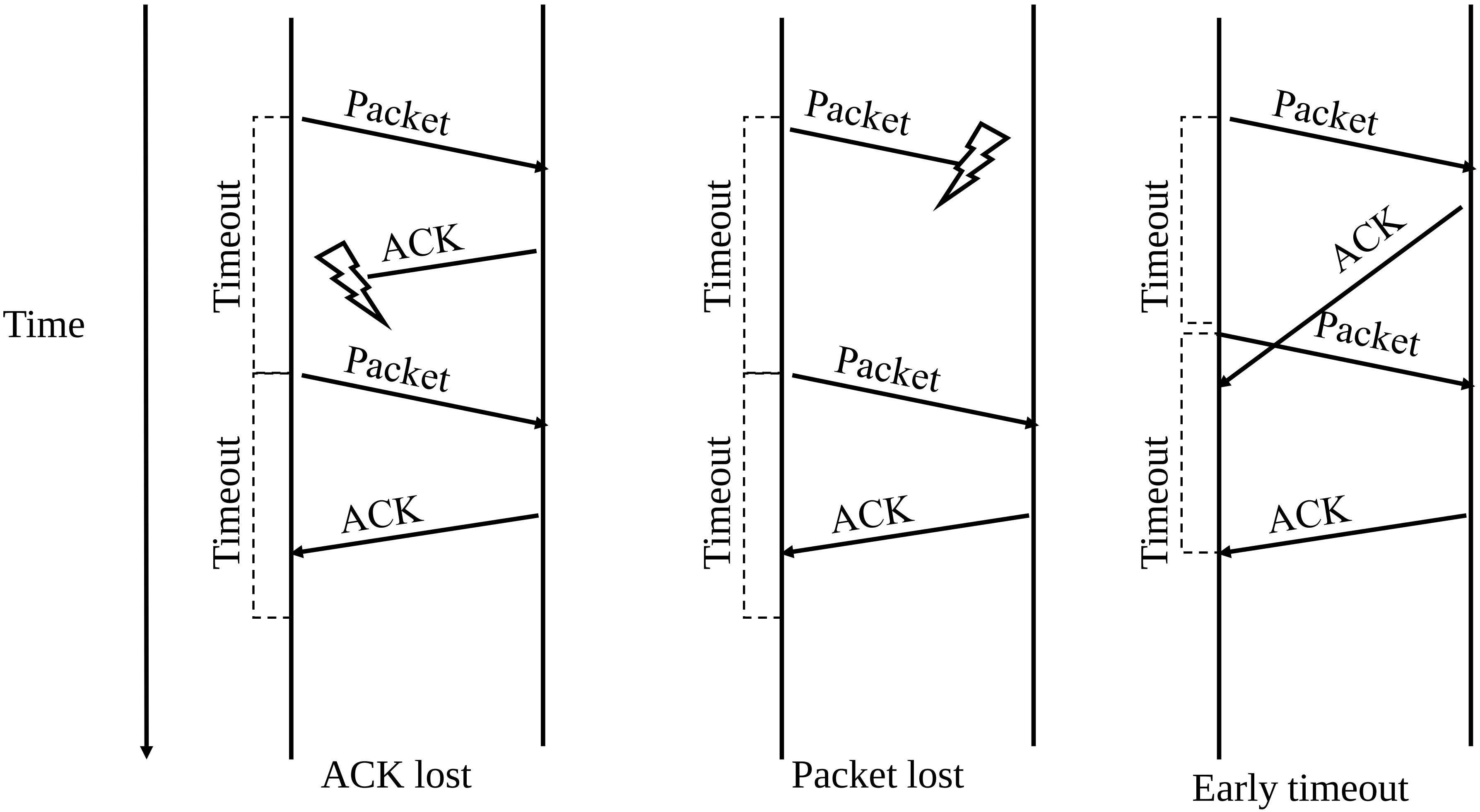
Simplest reliable protocol: Stop and Wait

Stop and Wait

- Send a packet, wait until ack arrives
  - retransmit if no ack within timeout
- Receiver acks each packet as it arrives



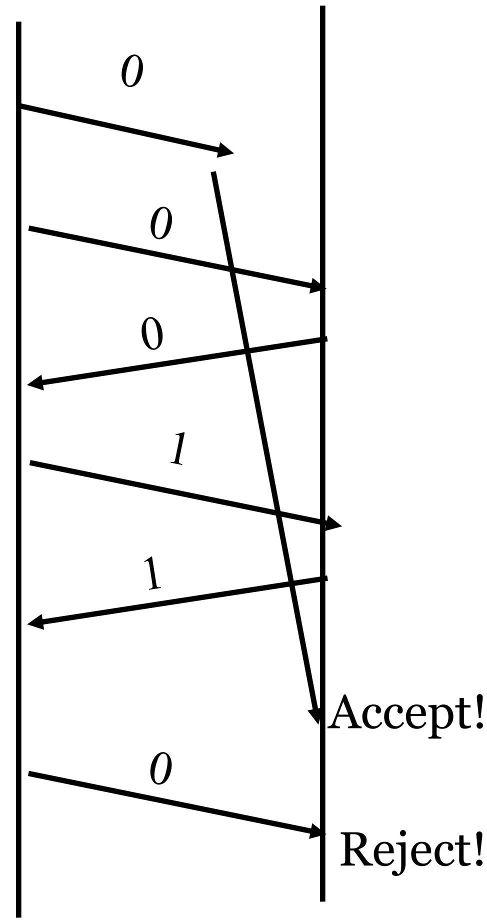
Recovering from error



What if packets can be delayed?

Solutions?

- Never reuse an ID?
- Change IP layer to eliminate packet reordering?
- Prevent very late delivery?
  - IP routers keep hop count per pkt, discard if exceeded
  - ID's not reused within delay bound
- TCP won't work without some bound on how late packets can arrive!



How do we keep the pipe full?

Unless the bandwidth*delay product is small, stop and wait can't fill pipe

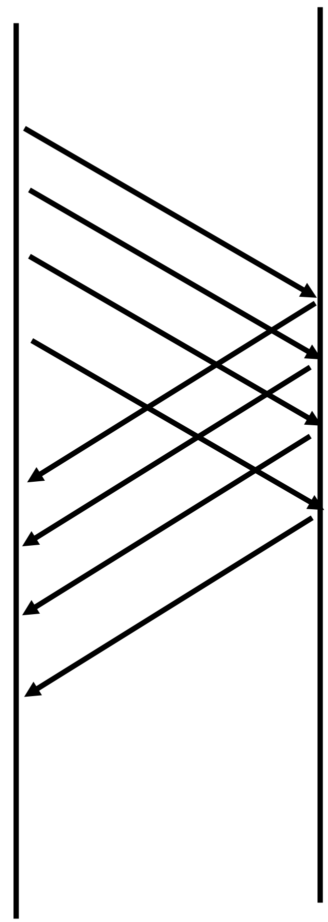
Solution: Send multiple packets without waiting for first to be acked

Reliable, unordered delivery:

- Send new packet after each ack
- Sender keeps list of unack'ed packets; resends after timeout
- Receiver same as stop&wait

How easy is it to write apps that handle out of order delivery?

- How easy is it to test those apps?



Sliding Window: Reliable, ordered delivery

Two constraints:

- Receiver can't deliver packet to application until all prior packets have arrived
- Sender must prevent buffer overflow at receiver

Solution: sliding window

- circular buffer at sender and receiver
  - packets in transit <= buffer size
  - advance when sender and receiver agree packets at beginning have been received
- How big should the window be?
  - bandwidth * round trip delay

Sender/Receiver State

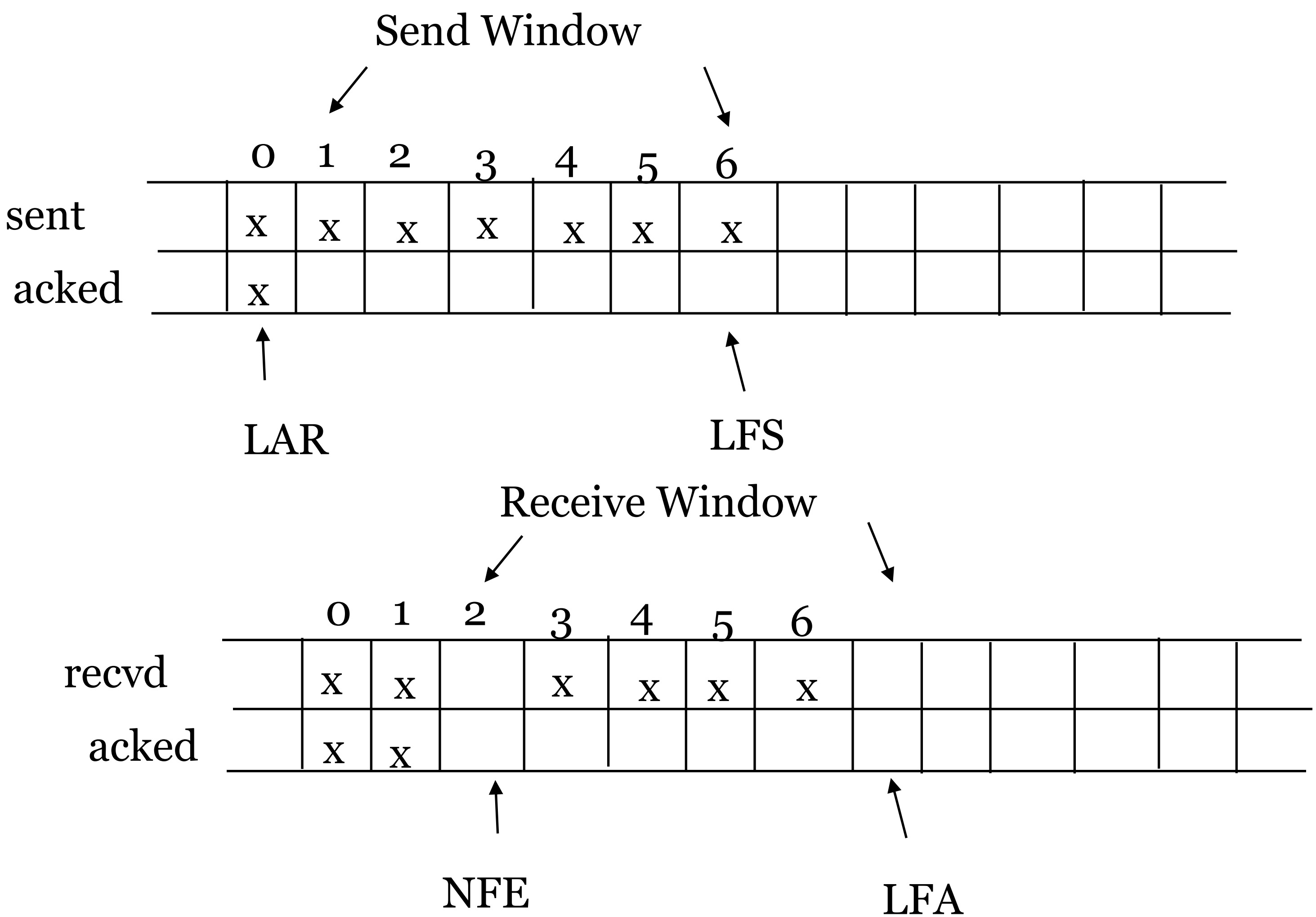
sender

- packets sent and acked (LAR = last ack recvd)
- packets sent but not yet acked
- packets not yet sent (LFS = last frame sent)

receiver

- packets received and acked (NFE = next frame expected)
- packets received out of order
- packets not yet received (LFA = last frame ok)

Sliding Window



What if we lose a packet?

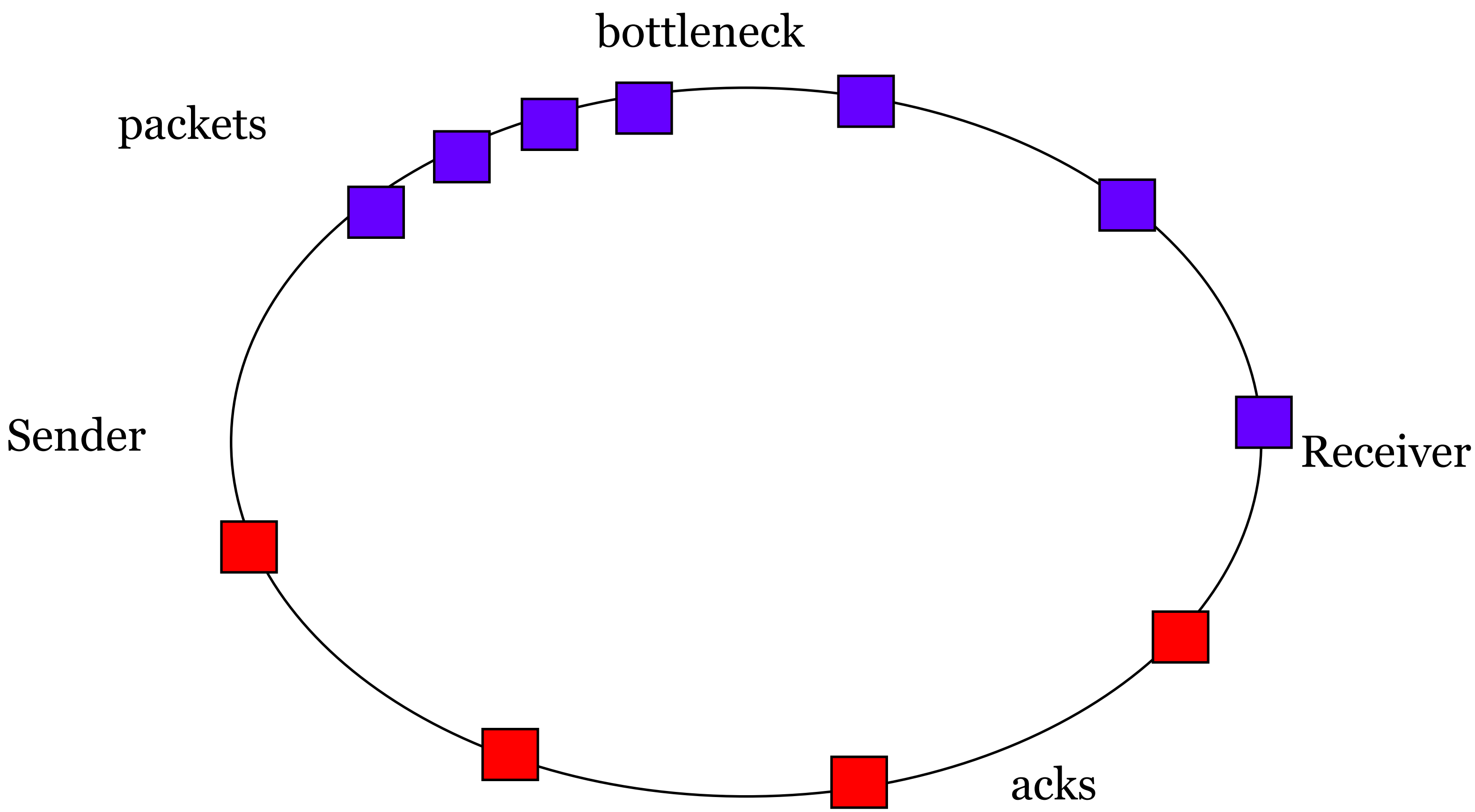
Go back N (original TCP)

- receiver acks “got up through k” (“cumulative ack”)
- ok for receiver to buffer out of order packets
- on timeout, sender restarts from k+1

Selective retransmission (RFC 2018)

- receiver sends ack for each pkt in window
- on timeout, resend only missing packet

Avoiding burstiness: ack pacing



Window size = round trip delay * bit rate

Transport: Practice

Protocols

- IP -- Internet protocol
- UDP -- user datagram protocol
- TCP -- transmission control protocol
- RPC -- remote procedure call
- HTTP -- hypertext transfer protocol
- And a bunch more...

How do we connect processes?

IP provides host to host packet delivery

- header has source, destination IP address

For applications to communicate, need to demux packets sent to host to target app

- Web browser (HTTP), Email servers (SMTP), hostname translation (DNS), RealAudio player (RTSP), etc.
- Process id is OS-specific and transient

Ports

- Port is a mailbox that processes "rent"
  - Uniquely identify communication endpoint as (IP address, protocol, port)
- How do we pick port #'s?
  - Client needs to know port # to send server a request
  - Servers bind to "well-known" port numbers
    - Ex: HTTP 80, SMTP 25, DNS 53, …
    - Ports below 1024 reserved for "well-known" services
  - Clients use OS-assigned temporary (ephemeral) ports
    - Above 1024, recycled by OS when client finished

Sockets

- OS abstraction representing communication endpoint
  - Layer on top of TCP, UDP, local pipes
- server (passive open)
  - bind -- socket to specific local port
  - listen -- wait for client to connect
- client (active open)
  - connect -- to specific remote port

User Datagram Protocol (UDP)

Provides application – application delivery

Header has source & dest port #'s

- IP header provides source, dest IP addresses

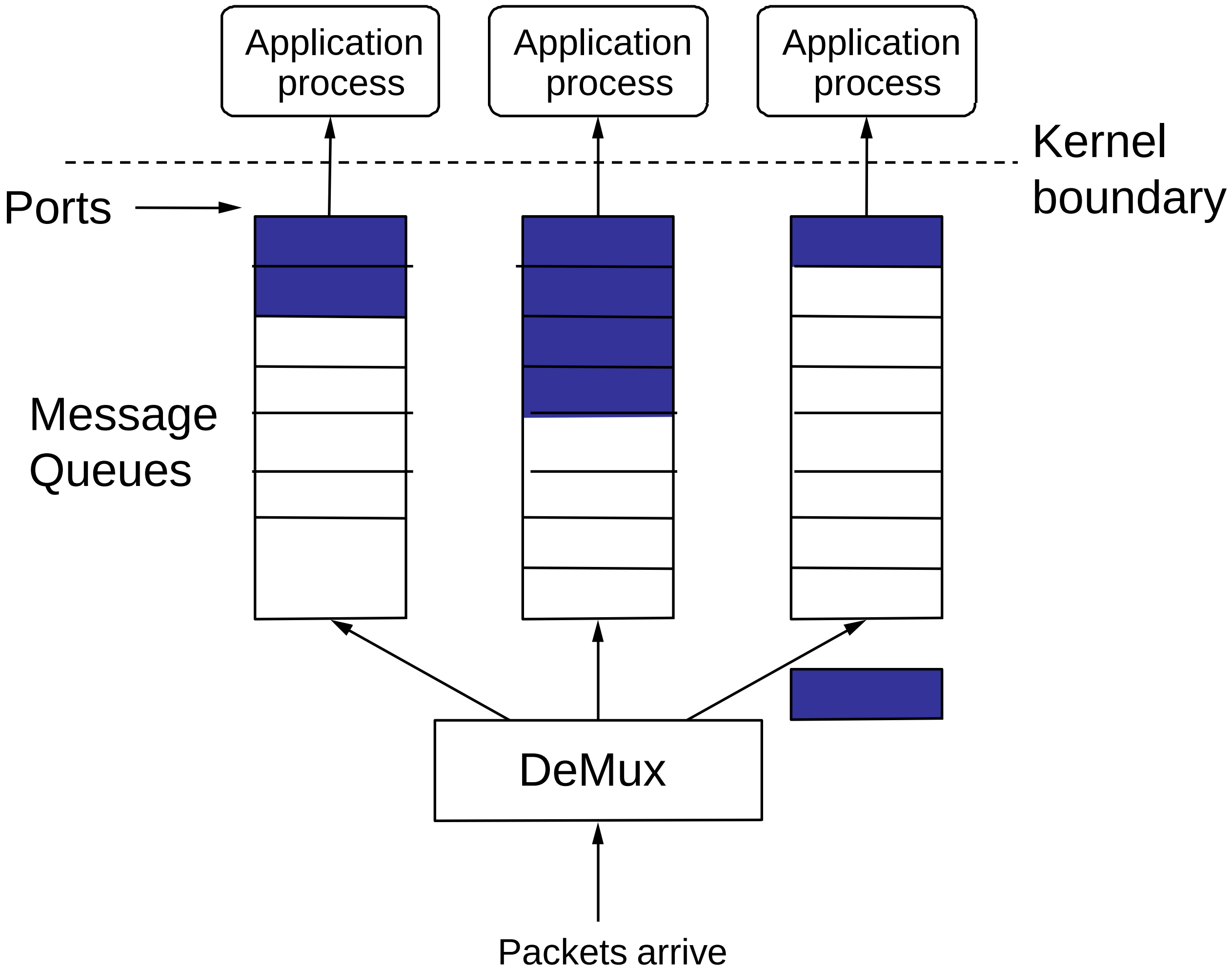
Deliver to destination port on dest machine

Reply returns to source port on source machine

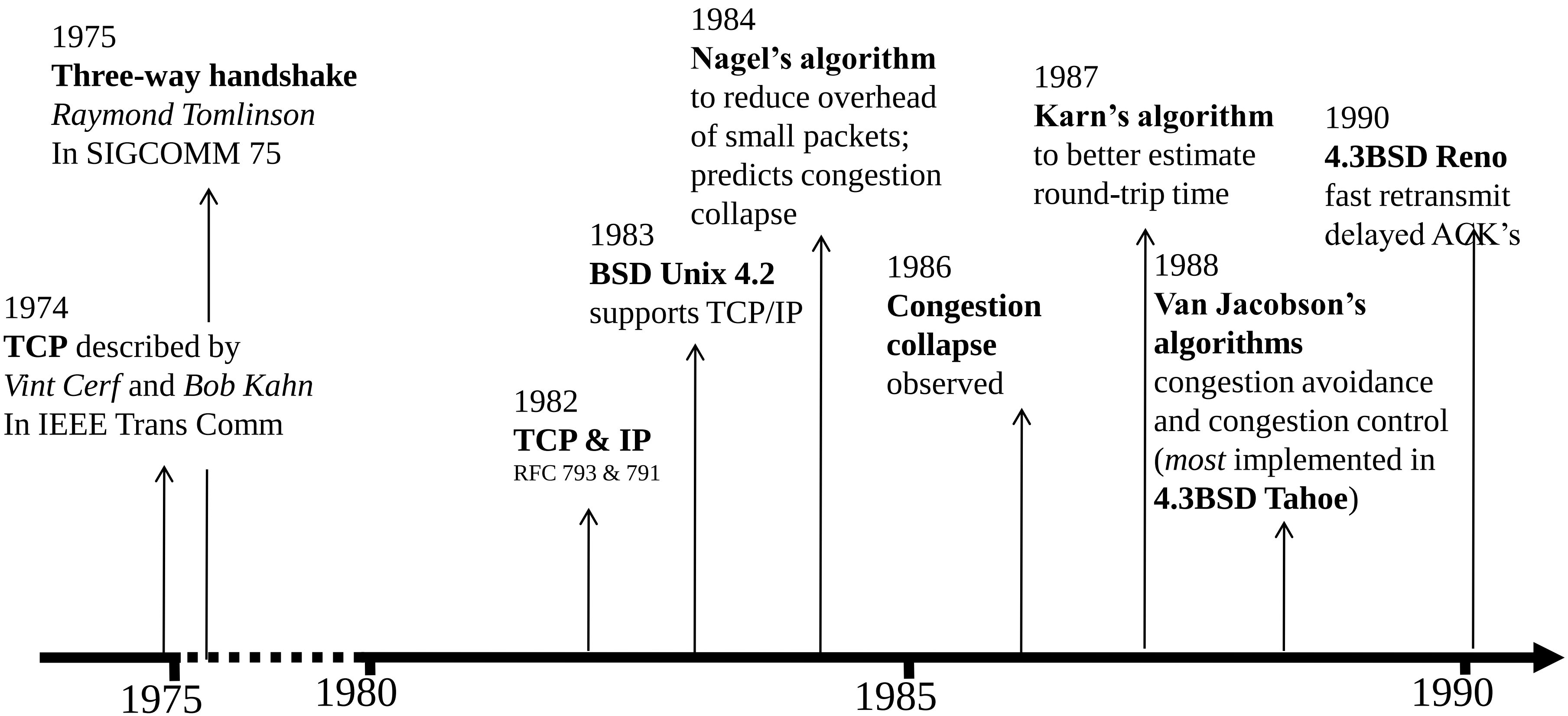
No retransmissions, no sequence #s

=> stateless

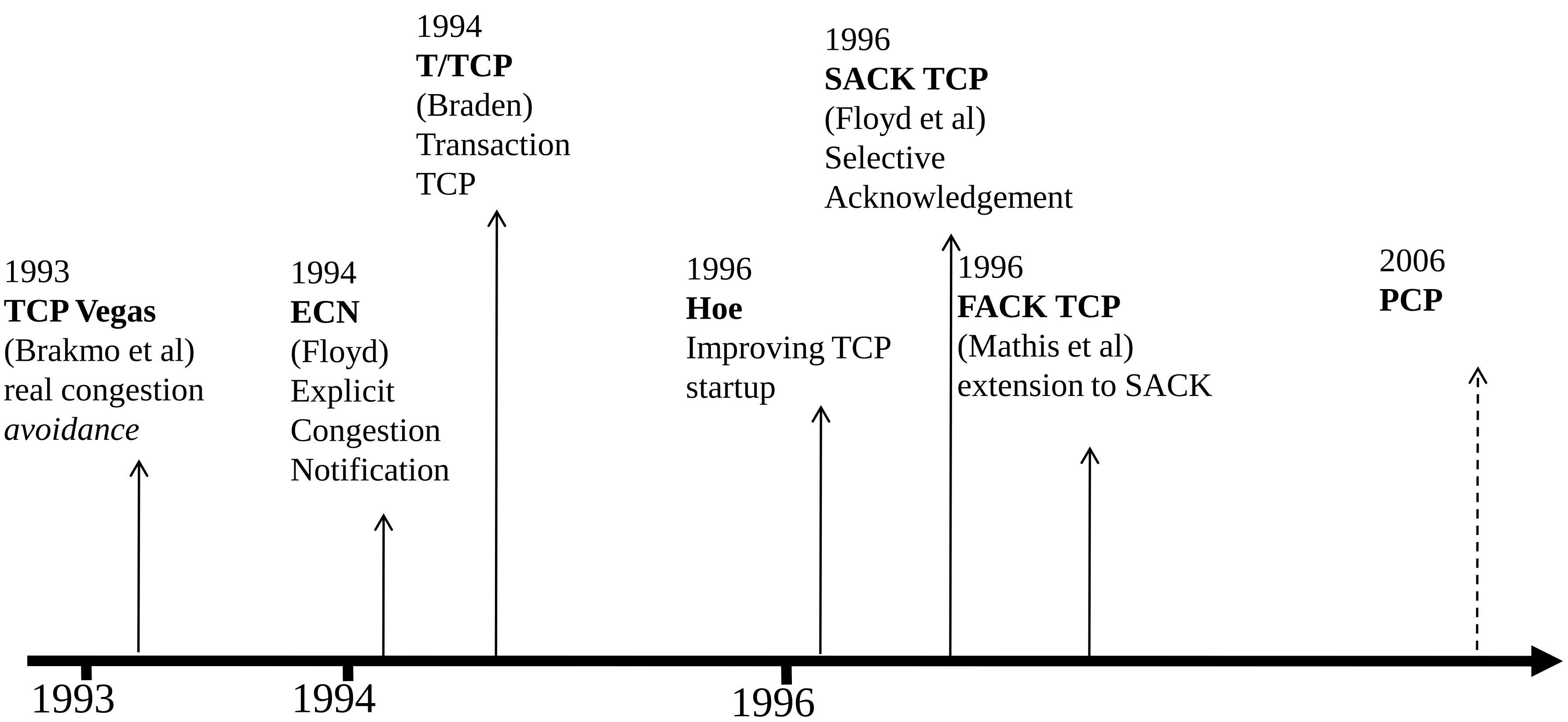
UDP Delivery



TCP: This is your life...



TCP: After 1990



Transmission Control Protocol (TCP)

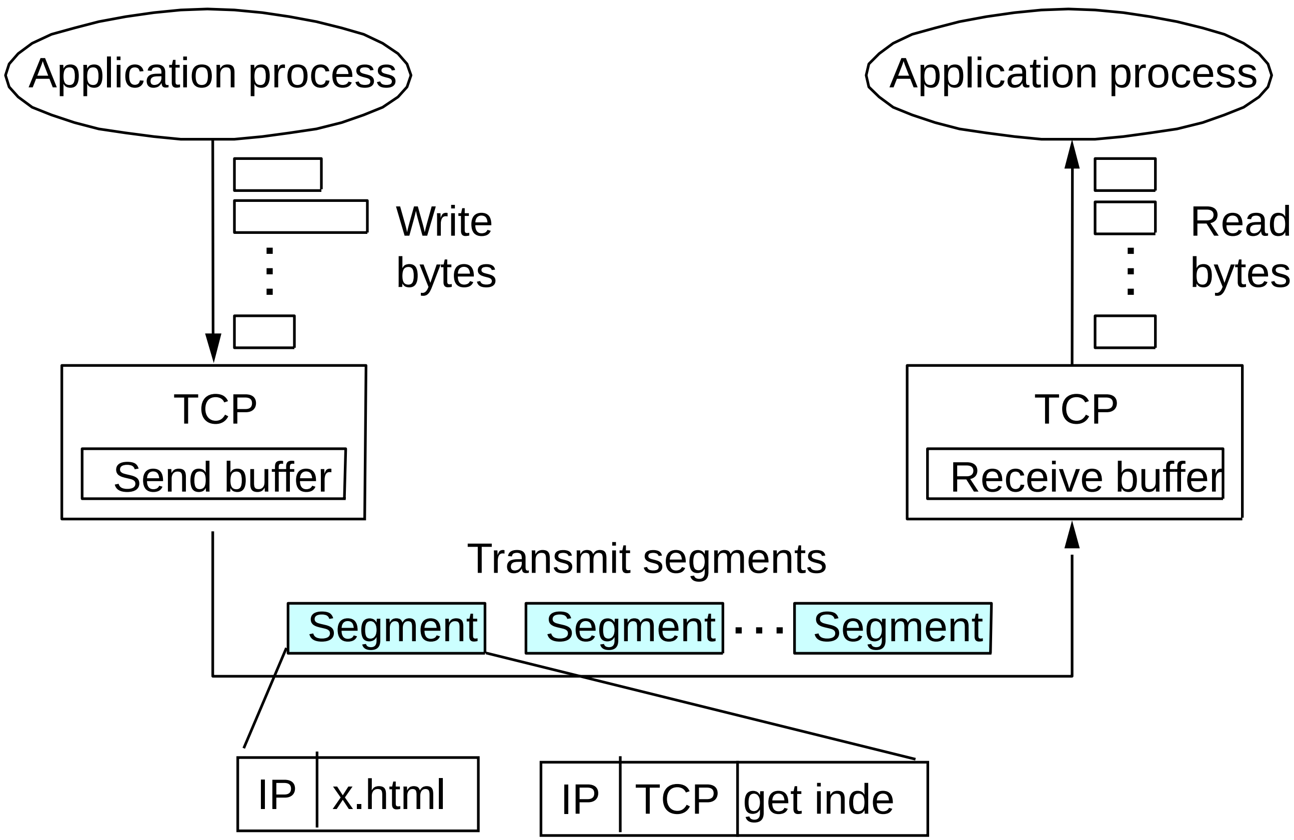
Reliable bi-directional byte stream
- No message boundaries
- Ports as application endpoints

Sliding window, go back N/SACK, RTT est, …
- Highly tuned congestion control algorithm

Flow control
- prevent sender from overrunning receiver buffers

Connection setup
- negotiate buffer sizes and initial seq #s
- Needs to work between all types of computers (supercomputer -> 8086)

TCP Delivery



TCP Sliding Window

Per-byte, not per-packet (why?)

- send packet says "here are bytes j-k"
- ack says "received up to byte k"

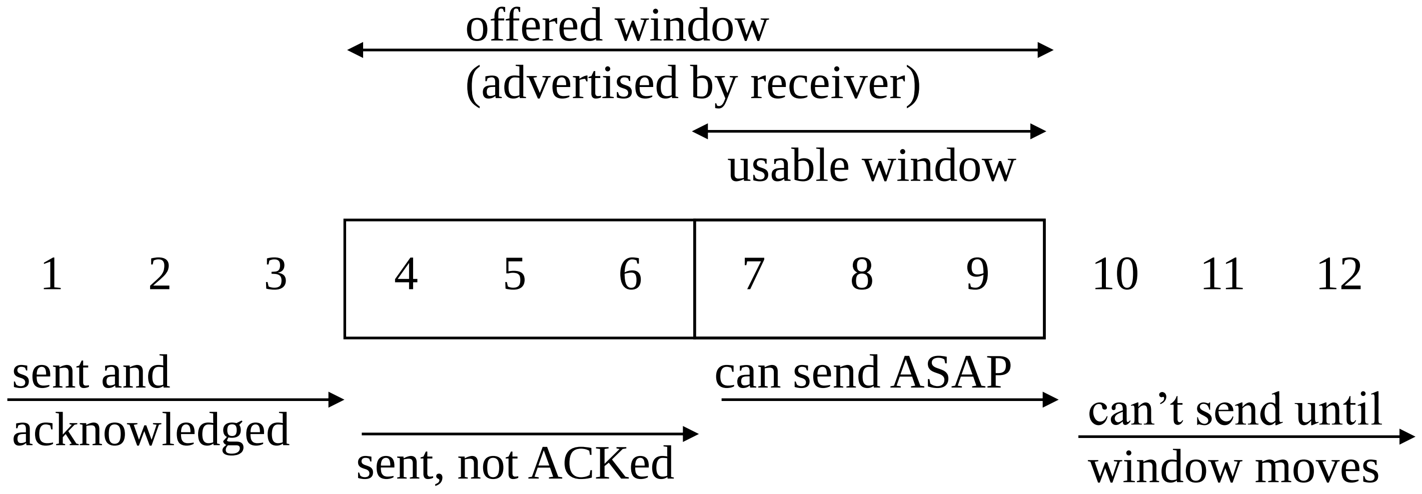
Send buffer >= send window

- can buffer writes in kernel before sending
- writer blocks if try to write past send buffer

Receive buffer >= receive window

- buffer acked data in kernel, wait for reads
- reader blocks if try to read past acked data

Visualizing the window

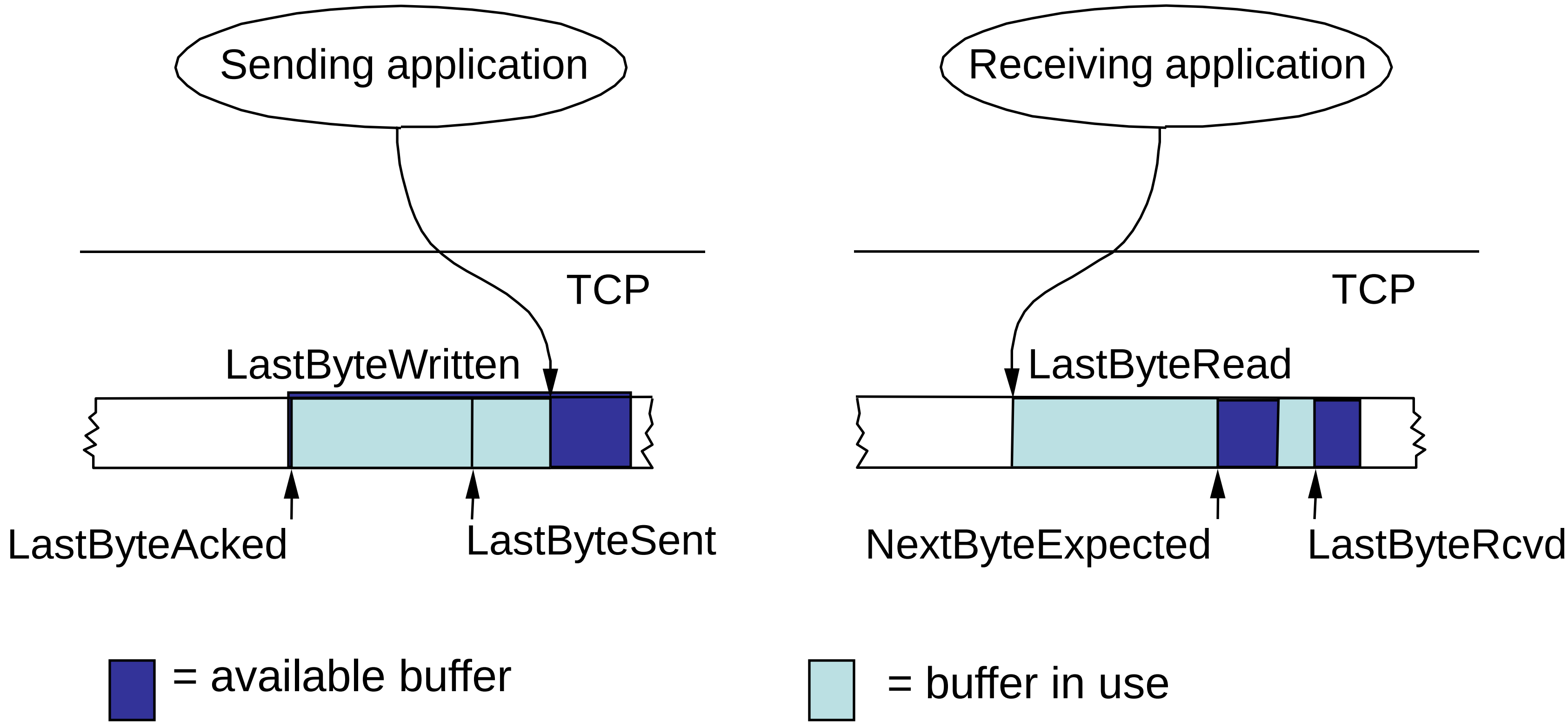


Left side of window advances when data is acknowledged.
Right side controlled by size of window advertisement

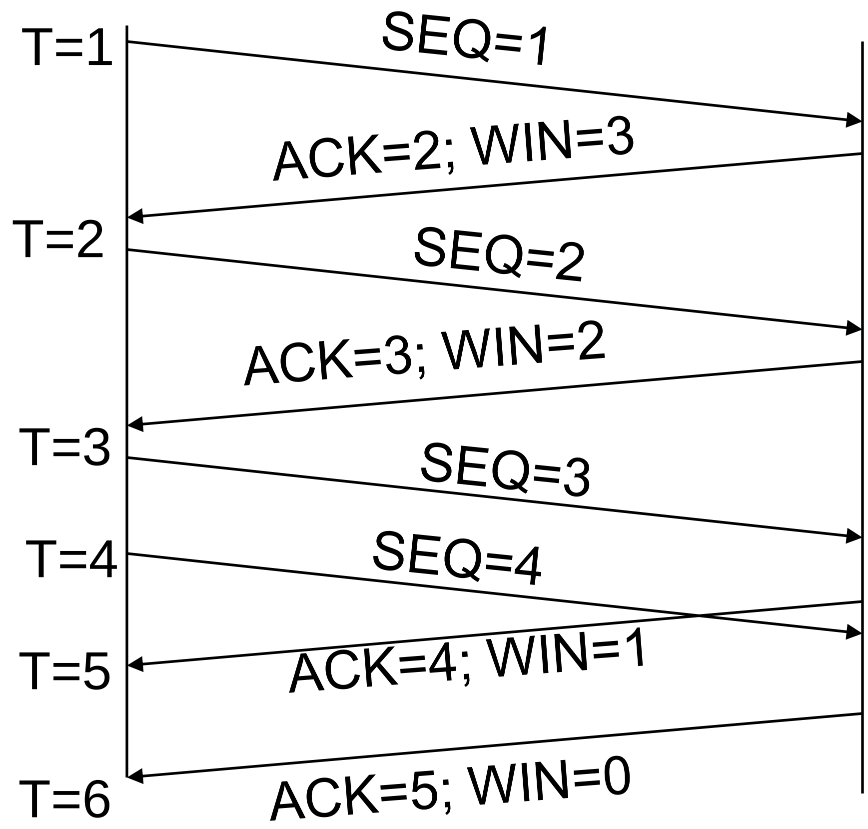
Flow Control

- What if sender process is faster than receiver process?
  - Data builds up in receive window
  - if data is acked, sender will send more!
  - If data is not acked, sender will retransmit!
- Sender must transmit data no faster than it can be consumed by the receiver
  - Receiver might be a slow machine
  - App might consume data slowly
- Sender sliding window <= free receiver buffer
  - Advertised window = # of free bytes; if zero, stop

Sender and Receiver Buffering



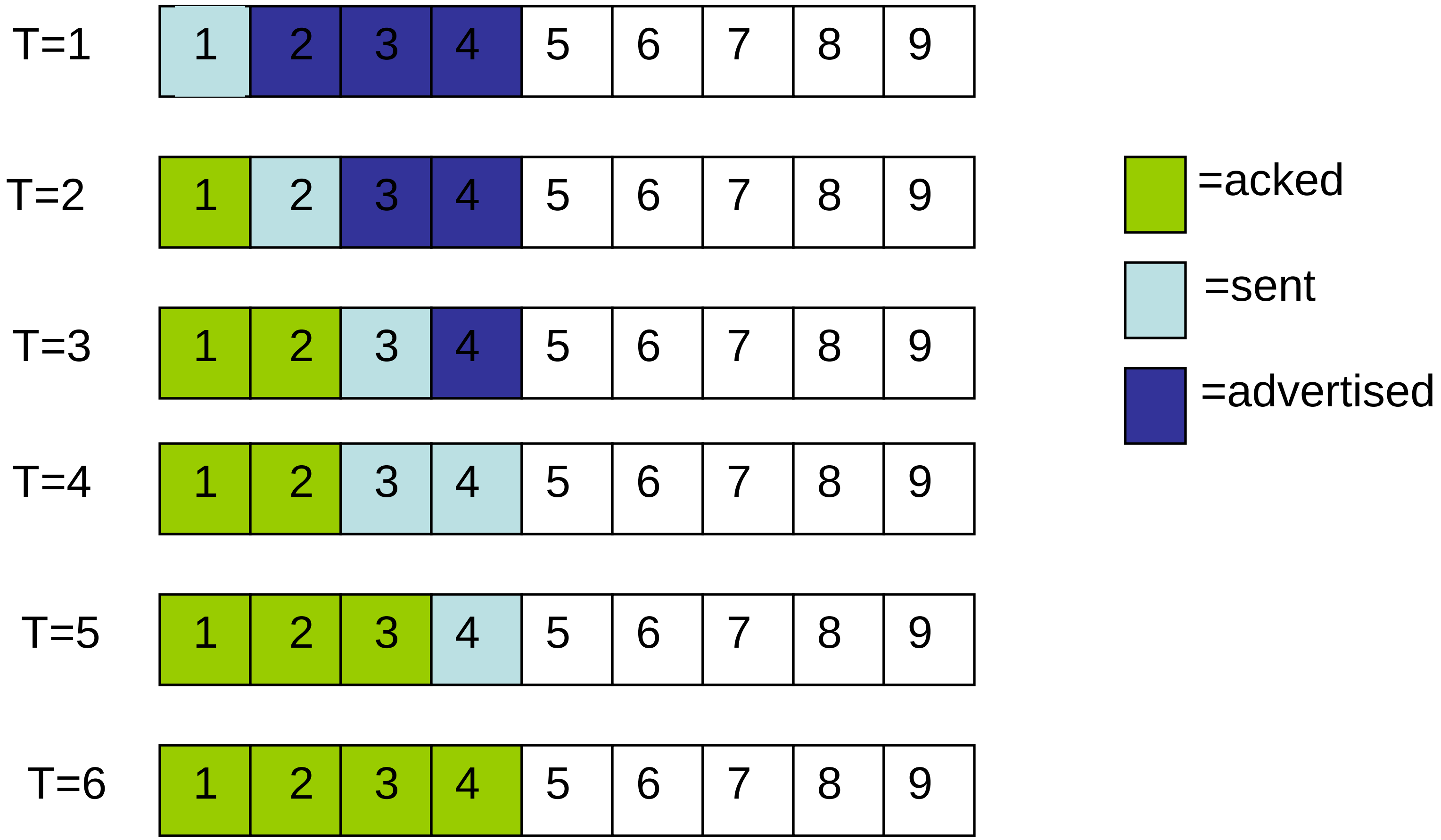
Example – Exchange of Packets



Receiver has buffer of size 4 and application doesn't read

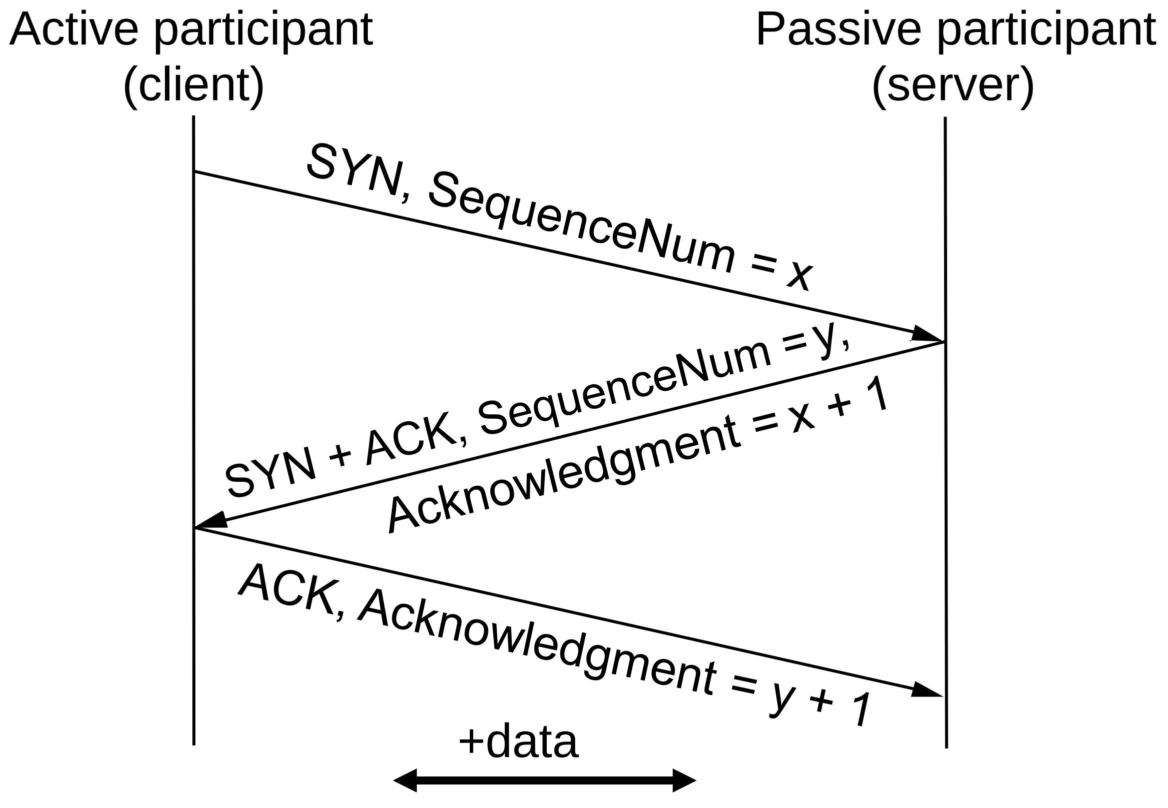
Stall due to flow control here

Example – Buffer at Sender



Three-Way Handshake

Opens both directions for transfer



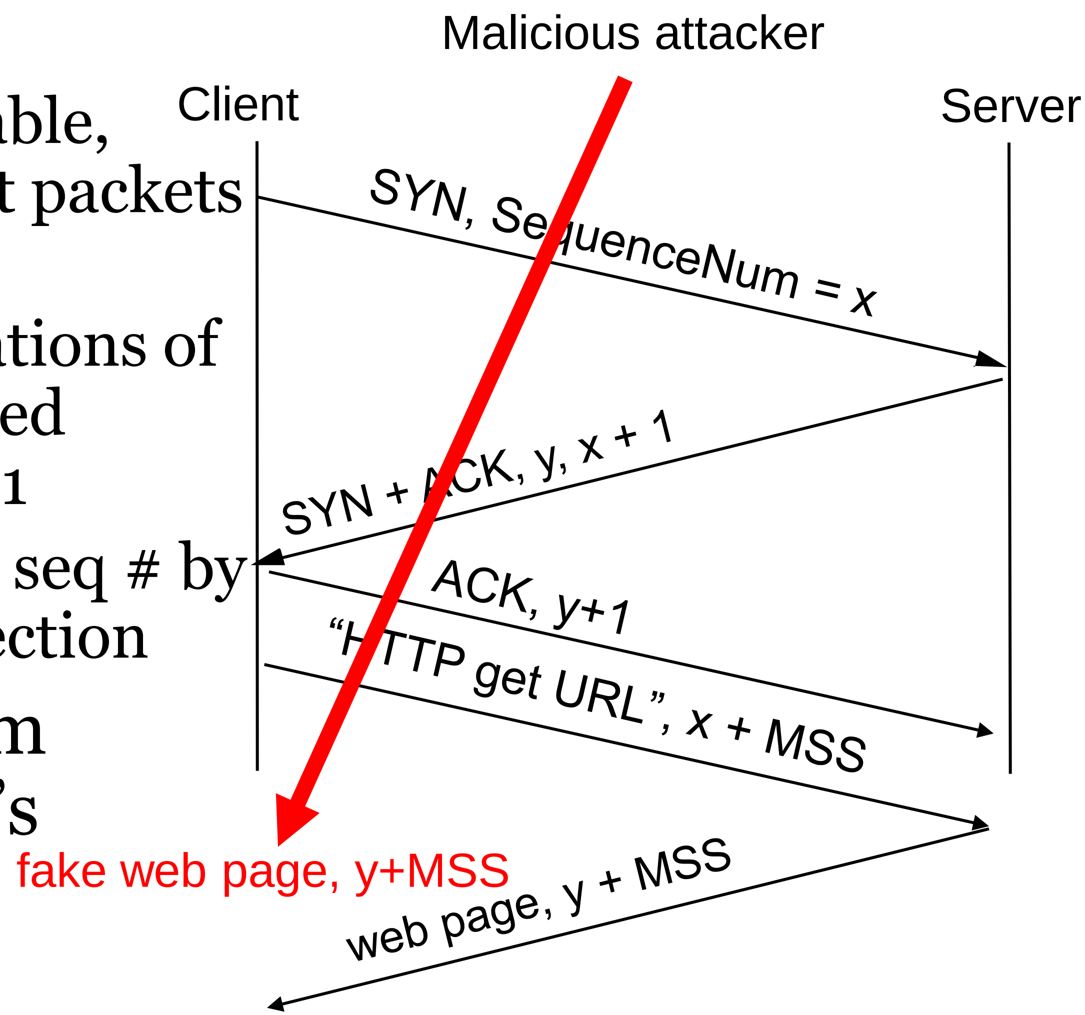
TCP Handshake in an Uncooperative Internet

TCP Hijacking

- if seq # is predictable, attacker can insert packets into TCP stream
- many implementations of TCP simply bumped previous seq # by 1
- attacker can learn seq # by setting up a connection

Solution: use random initial sequence #'s

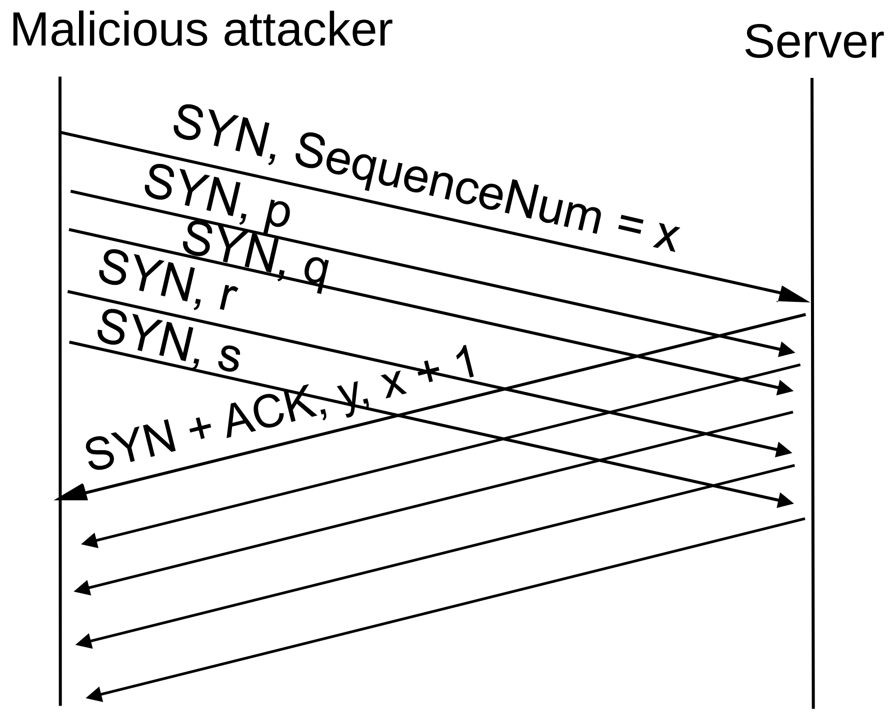
- weak form of authentication



TCP Handshake in an Uncooperative Internet

TCP SYN flood

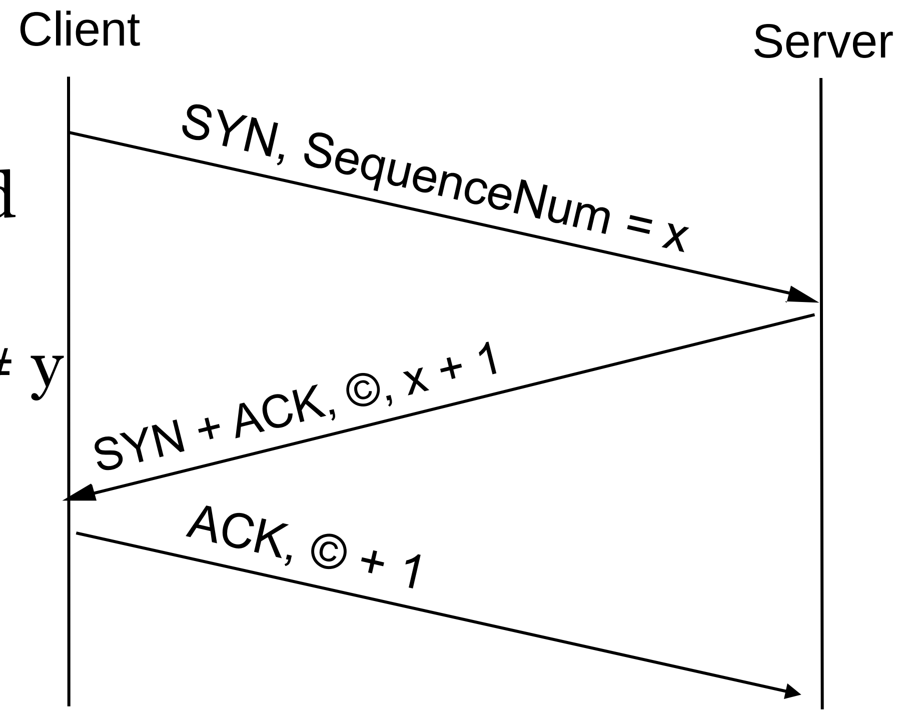
- server maintains state for every open connection
- if attacker spoofs source addresses, can cause server to open lots of connections
- eventually, server runs out of memory



TCP SYN cookies

Solution: SYN cookies

- Server keeps no state in response to SYN; instead makes client store state
- Server picks return seq # y = © that encrypts x
- Gets © +1 from sender; unpacks to yield x



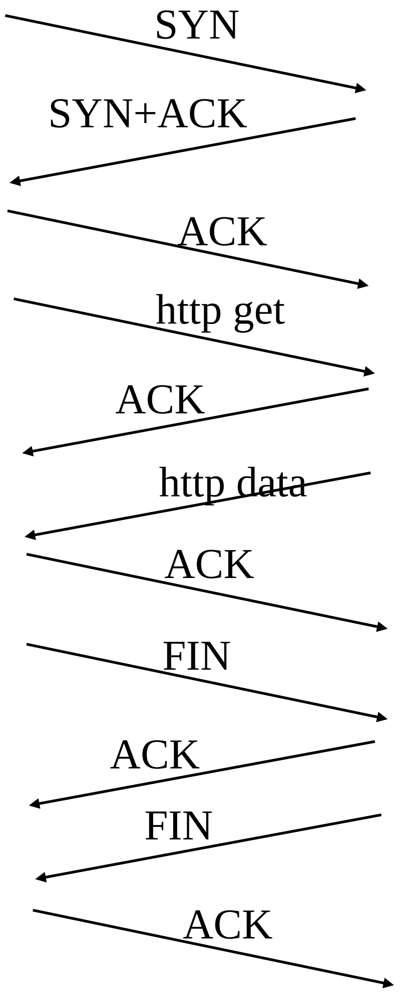
Can data arrive before ACK?

HTTP on TCP

How do we reduce the # of messages?

Delayed ack: wait for 200ms for reply or another pkt arrival

TCP RST from web server

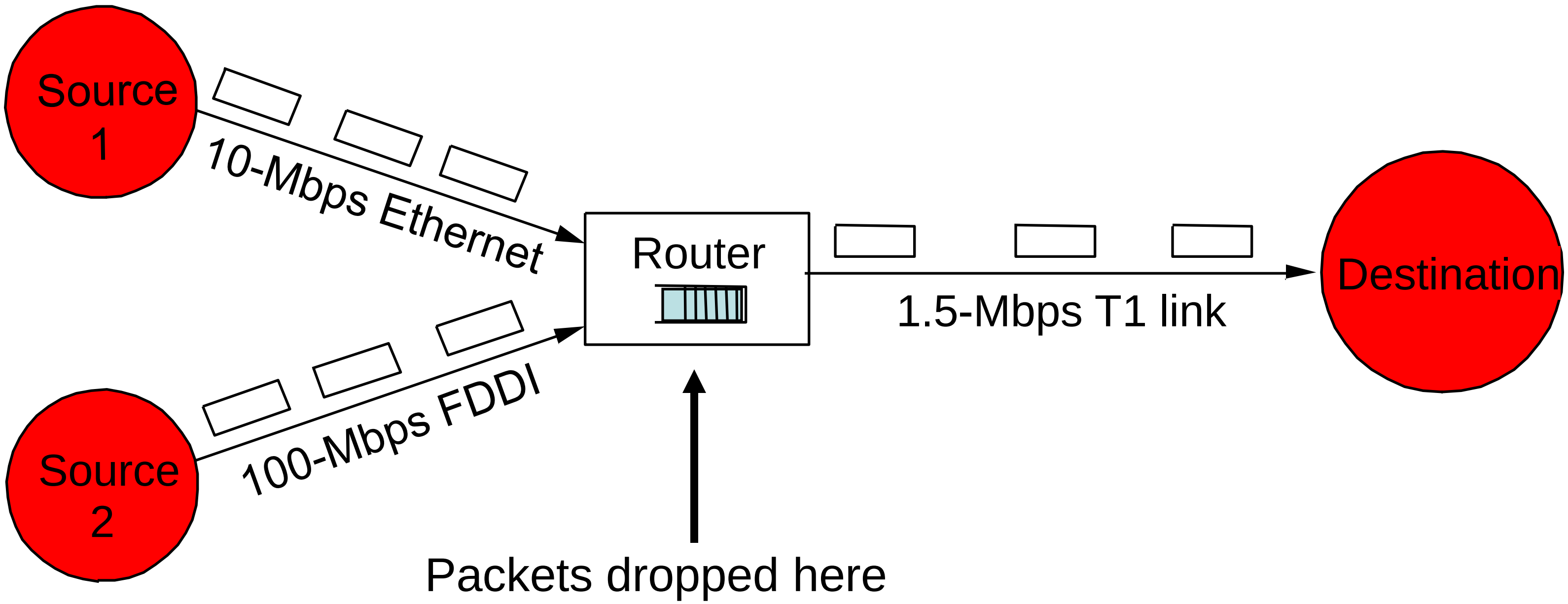


Bandwidth Allocation

How do we efficiently share network resources among billions of hosts?

- Congestion control
  - Sending too fast causes packet loss inside network -> retransmissions -> more load -> more packet losses -> …
  - Don't send faster than network can accept
- Fairness
  - How do we allocate bandwidth among different users?
  - Each user should (?) get fair share of bandwidth

Congestion



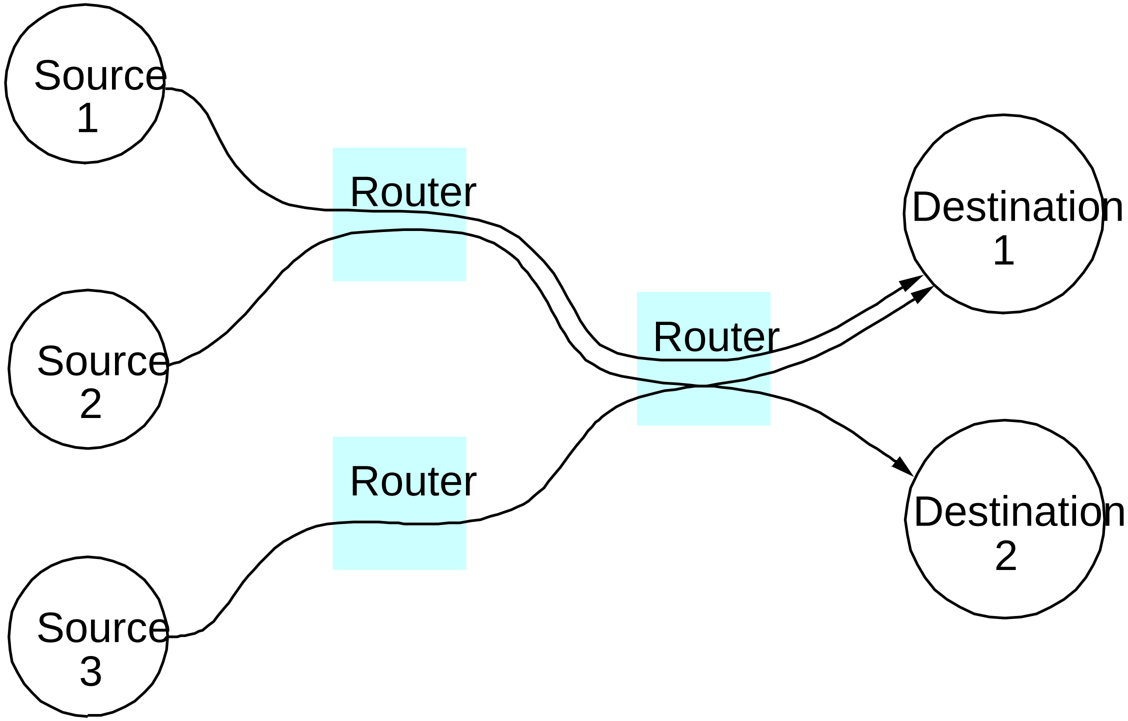
Buffer absorbs bursts when input rate > output

If sending rate is persistently > drain rate, queue builds

Dropped packets represent wasted work

Chapter 6, Figure 1

Fairness



Each flow from a source to a destination should (?) get an equal share of the bottleneck link … depends on paths and other traffic

Chapter 6, Figure 2

The Problem

Original TCP sent full window of data

When links become loaded, queues fill up, and this can lead to:

- C*ongestion collapse: w*hen round-trip time exceeds retransmit interval -- every packet is retransmitted many times
- Synchronized behavior: network oscillates between loaded and unloaded

TCP Congestion Control

Goal: efficiently and fairly allocate network bandwidth

- Robust RTT estimation
- Additive increase/multiplicative decrease
  - oscillate around bottleneck capacity
- Slow start
  - quickly identify bottleneck capacity
- Fast retransmit
- Fast recovery

Tracking the Bottleneck Bandwidth

Sending rate = window size/RTT

Multiplicative decrease

- Timeout => dropped packet => cut window size in half
  - and therefore cut sending rate in half

Additive increase

- Ack arrives => no drop => increase window size by one packet/window
  - and therefore increase sending rate a little

TCP “Sawtooth”

Oscillates around bottleneck bandwidth

- adjusts to changes in competing traffic

Slow start

How do we find bottleneck bandwidth?

- Start by sending a single packet
  - start slow to avoid overwhelming network
- Multiplicative increase until get packet loss
  - quickly find bottleneck
- Remember previous max window size
  - shift into linear increase/multiplicative decrease when get close to previous max ~ bottleneck rate
  - called “congestion avoidance”

Slow Start

Quickly find the bottleneck bandwidth

TCP Mechanics Illustrated

Source

Router

Dest

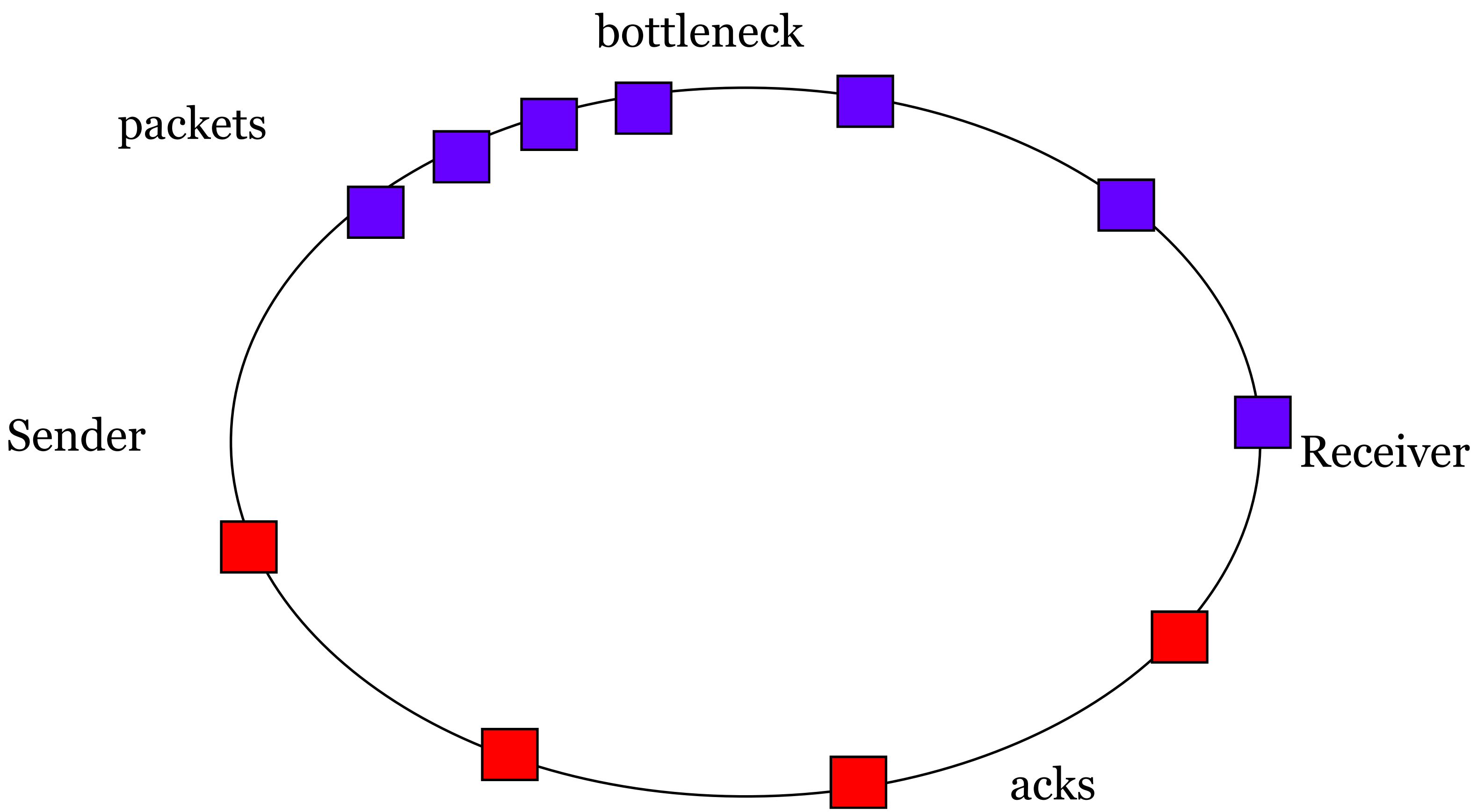
100 Mbps
0.9 ms latency

10 Mbps
0 latency

Slow Start Problems

- Bursty traffic source
  - will fill up router queues, causing losses for other flows
  - solution: ack pacing
- Slow start usually overshoots bottleneck
  - will lose many packets in window
  - solution: remember previous threshold
- Short flows
  - Can spend entire time in slow start!
  - solution: persistent connections?

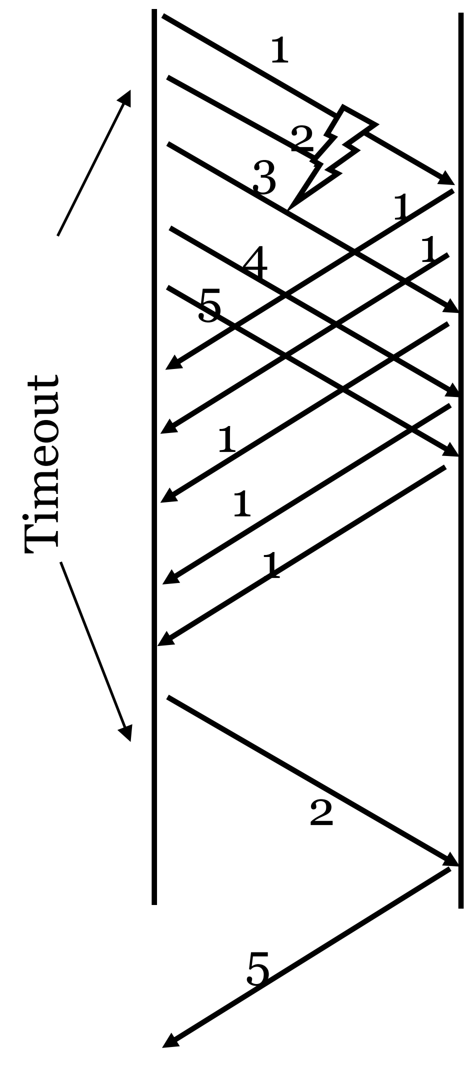
Avoiding burstiness: ack pacing



Window size = round trip delay * bit rate

Ack Pacing After Timeout

- Packet loss causes timeout, disrupts ack pacing
  - slow start/additive increase are *designed* to cause packet loss
- After loss, use slow start to regain ack pacing
  - switch to linear increase at last successful rate
  - "congestion avoidance"

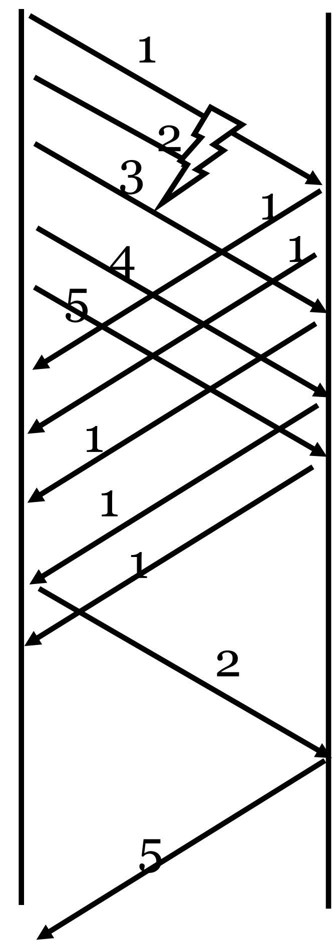


Putting It All Together

Timeouts dominate performance!

Fast Retransmit

- Can we detect packet loss without a timeout?
  - Receiver will reply to each packet with an ack for last byte received in order
- Duplicate acks imply either
  - packet reordering (route change)
  - packet loss
- TCP Tahoe
  - resend if sender gets three duplicate acks, without waiting for timeout



Fast Retransmit Caveats

Assumes in order packet delivery

- Recent proposal: measure rate of out of order delivery; dynamically adjust number of dup acks needed for retransmit

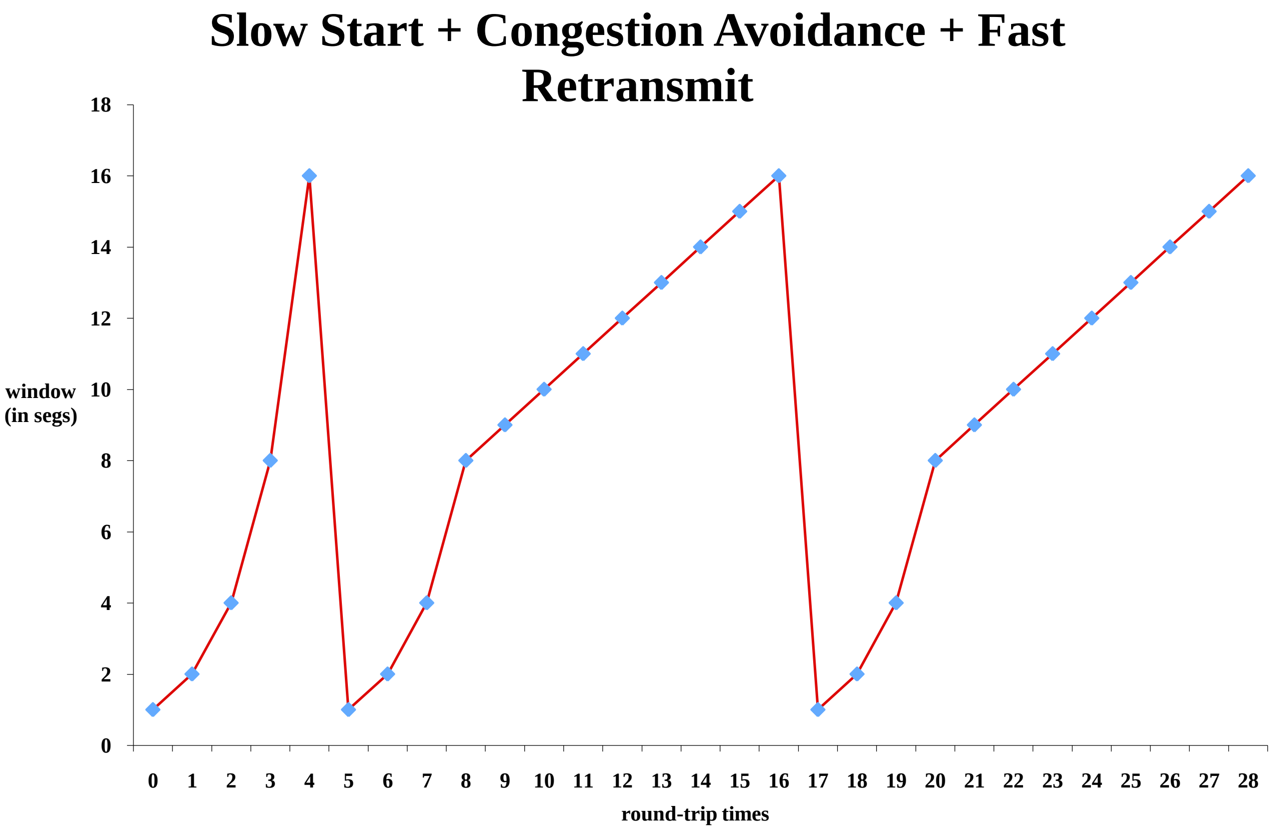
Doesn’t work with small windows (e.g. modems)

- what if window size <= 3

Doesn’t work if many packets are lost

- example: at peak of slow start, might lose many packets

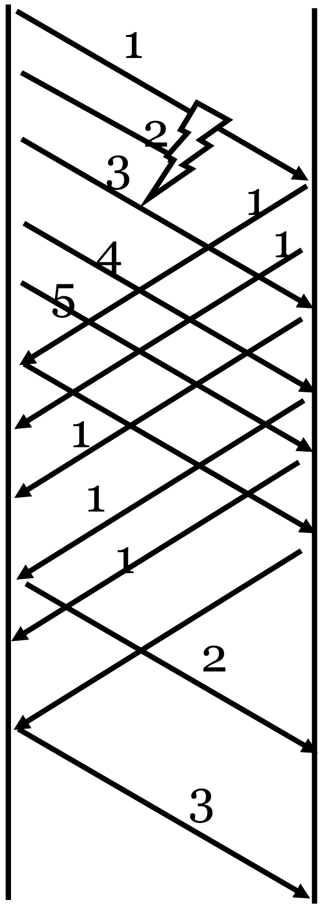
Fast Retransmit



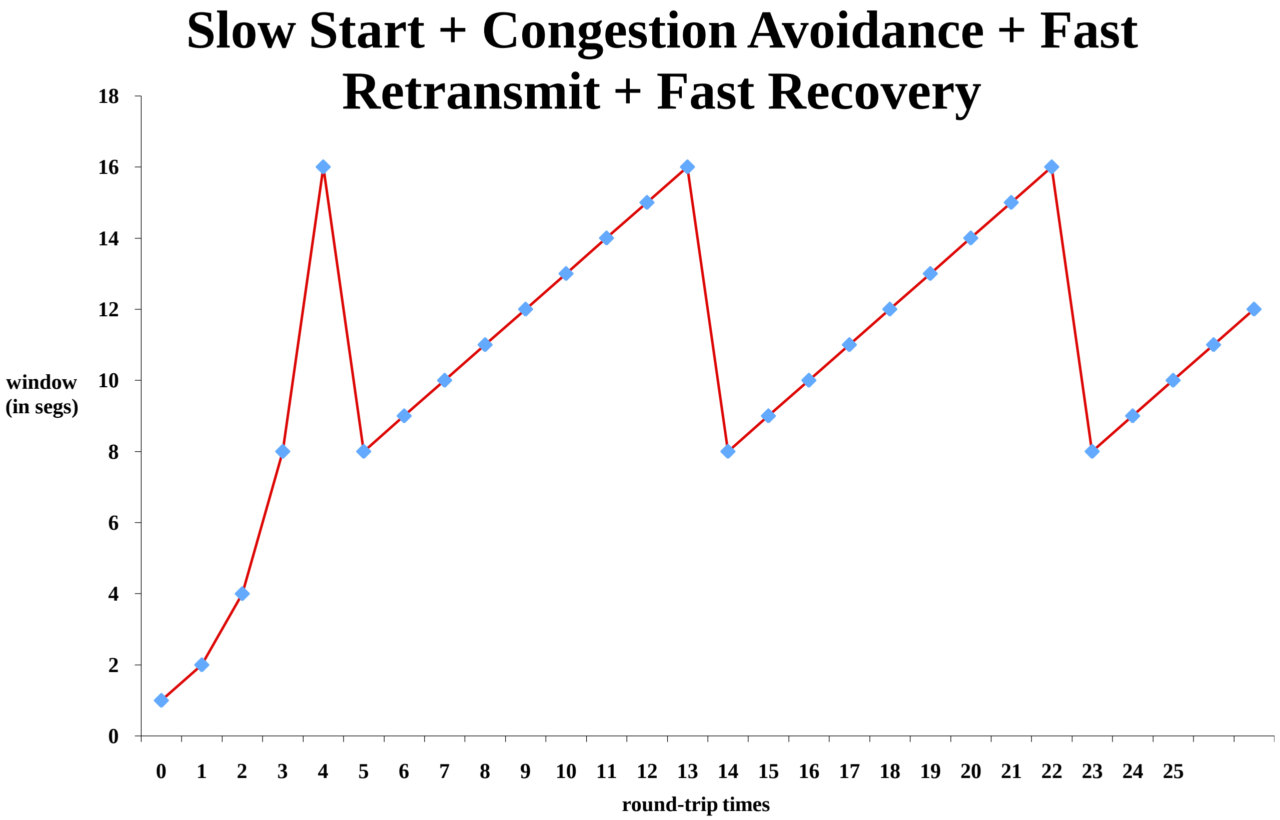
Regaining ack pacing limits performance

Fast Recovery

- Use duplicate acks to maintain ack pacing
  - duplicate ack => packet left network
  - after loss, send packet after every other acknowledgement
- Doesn't work if lose many packets in a row
  - fall back on timeout and slow start to reestablish ack pacing



Fast Recovery

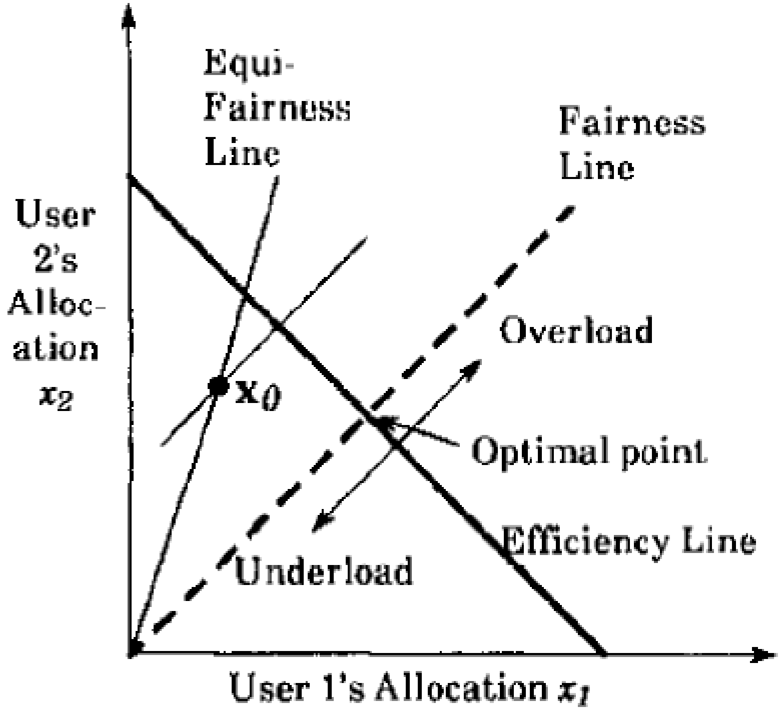


What if two TCPs share link?

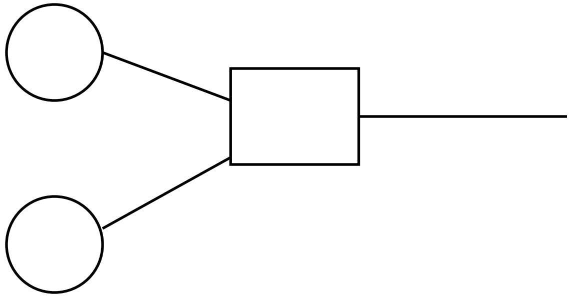
Reach equilibrium independent of initial bw

- assuming equal RTTs, “fair” drops at the router

Why AIMD?



Two users competing for bandwidth:



Consider the sequence of moves from AIMD, AIAD, MIMD, MIAD.

What if TCP and UDP share link?

Independent of initial rates, UDP will get priority!
TCP will take what's left.

What if two different TCP implementations share link?

If cut back more slowly after drops => will grab bigger share

If add more quickly after acks => will grab bigger share

Incentive to cause congestion collapse!

- Many TCP "accelerators"
- Easy to improve perf at expense of network

One solution: enforce good behavior at router

TCP Performance (Steady State)

Bandwidth as a function of

- RTT?
- Loss rate?
- Packet size?
- Receive window?

TCP over 10Gbps Pipes

What's the problem?

How might we fix it?

TCP over Wireless

What's the problem?

How might we fix it?

What if TCP connection is short?

Slow start dominates performance

- What if network is unloaded?
- Burstiness causes extra drops

Packet losses unreliable indicator

- can lose connection setup packet
- can get drop when connection near done
- signal unrelated to sending rate

In limit, have to signal every connection

- 50% loss rate as increase # of connections

Example: 10KB document

10Mb/s Ethernet,70ms RTT, 536 MSS

Ethernet ~ 10 Mb/s

64KB window, 70ms RTT ~ 7.5 Mb/s

can only use 10KB window ~ 1.2 Mb/s

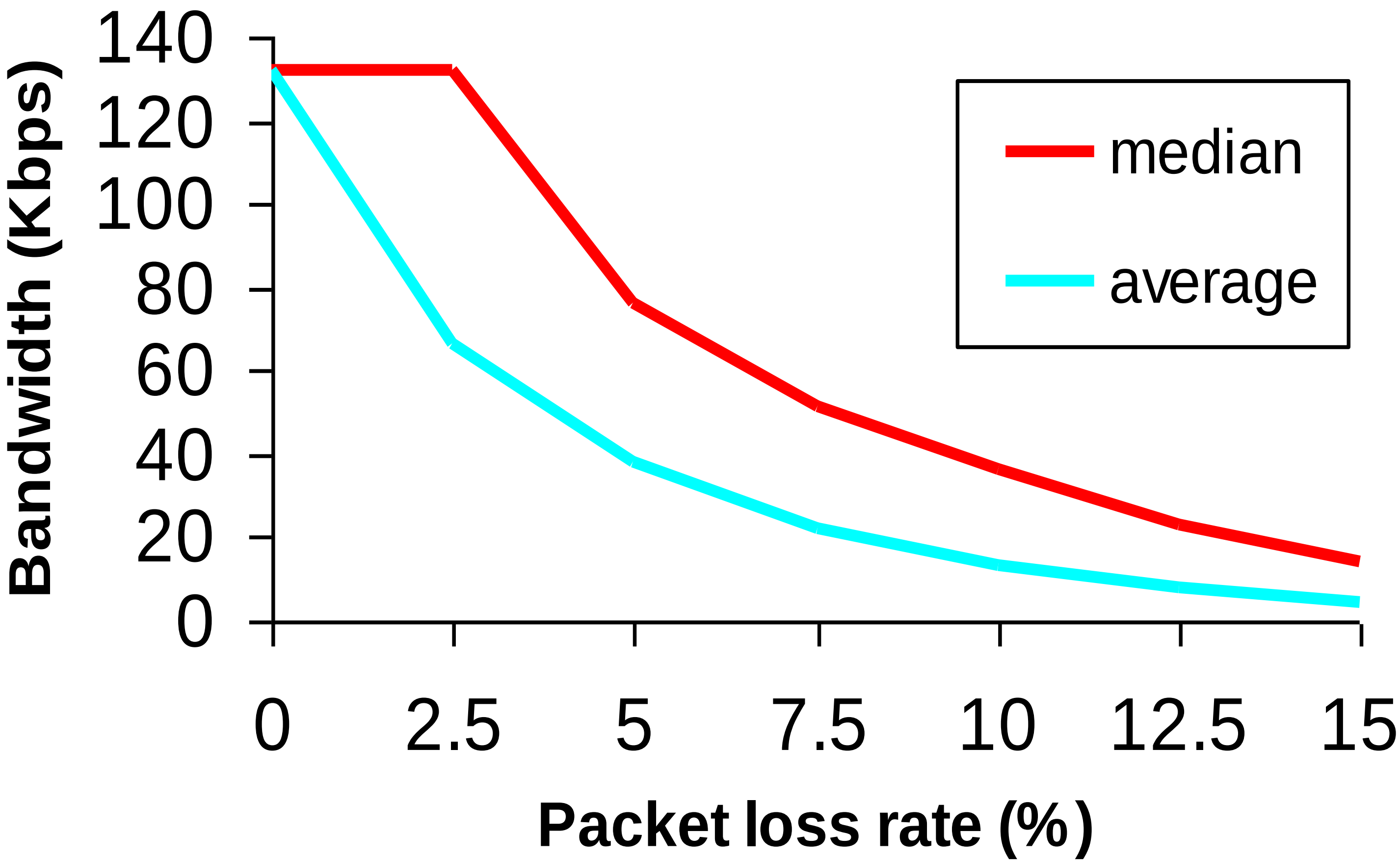
5% drop rate ~ 275 Kb/s (steady state)

model timeouts ~ 228 Kb/s

slow start, no losses ~ 140 Kb/s

slow start, with 5% drop ~ 75 Kb/s

Short flow bandwidth



Flow length=10Kbytes, RTT=70ms

Improving Short Flow Performance

Start with a larger initial window

- RFC 3390: start with 3-4 packets

Persistent connections

- HTTP: reuse TCP connection for multiple objects on same page
- Share congestion state between connections on same host or across host

Skip slow start?

Ignore congestion signals?

TCP and Real-time Flows

What's the problem?

How might we fix it?

Misbehaving TCP Receivers

- On server side, little incentive to cheat TCP
  - Mostly competing against other flows from same server
- On client side, high incentive to induce server to send faster
  - How?

Impact of Router Behavior on Congestion Control

Behavior of routers can have a large impact on the efficiency/fairness of congestion control

- buffer size
- queueing discipline (FIFO, round robin, priorities)
- drop policy -- Random Early Drop (RED)
- Early congestion notification (ECN)
- Weighted fair queueing
- Explicit rate control

Note that most solutions break layering

- change router to be aware of end to end transport

TCP Synchronization

Assumption for TCP equilibrium proof is that routers drop fairly

What if router's buffers are always full?

- anyone trying to send will experience drop
  - timeout and retry at reduced rate
- when router sends a packet, triggers an ack
  - causes that host to send another packet, refill buffers, causes other hosts to experience losses

One host can capture all of the bandwidth, even using TCP!

Router Buffer Space

- What is the effect of router queue size on network performance?
  - What if there were infinite buffers at each router?
    - what would happen to end to end latency?
  - What if only one packet could be buffered?
    - what would happen if multiple nodes wanted to share a link?
- Subtle interactions between TCP feedback loop and router configuration
  - rule of thumb: buffer space at each router should be equal to the end to end bandwidth delay product (how?)

Congestion Avoidance

TCP causes congestion as it probes for the available bandwidth and then recovers from it after the fact

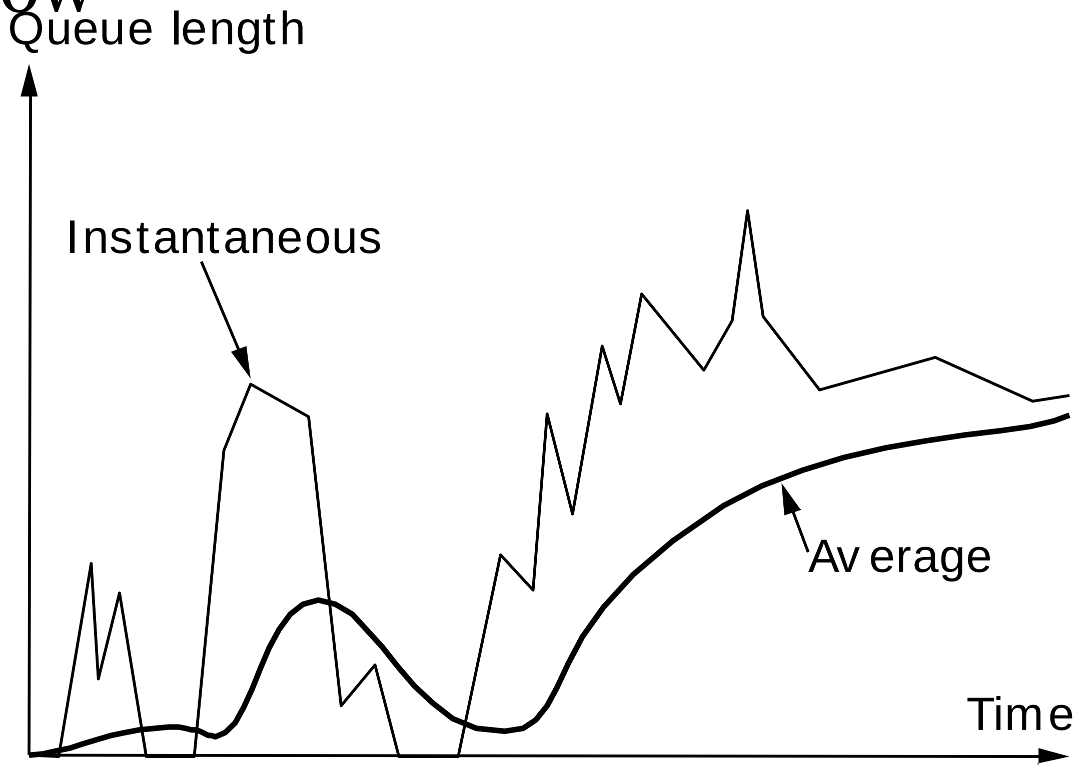
- Leads to loss, delay and bandwidth fluctuations (Yuck!)
- We want congestion avoidance, not congestion control

Congestion avoidance mechanisms

- Aim to detect incipient congestion, before loss. So monitor queues to see that they absorb bursts, but not build steadily

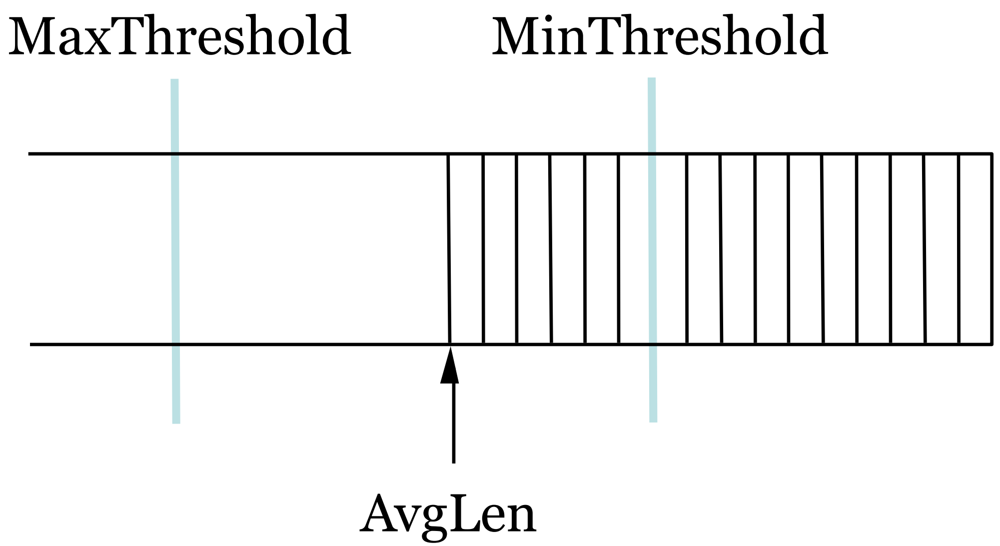
Incipient Congestion at a Router

Sustained overload causes queue to build and overflow



Random Early Detection (RED)

Have routers monitor average queue and send "early" signal to source when it builds by probabilistically dropping a packet



Paradox: early loss can improve performance!

Explicit Congestion Notification (ECN)

- Why drop packets to signal congestion?
  - Drops are a robust signal, but there are other means …
  - We need to be careful though: no extra packets
- ECN signals congestion with a bit in the IP header
- Receiver returns indication to the sender, who slows
  - Need to signal this reliably or we risk instability
- RED actually works by “marking” packets
  - Mark can be a drop or ECN signal if hosts understand ECN
  - Supports congestion avoidance without loss

Difficulties with RED

Nice in theory, hasn't caught on in practice.

Parameter issue:

- What should dropping probability (and average interval) be?
- Consider the cases of one large flow vs N very small flows

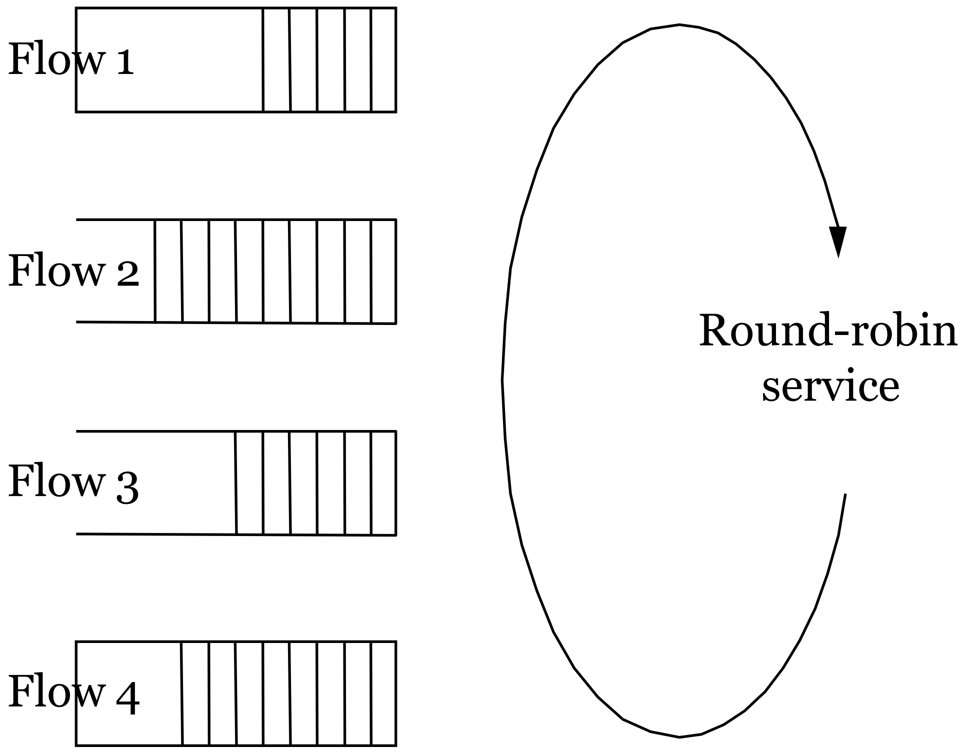
Incentive issue:

- Why should ISPs bother to upgrade?
  - RED doesn't increase utilization, the basis of charging
- Why should end-hosts bother to upgrade?
  - The network doesn't support RED

Fair Queuing (FQ)

- FIFO is not guaranteed (or likely) to be fair
  - Flows jostle each other and hosts must play by the rules
  - Routers don't discriminate traffic from different sources
- Fair Queuing is an alternative scheduling algorithm
  - Maintain one queue per traffic source (flow) and send packets from each queue in turn
    - Actually, not quite, since packets are different sizes
  - Provides each flow with its "fair share" of the bandwidth

Fair Queuing



WFQ implication

What should the endpoint do, if it knows router is using WFQ?

Traffic shaping

At enterprise edge, shape traffic:

- Avoid packet loss
- Maximize bandwidth utilization
- Prioritize traffic
- No changes to endpoints (as with NATs)

Mechanism?

TCP Known to be Suboptimal

Small to moderate sized connections

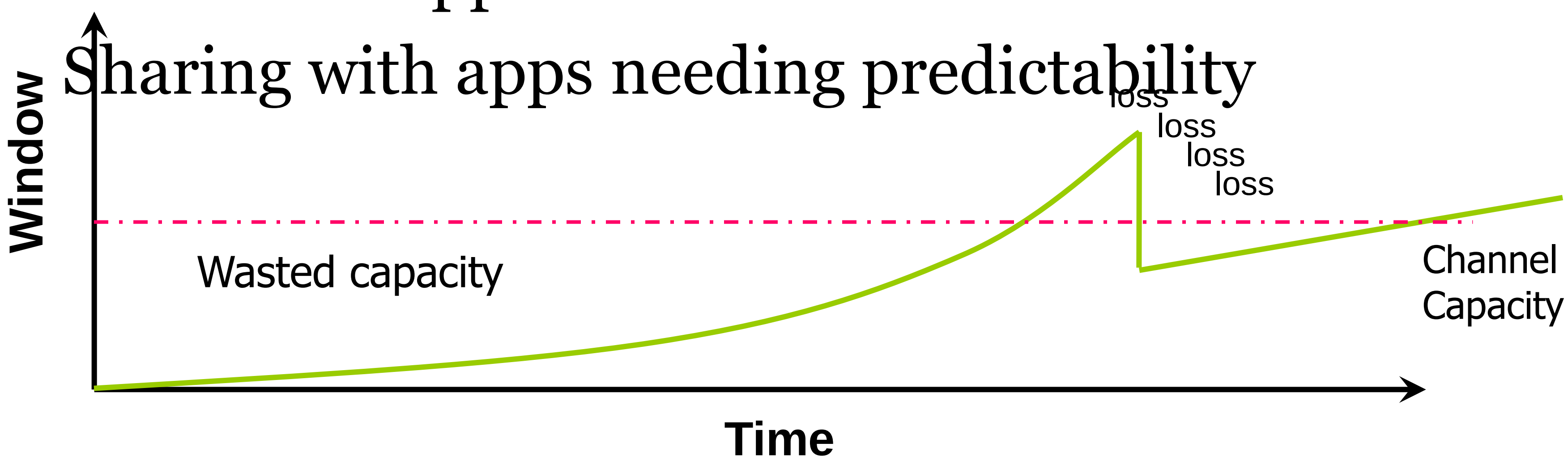
Paths with low to moderate utilization

Wireless transmission loss

High bandwidth; high delay

Interactive applications

Sharing with apps needing predictability



Observation

Trivial to be optimal with help from the network; e.g., ATM rate control

- Hosts send bandwidth request into network
- Network replies with safe rate (min across links in path)

Non-trivial to change the network

Question

Can endpoint congestion control be near optimal with *no* change to the network?

Assume: cooperating endpoints

- For isolation, implement fair queueing
- PCP does well both with and without fair queueing

PCP approach: directly emulate optimal router behavior!

Probe Control Protocol (PCP)

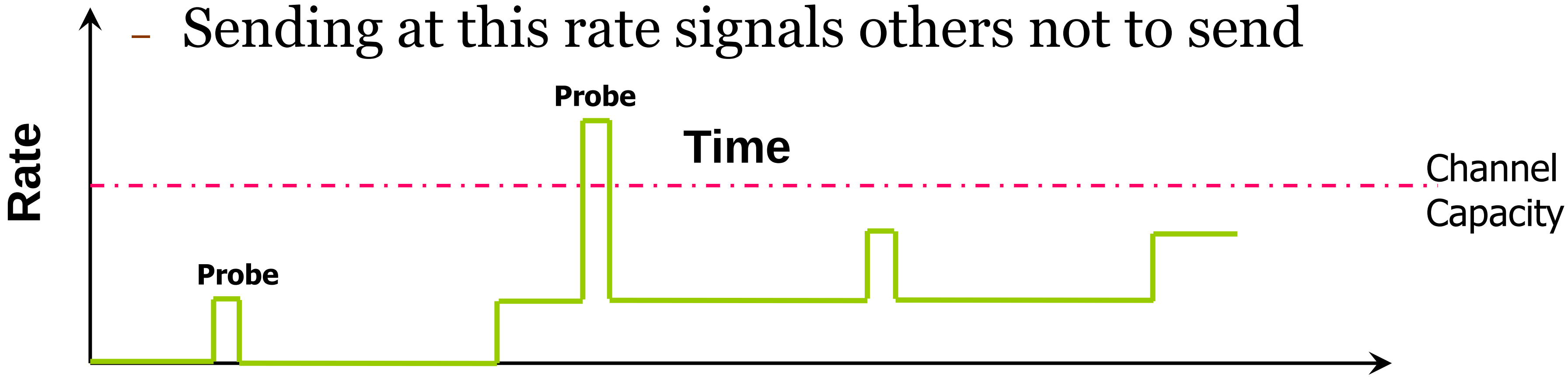
Probe for bandwidth using short burst of packets

- If bw available, send at the desired uniform rate (paced)
- If not, try again at a slower rate

Probe is a request

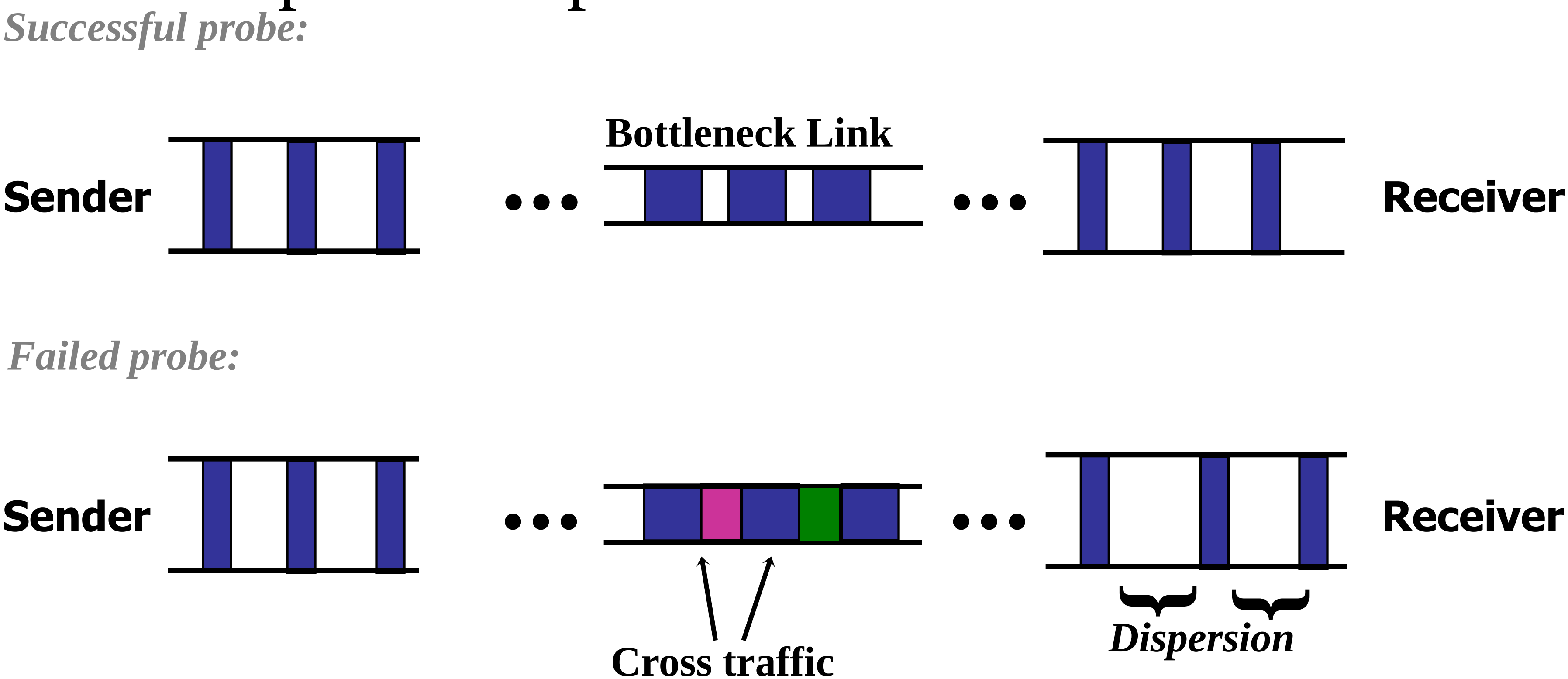
Successful probe sets the sending rate

- Sending at this rate signals others not to send



Probes

- Send packet train spaced to mimic desired rate
- Check packet dispersion at receiver



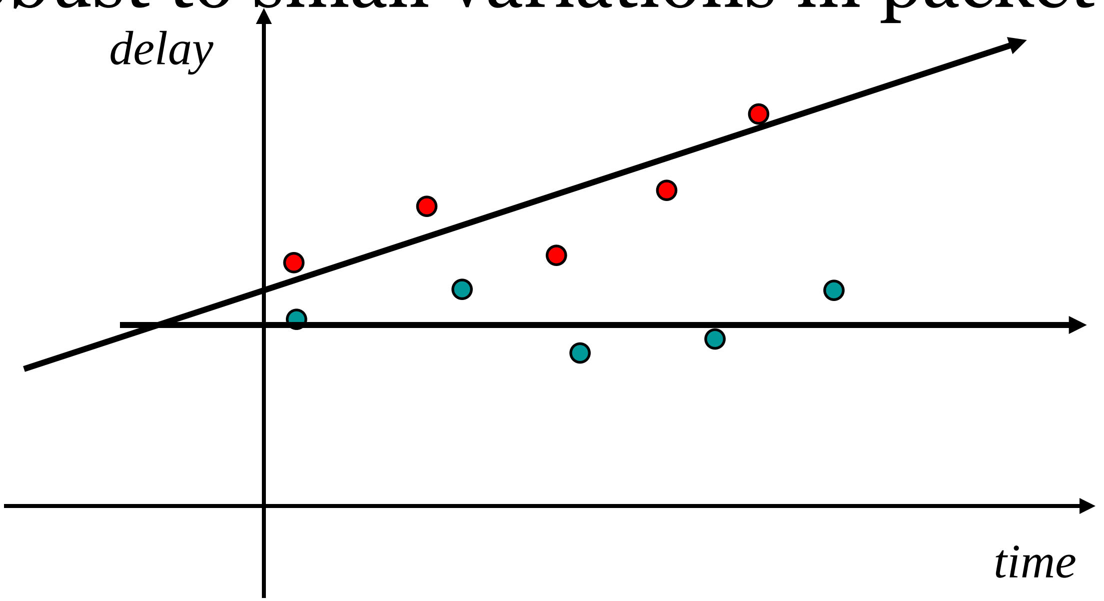
Probabilistic Accept

Randomly generate a slope consistent with the observed data

- same mean, variance as least squares fit

Accept if slope is not positive

Robust to small variations in packet scheduling



Rate Compensation

Queues can still increase:

- Failed probes, even if short, can result in additional queueing
- Simultaneous probes could allocate the same bandwidth
- Probabilistic accept may decide probe was successful, without sufficient underlying available bandwidth

PCP solution

- Detect increasing queues by measuring packet latency and inter-packet delay
- Each sender decreases their rate proportionately, to eliminate queues within a single round trip
- Emulates AIMD, and thus provides eventual fairness

TCP Compatibility

TCP increases its rate regardless of queue size

- Should PCP keep reducing its rate to compensate?

Solution: PCP becomes more aggressive in presence of non-responsive flows

- If rate compensation is ineffective, reduce speed of rate compensation: “tit for tat”
- When queues drain, revert to normal rate compensation

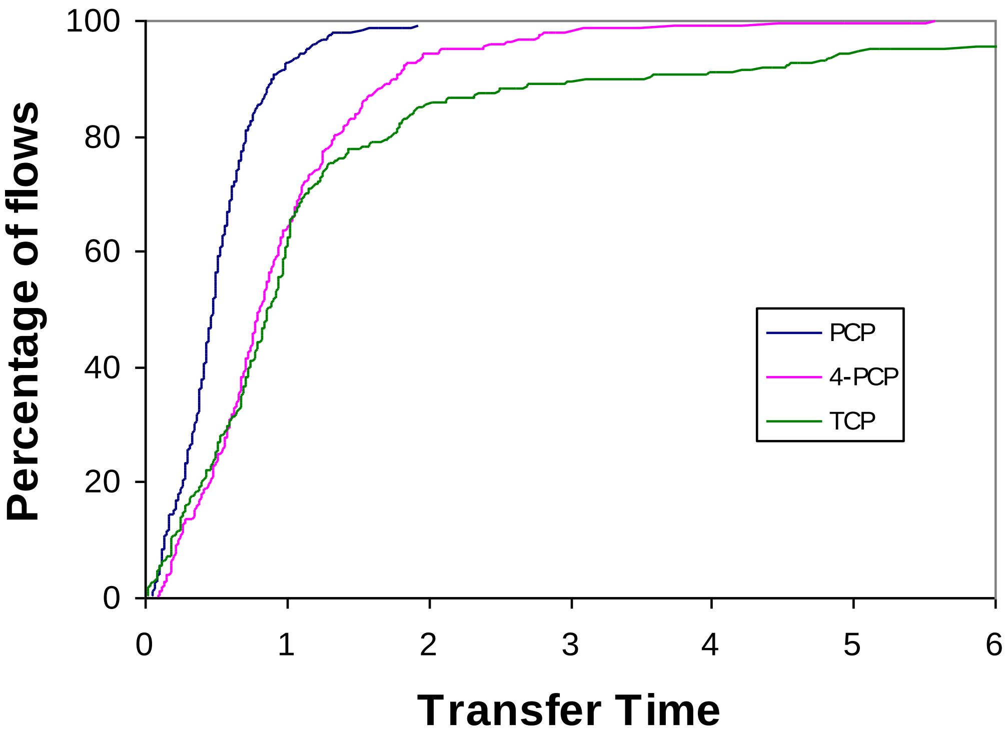
Otherwise compatible at protocol level

- PCP sender (receiver) induces TCP receiver (sender) to use PCP

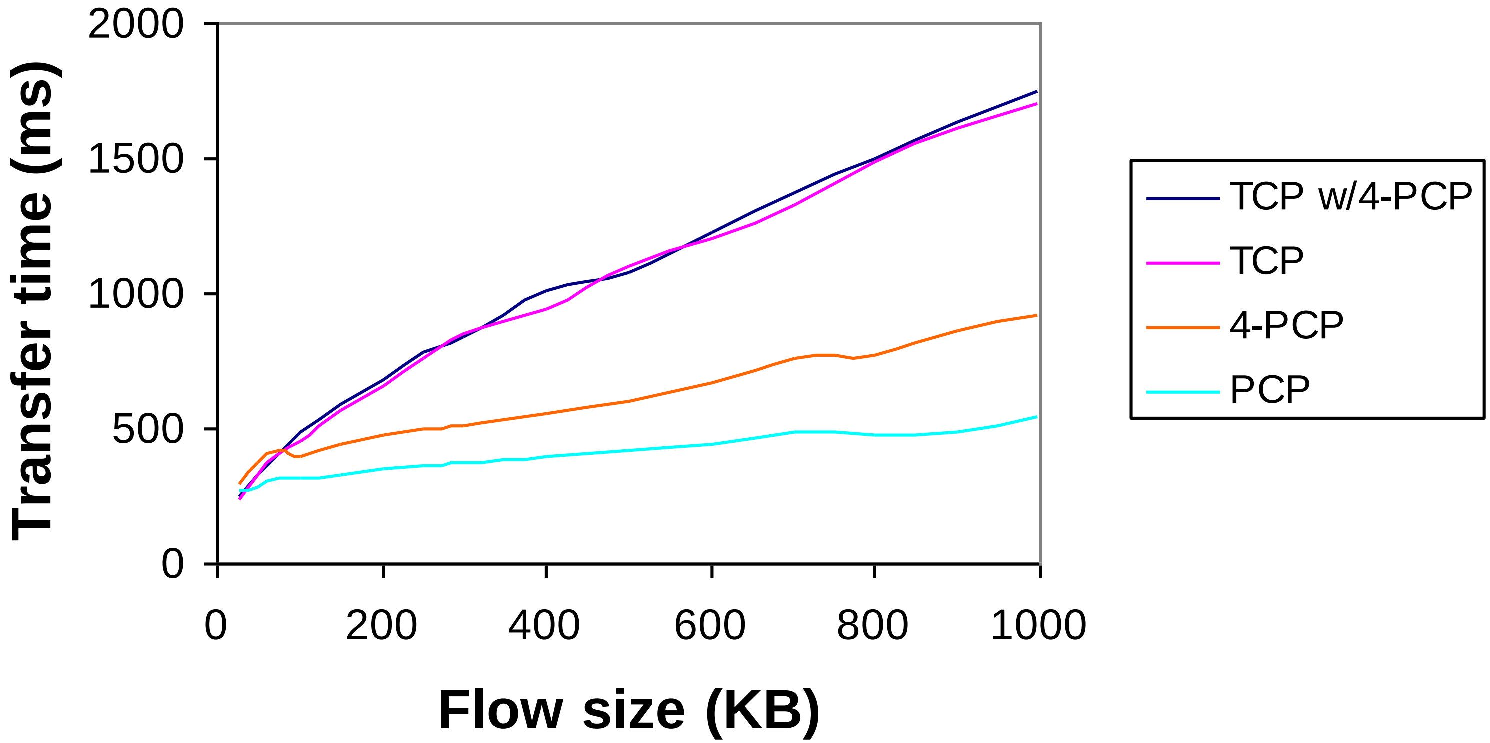
Performance

User-level implementation

- 250KB transfers between every pair of US RON nodes
- PCP vs. TCP vs. four concurrent PCP transmissions



Is PCP Cheating?



Roadmap – Various Mechanisms

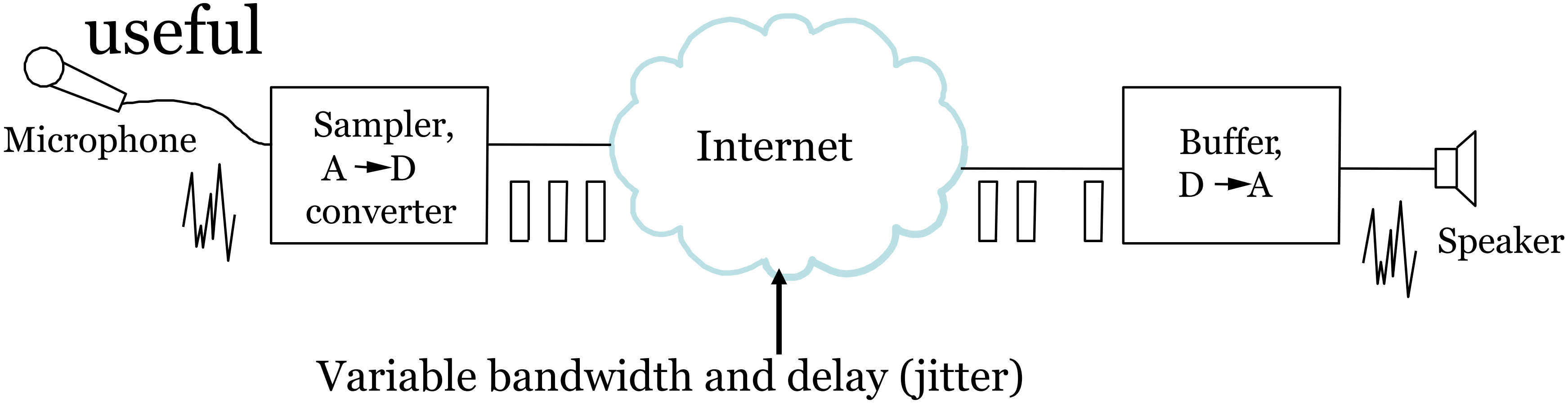
Simple to build,
Weak assurances

Complex to build,
Strong assurances

Classic Best Effort	FIFO with Drop Tail
Congestion Avoidance	FIFO with RED
Per Flow Fairness	Weighted Fair Queuing
Aggregate Guarantees	Differentiated Services
Per Flow Guarantees	Integrated Services

VoIP: A real-time audio example

VoIP is a real-time service in the sense that the audio must be received by a deadline to be useful



Real-time apps need assurances from the network

Q: What assurances does VoIP require?

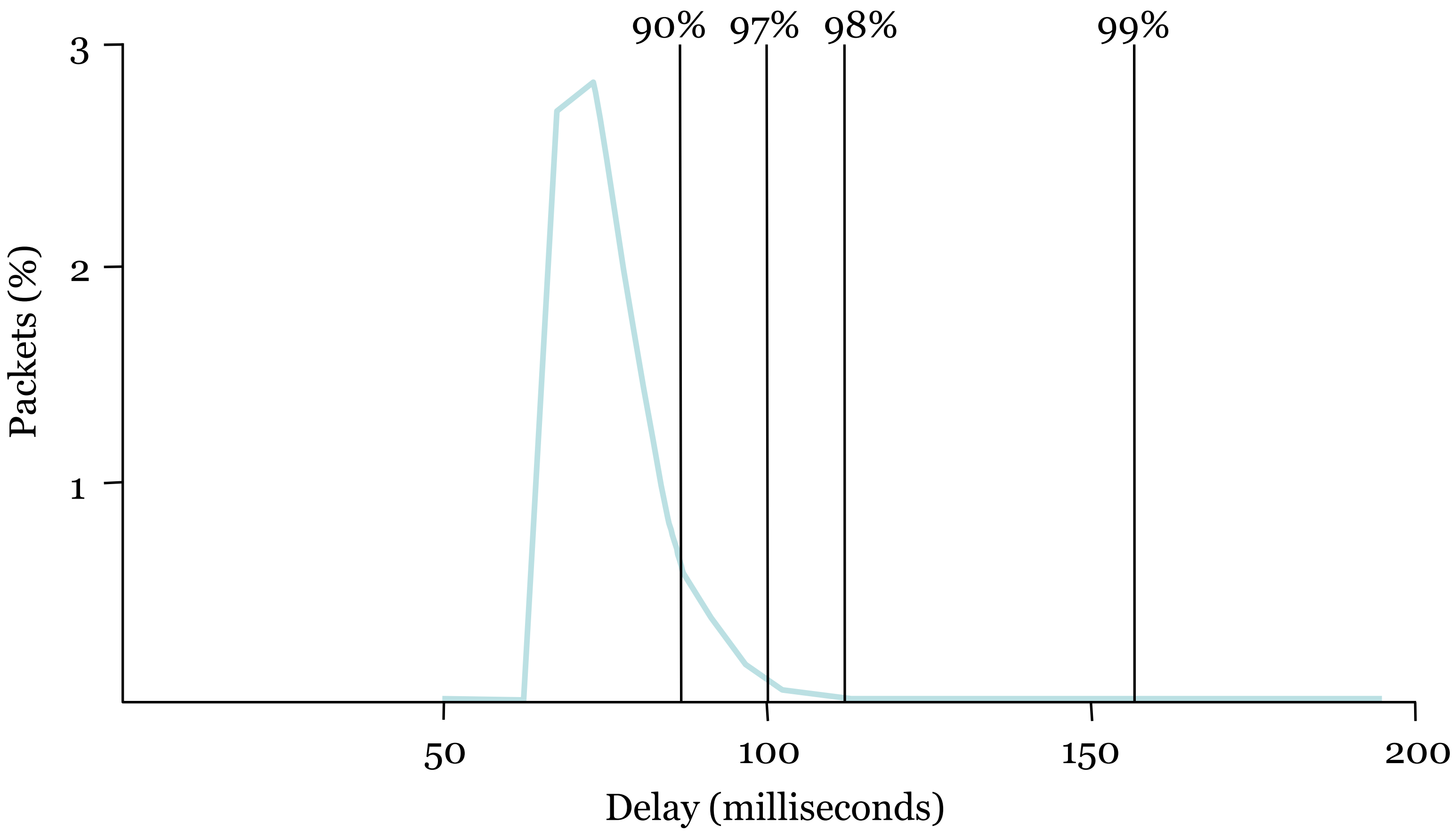
Network Support for VoIP

Bandwidth
- There must be enough on average
- But we can tolerate to short term fluctuations

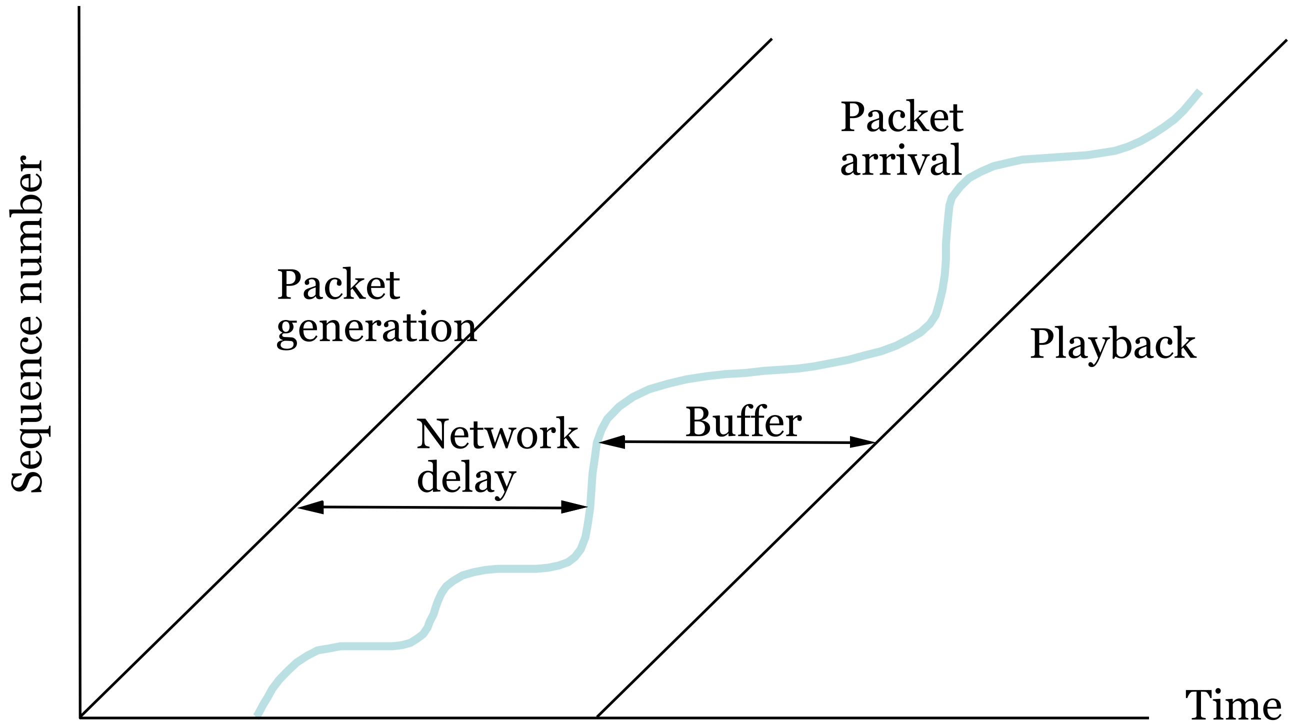
Delay
- Ideally it would be fixed
- But we can tolerate some variation (jitter)

Loss
- Ideally there would be none
- But we can tolerate some losses. (How?)

Example: Delay and Jitter



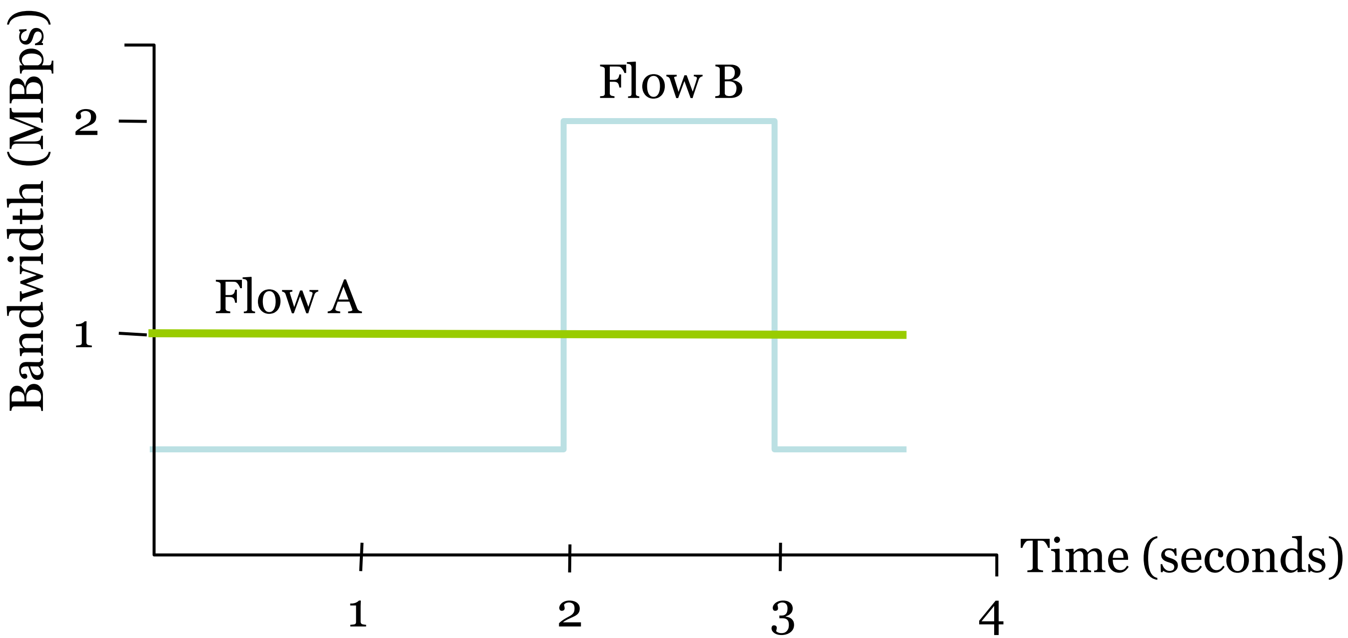
Tolerating Jitter with Buffering



Buffer before playout so that most late samples will have arrived

Specifying Bandwidth Needs

Problem: Many applications have variable bandwidth demands



Same average, but very different needs over time. One number. So how do we describe bandwidth to the network?

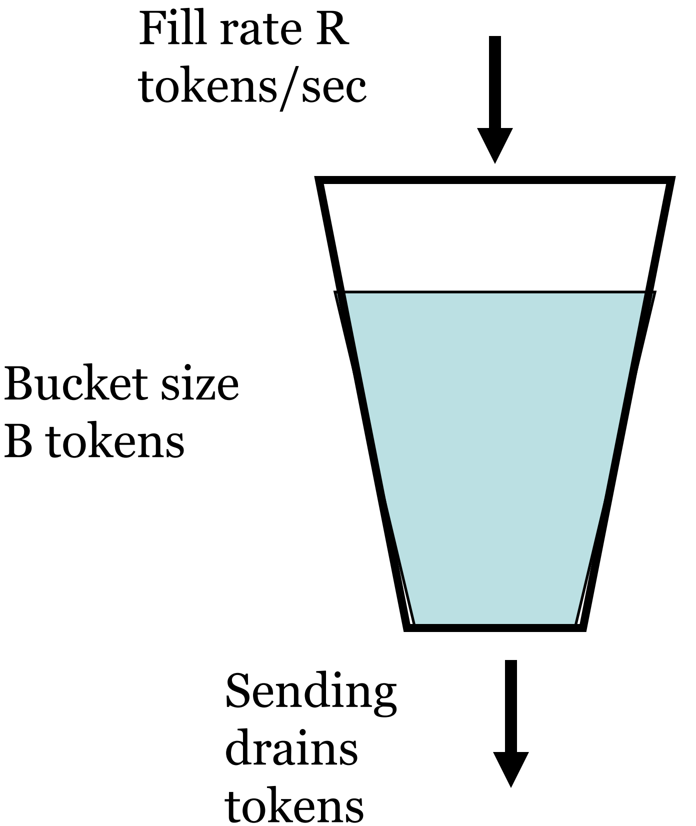
Token Buckets

Common, simple descriptor

Use tokens to send bits

Average bandwidth is R bps

Maximum burst is B bits



Supporting QOS Guarantees

1. Flowspecs. Formulate application needs
   - Need descriptor, e.g. token bucket, to ask for guarantee
2. Admission Control. Decide whether to support a new guarantee
   - Network must be able to control load to provide guarantees
3. Signaling. Reserve network resources at routers
   - Analogous to connection setup/teardown, but at routers
4. Packet Scheduling. Use different scheduling and drop mechanisms to implement the guarantees
   - e.g., set up a new queue and weight with WFQ at routers

The need for admission control

Suppose we have an <r,b> token bucket flow and we are interested in how much bandwidth the flow receives from the network.

Consider a network with FIFO nodes. What rate does the flow get?

Now consider a network with (W)FQ nodes. What rate does the flow get?

Now consider a network with (W)FQ nodes where $w(i) = r(i)$ and $\sum w(i) = W <$ capacity at each node. What rate does the flow get?

Bounding Bandwidth and Delay

WFQ with admission control can bound bandwidth and delay. Wow! (Parekh and Gallagher GPS result)

For a single node:

- Bandwidth determined by weights: g(i) = C * w(i)/W
- E2E delay <= propagation + burst/g(i) + packet/g(i) + packet/C

For multiple nodes:

- Bandwidth is determined by the minimum g(i) along the path
- E2E delay pays for burst smoothing only once, plus further transmission and pre-emption delays

IETF Integrated Services

Fine-grained (per flow) guarantees

- Guaranteed service (bandwidth and bounded delay)
- Controlled load (bandwidth but variable delay)

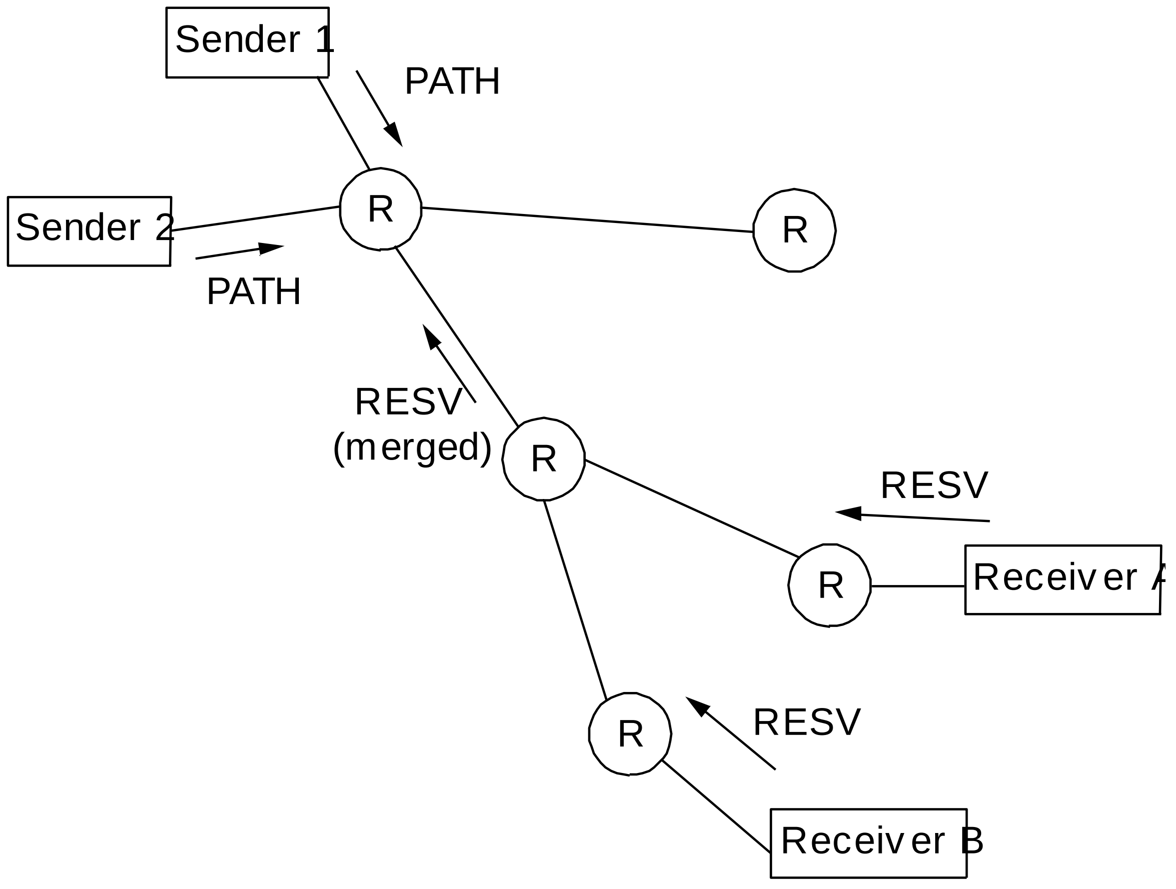
RSVP used to reserve resources at routers

- Receiver-based signaling that handles failures

WFQ used to implement guarantees

- Router classifies packets into a flow as they arrive
- Packets are scheduled using the flow's resources

Resource Reservation Protocol (RSVP)



RSVP Issues

RSVP is receiver-based to support multicast apps

Only want to reserve resources at a router if they are sufficient along the entire path

What if there are link failures and the route changes?

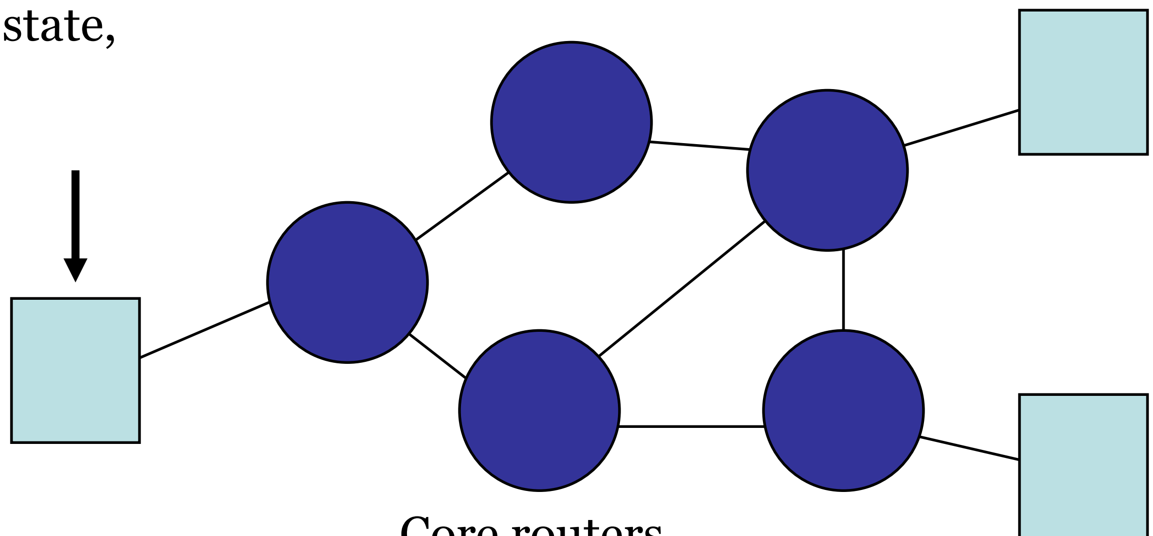
What if there are sender/receiver failures?

IETF Differentiated Services

- A more coarse-grained approach to QOS
  - Packets are marked as belonging to a small set of services, e.g, premium or best-effort, using the TOS bits in the IP header
- This marking is policed at administrative boundaries
  - Your ISP marks 10Mbps (say) of your traffic as premium depending on your service level agreement (SLAs)
  - SLAs change infrequently; much less dynamic than Intserv
- Routers understand only the different service classes
  - Might separate classes with WFQ, but not separate flows

Two-Tiered Architecture

Mark at Edge routers
(per flow state,
complex)



Core routers
stay simple
(no per-flow state,
few classes)

DiffServ Issues

How do ISPs provision?

- Traffic on your access link may follow different paths inside ISP network. Can we provide an access link guarantee efficiently?

What's the policy?

- Which traffic is gold, which silver, etc.?

Overprovisioning, other issues

An alternative:

- Provide more capacity than load; it's all a cost tradeoff
- Bandwidth to user limited mainly by their access capacity
- Delay through network limited mainly by propagation delay

Deploying QOS:

- What good is it if only one ISP deploys?
- Incentives for single ISP for distributed company using VoIP
- And incentive for inter-provider agreements
- Network QOS as an extension of single box packet shapers