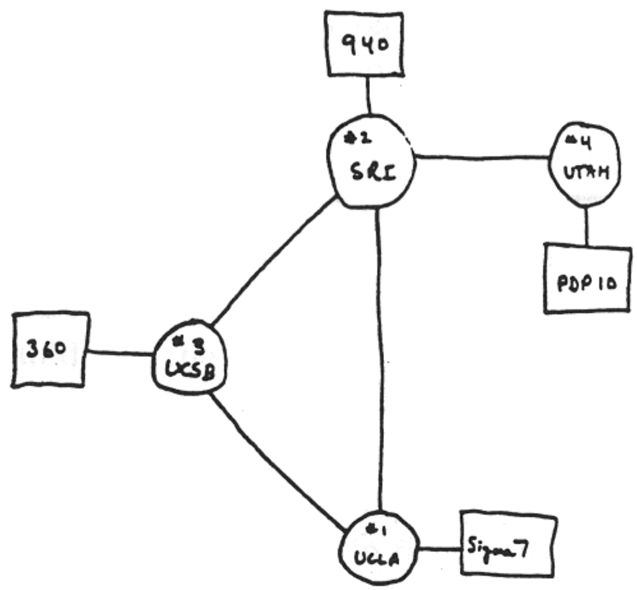
Networking: TCP/IP



THE ARPA NETWORK

DEC 1969

4 NODES

Network Protocol Stack Model

Application	User interaction	HTTP, FTP, SMTP
Presentation	Data representation	XML, cryptography
Session	Dialogue mangement	???
Transport	Reliable end-to-end link	TCP
Network	Routing via multiple nodes	IP
Data Link	Physical addressing	Ethernet
Physical	Metal or RF representation	802.11, Bluetooth

Open System Interconnection (OSI) Reference Model

Network Protocol Stack Model

Web

Application	User interaction	HTTP, FTP, SMTP
Presentation	Data representation	XML, cryptography
Session	Dialogue mangement	???
Transport	Reliable end-to-end link	TCP
Network	Routing via multiple nodes	IP
Data Link	Physical addressing	Ethernet
Physical	Metal or RF representation	802.11, Bluetooth

Internet

- The Web uses HTTP, but is built on TCP/IP
- Understanding HTTP requires understanding TCP, which requires IP

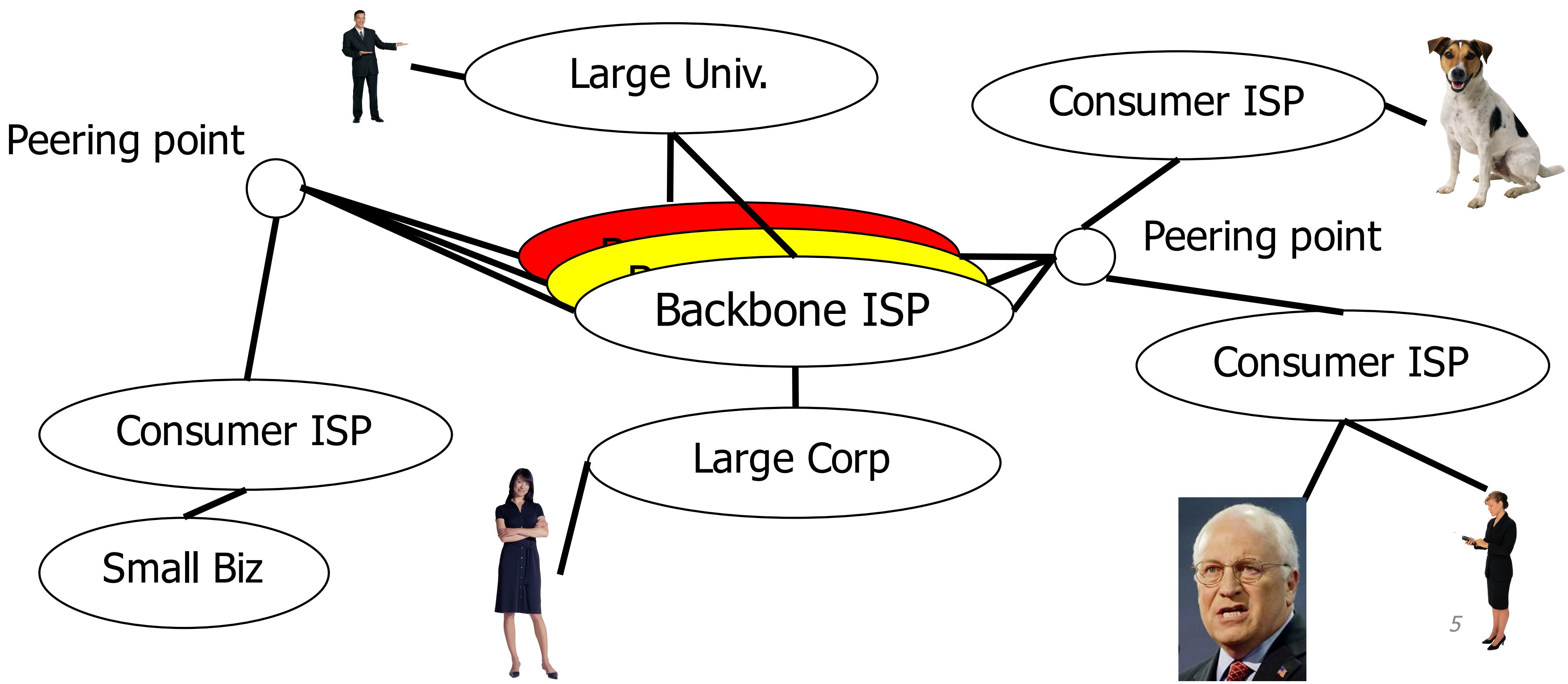
Network Protocol Stack Model

Application	User interaction	HTTP, FTP, SMTP
Presentation	Data representation	XML, cryptography
Session	Dialogue mangement	???
Transport	Reliable end-to-end link	TCP
Network	Routing via multiple nodes	IP
Data Link	Physical addressing	Ethernet
Physical	Metal or RF representation	802.11, Bluetooth

Today, we'll start with IP, and then move to TCP

IP

- Best effort packet-switched network
  - Basic unit is packet, sent by hosts
  - Packets may arrive late, or not at all
  - IP routers form the core of the internet



IP

- Design points:
  - IP is connectionless; “store-and-forward”
  - Different packets can take different paths
- IP addresses are 32 bits (in IPv4)
  - E.g., 192.168.1.1
- Forwarding-based networking
  - Each host has a routing table
  - Forward packet to “longest prefix match”
  - A major task: building the forwarding table

Destination	Gateway
Default	192.168.1.1
192.168.212.111	0:1f:6d:e8:18:0

Checking your IP configuration

```
$ ifconfig
em1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 141.213.74.56  netmask 255.255.255.0  broadcast 141.213.74.255
        inet6 fe80::ca1f:66ff:febb:6450  prefixlen 64  scopeid 0x20<link>
        ether c8:1f:66:bb:64:50  txqueuelen 1000  (Ethernet)
        RX packets 116371674  bytes 41215715226 (38.3 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 173430592  bytes 193165256406 (179.8 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 16
```

Datagrams

- A datagram is a term for packets in a packet-switched network
- IP is sometimes called a "datagram" service
- IP is frequently used as a subcomponent of TCP/IP
  - Reliable
- IP is sometimes used directly, as with UDP
  - Unreliable
  - Often used for time-sensitive applications, e.g., voice and video

Traceroute

- `traceroute` uses debug messages to expose packet's route to destination

```
$ traceroute google.com
traceroute to google.com (216.58.192.142), 64 hops max, 52 byte packets
1  172.27.35.1 (172.27.35.1)  258.783 ms  2.826 ms  3.114 ms
2  96.120.40.141 (96.120.40.141)  15.454 ms  195.473 ms  295.828 ms
3  68.85.85.33 (68.85.85.33)  364.012 ms  366.832 ms  367.044 ms
4  69.139.255.249 (69.139.255.249)  365.197 ms  16.660 ms  17.138 ms
5  be-33668-cr02.ashburn.va.ibone.comcast.net (68.86.90.13)  28.750 ms  31.213 ms  34.208
ms
6  be-10142-pe01.ashburn.va.ibone.comcast.net (68.86.86.34)  30.084 ms  28.648 ms  28.599
ms
7  173.167.57.234 (173.167.57.234)  28.986 ms  32.815 ms  34.034 ms
8  108.170.240.99 (108.170.240.99)  28.927 ms    108.170.246.3 (108.170.246.3)  28.755 ms
108.170.240.99 (108.170.240.99)  37.554 ms
9  216.239.48.95 (216.239.48.95)  29.829 ms    216.239.50.97 (216.239.50.97)  29.968 ms
216.239.49.197 (216.239.49.197)  32.562 ms
10  108.170.237.42 (108.170.237.42)  33.860 ms    209.85.143.170 (209.85.143.170)  36.263
ms    209.85.250.8 (209.85.250.8)  34.160 ms
11  209.85.243.172 (209.85.243.172)  32.889 ms    209.85.243.162 (209.85.243.162)  33.919
ms    72.14.232.163 (72.14.232.163)  33.880 ms
12  108.170.243.225 (108.170.243.225)  32.772 ms  38.971 ms  32.825 ms
13  216.239.42.149 (216.239.42.149)  35.472 ms    216.239.42.153 (216.239.42.153)  33.986
ms  44.282 ms
14  ord36s01-in-f14.1e100.net (216.58.192.142)  35.416 ms  33.993 ms  32.819 ms
```

Traceroute

- `traceroute` uses debug messages to expose packet’s route to destination

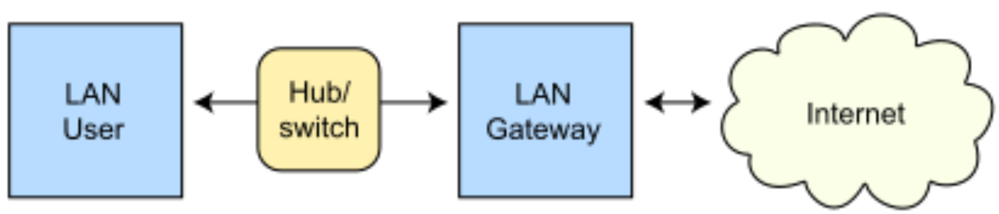
```
traceroute to google.com (74.125.95.99), 64 hops max, 52 byte packets

1 141.212.111.1 (141.212.111.1)  1.037 ms (Ann Arbor, MI)

2 13-caen-bin-arb.r-bin-arbl.umnet.umich.edu (192.12.80.177)  0.566 (Ann Arbor,
  MI)
…
7  216.239.48.154 (216.239.48.154)  6.785 ms (Mountain View, CA)
8  209.85.241.22 (209.85.241.22)  43.101 ms (Mountain View, CA)
```

IP’s Small Corners

- How does a host get an IP address?
  - Used to be static
  - Nowadays, often DHCP (Dynamic Host Configuration Protocol)
- How does a host get its gateway addr?
  - (Same)

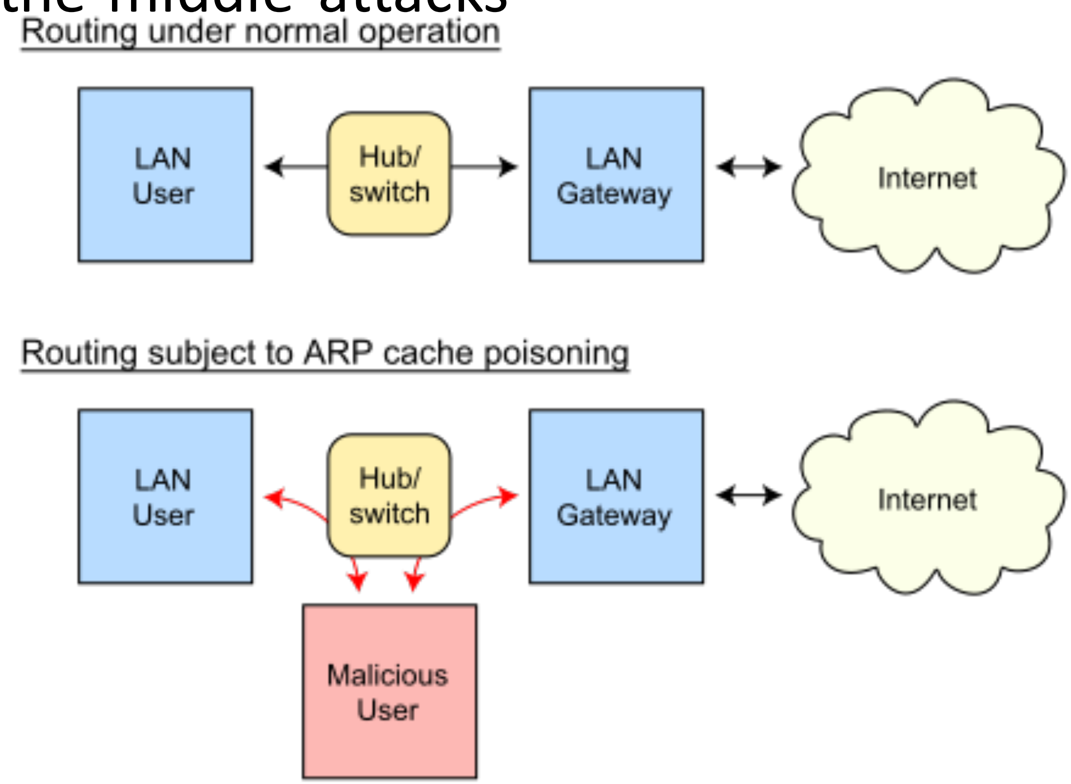


IP's Small Corners

- How does an IP address get mapped to a target Ethernet address?
  - ARP (Address Resolution Protocol)

ARP Spoofing

- ARP spoofing (or poisoning)
  - Associate attacker's MAC address with IP of another host(s)
  - Man-in-the-middle attacks



IP address on the Web

- How does a browser know which IP address to contact?
- By a Directory look-up, using Domain Name Service (DNS)
- Cache recently used domain names to reduce need for DNS look up
- DNS lookup

```
$ host umich.edu
umich.edu has address 141.211.243.44
```

TCP

- Transport Control Protocol
  - Reliable, ordered byte streams
  - Connection-oriented, unlike IP
  - “Virtual circuit” networking built on packet infrastructure
  - Looks like a circuit, but no reservations (on IP)

TCP

- Processes tied to “host:port” pairs
  - Ports are an OS-level concept
  - Server ports are “well-known” and associated with services; other ports are “ephemeral”

  - PORT QUIZ

TCP Port Quiz

- HTTP
- SSH
- HTTPS
- SMTP
- IMAP

TCP Port Quiz

- HTTP
  - 80
- SSH
  - 22
- HTTPS
  - 443
- SMTP
  - 25
- IMAP
  - 143

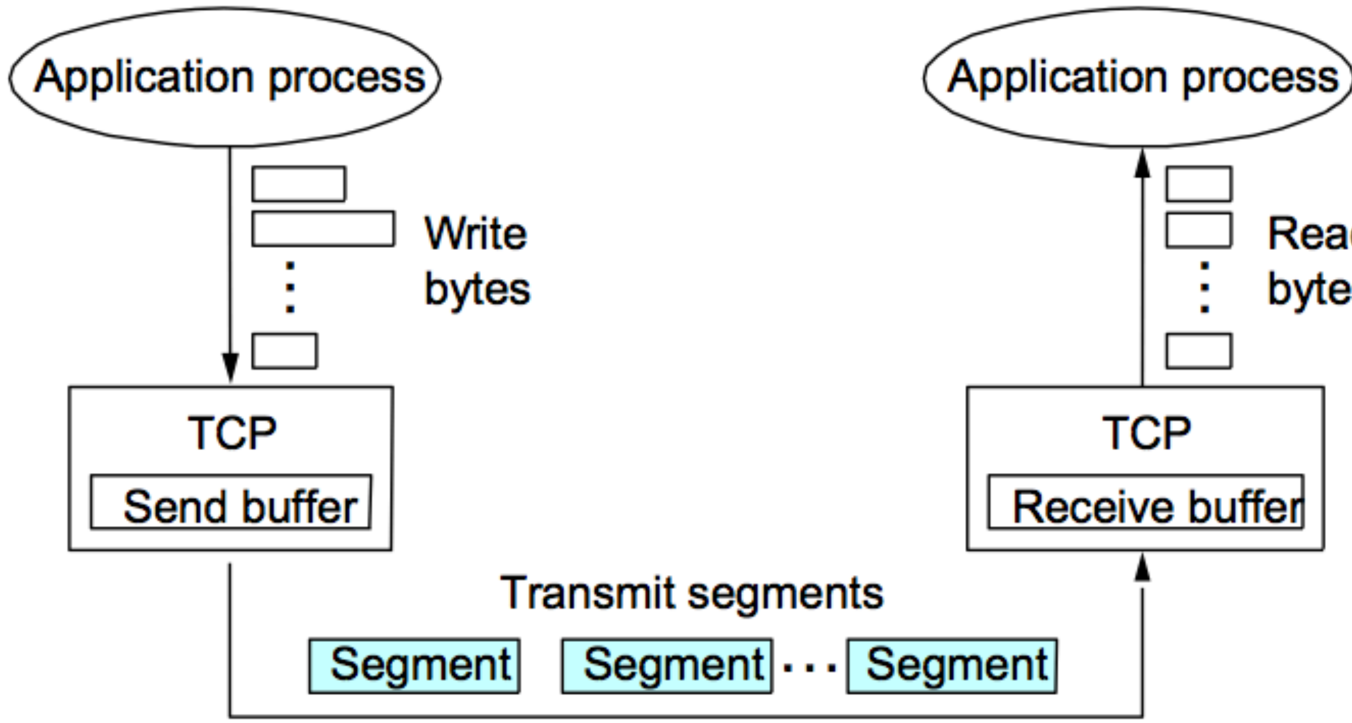
TCP/IP Properties

- *Connections* vs IP's *datagrams*
- *Reliable* delivery vs IP's *lost packets*
  - In-order
  - Delivered once and only once
  - User can ignore data size
- Message stream is bidirectional
- Overall, TCP is far easier for programmer for most applications
- But how does TCP do this?

Implementing Connections

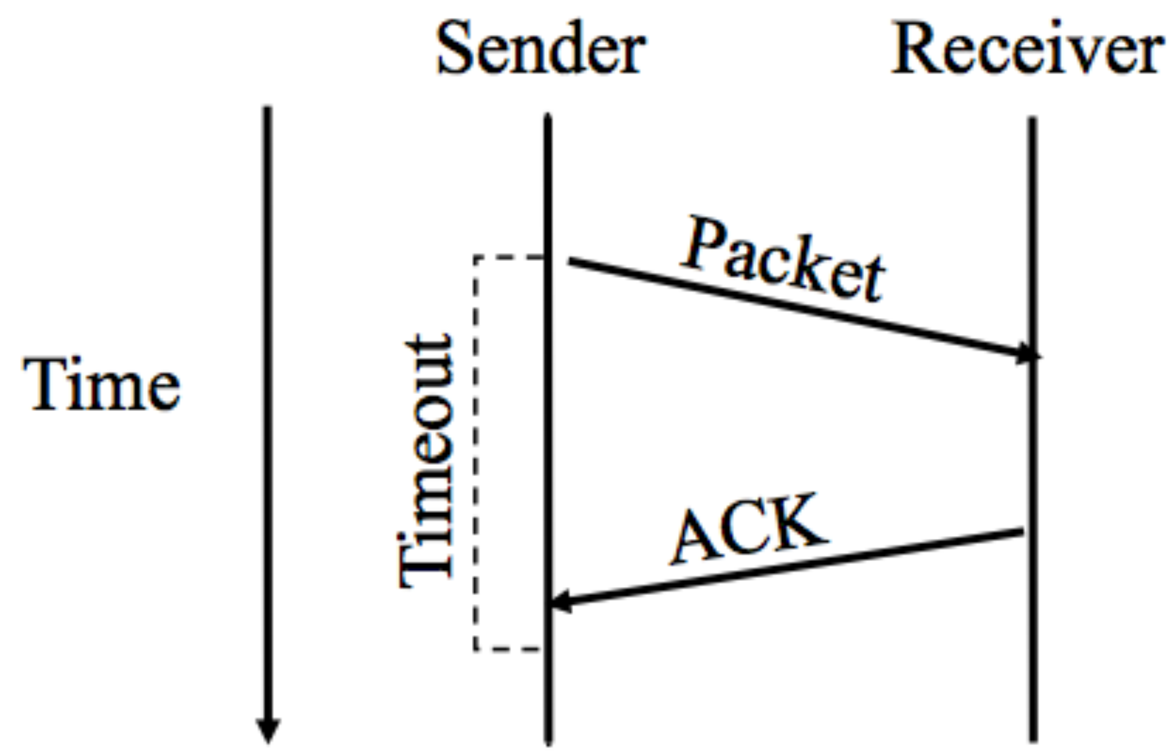
- Basic principle of reliable TCP connection is retransmission
  - Each packet has 32-bit Sequence Number
  - Every SeqNo is ACKed by receiver
  - When timeout expires, sender emits again
  - Simple!
- OK, maybe not quite that easy

TCP Delivery



Stop and Wait

- Send a packet, wait for ACK
- Receiver ACKs everything



TCP in action

- Watch HTTP traffic
  `tcpdump`

- If you like GUIs, try installing `wireshark`
  - For example, on OS X, download at:
  - https://www.wireshark.org/#download

- Note: you'll probably need sudo access to put your ethernet interface in promiscuous mode

Flow Control

- Any downside to Stop-and-Wait?

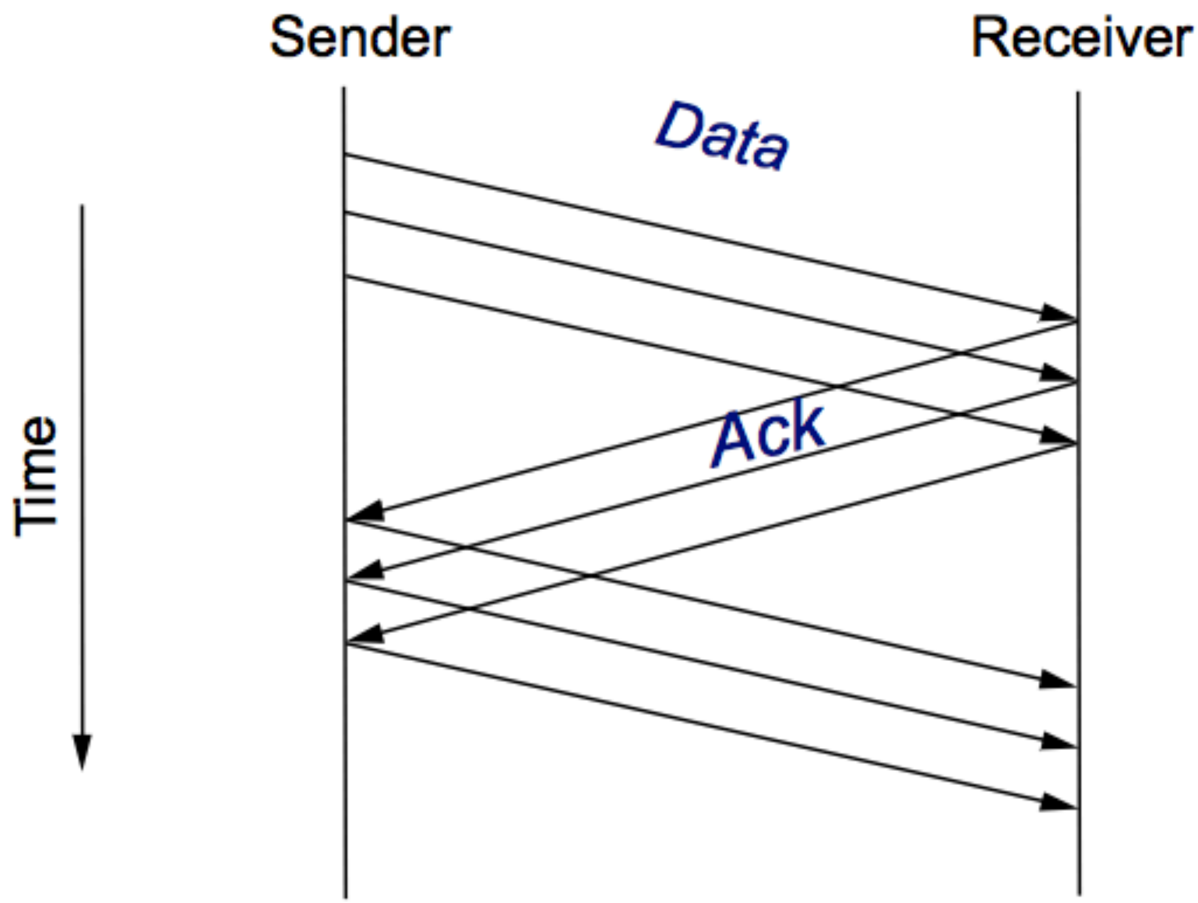
Flow Control

- Any downside to Stop-and-Wait?
- **Sliding window** technique for managing send/receive capacity
  - Receiver indicates “receive window” it is willing to buffer
  - Sender cannot have more packets in transit (unACKed) than what can fit in the window
  - Ideally, window is bandwidth * RT delay
    - Keeps as much data in transit as possible

Sliding Window

- Sliding Window algorithm has several roles
  - Reliable delivery, via ACKs, timeouts, and retransmissions
  - In-order delivery, via buffering and ACKing
  - Flow control, via advertised window for receiver

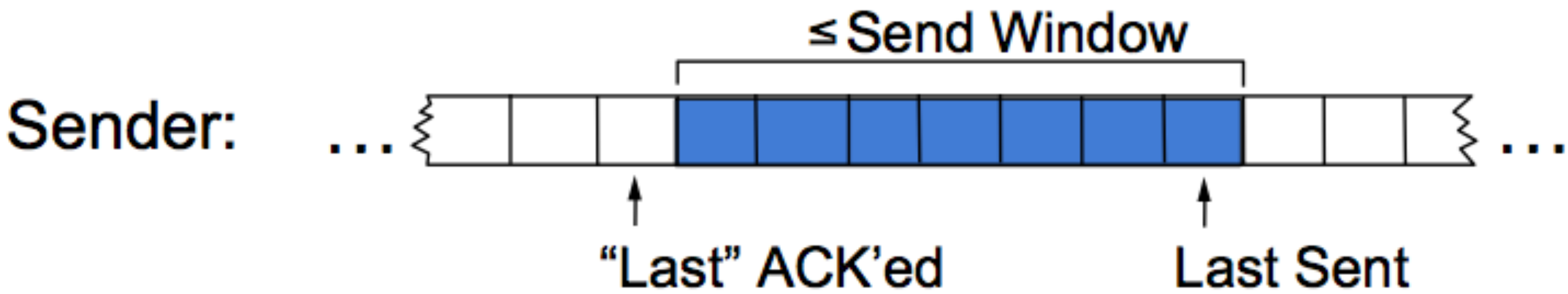
Sliding Window Ideal



Sliding Window Problems

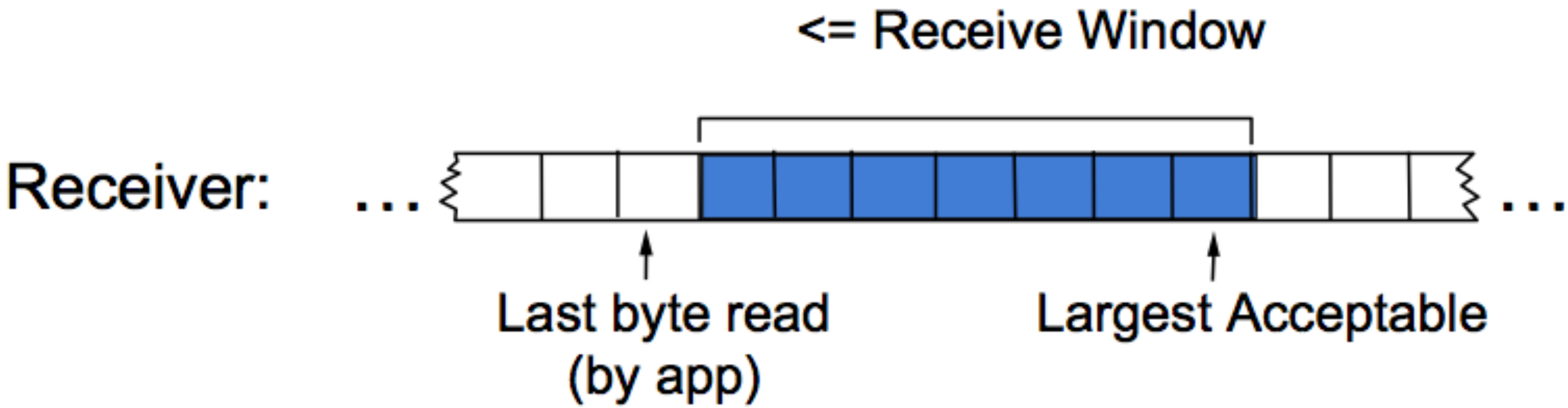
- What if sender transmits data too fast?
- Or receiver reads data too slow?

Sliding Window - Sender



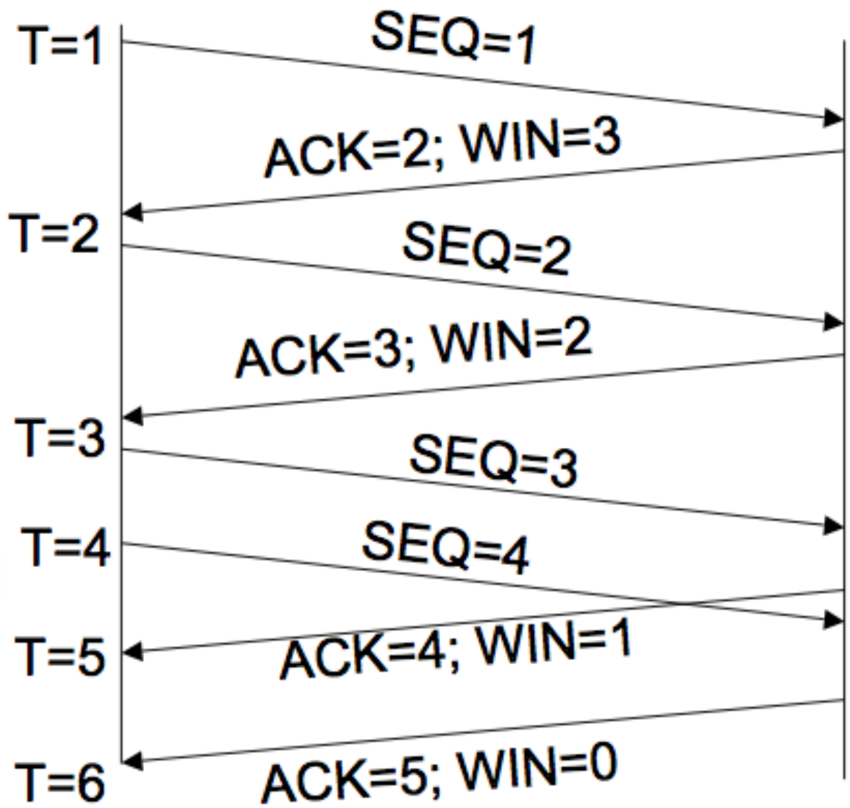
- Sender buffers unACKed data
- Only removes data from send buffer after it's been ACKed
- Send window determined by receiver's advertisements

Sliding Window - Receiver



- ACKs data as it arrives
- Removes data from buffer as app reads
- Also shrinks/expands advertised window in response to application behavior

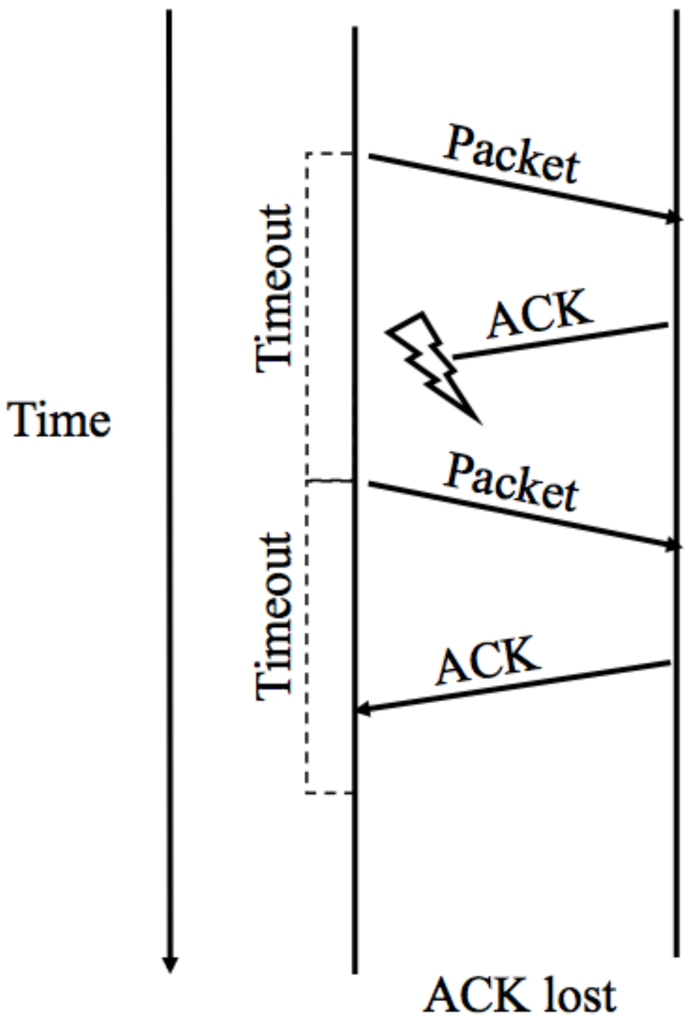
Sliding Window IN ACTION



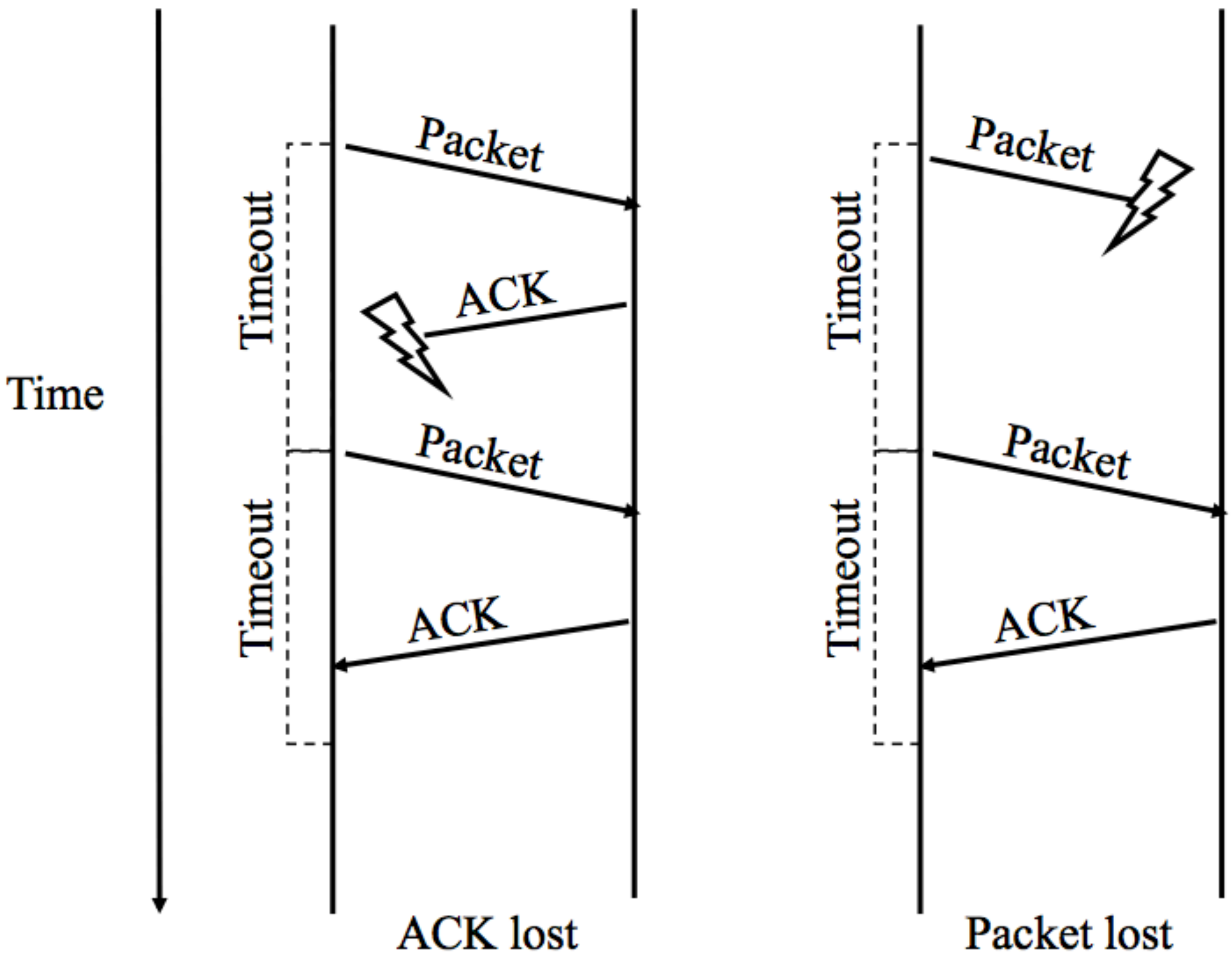
Stall due to flow control here

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

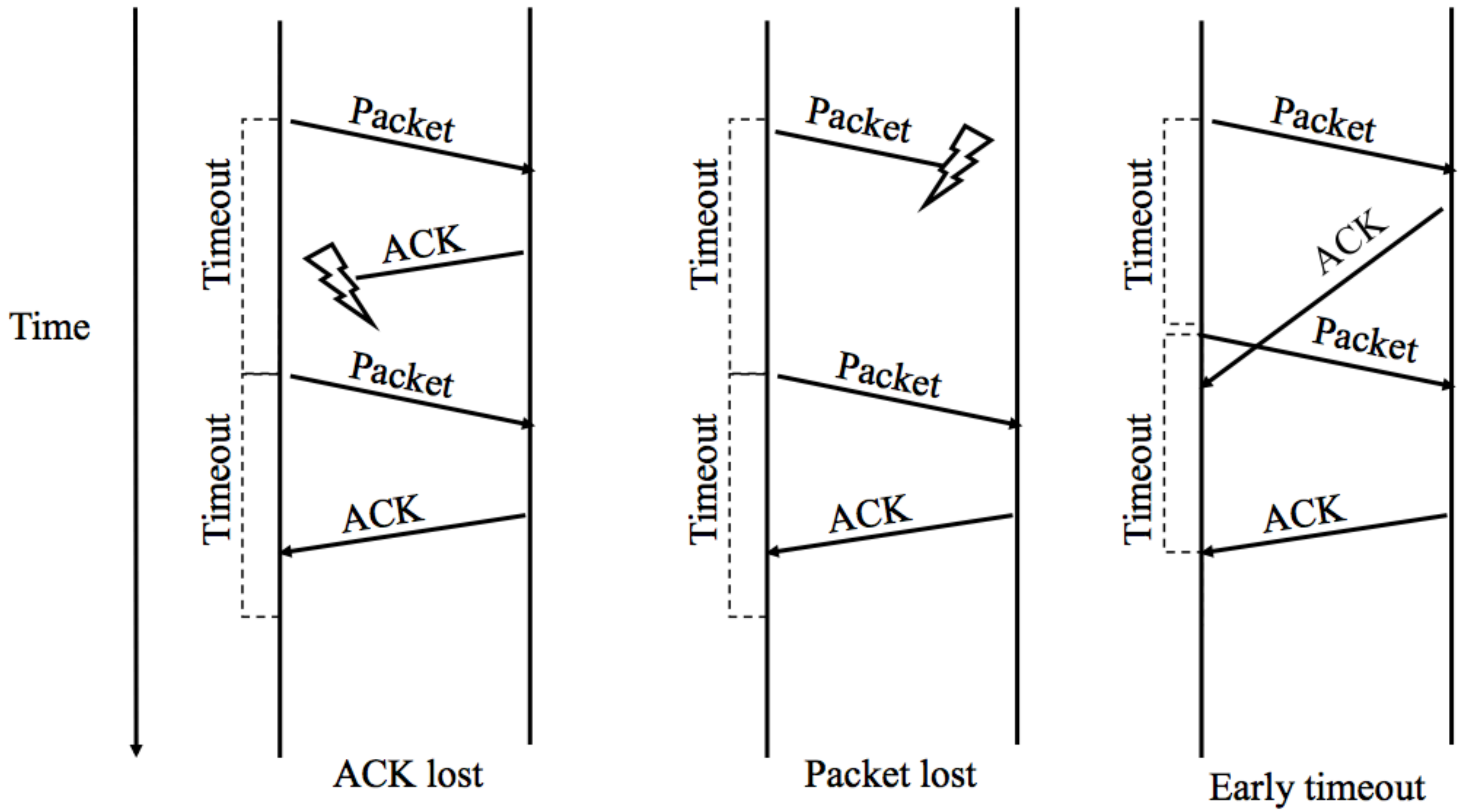
Recovering from errors



Recovering from errors

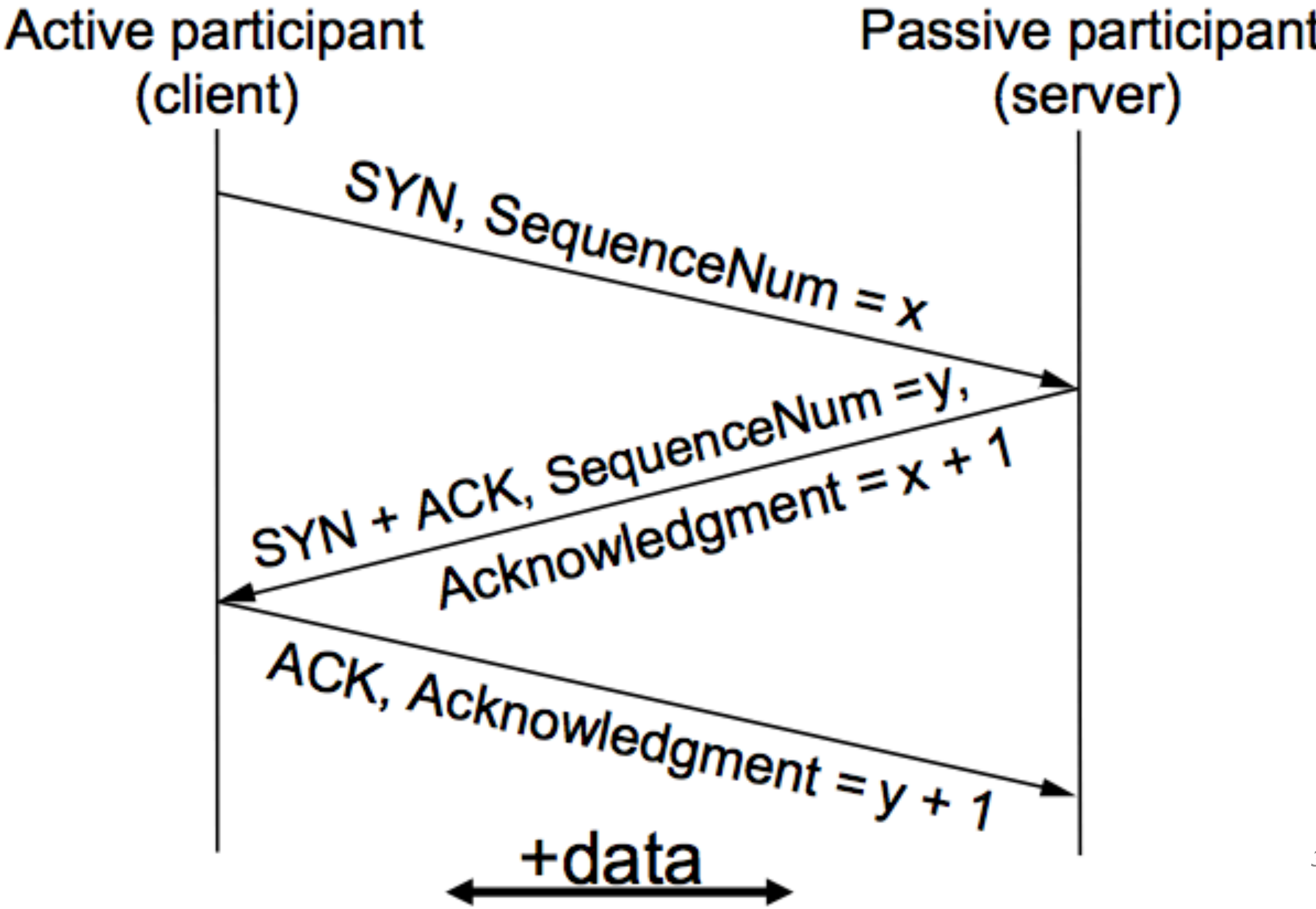


Recovering from errors



TCP Connection Setup

- The World Famous 3-way Handshake



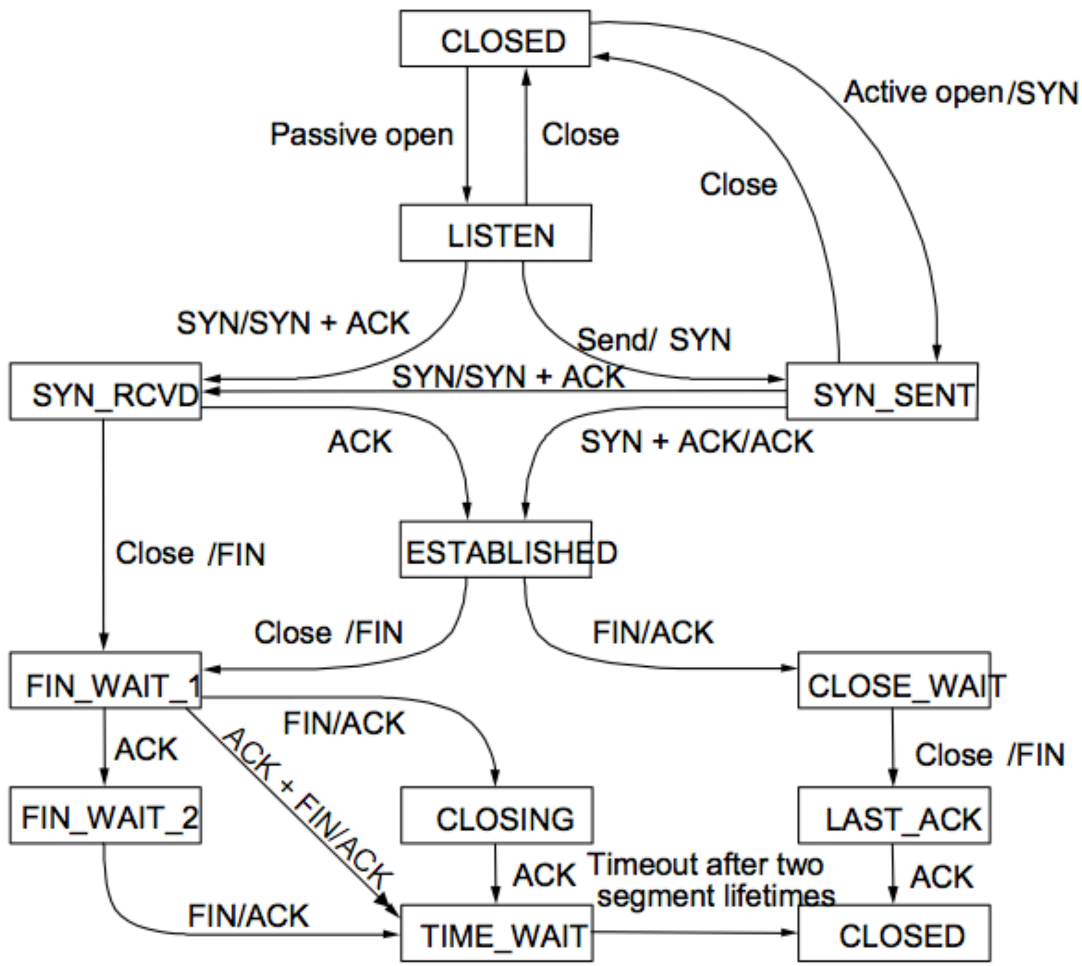
TCP in action

- Watch 3-way handshake
```
sudo tcpdump -S "tcp[tcpflags] & (tcp-syn|tcp-ack|tcp-fin) != 0"
```

- Let's capture a few packets while loading a web page
```
sudo tcpdump -S "host www.eecs.umich.edu and port 443 and (tcp[tcpflags] & (tcp-syn|tcp-ack|tcp-fin) != 0)" | tee packets.log
```

- Now browse to http://www.eecs.umich.edu/

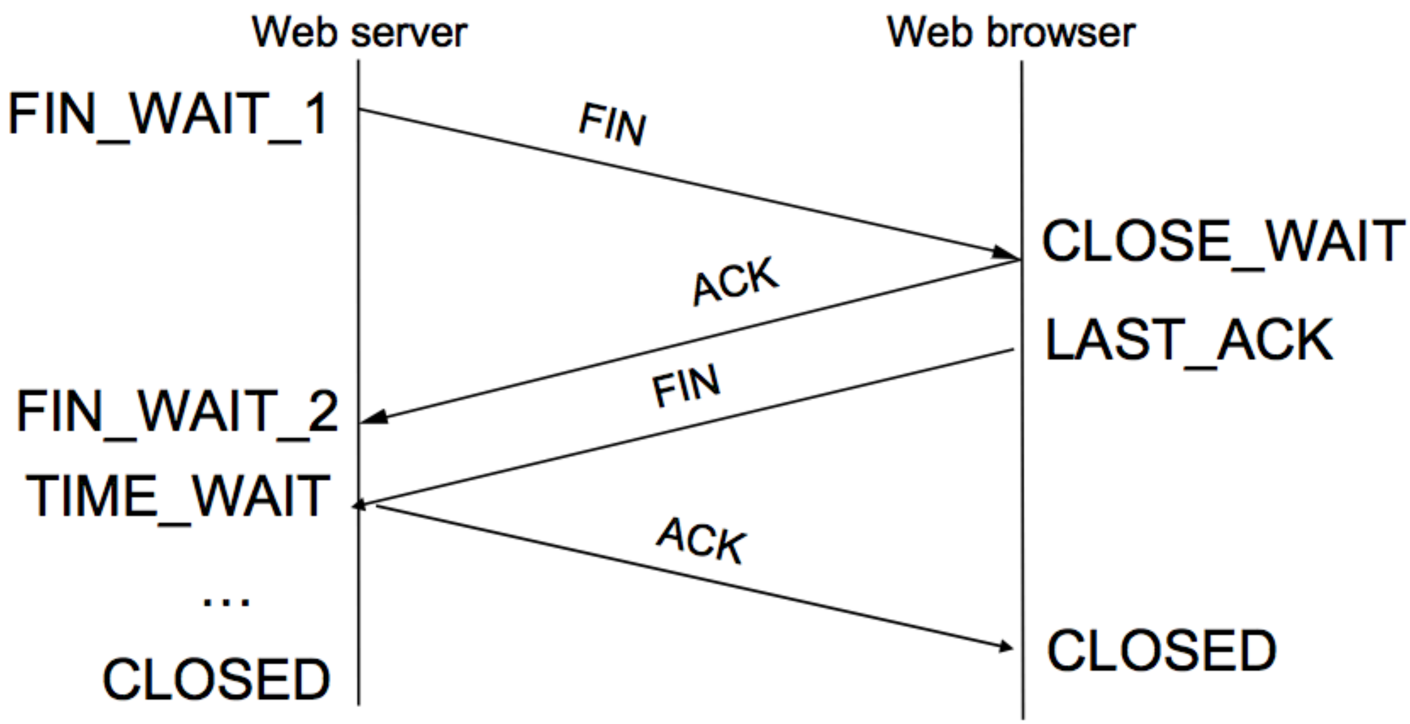
Transitions



Connection Teardown

- Orderly release by sender + receiver
  - “Hanging up the phone”
- Releases state on both sides

TCP Connection Teardown

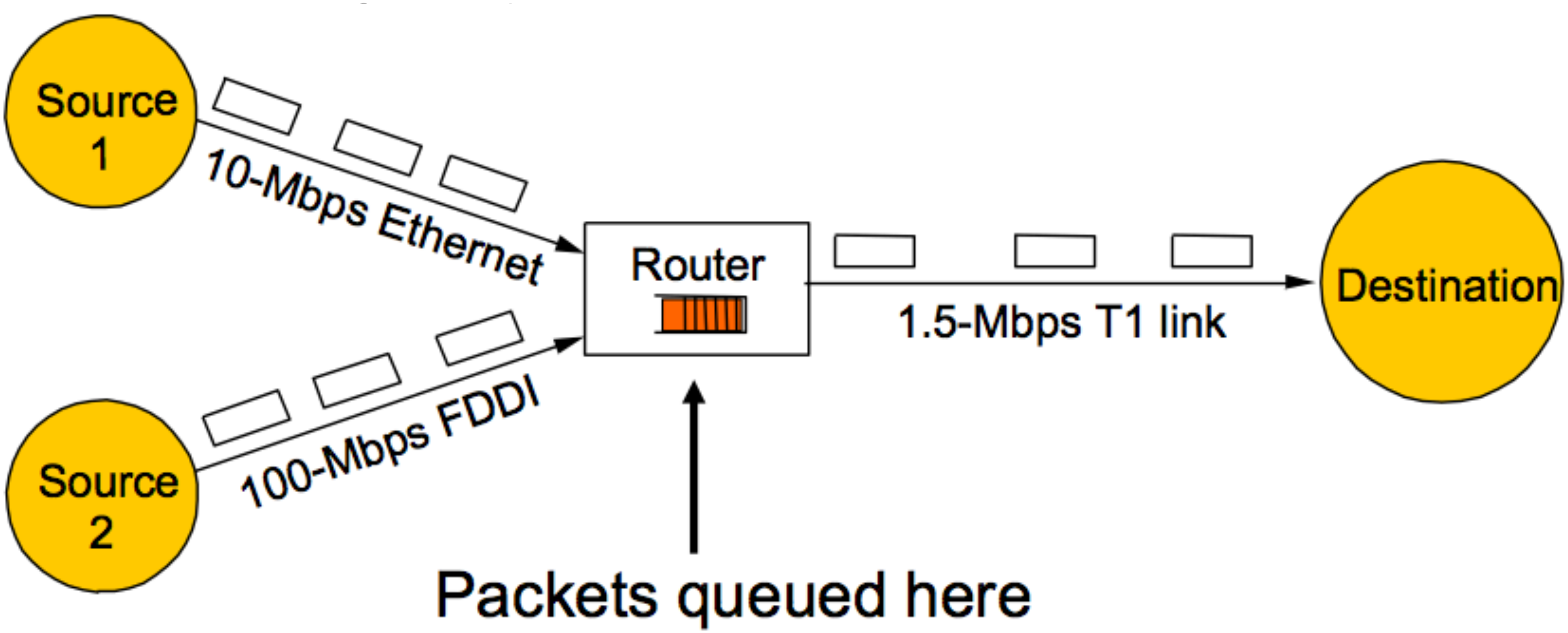


TCP/IP

- Two reasons for sender to slow down
  - We already discussed: Receiver can't handle input (and will have to drop packets)
  - Next we'll discuss: Network can't transmit as fast as incoming packets arrive
- Sliding window algorithm takes care of the receiver
  - Sender never transmits more than advertised window size win
- But what about the network?

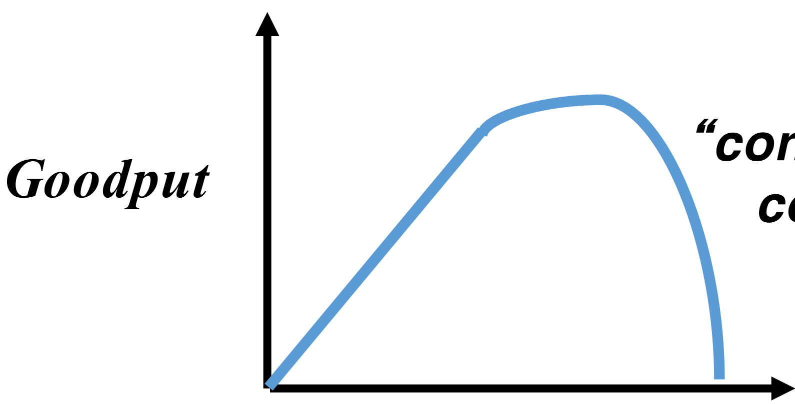
Part I: Congestion

- Buffers in middle of network can become overloaded



Congestion

- When has packet been lost?
  - Too long a timer, you're waiting pointlessly; too short, adds needless load
- Many retransmits can induce congestion collapse (it happened in late 1980s!)
  - Retransmits just add to congestion
  - Capacity of network falls dramatically

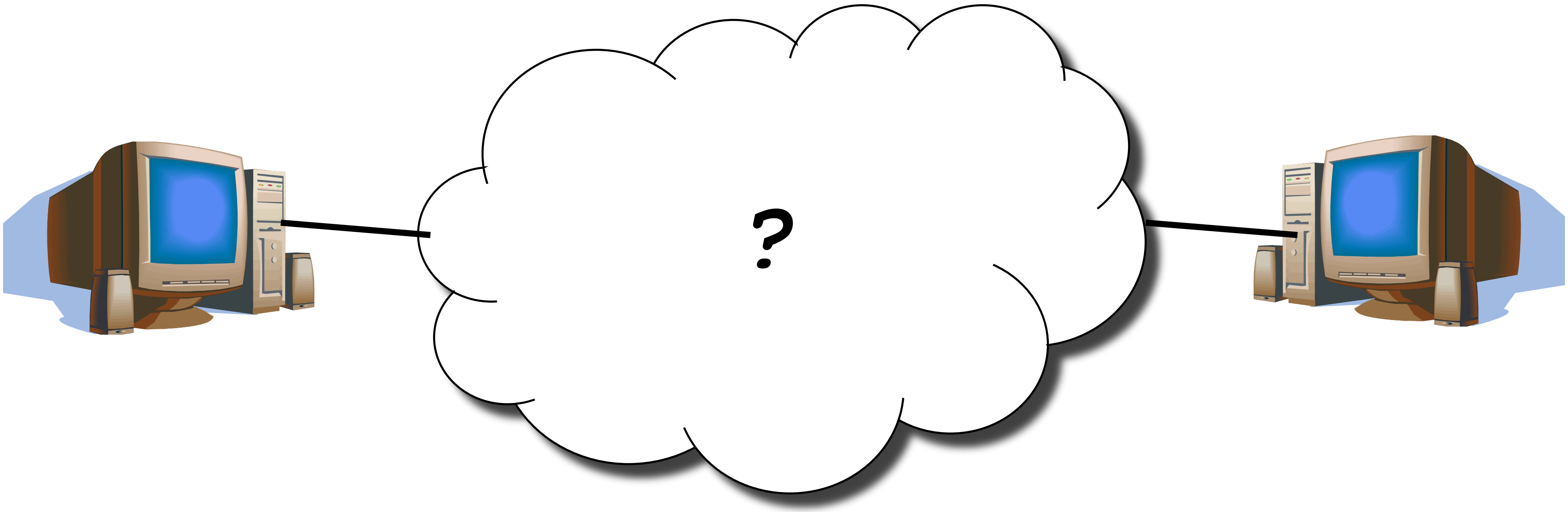


Increase in load that results in a decrease in useful work done.

Many Important Questions

- How does the sender know there is congestion?
  - Explicit feedback from the network?
  - Inference based on network performance?
- How should the sender adapt?
  - Explicit sending rate computed by the network?
  - End host coordinates with other hosts?
  - End host thinks globally but acts locally?
- What is the performance objective?
  - Maximizing goodput, even if some users suffer more?
  - Fairness? (Whatever the heck that means!)
- How fast should new TCP senders send?

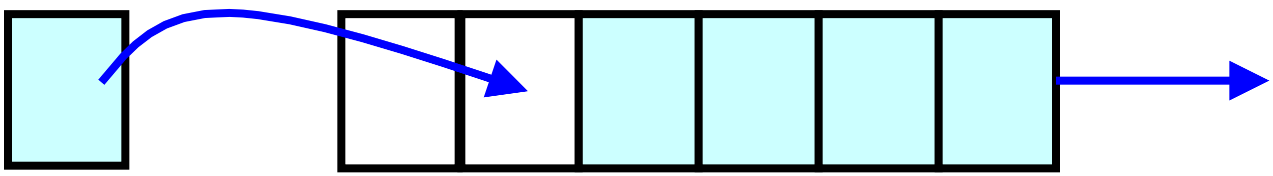
Inferring From Implicit Feedback



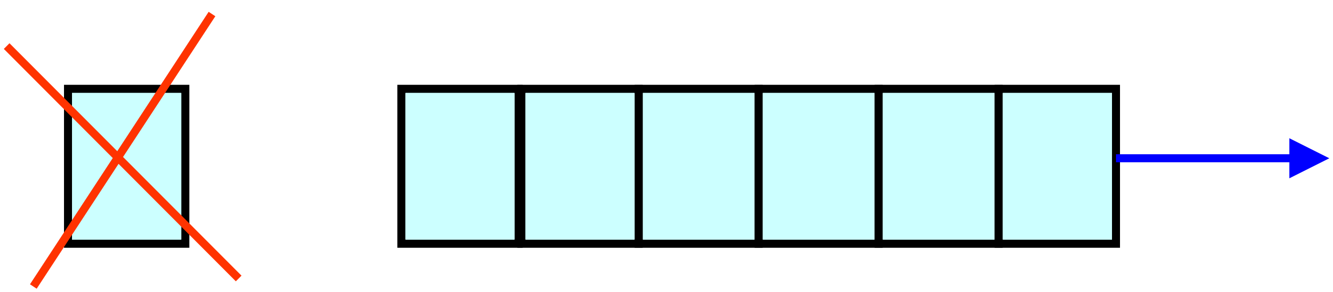
- What does the end host see?
- What can the end host change?

Where It Happens: Links

- Simple resource allocation: FIFO queue & drop-tail
- Access to the bandwidth: first-in first-out queue
  - Packets transmitted in the order they arrive



- Access to the buffer space: drop-tail queuing
  - If the queue is full, drop the incoming packet



How it Looks to the End Host

- Packet delay
  - Packet experiences high delay
- Packet loss
  - Packet gets dropped along the way
- How does TCP sender learn this?
  - Delay
    - Round-trip time estimate
  - Loss
    - Timeout
    - Duplicate acknowledgments

What Can the End Host Do?

- Upon detecting congestion (well, packet loss)
  - Decrease the sending rate
- But, what if conditions change?
  - Suppose there is more bandwidth available
  - Would be a shame to stay at a low sending rate
- Upon not detecting congestion
  - Increase the sending rate, a little at a time
  - And see if the packets are successfully delivered

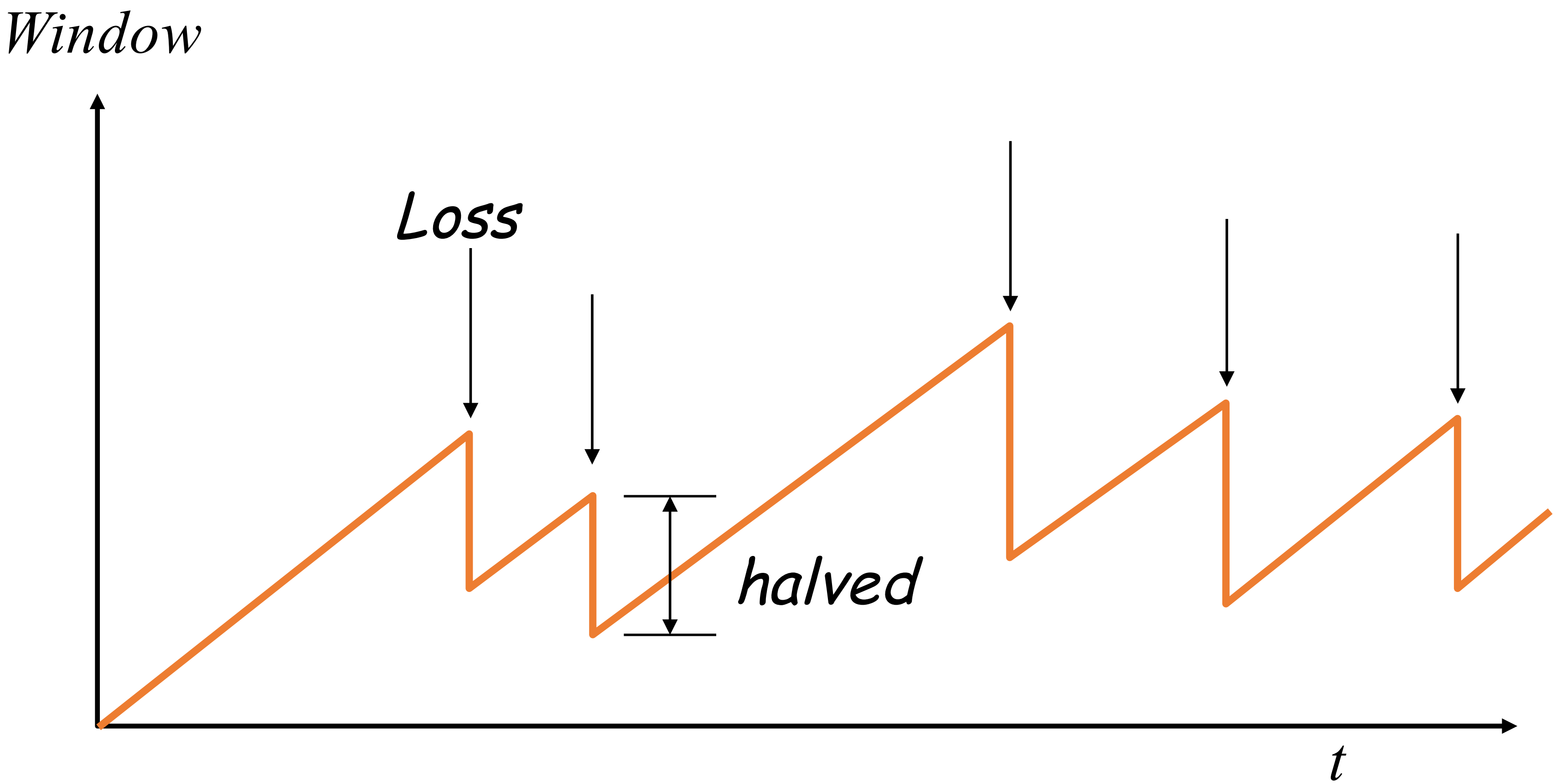
TCP Congestion Window

- Each TCP sender maintains a congestion window
  - Maximum number of bytes to have in transit
  - I.e., number of bytes still awaiting acknowledgments
- Adapting the congestion window
  - Decrease upon losing a packet: backing off
  - Increase upon success: optimistically exploring
  - Always struggling to find the right transfer rate
- Both good and bad
  - Pro: avoids having explicit feedback from network
  - Con: under-shooting and over-shooting the rate

Step 1: AIMD

- Additive Increase Multiplicative Decrease
- Why AIMD? Network load is really hard to get rid of: oversubscribed link must be undersubscribed to dissipate queue
  - Over: packets dropped and retransmitted
  - Under: somewhat lower throughput
- Want to be extremely cautious senders
- AIMD algorithm:
- After packet timeout, cwnd = cwnd/2
- If none, cwnd += 1 every RTT
- Sender always xmits min(win, cwnd)

AIMD Sawtooth



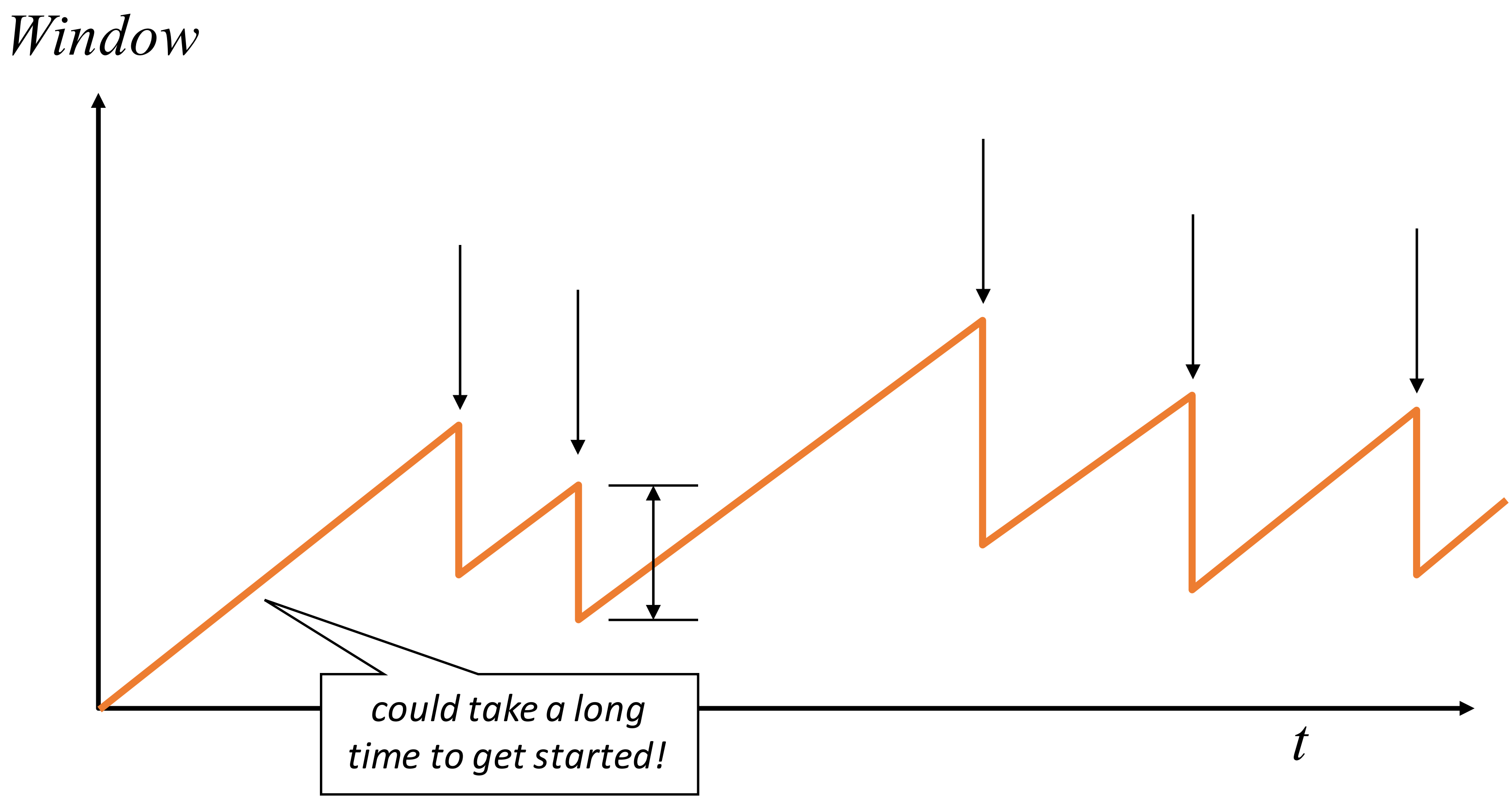
Receiver vs. Congestion Windows

- Flow control
  - Keep a *fast sender* from overwhelming a *slow receiver*
- Congestion control
  - Keep a *set of senders* from overloading the *network*

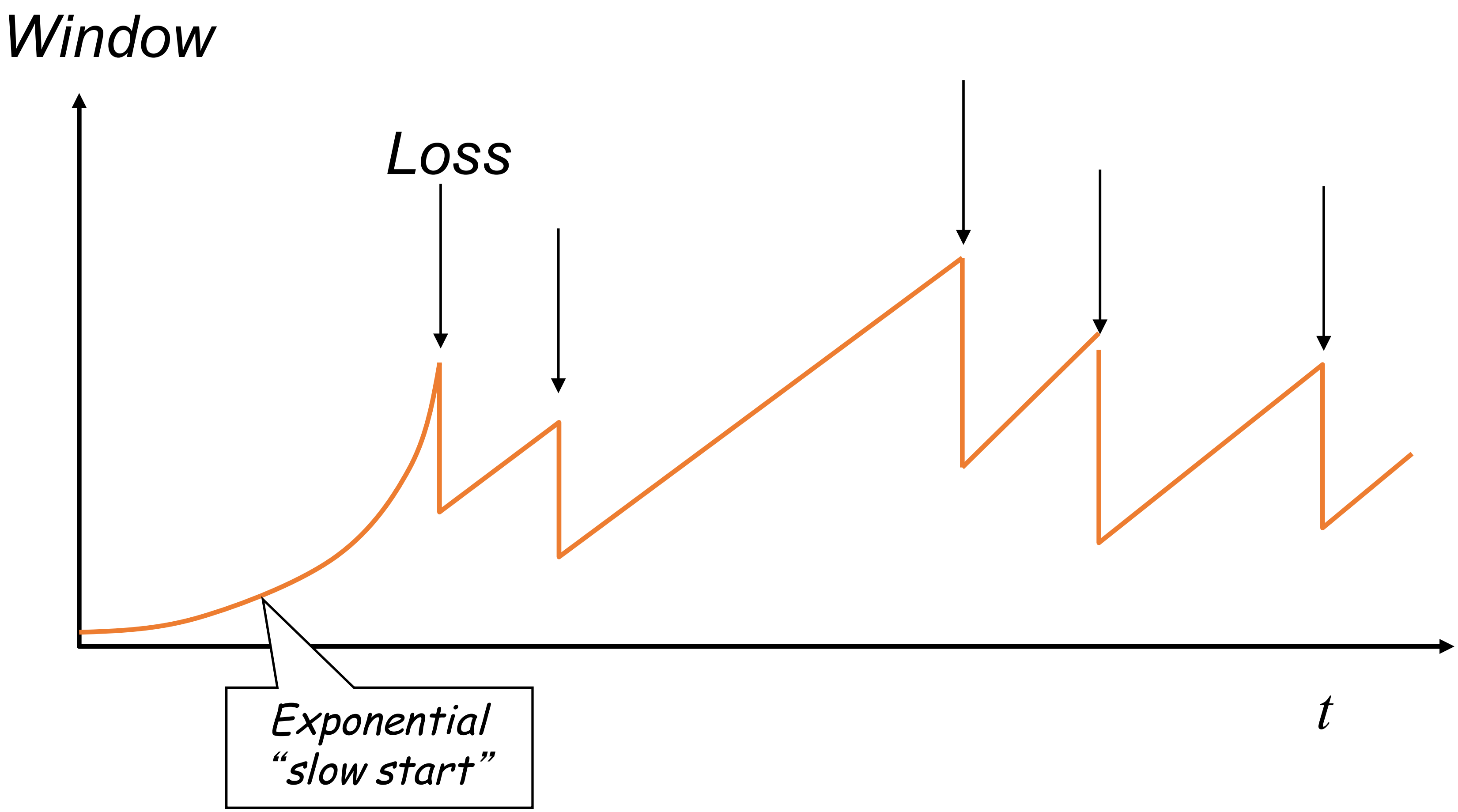
- Different concepts, but similar mechanisms
  - TCP flow control: receiver window
  - TCP congestion control: congestion window
  - TCP window: min { congestion window, receiver window }

How Should a New Flow Start

- Need to start with a small CWND to avoid overloading the network.



Slow Start and TCP Sawtooth



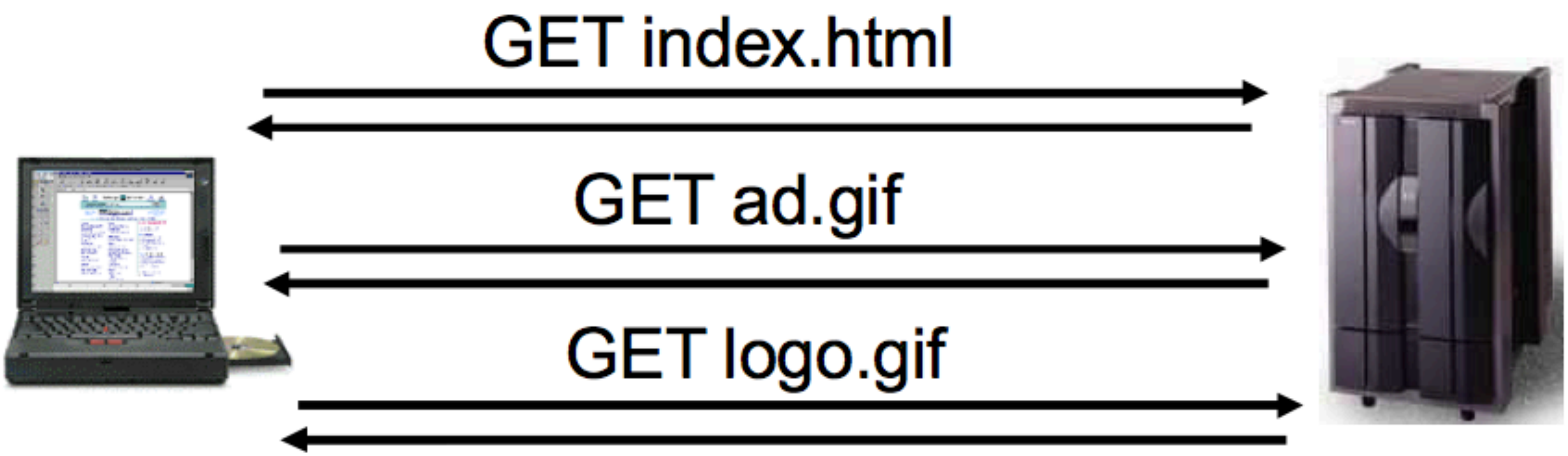
Why is it called slow-start? Because TCP originally had no congestion control mechanism. The source would just start by sending a whole receiver window's worth of data.

HTTP on TCP

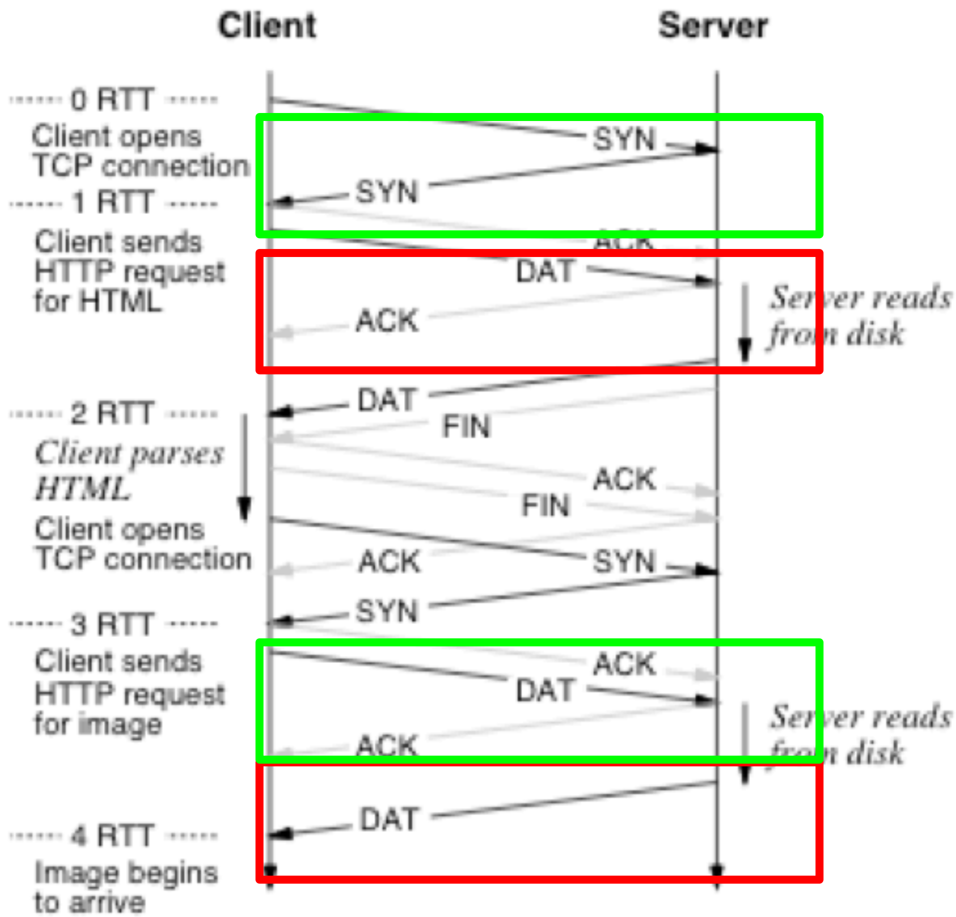
- TCP Summary
  - A roundtrip for each window of bytes
  - Takes some time to find best rate
  - OS buffer overhead for each connection
- HTTP Summary
  - One HTTP roundtrip per TCP connection
  - Connections short-lived
  - Small amounts of data per connection
  - Many connections per HTML page

HTTP 1.0 on TCP

- Every HTML-embedded item requires a GET



HTTP on TCP



Client
Server
0 RTT
Client opens TCP connection
SYN
SYN
1 RTT
Client sends HTTP request for HTML
ACK
DAT
ACK
Server reads from disk
DAT
2 RTT
FIN
Client parses HTML
ACK
FIN
Client opens TCP connection
ACK
SYN
SYN
3 RTT
Client sends HTTP request for image
ACK
DAT
ACK
Server reads from disk
DAT
4 RTT
Image begins to arrive

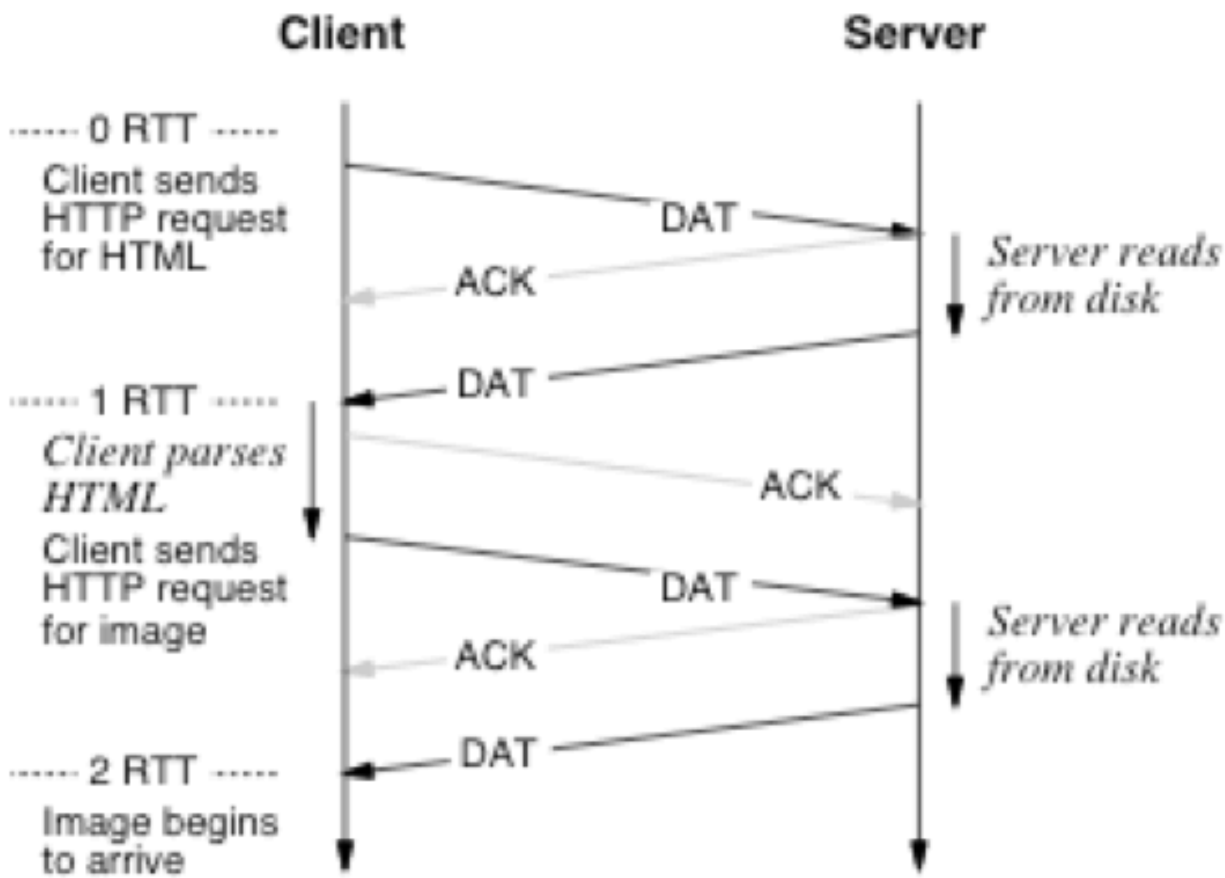
HTTP 1.0

- Naïve HTTP on TCP yields awful performance
- Why?
  - Many TCP conn creation roundtrips
  - Lots of slow-start delays

HTTP/1.1

- Persistent Connections
  - Server does not close the connection after sending the response
  - Client can re-use it
    - Especially improves small objects
    - Makes parallel downloads difficult
- Pipelining
  - Many HTTP requests can be “live” at once, on the same persistent connection
  - Send lots of HTTP requests at once, then get lots of answers
  - Server replies in the order received

HTTP/1.1 on TCP



HTTP 1.1 Performance

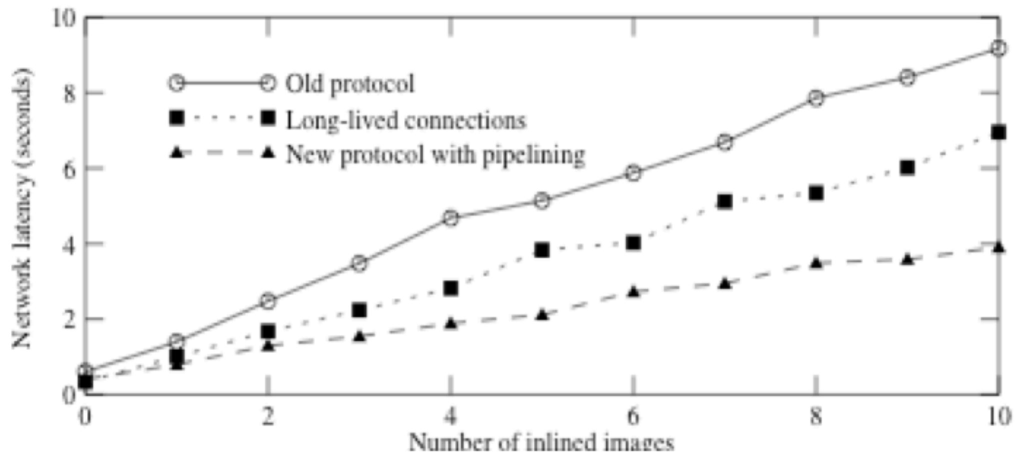


Image size=2544

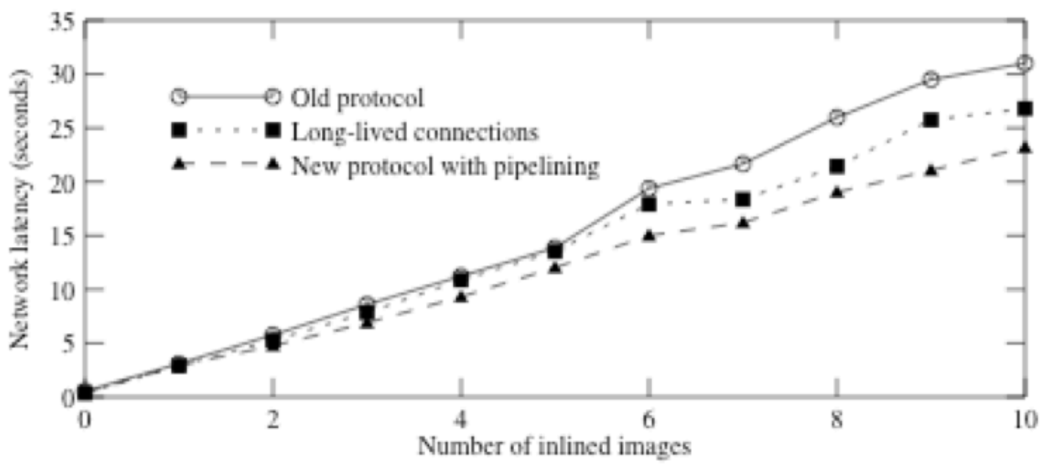


Image size=45566

HTTP/1.1

- Also, caching
  - GET + IF-MODIFIED-SINCE <timestamp>
  - When combined with browser cache, eliminates a lot of unneeded data transfer
  - Same number of roundtrips
  - Lots of different caches possible
    - Browser, department proxy, Akamai