CS162
Operating Systems and Systems Programming
Lecture 23

Networking and TCP/IP

Recall: The promise of distributed systems

Availability

Proportion of time system is in functioning condition

=> One machine goes down, use another

Fault-tolerance

System has well-defined behaviour when fault occurs

=> Store data in multiple locations

Scalability

Ability to add resources to system to support more work

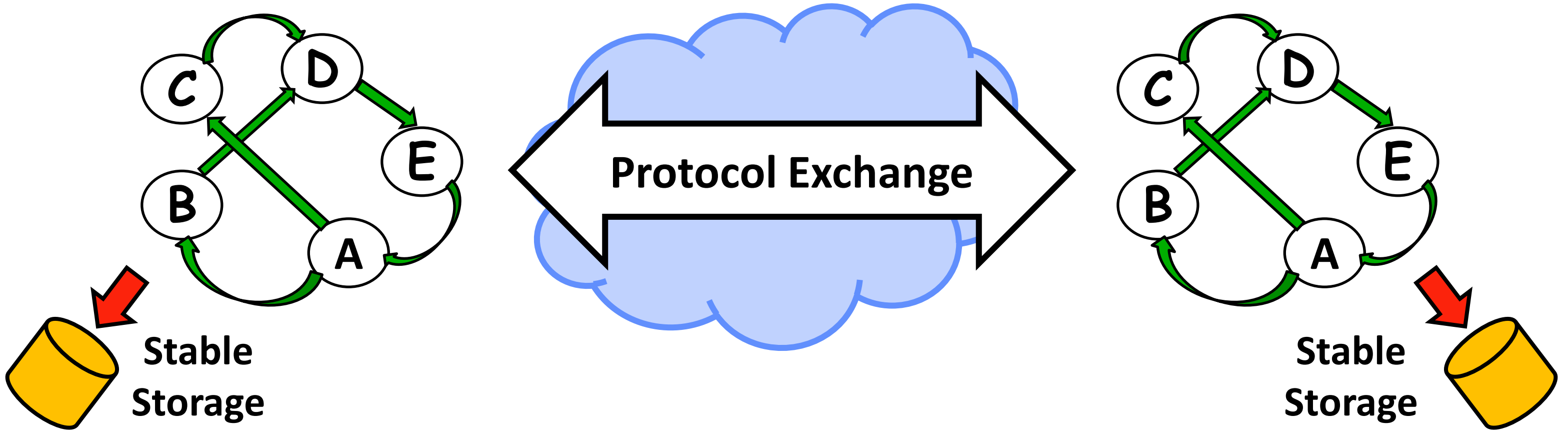
=> Just add machines when need more storage/processing power

Recall: Agreeing simultaneously: General's Paradox

- General's paradox:
  - Constraints of problem:
    - » Two generals, on separate mountains
    - » Can only communicate via messengers
    - » Messengers can be captured
  - Problem: need to coordinate attack
    - » If they attack at different times, they all die
    - » If they attack at same time, they win

Can messages over an unreliable network be used to guarantee two entities do something simultaneously?

Recall: How do entities communicate? A Protocol!



- A protocol is an agreement on how to communicate, including:
  - Syntax: how a communication is specified & structured
    - Format, order messages are sent and received
  - Semantics: what a communication means
    - Actions taken when transmitting, receiving, or when a timer expires

Recall: Eventual Agreement: Two-Phase Commit

Goal: determine whether should commit or abort a transaction

- All processes that reach a decision reach the same one (*Agreement*)
- A process cannot reverse its decision after it has reached one (*Finality*)
- If there are no failures and every process votes yes, the decision will be commit (*Consistency*)
- If all failures are repaired and there are no more failures, then all processes will eventually decide commit/abort (*Termination*)

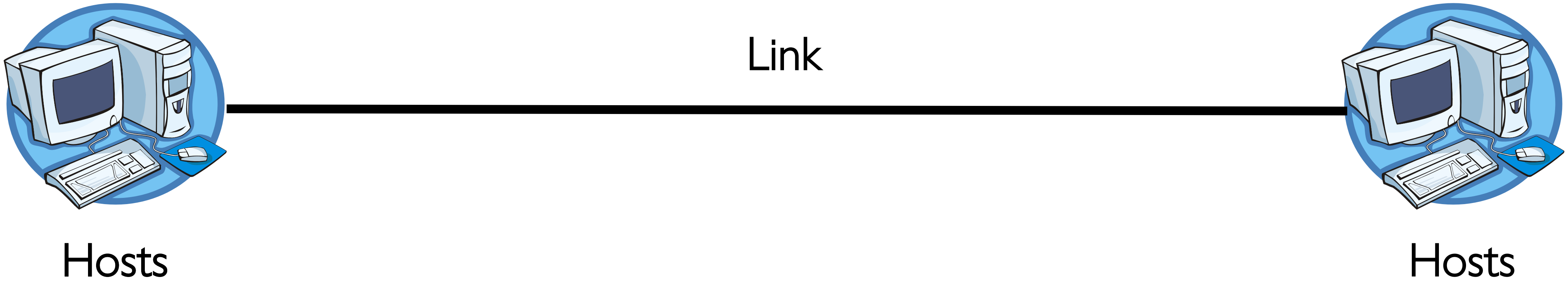
Recall: The Internet

- Layering
  - Layers “abstract” away hardware so that upper layers are agnostic to lower layers
  - The Hourglass shape
- The End-To-End Principle
  - Think twice before implementing functionality in the network
  - If hosts can implement functionality correctly, implement it in a lower layer only as a performance enhancement
  - But do so only if it does not impose burden on applications that do not require that functionality

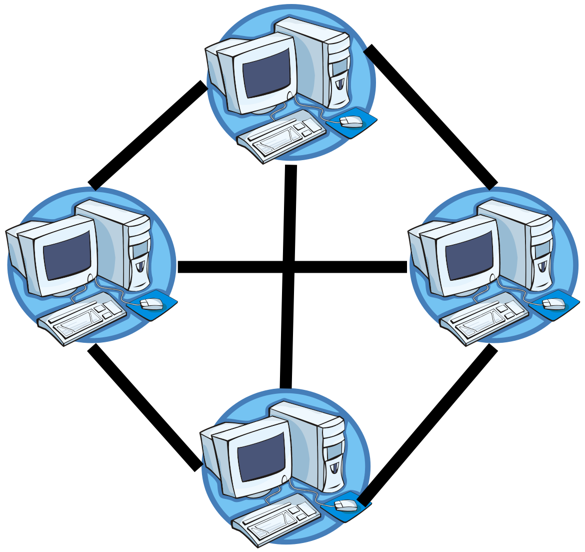
The Internet: Goals

- Robust to failures
- Support multiple types of delivery services (copper, optic, wireless)
- Accommodate a variety of networks
- Allow distributed management
- Easy host attachment
- Cost effective

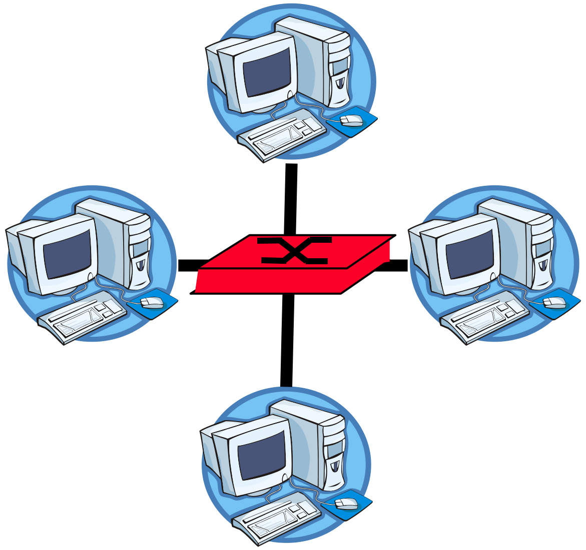
The Internet Through Graphs



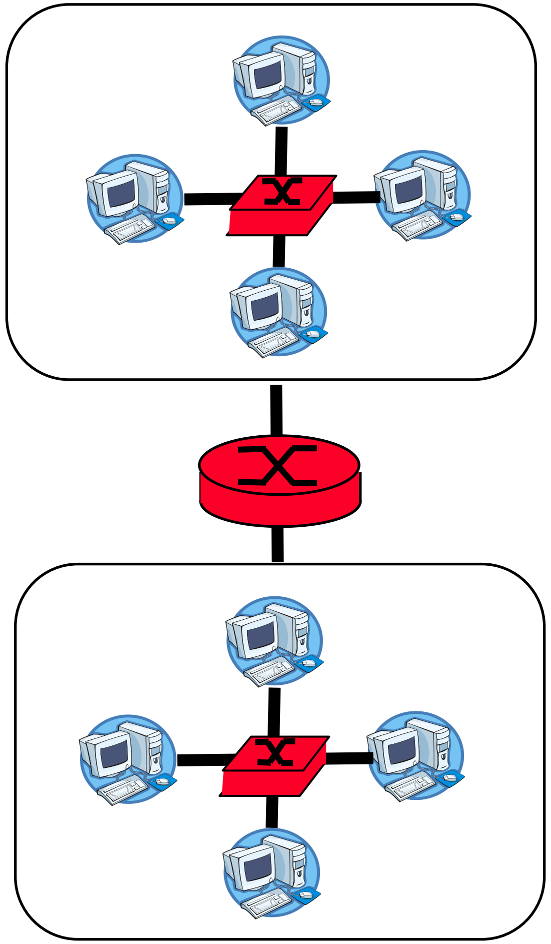
The Internet Through Graphs



Point-to-point links

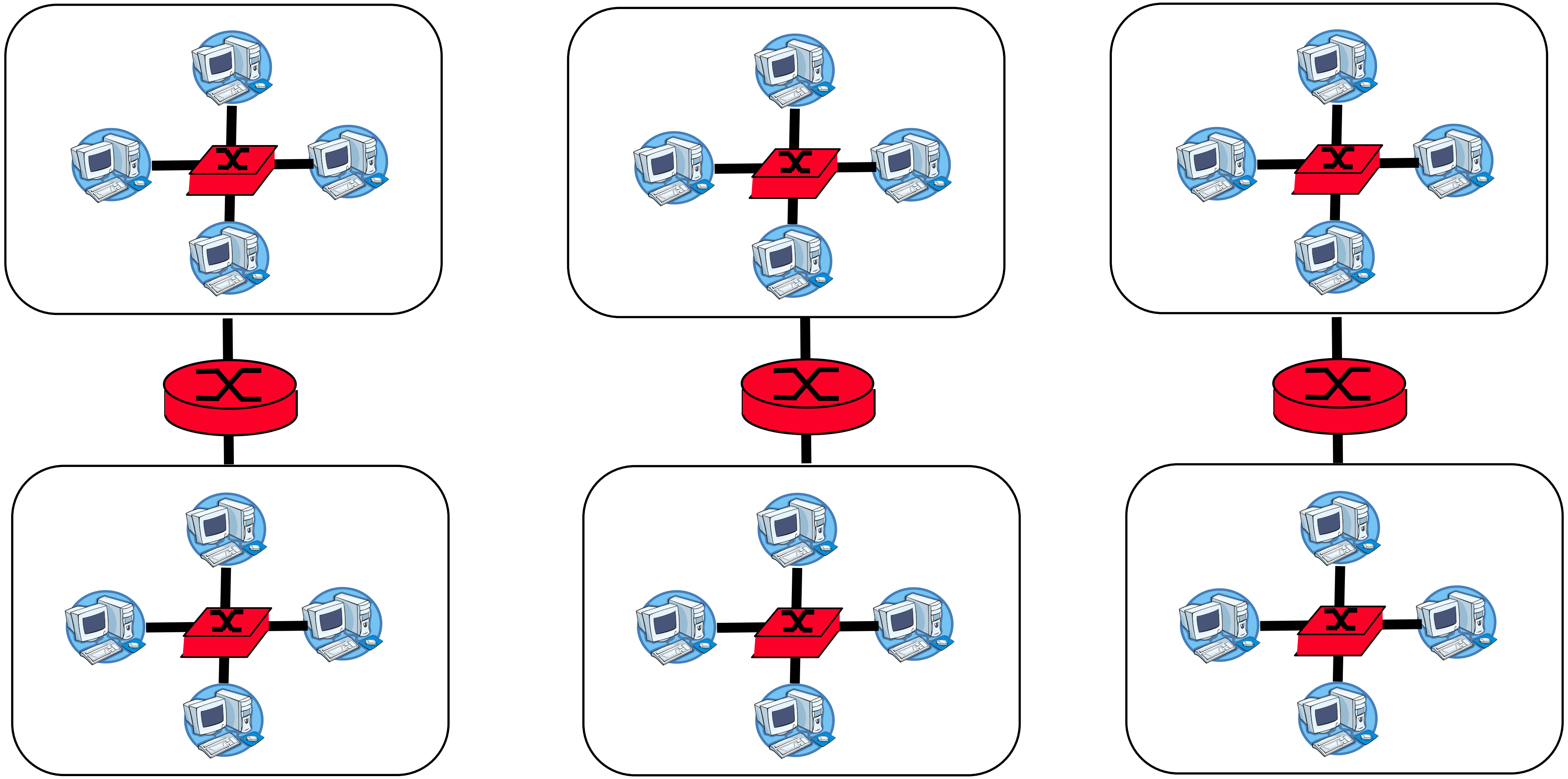


Share network links **switches**



Link local network through **routers**

The Internet: Networks of Networks



Hierarchy of networks
Scales to billions of hosts

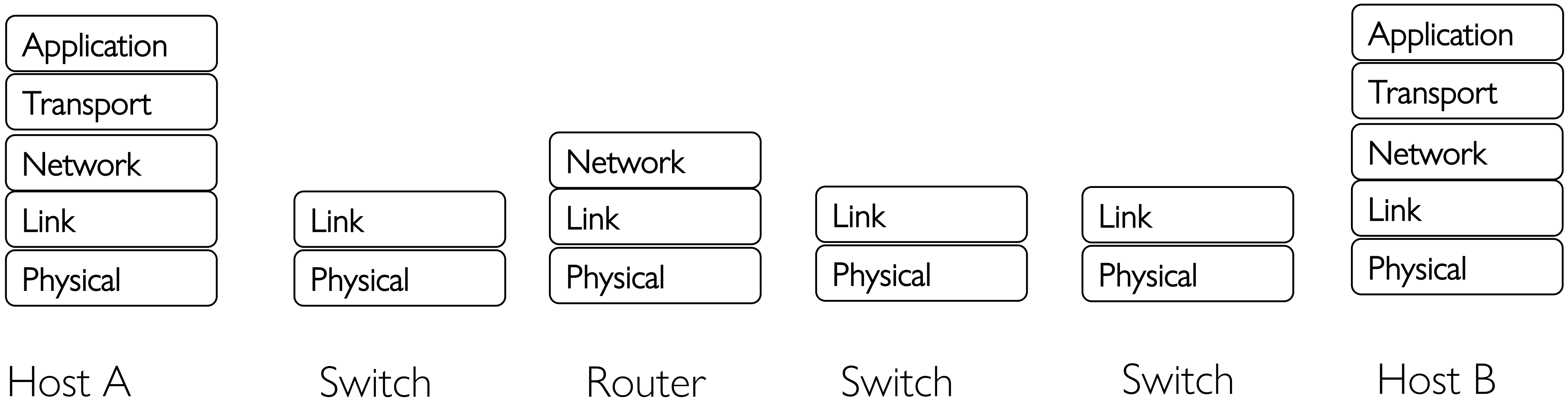
Layers, Layers, Layers

Application	Applications	HTTP , FTP , TLS/SSL
Transport	Reliable (or unreliable transport)	TCP , UDP
Network	Best-effort global packet delivery	IPv4, IPv6
Link	Best-effort local packet delivery	Ethernet, Wi-Fi
Physical	Physical transfer of bits	Coaxial, fiber optics, copper wires

Internet Entities

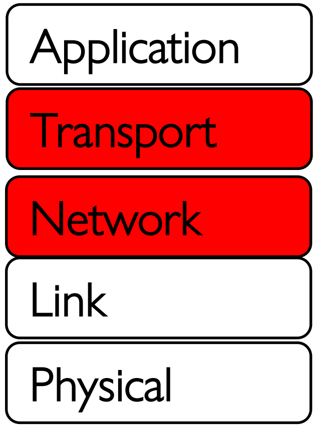
- Hosts
  - Implements all layers
  - Bits arrive on the wire, must make it up to the application
- Switches
  - Implement the physical and data layer.
  - Only responsible for transferring data within a small network
- Routers
  - Implement the physical, the data, and the network layer
  - Responsible for routing packets *across* networks

The life of a message



In this lecture

Focus on the transport and the network layer

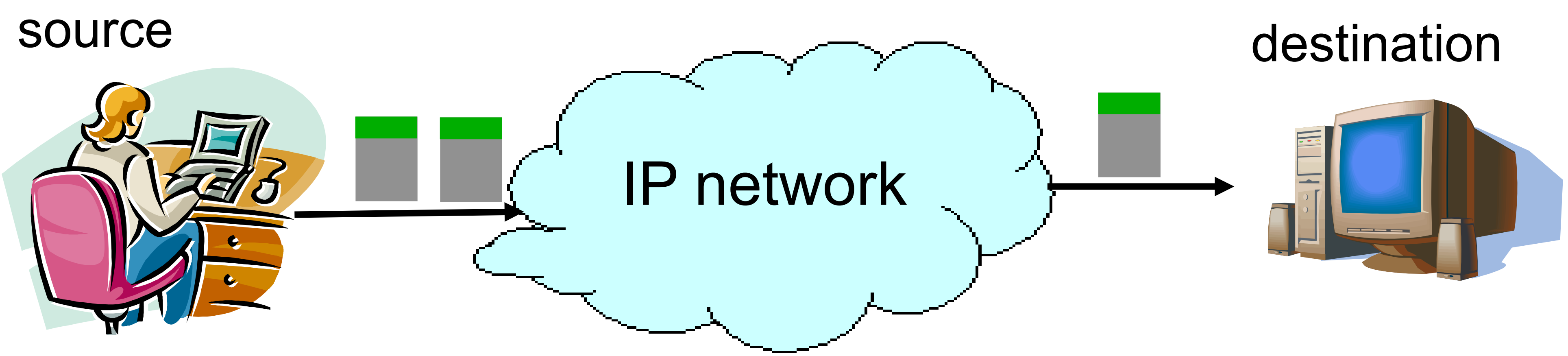


Take CS168 to learn more about the other layers!

The Internet Protocol (IP)

- Internet Protocol: Internet's network layer

- Service it provides: "Best-Effort" Packet Delivery
  - Tries it's "best" to deliver packet to its destination
  - Packets may be lost
  - Packets may be corrupted
  - Packets may be delivered out of order



What's an IP(v4) address?

- IP Address: a 32-bit integer used as destination of IP packet
  - Often written as four dot-separated integers, with each integer from
  - Example CS file server is: 169.229.60.83

- Host has one or more IP addresses used for routing

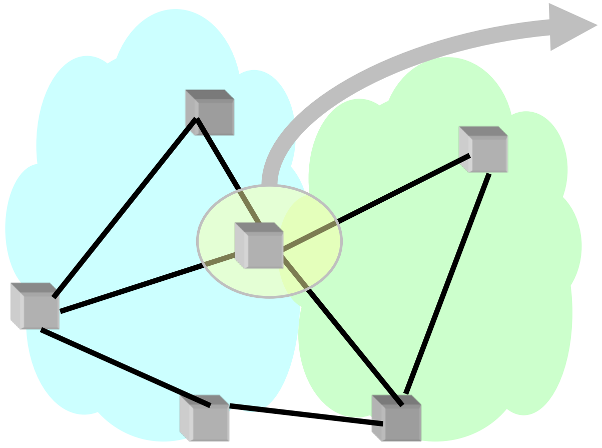
- Subnet: network connecting hosts with related IP addresses
  - A subnet is identified by 32-bit value, with the bits which differ set to zero, followed by a slash and a mask
    - » Example: 128.32.131.0/24 designates a subnet in which all the addresses look like 128.32.131.XX
  - Network of networks can be viewed as network of subnets

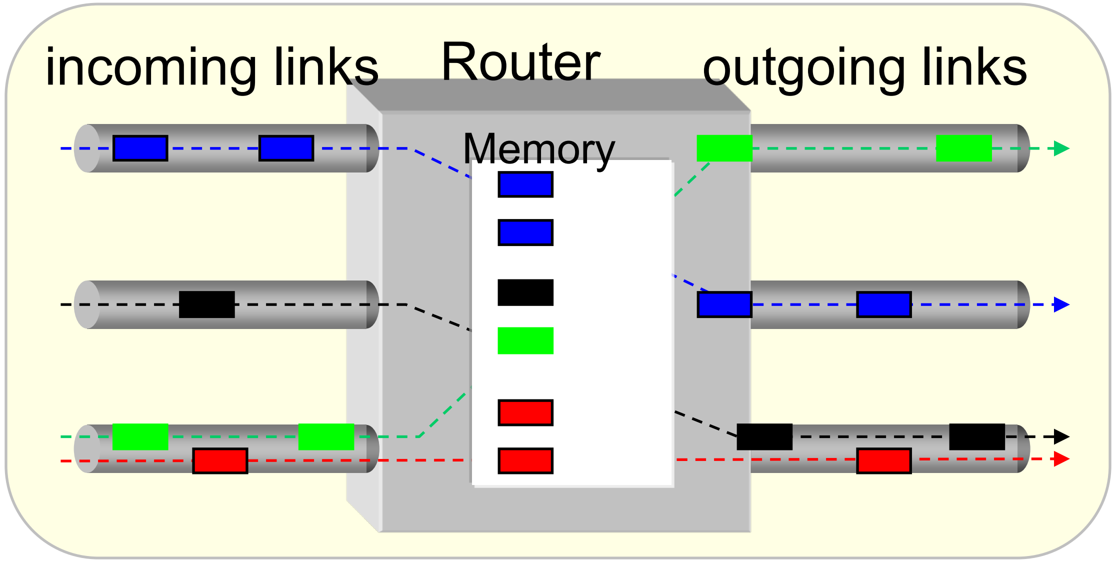
- How to get from IP 169.229.60.83 to 152.117.65.11?

Routers

- **Forward** each packet received on an **incoming link** to an **outgoing link** based on packet's destination IP address (towards its destination)

- **Forwarding table**: mapping between IP address and the output link

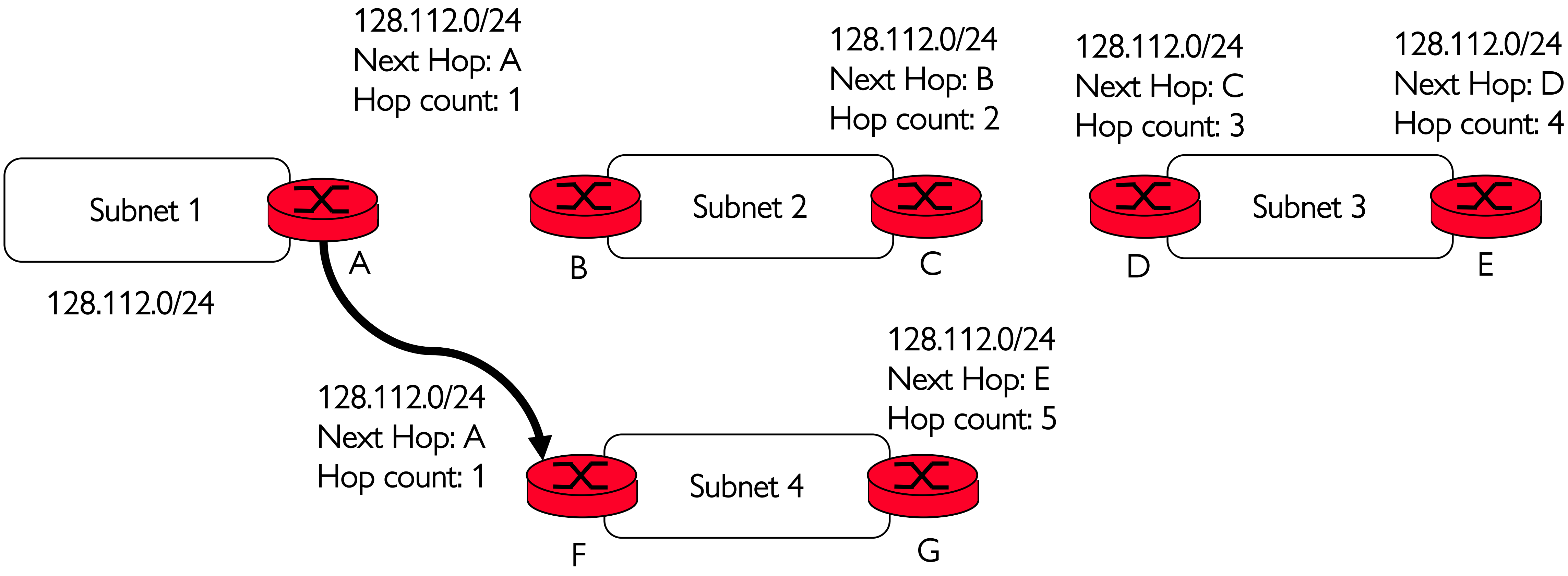




Setting up Routing Tables

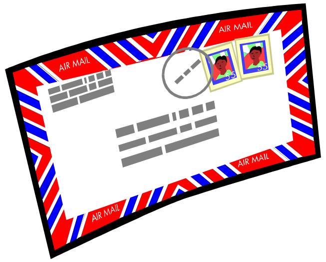
- How do you set up routing tables?
  - Internet has no centralized state!
    - » No single machine knows entire topology
    - » Topology constantly changing (faults, reconfiguration, etc.)
  - Need dynamic algorithm that acquires routing tables
    - » Ideally, have one entry per subnet or portion of address
    - » Could have "default" routes that send packets for unknown subnets to a different router that has more information
- Exchange routing information with neighbouring peers
  - Inform peers of best routes it knowns to reach a particular subnet!
- For more detail BGP!

Setting up routing tables



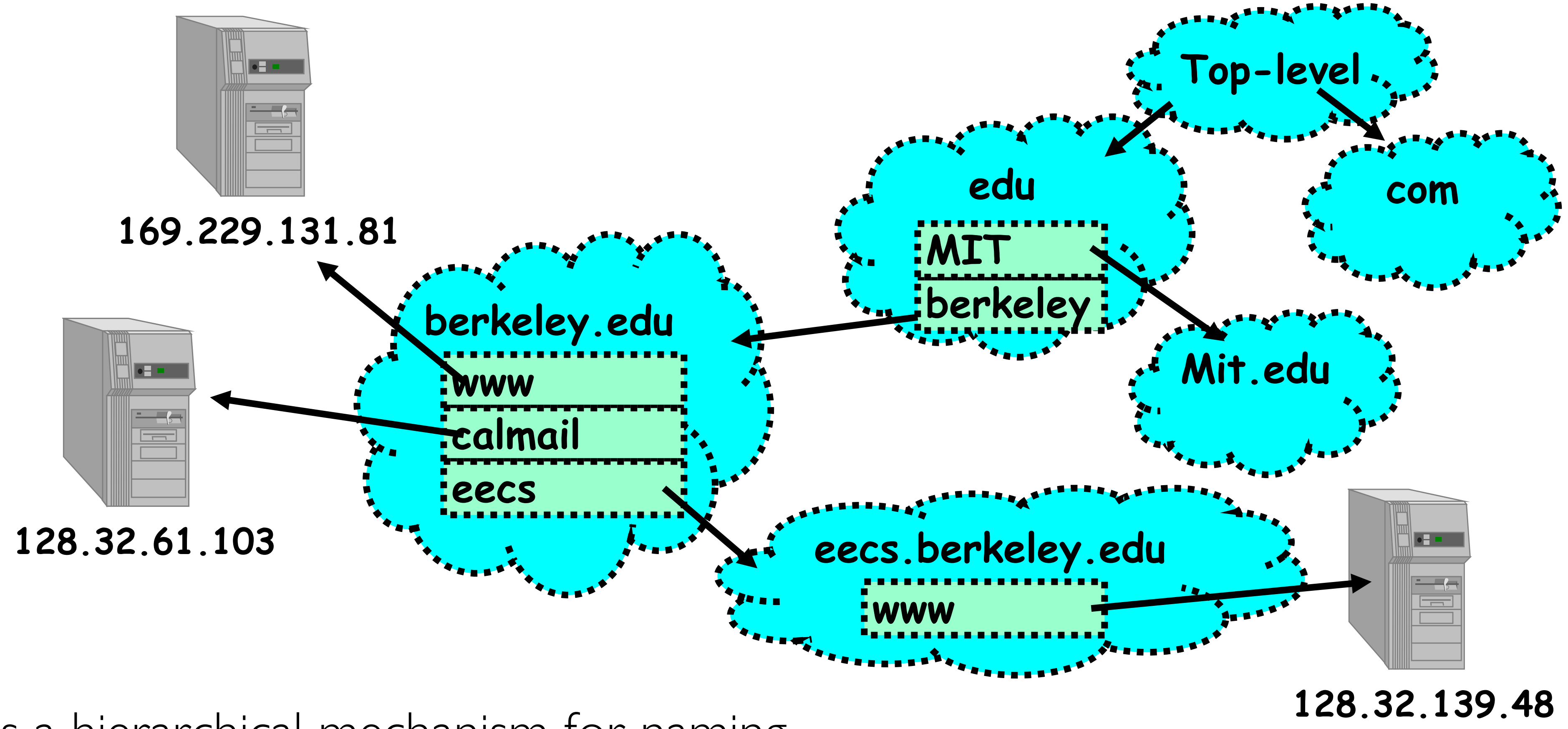
Naming in the Internet

Name ⇒ **Address**



- How to map human-readable names to IP addresses?
  - E.g. www.berkeley.edu ⇒ 128.32.139.48
  - E.g. www.google.com ⇒ different addresses depending on location, and load
- Why is this necessary?
  - IP addresses are hard to remember
  - IP addresses change:
    - » Say, Server 1 crashes gets replaced by Server 2
    - » Or – google.com handled by different servers
- Mechanism: Domain Naming System (DNS)

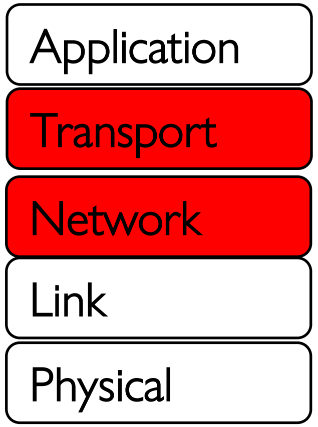
Domain Name System



- DNS is a hierarchical mechanism for naming
  - Name divided in domains, right to left: www.eecs.berkeley.edu

- Resolution: series of queries to successive servers

In this lecture



IP can reorder packets

IP can drop packets

$\Rightarrow$ How can we implement the *abstraction* of communication channels from host to host

- That can be ordered/reliable (if we want) - TCP
- That offer no guarantees (UDP)

Transport Layer

- Service:
  – Provide end-to-end communication between processes
  – Demultiplexing of communication between hosts
  – Possible other services:
    » Reliability in the presence of errors
    » Timing properties
    » Rate adaption (flow-control, congestion control)

- Interface: send message to “specific process” at given destination; local process receives messages sent to it
  – How are they named? Port Numbers

Internet Transport Protocols

- Datagram service (UDP): IP Protocol 17
  - No-frills extension of "best-effort" IP
  - Multiplexing/Demultiplexing among processes
- Reliable, in-order delivery (TCP): IP Protocol 6
  - Connection set-up & tear-down
  - Discarding corrupted packets
  - Retransmission of lost packets
  - Congestion control

Reliable Message Delivery: the Problem

- All physical networks can garble and/or drop packets
  - Physical media: packet not transmitted/received
  - Congestion: no place to put incoming packet
- Reliable Message Delivery on top of Unreliable Packets
  - Need some way to make sure that packets actually make it to receiver
    - » Every packet received at least once
    - » Every packet received at most once
  - Can combine with ordering: every packet received by process at destination exactly once and in order

Introducing TCP

- Reliable, in-order, and at most once delivery
- Stream oriented: messages can be of arbitrary length
- Provides multiplexing/demultiplexing to IP
- Provides congestion and flow control
- Application examples: file transfer, chat, http

TCP Service

1) Open connection: 3-way handshaking

2) Reliable byte stream transfer from (IPa, TCP_Port1) to (IPb, TCP_Port2)
   - Indication if connection fails: Reset

3) Close (tear-down) connection

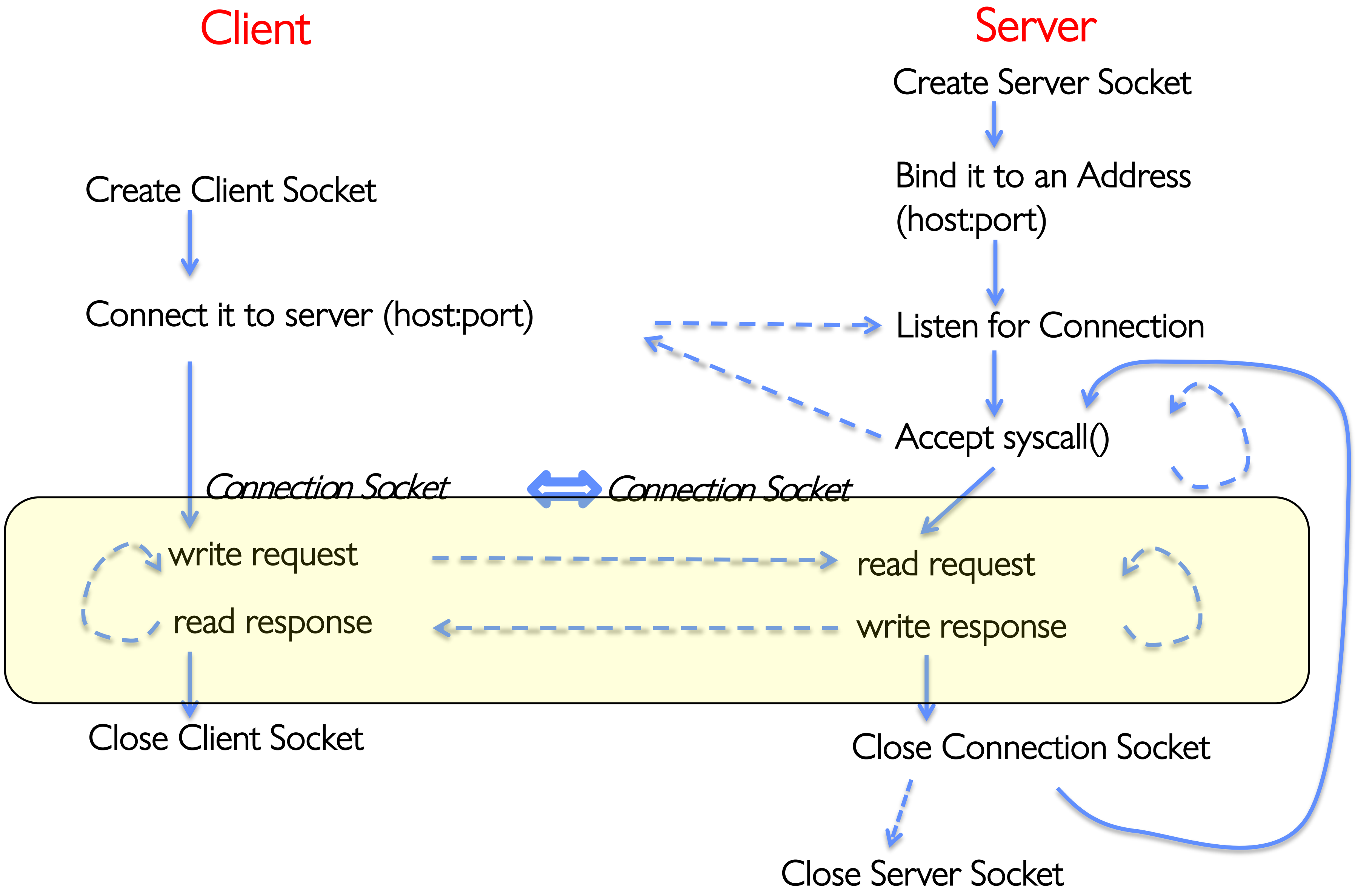
Recall: Socket creation and connection

- File systems provide a collection of permanent objects in structured name space
  - Processes open, read/write/close them
  - Files exist independent of the processes

- Sockets provide a means for processes to communicate (transfer data) to other processes.

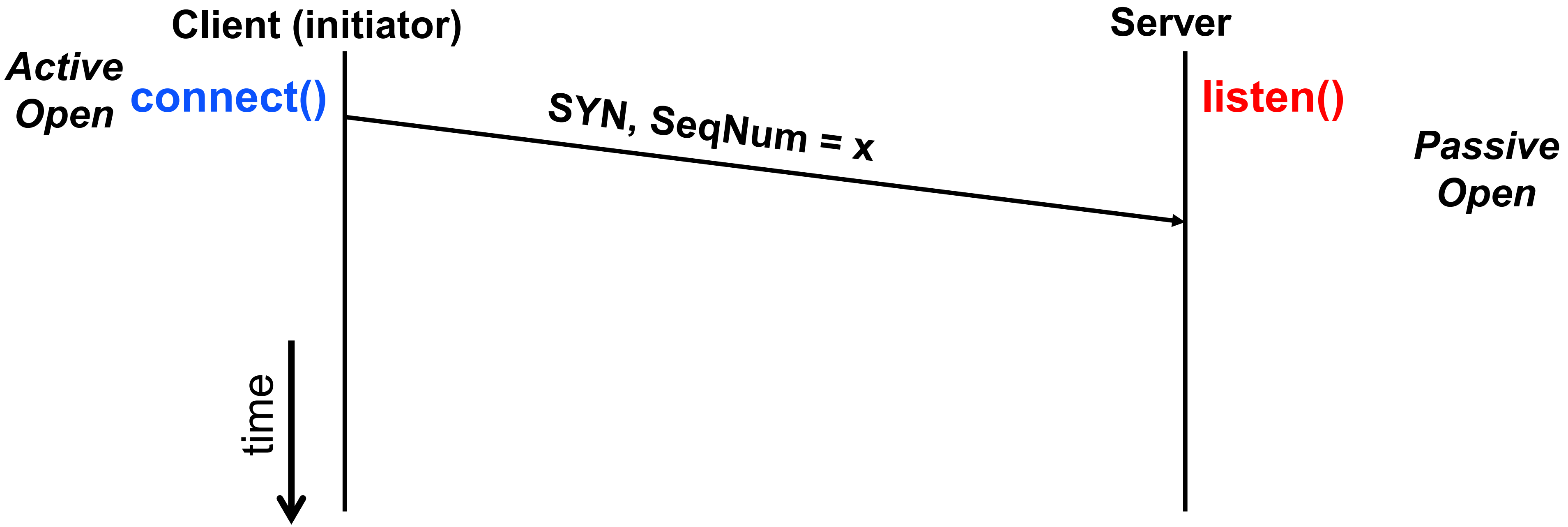
- Form 2-way pipes between processes
  - Possibly worlds away

Recall: Sockets in concept



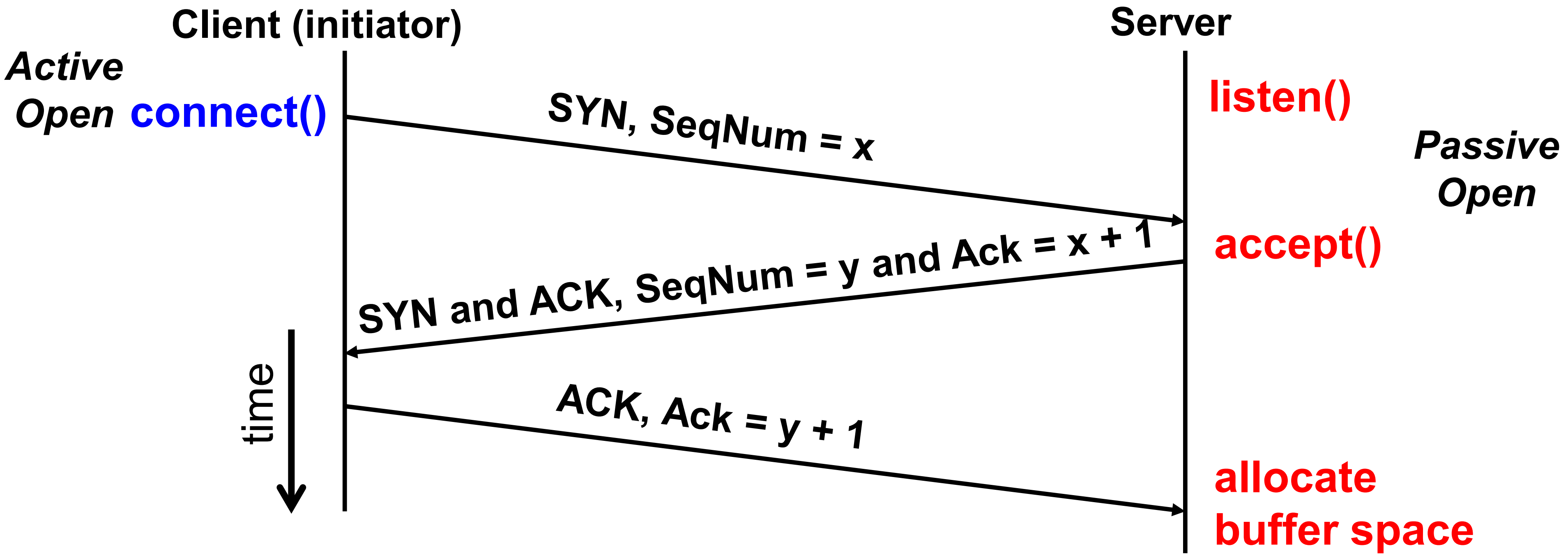
Open Connection: 3-Way Handshaking

- Server waits for new connection calling listen()
- Sender call connect() passing socket which contains server's IP address and port number
  - OS sends a special packet (SYN) containing a proposal for first sequence number, x



Open Connection: 3-Way Handshaking

- If it has enough resources, server calls accept() to accept connection, and sends back a SYN ACK packet containing
  - Client's sequence number incremented by one, (x + 1)
  - A sequence number proposal, y, for first byte server will send



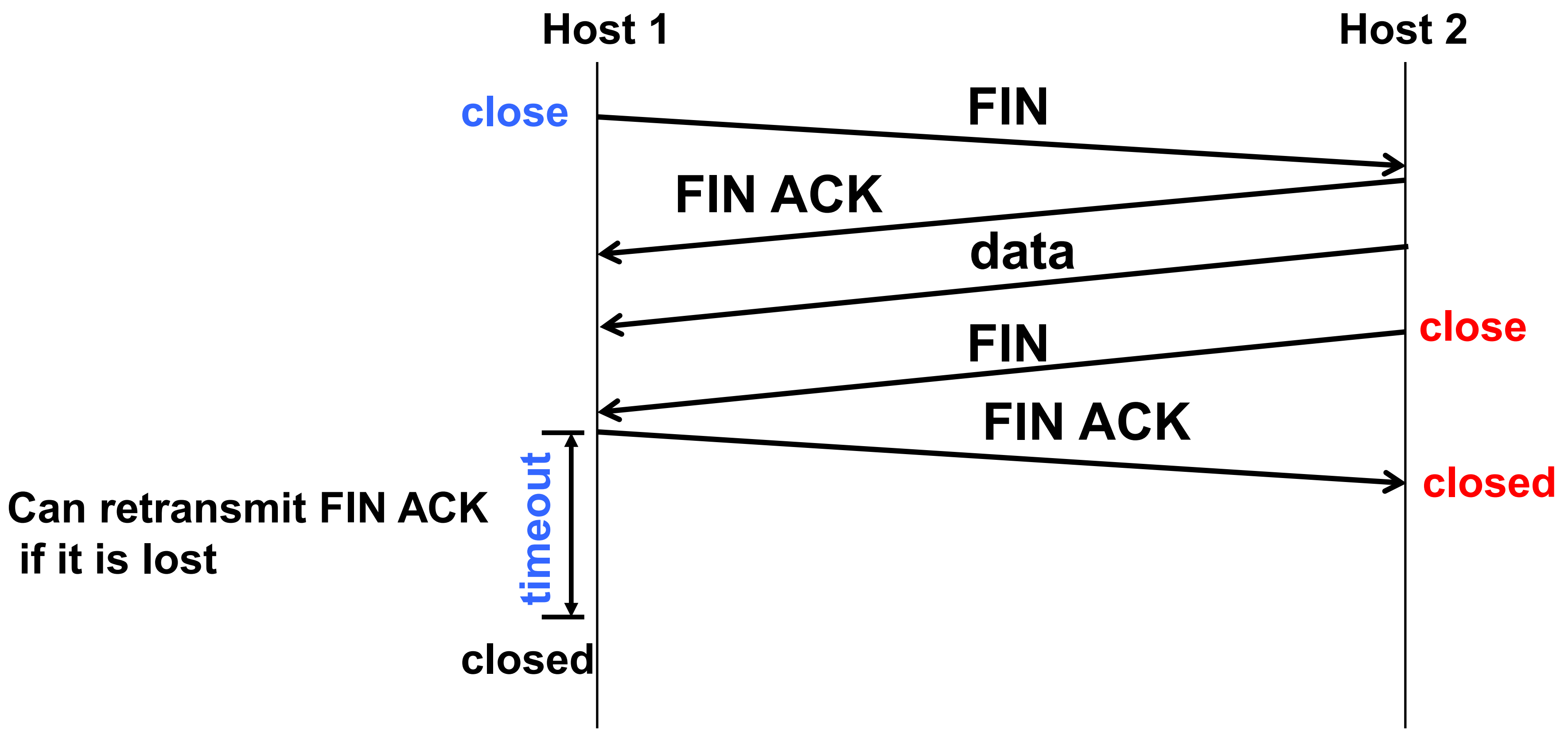
3-Way Handshaking (cont'd)

- Three-way handshake adds 1 RTT delay

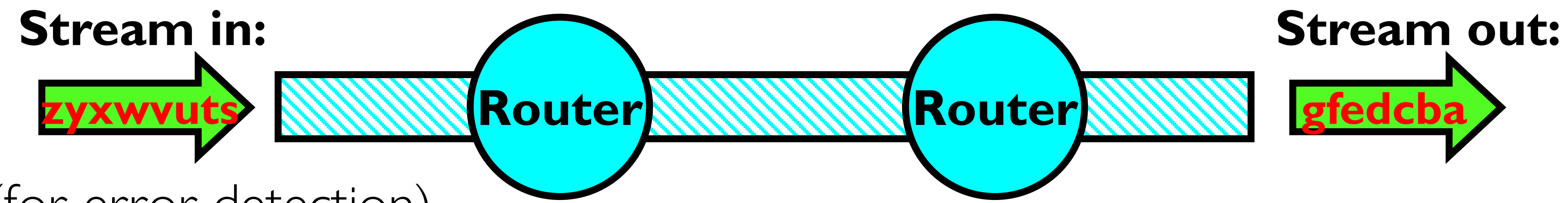
- Why?
  - Congestion control: SYN (40 byte) acts as cheap probe
  - Protects against delayed packets from other connection (would confuse receiver)

Close Connection

- Goal: both sides agree to close the connection
- 4-way connection tear down



Components of a solution for reliable transport



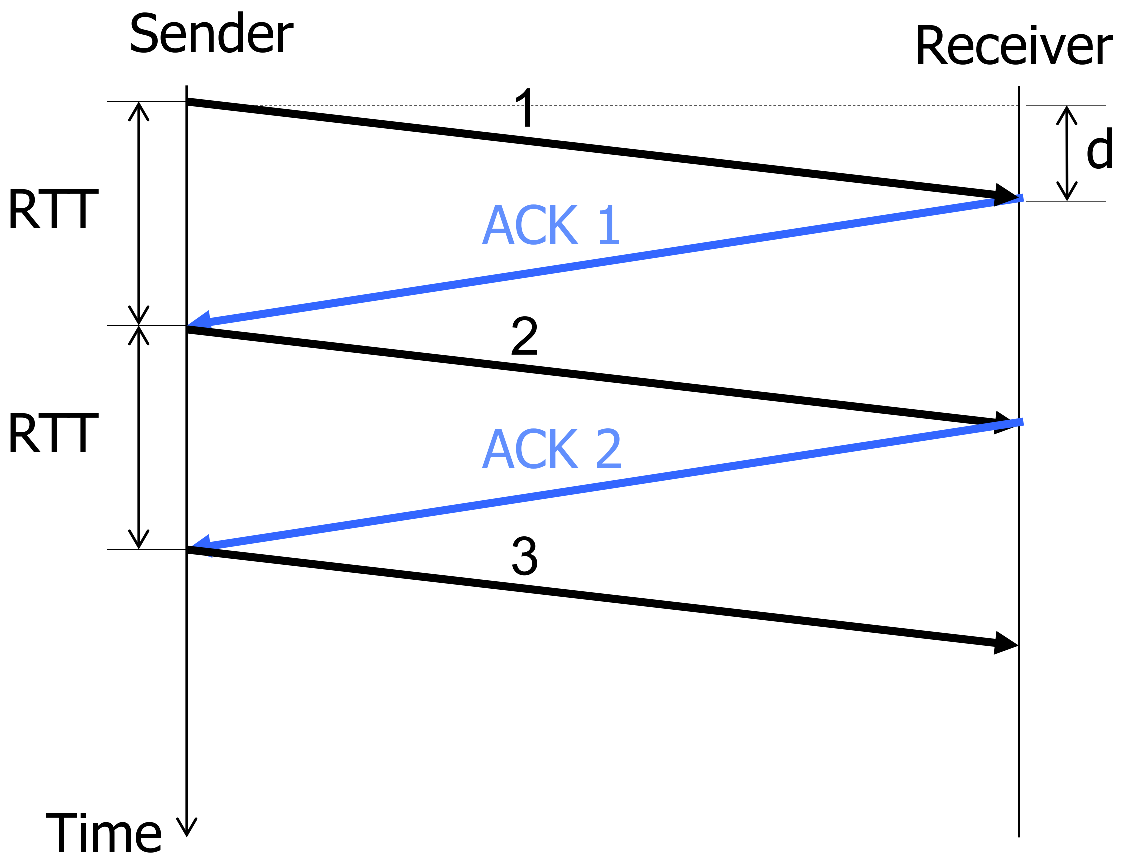
- Checksums (for error detection)
- Timers (for loss detection)
- Acknowledgments
  – Cumulative/selective
- Sequence numbers (duplicates, windows)
- Sliding Windows (for efficiency)
  – Go-Back-N (GBN) / Selective Replay (SR)

Detecting Packet Loss?

- Timeouts
  - Sender timeouts on not receiving ACK
- Missing ACKs
  - Receiver ACKs each packet
  - Sender detects a missing packet when seeing a gap in the sequence of ACKs
  - Need to be careful! Packets and ACKs might be reordered
- NACK: Negative ACK
  - Receiver sends a NACK specifying a packet it is missing

Stop & Wait w/o Errors

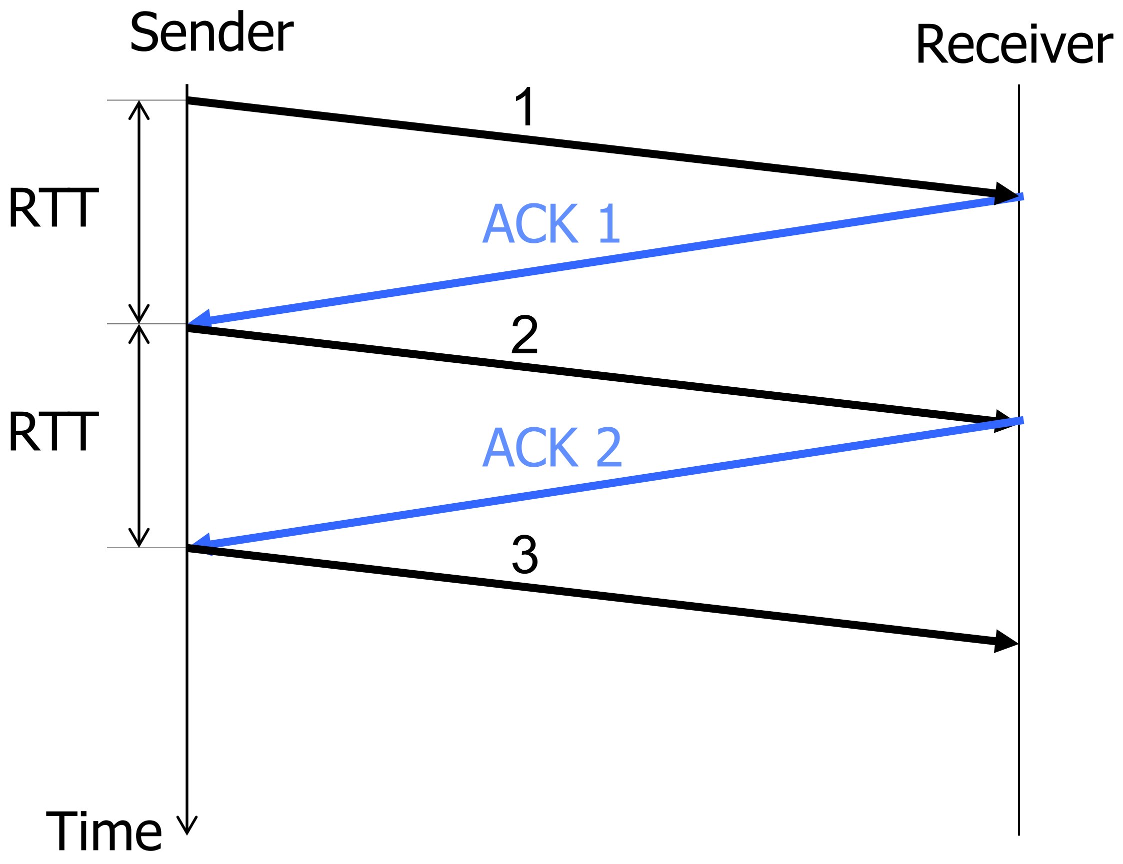
- Send; wait for ack; repeat
- RTT: Round Trip Time (RTT): time it takes a packet to travel from sender to receiver and back
  - One-way latency (d): one way delay from sender and receiver



RTT = 2*d (if latency is symmetric)

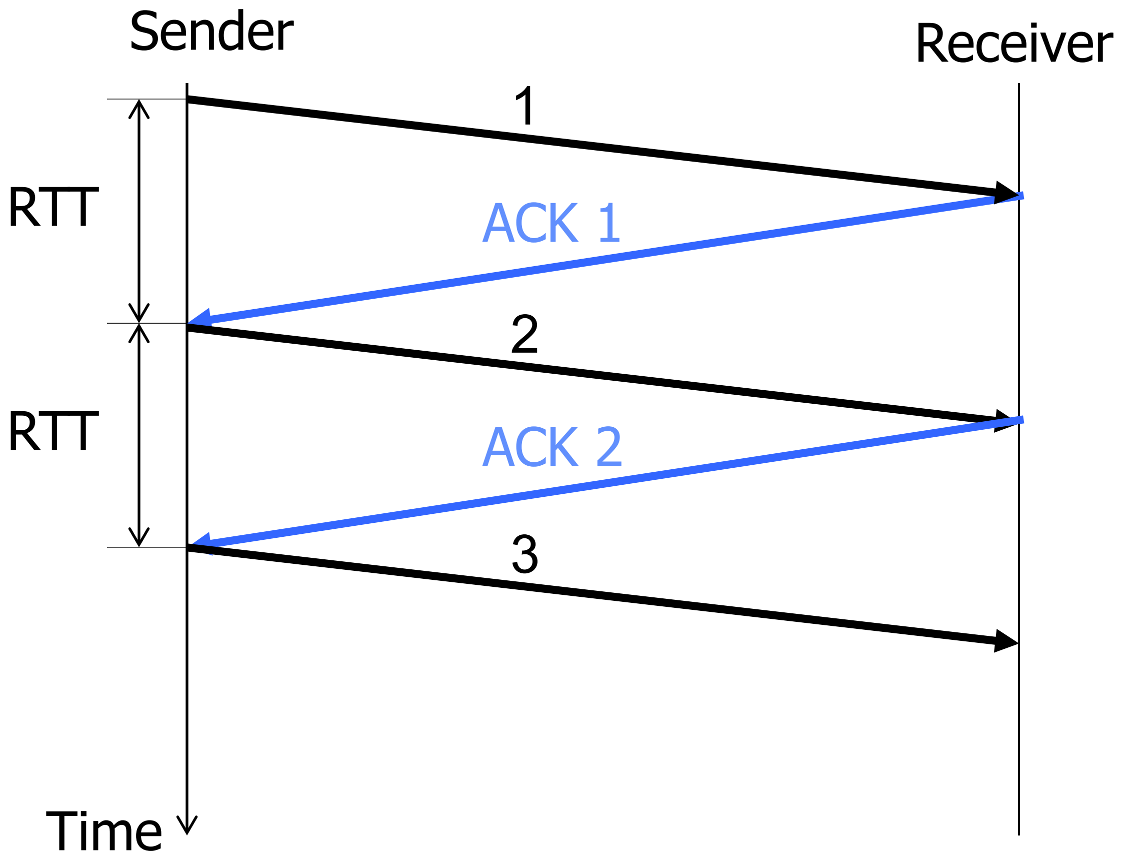
Stop & Wait w/o Errors

- How many packets can you send?
- 1 packet / RTT
- Throughput: number of bits delivered to receiver per sec



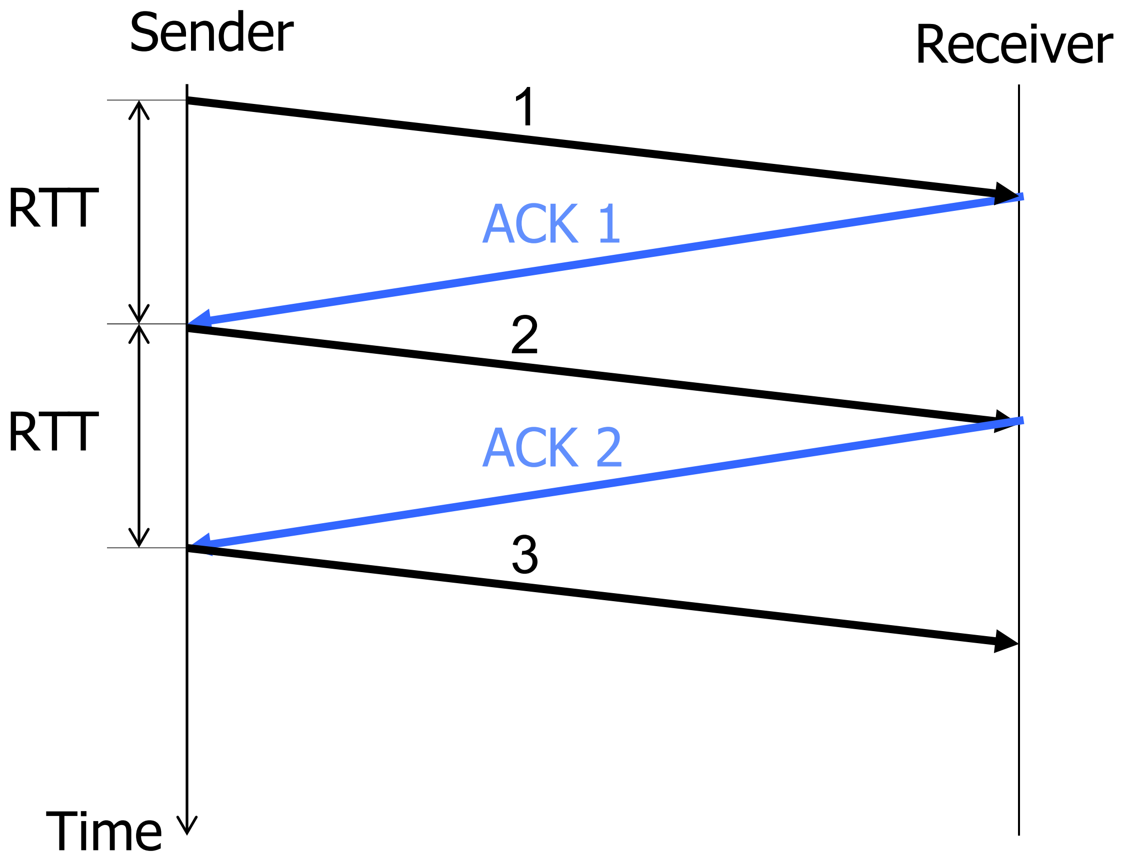
Stop & Wait w/o Errors

- Say, RTT = 100ms
- 1 packet = 1500 bytes
- Throughput = 1500*8bits/0.1s = 120 Kbps



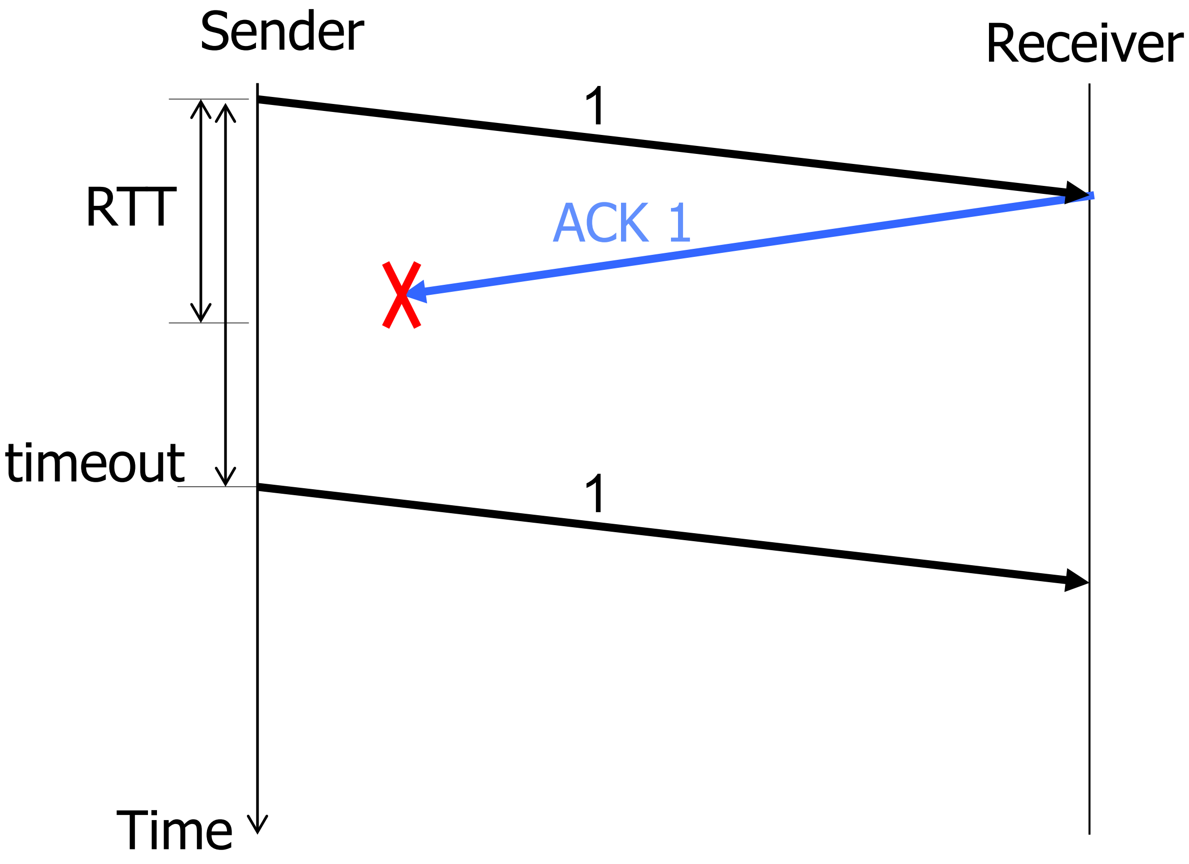
Stop & Wait w/o Errors

- Can be very inefficient for high capacity links
- Throughput doesn't depend on the network capacity → even if capacity is 1Gbps, we can only send 120 Kbps!



Stop & Wait with Errors

- If a loss wait for a retransmission timeout and retransmit
- How do you pick the timeout?

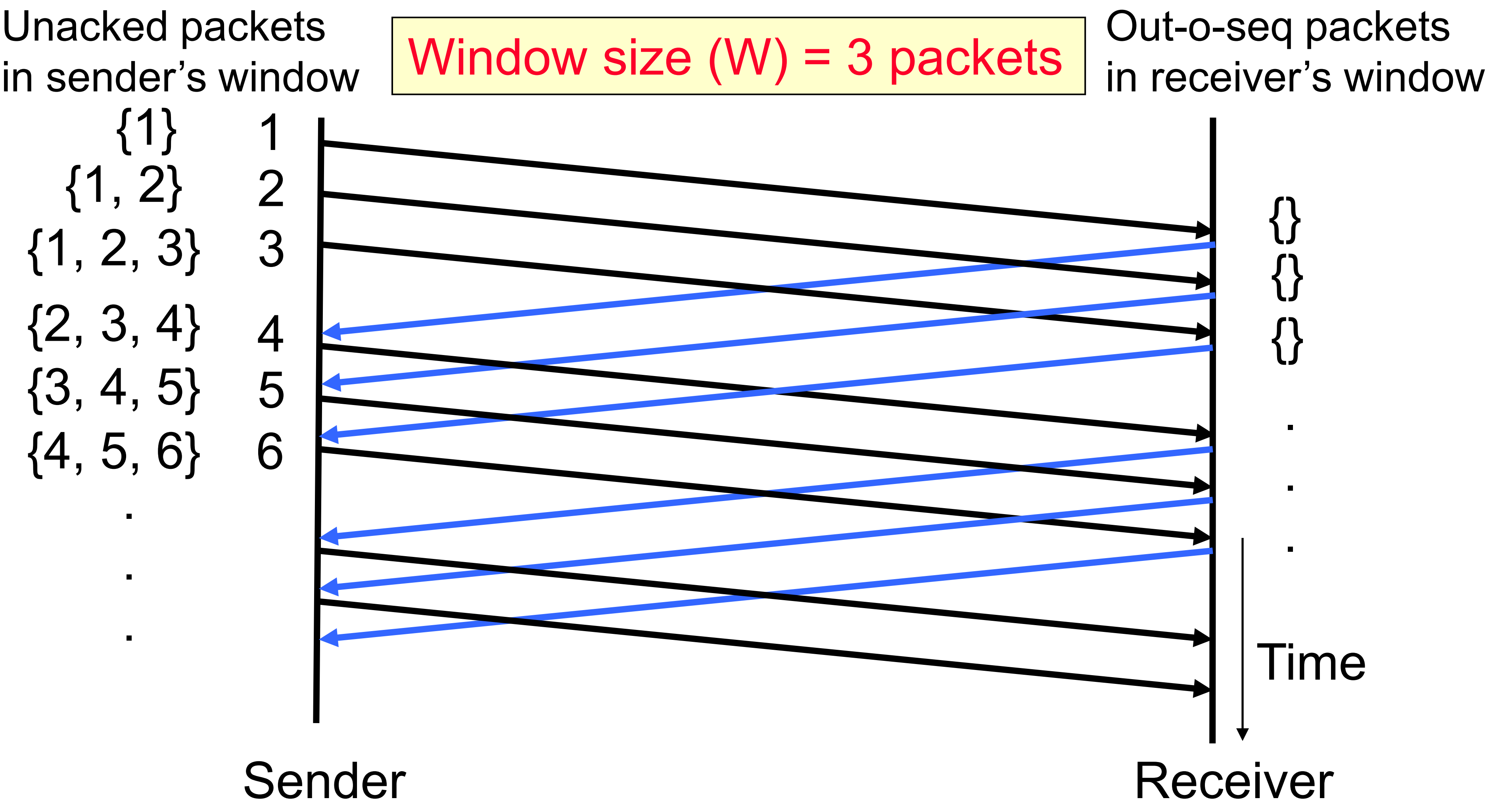


Sliding Window

- *window* = set of adjacent sequence numbers
- The size of the set is the *window size*
- Assume window size is n
- Let A be the last ACK'd packet of sender without gap; then window of sender = {A+1, A+2, …, A+n}
- Sender can send packets in its window
- Let B be the last received packet without gap by receiver, then window of receiver = {B+1,…, B+n}
- Receiver can accept out of sequence, if in window

Sliding Window w/o Errors

- Throughput = W*packet_size/RTT



Example: Sliding Window w/o Errors

- Assume
  - Link capacity, C = 1Gbps
  - Latency between end-hosts, RTT = 80ms
  - packet_length = 1000 bytes
- What is the window size W to match link's capacity, C?

- Solution

  We want Throughput = C

  Throughput = W*packet_size/RTT

  C = W*packet_size/RTT

  W = C*RTT/packet_size = 10^9bps*80*10^{-3}s/(8000b) = 10^4 packets

Window size ~ Bandwidth (Capacity), delay (RTT/2)

Sliding Window with Errors

- Two approaches
  - Go-Back-n (GBN)
  - Selective Repeat (SR)

- In the absence of errors they behave identically

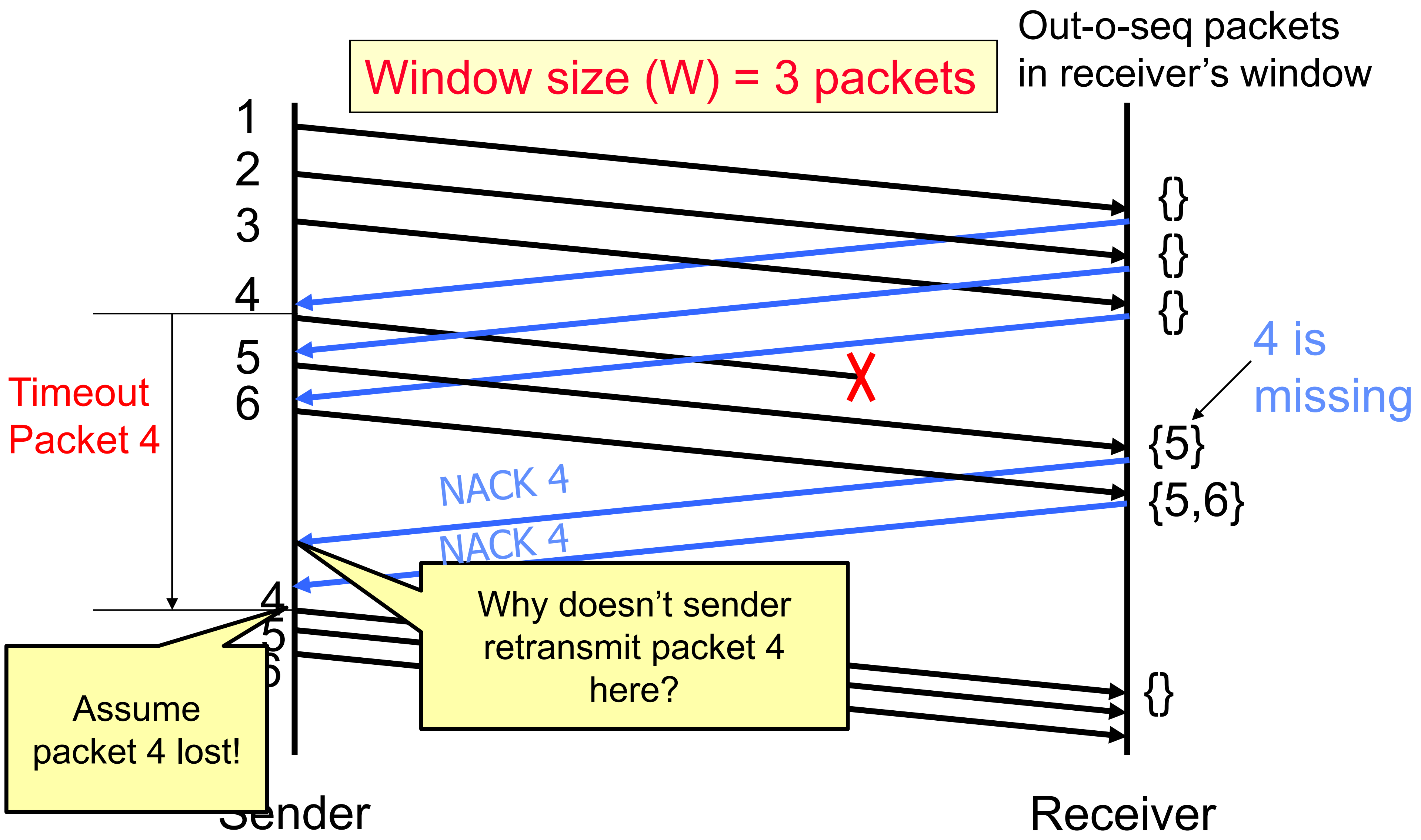
Go-Back-n (GBN)

- Sender transmits up to *n* unacknowledged packets

- Receiver only accepts packets in order
  - discards out-of-order packets (i.e., packets other than *B+1*)
- Receiver uses cumulative acknowledgements
  - i.e., sequence# in ACK = next expected in-order sequence#

- Sender sets timer for 1^{st} outstanding ack (A+1)
- If timeout, retransmit *A+1*, … , *A+n*

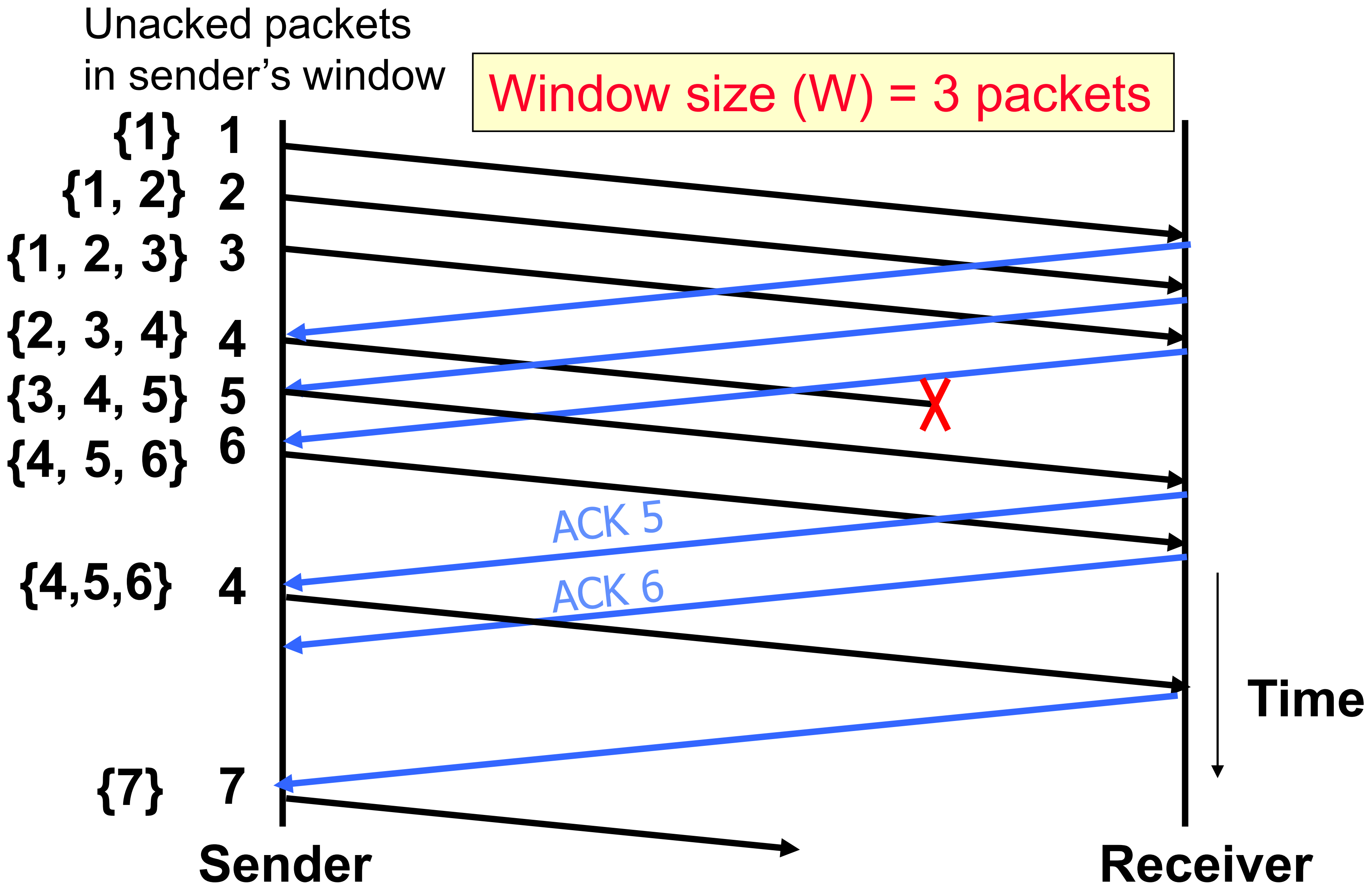
GBN Example with Errors



Selective Repeat (SR)

- Sender: transmit up to *n* unacknowledged packets
- Assume packet *k* is lost, *k+1* is not
- Receiver: indicates packet *k+1* correctly received
- Sender: retransmit only packet *k* on timeout
- Efficient in retransmissions but complex book-keeping
  - need a timer per packet

SR Example with Errors



Recap: components of a solution

- Checksums (for error detection)
- Timers (for loss detection)
- Acknowledgments
  - cumulative
  - selective
- Sequence numbers (duplicates, windows)
- Sliding Windows (for efficiency)

- Reliability protocols use the above to decide when and what to retransmit or acknowledge

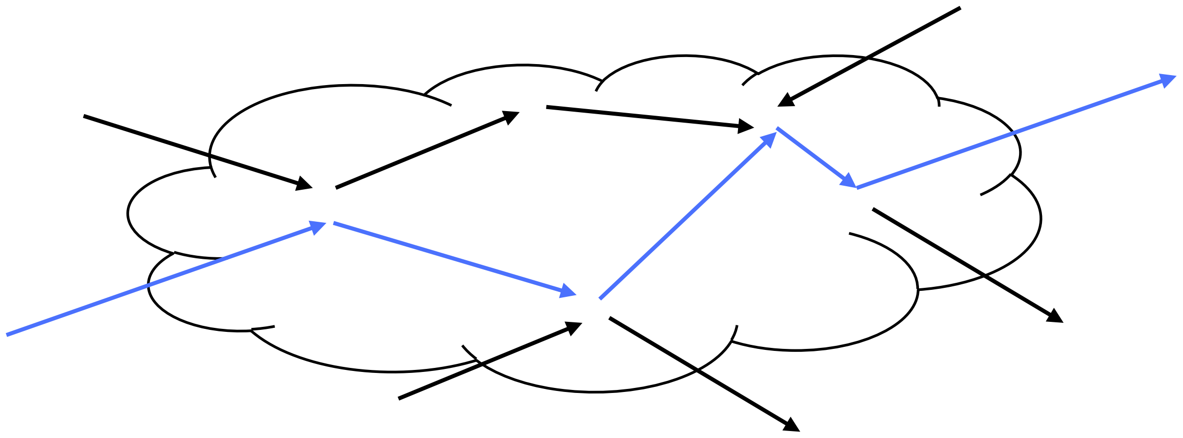
What does TCP do?

Most of our previous tricks + a few differences

- Sequence numbers are byte offsets
- Sender and receiver maintain a sliding window
- Receiver sends cumulative acknowledgements (like GBN)
- Sender maintains a single retry. timer
- Receivers do not drop out-of-sequence packets (like SR)
- Introduces fast retransmit : optimization that uses duplicate ACKs to trigger early retries
- Introduces timeout estimation algorithms

Congestion

- Too much data trying to flow through some part of the network



- IP's solution: **Drop** packets
- What happens to TCP connection?
  - Lots of retransmission – wasted work
  - Lots of waiting for timeouts – underutilized connection

Two Basic Questions

- How does the sender detect congestion?

- How does the sender adjust its sending rate?
  - To address three issues
    - » Finding available bottleneck bandwidth
    - » Adjusting to bandwidth variations
    - » Sharing bandwidth

Detecting Congestion

- Packet delays
  - Tricky: noisy signal (delay often varies considerably)

- Router tell end-hosts they're congested

- Packet loss
  - Fail-safe signal that TCP already has to detect
  - Complication: non-congestive loss (checksum errors)

- Two indicators of packet loss
  - No ACK after certain time interval: timeout
  - Multiple duplicate ACKs

Not All Losses the Same

- Duplicate ACKs: isolated loss
  – Still getting ACKs
- Timeout: much more serious
  – Not enough packets in progress to trigger duplicate-acks, OR
  – Suffered several losses
- We will adjust rate differently for each case

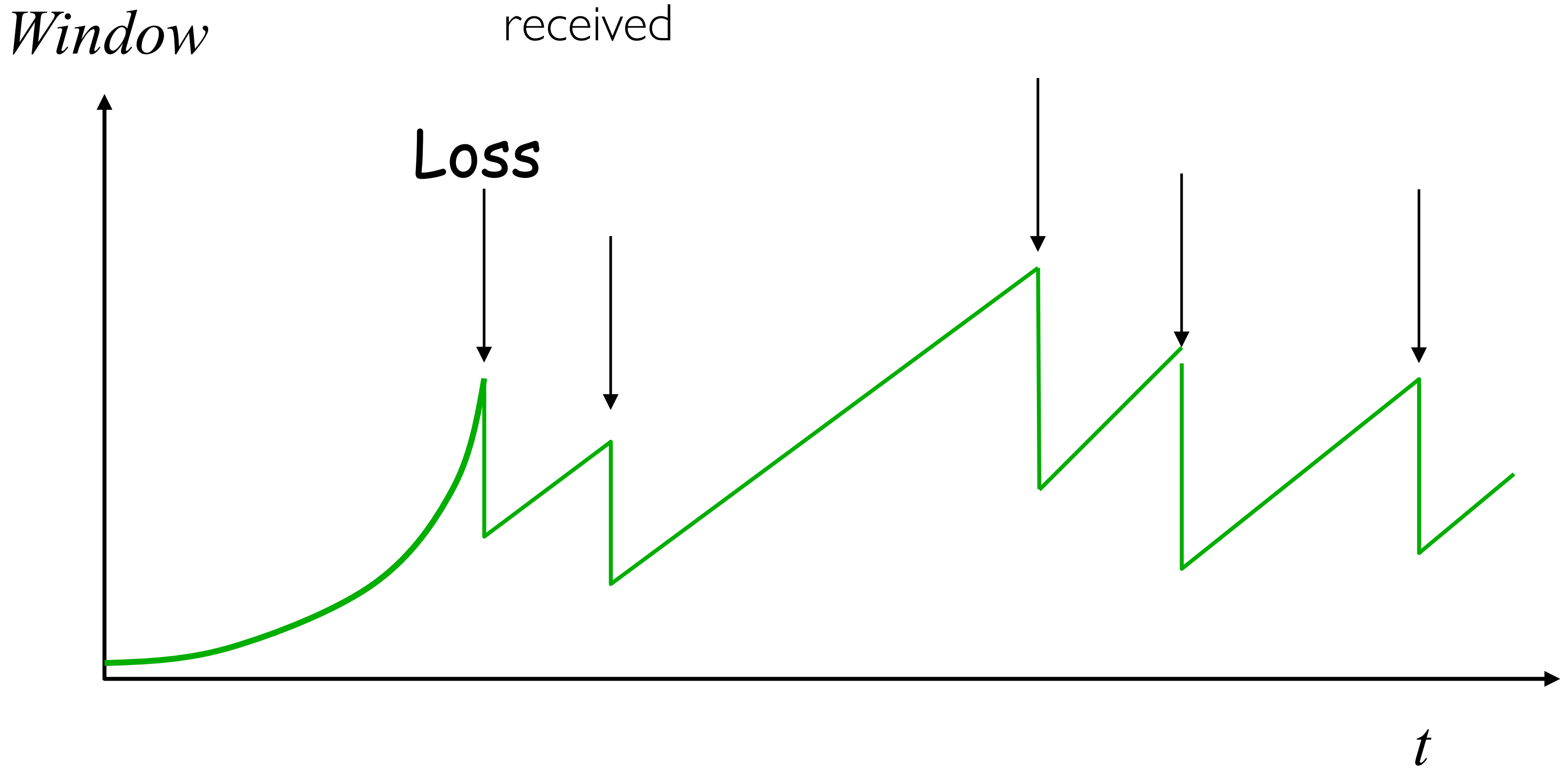
Rate Adjustment

- Basic structure:
  - Upon receipt of ACK (of new data): increase rate
  - Upon detection of loss: decrease rate

- How we increase/decrease the rate depends on the phase of congestion control we're in:
  - Discovering available bottleneck bandwidth *vs.*
  - Adjusting to bandwidth variations

Congestion Avoidance

- Solution: Adjust **Window Size**
- AIMD: Additive Increase, Multiplicative Decrease
  - When packet dropped (missed ack), cut window size in half
  - If no timeouts, increase window size by C for each acknowledgement received



Summary

- The Internet consists of 5 layers: Application, Transport, Network, Link, Physical
- IP layer: Hourglass of the Internet. Uses routers to route packets from one end to the internet through the other. DNS helps resolves human-readable addresses to IP
- TCP in the transport layer: uses sliding windows and acks to implement reliable delivery. Uses congestion control to rate-limit protocol