Overview

Introduction to Data Networks

2008. 3

Prof. Seung-Woo Seo

Lecture slides from J. Lexford

Goals for Today's Class

- Course overview
  - Goals of the course
  - Structure of the course
  - Learning the material
  - Programming assignments
  - Course grading
  - Academic policies
- Key concepts in data networking
  - Protocols
  - Layering
  - Resource allocation
  - Naming

What You Learn in This Course

- Skill: network programming
  - Socket programming
- Knowledge: how the Internet works
  - IP protocol suite
  - Internet architecture
  - Applications (Web, e-mail, P2P, VoIP, …)
- Insight: key concepts in networking
  - Protocols
  - Layering
  - Resource allocation
  - Naming

Structure of the Course (1st Half)

- Start at the top
  - – Sockets: how applications view the Internet
  - – Protocols: essential elements of a protocol
- Then study the "narrow waist" of IP
  - – IP best-effort packet-delivery service
  - – IP addressing and packet forwarding
- And how to build on top of the narrow waist
  - – Transport protocols (TCP, UDP)
  - – Domain Name System (DNS)
  - – Glue (ARP, DHCP, ICMP)
  - – End-system security and privacy (NAT, firewalls)
- Looking underneath IP
  - – Link technologies (Ethernet, wireless, …)

Structure of the Course (2^{nd} Half)

- And how to get the traffic from here to there
  - Internet routing architecture (the “inter” in Internet)
  - Intradomain and interdomain routing protocols
- Building applications
  - Web and content-distribution networks
  - E-mail
  - Peer-to-peer file sharing
  - Multimedia streaming and voice-over-IP
- Other approaching to building networks
  - Circuit switching (e.g., ATM, MPLS, …)
  - More on wireless networks, multicast, …

Policies: Write Your Own Code

Programming in an individual creative process much like composition. You must reach your own understanding of the problem and discover a path to its solution. During this time, discussions with friends are encouraged. However, when the time comes to write code that solves the problem, such discussions are no longer appropriate - the program must be your own work.

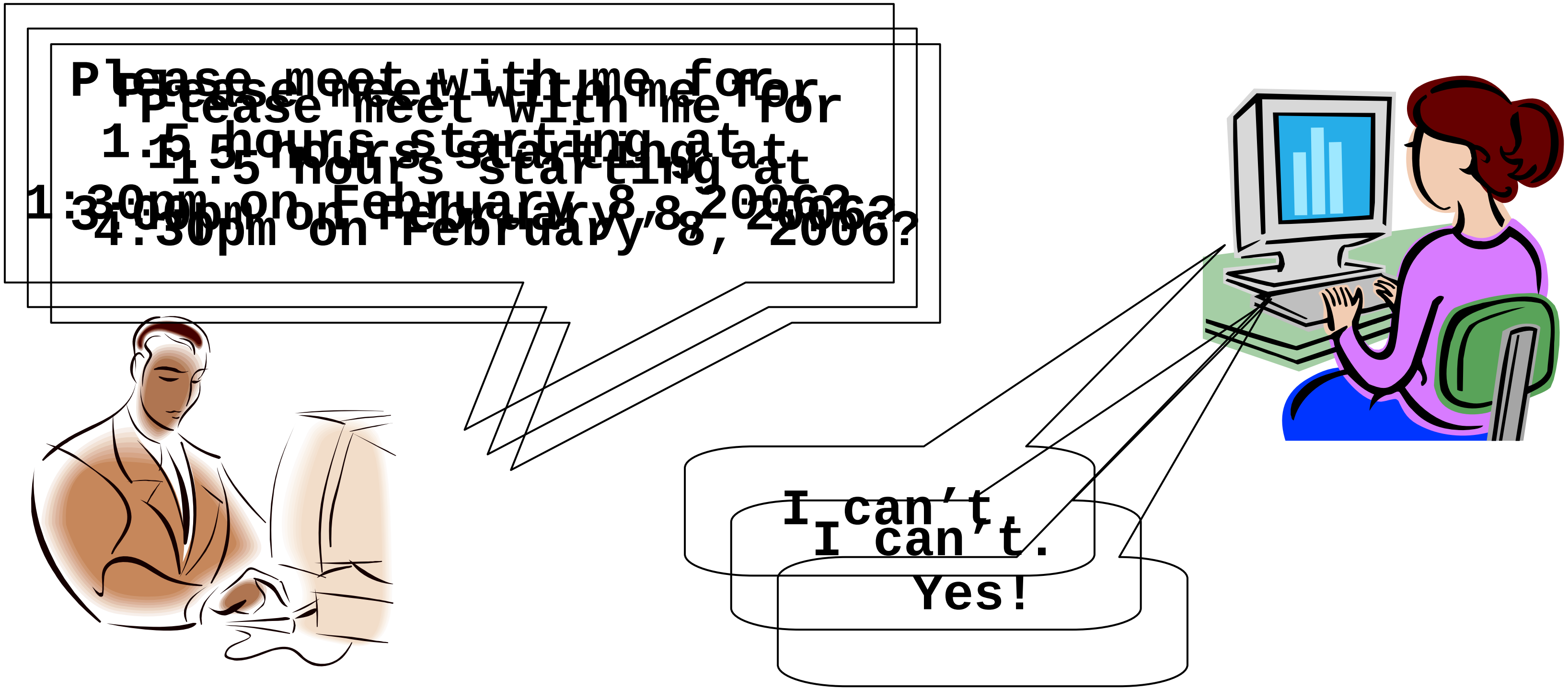
If you have a question about how to use some feature of C, UNIX, etc., you can certainly ask your friends or the TA, but **do not, under any circumstances, copy another person's program.** Letting someone copy your program or using someone else's code in any form is a **violation of academic regulations**. "Using someone else's code" includes using solutions or partial solutions to assignments provided by commercial web sites, instructors, preceptors, teaching assistants, friends, or students from any previous offering of this course or any other course.

Key Concepts in Networking

- Protocols
  - Speaking the same language
  - Syntax and semantics
- Layering
  - Standing on the shoulders of giants
  - A key to managing complexity
- Resource allocation
  - Dividing scare resources among competing parties
  - Memory, link bandwidth, wireless spectrum, paths, …
  - Distributed vs. centralized algorithms
- Naming
  - What to call computers, services, protocols, …

Protocols: Calendar Service

- Making an appointment with your advisor



Please meet with me for 1.5 hours starting at 1:30pm on February 8, 2006?

Please meet with me for 1.5 hours starting at 3:00pm on February 8, 2006?

Please meet with me for 1.5 hours starting at 4:30pm on February 8, 2006?

I can't.

I can't.

Yes!

- Specifying the messages that go back and forth
  - And an understanding of what each party is doing

Okay, So This is Getting Tedious

- You: When are you free to meet for 1.5 hours during the next two weeks?
- Advisor: 10:30am on Feb 8 and 1:15pm on Feb 9.
- You: Book me for 1.5 hours at 10:30am on Feb 8.
- Advisor: Yes.

Well, Not Quite Enough

- Student #1: When can you meet for 1.5 hours during the next two weeks?
- Advisor: 10:30am on Feb 8 and 1:15pm on Feb 9.
- Student #2: When can you meet for 1.5 hours during the next two weeks?
- Advisor: 10:30am on Feb 8 and 1:15pm on Feb 9.
- Student #1: Book me for 1.5 hours at 10:30am on Feb 8.
- Advisor: Yes.
- Student #2: Book me for 1.5 hours at 10:30am on Feb 8.
- Advisor: Uh… well… I can no longer meet then. I'm free at 1:15pm on Feb 9.
- Student #2: Book me for 1.5 hours at 1:15pm on Feb 9.
- Advisor: Yes.

Specifying the Details

- How to identify yourself?
  - Name? Social security number?
- How to represent dates and time?
  - Time, day, month, year? In what time zone?
  - Number of seconds since Jan 1, 1970?
- What granularities of times to use?
  - Any possible start time and meeting duration?
  - Multiples of five minutes?
- How to represent the messages?
  - Strings? Record with name, start time, and duration?
- What do you do if you don’t get a response?
  - Ask again? Reply again?

Example: HyperText Transfer Protocol

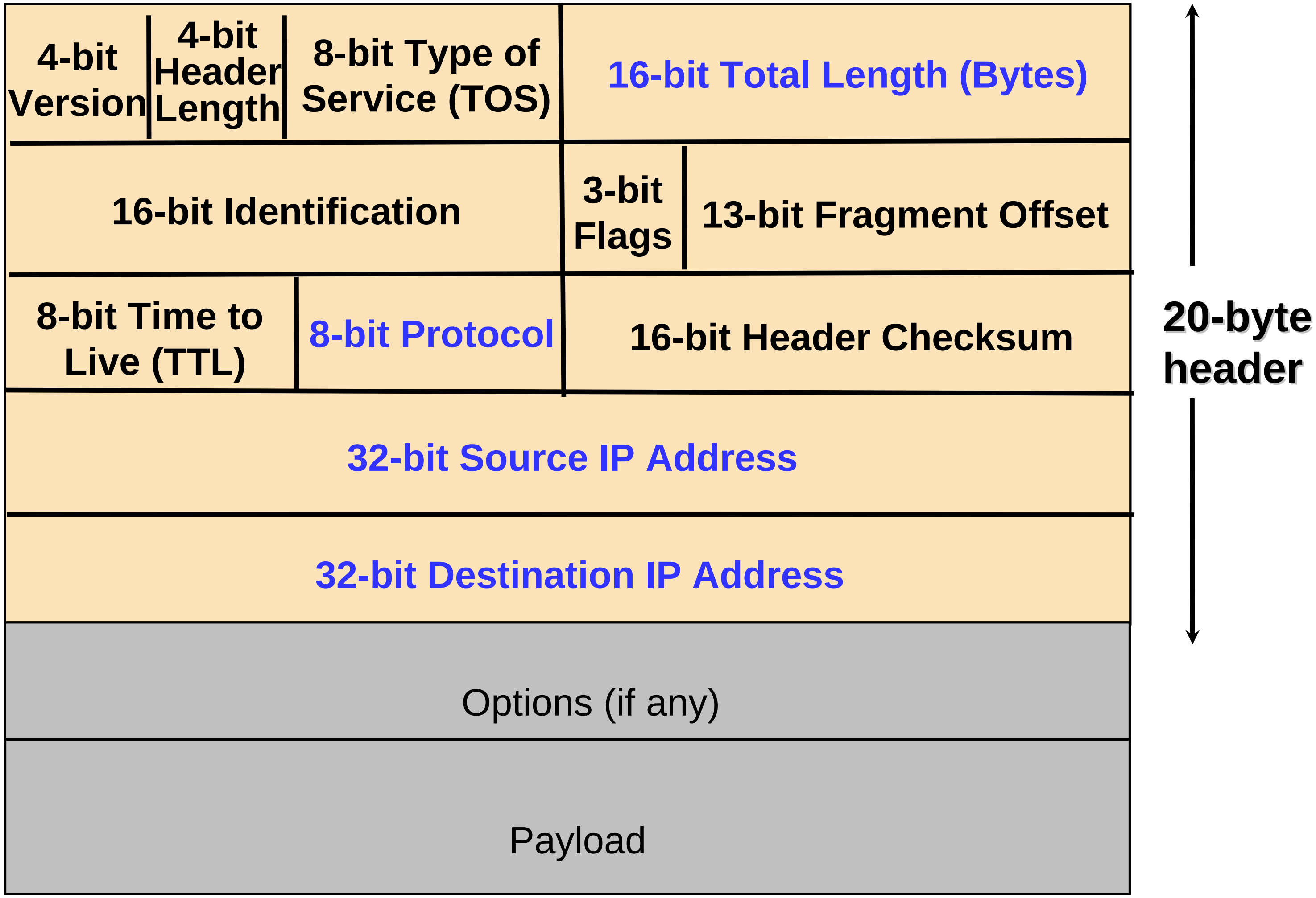
```
GET /courses/archive/spring08/c461/ HTTP/1.1
Host: www.ee.snu.ac.kr
User-Agent: Mozilla/4.03
CRLF
```

Request

Response

```
HTTP/1.1 200 OK
Date: Mon, 6 Feb 2008 13:09:03 GMT
Server: Netscape-Enterprise/3.5.1
Last-Modified: Mon, 6 Feb  2008 11:12:23 GMT
Content-Length: 21
CRLF
Site under construction
```

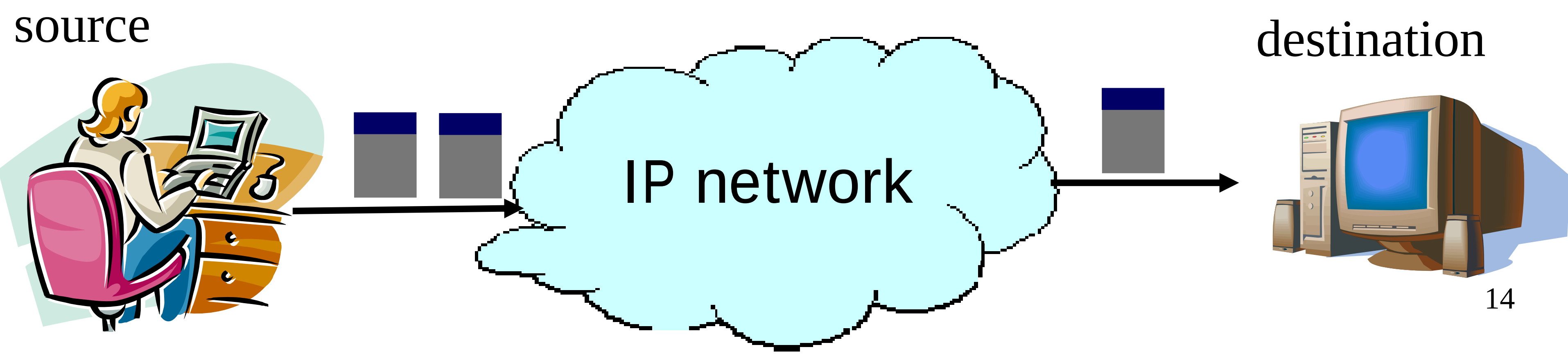
Example: IP Packet



<table>
<tr><td>4-bit Version</td><td>4-bit Header Length</td><td>8-bit Type of Service (TOS)</td><td colspan="2">16-bit Total Length (Bytes)</td></tr>
<tr><td colspan="3">16-bit Identification</td><td>3-bit Flags</td><td>13-bit Fragment Offset</td></tr>
<tr><td colspan="2">8-bit Time to Live (TTL)</td><td>8-bit Protocol</td><td colspan="2">16-bit Header Checksum</td></tr>
<tr><td colspan="5">32-bit Source IP Address</td></tr>
<tr><td colspan="5">32-bit Destination IP Address</td></tr>
<tr><td colspan="5">Options (if any)</td></tr>
<tr><td colspan="5">Payload</td></tr>
</table>

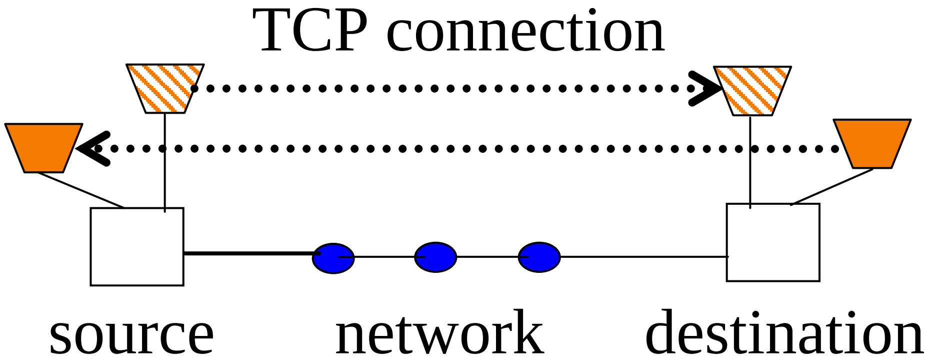
IP: Best-Effort Packet Delivery

- Packet switching
  - Send data in packets
  - Header with source & destination address
- Best-effort delivery
  - Packets may be lost
  - Packets may be corrupted
  - Packets may be delivered out of order



Example: Transmission Control Protocol

- Communication service (socket)
  - Ordered, reliable byte stream
  - Simultaneous transmission in both directions
- Key mechanisms at end hosts
  - Retransmit lost and corrupted packets
  - Discard duplicate packets and put packets in order
  - Flow control to avoid overloading the receiver buffer
  - Congestion control to adapt sending rate to network load

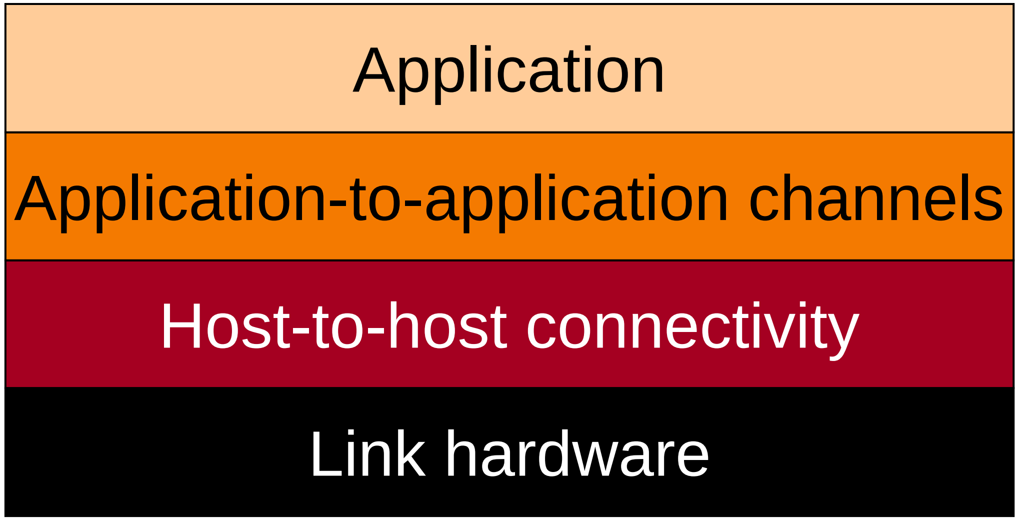


Protocol Standardization

- Communicating hosts speaking the same protocol
  - Standardization to enable multiple implementations
  - Or, the same folks have to write all the software
- Standardization: Internet Engineering Task Force
  - Based on working groups that focus on specific issues
  - Produces “Request For Comments” (RFCs)
    - Promoted to standards via rough consensus and running code
    - E.g., RFC 959 on “File Transfer Protocol”
  - IETF Web site is http://www.ietf.org
- De facto standards: same folks writing the code
  - P2P file sharing, Skype, <your protocol here>…

Layering: A Modular Approach

- Sub-divide the problem
  - – Each layer relies on services from layer below
  - – Each layer exports services to layer above
- Interface between layers defines interaction
  - – Hides implementation details
  - – Layers can change without disturbing other layers



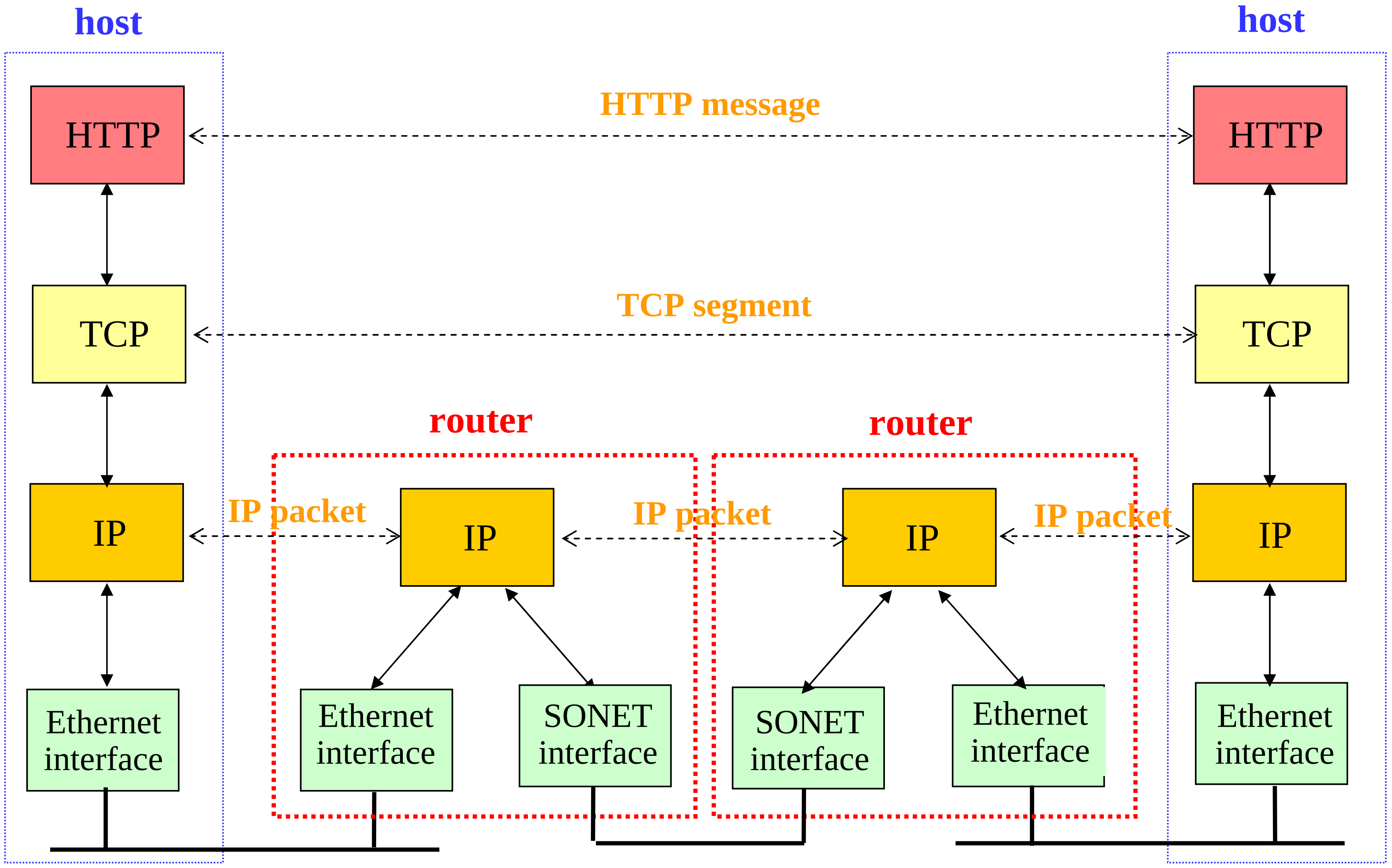
Application

Application-to-application channels

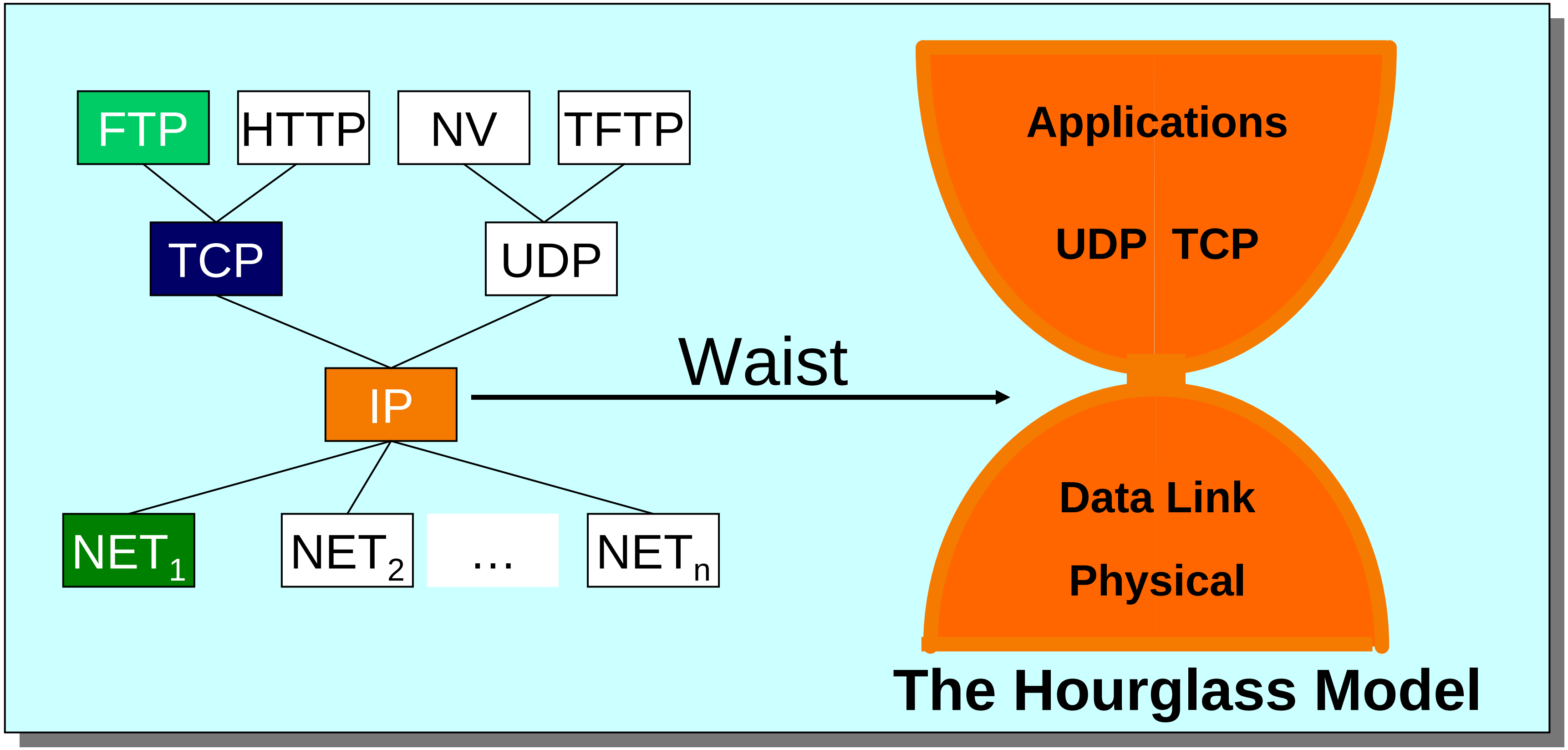
Host-to-host connectivity

Link hardware

IP Suite: End Hosts vs. Routers

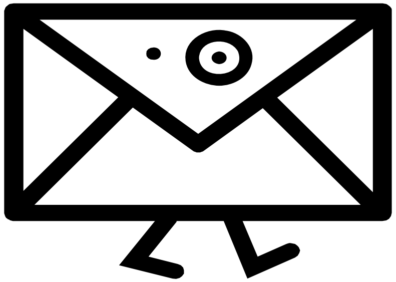


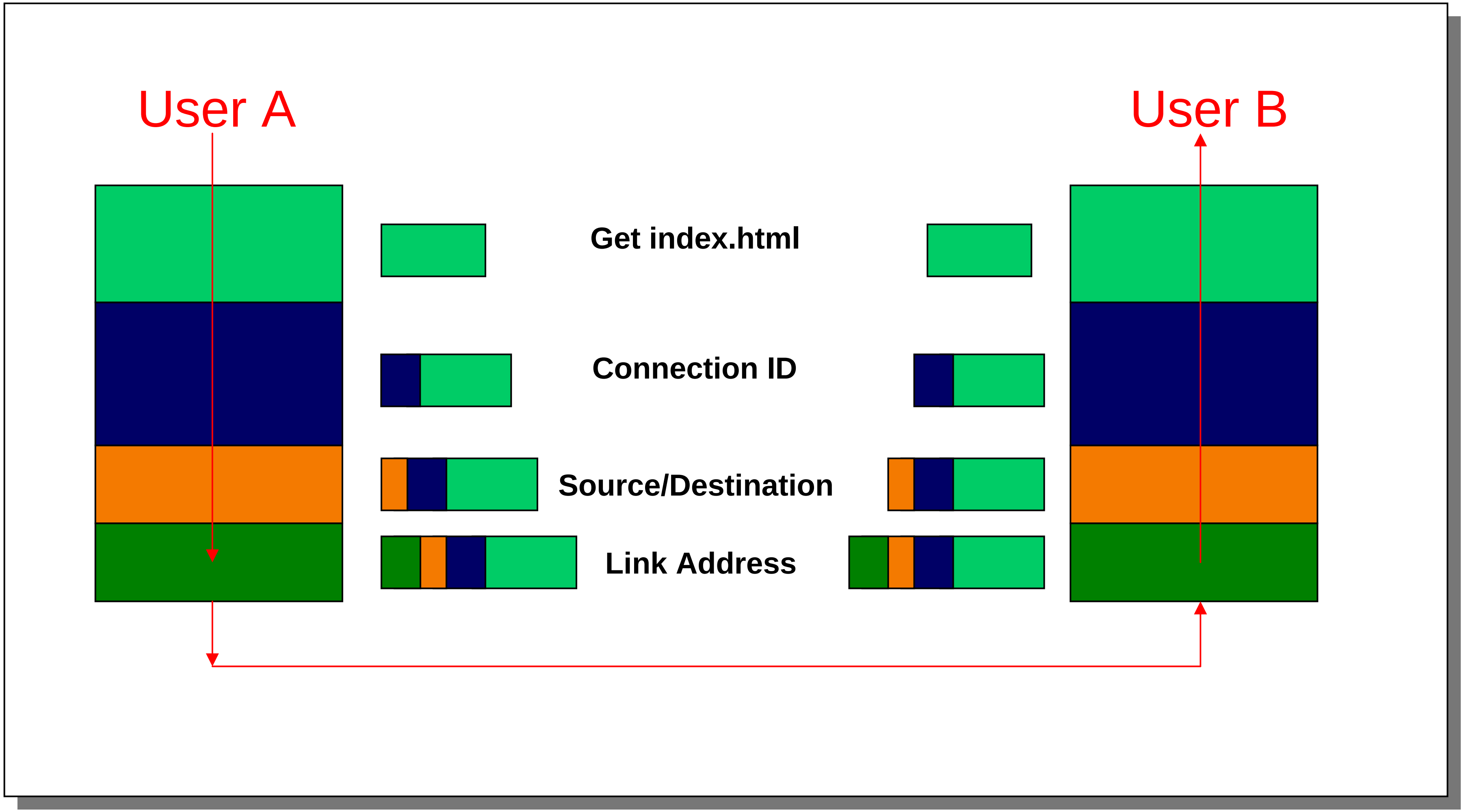
The Internet Protocol Suite



The waist facilitates interoperability

Layer Encapsulation





User A

User B

Get index.html

Connection ID

Source/Destination

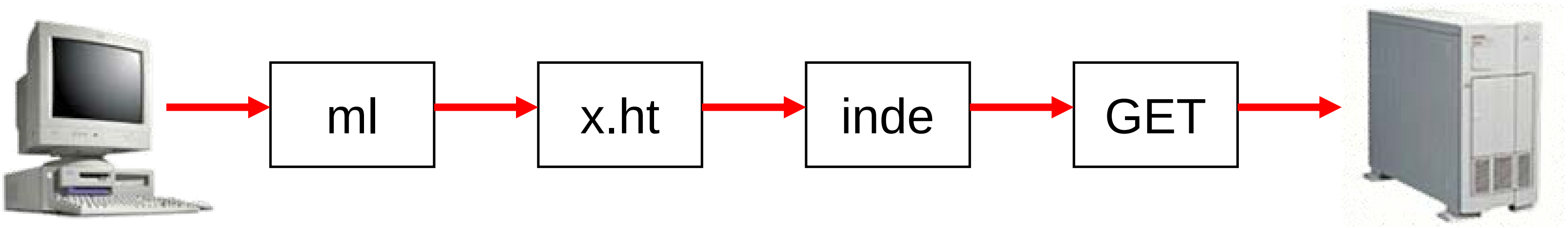
Link Address

What if the Data Doesn’t Fit?

Problem: Packet size

- On Ethernet, max IP packet is 1500 bytes
- Typical Web page is 10 kbytes

Solution: Split the data across multiple packets



ml

x.ht

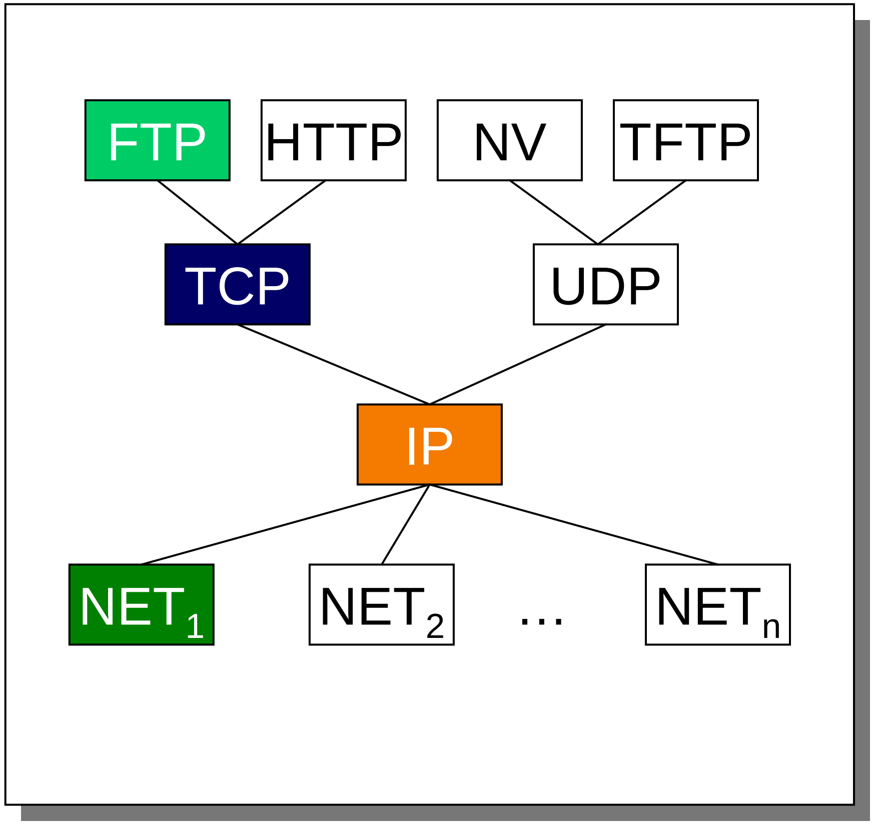
inde

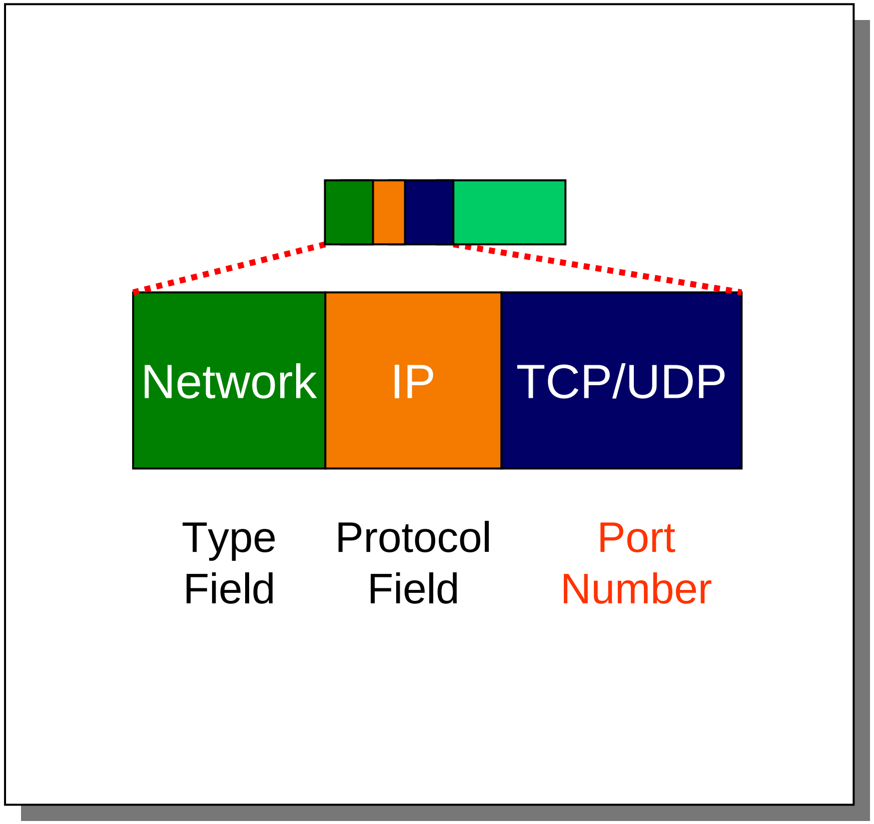
GET

GET index.html

Protocol Demultiplexing

- Multiple choices at each layer

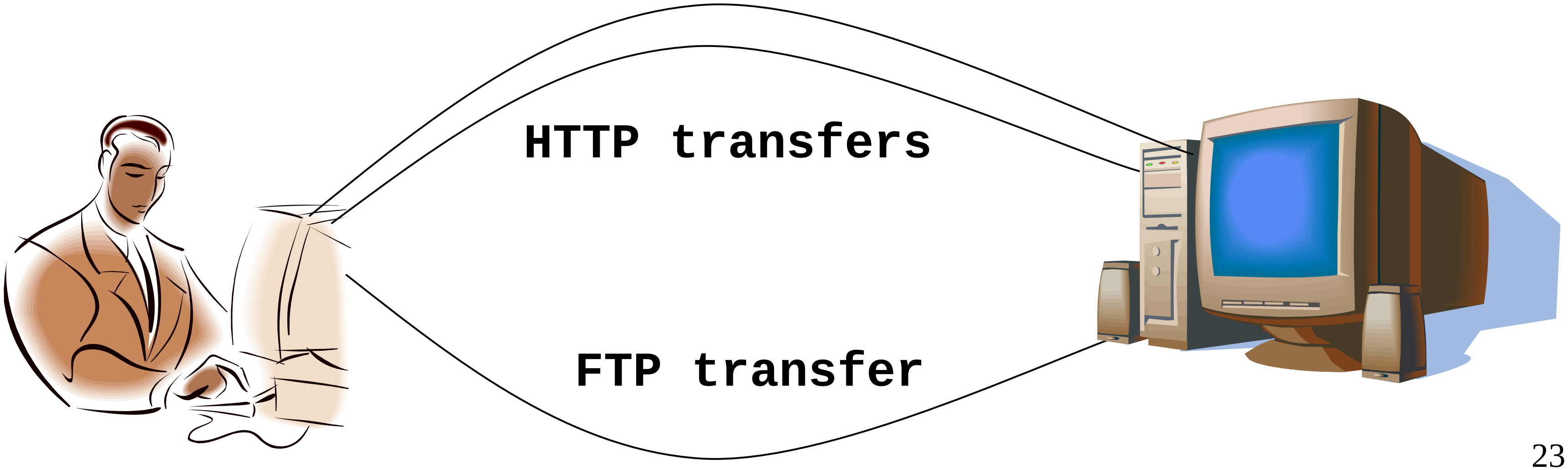




Demultiplexing: Port Numbers

- Differentiate between multiple transfers
  - Knowing source and destination host is not enough
  - Need an id for *each transfer* between the hosts

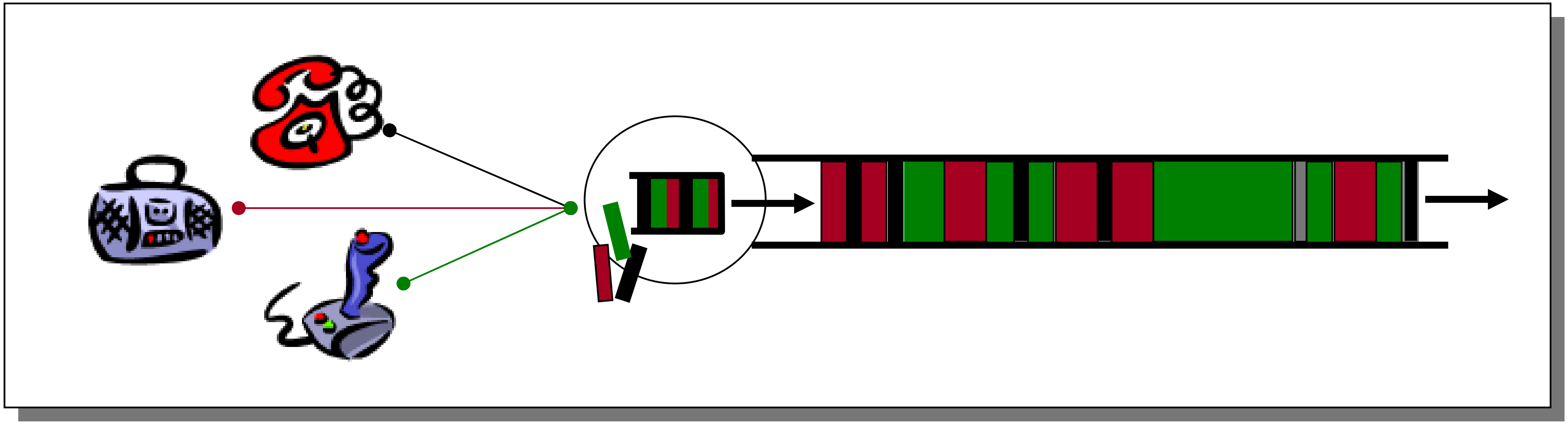
- Specify a particular service running on a host
  - E.g., HTTP server running on port 80
  - E.g., FTP server running on port 21



Is Layering Harmful?

- Layer N may duplicate lower level functionality
  - E.g., error recovery to retransmit lost data
- Layers may need same information
  - E.g., timestamps, maximum transmission unit size
- Strict adherence to layering may hurt performance
  - E.g., hiding details about what is really going on
- Some layers are not always cleanly separated
  - Inter-layer dependencies for performance reasons
  - Some dependencies in standards (header checksums)
- Headers start to get really big
  - Sometimes more header bytes than actual content

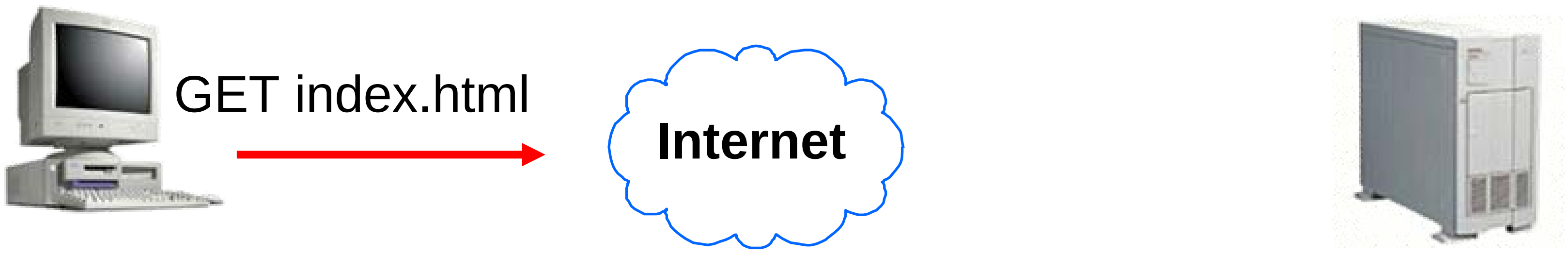
Resource Allocation: Queues



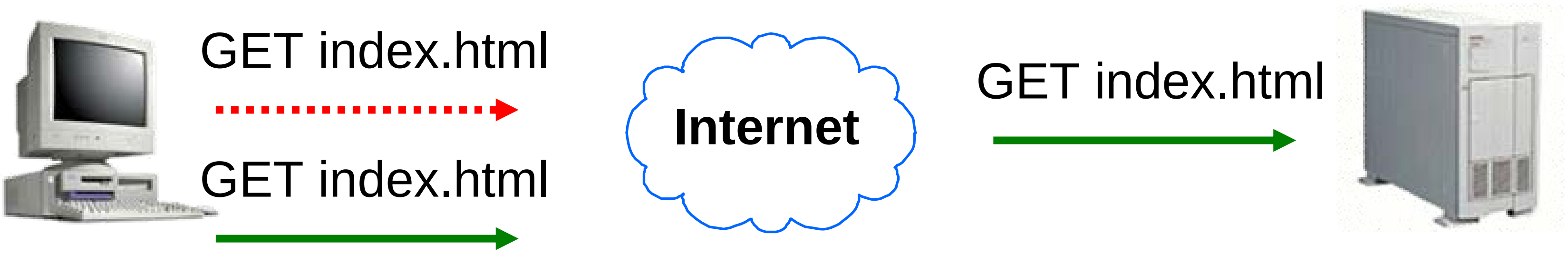
- Sharing access to limited resources
  - E.g., a link with fixed service rate
- Simplest case: first-in-first out queue
  - Serve packets in the order they arrive
  - When busy, store arriving packets in a buffer
  - Drop packets when the queue is full

What if the Data gets Dropped?

Problem: Lost Data

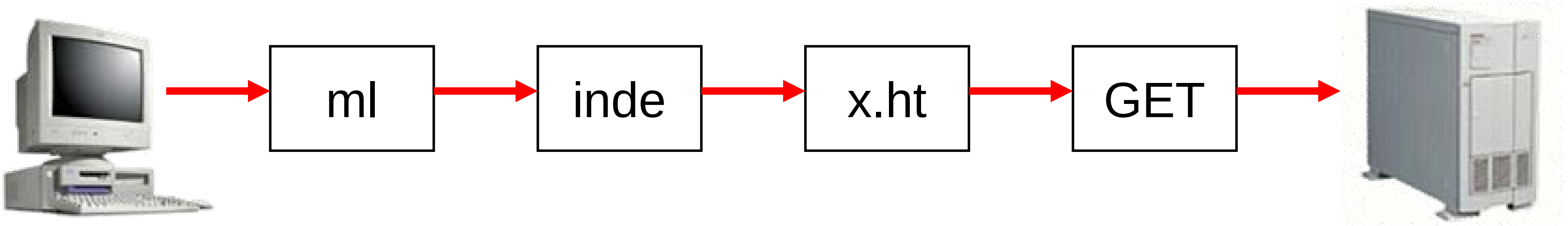


Solution: Timeout and Retransmit



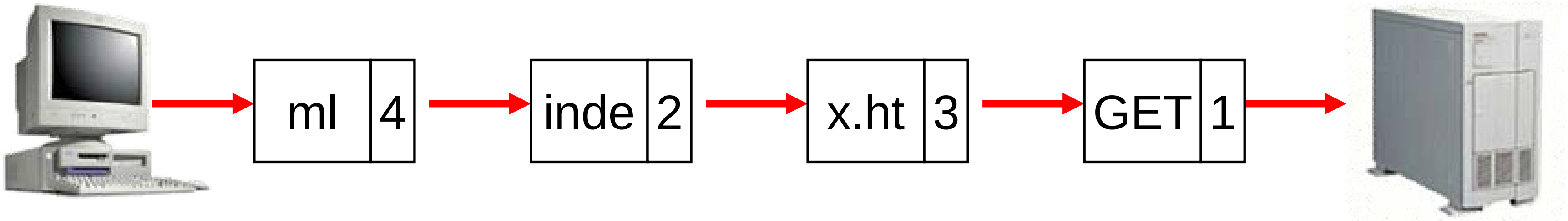
What if the Data is Out of Order?

Problem: Out of Order



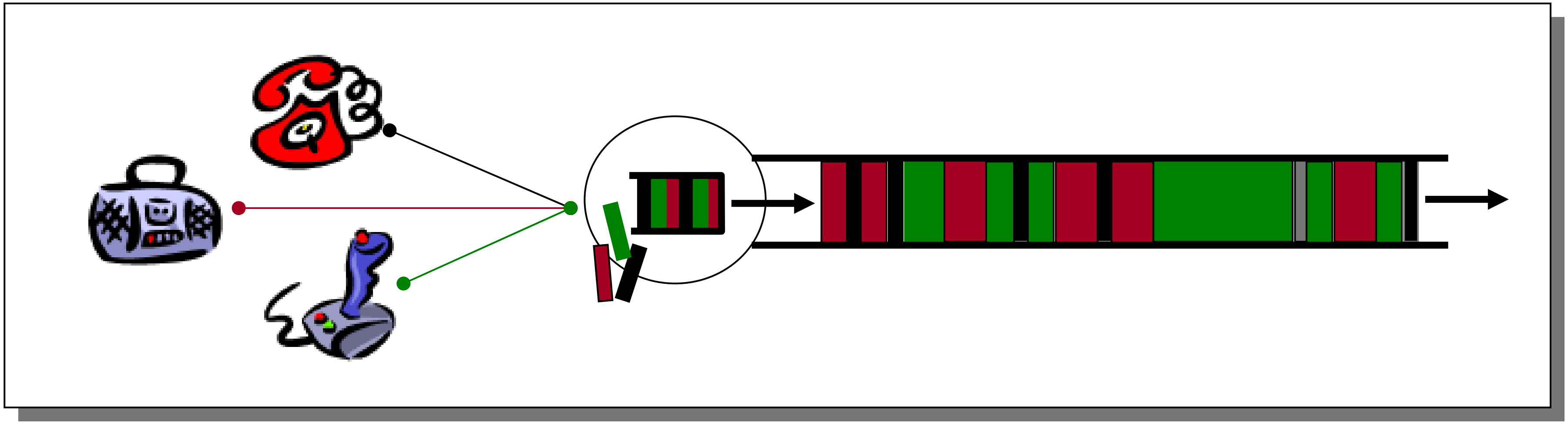
GET x.htindeml

Solution: Add Sequence Numbers



GET index.html

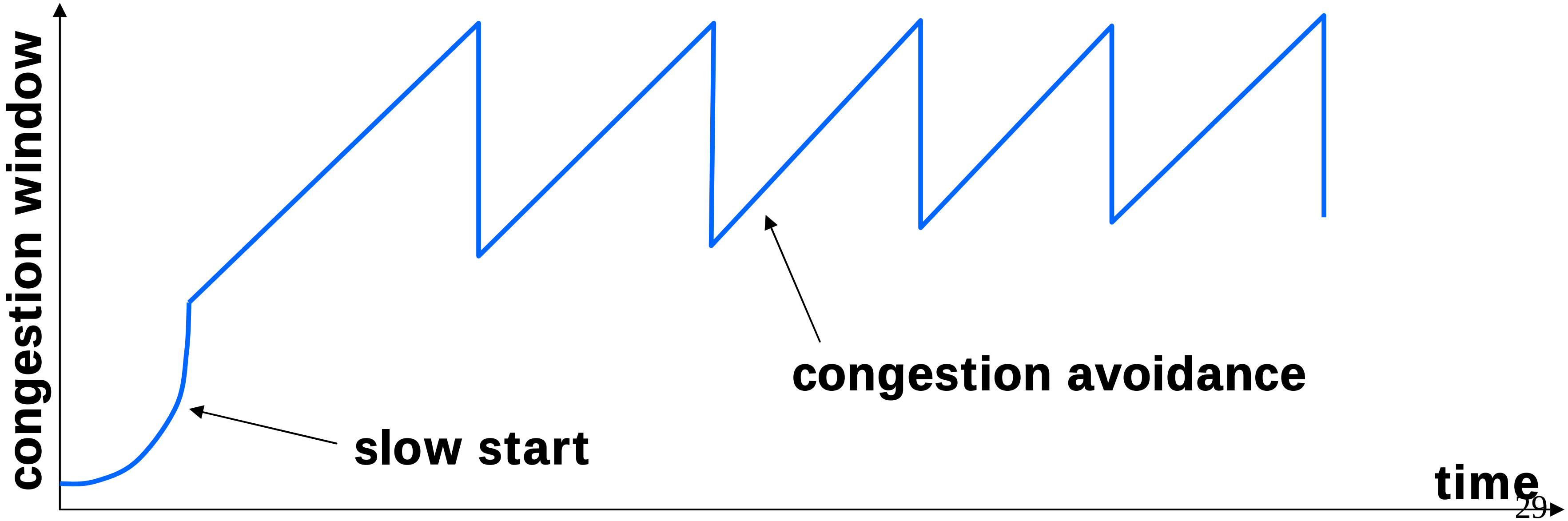
Resource Allocation: Congestion Control



- What if too many folks are sending data?
  - Senders agree to slow down their sending rates
  - … in response to their packets getting dropped
- The essence of TCP congestion control
  - Key to preventing congestion collapse of the Internet

Transmission Control Protocol

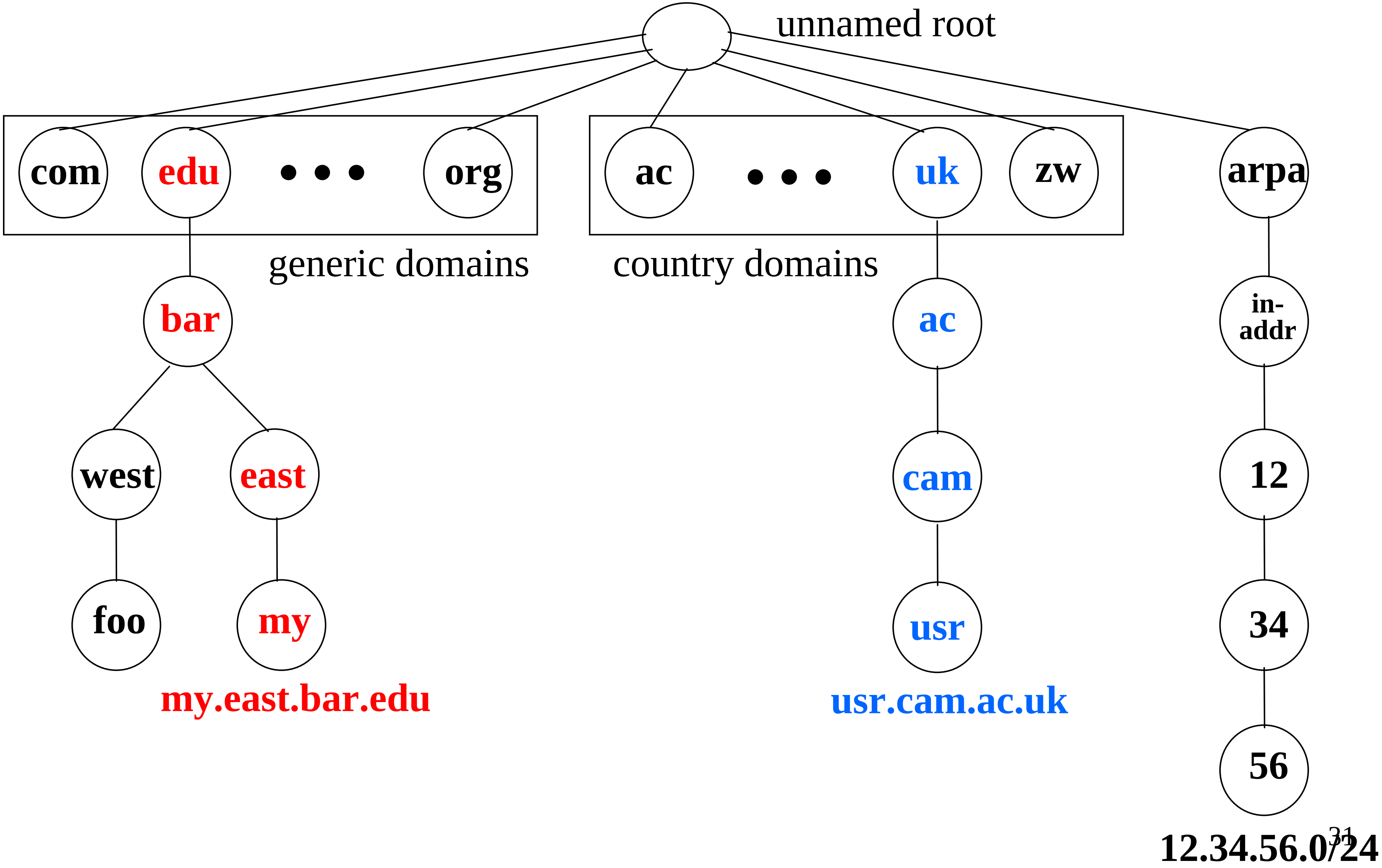
- Flow control: window-based
  - Sender limits number of outstanding bytes (window size)
  - *Receiver window* ensures data does not overflow receiver
- Congestion control: adapting to packet losses
  - *Congestion window* tries to avoid overloading the network (increase with successful delivery, decrease with loss)
  - TCP connection starts with small initial congestion window



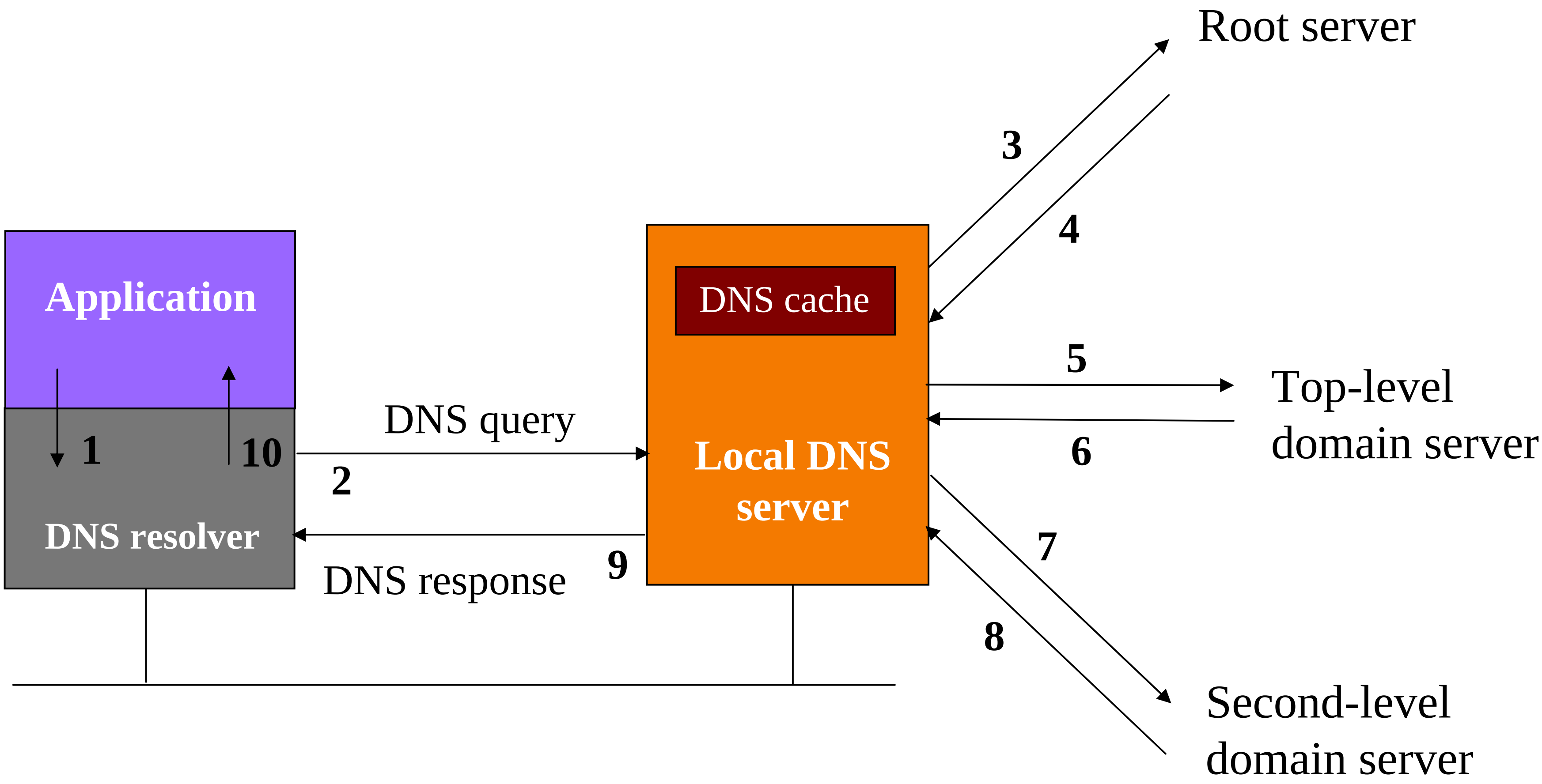
Naming: Domain Name System (DNS)

- Properties of DNS
  - Hierarchical name space divided into zones
  - Translation of names to/from IP addresses
  - Distributed over a collection of DNS servers
- Client application
  - Extract server name (e.g., from the URL)
  - Invoke system call to trigger DNS resolver code
    - E.g., *gethostbyname()* on “www.ee.snu.ac.kr”
- Server application
  - Extract client IP address from socket
  - Optionally invoke system call to translate into name
    - E.g., *gethostbyaddr()* on “12.34.158.5”

Domain Name System



DNS Resolver and Local DNS Server



Caching based on a time-to-live (TTL) assigned by the DNS server responsible for the host name to reduce latency in DNS translation.

Conclusions

- Course objectives
  - Network programming, how the Internet works, and key concepts in networking
- Key concepts in networking
  - Protocols, layers, resource allocation, and naming
- Next lecture: network programming
  - Socket abstraction (important for assignment #1)
  - Read Chapter 1 of the Peterson/Davie book
  - Skim the online reference material on sockets
  - (Re)familiarize yourself with C programming on “hats”

Networked Applications: Sockets

Introduction to Data Networks

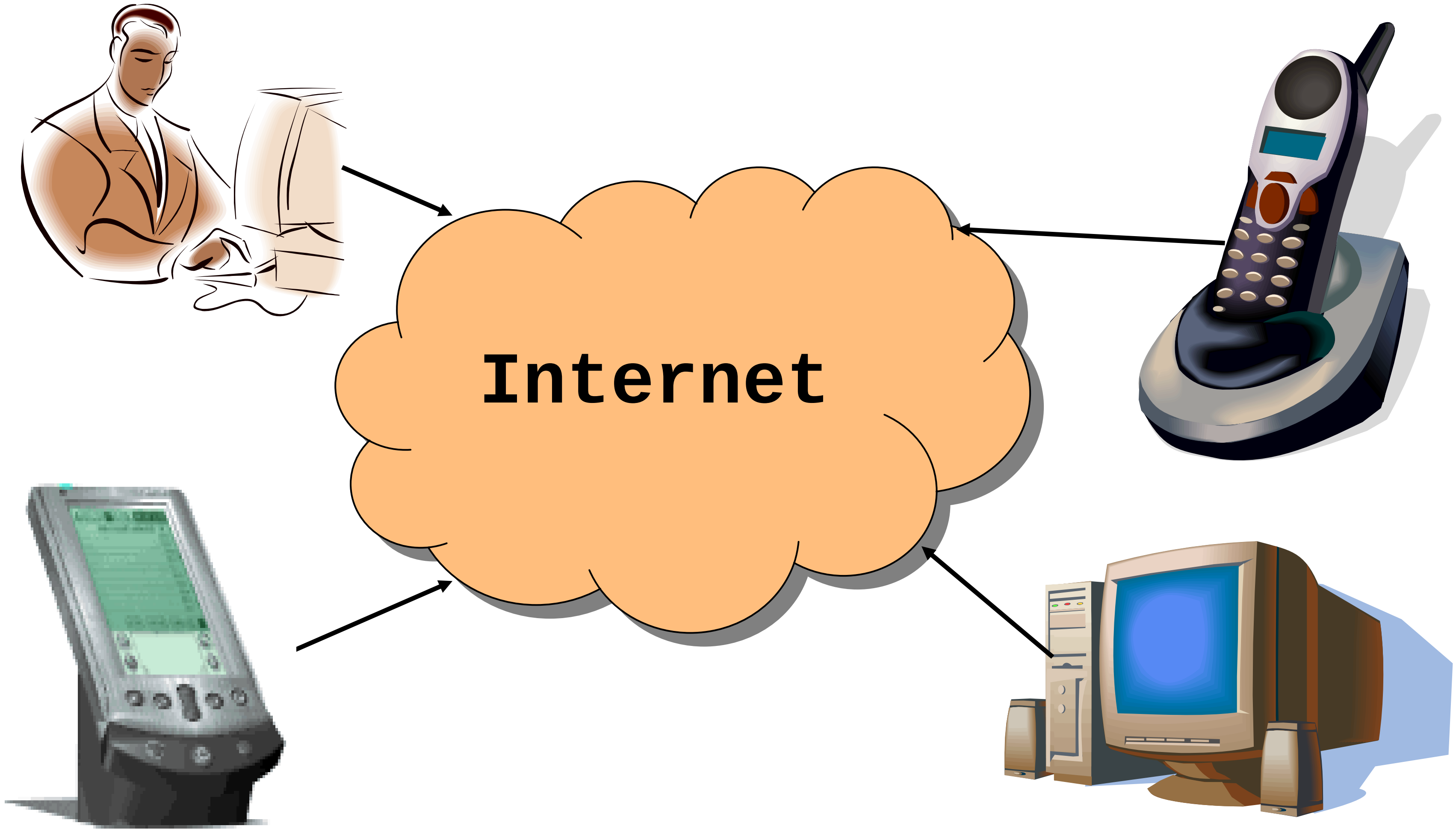
2008.3

Lecture slides from J. Lexford

Goals of Today’s Lecture

- Client-server paradigm
  - End systems
  - Clients and servers
- Sockets
  - Socket abstraction
  - Socket programming
- File-Transfer Protocol (FTP)
  - Uploading and downloading files
  - Separate control and data connections

End System: Computer on the ‘Net



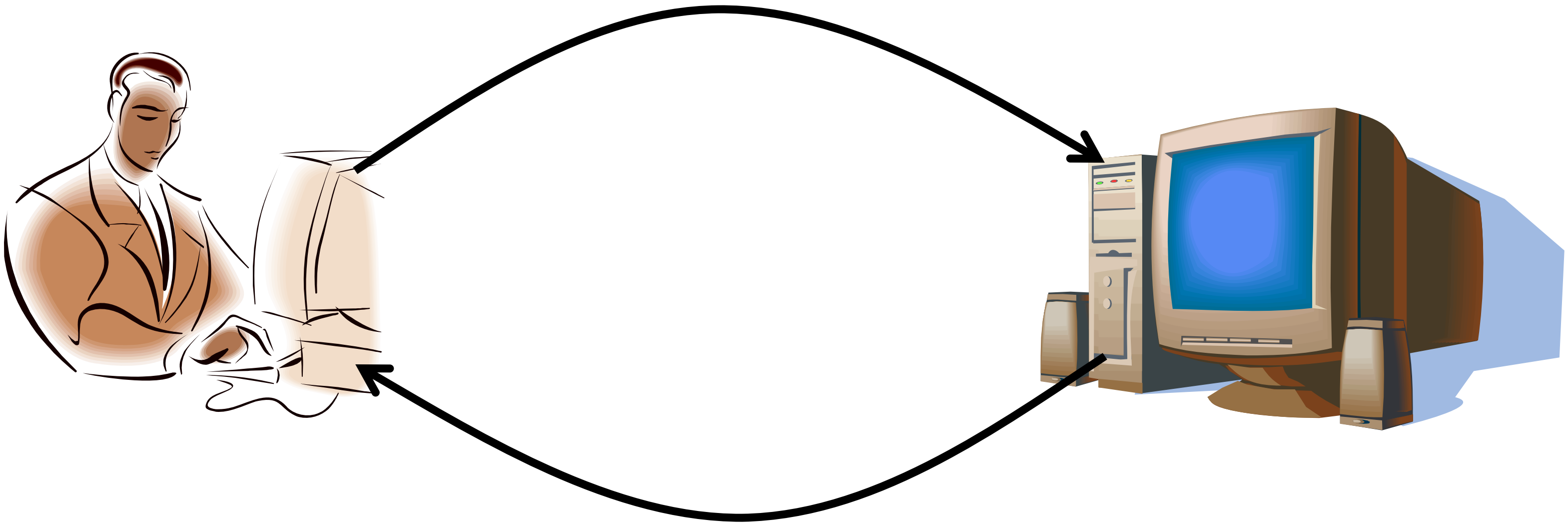
Internet

Also known as a “host”…

Clients and Servers

- Client program
  - –Running on end host
  - –Requests service
  - –E.g., Web browser

- Server program
  - –Running on end host
  - –Provides service
  - –E.g., Web server



`GET /index.html`

`“Site under construction”`

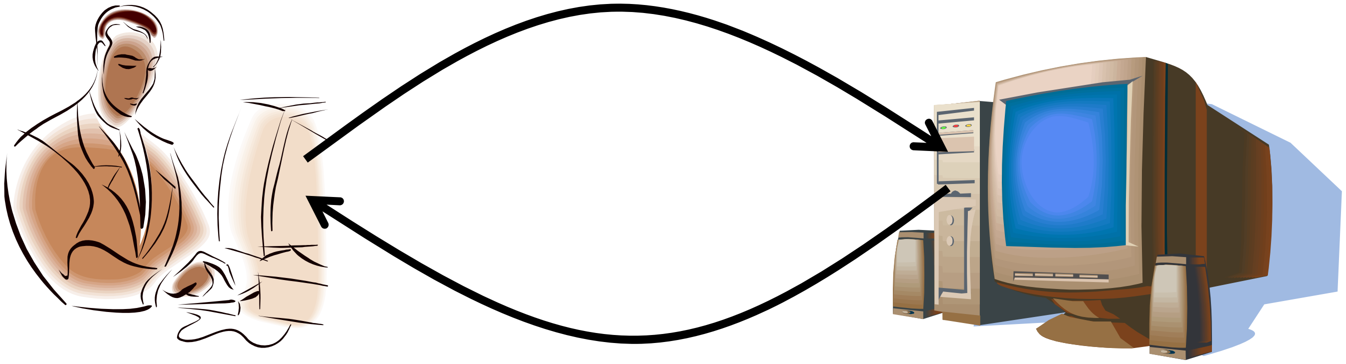
Clients Are Not Necessarily Human

- Example: Web crawler (or spider)
  - Automated client program
  - Tries to discover & download many Web pages
  - Forms the basis of search engines like Google
- Spider client
  - Start with a base list of popular Web sites
  - Download the Web pages
  - Parse the HTML files to extract hypertext links
  - Download these Web pages, too
  - And repeat, and repeat, and repeat…

Client-Server Communication

- Client "sometimes on"
  - Initiates a request to the server when interested
  - E.g., Web browser on your laptop or cell phone
  - Doesn't communicate directly with other clients
  - Needs to know the server's address

- Server is "always on"
  - Services requests from many client hosts
  - E.g., Web server for the www.cnn.com Web site
  - Doesn't initiate contact with the clients
  - Needs a fixed, well-known address



Peer-to-Peer Communication

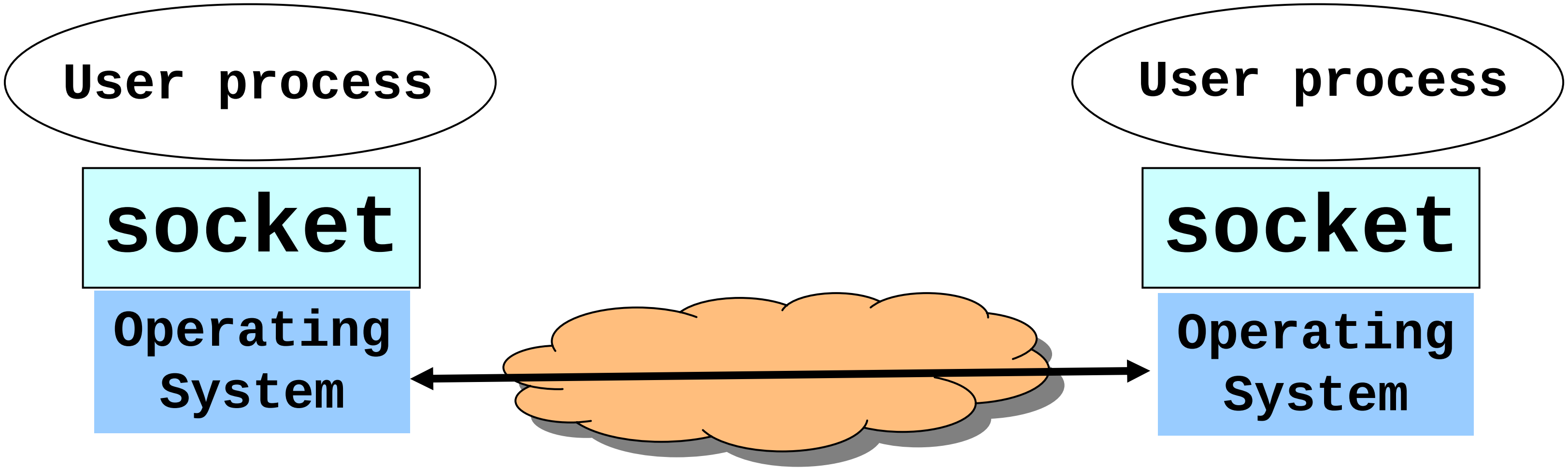
- No always-on server at the center of it all
  - Hosts can come and go, and change addresses
  - Hosts may have a different address each time
- Example: peer-to-peer file sharing
  - Any host can request files, send files, query to find where a file is located, respond to queries, and forward queries
  - Scalability by harnessing millions of peers
  - Each peer acting as both a client and server

Client and Server Processes

- Program vs. process
  - Program: collection of code
  - Process: a running program on a host
- Communication between processes
  - Same end host: inter-process communication
    - Governed by the operating system on the end host
  - Different end hosts: exchanging messages
    - Governed by the network protocols
- Client and server processes
  - Client process: process that initiates communication
  - Server process: process that waits to be contacted

Socket: End Point of Communication

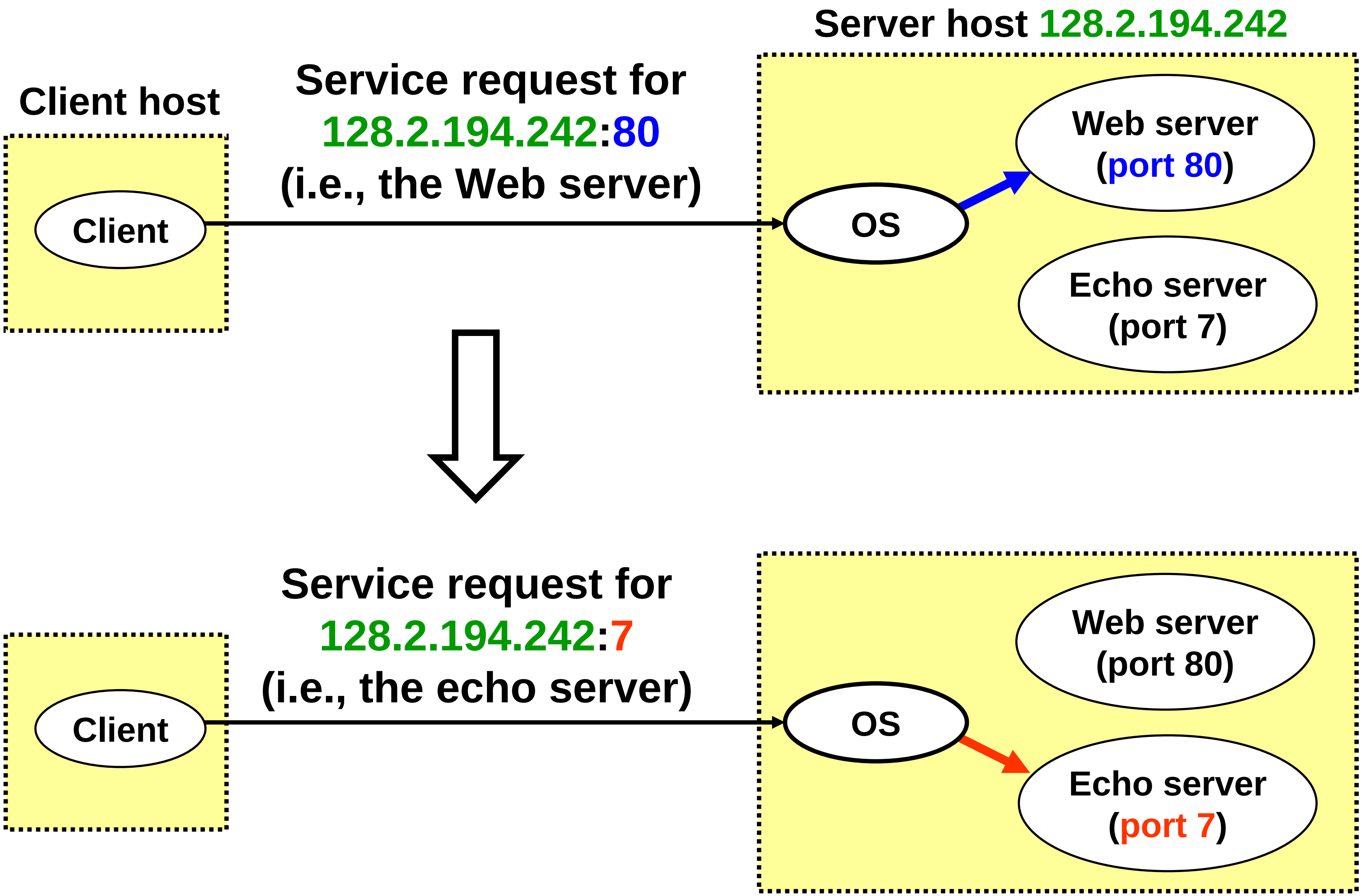
- Sending message from one process to another
  - Message must traverse the underlying network
- Process sends and receives through a "socket"
  - In essence, the doorway leading in/out of the house
- Socket as an Application Programming Interface
  - Supports the creation of network applications



Identifying the Receiving Process

- Sending process must identify the receiver
  - Name or address of the receiving end host
  - Identifier that specifies the receiving process
- Receiving host
  - Destination address that uniquely identifies the host
  - An IP address is a 32-bit quantity
- Receiving process
  - Host may be running many different processes
  - Destination port that uniquely identifies the socket
  - A port number is a 16-bit quantity

Using Ports to Identify Services



Knowing What Port Number To Use

- Popular applications have well-known ports
  - E.g., port 80 for Web and port 25 for e-mail
  - Well-known ports listed at http://www.iana.org
- Well-known vs. ephemeral ports
  - Server has a well-known port (e.g., port 80)
    - Between 0 and 1023
  - Client picks an unused ephemeral (i.e., temporary) port
    - Between 1024 and 65535
- Uniquely identifying the traffic between the hosts
  - Two IP addresses and two port numbers
  - Underlying transport protocol (e.g., TCP or UDP)

Delivering the Data: Division of Labor

- Network
  - –Deliver data packet to the destination host
  - –Based on the destination IP address

- Operating system
  - –Deliver data to the destination socket
  - –Based on the protocol and destination port #
- Application
  - –Read data from the socket
  - –Interpret the data (e.g., render a Web page)

UNIX Socket API

- Socket interface
  - Originally provided in Berkeley UNIX
  - Later adopted by all popular operating systems
  - Simplifies porting applications to different OSes
- In UNIX, everything is like a file
  - All input is like reading a file
  - All output is like writing a file
  - File is represented by an integer file descriptor
- System calls for sockets
  - Client: create, connect, write, read, close
  - Server: create, bind, listen, accept, read, write, close

Typical Client Program

- Prepare to communicate
  - Create a socket
  - Determine server address and port number
  - Initiate the connection to the server
- Exchange data with the server
  - Write data to the socket
  - Read data from the socket
  - Do stuff with the data (e.g., render a Web page)
- Close the socket

Creating a Socket: socket()

- Operation to create a socket
  - *int socket(int domain, int type, int protocol)*
  - Returns a descriptor (or handle) for the socket
  - Originally designed to support any protocol suite
- Domain: protocol family
  - PF_INET for the Internet
- Type: semantics of the communication
  - SOCK_STREAM: reliable byte stream
  - SOCK_DGRAM: message-oriented service
- Protocol: specific protocol
  - UNSPEC: unspecified
  - (PF_INET and SOCK_STREAM already implies TCP)

Connecting the Socket to the Server

- Translating the server's name to an address
  - *struct hostent *gethostbyname(char *name)*
  - Argument: the name of the host (e.g., "www.cnn.com")
  - Returns a structure that includes the host address
- Identifying the service's port number
  - *struct servent *getservbyname(char *name, char *proto)*
  - Arguments: service (e.g., "ftp") and protocol (e.g., "tcp")
- Establishing the connection
  - *int connect(int sockfd, struct sockaddr *server_address, socketlen_t addrlen)*
  - Arguments: socket descriptor, server address, and address size
  - Returns 0 on success, and -1 if an error occurs

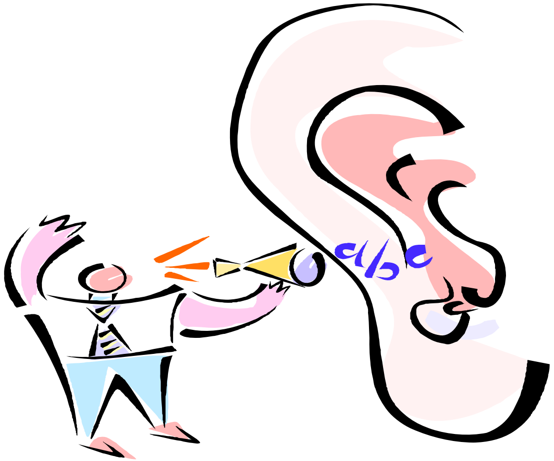
Sending and Receiving Data

- Sending data
  - *ssize_t write(int sockfd, void *buf, size_t len)*
  - Arguments: socket descriptor, pointer to buffer of data to send, and length of the buffer
  - Returns the number of characters written, and -1 on error

- Receiving data
  - *ssize_t read(int sockfd, void *buf, size_t len)*
  - Arguments: socket descriptor, pointer to buffer to place the data, size of the buffer
  - Returns the number of characters read (where 0 implies "end of file"), and -1 on error

- Closing the socket
  - *int close(int sockfd)*

Servers Differ From Clients



- Passive open
  - Prepare to accept connections
  - … but don’t actually establish one
  - … until hearing from a client
- Hearing from multiple clients
  - Allow a backlog of waiting clients
  - ... in case several try to start a connection at once
- Create a socket for each client
  - Upon accepting a new client
  - … create a *new* socket for the communication

Typical Server Program

- Prepare to communicate
  - Create a socket
  - Associate local address and port with the socket
- Wait to hear from a client (passive open)
  - Indicate how many clients-in-waiting to permit
  - Accept an incoming connection from a client
- Exchange data with the client over new socket
  - Receive data from the socket
  - Do stuff to handle the request (e.g., get a file)
  - Send data to the socket
  - Close the socket
- Repeat with the next connection request

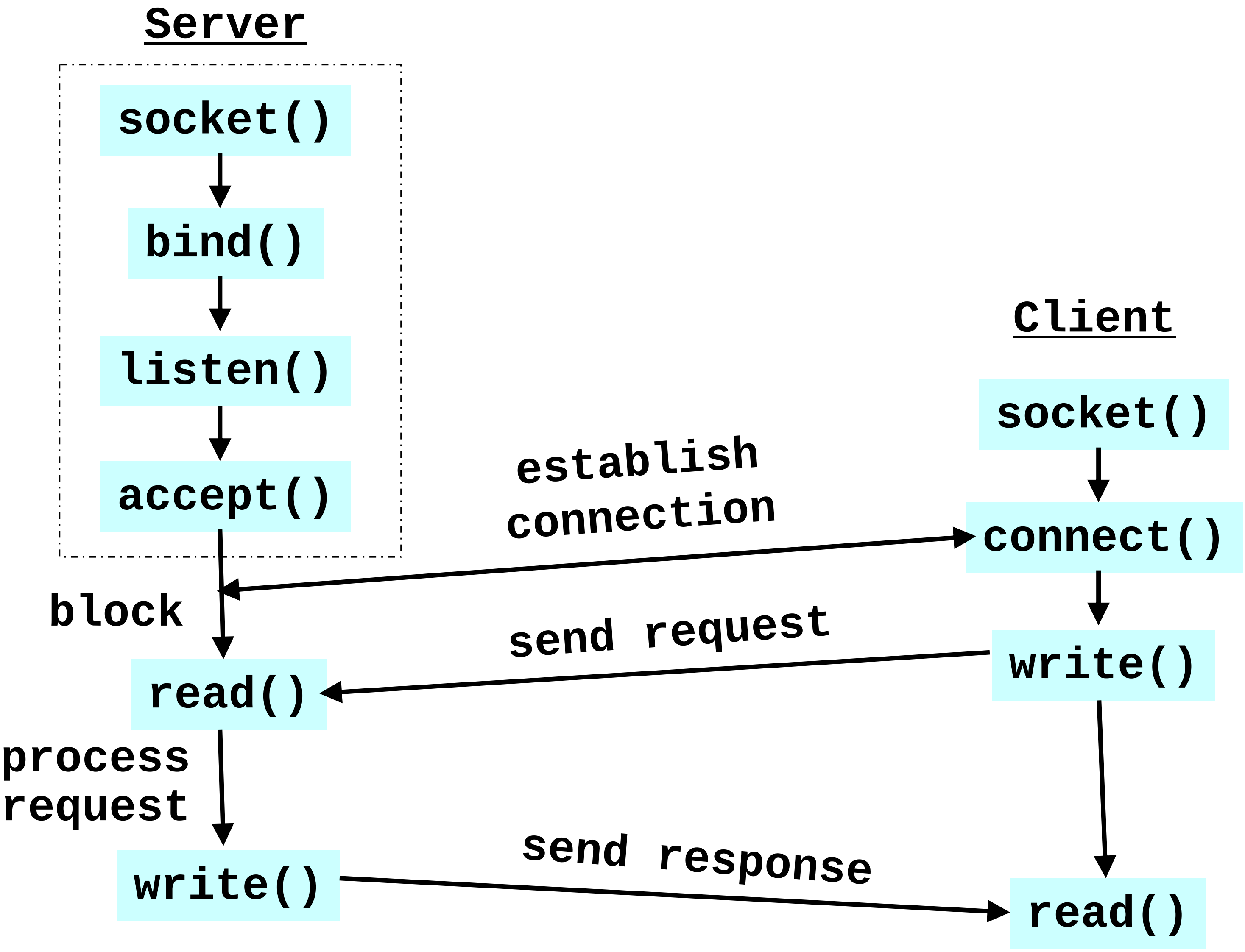
Server Preparing its Socket

- Bind socket to the local address and port number
  - *int bind (int sockfd, struct sockaddr *my_addr, socklen_t addrlen)*
  - Arguments: socket descriptor, server address, address length
  - Returns 0 on success, and -1 if an error occurs
- Define how many connections can be pending
  - *int listen(int sockfd, int backlog)*
  - Arguments: socket descriptor and acceptable backlog
  - Returns 0 on success, and -1 on error

Accepting a New Client Connection

- Accept a new connection from a client
  - *int accept(int sockfd, struct sockaddr *addr, socketlen_t *addrlen)*
  - Arguments: socket descriptor, structure that will provide client address and port, and length of the structure
  - Returns descriptor for a new socket for this connection
- Questions
  - What happens if no clients are around?
    - The *accept()* call blocks waiting for a client
  - What happens if too many clients are around?
    - Some connection requests don’t get through
    - … But, that’s okay, because the Internet makes no promises

Putting it All Together



Serving One Request at a Time?

- Serializing requests is inefficient
  - Server can process just one request at a time
  - All other clients must wait until previous one is done
- Need to time share the server machine
  - Alternate between servicing different requests
    - Do a little work on one request, then switch to another
    - Small tasks, like reading HTTP request, locating the associated file, reading the disk, transmitting parts of the response, etc.
  - Or, start a new process to handle each request
    - Allow the operating system to share the CPU across processes
  - Or, some hybrid of these two approaches

Wanna See Real Clients and Servers?

- Apache Web server
  - Open source server first released in 1995
  - Name derives from “a patchy server” ;-)
  - Software available online at http://www.apache.org
- Mozilla Web browser
  - http://www.mozilla.org/developer/
- Sendmail
  - http://www.sendmail.org/
- BIND Domain Name System
  - Client resolver and DNS server
  - http://www.isc.org/index.pl?/sw/bind/
- …

Socket Programming

Socket programming

Goal: learn how to build client/server application that communicate using sockets

Socket API

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

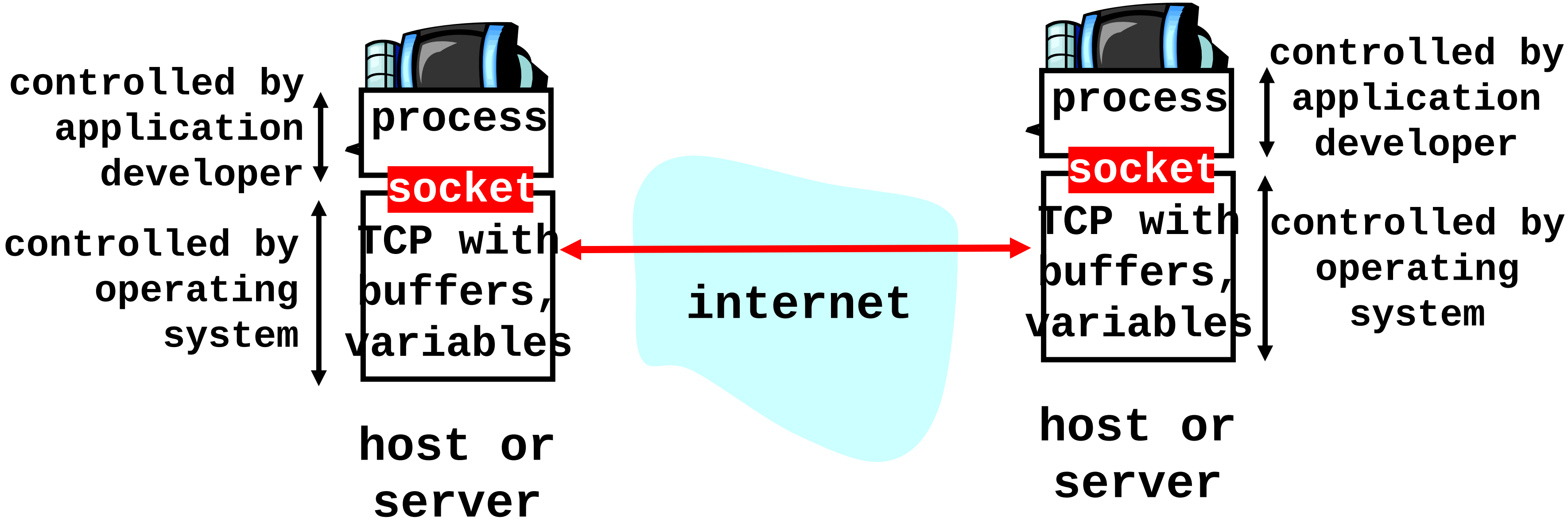
socket

a ***host-local***, ***application-created***, ***OS-controlled*** interface (a "door") into which application process can **both send and receive** messages to/from another application process

Socket-programming using TCP

Socket: a door between application process and end-end-transport protocol (UCP or TCP)

TCP service: reliable transfer of **bytes** from one process to another


controlled by application developer
controlled by operating system
process
socket
TCP with buffers, variables
host or server
internet
process
socket
TCP with buffers, variables
host or server
controlled by application developer
controlled by operating system

Socket programming *with TCP*

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:

- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint

TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

Stream jargon

- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, e.g., keyboard or socket.
- An output stream is attached to an output source, e.g., monitor or socket.

Socket programming with TCP

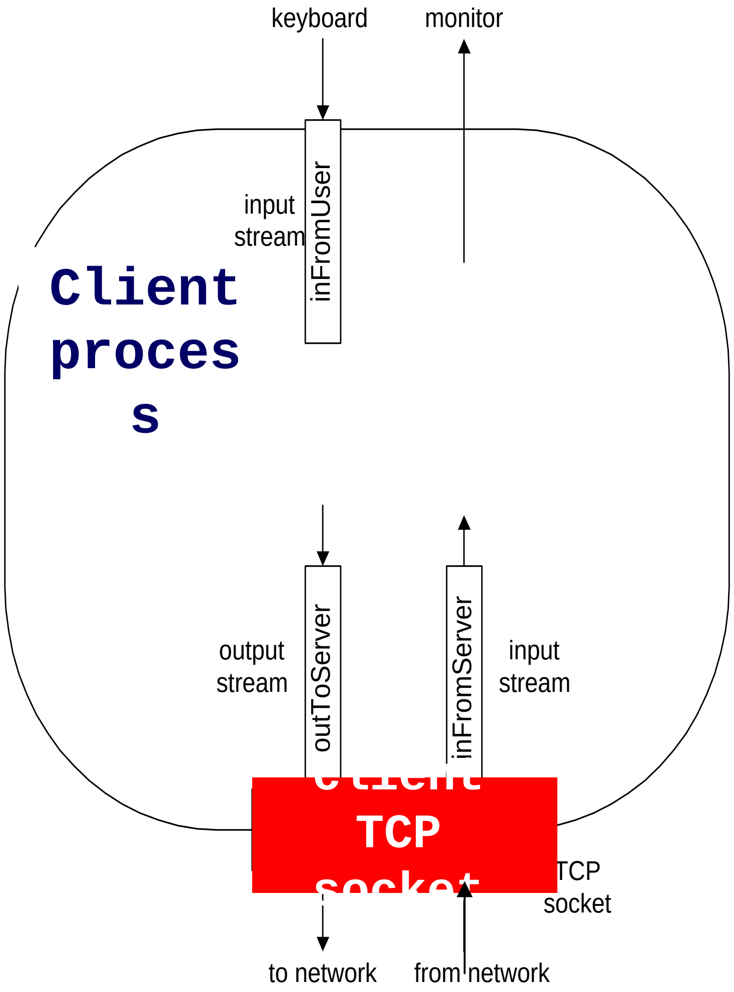
Example client-server app:

1) client reads line from standard input (**inFromUser** stream) , sends to server via socket (**outToServer** stream)

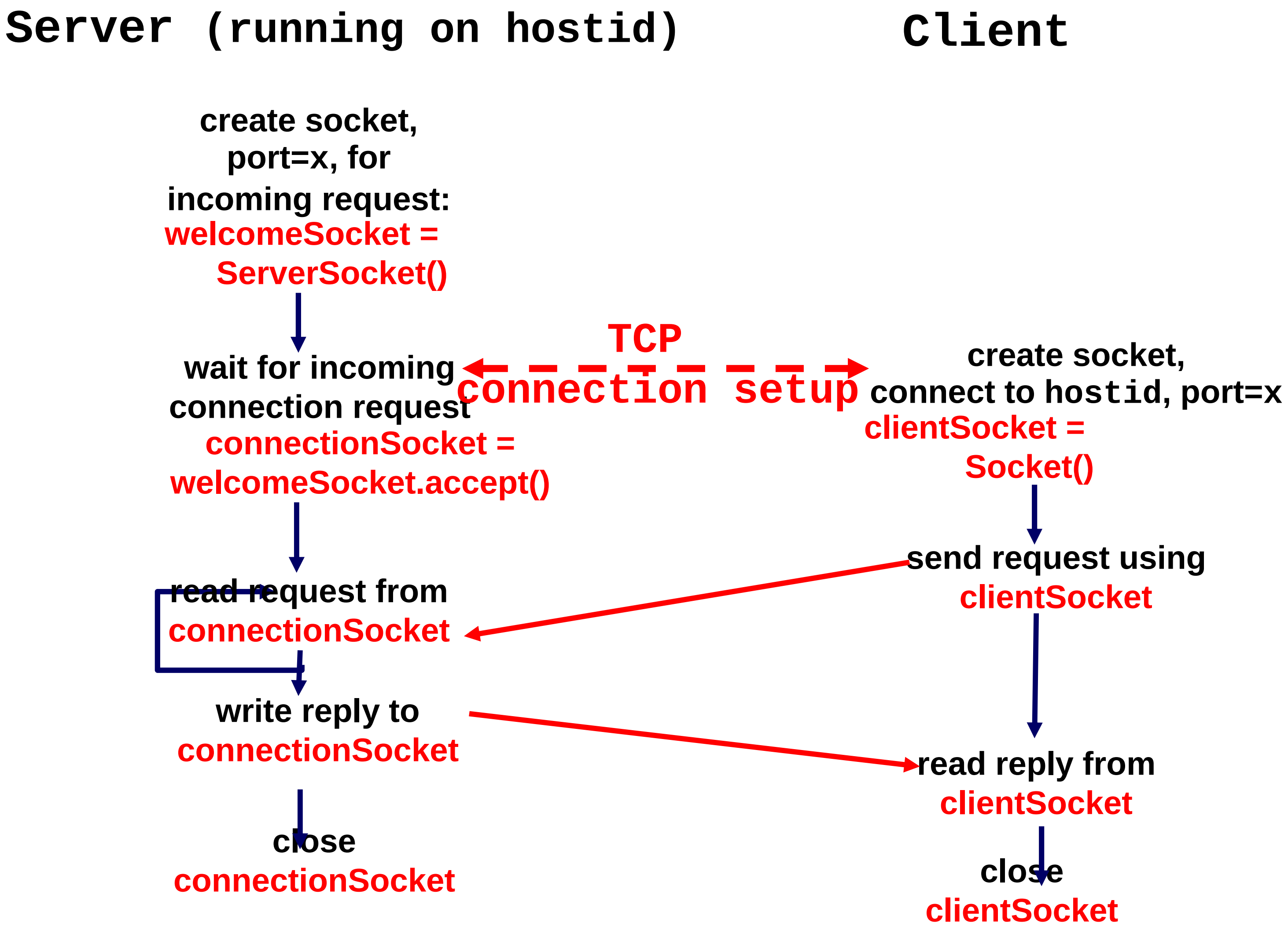
2) server reads line from socket

3) server converts line to uppercase, sends back to client

4) client reads, prints modified line from socket (**inFromServer** stream)



Client/server socket interaction: TCP



Server (running on hostid)

Client

create socket,
port=x, for
incoming request:
welcomeSocket =
ServerSocket()

wait for incoming
connection request
connectionSocket =
welcomeSocket.accept()

TCP
connection setup

create socket,
connect to hostid, port=x
clientSocket =
Socket()

send request using
clientSocket

read request from
connectionSocket

write reply to
connectionSocket

read reply from
clientSocket

close
connectionSocket

close
clientSocket

Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

public static void main(String argv[]) throws Exception
{
String sentence;
String modifiedSentence;
```

Create
input stream

```
BufferedReader inFromUser =
new BufferedReader(new InputStreamReader(System.in));
```

Create
client socket,
onnect to server

```
Socket clientSocket = new Socket("hostname", 6789);
```

Create
output stream
tached to socket

```
DataOutputStream outToServer =
new DataOutputStream(clientSocket.getOutputStream());
```

Example: Java client (TCP), cont.

Create input stream attached to socket

```
BufferedReader inFromServer =
  new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();
```

Send line to server

```
outToServer.writeBytes(sentence + '\n');
```

Read line from server

```
modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

    }
}
```

Example: Java server (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

public static void main(String argv[]) throws Exception
{
String clientSentence;
String capitalizedSentence;
```

Create
welcoming socket
at port 6789

```
ServerSocket welcomeSocket = new ServerSocket(6789);

while(true) {
```

ait, on welcoming
ocket for contact
by client

```
Socket connectionSocket = welcomeSocket.accept();
```

Create input
stream, attached
to socket

```
BufferedReader inFromClient =
new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
```

Example: Java server (TCP), cont

Create output stream, attached to socket

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());
```

Read in line from socket

```
clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Write out line to socket

```
outToClient.writeBytes(capitalizedSentence);
       }
     }
}
```

End of while loop, loop back and wait for another client connection

Building a simple Web server

- handles one HTTP request
- accepts the request
- parses header
- obtains requested file from server's file system
- creates HTTP response message:
  - header lines + file
- sends response to client

- after creating server, you can request file using a browser (e.g., IE explorer)

Data Links

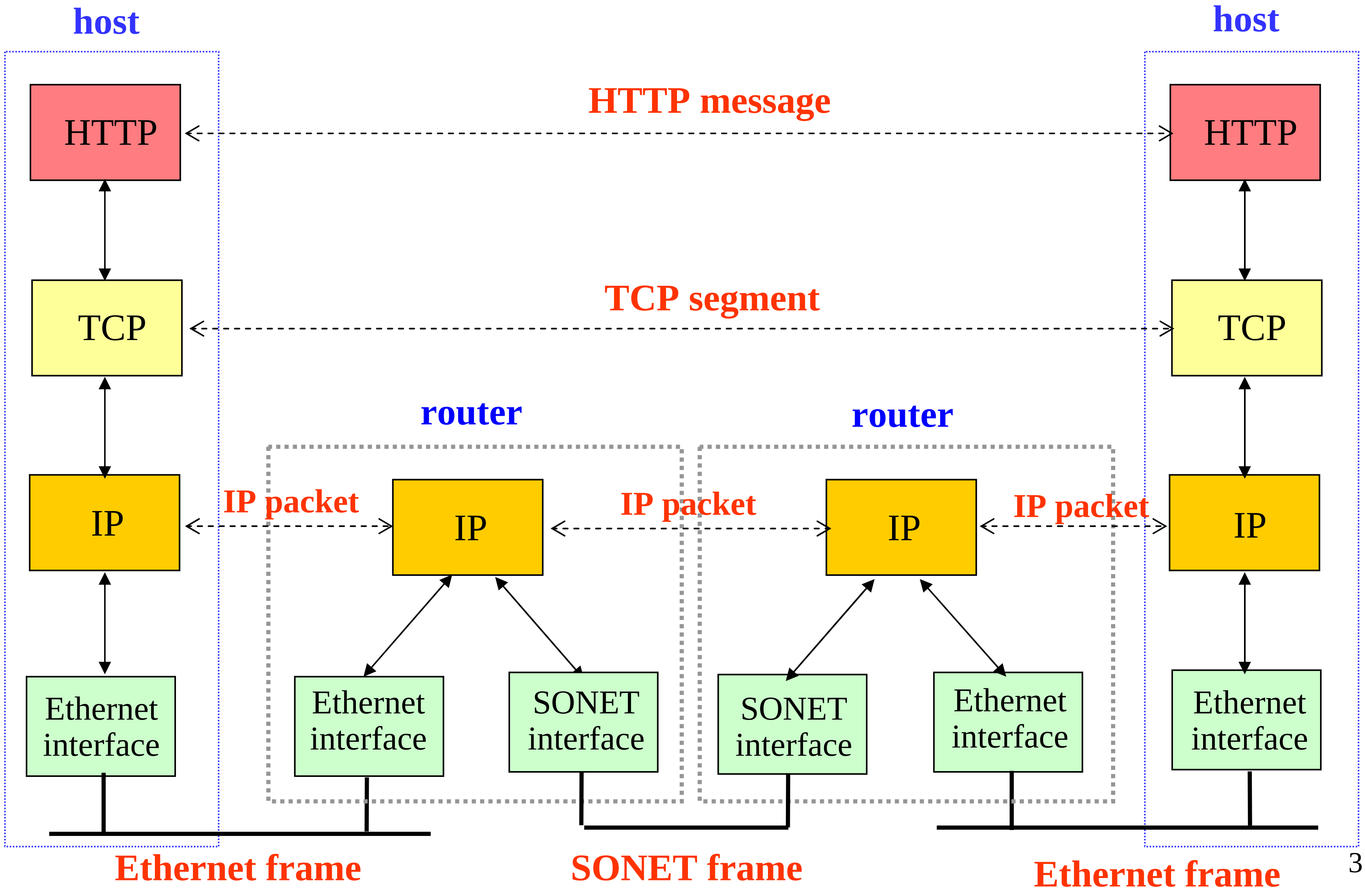
Introduction to Data Networks

2008.3

Goals of Today’s Lecture

- Link-layer services
  - Encoding, framing, and error detection
  - Error correction and flow control
- Sharing a shared media
  - Channel partitioning
  - Taking turns
  - Random access
- Ethernet protocol
  - Carrier sense, collision detection, and random access
  - Frame structure
  - Hubs and switches

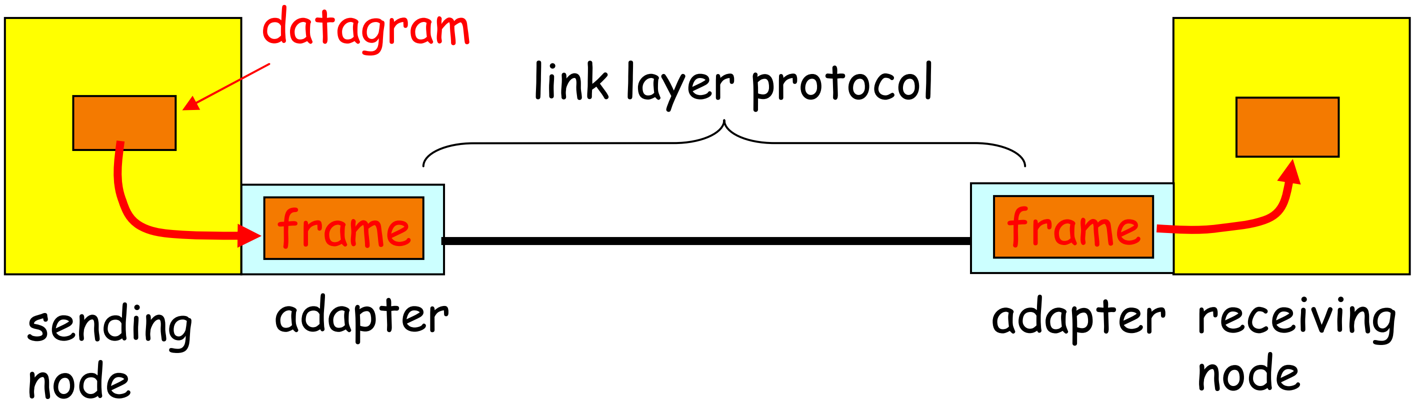
Message, Segment, Packet, and Frame



Link Layer Protocol for Each Hop

- IP packet transferred over multiple hops
  - Each hop has a link layer protocol
  - May be different on different hops
- Analogy: trip from Princeton to Lausanne
  - Limo: Princeton to JFK
  - Plane: JFK to Geneva
  - Train: Geneva to Lausanne
- Refining the analogy
  - Tourist == packet
  - Transport segment == communication link
  - Transportation mode == link-layer protocol
  - Travel agent == routing algorithm

Adaptors Communicating



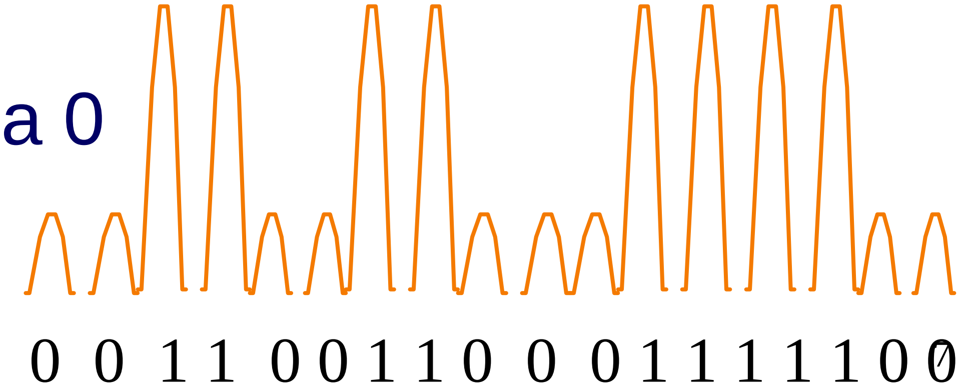
- Link layer implemented in adaptor (network interface card)
  - Ethernet card, PCMCI card, 802.11 card
- Sending side:
  - Encapsulates datagram in a frame
  - Adds error checking bits, flow control, etc.
- Receiving side
  - Looks for errors, flow control, etc.
  - Extracts datagram and passes to receiving node

Link-Layer Services

- Encoding
  - Representing the 0s and 1s
- Framing
  - Encapsulating packet into frame, adding header, trailer
  - Using MAC addresses, rather than IP addresses
- Error detection
  - Errors caused by signal attenuation, noise.
  - Receiver detecting presence of errors
- Error correction
  - Receiver correcting errors without retransmission
- Flow control
  - Pacing between adjacent sending and receiving nodes

Encoding

- Signals propagate over physical links
  - –Source node encodes the bits into a signal
  - –Receiving node decodes the signal back into bits
- Simplify some electrical engineering details
  - –Assume two discrete signals, high and low
  - –E.g., could correspond to two different voltages
- Simple approach
  - –High for a 1, low for a 0

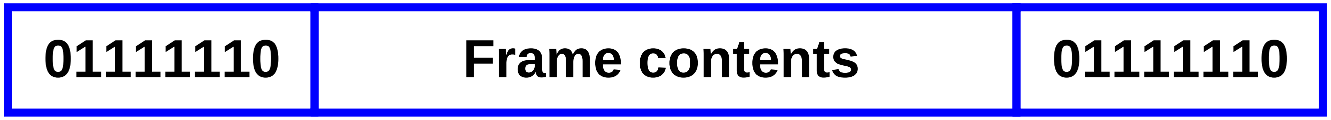


Problem With Simple Approach

- Long strings of 0s or 1s introduce problems
  - No transitions from low-to-high, or high-to-low
- Receiver keeps average of signal it has received
  - Uses the average to distinguish between high and low
  - Long flat strings make receiver sensitive to small change
- Transitions also necessary for clock recovery
  - Receiver uses transitions to drive its own clock
  - Long flat strings do not produce any transitions
  - Can lead to clock drift at the receiver
- Alternatives (see Section 2.2)
  - Non-return to zero inverted, and Manchester encoding

Framing

- Break sequence of bits into a frame
  - Typically implemented by the network adaptor
- Sentinel-based
  - Delineate frame with special pattern (e.g., 01111110)



  - Problem: what if special patterns occurs within frame?
  - Solution: escaping the special characters
    - E.g., sender always inserts a 0 after five 1s
    - … and receiver always removes a 0 appearing after five 1s
  - Similar to escaping special characters in C programs

Framing (Continued)

- Counter-based
  - Include the payload length in the header
  - … instead of putting a sentinel at the end
  - Problem: what if the count field gets corrupted?
    - Causes receiver to think the frame ends at a different place
  - Solution: catch later when doing error detection
    - And wait for the next sentinel for the start of a new frame
- Clock-based
  - Make each frame a fixed size
  - No ambiguity about start and end of frame
  - But, may be wasteful

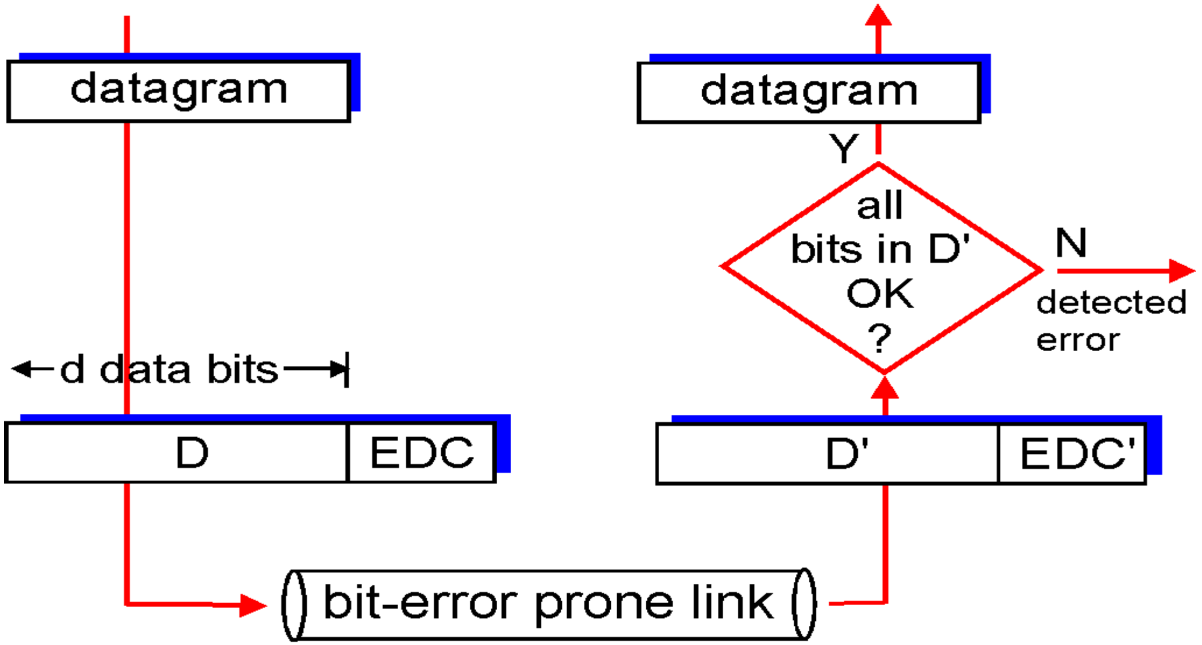
Error Detection

- Errors are unavoidable
  - Electrical interference, thermal noise, etc.
- Error detection
  - Transmit extra (redundant) information
  - Use redundant information to detect errors
  - Extreme case: send two copies of the data
  - Trade-off: accuracy vs. overhead
- Techniques for detecting errors
  - Parity checking
  - Checksum
  - Cyclic Redundancy Check (CRC)

Error Detection

EDC= Error Detection and Correction bits (redundancy)
D = Data protected by error checking, may include header fields

- **Error detection not 100% reliable!**
- **protocol may miss some errors, but rarely**
- **larger EDC field yields better detection and correction**



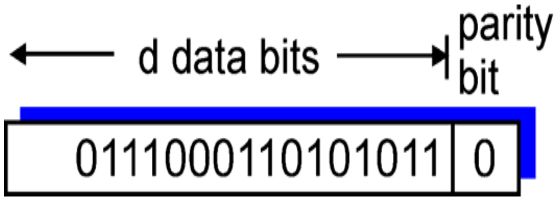
Error Detection Techniques

- Parity check
  - Add an extra bit to a 7-bit code
  - Odd parity: ensure an odd number of 1s
    - E.g., 0101011 becomes 01010111
  - Even parity: ensure an even number of 1s
    - E.g., 0101011 becomes 01010110
- Checksum
  - Treat data as a sequence of 16-bit words
  - Compute a sum of all the 16-bit words, with no carries
  - Transmit the sum along with the packet
- Cyclic Redundancy Check (CRC)

Parity Checking

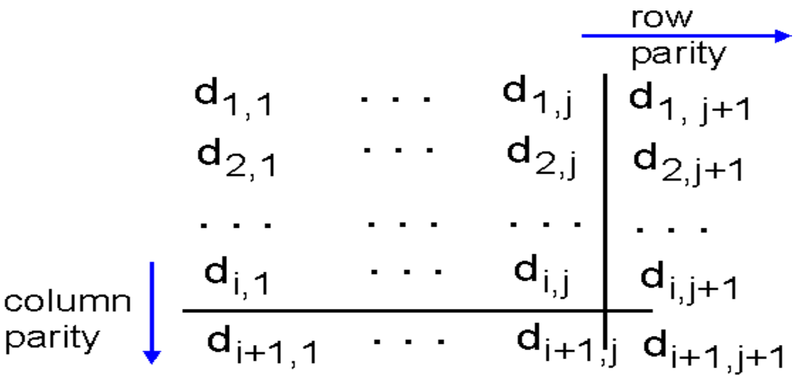
Single Bit Parity:

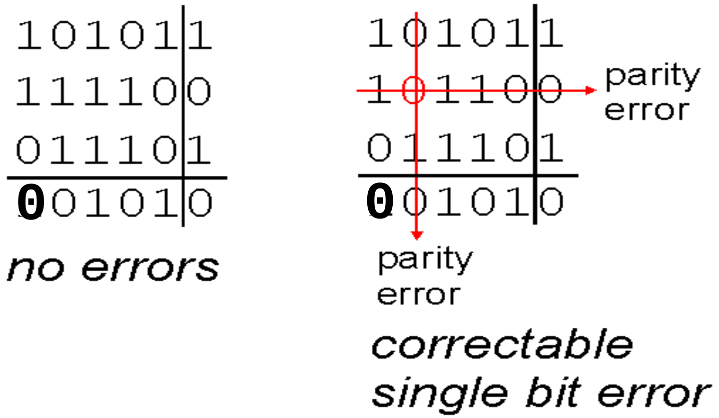
Detect single bit errors



Two Dimensional Bit Parity:

Detect *and correct* single bit errors


row parity
column parity
$d_{1,1}$ . . . $d_{1,j}$ $d_{1,\,j+1}$
$d_{2,1}$ . . . $d_{2,j}$ $d_{2,j+1}$
. . . .
$d_{i,1}$ . . . $d_{i,j}$ $d_{i,j+1}$
$d_{i+1,1}$ . . . $d_{i+1,j}$ $d_{i+1,j+1}$



Internet checksum

Goal: **detect "errors" (e.g., flipped bits) in transmitted segment (note: used at transport layer *only*)**

Sender:

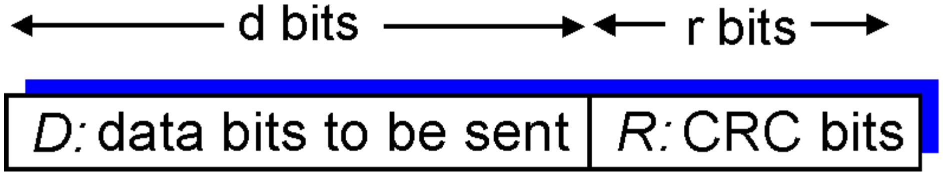
- treat segment contents as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?* More later ….

Cyclic Redundancy Check

- view data bits, D, as a binary number
- choose r+1 bit pattern (generator), G
- goal: choose r CRC bits, R, such that
  - <D,R> exactly divisible by G (modulo 2)
  - receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
  - can detect all burst errors less than r+1 bits
- widely used in practice (ATM, HDCL)



bit pattern

$D * 2^r$ XOR R

mathematical formula

CRC Example

Want:

$D \cdot 2^r$ XOR R = nG

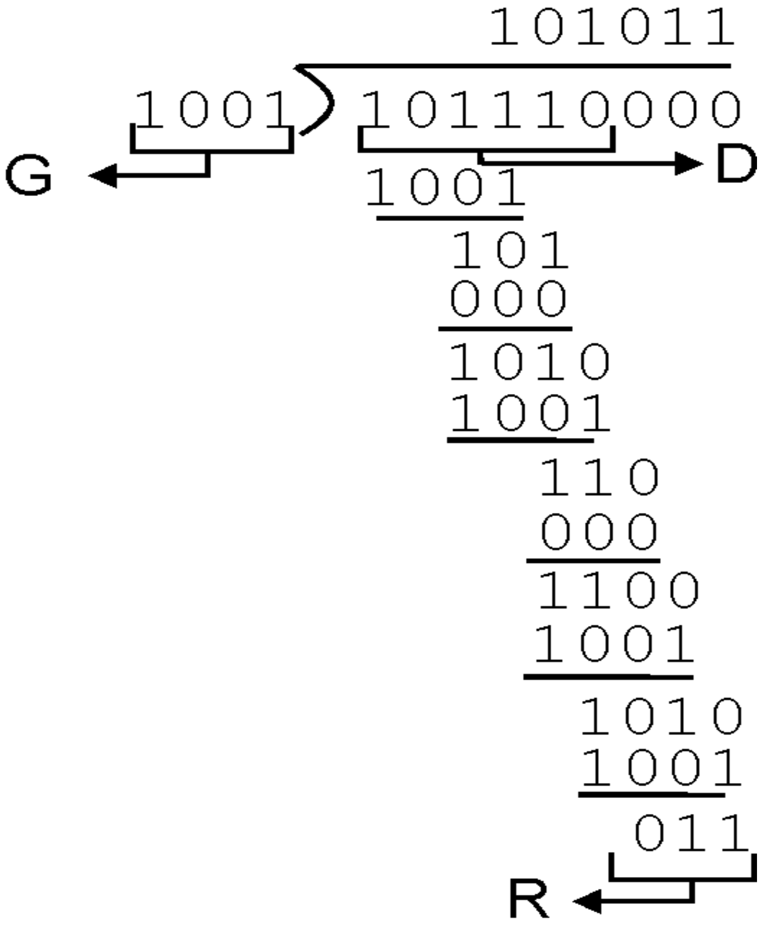
equivalently:

$D \cdot 2^r$ = nG XOR R

equivalently:

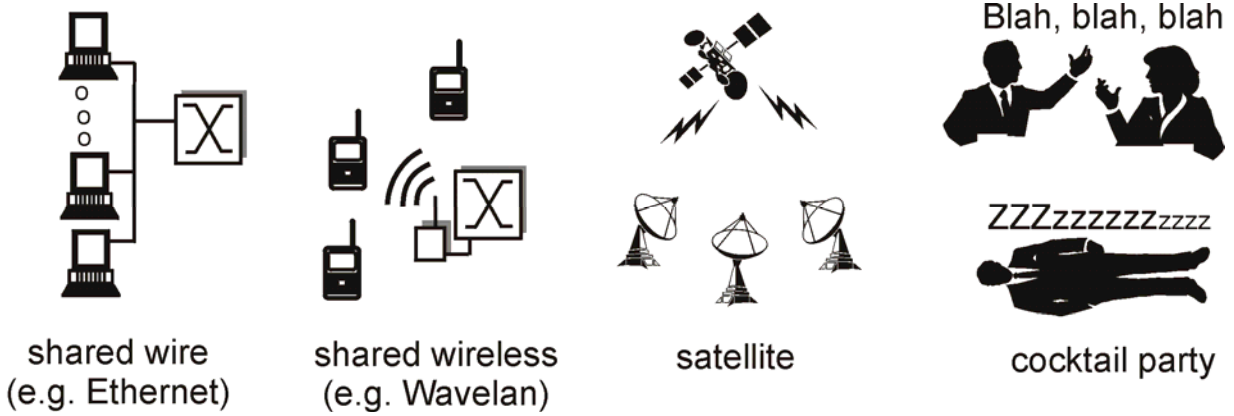
if we divide $D \cdot 2^r$ by G, want remainder R

$$\boxed{R = \text{remainder}\left[\frac{D \cdot 2^r}{G}\right]}$$



Point-to-Point vs. Broadcast Media

- Point-to-point
  - PPP for dial-up access
  - Point-to-point link between Ethernet switch and host
- Broadcast (shared wire or medium)
  - Traditional Ethernet
  - 802.11 wireless LAN



shared wire (e.g. Ethernet)

shared wireless (e.g. Wavelan)

satellite

cocktail party

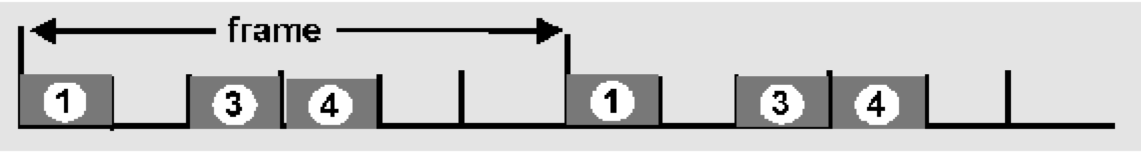
Multiple Access Protocol

- Single shared broadcast channel
  - Avoid having multiple nodes speaking at once
  - Otherwise, collisions lead to garbled data
- Multiple access protocol
  - Distributed algorithm for sharing the channel
  - Algorithm determines which node can transmit
- Classes of techniques
  - Channel partitioning: divide channel into pieces
  - Taking turns: passing a token for the right to transmit
  - Random access: allow collisions, and then recover

Channel Partitioning: TDMA

TDMA: time division multiple access

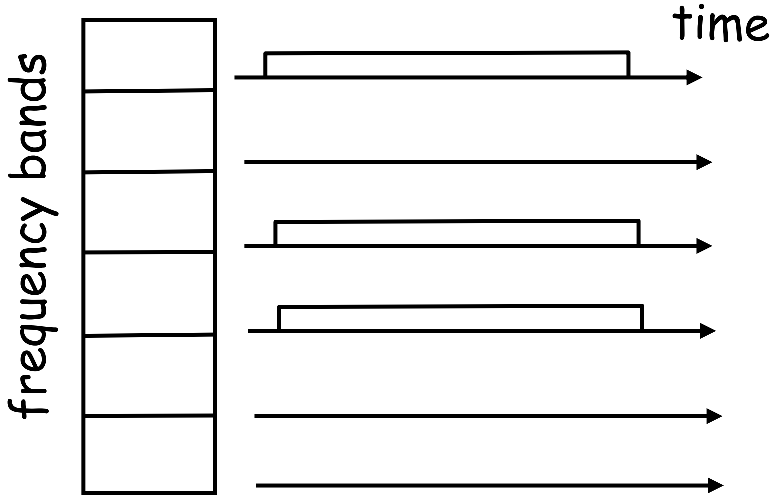
- Access to channel in "rounds"
  - Each station gets fixed length slot in each round
- Time-slot length is packet transmission time
  - Unused slots go idle
- Example: 6-station LAN with slots 1, 3, and 4



Channel Partitioning: FDMA

FDMA: frequency division multiple access

- Channel spectrum divided into frequency bands
  - Each station assigned fixed frequency band
- Unused transmission time in bands go idle
- Example: 6-station LAN with bands 1, 3, and 4



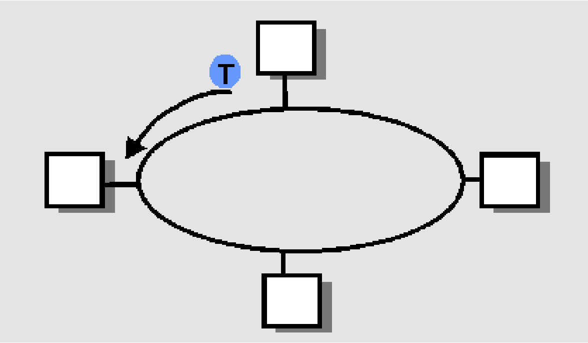
“Taking Turns” MAC protocols

Polling

- Master node “invites” slave nodes to transmit in turn
- Concerns:
  - Polling overhead
  - Latency
  - Single point of failure (master)

Token passing

- Control token passed from one node to next sequentially
- Token message
- Concerns:
  - Token overhead
  - Latency
  - Single point of failure (token)



Random Access Protocols

- When node has packet to send
  - Transmit at full channel data rate R.
  - No a priori coordination among nodes
- Two or more transmitting nodes ➜ “collision”,
- Random access MAC protocol specifies:
  - How to detect collisions
  - How to recover from collisions
- Examples
  - ALOHA and Slotted ALOHA
  - CSMA, CSMA/CD, CSMA/CA

Key Ideas of Random Access

- Carrier sense
  - *Listen before speaking, and don’t interrupt*
  - Checking if someone else is already sending data
  - … and waiting till the other node is done
- Collision detection
  - *If someone else starts talking at the same time, stop*
  - Realizing when two nodes are transmitting at once
  - …by detecting that the data on the wire is garbled
- Randomness
  - *Don’t start talking again right away*
  - Waiting for a random time before trying again

Slotted ALOHA

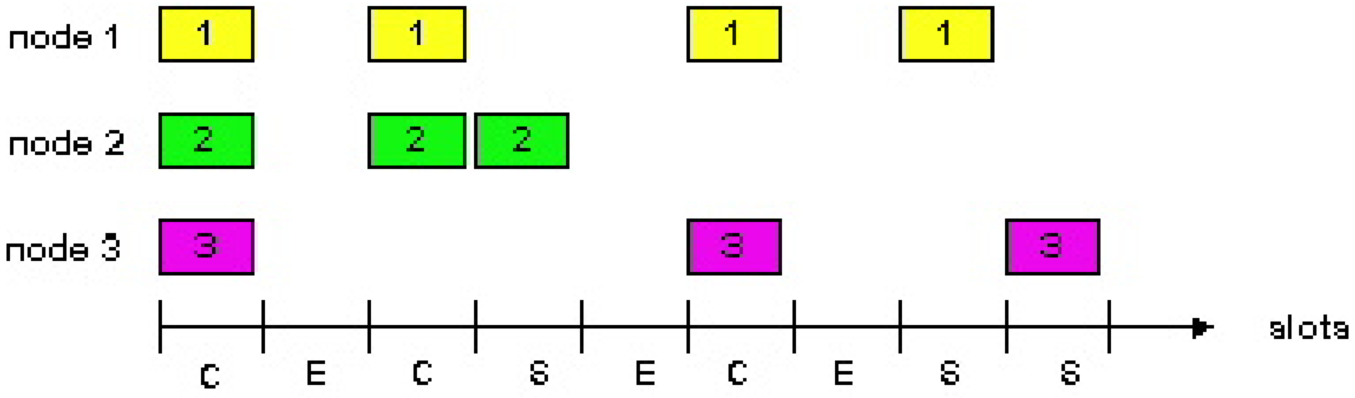
Assumptions

- All frames same size
- Time divided into equal slots (time to transmit a frame)
- Nodes start to transmit frames only at start of slots
- Nodes are synchronized
- If two or more nodes transmit, all nodes detect collision

Operation

- When node obtains fresh frame, transmits in next slot
- No collision: node can send new frame in next slot
- Collision: node retransmits frame in each subsequent slot with probability p until success

Slotted ALOHA



Pros

- Single active node can continuously transmit at full rate of channel
- Highly decentralized: only slots in nodes need to be in sync
- Simple

Cons

- Collisions, wasting slots
- Idle slots
- Nodes may be able to detect collision in less than time to transmit packet
- Clock synchronization

Slotted Aloha efficiency

Efficiency is the long-run fraction of successful slots when there are many nodes, each with many frames to send

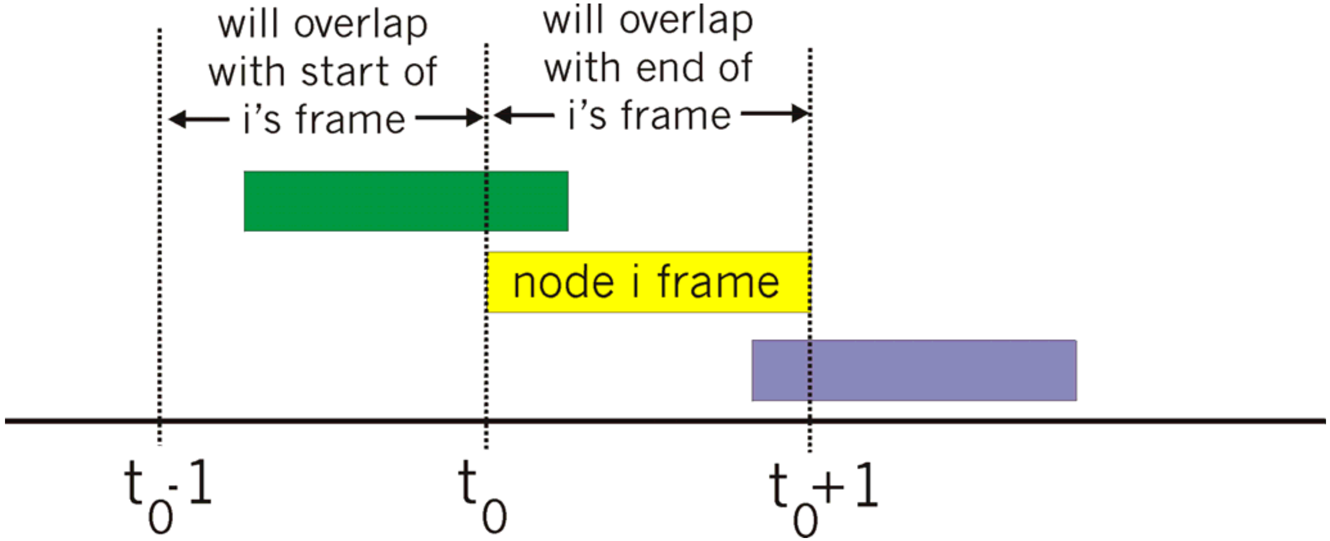
- Suppose N nodes with many frames to send, each transmits in slot with probability *p*
- prob that node 1 has success in a slot = $p(1-p)^{N-1}$
- prob that any node has a success = $Np(1-p)^{N-1}$

- For max efficiency with N nodes, find p* that maximizes $Np(1-p)^{N-1}$
- For many nodes, take limit of $Np^*(1-p^*)^{N-1}$ as N goes to infinity, gives 1/e = .37

At best: **channel used for useful transmissions 37% of time!**

Pure (unslotted) ALOHA

- unslotted Aloha: simpler, no synchronization
- when frame first arrives
  - transmit immediately
- collision probability increases:
  - frame sent at t_0 collides with other frames sent in $[t_0-1, t_0+1]$



will overlap with start of i's frame
will overlap with end of i's frame
node i frame
t_0-1
t_0
t_0+1

Pure Aloha efficiency

P(success by given node) = P(node transmits) ·

P(no other node transmits in $[p_0\text{-}1,p_0]$ ·

P(no other node transmits in $[p_0\text{-}1,p_0]$

$= p \cdot (1-p)^{N-1} \cdot (1-p)^{N-1}$

$= p \cdot (1-p)^{2(N-1)}$

… choosing optimum p and then letting n -> infty ...

Even worse !

= 1/(2e) = .18

CSMA (Carrier Sense Multiple Access)

- Collisions hurt the efficiency of ALOHA protocol
  - At best, channel is useful 37% of the time
- CSMA: listen before transmit
  - If channel sensed idle: transmit entire frame
  - If channel sensed busy, defer transmission
- Human analogy: don’t interrupt others!

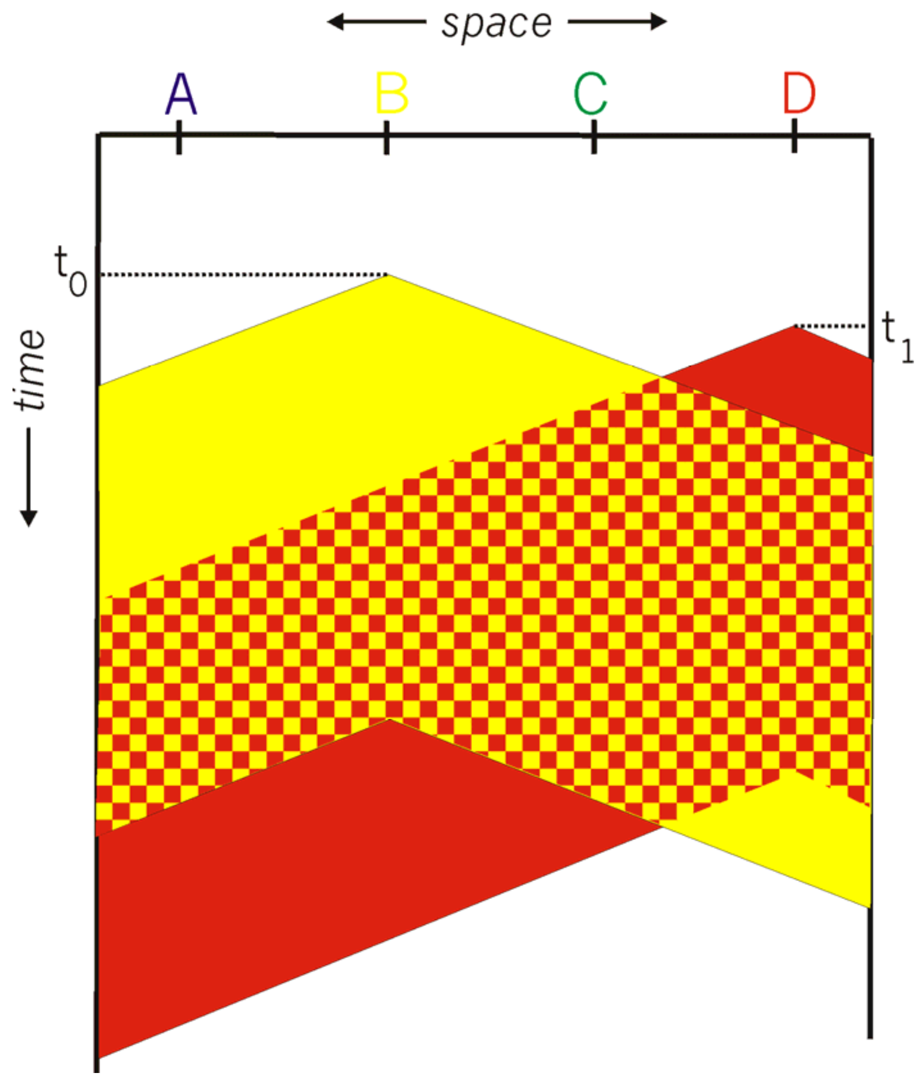
CSMA Collisions

Collisions *can* still occur:

propagation delay means two nodes may not hear each other's transmission

Collision:

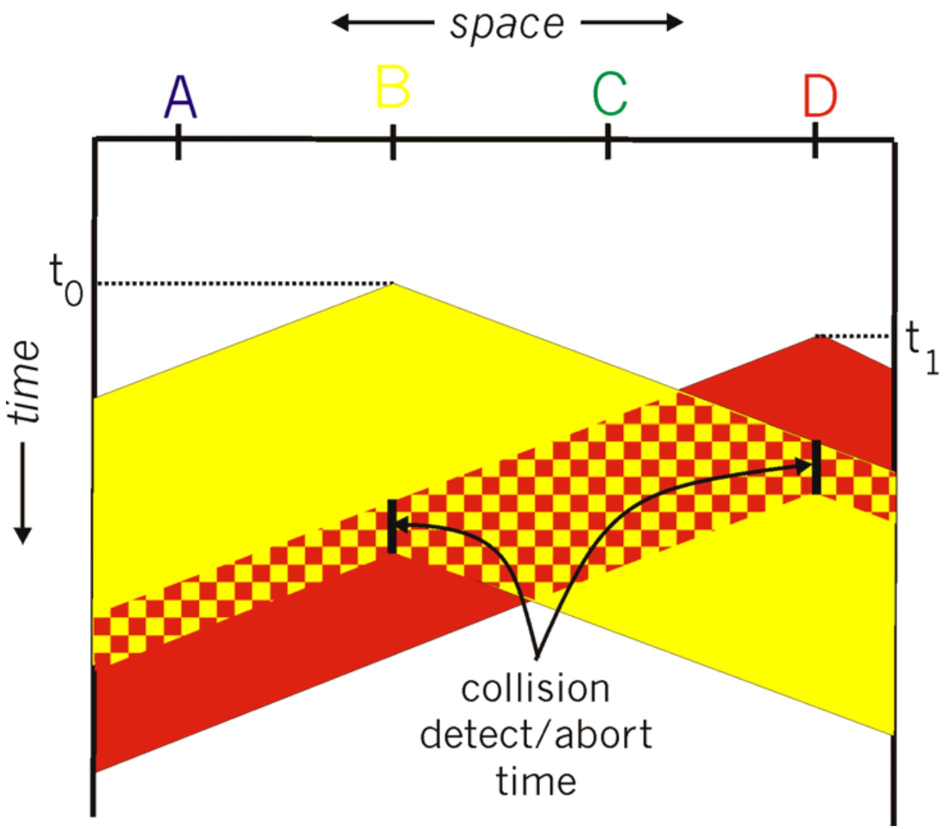
entire packet transmission time wasted



CSMA/CD (Collision Detection)

- CSMA/CD: carrier sensing, deferral as in CSMA
  - Collisions detected within short time
  - Colliding transmissions aborted, reducing wastage
- Collision detection
  - Easy in wired LANs: measure signal strengths, compare transmitted, received signals
  - Difficult in wireless LANs: receiver shut off while transmitting
- Human analogy: the polite conversationalist

CSMA/CD Collision Detection

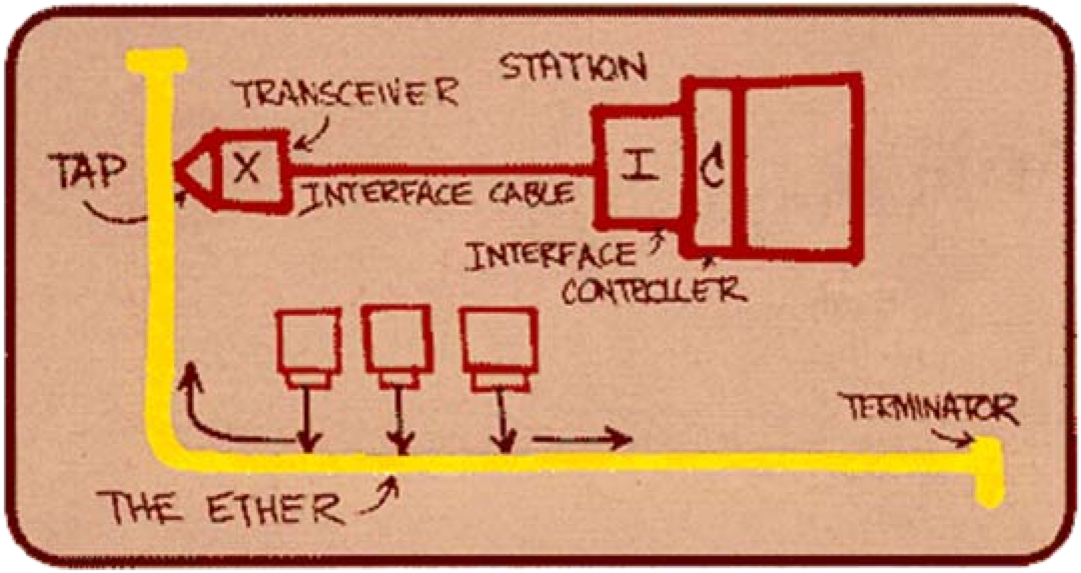


Three Ways to Share the Media

- Channel partitioning MAC protocols:
  - Share channel efficiently and fairly at high load
  - Inefficient at low load: delay in channel access, 1/N bandwidth allocated even if only 1 active node!
- “Taking turns” protocols
  - Eliminates empty slots without causing collisions
  - Vulnerable to failures (e.g., failed node or lost token)
- Random access MAC protocols
  - Efficient at low load: single node can fully utilize channel
  - High load: collision overhead

Ethernet

- Dominant wired LAN technology:
- First widely used LAN technology
- Simpler, cheaper than token LANs and ATM
- Kept up with speed race: 10 Mbps – 10 Gbps

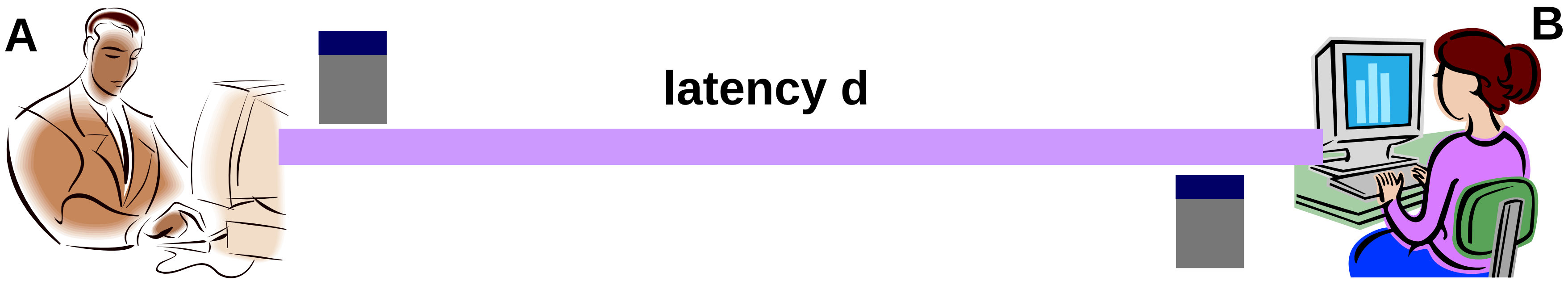


Metcalfe's
Ethernet
sketch

Ethernet Uses CSMA/CD

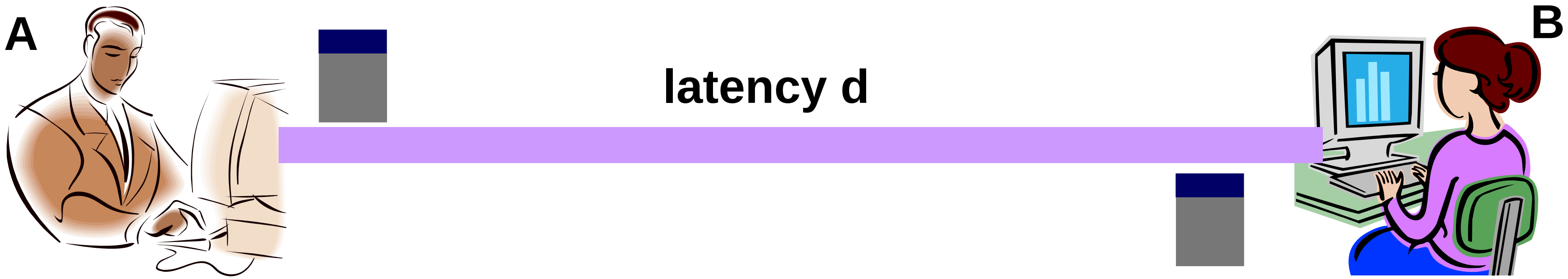
- Carrier sense: wait for link to be idle
  - Channel idle: start transmitting
  - Channel busy: wait until idle
- Collision detection: listen while transmitting
  - No collision: transmission is complete
  - Collision: abort transmission, and send jam signal
- Random access: exponential back-off
  - After collision, wait a random time before trying again
  - After mth collision, choose K randomly from {0, …, 2^m-1}
  - … and wait for K*512 bit times before trying again

Limitations on Ethernet Length



- Latency depends on physical length of link
  - Time to propagate a packet from one end to the other
- Suppose A sends a packet at time t
  - And B sees an idle line at a time just before t+d
  - … so B happily starts transmitting a packet
- B detects a collision, and sends jamming signal
  - But A doesn’t see collision till t+2d

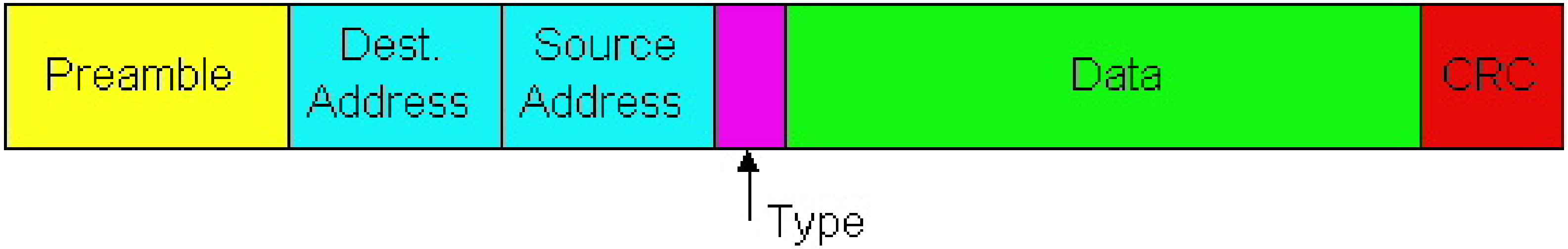
Limitations on Ethernet Length



- A needs to wait for time 2d to detect collision
  - So, A should keep transmitting during this period
  - … and keep an eye out for a possible collision
- Imposes restrictions on Ethernet
  - Maximum length of the wire: 2500 meters
  - Minimum length of the packet: 512 bits (64 bytes)

Ethernet Frame Structure

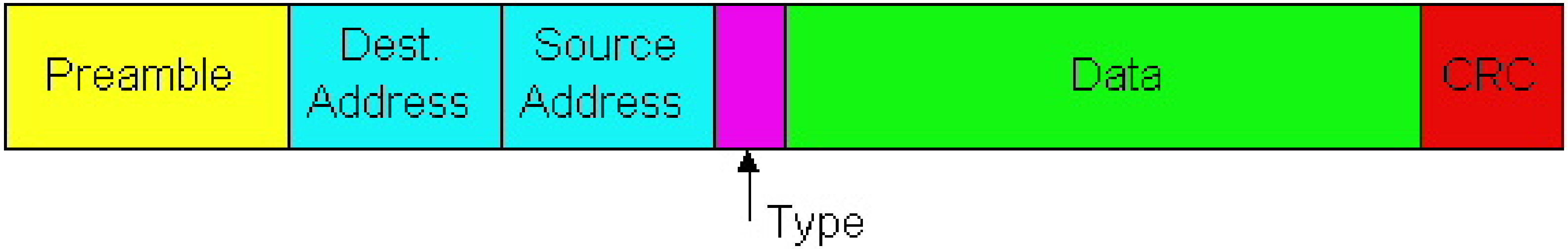
- Sending adapter encapsulates packet in frame



- Preamble: synchronization
  - Seven bytes with pattern 10101010, followed by one byte with pattern 10101011
  - Used to synchronize receiver, sender clock rates

Ethernet Frame Structure (Continued)

- Addresses: source and destination MAC addresses
  - – Adaptor passes frame to network-level protocol
    - If destination address matches the adaptor
    - Or the destination address is the broadcast address
  - – Otherwise, adapter discards frame
- Type: indicates the higher layer protocol
  - – Usually IP
  - – But also Novell IPX, AppleTalk, …
- CRC: cyclic redundancy check
  - – Checked at receiver
  - – If error is detected, the frame is simply dropped

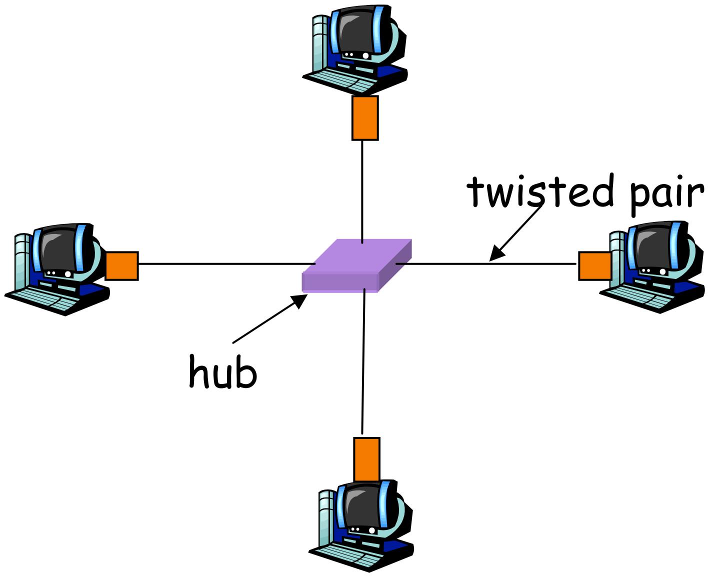


Unreliable, Connectionless Service

- Connectionless
  - No handshaking between sending and receiving adapter.
- Unreliable
  - Receiving adapter doesn’t send ACKs or NACKs
  - Packets passed to network layer can have gaps
  - Gaps will be filled if application is using TCP
  - Otherwise, the application will see the gaps

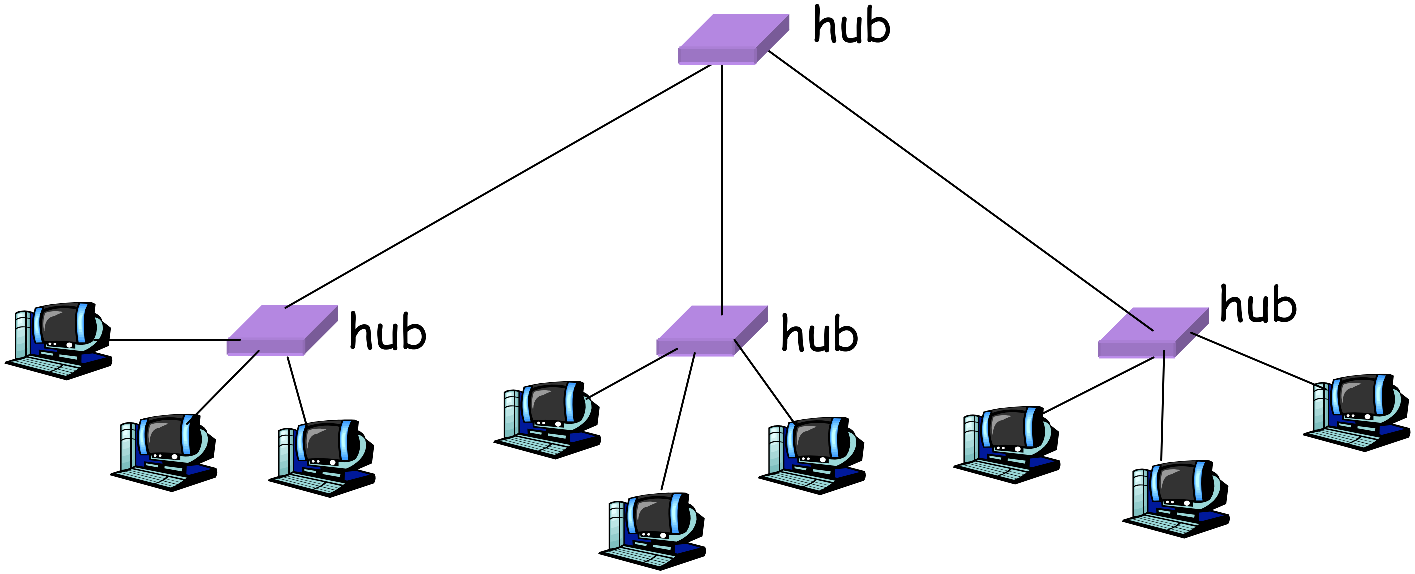
Hubs: Physical-Layer Repeaters

- Hubs are physical-layer repeaters
  - Bits coming from one link go out all other links
  - At the same rate, with no frame buffering
  - No CSMA/CD at hub: adapters detect collisions



Interconnecting with Hubs

- Backbone hub interconnects LAN segments
- All packets seen everywhere, forming one large collision domain
- Can’t interconnect Ethernets of different speeds

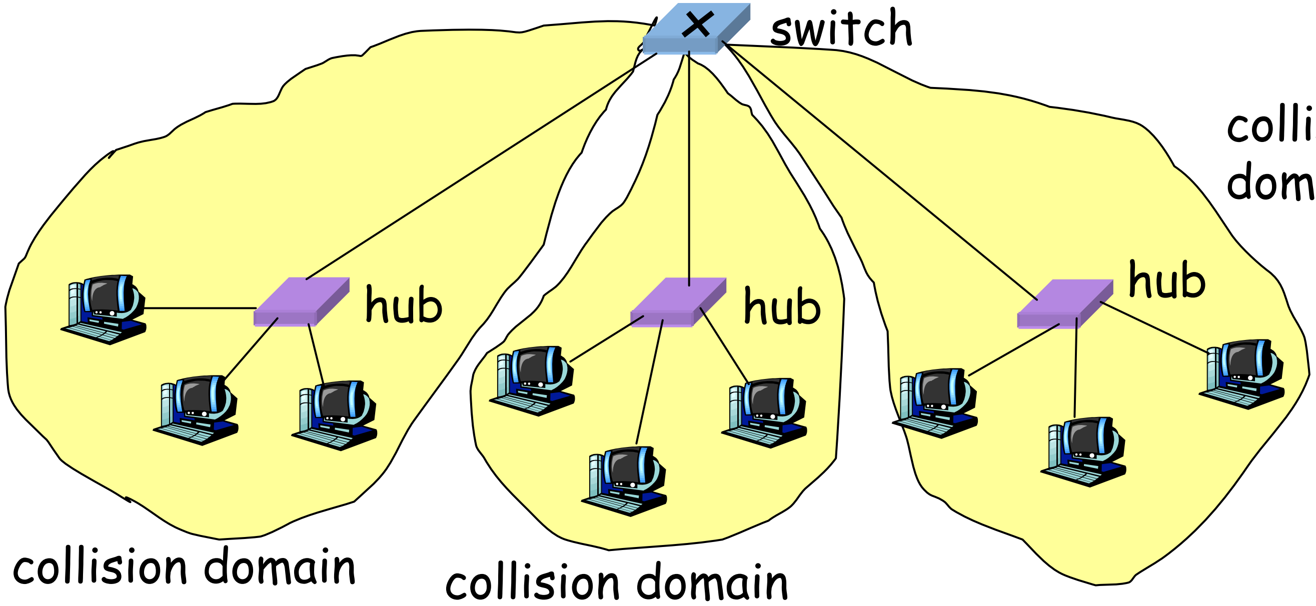


Switch

- Link layer device
  - Stores and forwards Ethernet frames
  - Examines frame header and selectively forwards frame based on MAC dest address
  - When frame is to be forwarded on segment, uses CSMA/CD to access segment
- Transparent
  - Hosts are unaware of presence of switches
- Plug-and-play, self-learning
  - Switches do not need to be configured

Switch: Traffic Isolation

- Switch breaks subnet into LAN segments
- Switch filters packets
  - Same-LAN-segment frames not usually forwarded onto other LAN segments
  - Segments become separate collision domains



Benefits of Ethernet

- Easy to administer and maintain
- Inexpensive
- Increasingly higher speed

- Moved from shared media to switches
  - Change everything except the frame format
  - A good general lesson for evolving the Internet

Conclusions

- IP runs on a variety of link layer technologies
  - Point-to-point links vs. shared media
  - Wide varieties within each class
- Link layer performs key services
  - Encoding, framing, and error detection
  - Optionally error correction and flow control
- Shared media introduce interesting challenges
  - Decentralized control over resource sharing
  - Partitioned channel, taking turns, and random access
  - Ethernet as a wildly popular example
- Next time: switches and bridges

Circuit vs. Packet Switching

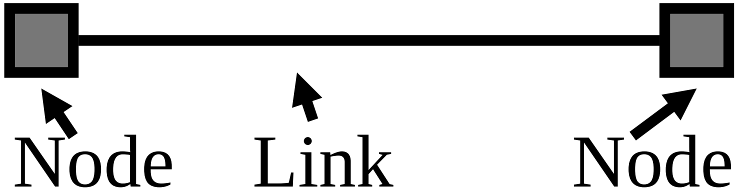
Introduction to Data Networks

2008.3

Goals of Today’s Lecture

- Connectivity
  - Links and nodes
  - Circuit switching
  - Packet switching
- IP service model
  - Best-effort packet delivery
  - IP as the Internet’s “narrow waist”
  - Design philosophy of IP
- IP packet structure
  - Fields in the IP header
  - Traceroute using TTL field
  - Source-address spoofing

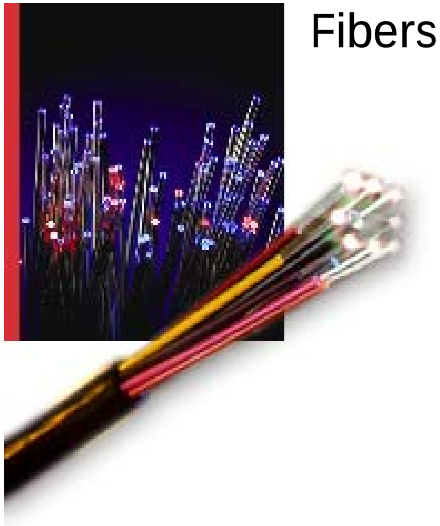
Simple Network: Nodes and a Link



- Node: computer
  - End host: general-purpose computer, cell phone, PDA
  - Network node: switch or router
- Link: physical medium connecting nodes
  - Twisted pair: the wire that connects to telephones
  - Coaxial cable: the wire that connects to TV sets
  - Optical fiber: high-bandwidth long-distance links
  - Space: propagation of radio waves, microwaves, …

Network Components

Links

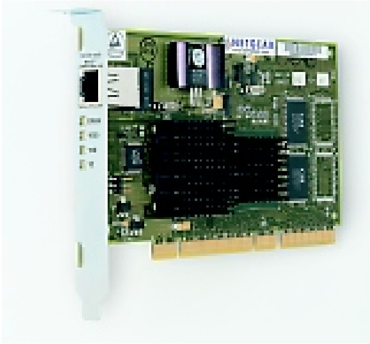


Fibers

Coaxial Cable

Interfaces

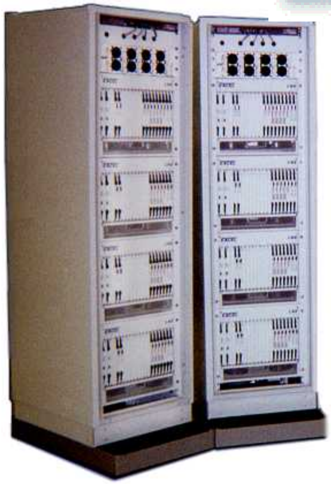
Ethernet card



Wireless card

Switches/routers

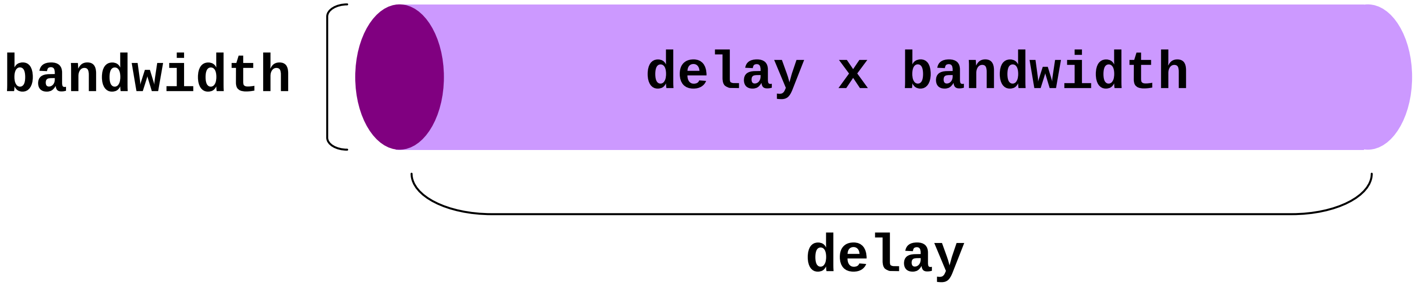
Large router



Telephone switch

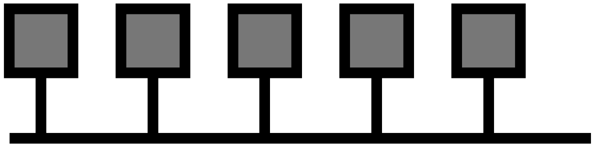
Links: Delay and Bandwidth

- Delay
  - –Latency for propagating data along the link
  - –Corresponds to the “length” of the link
  - –Typically measured in seconds
- Bandwidth
  - –Amount of data sent (or received) per unit time
  - –Corresponds to the “width” of the link
  - –Typically measured in bits per second

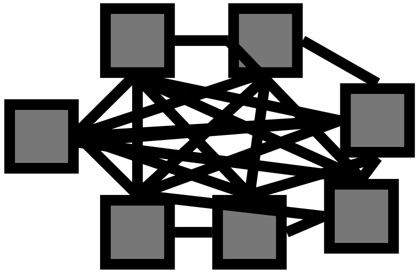


Connecting More Than Two Hosts

- Multi-access link: Ethernet, wireless
  - Single physical link, shared by multiple nodes
  - Limitations on distance and number of nodes
- Point-to-point links: fiber-optic cable
  - Only two nodes (separate link per pair of nodes)
  - Limitations on the number of adapters per node

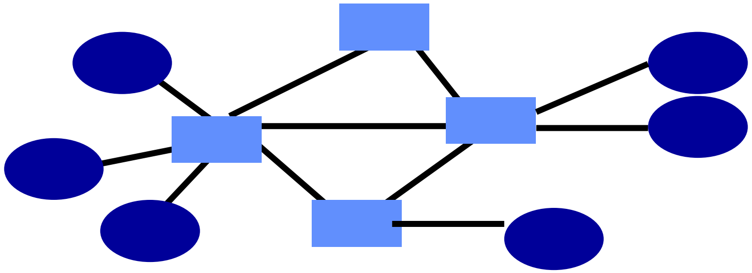


multi-access link



point-to-point links

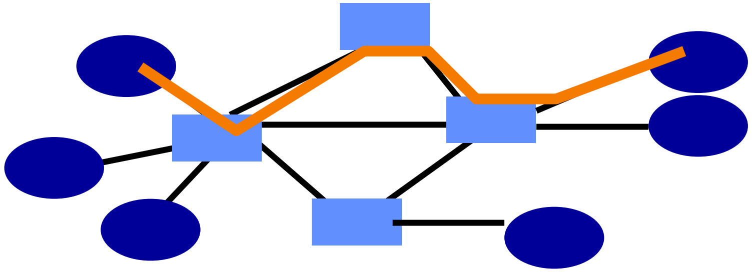
Beyond Directly-Connected Networks



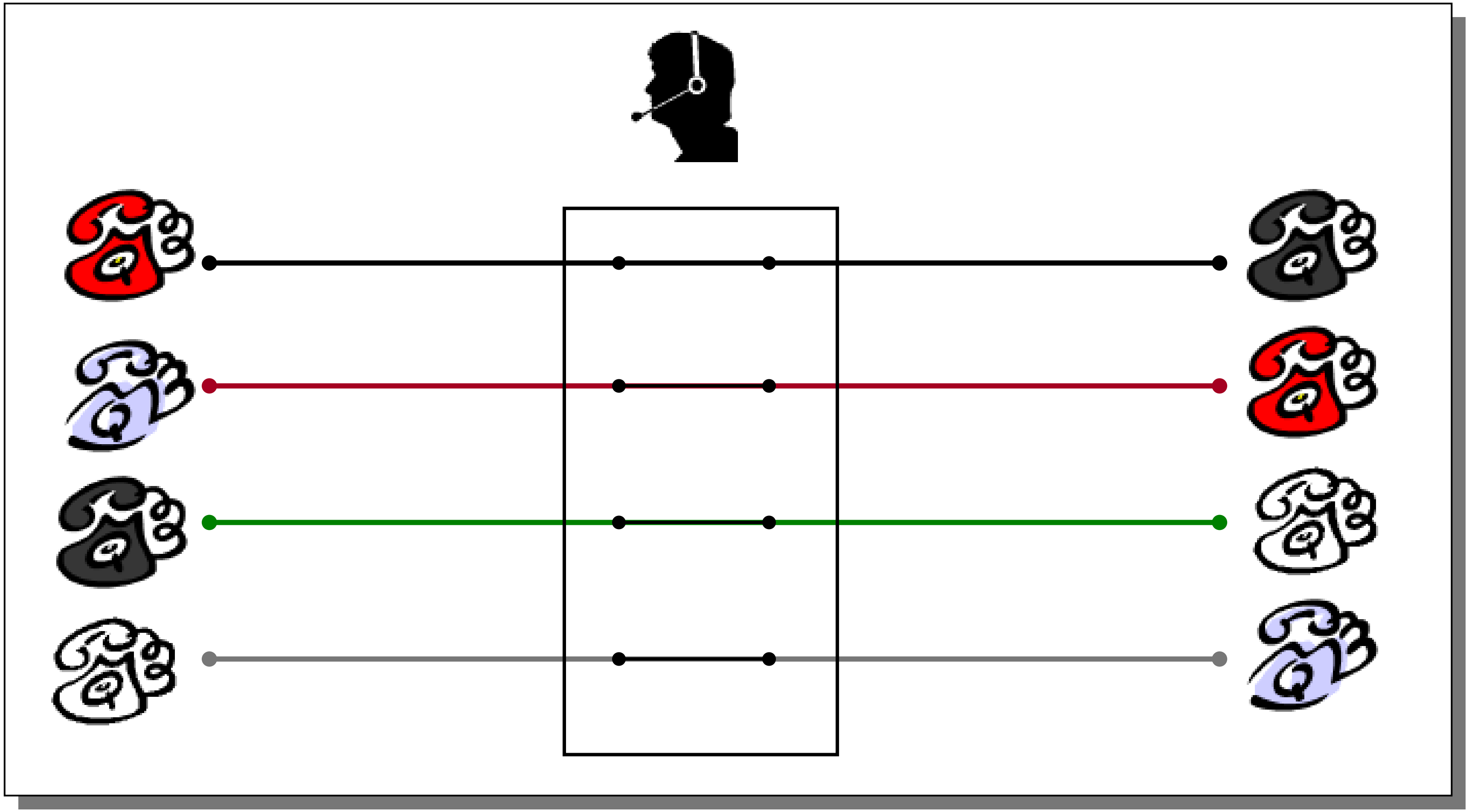
- Switched network
  - –End hosts at the edge
  - –Network nodes that switch traffic
  - –Links between the nodes
- Multiplexing
  - –Many end hosts communicate over the network
  - –Traffic shares access to the same links

Circuit Switching (e.g., Phone Network)

- Source establishes connection to destination
  - Node along the path store connection info
  - Nodes may reserve resources for the connection
- Source sends data over the connection
  - No destination address, since nodes know path
- Source tears down connection when done

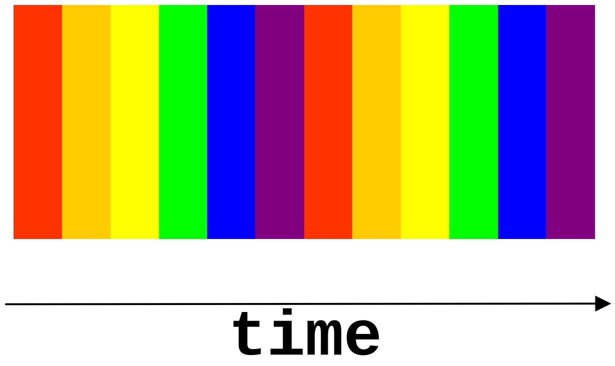


Circuit Switching With Human Operator

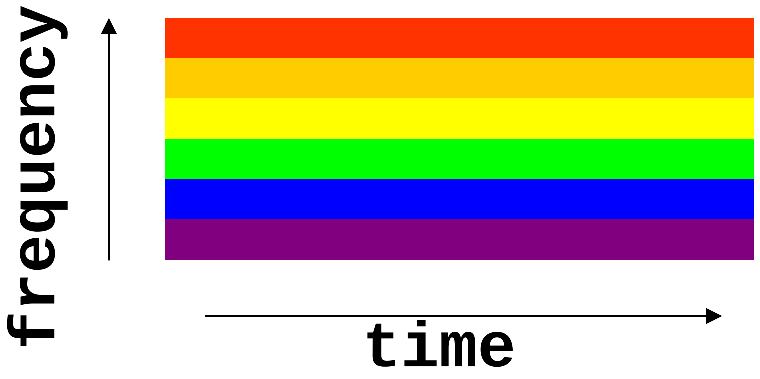


Circuit Switching: Multiplexing a Link

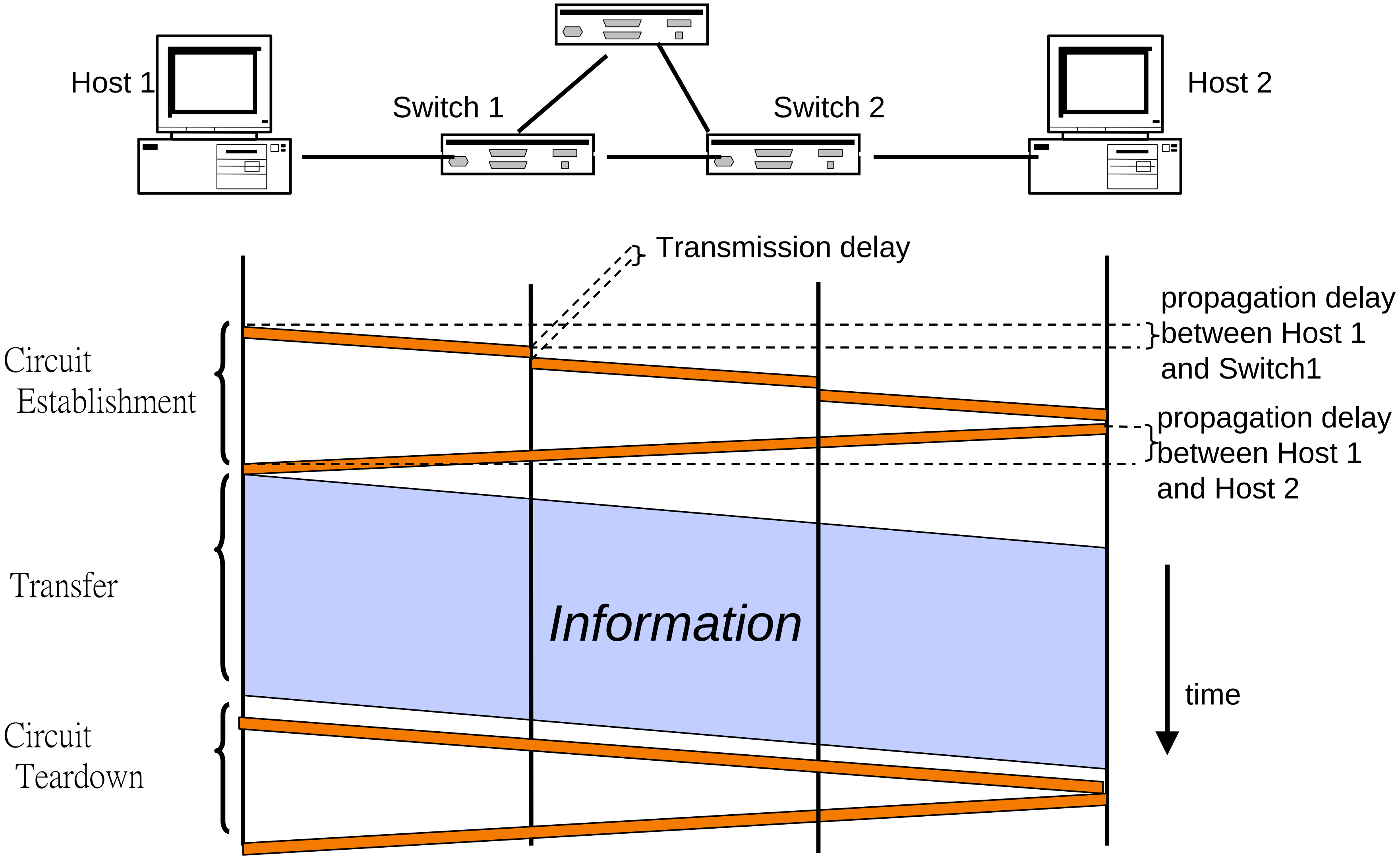
- Time-division
  - Each circuit allocated certain time slots



- Frequency-division
  - Each circuit allocated certain frequencies



Timing in Circuit Switching



Advantages of Circuit Switching

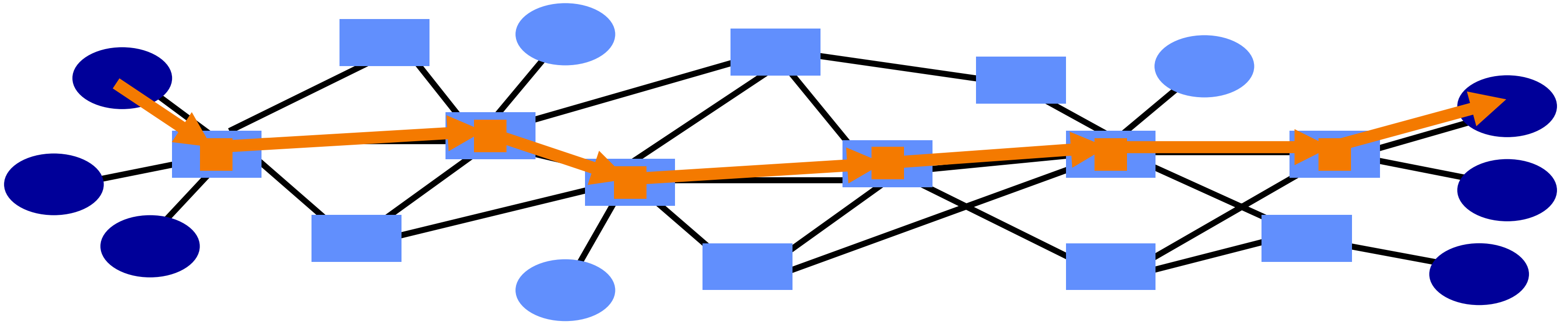
- Guaranteed bandwidth
  - Predictable communication performance
  - Not “best-effort” delivery with no real guarantees
- Simple abstraction
  - Reliable communication channel between hosts
  - No worries about lost or out-of-order packets
- Simple forwarding
  - Forwarding based on time slot or frequency
  - No need to inspect a packet header
- Low per-packet overhead
  - Forwarding based on time slot or frequency
  - No IP (and TCP/UDP) header on each packet

Disadvantages of Circuit Switching

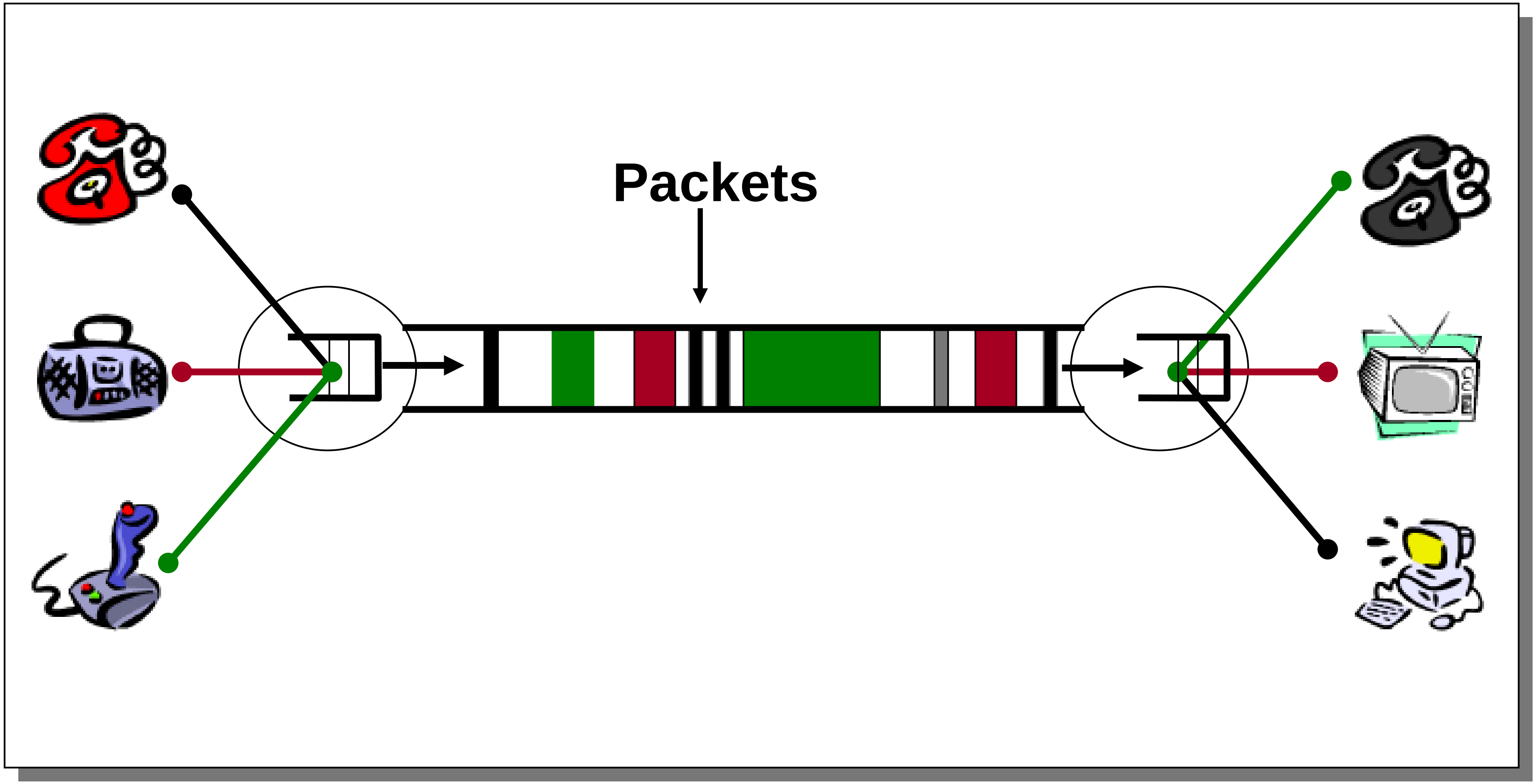
- Wasted bandwidth
  - Bursty traffic leads to idle connection during silent period
  - Unable to achieve gains from statistical multiplexing
- Blocked connections
  - Connection refused when resources are not sufficient
  - Unable to offer “okay” service to everybody
- Connection set-up delay
  - No communication until the connection is set up
  - Unable to avoid extra latency for small data transfers
- Network state
  - Network nodes must store per-connection information
  - Unable to avoid per-connection storage and state

Packet Switching (e.g., Internet)

- Data traffic divided into packets
  - –Each packet contains a header (with address)
- Packets travel separately through network
  - –Packet forwarding based on the header
  - –Network nodes may store packets temporarily
- Destination reconstructs the message

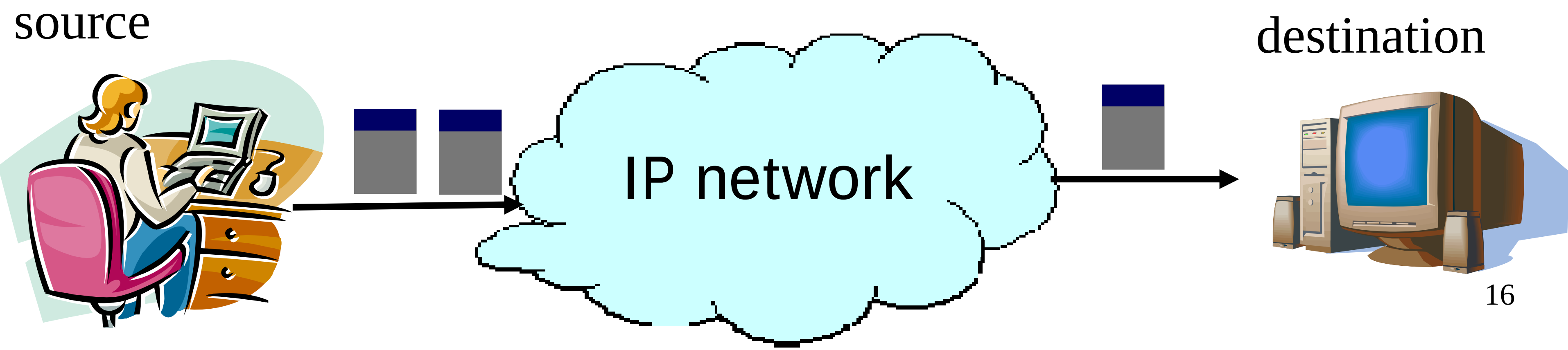


Packet Switching: Statistical Multiplexing



IP Service: Best-Effort Packet Delivery

- Packet switching
  - –Divide messages into a sequence of packets
  - –Headers with source and destination address
- Best-effort delivery
  - –Packets may be lost
  - –Packets may be corrupted
  - –Packets may be delivered out of order



IP Service Model: Why Packets?

- Data traffic is bursty
  - Logging in to remote machines
  - Exchanging e-mail messages
- Don’t want to waste reserved bandwidth
  - No traffic exchanged during idle periods
- Better to allow multiplexing
  - Different transfers share access to same links
- Packets can be delivered by most anything
  - RFC 2549: IP over Avian Carriers (aka birds)
- … still, packet switching can be inefficient
  - Extra header bits on every packet

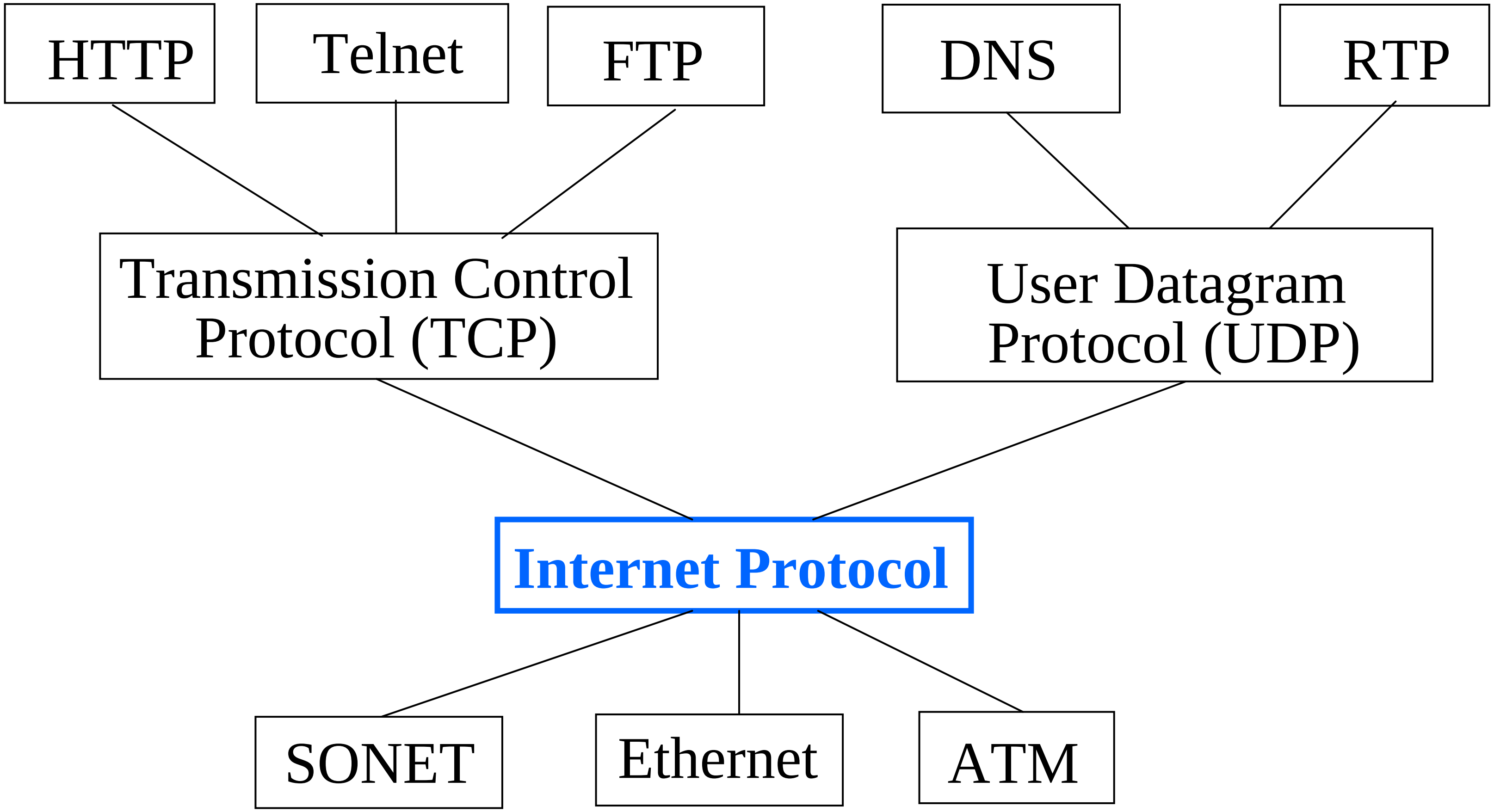
IP Service Model: Why Best-Effort?

- IP means never having to say you're sorry…
  - Don't need to reserve bandwidth and memory
  - Don't need to do error detection & correction
  - Don't need to remember from one packet to next
- Easier to survive failures
  - Transient disruptions are okay during failover
- … but, applications *do* want efficient, accurate transfer of data in order, in a timely fashion

IP Service: Best-Effort is Enough

- No error detection or correction
  - Higher-level protocol can provide error checking
- Successive packets may not follow the same path
  - Not a problem as long as packets reach the destination
- Packets can be delivered out-of-order
  - Receiver can put packets back in order (if necessary)
- Packets may be lost or arbitrarily delayed
  - Sender can send the packets again (if desired)
- No network congestion control (beyond “drop”)
  - Sender can slow down in response to loss or delay

Layering in the IP Protocols



HTTP

Telnet

FTP

DNS

RTP

Transmission Control Protocol (TCP)

User Datagram Protocol (UDP)

Internet Protocol

SONET

Ethernet

ATM

History: Why IP Packets?

- IP proposed in the early 1970s
  - Defense Advanced Research Project Agency (DARPA)
- Goal: connect existing networks
  - To develop an effective technique for multiplexed utilization of existing interconnected networks
  - E.g., connect packet radio networks to the ARPAnet
- Motivating applications
  - Remote login to server machines
  - Inherently bursty traffic with long silent periods
- Prior ARPAnet experience with packet switching
  - Previous DARPA project
  - Demonstrated store-and-forward packet switching

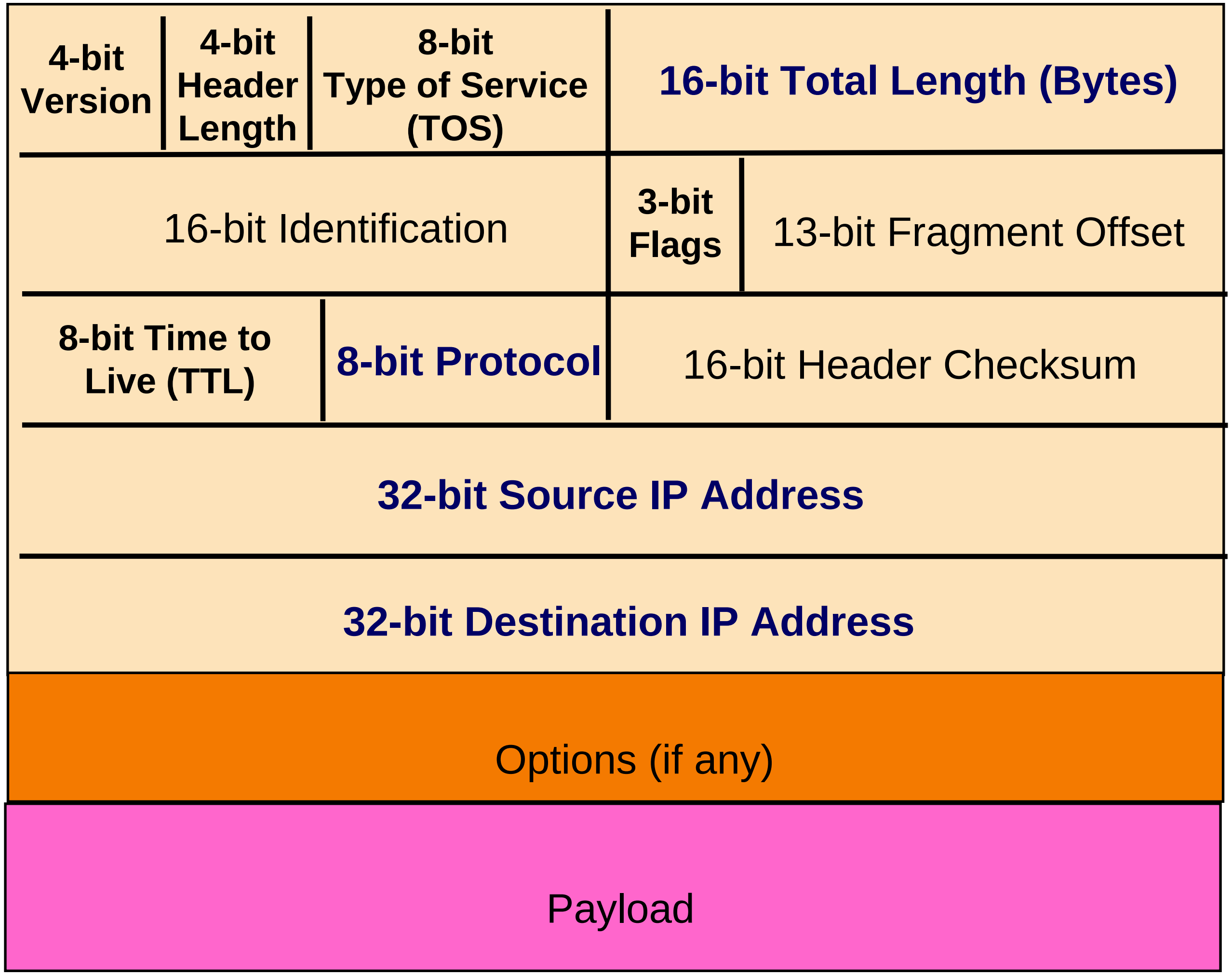
Other Main Driving Goals (In Order)

- Communication should continue despite failures
  - Survive equipment failure or physical attack
  - Traffic between two hosts continue on another path
- Support multiple types of communication services
  - Differing requirements for speed, latency, & reliability
  - Bidirectional reliable delivery vs. message service
- Accommodate a variety of networks
  - Both military and commercial facilities
  - Minimize assumptions about the underlying network

Other Driving Goals, Somewhat Met

- Permit distributed management of resources
  - Nodes managed by different institutions
  - … though this is still rather challenging
- Cost-effectiveness
  - Statistical multiplexing through packet switching
  - … though packet headers and retransmissions wasteful
- Ease of attaching new hosts
  - Standard implementations of end-host protocols
  - … though still need a fair amount of end-host software
- Accountability for use of resources
  - Monitoring functions in the nodes
  - … though this is still fairly limited and immature

IP Packet Structure



4-bit Version

4-bit Header Length

8-bit Type of Service (TOS)

16-bit Total Length (Bytes)

16-bit Identification

3-bit Flags

13-bit Fragment Offset

8-bit Time to Live (TTL)

8-bit Protocol

16-bit Header Checksum

32-bit Source IP Address

32-bit Destination IP Address

Options (if any)

Payload

IP Packet Header Fields

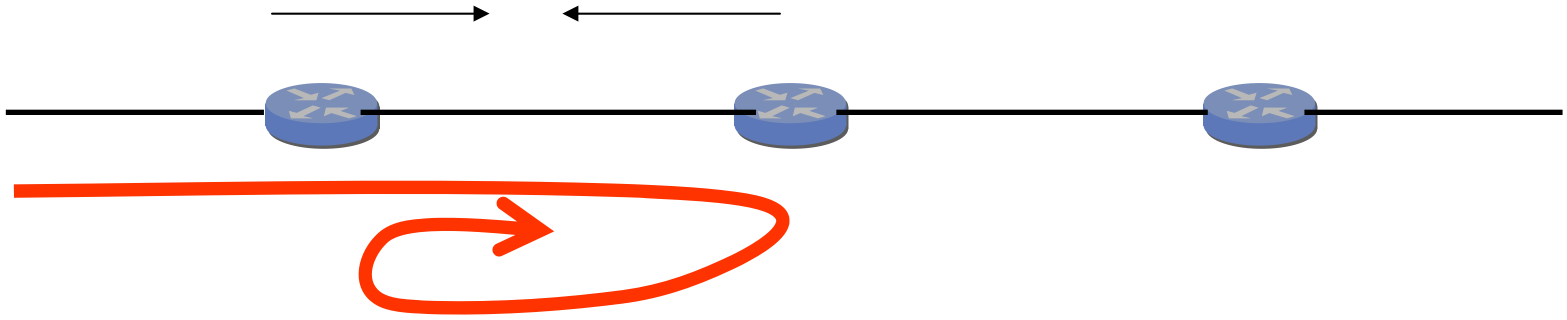
- Version number (4 bits)
  - Indicates the version of the IP protocol
  - Necessary to know what other fields to expect
  - Typically “4” (for IPv4), and sometimes “6” (for IPv6)
- Header length (4 bits)
  - Number of 32-bit words in the header
  - Typically “5” (for a 20-byte IPv4 header)
  - Can be more when “IP options” are used
- Type-of-Service (8 bits)
  - Allow packets to be treated differently based on needs
  - E.g., low delay for audio, high bandwidth for bulk transfer

IP Packet Header Fields (Continued)

- Total length (16 bits)
  - Number of bytes in the packet
  - Maximum size is 63,535 bytes (2^{16} -1)
  - … though underlying links may impose harder limits
- Fragmentation information (32 bits)
  - Packet identifier, flags, and fragment offset
  - Supports dividing a large IP packet into fragments
  - … in case a link cannot handle a large IP packet
- Time-To-Live (8 bits)
  - Used to identify packets stuck in forwarding loops
  - … and eventually discard them from the network

Time-to-Live (TTL) Field

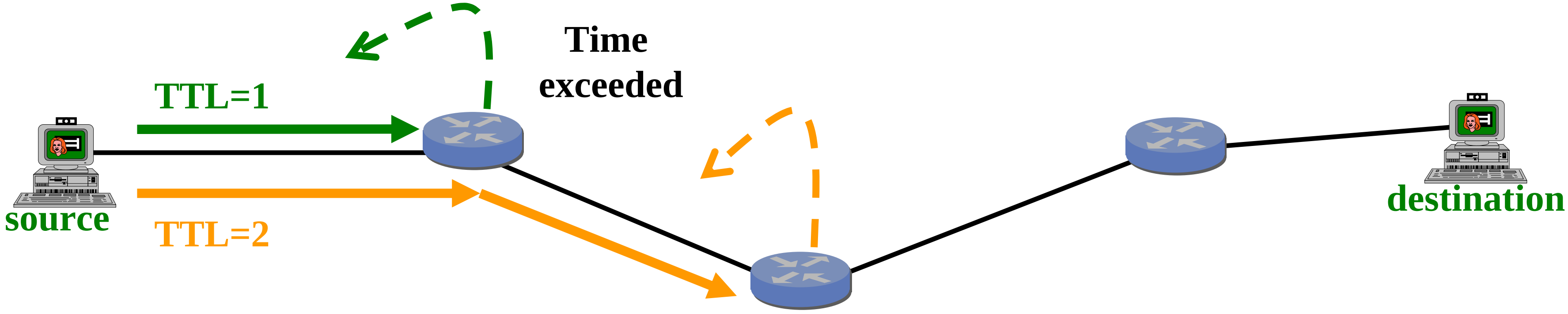
- Potential robustness problem
  - Forwarding loops can cause packets to cycle forever
  - Confusing if the packet arrives much later



- Time-to-live field in packet header
  - TTL field decremented by each router on the path
  - Packet is discarded when TTL field reaches 0…
  - …and “time exceeded” message is sent to the source

Application of TTL in Traceroute

- Time-To-Live field in IP packet header
  - Source sends a packet with a TTL of *n*
  - Each router along the path decrements the TTL
  - "TTL exceeded" sent when TTL reaches *0*
- Traceroute tool exploits this TTL behavior



Send packets with TTL=1, 2, … and record source of "time exceeded" message

Example Traceroute: Berkeley to CNN

Hop number, IP address, DNS name

```
 1  169.229.62.1       inr-daedalus-0.CS.Berkeley.EDU
 2  169.229.59.225     soda-cr-1-1-soda-br-6-2
 3  128.32.255.169     vlan242.inr-202-doecev.Berkeley.EDU
 4  128.32.0.249       gigE6-0-0.inr-666-doecev.Berkeley.EDU
 5  128.32.0.66        qsv-juniper--ucb-gw.calren2.net
 6  209.247.159.109    POS1-0.hsipaccess1.SanJose1.Level3.net
 7  *                  ?
 8  64.159.1.46        ?
 9  209.247.9.170      pos8-0.hsa2.Atlanta2.Level3.net
10  66.185.138.33      pop2-atm-P0-2.atdn.net
11  *                  ?
12  66.185.136.17      pop1-atl-P4-0.atdn.net
13  64.236.16.52       www4.cnn.com
```

No response from router

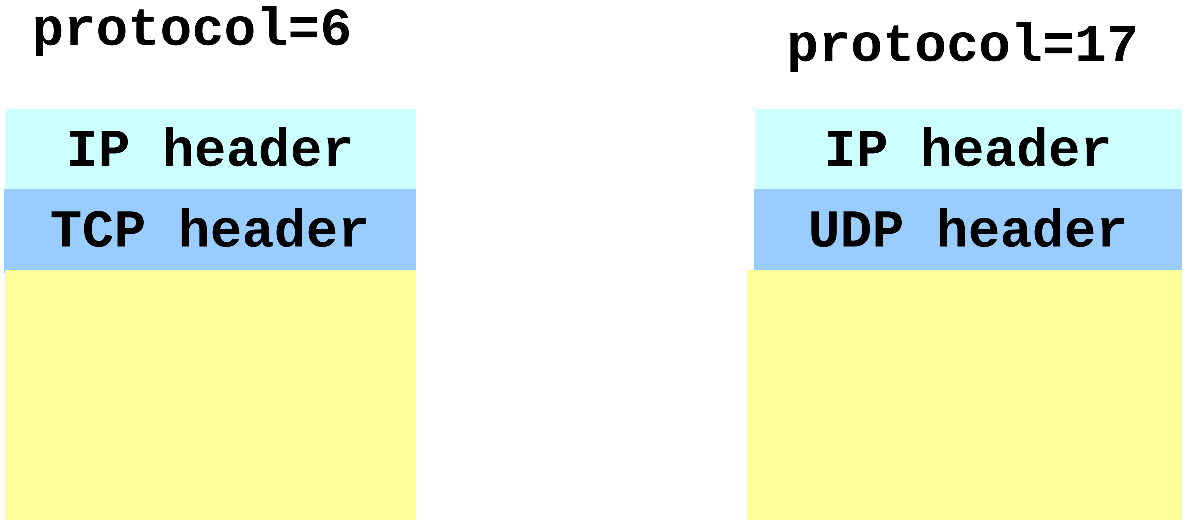
No name resolution

Try Running Traceroute Yourself

- On UNIX machine
  - Traceroute
  - E.g., “traceroute www.cnn.com” or “traceroute 12.1.1.1”
- On Windows machine
  - Tracert
  - E.g., “tracert www.cnn.com” or “tracert 12.1.1.1”
- Common uses of traceroute
  - Discover the topology of the Internet
  - Debug performance and reachability problems

IP Packet Header Fields (Continued)

- Protocol (8 bits)
  - Identifies the higher-level protocol
    - E.g., “6” for the Transmission Control Protocol (TCP)
    - E.g., “17” for the User Datagram Protocol (UDP)
  - Important for demultiplexing at receiving host
    - Indicates what kind of header to expect next



protocol=6

IP header

TCP header

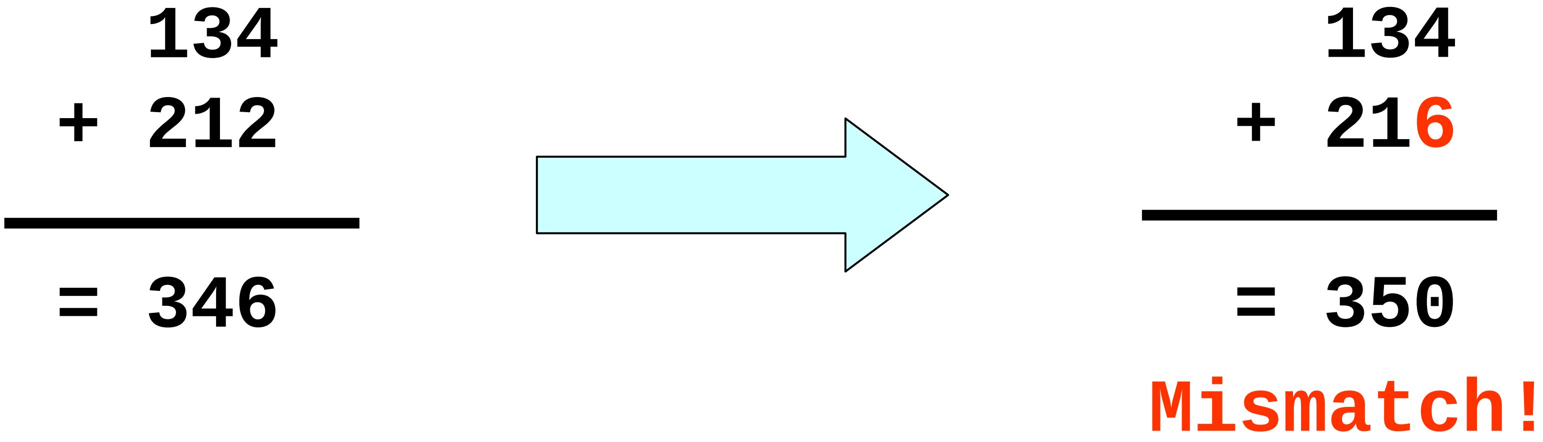
protocol=17

IP header

UDP header

IP Packet Header Fields (Continued)

- Checksum (16 bits)
  - Sum of all 16-bit words in the IP packet header
  - If any bits of the header are corrupted in transit
  - … the checksum won't match at receiving host
  - Receiving host discards corrupted packets
    - Sending host will retransmit the packet, if needed



```
  134
+ 212
-----
= 346
```

```
  134
+ 216
-----
= 350
```

Mismatch!

IP Packet Header (Continued)

- Two IP addresses
  - Source IP address (32 bits)
  - Destination IP address (32 bits)
- Destination address
  - Unique identifier for the receiving host
  - Allows each node to make forwarding decisions
- Source address
  - Unique identifier for the sending host
  - Recipient can decide whether to accept packet
  - Enables recipient to send a reply back to source

What if the Source Lies?

- Source address should be the sending host
  - But, who’s checking, anyway?
  - You could send packets with any source you want
- Why would someone want to do this?
  - Launch a denial-of-service attack
    - Send excessive packets to the destination
    - … to overload the node, or the links leading to the node
  - Evade detection by “spoofing”
    - But, the victim could identify you by the source address
    - So, you can put someone else’s source address in the packets
  - Also, an attack against the spoofed host
    - Spoofed host is wrongly blamed
    - Spoofed host may receive return traffic from the receiver

Summary: Packet Switching Review

- Efficient
  - Can send from any input that is ready
- General
  - Multiple types of applications
- Accommodates bursty traffic
  - Addition of queues
- Store and forward
  - Packets are self contained units
  - Can use alternate paths – reordering
- Contention (i.e., no isolation)
  - Congestion
  - Delay

Switches

Reading: Section 3.2

COS 461: Computer Networks

Jennifer Rexford

Switches

Introduction to Data Networks

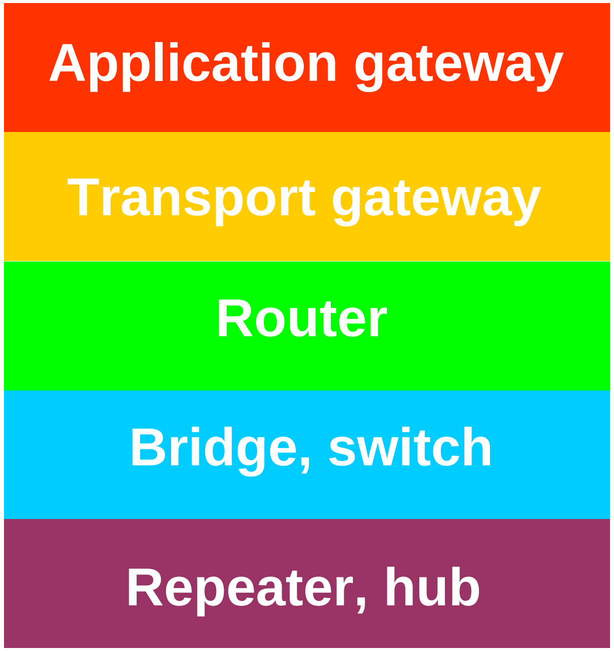
2008.3

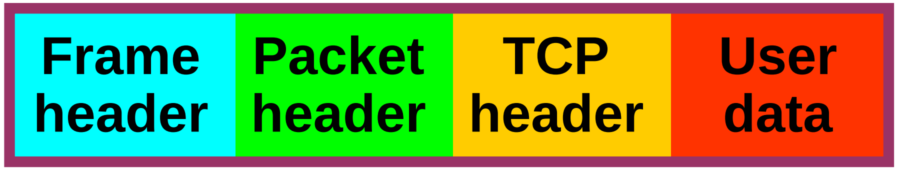
Goals of Today’s Lecture

- Devices that shuttling packets at different layers
  - Repeaters and hubs
  - Bridges and switches
  - Routers
- Switch protocols and mechanisms
  - Dedicated access and full-duplex transfers
  - Cut-through switching
  - Self learning of the switch table
  - Spanning trees
  - Virtual LANs (VLANs)

Shuttling Data at Different Layers

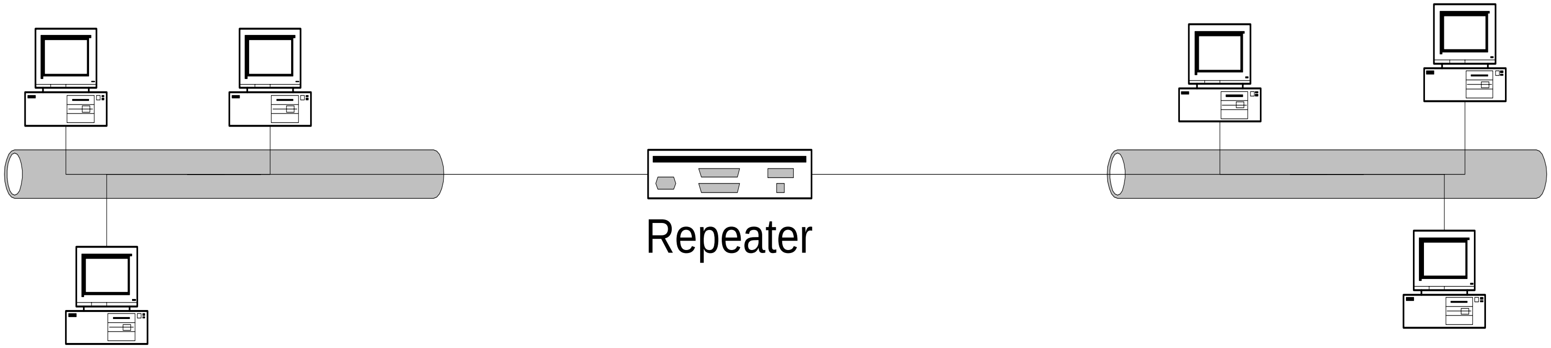
- Different devices switch different things
  - Physical layer: electrical signals (repeaters and hubs)
  - Link layer: frames (bridges and switches)
  - Network layer: packets (routers)





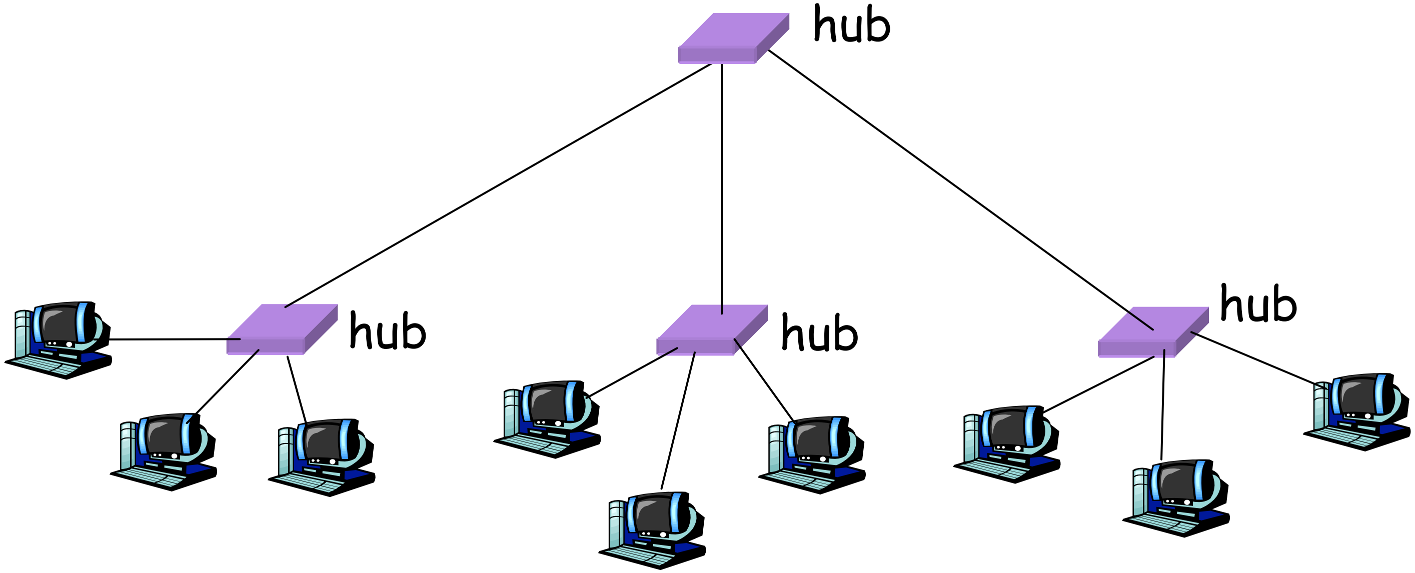
Physical Layer: Repeaters

- Distance limitation in local-area networks
  - Electrical signal becomes weaker as it travels
  - Imposes a limit on the length of a LAN
- Repeaters join LANs together
  - Analog electronic device
  - Continuously monitors electrical signals on each LAN
  - Transmits an amplified copy



Physical Layer: Hubs

- Joins multiple input lines electrically
  - Designed to hold multiple line cards
  - Do not necessarily amplify the signal
- Very similar to repeaters
  - Also operates at the physical layer



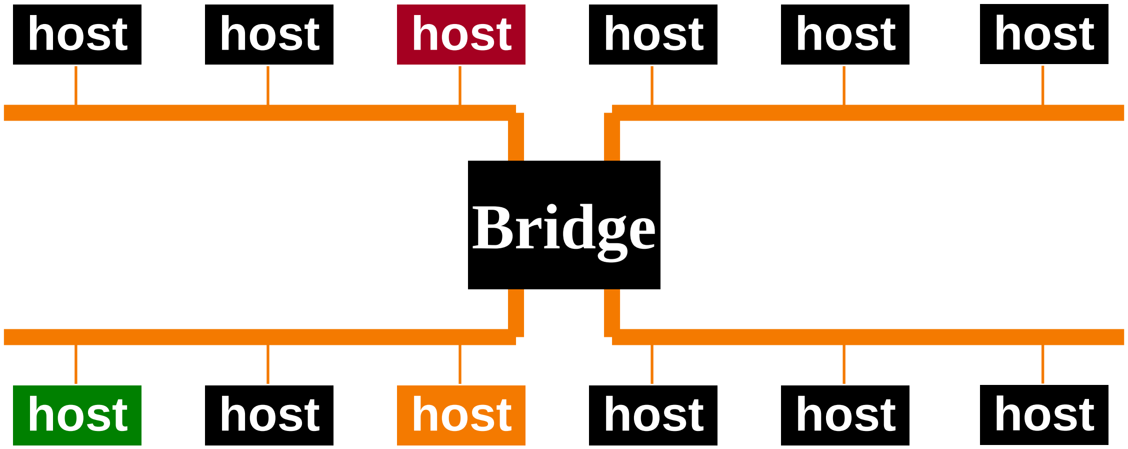
Limitations of Repeaters and Hubs

- One large collision domain
  - Every bit is sent everywhere
  - So, aggregate throughput is limited
  - E.g., three departments each get 10 Mbps independently

    … and then connect via a hub and must share 10 Mbps
- Cannot support multiple LAN technologies
  - Does not buffer or interpret frames
  - So, can’t interconnect between different rates or formats
  - E.g., 10 Mbps Ethernet and 100 Mbps Ethernet
- Limitations on maximum nodes and distances
  - Does not circumvent the limitations of shared media
  - E.g., still cannot go beyond 2500 meters on Ethernet

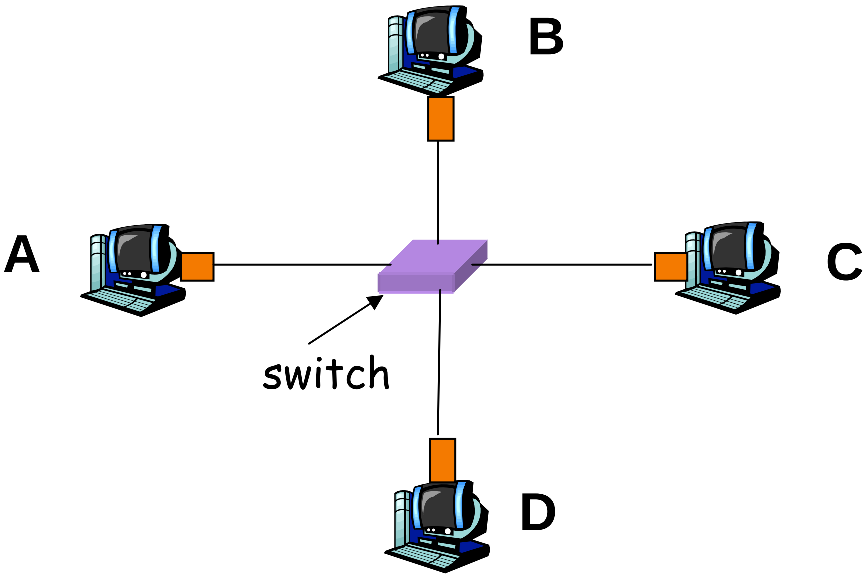
Link Layer: Bridges

- Connects two or more LANs at the link layer
  - Extracts destination address from the frame
  - Looks up the destination in a table
  - Forwards the frame to the appropriate LAN segment
- Each segment is its own collision domain



Link Layer: Switches

- Typically connects individual computers
  - A switch is essentially the same as a bridge
  - … though typically used to connect hosts, not LANs
- Like bridges, support concurrent communication
  - Host A can talk to C, while B talks to D

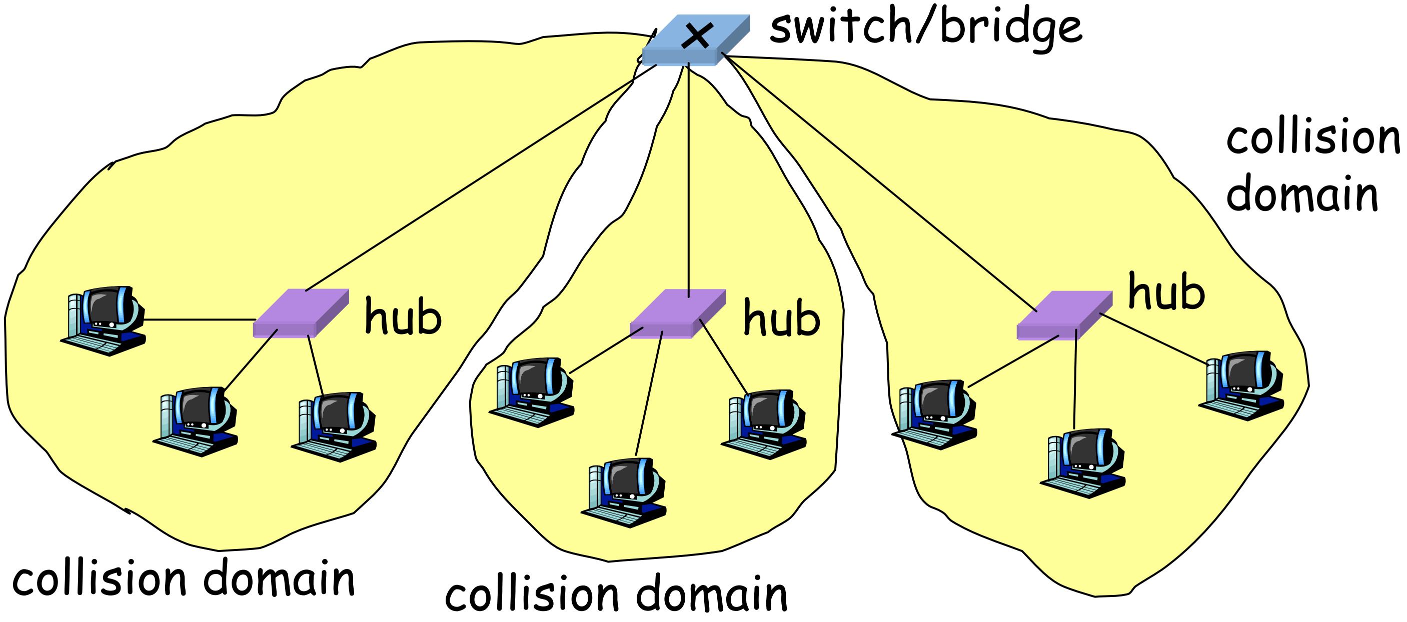


Dedicated Access and Full Duplex

- Dedicated access
  - Host has direct connection to the switch
  - … rather than a shared LAN connection
- Full duplex
  - Each connection can send in both directions
  - Host sending to switch, and host receiving from switch
  - E.g., in 10BaseT and 100Base T
- Completely avoids collisions
  - Each connection is a bidirectional point-to-point link
  - No need for carrier sense, collision detection, and so on

Bridges/Switches: Traffic Isolation

- Switch breaks subnet into LAN segments
- Switch filters packets
  - Frame only forwarded to the necessary segments
  - Segments become separate collision domains



Advantages Over Hubs/Repeaters

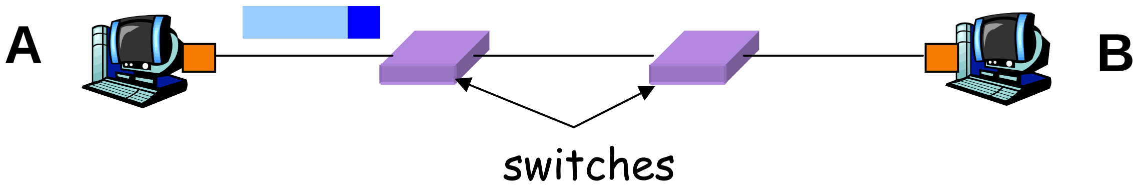
- Only forwards frames as needed
  - Filters frames to avoid unnecessary load on segments
  - Sends frames only to segments that need to see them
- Extends the geographic span of the network
  - Separate collision domains allow longer distances
- Improves privacy by limiting scope of frames
  - Hosts can “snoop” the traffic traversing their segment
  - … but not all the rest of the traffic
- Applies carrier sense and collision detection
  - Does not transmit when the link is busy
  - Applies exponential back-off after a collision
- Joins segments using different technologies

Disadvantages Over Hubs/Repeaters

- Delay in forwarding frames
  - Bridge/switch must receive and parse the frame
  - … and perform a look-up to decide where to forward
  - Storing and forwarding the packet introduces delay
  - Solution: cut-through switching
- Need to learn where to forward frames
  - Bridge/switch needs to construct a forwarding table
  - Ideally, without intervention from network administrators
  - Solution: self-learning
- Higher cost
  - More complicated devices that cost more money

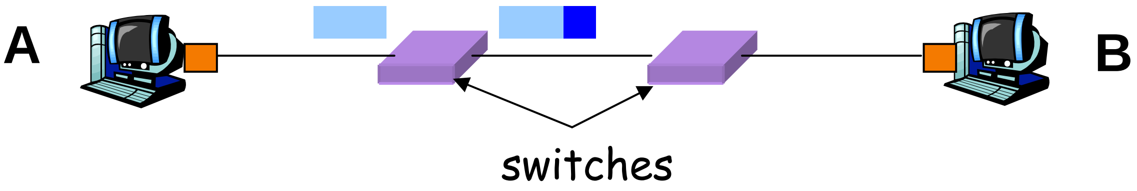
Motivation For Cut-Through Switching

- Buffering a frame takes time
  - Suppose L is the length of the frame
  - And R is the transmission rate of the links
  - Then, receiving the frame takes L/R time units
- Buffering delay can be a high fraction of total delay
  - Propagation delay is small over short distances
  - Making buffering delay a large fraction of total
  - Analogy: large group walking through NYC



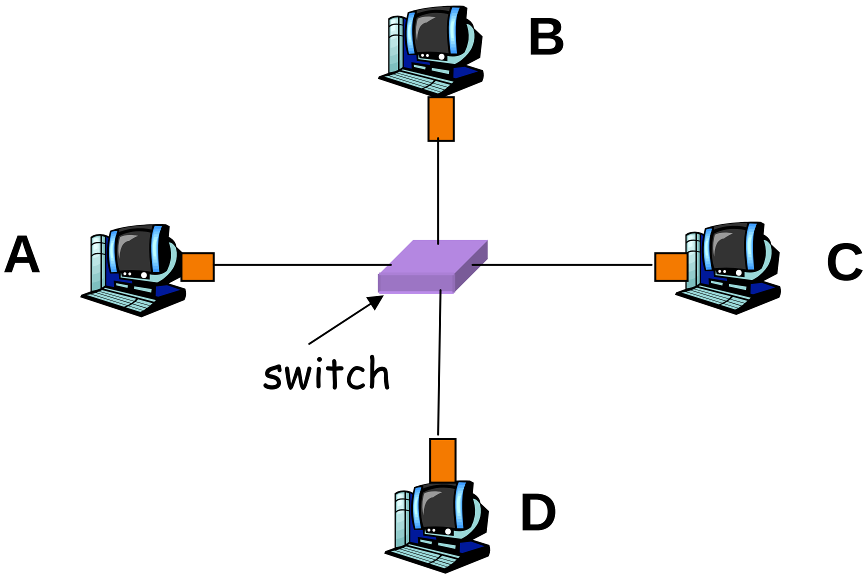
Cut-Through Switching

- Start transmitting as soon as possible
  - Inspect the frame header and do the look-up
  - If outgoing link is idle, start forwarding the frame
- Overlapping transmissions
  - Transmit the head of the packet via the outgoing link
  - … while still receiving the tail via the incoming link
  - Analogy: different folks crossing different intersections



Motivation For Self Learning

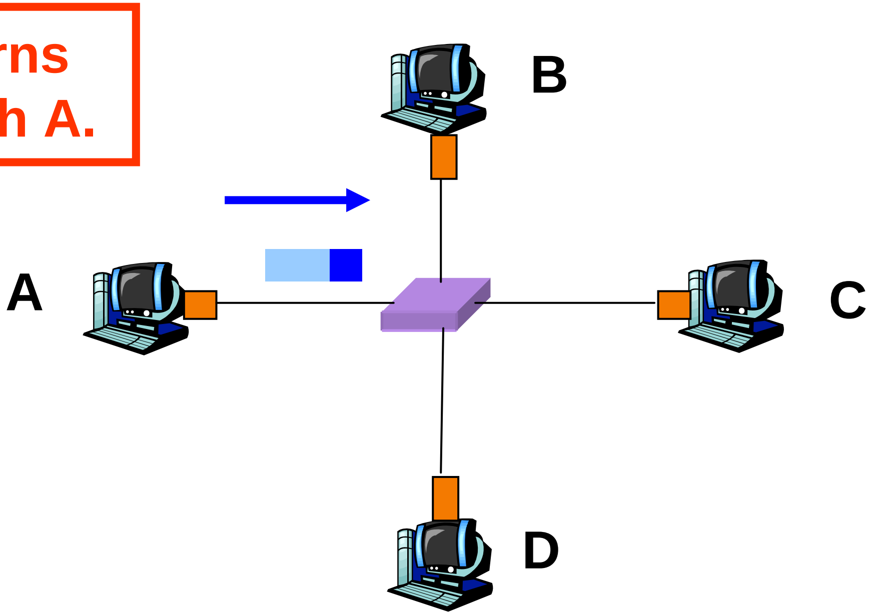
- Switches forward frames selectively
  - Forward frames only on segments that need them
- Switch table
  - Maps destination MAC address to outgoing interface
  - Goal: construct the switch table automatically



Self Learning: Building the Table

- When a frame arrives
  - Inspect the *source* MAC address
  - Associate the address with the *incoming* interface
  - Store the mapping in the switch table
  - Use a time-to-live field to eventually forget the mapping

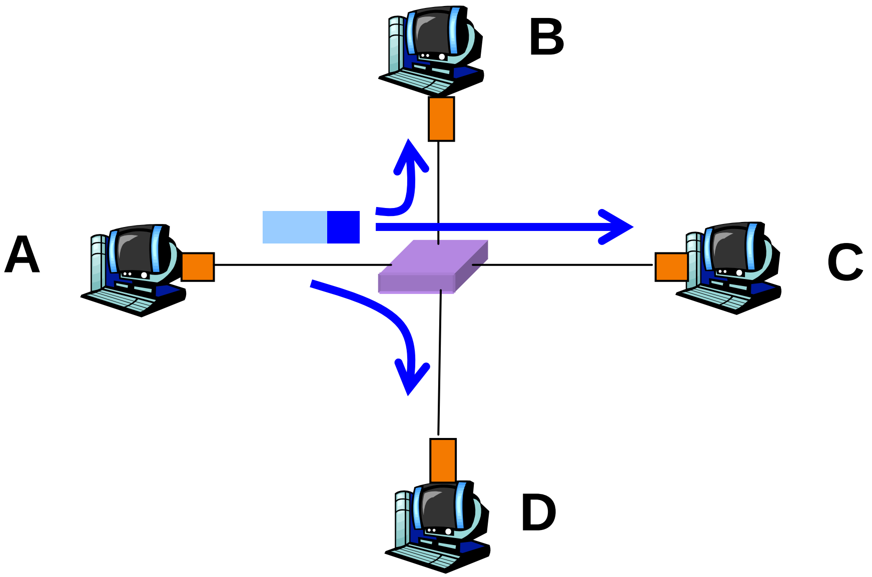
Switch learns how to reach A.



Self Learning: Handling Misses

- When frame arrives with unfamiliar destination
  - Forward the frame out all of the interfaces
  - … except for the one where the frame arrived
  - Hopefully, this case won't happen very often

When in doubt, shout!



Switch Filtering/Forwarding

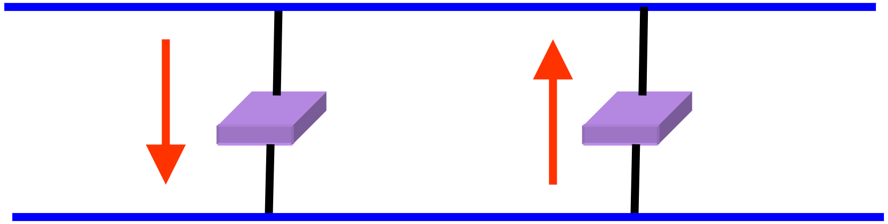
When switch receives a frame:

```
index switch table using MAC dest address
if entry found for destination
  then{
    if dest on segment from which frame arrived
        then drop the frame
         else forward the frame on interface indicated
      }
   else flood
```

forward on all but the interface on which the frame arrived

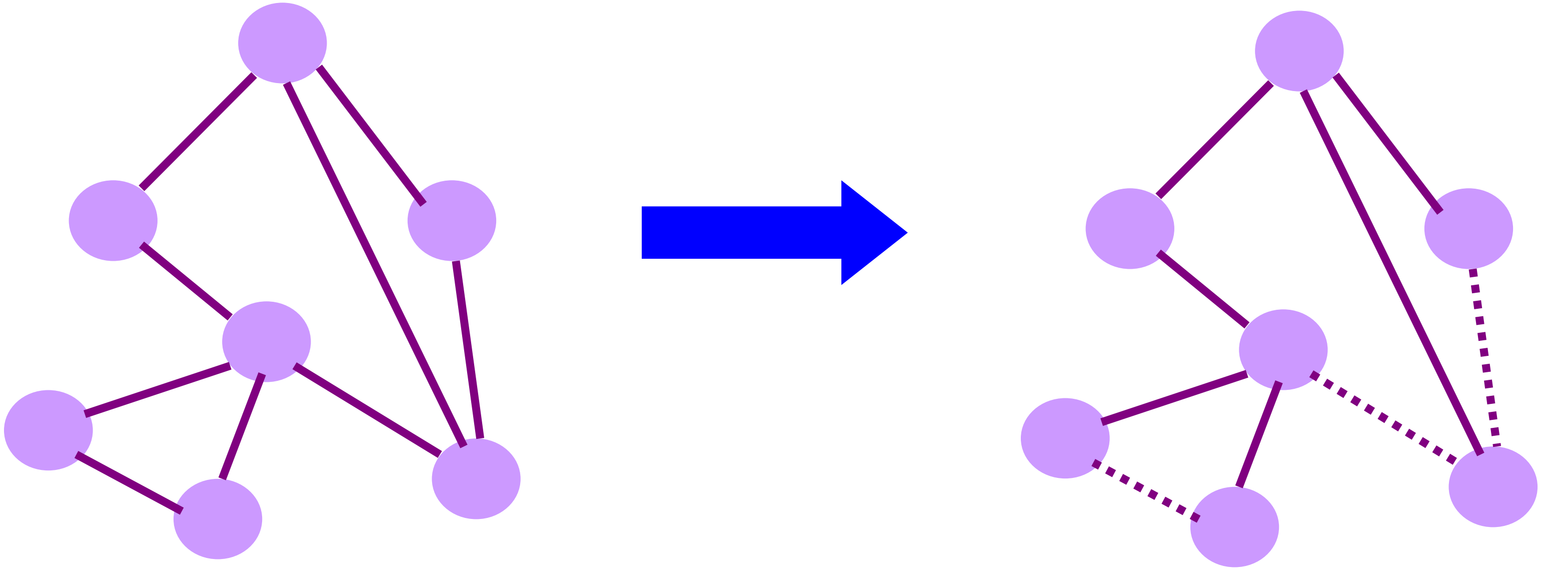
Flooding Can Lead to Loops

- Switches sometimes need to broadcast frames
  - Upon receiving a frame with an unfamiliar destination
  - Upon receiving a frame sent to the broadcast address
- Broadcasting is implemented by flooding
  - Transmitting frame out every interface
  - … except the one where the frame arrived
- Flooding can lead to forwarding loops
  - E.g., if the network contains a cycle of switches
  - Either accidentally, or by design for higher reliability



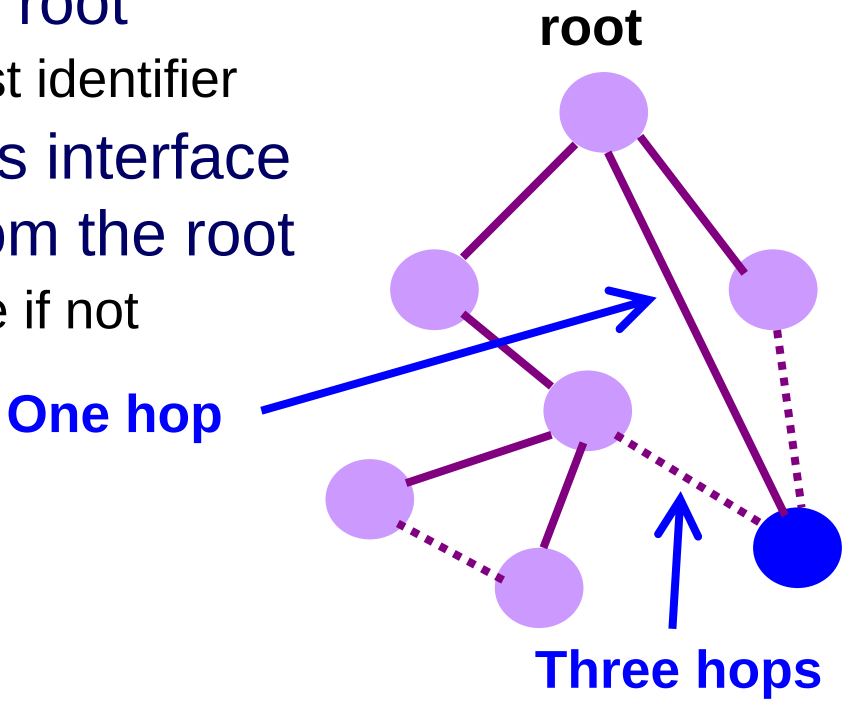
Solution: Spanning Trees

- Ensure the topology has no loops
  - Avoid using some of the links when flooding
  - … to avoid forming a loop
- Spanning tree
  - Sub-graph that covers all vertices but contains no cycles
  - Links not in the spanning tree do not forward frames



Constructing a Spanning Tree

- Need a distributed algorithm
  - Switches cooperate to build the spanning tree
  - … and adapt automatically when failures occur
- Key ingredients of the algorithm
  - Switches need to elect a “root”
    - The switch with the smallest identifier
  - Each switch identifies if its interface is on the shortest path from the root
    - And it exclude from the tree if not
  - Messages (Y, d, X)
    - From node X
    - Claiming Y is the root
    - And the distance is d

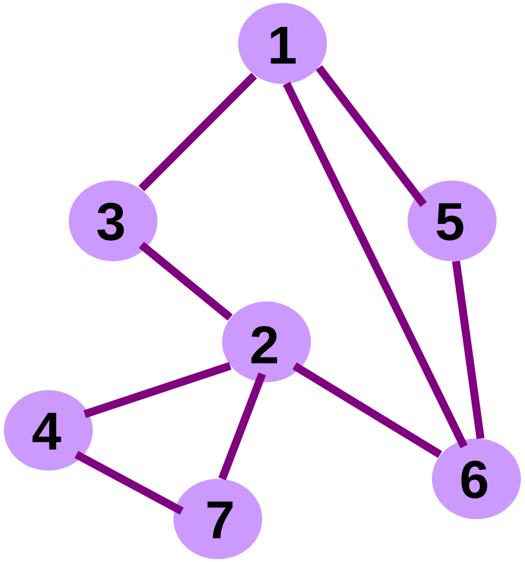


Steps in Spanning Tree Algorithm

- Initially, each switch thinks it is the root
  - Switch sends a message out every interface
  - … identifying itself as the root with distance 0
  - Example: switch X announces (X, 0, X)
- Switches update their view of the root
  - Upon receiving a message, check the root id
  - If the new id is smaller, start viewing that switch as root
- Switches compute their distance from the root
  - Add 1 to the distance received from a neighbor
  - Identify interfaces not on a shortest path to the root
  - … and exclude them from the spanning tree

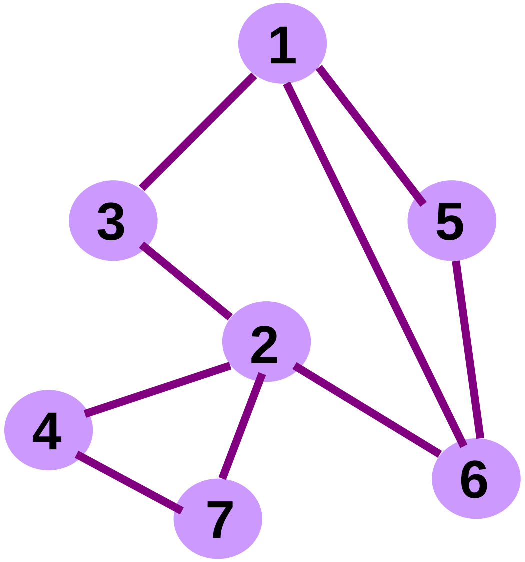
Example From Switch #4's Viewpoint

- Switch #4 thinks it is the root
  - Sends (4, 0, 4) message to 2 and 7
- Then, switch #4 hears from #2
  - Receives (2, 0, 2) message from 2
  - … and thinks that #2 is the root
  - And realizes it is just one hop away
- Then, switch #4 hears from #7
  - Receives (2, 1, 7) from 7
  - And realizes this is a longer path
  - So, prefers its own one-hop path
  - And removes 4-7 link from the tree



Example From Switch #4's Viewpoint

- Switch #2 hears about switch #1
  - Switch 2 hears (1, 1, 3) from 3
  - Switch 2 starts treating 1 as root
  - And sends (1, 2, 2) to neighbors
- Switch #4 hears from switch #2
  - Switch 4 starts treating 1 as root
  - And sends (1, 3, 4) to neighbors
- Switch #4 hears from switch #7
  - Switch 4 receives (1, 3, 7) from 7
  - And realizes this is a longer path
  - So, prefers its own three-hop path
  - And removes 4-7 link from the tree



Robust Spanning Tree Algorithm

- Algorithm must react to failures
  - Failure of the root node
    - Need to elect a new root, with the next lowest identifier
  - Failure of other switches and links
    - Need to recompute the spanning tree
- Root switch continues sending messages
  - Periodically reannouncing itself as the root (1, 0, 1)
  - Other switches continue forwarding messages
- Detecting failures through timeout (soft state!)
  - Switch waits to hear from others
  - Eventually times out and claims to be the root

See Section 3.2.2 in the textbook for details and another example

Evolution Toward Virtual LANs

- In the olden days…
  - Thick cables snaked through cable ducts in buildings
  - Every computer they passed was plugged in
  - All people in adjacent offices were put on the same LAN
  - Independent of whether they belonged together or not
- More recently…
  - Hubs and switches changed all that
  - Every office connected to central wiring closets
  - Often multiple LANs (*k* hubs) connected by switches
  - Flexibility in mapping offices to different LANs

Group users based on organizational structure, rather than the physical layout of the building.

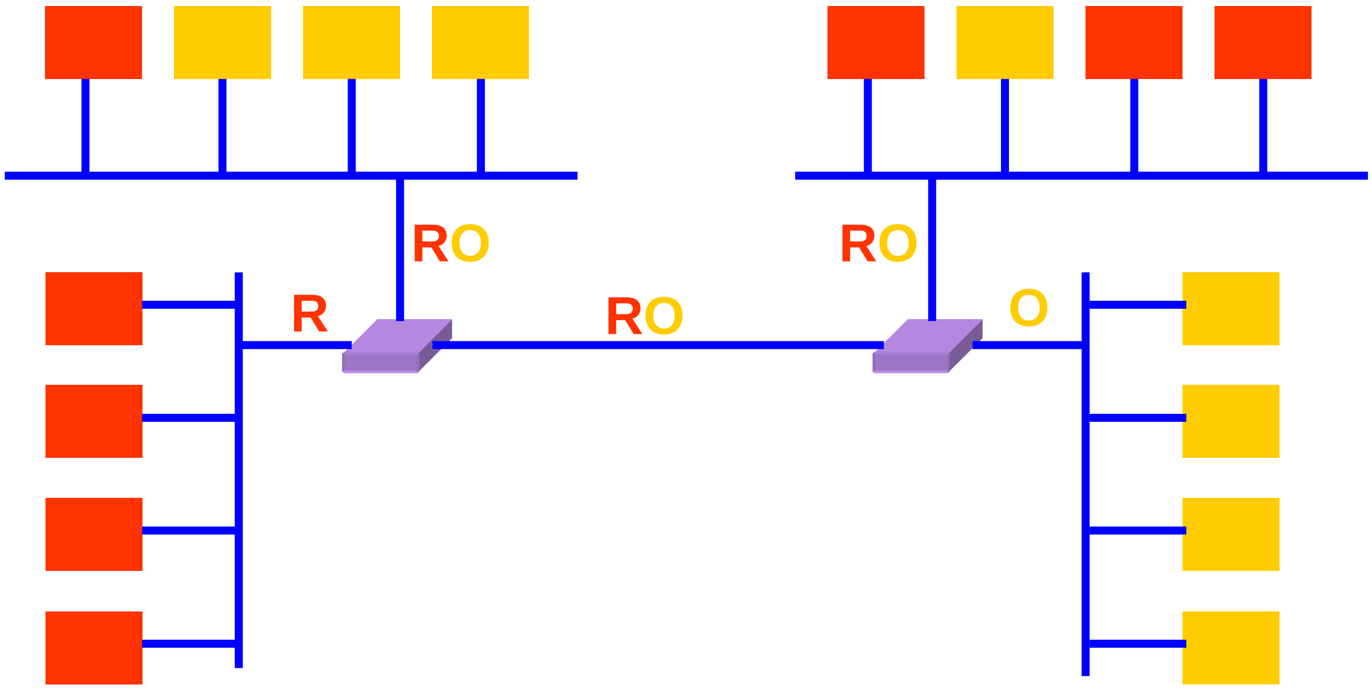
Why Group by Organizational Structure?

- Security
  - Ethernet is a shared media
  - Any interface card can be put into “promiscuous” mode
  - … and get a copy of all of the traffic (e.g., midterm exam)
  - So, isolating traffic on separate LANs improves security
- Load
  - Some LAN segments are more heavily used than others
  - E.g., researchers running experiments get out of hand
  - … can saturate their own segment and not the others
  - Plus, there may be natural locality of communication
  - E.g., traffic between people in the same research group

People Move, and Roles Change

- Organizational changes are frequent
  - E.g., faculty office becomes a grad-student office
  - E.g., graduate student becomes a faculty member
- Physical rewiring is a major pain
  - Requires unplugging the cable from one port
  - … and plugging it into another
  - … and hoping the cable is long enough to reach
  - … and hoping you don’t make a mistake
- Would like to “rewire” the building in software
  - The resulting concept is a Virtual LAN (VLAN)

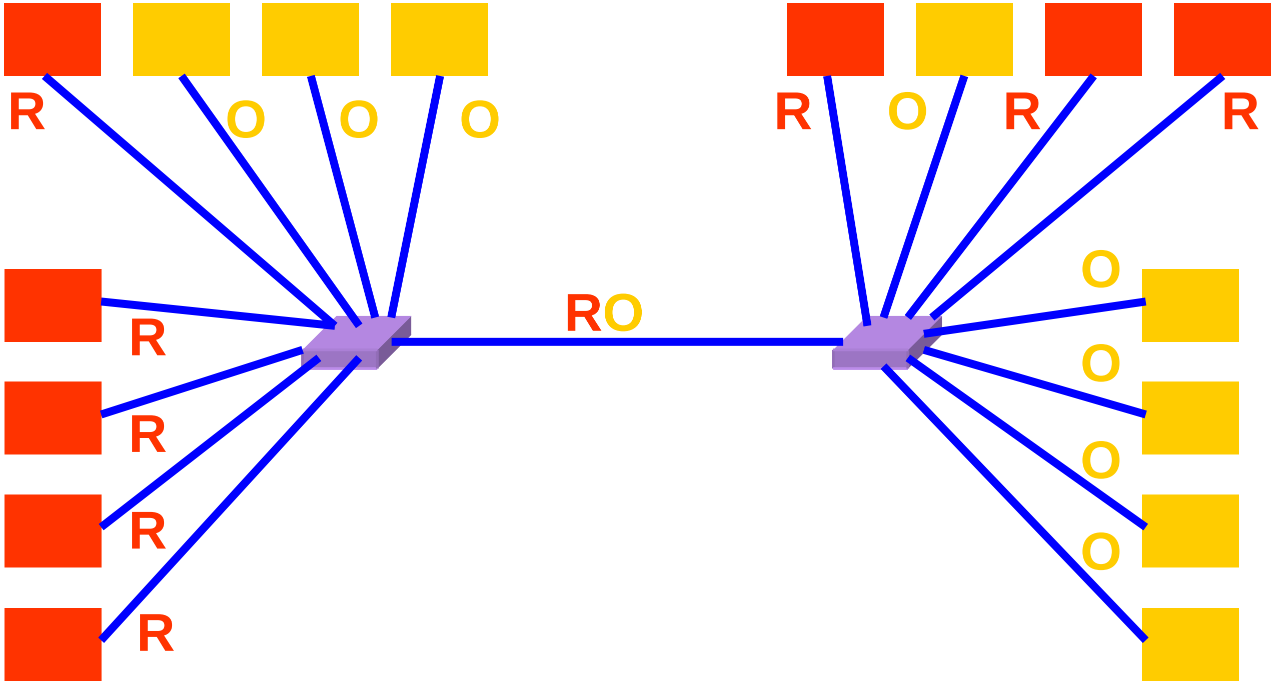
Example: Two Virtual LANs



Red VLAN and Orange VLAN

Bridges forward traffic as needed

Example: Two Virtual LANs



Red VLAN and Orange VLAN
Switches forward traffic as needed

Making VLANs Work

- Bridges/switches need configuration tables
  - Saying which VLANs are accessible via which interfaces
- Approaches to mapping to VLANs
  - Each interface has a VLAN color
    - Only works if all hosts on same segment belong to same VLAN
  - Each MAC address has a VLAN color
    - Useful when hosts on same segment belong to different VLANs
    - Useful when hosts move from one physical location to another
- Changing the Ethernet header
  - Adding a field for a VLAN tag
  - Implemented on the bridges/switches
  - … but can still interoperate with old Ethernet cards

Moving From Switches to Routers

- Advantages of switches over routers
  - Plug-and-play
  - Fast filtering and forwarding of frames
  - No pronunciation ambiguity (e.g., “rooter” vs. “rowter”)
- Disadvantages of switches over routers
  - Topology is restricted to a spanning tree
  - Large networks require large ARP tables
  - Broadcast storms can cause the network to collapse

Comparing Hubs, Switches, & Routers

	hubs	routers	switches
traffic isolation	no	yes	yes
plug & play	yes	no	yes
optimal routing	no	yes	no
cut through	yes	no	yes

Conclusion

- Shuttling data from one link to another
  - Bits, frames, packets, …
  - Repeaters/hubs, bridges/switches, routers, …
- Key ideas in switches
  - Cut-through switching
  - Self learning of the switch table
  - Spanning trees
  - Virtual LANs (VLANs)
- Next time
  - Routing
  - Application-level protocols

IP Addressing and Forwarding

Introduction to Data Networks

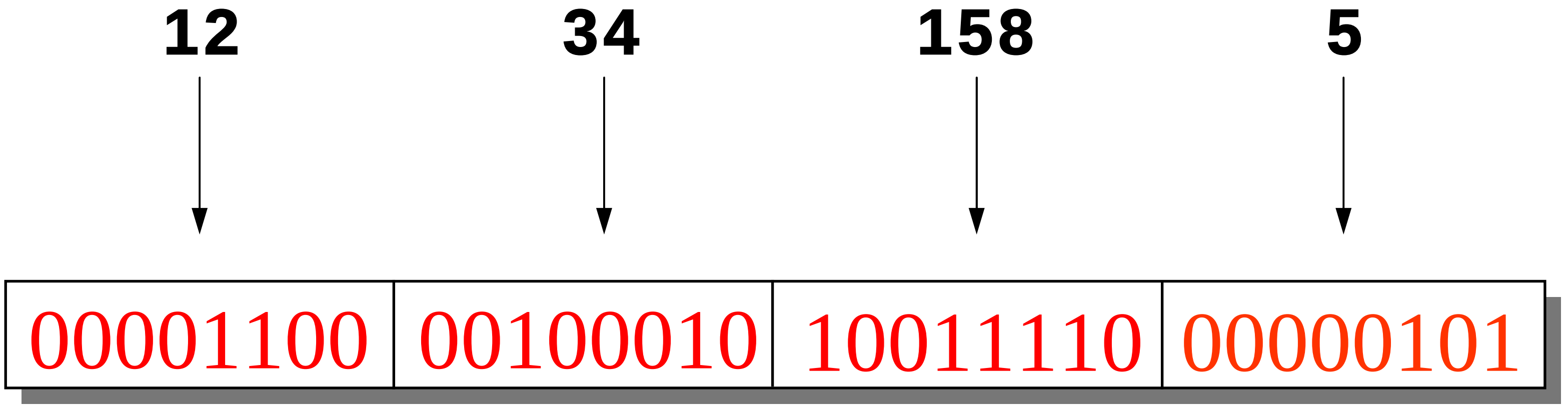
2008.4

Goals of Today’s Lecture

- IP addresses
  - Dotted-quad notation
  - IP prefixes for aggregation
- Address allocation
  - Classful addresses
  - Classless InterDomain Routing (CIDR)
  - Growth in the number of prefixes over time
- Packet forwarding
  - Forwarding tables
  - Longest-prefix match forwarding
  - Where forwarding tables come from

IP Address (IPv4)

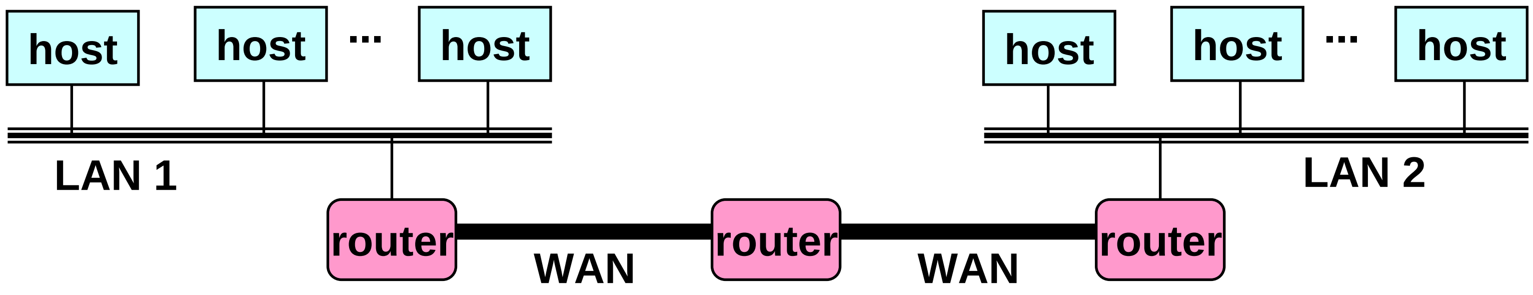
- A unique 32-bit number
- Identifies an interface (on a host, on a router, …)
- Represented in dotted-quad notation



12	34	158	5
00001100	00100010	10011110	00000101

Grouping Related Hosts

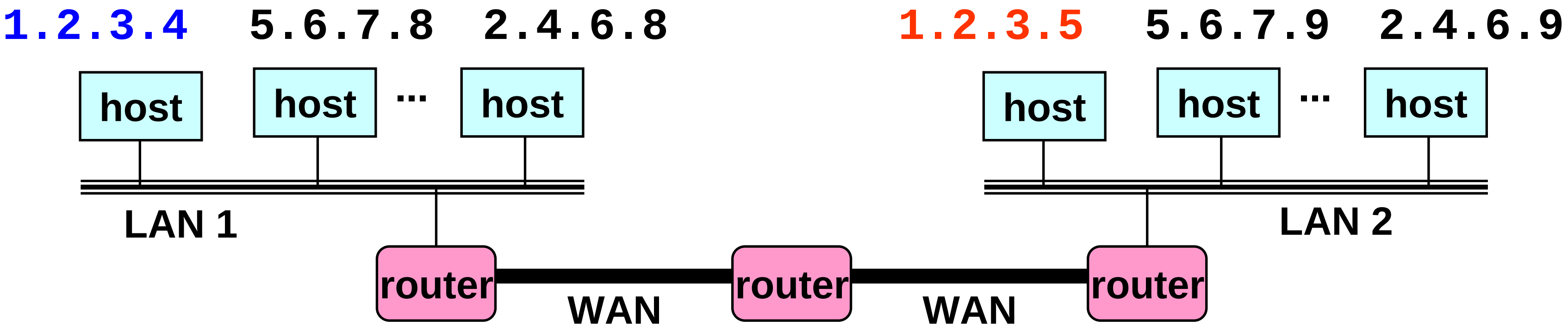
- The Internet is an “inter-network”
  - Used to connect *networks* together, not *hosts*
  - Needs a way to address a network (i.e., group of hosts)



LAN = Local Area Network
WAN = Wide Area Network

Scalability Challenge

- Suppose hosts had arbitrary addresses
  - Then every router would need a lot of information
  - …to know how to direct packets toward the host

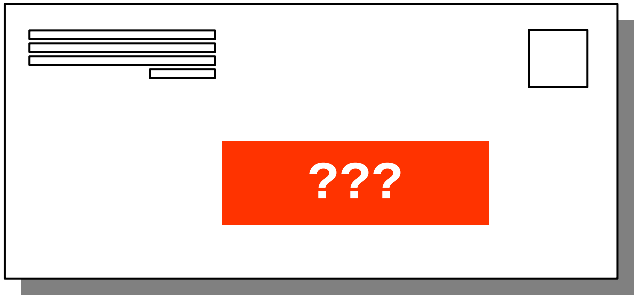


1.2.3.4	←
1.2.3.5	→
⋮	

forwarding table

Hierarchical Addressing in U.S. Mail

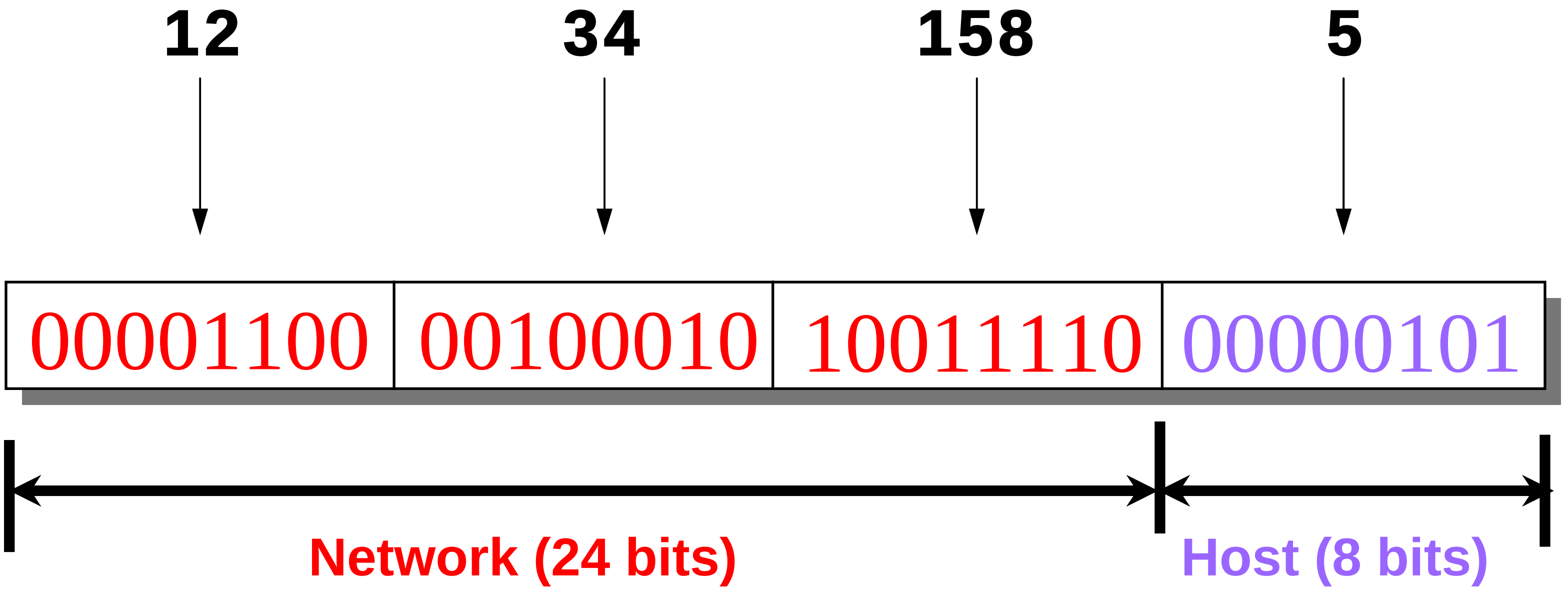
- Addressing in the U.S. mail
  - Zip code: 08540
  - Street: Olden Street
  - Building on street: 35
  - Room in building: 306
  - Name of occupant: Jennifer Rexford



- Forwarding the U.S. mail
  - Deliver letter to the post office in the zip code
  - Assign letter to mailman covering the street
  - Drop letter into mailbox for the building/room
  - Give letter to the appropriate person

Hierarchical Addressing: IP Prefixes

- Divided into network & host portions (left and right)
- 12.34.158.0/24 is a 24-bit prefix with 2^8 addresses



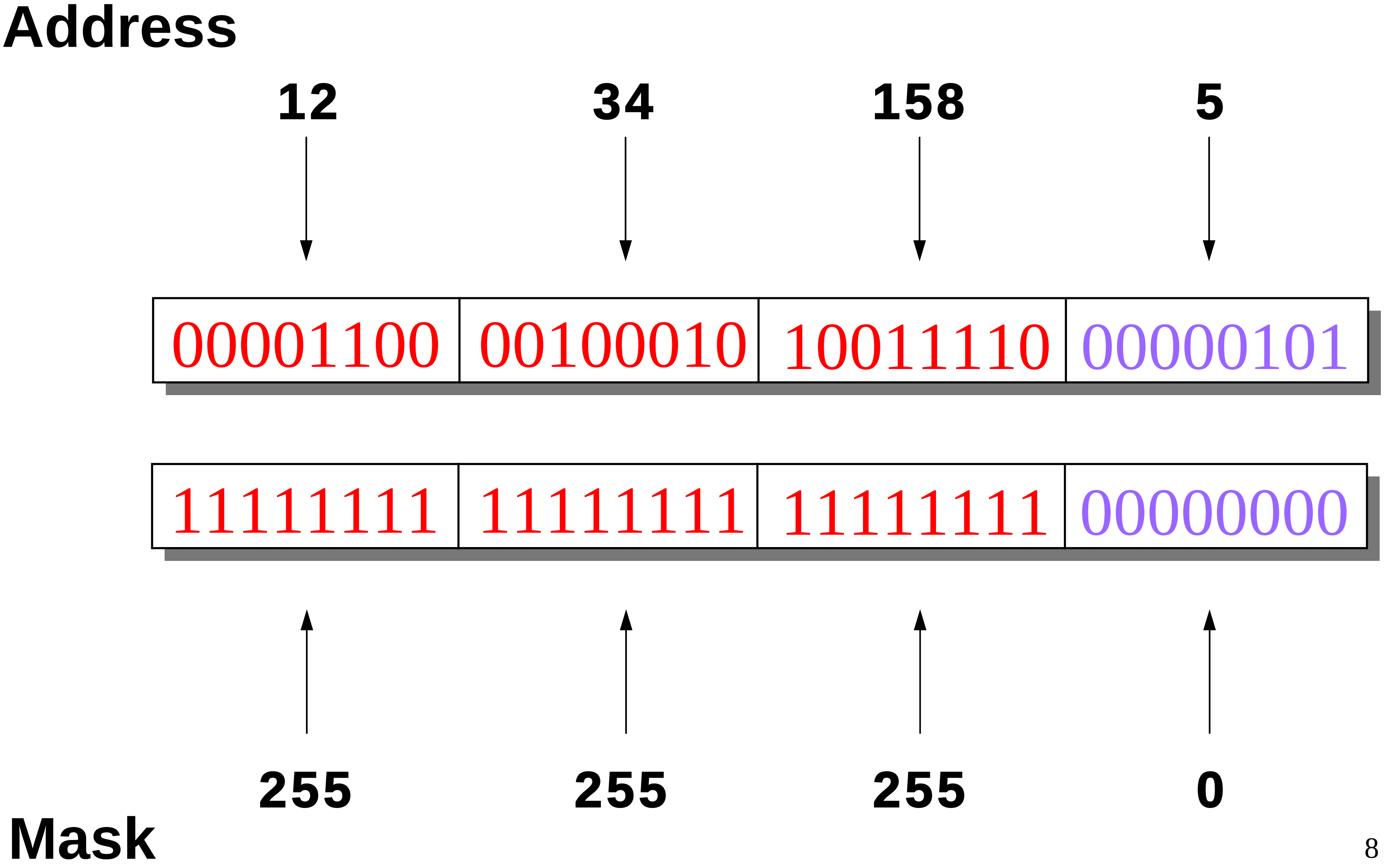
12 34 158 5

00001100 00100010 10011110 00000101

Network (24 bits)

Host (8 bits)

IP Address and a 24-bit Subnet Mask



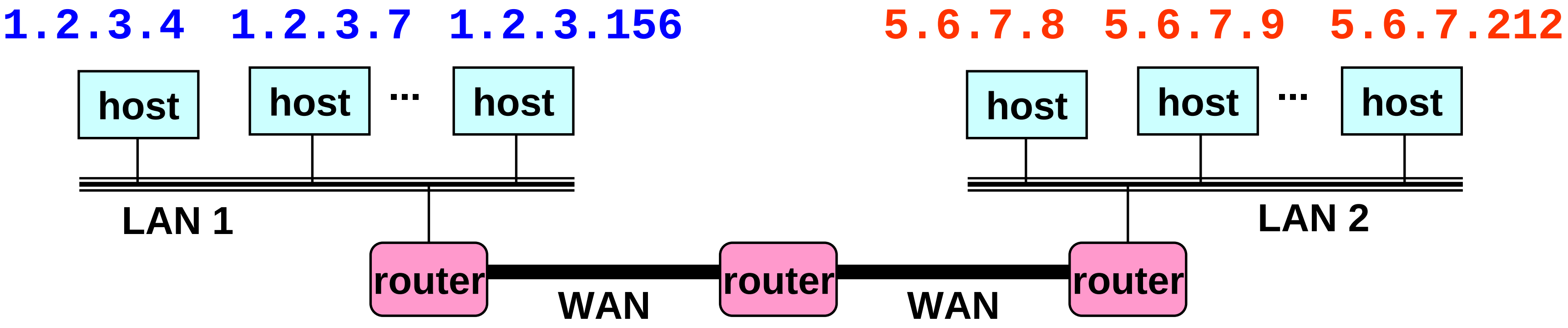
Address

12	34	158	5
00001100	00100010	10011110	00000101
11111111	11111111	11111111	00000000
255	255	255	0

Mask

Scalability Improved

- Number related hosts from a common subnet
  - 1.2.3.0/24 on the left LAN
  - 5.6.7.0/24 on the right LAN

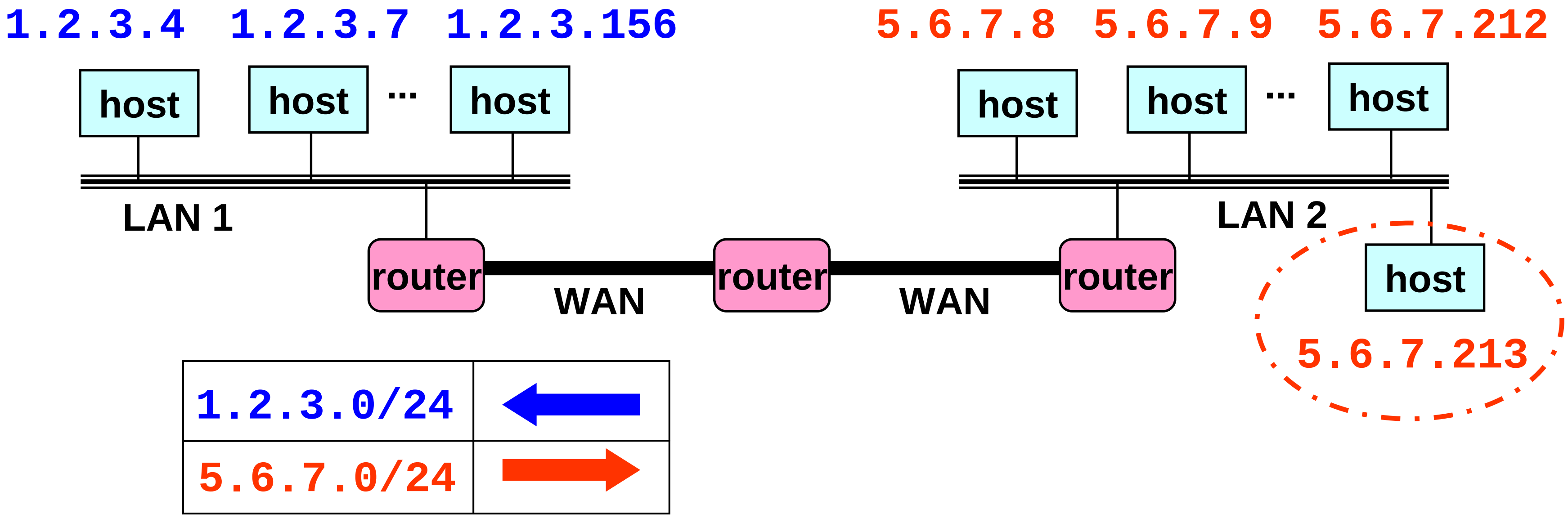


1.2.3.0/24	←
5.6.7.0/24	→

forwarding table

Easy to Add New Hosts

- No need to update the routers
  - E.g., adding a new host 5.6.7.213 on the right
  - Doesn't require adding a new forwarding entry



1.2.3.0/24	←
5.6.7.0/24	→

forwarding table

Address Allocation

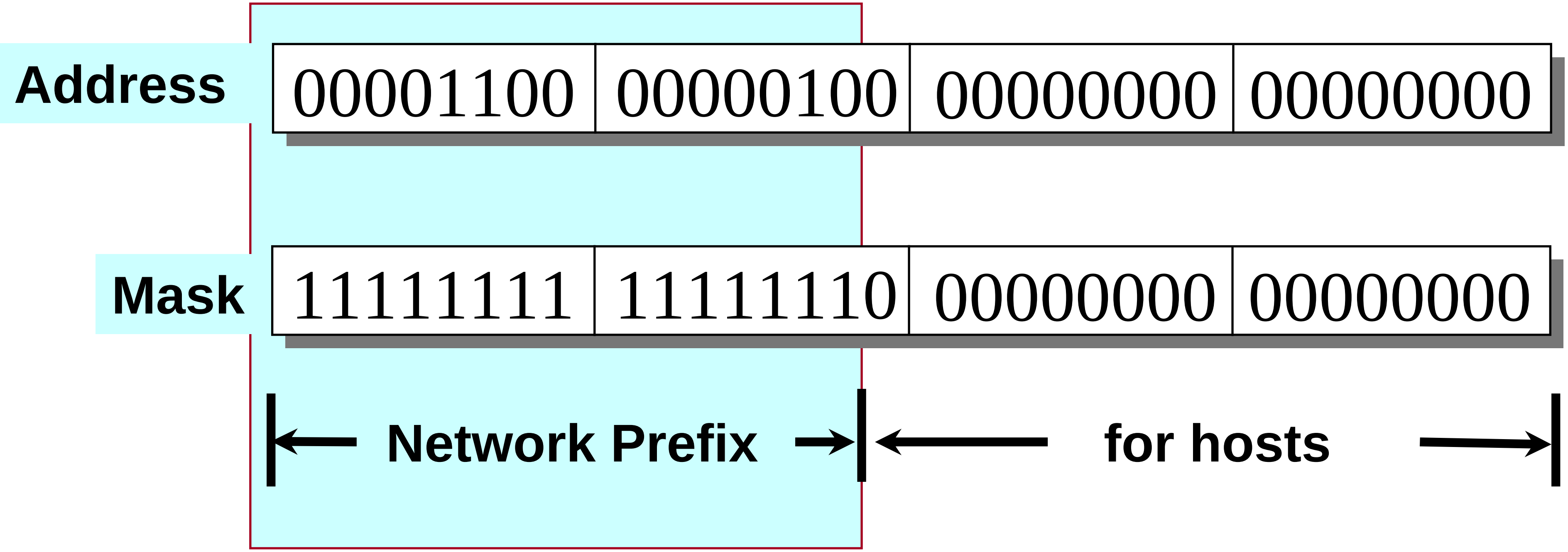
Classful Addressing

- In the olden days, only fixed allocation sizes
  - Class A: 0*
    - Very large /8 blocks (e.g., MIT has 18.0.0.0/8)
  - Class B: 10*
    - Large /16 blocks (e.g,. Princeton has 128.112.0.0/16)
  - Class C: 110*
    - Small /24 blocks (e.g., AT&T Labs has 192.20.225.0/24)
  - Class D: 1110*
    - Multicast groups
  - Class E: 11110*
    - Reserved for future use
- This is why folks use dotted-quad notation!

Classless Inter-Domain Routing (CIDR)

Use two 32-bit numbers to represent a network.
Network number = IP address + Mask

IP Address : 12.4.0.0   IP Mask: 255.254.0.0

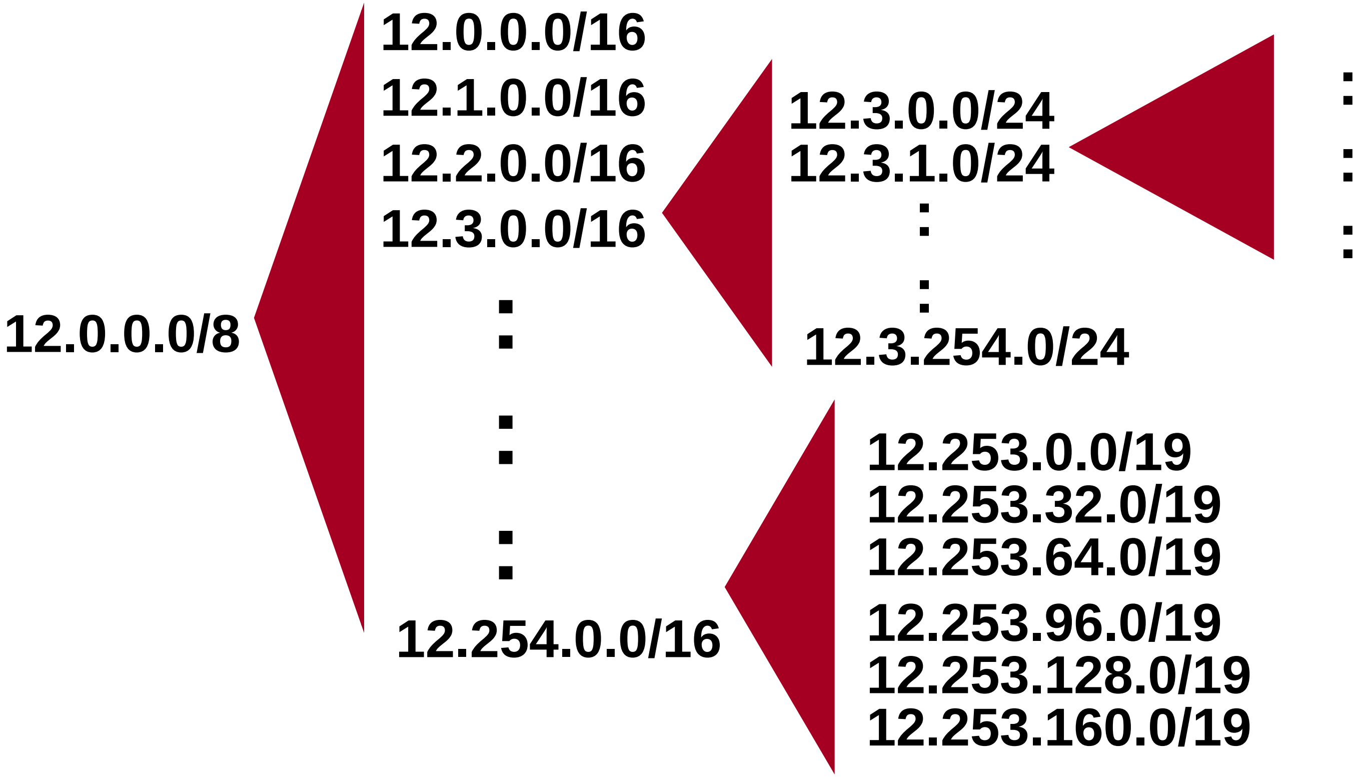


Address	00001100	00000100	00000000	00000000
Mask	11111111	11111110	00000000	00000000

Written as 12.4.0.0/15

CIDR: Hierarchal Address Allocation

- Prefixes are key to Internet scalability
  - Address allocated in contiguous chunks (prefixes)
  - Routing protocols and packet forwarding based on prefixes
  - Today, routing tables contain ~150,000-200,000 prefixes



12.0.0.0/8

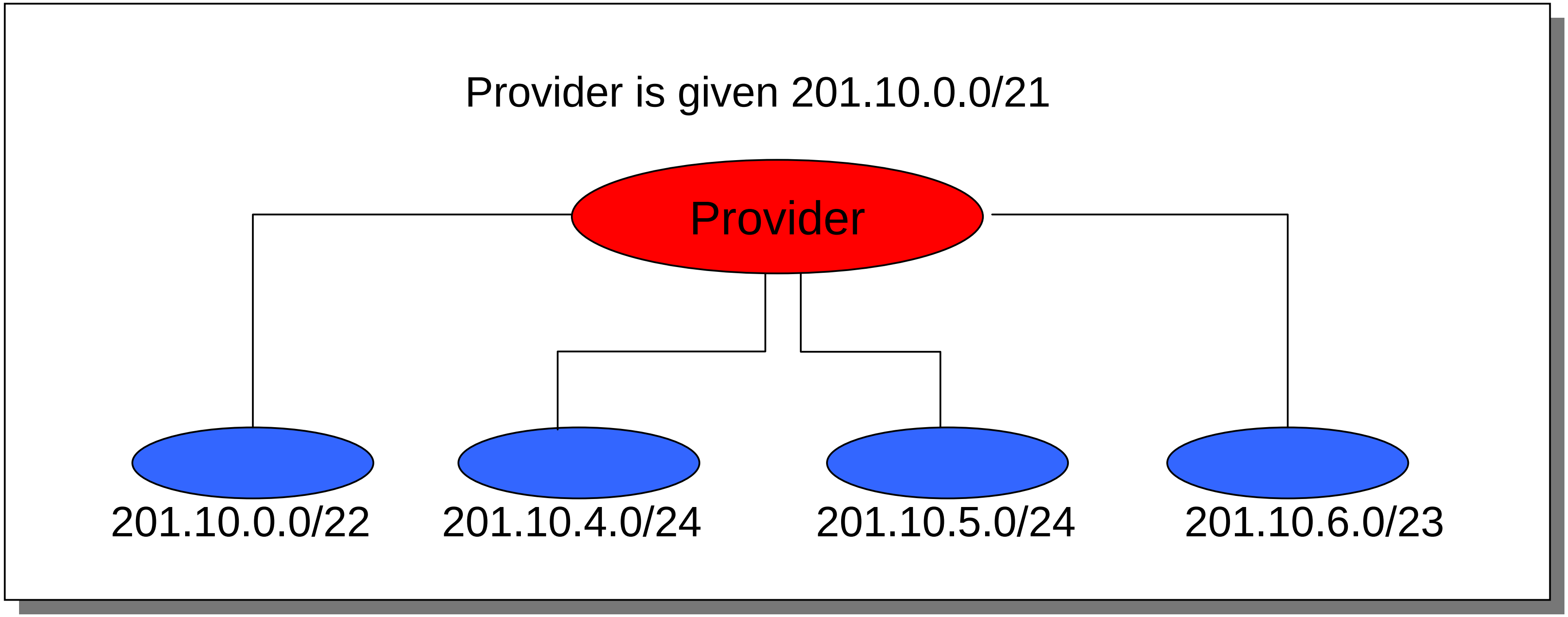
12.0.0.0/16
12.1.0.0/16
12.2.0.0/16
12.3.0.0/16
:
:
:
12.254.0.0/16

12.3.0.0/24
12.3.1.0/24
:
:
12.3.254.0/24

:
:
:

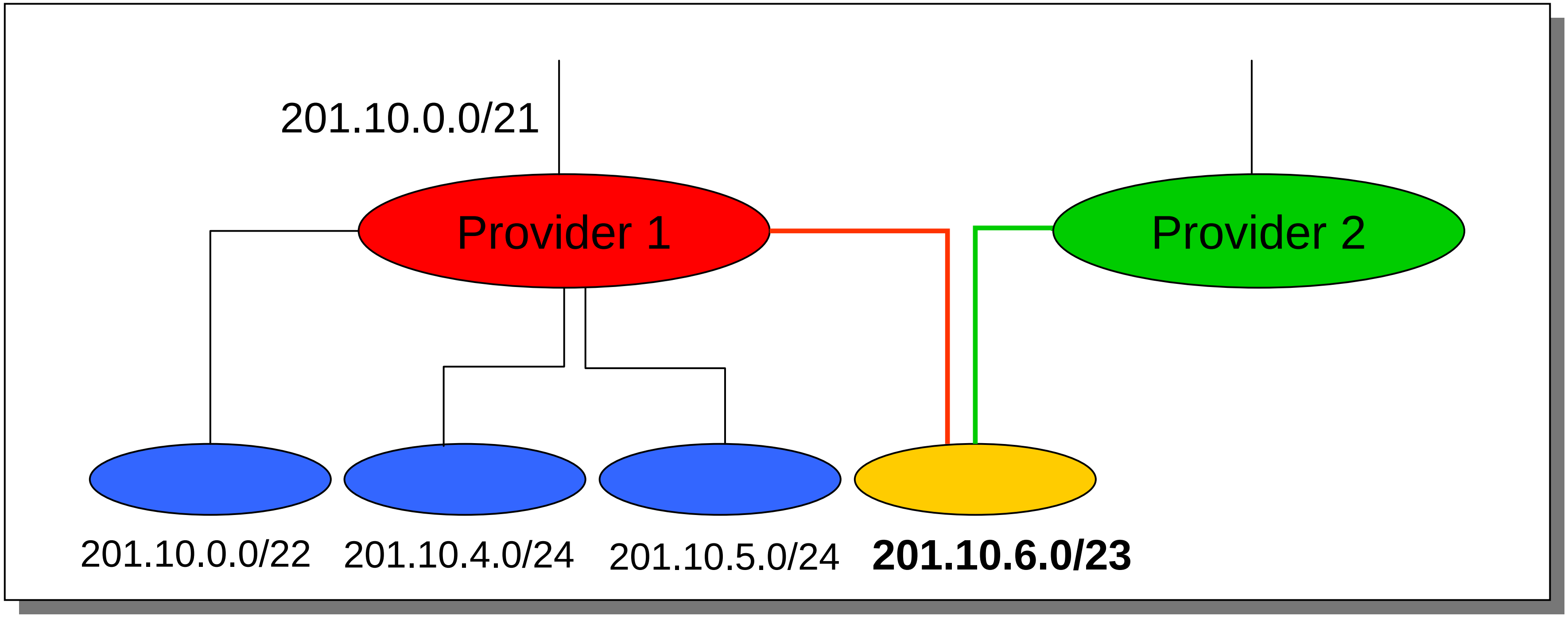
12.253.0.0/19
12.253.32.0/19
12.253.64.0/19
12.253.96.0/19
12.253.128.0/19
12.253.160.0/19

Scalability: Address Aggregation



Routers in the rest of the Internet just need to know how to reach 201.10.0.0/21. The provider can direct the IP packets to the appropriate customer.

But, Aggregation Not Always Possible



Multi-homed **customer with 201.10.6.0/23 has two providers. Other parts of the Internet need to know how to reach these destinations through** ***both*** **providers.**

Summary : Scalability Through Hierarchy

- Hierarchical addressing
  - Critical for scalable system
  - Don’t require everyone to know everyone else
  - Reduces amount of updating when something changes
- Non-uniform hierarchy
  - Useful for heterogeneous networks of different sizes
  - Initial class-based addressing was far too coarse
  - Classless InterDomain Routing (CIDR) helps

Obtaining a Block of Addresses

- Separation of control
  - Prefix: assigned *to* an institution
  - Addresses: assigned *by* the institution to their nodes
- Who assigns prefixes?
  - Internet Corporation for Assigned Names and Numbers
    - Allocates large address blocks to Regional Internet Registries
  - Regional Internet Registries (RIRs)
    - E.g., ARIN (American Registry for Internet Numbers)
    - Allocates address blocks within their regions
    - Allocated to Internet Service Providers and large institutions
  - Internet Service Providers (ISPs)
    - Allocate address blocks to their customers
    - Who may, in turn, allocate to their customers…

Figuring Out Who Owns an Address

- Address registries
  - –Public record of address allocations
  - –Internet Service Providers (ISPs) should update when giving addresses to customers
  - –However, records are notoriously out-of-date
- Ways to query
  - –UNIX: “whois –h whois.arin.net 128.112.136.35”
  - –http://www.arin.net/whois/
  - –http://www.geektools.com/whois.php
  - –…

Example Output for 128.112.136.35

```
OrgName: Princeton University
OrgID: PRNU
Address: Office of Information Technology
Address: 87 Prospect Avenue
City: Princeton
StateProv: NJ
PostalCode: 08544-2007
Country: US
NetRange: 128.112.0.0 - 128.112.255.255
CIDR: 128.112.0.0/16
NetName: PRINCETON
NetHandle: NET-128-112-0-0-1
Parent: NET-128-0-0-0-0
NetType: Direct Allocation
RegDate: 1986-02-24
```

Are 32-bit Addresses Enough?

- Not all that many unique addresses
  - 2^{32} = 4,294,967,296 (just over four billion)
  - Plus, some are reserved for special purposes
  - And, addresses are allocated in larger blocks
- And, many devices need IP addresses
  - Computers, PDAs, routers, tanks, toasters, …
- Long-term solution: a larger address space
  - IPv6 has 128-bit addresses ($2^{128} = 3.403 \times 10^{38}$)
- Short-term solutions: limping along with IPv4
  - Private addresses
  - Network address translation (NAT)
  - Dynamically-assigned addresses (DHCP)

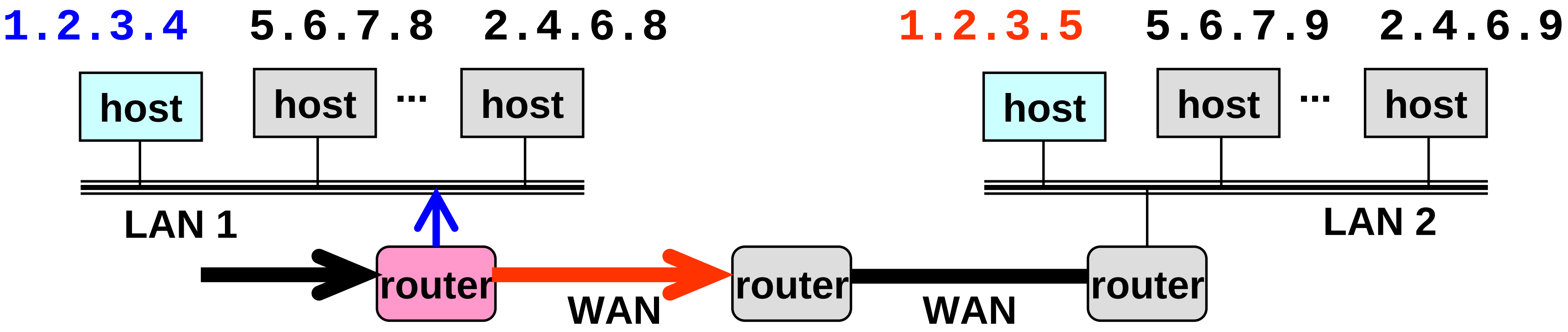
Packet Forwarding

Hop-by-Hop Packet Forwarding

- Each router has a forwarding table
  - Maps destination addresses…
  - … to outgoing interfaces
- Upon receiving a packet
  - Inspect the destination IP address in the header
  - Index into the table
  - Determine the outgoing interface
  - Forward the packet out that interface
- Then, the next router in the path repeats
  - And the packet travels along the path to the destination

Separate Table Entries Per Address

- If a router had a forwarding entry per IP address
  - Match *destination address* of incoming packet
  - … to the *forwarding-table entry*
  - … to determine the *outgoing interface*

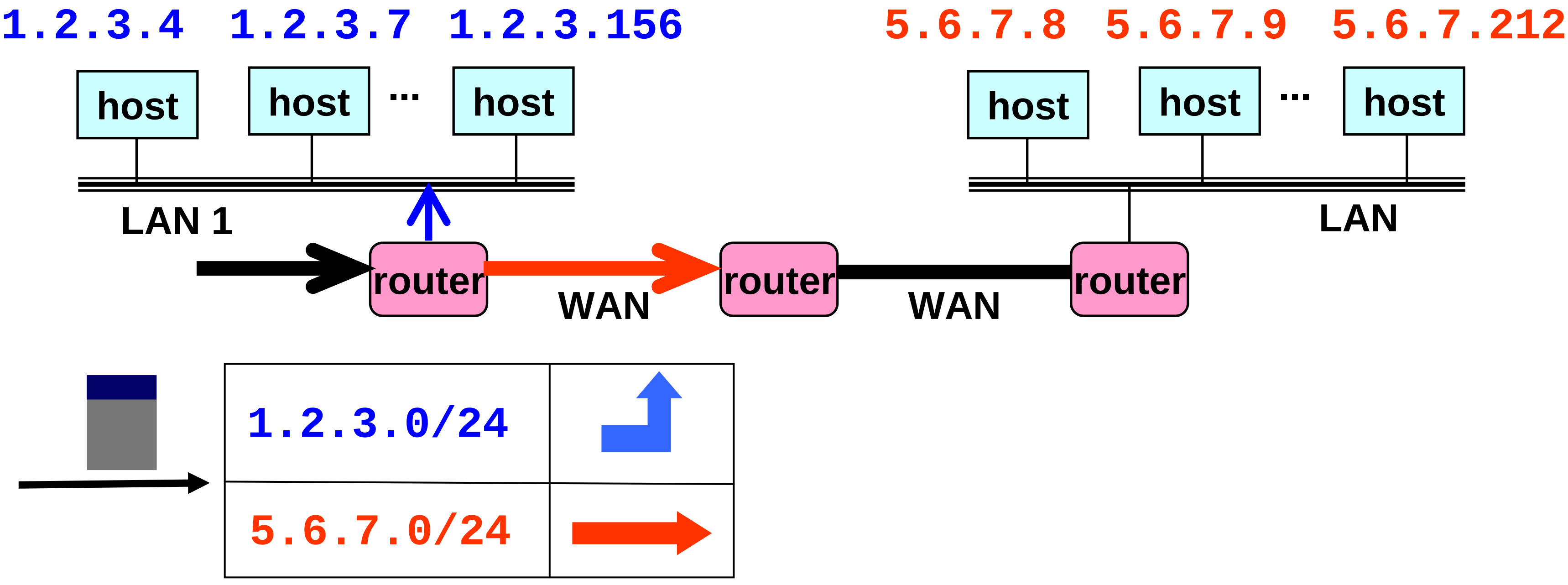


1.2.3.4	↱
1.2.3.5	→

forwarding table

Separate Entry Per 24-bit Prefix

- If the router had an entry per 24-bit prefix
  - Look only at the top 24 bits of the destination address
  - Index into the table to determine the next-hop interface



1.2.3.0/24	↱
5.6.7.0/24	→

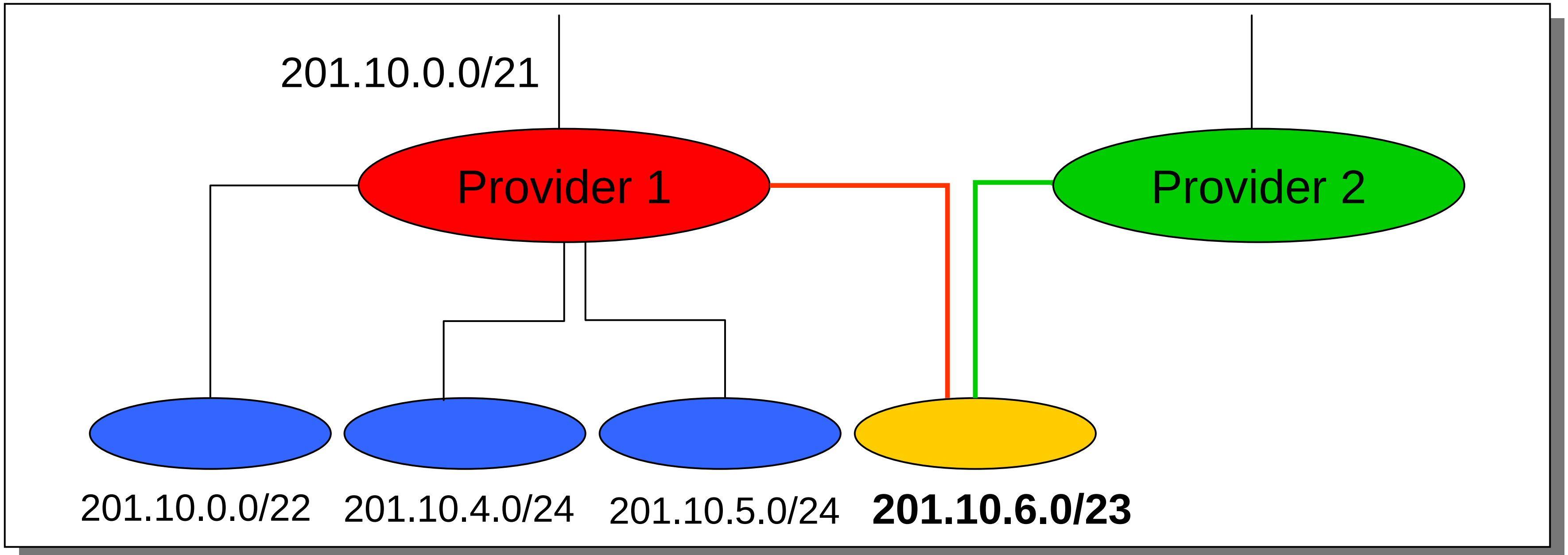
forwarding table

Separate Entry Classful Address

- If the router had an entry per classful prefix
  - Mixture of Class A, B, and C addresses
  - Depends on the first couple of bits of the destination
- Identify the mask automatically from the address
  - First bit of 0: class A address (/8)
  - First two bits of 10: class B address (/16)
  - First three bits of 110: class C address (/24)
- Then, look in the forwarding table for the match
  - E.g., 1.2.3.4 maps to 1.2.3.0/24
  - Then, look up the entry for 1.2.3.0/24
  - … to identify the outgoing interface

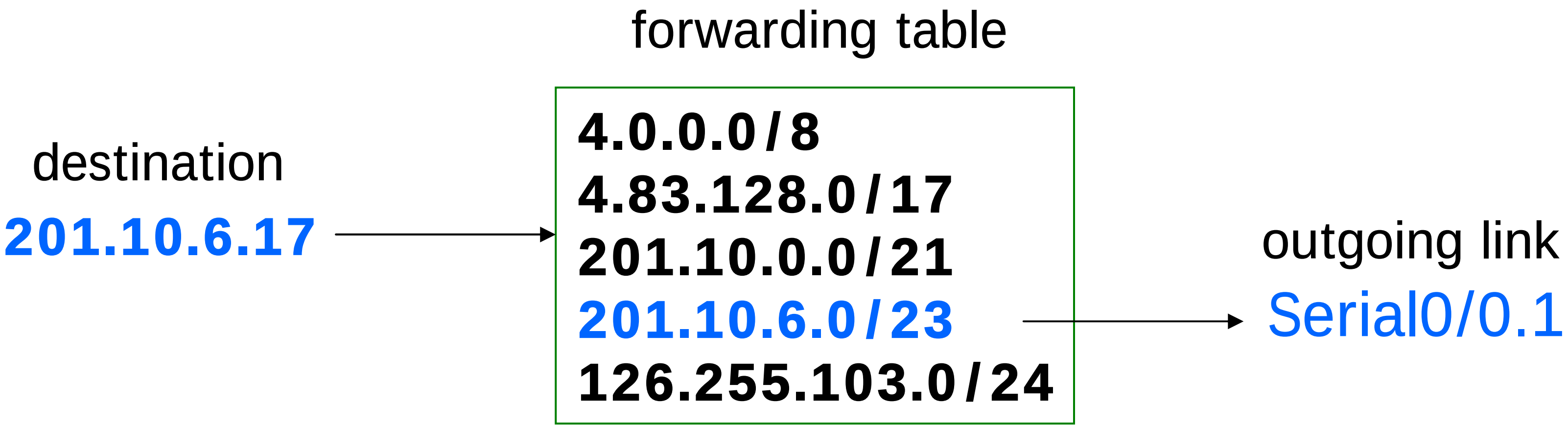
CIDR Makes Packet Forwarding Harder

- There's no such thing as a free lunch
  - CIDR allows efficient use of the limited address space
  - But, CIDR makes packet forwarding much harder
- Forwarding table may have many matches
  - E.g., table entries for 201.10.0.0/21 and 201.10.6.0/23
  - The IP address 201.10.6.17 would match *both*!



Longest Prefix Match Forwarding

- Forwarding tables in IP routers
  - Maps each IP prefix to next-hop link(s)
- Destination-based forwarding
  - Packet has a destination address
  - Router identifies longest-matching prefix
  - Cute algorithmic problem: very fast lookups



forwarding table

destination

201.10.6.17

4.0.0.0/8
4.83.128.0/17
201.10.0.0/21
201.10.6.0/23
126.255.103.0/24

outgoing link

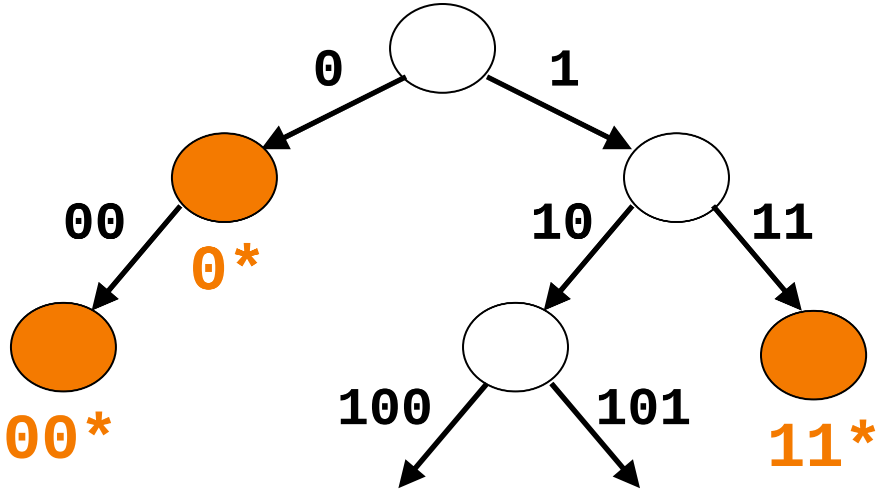
Serial0/0.1

Simplest Algorithm is Too Slow

- Scan the forwarding table one entry at a time
  - See if the destination matches the entry
  - If so, check the size of the mask for the prefix
  - Keep track of the entry with longest-matching prefix
- Overhead is linear in size of the forwarding table
  - Today, that means 150,000-200,000 entries!
  - And, the router may have just a few nanoseconds
  - … before the next packet is arriving
- Need greater efficiency to keep up with *line rate*
  - Better algorithms
  - Hardware implementations

Patricia Tree

- Store the prefixes as a tree
  - One bit for each level of the tree
  - Some nodes correspond to valid prefixes
  - ... which have next-hop interfaces in a table
- When a packet arrives
  - Traverse the tree based on the destination address
  - Stop upon reaching the longest matching prefix



Even Faster Lookups

- Patricia tree is faster than linear scan
  - Proportional to number of bits in the address
- Patricia tree can be made faster
  - Can make a k-ary tree
    - E.g., 4-ary tree with four children (00, 01, 10, and 11)
  - Faster lookup, though requires more space
- Can use special hardware
  - Content Addressable Memories (CAMs)
  - Allows look-ups on a key rather than flat address
- Huge innovations in the mid-to-late 1990s
  - After CIDR was introduced (in 1994)
  - … and longest-prefix match was a major bottleneck

Where do Forwarding Tables Come From?

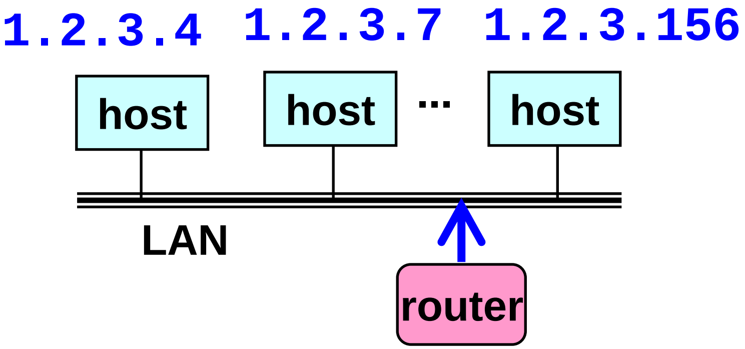
- Routers have forwarding tables
  - Map prefix to outgoing link(s)
- Entries can be statically configured
  - E.g., “map 12.34.158.0/24 to Serial0/0.1”
- But, this doesn’t adapt
  - To failures
  - To new equipment
  - To the need to balance load
  - …
- That is where other technologies come in…
  - Routing protocols, DHCP, and ARP (later in course)

What End Hosts Sending to Others?

- End host with single network interface
  - – PC with an Ethernet link
  - – Laptop with a wireless link
- Don’t need to run a routing protocol
  - – Packets to the host itself (e.g., 1.2.3.4/32)
    - Delivered locally
  - – Packets to other hosts on the LAN (e.g., 1.2.3.0/24)
    - Sent out the interface
  - – Packets to external hosts (e.g., 0.0.0.0/0)
    - Sent out interface to local gateway
- How this information is learned
  - – Static setting of address, subnet mask, and gateway
  - – Dynamic Host Configuration Protocol (DHCP)

What About Reaching the End Hosts?

- How does the last router reach the destination?



- Each interface has a persistent, global identifier
  - MAC (Media Access Control) address
  - Burned in to the adaptors Read-Only Memory (ROM)
  - Flat address structure (i.e., no hierarchy)
- Constructing an address resolution table
  - Mapping MAC address to/from IP address
  - Address Resolution Protocol (ARP)

Conclusions

- IP address
  - A 32-bit number
  - Allocated in prefixes
  - Non-uniform hierarchy for scalability and flexibility
- Packet forwarding
  - Based on IP prefixes
  - Longest-prefix-match forwarding
- We'll cover some topics later
  - Routing protocols, DHCP, and ARP

Internet Control Protocols

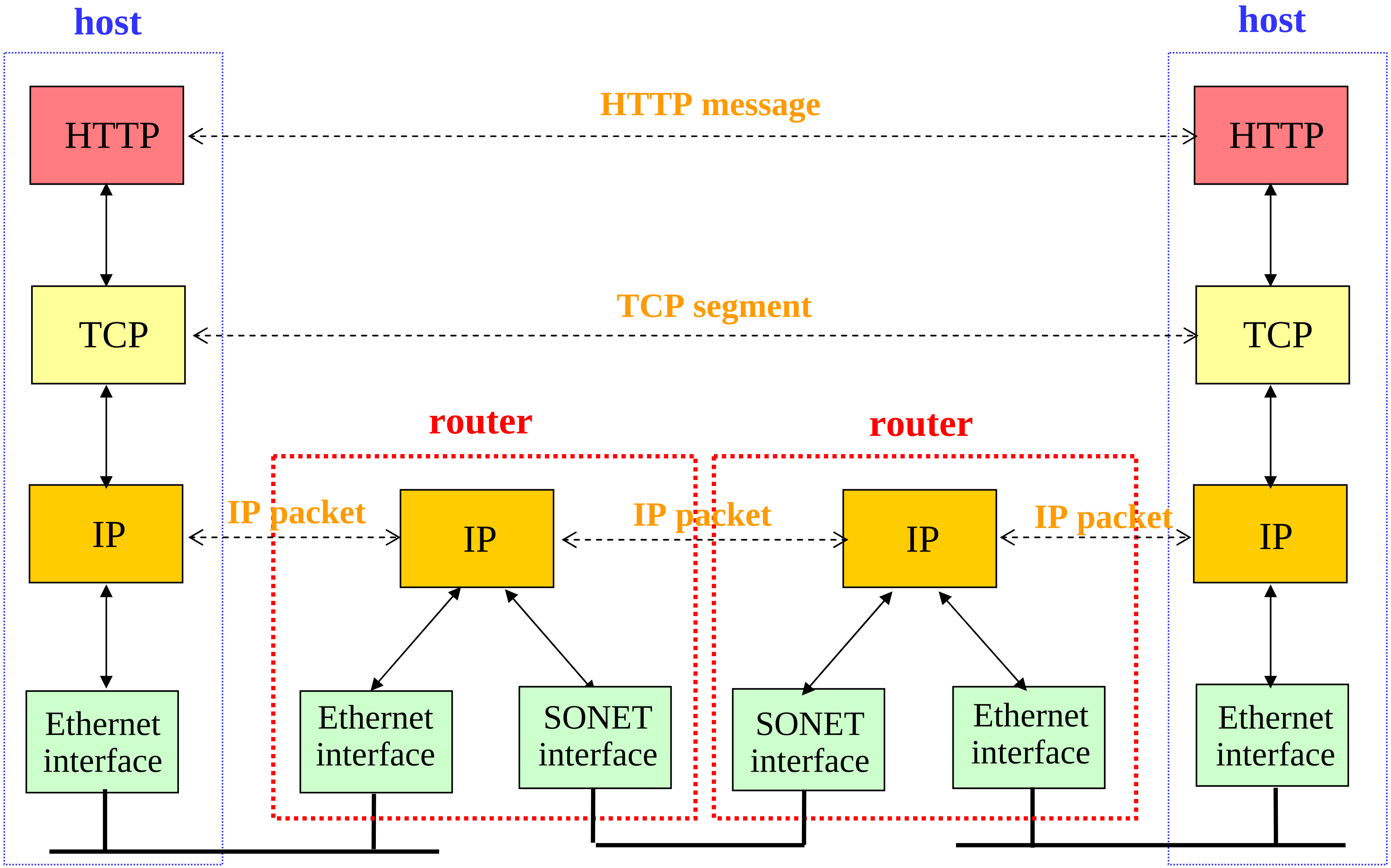
Introduction to Data Networks

2008.4

Goals of Today's Lecture

- Bootstrapping an end host
  - Learning its own configuration parameters (DHCP)
  - Learning the link-layer addresses of other nodes (ARP)
- IP routers
  - Line cards, switching fabric, and route processor
  - Error reporting and monitoring (with ICMP)

Thus Far in the Class…

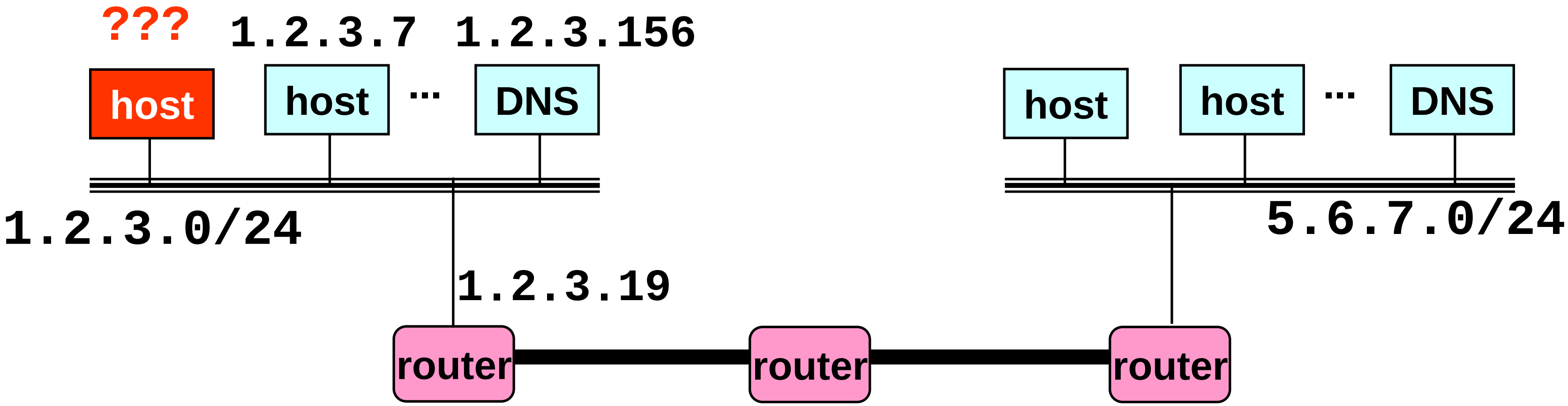


At Each Layer …

- Application protocols
  - Socket abstraction
  - HyperText Transfer Protocol, File Transfer Protocol (FTP)
- Transport services built on IP
  - TCP: reliable byte stream with congestion control
  - UDP: unreliable message delivery
- Name/address translation
  - DNS: mapping host names to/from IP addresses
- Internet Protocol (IP)
  - Best-effort packet delivery service
  - IP addresses and IP prefixes
  - Packet forwarding based on longest-prefix match

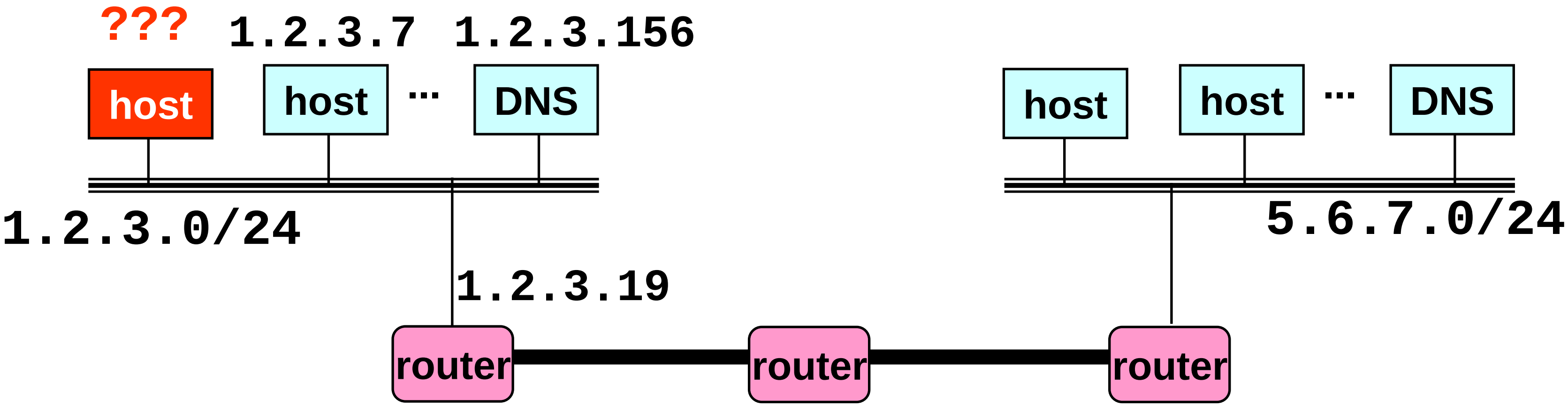
How To Bootstrap an End Host?

- What IP address the host should use?
- What local Domain Name System server to use?
- How to send packets to remote destinations?
- How to ensure incoming packets arrive?



Avoiding Manual Configuration

- Dynamic Host Configuration Protocol (DHCP)
  - End host learns how to send packets
  - Learn IP address, DNS servers, and gateway
- Address Resolution Protocol (ARP)
  - Others learn how to send packets to the end host
  - Learn mapping between IP address and MAC address



Key Ideas in Both Protocols

- Broadcasting: when in doubt, shout!
  - Broadcast query to all hosts in the local-area-network
  - … when you don’t know how to identify the right one
- Caching: remember the past for a while
  - Store the information you learn to reduce overhead
  - Remember your own address & other host’s addresses
- Soft state: eventually forget the past
  - Associate a time-to-live field with the information
  - … and either refresh or discard the information
  - Key for robustness in the face of unpredictable change

Need Yet Another Kind of Identity

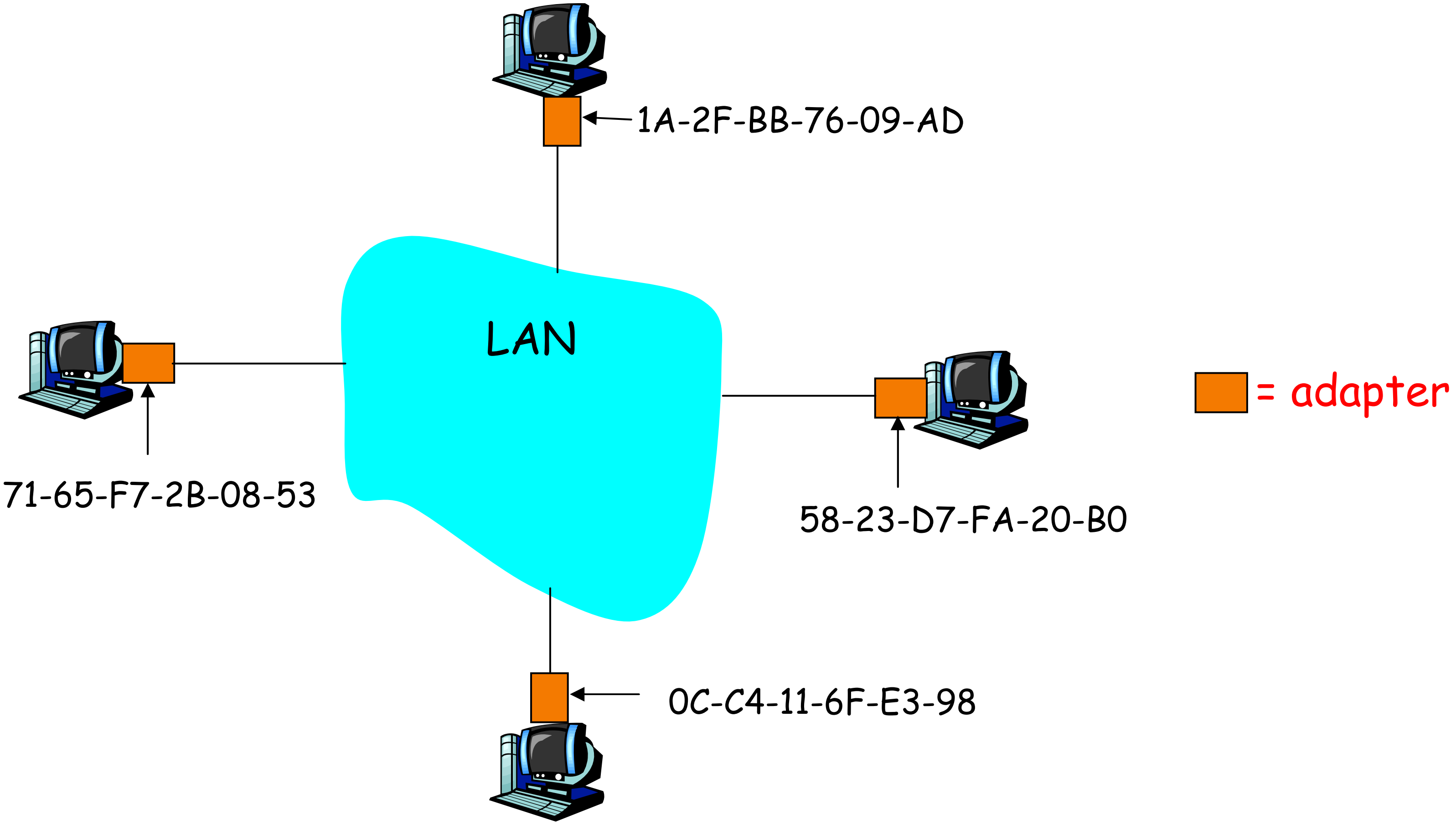
- LANs are designed for arbitrary network protocols
  - Not just for IP and the Internet
- Using IP address would require reconfiguration
  - Every time the adapter was moved or powered up
- Broadcasting all data to all adapters is expensive
  - Requires every host on the LAN to inspect each packet

Motivates separate Medium Access Control (MAC) addresses

MAC Address vs. IP Address

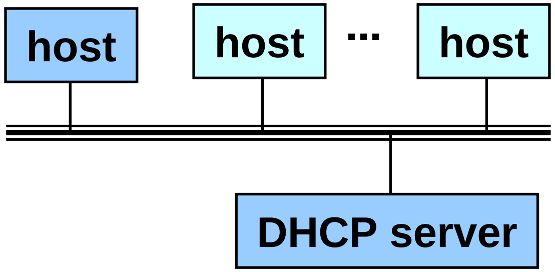
- MAC addresses
  - Hard-coded in read-only memory when adaptor is built
  - Like a social security number
  - Flat name space of 48 bits (e.g., 00-0E-9B-6E-49-76)
  - Portable, and can stay the same as the host moves
  - Used to get packet between interfaces on same network
- IP addresses
  - Configured, or learned dynamically
  - Like a postal mailing address
  - Hierarchical name space of 32 bits (e.g., 12.178.66.9)
  - Not portable, and depends on where the host is attached
  - Used to get a packet to destination IP subnet

MAC Addresses on a LAN



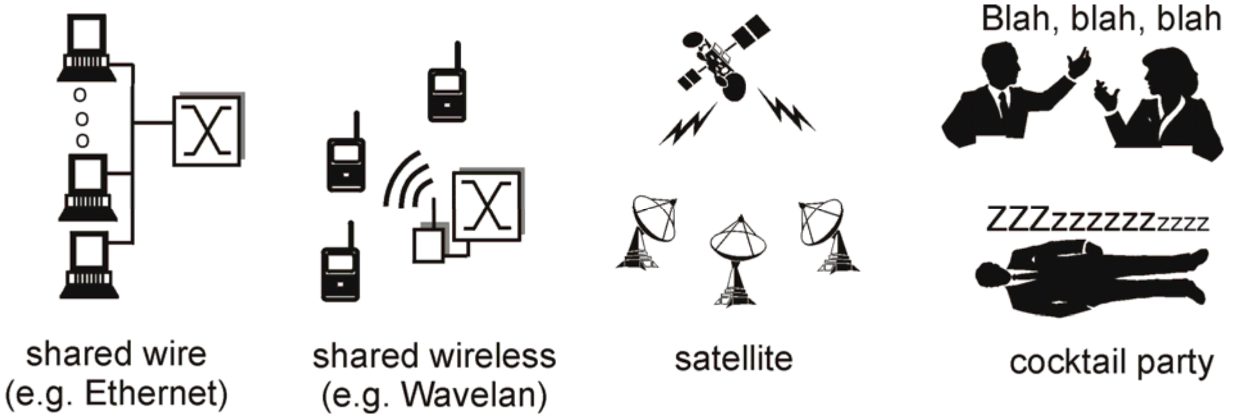
Bootstrapping Problem

- Host doesn't have an IP address yet
  - So, host doesn't know what source address to use
- Host doesn't know who to ask for an IP address
  - So, host doesn't know what destination address to use
- Solution: shout to discover a server who can help
  - Broadcast a server-discovery message
  - Server sends a reply offering an address



Broadcasting

- Broadcasting: sending to everyone
  - Special destination address: FF-FF-FF-FF-FF-FF
  - All adapters on the LAN receive the packet
- Delivering a broadcast packet
  - Easy on a “shared media”
  - Like shouting in a room – everyone can hear you
  - E.g., Ethernet, wireless, and satellite links



shared wire (e.g. Ethernet)

shared wireless (e.g. Wavelan)

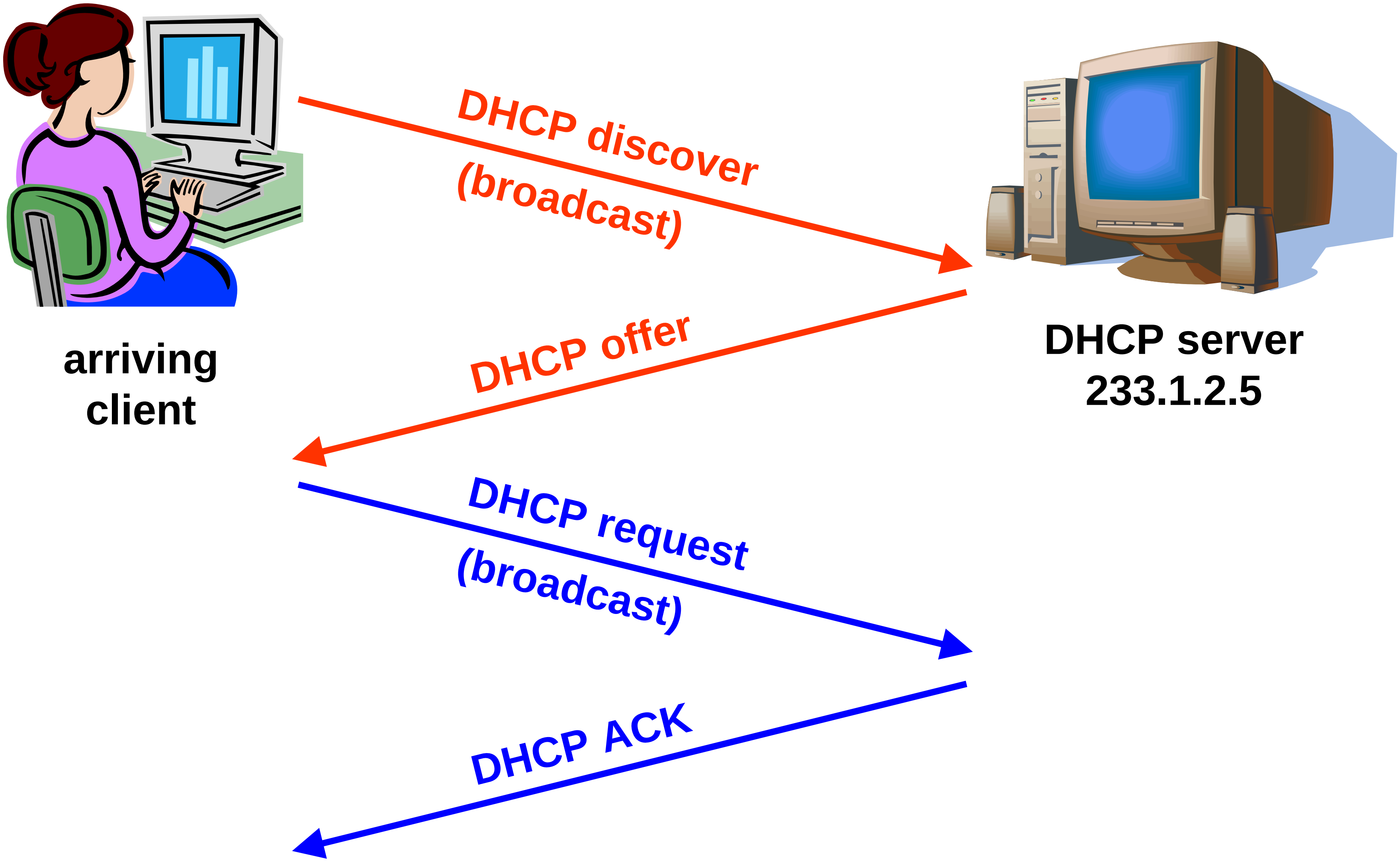
satellite

cocktail party

Response from the DHCP Server

- DHCP “offer message” from the server
  - Configuration parameters (proposed IP address, mask, gateway router, DNS server, ...)
  - Lease time (the time the information remains valid)
- Multiple servers may respond
  - Multiple servers on the same broadcast media
  - Each may respond with an offer
  - The client can decide which offer to accept
- Accepting one of the offers
  - Client sends a DHCP request echoing the parameters
  - The DHCP server responds with an ACK to confirm
  - … and the other servers see they were not chosen

Dynamic Host Configuration Protocol



DHCP discover (broadcast)

DHCP offer

DHCP request (broadcast)

DHCP ACK

arriving client

DHCP server 233.1.2.5

Deciding What IP Address to Offer

- Server as centralized configuration database
  - All parameters are statically configured in the server
  - E.g., a dedicated IP address for each MAC address
  - Avoids complexity of configuring hosts directly
  - … while still having a permanent IP address per host
- Or, dynamic assignment of IP addresses
  - Server maintains a pool of available addresses
  - … and assigns them to hosts on demand
  - Leads to less configuration complexity
  - … and more efficient use of the pool of addresses
  - Though, it is harder to track the same host over time

Soft State: Refresh or Forget

- Why is a lease time necessary?
  - Client can release the IP address (DHCP RELEASE)
    - E.g., "ipconfig /release" at the DOS prompt
    - E.g., clean shutdown of the computer
  - But, the host might not release the address
    - E.g., the host crashes (blue screen of death!)
    - E.g., buggy client software
  - And you don't want the address to be allocated forever

- Performance trade-offs
  - Short lease time: returns inactive addresses quickly
  - Long lease time: avoids overhead of frequent renewals

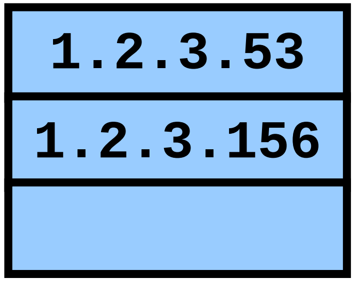
So, Now the Host Knows Things

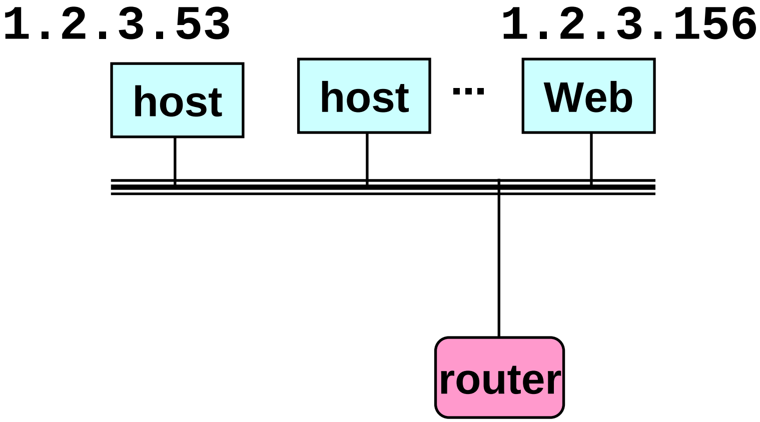
- IP address
- Mask
- Gateway router
- DNS server
- …

- And can send packets to other IP addresses
  - But, how to learn the MAC address of the destination?

Sending Packets Over a Link

IP packet





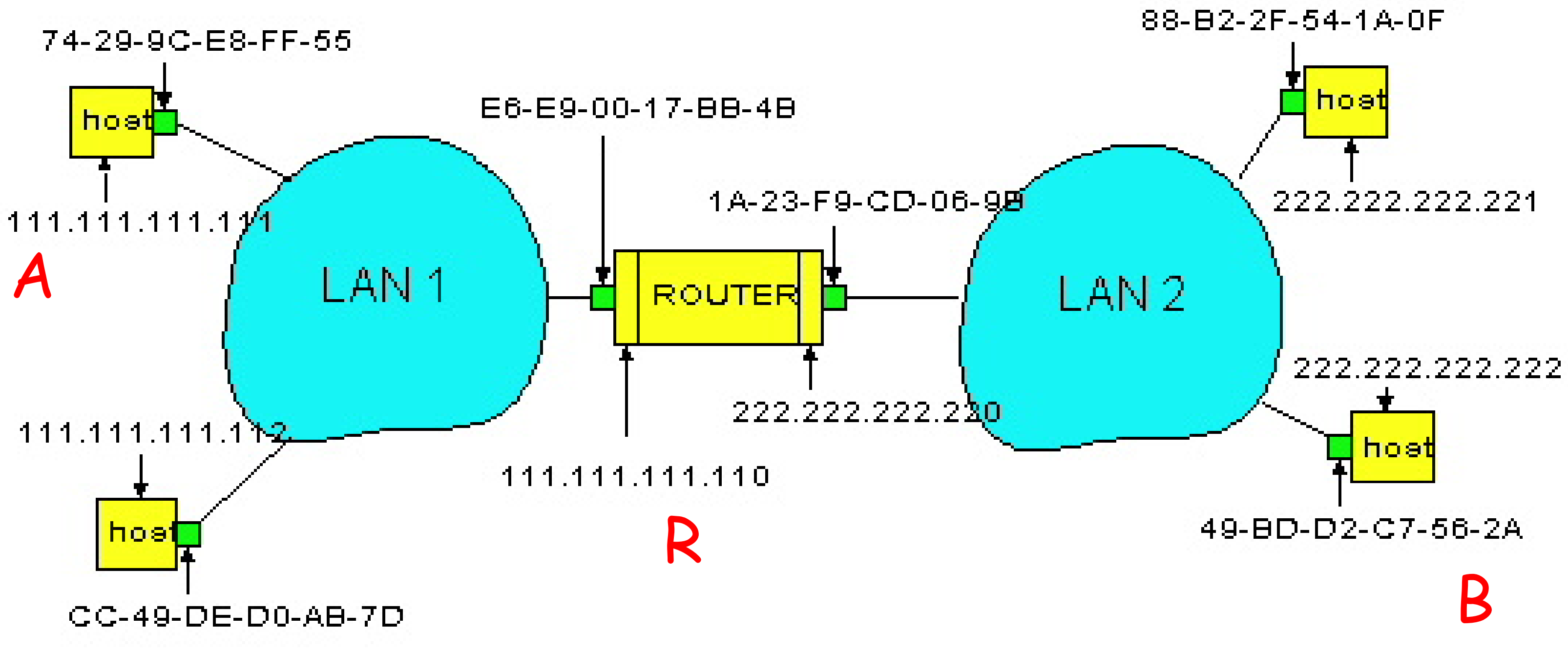
- Adaptors only understand MAC addresses
  - Translate the destination IP address to MAC address
  - Encapsulate the IP packet inside a link-level frame

Address Resolution Protocol Table

- Every node maintains an ARP table
  - (IP address, MAC address) pair
- Consult the table when sending a packet
  - Map destination IP address to destination MAC address
  - Encapsulate and transmit the data packet
- But, what if the IP address is not in the table?
  - Sender broadcasts: “Who has IP address 1.2.3.156?”
  - Receiver responds: “MAC address 58-23-D7-FA-20-B0”
  - Sender caches the result in its ARP table
- No need for network administrator to get involved

Example: A Sending a Packet to B

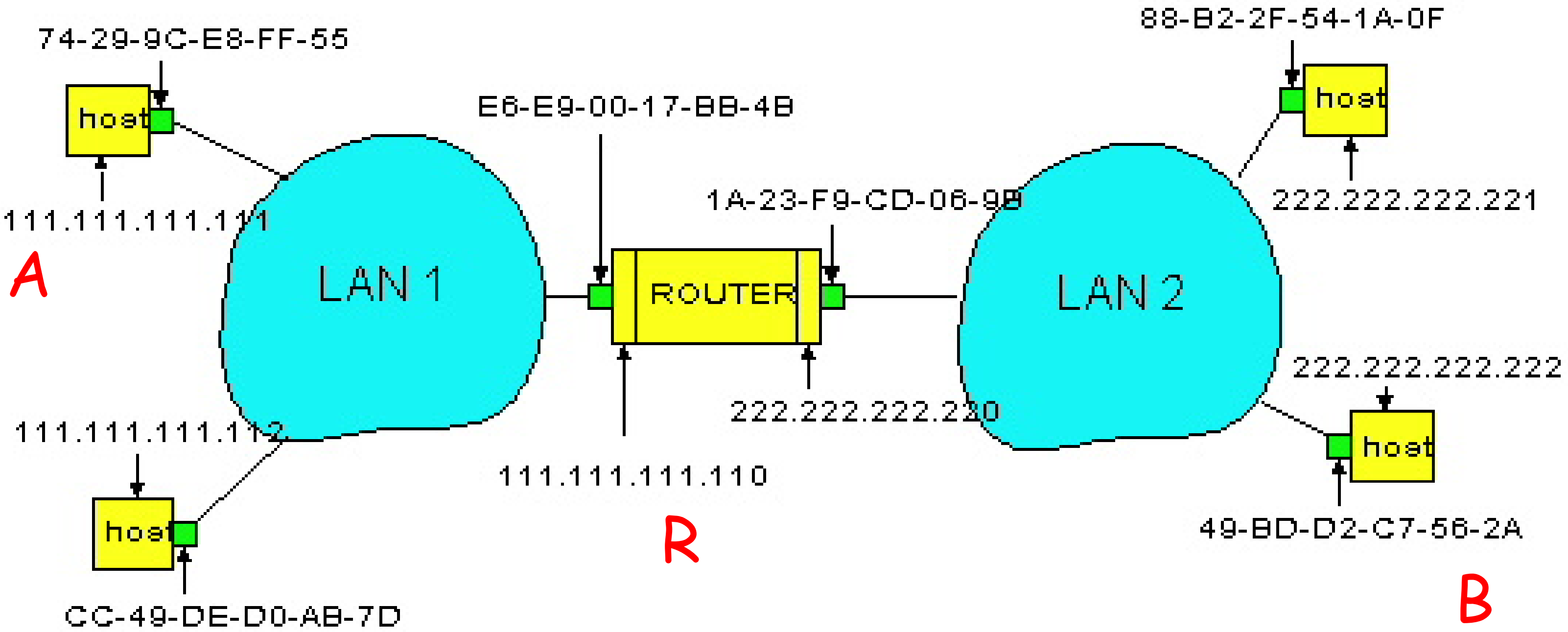
How does host A send an IP packet to host B?



A sends packet to R, and R sends packet to B.

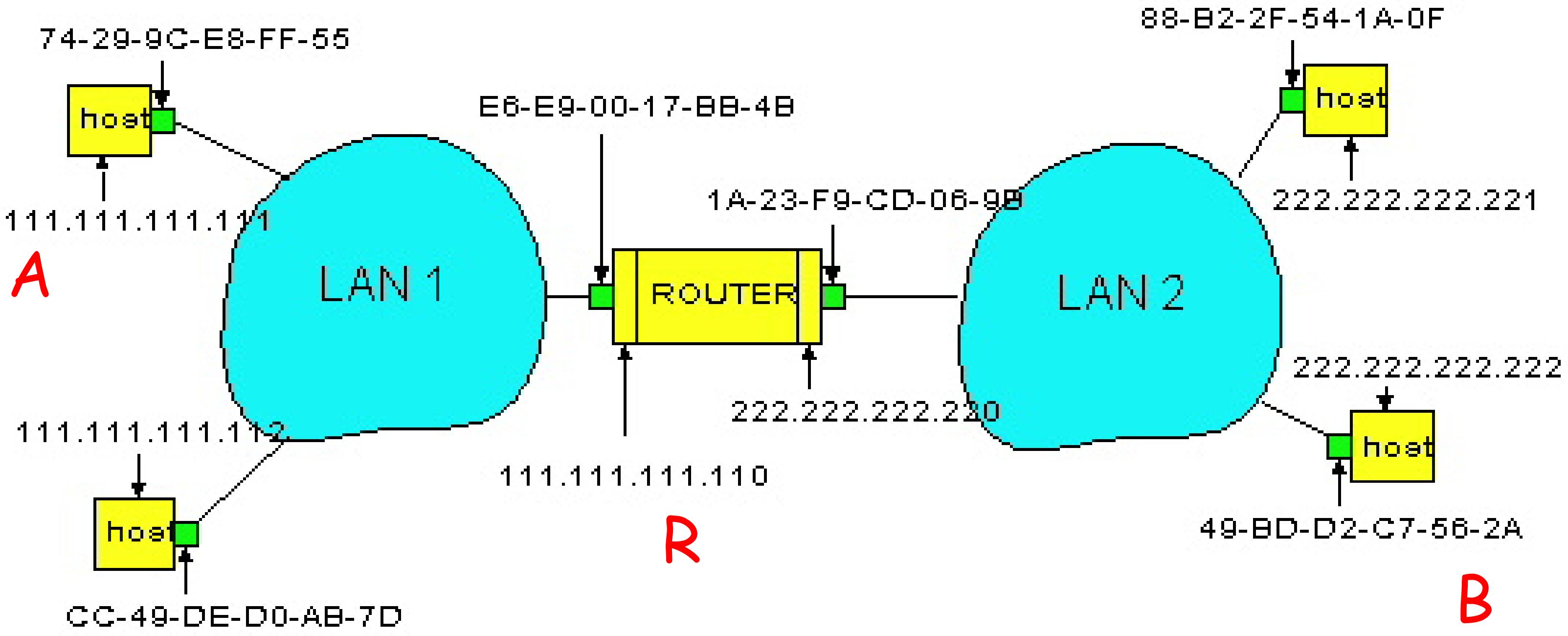
Host A Decides to Send Through R

- Host A constructs an IP packet to send to B
  - Source 111.111.111.111, destination 222.222.222.222
- Host A has a gateway router R
  - Used to reach destinations outside of 111.111.111.0/24
  - Address 111.111.111.110 for R learned via DHCP



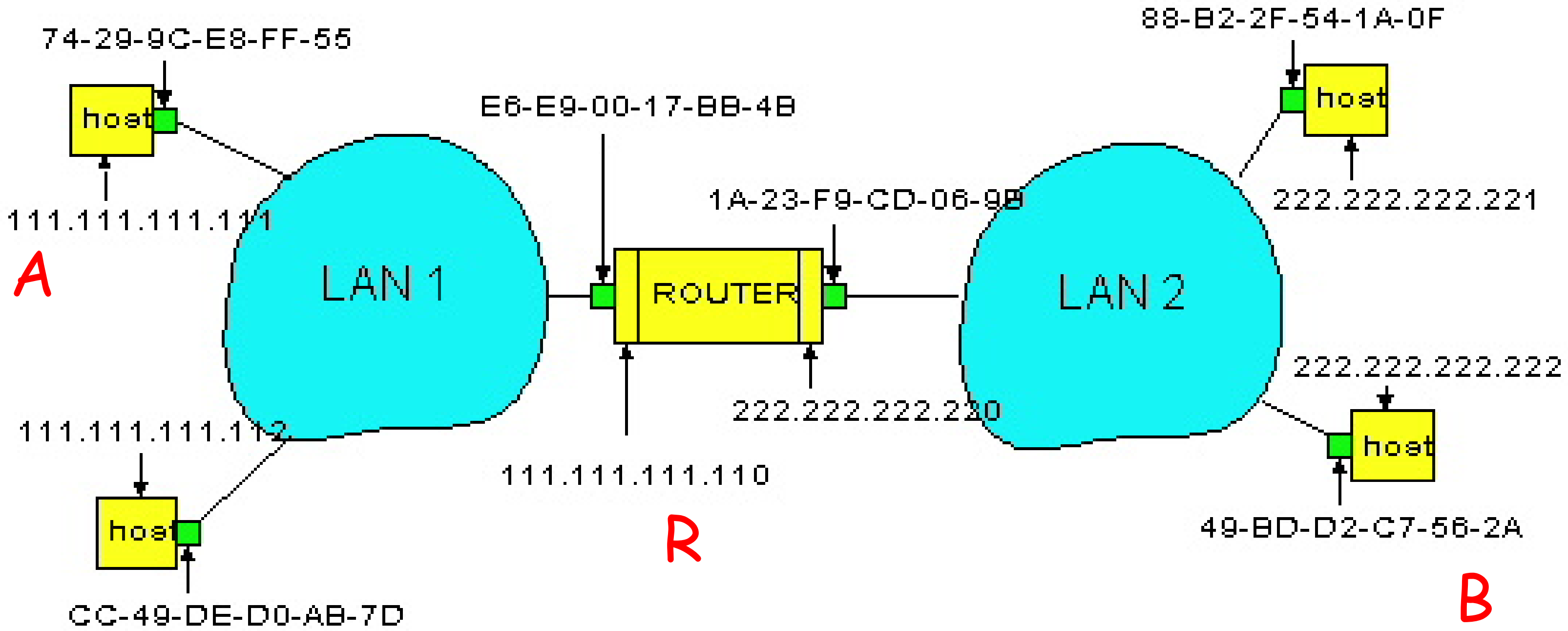
Host A Sends Packet Through R

- Host A learns the MAC address of R's interface
  - ARP request: broadcast request for 111.111.111.110
  - ARP response: R responds with E6-E9-00-17-BB-4B
- Host A encapsulates the packet and sends to R



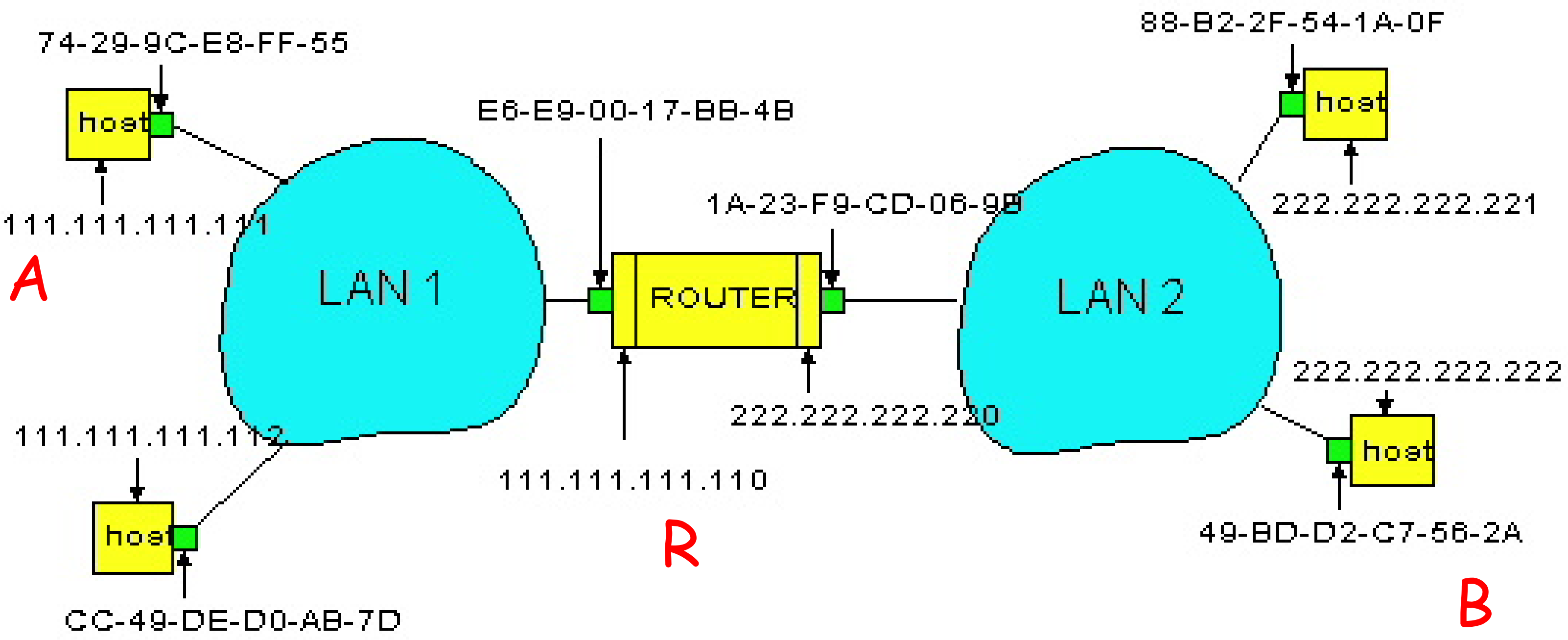
R Decides how to Forward Packet

- Router R’s adaptor receives the packet
  - R extracts the IP packet from the Ethernet frame
  - R sees the IP packet is destined to 222.222.222.222
- Router R consults its forwarding table
  - Packet matches 222.222.222.0/24 via other adaptor



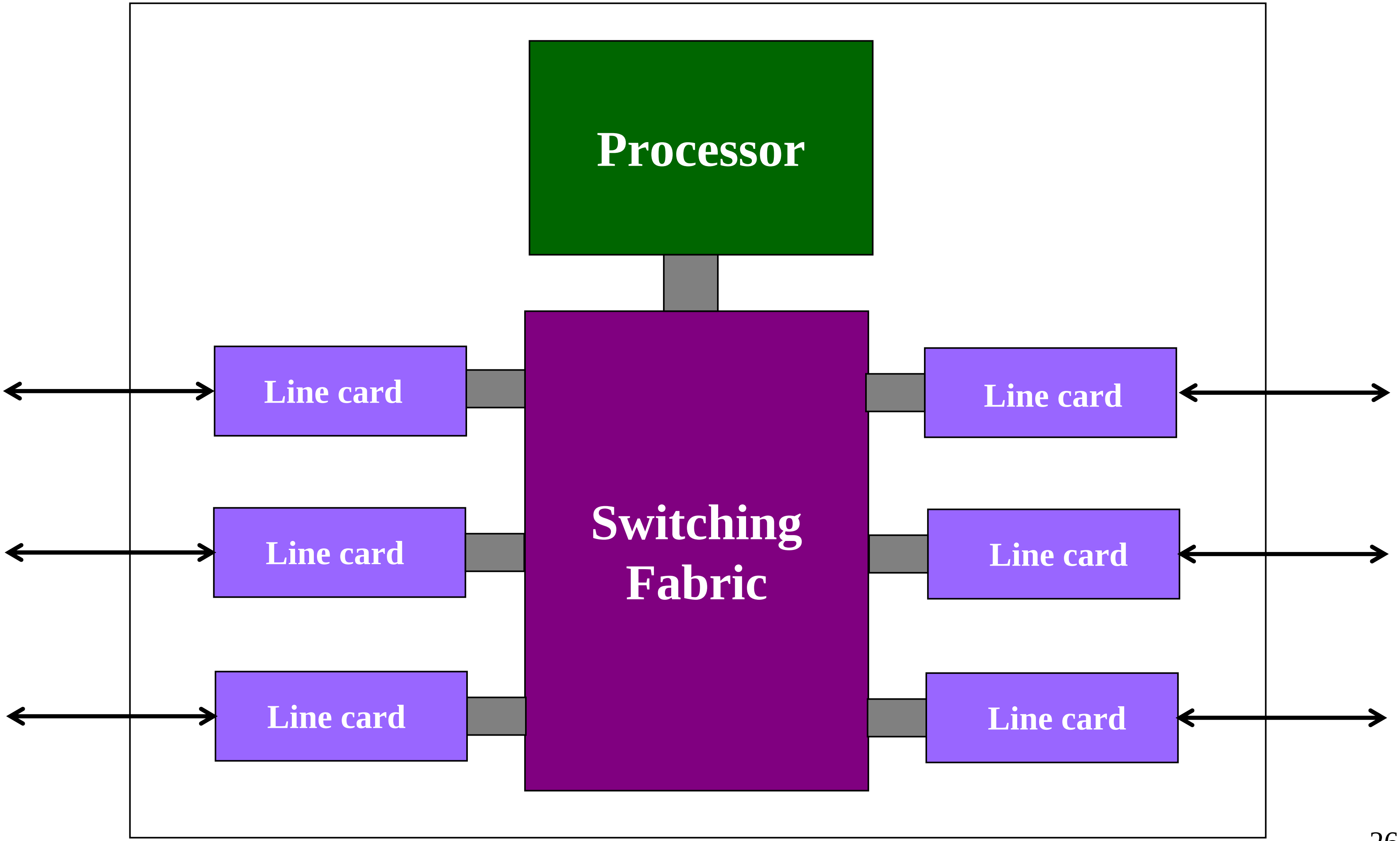
R Sends Packet to B

- Router R's learns the MAC address of host B
  - ARP request: broadcast request for 222.222.222.222
  - ARP response: B responds with 49-BD-D2-C7-56-2A
- Router R encapsulates the packet and sends to B

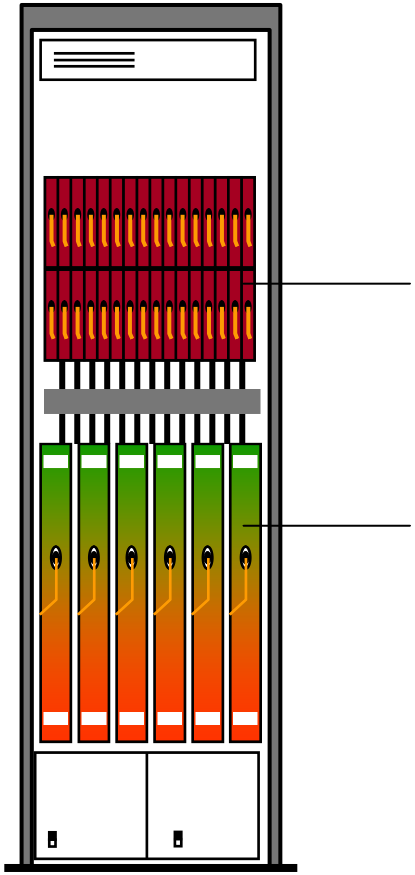


IP Routers

Inside a High-End Router



Router Physical Layout



Crossbar

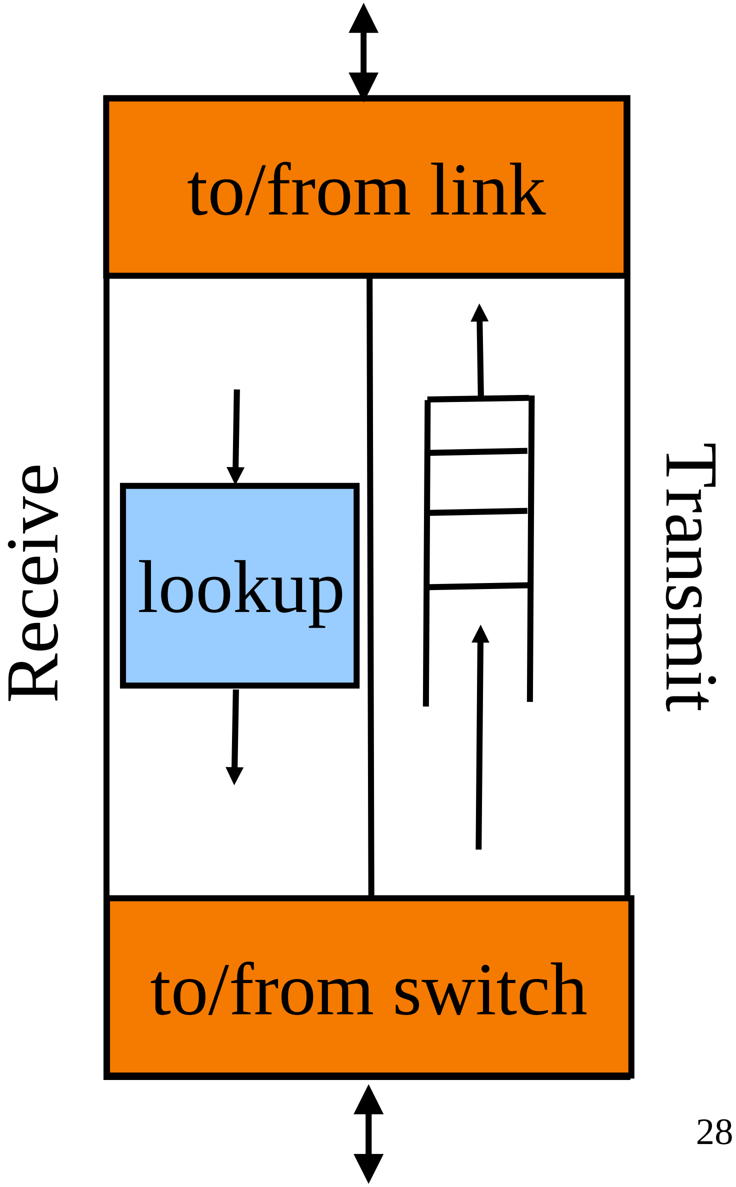
Linecards

Juniper T series

Cisco 12000

Line Cards (Interface Cards, Adaptors)

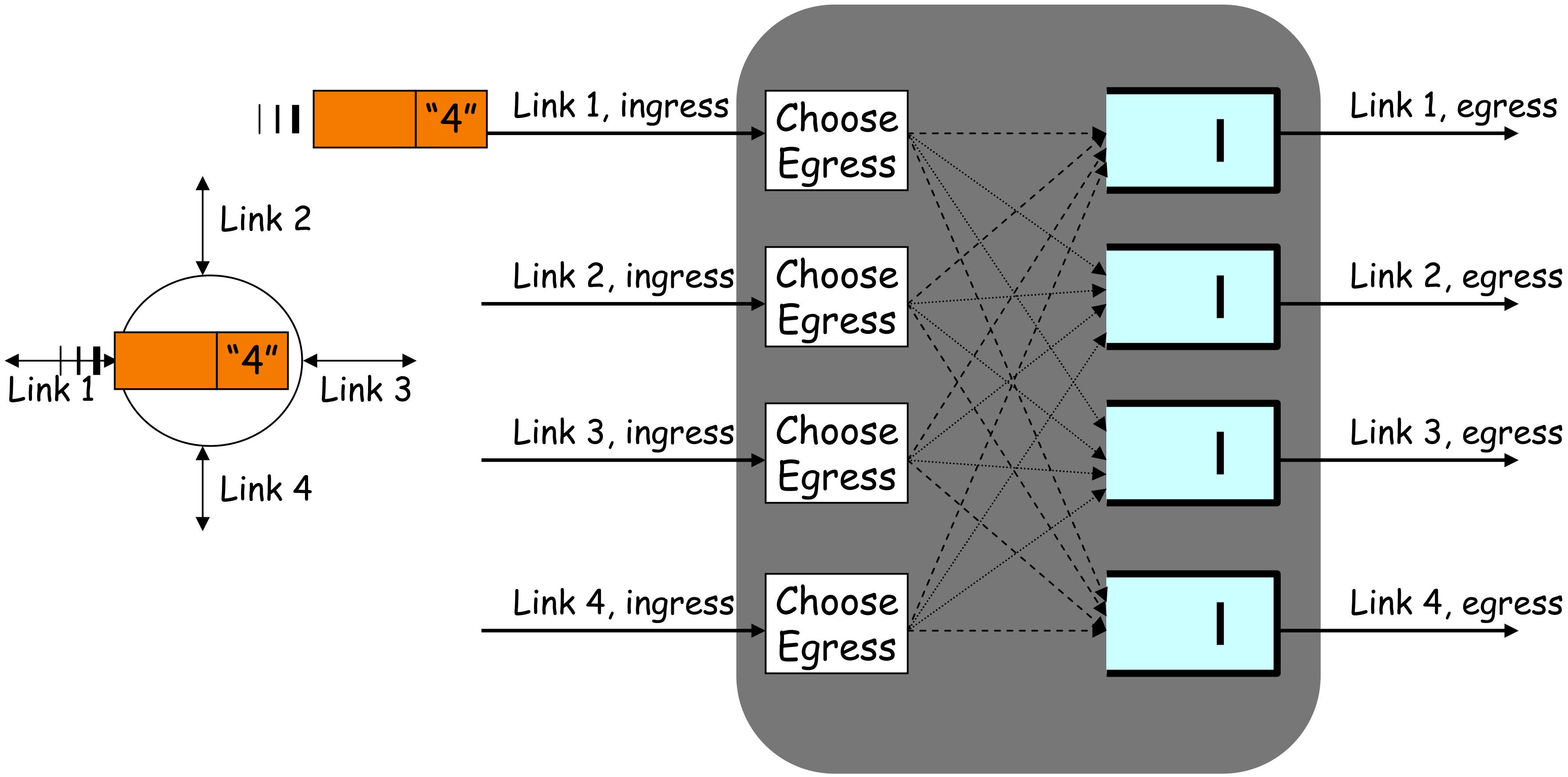
- Interfacing
  - Physical link
  - Switching fabric
- Packet handling
  - Packet forwarding
  - Decrement time-to-live
  - Buffer management
  - Link scheduling
  - Packet filtering
  - Rate limiting
  - Packet marking
  - Measurement



Switching Fabric

- Deliver packet inside the router
  - From incoming interface to outgoing interface
  - A small network in and of itself
- Must operate very quickly
  - Multiple packets going to same outgoing interface
  - Switch scheduling to match inputs to outputs
- Implementation techniques
  - Bus, crossbar, interconnection network, …
  - Running at a faster speed (e.g., 2X) than links
  - Dividing variable-length packets into cells

Packet Switching



Router Processor

- So-called “Loopback” interface
  - –IP address of the CPU on the router
- Control-plane software
  - –Implementation of the routing protocols
  - –Creation of forwarding table for the line cards
- Interface to network administrators
  - –Command-line interface for configuration
  - –Transmission of measurement statistics
- Handling of special data packets
  - –Packets with IP options enabled
  - –Packets with expired Time-To-Live field

Error Reporting

- Examples of errors a router may see
  - Router doesn’t know where to forward a packet
  - Packet’s time-to-live field expires
- Router doesn’t really need to respond
  - Best effort means never having to say you’re sorry
  - So, IP could conceivably just silently drop packets
- But, silent failures are really hard to diagnose
  - IP includes basic feedback about network problems
  - Internet Control Message Protocol (ICMP)

Internet Control Message Protocol

- ICMP runs on top of IP
  - In parallel to TCP and UDP
  - Though still viewed as an integral part of IP
- Diagnostics
  - Triggered when an IP packet encounters a problem
    - E.g., time exceeded or destination unreachable
  - ICMP packet sent back to the source IP address
    - Includes the error information (e.g., type and code)
    - … and an excerpt of the original data packet for identification
  - Source host receives the ICMP packet
    - And inspects the except of the packet (e.g., protocol and ports)
    - … to identify which socket should receive the error

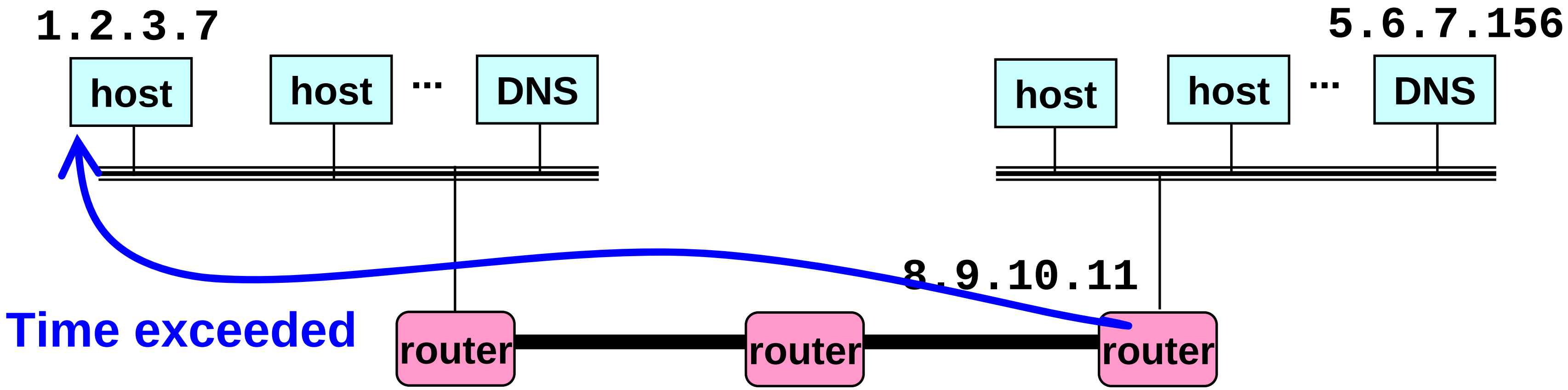
ICMP: Internet Control Message Protocol

- used by hosts & routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer "above" IP:
  - ICMP msgs carried in IP datagrams
- ICMP message: type, code plus first 8 bytes of IP datagram causing error

Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

Example: Time Exceeded

- Host sends an IP packet
  - Each router decrements the time-to-live field
- If time-to-live field reaches 0
  - Router generates an ICMP message
  - Sends a “time exceeded” message back to the source



Traceroute and ICMP

- Source sends series of UDP segments to dest
  - First has TTL =1
  - Second has TTL=2, etc.
  - Unlikely port number
- When nth datagram arrives to nth router:
  - Router discards datagram
  - And sends to source an ICMP message (type 11, code 0)
  - Message includes name of router& IP address

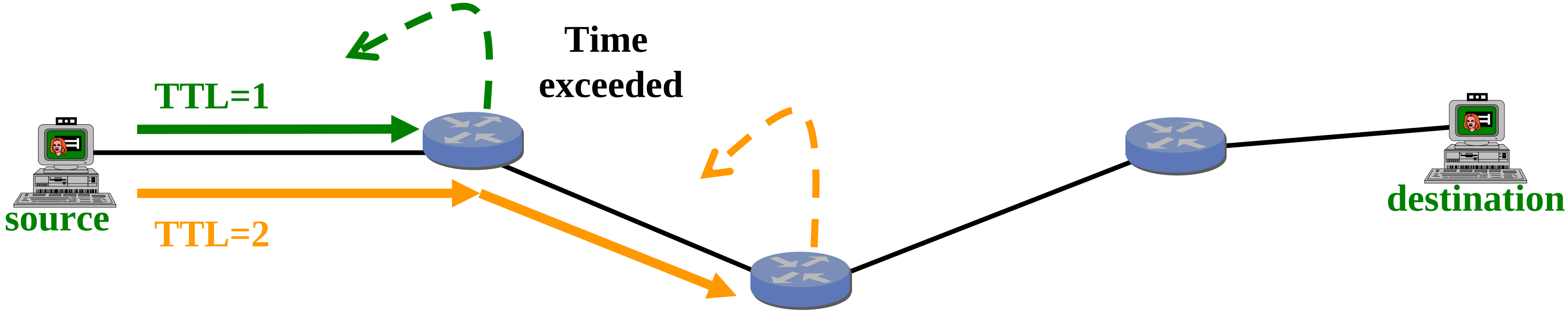
- When ICMP message arrives, source calculates RTT
- Traceroute does this 3 times

Stopping criterion

- UDP segment eventually arrives at destination host
- Destination returns ICMP “host unreachable” packet (type 3, code 3)
- When source gets this ICMP, stops.

Traceroute: Exploiting “Time Exceeded”

- Time-To-Live field in IP packet header
  - Source sends a packet with a TTL of *n*
  - Each router along the path decrements the TTL
  - “TTL exceeded” sent when TTL reaches *0*
- Traceroute tool exploits this TTL behavior



Send packets with TTL=1, 2, … and record source of “time exceeded” message

Ping: Echo and Reply

- ICMP includes a simple “echo” function
  - Sending node sends an ICMP “echo” message
  - Receiving node sends an ICMP “echo reply”
- Ping tool
  - Tests the connectivity with a remote host
  - … by sending regularly spaced echo commands
  - … and measuring the delay until receiving the reply
- Pinging a host
  - “ping www.cs.princeton.edu” or “ping 12.157.34.212”
  - Used to test if a machine is reachable and alive
  - (However, some nodes have ICMP disabled… ☹)

Conclusion

- Important control functions
  - Bootstrapping
  - Error reporting and monitoring
- Internet control protocols
  - Dynamic Host Configuration Protocol (DHCP)
  - Address Resolution Protocol (ARP)
  - Internet Control Message Protocol (ICMP)
- Components of an IP router
  - Line cards, switching fabric, and route processor

Shortest-Path Routing

Introduction to Data Networks

2008.4

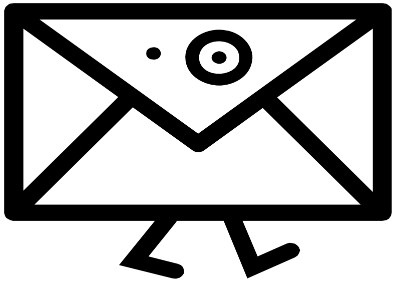
Goals of Today’s Lecture

- Path selection
  - Minimum-hop and shortest-path routing
  - Dijkstra and Bellman-Ford algorithms
- Topology change
  - Using beacons to detect topology changes
  - Propagating topology or path information
- Routing protocols
  - Link state: Open Shortest Path First
  - Distance vector: Routing Information Protocol

What is Routing?

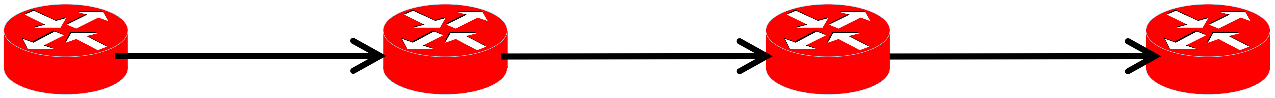
- A famous quotation from RFC 791

“A *name* indicates what we seek.
An *address* indicates where it is.
A *route* indicates how we get there.”
-- Jon Postel



Forwarding vs. Routing

- Forwarding: data plane
  - –Directing a data packet to an outgoing link
  - –Individual router *using* a forwarding table
- Routing: control plane
  - –Computing paths the packets will follow
  - –Routers talking amongst themselves
  - –Individual router *creating* a forwarding table

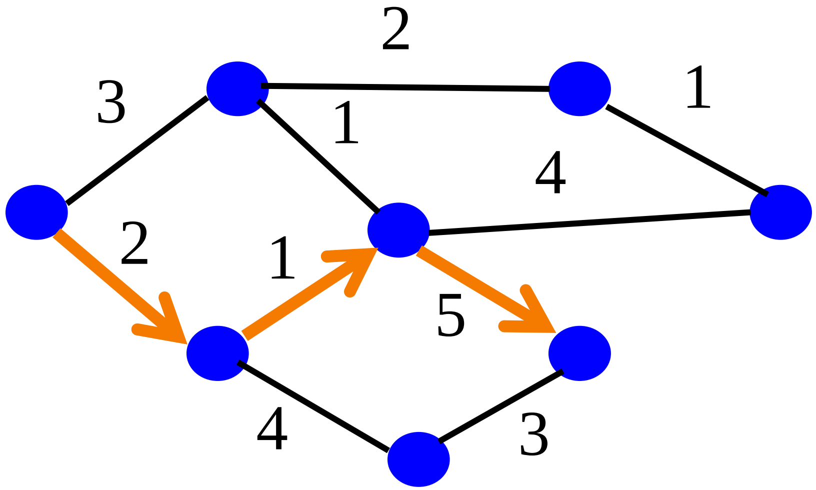


Why Does Routing Matter?

- End-to-end performance
  - Quality of the path affects user performance
  - Propagation delay, throughput, and packet loss
- Use of network resources
  - Balance of the traffic over the routers and links
  - Avoiding congestion by directing traffic to lightly-loaded links
- Transient disruptions during changes
  - Failures, maintenance, and load balancing
  - Limiting packet loss and delay during changes

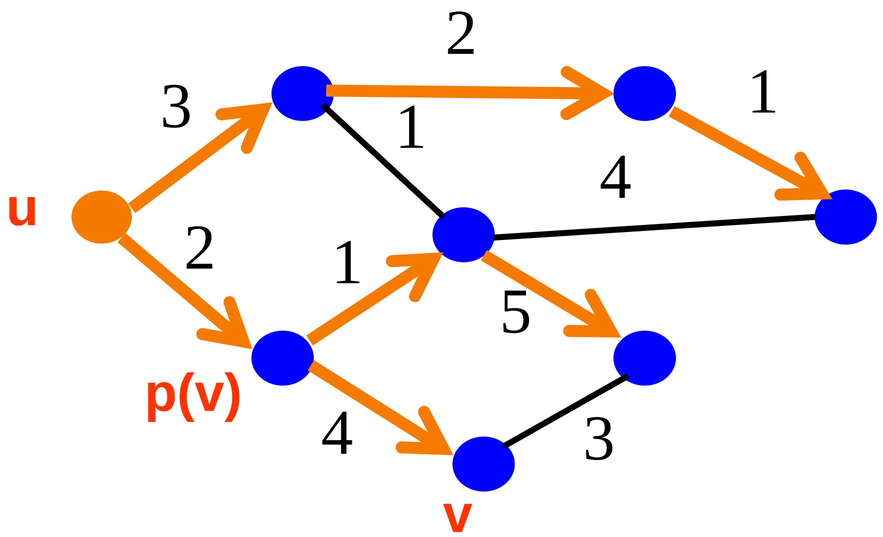
Shortest-Path Routing

- Path-selection model
  - –Destination-based
  - –Load-insensitive (e.g., static link weights)
  - –Minimum hop count or sum of link weights



Shortest-Path Problem

- Given: network topology with link costs
  - **c(x,y)**: link cost from node x to node y
  - Infinity if x and y are not direct neighbors
- Compute: least-cost paths to all nodes
  - From a given source u to all other nodes
  - **p(v)**: predecessor node along path from source to v



Routing Algorithm classification

Global or decentralized information?

Global:

- all routers have complete topology, link cost info
- "link state" algorithms

Decentralized:

- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- "distance vector" algorithms

Static or dynamic?

Static:

- routes change slowly over time

Dynamic:

- routes change more quickly
  - periodic update
  - in response to link cost changes

Dijkstra’s Shortest-Path Algorithm

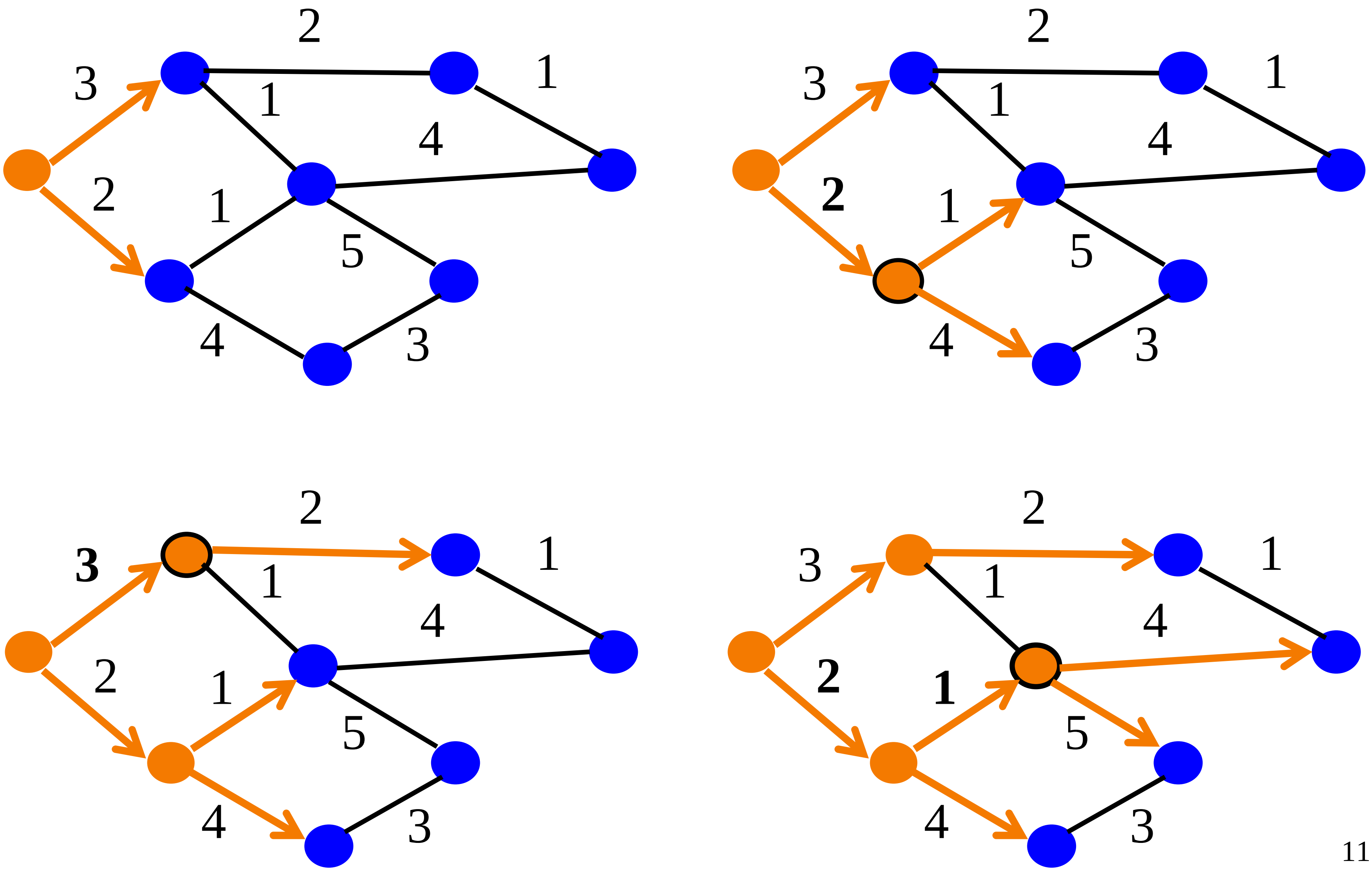
- Iterative algorithm
  - After k iterations, know least-cost path to k nodes
- **S**: nodes whose least-cost path definitively known
  - Initially, **S = {u}** where u is the source node
  - Add one node to S in each iteration
- **D(v)**: current cost of path from source to node v
  - Initially, **D(v) = c(u,v)** for all nodes v adjacent to u
  - … and **D(v) = ∞** for all other nodes v
  - Continually update D(v) as shorter paths are learned

Dijsktra’s Algorithm

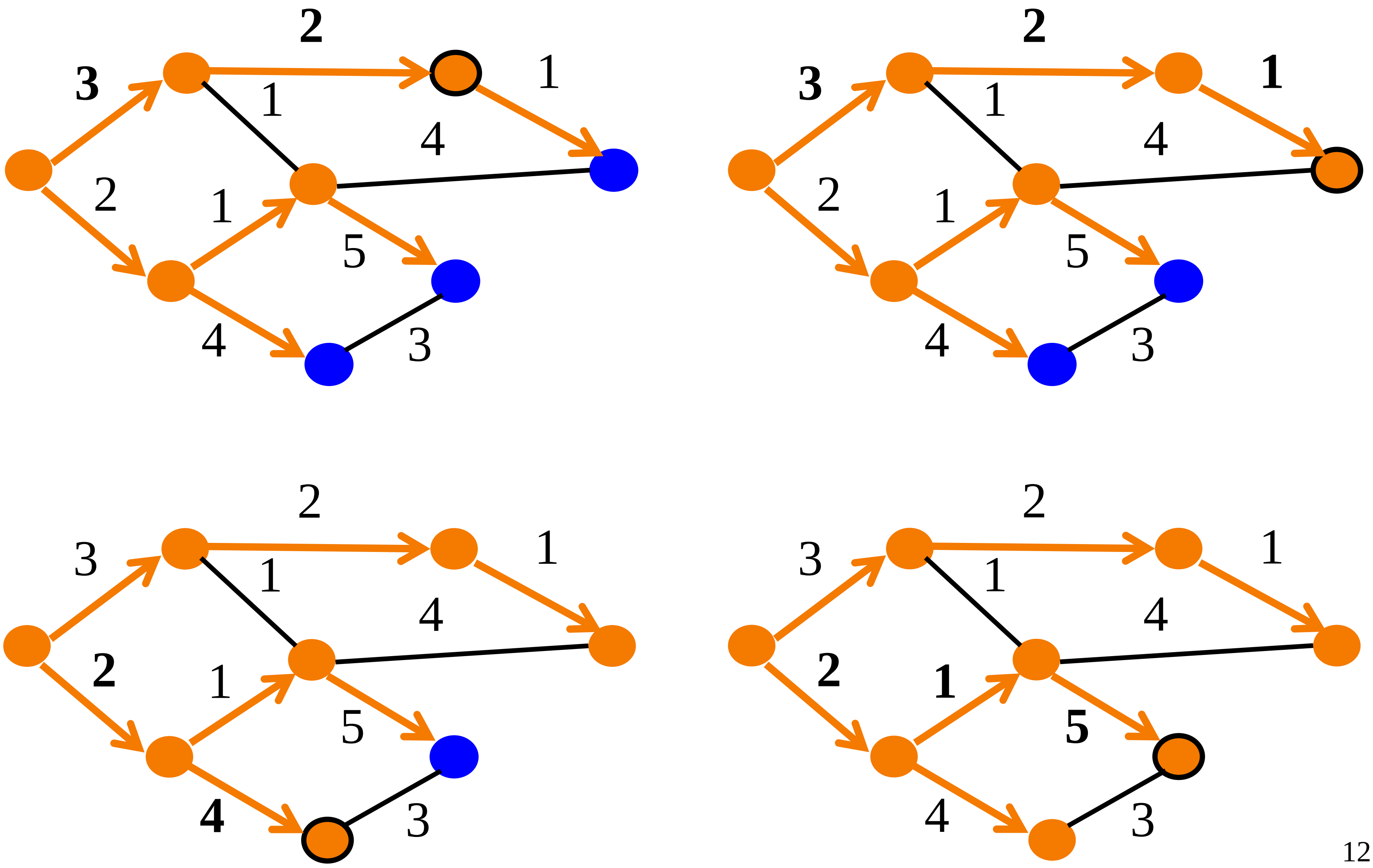
```
Initialization:
  S = {u}
  for all nodes v
    if v adjacent to u {
        D(v) = c(u,v)
    else D(v) = ∞

Loop
  find w not in S with the smallest D(w)
  add w to S
  update D(v) for all v adjacent to w and not in S:
      D(v) = min{D(v), D(w) + c(w,v)}
until all nodes in S
```

Dijkstra's Algorithm Example

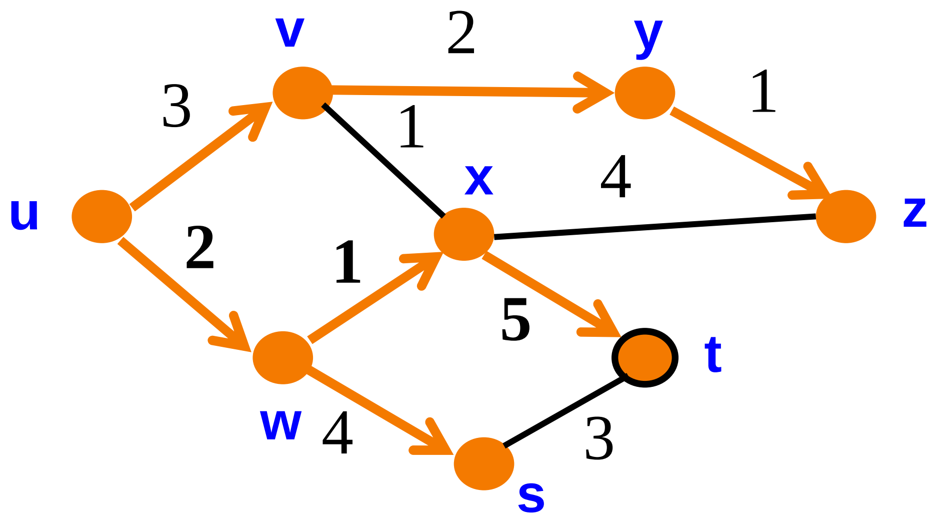


Dijkstra’s Algorithm Example



Shortest-Path Tree

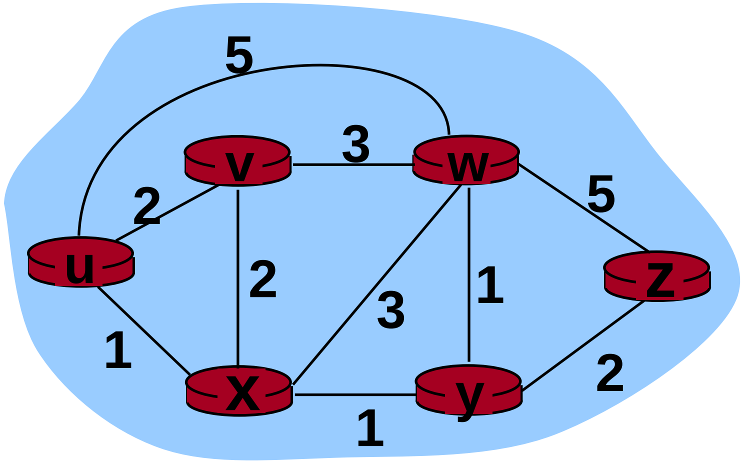
- Shortest-path tree from u
- Forwarding table at u



	link
v	(u,v)
w	(u,w)
x	(u,w)
y	(u,v)
z	(u,v)
s	(u,w)
t	(u,w)

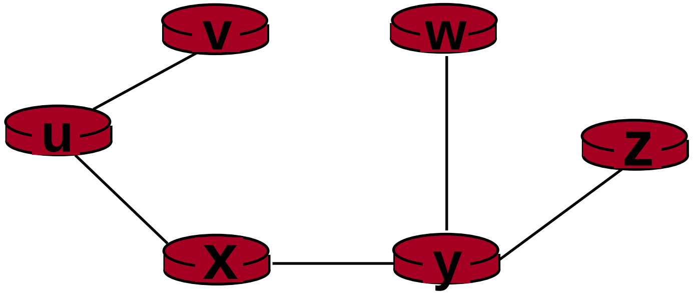
Dijkstra’s algorithm: Another example

Step	S	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



Dijkstra’s algorithm: Another example

Resulting shortest-path tree from u:



Resulting forwarding table in u:

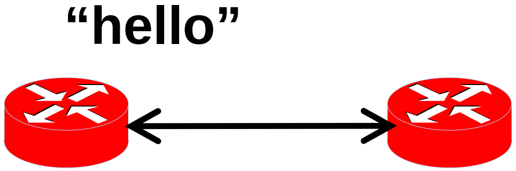
destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

Link-State Routing

- Each router keeps track of its incident links
  - Whether the link is up or down
  - The cost on the link
- Each router broadcasts the link state
  - To give every router a complete view of the graph
- Each router runs Dijkstra’s algorithm
  - To compute the shortest paths
  - … and construct the forwarding table
- Example protocols
  - Open Shortest Path First (OSPF)
  - Intermediate System – Intermediate System (IS-IS)

Detecting Topology Changes

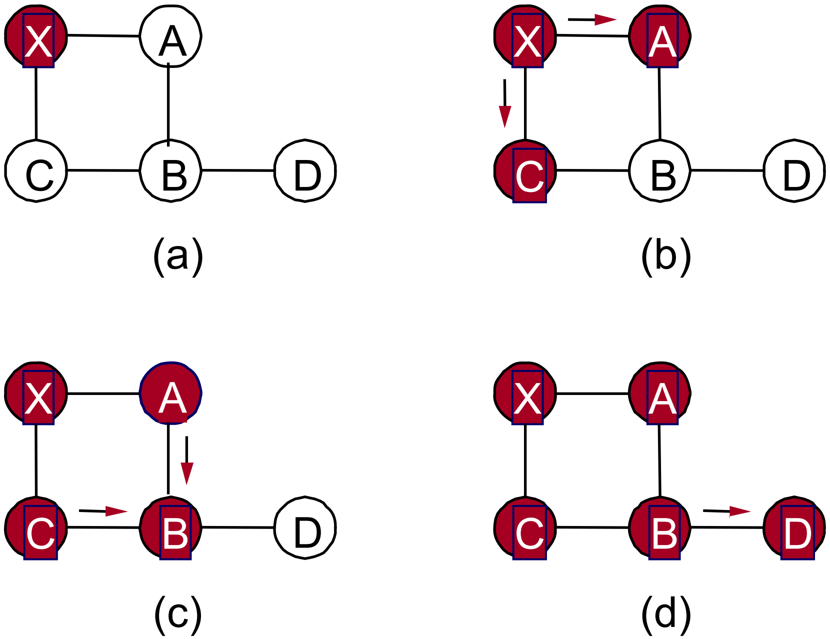
- Beaconing
  - Periodic “hello” messages in both directions
  - Detect a failure after a few missed “hellos”



- Performance trade-offs
  - Detection speed
  - Overhead on link bandwidth and CPU
  - Likelihood of false detection

Broadcasting the Link State

- Flooding
  - –Node sends link-state information out its links
  - –And then the next node sends out all of its links
  - –… except the one where the information arrived



Broadcasting the Link State

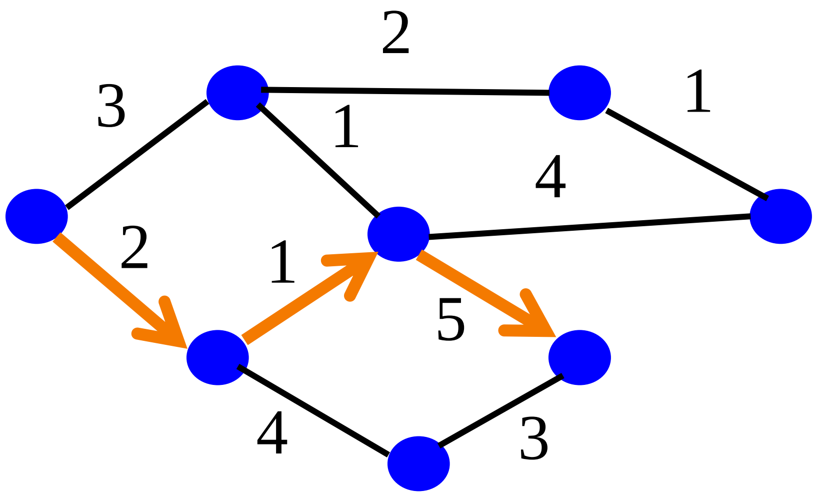
- Reliable flooding
  - Ensure all nodes receive link-state information
  - … and that they use the latest version
- Challenges
  - Packet loss
  - Out-of-order arrival
- Solutions
  - Acknowledgments and retransmissions
  - Sequence numbers
  - Time-to-live for each packet

When to Initiate Flooding

- Topology change
  - Link or node failure
  - Link or node recovery
- Configuration change
  - Link cost change
- Periodically
  - Refresh the link-state information
  - Typically (say) 30 minutes
  - Corrects for possible corruption of the data

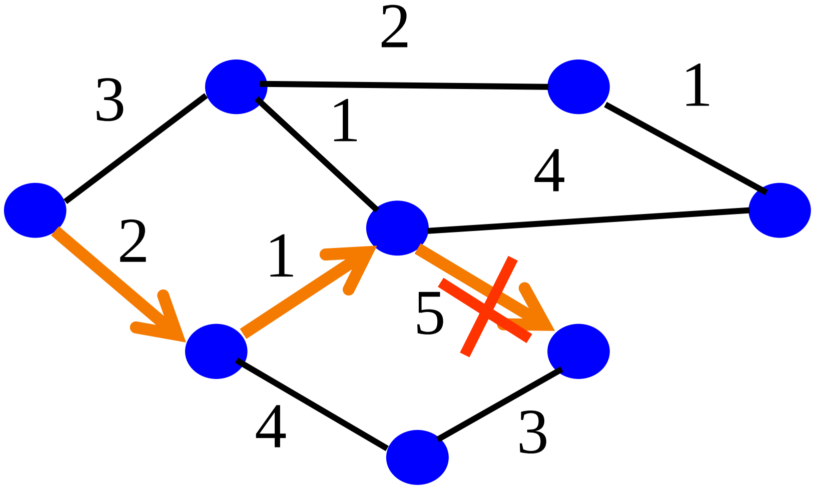
Convergence

- Getting consistent routing information to all nodes
  - E.g., all nodes having the same link-state database
- Consistent forwarding after convergence
  - All nodes have the same link-state database
  - All nodes forward packets on shortest paths
  - The next router on the path forwards to the next hop



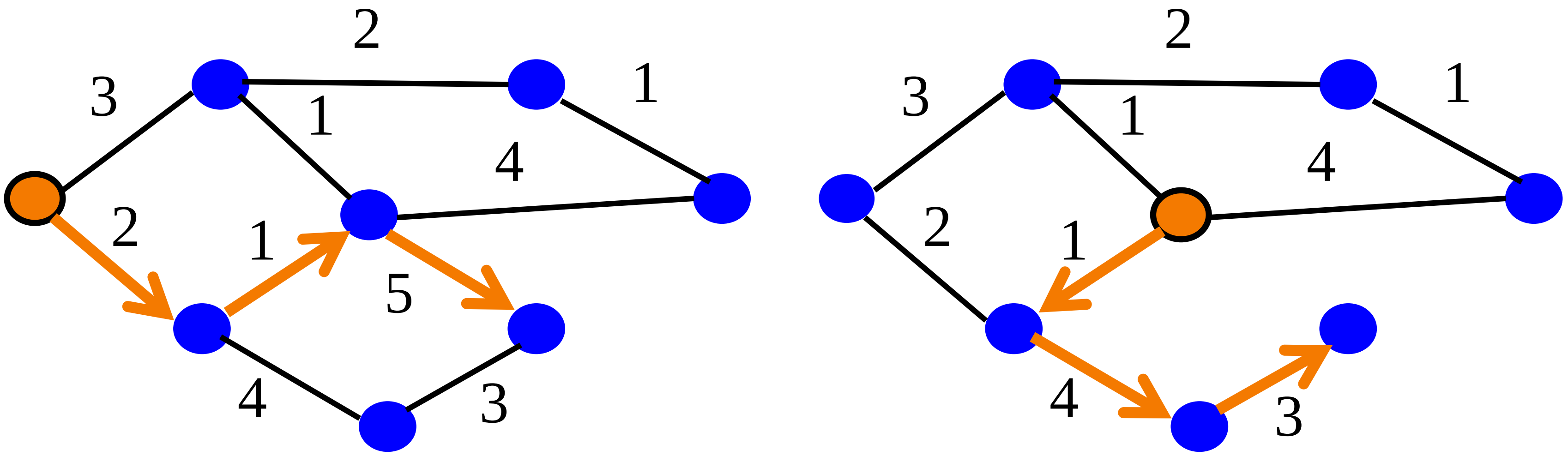
Transient Disruptions

- Detection delay
  - A node does not detect a failed link immediately
  - … and forwards data packets into a “blackhole”
  - Depends on timeout for detecting lost hellos



Transient Disruptions

- Inconsistent link-state database
  - Some routers know about failure before others
  - The shortest paths are no longer consistent
  - Can cause transient forwarding loops



Convergence Delay

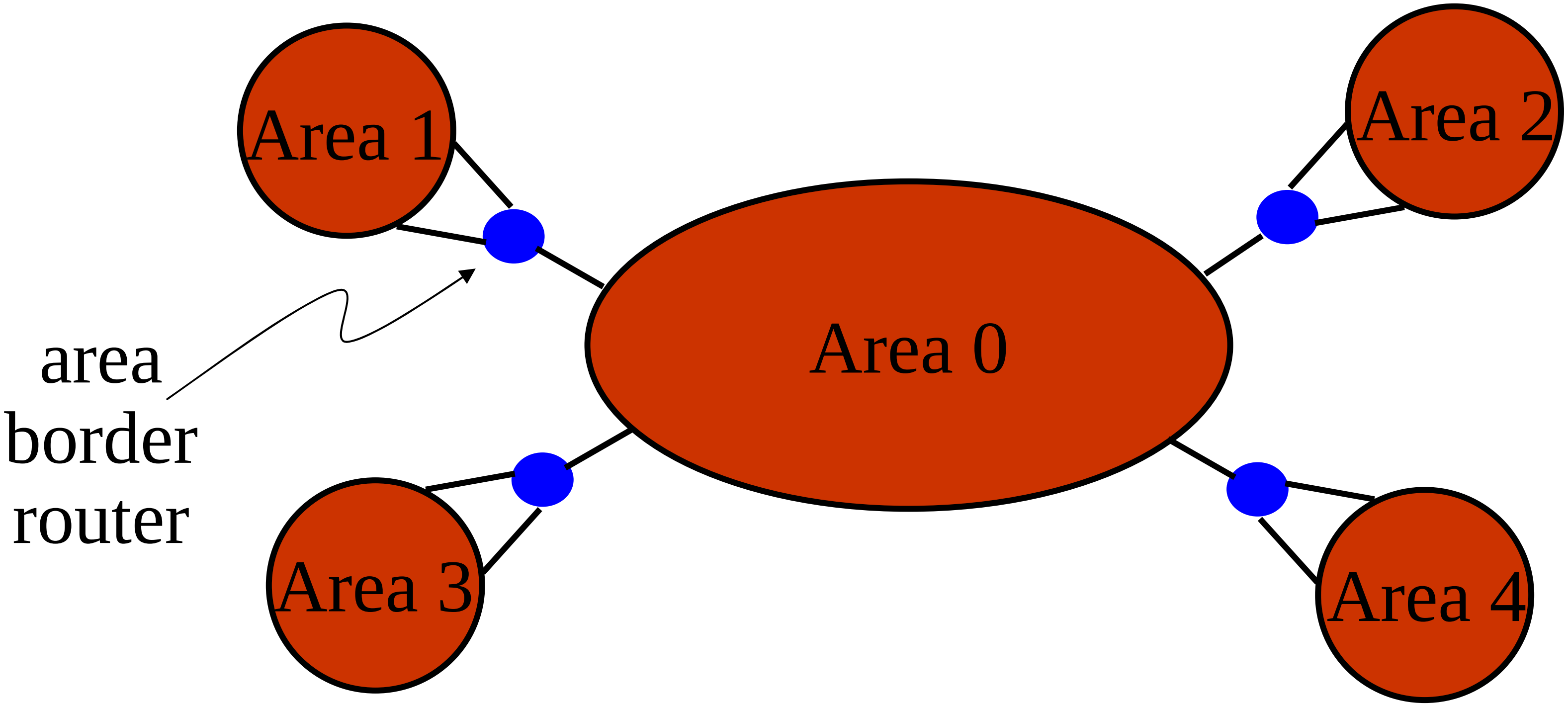
- Sources of convergence delay
  - Detection latency
  - Flooding of link-state information
  - Shortest-path computation
  - Creating the forwarding table
- Performance during convergence period
  - Lost packets due to blackholes and TTL expiry
  - Looping packets consuming resources
  - Out-of-order packets reaching the destination
- Very bad for VoIP, online gaming, and video

Reducing Convergence Delay

- Faster detection
  - Smaller hello timers
  - Link-layer technologies that can detect failures
- Faster flooding
  - Flooding immediately
  - Sending link-state packets with high-priority
- Faster computation
  - Faster processors on the routers
  - Incremental Dijkstra algorithm
- Faster forwarding-table update
  - Data structures supporting incremental updates

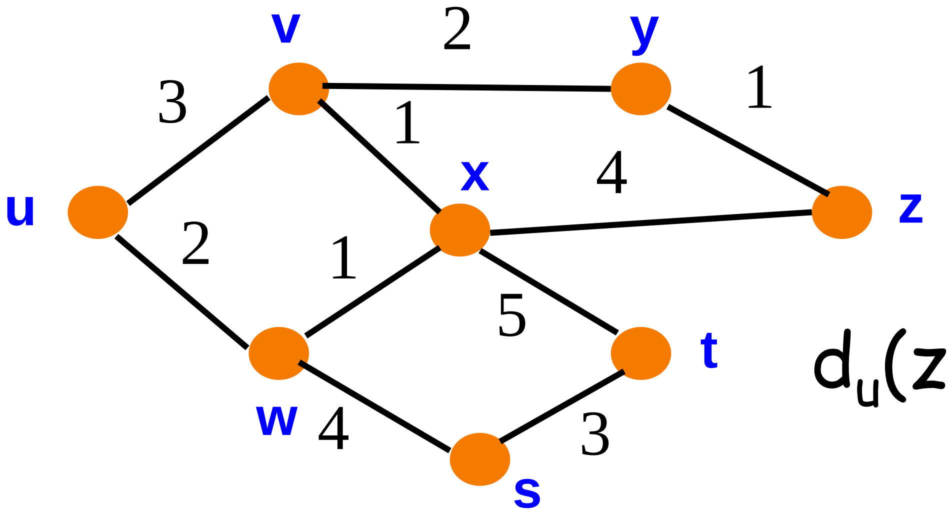
Scaling Link-State Routing

- Overhead of link-state routing
  - Flooding link-state packets throughout the network
  - Running Dijkstra’s shortest-path algorithm
- Introducing hierarchy through “areas”



Bellman-Ford Algorithm

- Define distances at each node x
  - – $d_x(y)$ = cost of least-cost path from x to y
- Update distances based on neighbors
  - – $d_x(y) = \min \{c(x,v) + d_v(y)\}$ over all neighbors v



$$d_u(z) = \min\{c(u,v) + d_v(z),\ c(u,w) + d_w(z)\}$$

Distance Vector Algorithm

- $c(x,v)$ = cost for direct link from x to v
  - Node x maintains costs of direct links $c(x,v)$
- $D_x(y)$ = estimate of least cost from x to y
  - Node x maintains distance vector $\mathbf{D}_x = [D_x(y): y \in N]$
- Node x maintains its neighbors' distance vectors
  - For each neighbor v, x maintains $\mathbf{D}_v = [D_v(y): y \in N]$
- Each node v periodically sends D_v to its neighbors
  - And neighbors update their own distance vectors
  - $D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\}$ for each node $y \in N$
- Over time, the distance vector D_x converges

Distance Vector Algorithm

Iterative, asynchronous:

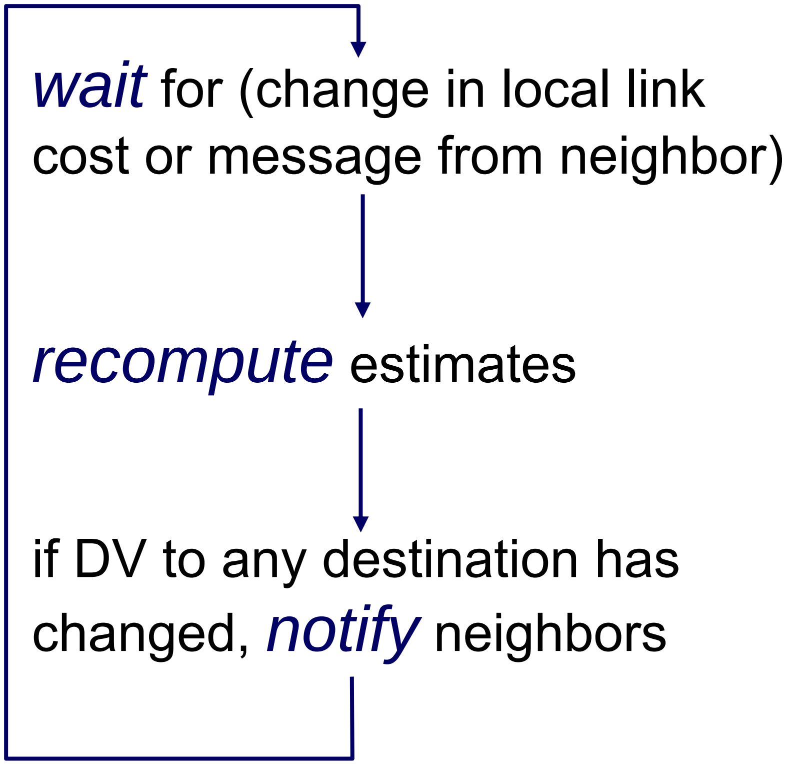
each local iteration caused by:

- Local link cost change
- Distance vector update message from neighbor

Distributed:

- Each node notifies neighbors *only* when its DV changes
- Neighbors then notify their neighbors if necessary

Each node:



wait for (change in local link cost or message from neighbor)

recompute estimates

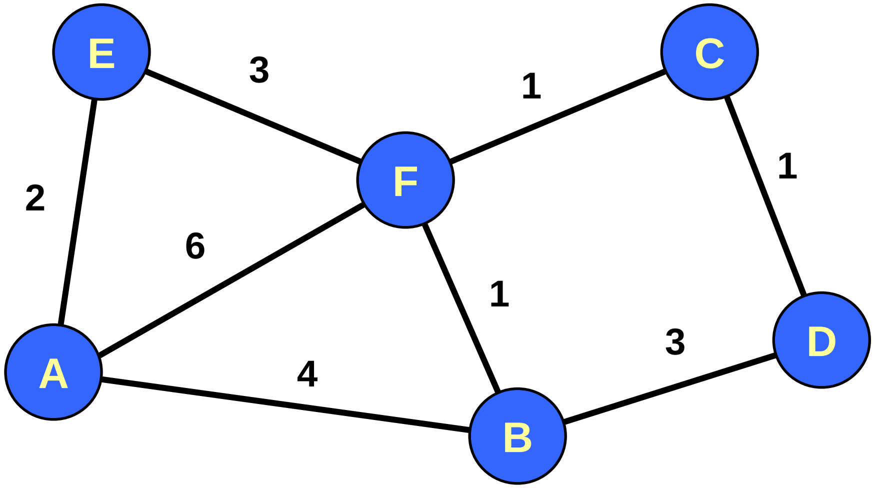
if DV to any destination has changed, *notify* neighbors

Distance Vector Example: Step 0

Optimum 1-hop paths

Table for A			Table for B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A
B	4	B	B	0	B
C	∞	–	C	∞	–
D	∞	–	D	3	D
E	2	E	E	∞	–
F	6	F	F	1	F

Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	∞	–	A	∞	–	A	2	A	A	6	A
B	∞	–	B	3	B	B	∞	–	B	1	B
C	0	C	C	1	C	C	∞	–	C	1	C
D	1	D	D	0	D	D	∞	–	D	∞	–
E	∞	–	E	∞	–	E	0	E	E	3	E
F	1	F	F	∞	–	F	3	F	F	0	F



Distance Vector Example: Step 2

Optimum 2-hop paths

Table for A			Table for B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A
B	4	B	B	0	B
C	7	F	C	2	F
D	7	B	D	3	D
E	2	E	E	4	F
F	5	E	F	1	F

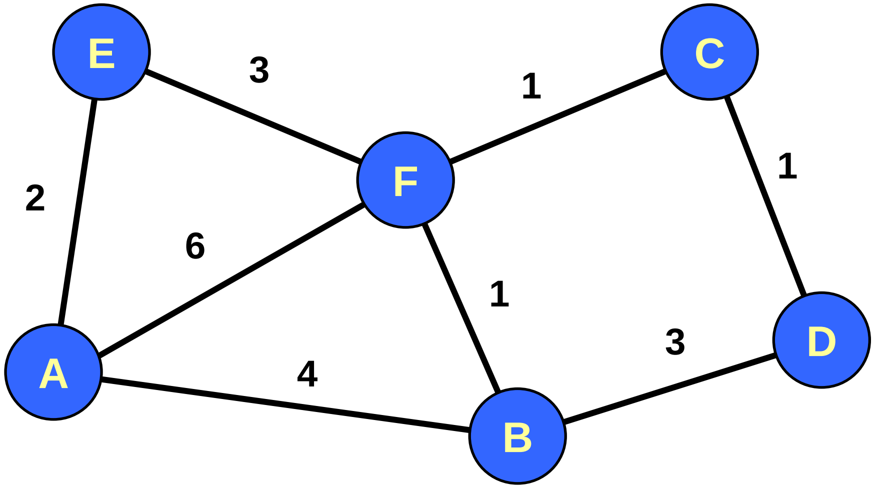


Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	7	F	A	7	B	A	2	A	A	5	B
B	2	F	B	3	B	B	4	F	B	1	B
C	0	C	C	1	C	C	4	F	C	1	C
D	1	D	D	0	D	D	∞	–	D	2	C
E	4	F	E	∞	–	E	0	E	E	3	E
F	1	F	F	2	C	F	3	F	F	0	F

Distance Vector Example: Step 3

Optimum 3-hop paths

Table for A			Table for B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A
B	4	B	B	0	B
C	6	E	C	2	F
D	7	B	D	3	D
E	2	E	E	4	F
F	5	E	F	1	F

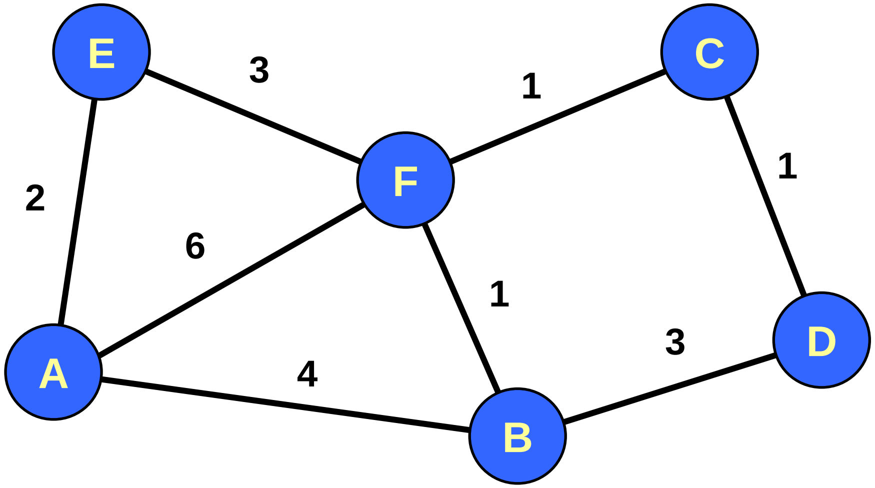
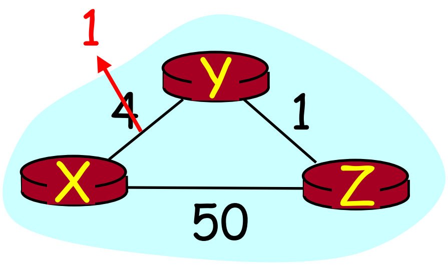


Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	6	F	A	7	B	A	2	A	A	5	B
B	2	F	B	3	B	B	4	F	B	1	B
C	0	C	C	1	C	C	4	F	C	1	C
D	1	D	D	0	D	D	5	F	D	2	C
E	4	F	E	5	C	E	0	E	E	3	E
F	1	F	F	2	C	F	3	F	F	0	F

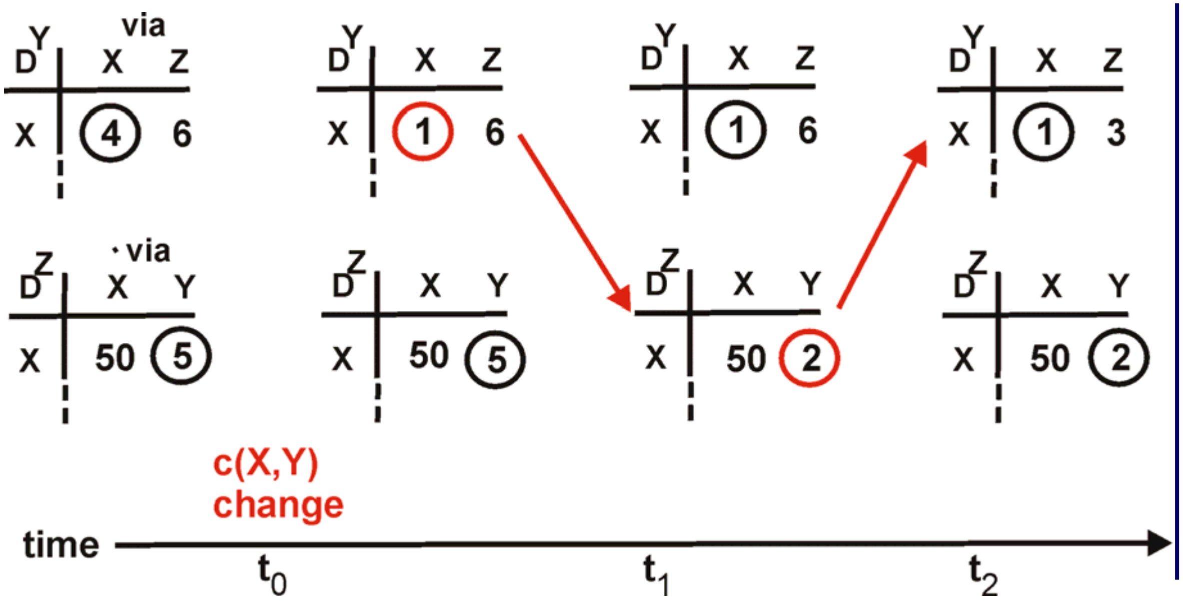
Distance Vector: Link Cost Changes

Link cost changes:

- Node detects local link cost change
- Updates the distance table
- If cost change in least cost path, notify neighbors



"good news travels fast"



D^Y	X	Z
X	4	6

D^Y	X	Z
X	1	6

D^Y	X	Z
X	1	6

D^Y	X	Z
X	1	3

D^Z	X	Y
X	50	5

D^Z	X	Y
X	50	5

D^Z	X	Y
X	50	2

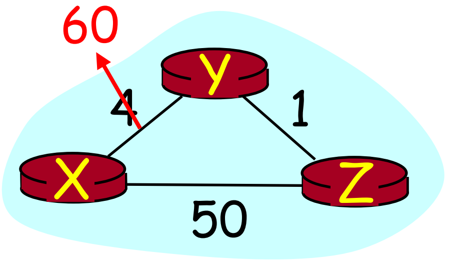
D^Z	X	Y
X	50	2

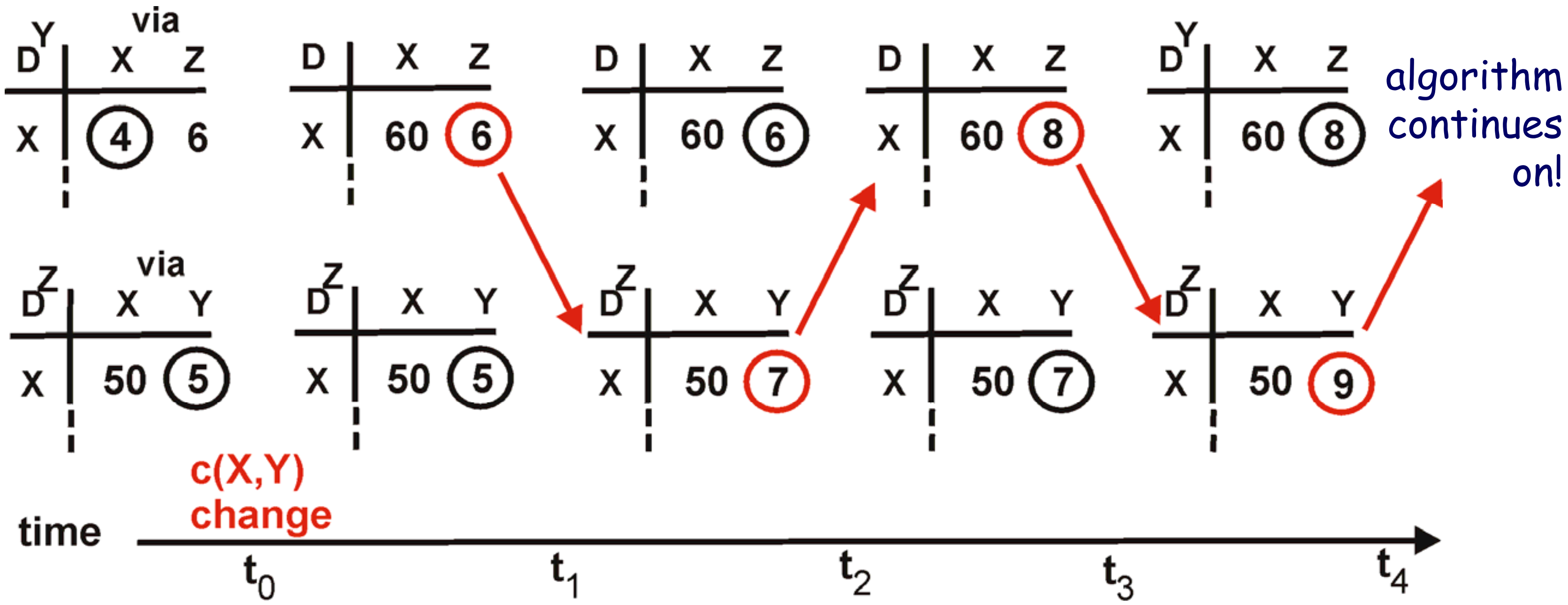
algorithm terminates

Distance Vector: Link Cost Changes

Link cost changes:

- Good news travels fast
- Bad news travels slow - “count to infinity” problem!





via

D^Y	X	Z
X	4	6

D	X	Z
X	60	6

D	X	Z
X	60	6

D	X	Z
X	60	8

D^Y	X	Z
X	60	8

algorithm continues on!

via

D^Z	X	Y
X	50	5

D^Z	X	Y
X	50	5

D^Z	X	Y
X	50	7

D^Z	X	Y
X	50	7

D^Z	X	Y
X	50	9

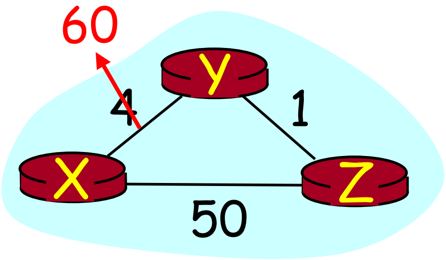
c(X,Y) change

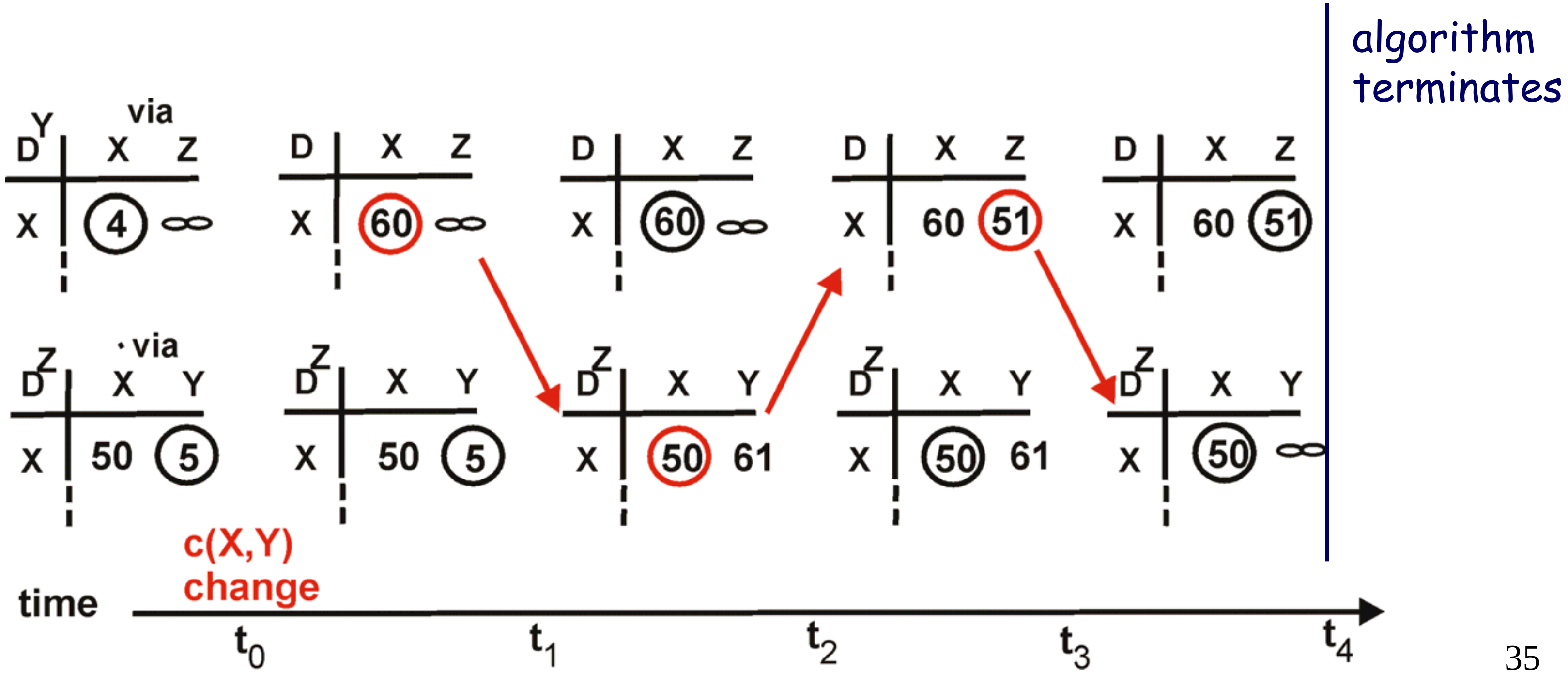
time t_0 t_1 t_2 t_3 t_4

Distance Vector: Poison Reverse

If Z routes through Y to get to X :

- Z tells Y its (Z’s) distance to X is infinite (so Y won’t route to X via Z)
- Still, can have problems when more than 2 routers are involved





D^Y (via):

D	X	Z
X	4	∞

D	X	Z
X	60	∞

D	X	Z
X	60	∞

D	X	Z
X	60	51

D	X	Z
X	60	51

D^Z (via):

D	X	Y
X	50	5

D	X	Y
X	50	5

D	X	Y
X	50	61

D	X	Y
X	50	61

D	X	Y
X	50	∞

Routing Information Protocol (RIP)

- Distance vector protocol
  - Nodes send distance vectors every 30 seconds
  - … or, when an update causes a change in routing
- Link costs in RIP
  - All links have cost 1
  - Valid distances of 1 through 15
  - … with 16 representing infinity
  - Small "infinity" → smaller "counting to infinity" problem
- RIP is limited to fairly small networks
  - E.g., used in the Princeton campus network

Comparison of LS and DV algorithms

Message complexity

- LS: with n nodes, E links, O(nE) messages sent
- DV: exchange between neighbors only
  - Convergence time varies

Speed of Convergence

- LS: O(n^2) algorithm requires O(nE) messages
- DV: convergence time varies
  - May be routing loops
  - Count-to-infinity problem

Robustness: what happens if router malfunctions?

LS:

- Node can advertise incorrect *link* cost
- Each node computes only its *own* table

DV:

- DV node can advertise incorrect *path* cost
- Each node's table used by others (error propagates)

Conclusions

- Routing is a distributed algorithm
  - React to changes in the topology
  - Compute the shortest paths
- Two main shortest-path algorithms
  - Dijkstra → link-state routing (e.g., OSPF and IS-IS)
  - Bellman-Ford → distance vector routing (e.g., RIP)
- Convergence process
  - Changing from one topology to another
  - Transient periods of inconsistency across routers
- Next time: policy-based path-vector routing

Policy-Based Path-Vector Routing

Introduction to Data Networks

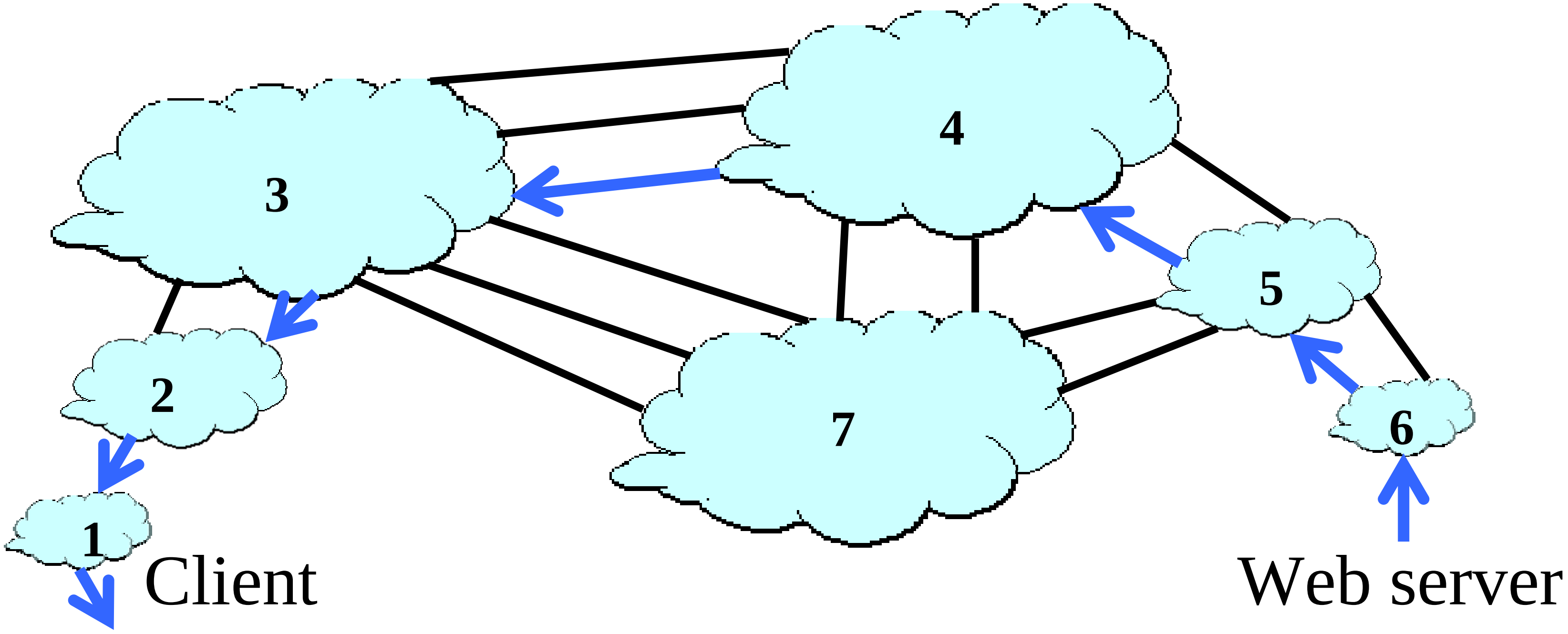
2008.4

Goals of Today’s Lecture

- Challenges of interdomain routing
  - Scale, privacy, and policy
  - Limitations of link-state and distance-vector routing
- Path-vector routing
  - Faster loop detection than distance-vector routing
  - More flexibility than shortest-path routing
- Border Gateway Protocol (BGP)
  - Incremental, prefix-based, path-vector protocol
  - Programmable import and export policies
  - Multi-step decision process for selecting “best” route
- Multiple routers within an AS
- BGP convergence delay

Interdomain Routing

- AS-level topology
  - –Destinations are IP prefixes (e.g., 12.0.0.0/8)
  - –Nodes are Autonomous Systems (ASes)
  - –Links are connections & business relationships

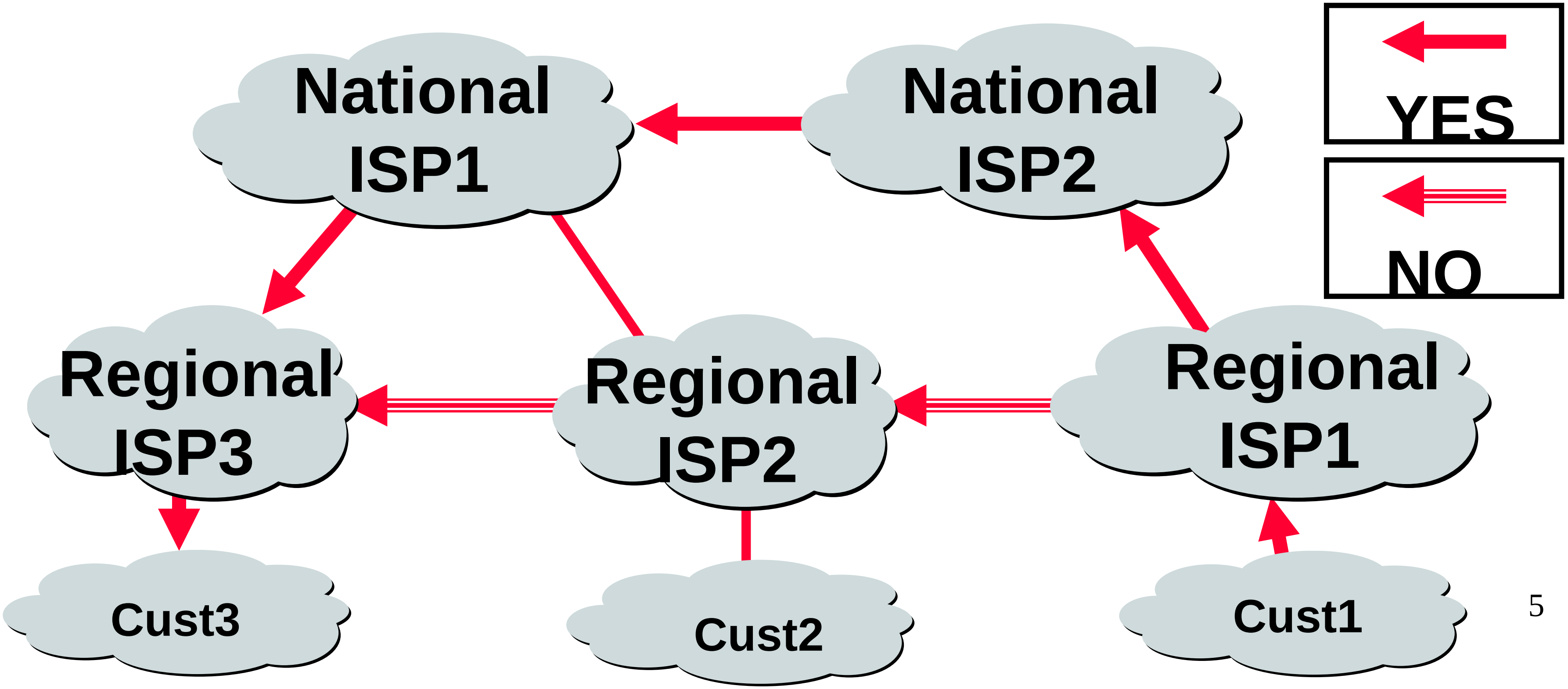


Challenges for Interdomain Routing

- Scale
  - Prefixes: 150,000-200,000, and growing
  - ASes: 20,000 visible ones, and growing
  - AS paths and routers: at least in the millions…
- Privacy
  - ASes don't want to divulge internal topologies
  - … or their business relationships with neighbors
- Policy
  - No Internet-wide notion of a link cost metric
  - Need control over where you send traffic
  - … and who can send traffic through you

Shortest-Path Routing is Restrictive

- All traffic must travel on shortest paths
- All nodes need common notion of link costs
- Incompatible with commercial relationships



Link-State Routing is Problematic

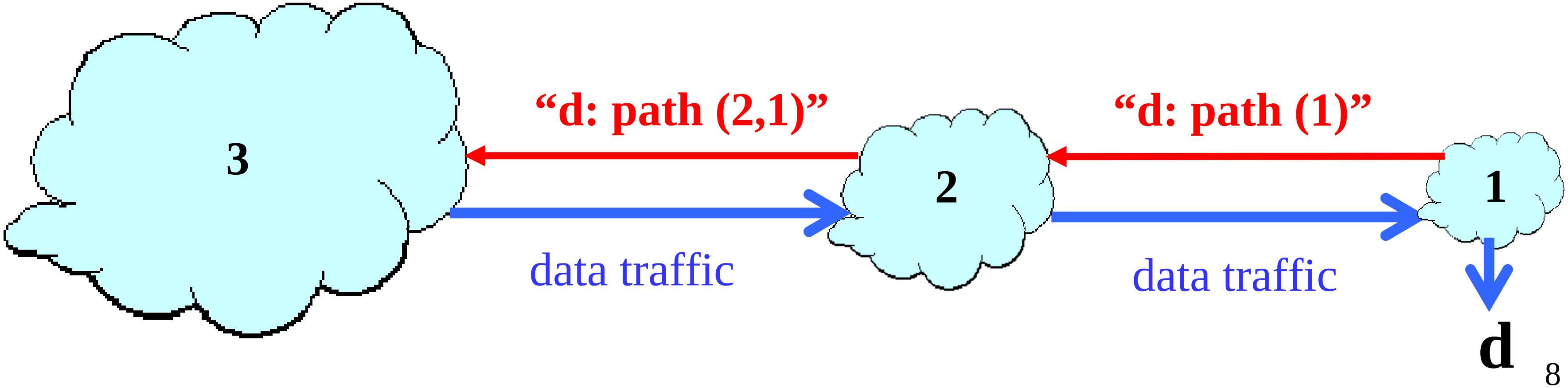
- Topology information is flooded
  - High bandwidth and storage overhead
  - Forces nodes to divulge sensitive information
- Entire path computed locally per node
  - High processing overhead in a large network
- Minimizes some notion of total distance
  - Works only if policy is shared and uniform
- Typically used only inside an AS
  - E.g., OSPF and IS-IS

Distance Vector is on the Right Track

- Advantages
  - Hides details of the network topology
  - Nodes determine only “next hop” toward the dest
- Disadvantages
  - Minimizes some notion of total distance, which is difficult in an interdomain setting
  - Slow convergence due to the counting-to-infinity problem (“bad news travels slowly”)
- Idea: extend the notion of a distance vector

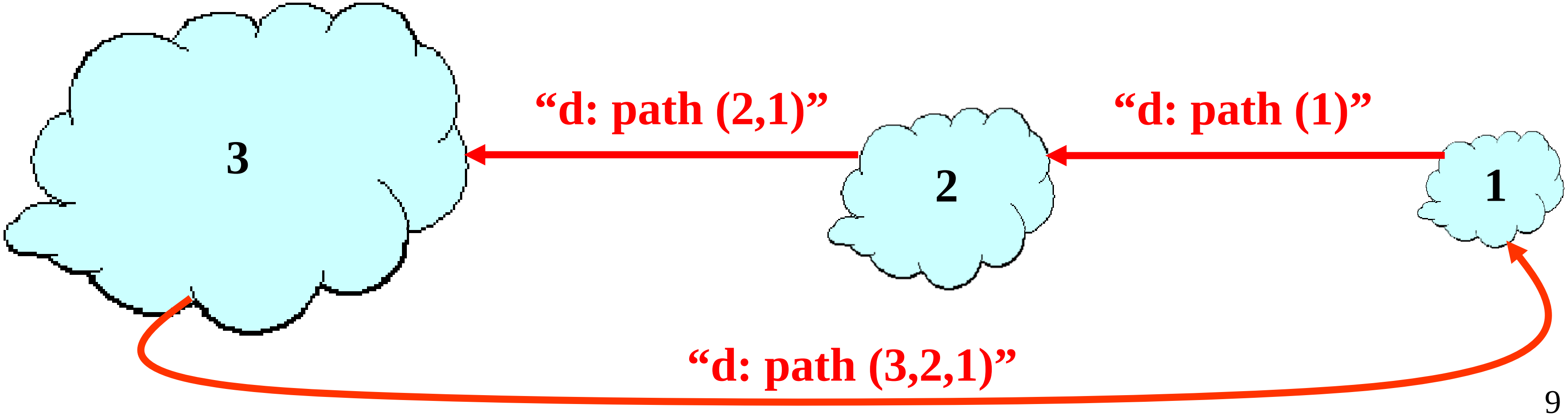
Path-Vector Routing

- Extension of distance-vector routing
  - Support flexible routing policies
  - Avoid count-to-infinity problem
- Key idea: advertise the entire path
  - Distance vector: send *distance metric* per dest d
  - Path vector: send the *entire path* for each dest d



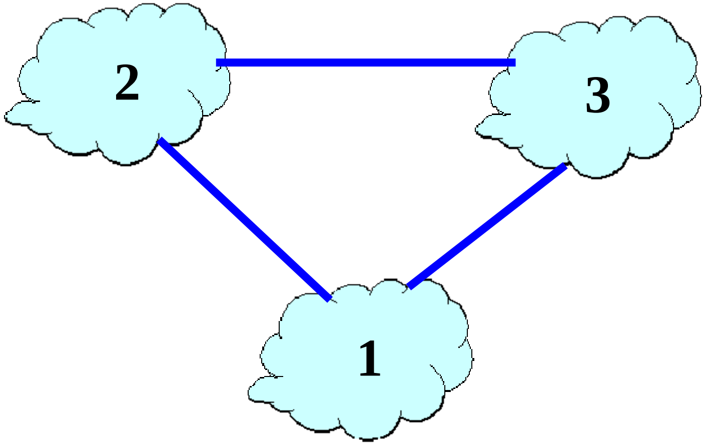
Faster Loop Detection

- Node can easily detect a loop
  - –Look for its own node identifier in the path
  - –E.g., node 1 sees itself in the path “3, 2, 1”
- Node can simply discard paths with loops
  - –E.g., node 1 simply discards the advertisement



Flexible Policies

- Each node can apply local policies
  - –Path selection: Which path to use?
  - –Path export: Which paths to advertise?
- Examples
  - –Node 2 may prefer the path “2, 3, 1” over “2, 1”
  - –Node 1 may not let node 3 hear the path “1, 2”

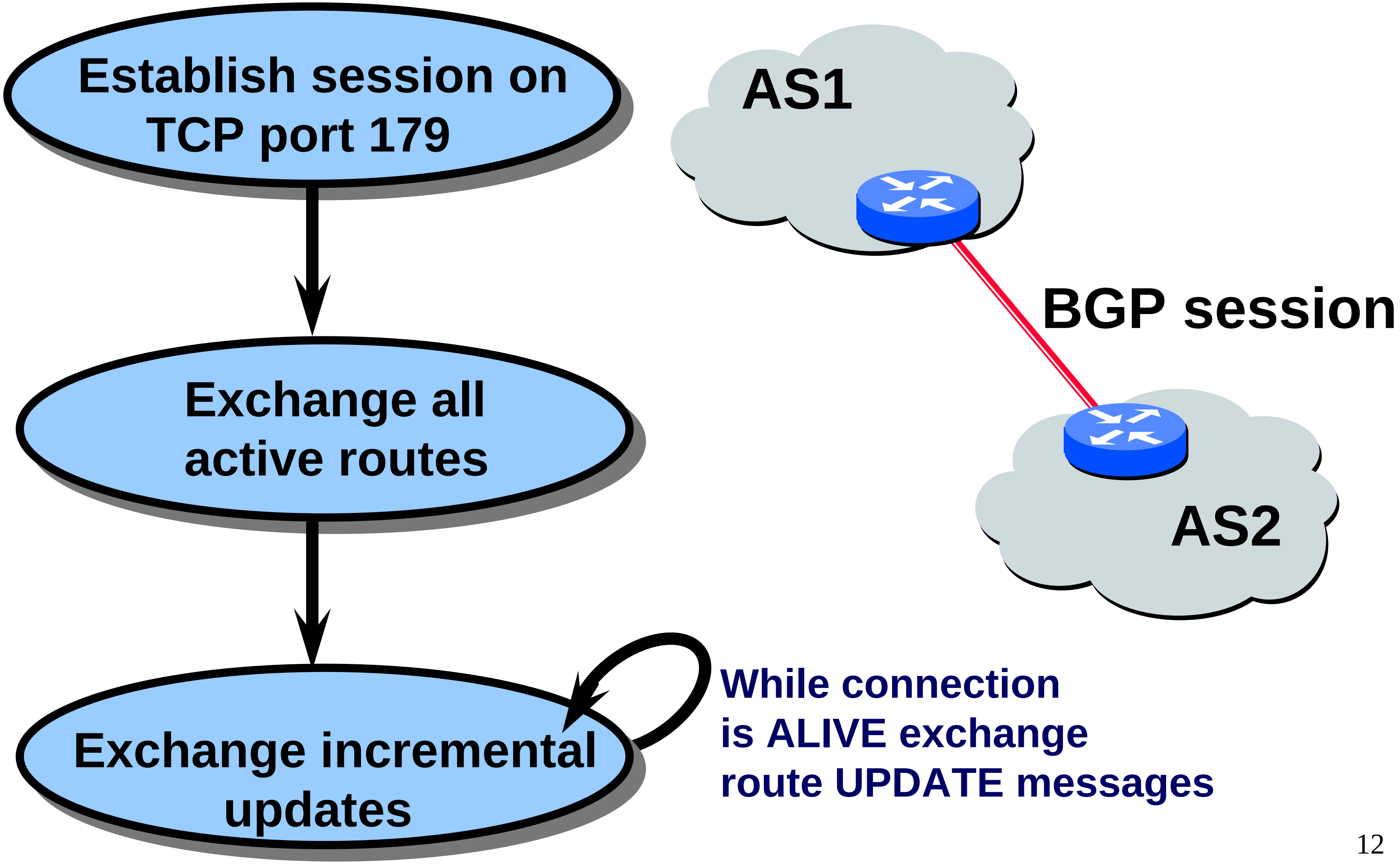


Border Gateway Protocol

- Interdomain routing protocol for the Internet
  - Prefix-based path-vector protocol
  - Policy-based routing based on AS Paths
  - Evolved during the past 15 years

- 1989 : BGP-1 [RFC 1105]
  - Replacement for EGP (1984, RFC 904)
- 1990 : BGP-2 [RFC 1163]
- 1991 : BGP-3 [RFC 1267]
- 1995 : BGP-4 [RFC 1771]
  - Support for Classless Interdomain Routing (CIDR)

BGP Operations



Establish session on TCP port 179

Exchange all active routes

Exchange incremental updates

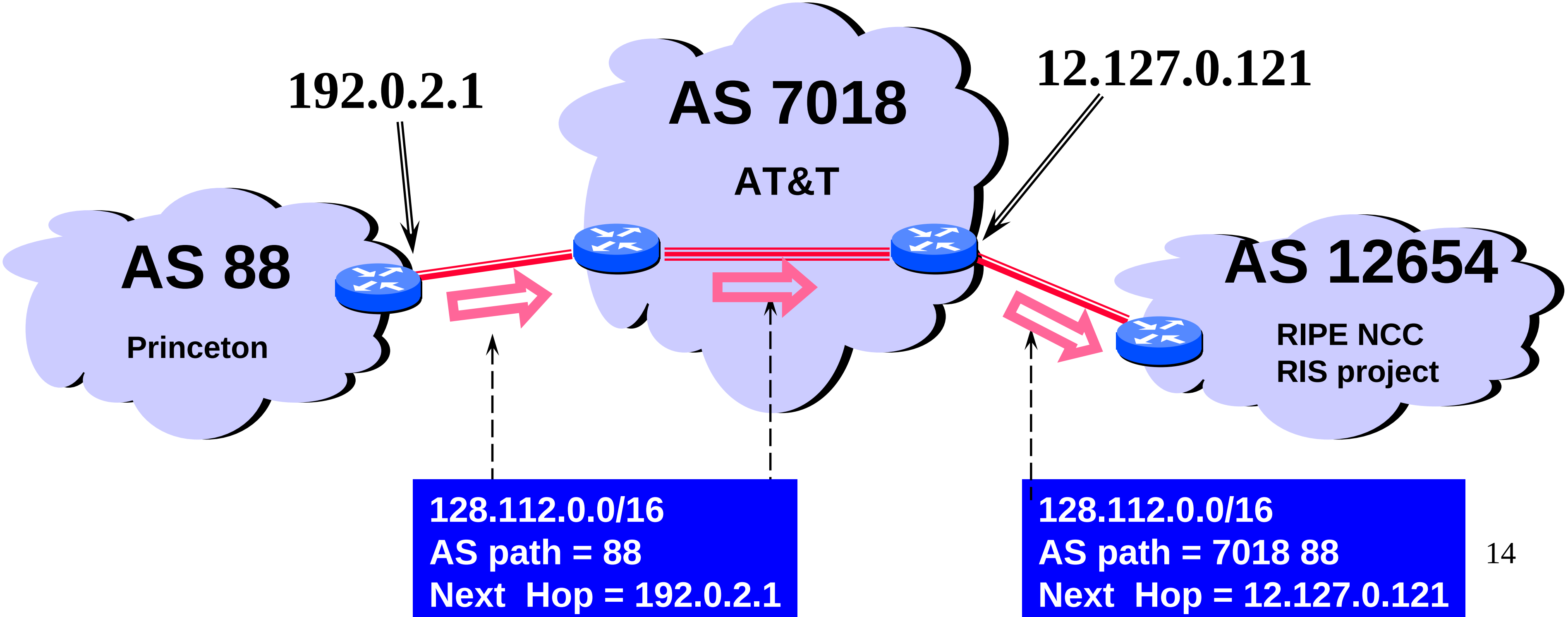
While connection is ALIVE exchange route UPDATE messages

Incremental Protocol

- A node learns multiple paths to destination
  - Stores all of the routes in a routing table
  - Applies policy to select a single active route
  - … and may advertise the route to its neighbors
- Incremental updates
  - Announcement
    - Upon selecting a new active route, add node id to path
    - … and (optionally) advertise to each neighbor
  - Withdrawal
    - If the active route is no longer available
    - … send a withdrawal message to the neighbors

BGP Route

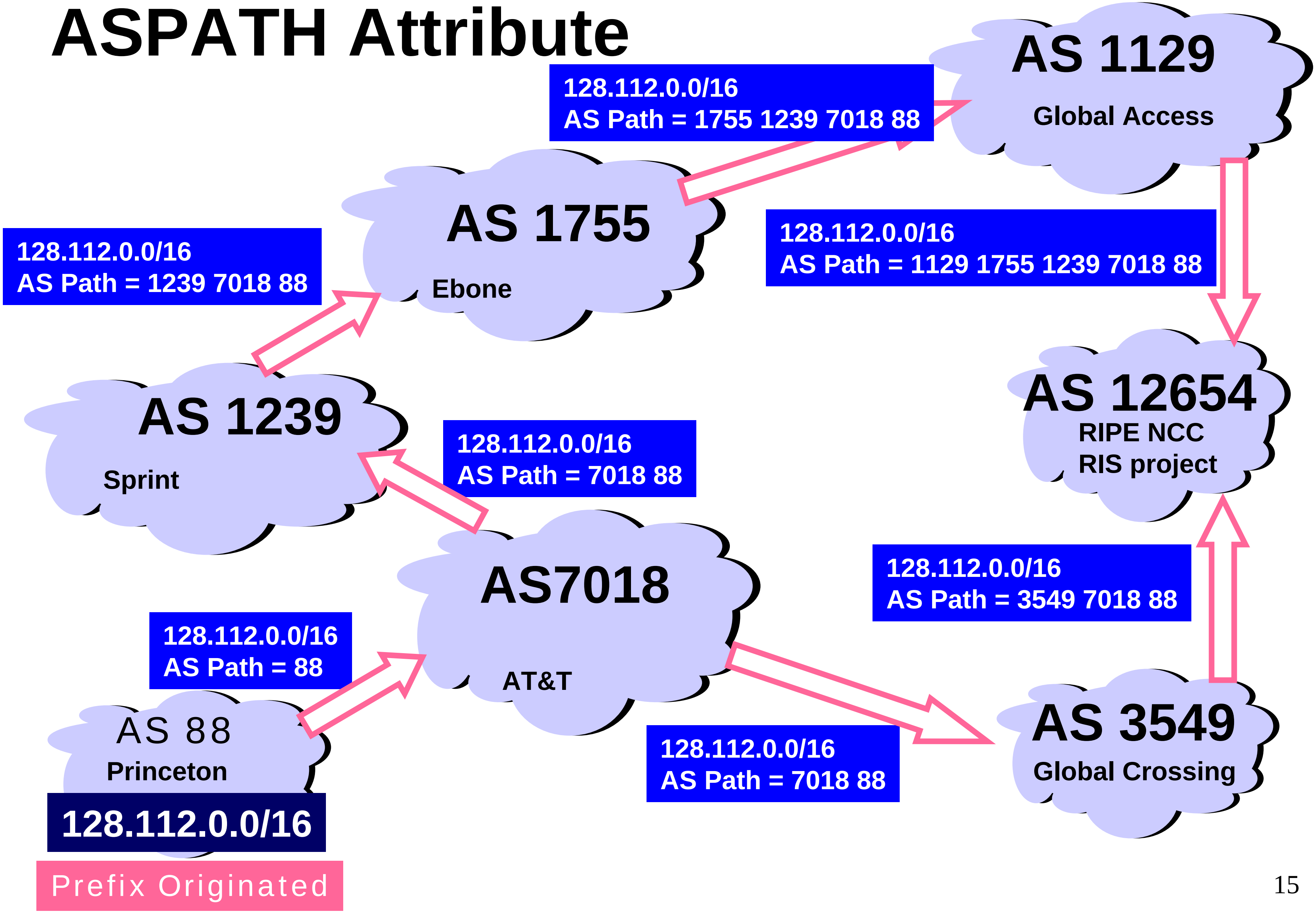
- Destination prefix (e.g,. 128.112.0.0/16)
- Route attributes, including
  - AS path (e.g., “7018 88”)
  - Next-hop IP address (e.g., 12.127.0.121)



128.112.0.0/16
AS path = 88
Next Hop = 192.0.2.1

128.112.0.0/16
AS path = 7018 88
Next Hop = 12.127.0.121

ASPATH Attribute



AS 88
Princeton

128.112.0.0/16

Prefix Originated

128.112.0.0/16
AS Path = 88

AS7018
AT&T

128.112.0.0/16
AS Path = 7018 88

AS 1239
Sprint

128.112.0.0/16
AS Path = 1239 7018 88

AS 1755
Ebone

128.112.0.0/16
AS Path = 1755 1239 7018 88

AS 1129
Global Access

128.112.0.0/16
AS Path = 1129 1755 1239 7018 88

AS 12654
RIPE NCC
RIS project

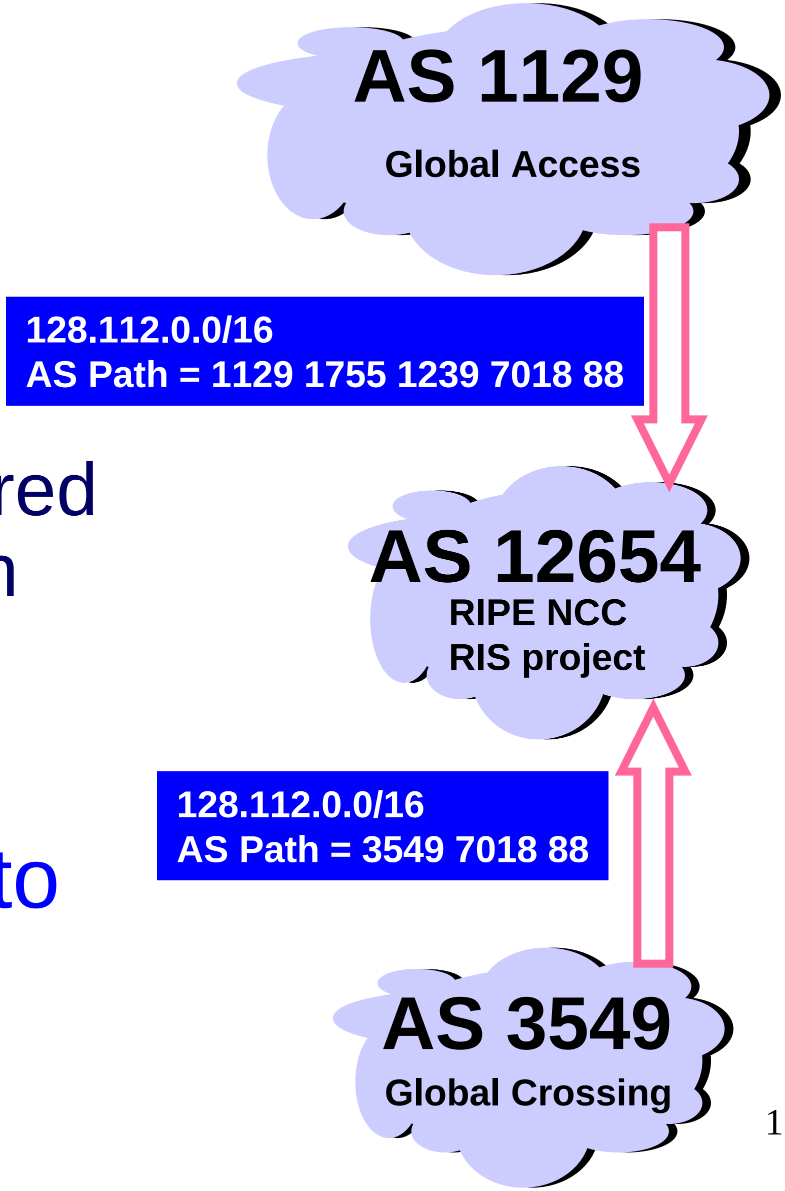
128.112.0.0/16
AS Path = 7018 88

AS 3549
Global Crossing

128.112.0.0/16
AS Path = 3549 7018 88

BGP Path Selection

- Simplest case
  - Shortest AS path
  - Arbitrary tie break
- Example
  - Four-hop AS path preferred over a three-hop AS path
  - AS 12654 prefers path through Global Crossing
- But, BGP is not limited to shortest-path routing
  - Policy-based routing

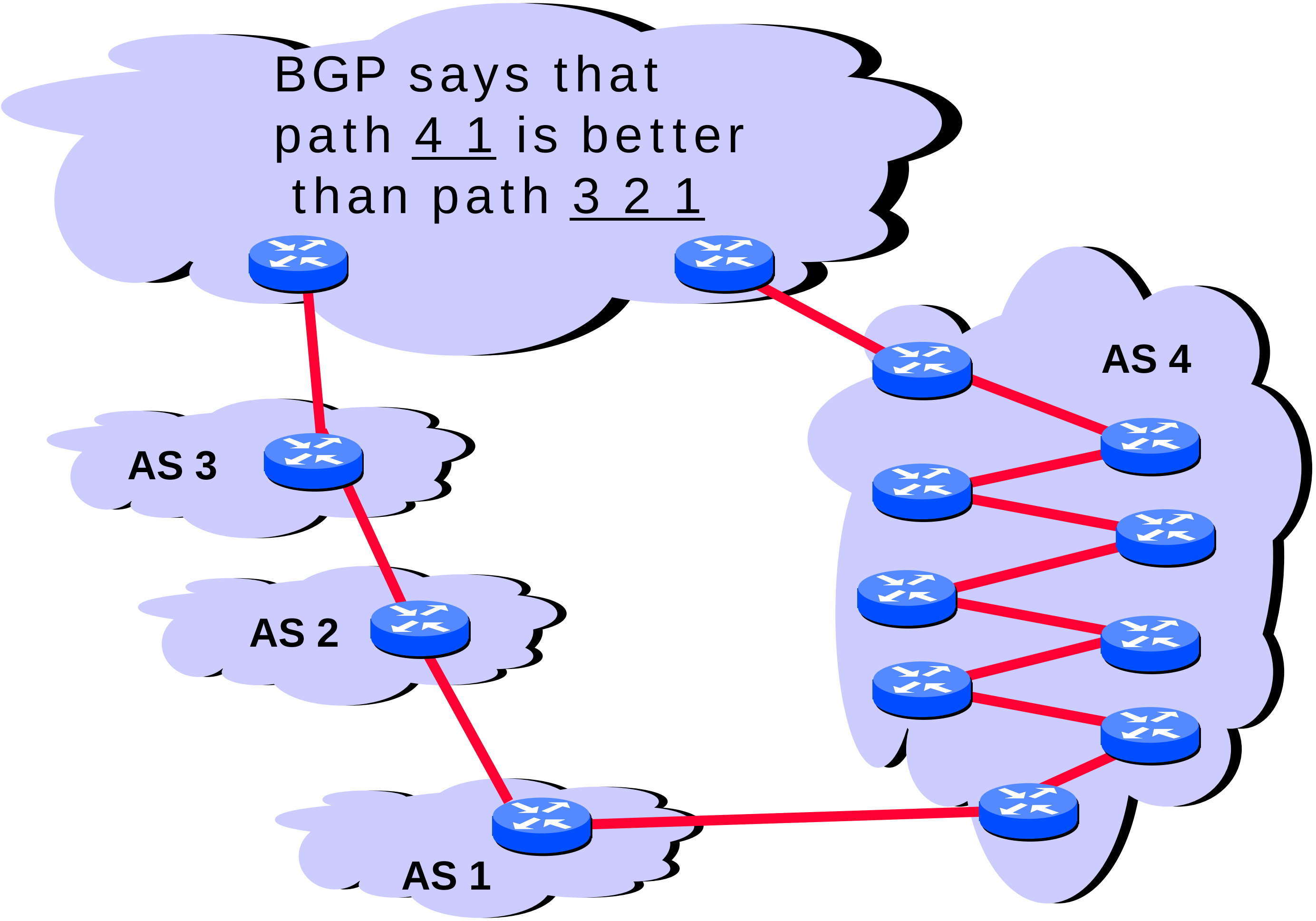


BGP Policy: Applying Policy to Routes

- Import policy
  - Filter unwanted routes from neighbor
    - E.g. prefix that your customer doesn’t own
  - Manipulate attributes to influence path selection
    - E.g., assign local preference to favored routes
- Export policy
  - Filter routes you don’t want to tell your neighbor
    - E.g., don’t tell a peer a route learned from other peer
  - Manipulate attributes to control what they see
    - E.g., make a path look artificially longer than it is

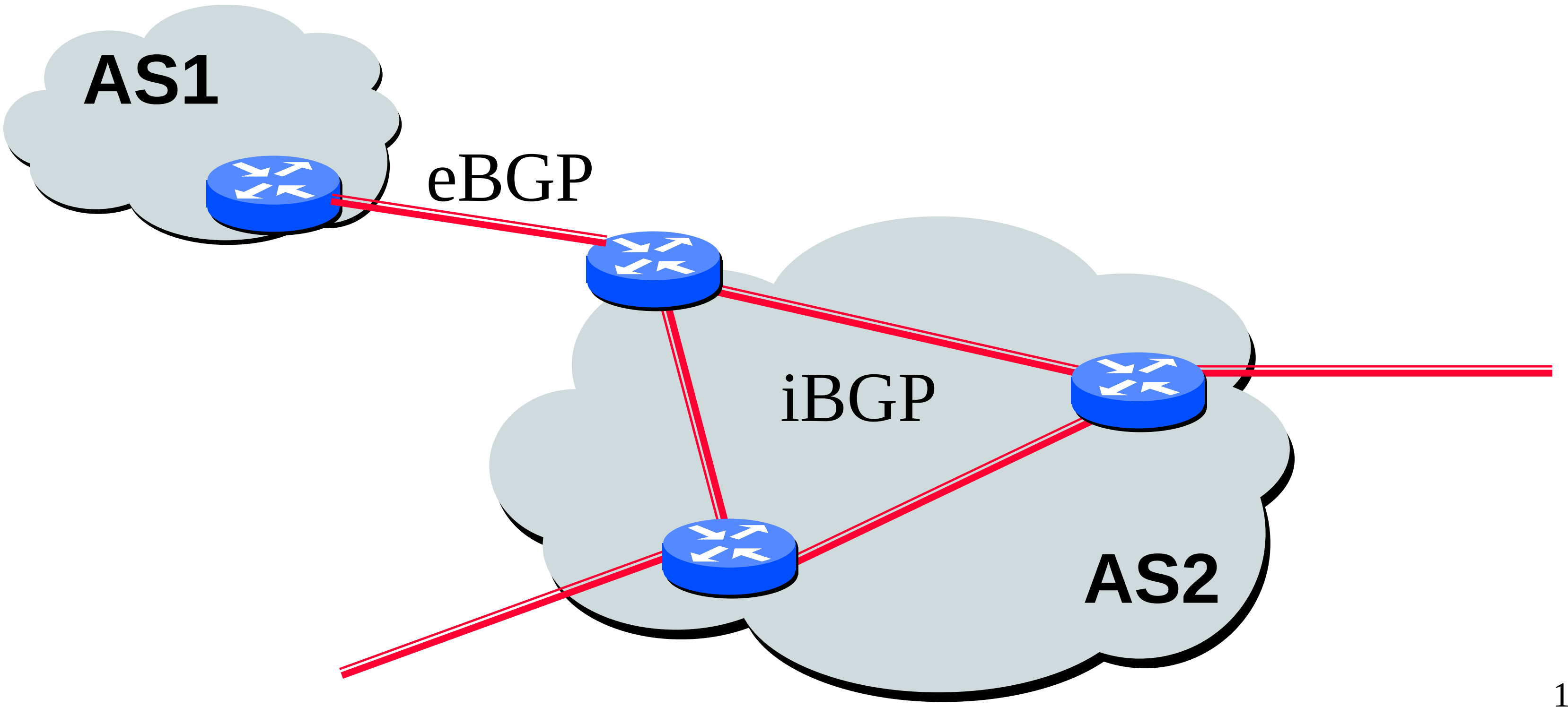
AS is Not a Single Node

- AS path length can be misleading
  - An AS may have many router-level hops



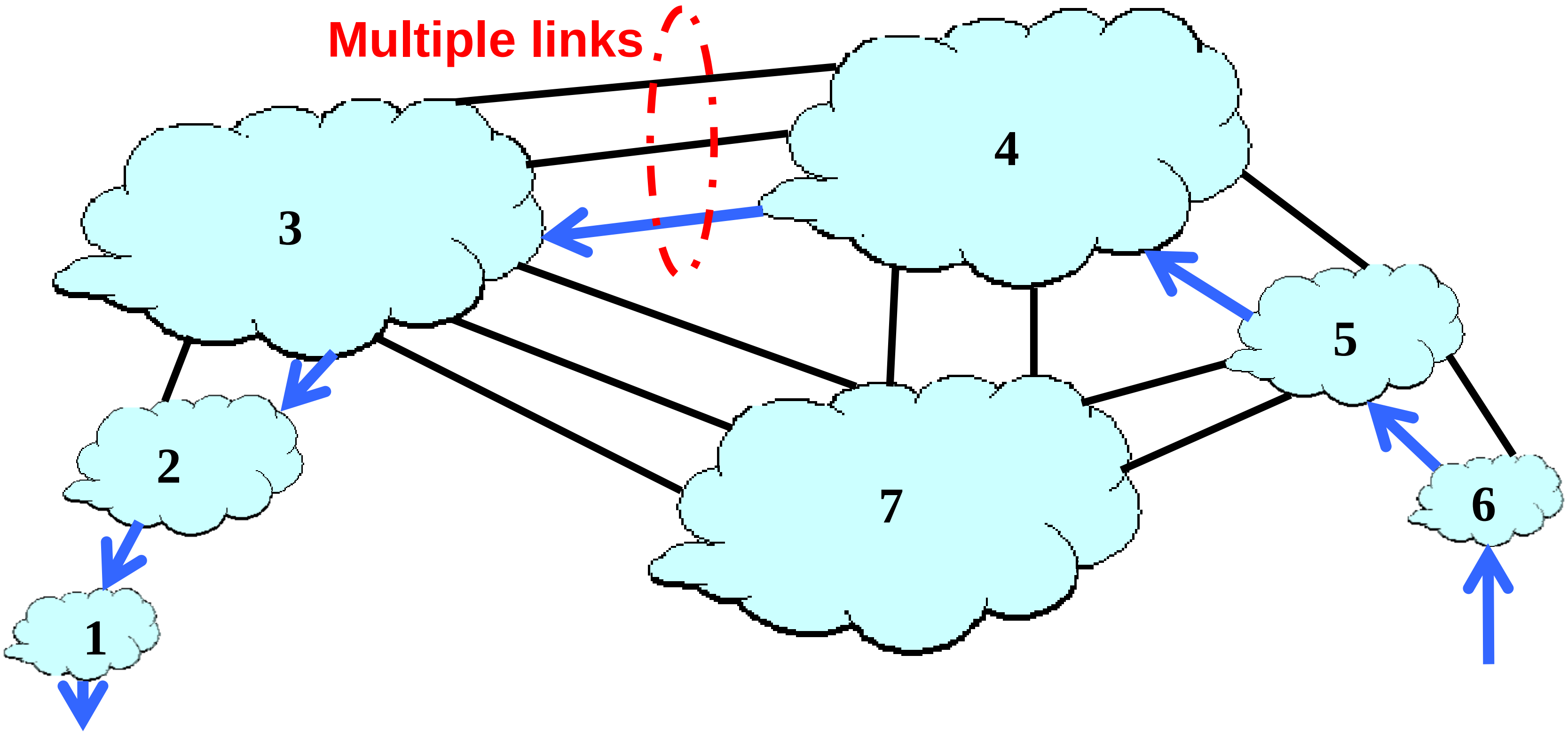
An AS is Not a Single Node

- Multiple routers in an AS
  - –Need to distribute BGP information within the AS
  - –Internal BGP (iBGP) sessions between routers



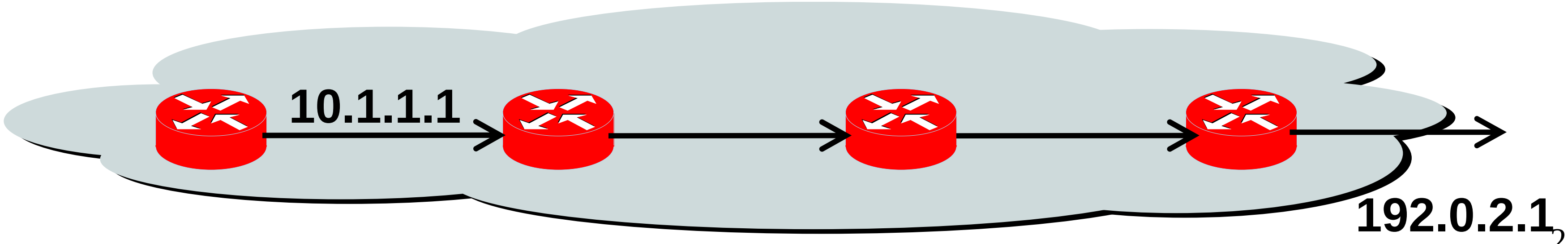
An AS is Not a Single Node

- Multiple connections to neighboring ASes
  - –Multiple border routers may learn good routes
  - –… with the same local-pref and AS path length

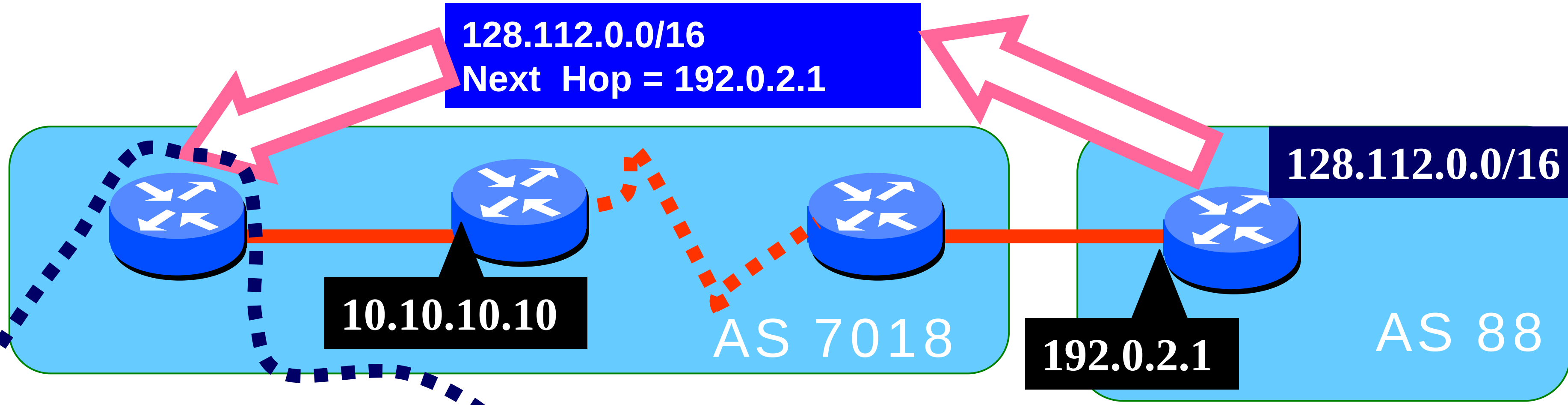


Joining BGP and IGP Information

- Border Gateway Protocol (BGP)
  - Announces reachability to external destinations
  - Maps a destination prefix to an egress point
    - 128.112.0.0/16 reached via 192.0.2.1
- Interior Gateway Protocol (IGP)
  - Used to compute paths within the AS
  - Maps an egress point to an outgoing link
    - 192.0.2.1 reached via 10.1.1.1



Joining BGP with IGP Information



Forwarding Table

destination	next hop
192.0.2.0/30	10.10.10.10

+

BGP

destination	next hop
135.207.0.0/16	192.0.2.1

Forwarding Table

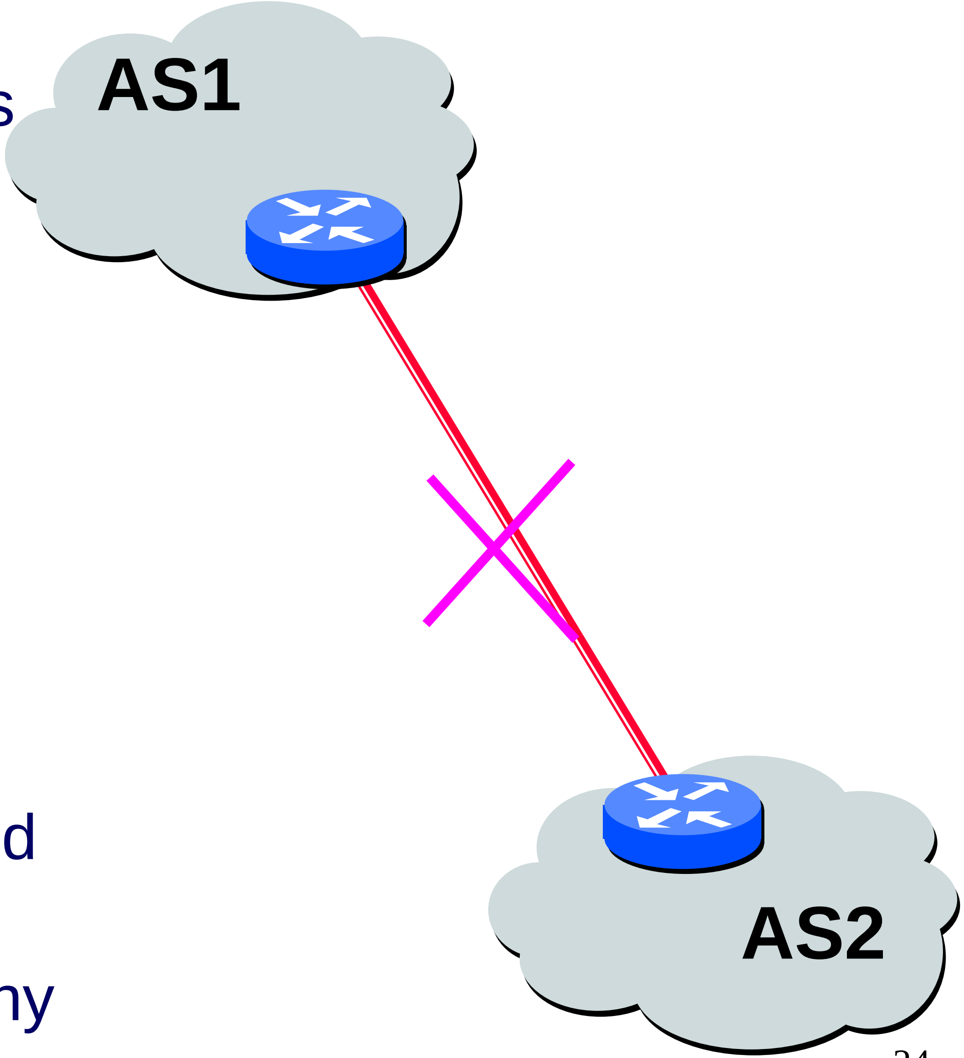
destination	next hop
135.207.0.0/16	10.10.10.10
192.0.2.0/30	10.10.10.10

Causes of BGP Routing Changes

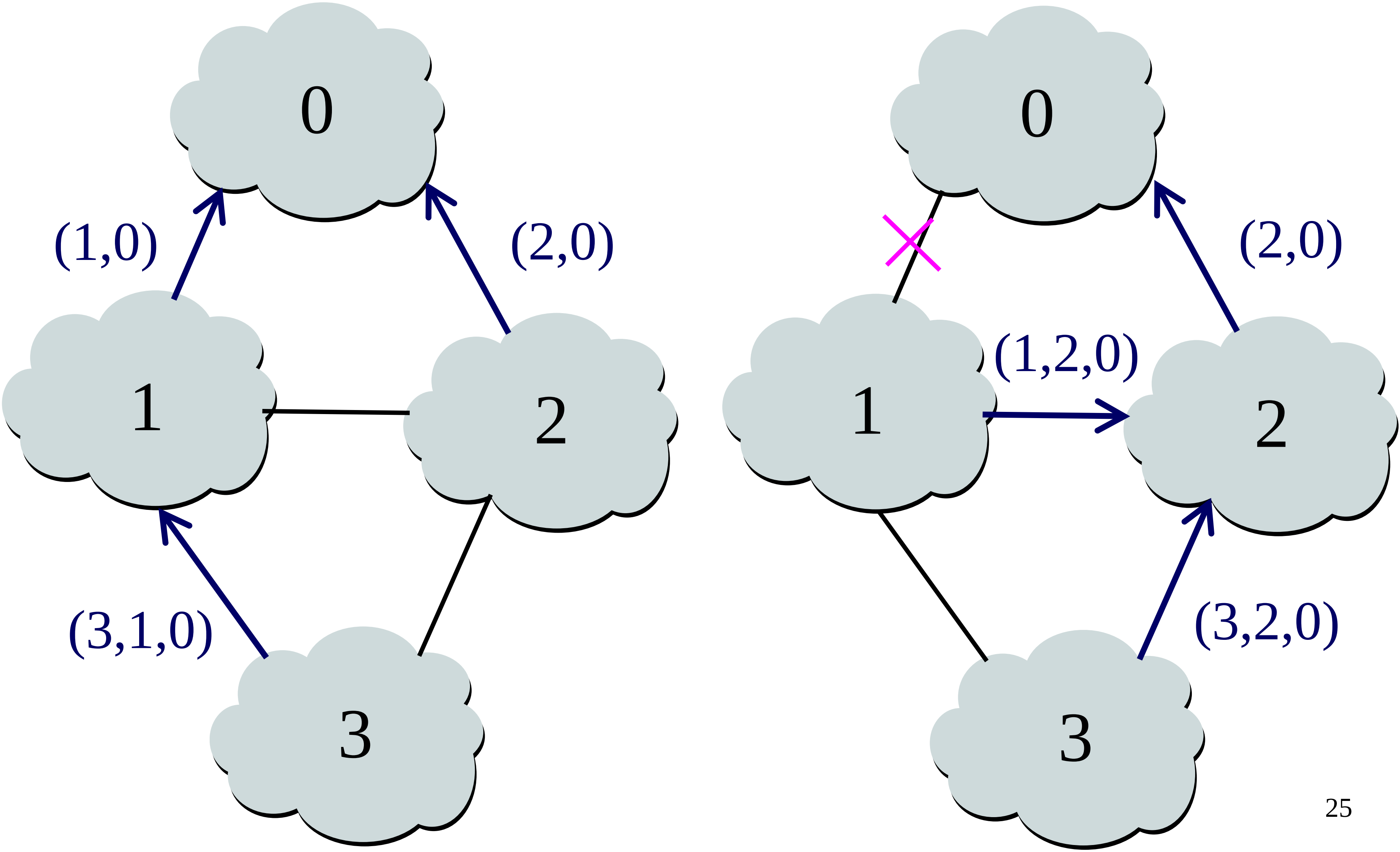
- Topology changes
  - Equipment going up or down
  - Deployment of new routers or sessions
- BGP session failures
  - Due to equipment failures, maintenance, etc.
  - Or, due to congestion on the physical path
- Changes in routing policy
  - Reconfiguration of preferences
  - Reconfiguration of route filters
- Persistent protocol oscillation
  - Conflicts between policies in different ASes

BGP Session Failure

- BGP runs over TCP
  - BGP only sends updates when changes occur
  - TCP doesn't detect lost connectivity on its own
- Detecting a failure
  - Keep-alive: 60 seconds
  - Hold timer: 180 seconds
- Reacting to a failure
  - Discard all routes learned from the neighbor
  - Send new updates for any routes that change

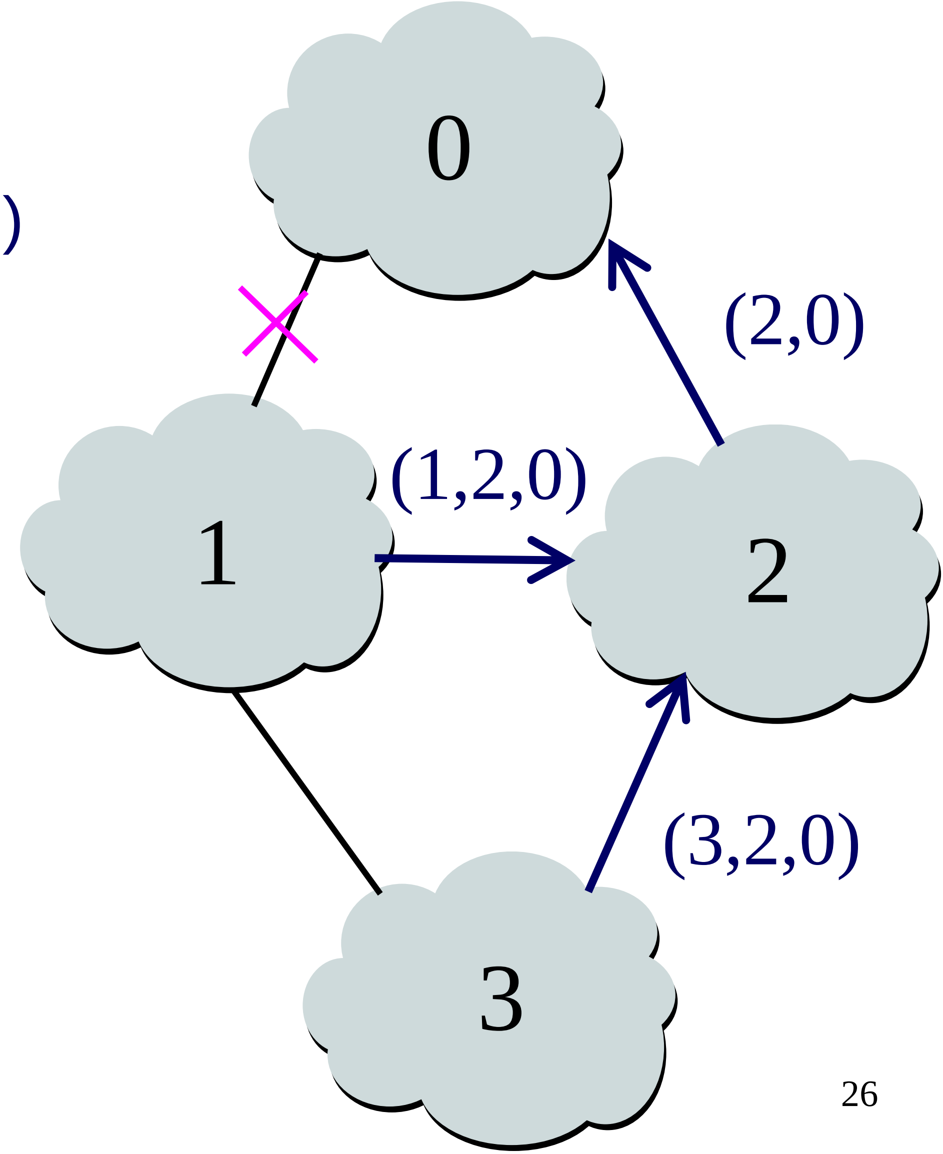


Routing Change: Before and After



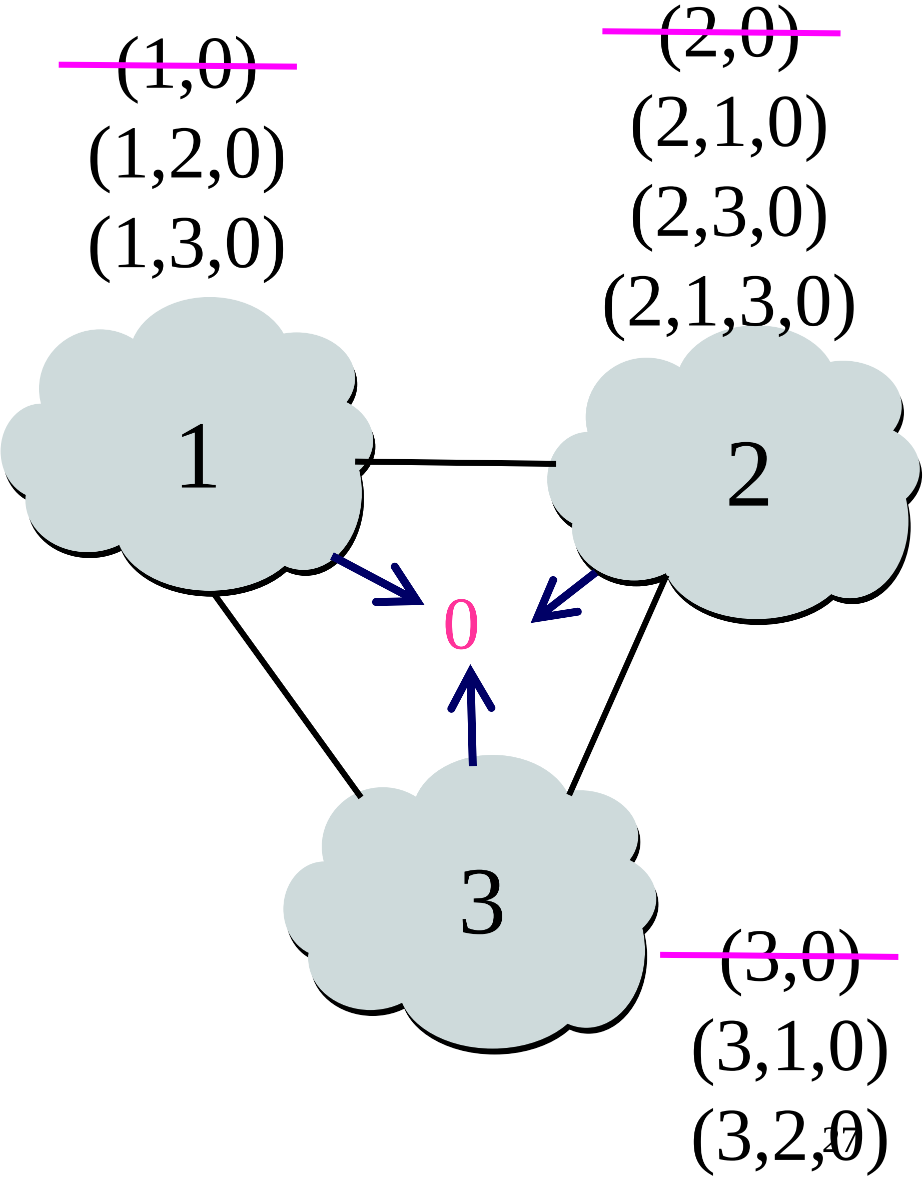
Routing Change: Path Exploration

- AS 1
  - Delete the route (1,0)
  - Switch to next route (1,2,0)
  - Send route (1,2,0) to AS 3
- AS 3
  - Sees (1,2,0) replace (1,0)
  - Compares to route (2,0)
  - Switches to using AS 2



Routing Change: Path Exploration

- Initial situation
  - Destination 0 is alive
  - All ASes use direct path
- When destination dies
  - All ASes lose direct path
  - All switch to longer paths
  - Eventually withdrawn
- E.g., AS 2
  - (2,0) → (2,1,0)
  - (2,1,0) → (2,3,0)
  - (2,3,0) → (2,1,3,0)
  - (2,1,3,0) → null



BGP Converges Slowly, if at All

- Path vector avoids count-to-infinity
  - But, ASes still must explore many alternate paths
  - … to find the highest-ranked path that is still available
- Fortunately, in practice
  - Most popular destinations have very stable BGP routes
  - And most instability lies in a few unpopular destinations
- Still, lower BGP convergence delay is a goal
  - Can be tens of seconds to tens of minutes
  - High for important interactive applications
  - … or even conventional application, like Web browsing

Conclusions

- BGP is solving a hard problem
  - Routing protocol operating at a global scale
  - With tens of thousands of independent networks
  - That each has their own policy goals
  - And all want fast convergence
- Key features of BGP
  - Prefix-based path-vector protocol
  - Incremental updates (announcements and withdrawals)
  - Policies applied at import and export of routes
  - Internal BGP to distribute information within an AS
  - Interaction with the IGP to compute forwarding tables

Transport Protocols

Sections 2.5, 5.1, and 5.2

Introduction to Data Networks

2008.4

Goals for Today's Lecture

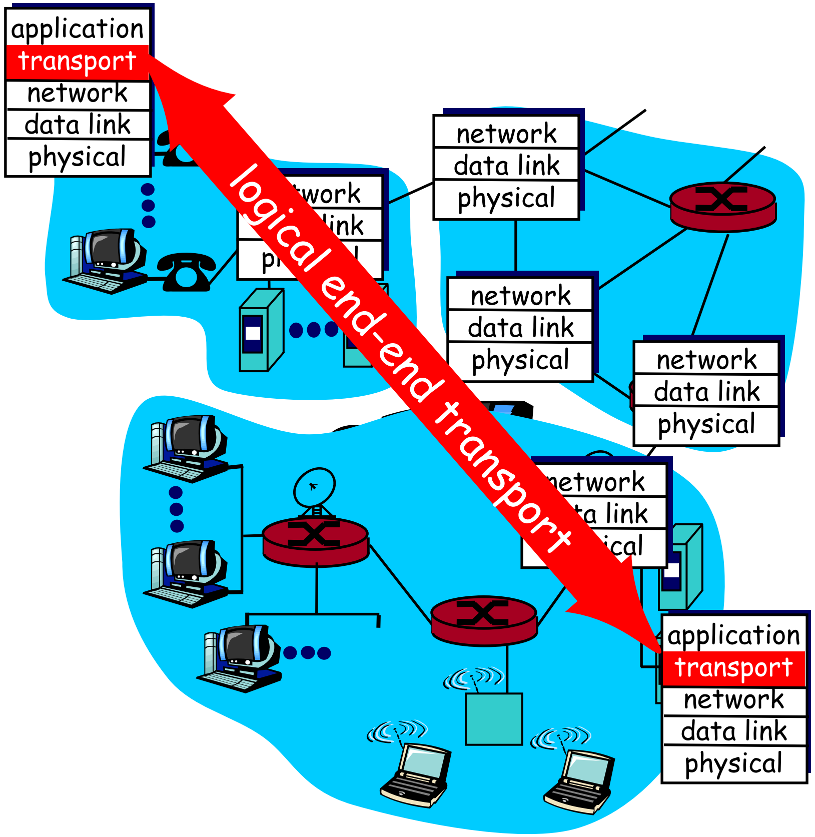
- Principles underlying transport-layer services
  - (De)multiplexing
  - Detecting corruption
  - Reliable delivery
  - Flow control
- Transport-layer protocols in the Internet
  - User Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)

Role of Transport Layer

- Application layer
  - Communication for specific applications
  - E.g., HyperText Transfer Protocol (HTTP), File Transfer Protocol (FTP), Network News Transfer Protocol (NNTP)
- Transport layer
  - Communication between processes (e.g., socket)
  - Relies on network layer and serves the application layer
  - E.g., TCP and UDP
- Network layer
  - Logical communication between nodes
  - Hides details of the link technology
  - E.g., IP

Transport Protocols

- Provide *logical communication* between application processes running on different hosts
- Run on end hosts
  - Sender: breaks application messages into segments, and passes to network layer
  - Receiver: reassembles segments into messages, passes to application layer
- Multiple transport protocol available to applications
  - Internet: TCP and UDP

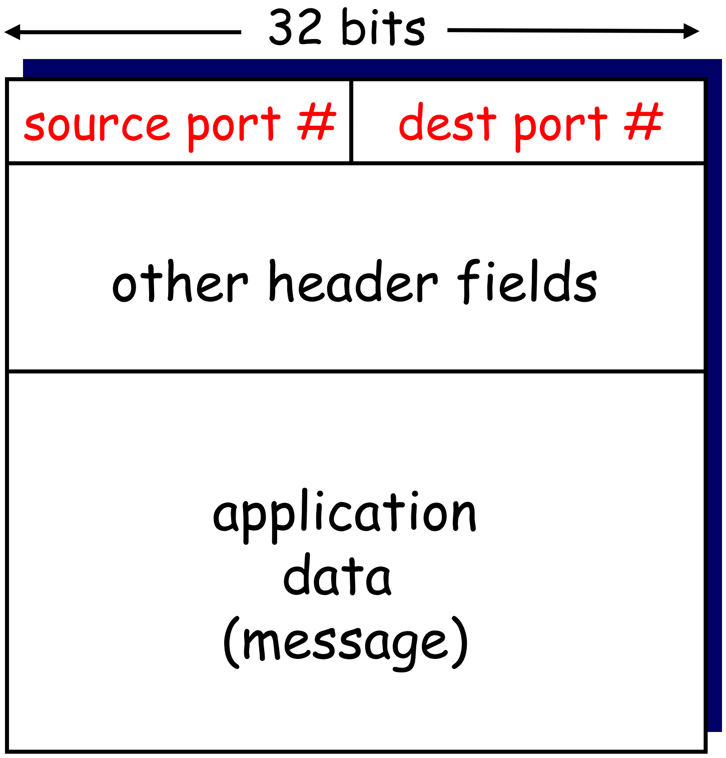


Internet Transport Protocols

- Datagram messaging service (UDP)
  - No-frills extension of “best-effort” IP
- Reliable, in-order delivery (TCP)
  - Connection set-up
  - Discarding of corrupted packets
  - Retransmission of lost packets
  - Flow control
  - Congestion control (next lecture)
- Other services not available
  - Delay guarantees
  - Bandwidth guarantees

Multiplexing and Demultiplexing

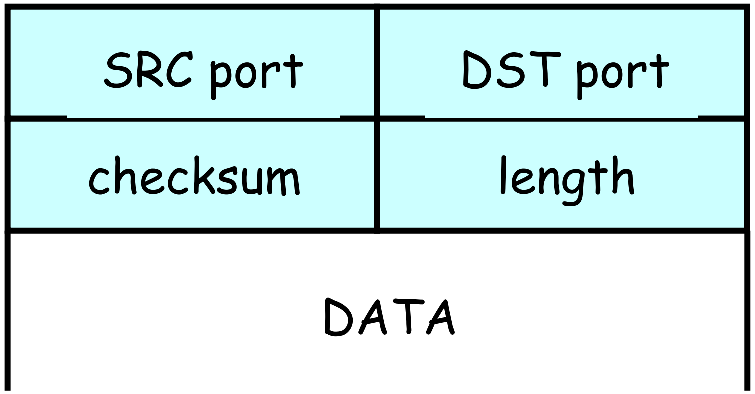
- Host receives IP datagrams
  - Each datagram has source and destination IP address,
  - Each datagram carries one transport-layer segment
  - Each segment has source and destination port number
- Host uses IP addresses and port numbers to direct the segment to appropriate socket



TCP/UDP segment format

Unreliable Message Delivery Service

- Lightweight communication between processes
  - Avoid overhead and delays of ordered, reliable delivery
  - Send messages to and receive them from a socket
- User Datagram Protocol (UDP)
  - IP plus port numbers to support (de)multiplexing
  - Optional error checking on the packet contents



SRC port	DST port
checksum	length
DATA	

Connectionless demultiplexing

- Create sockets with port numbers:

```
DatagramSocket mySocket1 = new
  DatagramSocket(99111);
```

```
DatagramSocket mySocket2 = new
  DatagramSocket(99222);
```

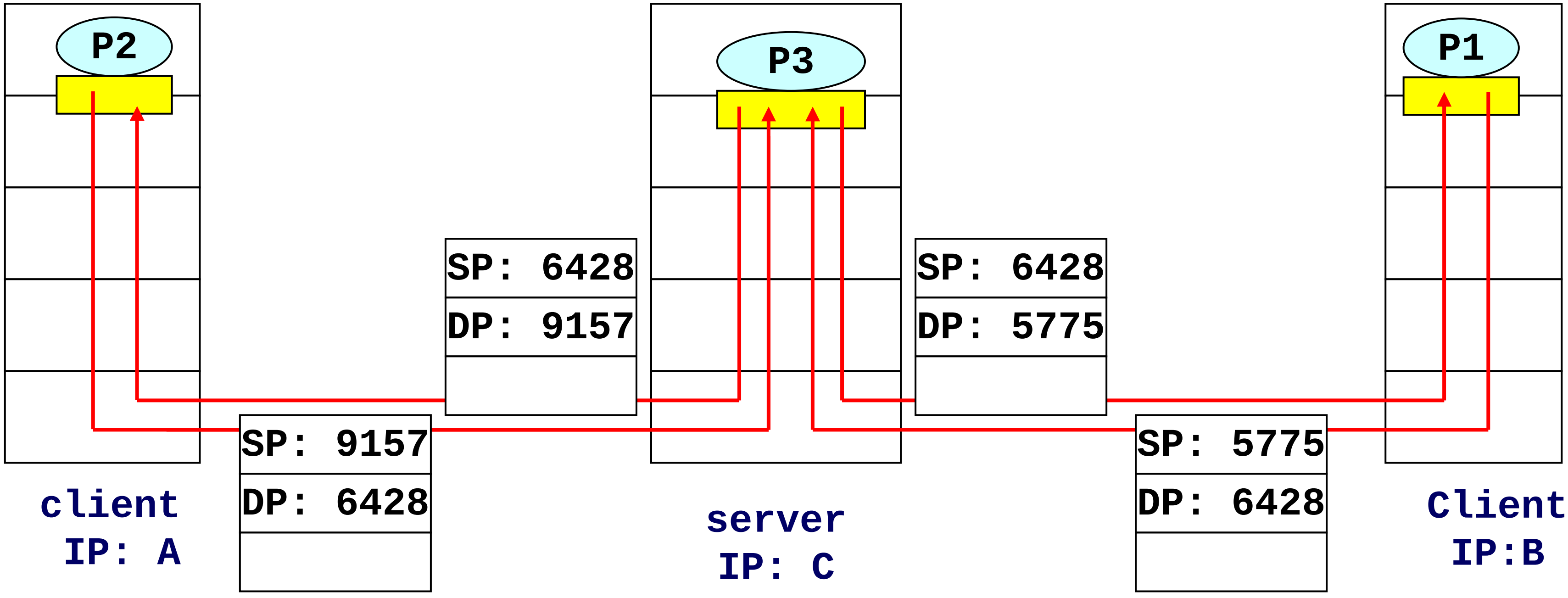
- UDP socket identified by two-tuple:

(dest IP address, dest port number)

- When host receives UDP segment:
  - checks destination port number in segment
  - directs UDP segment to socket with that port number
- IP datagrams with different source IP addresses and/or source port numbers directed to same socket

Connectionless demux (cont)

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```



SP provides "return address"

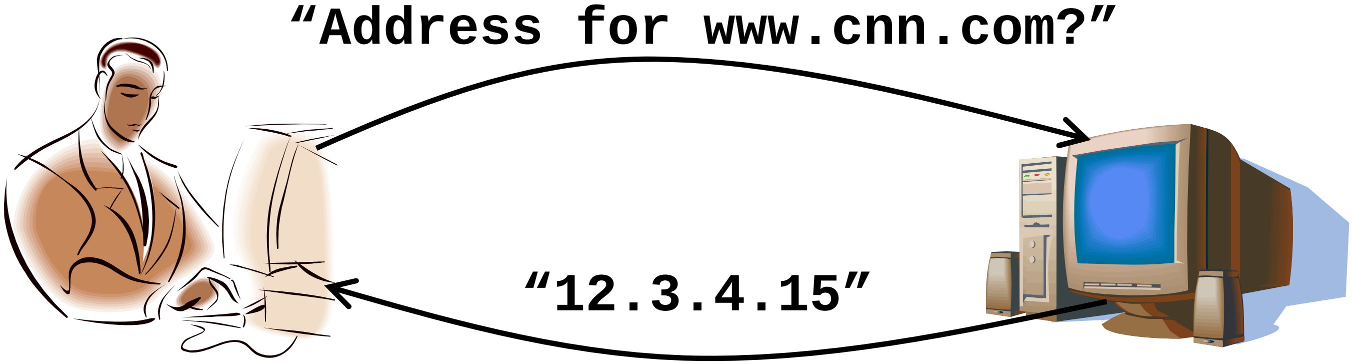
Why Would Anyone Use UDP?

- Finer control over what data is sent and when
  - As soon as an application process writes into the socket
  - … UDP will package the data and send the packet
- No delay for connection establishment
  - UDP just blasts away without any formal preliminaries
  - … which avoids introducing any unnecessary delays
- No connection state
  - No allocation of buffers, parameters, sequence #s, etc.
  - … making it easier to handle many active clients at once
- Small packet header overhead
  - UDP header is only eight-bytes long

Popular Applications That Use UDP

- Multimedia streaming
  - Retransmitting lost/corrupted packets is not worthwhile
  - By the time the packet is retransmitted, it’s too late
  - E.g., telephone calls, video conferencing, gaming

- Simple query protocols like Domain Name System
  - Overhead of connection establishment is overkill
  - Easier to have application retransmit if needed



“Address for www.cnn.com?”

“12.3.4.15”

Transmission Control Protocol (TCP)

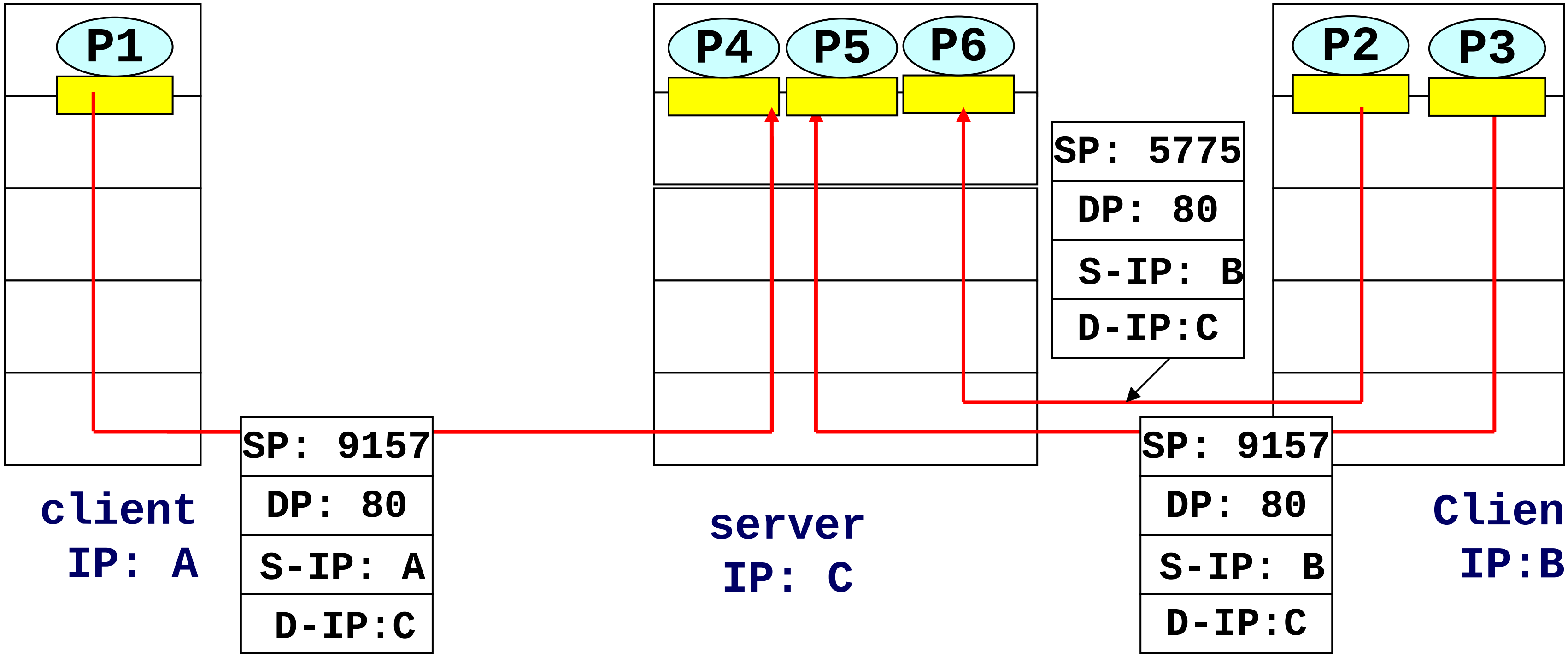
- Connection oriented
  - Explicit set-up and tear-down of TCP session
- Stream-of-bytes service
  - Sends and receives a stream of bytes, not messages
- Reliable, in-order delivery
  - Checksums to detect corrupted data
  - Acknowledgments & retransmissions for reliable delivery
  - Sequence numbers to detect losses and reorder data
- Flow control
  - Prevent overflow of the receiver’s buffer space
- Congestion control
  - Adapt to network congestion for the greater good

Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- recv host uses all four values to direct segment to appropriate socket

- Server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- Web servers have different sockets for each connecting client
  - HTTP will have different socket for each request

Connection-oriented demux (cont)



An Analogy: Talking on a Cell Phone

- Alice and Bob on their cell phones
  - Both Alice and Bob are talking
- What if Alice couldn’t understand Bob?
  - Bob asks Alice to repeat what she said
- What if Bob hasn’t heard Alice for a while?
  - Is Alice just being quiet?
  - Or, have Bob and Alice lost reception?
  - How long should Bob just keep on talking?
  - Maybe Alice should periodically say “uh huh”
  - … or Bob should ask “Can you hear me now?” ☺

Some Take-Aways from the Example

- Acknowledgments from receiver
  - Positive: “okay” or “ACK”
  - Negative: “please repeat that” or “NACK”
- Timeout by the sender (“stop and wait”)
  - Don’t wait indefinitely without receiving some response
  - … whether a positive or a negative acknowledgment
- Retransmission by the sender
  - After receiving a “NACK” from the receiver
  - After receiving no feedback from the receiver

Challenges of Reliable Data Transfer

- Over a perfectly reliable channel
  - All of the data arrives in order, just as it was sent
  - Simple: sender sends data, and receiver receives data
- Over a channel with bit errors
  - All of the data arrives in order, but some bits corrupted
  - Receiver detects errors and says “please repeat that”
  - Sender retransmits the data that were corrupted
- Over a lossy channel with bit errors
  - Some data are missing, and some bits are corrupted
  - Receiver detects errors but cannot always detect loss
  - Sender must wait for acknowledgment (“ACK” or “OK”)
  - … and retransmit data after some time if no ACK arrives

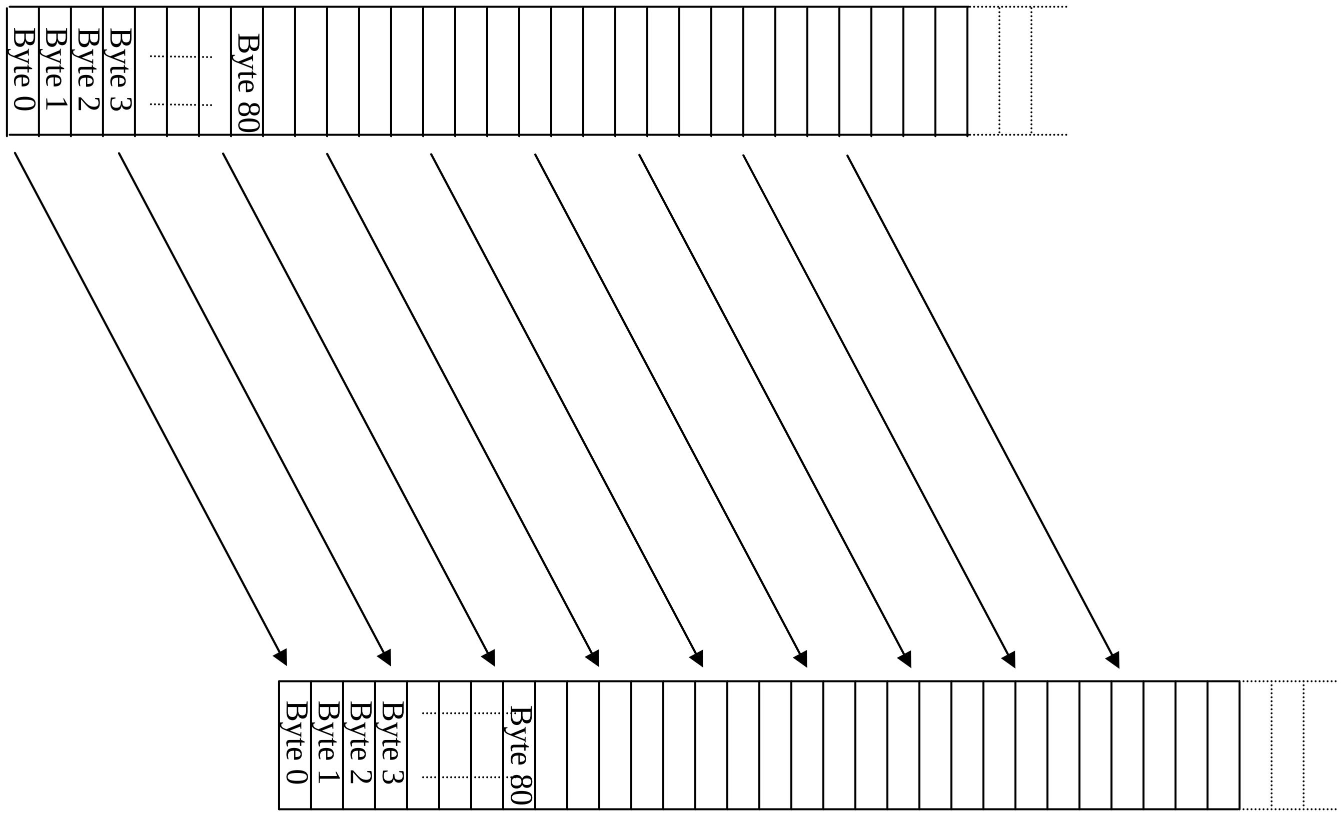
TCP Support for Reliable Delivery

- Checksum
  - Used to detect corrupted data at the receiver
  - …leading the receiver to drop the packet
- Sequence numbers
  - Used to detect missing data
  - ... and for putting the data back in order
- Retransmission
  - Sender retransmits lost or corrupted data
  - Timeout based on estimates of round-trip time
  - Fast retransmit algorithm for rapid retransmission

TCP Segments

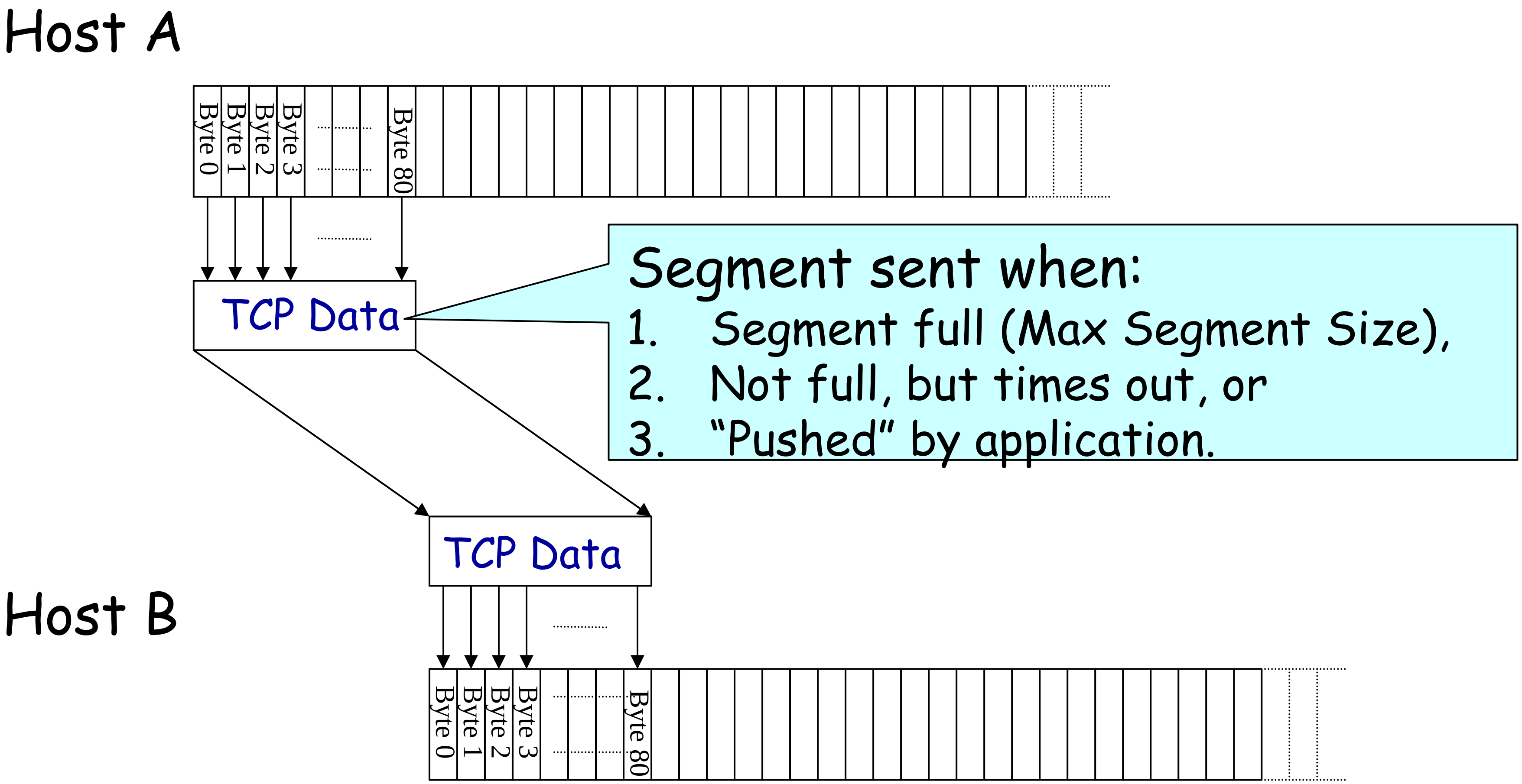
TCP “Stream of Bytes” Service

Host A



Host B

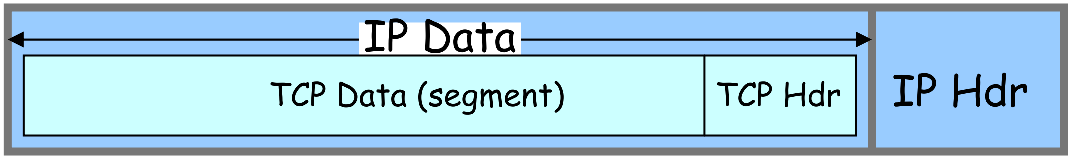
…Emulated Using TCP “Segments”



Segment sent when:

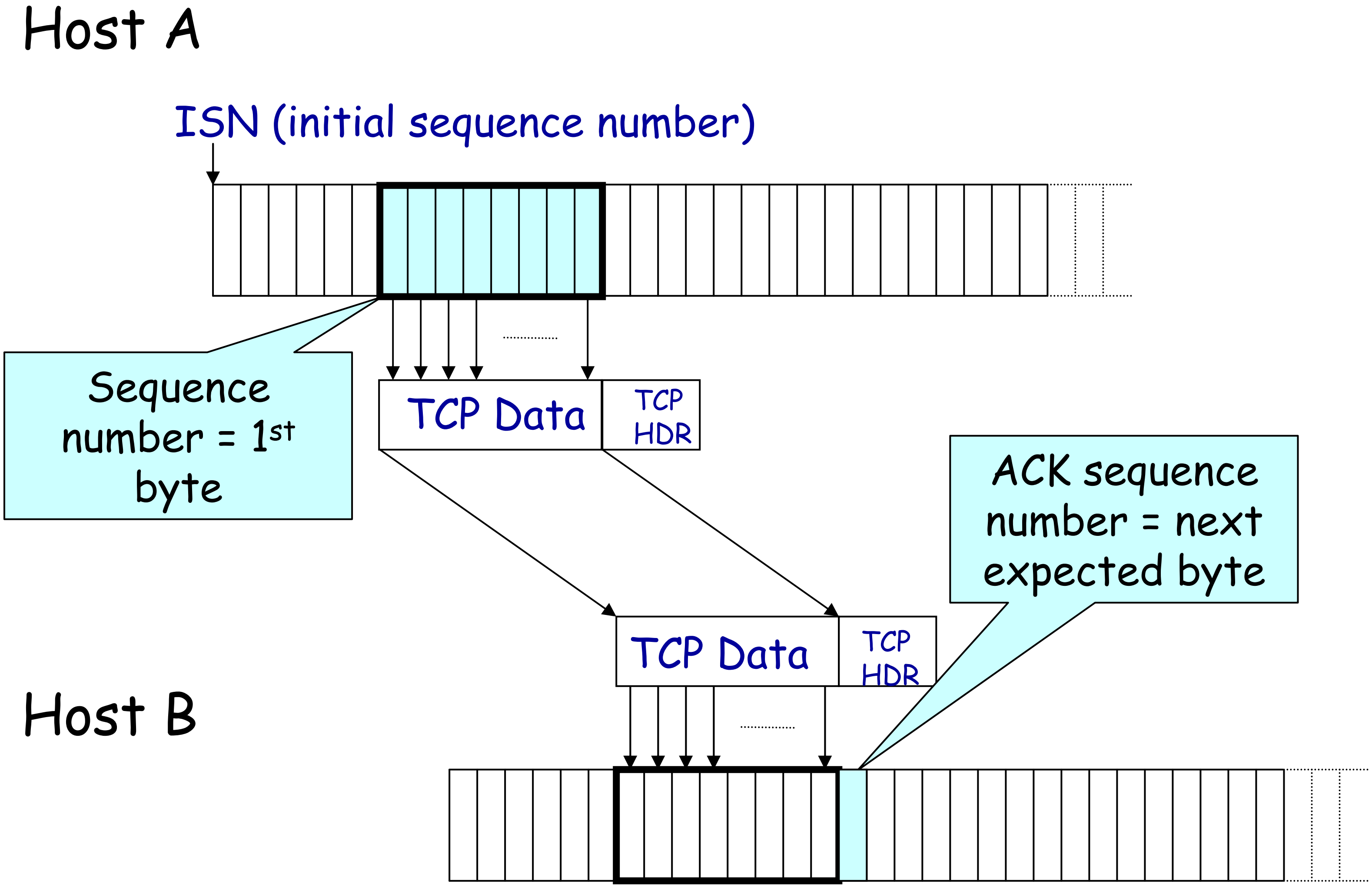
1. Segment full (Max Segment Size),
2. Not full, but times out, or
3. “Pushed” by application.

TCP Segment



- IP packet
  - No bigger than Maximum Transmission Unit (MTU)
  - E.g., up to 1500 bytes on an Ethernet
- TCP packet
  - IP packet with a TCP header and data inside
  - TCP header is typically 20 bytes long
- TCP segment
  - No more than Maximum Segment Size (MSS) bytes
  - E.g., up to 1460 consecutive bytes from the stream

Sequence Numbers

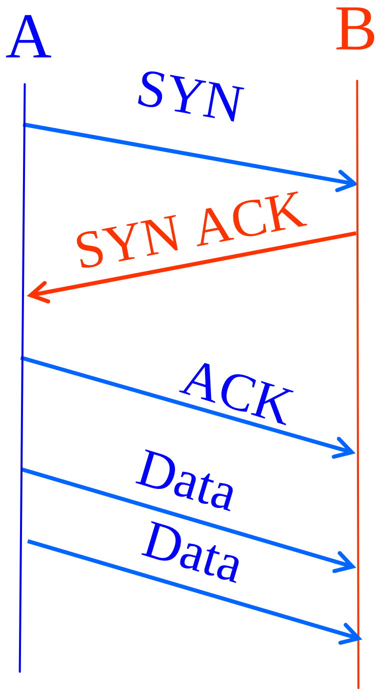

Host A
ISN (initial sequence number)
Sequence number = 1st byte
TCP Data
TCP HDR
ACK sequence number = next expected byte
TCP Data
TCP HDR
Host B

Initial Sequence Number (ISN)

- Sequence number for the very first byte
  - E.g., Why not a de facto ISN of 0?
- Practical issue
  - IP addresses and port #s uniquely identify a connection
  - Eventually, though, these port #s do get used again
  - … and there is a chance an old packet is still in flight
  - … and might be associated with the new connection
- So, TCP requires changing the ISN over time
  - Set from a 32-bit clock that ticks every 4 microseconds
  - … which only wraps around once every 4.55 hours!
- But, this means the hosts need to exchange ISNs

TCP Three-Way Handshake

Establishing a TCP Connection

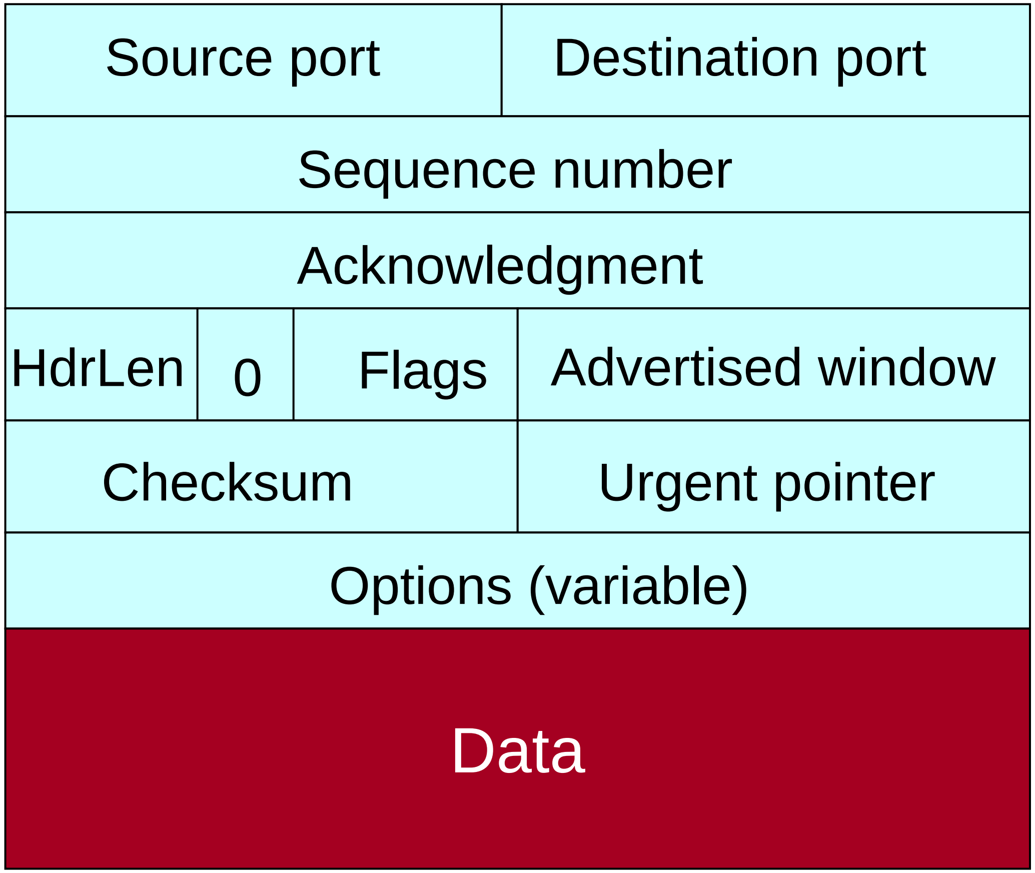


Each host tells its ISN to the other host.

- Three-way handshake to establish connection
  - Host A sends a **SYN** (open) to the host B
  - Host B returns a SYN acknowledgment (**SYN ACK**)
  - Host A sends an **ACK** to acknowledge the SYN ACK

TCP Header

Flags: SYN
FIN
RST
PSH
URG
ACK



Source port

Destination port

Sequence number

Acknowledgment

HdrLen

0

Flags

Advertised window

Checksum

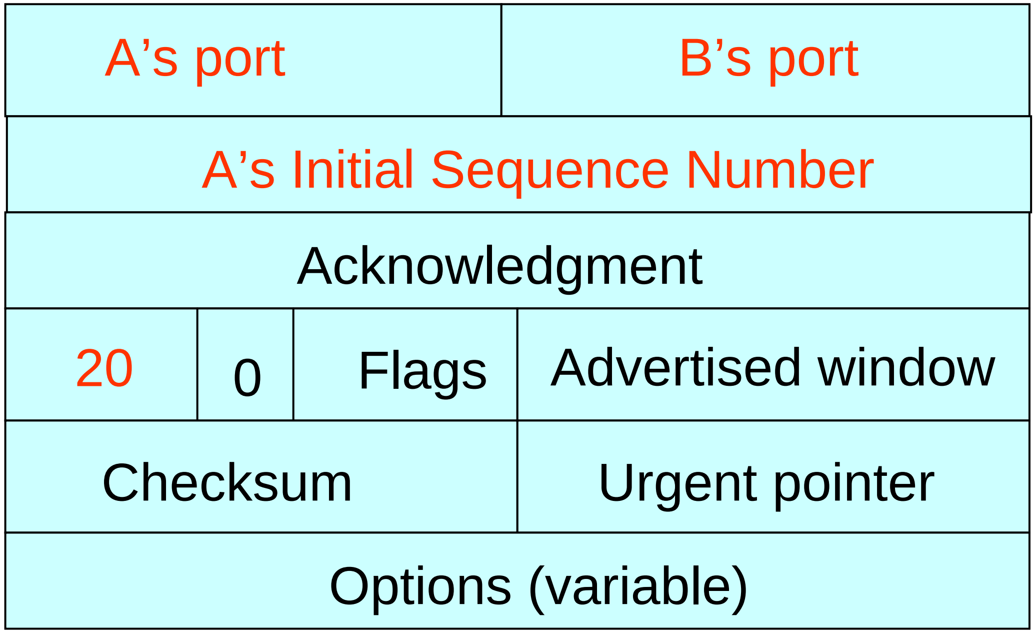
Urgent pointer

Options (variable)

Data

Step 1: A’s Initial SYN Packet

Flags: SYN
FIN
RST
PSH
URG
ACK

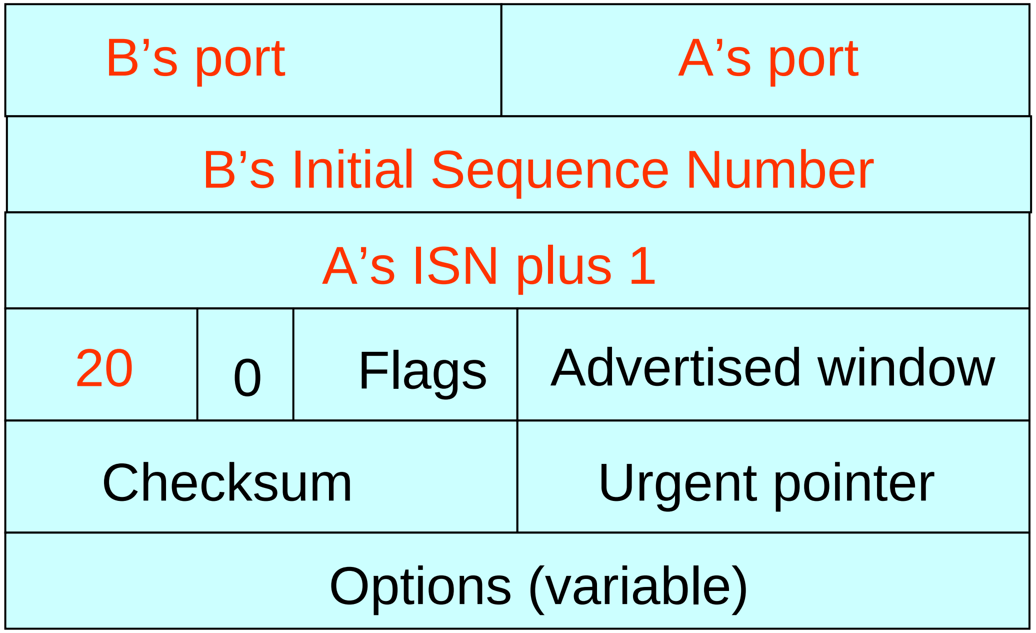


A’s port
B’s port
A’s Initial Sequence Number
Acknowledgment
20
0
Flags
Advertised window
Checksum
Urgent pointer
Options (variable)

A tells B it wants to open a connection…

Step 2: B’s SYN-ACK Packet

Flags: SYN
FIN
RST
PSH
URG
ACK



B’s port | A’s port

B’s Initial Sequence Number

A’s ISN plus 1

20 | 0 | Flags | Advertised window

Checksum | Urgent pointer

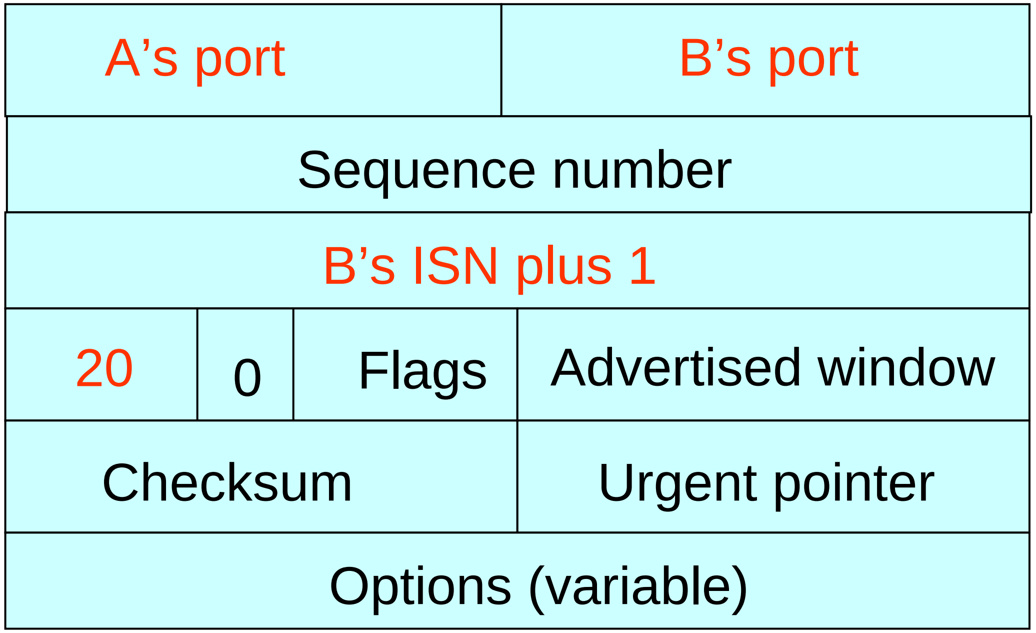
Options (variable)

B tells A it accepts, and is ready to hear the next byte…

… upon receiving this packet, A can start sending data

Step 3: A's ACK of the SYN-ACK

Flags:
- SYN
- FIN
- RST
- PSH
- URG
- ACK


A's port

B's port

Sequence number

B's ISN plus 1

20

0

Flags

Advertised window

Checksum

Urgent pointer

Options (variable)

A tells B it wants is okay to start sending

… upon receiving this packet, B can start sending data

What if the SYN Packet Gets Lost?

- Suppose the SYN packet gets lost
  - Packet is lost inside the network, or
  - Server rejects the packet (e.g., listen queue is full)
- Eventually, no SYN-ACK arrives
  - Sender sets a timer and wait for the SYN-ACK
  - … and retransmits the SYN if needed
- How should the TCP sender set the timer?
  - Sender has no idea how far away the receiver is
  - Hard to guess a reasonable length of time to wait
  - Some TCPs use a default of 3 or 6 seconds

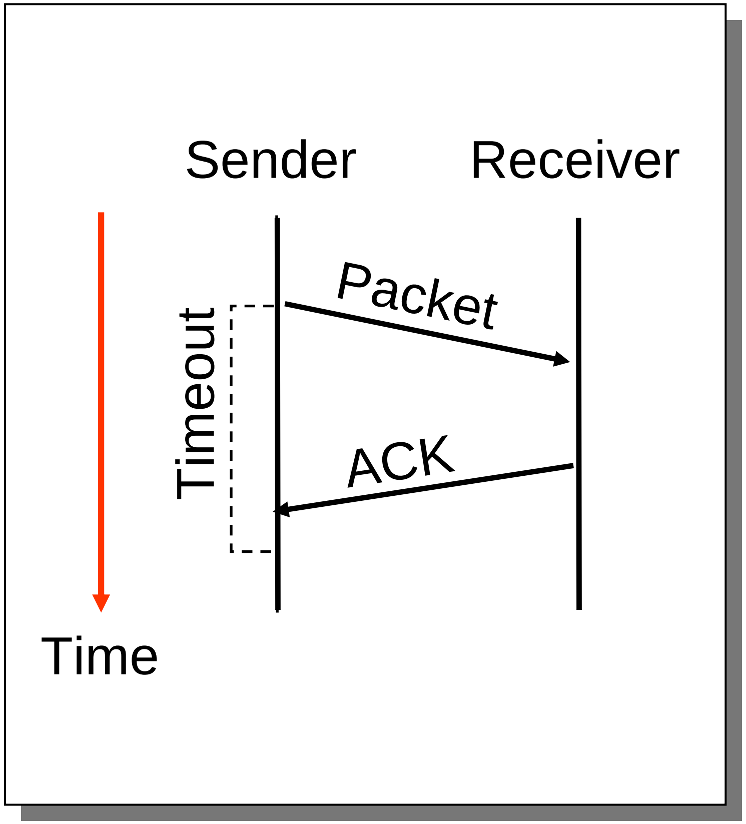
SYN Loss and Web Downloads

- User clicks on a hypertext link
  - Browser creates a socket and does a “connect”
  - The “connect” triggers the OS to transmit a SYN
- If the SYN is lost…
  - The 3-6 seconds of delay may be very long
  - The user may get impatient
  - … and click the hyperlink again, or click “reload”
- User triggers an “abort” of the “connect”
  - Browser creates a new socket and does a “connect”
  - Essentially, forces a faster send of a new SYN packet!
  - Sometimes very effective, and the page comes fast

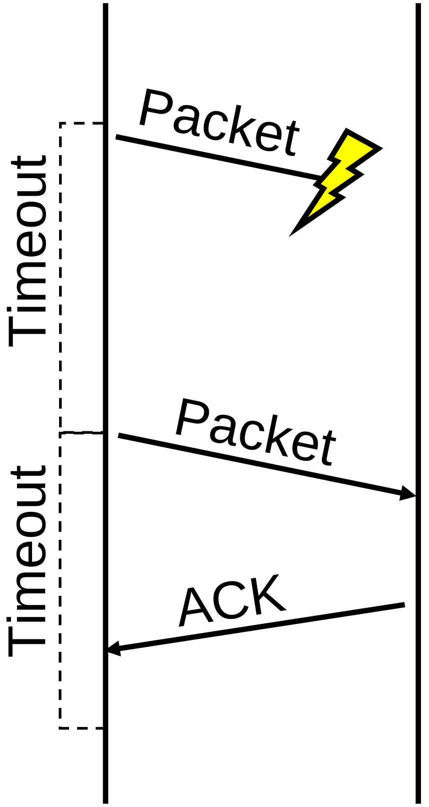
TCP Retransmissions

Automatic Repeat reQuest (ARQ)

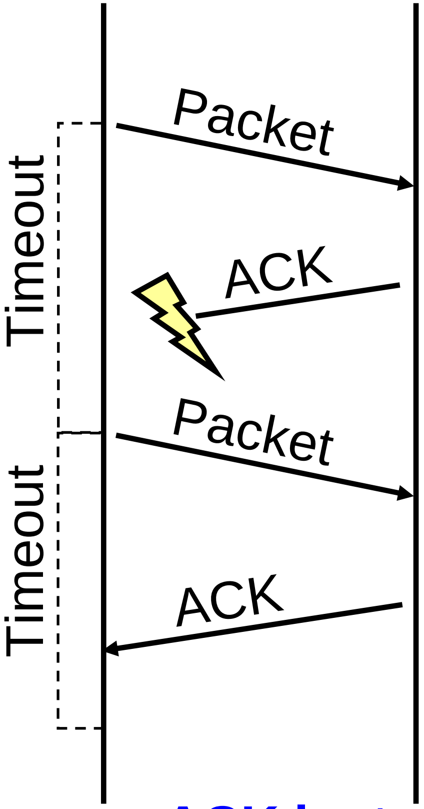
- Automatic Repeat Request
  - Receiver sends acknowledgment (ACK) when it receives packet
  - Sender waits for ACK and timeouts if it does not arrive within some time period
- Simplest ARQ protocol
  - Stop and wait
  - Send a packet, stop and wait until ACK arrives



Reasons for Retransmission

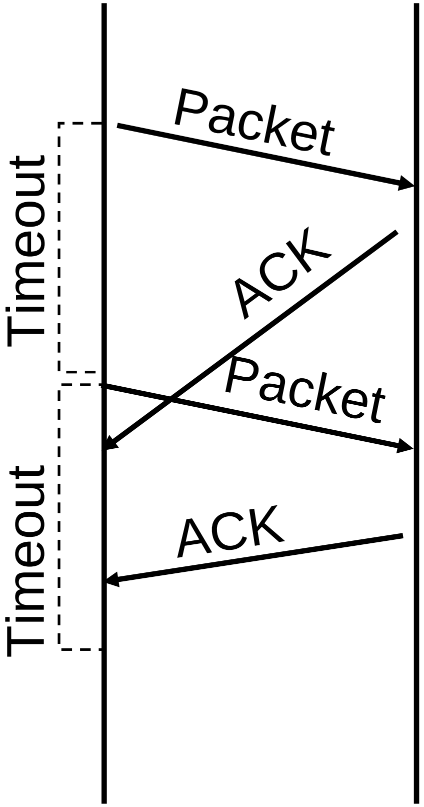


Packet lost



ACK lost

DUPLICATE
PACKET



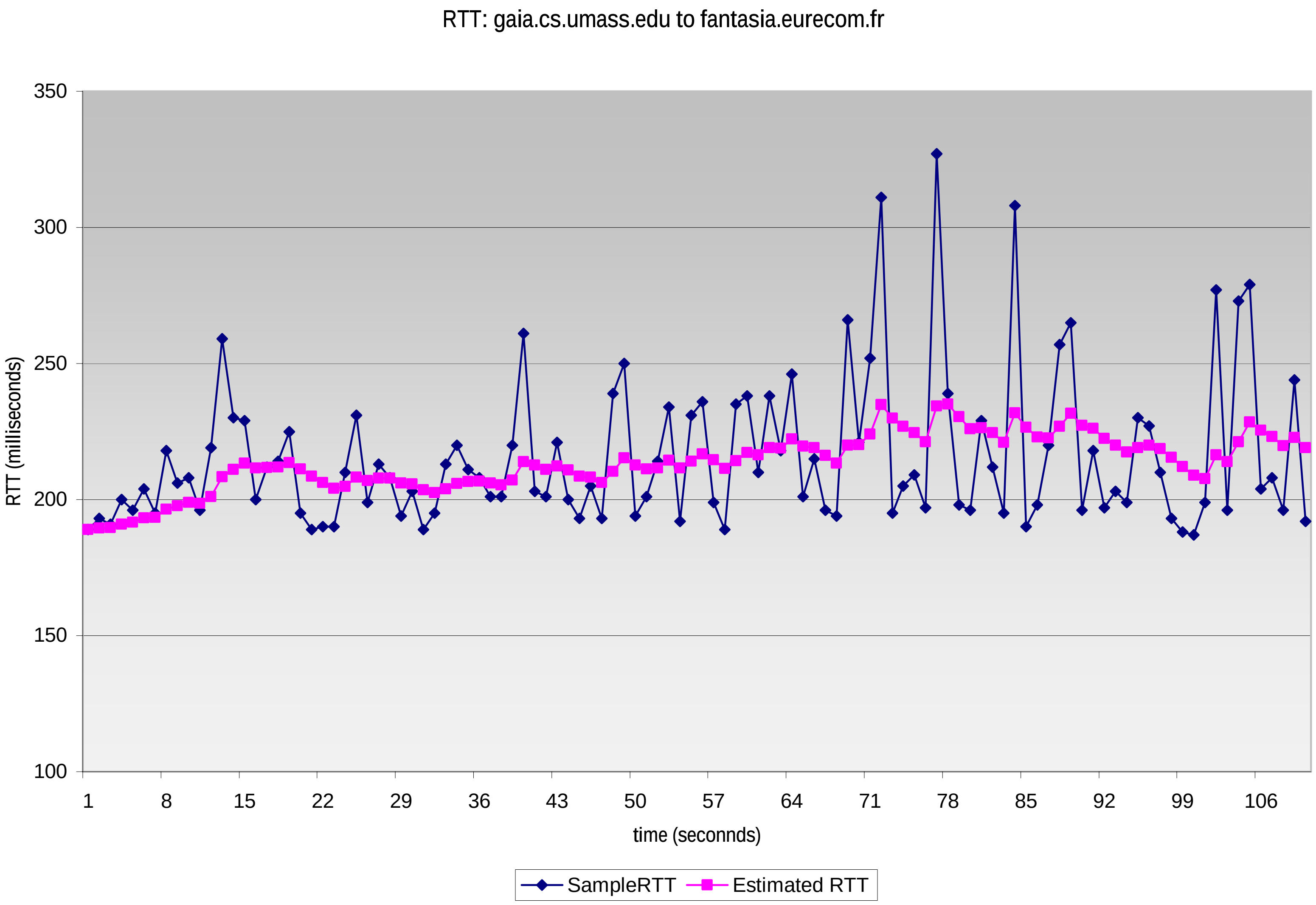
Early timeout

DUPLICATE
PACKETS

How Long Should Sender Wait?

- Sender sets a timeout to wait for an ACK
  - Too short: wasted retransmissions
  - Too long: excessive delays when packet lost
- TCP sets timeout as a function of the RTT
  - Expect ACK to arrive after an RTT
  - … plus a fudge factor to account for queuing
- But, how does the sender know the RTT?
  - Can estimate the RTT by watching the ACKs
  - Smooth estimate: keep a running average of the RTT
    - EstimatedRTT = a * EstimatedRTT + (1 –a) * SampleRTT
  - Compute timeout: TimeOut = 2 * EstimatedRTT

Example RTT Estimation



A Flaw in This Approach

- An ACK doesn’t really acknowledge a transmission
  - Rather, it acknowledges receipt of the data
- Consider a retransmission of a lost packet
  - If you assume the ACK goes with the 1st transmission
  - … the SampleRTT comes out way too large
- Consider a duplicate packet
  - If you assume the ACK goes with the 2nd transmission
  - … the Sample RTT comes out way too small
- Simple solution in the Karn/Partridge algorithm
  - Only collect samples for segments sent one single time

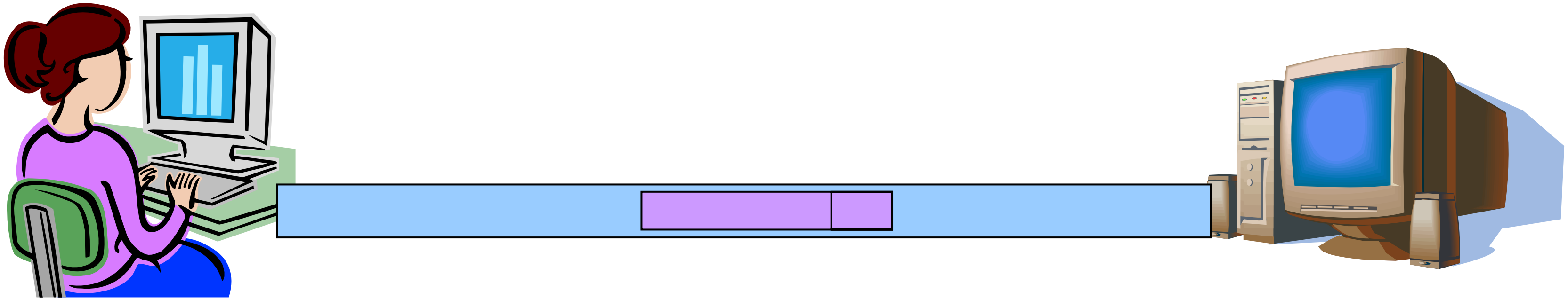
Yet Another Limitation…

- Doesn’t consider variance in the RTT
  - If variance is small, the EstimatedRTT is pretty accurate
  - … but, if variance is large, the estimate isn’t all that good
- Better to directly consider the variance
  - Consider difference: SampleRTT – EstimatedRTT
  - Boost the estimate based on the difference
- Jacobson/Karels algorithm
  - See Section 5.2 of the Peterson/Davie book for details

TCP Sliding Window

Motivation for Sliding Window

- Stop-and-wait is inefficient
  - Only one TCP segment is “in flight” at a time
  - Especially bad when delay-bandwidth product is high
- Numerical example
  - 1.5 Mbps link with a 45 msec round-trip time (RTT)
    - Delay-bandwidth product is 67.5 Kbits (or 8 KBytes)
  - But, sender can send at most one packet per RTT
    - Assuming a segment size of 1 KB (8 Kbits)
    - … leads to 8 Kbits/segment / 45 msec/segment → 182 Kbps
    - That’s just one-eighth of the 1.5 Mbps link capacity



Performance of Stop & Wait

- Example: 1 Gbps link, 15 ms e-e prop. delay, 1KB packet:

$$T_{\text{transmit}} = \frac{\text{L (packet length in bits)}}{\text{R (transmission rate, bps)}} = \frac{\text{8kb/pkt}}{\text{10**9 b/sec}} = \text{8 microsec}$$

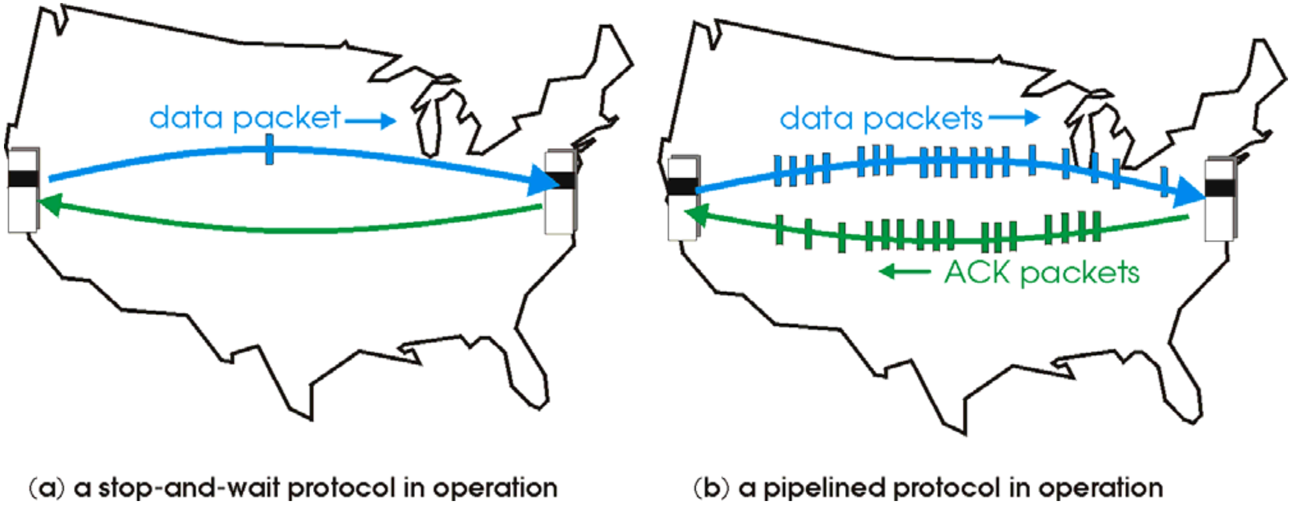
$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

- U $_{\text{sender}}$: utilization – fraction of time sender busy sending
- 1KB pkt every 30 msec -> 33kB/sec thruput over 1 Gbps link
- network protocol limits use of physical resources!

Pipelined protocols

Pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts

- range of sequence numbers must be increased
- buffering at sender and/or receiver

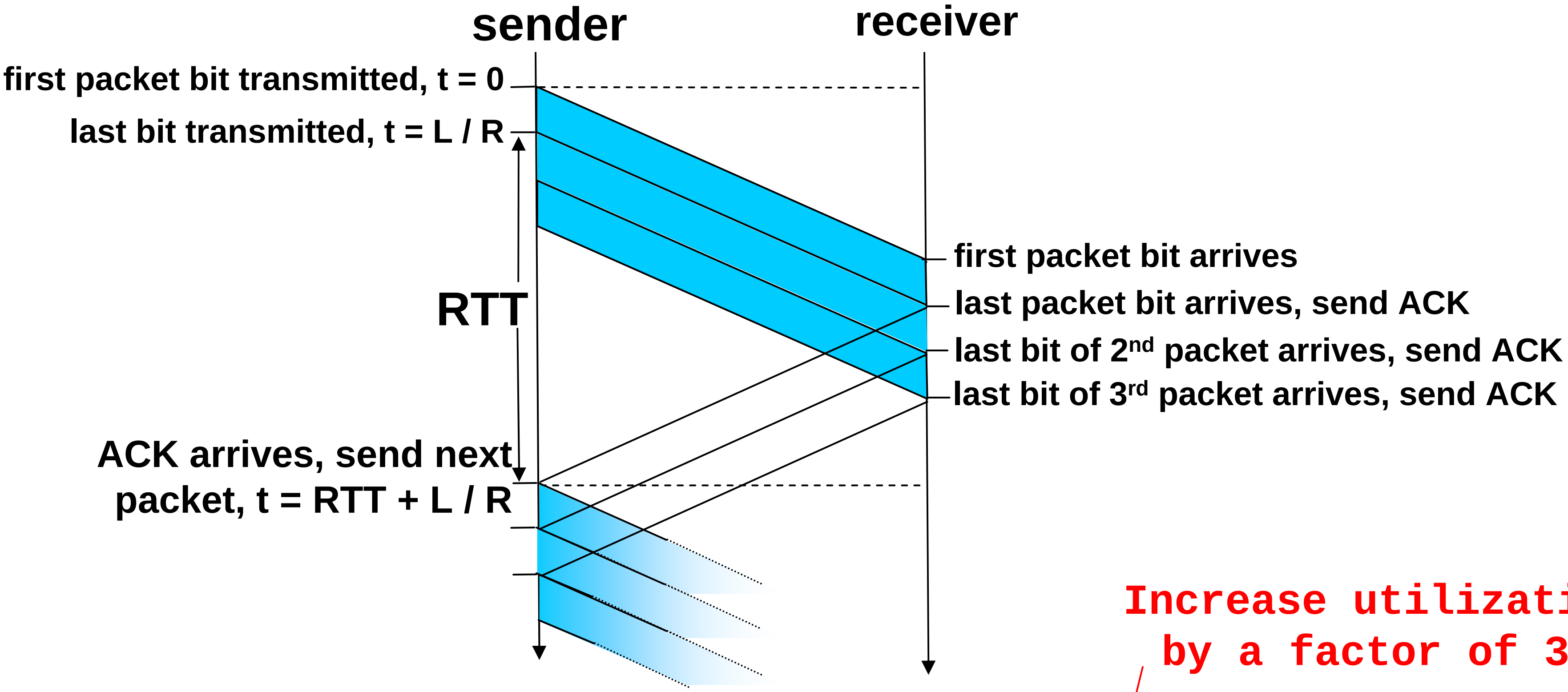


(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

- Two generic forms of pipelined protocols: *go-Back-N, selective repeat*

Pipelining: increased utilization



sender

receiver

first packet bit transmitted, t = 0

last bit transmitted, t = L / R

RTT

ACK arrives, send next packet, t = RTT + L / R

first packet bit arrives

last packet bit arrives, send ACK

last bit of 2nd packet arrives, send ACK

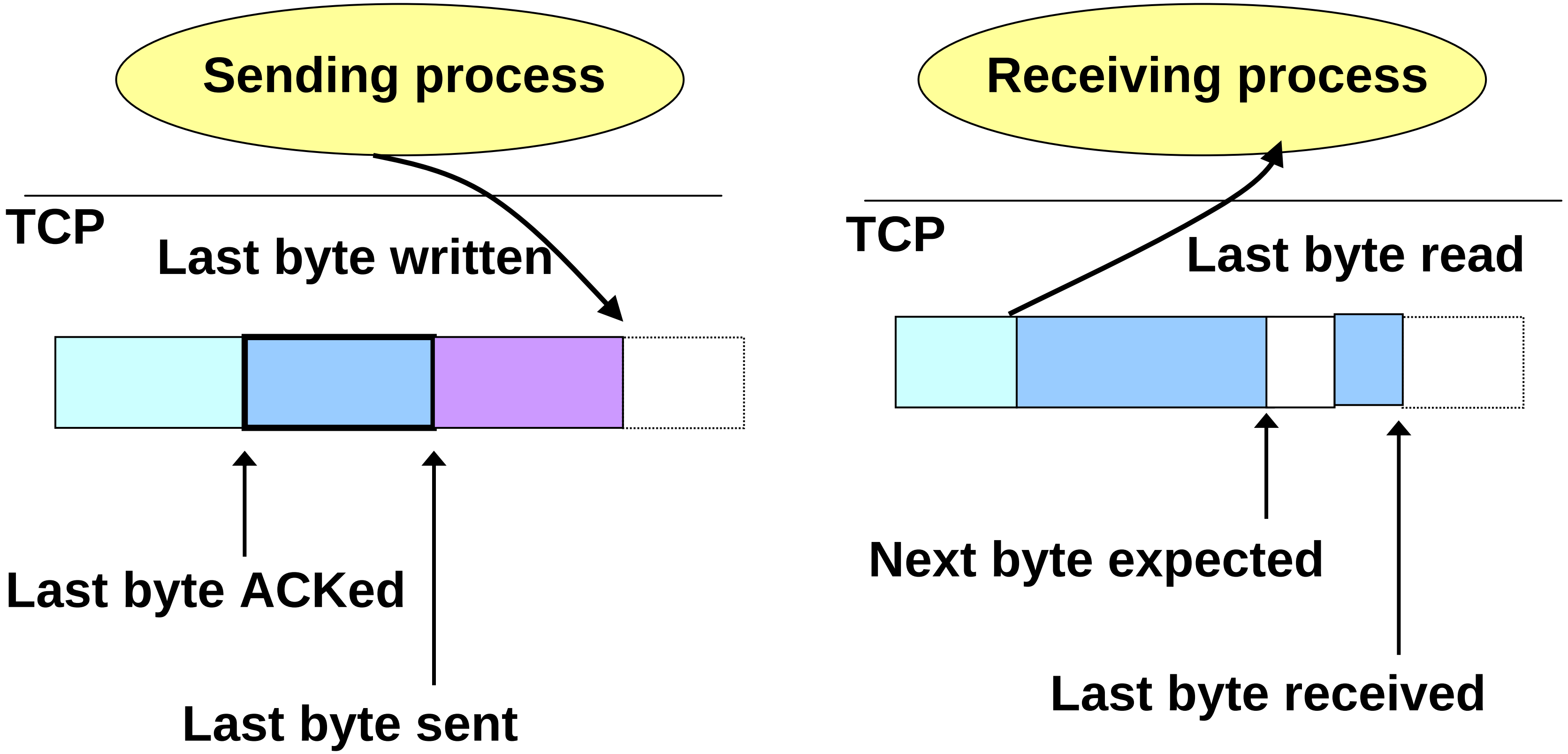
last bit of 3rd packet arrives, send ACK

$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

Increase utilization by a factor of 3!

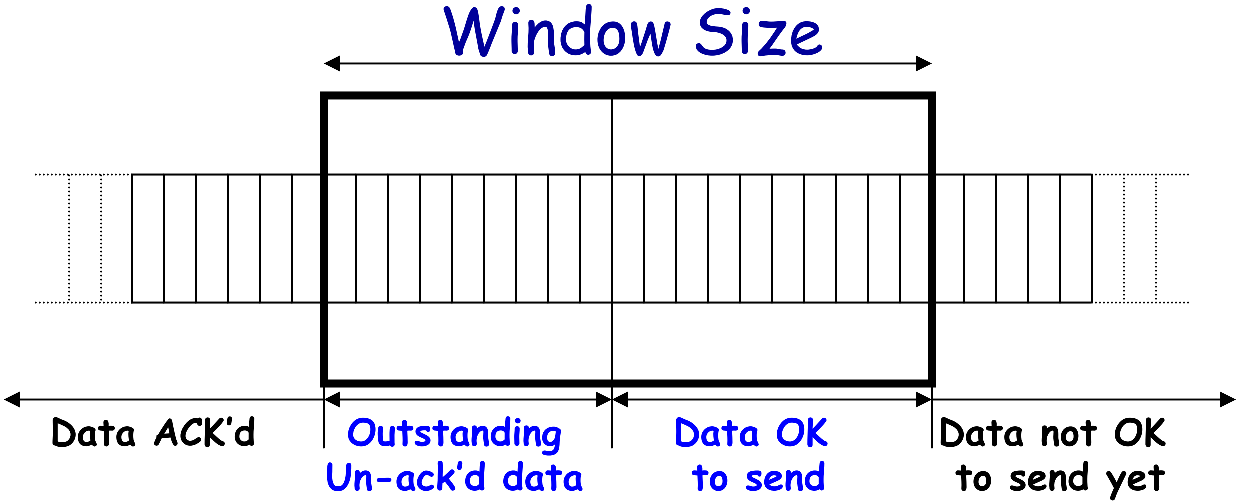
Sliding Window

- Allow a larger amount of data “in flight”
  - Allow sender to get ahead of the receiver
  - … though not *too far* ahead



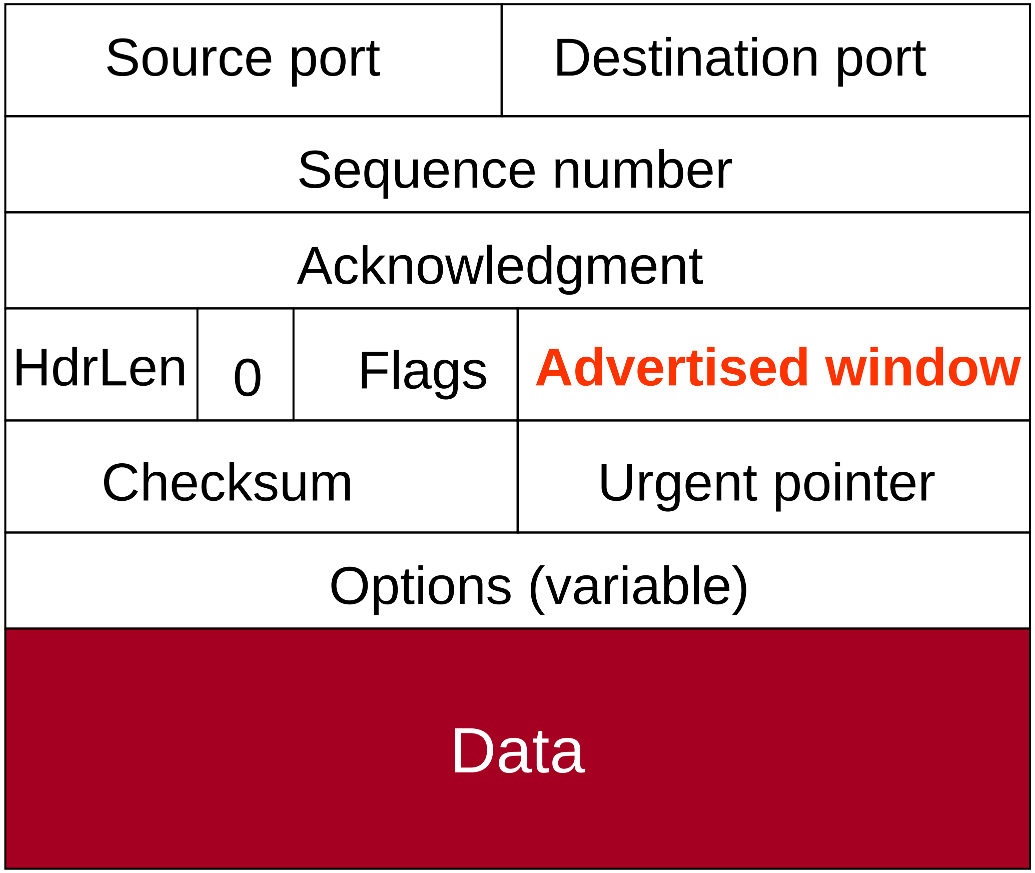
Receiver Buffering

- Window size
  - Amount that can be sent without acknowledgment
  - Receiver needs to be able to store this amount of data
- Receiver advertises the window to the receiver
  - Tells the receiver the amount of free space left
  - … and the sender agrees not to exceed this amount



TCP Header for Receiver Buffering

Flags: SYN
FIN
RST
PSH
URG
ACK



Source port

Destination port

Sequence number

Acknowledgment

HdrLen

0

Flags

Advertised window

Checksum

Urgent pointer

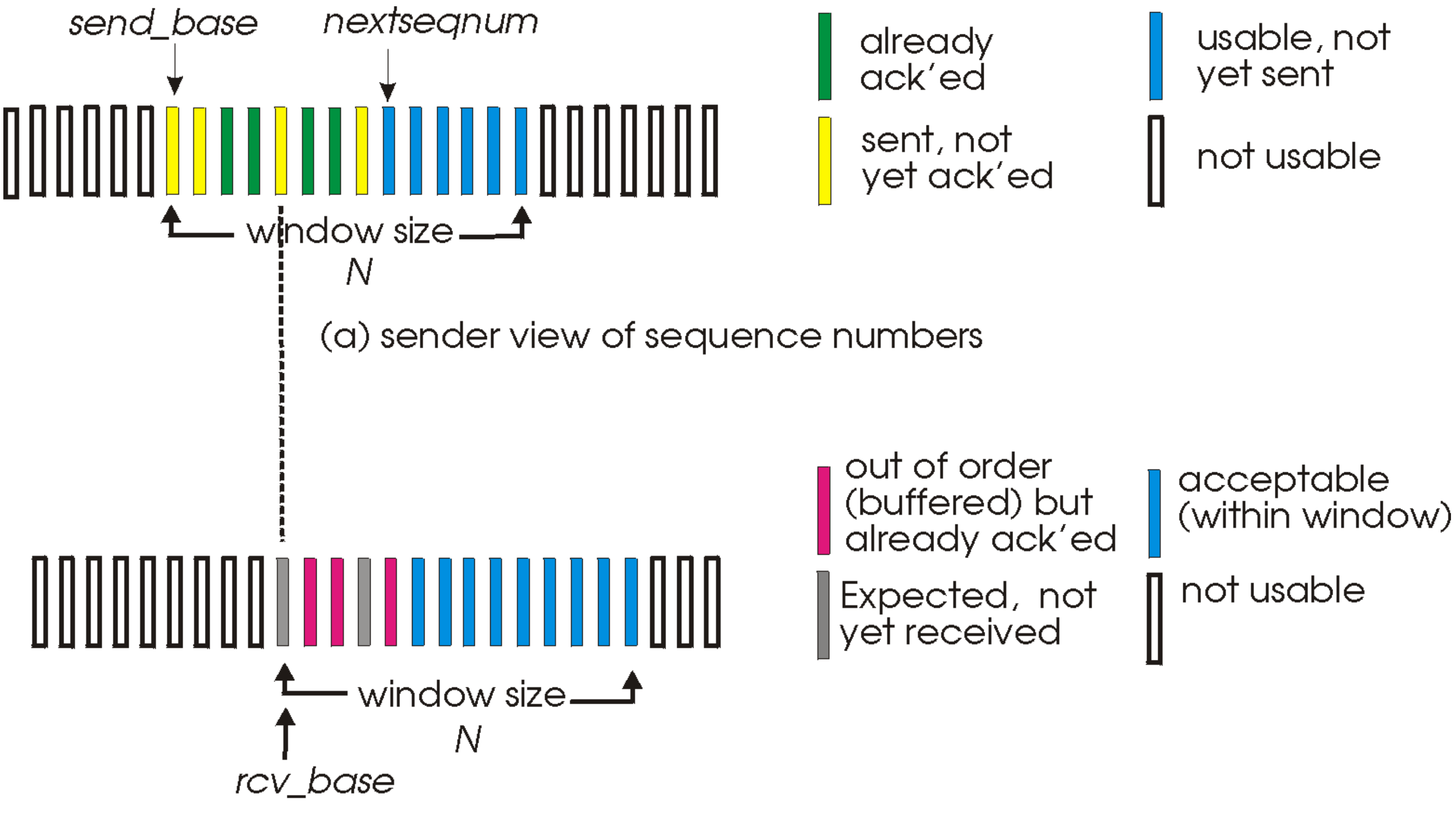
Options (variable)

Data

Selective Repeat

- receiver *individually* acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt
- sender window
  - N consecutive seq #’s
  - again limits seq #s of sent, unACKed pkts

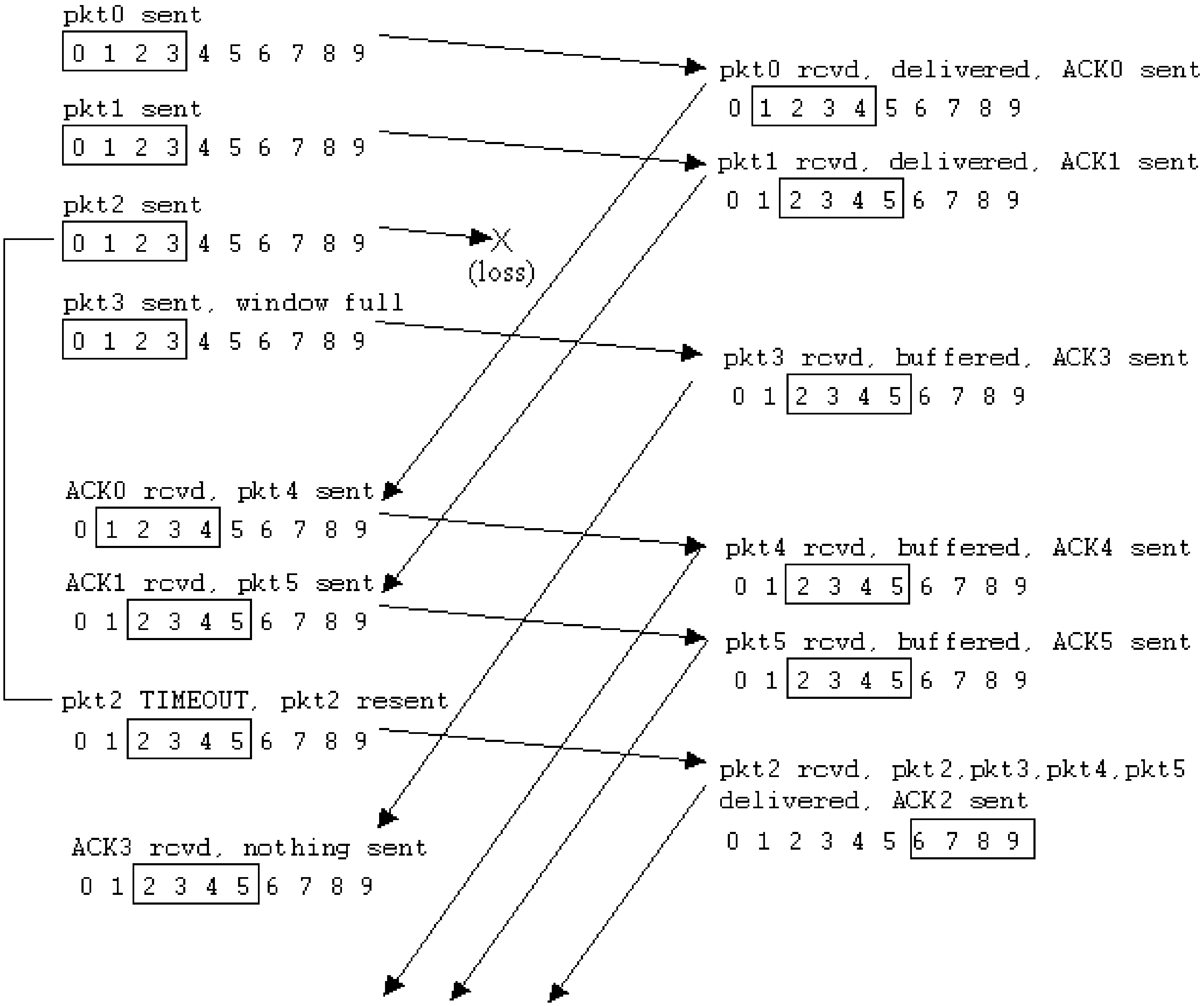
Selective repeat: sender, receiver windows



(a) sender view of sequence numbers

(b) receiver view of sequence numbers

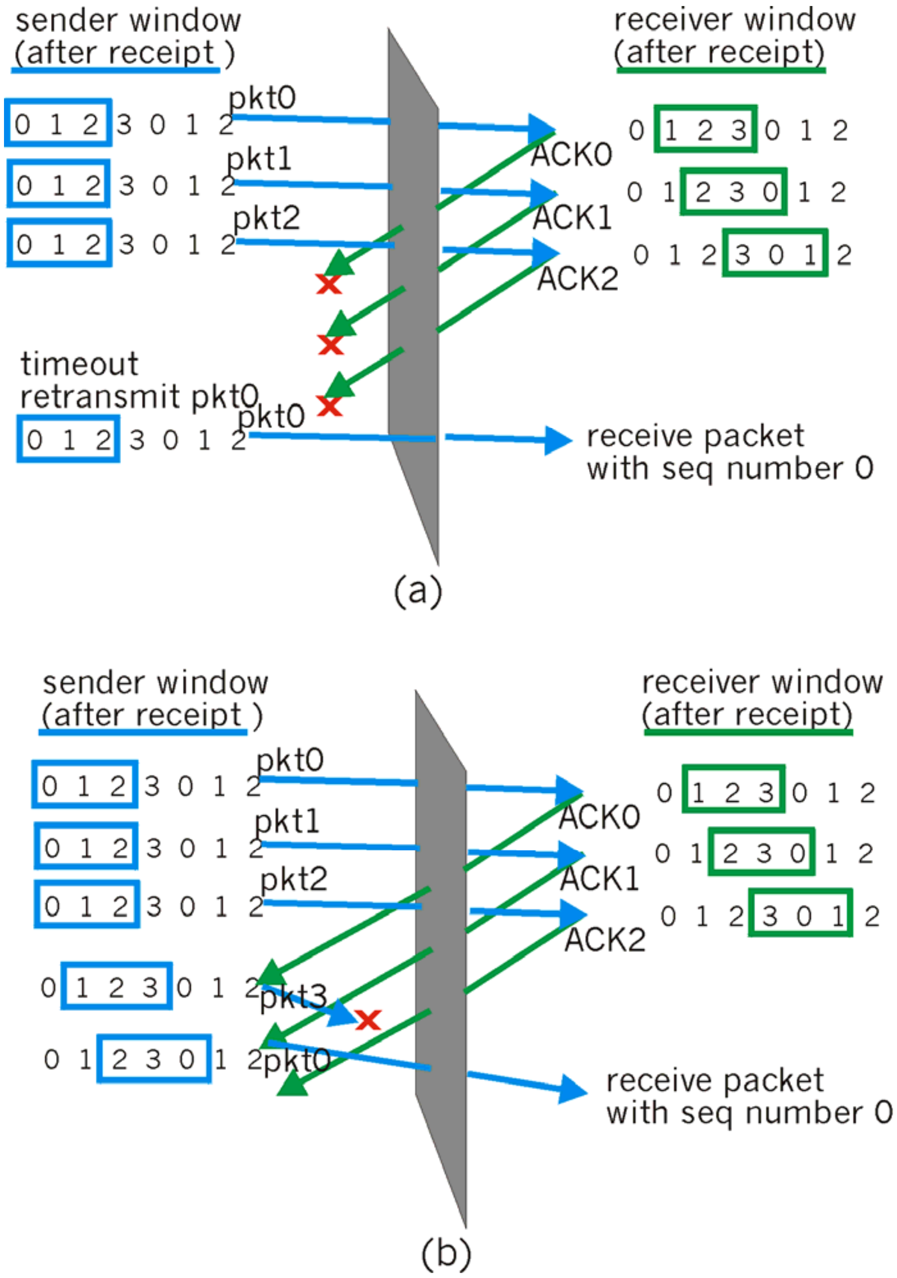
Selective repeat in action



Selective repeat: dilemma

Example:

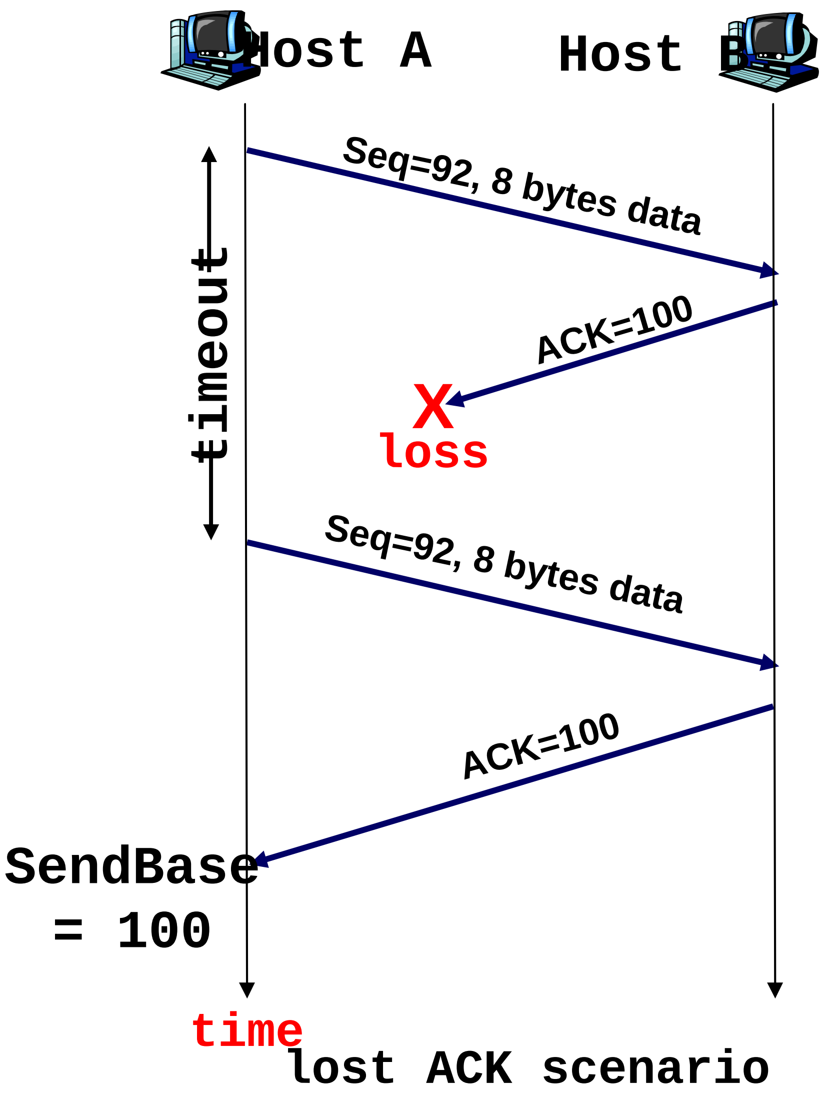
- seq #'s: 0, 1, 2, 3
- window size=3

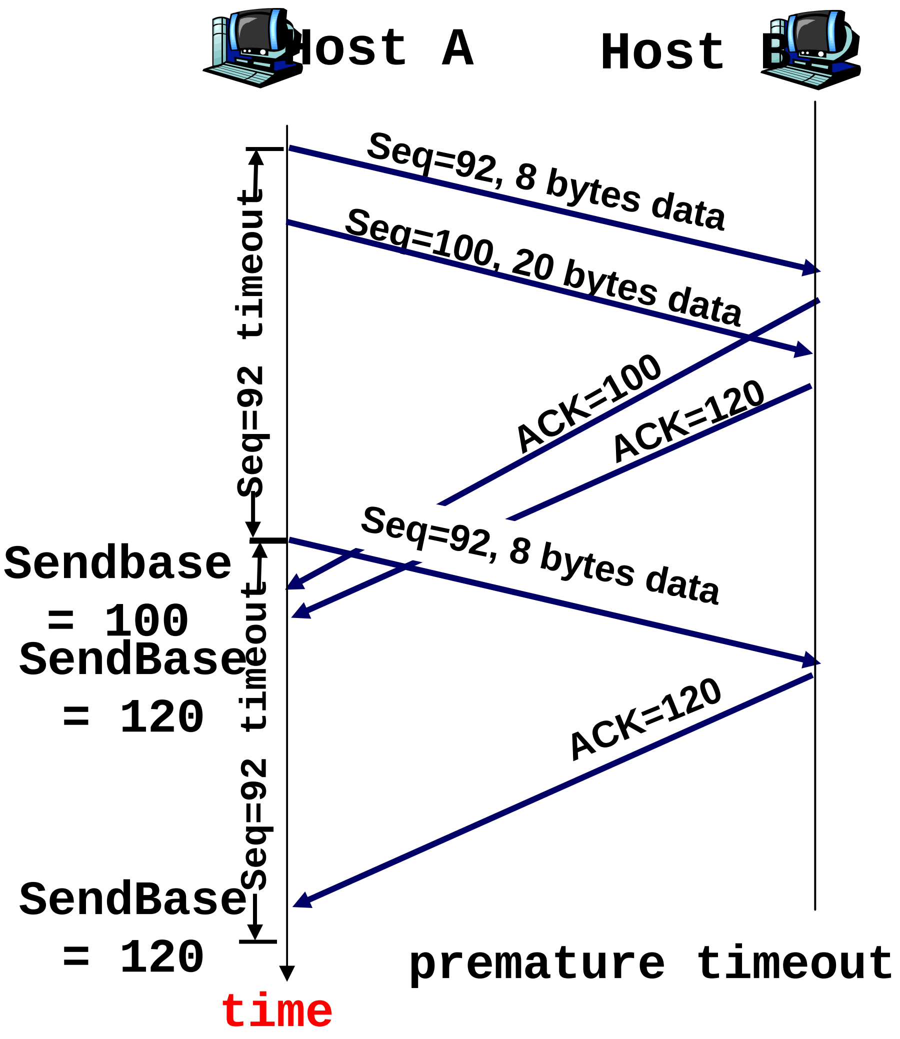


- receiver sees no difference in two scenarios!
- incorrectly passes duplicate data as new in (a)

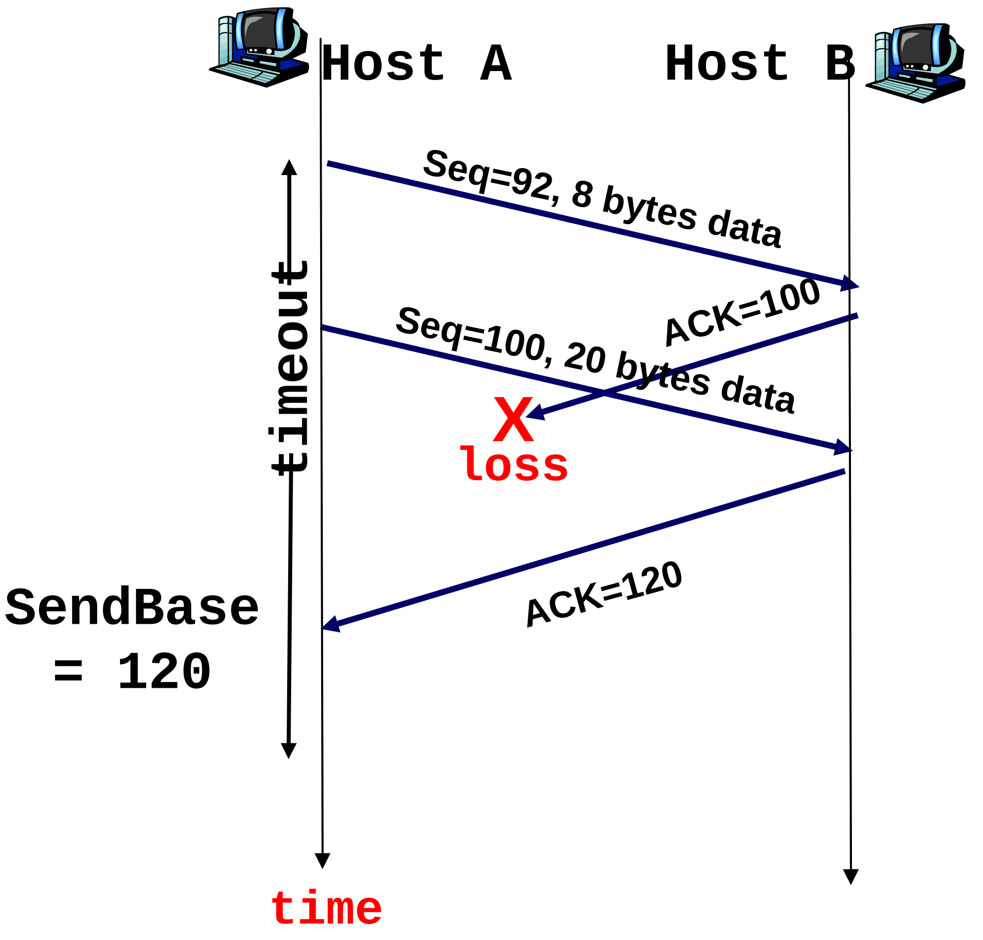
Q: what relationship between seq # size and window size?

TCP: retransmission scenarios





TCP retransmission scenarios (more)



Cumulative ACK scenario

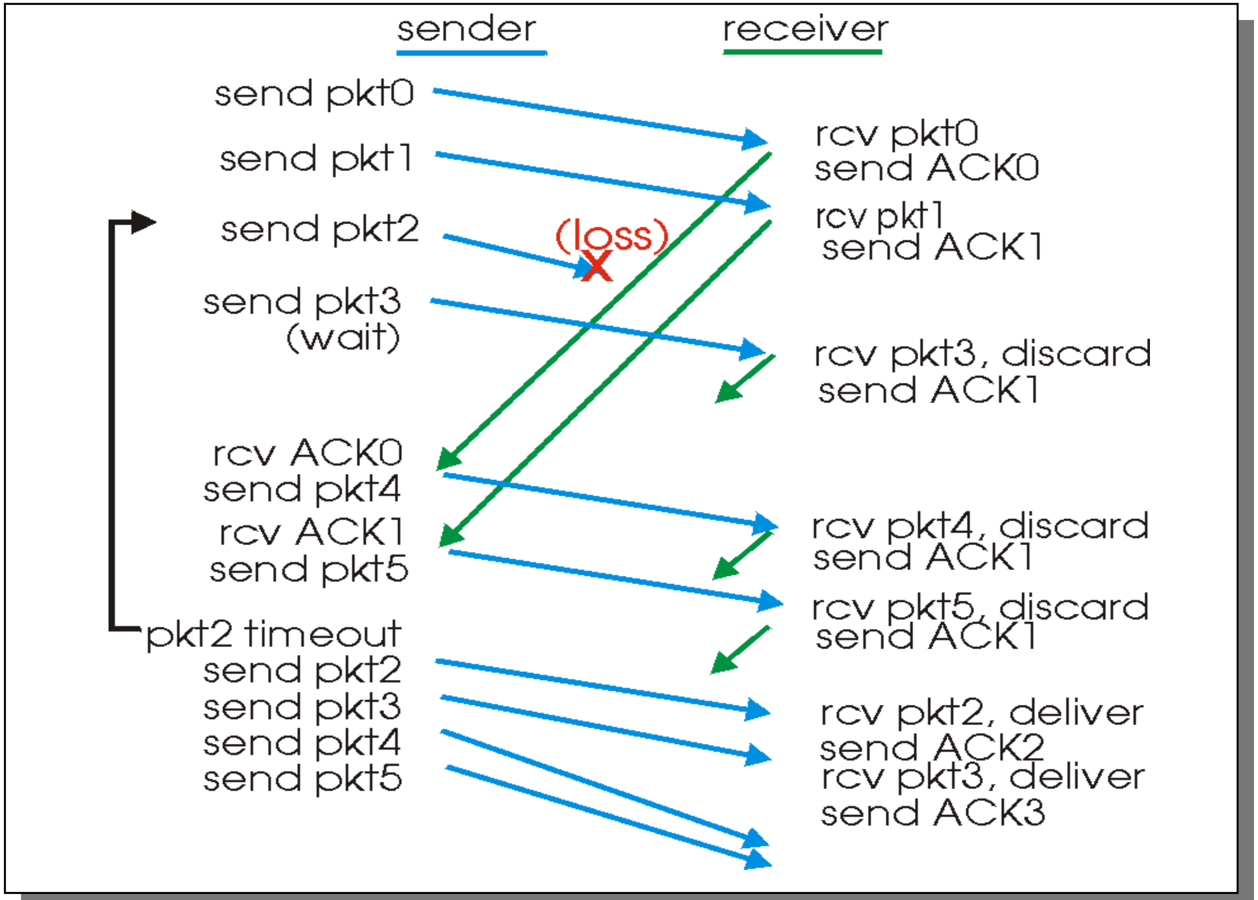
TCP ACK generation [RFC 1122, RFC 2581]

Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq. # . Gap detected	Immediately send duplicate ACK, indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediately send ACK, provided that segment starts at lower end of gap

Fast Retransmission

Timeout is Inefficient

- Timeout-based retransmission
  - Sender transmits a packet and waits until timer expires
  - … and then retransmits from the lost packet onward



Fast Retransmission

- Better solution possible under sliding window
  - Although packet n might have been lost
  - … packets n+1, n+2, and so on might get through
- Idea: have the receiver send ACK packets
  - ACK says that receiver is still awaiting nth packet
    - And *repeated* ACKs suggest later packets have arrived
  - Sender can view the “duplicate ACKs” as an early hint
    - … that the nth packet must have been lost
    - … and perform the retransmission early
- Fast retransmission
  - Sender retransmits data after the triple duplicate ACK

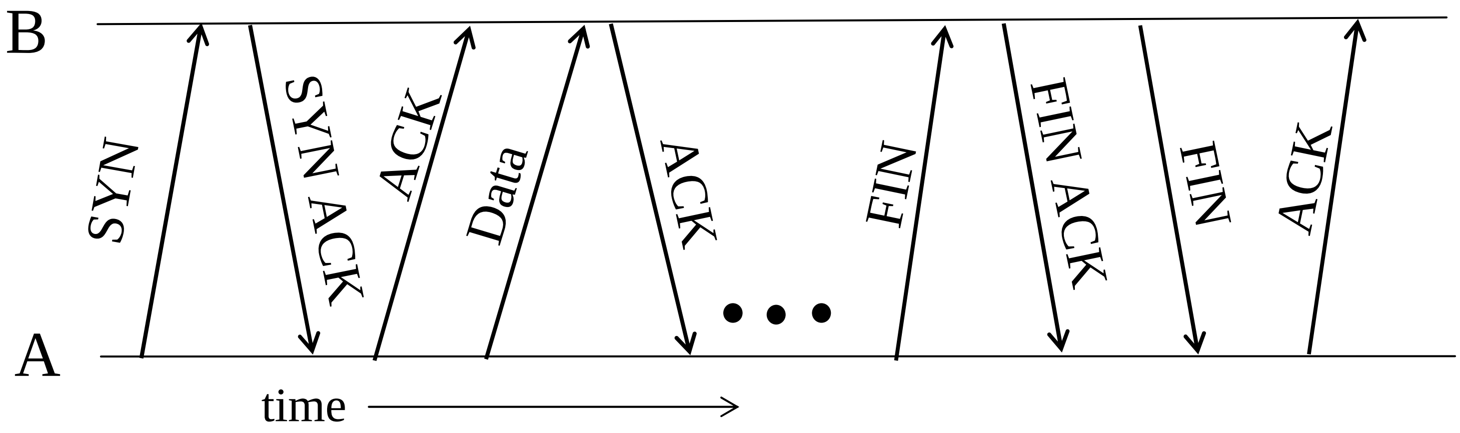
Effectiveness of Fast Retransmit

- When does Fast Retransmit work best?
  - Long data transfers
    - High likelihood of many packets in flight
  - High window size
    - High likelihood of many packets in flight
  - Low burstiness in packet losses
    - Higher likelihood that later packets arrive successfully

- Implications for Web traffic
  - Most Web transfers are short (e.g., 10 packets)
    - Short HTML files or small images
  - So, often there aren’t many packets in flight
  - … making fast retransmit less likely to “kick in”
  - Forcing users to like “reload” more often… ☺

Tearing Down the Connection

Tearing Down the Connection



- Closing the connection
  - Finish (FIN) to close and receive remaining bytes
  - And other host sends a FIN ACK to acknowledge
  - Reset (RST) to close and not receive remaining bytes

Sending/Receiving the FIN Packet

- Sending a FIN: close()
  - Process is done sending data via the socket
  - Process invokes “close()” to close the socket
  - Once TCP has sent all of the outstanding bytes…
  - … then TCP sends a FIN

- Receiving a FIN: EOF
  - Process is reading data from the socket
  - Eventually, the attempt to read returns an EOF

Conclusions

- Transport protocols
  - Multiplexing and demultiplexing
  - Sequence numbers
  - Window-based flow control
  - Timer-based retransmission
  - Checksum-based error detection
- Reading for this week
  - Sections 2.5, 5.1-5.2, and 6.1-6.4
- Next lecture
  - Congestion control

Congestion Control

Sections 6.1-6.4

Introduction to Data Networks

2008.4

Goals of Today’s Lecture

- Principles of congestion control
  - Learning that congestion is occurring
  - Adapting to alleviate the congestion
- TCP congestion control
  - Additive-increase, multiplicative-decrease
  - Slow start and slow-start restart
- Related TCP mechanisms
  - Nagle’s algorithm and delayed acknowledgments
- Active Queue Management (AQM)
  - Random Early Detection (RED)
  - Explicit Congestion Notification (ECN)

Resource Allocation vs. Congestion Control

- Resource allocation
  - How nodes meet competing demands for resources
  - E.g., link bandwidth and buffer space
  - When to say no, and to whom
- Congestion control
  - How nodes prevent or respond to overload conditions
  - E.g., persuade hosts to stop sending, or slow down
  - Typically has notions of fairness (i.e., sharing the pain)

Flow Control vs. Congestion Control

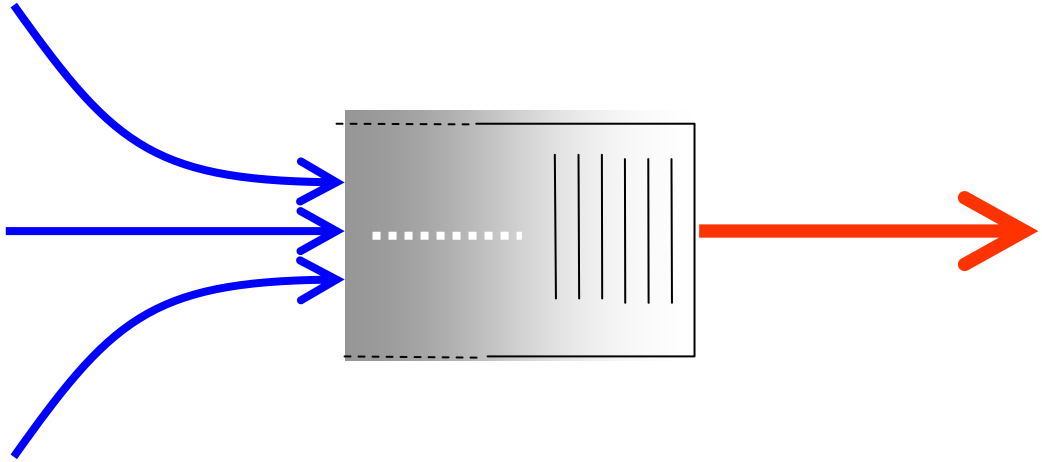
- Flow control
  - Keeping *one fast sender* from overwhelming *a slow receiver*
- Congestion control
  - Keep a *set of senders* from overloading the *network*
- Different concepts, but similar mechanisms
  - TCP flow control: receiver window
  - TCP congestion control: congestion window
  - TCP window: min{congestion window, receiver window}

Three Key Features of Internet

- Packet switching
  - A given source may have enough capacity to send data
  - … and yet the packets may encounter an overloaded link
- Connectionless flows
  - No notions of connections inside the network
  - … and no advance reservation of network resources
  - Still, you can view related packets as a group (“flow”)
  - … e.g., the packets in the same TCP transfer
- Best-effort service
  - No guarantees for packet delivery or delay
  - No preferential treatment for certain packets

Congestion is Unavoidable

- Two packets arrive at the same time
  - The node can only transmit one
  - … and either buffer or drop the other
- If many packets arrive in a short period of time
  - The node cannot keep up with the arriving traffic
  - … and the buffer may eventually overflow



Congestion Collapse

- Definition: Increase in network load results in a decrease of useful work done

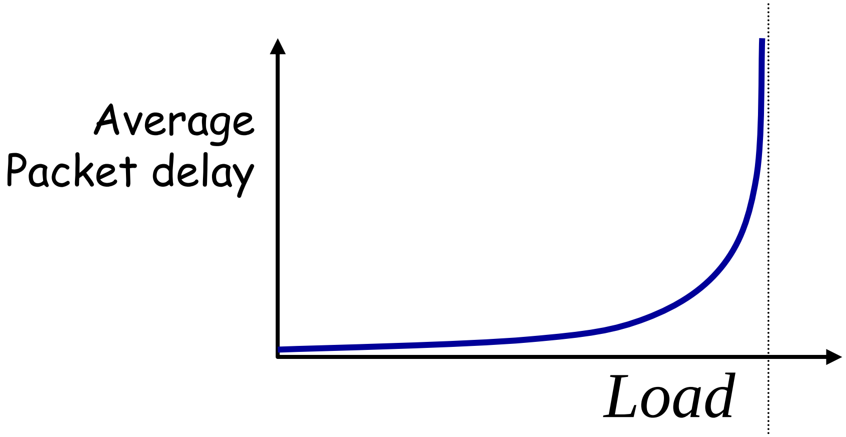
- Many possible causes
  - Spurious retransmissions of packets still in flight
    - Classical congestion collapse
    - Solution: better timers and TCP congestion control
  - Undelivered packets
    - Packets consume resources and are dropped elsewhere in network
    - Solution: congestion control for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

What Do We Want, Really?

- High throughput
  - Throughput: measured performance of a system
  - E.g., number of bits/second of data that get through
- Low delay
  - Delay: time required to deliver a packet or message
  - E.g., number of msec to deliver a packet
- These two metrics are sometimes at odds
  - E.g., suppose you drive a link as hard as possible
  - … then, throughput will be high, but delay will be, too

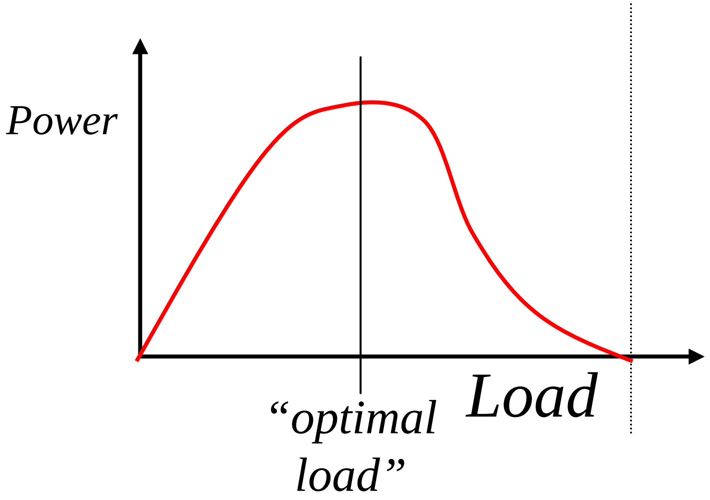
Load, Delay, and Power

Typical behavior of queuing systems with random arrivals:



A simple metric of how well the network is performing:

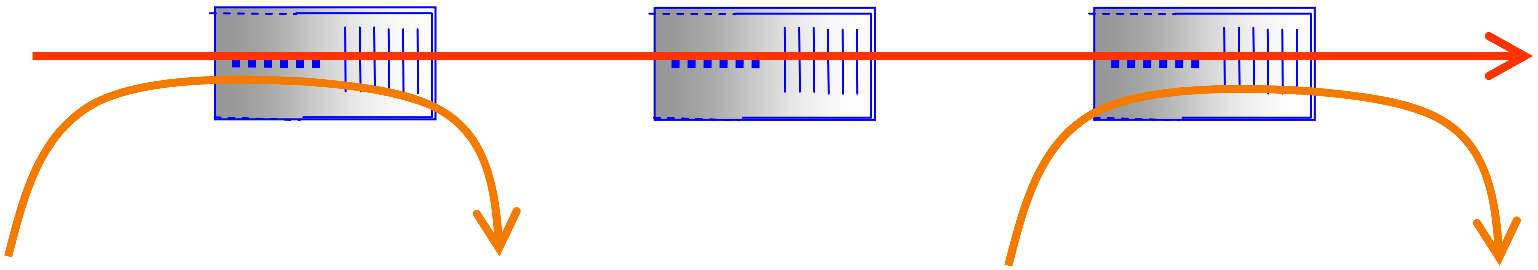
$$Power = \frac{Load}{Delay}$$



Goal: maximize power

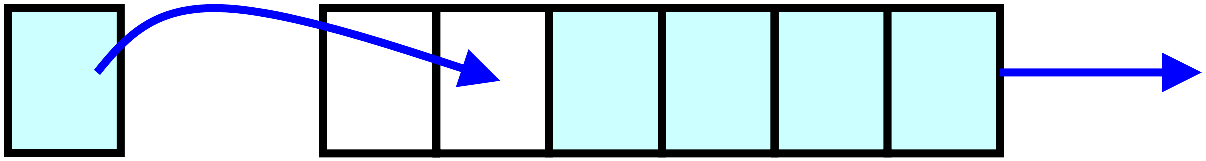
Fairness

- Effective utilization is not the only goal
  - We also want to be *fair* to the various flows
  - … but what the heck does *that* mean?
- Simple definition: equal shares of the bandwidth
  - N flows that each get 1/N of the bandwidth?
  - But, what if the flows traverse different paths?

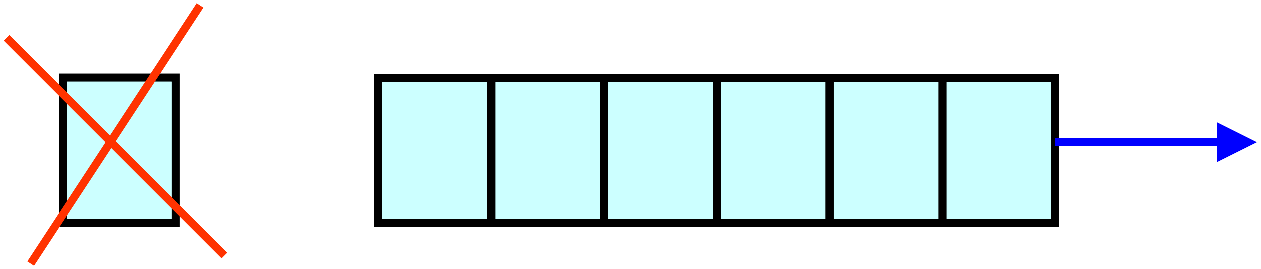


Simple Resource Allocation

- Simplest approach: FIFO queue and drop-tail
- Link bandwidth: first-in first-out queue
  - Packets transmitted in the order they arrive



- Buffer space: drop-tail queuing
  - If the queue is full, drop the incoming packet



Simple Congestion Detection

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

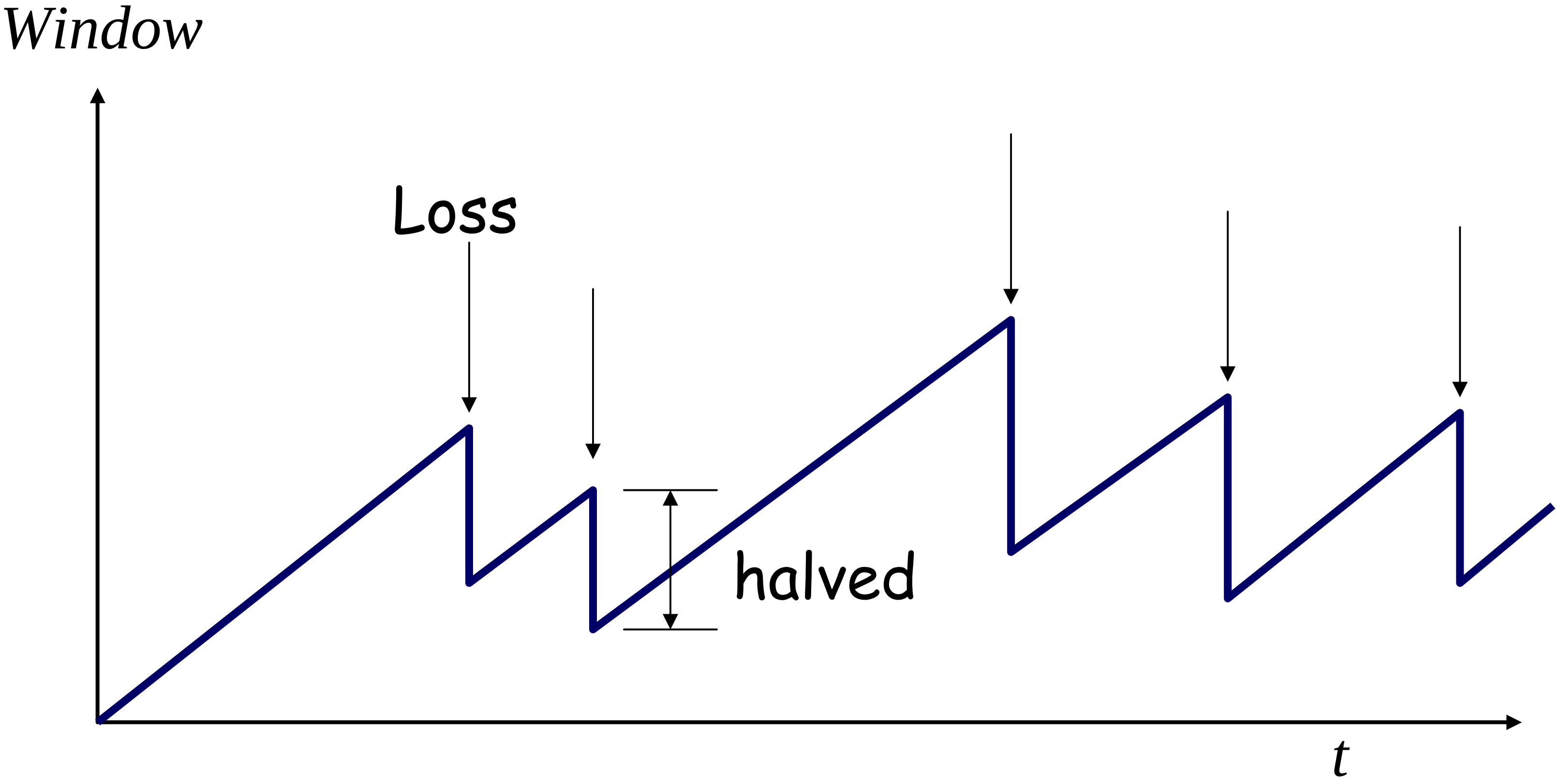
Idea of TCP Congestion Control

- Each source determines the available capacity
  - … so it knows how many packets to have in transit
- Congestion window
  - Maximum # of unacknowledged bytes to have in transit
  - The congestion-control equivalent of receiver window
  - MaxWindow = min{congestion window, receiver window}
  - Send at the rate of the slowest component
- Adapting the congestion window
  - Decrease upon losing a packet: backing off
  - Increase upon success: optimistically exploring

Additive Increase, Multiplicative Decrease

- How much to increase and decrease?
  - Increase linearly, decrease multiplicatively
  - A necessary condition for stability of TCP
  - Consequences of over-sized window are much worse than having an under-sized window
    - Over-sized window: packets dropped and retransmitted
    - Under-sized window: somewhat lower throughput
- Multiplicative decrease
  - On loss of packet, divide congestion window in half
- Additive increase
  - On success for last window of data, increase linearly

Leads to the TCP “Sawtooth”

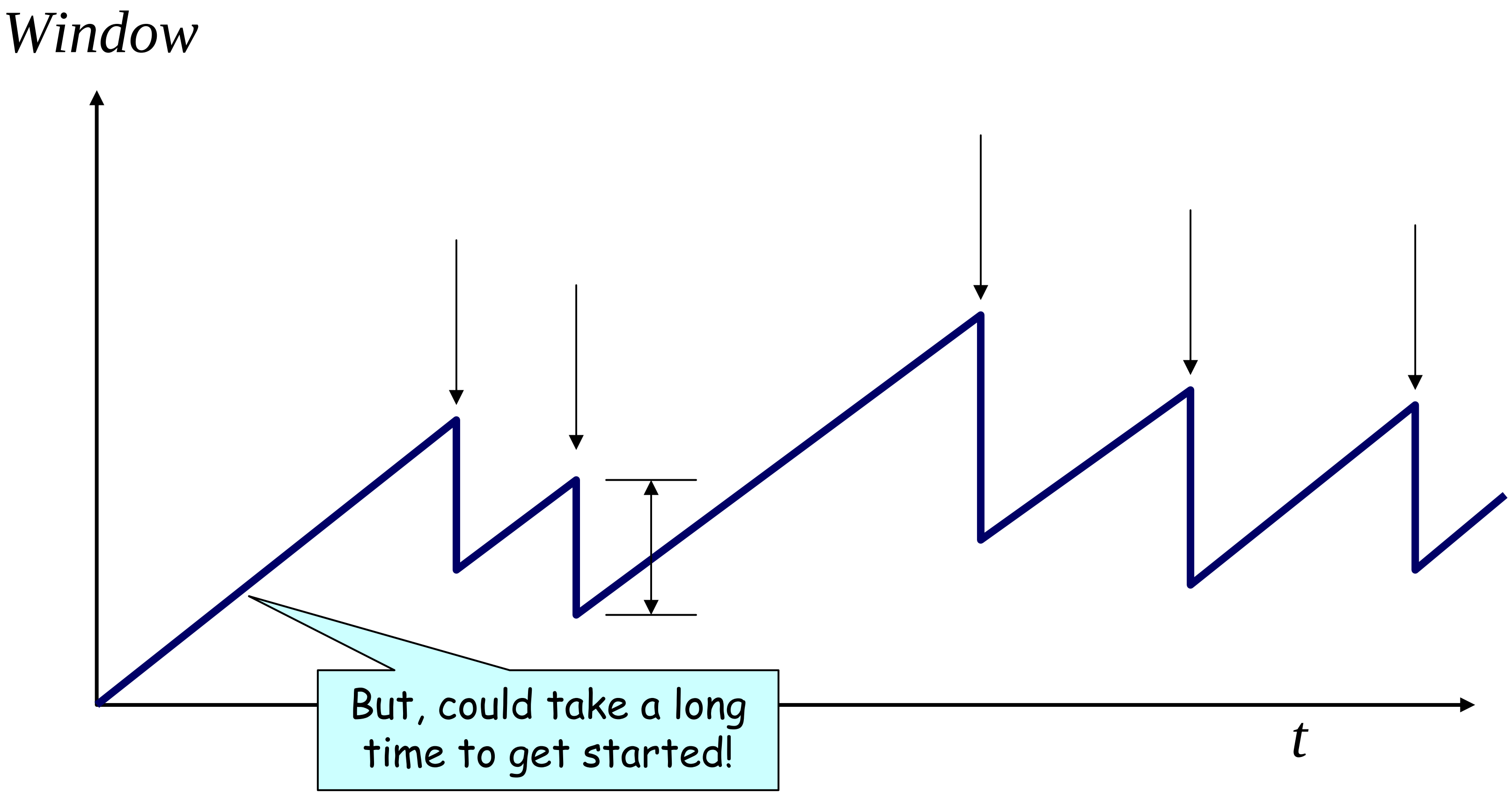


Practical Details

- Congestion window
  - Represented in bytes, not in packets (Why?)
  - Packets have MSS (Maximum Segment Size) bytes
- Increasing the congestion window
  - Increase by MSS on success for last window of data
  - In practice, increase a fraction of MSS per received ACK
    - # packets per window: CWND / MSS
    - Increment per ACK: MSS * (MSS / CWND)
- Decreasing the congestion window
  - Never drop congestion window below 1 MSS

Getting Started

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



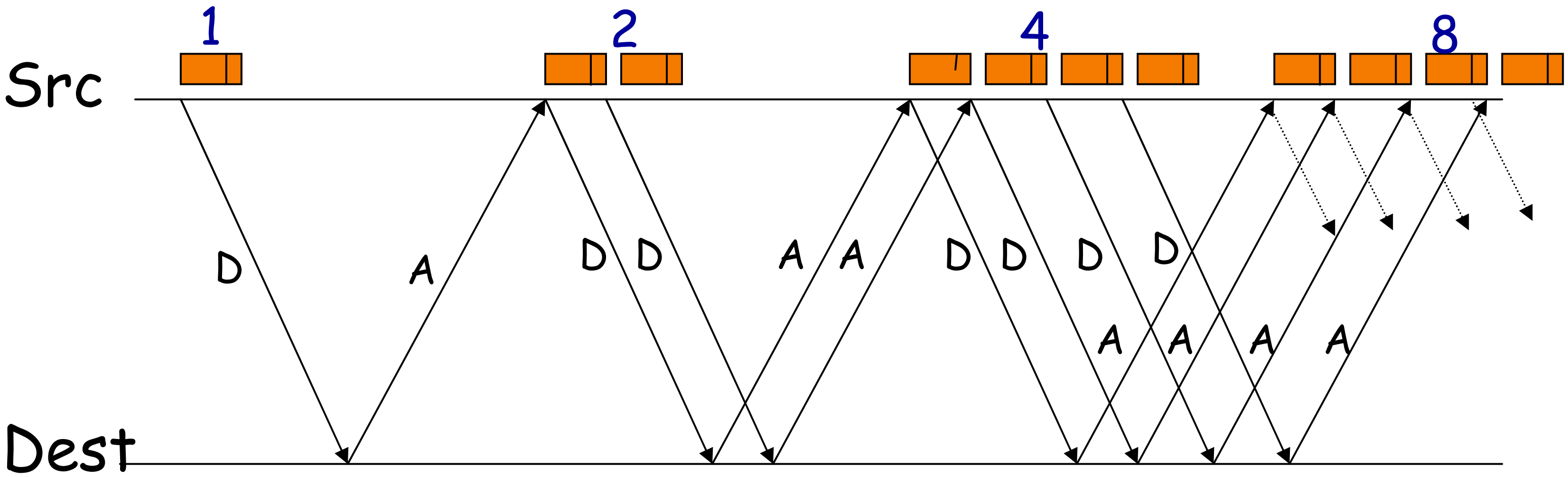
But, could take a long time to get started!

“Slow Start” Phase

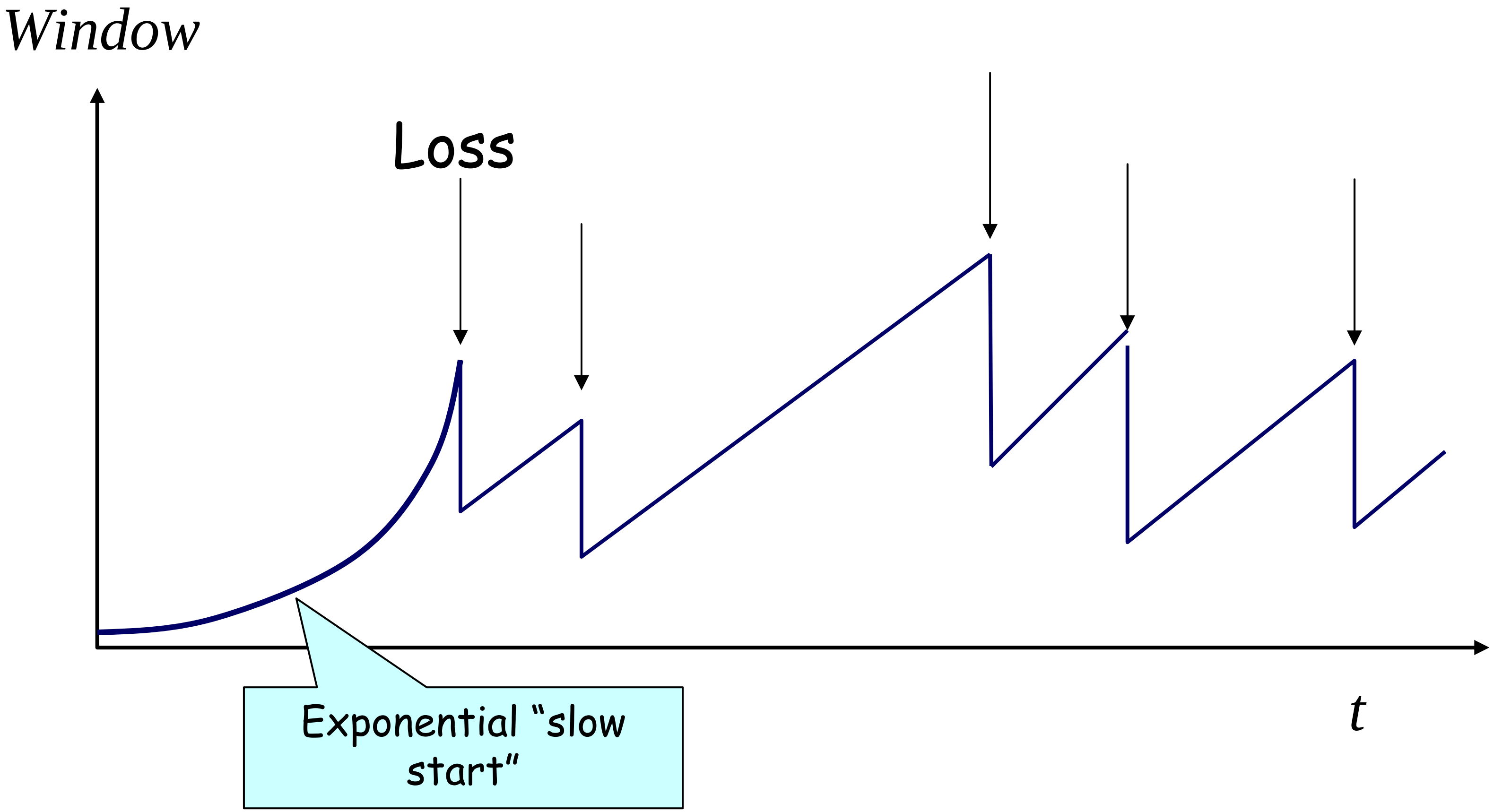
- Start with a small congestion window
  - Initially, CWND is 1 MSS
  - So, initial sending rate is MSS/RTT
- That could be pretty wasteful
  - Might be much less than the actual bandwidth
  - Linear increase takes a long time to accelerate
- Slow-start phase (really “fast start”)
  - Sender starts at a slow rate (hence the name)
  - … but increases the rate exponentially
  - … until the first loss event

Slow Start in Action

Double CWND per round-trip time



Slow Start and the TCP Sawtooth



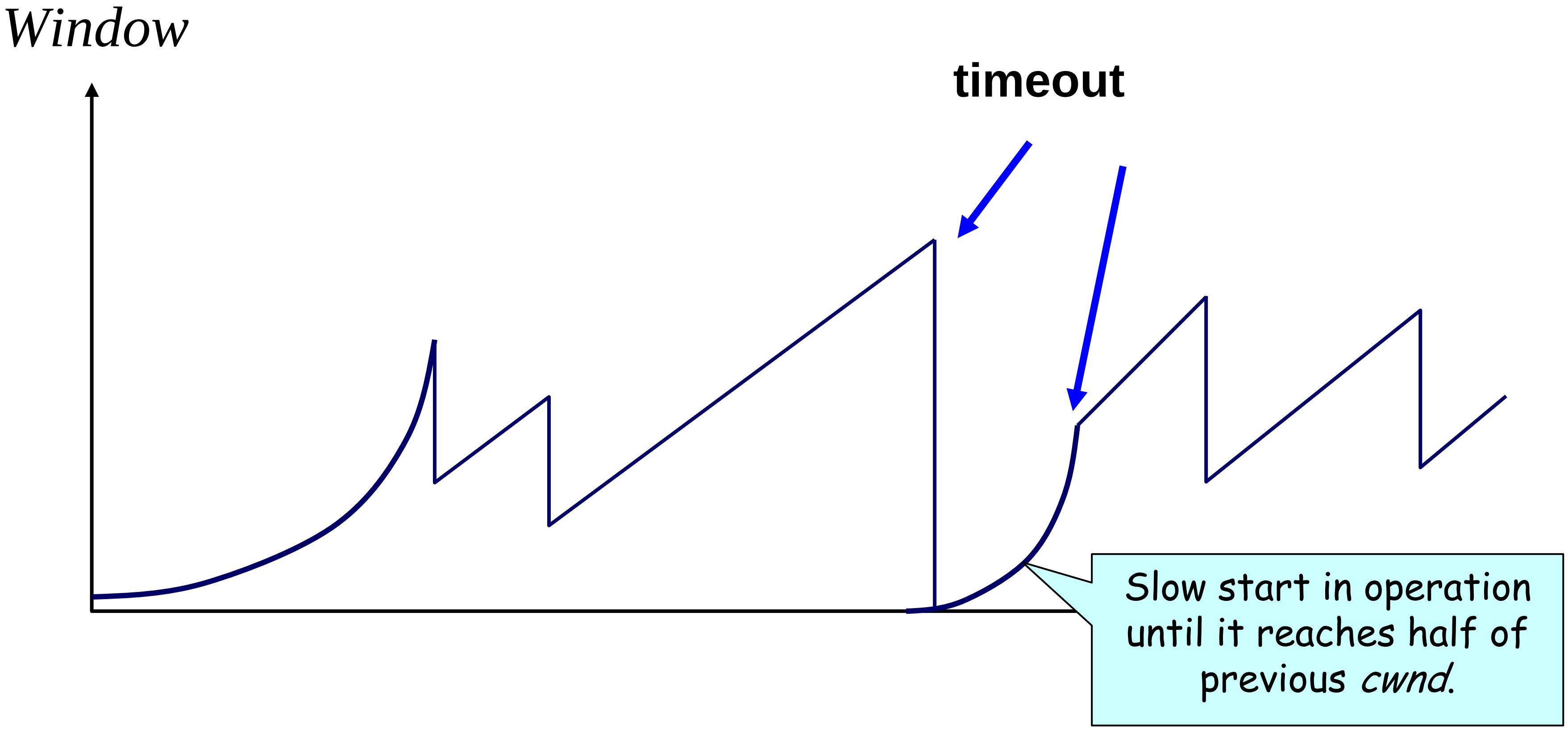
Exponential "slow start"

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

Two Kinds of Loss in TCP

- Triple duplicate ACK
  - Packet n is lost, but packets n+1, n+2, etc. arrive
  - Receiver sends duplicate acknowledgments
  - … and the sender retransmits packet n quickly
  - Do a multiplicative decrease and keep going
- Timeout
  - Packet n is lost and detected via a timeout
  - E.g., because all packets in flight were lost
  - After the timeout, blasting away for the entire CWND
  - … would trigger a very large burst in traffic
  - So, better to start over with a low CWND

Repeating Slow Start After Timeout



Slow start in operation until it reaches half of previous *cwnd*.

Slow-start restart: Go back to CWND of 1, but take advantage of knowing the previous value of CWND.

Repeating Slow Start After Idle Period

- Suppose a TCP connection goes idle for a while
  - E.g., Telnet session where you don’t type for an hour
- Eventually, the network conditions change
  - Maybe many more flows are traversing the link
  - E.g., maybe everybody has come back from lunch!
- Dangerous to start transmitting at the old rate
  - Previously-idle TCP sender might blast the network
  - … causing excessive congestion and packet loss
- So, some TCP implementations repeat slow start
  - Slow-start restart after an idle period

TCP sender congestion control

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to “Congestion Avoidance”	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT

SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2,
		CongWin = Threshold,
		Set state to “Congestion Avoidance”
			Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.

SS or CA	Timeout	Threshold = CongWin/2,
		CongWin = 1 MSS,
		Set state to “Slow Start”
			Enter slow start

SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

Other TCP Mechanisms

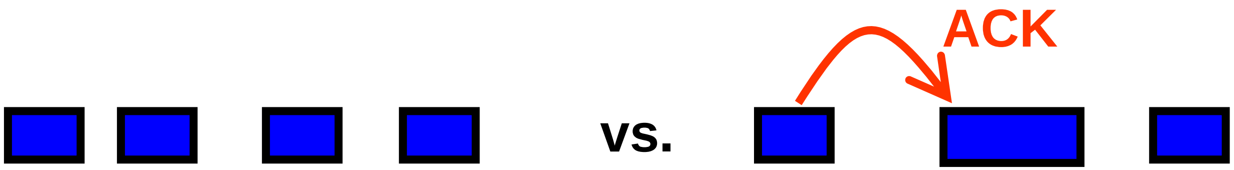
Nagle’s Algorithm and Delayed ACK

Motivation for Nagle's Algorithm

- Interactive applications
  - Telnet and rlogin
  - Generate many small packets (e.g., keystrokes)
- Small packets are wasteful
  - Mostly header (e.g., 40 bytes of header, 1 of data)
- Appealing to reduce the number of packets
  - Could force every packet to have some minimum size
  - … but, what if the person doesn't type more characters?
- Need to balance competing trade-offs
  - Send larger packets
  - … but don't introduce much delay by waiting

Nagle’s Algorithm

- Wait if the amount of data is small
  - Smaller than Maximum Segment Size (MSS)
- And some other packet is already in flight
  - I.e., still awaiting the ACKs for previous packets
- That is, send at most one small packet per RTT
  - … by waiting until all outstanding ACKs have arrived



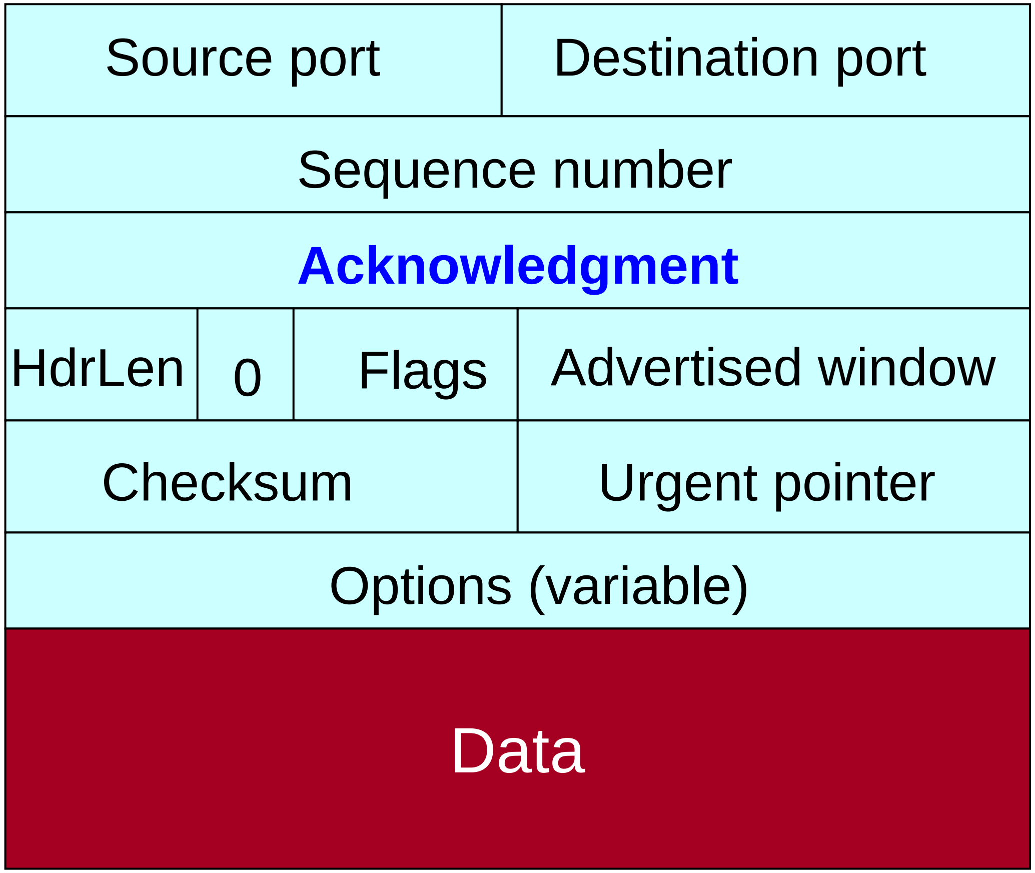
- Influence on performance
  - Interactive applications: enables batching of bytes
  - Bulk transfer: transmits in MSS-sized packets anyway

Motivation for Delayed ACK

- TCP traffic is often bidirectional
  - Data traveling in both directions
  - ACKs traveling in both directions
- ACK packets have high overhead
  - 40 bytes for the IP header and TCP header
  - … and zero data traffic
- Piggybacking is appealing
  - Host B can send an ACK to host A
  - … as part of a data packet from B to A

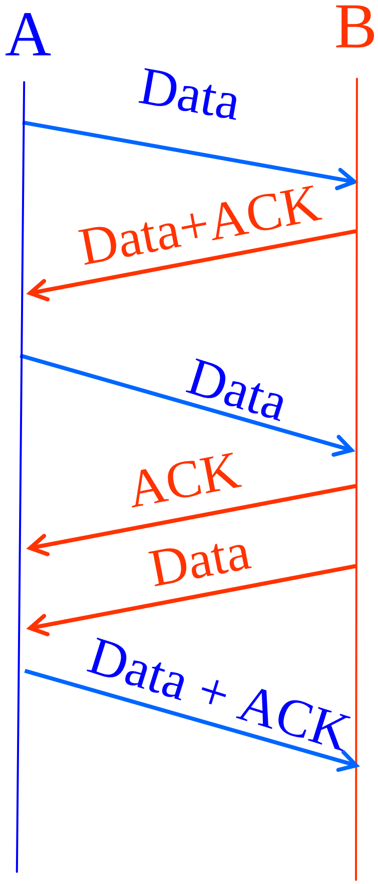
TCP Header Allows Piggybacking

Flags: SYN
FIN
RST
PSH
URG
ACK



Source port
Destination port
Sequence number
Acknowledgment
HdrLen
0
Flags
Advertised window
Checksum
Urgent pointer
Options (variable)
Data

Example of Piggybacking



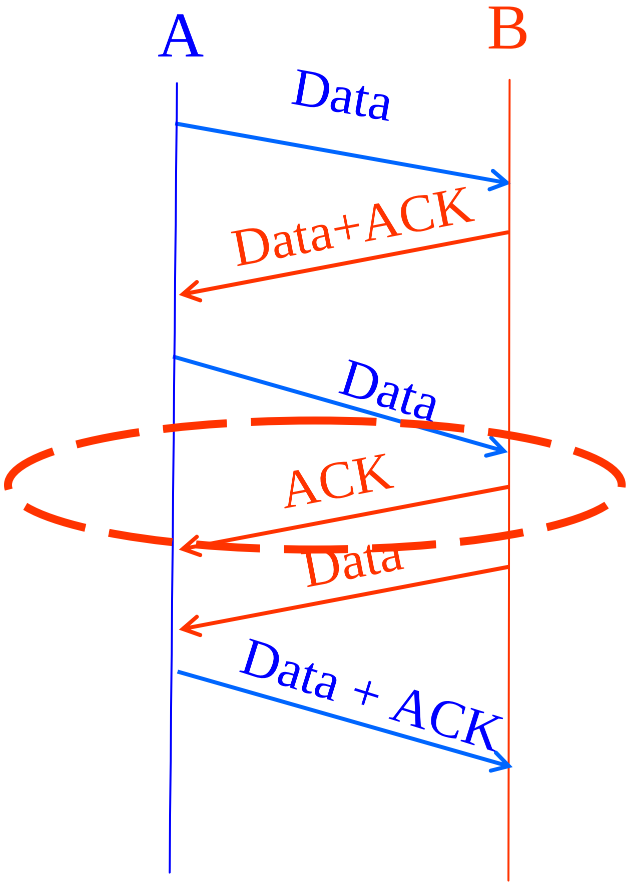
B has data to send

B doesn't have data to send

A has data to send

Increasing Likelihood of Piggybacking

- Increase piggybacking
  - TCP allows the receiver to *wait* to send the ACK
  - … in the hope that the host will have data to send
- Example: rlogin or telnet
  - Host A types characters at a UNIX prompt
  - Host B receives the character and executes a command
  - … and then data are generated
  - Would be nice if B could send the ACK with the new data



Delayed ACK

- Delay sending an ACK
  - Upon receiving a packet, the host B sets a timer
    - Typically, 200 msec or 500 msec
  - If B’s application generates data, go ahead and send
    - And piggyback the ACK bit
  - If the timer expires, send a (non-piggybacked) ACK
- Limiting the wait
  - Timer of 200 msec or 500 msec
  - ACK every other full-sized packet

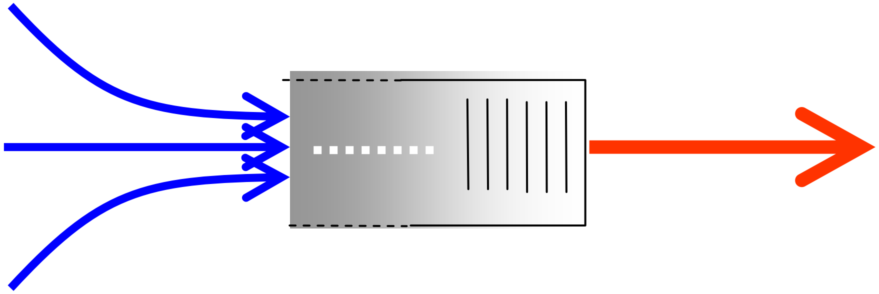
Queuing Mechanisms

Random Early Detection (RED)

Explicit Congestion Notification (ECN)

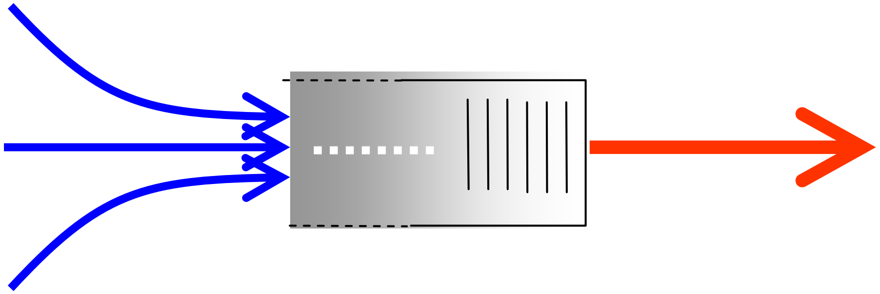
Bursty Loss From Drop-Tail Queuing

- TCP depends on packet loss
  - Packet loss is the indication of congestion
  - In fact, TCP *drives* the network into packet loss
  - … by continuing to increase the sending rate
- Drop-tail queuing leads to *bursty* loss
  - When a link becomes congested…
  - … many arriving packets encounter a full queue
  - And, as a result, many flows divide sending rate in half
  - … and, many individual flows lose multiple packets



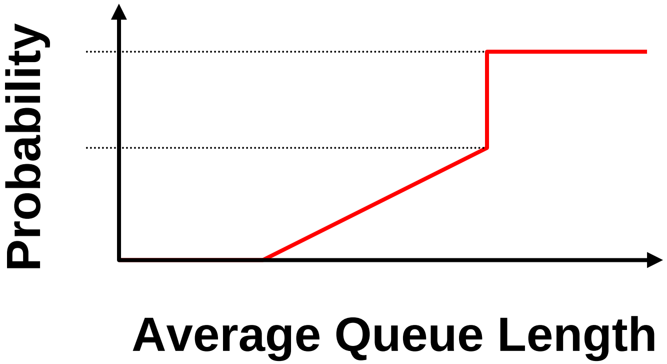
Slow Feedback from Drop Tail

- Feedback comes when buffer is completely full
  - … even though the buffer has been filling for a while
- Plus, the filling buffer is increasing RTT
  - … and the variance in the RTT
- Might be better to give early feedback
  - Get one or two flows to slow down, not all of them
  - Get these flows to slow down before it is too late



Random Early Detection (RED)

- Basic idea of RED
  - Router notices that the queue is getting backlogged
  - … and randomly drops packets to signal congestion
- Packet drop probability
  - Drop probability increases as queue length increases
  - If buffer is below some level, don't drop anything
  - … otherwise, set drop probability as function of queue



Properties of RED

- Drops packets before queue is full
  - In the hope of reducing the rates of some flows
- Drops packet in proportion to each flow's rate
  - High-rate flows have more packets
  - … and, hence, a higher chance of being selected
- Drops are spaced out in time
  - Which should help desynchronize the TCP senders
- Tolerant of burstiness in the traffic
  - By basing the decisions on *average* queue length

Problems With RED

- Hard to get the tunable parameters just right
  - How early to start dropping packets?
  - What slope for the increase in drop probability?
  - What time scale for averaging the queue length?
- Sometimes RED helps but sometimes not
  - If the parameters aren't set right, RED doesn't help
  - And it is hard to know how to set the parameters
- RED is implemented in practice
  - But, often not used due to the challenges of tuning right
- Many variations
  - With cute names like "Blue" and "FRED"… ☺

Explicit Congestion Notification

- Early dropping of packets
  - Good: gives early feedback
  - Bad: has to drop the packet to give the feedback
- Explicit Congestion Notification
  - Router marks the packet with an ECN bit
  - … and sending host interprets as a sign of congestion
- Surmounting the challenges
  - Must be supported by the end hosts and the routers
  - Requires two bits in the IP header (one for the ECN mark, and one to indicate the ECN capability)
  - Solution: borrow two of the Type-Of-Service bits in the IPv4 packet header

Conclusions

- Congestion is inevitable
  - Internet does not reserve resources in advance
  - TCP actively tries to push the envelope
- Congestion can be handled
  - Additive increase, multiplicative decrease
  - Slow start, and slow-start restart
- Active Queue Management can help
  - Random Early Detection (RED)
  - Explicit Congestion Notification (ECN)

Virtual Circuit Switching and QoS

Reading: 3.1.2, 3.3, 4.5, and 6.5

Introduction to Data Networks

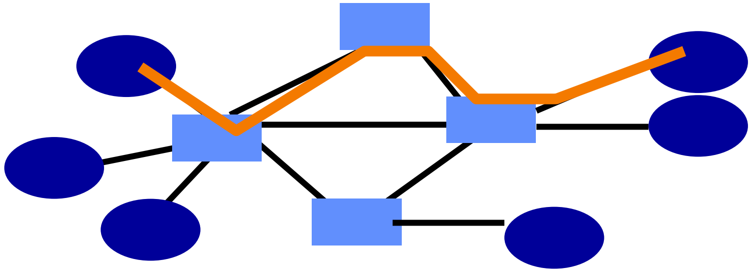
2008.5

Goals of Today’s Lecture

- Circuit switching
  - Establish, transfer, and teardown
  - Comparison with packet switching
  - Virtual circuits as a hybrid scheme
- Quality of service in virtual-circuit networks
  - Traffic specification and enforcement
  - Admission control and resource reservation
  - Link scheduling (FIFO, priority, and weighted fairness)
  - Path selection (quality-of-service routing)
- Quality of service for IP traffic
  - IP over virtual circuits
  - Differentiated services

Circuit Switching (e.g., Phone Network)

- Establish: source creates circuit to destination
  - Node along the path store connection info
  - Nodes may reserve resources for the connection
- Transfer: source sends data over the circuit
  - No destination address, since nodes know path
- Teardown: source tears down circuit when done



Advantages of Circuit Switching

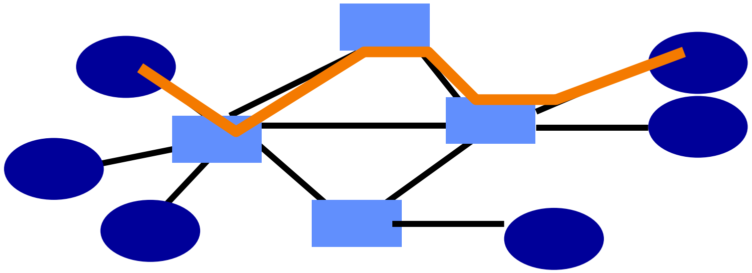
- Guaranteed bandwidth
  - Predictable communication performance
  - Not “best-effort” delivery with no real guarantees
- Simple abstraction
  - Reliable communication channel between hosts
  - No worries about lost or out-of-order packets
- Simple forwarding
  - Forwarding based on time slot or frequency
  - No need to inspect a packet header
- Low per-packet overhead
  - Forwarding based on time slot or frequency
  - No IP (and TCP/UDP) header on each packet

Disadvantages of Circuit Switching

- Wasted bandwidth
  - Bursty traffic leads to idle connection during silent period
  - Unable to achieve gains from statistical multiplexing
- Blocked connections
  - Connection refused when resources are not sufficient
  - Unable to offer “okay” service to everybody
- Connection set-up delay
  - No communication until the connection is set up
  - Unable to avoid extra latency for small data transfers
- Network state
  - Network nodes must store per-connection information
  - Unable to avoid per-connection storage and state

Virtual Circuit (VC)

- Hybrid of packets and circuits
  - Circuits: establish and teardown along end-to-end path
  - Packets: divide the data into packets with identifiers
- Packets carry a virtual-circuit identifier
  - Associates each packet with the virtual circuit
  - Determines the next link along the path
- Intermediate nodes maintain state VC
  - Forwarding table entry
  - Allocated resources

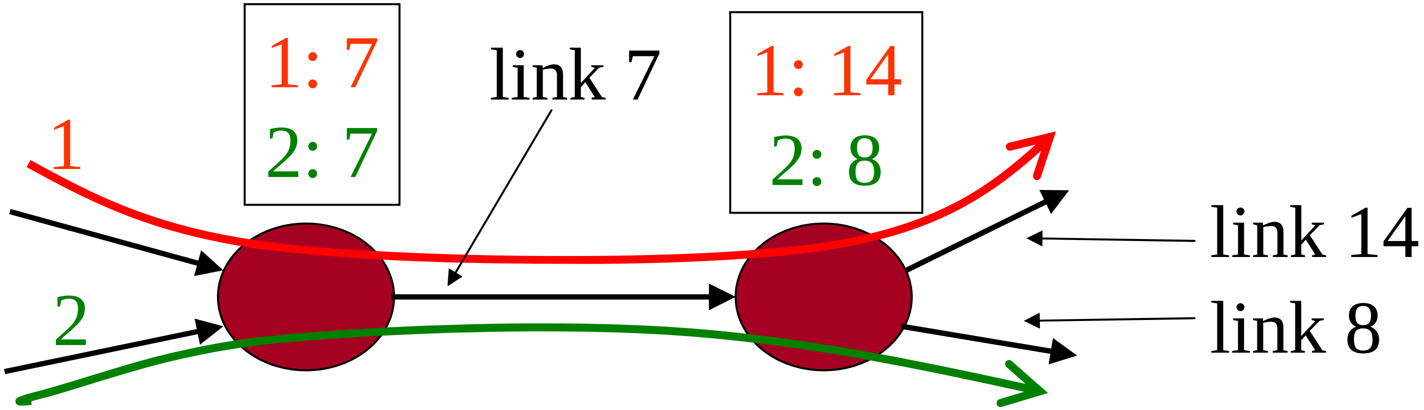


Establishing the Circuit

- Signaling
  - Creating the entries in the forwarding tables
  - Reserving resources for the virtual circuit, if needed
- Two main approaches to signaling
  - Network administrator configures each node
  - Source sends set-up message along the path
- Set-up latency
  - Time for the set-up message to traverse the path
  - … and return back to the source
- Routing
  - End-to-end path is selected during circuit set-up

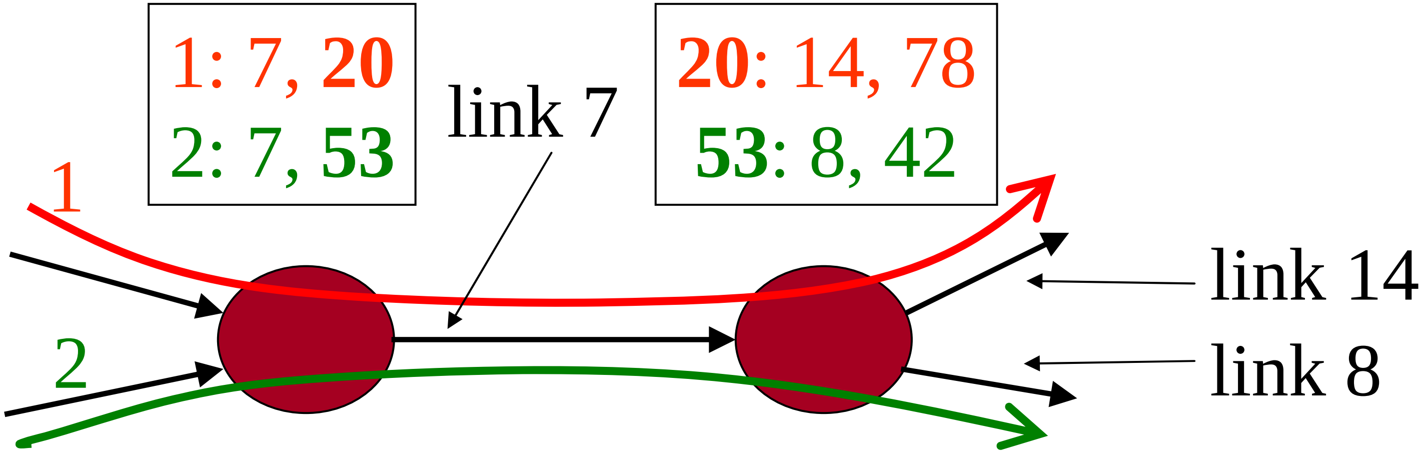
Virtual Circuit Identifier (VC ID)

- Virtual Circuit Identifier (VC ID)
  - Source set-up: establish path for the VC
  - Switch: mapping VC ID to an outgoing link
  - Packet: fixed length label in the header



Swapping the Label at Each Hop

- Problem: using VC ID along the whole path
  - Each virtual circuit consumes a unique ID
  - Starts to use up all of the ID space in the network
- Label swapping
  - Map the VC ID to a new value at each hop
  - Table has old ID, and next link and new ID



1: 7, **20**
2: 7, **53**

20: 14, 78
53: 8, 42

Virtual Circuits Similar to IP Datagrams

- Data divided in to packets
  - Sender divides the data into packets
  - Packet has address (e.g., IP address or VC ID)
- Store-and-forward transmission
  - Multiple packets may arrive at once
  - Need buffer space for temporary storage
- Multiplexing on a link
  - No reservations: statistical multiplexing
    - Packets are interleaved without a fixed pattern
  - Reservations: resources for group of packets
    - Guarantees to get a certain number of “slots”

Virtual Circuits Differ from IP Datagrams

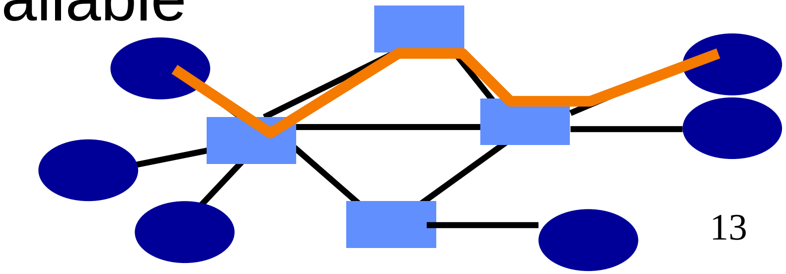
- Forwarding look-up
  - Virtual circuits: fixed-length connection id
  - IP datagrams: destination IP address
- Initiating data transmission
  - Virtual circuits: must signal along the path
  - IP datagrams: just start sending packets
- Router state
  - Virtual circuits: routers know about connections
  - IP datagrams: no state, easier failure recovery
- Quality of service
  - Virtual circuits: resources and scheduling per VC
  - IP datagrams: difficult to provide QoS

Quality of Service

- Allocating resources to the virtual circuit
  - E.g., guaranteed bandwidth on each link in the path
  - E.g., guaranteeing a maximum delay along the path
- Admission control
  - Check during signaling that the resources are available
  - Saying “no” if they are not, and reserving them if they are
- Resource scheduling
  - Apply scheduling algorithms during the data transfer
  - To ensure that the performance guarantees are met

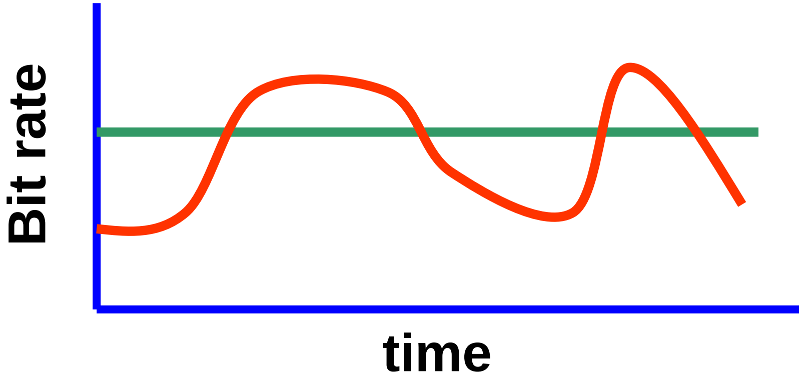
Admission Control

- Source sends a reservation message
  - E.g., “this virtual circuit needs 5 Mbps”
- Each switch along the path
  - Keeps track of the reserved resources
    - E.g., “the link has 6 Mbps left”
  - Checks if enough resources remain
    - E.g., “6 Mbps > 5 Mbps, so circuit can be accepted”
  - Creates state for circuit and reserves resources
    - E.g., “now only 1 Mbps is available”



Admission Control: Flowspec

- Flowspec: information about the traffic
  - The traffic characteristics of the flow
  - The service requested from the network
- Specifying the traffic characteristics
  - Simplest case: constant bit rate (some # of bits per sec)
  - Yet, many applications have variable bit rates
  - … and will send more than their average bit rate

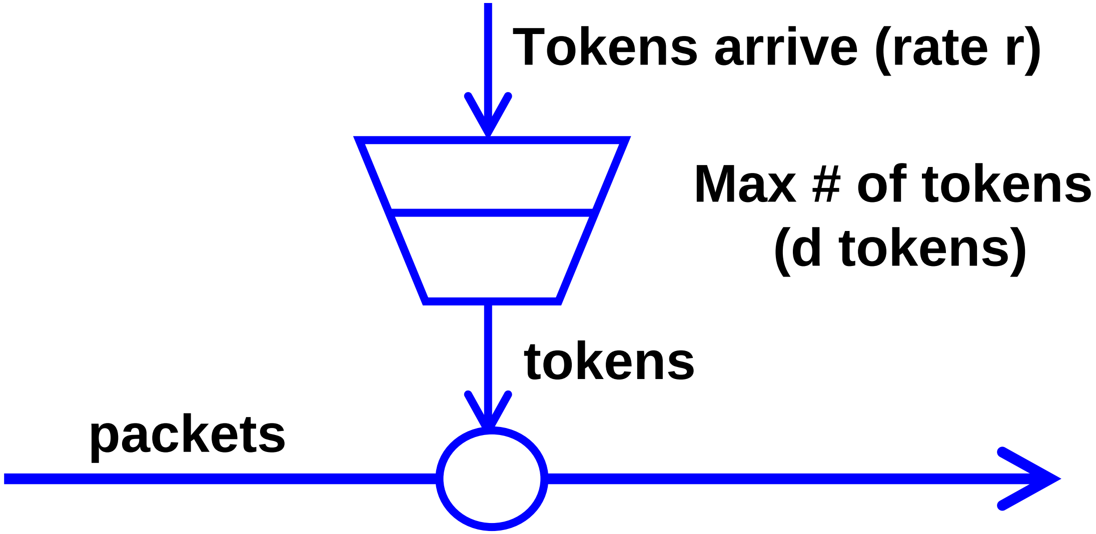


Specifying Bursty Traffic

- Option #1: Specify the maximum bit rate
  - Maximum bit rate may be much higher average
  - Reserving for the worst case is wasteful
- Option #2: Specify the average bit rate
  - Average bit rate is not sufficient
  - Network will not be able to carry all of the packets
  - Reserving for average case leads to bad performance
- Option #3: Specify the burstiness of the traffic
  - Specify both the average rate and the burst size
  - Allows the sender to transmit bursty traffic
  - … and the network to reserve the necessary resources

Leaky Bucket Traffic Model

- Traffic characterization with two parameters
  - Token rate r
  - Bucket depth d
- Sending data requires a token
  - Can send at rate r all the time
  - Can send at a higher rate for a short time



Service Requested From the Network

- Variety of service models
  - Bandwidth guarantee (e.g., 5 Mbps)
  - Delay guarantee (e.g., no more than 100 msec)
  - Loss rate (e.g., no more than 1% packet loss)
- Signaling during admission control
  - Translate end-to-end requirement into per-hop
  - Easy for bandwidth (e.g., 5 Mbps on each hop)
  - Harder for delay and loss
  - … since each hop contributes to the delay and loss
- Per-hop admission control
  - Router takes the service requirement and traffic spec
  - … and determines whether it can accept the circuit

Ensuring the Source Behaves

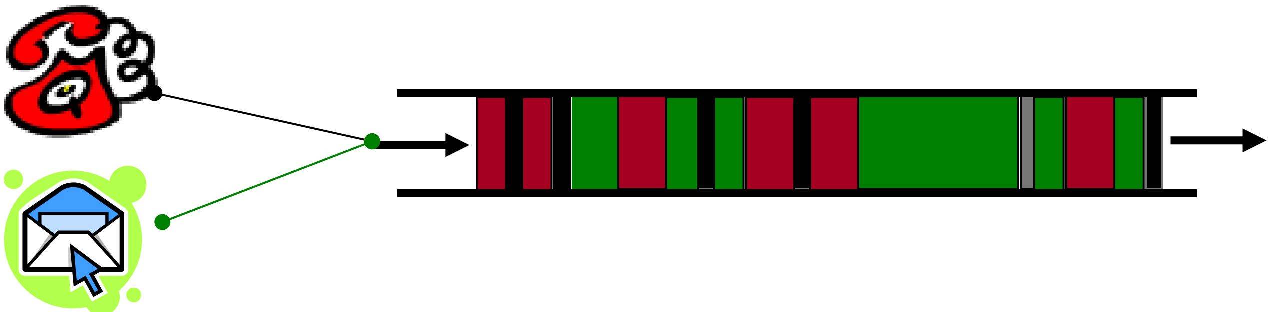
- Guarantees depend on the source behaving
  - Extra traffic might overload one or more links
  - Leading to congestion, and resulting delay and loss
  - Solution: need to enforce the traffic specification
- Solution #1: policing
  - Drop all data in excess of the traffic specification
- Solution #2: shaping
  - Delay the data until it obeys the traffic specification
- Solution #3: marking
  - Mark all data in excess of the traffic specification
  - … and give these packets lower priority in the network

Enforcing Behavior

- Applying a leaky bucket to the traffic
  - Simulating a leaky bucket (r, d) at the edge
  - Discarding, delaying, or marking packets accordingly
- Ensures that the incoming traffic obeys the profile
  - So that the network can provide the guarantees
- Technical challenge
  - Applying leaky buckets for many flows at a high rate

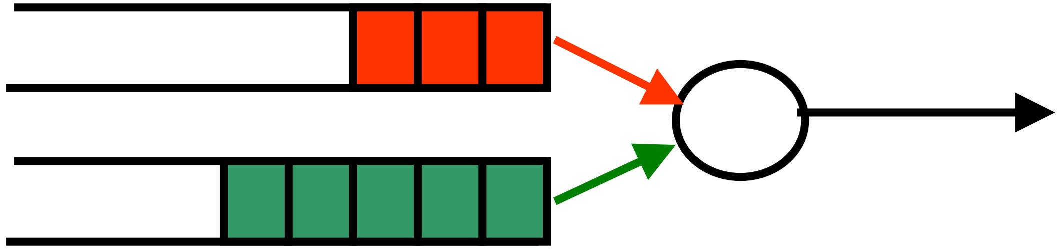
Link Scheduling: FIFO

- First-in first-out scheduling
  - Simple to implement
  - But, restrictive in providing guarantees
- Example: two kinds of traffic
  - Video conferencing needs high bandwidth and low delay
    - E.g., 1 Mbps and 100 msec delay
  - E-mail transfers are not that sensitive about delay
- Cannot admit much e-mail traffic
  - Since it will interfere with the video conference traffic



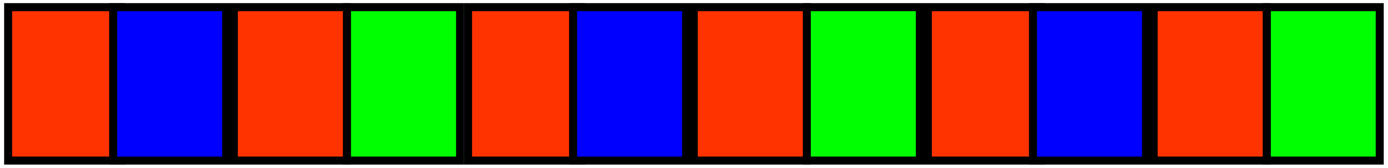
Link Scheduling: Strict Priority

- Strict priority
  - – Multiple levels of priority
  - – Always transmit high-priority traffic, when present
  - – .. and force the lower priority traffic to wait
- Isolation for the high-priority traffic
  - – Almost like it has a dedicated link
  - – Except for the (small) delay for packet transmission
    - High-priority packet arrives during transmission of low-priority
    - Router completes sending the low-priority traffic first



Link Scheduling: Weighted Fairness

- Limitations of strict priority
  - Lower priority queues may starve for long periods
  - … even if the high-priority traffic can afford to wait
- Weighted fair scheduling
  - Assign each queue a fraction of the link bandwidth
  - Rotate across the queues on a small time scale
  - Send extra traffic from one queue if others are idle



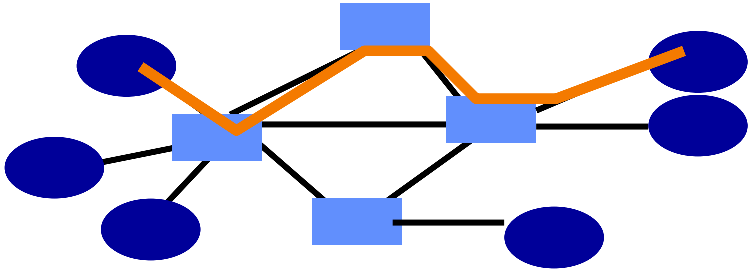
50% red, 25% blue, 25% green

Link Schedulers: Trade-Offs

- Implementation complexity
  - FIFO is easy
    - One queue, trivial scheduler
  - Strict priority is a little harder
    - One queue per priority level, simple scheduler
  - Weighted fair scheduling
    - One queue per virtual circuit, and more complex scheduler
- Admission control
  - Using more sophisticated schedulers can allow the router to admit more virtual circuits into the network
  - Getting close to making full use of the network resources
  - E.g., FIFO requires very conservative admission control

Routing in Virtual Circuit Networks

- Routing decisions take place at circuit set-up
  - Resource reservations made along end-to-end path
  - Data packets flow along the already-chosen path
- Simplest case: routing based only on the topology
  - Routing based on the topology and static link weights
  - Source picks the end-to-end path, and signals along it
  - If the path lacks sufficient resources, that’s too bad!

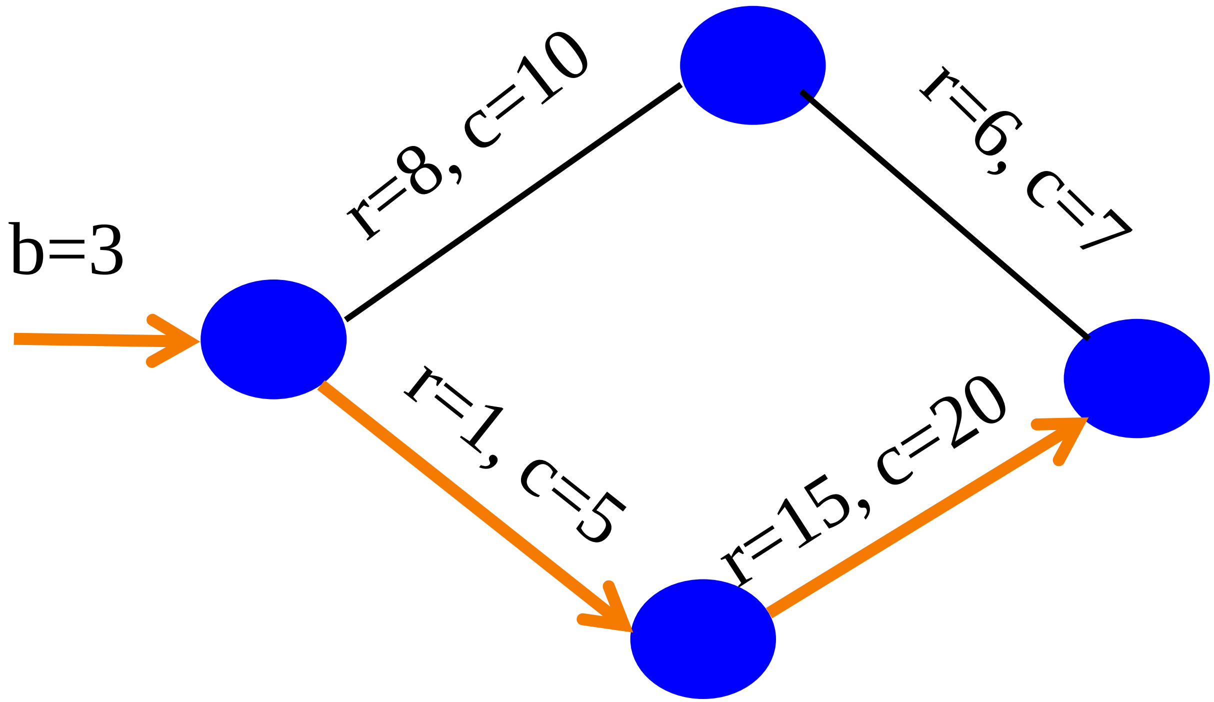


Quality-of-Service Routing

- QoS routing: source selects the path intelligently
  - Tries to find a path that can satisfy the requirements
- Traffic performance requirement
  - Guaranteed bandwidth b per connection
- Link resource reservation
  - Reserved bandwidth r_i on link l
  - Capacity c_i on link i
- Signaling: admission control on path P
  - Reserve bandwidth b on each link i on path P
  - Block: if ($r_i+b>c_i$) then reject (or try again)
  - Accept: else $r_i = r_i + b$

Source-Directed QoS Routing

- New connection with *b =3*
  - Routing: select path with available resources
  - Signaling: reserve bandwidth along the path (*r* = *r* +3)
  - Forward data packets along the selected path
  - Teardown: free the link bandwidth (*r* =*r* -3)



QoS Routing: Link-State Advertisements

- Advertise available resources per link
  - E.g., advertise available bandwidth ($c_i - r_i$) on link *i*
  - Every *T* seconds, independent of changes
  - … or, when the metric changes beyond threshold
- Each router constructs view of topology
  - Topology including the latest link metrics
- Each router computes the paths
  - Looks at the requirements of the connection
  - … as well as the available resources in the network
  - And selects a path that satisfies the needs
- Then, the router signals to set up the path
  - With a high likelihood that the request is accepted

QoS Routing: Example Path Selection

- Shortest widest path
  - Find the path with highest available bandwidth
    - To increase the likelihood that set-up is successful
  - That is, consider paths with largest $min_i(c_i-r_i)$
    - Tie-break on smallest number of hops
- Widest shortest path
  - Consider only paths with minimum hops
    - To minimize the total amount of resources required
  - Tie-break on largest value of $min_i(c_i-r_i)$
    - To increase the likelihood that set-up is successful

Asynchronous Transfer Mode (ATM)

- ATM history
  - Important technology in the 1980s and early 1990s
  - Embraced by the telecommunications industry
- ATM goals
  - A single unified network standard
  - Supporting synchronous and packet-based networking
  - With multiple levels of quality of service
- ATM technology
  - Virtual circuits
  - Small, fixed-sized packets (called cells)
    - Fixed size simplifies the switch design
    - Small makes it easier to support delay-sensitive traffic

Picking the ATM Cell Size

- Cell size too small
  - Header overhead relative to total packet size
  - Processing overhead on devices
- Cell size too large
  - Wasted padding when the data is smaller
  - Delay to wait for transmission of previous packet

- ATM cell: 53 bytes (designed by committee!)
  - The U.S. wanted 64 bytes, and Europe wanted 32
  - Smaller packets avoid the need for echo cancellation
  - Compromise: 5-byte header and 48 bytes of data

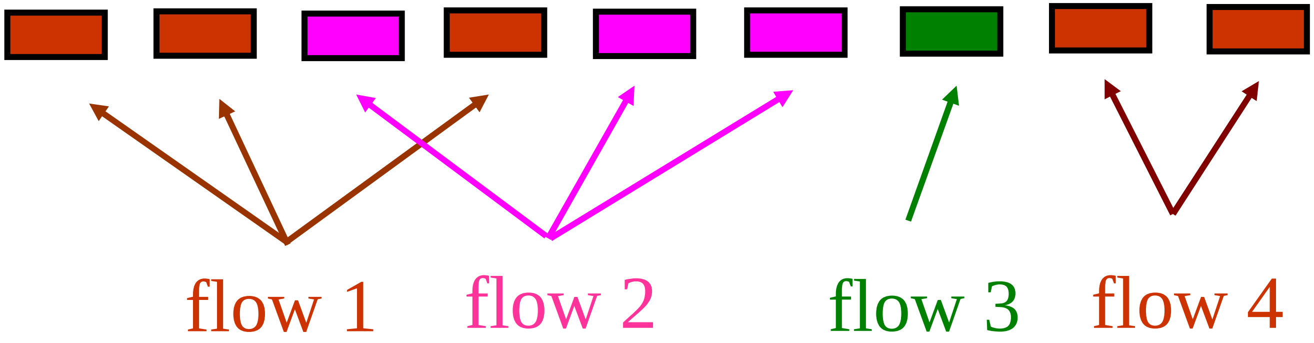
Interfacing to End Hosts

- ATM works best as an end-to-end technology
  - End host requests a virtual circuit to another host
  - … with a traffic specification and QoS requirements
  - And the network establishes an end-to-end circuit
- But, requires some support in the end host
  - To initiate the circuit establishment process
  - And for applications to specify the traffic and the QoS
- What to do if the end hosts don’t support ATM?
  - Carry packets from the end host to a network device
  - And, then have the network device create the circuit

Inferring the Need for a Virtual Circuit

- Which IP packets go on a virtual circuit?
  - All packets in the same TCP or UDP transfer?
  - All packets between same pair of end hosts?
  - All packets between same pair of IP subnets?
- Edge router can infer the need for a circuit
  - Match on packet header bits
    - E.g., source, destination, port numbers, etc.
  - Apply policy for picking bandwidth parameters
    - E.g., Web traffic get 10 Kbps, video gets 2 Mbps
  - Trigger establishment of circuit for the traffic
    - Select path based on load and requirements
    - Signal creation of the circuit
    - Tear down circuit after an idle period

Grouping IP Packets Into Flows



- Group packets with the “same” end points
  - Application level: single TCP connection
  - Host level: single source-destination pair
  - Subnet level: single source prefix and dest prefix
- Group packets that are close together in time
  - E.g., 60-sec spacing between consecutive packets

Challenges for IP Over ATM

- Many IP flows are short
  - Most Web transfers are less than 10 packets
  - Is it worthwhile to set up a circuit?
- Subdividing an IP packet into cells
  - Wasted space if packet is not multiples of 48 bytes
- Difficult to know what resources to reserve
  - Internet applications don’t specify traffic or QoS
- Two separate addressing schemes
  - IP addresses and ATM end-points
- Complexity of two sets of protocols
  - Supporting both IP and ATM protocols

ATM Today

- Still used in some contexts
  - Some backbones and edge networks
  - But, typically the circuits are not all that dynamic
  - E.g., ATM circuit used as a link for aggregated traffic
- Some key ideas applicable to other technologies
  - Huge body of work on quality of service
  - Idea of virtual circuits (becoming common now in MultiProtocol Label Switching)

Differentiated Services in IP

- Compromise solution for QoS
  - Not as strong guarantees as per-circuit solutions
  - Not as simple as best-effort service
- Allocate resources for classes of traffic
  - Gold, silver, and bronze
- Scheduling resources based on ToS bits
  - Put packets in separate queues based on ToS bits
- Packet classifiers to set the ToS bits
  - Mark the “Type of Service” bits in the IP packet header
  - Based on classification rules at the network edge

Example Packet Classifier

- Gold traffic
  - All traffic to/from Shirley Tilgman’s IP address
  - All traffic to/from the port number for DNS
- Silver traffic
  - All traffic to/from academic and administrative buildings
- Bronze traffic
  - All traffic on the public wireless network
- Then, schedule resources accordingly
  - E.g., 50% for gold, 30% for silver, and 20% for bronze

Real Guarantees?

- It depends…
  - Must limit volume of traffic that can be classified as gold
  - E.g., by marking traffic “bronze” by default
  - E.g., by policing traffic at the edge of the network
- QoS through network management
  - Configuring packet classifiers
  - Configuring policers
  - Configuring link schedulers
- Rather than through dynamic circuit set-up

Example Uses of QoS Today

- Virtual Private Networks
  - Corporate networks interconnecting via the Internet
  - E.g., IBM sites throughout the world on AT&T backbone
  - Carrying VPN traffic in “gold” queue protects the QoS
  - Limiting the amount of gold traffic avoids overloads
- Routing-protocol traffic
  - Routing protocol messages are “in band”
  - So, routing messages may suffer from congestion
  - Carrying routing messages in the “gold” queue helps
- Challenge: end-to-end QoS across domains… ☹

Conclusions

- Virtual circuits
  - Establish a path and reserve resources in advance
  - Enable end-to-end quality-of-service guarantees
  - Importance of admission control, policing, & scheduling
- Best effort vs. QoS
  - IP won the “IP vs. ATM” competition
  - Yet, QoS is increasingly important, for multimedia, business transactions, protecting against attacks, etc.
  - And, virtual circuits are useful for controlling the flow of traffic, providing value-added services, and so on
  - So, virtual circuits and QoS exist in some form today
  - … and the debate continues about the role in the future

Multimedia Networking

Introduction to Data Networks

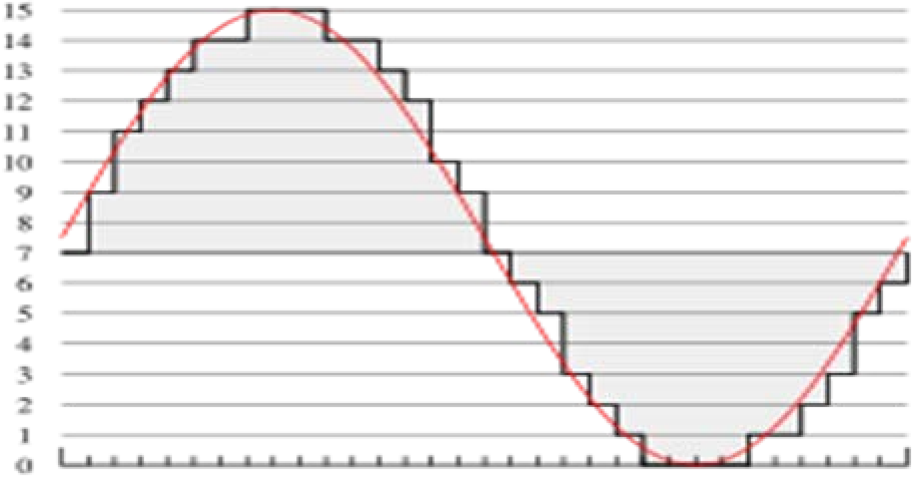
2008.5

Goals of Today's Lecture

- Digital audio and video
  - Sampling, quantizing, and compressing
- Multimedia applications
  - Streaming audio and video for playback
  - Live, interactive audio and video
- Multimedia transfers over a best-effort network
  - Tolerating packet loss, delay, and jitter
  - Forward error correction and playout buffers
- Improving the service the network offers
  - Marking, policing, scheduling, and admission control

Digital Audio

- Sampling the analog signal
  - Sample at some fixed rate
  - Each sample is an arbitrary real number
- Quantizing each sample
  - Round each sample to one of a finite number of values
  - Represent each sample in a fixed number of bits



4 bit representation (values 0-15)

Audio Examples

- Speech
  - Sampling rate: 8000 samples/second
  - Sample size: 8 bits per sample
  - Rate: 64 kbps

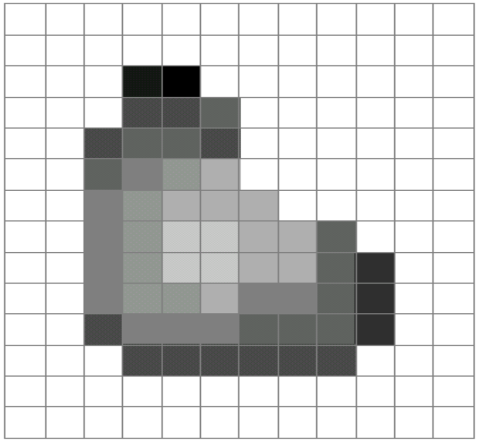
- Compact Disc (CD)
  - Sampling rate: 44,100 samples/second
  - Sample size: 16 bits per sample
  - Rate: 705.6 kbps for mono,
    1.411 Mbps for stereo

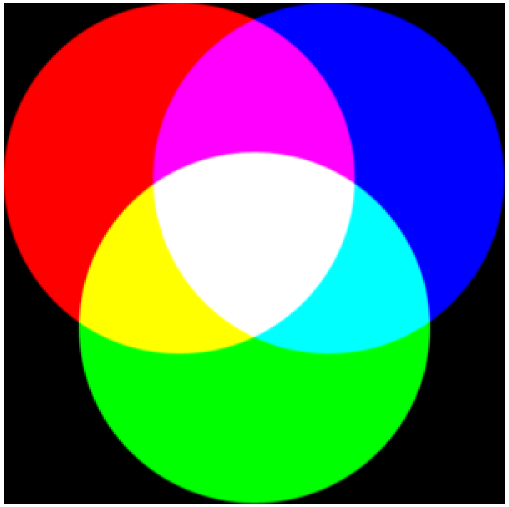
Audio Compression

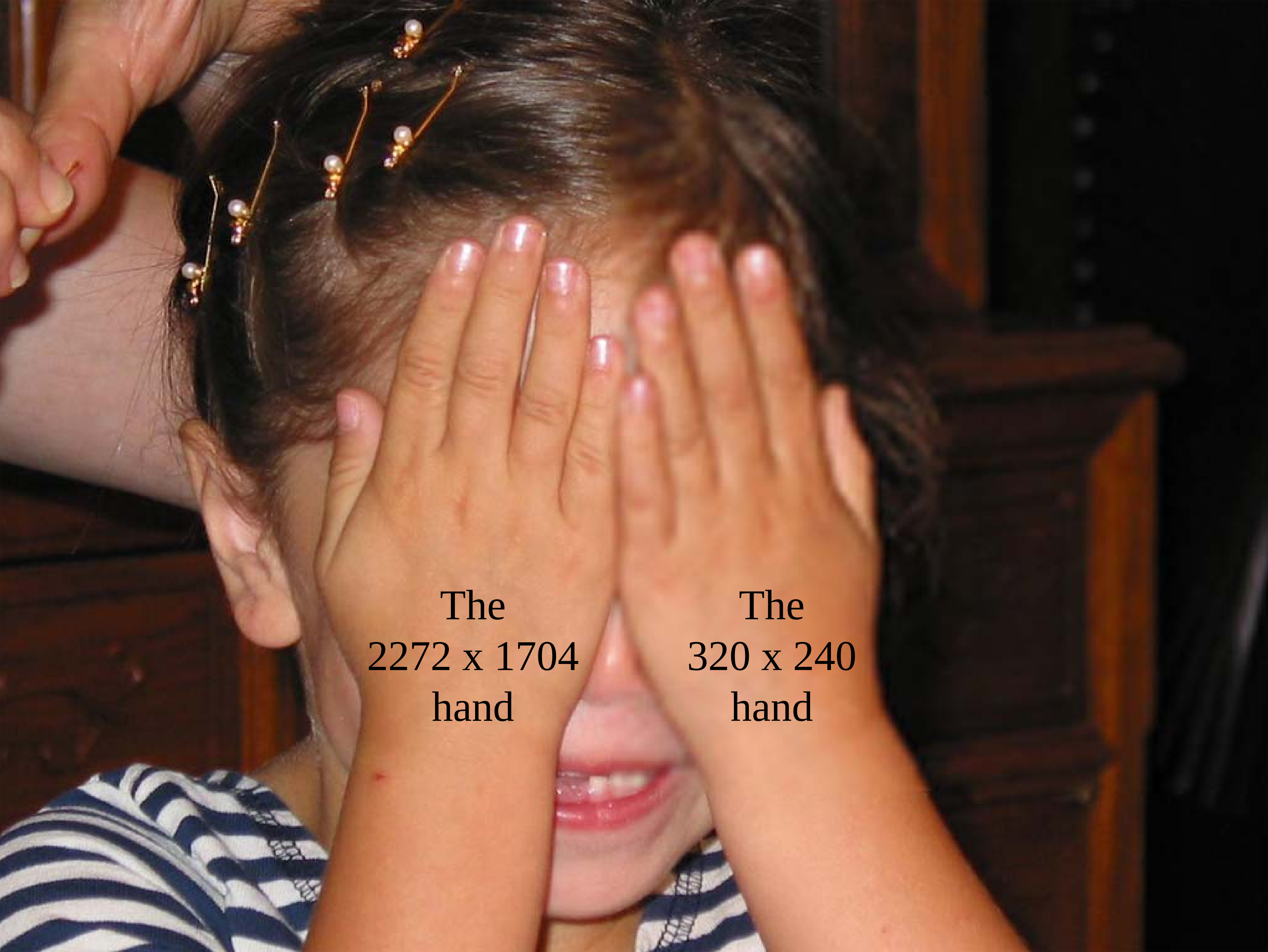
- Audio data requires too much bandwidth
  - Speech: 64 kbps is too high for a dial-up modem user
  - Stereo music: 1.411 Mbps exceeds most access rates
- Compression to reduce the size
  - Remove redundancy
  - Remove details that human tend not to perceive
- Example audio formats
  - Speech: GSM (13 kbps), G.729 (8 kbps), and G.723.3 (6.4 and 5.3 kbps)
  - Stereo music: MPEG 1 layer 3 (MP3) at 96 kbps, 128 kbps, and 160 kbps

Digital Video

- Sampling the analog signal
  - Sample at some fixed rate (e.g., 24 or 30 times per sec)
  - Each sample is an image
- Quantizing each sample
  - Representing an image as an array of picture elements
  - Each pixel is a mixture of colors (red, green, and blue)
  - E.g., 24 bits, with 8 bits per color

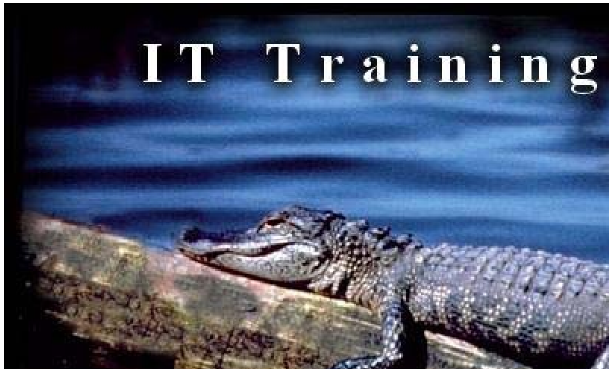




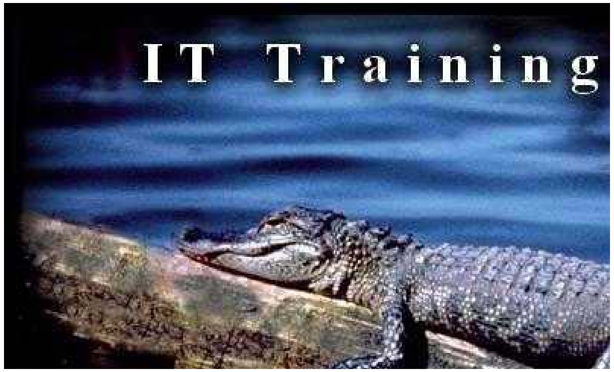

The
2272 x 1704
hand
The
320 x 240
hand

Video Compression: Within an Image

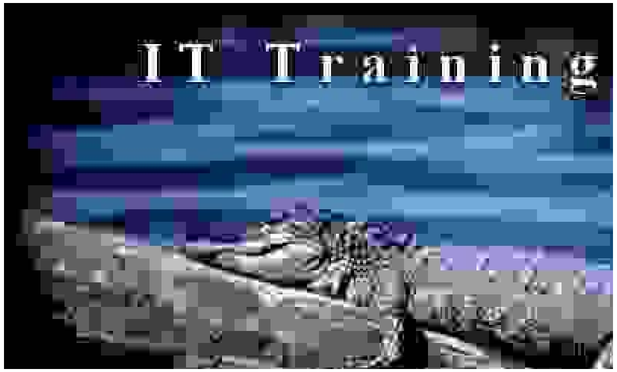
- Image compression
  - Exploit spatial redundancy (e.g., regions of same color)
  - Exploit aspects humans tend not to notice
- Common image compression formats
  - Joint Pictures Expert Group (JPEG)
  - Graphical Interchange Format (GIF)



Uncompressed: 167 KB



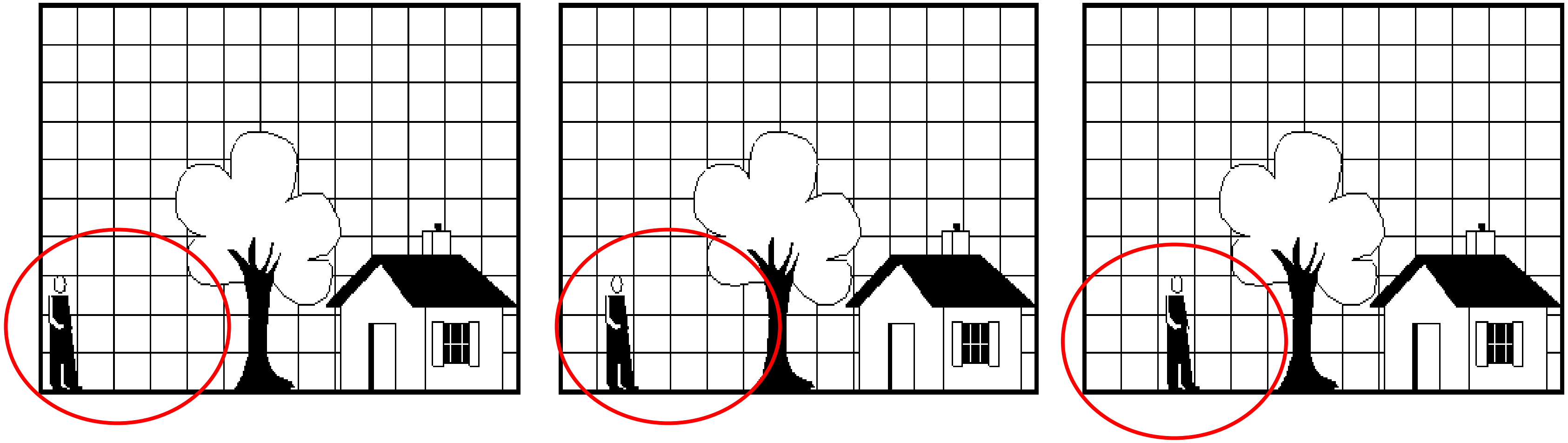
Good quality: 46 KB



Poor quality: 9 KB

Video Compression: Across Images

- Compression across images
  - Exploit temporal redundancy across images
- Common video compression formats
  - MPEG 1: CD-ROM quality video (1.5 Mbps)
  - MPEG 2: high-quality DVD video (3-6 Mbps)
  - Proprietary protocols like QuickTime and RealNetworks

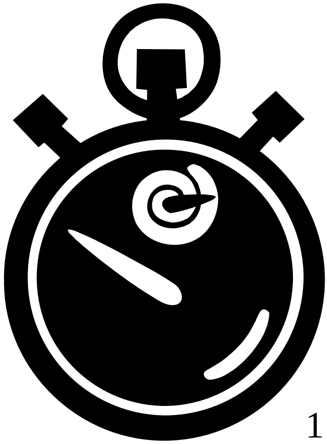


Transferring Audio and Video Data

- Simplest case: just like any other file
  - Audio and video data stored in a file
  - File downloaded using conventional protocol
  - Playback does not overlap with data transfer
- A variety of more interesting scenarios
  - Live vs. pre-recorded content
  - Interactive vs. non-interactive
  - Single receiver vs. multiple receivers
- Examples
  - Streaming audio and video data from a server
  - Interactive audio in a phone call

Streaming Stored Audio and Video

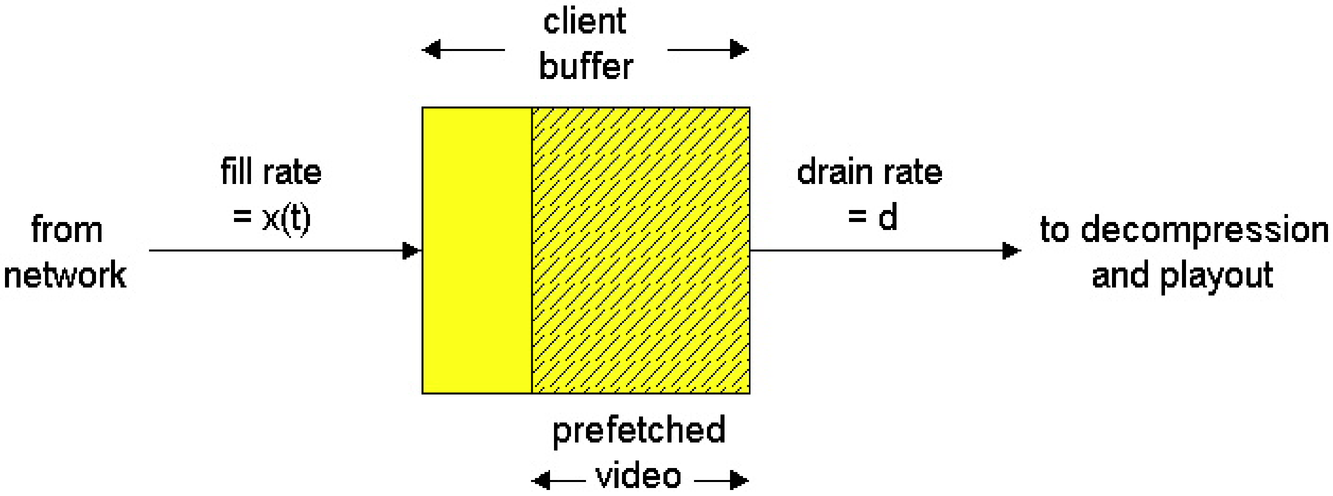
- Client-server system
  - Server stores the audio and video files
  - Clients request files, play them as they download, and perform VCR-like functions (e.g., rewind and pause)
- Playing data at the right time
  - Server divides the data into segments
  - … and labels each segment with timestamp or frame id
  - … so the client knows when to play the data
- Avoiding starvation at the client
  - The data must arrive quickly enough
  - … otherwise the client cannot keep playing



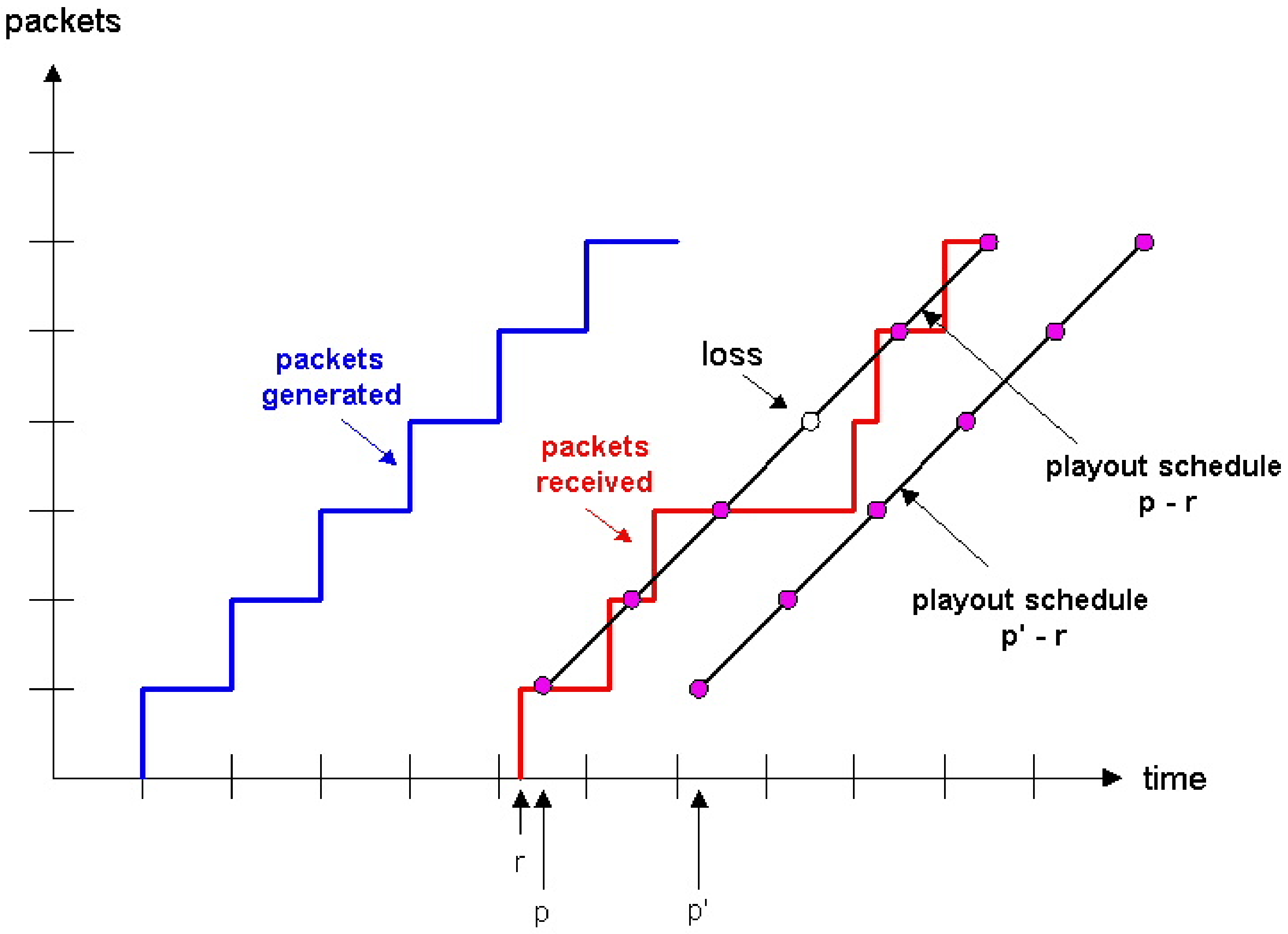
Playout Buffer

- Client buffer
  - Store the data as it arrives from the server
  - Play data for the user in a continuous fashion

- Playout delay
  - Client typically waits a few seconds to start playing
  - … to allow some data to build up in the buffer
  - … to help tolerate some delays down the road



Influence of Playout Delay

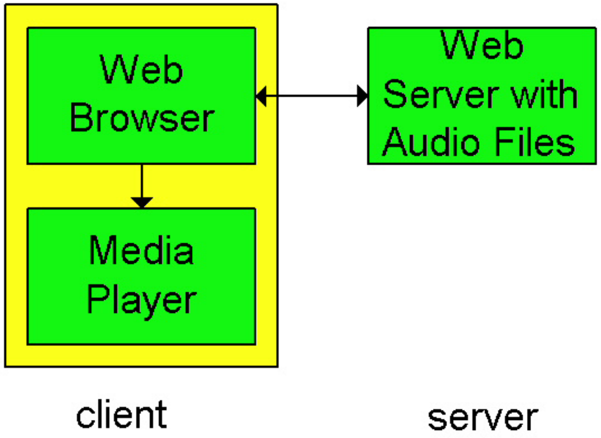


Requirements for Data Transport

- Delay
  - Some small delay at the beginning is acceptable
  - E.g., start-up delays of a few seconds are okay
- Jitter
  - Variability of packet delay within the same packet stream
  - Client cannot tolerate high variation if the buffer starves
- Loss
  - Small amount of missing data does not disrupt playback
  - Retransmitting a lost packet might take too long anyway

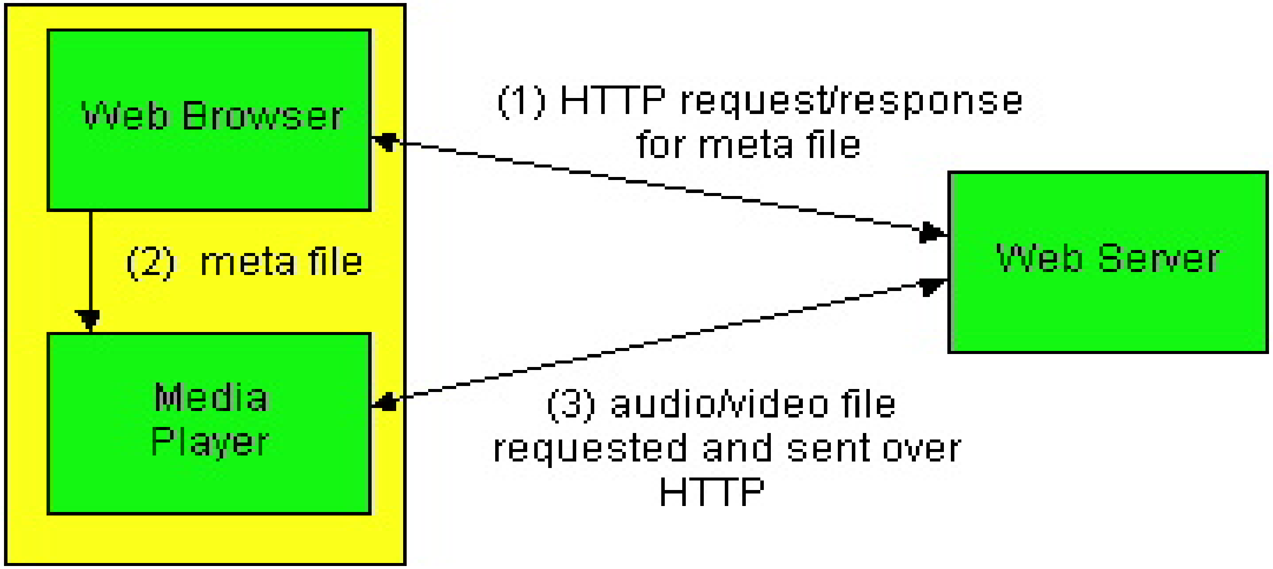
Streaming From Web Servers

- Data stored in a file
  - Audio: an audio file
  - Video: interleaving of audio and images in a single file
- HTTP request-response
  - TCP connection between client and server
  - Client HTTP request and server HTTP response
- Client invokes the media player
  - Content-type indicates the encoding
  - Browser launches the media player
  - Media player then renders the file



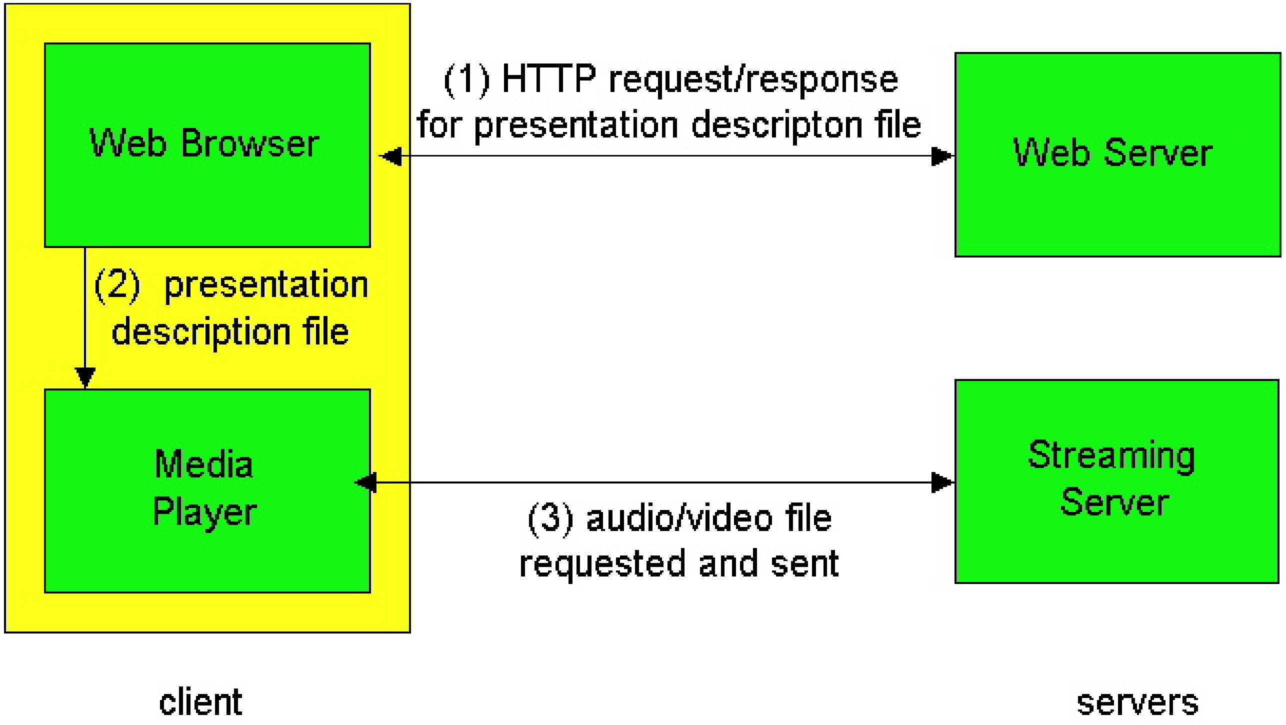
Initiating Streams from Web Servers

- Avoid passing all data through the Web browser
  - Web server returns a meta file describing the object
  - Browser launches media player and passes the meta file
  - The player sets up its own connection to the Web server


Web Browser
(1) HTTP request/response for meta file
Web Server
(2) meta file
Media Player
(3) audio/video file requested and sent over HTTP

Using a Streaming Server

- Avoiding the use of HTTP (and perhaps TCP, too)
  - Web server returns a meta file describing the object
  - Player requests the data using a different protocol



TCP is Not a Good Fit

- Reliable delivery
  - Retransmission of lost packets
  - … even though it may not be useful
- Adapting the sending rate
  - Slowing down after a packet loss
  - … even though it may cause starvation at the client
- Protocol overhead
  - TCP header of 20 bytes in every packet
  - … which is large for sending audio samples
  - Sending ACKs for every other packet
  - … which may be more feedback than needed

Better Ways of Transporting Data

- User Datagram Protocol (UDP)
  - No automatic retransmission of lost packets
  - No automatic adaptation of sending rate
  - Smaller packet header

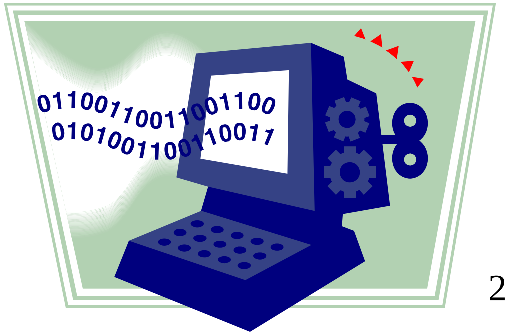
- UDP leaves many things up to the application
  - When to transmit the data
  - Whether to retransmit lost data
  - Whether to adapt the sending rate
  - … or adapt the quality of the audio/video encoding

Recovering From Packet Loss

- Loss is defined in a broader sense
  - Does a packet arrive in time for playback?
  - A packet that arrives late is as good as lost
  - Retransmission is not useful if the deadline has passed
- Selective retransmission
  - Sometimes retransmission is acceptable
  - E.g., if the client has not already started playing the data
  - Data can be retransmitted within the time constraint

Forward Error Correction (FEC)

- Forward error correction
  - Add redundant information to the packet stream
  - So the client can reconstruct data even after a loss
- Send redundant chunk after every n chunks
  - E.g., extra chunk is an XOR of the other n chunks
  - Receiver can recover from losing a single chunk
- Send low-quality version along with high quality
  - E.g., 13 kbps audio along with 64 kbps version
  - Receiver can play low quality version if the high-quality version is lost

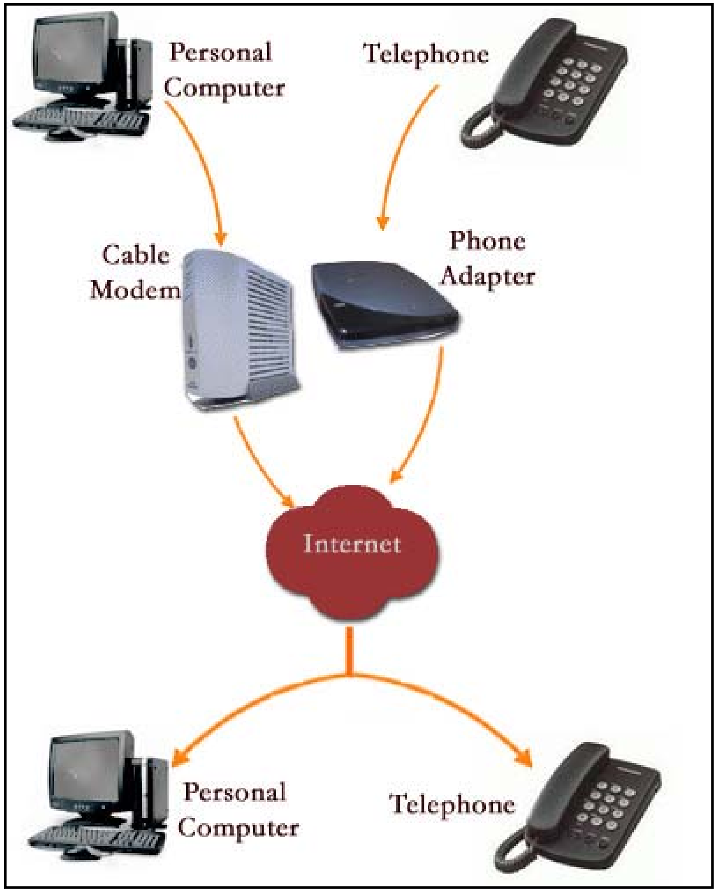


Interactive Audio and Video

- Two or more users interacting
  - Telephone call
  - Video conference
  - Video game
- Strict delay constraints
  - Delays over 150-200 msec are very noticeable
  - … and delays over 400 msec are a disaster for voice
- Much harder than streaming applications
  - Receiver cannot introduce much playout delay
  - Difficult if the network does not guarantee performance

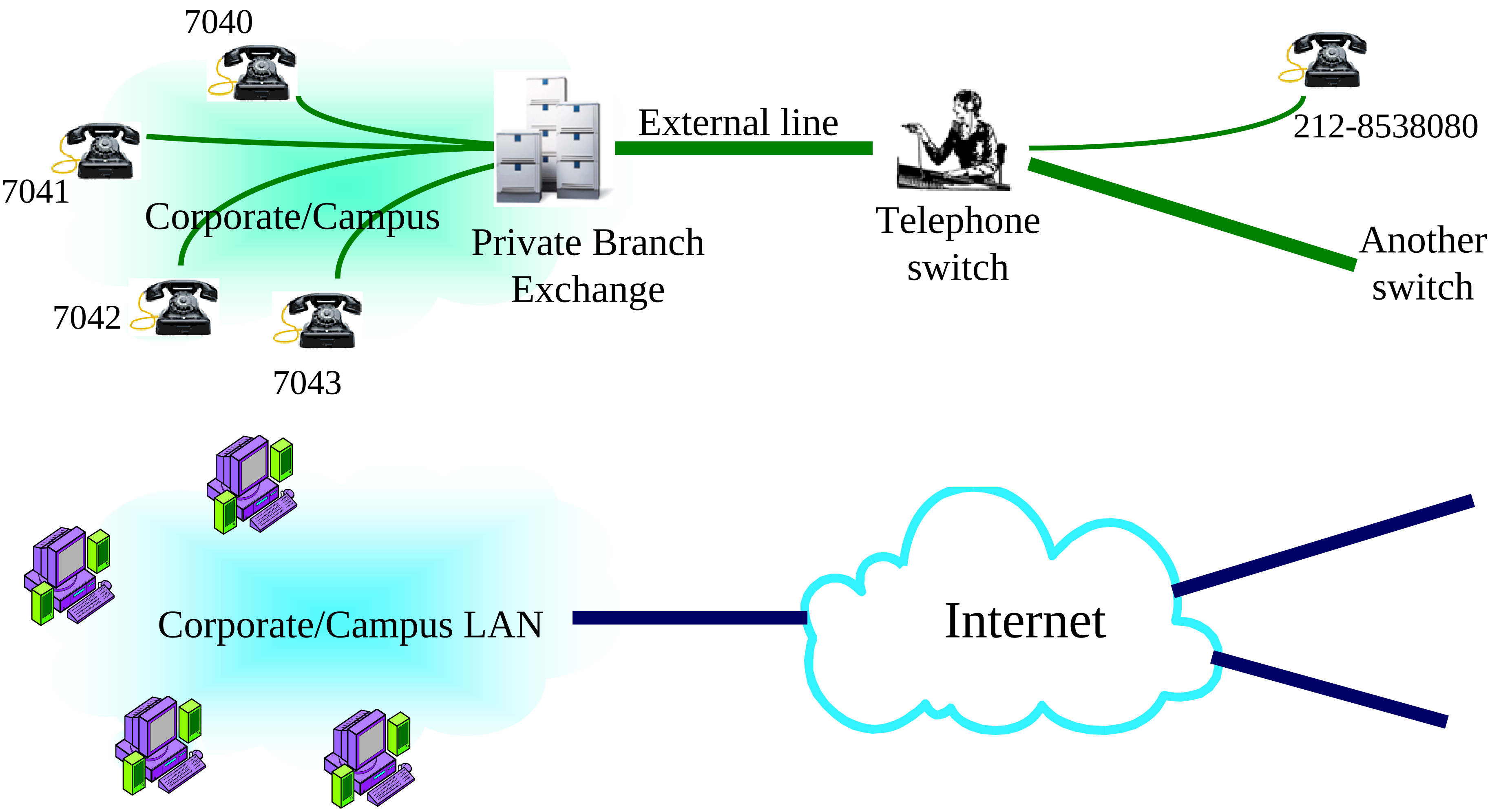
Voice Over IP (VoIP)

- Delivering phone calls over IP
  - Computer to computer
  - Analog phone to/from computer
  - Analog phone to analog phone



- Motivations for VoIP
  - Cost reduction
  - Simplicity
  - Advanced applications
    - Web-enabled call centers
    - Collaborative white boarding
    - Do Not Disturb, Locate Me, etc.
    - Voicemail sent as e-mail

Traditional Telecom Infrastructure



7040

7041

Corporate/Campus

7042

7043

Private Branch Exchange

External line

Telephone switch

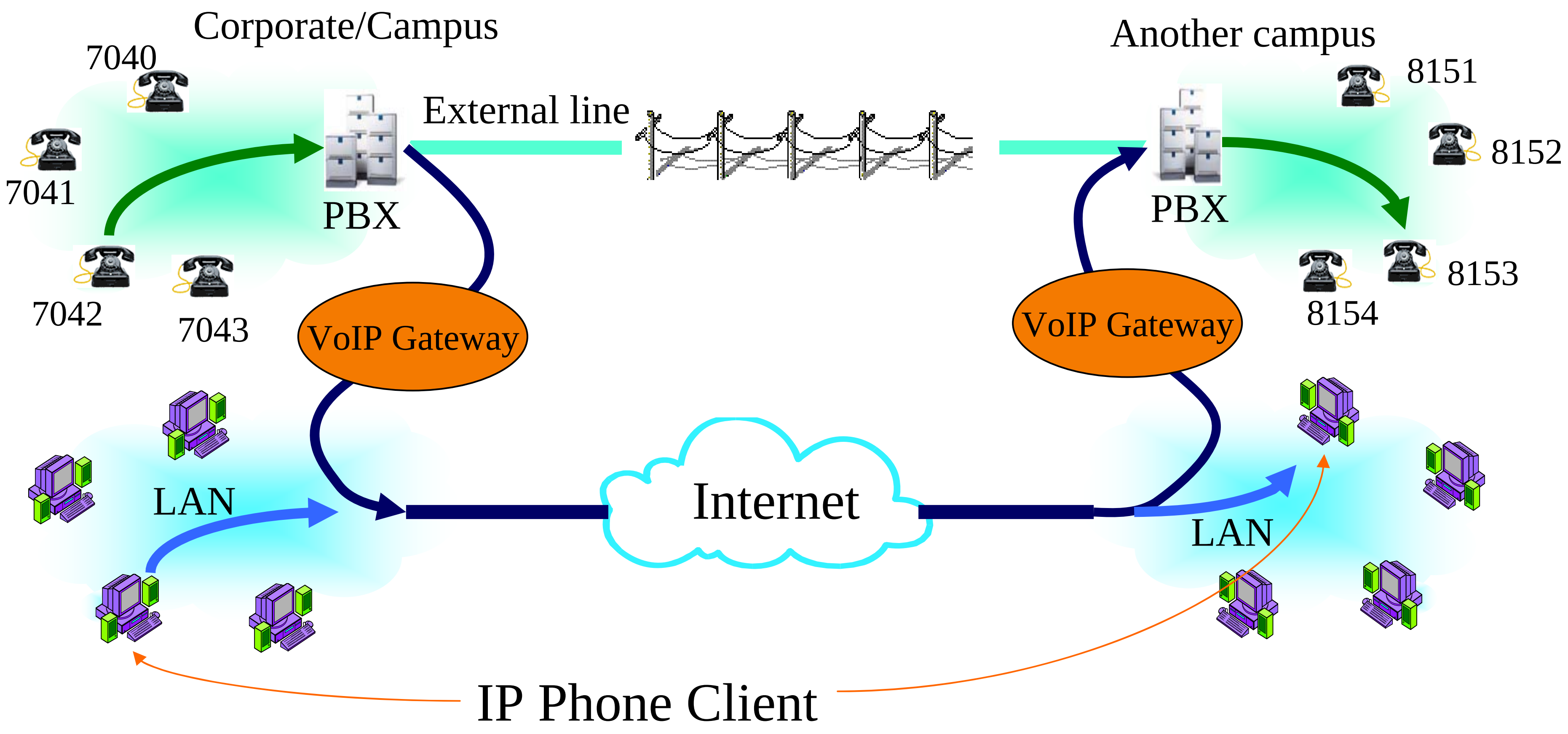
212-8538080

Another switch

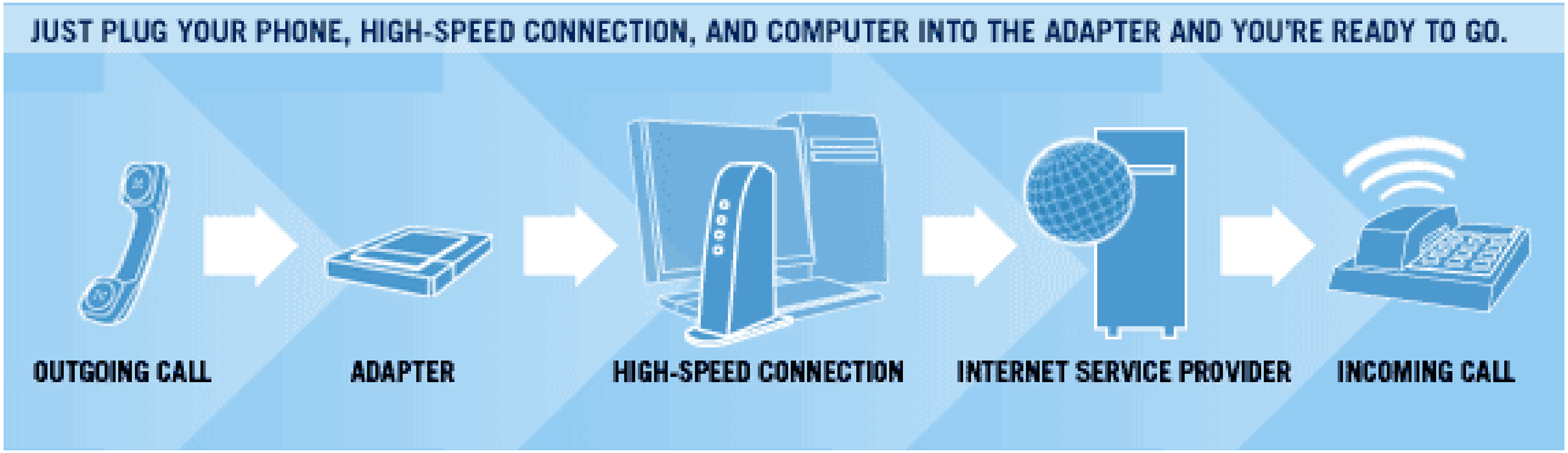
Corporate/Campus LAN

Internet

VoIP Gateways



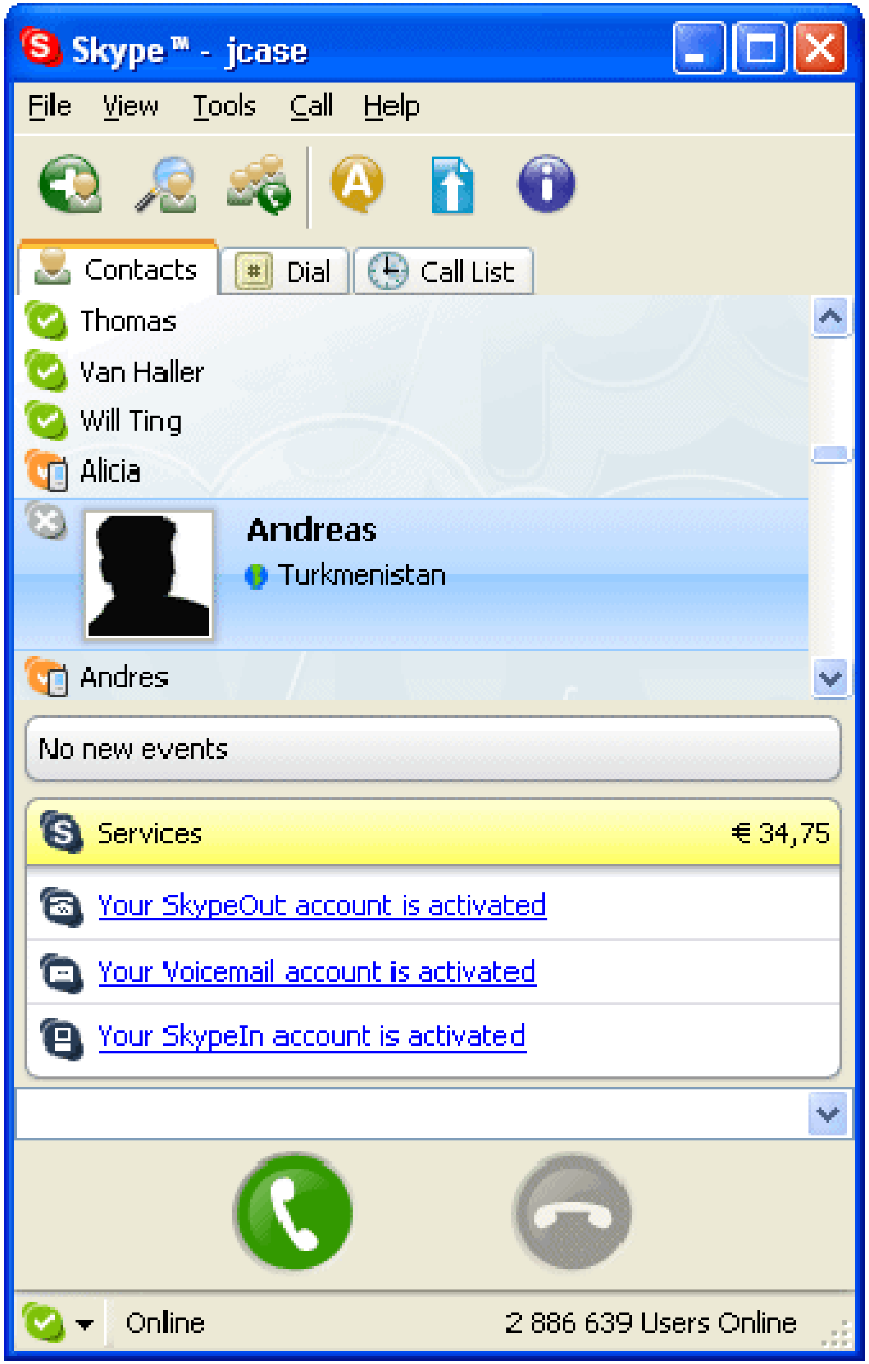
VoIP With an Analog Phone


JUST PLUG YOUR PHONE, HIGH-SPEED CONNECTION, AND COMPUTER INTO THE ADAPTER AND YOU'RE READY TO GO.
OUTGOING CALL
ADAPTER
HIGH-SPEED CONNECTION
INTERNET SERVICE PROVIDER
INCOMING CALL

- Adapter
  - Converts between analog and digital
  - Sends and receives data packets
  - Communicates with the phone in standard way

Skype

- Niklas Zennström and Janus Friis in 2003
- Developed by KaZaA
- Instant Messenger (IM) with voice support
- Based on peer-to-peer (P2P) networking technology



Thomas
Van Haller
Will Ting
Alicia
Andreas
Turkmenistan
Andres

No new events

Services € 34,75

Your SkypeOut account is activated

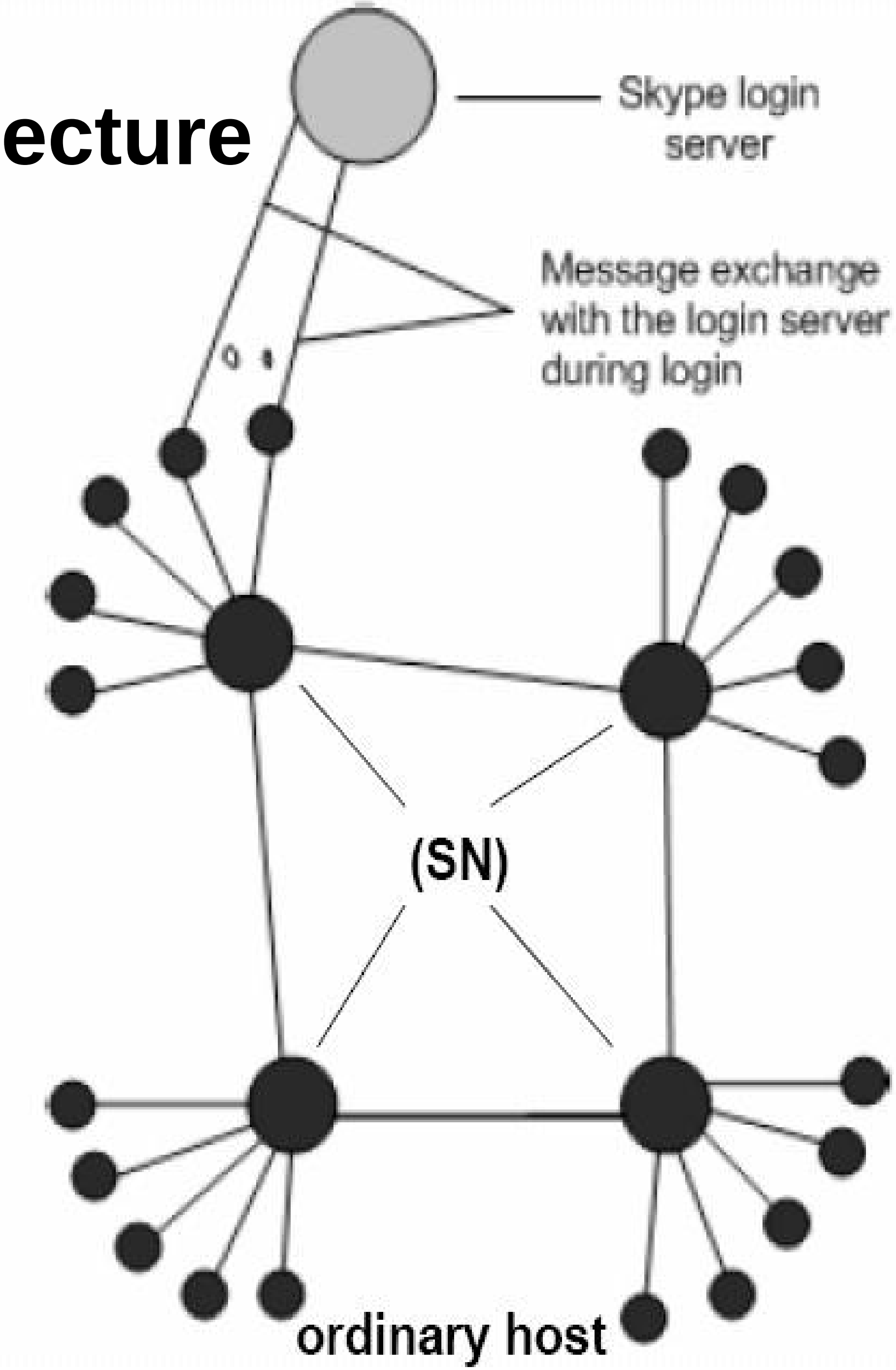
Your Voicemail account is activated

Your SkypeIn account is activated

Online 2 886 639 Users Online

Skype Network Architecture

- Login server is the only central server (consisting of multiple machines)
- Both ordinary host and super nodes are Skype clients
- Any node with a public IP address and having sufficient resources can become a super node
- Skype maintains their own super nodes



Skype Data Transfer

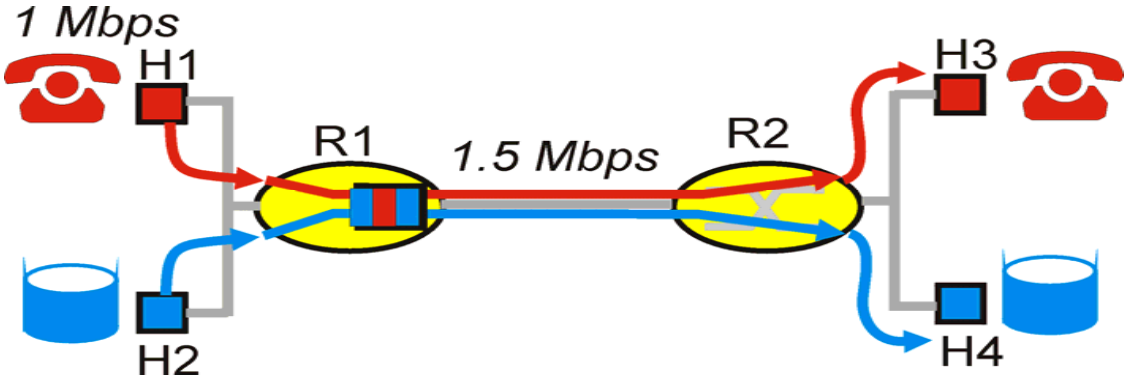
- Audio compression
  - Voice packets around 67 bytes
  - Up to 140 packets per second
  - Around 5 KB/sec (40 kbps) in each direction
- Encryption
  - Data packets are encrypted in both directions
  - To prevent snooping on the phone call
  - … by someone snooping on the network
  - … or by the intermediate peers forwarding data

VoIP Quality

- The application can help
  - Good audio compression algorithms
  - Avoiding hops through far-away hosts
  - Forward error correction
  - Adaptation to the available bandwidth
- But, ultimately the network is a major factor
  - Long propagation delay?
  - High congestion?
  - Disruptions during routing changes?
- Leads to an interest in Quality of Service

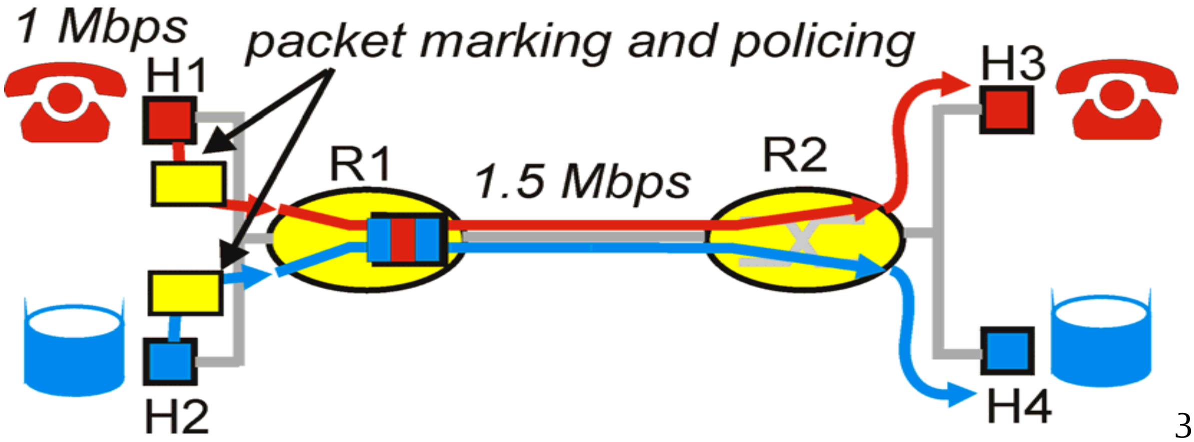
Principles for QoS Guarantees

- Applications compete for bandwidth
  - Consider a 1 Mbps VoIP application and an FTP transfer sharing a single 1.5 Mbps link
  - Bursts of FTP traffic can cause congestion and losses
  - We want to give priority to the audio packets over FTP
- Principle 1: Packet marking
  - Marking of packets is needed for the router
  - To distinguish between different classes



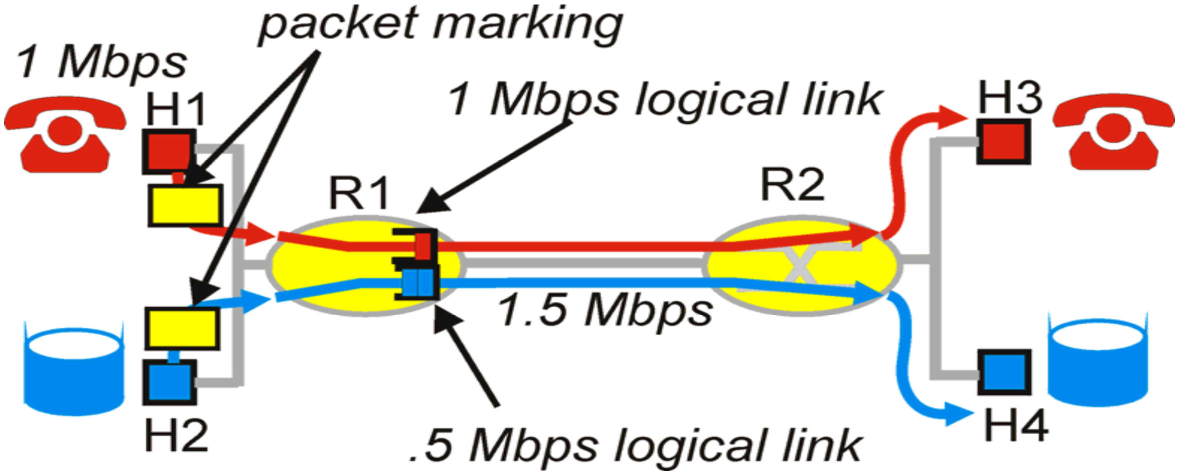
Principles for QoS Guarantees

- Applications misbehave
  - Audio sends packets at a rate higher than 1 Mbps
- Principle 2: Policing
  - Provide protection for one class from other classes
  - Ensure sources adhere to bandwidth restrictions
  - Marking and policing need to be done at the edge



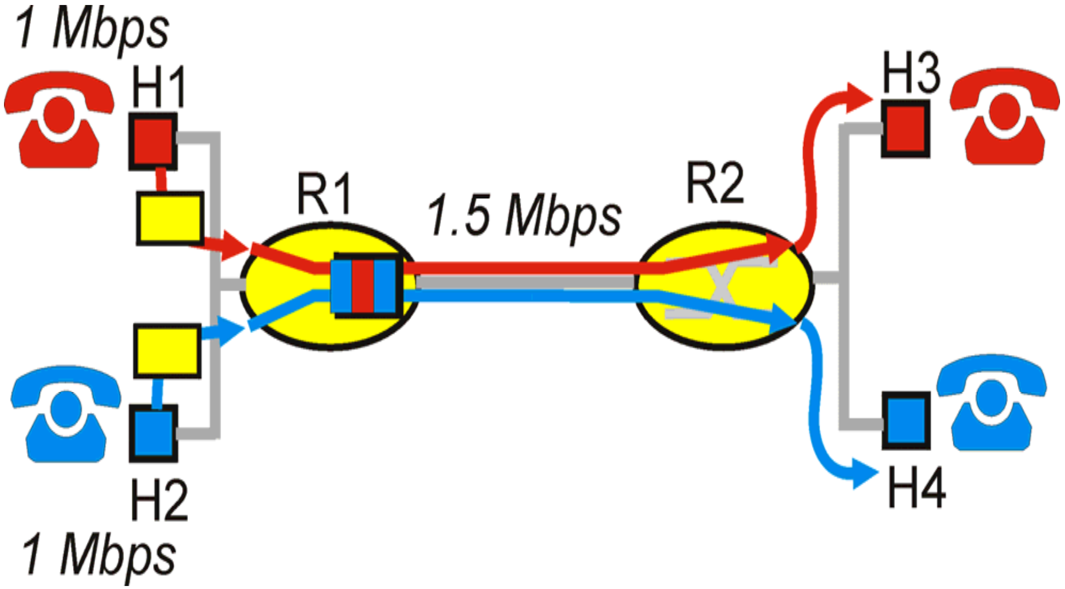
Principles for QoS Guarantees

- Alternative to marking and policing
  - Allocate fixed bandwidth to each application flow
  - But, this can lead to inefficient use of bandwidth
  - … if one of the flows does not use its allocation
- Principle 3: Link scheduling
  - While providing isolation, it is desirable to use resources as efficiently as possible
  - E.g., weighted fair queuing or round-robin scheduling



Principles for QoS Guarantees

- Cannot support traffic beyond link capacity
  - If total traffic exceeds capacity, you are out of luck
  - Degrade the service for all, or deny someone access
- Principle 4: Admission control
  - Application flow declares its needs in advance
  - The network may block call if it cannot satisfy the needs



Quality of Service

- Significant change to Internet architecture
  - Guaranteed service rather than best effort
  - Routers keeping state about the traffic
- A variety of new protocols and mechanisms
  - Reserving resources along a path
  - Identifying paths with sufficient resources
  - Link scheduling and buffer management
  - Packet marking with the Type-of-Service bits
  - Packet classifiers to map packets to ToS classes
  - …
- Seeing some deployment within individual ASes
  - E.g., corporate/campus networks, and within an ISP

Conclusions

- Digital audio and video
  - Increasingly popular media on the Internet
  - Video on demand, VoIP, online gaming, IPTV, …
- Interaction with the network
  - Adapt to delivering the data over a best-effort network
  - Adapt the network to offer better quality-of-service

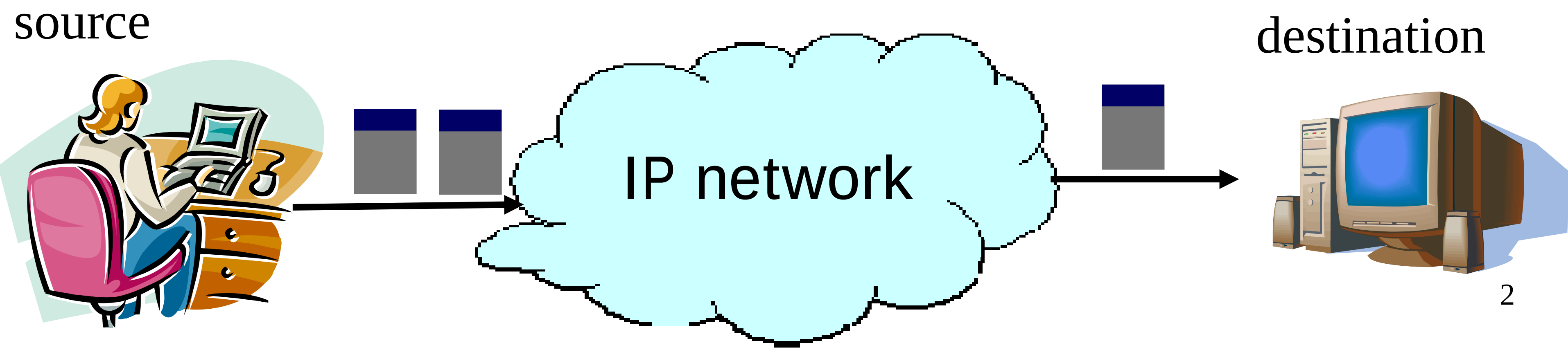
Middleboxes

Introduction to Data Networks

2008.5

Network-Layer Principles

- Globally unique identifiers
  - Each node has a unique, fixed IP address
  - … reachable from everyone and everywhere
- Simple packet forwarding
  - Network nodes simply forward packets
  - … rather than modifying or filtering them



source

IP network

destination

Internet Reality

- Host mobility
  - –Changes in IP addresses as hosts move
- IP address depletion
  - –Dynamic assignment of IP addresses
  - –Use of private addresses
- Security concerns
  - –Discarding suspicious or unwanted packets
  - –Detecting suspicious traffic
- Performance concerns
  - –Controlling how link bandwidth is allocated
  - –Storing popular Web content near the clients

Middleboxes

- Middleboxes are intermediaries
  - Interposed in-between the communicating hosts
  - Often without knowledge of one or both parties
- Examples
  - Network address translators
  - Firewalls
  - Traffic shapers
  - Intrusion detection systems
  - Transparent Web proxy caches

Two Views of Middleboxes

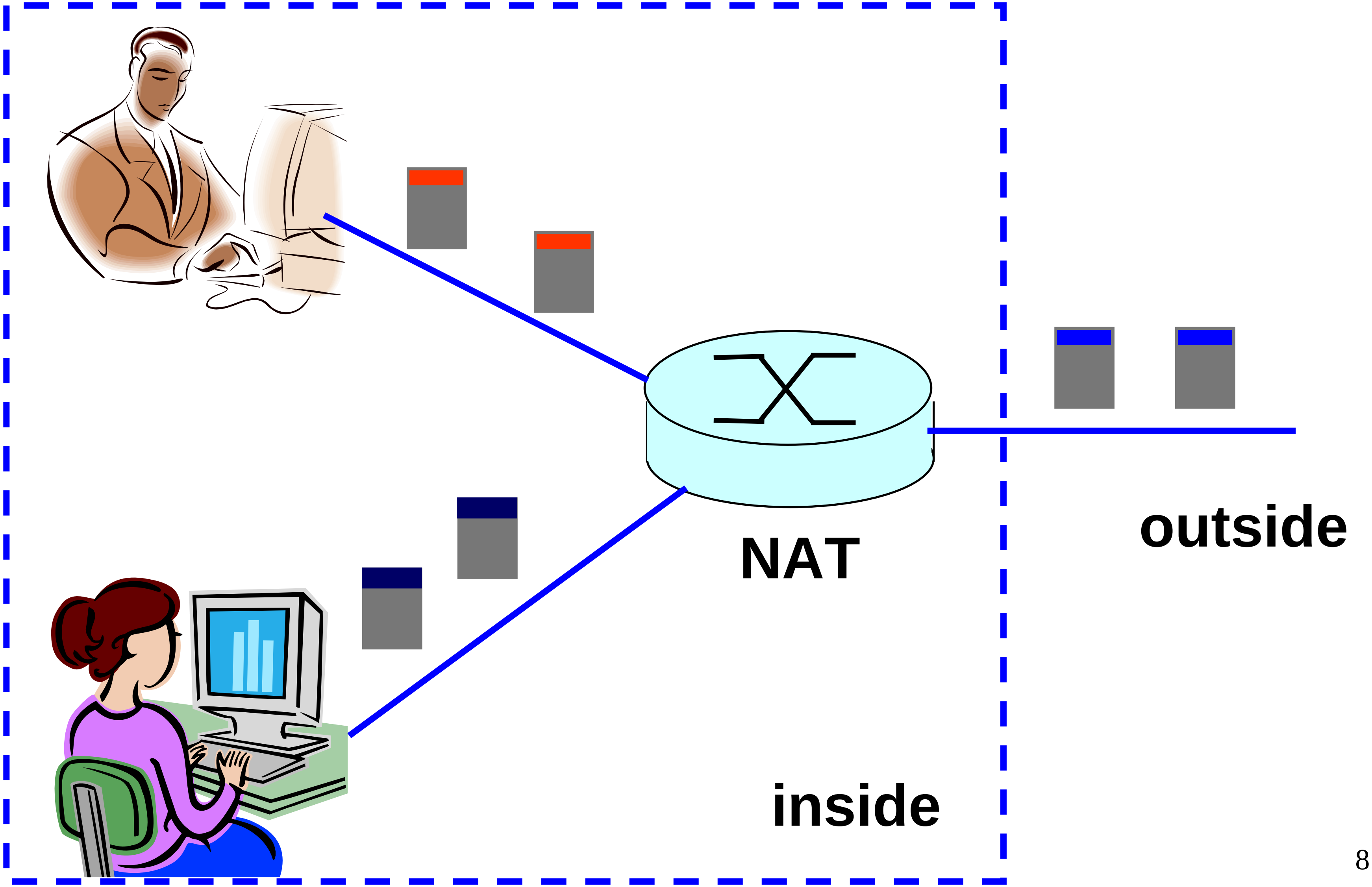
- An abomination
  - –Violation of layering
  - –Cause confusion in reasoning about the network
  - –Responsible for many subtle bugs
- A necessity
  - –Solving real and pressing problems
  - –Needs that are not likely to go away

Network Address Translation

History of NATs

- IP address space depletion
  - Clear in early 90s that 2^{32} addresses not enough
  - Work began on a successor to IPv4
- In the meantime…
  - Share addresses among numerous devices
  - … without requiring changes to existing hosts
- Meant to provide temporary relief
  - Intended as a short-term remedy
  - Now, NAT are very widely deployed
  - … much moreso than IPv6

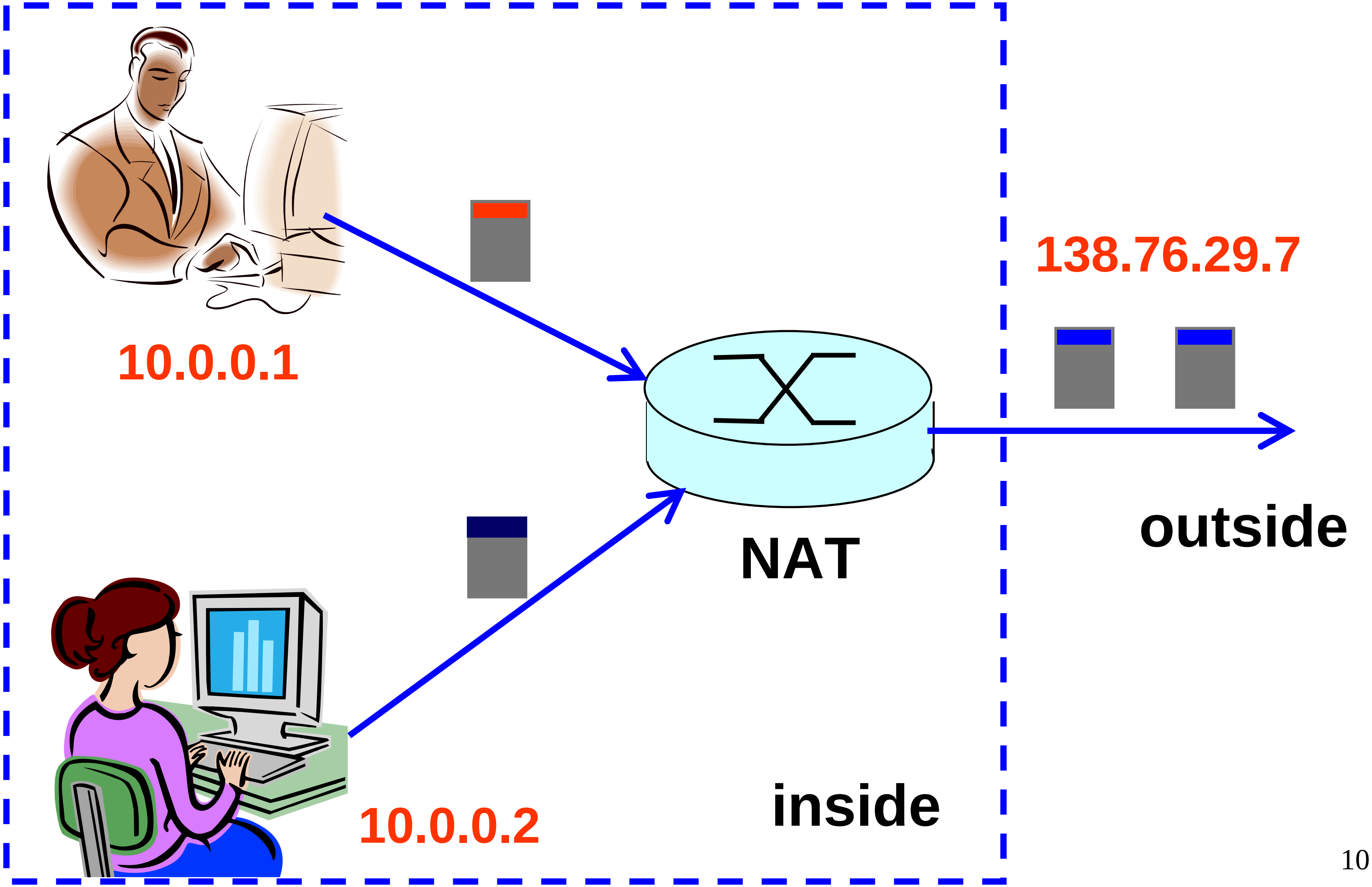
Active Component in the Data Path



IP Header Translators

- Local network addresses not globally unique
  - E.g., private IP addresses (in 10.0.0.0/8)
- NAT box rewrites the IP addresses
  - Make the “inside” look like a single IP address
  - … and change header checksums accordingly
- Outbound traffic: from inside to outside
  - Rewrite the source IP address
- Inbound traffic: from outside to inside
  - Rewrite the destination IP address

Using a Single Source Address



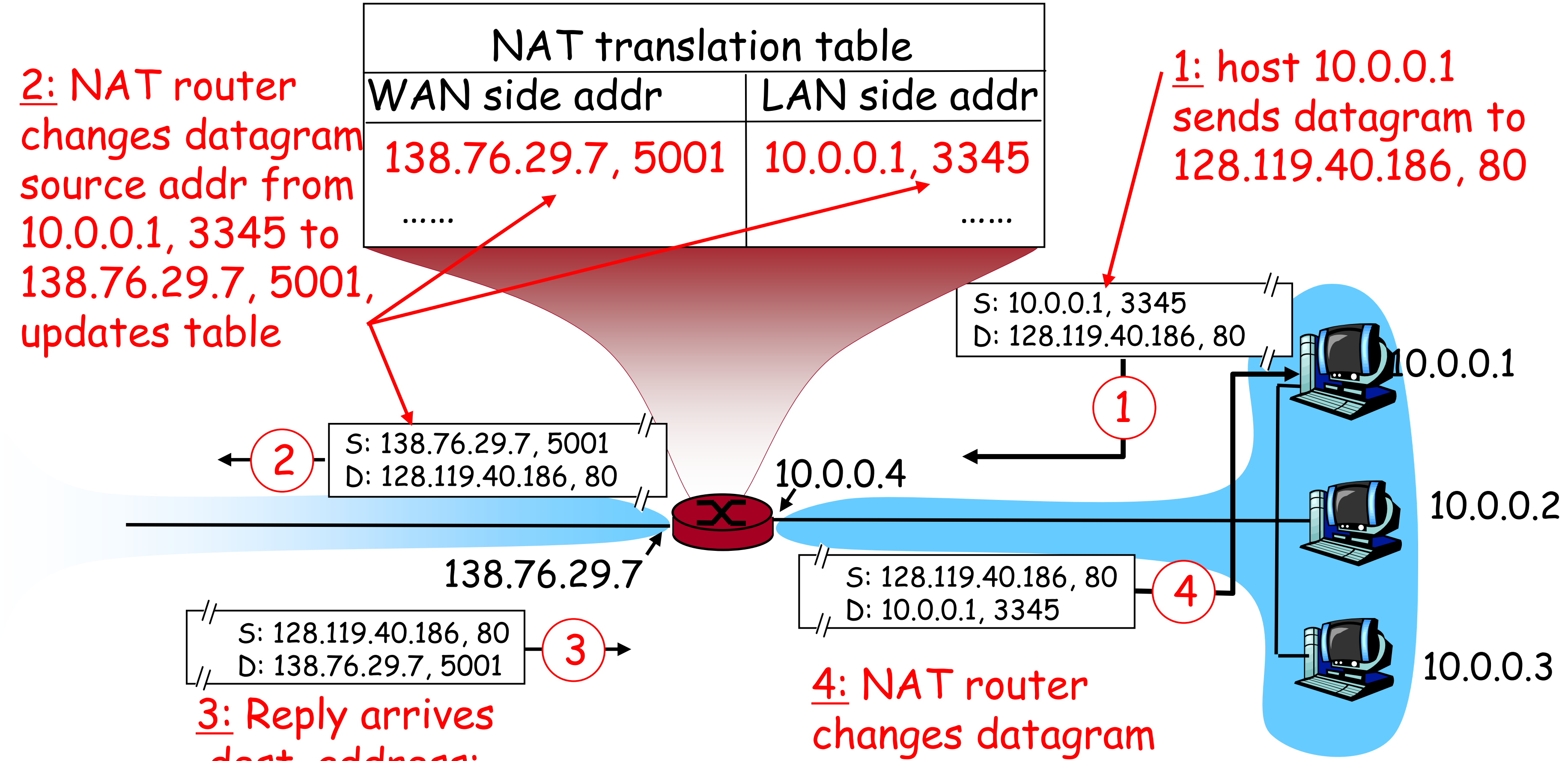
What if Both Hosts Contact Same Site?

- Suppose hosts contact the same destination
  - E.g., both hosts open a socket with local port 3345 to destination 128.119.40.186 on port 80
- NAT gives packets same source address
  - All packets have source address 138.76.29.7
- Problems
  - Can destination differentiate between senders?
  - Can return traffic get back to the correct hosts?

Port-Translating NAT

- Map outgoing packets
  - Replace source address with NAT address
  - Replace source port number with a new port number
  - Remote hosts respond using (NAT address, new port #)
- Maintain a translation table
  - Store map of (source address, port #) to (NAT address, new port #)
- Map incoming packets
  - Consult the translation table
  - Map the destination address and port number
  - Local host receives the incoming packet

Network Address Translation Example



NAT translation table	
WAN side addr	LAN side addr
138.76.29.7, 5001	10.0.0.1, 3345
……	……

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80

S: 10.0.0.1, 3345
D: 128.119.40.186, 80

2: NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

S: 138.76.29.7, 5001
D: 128.119.40.186, 80

S: 128.119.40.186, 80
D: 138.76.29.7, 5001

3: Reply arrives dest. address: 138.76.29.7, 5001

S: 128.119.40.186, 80
D: 10.0.0.1, 3345

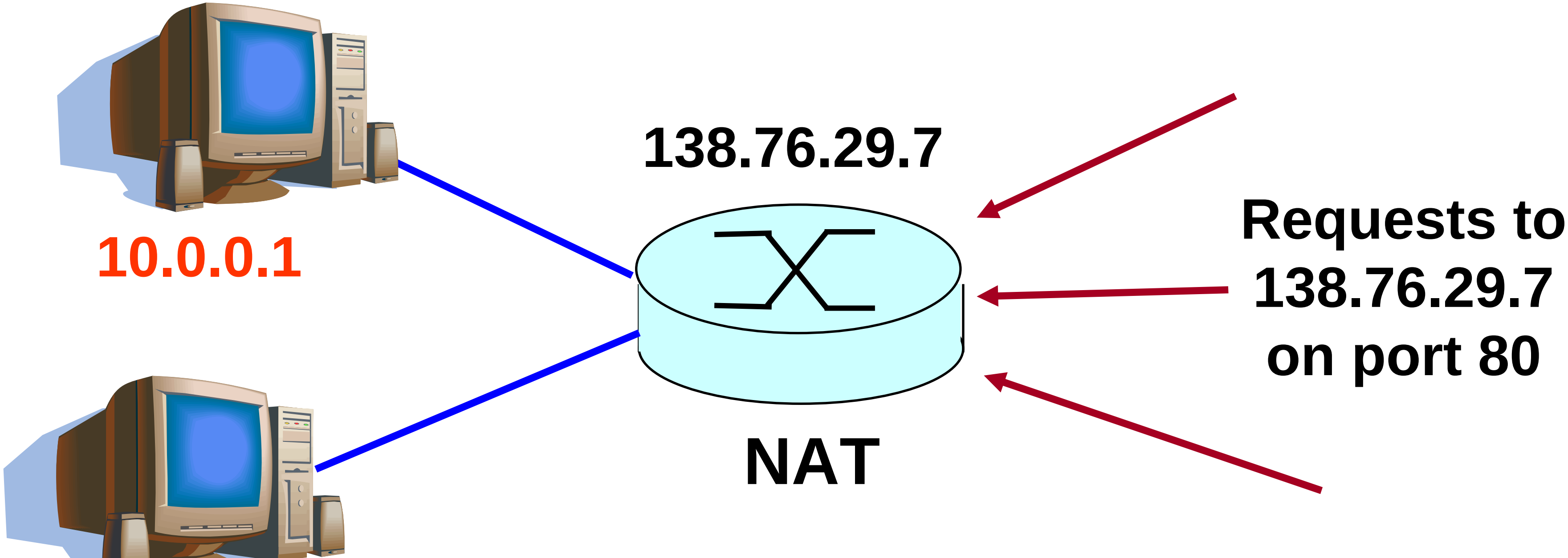
4: NAT router changes datagram dest addr from 138.76.29.7, 5001 to 10.0.0.1, 3345

Maintaining the Mapping Table

- Create an entry upon seeing a packet
  - –Packet with new (source addr, source port) pair
- Eventually, need to delete the map entry
  - –But when to remove the binding?
- If no packets arrive within a time window
  - –… then delete the mapping to free up the port #s
- Yet another example of “soft state”
  - –I.e., removing state if not refreshed for a while

Objections Against NAT

- Port #s are meant for addressing processes
  - Yet, NAT uses them to identify end hosts
  - Makes it hard to run a server behind a NAT



138.76.29.7

10.0.0.1

NAT

Requests to 138.76.29.7 on port 80

10.0.0.2

Which host should get the request???

Objections Against NAT

- Difficult to support peer-to-peer applications
  - P2P needs a host to act as a server
  - … difficult if both hosts are behind NATs
- Routers are not supposed to look at port #s
  - Network layer should care only about IP header
  - … and not be looking at the port numbers at all
- NAT violates the end-to-end argument
  - Network nodes should not modify the packets
- IPv6 is a cleaner solution
  - Better to migrate than to limp along with a hack

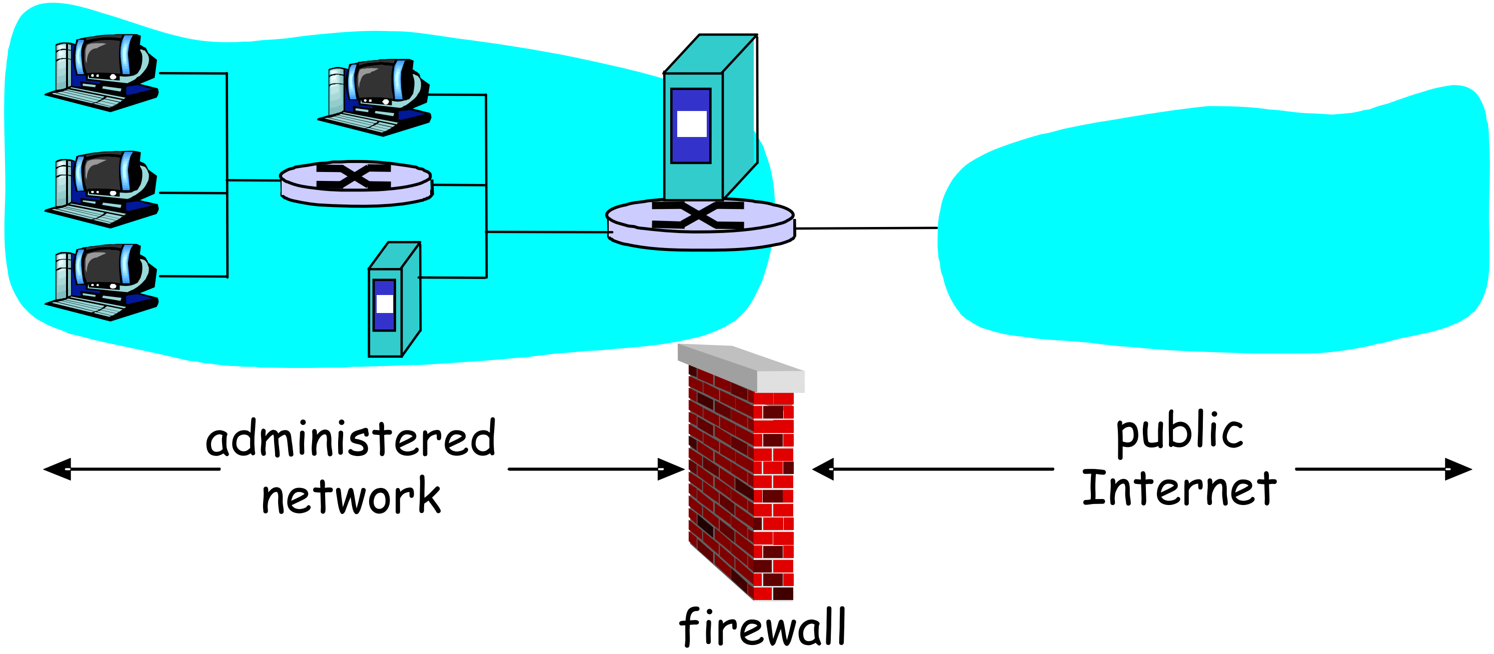
Where is NAT Implemented?

- Home router (e.g., Linksys box)
  - Integrates router, DHCP server, NAT, etc.
  - Use single IP address from the service provider
  - … and have a bunch of hosts hiding behind it
- Campus or corporate network
  - NAT at the connection to the Internet
  - Share a collection of public IP addresses
  - Avoid complexity of renumbering end hosts and local routers when changing service providers

Firewalls

Firewalls

Isolates organization's internal net from larger Internet, allowing some packets to pass, blocking others.



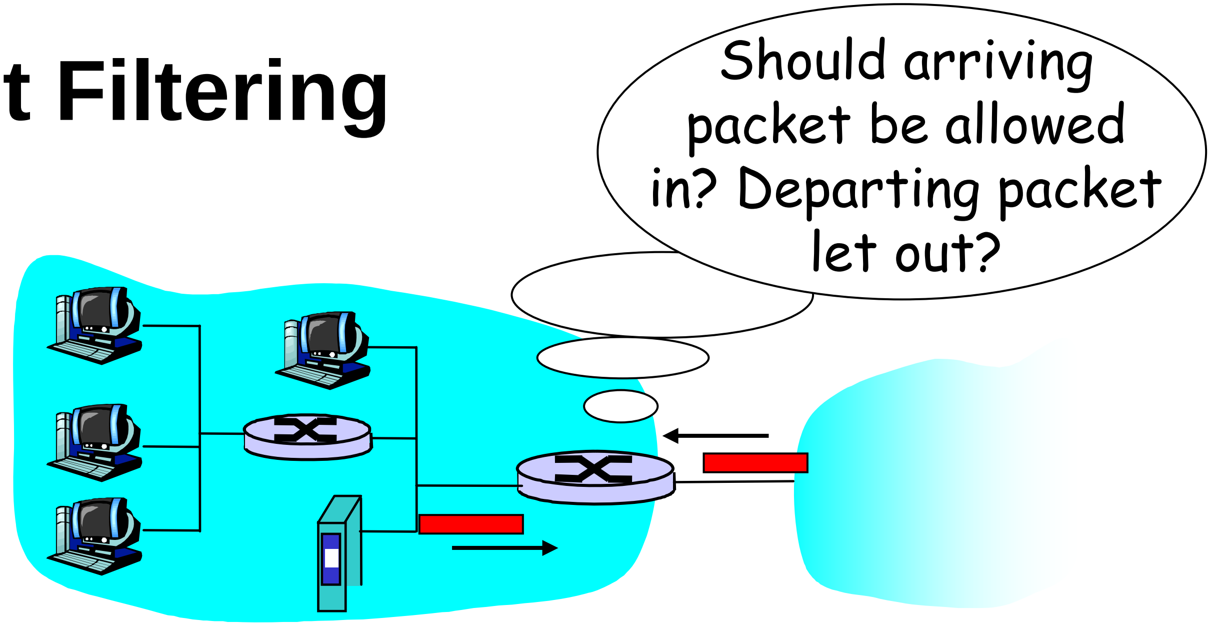
Internet Attacks: Denial of Service

- Denial-of-service attacks
  - Outsider overwhelms the host with unsolicited traffic
  - … with the goal of preventing any useful work
- Example
  - Bad guys take over a large collection of hosts
  - … and program these hosts to send traffic to your host
  - Leading to excessive traffic
- Motivations for denial-of-service attacks
  - Malice (e.g., just to be mean)
  - Revenge (e.g. for some past perceived injustice)
  - Greed (e.g., blackmailing)

Internet Attacks: Break-Ins

- Breaking in to a host
  - Outsider exploits a vulnerability in the end host
  - … with the goal of changing the behavior of the host
- Example
  - Bad guys know a Web server has a buffer-overflow vulnerability
  - … and, say, send an HTTP request with a long URL
  - Allowing them to break in
- Motivations for break-ins
  - Take over the machine to launch other attacks
  - Steal information stored on the machine
  - Modify/replace the content the site normally returns

Packet Filtering


Should arriving packet be allowed in? Departing packet let out?

- Internal network connected to Internet via firewall
- Firewall filters packet-by-packet, based on:
  - Source IP address, destination IP address
  - TCP/UDP source and destination port numbers
  - ICMP message type
  - TCP SYN and ACK bits

Packet Filtering Examples

- Block all packets with IP protocol field = 17 and with either source or dest port = 23.
  - All incoming and outgoing UDP flows blocked
  - All Telnet connections are blocked
- Block inbound TCP packets with SYN but no ACK
  - Prevents external clients from making TCP connections with internal clients
  - But allows internal clients to connect to outside
- Block all packets with TCP port of Doom3

Firewall Configuration

- Firewall applies a set of rules to each packet
  - To decide whether to permit or deny the packet
- Each rule is a test on the packet
  - Comparing IP and TCP/UDP header fields
  - … and deciding whether to permit or deny
- Order matters
  - Once the packet matches a rule, the decision is done

Firewall Configuration Example

- Alice runs a network in 222.22.0.0/16
  - Wants to let Bob's school access certain hosts
    - Bob is on 111.11.0.0/16
    - Alice's special hosts on 222.22.22.0/24
  - Alice doesn't trust Trudy, inside Bob's network
    - Trudy is on 111.11.11.0/24
  - Alice doesn't want any other traffic from Internet
- Rules
  - #1: Don't let Trudy machines in
    - Deny (src = 111.11.11.0/24, dst = 222.22.0.0/16)
  - #2: Let rest of Bob's network in to special dsts
    - Permit (src=111.11.0.0/16, dst = 222.22.22.0/24)
  - #3: Block the rest of the world
    - Deny (src = 0.0.0.0/0, dst = 0.0.0.0/0)

A Variation: Traffic Management

- Permit vs. deny is a too binary decision
  - Maybe better to classify the traffic based on rules
  - … and then handle the classes of traffic differently
- Traffic shaping (rate limiting)
  - Limit the amount of bandwidth for certain traffic
  - E.g., rate limit on Web or P2P traffic
- Separate queues
  - Use rules to group related packets
  - And then do round-robin scheduling across the groups
  - E.g., separate queue for each internal IP address

Firewall Implementation Challenges

- Per-packet handling
  - Must inspect every packet
  - Challenging on very high-speed links
- Complex filtering rules
  - May have large # of rules
  - May have very complicated rules
- Location of firewalls
  - Complex firewalls near the edge, at low speed
  - Simpler firewalls in the core, at higher speed

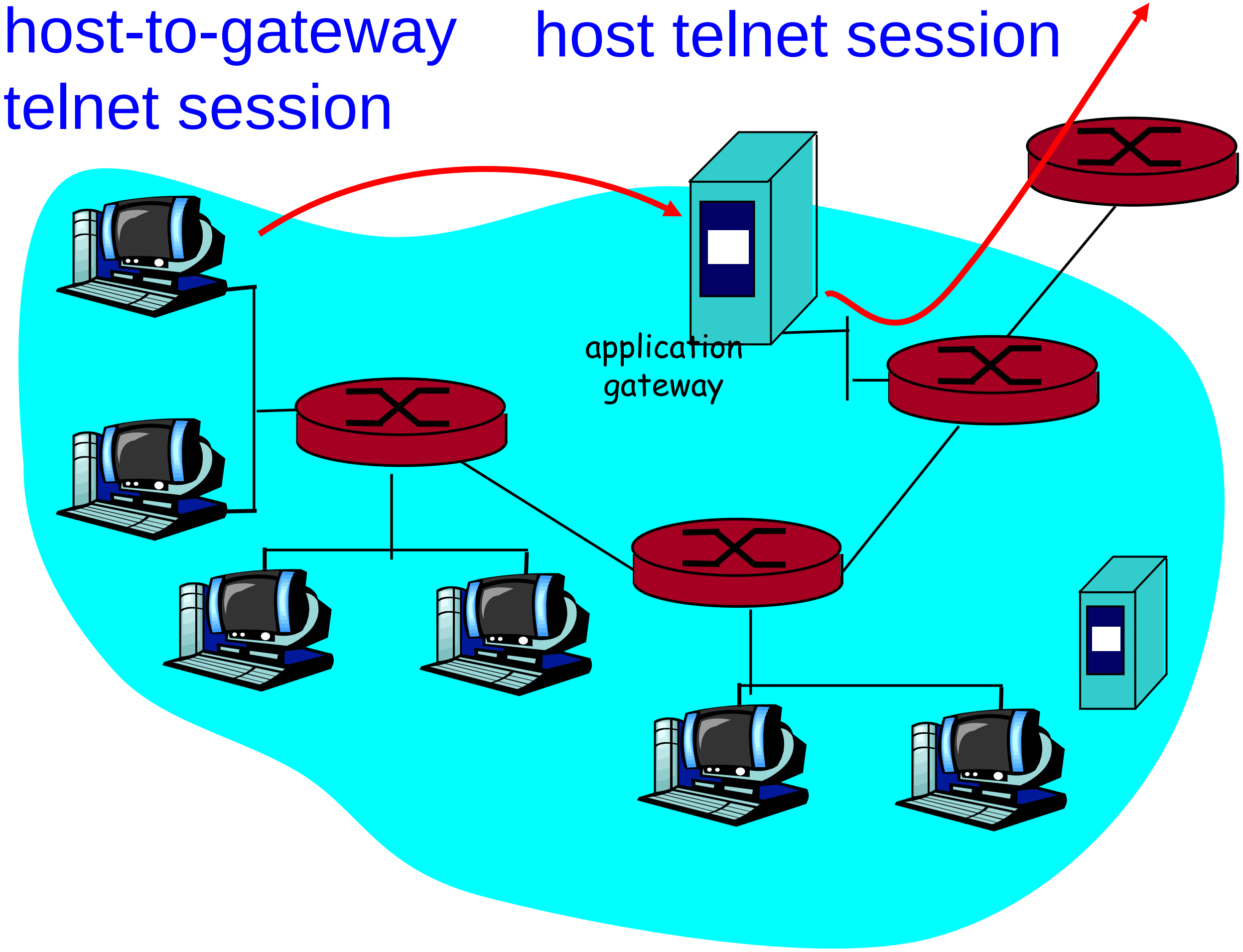
Clever Users Subvert Firewalls

- Example: filtering dorm access to a server
  - Firewall rule based on IP addresses of dorms
  - … and the server IP address and port number
  - Problem: users may log in to another machine
    - E.g., connect from the dorms to another host
    - … and then onward to the blocked server
- Example: filtering P2P based on port #s
  - Firewall rule based on TCP/UDP port numbers
    - E.g., allow only port 80 (e.g., Web) traffic
  - Problem: software using non-traditional ports
    - E.g., write P2P client to use port 80 instead

Application Gateways

- Filter packets on application data
  - Not just on IP and TCP/UDP headers
- Example: restricting Telnet usage
  - Don’t allow any external clients to Telnet inside
  - Only allow certain internal users to Telnet outside
- Solution: Telnet gateway
  - Force all Telnet traffic to go through a gateway
  - I.e. filter Telnet traffic that doesn’t originate from the IP address of the gateway
- At the gateway…
  - Require user to login and provide password
  - Apply policy to decide whether they can proceed

Telnet Gateway Example



gateway-to-remote host telnet session

host-to-gateway telnet session

firewall

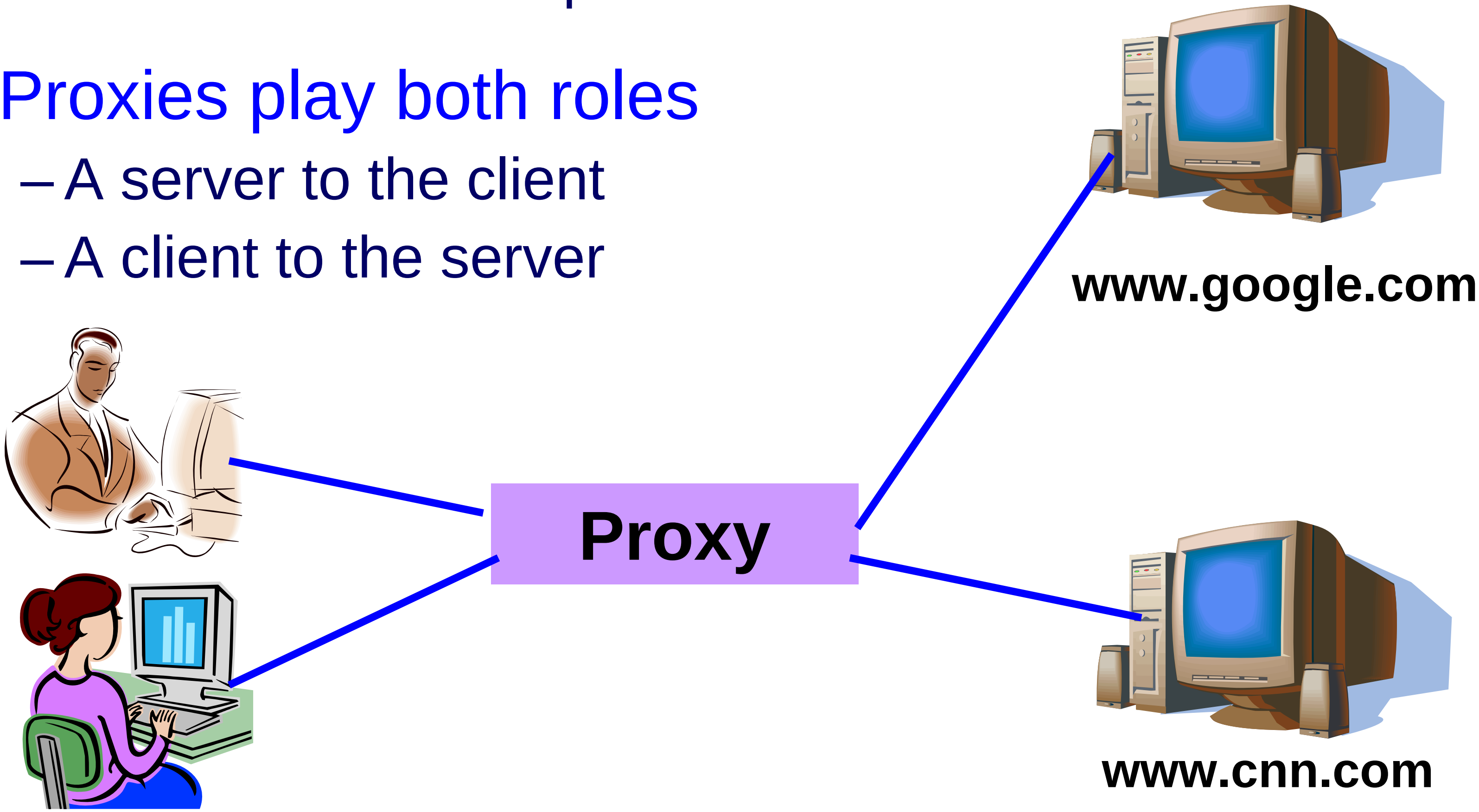
Motivation for Gateways

- Enable more detailed policies
  - E.g., login id and password at Telnet gateway
- Avoid rogue machines sending traffic
  - E.g., e-mail “server” running on user machines
  - … probably a sign of a spammer
- Enable a central place to perform logging
  - E.g., forcing all Web accesses through a gateway
  - … to log the IP addresses and URLs
- Improve performance through caching
  - E.g., forcing all Web accesses through a gateway
  - … to enable caching of the popular content

Web Proxies

Web Clients and Servers

- Web is a client-server protocol
  - Client sends a request
  - Server sends a response
- Proxies play both roles
  - A server to the client
  - A client to the server



Proxy Caching

- Client #1 requests http://www.foo.com/fun.jpg
  - Client sends “GET fun.jpg” to the proxy
  - Proxy sends “GET fun.jpg” to the server
  - Server sends response to the proxy
  - Proxy stores the response, and forwards to client
- Client #2 requests http://www.foo.com/fun.jpg
  - Client sends “GET fun.jpg” to the proxy
  - Proxy sends response to the client from the cache
- Benefits
  - Faster response time to the clients
  - Lower load on the Web server
  - Reduced bandwidth consumption inside the network

Getting Requests to the Proxy

- Explicit configuration
  - –Browser configured to use a proxy
  - –Directs all requests through the proxy
  - –Problem: requires user action
- Transparent proxy (or “interception proxy”)
  - –Proxy lies in path from the client to the servers
  - –Proxy intercepts packets en route to the server
  - –… and interposes itself in the data transfer
  - –Benefit: does not require user action

Challenges of Transparent Proxies

- Must ensure all packets pass by the proxy
  - By placing it at the only access point to the Internet
  - E.g., at the border router of a campus or company
- Overhead of reconstructing the requests
  - Must intercept the packets as they fly by
  - … and reconstruct into the ordered by stream
- May be viewed as a violation of user privacy
  - The user does not know the proxy lies in the path
  - Proxy may be keeping logs of the user’s requests

Other Functions of Web Proxies

- Anonymization
  - Server sees requests coming from the proxy address
  - … rather than the individual user IP addresses
- Transcoding
  - Converting data from one form to another
  - E.g., reducing the size of images for cell-phone browsers
- Prefetching
  - Requesting content before the user asks for it
- Filtering
  - Blocking access to sites, based on URL or content

Conclusions

- Middleboxes address important problems
  - Using fewer IP addresses
  - Blocking unwanted traffic
  - Making fair use of network resources
  - Improving Web performance
- Middleboxes cause problems of their own
  - No longer globally unique IP addresses
  - No longer can assume network simply delivers packets

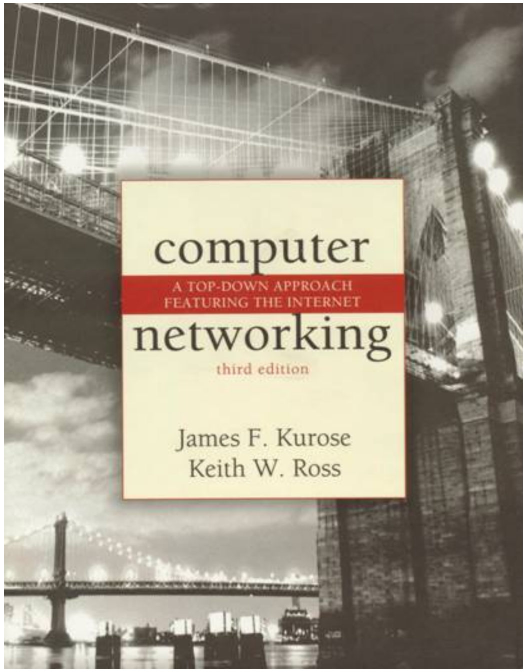
Chapter 8
Network Security

A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- ❑ If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- ❑ If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR



Computer Networking: A Top Down Approach Featuring the Internet, 3rd edition.
Jim Kurose, Keith Ross
Addison-Wesley, July 2004.

Chapter 8: Network Security

Chapter goals:

- understand principles of network security:
  - cryptography and its *many* uses beyond "confidentiality"
  - authentication
  - message integrity
  - key distribution
- security in practice:
  - security in application, transport, network, link layers

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Authentication

8.4 Integrity

8.5 Key Distribution and certification

8.6 Access control: firewalls

8.7 Attacks and counter measures

8.8 Security in many layers

What is network security?

Confidentiality: only sender, intended receiver should “understand” message contents

- sender encrypts message
- receiver decrypts message

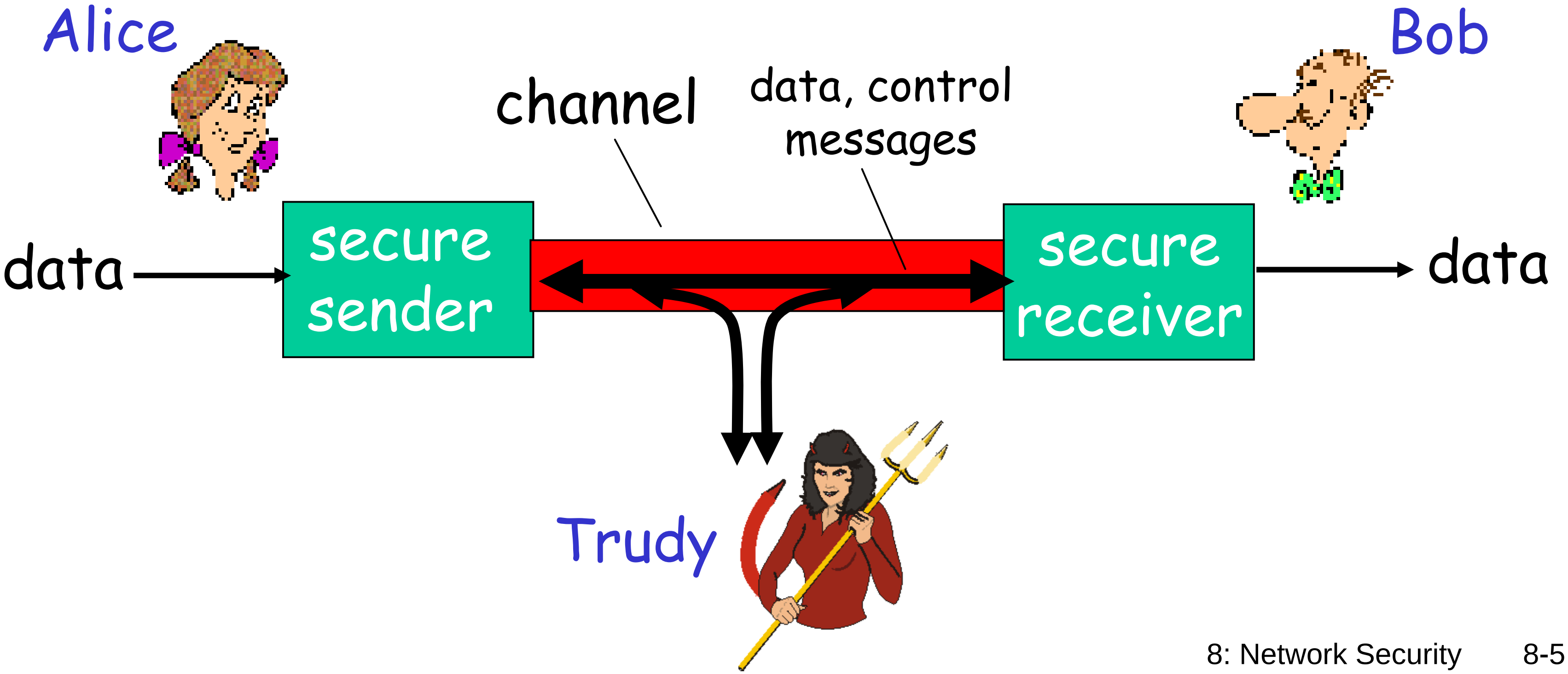
Authentication: sender, receiver want to confirm identity of each other

Message Integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

Access and Availability: services must be accessible and available to users

Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice (lovers!) want to communicate “securely”
- Trudy (intruder) may intercept, delete, add messages



Who might Bob, Alice be?

- … well, *real-life* Bobs and Alices!
- Web browser/server for electronic transactions (e.g., on-line purchases)
- on-line banking client/server
- DNS servers
- routers exchanging routing table updates
- other examples?

There are bad guys (and girls) out there!

Q: What can a “bad guy” do?

A: a lot!

- *eavesdrop:* intercept messages
- actively *insert* messages into connection
- *impersonation:* can fake (spoof) source address in packet (or any field in packet)
- *hijacking:* “take over” ongoing connection by removing sender or receiver, inserting himself in place
- *denial of service*: prevent service from being used by others (e.g., by overloading resources)

more on this later ……

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Authentication

8.4 Integrity

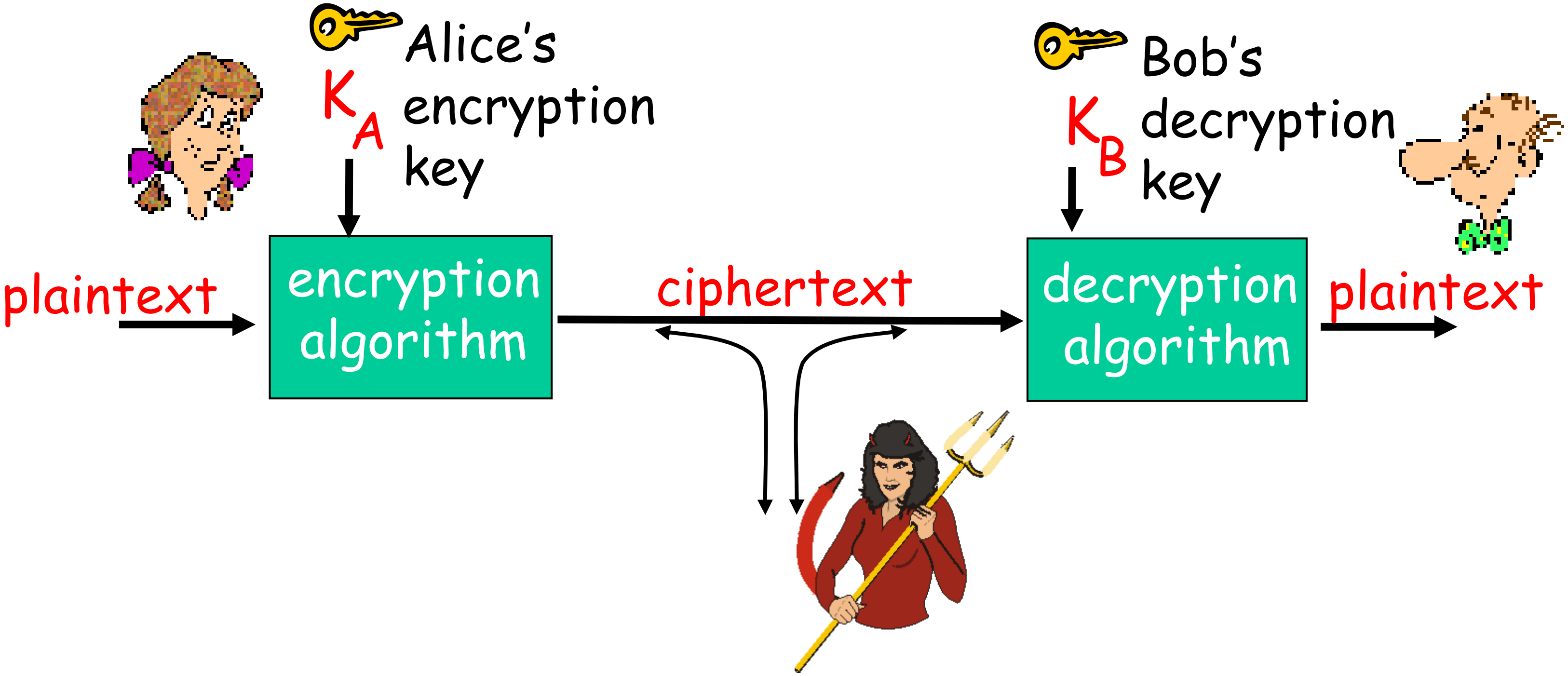
8.5 Key Distribution and certification

8.6 Access control: firewalls

8.7 Attacks and counter measures

8.8 Security in many layers

The language of cryptography



symmetric key crypto: sender, receiver keys *identical*

public-key crypto: encryption key *public*, decryption key *secret* (private)

Symmetric key cryptography

substitution cipher: substituting one thing for another

- monoalphabetic cipher: substitute one letter for another

```
 plaintext:  abcdefghijklmnopqrstuvwxyz

ciphertext:  mnbvcxzasdfghjklpoiuytrewq
```

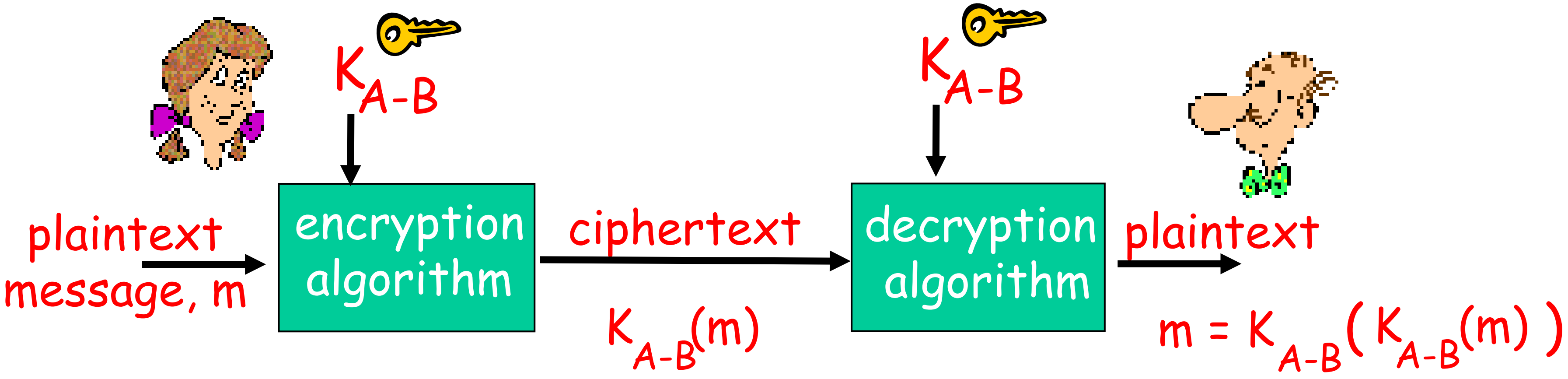
E.g.:

```
 Plaintext: bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc
```

Q: How hard to break this simple cipher?:

- □ brute force (how hard?)
- □ other?

Symmetric key cryptography



plaintext message, m

K_{A-B}

encryption algorithm

ciphertext

$K_{A-B}(m)$

K_{A-B}

decryption algorithm

plaintext

$m = K_{A-B}(K_{A-B}(m))$

symmetric key crypto: Bob and Alice share know same (symmetric) key: K_{A-B}

- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher
- Q: how do Bob and Alice agree on key value?

Symmetric key crypto: DES

DES: Data Encryption Standard

- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- How secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase ("Strong cryptography makes the world a safer place") decrypted (brute force) in 4 months
  - no known "backdoor" decryption approach
- making DES more secure:
  - use three keys sequentially (3-DES) on each datum
  - use cipher-block chaining

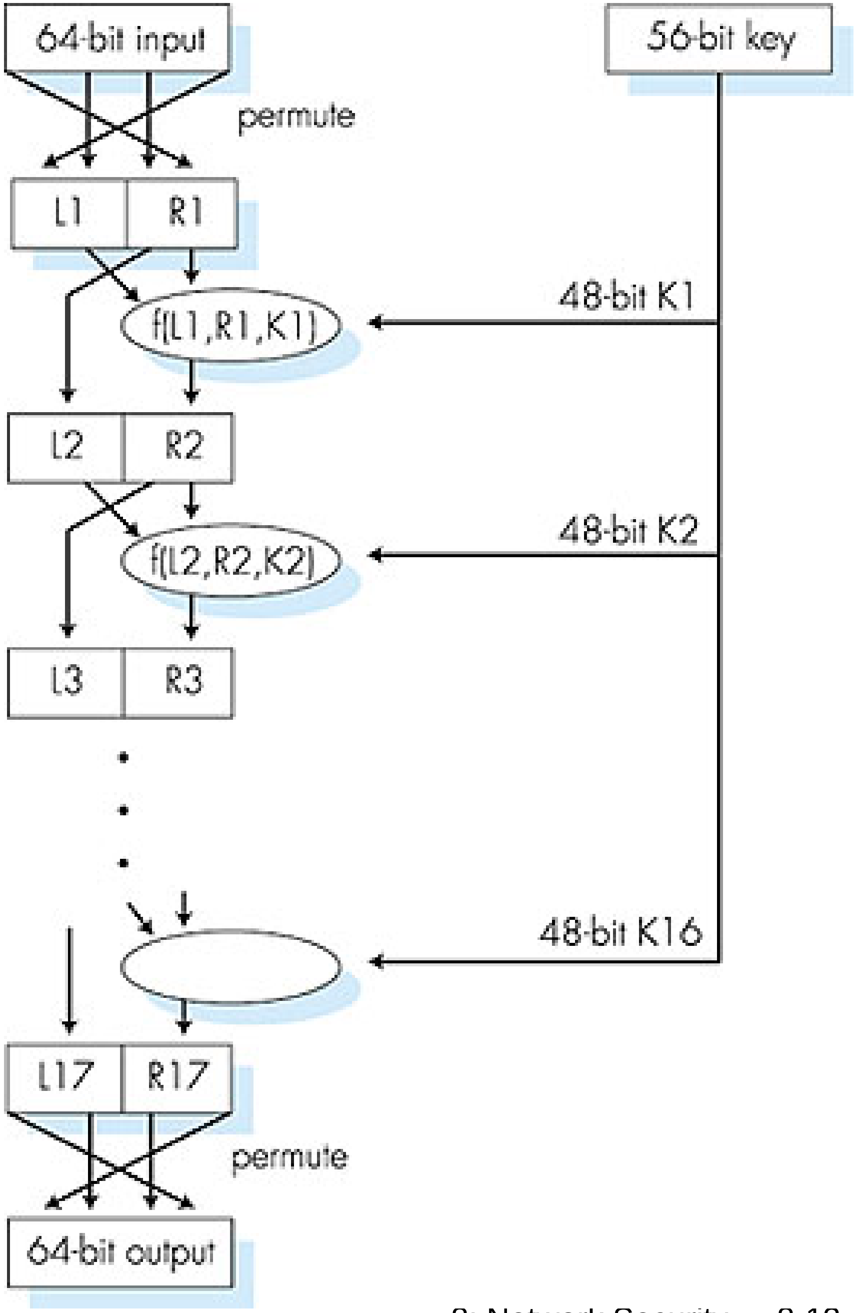
Symmetric key crypto: DES

DES operation

initial permutation

16 identical "rounds" of function application, each using different 48 bits of key

final permutation



AES: Advanced Encryption Standard

- new (Nov. 2001) symmetric-key NIST standard, replacing DES
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

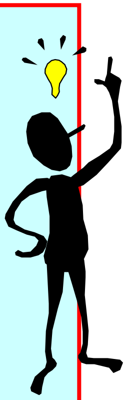
Public Key Cryptography

symmetric key crypto

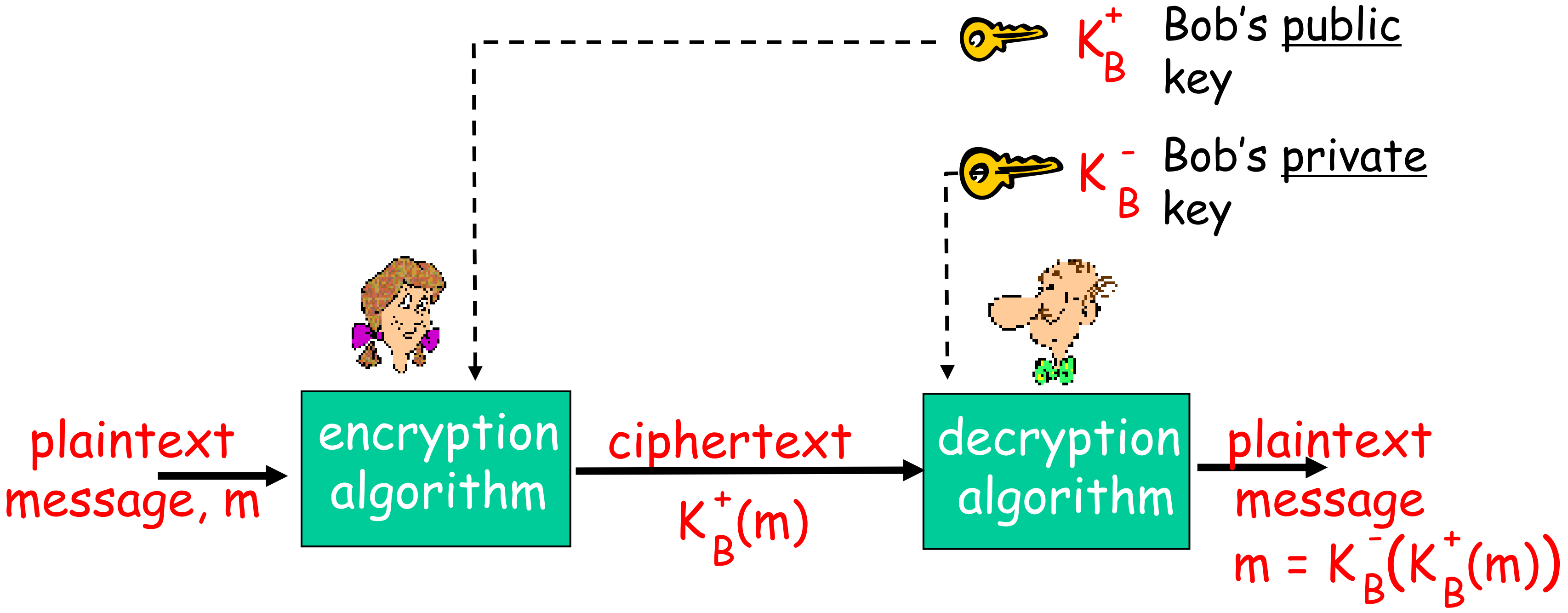
- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never "met")?

public key cryptography

- radically different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver



Public key cryptography



K_B^+ Bob's public key

K_B^- Bob's private key

plaintext message, m

encryption algorithm

ciphertext

$K_B^+(m)$

decryption algorithm

plaintext message

$m = K_B^-(K_B^+(m))$

Public key encryption algorithms

Requirements:

1. need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that

$$K_B^-(K_B^+(m)) = m$$

2. given public key K_B^+, it should be impossible to compute private key K_B^-

RSA: Rivest, Shamir, Adelson algorithm

RSA: Choosing keys

1. Choose two large prime numbers *p, q.* (e.g., 1024 bits each)

2. Compute $n = pq$, $z = (p-1)(q-1)$

3. Choose e (with $e<n$) that has no common factors with z. (e, z are "relatively prime").

4. Choose d such that $ed-1$ is exactly divisible by z. (in other words: $ed \bmod z = 1$).

5. *Public* key is $\underbrace{(n,e)}_{K_B^+}$. *Private* key is $\underbrace{(n,d)}_{K_B^-}$.

RSA: Encryption, decryption

0. Given (n,e) and (n,d) as computed above

1. To encrypt bit pattern, m, compute
   $c = m^e \bmod n$ (i.e., remainder when m^e is divided by n)

2. To decrypt received bit pattern, c, compute
   $m = c^d \bmod n$ (i.e., remainder when c^d is divided by n)

Magic happens!

$$m = (\underbrace{m^e \bmod n}_{c})^d \bmod n$$

RSA example:

Bob chooses *p=5, q=7*. Then *n=35, z=24*.
e=5 (so *e, z* relatively prime).
d=29 (so *ed-1* exactly divisible by z.

	letter	m	m^e	$c = m^e \bmod n$
encrypt:	l	12	1524832	17

	c	c^d	$m = c^d \bmod n$	letter
decrypt:	17	481968572106750915091411825223071697	12	l

RSA: Why is that $m = (m^e \bmod n)^d \bmod n$

Useful number theory result: If p,q prime and $n = pq$, then:

$$x^y \bmod n = x^{y \bmod (p-1)(q-1)} \bmod n$$

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$

$$= m^{ed \bmod (p-1)(q-1)} \bmod n$$

(using number theory result above)

$$= m^1 \bmod n$$

(since we chose ed to be divisible by $(p-1)(q-1)$ with remainder 1)

$$= m$$

RSA: another important property

The following property will be *very* useful later:

$$\underbrace{K_B^-(K_B^+(m))}_{\substack{\text{use public key}\\ \text{first, followed}\\ \text{by private key}}} = m = \underbrace{K_B^+(K_B^-(m))}_{\substack{\text{use private key}\\ \text{first, followed}\\ \text{by public key}}}$$

Result is the same!

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Authentication

8.4 Integrity

8.5 Key Distribution and certification

8.6 Access control: firewalls

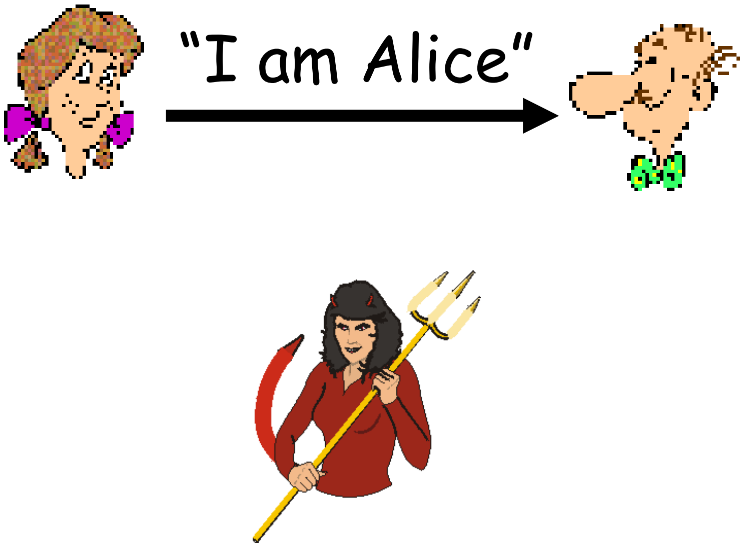
8.7 Attacks and counter measures

8.8 Security in many layers

Authentication

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap1.0: Alice says “I am Alice”

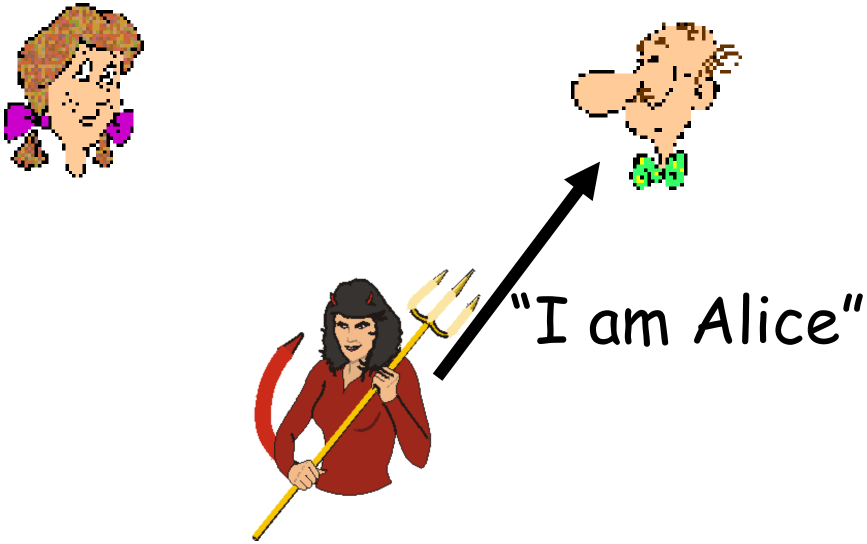


Failure scenario??

Authentication

Goal: Bob wants Alice to "prove" her identity to him

Protocol ap1.0: Alice says "I am Alice"

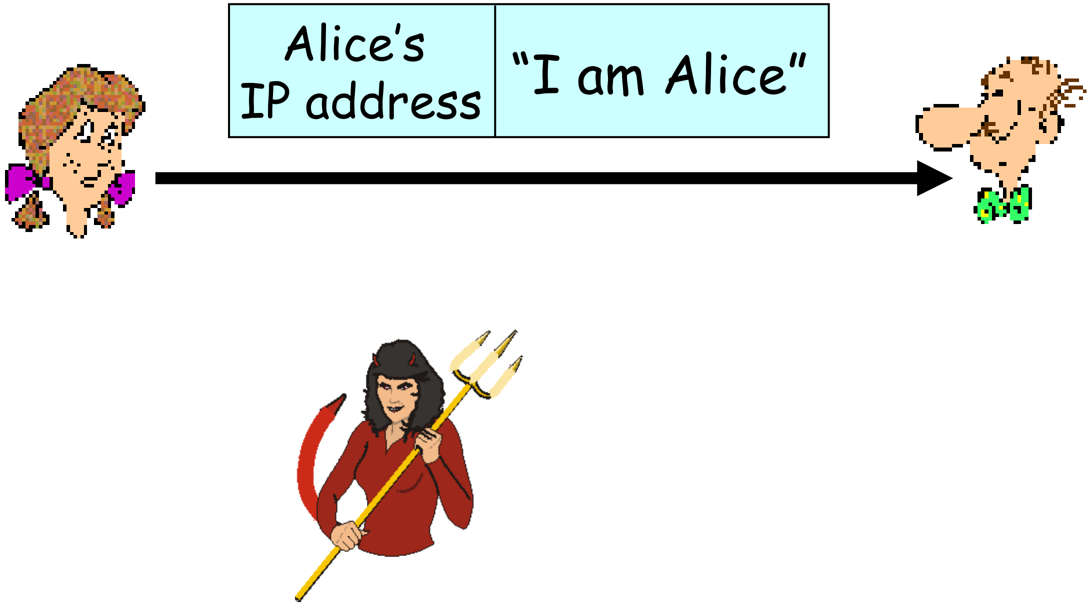


"I am Alice"

in a network,
Bob can not "see"
Alice, so Trudy simply
declares
herself to be Alice

Authentication: another try

Protocol ap2.0: Alice says “I am Alice” in an IP packet containing her source IP address



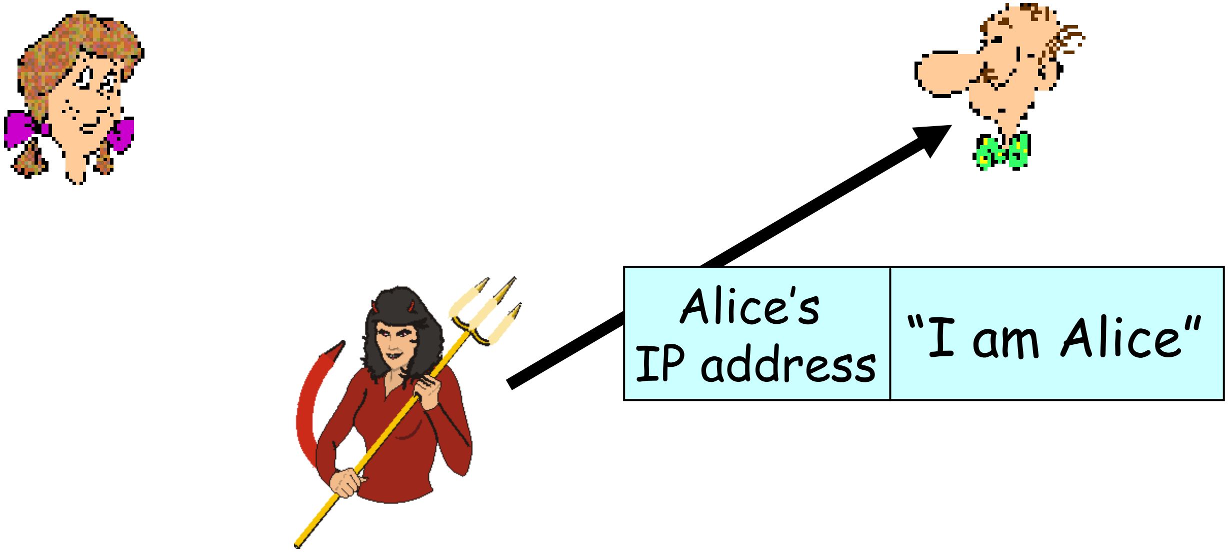
Alice’s IP address

“I am Alice”

Failure scenario??

Authentication: another try

Protocol ap2.0: Alice says “I am Alice” in an IP packet containing her source IP address



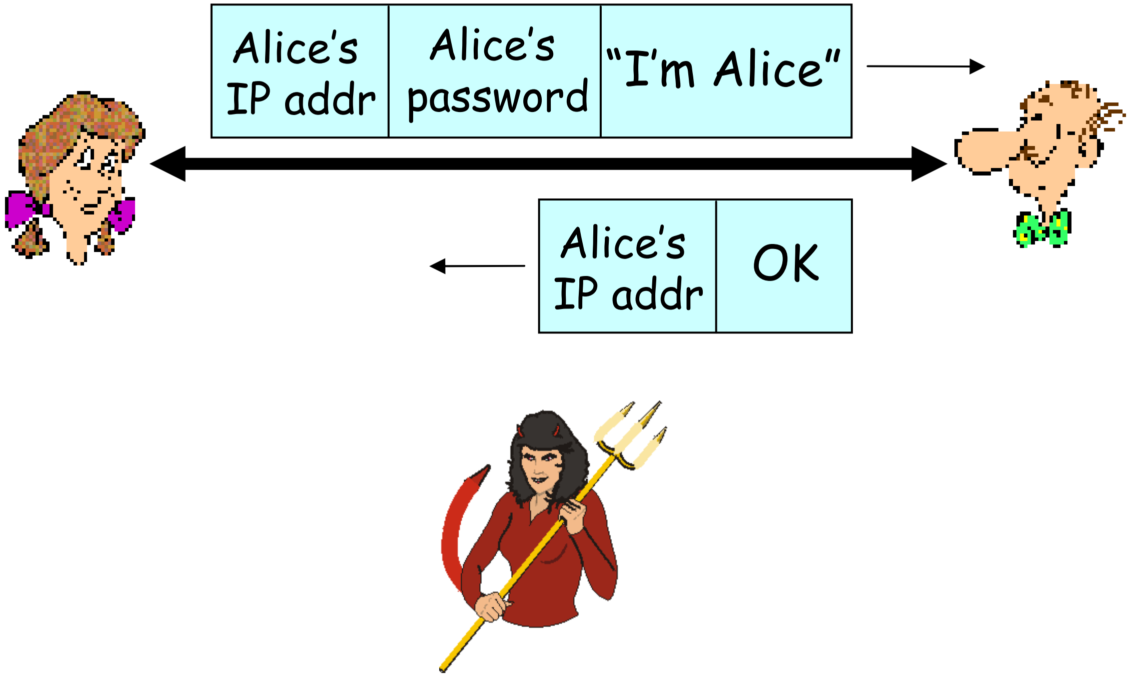
Alice’s IP address

“I am Alice”

Trudy can create a packet “spoofing” Alice’s address

Authentication: another try

Protocol ap3.0: Alice says “I am Alice” and sends her secret password to “prove” it.



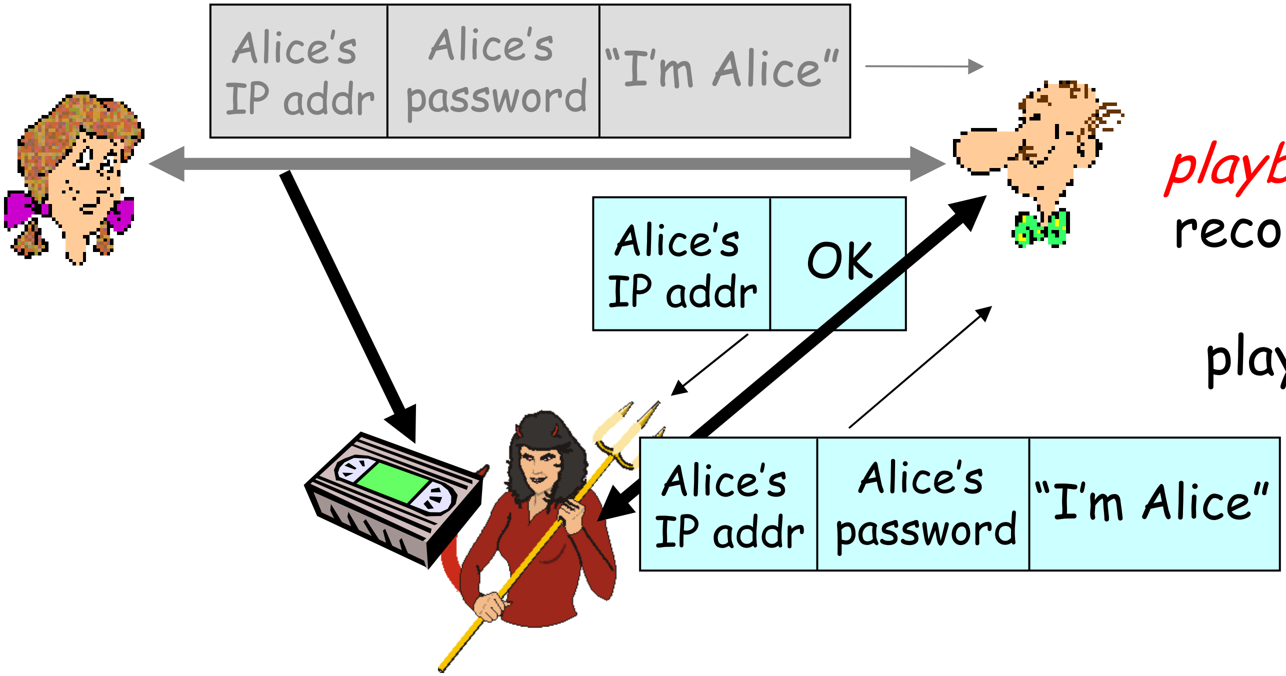
Alice’s IP addr	Alice’s password	“I’m Alice”

Alice’s IP addr	OK

Failure scenario??

Authentication: another try

Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



Alice's IP addr | Alice's password | "I'm Alice"

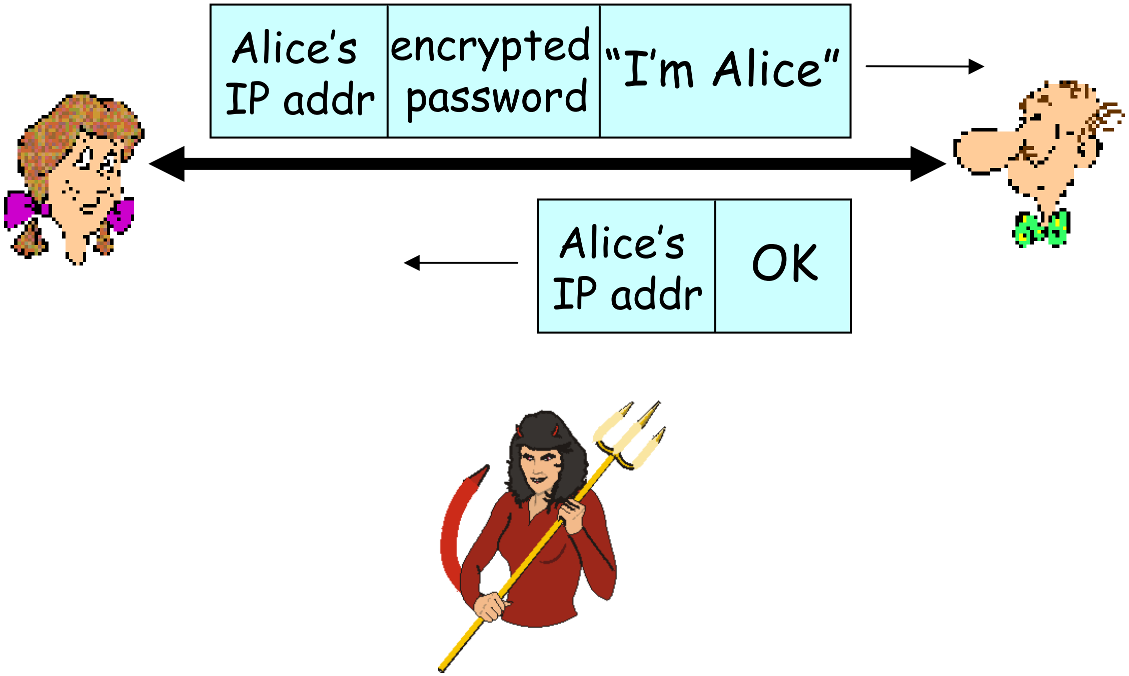
Alice's IP addr | OK

Alice's IP addr | Alice's password | "I'm Alice"

playback attack: Trudy records Alice's packet and later plays it back to Bob

Authentication: yet another try

Protocol ap3.1: Alice says “I am Alice” and sends her *encrypted* secret password to “prove” it.



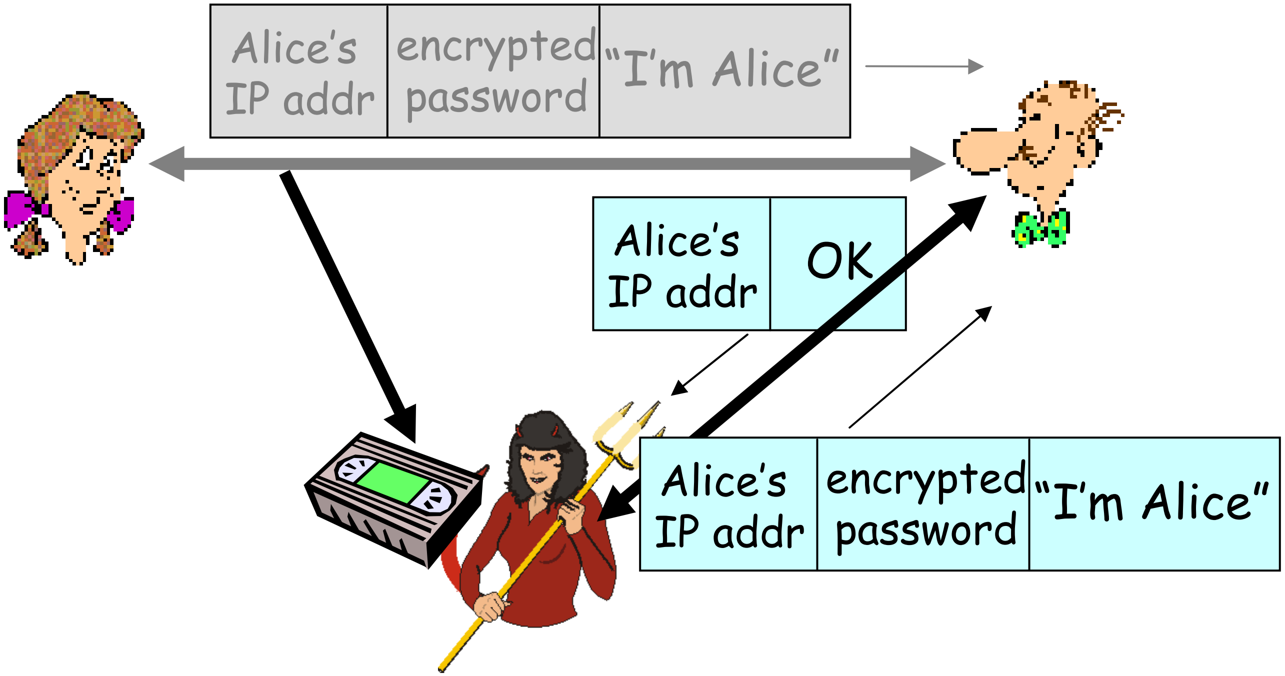
Alice’s IP addr | encrypted password | “I’m Alice”

Alice’s IP addr | OK

Failure scenario??

Authentication: another try

Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



Alice's IP addr	encrypted password	"I'm Alice"

Alice's IP addr	OK

Alice's IP addr	encrypted password	"I'm Alice"

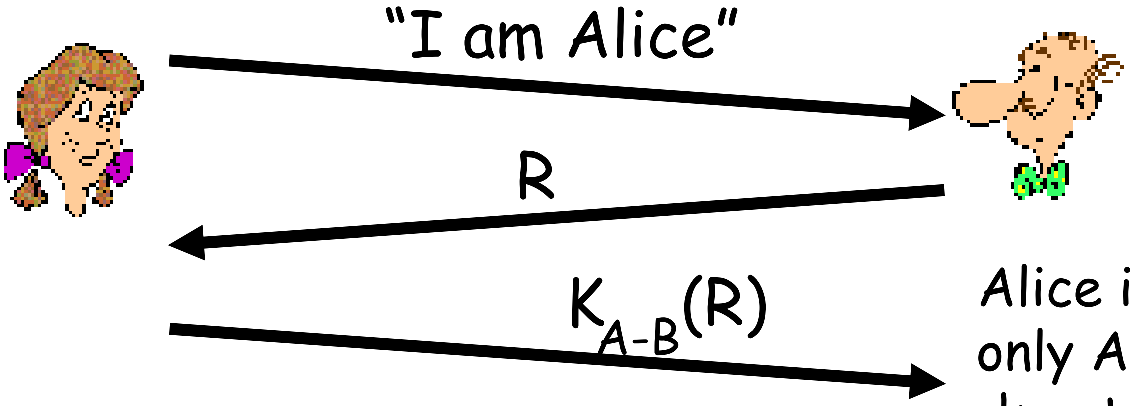
record and playback still works!

Authentication: yet another try

Goal: avoid playback attack

Nonce: number (R) used only *once –in-a-lifetime*

ap4.0: to prove Alice "live", Bob sends Alice nonce, R. Alice must return R, encrypted with shared secret key



"I am Alice"

R

$K_{A-B}(R)$

Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

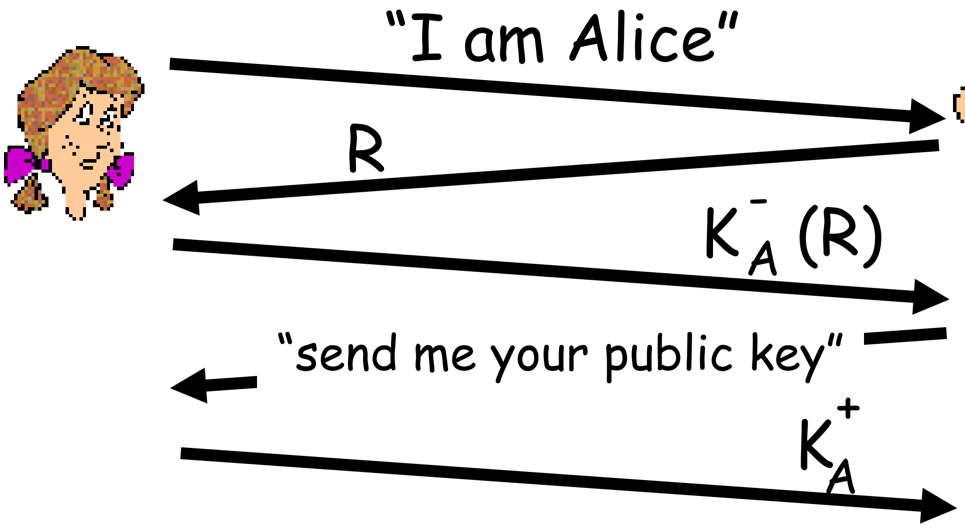
Failures, drawbacks?

Authentication: ap5.0

ap4.0 requires shared symmetric key

- ❒ can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography



"I am Alice"

R

$K_A^-(R)$

"send me your public key"

K_A^+

Bob computes

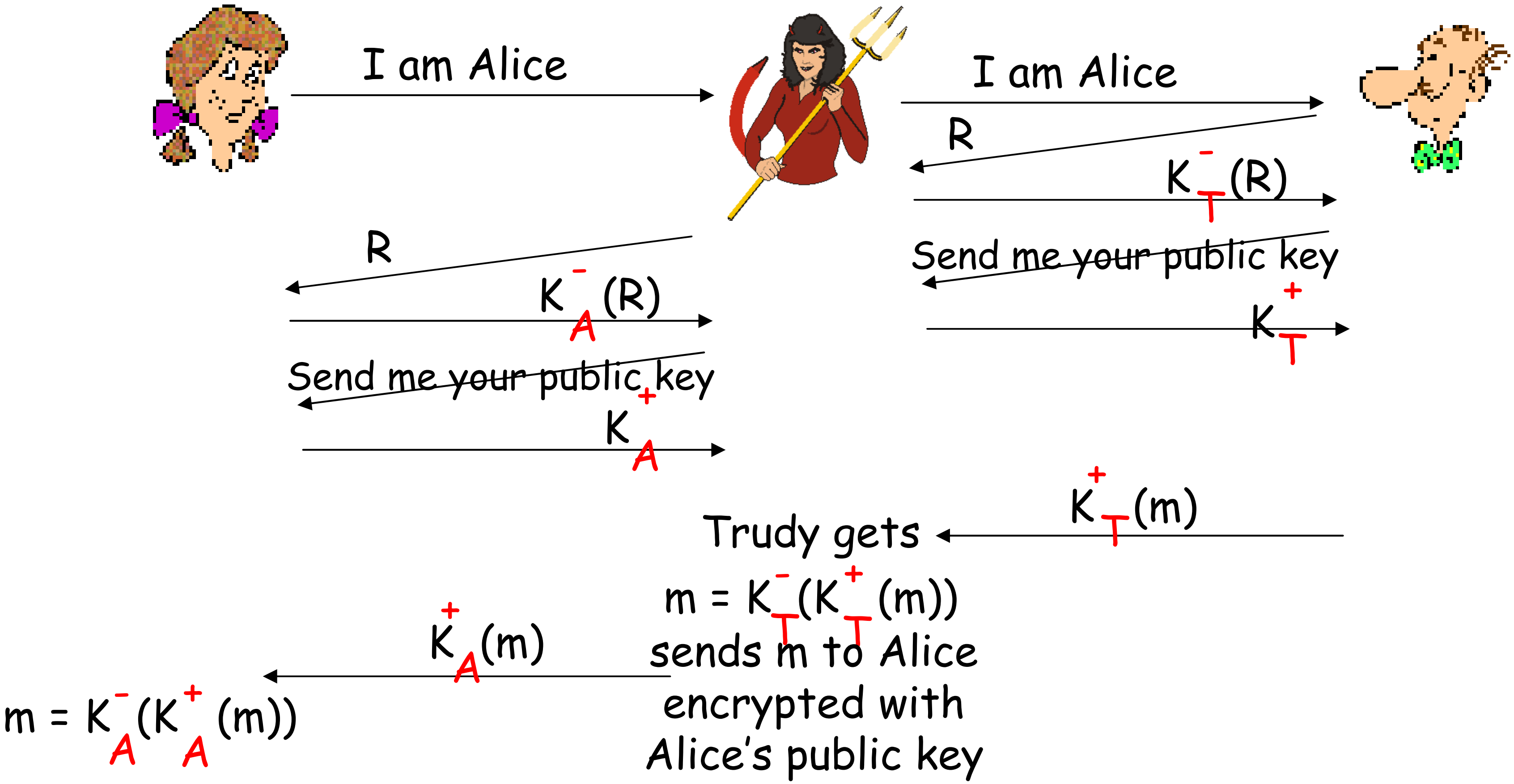
$$K_A^+(K_A^-(R)) = R$$

and knows only Alice could have the private key, that encrypted R such that

$$K_A^+(K_A^-(R)) = R$$

ap5.0: security hole

Man (woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



I am Alice

I am Alice

R

$K_T^-(R)$

Send me your public key

K_T^+

R

$K_A^-(R)$

Send me your public key

K_A^+

$K_T^+(m)$

Trudy gets

$m = K_T^-(K_T^+(m))$

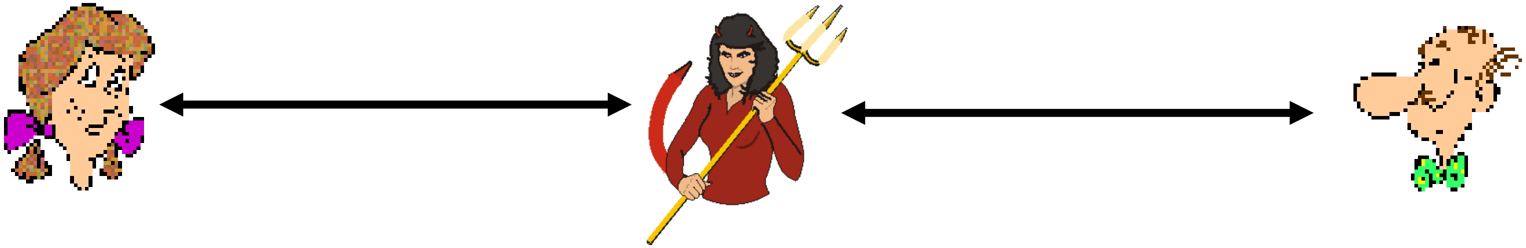
sends m to Alice encrypted with Alice's public key

$K_A^+(m)$

$m = K_A^-(K_A^+(m))$

ap5.0: security hole

Man (woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



Difficult to detect:

- Bob receives everything that Alice sends, and vice versa. (e.g., so Bob, Alice can meet one week later and recall conversation)
- problem is that Trudy receives all messages as well!

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Authentication

8.4 Message integrity

8.5 Key Distribution and certification

8.6 Access control: firewalls

8.7 Attacks and counter measures

8.8 Security in many layers

Digital Signatures

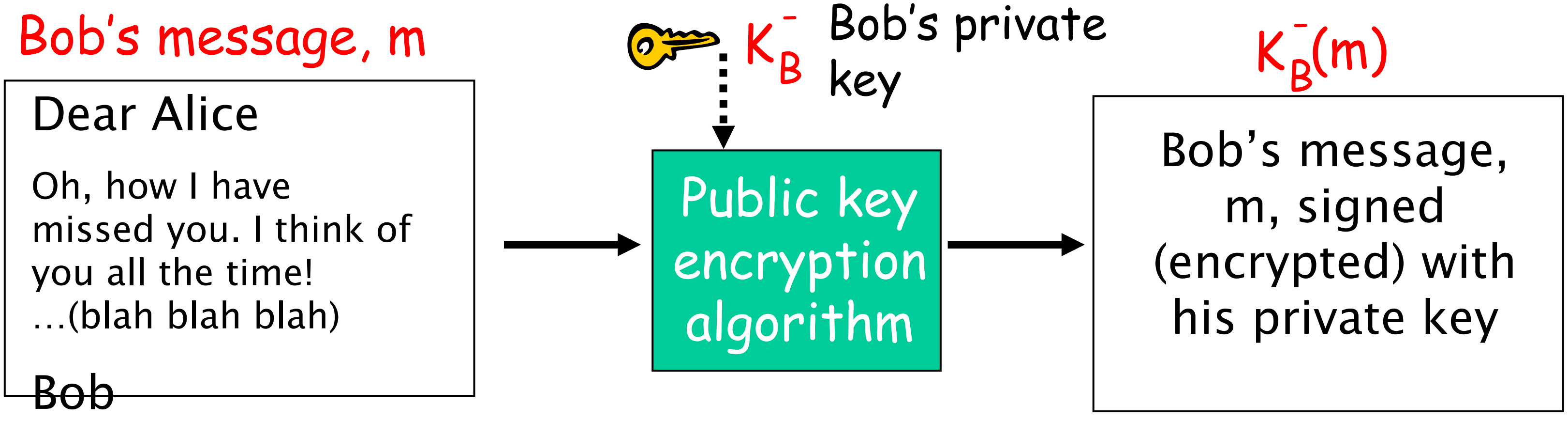
Cryptographic technique analogous to hand-written signatures.

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- verifiable, nonforgeable: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

Digital Signatures

Simple digital signature for message m:

- Bob signs m by encrypting with his private key K_B^-, creating "signed" message, $K_B^-(m)$



Bob's message, m

Dear Alice

Oh, how I have missed you. I think of you all the time! …(blah blah blah)

Bob

K_B^- Bob's private key

Public key encryption algorithm

$K_B^-(m)$

Bob's message, m, signed (encrypted) with his private key

Digital Signatures (more)

- Suppose Alice receives msg m, digital signature $K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key K_B^+ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$.
- If $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key.

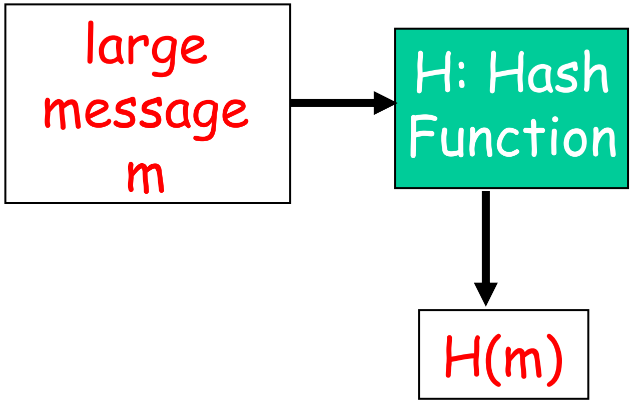
Alice thus verifies that:

- ✓ Bob signed m.
- ✓ No one else signed m.
- ✓ Bob signed m and not m'.

Non-repudiation:

- ✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m.

Message Digests



Computationally expensive to public-key-encrypt long messages

Goal: fixed-length, easy-to-compute digital "fingerprint"

- apply hash function H to *m*, get fixed size message digest, *H(m).*

Hash function properties:

- many-to-1
- produces fixed-size msg digest (fingerprint)
- given message digest x, computationally infeasible to find m such that x = H(m)

Internet checksum: poor crypto hash function

Internet checksum has some properties of hash function:

- ✓ produces fixed length digest (16-bit sum) of message
- ✓ is many-to-one

But given message with given hash value, it is easy to find another message with same hash value:

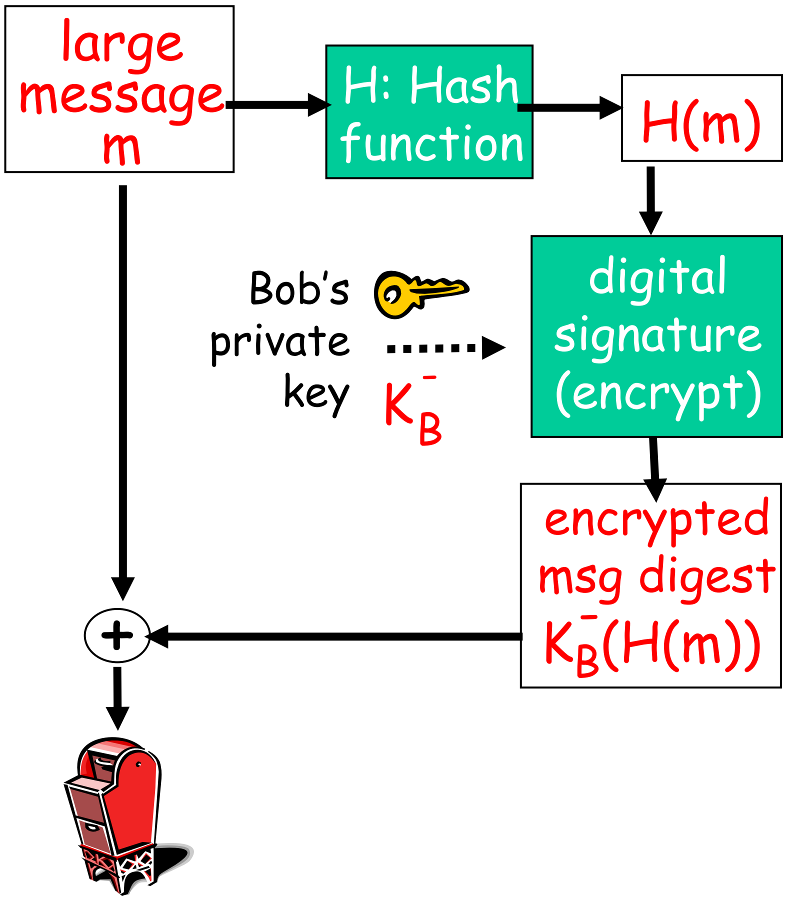
message	ASCII format
I O U 1	49 4F 55 31
0 0 . 9	30 30 2E 39
9 B O B	39 42 D2 42
	B2 C1 D2 AC

message	ASCII format
I O U 9	49 4F 55 39
0 0 . 1	30 30 2E 31
9 B O B	39 42 D2 42
	B2 C1 D2 AC

different messages but identical checksums!

Digital signature = signed message digest

Bob sends digitally signed message:



large message m

H: Hash function

H(m)

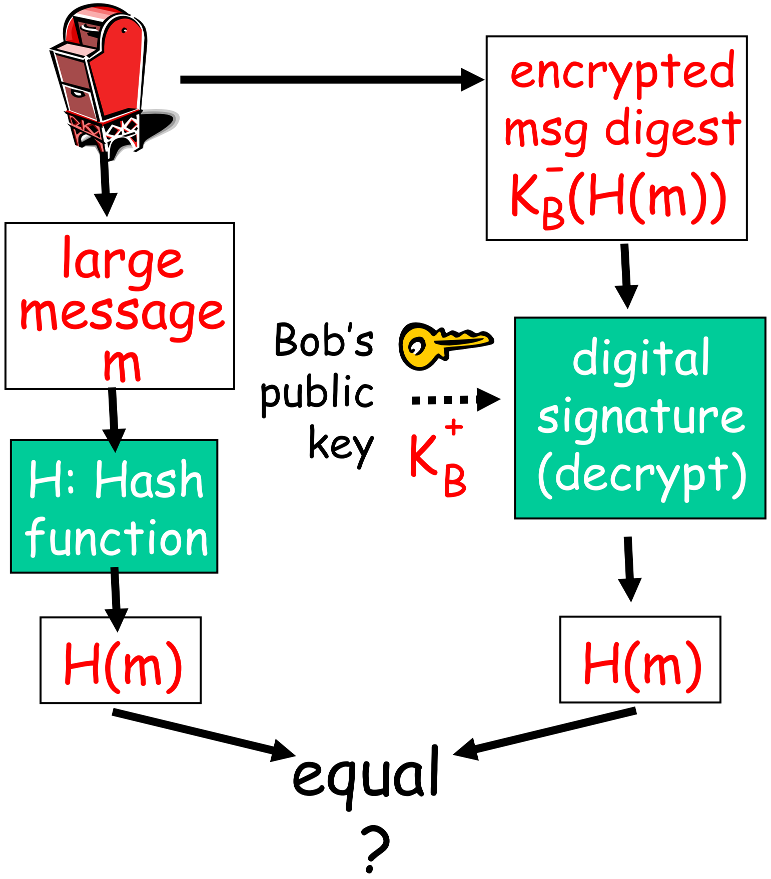
Bob's private key K_B^-

digital signature (encrypt)

encrypted msg digest $K_B^-(H(m))$

+

Alice verifies signature and integrity of digitally signed message:



encrypted msg digest $K_B^-(H(m))$

large message m

H: Hash function

Bob's public key K_B^+

digital signature (decrypt)

H(m)

H(m)

equal ?

Hash Function Algorithms

- MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string x, appears difficult to construct msg m whose MD5 hash is equal to x.
- SHA-1 is also used.
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Authentication

8.4 Integrity

8.5 Key distribution and certification

8.6 Access control: firewalls

8.7 Attacks and counter measures

8.8 Security in many layers

Trusted Intermediaries

Symmetric key problem:

- How do two entities establish shared secret key over network?

Solution:

- trusted key distribution center (KDC) acting as intermediary between entities

Public key problem:

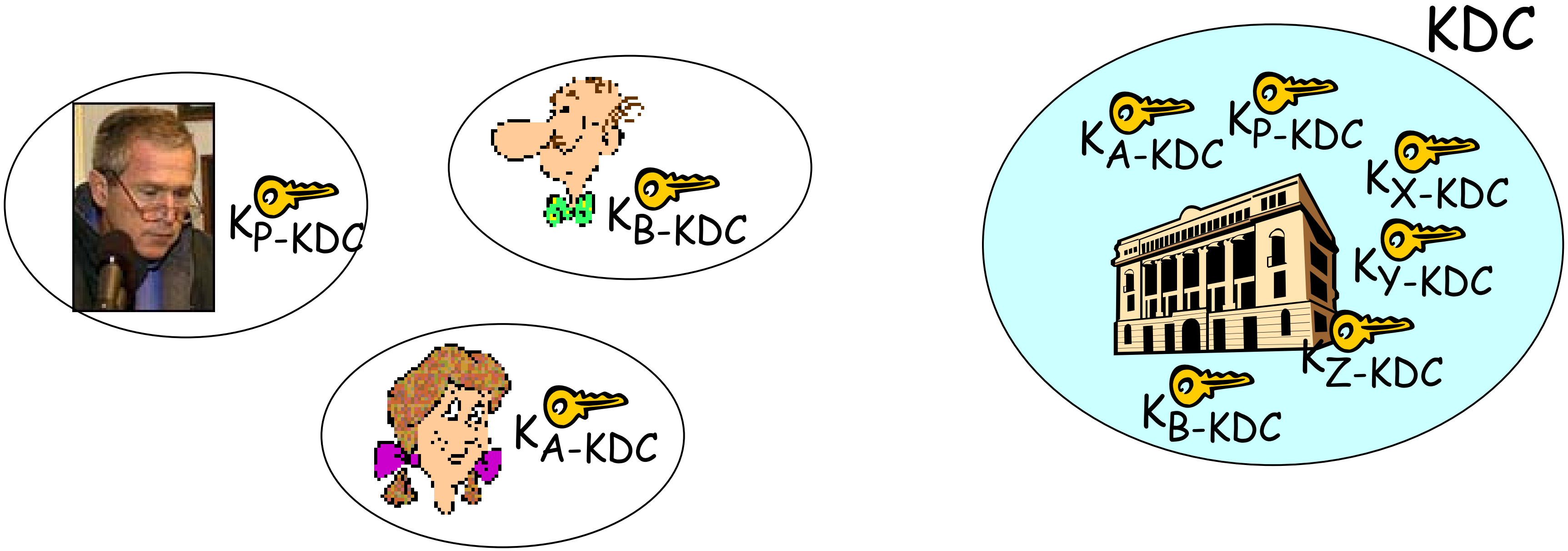
- When Alice obtains Bob's public key (from web site, e-mail, diskette), how does she know it is Bob's public key, not Trudy's?

Solution:

- trusted certification authority (CA)

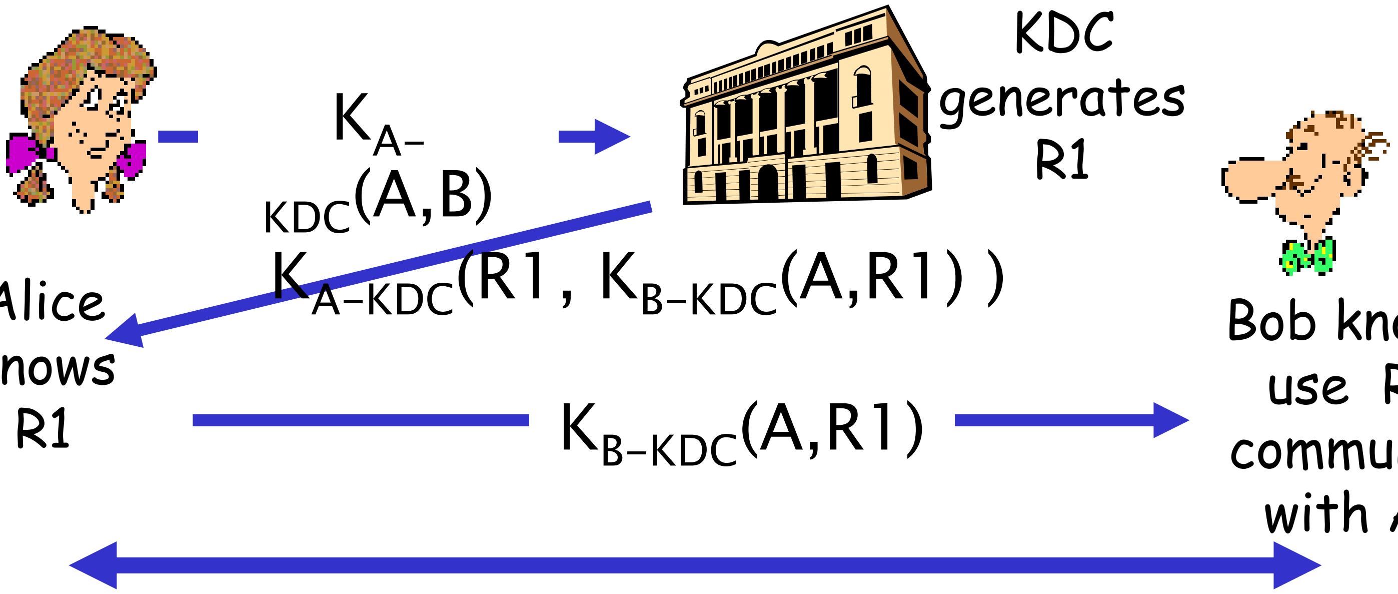
Key Distribution Center (KDC)

- Alice, Bob need shared symmetric key.
- KDC: server shares different secret key with *each* registered user (many users)
- Alice, Bob know own symmetric keys, $K_{A\text{-}KDC}$ $K_{B\text{-}KDC}$, for communicating with KDC.



Key Distribution Center (KDC)

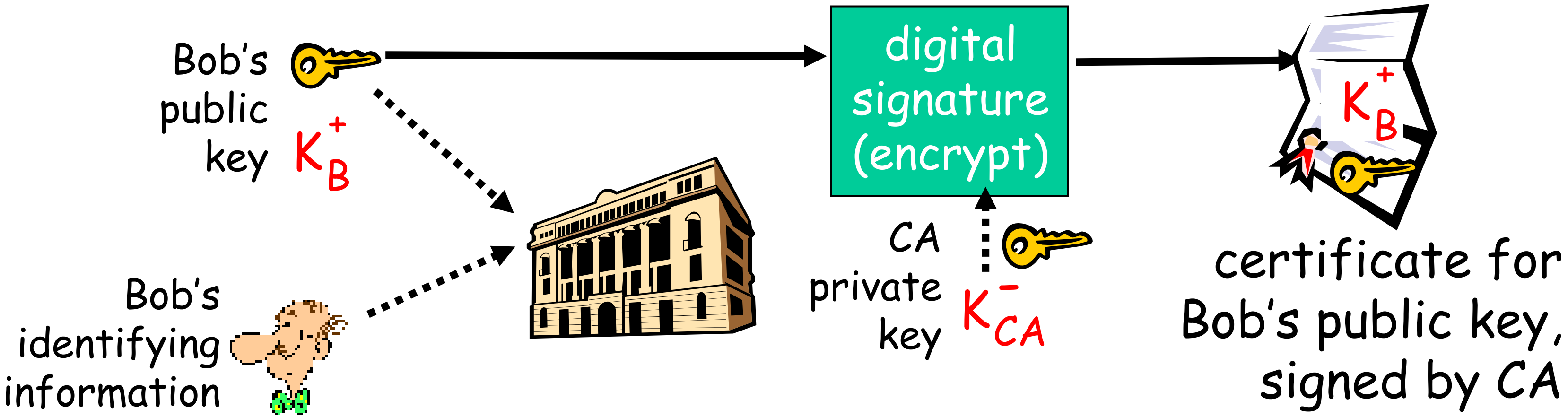
Q: How does KDC allow Bob, Alice to determine shared symmetric secret key to communicate with each other?


$K_{A-KDC}(A,B)$
KDC generates R1
$K_{A-KDC}(R1, K_{B-KDC}(A,R1))$
Alice knows R1
$K_{B-KDC}(A,R1)$
Bob knows to use R1 to communicate with Alice

Alice and Bob communicate: using R1 as *session key* for shared symmetric encryption

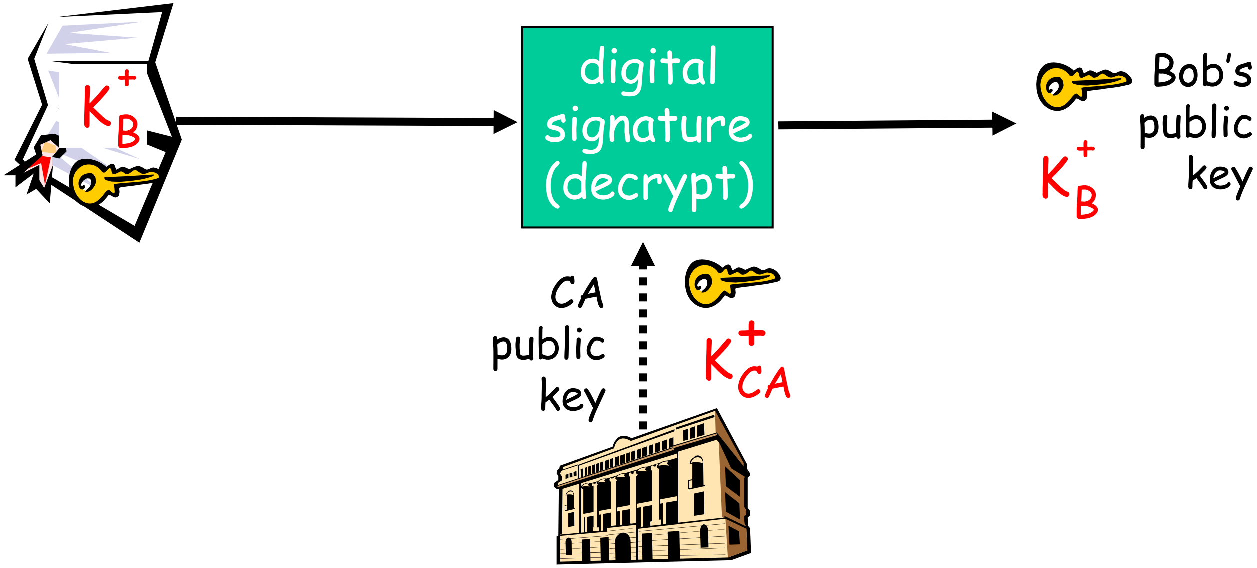
Certification Authorities

- Certification authority (CA): binds public key to particular entity, E.
- E (person, router) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key digitally signed by CA – CA says "this is E's public key"


Bob's public key
K_B^+
Bob's identifying information
digital signature (encrypt)
CA private key
K_{CA}^-
K_B^+
certificate for Bob's public key, signed by CA

Certification Authorities

- When Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere).
  - apply CA's public key to Bob's certificate, get Bob's public key



Information security future

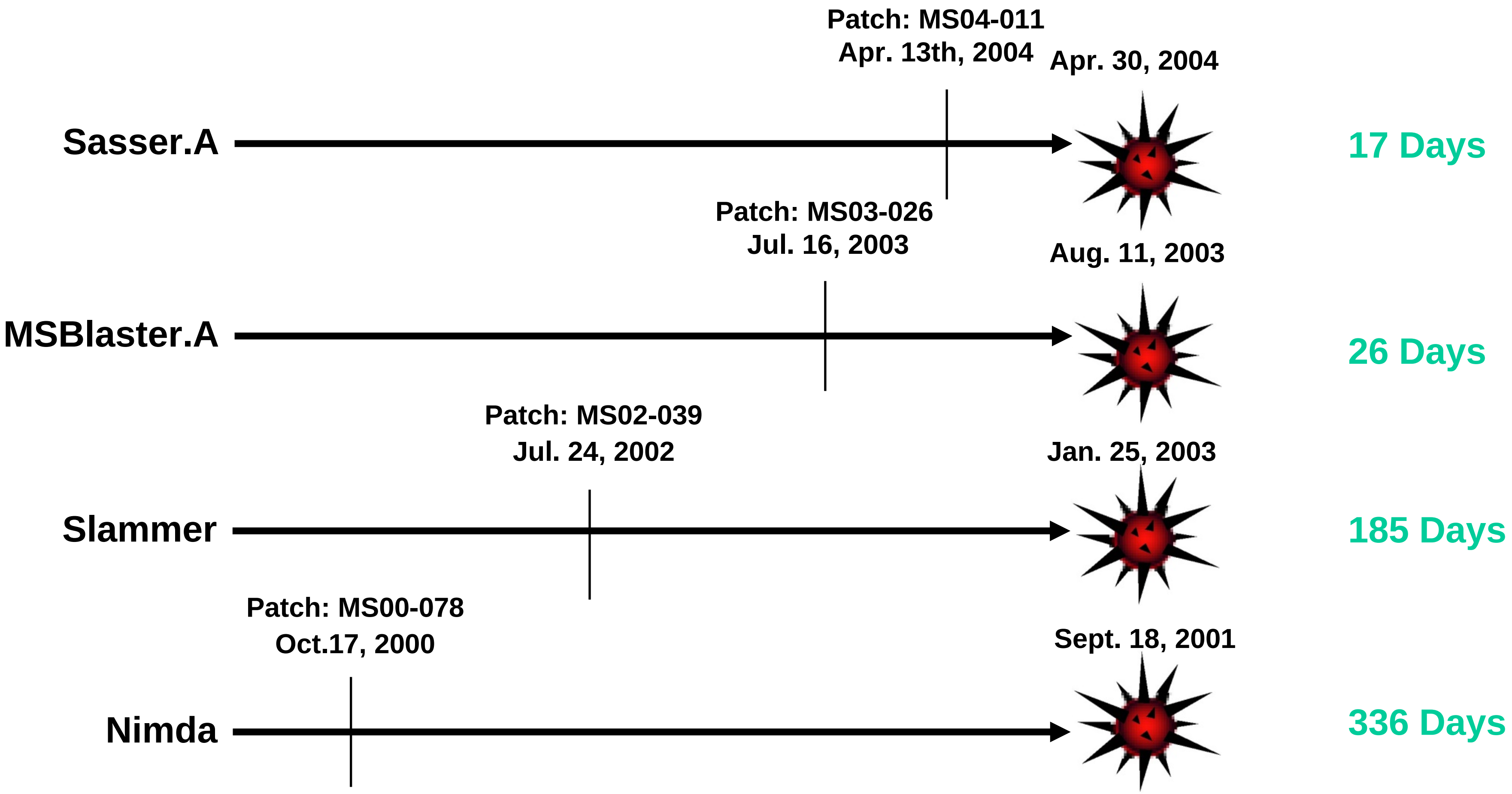
- World War II: Enigma

History

Getting Enigma

- Modern information war
  - Cryptographic issues
  - Non-cryptographic issues

Window of Time from Patch Availability to Outbreak Getting Shorter



Sasser.A

Patch: MS04-011
Apr. 13th, 2004

Apr. 30, 2004

17 Days

MSBlaster.A

Patch: MS03-026
Jul. 16, 2003

Aug. 11, 2003

26 Days

Slammer

Patch: MS02-039
Jul. 24, 2002

Jan. 25, 2003

185 Days

Nimda

Patch: MS00-078
Oct.17, 2000

Sept. 18, 2001

336 Days

Customers need an innovative systems approach to preventing and containing infections

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Authentication

8.4 Integrity

8.5 Key Distribution and certification

8.6 Access control: firewalls

8.7 Attacks and counter measures

8.8 Security in many layers

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Authentication

8.4 Integrity

8.5 Key Distribution and certification

8.6 Access control: firewalls

8.7 Attacks and counter measures

8.8 Security in many layers

Internet security threats

Mapping:

- before attacking: “case the joint” – find out what services are implemented on network
- Use `ping` to determine what hosts have addresses on network
- Port-scanning: try to establish TCP connection to each port in sequence (see what happens)
- nmap (http://www.insecure.org/nmap/) mapper: “network exploration and security auditing”

Countermeasures?

Internet security threats

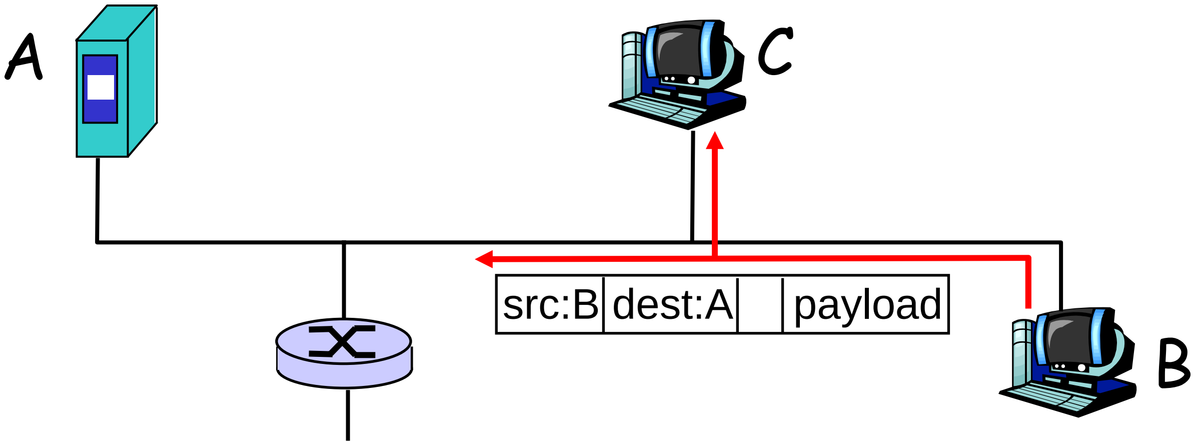
Mapping: countermeasures

- record traffic entering network
- look for suspicious activity (IP addresses, pots being scanned sequentially)

Internet security threats

Packet sniffing:

- broadcast media
- promiscuous NIC reads all packets passing by
- can read all unencrypted data (e.g. passwords)
- e.g.: C sniffs B's packets

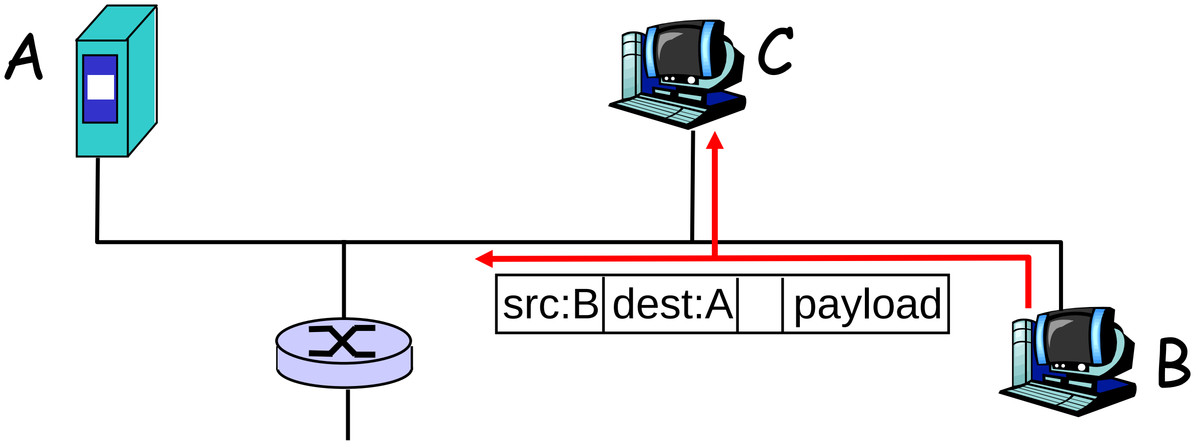


Countermeasures?

Internet security threats

Packet sniffing: countermeasures

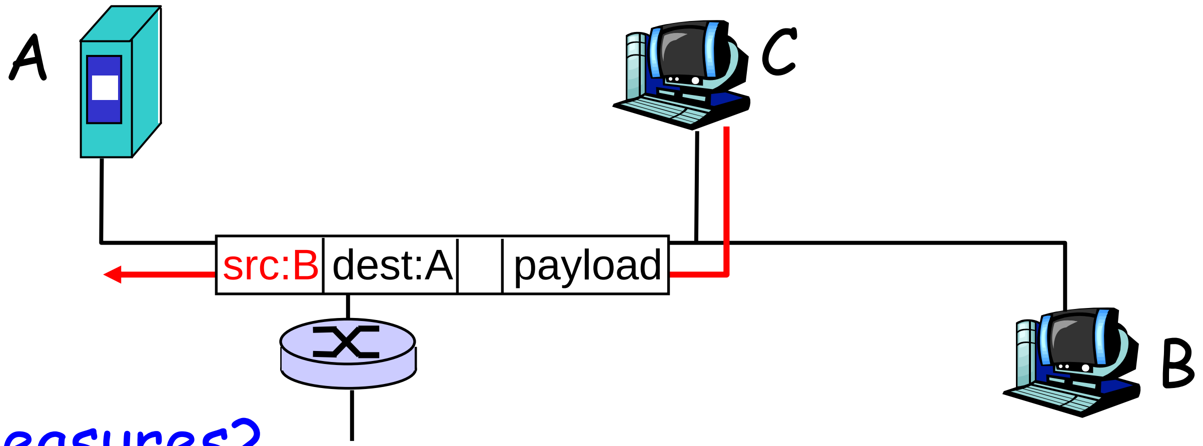
- all hosts in organization run software that checks periodically if host interface in promiscuous mode.
- one host per segment of broadcast media (switched Ethernet at hub)



Internet security threats

IP Spoofing:

- can generate "raw" IP packets directly from application, putting any value into IP source address field
- receiver can't tell if source is spoofed
- e.g.: C pretends to be B

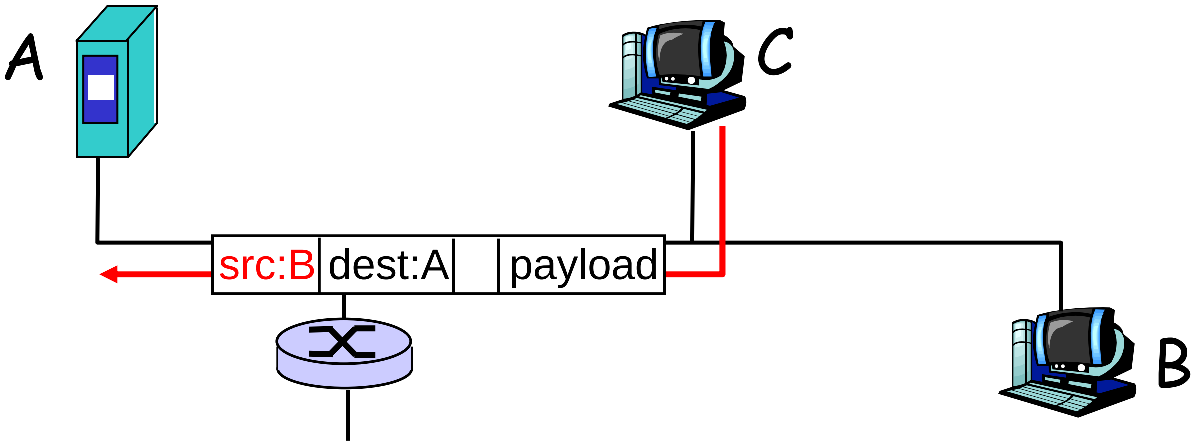


Countermeasures?

Internet security threats

IP Spoofing: ingress filtering

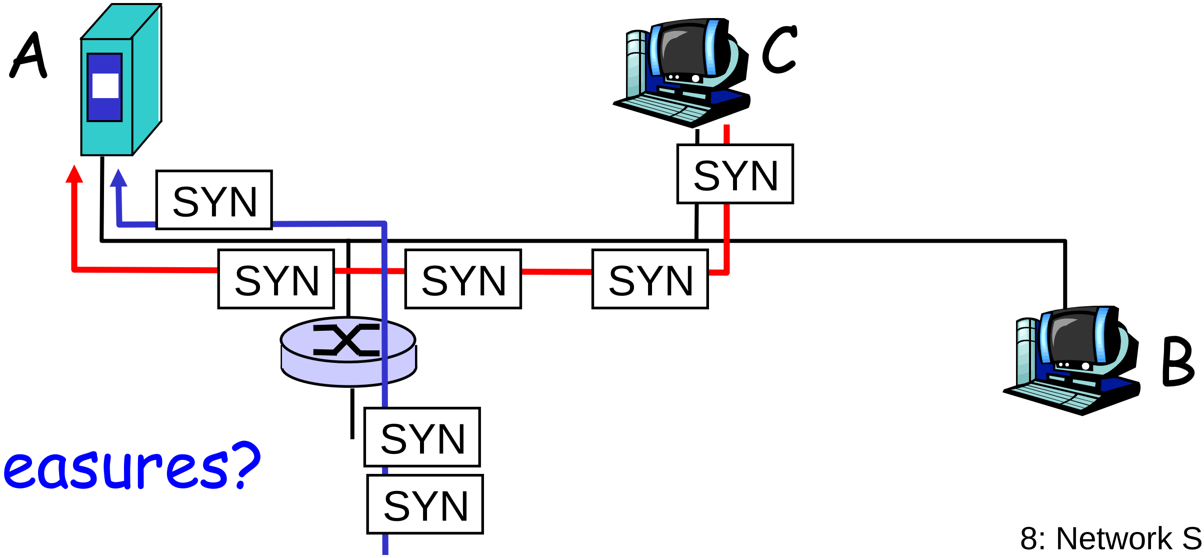
- routers should not forward outgoing packets with invalid source addresses (e.g., datagram source address not in router's network)
- great, but ingress filtering can not be mandated for all networks



Internet security threats

Denial of service (DOS):

- flood of maliciously generated packets "swamp" receiver
- Distributed DOS (DDOS): multiple coordinated sources swamp receiver
- e.g., *C* and remote host SYN-attack *A*

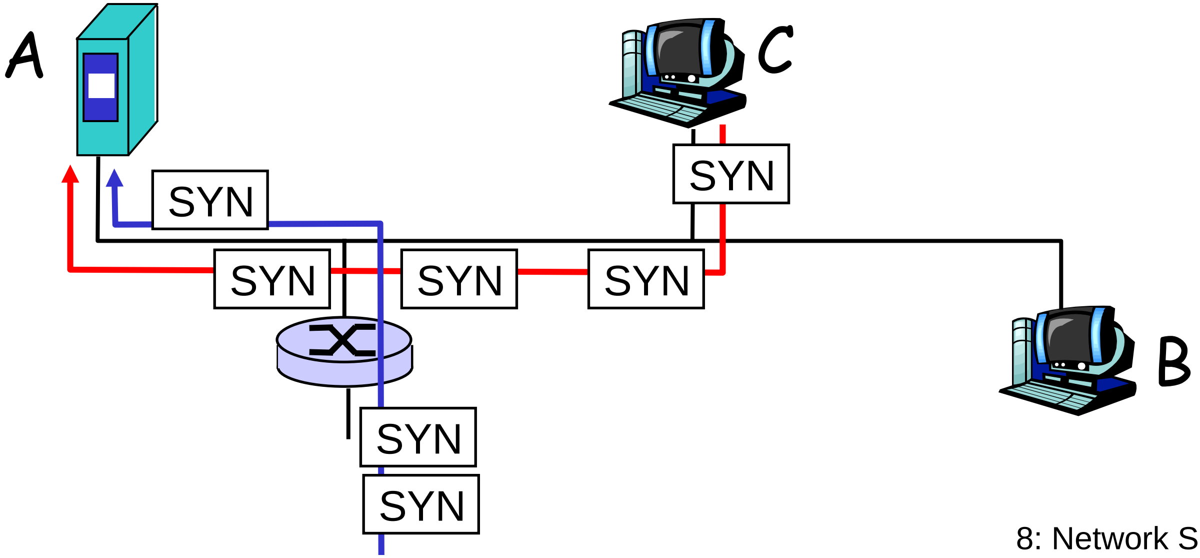


Countermeasures?

Internet security threats

Denial of service (DOS): countermeasures

- filter out flooded packets (e.g., SYN) before reaching host: throw out good with bad
- traceback to source of floods (most likely an innocent, compromised machine)



Chapter 8 roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Authentication
8.4 Integrity
8.5 Key Distribution and certification
8.6 Access control: firewalls
8.7 Attacks and counter measures
8.8 Security in many layers
- 8.8.1. Secure email
- 8.8.2. Secure sockets
- 8.8.3. IPsec
- 8.8.4. Security in 802.11

Pretty good privacy (PGP)

- Internet e-mail encryption scheme, de-facto standard.
- uses symmetric key cryptography, public key cryptography, hash function, and digital signature as described.
- provides secrecy, sender authentication, integrity.
- inventor, Phil Zimmerman, was target of 3-year federal investigation.

A PGP signed message:

```
---BEGIN PGP SIGNED MESSAGE---
Hash: SHA1

Bob:My husband is out of town
   tonight.Passionately yours,
   Alice

---BEGIN PGP SIGNATURE---
Version: PGP 5.0
Charset: noconv
yhHJRHhGJGhgg/12EpJ+lo8gE4vB3mqJ
   hFEvZP9t6n7G6m5Gw2
---END PGP SIGNATURE---
```

Secure sockets layer (SSL)

- transport layer security to any TCP-based app using SSL services.
- used between Web browsers, servers for e-commerce (shttp).
- security services:
  - server authentication
  - data encryption
  - client authentication (optional)
- server authentication:
  - SSL-enabled browser includes public keys for trusted CAs.
  - Browser requests server certificate, issued by trusted CA.
  - Browser uses CA's public key to extract server's public key from certificate.
- check your browser's security menu to see its trusted CAs.

SSL (continued)

Encrypted SSL session:

- Browser generates *symmetric session key*, encrypts it with server's public key, sends encrypted key to server.
- Using private key, server decrypts session key.
- Browser, server know session key
  - All data sent into TCP socket (by client or server) encrypted with session key.

- SSL: basis of IETF Transport Layer Security (TLS).
- SSL can be used for non-Web applications, e.g., IMAP.
- Client authentication can be done with client certificates.

IPsec: Network Layer Security

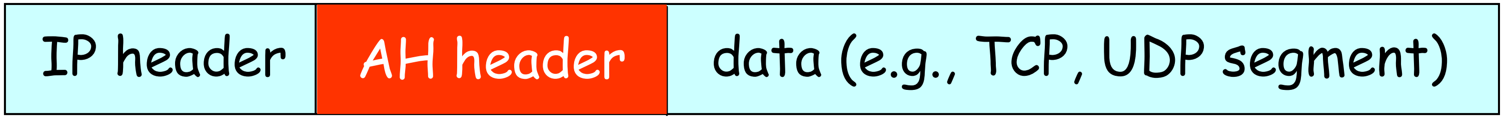
- Network-layer secrecy:
  - sending host encrypts the data in IP datagram
  - TCP and UDP segments; ICMP and SNMP messages.
- Network-layer authentication
  - destination host can authenticate source IP address
- Two principle protocols:
  - authentication header (AH) protocol
  - encapsulation security payload (ESP) protocol
- For both AH and ESP, source, destination handshake:
  - create network-layer logical channel called a security association (SA)
- Each SA unidirectional.
- Uniquely determined by:
  - security protocol (AH or ESP)
  - source IP address
  - 32-bit connection ID

Authentication Header (AH) Protocol

- provides source authentication, data integrity, no confidentiality
- AH header inserted between IP header, data field.
- protocol field: 51
- intermediate routers process datagrams as usual

AH header includes:

- connection identifier
- authentication data: source- signed message digest calculated over original IP datagram.
- next header field: specifies type of data (e.g., TCP, UDP, ICMP)

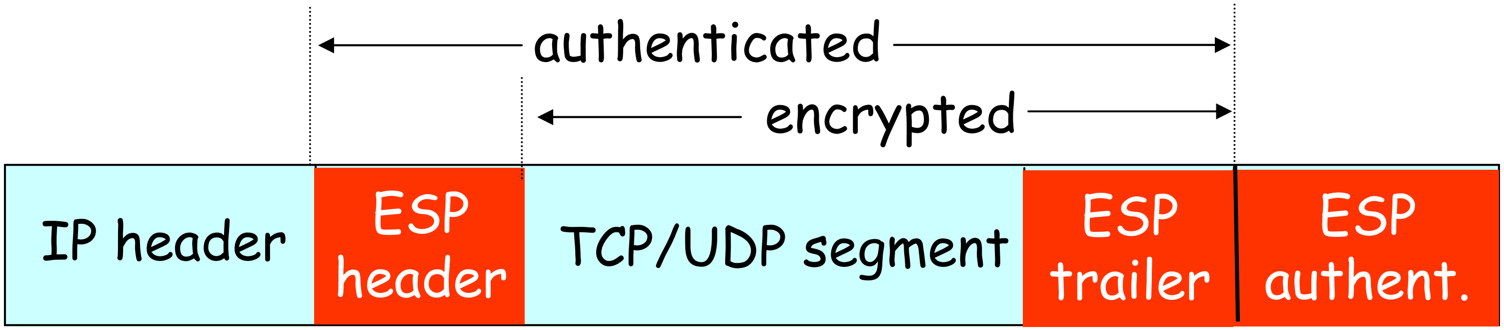

IP header

AH header

data (e.g., TCP, UDP segment)

ESP Protocol

- provides secrecy, host authentication, data integrity.
- data, ESP trailer encrypted.
- next header field is in ESP trailer.
- ESP authentication field is similar to AH authentication field.
- Protocol = 50.



IEEE 802.11 security

- *War-driving:* drive around Bay area, see what 802.11 networks available?
  - More than 9000 accessible from public roadways
  - 85% use no encryption/authentication
  - packet-sniffing and various attacks easy!
- Securing 802.11
  - encryption, authentication
  - first attempt at 802.11 security: Wired Equivalent Privacy (WEP): a failure
  - current attempt: 802.11i

Wired Equivalent Privacy (WEP):

- authentication as in protocol *ap4.0*
  - host requests authentication from access point
  - access point sends 128 bit nonce
  - host encrypts nonce using shared symmetric key
  - access point decrypts nonce, authenticates host
- no key distribution mechanism
- authentication: knowing the shared key is enough

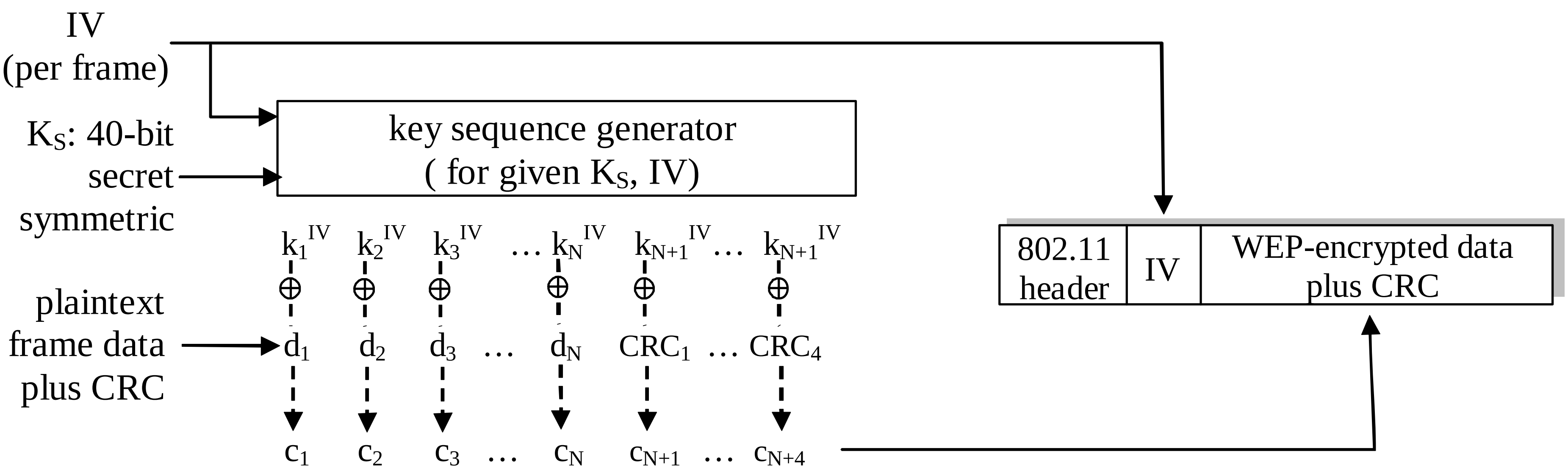
WEP data encryption

- Host/AP share 40 bit symmetric key (semi-permanent)
- Host appends 24-bit initialization vector (IV) to create 64-bit key
- 64 bit key used to generate stream of keys, k_i^{IV}
- k_i^{IV} used to encrypt ith byte, d_i, in frame:

$$c_i = d_i \text{ XOR } k_i^{IV}$$

- IV and encrypted bytes, c_i sent in frame

802.11 WEP encryption


IV
(per frame)
K_S: 40-bit
secret
symmetric
plaintext
frame data
plus CRC
key sequence generator
(for given K_S, IV)
k_1^{IV} k_2^{IV} k_3^{IV} … k_N^{IV} k_{N+1}^{IV} … k_{N+1}^{IV}
$\oplus$
d_1 d_2 d_3 … d_N CRC_1 … CRC_4
c_1 c_2 c_3 … c_N c_{N+1} … c_{N+4}
802.11
header
IV
WEP-encrypted data
plus CRC

Sender-side WEP encryption

Breaking 802.11 WEP encryption

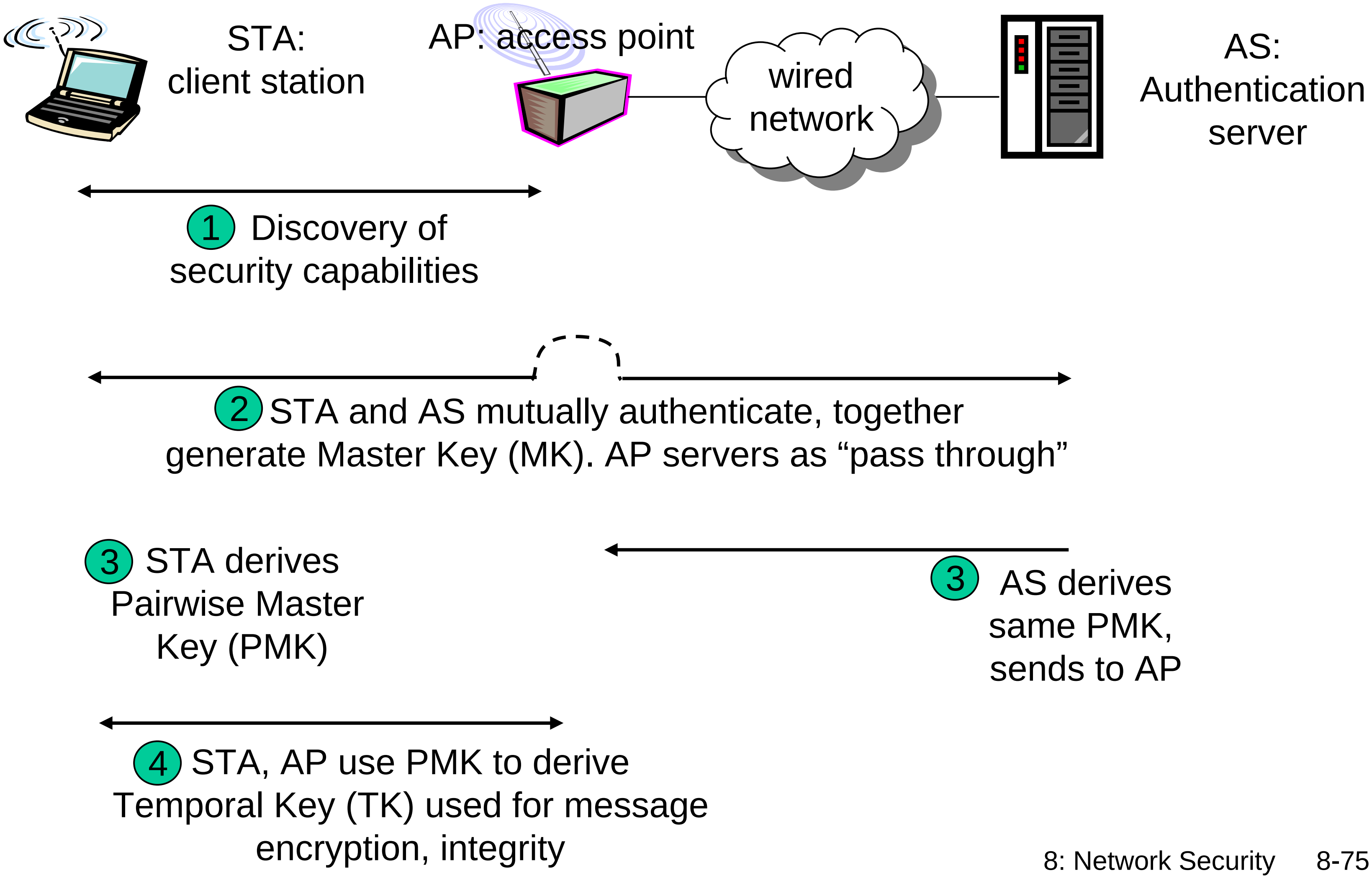
Security hole:

- 24-bit IV, one IV per frame, -> IV's eventually reused
- IV transmitted in plaintext -> IV reuse detected
- **Attack:**
  - Trudy causes Alice to encrypt known plaintext d_1 d_2 d_3 d_4 …
  - Trudy sees: $c_i = d_i$ XOR k_i^{IV}
  - Trudy knows c_i d_i, so can compute k_i^{IV}
  - Trudy knows encrypting key sequence k_1^{IV} k_2^{IV} k_3^{IV} …
  - Next time IV is used, Trudy can decrypt!

802.11i: improved security

- numerous (stronger) forms of encryption possible
- provides key distribution
- uses authentication server separate from access point

802.11i: four phases of operation



STA:
client station

AP: access point

wired
network

AS:
Authentication
server

1 Discovery of security capabilities

2 STA and AS mutually authenticate, together generate Master Key (MK). AP servers as “pass through”

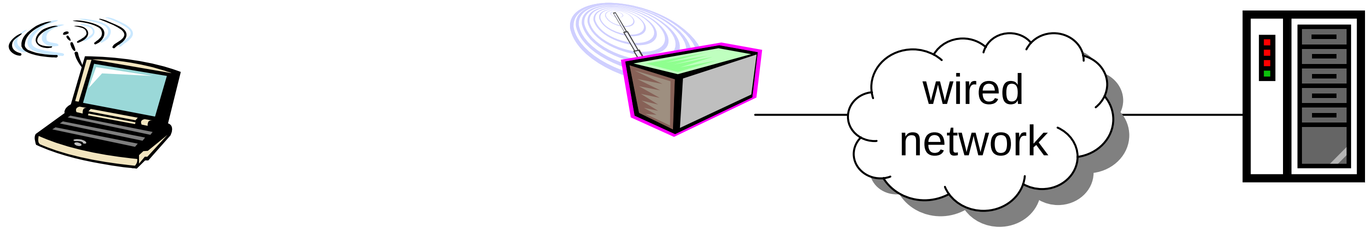
3 STA derives Pairwise Master Key (PMK)

3 AS derives same PMK, sends to AP

4 STA, AP use PMK to derive Temporal Key (TK) used for message encryption, integrity

EAP: extensible authentication protocol

- EAP: end-end client (mobile) to authentication server protocol
- EAP sent over separate “links”
  - mobile-to-AP (EAP over LAN)
  - AP to authentication server (RADIUS over UDP)



EAP TLS	
EAP	
EAP over LAN (EAPoL)	RADIUS
IEEE 802.11	UDP/IP

Network Security (summary)

Basic techniques......

- cryptography (symmetric and public)
- authentication
- message integrity
- key distribution

.... used in many different security scenarios

- secure email
- secure transport (SSL)
- IP sec
- 802.11