

# *Computer Networks*



***NETWORKS,  
LARGE AND SMALL***

# Computer networks

- a) **A computer network is a combination of systems connected through transmission media.**
- **Local area network (LAN)**
  - **Metropolitan area network (MAN)**
  - **Wide area network (WAN)**

# Model and protocol

- a) A **model** is the specification set by a standards organization as a guideline for designing networks.
- b) A **protocol** is a set of rules that controls the interaction of different devices in a network or an internetwork.

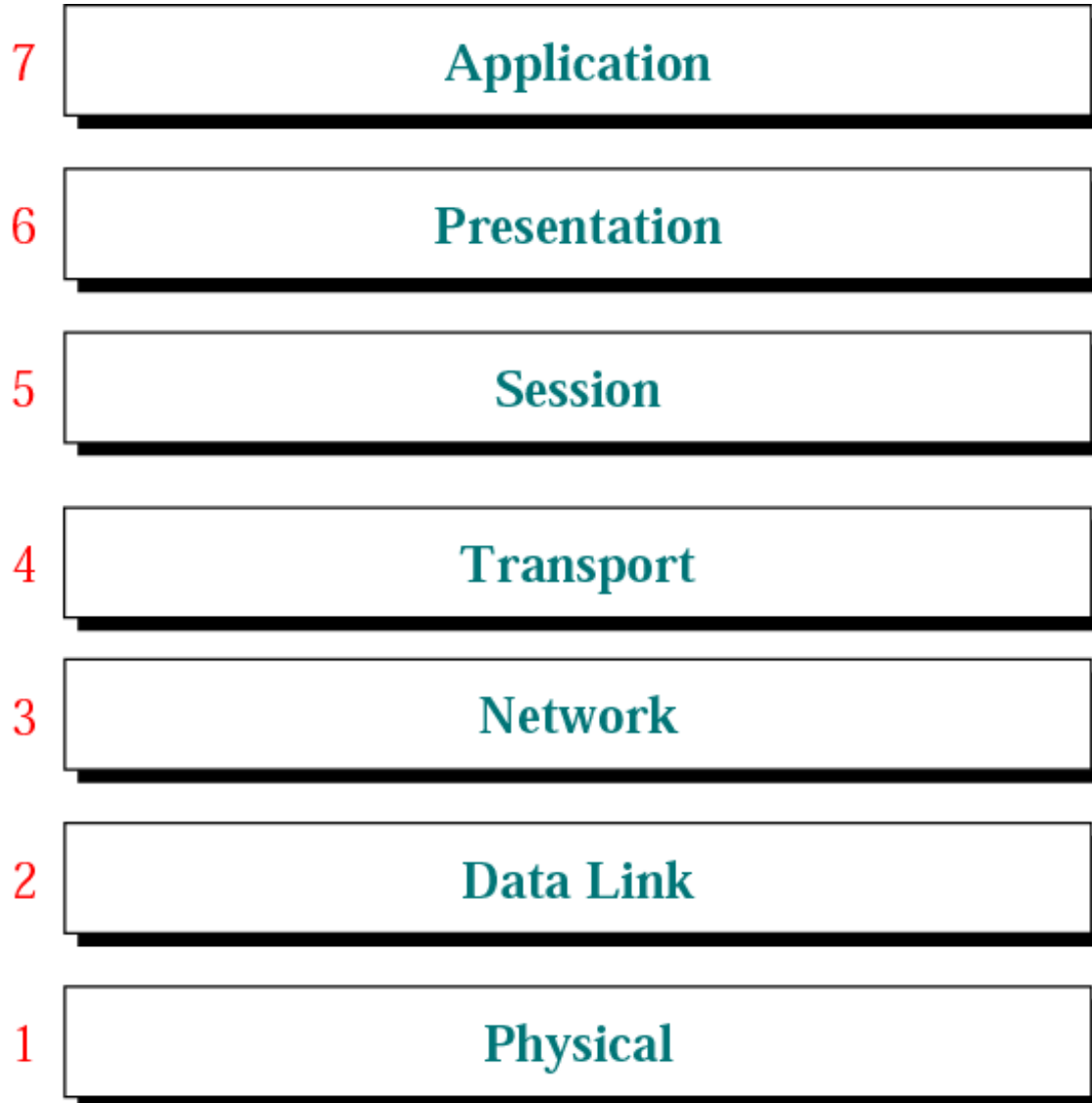
# ***OSI MODEL***



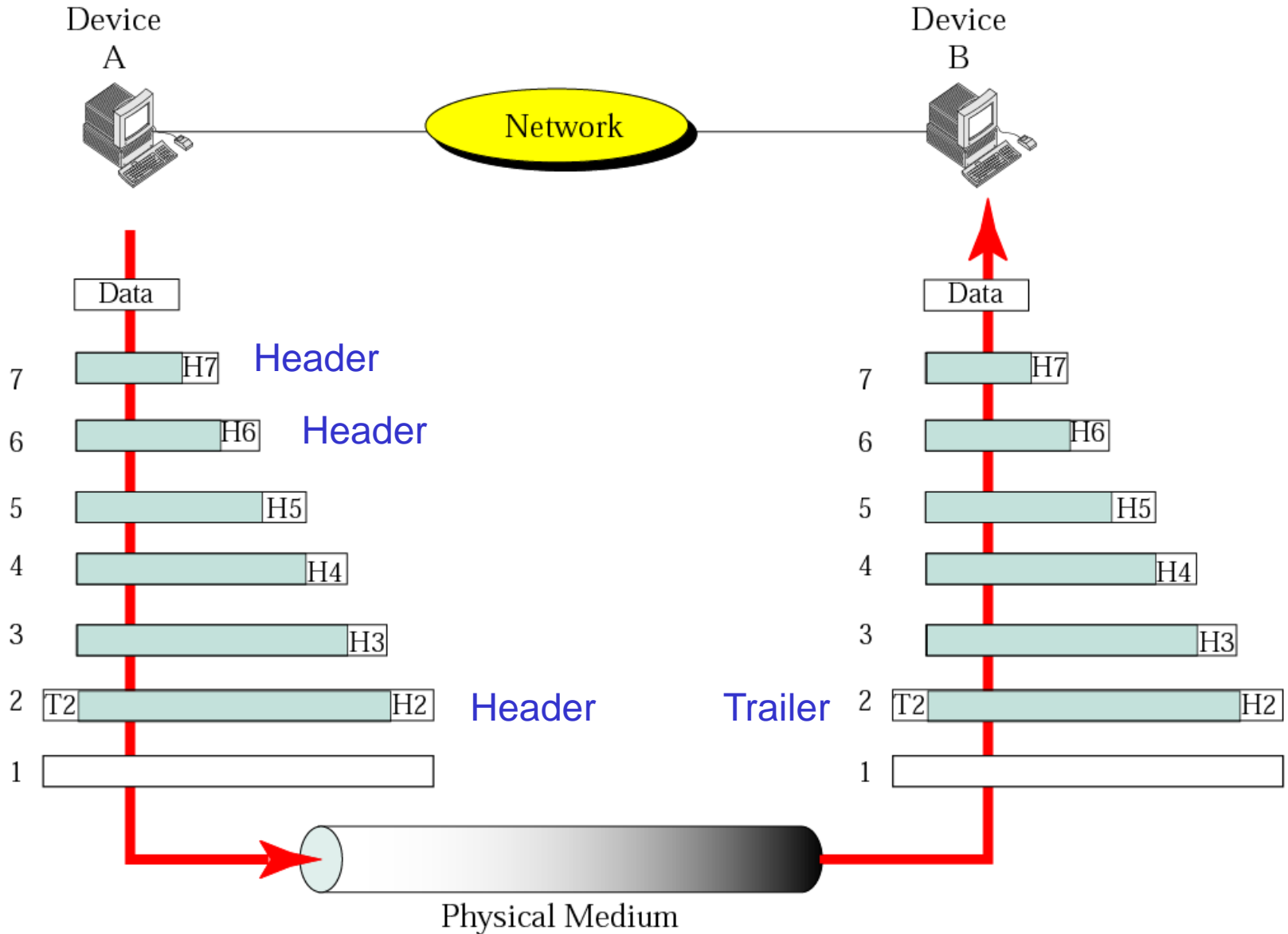
**Note:**

*The Open Systems Interconnection (OSI) model is a theoretical model that shows how any two different systems can **communicate** with each other.*

# The OSI model



# Flow of data in the OSI model





# Seven layers of OSI model

- a) **Physical layer**
- b) **Data-link layer**
- c) **Network layer**
- d) **Transport layer**
- e) **Session layer**
- f) **Presentation layer**
- g) **Application layer**

# Physical layer

- a) **The physical layer is responsible for transmitting a bit stream over a physical medium.**
- b) **It encodes and decodes bits into groups of bits.**
- c) **It then transforms a stream of bits into a signal.**

# Data-link layer

- a) The data-link layer organizes bits into **logical units** called **frames**.
- b) The data-link layer is responsible only for **node-to-node delivery** of the frame.
- c) The data-link layer is often responsible for **error handling** between two adjacent stations.

# Network layer

- a) **The network layer is responsible for delivery of a **packet** between the original source and final destination.**
- b) **Using **logical addresses (IP addresses)** instead of physical addresses.**
- c) **Example of IP address**
  - **140.122.76.121 (4 Bytes)**

# Transport layer

- a) The transport layer is responsible for source-to-destination delivery of the **entire message**.
- b) The transport layer is responsible for breaking the **entire message** into **several packets** and delivery them to the network layer.
- c) The transport layer is responsible for ensuring that the whole message is transmitted.
- d) If packets arrive **out of order**, they must be reorganized.

# Session layer

- a) The session layer is designed to control the **dialog** between users.
- b) The **synchronization** (同步的) **points** divides a long message into smaller ones and ensure that each section is received and acknowledged by the receiver.
- c) Most network implementations **today** do **not** use a separate session layer, their services are usually included in the **application** layer.

# Presentation layer

- a) The presentation layer is concerned with the syntax and semantics of the **information exchanged** between two systems.
- b) It deals with the fact that different systems use **different coding methods**.
  - Compress and decompress data
  - Encrypt and decrypt data
- c) Most implementations do **not** use a presentation layer **today**, their services are usually included in other layer.

# Application layer

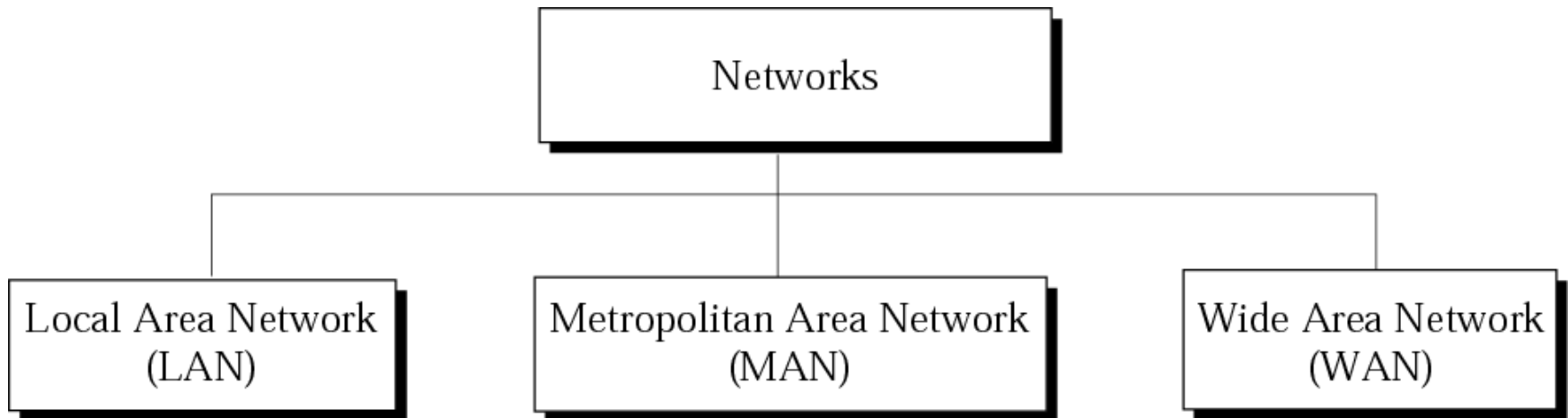
- a) The application layer enables the **user** to access the network.
- b) It defines **common applications** that can be implemented to make the job of the user **simpler**.
- c) Will be discussed later...





***CATEGORIES  
OF  
NETWORKS***

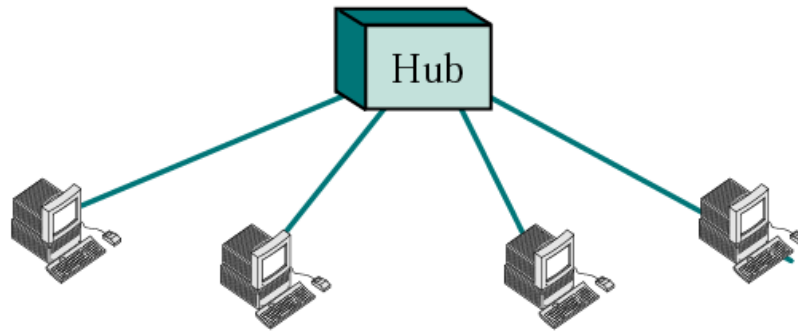
# Categories of networks



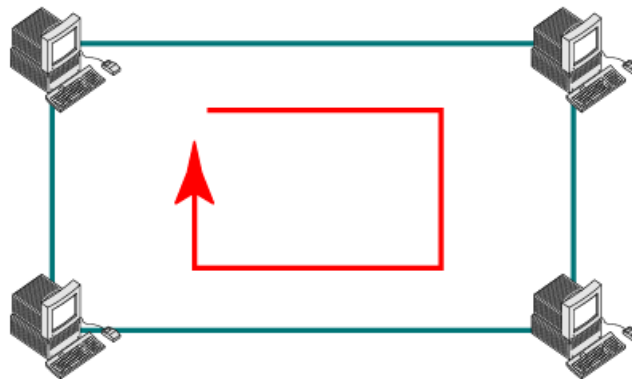
# LANs



**a. Bus LAN**



**b. Star LAN**

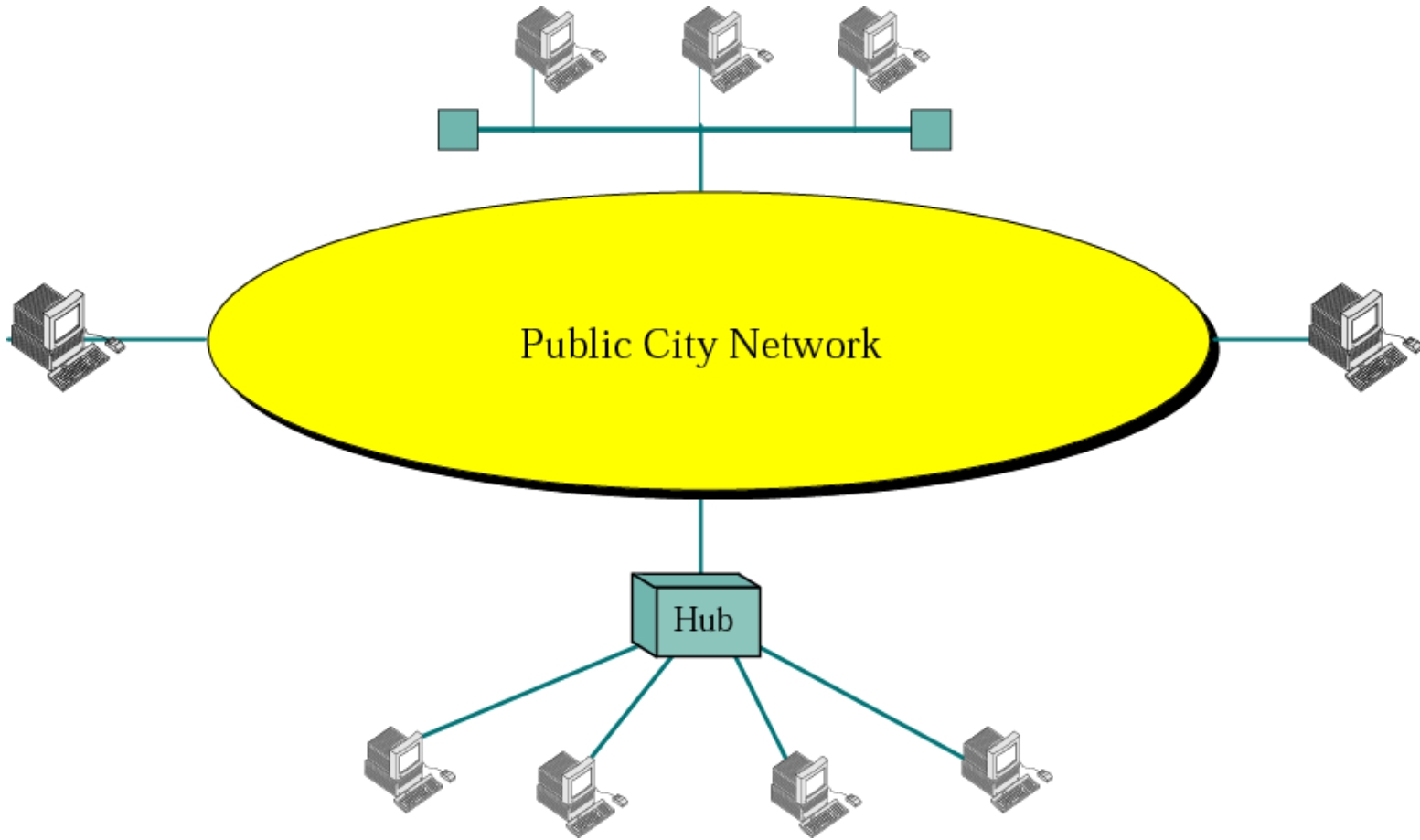


**c. Ring LAN**

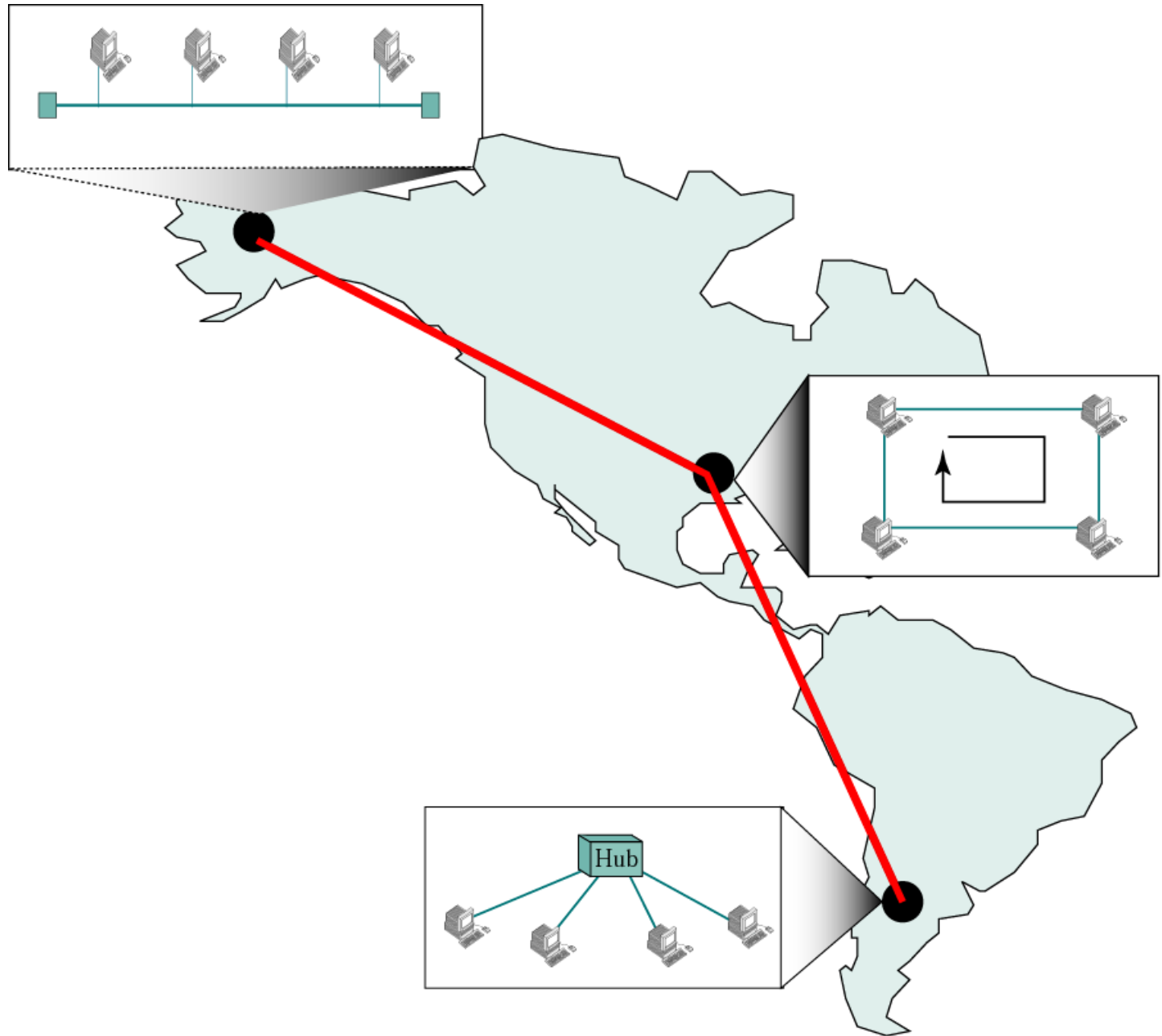
# Local area network

- a) Bus topology
  - When a station sends a frame to another computer, **all** computer receiver the frame and check its destination address.
- b) Star topology
  - Hub (all, like a bus) v.s. switch (one)
- c) Ring topology
  - When a computer needs to send a frame to another computer, it sends it to its **neighbor**.

# MAN



WAN



# Wide area network

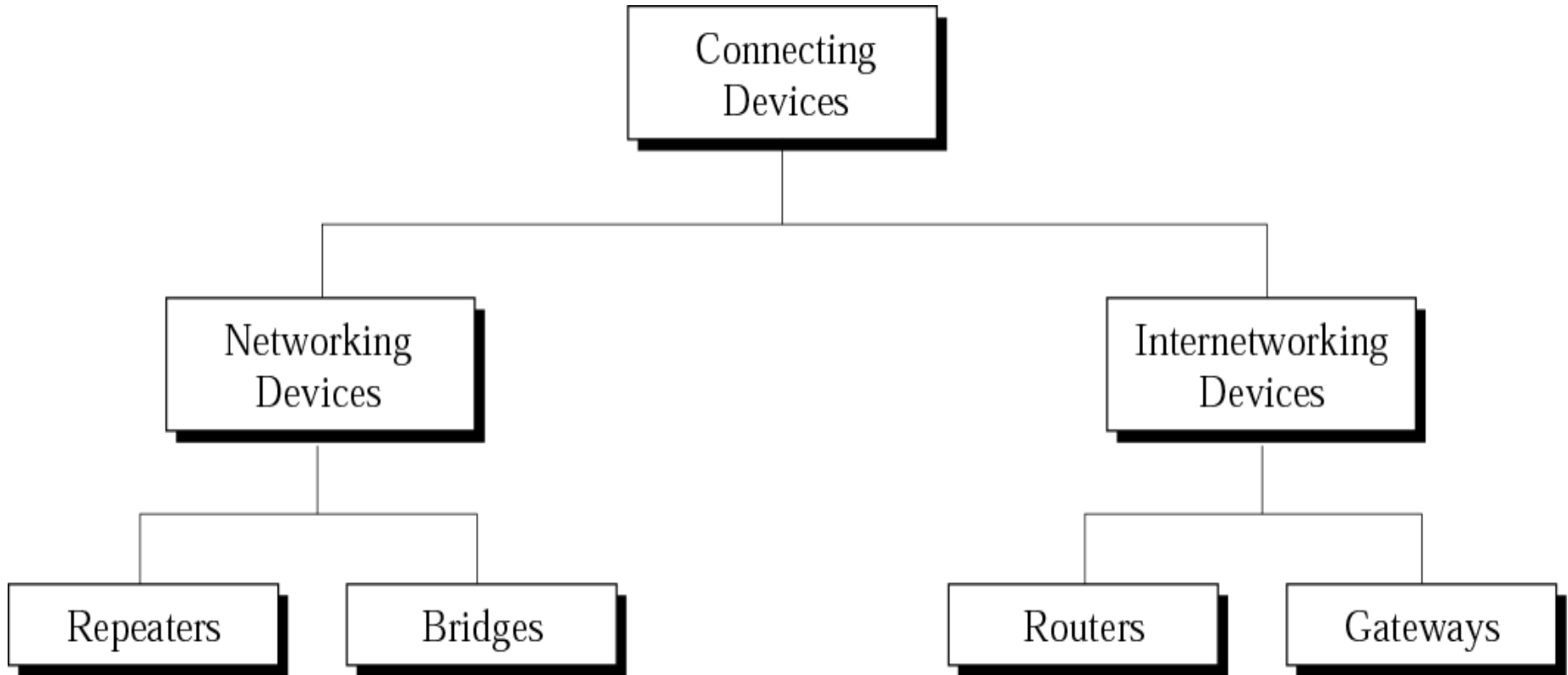
- a) WAN is the connection of individual computers or LANs over a large area.
- b) A person using a telephone line to connect to an ISP is using a WAN.
- c) ISP: internet service provider



***CONNECTING  
DEVICES***



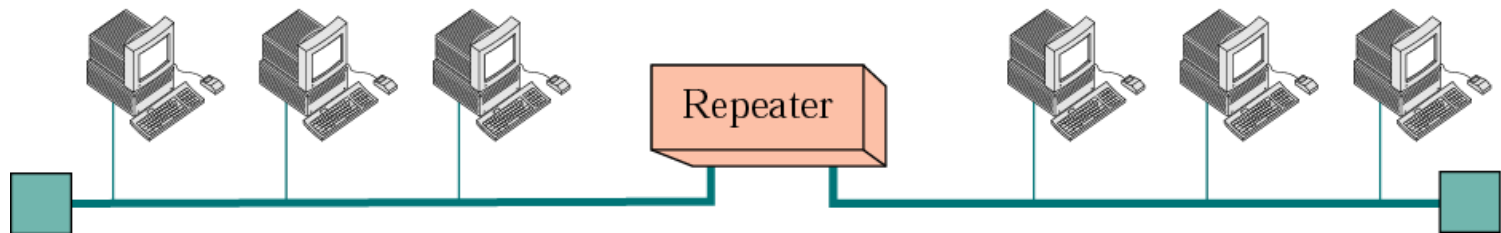
# Connecting devices



# Repeater



a. Without Repeater



b. With Repeater

- a) A repeater is an electronic device and operate only in the **physical layer** of the OSI model.
- b) A repeater can regenerate the signal and send it to the rest of the network.



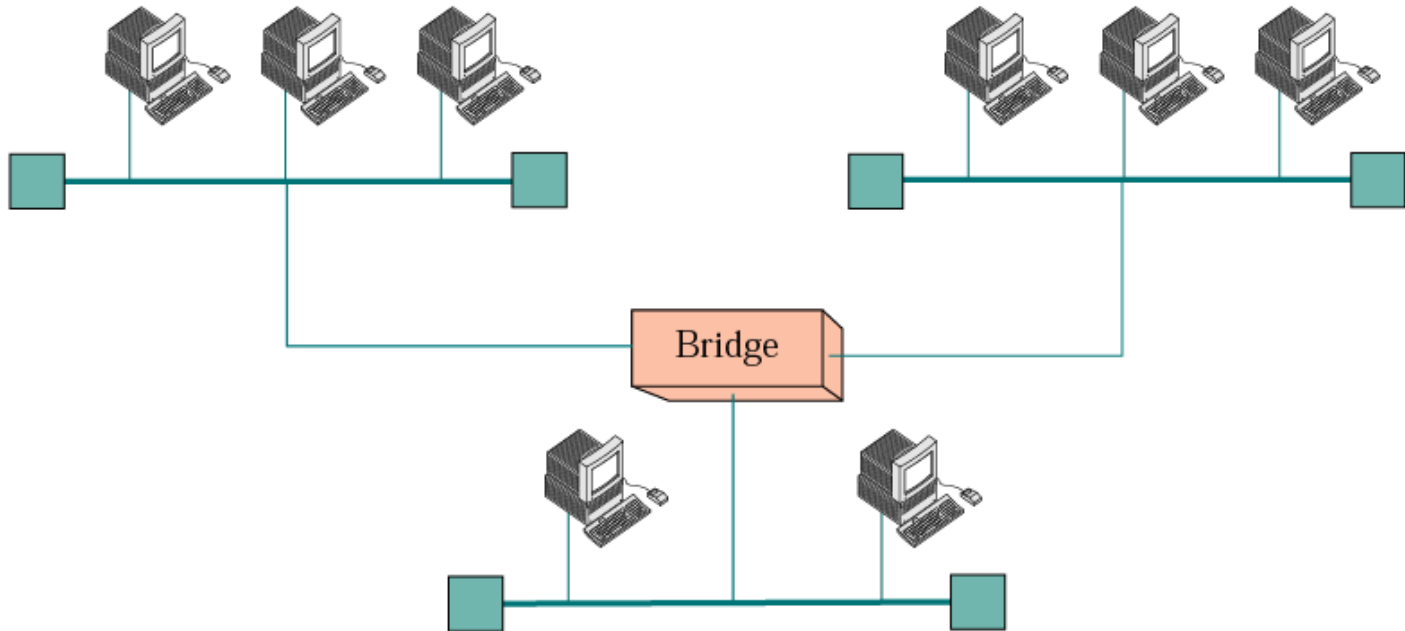
**Note:**

***Repeaters operate at the first layer  
(physical layer) of the  
OSI model.***

# Bridge



a. Without Bridge



b. With Bridge

# Bridges

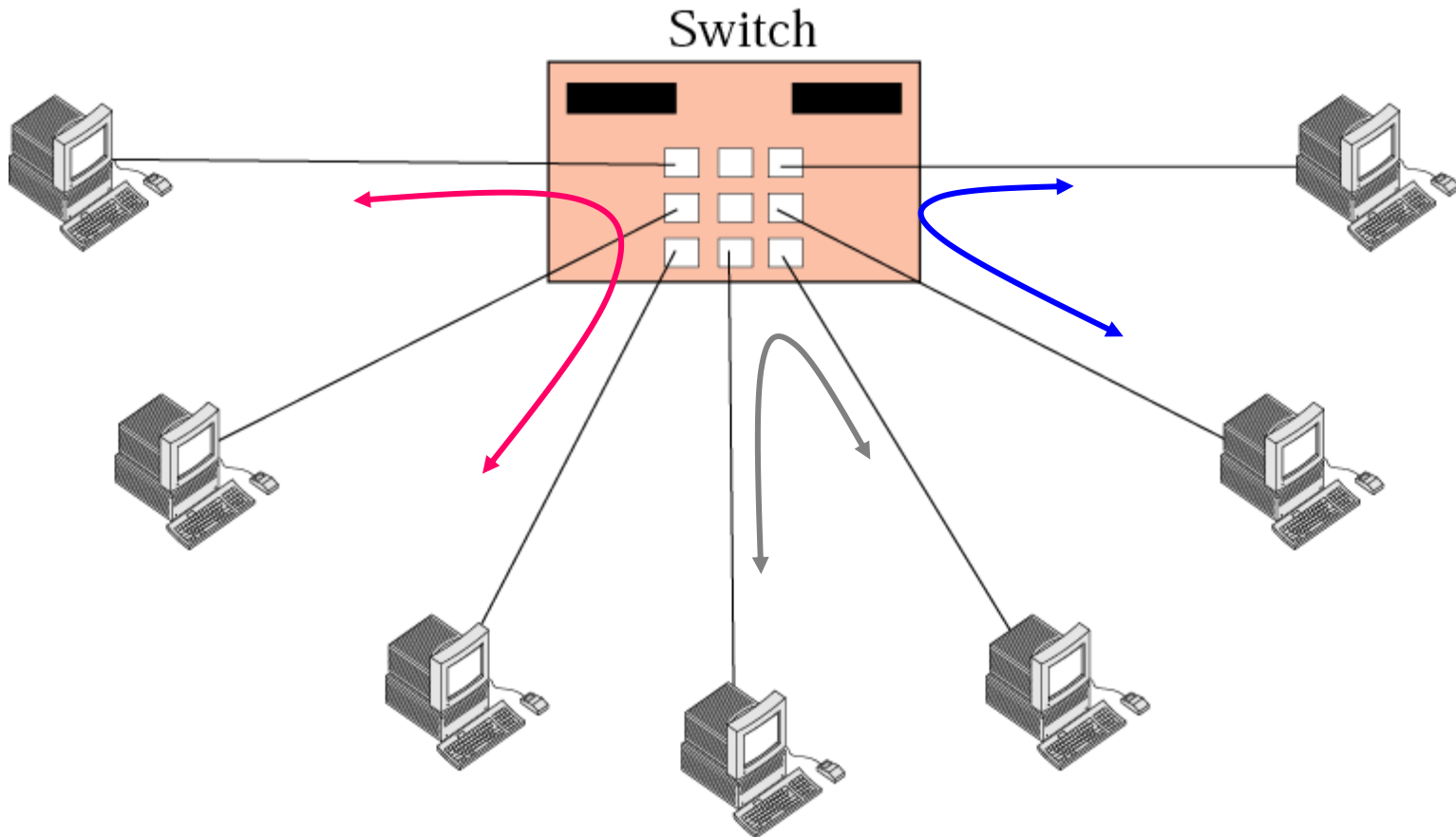
- a) In bus topology, a bridge is a traffic controller.
- It can divide a long bus into smaller segments so that each segment is independent trafficwise.
  - The bridge uses a table to decide if the frame needs to be forwarded to another segment.
  - With a bridge, two or more pairs of stations can communicate at the same time.



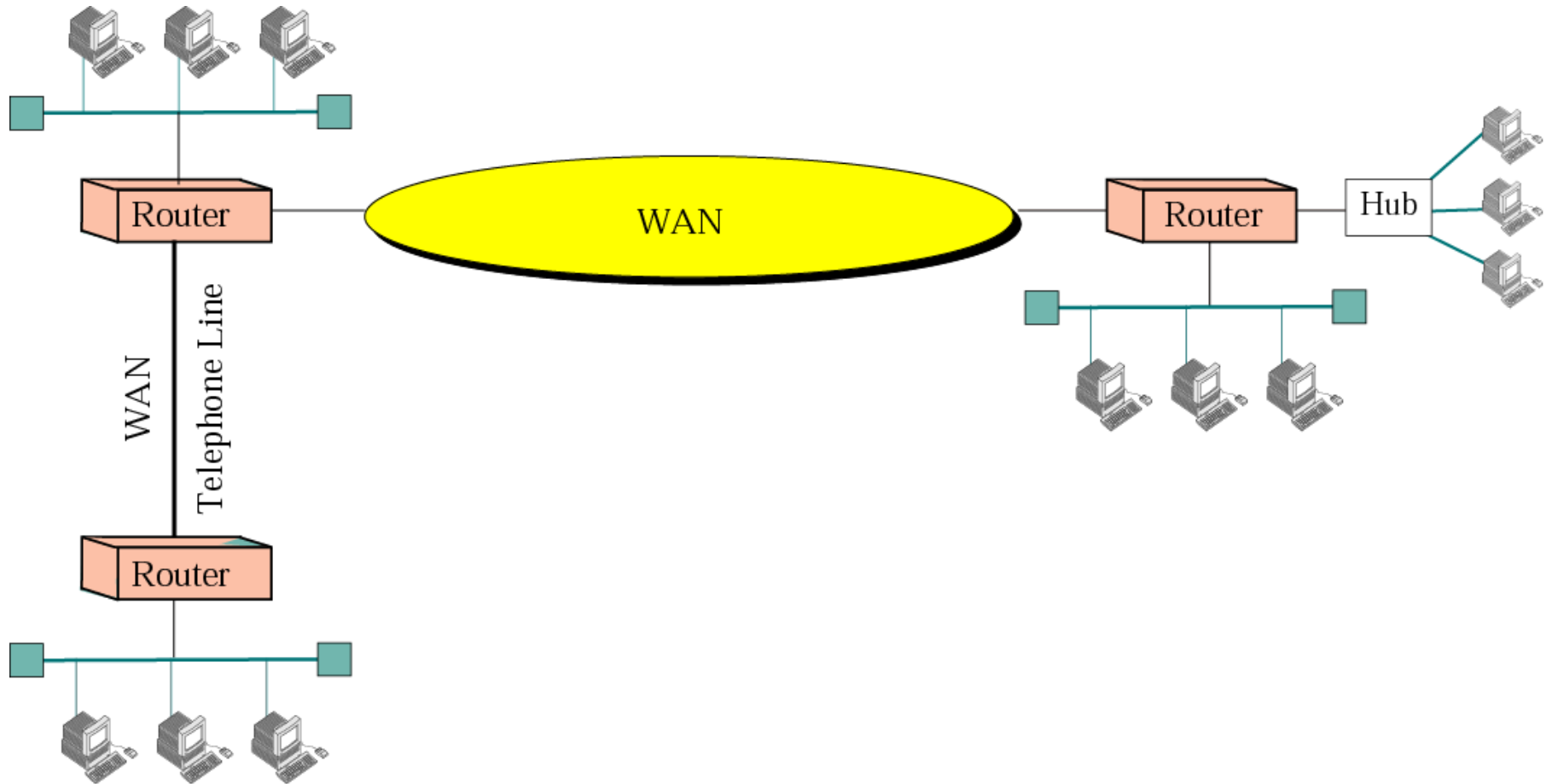
**Note:**

***Bridges operate at the first two  
layers  
(physical layer and data-link layer)  
of the OSI model.***

# Switch—one kind of dynamic bridge



# Routers in an internet







**Note:**

***Routers operate at the first three  
layers  
(physical, data-link, and network  
layer)  
of the OSI model.***

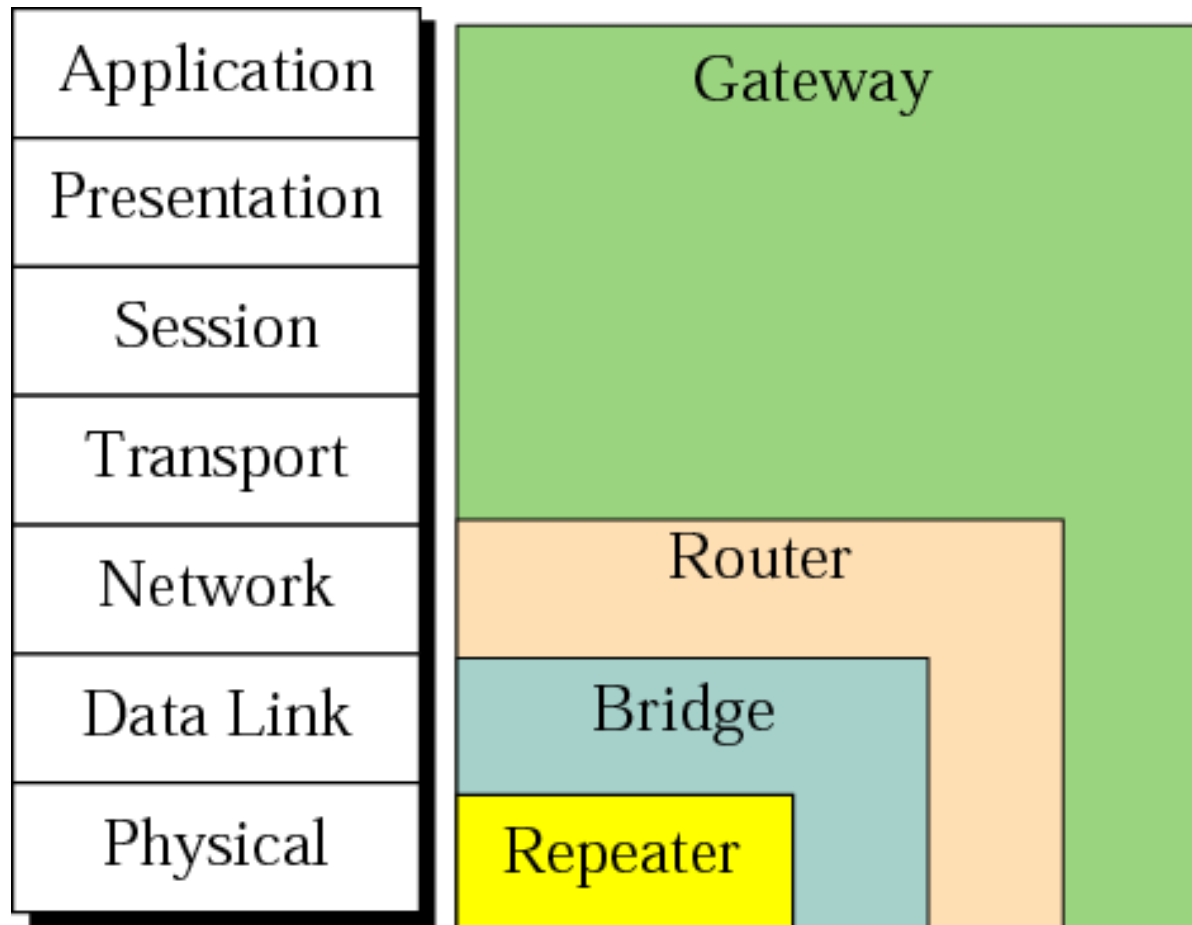
# Routers

- a) Whereas a bridge filters a frame based on the physical address of the frame, a router routes a packet based on the logical address of the packet.
- b) Whereas a bridge may connect two segments of a LAN, a router can connect two independent networks.

# Gateways

- a) A gateway is a connecting device that acts as a protocol converter.
- b) A gateway is usually a computer installed with the necessary software.
- c) Today the term gateway is used interchangeably with the term router. The distinction between the two terms is disappearing.

# Connecting devices and the OSI model



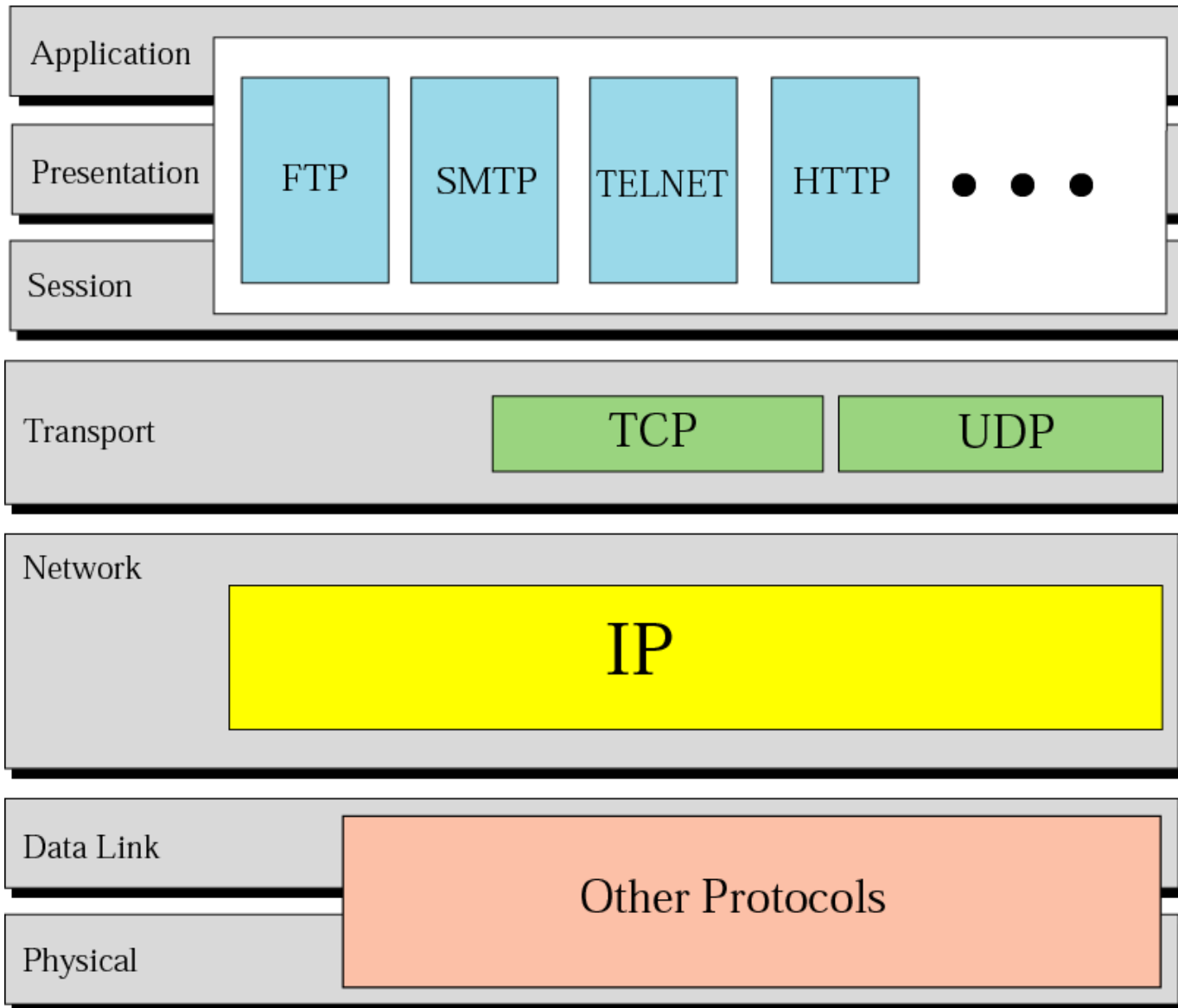


***THE INTERNET  
AND  
TCP/IP***

# Protocols

- a) **TCP**: Transmission control protocol
- b) **UDP**: User datagram protocol
- c) **IP**: Internet protocol
- d) **FTP**: File transfer protocol
- e) **SMTP**: Simple mail transfer protocol
- f) **POP**: Post office protocol
- g) **HTTP**: Hypertext transfer protocol

# TCP/IP and OSI model



# Network layer

## IP addresses in dotted-decimal notation

Bit Pattern

10000001 00001010 00000111 00011110



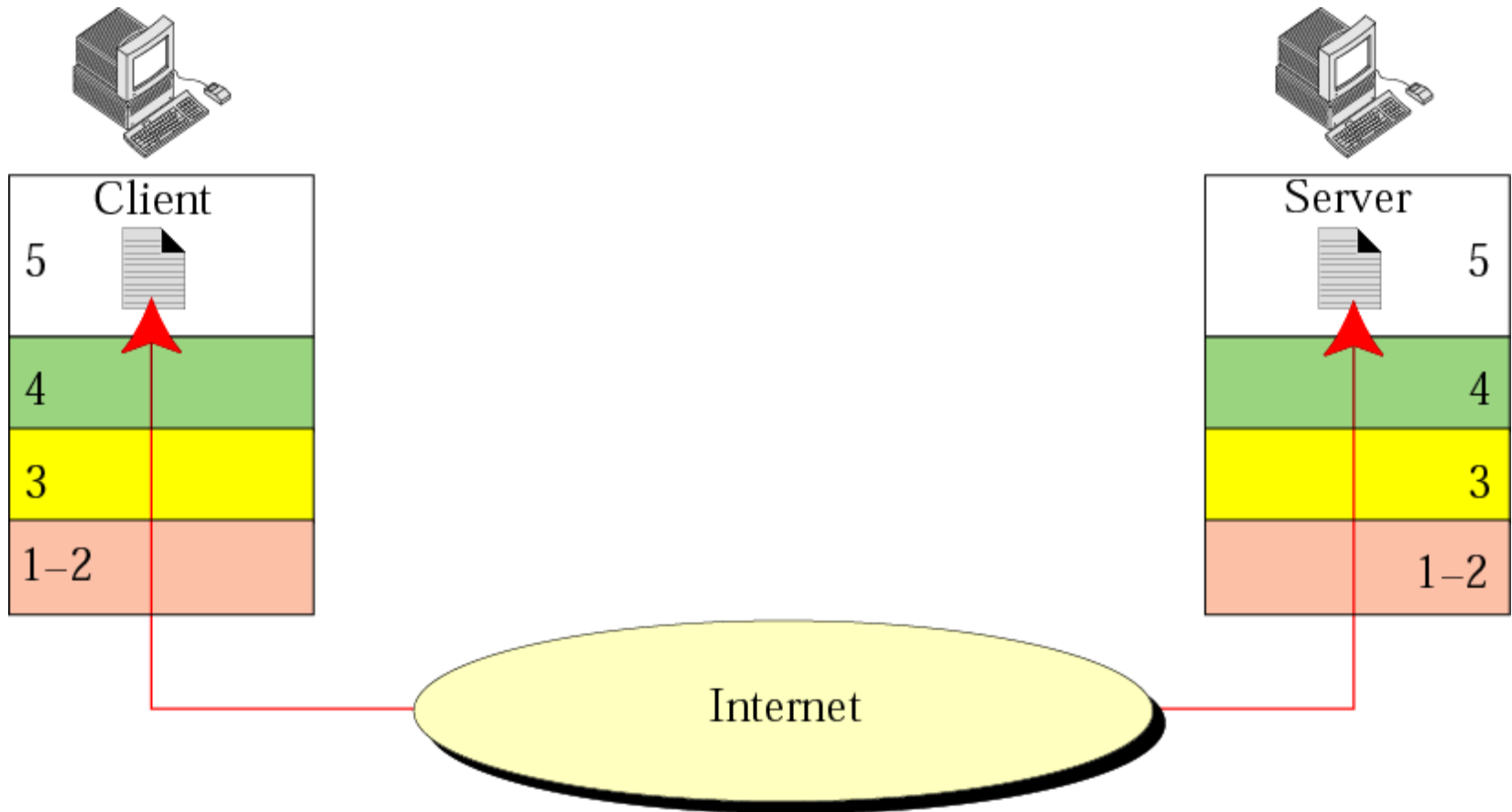
**129.10.7.30**

Dotted-Decimal Notation



# Application layer

## Client-server model

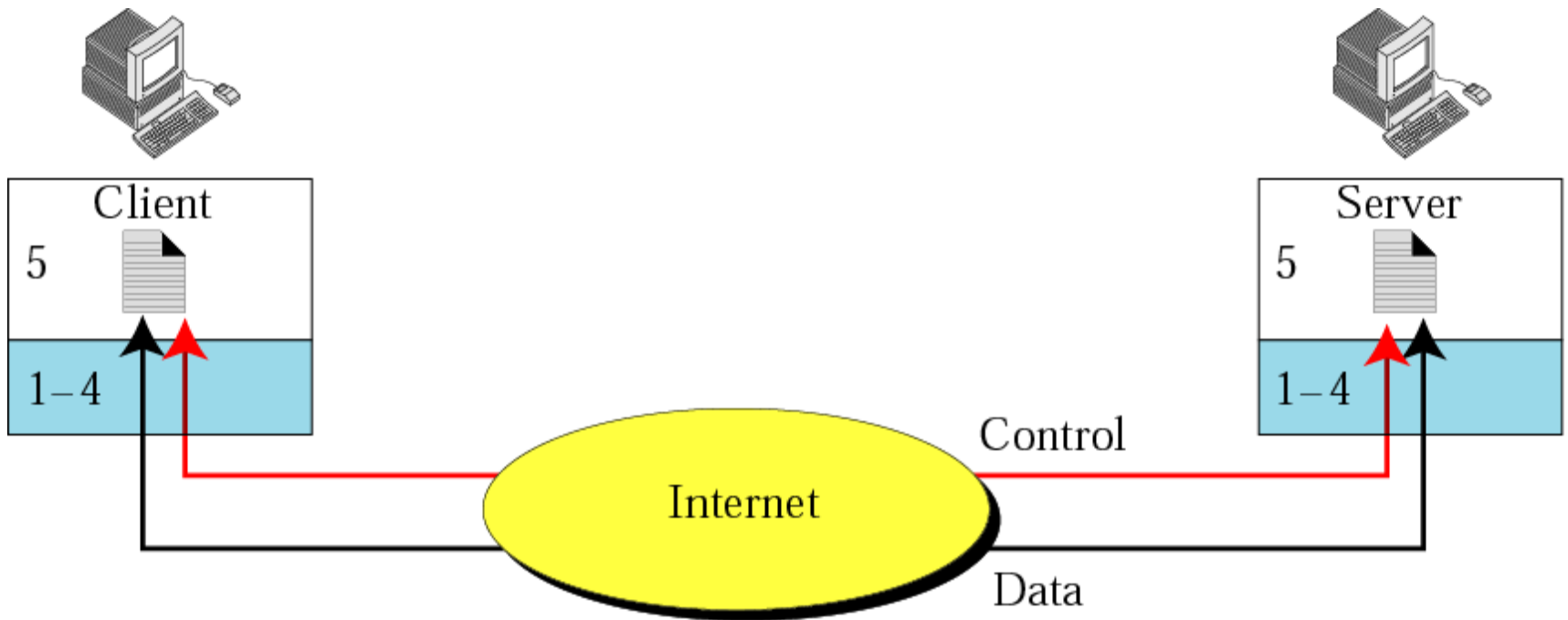


# Client-server model

- a) **Client:** an application program running on a local machine
- b) **Server:** an application program running on a remote machine
- c) A client requests a service from a server.
- d) The server program is always running, and the client program runs only when needed.

# FTP -- Client-server model

## FTP

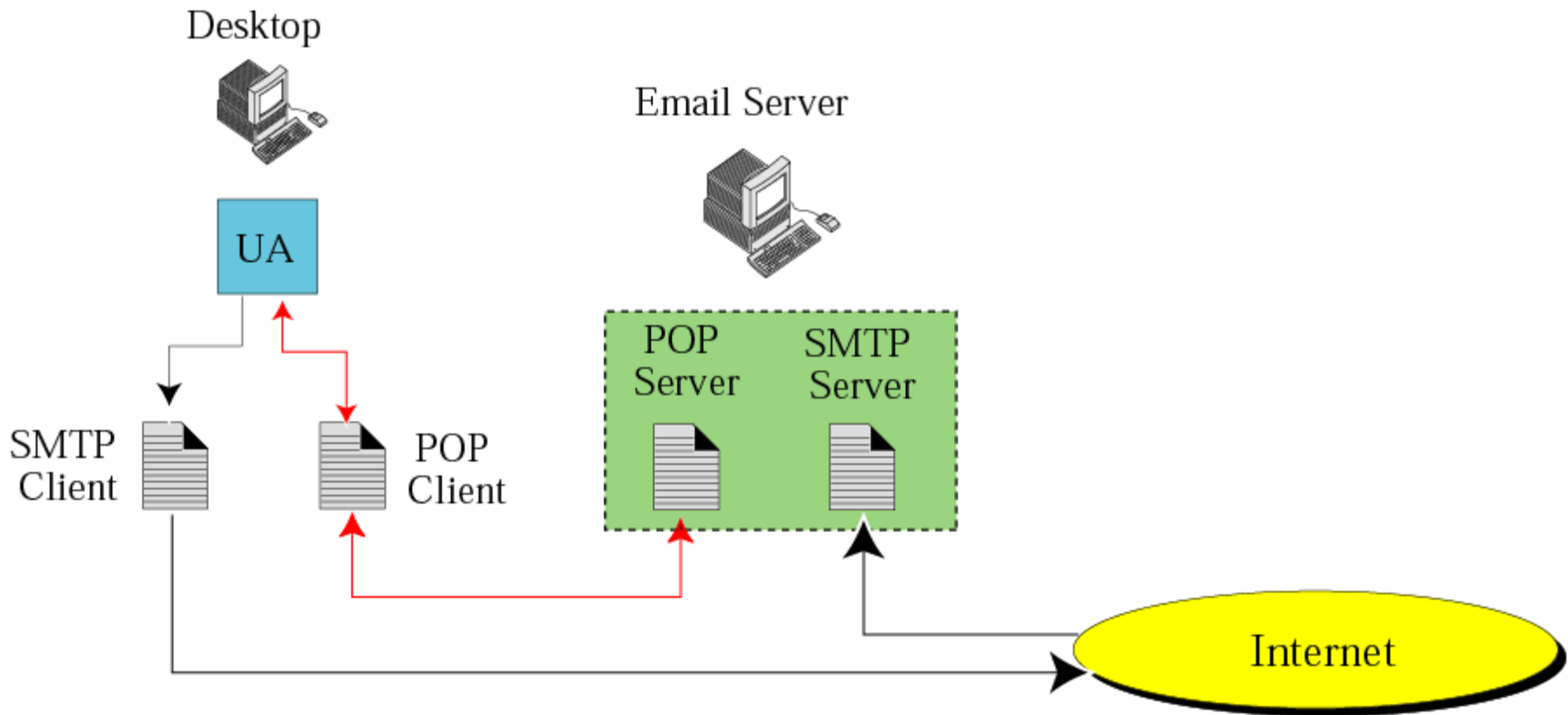


# FTP -- Client-server model

- a) FTP was designed to resolve two problems:
- Different coding systems in use
    - One machine may use ASCII, and other may use Unicode
  - Different file formats in use

# SMTP -- Client-server model

## SMTP/POP

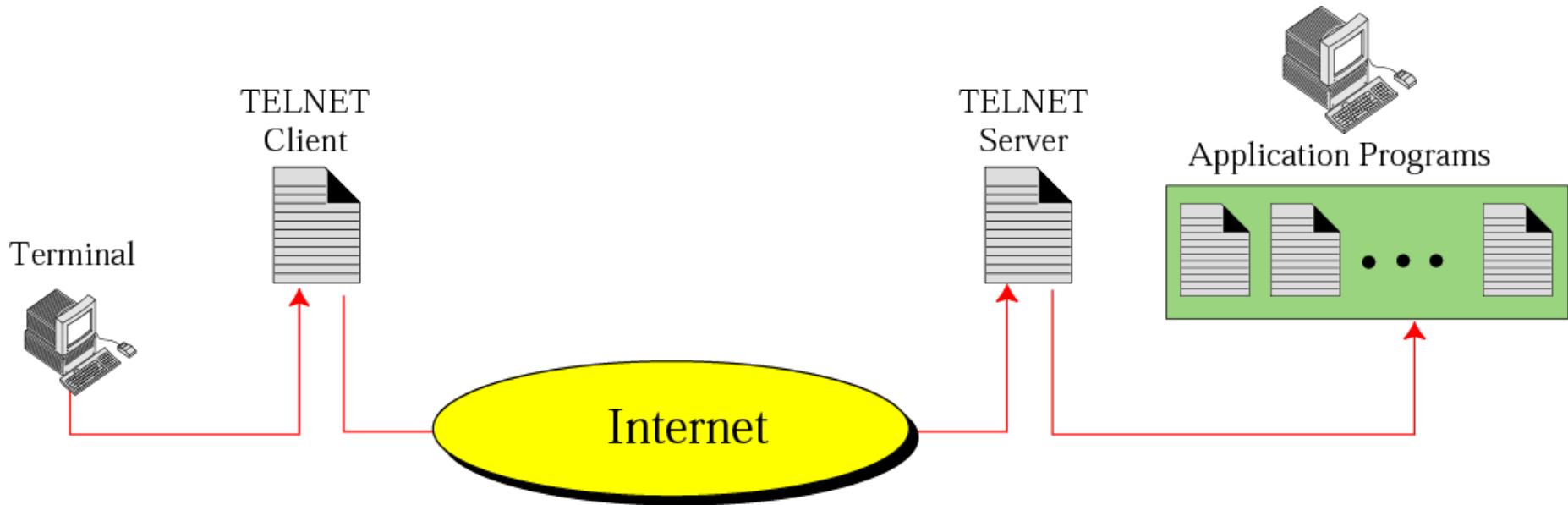


# Email address



violet@ice.ntnu.edu.tw

# TELNET-- a general client-server model



- a) Local login v.s. remote login
- b) TELNET is a general client-server program on the Internet that allow remote login

# HTTP

## URL



<http://www.ice.ntnu.edu.tw/~violet>

<ftp://140.122.77.121>

- a) URL: Uniform resource locator
  - A special kind of addressing using by HTTP

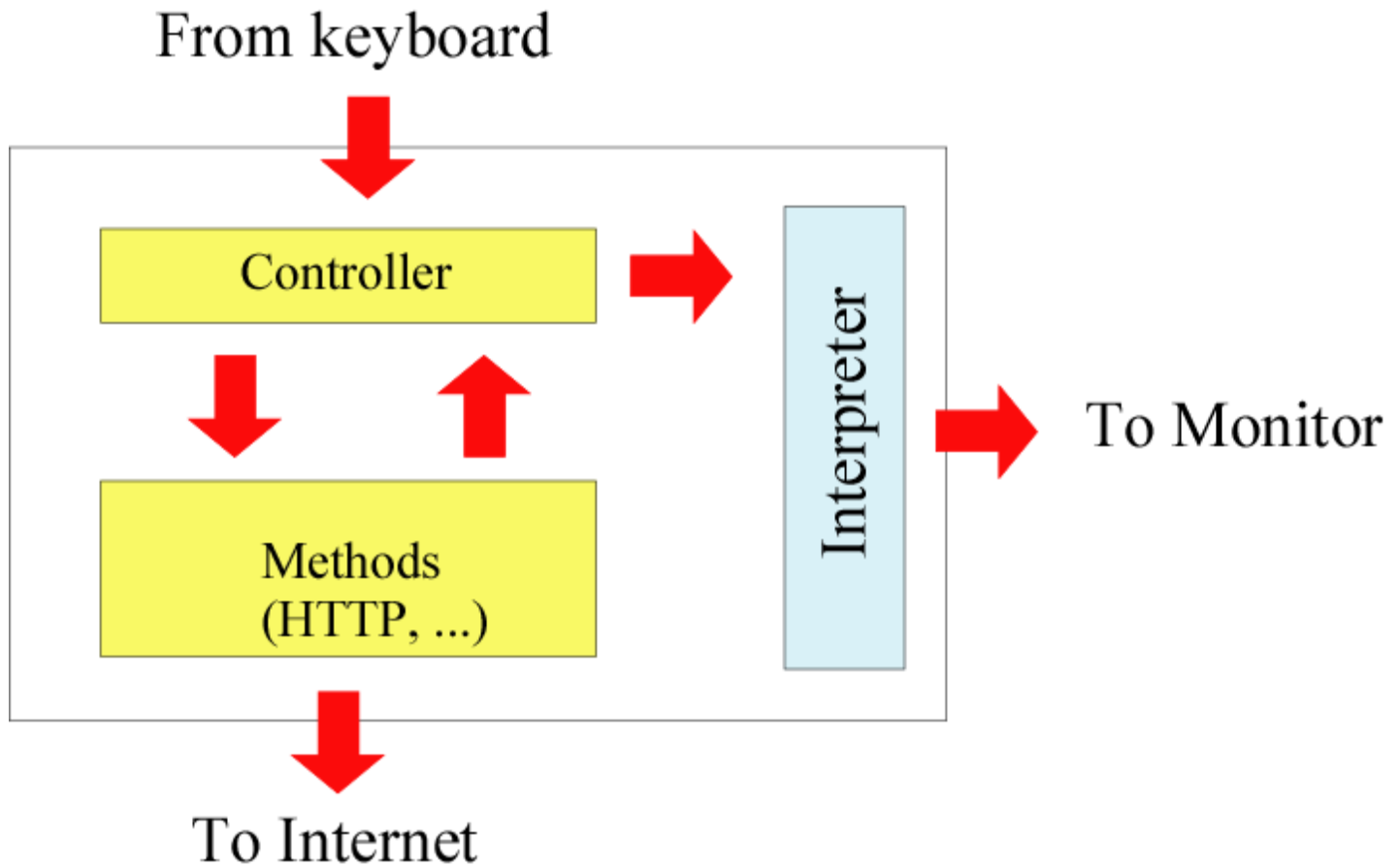


# WWW

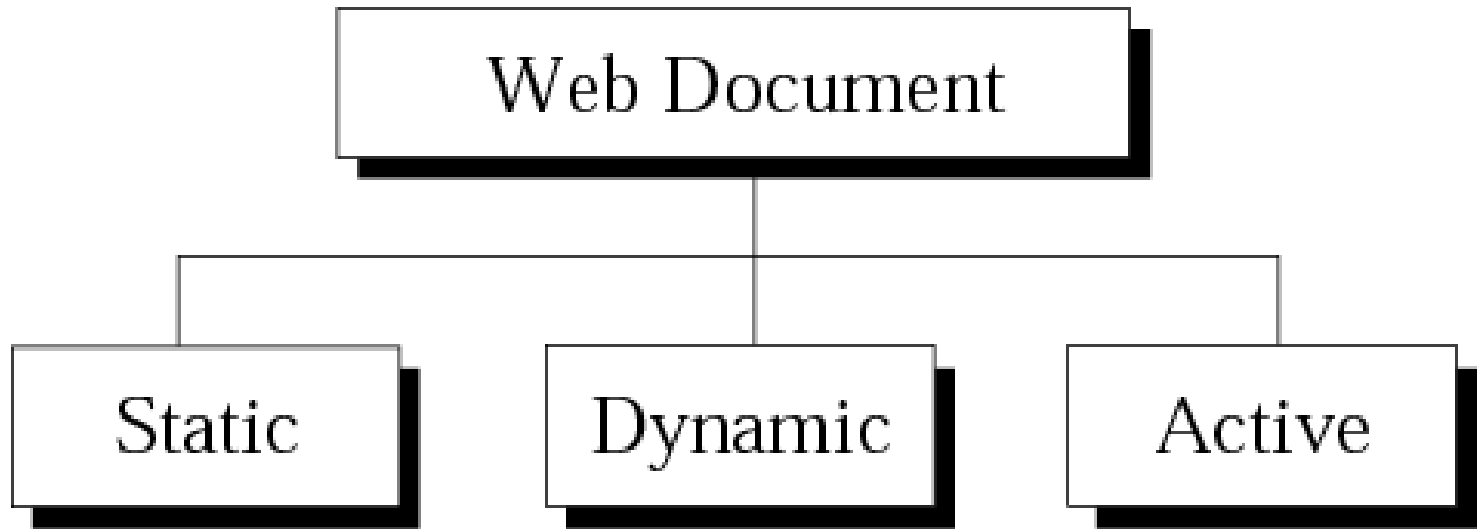
## a) WWW: World wide web

- Hypertext: home page
- Browser
- Document Types
  - Static documents
  - Dynamic documents
  - Active documents

# Browser



# Categories of Web documents



- Static documents
  - HTML: Hypertext Markup Language
- Dynamic documents
  - CGI: Common Gateway Interface (Perl)
- Active documents
  - Java language

# The Physical Layer

# Outline of Physical Layer

- It defines the electrical, timing and their interfaces by which bits are sent as signals over the channel.
- Foundation on which the network is built.
- The properties of different kinds of physical channel determine the performance – throughput, latency, and error rate.
- Data Transmission rate is limited
- Three types of transmission media:
  - Guided (copper wire and fiber optics)
  - Wireless (terrestrial radio), and
  - Satellite

# Outline of Physical Layer

- Digital Modulation: How analog signals are converted into digital bits and back again.
- Multiplexing Schemes: Exploring how multiple conversations can be put on the same transmission medium at the same time without interfering with one another.
- Examples of communication systems used in practice for wide area computer networks:
  - The (fixed) telephone system
  - The mobile phone system, and
  - The cable television system.

# Theoretical Basis for Data Communication

- Fourier analysis
- Bandwidth-limited signals
- Maximum data rate of a channel

# Fourier Analysis

- We model the behavior of variation of voltage or current with mathematical functions
- **Fourier series** is used to expand any periodic function with period  $T$

$$g(t) = \frac{1}{2}c + \sum_{n=1}^{\infty} a_n \sin(2\pi nft) + \sum_{n=1}^{\infty} b_n \cos(2\pi nft)$$

- $f=1/T$  – *fundamental frequency*.
- $a_n, b_n$  – *are the sine and cosine amplitudes of the  $n$ 'th harmonic*.
- $c$  – *is a constant*.

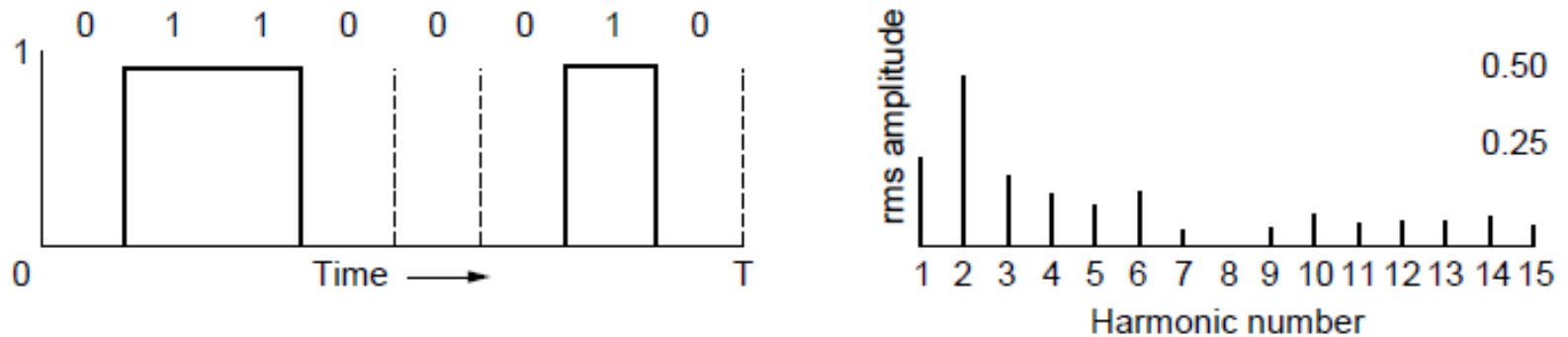


# Fourier Analysis

- Function reconstructed with

$$a_n = \frac{2}{T} \int_0^T g(t) \sin(2\pi nft) dt \quad b_n = \frac{2}{T} \int_0^T g(t) \cos(2\pi nft) dt \quad c = \frac{2}{T} \int_0^T g(t) dt$$

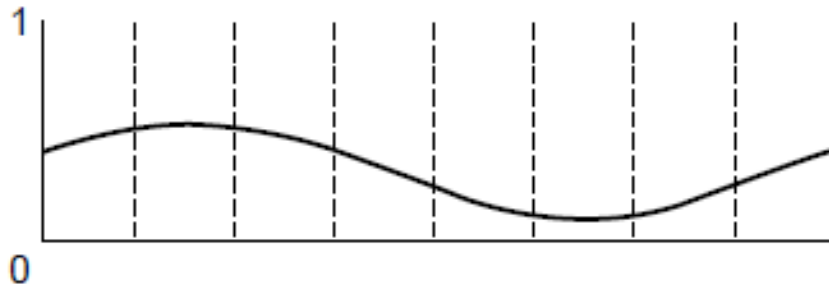
# Bandwidth-Limited Signals (1)



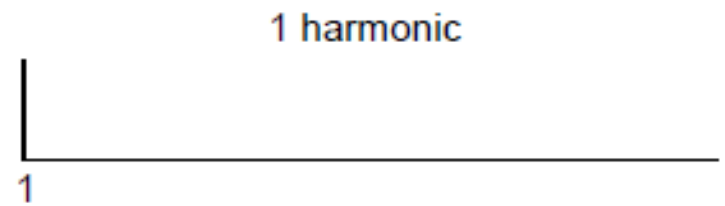
(a)

A binary signal and its root-mean-square  
Fourier amplitudes.

# Bandwidth-Limited Signals (2)

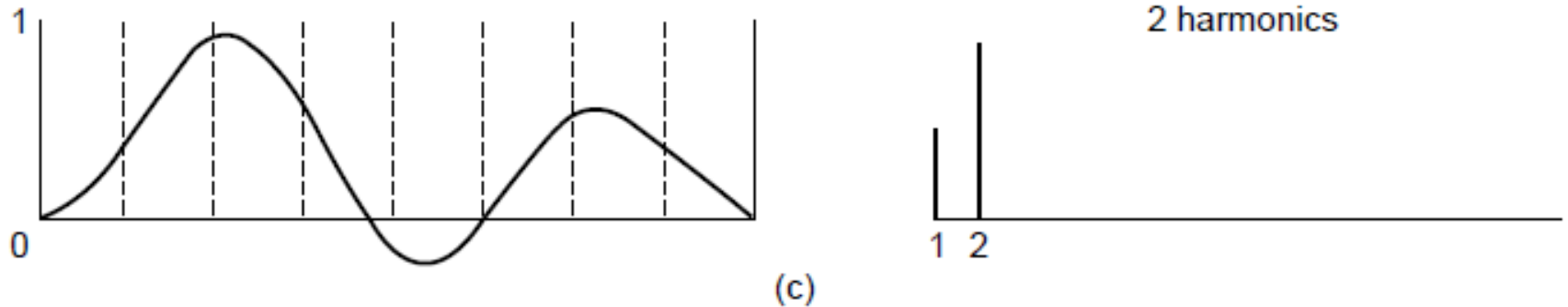


(b)



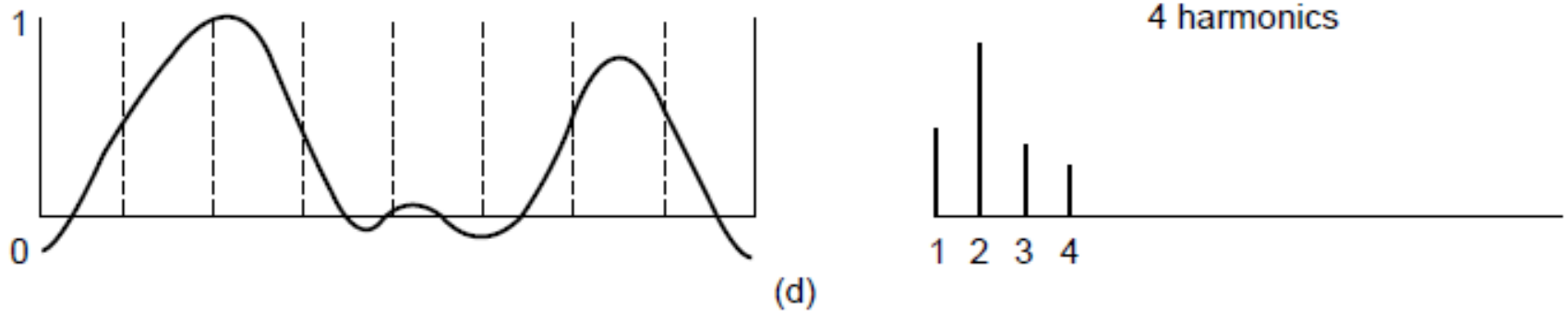
(b)-(e) Successive approximations to the original signal.

# Bandwidth-Limited Signals (3)



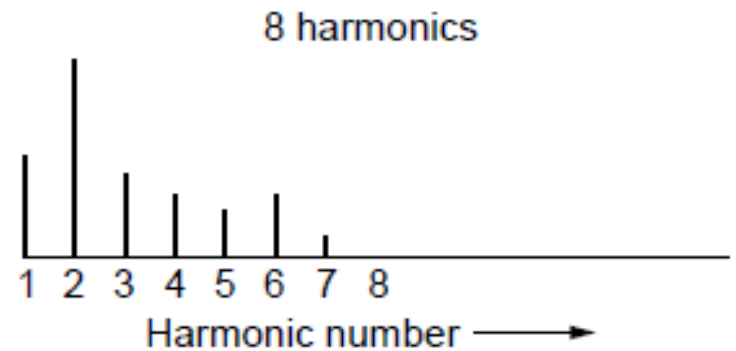
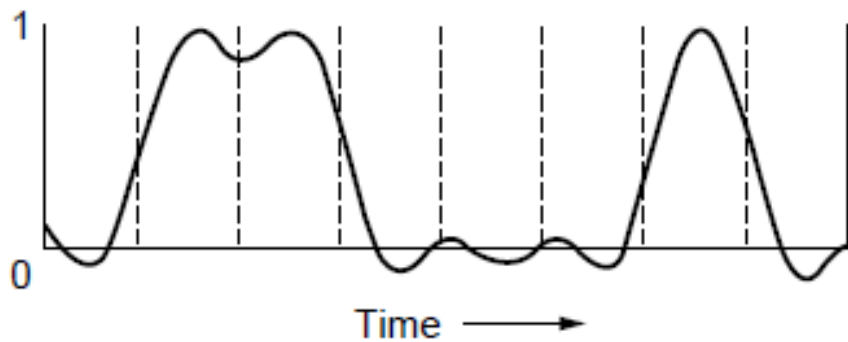
(b)-(e) Successive approximations to the original signal.

# Bandwidth-Limited Signals (4)



(b)-(e) Successive approximations to the original signal.

# Bandwidth-Limited Signals (5)



(e)

(b)-(e) Successive approximations to the original signal.

# Bandwidth-Limited Signals (6)

<b>Bps</b>	<b>T (msec)</b>	<b>First harmonic (Hz)</b>	<b># Harmonics sent</b>
300	26.67	37.5	80
600	13.33	75	40
1200	6.67	150	20
2400	3.33	300	10
4800	1.67	600	5
9600	0.83	1200	2
19200	0.42	2400	1
38400	0.21	4800	0

Relation between data rate and harmonics for our example.

# The Maximum Data Rate of a Channel

- Nyquist's theorem  
maximum data rate =  $2 B \log_2$  bits / sec
- Shannon's formula for capacity of a noisy channel  
maximum number of bits / sec =  $B \log_2 (1 + S / N)$



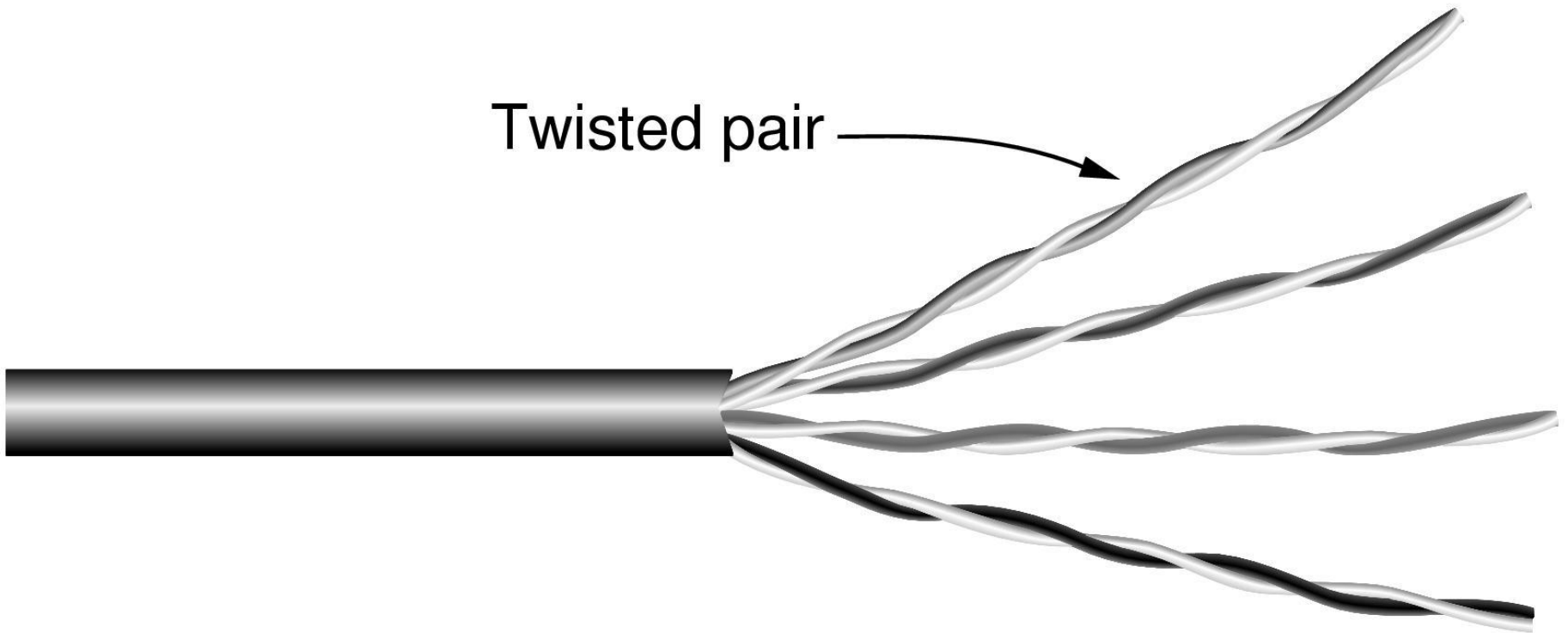
# Guided Transmission Media

- Magnetic media
- Twisted pairs
- Coaxial cable
- Power lines
- Fiber optics

# Magnetic Media

- Write data onto magnetic media
  - Disks
  - Tapes
- Data transmission speed
  - Never underestimate the bandwidth of a station wagon full of tapes hurtling down the highway.

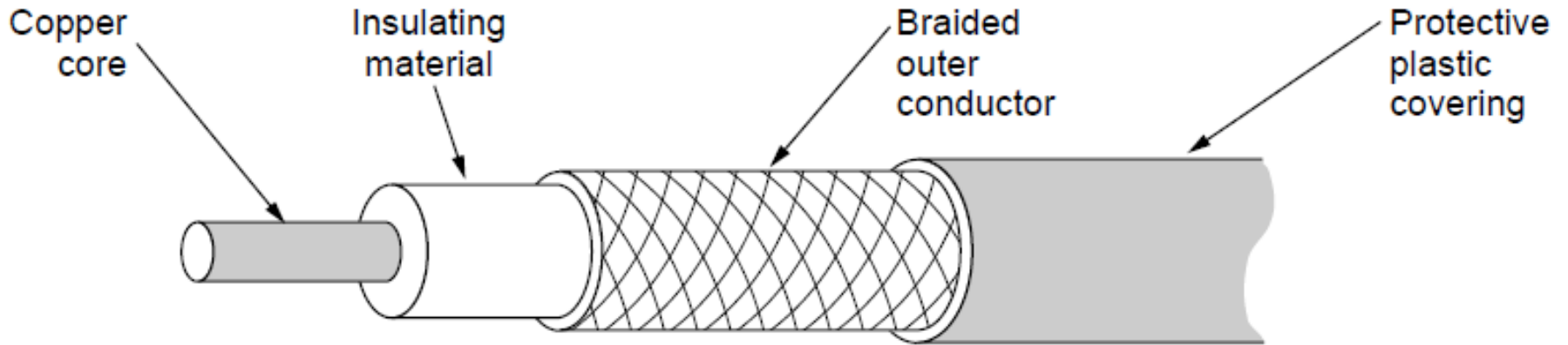
# Twisted Pairs



Twisted pair

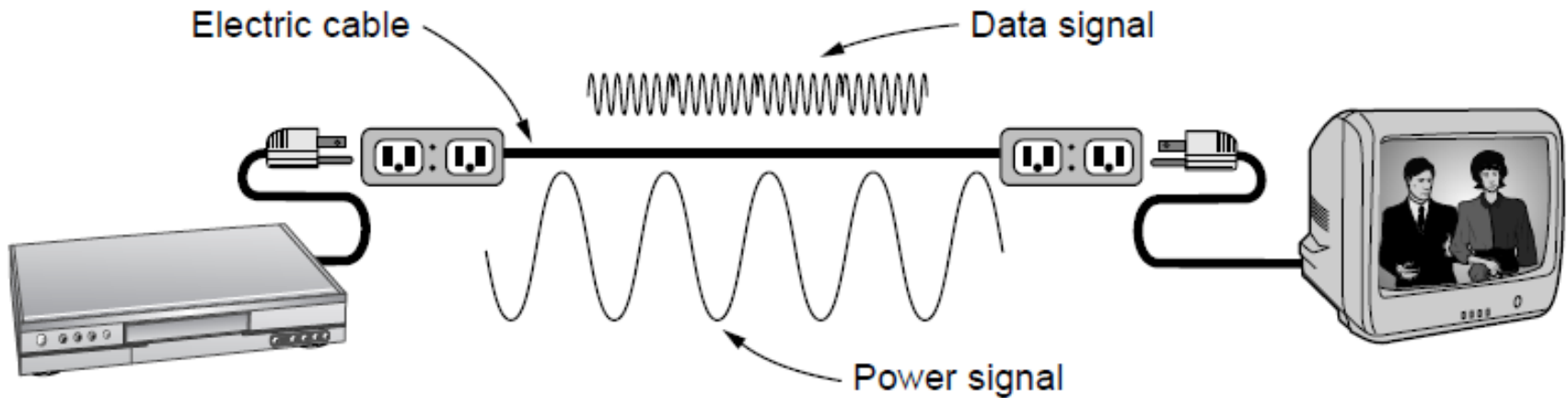
Category 5 UTP cable with four twisted pairs

# Coaxial Cable



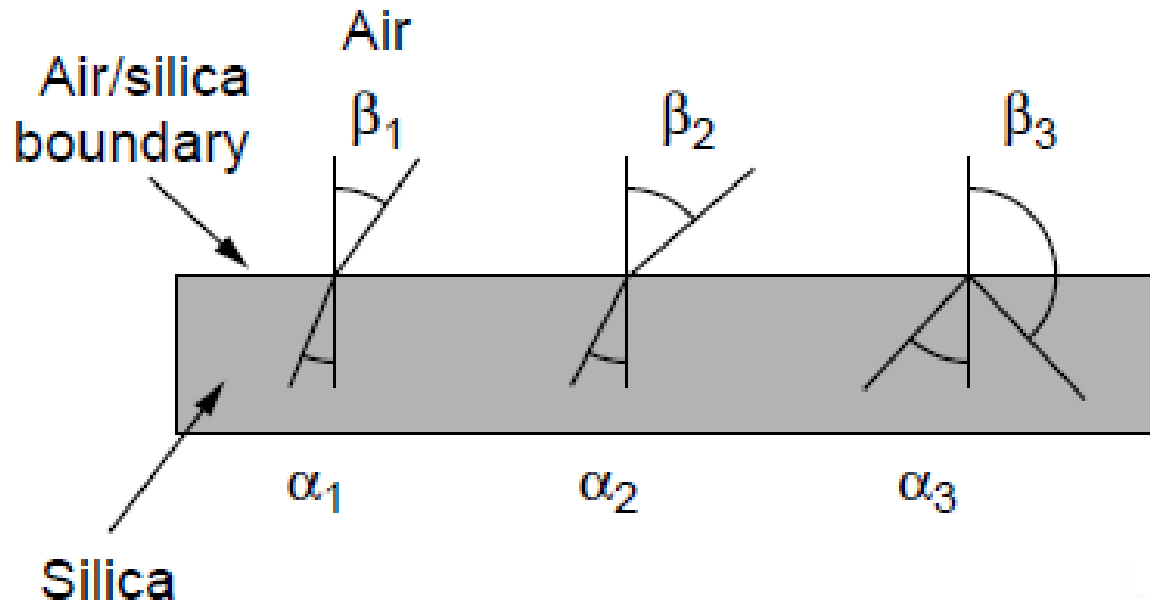
A coaxial cable

# Power Lines



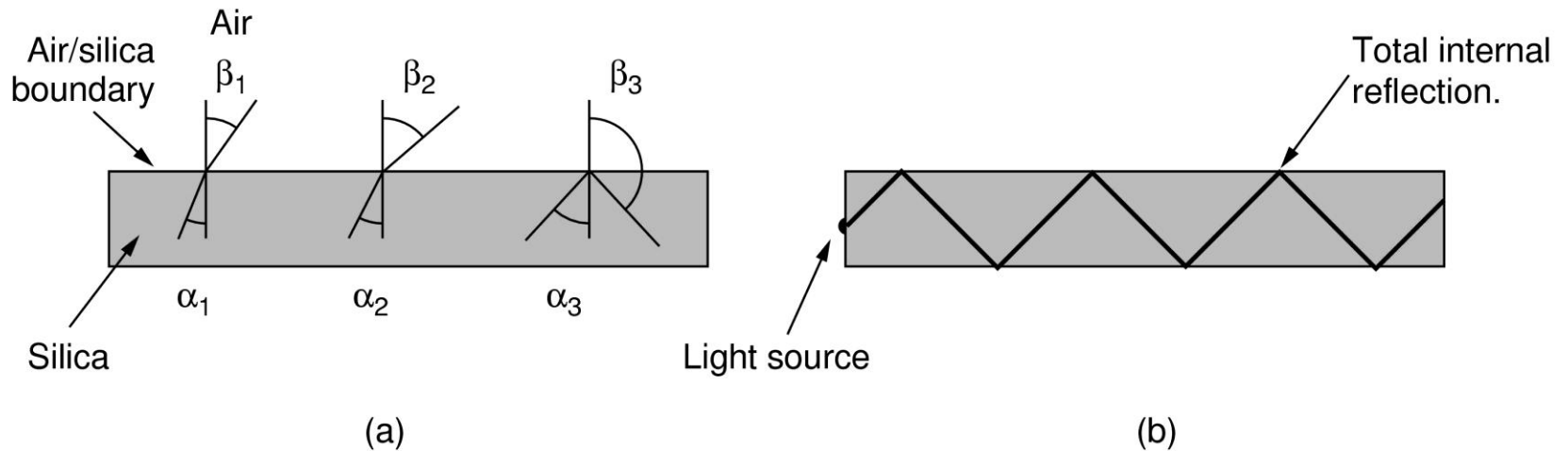
A network that uses household electrical wiring.

# Fiber Optics (1)



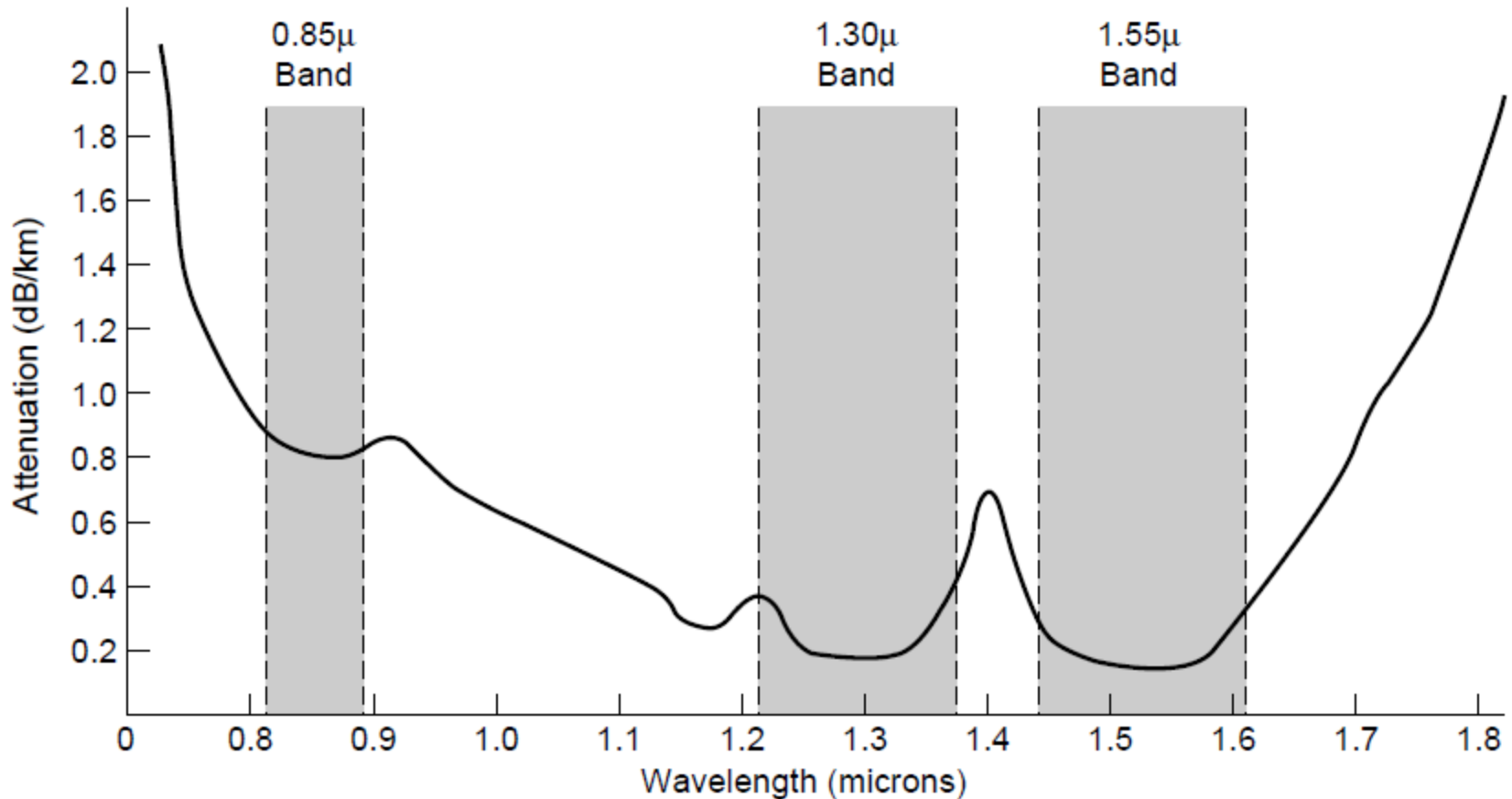
Three examples of a light ray from inside a silica fiber impinging on the air/silica boundary at different angles.

# Fiber Optics (2)



Light trapped by total internal reflection.

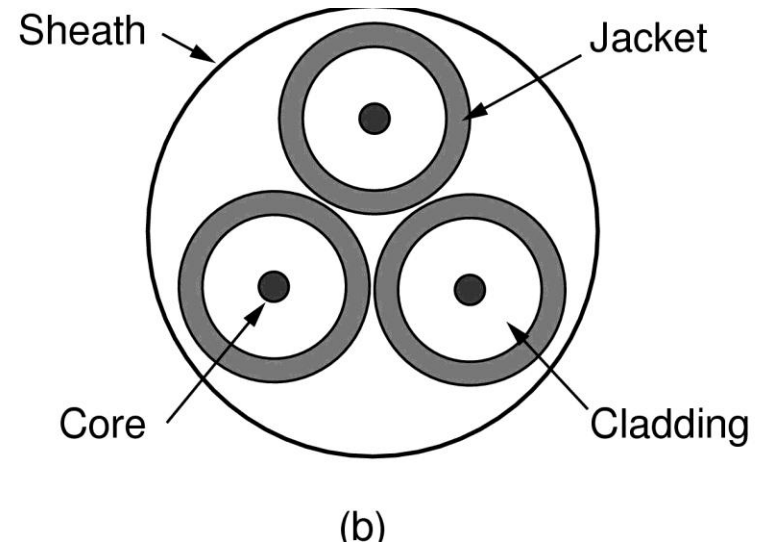
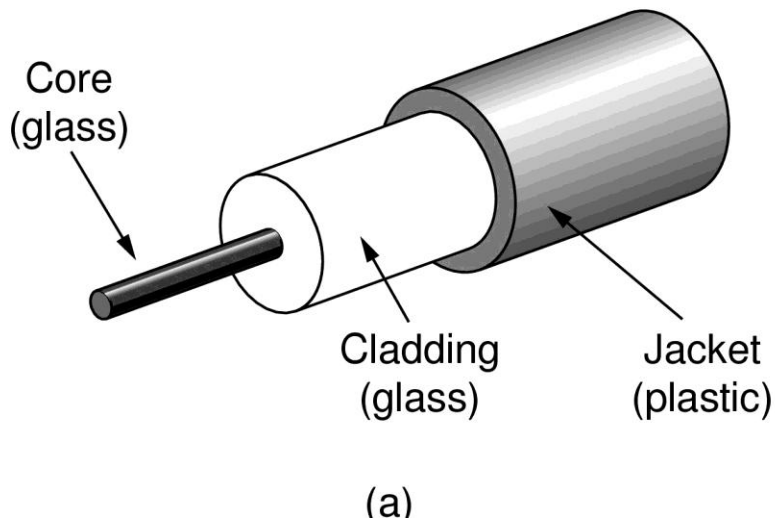
# Transmission of Light Through Fiber



Attenuation of light through fiber  
in the infrared region



# Fiber Cables (1)



Views of a fiber cable

# Fiber Cables (2)

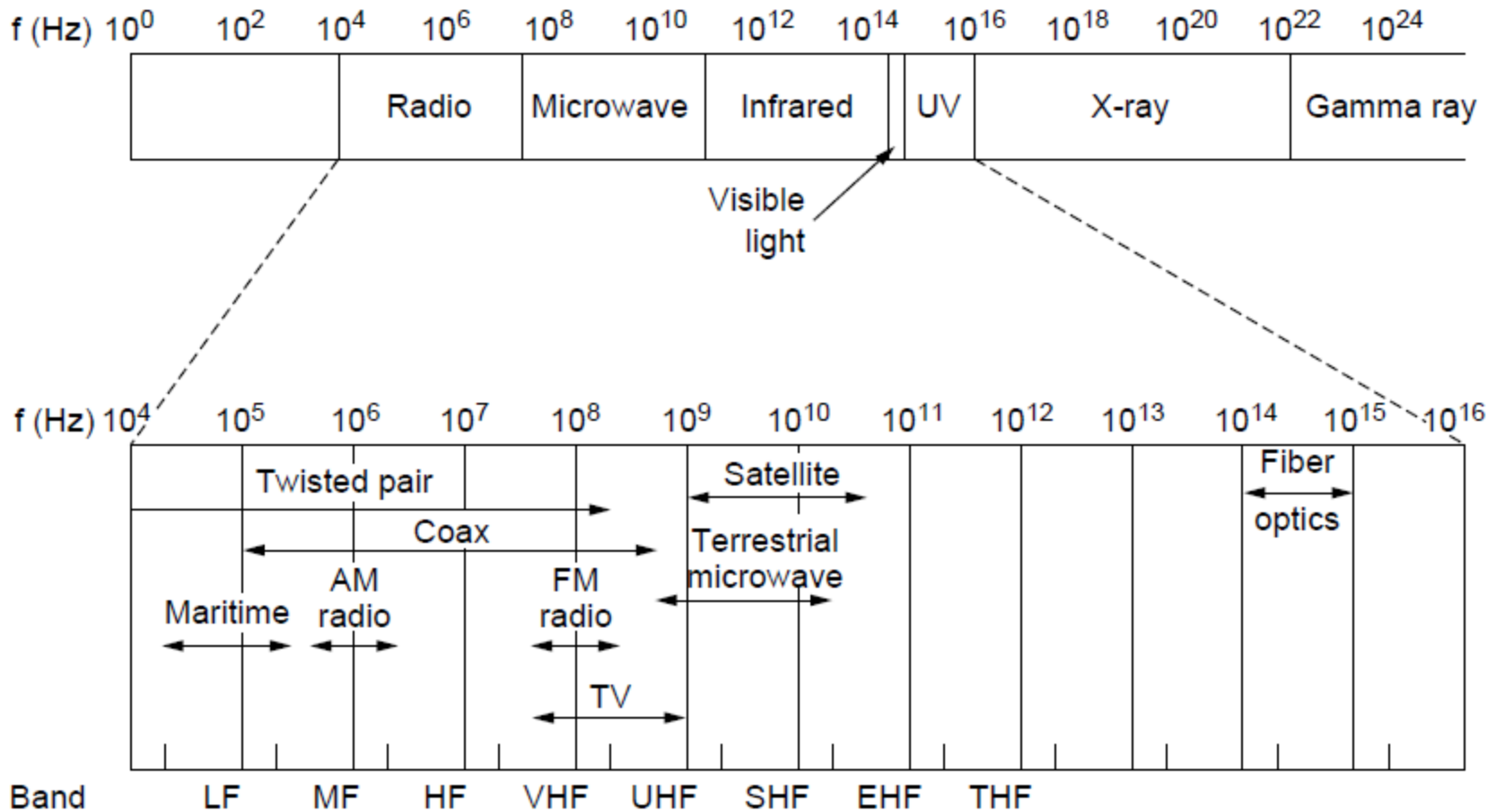
<b>Item</b>	<b>LED</b>	<b>Semiconductor laser</b>
Data rate	Low	High
Fiber type	Multi-mode	Multi-mode or single-mode
Distance	Short	Long
Lifetime	Long life	Short life
Temperature sensitivity	Minor	Substantial
Cost	Low cost	Expensive

A comparison of semiconductor diodes  
and LEDs as light sources

# Wireless Transmission

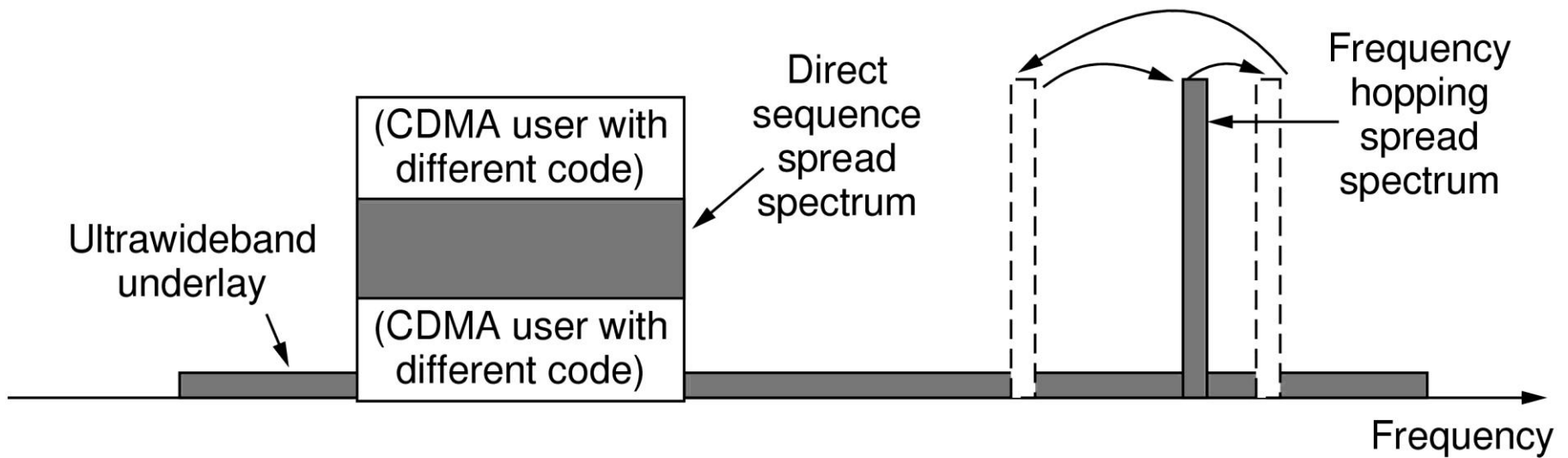
- The Electromagnetic Spectrum
- Radio Transmission
- Microwave Transmission
- Infrared Transmission
- Light Transmission

# The Electromagnetic Spectrum (1)



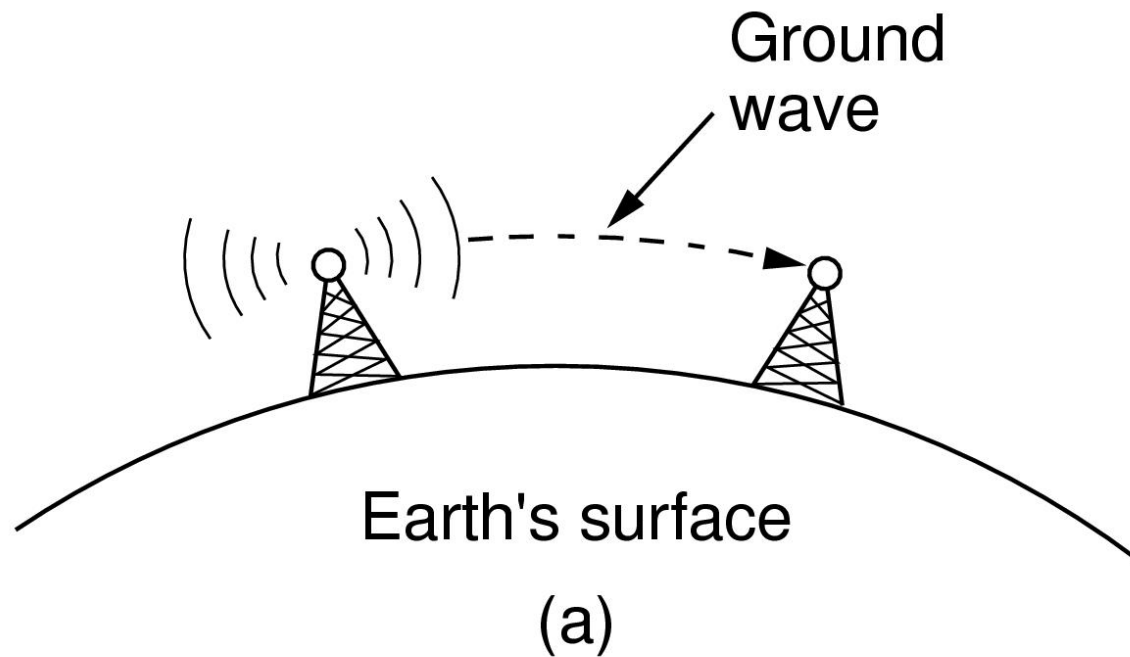
The electromagnetic spectrum and its uses for communication

# The Electromagnetic Spectrum (2)



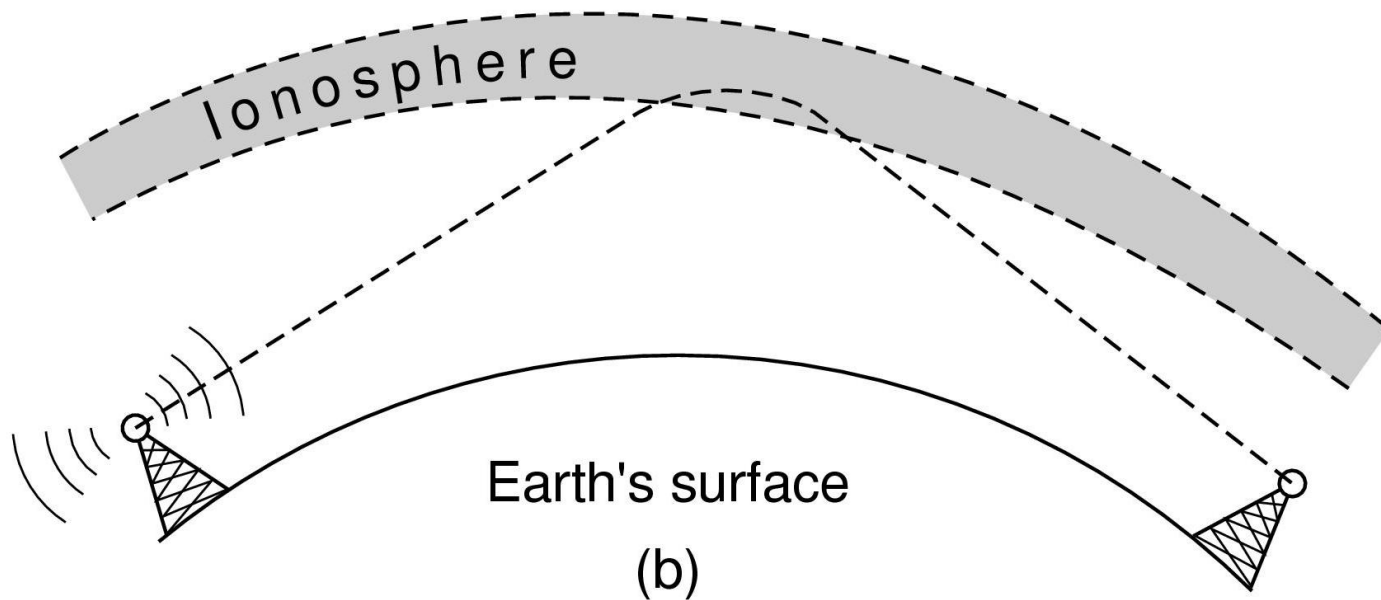
Spread spectrum and ultra-wideband (UWB) communication

# Radio Transmission (1)



In the VLF, LF, and MF bands, radio waves follow the curvature of the earth

# Radio Transmission (2)



In the HF band, they bounce off the ionosphere.

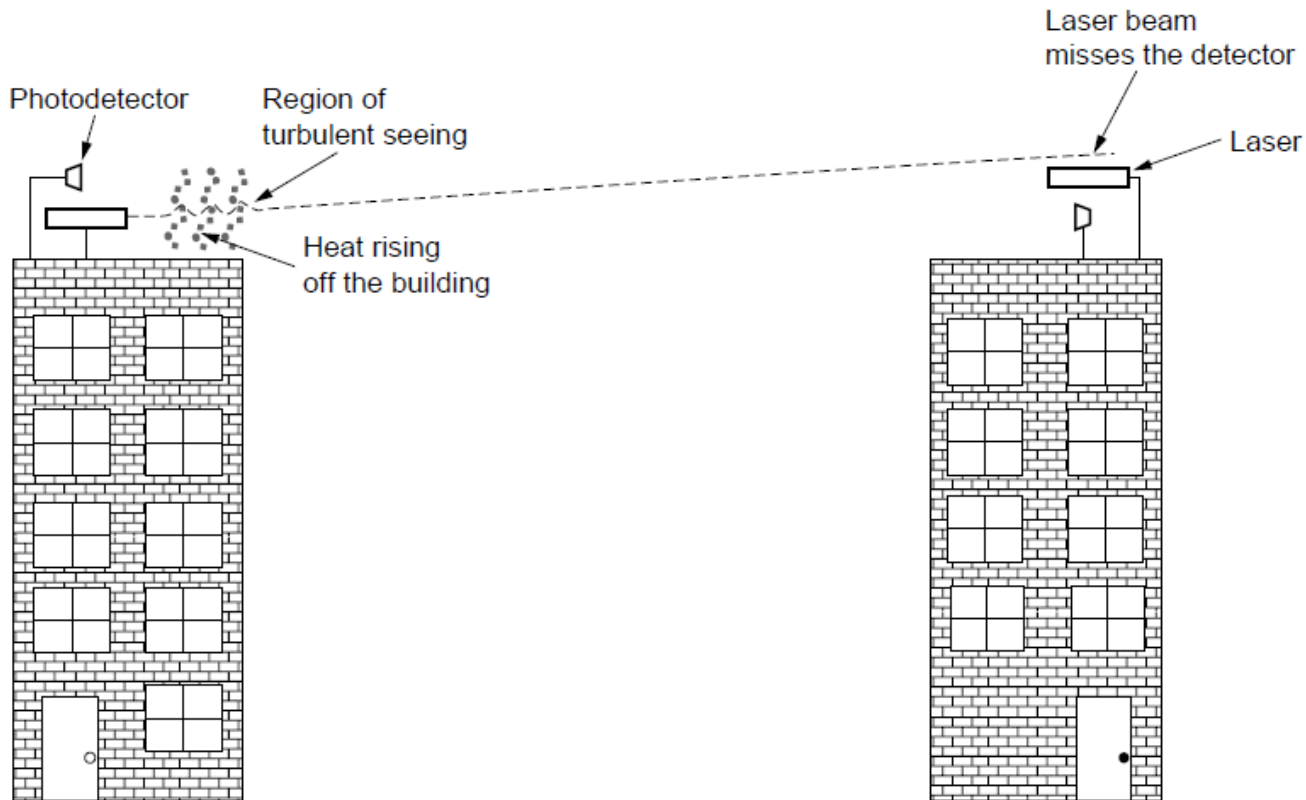
# The Politics of the Electromagnetic Spectrum



ISM and U-NII bands used in the United States by wireless devices



# Light Transmission

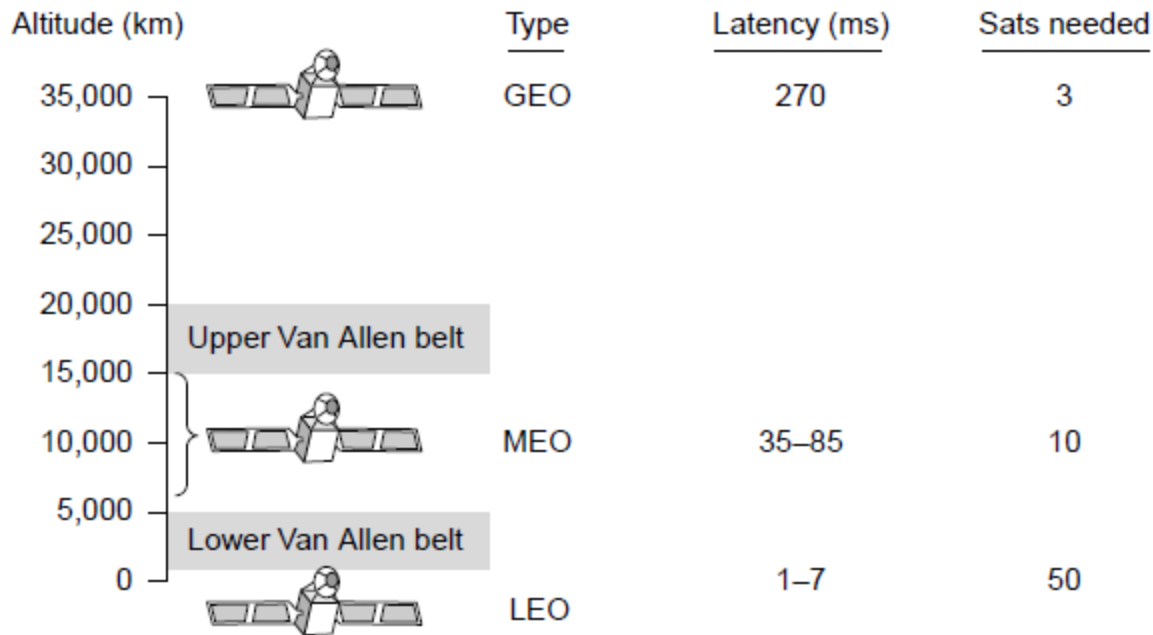


Convection currents can interfere with laser communication systems. A bidirectional system with two lasers is pictured here.

# Communication Satellites

- Geostationary Satellites
- Medium-Earth Orbit Satellites
- Low-Earth Orbit Satellites
- Satellites Versus Fiber

# Communication Satellites



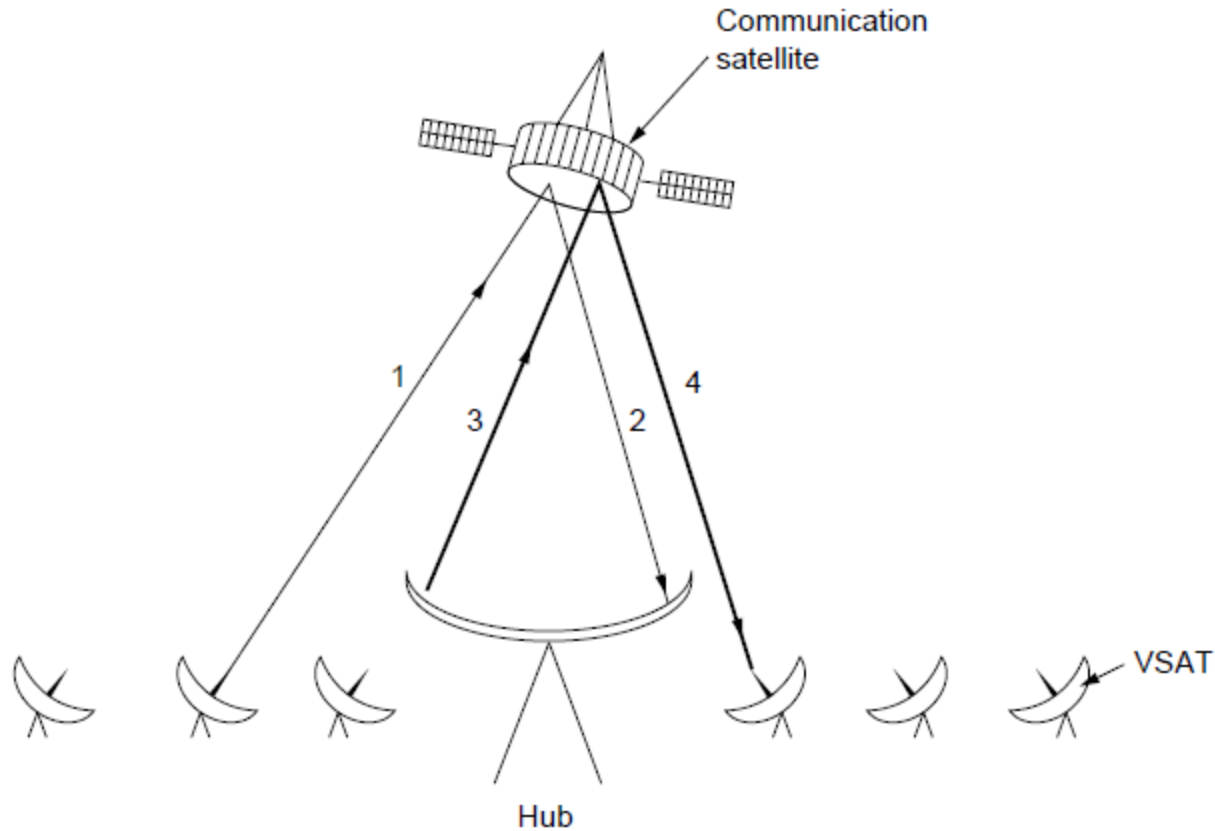
Communication satellites, some properties, including: altitude above earth, round-trip delay time, number of satellites for global coverage.

# Geostationary Satellites (1)

<b>Band</b>	<b>Downlink</b>	<b>Uplink</b>	<b>Bandwidth</b>	<b>Problems</b>
L	1.5 GHz	1.6 GHz	15 MHz	Low bandwidth; crowded
S	1.9 GHz	2.2 GHz	70 MHz	Low bandwidth; crowded
C	4.0 GHz	6.0 GHz	500 MHz	Terrestrial interference
Ku	11 GHz	14 GHz	500 MHz	Rain
Ka	20 GHz	30 GHz	3500 MHz	Rain, equipment cost

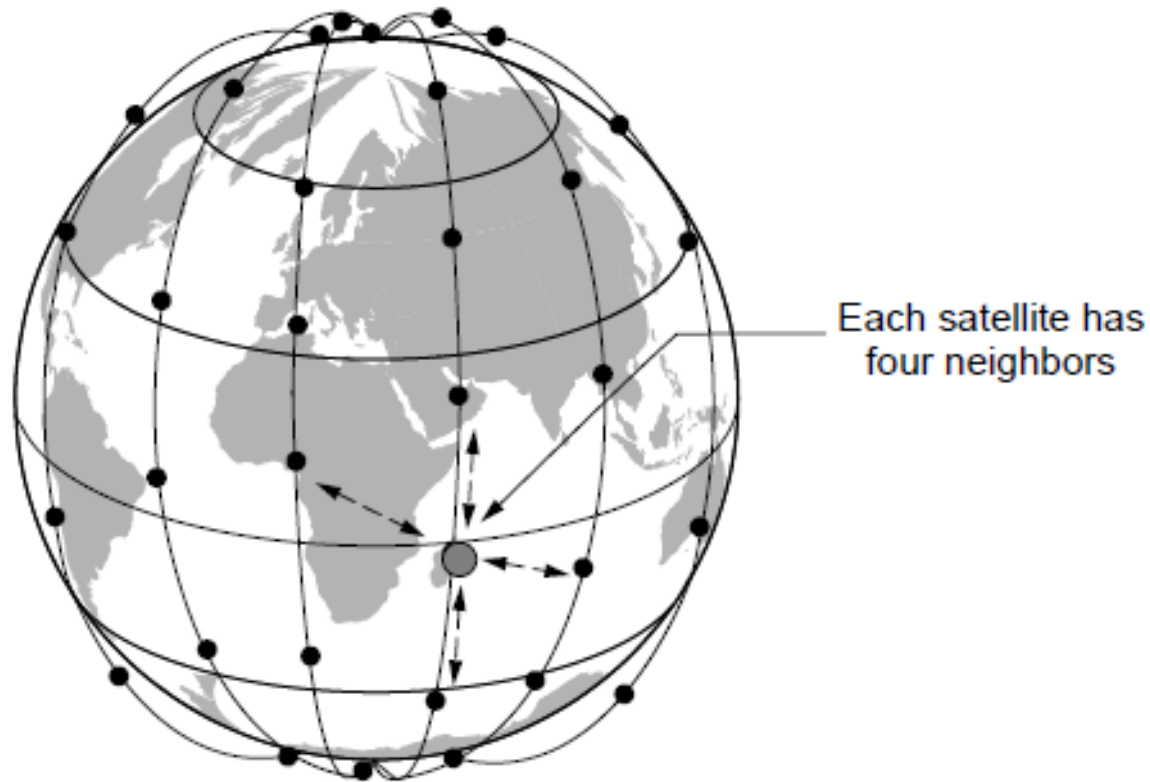
The principal satellite bands

# Geostationary Satellites (2)



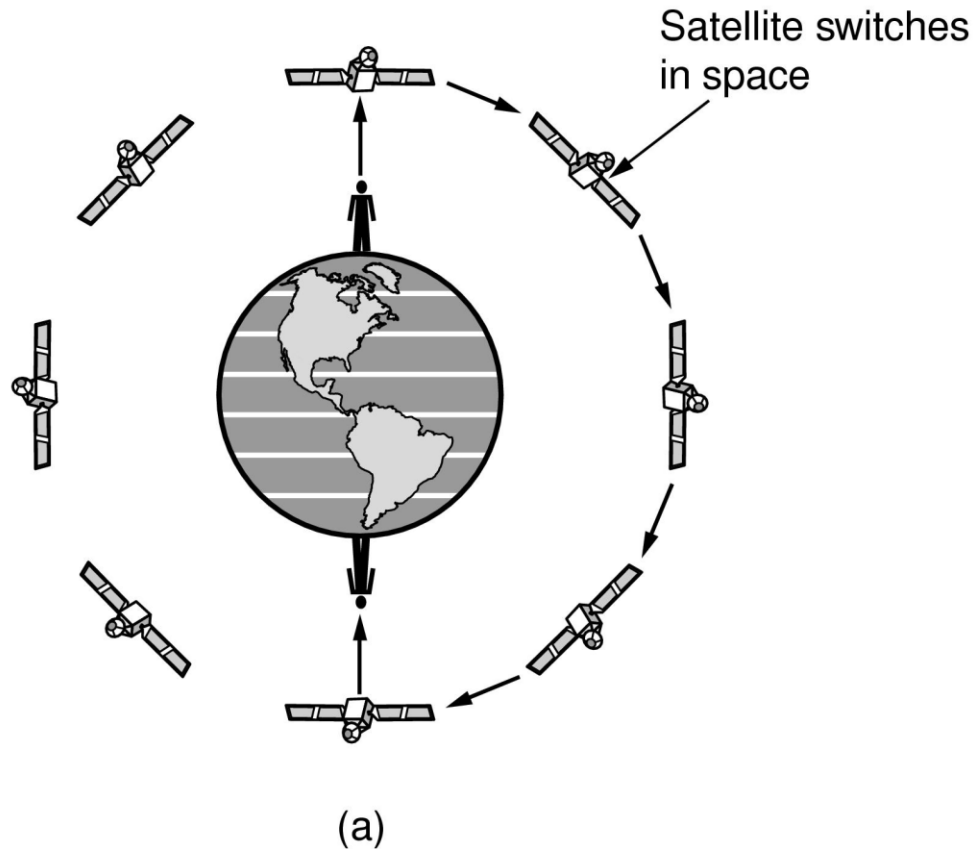
VSATs using a hub.

# Low-Earth Orbit Satellites (1)



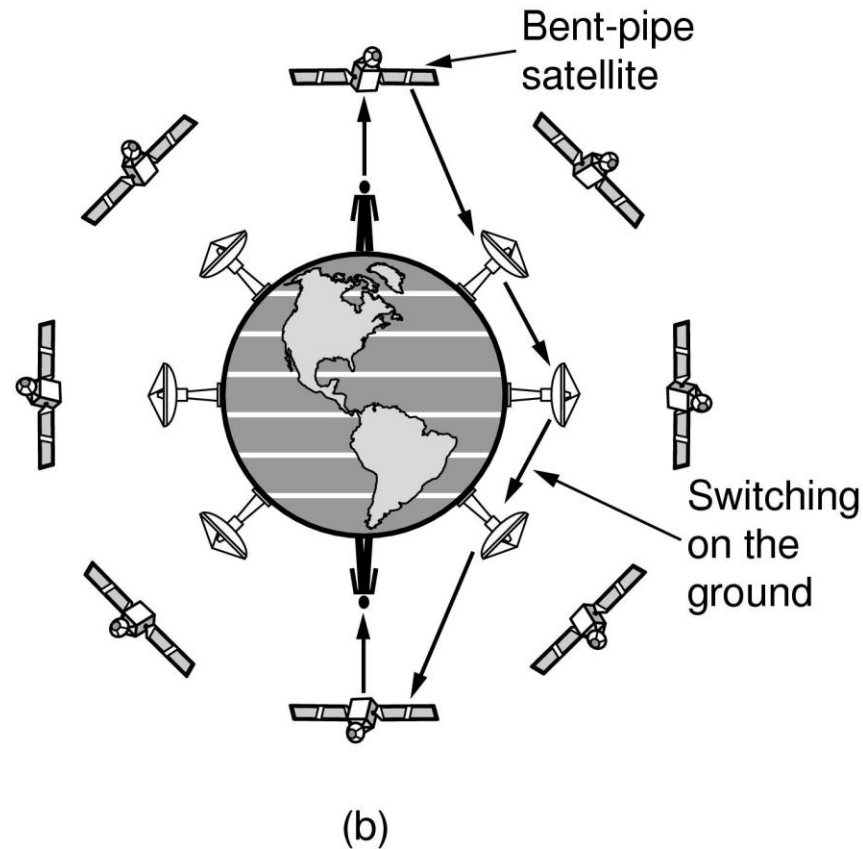
The Iridium satellites form six necklaces around the earth.

# Low-Earth Orbit Satellites (2)



Relaying in space.

# Low-Earth Orbit Satellites (3)



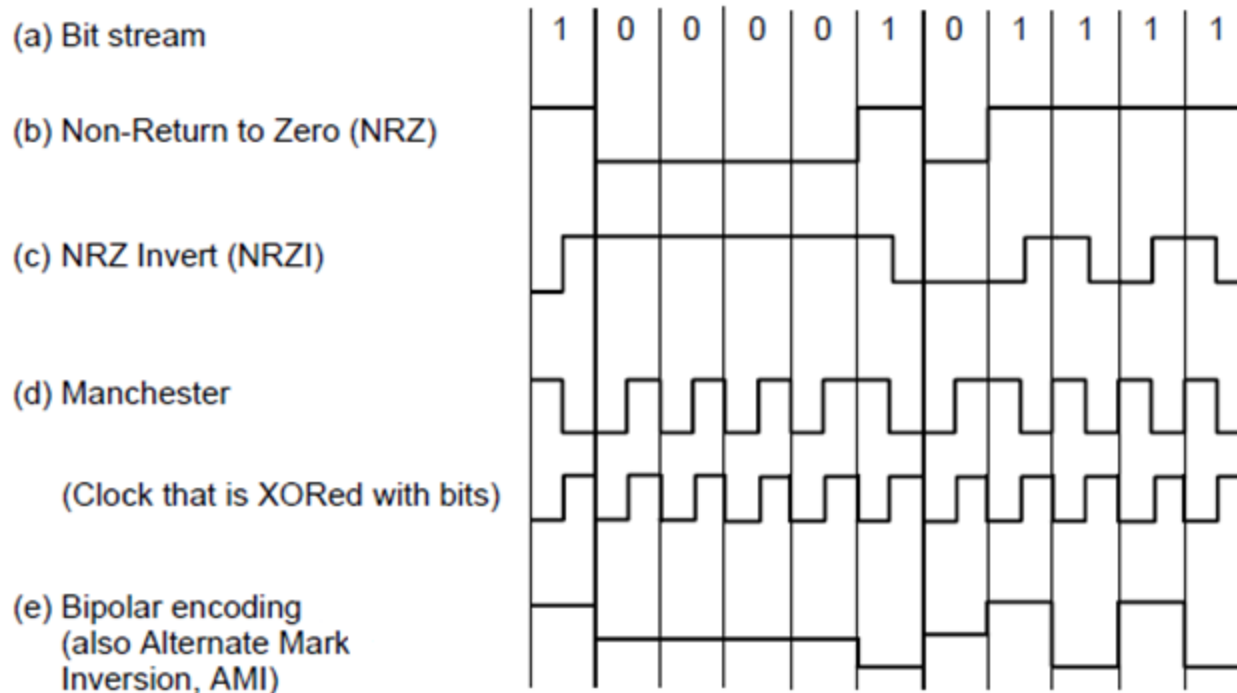
Relaying on the ground



# Digital Modulation and Multiplexing

- Baseband Transmission
- Passband Transmission
- Frequency Division Multiplexing
- Time Division Multiplexing
- Code Division Multiplexing

# Baseband Transmission



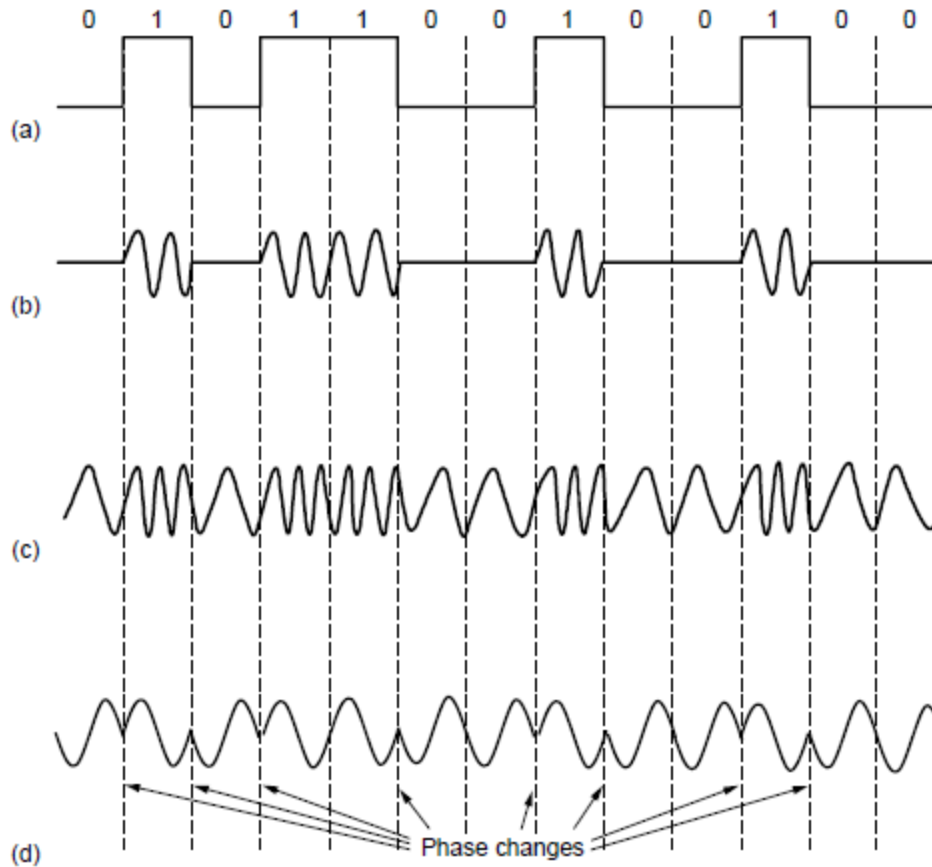
Line codes: (a) Bits, (b) NRZ, (c) NRZI,  
(d) Manchester, (e) Bipolar or AMI.

# Clock Recovery

<b>Data (4B)</b>	<b>Codeword (5B)</b>	<b>Data (4B)</b>	<b>Codeword (5B)</b>
0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

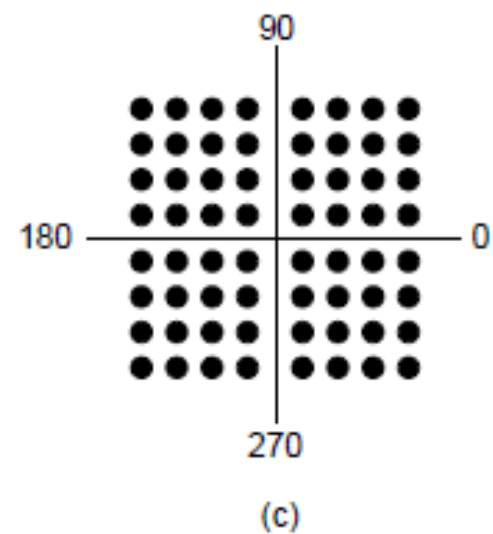
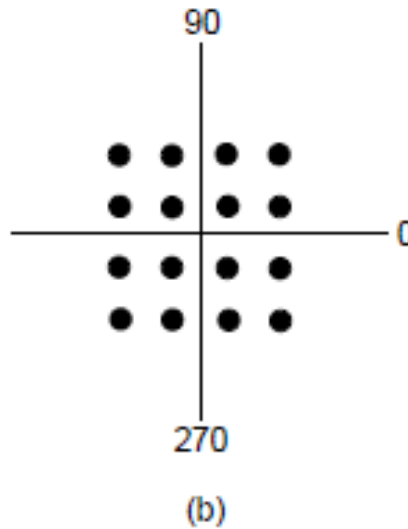
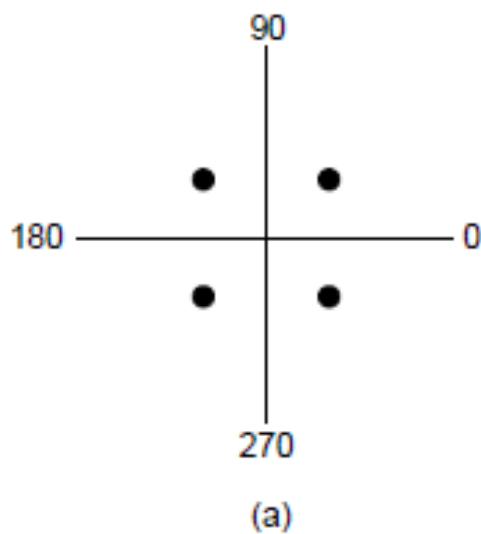
4B/5B mapping.

# Passband Transmission (1)



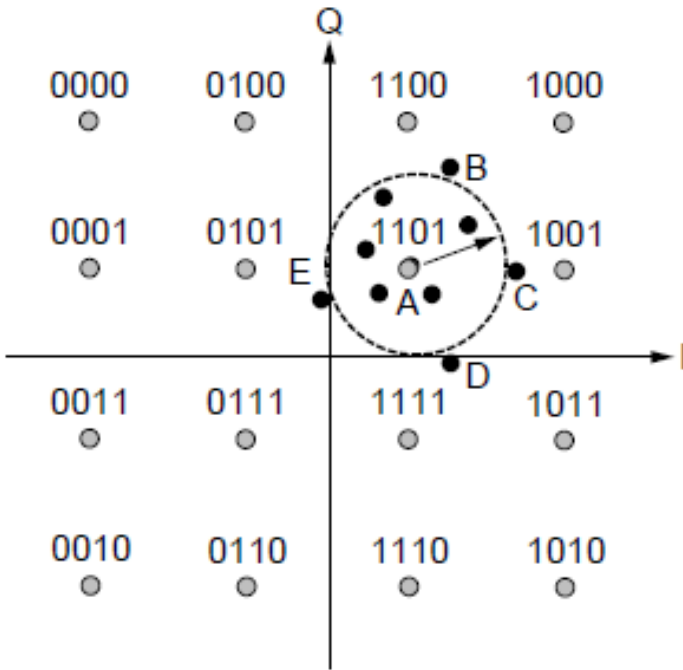
- (a) A binary signal. (b) Amplitude shift keying.  
(c) Frequency shift keying. (d) Phase shift keying.

# Passband Transmission (2)



(a) QPSK. (b) QAM-16. (c) QAM-64.

# Frequency Division Multiplexing (1)

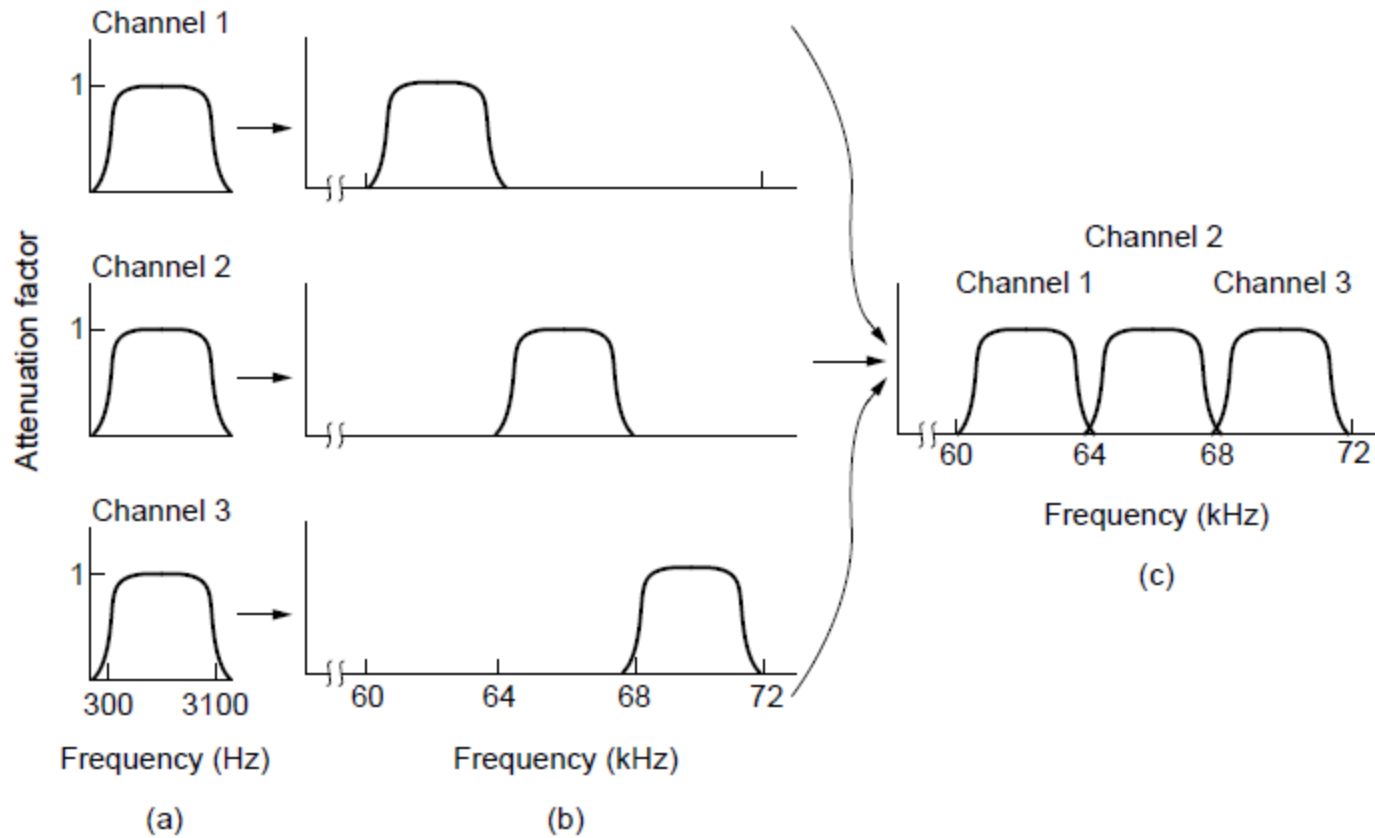


When 1101 is sent:

Point	Decodes as	Bit errors
A	1101	0
B	11 <u>0</u> 0	1
C	<u>1</u> 001	1
D	11 <u>1</u> 1	1
E	<u>0</u> 101	1

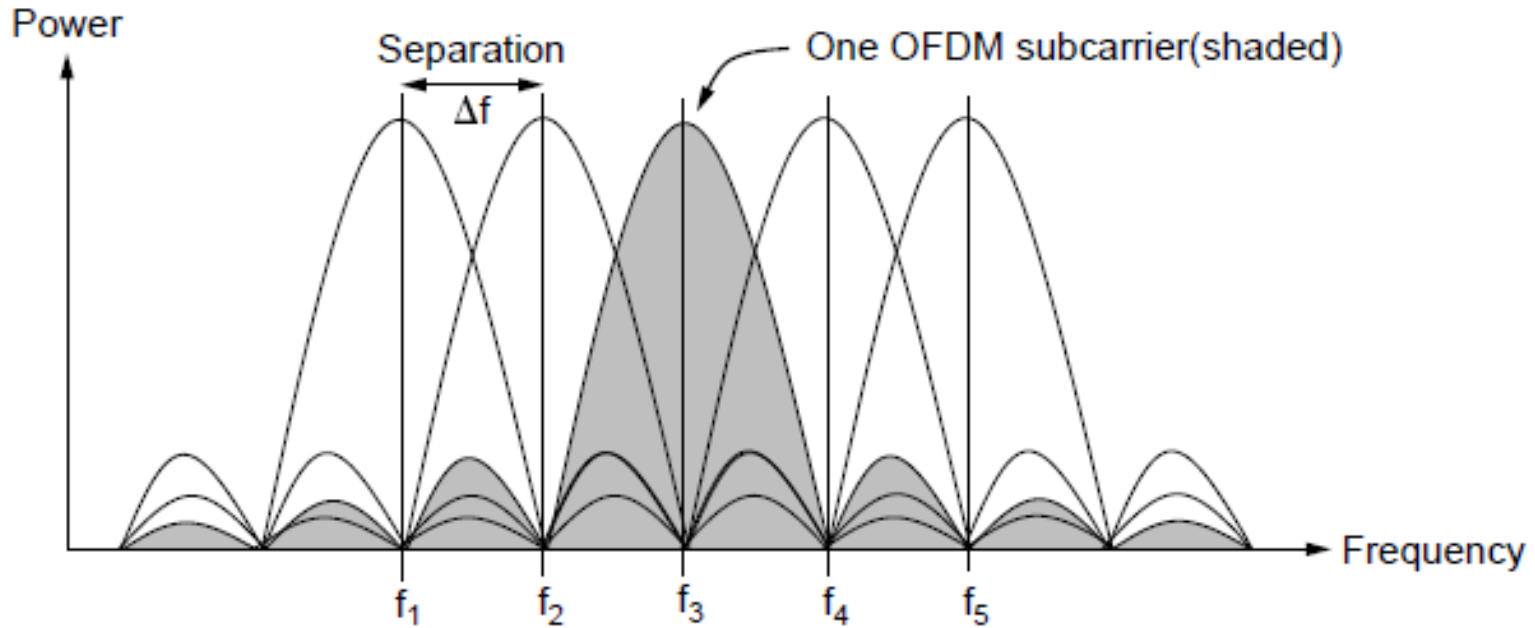
Gray-coded QAM-16.

# Frequency Division Multiplexing (2)



Frequency division multiplexing. (a) The original bandwidths.  
(b) The bandwidths raised in frequency.  
(c) The multiplexed channel.

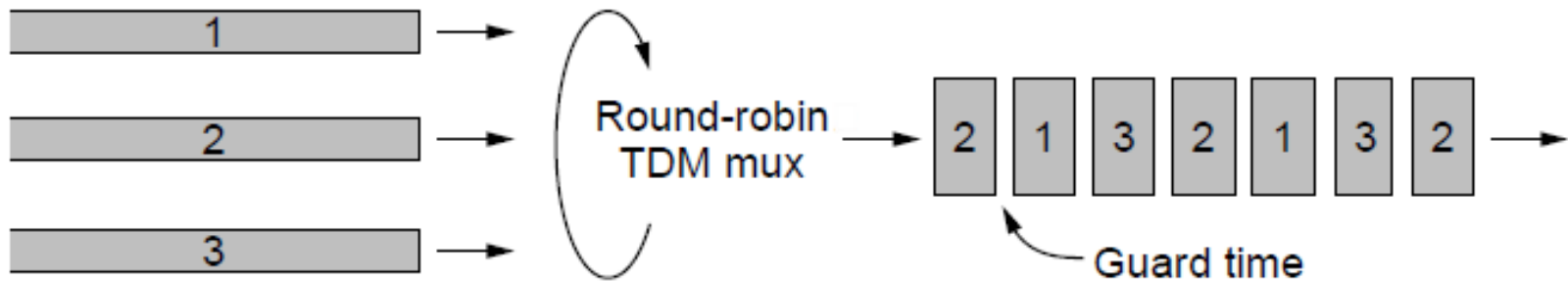
# Frequency Division Multiplexing (3)



Orthogonal frequency division multiplexing (OFDM).



# Time Division Multiplexing



Time Division Multiplexing (TDM).

# Code Division Multiplexing (1)

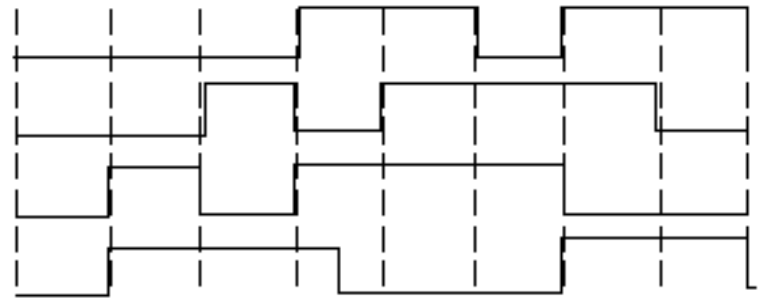
$$A = (-1 \ -1 \ -1 \ +1 \ +1 \ -1 \ +1 \ +1)$$

$$B = (-1 \ -1 \ +1 \ -1 \ +1 \ +1 \ +1 \ -1)$$

$$C = (-1 \ +1 \ -1 \ +1 \ +1 \ +1 \ -1 \ -1)$$

$$D = (-1 \ +1 \ -1 \ -1 \ -1 \ -1 \ +1 \ -1)$$

(a)



(b)

(a) Chip sequences for four stations.

(b) Signals the sequences represent

# Code Division Multiplexing (2)

$S_1 = C$	$= (-1 +1 -1 +1 +1 +1 -1 -1)$	$S_1 \bullet C = [1+1-1+1+1+1-1-1]/8 = 1$
$S_2 = B+\underline{C}$	$= (-2 0 0 0 +2 +2 0 -2)$	$S_2 \bullet C = [2+0+0+0+2+2+0+2]/8 = 1$
$S_3 = A+\underline{B}$	$= ( 0 0 -2 +2 0 -2 0 +2)$	$S_3 \bullet C = [0+0+2+2+0-2+0-2]/8 = 0$
$S_4 = A+\underline{B}+C$	$= (-1 +1 -3 +3 +1 -1 -1 +1)$	$S_4 \bullet C = [1+1+3+3+1-1+1-1]/8 = 1$
$S_5 = A+B+\underline{C}+D$	$= (-4 0 -2 0 +2 0 +2 -2)$	$S_5 \bullet C = [4+0+2+0+2+0-2+2]/8 = 1$
$S_6 = A+B+\underline{C}+D$	$= (-2 -2 0 -2 0 -2 +4 0)$	$S_6 \bullet C = [2-2+0-2+0-2-4+0]/8 = -1$

(c) (d)

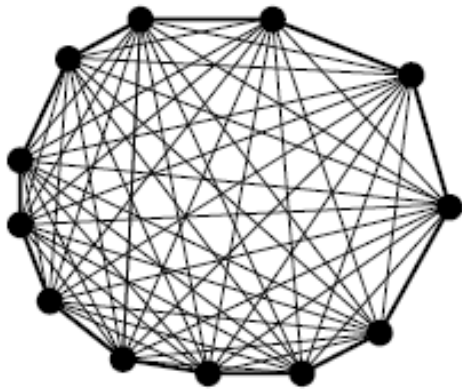
(a) Six examples of transmissions.

(b) Recovery of station C's

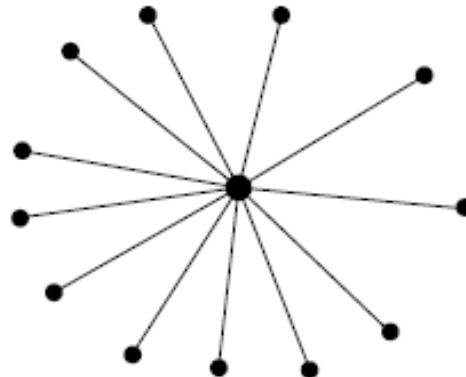
# The Public Switched Telephone Network

- Structure of the telephone system
- Politics of telephones
- Local loop: modems, ADSL, and fiber
- Trunks and multiplexing
- Switching

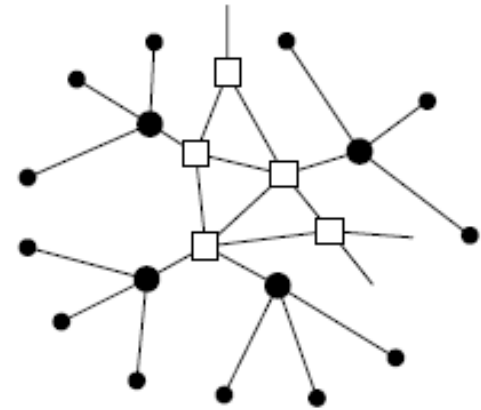
# Structure of the Telephone System (1)



(a)



(b)



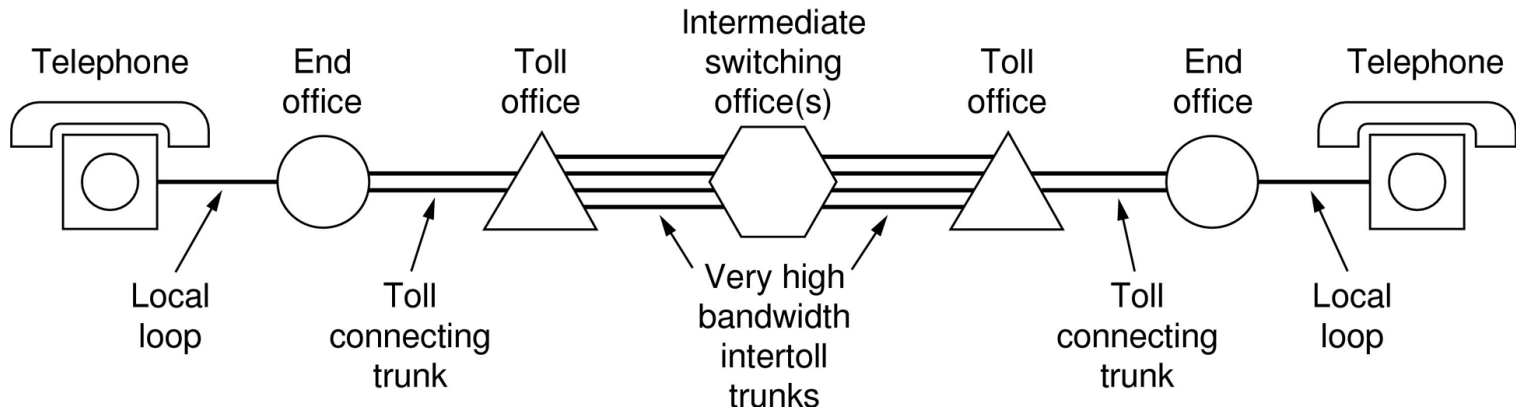
(c)

(a) Fully interconnected network.

(b) Centralized switch.

(c) Two-level hierarchy.

# Structure of the Telephone System (2)



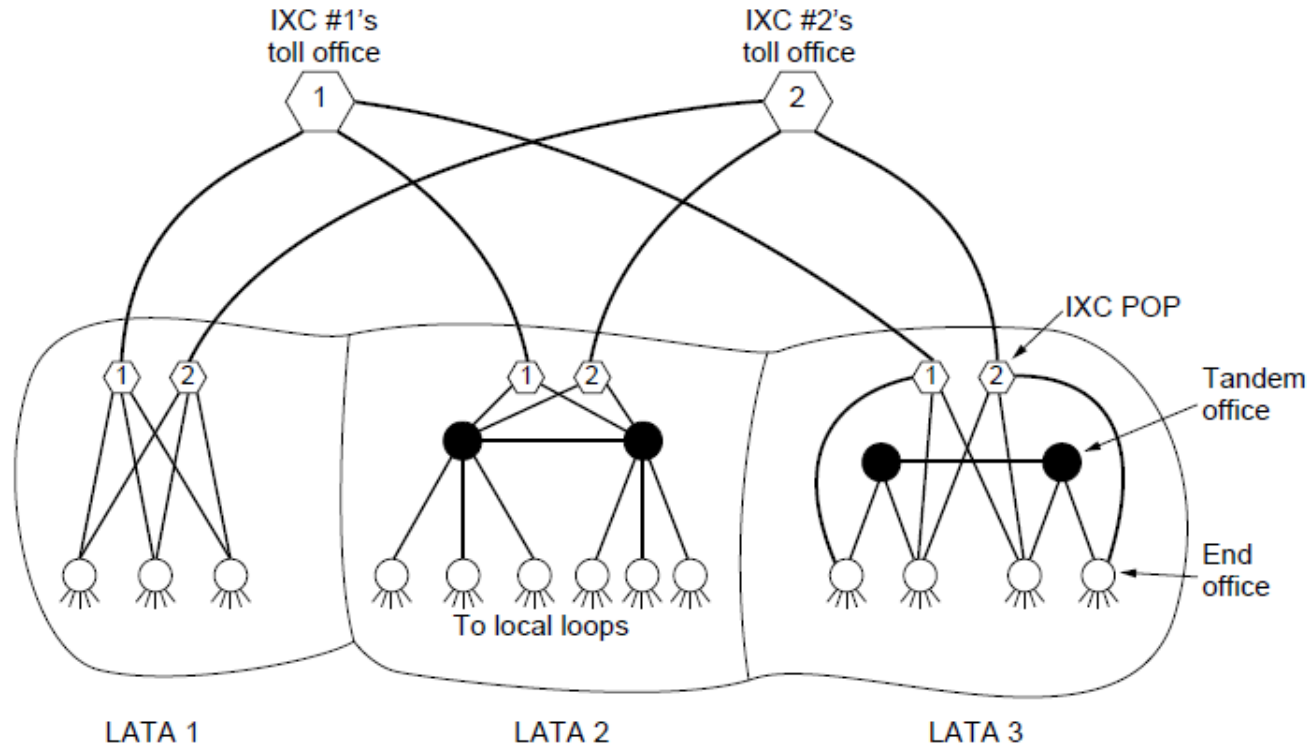
A typical circuit route for a long-distance call.

# Structure of the Telephone System (3)

## Major Components

1. Local loops analog twisted pairs to houses, businesses).
2. Trunks (digital fiber optic links between switching offices).
3. Switching offices (calls are moved from one trunk to another).

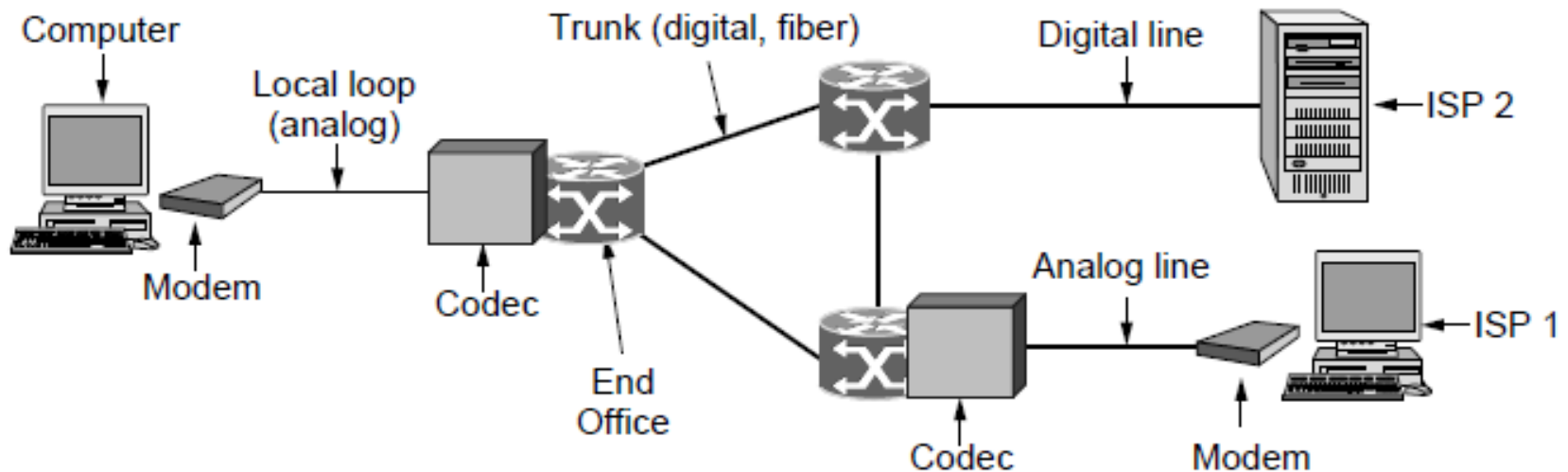
# The Politics of Telephones



The relationship of LATAs, LECs, and IXCs. Circles are LEC switching offices. Hexagons belong to IXC whose number is in it.

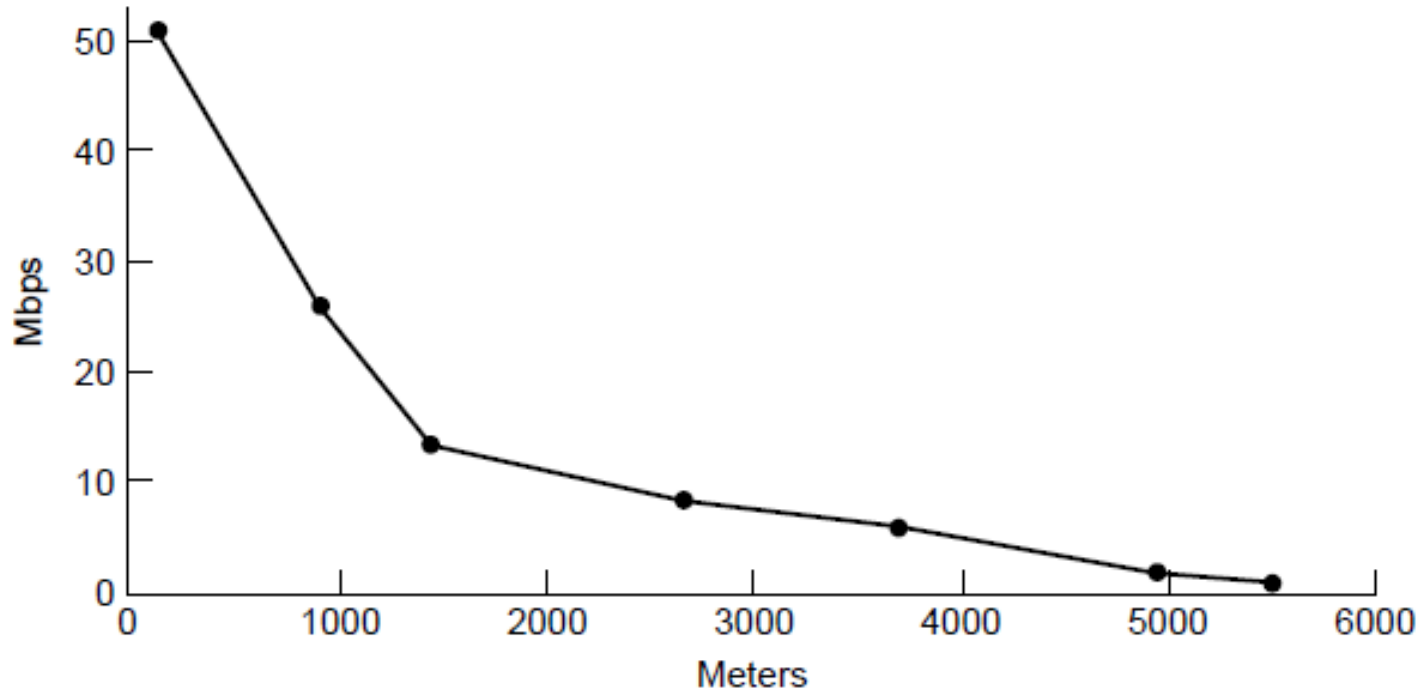


# Telephone Modems



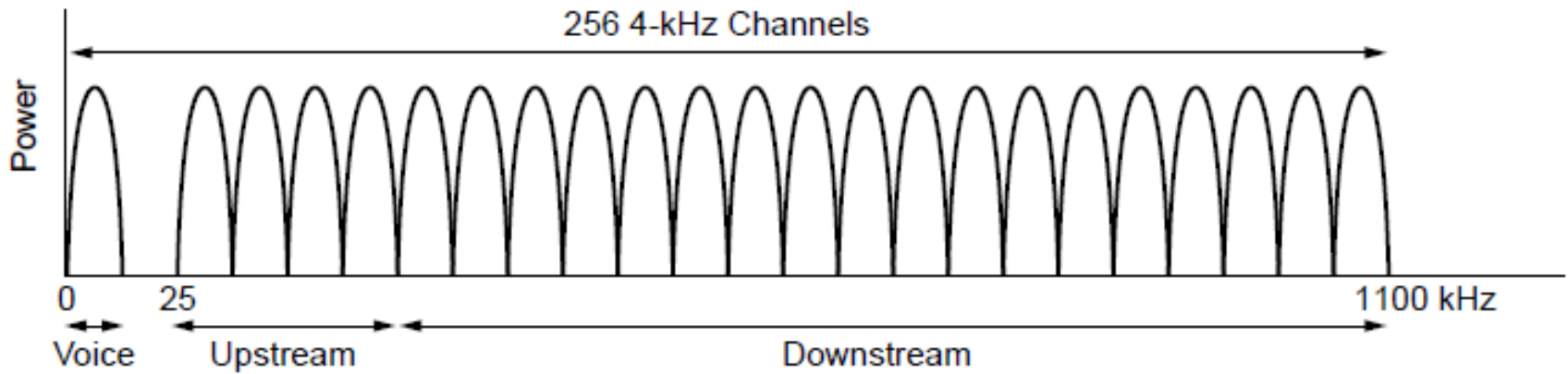
Use of both analog and digital transmission for computer-to-computer call. Conversion done by modems and codecs.

# Digital Subscriber Lines (1)



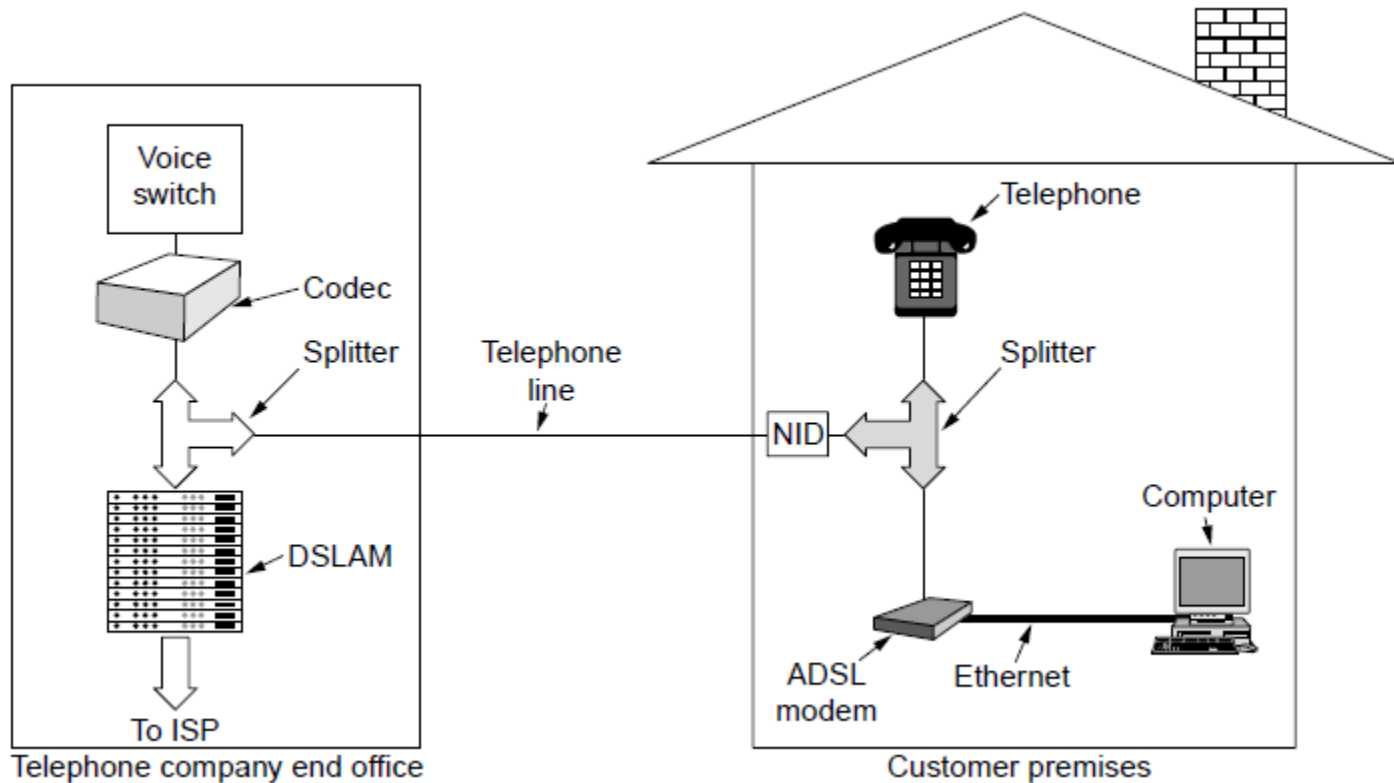
Bandwidth versus distance over Category 3  
UTP for DSL.

# Digital Subscriber Lines (2)



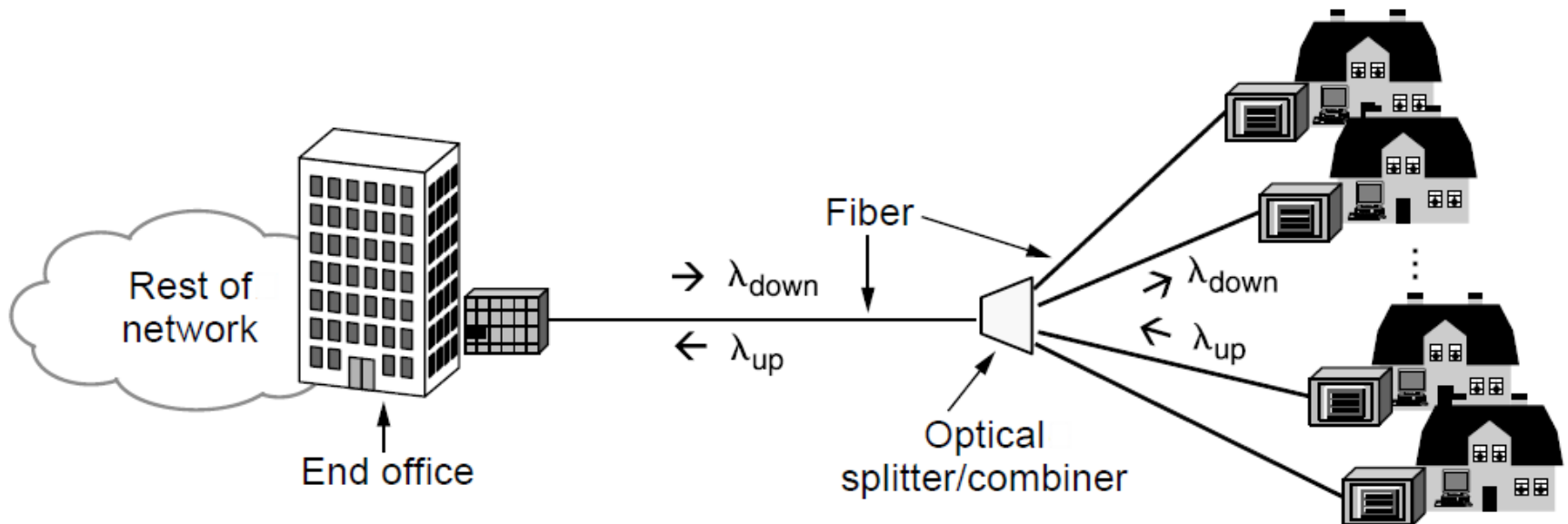
Operation of ADSL using discrete  
multitone modulation.

# Digital Subscriber Lines (3)



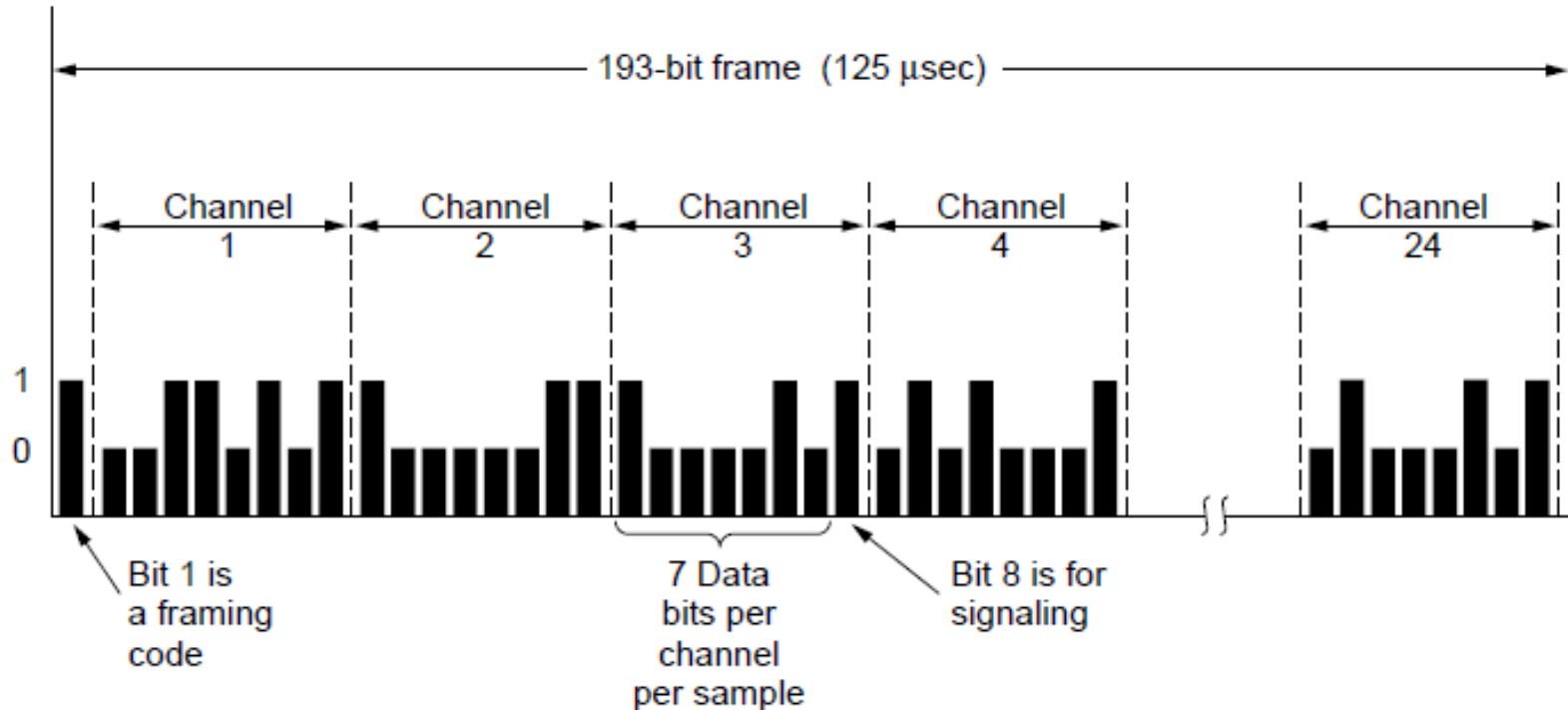
A typical ADSL equipment configuration.

# Fiber To The Home



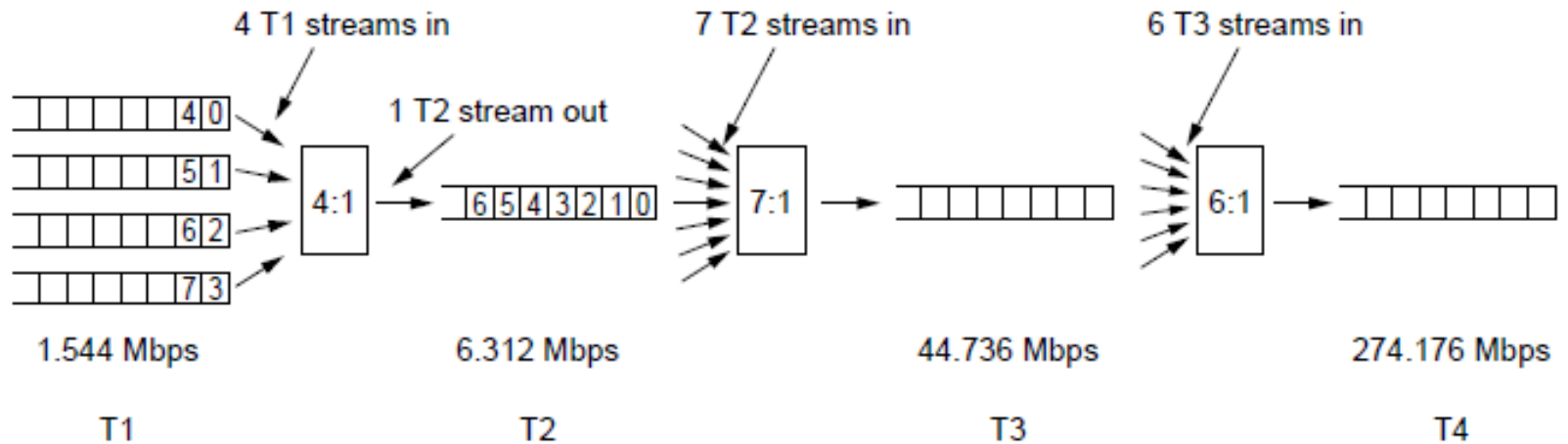
Passive optical network for Fiber To The Home.

# Time Division Multiplexing (1)



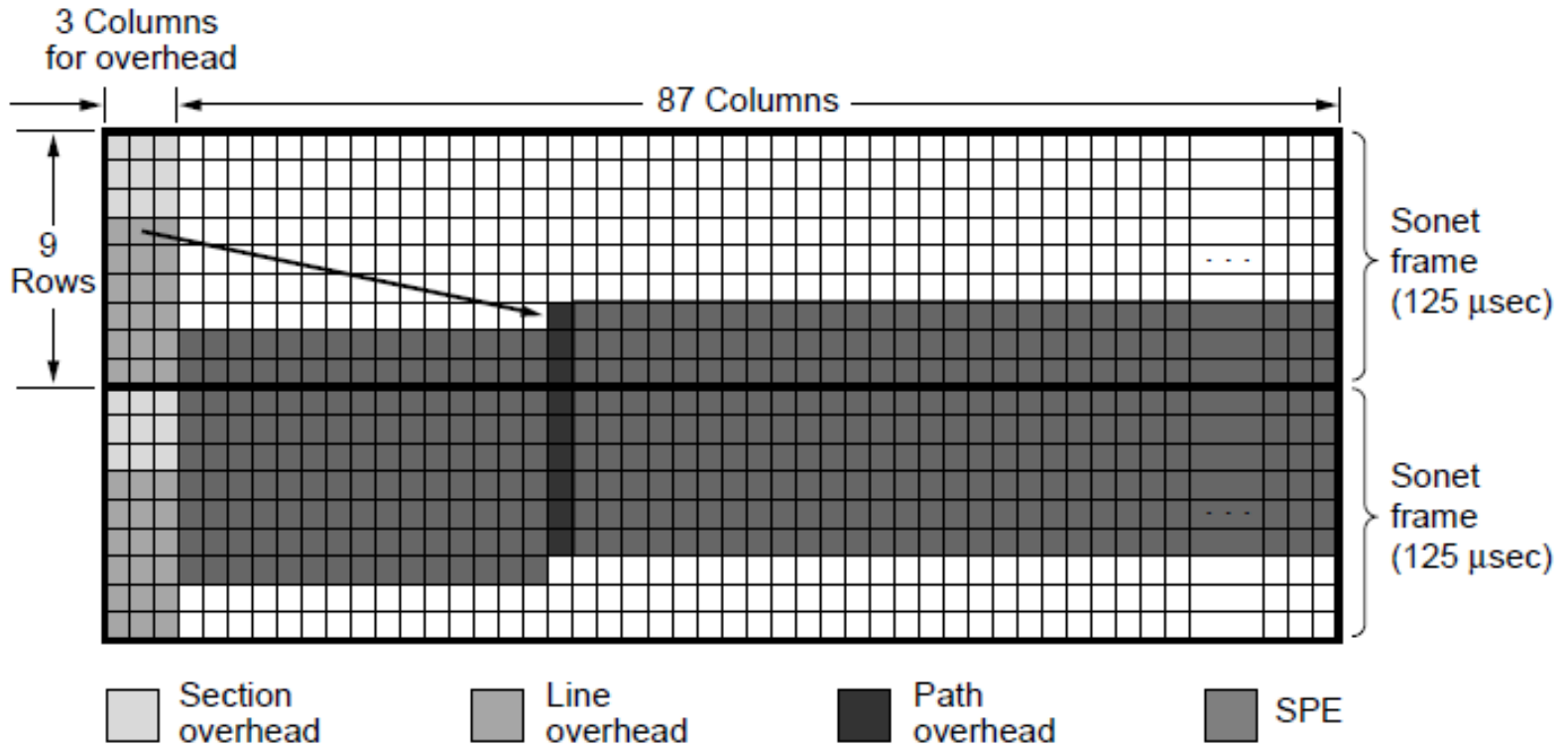
The T1 carrier (1.544 Mbps).

# Time Division Multiplexing (2)



Multiplexing T1 streams into higher carriers

# SONET/SDH (1)



Two back-to-back SONET frames.

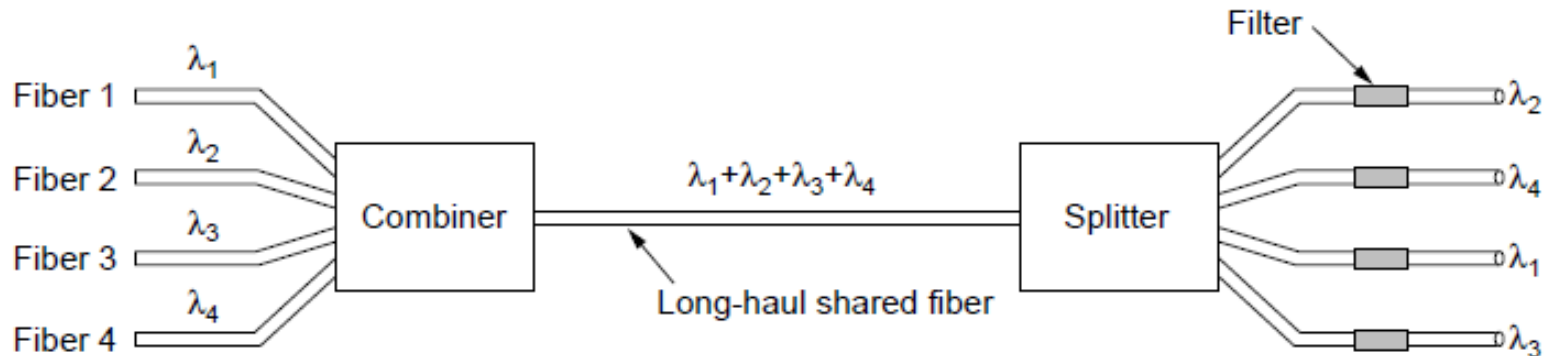
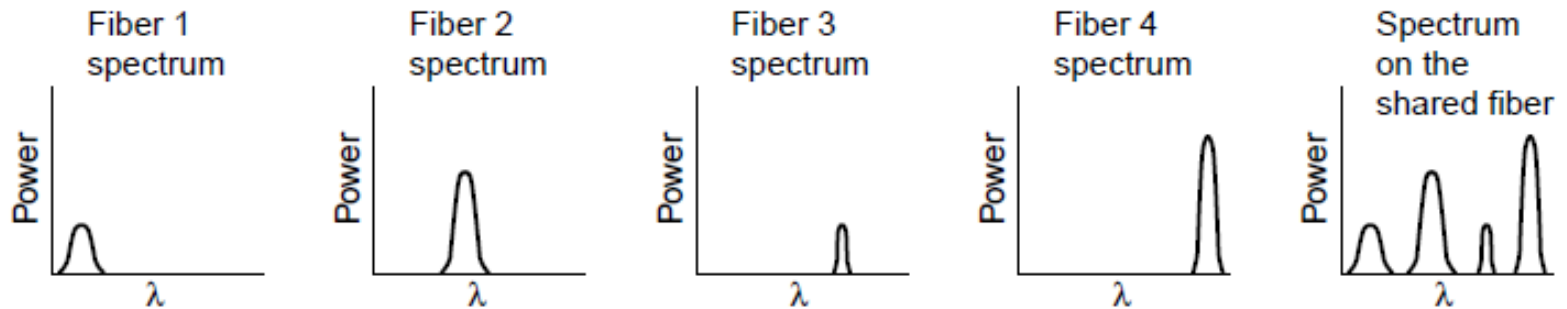


# SONET/SDH (2)

SONET		SDH	Data rate (Mbps)		
Electrical	Optical	Optical	Gross	SPE	User
STS-1	OC-1		51.84	50.112	49.536
STS-3	OC-3	STM-1	155.52	150.336	148.608
STS-12	OC-12	STM-4	622.08	601.344	594.432
STS-48	OC-48	STM-16	2488.32	2405.376	2377.728
STS-192	OC-192	STM-64	9953.28	9621.504	9510.912
STS-768	OC-768	STM-256	39813.12	38486.016	38043.648

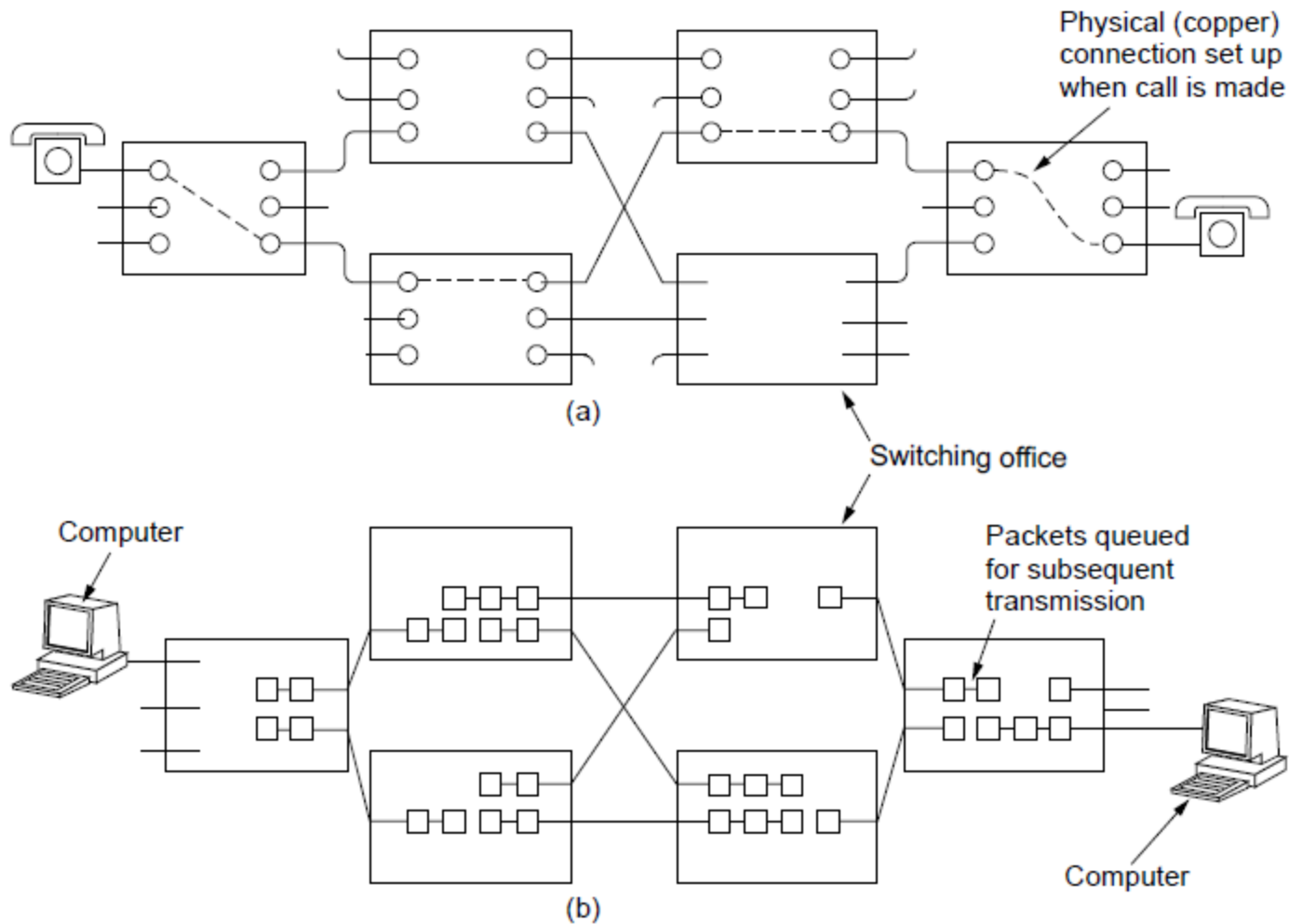
SONET and SDH multiplex rates.

# Wavelength Division Multiplexing



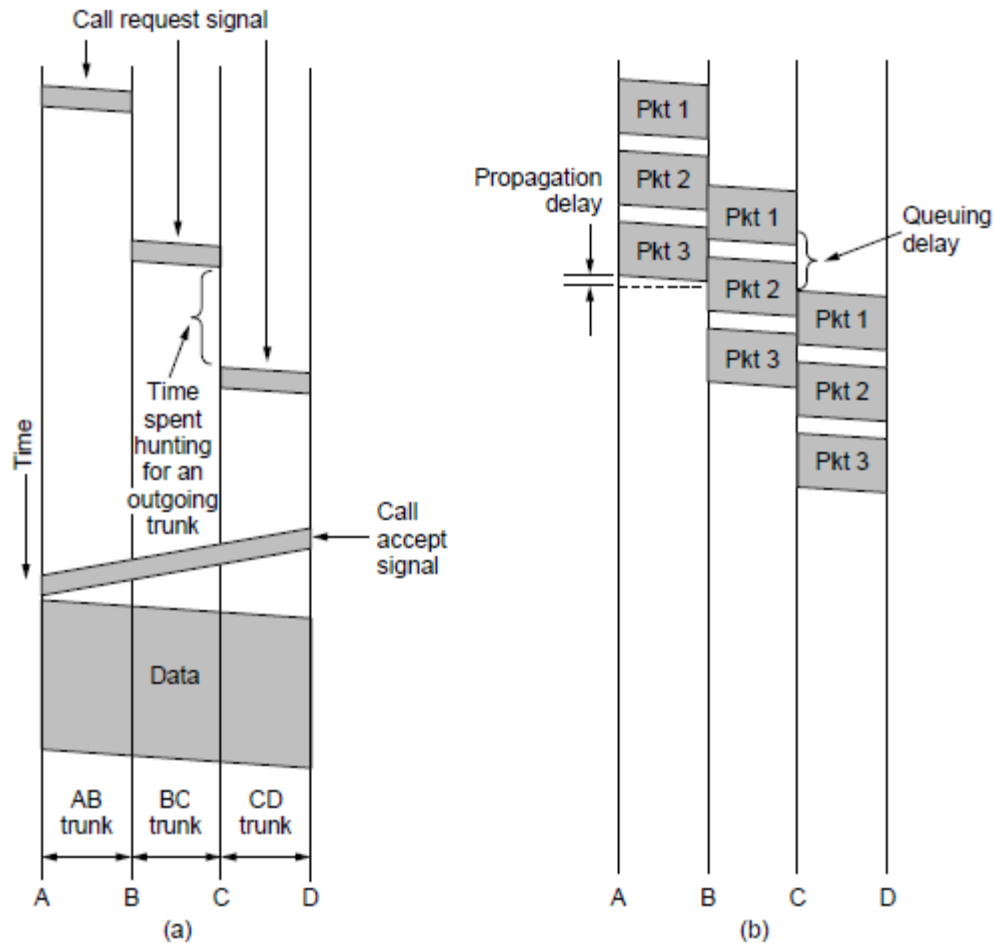
Wavelength division multiplexing

# Circuit Switching/Packet Switching (1)



(a) Circuit switching. (b) Packet switching.

# Circuit Switching/Packet Switching (2)



Timing of events in (a) circuit switching,  
(b) packet switching

# Circuit Switching/Packet Switching (3)

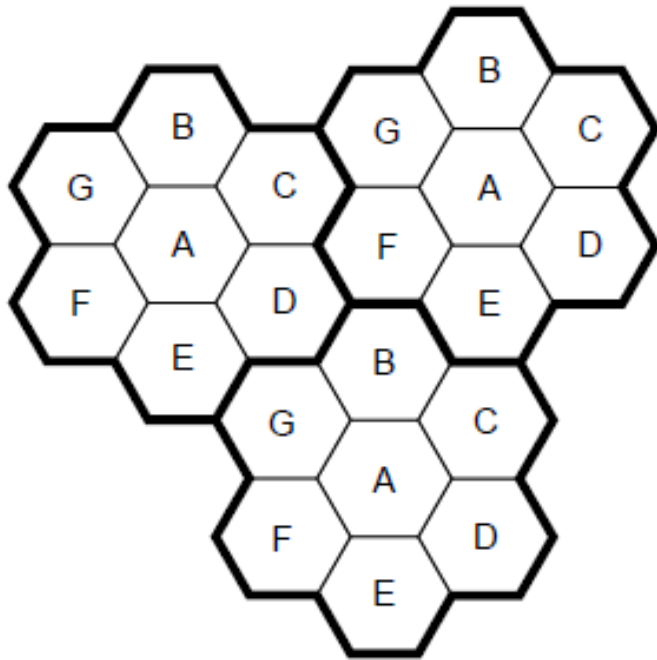
<b>Item</b>	<b>Circuit switched</b>	<b>Packet switched</b>
Call setup	Required	Not needed
Dedicated physical path	Yes	No
Each packet follows the same route	Yes	No
Packets arrive in order	Yes	No
Is a switch crash fatal	Yes	No
Bandwidth available	Fixed	Dynamic
Time of possible congestion	At setup time	On every packet
Potentially wasted bandwidth	Yes	No
Store-and-forward transmission	No	Yes
Charging	Per minute	Per packet

A comparison of circuit-switched and packet-switched networks.

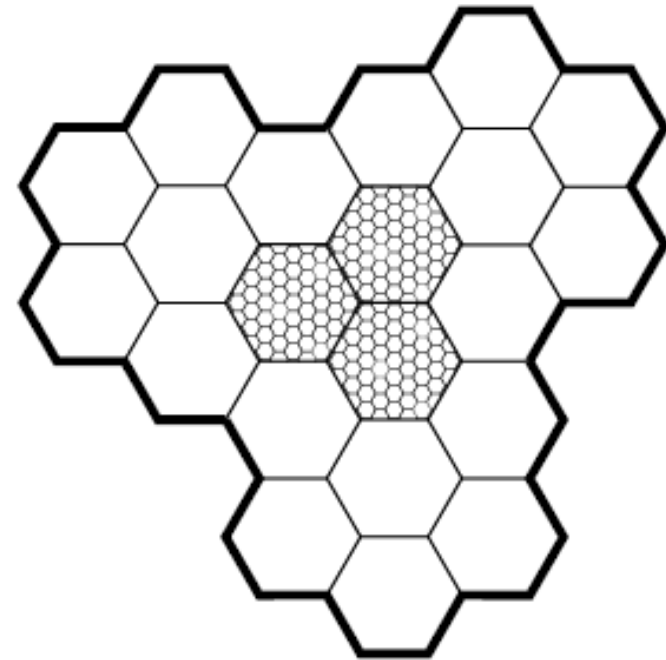
# Mobile Telephone System

- First-Generation (1G) Mobile Phones Analog Voice
- Second-Generation (2G) Mobile Phones Digital Voice
- Third-Generation (3G) Mobile Phones Digital Voice + Data

# Advanced Mobile Phone System



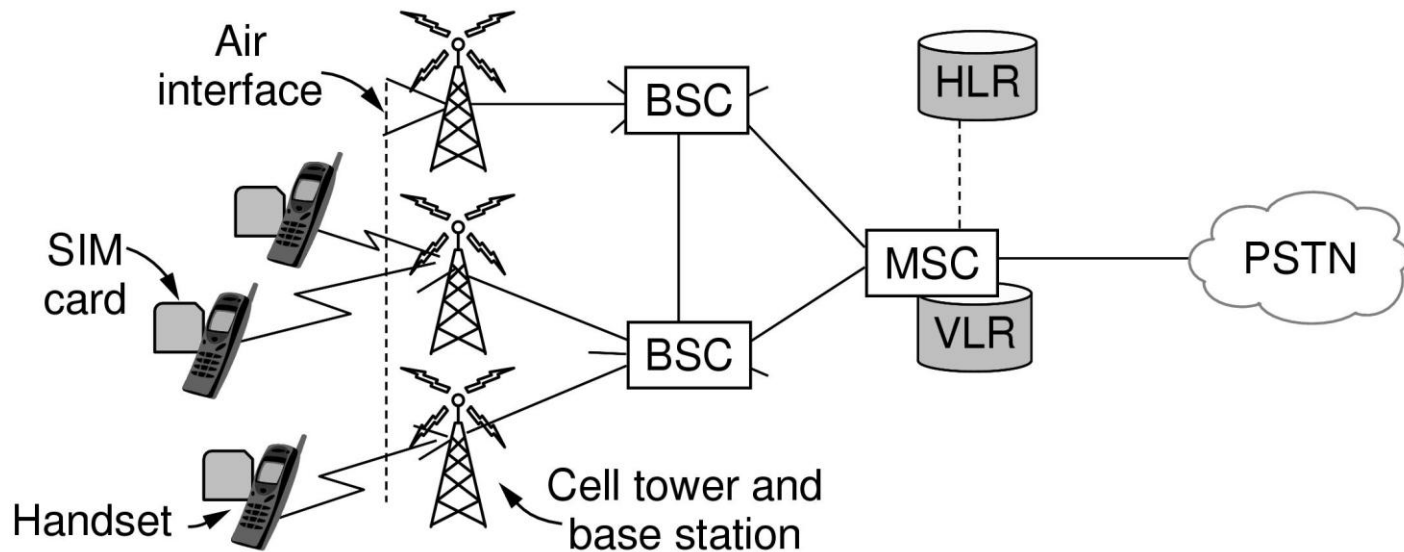
(a)



(b)

- (a) Frequencies are not reused in adjacent cells.
- (b) To add more users, smaller cells can be used.

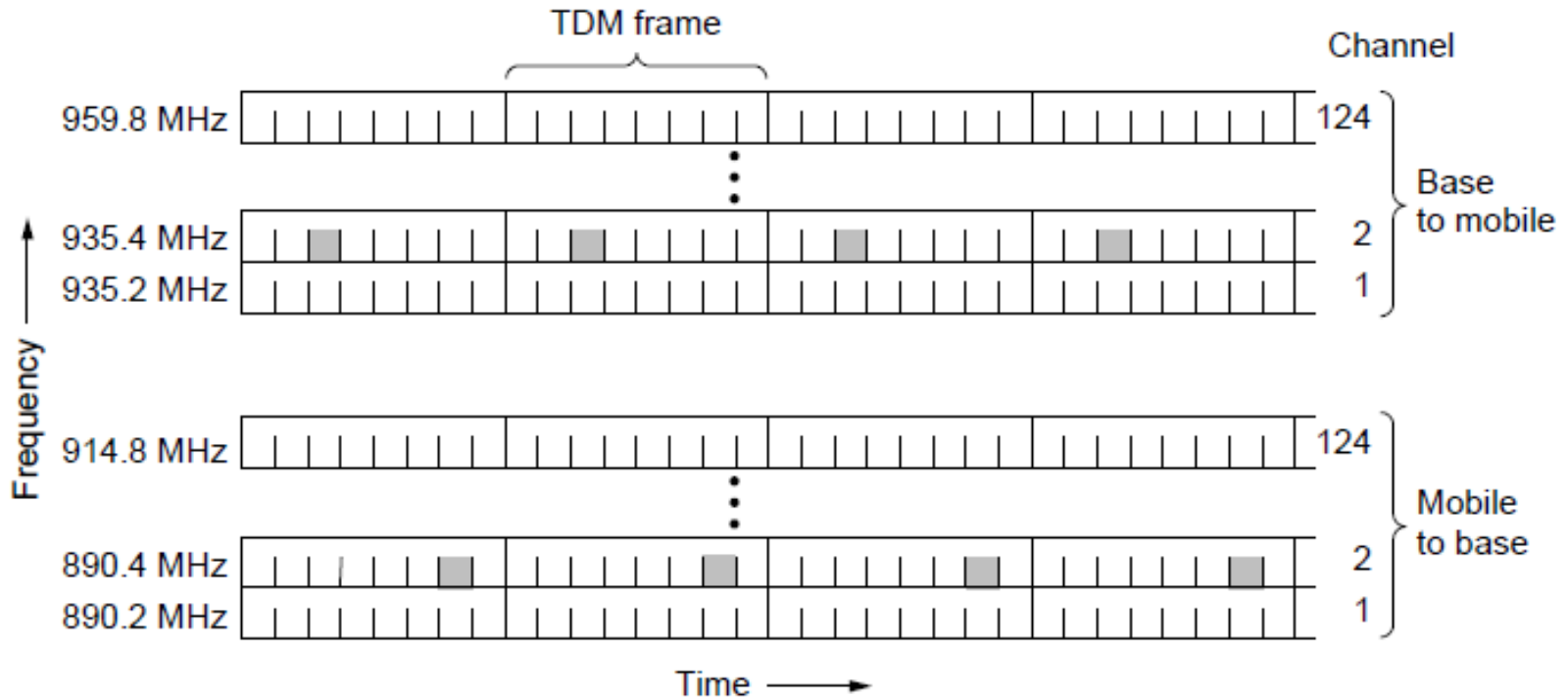
# GSM—The Global System for Mobile Communications (1)



GSM mobile network architecture.

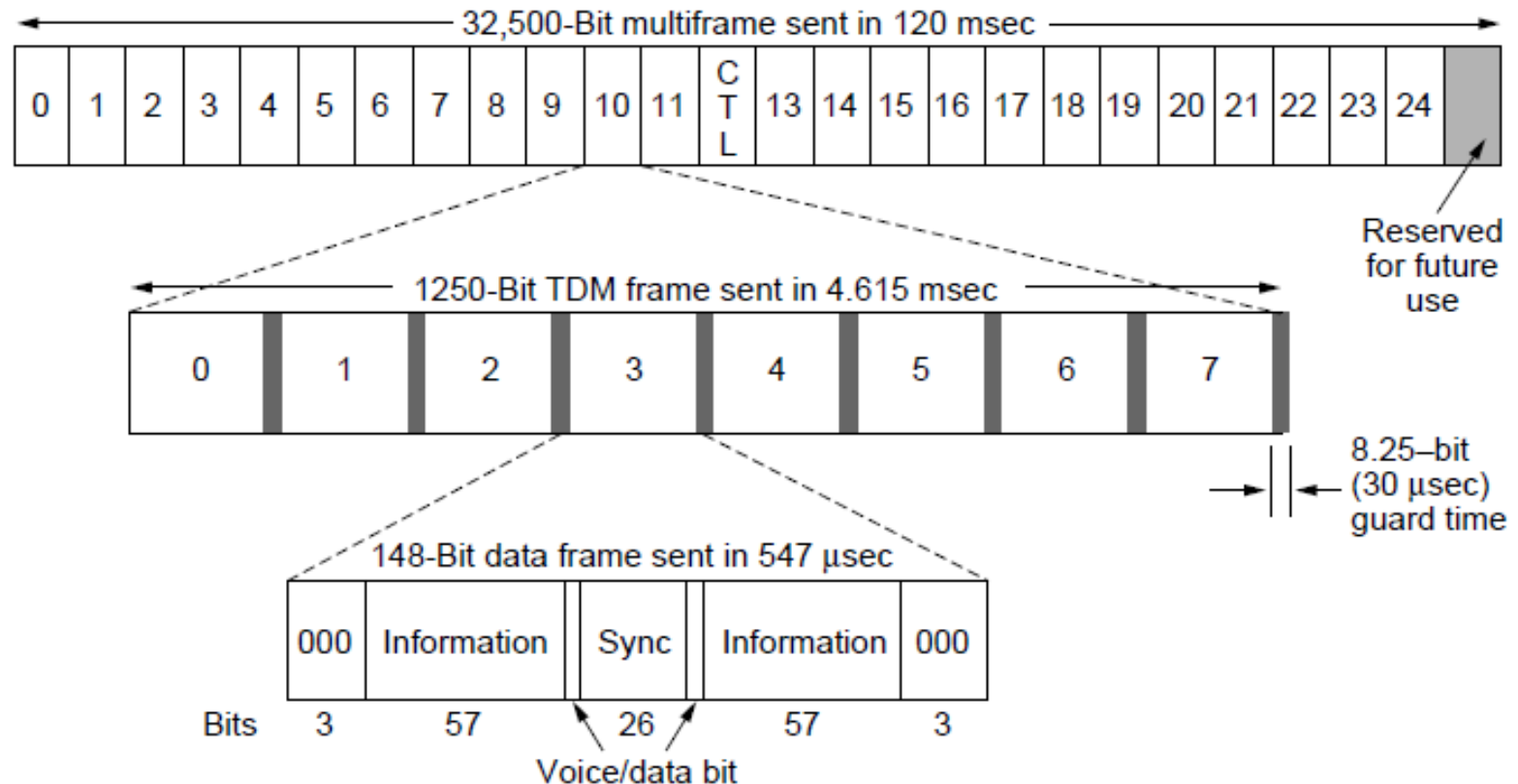


# GSM—The Global System for Mobile Communications (2)



GSM uses 124 frequency channels, each of which uses an eight-slot TDM system.

# GSM—The Global System for Mobile Communications (3)



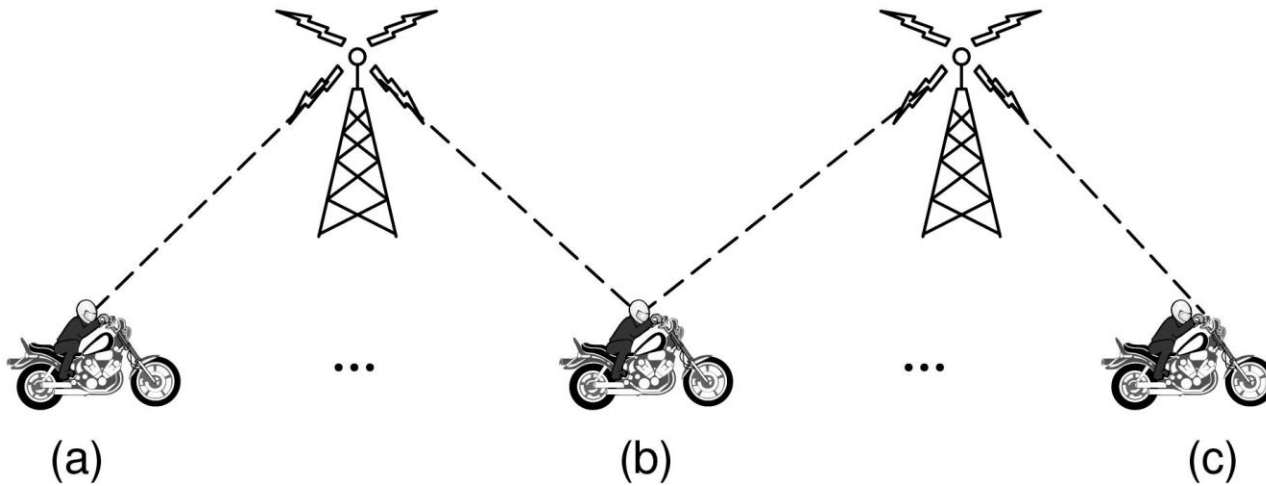
A portion of the GSM framing structure.

# Digital Voice and Data (1)

Basic services intended by IMT-2000 network

- High-quality voice transmission.
- Messaging (replacing email, fax, SMS, chat).
- Multimedia (music, videos, films, television).
- Internet access (Web surfing, incl. audio, video).

# Digital Voice and Data (2)

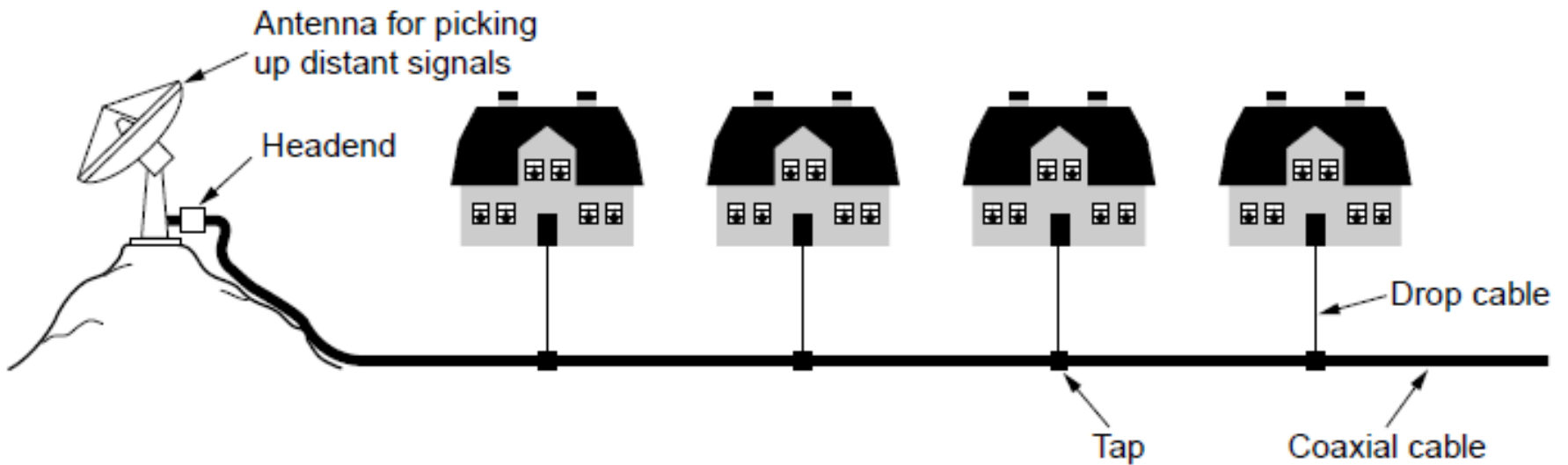


Soft handoff (a) before, (b) during, and (c) after.

# Cable Television

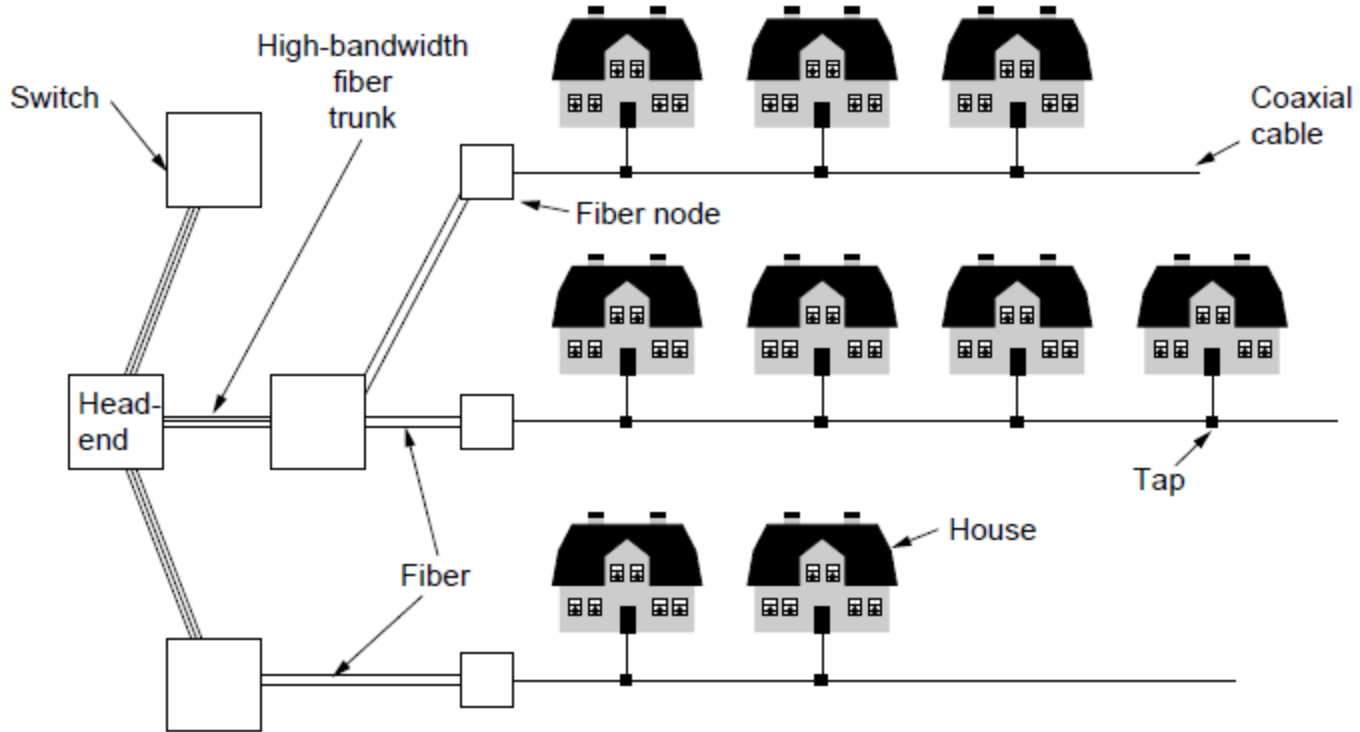
- Community antenna television
- Internet over cable
- Spectrum allocation
- Cable modems
- ADSL versus cable

# Community Antenna Television



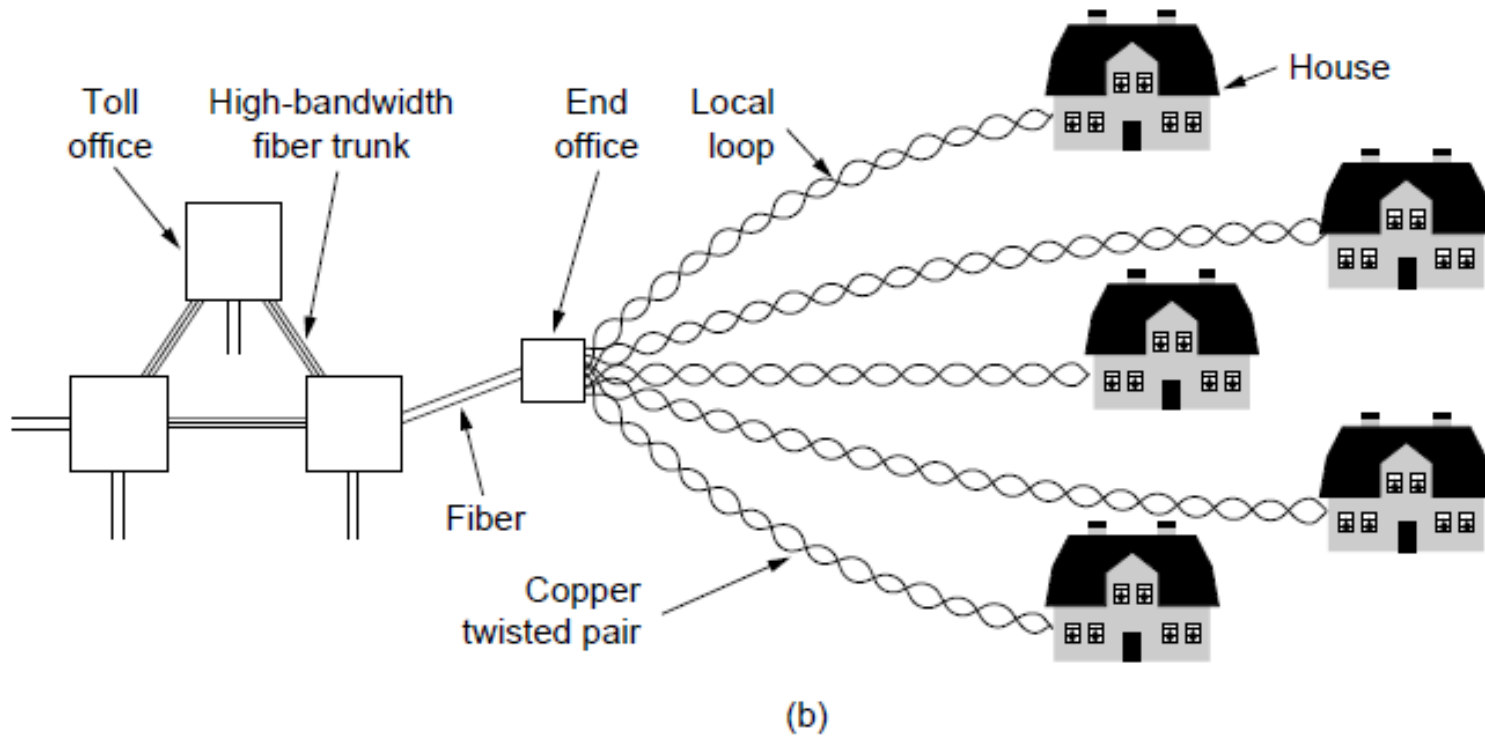
An early cable television system

# Internet over Cable (1)



## Cable television

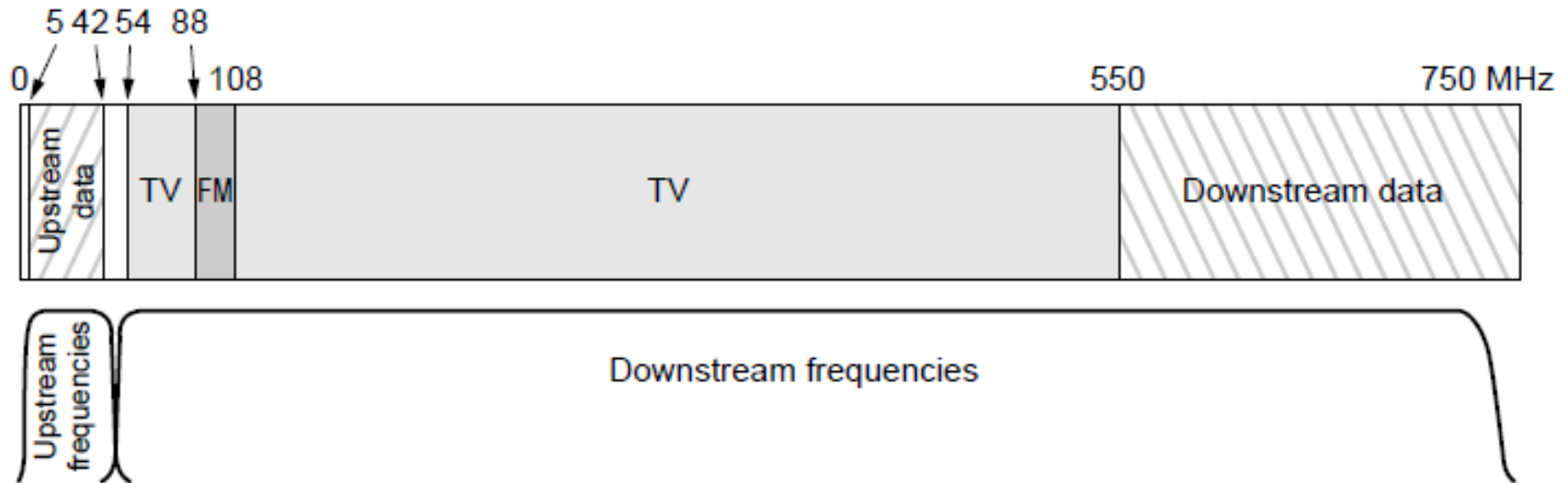
# Internet over Cable (2)



The fixed telephone system.

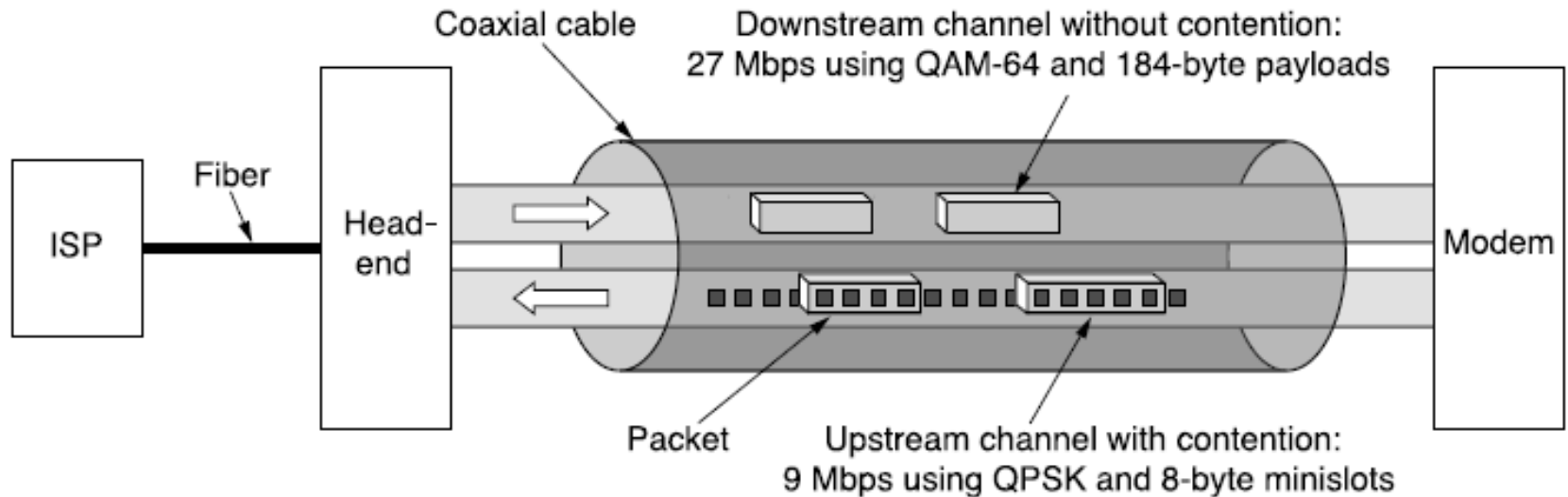


# Spectrum Allocation



Frequency allocation in a typical cable TV system used for Internet access.

# Cable Modems



Typical details of the upstream and downstream channels in North America.

# The Data Link Layer

# Data Link Layer Design Issues

- Network layer services
- Framing
- Error control
- Flow control

# Data Link Layer

- Algorithms for achieving:
  - Reliable, +
  - Efficient,  
communication of a whole units – frames (as opposed to bits – Physical Layer) between two machines.
- Two machines are connected by a communication channel that acts conceptually like a wire (e.g., telephone line, coaxial cable, or wireless channel).
- Essential property of a channel that makes it “wire-like” connection is that the bits are delivered in exactly the same order in which they are sent.

# Data Link Layer

- For ideal channel (no distortion, unlimited bandwidth and no delay) the job of data link layer would be trivial.
- However, limited bandwidth, distortions and delay makes this job very difficult.

# Data Link Layer Design Issues

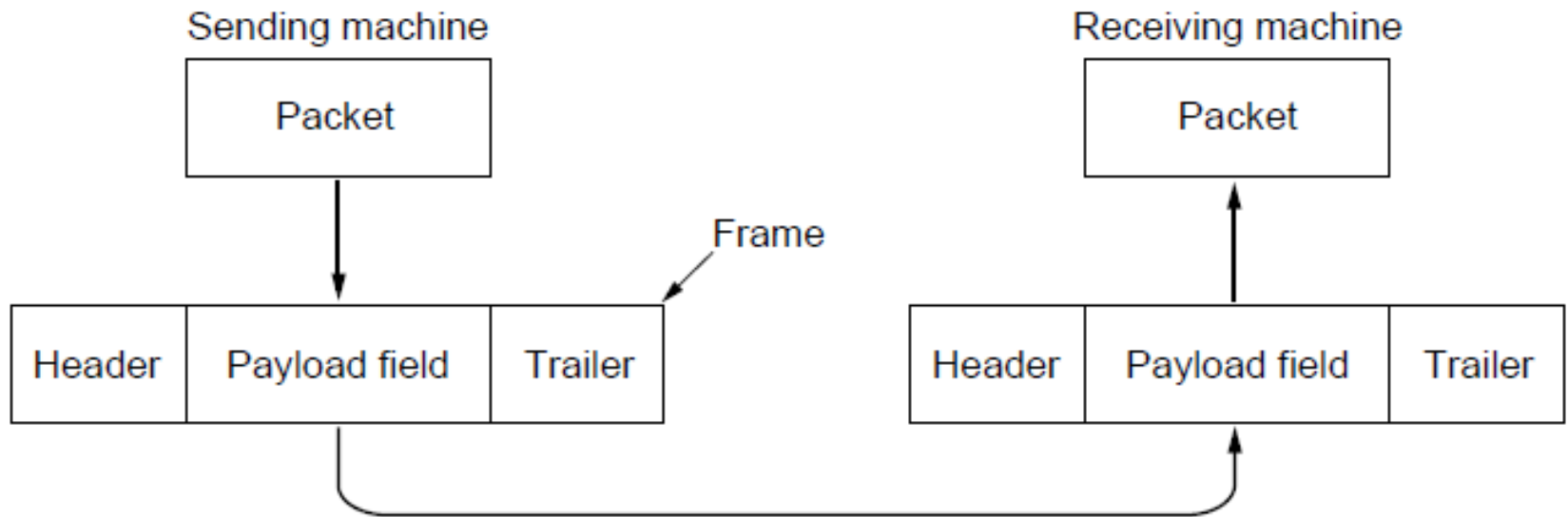
- Physical layer delivers bits of information to and from data link layer. The functions of Data Link Layer are:
  1. Providing a well-defined service interface to the network layer.
  2. Dealing with transmission errors.
  3. Regulating the flow of data so that slow receivers are not swamped by fast senders.
- Data Link layer
  - Takes the packets from Physical layer, and
  - Encapsulates them into **frames**

# Data Link Layer Design Issues

- Each frame has a
  - frame header – a field for holding the packet, and
  - frame trailer.
- Frame Management is what Data Link Layer does.
- See figure in the next slide:



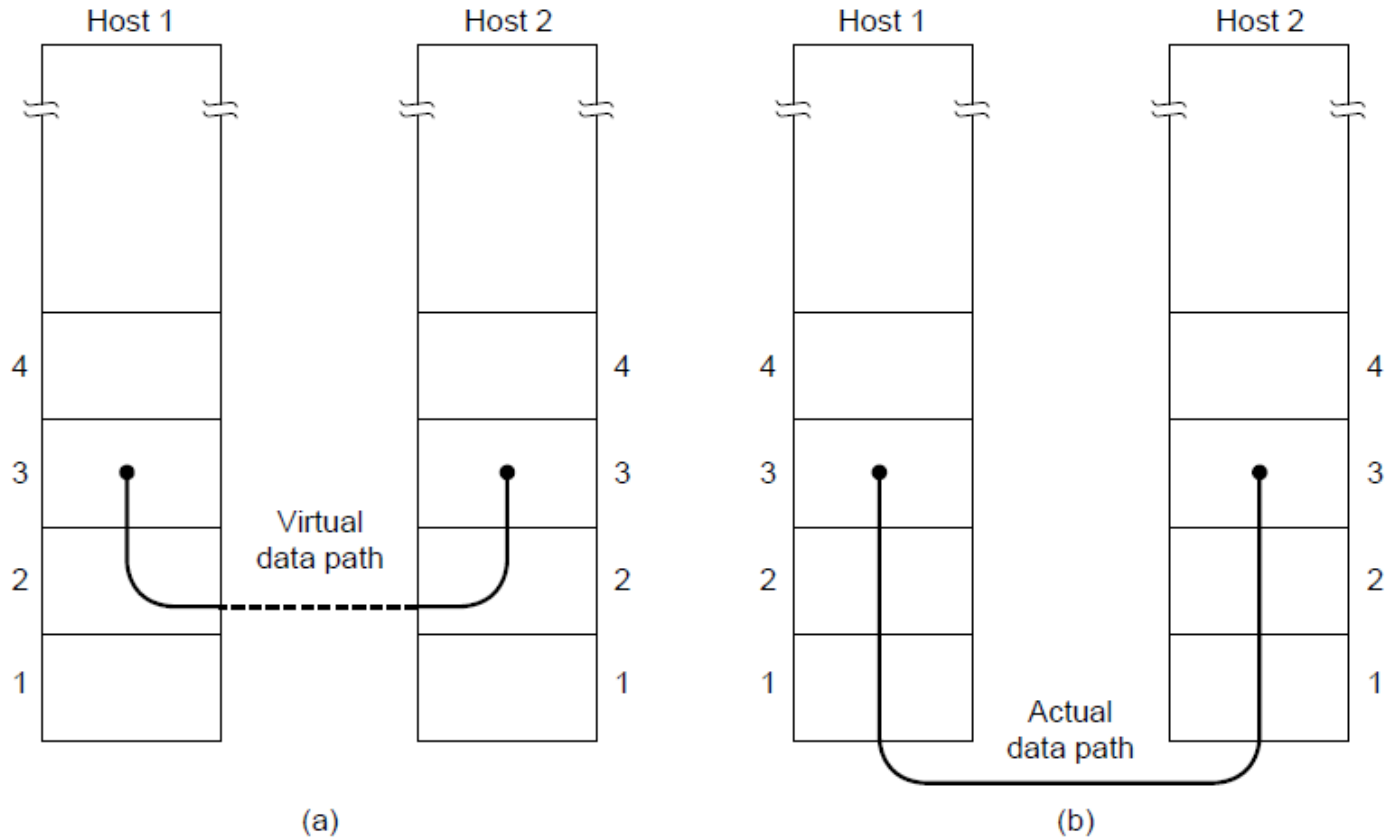
# Packets and Frames



# Services Provided to the Network Layer

- Principal Service Function of the data link layer is to transfer the data from the network layer on the source machine to the network layer on the destination machine.
  - Process in the network layer that hands some bits to the data link layer for transmission.
  - Job of data link layer is to transmit the bits to the destination machine so they can be handed over to the network layer there (see figure in the next slide).

# Network Layer Services



# Possible Services Offered

1. Unacknowledged connectionless service.
2. Acknowledged connectionless service.
3. Acknowledged connection-oriented service.

# Unacknowledged Connectionless Service

- It consists of having the source machine send independent frames to the destination machine without having the destination machine acknowledge them.
- Example: Ethernet, Voice over IP, etc. in all the communication channel where real time operation is more important than quality of transmission.

# Acknowledged Connectionless Service

- Each frame send by the Data Link layer is acknowledged and the sender knows if a specific frame has been received or lost.
- Typically the protocol uses a specific time period that if has passed without getting acknowledgment it will re-send the frame.
- This service is useful for commutation when an unreliable channel is being utilized (e.g., 802.11 WiFi).
- Network layer does not know frame size of the packets and other restriction of the data link layer. Hence it becomes necessary for data link layer to have some mechanism to optimize the transmission.

# Acknowledged Connection Oriented Service

- Source and Destination establish a connection first.
- Each frame sent is numbered
  - Data link layer guarantees that each frame sent is indeed received.
  - It guarantees that each frame is received only once and that all frames are received in the correct order.
- Examples:
  - Satellite channel communication,
  - Long-distance telephone communication, etc.

# Acknowledged Connection Oriented Service

- Three distinct phases:
  1. Connection is established by having both side initialize variables and counters needed to keep track of which frames have been received and which ones have not.
  2. One or more frames are transmitted.
  3. Finally, the connection is released – freeing up the variables, buffers, and other resources used to maintain the connection.



# Framing

- To provide service to the network layer the data link layer must use the service provided to it by physical layer.
- Stream of data bits provided to data link layer is not guaranteed to be without errors.
- Errors could be:
  - Number of received bits does not match number of transmitted bits (deletion or insertion)
  - Bit Value
- It is up to data link layer to correct the errors if necessary.

# Framing

- Transmission of the data link layer starts with breaking up the bit stream
  - into discrete frames
  - Computation of a checksum for each frame, and
  - Include the checksum into the frame before it is transmitted.
- Receiver computes its checksum error for a receiving frame and if it is different from the checksum that is being transmitted will have to deal with the error.
- Framing is more difficult than one could think!

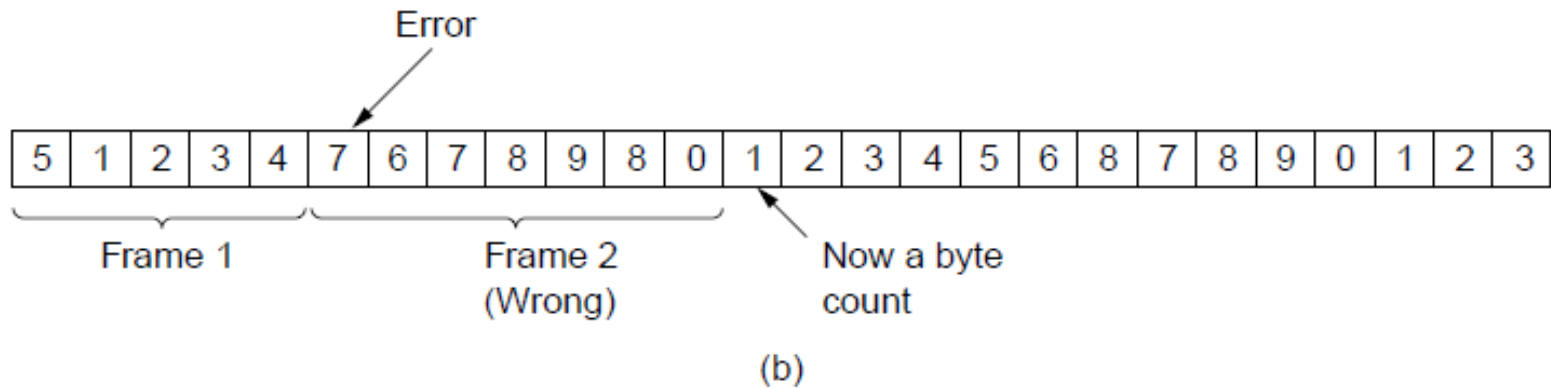
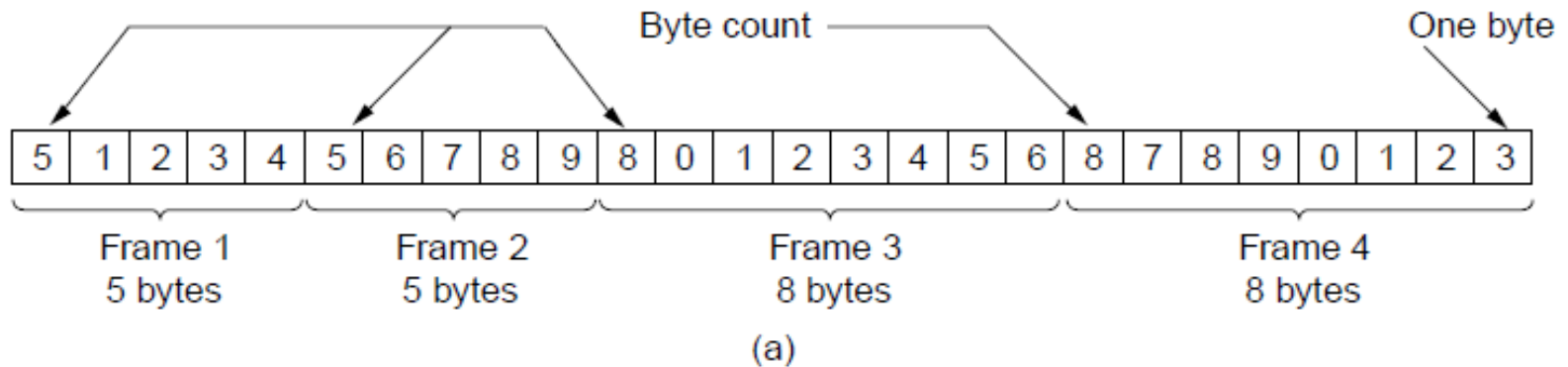
# Framing Methods

1. Byte count.
2. Flag bytes with byte stuffing.
3. Flag bits with bit stuffing.
4. Physical layer coding violations.

# Byte Count Framing Method

- It uses a field in the header to specify the number of bytes in the frame.
- Once the header information is being received it will be used to determine end of the frame.
- See figure in the next slide:
- Trouble with this algorithm is that when the count is incorrectly received the destination will get out of synch with transmission.
  - Destination may be able to detect that the frame is in error but it does not have a means (in this algorithm) how to correct it.

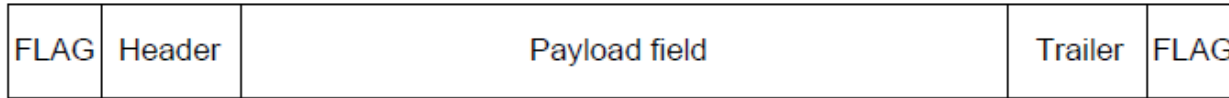
# Framing (1)



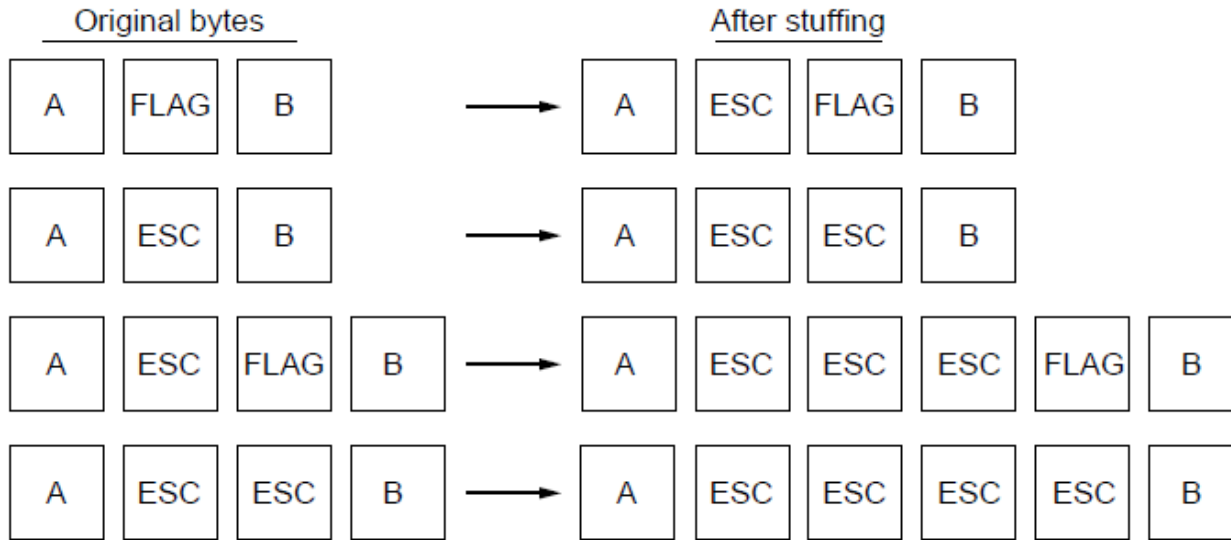
# Flag Bytes with Byte Staffing Framing Method

- This method gets around the boundary detection of the frame by having each appended by the frame start and frame end special bytes.
- If they are the same (beginning and ending byte in the frame) they are called **flag byte**.
- In the next slide figure this byte is shown as FLAG.
- If the actual data contains a byte that is identical to the FLAG byte (e.g., picture, data stream, etc.) the convention that can be used is to have escape character inserted just before the “FLAG” character.

# Framing (2)



(a)



(b)

- A frame delimited by flag bytes.
- Four examples of byte sequences before and after byte stuffing.

# Flag Bits with Bit Stuffing Framing Method

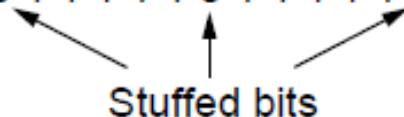
- This method achieves the same thing as Byte Stuffing method by using Bits (1) instead of Bytes (8 Bits).
- It was developed for High-level Data Link Control (HDLC) protocol.
- Each frame begins and ends with a special bit pattern:
  - 01111110 or 0x7E ← Flag Byte
  - Whenever the sender's data link layer encounters five consecutive 1s in the data it automatically stuffs a 0 bit into the outgoing bit stream.
  - USB uses bit stuffing.



# Framing (3)

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0



Stuffed bits

(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing.

# Framing

- Many data link protocols use a combination of presented methods for safety. For example in Ethernet and 802.11 each frame begin with a well-defined pattern called a preamble.
- Preamble is typically 72 bits long.
- It is then followed by a length field.

# Error Control

- After solving the marking of the frame with start and end the data link layer has to handle eventual errors in transmission or detection.
  - Ensuring that all frames are delivered to the network layer at the destination and in proper order.
- Unacknowledged connectionless service: it is OK for the sender to output frames regardless of its reception.
- Reliable connection-oriented service: it is NOT OK.

# Error Control

- Reliable connection-oriented service usually will provide a sender with some feedback about what is happening at the other end of the line.
  - Receiver Sends Back Special Control Frames.
  - If the Sender Receives positive Acknowledgment it will know that the frame has arrived safely.
- Timer and Frame Sequence Number for the Sender is Necessary to handle the case when there is no response (positive or negative) from the Receiver .

# Flow Control

- Important Design issue for the cases when the sender is running on a fast powerful computer and receiver is running on a slow low-end machine.
- Two approaches:
  1. Feedback-based flow control
  2. Rate-based flow control

# Feedback-based Flow Control

- Receiver sends back information to the sender giving it permission to send more data, or
- Telling sender how receiver is doing.

# Rate-based Flow Control

- Built in mechanism that limits the rate at which sender may transmit data, without the need for feedback from the receiver.

# Error Detection and Correction

- Two basic strategies to deal with errors:
  1. Include enough redundant information to enable the receiver to deduce what the transmitted data must have been.

## **Error correcting codes.**

2. Include only enough redundancy to allow the receiver to deduce that an error has occurred (but not which error).

## **Error detecting codes.**



# Error Detection and Correction

- Error codes are examined in Link Layer because this is the first place that we have run up against the problem of reliability transmitting groups of bits.
  - Codes are reused because reliability is an overall concern.
  - The error correcting code are also seen in the physical layer for noise channels.
  - Commonly they are used in link, network and transport layer.

# Error Detection and Correction

- Error codes have been developed after long fundamental research conducted in mathematics.
- Many protocol standards get codes from the large field in mathematics.

# Error Detection & Correction Code (1)

1. Hamming codes.
2. Binary convolutional codes.
3. Reed-Solomon codes.
4. Low-Density Parity Check codes.

# Error Detection & Correction Code

- All the codes presented in previous slide add redundancy to the information that is being sent.
- A frame consists of
  - $m$  data bits (message) and
  - $r$  redundant bits (check).
- **Block code** - the  $r$  check bits are computed solely as function of the  $m$  data bits with which they are associated.
- **Systemic code** – the  $m$  data bits are send directly along with the check bits.
- **Linear code** – the  $r$  check bits are computed as a linear function of the  $m$  data bits.

# Error Detection & Correction Code

- $n$  – total length of a block (i.e.,  $n = m + r$ )
- $(n, m)$  – code
- $n$  – bit **codeword** containing  $n$  bits.
- $m/n$  – code rate (range  $\frac{1}{2}$  for noisy channel and close to 1 for high-quality channel).

# Error Detection & Correction Code

## *Example*

- Transmitted:           10001001
- Received:               10110001

XOR operation gives number of bits that are different.

- XOR:                    00111000
  
- Number of bit positions in which two codewords differ is called *Hamming Distance*. It shows that two codes are *d* distance apart, and it will require *d* errors to convert one into the other.

# Error Detection & Correction Code

- All  $2^m$  possible data messages are legal, but due to the way the check bits are computers not all  $2^n$  possible code words are used.
- Only small fraction of  $2^m/2^n=1/2^r$  *are possible will be legal codewords.*
- The error-detecting and error-correcting codes of the block code depend on this Hamming distance.
- To reliably detect  $d$  error, one would need a distance  $d+1$  code.
- To correct  $d$  error, one would need a distance  $2d+1$  code.

# Error Detection & Correction Code

- All  $2^m$  possible data messages are legal, but due to the way the check bits are computers not all  $2^n$  possible code words are used.
- Only small fraction of  $2^m/2^n=1/2^r$  *are possible will be legal codewords.*
- The error-detecting and error-correcting codes of the block code depend on this Hamming distance.
- To reliably detect  $d$  error, one would need a distance  $d+1$  code.
- To correct  $d$  error, one would need a distance  $2d+1$  code.



# Error Detection & Correction Code

## Example:

- 4 valid codes:
  - 0000000000
  - 0000011111
  - 1111100000
  - 1111111111
- Minimal Distance of this code is 5  $\Rightarrow$  can correct and double errors and it detect quadruple errors.
- 0000000111  $\Rightarrow$  single or double – bit error. Hence the receiving end must assume the original transmission was 0000011111.
- 0000000000  $\Rightarrow$  had triple error that was received as 0000000111 it would be detected in error.

# Error Detection & Correction Code

- One cannot perform double errors and at the same time detect quadruple errors.
- Error correction requires evaluation of each candidate codeword which may be time consuming search.
- Through design this search time can be minimized.
- In theory if  $n = m + r$ , this requirement becomes:
  - $(m + r + 1) \leq 2^r$

# Hamming Code

- Codeword:  $b_1 b_2 b_3 b_4 \dots$
- Check bits: The bits that are powers of 2 ( $p_1, p_2, p_4, p_8, p_{16}, \dots$ ).
- The rest of bits ( $m_3, m_5, m_6, m_7, m_9, \dots$ ) are filled with  $m$  data bits.
- Example of the Hamming code with  $m = 7$  data bits and  $r = 4$  check bits is given in the next slide.

# The Hamming Code

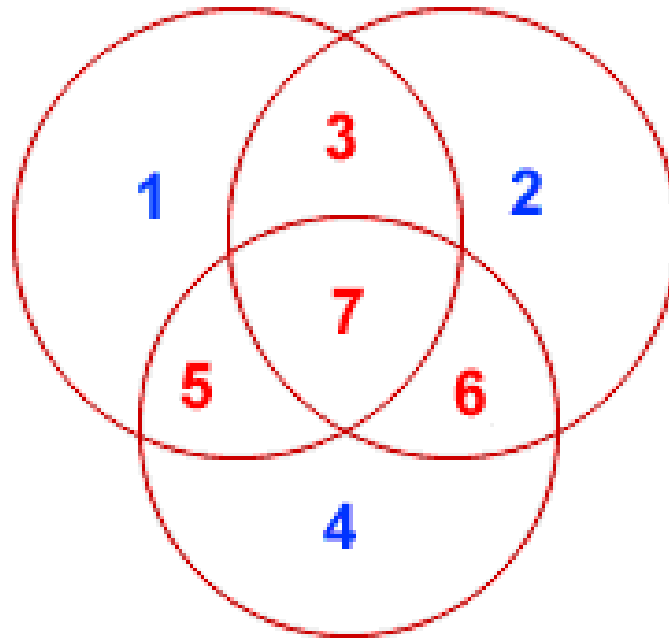
Consider a message having four data bits (D) which is to be transmitted as a 7-bit codeword by adding three error control bits. This would be called a (7,4) code. The three bits to be added are three EVEN Parity bits (P), where the parity of each is computed on different subsets of the message bits as shown below.

7	6	5	4	3	2	1	
D	D	D	P	D	P	P	7-BIT CODEWORD
D	-	D	-	D	-	P	(EVEN PARITY)
D	D	-	-	D	P	-	(EVEN PARITY)
D	D	D	P	-	-	-	(EVEN PARITY)

# Hamming Code

- **Why Those Bits?** - The three parity bits (1,2,4) are related to the data bits (3,5,6,7) as shown at right. In this diagram, each overlapping circle corresponds to one parity bit and defines the four bits contributing to that parity computation. For example, data bit 3 contributes to parity bits 1 and 2. Each circle (parity bit) encompasses a total of four bits, and each circle must have EVEN parity. Given four data bits, the three parity bits can easily be chosen to ensure this condition.
- It can be observed that changing any one bit numbered 1..7 uniquely affects the three parity bits. Changing bit 7 affects all three parity bits, while an error in bit 6 affects only parity bits 2 and 4, and an error in a parity bit affects only that bit. The location of any single bit error is determined directly upon checking the three parity circles.

# Hamming Code



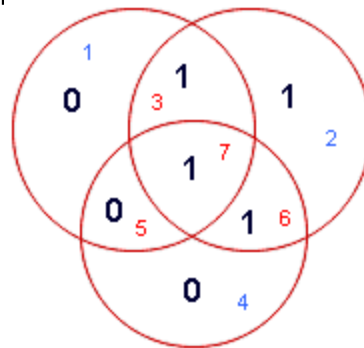
# Hamming Code

- For example, the message 1101 would be sent as 1100110, since:

7	6	5	4	3	2	1	
1	1	0	0	1	1	0	7-BIT CODEWORD
1	-	0	-	1	-	0	(EVEN PARITY)
1	1	-	-	1	1	-	(EVEN PARITY)
1	1	0	0	-	-	-	(EVEN PARITY)

# Hamming Codes

- When these seven bits are entered into the parity circles, it can be confirmed that the choice of these three parity bits ensures that the parity within each circle is EVEN, as shown here.





# Hamming Code

- It may now be observed that if an error occurs in any of the seven bits, that error will affect different combinations of the three parity bits depending on the bit position.
- For example, suppose the above message 1100110 is sent and a single bit error occurs such that the codeword 1110110 is received:

transmitted message

1 1 0 0 1 1 0  
BIT: 7 6 5 4 3 2 1

----->

received message

1 1 1 0 1 1 0  
BIT: 7 6 5 4 3 2 1

The above error (in bit 5) can be corrected by examining which of the three parity bits was affected by the bad bit:

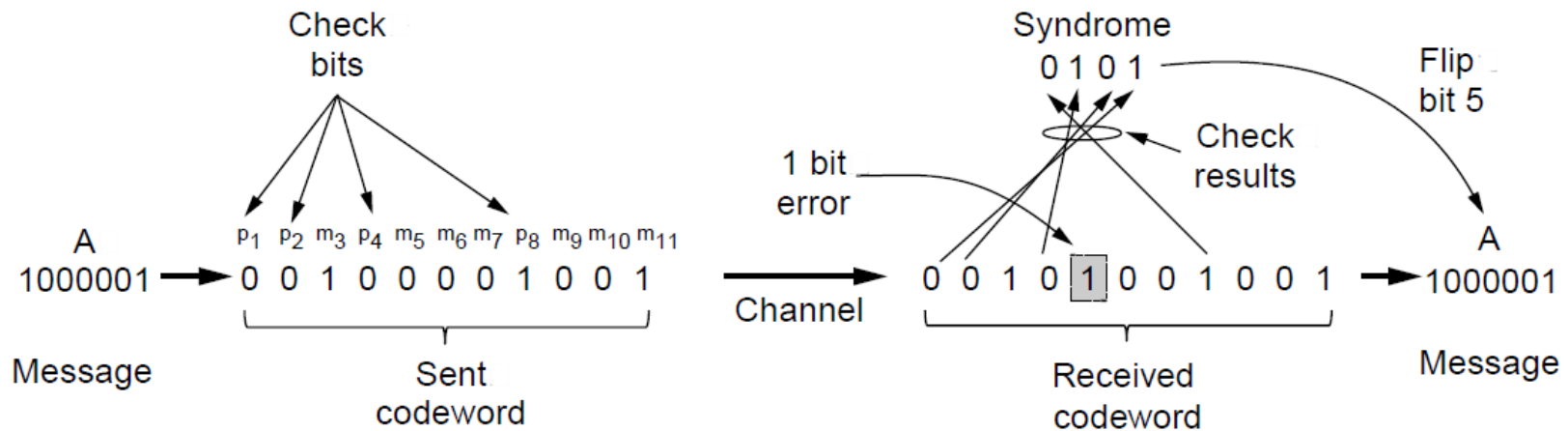
# Hamming Code

7	6	5	4	3	2	1			
1	1	1	0	1	1	0	7-BIT CODEWORD		
1	-	1	-	1	-	0	(EVEN PARITY)	NOT!	1
1	1	-	-	1	1	-	(EVEN PARITY)	OK!	0
1	1	1	0	-	-	-	(EVEN PARITY)	NOT!	1

# Hamming Code

- *In fact, the bad parity bits labeled **101** point directly to the bad bit since **101** binary equals **5**. Examination of the 'parity circles' confirms that any single bit error could be corrected in this way.*
- The value of the Hamming code can be summarized:
  1. Detection of 2 bit errors (assuming no correction is attempted);
  2. Correction of single bit errors;
  3. Cost of 3 bits added to a 4-bit message.
- The ability to correct single bit errors comes at a cost which is less than sending the entire message twice. (Recall that simply sending a message twice accomplishes no error correction.)

# Error Detection Codes (2)

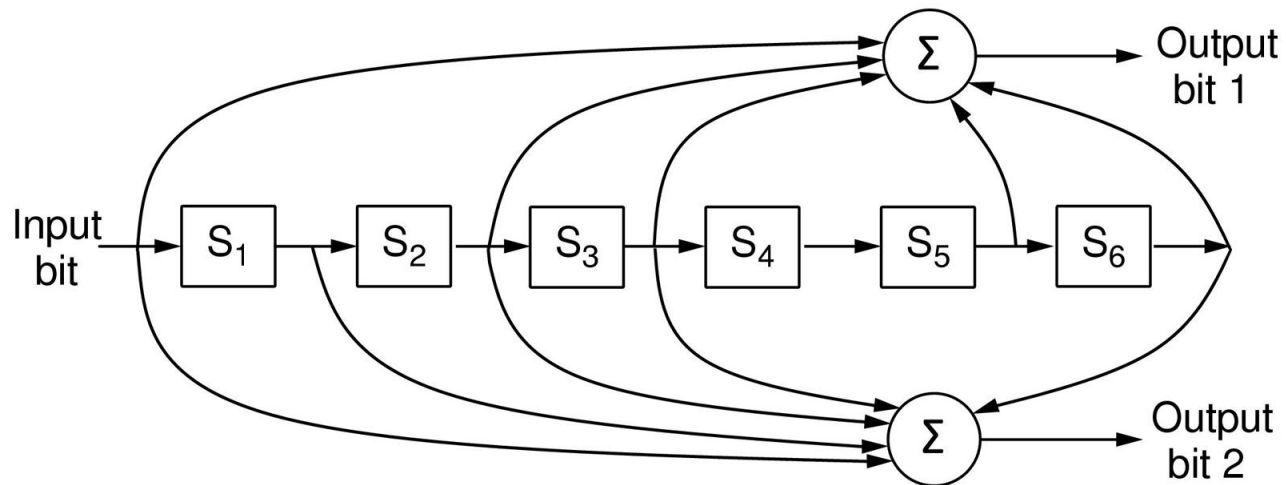


Example of an (11, 7) Hamming code correcting a single-bit error.

# Convolutional Codes

- Not a block code
- There is no natural message size or encoding boundary as in a block code.
- The output depends on the current and previous input bits. Encoder has memory.
- The number of previous bits on which the output depends is called the **constraint length** of the code.
- They are deployed as part of the
  - GSM mobile phone system
  - Satellite Communications, and
  - 802.11 (see example in the previous slide).

# Error Detection Codes (3)



The NASA binary convolutional code used in  
802.11.

# Convolutional Encoders

- Like any error-correcting code, a convolutional code works by adding some structured redundant information to the user's data and then correcting errors using this information.
- A convolutional encoder is a *linear system*.
- A binary convolutional encoder can be represented as a *shift register*. The outputs of the encoder are modulo 2 sums of the values in the certain register's cells. The input to the encoder is either the unencoded sequence (for *non-recursive codes*) or the unencoded sequence added with the values of some register's cells (for *recursive codes*).
- Convolutional codes can be *systematic* and *non-systematic*. Systematic codes are those where an unencoded sequence is a part of the output sequence. Systematic codes are almost always recursive, conversely, non-recursive codes are almost always non-systematic.

# Convolutional Encoders

- A combination of register's cells that forms one of the output streams (or that is added with the input stream for recursive codes) is defined by a *polynomial*. Let  $m$  be the maximum degree of the polynomials constituting a code, then  $K=m+1$  is a *constraint length* of the code.

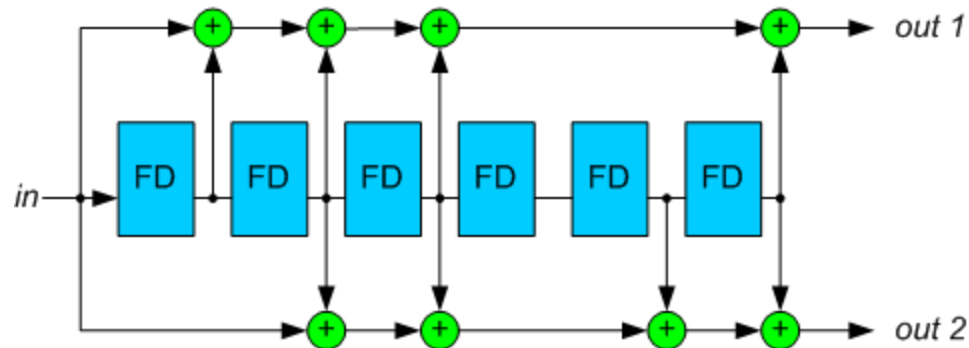


Figure 1. A standard NASA convolutional encoder with polynomials (171,133).



# Convolutional Encoders

- For example, for the decoder on the Figure 1, the polynomials are:

$$g_1(z) = 1 + z + z^2 + z^3 + z^6$$

$$g_2(z) = 1 + z^2 + z^3 + z^5 + z^6$$

- A code rate is an inverse number of output polynomials.
- For the sake of clarity, in this article we will restrict ourselves to the codes with rate  $R=1/2$ . Decoding procedure for other codes is similar.
- Encoder polynomials are usually denoted in the octal notation. For the above example, these designations are “1111001” = 171 and “1011011” = 133.
- The constraint length of this code is 7.
- An example of a recursive convolutional encoder is on the Figure 2.

# Example of the Convolutional Encoder

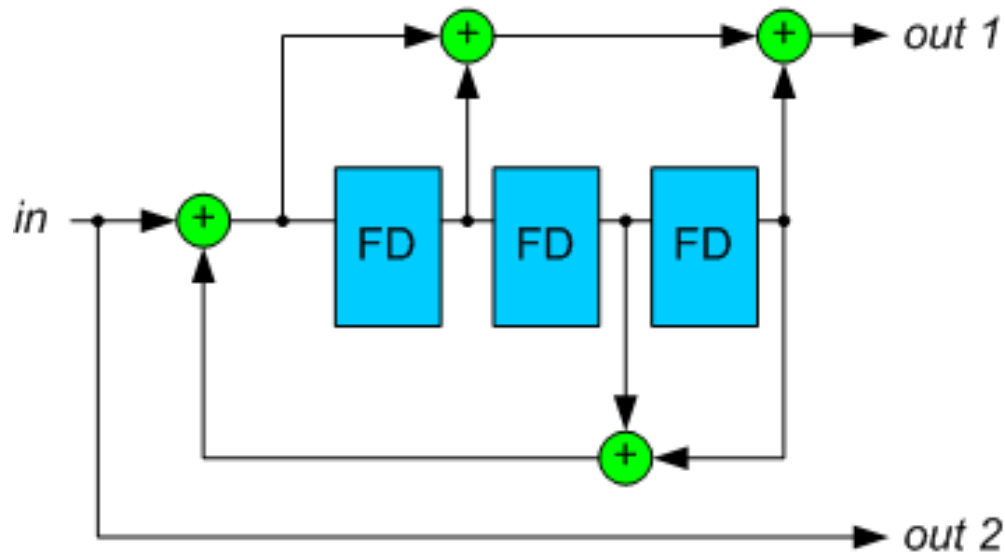


Figure 2. A recursive convolutional encoder.

# Trellis Diagram

- A convolutional encoder is often seen as a *finite state machine*. Each state corresponds to some value of the encoder's register. Given the input bit value, from a certain state the encoder can move to two other states. These state transitions constitute a diagram which is called a *trellis diagram*.
- A trellis diagram for the code on the Figure 2 is depicted on the Figure 3. A solid line corresponds to input 0, a dotted line – to input 1 (note that encoder states are designated in such a way that the rightmost bit is the newest one).
- Each path on the trellis diagram corresponds to a valid sequence from the encoder's output. Conversely, any valid sequence from the encoder's output can be represented as a path on the trellis diagram. One of the possible paths is denoted as red (as an example).

# Trellis Diagram

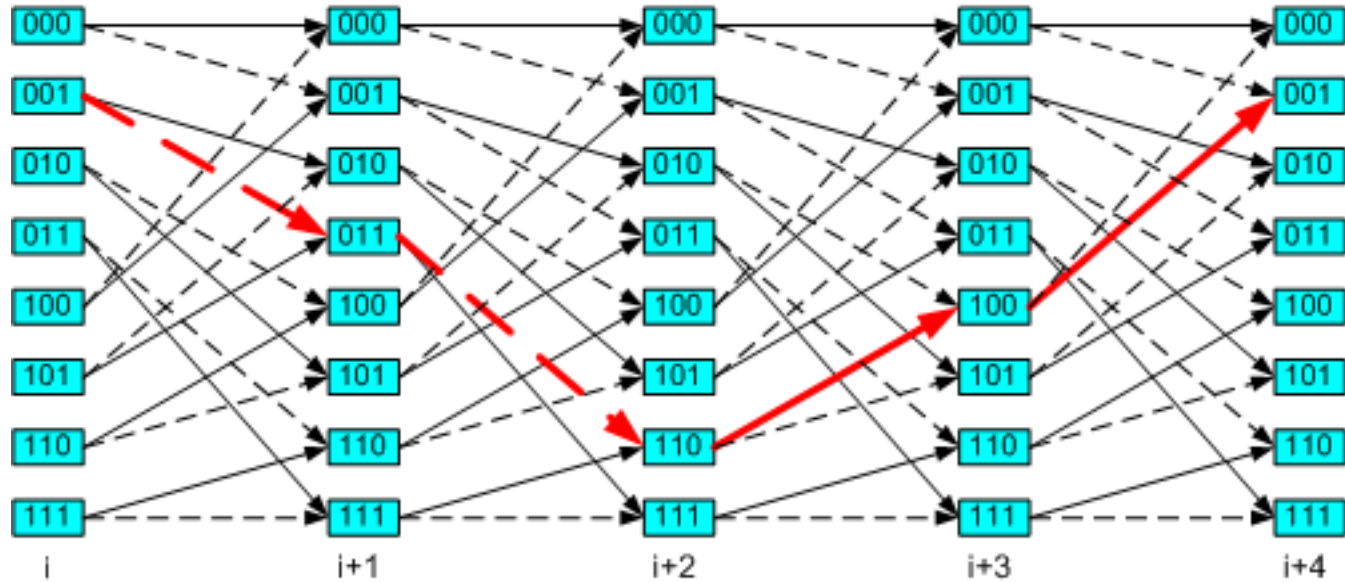


Figure 3. A trellis diagram corresponding to the encoder on the Figure 2.

# Trellis Diagram

- Note that each state transition on the diagram corresponds to a pair of output bits. There are only two allowed transitions for every state, so there are two allowed pairs of output bits, and the two other pairs are forbidden. If an error occurs, it is very likely that the receiver will get a set of forbidden pairs, which don't constitute a path on the trellis diagram. So, the task of the decoder is to find a path on the trellis diagram which is the closest match to the received sequence.

# Trellis Diagram

- Let's define a *free distance*  $d_f$  as a minimal Hamming distance between two different allowed binary sequences (a Hamming distance is defined as a number of differing bits).
- A free distance is an important property of the convolutional code. It influences a number of closely located errors the decoder is able to correct.

# Viterbi Algorithm

- Viterbi algorithm reconstructs the maximum-likelihood path for a given input sequence.

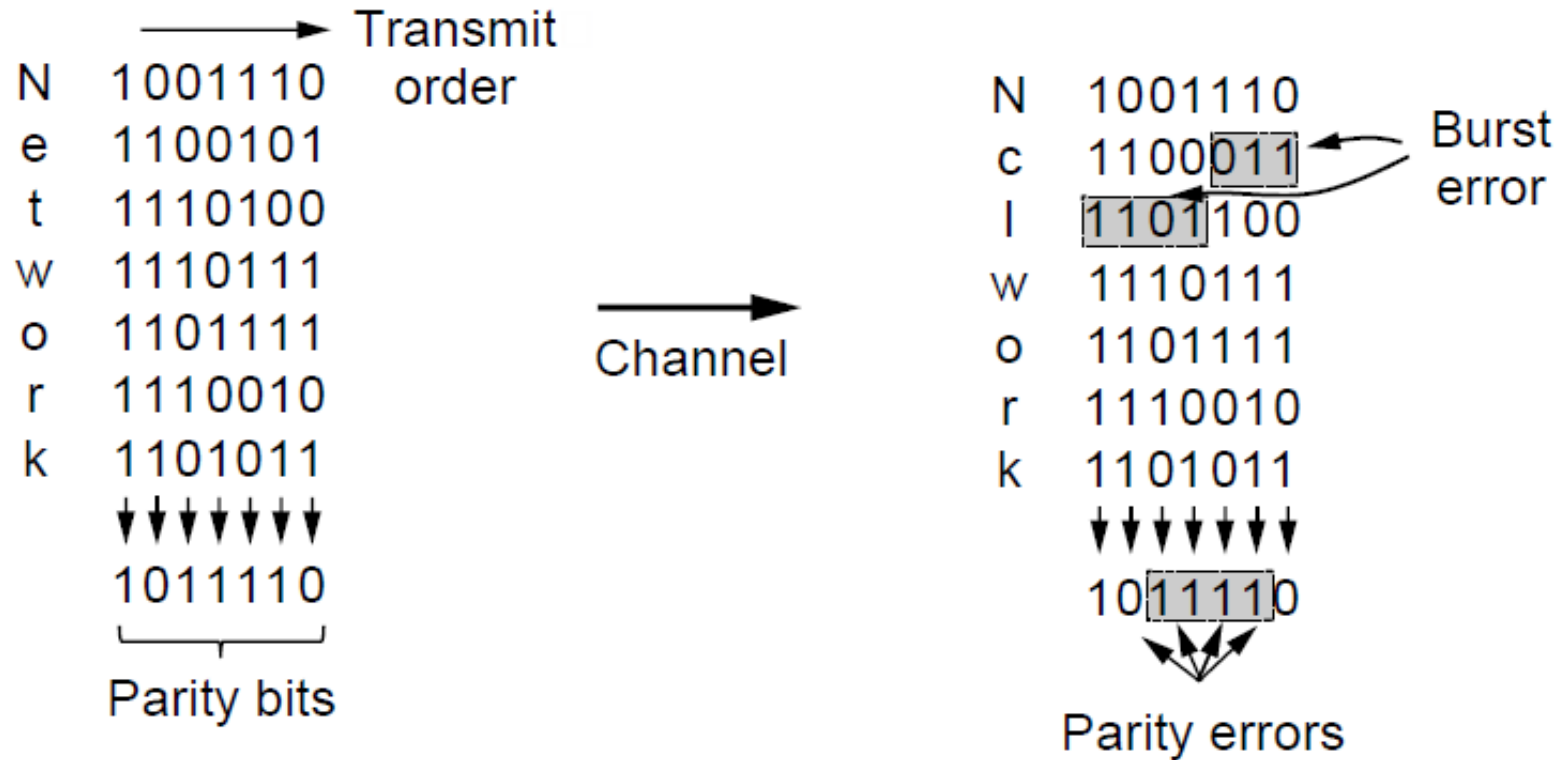
# Error-Detecting Codes (1)

Linear, systematic block codes

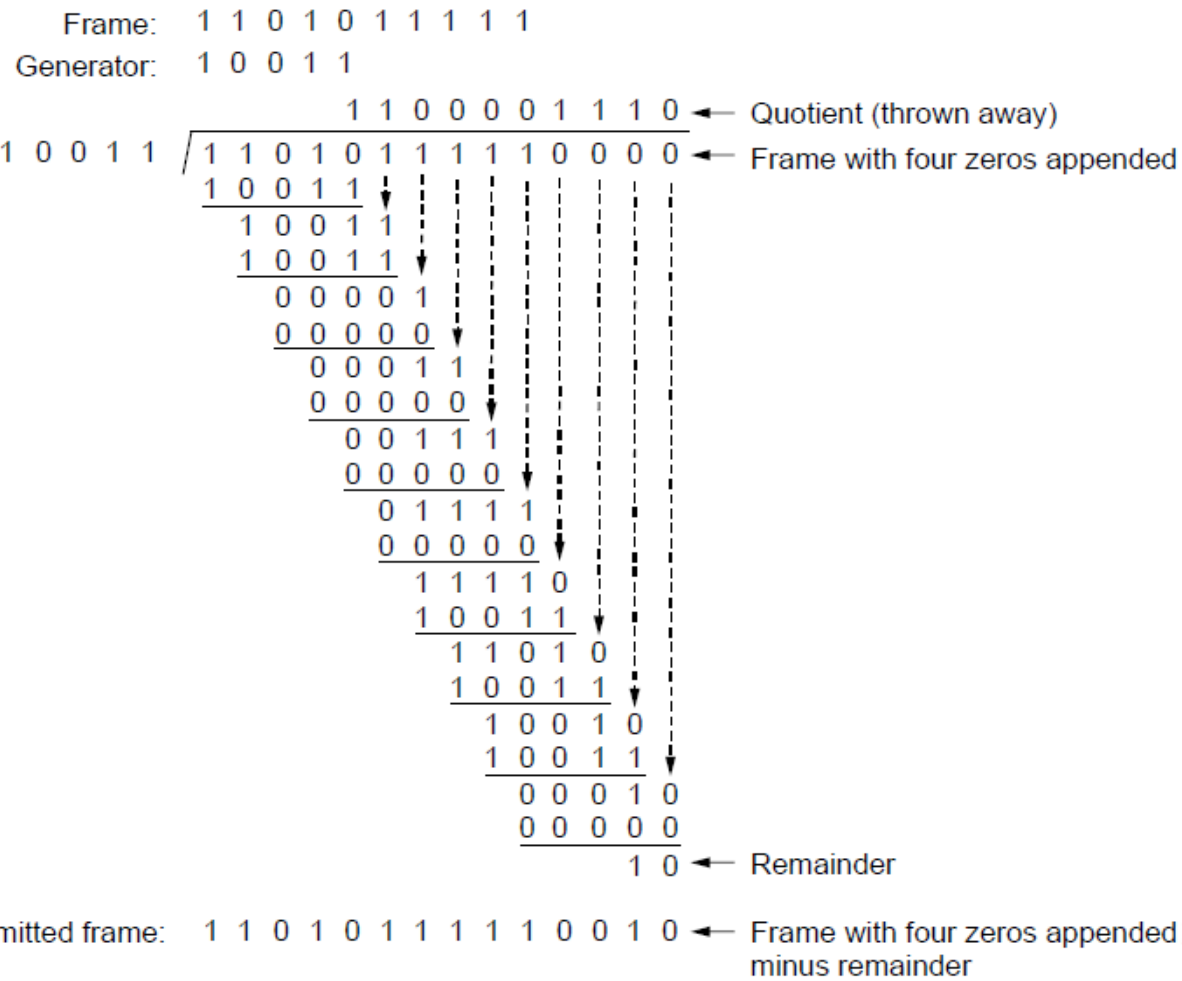
1. Parity.
2. Checksums.
3. Cyclic Redundancy Checks (CRCs).



# Error-Detecting Codes (2)



# Error-Detecting Codes (3)

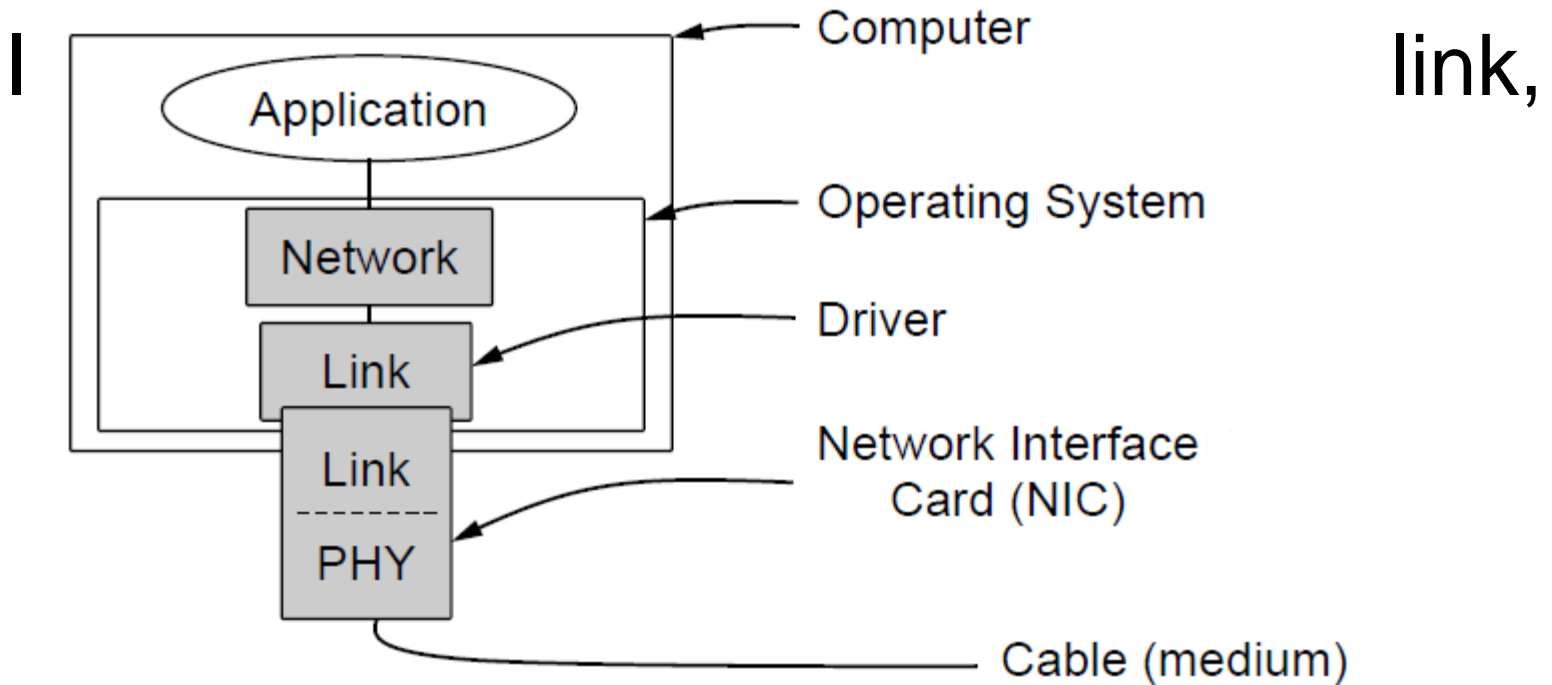


Example calculation of the CRC

# Elementary Data Link Protocols (1)

- Utopian Simplex Protocol
- Simplex Stop-and-Wait Protocol
  - Error-Free Channel
- Simplex Stop-and-Wait Protocol
  - Noisy Channel

# Elementary Data Link Protocols (2)



# Elementary Data Link Protocols (3)

```
#define MAX_PKT 1024                                /* determines packet size in bytes */

typedef enum {false, true} boolean;                 /* boolean type */
typedef unsigned int seq_nr;                         /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet; /* packet definition */
typedef enum {data, ack, nak} frame_kind;           /* frame_kind definition */

typedef struct {                                     /* frames are transported in this layer */
    frame_kind kind;                                /* what kind of frame is it? */
    seq_nr seq;                                     /* sequence number */
    seq_nr ack;                                     /* acknowledgement number */
    packet info;                                    /* the network layer packet */
} frame;

    . . .
```

# Elementary Data Link Protocols (4)

```
/* Wait for an event to happen; return its type in event. */  
void wait_for_event(event_type *event);
```

```
/* Fetch a packet from the network layer for transmission on the channel. */  
void from_network_layer(packet *p);
```

```
/* Deliver information from an inbound frame to the network layer. */  
void to_network_layer(packet *p);
```

```
/* Go get an inbound frame from the physical layer and copy it to r. */  
void from_physical_layer(frame *r);
```

```
/* Pass the frame to the physical layer for transmission. */  
void to_physical_layer(frame *s);
```

```
/* Start the clock running and enable the timeout event. */  
void start_timer(seq_nr k);
```

```
/* Stop the clock and disable the timeout event. */  
void stop_timer(seq_nr k); . . .
```

|s  
n

# Elementary Data Link Protocols (5)

```
/* Start an auxiliary timer and enable the ack_timeout event. */  
void start_ack_timer(void);
```

```
/* Stop the auxiliary timer and disable the ack_timeout event. */  
void stop_ack_timer(void);
```

```
/* Allow the network layer to cause a network_layer_ready event. */  
void enable_network_layer(void);
```

```
/* Forbid the network layer from causing a network_layer_ready event. */  
void disable_network_layer(void);
```

```
/* Macro inc is expanded in-line: increment k circularly. */  
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

ols  
in

# Utopian Simplex Protocol (1)

*/\* Protocol 1 (Utopia) provides for data transmission in one direction only, from sender to receiver. The communication channel is assumed to be error free and the receiver is assumed to be able to process all the input infinitely quickly. Consequently, the sender just sits in a loop pumping data out onto the line as fast as it can. \*/*

```
typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
    frame s;                /* buffer for an outbound frame */
    packet buffer;         /* buffer for an outbound packet */

    while (true) {
        from_network_layer(&buffer); /* go get something to send */
        s.info = buffer;           /* copy it into s for transmission */
        to_physical_layer(&s);    /* send it on its way */
    }                               /* Tomorrow, and tomorrow, and tomorrow,
                                   Creeps in this petty pace from day to day
                                   To the last syllable of recorded time.
                                   – Macbeth, V, v */
}

. . .
```

A utopian simplex protocol.



# Utopian Simplex Protocol (2)

```
void receiver1(void)
{
    frame r;
    event_type event;                /* filled in by wait, but not used here */

    while (true) {
        wait_for_event(&event);      /* only possibility is frame_arrival */
        from_physical_layer(&r);     /* go get the inbound frame */
        to_network_layer(&r.info);   /* pass the data to the network layer */
    }
}
```

# Simplex Stop-and-Wait Protocol for a Noisy Channel (1)

*/\* Protocol 2 (Stop-and-wait) also provides for a one-directional flow of data from sender to receiver. The communication channel is once again assumed to be error free, as in protocol 1. However, this time the receiver has only a finite buffer capacity and a finite processing speed, so the protocol must explicitly prevent the sender from flooding the receiver with data faster than it can be handled. \*/*

```
typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
    frame s;                /* buffer for an outbound frame */
    packet buffer;         /* buffer for an outbound packet */
    event_type event;      /* frame_arrival is the only possibility */

    while (true) {
        from_network_layer(&buffer); /* go get something to send */
        s.info = buffer;             /* copy it into s for transmission */
        to_physical_layer(&s);      /* bye-bye little frame */
        wait_for_event(&event);     /* do not proceed until given the go ahead */
    }
} . . .
```

A simplex stop-and-wait protocol.

# Simplex Stop-and-Wait Protocol for a Noisy Channel (2)

```
void receiver2(void)
{
    frame r, s;                /* buffers for frames */
    event_type event;         /* frame_arrival is the only possibility */
    while (true) {
        wait_for_event(&event); /* only possibility is frame_arrival */
        from_physical_layer(&r); /* go get the inbound frame */
        to_network_layer(&r.info); /* pass the data to the network layer */
        to_physical_layer(&s);    /* send a dummy frame to awaken sender */
    }
}
```

A simplex stop-and-wait protocol.

# Sliding Window Protocols (1)

```
/* Protocol 3 (PAR) allows unidirectional data flow over an unreliable channel. */
#define MAX_SEQ 1 /* must be 1 for protocol 3 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
    seq_nr next_frame_to_send; /* seq number of next outgoing frame */
    frame s; /* scratch variable */
    packet buffer; /* buffer for an outbound packet */
    event_type event;

    . . .
```

# Sliding Window Protocols (2)

```
next_frame_to_send = 0;           /* initialize outbound sequence numbers */
from_network_layer(&buffer);      /* fetch first packet */
while (true) {
    s.info = buffer;              /* construct a frame for transmission */
    s.seq = next_frame_to_send;   /* insert sequence number in frame */
    to_physical_layer(&s);        /* send it on its way */
    start_timer(s.seq);           /* if answer takes too long, time out */
    wait_for_event(&event);       /* frame_arrival, cksum_err, timeout */
    if (event == frame_arrival) {
        from_physical_layer(&s);  /* get the acknowledgement */
        if (s.ack == next_frame_to_send) {
            stop_timer(s.ack);    /* turn the timer off */
            from_network_layer(&buffer); /* get the next one to send */
            inc(next_frame_to_send); /* invert next_frame_to_send */
        }
    }
}
}
```

...

# Sliding Window Protocols (3)

```
void receiver3(void)
{
    seq_nr frame_expected;
    frame r, s;
    event_type event;

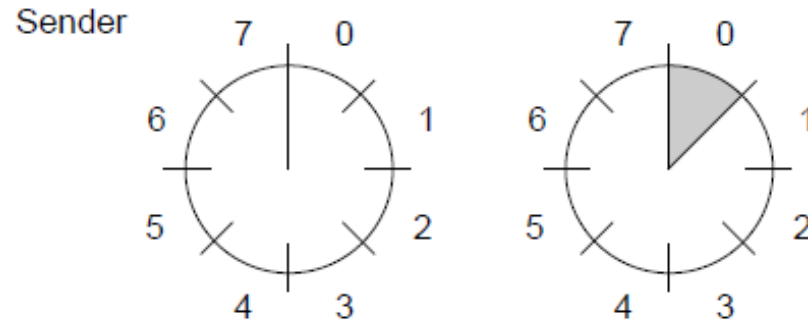
    frame_expected = 0;
    while (true) {
        wait_for_event(&event);
        if (event == frame_arrival) {
            from_physical_layer(&r);
            if (r.seq == frame_expected) {
                to_network_layer(&r.info);
                inc(frame_expected);
            }
            s.ack = 1 - frame_expected;
            to_physical_layer(&s);
        }
    }
}
```

*/\* possibilities: frame\_arrival, cksum\_err \*/*  
*/\* a valid frame has arrived \*/*  
*/\* go get the newly arrived frame \*/*  
*/\* this is what we have been waiting for \*/*  
*/\* pass the data to the network layer \*/*  
*/\* next time expect the other sequence nr \*/*  
  
*/\* tell which frame is being acked \*/*  
*/\* send acknowledgement \*/*

# Sliding Window Protocols (4)

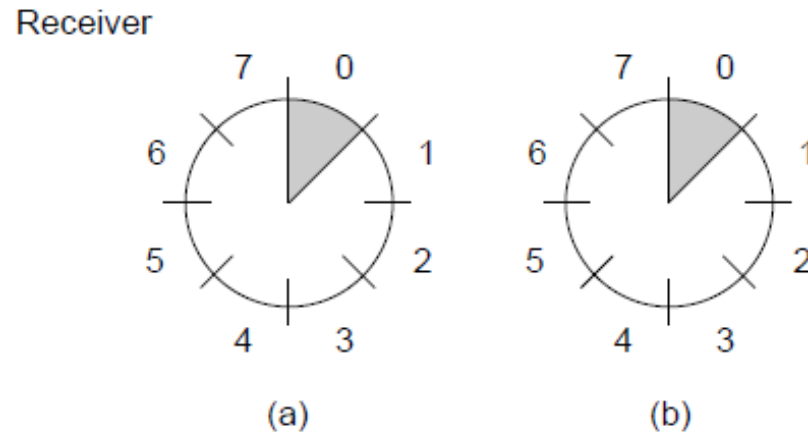
A sliding window

(a) If

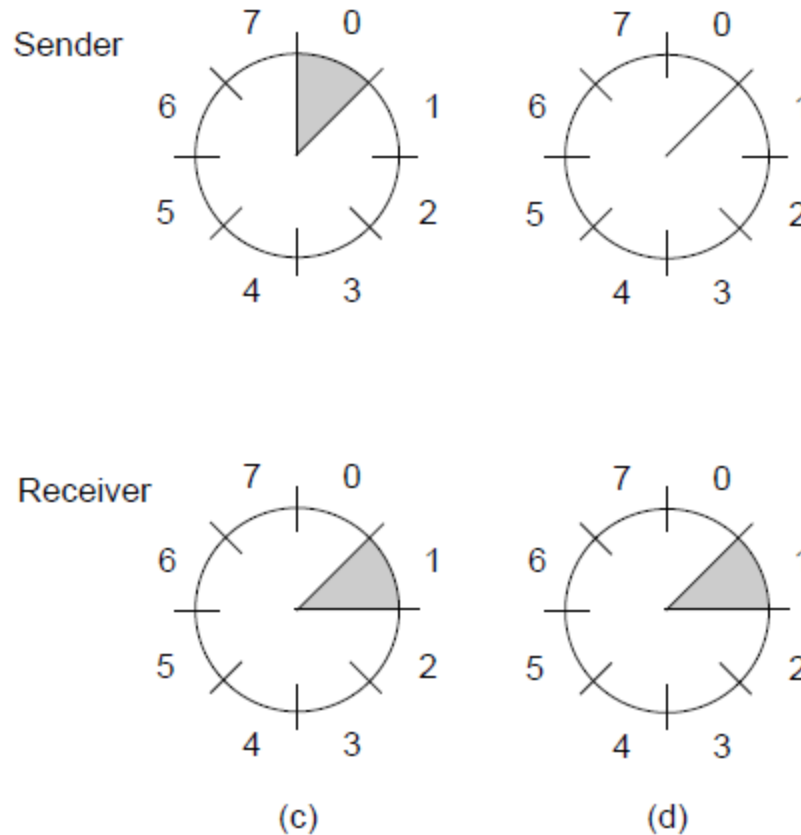


with a 3-bit

frame has



# Sliding Window Protocols (5)



A sliding window of size 1, with a 3-bit sequence number

(c) After the first frame has been received. (d)



# One-Bit Sliding Window Protocol (1)

```
/* Protocol 4 (Sliding window) is bidirectional. */

#define MAX_SEQ 1 /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
    seq_nr next_frame_to_send; /* 0 or 1 only */
    seq_nr frame_expected; /* 0 or 1 only */
    frame r, s; /* scratch variables */
    packet buffer; /* current packet being sent */
    event_type event;

    next_frame_to_send = 0; /* next frame on the outbound stream */
    frame_expected = 0; /* frame expected next */
    from_network_layer(&buffer); /* fetch a packet from the network layer */
    s.info = buffer; /* prepare to send the initial frame */
    s.seq = next_frame_to_send; /* insert sequence number into frame */
    s.ack = 1 - frame_expected; /* piggybacked ack */
    to_physical_layer(&s); /* transmit the frame */
    start_timer(s.seq); /* start the timer running */
    . . .
}
```

A 1-bit sliding window protocol.

# One-Bit Sliding Window Protocol (2)

```
while (true) {
    wait_for_event(&event);           /* frame_arrival, cksum_err, or timeout */
    if (event == frame_arrival) {     /* a frame has arrived undamaged */
        from_physical_layer(&r);      /* go get it */
        if (r.seq == frame_expected) { /* handle inbound frame stream */
            to_network_layer(&r.info); /* pass packet to network layer */
            inc(frame_expected);       /* invert seq number expected next */
        }
        if (r.ack == next_frame_to_send) { /* handle outbound frame stream */
            stop_timer(r.ack);         /* turn the timer off */
            from_network_layer(&buffer); /* fetch new pkt from network layer */
            inc(next_frame_to_send);   /* invert sender's sequence number */
        }
    }
}

. . .
```

A 1-bit sliding window protocol.

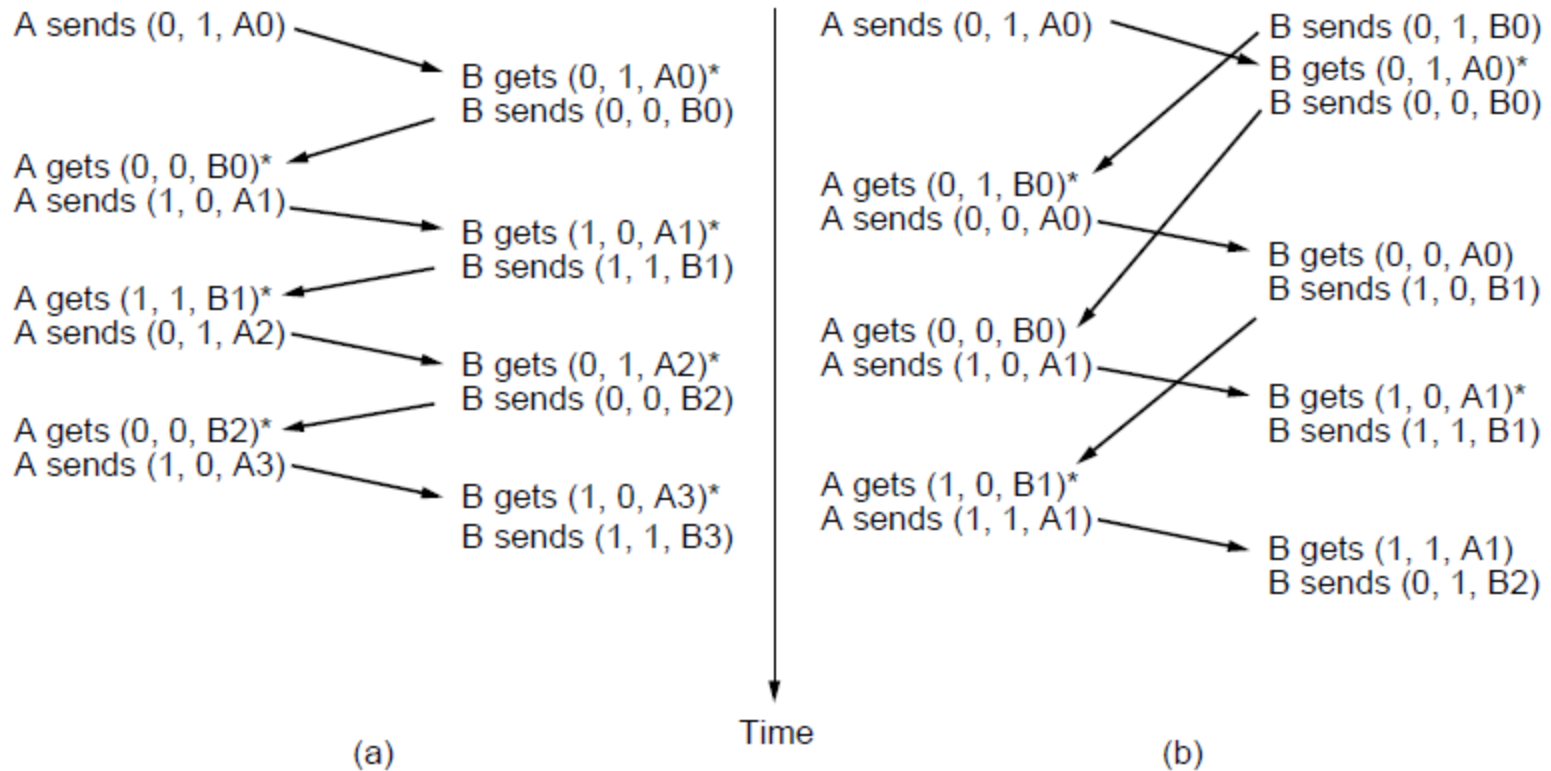
# One-Bit Sliding Window Protocol (3)

```
s.info = buffer;
s.seq = next_frame_to_send;
s.ack = 1 - frame_expected;
to_physical_layer(&s);
start_timer(s.seq);
}
}
```

*/\* construct outbound frame \*/*  
*/\* insert sequence number into it \*/*  
*/\* seq number of last received frame \*/*  
*/\* transmit a frame \*/*  
*/\* start the timer running \*/*

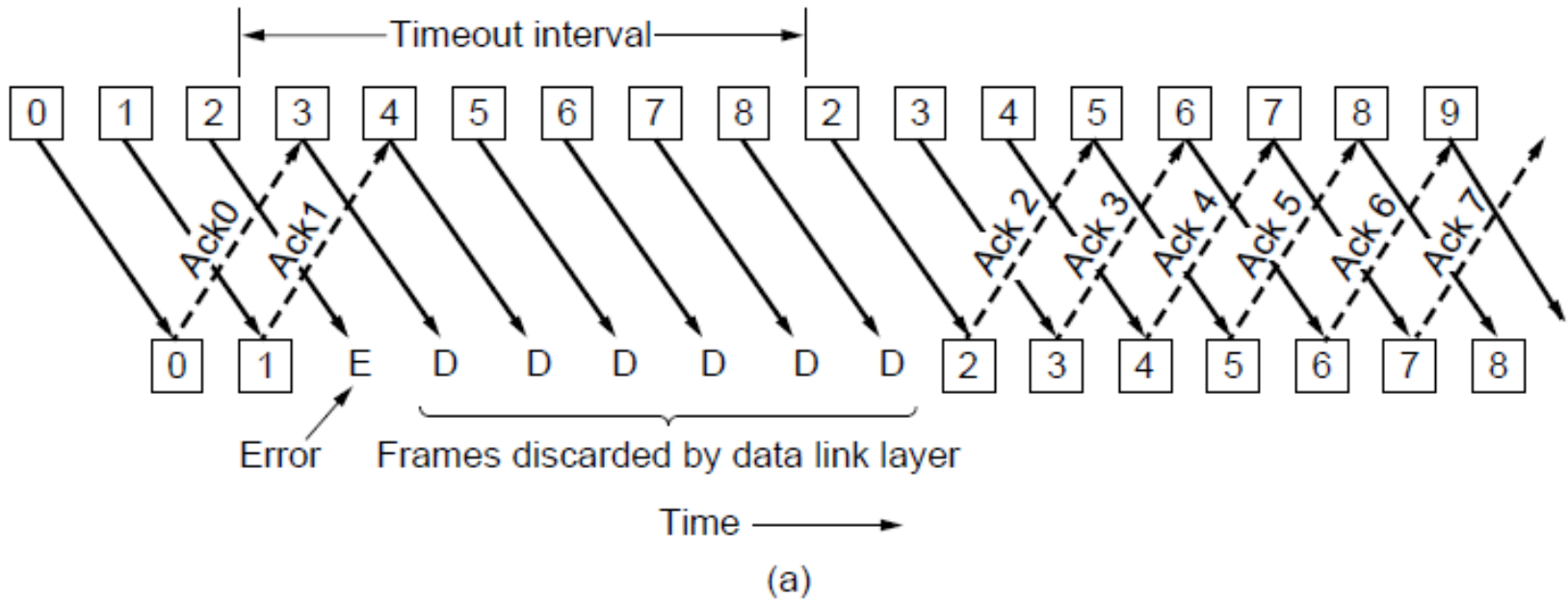
A 1-bit sliding window protocol.

# One-Bit Sliding Window Protocol (4)

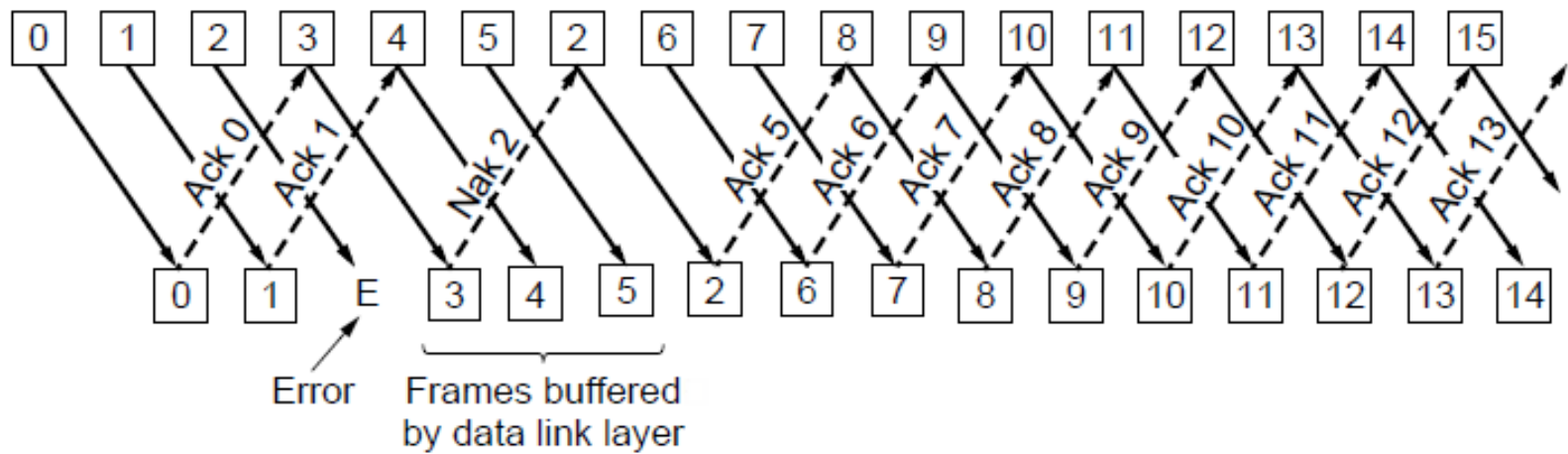


Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number).

# Protocol Using Go-Back-N (1)



# Protocol Using Go-Back-N (2)



(b)

# Protocol Using Go-Back-N (3)

/\* Protocol 5 (Go-back-n) allows multiple outstanding frames. The sender may transmit up to MAX\_SEQ frames without waiting for an ack. In addition, unlike in the previous protocols, the network layer is not assumed to have a new packet all the time. Instead, the network layer causes a network\_layer\_ready event when there is a packet to send. \*/

```
#define MAX_SEQ 7
```

```
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
```

```
#include "protocol.h"
```

```
static boolean between(seq_nr a, seq_nr b, seq_nr c)
```

```
{
```

```
/* Return true if a <= b < c circularly; false otherwise. */
```

```
if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
```

```
    return(true);
```

```
else
```

```
    return(false);
```

```
}
```

```
 . . .
```

# Protocol Using Go-Back-N (4)

```
static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
  /* Construct and send a data frame. */
  frame s;                               /* scratch variable */

  s.info = buffer[frame_nr];             /* insert packet into frame */
  s.seq = frame_nr;                       /* insert sequence number into frame */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1); /* piggyback ack */
  to_physical_layer(&s);                  /* transmit the frame */
  start_timer(frame_nr);                  /* start the timer running */
}
```

• • •



# Protocol Using Go-Back-N (5)

```
void protocol5(void)
{
    seq_nr next_frame_to_send;
    seq_nr ack_expected;
    seq_nr frame_expected;
    frame r;
    packet buffer[MAX_SEQ + 1];
    seq_nr nbuffered;
    seq_nr i;
    event_type event;

    . . .

    /* MAX_SEQ > 1; used for outbound stream */
    /* oldest frame as yet unacknowledged */
    /* next frame expected on inbound stream */
    /* scratch variable */
    /* buffers for the outbound stream */
    /* number of output buffers currently in use */
    /* used to index into the buffer array */
}
```

# Protocol Using Go-Back-N (6)

## A sliding window protocol using go-back-

```
enable_network_layer();           /* allow network_layer_ready events */
ack_expected = 0;                 /* next ack expected inbound */
next_frame_to_send = 0;          /* next frame going out */
frame_expected = 0;              /* number of frame expected inbound */
nbuffered = 0;                   /* initially no packets are buffered */

while (true) {
    wait_for_event(&event);       /* four possibilities: see event_type above */

```

...

# Protocol Using Go-Back-N (7)

```
switch(event) {
  case network_layer_ready:          /* the network layer has a packet to send */
    /* Accept, save, and transmit a new frame. */
    from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
    nbuffered = nbuffered + 1;      /* expand the sender's window */
    send_data(next_frame_to_send, frame_expected, buffer); /* transmit the frame */
    inc(next_frame_to_send);        /* advance sender's upper window edge */
    break;

  case frame_arrival:               /* a data or control frame has arrived */
    from_physical_layer(&r);        /* get incoming frame from physical layer */

    if (r.seq == frame_expected) {
      /* Frames are accepted only in order. */
      to_network_layer(&r.info);    /* pass packet to network layer */
      inc(frame_expected);          /* advance lower edge of receiver's window */
    }

  . . .
}
```

# Protocol Using Go-Back-N (8)

```
/* Ack n implies n - 1, n - 2, etc. Check for this. */
while (between(ack_expected, r.ack, next_frame_to_send)) {
    /* Handle piggybacked ack. */
    nbuffered = nbuffered - 1;      /* one frame fewer buffered */
    stop_timer(ack_expected);      /* frame arrived intact; stop timer */
    inc(ack_expected);              /* contract sender's window */
}
break;

case cksum_err: break;              /* just ignore bad frames */

case timeout:                       /* trouble; retransmit all outstanding frames */
    next_frame_to_send = ack_expected; /* start retransmitting here */
    for (i = 1; i <= nbuffered; i++) {
        send_data(next_frame_to_send, frame_expected, buffer); /* resend frame */
        inc(next_frame_to_send); /* prepare to send the next one */
    }
}
}
```

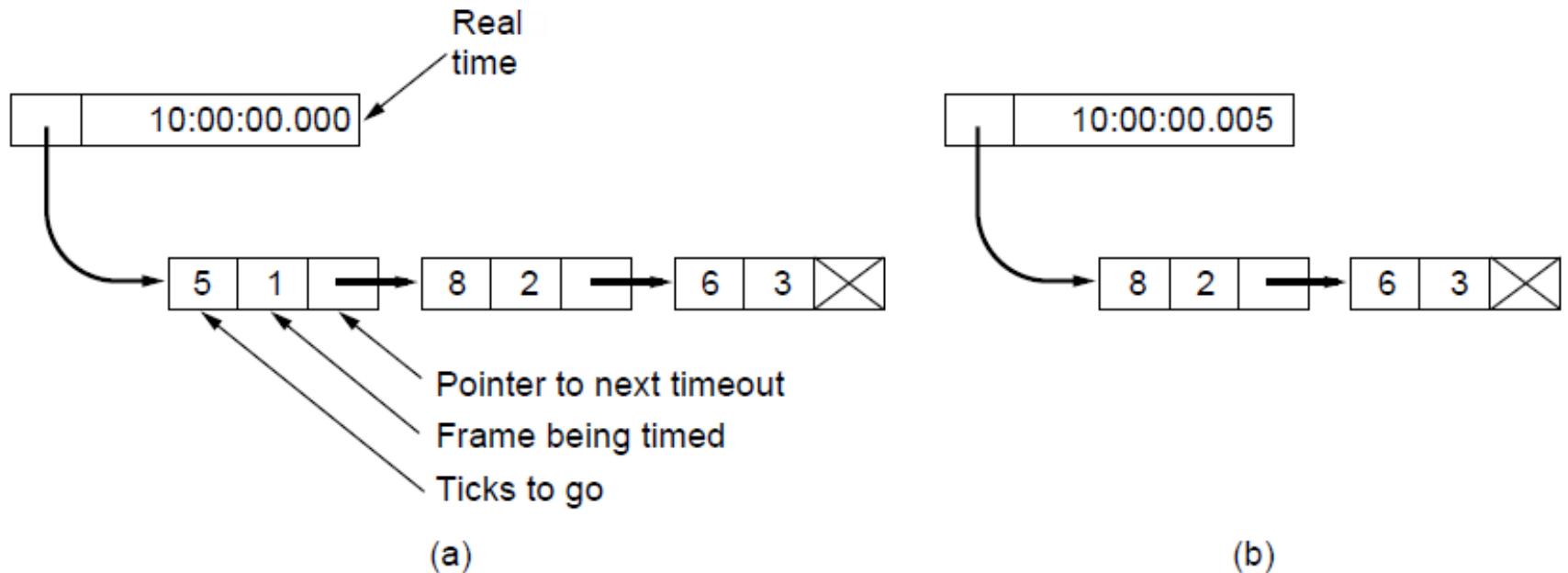
...

# Protocol Using Go-Back-N (9)

A sliding window protocol using go-back-

```
if (nbuffered < MAX_SEQ)
    enable_network_layer();
else
    disable_network_layer();
}
```

# Protocol Using Go-Back-N (10)



Simulation of multiple timers in software. (a) The queued timeouts (b) The situation after the first timeout

# Protocol Using Selective Repeat (1)

*/\* Protocol 6 (Selective repeat) accepts frames out of order but passes packets to the network layer in order. Associated with each outstanding frame is a timer. When the timer expires, only that frame is retransmitted, not all the outstanding frames, as in protocol 5. \*/*

```
/* should be 2^n - 1 */
#define MAX_SEQ 7
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout} event_type;
#include "protocol.h"
boolean no_nak = true; /* no nak has been sent yet */
seq_nr oldest_frame = MAX_SEQ + 1; /* initial value is only for the simulator */

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Same as between in protocol 5, but shorter and more obscure. */
return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}

. . .
```

# Protocol Using Selective Repeat (2)

```
static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
/* Construct and send a data, ack, or nak frame. */
  frame s;                               /* scratch variable */

  s.kind = fk;                            /* kind == data, ack, or nak */
  if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
  s.seq = frame_nr;                        /* only meaningful for data frames */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
  if (fk == nak) no_nak = false;          /* one nak per frame, please */
  to_physical_layer(&s);                  /* transmit the frame */
  if (fk == data) start_timer(frame_nr % NR_BUFS);
  stop_ack_timer();                        /* no need for separate ack frame */
}
```

• • •



# Protocol Using Selective Repeat (3)

```
void protocol6(void)
{
    seq_nr ack_expected;
    seq_nr next_frame_to_send;
    seq_nr frame_expected;
    seq_nr too_far;
    int i;
    frame r;
    packet out_buf[NR_BUFS];
    packet in_buf[NR_BUFS];
    boolean arrived[NR_BUFS];
    seq_nr nbuffered;
    event_type event;

    /* lower edge of sender's window */
    /* upper edge of sender's window + 1 */
    /* lower edge of receiver's window */
    /* upper edge of receiver's window + 1 */
    /* index into buffer pool */
    /* scratch variable */
    /* buffers for the outbound stream */
    /* buffers for the inbound stream */
    /* inbound bit map */
    /* how many output buffers currently used */

```

• • •

# Protocol Using Selective Repeat (4)

## A sliding window protocol using selective

```
enable_network_layer();           /* initialize */
ack_expected = 0;                 /* next ack expected on the inbound stream */
next_frame_to_send = 0;          /* number of next outgoing frame */
frame_expected = 0;
too_far = NR_BUFS;
nbuffered = 0;                   /* initially no packets are buffered */
for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
```

...

# Protocol Using Selective Repeat (5)

```
while (true) {  
    wait_for_event(&event);           /* five possibilities: see event_type above */  
    switch(event) {  
        case network_layer_ready:    /* accept, save, and transmit a new frame */  
            nbuffered = nbuffered + 1; /* expand the window */  
            from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]); /* fetch new packet */  
            send_frame(data, next_frame_to_send, frame_expected, out_buf); /* transmit the frame */  
            inc(next_frame_to_send); /* advance upper window edge */  
            break;
```

...

# Protocol Using Selective Repeat (6)

```
case frame_arrival:          /* a data or control frame has arrived */
  from_physical_layer(&r);   /* fetch incoming frame from physical layer */
  if (r.kind == data) {
    /* An undamaged frame has arrived. */
    if ((r.seq != frame_expected) && no_nak)
      send_frame(nak, 0, frame_expected, out_buf); else start_ack_timer();
    if (between(frame_expected,r.seq,too_far) && (arrived[r.seq%NR_BUFS]==false)) {
      /* Frames may be accepted in any order. */
      arrived[r.seq % NR_BUFS] = true;    /* mark buffer as full */
      in_buf[r.seq % NR_BUFS] = r.info;   /* insert data into buffer */
    }
  }
  . . .
```

# Protocol Using Selective Repeat (7)

```
while (arrived[frame_expected % NR_BUFS]) {
    /* Pass frames and advance window. */
    to_network_layer(&in_buf[frame_expected % NR_BUFS]);
    no_nak = true;
    arrived[frame_expected % NR_BUFS] = false;
    inc(frame_expected);    /* advance lower edge of receiver's window */
    inc(too_far);          /* advance upper edge of receiver's window */
    start_ack_timer();     /* to see if a separate ack is needed */
}
}
...

```

# Protocol Using Selective Repeat (8)

```
if((r.kind==nak) && between(ack_expected,(r.ack+1)%(MAX_SEQ+1),next_frame_to_send))
    send_frame(data, (r.ack+1) % (MAX_SEQ + 1), frame_expected, out_buf);

while (between(ack_expected, r.ack, next_frame_to_send)) {
    nbuffered = nbuffered - 1;          /* handle piggybacked ack */
    stop_timer(ack_expected % NR_BUFS); /* frame arrived intact */
    inc(ack_expected);                  /* advance lower edge of sender's window */
}
break;

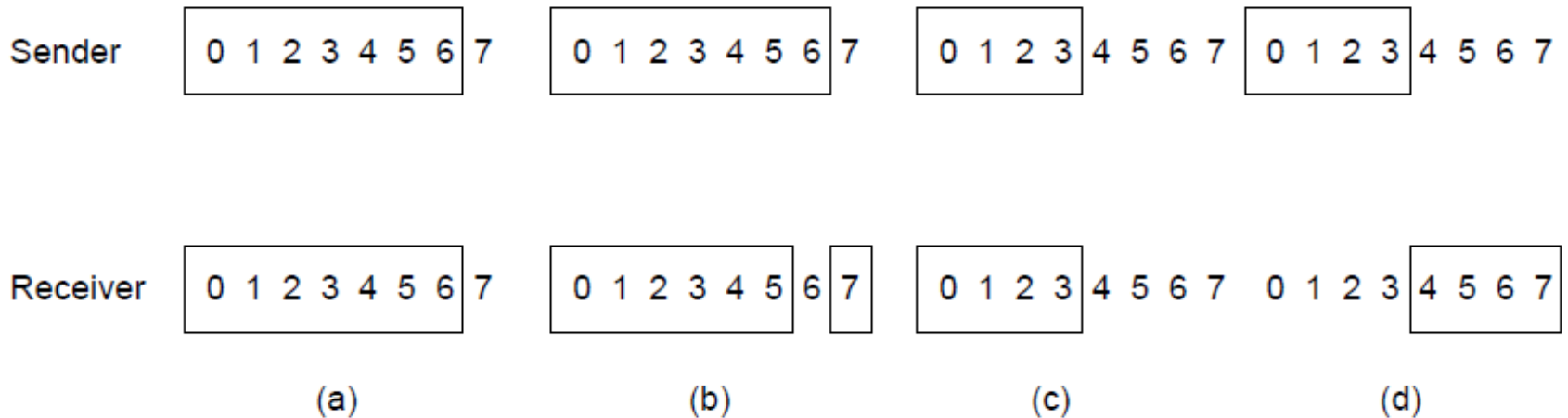
case cksum_err:
    if (no_nak) send_frame(nak, 0, frame_expected, out_buf); /* damaged frame */
    break;
. . .
```

# Protocol Using Selective Repeat (9)

## A sliding window protocol using selective

```
case timeout:
    send_frame(data, oldest_frame, frame_expected, out_buf); /* we timed out */
    break;
case ack_timeout:
    send_frame(ack,0,frame_expected, out_buf); /* ack timer expired; send ack */
}
if (nbuffered < NR_BUFS) enable_network_layer(); else disable_network_layer();
}
```

# Protocol Using Selective Repeat (10)



a) Initial situation with a window of size 7

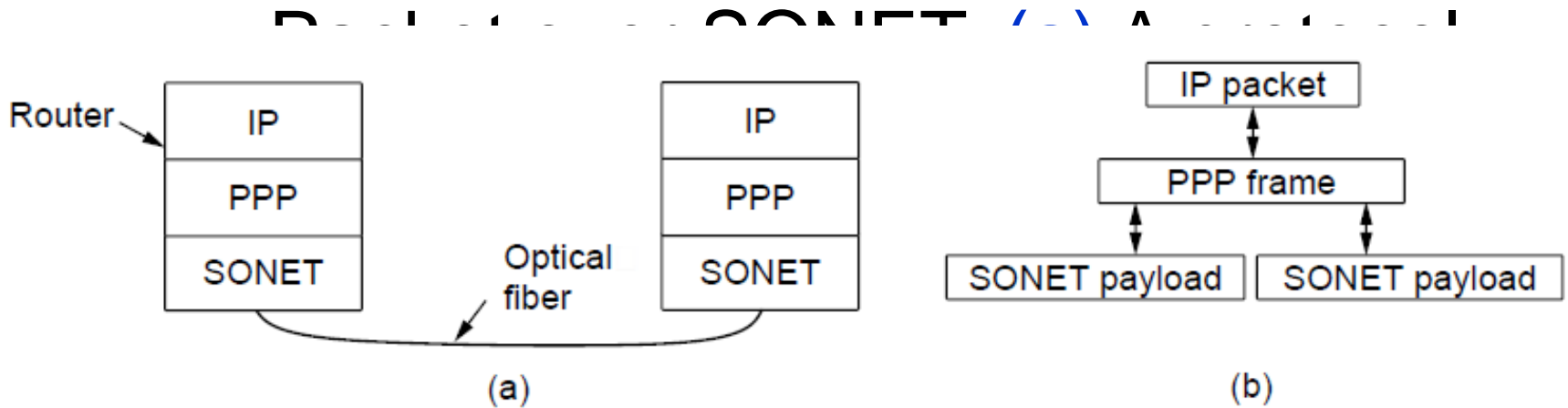
- After 7 frames sent and received but not acknowledged.
- Initial situation with a window size of 4.
- After 4 frames sent and received but not



# Example Data Link Protocols

1. Packet over SONET
2. ADSL (Asymmetric Digital Subscriber Loop)

# Packet over SONET (1)



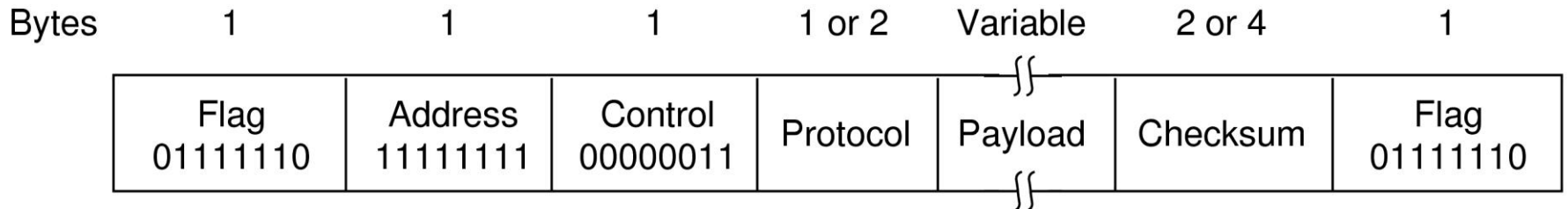
# Packet over SONET (2)

## PPP Features

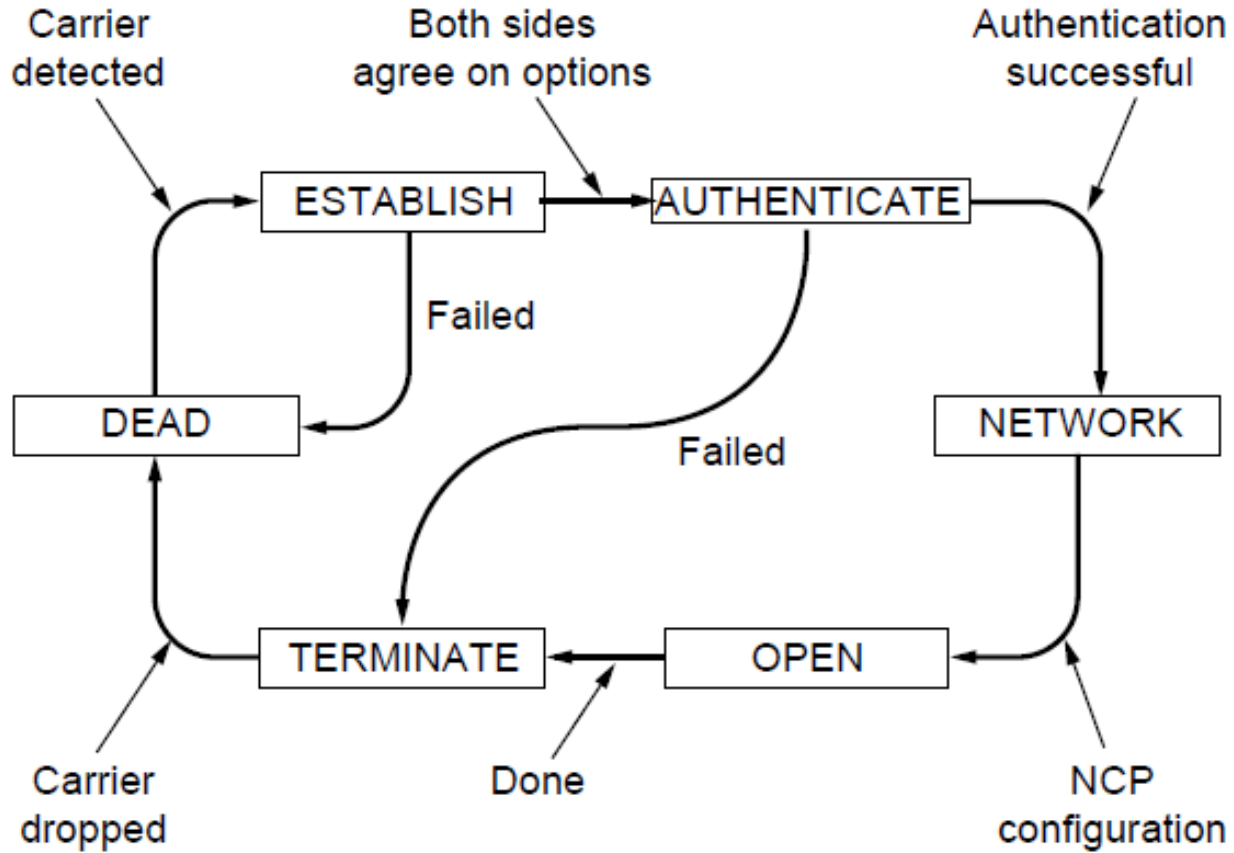
1. Separate packets, error detection
2. Link Control Protocol
3. Network Control Protocol

# Packet over SONET (3)

The PPP full frame format for unnumbered mode operation

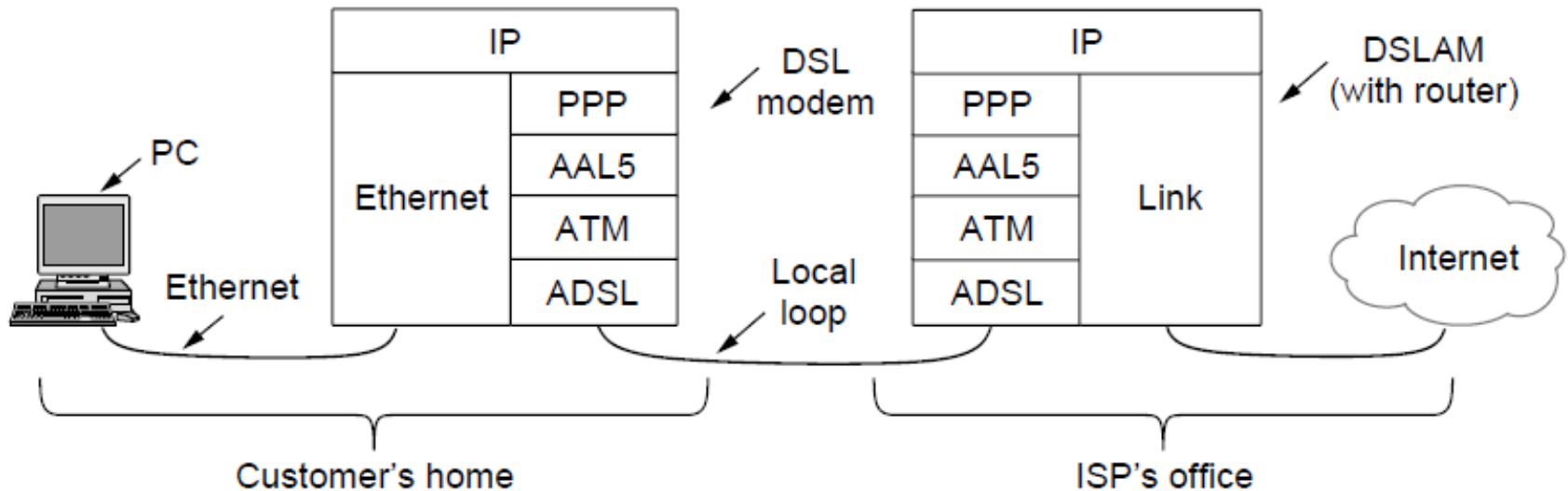


# Packet over SONET (1)



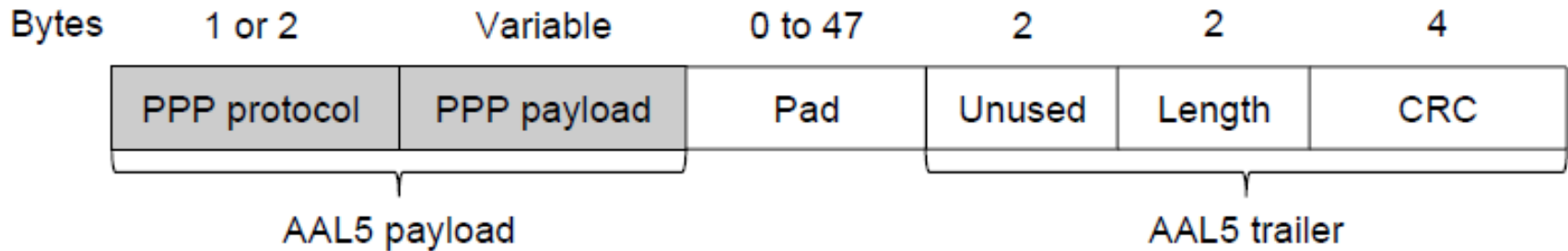
up

# ADSL (Asymmetric Digital Subscriber Loop) (1)



# ADSL (Asymmetric Digital Subscriber Loop) (1)

## AAL5 frame carrying PPP data



# Data Link Control



# Outline

- ◆ Media Access Control
- ◆ Transmission Error Control
  - Parity checking
  - Longitudinal redundancy checking (LRC)
  - Polynomial checking: check sum, CRC
- ◆ Data Link Layer Protocols
  - Asynchronous protocols
  - Ethernet
  - HDLC
  - Token Ring
  - Point-to-point Protocol

# Three Functions of Data Link Layer Protocols

A data link protocol provides three functions:

- ◆ Controls when computers transmit (media access control).
- ◆ Detects and corrects transmission errors (error control).
- ◆ Identifies the start and end of a message (message delineation).

# Media Access Control

Media access control (MAC) refers to the need to control circuits when devices transmit.

Two types of MAC:

- Polling
- Contention

# Transmission Error Control

- ◆ All transmission media have potential for introduction of errors
- ◆ All data link layer protocols must provide method for controlling errors
- ◆ Error control process has two components
  - Error detection
  - Error correction

# Error Control in Networks

There are two types of errors associated with networks.

- Human errors, controlled by application programs
- Network errors, controlled by the network

There are two categories of network errors.

- Corrupted data (that have been changed)
- Lost data

# What are Network Errors?

Network errors are a fact of life in data communications networks.

Normally errors occur in bursts.

- In a burst error, more than one data bit is changed by the error-causing condition.
- Errors are not uniformly distributed, regardless of error rate statistics.

Dial-up lines are more prone to errors because they have less stable parameters.

# Network Error

- ◆ One bit error in 1000 bits results in an error rate of 0.001. However, it also means that one character made up of 8-bit is not correct. The character error rate becomes  $1/125$ . It is 8 times of the bit error rate.

# Network Error

The fact that errors tend to be clustered in bursts rather than evenly dispersed is both good and bad:

- Good: long period error-free transmission
- Bad: it is more difficult to recover the data from the errors.
  - I h\*ve a p\*ncil in my p\*cket
  - I have a \*e\*\*il in my pocket
  - Give me your \*oney, or I' ll ki\*\* you.



# Error Detection

It is possible to develop data transmission methodologies that give very high error detection and correction performance.

The only way to do error detection and correction is to send extra data with each message.

In general, the larger the amount of error detection data sent, the greater the ability to detect an error.

# Error Detection

There are three common error detection methods.

- Parity Checking
- Longitudinal redundancy checking
- Polynomial checking
  - ◆ Checksum
  - ◆ Cyclic Redundancy

# Error Detection: Parity Bits

- ◆ Bit added to each character to make all bits add up to an even number (even parity) or odd number (odd parity)
- ◆ Good for detecting single-bit errors only
- ◆ High overhead (one extra bit per 7-bit character=12.5%)

# Parity Checking

One of the oldest and simplest method, parity checking adds 1 additional bit to each byte in the message. The value of this parity bit is dependent on the number of 1's in each byte transmitted. Even parity causes the sum of all bits (including the parity bit) to be even. Odd parity causes the sum to be odd.

Unfortunately if two bits are erroneous, the parity checking will fail. Parity checking results in about a 50% reliability rate.

# Parity Checking

Assume we are using **even parity** with 7-bit ASCII.

The letter V in 7-bit ASCII is encoded as 0110101.

Because there are four 1s (an even number), parity is set to zero.

This would be transmitted as: 01101010.

Assume we are using **odd parity** with 7-bit ASCII.

The letter W in 7-bit ASCII is encoded as 1010111 (Oct127).

Because there are five 1s (an odd number), parity is set to zero.

This would be transmitted as: 10101110.

# Longitudinal Redundancy Checking (LRC)

LRC was developed to overcome the problem with parity's low probability of detection.

LRC adds one additional character, called the block check character (BCC) to the end of the entire message or packet of data.

The value of the BCC is calculated much like the Parity bit, but for the entire message. Results in a 98% reliability rate.

# Longitudinal Redundancy Checking

For example, suppose we were to send the message “DATA” using odd parity and LRC with 7-bit ASCII:

Letter	ASCII	Parity bit
D	1000100	1
A	1000001	1
T	1010100	0
A	1000001	1
BCC	1101111	1

(Note that the parity bit in the BCC is determined by parity, not LRC.)

# Polynomial Checking

Like LRC, polynomial checking adds 1 or more characters to the end of the message based on a mathematical algorithm.

With checksum, 1 byte is added to the end of the message. It is obtained by summing the message values, and dividing by 255. The remainder is the checksum. (95% effective)

With Cyclical Redundancy Check (CRC), 8, 16, 24 or 32 bits are added, computed by calculating a remainder to a division problem.



# Error Detection: Cyclic Redundancy Check (CRC)

- ◆ Data in frame treated as a single binary number, divided by a unique prime binary, and remainder is attached to frame
- ◆ 17-bit divisor leaves 16-bit remainder, 33-bit divisor leaves 32-bit remainder
- ◆ For a CRC of length  $N$ , errors undetected are  $2^{-N}$
- ◆ Overhead is low (1-3%)

# Flow Control

- ◆ Necessary when data is being sent faster than it can be processed by receiver
- ◆ The simplest, most effective, least expensive, and most commonly used method for transmission error control.
- ◆ Using retransmission.
  - Computer to printer is typical setting
  - Can also be from computer to computer, when a processing program is limited in capacity

# Stop-and-Wait Flow Control

- ◆ Also called Stop-and-Wait Automatic Repeat reQuest (ARQ)
- ◆ It is the simplest form
- ◆ Source may not send new frame until receiver acknowledges the frame already sent
- ◆ Very inefficient, especially when a single message is broken into separate frames, or when the data link is long enough for significant delays to be introduced

# Error Correction

- ◆ A receiver that detects an error simply asks the sender to retransmit the message until it is received without error.
  - Lost frame
  - Damaged frame
- ◆ With Stop and Wait ARQ the sender stops and waits for a response from the receiver after each message or data package.
- ◆ Responses are:
  - Error detection
  - Positive acknowledgment (ACK)
  - Negative acknowledgment and retransmission (NAK)
  - Retransmission after time-out

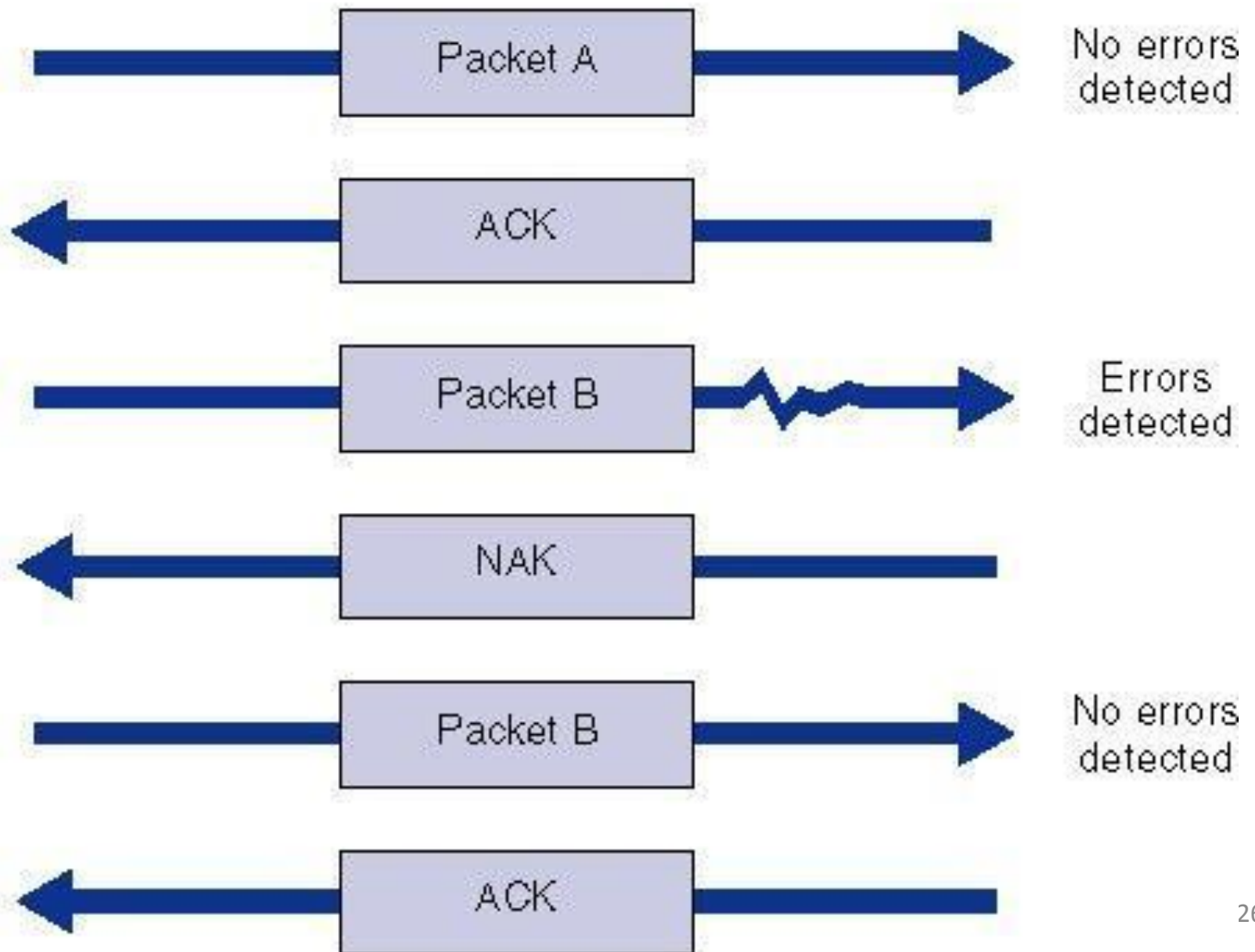
# Stop-and-Wait ARQ

- ◆ One frame received and handled at a time
- ◆ If frame is damaged, receiver discards it and sends no acknowledgment
  - Sender uses timer to determine whether or not to retransmit
  - Sender must keep a copy of transmitted frame until acknowledgment is received
- ◆ If acknowledgment is damaged, sender will know it because of numbering

# Stop and wait ARQ

Sender

Receiver



# Sliding-Window Flow Control

- Also called Continuous ARQ
- Allows multiple frames to be in transit
- Receiver sends acknowledgement with sequence number of anticipated frame
- Sender maintains list of sequence numbers it can send, receiver maintains list of sequence numbers it can receive
- ACK (acknowledgement, or RR – Receiver Ready) supplemented with RNR (receiver not ready)

# Go-Back-N ARQ

- ◆ Uses sliding-window flow control
- ◆ When receiver detects error, it sends negative acknowledgment (REJ)
- ◆ Sender must begin transmitting again from rejected frame
- ◆ Transmitter must keep a copy of all transmitted frames



# Error Correction via Retransmission

- ◆ With Continuous ARQ the sender does not wait for acknowledgement before sending next message. If it receives an REJ, it retransmits the needed messages.
- ◆ The packets that are retransmitted may be only those containing an error (called *Link Access Protocol for Modems* (LAP-M)), or may be the first packet with an error and all those that followed it (call *Go-Back-N ARQ*)
- ◆ Continuous ARQ is a full-duplex transmission technique.

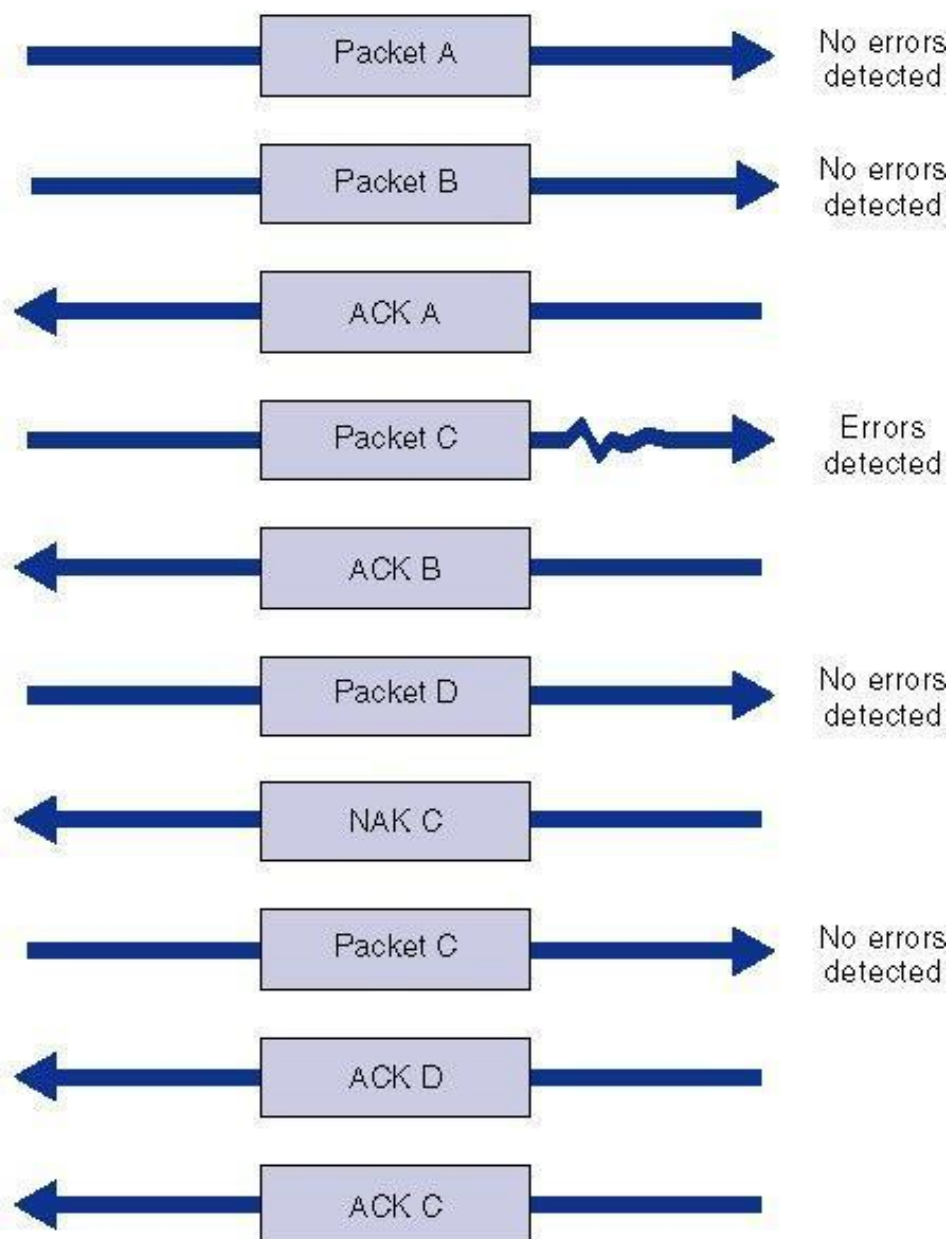
# Packet Loss

- How it could be? Two possibilities:
  - The receiver did not receive the packet
  - The sender did not receive the ACK message
- What can we do with packet loss?
  - Use “Time-out” to detect packet loss
  - Re-send the packet if there is a “time-out”

# Continuous ARQ

Sender

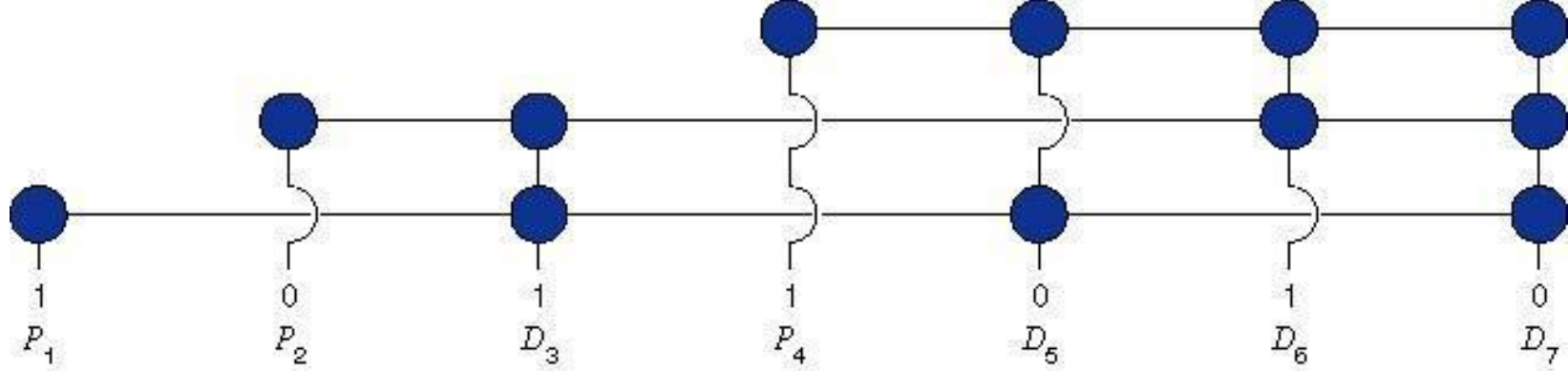
Receiver



# \*Forward Error Correction

Forward error correction uses codes containing sufficient redundancy to prevent errors by detecting and correcting them at the receiving end without retransmission of the original message.

- ◆ Hamming code is capable of correcting 1 bit error.
- ◆ More sophisticated techniques, such as Reed-Solomon, are commonly used.
- ◆ commonly used in satellite transmission.
- ◆ Chip implementations of FEC has been used in modems, e.g. V.34.



Checking relations between parity bits ( $P$ ) and data bits ( $D$ )

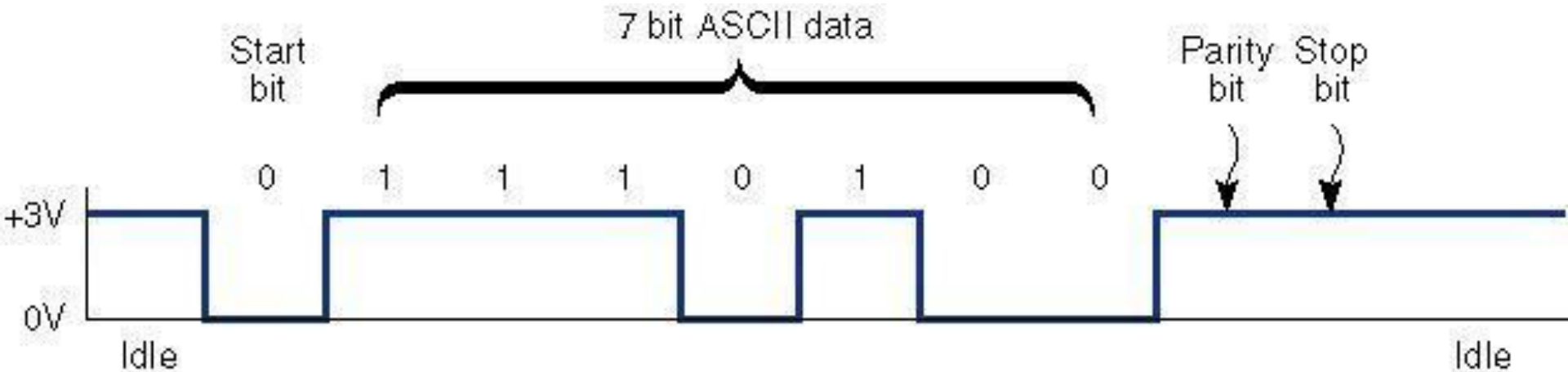
0 = Corresponding parity check is correct 1 = Corresponding parity check fails				Determines in which bit the error occurred
$P_4$	$P_2$	$P_1$		
0	0	0	→	no error
0	0	1	→	$P_1$
0	1	0	→	$P_2$
0	1	1	→	$D_3$
1	0	0	→	$P_4$
1	0	1	→	$D_5$
1	1	0	→	$D_6$
1	1	1	→	$D_7$

Interpreting parity bit patterns

# Data Link Control Protocols

- ◆ Asynchronous protocols
- ◆ High-level Data Link Control (HDLC)
- ◆ Ethernet and Token Ring
- ◆ SLIP and PPP

# Asynchronous Transmission



*Start-stop* transmission is a least efficient technology.  
We need to bundle bytes to reach high efficiency transmissions.

# Six Issues in Packet Design

- ◆ Error correction
- ◆ Addressing
- ◆ Delineation
- ◆ The transparency problem
- ◆ Transmission efficiency (packet size vs. bandwidth utilization ratio)
- ◆ Media access control



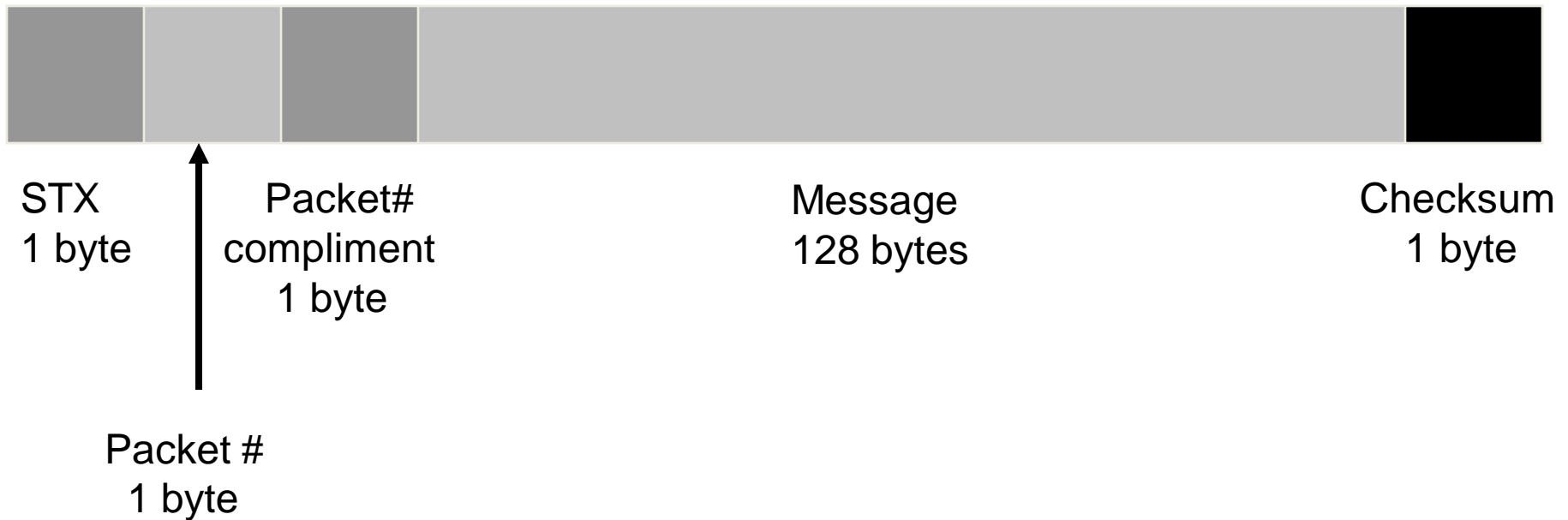
# Asynchronous File Transfer Protocols

In general, microcomputer file transfer protocols are used on asynchronous point-to-point circuits, typically across telephone lines via a modem.

- XMODEM (132 bytes)
  - ◆ XMODEM-CRC (CRC-8)
  - ◆ XMODEM-1K (CRC+1K blocks)
- YMODEM(CRC-16)
- ZMODEM (CRC-32)
- KERMIT (CRC-24)

# Asynchronous FTP

## XMODEM FORMAT



# Asynchronous FTP

## KERMIT

- developed by Columbia University
- support a variety of different packet sizes and error detection methods
- typically 1000bytes/packet with CRC-24 and adjustable during transmission
- uses both stop-and-wait ARQ and continuous ARQ

# Synchronous Transmission

With Synchronous Transmission all the letters or data in one group of data is transmitted at one time as a block of data called a frame or packet.

The start and end of each packet sometimes is marked by adding synchronization characters (SYN) at the start/end of each packet.

# Synchronous Transmission

There are many protocols for synchronous transmission that fall into three broad categories:

- Byte-oriented
- Bit-oriented
- Byte-count

# High-Level Data Link Control (HDLC)

- The OSI's data link protocol
- A bit-oriented protocol
- On transmitting side, HDLC receives data from an application, and delivers it to the receiver on the other side of the link
- On the receiving side, HDLC accepts the data and delivers it to the higher level application layer
- Both modules exchange control information, encoded into a frame

(IBM's SDLC has similar features and the format of HDLC)

# HDLC Frame Structure

- ◆ Flag: 01111110, at start and end
- ◆ Address: secondary station (for multidrop configurations)
- ◆ Information: the data to be transmitted
- ◆ Frame check sequence: 16- or 32-bit CRC
- ◆ Control: purpose or function of frame
  - Information frames: contain user data
  - Supervisory frames: flow/error control (ACK/ARQ)
  - Unnumbered frames: variety of control functions (see p.131)

# Transparency Problem

- The problem of *transparency*
  - Because HDLC uses 01111110 to mark the start and end of a packet, the character “01111110” will confuse the receiver.
- Solution - *Bit Stuffing*
  - Add a 0 after every five 1s at sender side and delete the 0 at receiver side.



# HDLC Operation

- ◆ Initialization: S-frames specify mode and sequence numbers, U-frames acknowledge
- ◆ Data Transfer: I-frames exchange user data, S-frames acknowledge and provide flow/error control
- ◆ Disconnect: U-frames initiate and acknowledge

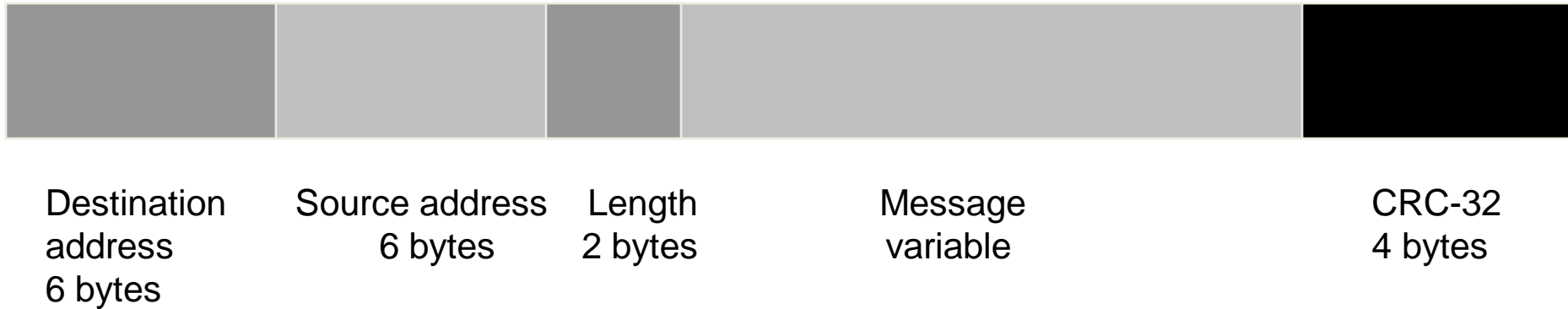
# Ethernet (IEEE 802.3)

Ethernet (IEEE 802.3) was conceived by Bob Metcalfe in 1973 and developed jointly by Digital, Intel, and Xerox in the 1970s.

- ◆ It is a byte-count protocol. Therefore, Ethernet has no transparency problem because it does not use number of bytes to delineate the message.
- ◆ It uses contention media access control.
- ◆ The address is 6-byte
- ◆ The maximum length of the message is 1492 bytes.
- ◆ CRC-32 frame check.

# Frame Formats

Ethernet (IEEE802.3) format



Token Ring (IEEE802.5) format



# Token Ring

Token Ring (IEEE 802.5) was developed by IBM in the early 1980s, and later became a formal standard of the IEEE. It uses a controlled access media access protocol.

- ◆ It is a byte-oriented protocol
- ◆ Does not suffer the same transparency problems as SDLC
  - Each token ring frame starts and ends with special electrical signal produced in a manner different from any other pattern of bits.
- ◆ The size of the message field is generally limited to 4500 bytes
- ◆ The frame check sequence is 32-bit CRC code.

# Serial Line Internet Protocol

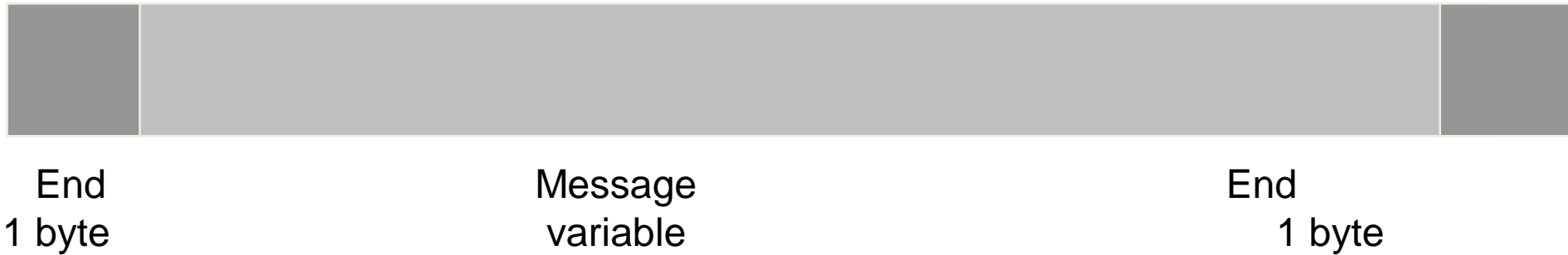
- Serial Line Internet Protocol (SLIP) is a byte-oriented protocol designed to connect two computers using Internet protocols over a point-to-point telephone line. It is developed in 1980s.
  - SLIP suffers the transparency problem because it uses “11000000” as END. So, it replaces the character in the message with “11011001” (ESC) + “11011100”. This trick is also used for ESC character itself, with a ESC + “11011101”.
  - Compressed SLIP (CSLIP) uses compression to reduce the amount of data transmitted.

# Point-to-Point Protocol

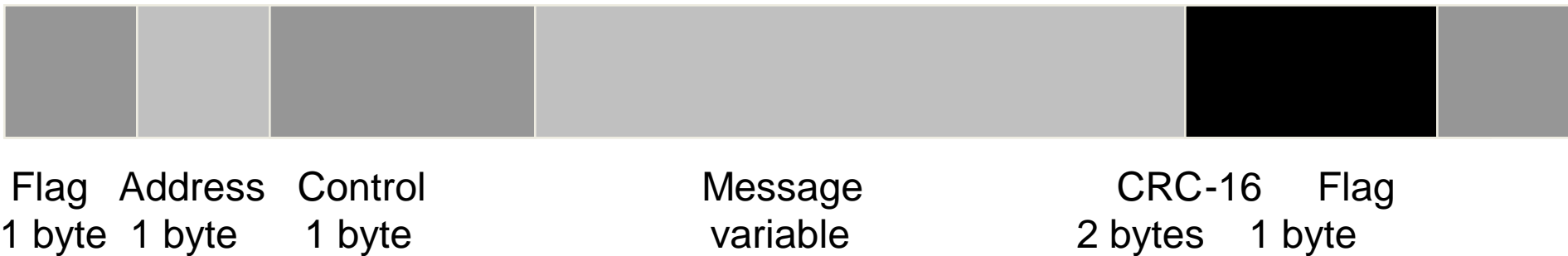
- Point-to-Point Protocol (PPP) is a byte-oriented protocol developed in 1990s as a replacement for SLIP.
  - The packet begins and ends with a flag “01111110”. Therefore, it suffers the transparency problem.
  - CRC-16 for error control
  - Supports network layer protocols other than just Internet protocol.
  - Message length is up to 1500 bytes.

# Packet Formats of SLIP and PPP

SLIP packet layout



PPP packet layout

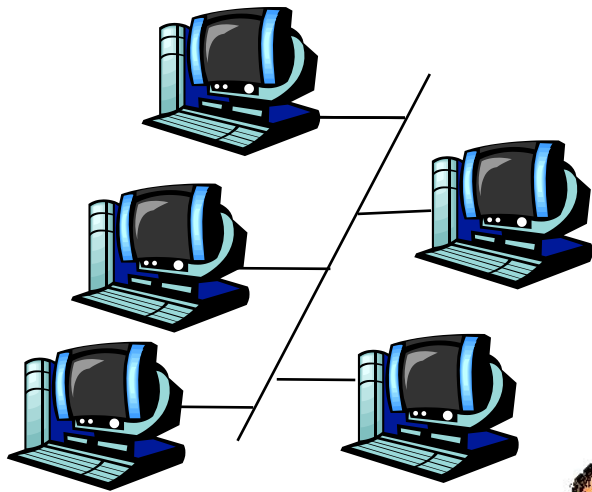


# The Medium Access Control Sublayer



# The Channel Allocation Problem

Which sender can use the broadcast medium?



shared wire (e.g.,  
cabled Ethernet)

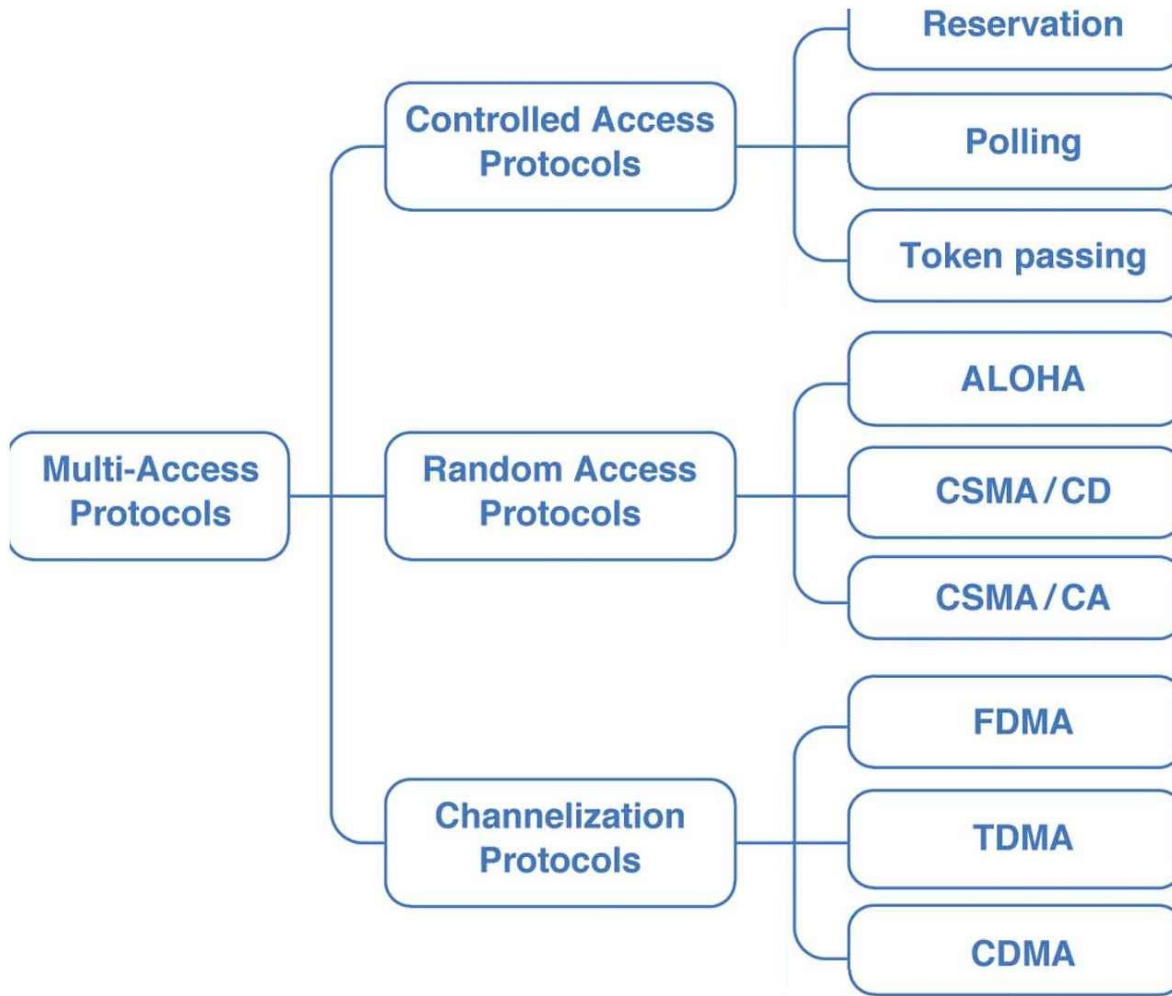


shared RF  
(e.g., 802.11 WiFi)



humans at a  
cocktail party  
(shared air, acoustical)

# Taxonomy



Collision free:  
guaranteed max.  
access time and  
min. data rate

Ethernet

Wireless ethernet

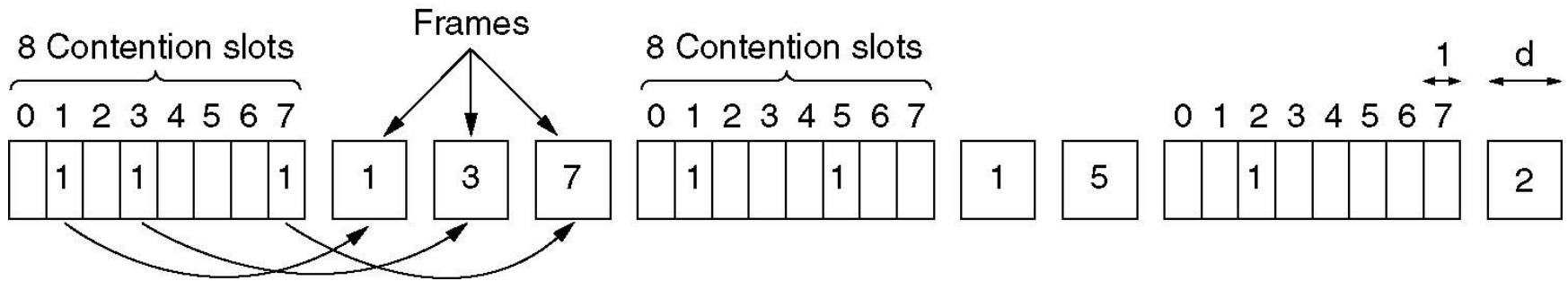
Extension of ch 2

# Explanation

- channelization divides the shared medium in multiple channels:
  - 1 sender/receiver: could use point-to-point protocol
  - more than 1: need broadcast protocol
- random access, 2 or more can send at the same time:
  - collisions can occur
  - some form of random waiting time needed
  - no central controller needed
- controlled access, collisions can not occur
  - guaranteed max. access time and min. data rate

# Controlled Access Protocols

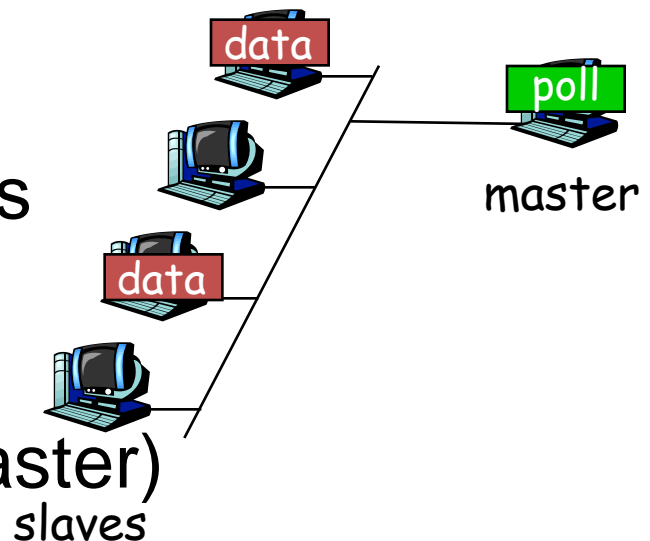
The basic bit-map or reservation protocol.



## Polling

typically used with “dumb” slaves

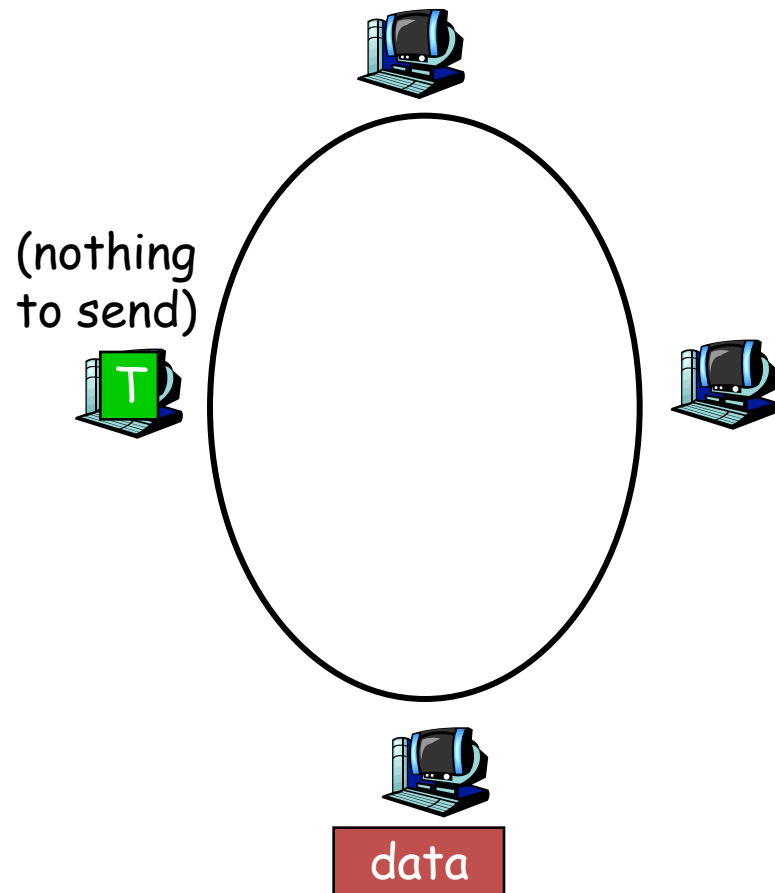
- polling overhead
- latency
- single point of failure (master)



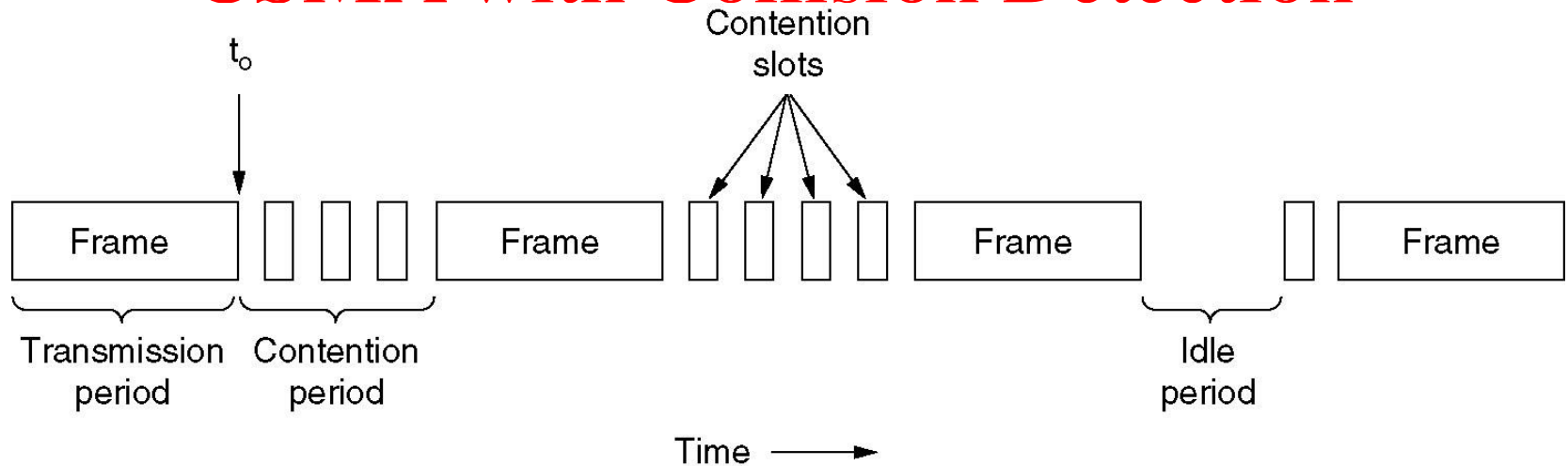
# Token passing protocol

- control **token** passed from one node to the next sequentially.
  - usually in a ring structure, might be “logical”
  - concerns:
    - token overhead
    - latency
    - single point of failure
- 2010 (token)

 :Token



# CSMA with Collision Detection



MA: multiple access

CS: carrier sense, a sender does not start when another is sending

CD: if sender detects collision it aborts sending, waits a random time to try sending again (if no one else is sending)

# IEEE 802.3 Ethernet (10Mb)

Name	Cable	Max. seg.	Nodes/seg.	Advantages
10Base5	Thick coax	500 m	100	Original cable; now obsolete
10Base2	Thin coax	185 m	30	No hub needed
10Base-T	Twisted pair	100 m	1024	Cheapest system
10Base-F	Fiber optics	2000 m	1024	Best between buildings

Repeater: receive, amplify and retransmit signals in both directions.

used for thick and thin coax cables to connect segments

Hub: logically connects UTP cables into 1 long ethernet cable

may contain electronics to detect and disconnect faulty

UTP

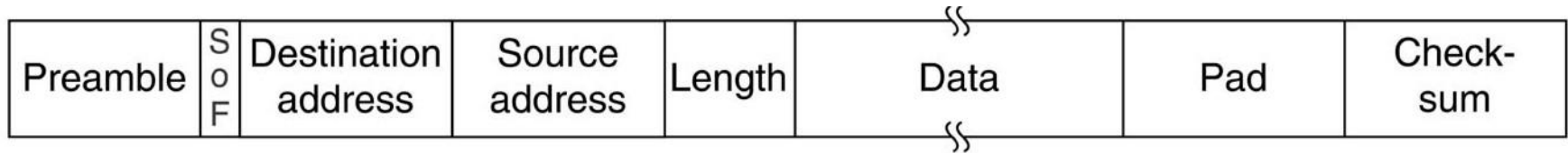
also reshapes the signals

UTP: Cat 3 unshielded twisted pair, 4 pairs per cable

already in use for telephone

only 2 pairs are used for 10Mb (and for telephone)

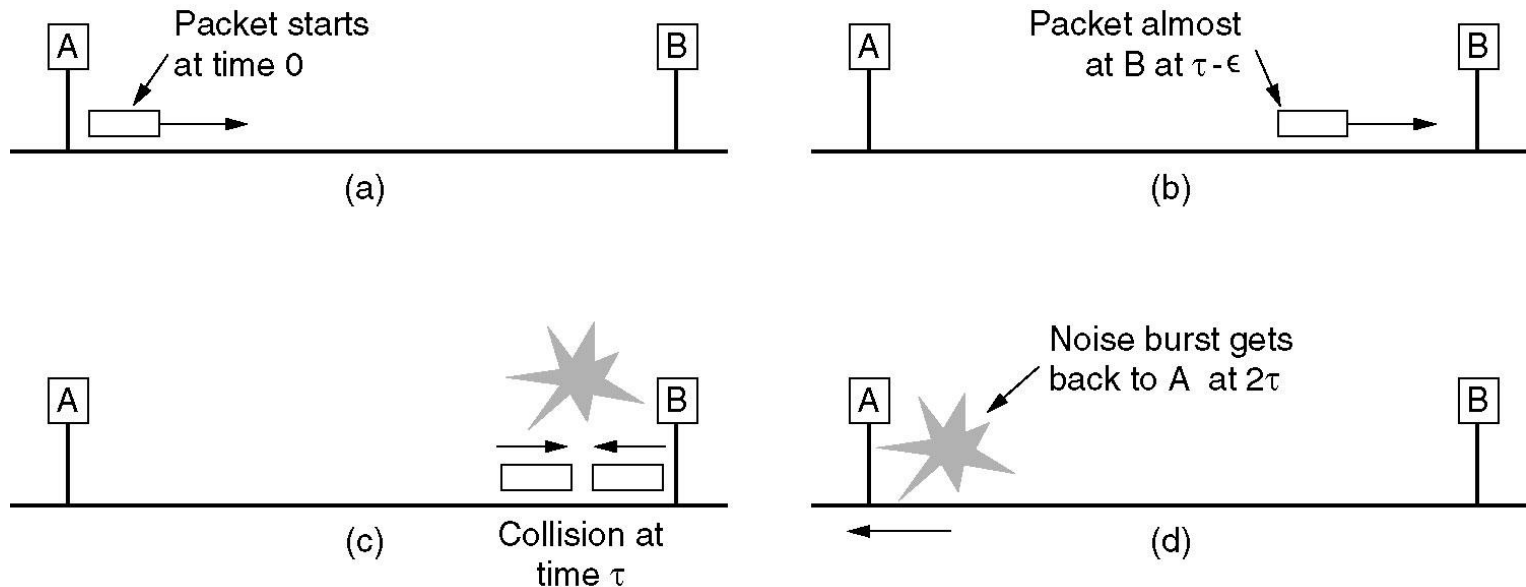
# Ethernet frame format



- Preamble of 7 bytes used to synchronize clocks generating 10 MHz square wave for 5.6  $\mu$ sec (synchronization)
- The Start Of Frame delimiter contains 10101011
- Addresses are 6 bytes.
- The addresses are unique in the world.
- The address containing all 1's is reserved for **broadcast**, a message destined to all receivers
- There were many variations in using the 2 byte length
- Most common now is to use it as a Type field, indicating that the data is a higher level protocol packet, e.g. 0x0800 for IPv4 and 0x86DD for IPv6



# Ethernet Collision Detection



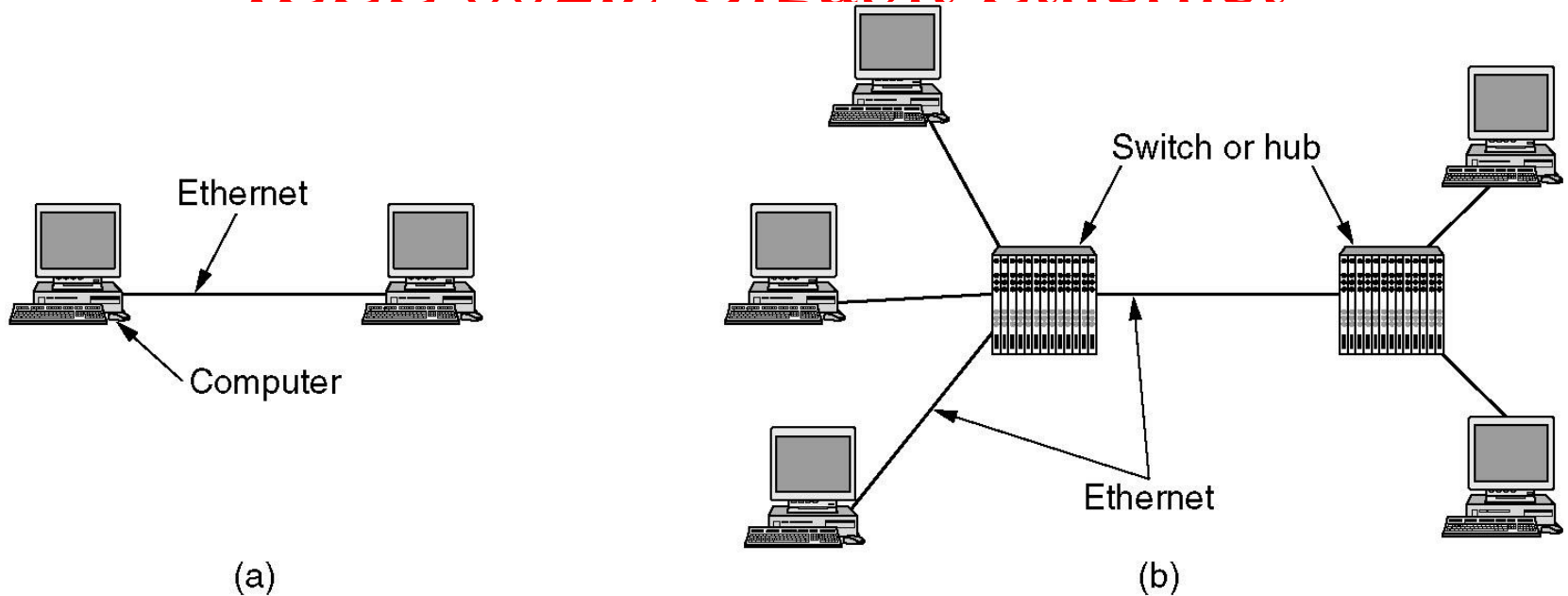
Collision detection takes maximal the roundtrip time  
When “B” detects a collision its stops sending and emits a 48 bit noise burst  
“A” must still be sending to detect the collision  
Frame must be minimal 64 bytes to allow a maximal (original) cable length of 2500 m including 4 repeaters

# IEEE 802.u 100Mb Fast Ethernet

- The basic idea was simple: keep all the old packet formats, interface and procedural rules, just reduce the bit time from 100 ns to 10 ns, to reach 100 Mbps.
- Coax cables are not used any more; in our old building in 2000 replaced by Cat5 UTP
- Cat 5 UTP: 2 pairs; 125 MHz, 4 bits encoded in 5 signals
- Auto-negotiation: mix 10 and 100 Mb

100Base-T4	Cat 3 UTP 4 pairs	8BT6	100 m	cheap cat 3
100Base-TX	Cat 5 UTP 2 pairs	4B5B	100 m	full duplex at 100 Mbps
100Base-F	Fiber multi-mode		2000 m	full duplex, long runs

# IEEE 802.7 Gigabit Ethernet



**(a)** A two-station Ethernet. **(b)** A multi-station Ethernet.

Hub's could be used, but then the total maximum length becomes only 25 m, because collisions must be detected. A designed extension of the

# Gigabit Ethernet Cat 5 UTP

Name	Cable	Max. segment	Advantages
1000Base-SX	Fiber optics	550 m	Multimode fiber (50, 62.5 microns)
1000Base-LX	Fiber optics	5000 m	Single (10 $\mu$ ) or multimode (50, 62.5 $\mu$ )
1000Base-CX	2 Pairs of STP	25 m	Shielded twisted pair
1000Base-T	4 Pairs of UTP	100 m	Standard category 5 UTP

All 4 pairs of Cat 5 cable are used to send 4 symbols, each representing 2 bits, in parallel.

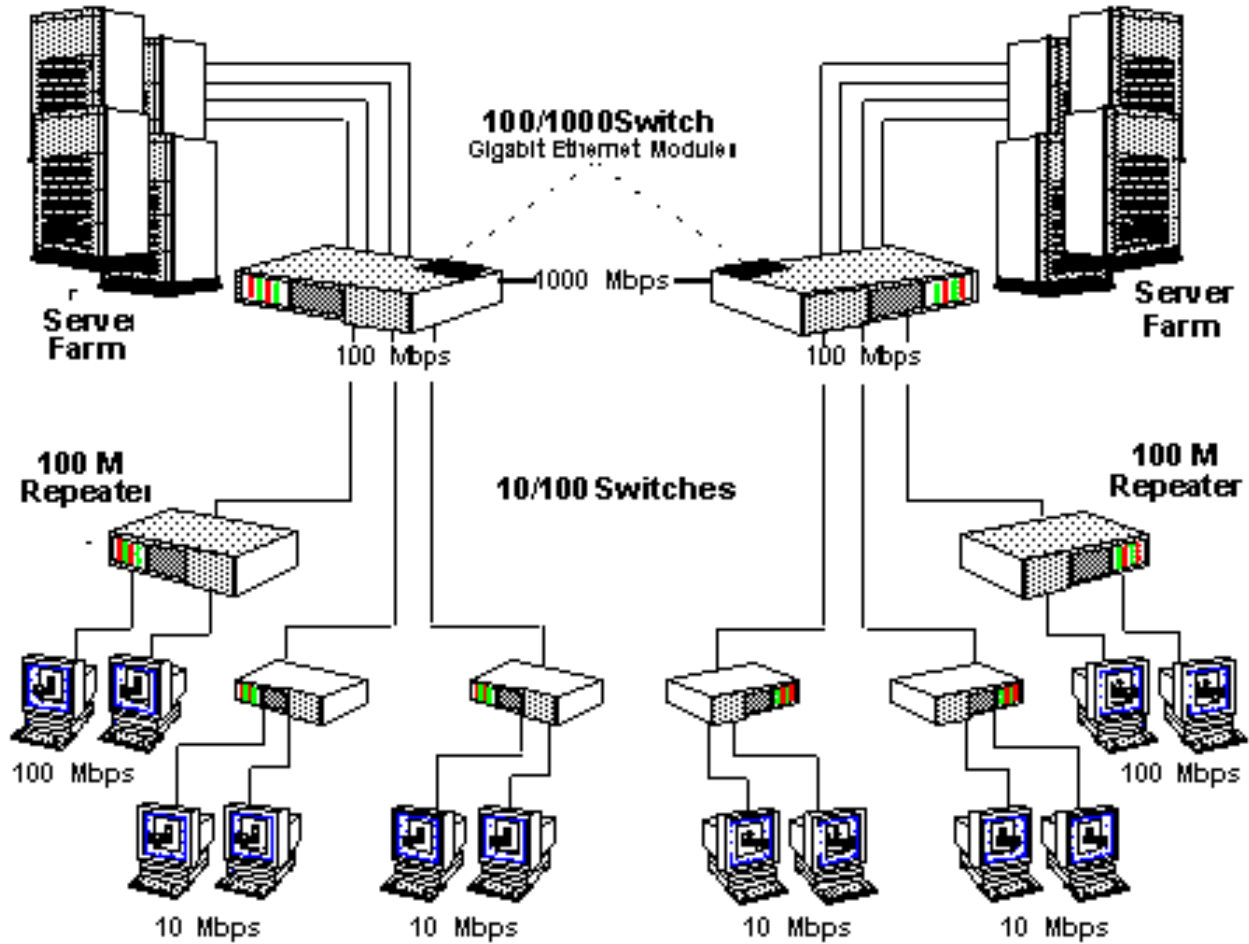
Thus 8 logical bits are send in parallel at 125MHz: 1Gb/s.

Each pair uses 5 voltage levels, this gives  $5^4=625$

possibilities. They are chosen not in a fixed way but using an elaborate Trellis coding and Viterbi decoding which allow for error detection and correction by the receiver.

Noise immunity is actually greater than for 100 Base-T.

# Easy upgrading



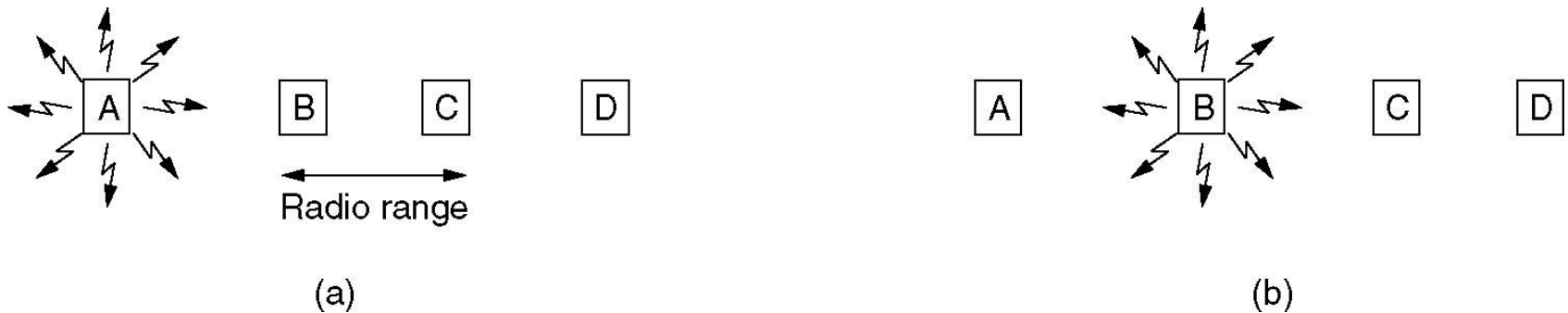
## Further developments

### IEEE 802.ae 10 GB Ethernet:

- Various Fiber possibilities to achieve distances up to 80 km.
- Cat 6 UTP ( 55 m), Cat 6a UTP (100m) or Cat 7 STP (100m)
  - 4 wires, 3 bits/wire , 833 MHz, 10 voltage levels
  - type RJ45 connector, easy upgrading
- many variants, still in development
- 40 Gb Dense wavelength-division multiplexing
  - 4 light carriers into one single-mode optical fiber

**40Gb and 100Gb Ethernet standards now in development**

# Wireless LAN Protocols



Not all stations are in reach of each other:

(a) hidden station problem: A sends to B, C sends to B

(b) exposed station problem: B sends to A, C might send to D

CA, Collision Avoidance protocols:

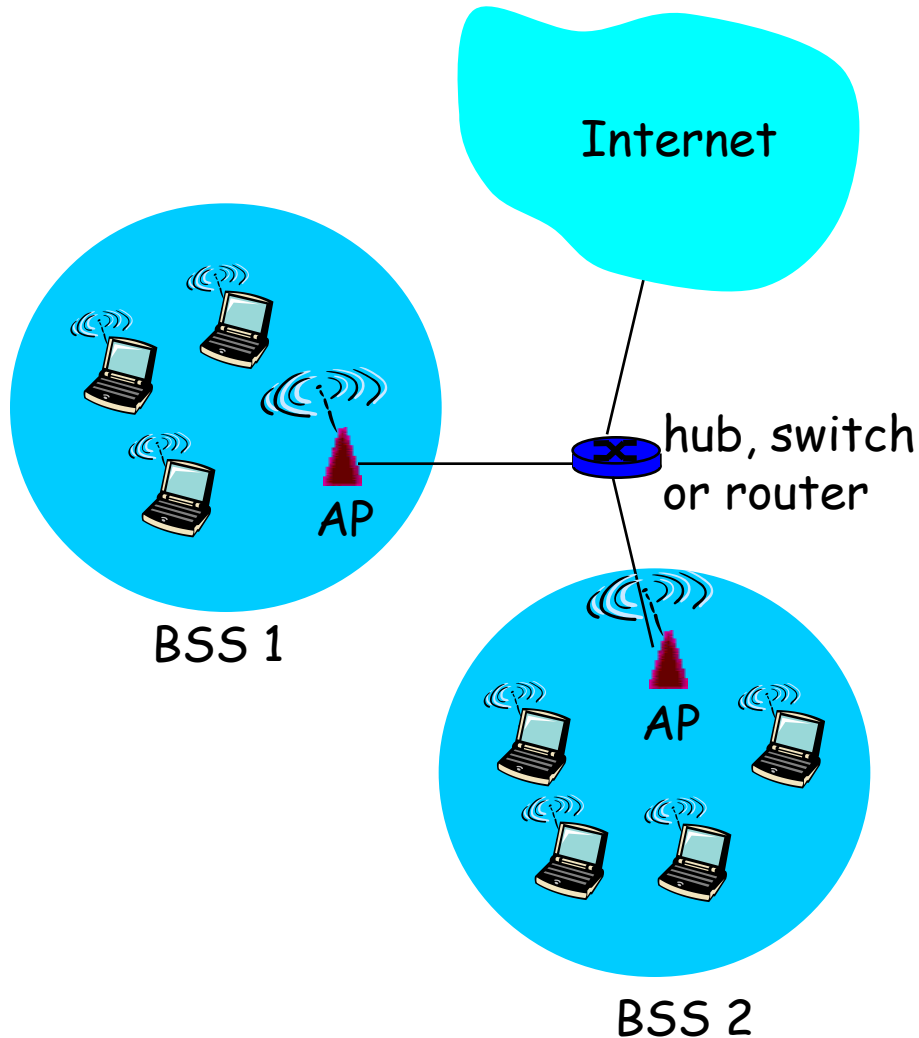
- <sup>2010</sup> RTS (request or ready to send) and CTS (clear <sup>303</sup>

# IEEE 802.11 WiFi

- 802.11b 2.4 GHz up to 11 Mb/s
  - 802.11a 5 GHz up to 54 Mb/s
  - 802.11g 2.4 GHz up to 54 Mb/s
  - 802.11n 5 GHz, more than 1 antenna, up to 200 Mb/s
- 
- They can operate together in one area.
  - Point (Access Point) or distributed control (ad-hoc network) can coexist in one cell
  - 4 types of frames: data, PC (point control), DC (distributed control) and management frames
- different waiting times after an ACK to avoid collisions
- also headers used by physical layer, e.g. for modulation methods to adapt the rate (integration of layers)



# Point or Distributed Control



## Point Control:

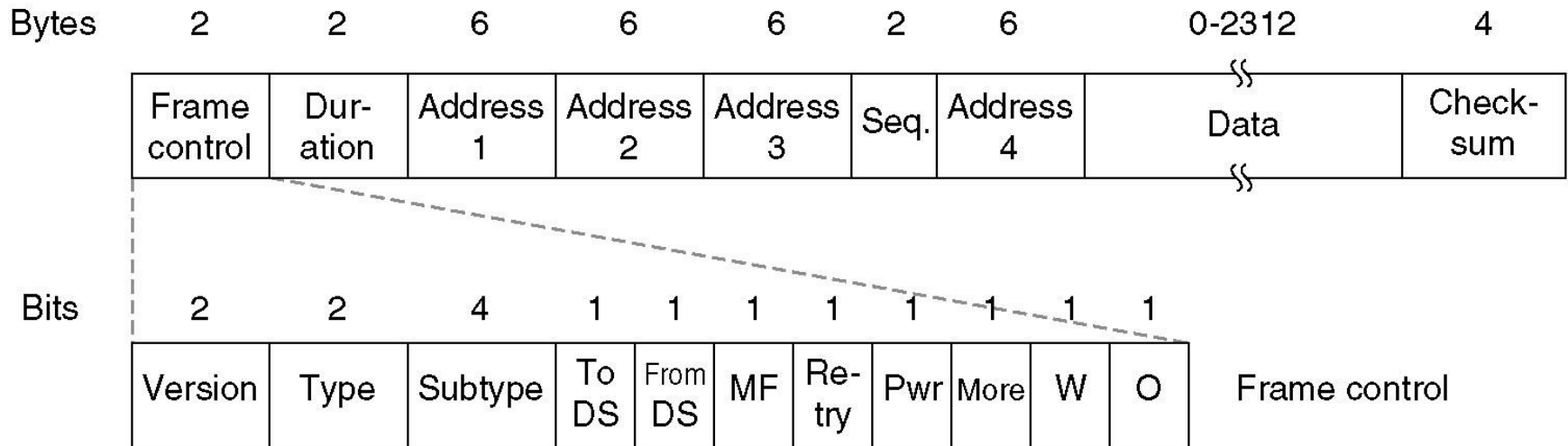
AP: access point (base station)

SSID: Service Set Identifier

11 overlapping frequency bands:  
1,6 and 11 are non-overlapping

## Distributed Control:

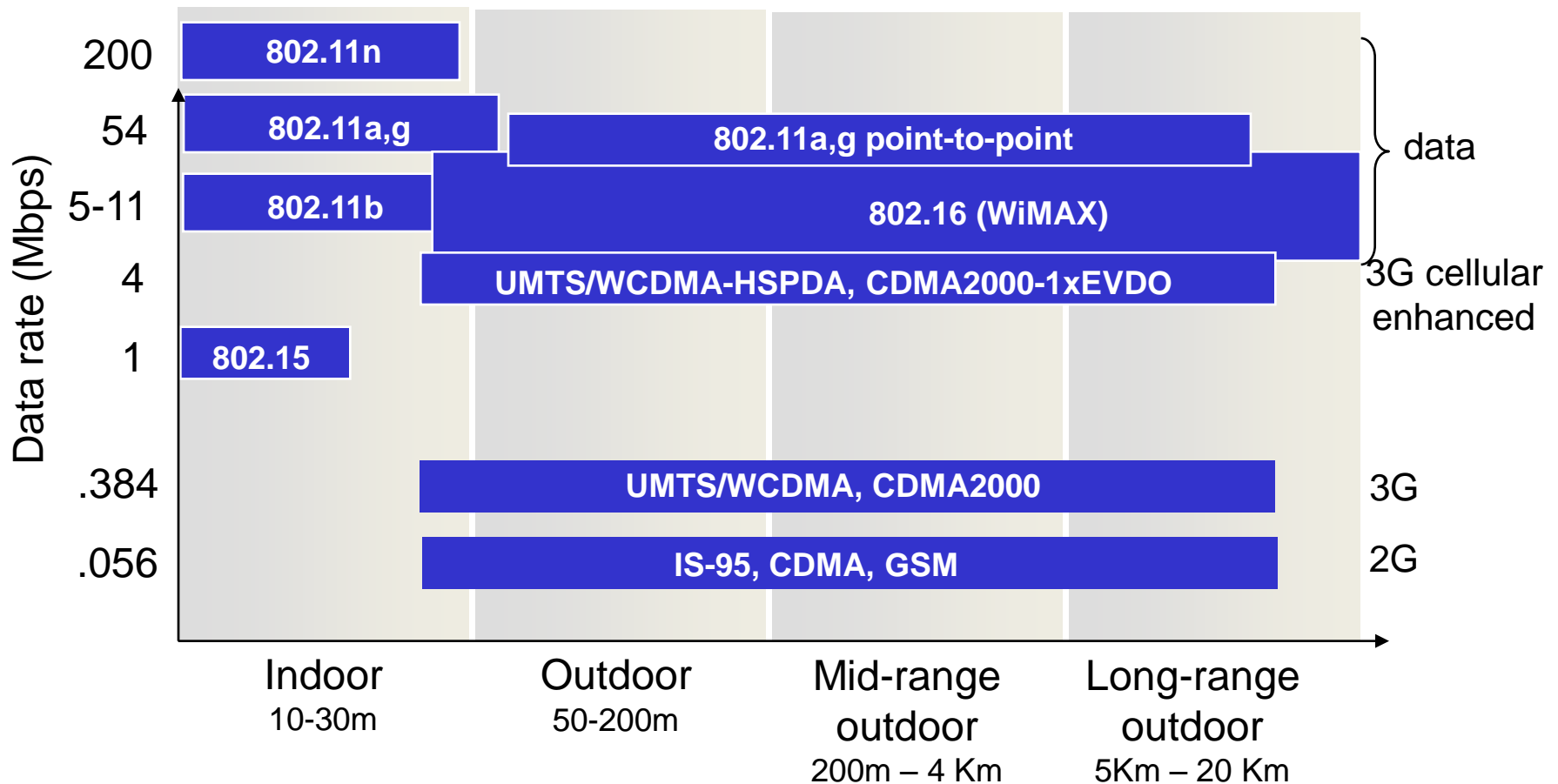
# The 802.11 Frame Structure



The data frame header contains 4 addresses, each in the standard 802 format. Two are used to identify the sending and receiving stations. The other two are used for the source and destination of base stations for intercell traffic.

The W bit indicates **WEP** (Wired Equivalent

# Wireless comparison

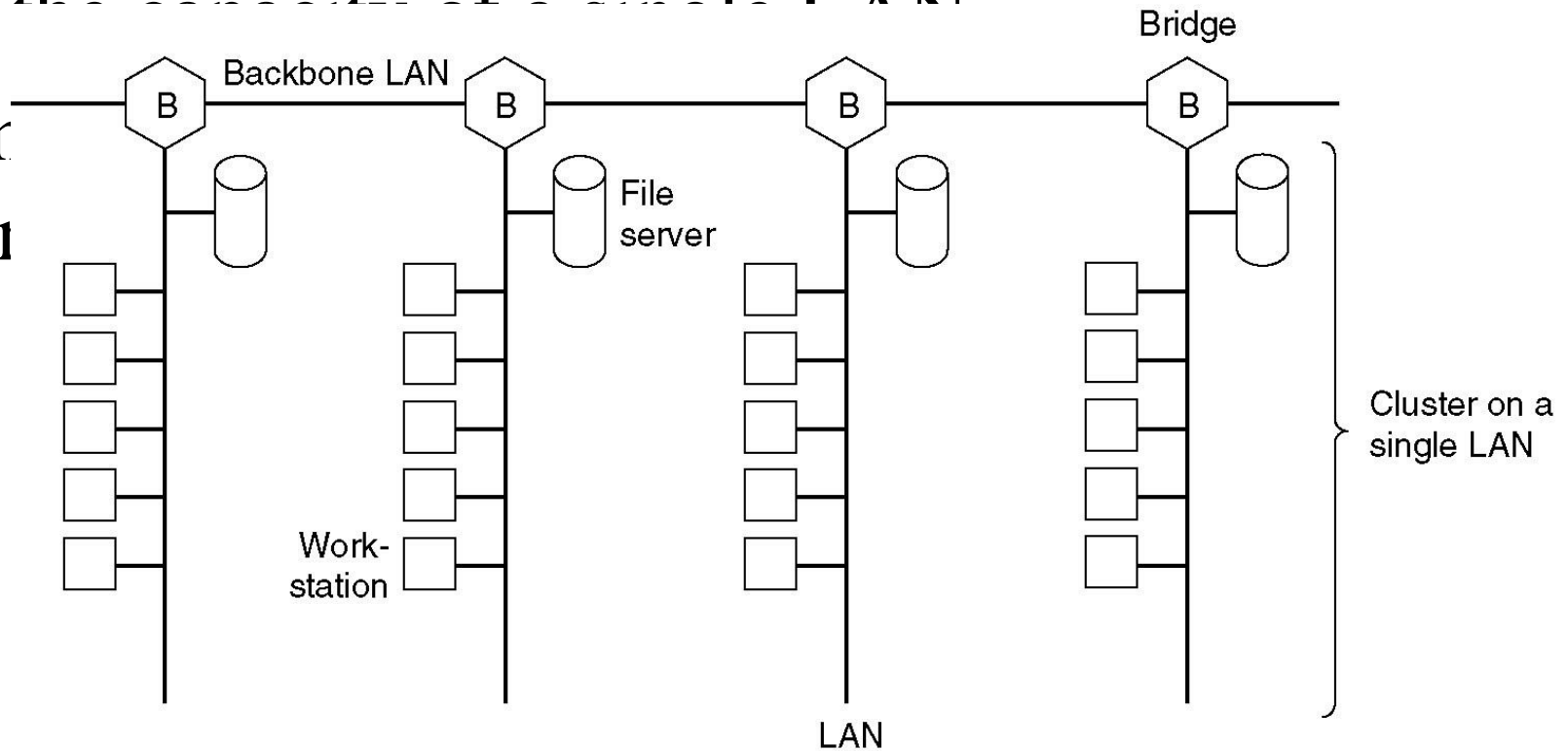


# Data Link Layer Switching

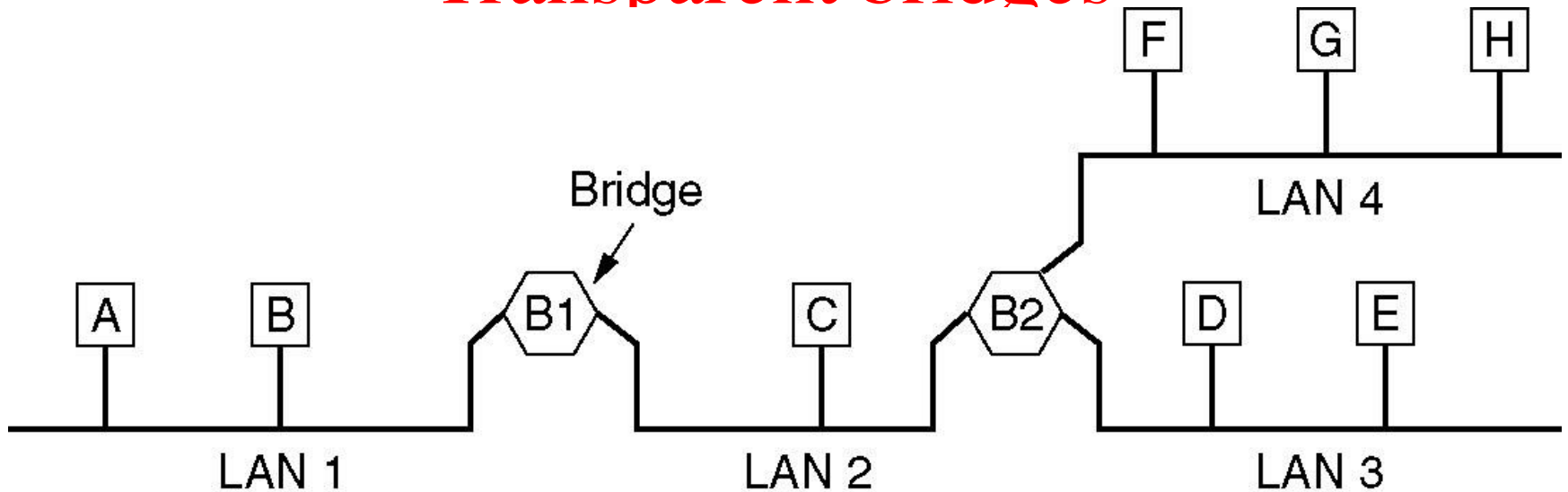
Multiple LANs connected by a backbone to handle a total load higher

than the capacity of any single LAN

Contrast



## Transparent bridges

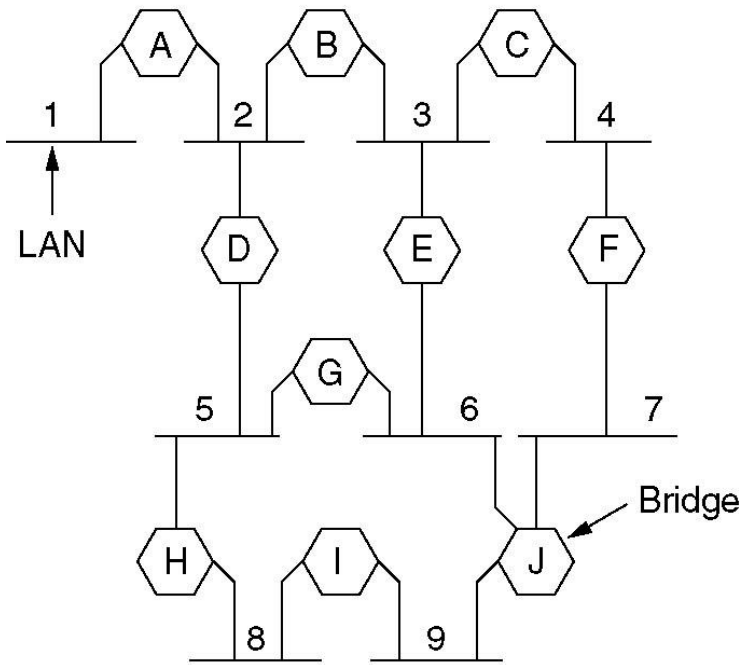


A **transparent** bridge only requires connecting it to the LAN's. No software changes or downloading routing tables or parameters.

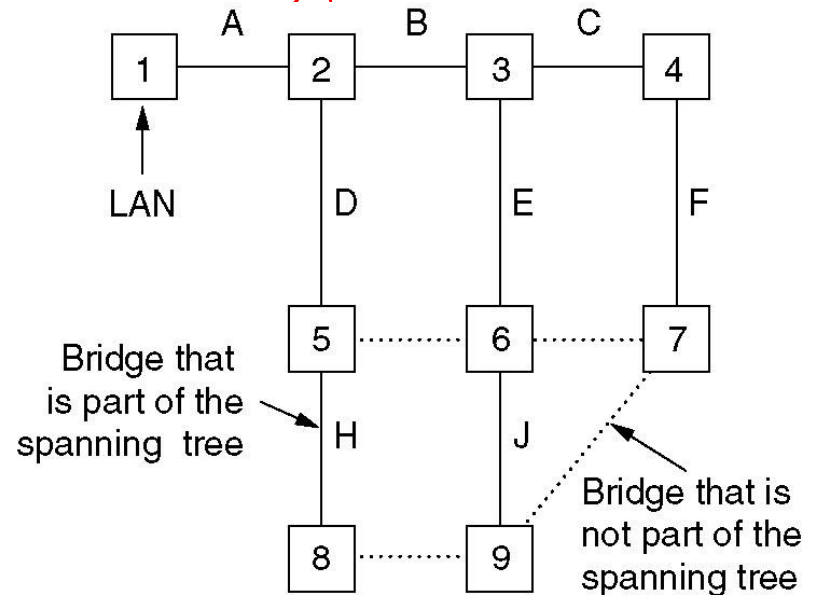
It learns which computers are on each LAN by looking at the sender address on each packet.

If it does not know the receiver address, it sends out

# Spanning Tree Bridges



(a)



(b)

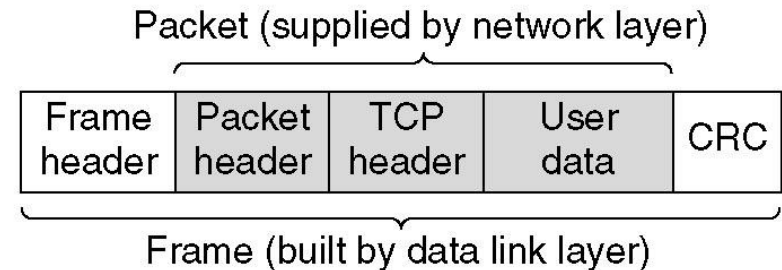
More than 1 route possible between LAN's

Bridges communicate with each other to construct a spanning tree

This is regularly repeated to detect and correct

# Repeaters, Hubs, Bridges, Switches, Routers and Gateways

Application layer	Application gateway
Transport layer	Transport gateway
Network layer	Router
Data link layer	Bridge, switch
Physical layer	Repeater, hub



A bridge connects two or more LAN's

A switch is more often used to connect individual computers.

A router gets the packet out of a frame and uses the information in the packet header, for example the IP addresses.

A transport gateway receives e.g. a TCP packet and uses the header information to decide what to do with the packet.

An application gateway understands the format and content of the data. It can translate messages from one format to another. Might also be used for security, e.g. blocking messages

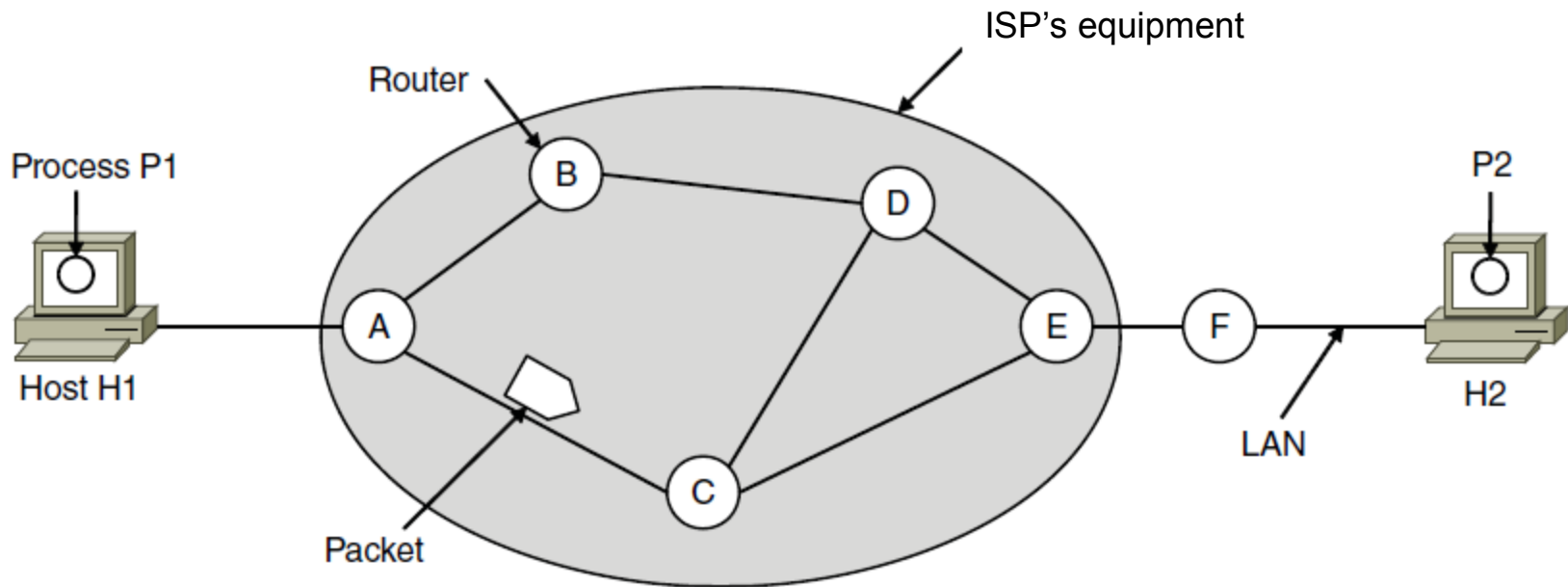
# The Network Layer



# Network Layer Design Issues

- Store-and-forward packet switching
- Services provided to transport layer
- Implementation of connectionless service
- Implementation of connection-oriented service
- Comparison of virtual-circuit and datagram networks

# Store-and-Forward Packet Switching

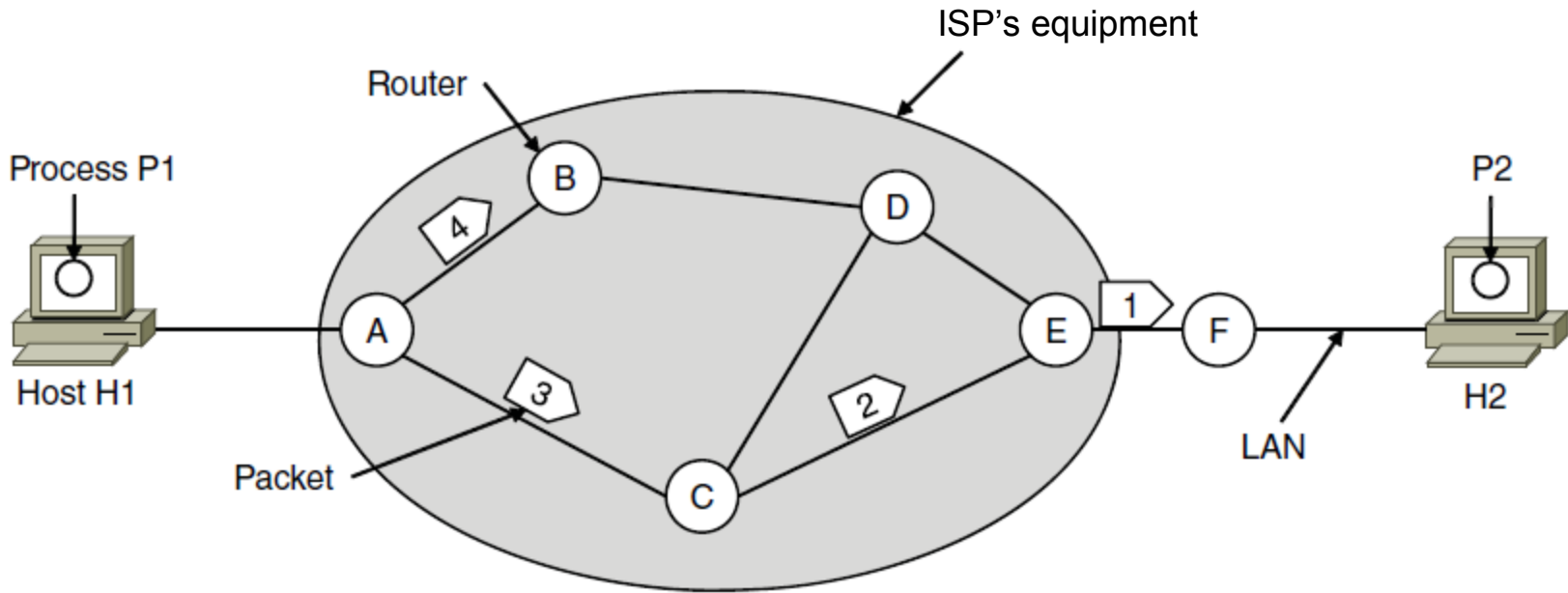


The environment of the network layer protocols.

# Services Provided to the Transport Layer

1. Services independent of router technology.
2. Transport layer shielded from number, type, topology of routers.
3. Network addresses available to transport layer use uniform numbering plan
  - even across LANs and WANs

# Implementation of Connectionless Service



A's table (initially)

A	
B	B
C	C
D	B
E	C
F	C

Dest. Line

A's table (later)

A	
B	B
C	C
D	B
E	D
F	D

C's Table

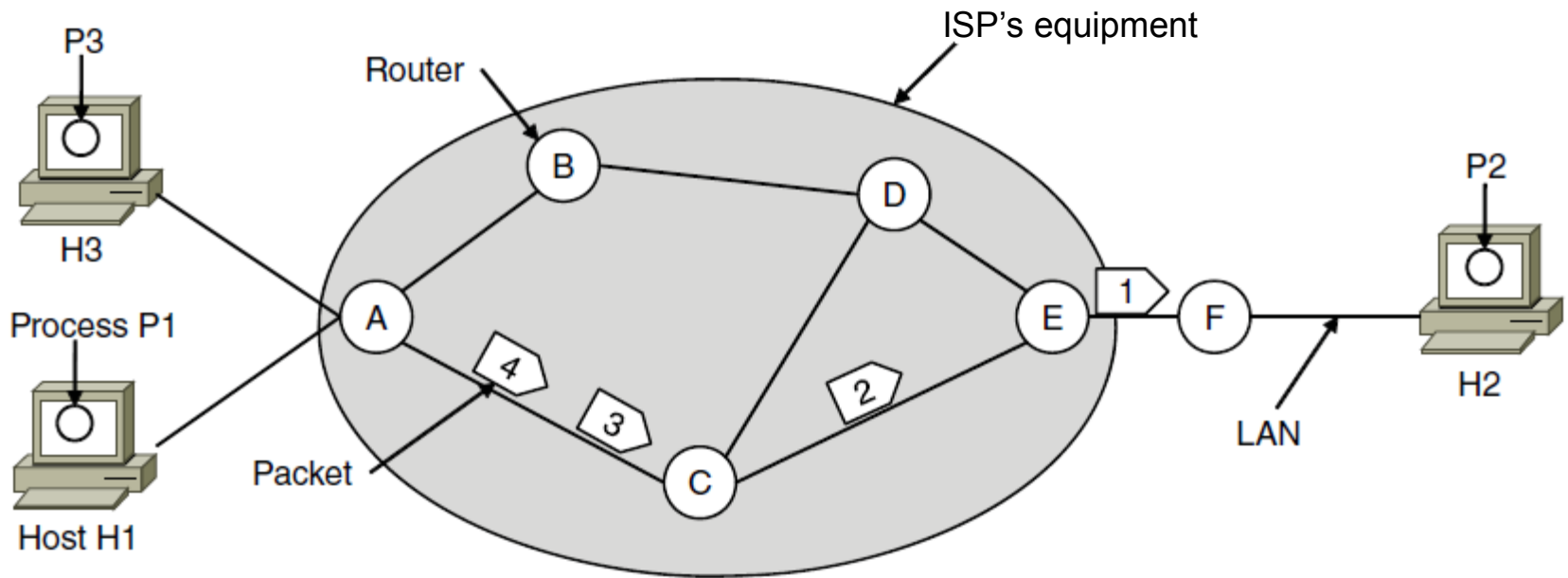
A	A
B	A
C	
D	E
E	E
F	E

E's Table

A	C
B	D
C	C
D	D
E	
F	F

Routing within a datagram network

# Implementation of Connection-Oriented Service



A's table

H1	1	C	1
H3	1	C	2
In		Out	

C's Table

A	1	E	1
A	2	E	2

E's Table

C	1	F	1
C	2	F	2

Routing within a virtual-circuit network

# Comparison of Virtual-Circuit and Datagram Networks

Issue	Datagram network	Virtual-circuit network
Circuit setup	Not needed	Required
Addressing	Each packet contains the full source and destination address	Each packet contains a short VC number
State information	Routers do not hold state information about connections	Each VC requires router table space per connection
Routing	Each packet is routed independently	Route chosen when VC is set up; all packets follow it
Effect of router failures	None, except for packets lost during the crash	All VCs that passed through the failed router are terminated
Quality of service	Difficult	Easy if enough resources can be allocated in advance for each VC
Congestion control	Difficult	Easy if enough resources can be allocated in advance for each VC

Comparison of datagram and virtual-circuit networks

# Routing Algorithms (1)

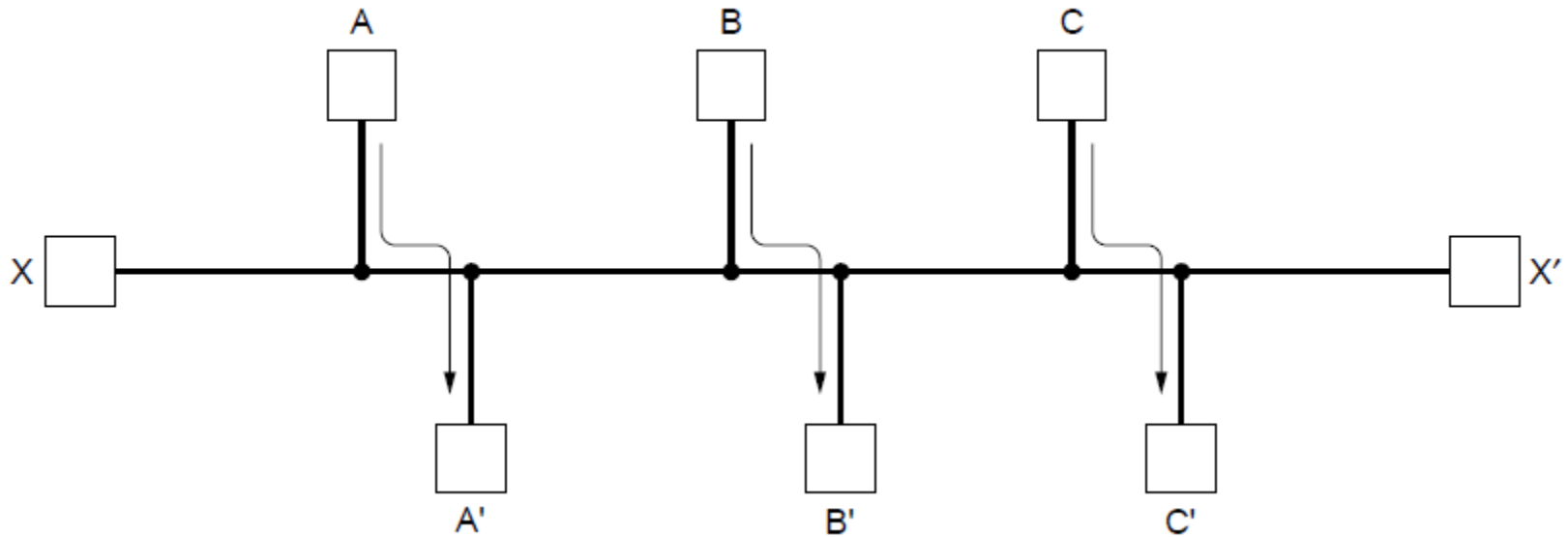
- Optimality principle
- Shortest path algorithm
- Flooding
- Distance vector routing
- Link state routing
- Routing in ad hoc networks

## Routing Algorithms (2)

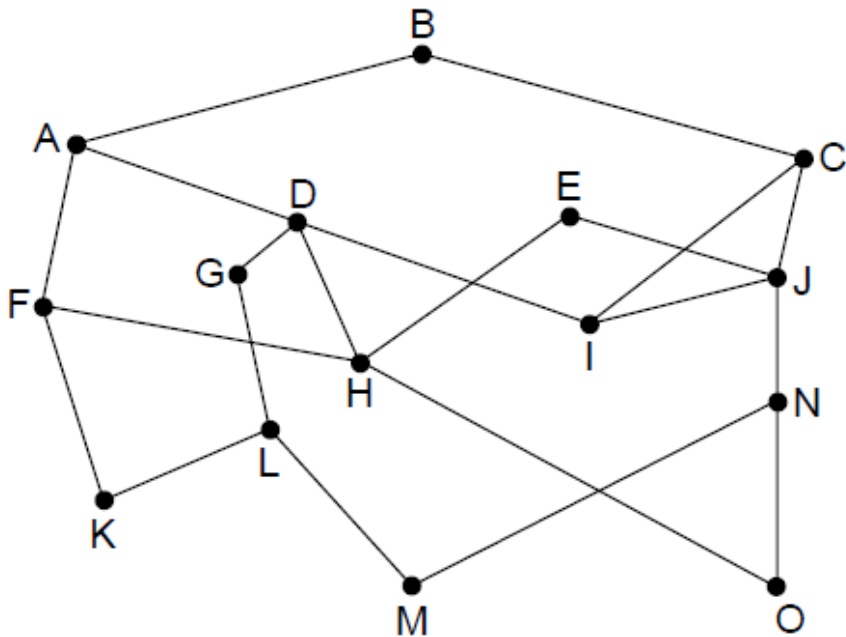
- Broadcast routing
- Multicast routing
- Anycast routing
- Routing for mobile hosts
- Routing in ad hoc networks



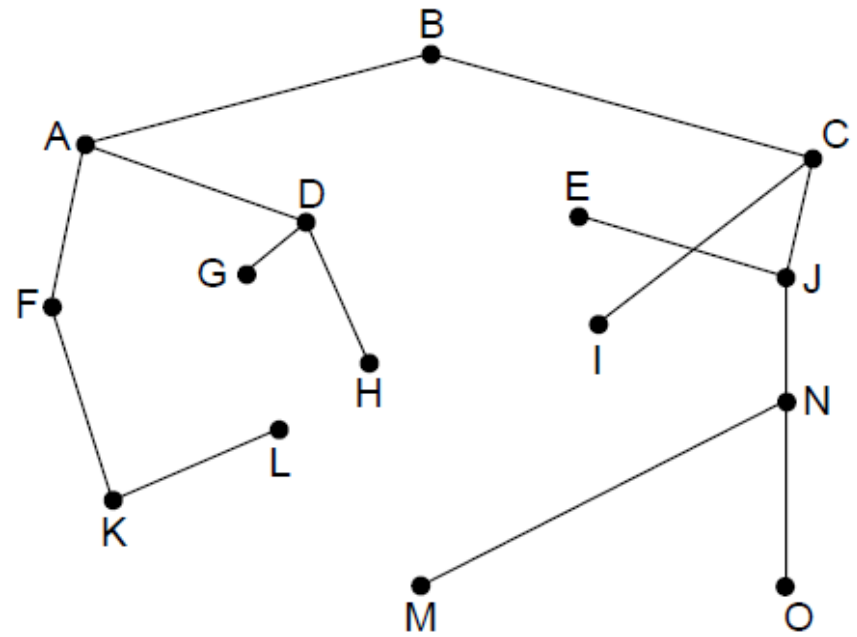
# Fairness vs. Efficiency



# The Optimality Principle



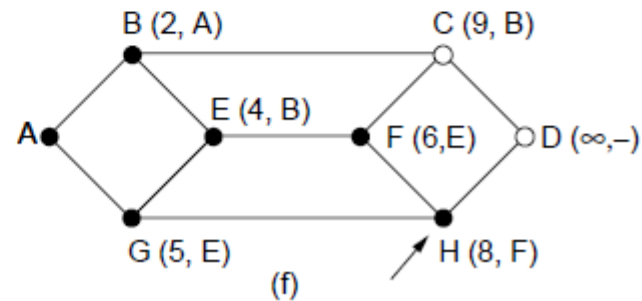
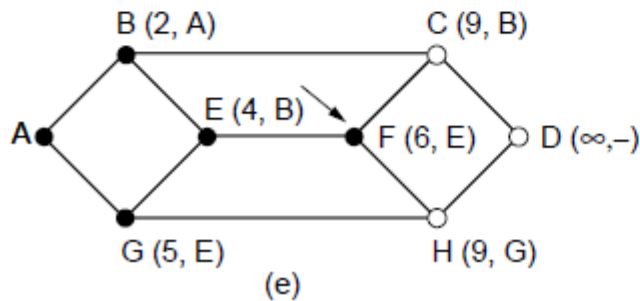
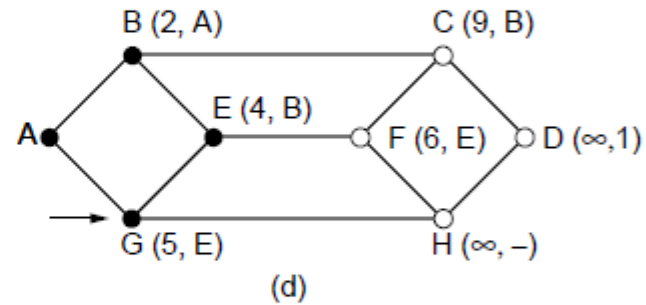
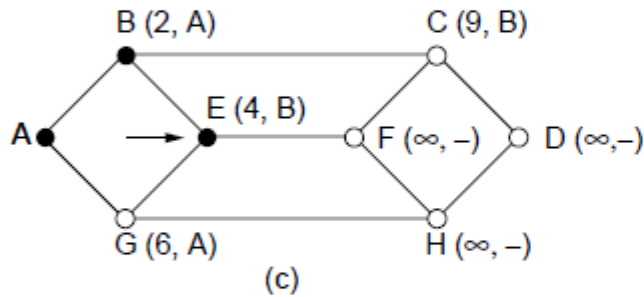
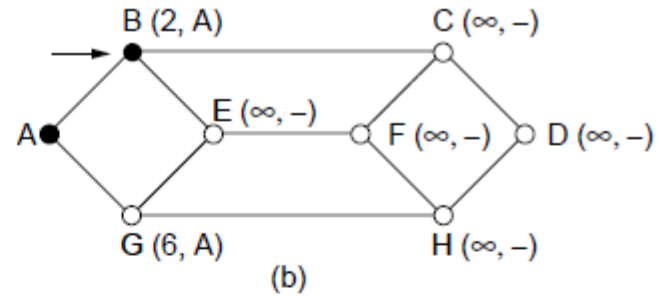
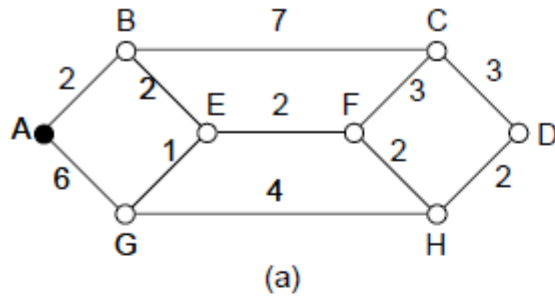
(a)



(b)

(a) A network. (b) A sink tree for router *B*.

# Shortest Path Algorithm (1)



The first five steps used in computing the shortest path from *A* to *D*. The arrows indicate

# Shortest Path Algorithm (2)

```
#define MAX_NODES 1024 /* maximum number of nodes */
#define INFINITY 1000000000 /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES]; /* dist[i][j] is the distance from i to j */

void shortest_path(int s, int t, int path[])
{ struct state { /* the path being worked on */
    int predecessor; /* previous node */
    int length; /* length from source to this node */
    enum {permanent, tentative} label; /* label state */
} state[MAX_NODES];

int i, k, min;
struct state *p;

. . .
```

Dijkstra's algorithm to compute the shortest path through a graph

# Shortest Path Algorithm (3)

...

```
for (p = &state[0]; p < &state[n]; p++) {      /* initialize state */
    p->predecessor = -1;
    p->length = INFINITY;
    p->label = tentative;
}
state[t].length = 0; state[t].label = permanent;
k = t;                                          /* k is the initial working node */
do {                                          /* Is there a better path from k? */
    for (i = 0; i < n; i++)                 /* this graph has n nodes */
        if (dist[k][i] != 0 && state[i].label == tentative) {
            if (state[k].length + dist[k][i] < state[i].length) {
                state[i].predecessor = k;
                state[i].length = state[k].length + dist[k][i];
            }
        }
}
}
```

...

Dijkstra's algorithm to compute the shortest path through a graph

# Shortest Path Algorithm (4)

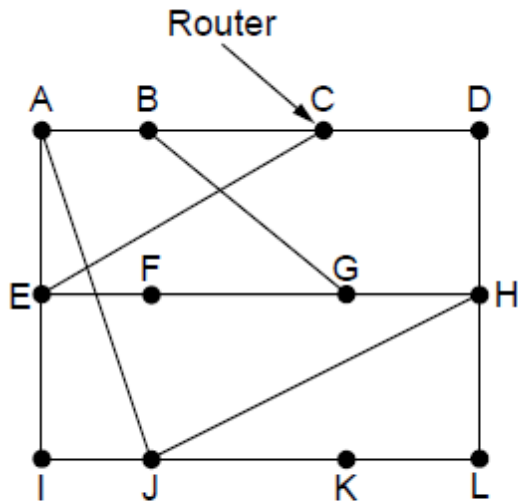
...

```
/* Find the tentatively labeled node with the smallest label. */
k = 0; min = INFINITY;
for (i = 0; i < n; i++)
    if (state[i].label == tentative && state[i].length < min) {
        min = state[i].length;
        k = i;
    }
state[k].label = permanent;
} while (k != s);

/* Copy the path into the output array. */
i = 0; k = s;
do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}
```

Dijkstra's algorithm to compute the shortest path through a graph

# Distance Vector Routing



(a)

To	A	I	H	K	New estimated delay from J	
					↓	Line
A	0	24	20	21	8	A
B	12	36	31	28	20	A
C	25	18	19	36	28	I
D	40	27	8	24	20	H
E	14	7	30	22	17	I
F	23	20	19	40	30	I
G	18	31	6	31	18	H
H	17	20	0	19	12	H
I	21	0	14	22	10	I
J	9	11	7	10	0	-
K	24	22	22	0	6	K
L	29	33	9	9	15	K

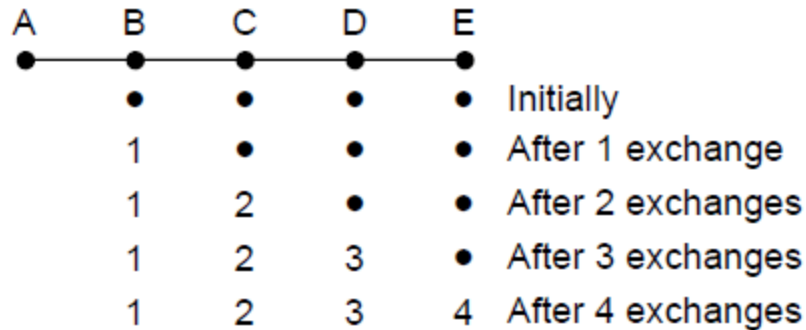
JA delay is 8	JI delay is 10	JH delay is 12	JK delay is 6	} New routing table for J
} Vectors received from J's four neighbors				

(b)

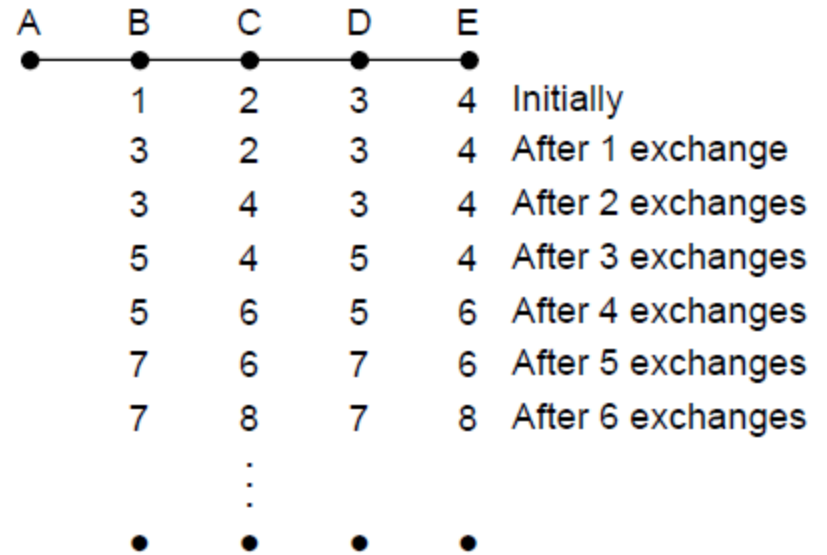
(a) A network.

(b) Input from A, I, H, K, and the new routing

# The Count-to-Infinity Problem



(a)



(b)

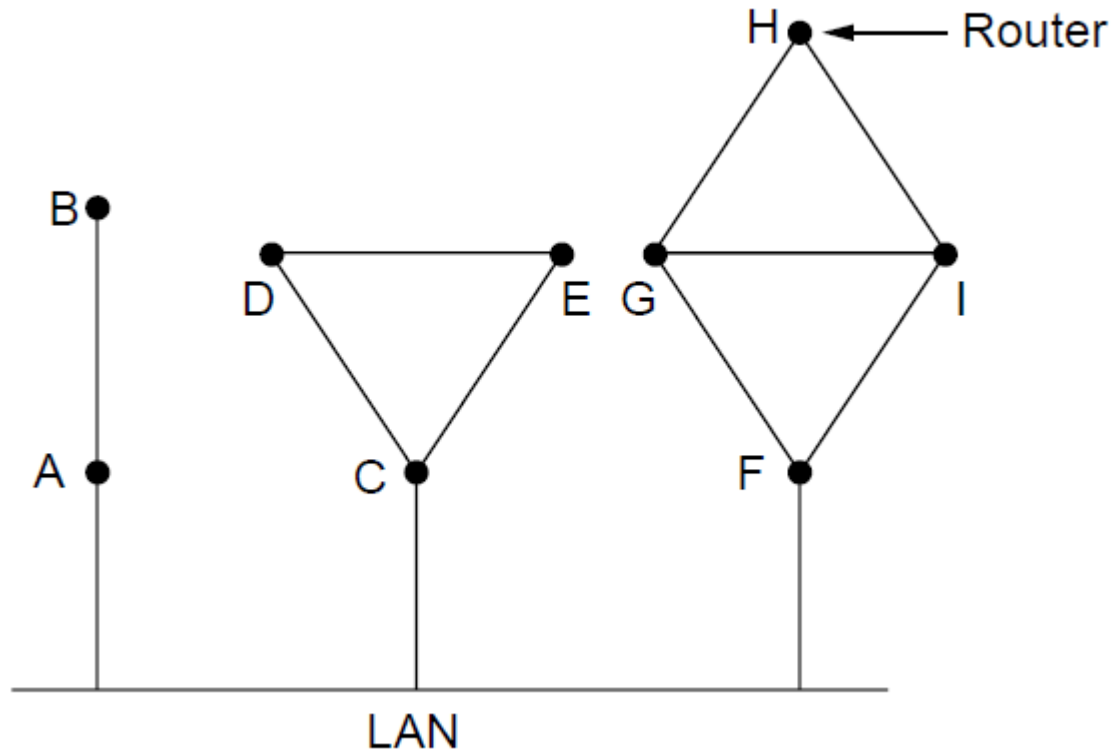
The count-to-infinity problem



# Link State Routing

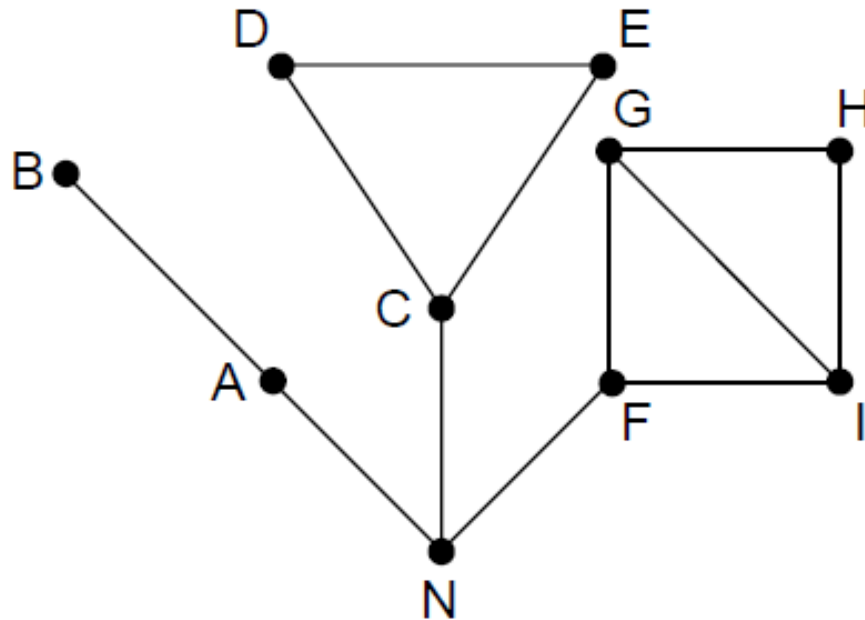
1. Discover neighbors, learn network addresses.
2. Set distance/cost metric to each neighbor.
3. Construct packet telling all learned.
4. Send packet to, receive packets from other routers.
5. Compute shortest path to every other router.

# Learning about the Neighbors (1)



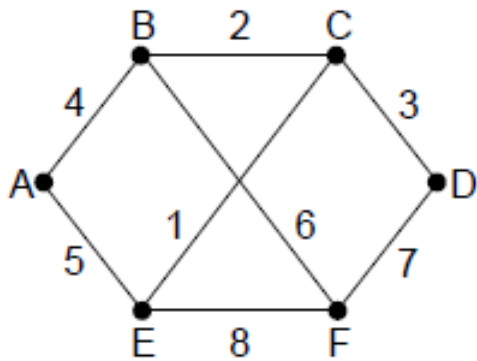
Nine routers and a broadcast LAN.

# Learning about the Neighbors (2)



A graph model of previous slide.

# Building Link State Packets



(a)

		Link		State		D		E		F	
A		B		C		D		E		F	
Seq.		Seq.		Seq.		Seq.		Seq.		Seq.	
Age		Age		Age		Age		Age		Age	
B	4	A	4	B	2	C	3	A	5	B	6
E	5	C	2	D	3	F	7	C	1	D	7
		F	6	E	1			F	8	E	8

(b)

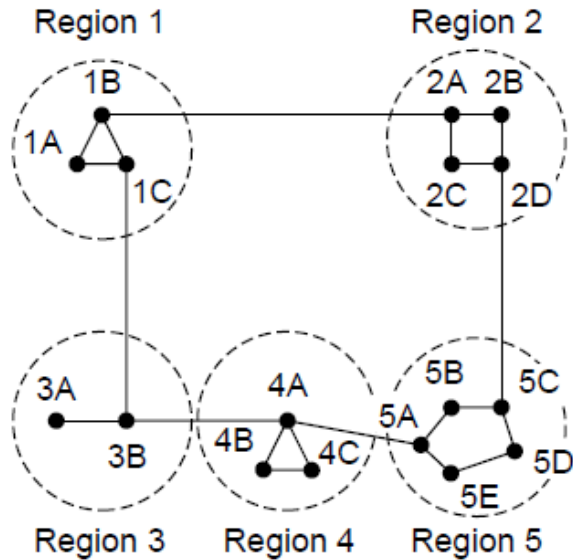
(a) A network. (b) The link state packets for this network.

# Distributing the Link State Packets

Source	Seq.	Age	Send flags			ACK flags			Data
			A	C	F	A	C	F	
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

The packet buffer for router *B* in previous slide

# Hierarchical Routing



(a)

Full table for 1A

Dest.	Line	Hops
1A	-	-
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

(b)

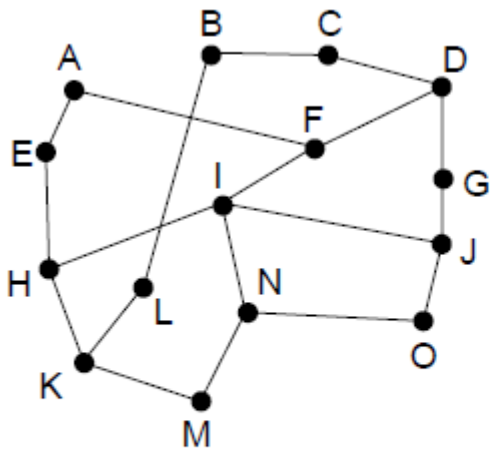
Hierarchical table for 1A

Dest.	Line	Hops
1A	-	-
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

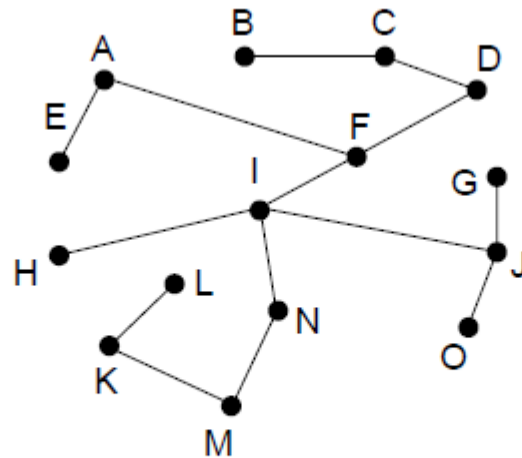
(c)

Hierarchical routing.

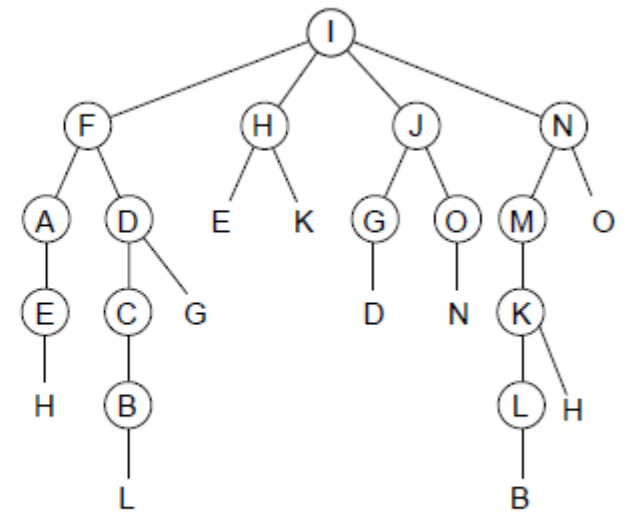
# Broadcast Routing



(a)



(b)

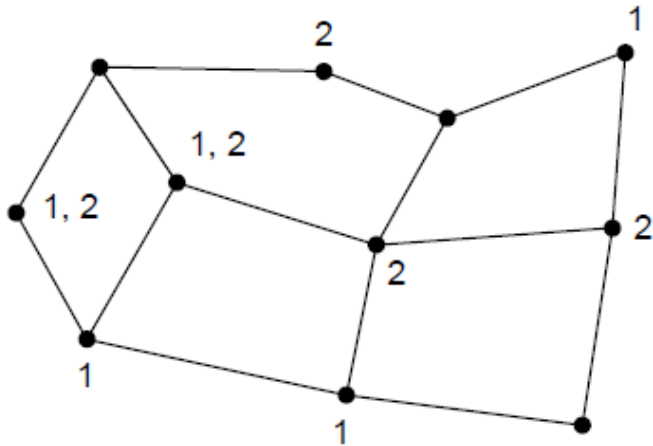


(c)

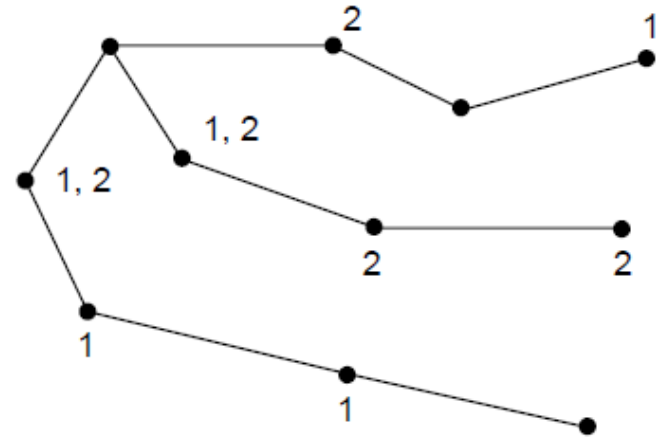
Reverse path forwarding. (a) A network. (b) A sink tree.

(c) The tree built by reverse path forwarding.

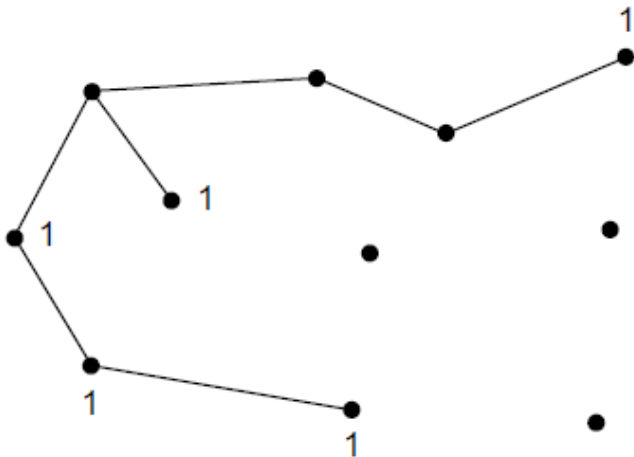
# Multicast Routing (1)



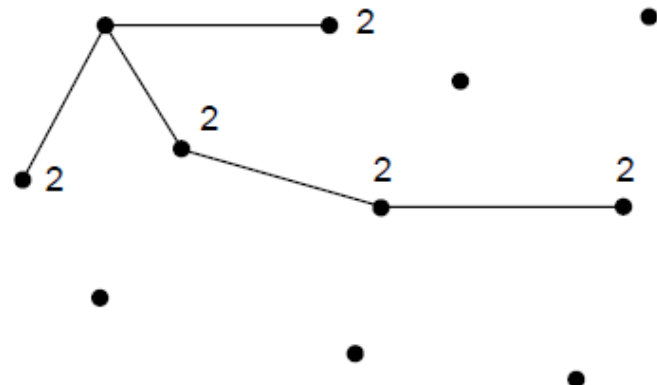
(a)



(b)



(c)

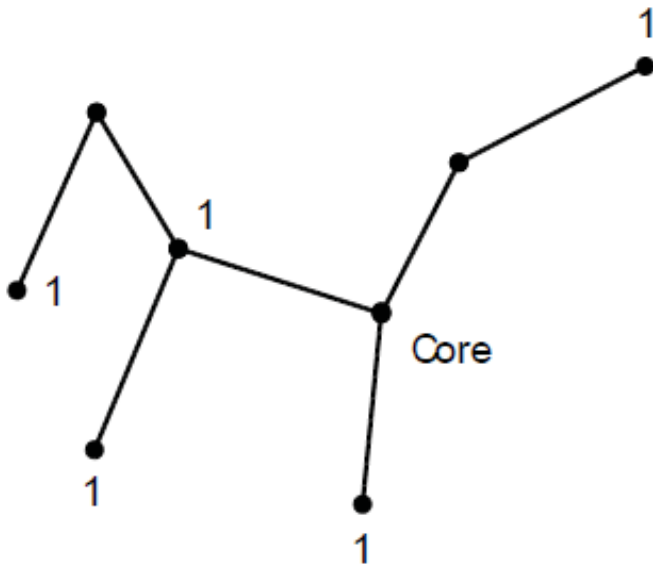


(d)

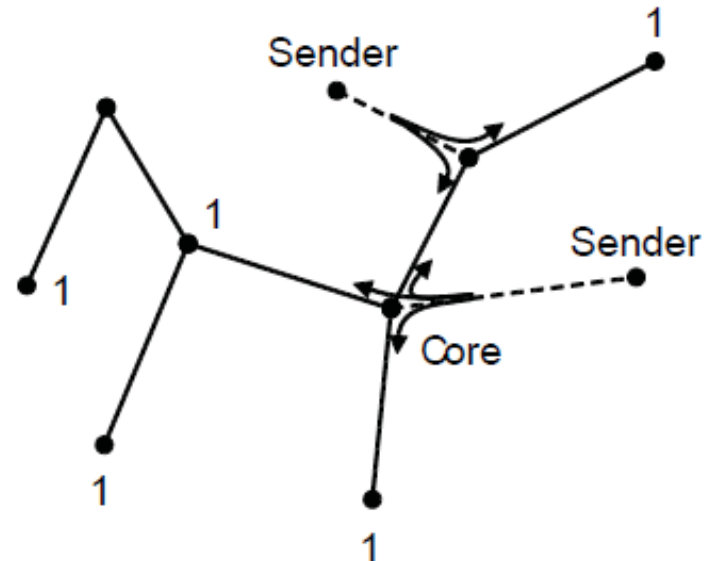
(a) A network. (b) A spanning tree for the leftmost router. (c) A



# Multicast Routing (2)



(a)



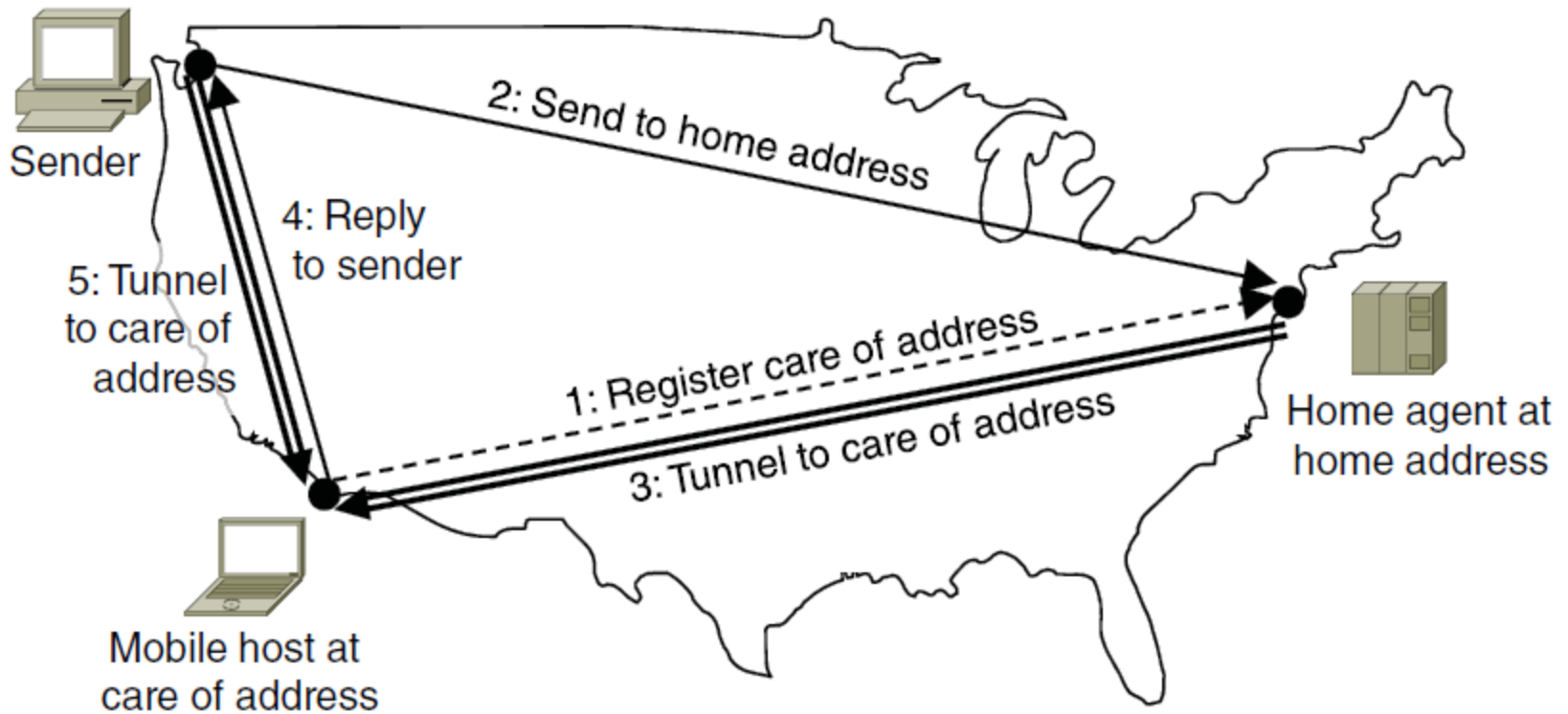
(b)

(a) Core-based tree for group 1.

(b) Sending to group 1.

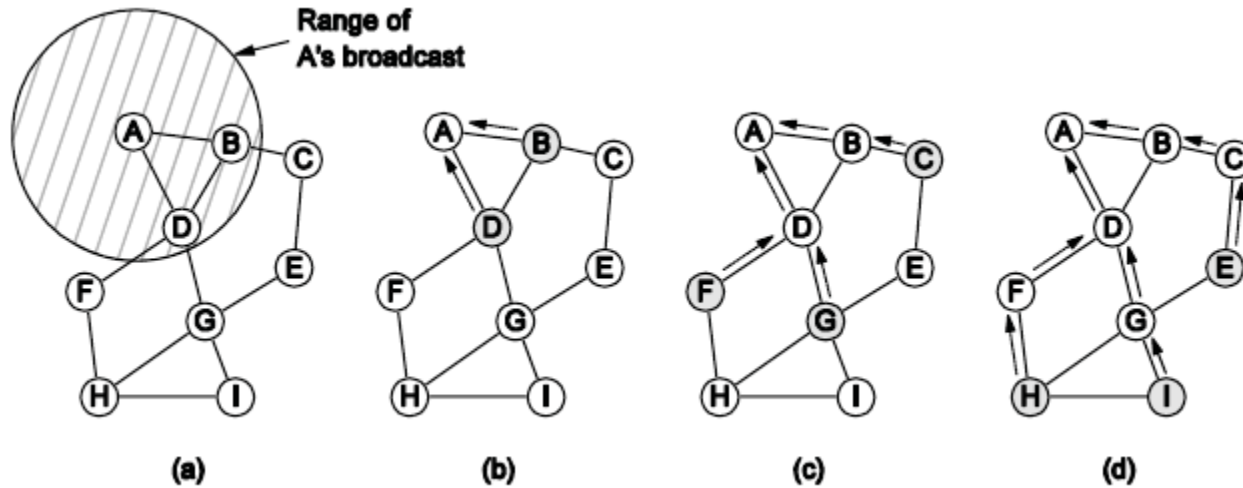


# Routing for Mobile Hosts



Packet routing for mobile hosts

# Routing in Ad Hoc Networks



(a) Range of A's broadcast.

(b) After B and D receive it.

(c) After C, F, and G receive it.

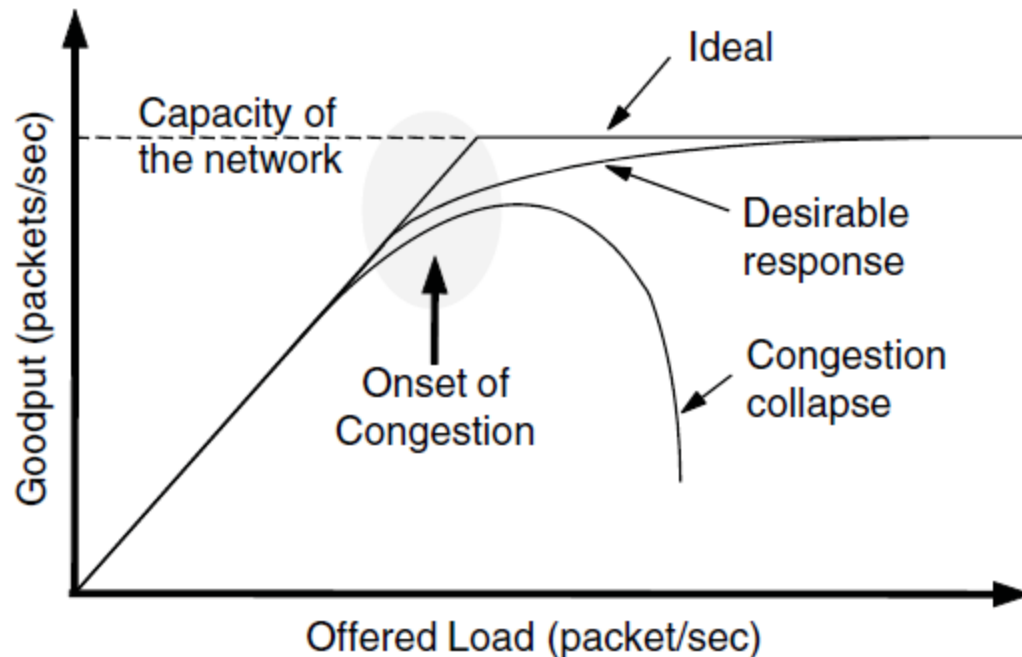
(d) After E, H, and I receive it.

The shaded nodes are new recipients. The dashed lines show possible reverse routes. The solid lines show the discovered route.

# Congestion Control Algorithms (1)

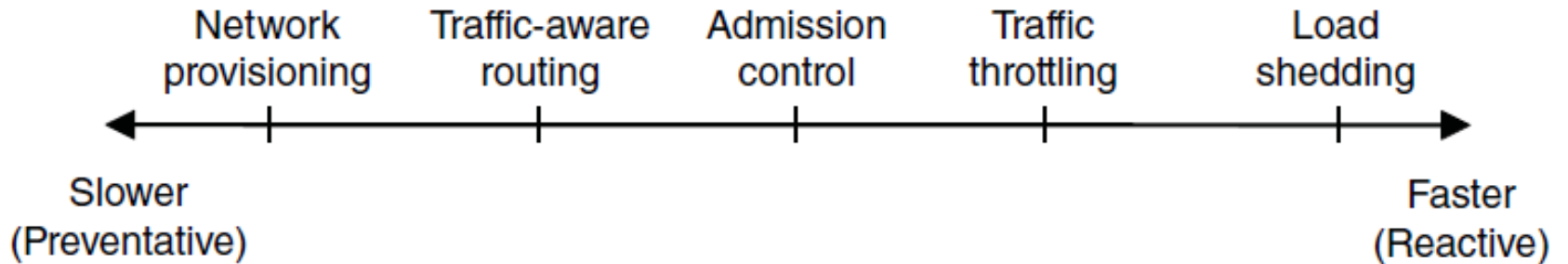
- Approaches to congestion control
- Traffic-aware routing
- Admission control
- Traffic throttling
- Load shedding

# Congestion Control Algorithms (2)



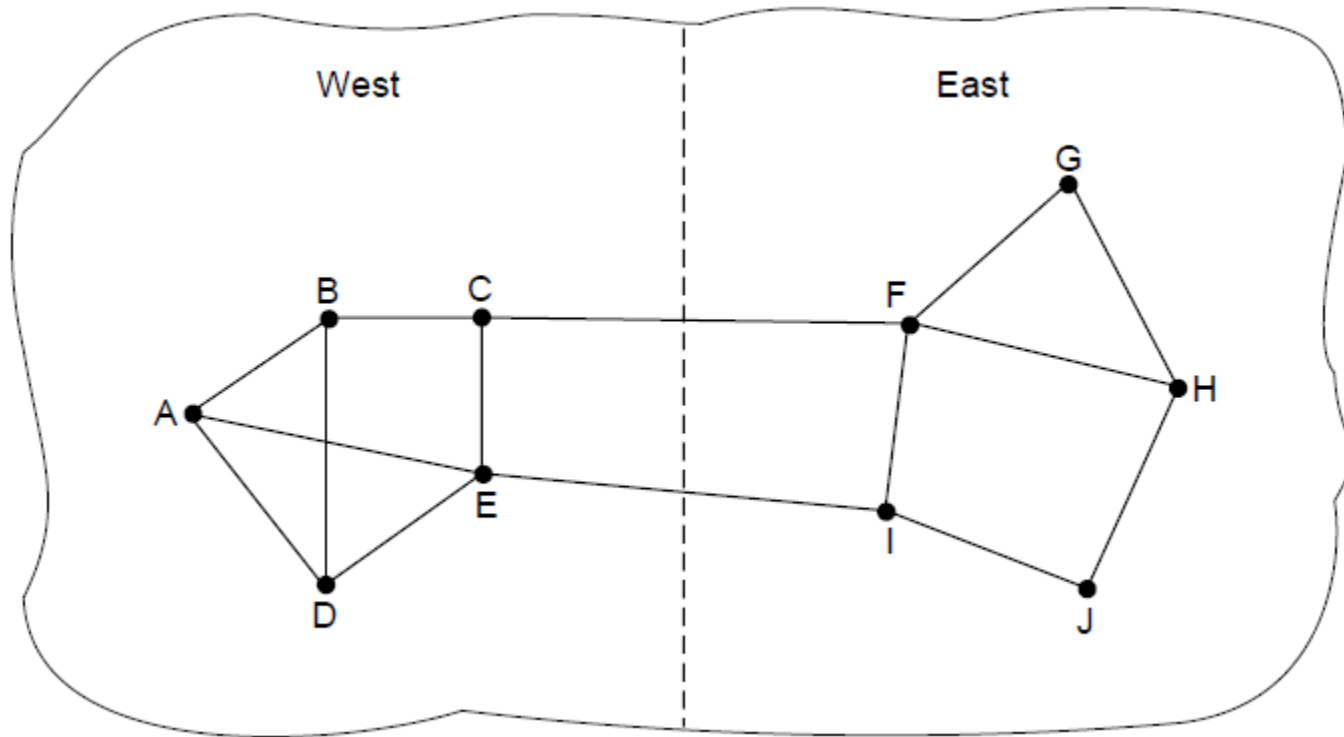
When too much traffic is offered, congestion sets in and performance degrades sharply.

# Approaches to Congestion Control



Timescales of approaches to congestion control

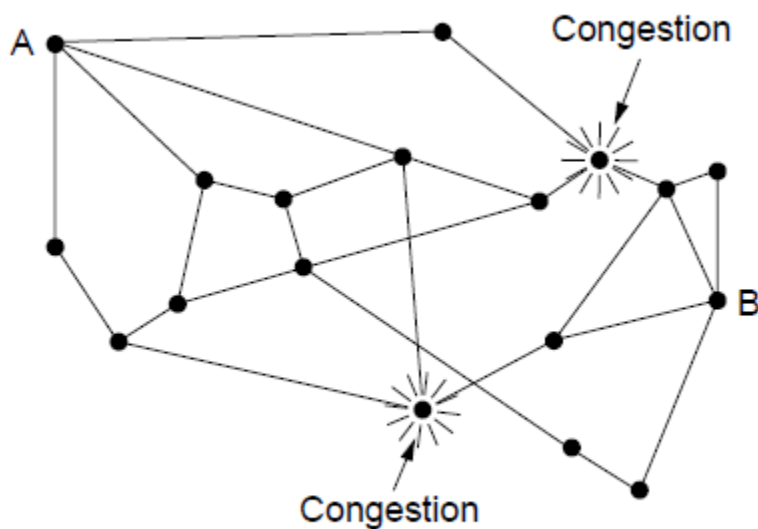
# Traffic-Aware Routing



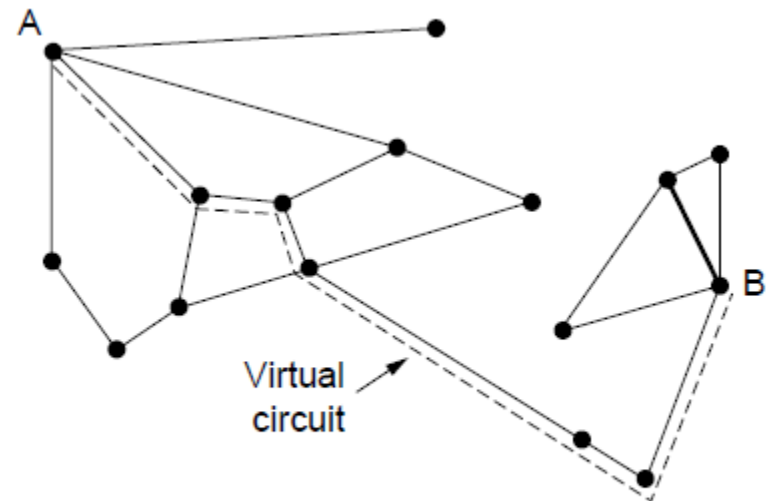
A network in which the East and West parts are connected by two links.



# Traffic Throttling (1)



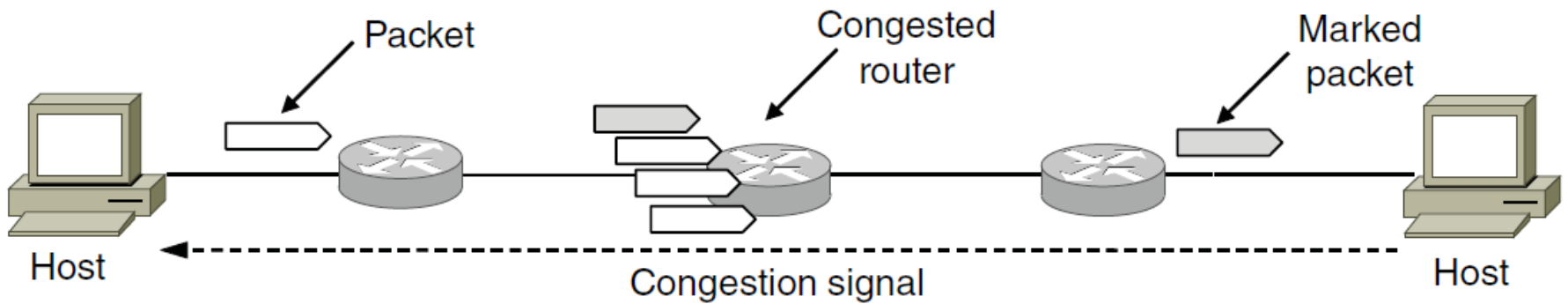
(a)



(b)

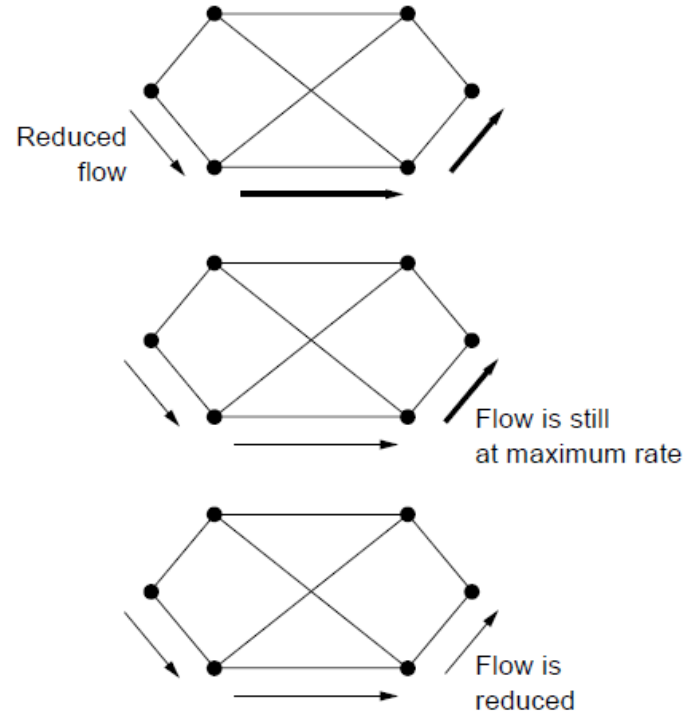
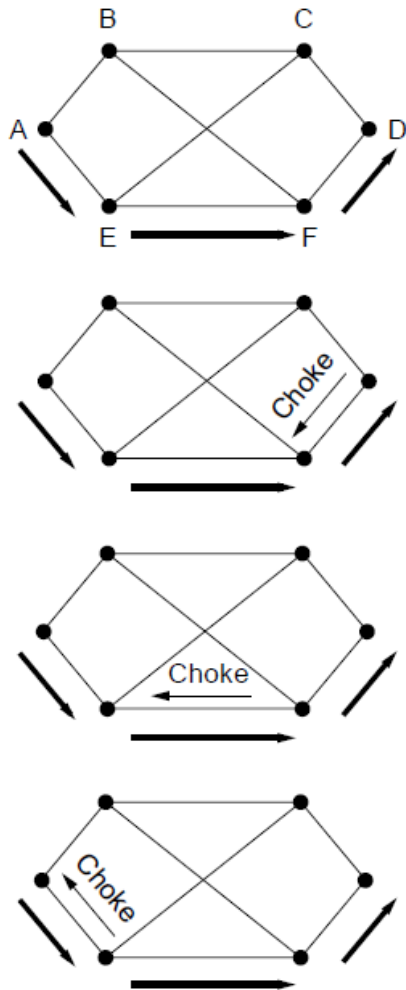
(a) A congested network. (b) The portion of the network that is not congested. A virtual circuit

# Traffic Throttling (2)



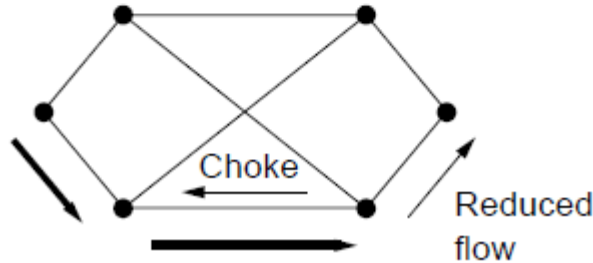
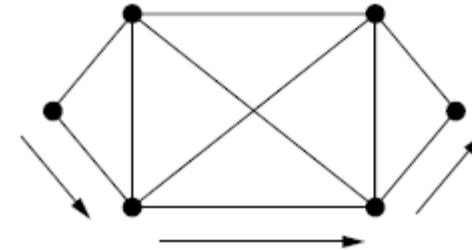
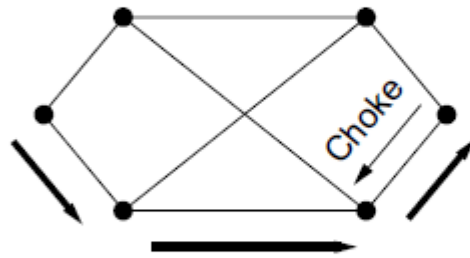
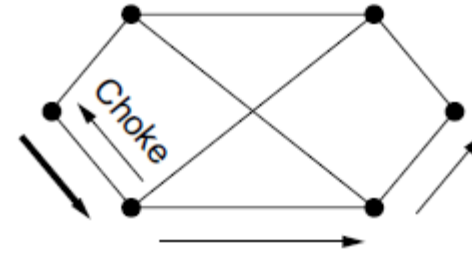
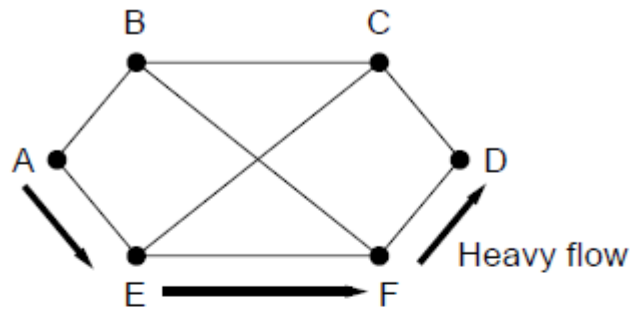
Explicit congestion notification

# Load Shedding (1)



A choke packet that affects only the source..

# Load Shedding (2)



A choke packet that affects each hop it passes through

# Quality of Service

- Application requirements
- Traffic shaping
- Packet scheduling
- Admission control
- Integrated services
- Differentiated services

# Application Requirements (1)

<b>Application</b>	<b>Bandwidth</b>	<b>Delay</b>	<b>Jitter</b>	<b>Loss</b>
Email	Low	Low	Low	Medium
File sharing	High	Low	Low	Medium
Web access	Medium	Medium	Low	Medium
Remote login	Low	Medium	Medium	Medium
Audio on demand	Low	Low	High	Low
Video on demand	High	Low	High	Low
Telephony	Low	High	High	Low
Videoconferencing	High	High	High	Low

How stringent the quality-of-service requirements are.

# Categories of QoS and Examples

## 1. Constant bit rate

- Telephony

## 2. Real-time variable bit rate

- Compressed videoconferencing

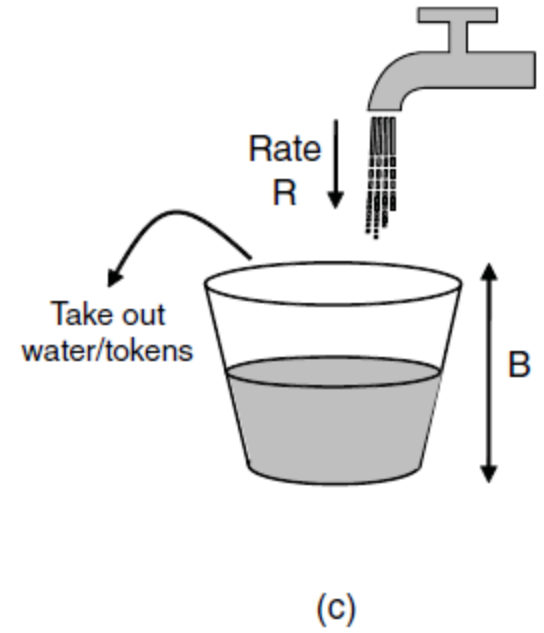
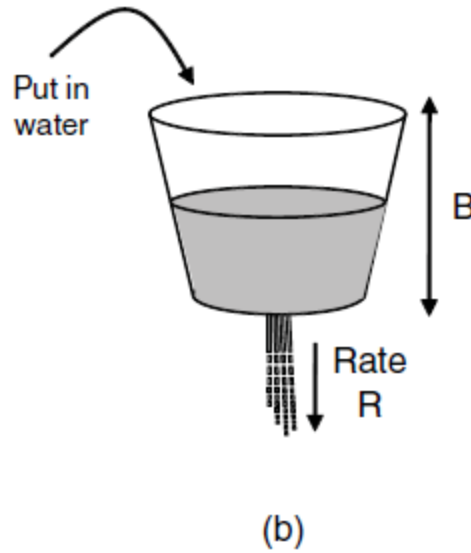
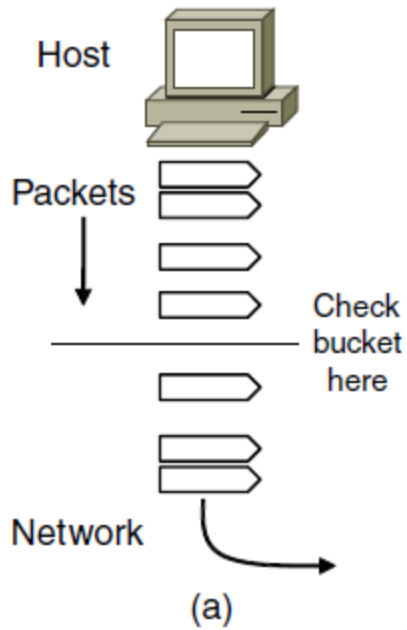
## 3. Non-real-time variable bit rate

- Watching a movie on demand

## 4. Available bit rate

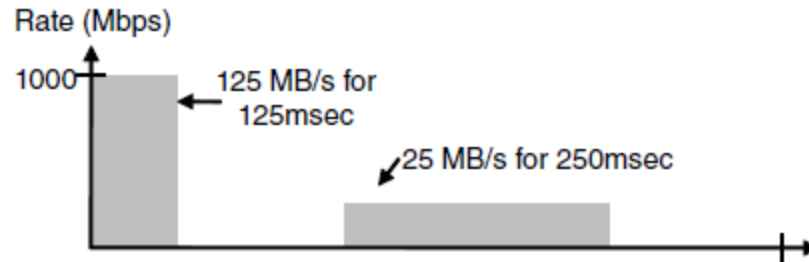
- File transfer

# Traffic Shaping (1)

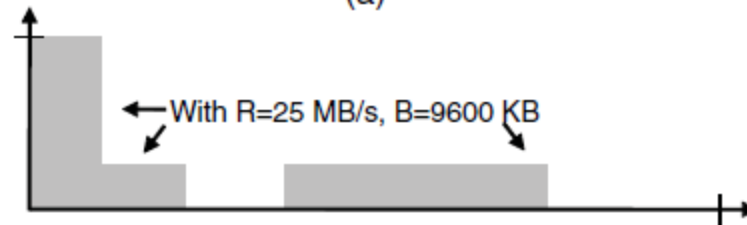




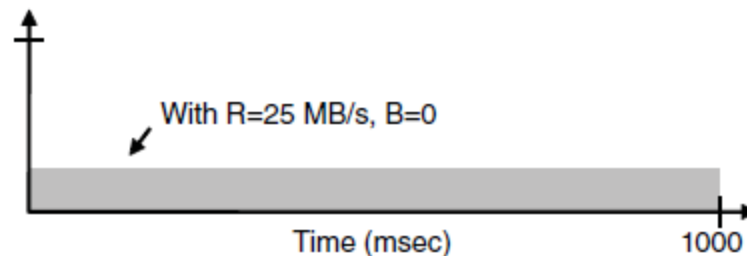
# Traffic Shaping (2)



(a)



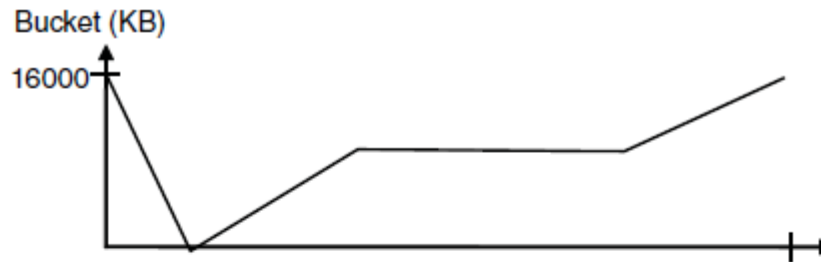
(b)



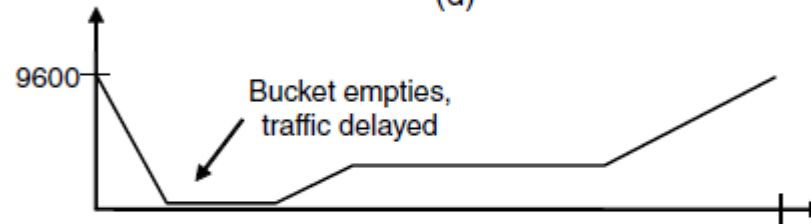
(c)

(a) Traffic from a host. Output shaped by a token bucket of rate 200 Mbps and capacity (b) 9600 KB,

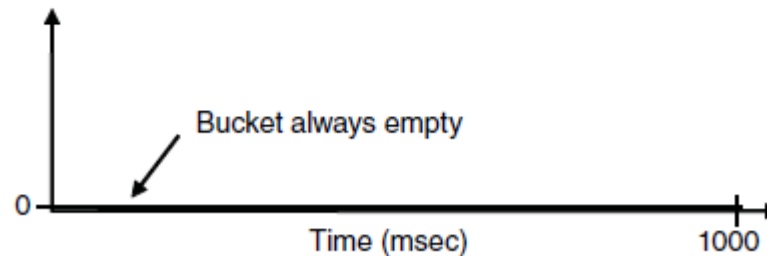
# Traffic Shaping (3)



(d)



(e)



(f)

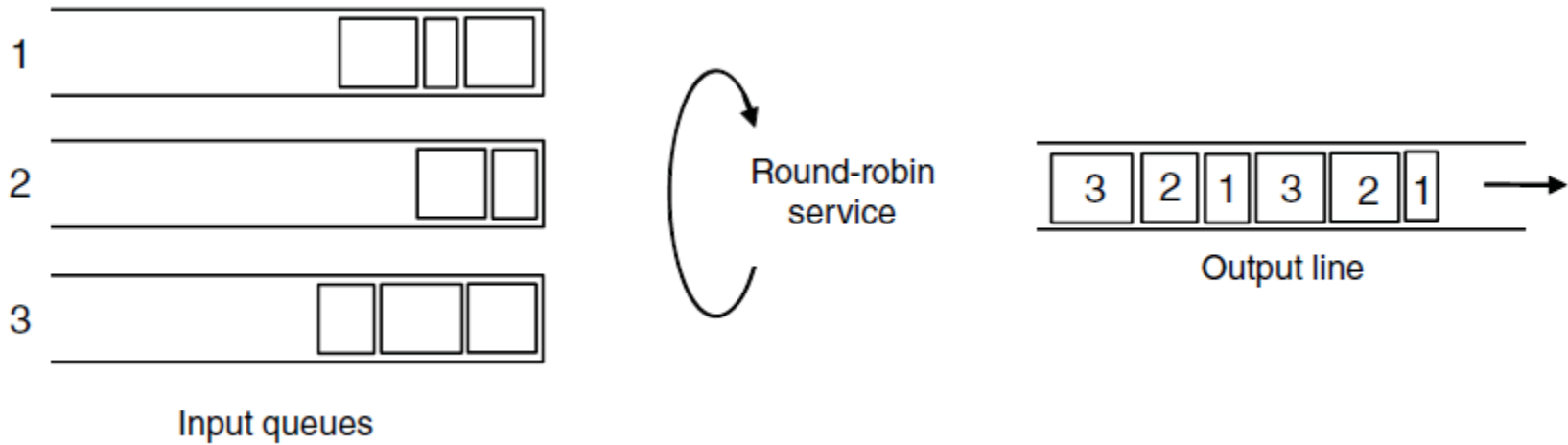
Token bucket level for shaping with rate 200 Mbps and capacity (d) 16000 KB, (e) 9600 KB, and (f)

# Packet Scheduling (1)

Kinds of resources can potentially be reserved for different flows:

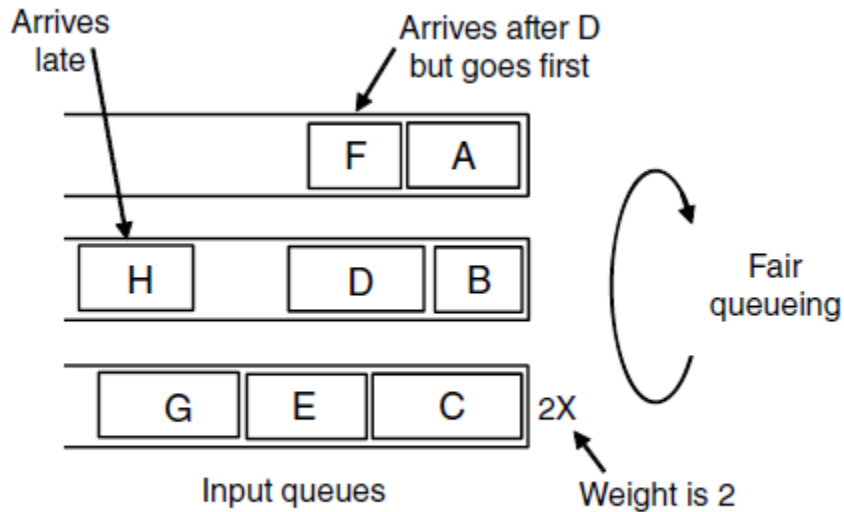
1. Bandwidth.
2. Buffer space.
3. CPU cycles.

# Packet Scheduling (2)



Round-robin Fair Queuing

# Packet Scheduling (3)



(a)

Packet	Arrival time	Length	Finish time	Output order
A	0	8	8	1
B	5	6	11	3
C	5	10	10	2
D	8	9	20	7
E	8	8	14	4
F	10	6	16	5
G	11	10	19	6
H	20	8	28	8

(b)

(a) Weighted Fair Queueing.

(b) Finishing times for the packets.

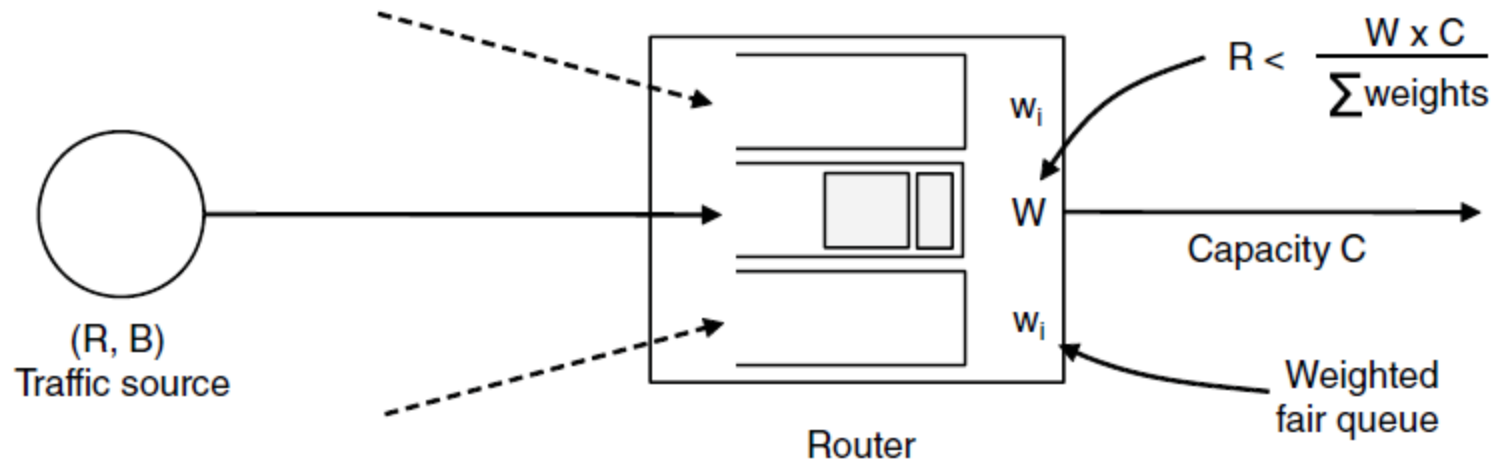
# Admission Control (1)

<b>Parameter</b>	<b>Unit</b>
Token bucket rate	Bytes/sec
Token bucket size	Bytes
Peak data rate	Bytes/sec
Minimum packet size	Bytes
Maximum packet size	Bytes

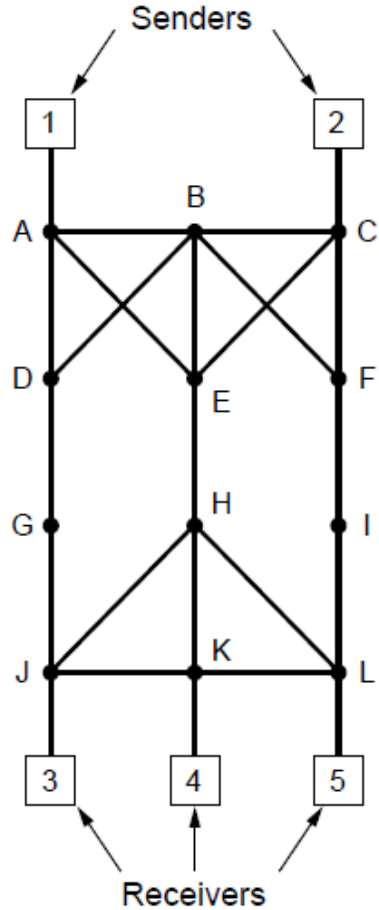
An example flow specification

# Admission Control (2)

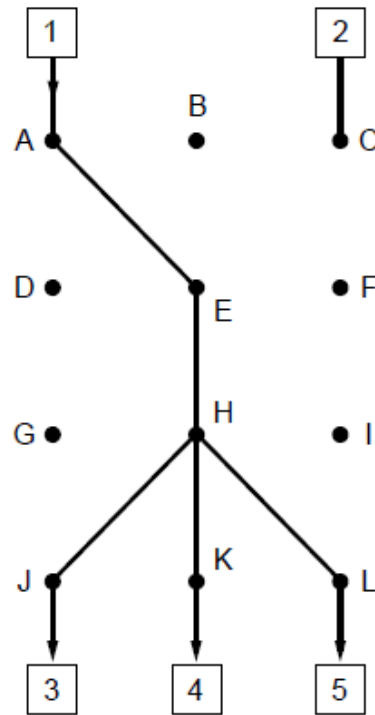
Bandwidth and delay guarantees with



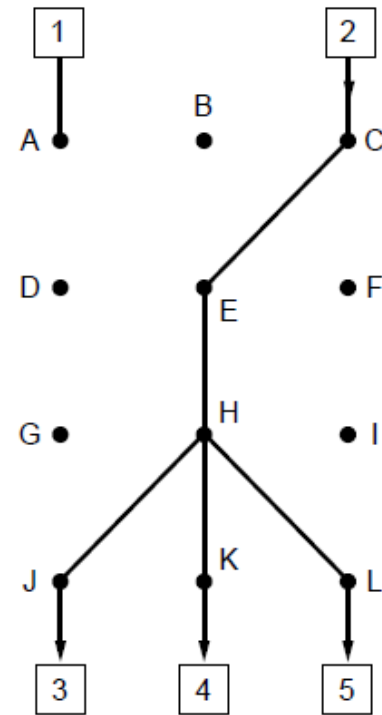
# Integrated Services (1)



(a)



(b)

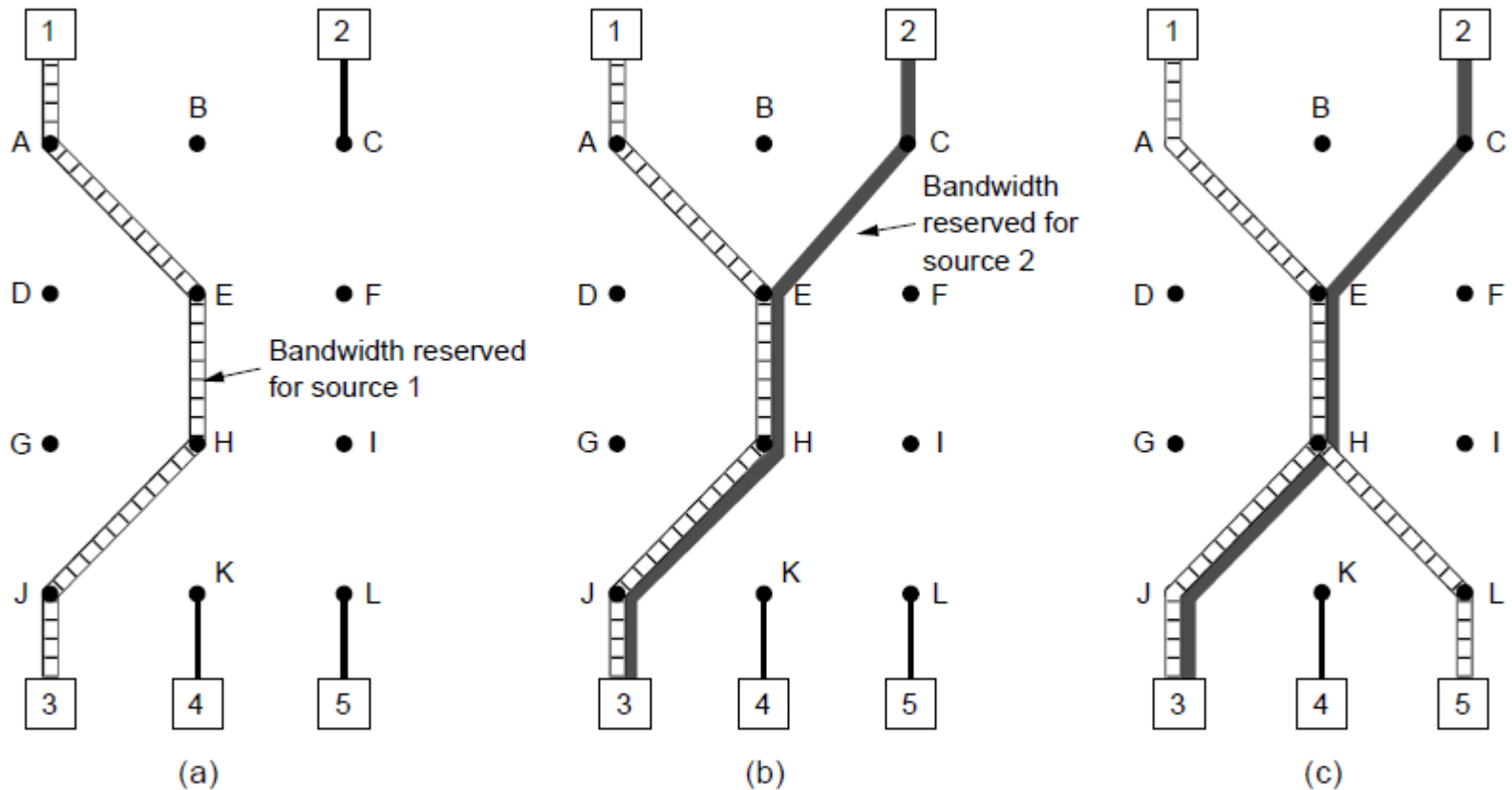


(c)

(a) A network. (b) The multicast spanning tree for host 1.

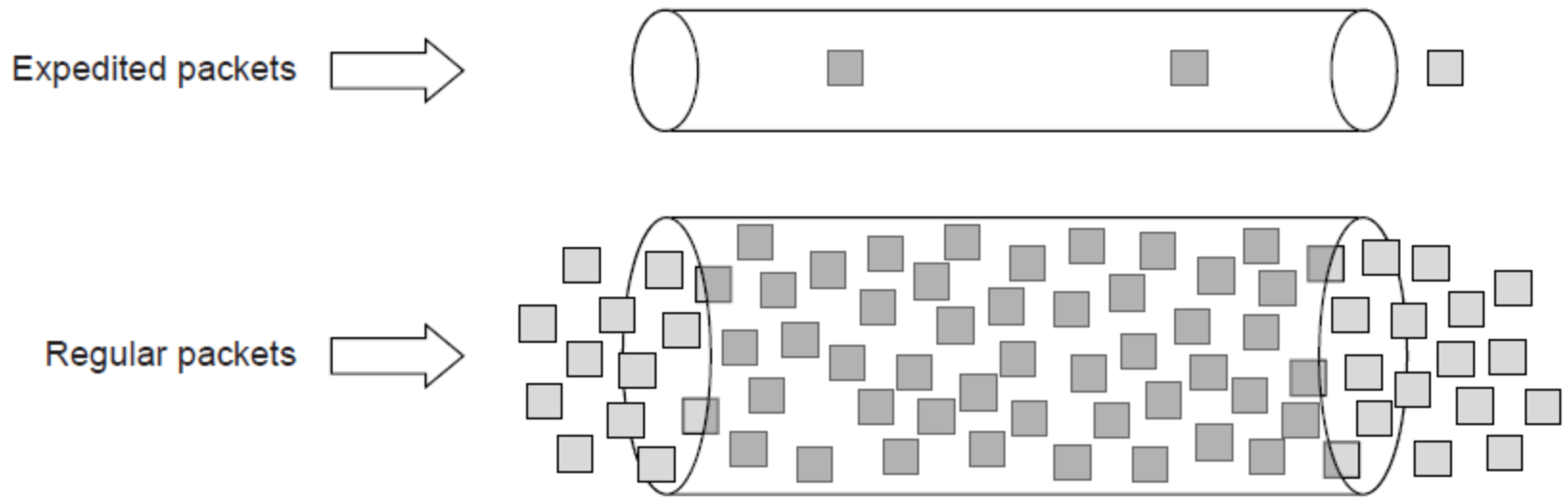


# Integrated Services (2)



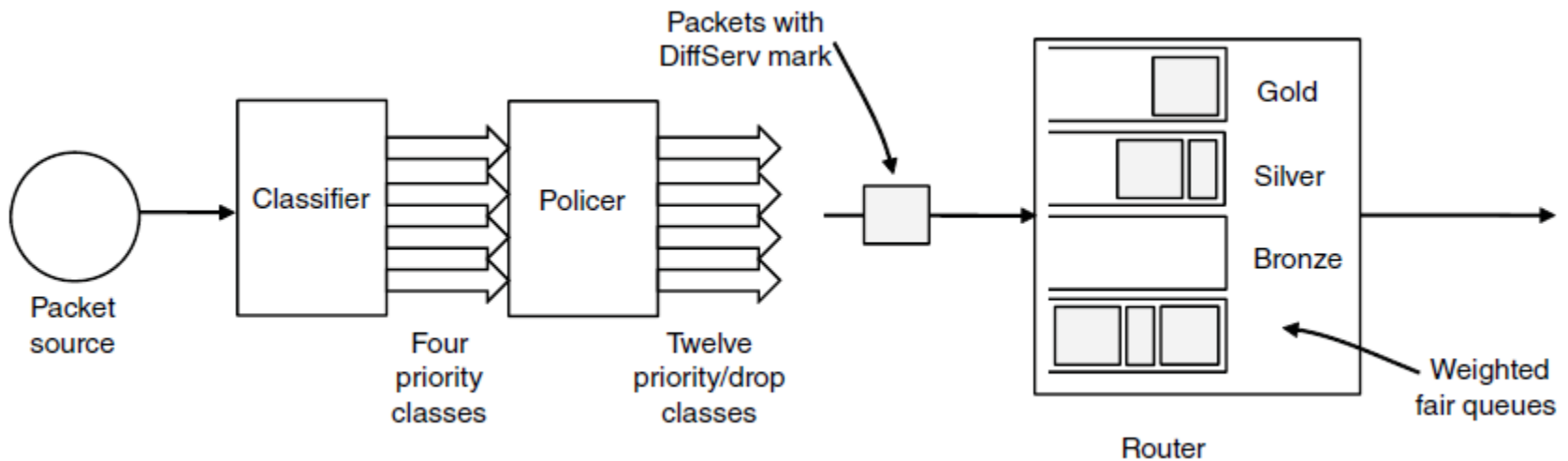
- (a) Host 3 requests a channel to host 1. (b) Host 3 then requests a second channel, to host 2. (c) Host 5 requests a channel to host 1

# Differentiated Services (1)



Expedited packets experience a traffic-free network

# Differentiated Services (2)



A possible implementation of assured forwarding

# Internetworking

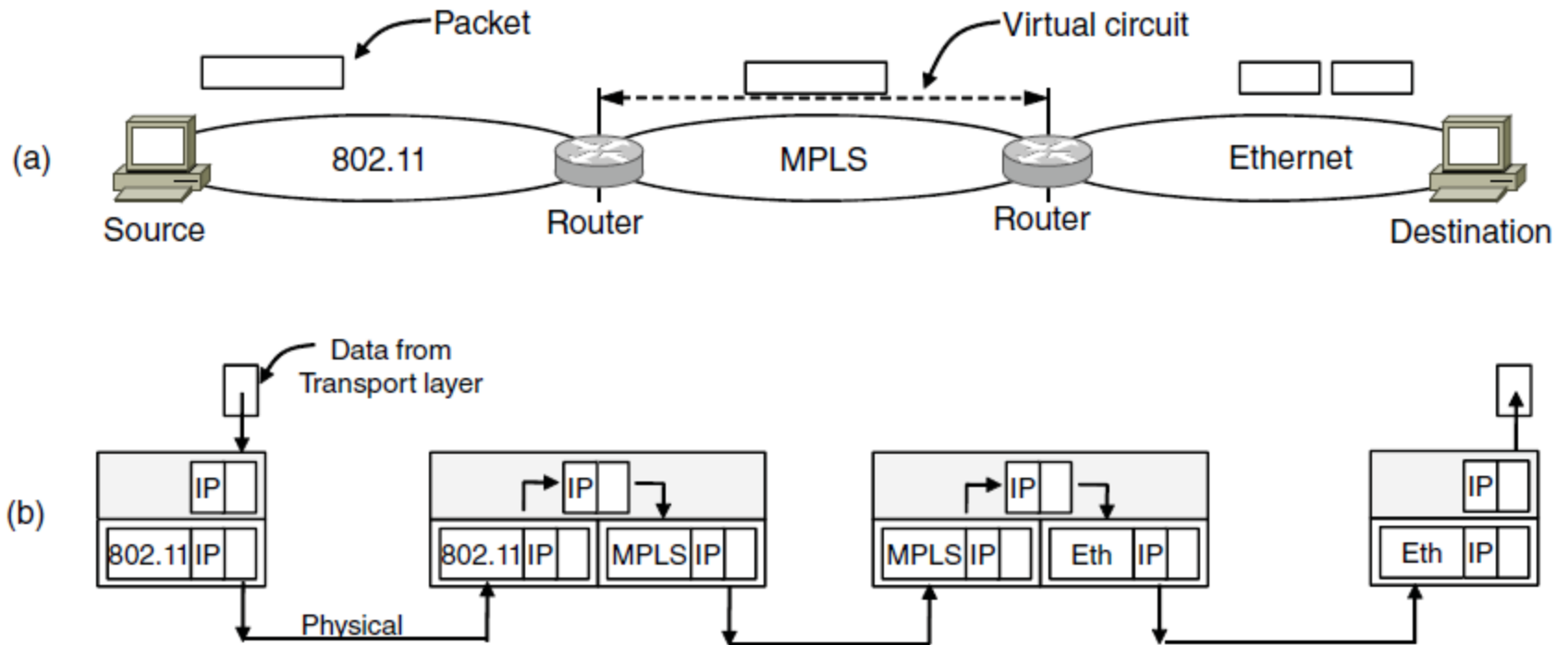
- How networks differ
- How networks can be connected
- Tunneling
- Internetwork routing
- Packet fragmentation

# How Networks Differ

Item	Some Possibilities
Service offered	Connectionless versus connection oriented
Addressing	Different sizes, flat or hierarchical
Broadcasting	Present or absent (also multicast)
Packet size	Every network has its own maximum
Ordering	Ordered and unordered delivery
Quality of service	Present or absent; many different kinds
Reliability	Different levels of loss
Security	Privacy rules, encryption, etc.
Parameters	Different timeouts, flow specifications, etc.
Accounting	By connect time, packet, byte, or not at all

Some of the many ways networks can differ

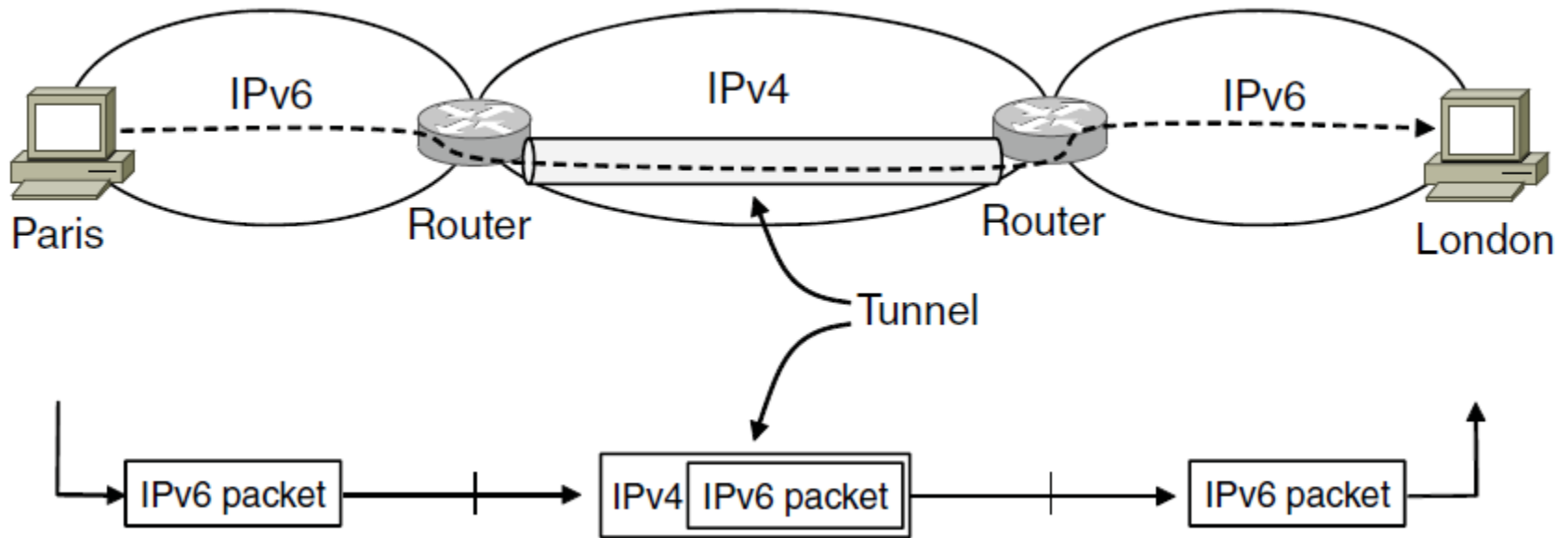
# How Networks Can Be Connected



(a) A packet crossing different networks.

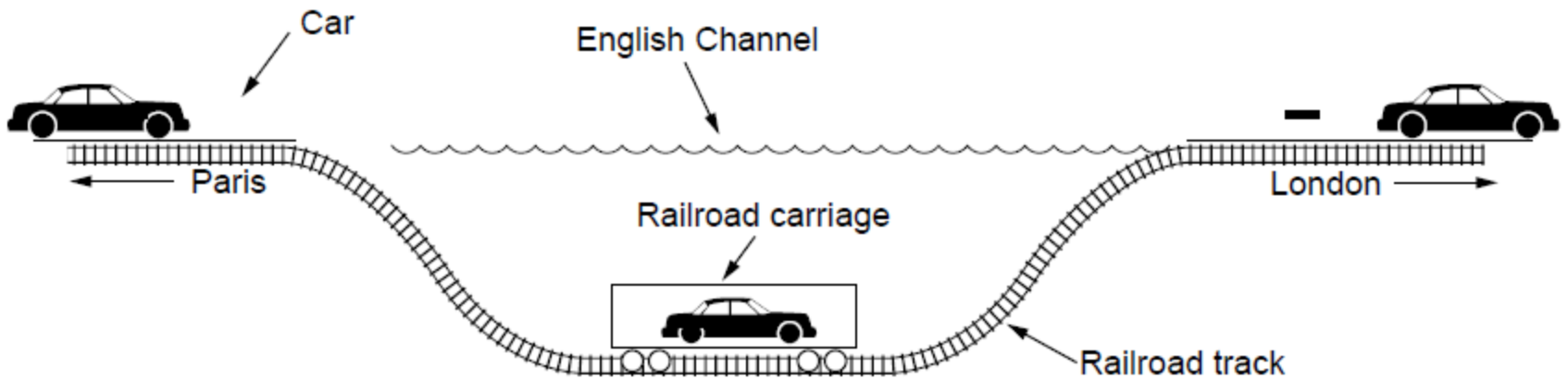
(b) Network and link layer protocol processing

# Tunneling (1)



Tunneling a packet from Paris to London.

# Tunneling (2)



Tunneling a car from France to England

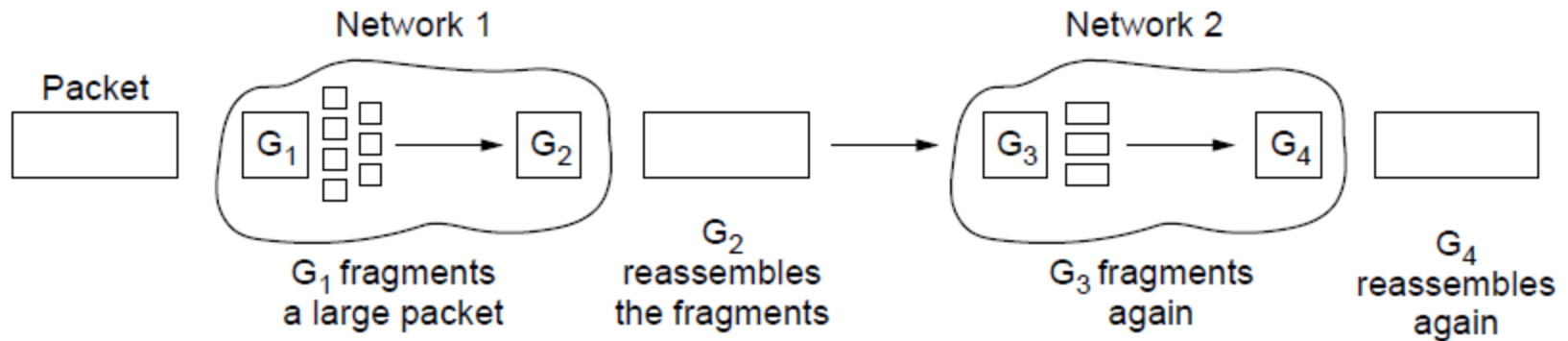


# Packet Fragmentation (1)

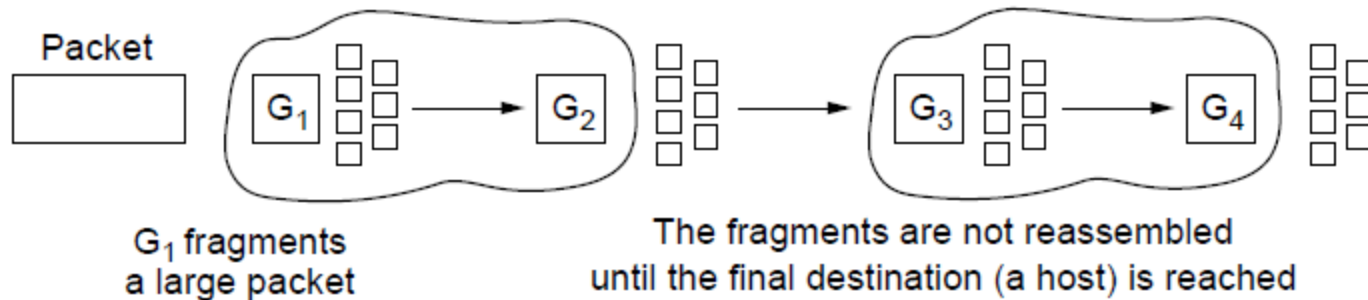
Packet size issues:

1. Hardware
2. Operating system
3. Protocols
4. Compliance with (inter)national standard.
5. Reduce error-induced retransmissions
6. Prevent packet occupying channel too long.

# Packet Fragmentation (2)



(a)



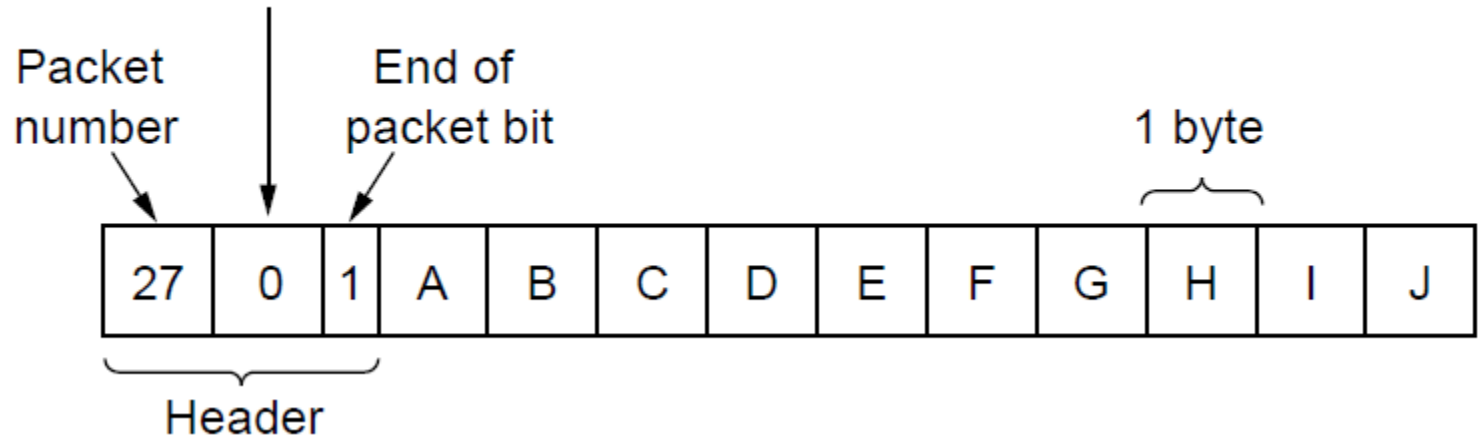
(b)

(a) Transparent fragmentation.

(b) Nontransparent fragmentation

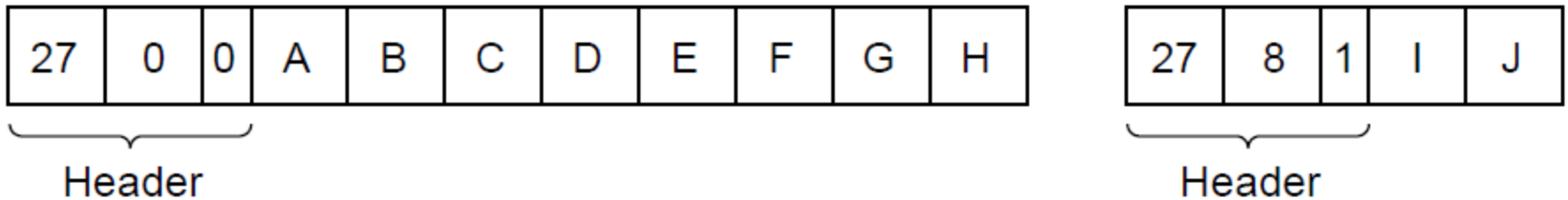
# Packet Fragmentation (3)

Number of the first elementary fragment in this packet



Fragmentation when the elementary data size is 1 byte.

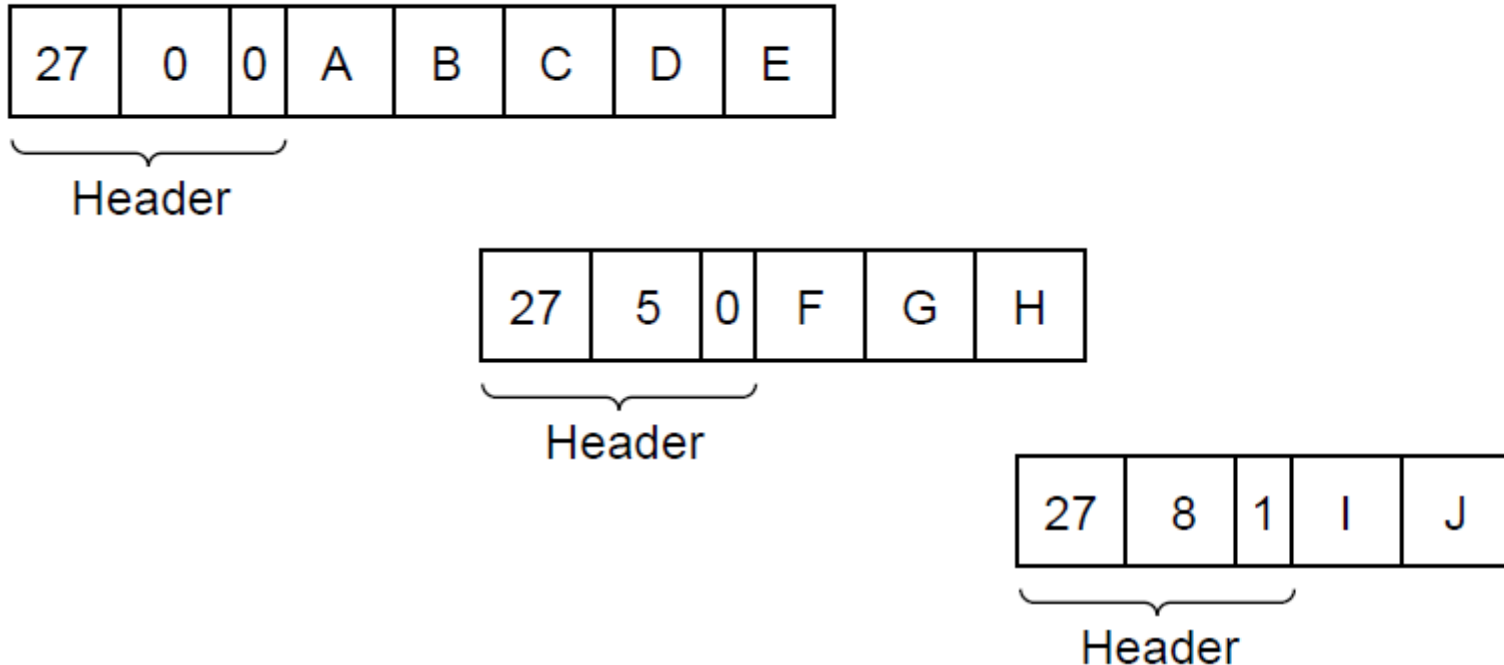
# Packet Fragmentation (4)



Fragmentation when the elementary data size is 1 byte

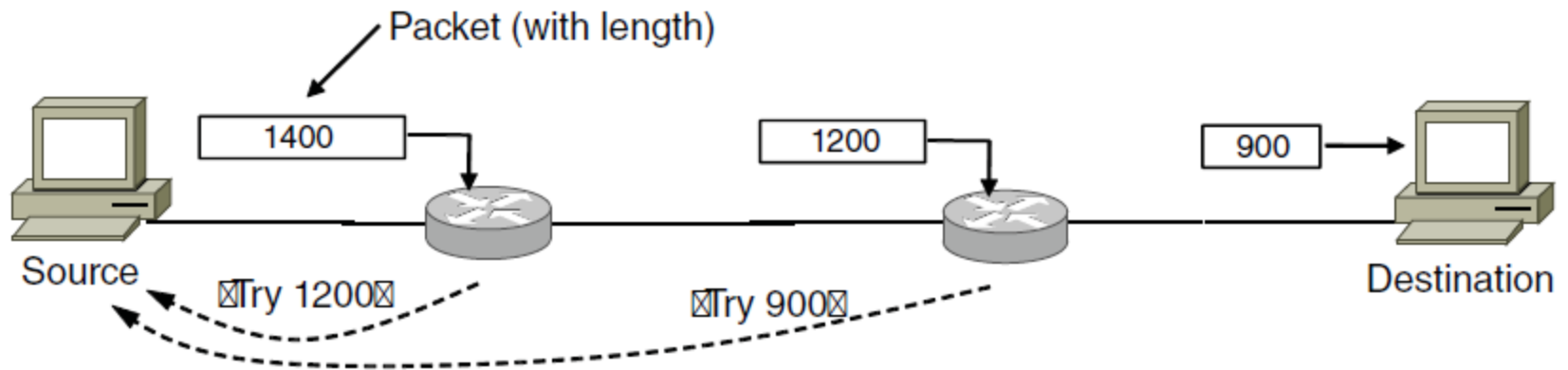
**(b)** Fragments after passing through a network

# Packet Fragmentation (5)



Fragmentation when the elementary data size is 1 byte  
(c) Fragments after passing through a size 5 gateway.

# Packet Fragmentation (6)



Path MTU Discovery

# The Network Layer Principles (1)

1. Make sure it works
  2. Keep it simple
  3. Make clear choices
  4. Exploit modularity
  5. Expect heterogeneity
- ...

# The Network Layer Principles (2)

...

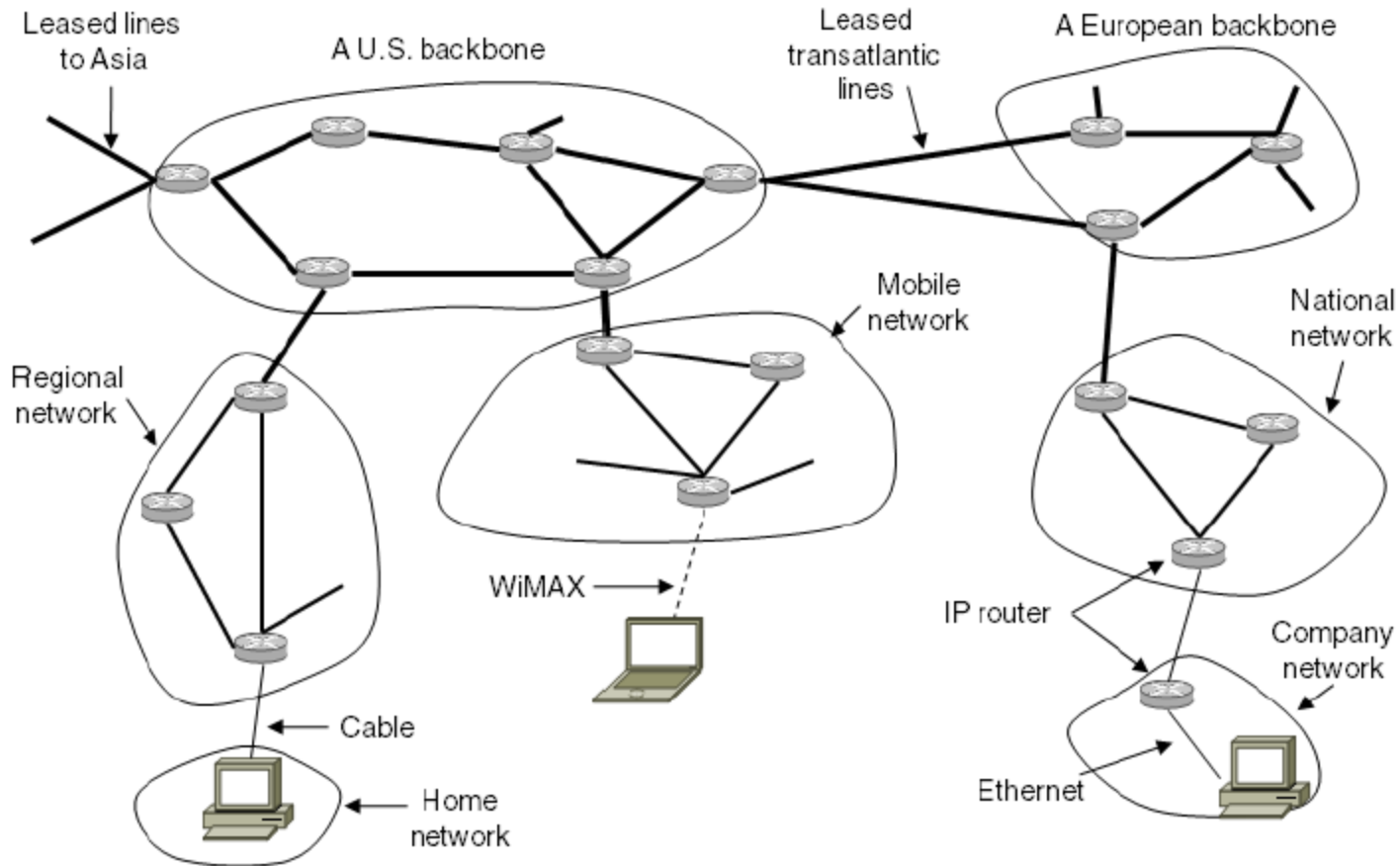
6. Avoid static options and parameters
7. Look for good design (not perfect)
8. Strict sending, tolerant receiving
9. Think about scalability
10. Consider performance and cost



# The Network Layer in the Internet (1)

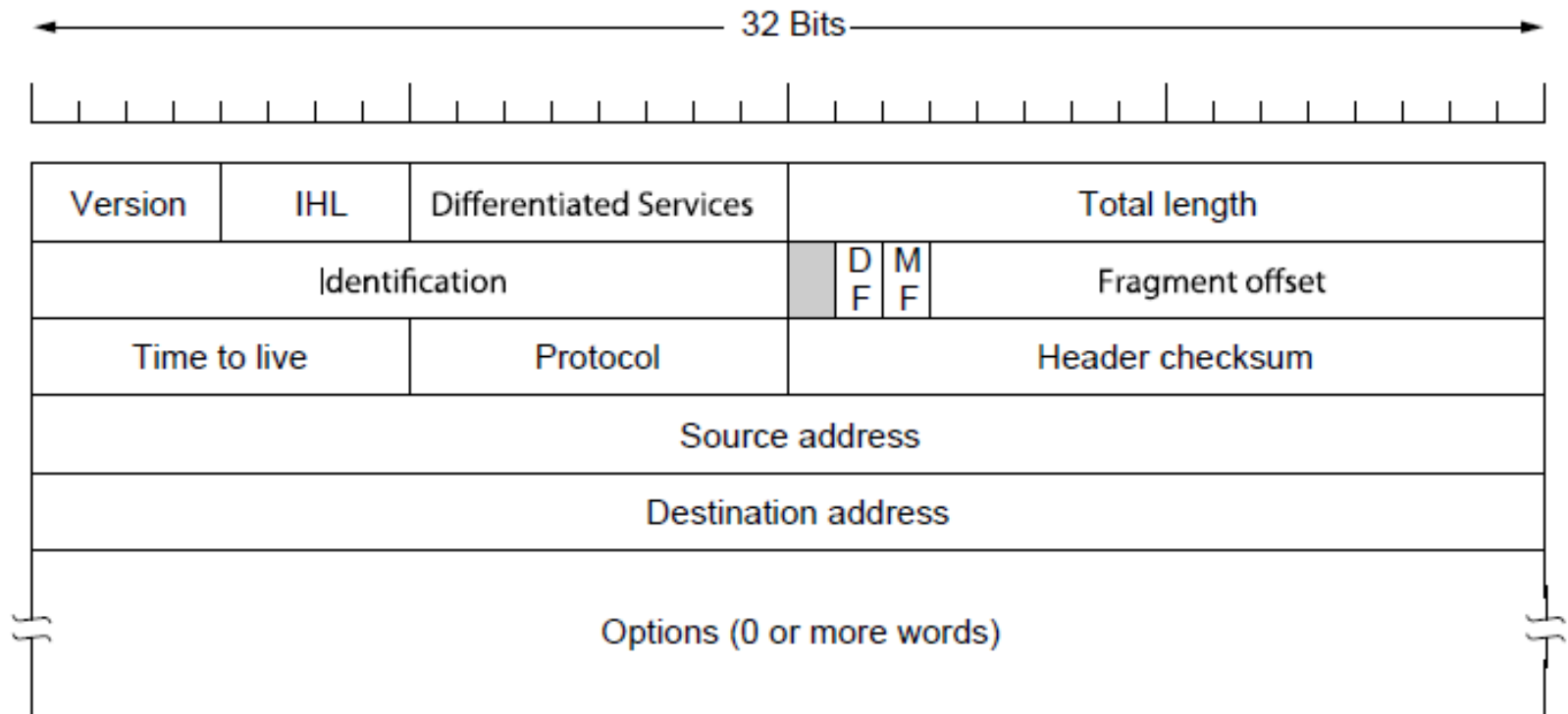
- The IP Version 4 Protocol
- IP Addresses
- IP Version 6
- Internet Control Protocols
- Label Switching and MPLS
- OSPF—An Interior Gateway Routing Protocol
- BGP—The Exterior Gateway Routing Protocol
- Internet Multicasting
- Mobile IP

# The Network Layer in the Internet (2)



The Internet is an interconnected collection of many networks.

# The IP Version 4 Protocol (1)



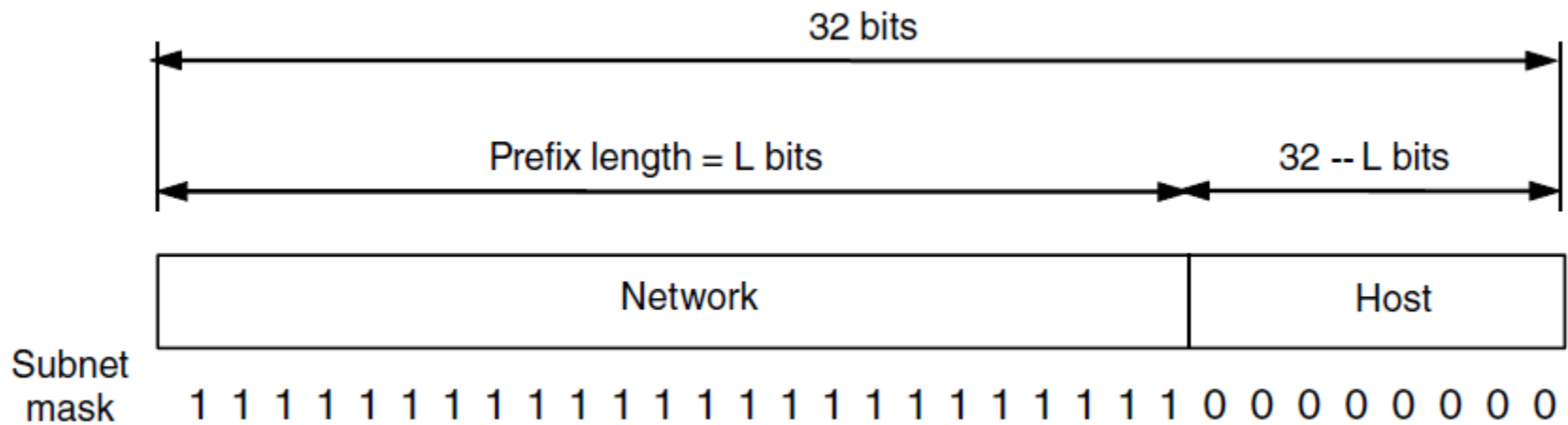
The IPv4 (Internet Protocol) header.

# The IP Version 4 Protocol (2)

Option	Description
Security	Specifies how secret the datagram is
Strict source routing	Gives the complete path to be followed
Loose source routing	Gives a list of routers not to be missed
Record route	Makes each router append its IP address
Timestamp	Makes each router append its address and timestamp

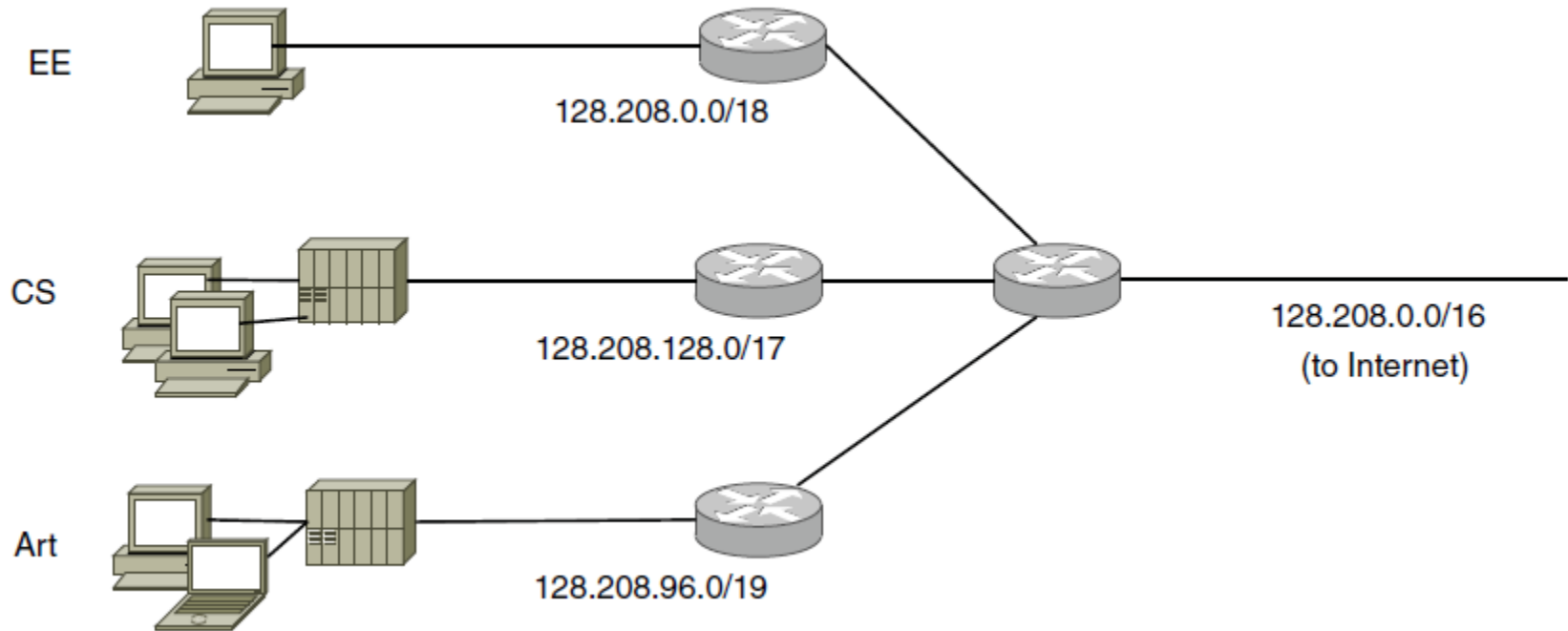
Some of the IP options.

# IP Addresses (1)



An IP prefix.

# IP Addresses (2)



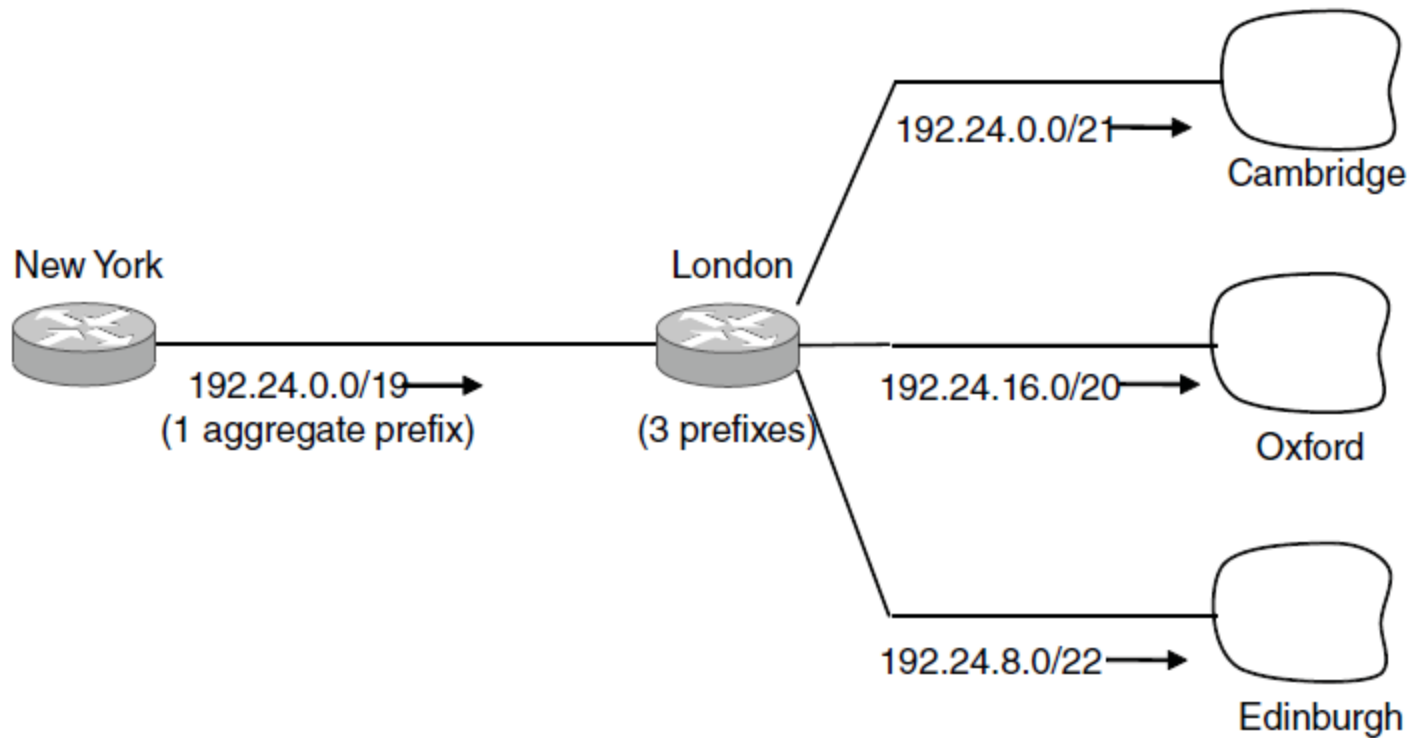
Splitting an IP prefix into separate networks with subnetting.

# IP Addresses (3)

<b>University</b>	<b>First address</b>	<b>Last address</b>	<b>How many</b>	<b>Prefix</b>
Cambridge	194.24.0.0	194.24.7.255	2048	194.24.0.0/21
Edinburgh	194.24.8.0	194.24.11.255	1024	194.24.8.0/22
(Available)	194.24.12.0	194.24.15.255	1024	194.24.12/22
Oxford	194.24.16.0	194.24.31.255	4096	194.24.16.0/20

A set of IP address assignments

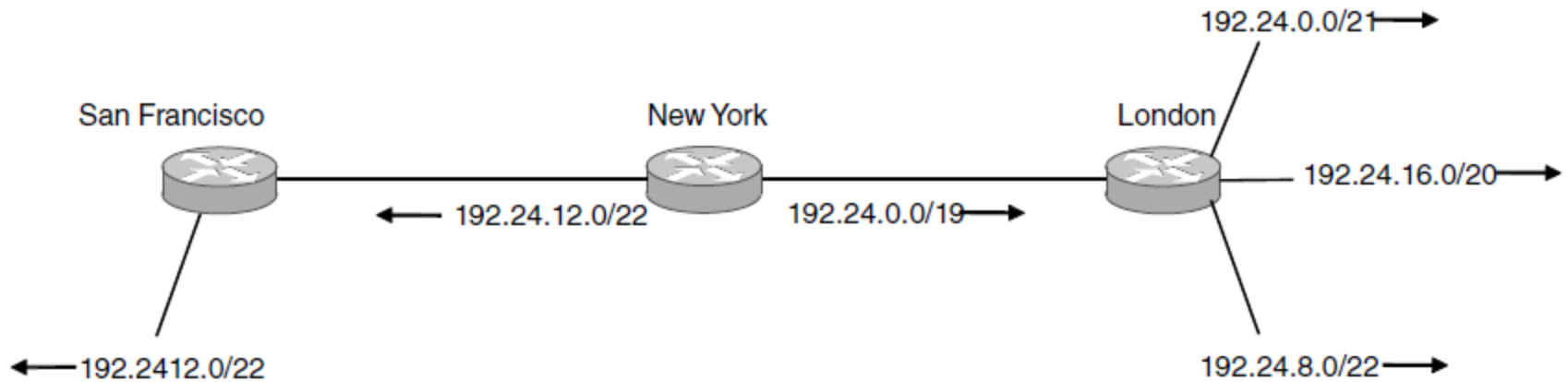
# IP Addresses (4)



Aggregation of IP prefixes

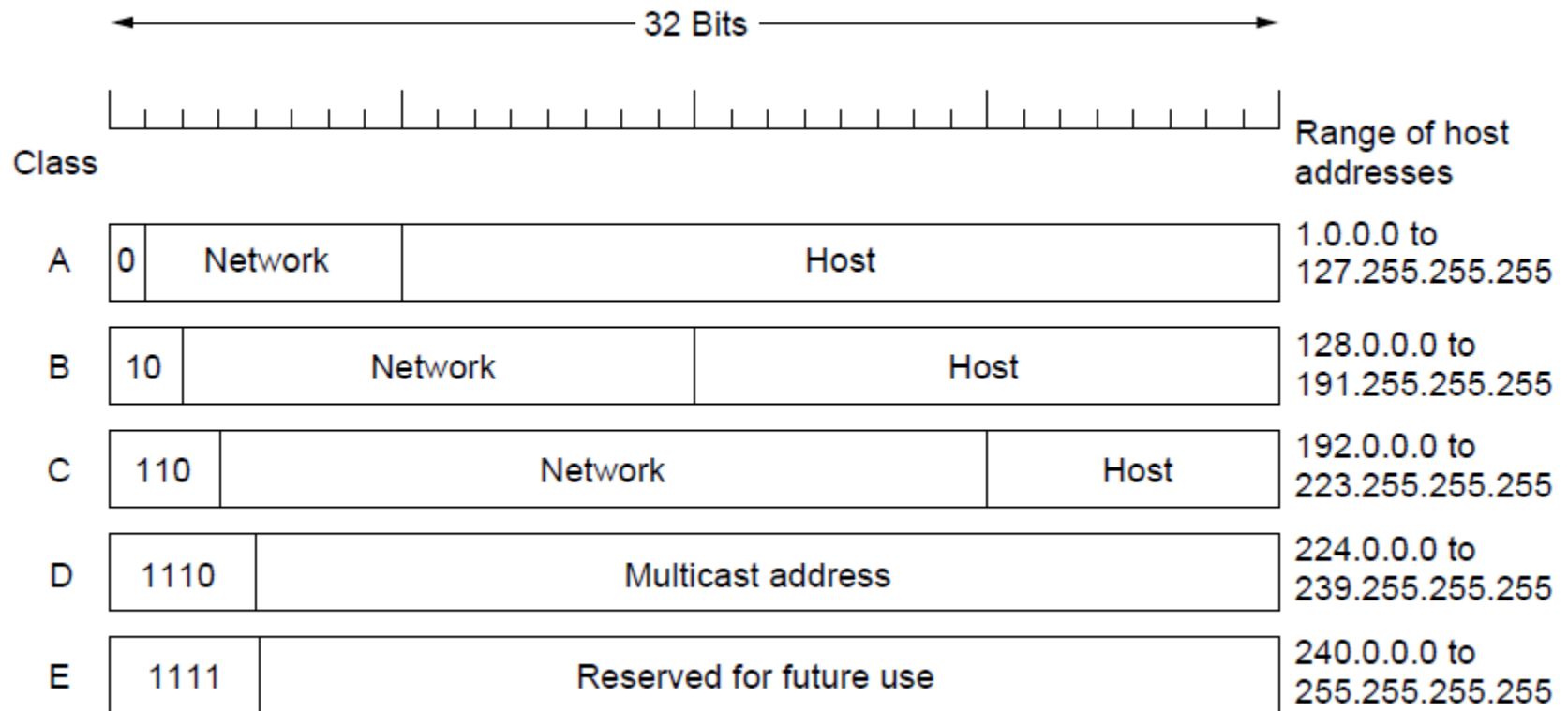


# IP Addresses (5)



Longest matching prefix routing at the New York router.

# IP Addresses (6)



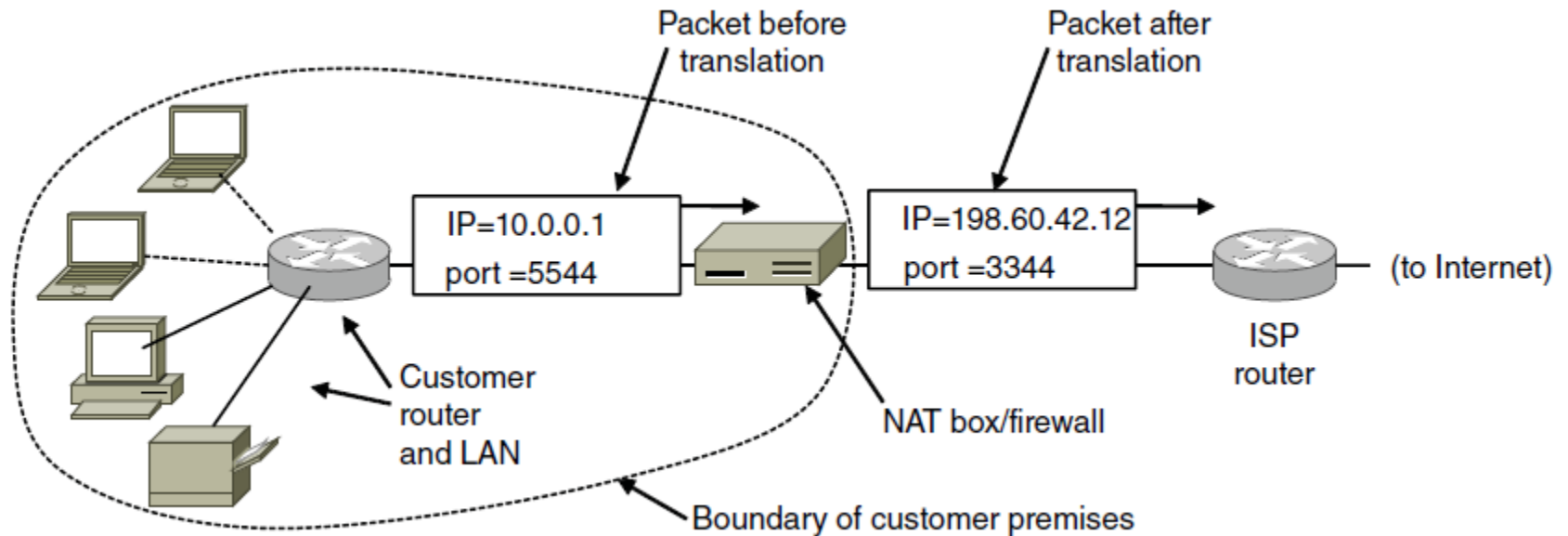
## IP address formats

# IP Addresses (7)

0 0		This host
0 0      ...      0 0	Host	A host on this network
1 1		Broadcast on the local network
Network	1 1 1 1      ...      1 1 1 1	Broadcast on a distant network
127	(Anything)	Loopback

Special IP addresses

# IP Addresses (8)

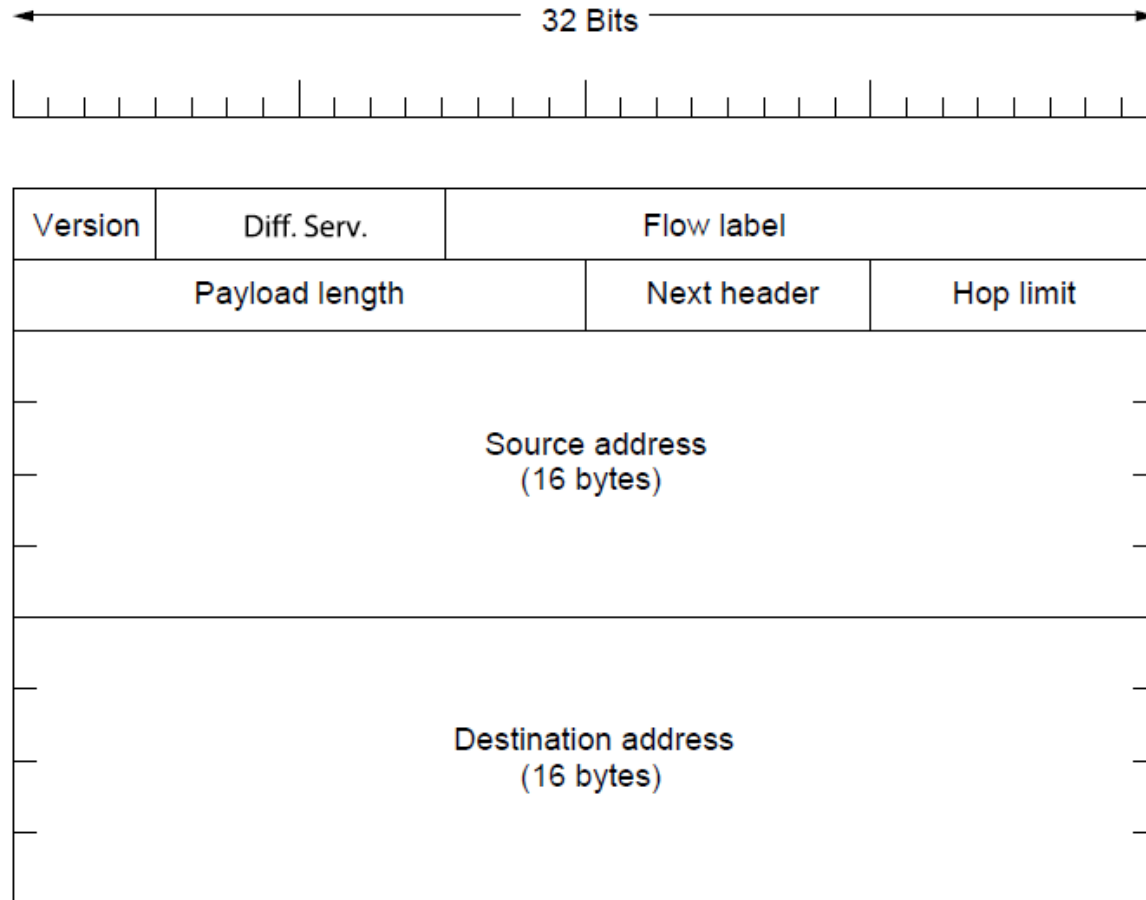


Placement and operation of a NAT box.

# IP Version 6 Goals

- Support billions of hosts
- Reduce routing table size
- Simplify protocol
- Better security
- Attention to type of service
- Aid multicasting
- Roaming host without changing address
- Allow future protocol evolution
- Permit coexistence of old, new protocols. . .

# IP Version 6 (1)



The IPv6 fixed header (required).

# IP Version 6 (2)

<b>Extension header</b>	<b>Description</b>
Hop-by-hop options	Miscellaneous information for routers
Destination options	Additional information for the destination
Routing	Loose list of routers to visit
Fragmentation	Management of datagram fragments
Authentication	Verification of the sender's identity
Encrypted security payload	Information about the encrypted contents

IPv6 extension headers

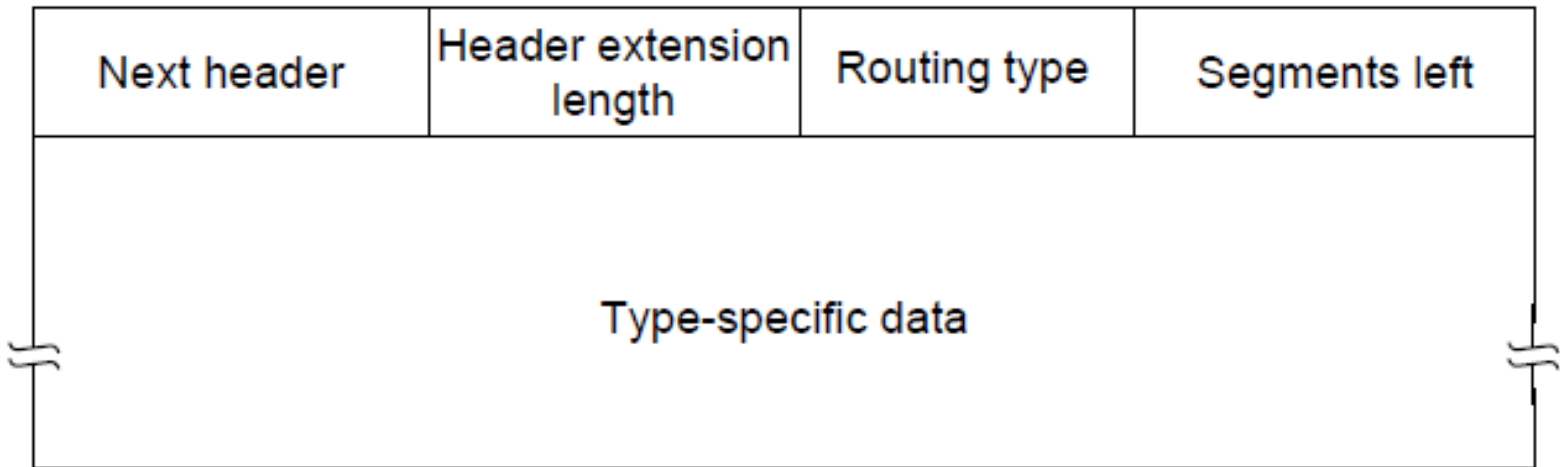
# IP Version 6 (3)

Next header	0	194	4
Jumbo payload length			

The hop-by-hop extension header for large datagrams (jumbograms).



# IP Version 6 (4)



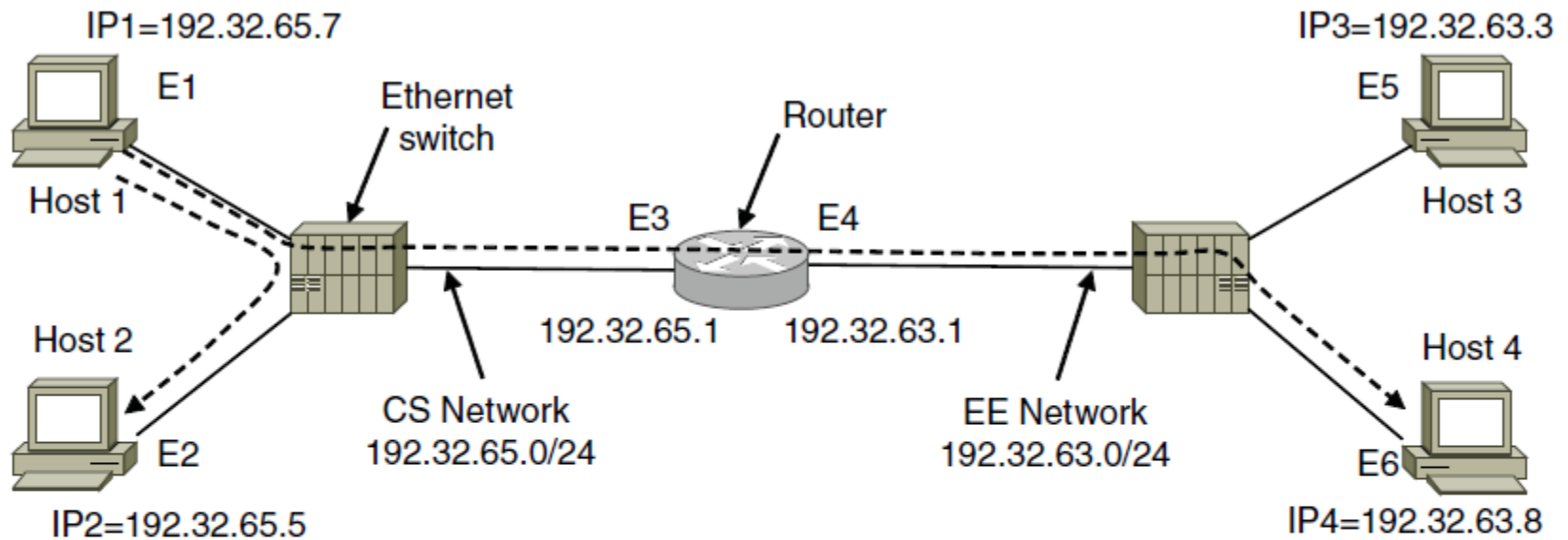
The extension header for routing.

# Internet Control Protocols (1)

Message type	Description
Destination unreachable	Packet could not be delivered
Time exceeded	Time to live field hit 0
Parameter problem	Invalid header field
Source quench	Choke packet
Redirect	Teach a router about geography
Echo and Echo reply	Check if a machine is alive
Timestamp request/reply	Same as Echo, but with timestamp
Router advertisement/solicitation	Find a nearby router

The principal ICMP message types.

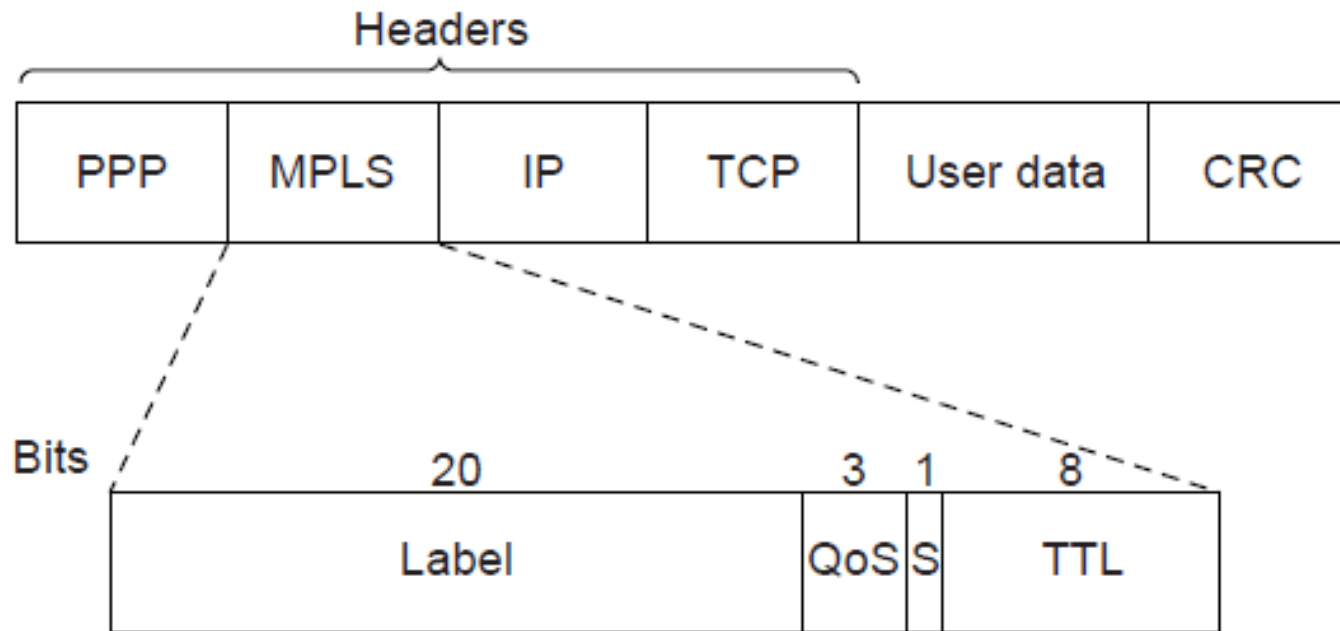
# Internet Control Protocols (2)



Frame	Source IP	Source Eth.	Destination IP	Destination Eth.
Host 1 to 2, on CS net	IP1	E1	IP2	E2
Host 1 to 4, on CS net	IP1	E1	IP4	E3
Host 1 to 4, on EE net	IP1	E4	IP4	E6

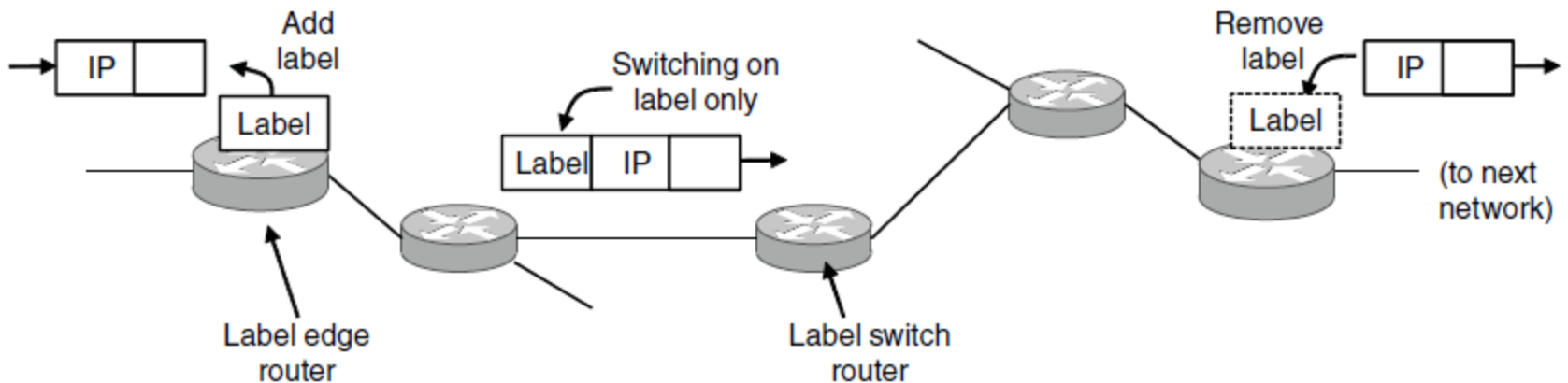
Two switched Ethernet LANs joined by a router

# Label Switching and MPLS (1)



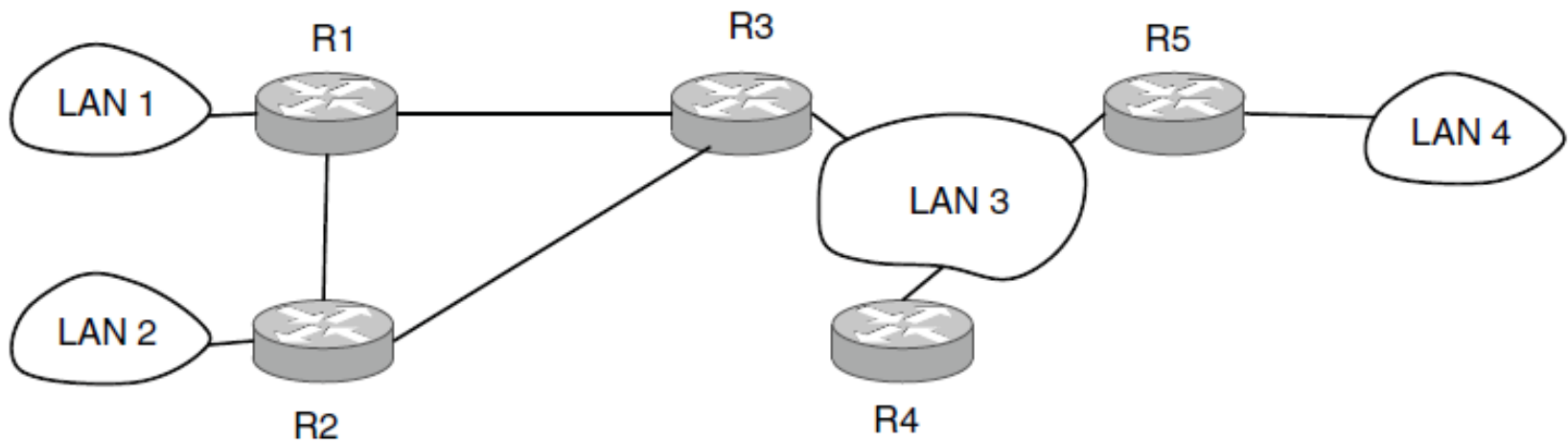
Transmitting a TCP segment using IP, MPLS, and PPP.

# Label Switching and MPLS (2)



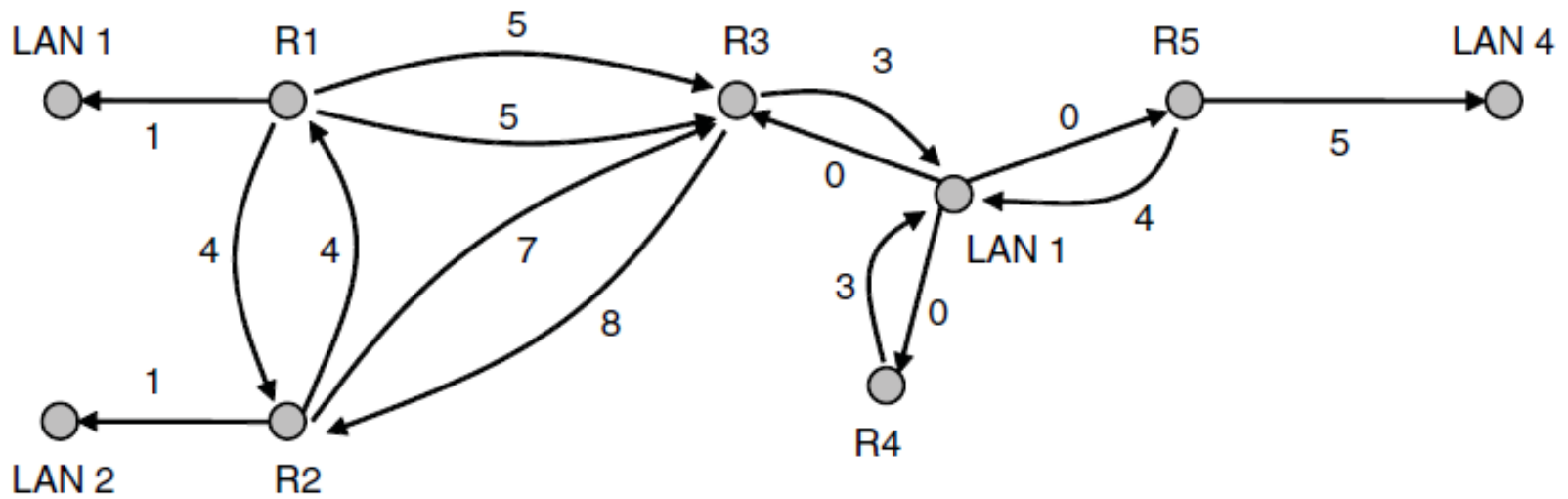
Forwarding an IP packet through an MPLS network

# OSPF—An Interior Gateway Routing Protocol (1)



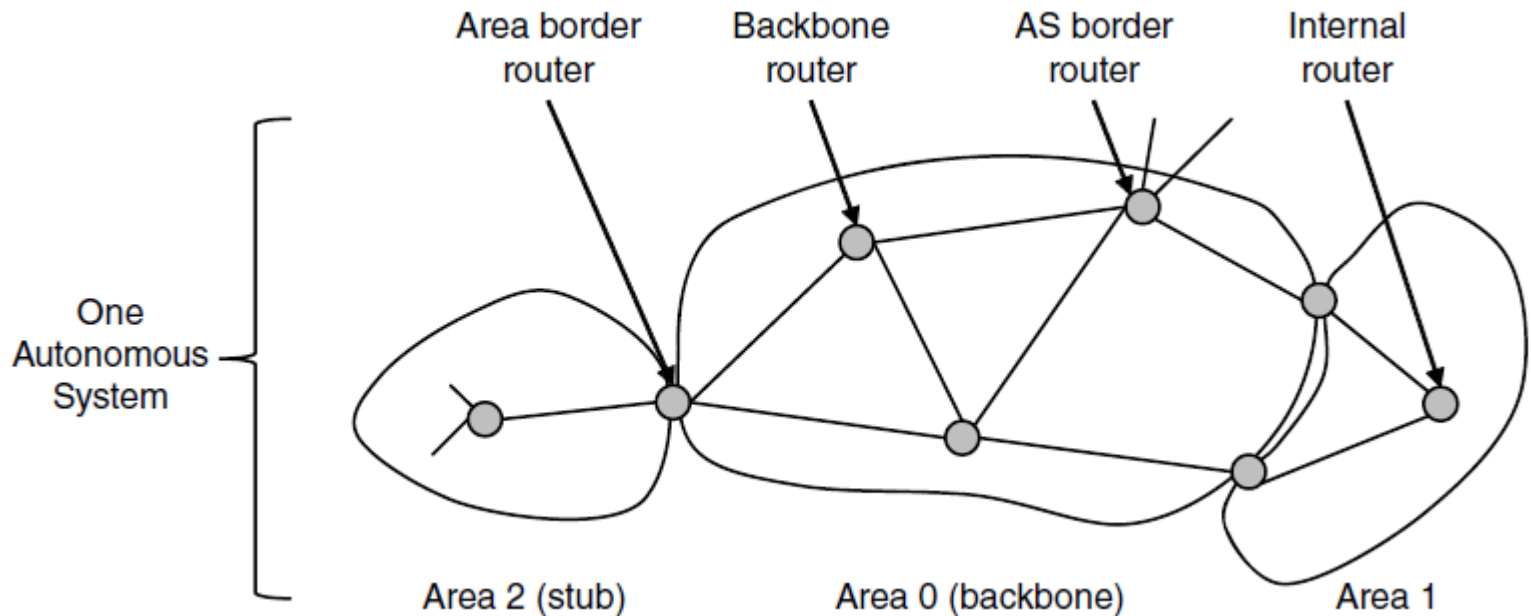
An autonomous system

# OSPF—An Interior Gateway Routing Protocol (2)



A graph representation of the previous slide.

# OSPF—An Interior Gateway Routing Protocol (3)



The relation between ASes, backbones, and areas in OSPF.



# OSPF—An Interior Gateway Routing Protocol (4)

<b>Message type</b>	<b>Description</b>
Hello	Used to discover who the neighbors are
Link state update	Provides the sender's costs to its neighbors
Link state ack	Acknowledges link state update
Database description	Announces which updates the sender has
Link state request	Requests information from the partner

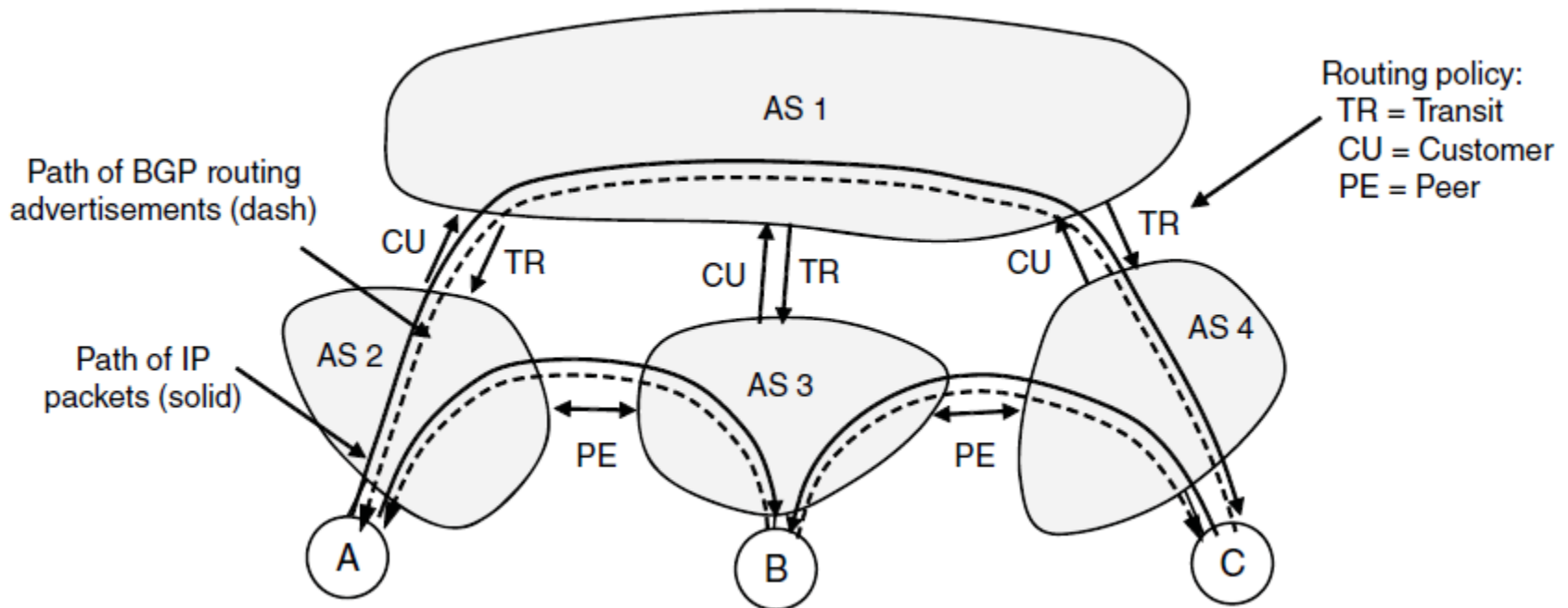
The five types of OSPF messages

# BGP—The Exterior Gateway Routing Protocol (1)

Examples of routing constraints:

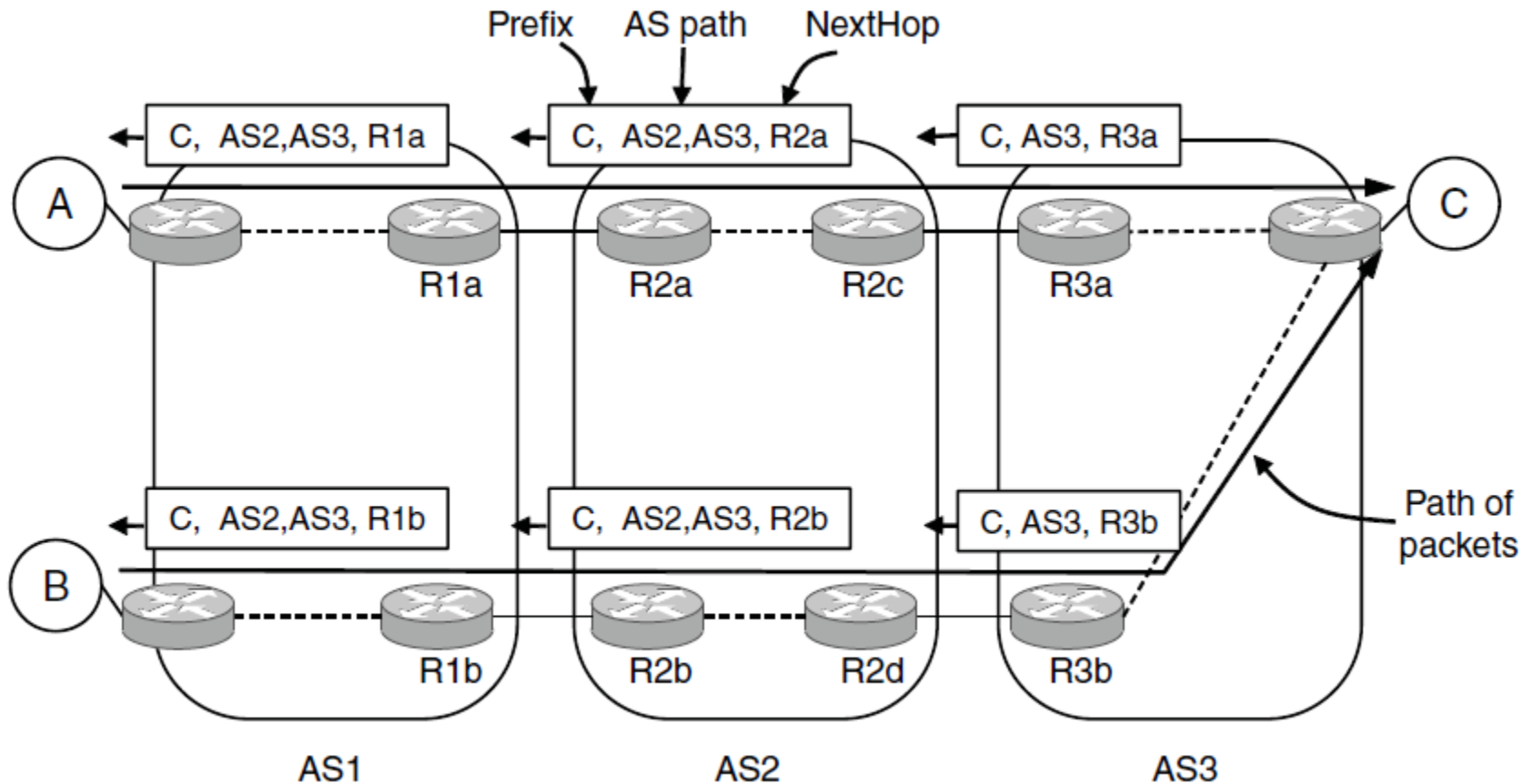
1. No commercial traffic for educat. network
2. Never put Iraq on route starting at Pentagon
3. Choose cheaper network
4. Choose better performing network
5. Don't go from Apple to Google to Apple

# BGP—The Exterior Gateway Routing Protocol (2)



Routing policies between four Autonomous Systems

# BGP—The Exterior Gateway Routing Protocol (3)



Propagation of BGP route advertisements

# Mobile IP

## Goals

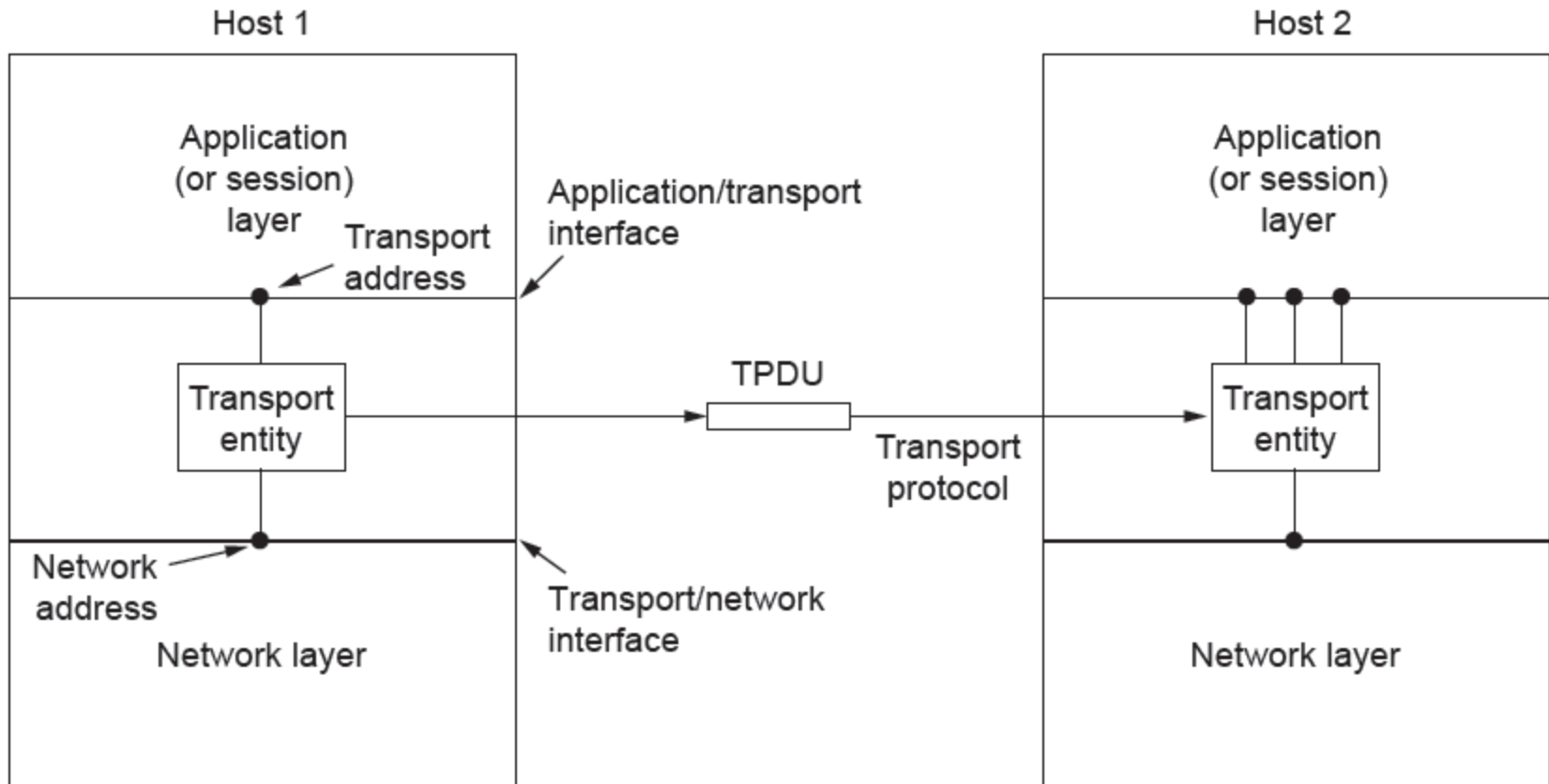
1. Mobile host use home IP address anywhere.
2. No software changes to fixed hosts
3. No changes to router software, tables
4. Packets for mobile hosts – restrict detours
5. No overhead for mobile host at home.

# The Transport Layer

# Transport Service

- Upper Layer Services
- Transport Service Primitives
- Berkeley Sockets
- Example of Socket Programming:  
Internet File Server

# Services Provided to the Upper Layers



The network, transport, and application layers

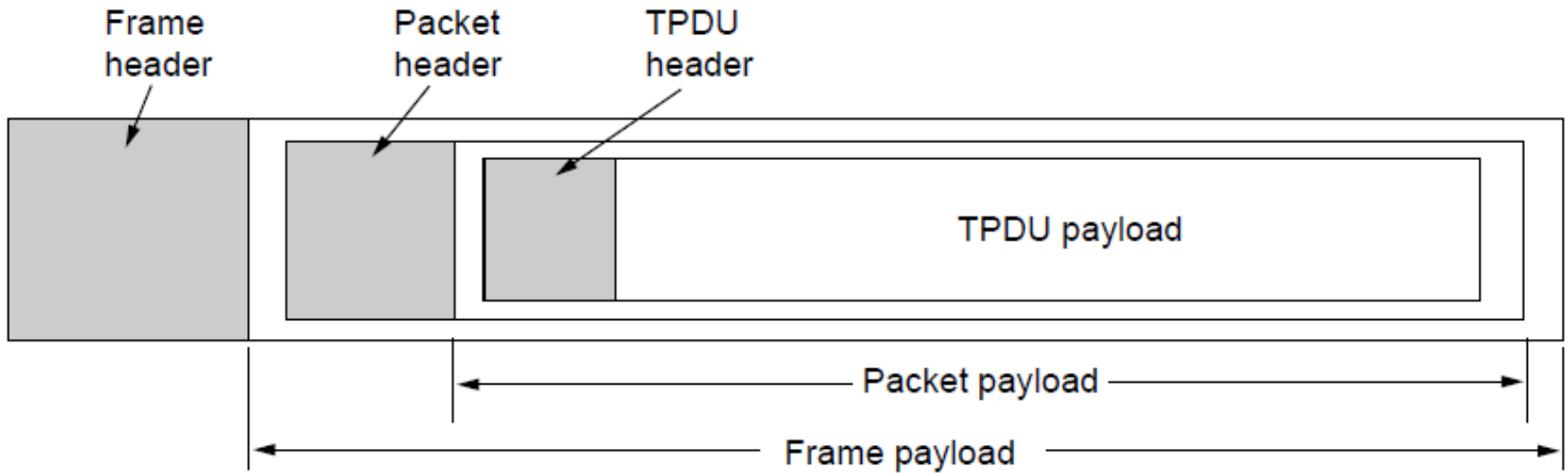


# Transport Service Primitives (1)

<b>Primitive</b>	<b>Packet sent</b>	<b>Meaning</b>
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

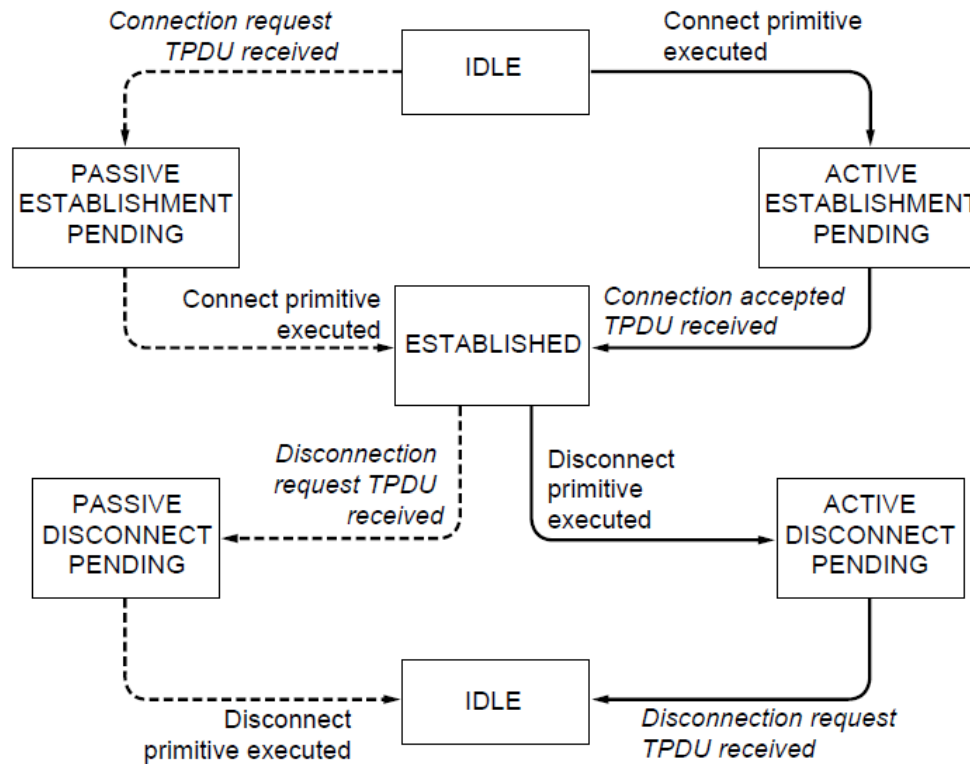
The primitives for a simple transport service

# Transport Service Primitives (2)



Nesting of TPDU, packets, and frames.

# Berkley Sockets (1)



A state diagram for a simple connection management scheme. Transitions labeled in italics are caused by packet arrivals. The solid lines show the client's state sequence. The

# Berkeley Sockets (2)

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

The socket primitives for TCP

# Example of Socket Programming: An Internet File Server (1)

```
/* This page contains a client program that can request a file from the server program
 * on the next page. The server responds by sending the whole file.
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345          /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096            /* block transfer size */

int main(int argc, char **argv)
{
    int c, s, bytes;
    char buf[BUF_SIZE];           /* buffer for incoming file */
    struct hostent *h;            /* info about server */
    struct sockaddr_in channel;   /* holds IP address */

    . . .
```

Client code using sockets

# Example of Socket Programming: An Internet File Server (2)

...

```
if (argc != 3) fatal("Usage: client server-name file-name");
h = gethostbyname(argv[1]);           /* look up host's IP address */
if (!h) fatal("gethostbyname failed");

s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s < 0) fatal("socket");
memset(&channel, 0, sizeof(channel));
channel.sin_family= AF_INET;
memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
channel.sin_port= htons(SERVER_PORT);

c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
if (c < 0) fatal("connect failed");
```

...

Client code using sockets

# Example of Socket Programming: An Internet File Server (3)

...

```
c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
if (c < 0) fatal("connect failed");

/* Connection is now established. Send file name including 0 byte at end. */
write(s, argv[2], strlen(argv[2])+1);

/* Go get the file and write it to standard output. */
while (1) {
    bytes = read(s, buf, BUF_SIZE);           /* read from socket */
    if (bytes <= 0) exit(0);                 /* check for end of file */
    write(1, buf, bytes);                     /* write to standard output */
}
}

fatal(char *string)
{
    printf("%s\n", string);
    exit(1);
}
```

Client code using sockets

# Example of Socket Programming: An Internet File Server (4)

```
#include <sys/types.h>                /* This is the server code */
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345              /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096                 /* block transfer size */
#define QUEUE_SIZE 10

int main(int argc, char *argv[])
{
    int s, b, l, fd, sa, bytes, on = 1;
    char buf[BUF_SIZE];                /* buffer for outgoing file */
    struct sockaddr_in channel;        /* holds IP address */
```

...

Server code



# Example of Socket Programming: An Internet File Server (5)

```
...  
/* Build address structure to bind to socket. */  
memset(&channel, 0, sizeof(channel));    /* zero channel */  
channel.sin_family = AF_INET;  
channel.sin_addr.s_addr = htonl(INADDR_ANY);  
channel.sin_port = htons(SERVER_PORT);  
  
/* Passive open. Wait for connection. */  
s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); /* create socket */  
if (s < 0) fatal("socket failed");  
setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));  
  
b = bind(s, (struct sockaddr *) &channel, sizeof(channel));  
if (b < 0) fatal("bind failed");  
  
l = listen(s, QUEUE_SIZE);                /* specify queue size */  
if (l < 0) fatal("listen failed");  
...
```

Server code

# Example of Socket Programming: An Internet File Server (6)

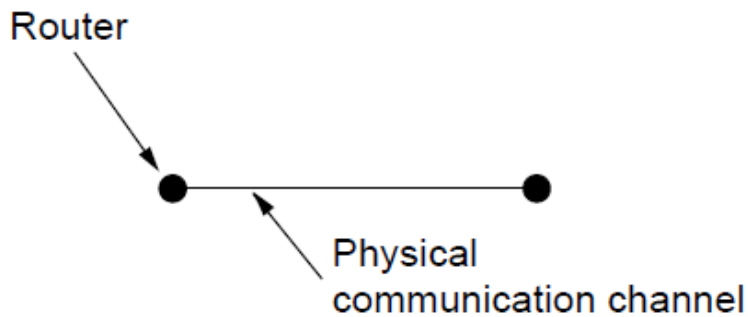
```
...  
/* Socket is now set up and bound. Wait for connection and process it. */  
while (1) {  
    sa = accept(s, 0, 0);           /* block for connection request */  
    if (sa < 0) fatal("accept failed");  
  
    read(sa, buf, BUF_SIZE);      /* read file name from socket */  
  
    /* Get and return the file. */  
    fd = open(buf, O_RDONLY);     /* open the file to be sent back */  
    if (fd < 0) fatal("open failed");  
  
    while (1) {  
        bytes = read(fd, buf, BUF_SIZE); /* read from file */  
        if (bytes <= 0) break;         /* check for end of file */  
        write(sa, buf, bytes);        /* write bytes to socket */  
    }  
    close(fd);                      /* close file */  
    close(sa);                      /* close connection */  
}  
}
```

Server code

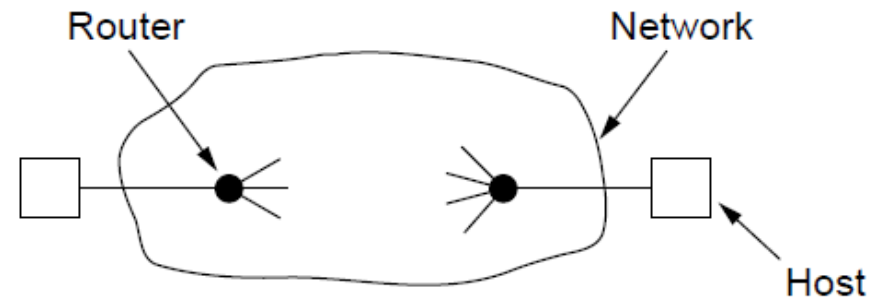
# Elements of Transport Protocols (1)

- Addressing
- Connection establishment
- Connection release
- Error control and flow control
- Multiplexing
- Crash recovery

# Elements of Transport Protocols (2)



(a)

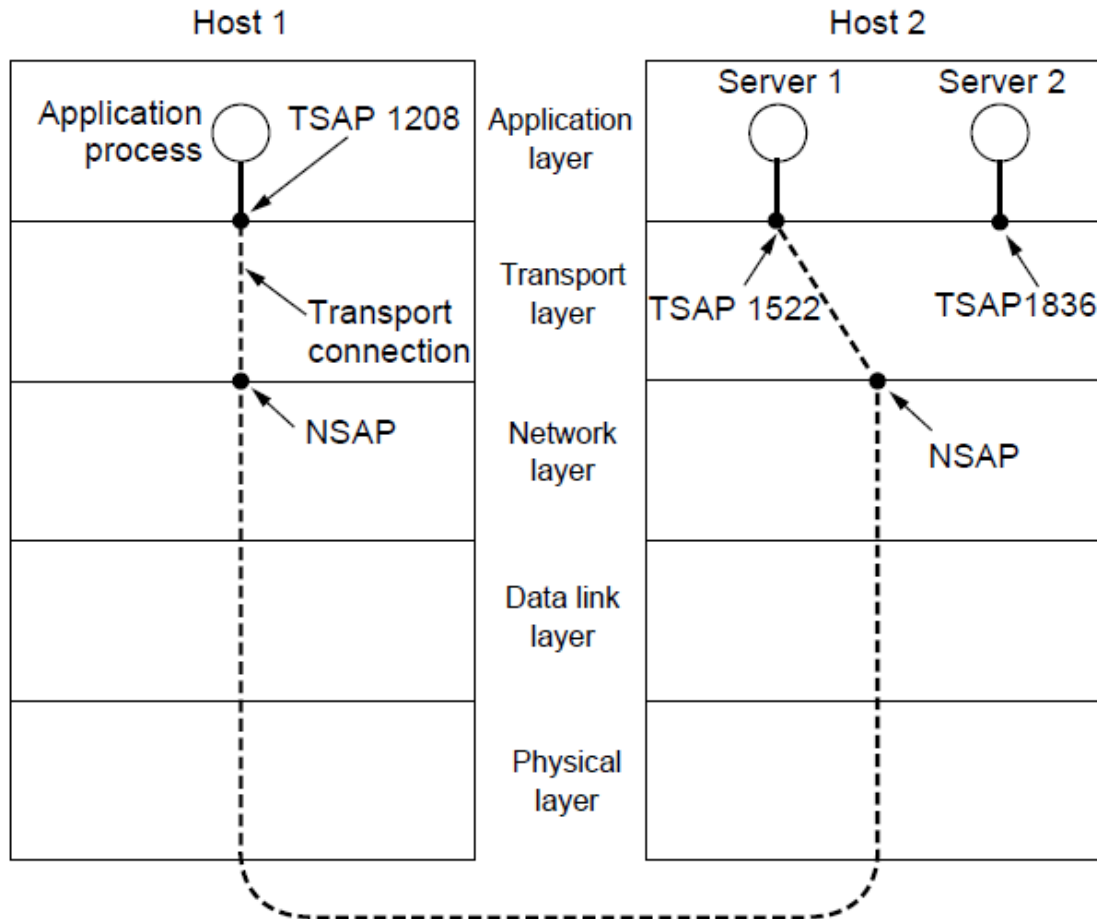


(b)

(a) Environment of the data link layer.

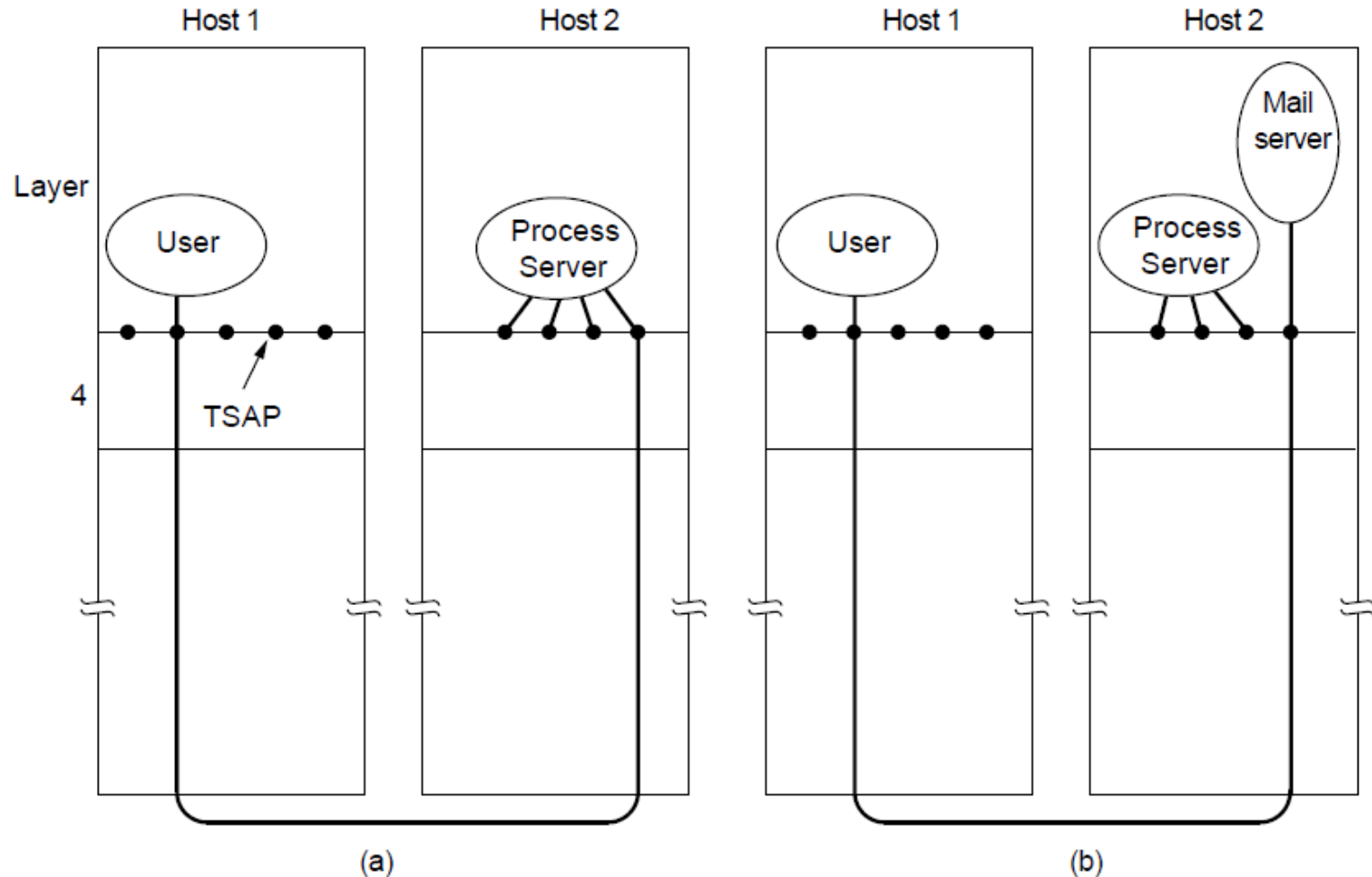
(b) Environment of the transport layer.

# Addressing (1)



rt

# Addressing (2)



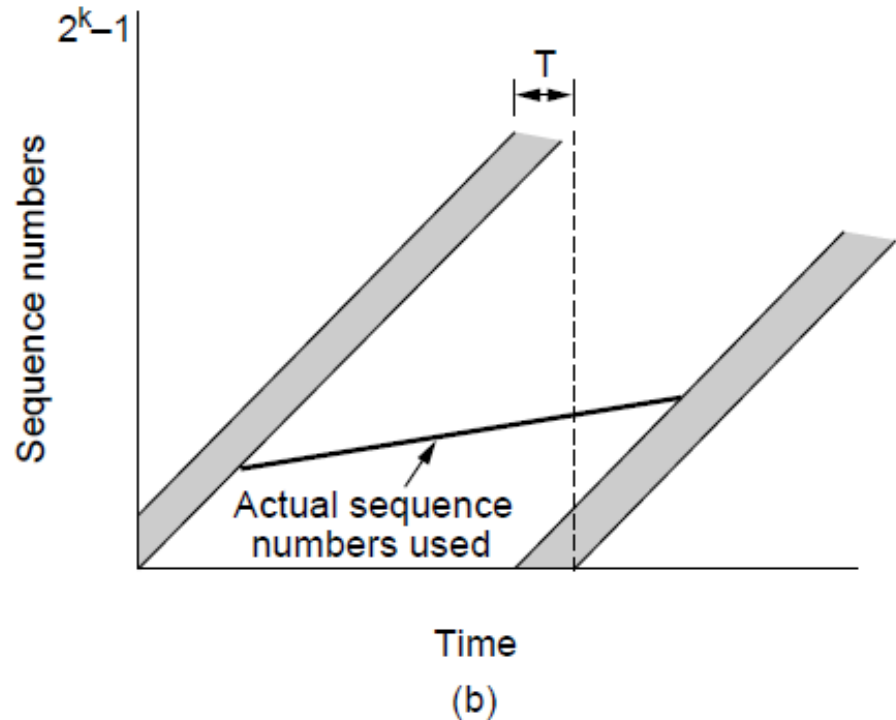
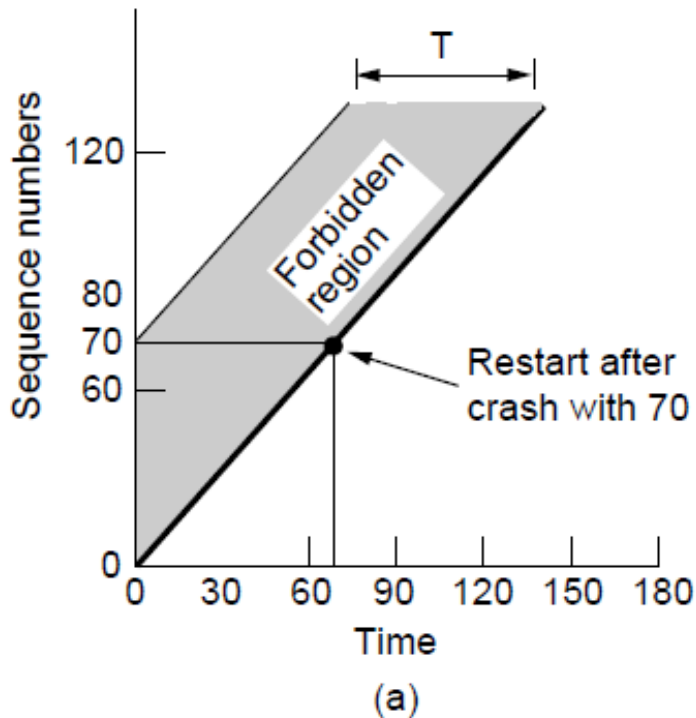
How a user process in host 1 establishes a connection

# Connection Establishment (1)

Techniques for restricting packet lifetime

- Restricted network design.
- Putting a hop counter in each packet.
- Timestamping each packet.

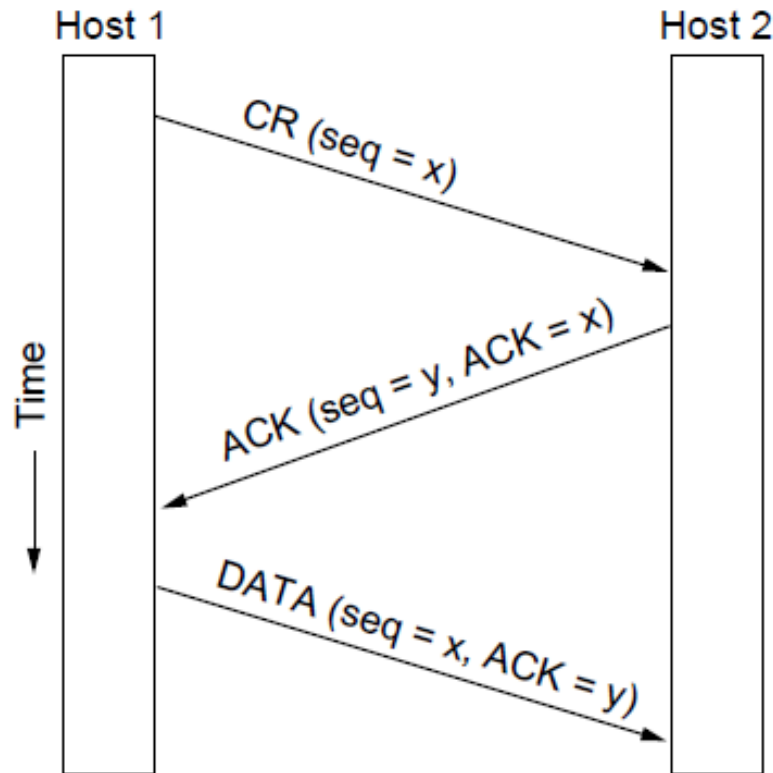
# Connection Establishment (2)



- (a) TPDUs may not enter the forbidden region.
- (b) The resynchronization problem.

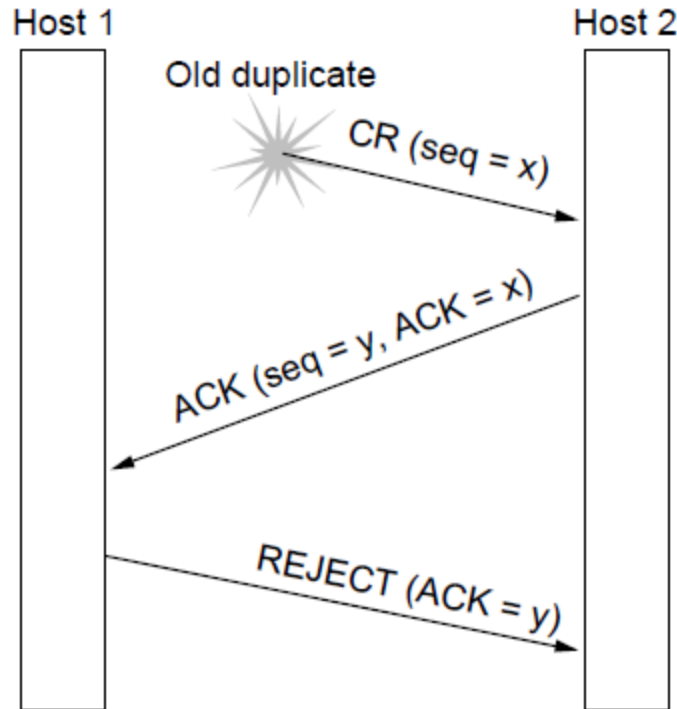


# Connection Establishment (3)



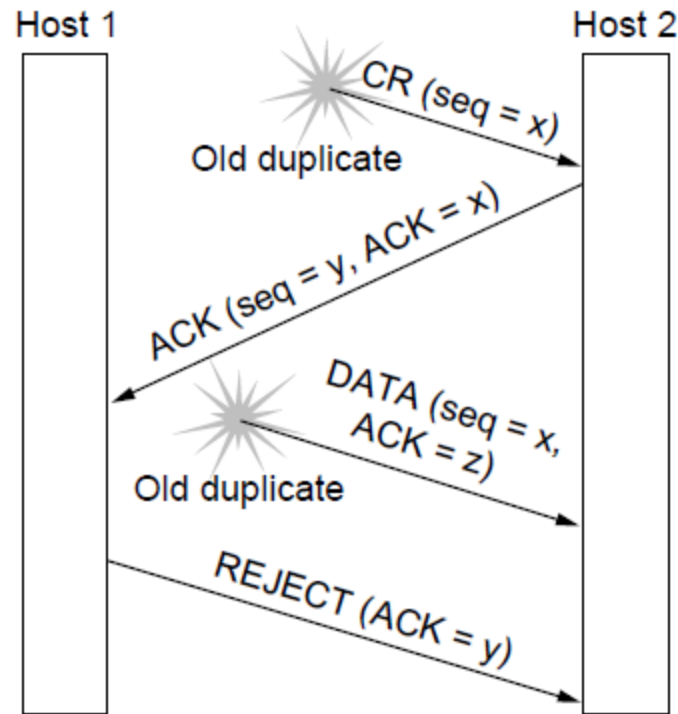
Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION

# Connection Establishment (4)



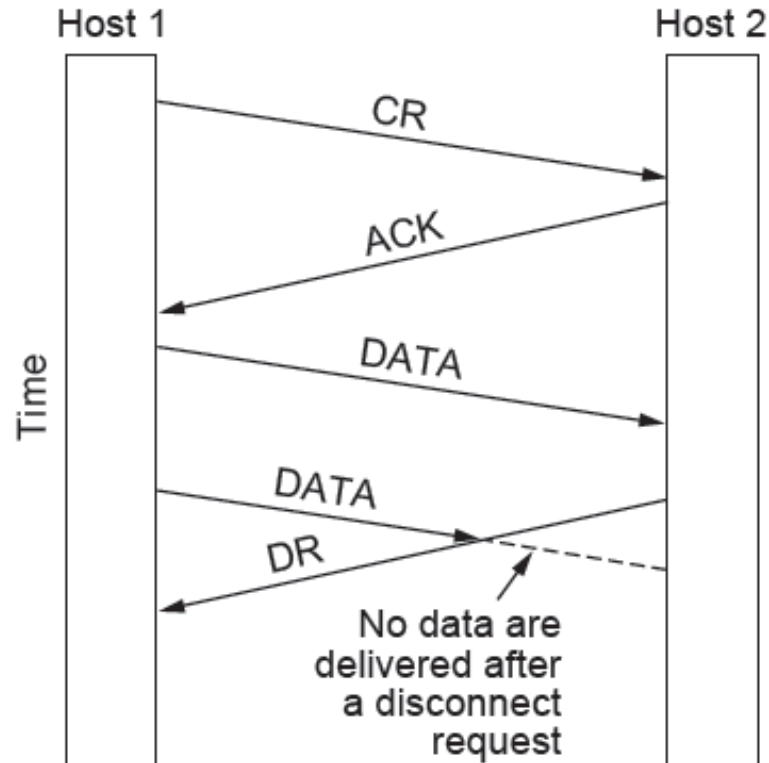
Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST. Old duplicate CONNECTION REQUEST appearing out of nowhere.

# Connection Establishment (5)



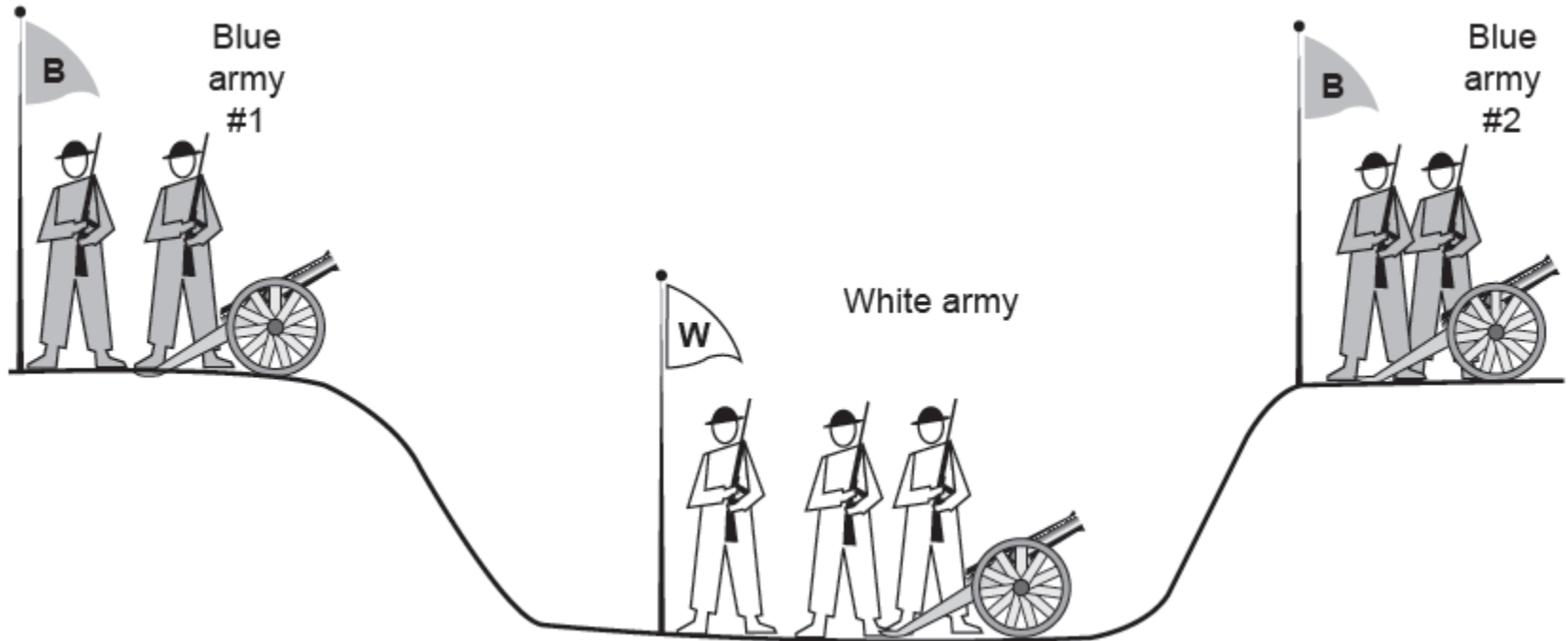
Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes

# Connection Release (1)

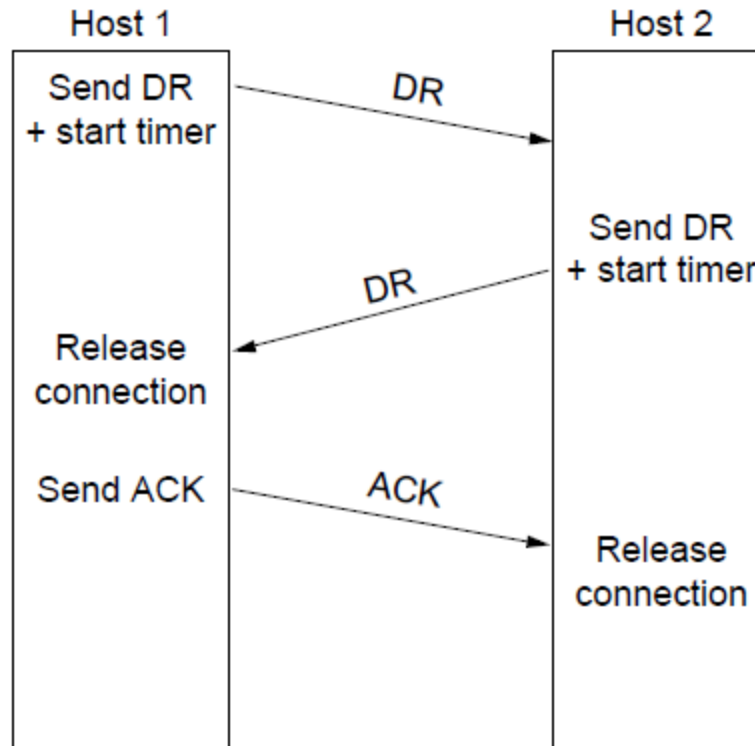


Abrupt disconnection with loss of data

# Connection Release (2)



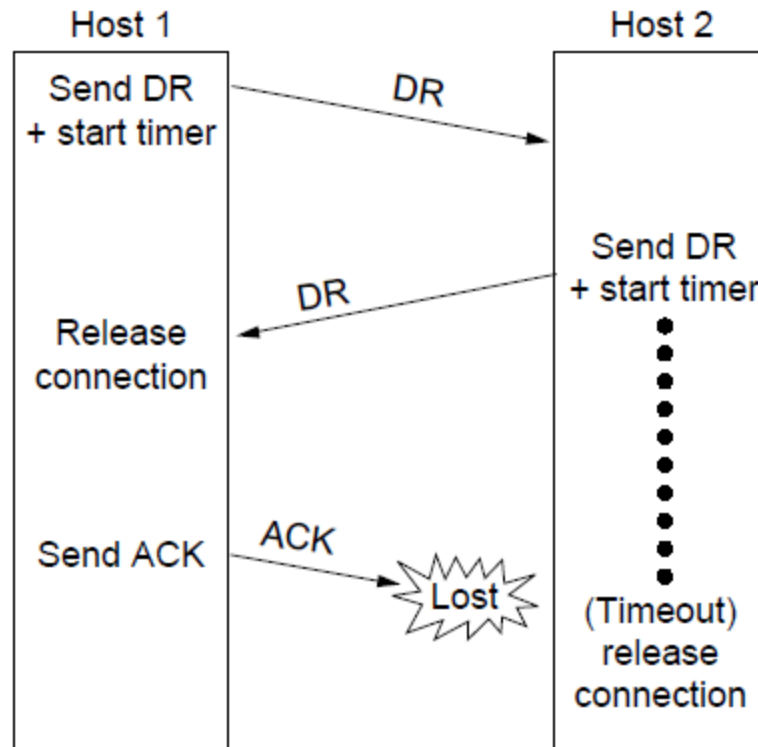
# Connection Release (3)



Four protocol scenarios for releasing a connection.

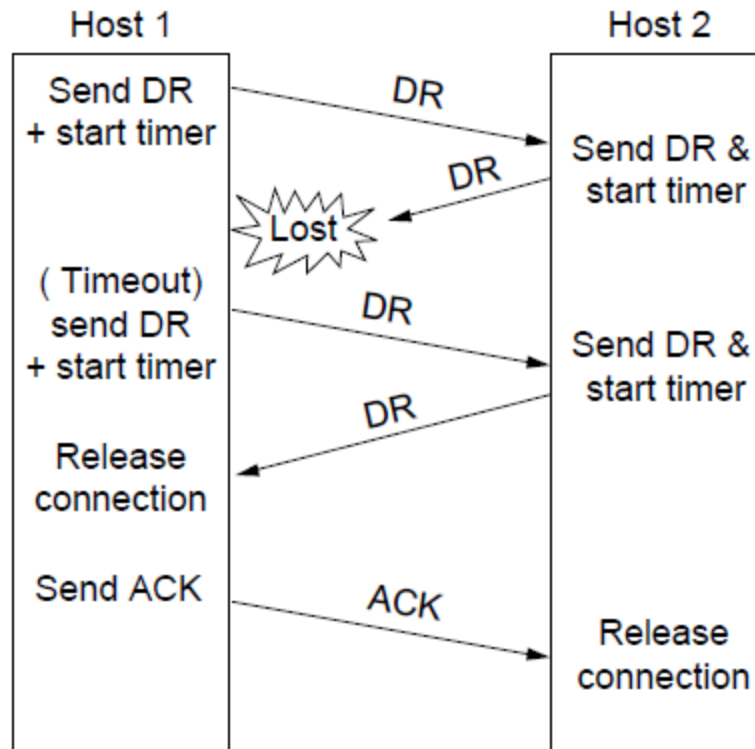
(a) Normal case of three-way handshake

# Connection Release (4)



Four protocol scenarios for releasing a connection.  
**(b)** Final ACK lost.

# Connection Release (5)

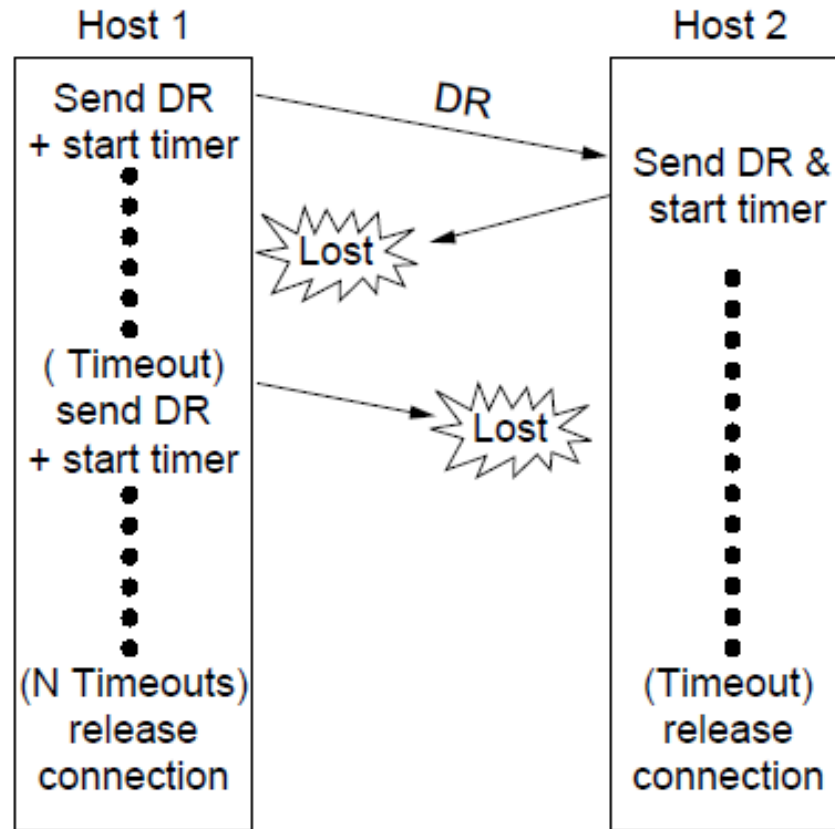


Four protocol scenarios for releasing a connection.

(c) Response lost

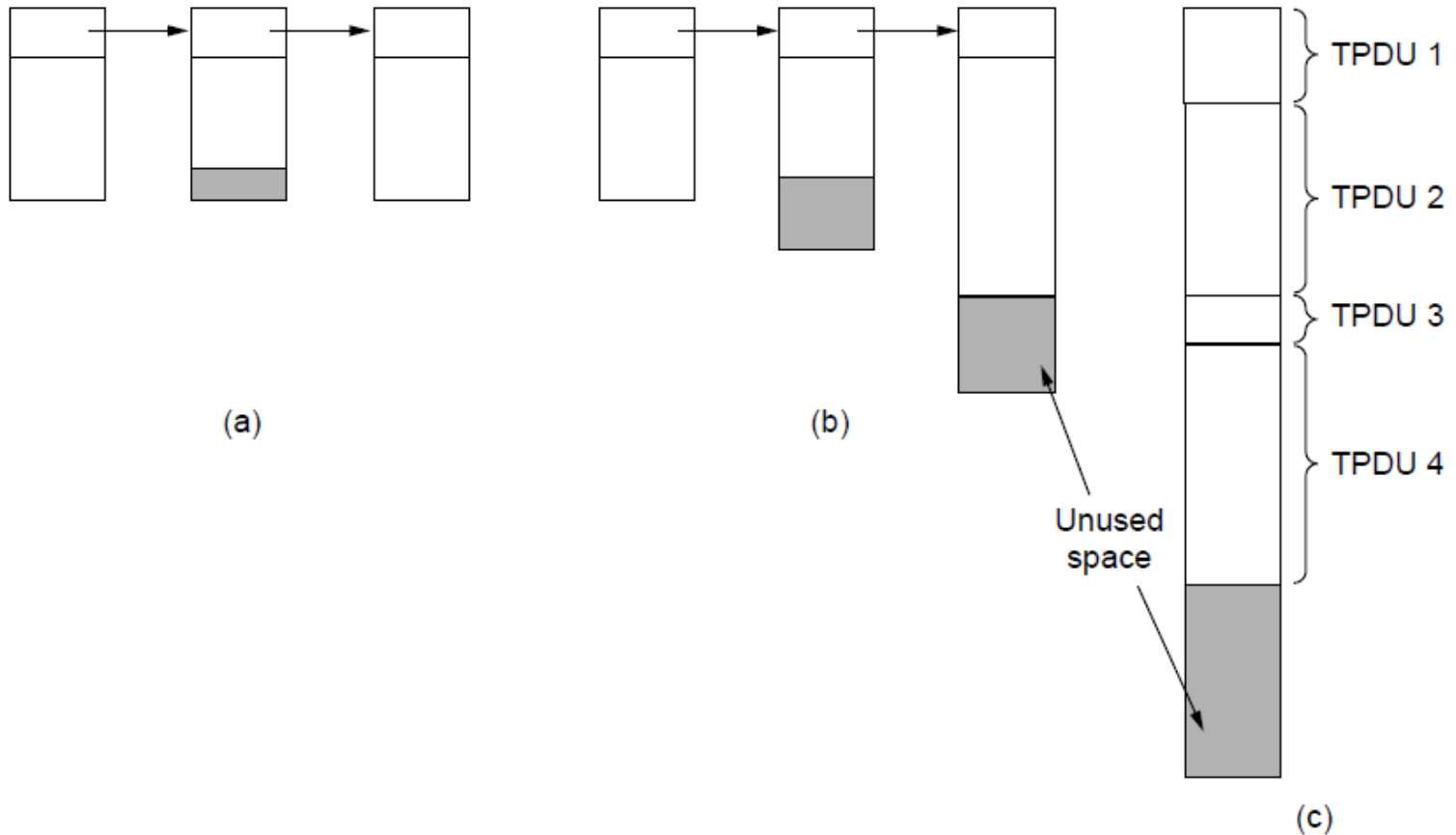


# Connection Release (6)



Four protocol scenarios for releasing a connection.  
(d) Response lost and subsequent DRs lost.

# Error Control and Flow Control (1)



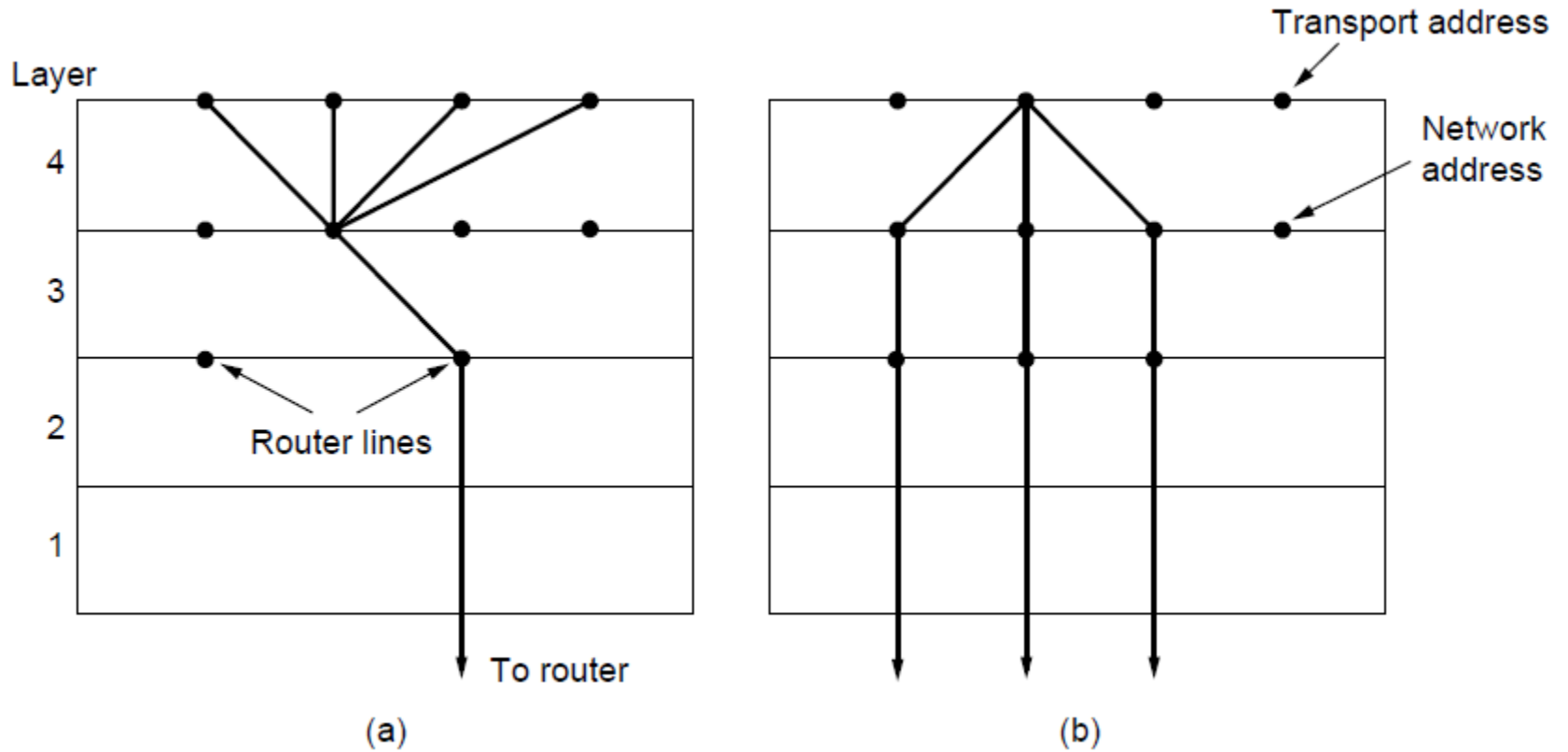
**(a)** Chained fixed-size buffers. **(b)** Chained variable-sized buffers. **(c)** One large circular buffer per

## Error Control and Flow Control (2)

<u>A</u>	<u>Message</u>	<u>B</u>	<u>Comments</u>	
1	→	< request 8 buffers>	→	A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	A has 2 buffers left now
5	→	<seq = 2, data = m2>	...	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	A may now send 5
12	←	<ack = 4, buf = 2>	←	B found a new buffer somewhere
13	→	<seq = 5, data = m5>	→	A has 1 buffer left
14	→	<seq = 6, data = m6>	→	A is now blocked again
15	←	<ack = 6, buf = 0>	←	A is still blocked
16	...	<ack = 6, buf = 4>	←	Potential deadlock

Dynamic buffer allocation. The arrows show the direction of transmission. An ellipsis (...)

# Multiplexing



**(a)** Multiplexing. **(b)** Inverse multiplexing.

# Crash Recovery

Strategy used by receiving host

← First ACK, then write      ← First write, then ACK →

Strategy used by sending host

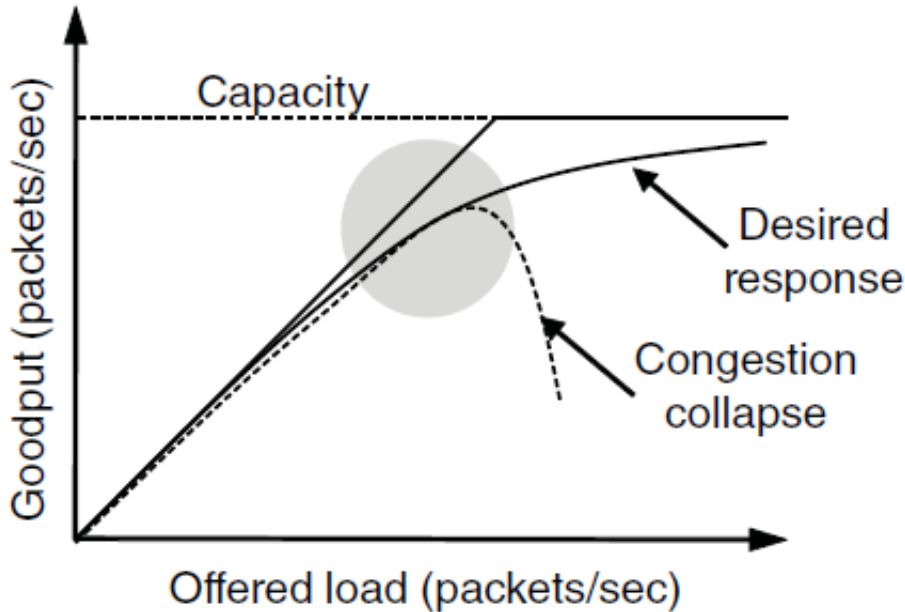
	First ACK, then write			First write, then ACK		
	AC(W)	AWC	C(AW)	C(WA)	W AC	WC(A)
Always retransmit	OK	DUP	OK	OK	DUP	DUP
Never retransmit	LOST	OK	LOST	LOST	OK	OK
Retransmit in S0	OK	DUP	LOST	LOST	DUP	OK
Retransmit in S1	LOST	OK	OK	OK	OK	DUP

- OK = Protocol functions correctly
- DUP = Protocol generates a duplicate message
- LOST = Protocol loses a message

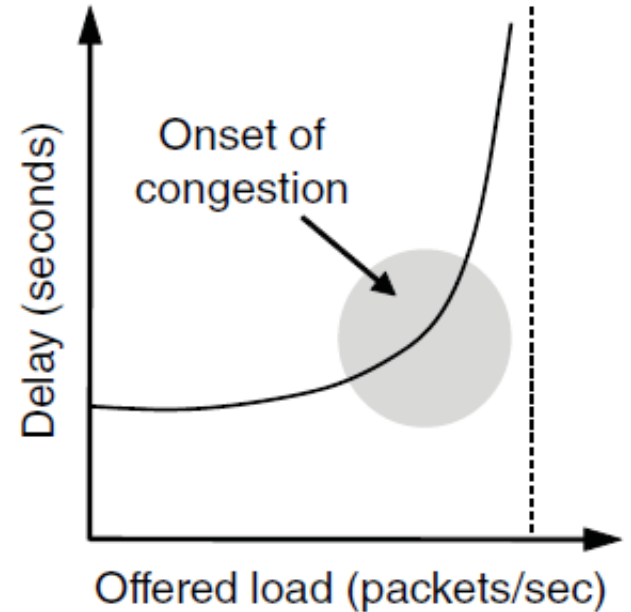
# Congestion Control

- Desirable bandwidth allocation
- Regulating the sending rate

# Desirable Bandwidth Allocation (1)



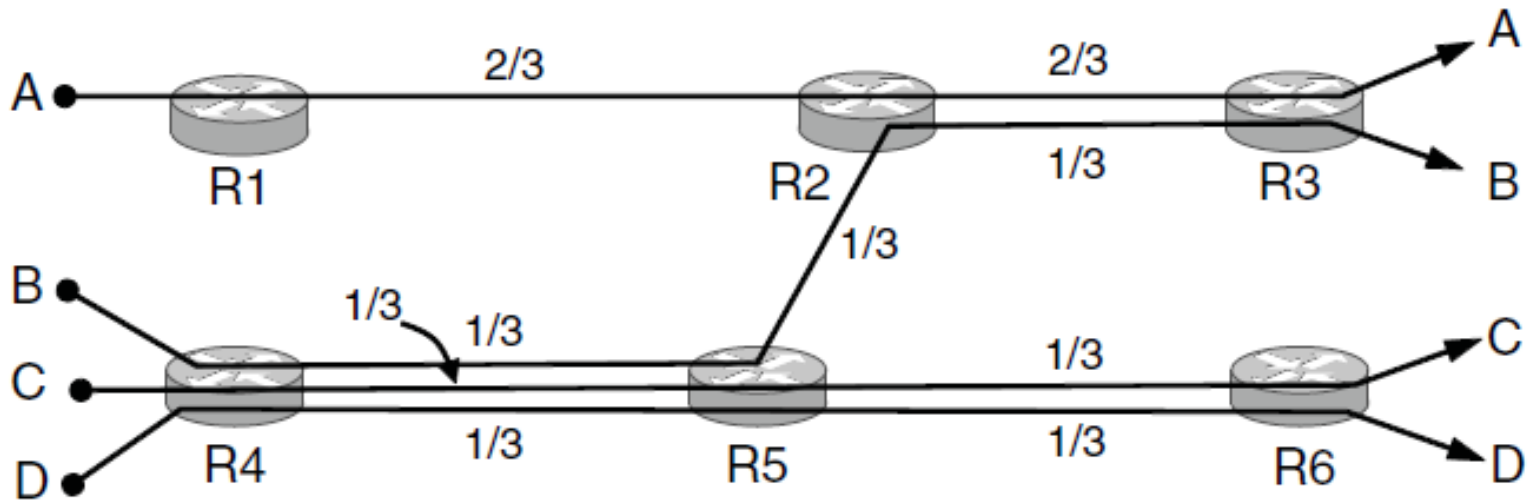
(a)



(b)

(a) Goodput and (b) delay as a function of offered load

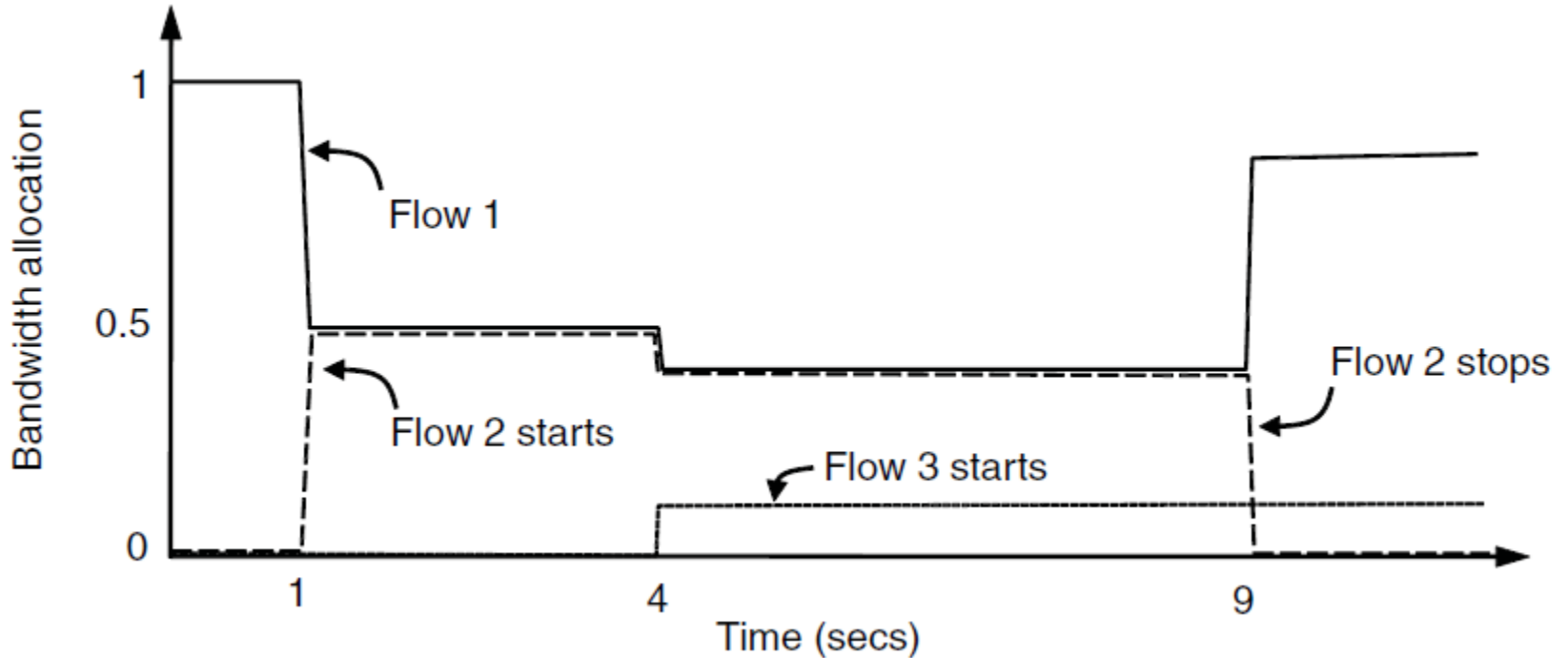
# Desirable Bandwidth Allocation (2)



Max-min bandwidth allocation for four flows

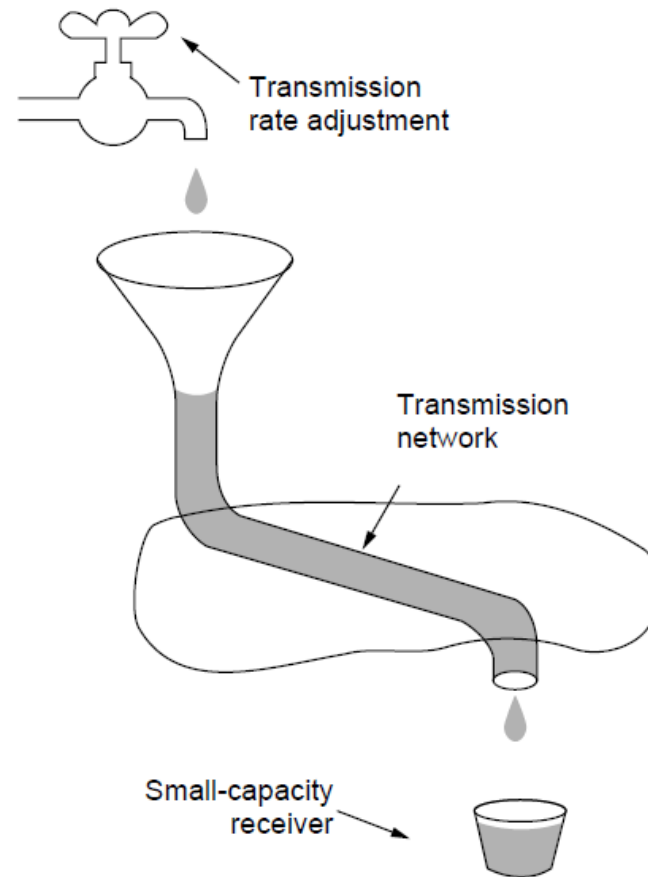


# Desirable Bandwidth Allocation (3)



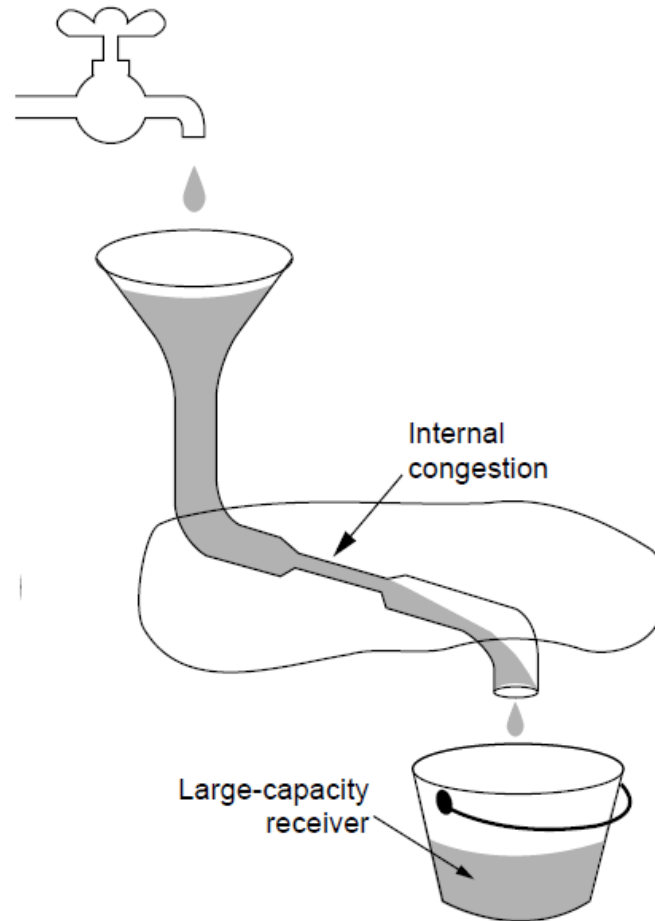
Changing bandwidth allocation over time

# Regulating the Sending Rate (1)



A fast network feeding a low-capacity receiver

# Regulating the Sending Rate (2)



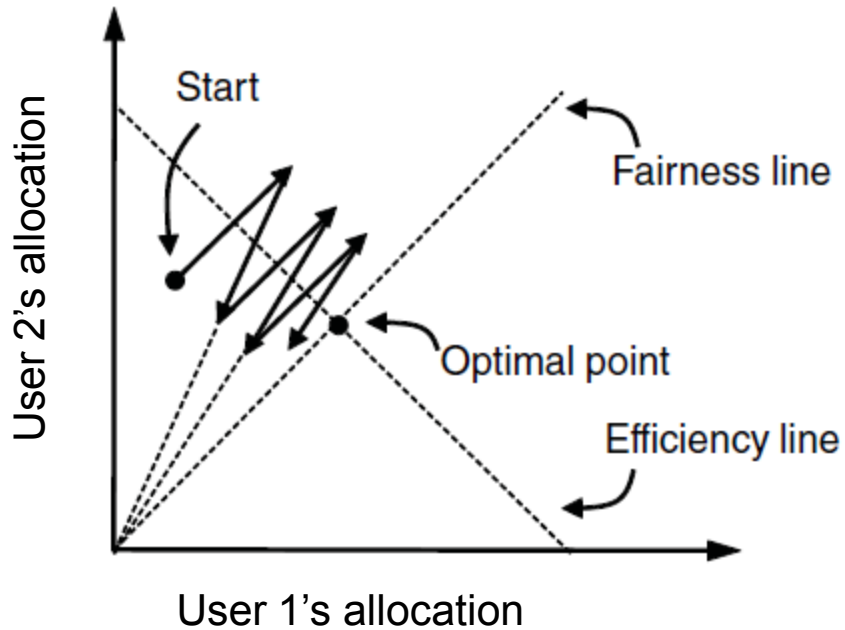
A slow network feeding a high-capacity receiver

# Regulating the Sending Rate (3)



<b>Protocol</b>	<b>Signal</b>	<b>Explicit?</b>	<b>Precise?</b>
XCP	Rate to use	Yes	Yes
TCP with ECN	Congestion warning	Yes	No
FAST TCP	End-to-end delay	No	Yes
CUBIC TCP	Packet loss	No	No
TCP	Packet loss	No	No

Some congestion control protocols

# Regulating the Sending Rate (4)



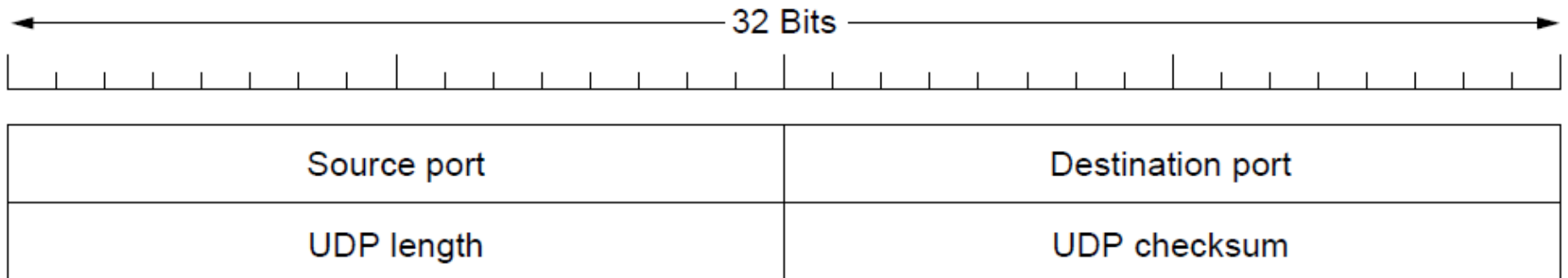
Legend:

-  = Additive increase (up at 45°)
-  = Multiplicative decrease (line points to origin)

# The Internet Transport Protocols: UDP

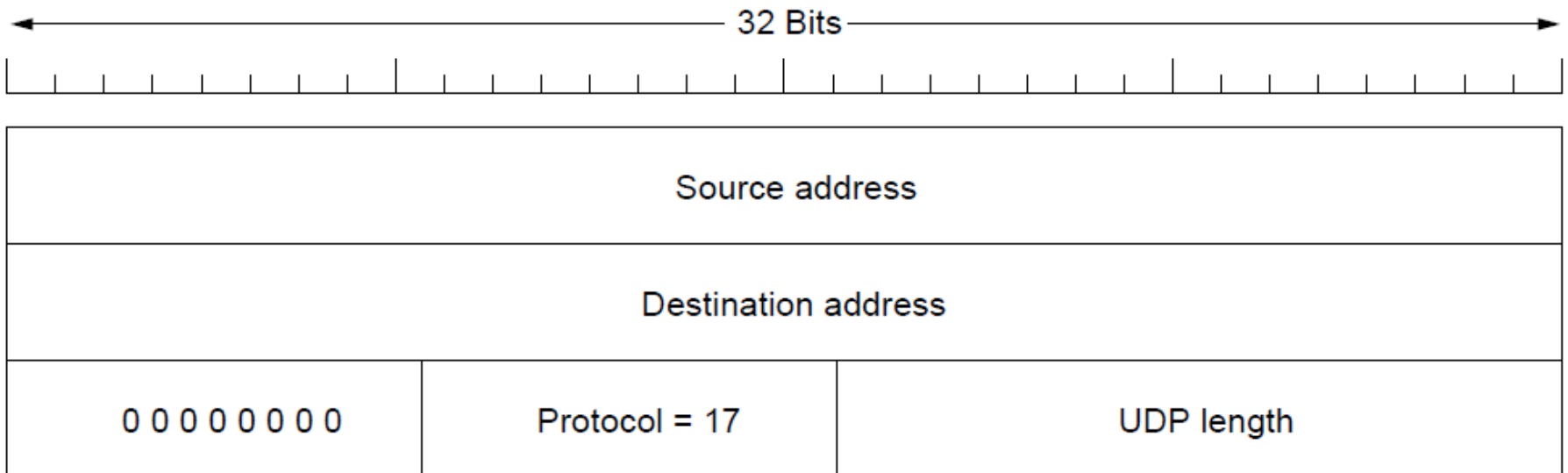
- Introduction to UDP
- Remote Procedure Call
- Real-Time Transport

# Introduction to UDP (1)



The UDP header.

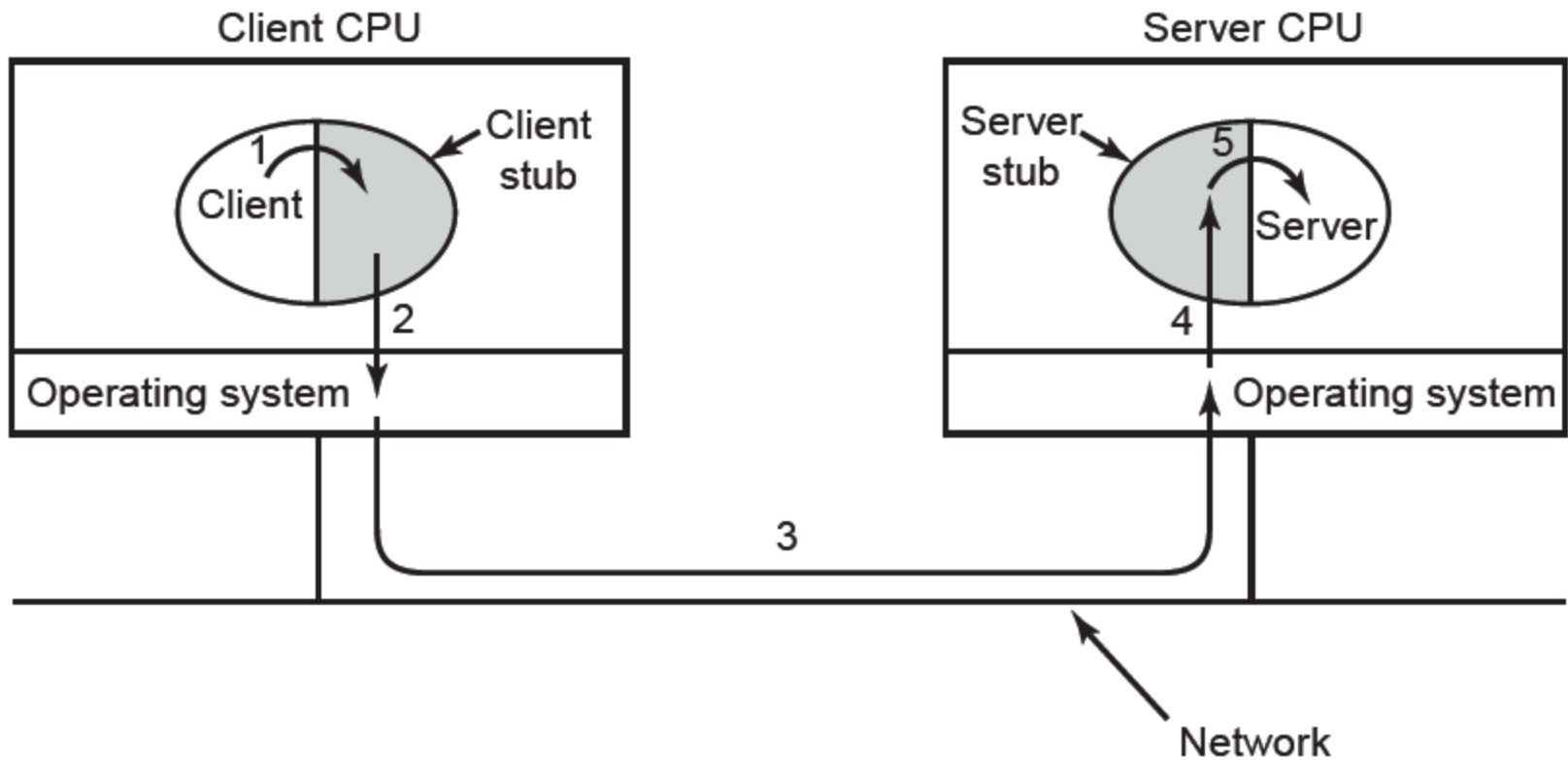
# Introduction to UDP (2)



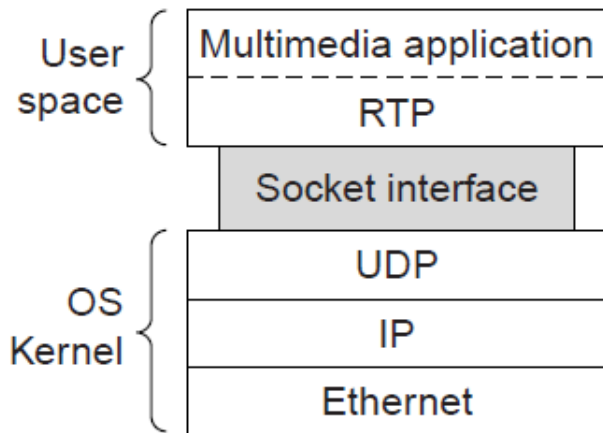
The IPv4 pseudoheader included in the UDP checksum.



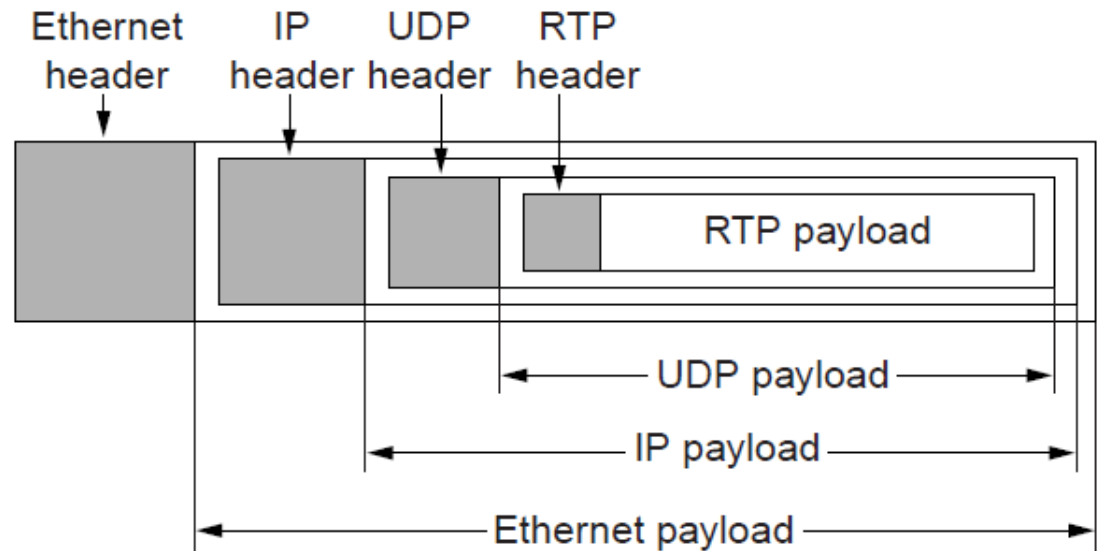
# Remote Procedure Call



# Real-Time Transport (1)

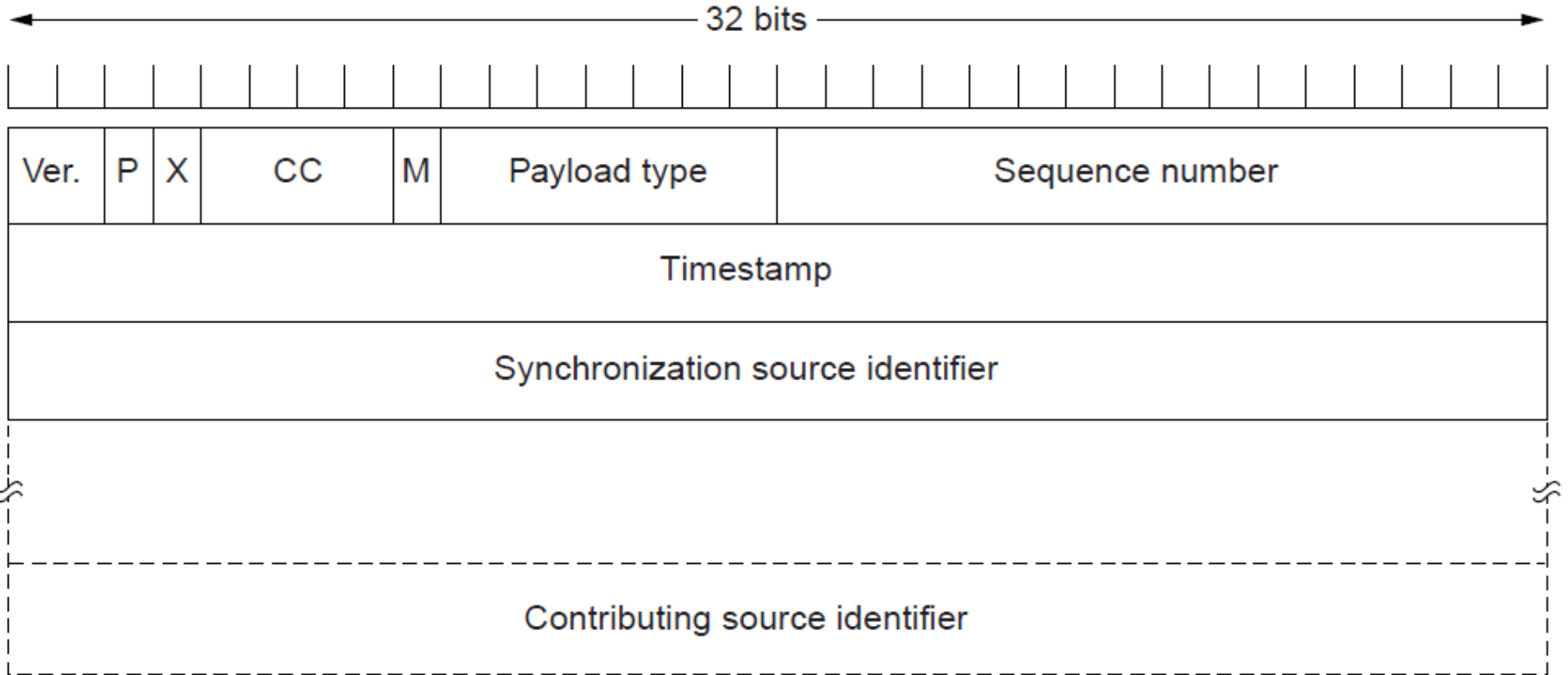


(a)

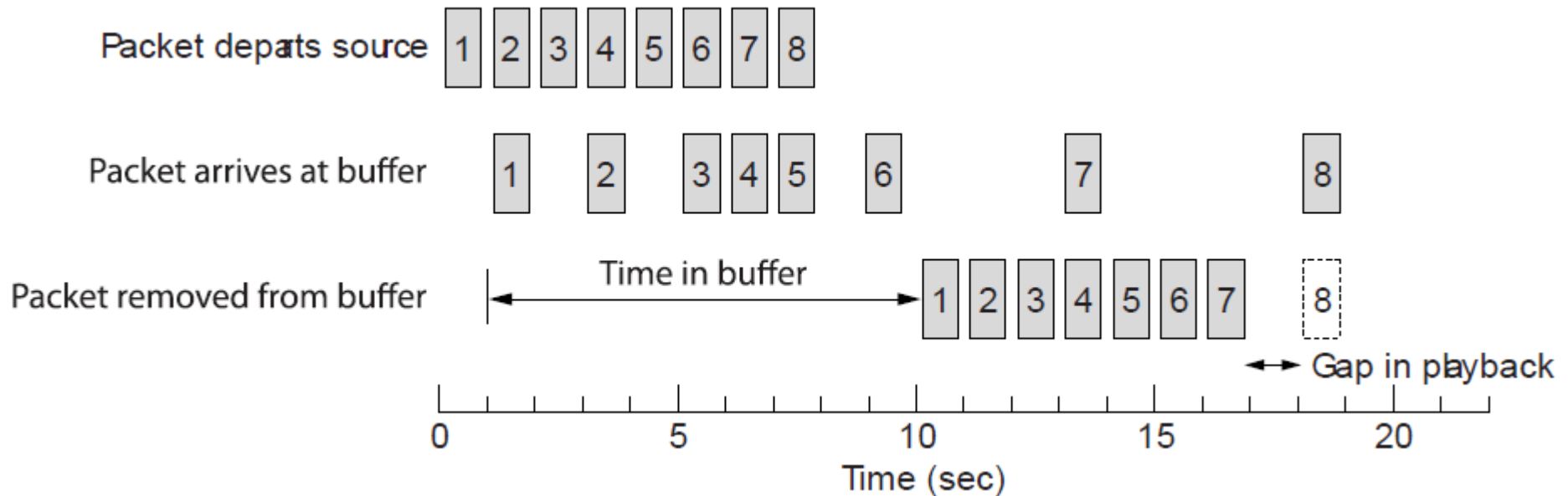


(b)

# Real-Time Transport (2)

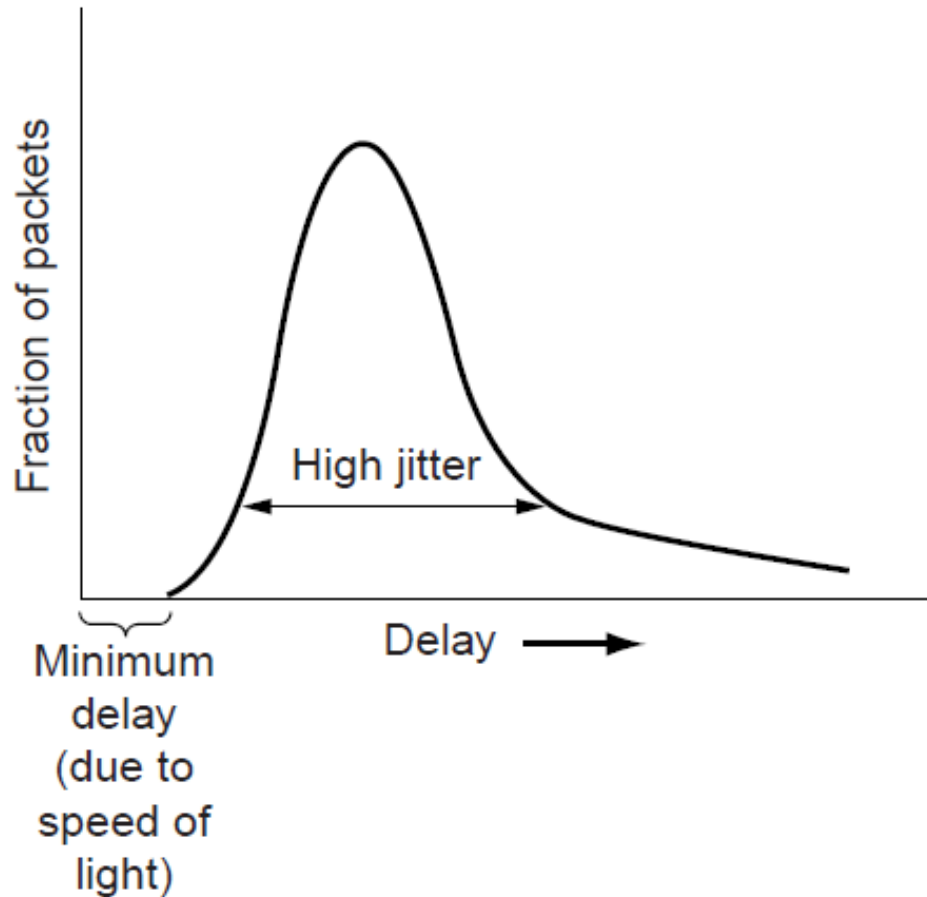


# Real-Time Transport (3)



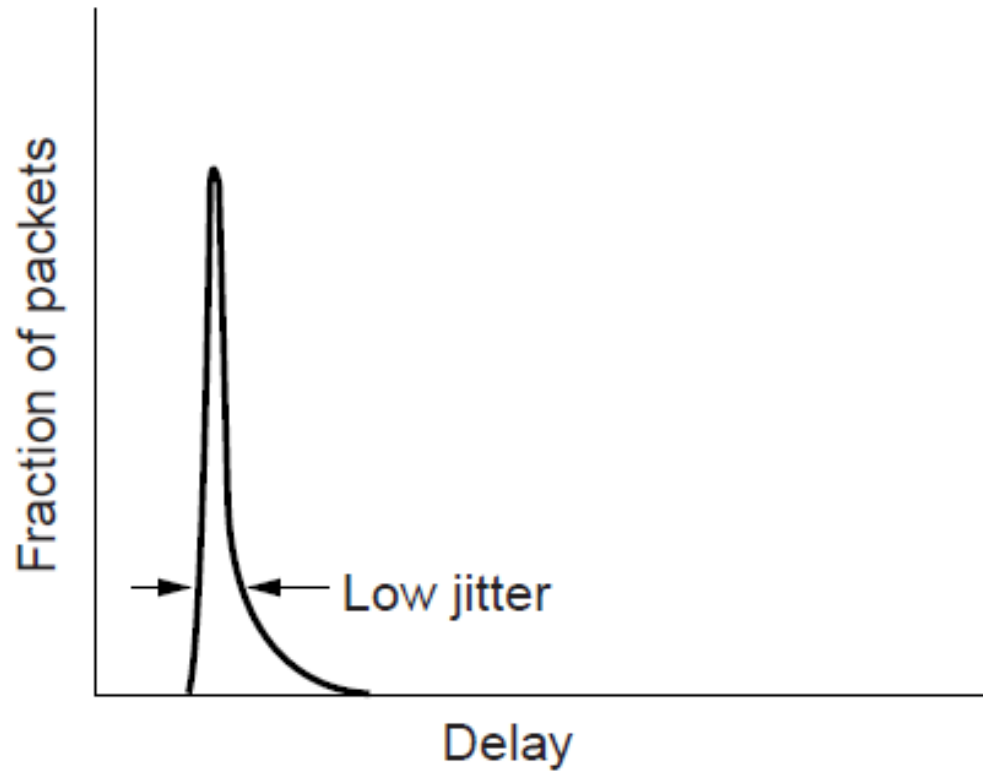
Smoothing the output stream by buffering packets

# Real-Time Transport (3)



High jitter

# Real-Time Transport (4)



Low jitter

# The Internet Transport Protocols: TCP (1)

- Introduction to TCP
- The TCP service model
- The TCP protocol
- The TCP segment header
- TCP connection establishment
- TCP connection release

# The Internet Transport Protocols: TCP (2)

- TCP connection management modeling
- TCP sliding window
- TCP timer management
- TCP congestion control
- TCP futures

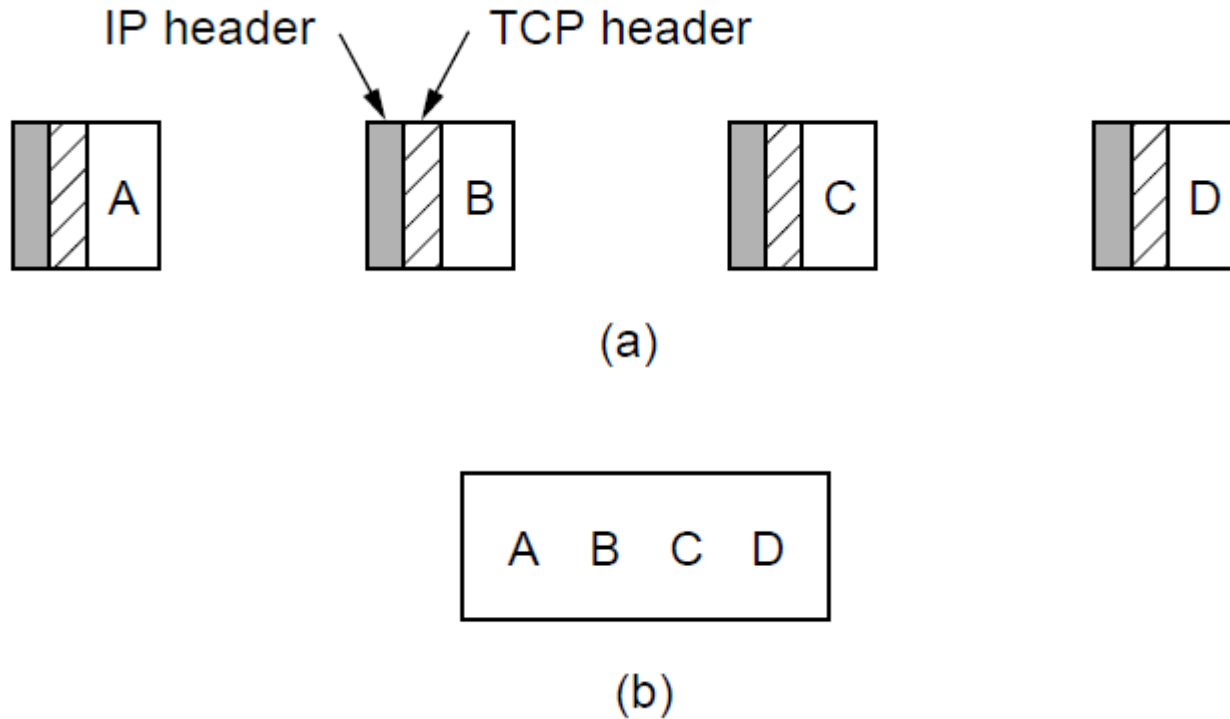


# The TCP Service Model (1)

Port	Protocol	Use
20, 21	FTP	File transfer
22	SSH	Remote login, replacement for Telnet
25	SMTP	Email
80	HTTP	World Wide Web
110	POP-3	Remote email access
143	IMAP	Remote email access
443	HTTPS	Secure Web (HTTP over SSL/TLS)
543	RTSP	Media player control
631	IPP	Printer sharing

Some assigned ports

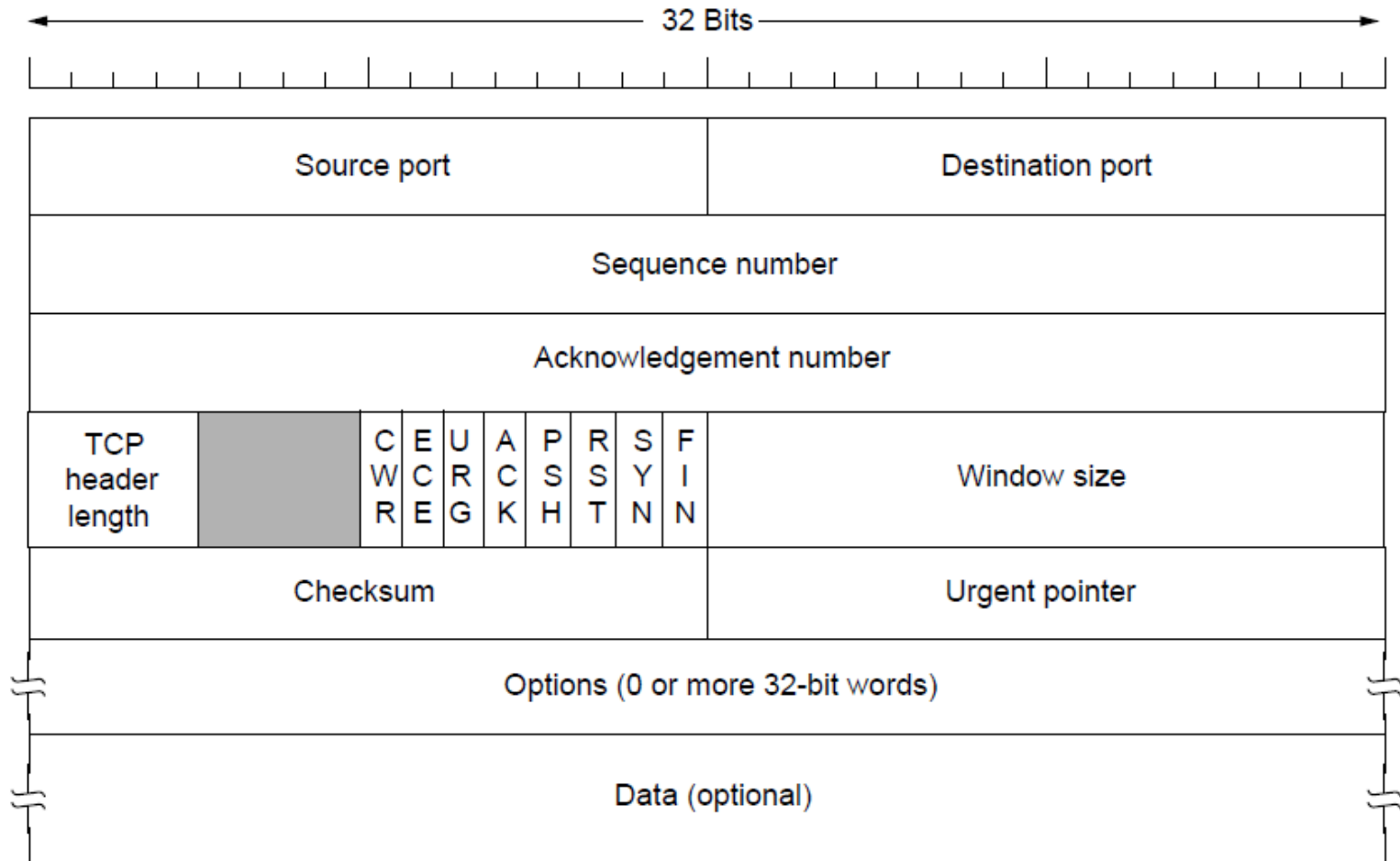
## The TCP Service Model (2)



(a) Four 512-byte segments sent as separate IP diagrams

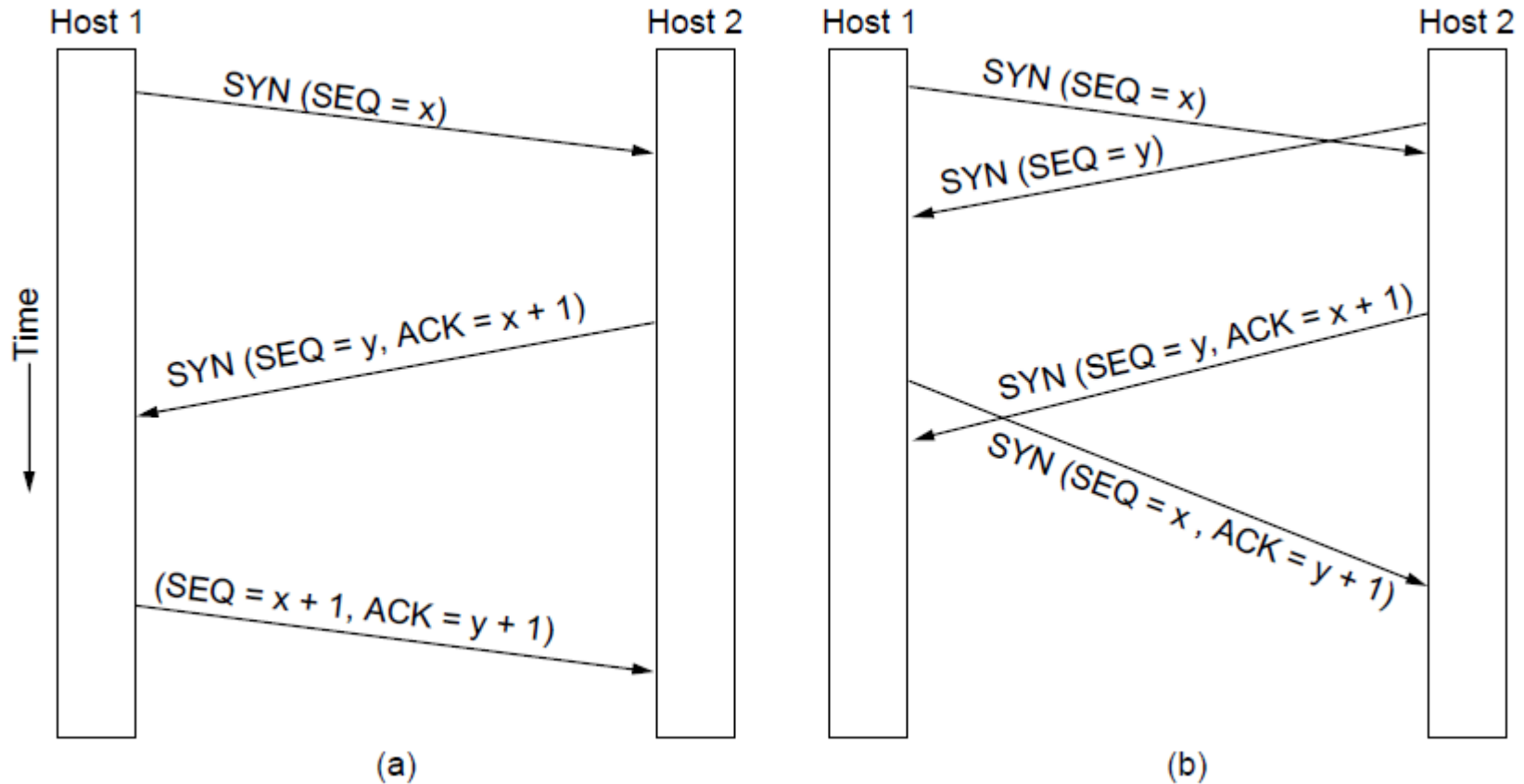
(b) The 2048 bytes of data delivered to the

# The TCP Segment Header



The TCP header.

# TCP Connection Establishment



(a) TCP connection establishment in the normal case.

# TCP Connection Management

## Modeling (1)

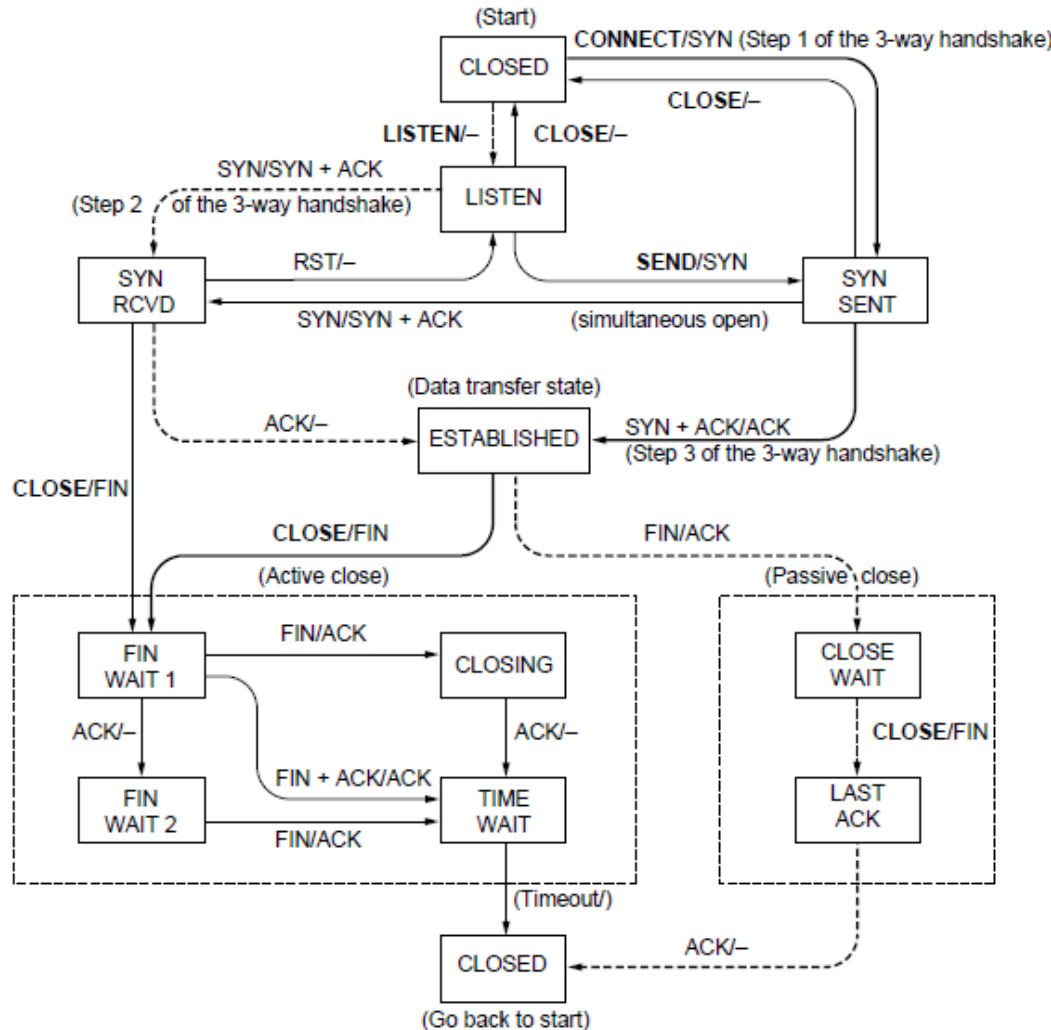
State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIME WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

n

# TCP Connection Management

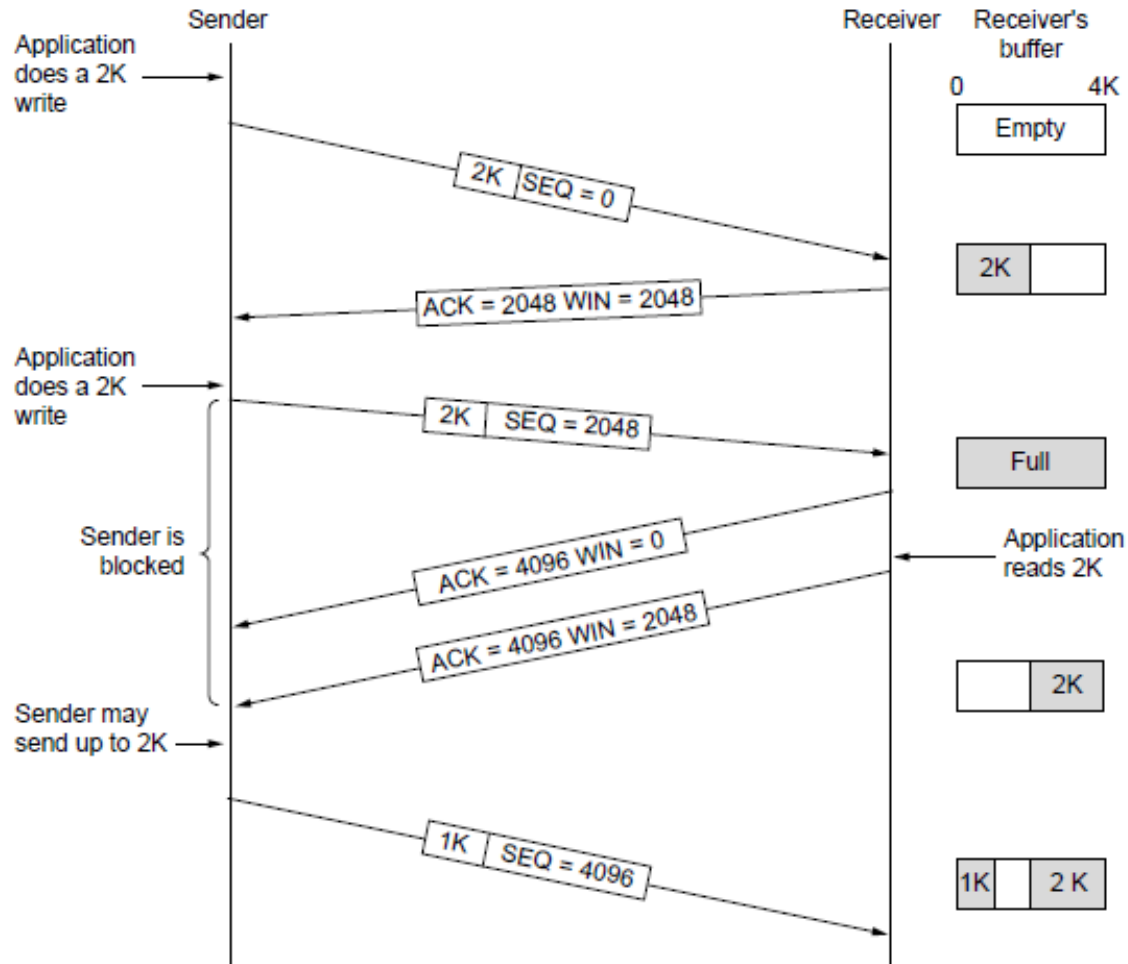
## Modeling (?)

TCP connection management finite state machine.



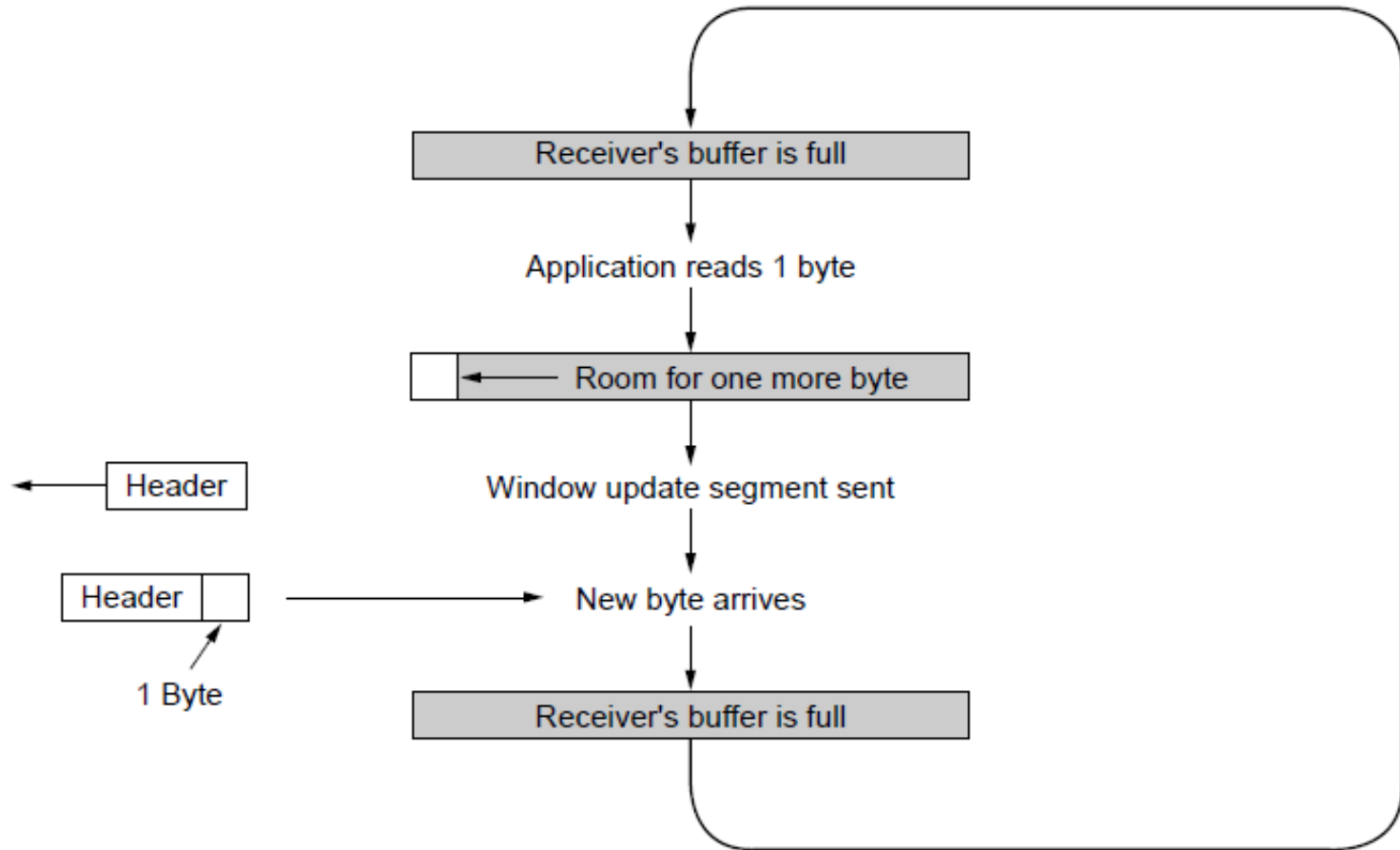
The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events. Each transition is labeled by the event causing it and the action resulting from it, separated by a slash.

# TCP Sliding Window (1)



Window management in TCP

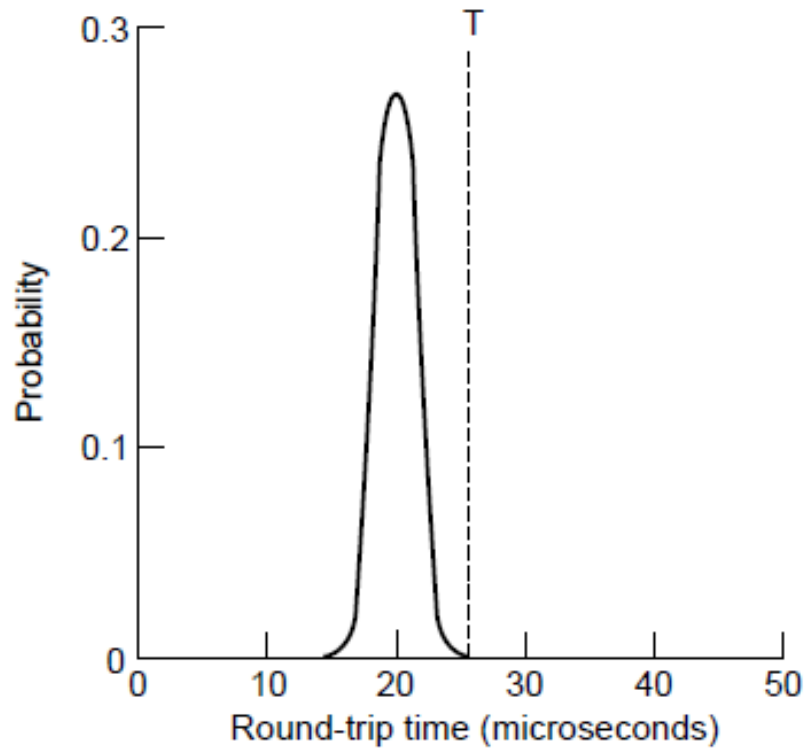
# TCP Sliding Window (2)



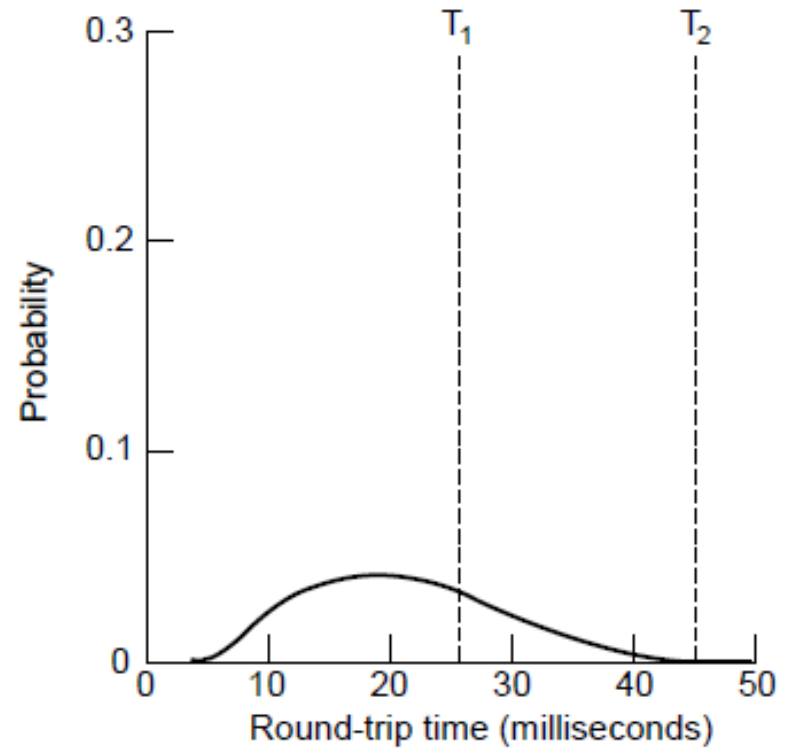
Silly window syndrome



# TCP Timer Management

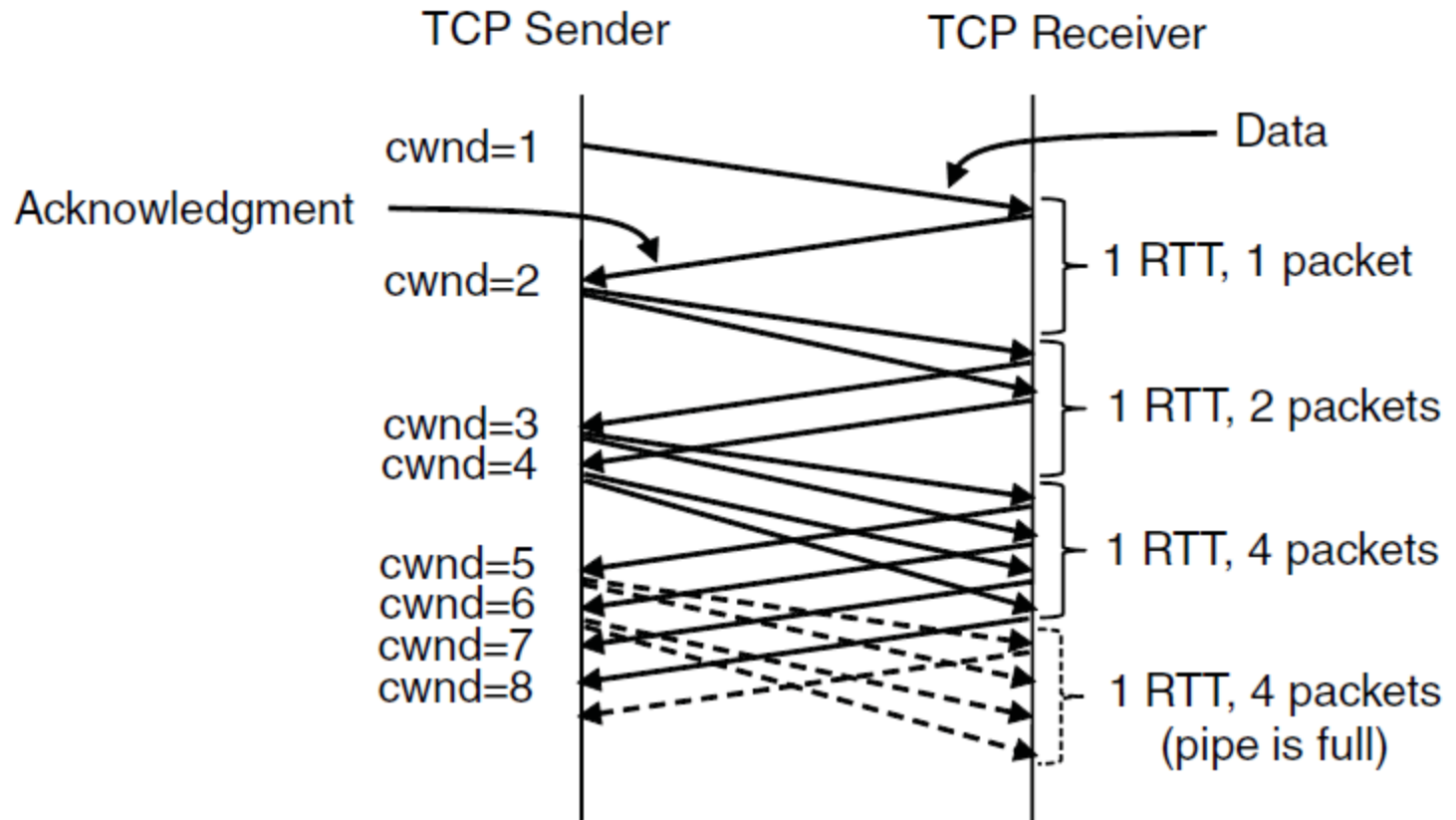


(a)

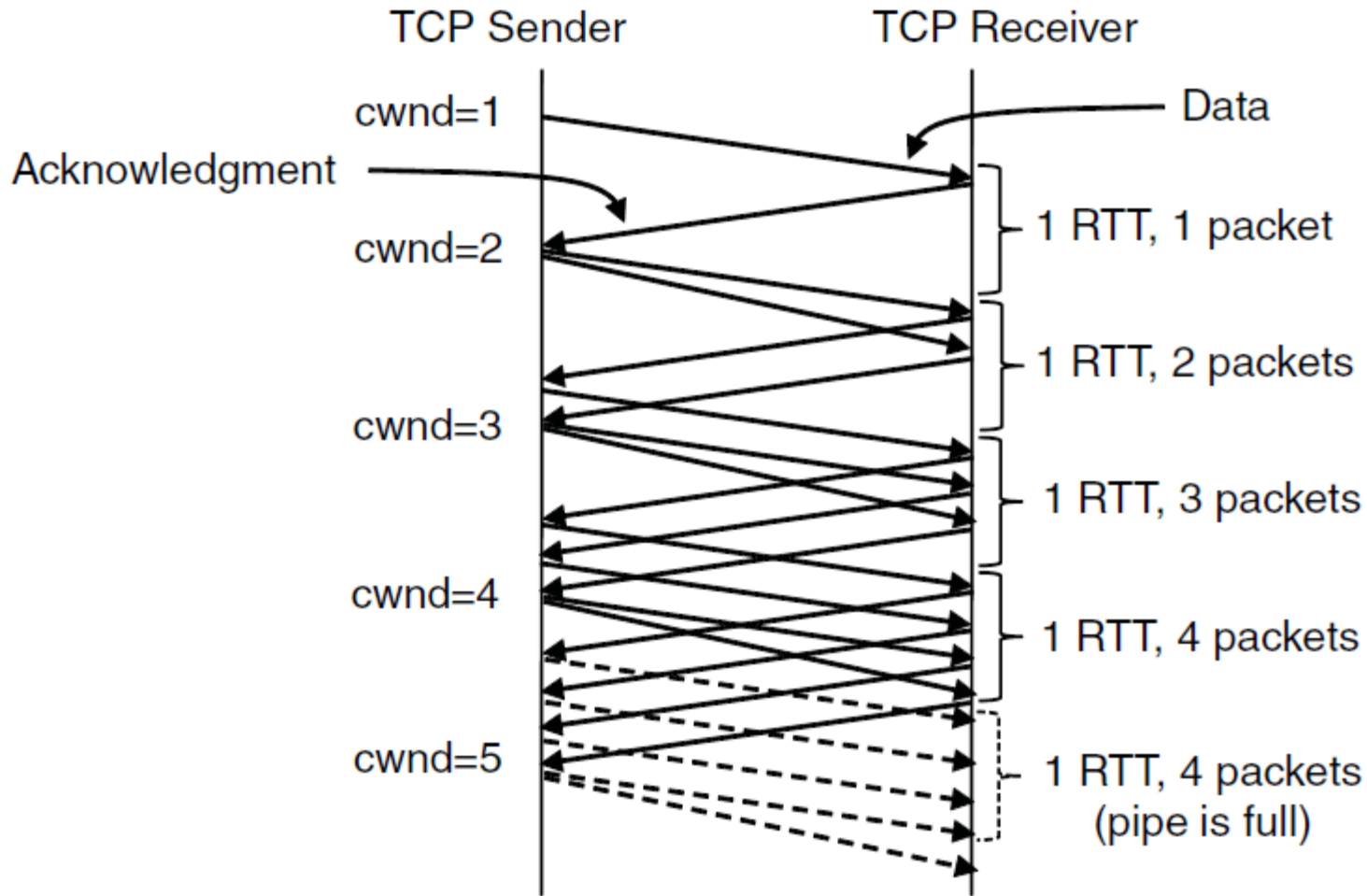


(b)

# TCP Congestion Control (1)

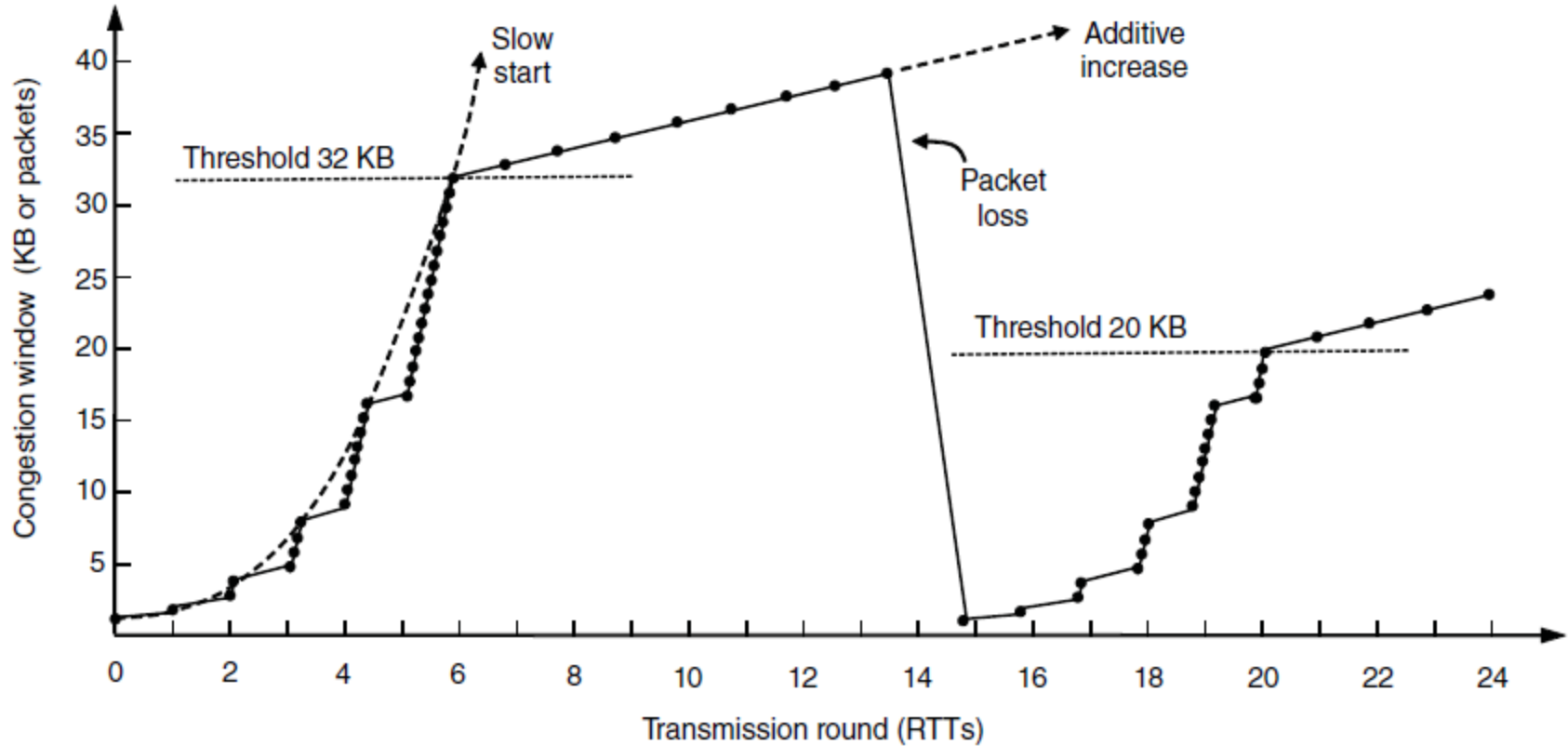


# TCP Congestion Control (2)



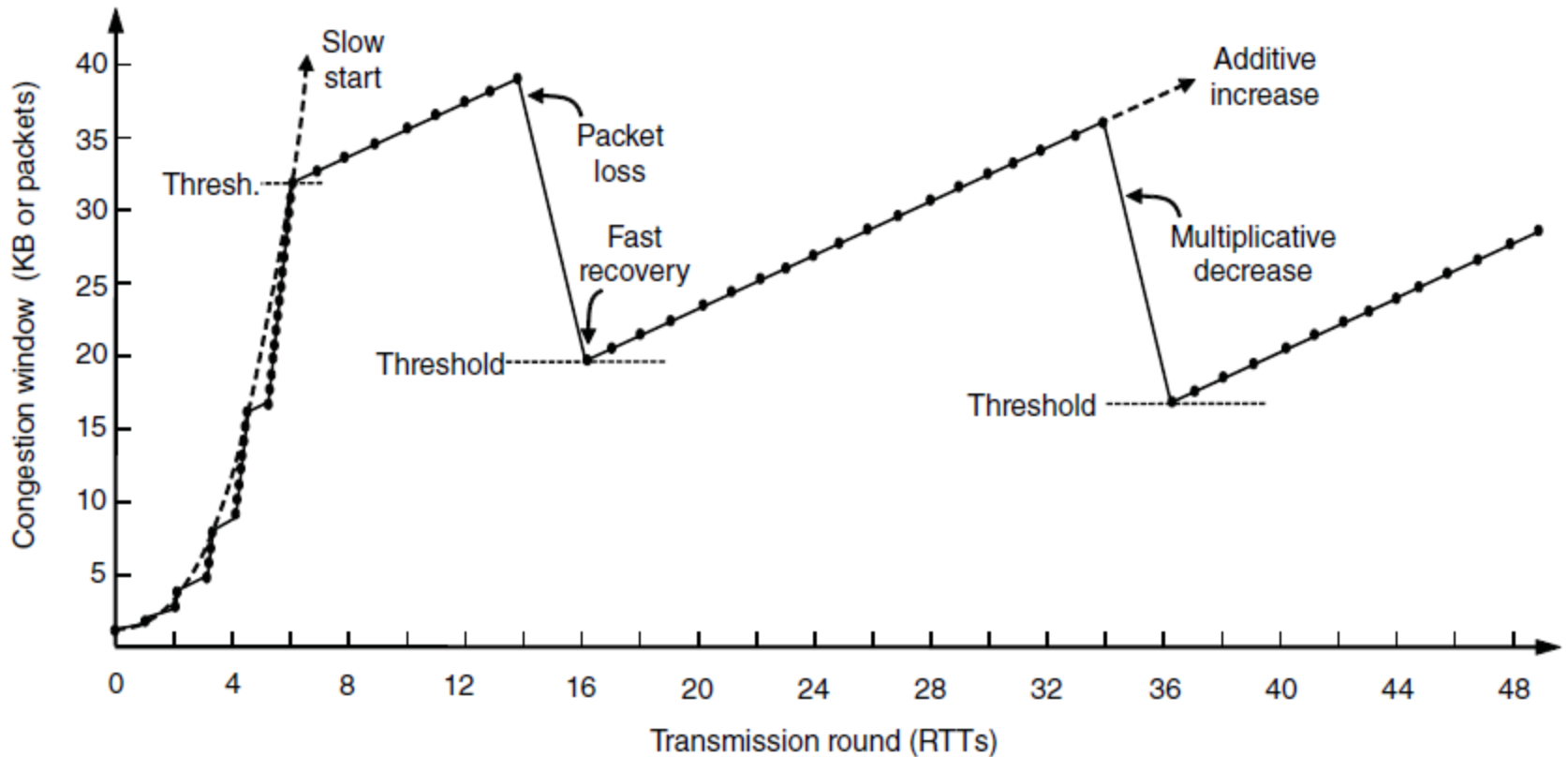
Additive increase from an initial congestion window of 1 segment.

# TCP Congestion Control (3)



Slow start followed by additive increase in TCP Tahoe.

# TCP Congestion Control (4)

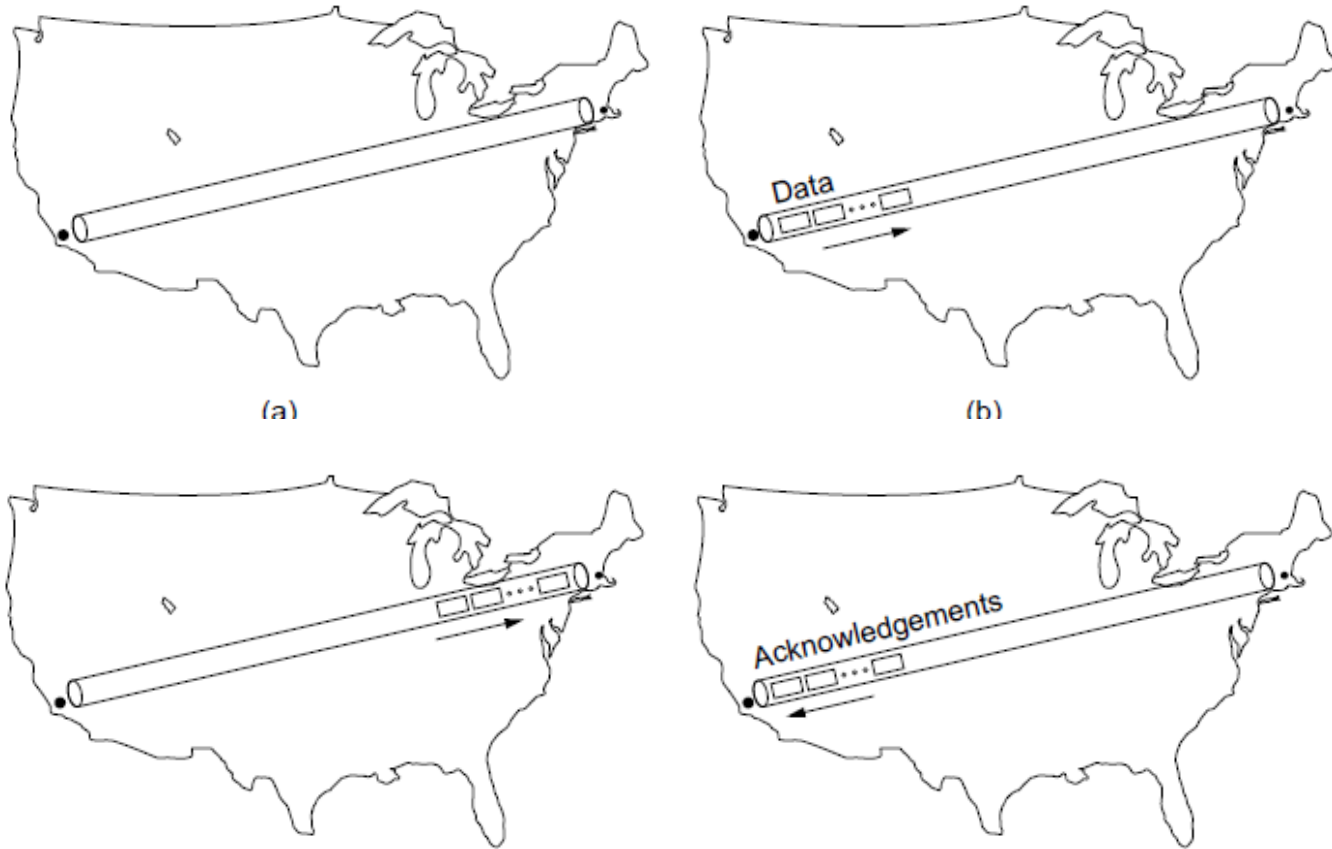


Fast recovery and the sawtooth pattern of TCP Reno.

# Performance Issues

- Performance problems in computer networks
- Network performance measurement
- System design for better performance
- Fast TPDU processing
- Protocols for high-speed networks

# Performance Problems in Computer



The state of transmitting one megabit from San Diego to Boston. (a) At  $t = 0$ . (b) After  $500 \mu sec$ .  
(c) After  $20 msec$ . (d) After  $40 msec$ .

# Network Performance Measurement (1)

Steps to performance improvement

1. Measure relevant network parameters, performance.
2. Try to understand what is going on.
3. Change one parameter.

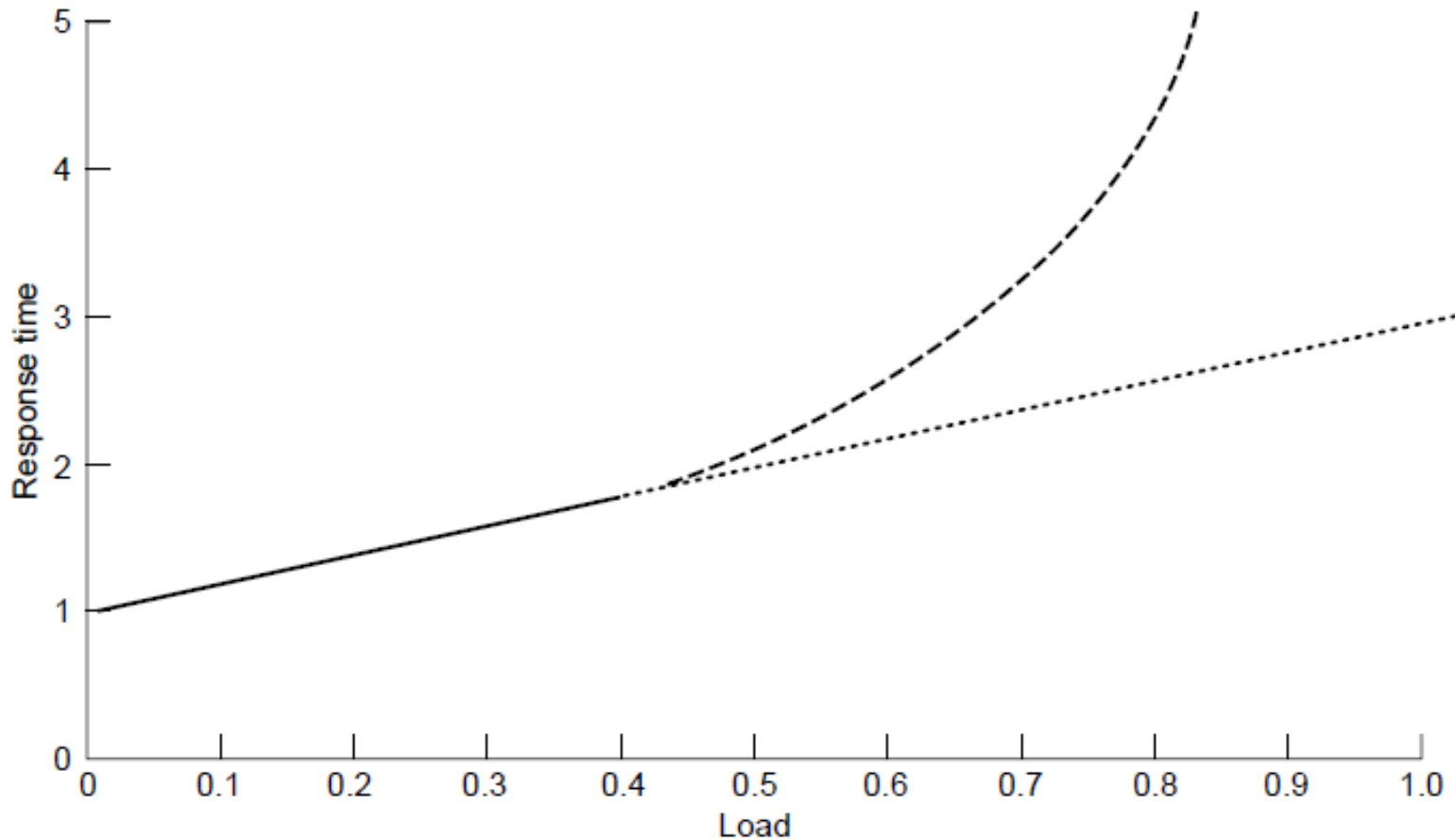


# Network Performance Measurement (2)

## Issues in measuring performance

- Sufficient sample size
- Representative samples
- Clock accuracy
- Measuring typical representative load
- Beware of caching
- Understand what you are measuring
- Extrapolate with care

# Network Performance Measurement (3)



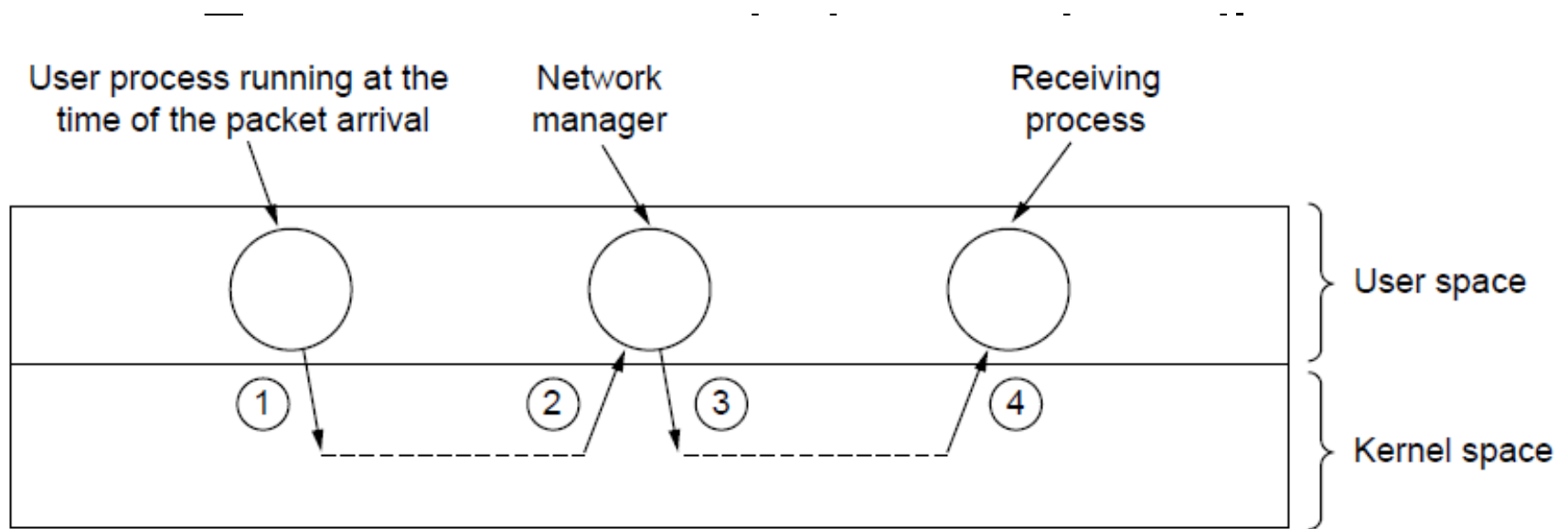
Response as a function of load.

# System Design for Better Performance (1)

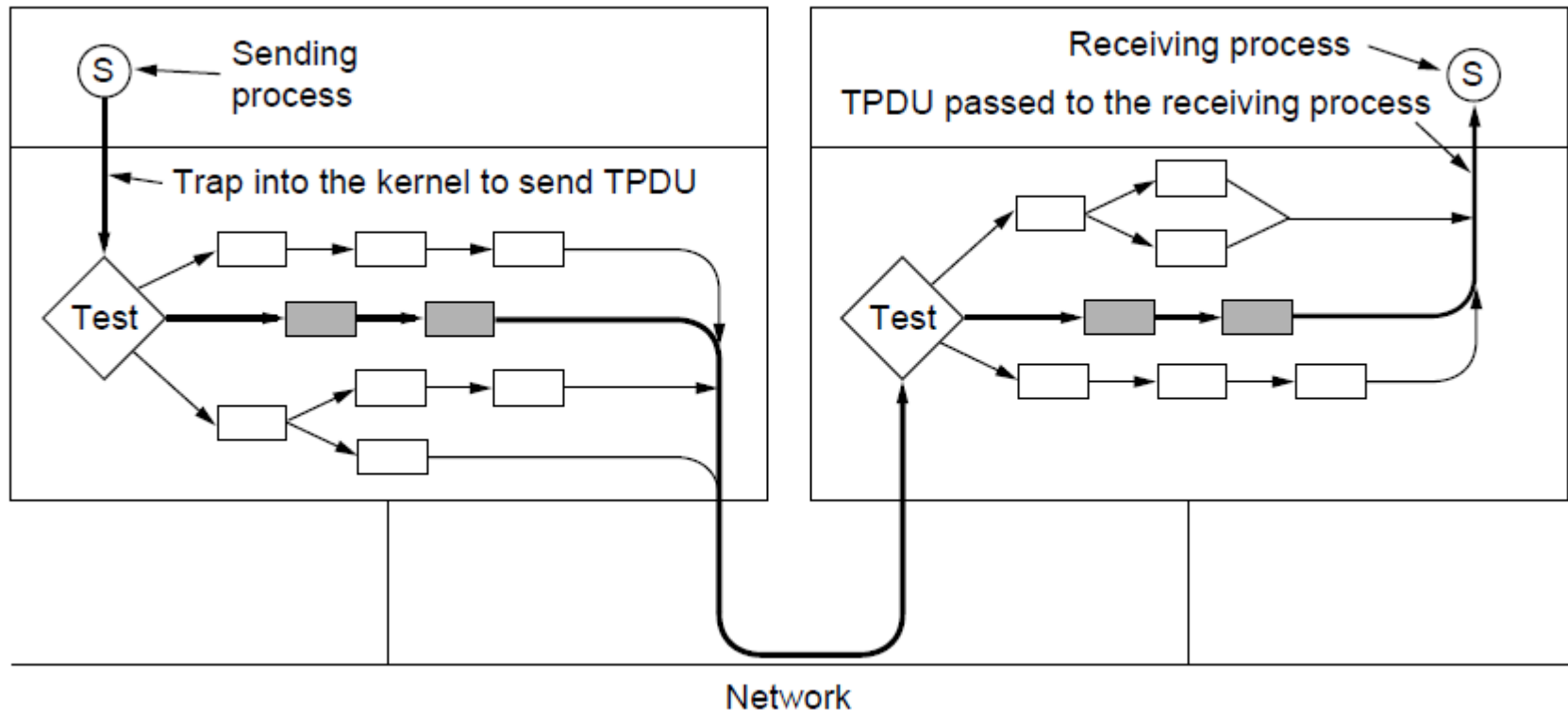
## Rules of thumb

1. CPU speed more important than network speed
2. Reduce packet count to reduce software overhead
3. Minimize data touching
4. Minimize context switches
5. Minimize copying
6. You can buy more bandwidth but not lower delay
7. Avoiding congestion is better than recovering from it
8. Avoid timeouts

# System Design for Better Performance (2)



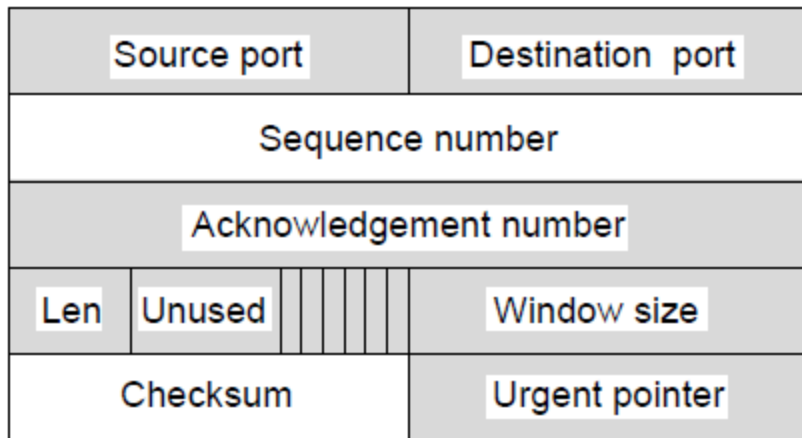
# Fast TPDU Processing (1)



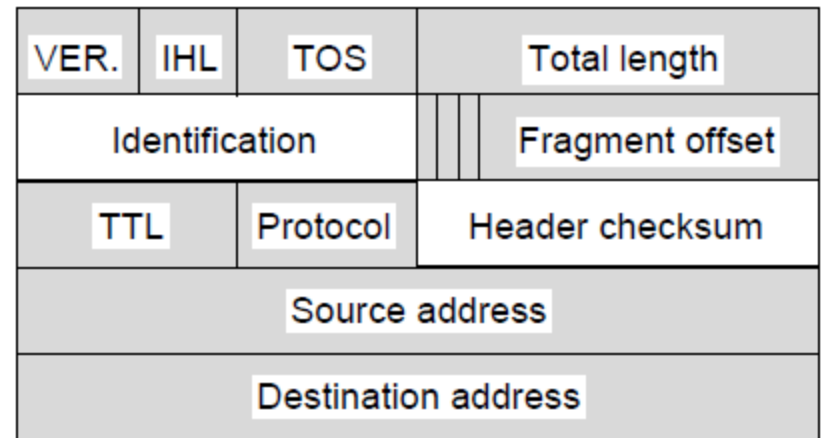
The fast path from sender to receiver is shown with a heavy line. The processing steps on this

# Fast TPDU Processing (2)

(a) TCP header (b) IP header In both cases

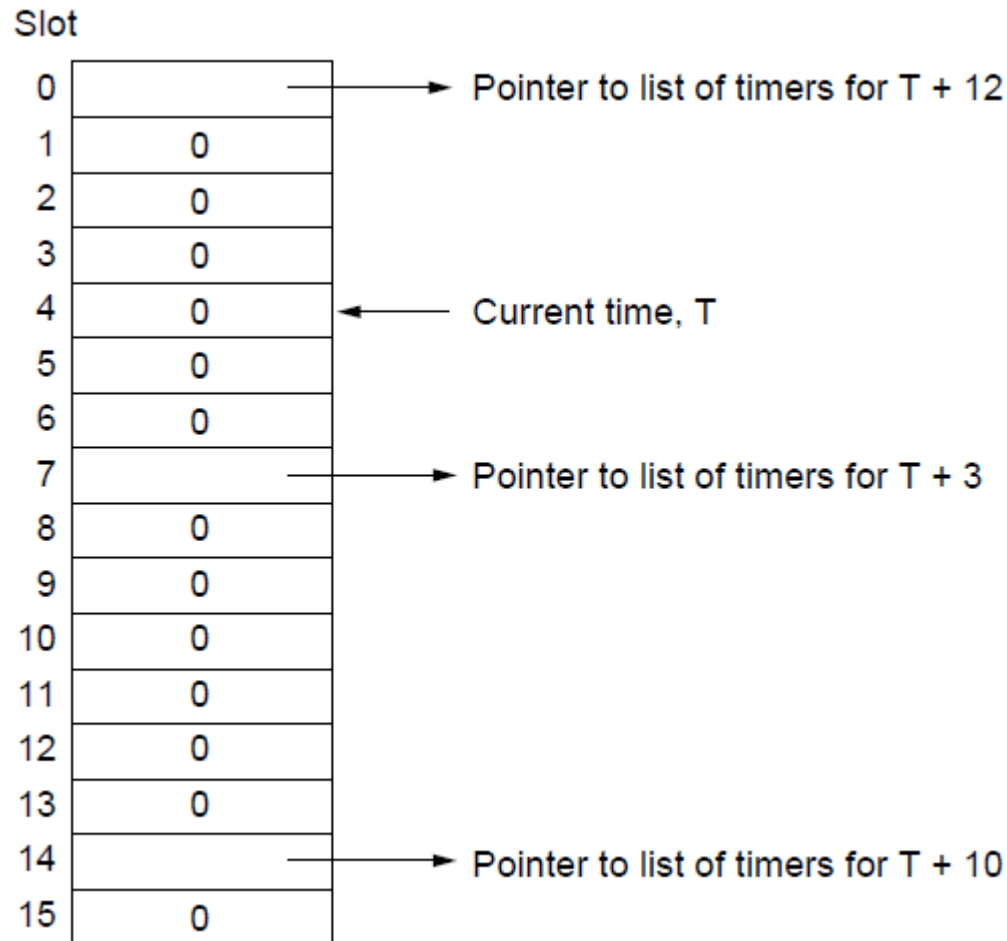


(a)



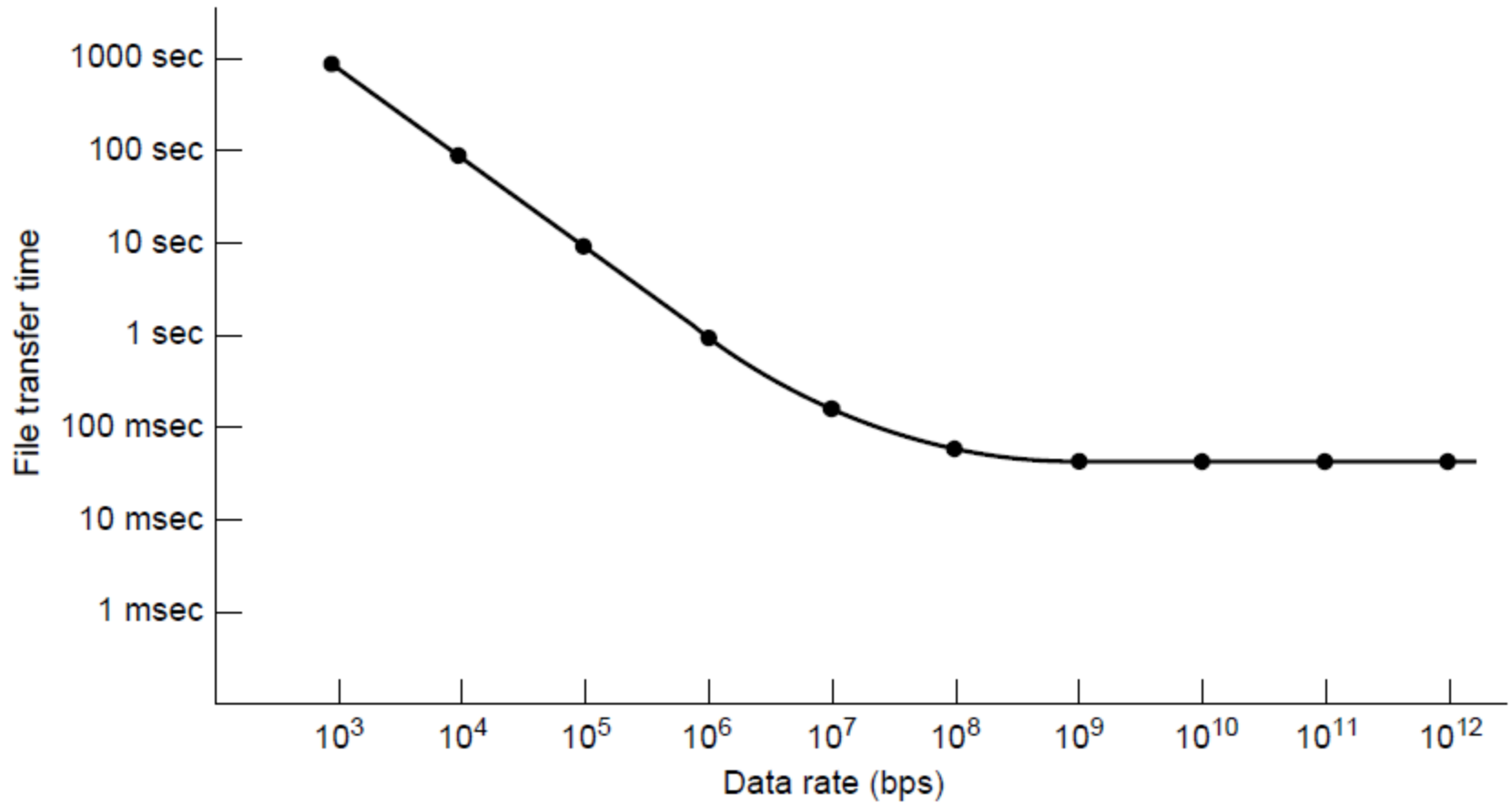
(b)

# Protocols for High-Speed Networks (1)



A timing wheel

# Protocols for High-Speed Networks (2)

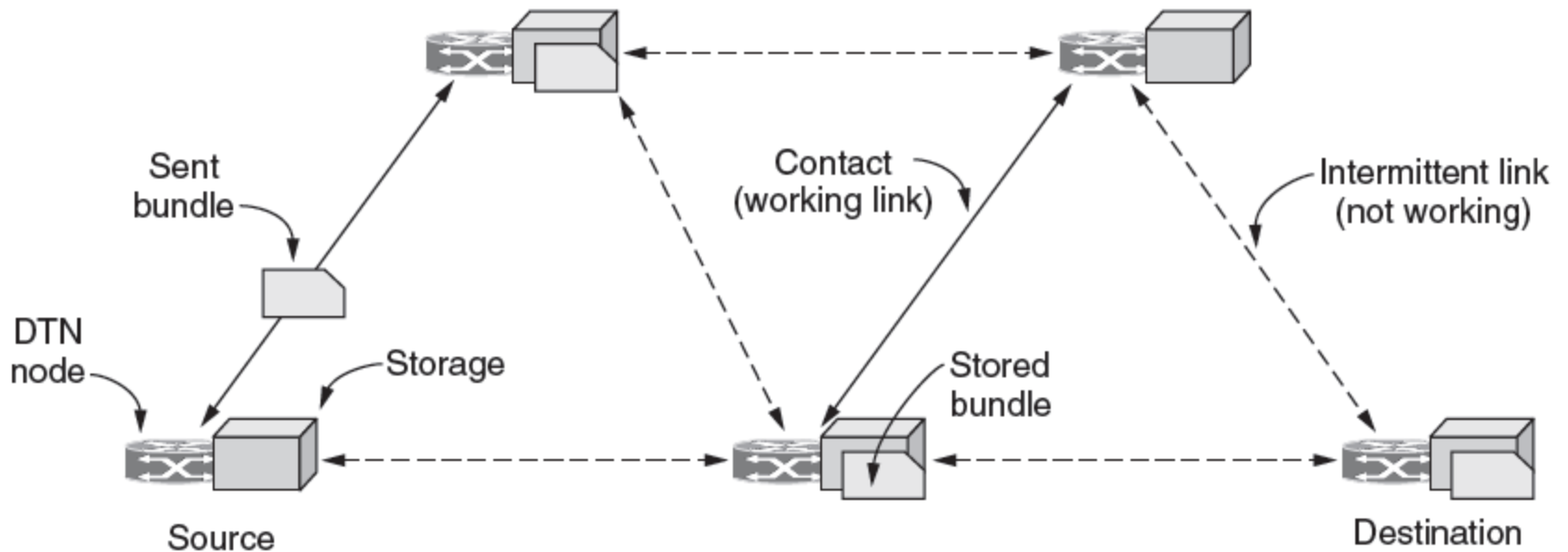




# Delay Tolerant Networking

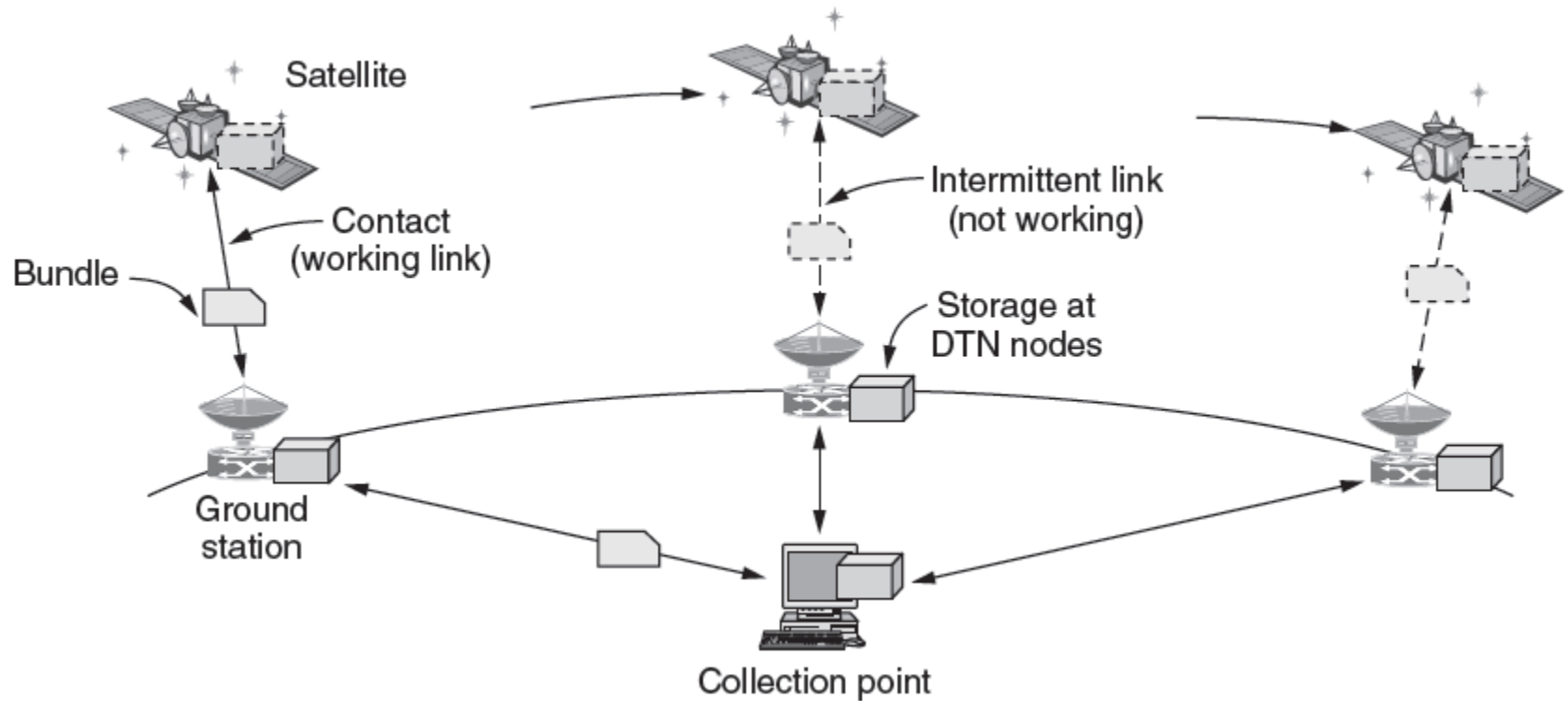
- DTN Architecture
- The Bundle Protocol

# DTN Architecture (1)



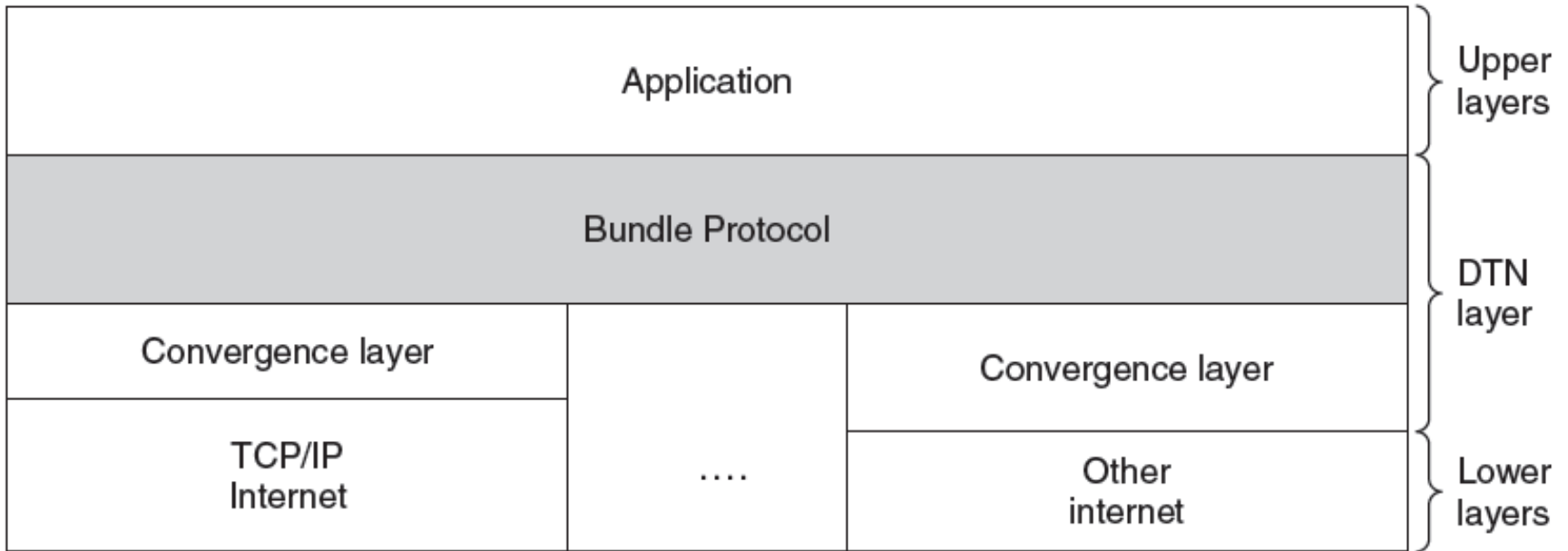
Delay-tolerant networking architecture

# DTN Architecture (2)



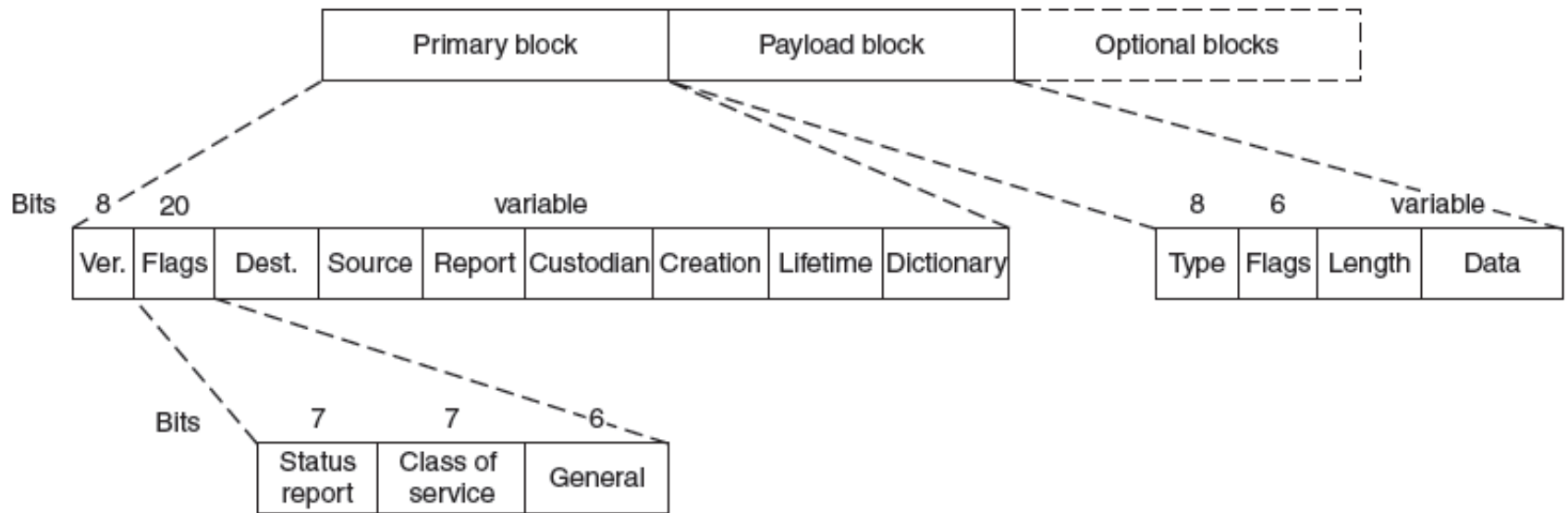
Use of a DTN in space.

# The Bundle Protocol (1)



Delay-tolerant networking protocol stack.

# The Bundle Protocol (2)



Bundle protocol message format.

Application Layer

# Processes communicating

## Process:

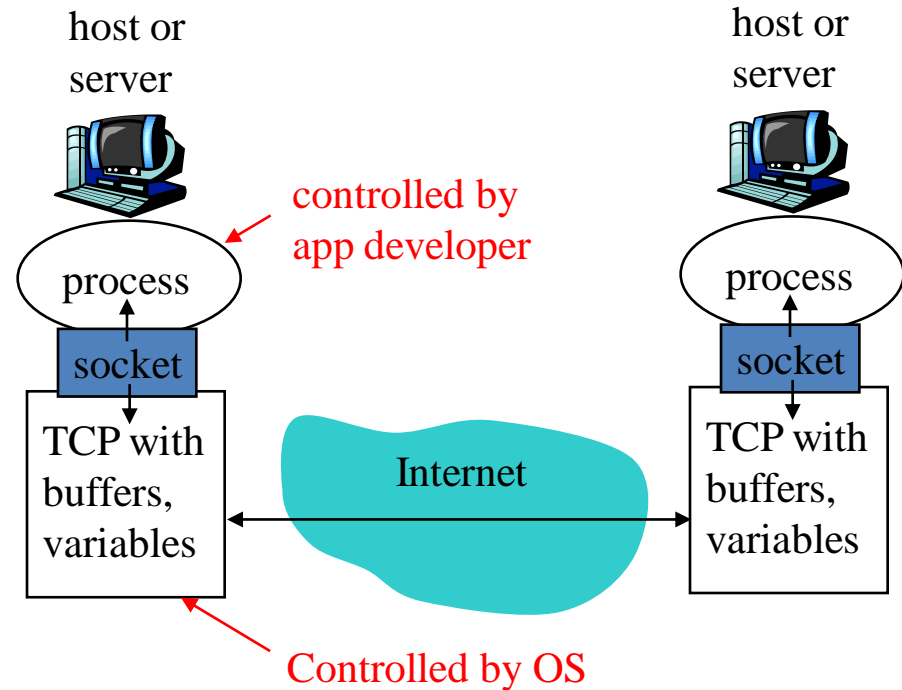
program running within a host

## Client process:

initiates communication

## Server process:

waits to be contacted



process sends/receives messages to/from its **socket**

*identifier* includes both **IP address** and **port numbers** associated with process on host.

# App-layer protocol defines

- Types of messages exchanged,
  - e.g., request, response
- Message syntax:
  - what fields in messages & how fields are delineated
- Message semantics
  - meaning of information in fields
- Rules for when and how processes send & respond to messages

## Public-domain protocols:

- ◆ defined in RFCs
- ◆ allows for interoperability
- ◆ e.g., HTTP, SMTP

## Proprietary protocols:

- ◆ e.g., Skype



# Transport service requirements of common apps

<b>Application</b>	<b>Data loss</b>	<b>Throughput</b>	<b>Time Sensitive</b>
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video: 10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

# Internet transport protocols services

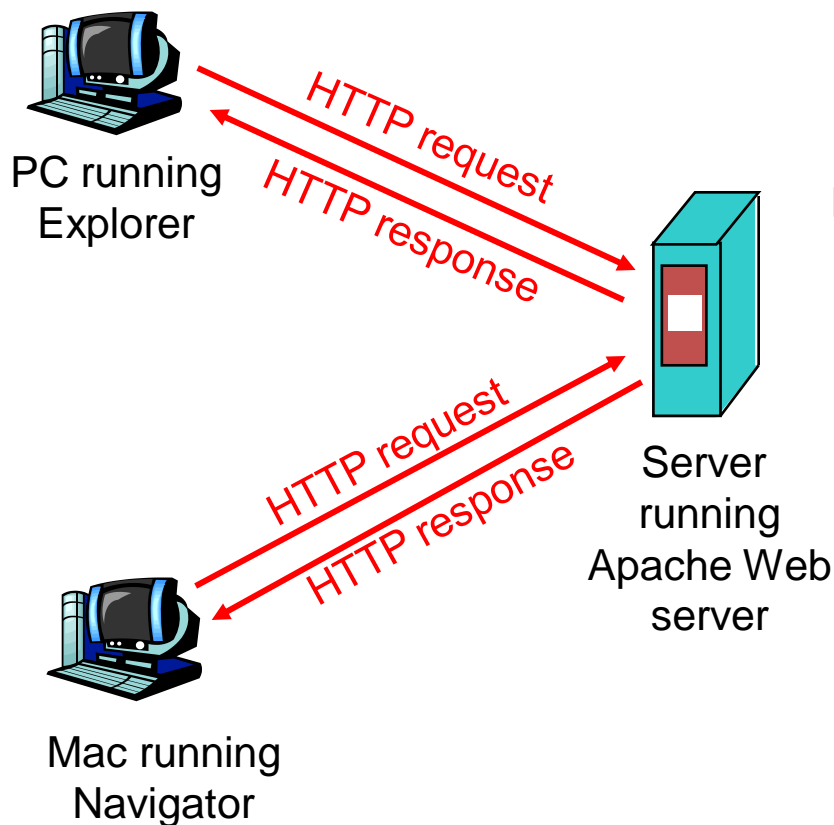
## TCP service:

- *connection-oriented*: setup required between client and server processes
- *reliable transport* between sending and receiving process
- *flow control*: sender won't overwhelm receiver
- *congestion control*: throttle sender when network overloaded
- *does not provide*: timing, minimum throughput guarantees, security

## UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security

# HTTP overview



- ❑ Web page consists of **base HTML-file** which includes several referenced **objects**
- ❑ Each object is addressable by a **URL**

**HTTP: hypertext transfer protocol**

- ❑ Web's application layer protocol
- ❑ client/server model
  - ❖ **client**: browser that requests, receives, "displays" Web objects
  - ❖ **server**: Web server sends objects in response to requests
- ❑ uses TCP
- ❑ is "**stateless**"

# HTTP connections

## Nonpersistent HTTP

- At most one object is sent over a TCP connection.

## Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.

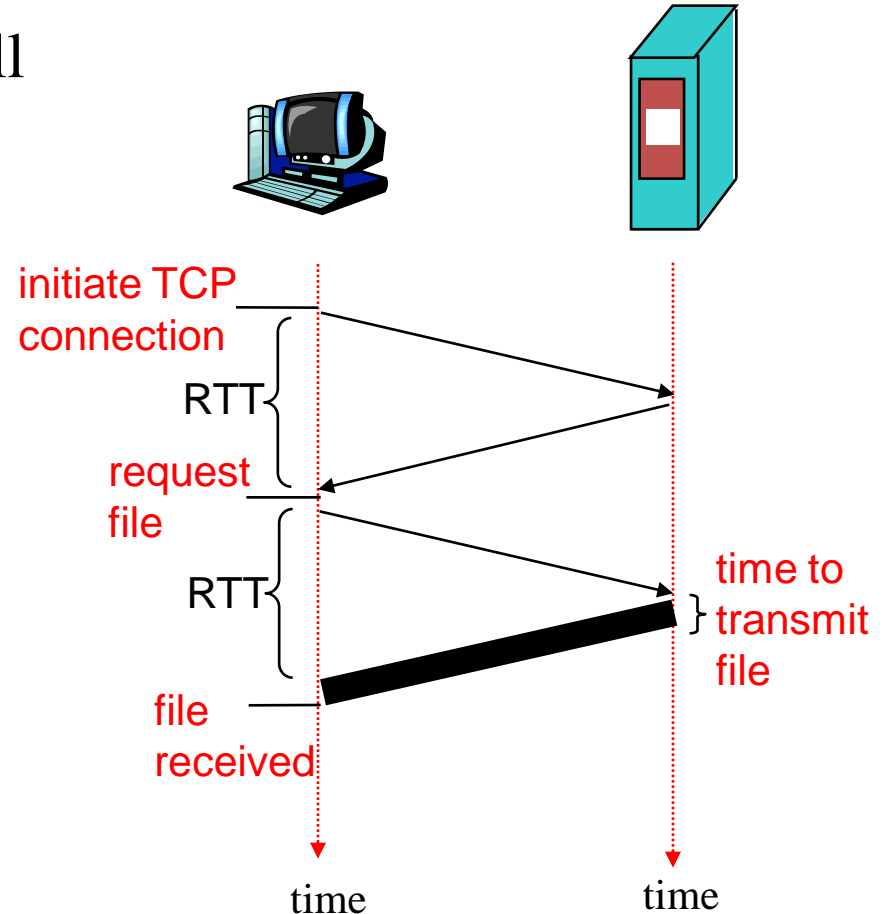
# Non-Persistent HTTP: Response time

**Definition of RTT:** time for a small packet to travel from client to server and back.

## Response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time

**total =  $2RTT + \text{transmit time}$**



# Persistent HTTP

## Nonpersistent HTTP issues:

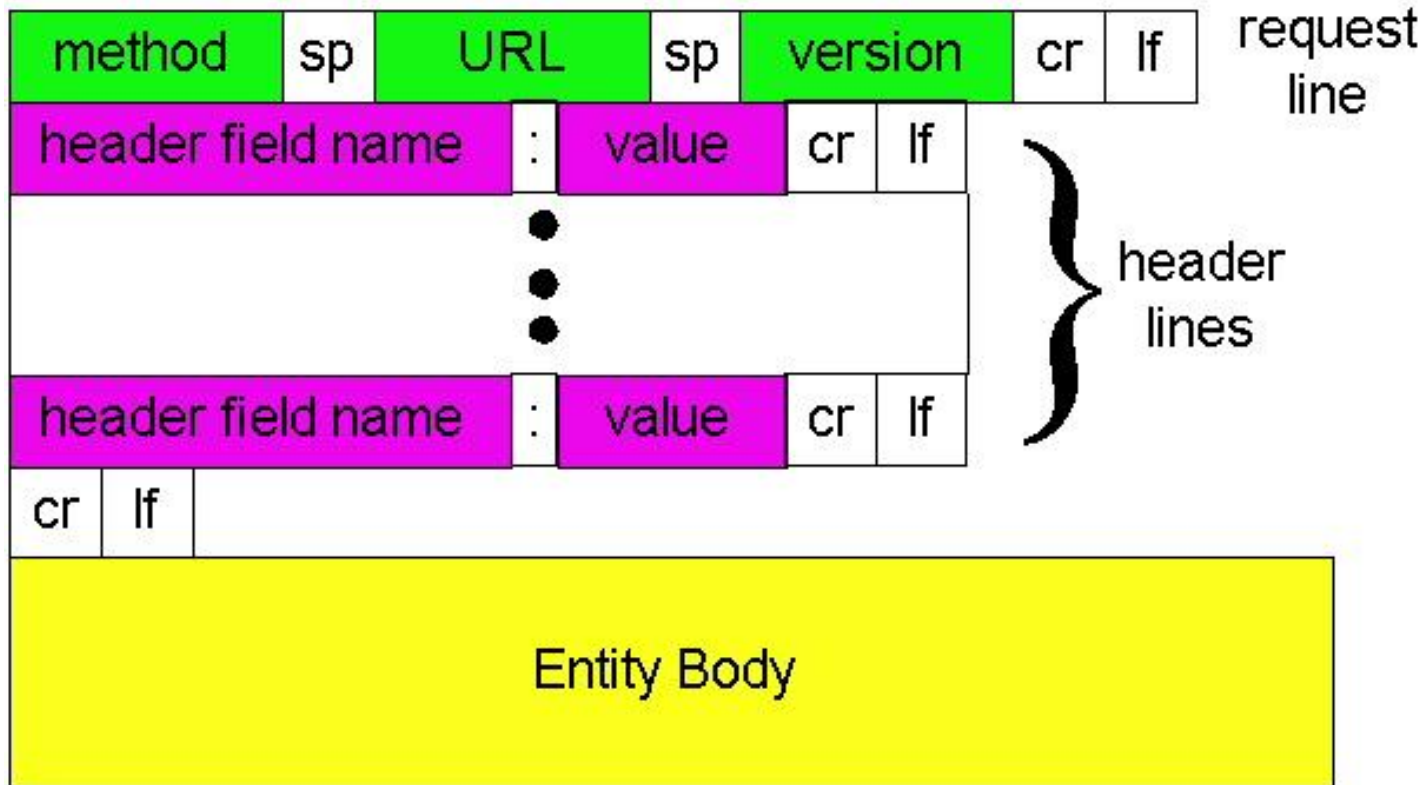
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

## Persistent HTTP

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

# HTTP messages

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ❖ ASCII (human-readable format)



# Method types

## HTTP/1.0

- GET
  - request an object from server
- POST
  - upload information using forms
- HEAD
  - asks server to leave requested object out of response

## HTTP/1.1

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field



# Cookies: Keeping state

## What cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

- aside
- Cookies and privacy:
- cookies permit sites to learn a lot about you
  - you may supply name and e-mail to sites

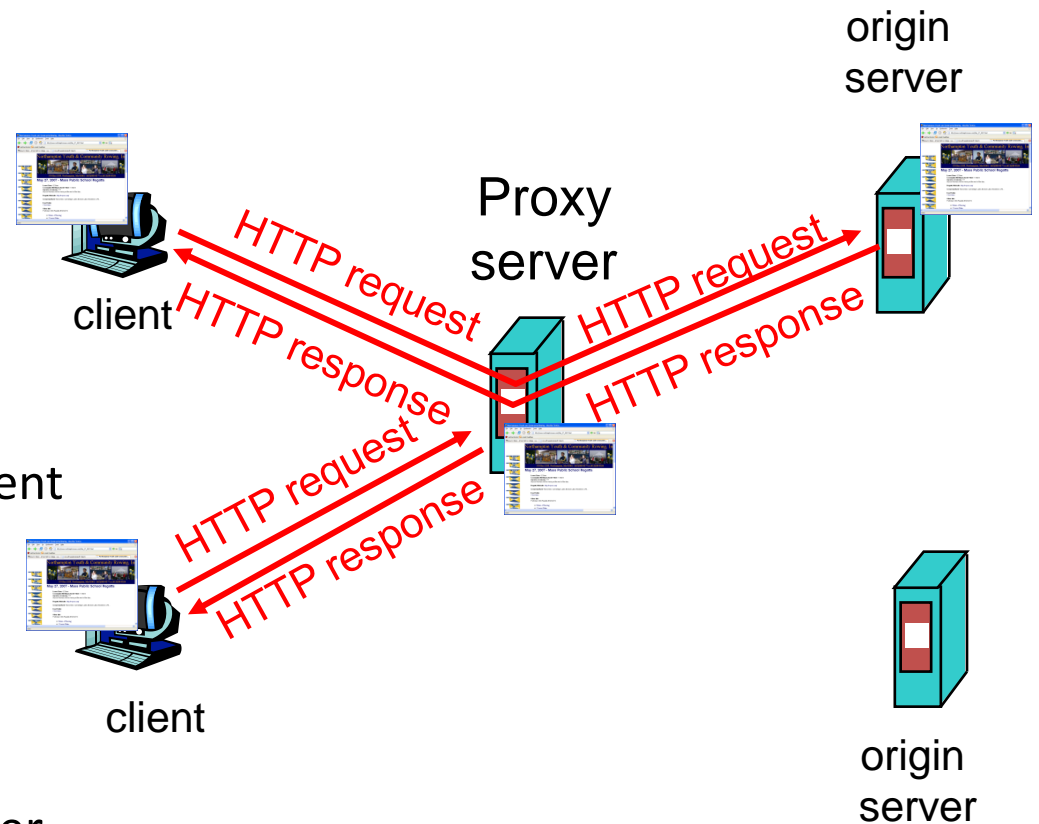
## How to keep “state”:

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

# Web caches (proxy server)

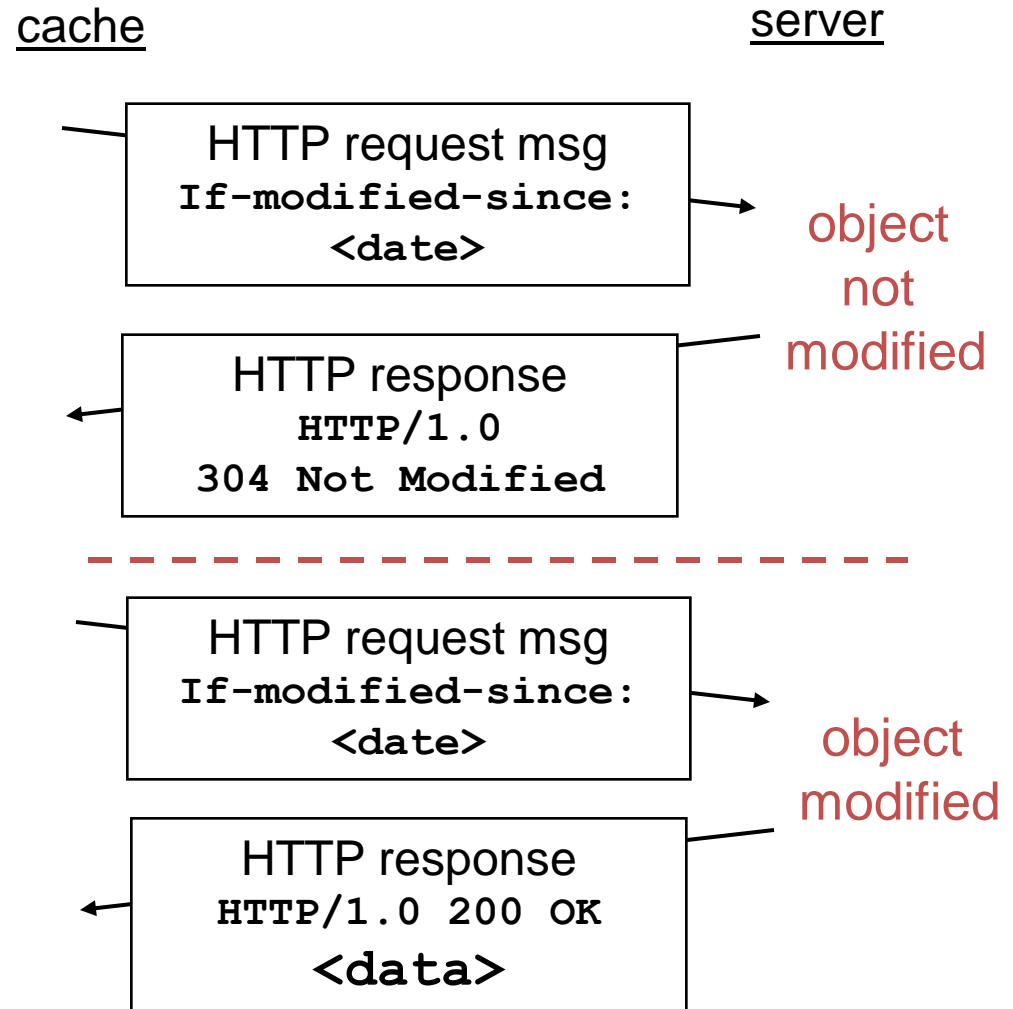
**Goal:** satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
- Why Web caching?
  - reduce response time for client request
  - reduce traffic on an institution's access link.
  - enables “poor” content providers to effectively deliver content

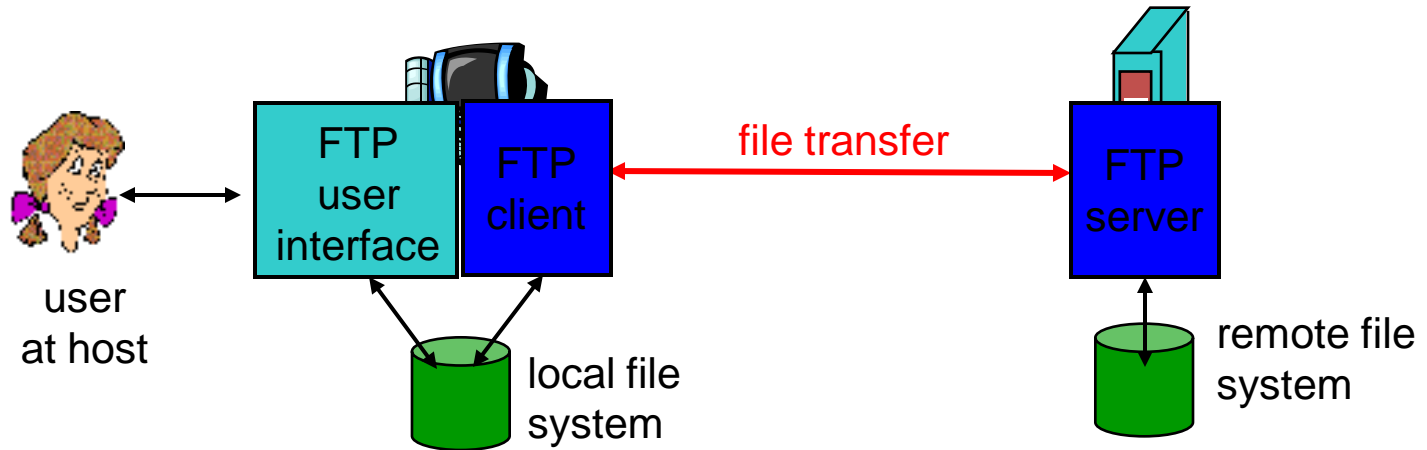


# Conditional GET

- **Goal:** don't send object if cache has up-to-date cached version
- **cache:** specify date of cached copy in HTTP request  
`If-modified-since: <date>`
- **server:** response contains no object if cached copy is up-to-date:  
`HTTP/1.0 304 Not Modified`



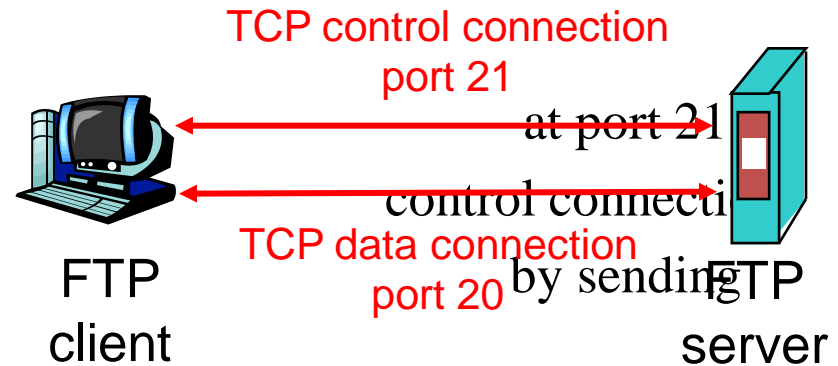
# FTP: the file transfer protocol



- transfer file to/from remote host
- client/server model
  - *client*: side that initiates transfer (either to/from remote)
  - *server*: remote host
- ftp: RFC 959
- ftp server: port 21

# FTP: separate control, data connections

- FTP client contacts FTP server
- client authorized over
- client browses remote directory commands over control connection.



- when server receives file transfer command, server opens 2<sup>nd</sup> TCP connection (for file) to client
- after transferring one file, server closes data connection.
- server opens another TCP data connection to transfer another file.
- control connection: “out of band”
- FTP server maintains “state”: current directory, earlier authentication

# FTP commands, responses

## Sample commands:

- sent as ASCII text over control channel
- **USER *username***
- **PASS *password***
- **LIST** return list of file in current directory
- **RETR *filename*** retrieves (gets) file
- **STOR *filename*** stores (puts) file onto remote host

## Sample return codes

- status code and phrase (as in HTTP)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

# FTP issues

- Multiple connections are used
  - for each directory listing and file transmission
- No integrity check at receiver
- Messages are sent in clear text
  - including **Passwords** and file contents
  - can be sniffed by eavesdroppers
- Solution
  - Secure FTP (SSH FTP)
    - allows a range of operations on remote files
  - FTPS ( FTP over Secure Sockets Layer (SSL) )
  - Transport Layer Security (TLS) encryption

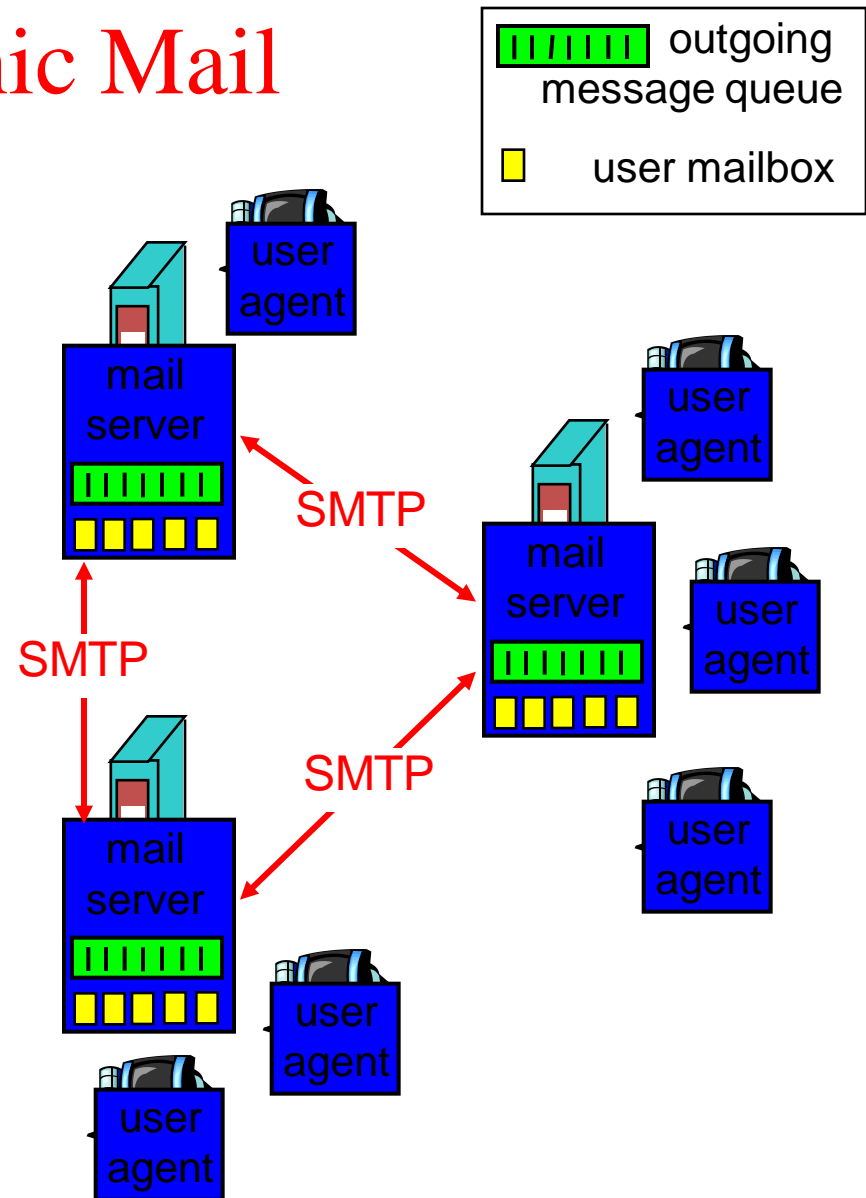
# Electronic Mail

## Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

## User Agent

- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Eudora, Outlook, elm, Mozilla Thunderbird
- outgoing, incoming messages stored on server

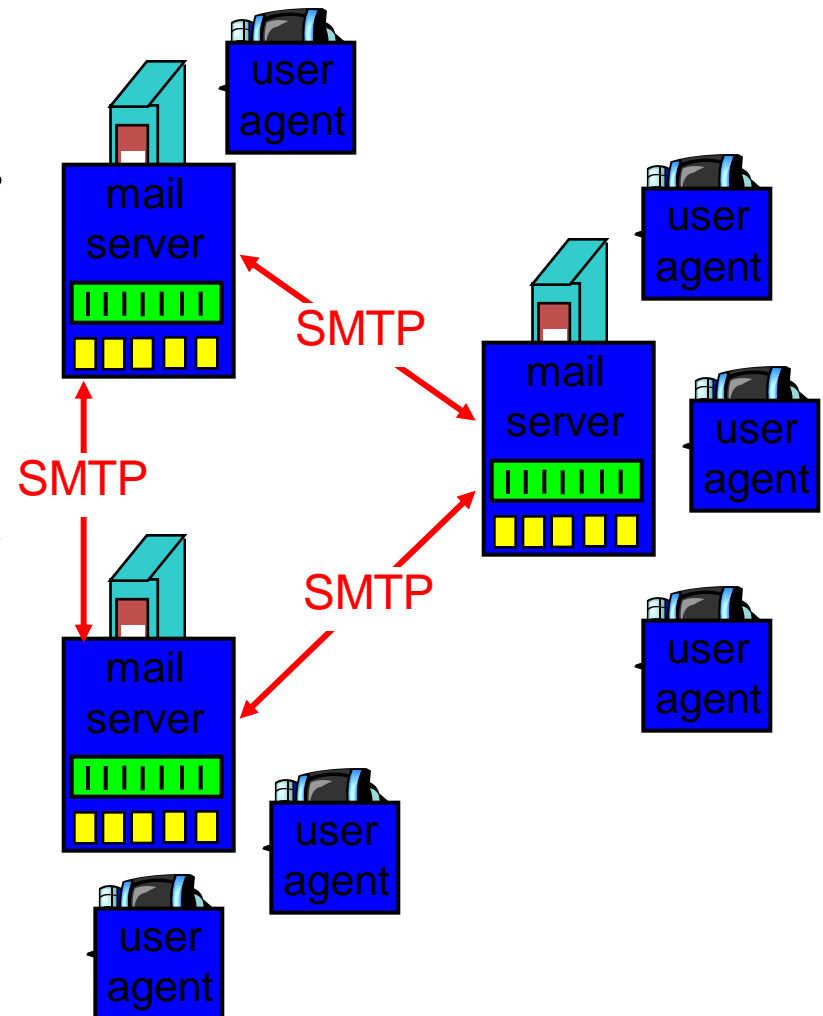




# Electronic Mail: mail servers

## Mail Servers

- **mailbox** contains incoming messages for user
- **message queue** of outgoing (to be sent) mail messages
- **SMTP protocol** between mail servers to send email messages
  - client: sending mail server
  - “server”: receiving mail server

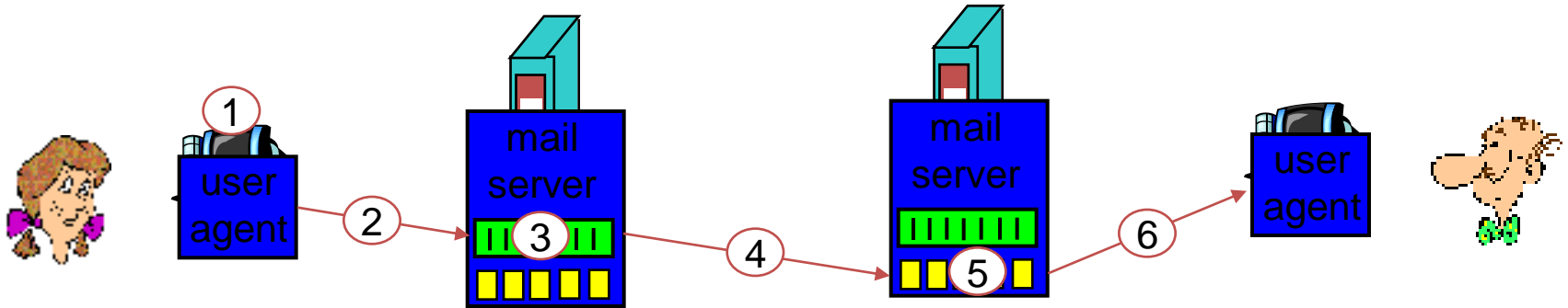


# Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server (port 25)
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction
  - **commands**: ASCII text
  - **response**: status code and phrase
- messages must be in 7-bit ASCII

# Scenario: Alice sends message to Bob

- 1) Alice uses UA to compose message and “to”  
bob@someschool.edu
- 2) Alice’s UA sends message to her mail server; message placed in message queue
- 3) Client side of SMTP opens TCP connection with Bob’s mail server
- 4) SMTP client sends Alice’s message over the TCP connection
- 5) Bob’s mail server places the message in Bob’s mailbox
- 6) Bob invokes his user agent to read message



# Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

# SMTP: final words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF . CRLF to determine end of message

## Comparison with HTTP:

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

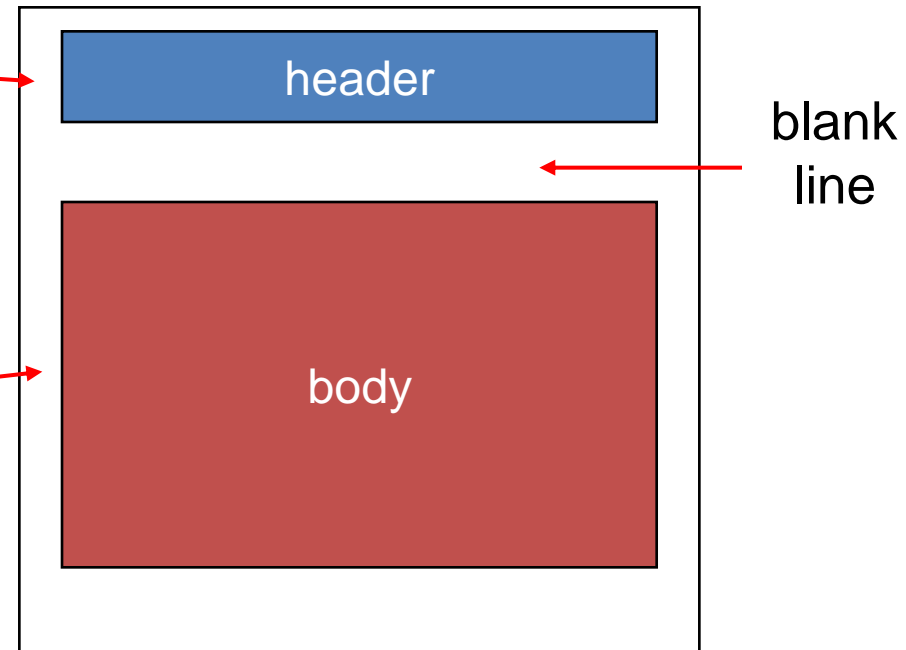
# Mail message format

SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

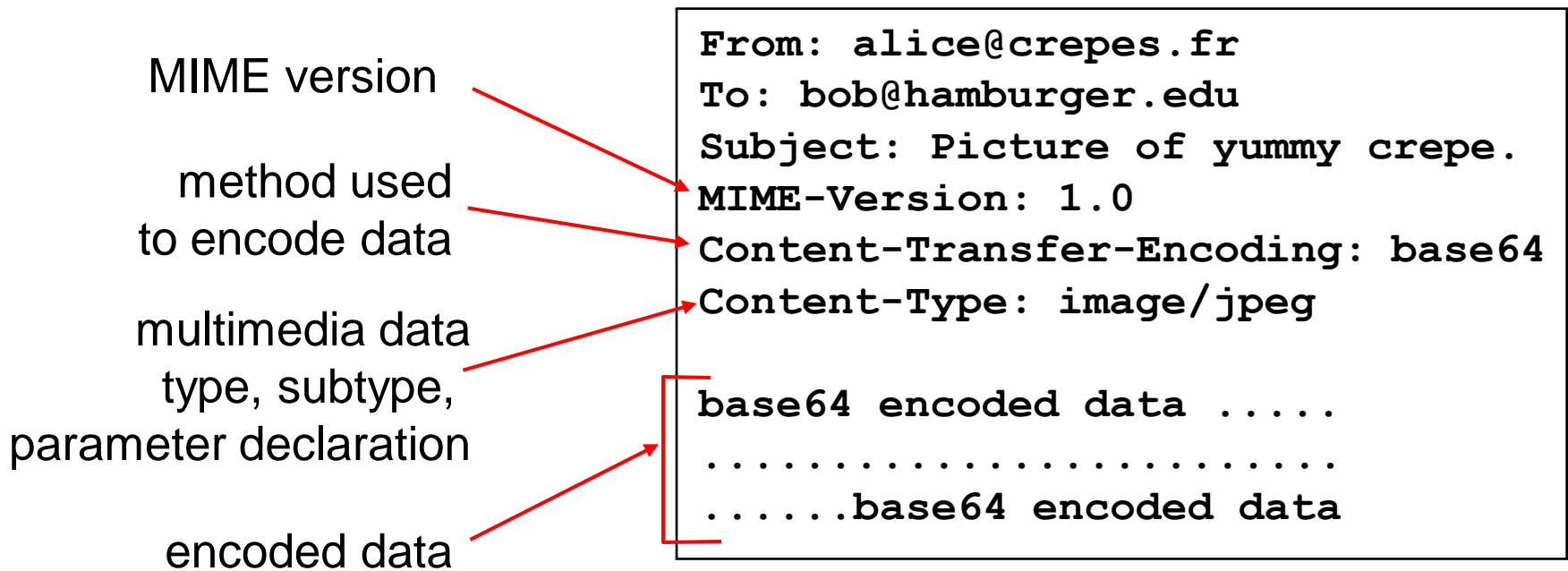
- header lines, e.g.,
  - To:
  - From:
  - Subject:*different from SMTP commands!*

- body
  - the “message”,  
*ASCII characters only*

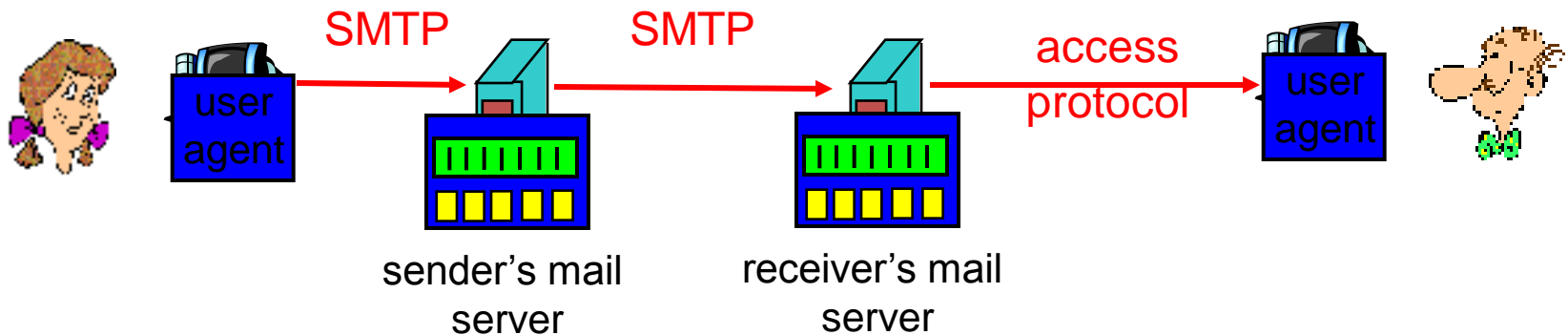


# Message format: multimedia extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- additional lines in msg header declare MIME content type



# Mail access protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored msgs on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.



# POP3 protocol

## authorization phase

- client commands:
  - **user**: declare username
  - **pass**: password
- server responses
  - **+OK**
  - **-ERR**

## transaction phase, client:

- **list**: list message numbers
- **retr**: retrieve message by number
- **dele**: delete
- **quit**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

# POP3 (more) and IMAP

## More about POP3

- Previous example uses “download and delete” mode.
- Bob cannot re-read e-mail if he changes client
- “Download-and-keep”: copies of messages on different clients
- POP3 is stateless across sessions

## IMAP

- Keep all messages in one place: the server
- Allows user to organize messages in folders
- IMAP keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name

## Try SMTP interaction for yourself:

- `telnet servername 25`
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

# DNS: Domain Name System

**People:** many identifiers:

- SSN, name, passport #

**Internet hosts, routers:**

- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., ww.yahoo.com - used by humans

**Domain Name System:**

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol* host, routers, name servers to communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network’s “edge”

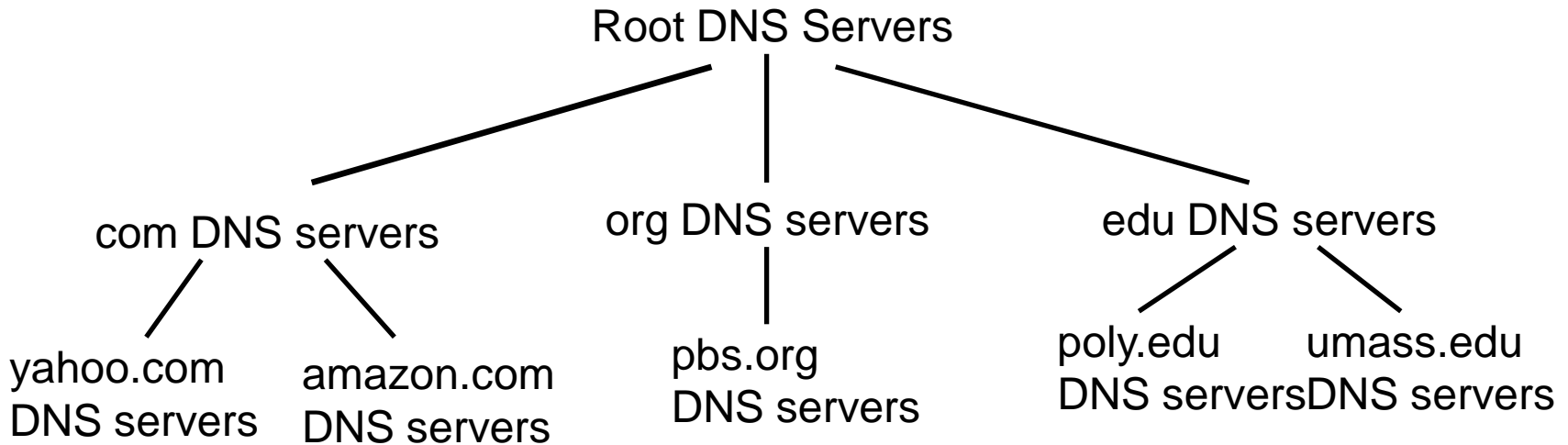
## DNS services

- hostname to IP address translation
- host aliasing
  - Canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: set of IP addresses for one canonical name

## Why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database *doesn't scale!*
- maintenance

# Distributed, Hierarchical Database



## Client wants IP for www.amazon.com; 1<sup>st</sup> approx:

- client queries a **root server** to find **com** DNS server
- client queries **com** DNS server to get **amazon.com** DNS server
- client queries **amazon.com** DNS server to get **IP** address for www.amazon.com

# Network Security

# Network Security

<b>Adversary</b>	<b>Goal</b>
Student	To have fun snooping on people's email
Cracker	To test out someone's security system; steal data
Sales rep	To claim to represent all of Europe, not just Andorra
Businessman	To discover a competitor's strategic marketing plan
Ex-employee	To get revenge for being fired
Accountant	To embezzle money from a company
Stockbroker	To deny a promise made to a customer by email
Con man	To steal credit card numbers for sale
Spy	To learn an enemy's military or industrial secrets
Terrorist	To steal germ warfare secrets

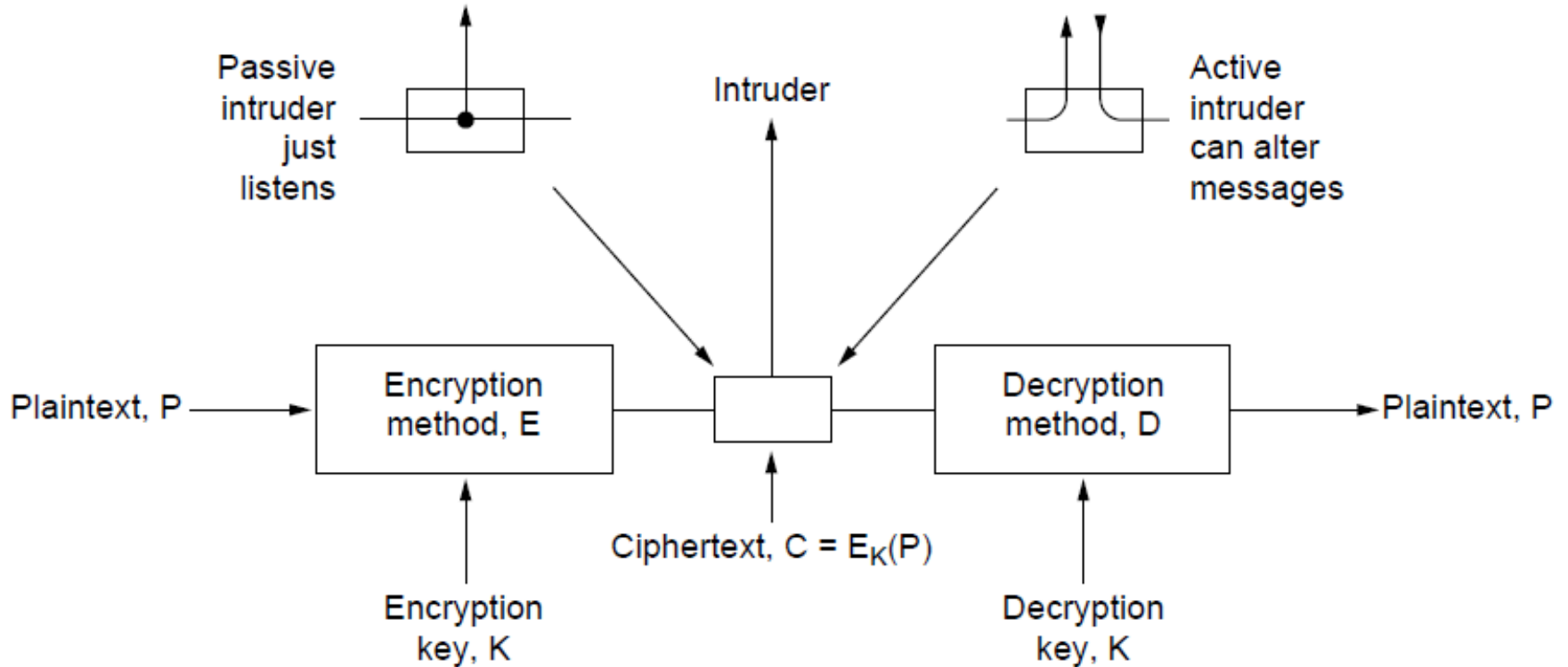
Some people who cause security problems and  
why



# Cryptography

- Introduction
- Substitution ciphers
- Transposition ciphers
- One-time pads
- Fundamental cryptographic principles

# Introduction



The encryption model (for a symmetric-key cipher).

# Substitution Ciphers

plaintext:	a b c d e f g h i j k l m n o p q r s t u v w x y z
ciphertext:	Q W E R T Y U I O P A S D F G H J K L Z X C V B N M

Monoalphabetic substitution

# Transposition Ciphers

M E G A B U C K

7 4 5 1 2 8 3 6

p l e a s e t r

a n s f e r o n

e m i l l i o n

d o l l a r s t

o m y s w i s s

b a n k a c c o

u n t s i x t w

o t w o a b c d

Plaintext

pleasetransferonemilliondollarsto  
myswissbankaccountsixtwo

Ciphertext

AFLLSKSOSELAWAIATOOSSCTCLNMOMANT  
ESILYNTWRNNTSOWDPAEDOBUEOERIRICXB

# One-Time Pads (1)

Message 1:	1001001 0100000 1101100 1101111 1110110 1100101 0100000 1111001 1101111 1110101 0101110
Pad 1:	1010010 1001011 1110010 1010101 1010010 1100011 0001011 0101010 1010111 1100110 0101011
Ciphertext:	0011011 1101011 0011110 0111010 0100100 0000110 0101011 1010011 0111000 0010011 0000101
Pad 2:	1011110 0000111 1101000 1010011 1010111 0100110 1000111 0111010 1001110 1110110 1110110
Plaintext 2:	1000101 1101100 1110110 1101001 1110011 0100000 1101100 1101001 1110110 1100101 1110011

The use of a one-time pad for encryption and the possibility of getting any possible plaintext from the ciphertext by the use of some other pad.

# One-Time Pads (2)

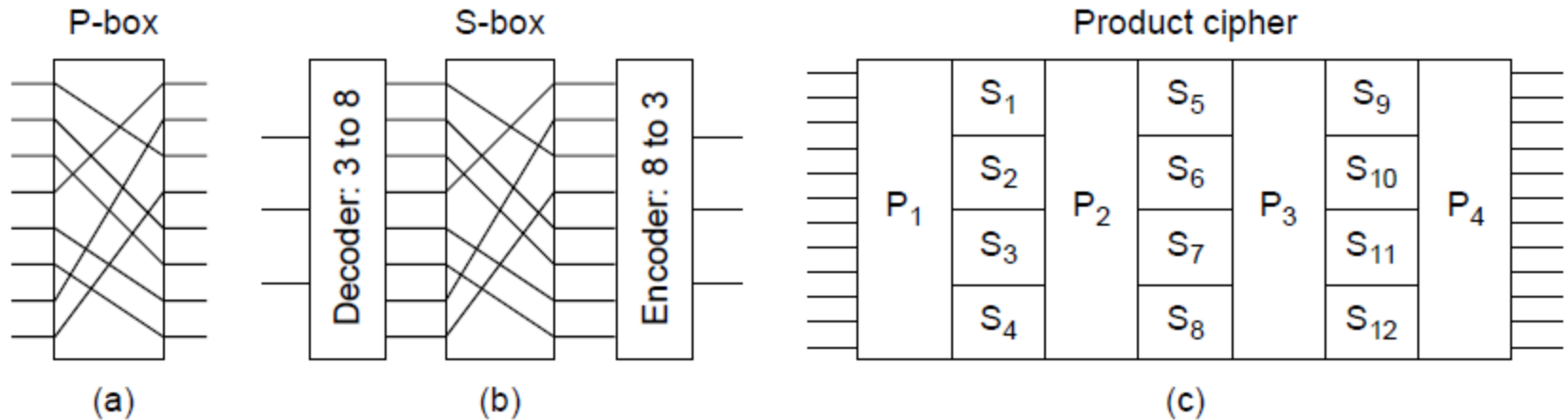
Bit number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Data	1	0	0	1	1	1	0	0	1	0	1	0	0	1	1	0	
(a)																	What Alice sends
(b)																	Bob's bases
(c)																	What Bob gets
(d)	No	Yes	No	Yes	No	No	No	Yes	Yes	No	Yes	Yes	Yes	No	Yes	No	Correct basis?
(e)		0		1				0	1		1	0	0		1		One-time pad
(f)																	Trudy's bases
(g)	x	0	x	1	x	x	x	?	1	x	?	?	0	x	?	x	Trudy's pad

An example of quantum cryptography

# Fundamental Cryptographic Principles

1. Messages must contain some redundancy
2. Some method is needed to foil replay attacks

# Symmetric-key Algorithms (1)



Basic elements of product ciphers.

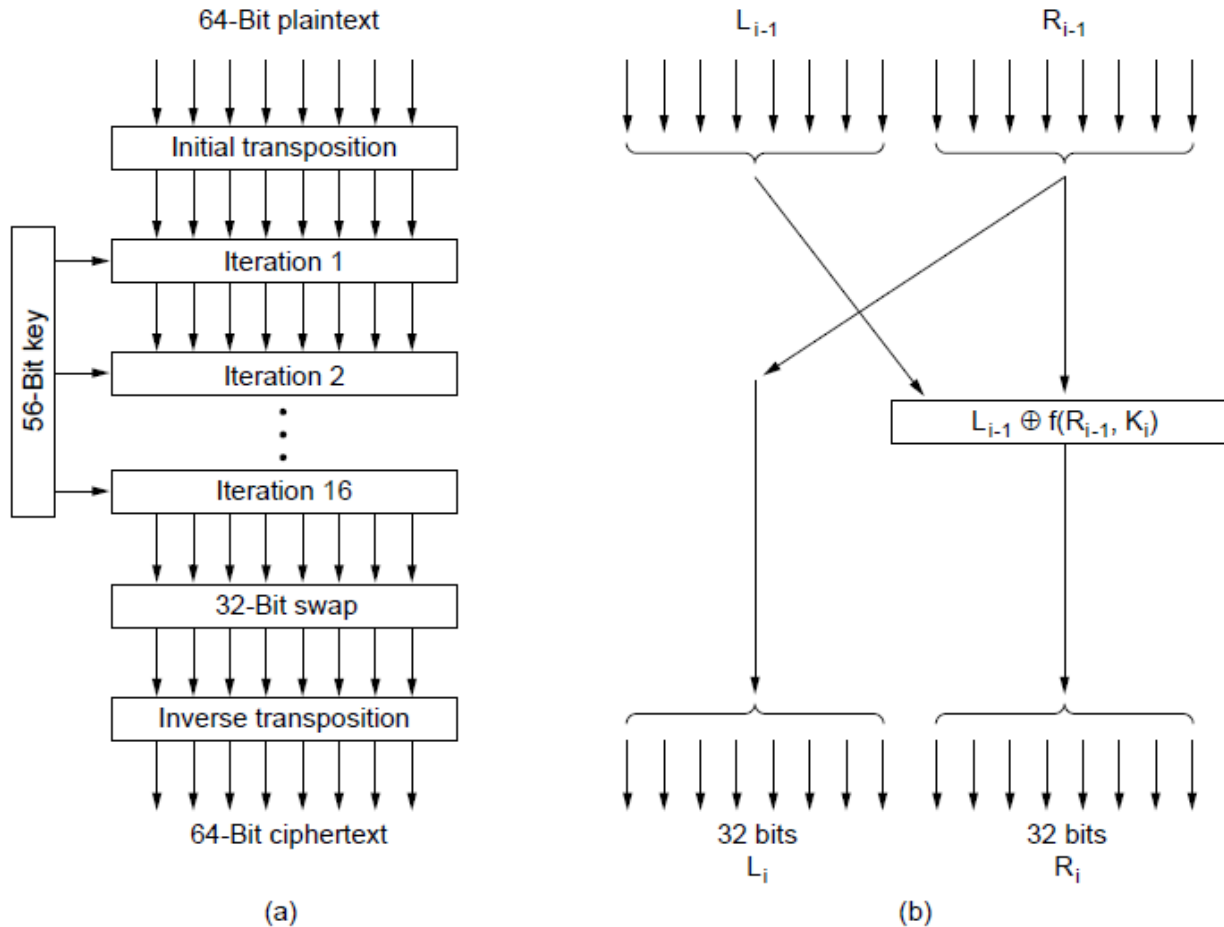
(a) P-box. (b) S-box. (c) Product.



# Symmetric-key Algorithms (2)

- Data encryption standard
- Advanced encryption standard
- Cipher modes
- Other ciphers
- Cryptanalysis

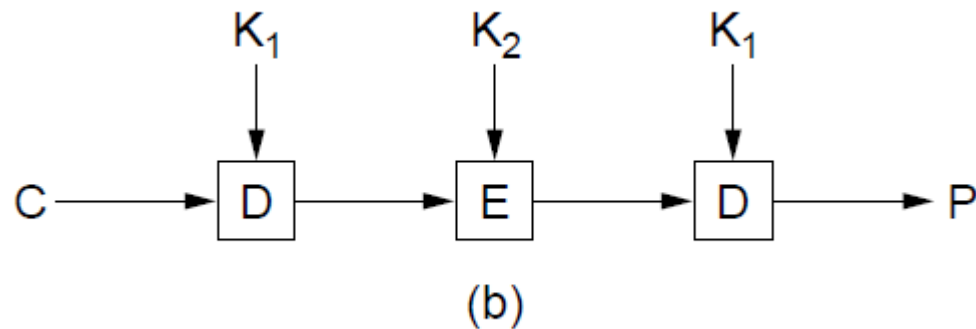
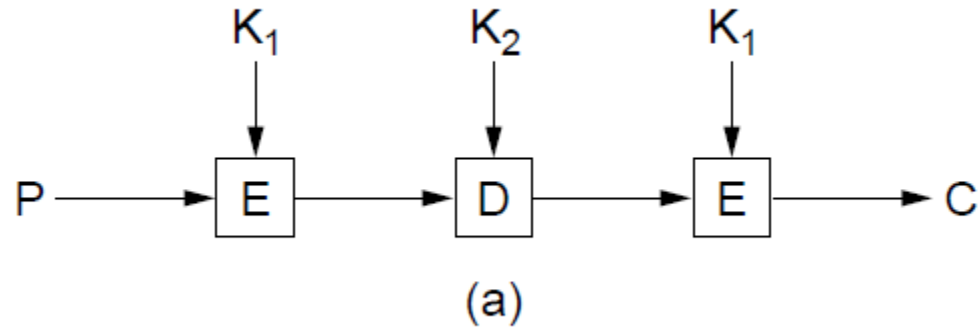
# Data Encryption Standard (1)



The data encryption standard. (a) General outline. (b) Detail of one iteration. The circled + means

exclusive OR

# Data Encryption Standard (2)



(a) Triple encryption using DES. (b) Decryption

# Advanced Encryption Standard (1)

1. Algorithm symmetric block cipher.
2. Full design must be public.
3. Key lengths of 128, 192, and 256 bits supported.
4. Software and hardware implementations possible.
5. Algorithm public or licensed on nondiscriminatory terms.

# Advanced Encryption Standard (2)

```
#define LENGTH 16 /* # bytes in data block or key */
#define NROWS 4 /* number of rows in state */
#define NCOLS 4 /* number of columns in state */
#define ROUNDS 10 /* number of iterations */
typedef unsigned char byte; /* unsigned 8-bit integer */

rijndael(byte plaintext[LENGTH], byte ciphertext[LENGTH], byte key[LENGTH])
{
    int r; /* loop index */
    byte state[NROWS][NCOLS]; /* current state */
    struct {byte k[NROWS][NCOLS];} rk[ROUNDS + 1]; /* round keys */

    expand_key(key, rk); /* construct the round keys */
    copy_plaintext_to_state(state, plaintext); /* init current state */
    xor_roundkey_into_state(state, rk[0]); /* XOR key into state */

    . . .
}
```

An outline of Rijndael

# Advanced Encryption Standard (3)

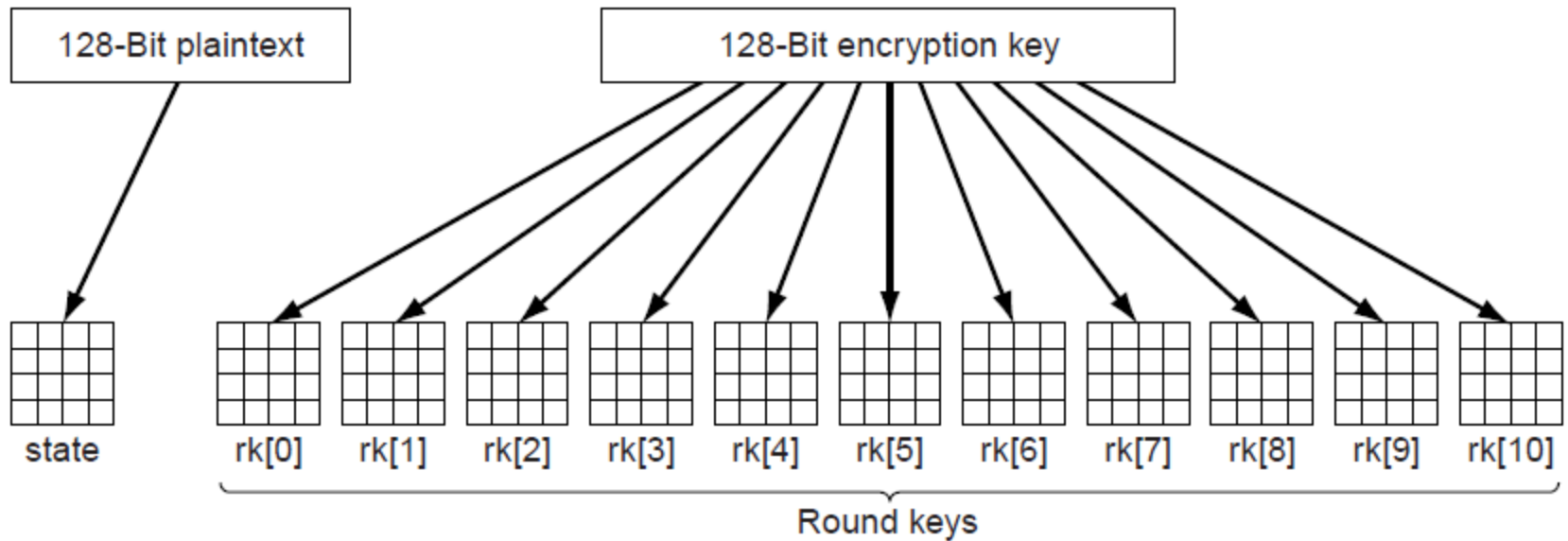
...

```
expand_key(key, rk); /* construct the round keys */
copy_plaintext_to_state(state, plaintext); /* init current state */
xor_roundkey_into_state(state, rk[0]); /* XOR key into state */

for (r = 1; r <= ROUNDS; r++) {
    substitute(state); /* apply S-box to each byte */
    rotate_rows(state); /* rotate row i by i bytes */
    if (r < ROUNDS) mix_columns(state); /* mix function */
    xor_roundkey_into_state(state, rk[r]); /* XOR key into state */
}
copy_state_to_ciphertext(ciphertext, state); /* return result */
}
```

An outline of Rijndael

# Advanced Encryption Standard (4)



Creating of the *state* and *rk* arrays

# Cipher Modes (1)

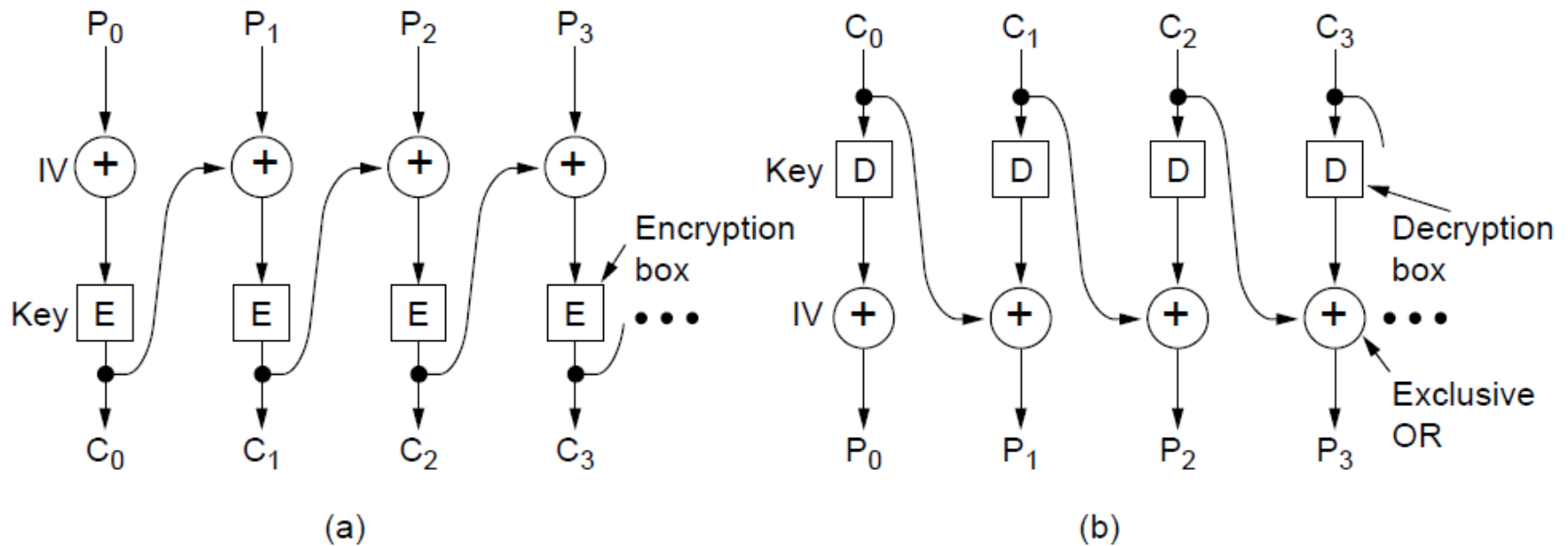
Name	Position	Bonus
A d a m s , L e s l i e	C l e r k	\$ 1 0
B l a c k , R o b i n	B o s s	\$ 5 0 0 , 0 0 0
C o l l i n s , K i m	M a n a g e r	\$ 1 0 0 , 0 0 0
D a v i s , B o b b i e	J a n i t o r	\$ 5

← 16      ← 8      ← 8 →

The plaintext of a file encrypted as 16 DES blocks.

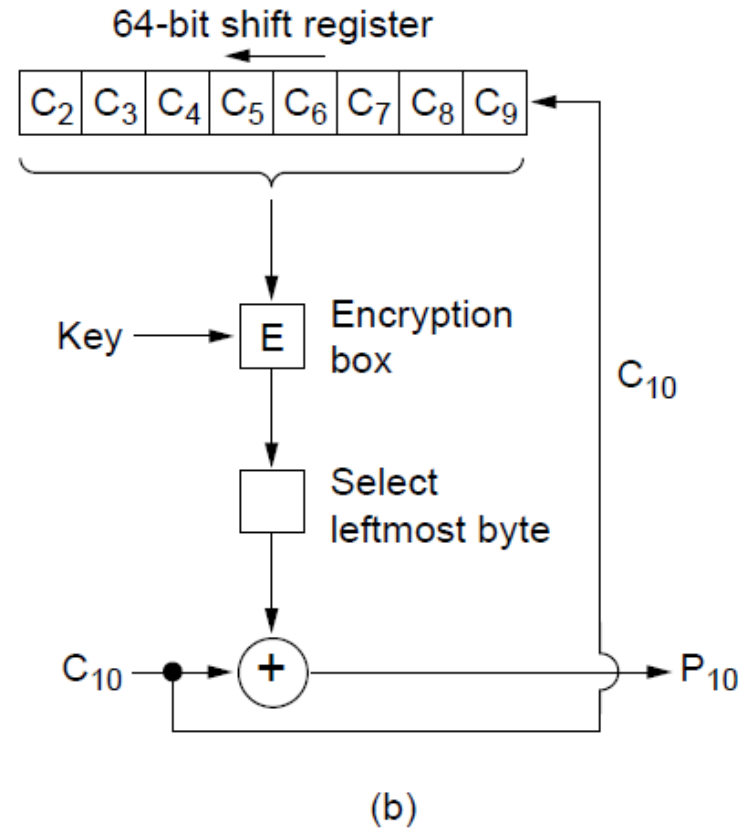
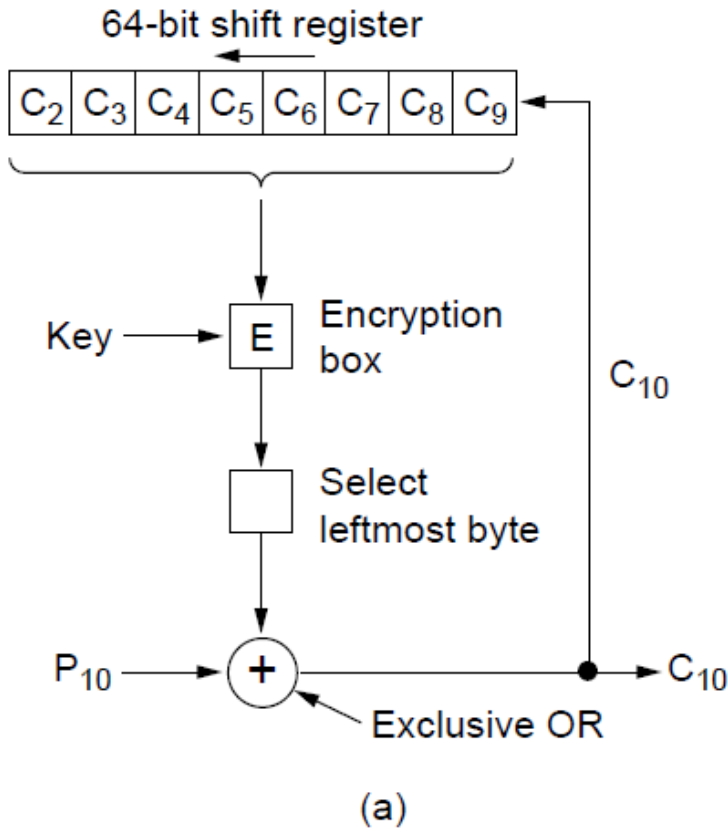


# Cipher Modes (2)



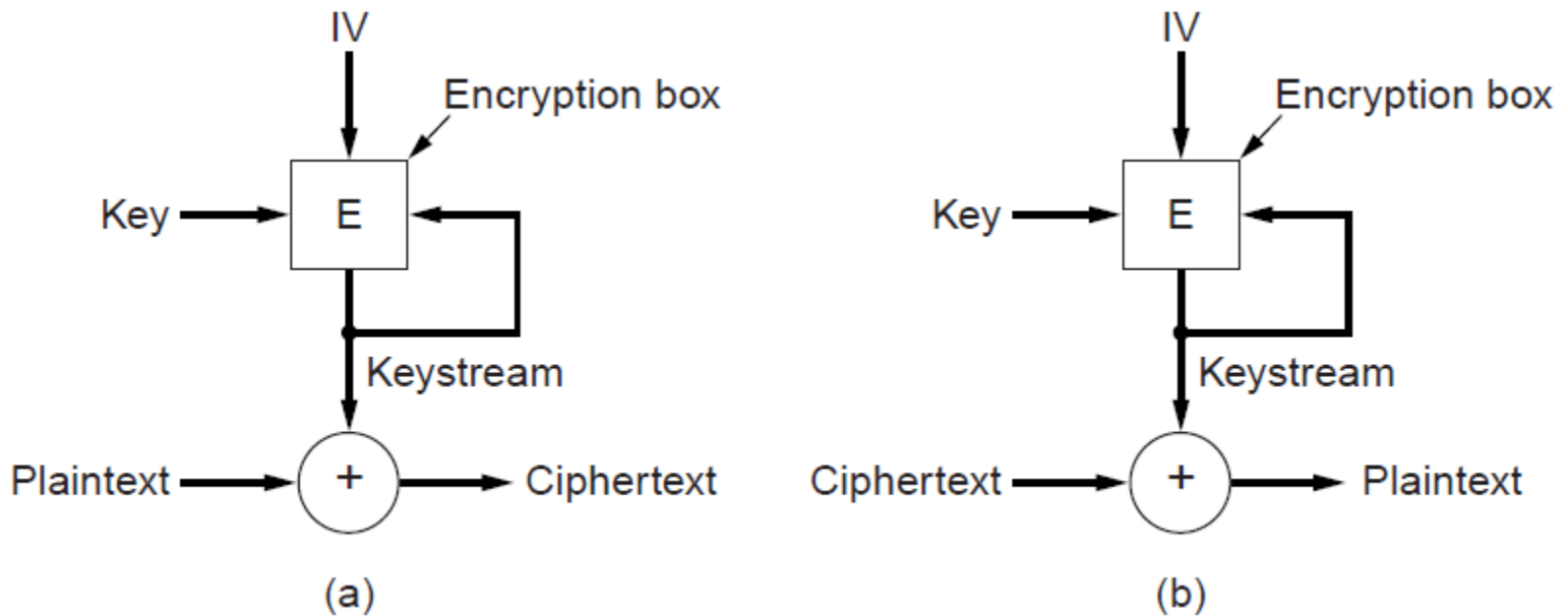
Cipher block chaining. (a) Encryption. (b) Decryption

# Cipher Modes (3)



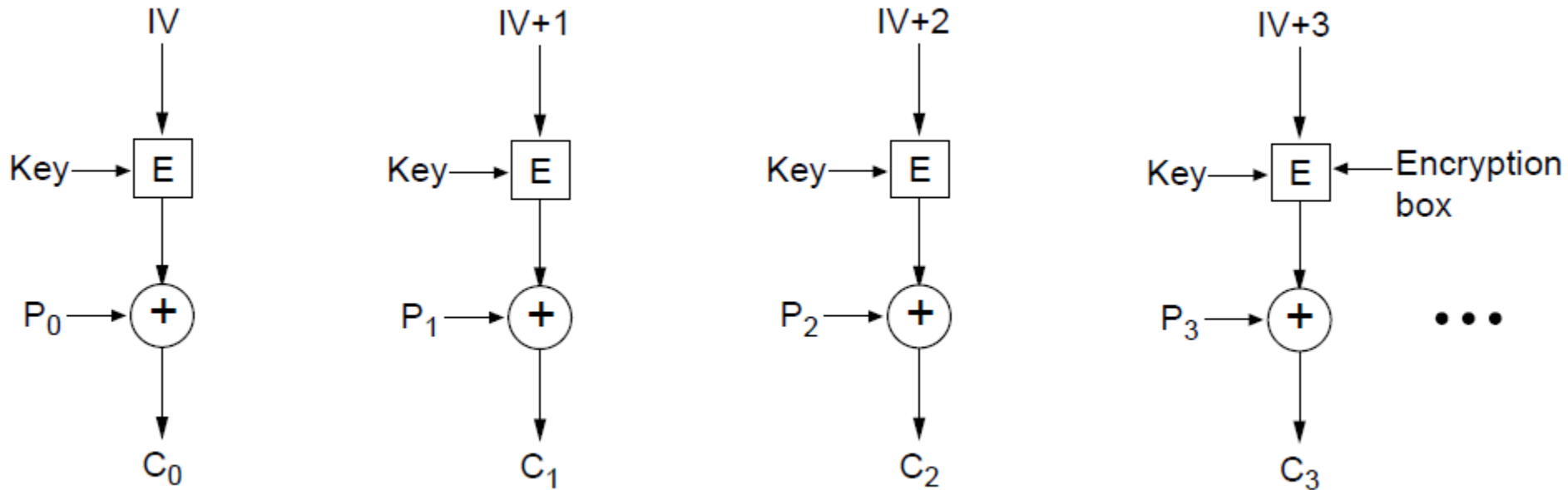
Cipher feedback mode. (a) Encryption. (b) Decryption

# Cipher Modes (4)



A stream cipher. (a) Encryption. (b) Decryption

# Cipher Modes (5)



Encryption using counter mode

# Other Ciphers

<b>Cipher</b>	<b>Author</b>	<b>Key length</b>	<b>Comments</b>
Blowfish	Bruce Schneier	1–448 bits	Old and slow
DES	IBM	56 bits	Too weak to use now
IDEA	Massey and Xuejia	128 bits	Good, but patented
RC4	Ronald Rivest	1–2048 bits	Caution: some keys are weak
RC5	Ronald Rivest	128–256 bits	Good, but patented
Rijndael	Daemen and Rijmen	128–256 bits	Best choice
Serpent	Anderson, Biham, Knudsen	128–256 bits	Very strong
Triple DES	IBM	168 bits	Second best choice
Twofish	Bruce Schneier	128–256 bits	Very strong; widely used

Some common symmetric-key cryptographic algorithms

# Public-key Algorithms

- RSA
  - Authors: *Rivest*, Shamir, Adleman
- Other Public-Key Algorithms

# RSA (1)

## Method Summary

1. Choose two large primes,  $p$  and  $q$
2. Compute  
 $n = p \times q$  and  $z = (p - 1) \times (q - 1)$ .
3. Choose number relatively prime to  $z$   
call it  $d$ .
4. Find  $e$  such that  $e \times d = 1 \pmod{z}$ .

# RSA (2)

Plaintext (P)		Ciphertext (C)			After decryption	
Symbolic	Numeric	$P^3$	$P^3 \pmod{33}$	$C^7$	$C^7 \pmod{33}$	Symbolic
S	19	6859	28	13492928512	19	S
U	21	9261	21	1801088541	21	U
Z	26	17576	20	1280000000	26	Z
A	01	1	1	1	01	A
N	14	2744	5	78125	14	N
N	14	2744	5	78125	14	N
E	05	125	26	8031810176	05	E

Sender's computation
Receiver's computation

An example of the RSA algorithm



# Digital Signatures (1)

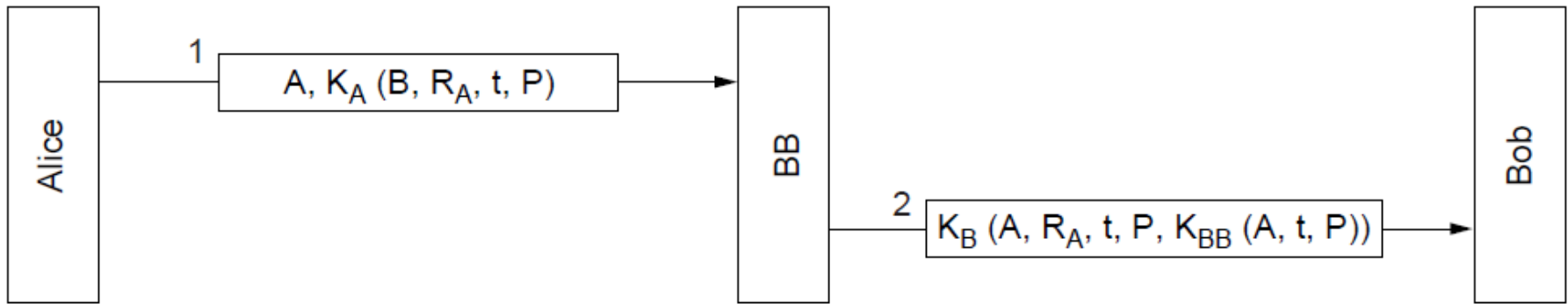
Required Conditions:

1. Receiver can verify claimed identity of sender.
2. Sender cannot later repudiate contents of message.
3. Receiver cannot have concocted message himself.

# Digital Signatures (2)

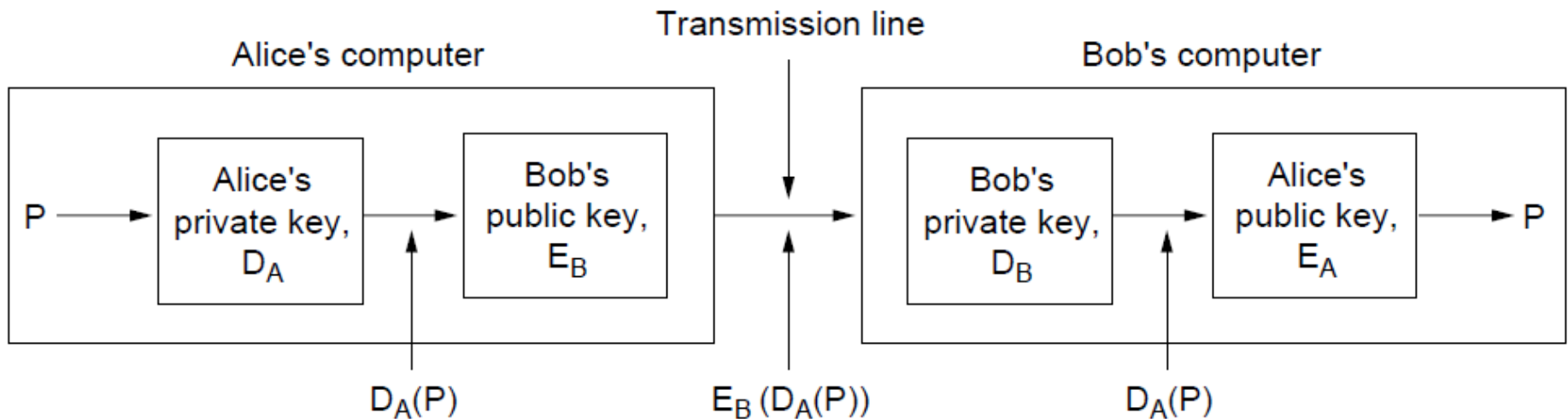
- Symmetric-key signatures
- Public-key signatures
- Message digests
- The birthday attack

# Symmetric-key Signatures



Digital signatures with Big Brother

# Public-Key Signatures (1)



Digital signatures using public-key cryptography.

# Public-Key Signatures (2)

## Criticisms of DSS:

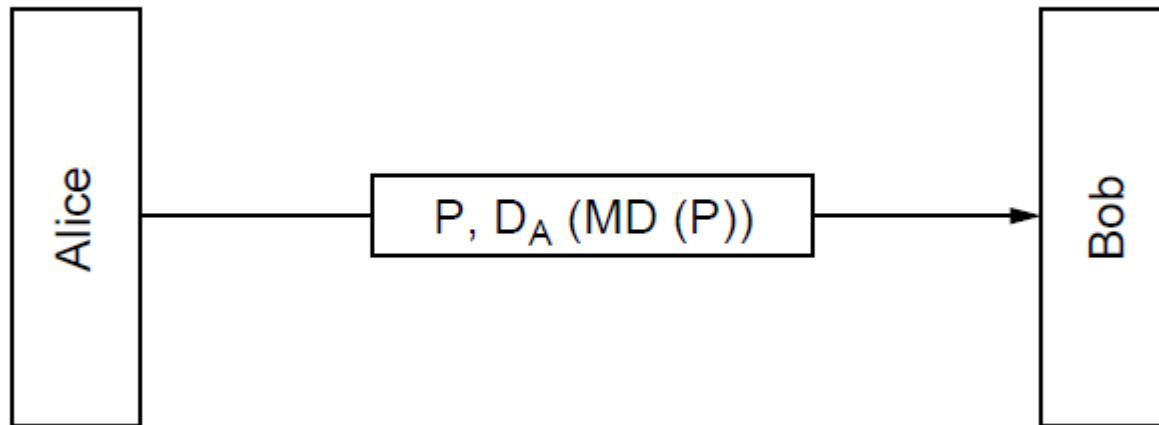
1. Too secret
2. Too slow
3. Too new
4. Too insecure

# Message Digests (1)

## Message Digest properties

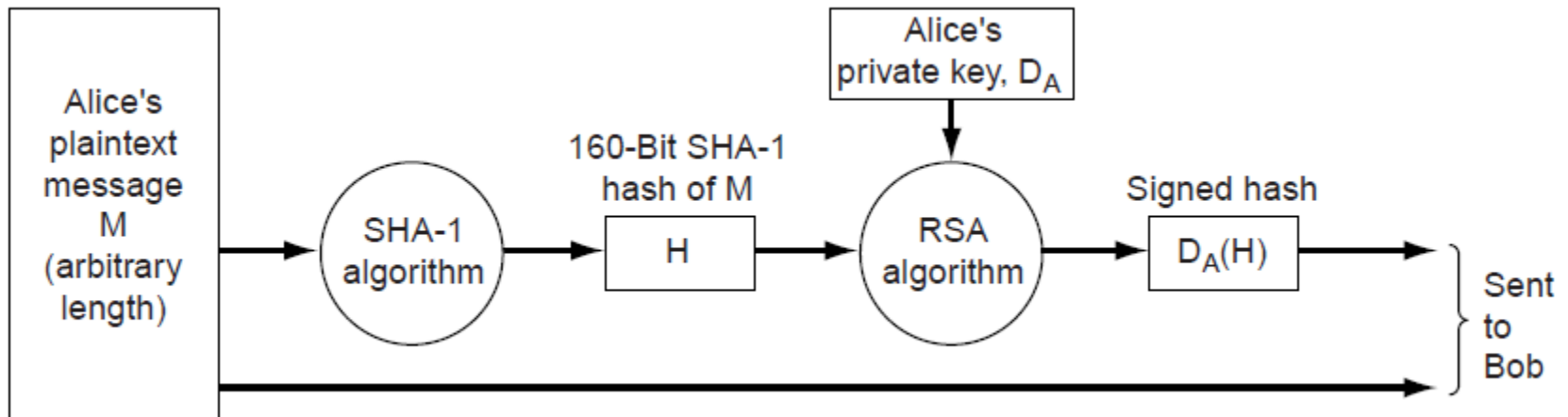
1. Given  $P$ , easy to compute  $MD(P)$ .
2. Given  $MD(P)$ , effectively impossible to find  $P$ .
3. Given  $P$  no one can find  $P'$  such that  $MD(P') = MD(P)$ .
4. Change to input of even 1 bit produces very different output.

# Message Digests (2)



Digital signatures using message digests

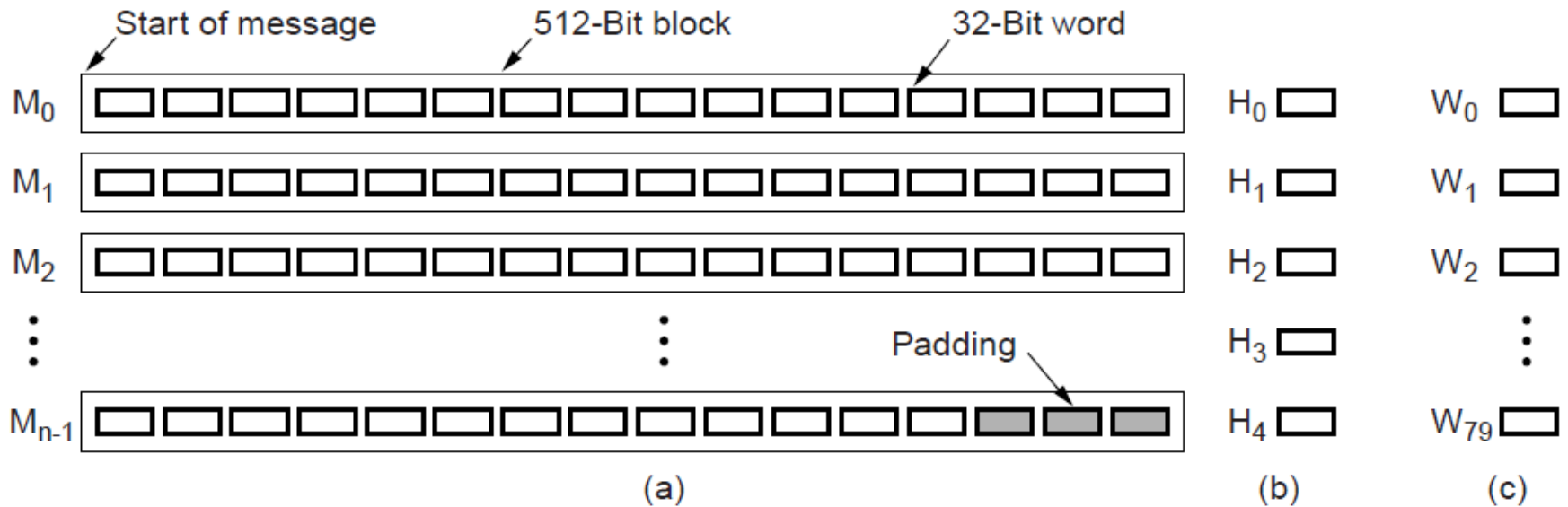
# Message Digests (3)



Use of SHA-1 and RSA for signing nonsecret messages



# Message Digests (4)

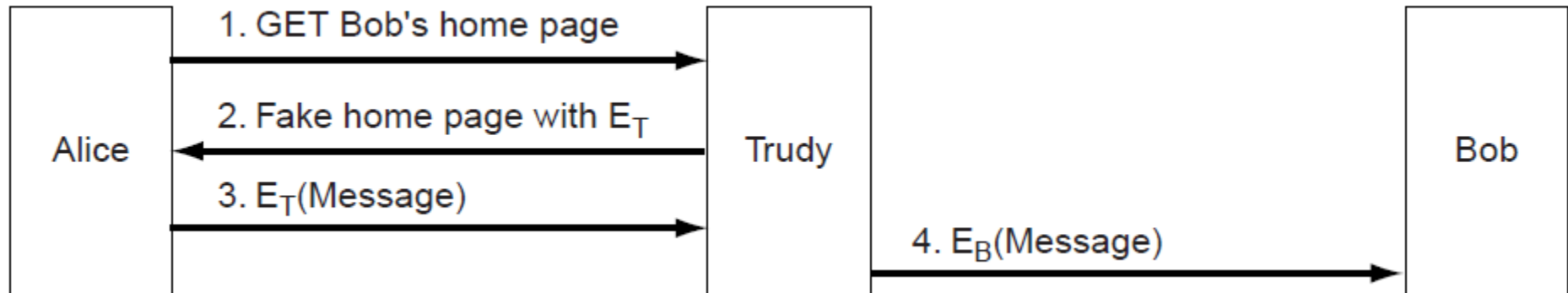


(a) A message padded out to a multiple of 512 bits.

(b) The output variables.

(c) The word array.

# Management of Public Keys (1)



A way for Trudy to subvert public-key encryption

# Management of Public Keys (2)

- Certificates
- X.509
- Public key infrastructures

# Certificates

I hereby certify that the public key

19836A8B03030CF83737E3837837FC3s87092827262643FFA82710382828282A

belongs to

Robert John Smith

12345 University Avenue

Berkeley, CA 94702

Birthday: July 4, 1958

Email: bob@superdupernet.com

SHA-1 hash of the above certificate signed with the CA's private key

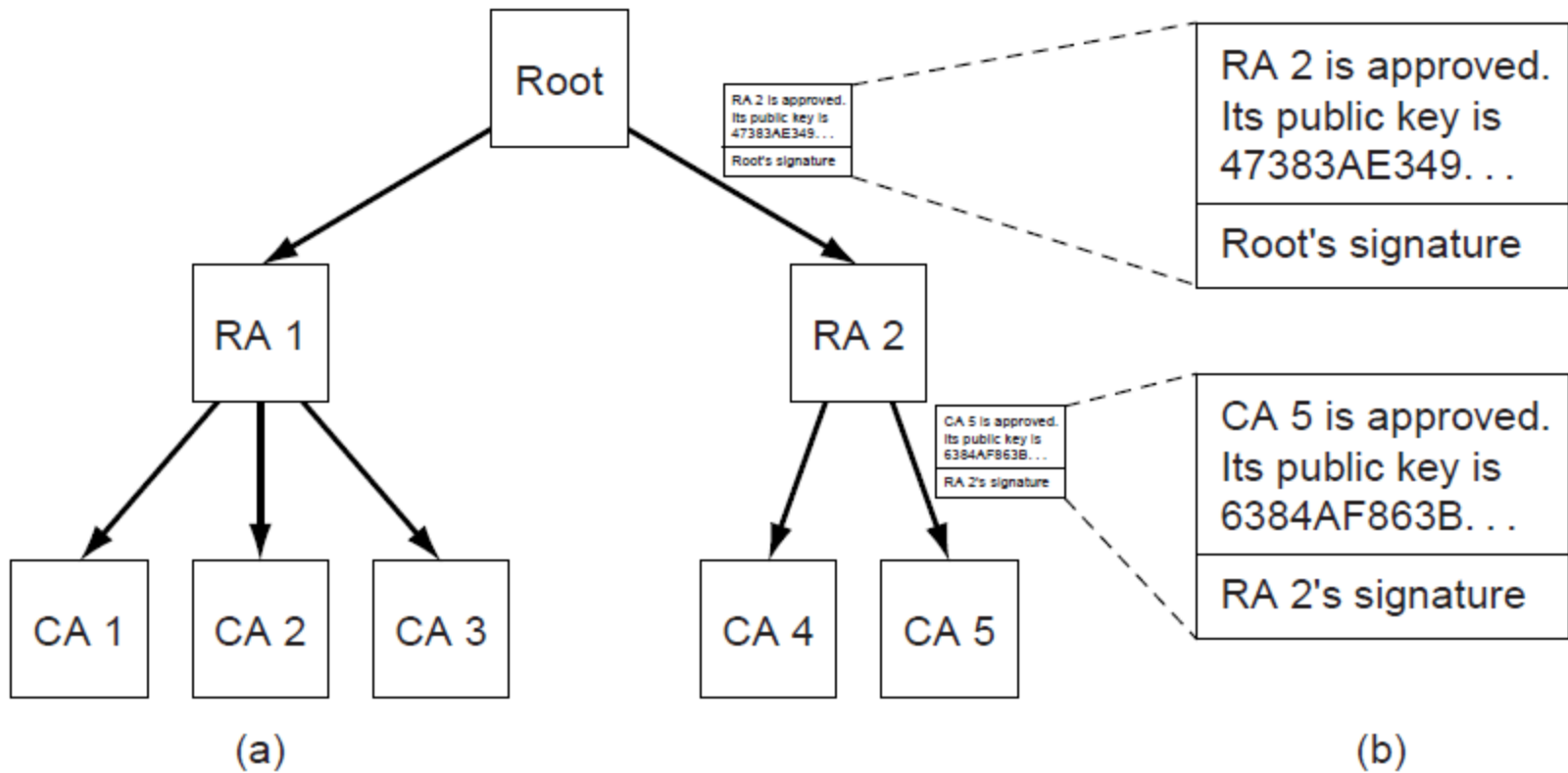
A possible certificate and its signed hash

# X.509

<b>Field</b>	<b>Meaning</b>
Version	Which version of X.509
Serial number	This number plus the CA's name uniquely identifies the certificate
Signature algorithm	The algorithm used to sign the certificate
Issuer	X.500 name of the CA
Validity period	The starting and ending times of the validity period
Subject name	The entity whose key is being certified
Public key	The subject's public key and the ID of the algorithm using it
Issuer ID	An optional ID uniquely identifying the certificate's issuer
Subject ID	An optional ID uniquely identifying the certificate's subject
Extensions	Many extensions have been defined
Signature	The certificate's signature (signed by the CA's private key)

The basic fields of an X.509 certificate

# Public Key Infrastructures

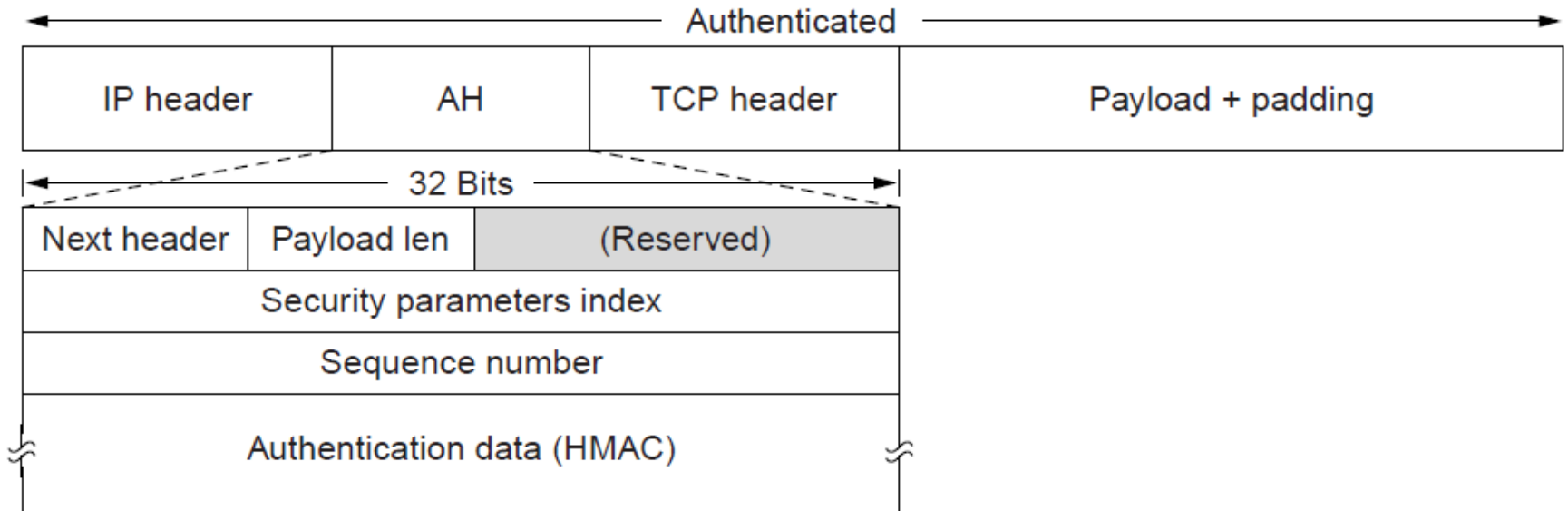


(a) A hierarchical PKI. (b) A chain of certificates.

# Communication Security

- IPsec
- Firewalls
- Virtual private networks
- Wireless security

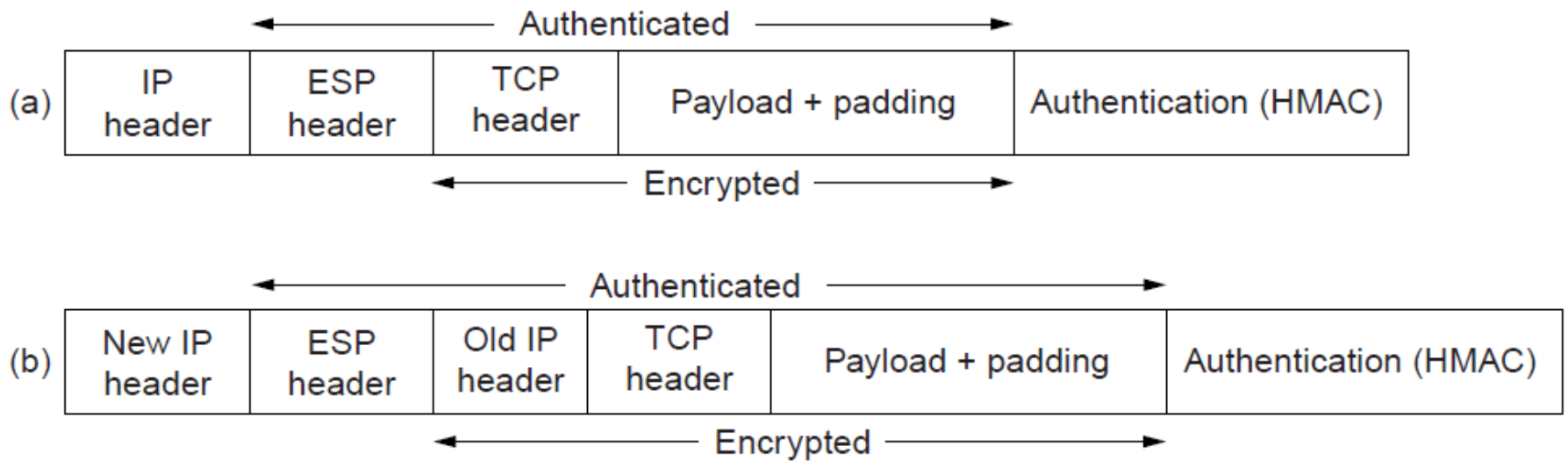
# IPsec (1)



The IPsec authentication header in transport mode for IPv4.

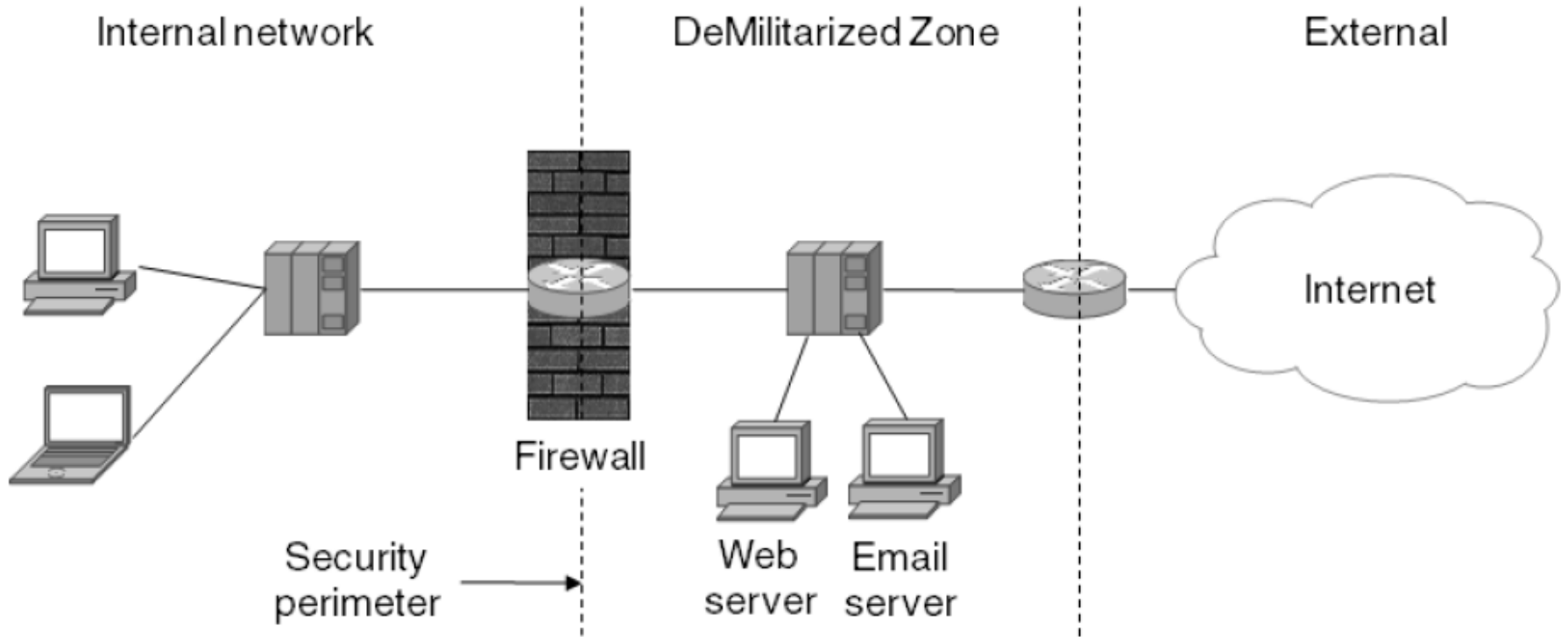


# IPsec (2)

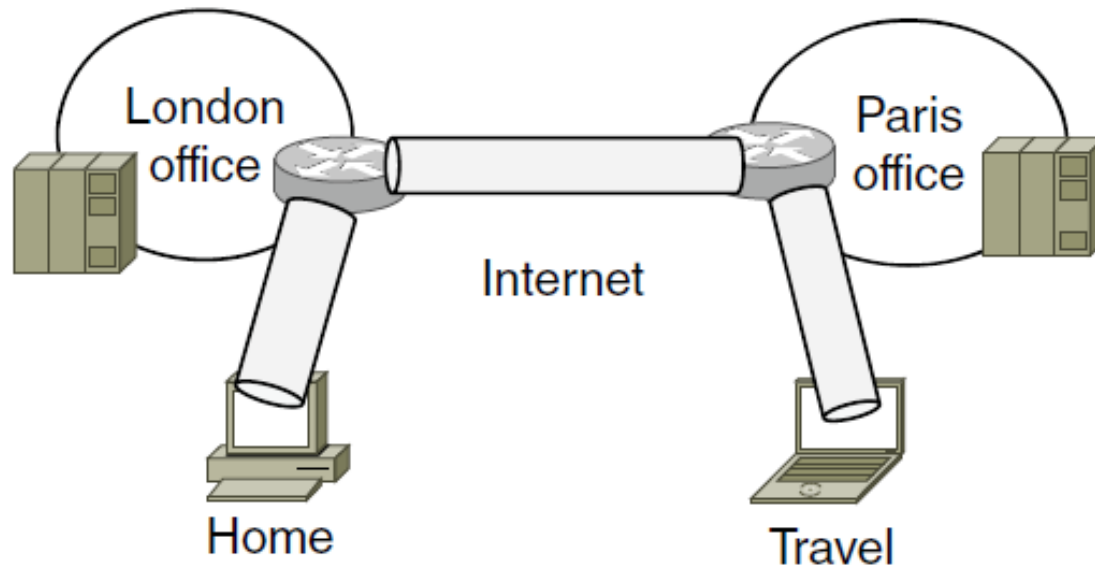


**(a)** ESP in transport mode. **(b)** ESP in tunnel mode.

# IPsec (3)

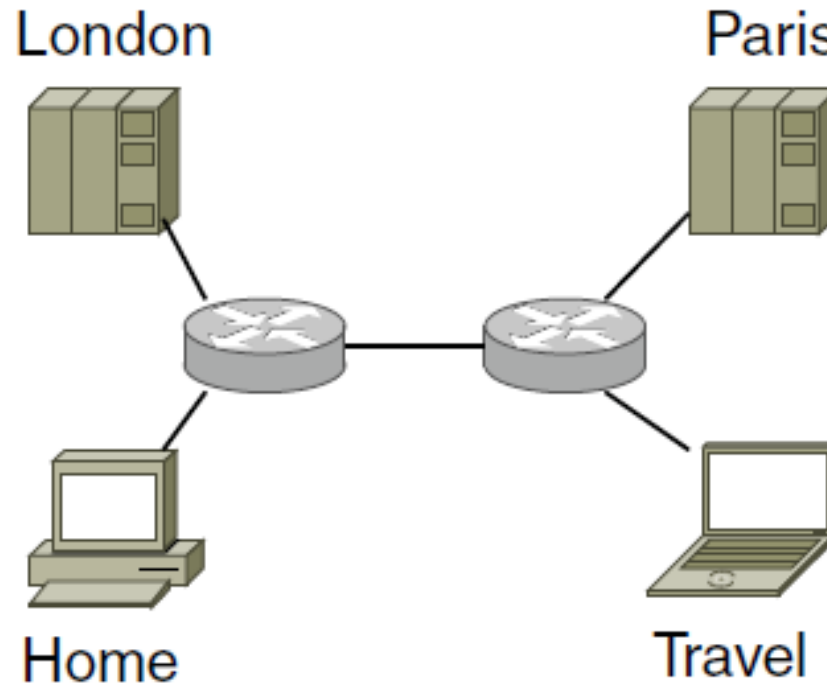


# Virtual Private Networks (1)



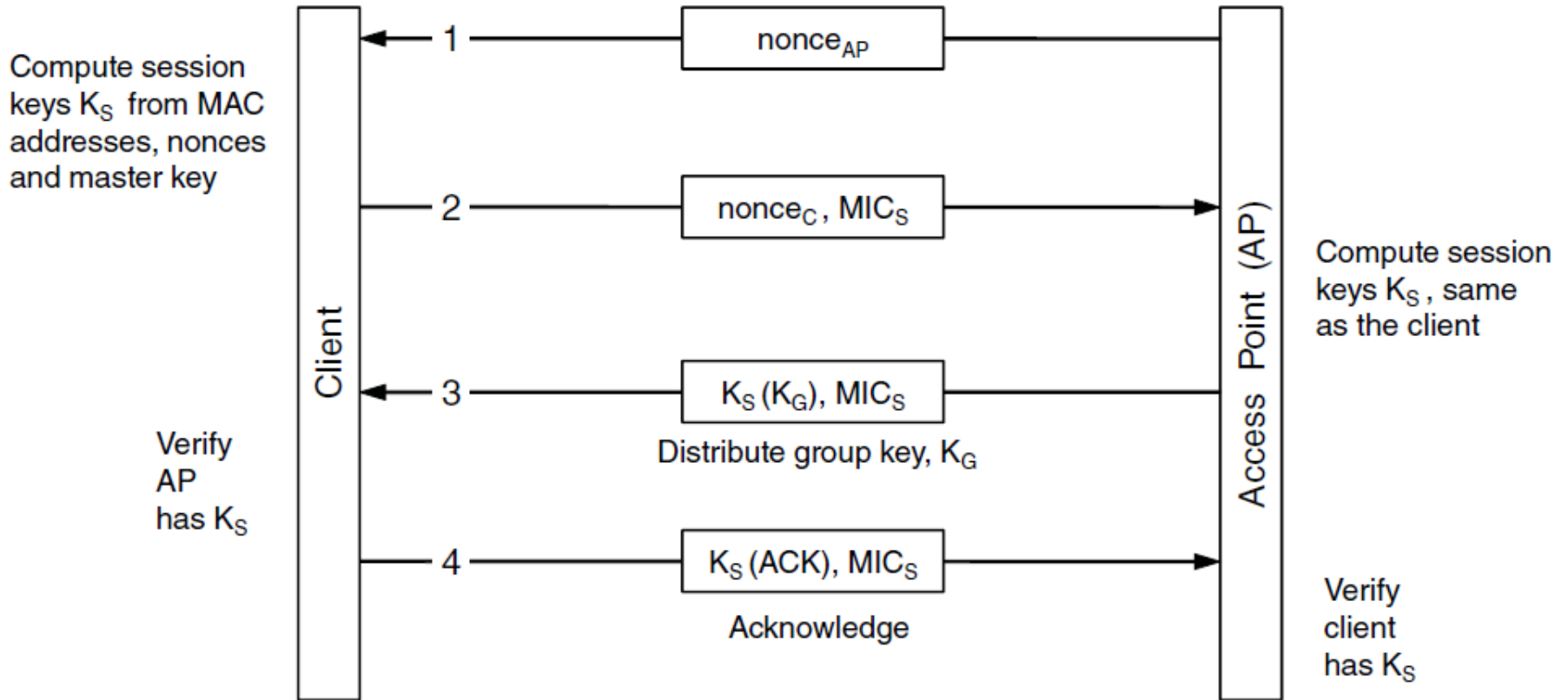
A virtual private network

# Virtual Private Networks (2)



Topology as seen from the inside

# Wireless Security



The 802.11i key setup handshake

# Authentication Protocols

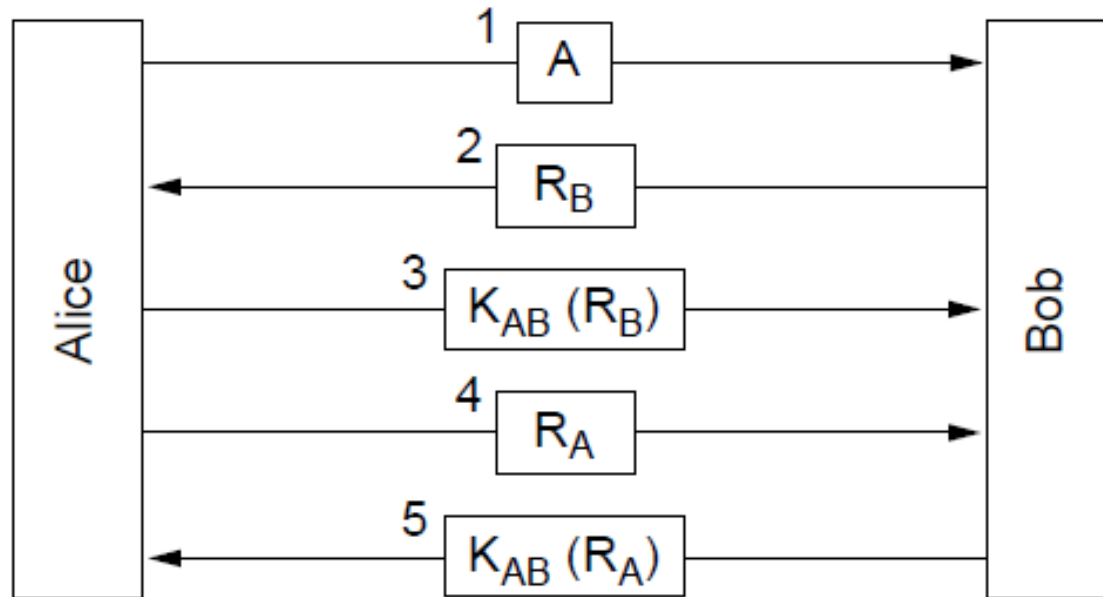
- Shared secret key
- Establishing a shared key:  
the Diffie-Hellman key exchange
- Key distribution center
- Kerberos
- Public-key cryptography

# Shared Secret Key (1)

Notation for discussing protocols

- $A, B$  are the identities of Alice and Bob.
- $R_i$ 's are the challenges, where the subscript identifies the challenger.
- $K_i$  are keys, where  $i$  indicates the owner.
- $K_S$  is the session key.

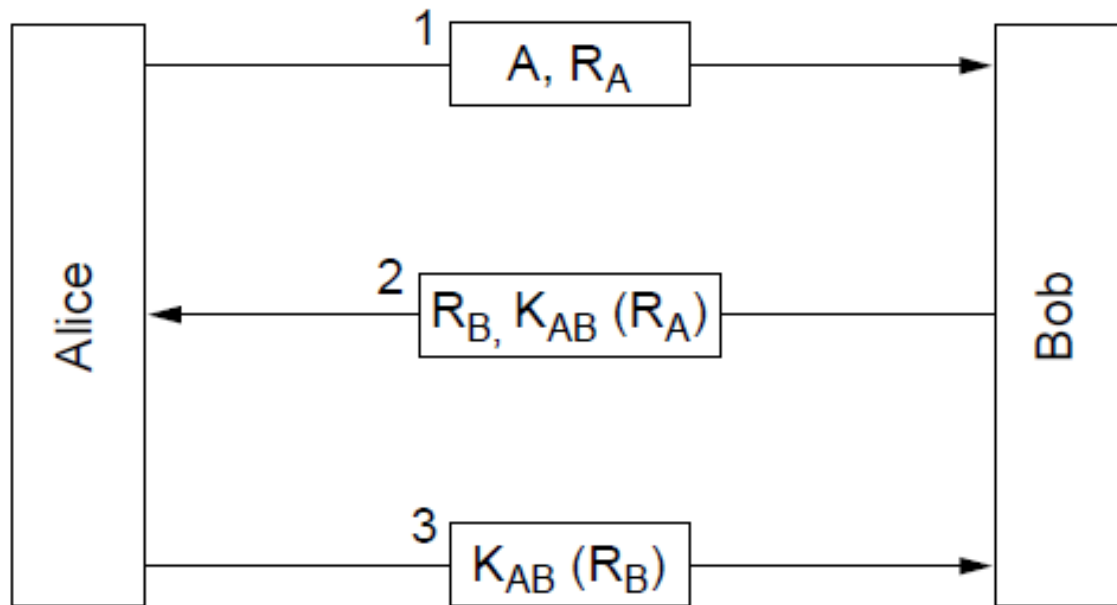
# Shared Secret Key (2)



Two-way authentication using a challenge-response protocol.

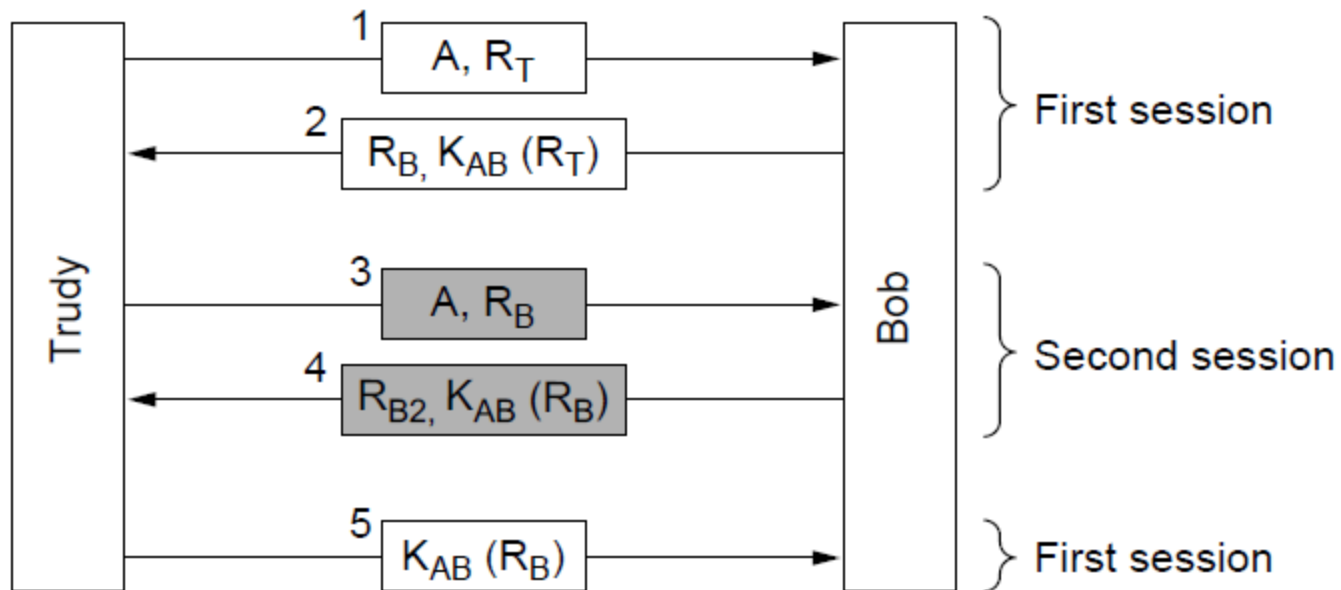


# Shared Secret Key (3)



A shortened two-way authentication protocol

# Shared Secret Key (4)



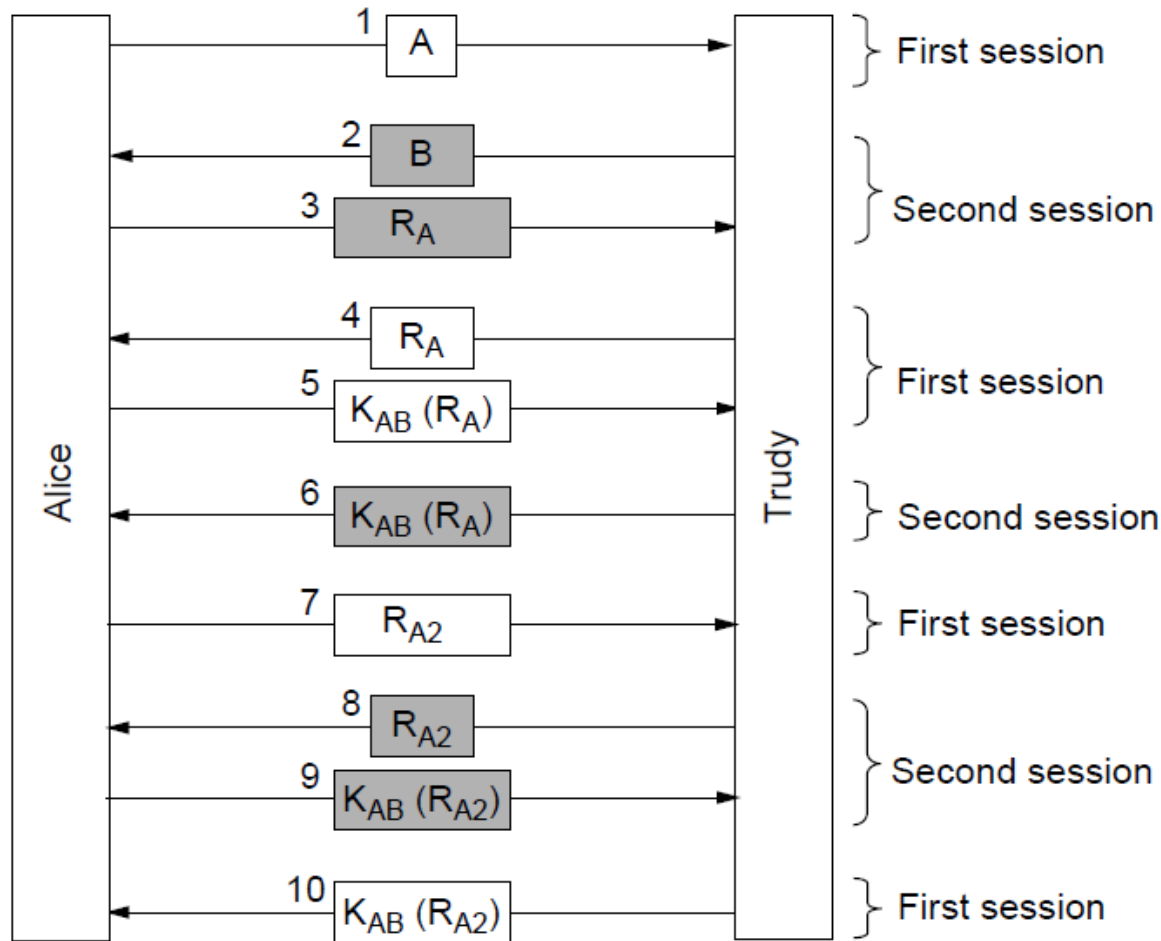
The reflection attack.

# Shared Secret Key (5)

## General design rules

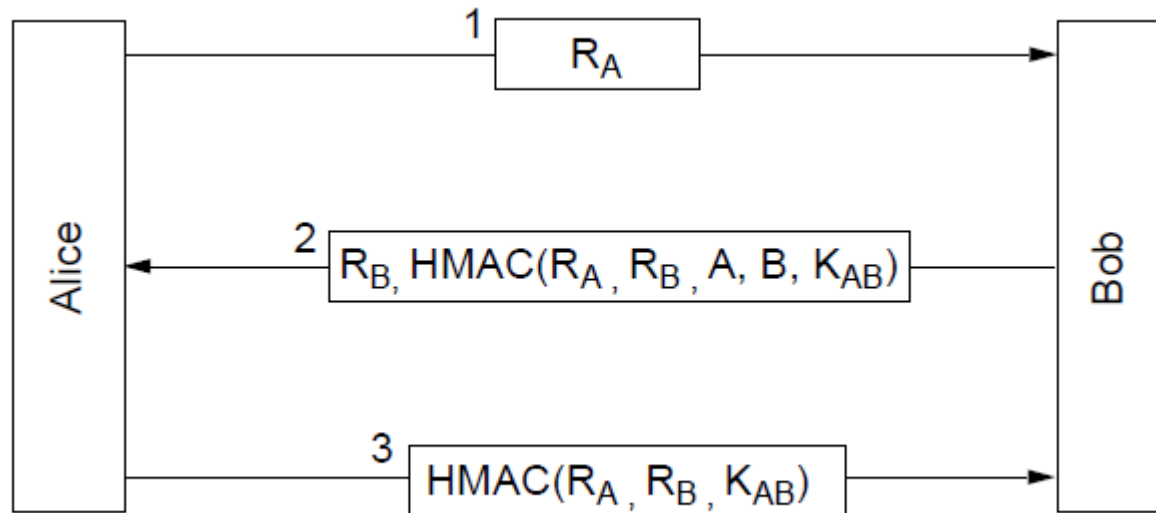
1. Have initiator prove who she is before responder
2. Initiator, responder use different keys
3. Draw challenges from different sets
4. Make protocol resistant to attacks involving second parallel session

# Shared Secret Key (6)



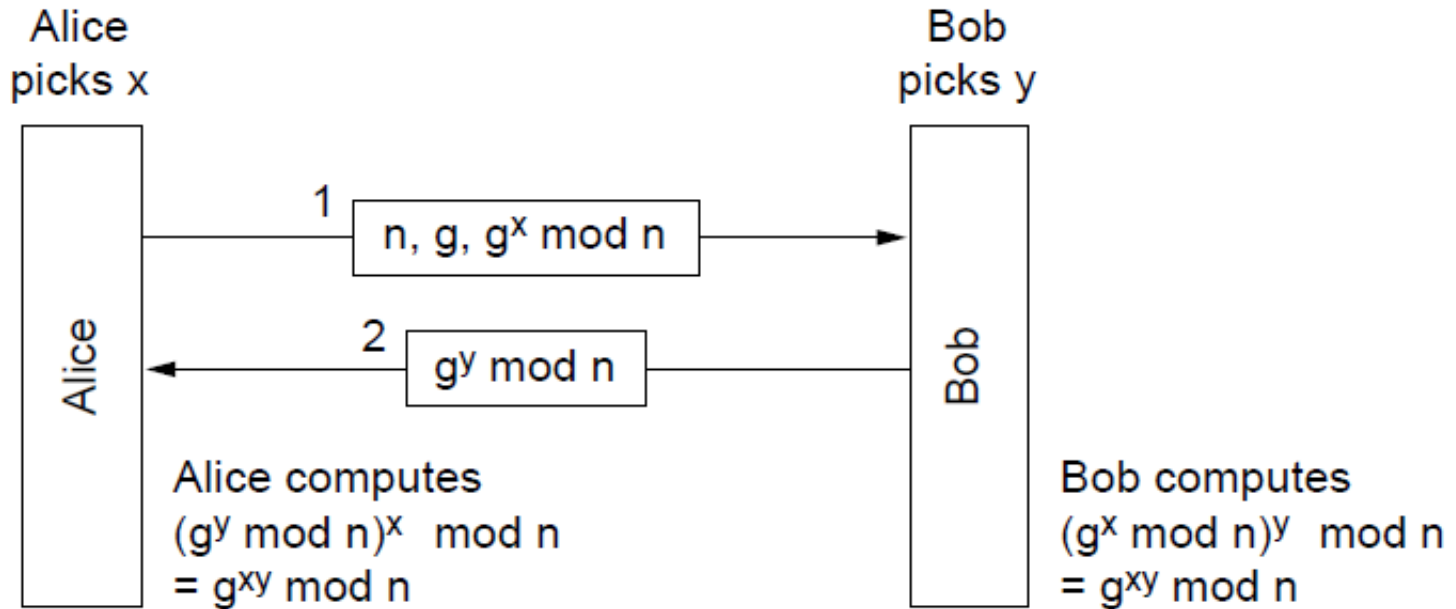
A reflection attack on the protocol of [Fig. 8-32](#)

# Shared Secret Key (7)



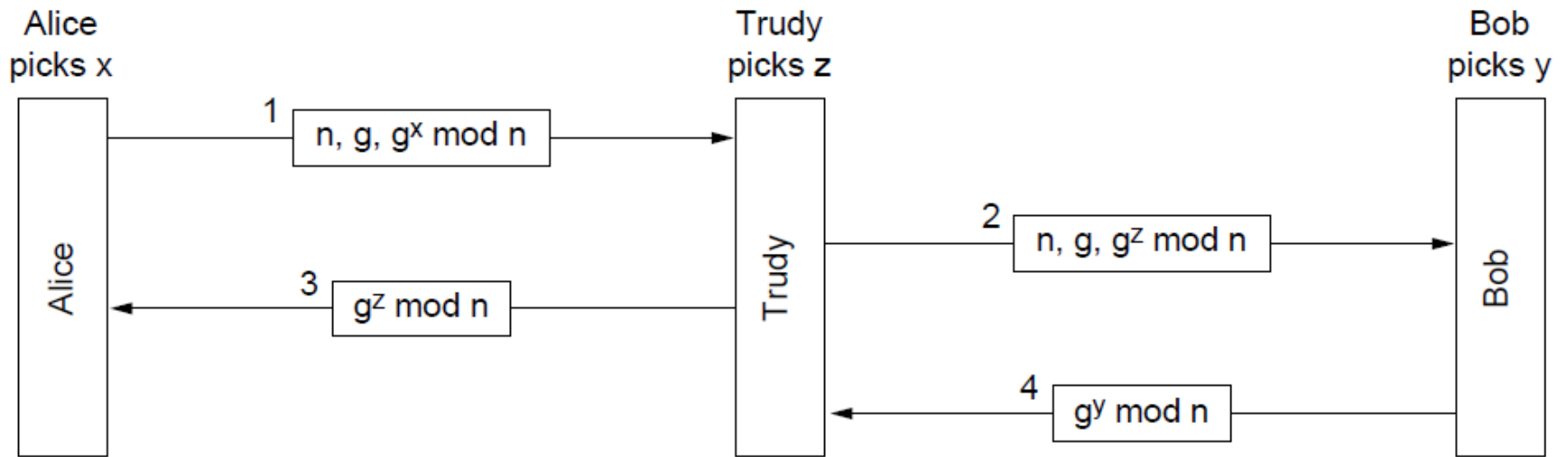
Authentication using HMACs

# The Diffie-Hellman Key Exchange (1)



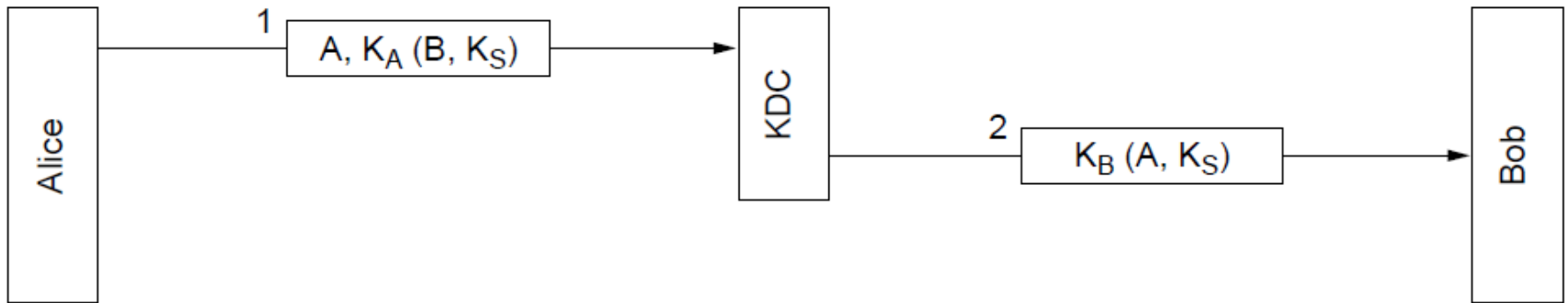
The Diffie-Hellman key exchange

# The Diffie-Hellman Key Exchange (2)



The man-in-the-middle attack

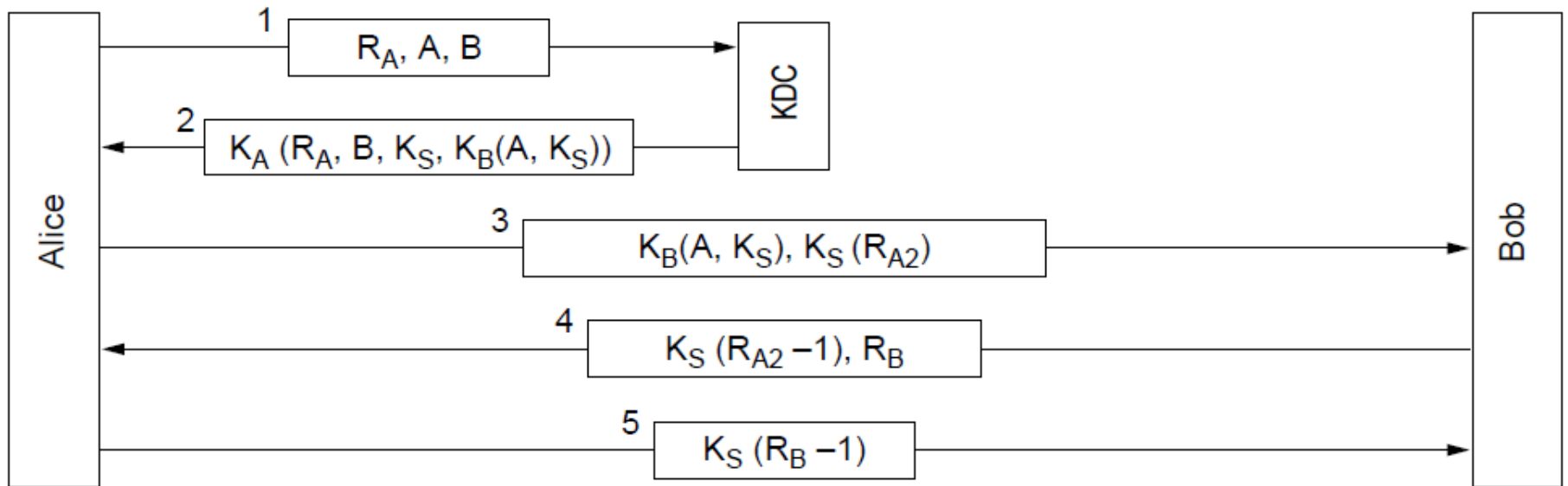
# Key Distribution Center (1)



A first attempt at an authentication protocol using a KDC.

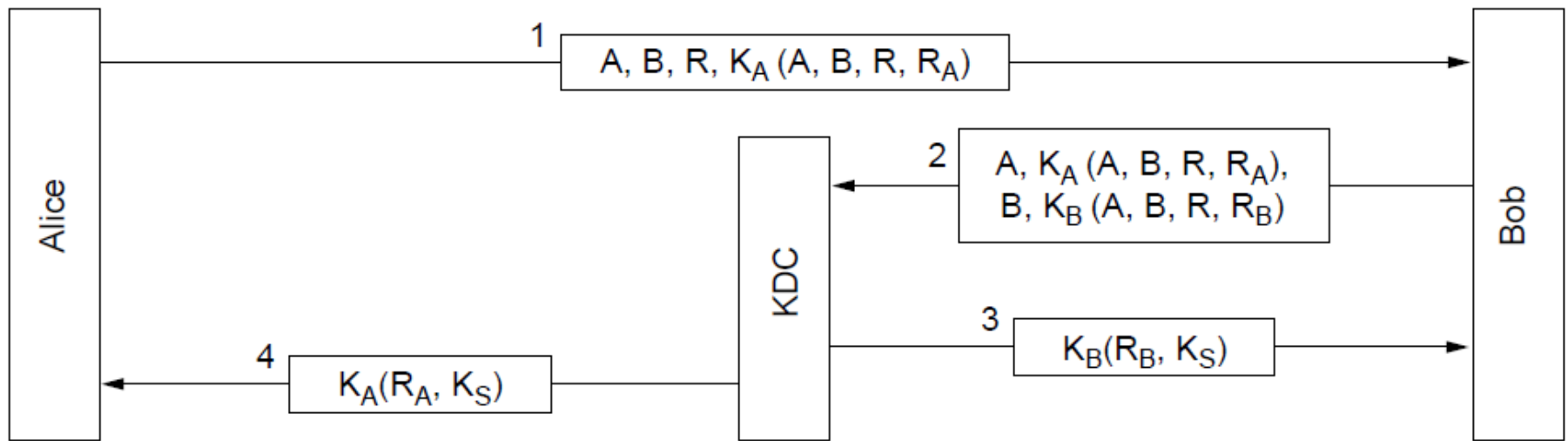


# Key Distribution Center (2)



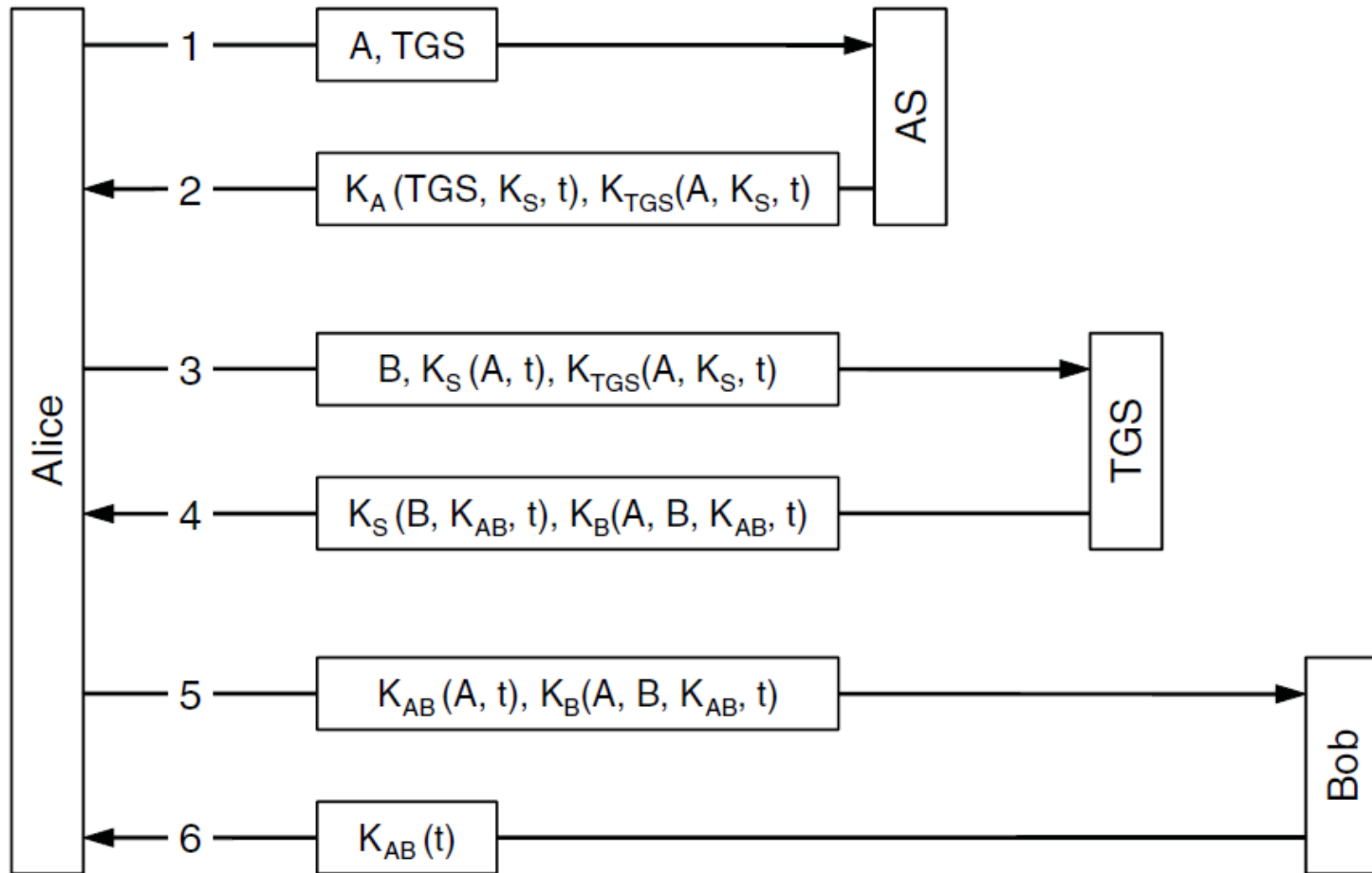
The Needham-Schroeder authentication protocol

# Key Distribution Center (3)



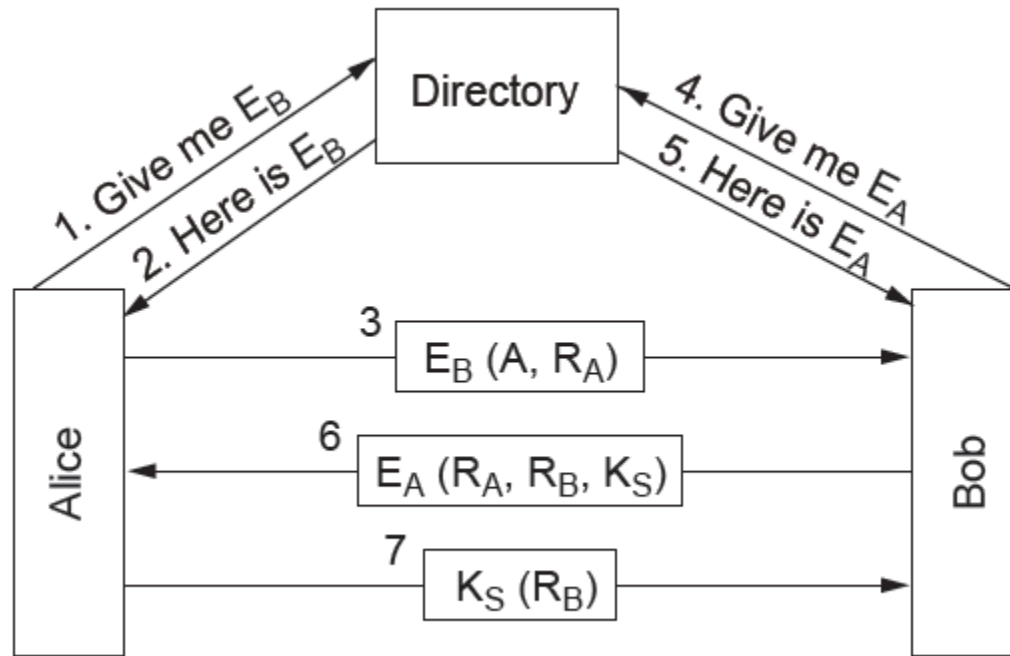
The Otway-Rees authentication protocol (slightly simplified).

# Kerberos



The operation of Kerberos V5

# Public-Key Cryptography



Mutual authentication using public-key cryptography

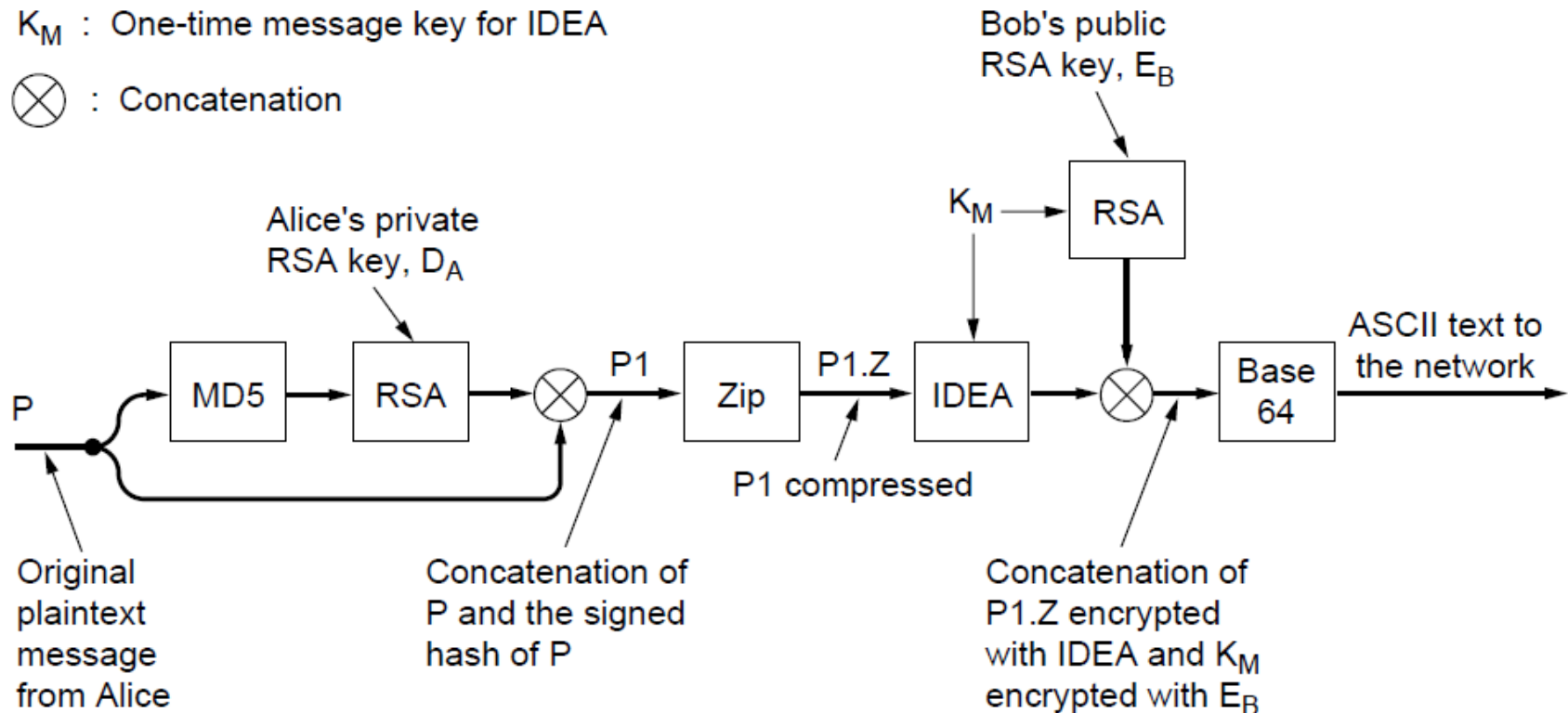
# Email Security

- PGP—Pretty Good Privacy
- S/MIME

# PGP—Pretty Good Privacy (1)

$K_M$  : One-time message key for IDEA

⊗ : Concatenation

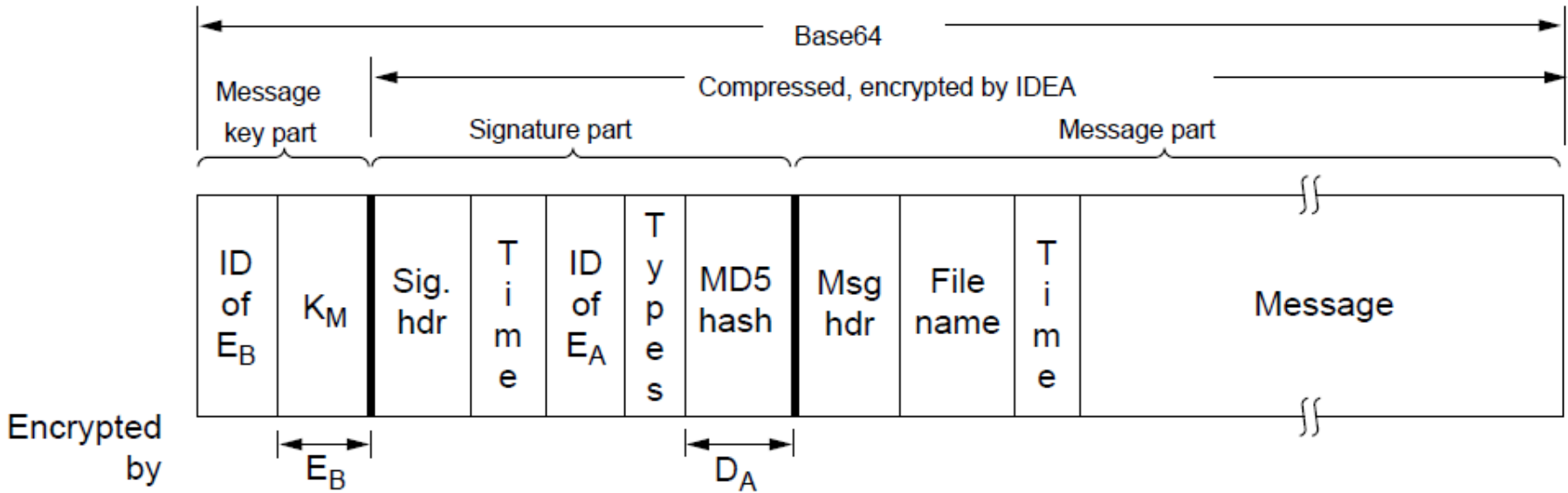


PGP in operation for sending a message

# PGP—Pretty Good Privacy (2)

- Casual (384 bits):
  - Can be broken easily today.
- Commercial (512 bits):
  - Breakable by three-letter organizations.
- Military (1024 bits):
  - Not breakable by anyone on earth.
- Alien (2048 bits):
  - Unbreakable by anyone on other planets

# PGP—Pretty Good Privacy (3)



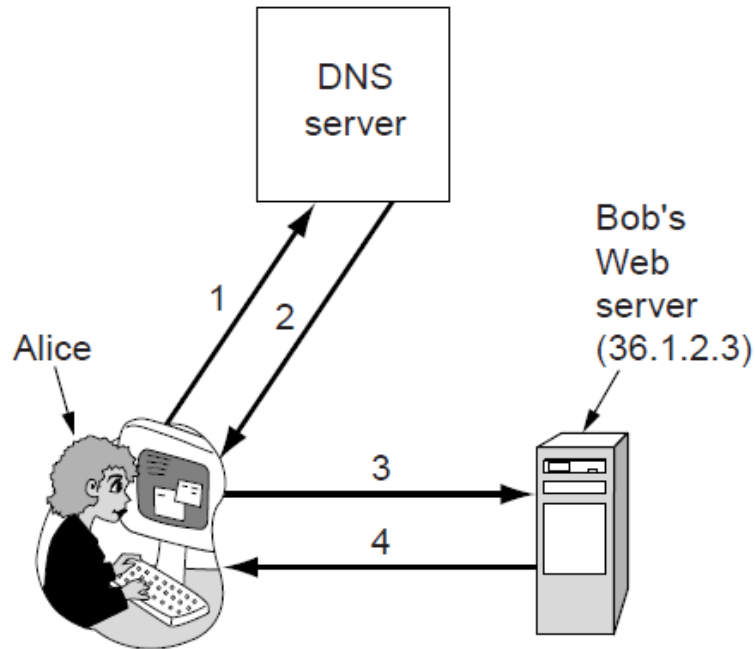
A PGP message



# Web Security

- Threats
- Secure naming
- SSL—the Secure Sockets Layer
- Mobile code security

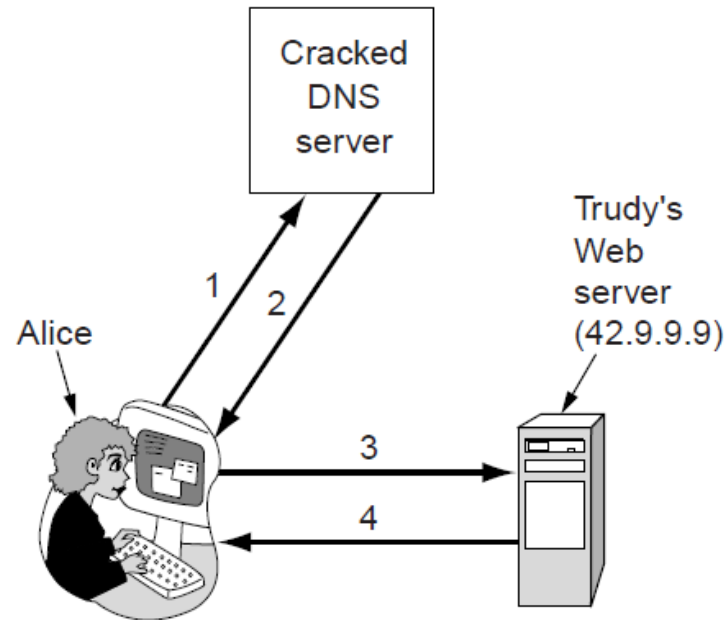
# Secure Naming (1)



1. Give me Bob's IP address
2. 36.1.2.3 (Bob's IP address)
3. GET index.html
4. Bob's home page

Normal situation

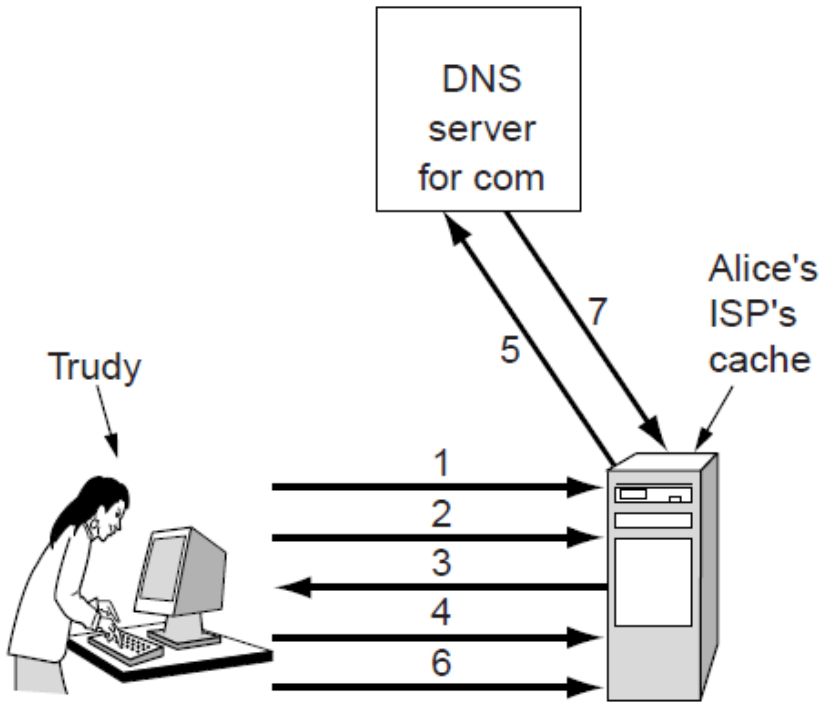
# Secure Naming (2)



1. Give me Bob's IP address
2. 42.9.9.9 (Trudy's IP address)
3. GET index.html
4. Trudy's fake of Bob's home page

An attack based on breaking into DNS and modifying Bob's record.

# Secure Naming (3)



1. Look up foobar.trudy-the-intruder.com (to force it into the ISP's cache)
2. Look up www.trudy-the-intruder.com (to get the ISP's next sequence number)
3. Request for www.trudy-the-intruder.com (Carrying the ISP's next sequence number,  $n$ )
4. Quick like a bunny, look up bob.com (to force the ISP to query the com server in step 5)
5. Legitimate query for bob.com with  $\text{seq} = n+1$
6. Trudy's forged answer: Bob is 42.9.9.9,  $\text{seq} = n+1$
7. Real answer (rejected, too late)

How Trudy spoofs Alice's ISP.

# Secure Naming (4)

DNSsec fundamental services:

- Proof of where the data originated.
- Public key distribution.
- Transaction and request authentication.

## Secure Naming (5)

Domain name	Time to live	Class	Type	Value
bob.com.	86400	IN	A	36.1.2.3
bob.com.	86400	IN	KEY	3682793A7B73F731029CE2737D...
bob.com.	86400	IN	SIG	86947503A8B848F5272E53930C...

An example RRSset for *bob.com*. The *KEY* record is Bob's public key. The *SIG* record is the top-level *com* server's signed hash of the *A* and *KEY* records to verify their authenticity.

# SSL—The Secure Sockets Layer (1)

Secure connection includes ...

- Parameter negotiation between client and server.
- Authentication of the server by client.
- Secret communication.
- Data integrity protection.

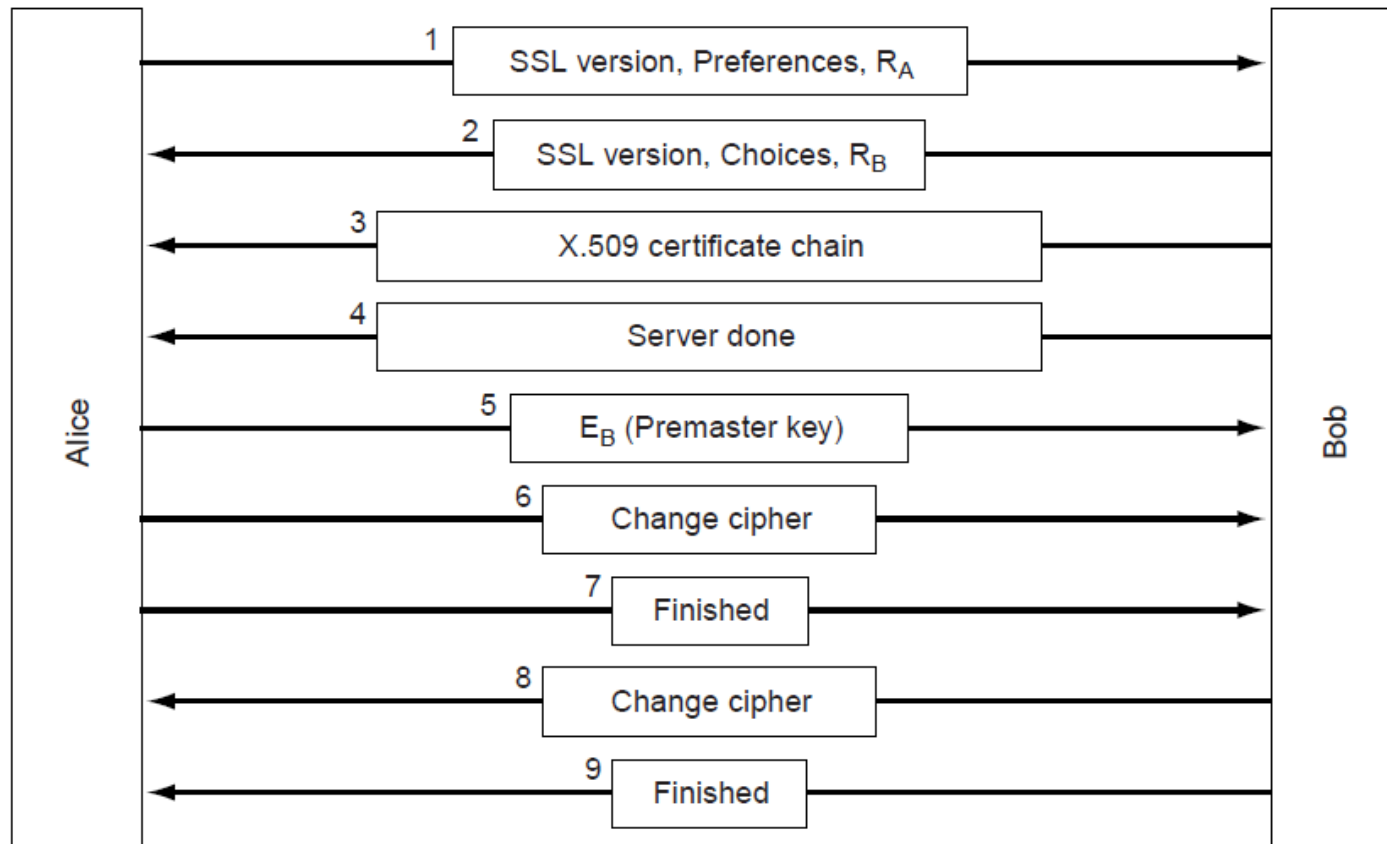
# SSL—The Secure Sockets Layer (2)

Application (HTTP)
Security (SSL)
Transport (TCP)
Network (IP)
Data link (PPP)
Physical (modem, ADSL, cable TV)

Layers (and protocols) for a home user browsing with SSL.

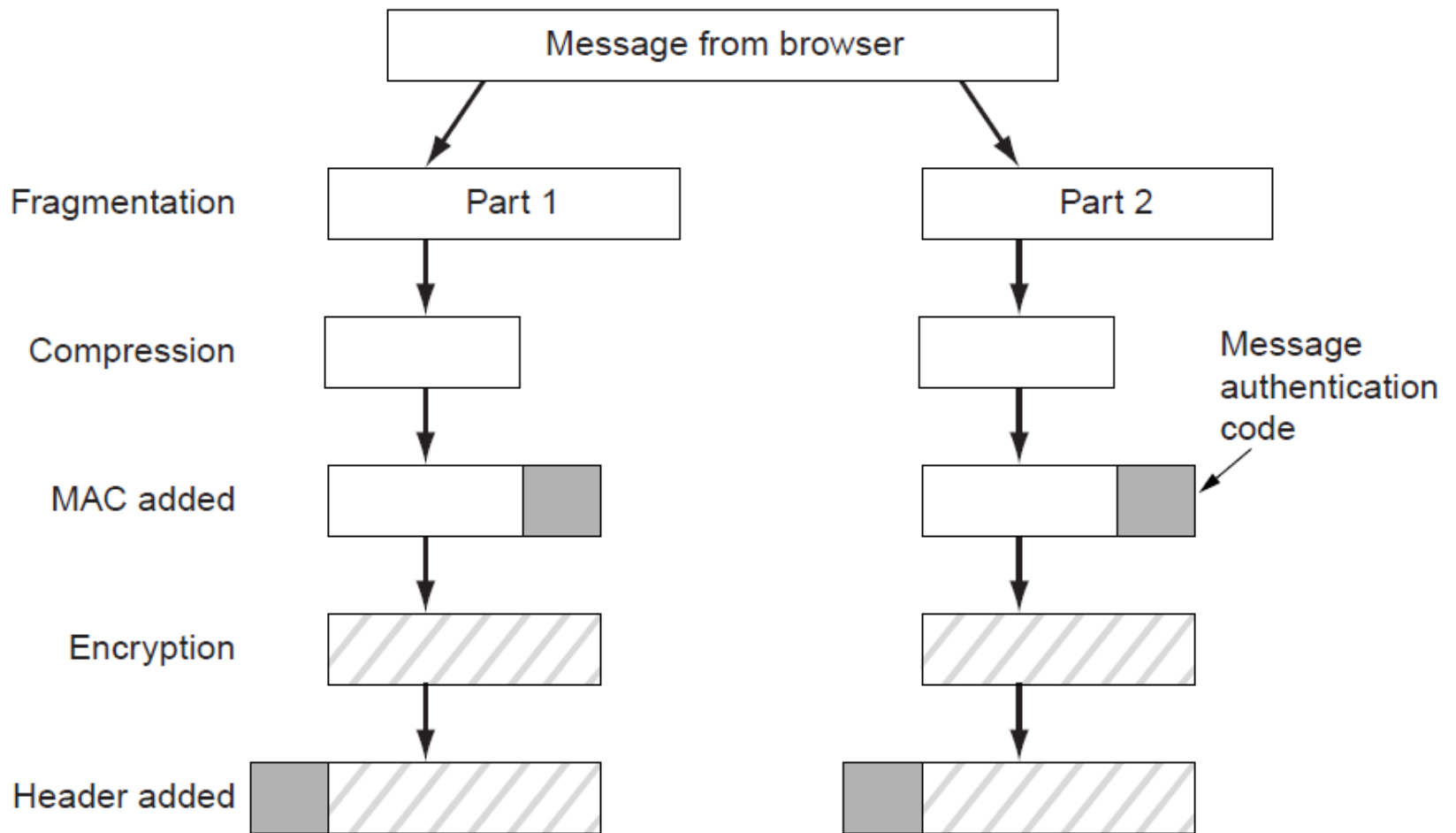


# SSL—The Secure Sockets Layer (3)



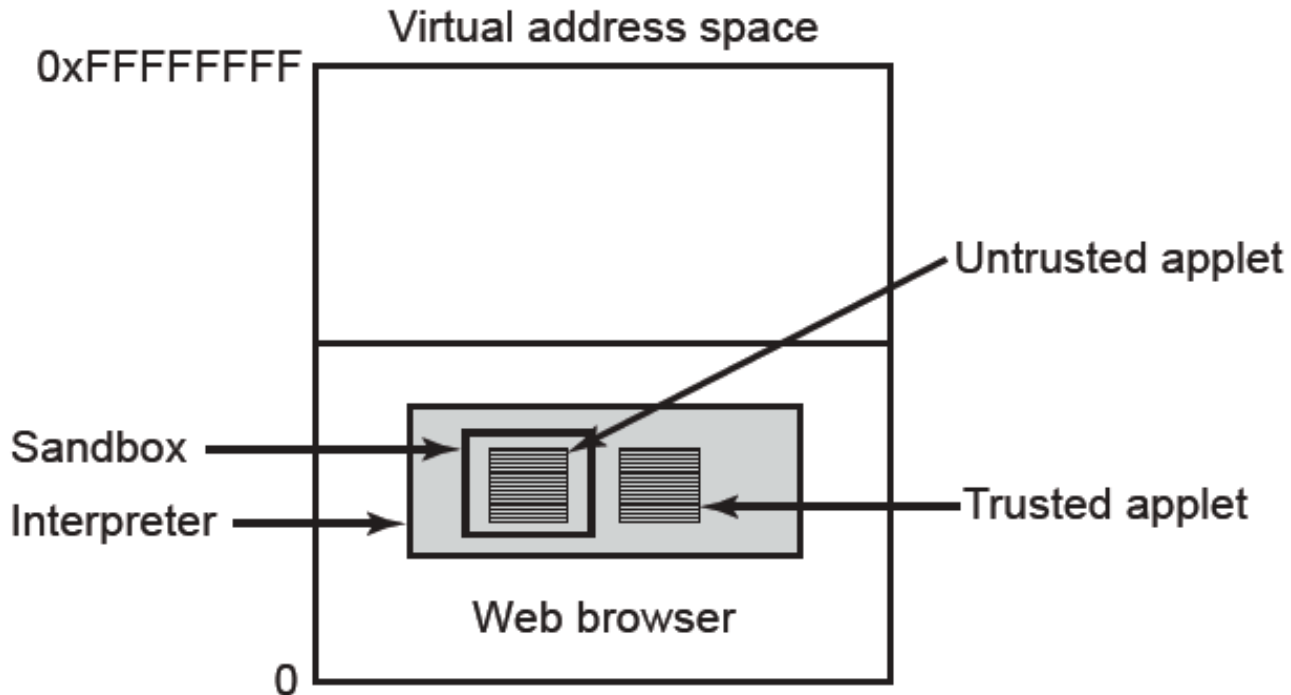
A simplified version of the SSL connection establishment subprotocol.

# SSL—The Secure Sockets Layer (4)



Data transmission using SSL

# Mobile Code Security

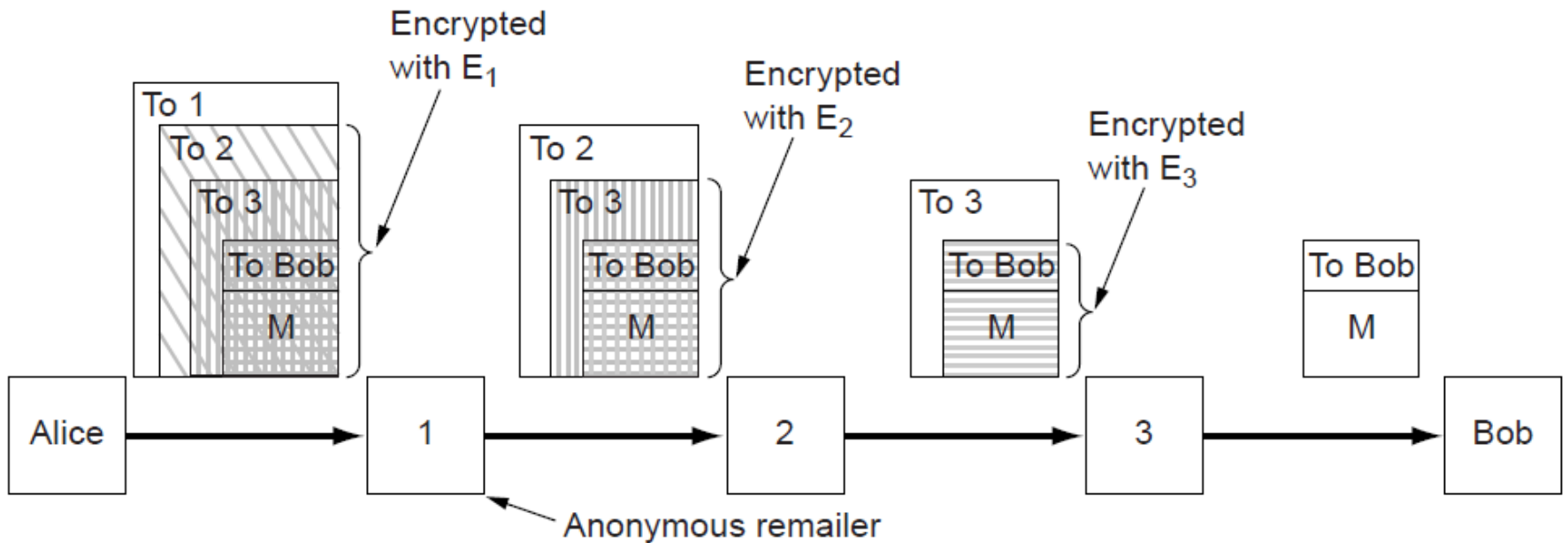


Applets can be interpreted by a Web browser

# Social Issues

- Privacy
- Freedom of speech
- Copyright

# Privacy



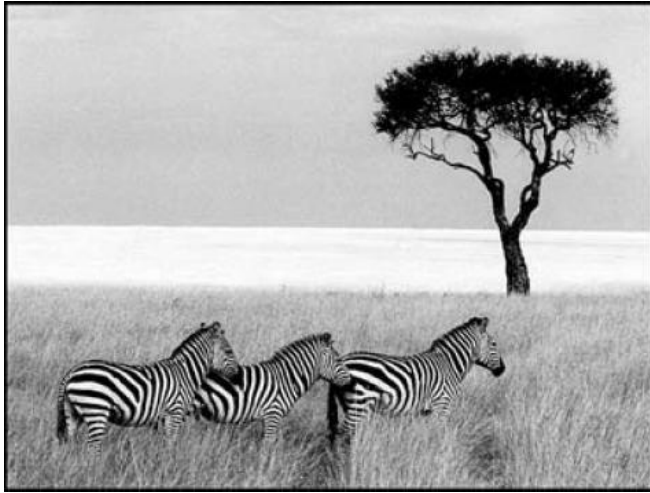
How Alice uses 3 remailers to send Bob a message

# Freedom of Speech (1)

Possible banned material:

- Inappropriate for children
- Hate aimed at various groups
- Information about democracy
- History that contradicts government position
- Manuals for potentially illegal activities

## Freedom of Speech (2)



(a)



(b)

(a) Three zebras and a tree.

(b) Three zebras, a tree, and the complete text of five plays by William Shakespeare.

End