COMP 5320/6320/6320-D01 Design and Analysis of Computer Networks

Introduction

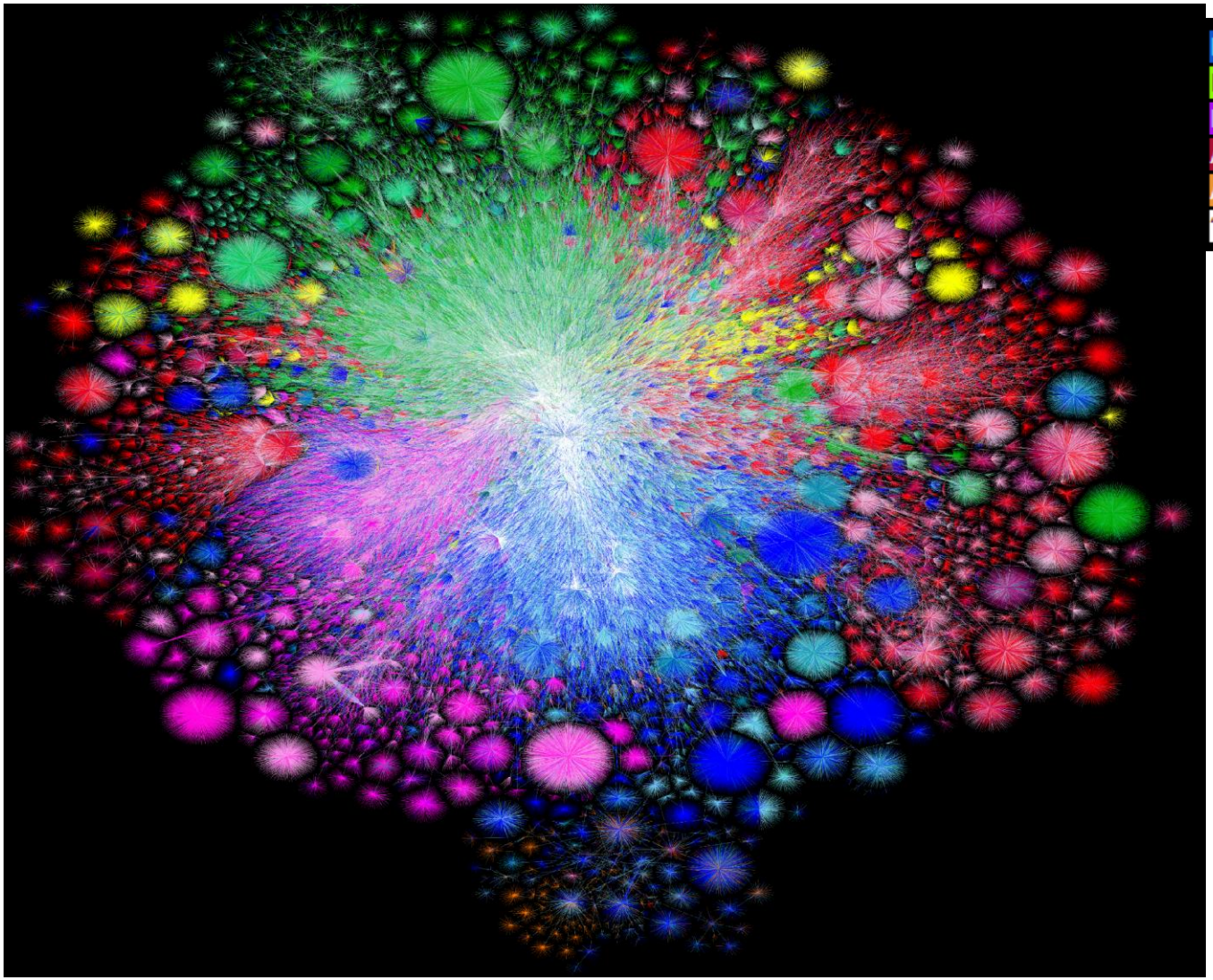
Instructor: Tao Shu, Ph.D.
Department of Computer Science and Software Engineering
Auburn University

Outline

- Motivation and the trend
- Course topics and expectations (and differences from 4320)
- Course organization and grading policy

Internet is complex

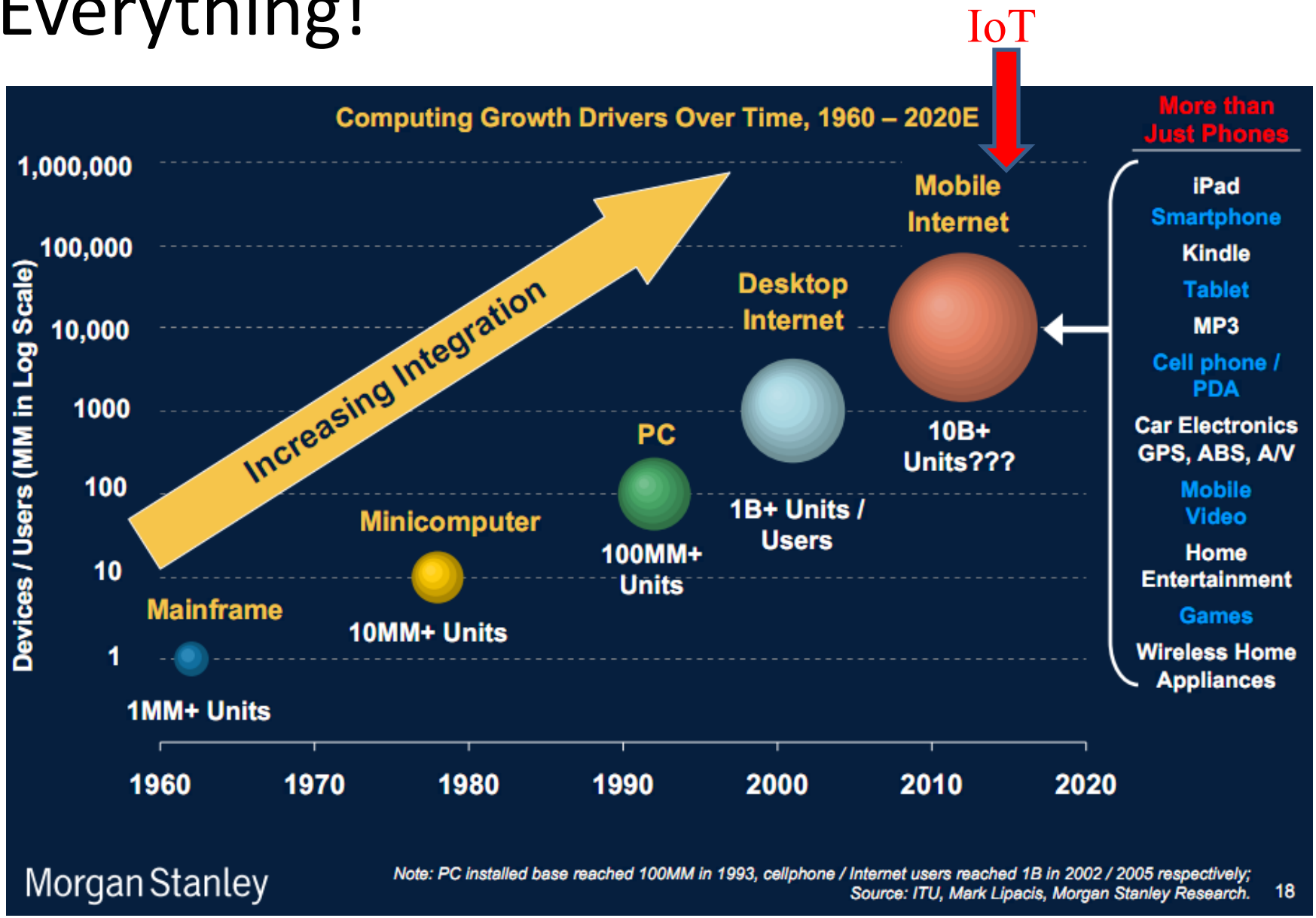
- The most complex man-made system on Earth.



Internet 2021

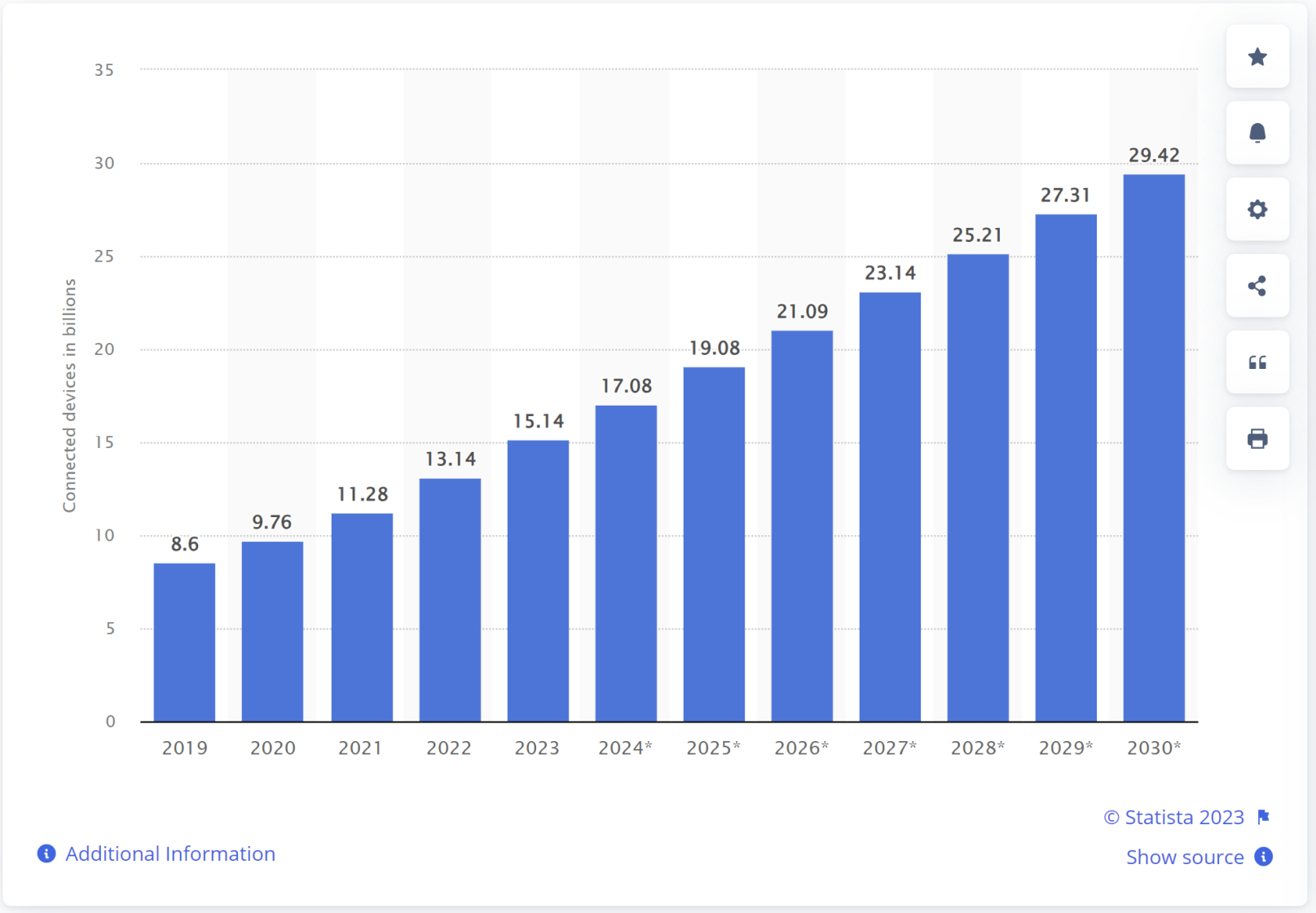
Source: Opte project

Where are we heading? Connecting Everything!



Number of Devices Connected by Internet

(in billions)



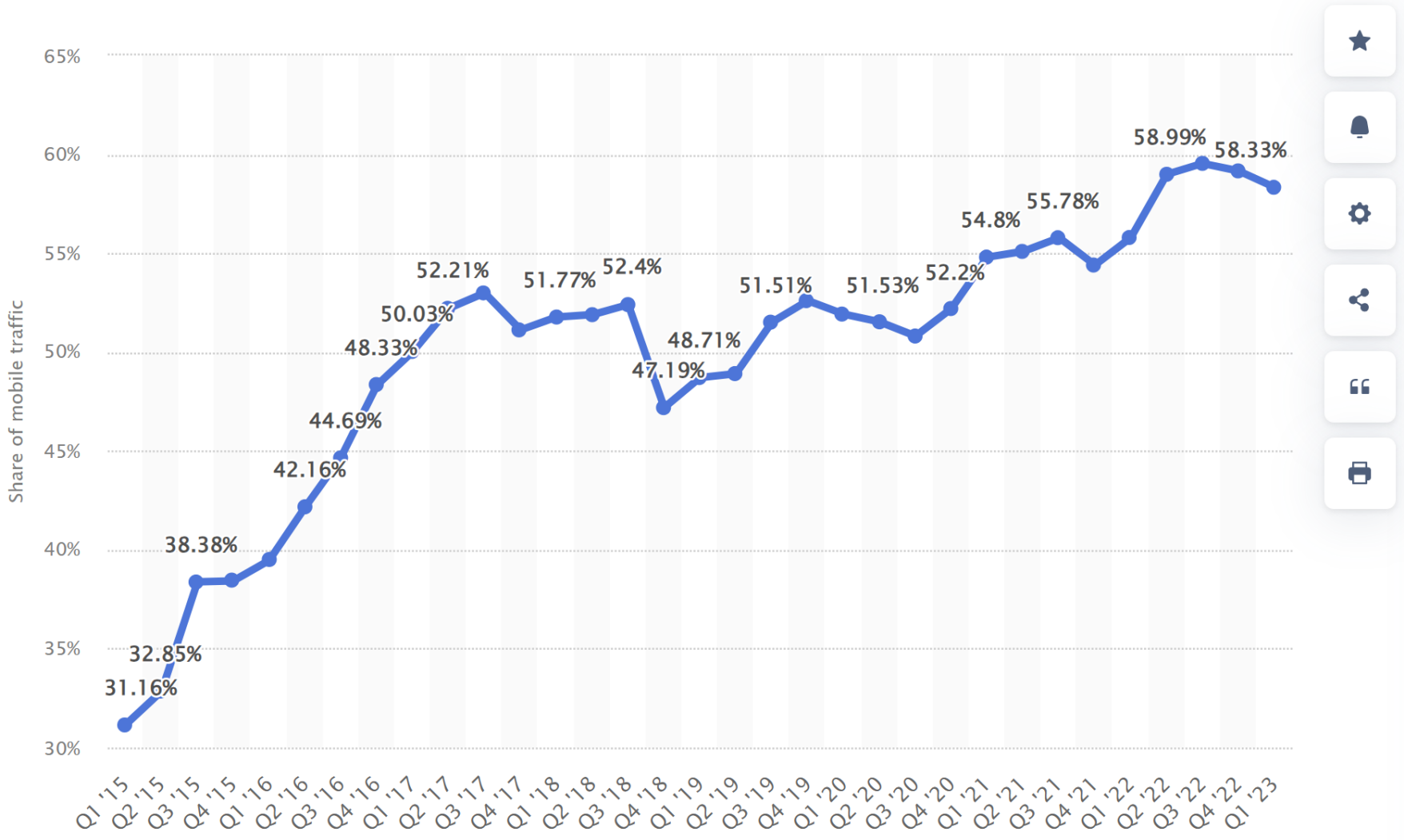
Source: Statista 2023

Challenges and Opportunities

Traditional goals: Scale, Connectivity, and Perf.

New trends arise → not a straightforward larger-version of the (old) Internet → new challenges, new features → Need to revise/re-develop new models/methods/protocols to accommodate new challenges → get a deeper understanding of current designs, so you can change them!

New Trends: go mobile!



Share of mobile Internet traffic

Additional Information

© Statista 2023

Show source

Trend: Mobile Internet

What are the new problems? Wireless? Mobile? S&P implicatons?

Source: TIME survey; June-July 2012

Today's Smart Wireless Device

Sensors:

- Fingerprint
- Barometer
- Three-axis gyroscope
- Proximity
- Compass
- Ambient light
- Dual cameras
- Dual microphones
- GPS
- Accelerometer

Computing

- 1.8GHz 64-bit Dual-core CPU
- GPU
- Motion coprocessor

Memory

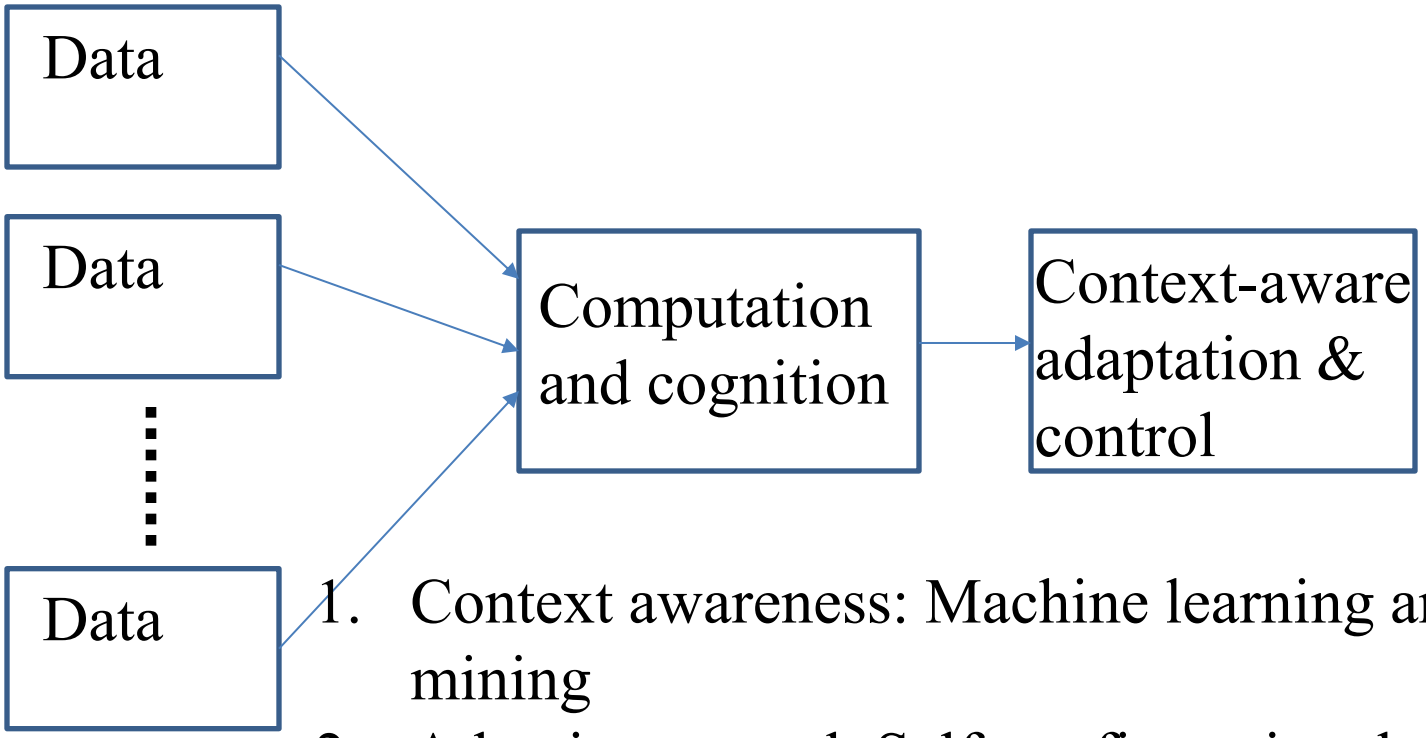
- 2 GB RAM
- 16 to 128 GB Storage

Comm.

- Cellular (LTE)
- WiFi
- Bluetooth
- NFC

Sensing+computing+storage, much more than a transceiver!

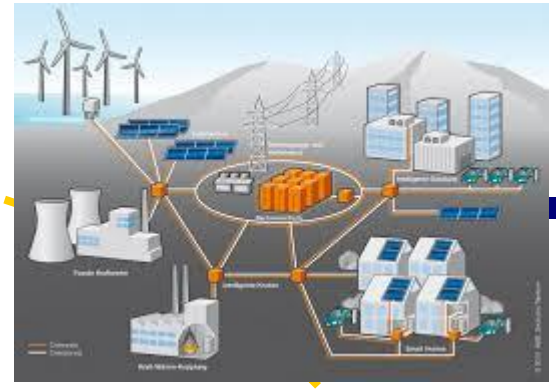
Context-aware Networking

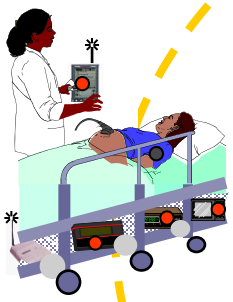


1. Context awareness: Machine learning and data mining
2. Adaptive control: Self-configuration design, robotics, swarm intelligence
3. Security and privacy: data, computation, learning security

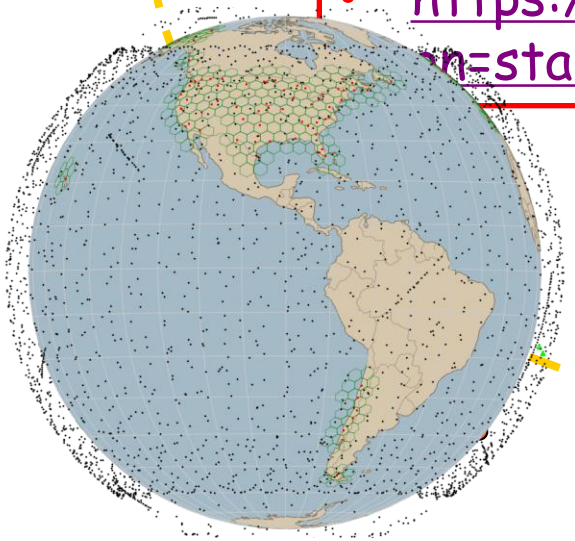
We need new knowledge that enables the above integration!

New Trend: be pervasive (M2M, D2D)

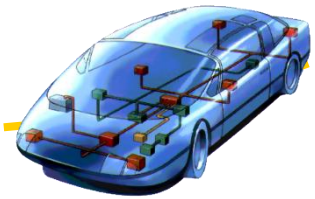




- Communication between any things! 1000x increase for wireless traffic!
- Anywhere connectivity, anywhere communication!
- https://satellitemap.space/?constellation=starlink#



6G

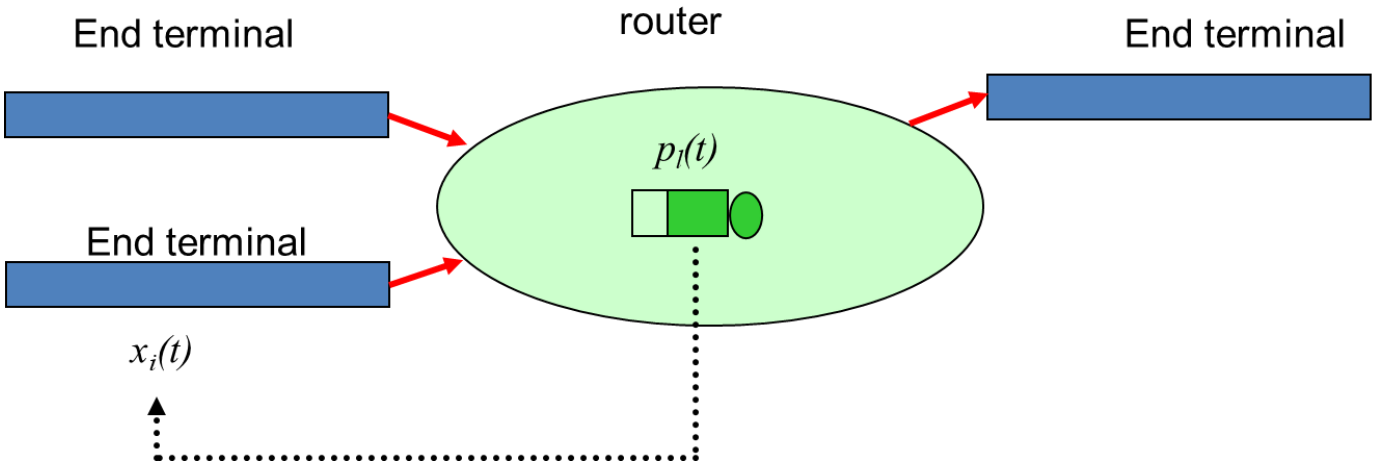


Example: Smart Warehouse

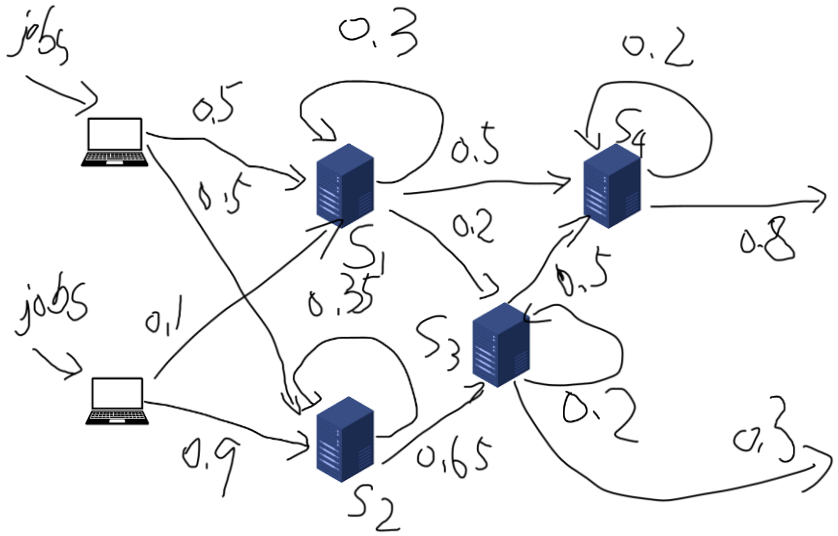
QoS?

Motivating Example Problem

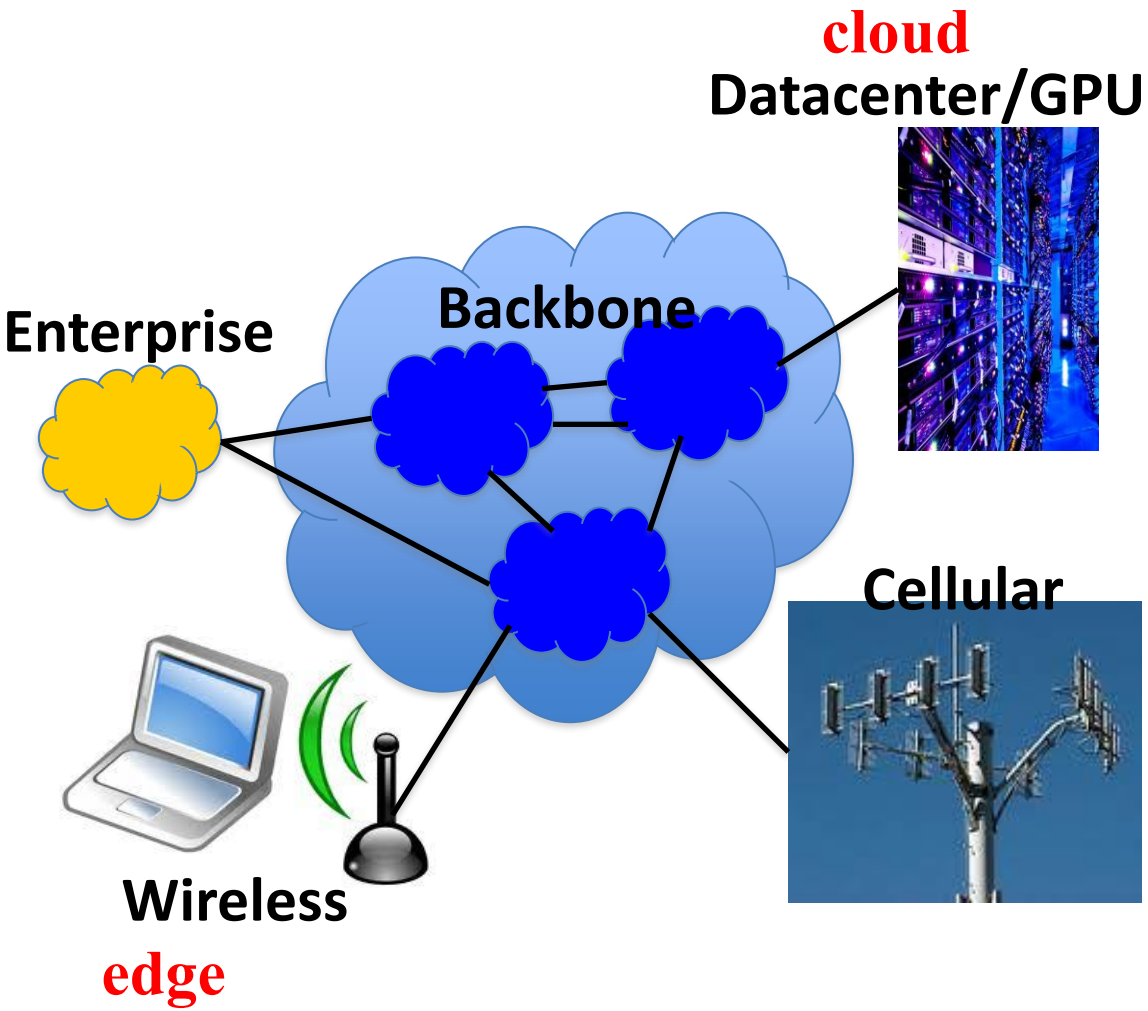
TCP over wireless



Network with wireless links



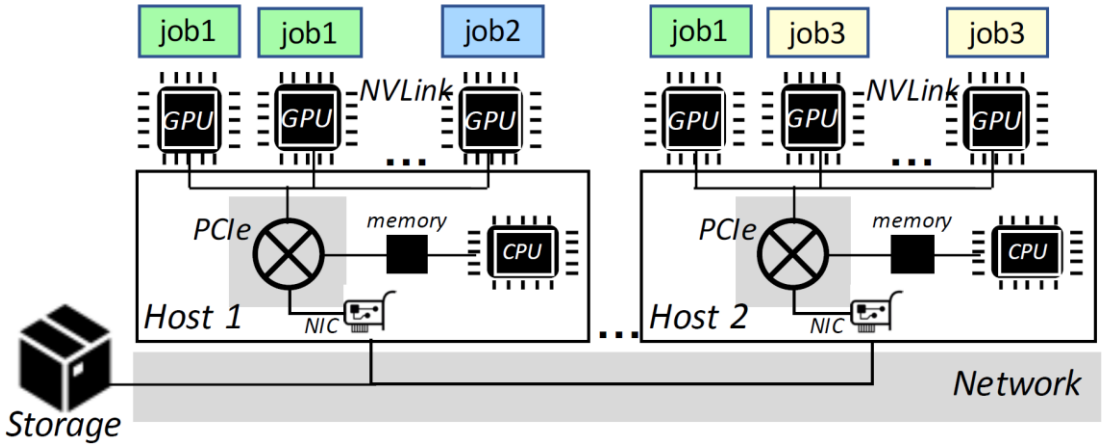
New Trend: Virtualization and Sharing of Resources



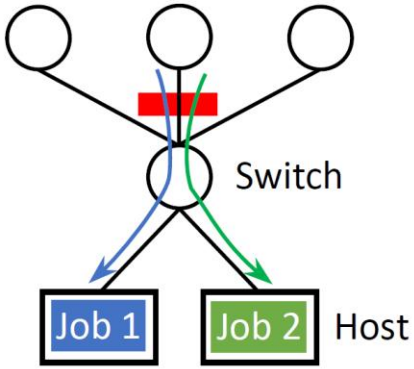
- Virtualizing resources and services (moving bits for datacenter/GPU farm)
- Infrastructure sharing
- Multi-tenancy
- Cloud, fog, and edge

…

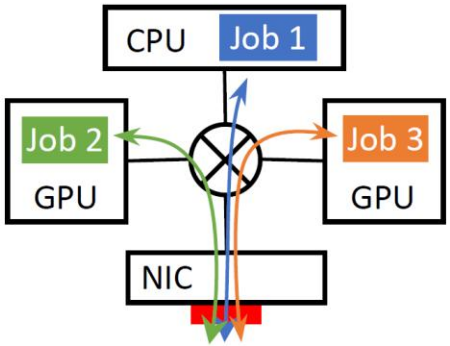
Motivating Example: Job Scheduling in Multi-tenant GPU Farm



Typical allocation of GPUs in a GPU farm



(a) Contention on network forwarding paths



(b) Contention on intra-host PCIe links or NVLinks

COMP 4320 Material (Introduction: *what*?)

- OSI Reference Model
  - Physical Layer
  - Link Layer
  - Network Layer
  - Transport Layer
  - Session
  - Presentation
  - Application

Differences 5320/6320 - 4320

Focusing on more rigorous and quantitative performance characterization (and optimization)

- **Models**
- **Methods**
- **Protocols**

- More *Why* and *How*:
- More in-depth and quantitative look at the subject
- Emphasize the new trend: mobile/wireless, AI, virtualization

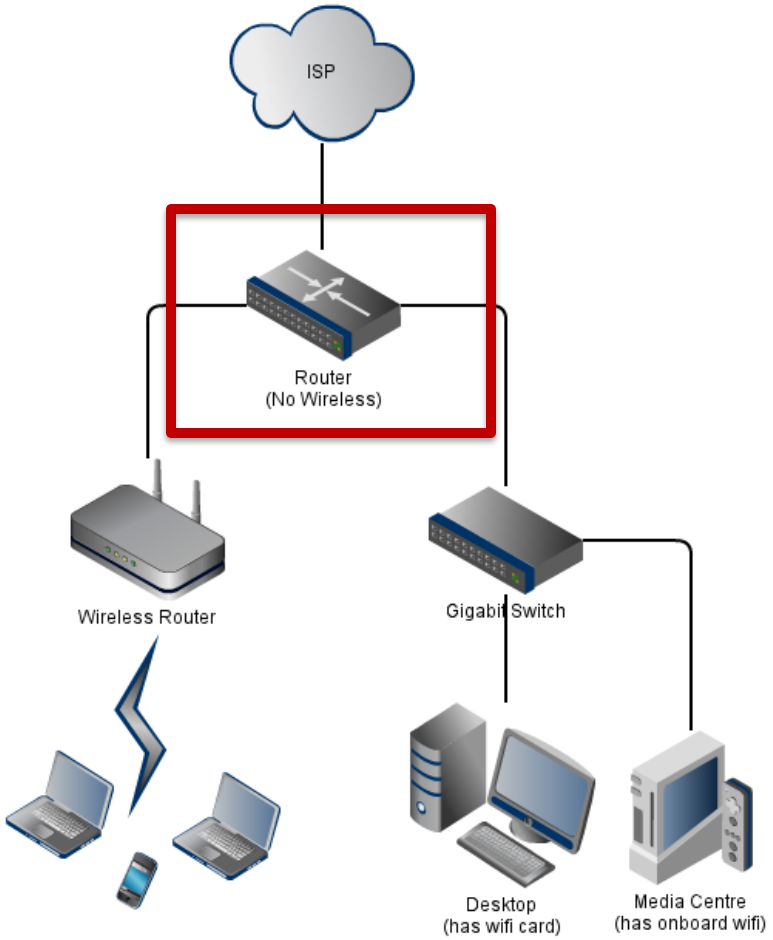
Sample Course Topics

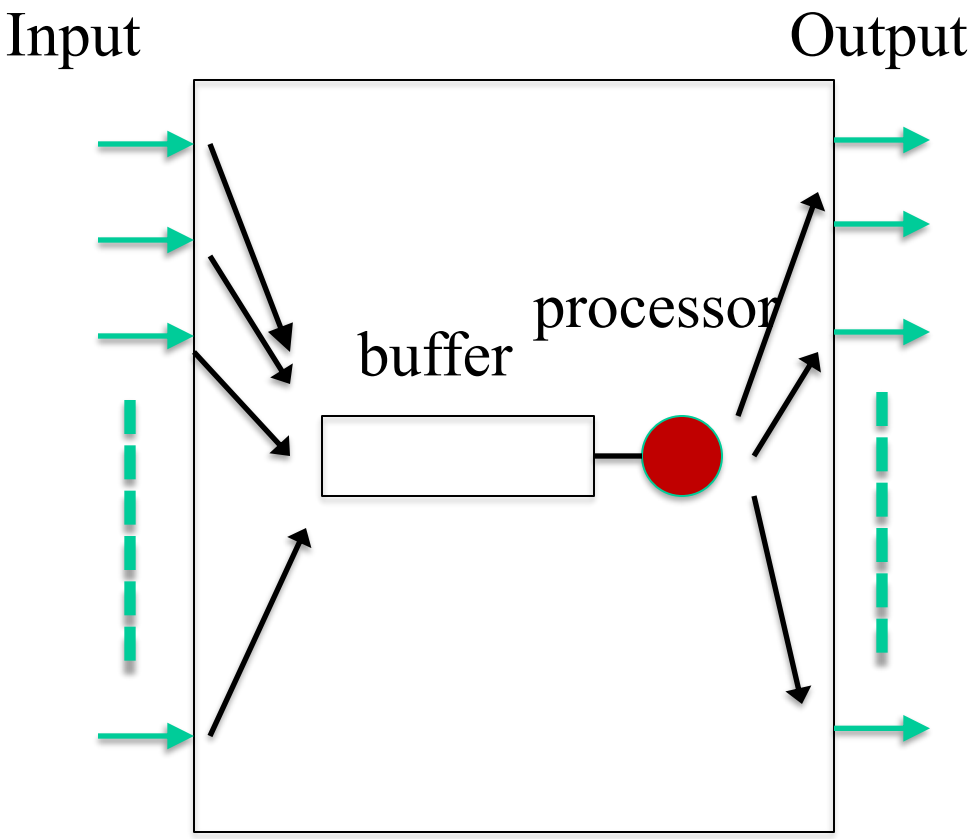
- Review of 7 layers
- Socket programming (quick start, being able to develop network applications)
- Queueing theory and models (key points)
- Medium access
- Packet scheduling
- Queue management
- Routing (wired and wireless)
- Transport flow and congestion control (wired and wireless)

Socket Programming

- Develop network applications
- Client-server model
- TCP/UDP socket primitives
- concurrent server design
- Mini programming projects

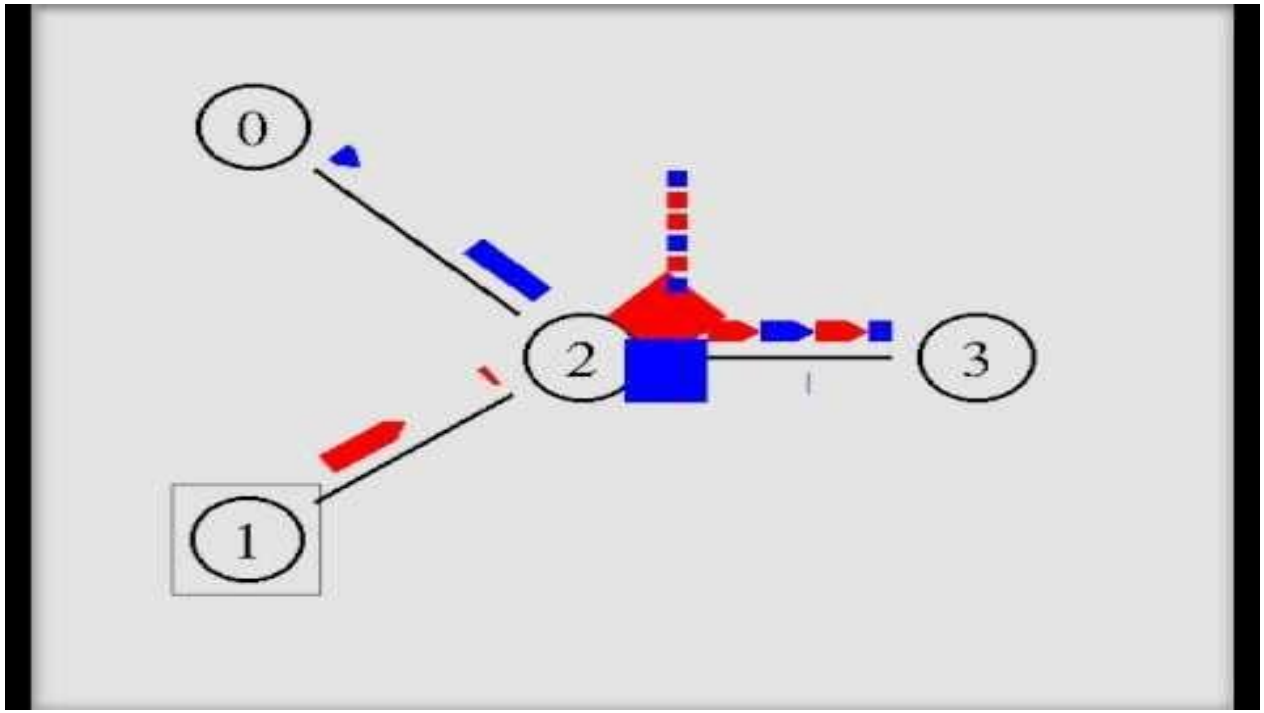
Network and Computer Modeling





A queueing model for router

Quantitative Performance Analysis?

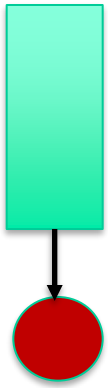


Queue length?
Delay?
Packet loss rate?

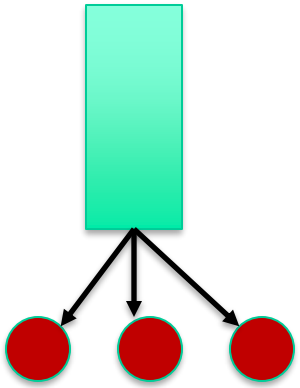
Traffic distribution: CBR vs. VBR?
Size of the buffer?
Server capacity (line rate)?

Network and Computer Modeling

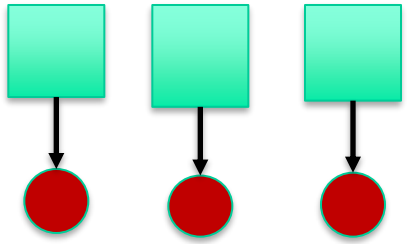
- Router (or VM) architectural design



Single processor, single buffer



Multi-processor sharing single buffer



Multi-processor with dedicated buffer

Service rules:

- FCFS
- random
- Priorities (preemptive, non-preemptive)

Quantitative performance metrics

- Queue length?
- Delay? Job loss rate?
- Processor utilization?
- **AI-based adaption?**

Medium Access Control

(Will require introduction to Queueing Theory)

- Aloha, slotted Aloha, CSMA/CD and CA: performance evaluation

- 802.11: detailed description, performance evaluation.

Packet Scheduling

- It is the selection of the next packet to transmit, usually happens at the edge
- First come first serve, round robin, priority based, weighted fair queue, etc.

Queue (Buffer) Management

- When buffer overflow, which packet to drop? Usually happens in the core
- Droptail, Drophead, RED, Choke….

Routing

- Distance vector
- Link state routing
- Routing in MANETs (AODV and DSR)
- Opportunistic routing and network coding

Congestion control

- TCP: Tahoe, Reno, Vegas
- TCP over wireless
  - Link level solution
  - Split connection
  - TCP-aware link layer
  - Explicit notification
  - Receiver/sender-based discrimination

Course Logistics

Class Meeting

MWF, 3 to 3:50 pm

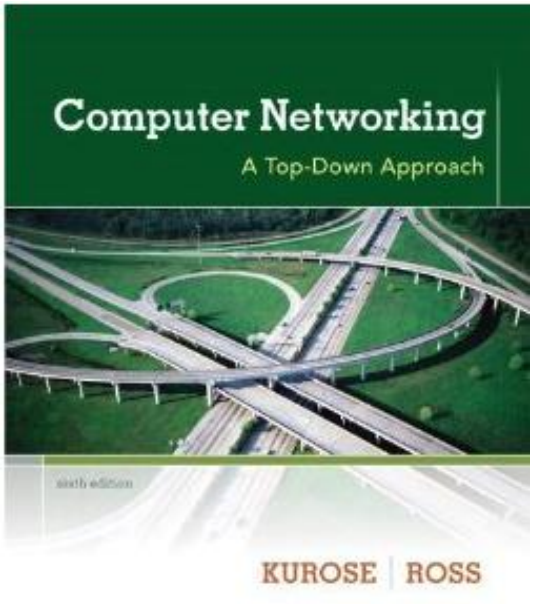
Shelby 1120

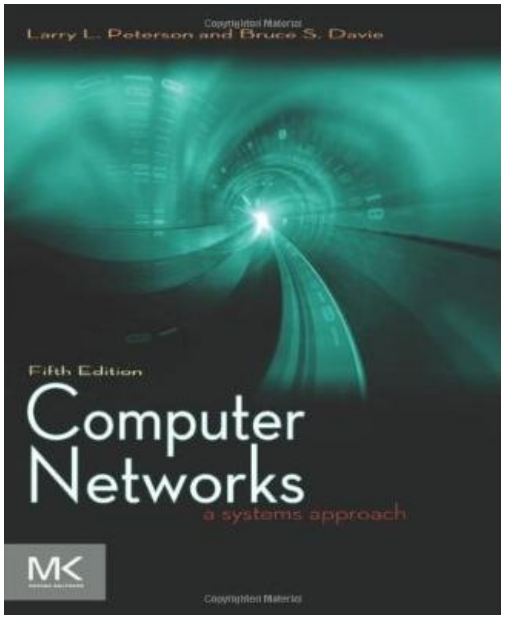
Reference Text

"Computer Networks: A Top-Down Approach", 5th Edition,
J. F. Kurose and K. W. Ross

"Computer Networks: A Systems Approach"
L. Peterson and B. Davie, 5th edition.

"Queueing systems, Vol.1: Theory" L. Kleinrock.





Course Website

Canvas

Lectures, Homework, Useful links,

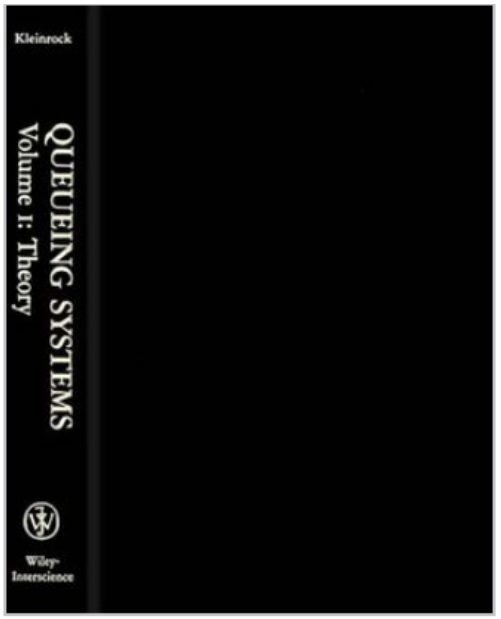
Supplementary material, Announcements

Get Your engineering account!!!

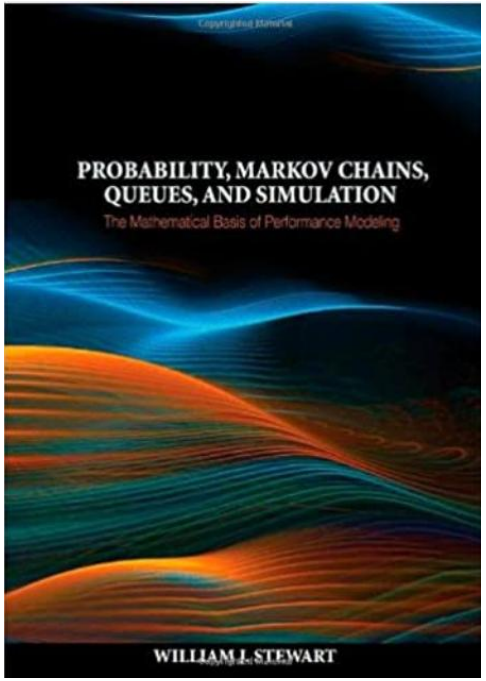
Course Logistics

Reference text for queueing theory

L. Kleinrock, “Queueing Systems, Vol. 1: Theory,” John Wiley & Sons, 1975



W. J. Stewart, “Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling,” Princeton University Press, 2009.



Where to Find Me

My office:

Shelby 3101K

Office hours: 2:10 pm – 3 pm MWF

or by appointment

My email: tshu@auburn.edu

TA: Guan Huang (gzh0040@auburn.edu)

Grading Policy

- Grade distribution

Hwk and programming assign.	10%
Queueing theory project	10%
Midterm 1	20%
Midterm 2	25%
Final	35%
Total	**100%**

- Late submission: zero credit (firmly executed, no discussion)
- Grade dispute: within 1 week after the grade has been posted

6320/D01: **A** (90%→ ...), **B** (80% → 90%$^{--}$), **C** (70% → 80%$^{--}$), **D** (60% → 70%$^{--}$), **F** .....

5320: A (85%→ ..), **B** (75% → 84%$^{--}$), **C** (65% → 74%$^{--}$), **D** (55% → 64%$^{--}$), **F** .....

Review of Network Protocol Stack: Basic Concepts of Networking

Outlines

- Protocols and layers (E2E and P2P)
- OSI reference model
- The case for the Internet (TCP/IP)
- Application layer protocols: DNS, TELNET, SMTP, HTTP

Communication over Networks

- A network is a set of ***independent*** devices which can communicate.
- Implications
  - Manufacturer/developer independent
  - Hardware independent
  - OS independent
  - …..
- Need complex functionalities to communicate

Protocols!

- A common set of rules accepted and followed by communicating parties
- *To communicate, we must follow*
  - Common set of rules
  - Common data format and presentation (language)
  - Example: telnet (virtual terminal), network calculator
- Different levels of communications

Building Applications

Network applications involve communication of two or more hosts

Often, complex functions need to be realized

Can think of application communication in an abstract way

To get this logical channel, you need additional functions to take care the physical details!

How to organize your protocol(s)?

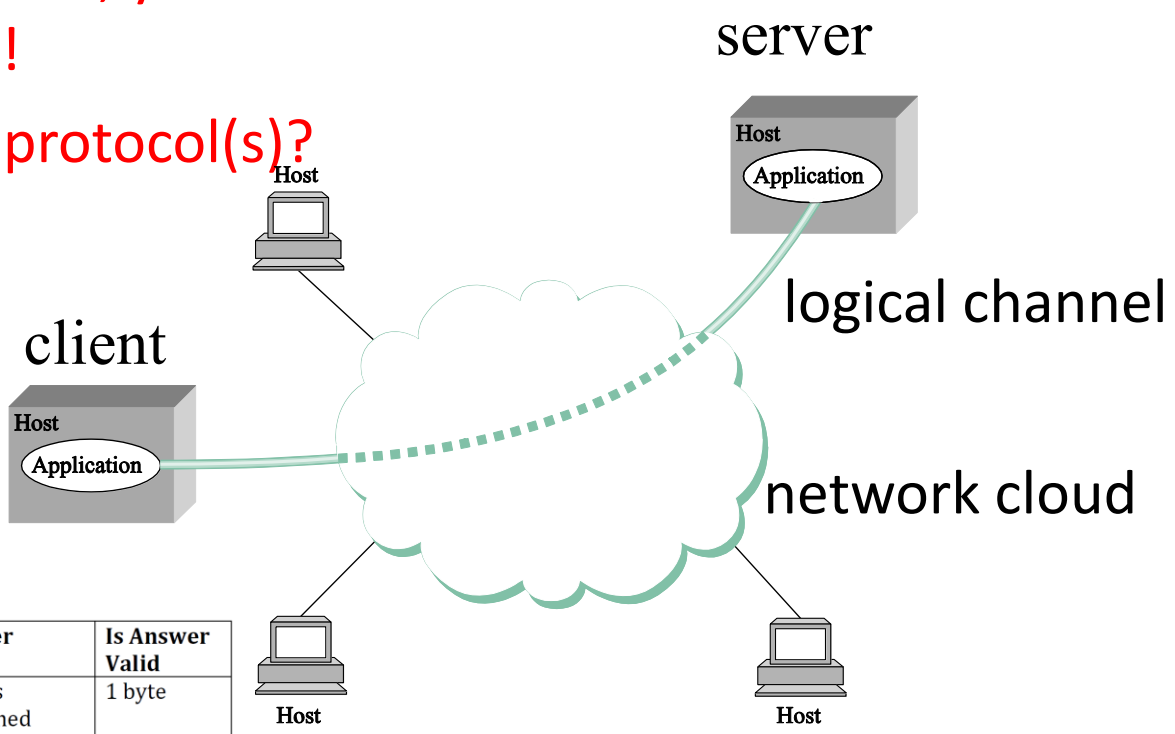
Example: Network calculator

Request Message

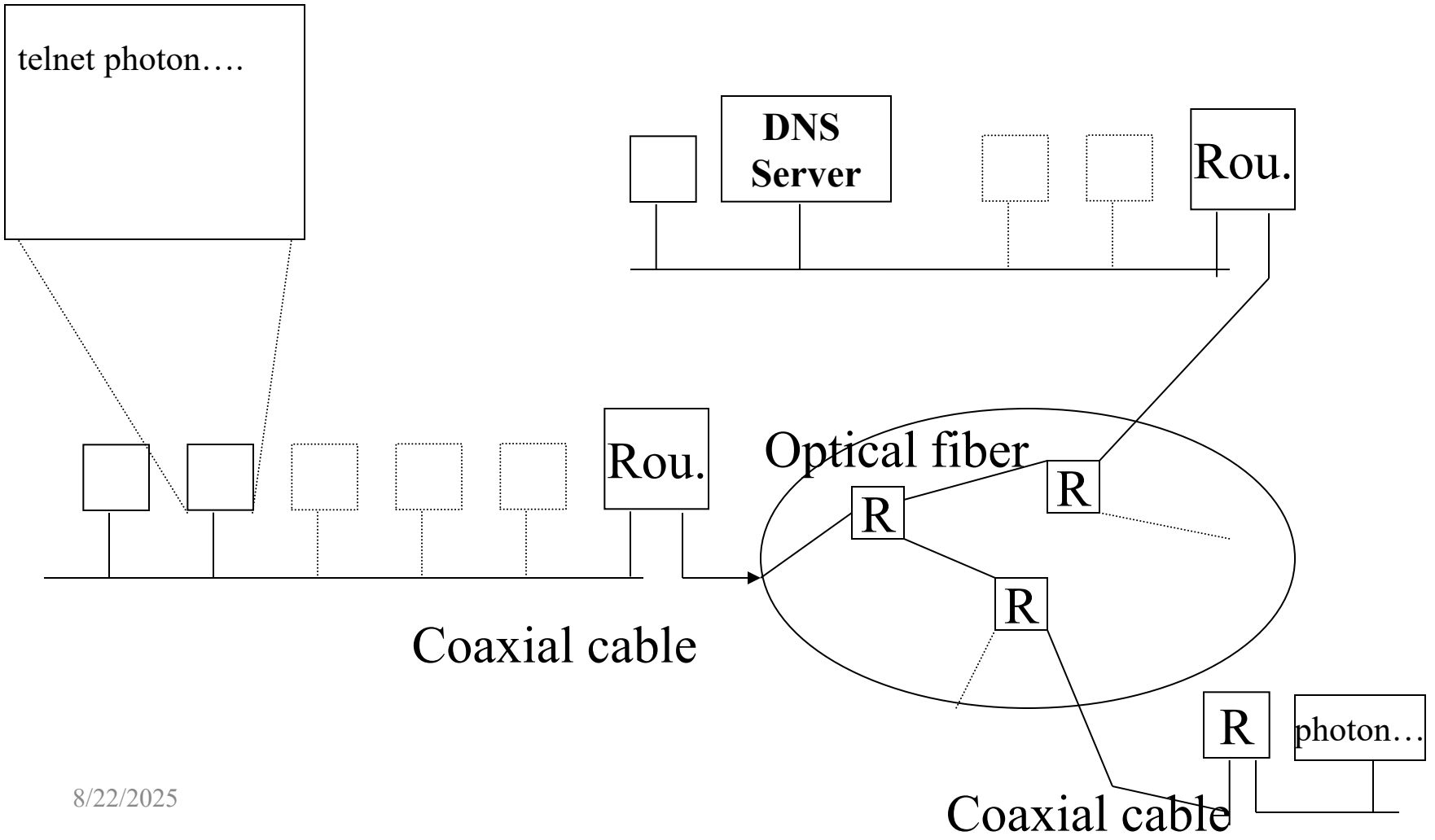
Operation Code	Operand A	Operand B
1 bytes '+' = 0x2b (43) '-' = 0x2d (45) 'x' = 0x78 (120) '/' = 0x2f (47)	4 bytes (unsigned integer)	4 bytes (unsigned integer)

Response Message

Operation Code	Operand A	Operand B	Answer	Is Answer Valid
1 bytes '+' = 0x2b (43) '-' = 0x2d (45) 'x' = 0x78 (120) '/' = 0x2f (47)	4 bytes (unsigned integer)	4 bytes (unsigned integer)	4 bytes (unsigned integer)	1 byte 1 – Valid 2 – Invalid (NaN result)

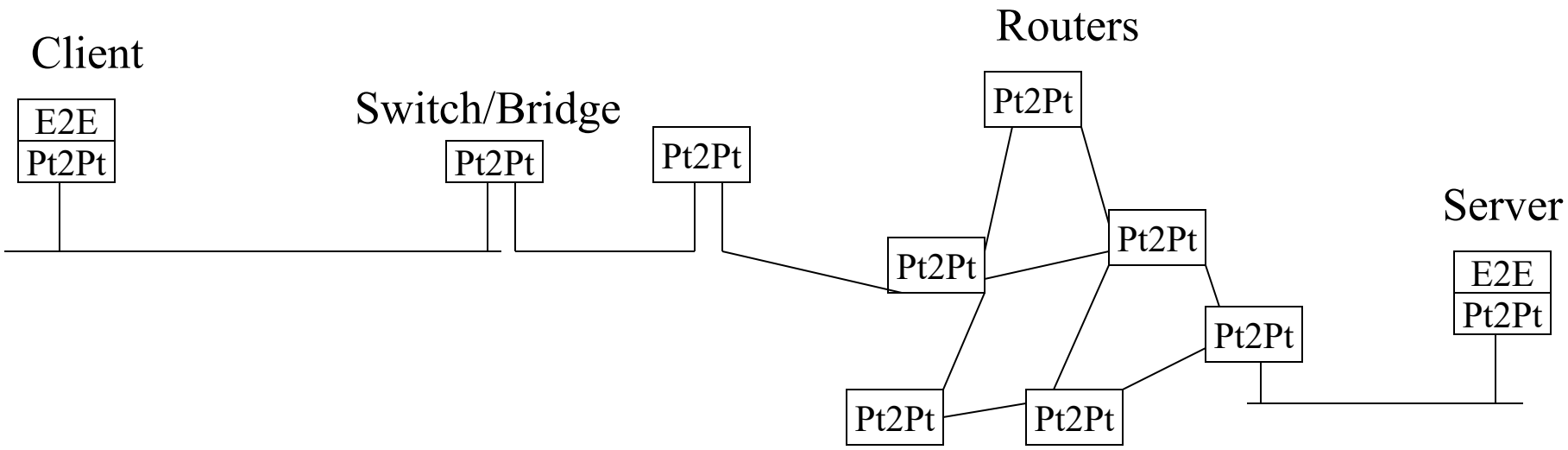


A Network


telnet photon….
DNS Server
Rou.
Rou.
Optical fiber
R
R
R
R
photon…
Coaxial cable
Coaxial cable

Two Types of Communication Entities

- **End-to-end**:
  - Used only at the end points of communication sessions
- **Point to point**:
  - Used on every relaying device



End-to-End Protocols (Examples)

- Applications: HTTP, FTP, Telnet, DNS

- Transport tools: TCP, UDP..

Point to Point

- Protocols: IEEE 802.x., HDLC, Bluetooth, PPP

- Devices: Ethernet NIC (IEEE 802.3), wireless card (IEEE 802.11), DSL, modem, ATM switch

The OSI Reference Model

(Open Systems Interconnection)

Flat or Layered?

Many functionalities to achieve: signal waveforms or modulations, coding, power control, medium access methods, routing, service requirement (realtime/non-realtime, loss/no loss)

K functions, N potential realizations/implementations for each function (attribute)

Complexity (# of protocols): N^K vs NK

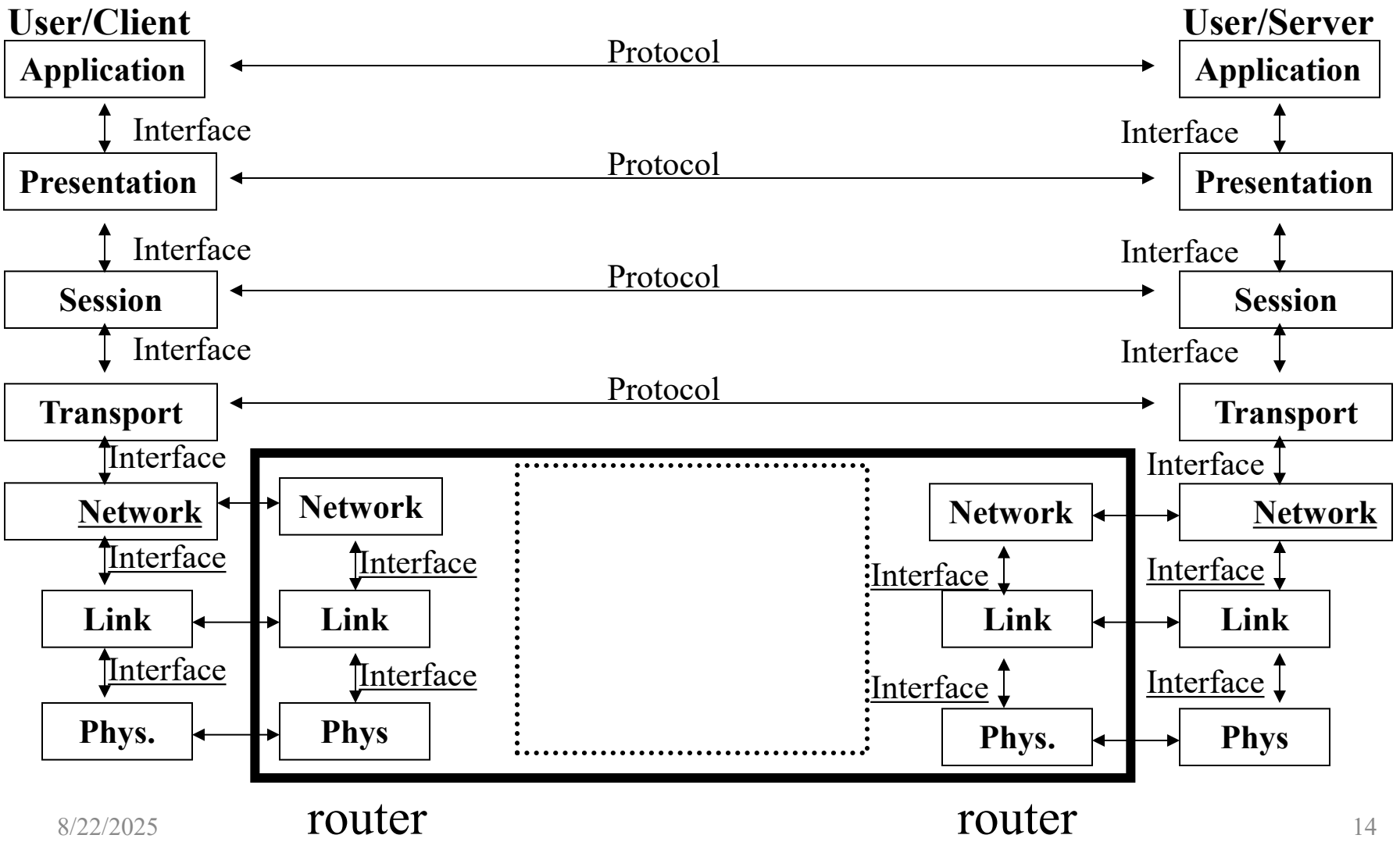
A Complex Problem: Divide and Conquer

- Communication should be divided/broken in layers such that:
  - Each layer has a well defined function
  - A lower layer hides its implementation details from upper layers
  - A lower layer provides service to upper layer through a neat interface
  - Higher layers use lower layer's service by calling the interface

Decomposing Communication in Layers

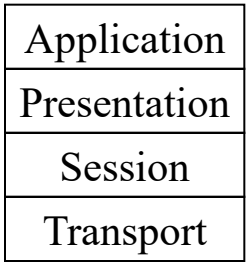
- International effort led by ISO (International Standards Organization)
- Economical conflicting interests
- Result: no viable stack implementation, but a nice formal reference model good for education and networking experts communication
- Neat definition of:
  - Functions at each layer
  - Interfaces between adjacent layers

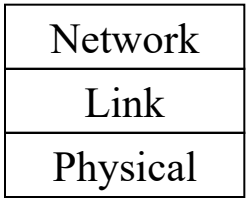
OSI Reference Model



Remarks

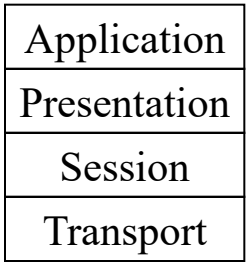
- A protocol is just a particular realization of the layer
- End-to-end layers (protocols):
  - Application
  - Presentation
  - Session
  - Transport
- Point-to-point layers (protocols):
  - Network
  - Link
  - Physical



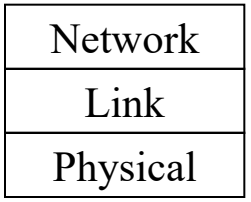


Implications

- **End-to-end** protocol: active instance of the protocol only on the end points.



- **Point-to-point protocol**: active instance throughout ALL the path, on endpoint nodes and intermediary nodes



Physical Layer

- Transmission of raw bits
- Deal with:
  - Electrical specs
  - Mechanical specs

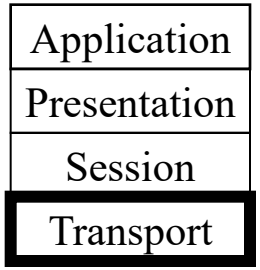
Link Layer

- Transmission trunk of bits with defined structure (frames)
- Point to point, has two fundamental functions:
  - Data Link Layer
    - Framing
    - Error control
    - Flow control
  - Medium Access Control Layer

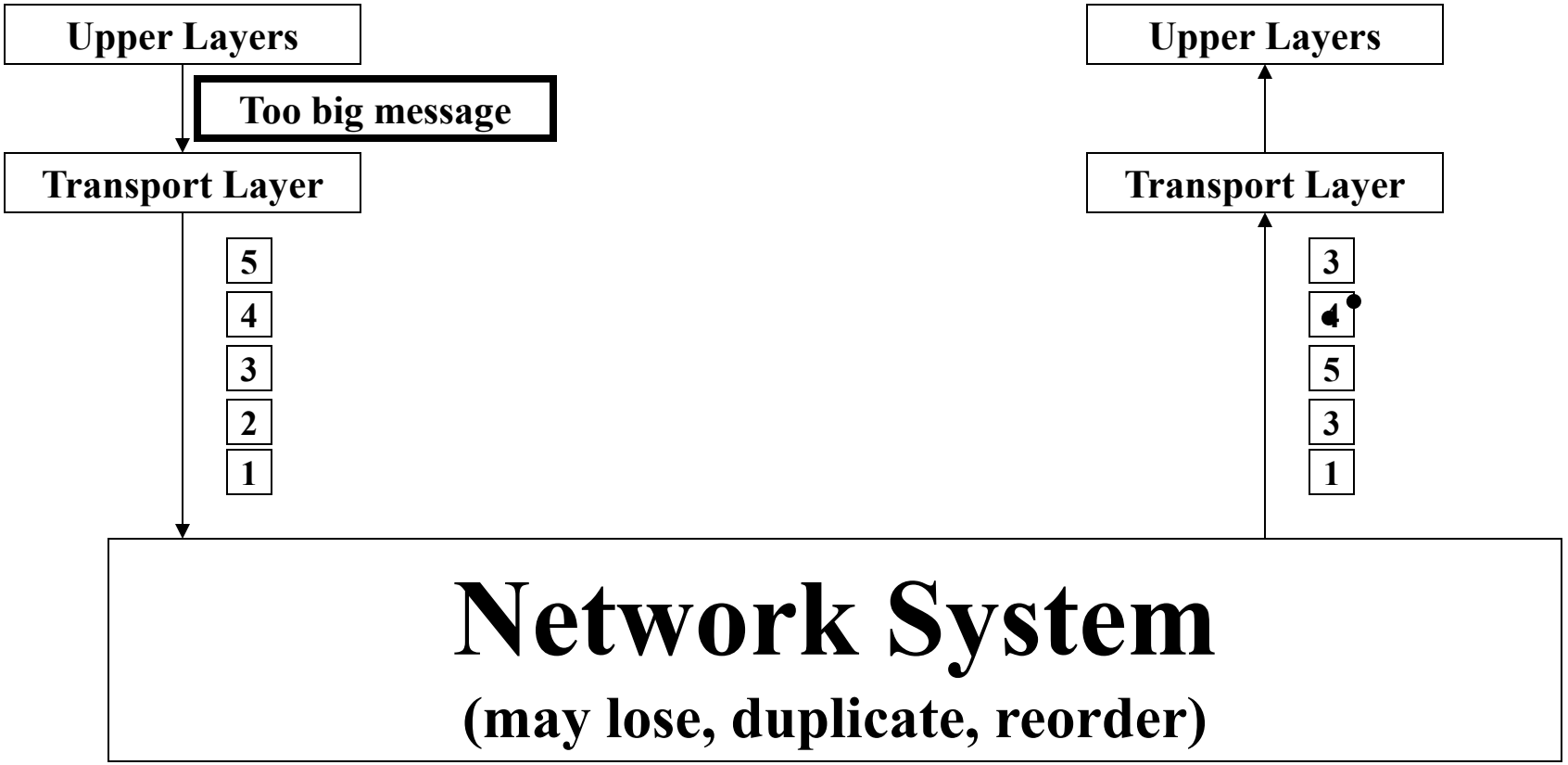
Network Layer

- Deals with the direction (multi-hop)
- Point to point
  - Addressing
  - Routing
  - Queue/buffer management
  - Administration

Transport Layer



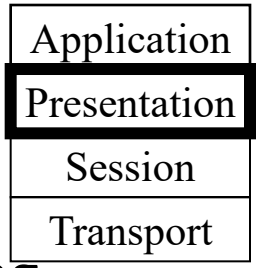
- End to end
- Hides/adapts defects and limitations of the network sys.



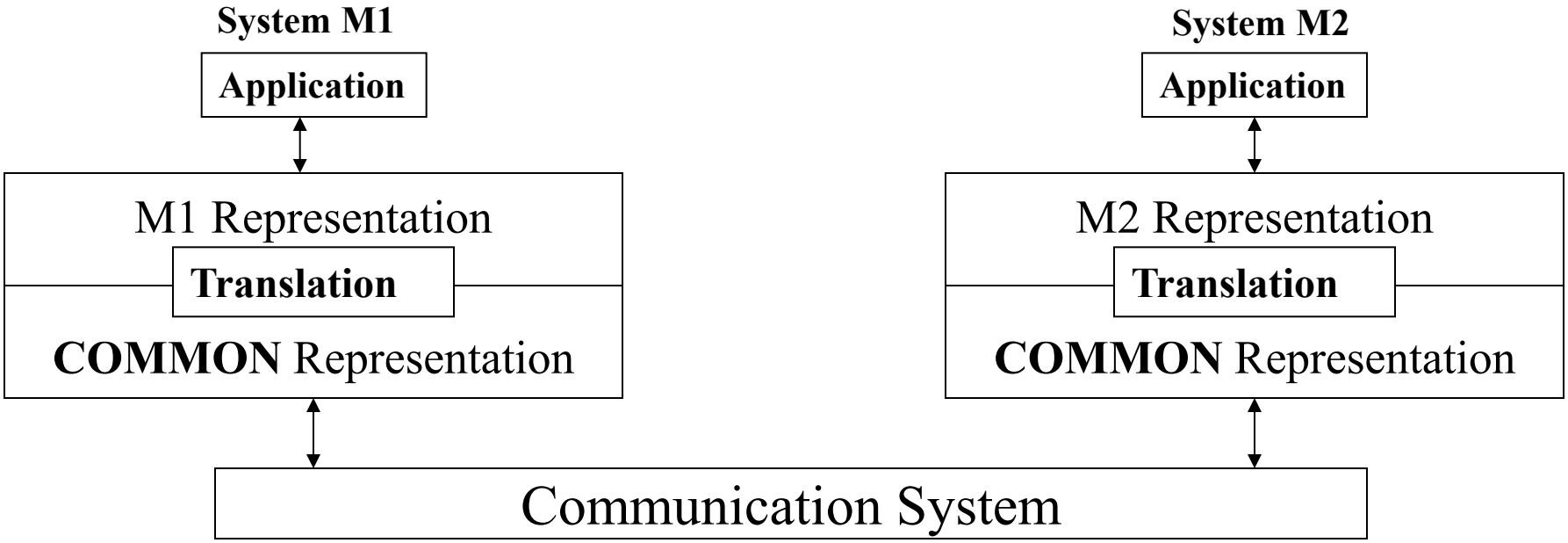
Session Layer

- End to end
- Allows establishing, managing, and closing communication sessions
  - Recover connection when it's lost
  - Close a connection if it's idle for some time
- X.225 (connection-oriented session protocol)

Presentation Layer

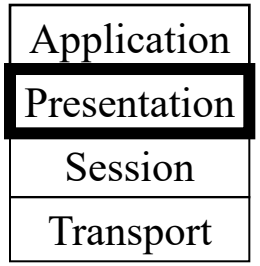


- End to end
- Deals with data presentation: the two communicating entities may be different (different vendors, different OS, different CPUs)



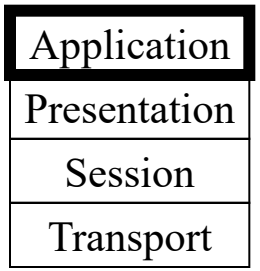
Presentation Layer (Cont'd): Examples of Common Rep's

- Network byte order:
  - Little endian
  - Big endian (network byte order)

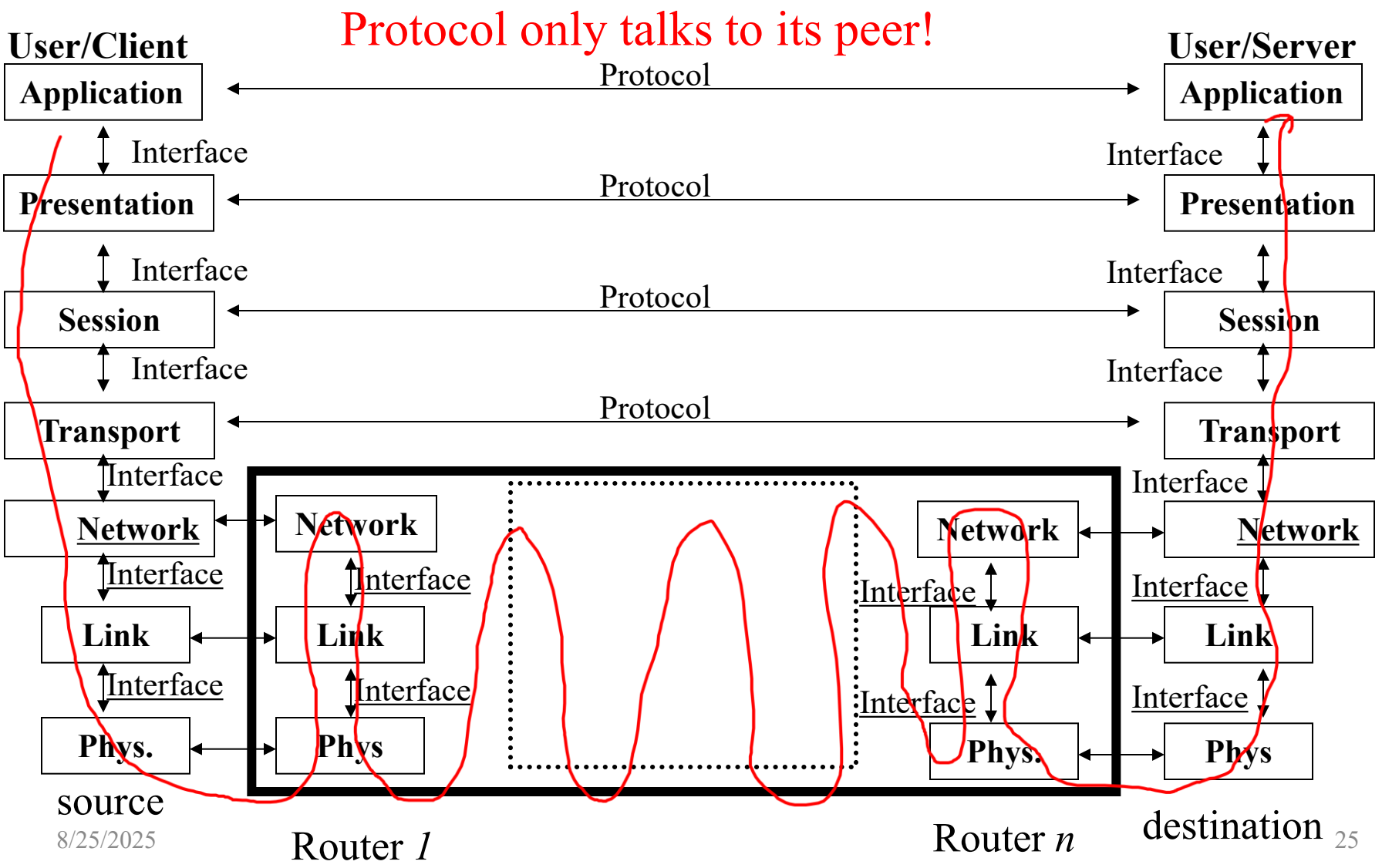


Application Layer

- End to end
- Communication services provided directly to the user (or by server):
  - File transfer
  - Email
  - http
  - X.400 (message handling systems)….



Flow of Information/data



Protocols only talks to its Peer

- When going down the stack, add header in front of the payload
- When going up the stack, strip away the appropriate header

PHY hd	Link hd	Network hd	Transport hd	App data

Conclusion

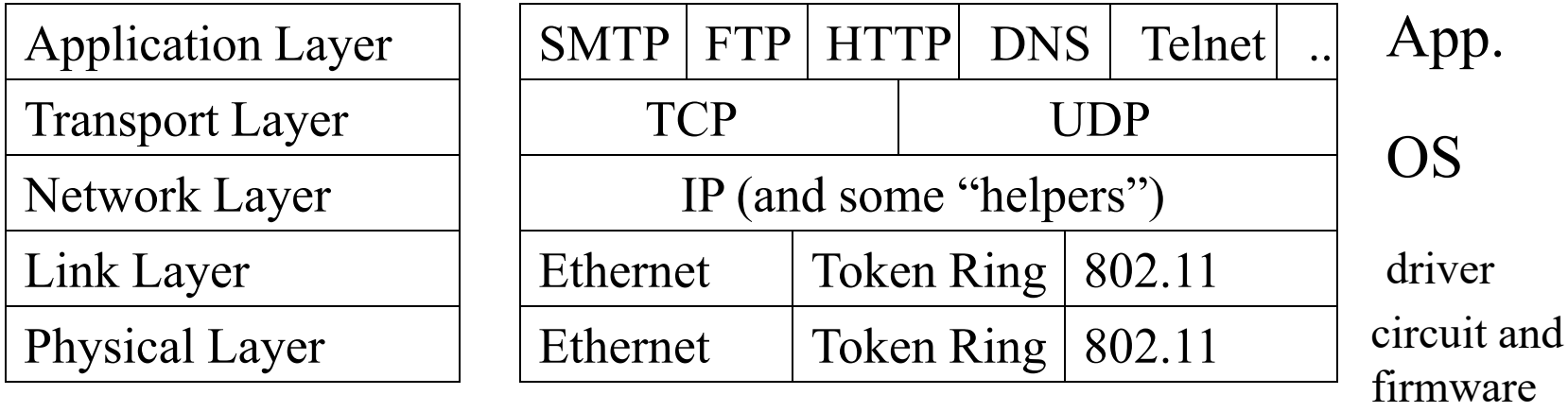
- The OSI reference model is a nice conceptual model
- There is no large scale implementation of an OSI protocol stack
- The OSI model is largely used by networking people (conceptually)

A Network Protocol Stack (Example)

Generalities

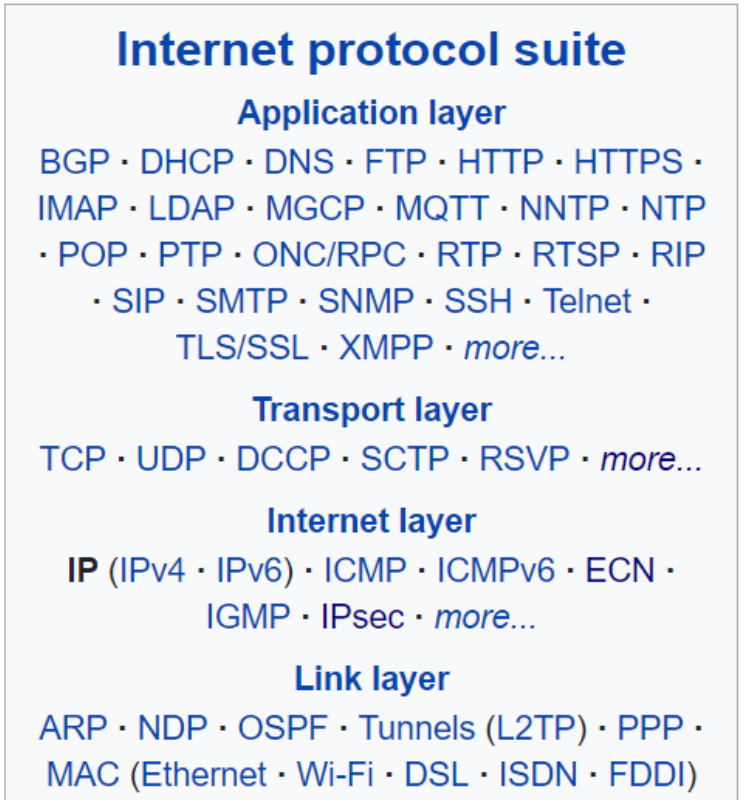
- Most existing protocol stacks have only 5 layers (set before OSI model):
  - TCP/IP
  - Novell IPX/SPX (for NetWare LAN)

TCP/IP


Application Layer
Transport Layer
Network Layer
Link Layer
Physical Layer
SMTP
FTP
HTTP
DNS
Telnet
..
TCP
UDP
IP (and some “helpers”)
Ethernet
Token Ring
802.11
Ethernet
Token Ring
802.11
App.
OS
driver
circuit and firmware

TCP/IP Protocol Stack

- **Application layer**: (why port number?)
  - SMTP: Simple Mail Transfer Protocol (Port 25)
  - FTP : File Transfer Protocol (Port 21)
  - Telnet : Terminal… (Port 23)
  - HTTP: HyperText Transfer Protocol (Port 80)
  - DNS : Domain Name System (port 53)
- **Transport layer**: (why protocol #?)
  - UDP: User Datagram Protocol (Protocol # 17)
  - TCP: Transmission Control P. (Protocol # 6)
- **Network layer**:
  - IP : Internet Protocol

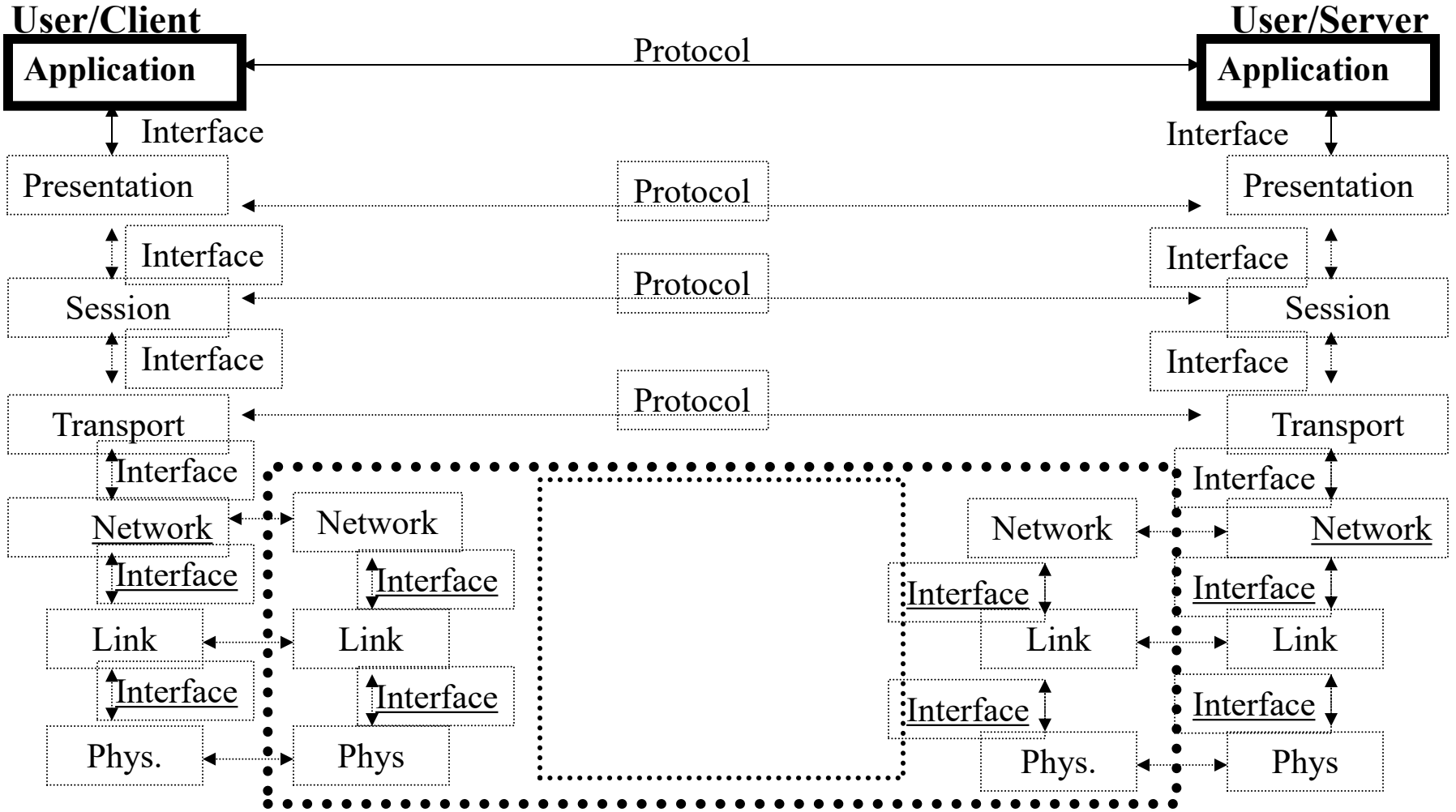


Schedule of Topics to be Studied

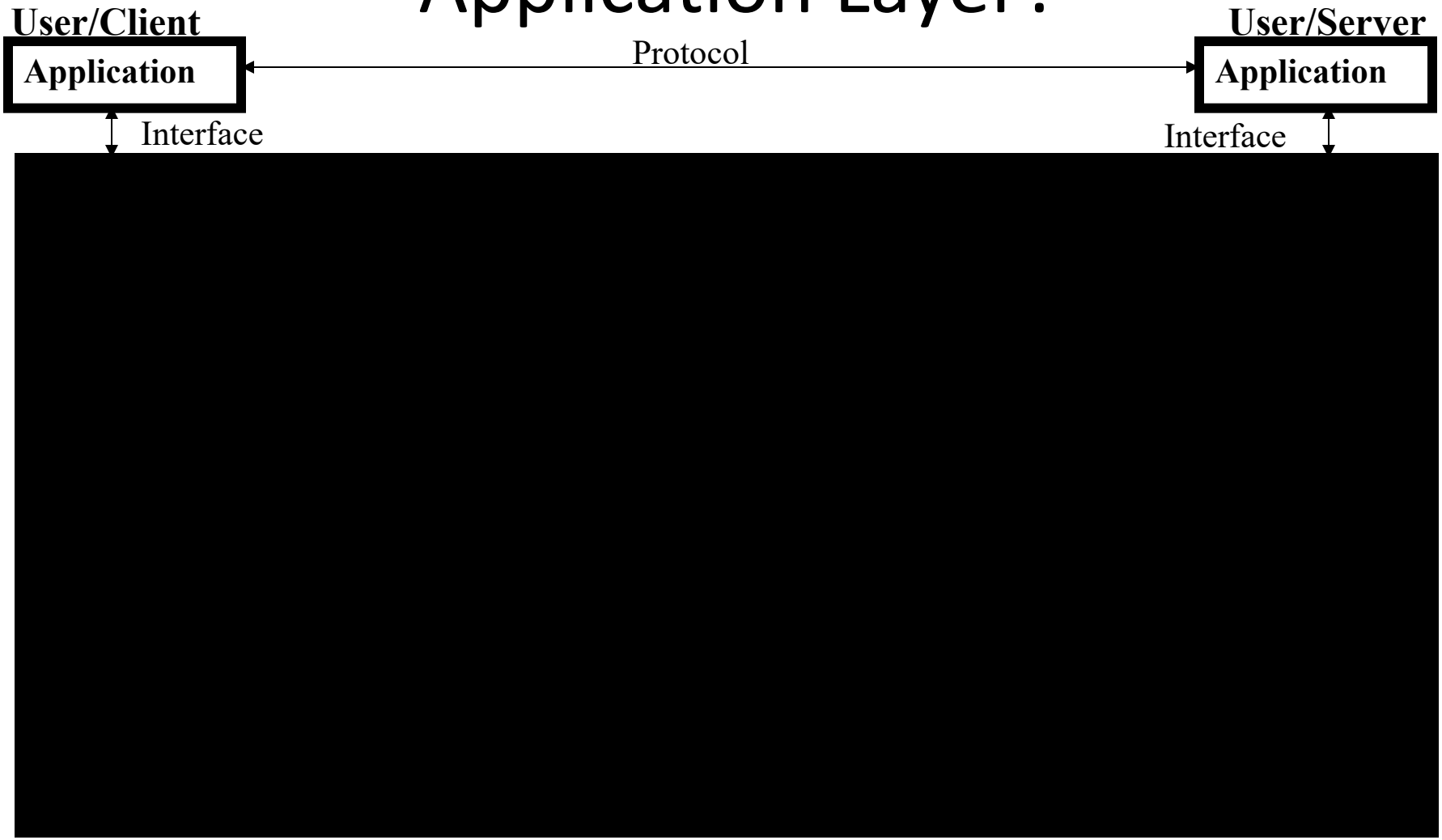
- Being able to develop your net applications quickly:
  - Application layers
  - TCP/UDP socket programming
- Solid foundation follows
  - Queueing theory
  - Link
  - Network
  - Transport

Application Layer

Where in the OSI Reference Model?



How Do Lower Layers Appear to the Application Layer?



Functions of the Application Layer

- Specialized service to the user
- Set services not offered by lower layers
- Improve services offered by lower layers but incomplete

Specialized Service To the User

- Virtual chat room
- Web browser
- File transfer
- Email
- Remote terminal
- Mapping host names to IP addresses
- ……

Set Services Not Offered by Lower Layers

- Want to set a reliable connectionless service over the internet that corrects only errors “recoverable” in less than 20ms (e.g., conference call such as Zoom):
  - Use TCP? Bad idea because of handshake, congestion control, of own retransmissions that cannot be controlled, coarse timer…
  - Moreover, TCP is connection-oriented
  - Use UDP? (connectionless indeed, but…)

Internet Applications Protocols:
DNS, Telnet, SMTP

Domain Name System (DNS)

RFC 1034 and RFC 1035

DNS: What ?

- It is a service used to map host names into IP addresses (aka resource record)
  - Based on a Client-Server architecture
  - Each machine connected to the net must know the IP address of its DNS server (how).

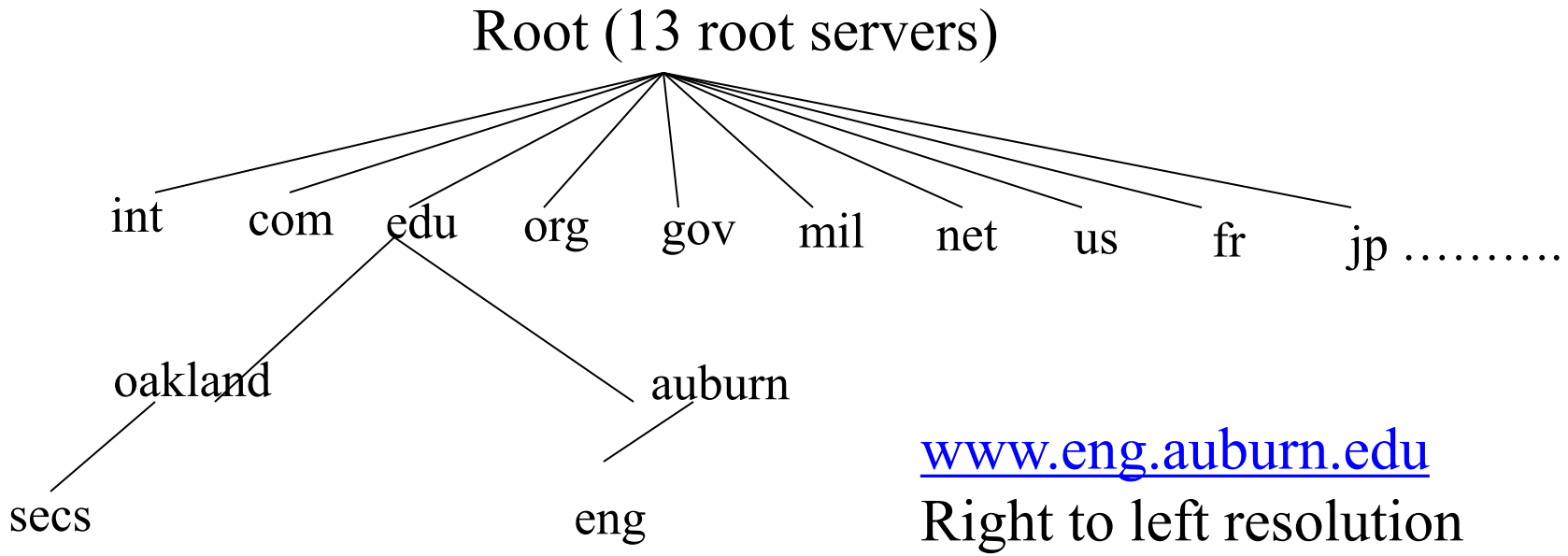
DNS: Why ?

- Human do not like to use IP addresses (as numbers)
- Human prefer to designate machines with names
- Interconnecting devices on the net use IP addresses (as numbers)
- Huge number of hosts on the Internet

DNS: How ?

- Use a file (e.g., hosts.txt) with all machine names and mapping?
- Use a local server that contains the database related to all hosts on the Internet?
- Use a distributed database system

DNS: A Hierarchical Distributed Database

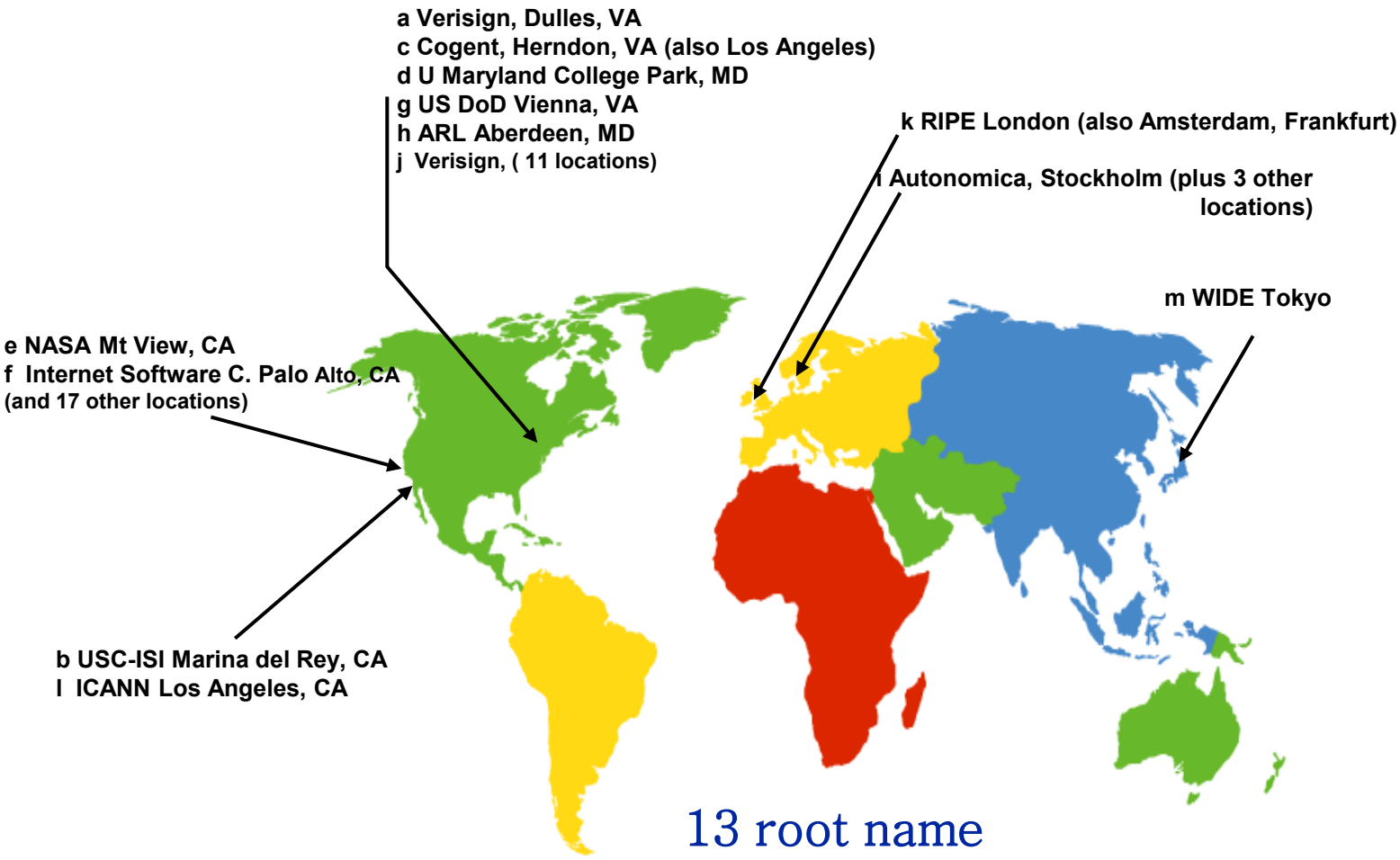


www.eng.auburn.edu

Right to left resolution

- Root server administered by the Internet Corporation for Assigned Names and Numbers (ICANN)
- Who administers the rest?

DNS: Root name servers



13 root name servers worldwide

Top level domains (TLDs)

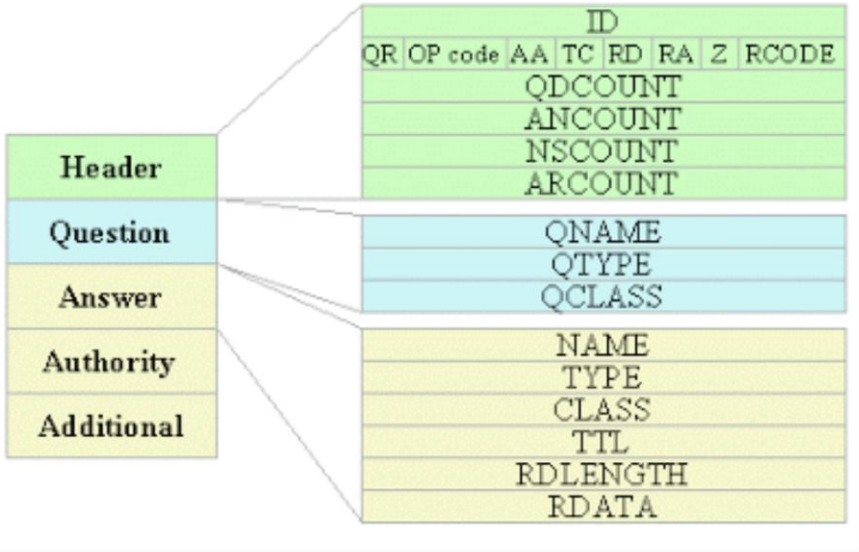
- **edu, gov, com, net, org, mil,** …
- Countries each have a top level domain (2 letter domain name).
- New top level domains include:

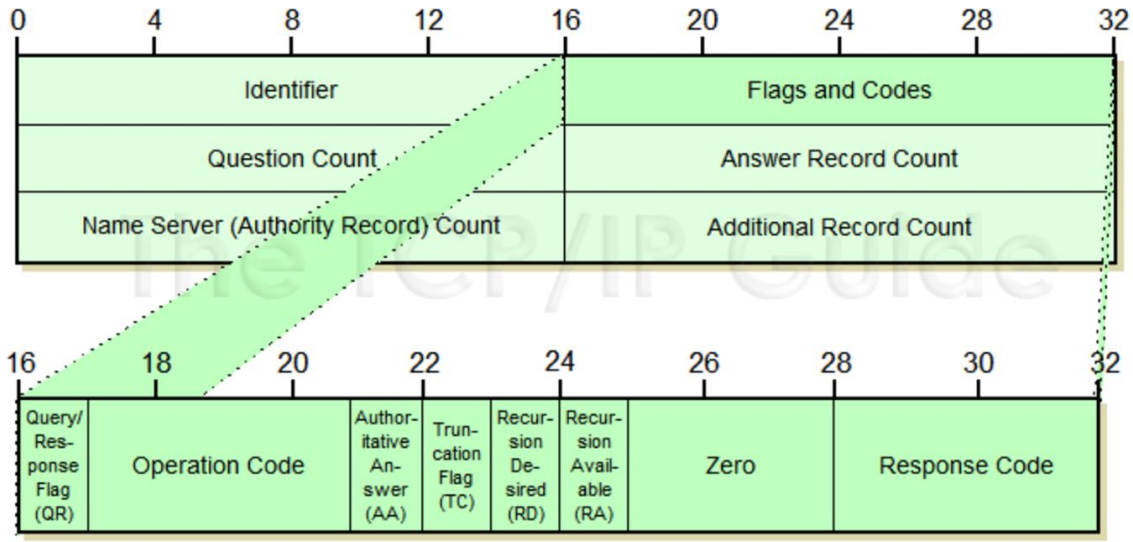
  .aero .biz .coop .info .name .pro

DNS Servers
(Port 53)

- At least two DNS servers per zone
  - A primary server (with a database)
  - One or more secondary servers (reliability and distribution of load)
  - List of resource record
    - <host name, ip address>
    - <hostname, name of another DNS server>
- A DNS server stores resource records of its domain, and also pointers to the DNS servers of its direct sub-domains

DNS packet structure





DNS Message Header Format

What Does the DNS Client Do?

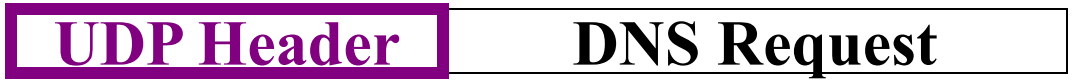
At the application layer (DNS)

- It prepares a name resolution request based on the DNS protocol
  - What is the IP address of “www.eng.auburn.edu”?

What Does the UDP Transport Agent Do?

At the transport layer (UDP)

- It encapsulates the DNS request with:
  - The DNS **client port number** (source)
  - The DNS **server port number** (53 destination)
  - Some extra information used by UDP



What Does the IP Network Agent Do?

At the network layer (IP)

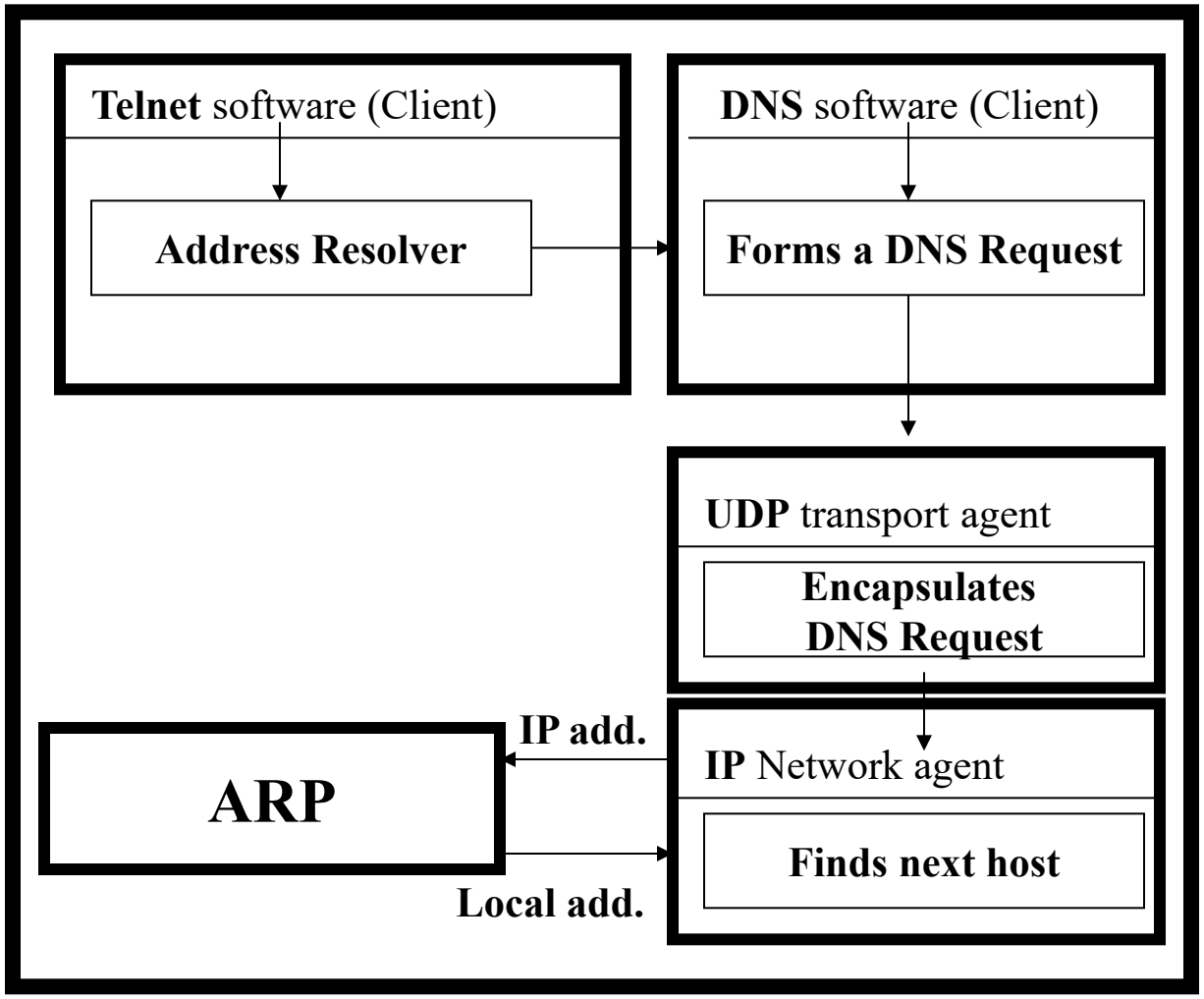
- IP **encapsulates** the UDP **datagram** with:
  - The DNS **client IP address** (configuration)
  - The DNS **server IP address** (configuration)
  - UDP stamp (The payload of the IP packet is a UDP datagram)
  - Some extra information used by IP
- It finds the next host to which to send the packet

What Does the IP Network Agent Do? (2)

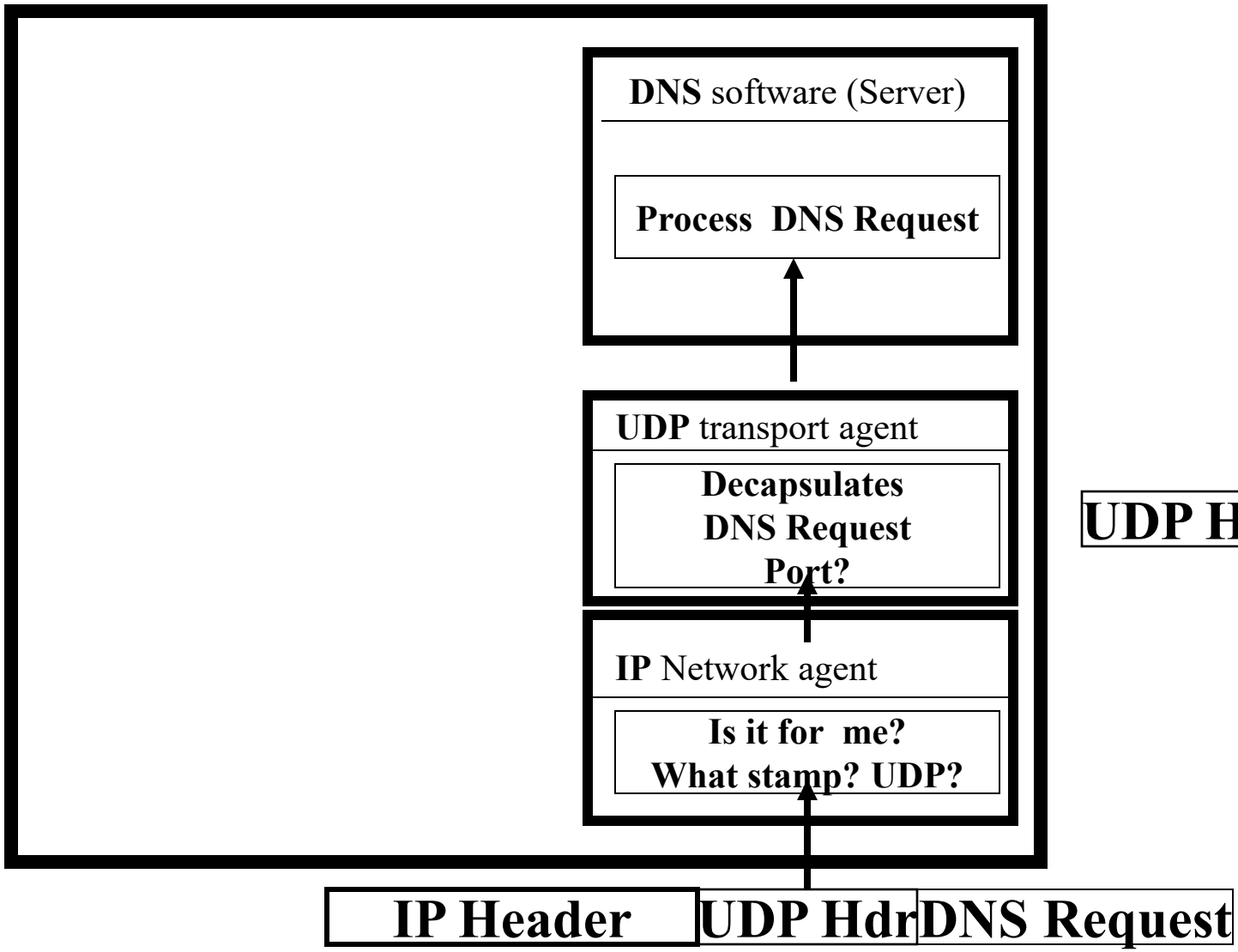
- To find the next host:
  - It checks if the final destination IP_D is local (attached to local segment)
  - If the final destination is not local:
    - Then IP looks into a routing table to find the **IP address** of next hop
    - Else… IP must find the local host that has IP address IP_D

How to Resolve IP Address → Local Address?

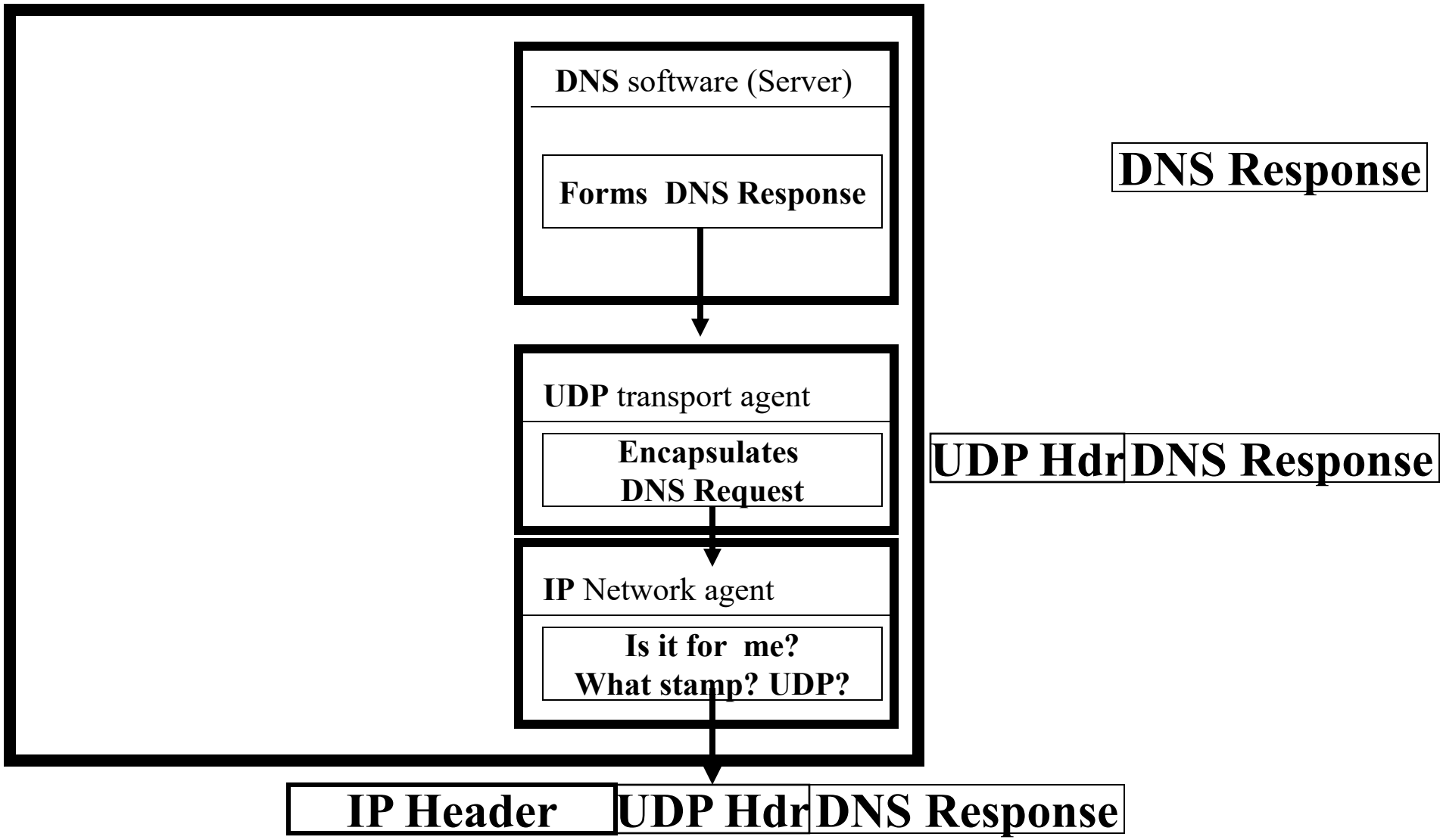
strangelove.eng.auburn.edu host



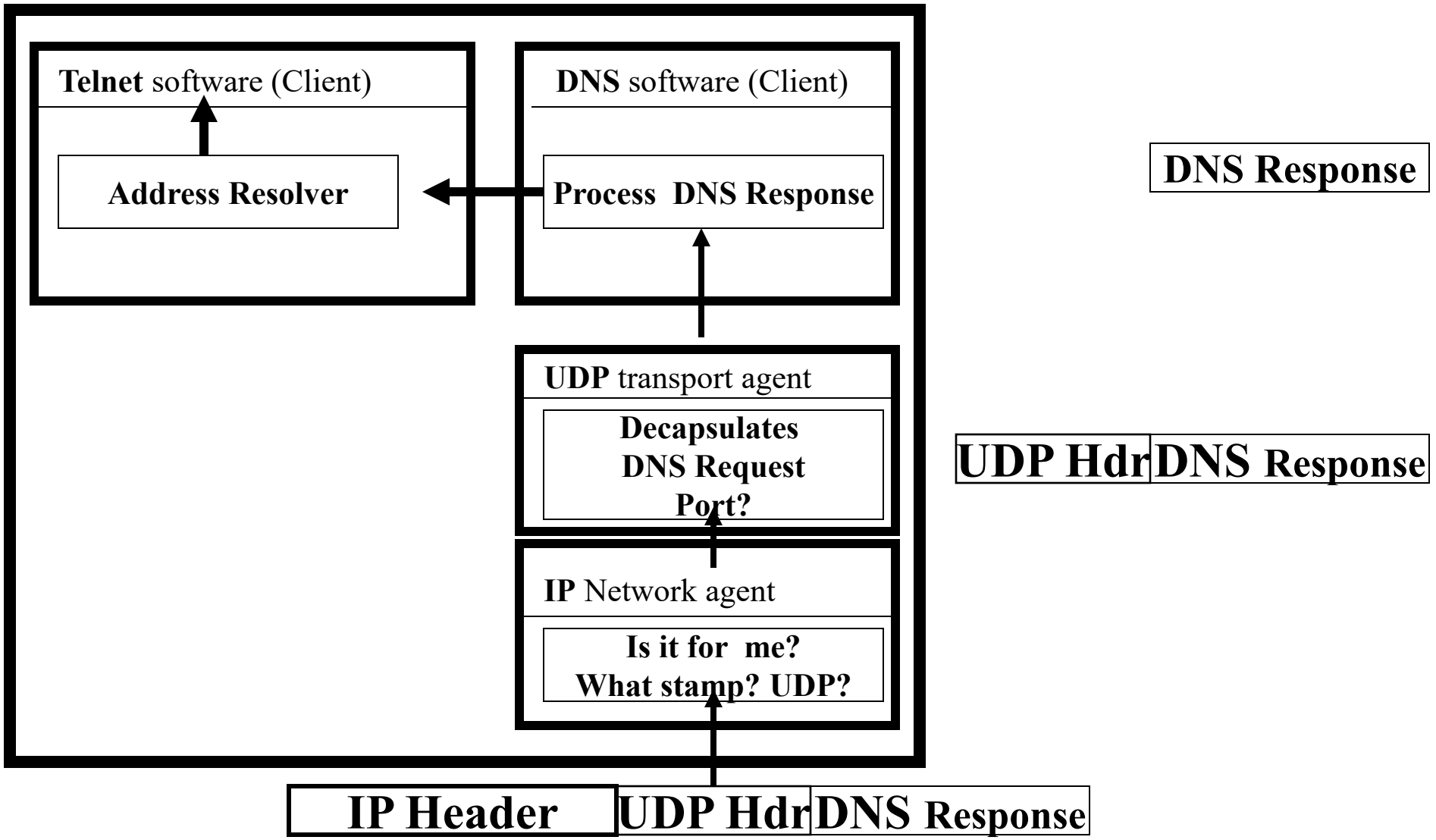
Query gets to the DNS Server



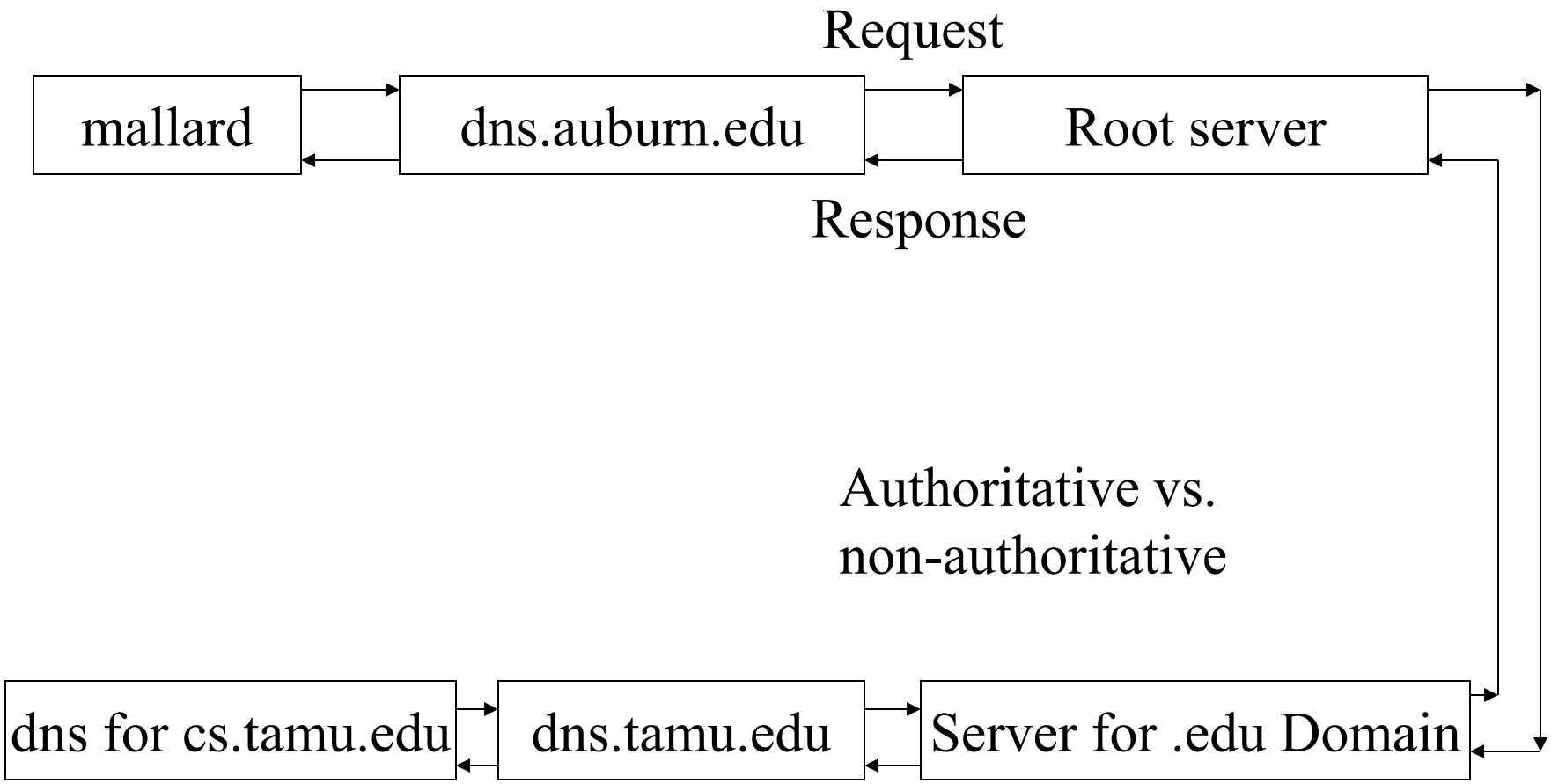
DNS Server Responds



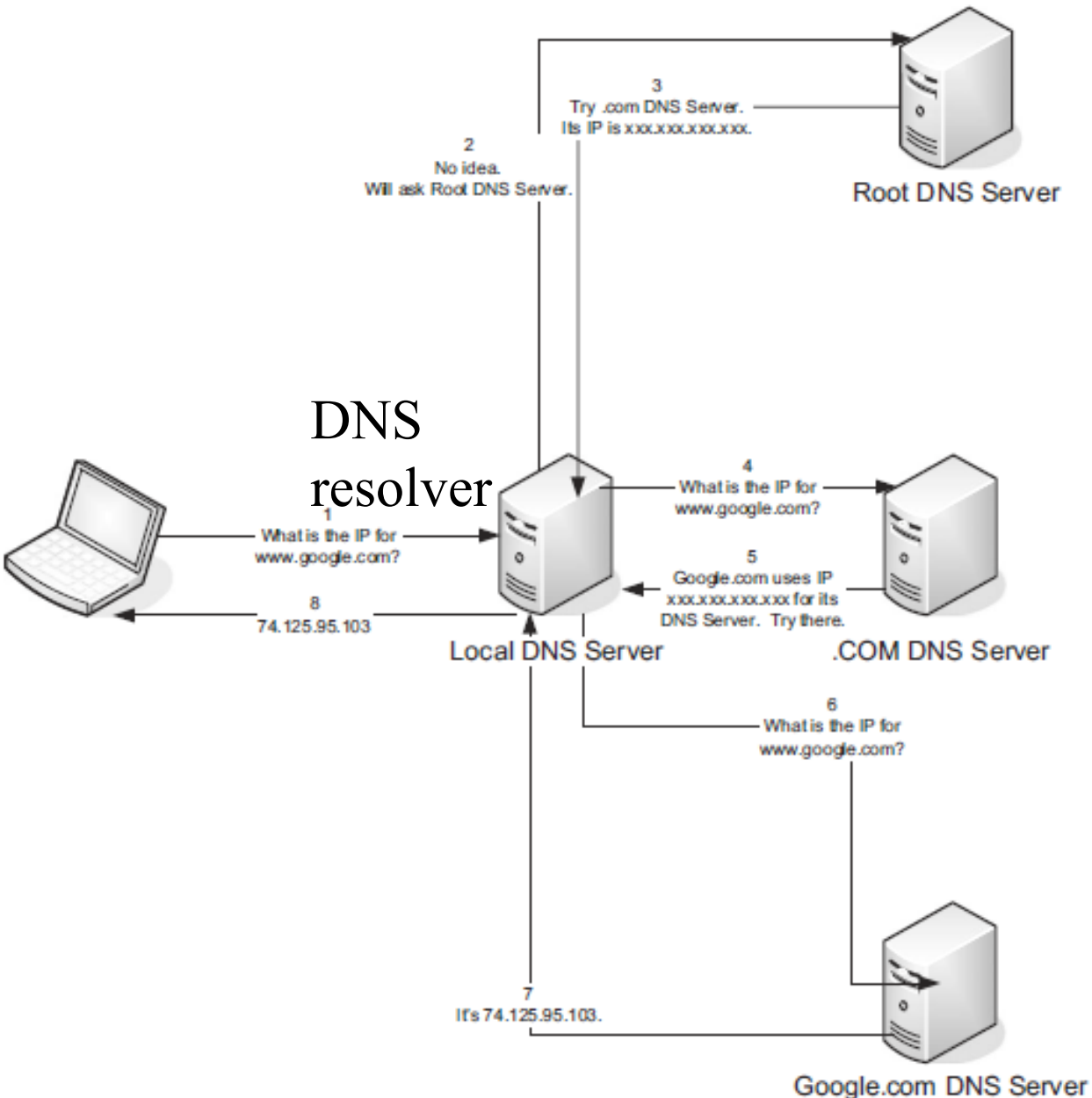
On Your Host


Telnet software (Client)
Address Resolver
DNS software (Client)
Process DNS Response
UDP transport agent
Decapsulates
DNS Request
Port?
IP Network agent
Is it for me?
What stamp? UDP?
DNS Response
UDP Hdr
DNS Response
IP Header
UDP Hdr
DNS Response

Requesting the IP of bach.cs.tamu.edu from mallard.eng.auburn.edu (Recursive)



Iterative DNS Query

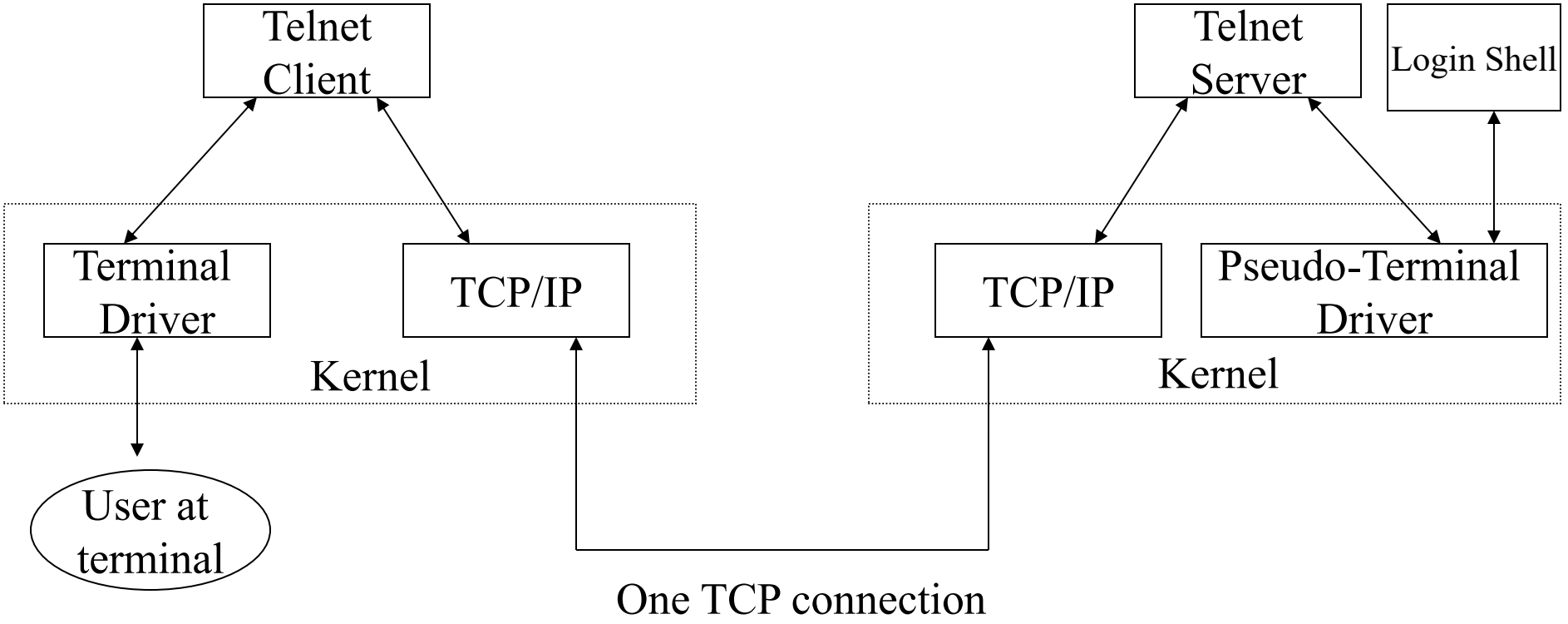


Iterative Query

Telnet

RFC 854

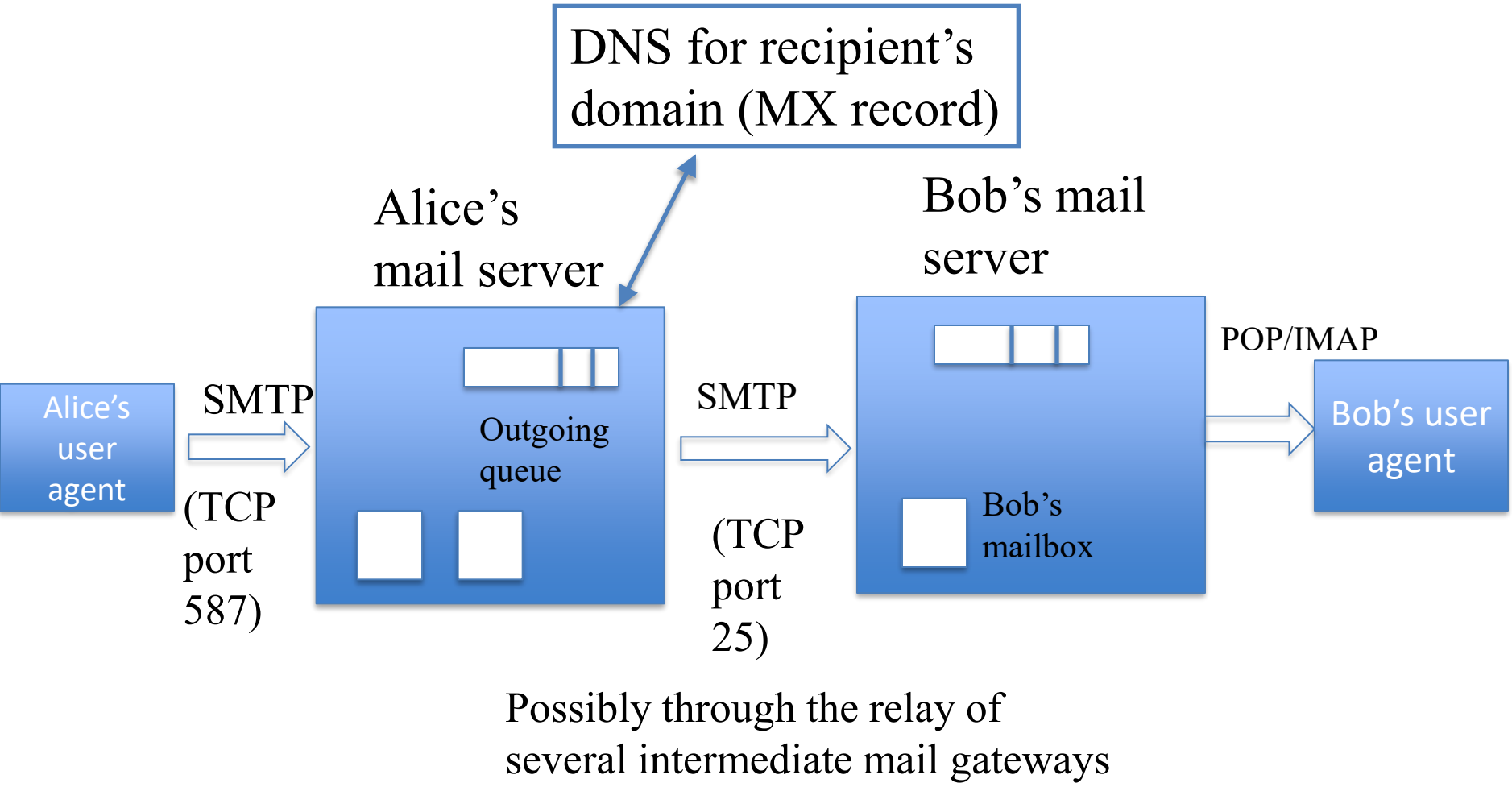
Telnet: the Big Picture
(Port 23)



SMTP

RFC 822

Electronic Mail



Conclusion

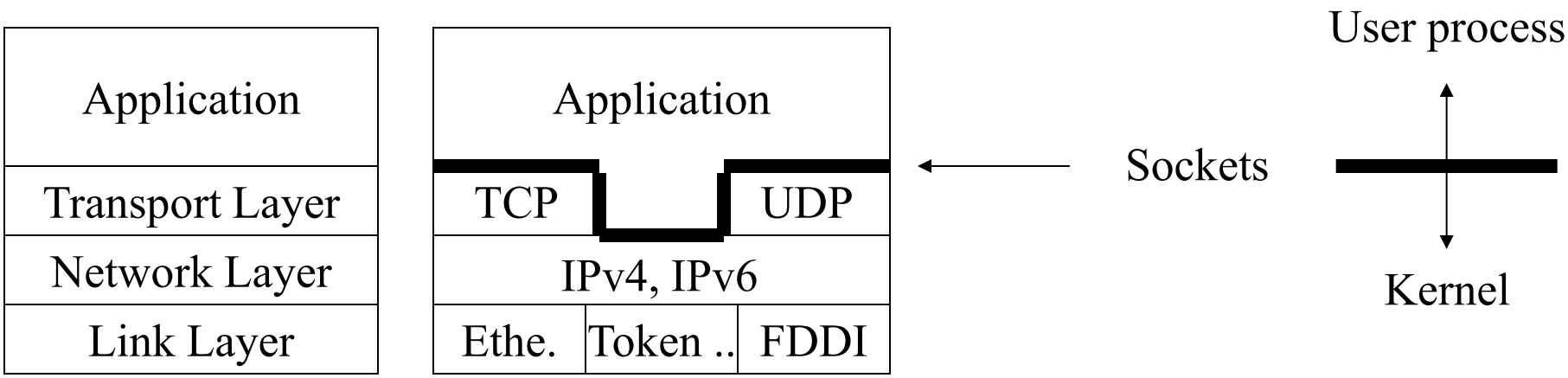
- You must know
  - The key Internet application protocols
  - Their functions
  - Port number binding

Socket Programming

Outlines

- Basic concept of sockets
- Client-server model
- TCP socket primitives
- Concurrent server design
- Network echo example
- UDP socket primitives

TCP/IP Model : Where are the Sockets?



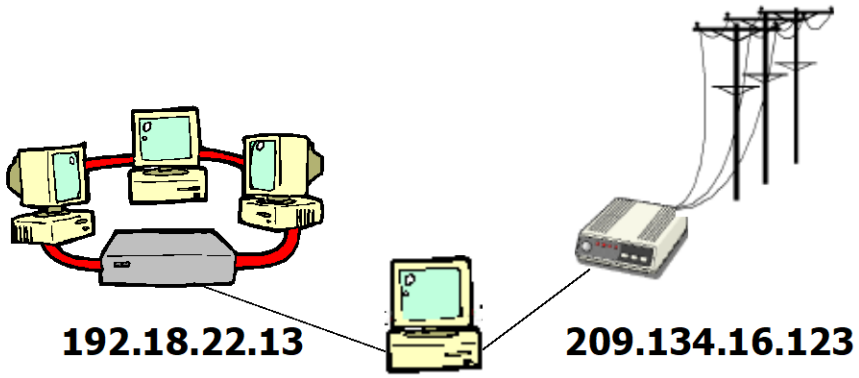
Transport Protocols: TCP and UDP

- TCP:
  - provides a reliable in-order byte stream
  - Adapts to network conditions
  - Adapts to receiver

- UDP:
  - Adds an extra layer over IP
  - No reliability is provided
  - Detects corruption (through checksum)

IP Address

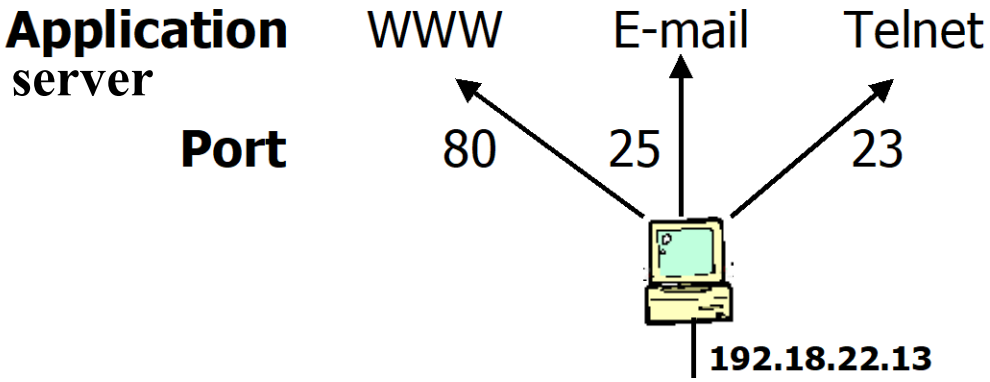
- 32-bit identifier
- Dotted-quad: 192.118.56.25
- www.mkp.com –> 167.208.101.28
- Identifies a host interface (not a host)

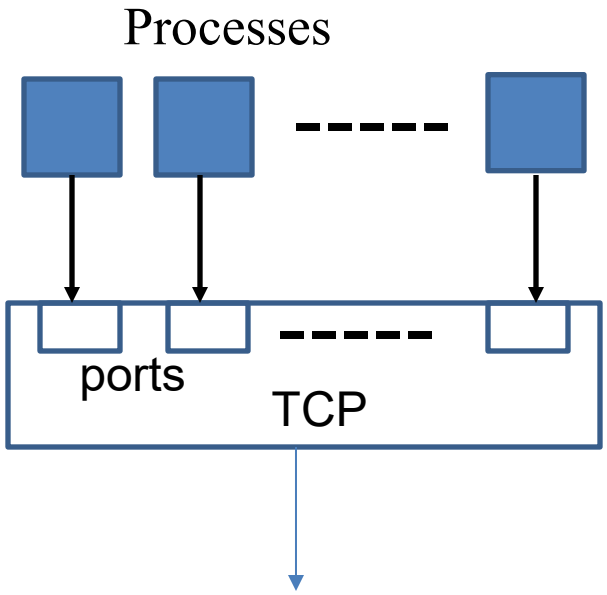


Ports

Identifying the ultimate destination process

- IP addresses identify hosts
- Host has many applications
- Ports (16-bit identifier)



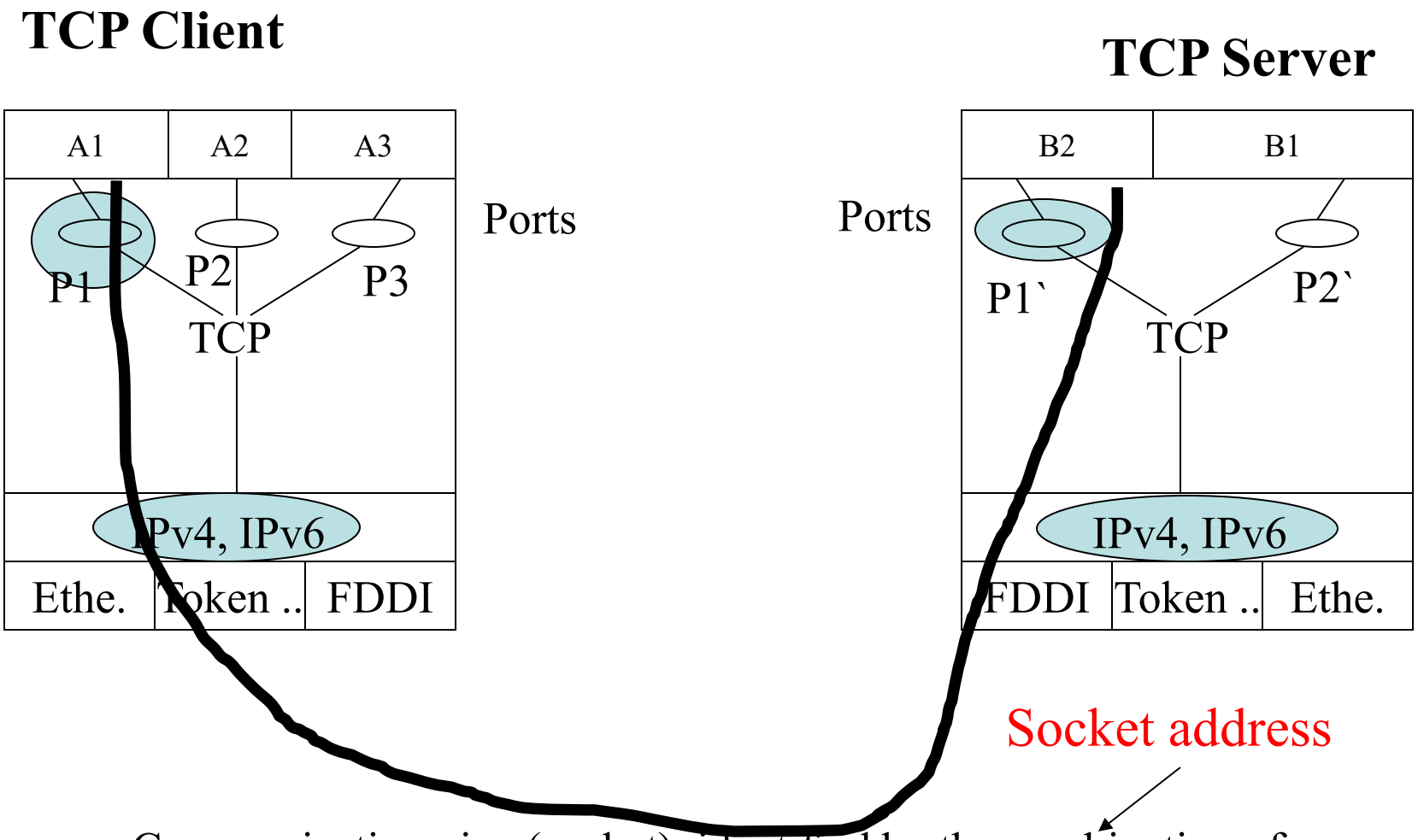


Client-side port # is user defined

Some Well Known Ports on Server

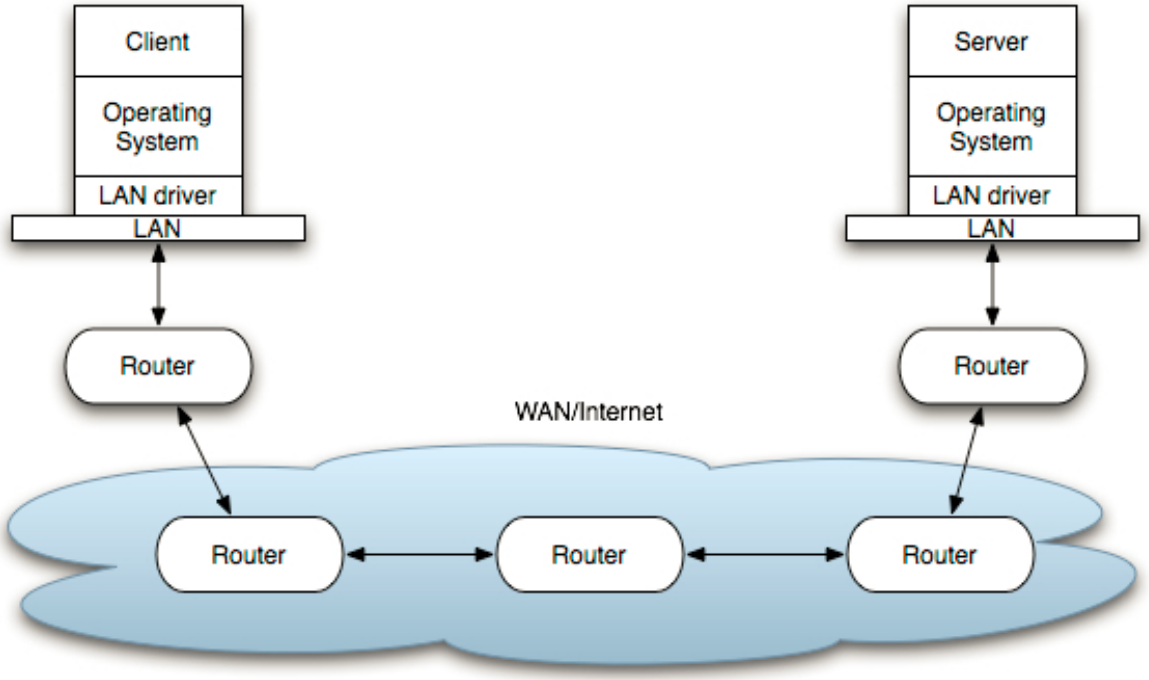
Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FTP, Data	File Transfer Protocol (data connection)
21	FTP, Control	File Transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

TCP Sockets



Communication pipe (socket): identified by the combination of source and destination IPs and port numbers and type of the socket (TCP/UDP)

Client-Server Model



- These are processes instead of machines (computers), they could be running on the same host!
- A client connects to and communicates with one server at a time
- A server may be communicating with several clients at the same time
- Server's IP address and port # is well known

TCP Protocol Briefing

- TCP Packet Header

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Source Port																Destination Port															
any																**any**															
TCP Sequence Number																															
sequence number																															
TCP Acknowledgement Number																															
acknowledgement number																															
Data Offset				Reserved						URG	ACK	PSH	RST	SYN	FIN	Window															
offset				*reserved*						**X**		**X**			**X**	*window*															
Checksum																Urgent Pointer															
checksum																*urgent pointer*															
TCP Options																												Padding			
TCP options																												*padding*			

TCP payload (application data)

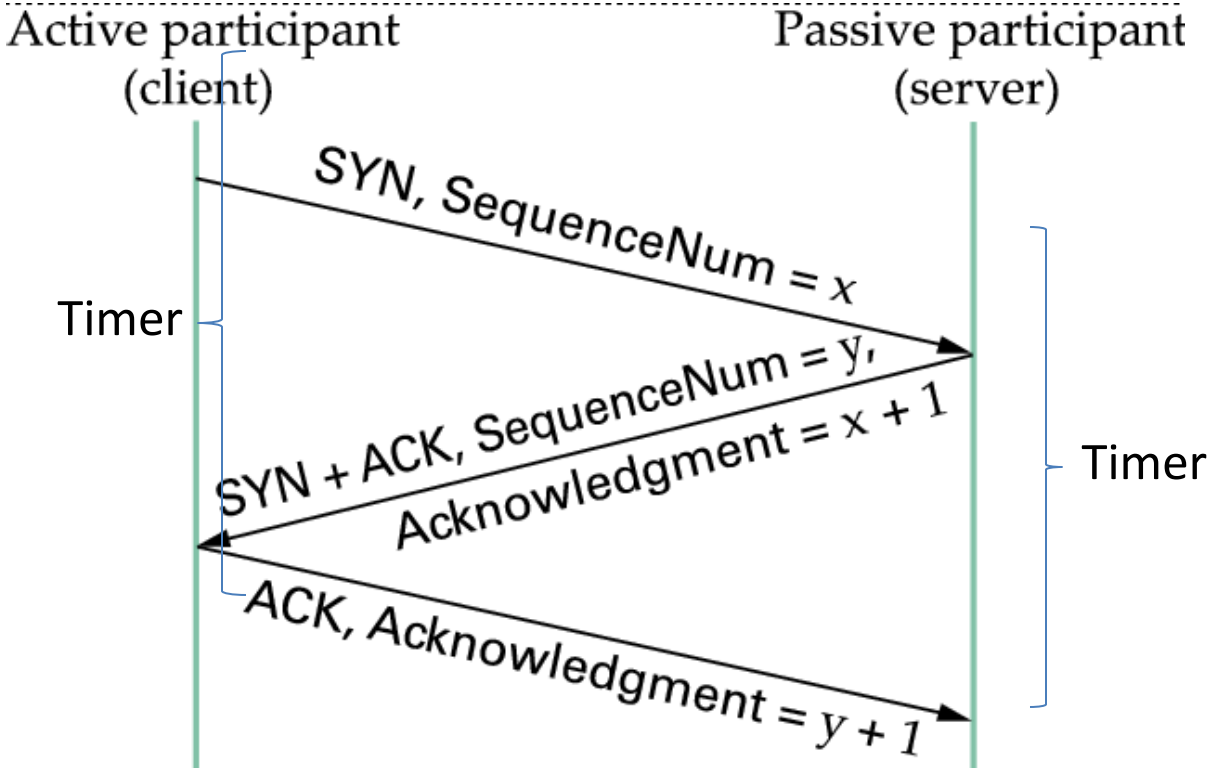
TCP Flag Definitions

Flag	
SYN	The beginning of a connection
ACK	Acknowledge receipt of a previous packet or transmission
FIN	Close a TCP connection
RST	Abort a TCP connection

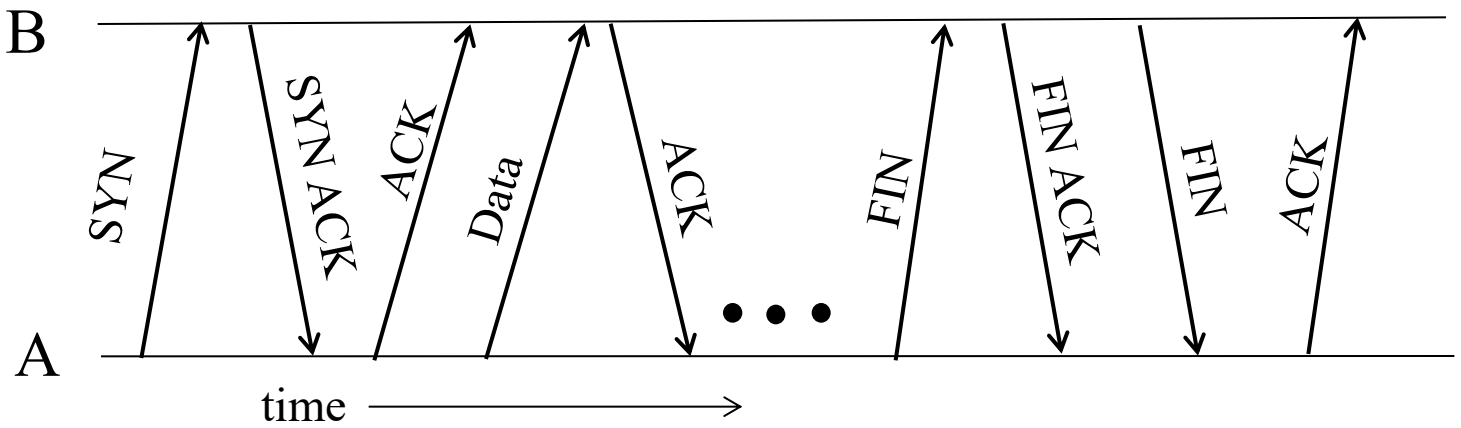
Connection Establishment (TCP Protocol)

Three-Way Handshake

Exchange of three messages



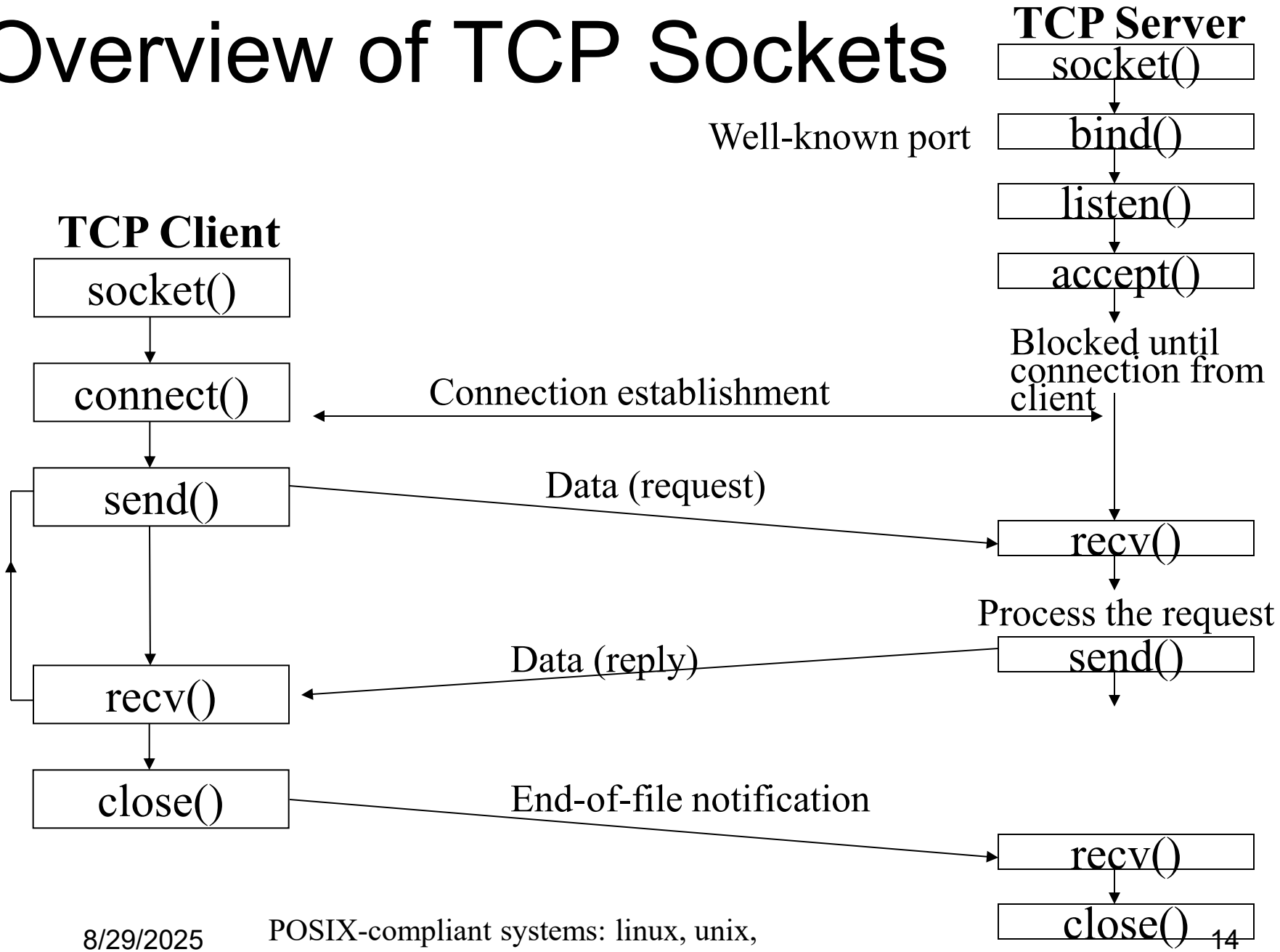
Entire TCP hand-shaking and data transaction procedure



Closing the connection

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

Overview of TCP Sockets



POSIX-compliant systems: linux, unix, MacOS, Cygwin under MS windows, etc.

Socket Primitives: Create a socket: socket()

```
#include <sys/socket.h>
int   socket(int family, int type, int protocol);
```

Parameters:

family (AF_INET, AF_INET6, AF_LOCAL…)
type (SOCK_STREAM, SOCK_DGRAM, SOCK_RAW)
protocol (0 except for raw sockets)

Result: non negative socket descriptor if OK, -1 otherwise

Socket Primitives: bind()

Function: assigns a local protocol address (IP address + port number) to a socket

#include <sys/socket.h>
int bind(int *sockfd*, const struct sockaddr **servaddr*,
socklen_t *addrlen*);

Parameters:

sockfd : file descriptor obtained thru socket() function
**servaddr* :pointer to a struct sockaddr.
addrlen length of the address structure

Result: 0 if OK, -1 otherwise

Socket address structure

- IPv4 socket address structure: #include<netinet/in.h>

```
struct in_addr{
in_addr_t s_addr; /*32 bit IPv4 network byte ordered address*/
};


struct sockaddr_in {
   uint8_t sin_len; /* length of structure (16)*/
   sa_family_t sin_family; /* AF_INET*/
   in_port_t sin_port; /* 16 bit TCP or UDP port number */
   struct in_addr sin_addr; /* 32 bit IPv4 address*/
   char sin_zero[8]; /* not used but always set to zero */
};
```

- Generic socket address structure

```
struct sockaddr {
   uint8_t sa_len;
   sa_family_t sa_family; /* address family: AD_xxx value */
   char sa_data[14];
};
```

- Type-cast IPv4 socket address to the generic socket address

Socket Primitives: listen()

Function: converts an unconnected socket into a passive socket.

#include <sys/socket.h>
int listen(int *sockfd*, int backlog);
Parameters:

sockfd : file descriptor obtained thru socket() function
backlog: maximum number of "waiting clients" in queue.

Result: 0 if OK, -1 otherwise

Socket Primitives: accept()

Function: A blocking operation, returns a descriptor
to a newly created socket that is connected with the client

```
#include <sys/socket.h>
int   accept(int sockfd, const struct sockaddr *cliaddr,
                    socklen_t  addrlen);
```

Parameters:

sockfd : descriptor of the listening socket of the server

**cliaddr* :pointer to the client's sockaddr (client issuing the connect()).

addrlen length of the address structure

Return: a new socket that is connected to the client

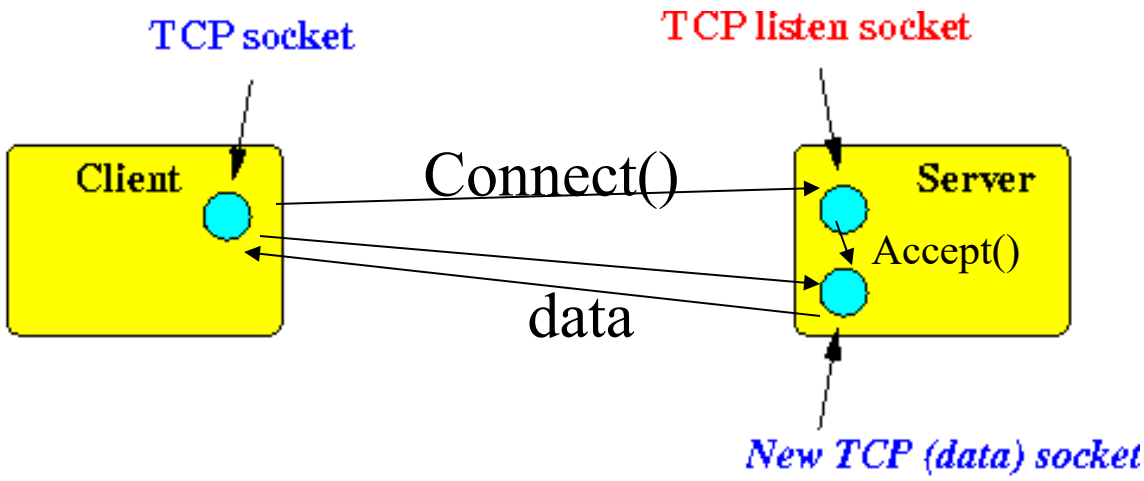
Client-side: connect()

- Used by a client to connect to a server

#include <sys/socket.h>
int connect(int *sockfd*, const struct sockaddr **servaddr*,
socklen_t *addrlen*);
Function: establishes a TCP connection
Parameters:
sockfd : file descriptor obtained thru socket() function
**servaddr* :pointer to a struct sockaddr (server's socket address).
addrlen length of the address structure

Result: 0 if OK, -1 otherwise

After the accept() call



Subsequent data transmission to and reception from the client should be based on the newly created data socket. The original listening socket stays on the listening port (concurrent server)

send()

```
#include <sys/socket.h>

ssize_t send(int sockfd, const void *buf, size_t nbytes, int flags)
```

Function: send (application) data through the socket

Parameters: sockfd: socket descriptor for sending data
buf: pointer to the buffer holding the data to send
nbytes: # of bytes in the buffer to be sent
flags: type of message transmission (0 in our class)

Return: # of bytes sent if call is successful; otherwise -1.

recv()

#include <sys/socket.h>

ssize_t recv(int sockfd, void *buf, size_t nbytes, int flags)

Function: receive data on the socket (blocking)

Parameters: sockfd: socket descriptor for receiving data
buf: pointer to the buffer holding the data received
nbytes: buffer size in bytes
flags: type of message transmission (0 in our class)

Return: # of bytes received if call is successful; otherwise -1.

Socket Primitives: close()

#include <sys/socket.h>
int close(int *sockfd*);
Function: closes the socket and terminates the TCP connection
Parameters:
sockfd : socket descriptor

Result: 0 if OK, -1 otherwise

A network echo example

- Client accepts user's input (a string), then sends this string to the Server
- Server receives the string on port 3003, displays it, then sends it back (echo) to the client
- Client receives the echoed string, and then displays it.

Server Implementation

//echoServer.c

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>

#define MAXLINE 4096 /*max text line length*/
#define SERV_PORT 3003 /*port*/
#define LISTENQ 8 /*maximum number of client connections */

int main (int argc, char **argv)
{
 int listenfd, connfd, n;
 socklen_t clilen;
 char buf[MAXLINE]; //buffer for data send and recv
 struct sockaddr_in cliaddr, servaddr;//server and client socket addresses
```

```
//creation of the socket
listenfd = socket (AF_INET, SOCK_STREAM, 0);

//preparation of the socket address
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY); //kernel chooses source ip address
                                                (any interface of the host)
servaddr.sin_port = htons(SERV_PORT); //host to network byte order conversion

bind(listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr));

listen(listenfd, LISTENQ);

printf("%s\n","Server running...waiting for connections.");
```

```
for ( ; ; ) {

  clilen = sizeof(cliaddr);
  connfd = accept(listenfd, (struct sockaddr *) &cliaddr, clilen);//get the new socket
  printf("%s\n","Received request...");

  while ( (n = recv(connfd, buf, MAXLINE,0)) > 0)  { //start receiving on the data sock
   printf("%s","String received from and resent to the client:");
   buf[n]=0;  //ending flag for the string
   puts(buf);
   send(connfd, buf, n, 0); //echo it back to the client
  }

 if (n < 0) {
  perror("Read error");
  exit(1);
 }
 close(connfd); //finish communication with one client; go back and wait for next client

 }
 //close listening socket
 close(listenfd);
}
```

Client Implementation

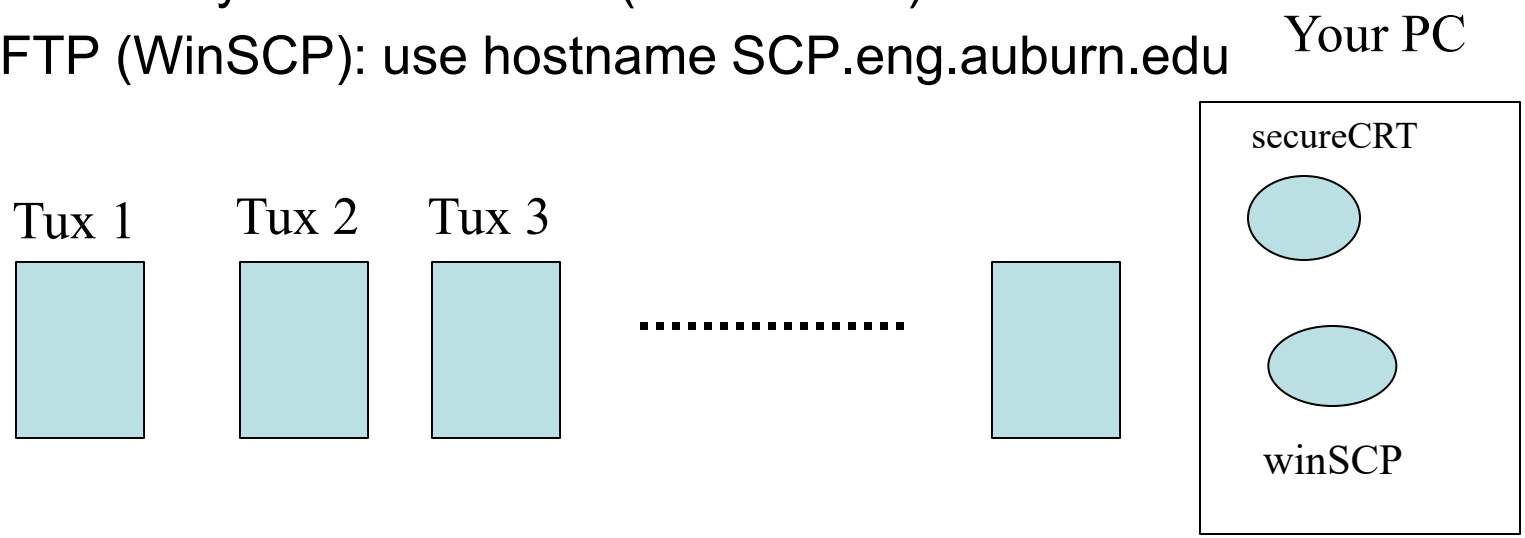
```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <arpa/inet.h>

#define MAXLINE 4096 /*max text line length*/
#define SERV_PORT 3003 /*port*/

int main(int argc, char **argv) //user specifies server ip address in command line
{
 int sockfd;
 struct sockaddr_in servaddr;
 char sendline[MAXLINE], recvline[MAXLINE];

 //basic check of the arguments
 //additional checks can be inserted
 if (argc !=2) {
  perror("Usage: echoClient IP address of the server");
  exit(1);
 }
```

```
//Create a socket for the client
 //If sockfd<0 there was an error in the creation of the socket
 if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) <0) {
  perror("Problem in creating the socket");
  exit(2);
 }

 //Creation of the socket address
 memset(&servaddr, 0, sizeof(servaddr));
 servaddr.sin_family = AF_INET;
 servaddr.sin_addr.s_addr= inet_addr(argv[1]); //convert number_dot string to binary
 servaddr.sin_port =  htons(SERV_PORT); //convert to network byte order

 //Connection of the client to the socket
 if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr))<0) {
  perror("Problem in connecting to the server");
  exit(3);
 }
```

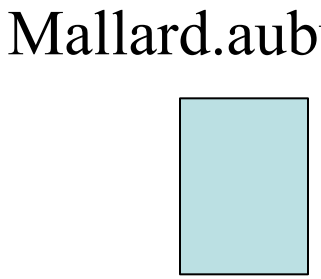
```
while (fgets(sendline, MAXLINE, stdin) != NULL) {

  send(sockfd, sendline, strlen(sendline), 0); // send input to server

  if (recv(sockfd, recvline, MAXLINE,0) == 0){ //waiting to receive echo from server
   //error: server terminated prematurely
   perror("The server terminated prematurely");
   exit(4);
  }
  printf("%s", "String received from the server: ");
  fputs(recvline, stdout);
 }

fclose(sockfd);
exit(0);
}
```

Project Development Environment

- Tux unix server cluster in CoE: port.eng.auburn.edu
  - Access by virtual terminal (secureCRT): the above
  - FTP (WinSCP): use hostname SCP.eng.auburn.edu



- Mallard server of AU: mallard.auburn.edu (the same hostname for virtual terminal and ftp)

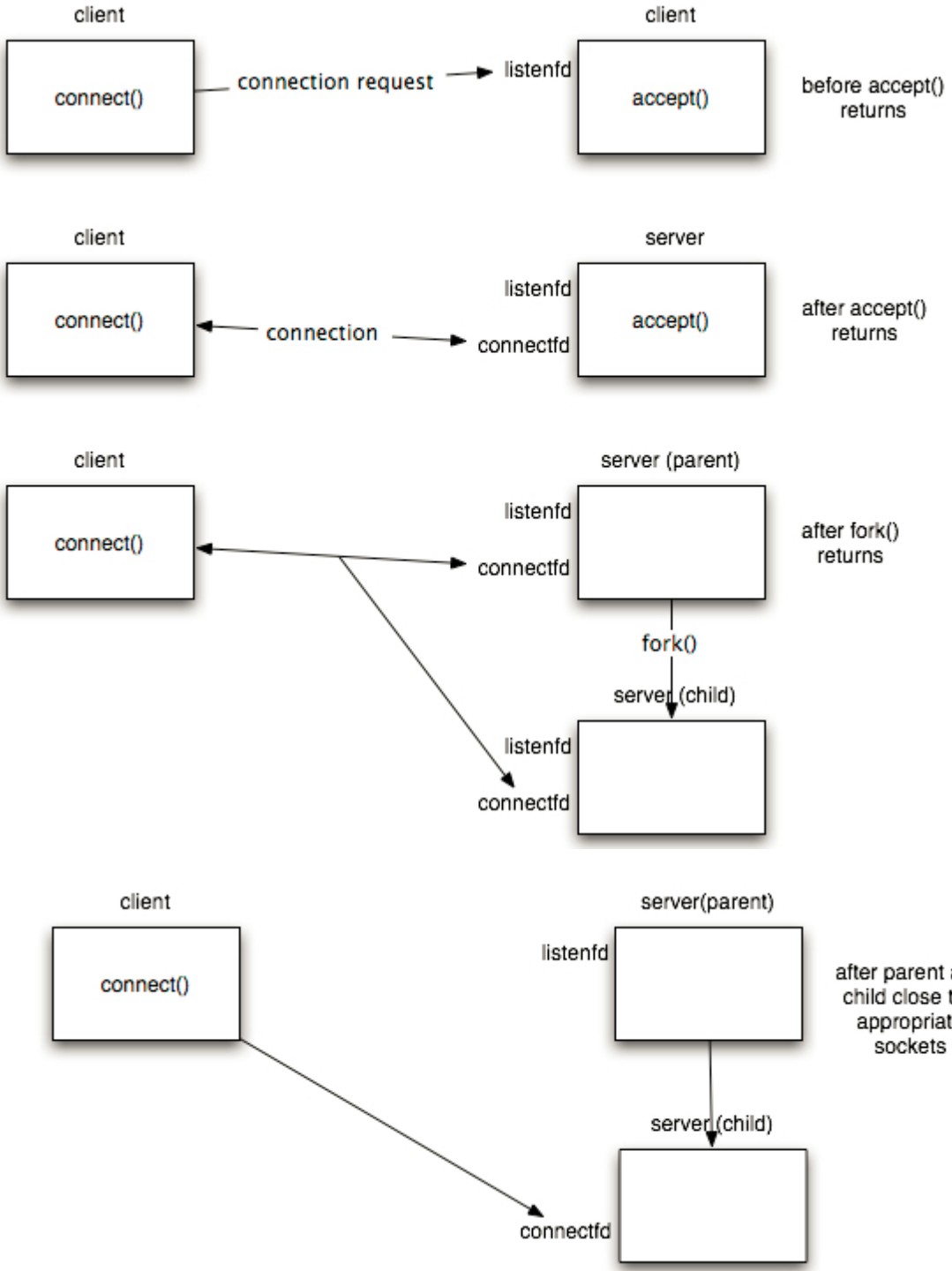


Concurrent Server

- Deals with multiple client connections at the same time
- Use the fork() to create child processes (#include<unist.h>)
- Fork() is called once, but return twice
  - Once in the calling process (parent) with return of the child id
  - Another in the child process with return of 0
  - Specify operation in child process or in parent process based on the returned value

Framework

```
pid_t pid;
int listenfd, connfd;
listenfd = socket(...);

/***fill the socket address with server's well known port***/

bind(listenfd, ...);
listen(listenfd, ...);

for ( ; ; ) {

   connfd = accept(listenfd, ...); /* blocking call */

   if ( (pid = fork()) == 0 ) {

      close(listenfd); /* child closes listening socket */

      /***process the request doing something using connfd ***/
      /* ................. */

      close(connfd);
      exit(0);  /* child terminates
    }
    close(connfd);  /*parent closes connected socket*/
}
}
```


client
client
connect()
connection request
listenfd
accept()
before accept()
returns
client
server
connect()
connection
listenfd
connectfd
accept()
after accept()
returns
client
server (parent)
connect()
listenfd
connectfd
after fork()
returns
fork()
server (child)
listenfd
connectfd
client
server(parent)
connect()
listenfd
after parent and
child close the
appropriate
sockets
server (child)
connectfd

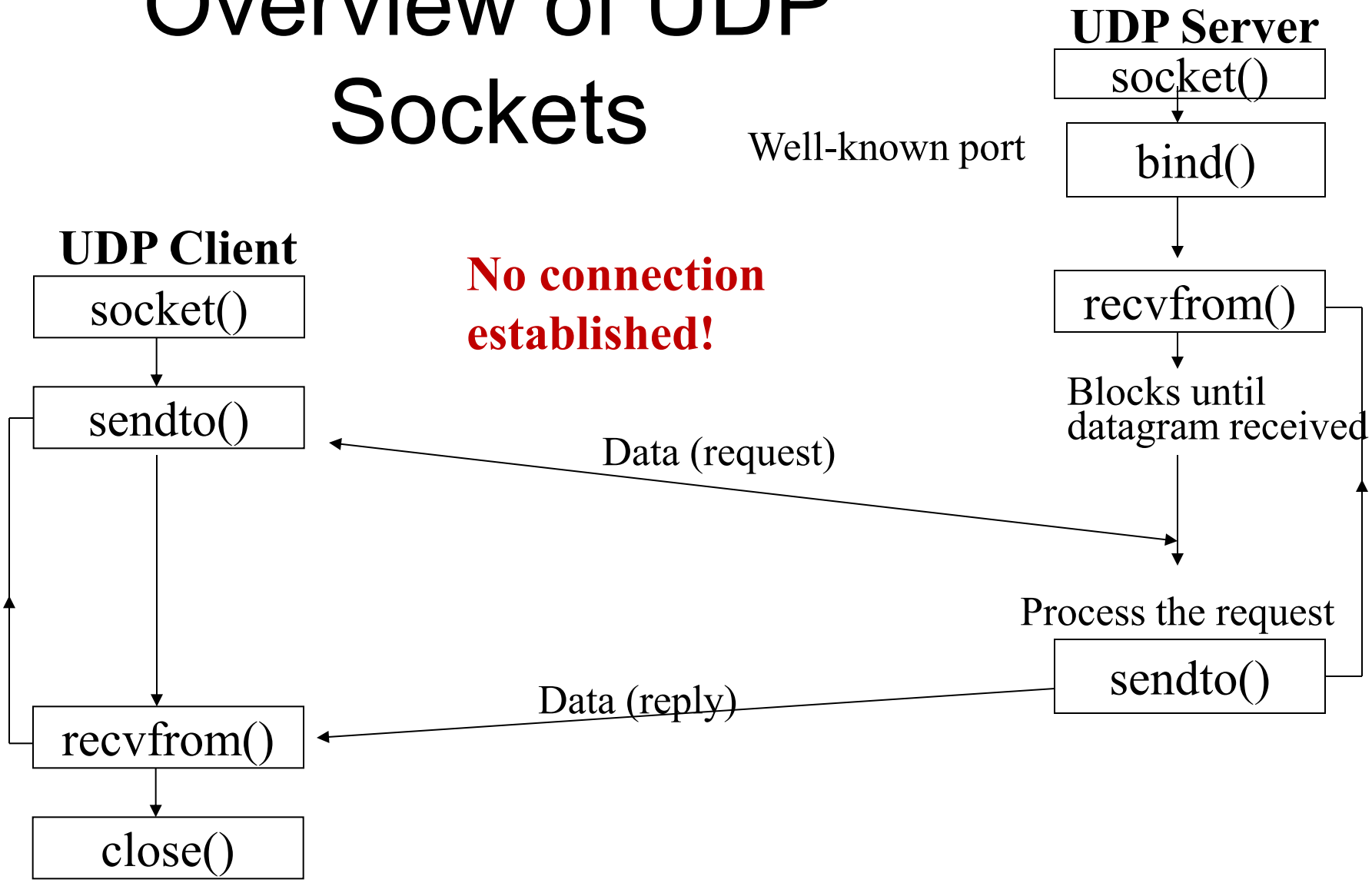
Concurrent Server Example

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#define MAXLINE 4096 /*max text line length*/
#define SERV_PORT 3003 /*port*/
#define LISTENQ 8 /*maximum number of client connections*/

int main (int argc, char **argv)
{
 int listenfd, connfd, n;
 pid_t childpid;
 socklen_t clilen;
 char buf[MAXLINE];
 struct sockaddr_in cliaddr, servaddr;

 //Create a socket for the server
 //If sockfd<0 there was an error in the creation of the socket
 if ((listenfd = socket (AF_INET, SOCK_STREAM, 0)) <0) {
  perror("Problem in creating the socket");
  exit(2);
 }
```

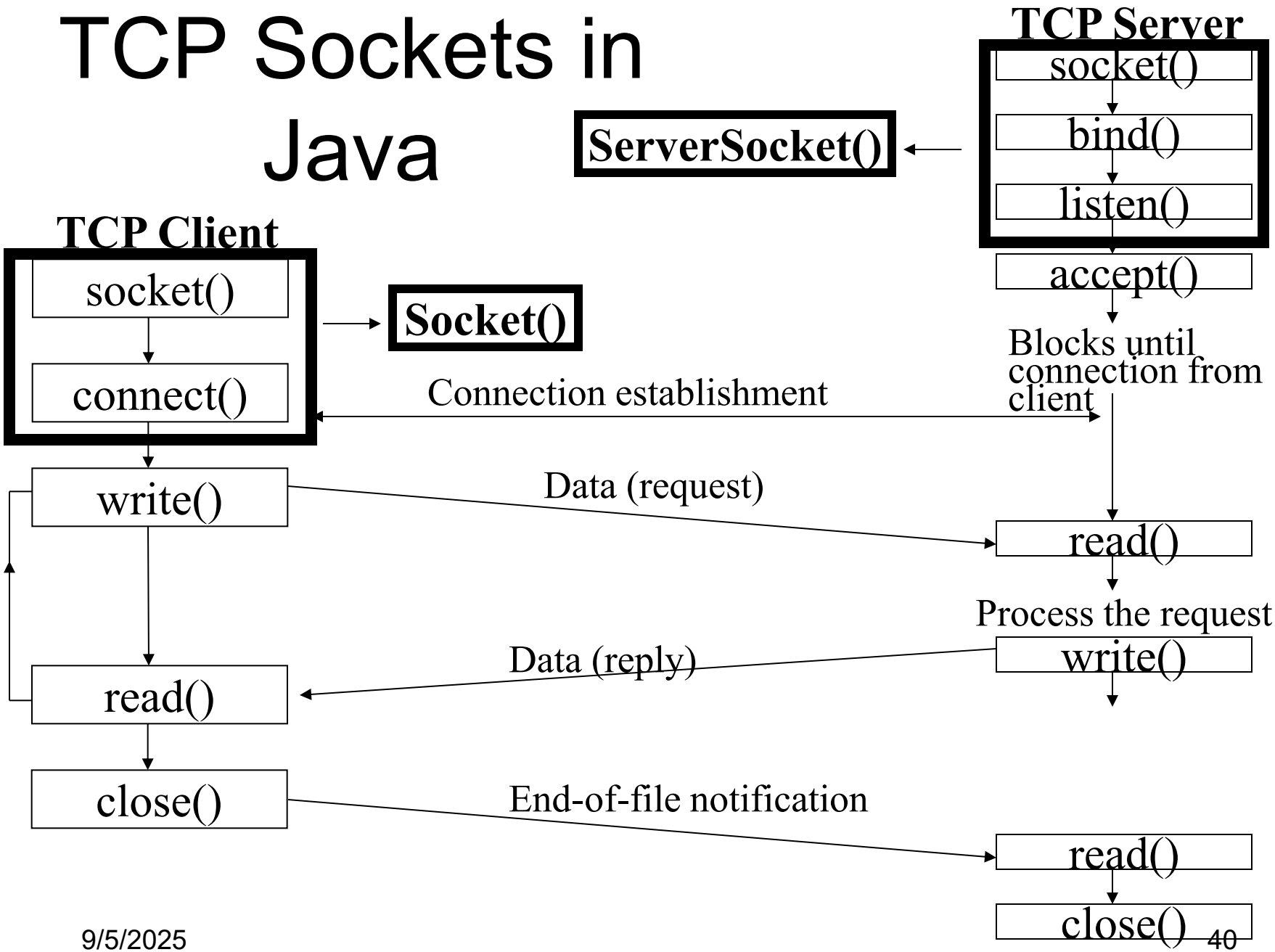
```
 //preparation of the socket address
 servaddr.sin_family = AF_INET;
 servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
 servaddr.sin_port = htons(SERV_PORT);

 //bind the socket
 bind (listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr));

 //listen to the socket by creating a connection queue, then wait for
clients
 listen (listenfd, LISTENQ);

 printf("%s\n","Server running...waiting for connections.");
```

```
 for ( ; ; ) {

  clilen = sizeof(cliaddr);
  //accept a connection
  connfd = accept (listenfd, (struct sockaddr *) &cliaddr, &clilen);//blocked, wait for connection

  printf("%s\n","Received request...");

  if ( (childpid = fork ()) == 0 ) {//if it is 0, it is the child process

    printf ("%s\n","Child created for dealing with client requests");

    //close listening socket
    close (listenfd);

    while ( (n = recv(connfd, buf, MAXLINE,0)) > 0)  {
     printf("%s","String received from and resent to the client:");
     puts(buf);
     send(connfd, buf, n, 0);
    }

    if (n < 0)
      printf("%s\n", "Read error");
    exit(0);  // child exit
}
  //close socket of the server
  close(connfd);
 }
}
```

Overview of UDP Sockets



TCP Sockets in Java



Client Socket Primitives Socket()

import java.net

Socket(**InetAddress** remoteaddr, **int** remotePort);

Socket(**String** remoteHost, **int** remotePort);

Socket(**InetAddress** remoteaddr, **int** remotePort,
InetAddress localaddr, **int** localPort);

Socket(**String** remotehost, **int** remotePort,
InetAddress localaddr, **int** localPort);

Server Socket Primitives
Socket()

import java.net

ServerSocket(**int** localPort);

ServerSocket(**int** localPort, **int** queuelimit);

ServerSocket(**int** localPort, **int** queuelimit,
InetAddress localaddr);

Conclusion

- A very, very fast overview of TCP and UDP sockets
- There are many “side” functions (to discover thru examples and programming tears and misery)

Read the simple, easy-to-understand tutorial
http://beej.us/guide/bgnet/

Advanced topics on Unix IPC (interprocess communication), e.g., shared memory, mmap, signal, pipe, etc.
https://beej.us/guide/bgipc/html/multi/index.html

Multi-threaded programming
http://www.cs.kent.edu/~ruttan/sysprog/lectures/multi-thread/multi-thread.html

- Best way: create a simple client and a simple server, then improve them step by step.

Error Control

Outline

- BER – PER conversion
- Error detection and correction by coding (FEC: forward error correction)
- Automatic repeat request (ARQ) schemes

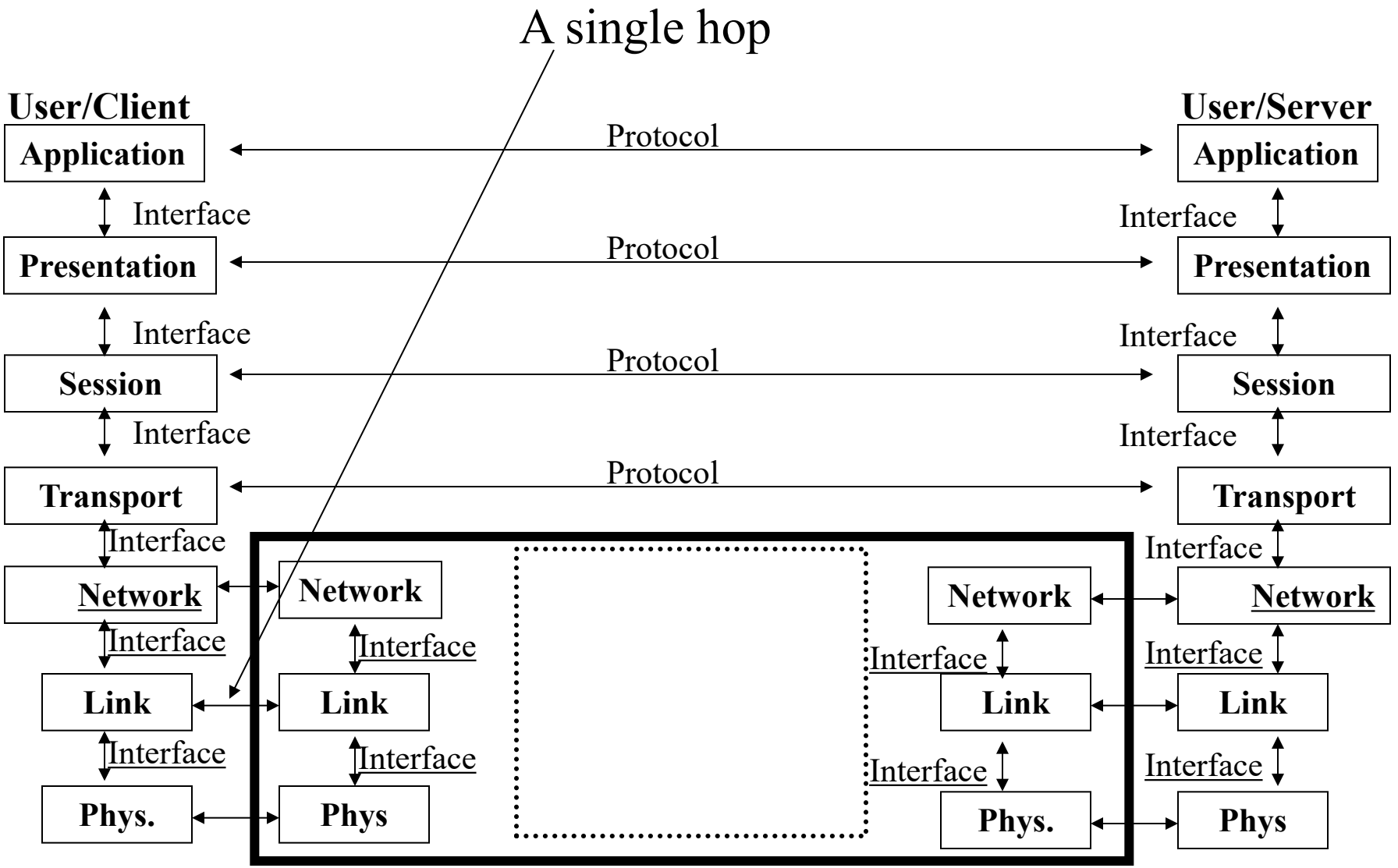
Error Control

- Why? Signal may get corrupted and packets may be dropped
- What? Technique(s) used to enable reliable delivery of data to a destination.
- How?

Error Control (Where?)

- Link Layer (point-to-point)
  - Point-to-point protocol (PPP) (*Detection only*)
  - High-level Data Link Control (HDLC) (*AQM protocol*)
- Transport Layer (end-to-end)
  - User Datagram Protocol (UDP) (*Detection only*)
  - Transmission Control Protocol (TCP) (*AQM*)
- Why you need both?

Positions in OSI Model



Bit Error and Packet Error

- Bit corruption: one or multiple bits in a frame are corrupted (0 flipped to 1, or 1 flipped to 0) during transmission, resulting in packet corruption.
- Independent bit errors and correlated bit errors
- Bit error rate (BER) for independent bit errors
- **Packet error vs bit error**: **BER vs. PER** (packet error rate, assuming a packet of N bits and no bit error can be corrected).

What is the probability that a packet is not corrupted? 1-PER

It is also the probability that no bit in a frame is corrupted $(1\text{-BER})^N$

Then $1\text{-PER} = (1\text{-BER})^N$

$$\text{PER} = 1-(1-\text{BER})^N$$
$$\sim\sim \text{N.BER if BER very small}$$

- Other packet errors: loss (due to congestion), duplication, or reordering

How Often do Errors Occur?

Channel BER = 10^{-7}, and a packet is 10^4 bit long, then the exact probability of a single error happens in the packet after its transmission is

$$\binom{10000}{1} \times (10^{-7}) \times (1 - 10^{-7})^{9999}$$

This is approximately $10^4 \times 10^{-7} = 10^{-3}$

Probability of exactly two errors:

For two bits *i*, *j* the probability of error is: $10^{-7} \times 10^{-7} \times (1-10^{-7})^{9998}$

Total # of 2-error patterns: $\binom{10^4}{2} = 10^4 \times (10^4 - 1)/2 \approx 5 \times 10^7$

Probability of two errors: $5 \times 10^7 \times 10^{-14} \times (1-10^{-7})^{9998} = 5 \times 10^{-7}$

PER when up to k bit errors are correctable?

Error Detection

- Consider a **word *w*** to be sent
- To detect corrupted bits on ***w*** at receiver, we need to add some **check bits**, forming a **codeword *cw***
- Example: Parity check scheme: let the word ***w***= ‘01100001’ to be sent, we could add a check bit equal to 0 if the total number of bit 1 in the word is even. Check bit is 1 otherwise.
- Such check bit is called a **parity bit**, codeword would be ***cw***= ‘011000011’

Why need check bits: Hamming distance!

Notation: (n, k)-code n-bit code, among which k bits are information.

Weight of a word: The number of ones in the word.

Ex.: W(100011101)= 5

Hamming distance between x, y: d(x,y) = W(x⊕y) (# of bits by which x and y differ from each other)

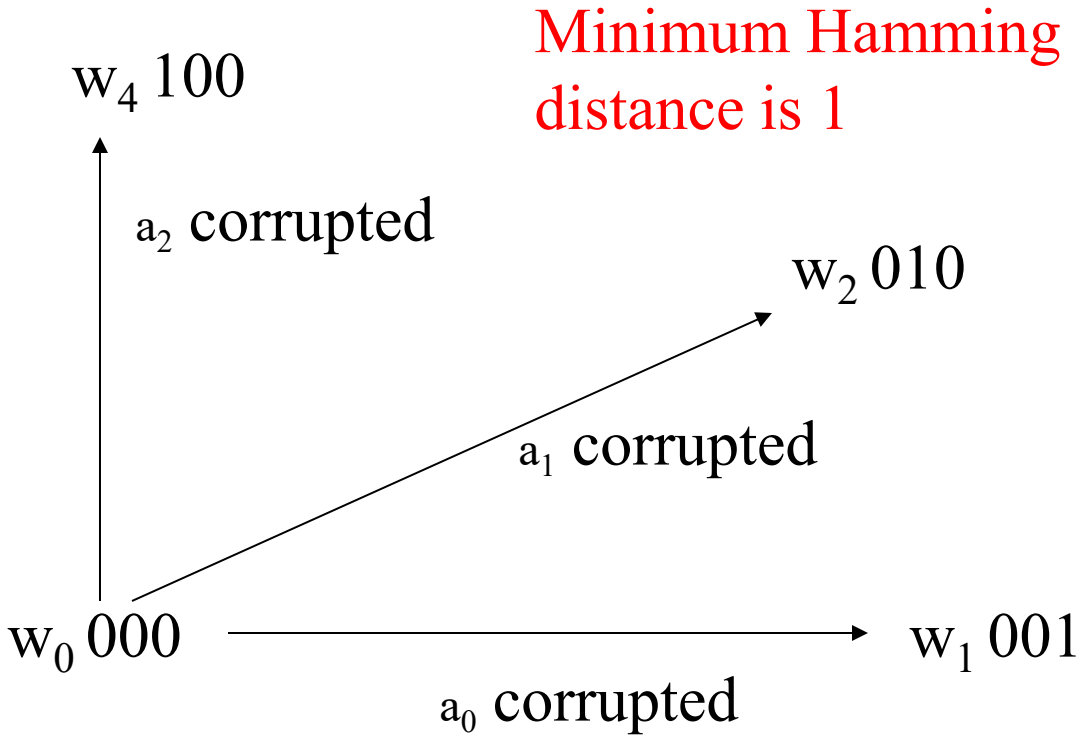
Ex. x = 10011101, y = 11100110, d = 6

Distance of codebook: the minimum Hamming distance out of all distances in a codebook

Why need check bits?

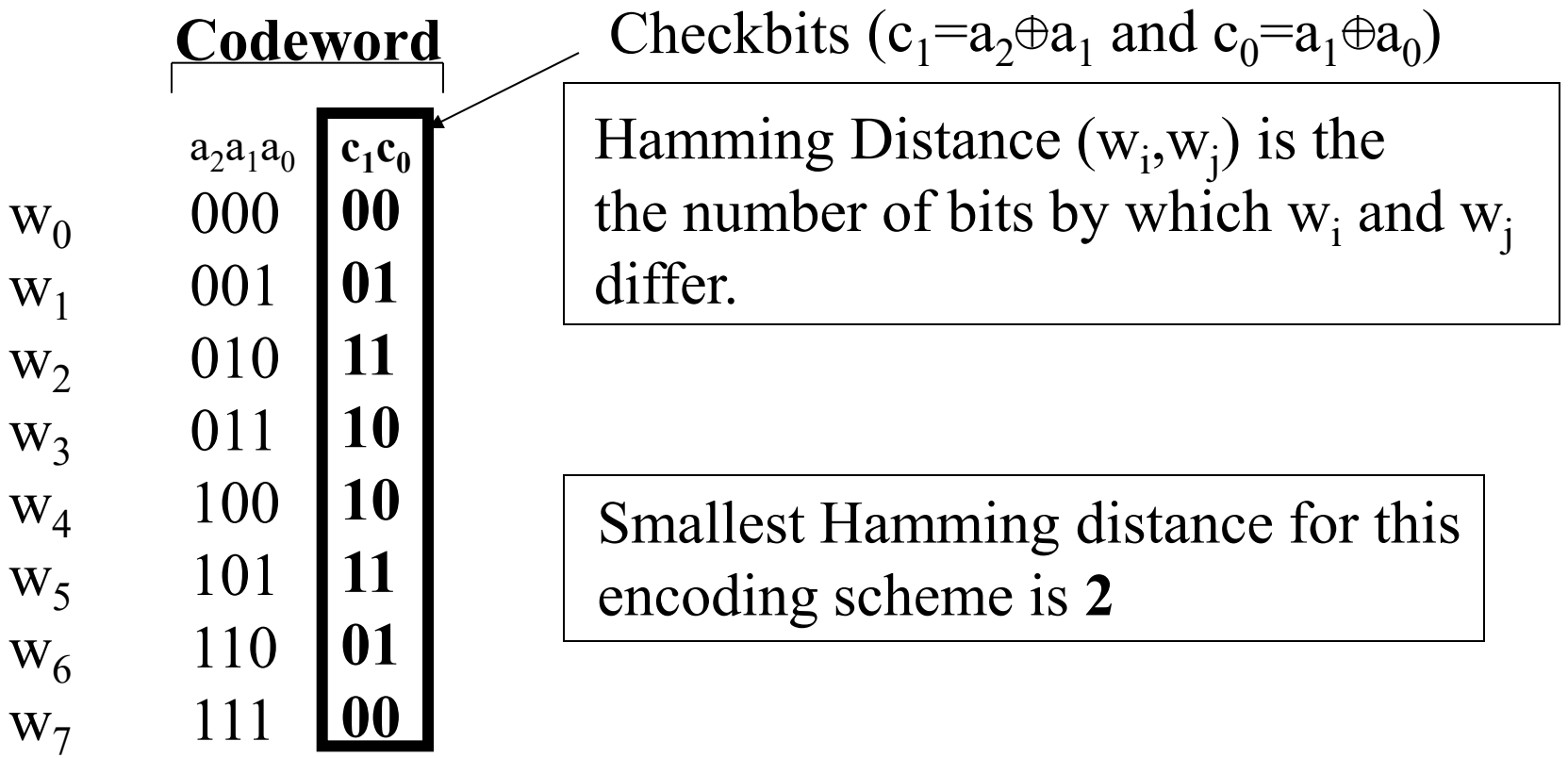
- Consider an alphabet of words $\{w_i\}$

	Words $a_2a_1a_0$
w_0	000
w_1	001
w_2	010
w_3	011
w_4	100
w_5	101
w_6	110
w_7	111

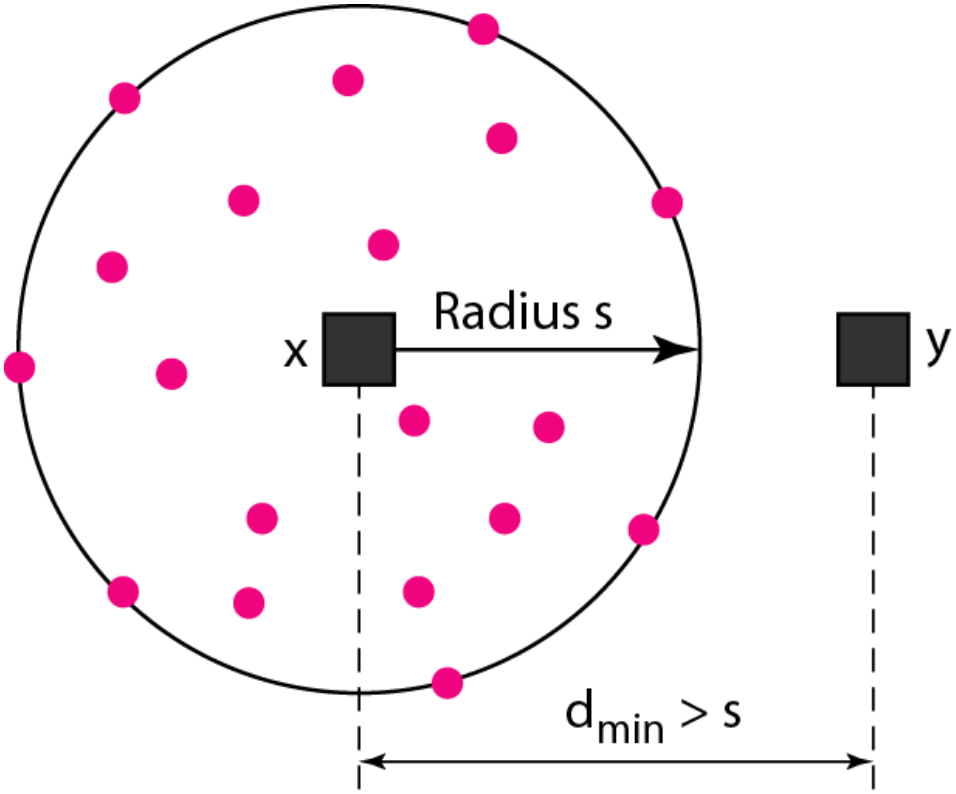


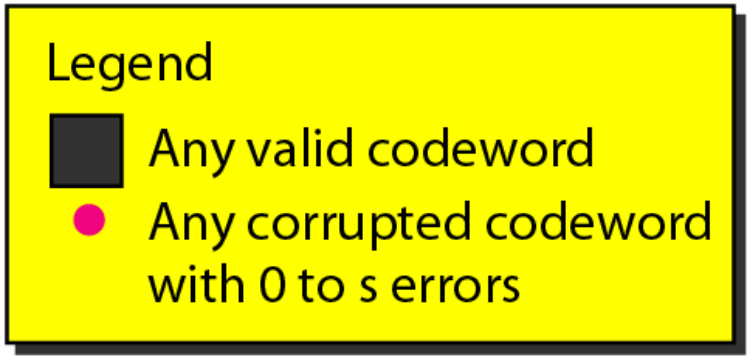
Error Detection (General)

- Consider an alphabet of words $\{w_i\}$



Given d_{min}, geometric concept for finding error detection capacity



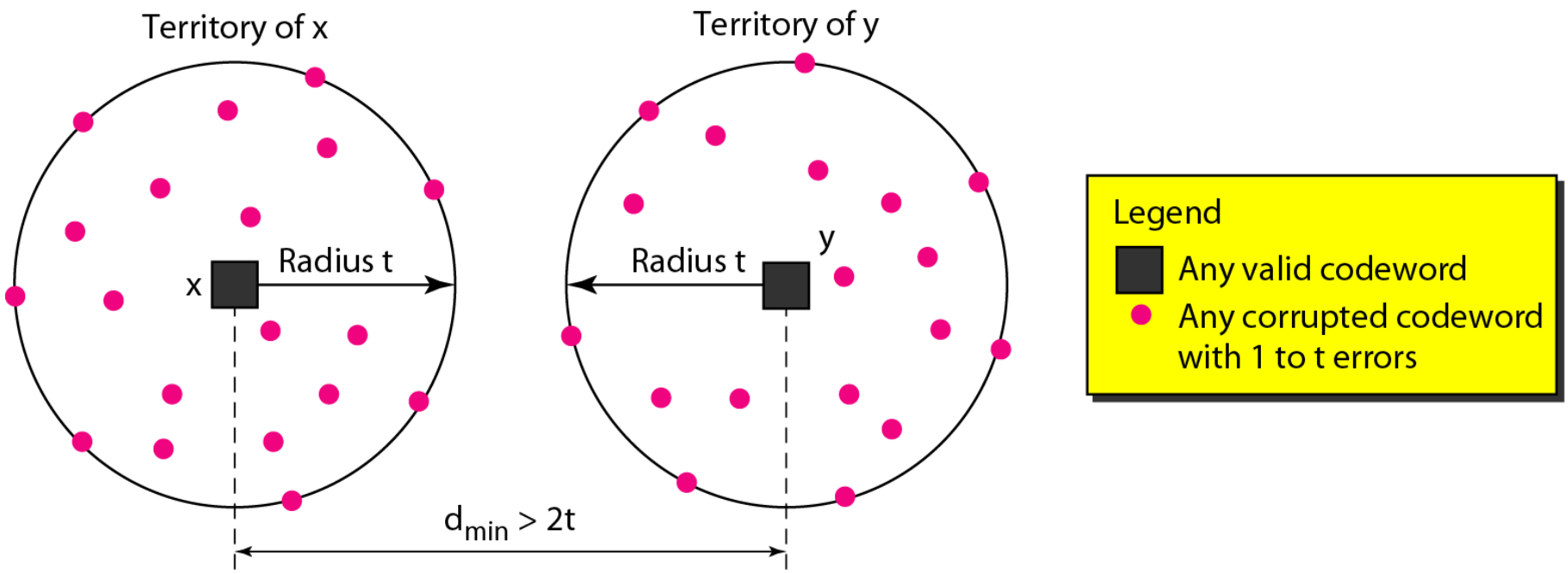


Requirements for Detection or Correction

- To **detect** up to ***m*** corrupted bits, the encoding scheme needs to have a minimal hamming distance of ***m+1***

- To **correct** up to ***m*** corrupted bits, the encoding scheme needs to have a minimal hamming distance of ***2m+1***

Given d_{min}, geometric concept for finding error correction capacity



Constraints on Generating Check bits

- We need to generate the **smallest number** of check bits and obtain the **highest** hamming distance between codewords.

- How to generate check bits under such constraints?

Examples

- Cyclic Redundancy Check (CRC)
- BCH (Bose-Chaudhury-Hocquenghem) codes
- Reed-Solomon code
- Turbo codes
- LDPC codes
- Polar codes
- Convolutional codes

Error Detection

- Consider a **word *w*** to be sent
- To detect corrupted bits on ***w*** at receiver, we need to add some **check bits**, forming a **codeword *cw***
- Example: Parity check scheme: let the word ***w***= ‘01100001’ to be sent, we could add a check bit equal to 0 if the total number of bit 1 in the word is even. Check bit is 1 otherwise.
- Such check bit is called a **parity bit**, codeword would be ***cw***= ‘011000011’

Why need check bits: Hamming distance!

Notation: (n, k)-code n-bit code, among which k bits are information.

Weight of a word: The number of ones in the word.

Ex.: W(100011101)= 5

Hamming distance between x, y: d(x,y) = W(x⊕y) (# of bits by which x and y differ from each other)

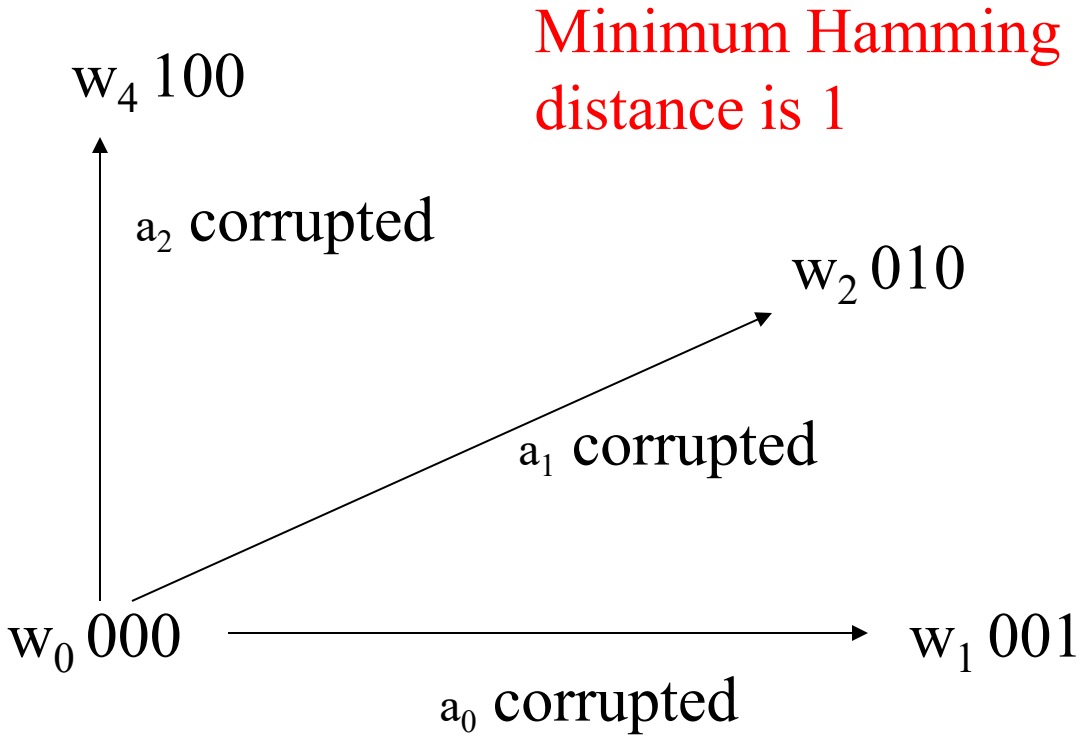
Ex. x = 10011101, y = 11100110, d = 6

Distance of codebook: the minimum Hamming distance out of all distances in a codebook

Why need check bits?

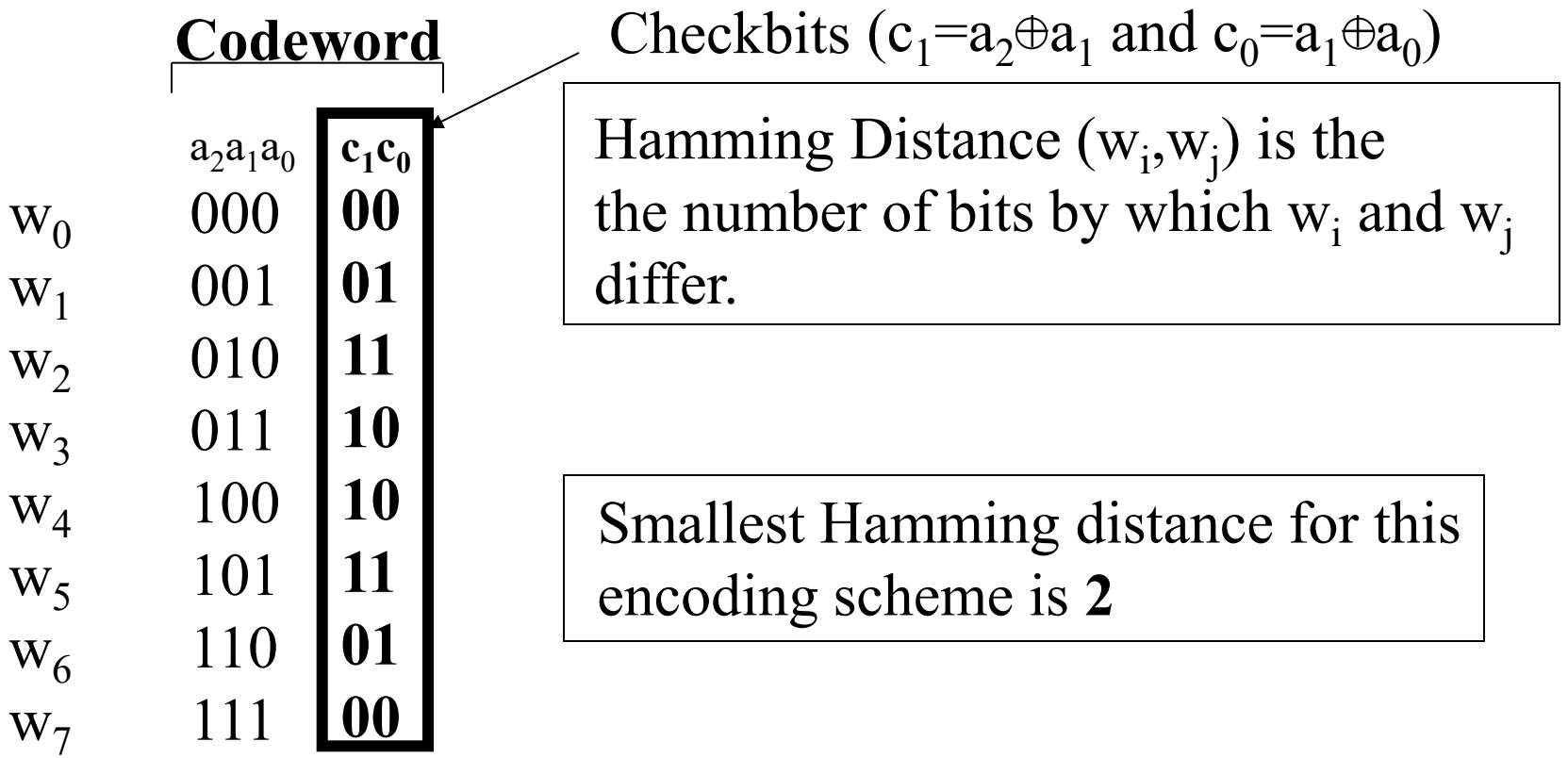
- Consider an alphabet of words $\{w_i\}$

	Words $a_2a_1a_0$
w_0	000
w_1	001
w_2	010
w_3	011
w_4	100
w_5	101
w_6	110
w_7	111

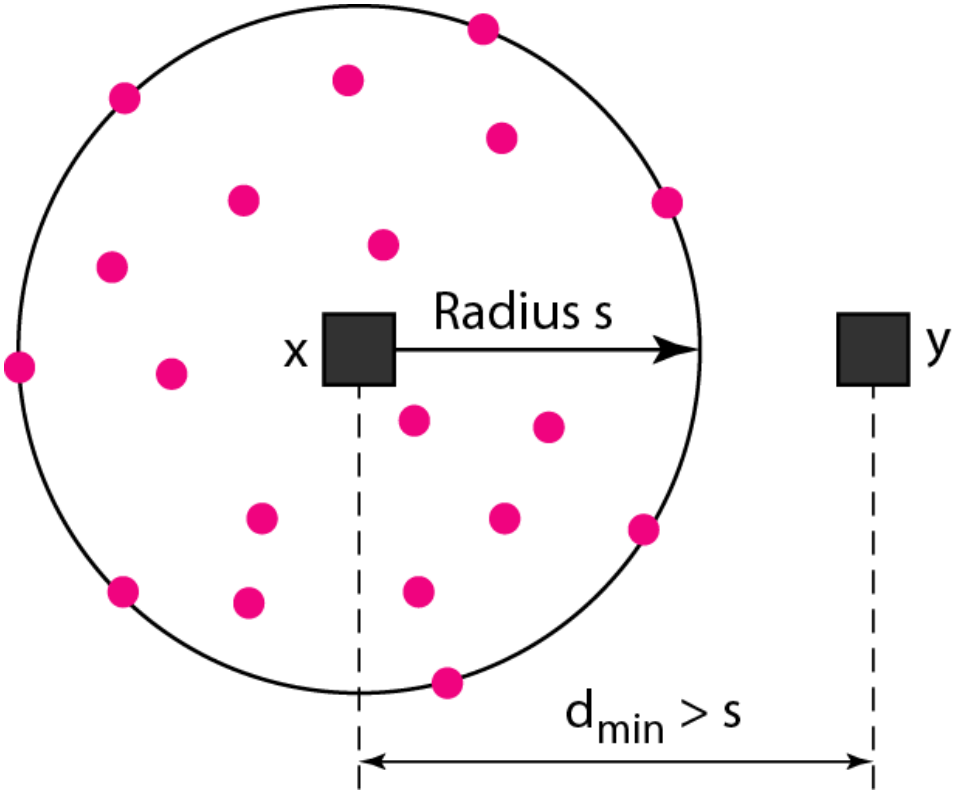


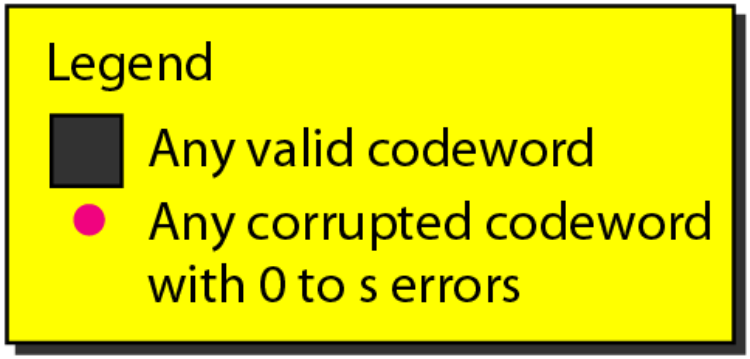
Error Detection (General)

- Consider an alphabet of words $\{w_i\}$



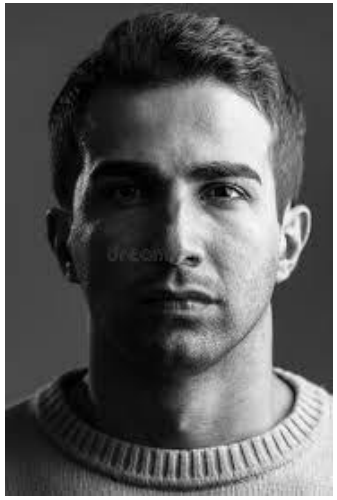
Given d_{min}, geometric concept for finding error detection capacity





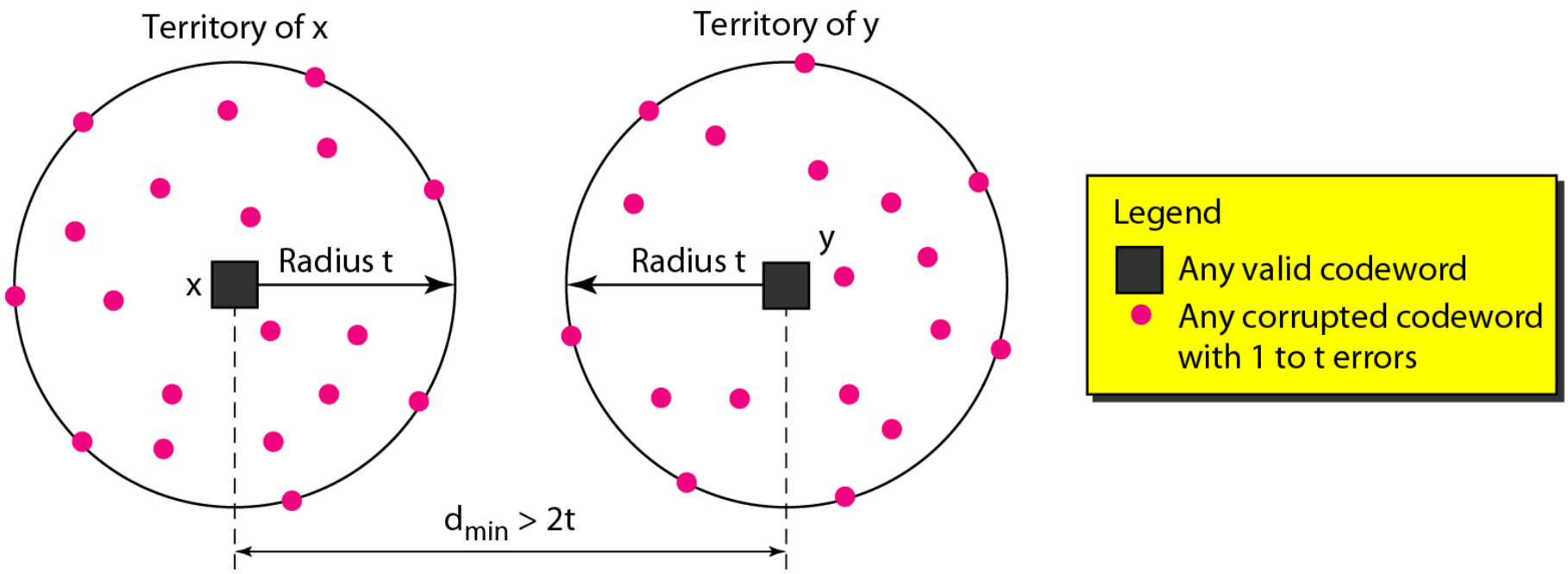
Error correction

Transmitted:



Received:

Given d_{min}, geometric concept for finding error correction capacity



Requirements for Detection or Correction

- To **detect** up to ***m*** corrupted bits, the encoding scheme needs to have a minimal hamming distance of ***m+1***

- To **correct** up to ***m*** corrupted bits, the encoding scheme needs to have a minimal hamming distance of ***2m+1***

Constraints on Generating Check bits

- We need to generate the **smallest number** of check bits and obtain the **highest** hamming distance between codewords.

- How to generate check bits under such constraints?

Examples

- Cyclic Redundancy Check (CRC)
- BCH (Bose-Chaudhury-Hocquenghem) codes
- Reed-Solomon code
- Turbo codes
- LDPC codes
- Polar codes
- Convolutional codes

Main Strategies for Error Correction

- By coding: Error Correction (forward error correction code or FEC): *data is encoded such that errors are detected and corrected with no retransmission.*

- By protocol: Error Detection and retransmission (automatic repeat request or ARQ): *data is encoded such that errors can be detected. When an error occurs, sender must retransmit the packet*

Error Correction or Error Detection/Re-Tx?

- Consider a **frame** of ***n*** bits of information data to be sent
  - To be able to detect up to **k** corrupted bits, we need to add ***m*** bits.
  - To be able to detect and correct up to **k** corrupted bits, we need ***m'*** bits.

$$m' > m$$

Error Correction or Error Detection? (Cont'd)

- If we use error detection/Re-Tx, each frame has size ***n*+*m***

- If we use error correction, each frame has size **n+m'**

$m' > m$

Error Correction or Error Detection? (Cont'd)

- If we use **error correction code**, a packet is always successfully received. So, the cost of sending each packet is equal to the time it takes to send (n+m') bits. Denote this time as T(n+m').
- If we use **error detection + re-tx**, the packet must be resent whenever the packet gets corrupted. Suppose that the ***cost*** of time to realize that the packet is lost is equal to ***A (protocol related!)***. How much time does it take to send successfully a packet when using error detection + re-tx?

Error Correction or Error Detection? (Cont'd)

- We are trying to answer the question: how much time does it take to send successfully a packet when using error detection?
- We can in fact derive the expected time it would take to send successfully a packet when using error detection.

The time is :

1) ***T****(n+m) if successful the first time with probability* ***(1-p)****, where p is PER*

2) *2T(n+m)+****A*** *if successful the second time with probability* ***p.(1-p)***

3) *3T(n+m)+2.****A*** *if successful the third time with probability* $\boldsymbol{p^2.(1-p)}$

4) **……..**

5) ***i****T(n+m)+ (****i****-1).****A*** *if successful the* i^{th} *time with probability* $\boldsymbol{p^{i-1}.(1-p)}$

- The expected time is $\boldsymbol{(T(n+m)+ p.A)/(1-p)}$

Error Correction or Error Detection? (Cont'd)

- Recall that expected time is $(T(n+m)+ p.A)/(1-p)$ with error detection and the time is $T(n+m')$ with error correction.
- If p is small, the expected cost becomes $T(n+m) + p.A$. The answer relies on whether $p.A$ compensates the extra bits due to m'
- If p is not small, error correction becomes more interesting
- Sometimes, we just cannot afford to retransmit (for example realtime audio), then error correction is necessary!
- In practice, usually both are used at the same time
  - For small # of bit errors, use code for correction (FEC: forward error correction)
  - For large # of errors, use retransmission

Retransmission (ARQ) Protocols

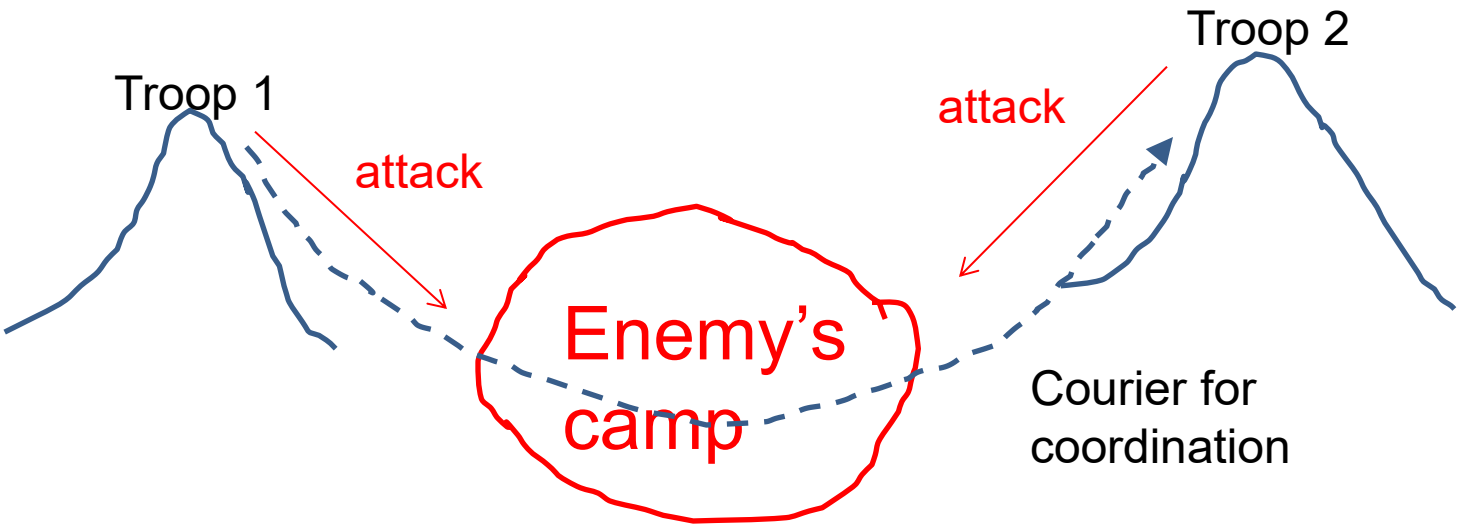
- Stop and wait protocol (SWP)
- Alternating Bit Protocol (ABP)
- Go back n (GBN)
- Selective repeat protocol (SRP)

What do we care about?

- **Correctness**: A retransmission protocol is correct if it delivers exactly one correct copy of each packet

- **Efficiency**: normalized throughput (throughput/bandwidth)

Motivating Example for Lossy Channel



- Courier through enemy's camp

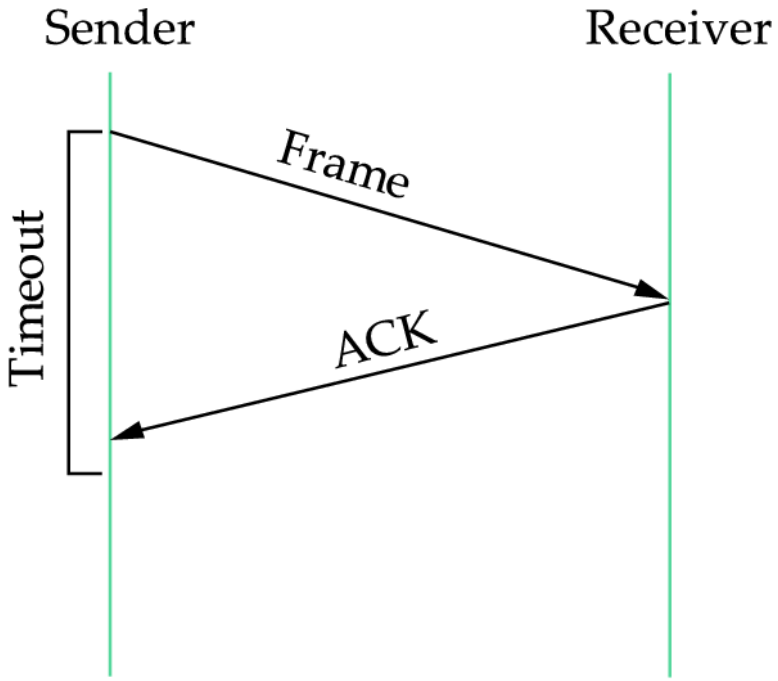
  ACK, timeout, and retransmit

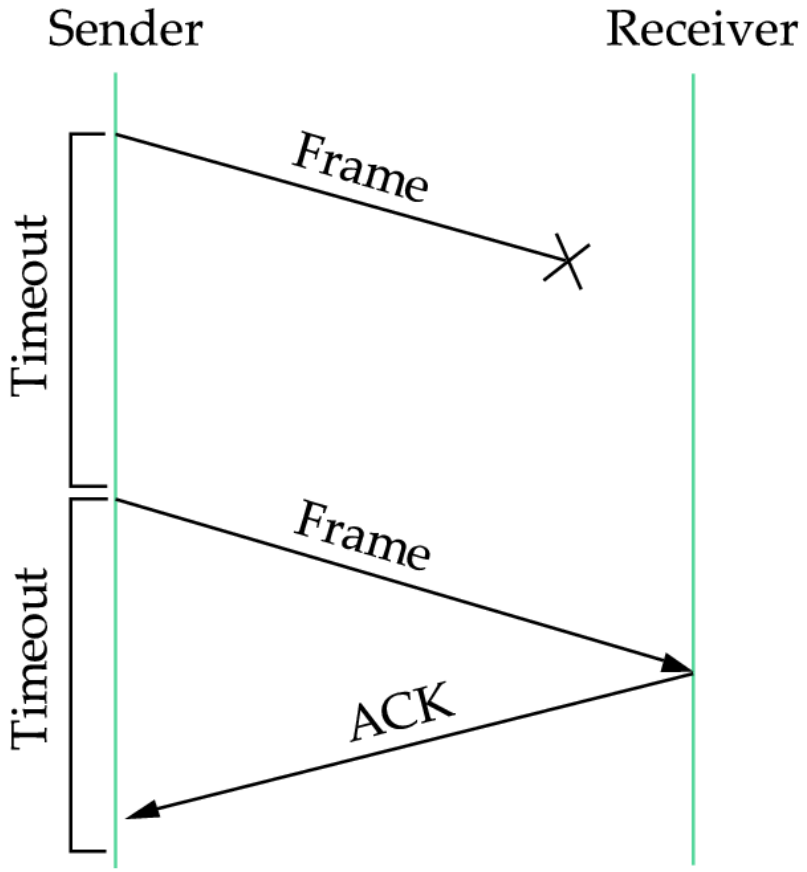
SWP

- **Operation of SWP**:
  - Whenever a receiver gets a correct packet, it sends back an acknowledgment (without sequence #).
  - The sender stops and waits after sending out a packet. The sender **resends** a packet previously sent if it did not get an acknowledgement from the receiver within a **known bound T_o** (Timeout value). The sender sends the next packet only after receiving an ACK before timeout.

SWP Cases

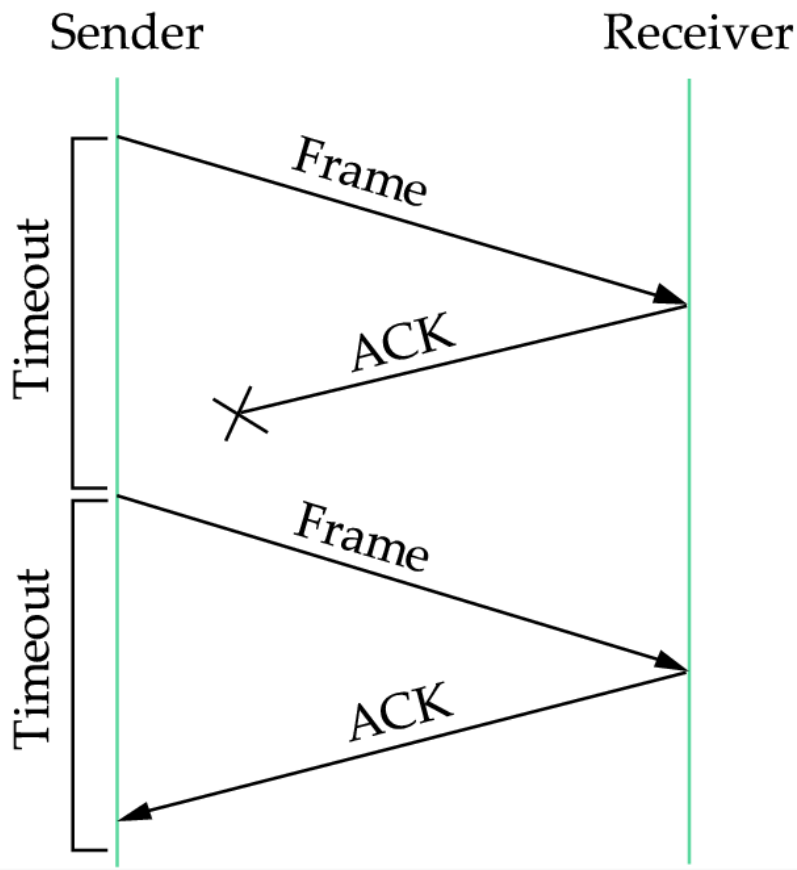
Stop-and-wait





Other Case

ACK is lost



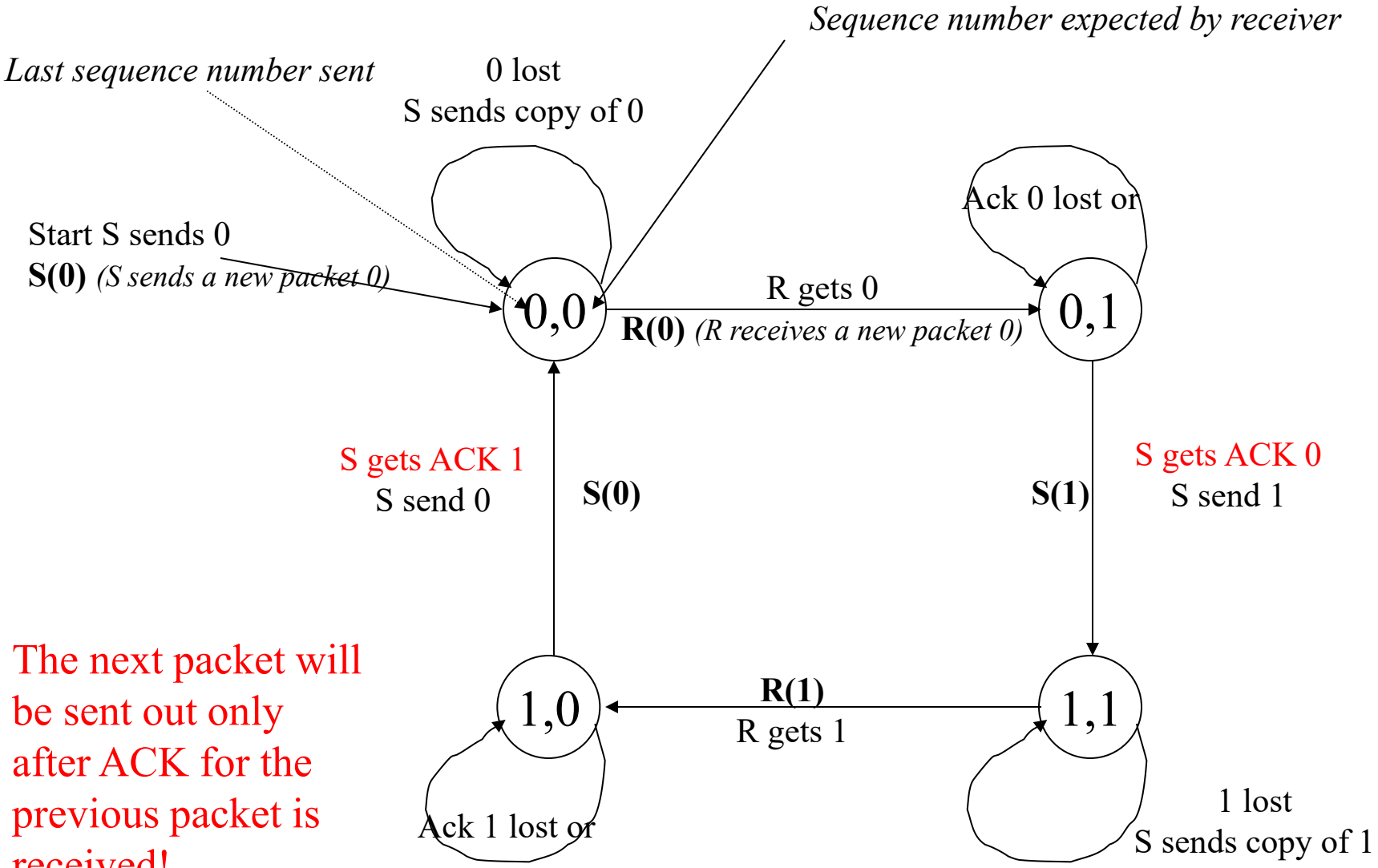
Correctness: Finite State Machine

- List all possible states of the system
- List all events leading to transitions from state to state
- Check the sequences of transitions to verify every packet is successfully delivered

SWP Correctness

- We have a sender S and a receiver R
- What are the possible states for such protocol?
- A state can be summarized by a couple (x,y):
  - X is the sequence number of the last packet sent
  - Y is the sequence number that the receiver is waiting for (0 initially).

SWP Correctness (2)



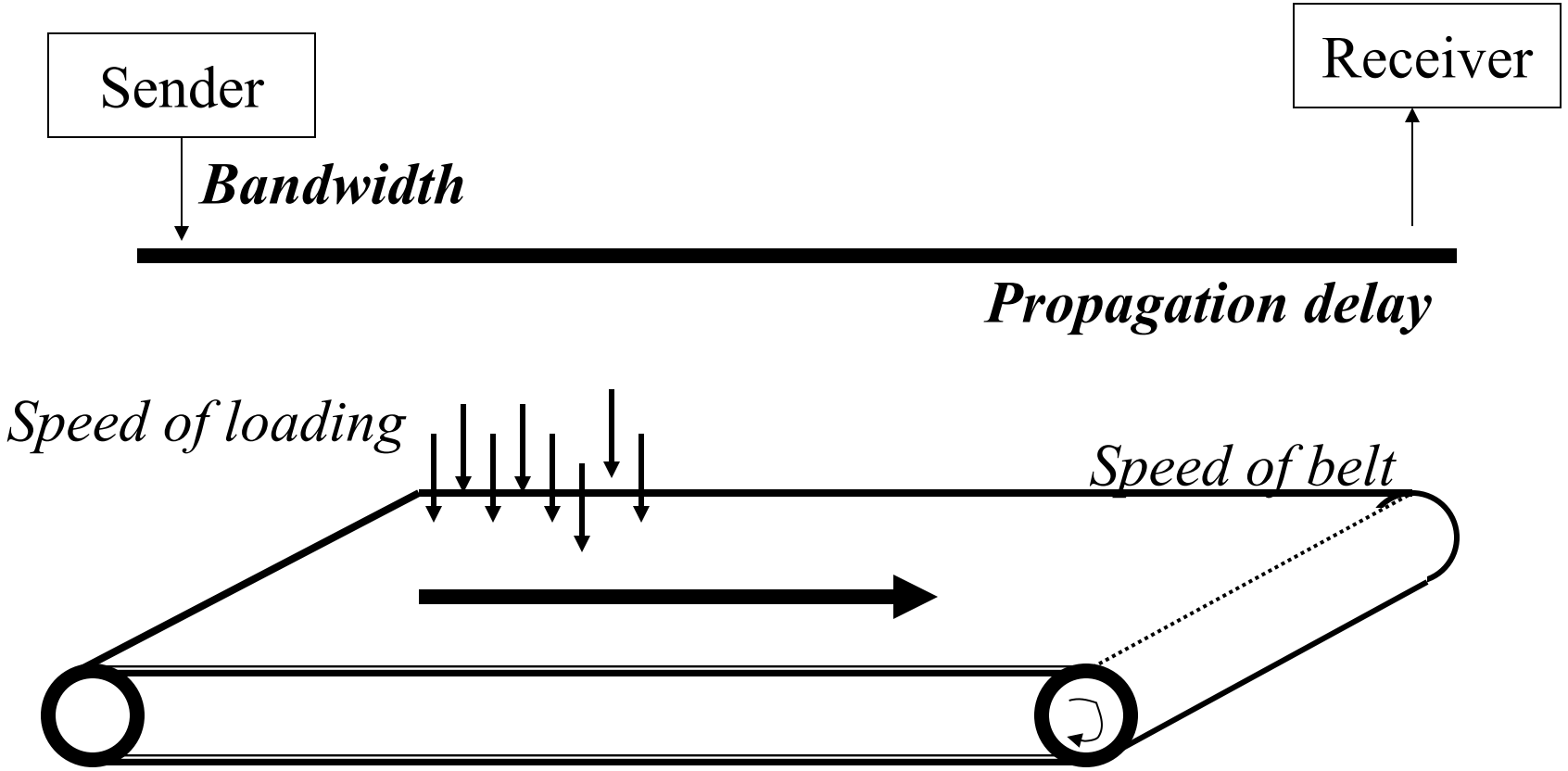
The next packet will be sent out only after ACK for the previous packet is received!

SWP Efficiency

- Case 1: Analysis without loss error. No packet is lost

- Case 2: We assume a packet error rate ***PER***

Simplified Model

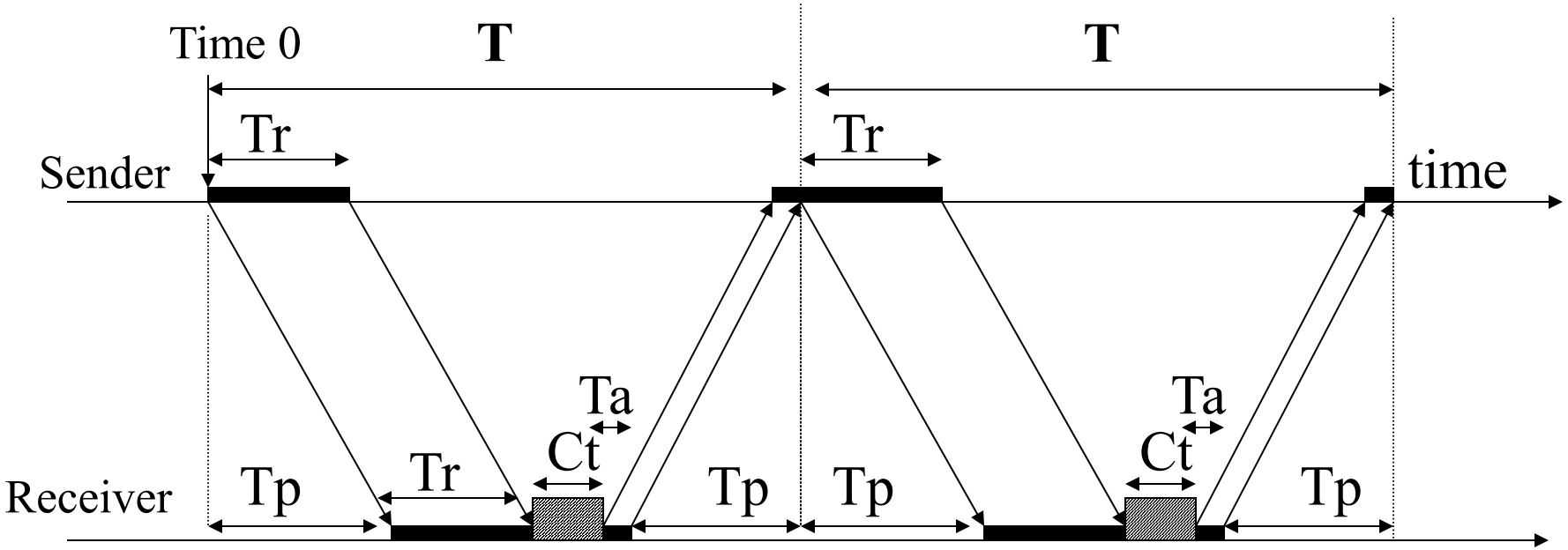


Performance metrics

- ***Bandwidth***: the number of bits sent to the medium per unit time. This is a measure of the time it takes to put a given number of bits on the wire.
- **Throughput**: rate of delivery (how many packets (bits) can you deliver per unit of time). The throughput of SWP is the inverse of the avg time it takes for the receiver to deliver a packet.
- ***Transmission delay***: packet size/bandwidth
- ***Propagation delay***: time it takes the first bit to travel from the source to the destination. *This parameter is related to and limited by the speed of light*.

Analysis

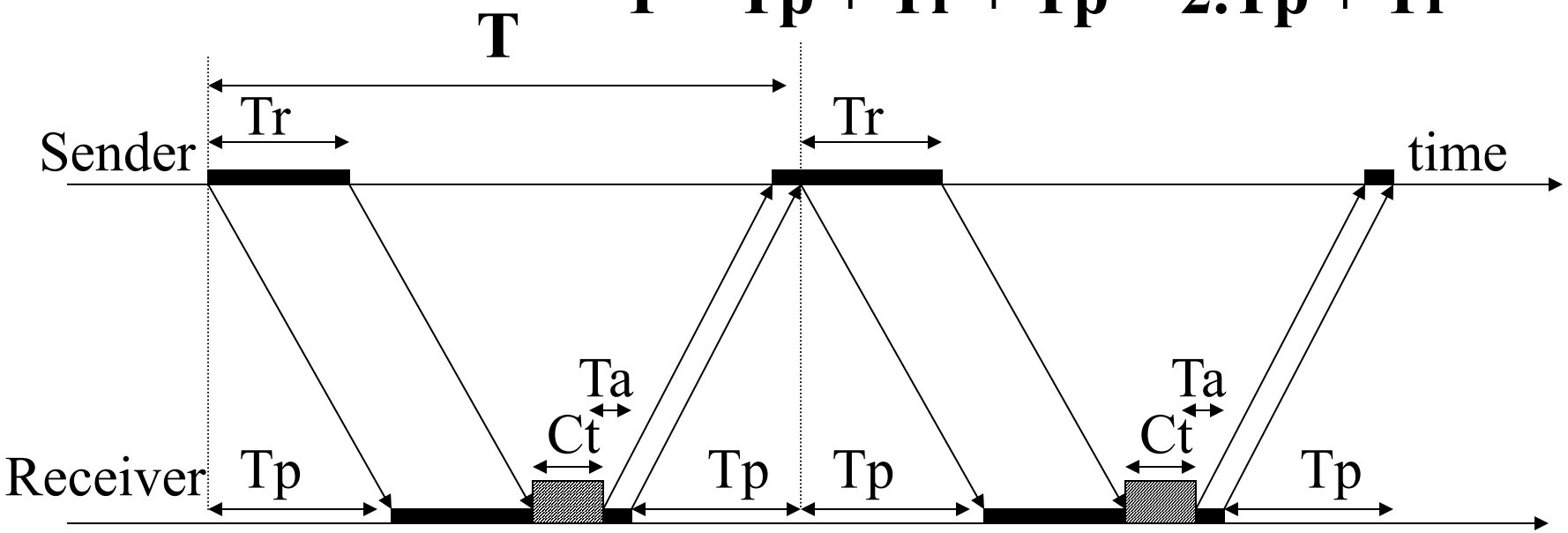
- What are the values we are interested in?
  - 1) Throughput = Amount of data received per unit time
  - How to compute that? What conditions are given?
  - ***Bw***: bandwidth in bits/s (given), ***Tp***=Propagation delay in s (given),
  - S= Size of frame in bits (given) Ct= Computing time negligible
  - Tr = transmission time of a frame = **S/Bw**
  - Ta = transmission of an ack = Sa/Bw



Analysis (cont'd)

- Between the time the sender starts sending a segment and the time the ack is received, a time **T** elapses.
- This time **T= Tp + Tr + Ct + 1/B + Tp**
- If we neglect the processing time Ct and 1/B, the time T becomes

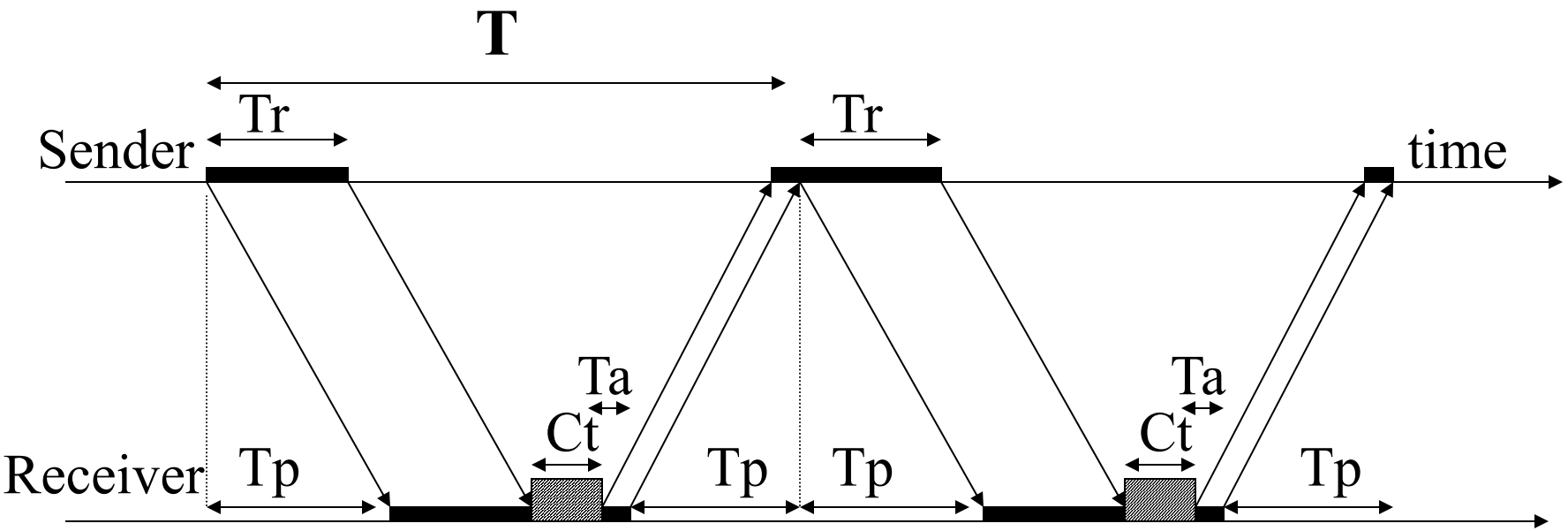
T = Tp + Tr + Tp = 2.Tp + Tr



Analysis (cont'd)

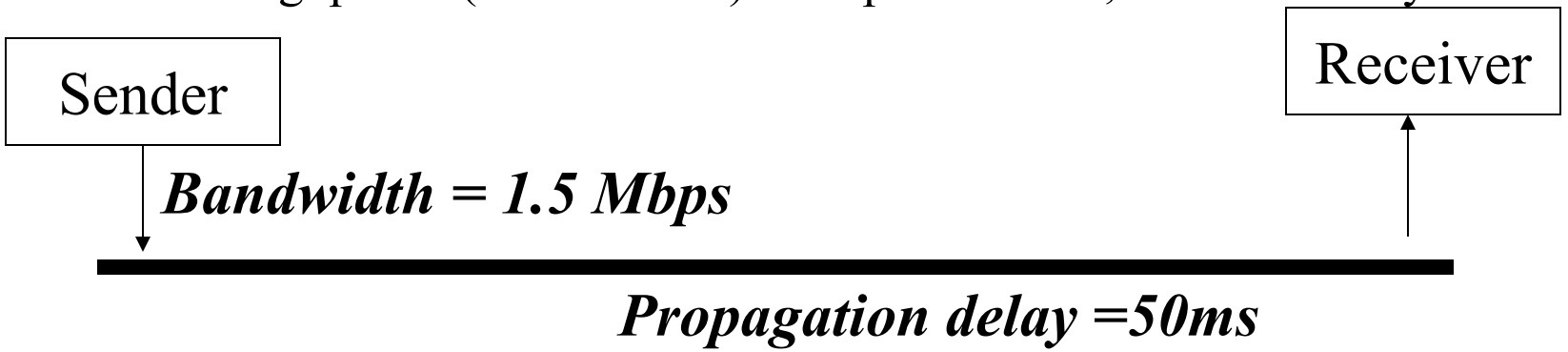
- **Conclusion**: every cycle T, we receive one frame. Therefore, the throughput = **1/(2.Tp+Tr)** **(in frames per sec)**

Does a large bandwidth (B) always lead to high throughput? Not necessarily!



Is Stop-And-Wait Efficient? Example

- **Frame size = *S* =** 100 bits, ***Bw*** = 1.5 Mbits/s, *Tp*=50 ms
- **Tr** = Frame size / Bandwidth = ***S*/*Bw*** = 100 / 1.5 Mbits/s= **66.67 μs**
- **Throughput**=***1/T*** = ***1/(2.Tp+Tr)*** =1/(2*50ms + 66.67 μs) which is about 1 frame per 100ms = 10 pkts/second
- Line rate (bandwidth in frames per seconds) = 1.5Mbps/100=15000pkts/second
- Efficiency is the ratio of (Throughput)/(Line rate)
- In this case, the efficiency is 10 / 15000 = 0.066%
- Now, let's increase the bandwidth to infinity, so **Tr** = 0
- New throughput: 1/(2*50 ms + 0) = 10 pkts/second, new efficiency = 0%



Observation?

- When bandwidth is 1.5 Mbps, the throughput is 10 packets/second, efficiency is **0.067% (extremely low)**.
- Increasing the bandwidth by infinite times, throughput remains the same, efficiency reduces to 0.
- Why? What's the problem?

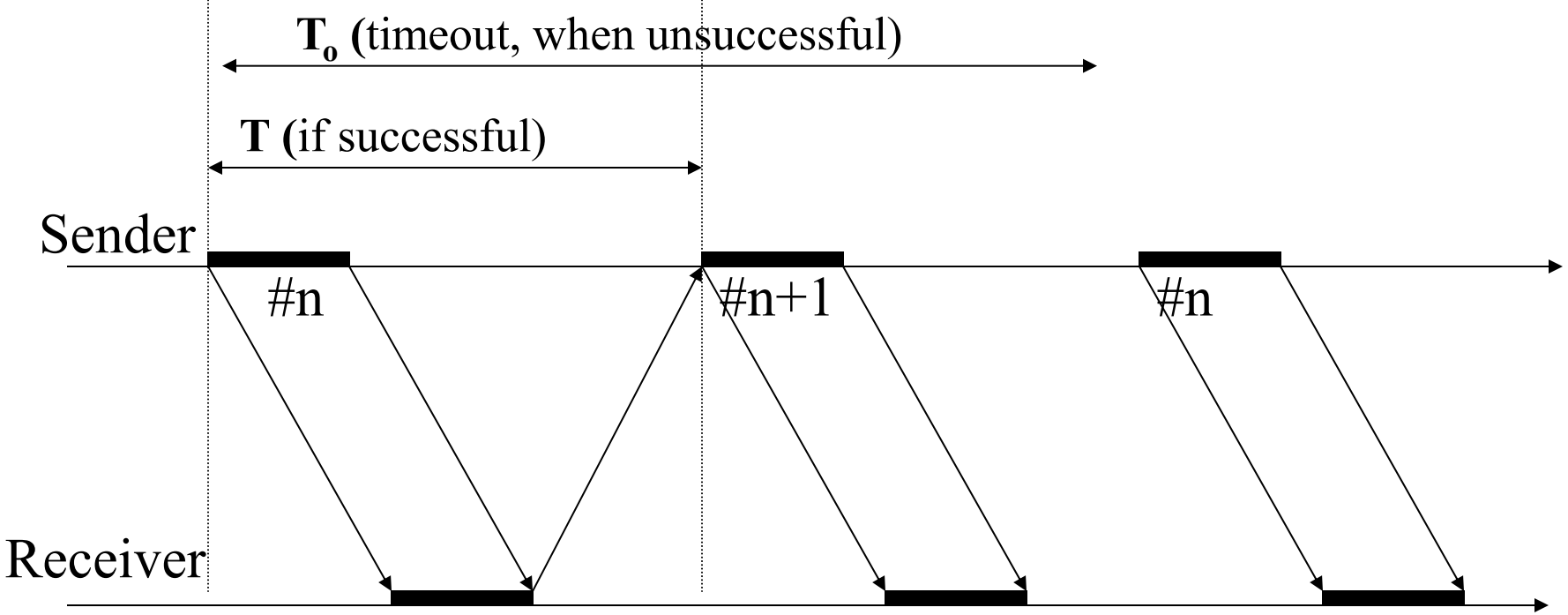
Keys Facts about SWP

$$\textbf{Throughput (in packets/second)} = \frac{1}{2.Tp + Tr}$$

- Propagation delay is dominant in this case!
- Even, if bandwidth is infinite, i.e., Tr = 0
- Throughput = 1/(2Tp). It's not infinite! It could actually be very small!

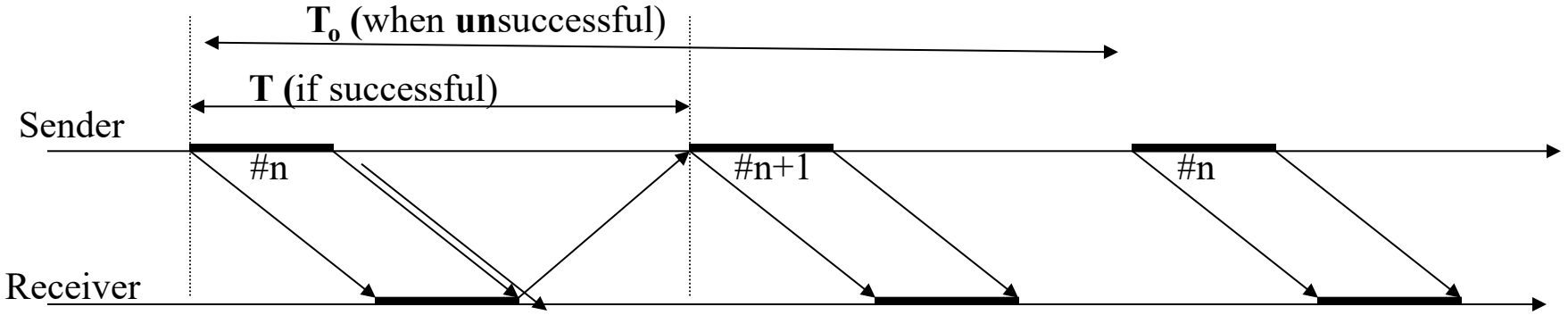
Case 2: Efficiency in Presence of Errors

- Let ***X*** = random variable representing the duration of a cycle (time between two different frames)
- We have to compute **E(X):** the expected value of ***X***



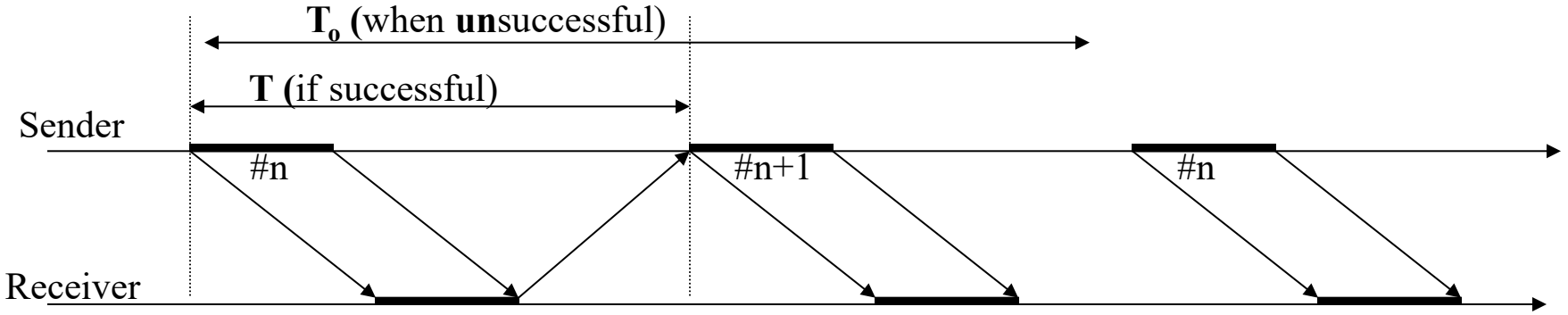
Case 2: Efficiency in Presence of Errors (2)

- We can say:
  - 1) When successful, $X = T$
  - 2) When unsucessful, $X = To + Y$, where Y is a random variable representing the time it would take for the re-tx before we send the next frame
- Based on above, we can say:
  - $X = T$ with probability $(1\text{-}PER)$ **(PER= Packet Error Rate)**
  - $X = To + Y$ with probability PER
- What is E(X), the expected value of X?



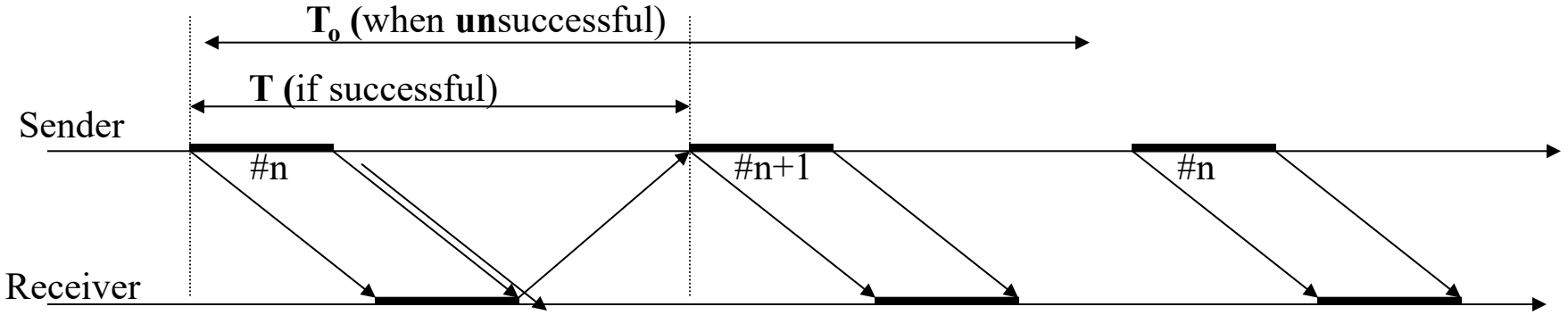
Case 2: Efficiency in Presence of Errors (3)

- Recall that:
  - $X = T$ with probability $(1\text{-}PER)$ **(PER= Packet Error Rate)**
  - $X = To + Y$ with probability PER
- By definition of E(X):
  - $E(X) = (1\text{-}PER).T + PER.(To + E(Y))$
- But, what is E(Y)?



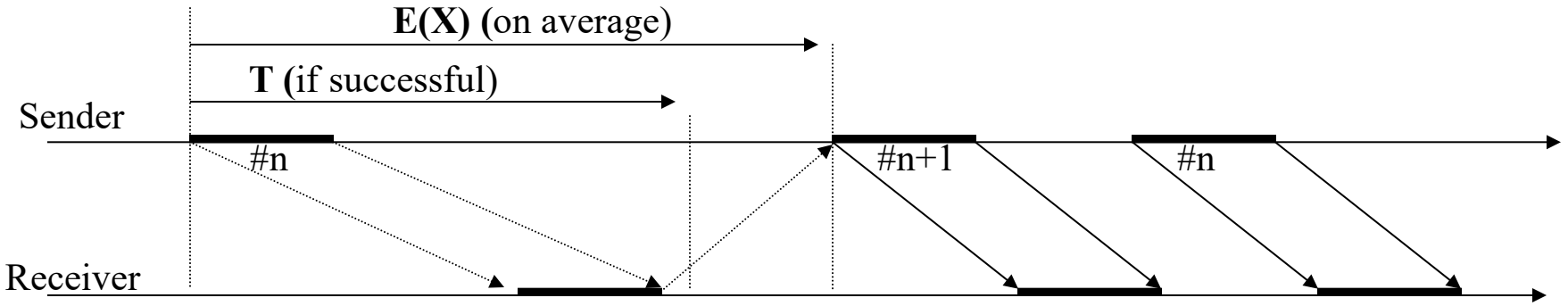
Case 2: Efficiency in Presence of Errors (4)

- Recall that by definition of E(X):
  - ***E(X) = (1-PER).T + PER.(To + E(Y))***
- But, what is E(Y)? Recall that Y is the time until we send the next frame. Then E(Y) is also the expected length of the cycle.
- Then, **E(Y) = E(X)**
- Then, ***E(X) = (1-PER).T + PER.(To + E(X))***
- We just have to solve for E(X)



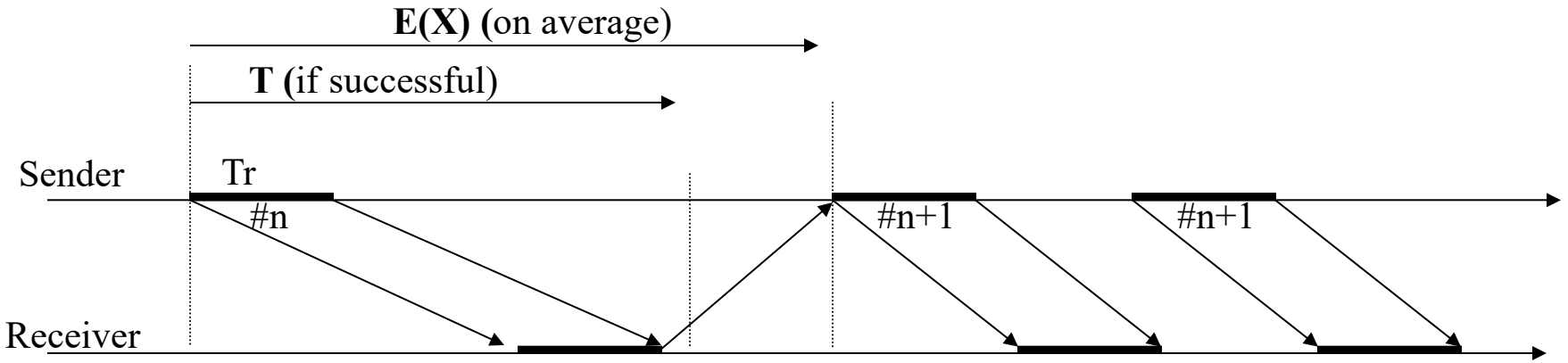
Case 2: Efficiency in Presence of Errors (5)

- Recall, ***E(X) = (1-PER).T + PER.(To + E(X))***
- Solving for E(X):
- ***E(X) = T + [To . PER/(1-PER)]***
- In words, the average cycle is ***E(X***).
- In other words, we can expect to successfully send/deliver one frame per ***E(X)***



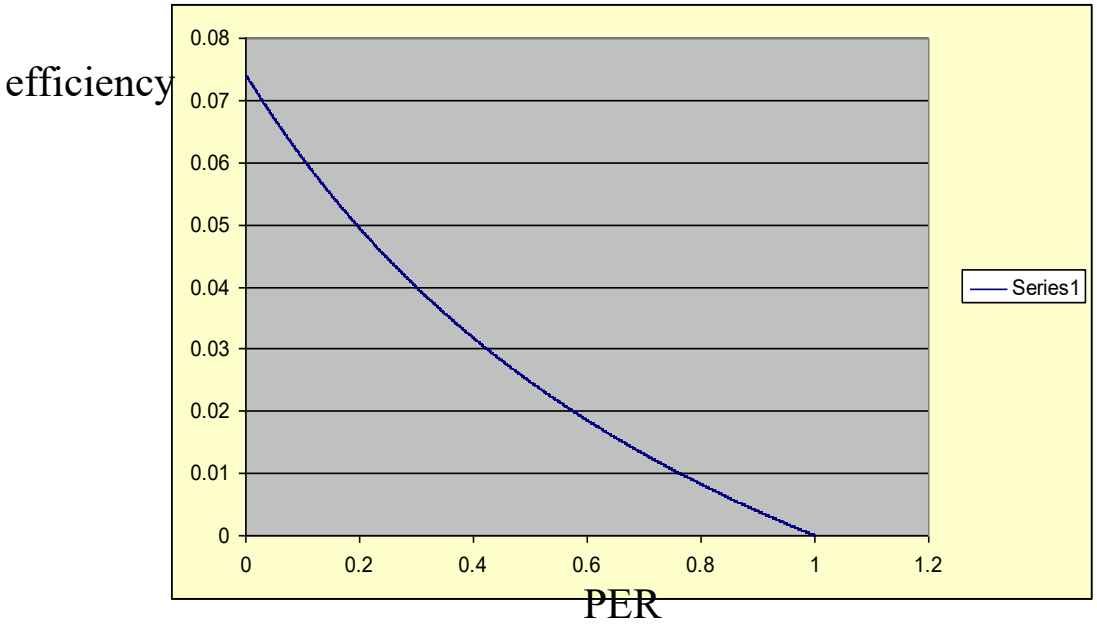
Case 2: Efficiency in Presence of Errors (6)

- Throughput is then equal to ***1/E(X)*** . (in frames per second)
- The efficiency is by definition Throughput/ Line Rate.
- The efficiency is then ***(1/E(X)) / (1/Tr)***
- The efficiency is then ***Tr/E(X) = Tr/(T + [To . PER/(1-PER)])***
- If we replace T with its value (slide # 29), I.e., T= 2. Tp + Tr
- The efficiency is ***Tr/E(X) = Tr/(2.Tp + Tr + [To . PER/(1-PER)])***



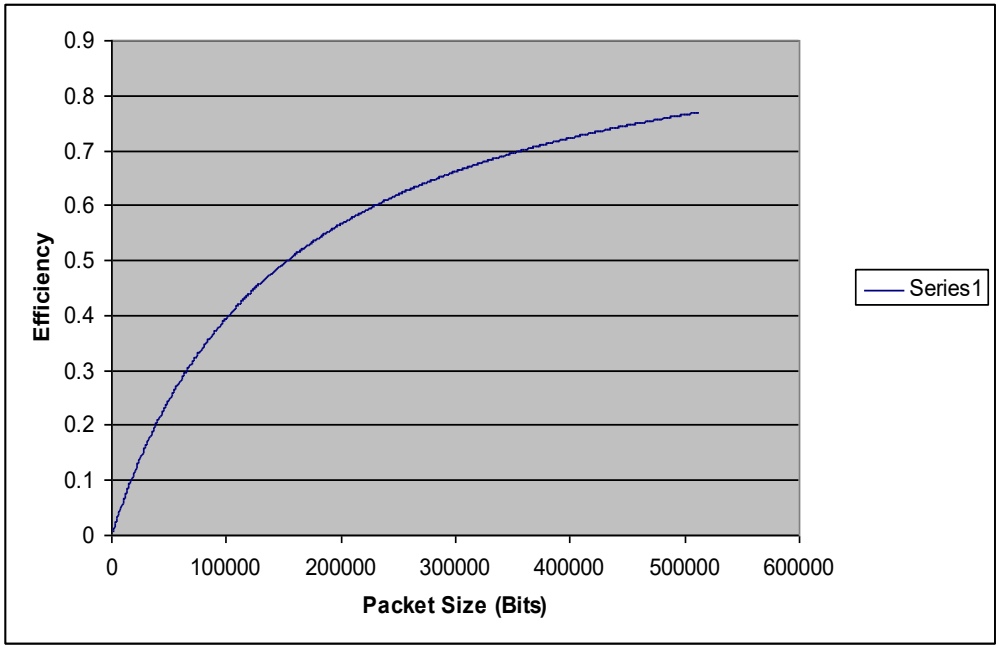
Impact of PER on SWP

- **Frame size = *S* =** 1500 bytes, ***Bw*** = 1.5 Mbits/s, ***Tp*** = 50ms
- ***Tr*** = S/Bw = 8 ms
- Let ***To = 2.T = 2. (2.Tp+Tr)*** = 216 ms
- Recall that the efficiency is ***Tr/E(X) = Tr/(2.Tp + Tr + [To . PER/(1-PER)])***



Impact of Frame Size on SWP

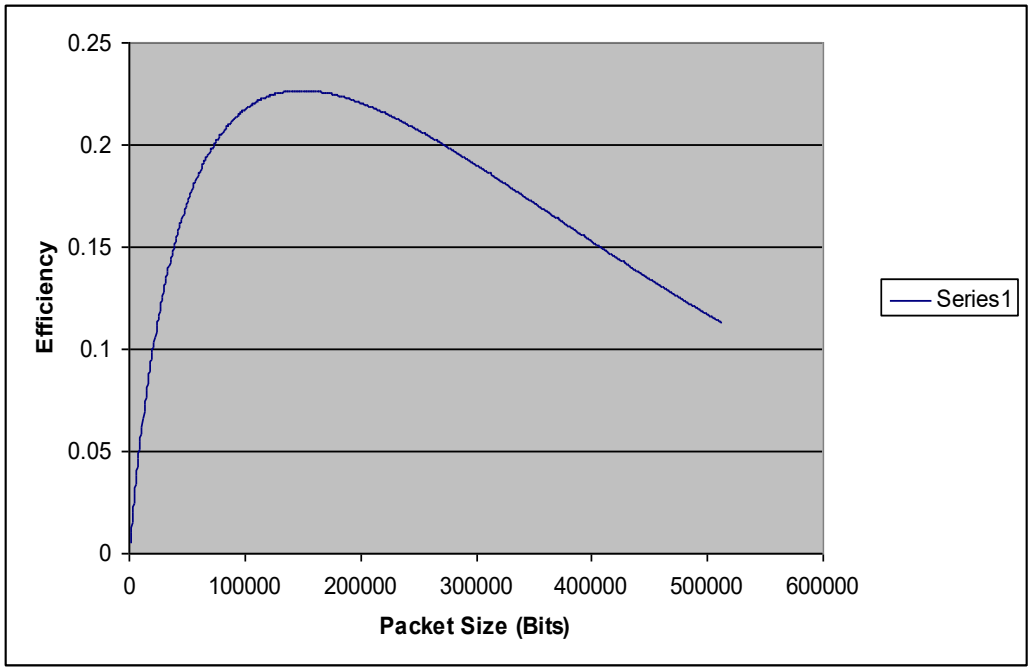
- **Frame size = *S* =** variable, ***Bw*** = 1.5 Mbits/s, ***Tp*** = 50ms, ***PER*** = 1% = 0.01
- ***Tr*** = S/Bw
- Let ***To*** = 216 ms
- Recall that the efficiency is ***Tr/E(X) = Tr/(2.Tp + Tr + [To . PER/(1-PER)])***



True Picture Impact of Frame Size on SWP

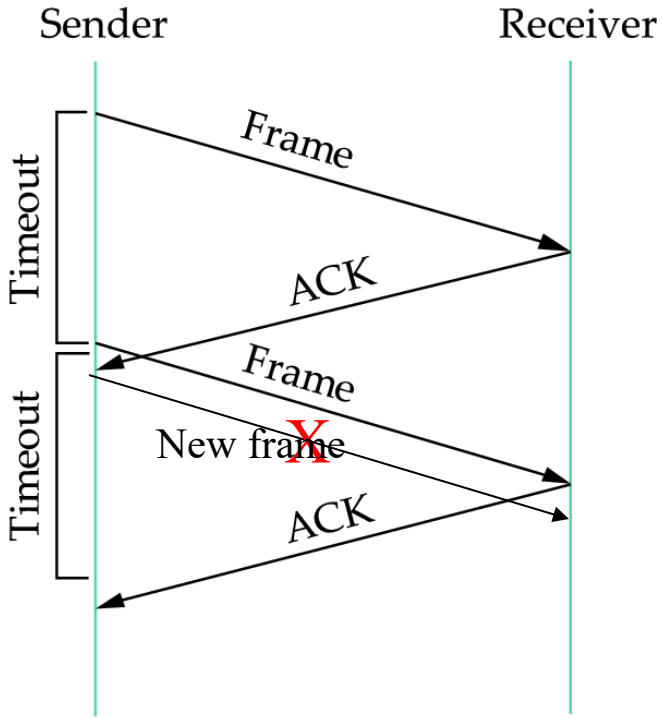
- **PER depends on packet size for a given BER**, as $PER = 1-(1-BER)^S$

Under a given BER



Alternating Bit Protocol (ABP)

- We do not know the right **bound T_o** (Timeout value is hard to decide).
- If the bound is unknown, we may confuse acknowledgements (is ack for new packet, or for some OLD packet thought lost).



- What to do?
- Put sequence numbers on acks

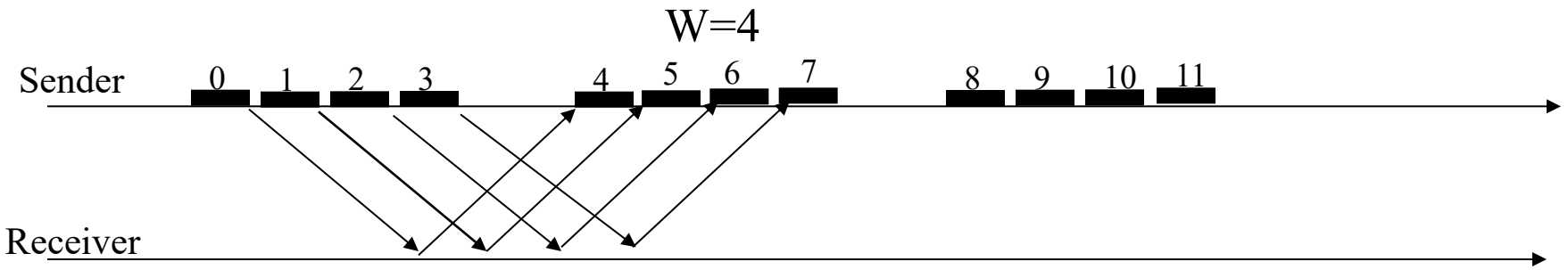
Alternating Bit Protocol (ABP)

- **Correctness**: ABP can be shown to be correct if acks are numbered (0,1).

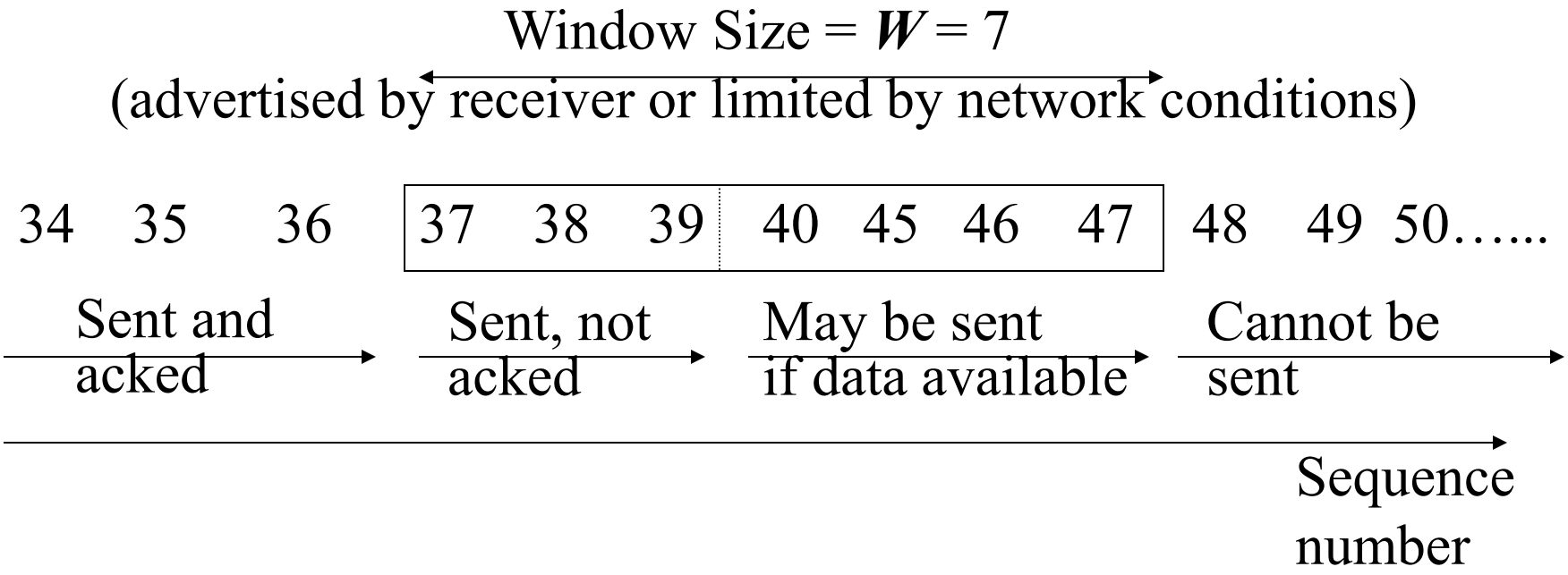
- **Efficiency**: same efficiency as SWP

Sliding Window Protocol

- Keep sending packets without waiting for acknowledgements.
- Outstanding packets: packets sent but hasn't been acked
- For SWP, # of outstanding packet is 1.
- How many packets should be outstanding without ack?
- The number of oustanding frames is the window size ***W***
- When there is no error, efficiency is improved by W times min(1, WTr/T)

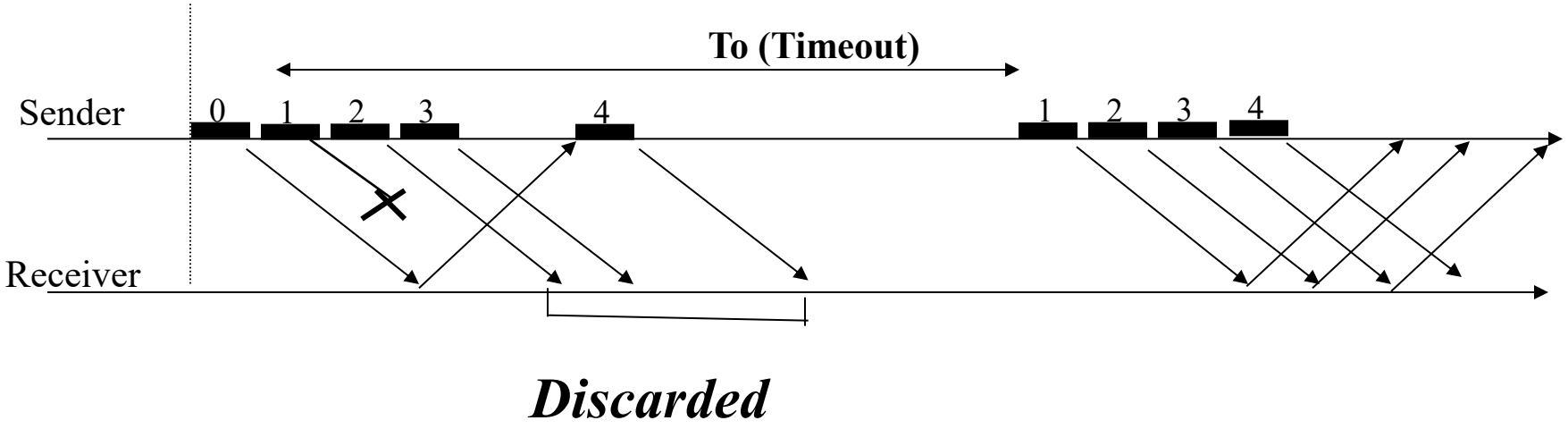


Window Principle (Maintained by the Sender)



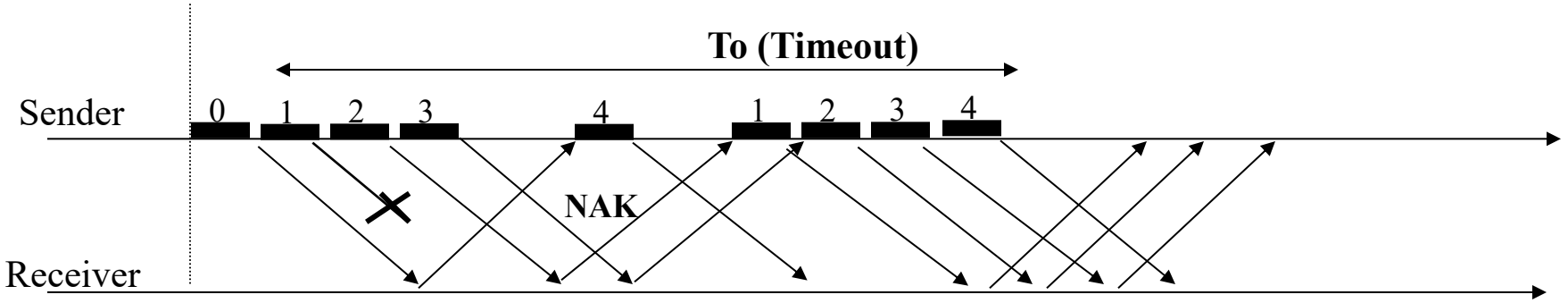
Go Back N

- If packet # ***i*** is not acked, the sender will resend all packets starting packet # ***i***.
- Receiver doesn't buffer (receive window size = 1), and ACK is in order (receiver only accepts packets in order)
- Example: ***W***=4 (this is also called N in the context of go-back-N protocol, i.e., N=4)frames



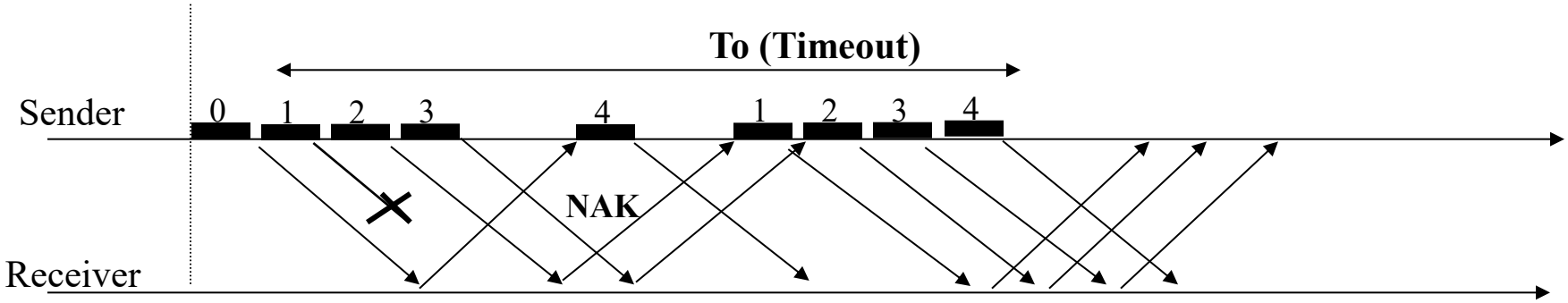
Speeding Up Go Back N

- Use negative acknowledgments



Go Back N

- Correctness: proof is made by induction starting with ABP
- Efficiency:
  - Without error ***E= min(1, NTr/T)***
  - Wasteful retransmissions in case of error



Go Back N

- Efficiency (with errors, PER=p)

Assume: N=T/Tr, i.e., the N that makes the efficiency exactly 1 when there is no error.

Note that when an error occurs, the entire window of N packets must be retransmitted.

Let X= the number of packets sent in order to transmit a packet successfully

$$E[X] = 1*(1-p)+ E(Y+N)p, \text{ where } E[X] = E[Y]$$

$$\text{so } E[X] = 1+N*p/(1-p)$$

so efficiency = 1/E[X] (under the assumption that N=T/Tr and NACK)

Selective Repeat Protocol

- Only packets not acknowledged are resent after timeout.
- Received packets are buffered at the receiver until a contiguous segment is received (receive window size = N (i.e., W))
- Efficiency:
  - Without error ***E= min(1, N.Tr/T)***

- Efficiency with errors (under idealized scenario): 1-p

Take-away Messages

- SWP is not efficient when propagation delay is high
- Sliding window protocols are needed on path with high propagation delay
- SRP is the best, but requires larger buffers than Go Back N.
- PER is critical to performance
- Frame size has an impact on performance

Design and Analysis of Computer Networks
COMP 5320/6320/6326

Midterm 1 Review

Instructor: Tao Shu, Ph.D.
CSSE
Auburn University

Topics

- ❑ OSI/Internet layered architecture
- ❑ Typical application-layer protocols
- ❑ Socket programming
- ❑ Error control and ARQs
- ❑ Discrete-time Markov chain

Protocols!

- A common set of rules accepted and followed by communicating parties (between peers of the same layer)
- *To communicate, we must use*
  - Common language
  - Common set of rules
  - Example: network calculator

- Different levels of communications

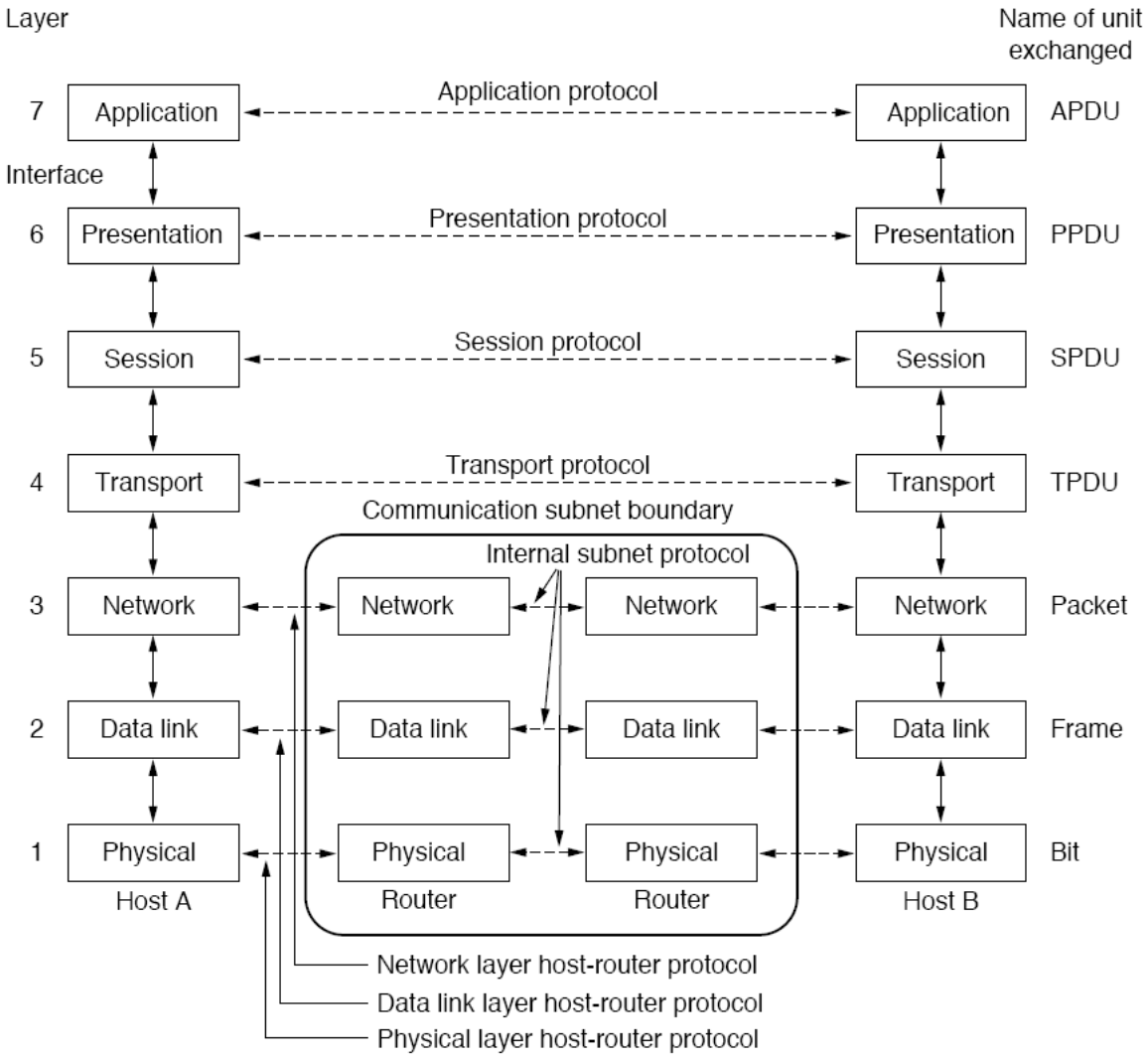
The OSI Network Architecture

OSI: Open Systems Interconnection

7 layers X. protocol specifications for each layer

Acts like a reference model rather than a real-world protocol graph

First three layers are implemented in all nodes



Layers of the OSI model

Physical Layer: Transmission/reception of raw bits

Data Link Layer: Maps bits into frames, dictates sharing of common medium, corrects/detects errors , re-orders frames

Network Layer: Routes packets to destination, may perform fragmentation and re-assembly.

Transport Layer: Flow (congestion) control, error control, transparent transport to upper layers

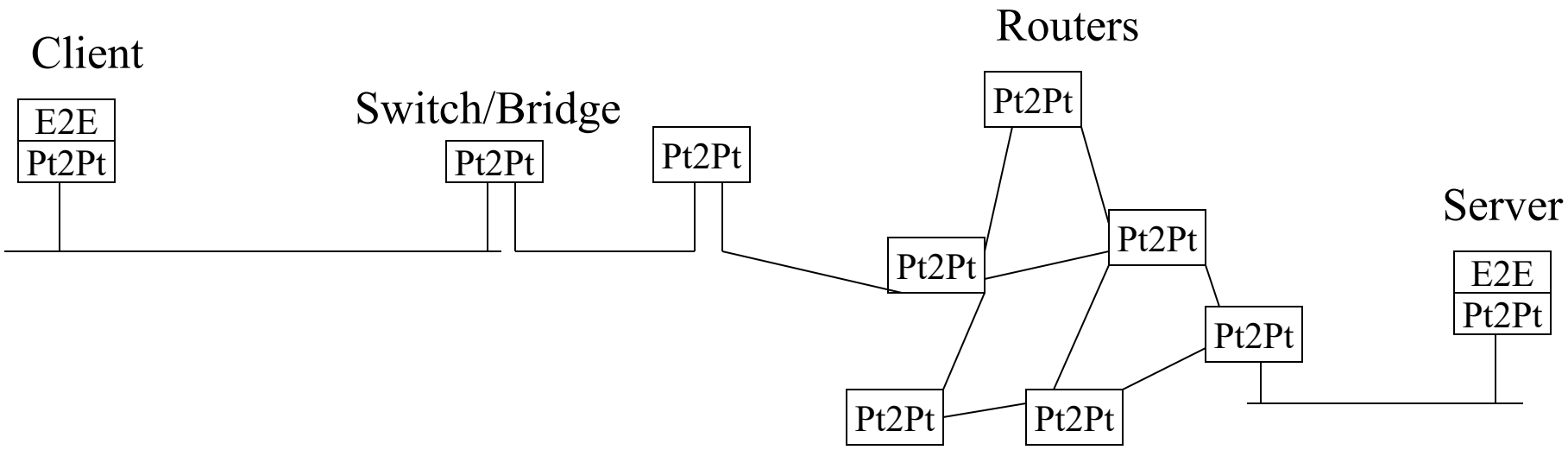
Session Layer: Establishes connection among hosts, duplex, half-duplex, graceful connection termination, combination of streams

Presentation Layer: Negotiation of format of data exchanged between hosts

Application layer: Application services such as FTP, X.400 (mail), HTTP

Two Types of Communication Entities

- **End-to-end**:
  - Used only at the end points of communication sessions
- **Point to point**:
  - Used on every relaying device



End-to-End Protocols (Examples)

- Applications: HTTP, FTP, Telnet, DNS

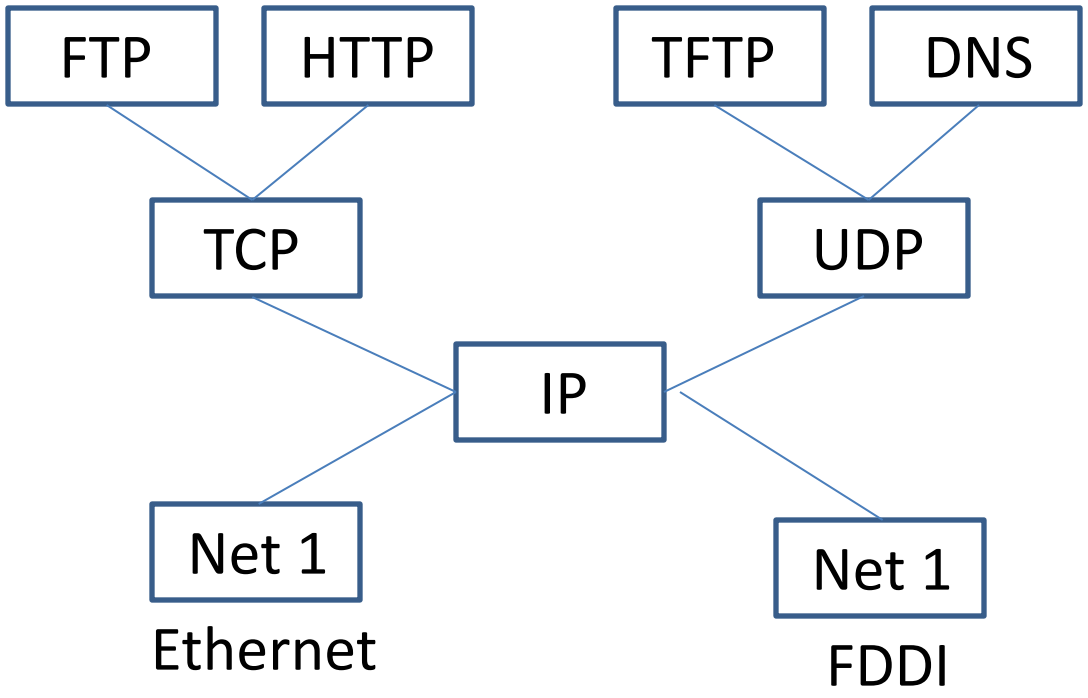
- Communication tools: TCP, UDP..

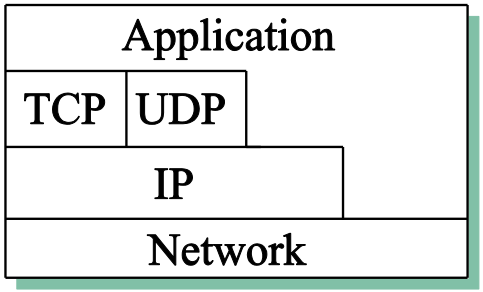
Point to Point

- Protocols: IEEE 802.x., HDLC, Bluetooth, SLIP, PPP

- Devices: Ethernet NIC (IEEE 802.3), wireless LAN card (IEEE 802.11), DSL, modem, ATM switch

The Internet Architecture





FTP: File Transfer Protocol

HTTP: Hypertext Transport Protocol

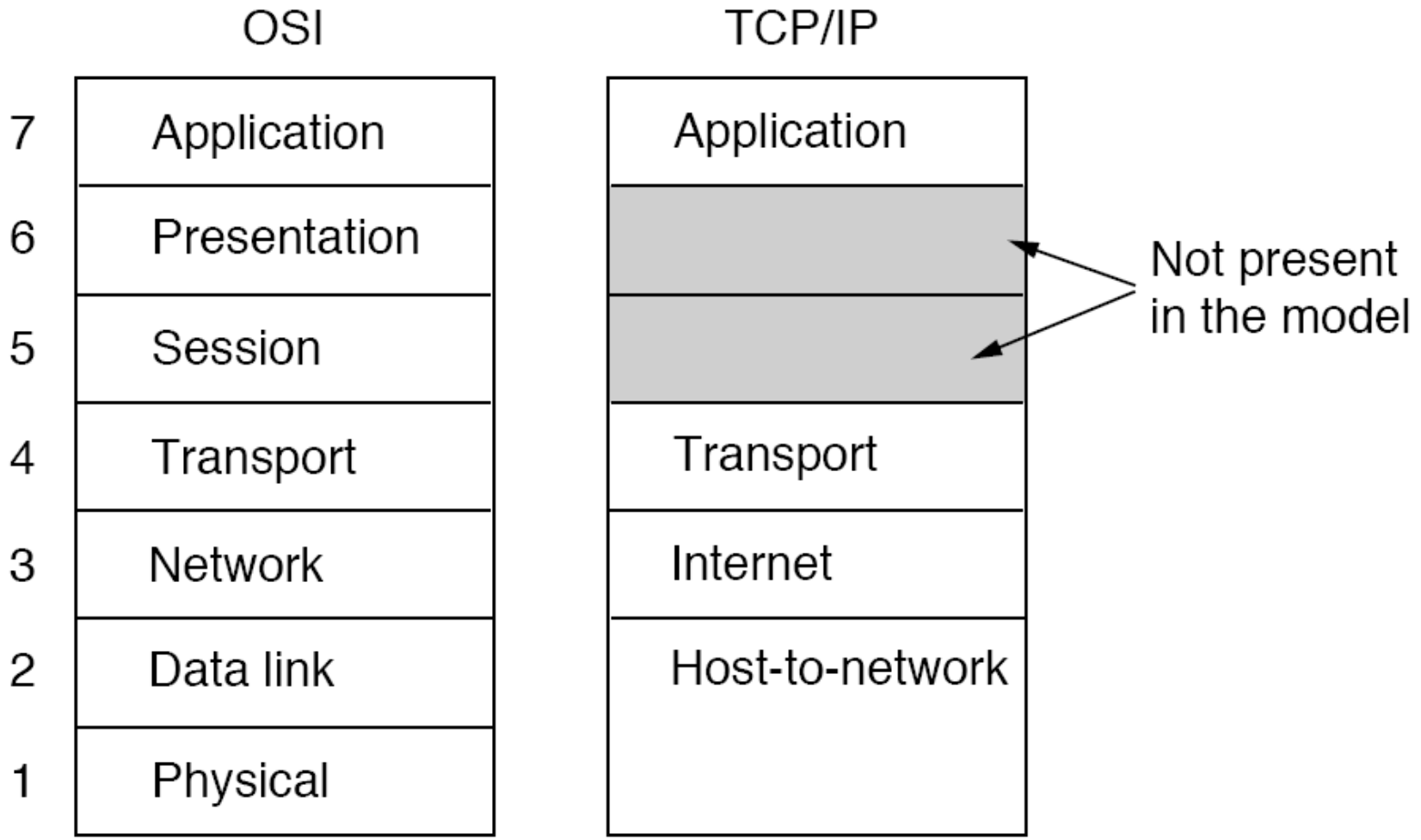
TFTP: Trivial File Transfer Protocol

TCP: Transmission Control Protocol

UDP: User Datagram Protocol

IP: Internet Protocol

Comparison of the two architectures



Domain Name System (DNS)

RFC 1034 and RFC 1035

DNS: What ?

- It is a service used to map host names into IP addresses for:
  - Web URLs
  - Email addresses

- Based on a Client-Server architecture
- Each machine connected to the net must know the IP address of its DNS server.

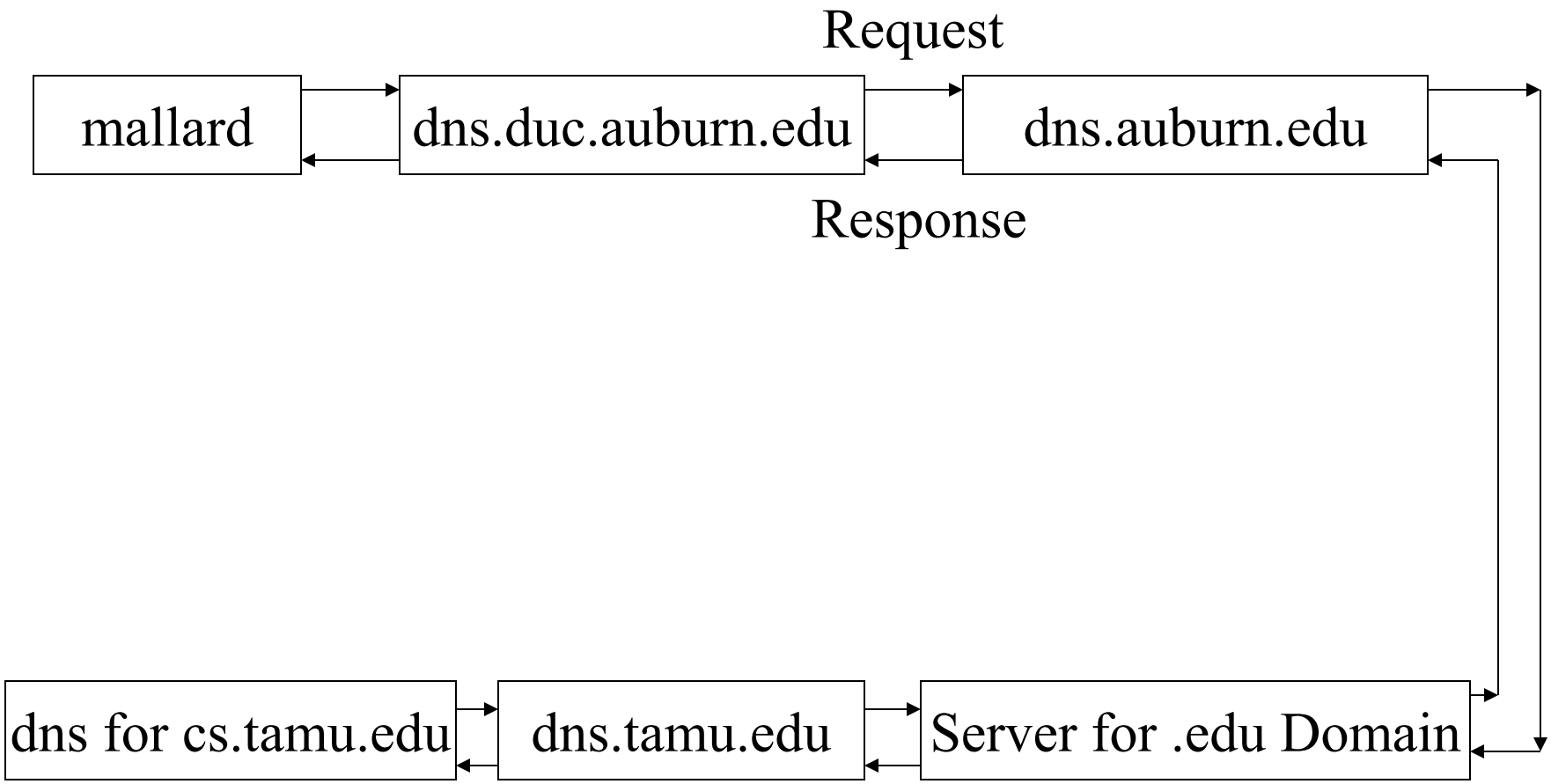
DNS: Why ?

- Human do not like to use IP addresses (as numbers)
- Human want prefer to designate machines with names
- Interconnecting devices on the net use IP addresses (as numbers)
- Huge number of hosts on the Internet

DNS: How ?

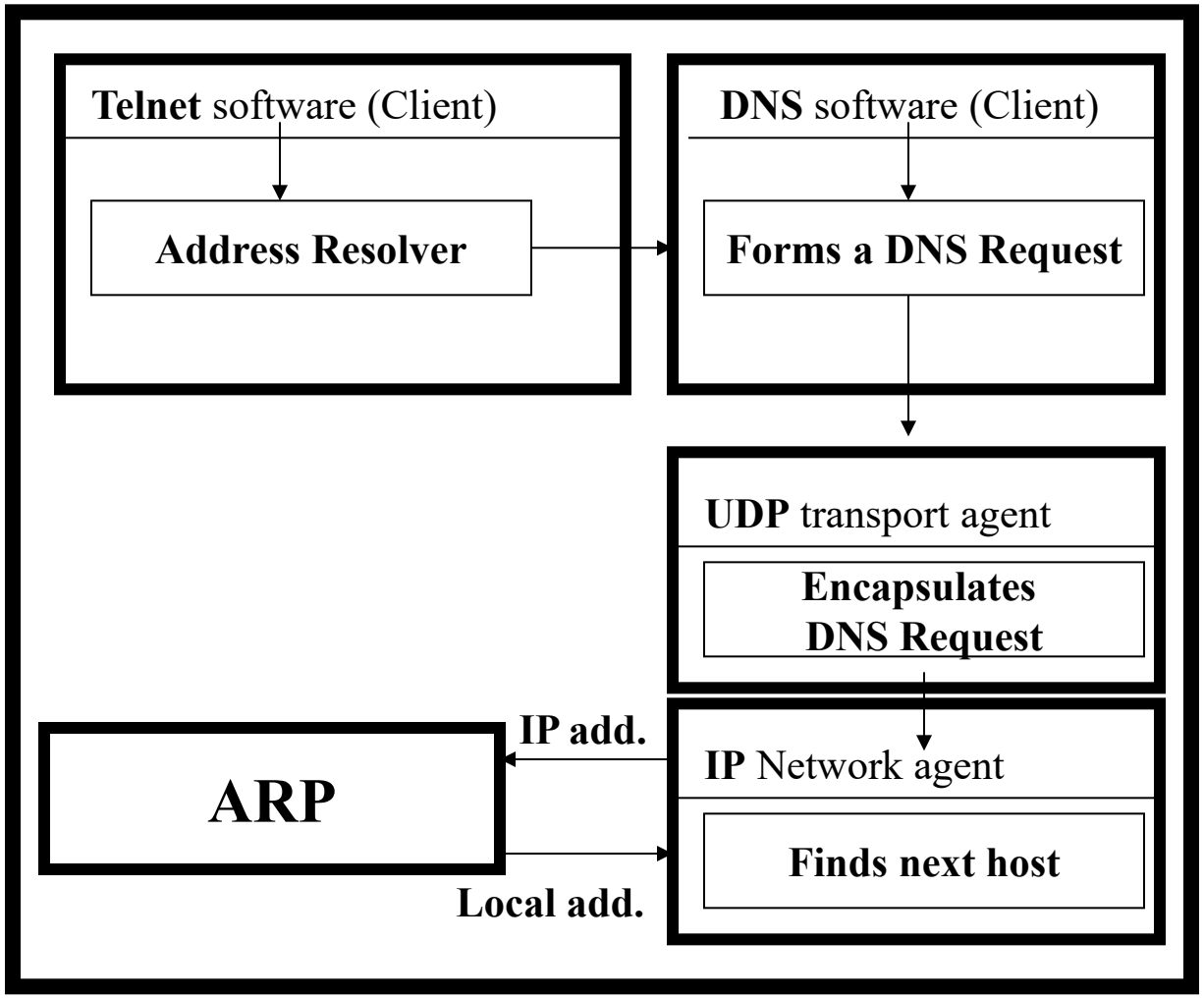
- Use a file (e.g., hosts.txt) with all machine names and mapping?
- Use a local server that contains the database related to all hosts on the Internet?
- Use a distributed database system

Requesting the IP of bach.cs.tamu.edu from mallard.eng.auburn.edu (Recursive)

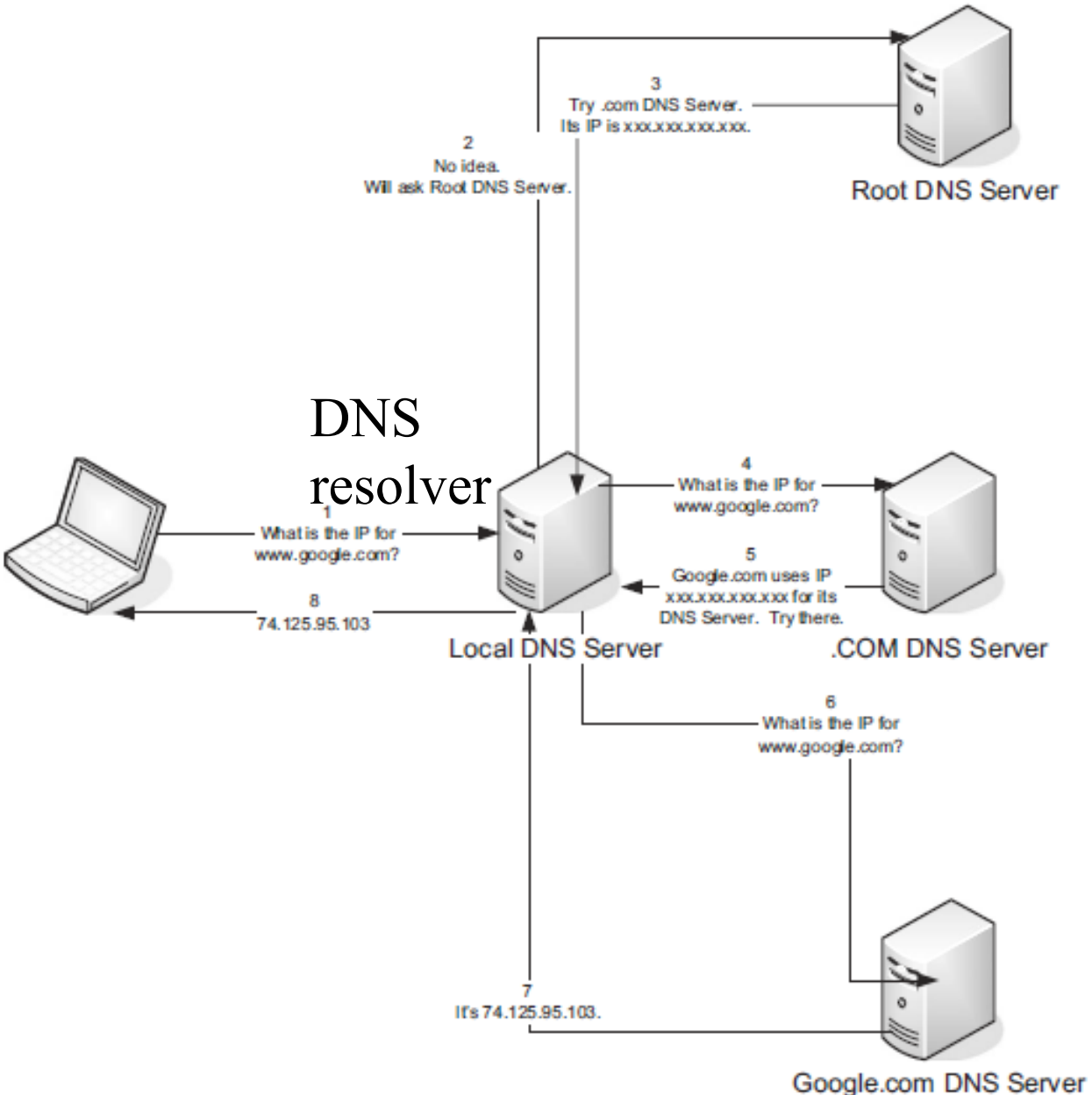


How to Resolve IP Address → Local Address?

strangelove.eng.auburn.edu host



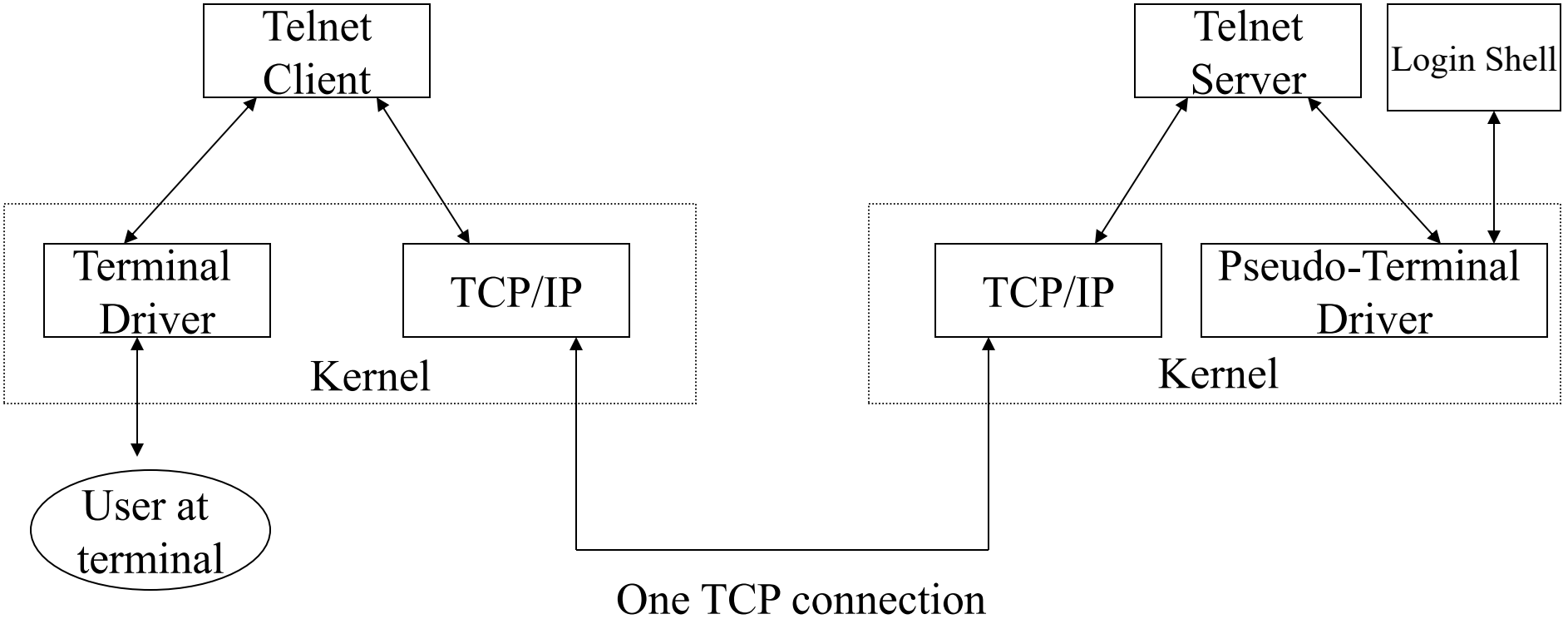
Iterative DNS Query



Iterative Query

Authoritative vs.
non-authoritative

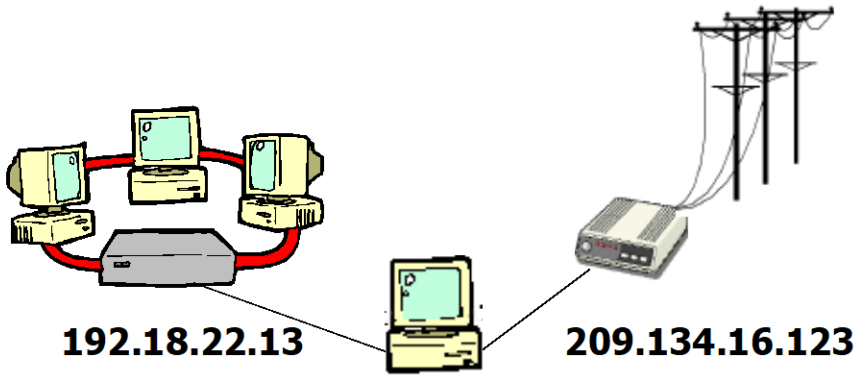
Telnet: the Big Picture
(Port 23)



Sockets...

IP Address

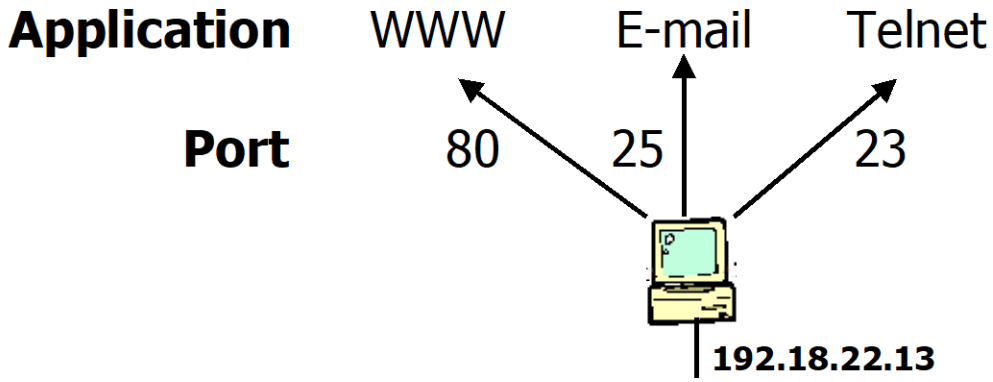
- 32-bit identifier
- Dotted-quad: 192.118.56.25
- www.mkp.com –> 167.208.101.28
- Identifies a host interface (not a host)



Ports

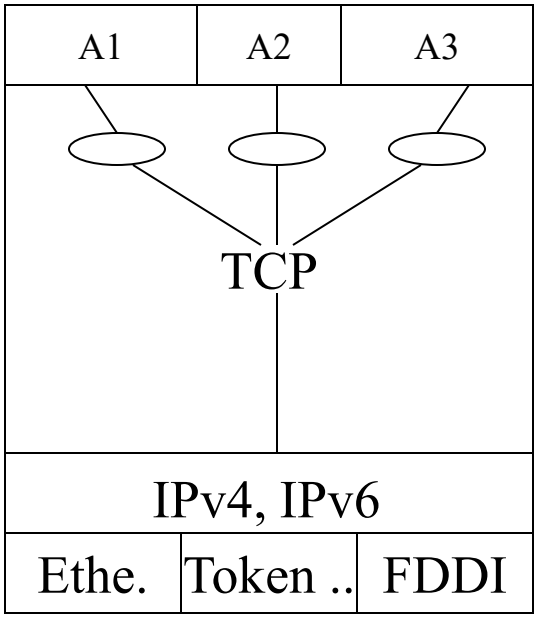
Identifying the ultimate destination

- IP addresses identify hosts
- Host has many applications
- Ports (16-bit identifier)

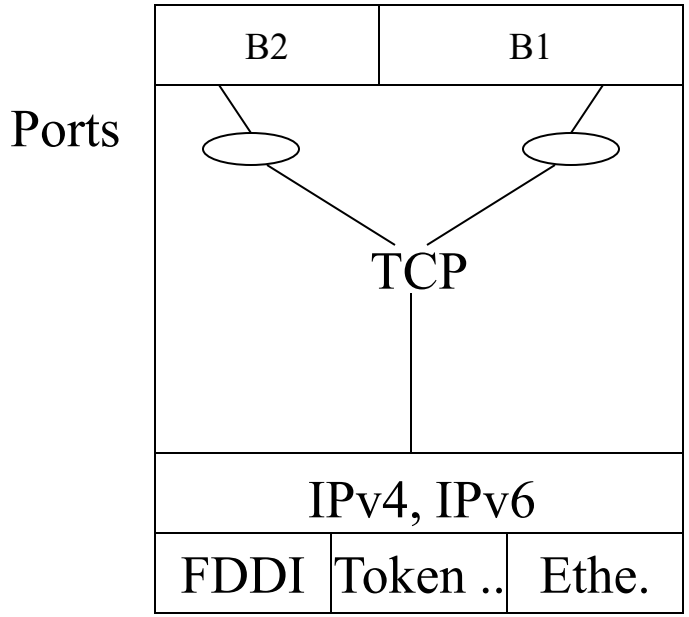


TCP Sockets

TCP Client



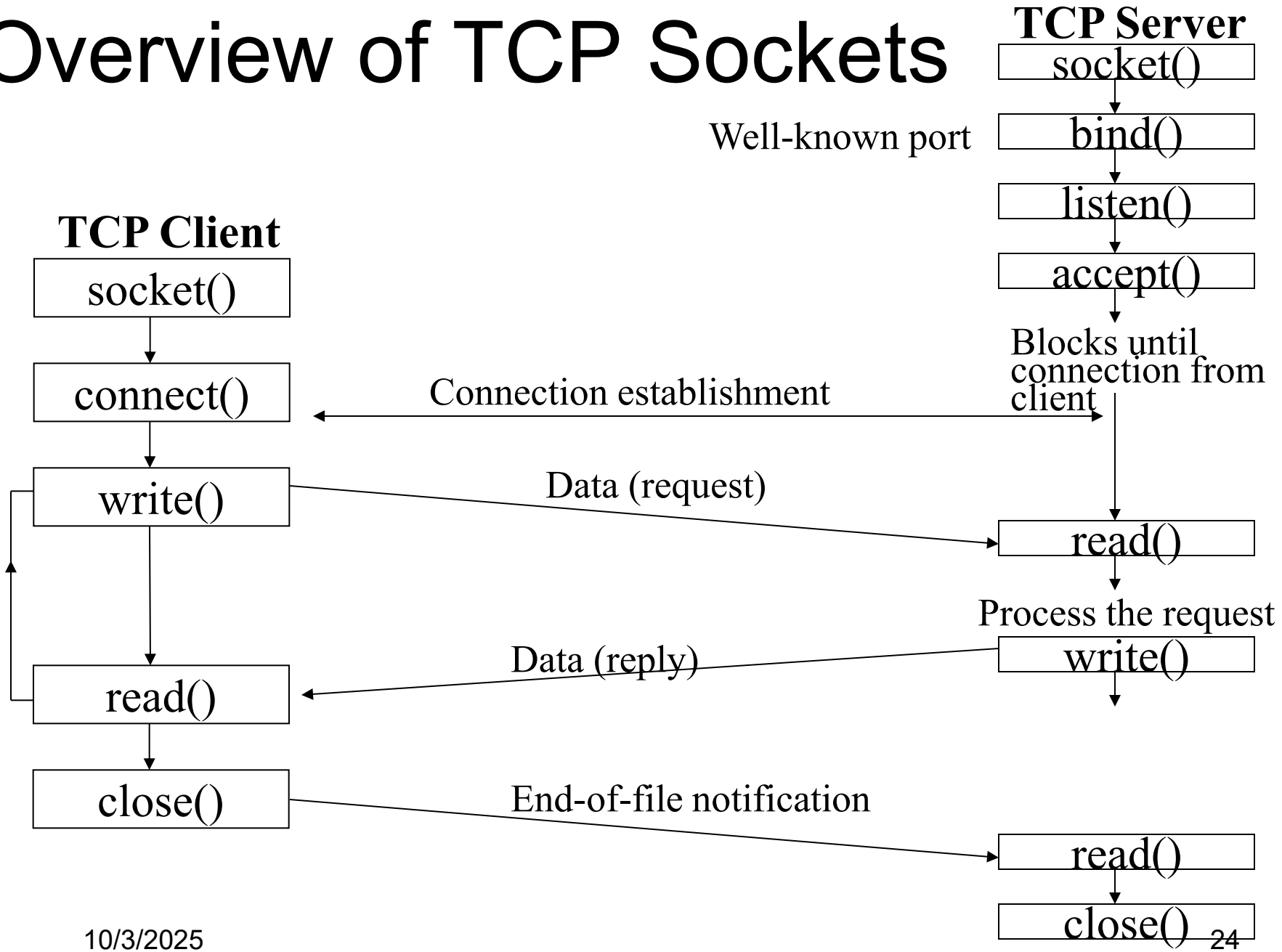
TCP Server



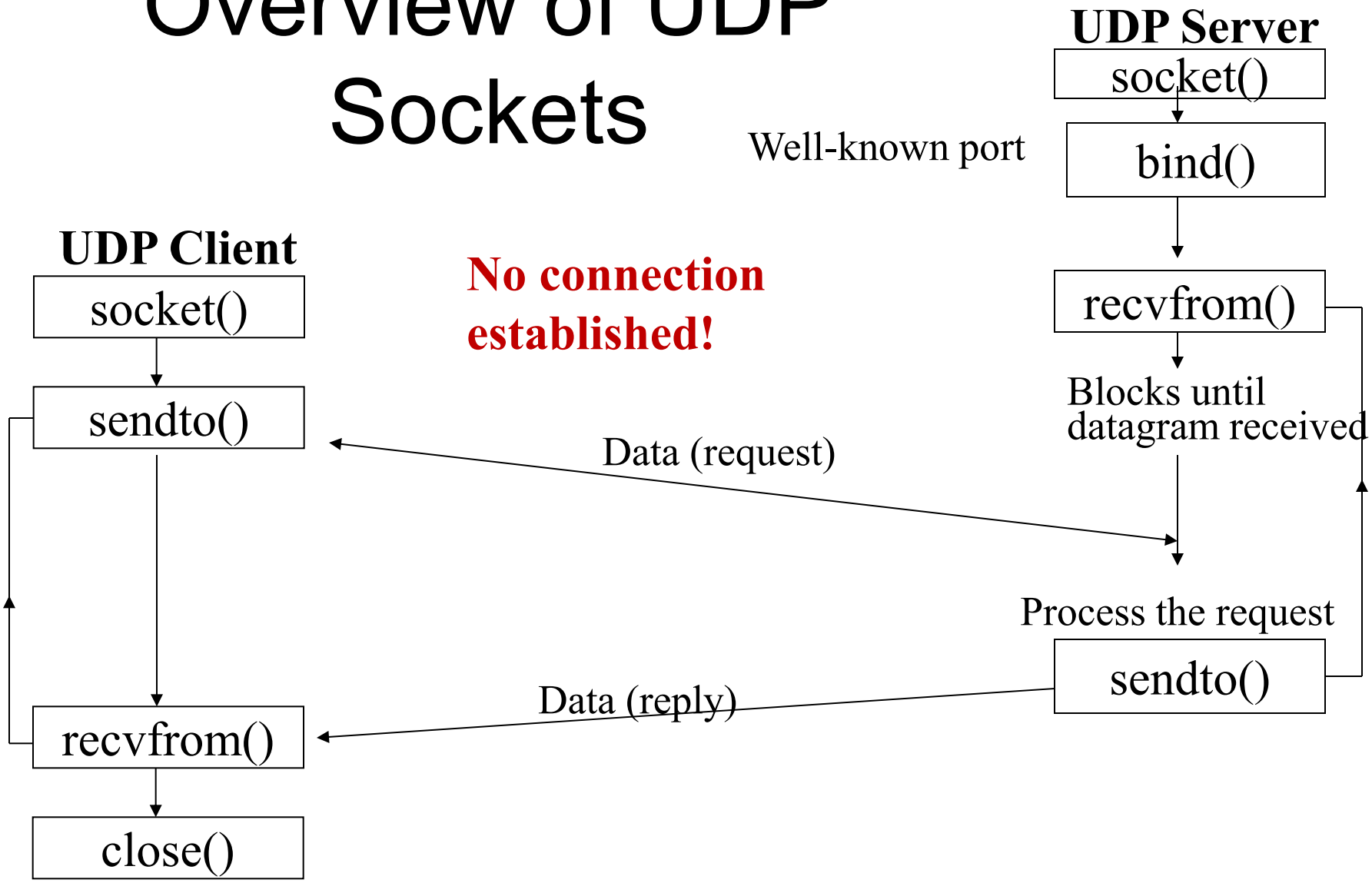
Some Well Known Ports

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FTP, Data	File Transfer Protocol (data connection)
21	FTP, Control	File Transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

Overview of TCP Sockets



Overview of UDP Sockets



Error Control

Types of Errors

- **Bit corruption:** one or multiple bits in a frame are corrupted (0 flipped to 1, or 1 flipped to 0) during transmission. Results in packet corruption. Bit error rate (BER) very low on terrestrial links. Derive relationship between BER and packet error rate PER (assuming a packet with n bits).

What is the probability that a packet is not corrupted? 1-PER

It is also the probability that no bit in a frame is corrupted $(1\text{-BER})^N$

Then $1\text{-PER} = (1\text{-BER})^N$

$$\text{PER} = 1-(1-\text{BER})^N$$
$$\sim\sim N.\text{BER if BER very small}$$

- **Packet error:** loss (due to congestion), duplication, or reordering

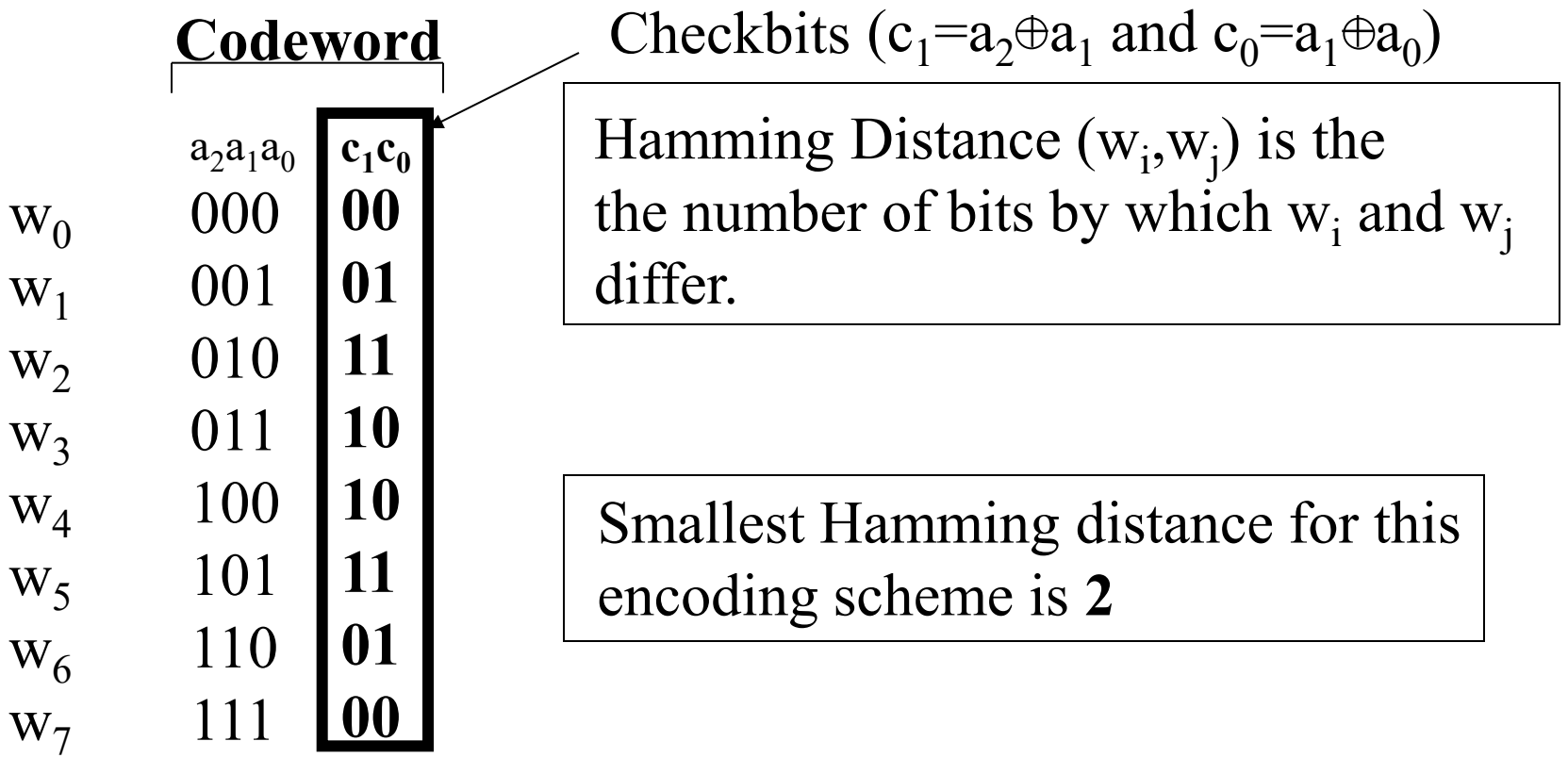
Error Control (Where?)

- Link Layer (point-to-point)
  - Point-to-point protocol (PPP) (*Detection only*)
  - High-level Data Link Control (HDLC) (*AQM protocol*)

- Transport Layer (end-to-end)
  - User Datagram Protocol (UDP) (*Detection only*)
  - Transmission Control Protocol (TCP) (*AQM*)

Error Detection (General)

- Consider an alphabet of words $\{w_i\}$



Requirements for Detection or Correction

- To **detect *d*** corrupted bits, the encoding scheme needs to have a minimal hamming distance of ***d+1***

- To **correct *d*** corrupted bits, the encoding scheme needs to have a minimal hamming distance of ***2.d+1***

Examples

- Cyclic Redundancy Check (CRC)
- BCH (Bose-Chaudhury-Hocquenghem) codes
- Reed-Solomon code
- Turbo codes
- LDPC codes
- Convolutional codes

Main Strategies

- Error Detection and retransmission: *data is encoded such that errors can be detected. When an error occurs, sender must retransmit the packet*

- Error Correction: *data is encoded such that errors are detected and corrected with no retransmission.*

Retransmission Protocols

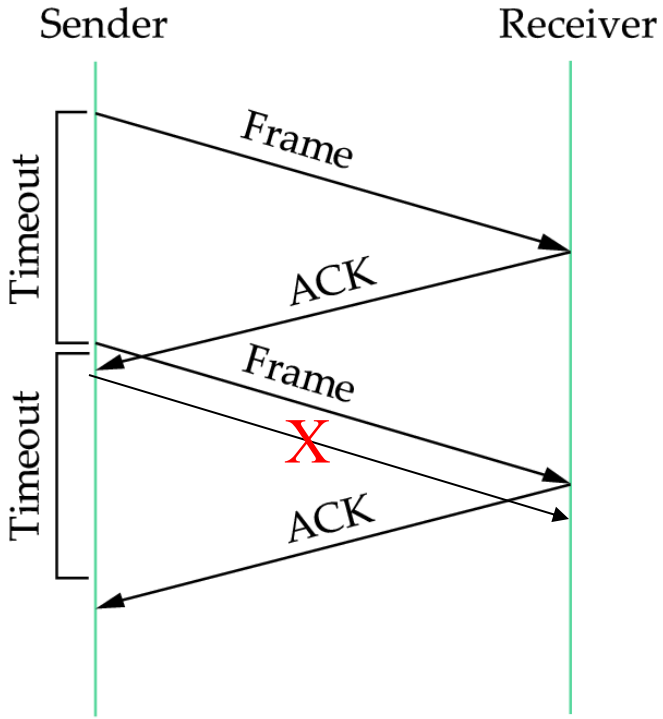
- Stop and wait protocol (SWP)
- Alternating Bit Protocol (ABP)
- Selective repeat protocol (SRP)
- Go back n (GBN)

Correctness of SWP

- **Operation of SWP**:
  - Whenever a receiver gets a correct packet, it sends back an acknowledgment (without sequence #).
  - The sender resends a packet previously sent if it did not get an acknowledgement from the receiver within a **known bound T_o** (Timeout value)
  - Sender, alternately, numbers the packets with 0 or 1.

Alternating Bit Protocol (ABP)

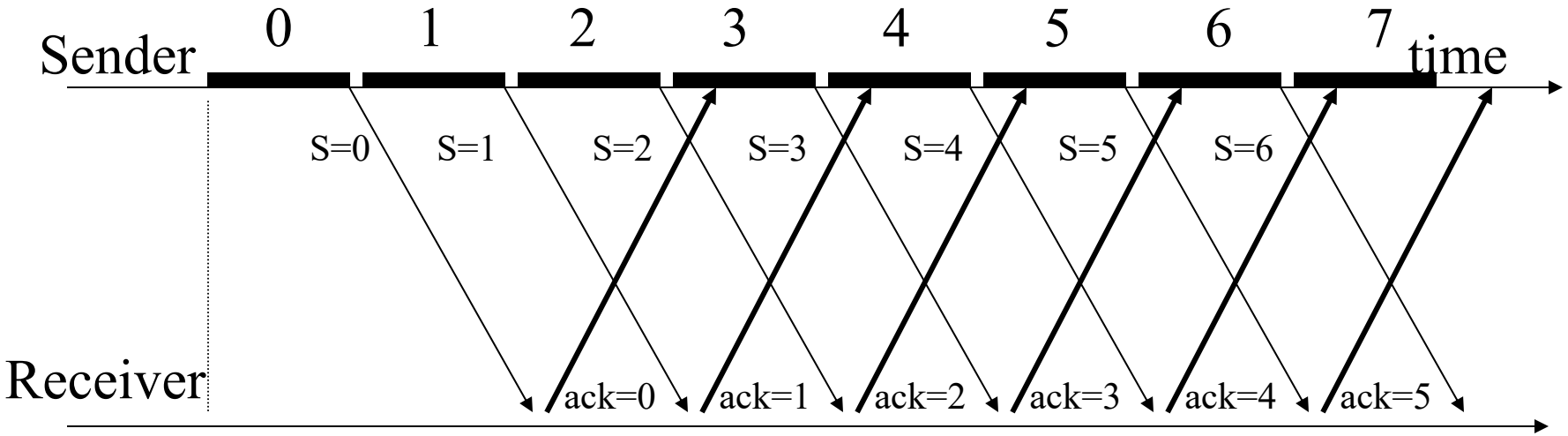
- We do not know the right **bound T_o** (Timeout value is hard to decide).
- If the bound is unknown, we may confuse acknowledgements (is ack for new packet, or for some OLD packet thought lost).



- What to do?
- Put sequence numbers on acks

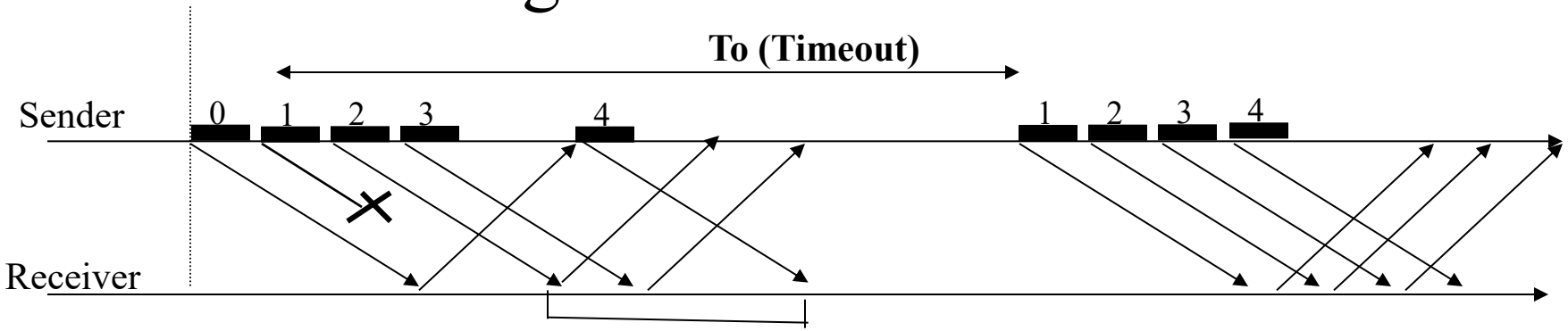
Sliding Window Protocol

- Keep sending packets without waiting for acknowledgements.
- How many packets should be outstanding without ack?
- The number of oustanding frames is the window size ***W***



Go Back N

- If packet # ***i*** is not acked, the sender will resend all packets starting packet # ***i***. (receiver doesn't buffer)
- We are using a window of ***W***=4 frames



Discarded

Selective Repeat Protocol

- Only packets not acknowledged are resent after timeout.
- Received packets are buffered at the receiver until a contiguous segment is received
- Efficiency:
  - Without error ***E= min(1, N.Tr/T)***

- Efficiency with errors (under idealized scenario): 1-p

DTMC

- Memoryless property
- Poisson process and exponential inter-arrival time
- State transition diagram and classification of states
- Stationary distribution
- First passage time