# Introduction to Computer Network

Chapter 1 :: Foundation

Warin Wattanapornprom PhD.

# Course Objectives

- To help you gain a general understanding of the principles and concepts governing the operations of computer networks;
- To provide you with the opportunity to become skillful in the implementation and use of communication protocols;
- To help you grasp the basic research methodologies in the field of computer networks

# Textbooks

- Computer Networks, Sixth Edition, Andrew S. Tanenbaum
- Computer Networks: A Systems Approach, 5e , Larry L. Peterson and Bruce S. Davie
- Computer networks: Performance and quality of service, Marsic, Ivan
- Computer Networking: A Top-Down Approach 6e, J.F. Kurose and K.W. Ross

# Topics

- Overview of network layers and protocols
- The Physical Layer
  - transmission media
  - the Nyquist limit and the Shannon limit
- The Data Link Layer
  - framing
  - error detecting and correcting codes
  - sliding window protocols
- The Medium Access Control Layer
  - IEEE 802.3
  - IEEE 802.11

# Topics (cont)

- The Network Layer
  - routing algorithms
  - congestion control
  - IP
- The Transport Layer
  - connection establishment/termination
  - multiplexing
  - flow control
  - TCP and UDP
- Utility Protocols

# Evaluation

- Assignment: 20%
- Midterm exam: 40%
- Final exam: 40%

# Introduction to Computer Network

Chapter 1 :: Foundation

Warin Wattanapornprom PhD.

# Problems

- How to build a scalable network that will support different applications?
- What is a computer network?
- How is a computer network different from other types of networks?
- What is a computer network architecture?

# Chapter Goal

- Exploring the requirements that different applications and different communities place on the computer network
- Introducing the idea of **network architecture**
- Introducing some key elements in implementing Network Software
- Define key metrics that will be used to evaluate the performance of computer network

# Chapter Outline

- Applications
- Requirements
- Network Architecture
- Implementing Network Software
- Performance

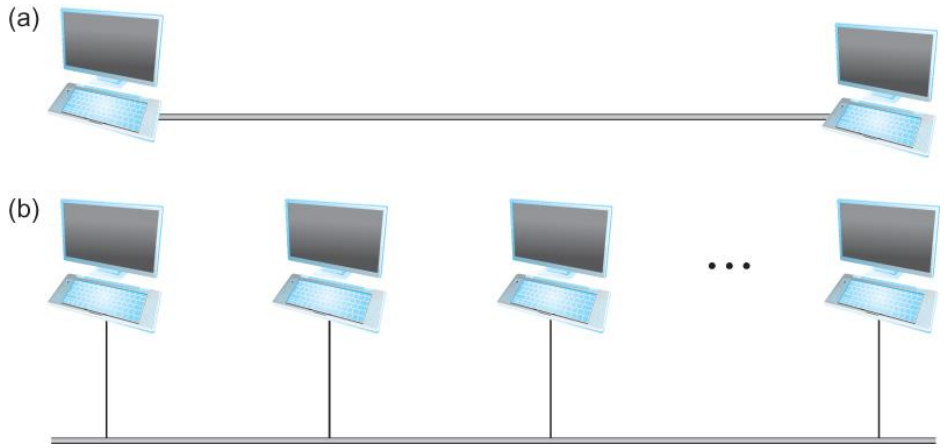
# Applications

- Most people know about the Internet (a computer network) through applications
  - World Wide Web
  - Email
  - Online Social Network
  - Streaming Audio Video
  - File Sharing
  - Instant Messaging
  - …

# Requirements

- Application Programmer
  - List the services that his application needs: delay bounded delivery of data
- Network Designer
  - Design a cost-effective network with sharable resources
- Network Provider
  - List the characteristics of a system that is easy to manage

# Connectivity




Point-to-point

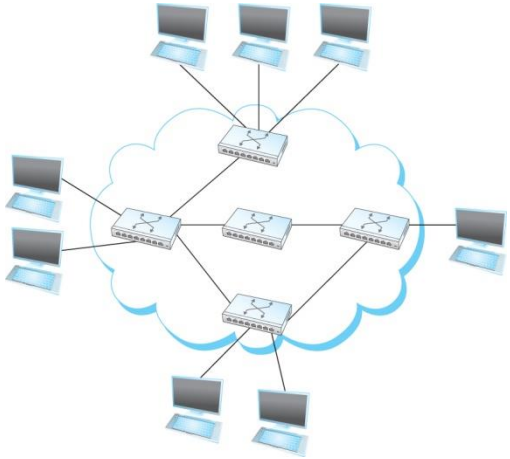
Multiple access

Need to understand the following terminologies

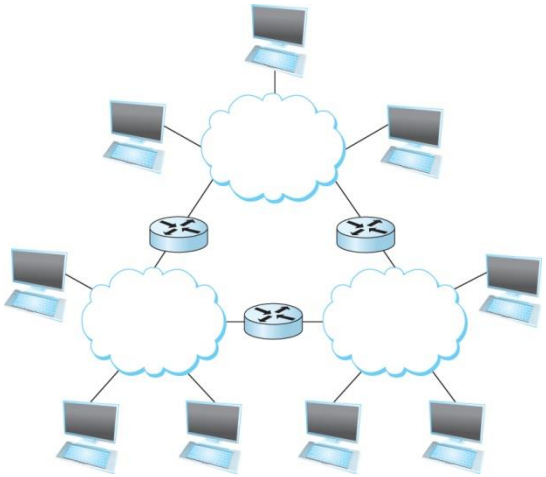
- Scale
- Link
- Nodes
- Point-to-point
- Multiple access
- Switched Network
  - Circuit Switched
  - Packet Switched
- Packet, message
- Store-and-forward

# Connectivity

(a)



(b)



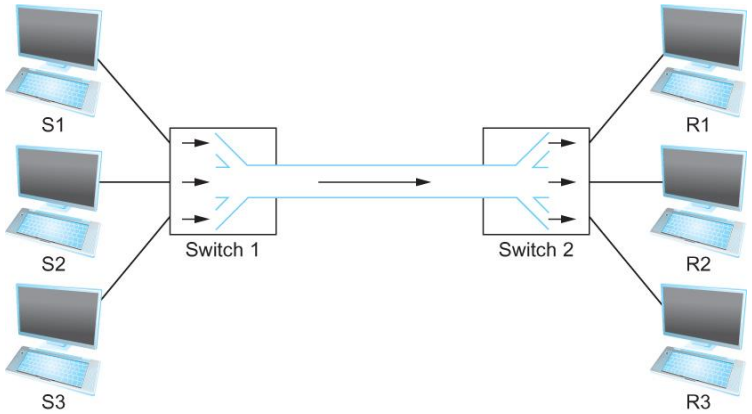
A switched network

Interconnection of networks

## Terminologies (contd.)

- Cloud
- Hosts
- Switches
- internetwork
- Router/gateway
- Host-to-host connectivity
- Address
- Routing
- Unicast/broadcast/multicast

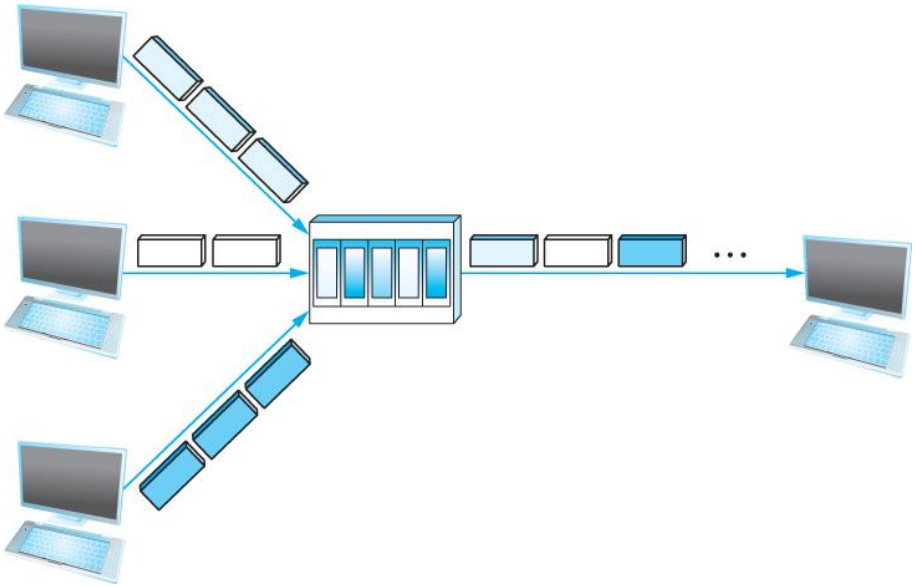
# Cost-Effective Resource Sharing




Multiplexing multiple logical flows over a single physical link

- Resource: links and nodes
- How to share a link?
  - Multiplexing
  - De-multiplexing
  - Synchronous Time-division Multiplexing
    - Time slots/data transmitted in predetermined slots

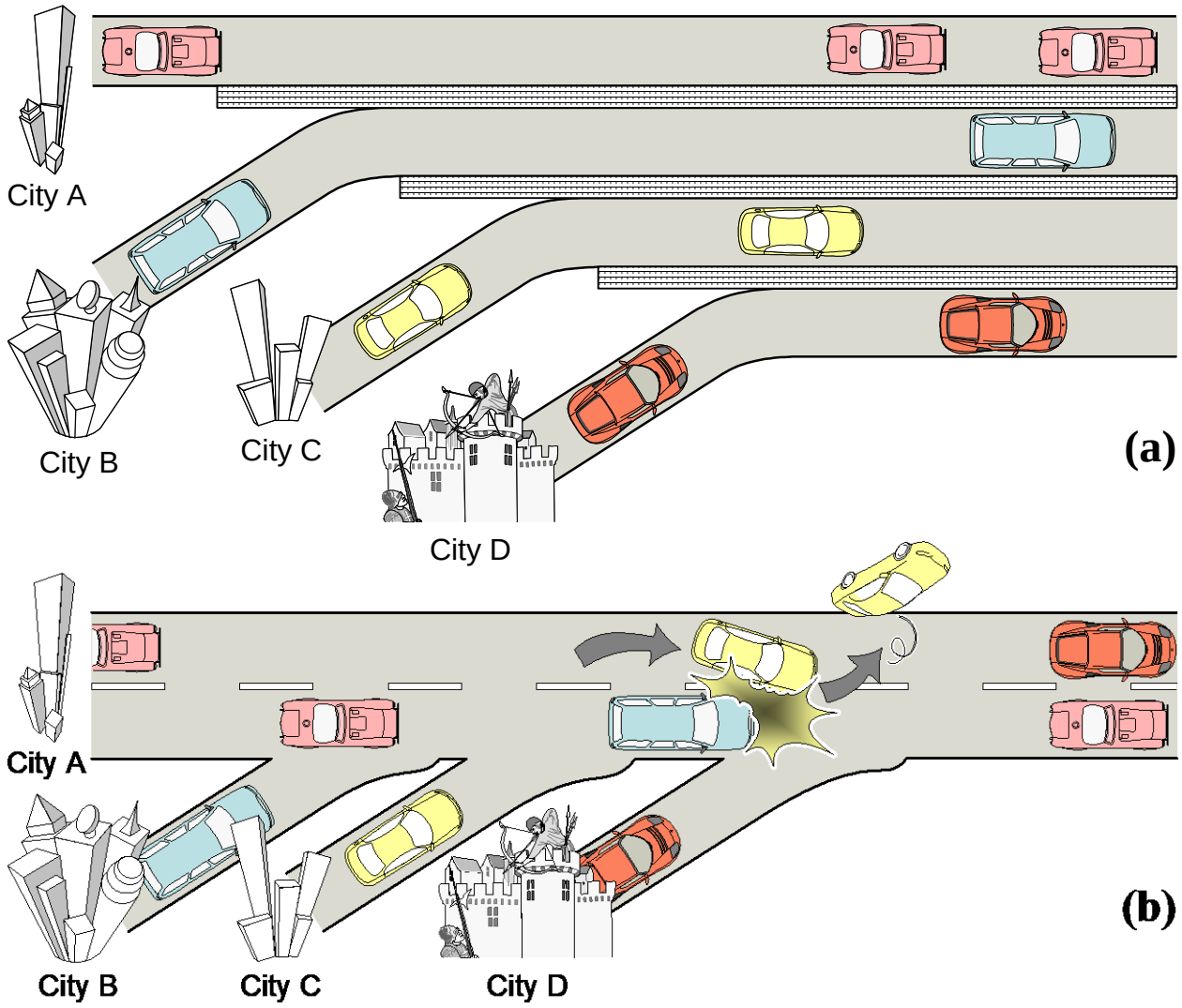
# Cost-Effective Resource Sharing



A switch multiplexing packets from multiple sources onto one shared link

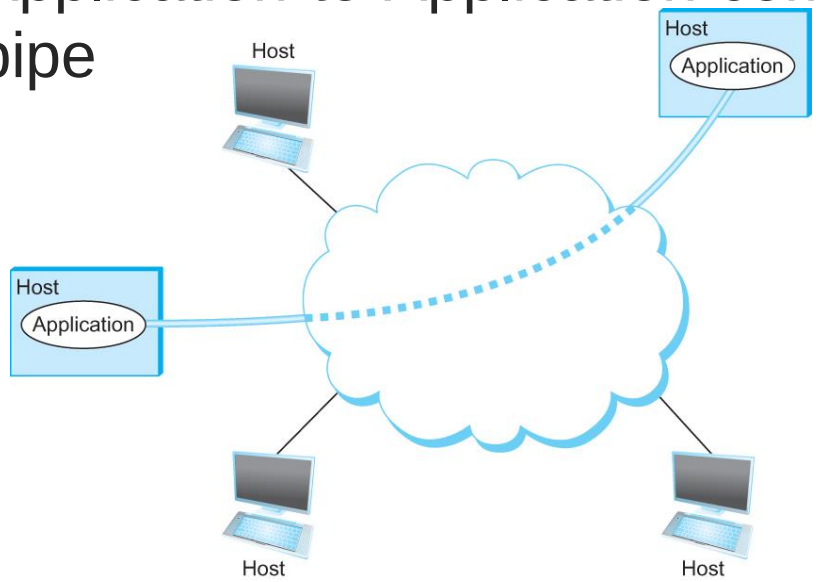
- FDM: Frequency Division Multiplexing
- Statistical Multiplexing
  - Data is transmitted based on demand of each flow.
  - What is a flow?
  - Packets vs. Messages
  - FIFO, Round-Robin, Priorities (Quality-of-Service (QoS))
  - Congested?
- LAN, MAN, WAN
- SAN (System Area Networks

# Statistical Multiplexing



# Support for Common Services

- Logical Channels
  - Application-to-Application communication path or a pipe




Process communicating over an abstract channel

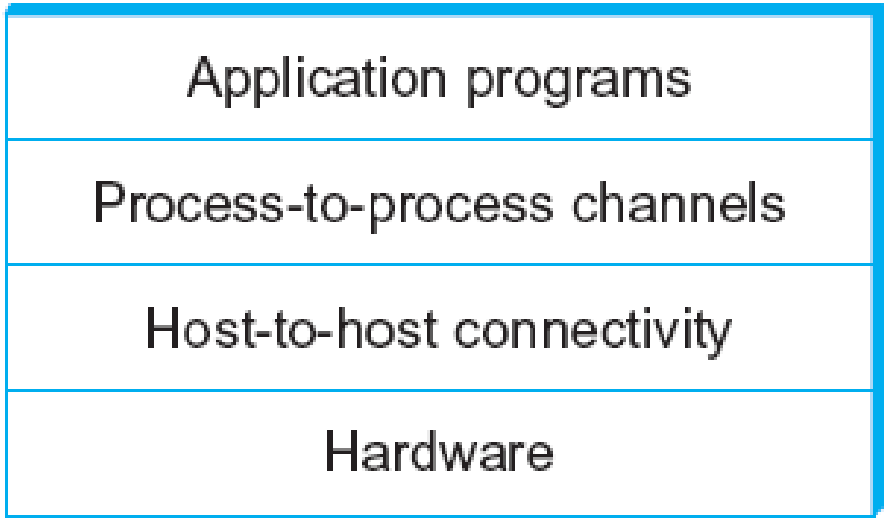
# Common Communication Patterns

- Client/Server
- Two types of communication channel
  - Request/Reply Channels
  - Message Stream Channels

# Reliability

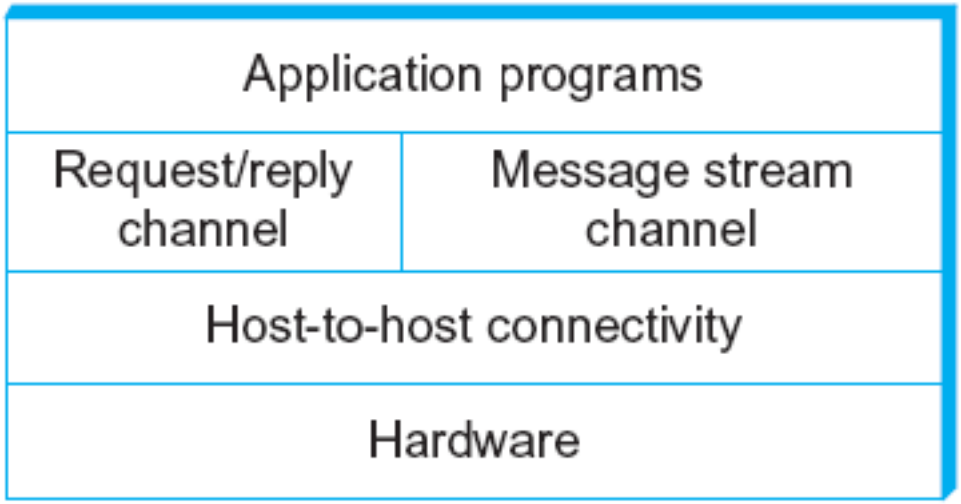
- Network should hide the errors
- Bits are lost
  - Bit errors (1 to a 0, and vice versa)
  - Burst errors – several consecutive errors
- Packets are lost (Congestion)
- Links and Node failures
- Messages are delayed
- Messages are delivered out-of-order
- Third parties eavesdrop

# Network Architecture




Example of a layered network system

# Network Architecture




Layered system with alternative abstractions available at a given layer

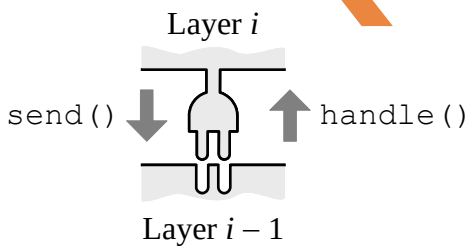
# Protocols

Basically, a protocol is an **agreement** between the communicating **peers** on how communication is to proceed.

# Protocols

- Protocol defines the interfaces between the layers in the same system and with the layers of peer system
- Building blocks of a network architecture
- Each protocol object has two different interfaces
  - service interface: operations on this protocol
  - peer-to-peer interface: messages exchanged with peer
- Term “protocol” is overloaded
  - specification of peer-to-peer interface
  - module that implements this interface

# Protocols




User-to-user interactions obey social norms

Letter (Message)

Person A

Person B

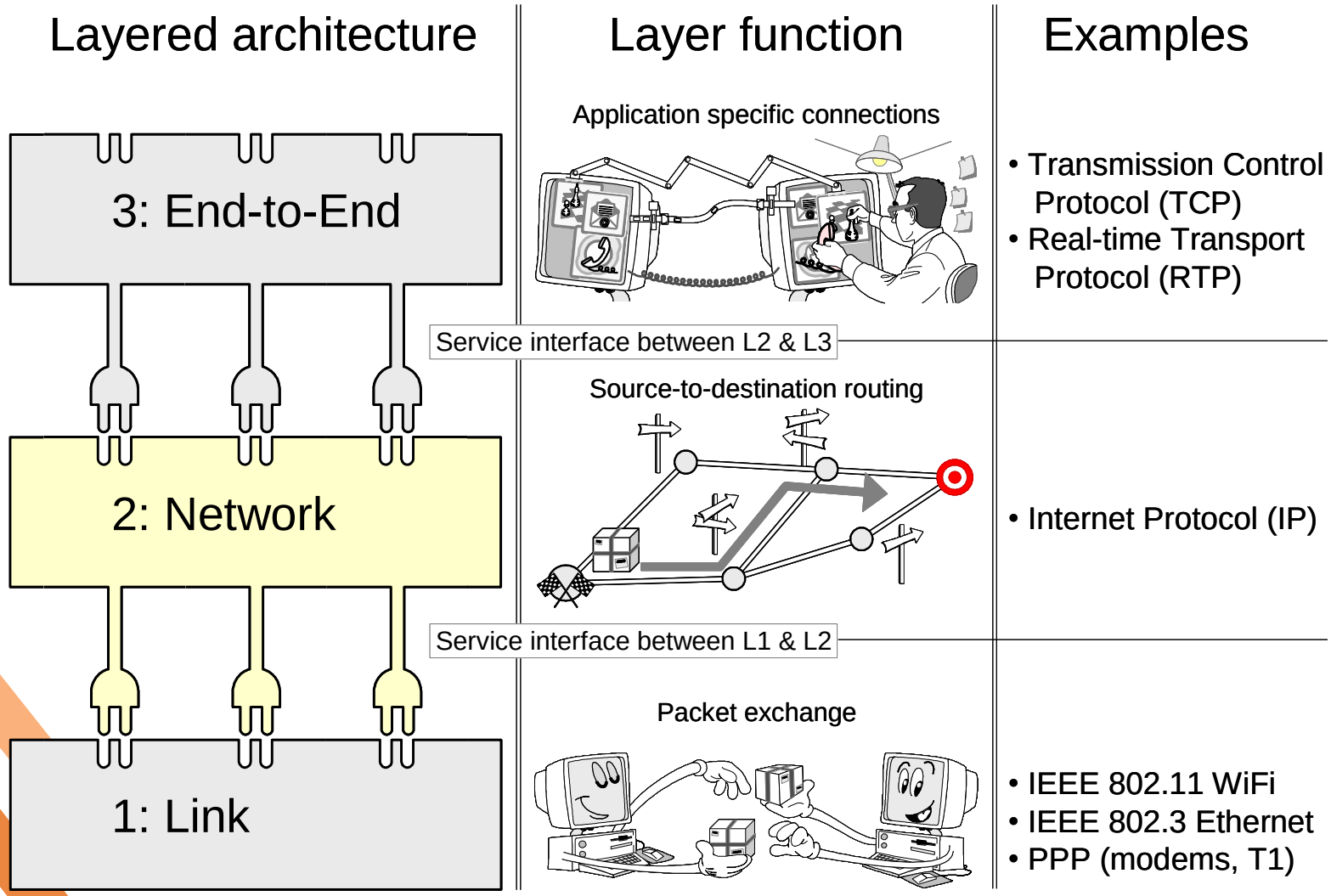
Customer interaction obeys mail acceptance and delivery procedures (Postal Service's Mail Manual)

Letter in envelope (Packet)

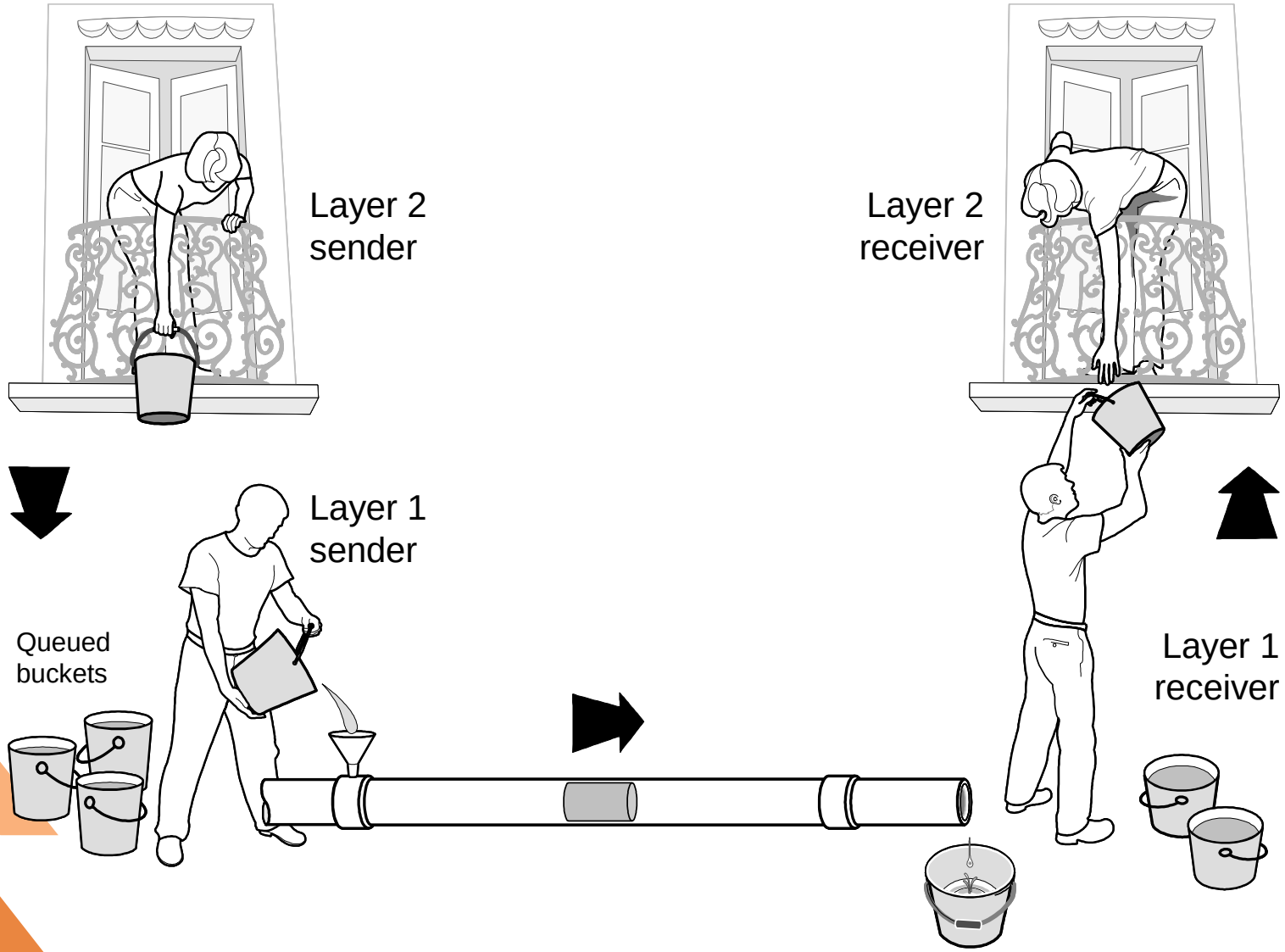
Postal-vehicle service-transportation routes obey carrier-route maps and delivery timetables

Physical transport obeys transportation and traffic rules

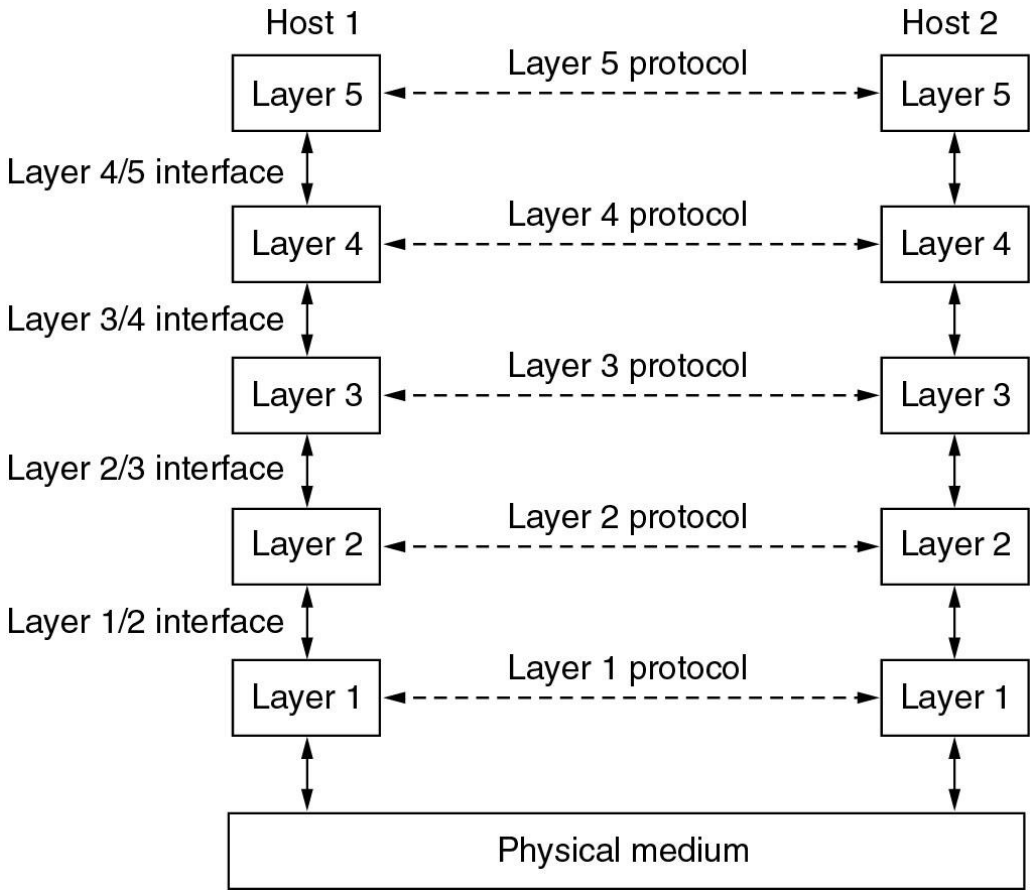
# 3-Layer Protocol Stack



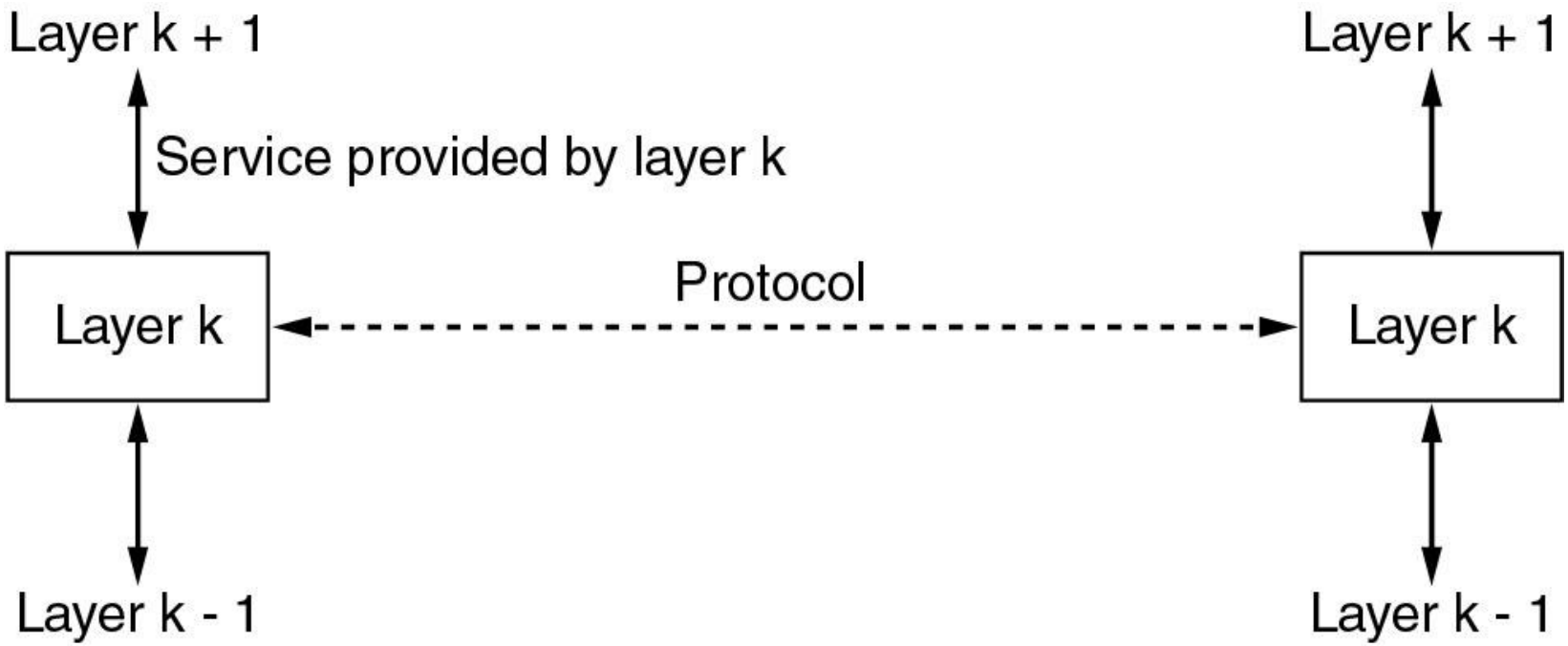
# Fluid Flow Analogy



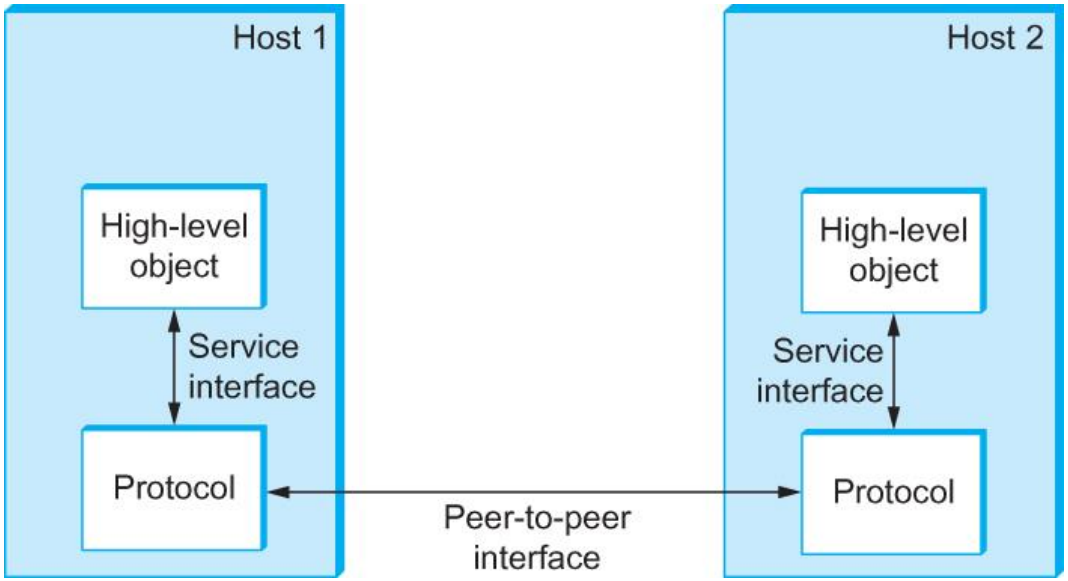
# The layering Principle



# The relationship of Service

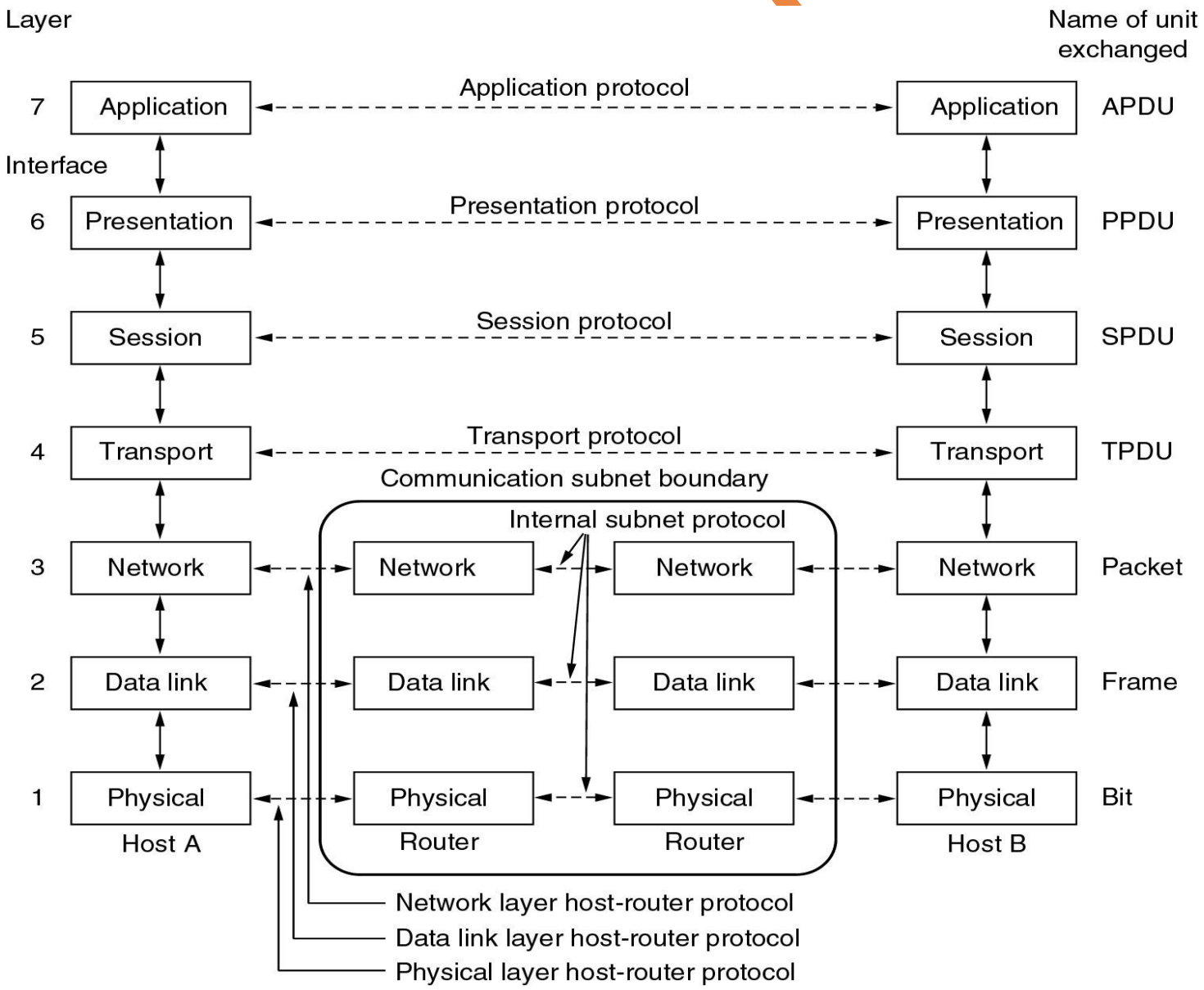


# Interfaces




Service and Peer Interfaces

# The OSI Reference Model



# Description of Layers

- Physical Layer
  - Handles the transmission of raw bits over a communication link
- Data Link Layer
  - Collects a stream of bits into a larger aggregate called a *frame*
  - Network adaptor along with device driver in OS implement the protocol in this layer
  - Frames are actually delivered to hosts
- Network Layer
  - Handles routing among nodes within a packet-switched network
  - Unit of data exchanged between nodes in this layer is called a *packet*

The lower three layers are implemented on all network nodes

# Description of Layers

- Transport Layer
  - Implements a process-to-process channel
  - Unit of data exchanges in this layer is called a *message*
- Session Layer
  - Provides a name space that is used to tie together the potentially different transport streams that are part of a single application
- Presentation Layer
  - Concerned about the format of data exchanged between peers
- Application Layer
  - Standardize common type of exchanges

The transport layer and the higher layers typically run only on end-hosts and not on the intermediate switches and routers

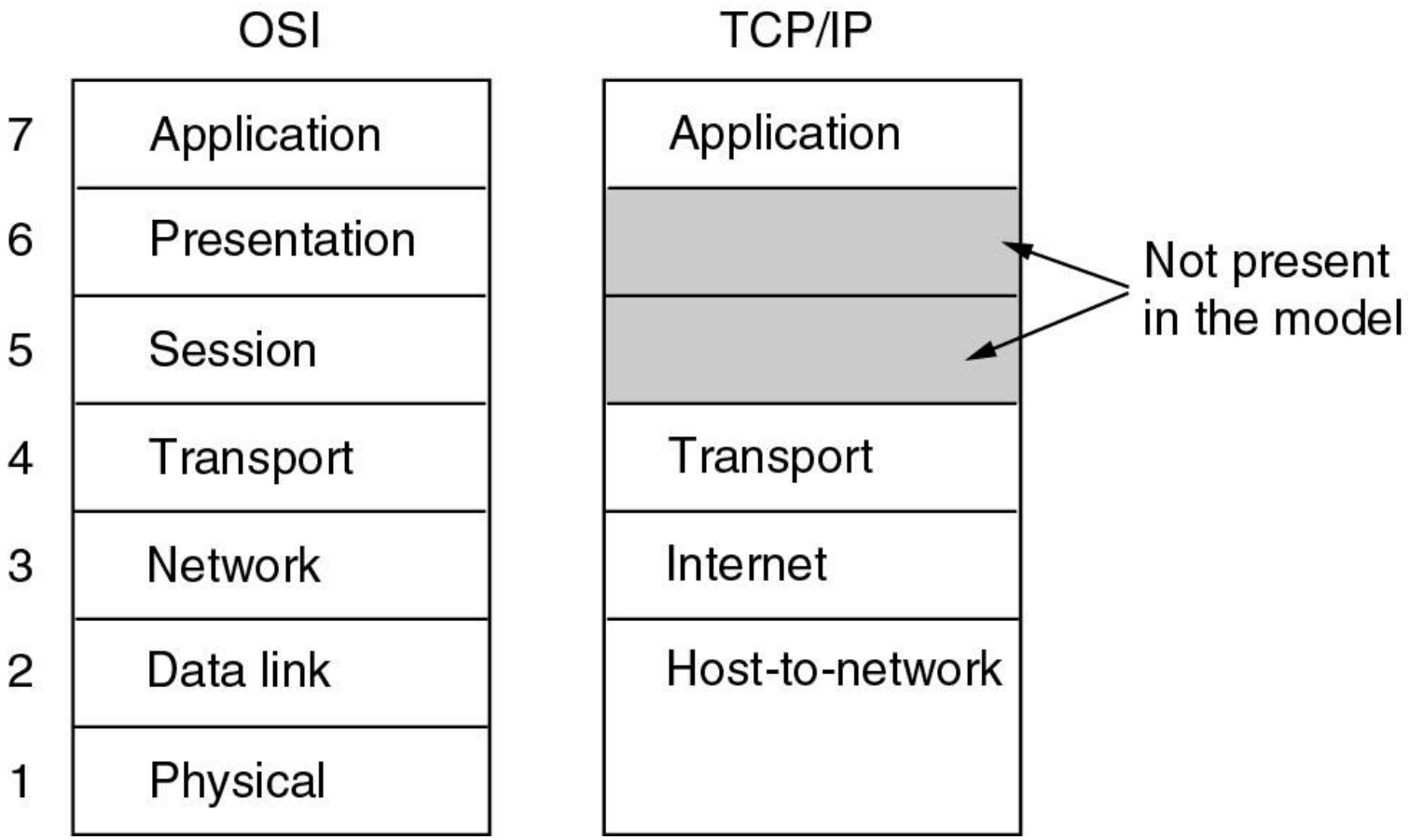
# 7 Layer

- Layer 1: Physical Layer
  - ทำหน้าที่เชื่อมต่อผ่าน Physical Medium รับผิดชอบแปลงบิตเป็นสัญญาณ เรื่องของการ Interface, สายนำสัญญาณ ,มองเห็นข้อมูลในลักษณะ Bit Stream
- Layer 2: Data Link Layer
  - ประกอบข้อมูลเป็น Frame, รับผิดชอบในการสื่อสารผ่าน แต่ละ Link ทำ Error Control, Flow Control ผ่าน Link
- Layer 3: Network Layer
  - รับผิดชอบในการส่งข้อมูลผ่าน Network, หาทิศทางข้อมูล, เชื่อมต่อกับ Layer บนเข้ากับ Network หลายๆแบบ มองเห็นข้อมูลในลักษณะ Packet

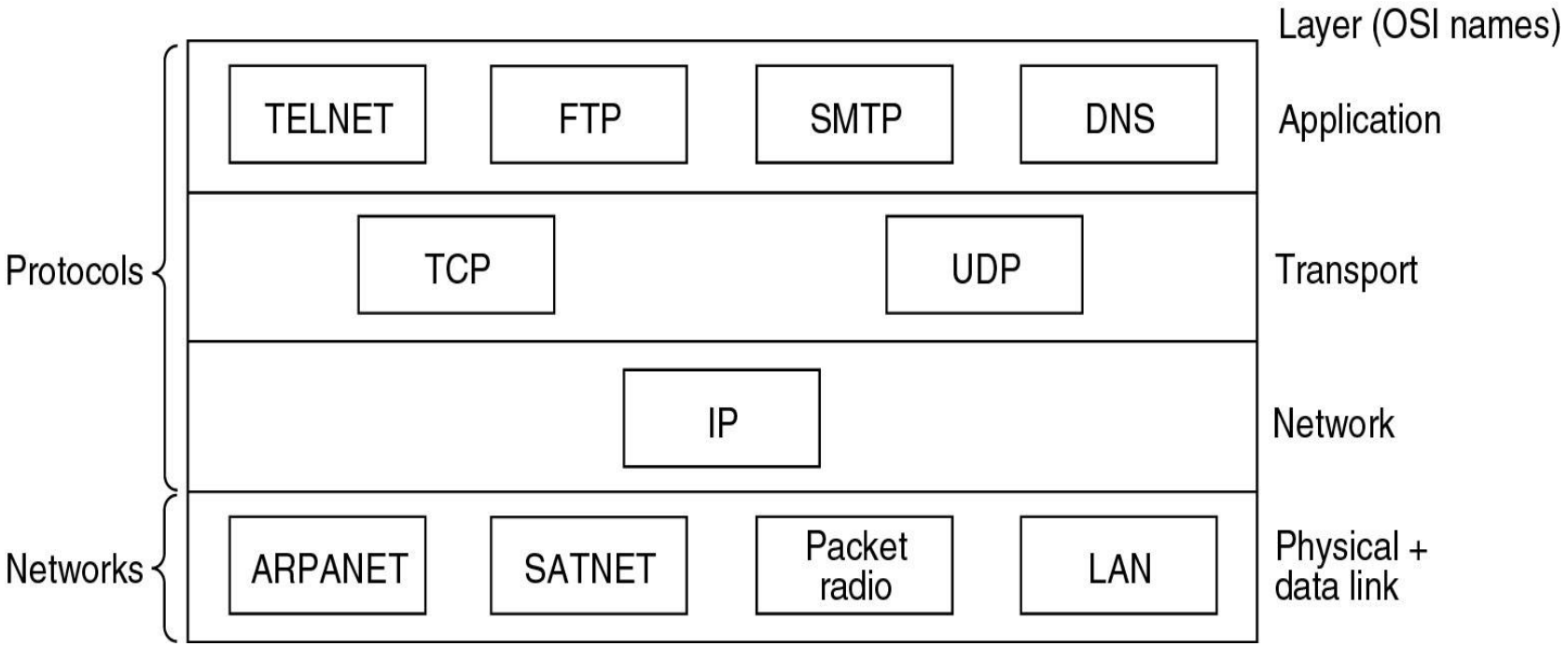
# 7 Layer

- Layer 4: Transport Layer
  - รับผิดชอบการส่งข้อมูลให้ถูกต้องจากต้นทางถึงปลายทาง(End-to-End), จัดการในเรื่อง Error และ Flow Control ในระดับต้นทางถึงปลายทาง ข้อมูลที่ส่งจะถูกแบ่งเป็น Segment
- Layer 5: Session Layer
  - ทำหน้าที่จัดตั้ง ดูแล การเชื่อมต่อ(Connection) ระหว่าง Applicationต้นทางและปลายทาง แบ่งการเชื่อมต่อสื่อสารออกเป็น Session
- Layer 6: Presentation Layer
  - รับผิดชอบในเรื่องรูปแบบและ Format ของข้อมูล การทำ Encryption รวมถึงการทำ Data Compression ให้อยู่ในรูปแบบที่สื่อสารได้
- Layer 7: Application Layer
  - ทำหน้าที่เชื่อมต่อกับ Application และผู้ใช้

# The TCP/IP Reference Model



# The TCP/IP Reference Model

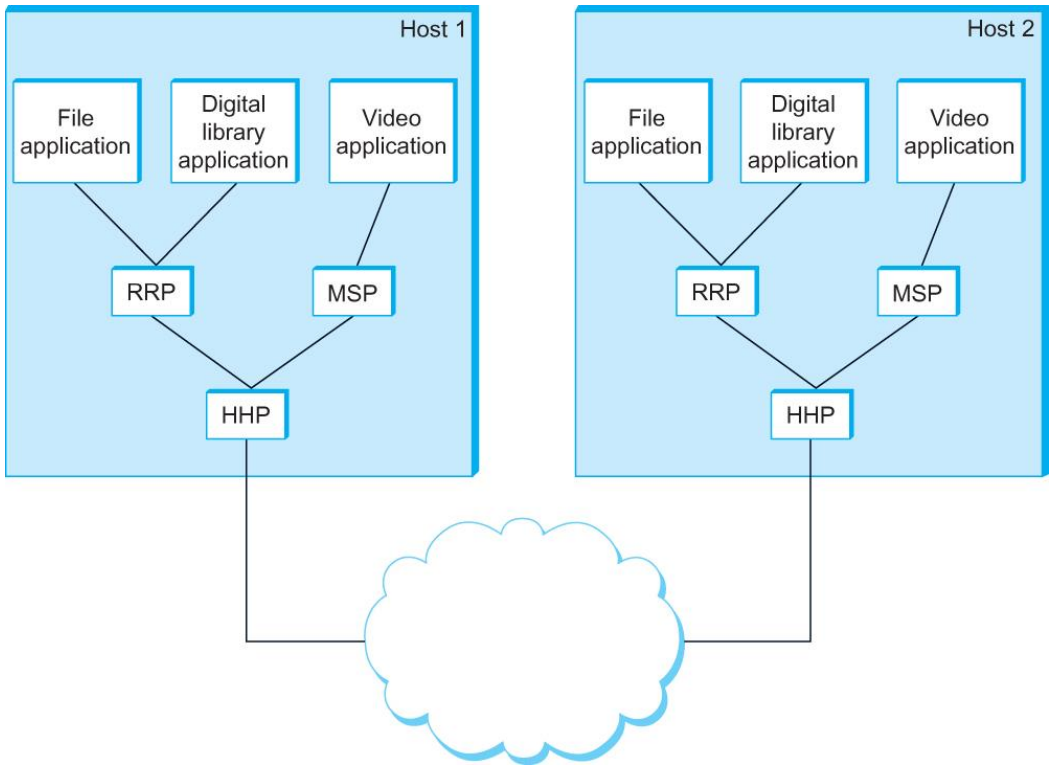



Protocols and networks in the TCP/IP model initially.

# Application Protocol

- URL
  - Uniform resource locater
  - http://www.cs.princeton.edu/~llp/index.html
- HTTP
  - Hyper Text Transfer Protocol
- TCP
  - Transmission Control Protocol
- 17 messages for one URL request
  - 6 to find the IP (Internet Protocol) address
  - 3 for connection establishment of TCP
  - 4 for HTTP request and acknowledgement
    - Request: I got your request and I will send the data
    - Reply: Here is the data you requested; I got the data
  - 4 messages for tearing down TCP connection

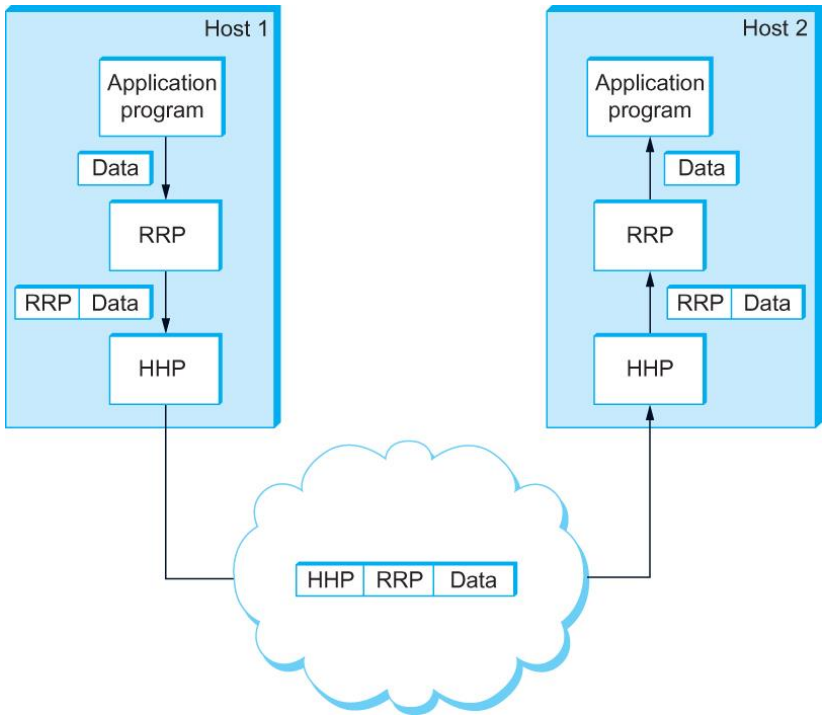
# Protocol Graph




Example of a protocol graph

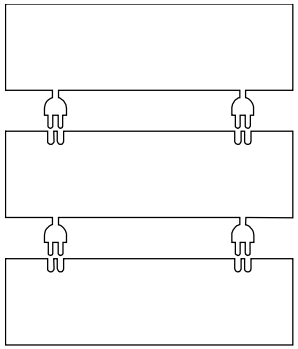
nodes are the protocols and links the “depends-on” relation

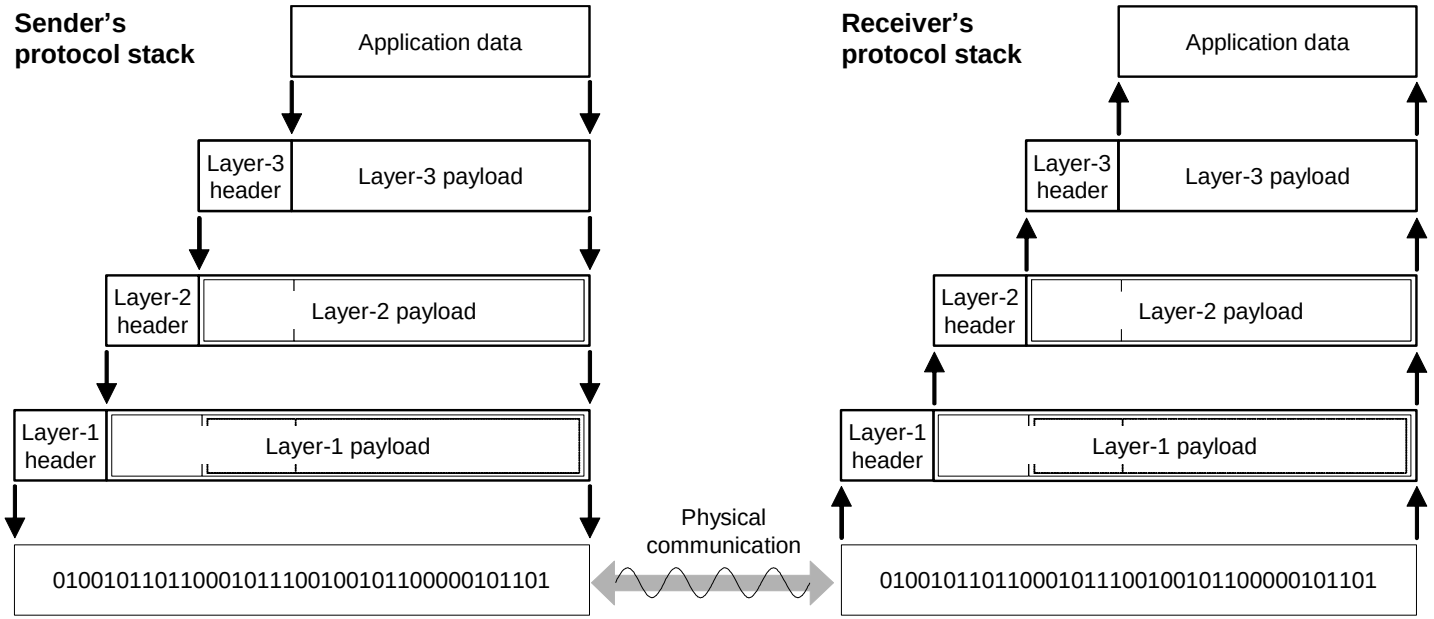
# Encapsulation



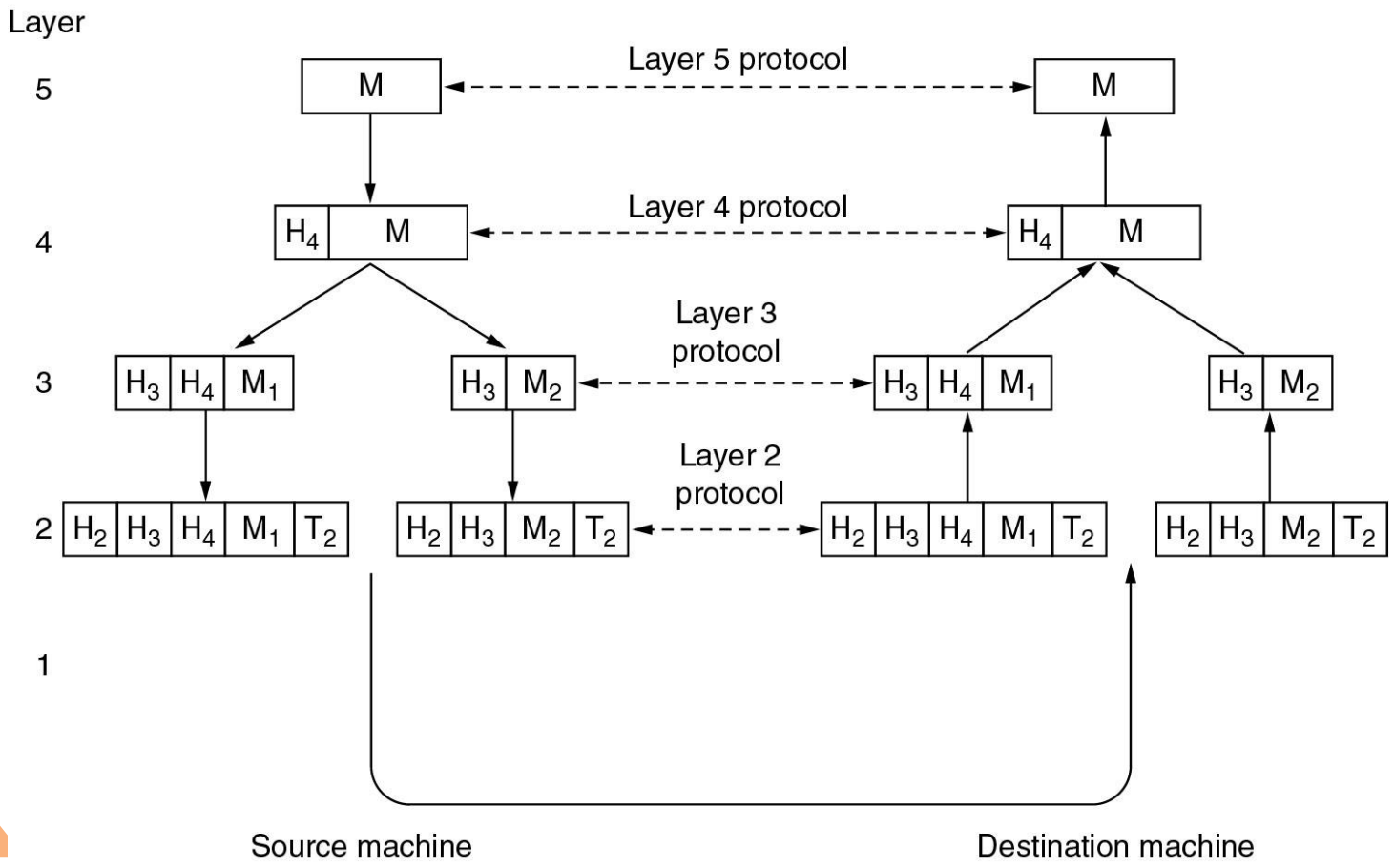

High-level messages are encapsulated inside of low-level messages

# Packet Nesting Across Layers

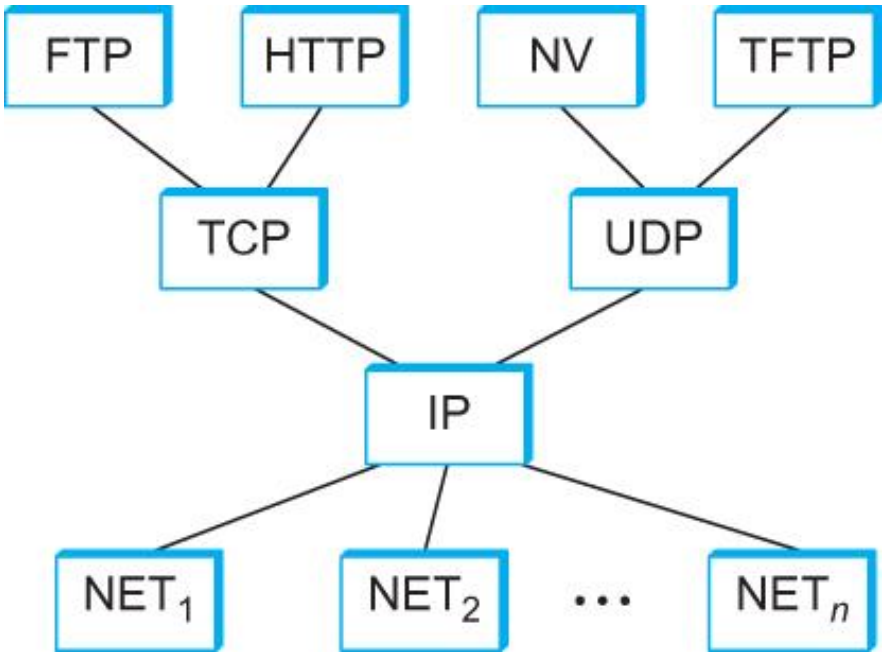




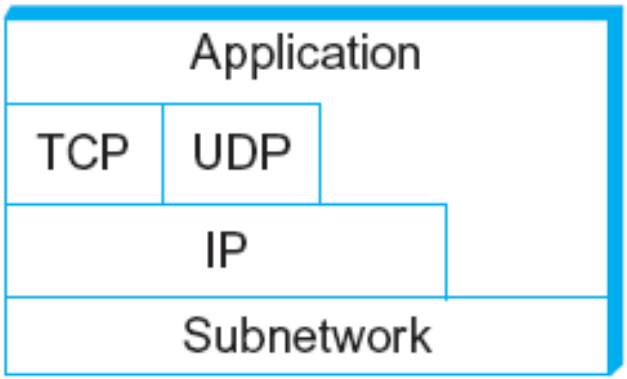
# Encapsulation



# Internet Architecture




Internet Protocol Graph




Alternative view of the Internet architecture. The "Network" layer shown here is sometimes referred to as the "sub-network" or "link" layer.

# Internet Architecture

- Defined by IETF (Internet Engineering Task Force)
- Three main features
  - Does not imply strict layering. The application is free to bypass the defined transport layers and to directly use IP or other underlying networks
  - An hour-glass shape – wide at the top, narrow in the middle and wide at the bottom. IP serves as the focal point for the architecture
  - In order for a new protocol to be officially included in the architecture, there needs to be both a protocol specification and at least one (and preferably two) representative implementations of the specification

# Connection-Oriented vs. Connectionless

| | Service | Example |
|---|---|---|
| Connection-oriented | Reliable message stream | Sequence of pages |
| | Reliable byte stream | Remote login |
| | Unreliable connection | Digitized voice |
| Connectionless | Unreliable datagram | Electronic junk mail |
| | Acknowledged datagram | Registered mail |
| | Request-reply | Database query |

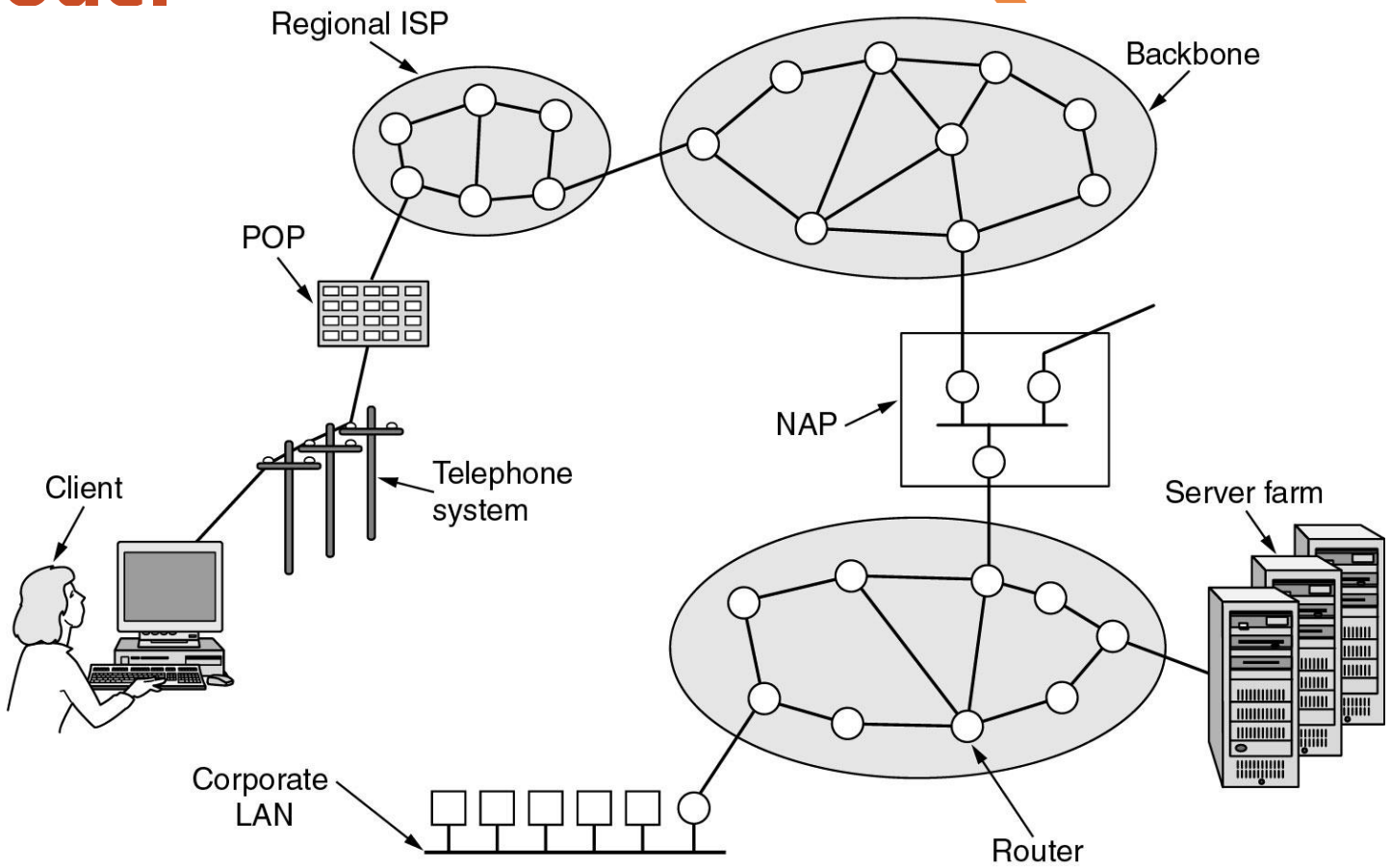
# Service Primitives

A service is formally specified by a set of primitives (basic operations) available to a user or other entity to access the service.

| Primitive | Meaning |
|---|---|
| LISTEN | Block waiting for an incoming connection |
| CONNECT | Establish a connection with a waiting peer |
| RECEIVE | Block waiting for an incoming message |
| SEND | Send a message to the peer |
| DISCONNECT | Terminate a connection |

Example: five service primitives for implementing a simple connection-oriented service.

# The TCP/IP Reference Model




Overview of the Internet.

# Application Programming Interface

- Interface exported by the network
- Since most network protocols are implemented (those in the high protocol stack) in software and nearly all computer systems implement their network protocols as part of the operating system, when we refer to the interface "*exported by the network*", we are generally referring to the interface that the OS provides to its networking subsystem
- The interface is called the network Application Programming Interface (API)

# Application Programming Interface (Sockets)

- Socket Interface was originally provided by the Berkeley distribution of Unix
  - Now supported in virtually all operating systems

- Each protocol provides a certain set of *services*, and the API provides a syntax by which those services can be invoked in this particular OS

# Socket

- What is a socket?
  - The point where a local application process attaches to the network
  - An interface between an application and the network
  - An application creates the socket
- The interface defines operations for
  - Creating a socket
  - Attaching a socket to the network
  - Sending and receiving messages through the socket
  - Closing the socket

# Socket

- Socket Family
  - PF_INET denotes the Internet family
  - PF_UNIX denotes the Unix pipe facility
  - PF_PACKET denotes direct access to the network interface (i.e., it bypasses the TCP/IP protocol stack)
- Socket Type
  - SOCK_STREAM is used to denote a byte stream
  - SOCK_DGRAM is an alternative that denotes a message oriented service, such as that provided by UDP

# Creating a Socket

```
int sockfd = socket(address_family, type, protocol);
```

- The socket number returned is the socket descriptor for the newly created socket

- `int sockfd = socket (PF_INET, SOCK_STREAM, 0);`
- `int sockfd = socket (PF_INET, SOCK_DGRAM, 0);`

The combination of PF_INET and SOCK_STREAM implies TCP

# Client-Server Model with TCP

Server

- Passive open
- Prepares to accept connection, does not actually establish a connection

Server invokes

```
int bind (int socket, struct sockaddr *address,
                                    int addr_len)
int listen (int socket, int backlog)
int accept (int socket, struct sockaddr *address,
                                int *addr_len)
```

# Client-Server Model with TCP

## Bind

- Binds the newly created socket to the specified address i.e. the network address of the local participant (the server)
- Address is a data structure which combines IP and port

## Listen

- Defines how many connections can be pending on the specified socket

# Client-Server Model with TCP

Accept

- Carries out the passive open
- Blocking operation
  - Does not return until a remote participant has established a connection
  - When it does, it returns a new socket that corresponds to the new established connection and the address argument contains the remote participant's address

# Client-Server Model with TCP

Client

- Application performs active open
- It says who it wants to communicate with

Client invokes

```
int connect (int socket, struct sockaddr *address,int addr_len)
```

Connect

- Does not return until TCP has successfully established a connection at which application is free to begin sending data
- Address contains remote machine's address

# Client-Server Model with TCP

In practice

- The client usually specifies only remote participant's address and let's the system fill in the local information
- Whereas a server usually listens for messages on a well-known port
- A client does not care which port it uses for itself, the OS simply selects an unused one

# Client-Server Model with TCP

Once a connection is established, the application process invokes two operation

```
int send (int socket, char *msg, int msg_len, int flags)
```

```
int recv (int socket, char *buff, int buff_len, int flags)
```

# Example Application: Client

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

#define SERVER_PORT 5432
#define MAX_LINE 256

int main(int argc, char * argv[])
{
    FILE *fp;
    struct hostent *hp;
    struct sockaddr_in sin;
    char *host;
    char buf[MAX_LINE];
    int s;
    int len;
    if (argc==2) {
        host = argv[1];
    }
    else {
        fprintf(stderr, "usage: simplex-talk host\n");
    exit(1);
    }
```

# Example Application: Client

```
    /* translate host name into peer's IP address */
    hp = gethostbyname(host);
    if (!hp) {
    host)fprintf(stderr, "simplex-talk: unknown host: %s\n",
        exit(1);
    }
    /* build address data structure */
    bzero((char *)&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
    sin.sin_port = htons(SERVER_PORT);
    /* active open */
    if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("simplex-talk: socket");
        exit(1);
    }
    if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
        perror("simplex-talk: connect");
        close(s);
        exit(1);
    }
    /* main loop: get and send lines of text */
    while (fgets(buf, sizeof(buf), stdin)) {
        buf[MAX_LINE-1] = '\0';
        len = strlen(buf) + 1;
        send(s, buf, len, 0);
    }
}
```

# Example Application: Server

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 5432
#define MAX_PENDING 5
#define MAX_LINE 256

int main()
{
    struct sockaddr_in sin;
    char buf[MAX_LINE];
    int len;
    int s, new_s;
    /* build address data structure */
    bzero((char *)&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons(SERVER_PORT);

    /* setup passive open */
    if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("simplex-talk: socket");
        exit(1);
    }
```

# Example Application: Server

```
    if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
        perror("simplex-talk: bind");
        exit(1);
    }
    listen(s, MAX_PENDING);
    /* wait for connection, then receive and print text */
    while(1) {
        if ((new_s = accept(s, (struct sockaddr *)&sin, &len)) < 0) {
        perror("simplex-talk: accept");
        exit(1);
    }
    while (len = recv(new_s, buf, sizeof(buf), 0))
        fputs(buf, stdout);
        close(new_s);
    }
}
```

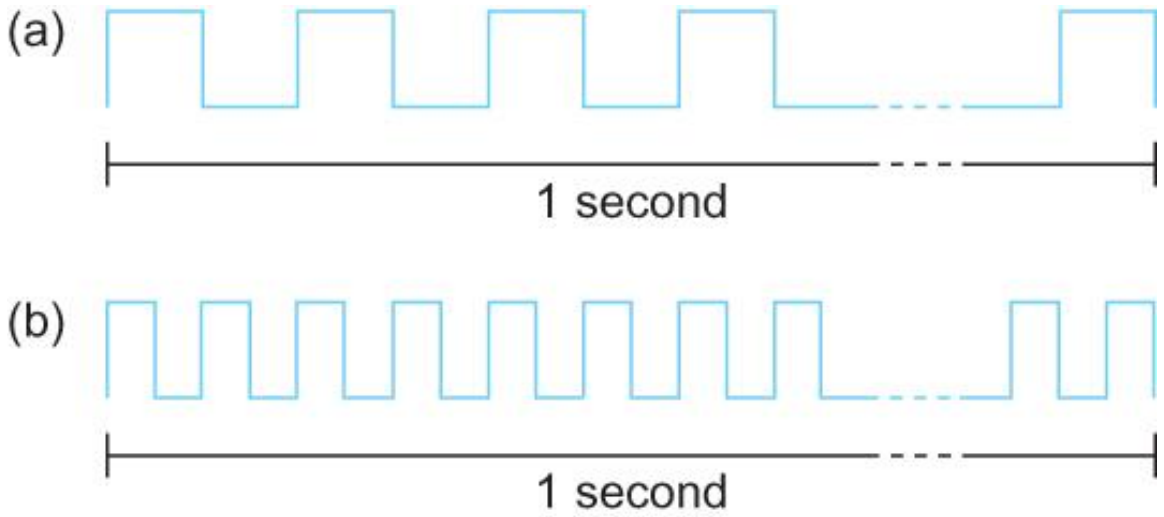
# Homework

- A group of 2
- Try the client and server socket communication code
- Transmit mathematics problem and calculate the solution back and capture the output

# Performance

- Bandwidth
  - Width of the frequency band
  - Number of bits per second that can be transmitted over a communication link
- 1 Mbps: $1 \times 10^6$ bits/second = $1x2^{20}$ bits/sec
- $1 \times 10^{-6}$ seconds to transmit each bit or imagine that a timeline, now each bit occupies 1 micro second space.
- On a 2 Mbps link the width is 0.5 micro second.
- Smaller the width more will be transmission per unit time.

# Bandwidth

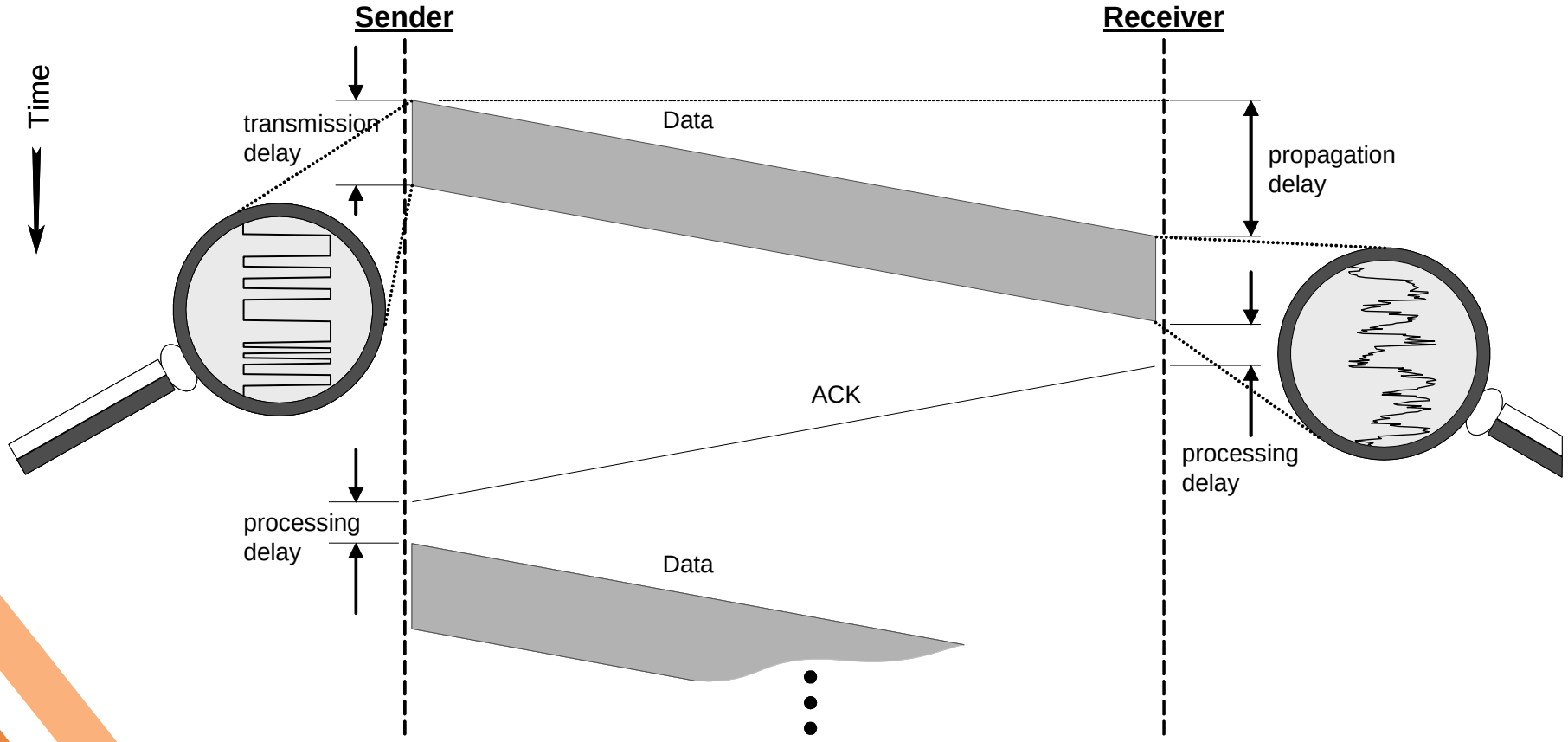



Bits transmitted at a particular bandwidth can be regarded as having some width:

(a) bits transmitted at 1Mbps (each bit 1 μs wide);

(b) bits transmitted at 2Mbps (each bit 0.5 μs wide).

# Packet Transmission (2)



# Performance

- $Latency = Propagation + Transmit + queue$
- $Propagation = distance/speed\ of\ light$
- $Transmit = size/bandwidth$

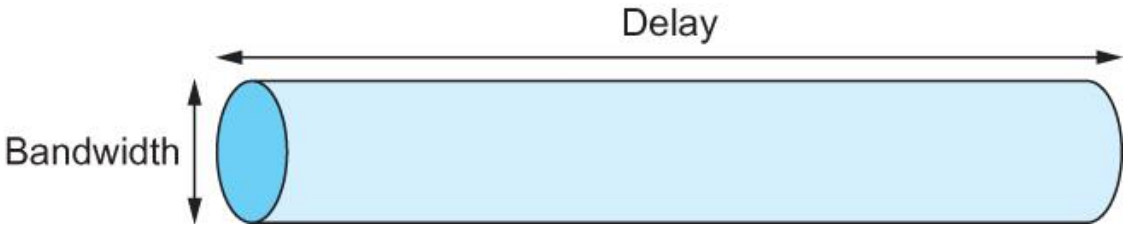
- One bit transmission => propagation is important
- Large bytes transmission => bandwidth is important

# Delay X Bandwidth

- Let's imagine that the channel between a pair of processes is a hollow pipe
- Latency (delay) length of the pipe and bandwidth the width of the pipe
- Delay of 50 ms and bandwidth of 45 Mbps

$\Rightarrow$ 50 x $10^{-3}$ seconds x 45 x $10^{6}$ bits/second

$\Rightarrow$ 2.25 x $10^{6}$ bits = 280 KB data.




Network as a pipe

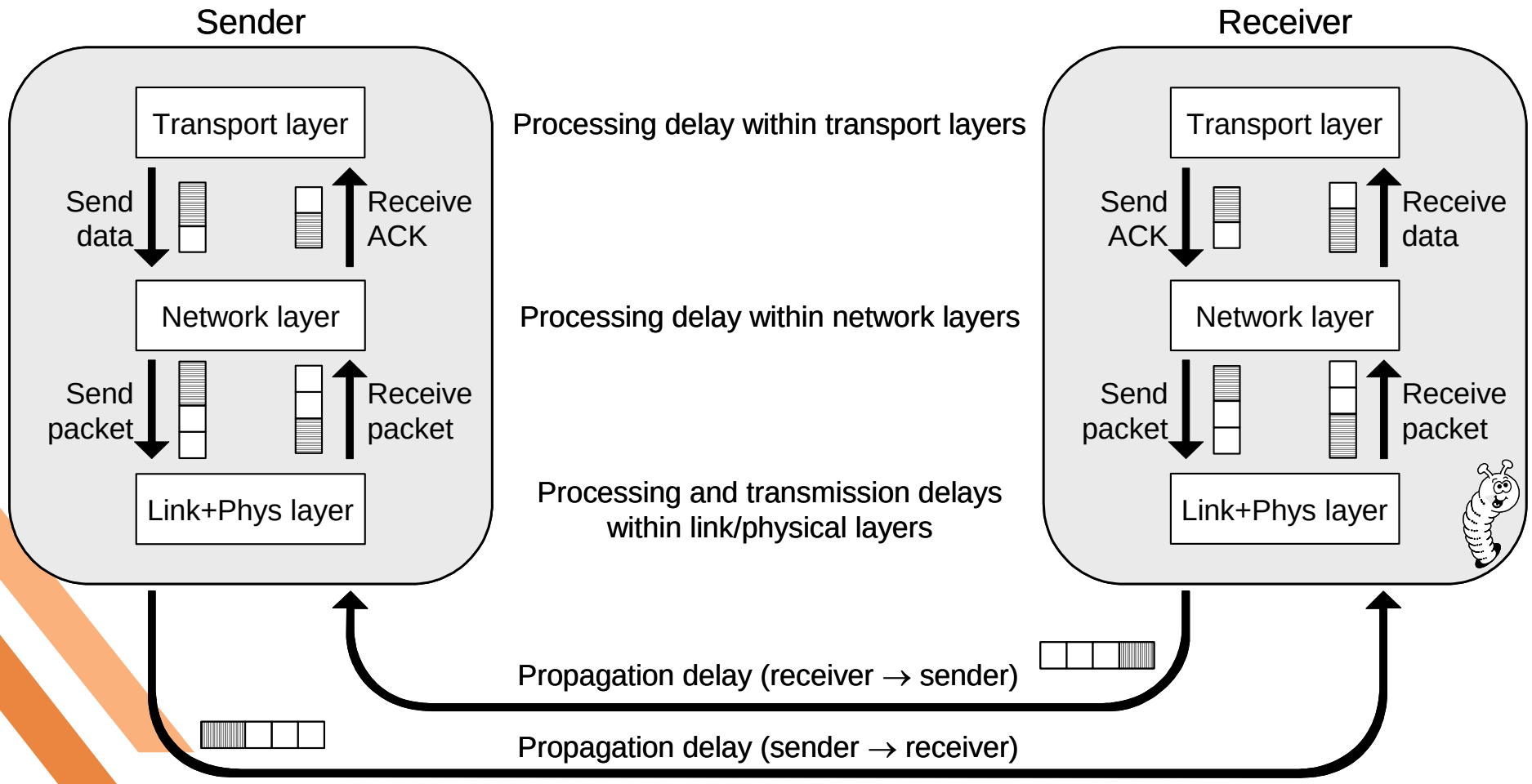
# Delay X Bandwidth

- Relative importance of bandwidth and latency depends on application
  - For large file transfer, bandwidth is critical
  - For small messages (HTTP, NFS, etc.), latency is critical
  - Variance in latency (jitter) can also affect some applications (*e.g.*, audio/video conferencing)

# Delay X Bandwidth

- How many bits the sender must transmit before the first bit arrives at the receiver if the sender keeps the pipe full
- Takes another one-way latency to receive a response from the receiver
- If the sender does not fill the pipe—send a whole delay × bandwidth product's worth of data before it stops to wait for a signal—the sender will not fully utilize the network

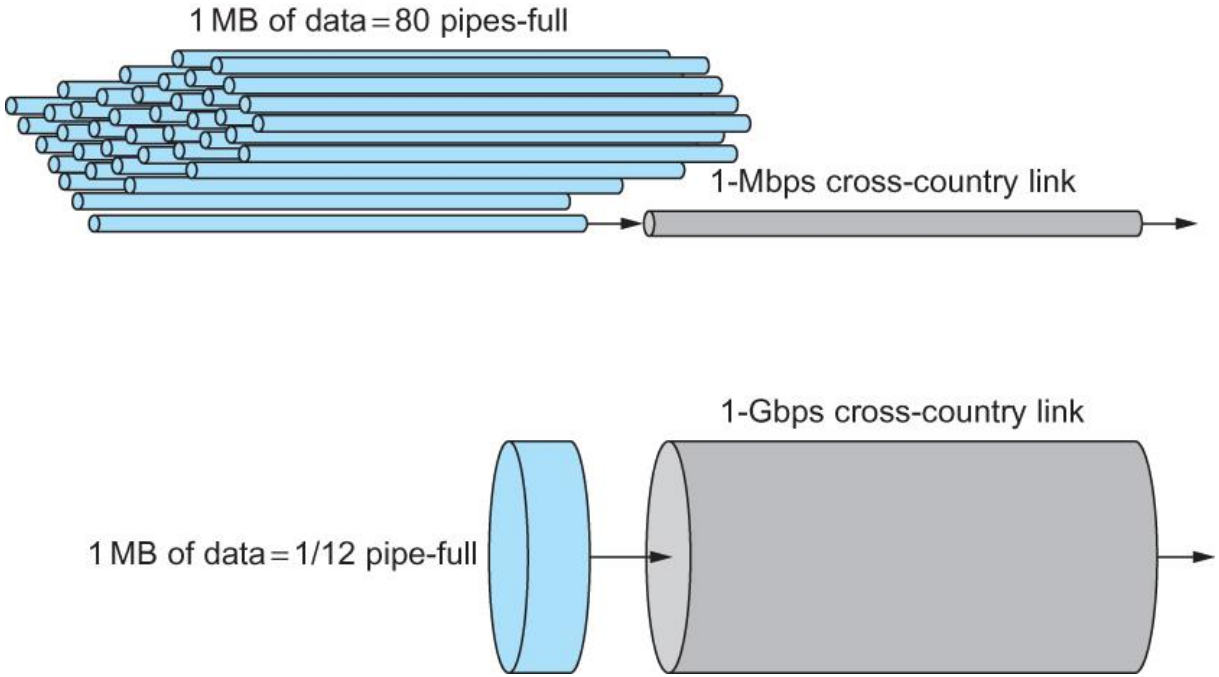
# What Contributes to RTT



# Delay X Bandwidth

- Infinite bandwidth
  - RTT (Round Trip Time) dominates
  - $Throughput = TransferSize / TransferTime$
  - $TransferTime = RTT + 1/Bandwidth \; x \; TransferSize$
- Its all relative
  - 1-MB file to 1-Gbps link looks like a 1-KB packet to 1-Mbps link

# Relationship between bandwidth and latency




A 1-MB file would fill the 1-Mbps link 80 times, but only fill the 1-Gbps link 1/12 of one time

# Summary

- We have identified what we expect from a computer network
- We have defined a layered architecture for computer network that will serve as a blueprint for our design
- We have discussed the socket interface which will be used by applications for invoking the services of the network subsystem
- We have discussed two performance metrics using which we can analyze the performance of computer networks