

## Chapter 1

### Foundation

# 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 Outline

---

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

# 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

# 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
  - ...



# Application Protocol

- URL
  - Uniform resource locator
  - <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

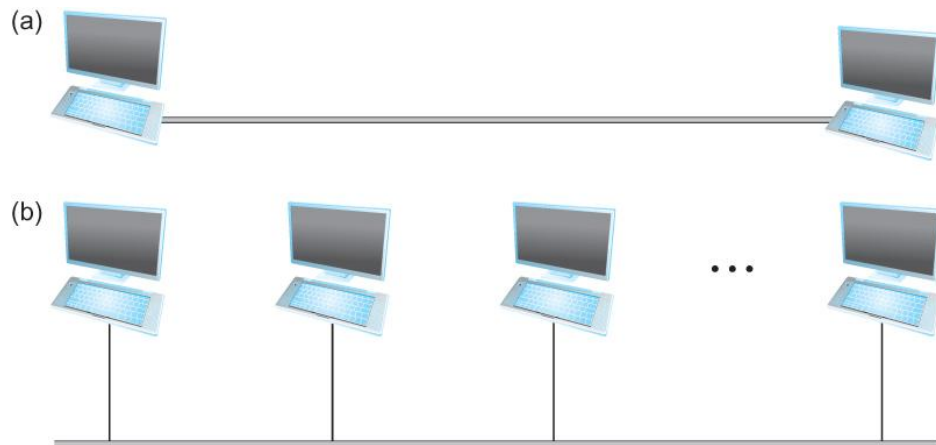
# Requirements

---

- 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

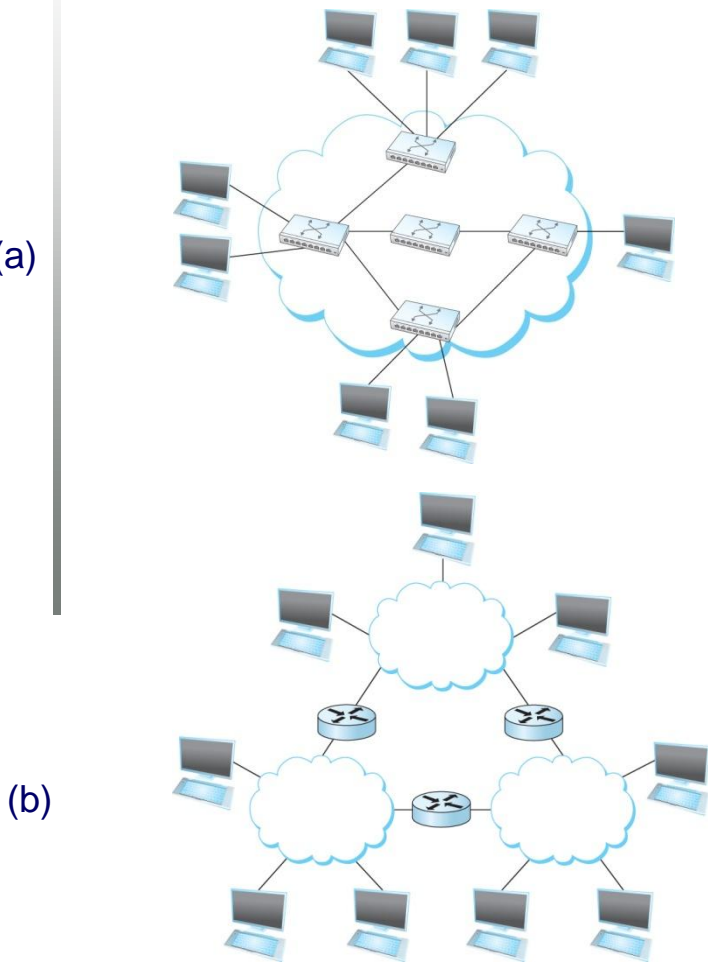


- Need to understand the following terminologies
  - Scale
  - Link
  - Nodes
  - Point-to-point
  - Multiple access
  - Switched Network
    - Circuit Switched
    - Packet Switched
  - Packet, message
  - Store-and-forward

- (a) Point-to-point
- (b) Multiple access

# Connectivity

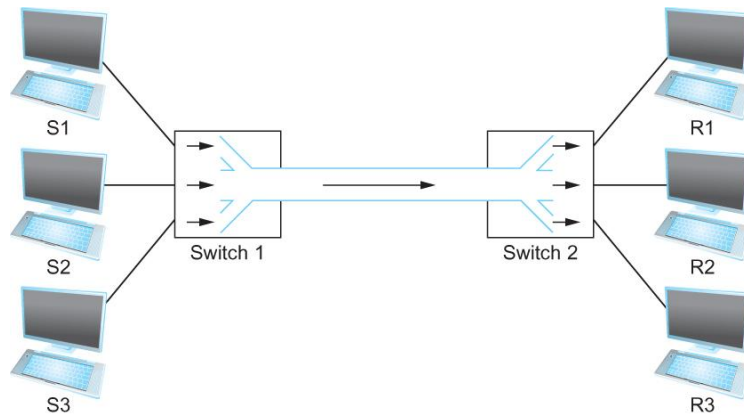
- Terminologies (contd.)
  - Cloud
  - Hosts
  - Switches
  - internetwork
  - Router/gateway
  - Host-to-host connectivity
  - Address
  - Routing
  - Unicast/broadcast/multicast



(a) A switched network

(b) Interconnection of networks

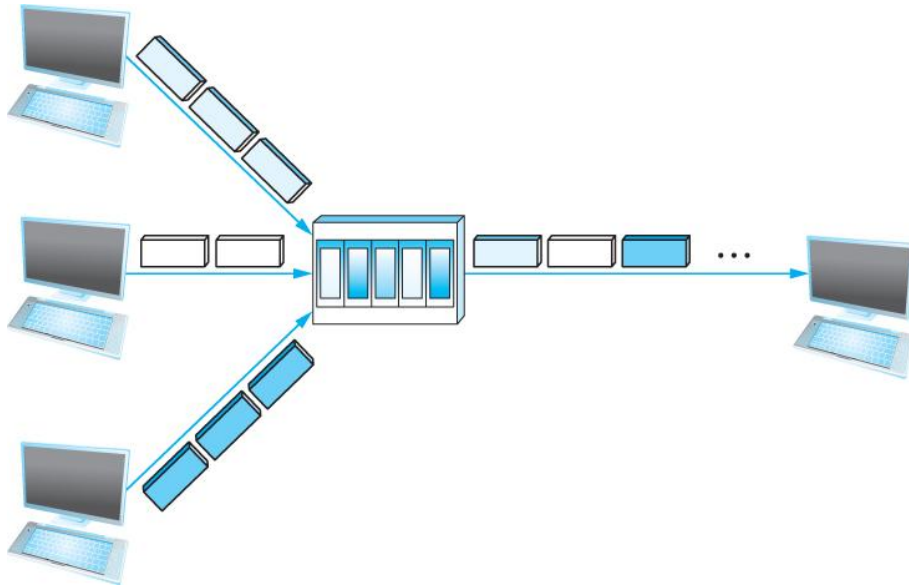
# Cost-Effective Resource Sharing



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

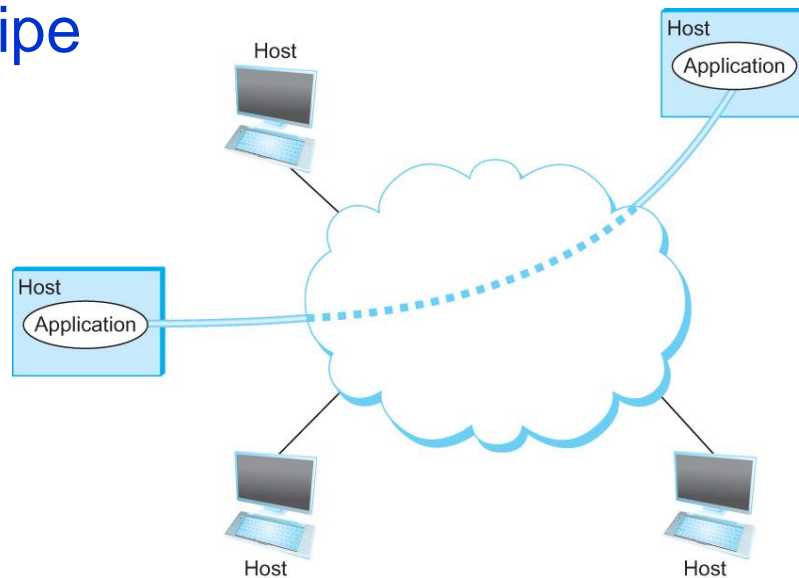


A switch multiplexing packets from multiple sources onto one shared link

- 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)

# Support for Common Services

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



Process communicating over an abstract channel

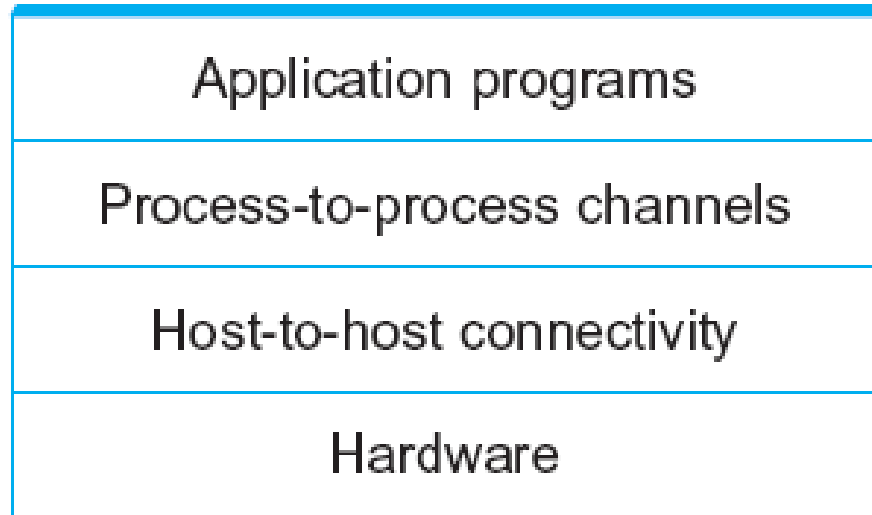
# Common Communication Patterns

- 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

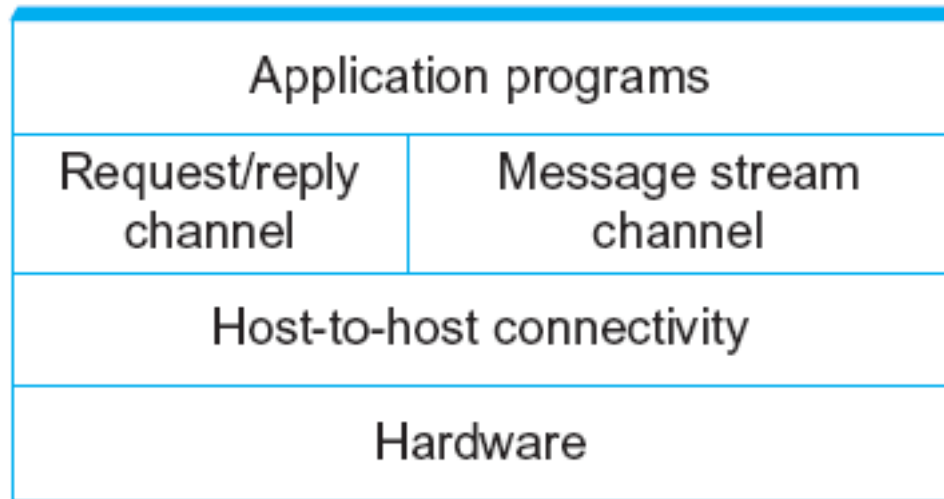
# Network Architecture



Example of a layered network system



# Network Architecture

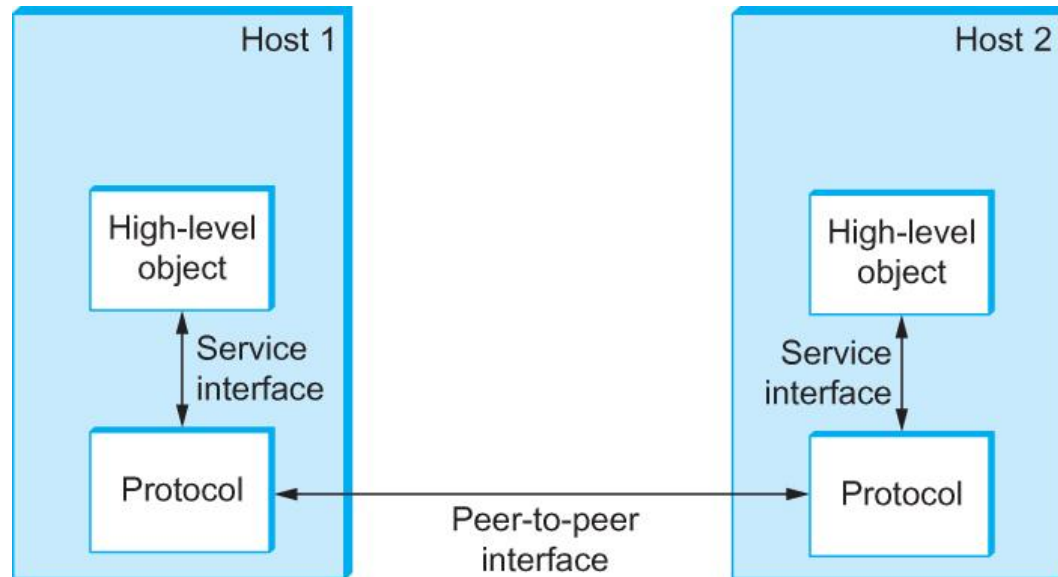


Layered system with alternative abstractions available at a given layer

# Protocols

- 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

# Interfaces



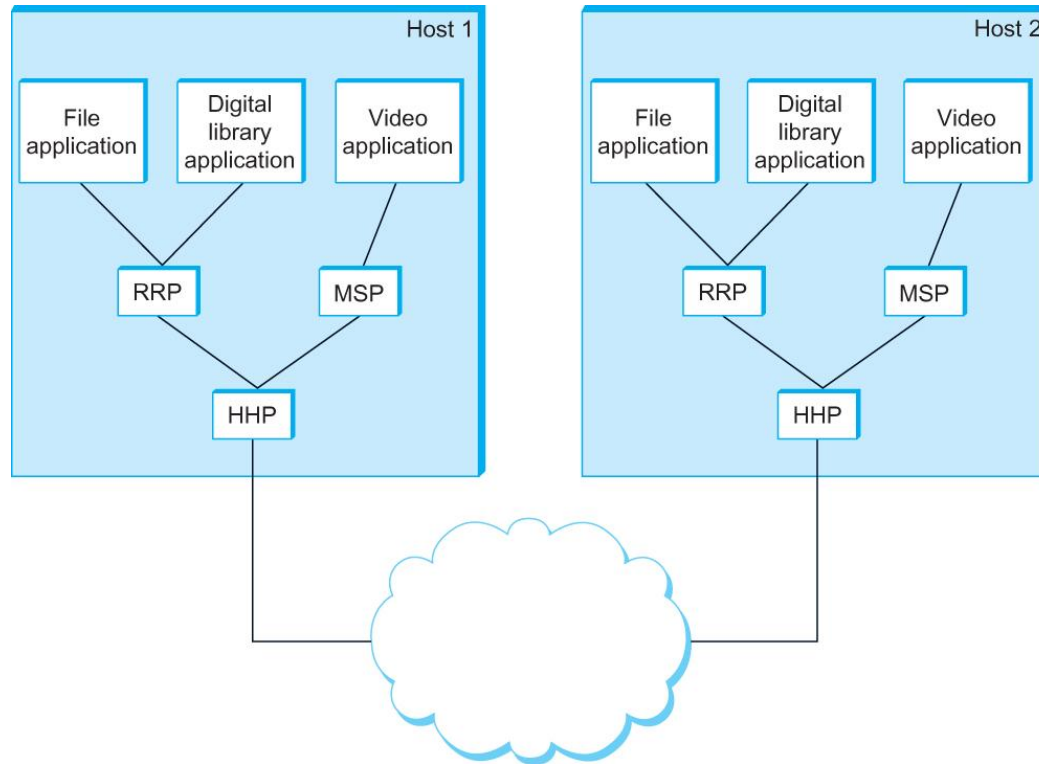
Service and Peer Interfaces

# Protocols

---

- Protocol Specification: prose, pseudo-code, state transition diagram
- Interoperable: when two or more protocols that implement the specification accurately
- IETF: Internet Engineering Task Force

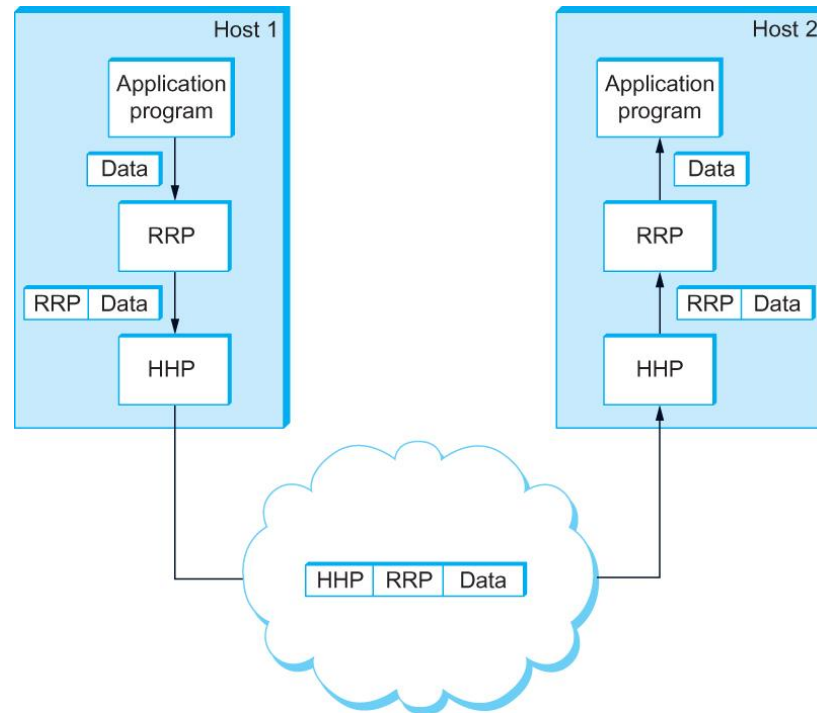
# Protocol Graph



Example of a protocol graph

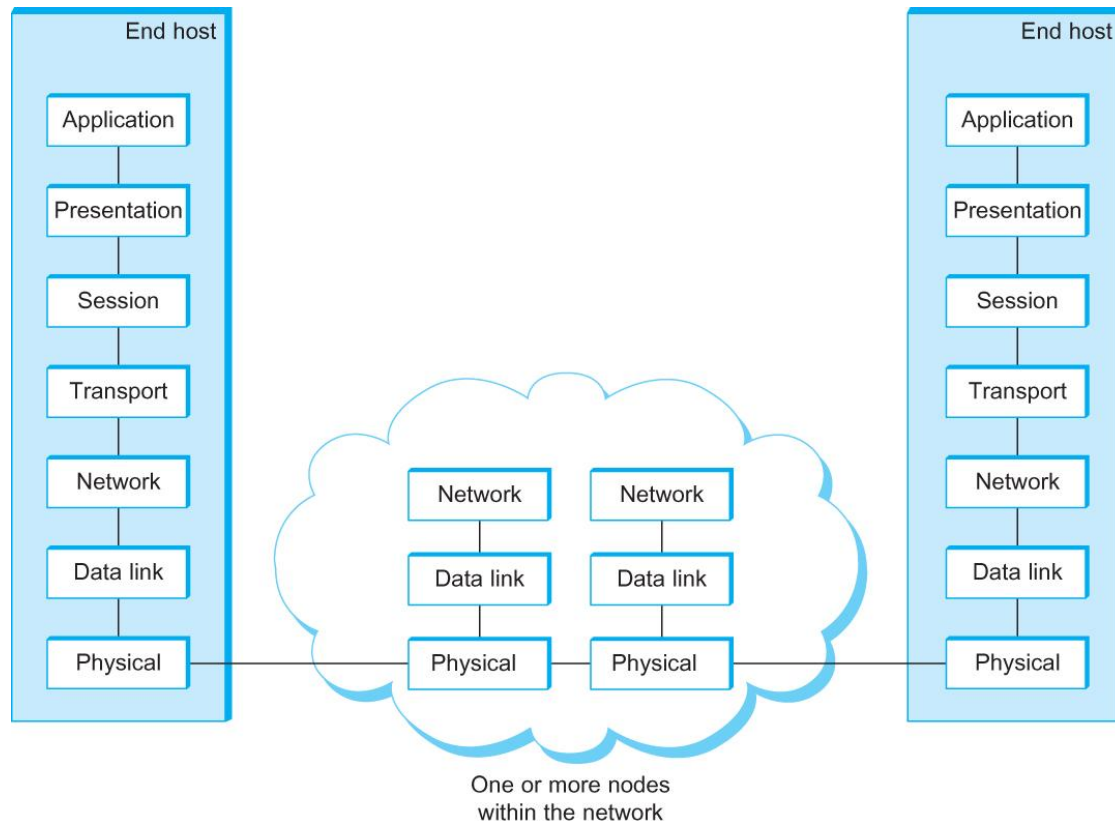
nodes are the protocols and links the “depends-on” relation

# Encapsulation



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

# OSI Architecture



## The OSI 7-layer Model

### OSI – Open Systems Interconnection

# 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

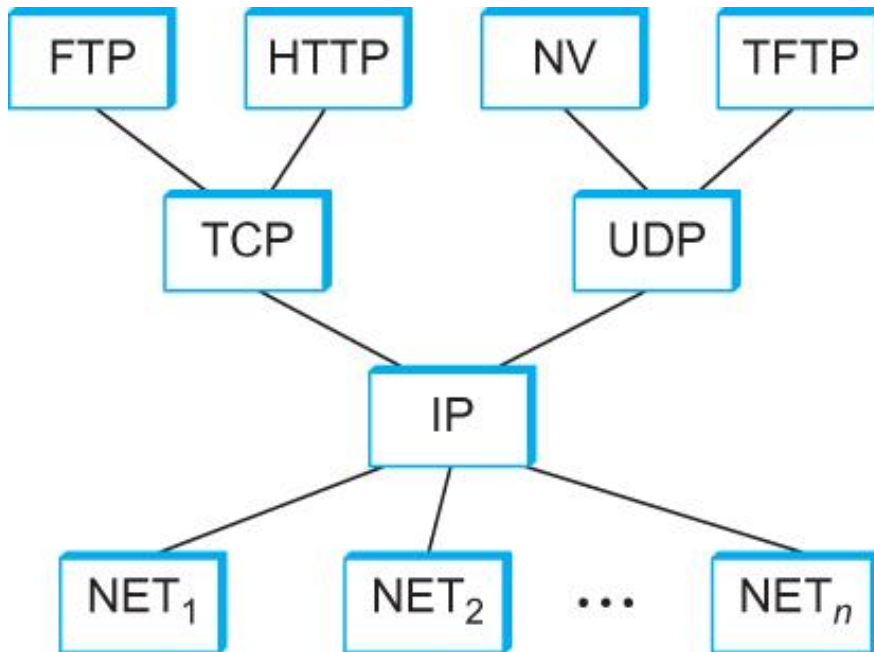


# Description of Layers

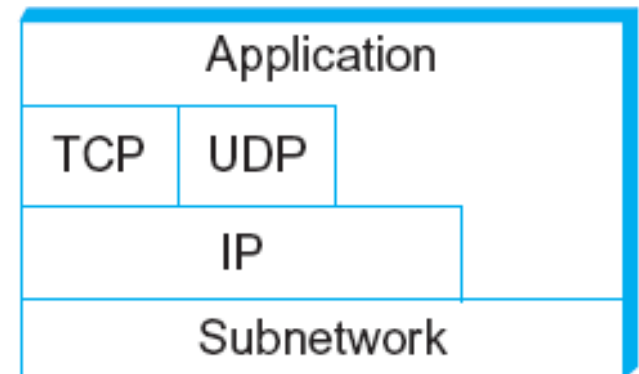
- Transport Layer
  - Implements a process-to-process channel
  - Unit of data exchanges in this layer is called a *message*
- Session Layer
  - Provides a name space that is used to tie together the potentially different transport streams that are part of a single application
- Presentation Layer
  - 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

# 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
- 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

# 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-Serve 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-Serve 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-Serve 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-Serve 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-Serve 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-Serve 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) {
    fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
    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);
    }
}
```

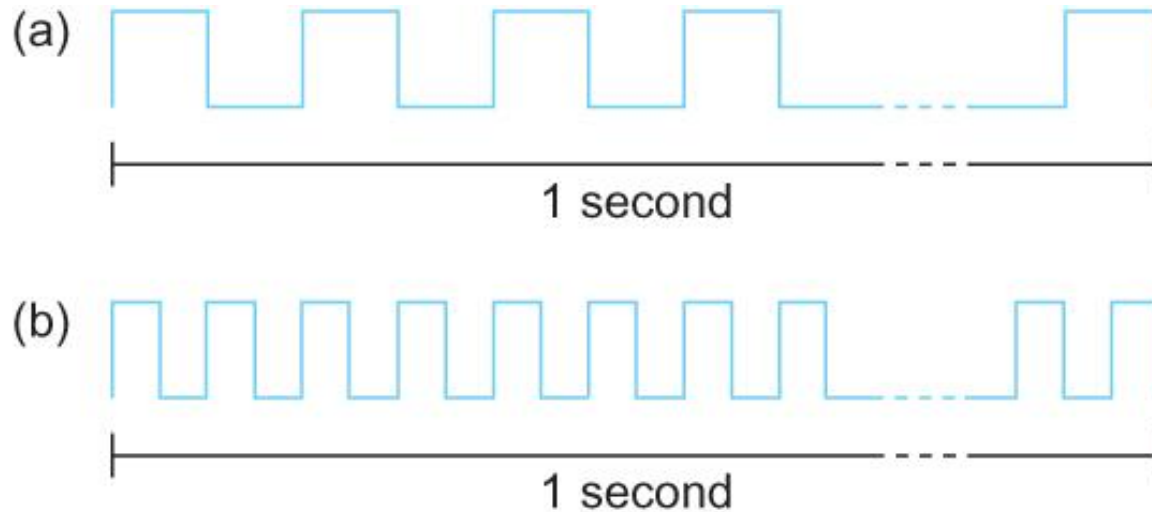
# Example Application: Server

```
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);
}
}
```

# 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 =  $1 \times 2^{20}$  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  $1 \mu\text{s}$  wide);
- (b) bits transmitted at 2Mbps (each bit  $0.5 \mu\text{s}$  wide).

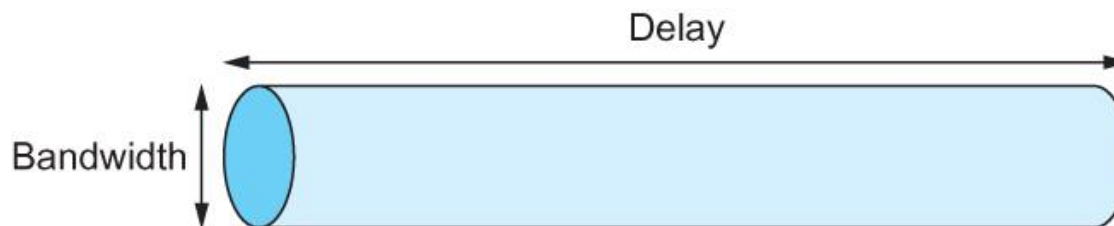
# Performance

---

- 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

- We think the channel between a pair of processes as 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
  - ⇒  $50 \times 10^{-3}$  seconds  $\times$   $45 \times 10^6$  bits/second
  - ⇒  $2.25 \times 10^6$  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

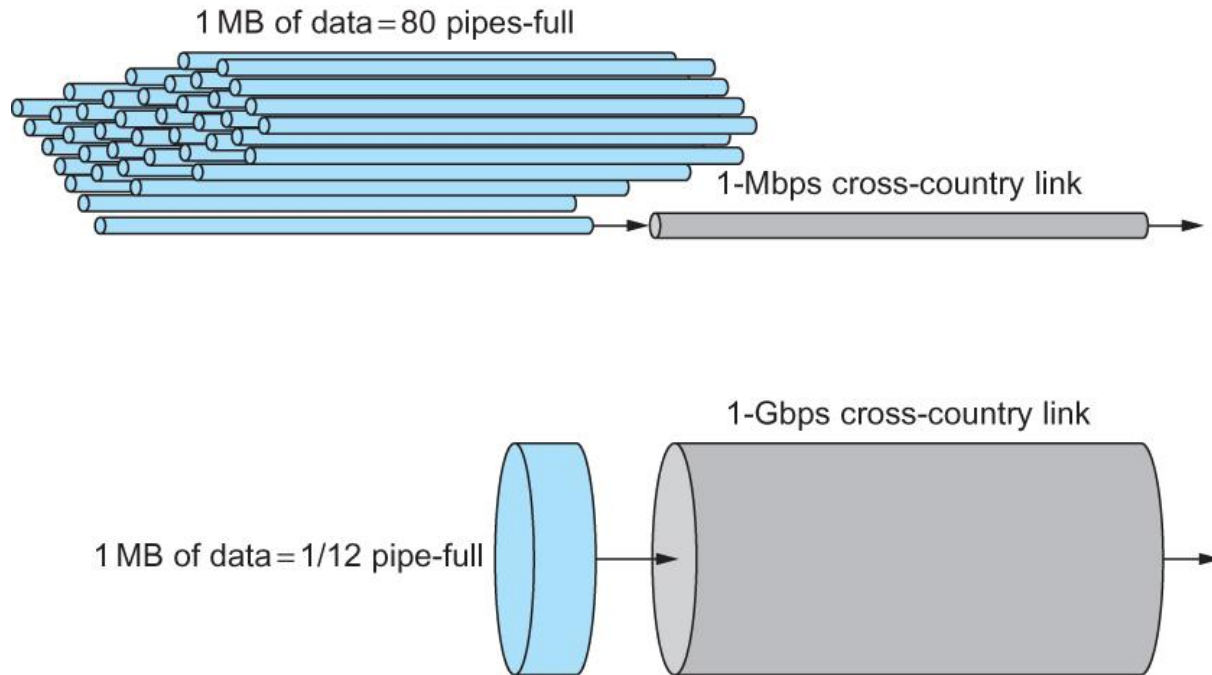
- How many bits the sender must transmit before the first bit arrives at the receiver if the sender keeps the pipe full
- Takes another one-way latency to receive a response from the receiver
- If the sender does not fill the pipe—send a whole delay  $\times$  bandwidth product's worth of data before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



# Delay X Bandwidth

- Infinite bandwidth
  - RTT dominates
  - $\text{Throughput} = \text{TransferSize} / \text{TransferTime}$
  - $\text{TransferTime} = \text{RTT} + 1/\text{Bandwidth} \times \text{TransferSize}$
- 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