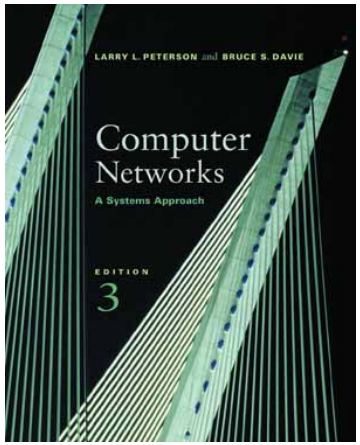
Part I – Networking

1. Foundations of Networking
2. Sliding Window, Ethernet and WiFi
3. Internetworking (IP, Routing)
4. End-to-end Protocols: TCP and UDP

Textbooks for Part I

- Main textbook:



  Larry L. Peterson, Bruce S. Davie: Computer Networks – A Systems Approach, 3rd Edition, Morgan Kaufmann, 2003. ISBN 1-55860-833-8.

- Further reading:
  - Andrew S. Tanenbaum: Computer Networks, 4th Edition, Pearson, 2003, ISBN 0-13-038488-7.
  - William Stallings: Data and Computer Communications, 7th Edition, Pearson, 2004, ISBN 0-13-183311-1.

1. Foundations of Networking

Networks

- **Computer Networks:**
  - Built from general-purpose hardware
  - Not optimized for one specific application
  - Able to carry many different types of data and support a wide range of applications: e-mail, WWW, FTP, video-on-demand, electronic commerce, distributed computing, digital libraries, p2p file sharing, ...
- As opposed to special-purpose networks:
  - Voice telephone network
  - Cable network
  - Network connecting terminals to mainframe

Application 1: World Wide Web

- When you enter "http://www.mkp.com/pd3e" into your browser, as many as 17 messages may be exchanged over the Internet:
  - Up to 6 messages to translate www.mkp.com into its IP address (129.35.69.7).
  - 3 messages to set up a TCP connection.
  - 4 messages to request and download the page.
  - 4 messages to tear down the TCP conncection.

Application 2: Audio/Video Streaming

- Instead of downloading the whole audio or video file, "stream" it over the Internet and play it as it arrives.
- Video-on-demand, video conferencing etc.
- For interactive applications such as video conferencing, it is important to have small delays.

Video Application: vic

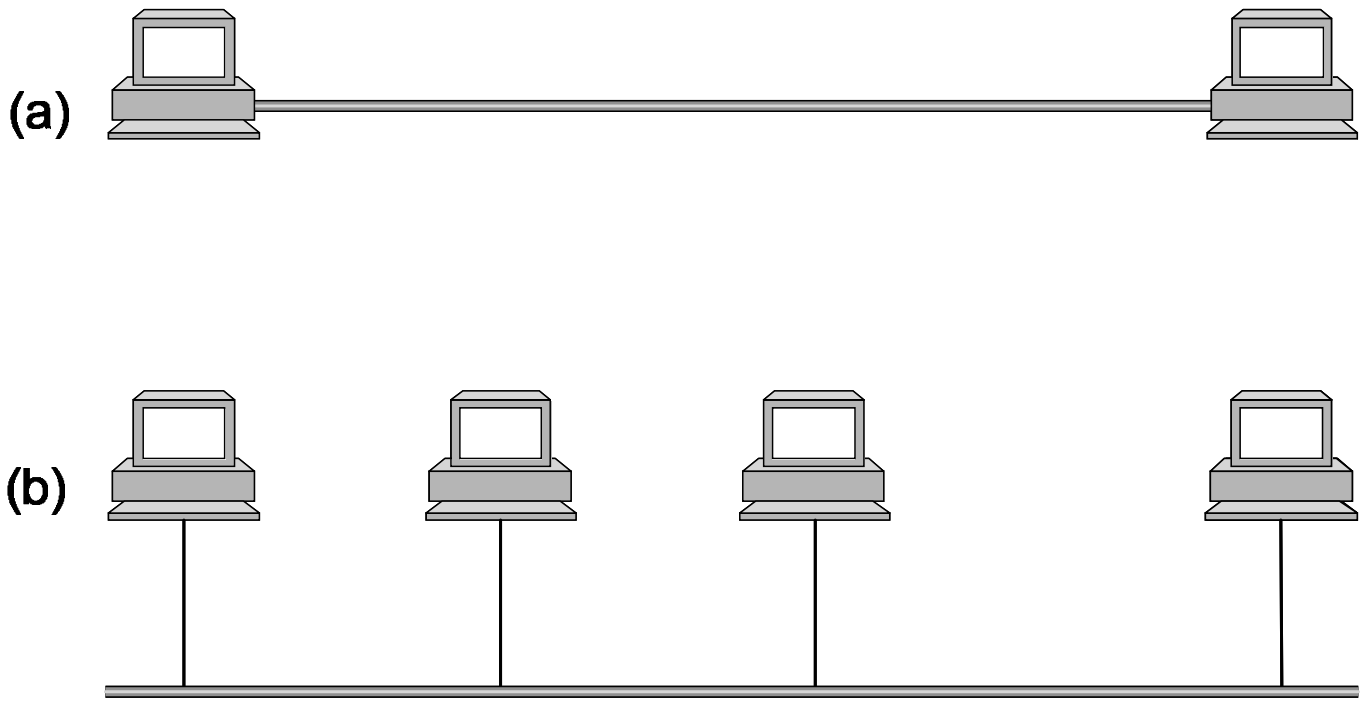
1.2 Requirements

- Application programmer: wants the network to provide the services needed by his or her application, e.g. reliable delivery of messages
- Network designer: wants a cost-effective design, e.g. efficient resource usage and fair bandwidth allocation to users
- Network provider: wants a network that is easy to administer and manage, e.g. easy isolation of faults and easy accounting for usage

1.2.1 Connectivity

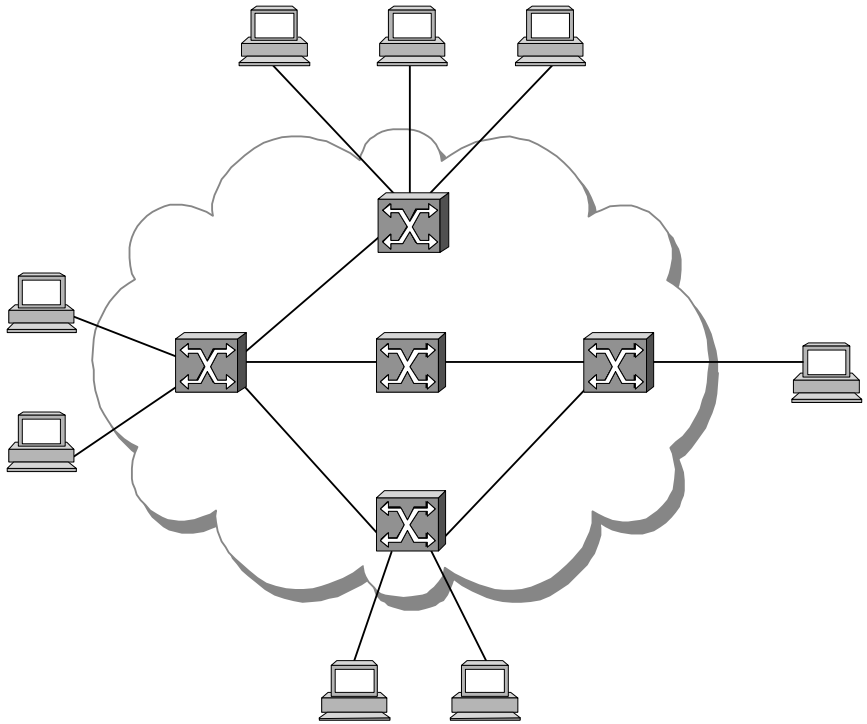
- Each computer (node) on the network should be able to communicate with every other computer (node).
- If the network is to support a large number of computers (e.g. the Internet), the network design must **scale** or **be scalable**.
- At the lowest level, we have physical **links** connecting a pair of nodes (point-to-point) or a larger set of nodes (multiple-access).

Types of Physical Links



Direct links: (a) point-to-point; (b) multiple-access.

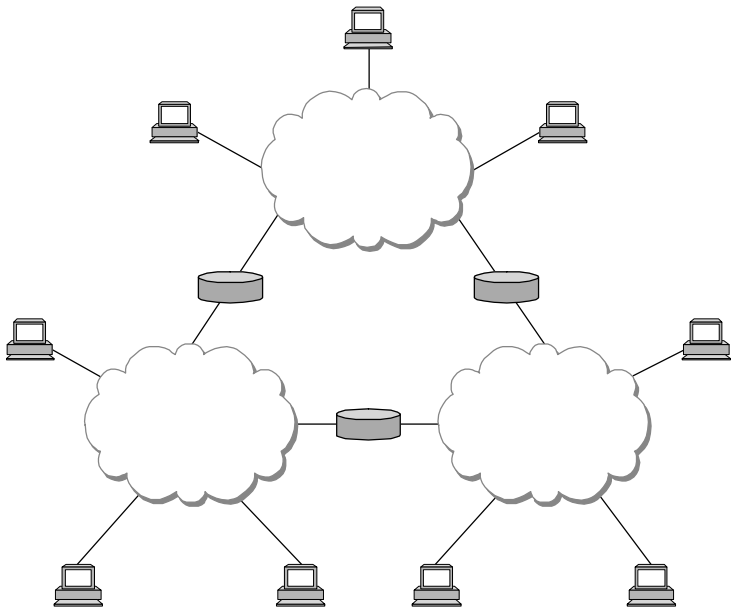
Switched Networks



- Packet switched: most computer networks
- Circuit switched: telephone network

Nodes inside "cloud" are **switches**, outside are **hosts**.

Internetworks



- Independent networks ("clouds") are interconnected to form an **internetwork**.
- The main example of an internetwork is the Internet.
- A node connected to several networks is called **router** or **gateway**.

Network Characterization by Size

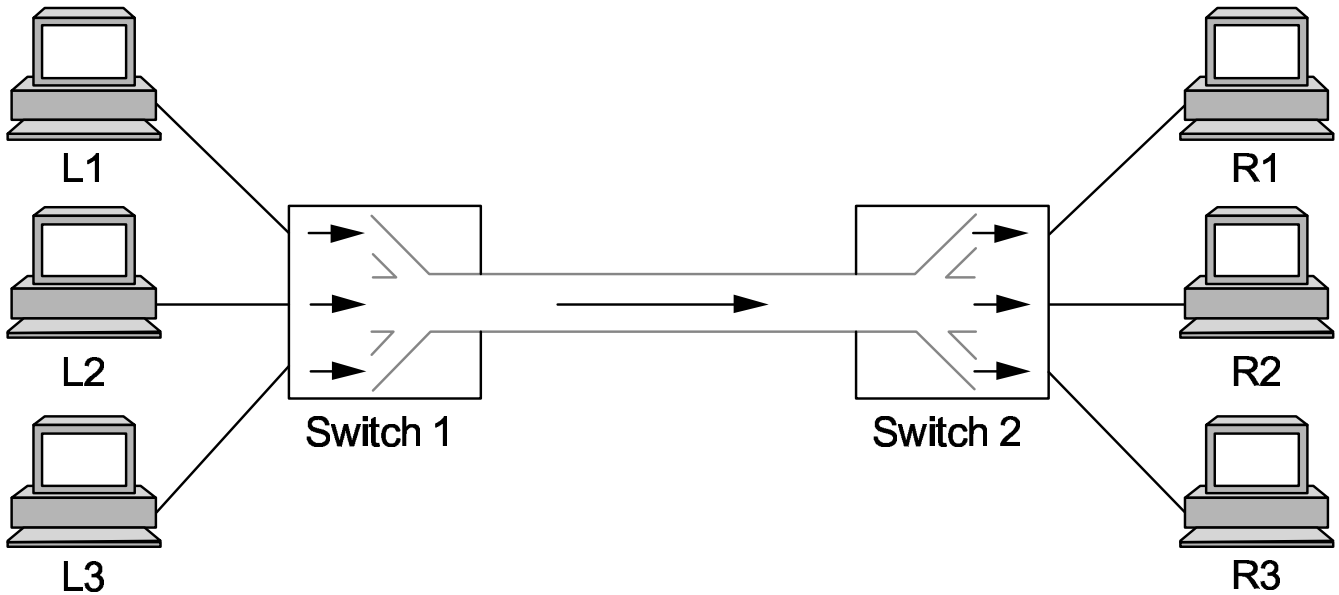
- LAN (local area network): less than 1km
- MAN (metropolitan area network): tens of kilometers
- WAN (wide area network): national or international

Addresses and Routing

- Each node on the network must have an address (a byte string that distinguishes it from all other nodes).
- When a source node wants to send a message to a destination node, it specifies the address of the destination node.
- The switches and routers of the network use the destination address to decide how to forward the message toward the destination. This process is called routing.
- Apart from unicast (sending to one node), there is also multicast (sending to a set of nodes) and broadcast (sending to all nodes of a network).

1.2.2 Cost-Effective Resource Sharing

All nodes must share the network resources. Example:



- The three flows of data are multiplexed onto a single link by Switch 1.
- They are demultiplexed into separate flows by Switch 2.

Types of Multiplexing

- STDM (synchronous time-division multiplexing): divide time into equal-sized quanta, assign quanta to flows in round-robin fashion
- FDM (frequency-division multiplexing): transmit different flows at different frequencies

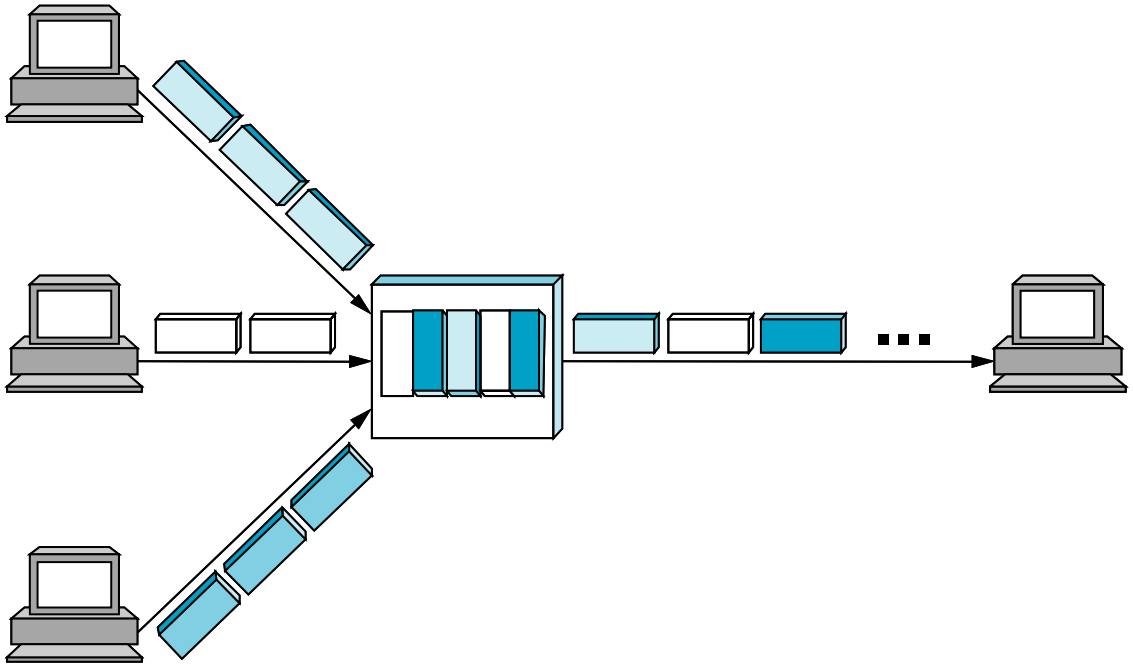
Both are not ideal if the number of flows changes or if flows are idle. Computer networks mostly use:

- Statistical Multiplexing
  - Share link over time as in STDM, but transmit data from each flow "on demand" rather than during a predetermined time slot.

Packets

- In statistical multiplexing, the size of data blocks that can be transmitted at one time is usually limited.
- This limited-size data block is referred to as packet.
- A message can be of arbitrary size and is broken into packets for transmission (fragmentation) and reassembled at the receiver.

Packet Multiplexing Example

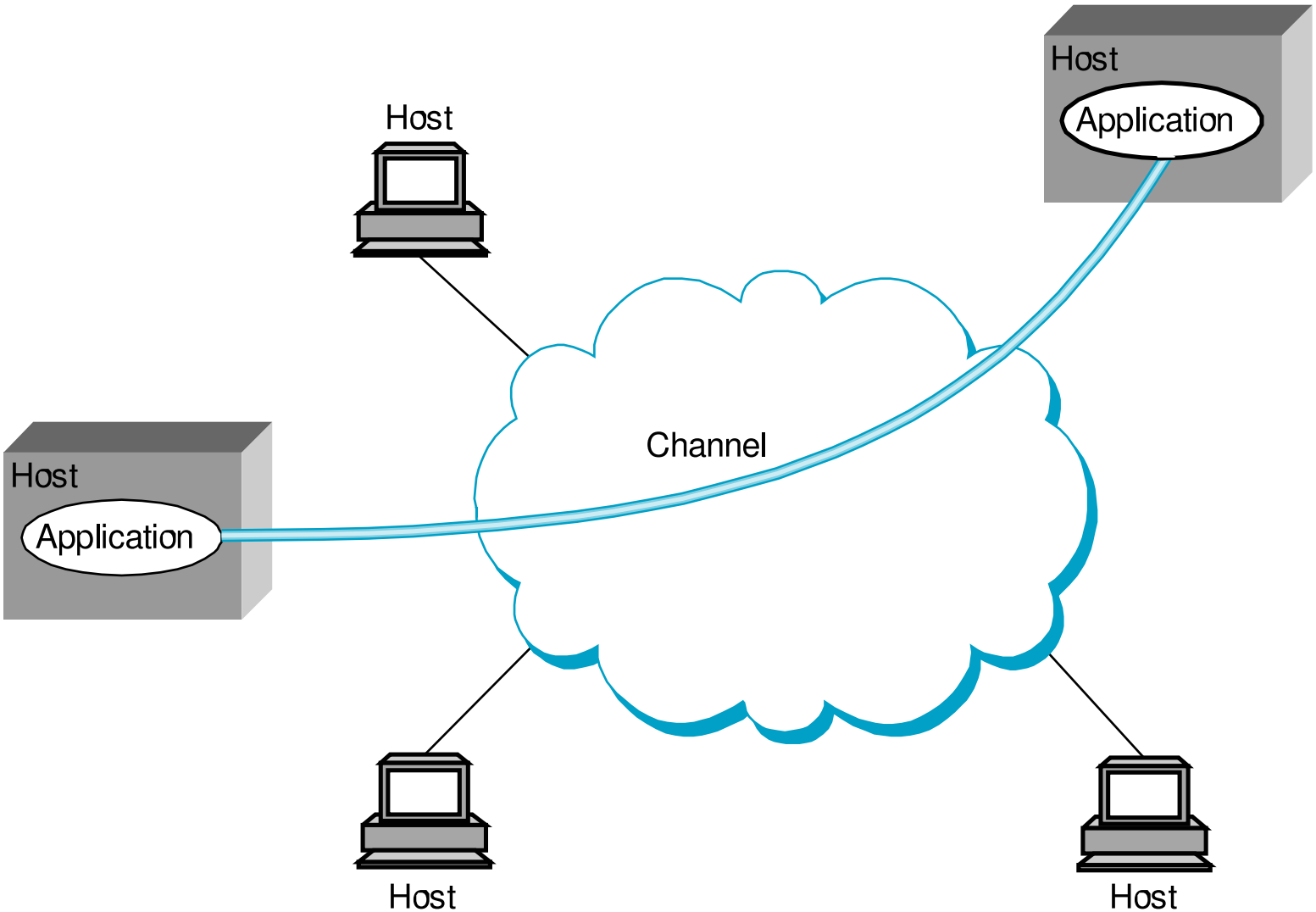


- Switch must decide in which order to service packets, e.g. FIFO (first-in first-out) or round-robin.
- If more packets arrive than can be forwarded by the switch, they are stored in a buffer or, if the switch runs out of buffer space, discarded (congestion).

1.2.3 Support for Common Services

- A network does not only deliver packets; it must provide means for application programs running on the hosts connected to the network to communicate in a meaningful way.
- Many application programs need similar services; the network designer must identify the right set of common services so that they can be implemented once, rather than in each application program separately.
- Intuitively, we want to have **channels** over which application-level processes can communicate with each other.

Abstract Channel between Processes



Channel Functionality

What functionality should channels provide?

- Do all messages need to be delivered, or is it acceptable if some fail to arrive?
- Must all messages arrive in the same order in which they are sent?
- Does the network need to ensure that no third parties are able to eavesdrop on the channel?

Examples of basic types of channels:

- Request/reply channel: client sends request, server answers with reply; e.g. FTP, digital library, WWW
- Message stream channel: continuous stream of data, e.g. videoconferencing or video-on demand

Where to Implement It

- Reoccurring issue in network design: **Where** should functionality be implemented?
  - Network provides bit pipe, all high-level functionality is implemented at end hosts.

    versus

  - Push functionality onto the switches, allow end hosts to be "dumb" devices (e.g., telephone handsets).

Reliability

- Reliable message delivery is highly desirable.
- Possible causes for failures:
  - Machine crashes, fibre cuts
  - Transmission errors due to interference
  - Buffer overflow at switches
  - Software bugs
- Main classes of failures:
  - Bit errors (or, if consecutive bits are affected, burst errors): affect one out of 10^6 to 10^7 bits on copper-based cable, and one out of 10^{12} to 10^{13} bits on optical fibre.
  - Packet loss
  - Node or link failure

1.3 Network Architecture

- A network architecture is a general blueprint that guides the design and implementation of networks.
- Two widely referenced architectures:
  - OSI architecture
  - Internet architecture
- Identify abstractions that provide a useful service and can be efficiently implemented in the underlying system.
- Abstraction naturally leads to **layering**.

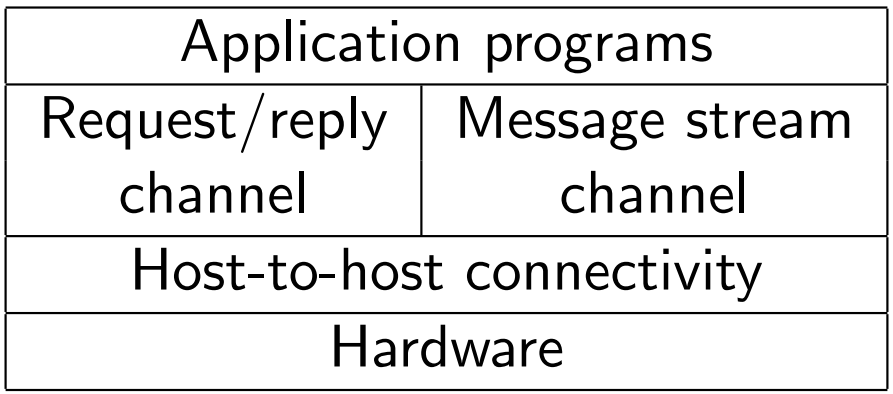
1.3.1 Layering and Protocols

- Layering: Start with the services provided by the underlying hardware, and then add a sequence of layers, each providing a higher level of service.
- Services provided at higher layer are implemented in terms of services provided by lower layers.
- Simple example:

Application programs
Process-to-process channels
Host-to-host connectivity
Hardware

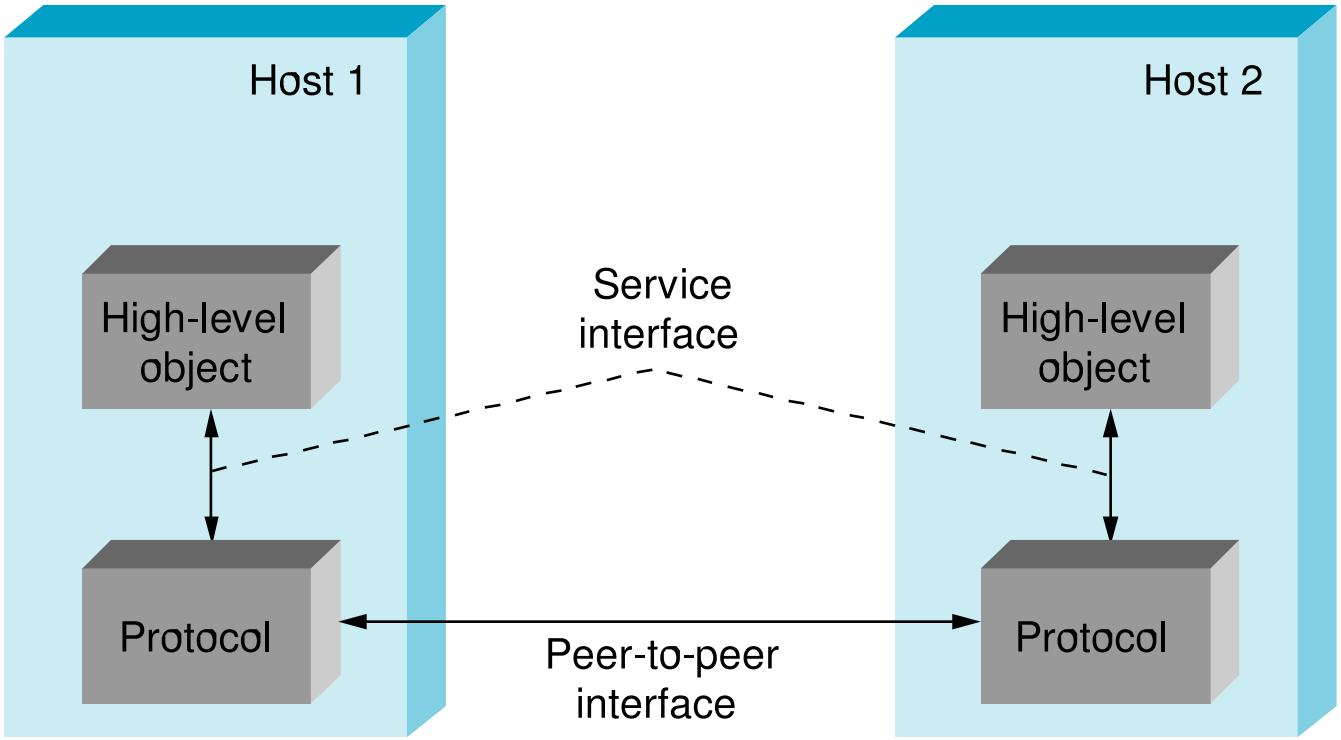
Alternative Abstractions

- Slightly more general example:



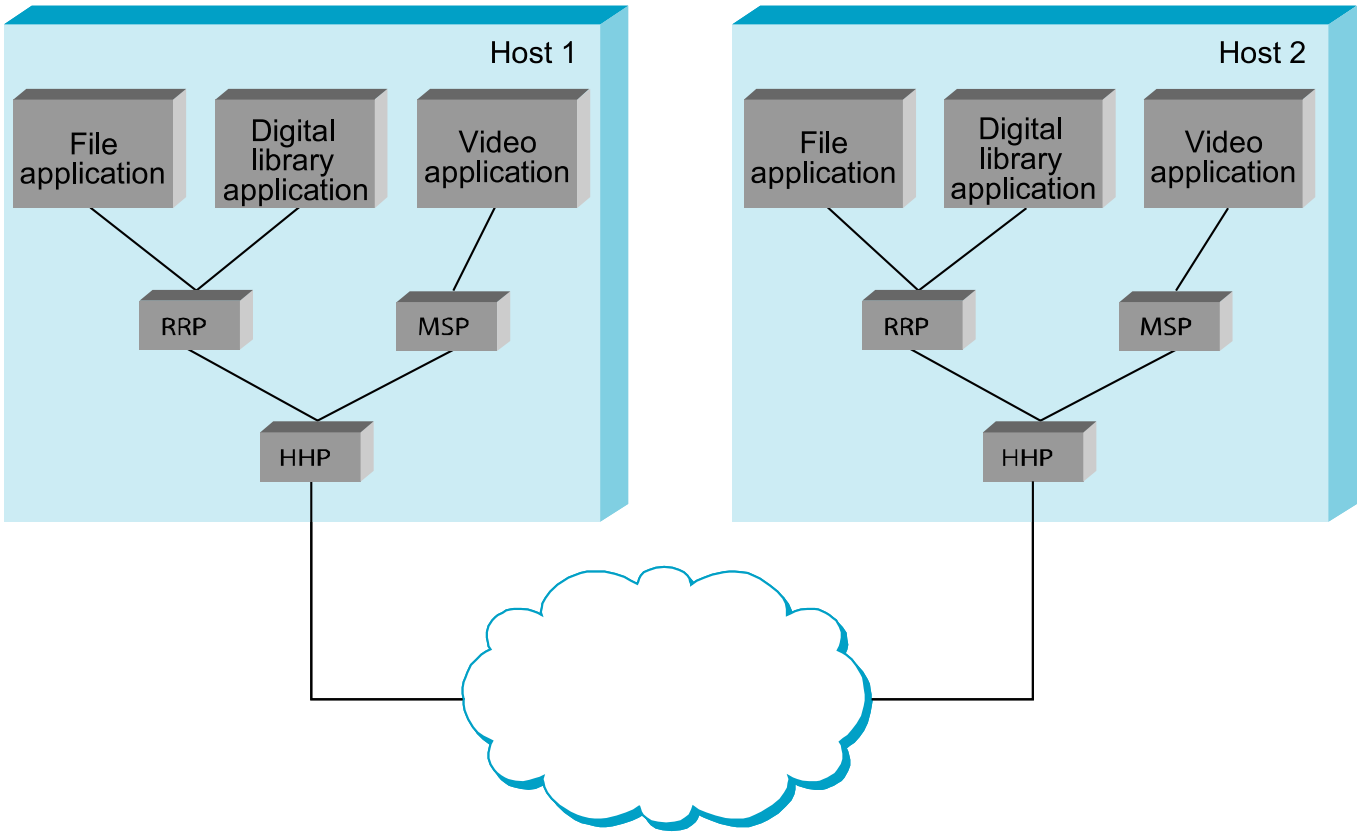
- The abstract objects that make up the layers are called **protocols**.
- Each protocol defines two interfaces:
  - **Service interface** to objects on same node
  - **Peer interface** (or peer-to-peer interface) to its counterpart (peer) on another node

Protocol Interfaces



- At the hardware level, the peers communicate directly.
- Higher-level protocols communicate with their peers by passing messages to some lower-level protocol.

Protocol Graph

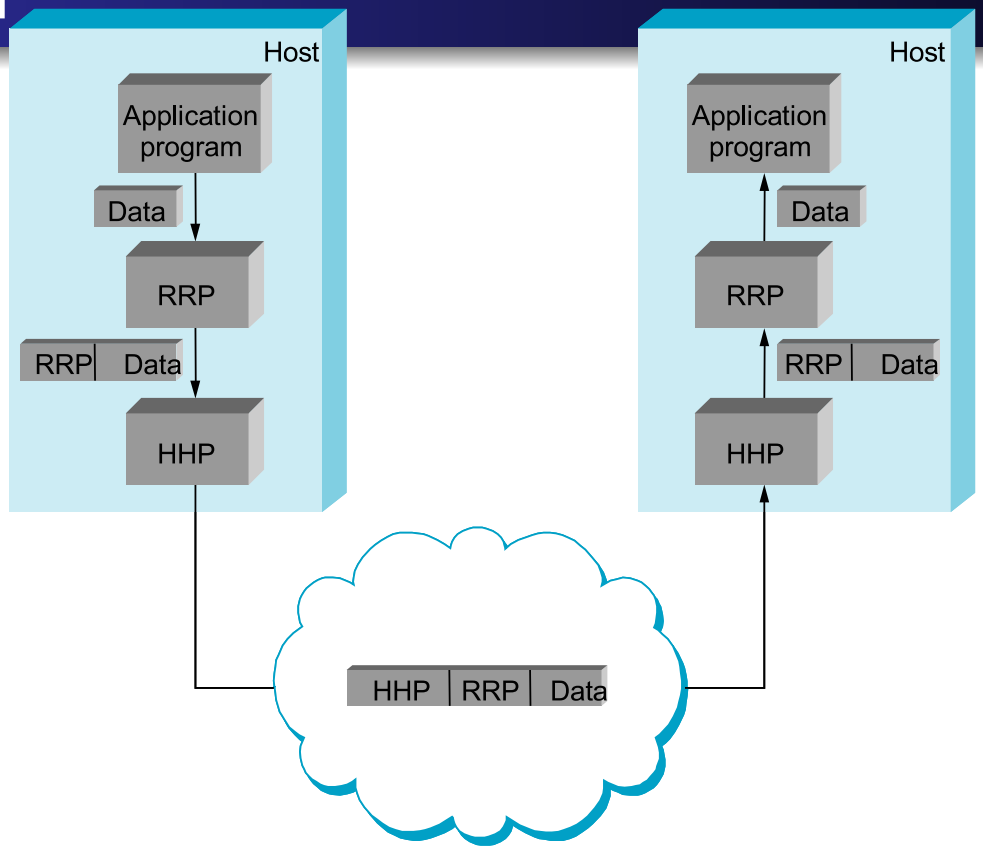


- In this example: HHP = host-to-host protocol, RRP = request/reply protocol, MSP = message stream protocol
- We say, e.g., that the file application uses the services of the **protocol stack** RRP/HHP.

Specification and Implementation

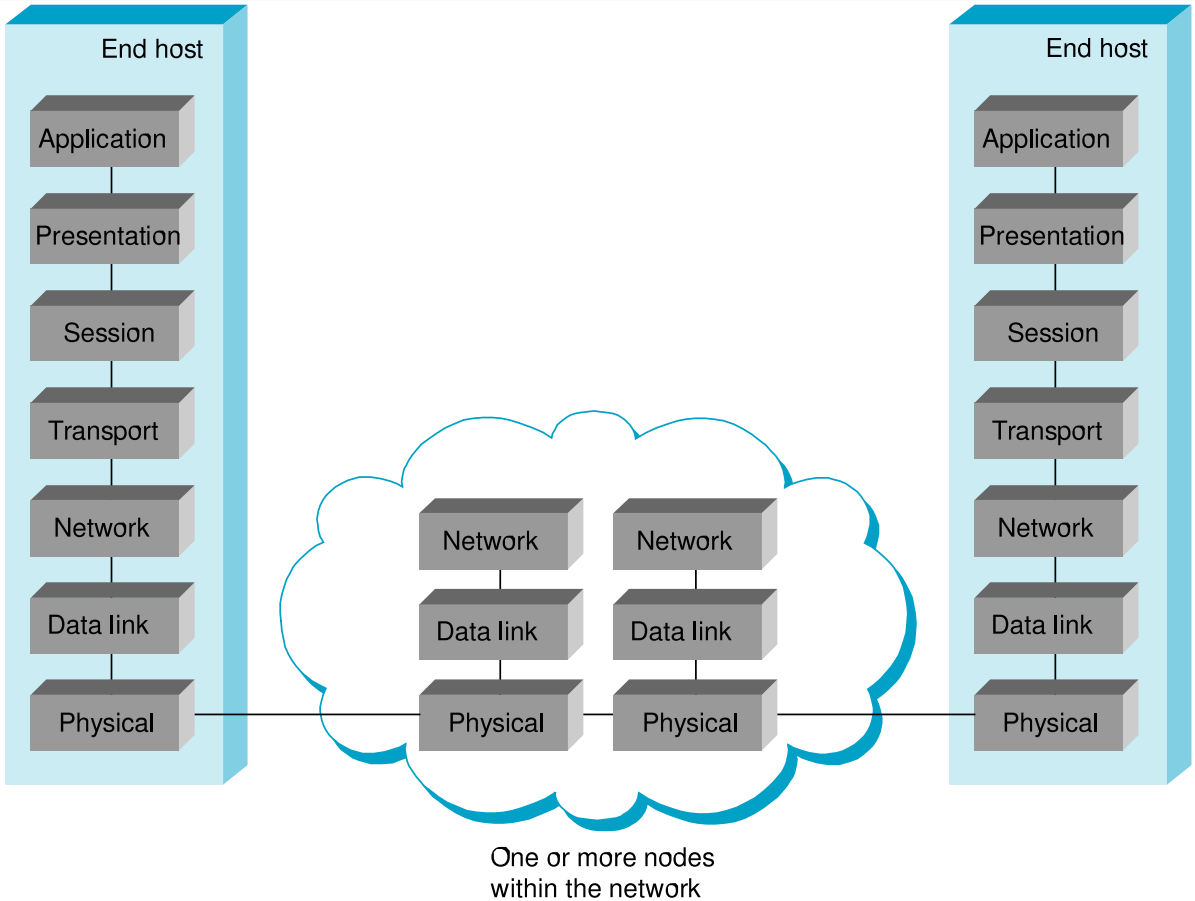
- The term "protocol" is used in two different ways:
  - It can refer to the abstract interfaces (service interface and peer interface), the **protocol specification**.
  - It can refer to the module that implements these two interfaces.
- Two different implementations that accurately implement a protocol specification are said to **interoperate**.

Encapsulation



- Each layer encapsulates message it gets from higher layer by adding its **header** to the **body** (**payload**).
- At destination, message is processed in opposite order (using a **demux key** to select the right higher-level protocol or application, if necessary).

Open Systems Interconnection (OSI)



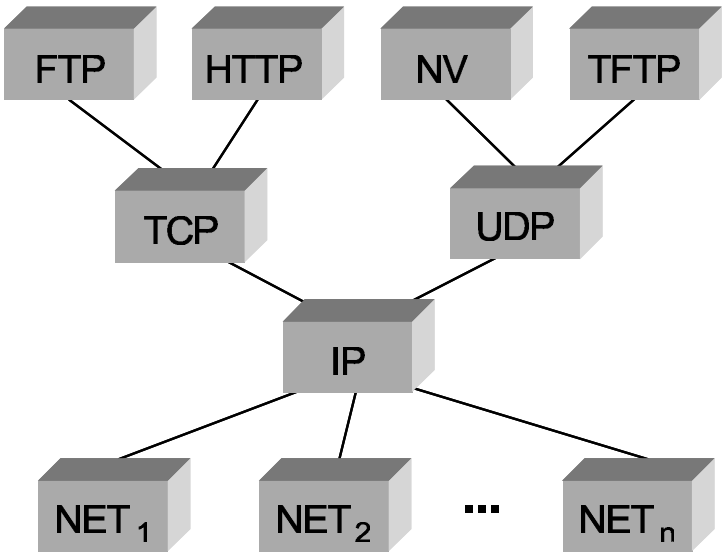
- OSI architecture: reference model defined by ISO (International Standards Organization).
- Seven layers, one or more protocols at each layer.

OSI Layers

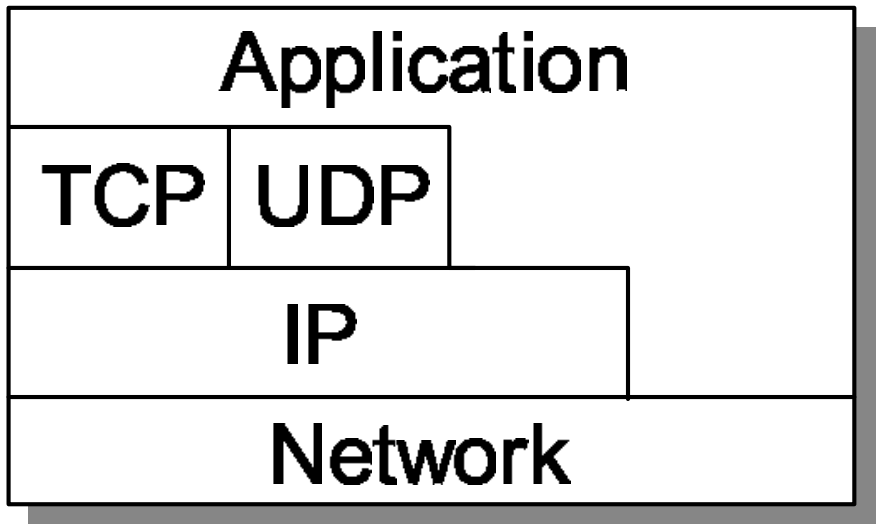
- Layer 1 (**physical**): transmission of raw bits over a link
- Layer 2 (**data link**): collects streams of bits into a larger aggregate called **frame**; typically implemented by network adaptors and device drivers
- Layer 3 (**network**): handles routing of **packets** among the nodes
- Layer 4 (**transport**): implements "process-to-process" channel, unit of data exchanged is called **message**
- Layer 5 (**session**): provides name space to tie together different transport streams of a single application
- Layer 6 (**presentation**): format of data to be exchanged (bit order, video encoding, etc.)
- Layer 7 (**application**): FTP, HTTP, etc.

Internet Architecture

Protocol graph:



Alternative view:



- Internet architecture (or: TCP/IP architecture) has evolved from ARPANET.
- Can be fitted into 7-layer OSI model, but is usually viewed as 4-layer model.
- IETF (Internet Engineering Task Force)

Internet Architecture Layers

- Lowest layer (Network): variety of network protocols, implemented by combination of hardware and software (e.g. Ethernet, FDDI, ATM, etc.)
- Second layer (IP): Internet Protocol, a single protocol that supports the interconnection of multiple networking technologies into a single, logical internetwork
- Third layer: Two main protocols (end-to-end protocols) that provide alternative logical channels:
  - TCP (Transmission Control Protocol): reliable byte-stream
  - UDP (User Datagram Protocol): unreliable datagram delivery
- Top layer: Range of application protocols such as HTTP, FTP, Telnet, SMTP (Simple Mail Transfer Protocol), etc.

Internet Architecture Features

- **No strict layering**: Application is free to bypass layers and to directly use IP or even the underlying networks
- Central philosophy: **IP is the focal point** of the architecture (cf. hourglass shape of protocol graph); issue of message delivery is completely separated from process-to-process communication service.
- IETF culture: In order for someone to propose a new protocol to be included in the architecture, they must produce both a protocol specification and **at least one representative implementation**.

"We reject kings, presidents, and voting. We believe in rough consensus and running code." (Dave Clark)

1.4 Implementing Network Software

- Phenomenal success of the Internet: number of computers connected to the Internet has been doubling every year since 1981; total Internet traffic surpassed voice phone system in 2001.
- Contributing factor: much of the Internet functionality is provided by software running on general-purpose hardware.
- New functionality can be added with "just a small matter of programming."
- New applications and services (e-commerce, videoconferencing, IP telephony, etc) have been showing up at a phenomenal pace.

1.4.1 API (Sockets)

- The interface exported by the network is called **application programming interface (API)** and is typically provided by the operating system (OS) on a computer.
- The API provides the syntax by which the services of the protocols can be invoked.
- The **socket interface**, originally provided by Berkeley Unix, is now supported by virtually all OSs.
- Main abstraction: the socket. Think of it as the point where a local application process attaches to the network.

Socket Operations

- create a socket
- attach a socket to the network
- send messages through a socket
- receive messages through a socket
- close socket

Socket interface in C/C++:

- int socket(int domain, int type, int protocol)

  domain: PF_INET, PF_UNIX, PF_PACKET
  type: SOCK_STREAM, SOCK_DGRAM
  protocol: UNSPEC for TCP and UDP
  ➠ returns a handle for the socket created

Passive Open (Server)

- int bind(int socket, struct sockaddr *address, int addr_len)
  binds socket to the specified address (IP address and port number) on local machine
- int listen(int socket, int backlog)
  specifies how many connections can be pending on that socket
- int accept(int socket, struct sockaddr *address, int *addr_len)
  blocks until a remote participant has established a connection, then returns a **new** socket for that connection (address of remote participant is stored in address)

Active Open (Client)

- int connect(int socket, struct sockaddr *address, int addr_len) establishes connection to the remote participant specified by address

Once the connection is established:

- int send(int socket, char *message, int msg_len, int flags) sends message over specified socket
- int recv(int socket, char *buffer, int buf_len, int flags) receives message from specified socket into buffer

Sockets in Java

- Networking-related classes are available in the package java.net.
- Classes Socket and ServerSocket implement TCP sockets (client socket and server socket, respectively).
- Class DatagramSocket implements UDP socket.
- Classes InetAddress and InetSockAddress implement IP addresses and socket addresses (IP address + port number), respectively.

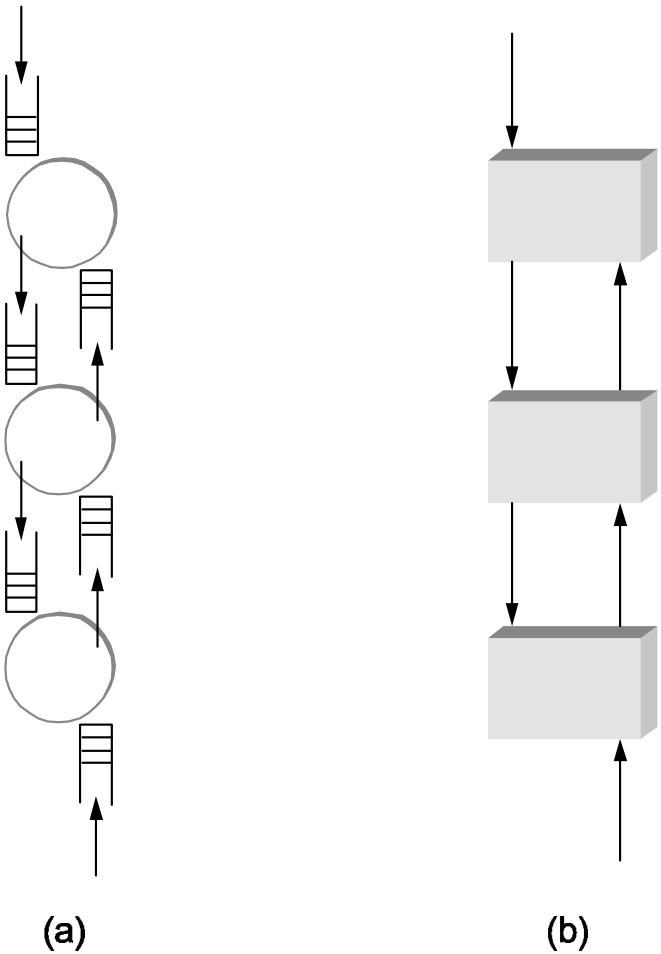
1.4.2 Example Application

- Simple talk application
- Server accepts connections on port 5432 and displays all strings received.
- Client connects to server and sends strings typed in by the user.
- Only one connection between server and client at any time.
- C source files: client.c and server.c
- Java source files: Client.java and Server.java

1.4.3 Protocol Implementation Issues

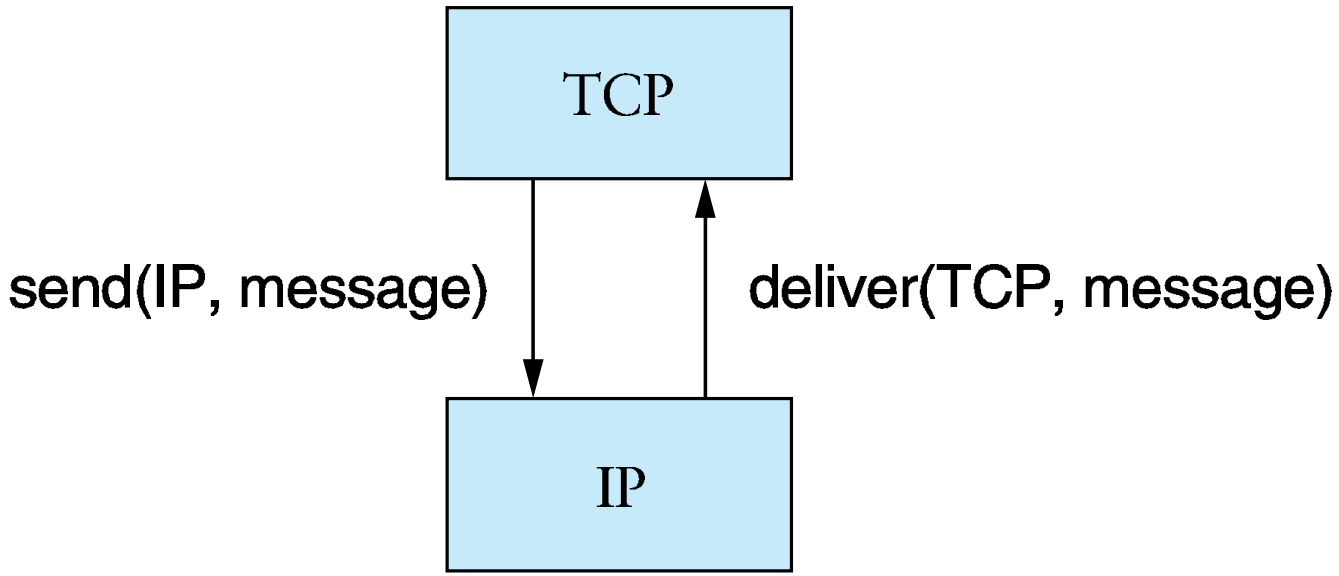
- Applications access TCP in a similar way as TCP accesses IP.
- However, different layers of the protocol stack do not use the socket interface to call each other.
- Protocol-to-protocol interfaces are optimised for efficiency.

Process Model



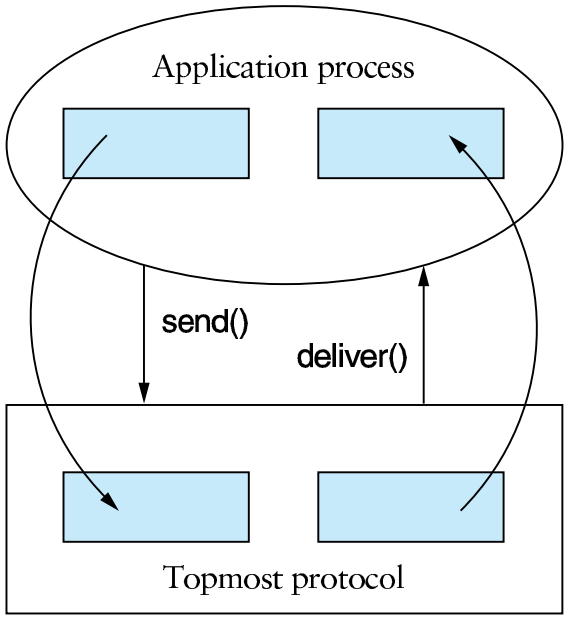
- Messages are passed through several layers of the protocol stack.
- Two choices for assigning message processing to processes (threads):
  - (a) process-per-protocol: requires many context switches
  - (b) process-per-message: uses procedure calls, usually more efficient

Receive vs. Deliver



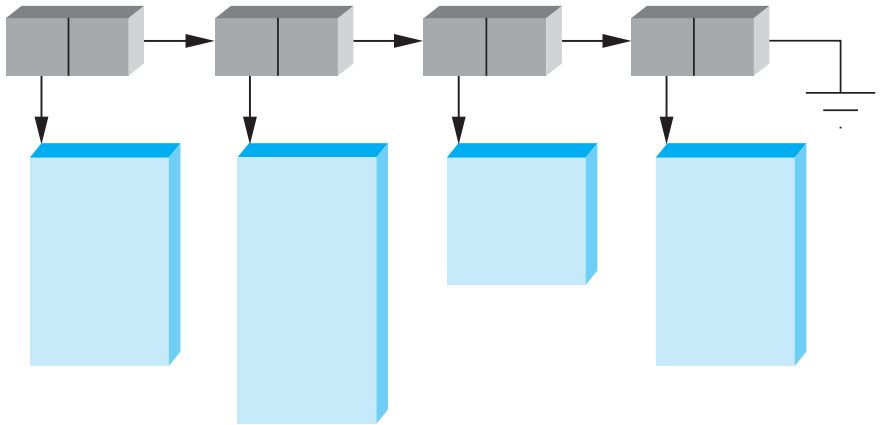
- Incoming messages are handed from lower layers to higher layers.
- Inefficient to use receive operation initiated by higher layer.
- Typically a protocol uses an **upcall** to deliver a received message to a high-level protocol.

Message Buffers



- Second inefficiency of socket interface: message must be copied from application's buffer to network buffer, and vice versa.
- Copying of messages is very time-consuming. (Note: Processor speed is growing much faster than memory speed.)

Copy-Free Message Abstraction



- Most network subsystems define an abstract data type for messages that is shared by all protocols.
- This allows to pass messages up and down the protocol graph without copying.
- Copy-free manipulation of messages (adding and stripping header, etc.).
- Implementation usually involves list of pointers to message buffers.

1.5 Performance

- Computer networks are expected to perform well.
- In networking, "design for performance" is often better than "first get it right and then make it fast."
- **Bandwidth and throughput**: confusing terms
  - Literally, bandwidth is the width of a frequency band, e.g. a telephone line supporting 300 Hz to 3,300 Hz has bandwidth 3,000 Hz.
  - Commonly, bandwidth refers to the number of **bits per second** that can be transmitted over a link (e.g. 10 Mbps Ethernet).
  - **Throughput** refers to the **measured performance**, e.g. an application may only be able to transmit 2 Mbps over a 10 Mbps link.

1.5.1 Bandwidth and Latency

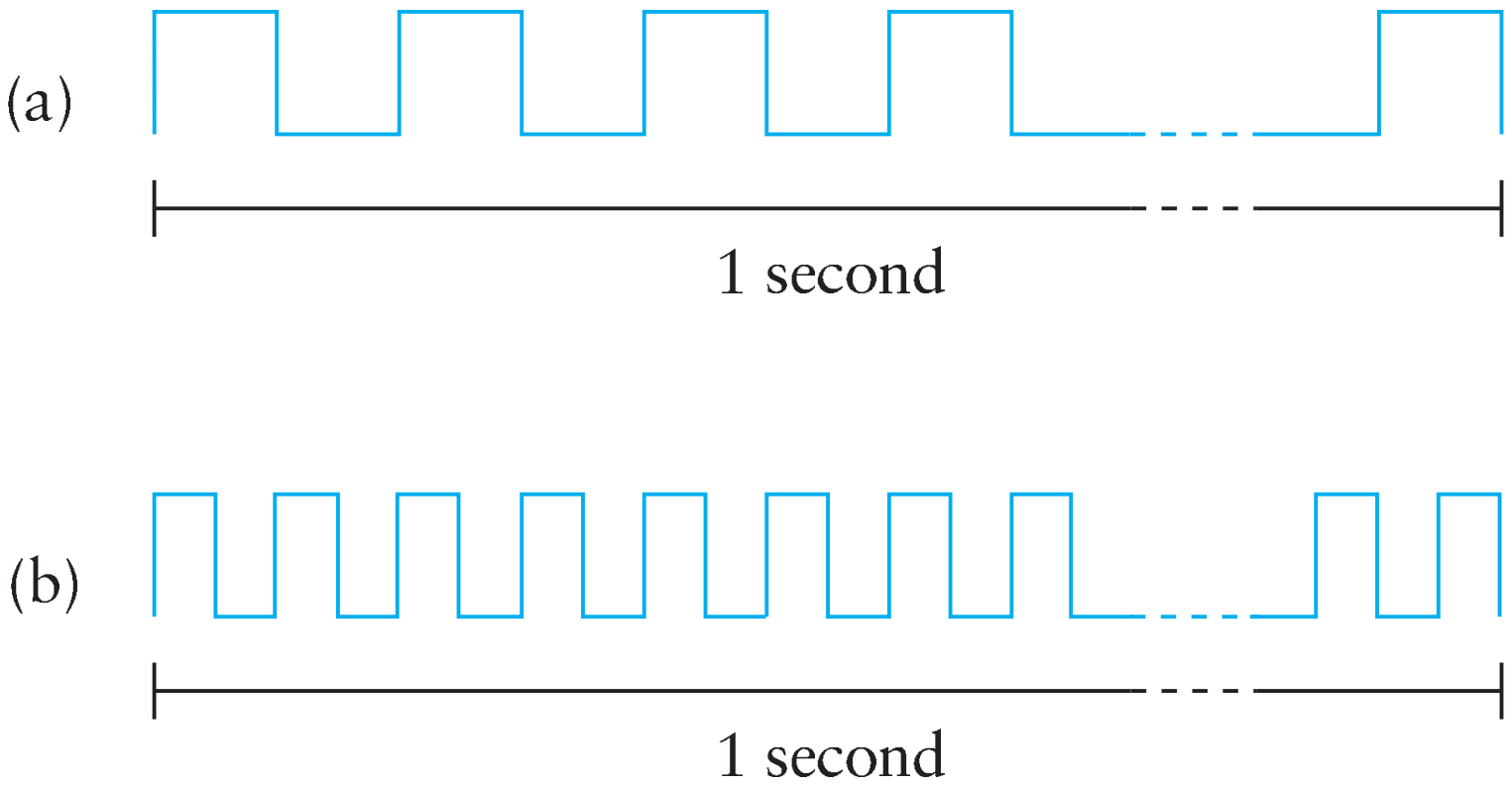
Bandwidth (throughput):

- Number of bits that can be transmitted over the network or link in a certain time.
- Example: 10 Mbps = 10 million bits per second
- It takes 0.1 μs to transmit one bit over a 10 Mbps link.

Latency (delay):

- Corresponds to how long it takes a message to travel from one end of the network or link to the other
- **Round-trip time (RTT)**: time it takes to send a bit from one node to another and back.
- Typical RTT: 1 ms in LAN, 100 ms cross-country (USA)

The "Width" of a Bit



- (a) 1 Mbps link: each bit is 1 μs wide
- (b) 2 Mbps link: each bit is 0.5 μs wide

Latency Calculation

- Latency = Propagation + Transmit + Queue
- Propagation = Distance / SpeedOfLight
- Transmit = MessageSize / Bandwidth

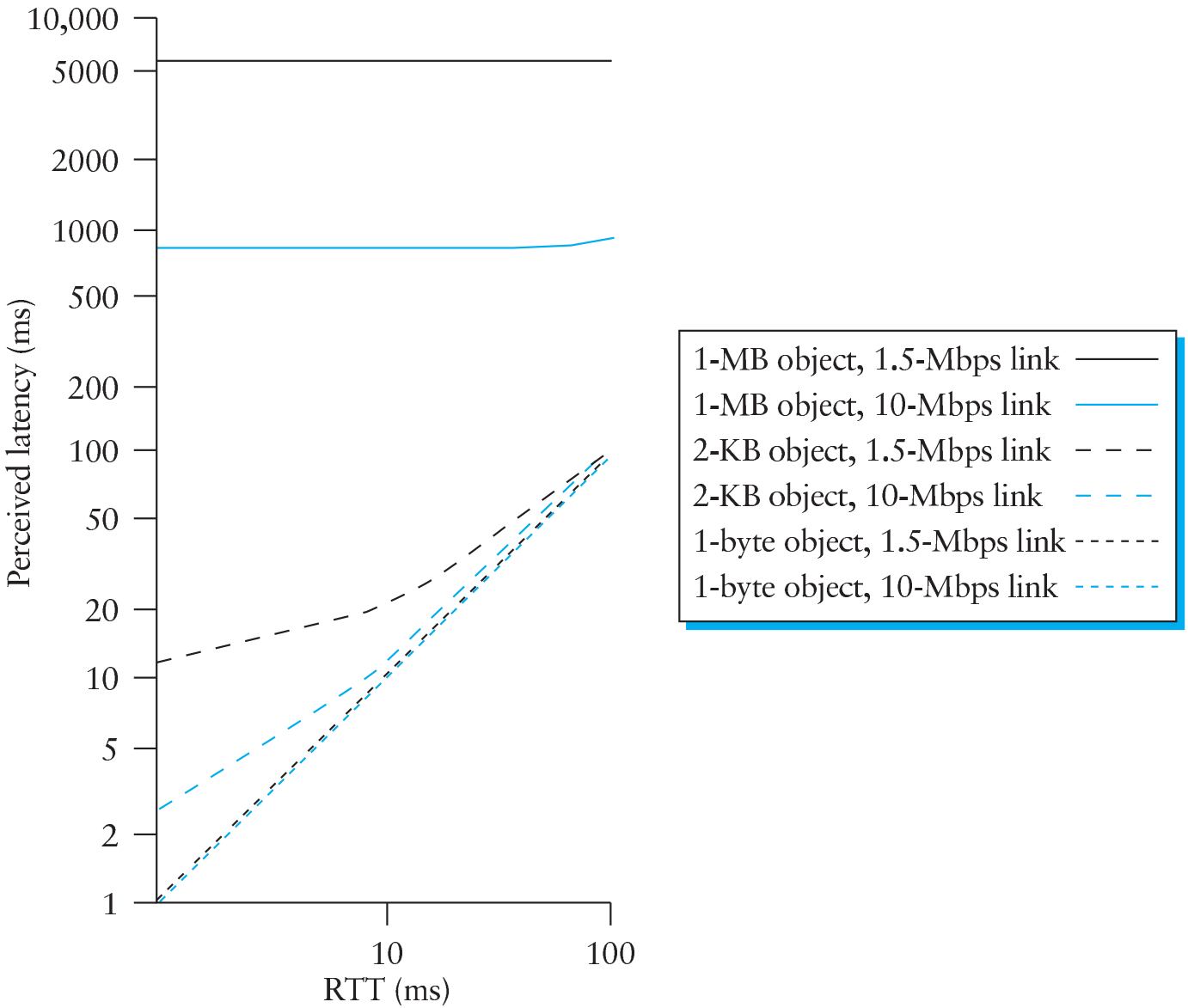
Speed of Light:

- In vacuum: 3.0×10^8 m/s
- In cable: 2.3×10^8 m/s
- In fibre: 2.0×10^8 m/s

Examples

- Transmitting 1 byte to a server and receiving a 1 byte response:
  - Transmit time: 8 μs on 1 Mbps link
    0.08 μs on 100 Mbps link
  - ➠ RTT is important, bandwidth does not matter
- Downloading a 25 MB image file from a digital library
  - Transmit time: 20 seconds on 10 Mbps link
  - ➠ RTT is irrelevant, bandwidth matters
- **Remark**: Kbps and Mbps usually refer to 10^3 bps and 10^6 bps, while KB and MB usually mean $2^{10} = 1{,}024$ bytes and $2^{20} = 1{,}048{,}576$ bytes, respectively.

Illustration: Bandwidth and Latency



1.5.2 Delay × Bandwidth Product

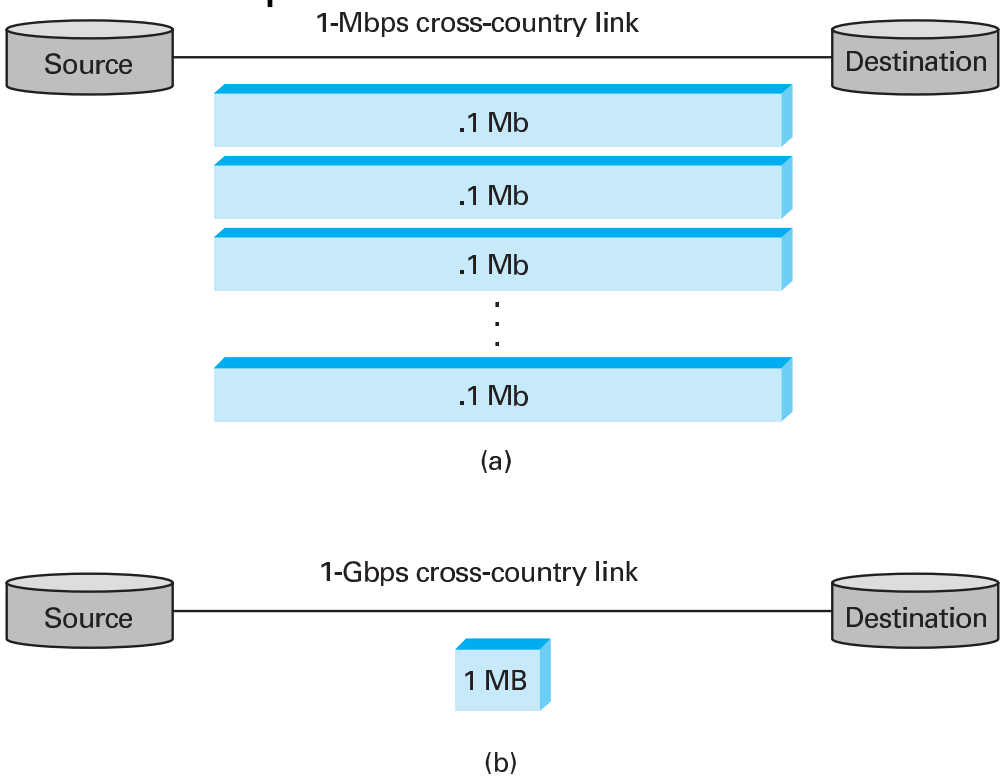
- Delay × bandwidth represents the number of bits that are "in flight" on a link (volume of the "pipe").
- For example, a 45 Mbps link with 50 ms one-way delay holds

$$50 \times 10^{-3} \text{ seconds} \times 45 \times 10^{6} \text{ bits/second}$$

$= 2.25 \times 10^{6}$ bits, or about 275 KB of data.
If the sender transmits at full rate, 45 Mbps × 100 ms = 550 KB of data have been transmitted by the time the first response from the receiver arrives at the sender!

1.5.3 High-Speed Networks

- Bandwidth is ever increasing, but latency remains essentially the same.
- Transmitting a 1 MB file corresponds to (a) 80 pipes full of data over a 1 Mbps cross-country link, but (b) only 1/12 of one pipe on a 1 Gbps link.



Effective Throughput

- Effective end-to-end throughput:

$$\text{Throughput} = \frac{\text{TransferSize}}{\text{TransferTime}}$$

where: $\text{TransferTime} = \frac{1}{2}\text{RTT} + \frac{\text{TransferSize}}{\text{Bandwidth}}$

Here, $\frac{1}{2}$RTT represents the propagation delay of the link.

- Sending a 1 MB file over a 1 Gbps network with an RTT of 200 ms requires a transfer time of

$$100 \text{ ms} + 1 \text{ MB} \,/\, 1 \text{ Gbps} = 108 \text{ ms}.$$

Effective throughput: 1 MB / 108 ms = 74.1 Mbps.

1.5.4 Application Performance Needs

- The needs of applications are not always as simple as wanting "as much bandwidth as the network can provide" or "as little delay as possible".
- For example, a video stream with 352×240 pixels, 24 bits per pixel, and 30 frames per second would require

$$352 \times 240 \times 24 \times 30 \text{ bps} \approx 61 \text{ Mbps}$$

  throughput. Additional bandwidth is of no interest.
- In practice, video streams are compressed and have a lower average transmission rate, but occasional bursts.

Jitter

- **Jitter** is the variation in latency from packet to packet.
- If video stream source transmits one packet every 33 ms (i.e. 30 packets per second), then it is desirable that the packets arrive at the same rate at the receiver.
- If packets incur different delays, the interpacket gaps are variable at the receiver:

- Jitter is usually caused by queueing delays at intermediate nodes.
- Limited jitter can be handled by buffering (e.g. delaying the playback of a video stream).