Networking

Operating Systems

Oleg Goldshmidt

`ogoldshmidt@computer.org`

Lecture 12

Networked Systems

- various resources (CPUs, disks, clocks, etc.) are connected via one or more communication networks
- much of the low level issues (drivers, interrupt handling, buffering, etc.) is “just” I/O— already covered
- “local” and “remote” resources
  - hardware: disks, printers, CPUs, specialized resources
  - software: files, programs (“services”)
- client/server and peer-to-peer

Networked And Distributed Systems

- networked systems
  - users are aware of the network connecting the resources
  - resources are accessed by logging on to remote machines (`ssh`, `telnet`, etc.) or by transfering data between machines (`HTTP`, `FTP`, etc.)
- distributed systems
  - users are not (or need not be) aware of the network
  - resources are accessed transparently

Why?

- resource sharing
  - different capabilities at different sites
  - reduce resource duplication / increase utilization
  - file sharing
  - processing information in a distributed database
  - using remote specialized HW devices
- computation speedup via load sharing
- reliability: failure detection, failover, recovery, reintegration
- communication: message passing and RPC

Node Types

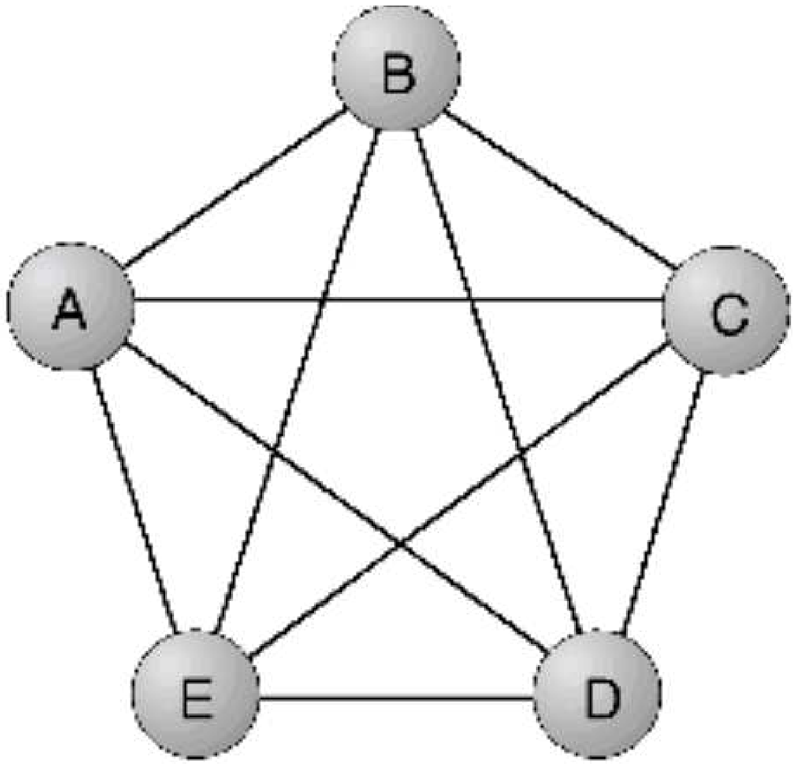
- mainframes: ($\approx 70\%$ of the world's data)
  - large DB: airline reservations, banking, insurance
  - many large attached disks
- servers
  - web, ftp, DB, applications
  - attached or networked disks
- workstations
  - CAD, Office, email private DBs, etc.
  - no disks or a few small/medium attached disks
- computational nodes of HPC systems
  - (floating point) computations
  - normally no disks, rarely system disks

Topology

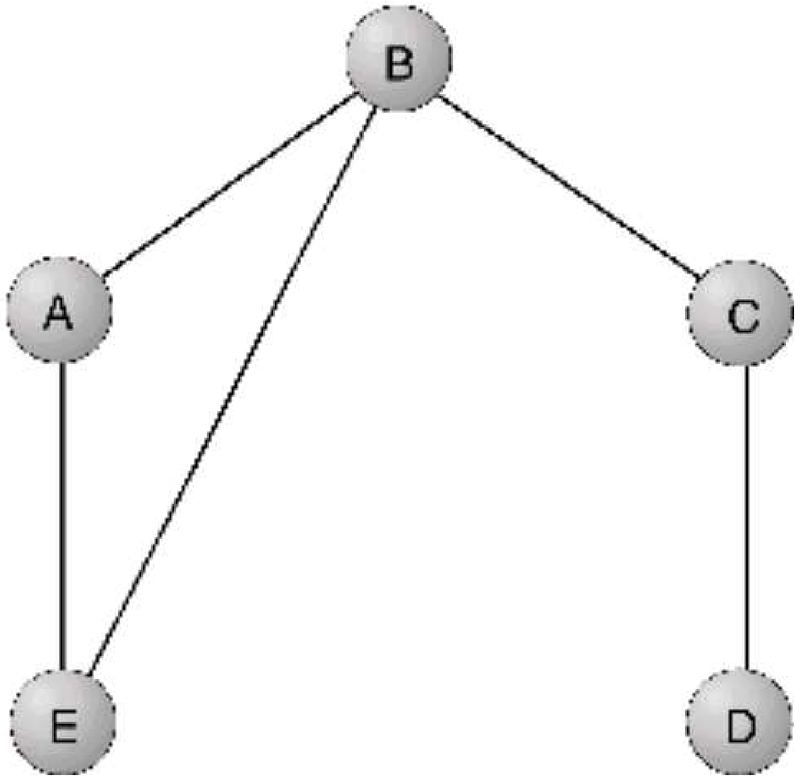
- a lot of different ways to connect sites (nodes) of a system
- design criteria
  - basic cost: how expensive is it to link the various sites?
  - communication cost: how long and how many resources does it take to send a message from site A to site B?
  - reliability: if a link or a site fails, can the remaining sites communicate with each other?
- topologies are usually represented as graphs

Topology Examples I

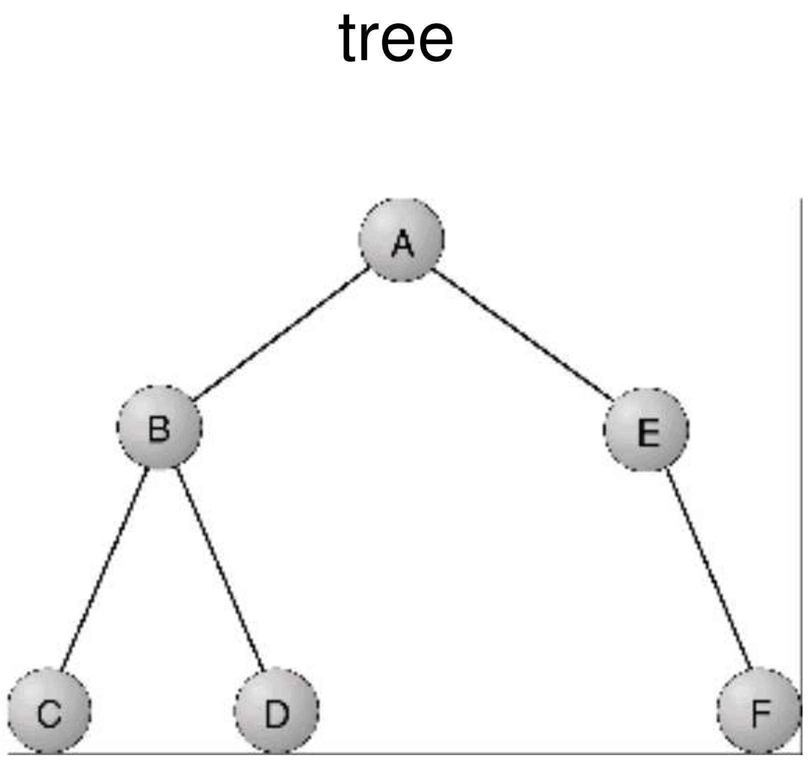
fully connected

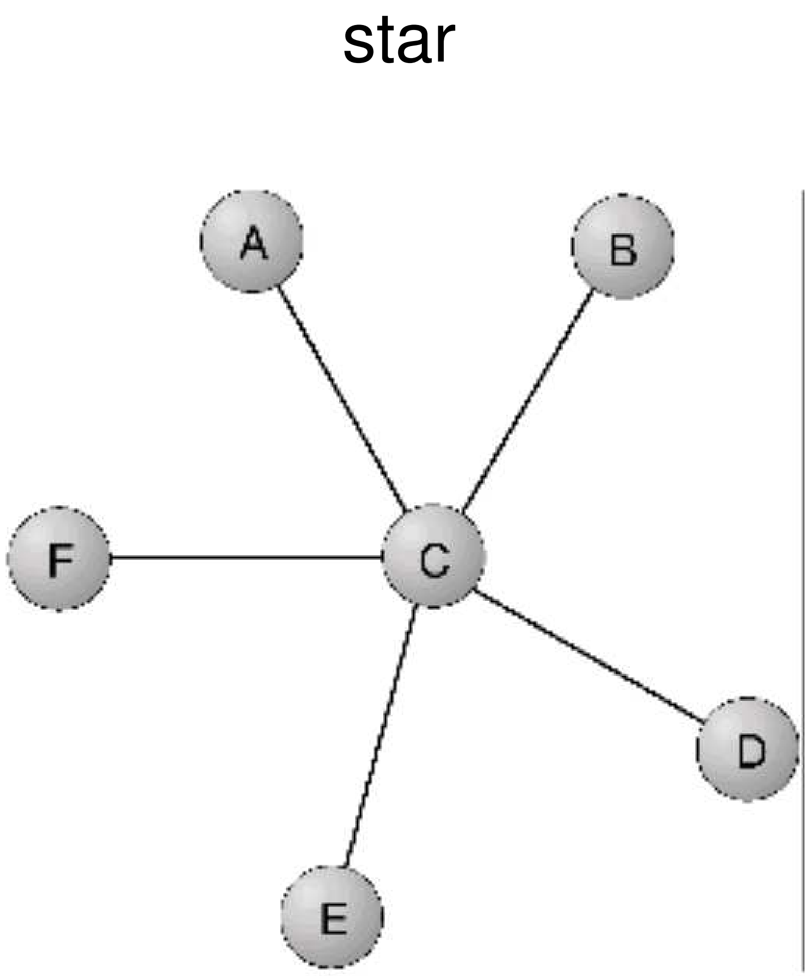


partially connected



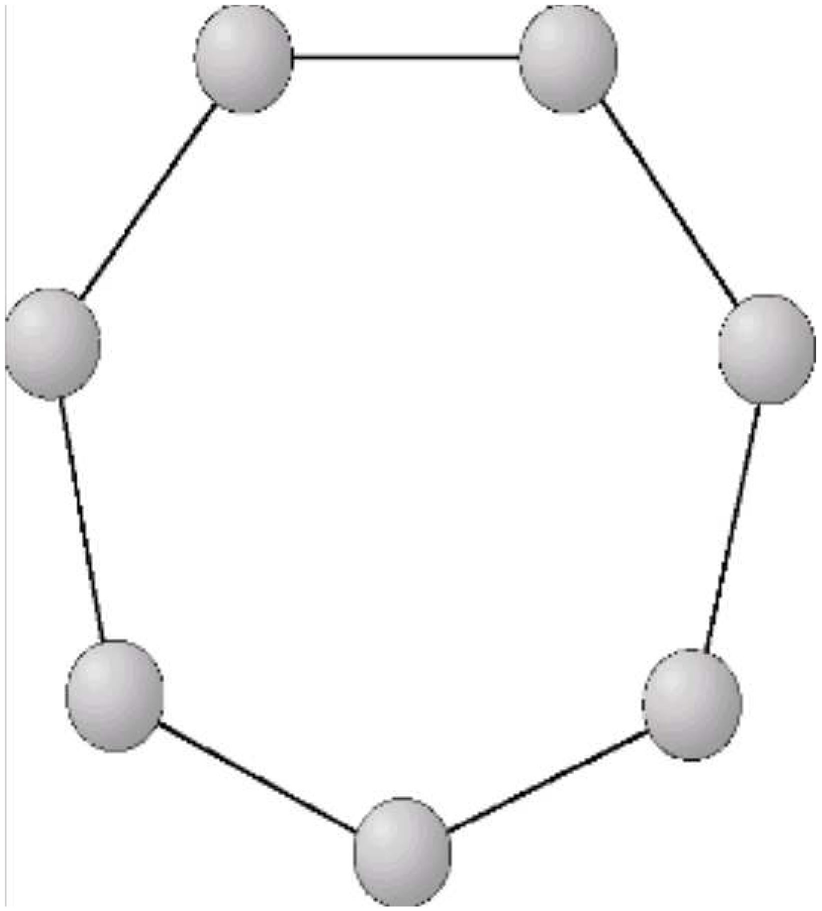
Topology Examples II



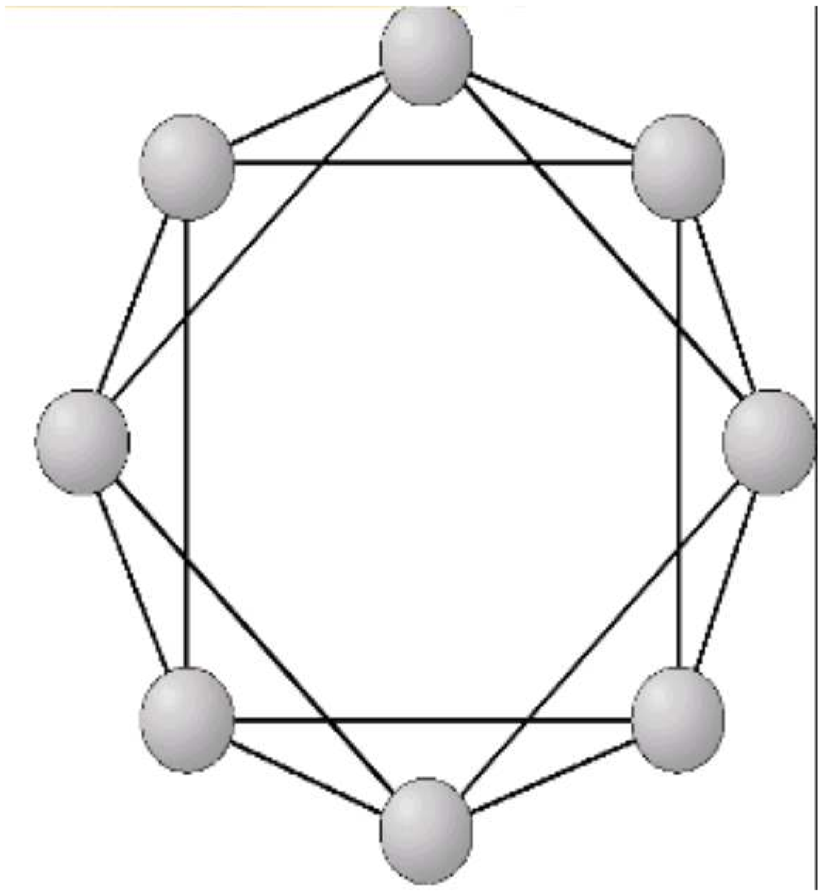


Topology Examples III

ring

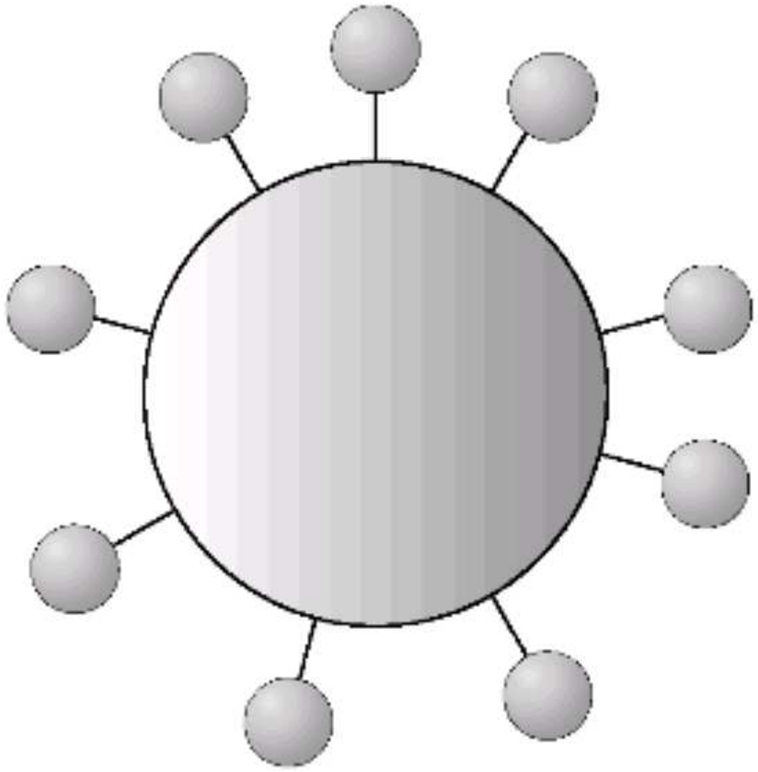


more complex ring

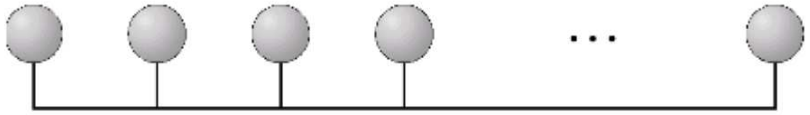


Topology Examples IV

ring bus

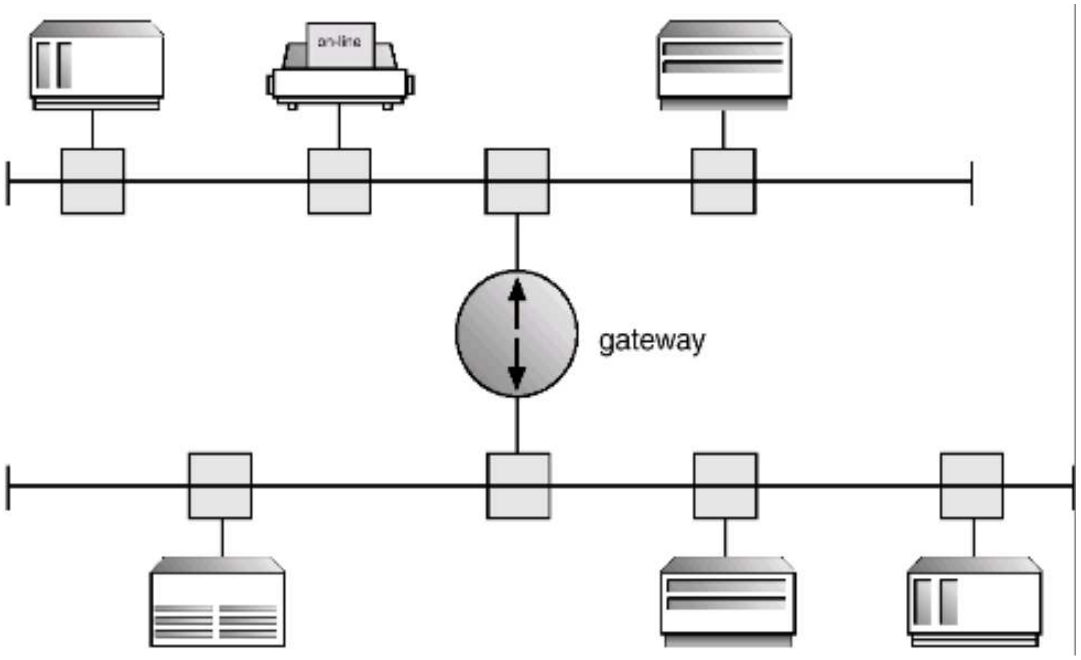


linear bus



Network Types: LAN

- up to a few buildings
- multiaccess bus, ring, or star
- $10 \div 1000\,\mathrm{Mb/s}$
- broadcast is fast and cheap
- nodes:
  - workstations
  - PCs
  - servers
  - an occasional mainframe



Network Types: WAN

- links geographically separated sites
- connections over long-haul lines
  - usually leased
  - involves multiple networks
- speed: non-uniform
  - very high bandwidth in the backbone
  - $0.05 \div 10\,\mathrm{Mb/s}$ — the "last mile" problem
  - asymmetric for download and upload
- broadcast requires multiple messages
- nodes
  - high percentage of servers and mainframes
  - PCs are often connected via LANs

Communications: Issues

- naming and name resolution: how do two processes locate each other and communicate?
- routing strategies: how are messages sent through the network?
- connection strategies: how do two processes send a sequence of messages to each other?
- contention: the network is a shared resource, how do we resolve conflicting demands for its use?
- reliability and survivability: what happens if some of the links and/or nodes fail?
- quality of service: what are the performance guarantees?

Naming And Name Resolution

- local: use `PID` to send a message to a particular process
- remote: identify processes on remote systems using a $\langle$`host-id,port`$\rangle$ pair
- domain name service (DNS): specifies the naming structure of the hosts, as well as name to address resolution
  - humans prefer names
  - computers prefer numbers
  - cannot have a translation table of every host — non-scalable and unreliable
  - hierarchical system of “name servers”
  - local caches are kept

Elementary Name Resolution

- popular name server: BIND (Berkeley Internet Name Domain)
- basic API:

```
#include <netdb.h>
struct hostent {
  char *h_name;        /* official name of host */
  char **h_aliases;    /* alias list */
  int  h_addrtype;     /* host address type */
  int  h_length;       /* length of address */
  char **h_addr_list; /* list of addresses */
}
struct hostent *gethostbyname(const char *name);
struct hostent *gethostbyname2(const char *name, int af);
#include <sys/socket.h>   /* for AF_INET */
struct hostent
*gethostbyaddr(const char *addr,int len,int type);
```

Elementary Service Names

- we would like to refer to services by names, not by port numbers (`/etc/services` or similar provides a map)

```
#include <netdb.h>
struct servent {
  char *s_name;      /* official service name */
  char **s_aliases; /* alias list */
  int  s_port;       /* port number */
  char *s_proto;     /* protocol to use */
}
struct servent *getservbyname(const char *name, const char *proto);
struct servent *getservbyport(int port, const char *proto);

struct servent *s;
s = getservbyname("domain", "udp"); /* DNS using UDP */
s = getservbyname("ftp", "tcp");    /* FTP using TCP */
s = getservbyname("http", NULL);    /* HTTP using TCP */
s = getservbyname("kerberos-adm", "udp"); /* this will fail! */
```

Routing Strategies I

- fixed routing: a path from A to B is specified in advance; path changes only if a hardware failure disables it
  - since the shortest path is usually chosen, communication costs are minimized
  - cannot adapt to load changes
  - ensures that messages will be delivered in the order in which they were sent
- virtual circuit: a path from A to B is fixed for the duration of one session; different sessions involving messages from A to B may have different paths
  - partial solution to the problem of adapting to load changes
  - ensures that messages will be delivered in the order in which they were sent

Routing Strategies II

- dynamic routing: the path used to send a message form site A to site B is chosen only when a message is sent
  - usually a site sends a message to another site on the link least used at that particular time (load balancing)
  - adapts to load changes by avoiding routing messages on heavily used paths
  - messages may arrive out of order; this problem can be remedied by appending a sequence number to each message

Connection Strategies I

- circuit switching: a permanent physical link is established for the duration of the communication (i.e., telephone system)
- message switching: a temporary link is established for the duration of one message transfer (i.e., post-office mailing system)
- packet switching: messages of variable length are divided into fixed-length packets which are sent to the destination
  - each packet may take a different path through the network
  - the packets must be reassembled into messages as they arrive

Connection Strategies II

- circuit switching requires setup time, but incurs less overhead for shipping each message, and may waste network bandwidth
- message and packet switching require less setup time, but incur more overhead per message
- with circuit switching the sender and receiver can use any bit rate, format, or framing method
- with packet switching the carrier determines the basic parameters (“road vs. railroad”).

Shared Media

- TDM — Time Division Multiplexing
- FDM — Frequency Division Multiplexing
- CDMA — Code Division Multiple Access

Contention I

- CSMA/CD — Carrier Sense with Multiple Access / Collision Detection
  - a site determines whether another message is currently being transmitted over that link
  - if two or more sites begin transmitting at exactly the same time, then they will register a CD and will stop transmitting
  - when the system is very busy, many collisions may occur, and thus performance may be degraded
  - CSMA/CD is used successfully in Ethernet

Contention II

- token passing
  - a unique message type, known as a token, continuously circulates in the system (usually a ring structure)
  - a site that wants to transmit information must wait until the token arrives
  - when the site completes its round of message passing, it retransmits the token
  - used in IBM TokenRing and Apollo networks (obsolete)

Layered Networks

- “layer” (or “level”) structure to reduce complexity
- layer n on host A “talks” to layer n on host B
  - rules and conventions are referred to as “layer n protocol”
  - hosts A and B are often called “peers”
  - no direct “conversation”: data are passed to lower layers on sender, signal is transferred over the physical medium, then data are passed to successive upper layers on receiver
  - inter-layer interfaces
  - protocol stack and headers

Services And Protocols

- connection-oriented and connectionless services
- service primitives
- services and protocols
  - service is a collection of primitives
  - protocol is a set of format rules — implementation

Layering Principles

- each layer represents a different level of abstraction
- each layer performs a well-defined function
- an eye on standardization
- information flow across the interface boundaries should be minimized
  - a general rule for interface design
- optimal number of layers
  - large enough in order not to bundle distinct functions together in the same layer
  - small enough so that the architecture does not become unwieldy
- reference models: OSI and TCP/IP

OSI Reference Model I

- physical layer — handles the mechanical and electrical details of the physical transmission of a bit stream
- data-link layer — handles the frames, or fixed-length parts of packets, including any error detection and recovery that occurred in the physical layer
- network layer — provides connections and routes packets in the communication network, including handling the address of outgoing packets, decoding the address of incoming packets, and maintaining routing information for proper response to changing load levels

OSI Reference Model II

- transport layer — responsible for low-level network access and for message transfer between clients, including partitioning messages into packets, maintaining packet order, controlling flow, and generating physical addresses
- session layer — implements sessions, or process-to-process communications protocols
- presentation layer — resolves the differences in formats among the various sites in the network, including character conversions, and half duplex/full duplex (echoing)
- application layer — interacts directly with the users, deals with file transfer, remote login protocols, email, schemas for distributed databases

TCP/IP Reference Model I

- physical layer ("Layer 1")
- data-link layer ("Layer 2", "host-to-network layer", "great void")
- internet layer ("Layer 3")
  - IP — internet protocol
  - major function — routing
  - similar to OSI's network layer
- transport layer ("Layer 4")
  - TCP — transmission control protocol
    - reliable, connection-oriented
  - UDP — user datagram protocol
    - unreliable, connectionless

TCP/IP Reference Model II

- no session or presentation layers
  - no need perceived
- application layer (“Layer 5”)
  - telnet
  - FTP
  - HTTP
  - SMTP
  - SNMP
  - DNS
  - etc.

OSI vs. TCP/IP Comparison I

- both based on layering and the concept of protocol stack
- the functionality of the layers is roughly similar
  - up to and including the transport layer — end-to-end network-independent transport service
  - above transport layer — application-specific, user-oriented
- OSI contribution — service, interface, protocol abstractions
  - Protocols in OSI are better hidden — a main purpose of layering
- OSI was devised before protocols — generality

OSI vs. TCP/IP Comparison II

- in the case of TCP/IP the protocols came first, the model is a description of the existing system
  - poorly suited to describing any protocol stack other than TCP/IP
- OSI has support for connection-based and connectionless communications in the network layer, only connection-based in the transport layer (visible to the users)
- TCP/IP has only connectionless support in the internet layer, gives users a choice via dual support in the transport layer
- OSI model and TCP/IP protocols have become very popular

The TCP/IP World

- the Internet — a collection of networks
  - different owners
  - different infrastructure
  - different (internal) protocols
  - different topology
  - different performance
- IP addresses and the notion of subnets
- IP — routing packets around
  - routing tables
  - gateways
  - link-state protocol
- address resolution — ARP

Quality Of Service

- the Internet is a “best effort” network
- sometimes guarantees are needed
  - all packets reach the destination
  - all packets arrive in order
  - latency restrictions
  - jitter
- example: VoIP
  - latency above $150\,\mathrm{ms}$ — poor quality
    - $200,000\,\mathrm{km}$ at $2 \times 10^{10}\,\mathrm{cm}/\,\mathrm{s}$ — $50\,\mathrm{ms}$ delay
    - packetization, buffering, etc. end-to-end must satisfy the constraint
  - can lose a few per cent of packets
  - how large should the buffers be?

OS Network Interfaces I

- sockets — the common API in UNIX (POSIX)
  - socket address structure (`struct sockaddr_in`)
    - address family (.e.g., `AF_INET`)
    - IP address (32-bit for IPv4)
    - port number (16-bit)
    - a length member added for OSI support
  - Well known numbers
    - /etc/services
- in Windows — `winsock`

OS Network Interfaces II

- TCP sockets
  - `socket()` — create a socket
  - `bind()` — assign a local protocol address
    - can assign an IP address, but rarely done
  - `listen()` — server “listening” to incoming requests
  - `connect()` — client connects to a server
    - normally does not call `bind()`
  - `accept()` — returns a “connected socket”
    - a server normally creates one “listening socket” and a “connected socket” for each client connection that was `accept()`-ed
- need to be careful about network and host byte order

Typical Server Program Flow

```
int listenfd, connfd;
struct sockaddr_in servaddr;

listenfd = socket(AF_INET,SOCK_STREAM,0);

servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(<port_number>);

bind(listenfd,(SA*)&servaddr,sizeof(servaddr));

listen(listenfd,LISTENQ);

while(1) {
        connfd = accept(listenfd,(SA*)NULL,NULL);
        <write(connfd,data,datalen);>
        close(connfd);
}
```

Typical Client Program Flow

```
int sockfd;
struct sockaddr_in servaddr;
struct in_addr **addr;

struct hostent host = gethostbyname(<hostname>);
struct servent serv = getservbyname(<servname>,<protocol>);

for (addr = host->h_addr_list; addr != NULL; addr++) {
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    bzero(&servaddr,sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = serv->s_port;
    memcpy(&servaddr.sin_addr,*addr,sizeof(struct in_addr));
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) == 0)
       break; /* SUCCESS */
    close(sockfd);
}
if (*addr == NULL) exit(EXIT_FAILURE); /* unable to connect */
/* use the socket here */
```