28 Networks and Communication Protocols

Trend in computer systems today: personal computers (PC, WS).

Reasons why:

- Cost: avoid huge initial investment in timesharing system.
- Performance: personal computers provide almost same power as previous timesharing systems.
- Reliability: if one person's computer crashes, doesn't affect other people.

Problem: communication and cooperation are difficult.

- How do people on the same project share files?
- How does new software get distributed to all users?
- How is electronic mail handled?

Solution: tie machines together with networks, develop message *protocols* that allow communication and cooperation again.

Goal: get same effect as with timesharing, except lots of CPU power.

Examples of networks:

- ARPAnet: 1st widely-used network, developed early 70's, replaced in early 90's. Connected together large timesharing systems all over the country using leased phone lines. Provided mail, file transfer, remote login.
- Usenet: Developed late 70's, early 80's. Unix systems phone each other up to send mail and transfer files.
- Local area networks: Developed early 80's to hook together personal computers. Most popular interconnection for LANs is Ethernet. LANs are used very differently than wide-area networks.
- Internet: ties together many existing networks.

Network hardware:

- Point-to-point links (used by most early networks, current long-haul networks). Examples are leased phone lines (56kbits/sec-ARPAnet), leased trunks (1Mbit/sec—2.4Gbit/sec), ISDN (digital phone lines), analog phone line plus modem, cable modems. Different styles:
  - Fully connected: every site can talk directly to any other site (e.g. Usenet).
  - Partially connected: star and ring are most popular. Intermediate nodes must forward messages.
- Multi-access bus (used by most LANs today). A single cable or group of cables connects many machines together. Best examples are Ethernet (one wire), FDDI, satellite links.
- Networks are usually characterized in terms of two things:
  - Latency: the minimum time to get the minimum amount of information between two sites.
  - Bandwidth: once information is flowing, how many bits per second can be transmitted (i.e. the marginal cost per bit).

  Latency and bandwidth are often used to characterize higher-level protocols too.

Protocols: these are the key to networks. A protocol is just an agreement between the parties on the network about how information will be transmitted between them. There are many different protocols to do different things (e.g. mail, file transfer, remote login). Typically, protocols are built up in layers.

Lowest protocol layer: physical layer. Determines the physical mechanisms (electrical/optical) for transmitting bits: voltages, delays, currents, etc.

Data Link protocol layer: how to get frames between two directly-connected components.

Broadcast networks: single shared communication medium, no central controller to allocate access to it.

- Simplest scheme is the Aloha mechanism: just broadcast blindly, use recovery protocols if packet doesn't get through. This system has stability problems: can't get more than 18% utilization of channel, and system completely falls apart under heavy loads.
- Ethernet adds two things. First is carrier sense: listen before broadcasting, defer until channel is clear, then broadcast.
- Also listen while broadcasting. Collision can still happen if two stations start up at exactly the same time. If collision detected, jam network so that everyone will know about collision (don't waste time transmitting junk). Then wait a "random" interval, retry. If repeated collisions, wait longer and longer intervals.
- This is called CSMA/CD (carrier sense multiple access, with collision detection).

Problem with Ethernet:

- Reliability: if any station jams network, nobody can do anything, can't even figure out who's doing it.
- Fairness: there's no guarantee against starvation. People with real-time needs don't like this, but it hasn't been a problem in practice.

Ring networks: an additional protocol built on top of a ring-structured set of point-to-point links.

- Normally, an electronic token (special packet) circulates at high speed around the ring. If a station doesn't have anything to broadcast, it just retransmits everything it receives.
- When ready to broadcast, a station waits until the token passes by. Instead of retransmitting the token, send packet instead.
- When packet has been transmitted, put token back on ring for next station to use.
- Packet loops all the way around, gets swallowed by sender when it comes back again (recognize self as destination).

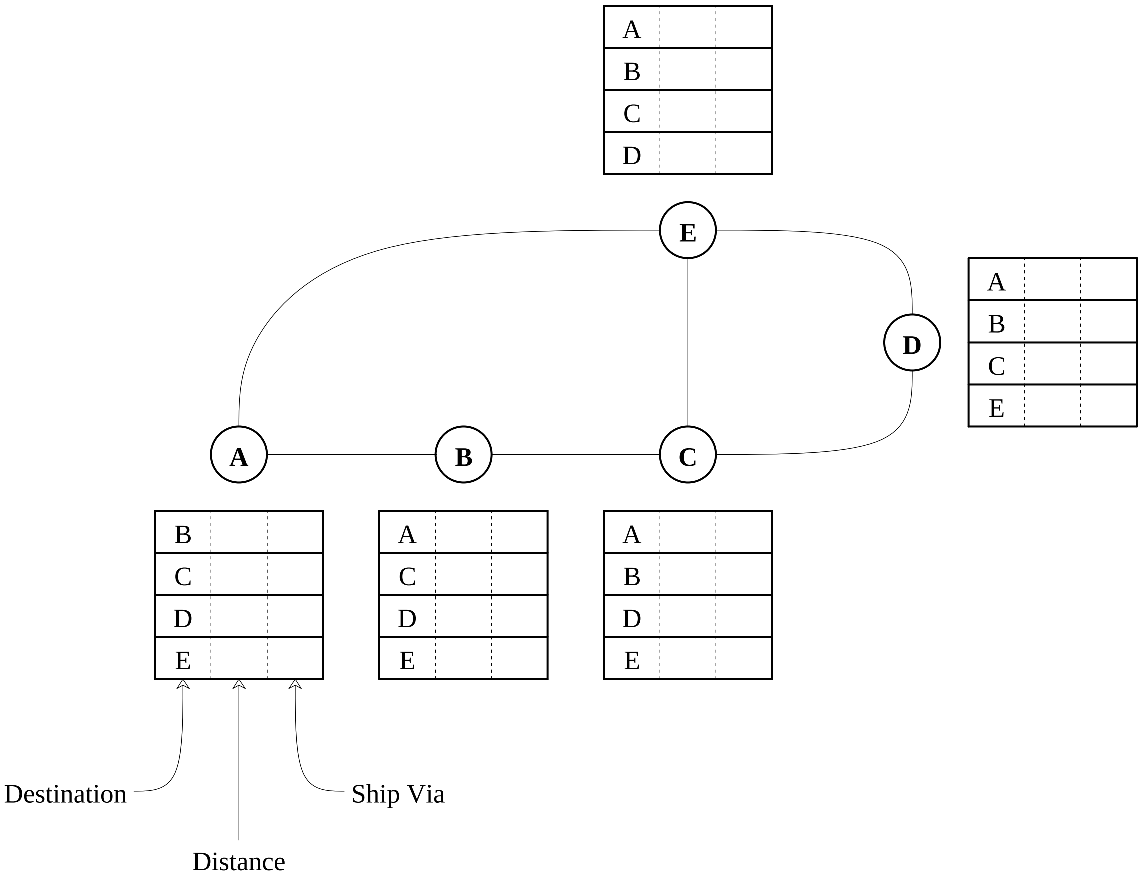
Problem with ring system: if any station dies, token can't circulate so ring dies.

At this point, packets can be transmitted from one host on a local network to another. Next step: get packets from any machine in the *internetwork* to any other machine. This is called the *network layer* of protocols in the ISO terminology.

- Packets must be forwarded from machine to machine until they reach the destination. Forwarding machines are called *routers*.
- Distinguish between circuit switching and message switching (also called packet switching).
- Names vs. addresses vs. routes:
  - Name: a symbolic term for something: "Alice", or "owlnet". Good for people to remember.
  - Address: where the thing is: in an internetwork situation, usually consists of the number of the network, the number of the host on the network, and perhaps also the id of a process or mailbox within a host.
  - Route: directions for how to get there from here (a sequence of hosts and links to pass through to reach the destination).

  Compare to names, addresses, and routes for humans.
- Sometimes the sender has to provide the route, e.g. in UUCP: hplabs!hp-pcd!hpcvc0!cliff. All each machine has to be able to do is remember its neighbors and forward messages. This is clumsy for users.
- It's better if the hosts of the internetwork can figure out the routing stuff for themselves. This involves a special protocol between the hosts to build routing tables. E.g. in ARPAnet, hosts send messages to nearest neighbors, build up tables of most direct paths from each host to each other host (fewest hops).
- In LANs, only gateways have to worry about routing: all the other hosts just ship packets to gateway unless for host on local net.

Building Routing Tables



Unfortunately, in the above scheme, packets can get lost in a number of ways:

- Transmission errors.
- A host has all its packet buffers full so it has no place to put another incoming packet:
  - Can happen at intermediate host if packets are arriving on a fast network but have to be forwarded onto a much slower network. This is called *network congestion*.
  - Can happen at destination if user process can't work fast enough to process all the packets as they arrive.

Another problem is that packets can arrive out of order: if some hosts suddenly go down, or if routing tables change, packets might wander off into the network and come back much later. Most protocols include a *time-to-live* mechanism: after a certain time, packets are killed so that they don't wander endlessly.

The basic network protocols described above are sometimes called *datagram* protocols. They are used to deliver individual packets, and the packets are not guaranteed to get through or to arrive in any particular order. This is useful for some applications, but not very many.

Most applications would like guarantees about delivery and order. To do this, the sender and receiver must remember *state* about what has been happening. This is called a *connection*, and the protocols to implement it are called *virtual circuit* or *transport* protocols.

- Simple acknowledgement-based protocol:
  - Store a serial number in each packet. Sender assigns serial numbers, increments for each packet.
  - Whenever receiver gets a packet, it sends back an *acknowledgement* packet.
  - Sender waits for acknowledgement before sending next packet (must also save old packet!).
  - If sender doesn't receive acknowledgement within a reasonable time, it assumes that the packet got lost and retransmits it.
  - Retransmission could result in receiver getting two packets with same serial number: it checks serial numbers and throws away duplicates and out-of-order packets.
- Problems with simple ack scheme: requires one ack for every data packet, may be slow in high-latency situations like satellite links (must wait for ack before sending next packet).
- Most virtual-circuit protocols are more elaborate: sender can transmit several packets before waiting for acks, receiver sends a single ack for several received packets. Sender and receiver must negotiate about how far ahead the sender can send: otherwise the receiver might run out of buffer space and have to discard packets. This is called the *flow control* problem.
- Solution: sliding-window protocols.

- No matter what the virtual circuit mechanism, setting up the connection is complex and time-consuming. It's tricky to get two hosts to agree to communicate with each other and get their state initialized correctly.

Another problem: different machines may use different data formats (e.g. floating-point numbers may have different bit arrangements). It may be necessary to do conversion whenever information is shipped. The protocols to do this conversion are called the *presentation* layer in the ISO classification.

Lastly, there is the *application* layer, which uses virtual circuits to perform various functions like mail, file transfer, remote login, etc. Typically, various machines on the network contain application programs to provide particular *services*: mail delivery, time-of-day, file access, name-to-address binding, etc. These machines are called *servers*.

Peer-to-peer protocols: movement through levels, header additions and removals.

The mechanisms described above form the basis for tying together distributed systems. Until the mid 80s, though, they've only been used for loose coupling:

- Each machine is completely autonomous: separate accounting, separate file system, separate password file, etc.
- Can send mail between machines.
- Can transfer files between machines (but only with special commands).
- Can execute commands remotely.
- Can login remotely.

Loose coupling like this is OK for a network with only a few machines spread all over the country, but not for a high-performance LAN where every user has a private machine. COMP 520, the course on distributed systems, focuses on principles and issues of providing tighter integration over a network.