Course Snapshot

We have covered all the fundamental OS components:

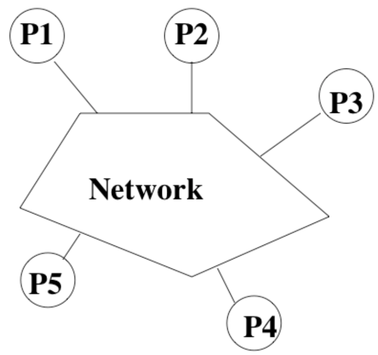
- Architecture and OS interactions
- Processes and threads
- Synchronization and deadlocks
- Memory management
- File systems and I/O

Remaining Topics

- Distributed Systems
  - Networking basics
  - Distributed services (email, www, telnet)
  - Distributed operating systems
  - Distributed file systems
- Protection
- Cloud Computing

Distributed Systems

- **Distributed system:** a set of physically separate processors connected by one or more communication links



- Nearly all systems today are distributed in some way
  - Email, file servers, network printers, remote backup, world wide web

Parallel versus Distributed Systems

- **Tightly-coupled systems:** "parallel processing"
  - Processors share clock, memory, and run one OS
  - Frequent communication

- **Loosely-coupled systems:** "distributed computing"
  - Each processor has its own memory
  - Each processor runs an independent OS
  - Communication should be less frequent

Advantages of Distributed Systems

- **Resource sharing:**
  - Resources need not be replicated at each processor (e.g., shared files)
  - Expensive (scarce) resources can be shared (e.g., printers)
  - Each processor can present the same environment to the user (e.g., keeping files on a file server)

- **Computational speedup:**
  - *n* processors potentially gives you *n* times the computational power
  - Problems must be decomposable into subproblems
  - Coordination and communication between cooperating processes (synchronization, exchange of results) is needed.

Advantages of Distributed Systems

- **Reliability:**
  - Replication of resources yields fault tolerance.
  - For example, if one node crashes, the user can work on another.
  - Performance will degrade, but system remains operational.
  - However, if some component of the system is centralized, a single point of failure may result
  - **Example:** If an Edlab workstation crashes, you can use another workstation. If the file server crashes, none of the workstations are useful.

- **Communication:**
  - Users/processes on different systems can communicate.
  - For example, mail, transaction processing systems like airlines, and banks, WWW.

Distributed Systems

- Modern work environments are distributed => operating systems need to be distributed
- What do we need to consider when building these systems?
  - Communication and networks
  - Transparency (how visible is the distribution?)
  - Security
  - Reliability
  - Performance and scalability
  - Programming models

Distributed System Design

What gets harder when we move from a stand alone system to a distributed environment?

- resource sharing
- timing (e.g., synchronization)
- critical sections
- deadlock detection and recovery
- failure recovery

Networks

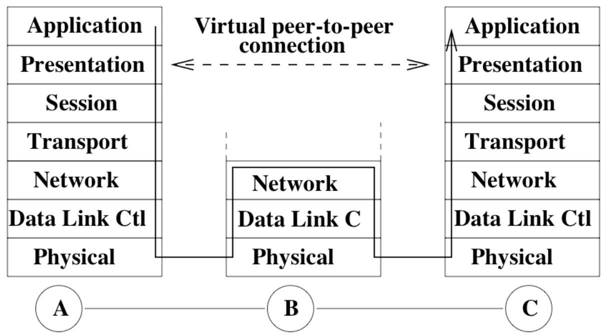
- Networks are usually concerned with providing efficient, correct, and robust message passing between two separate nodes.
- **Local Area Network (LAN)** usually connects nodes in a single building and needs to be fast and reliable (e.g., Ethernet).
  - **Media:** twisted-pair, coaxial cable, fiber optics
  - **Typical bandwidth:** 10-100-1000 Mb/s (10Gb/s now available)
- **Wide Area Network (WAN)** connects nodes across the state, country, or planet.
  - WANs are typically slower and less reliable than LAN (for example, Internet).
  - **Media:** telephone lines (T1 service), microwave links, satellite channels, cellular networks (3G/4G/LTE)
  - **Typical bandwidth:** 1.544 Mb/s (T1), 45 Mb/s (T3)

Principles of Network Communication

- Data sent into the network is chopped into “packets”, the network's basic transmission unit.
- Packets are sent through the network.
- Routers at the switching points control the packet flow.
- **Analogy:** cars/road/road signs, lights - packets/network/routers
- Shared resources can lead to contention (traffic jams).

Communication Protocols

- Protocol: a set of rules for communication that are agreed to by all parties
- Protocol stack: networking software is structured into layers
  - Each layer N, provides a service to layer N+1, by using its own layer N procedures and the interface to the N-1 layer.
  - Example: International Standards Organization / Open Systems Interconnect (ISO/OSI)

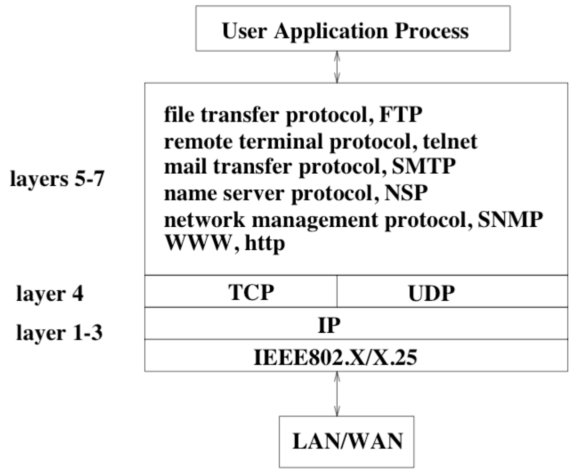


ISO Network Protocol Stack

- **Application layer:** applications that use the net, e.g., mail, netscape, X-services, ftp, telnet, provide a UI
- **Presentation layer:** data format conversion, e.g., big/little endian integer format)
- **Session layer:** implements the communication strategy, such as RPC. Provided by libraries.
- **Transport layer:** reliable end-to-end communication between any set of nodes. Provided by OS.
- **Network layer:** routing and congestion control. Usually implemented in OS.
- **Data Link Control layer:** reliable point-to-point communication of packets over an unreliable channel. Sometimes implemented in hardware, sometimes in software (PPP).
- **Physical layer:** electrical/optical signaling across a “wire”. Deals with timing issues. Implemented in hardware.

TCP/IP Protocol Stack

- Most Internet sites use TCP/IP - Transmission Control Protocol/Internet Protocol.
  - It has fewer layers than ISO to increase efficiency.
  - Consists of a suite of protocols: UDP, TCP, IP...
  - TCP is a **reliable** protocol -- packets are received in the order they are sent
  - UDP (user datagram protocol) an **unreliable** protocol (no guarantee of delivery).



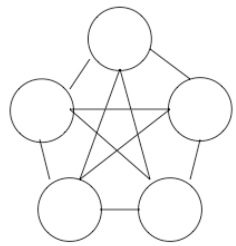
Packet

- Each message is chopped into packets.
  - Each packet contains all the information needed to recreate the original message.
  - For example, packets may arrive out of order and the destination node must be able to put them back into order.
  - Ethernet Packet Contents

bytes		
7	preamble - start of packet	fixed pattern so packet start is recognizable
1	start of frame delimiter	
6	destination address	
6	source address	
2	length of data section	
0-1500	data	
0-46	pad(optional)	packet must be > 63 bytes long
4	frame checksum	

  - The data segment of the packet contains headers for higher protocol layers and actual application data

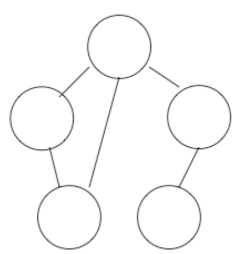
Point-to-Point Network Topologies



Fully Connected

- **Fully connected:** all nodes connected to all other nodes
  - Each message takes only a single "hop", i.e., goes directly to the destination without going through any other node
  - Failure of any one node does not affect communication between other nodes
  - Expensive, especially with lots of nodes, not practical for WANs

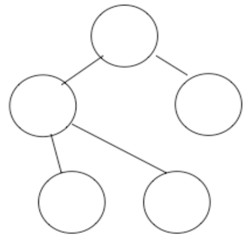
Point-to-Point Network Topologies



Partially Connected

- **Partially connected:** links between some, but not all nodes
  - Less expensive, but less tolerant to failures. A single failure can partition the network.
  - Sending a message to a node may have to go through several other nodes => need routing algorithms.
  - WANs typically use this structure.

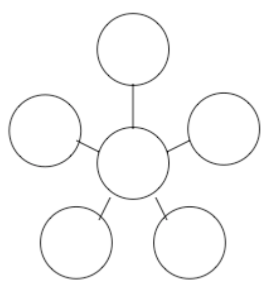
Point-to-Point Network Topologies



Tree Structured

- **Tree structured:** network hierarchy
  - All messages between direct descendants are fast, but messages between "cousins" must go up to a common ancestor and then back down.
  - Some corporate networks use this topology, since it matches a hierarchical world view...
  - Not tolerant of failures. If any interior node fails, the network is partitioned.

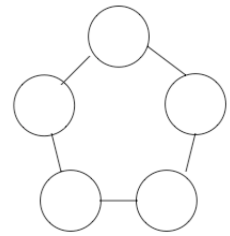
Point-to-Point Network Topologies



Star

- **Star:** all nodes connect to a single centralized node
  - The central site is generally dedicated to network traffic.
  - Each message takes only two hops.
  - If one piece of hardware fails, that disconnects the entire network.
  - Inexpensive, and sometimes used for LAN

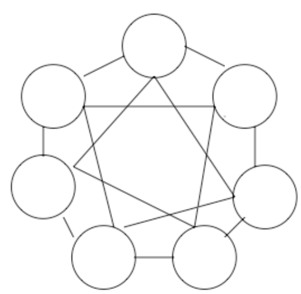
Ring Network Topologies



Ring

- **One directional ring** - nodes can only send in one direction.
  - Given *n* nodes, message may need to go *n-1* hops.
  - Inexpensive, but one failure partitions the network.
- **Bi-directional ring** - nodes can send in either direction.
  - With *n* nodes, a message needs to go at at most *n*/2 hops.
  - Inexpensive, tolerates a single failure by increasing message hops. Two failures partition the network.

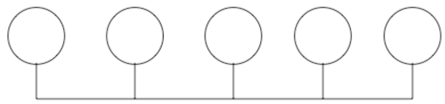
Ring Network Topologies



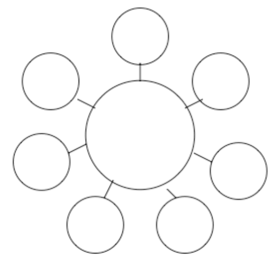
Doubly Linked Ring

- **Doubly connected ring** nodes connected to neighbors and one away neighbors
  - A message takes at most *n*/4 hops.
  - More expensive, but more tolerant of failures.

Bus Network Topologies



Linear Bus



Ring Bus

- **Bus** nodes connect to a common network
- **Linear bus** - single shared link
  - Nodes connect directly to each other using multiaccess bus technology.
  - Inexpensive (linear in the number of nodes) and tolerant of node failures.
  - Ethernet LAN uses this structure.
- **Ring bus** - single shared circular link
  - Same technology and tradeoffs as a linear bus.

Resource Sharing

There are many mechanisms for sharing (hardware, software, data) resources.

- **Data Migration:** moving the data around
- **Computation Migration:** move the computation to the data
- **Job Migration**: moving the job (computation and data) or part of the job

=> The fundamental tradeoff in resource sharing is to complete user instructions as fast and as cheaply as possible. (Fast and cheap are usually incompatible.)

If communication is cheap: use all resources

If computation is slow/expensive: local processing

Reality is somewhere in between

Client/Server Model

- One of the most common models for structuring distributed computation is by using the *client/server* paradigm.
  - A *server* is a process or collection of processes that provide a service, e.g., name service, file service, database service, etc.
  - The server may exist on one or more nodes.
  - A *client* is a program that uses the service.
  - A client first binds to the server, i.e., locates it in the network and establishes a connection.
  - The client then sends the server a request to perform some action. The server sends back a response.
  - RPC is one common way this structure is implemented.

Remote Procedure Call

Basic idea:

- Servers export procedures for some set of clients to call.
- To use the server, the client does a procedure call.
- OS manages the communication.

Remote Procedure Call: Implementation Issues

For each procedure on which we want to support RPC:

- The RPC mechanism uses the procedure *signature* (number and type of arguments and return value)
  - to generate a client stub that bundles up the RPC arguments and sends it off to the server, and
  - to generate the server stub that unpacks the message, and makes the procedure call.

Remote Procedure Call: Implementation Issues

Client Stub:

build message
send message
wait for response
unpack reply
return result

Server Stub:

create threads
loop
    wait for a command
    unpack request parameters
    call procedure with thread
    build reply with result(s)
    send reply
end loop

Comparison between RPC and a regular procedure call

- Name of procedure
- Parameters
- Result
- Return address

Remote Procedure Call

- How does the client know the location of the server?
  - The binding can be static - fixed at compile time.
  - Or the binding can be dynamic - fixed at runtime.
- In most RPC systems, dynamic binding is performed using a name server.
  - When the server starts up, it exports its interface and identifies itself to a network name server
  - The client, before issuing any calls, asks the name service for the location of a server whose name it knows and then establishes a connection with the server.

Example: Remote Method Invocation (RMI) in Java

- **Naming:** class that provides the calls to communicate with the remote object registry (i.e., name server)
  - public static void bind(String name, Remote obj) - Binds a (remote) server to a name.
  - public static Remote lookup(String name) - Returns the server object that corresponds to a name.
- **UnicastRemoteObject:** a remote object over TCP, exports the interface automatically when the server object is constructed
- Java provides the following tools:
  - **rmiregistry:** server-side name server
  - **rmic:** given the server interface, generates client and server stubs that create and interpret packets

Example: RMI Remote Interface

Declare the methods that the server provides:

```
// All servers must extend the Remote interface.
public interface SearchEngine extends java.rmi.Remote {

    // Any remote method might throw RemoteException.
    // Indicates network failure.
    public String searchFor(String queryString) throws java.rmi.RemoteException;

}
```


Example: RMI Server

```
public class MySearchEngine extends UnicastRemoteObject implements SearchEngine {

  public MySearchEngine() throws RemoteException {
    // The superclass constructor exports the interface and gets a port
     super();
  }

  public String searchFor(String queryString) throws RemoteException {
     String resultPage = ... ; // do the actual work
     return  resultPage;
  }

  public static void main(String args[]) {
     // Construct the server object.
    MySearchEngine obj  = new MySearchEngine();

     // Register the server with the (local) name server.
     Naming.bind("SearchServer", obj);
  }
 }
```

Example: RMI Client

```
public class SearchClient  {

    public SearchClient(String remoteHost, String searchQuery) {
      try {

          // Looks up the (remote) search engine using the name server and the previously bound name
          MySearchEngine remoteSearchEngine = (MySearchEngine) Naming.lookup(
                  "//" + remoteHost + "/SearchServer");

          // Calls the remote object
          String resultPage = remoteSearchEngine.searchFor(searchQuery);

    } catch (RemoteException e) {
      System.out.println("Network error");
    }
  }
}
```


Summary

- Virtually all computer systems contain distributed components
- Networks hook them together
- Networks make tradeoffs between speed, reliability, and expense