Communication

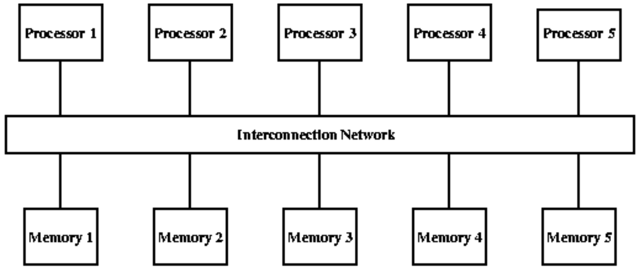
527950-1
Fall 2019
10/17/2019
Kyoung Shin Park
Applied Computer Engineering
Dankook University

Outline

- Distributed system
  - A set of interconnected CPUs communicating by messages over a network.
  - It is important to consider the many different means of communication possible.
- Interconnection Networks
- Network Protocols
- Inter-Process Communication (IPC)
  - Message Passing
  - Sockets
  - Client/Server Model
  - Remote Procedure Call (RPC)

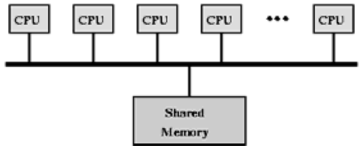
Interconnection Networks

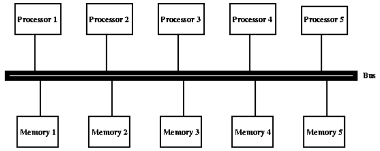
- There are many ways in which the processors and memories of a distributed system can be interconnected.
- In most cases, the interconnection networks can be used for symmetric multiprocessors (SMPs)
  - **Bus, ring, crossbar, hypercube, shuffle-exchange**



Interconnection Network: Bus

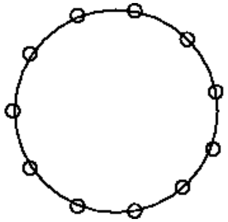
- A **simple, static** interconnection scheme used by early shared memory multiprocessors
- There can be a single large shared memory or there can be a number of shared memory units.
- Bus is **shared by all attached processors and memories**
- **No routing is required**
- All data transmissions occur on bus (i.e., The bus can become a bottleneck)
- The scalability is limited by the bandwidth of the bus

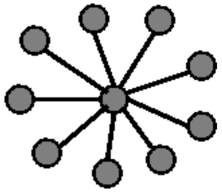




Interconnection Network: Ring

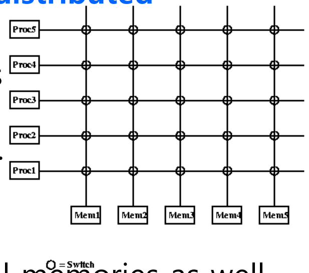
- One simple interconnection scheme **connects processors in a ring**.
- The information flow between nodes may travel in just **one direction** or in **both directions**.
- This means the distance (number of hops) between any two nodes varies.
- A related interconnection scheme is called a **star formation**.





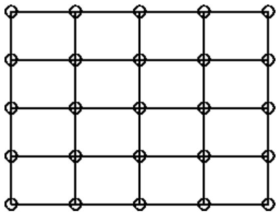
Interconnection Network: Crossbar

- A interconnection scheme used by **distributed memory multiprocessors**.
- **This 2D array of switches** connects the processors and memory units permitting **dynamic configurations**.
- There are **many paths from one processor to another**.
- And all processors have access to all memories as well as to each other.
- There are ***n*** memory modules times ***n*** processors. This means ***$O(n^2)$*** switches are required (expensive).
- The scalability is limited by the number of switches that is affordable.



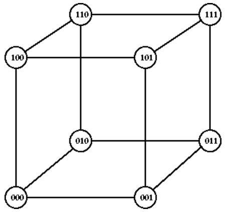
Interconnection Network: Mesh or Grid

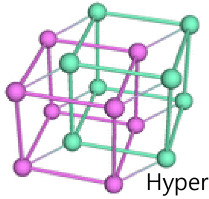
- A similar interconnection scheme used by **processors in a grid**.
- This 2D array connects **n x m processors** in a grid.
- There are **many paths from one processor to another**.
- There are ***n*** x m processors. This means ***$O(nm)$*** **processors** in a grid.



Interconnection Network: Hypercube

- Another **static** interconnection scheme is hypercube
- A *k*-dimensional hypercube contains ***2^k*** **processors** (nodes)
- Each processing node contains a switch
- Distance between nodes easy to compute

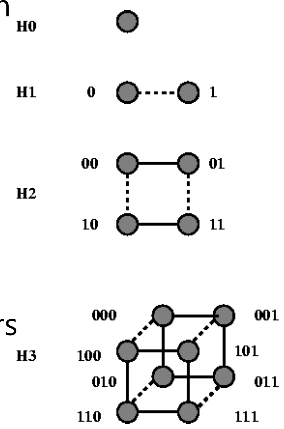




Hypercube of dimension 4

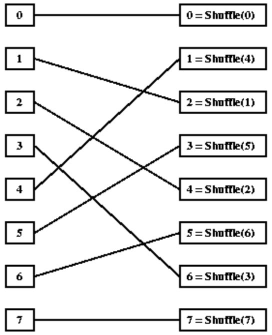
Interconnection Network: Hypercube

- This example is hypercube of dimension 0 through 3.
- The dotted edges shown where the hypercube is being extending from the next lower dimension hypercube
- The nodes are numbered from 0 to $2^k - 1$, usually in binary
- Two nodes are separated by a single link if the binary values of their numbers differ by one bit.
  - E.g. Node 000 is only one hope from node 010 and node 000 is two hops from node 011 (by path 000->001->011 or 000->010->011)



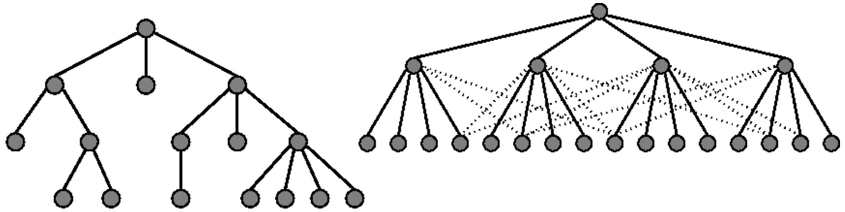
Interconnection Network: Shuffle-Exchange

- Permits **dynamic reconfigurations**
- Each processing node contains a switch.
- The switches have four modes of operation
  - That are controlled by switch control bits.
- This makes the routing quite simple
  - As the address bits are used as control bits



Interconnection Network: Tree

- Another interconnection scheme connects processors in a **tree formation**.
- This tree formation may connect any number of processors.
- Notice that an interconnection tree does not need to be a binary tree.
- In fact, the Connection Machine 5 used what they called a fat tree



Network Protocols

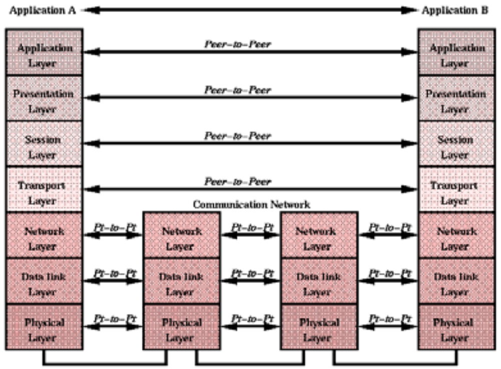
- In a distributed system, **the network** is the medium that connects the computers.
- On the lowest layer, it takes the form of a physical transmission medium. In current use are copper wire, optical fiber, and wireless media.
- Above the physical layer, we have a hierarchy of protocols.
- A protocol is an agreed-upon set of rules, which describe actions or sequences of actions that initiate and control the transmission of data along the physical connections.
- In the protocol hierarchy, each layer provides a richer functionality than the layer below it, and each layer implements its functionality on the basis of the lower layer's functionality. A variety of protocol hierarchies and individual protocols are in current use.

Network Protocols

- Protocol is a set of conventions for formatting data in an electronic communications system
- A method for
  - Establishing a connection between two sites
  - Sending a communication over the connection
  - Acknowledging receipt of message
  - Terminating the connection
- Examples: ISO/OSI, TCP/IP, SMTP

ISO/OSI Protocol

- Probably most popular network protocol model
- Implementation often takes efficiency-related shortcuts
- Includes **7 layers**, grouped into 3 types
  - **Application**
  - **Operating system**
  - **Communication service**



ISO/OSI

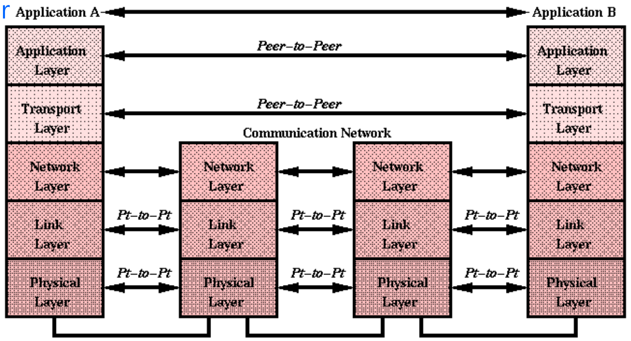
- Application Layer
  - High-level APIs
  - Resource sharing, remote file access, directory services and virtual terminals
- Presentation Layer
  - Translation of data between a networking service and an application
  - Character encoding, data compression and encryption/decryption
- Session Layer
  - Managing communication sessions
  - Continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes

ISO/OSI

- **Transport Layer**
  - Reliable transmission of data segments between points on a network
  - Segmentation, acknowledgement and multiplexing
- **Network Layer**
  - Structuring and managing a multi-node network
  - Addressing, routing and traffic control
- **Data Link Layer**
  - Reliable transmission of data frames between two nodes connected by a physical layer
- **Physical Layer**
  - Transmission and reception of raw bit streams over a physical medium

Internet Protocol

- **A 5-layer protocol**
  - Application Layer
  - Transport Layer
  - Network Interface
  - Link Layer
  - Physical Layer



Internet Protocol

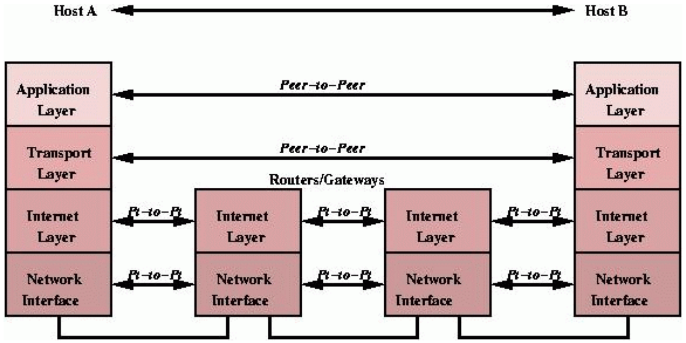
- **Application Layer**
  - Protocols used by most applications for providing user services or exchanging application data over the network connections
  - BGP. DHCP, DNS, FTP, HTTP, IMAP, LDAP, MGCP, NNTP, NTP, SSH, Telnet, TLS/SSL
- **Transport Layer**
  - Establishes basic data channels that applications use for task-specific data exchange.
  - End-to-end services that are independent of the structure of user data and the logistics of exchanging information for any particular specific purpose.
  - End-to-end message transfer independent of the underlying network, along with error control, segmentation, flow control, congestion control, and application addressing (port numbers).
  - TCP, UDP, DCCP, SCTP, RSVP

Internet Protocol

- **Internet Layer**
  - Responsibility of sending packets across potentially multiple networks
  - Internetworking requires sending data from the source network to the destination network. This process is called routing.[
  - IP (IPv4, IPv6), ICMP, IGMP
- **Link Layer**
  - Networking scope of the local network connection to which a host is attached.
  - The link layer is used to move packets between the Internet layer interfaces of two different hosts on the same link.
  - These perform data link functions such as adding a packet header to prepare it for transmission, then actually transmit the frame over a physical medium
  - ARP, NDP, PPP, MAC (Ethernet, DSL, ISDN, FDDI)

TCP/IP

- A **4-layer protocol** on top of hardware (physical layer)
  - **Application Layer**
  - **Transport Layer**
  - **Internet Layer (Network Layer)**
  - **Network Interface Layer (Data Link Layer)**



TCP/IP

Application Layer

- Message exchange between standard or user applications
- HTTP (Hypertext Transfer Protocol)
- FTP (File Transfer Protocol)
- SMTP (Simple Mail Transfer Protocol)
- Telnet

Transport Layer

- Functionalities for delivering data packets to a specific process on a remote computer
- TCP (Transmission Control Protocol)
- UDP (User Datagram Protocol)

TCP/IP

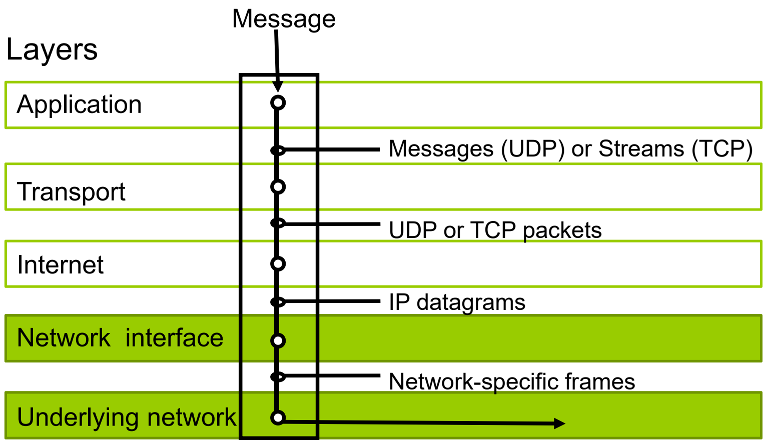
Network Layer (also known as Internet Layer)

- Organize or handle the movement of data on network
- IP (Internet Protocol) - a packet of data to be addressed to a remote computer and delivered
- ICMP (Internet Control Message Protocol)
- IGMP (Internet Group Management Protocol)

Data Link Layer (also known as Network Interface Layer)

- Device drivers in the OS and the network interface.
- Functionalities for transmission of signals representing a stream of data from one computer to another.
- ARP (Address Resolution Protocol)
- PPP (Point to Point Protocol)

TCP/IP



Message

Layers

Application

Messages (UDP) or Streams (TCP)

Transport

UDP or TCP packets

Internet

IP datagrams

Network interface

Network-specific frames

Underlying network

Inter-Process Communication (IPC)

Message Passing

- Primitive commands
- Blocking / Unblocking
- Synchronous / Asynchronous
- Buffered / Unbuffered
- MPI (Message Passing Interface) and PVM (Parallel Virtual Machine)

Client/Server Model

- Sockets
- Remote Procedure Call (RPC)

Message Passing Primitives

- **Message passing** is a form of communication between two processes.
- A physical copy of the message is sent from one process to the other.
- **Message passing primitive commands**
  - **SEND (msg, dest)**
  - **RECEIVE (src, buffer)**
- This is a low-level approach to IPC, and puts the burden of communication on the programmer
- Message passing is the basis of **MPI (Message Passing Interface)** and **PVM (Parallel Virtual Machine)**

Message Passing Primitives

- **Buffering methods**
  - **Standard method**
    - **User buffer -> Sender's kernel buffer -> Receiver's kernel buffer -> User buffer**
  - **Unbuffered method**
    - **User buffer -> User buffer** (The sender & receiver need to know if the buffer at the other end is in use or if it is static)
- Synchronous vs Asynchronous methods
  - **Synchronous method**
    - Two communicating processes rendezvous
    - No buffer is used
  - **Asynchronous method**
    - Just buffer the message
    - Problems
      - Need to allocate and deallocate buffers.
      - What is the buffer is full/empty? What if receiver's machine dies?

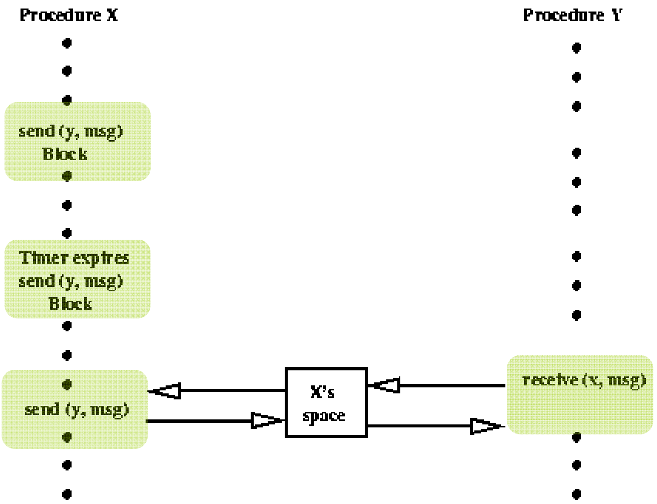
Blocking

- This is a form of **synchronous** communication
- The primitive commands wait for the message to be delivered
- That is, **the processes are blocked** from continuing to process
- The sending process must wait after send until an acknowledgement is made by the receiver
- The receiving process must wait for expected message from the sending process
- The receipt is determined
  - By polling common buffer
  - By interrupt

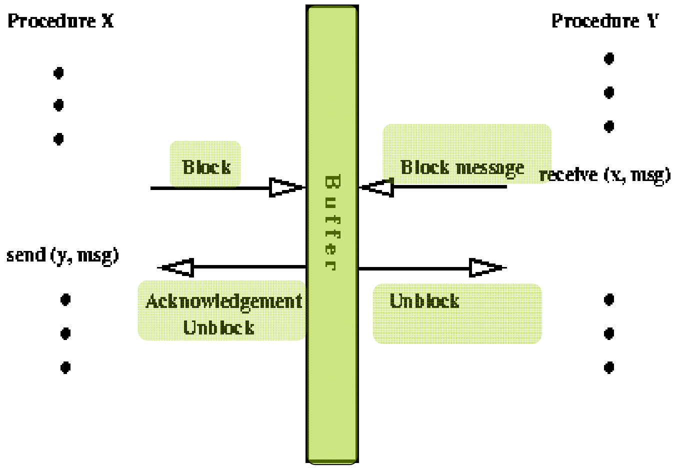
Non-Blocking

- This is a form of **asynchronous** communication
- The sender can continue as soon as the user buffer is copied from.
- The sending process may continue immediately after sending a message - **No wait is needed**
- The receiver signals that it wants to receive
- If nothing is available, it can check back periodically or it can wait for a signal
- The receiving process accepts and processes message - Then, it continues on
- Control
  - Buffer – receiver can tell if message still there
  - Interrupt

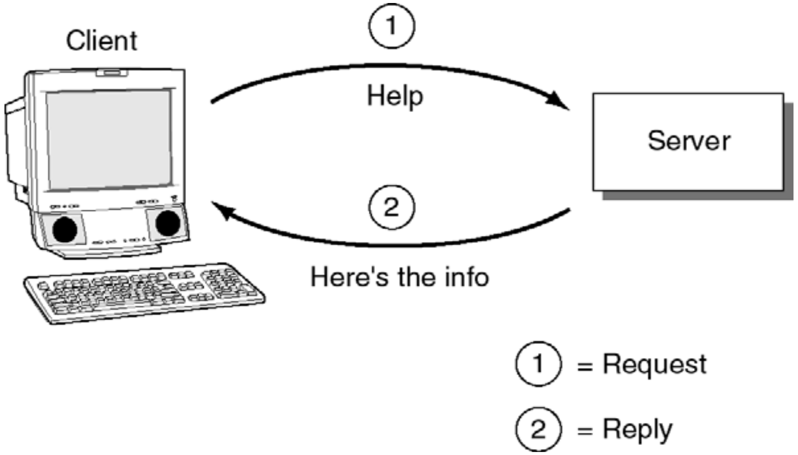
Blocking Send and Receive Primitives No Buffer



Blocking Send and Receive Primitives with Buffer



Client/Server Model



Client/Server Model

- "The most common paradigm of distributed computing at present. This paradigm describes an asymmetric relationship between two processes, of which one is the **client**, and the other is the **server**."
- "In the client/server paradigm, a server process offers a service that is used by the client process. Client and server typically run at different locations."

The BSD Sockets Architecture

- *When an application sends a packet, the host must make sure that it gets sent to the right destination, and when a host receives a packet, it must make sure that it is delivered to the correct application. To achieve these two tasks, most hosts on the Internet use the Berkeley Software Distribution (BSD) Sockets network architecture to keep track of applications and network connections.*
- This architecture first gained wide acceptance in the Unix operating system, but today, it is implemented on virtually all of the major commercial operating systems on the market. The WinSock library used on Microsoft Windows platforms is a derivative of the BSD interfaces.

Sockets & Ports

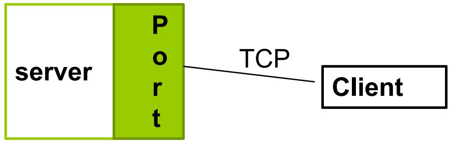
- **Socket**
  - A software representation of the endpoint to a communication channel
  - Can represent many different types of channels (i.e., **reliable/unreliable** communication, single/multiple destinations, etc)
  - IP address + **UDP/TCP** + port number
  - 131.120.1.13, UDP, 51
  - 131.120.1.13, TCP, 51
- **Port**
  - A specific numerical identifier for an individual application

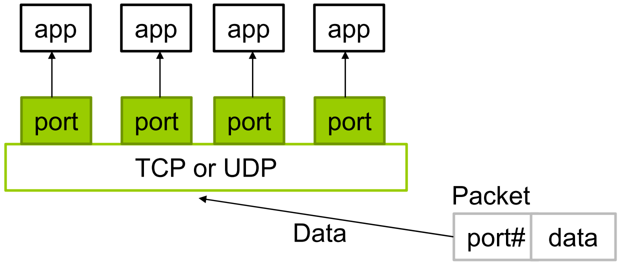
Sockets

- A socket identifies several pieces of information about a communication channel:
  - **Protocol**: How the operating systems exchange application data
  - **Destination host**: The destination host address(es) for packets sent on this socket
  - **Destination application ID or port**: Identifies the appropriate socket on the destination host
  - **Source host**: Identifies which host is sending the data
  - **Local application ID/port**: A 16 bit integer that identifies which application is sending data along this socket

Port

- The TCP and UDP protocols use *ports* to map incoming data to a particular *process* running on a computer.





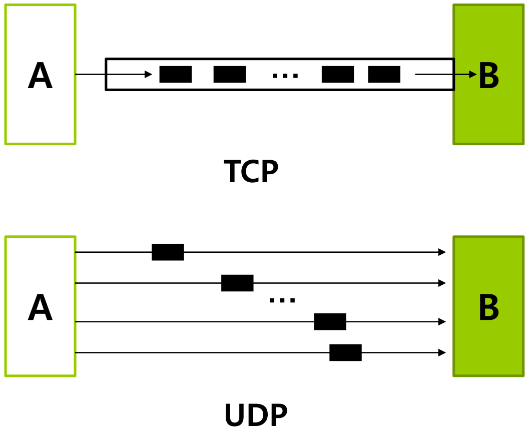
Port

- **Port** is represented by a positive (16-bit) integer value
- Port numbers 1 - 1024 are reserved for "well-known" applications/OS services
- 1025 - 10,000 are registered for certain "well-known" protocols
- Example:
  - port 21 is reserved for FTP
  - port 23 is reserved for Telnet
  - port 25 is reserved for SMTP (simple mail transfer protocol)
  - port 80 is reserved for HTTP
  - port 1080 is used by SOCKS (network firewall security)
- User-level processes/services generally use port number value >= 1024

Internet Protocols for Networked Application

- Common Internet Protocols
  - Internet Protocol
  - TCP
  - UDP
- Broadcasting
- Multicasting

TCP vs UDP



Transmission Control Protocol (TCP)

- **TCP** is a **connection-oriented reliable stream** transport protocol
- Most common protocol in use today
- Layered on top of IP referred to as TCP/IP
- Provides illusion of point to point connection to an application running on another machine
- Each endpoint can regard a TCP/IP connection as a bi-directional stream of bytes between two endpoints
- Application can detect when other end of connection has gone away/disconnected

User Datagram Protocol (UDP)

- **UDP** is a **connectionless unreliable datagram** transport protocol
- The User Datagram Protocol (UDP) is a lightweight communication protocol
- Differs from TCP in three respects:
  - connection-less transmission
  - best-efforts delivery
  - packet-based data semantics
- Does not establish peer-to-peer connections

User Datagram Protocol (UDP)

- Sender and recipient of do not keep any information about the state of the communication session between the two hosts
- Simply provides **best-efforts delivery**, i.e. no guarantee that data is delivered reliably or in order
- Endpoints do not maintain state information about the communication, UDP data is sent and received on a **packet-by-packet basis**
- **Datagrams** must not be too big, because if they must be fragmented, some pieces might get lost in transit

UDP Advantages

- Simplicity
- Does not include the overhead needed to detect reliability and maintain connection-oriented semantics
  - UDP packets require considerably less processing at the transmitting and receiving hosts
- Does not maintain the illusion of a data stream
  - packets can be transmitted as soon as they are sent by the application instead of waiting in line behind other data in the stream; similarly, data can be delivered to the application as soon as it arrives at the receiving host instead of waiting in line behind missing data

UDP Advantages

- Many operating systems impose limits on how many simultaneous TCP/IP connections they can support.
- Operating system does not need to keep UDP connection information for every peer host, UDP/IP is more appropriate for large-scale distributed systems where each host communicates with many destinations simultaneously

UDP Disadvantages

- When a socket is receiving data on a UDP port, it will receive packets sent to it by any host, whether it is participating in the application or not
- This possibility can represent a security problem for some applications that do not robustly distinguish between expected and unexpected packets
- For this reason, many network firewall administrators block UDP data from being sent to a protected host from outside the security perimeter

UDP Broadcasting

- With UDP/IP, an application can direct a packet to be sent to one other application endpoint
- Could send the same packet to multiple destinations by repeatedly calling **sendto()** (in C) or **DatagramSocket.send()** (in Java)
- This approach has two disadvantages:
  - Excessive network bandwidth is required because the same packet is sent over the network multiple times
  - Each host must maintain an up-to-date list of all other application endpoints who are interested in its data

UDP Broadcasting

- UDP broadcasting provides a partial solution to these issues
- Allows a single transmission to be delivered to all applications on a network who are receiving on a particular port
- Useful for small networked applications
- Expensive because every host on network must receive and process every broadcast packet
- Not used for large networked applications (use IP Multicast)

IP Multicasting

- UDP broadcasting can only be used in a LAN environment
- Even if no application on that host is actually interested in receiving the packet each host on the LAN must:
  - receive packet
  - process the packet
- Multicasting is the solution to both of these concerns
- Appropriate for Internet use, as well as LAN use
- Does not impose burdens on hosts that are not interested in receiving the multicast data

IP Multicasting

- IP addresses in the range 224.0.0.0 through 239.255.255.255 are designated as multicast addresses
- The 224.*.*.* addresses are reserved for use by the management protocols on a LAN, and packets sent to the 239.*.*.* addresses are typically only sent to hosts within a single organization
- Internet-based networked application should therefore use one or more random addresses in the 225.*.*.* to 238.*.*.* range
- The sender transmits data to a multicast IP address, and a subscriber receives the packet if it has explicitly joined that address

IP Multicasting

- Rapidly emerging as the recommended way to build large-scale networked applications over the Internet
- Provides:
  - desirable network efficiency
  - allows the networked application to partition different types of data by using multiple multicast addresses
- Using a well-known multicast address, networked application participants can announce their presence and learn about the presence of other participants

IP Multicasting

- Also an appropriate technique for discovering the availability of other networked application resources such as terrain servers
- These features make multicasting desirable even for LAN-based networked applications.

IP Multicasting Limitations

- Limitations generally related to its infancy
- Although an increasing number of routers are multicast-capable, many older routers are still not capable of handling multicast subscriptions
- In the meantime, multicast-aware routers communicate directly with each other, “tunneling” data past the routers that cannot handle multicast data

Selecting an Network Protocol

- Multiple protocols can be used in a single system
- Not which protocol should I use in my networked application but which protocol should I use to transmit this piece of information?
- Using TCP
  - Reliable data transmission between two hosts
  - Packets are delivered in order, error handling
  - Relatively easy to use
  - Point-to-point limits its use in large-scale networked applications
  - Bandwidth overhead

Selecting an Network Protocol

- Using UDP
  - Lightweight
  - Offers no reliability nor guarantees the order of packets
  - Packets can be sent to multiple hosts
  - Deliver time-sensitive information among a large number of hosts
  - More complex services have to be implemented in the application
    - Serial numbers, timestamps
  - Recovery of lost packets
    - Positive acknowledgement scheme
    - Negative acknowledgement scheme
    - More effective when the destination knows the sources and their frequency
  - Transmit a quench packet if packets are received too often

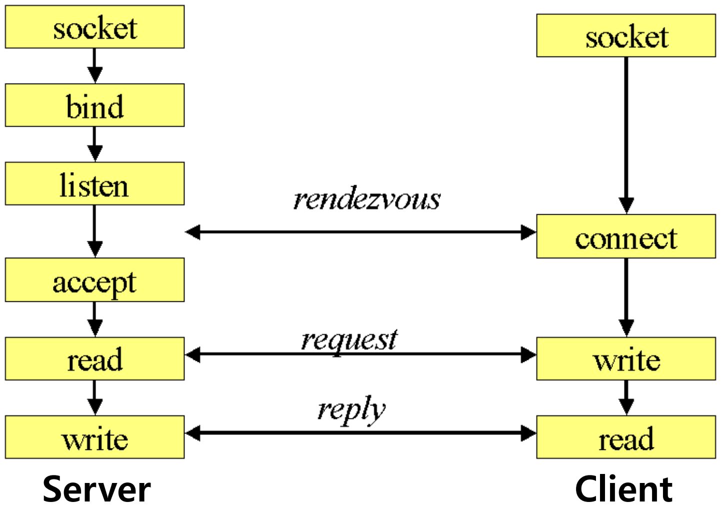
Selecting an Network Protocol

- Using IP Broadcasting
  - Design considerations similar to UNICAST UDP/IP
  - Limited to LAN
  - Not for large-scale networked application (with a large number of participants)
  - To distinguish different applications using the same port number (or multicast address)
    - Avoid the problem entirely – assign the necessary number
    - Detect conflict and re-negotiate – notify the participants and direct them to migrate a new port number
    - Use protocol and instance magic numbers – each packet includes a magic number at a well-known position
    - Use encryption

Selecting an Network Protocol

- Using IP Multicasting
  - Provides a quite efficient way to transmit information among a large number of hosts
  - Information delivery is restricted
    - Time-to-live
    - Group subscription
  - Preferred method for large-scale networked application
  - How to separate the information flow among different multicast groups
    - A single group/address for all information
    - Several multicast groups to segment the information

TCP



C/C++ TCP Socket Implementation

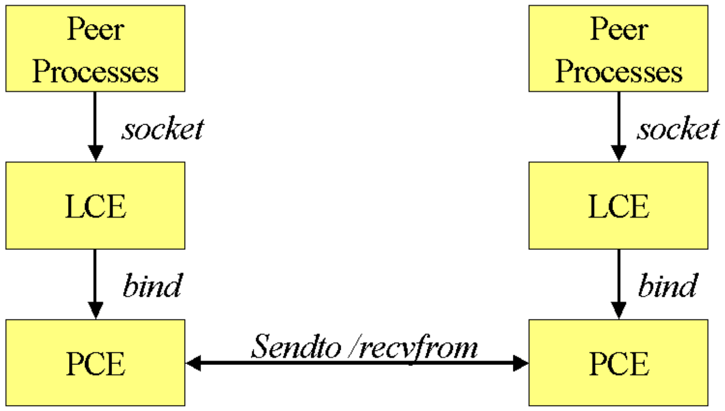
CLIENT ACTIONS:

1. Obtain a socket
2. Connect to the server
3. Communicate with server
   - * Send data/requests
   - * Receive data/replys
4. Close the socket

SERVER ACTIONS:

1. Obtain a socket
2. Bind the socket to a ‘well known’ port
3. Receive connections from clients
4. Communicate with clients
   - * Receive data/requests
   - * Send data/replys
5. Close the socket

UDP



C/C++ UDP Socket Implementation

STEPS TO IMPLEMENT A UDP SOCKET

1) Obtain a socket
2) Bind the socket to a ‘well known’ port
3) Transmit Data
4) Receive Data
5) Close the socket

* Above process is ‘a way’ not the only way

C/C++ Multicasting Socket Implementation

* **TO TRANSMIT DATA:**

- Multicast transmission is nearly identical to UDP/IP. Make sure the packets are sent to a multicast address
- The SO_BROADCAST option need not be set
- Can set the Time To Live field as shown below

```
unsigned char ttl = 31;
setsockopt(sock, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(ttl));
```

C/C++ Multicasting Socket Implementation

* **TO RECEIVE DATA:**

- The application must subscribe the socket to a multicast address
- Subscribing to a multicast address is accomplished by calling setsockopt() with the IP_ADD_MEMBERSHIP option

```
struct ip_mreq joinAddr;

// Specify the multicast address to join
joinAddr.imr_multiaddr = inet_addr("245.8.2.58");

// Specify which local IP address will do the multicast join
joinAddr.imr_interface = INADDR_ANY;

setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, &joinAddr, sizeof(joinAddr))
```

C / C++ Multicasting Socket Implementation

* **TO RECEIVE DATA cont:**

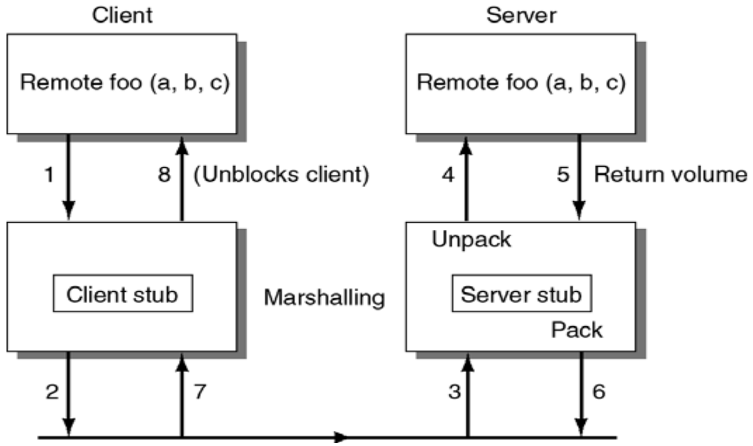
- To cancel a multicast subscription call setsockopt() with the IP_DROP_MEMBERSHIP option

```
struct ip_mreq joinAddr;

// Specify the multicast address to drop
joinAddr.imr_multiaddr = inet_addr("245.8.2.58");

// Specify which local IP address will do the multicast drop
joinAddr.imr_interface = INADDR_ANY;

setsockopt(sock, IPPROTO_IP, IP_DROP_MEMBERSHIP, &joinAddr, sizeof(joinAddr))
```

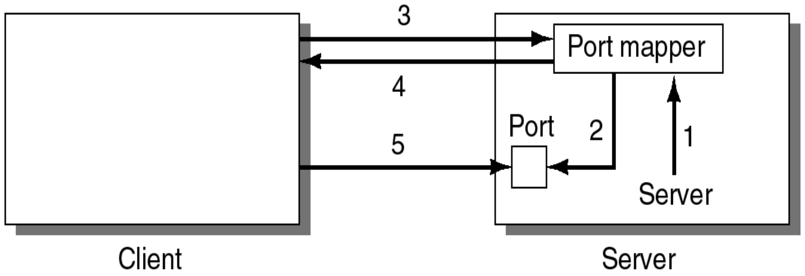
Remote Procedure Calls (RPC)

- RPC is an inter-process communication that resembles a normal procedure call.
- RPC is a client/server communication mechanism
  - The called procedure takes the role of the server, and the caller takes the role of the client

Remote Procedure Call (RPC) Stubs



Establishing Communication for RPC



1. I need a port
2. Here is your port
3. I need a handle
5. Communicate using handle

Reference

- http://www.cs.colostate.edu/~cs551/CourseNotes/Communication/CommTOC.html
- http://www.cs.colostate.edu/~cs551/CourseNotes/LM/LM.Lecture3.ppt