UNIT II : DISTRIBUTED COMPUTING

Introduction - Distributed Systems - Hardware and Software concepts - Design issues; Communication in Distributed systems: Layered protocols - ATM networks - Client Server model - Remote Procedure Calls

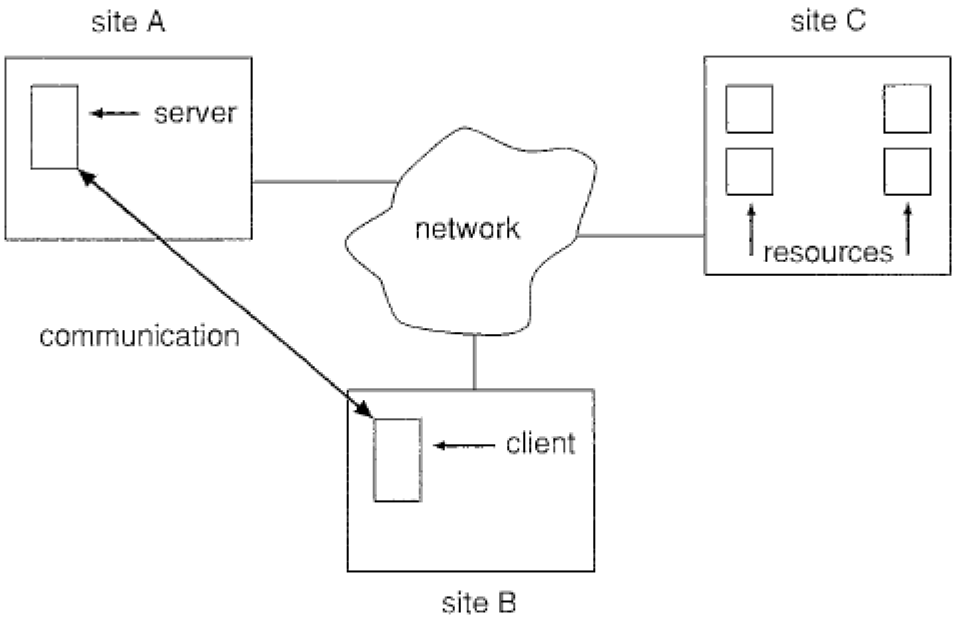
ADVANTAGES OF DISTRIBUTED OPERATING SYSTEM

Advantages over Centralized Systems

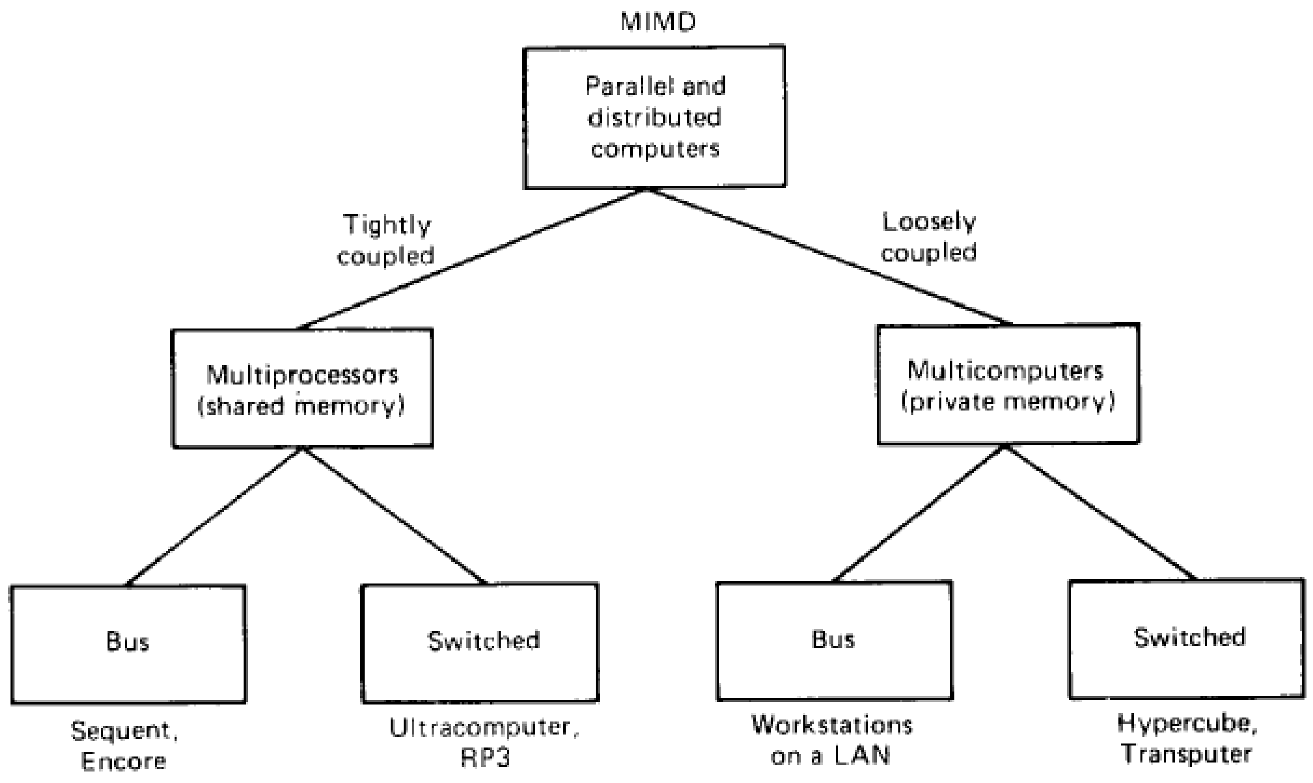
Economics	Microprocessors offer a better price/performance than mainframes
Speed	A distributed system may have more total computing power than a mainframe
Inherent distribution	Some applications involve spatially separated machines
Reliability	If one machine crashes, the system as a whole can still survive
Incremental growth	Computing power can be added in small increments

Advantages over independent PC's

Data sharing	Allow many users access to a common database (Ex: Airline reservation)
Device sharing	Allow many users to share expensive peripherals like color laser printers, phototypesetters and massive archival storage devices like optical jukeboxes
Communication	Make human to human communications easier (Ex: email)
Flexibility	Spread the workload over the available machines in the most cost effective way



HARDWARE CONCEPTS OF DISTRIBUTED OS



Based on the number of instruction and data streams

- SISD
    - A computer with a single instruction stream and a single data stream is called SISD.
    - All traditional uniprocessor computers (i.e., those having only one CPU) fall in this category, from personal computers to large mainframes
- SIMD
    - single instruction stream, multiple data stream.
    - Array processors with one instruction unit that fetches an instruction, and then commands many data units to carry it out in parallel, each with its own data.
- MISD
    - multiple instruction stream, single data stream
- MIMD
    - a group of independent computers, each with its own PC, program, and data.
    - all distributed systems are MIMD

MIMD Groups

- Multiprocessors
    - have shared memory
    - single virtual address space that is shared by all CPUs
- Multi computers
    - every machine has its own private memory

Based on the architecture of the interconnection network

- Bus
    - single network, backplane, bus, cable, or other medium that connects all the machines
    - Example: Cable television
- Switched
    - individual wires from machine to machine, with many different wiring patterns
    - Messages move along the wires, with an explicit switching decision made at each step to route the message along one of the outgoing wires. Example: PSTN

Based on their communication

- **Tightly coupled**
    - Delay experienced when a message is sent from one computer to another is short, and the data rate is high
    - used more as parallel systems
    - two CPU chips on the same PCB and connected by wires etched onto the board
    - multiprocessors tend to be more tightly coupled because they can exchange data at memory speeds
- **Loosely coupled**
    - The inter machine message delay is large and the data rate is low
    - used as distributed systems
    - two computers connected by a 2400 bit/sec modem over the telephone system
    - some fiber optic based multicomputers can also work at memory speeds

Bus Based Multiprocessors

- Consists of some number of CPUs all connected to a common bus, along with a memory module
- Coherent - Shared memory access between CPUs
- Cache memory: To speed up the performance; high speed; hit rate;
- Write through cache
- Snoopy cache
    - All caches constantly monitors the bus; either removes that entry or updates the cache entry with the new value

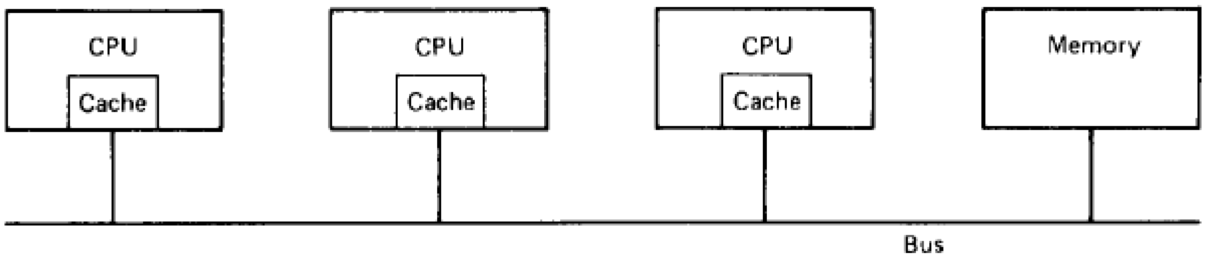


Fig. 1-5. A bus-based multiprocessor.

Switched Multiprocessor

To build a MP with more than 64 processors

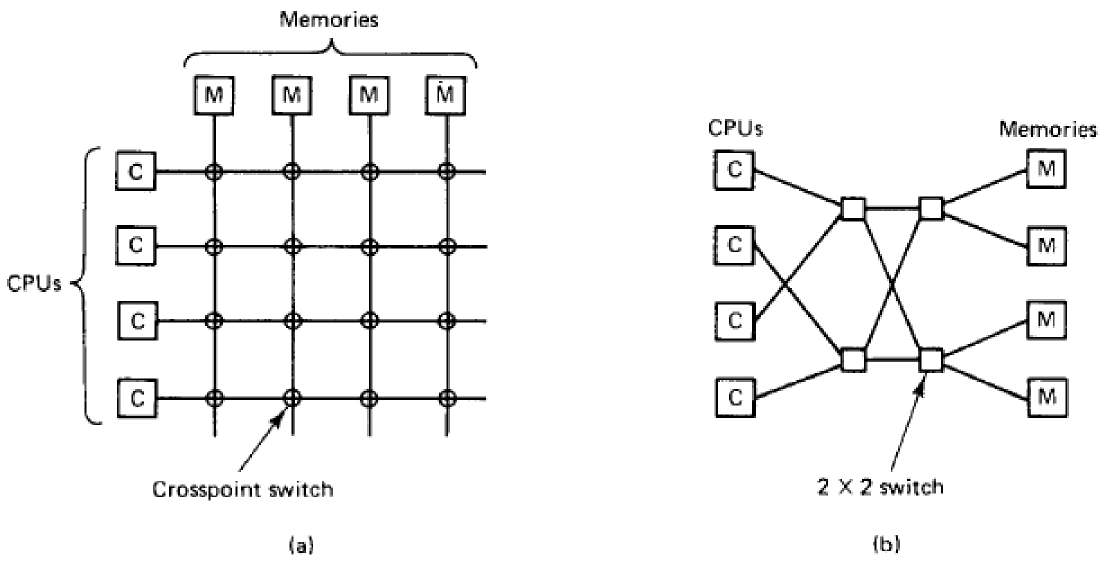


Fig. 1-6. (a) A crossbar switch. (b) An omega switching network.

Crossbar switch

- Divide the memory up into modules and then connect them to the CPUs with a crossbar switch
- Many CPUs can be accessing memory at the same time
- We need n x n crosspoint switches with n CPUs and n memories

Omega network

- 2x2 switches
- Switches can be set in nano seconds or less
- every CPU can access every memory

NUMA

- Requires complex algorithm for good software placement

Bus Based Multicomputers

- Less traffic; need not be a high speed backplane bus
- Lower speed LAN (10-100 Mbps compared to 300 Mbps & above for a backplane bus)

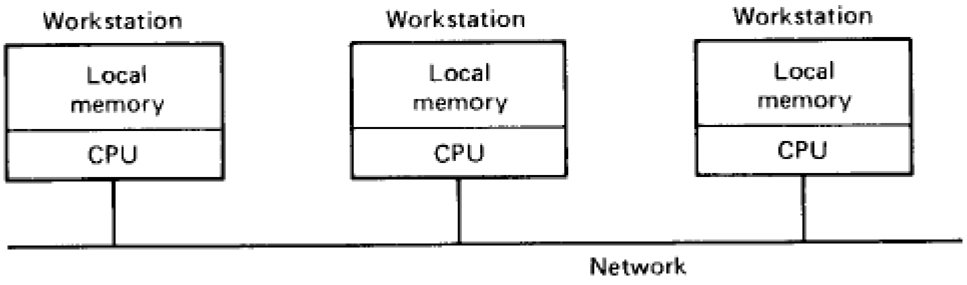


Fig. 1-7. A multicomputer consisting of workstations on a LAN.

Switched Multicomputers

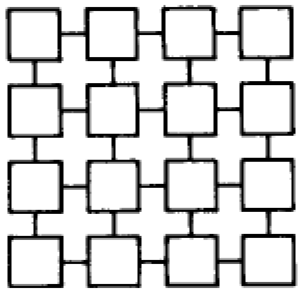
Each CPU has direct and exclusive access to its own, private memory

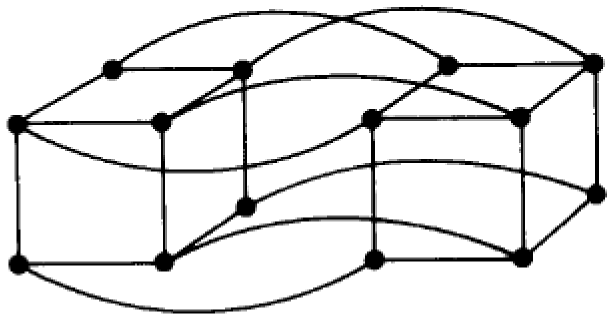
Grid Topology

Best suited for problems that have an inherent 2D nature like graph theory or vision (Ex: robot eyes, analysing photograph)

Hypercube

- n-dimensional cube
- Each vertex is a CPU
- Each edge is a connection between two CPUs
- The corresponding vertices in each of the two cubes are connected





SOFTWARE CONCEPTS OF DISTRIBUTED OS

- Network Operating Systems
- True Distributed Systems
- Multiprocessor Timesharing Systems

System	**Description**	**Main Goal**
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

Network Operating Systems

- In this model, each user has a workstation for his exclusive use.
- It may or may not have a hard disk.
- It definitely has its own operating system.
- All commands are normally run locally, right on the workstation.
- It is possible for a user to log into another workstation remotely
- Workstations can import or mount these file systems, augmenting their local file systems with those located on the servers

True Distributed Systems

- Tightly-coupled software on the same loosely-coupled (i.e., multicomputer) hardware.
- Users should not have to be aware of the existence of multiple CPUs in the system
- There must be a single, global inter-process communication mechanism so that any process can talk to any other process
- Global protection scheme
- Process management must also be the same

Multiprocessor Timesharing Systems

- Tightly-coupled software on tightly-coupled hardware
- Existence of a single run queue: a list of all the processes in the system that are logically unblocked and ready to run

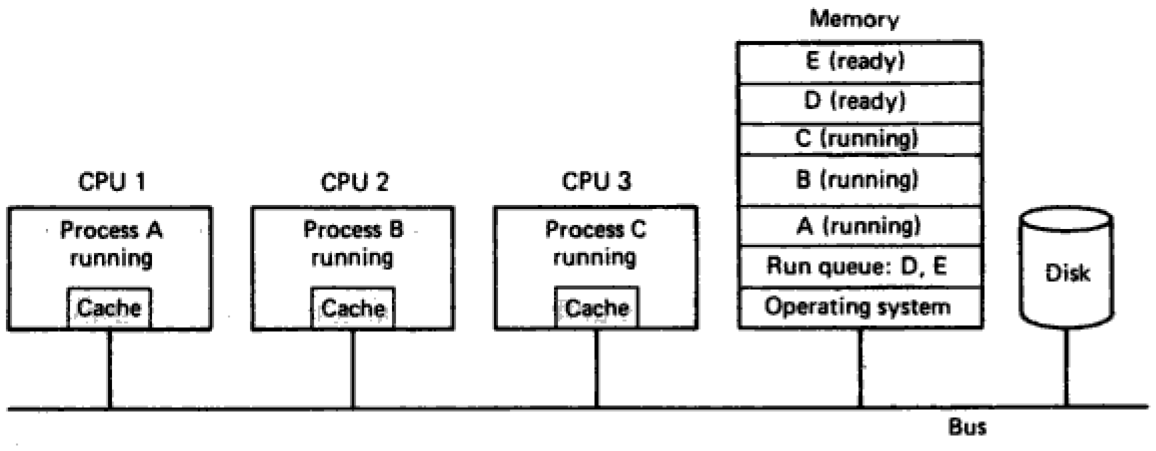


Fig. 1-11. A multiprocessor with a single run queue.

Comparison of three different ways of organizing n CPUs

Item	Network OS	Distributed OS	Multiprocessor OS
Dos it look like a virtual uniprocessor	No	Yes	Yes
Do all have to run the same OS	No	Yes	Yes
Have many copies of the OS are there?	N	N	1
How is communication achieved?	Shared files	Messages	Shared memory
Are agreed upon network protocols required	Yes	Yes	No
Is there a single run queue?	No	No	Yes
Dos file sharing have well defined semantics	Usually No	Yes	Yes

DESIGN ISSUES IN DISTRIBUTED OPERATING SYSTEMS

- Transparency
- Flexibility
- Reliability
- Performance
- Scalability

Transparency: LMR-CP

- how to achieve the single-system image
- how do the system designers makes everyone into thinking that the collection of machines is simply a timesharing system

Types of Transparency

Location	The users cannot tell where resources are located
Migration	Resources can move at will without changing their names
Replication	The users cannot tell how many copies exist
Concurrency	Multiple users can share resources automatically
Parallelism	Activities can happen in parallel without users knowing

Location transparency refers to the fact that in a true distributed system, users cannot tell where hardware and software resources such as CPUs, printers, files, and data bases are located.

The name of the resource must not secretly encode the location of the resource, so names like machine1:prog.c or /machine1/prog.care not acceptable

Migration transparency means that resources must be free to move from one location to another without having their names change

Since a path like /work/news does not reveal the location of the server, it is location transparent.

With replication transparency, the operating system is free to make additional copies of files and other resources on its own without the users noticing

Flexibility

Flexibility in a distributed operating system is enhanced through the modular and characteristics of the distributed OS, and by providing a richer set of higher-level services.

Monolithic: Kernel that provides most services itself

Microkernel: Kernel should provide as little as possible, with the bulk of the operating system services available from user-level servers.

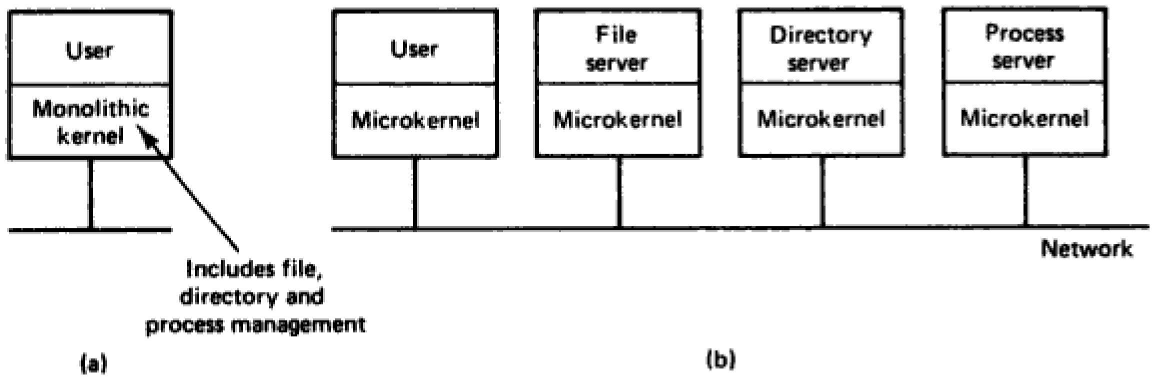


Fig. 1-14. (a) Monolithic kernel. (b) Microkernel.

Reliability

If a machine goes down, some other machine takes over the job

- **Availability**
  - Fraction of time that the system is usable
- **Security**
  - Files and other resources must be protected from unauthorized usage
- **Fault Tolerance**

Performance

It should not be appreciably worse than running the same application on a single processor

- Response time
- Throughput
- System utilization
- Network capacity consumed
- Grain size of all computations
  - **Fine-grained parallelism**
    - Jobs that interact highly with one another
  - **Coarse-grained parallelism**
    - Jobs that involve large computations, low interaction rates, and little data
- Fault tolerance

Scalability

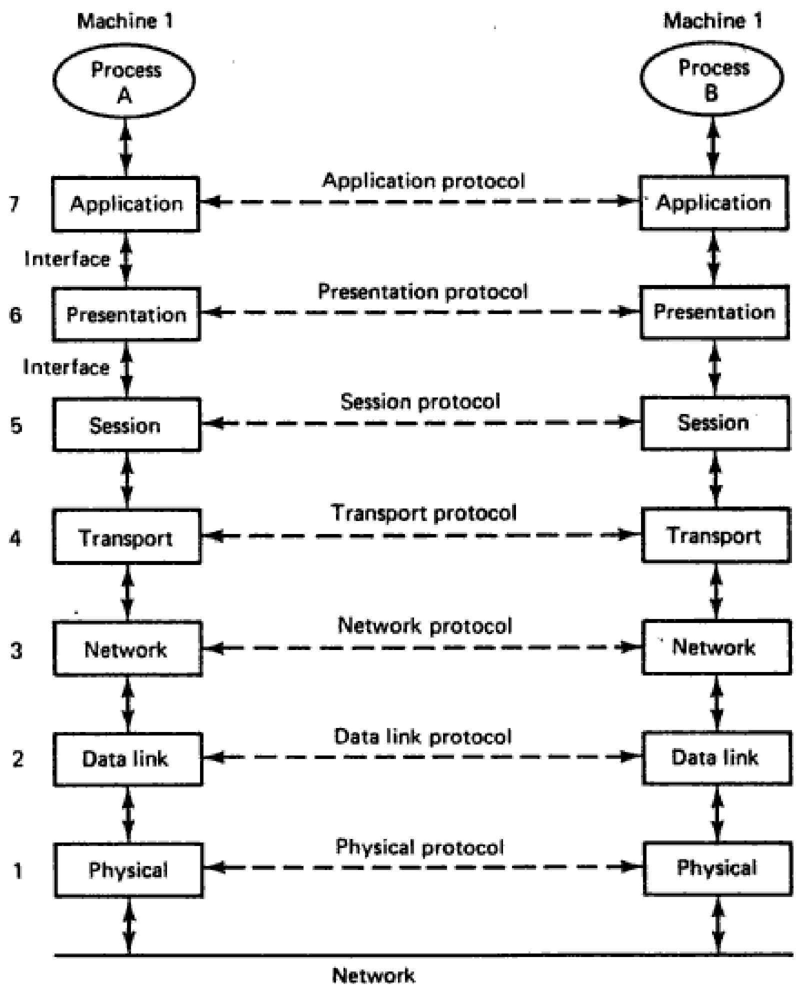
Potential bottlenecks that designers should try to avoid in very large distributed systems

Centralized components	A single mail server for all users
Centralized tables	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

Only decentralized algorithms should be used with the following characteristics

1. No machine has complete information about the system state.
2. Machines make decisions based only on local information.
3. Failure of one machine does not ruin the algorithm.
4. There is no implicit assumption that a global clock exists

LAYERED PROTOCOL IN COMMUNICATION



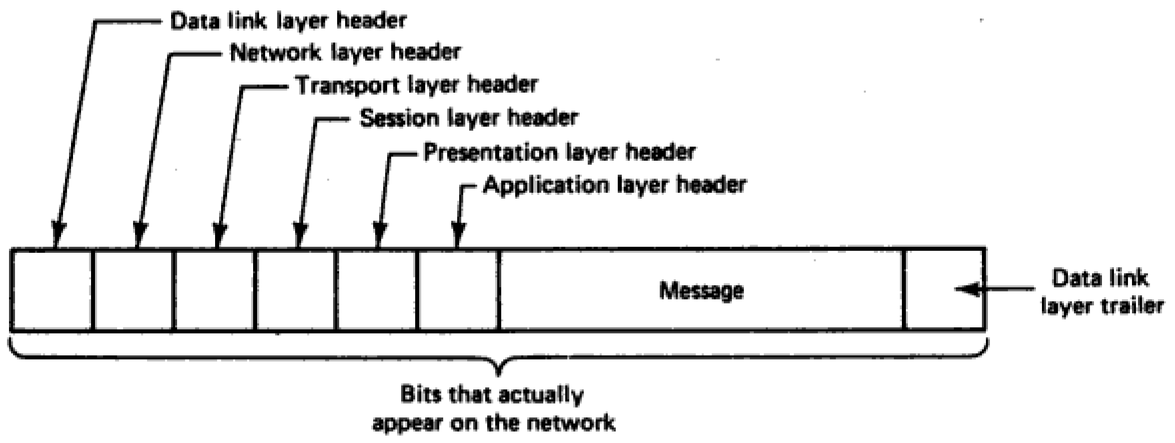


Fig. 2-2. A typical message as it appears on the network.

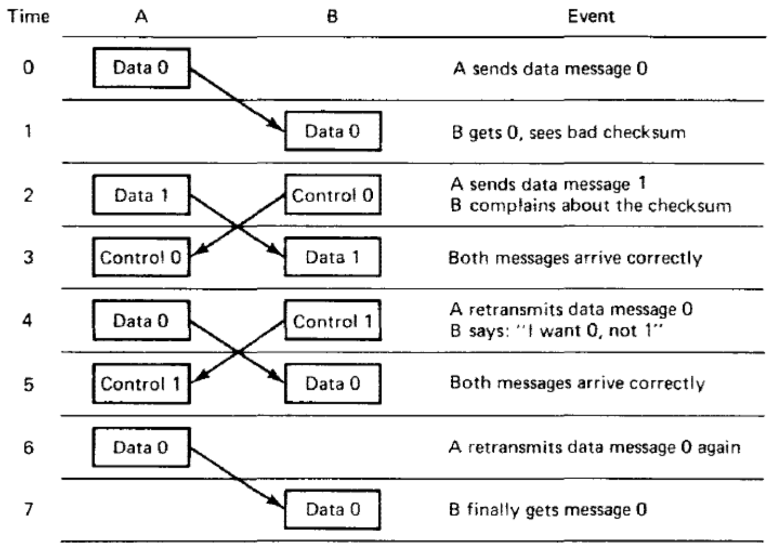
- The OSI model is designed to allow open systems to communicate.
- An open system is one that is prepared to communicate with any other open system by using standard rules that govern the format, contents, and meaning of the messages sent and received.
- These rules are formalized as protocols.
- A protocol is an agreement between the communicating parties on how communication is to proceed.
- The collection of protocols used in a particular system is called a protocol suite or protocol stack

Physical Layer

- Concerned with transmitting the 0s and 1s.
- How many volts to use for 0 and 1, how many bits per second can be sent, and whether transmission can take place in both directions simultaneously
- The size and shape of the network connector (plug), the number of pins and their meaning
- Deals with standardizing the electrical, mechanical, and signaling interfaces
- Many physical layer standards like, the RS-232-C for serial communication lines.

The Data Link Layer

- Group the bits into units, called frames, and see that each frame is correctly received
- Frames are assigned sequence numbers (in the header) and appended with checksum
- If the checksum failed, then the sender is asked to retransmit



Time
A
B
Event
0
Data 0
A sends data message 0
1
Data 0
B gets 0, sees bad checksum
2
Data 1
Control 0
A sends data message 1
B complains about the checksum
3
Control 0
Data 1
Both messages arrive correctly
4
Data 0
Control 1
A retransmits data message 0
B says: "I want 0, not 1"
5
Control 1
Data 0
Both messages arrive correctly
6
Data 0
A retransmits data message 0 again
7
Data 0
B finally gets message 0

Fig. 2-3. Discussion between a receiver and a sender in the data link layer.

The Network Layer

- Routing is the primary task of the network layer
- The shortest route is not always the best route
- Two network-layer protocols are in widespread use,
  - Connection-oriented - X.25
  - Connectionless – IP

The Transport Layer

- Provides reliable connection
- The transport layer breaks it into pieces small enough for each to fit in a single packet, assigns each one a sequence number, and then sends them all.
- ISO transport protocol has five variants, known as TP0 through TP4
- Transmission Control Protocol (TP4)
- Universal Datagram Protocol

The Session Layer

- It provides dialog control, to keep track of which party is currently talking, and it provides synchronization facilities.
- The latter are useful to allow users to insert checkpoints into long transfers, so that in the event of a crash it is only necessary to go back to the last checkpoint, rather than all the way back to the beginning.

The Presentation Layer

- Concerned with the meaning of the bits.
- Most messages consist of structured information such as people's names, addressesetc
- In the presentation layer it is possible to define records containing fields like these and then have the sender notify the receiver that a message contains a particular record in a certain format.
- This makes it easier for machines with different internal representations to communicate.

The Application Layer

- The application layer is really just a collection of miscellaneous protocols for common activities such as electronic mail, file transfer, and connecting remote terminals to computers over a network.
- Examples: X.400 electronic mail protocol and the X.500 directory server.

ATM NETWORKS

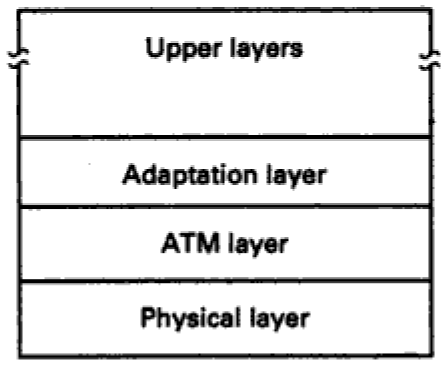


Fig. 2-4. The ATM reference model.

- The ATM model is that a sender first establishes a connection (i.e., a virtual circuit) to the receiver or receivers.
- During connection establishment, a route is determined from the sender to the receiver(s) and routing information is stored in the switches along the way.
- Using this connection, packets can be sent, but they are chopped up by the hardware into small, fixed-sized units called cells.
- The cells for a given virtual circuit all follow the path stored in the switches.
- When the connection is no longer needed, it is released and the routing information purged from the switches.
- Cell switching is that it can handle both point-to-point and multicasting efficiently
- Fixed-size cells allow rapid switching

The ATM Physical Layer

- An ATM adaptor board plugged into a computer can put out a stream of cells onto a wire or fiber.
- The transmission stream must be continuous.
- When there are no data to be sent, empty cells are transmitted, which means that in the physical layer, ATM is really synchronous, not asynchronous.
- Within a virtual circuit, however, it is asynchronous.
- The adaptor board can also use SONET (Synchronous Optical NETwork) in the physical layer

The ATM Layer

- 53-byte cell containing a 48-byte data field
- ATM cells will span SONET frames
- Two separate levels of synchronization are thus needed:
  - one to detect the start of a SONET frame
  - one to detect the start of the first full ATM cell within the SONET payload.
- A standard for packing ATM cells into SONET frames exists, and the entire layer can be done in hardware.

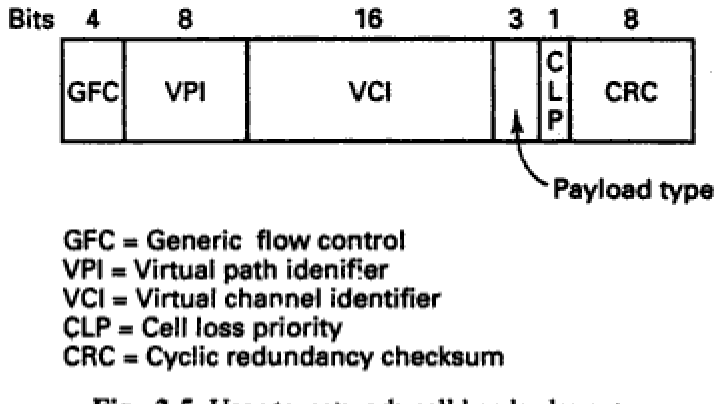

Bits 4 8 16 3 1 8
GFC
VPI
VCI
CLP
CRC
Payload type
GFC = Generic flow control
VPI = Virtual path idenifier
VCI = Virtual channel identifier
CLP = Cell loss priority
CRC = Cyclic redundancy checksum

Fig. 2-5. User-to-network cell header layout.

The ATM Adaptation Layer

A mechanism to allow a computer to send a packet and to have the ATM hardware break it into cells, transmit the cells, and then have them reassembled at the other end, generating one interrupt per packet, not per cell.

Four classes of traffic:

1. Constant bit rate traffic (for audio and video).
2. Variable bit rate traffic but with bounded delay.
3. Connection-oriented data traffic.
4. Connectionless data traffic

- Simple and Efficient Adaptation Layer
  - It uses only one bit in the ATM header, one of the bits in the Payload type field. This bit is normally 0, but is set to 1 in the last cell of a packet.

ATM Switching

- ATM networks are built up of copper or optical cables and switches.
- Cells originating at any of the eight computers attached to the system can be switched to any of the other computers by traversing one or more switches.
- Each of these switches has four ports, each used for both input and output.

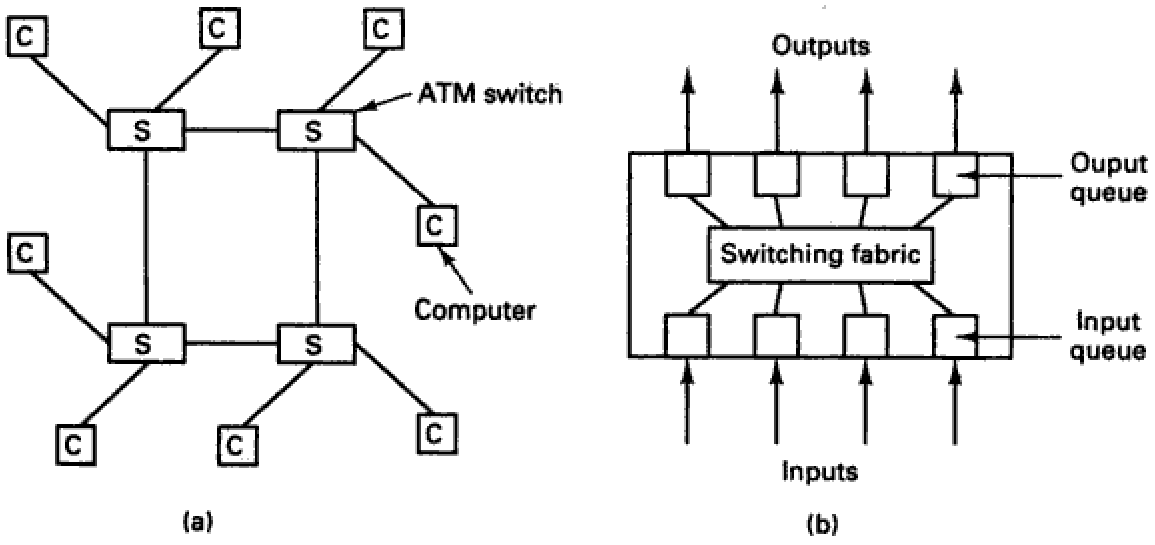


Fig. 2-6. (a) An ATM switching network. (b) Inside one switch.

CLIENT SERVER MODEL

- Structure the operating system as a group of co-operating processes called servers
- Servers offers services to the users called clients
- Client server machines normally run the same micro kernel
- A machine may run a single process or it may run multiple clients, multiple servers or a mixture of the two
- This model is based on a simple, connectionless request / reply protocol

Advantage

- Simplicity
- Efficiency - Protocol stack is shorter
- Communication services can be reduced to two system calls like send (dest, &mptr) and receive (addr, &mptr)

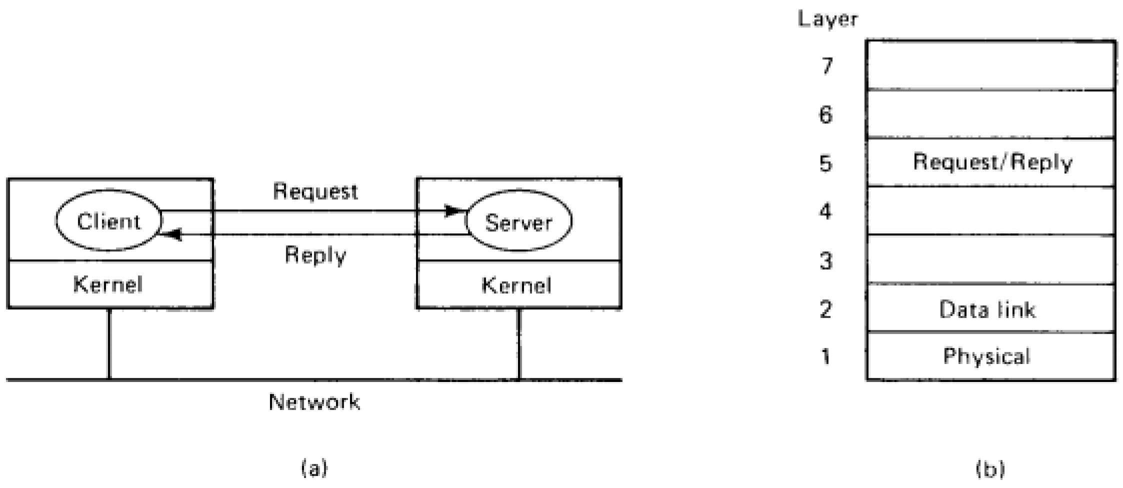


Fig. 2-7. The client-server model. Although all message passing is actually done by the kernels, this simplified form of drawing will be used when there is no ambiguity.

Four Design issues for the communication primitives and solutions

Types of addressing

Machine.process addressing

- Hardwire machine.number into client code
- machine.process like 243.0 to 199.0
- Variant: machine.local-id
  - local-id: randomly chosen 16-bit or 32-bit integer (or the next one in a sequence)
  - Berkeley Unix uses this method
- Disadv: Not transparent - Client knows where the server is

Process addressing with broadcasting

- Processes pick random addresses; locate them by broadcasting
- sender can broadcast a special locate packet containing the address of the destination process
- All the kernels check to see if the address is theirs
- If so, it sends back a here I am message giving their NW address (machine number)
- The sending kernel uses this address, and caches it to avoid broadcasting the next time

Address lookup via a name server

- Put ASCII server names in clients; look them up at run time
- To avoid the extra load on the system in broadcasting, a special mapping server called name server is used
- The name server returns the address of server

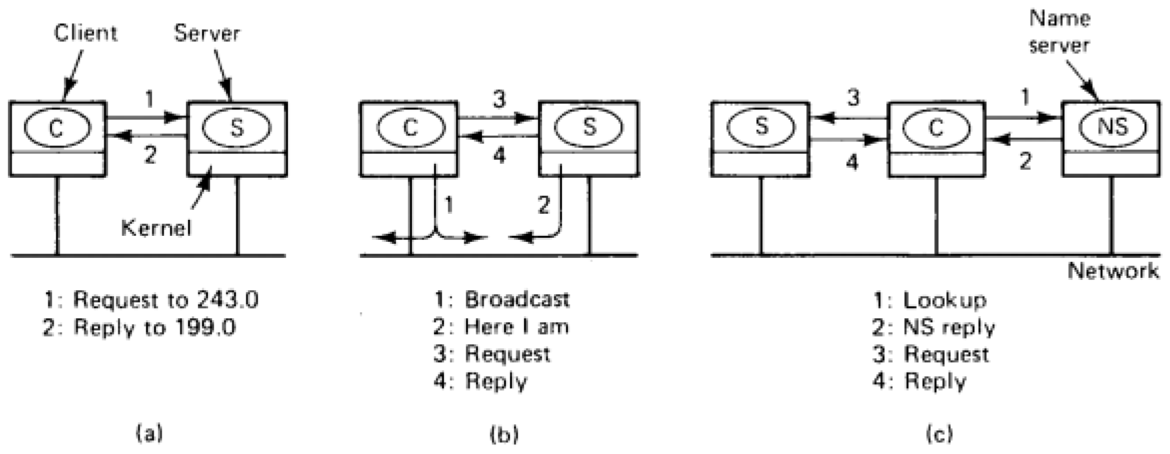


Fig. 2-10. (a) Machine.process addressing. (b) Process addressing with broadcasting. (c) Address lookup via a name server.

Blocking versus Nonblocking Primitives

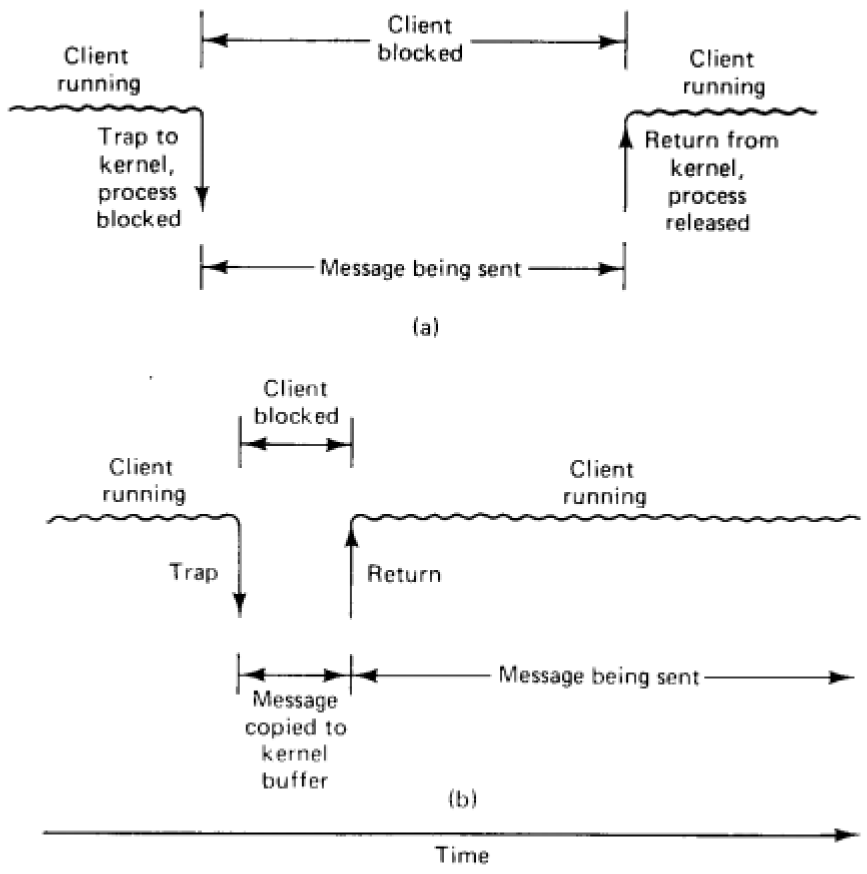


Fig. 2-11. (a) A blocking send primitive. (b) A nonblocking send primitive.

Buffered versus Unbuffered Primitives

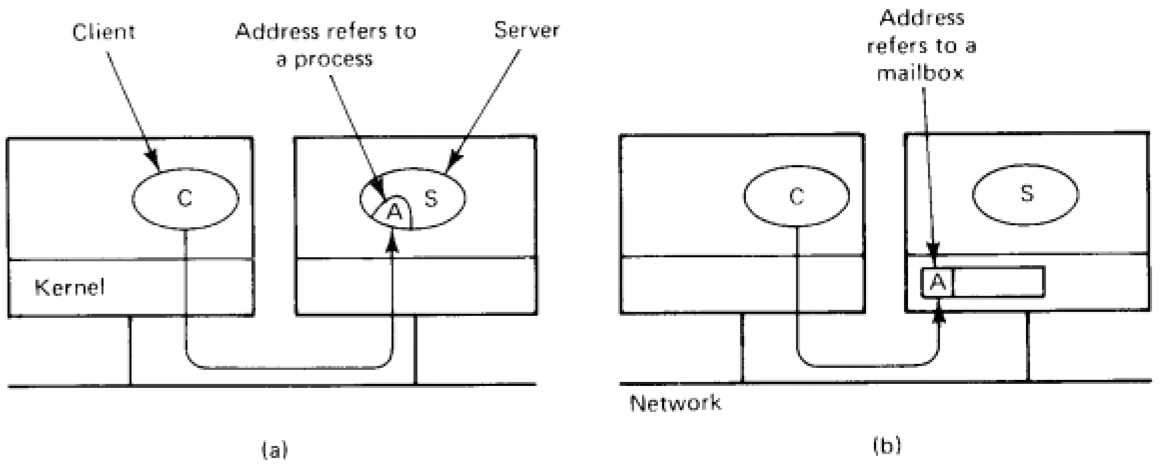


Fig. 2-12. (a) Unbuffered message passing. (b) Buffered message passing.

Reliable versus Unreliable Primitives

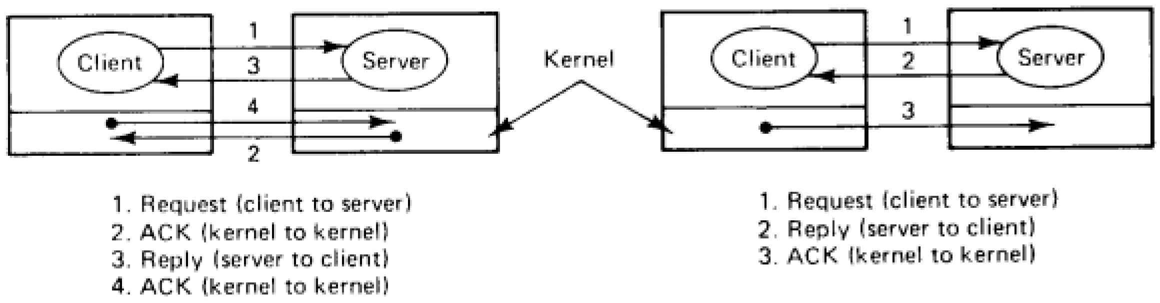

Client
Server
1
3
4
2
Kernel
Client
Server
1
2
3
1. Request (client to server)
2. ACK (kernel to kernel)
3. Reply (server to client)
4. ACK (kernel to kernel)
1. Request (client to server)
2. Reply (server to client)
3. ACK (kernel to kernel)

Fig. 2-13. (a) Individually acknowledged messages. (b) Reply being used as the acknowledgement of the request. Note that the ACKs are handled entirely within the kernels.

Summary of Four Design Issues:-

Item	**Option 1**	**Option 2**	**Option 3**
Addressing	Machine Number	Sparse process addresses	ASCII names looked up via server
Blocking	Blocking primitives	Nonblocking with copy to kernel	Nonblocking with interrupt
Buffering	Unbuffered, discarding unexpected messages	Unbuffered, temporarily keeping unexpected messages	Mailboxes
Reliability	Unreliable	Request-Ack-Reply Ack	Request-Reply-Ack

Implementing the Client-Server Model

Packet types used in Client-Server protocols

Code	Packet Type	From	To	Description
REQ	Request	Client	Server	The client wants service
REP	Reply	Server	Client	Reply from the server to the client
ACK	Ack	Either	Other	The previous packet arrived
AYA	Are You Alive?	Client	Server	Probe to see if the server has crashed
IAA	I Am Alive	Server	Client	The server has not crashed
TA	Try Again	Server	Client	The server has no room
AU	Address Unknown	Server	Client	No process is using this address

Various types of packet sequences

- Straight request/reply with no acknowledgement
- Each message is acknowledged individually
- Reply acting as the ack
- Nervous client checking to see if the server is still there

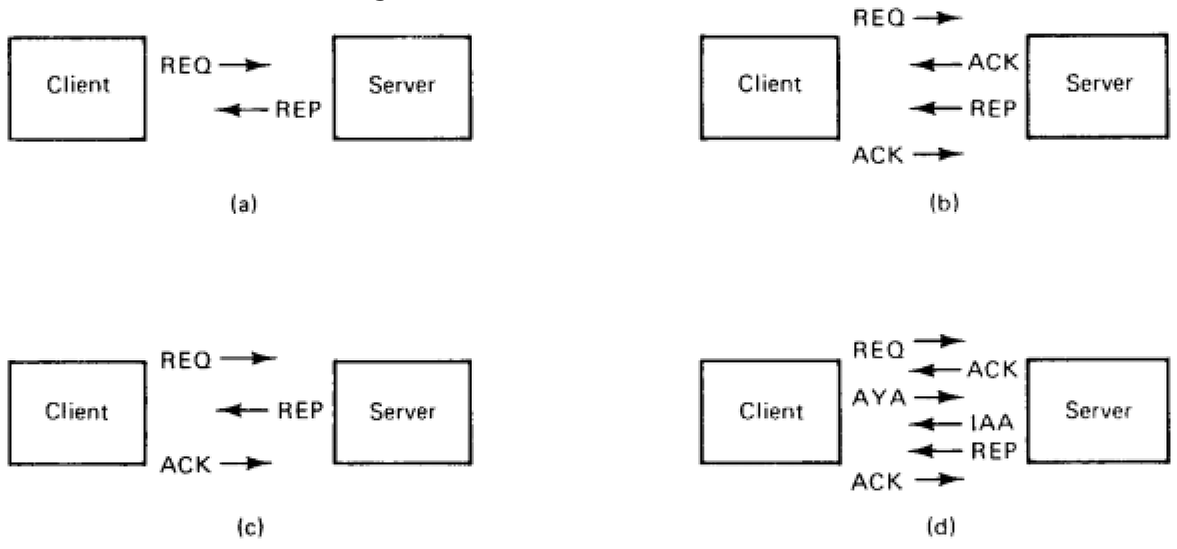
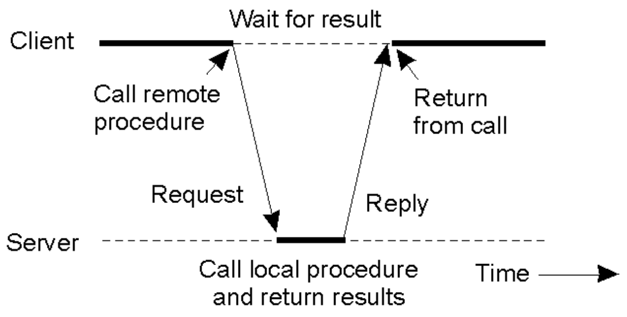


Fig. 2-16. Some examples of packet exchanges for client-server communication.

REMOTE PROCEDURE CALL (RPC)

- Allowing programs to call procedures located on other machines
- When a process on machine A calls a procedure on machine B,
  the calling process on A is suspended
- Execution of the called procedure takes place on B
- Information can be transported from the caller to the callee in the parameters and can come back in the procedure result
- No message passing or I/O at all is visible to the programmer



Basic RPC Operation

- The idea behind RPC is to make a remote procedure call look as much as possible like a local one
- RPC achieves its transparency in an analogous way
  - When read is a remote procedure, a different version of read called client stub is put into the library
  - This stub packs the parameters into a message and asks the kernel to send the message to the server
  - Following the call to send, the client stub calls receive, blocking itself until the reply comes back
  - When the message arrives at the server, the kernel passes it up to a server stub that is bound with the actual server
  - The server stub unpacks the parameters from the message and then calls the server procedure in the usual way

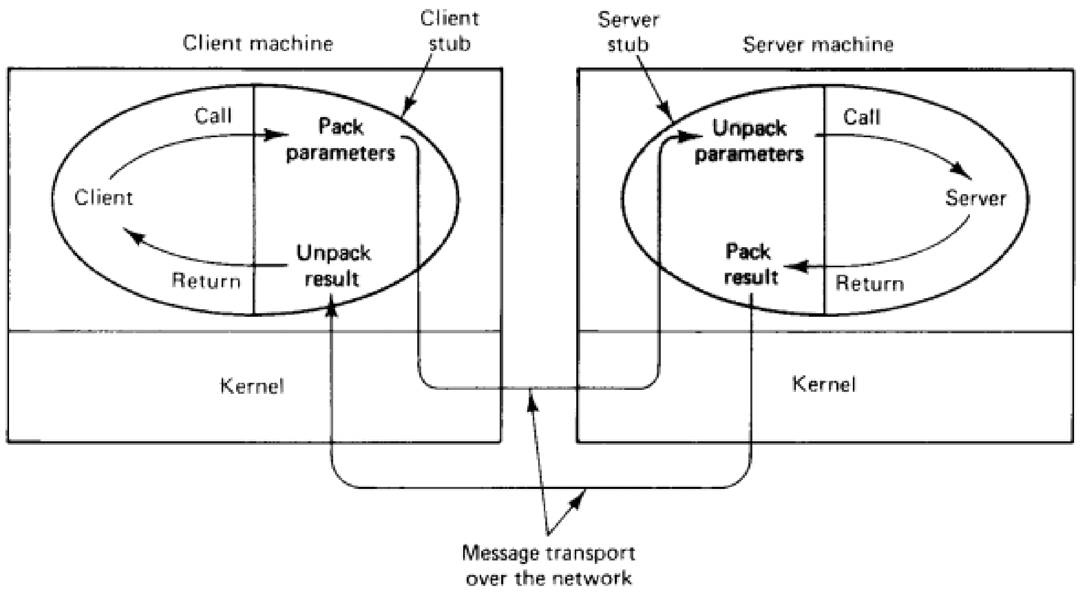


Fig. 2-18. Calls and messages in an RPC. Each ellipse represents a single process, with the shaded portion being the stub.

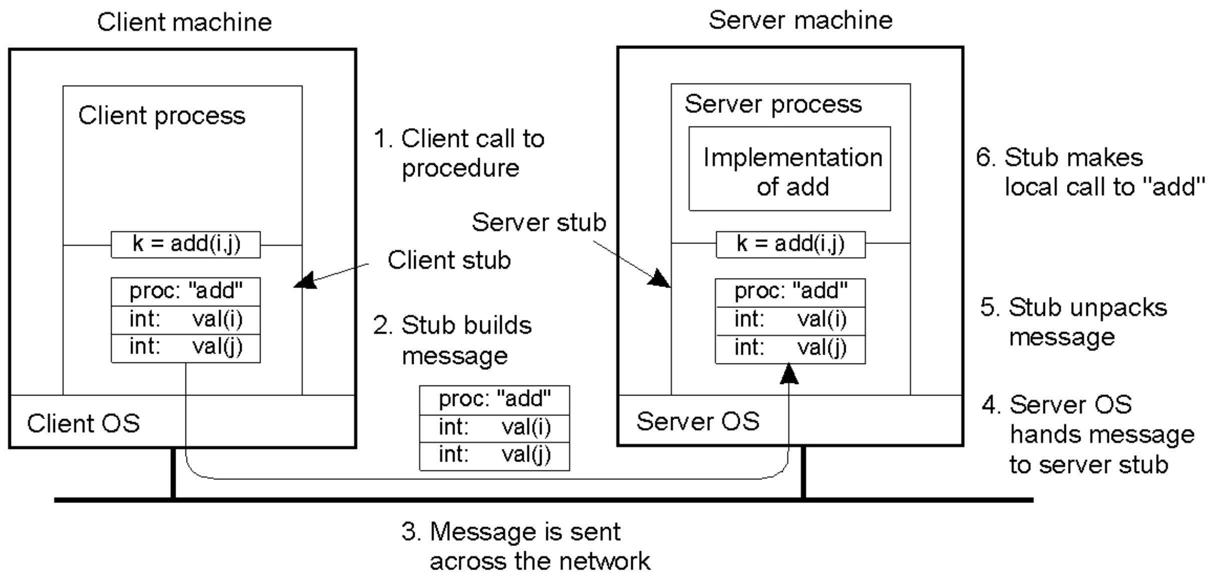
Steps involved in a remote procedure call

1. The client procedure calls the client stub in the normal way
2. The client stub builds a message and traps to the kernel
3. The kernel sends the message to the remote kernel
4. The remote kernel gives the message to the server stub
5. The server stub unpack the parameters and calls the server
6. The server does the work and returns the result to the stub
7. The server stub packs it in a message and traps to the kernel
8. The remote kernel sends the message to the client's kernel
9. The client's kernel gives the message to the client stub
10. The stub unpacks the result and returns to the client

The net effect of all these steps is to convert the local call by the client procedure to the client stub to a local call to the server procedure without either client or server being aware of the intermediate steps.

Parameter Passing

- Parameter marshaling
    - Packing parameters into a message
    - This model works fine as long as the client and server machines are identical and all the parameters and results are scalar types such as integers, characters …
    - In a large distributed system, multiple machine types are present. Each machine has its own representation of numbers, characters and other data items like EBCDIC code versus ASCII, representation of integers (1s complement vs 2s complement), floating point numbers, byte order (little endian vs big endian)
    - Solution is to use a network standard or canonical form for data types are defined
    - All senders convert their internal types to this form while marshaling
    - In-efficient due to two conversions needed
- Alternative to marshaling is that the client uses its own native format and indicates in the first byte of the message which format this is
    - Ex: Little endian client builds a little endian message and server checks the 1st byte for known messages



Dynamic Binding

- The server performs registering with binder during initialize and exports its interface
- A handle is used to locate the server
- The client sends a message to the binder asking to import the server interface
- If a suitable interface exists, the binder returns the server handle. Otherwise, the RPC fails

The Binder Interface

Call	**Input**	**Output**
Register	Name, version, handle, unique id	
Deregister	Name, version, unique id	
Lookup	Name, version	Handle, unique id

RPC SEMANTICS IN THE PRESENCE OF FAILURES

Five classes of failures

1. The client is unable to locate the server
2. The request message from the client to the server is lost
3. The reply message from the server to the client is lost
4. The server crashes after receiving a request
5. The client crashes after sending a request

1) Client Cannot Locate the Server

- Client cannot locate a suitable server
- The server might be down
- The server evolves and a new version of the interface is installed with new stubs

Solutions

- Return with a special error code (-1), not reliable always (Ex: -1 is legitimate value for sum (7, -8))
- Raise exceptions (java) , special procedure (ada), signal handlers (c) etc
    - Having to write an exception or signal handler destroys the transparency

2) Lost Request Messages

- The kernel starts a timer when sending the request.
- If the timer expires before a reply or acknowledgement comes back, the kernel sends the message again.

3) Lost Reply messages

- If no reply is forthcoming within a reasonable period, just send the request once more.
- A request which has no side effects and can be executed as often as necessary without any harm being done is said to be idempotent (Ex: asking for the first 1024 bytes of a file). Request such as transferring money is not idempotent
- Try to structure all requests in an idempotent way
- Another method is to have the client's kernel assign each request a sequence number
- To have a bit in the message header that is used to distinguish initial requests from retransmissions

4) Server Crashes

- At least once semantics
    - Wait until the server reboots (or rebinds to a new server) and try the operation again.
    - Guarantees that the RPC has been carried out at least one time, but possibly more.
- At most once semantics
    - Gives up immediately and reports back failure.
    - Guarantees that the RPC has been carried out at most one time, but possibly none at all.
- Exactly once semantics
    - The client gets no help and no promises.
    - Guarantee nothing

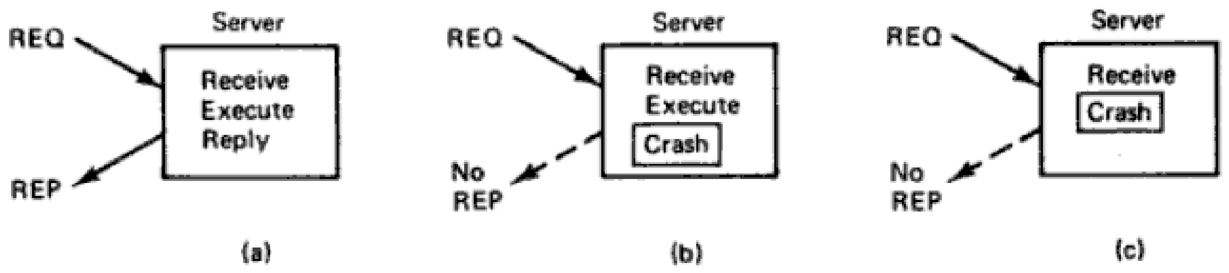


Fig. 2-24. (a) Normal case. (b) Crash after execution. (c) Crash before execution.

5) Client Crashes

- Client sends a request to a server to do some work and crashes before the server replies
- A computation which is active and no parent is waiting for the result is called an orphan
- They waste CPU cycles

Solutions

- Extermination
    - A log is kept on a medium that could survive the crash
    - The log is examined after a reboot to detect the orphans
    - Orphan is explicitly killed off
- Reincarnation
    - When a client reboots, it broadcasts a message to all machines declaring the start of a new epoch.
    - When such a broadcast comes in, all remote computations are killed
- Gentle reincarnation
    - When an epoch broadcast comes in, each machine checks to see if it has any remote computations
    - If so, tries to locate their owner.
    - Only if the owner cannot be found is the computation killed.
- Expiration
    - Each RPC is given a standard amount of time T to do the job
    - If it can not finish, it must explicitly ask for another quantum

RPC IMPLEMENTATION ISSUES

- RPC Protocol
    - Connection Oriented or Connectionless
    - Whether to use a general purpose protocol (like IP & UDP) or one specifically designed for RPC
- Acknowledgements
    - Should individual packets be acknowledged or not
    - Stop-and-wait protocol
        - Client send packet 0 with 1st 1K
        - Wait for ack from server
        - Then send the 2nd 1K
    - Blast Protocol
        - Client sends all packets as fast as it can
        - Server decides whether to process or do nothing
        - It can perform selective repeat as ask for specific packet

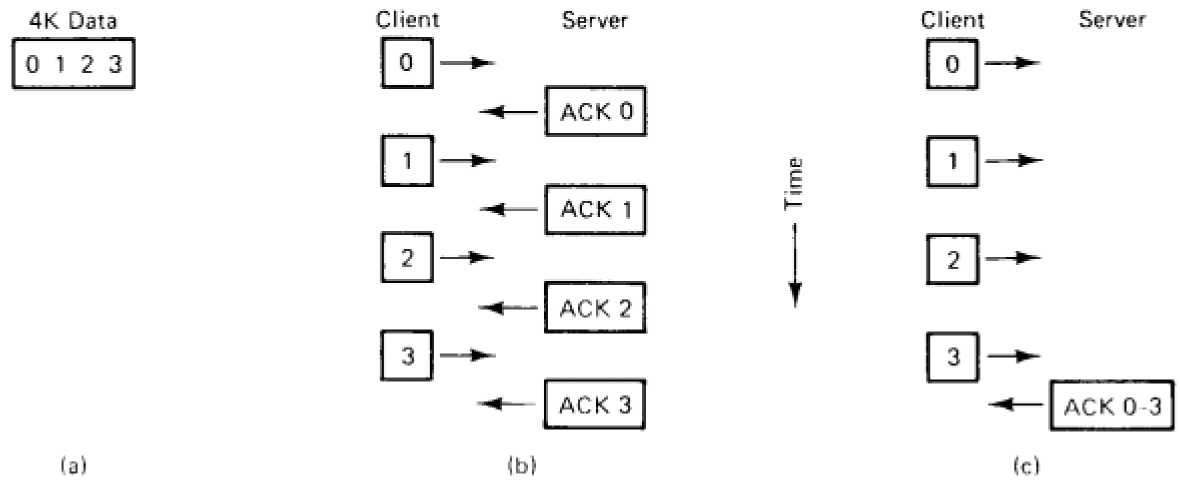


Fig. 2-25. (a) A 4K message. (b) A stop-and-wait protocol. (c) A blast protocol.

- Critical Path
    - The sequence of instructions that is executed on every RPC is called the critical path
    - Copying
    - The number of times a message must be copied from kernel and user address
    - Depends on hardware, software and type of call

1. Call stub
2. Get message buffer
3. Marshal parameters
4. Fill in headers
5. Compute UDP checksum
6. Trap to kernel
7. Queue packet for transmission
8. Move packet to controller over the QBus
9. Ethernet transmission time
10. Get packet from controller
11. Interrupt service routine
12. Compute UDP checksum
13. Context switch to user space
14. Server stub code

- Timer Management
    - The amount of machine time that goes into managing the timers
    - Algorithms that operate by periodically making a sequential pass through a table are called sweep algorithms

GROUP COMMUNICATION – OUT OF SYLLABUS

- A group is a collection of processes that act together in some system or user-specified way
- When a message is sent to the group itself, all members of the group receive it
- It is a form of one-to-many communication (one sender, many receivers)

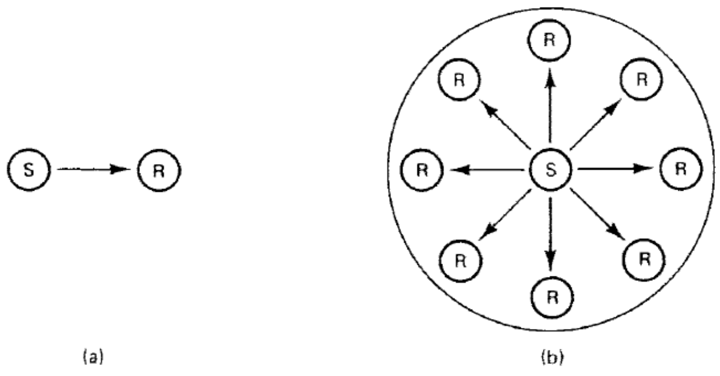


Fig. 2-30. (a) Point-to-point communication is from one sender to one receiver. (b) One-to-many communication is from one sender to multiple receivers.

Group communication implementation

- Multicasting
  - create a special network address to which multiple machines can listen
  - (example, indicated by setting one of the high-order bits to 1),
  - When a packet is sent to one of these addresses, it is automatically delivered to all machines listening to the address
- Broadcasting
  - packets containing a certain address (e.g., 0) are delivered to all machines
  - software must check to see if the packet is intended for it else the packet is discarded
- Unicasting
  - sender transmit separate packets to each of the members of the group
  - For a group with n members, n packets are required

Design Issues in Group Communication

- Closed Groups versus Open Groups
- Peer Groups versus Hierarchical Groups
- Group Membership
- Group Addressing
- Send and Receive Primitives
- Atomicity
- Message Ordering
- Scalability

Closed Groups versus Open Groups

- Closed groups
  - only the members of the group can send to the group.
  - Outsiders cannot send messages to the group as a whole, although they may be able to send messages to individual members.
  - typically used for parallel processing
- Open groups
  - any process in the system can send to any group.

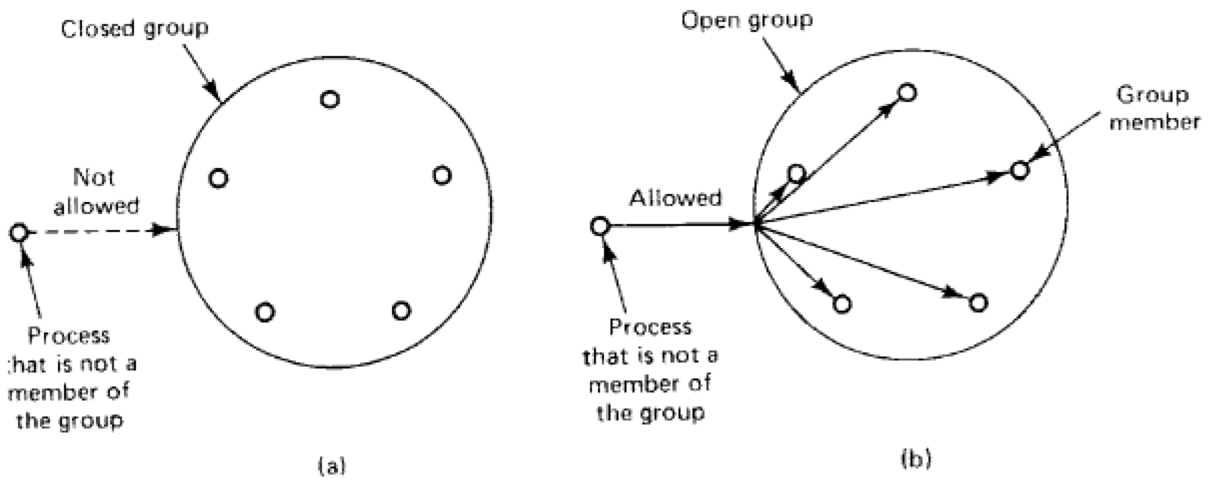

Closed group
Not allowed
Process that is not a member of the group
(a)
Open group
Group member
Allowed
Process that is not a member of the group
(b)

Fig. 2-31. (a) Outsiders may not send to a closed group. (b) Outsiders may send to an open group.

Peer Groups versus Hierarchical Groups

- Peer groups
  - all the processes are equal.
  - No one is boss and all decisions are made collectively.
  - Adv: symmetric and has no single point of failure
  - Disadv: decision making is more complicated. To decide anything, a vote has to be taken, incurring some delay and overhead
- Hierarchical groups
  - some kind of hierarchy exists.
  - one process is the coordinator and all the others are workers

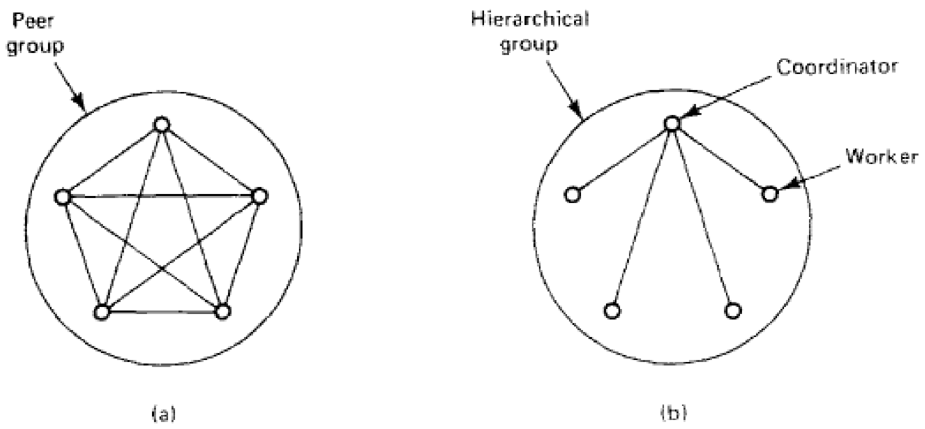


Fig. 2-32. (a) Communication in a peer group. (b) Communication in a simple hierarchical group.

Group Membership

- creating and deleting groups, as well as for allowing processes to join and leave groups
- Managing group membership can be done in two ways
- Group server
  - The group server can then maintain a complete data base of all the groups and their exact membership.
  - This method is straightforward, efficient, and easy to implement
  - If the group server crashes, group management ceases to exist
- Distributed
  - an outsider can send a message to all group members announcing its presence.
  - To leave a group, a member just sends a goodbye message to everyone.
  - member crashes
    - no polite announcement
    - leaving and joining have to be synchronous with messages being sent
- if so many machines go down that the group can no longer function at all
  - Some protocol is needed to rebuild the group

Group Addressing

Three ways of addressing

- give each group a unique address
- require the sender to provide an explicit list of all destinations (e.g., IP addresses)
- predicate addressing
  - each message is sent to all members of the group
  - contains a predicate (Boolean expression) to be evaluated
  - If the predicate evaluates to TRUE, the message is accepted.

- If it evaluates to FALSE, the message is discarded.

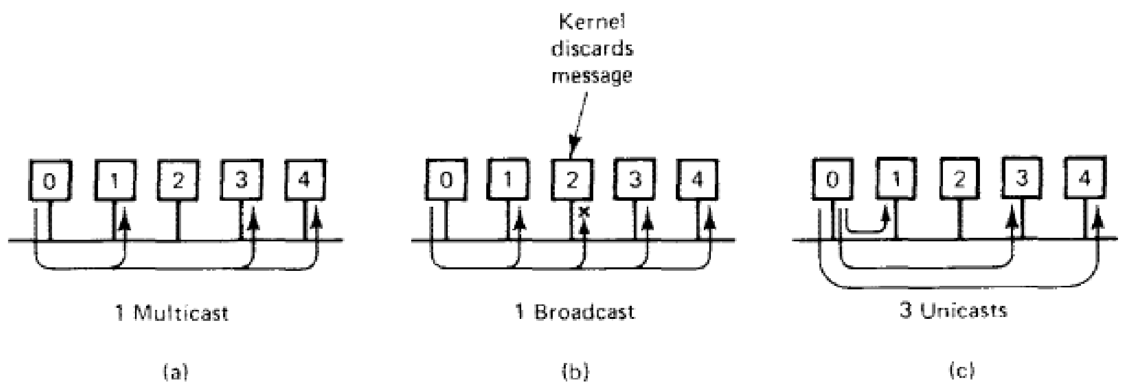


Fig. 2-33. Process 0 sending to a group consisting of processes 1, 3, and 4. (a) Multicast implementation. (b) Broadcast implementation. (c) Unicast implementation.

Send and Receive Primitives

- send, receive
- RPC
  - the client sends one message to the server and gets back one answer.
- Group communication
  - n different replies
- Buffered / unbuffered, blocking or non-blocking, reliable or not reliable
- group_send, group_receive

Atomicity

- when a message is sent to a group, it will either arrive correctly at all members of the group, or at none of them.
- all-or-nothing delivery is called atomicity or atomic broadcast
- When any process sends a message to the group, it does not have to worry about what to do if some of them do not get it.
- every destination receives every message is to require them to send back an acknowledgement upon message receipt.
- The sender starts out by sending a message to all members of the group.
- Timers are set and retransmissions sent where necessary.
- When a process receives a message, if it has not yet seen this particular message, it, too, sends the message to all members of the group, resulting in a reliable delivery

Message Ordering

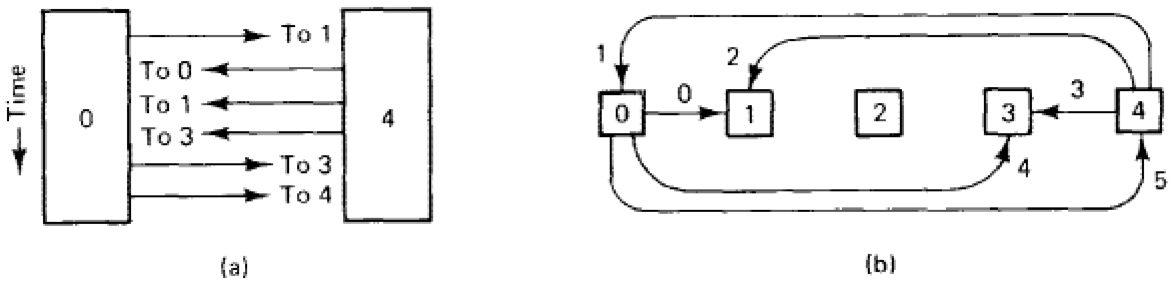


Fig. 2-34. (a) The three messages sent by processes 0 and 4 are interleaved in time. (b) Graphical representation of the six messages, showing the arrival order.

- global time ordering
  - delivers all messages in the exact order in which they were sent
- consistent time ordering
  - if two messages, say A and B, are sent close together in time, the system picks one of them as being "first" and delivers it to all group members, followed by the other.

Overlapping Groups

although there is a global time ordering within each group, there is not necessarily any coordination among multiple groups

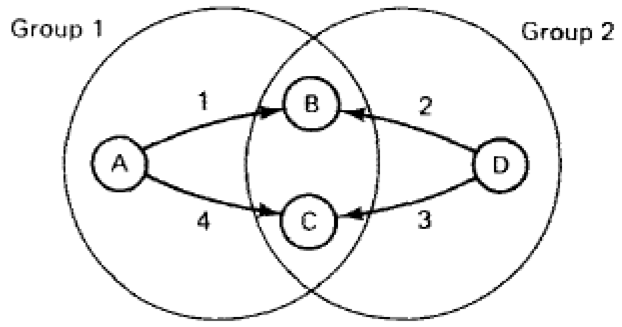


Fig. 2-35. Four processes, *A*, *B*, *C*, and *D*, and four messages. Processes *B* and *C* get the messages from *A* and *D* in a different order.

Scalability

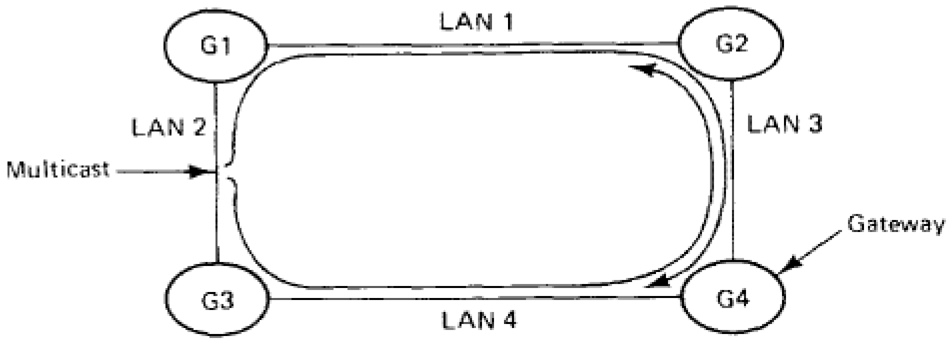


Fig. 2-36. Multicasting in an internetwork causes trouble.

- Many algorithms work fine as long as all the groups only have a few members, but what happens when there are tens, hundreds, or even thousands of members per group?
- Or thousands of groups? Also, what happens when the system is so large that it no longer fits on a single LAN, so multiple LANs and gateways are required?
- And what happens when the groups are spread over several continents?

Comments & Feedback

Thanks to my family members who supported me while I spent hours and hours to prepare this.
Your feedback is welcome at GHCRajan@gmail.com