Chapter 4: Communication

Overview

- 4.1 Foundations
- 4.2 Remote Procedure Call
- 4.3 Message-Oriented Communication
- 4.4 Multicast Communication
- 4.5 Case Study: Java RMI and Message
- Passing Interface (MN)

Introduction

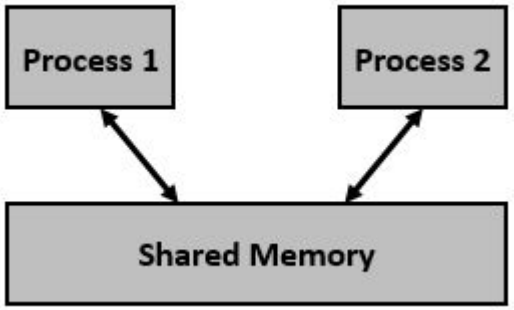
- In a distributed system, processes run on different machines.

- Processes can only exchange information through message passing.

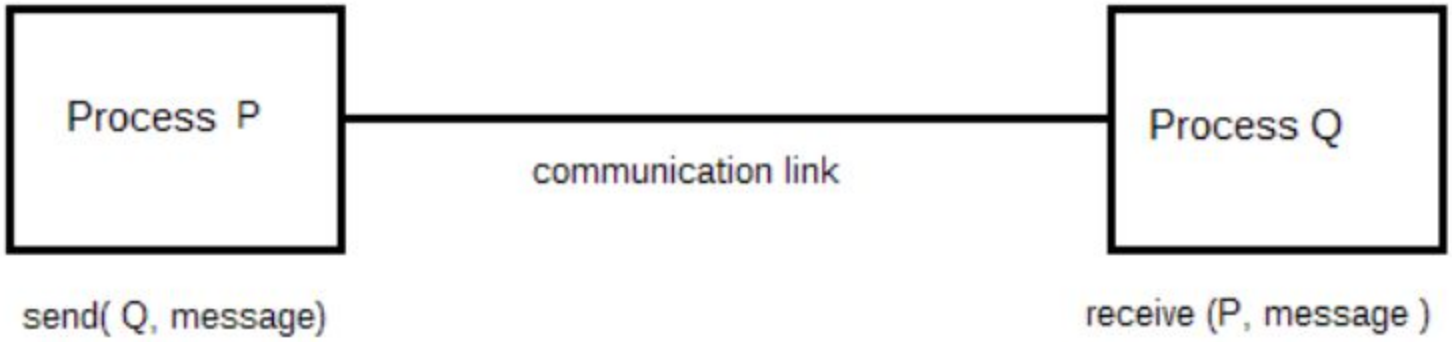
Introduction

- A communication network provides data exchange between two (or more) end points.
- Early examples: telegraph or telephone system.
- In a computer network, the end points of the data exchange are computers and/or terminals. (nodes, sites, hosts, etc., ...)
- Networks can use switched, broadcast, or multicast technology

- In a uniprocessor system, interprocess communication assumes the existence of shared memory.
- A typical example is the producer-consumer problem.



- In a distributed system, there's no shared memory.
- All communication in distributed system is based on **message passing**.
- Three widely-used model for communication:
  - RPC
  - Message –Oriented Middleware
  - Data streaming


Process P
communication link
Process Q
send(Q, message)
receive (P, message)

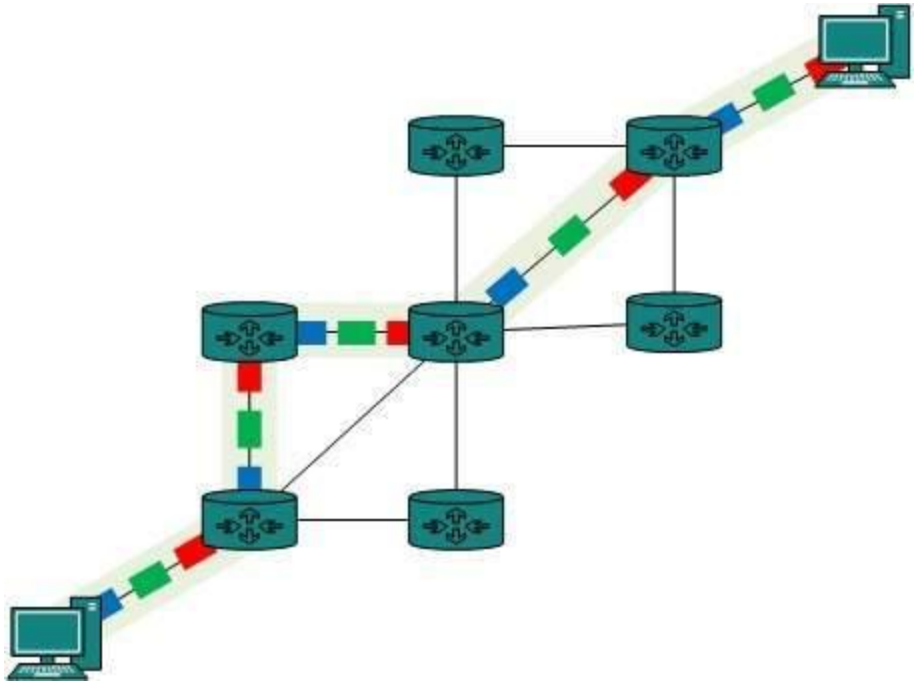
Fundamentals of Communication

Network Communication Technologies

1. Switched Networks

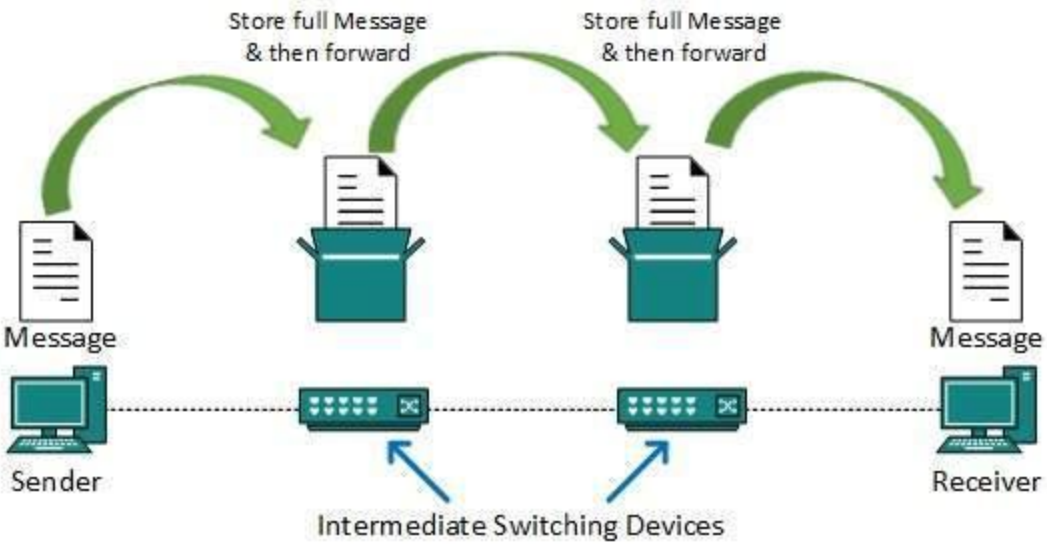
- Switched communication networks are those in which data transferred from source to destination is routed between various intermediate nodes.
- Switching is the technique by which nodes control or switch data to transmit it between specific points on a network. There are 3 common switching techniques:
- Circuit Switching
- Packet Switching
- Message Switching

- Circuit Switching
- When two nodes communicate with each other over a dedicated communication path, it is called circuit switching.
- There 'is a need of pre-specified route from which data will travels and no other data is permitted.
- In circuit switching, to transfer the data, circuit must be established so that the data transfer can take place.
- Circuits can be permanent or temporary. Applications which use circuit switching may have to go through three phases:
- Establish a circuit
- Transfer the data

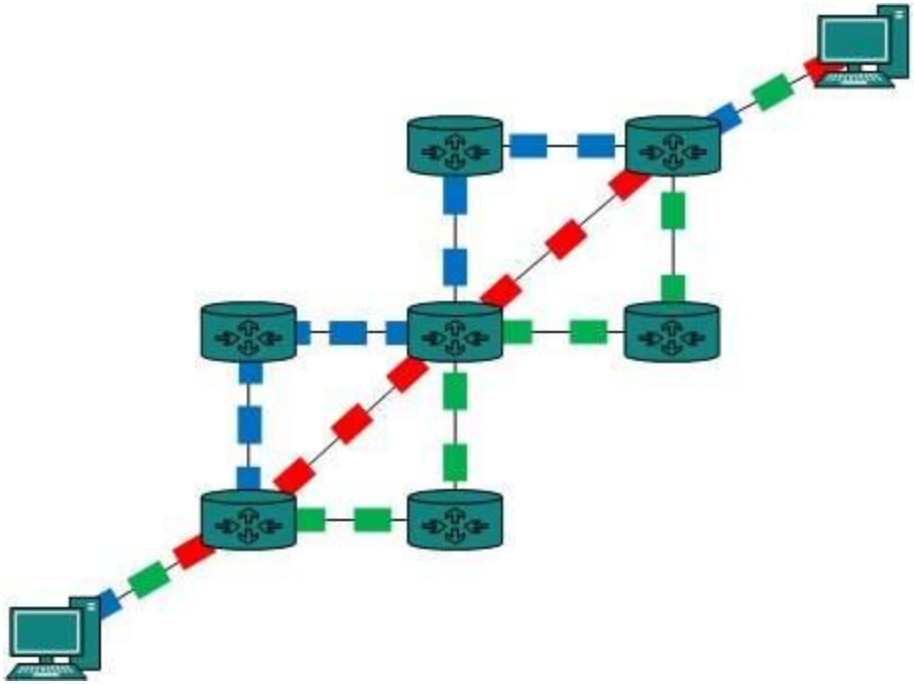


Message Switching

- This technique was somewhere in middle of circuit switching and packet switching.
- In message switching, the whole message is treated as a data unit and is switching / transferred in its entirety.
- A switch working on message switching, first receives the whole message and buffers it until there are resources available to transfer it to the next hop.
- If the next hop is not having enough resource to accommodate large size message, the message is stored and switch

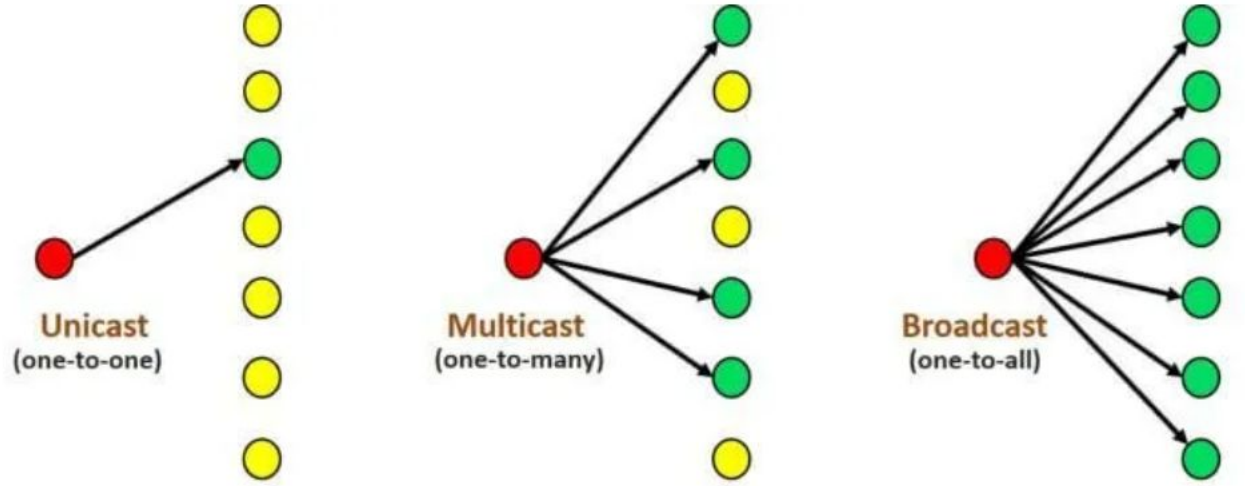

Store full Message
& then forward
Store full Message
& then forward
Message
Message
Sender
Receiver
Intermediate Switching Devices

- Shortcomings of message switching gave birth to an idea of packet switching.
- The entire message is broken down into smaller chunks called packets.
- The switching information is added in the header of each packet and transmitted independently.
- It is easier for intermediate networking devices to store small size packets and they do not take much resources either on carrier path or in the internal memory of switches.



Types of Communications

- **Unicast** means **one-to-one**, data send to only one device means sender sends data to only one device.
- **Multicast** means **one-to-many** (or many-to-many), data sent to multiple devices means sender sends data to many devices (not all devices like the broadcast).
- **Broadcast** means **one-to-all**, data sent to all devices means sender sends data to all devices.


Unicast
(one-to-one)
Multicast
(one-to-many)
Broadcast
(one-to-all)

LAN ,WAN, MAN

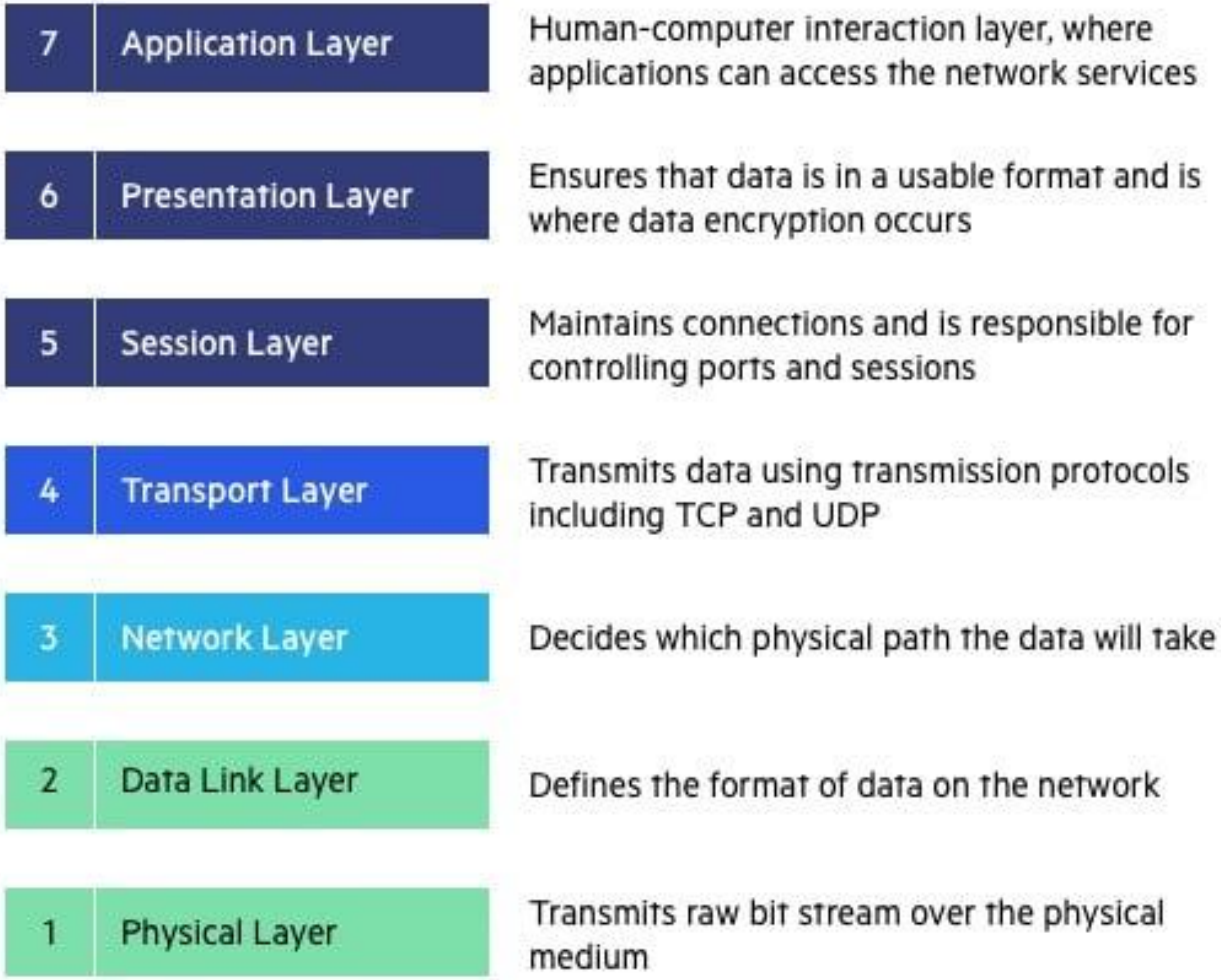
Protocols

- A protocol is a set of rules that defines how two entities interact.
  - For example: HTTP, FTP, TCP/IP,

Open Systems Interconnection Reference Model (OSI)

- OSI stands for **Open System Interconnection** is a reference model that describes how information from a software application in one computer moves through a physical medium to the software application in another computer.

- OSI consists of seven layers, and each layer performs a particular network function.
- OSI model was developed by the International Organization for Standardization (ISO) in 1984, and it is now considered as an architectural model for the inter-computer communications.
- OSI model divides the whole task into seven smaller and manageable tasks. Each layer is assigned a particular task.


7
Application Layer
Human-computer interaction layer, where applications can access the network services
6
Presentation Layer
Ensures that data is in a usable format and is where data encryption occurs
5
Session Layer
Maintains connections and is responsible for controlling ports and sessions
4
Transport Layer
Transmits data using transmission protocols including TCP and UDP
3
Network Layer
Decides which physical path the data will take
2
Data Link Layer
Defines the format of data on the network
1
Physical Layer
Transmits raw bit stream over the physical medium

Lower-level Protocols

- **Physical**: standardizes electrical, mechanical, and signaling interfaces; e.g.,
  - # of volts that signal 0 and 1 bits
  - # of bits/sec transmitted
  - Plug size and shape, # of pins, etc.
- **Data Link**: provides low-level error checking
  - Appends start/stop bits to a frame
  - Computes and checks checksums
- **Network**: routing (generally based on IP)
  - IP packets need no setup
  - Each packet in a message is routed independently

Transport Protocols

- **Transport layer**: Receives message from higher layers, divides into packets, assigns sequence #
- Reliable transport (connection-oriented) can be built on top of connection-oriented or connectionless networks
  - When a connectionless network is used the transport layer re-assembles messages in order at the receiving end.
- Most common transport protocols: TCP/IP

TCP/IP Protocols

- Developed originally for Army research network ARPANET.
- Major protocol suite for the Internet
- Can identify 4 layers, although the design was not developed in a layered manner:
  - Application (FTP, HTTP, etc.)
  - Transport: TCP & UDP
  - IP: routing across multiple networks (IP)

Reliable/Unreliable Communication

- TCP guarantees reliable message transmission even if packets are lost or delayed.
- Packets must be acknowledged by the receiver– if ACK not received in a certain time period, resend.
- Reliable communication is considered *connection-oriented* because it "looks like" communication in circuit switched networks.

Reliable/Unreliable Communication

- For applications that value speed over absolute correctness, TCP/IP provides a *connectionless* protocol: UDP
  - UDP = User Datagram Protocol

- **Session layer**: rarely supported
  - Provides dialog control;
  - Keeps track of who is transmitting
- **Presentation**: also not generally used
  - Cares about the meaning of the data
    - Record format, encoding schemes, mediates between different internal representations
- **Application**: Originally meant to be a set of basic services; now holds applications and protocols that don't fit elsewhere

Middleware

- Middleware is software that lies between an operating system and the applications running on it.

- Essentially functioning as hidden translation layer, middleware enables communication and data management for distributed applications.

- It's sometimes called plumbing, as it connects two applications together so data and databases can be easily passed between the "pipe."

- Using middleware allows users to perform such requests as submitting forms on a web browser, or allowing the web server to return dynamic web pages based on a user's profile.

Apart from communication function , other function include :

security authentication,

transaction management,

message queues,

applications servers, web servers, and directories.

Middleware Protocols

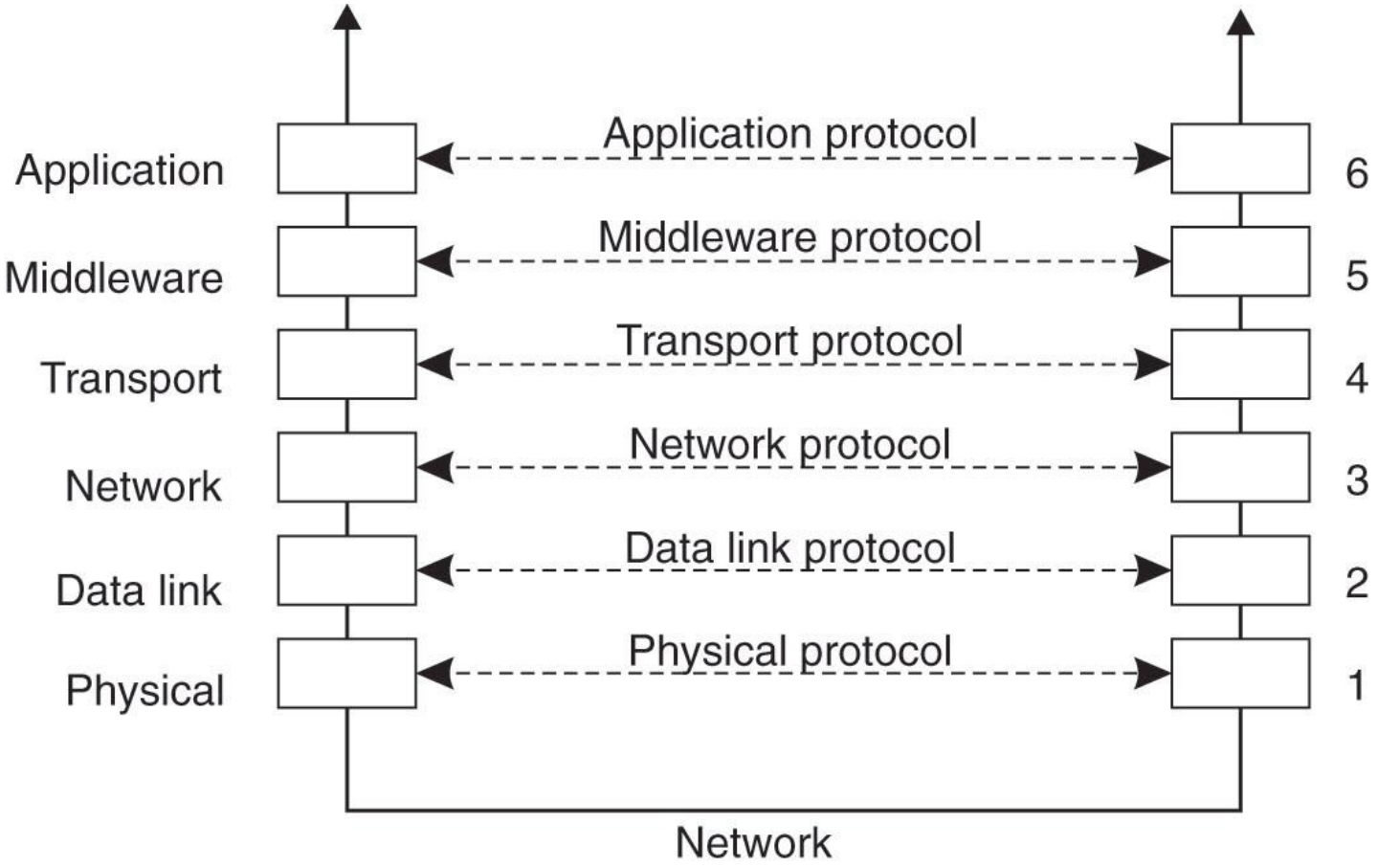


Figure 4-3. An adapted reference model for networked communication.

Middleware Communication Techniques

- Remote Procedure Call
- Message-Oriented Communication
- Stream-Oriented Communication
- Multicast Communication

1. Remote Procedure Call

- A remote procedure call is an interprocess communication technique that is used for client-server based applications.
- It is also known as a subroutine call or a function call.
- A client has a request message that the RPC translates and sends to the server.

- This request may be a procedure or a function call to a remote server.
- When the server receives the request, it sends the required response back to the client.
- The client is blocked while the server is processing the call and only resumed execution after the server is finished.

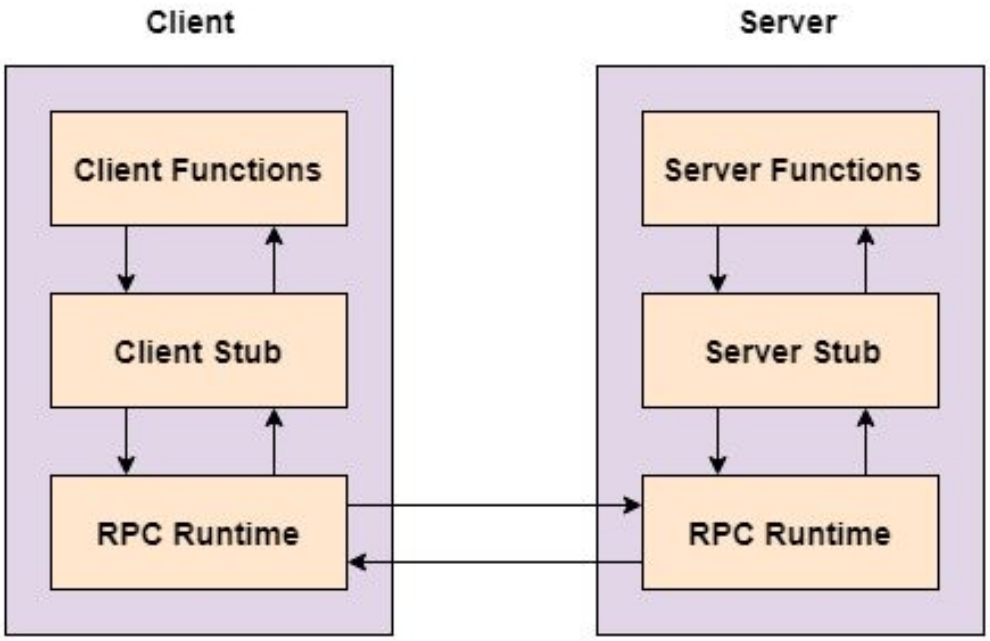
NOTE:

A stub in distributed computing is a piece of code that converts parameters passed between client and server during a remote procedure call.

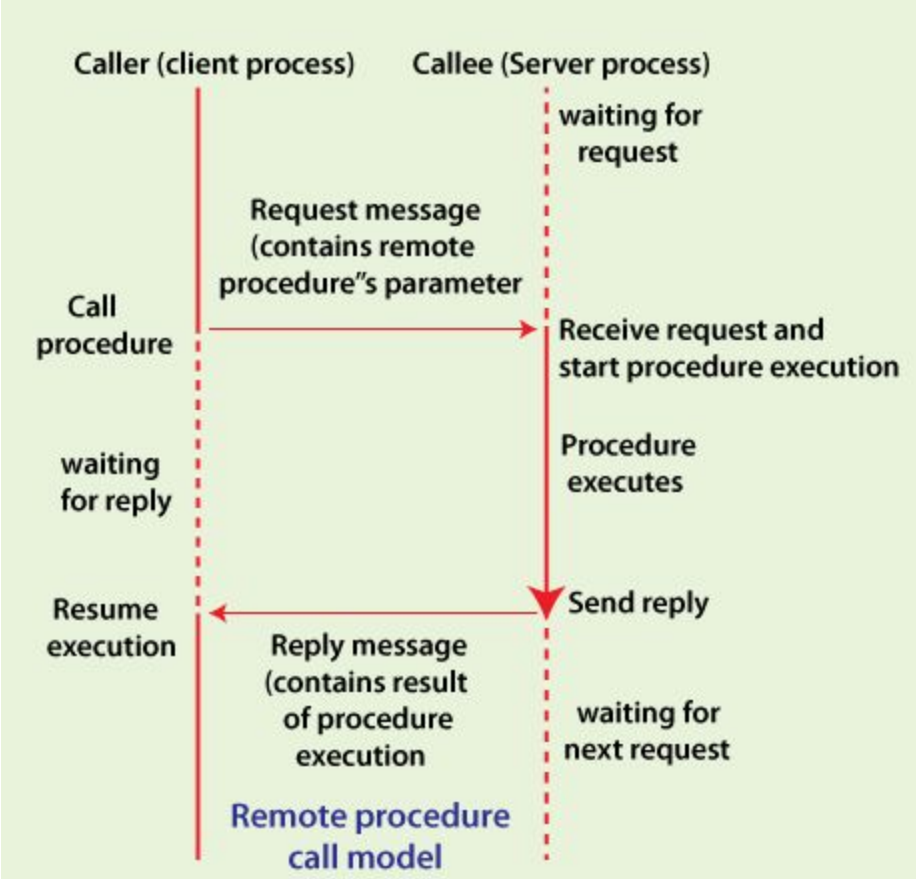
The sequence of events in a remote procedure call are given as follows –

- The client stub is called by the client.
- The client stub makes a system call to send the message to the server and puts the parameters in the message.

- The message is sent from the client to the server by the client's operating system.
- The message is passed to the server stub by the server operating system.
- The parameters are removed from the message by the server stub.
- Then, the server procedure is called by the server stub.


Client
Server
Client Functions
Client Stub
RPC Runtime
Server Functions
Server Stub
RPC Runtime

RPC Mechanism



Passing Value Parameters

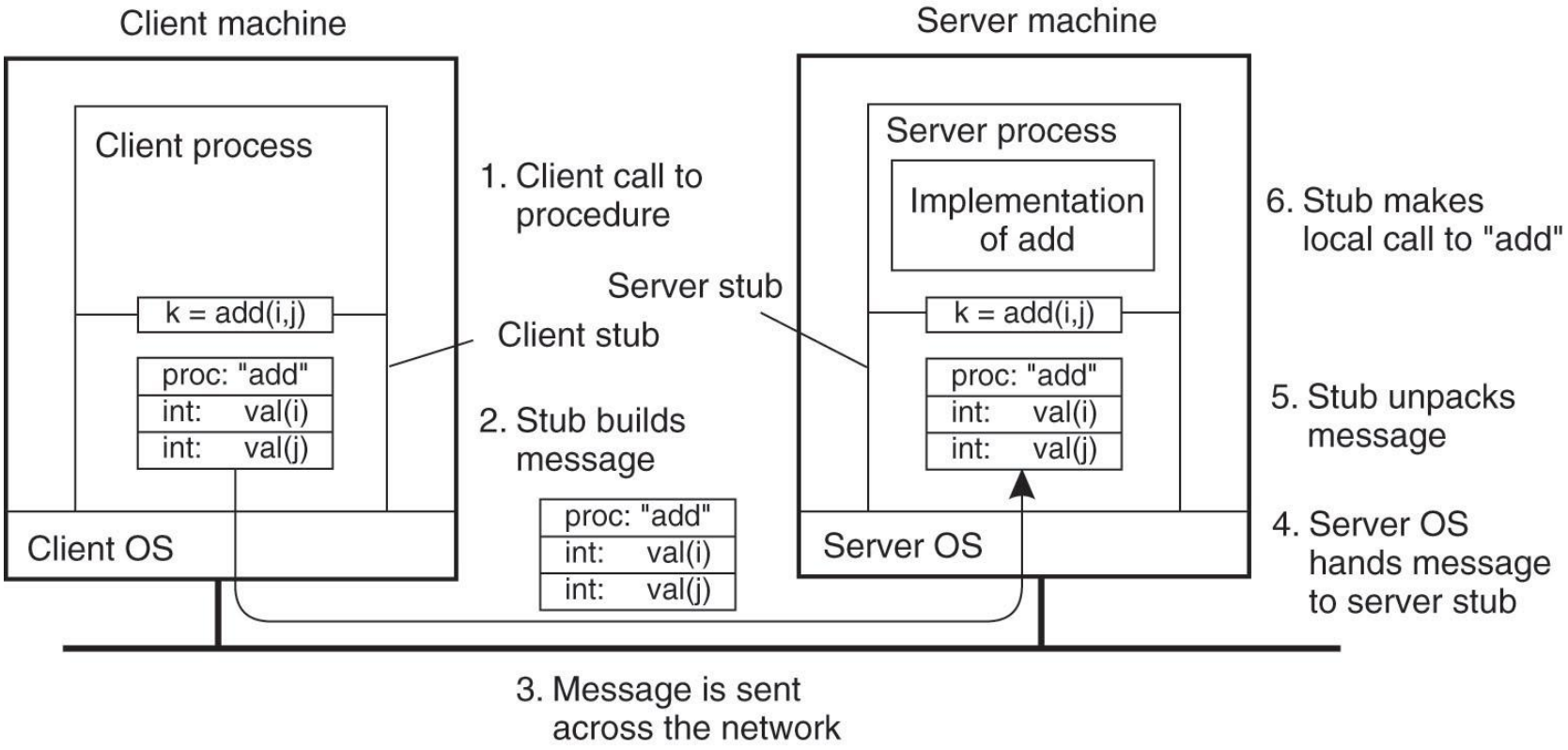


Figure 4-7. The steps involved in a doing a remote computation through RPC.

Advantages of Remote Procedure Call

- RPC support process oriented and thread oriented models.
- The internal message passing mechanism of RPC is hidden from the user.
- The effort to re-write and re-develop the code is minimum in remote procedure calls.
- RPC can be used in distributed environment as well as the local environment.
- Many of the protocol layers are omitted by RPC to improve performance.

Disadvantages of Remote Procedure Call

- The remote procedure call is a concept that can be implemented in different ways. It is not a standard.
- There is no flexibility in RPC.
- There is an increase in costs because of remote procedure call.

Protocols to Support Services

- Authentication protocols, to prove identity
- Authorization protocols, to grant resource access to authorized users
- Distributed commit protocols, used to allow a group of processes to decided to commit or abort a transaction (ensure atomicity)
- Locking protocols to ensure mutual exclusion on a shared resource in a distributed environment.

Middleware Protocols to Support Communication

- Protocols for remote procedure call (RPC) or remote method invocation (RMI)
- Protocols to support message-oriented services
- Protocols to support streaming real-time data, as for multimedia applications
- Protocols to support reliable multicast service across a wide-area network

These protocols are built on top of low-level message passing, as supported by the transport layer.

2. The Message Passing Model

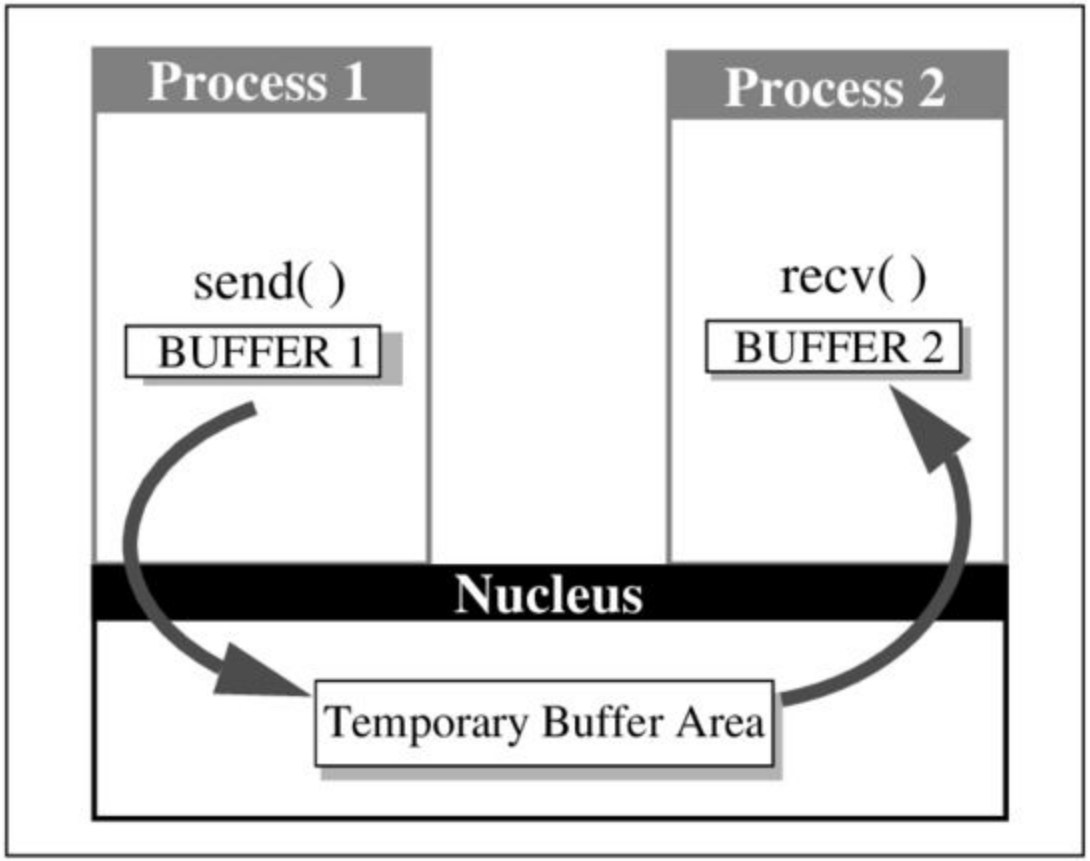
- This is the fundamental process that underlies other techniques for network communication.
- SEND and RECEIVE are the basic primitives
- SEND(*destination, msg*), where *destination* designates the receiver of the message, and *msg* is a pointer to the actual message.
- RECEIVE(*source, msg*), where source is the sender and *msg* is a pointer to the location where the incoming message should be stored.


Process P
communication link
Process Q
send(Q, message)
receive (P, message)

Buffered Message Passing

In this implementation a message is copied three times.

- from the sender's message buffer to a buffer in the communication software (middleware or OS kernel)
- from the middleware/kernel buffer on the sender's machine to the middleware/kernel buffer on the receiving machine
- from the middleware/kernel buffer to the receiver's buffer.


Process 1
Process 2
send( )
recv( )
BUFFER 1
BUFFER 2
Nucleus
Temporary Buffer Area

Types of Communication

- Persistent versus transient
- Synchronous versus asynchronous
- Discrete versus streaming

Persistent versus Transient Communication

Transient communication: Comm. server discards message when it cannot be delivered at the next server, or at the receiver.

Persistent communication: A message is stored at a communication server as long as it takes to deliver it.

Communication between caller & callee can be hidden by using procedure-call mechanism.

Synchronous versus Asynchronous Communication

- **Synchronous communication** happens when messages can only be exchanged in real time.
- It requires that the transmitter and receiver are present in the same time and/or space.
- Examples of synchronous communication are phone calls or video meetings.

- **Asynchronous communication** happens when information can be exchanged independent of time.

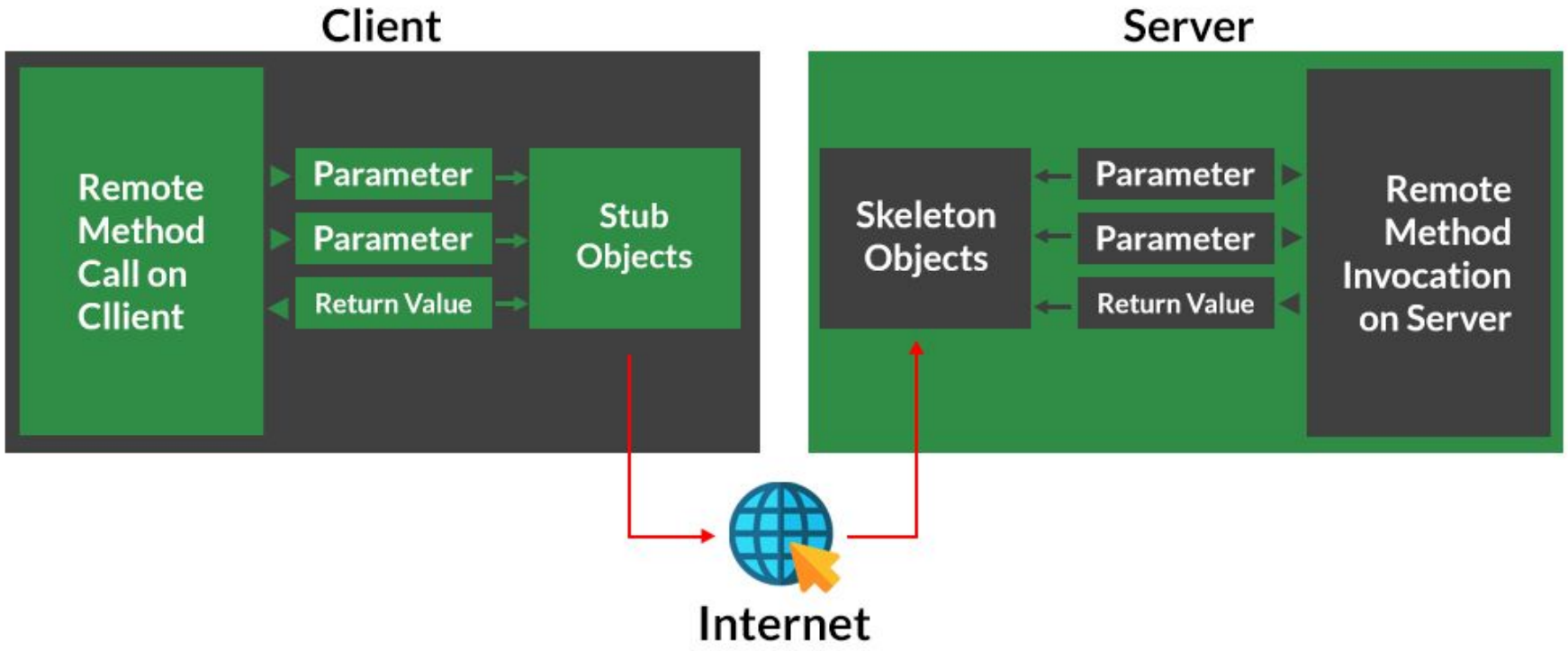
- It doesn't require the recipient's immediate attention, allowing them to respond to the message at their convenience.

- Examples of asynchronous communication are emails, online forums, and collaborative documents.

Remote Method Invocation (RMI)

- Similar to RPC; allows a Java process running on one virtual machine to call a method of an object running on another virtual machine
- Supports creation of distributed Java systems

Working of RMI



ASSIGNMENT: RMI

Sockets

- A communication endpoint used by applications to write and read to/from the network.
- Sockets provide a basic set of primitive operations
- Sockets are an abstraction of the actual communication endpoint used by local OS

Primitive	Meaning
Socket	Create new communication end point
Bind	Attach a local address to a socket
Listen*	Willing to accept *connections* (non-blocking)
Accept	Block caller until connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

How a Server Uses Sockets

Internetworking with TCP/IP, Douglas E. Comer & David L. Stevens, Prentice Hall, 1996

System Calls

- Socket
- Bind
- Listen
- Accept
- Read
- Write
- Close

Meaning

- Create socket descriptor
- Bind local IP address/ port # to the socket
- Place in passive mode, set up request queue
- Get the next message
- Read data from the network
- Write data to the network
- Terminate connection

How a Client Uses Sockets

Internetworking with TCP/IP, Douglas E. Comer & David L. Stevens, Prentice Hall, 1996

System Calls	Meaning
• Socket	• Create socket descriptor
• Connect	• Connect to a remote server
• Write	• Write data to the network
• Read	• Read data from the network
• Close	• Terminate connection

Socket Communication

- Using sockets, clients and servers can set up a connection-oriented communication session.
- Servers execute first four primitives (socket, bind, listen, accept) while clients execute socket and connect primitives)
- Then the processing is client/write, server/read, server/write, client/read, all close connection.

Message-Passing Interface (MPI)

Message Passing Interface (MPI) is a standardized and portable message-passing system designed for parallel computing.

It enables communication between multiple processes running on the same or different machines in a distributed memory environment.

Features of MPI:

1. **Process Communication** – Allows processes to exchange data using messages.
2. **Scalability** – Works efficiently on small and large clusters.
3. **Portability** – Runs on various platforms without modification.
4. **Performance** – Optimized for high-speed interconnects in HPC (High-Performance Computing).
5. **Fault Tolerance** – Some implementations support fault recovery mechanisms.

Common MPI Functions

1. Initialization and Finalization

- `MPI_Init(&argc, &argv);` → Initializes MPI environment.
- `MPI_Finalize();` → Cleans up the MPI environment.

2. Communication Between Processes

- **Point-to-Point Communication**
  - `MPI_Send()` → Sends a message.
  - `MPI_Recv()` → Receives a message.

2. Message-Oriented Communication

- Message-Oriented Communication is a communication paradigm where processes or applications exchange data through messages rather than direct function calls or shared memory.
-
- This approach is widely used in distributed systems, messaging middleware, and networked applicat

1. **Asynchronous Communication**
    - Senders and receivers do not need to interact simultaneously.
    - Messages are stored in queues until the receiver is ready.
2. **Decoupling**
    - Sender and receiver operate independently, improving scalability.
    - Useful in microservices, cloud computing, and distributed applications.
3. **Reliability & Persistence**
    - Messages can be stored (persisted) to ensure delivery, even if the recipient is temporarily unavailable.
4. **Message Brokers & Queues**
    - Middleware like **RabbitMQ**, **Apache Kafka**, **ActiveMQ**, and **ZeroMQ** manage message delivery.

Types of Message-Oriented Communication

1. **Point-to-Point (P2P) Messaging**
   - One sender, one receiver.
   - Example: **Message Queues** (e.g., RabbitMQ, IBM MQ).
2. **Publish-Subscribe (Pub/Sub) Model**
   - A sender (publisher) sends messages to multiple receivers (subscribers).
   - Example: **Kafka Topics**, **Redis Pub/Sub**.
3. **Request-Reply Messaging**
   - A sender sends a request and waits for a response.
   - Example: **Remote Procedure Call (RPC) over a message bus**.
4. **Multicast Messaging**
   - A message is sent to multiple recipients but only to interested ones.
   - Example: **UDP Multicast, MQTT**.

Advantages of Message-Oriented Communication

✅ **Scalability** – Decoupled architecture supports distributed systems.
✅ **Fault Tolerance** – Messages persist even if a service fails.
✅ **Asynchronous Processing** – Reduces blocking and increases efficiency.
✅ **Load Balancing** – Messages can be distributed across multiple consumers.

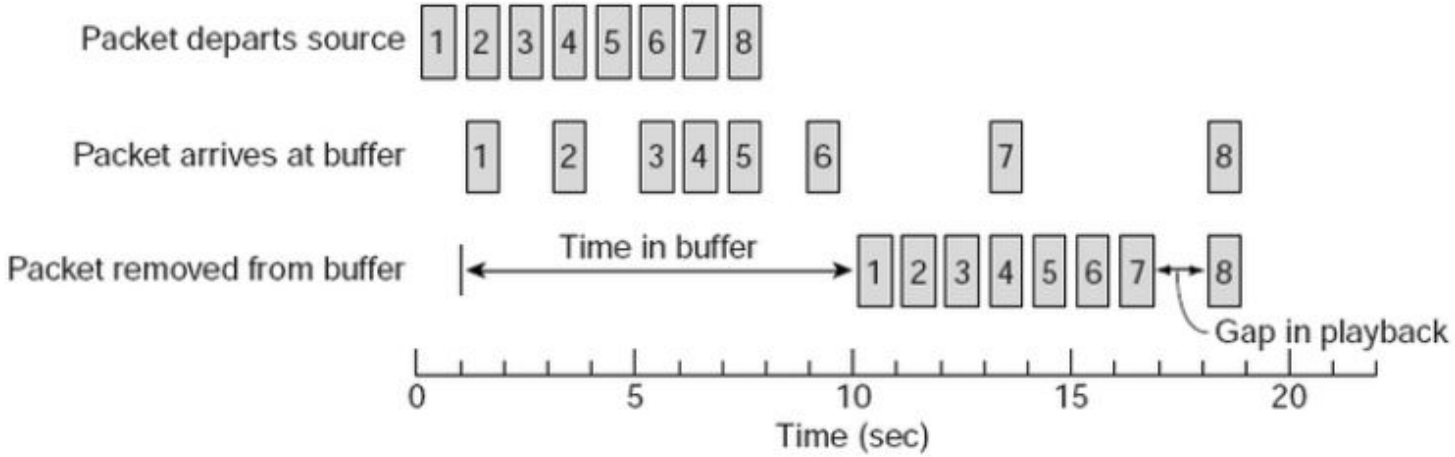
3. Stream-Oriented Communication

- Stream-oriented communication is a form of communication in which timing plays an important role.

- Stream-oriented communication is also referred to as continuous streams of data.

- In stream-oriented communication the message content must be delivered at a certain rate, as well as correctly.
  - e.g., music or video


STREAM ORIENTED


Packet departs source
1 2 3 4 5 6 7 8
Packet arrives at buffer
1 2 3 4 5 6 7 8
Packet removed from buffer
Time in buffer
1 2 3 4 5 6 7 8
Gap in playback
0 5 10 15 20
Time (sec)

Features

1. Supports for continuous media
2. Streams in distributed systems
3. Stream management

Characteristics

1. Streams are unidirectional.
2. Generally a single source, one or more sinks.
3. Often either sink/source is wrapped around hardware (e.g., Camera, CD device, Tv monitor).
4. Simplex Stream: Single way to flow data.
5. Complex stream: Multiple ways to flow data. Example: Video with subtitles.

Transmission mode

1. Synchronous
2. Asynchronous
3. Isochronous

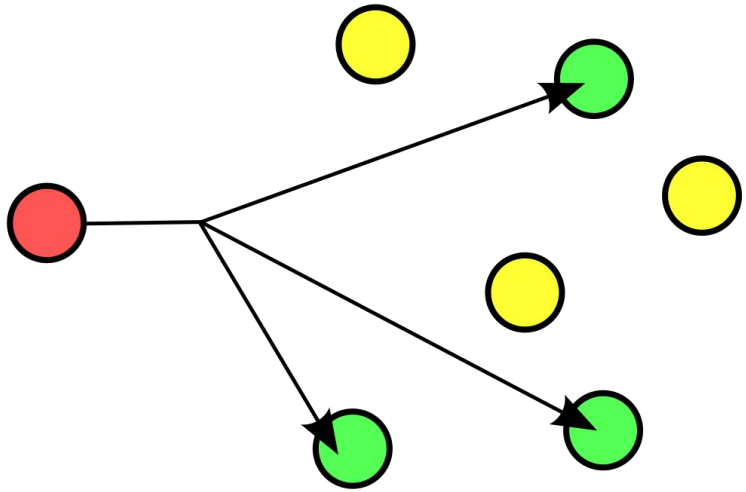
1. **Asynchronous:** This is transmission at any time, with arbitrary delay between transmission of any two successive data items.
2. **Synchronous:** This is continuous transmission with no gaps between transmission of successive data items.
3. **Isochronous:** This is transmission at regular intervals with a fixed gap between the transmission of successive data items.
4.

ASSIGNMENT 2

EXPLAIN **asynchronous synchronous and isochronous transmission IN detail.**

4. Multicast Communication

- Multicast: sending data to multiple receivers.
- Network- and transport-layer protocols for multicast bogged down at the issue of setting up the communication paths to all receivers.
- Peer-to-peer communication using structured overlays can use application-layer protocols to support multicast



Application-Level Multicasting