Unit 4: Communication

By Prashant Gautam

Outlines

Background

- Foundation
- RPC
- Message-Oriented Communication
- Multicast Communication
- Case Study: Java RMI and MPI

Background

- Inter-process communication is at the heart of all distributed systems.
- It makes no sense to study distributed systems without carefully examining the ways that processes on different machines can exchange information.
- Communication in distributed systems is always based on low-level message passing as offered by the underlying network.
- Expressing communication through message passing is harder than using primitives based on shared memory, as available for non-distributed platforms.

- Modern distributed systems often consist of thousands or even millions of processes scattered across a network with unreliable communication such as the Internet.
- Unless the primitive communication facilities of computer networks are replaced by something else, development of large-scale distributed applications is extremely difficult.

What we will study ?

- The rules that communicating processes must adhere to, known as protocols, and concentrate on structuring those protocols in the form of layers.
- Widely-used models for communication:
  - Remote Procedure Call (RPC),
  - Message-Oriented Middleware (MOM), and
  - data streaming.
- Multicasting concept: the general problem of sending data to multiple receivers

Before we start our discussion on communication in distributed systems, we first recapitulate some of the fundamental issues related to communication.

Communication Protocol

- Protocol: Set of rules on communication
- To allow a group of computers to communicate over a network, they must all agree on the protocols to be used.
- Protocols can be connectionless and connection oriented

Connection-oriented Protocol

- before exchanging data the sender and receiver first explicitly establish a connection.
- When they are done, they must release (terminate) the connection.
- The telephone is a connection-oriented communication system

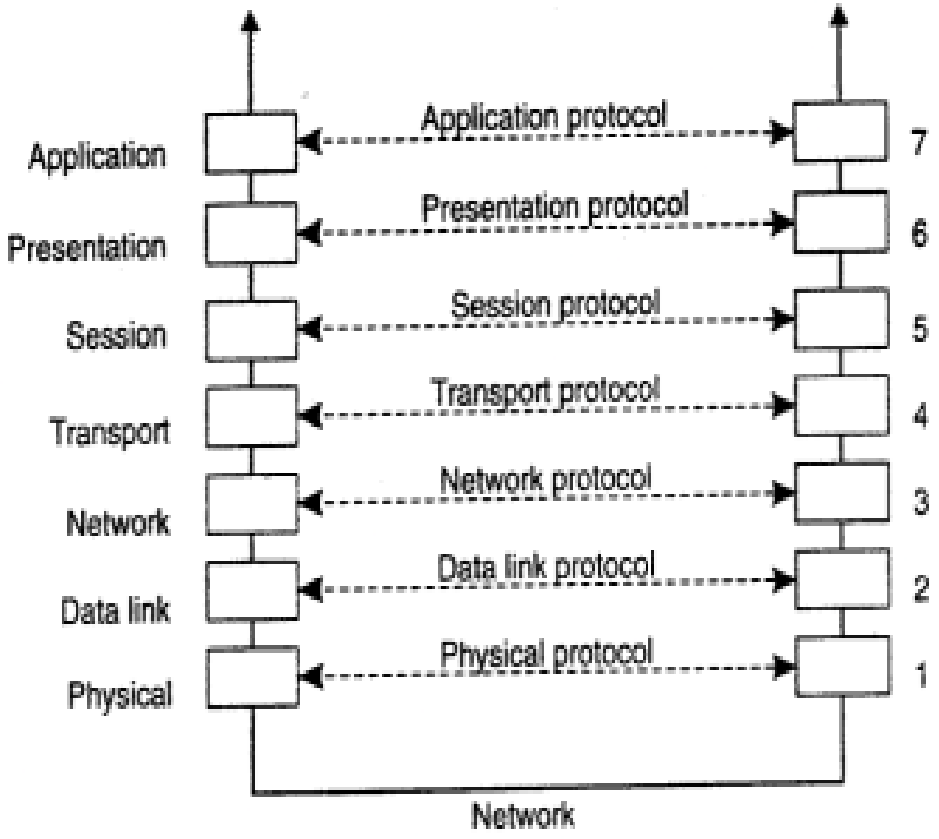
Connectionless Protocol

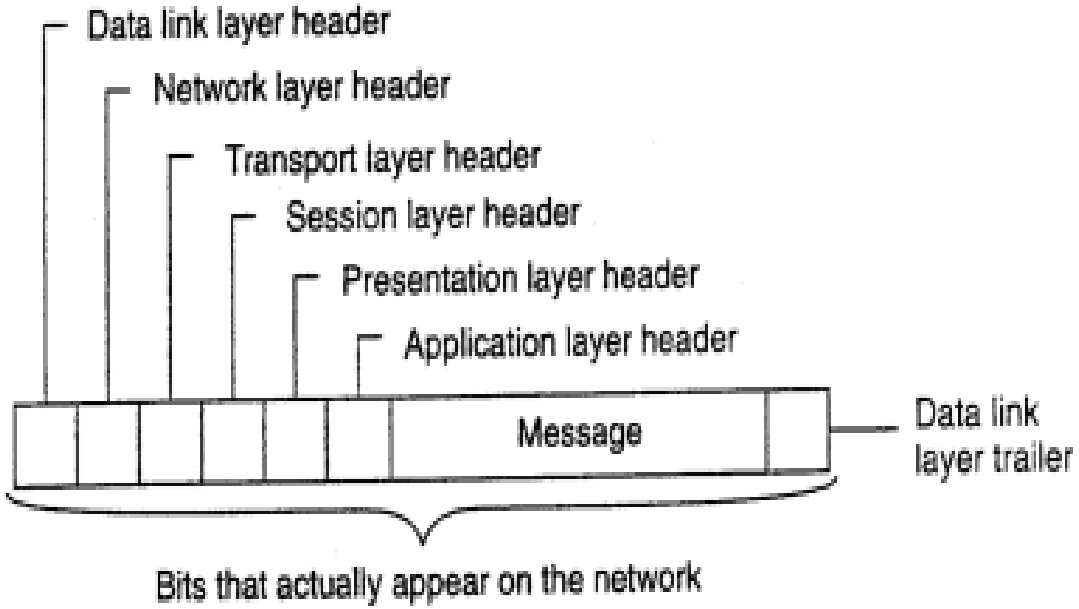
- No setup in advance is needed.
- The sender just transmits the first message when it is ready.
- Dropping a letter in a mailbox is an example of connectionless communication.

With computers, both connection-oriented and connectionless communication are common.

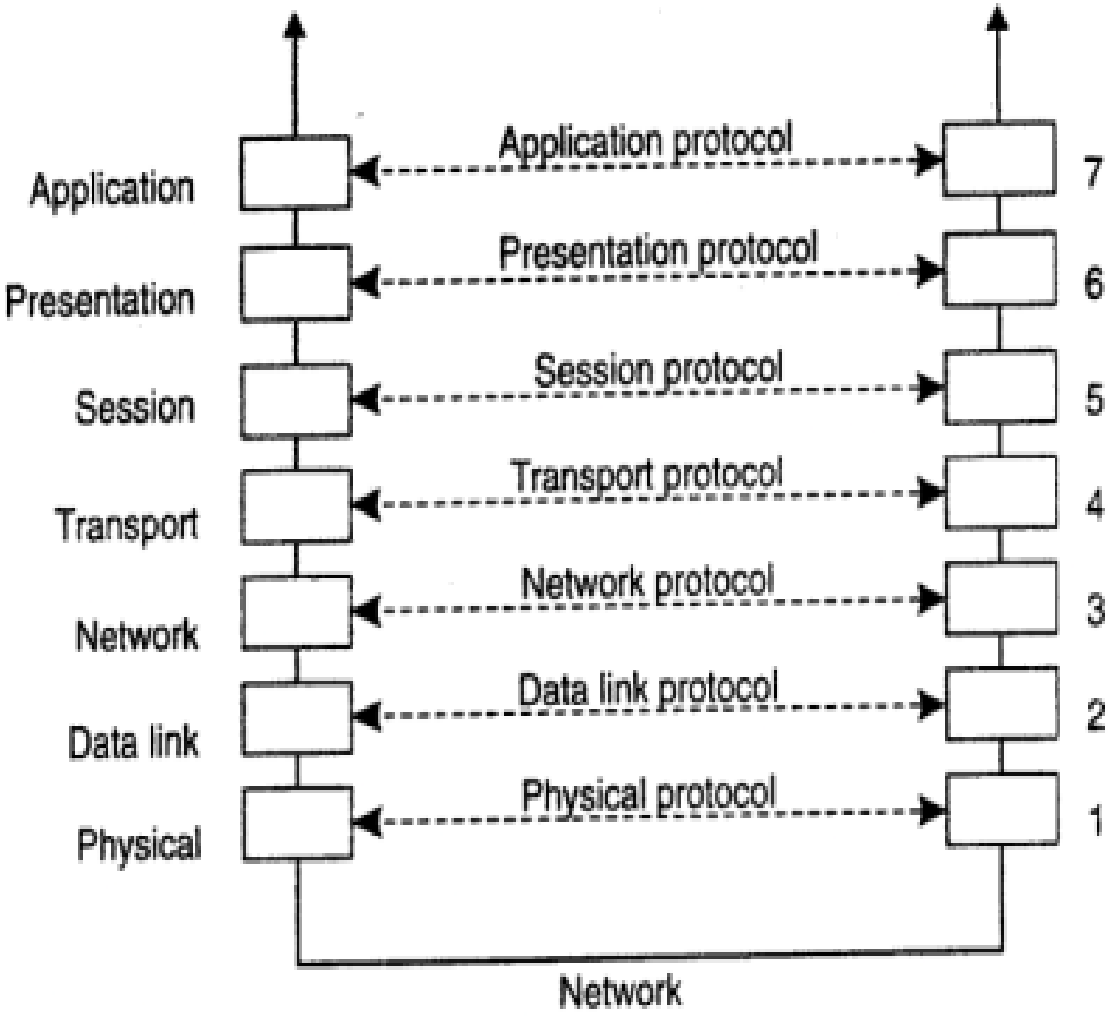
OSI model

- In the OSI model, communication is divided up into seven levels or layers, as shown in Fig.
- Each layer deals with one specific aspect of the communication.
- In this way, the problem can be divided up into manageable pieces, each of which can be solved independent of the others.
- Each layer provides an interface to the one above it.
- The interface consists of a set of operations that together define the service the layer is prepared to offer its users.

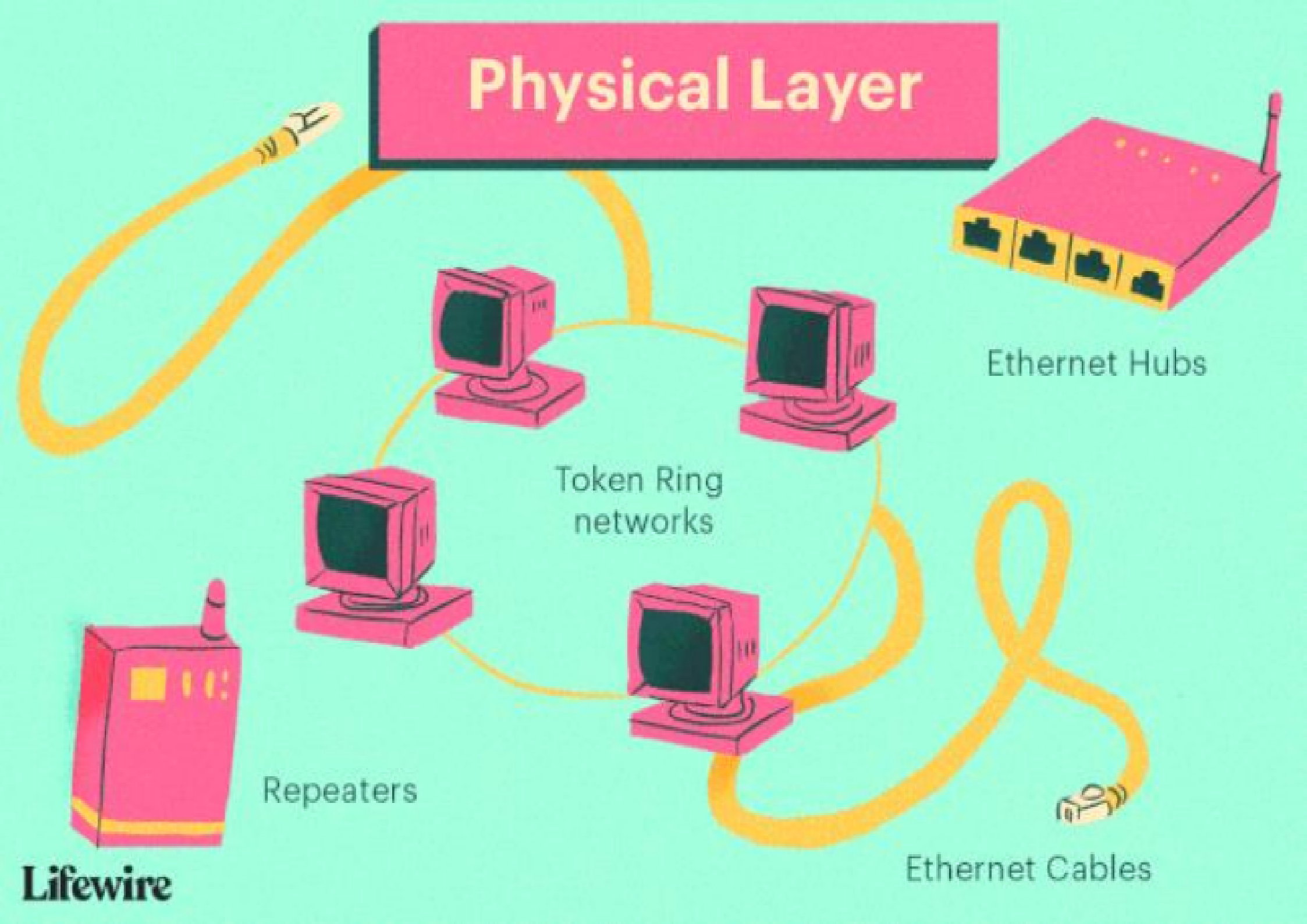




A typical message as it appears on the network



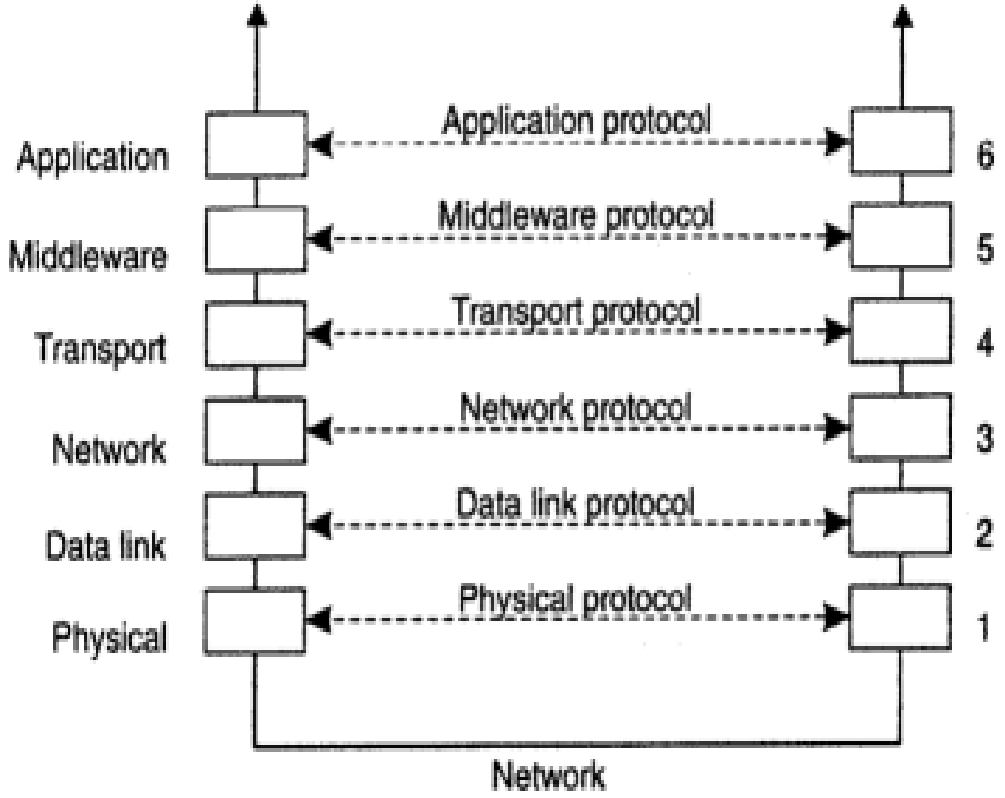
Layers, Interfaces, and Protocols in the OSI Model


Physical Layer
Ethernet Hubs
Token Ring networks
Repeaters
Ethernet Cables
Lifewire

Video

Protocols

- Lower Level Protocols
- Transport Protocols
- Higher- Level Protocols
- Middleware Protocols



Lower Level Protocols

- We start with discussing the three lowest layers of the OSI protocol suite. Together, these layers implement the basic functions that encompass a computer network.

- The physical layer is concerned with transmitting the 0s and 1s.
- Key Issues in physical layer:
  - 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), as well as the number of pins and meaning of each are of concern here.
- The physical layer protocol deals with standardizing the electrical, mechanical, and signaling interfaces so that when one machine sends a 0 bit it is actually received as a 0 bit and not a 1 bit.
- Many physical layer standards have been developed (for different media), for example, the RS-232-C standard for serial communication lines.

- The physical layer just sends bits.
- As long as no errors occur, all is well.
- However, real communication networks are subject to errors, so some mechanism is needed to detect and correct them.
- This mechanism is the main task of the data link layer.
- What it does is to group the bits into units, sometimes called frames, and see that each frame is correctly received.
- The data link layer does its work by putting a special bit pattern on the start and end of each frame to mark them, as well as computing a checksum by adding up all the bytes in the frame in a certain way.

- The data link layer appends the checksum to the frame.
- When the frame arrives, the receiver recomputes the checksum from the data and compares the result to the checksum following the frame.
- If the two agree, the frame is considered correct and is accepted.
- If they disagree. the receiver asks the sender to retransmit it. Frames are assigned sequence numbers (in the header), so everyone can tell which is which.

- On a LAN, there is usually no need for the sender to locate the receiver.
- It just puts the message out on the network and the receiver takes it off.
- A wide-area network, however, consists of a large number of machines, each with some number of lines to other machines, rather like a large-scale map showing major cities and roads connecting them.
- For a message to get from the sender to the receiver it may have to make a number of hops, at each one choosing an outgoing line to use. The question of how to choose the best path is called routing, and is essentially the primary task of the network layer.

- The problem is complicated by the fact that the shortest route is not always the best route.
- What really matters is the amount of delay on a given route, which, in tum, is related to the amount of traffic and the number of messages queued up for transmission over the various lines.
- The delay can thus change over the course of time. Some routing algorithms try to adapt to changing loads, whereas others are content to make decisions based on long-term averages.

- At present, the most widely used network protocol is the connectionless IP (Internet Protocol), which is part of the Internet protocol suite.
- An IP packet (the technical term for a message in the network layer) can be sent without any setup.
- Each IP packet is routed to its destination independent of all others. No internal path is selected and remembered.

Transport Protocols

- The transport layer forms the last part of what could be called a basic network protocol stack, in the sense that it implements all those services that are not provided at the interface of the network layer, but which are reasonably needed to build network applications.
- In other words, the transport layer turns the underlying network into something that an application developer can use.
- Packets can be lost on the way from the sender to the receiver.
- Although some applications can handle their own error recovery, others prefer a reliable connection.
- The job of the transport layer is to provide this service.
- The idea is that the application layer should be able to deliver a message to the transport layer with the expectation that it will be delivered without loss.

- Upon receiving a message from the application layer, the transport layer breaks it into pieces small enough for transmission, assigns each one a sequence number, and then sends them all.
- The discussion in the transport layer header concerns which packets have been sent, which have been received, how many more the receiver has room to accept, which should be retransmitted, and similar topics.
- Reliable transport connections (which by definition are connection oriented) can be built on top of connection-oriented or connectionless network services.
- In the former case all the packets will arrive in the correct sequence (if they arrive at all), but in the latter case it is possible for one packet to take a different route and arrive earlier than the packet sent before it.
- It is up to the transport layer software to put everything back in order to maintain the illusion that a transport connection is like a big tube-you put messages into it and they come out undamaged and in the same order in which they went in. Providing this end-to-end communication behavior is an important aspect of the transport layer.

- The combination TCP/IP is now used as a de facto standard for network communication.
- The Internet protocol suite also supports a connectionless transport protocol called UDP (Universal Datagram Protocol), which is essentially just IP with some minor additions.
- User programs that do not need a connection-oriented protocol normally use UDP.
- Additional transport protocols are regularly proposed.
- For example, to support real-time data transfer, the Real-time Transport Protocol (RTP) has been defined.
- RTP is a framework protocol in the sense that it specifies packet formats for real-time data without providing the actual mechanisms for guaranteeing data delivery.

Higher- Level Protocols

- Above the transport layer, OSI distinguished three additional layers.
- In practice, only the application layer is ever used.
- In fact, in the Internet protocol suite, everything above the transport layer is grouped together.
- In the face of middleware systems, we shall see in this section that neither the OSI nor the Internet approach is really appropriate.

- The session layer is essentially an enhanced version of the transport 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 necessary to go back only to the last checkpoint, rather than all the way back to the beginning.
- In practice, few applications are interested in the session layer and it is rarely supported.
- It is not even present in the Internet protocol suite.
- However, in the context of developing middleware solutions, the concept of a session and its related protocols has turned out to be quite relevant, notably when defining higher-level communication protocols.

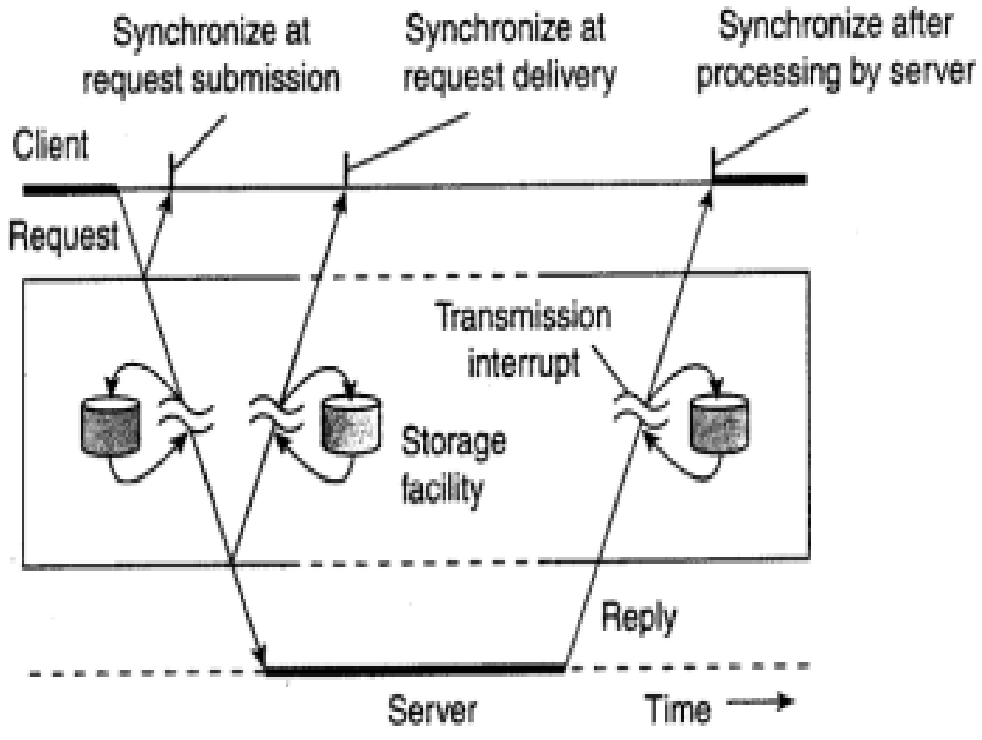
- Unlike the lower layers, which are concerned with getting the bits from the sender to the receiver reliably and efficiently, the presentation layer is concerned with the meaning of the bits.
- Most messages do not consist of random bit strings, but more structured information such as people's names, addresses, amounts of money, and so on.
- 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 with each other.

Middleware Protocols

- Middleware is an application that logically lives (mostly) in the application layer, but which contains many general-purpose protocols that warrant their own layers, independent of other, more specific applications.
- A distinction can be made between high-level communication protocols and protocols for establishing various middleware services.
- Authentication protocols
- Authorization protocols

Types of Communication

- To understand the various alternatives in communication that middleware can offer to applications, we view the middleware as an additional service in client-server computing, as shown in Fig.



Example: electronic mail system

- In principle, the core of the mail delivery system can be seen as a middleware communication service.
- Each host runs a user agent allowing users to compose, send, and receive e-mail.
- A sending user agent passes such mail to the mail delivery system, expecting it, in tum, to eventually deliver the mail to the intended recipient.
- Likewise, the user agent at the receiver's side connects to the mail delivery system to see whether any mail has come in.
- If so, the messages are transferred to the user agent so that they can be displayed and read by the user.

Persistence and Synchronicity

- Persistent vs Transient Communication
  - Persistent: submitted messages are stored by the comm. system as long as it takes to deliver it the receiver
  - Transient: messages are stored by the comm system as long as the sending and receiving app. are executing.
- Asynchronous vs synchronous
  - Sync: a client is blocked until its message is stored in a local buffer at the receiving host, or actually delivered to the receiver
  - Async: A sender continues immediately after it has submitted its message for transmission. (The message is stored in a local buffer at the sending host, or otherwise at the first communication server.

Persistent Communication

- An electronic mail system is a typical example in which communication is persistent.
- With persistent communication, a message that has been submitted for transmission is stored by the communication middleware as long as it takes to deliver it to the receiver.
- In this case, the middleware will store the message at one or several of the storage facilities shown in Fig in previous slide.
- As a consequence, it is not necessary for the sending application to continue execution after submitting the message.
- Likewise, the receiving application need not be executing when the message is submitted.

Transient Communication

- In contrast, with transient communication, a message is stored by the communication system only as long as the sending and receiving application are executing.
- More precisely, in terms of Fig in previous slide, the middleware cannot deliver a message due to a transmission interrupt, or because the recipient is currently not active, it will simply be discarded.
- Typically, all transport-level communication services offer only transient communication.
- In this case, the communication system consists traditional store-and-forward routers.
- If a router cannot deliver a message to the next one or the destination host, it will simply drop the message

Asynchronous or Synchronous Communication

- The characteristic feature of asynchronous communication is that a sender continues immediately after it has submitted its message for transmission.
- This means that the message is (temporarily) stored immediately by the middleware upon submission.

- With synchronous communication, the sender is blocked until its request is known to be accepted.
- There are essentially three points where synchronization can take place.
  - First, the sender may be blocked until the middleware notifies that it will take over transmission of the request.
  - Second, the sender may synchronize until its request has been delivered to the intended recipient.
  - Third, synchronization may take place by letting the sender wait until its request has been fully processed, that is, up the time that the recipient returns a response.

Assignment-IV (Deadline: Feb 20)

- Explain OSI Refence Model for data communication in details.
- Explain the different types of communication based on persistence and synchronicity.

REMOTE PROCEDURE CALL

- Background

Many distributed systems have been based on explicit message exchange between processes.

However, the procedures send and receive do not conceal communication at all, which is important to achieve access transparency in distributed system.

RPC: Definition

- Concept: 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, and 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 at all is visible to the programmer.
- This method is known as Remote Procedure Call, or often just RPC.

- While the basic idea sounds simple and elegant, subtle problems exist.
- To start with, because the calling and called procedures run on different machines, they execute in different address spaces, which causes complications.
- Parameters and results also have to be passed, which can be complicated, especially if the machines are not identical.
- Finally, either or both machines can crash and each of the possible failures causes different problems.
- Still, most of these can be dealt with, and RPC is a widely-used technique that underlies many distributed systems.

Basic RPC Operation

- We first start with discussing conventional procedure calls, and then explain how the call itself can be split into a client and server part that are each executed on different machines.

Conventional Procedure Call

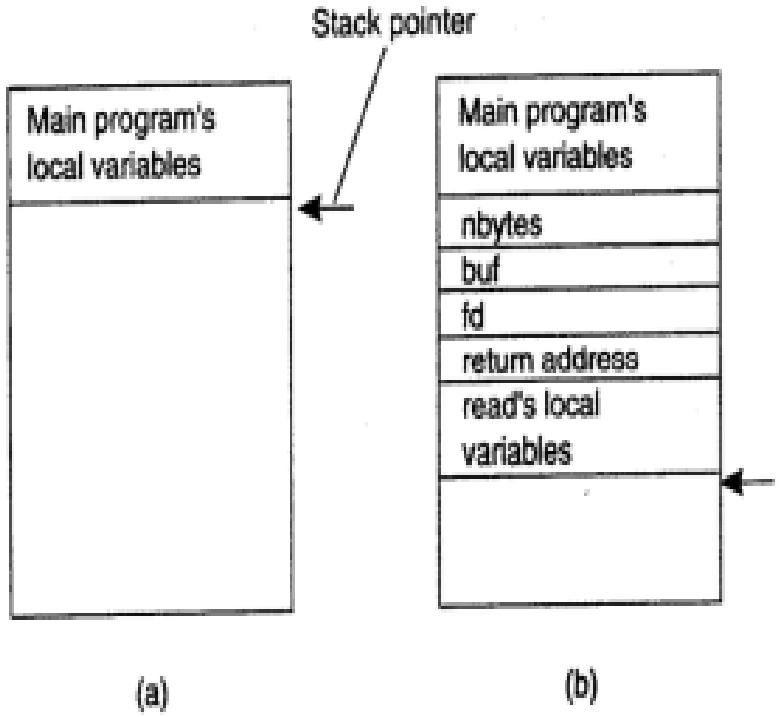
- Conventional (i.e., single machine) Procedure Call
- Consider a call in C like:

Count = tead(fd, but, nbytes)

Where,

fd →integer indicating a file,
buf →an array of characters into which data are read, and
nbytes → another integer telling how many bytes to read

- If the call is made from the main program, the stack will be as shown in Fig. (a) before the call.
- To make the call, the caller pushes the parameters onto the stack in order, last one first, as shown in Fig. (b).
- After the read procedure has finished running, it puts the return value in a register, removes the return address, and transfers control back to the caller.
- The caller then removes the parameters from the stack, returning the stack to the original state it had before the call.



(a) Parameter passing in a local procedure call: the stack before the call to read. (b) The stack while the called procedure is active.

- parameters can be call-by value or call-by-reference.

call-by value

- A value parameter, such as fd or nbytes, is simply copied to the stack as shown in Fig.(b).
- To the called procedure, a value parameter is just an initialized local variable.
- The called procedure may modify it, but such changes do not affect the original value at the calling side.

Call-by-reference

- A reference parameter in C is a pointer to a variable (i.e., the address of the variable), rather than the value of the variable.
- In the call to read. the second parameter is a reference parameter because arrays are always passed by reference in C.
- What is actually pushed onto the stack is the address of the character array.
- If the called procedure uses this parameter to store something into the character
- array, it does modify the array in the calling procedure.

pass by reference

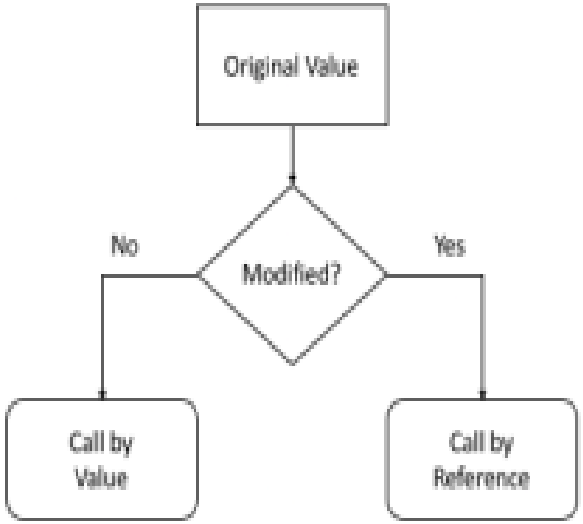
cup =

fillCup( )

pass by value

cup =

fillCup( )



PARAMETERS	CALL BY VALUE	CALL BY REFERENCE
Basic	A copy of the variable is passed.	A variable itself is passed.
effect	Change in a copy of variable doesn't modify the original value of variable.	Change in a copy of variable modify the original value of variable.
Syntax	function_name(variable_name1, variable_name2...)	function_name(&variable_name1, &variable_name2...)
Default calling	Primitive type are passed using "call_by_value".	Objects are implicitly passed using "call_by_reference".

Elements of RPC: Client Server Model

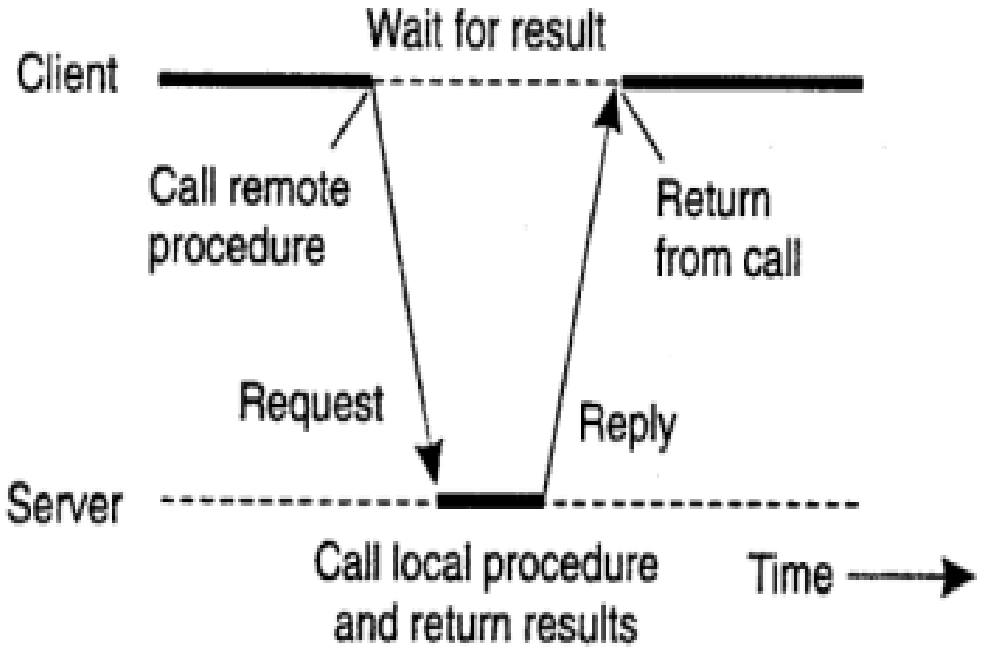
- RPC Uses client server model.

Mainly five components:

- The Client
- The Client Stub

  Stub: Piece of code used for converting parameter
- The RPC Routine

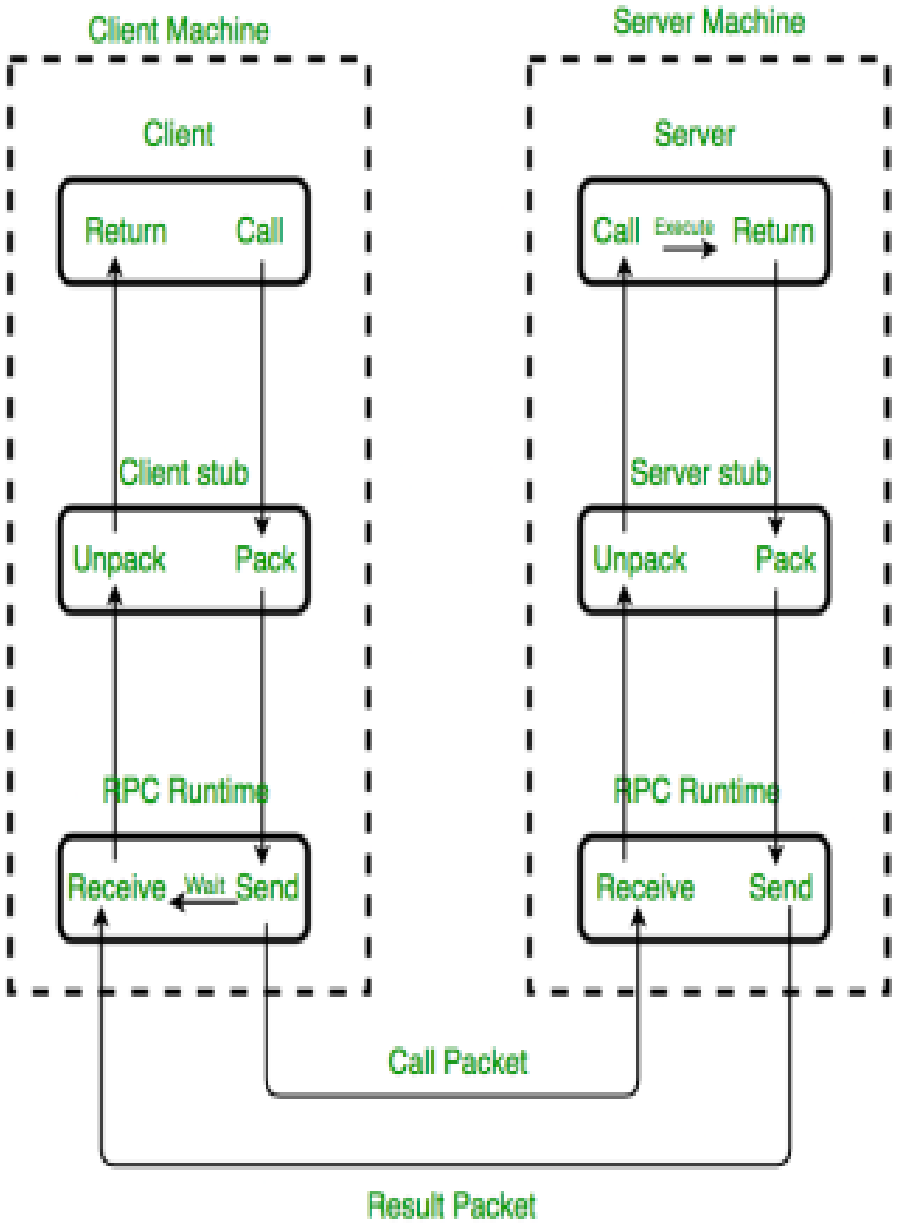
  (Communication Package)
- The Server Stubs
- The Server



Principle of RPC between a client and server program.

- The Client:
  - It is user process which initiates a RPC.
  - The Client makes a perfectly normal call that involves a corresponding procedure in the client stub.
- The Client Stub:
  - On receipt of a request it packs a requirement into a message and asks to RPC runtime to send.
  - On receipt of a result it unpacks the result and passes it to client.

- RPC Runtime:
  - It handles transmission of message between client and Server.
- The Server Stub:
  - It unpacks a call request and make a perfectly normal call to invoke the appropriate procedure in the Server.
- Server:
  - It executes an appropriate procedure and returns the result from a server stub.



Implementation of RPC mechanism

RPC steps

1. The client procedure calls the client stub in the normal way.

2. The client stub builds a message and calls the local operating system.

3. The client's OS sends the message to the remote as.

4. The remote OS gives the message to the server stub.

5. The server stub unpacks the parameters and calls the server.

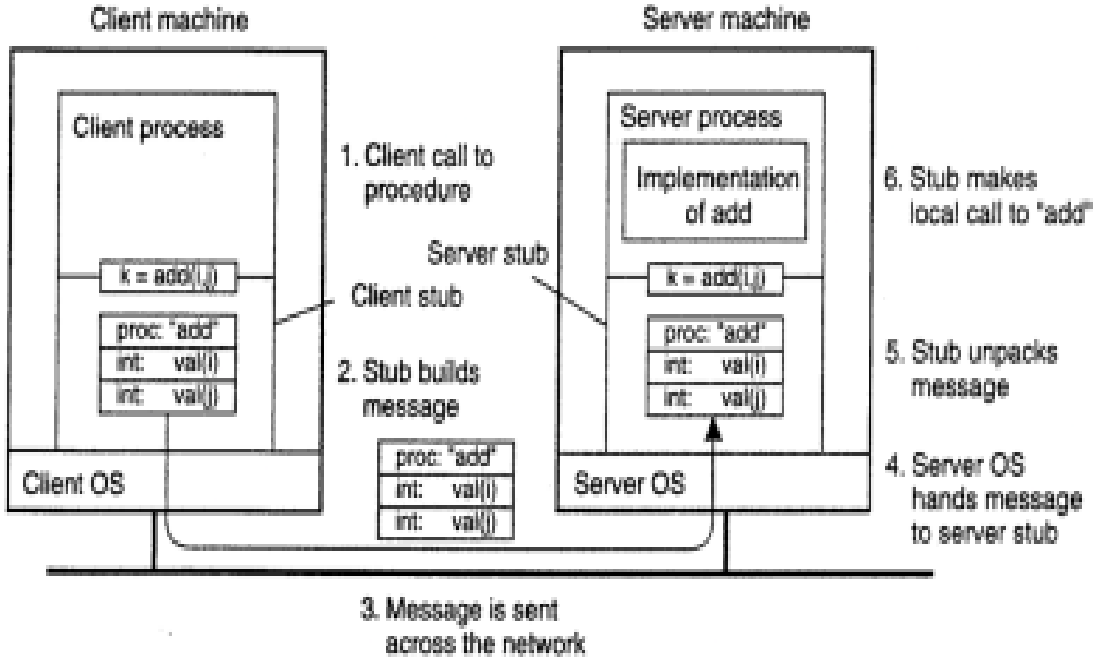
6. The server does the work and returns the result to the stub.

7. The server stub packs it in a message and calls its local OS.

8. The server's OS sends the message to the client's as.

9. The client's OS gives the message to the client stub.

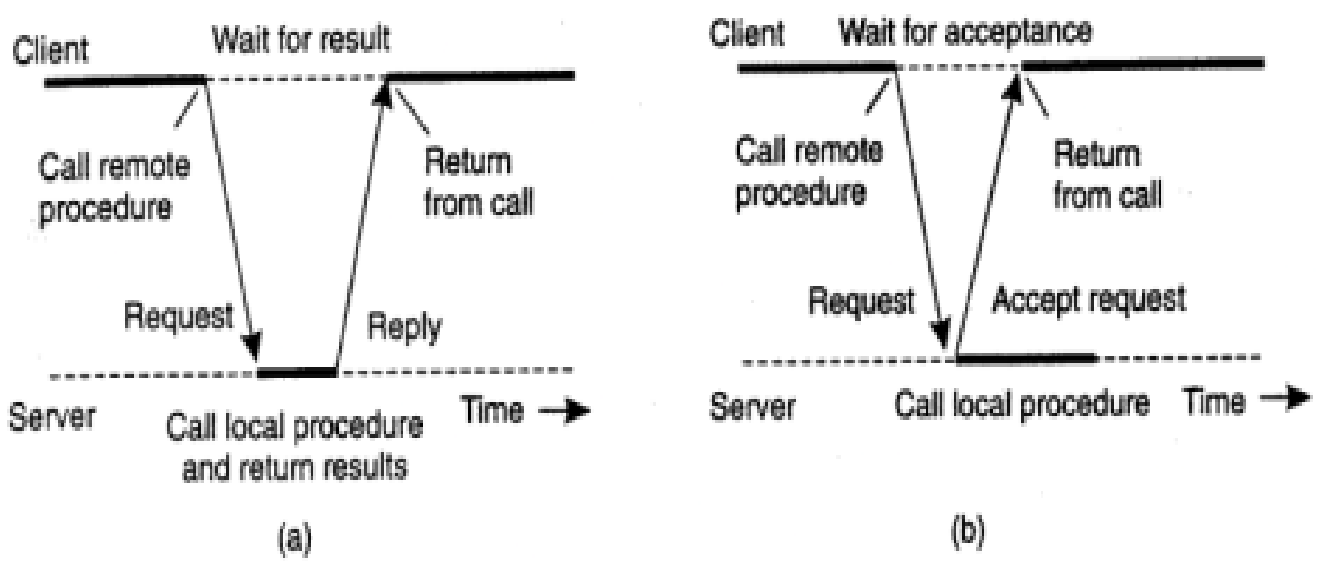
10. The stub unpacks the result and returns to the client.



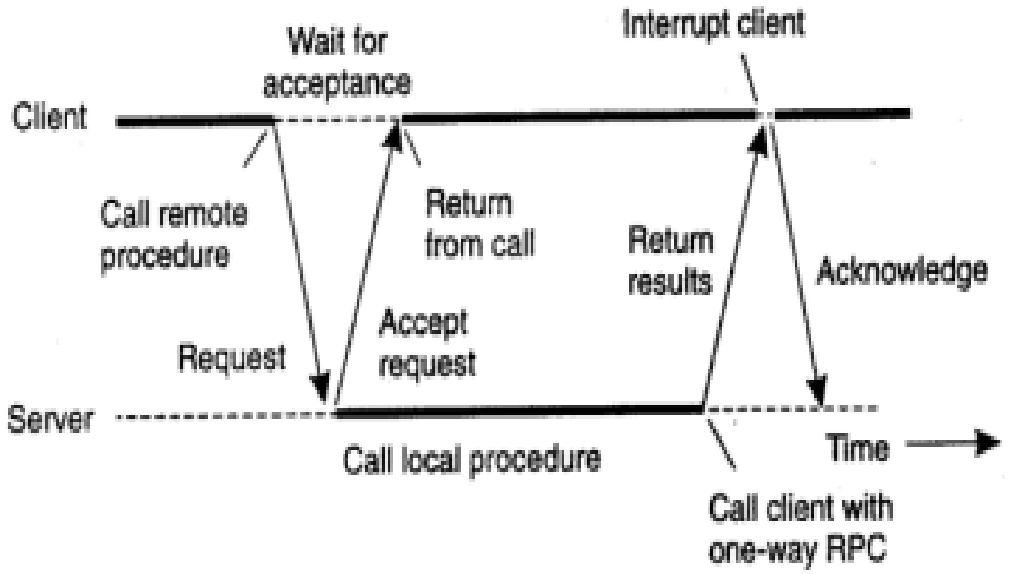
Asynchronous RPC

- As in conventional procedure calls, when a client calls a remote procedure, the client will block until a reply is returned.
- This strict request-reply behavior is unnecessary when there is no result to return, and only leads to blocking the client while it could have proceeded and have done useful work just after requesting the remote procedure to be called.
- Examples of where there is often no need to wait for a reply include:
  - transferring money from one account to another,
  - adding entries into a database,
  - starting remote services,
  - batch processing, and
  - so on.

- To support such situations, RPC systems may provide facilities for what are called asynchronous RPCs, by which a client immediately continues after issuing the RPC request.
- With asynchronous RPCs, the server immediately sends a reply back to the client the moment the RPC request is received, after which it calls the requested procedure.
- The reply acts as an acknowledgment to the client that the server is going to process the RPC.
- The client will continue without further blocking as soon as it has received the server's acknowledgment.



(a) The interaction between client and server in a traditional RPC.
(b) The interaction using asynchronous RPC.



A client and server interacting through two asynchronous RPCs.

MESSAGE-ORIENTED COMMUNICATION

- Remote procedure calls and remote object invocations contribute to hiding communication in distributed systems, that is, they enhance access transparency.
- Unfortunately, neither mechanism is always appropriate.
- In particular, when it cannot be assumed that the receiving side is executing at the time a request is issued, alternative communication services are needed.
- Likewise, the inherent synchronous nature of RPCs, by which a client is blocked until its request has been processed, sometimes needs to be replaced by something else.

- That something else is messaging.
- What we discuss here:
  - what exactly synchronous behavior is and what its implications are
  - messaging systems that assume that parties are executing at the time of communication.
  - examine message-queuing systems that allow processes to exchange information, even if the other party is not executing at the time communication is initiated.

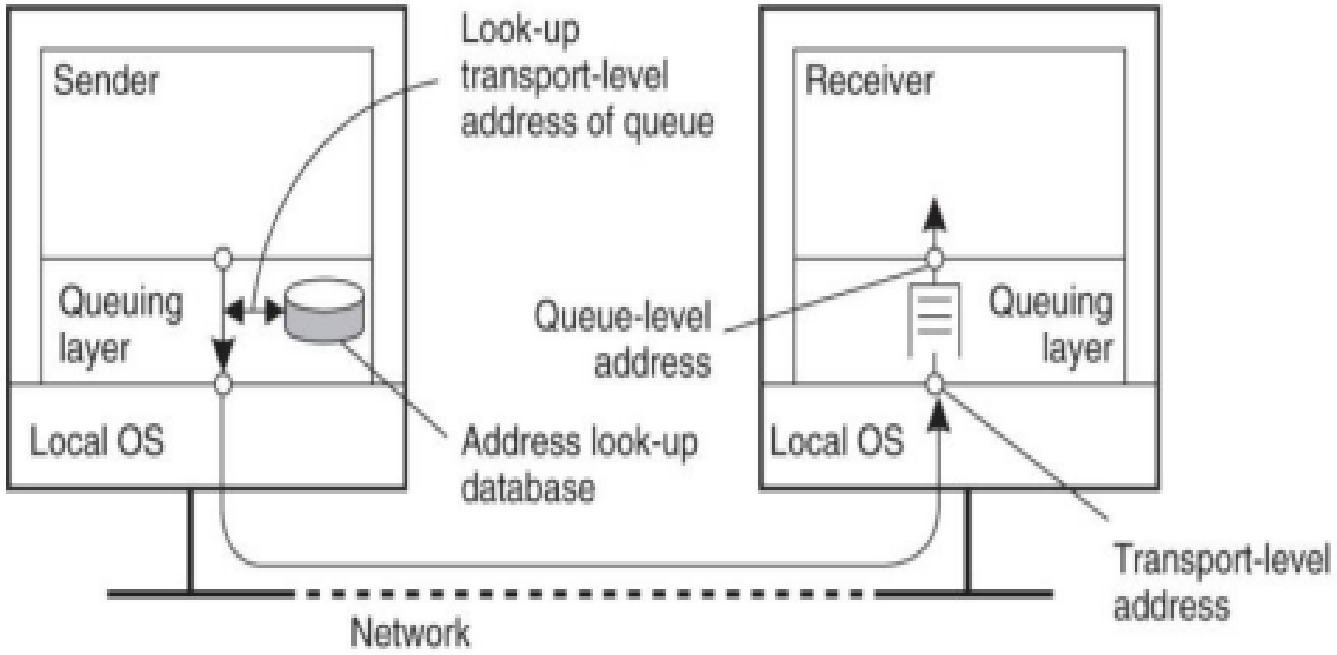
- Message-Oriented Transient Communication
  - Berkeley Socket Primitives, and
  - Message-Passing Interface (MPI)
- Message-Oriented Persistent Communication
  - Message Queuing Systems (MQS)

Message-Oriented Persistent Communication

- Message-queuing systems – a.k.a. Message-Oriented Middleware (MOM)
- Basic idea: MOM provides message storage service.
- A message is put in a queue by the sender, and delivered to a destination queue .
- The target(s) can retrieve their messages from the queue.
- Time uncoupling between sender and receiver
- Example: IBM's WebSphere

Time uncoupling: a sender can send a message even if the receiver is still not available. The message is stored and picked up at a later moment.

Time coupled interactions are observable when communication cannot take place unless both endpoints are operating at the same time.



General architecture of a message-queuing system

Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block
Notify	Install a handler to be called when a message is put into the specified queue

Figure 4-18. Basic interface to a queue in a message-queuing system.

Message-Oriented Transient Communication

- Messages are sent through a channel abstraction.
- The channel connects two running processes.
- Time coupling between sender and receiver.
- Transmission time is measured in terms of milliseconds, typically
- Examples:
  - Berkeley Sockets — typical in TCP/IP-based networks
  - MPI (Message-Passing Interface) — typical in high-speed interconnection networks among parallel processes

Assignment-V (Deadline : feb 23)

- Explain the merits and demerits of RPC and also explain how Message Oriented communication overcome the demerits of RPC.

Berkeley Sockets

- Special attention has been paid to standardizing the interface of the transport layer to allow programmers to make use of its entire suite of (messaging) protocols through a simple set of primitives. Also, standard interfaces make it easier to port an application to a different machine.
- Conceptually, a socket is a communication end point to which an application can write data that are to be sent out over the underlying network, and from which incoming data can be read. A socket forms an abstraction over the actual communication end point that is used by the local operating system for a specific transport protocol.

- Servers generally execute the first four primitives, normally in the order given.
- When calling the socket primitive, the caller creates a new communication end point for a specific transport protocol.
- Internally, creating a communication end point means that the local operating system reserves resources to accommodate sending and receiving messages for the specified protocol.

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a 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

Figure 4-14. The socket primitives for TCPIIP.

a server should bind the IP address of its machine together with a (possibly well-known) port number to a socket. Binding tells the operating system that the server wants to receive messages only on the specified address and port.

- Let us now take a look at the client side. Here, too, a socket must first be created using the socket primitive, but explicitly binding the socket to a local address is not necessary, since the operating system can dynamically allocate a port when the connection is set up.
- The connect primitive requires that the caller specifies the transport-level address to which a connection request is to be sent.
- The client is blocked until a connection has been set up successfully, after which both sides can start exchanging information through the send and receive primitives.
- Finally, closing a connection is symmetric when using sockets, and is established by having both the client and server call the close primitive.

general pattern followed by a client and server for connection-oriented communication using sockets

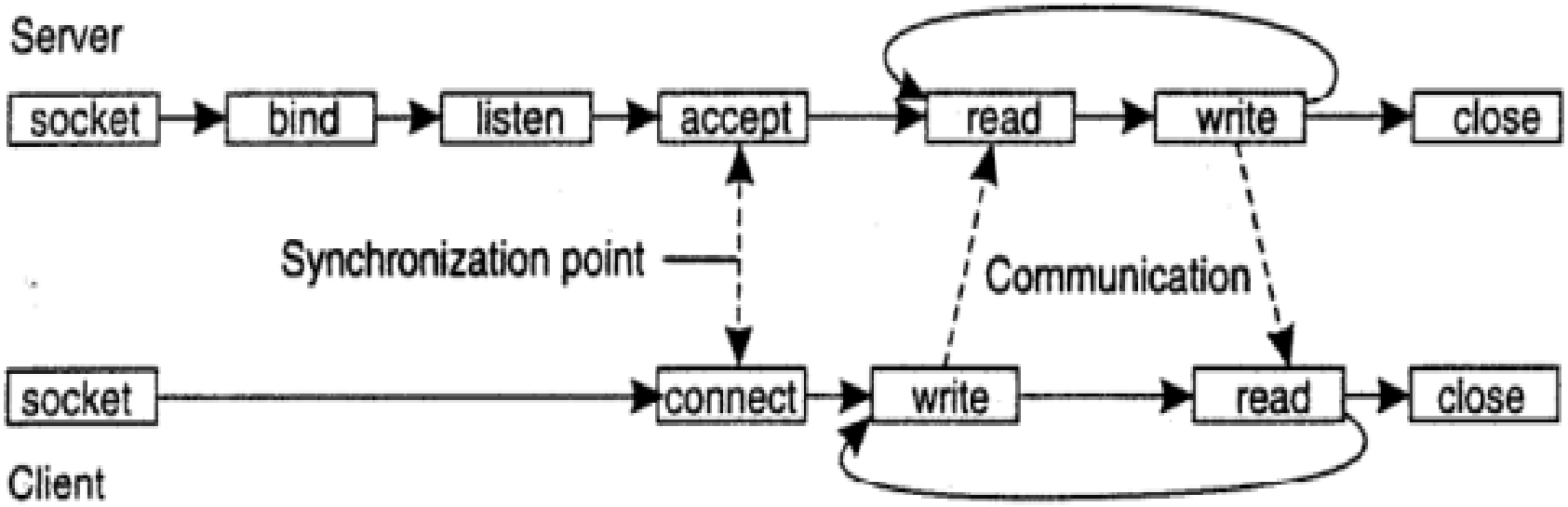


Figure 4-15. Connection-oriented communication pattern using sockets.

The Message-Passing Interface (MPI)

- Group of message-oriented primitives that would allow developers to easily write highly efficient applications.
- Sockets insufficient because:
  - – at the wrong level of abstraction supporting only send and receive primitives,
  - – designed to communicate using general-purpose protocol stacks such as TCP/IP, not suitable in high-speed interconnection networks, such as those used in COWs and MPPs (with different forms of buffering and synchronization).

MPI assumptions:

- – communication within a known group of processes,
- – each group with assigned id,
- – each process withing a group also with assigned id,
- – all serious failures (process crashes, network partitions) assumed as fatal and without any recovery,
- – a (groupID, processID) pair used to identify source and destination of the message,
- – only receipt-based transient synchronous communication (d) not supported, other supported.

The Message-Passing Interface (MPI)

Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there is none
MPI_irecv	Check if there is an incoming message, but do not block

Figure 4-16. Some of the most intuitive message-passing primitives of MPI.

STREAM-ORIENTED COMMUNICATION

Multicast Communication

- **Rationale**: Often need to a Send-to-Many in Distributed Systems

Examples:

- Financial services: Delivery of news, stock quotes etc.
- E-learning: Streaming content to many students at different levels.

Introduction

- forms of communication in which timing plays a crucial role,
- example:
  - an audio stream built up as a sequence of 16-bit samples each representing the amplitude of the sound wave as it is done through PCM (Pulse Code Modulation),
  - audio stream represents CD quality, i.e. 44100Hz,
  - samples to be played at intervals of exactly 1/44100,

- which facilities a distributed system should offer to exchange time-dependent information such as audio and video streams?
  - support for the exchange of time-dependent information = **support for continuous media,**
  - **continuous** (representation) media vs. **discrete** (representation) media.

Support for Continuous Media

- In continuous media :
  - temporal relationships between data items fundamental to correctly interpreting the data,
  - timing is crucial.

- **Asynchronous transmission mode**

Data items in a stream are transmitted one after the other, but there are no further timing constraints on when transmission of items should take place.

- **Synchronous transmission mode**

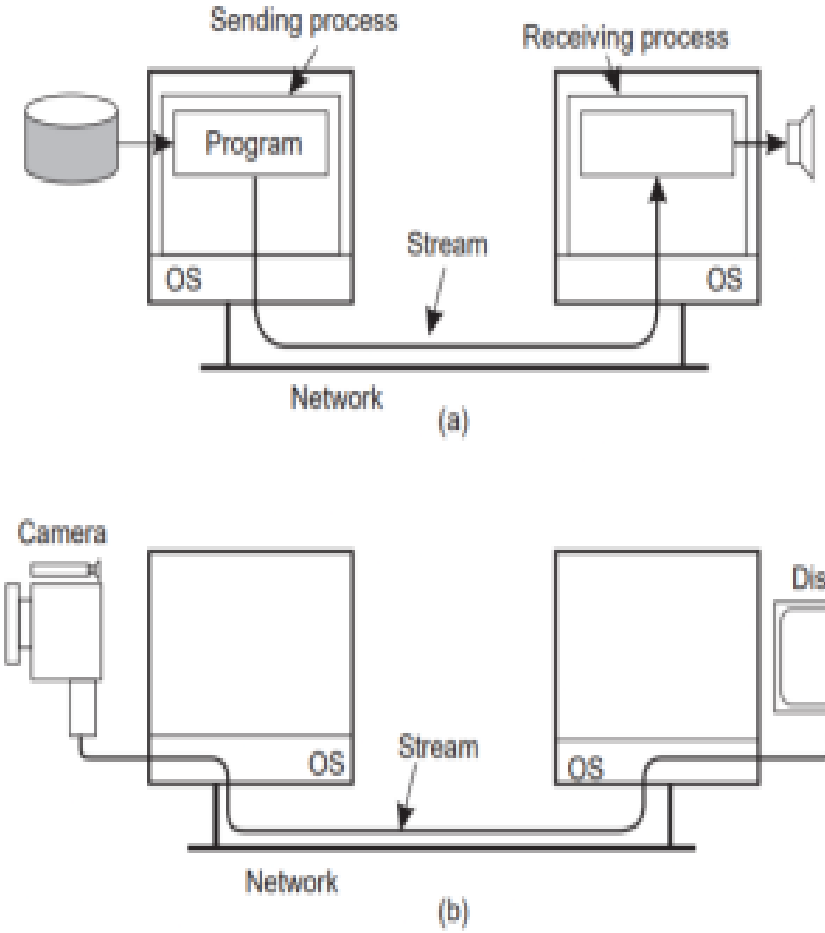
Maximum end-to-end delay defined for each unit in a data stream.

- **Isochronous transmission mode**

It is necessary that data units are transferred on time. Data transfer is subject to bounded (delay) jitter.

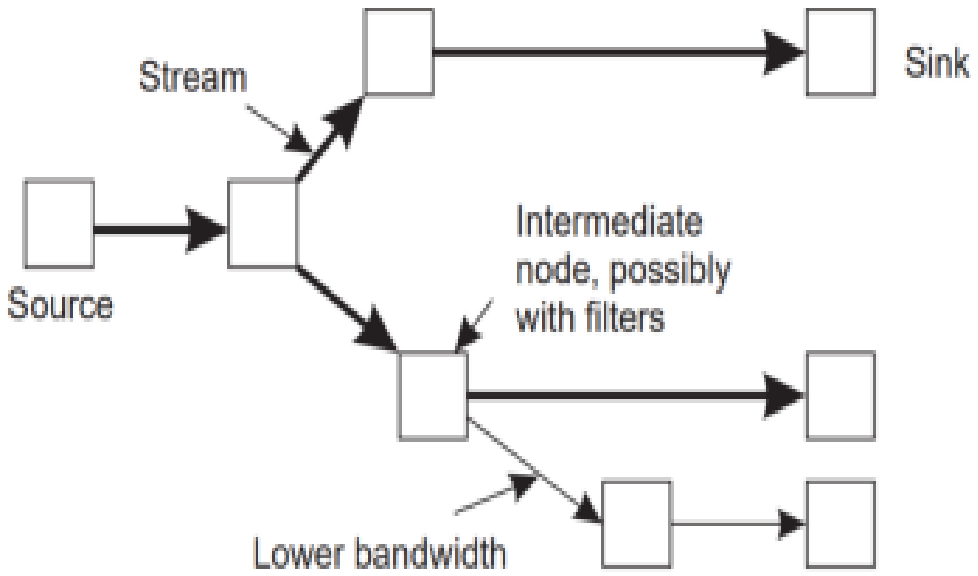
Data Stream

a. Setting up a stream between two processes across a network,
b. Setting up a stream directly between two devices.
   – stream sequence of data units, may be considered as a virtual connection between a source and a sink,
   – simple stream vs. complex stream (consisting of several related sub-streams).



Data Stream

- An example of multicasting a stream to several receivers.
  - – problem with receivers having different requirements with respect to the quality of the stream,
  - – filters to adjust the quality of an incoming stream, differently for outgoing streams.

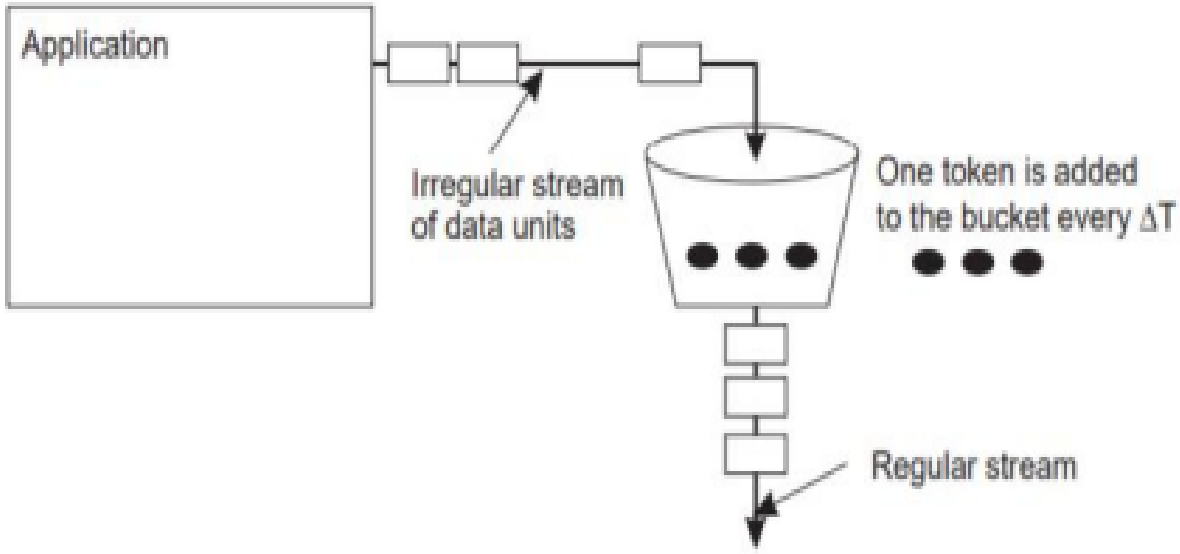


Specifying QoS: Token Bucket Algorithm

The principle of a token bucket algorithm.

– tokens generated at a constant rate,

– tokens buffered in a bucket which has limited capacity.



Summary

- Middleware enables much functionality in DS
- Especially the many types of interaction/communications necessary
- With rational reasons for every one!
  - Remote Procedure Call (RPC) enables transparency
  - But Message Queuing Systems necessary for persistent communications
    - IBM WebSphere is ok but a bit old, clunky & tired at this stage
    - AMQP open source, more flexible, better Industrial support
  - Multicast Communications are often necessary in DS