Distributed Systems
MTech
Delhi Technological University
Communication
Instructor: Divyashikha Sethia
divyashikha@dce.edu

Objective

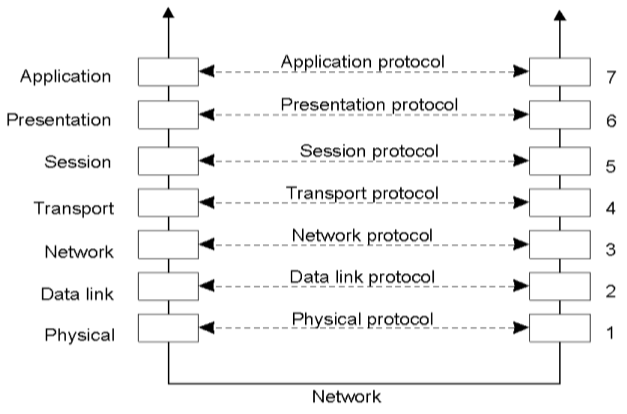
Interprocess communication is at the heart of all distributed systems

• Protocols

•Three models of communication: Remote Procedure Call (RPC), Message-Oriented Middleware (MOM), and data streaming

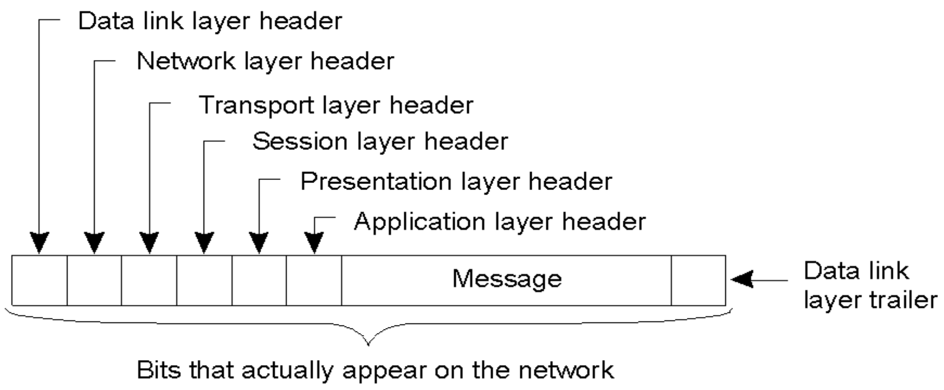
•Multicasting (Application Level Multicast)

Layered Protocols (1)



Layers, interfaces, and protocols in the OSI model.

Layered Protocols (2)



A typical message as it appears on the network.

OSI Layers

•**Physical layer:**
-deals with standardizing the electrical, mechanical, and signaling interfaces
•**Data link Layer:**
-Group bits into units, called frames, and ensure that each frame is correctly received
-Error correction and detection (checksum, sequencing)
-Flow control
•**Network Layer:**
-End point logical address
-Routing
-Congestion control
•**Transport Layer:**
-End to End communication
-Reliable connection
-Flow control

OSI Layers

•**Session Layer:**
-Dialog Control
-Synchronization facility (checkpoints into long transfers to avoid going to beginning in case of crash)
•**Presentation Layer:**
-Representation of data
•**Application Layer:**
- Network applications like electronic transfer, file transfer terminal emulation

Middleware Protocols

Application that logically belongs to application layer, but contains many general-purpose protocols with their own layers, independent of other, for more specific applications

Eg:

-**Authentication protocols** :

service to provide various ways to establish authentication or proof of identity

-**Authorization protocols**:

authenticated users and processes are granted access only to those resources for which they have authorization.

-**Commit protocols**:

in group of processes either all processes carry out a particular operation, or that the operation is not carried out at all - referred to as atomicity and is widely applied in transactions.

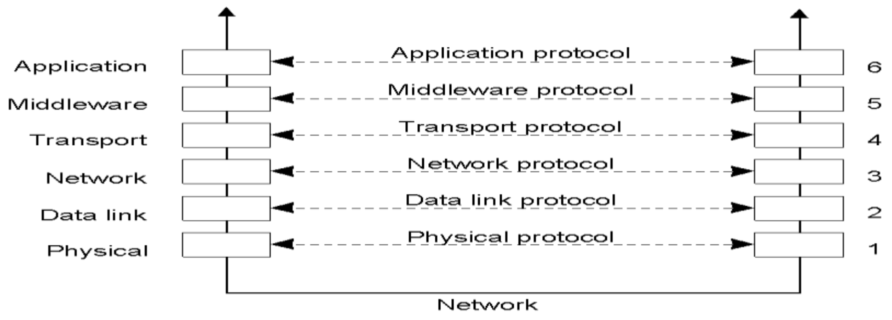
-

Middleware Protocols

-**Distributed locking protocol**:

a resource can be protected against simultaneous access by a collection of processes that are distributed across multiple machines

Middleware Protocols



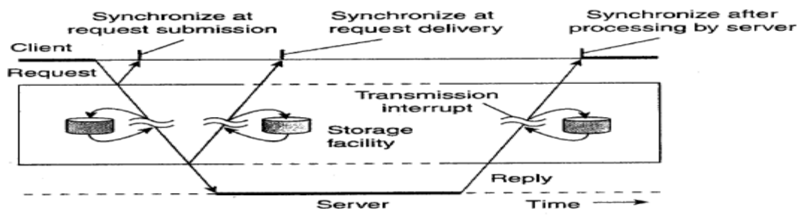
An adapted reference model for networked communication.

- Middleware communication protocols support high-level communication services.
- Session and presentation layer have been replaced by a single middleware layer that contains application-independent protocols

Middleware services

•remote procedure calls
•message queuing services
•support for communication of continuous media through streams
•multicasting.

middleware as an intermediate (distributed) service


Synchronize at request submission
Synchronize at request delivery
Synchronize after processing by server
Client
Request
Transmission interrupt
Storage facility
Reply
Server
Time

- Mail delivery system can be seen as a middleware communication service.

•Host runs user agent allowing users to compose, send, and receive e-mail.
•Sending user agent passes mail to mail delivery system which will deliver the mail to recipient.
•User agent at receiver's side connects to 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.

Remote Procedure Call

•Programs call procedures located on other machines

•Process on machine A calls procedure on machine B

•Calling process on A is suspended, and execution of called procedure takes place on B.

•Information is transported from caller (A) to callee (B) in the parameters and is returned back in the procedure result.

•Message passing is not visible to programmer.

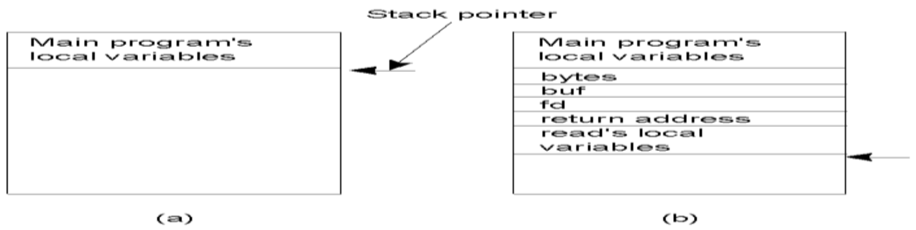
Issues

•Calling and called procedures run on different machines, they execute in different address spaces

•Passing Parameters and results to heterogeneous machines

•Situation in which either machine crashes

Conventional Procedure Call

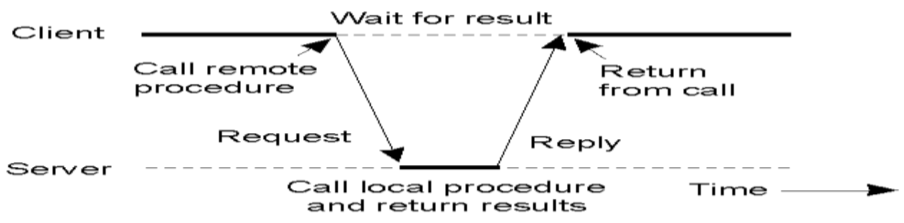


Count = read (fd, buf, bytes)

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

- caller pushes parameters onto stack in order, last one first so that first parameter is accessible first
- Procedure after computation puts return value in a register, removes return address, and transfers control back to caller.
- caller then removes parameters from stack, returning stack to original state

Client and Server Stubs



•Read does a system call by pushing the parameters onto the stack
•For a read implemented as RPC (e.g., one that will run on the file server's machine), a different version of read, called a client stub, is put into the library.
•It does a call to the local operating system.
•Instead of OS giving data, it packs the parameters into a message and requests that message to be sent to the server.
•After call to send, client stub calls receive, blocking until reply comes back.

RPC –server side

•Server's OS passes it up to server stub (equivalent of a client stub) which transforms requests coming in over network into local procedure calls.

•Server stub calls receive and remains blocked waiting for incoming messages.

•Server stub unpacks parameters from message and then calls server procedure usual way using the stack as if it is being called directly by client, the parameters and return address are all on stack.
•After work completion server returns result to caller
  - Eg case of read, server will fill buffer, pointed to by the second parameter, with the data. This buffer will be internal to the server stub.
  - When server stub gets control back after the call has completed, it packs result (the buffer) in a message and calls send to return it to the client.
•Server stub does a call to receive again, to wait for next incoming request.

RPC client receiving results

•Client's OS sees that it is addressed to client process (client stub)

•Message is copied to waiting buffer and client process unblocked.

•Client stub inspects message, unpacks result, copies it to its caller, and returns the usual way.

•Caller gets control following the call to read, and gets data available unaware of fact that the work was done remotely

Steps of a Remote Procedure Call

1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

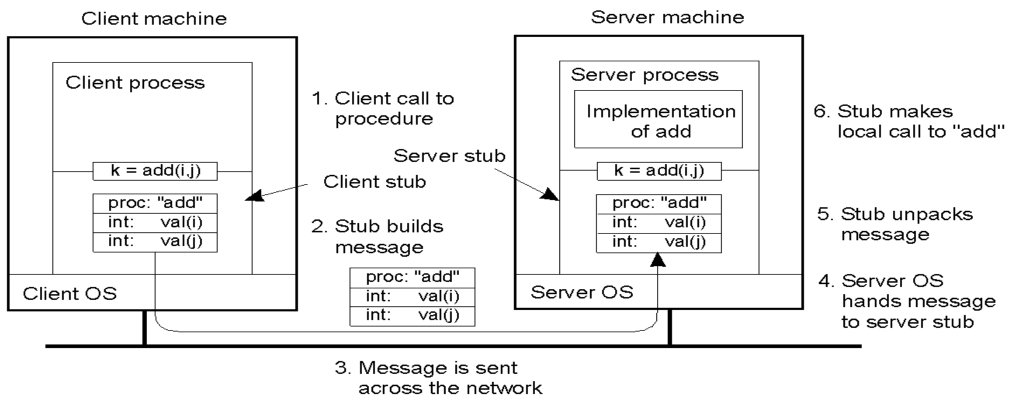
Parameter Passing

•Passing Value Parameters

•Passing Reference Parameters

•Parameter Specification and Stub Generation

Passing Value Parameters (1)



Steps involved in doing remote computation through RPC

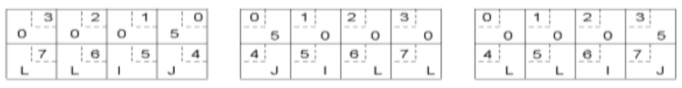
Packing parameters into a message is called **parameter marshaling**.

Passing Value Parameters(2)
Issues

•Different representation for numbers, characters, and other data items. Eg: EBCDIC character code, whereas IBM personal computers use ASCII

•Incorrect interpretation of characters
Format of code: Intel Pentium, number their bytes from right to left, whereas others, such as the Sun SPARC, number them the other way.

Passing Value Parameters (3)



(a) (b) (c)

Procedure with two parameters, an integer and a four-character String (5, JILL)

Each parameter requires one 32-bit word (each box is a byte)

a) Original message on the Pentium

Message transferred byte for byte (first byte sent is first received)

Intel Pentium number their bytes from right to left – little endian

b) The message after receipt on the SPARC

Sparc numbers bytes from left to right – big endian

c) The message after being inverted. The little numbers in boxes indicate the address of each byte 5 interpreted as 83,886,080 (5 x

Passing References (1)

•Pointer is meaningful only within address space of process in which it is being used eg: Address 1000 on server might be in middle of program text

•Solution – avoid pointers call-by-reference replaced by copy/restore.
- If client stub knows parameter point to array of characters and length of array - copy array into message and send it to server
- server stub then calls the server with a pointer to this array
- Changes made by server using the pointer (e.g., storing data into it) directly affects message buffer inside server stub
- When finished original message can be sent back to client stub, which copies it back to client
- Optimization: if buffer an input parameter (eg write) then copying can be eliminated since it does not have to be sent back.

Passing References (2)

•pointer to an arbitrary data structure such as a complex graph:

- pass pointer to server stub and special code in server to handle such pointers eg: send back request to client to provide referenced data

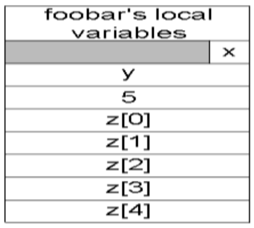
Parameter Specification and Stub Generation

1. Both sides to follow same format of messages they exchange
Eg: procedure foobar with three param.
word is four bytes

client and the server to agree on representation of simple data structures, such as integers, characters, Booleans, etc.
Eg: integers are represented in two's complement

```
foobar( char x; float y; int z[5] )
{
   ....
}
```

(a)



(b)

a) A procedure b)The corresponding message.

Parameter Specification and Stub Generation

2. Caller and callee agree on the actual exchange of messages using:
- connection oriented data like TCP
- unreliable datagram service and let client and server implement an error control scheme as part of RPC

3. Client and server stubs implemented :
- Has an interface by means of Interface Definition Language (IDL) which is compiled into client and server stub.
- An interface consists of collection of procedures that can be called by client, and which are implemented by server

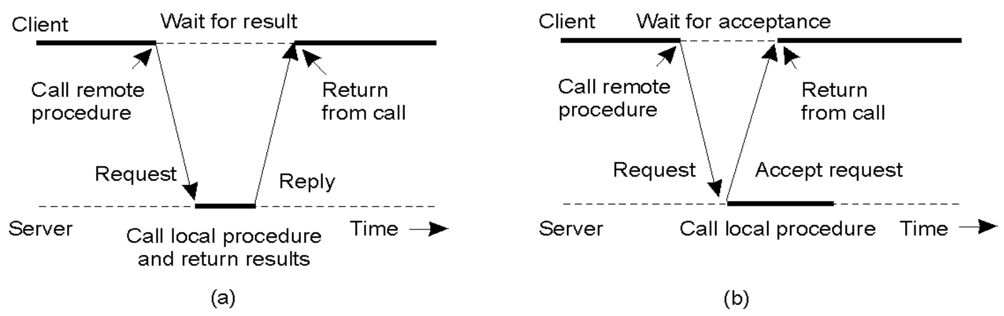
Asynchronous RPC

•Conventional procedure calls client blocked until reply is returned

•Asynchronous RPC: client is unblocked after initiating an RPC and does not wait for reply to return
Eg: transferring money from one account to another, adding entries into a database, starting remote services, batch processing

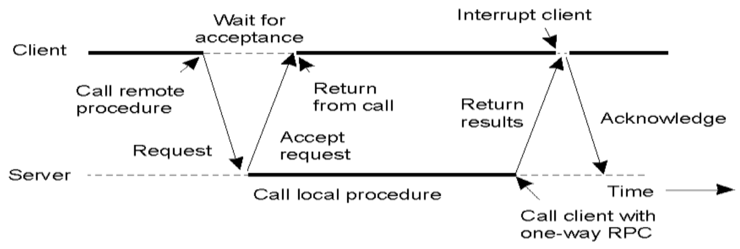
•On receiving request, server immediately sends an ACK to client for proceeding with RPC request

Asynchronous RPC (1)



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

Asynchronous RPC (2)



A client and server interacting through two asynchronous RPCs

- **Deferred synchronous RPC**: combining two asynchronous RPCs

eg: client wants to prefetch network addresses of a set of hosts. While remote server naming service collects addresses, client may want to do other things.
RPC1: client first calls server to hand over list of host names that should be looked up, and continues when the server has acknowledged receipt of that list.
RPC2: server, calls client to hand over addresses found.

Asynchronous RPC (Variations)

Deferred synchronous RPC (with variation):
• Client may poll server to see whether results are available yet instead of letting server calling back client

•**One-way RPC**:
• Variant of Asynchronous RPC
- Client does not wait for ack of server's acceptance of request
- Reliability is not guaranteed

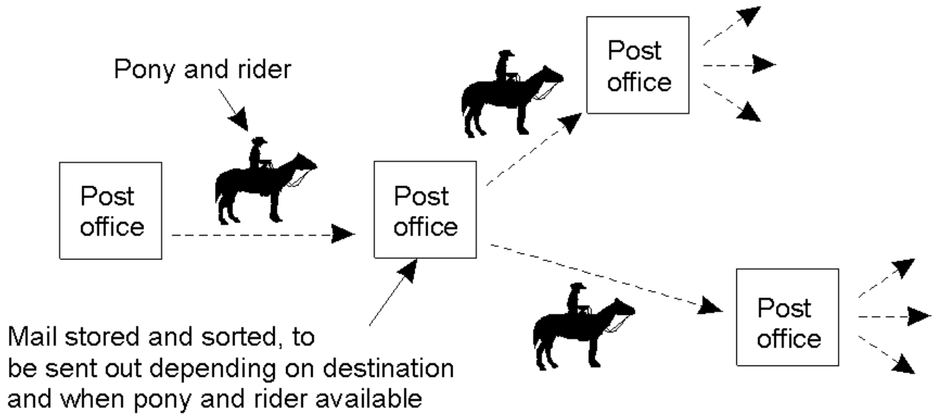
Message-Oriented Communication

- RPCs and RMIs are not always appropriate
  - Both assume receiver is executing when request is issued
  - Inherently synchronous
- Buffer-based network model
  - Hosts connected to communication servers
  - Message buffers at end hosts and communication servers
  - Example: Email system

Types of Communication

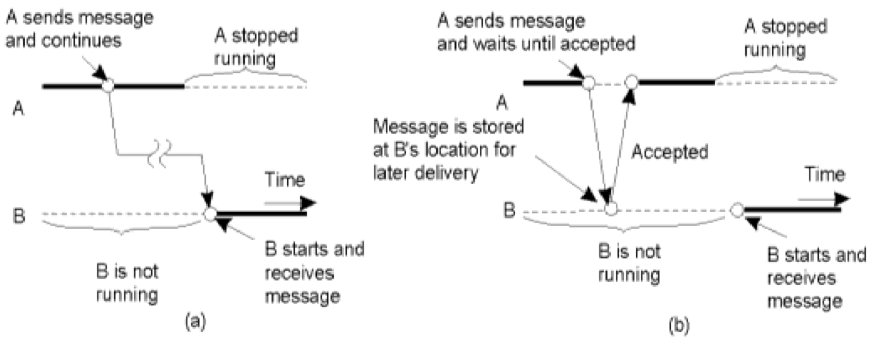
- **Persistence**
  - Persistent communication – Stores message until communicated to user
  - Transient communication – Stored only when sending and receiving processes are alive
    - Transport level protocols provide transient communication
- **Synchronicity**
  - Asynchronous – Sender continues after sending message
  - Synchronous – Sender blocks until message is stored at receiver's local buffer, delivered to receiver or processed by receiver

Example of Persistent Asynchronous Comm.


Pony and rider
Post office
Post office
Post office
Post office
Mail stored and sorted, to be sent out depending on destination and when pony and rider available

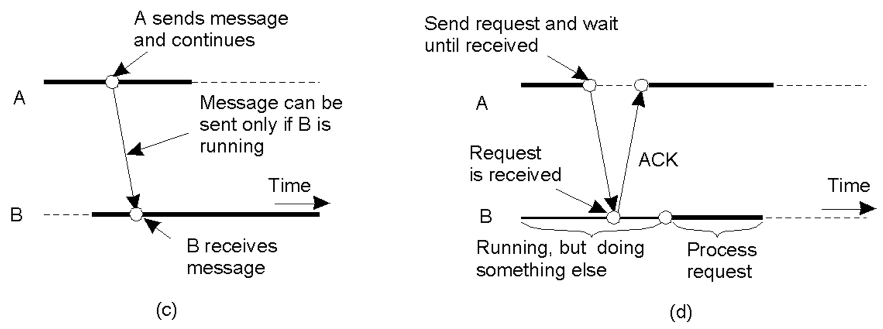
Persistent communication of letters back in the days of the Pony Express.

Persistence and Synchronicity in Communication (3)


A sends message and continues
A stopped running
A
B
Time
B is not running
B starts and receives message
(a)
A sends message and waits until accepted
A stopped running
A
Message is stored at B's location for later delivery
Accepted
Time
B
B is not running
B starts and receives message
(b)

a) Persistent asynchronous communication
b) Persistent synchronous communication

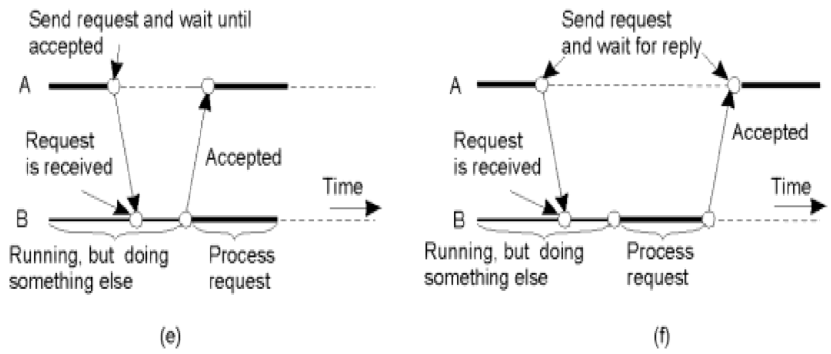
Persistence and Synchronicity in Communication (4)


A sends message and continues
A
Message can be sent only if B is running
Time
B
B receives message
(c)
Send request and wait until received
A
Request is received
ACK
Time
B
Running, but doing something else
Process request
(d)

c) Transient asynchronous communication

d) Receipt-based transient synchronous communication

Persistence and Synchronicity in Communication (5)


Send request and wait until accepted
A
Request is received
Accepted
Time
B
Running, but doing something else
Process request
(e)
Send request and wait for reply
A
Request is received
Accepted
Time
B
Running, but doing something else
Process request
(f)

e) Delivery-based transient synchronous communication at message delivery

f) Response-based transient synchronous communication

Message Oriented Transient Communication

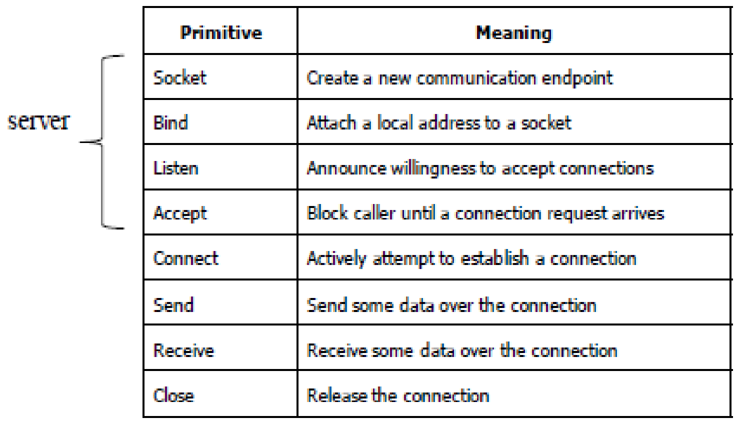
•Remote procedure calls and remote object invocations
 - Hide communication in Distributed systems & provide access transparency
-Assumes receiving side is executing at the time a request is issued
- Synchronous nature of RPC causes client to be blocked until its request has been processed

•Message oriented communication in distributed systems - synchronous message-queuing systems: allows processes to exchange information, even if other party is not executing at time communication is initiated.

Message-Oriented Transient Communication

Berkley sockets:
•Socket is a communication end point to which an application can write data that are to be sent out over underlying network, and from which incoming data can be read.
A socket forms an abstraction over actual communication end point that is used by local operating system for a specific transport protocol

Berkeley Sockets (1)

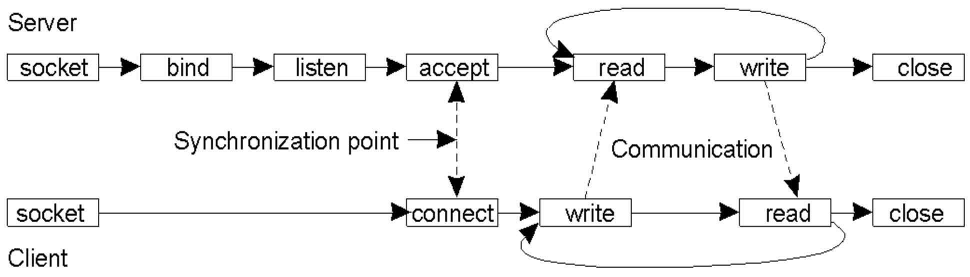


Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Socket primitives for TCP/IP.

Divyashikha Sethia
(DTU)

Berkeley Sockets (2)



Connection-oriented communication pattern using sockets.

Message Passing Interfaces (MPI)

Sockets were deemed insufficient
- They are at wrong level of abstraction by supporting only simple send and receive primitives.
- Have been designed to communicate across networks using general-purpose protocol stacks such as TCPIIP and are not considered suitable for the proprietary protocols developed for high-speed interconnection networks, such as those used in high-performance server clusters.

Message-Passing Interface or MPI: designed for parallel applications and as such is tailored *to transient communication*.
- assumes that serious failures such as process crashes or network partitions are fatal and do not require automatic recovery

Message Passing Interface (MPI)

•Assumes communication takes place within a known group of processes.
• Each group is assigned an identifier.
• (group/D, process/D) pair therefore uniquely identifies source or destination of a message

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 are none
MPI_irecv	Check if there is an incoming message, but do not block

Some of the most intuitive message-passing primitives of MPI.

MPI

•MPI_bsend:

- sender submits a message for transmission, which is first copied to a local buffer in MPI runtime system

-local MPI runtime system will remove the message from its local buffer and take care of transmission as soon as a receiver has called a receive primitive.

Message-Oriented Persistent Communication

Message-queuing systems/ **Message-Oriented Middleware (MOM)** : provide extensive support *for persistent asynchronous communication*

- Offer intermediate-term storage capacity for messages, without requiring either sender or receiver to be active during message transmission.

- Support message transfers that take minutes instead of seconds/milliseconds

Message Queuing Model

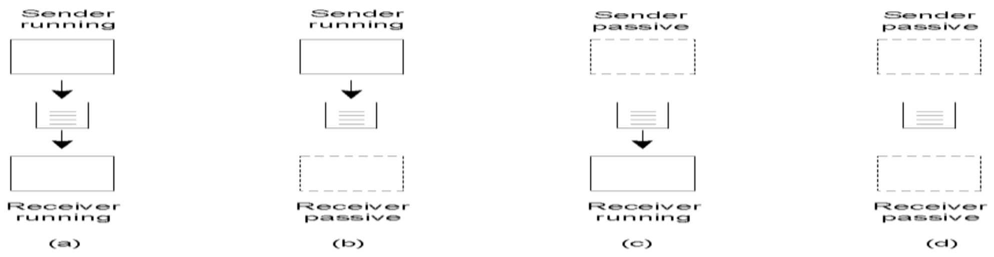
• Applications communicate by inserting messages in specific queues.

•Messages are forwarded over series of communication servers and eventually delivered to destination, even if it was down when message was sent

• Each application has its own private queue to which other applications can send messages.

• A queue can be read only by its associated application, but it is also possible for multiple applications to share a single queue.

Message-Queuing Model (1)



Four combinations for loosely-coupled communications using queues.

•No need for receiver to be executing when message is being sent to its queue
•No need for sender to be executing at moment its message is picked up by receiver
•Once message has been deposited in queue, it remains there until removed

Message Queues

•Messages contain data and must be properly addressed

•Addressing - system wide unique name of destination queue.

•Message size may be limited

•System can also take care of fragmenting and assembling large messages in a way that is completely transparent to applications

Message-Queuing Model (2)

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.

Basic interface to a queue in a message-queuing system.

•**Put** – nonblocking call
•**Get**:
 - blocking call untill queue is empty
 - authorized process can remove longest pending message in specified queue
 - searching for specific message in queue using a priority, or matching pattern
Poll – non-blocking, if queue is empty, or specific message could not be found, calling process simply continues

Message-Queue Model

Callback function:

- Process may install handler as callback function, which is automatically invoked whenever message is put into queue.

 - Can also be used to automatically start process that will fetch messages from queue if no process is currently executing. Eg: implementation of a daemon on receiver's side that continuously monitors queue for incoming

General Architecture of Message-Queuing System

•Source Queue:
 - messages can be put only into queues that are local to sender or on nearby machine on same LAN that can be reached via RPC

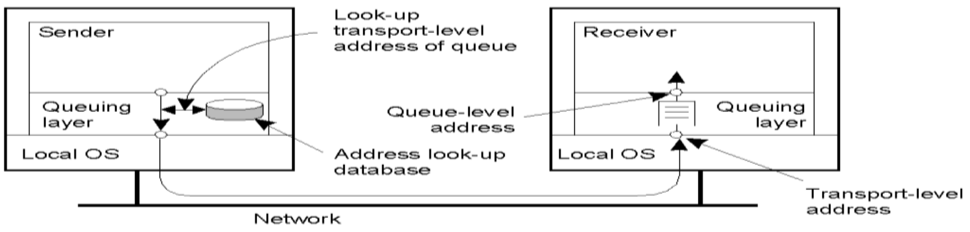
•Message can be read from local queues

•Message put on a source queue has specification of destination queue

•Message queuing system provides queues to sender and receiver and takes care of the transfer of messages

•Maintain a mapping of queue names to network locations in the form of database analogous to DNS for email.

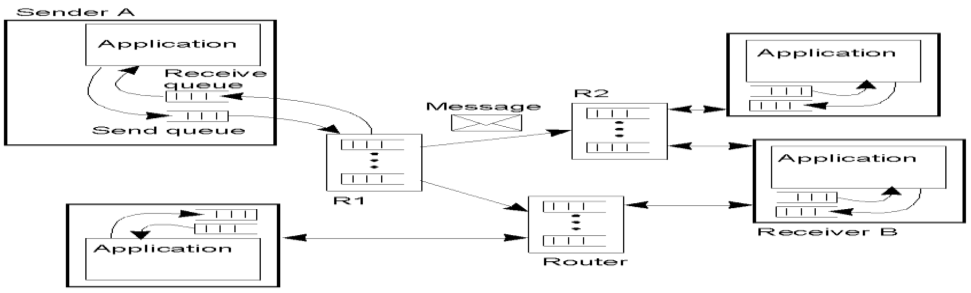
General Architecture of a Message-Queuing System (1)



Relationship between queue-level addressing & network-level addressing.

•Queues are managed by queue managers.
•Special queue managers operate as **routers, or relays**: they forward incoming messages to other queue managers
• Message queuing system gradually grow into application-level, overlay network, on top of existing computer network

General Architecture of a Message-Queuing System (2)



The general organization of a message-queuing system with routers.

•A puts a message for destination B in its local queue,
•Message is first transferred to nearest router, say Rl
•Router forwards message in direction of B: forwarded to router R2.
•Routers need to be updated when queues are added or removed.
•Queue manager has to know only where the nearest router is.

Other uses of relays

•Allow for secondary processing of messages

- messages may need to be logged for reasons of security or fault tolerance.

- transforming messages into format that can be understood by receiver

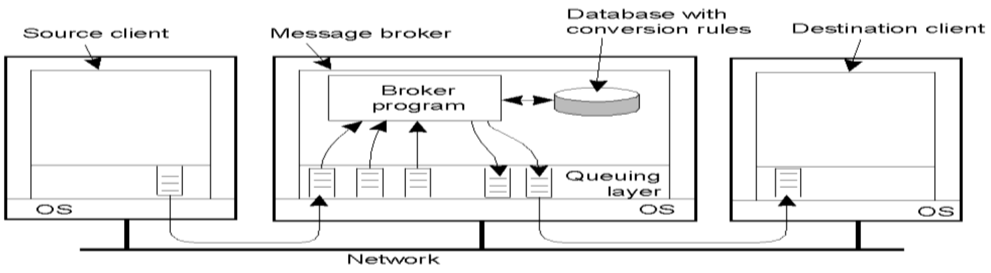
• Multicasting: incoming message is simply put into each send queue

Message Broker

In message-queuing systems, conversions are handled by special nodes known as message brokers.

A message broker acts as an application-level gateway in a message-queuing system.

Message Brokers



General organization of a message broker in a message-queuing system.

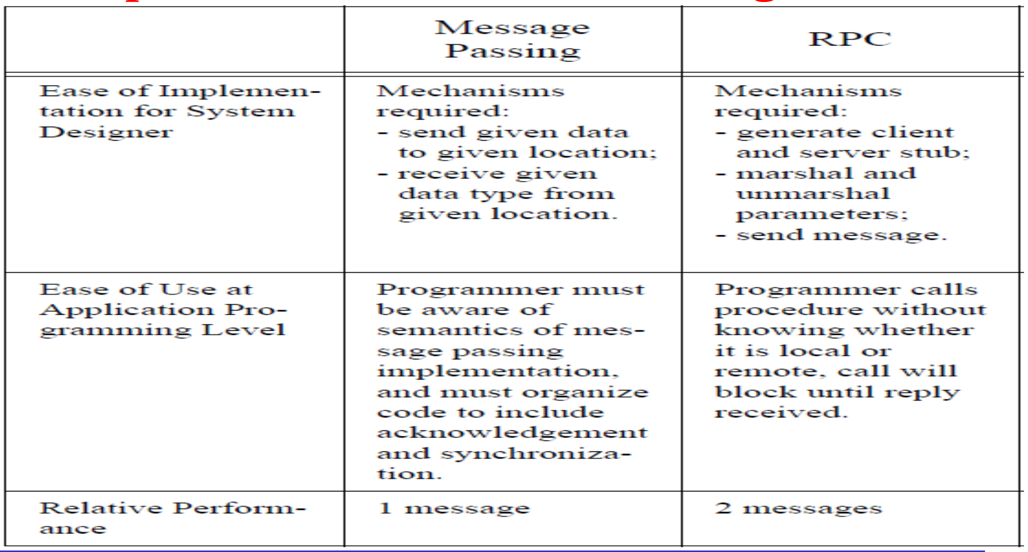
Message broker for advanced enterprise application integration (EAI):
 - applications send messages in the form of publishing, i.e publish a message on topic X, which is then sent to the broker.
 - Applications that have stated their interest in messages on topic X, that is, who have subscribed to those messages, will then receive these messages from the broker
 - repository of rules and programs that can transform a message TI to T2

Queuing Systems

•Wide range of applications: e-mail, workflow, groupware, and batch processing.
• Most important application area: integration of a (possibly widely-dispersed) collection of databases and applications into a federated information system

eg: a query expanding several databases may need to be split into subqueries that are forwarded to individual databases. Message-queuing systems assist by providing the basic means to package each subquery into a message and routing it to the appropriate database.

Comparison of RPC and Message Queues



	Message Passing	RPC
Ease of Implemen-tation for System Designer	Mechanisms required: - send given data to given location; - receive given data type from given location.	Mechanisms required: - generate client and server stub; - marshal and unmarshal parameters; - send message.
Ease of Use at Application Pro-gramming Level	Programmer must be aware of semantics of mes-sage passing implementation, and must organize code to include acknowledgement and synchroniza-tion.	Programmer calls procedure without knowing whether it is local or remote, call will block until reply received.
Relative Perform-ance	1 message	2 messages

http://www.deakin.edu.au/scitech/sit/dsapp/archive/techreport/TR-C95-20.pdf

Stream Oriented communication

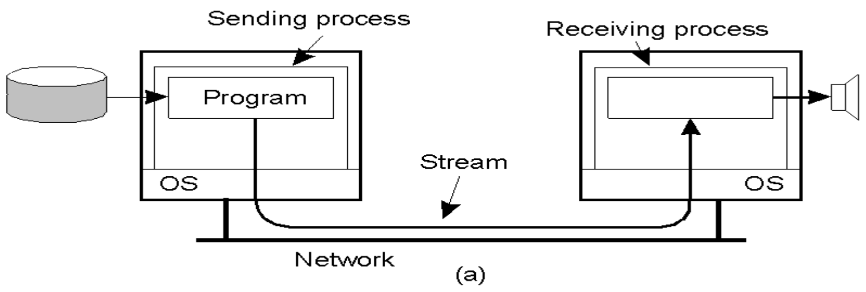
Communication **of complete units of information -** does not matter at what particular point in time communication takes place:
Eg: request for invoking a procedure, the reply to such a request, and messages exchanged between applications as in message-queuing systems

For communication **of streams (audio and video) timing is crucial**:
- original sound wave has been sampled at a frequency of 44, 100 Hz.
-Reproducing original sound, essential that samples in audio stream played out in order they appear in stream, and also at intervals of exactly 1/44,100 sec.
Playing out at a different rate will produce incorrect version of original sound.

Support for Continuous Media

•Exchange of time-dependent information requires continuous media
•Medium for information exchanges involves information representation encoding
•Continuous (representation) media: temporal relationships between different data items are fundamental to correctly interpret data
Eg:
Motion represented by series of images in which successive images must be displayed at uniform spacing T in time (30 - 40 msec per image)
Correct reproduction requires
- showing the stills in the correct order
- Also at constant frequency of 1/T images per second

Data Stream (1)



Setting up a stream between two processes across a network.

A data stream is nothing but a sequence of data units

Data Stream(2)

Asynchronous transmission mode: data items in stream are transmitted one after other, but there are no further timing constraints on when transmission of items should take place

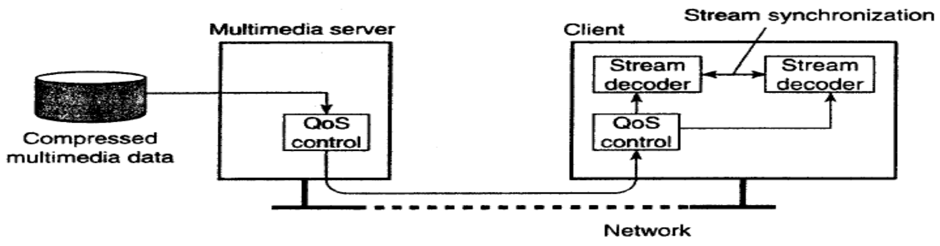
Synchronous transmission mode: there is a maximum end-to-end delay defined for each unit in a data stream

simple stream: consists of only a single sequence of data

complex stream: consists of several related simple streams, called substreams. (eg: stereo audio transmitted by complex stream consisting of two substreams, each used for a single audio channel requires two substreams are continuously synchronized)

Complex stream for transmitting a movie :single video stream, along with two streams for transmitting the sound of the movie in stereo. A fourth stream might contain subtitles for the deaf, or a translation into a different language than the audio.

Streaming stored data



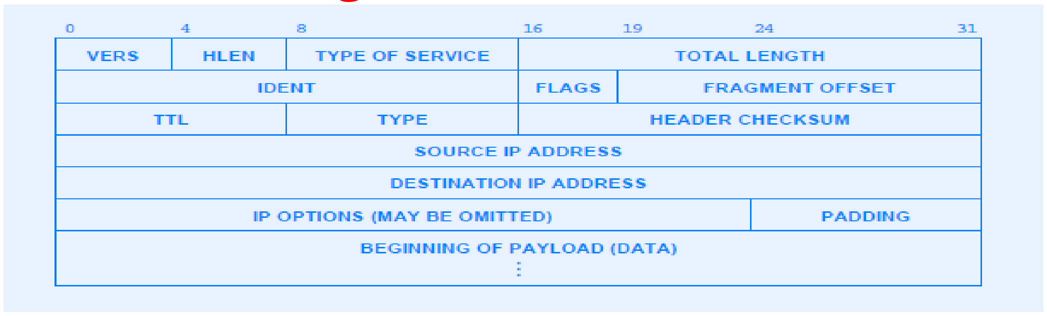
•**Streaming live data:**
- Data captured in real time and sent over network to recipients
- Less opportunities for tuning a stream.

•**Streaming stored data:**
- compressed substantially to reduce required storage and network capacity
- Important to controlling quality of transmission and synchronization

Streams and Quality of Service

Timing (and other nonfunctional) requirements are generally expressed as Quality of Service (QoS) requirements
Need to ensure the temporal relationships in a stream can be preserved
Qos for a stream requires to specify following properties:
1. The required bit rate at which data should be transported.
2. The maximum delay until a session has been set up (i.e., when an application can start sending data).
3. The maximum end-to-end delay (i.e., how long it will take until a data unit makes it to a recipient).
4. The maximum delay variance, or jitter.
5. The maximum round-trip delay.

Datagram Header Format



•Type of Service (TOS) : how datagram should be handled

0	1	2	3	4	5	6	7
Precedence			D	T	R	Unused	

Datagram Header Format - TOS

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Precedence			**D**	**T**	**R**	**Unused**	

•**Type of Service (TOS)** : how datagram should be handled
•**Precedence bits** 0 (normal) – 7 (network control) – sender can indicate importance to datagram eg:
- if control info then it must have precedence over data
- routers with precedence val 6,7 to exchange routing info over congested network

•Bit D- requests low delay (eg: keystrokes from user to remote computer)
•Bit T – requests high throughput (eg: bulk file transfer to travel over a high capacity link)
•Bit R requests high reliability
•Bits give a hint to the routing algorithm to choose over multiple paths for the desired request.

Differentiated Services (DiffServ)

0	1	2	3	4	5	6	7
CODEPOINT						UNUSED	

•Late **1990's redefined meaning of the Service Type field for DiffServ for Qos**
•First six bits form codepoint **– Differentiated services Code point (DSCP),** last 2 bits are unused
•Few services and codepoints under it.
•Backward compatibility – when last three bits of the codepoint are zero precedence bits define class of service xxx000
The default DSCP is 000 000. Class selector DSCPs are values that are backward compatible with IP precedence. When converting between IP precedence and DSCP, match the three most significant bits. In other words:
IP Prec **5** (101) maps to IP DSCP **101 000**
•Router honors original precedence scheme for high priority traffic for val 6 and 7 even when it is set for Diff services

Differentiated Services (DiffServ)

Pool	Codepoint	Assigned By
1	xxxxx0	Standard
2	xxxx11	Local/exper
3	xxxx01	Local/exper

The 64 codepoints are divided into three administrative groups

Enforcing Qos - IP

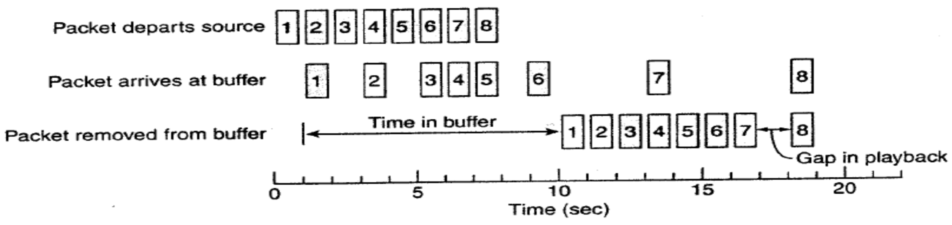
•Differentiating classes of data by means of its differentiated services.

•Sending host marks outgoing packets as belonging to one of several classes:

- **Expedited forwarding class**: specifies packet should be forwarded by current router with absolute priority

- **Assured forwarding class**: traffic is divided into four subclasses, along with three ways to drop packets if network gets congested. Defines a range of priorities assigned to packets, and allows applications to differentiate time-sensitive packets from noncritical ones.

Enforcing Qos - Buffers



•Buffers to reduce jitter:
- packets are delayed with certain variance when transmitted over network
-receiver simply stores them in buffer for maximum amount of time.
-Allows receiver to pass packets to application at regular rate, since there will be enough packets entering buffer to be played back at that rate

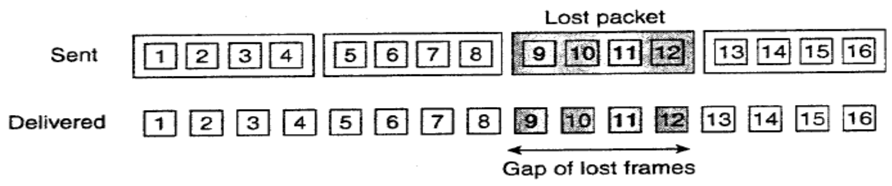
•Receiver's buffer: 9 seconds of packets
- packet #8 took 11 seconds to reach the receiver when buffer will have been completely emptied => gap in the playback at the application
- Increased buffer size solves it: but increases delay of start of stream

Enforcing Qos – Forward Error Correction

•Packets may be lost: compensate loss in quality of service by applying error correction techniques

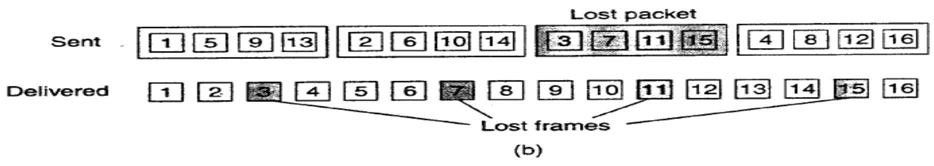
•Forward error correction (FEC): encode outgoing packets in such a way that *any k out of n received packets is enough to reconstruct k correct packets. (k<n)*

Effect of packet loss



•Single packet may contain multiple audio &video frames
•On packet loss receiver gets a large gap when playing out frames.

Effect of packet loss – Solution: interleaving frames



- On packet loss resulting gap in successive frames is distributed over time.
- This approach requires a larger receive buffer in comparison to noninterleaving, and thus imposes a higher start delay for the receiving application

  - Noninterleaved transmission buffer requirements: 1

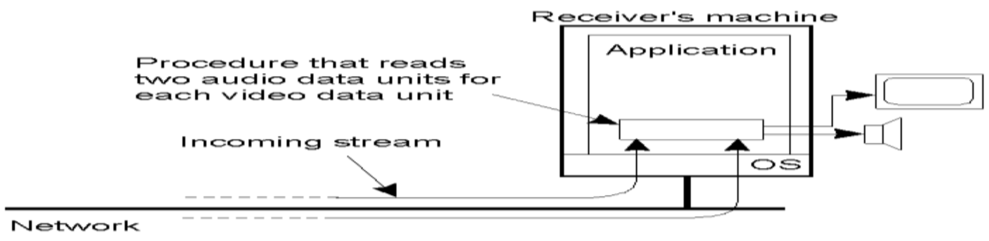
  - Interleaved transmission buffer requirements: 4

Stream Synchronization

•Complex stream require mutually synchronized : maintaining temporal relations between streams

•Types:
1. **Between discrete data stream and continuous data stream** (eg: slide show on the Web that has been enhanced with audio)
 - Continuous audio stream is to be synchronized with the discrete slides.
2. **Between continuous data streams** (eg:
 - playing movie in which video stream needs to be synchronized with audio - lip synchronization
 - playing stereo audio stream consisting of two substreams, one for each channel requires two substreams are tightly synchronized for proper playout: difference of more than 20 usee can distort stereo effect.

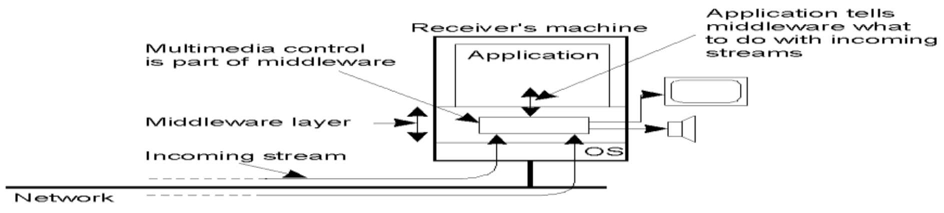
Synchronization Mechanisms (1)


Receiver's machine
Application
Procedure that reads two audio data units for each video data unit
Incoming stream
OS
Network

The principle of explicit synchronization on the level data units.

Drawback: application is made completely responsible for implementing synchronization while it has only low-level facilities available

Synchronization Mechanisms (2)


Receiver's machine
Application tells middleware what to do with incoming streams
Multimedia control is part of middleware
Application
Middleware layer
Incoming stream
OS
Network

The principle of synchronization as supported by high-level interfaces.

better approach: application accesses interface that allows to easily control streams & devices

video display and audio has a control interface that allows :

- to specify the rate at which images/audio should be displayed
- facility to register user-defined handler that is called each time *k new images/ audio have arrived*

application developer writes monitor program consisting of two handlers, one for each stream, to check if video and audio stream are synchronized, and adjust rate at which video or audio units are presented.

Multicast communication

•ISPs have shown to be reluctant to support multicasting
•Application-level multicasting techniques provide communication paths for multicast
 - nodes organize into overlay network, which is used to disseminate information to its members
 - network routers are not involved in group membership
 - connections between nodes in the overlay network may cross several physical links (routing messages within the overlay may not be optimal in comparison to what could have been achieved by network-level routing)

Overlay Construction

Two ways of overlay construction:

1. Nodes organize themselves directly into tree, and there is a unique (overlay) path between every pair of nodes

2. Nodes organize into mesh network in which every node will have multiple neighbors and there are multiple paths between every pair of nodes.
   - Provides better robustness if connection breaks due to node failure

Application multicast Scribe

Application multicast uses peer to peer DHT scheme for communication
Original DHT for Scribe was Chord
Scribe actually build over Pastry

Starting multicast session in in Chord

Node S wants to start a multicast session :

– generates a multicast identifier: mid (randomly-chosen 160-bit key)

- looks up succ(mid), which is node responsible for key mid , and promotes it to become root of multicast tree that will be used to sending data to interested nodes like say node P

-Node P executes LOOKUP(mid) – lookup request for joining multicast group mid routed to succ(mid).

-While being routed to succ(mid) join request will pass several nodes

Starting multicast session in in Chord

-If it reaches node Q which has never seen join request for mid before becomes forwarder to group mid

P will become child of Q and Q will forward join request further to root.

```
 Q
/
P
```

-If next node is R which has also not seen join req for mid it will become forwarded. Now Q become child of R

```
   R
  /
  Q
 /
P
```

Multicast tree in Chord

-For any next join request by node X which reaches Q or R it would already been a forwarder so need not send the join request to root since Q and R are part of the multicast tree.

-Q and R are pure forwarders that act as helpers

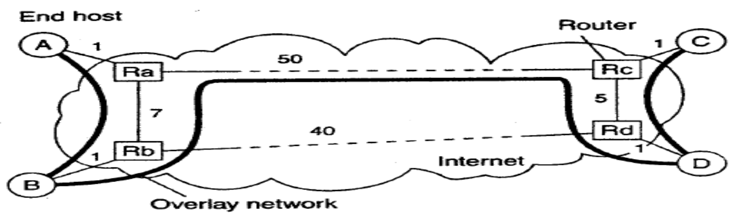
-P which explicitly requested to join is a forwarder

-Multicast tree consists of set of these nodes helpers and forwarders.

- Multicasting is now simple: a node merely sends a multicast message toward the root of the tree by again executing the LOOKUP(mid) operation, after which that message can be sent along the tree

Overlay Construction

Overlay Tree may not be an efficient tree since it does not take in account any performance metric purely based on logical routing of messages overoverlay



Simple overlay with A as the root distributing to B,D and C

Cost of transmission over each link

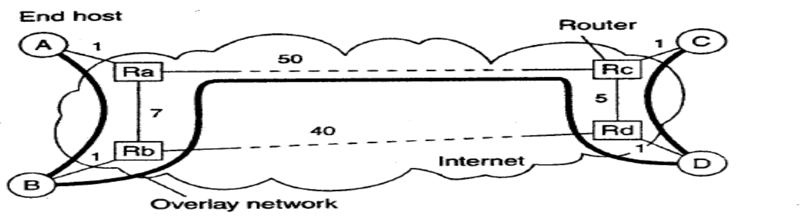
- <.*B,Rb»*, <*Ra*, *Rb*>, *«Rc, Rd»*,and <*D, Rd» traversed twice.*

- Would be efficient to have overlay link from A-C instead of B->D (save double traversal across links *«Ra, Rb> and <Rc, Rd>*

Quality of application-level multicast tree

1. Link stress is defined per link and counts how often packet crosses same link . Link stress > 1 => although at ;logical level packet may be forwarded along two different connections, part of those connections may actually correspond to same physical link

Quality of application-level multicast tree



2. Stretch or Relative Delay Penalty (RDP) measures the ratio in the delay between two nodes in overlay, and delay that those two nodes would experience in underlying network

Eg: *B to C follow the route B ~ Rb ~ Ra ~ Rc ~ C, having a total cost* of 59 units. However, messages would have been routed in the underlying network along the path *B ~ Rb ~ Rd ~ Rc ~ C, with a total cost of 47 units*

RDP = 1.255

minimize the aggregated stretch, or similarly, the average RDP measured over all node pairs

Quality of application-level multicast tree

3. Tree cost is a global metric, generally related to minimizing the aggregated link costs.

Eg: if cost of a link : delay between its two end nodes, then optimization of tree requires finding a minimal spanning tree in which the total time for disseminating information to all nodes is minimal.

These metrics can be used to determine the best parent for new node joining a multicast

Selecting best parent in the overlay

•multicast group has an associated and well-known node that keeps track of the nodes that have joined the tree

•new node issues a join request, it contacts this rendezvous node to obtain a (potentially partial) list of members

•select best member that can operate as the new node's parent in the tree

Selecting best parent in the overlay

•Methods:
i. If multicast group has only a single source. In this case, the selection of the best node is source to make stretch 1 . To avoid overload of source explore nodes in the overlay which have less than k children.

Switch Trees

•Switch-trees are peer-to-peer algorithms for building and improving end-host multicast trees.
•Nodes switch parents to reduce tree cost or lower source-member latency.
•Node switches parents by disconnecting from its parent and reconnecting to a new parent.
•Well chosen new parent, improves overall tree performance.

Paper: **End-Host Multicast Communication Using Switch-Trees Protocols**

Switch-Trees

•In a multicast tree with a single source as root, a node P can switch parents by dropping the link to its current parent in favor of a link to another node
•Constraints:
-new parent can never be a member of subtree rooted at P (as this
would partition the tree and create a loop)
- new parent will not have too many immediate children (avoid overload)
•switch parents: to optimize route to source, effectively minimizing the delay when a message is to be multicast.
•Node failure: Switch trees node notices that its parent has failed, it simply attaches itself to the root optimization protocol can proceed as usual and will eventually place the node at a good point in the multicast tree.

Switch Tree

•Switch to another parent based on lower delay to potential parent:
 - each node connects to a parent with minimum delay then resulting tree will be minimal
Eg: Node P receives information about neighbors of its parent R = these will be parent R's parent and siblings of parent R.
P evaluates delay to these neighbors and can find a node Q which has min delay and can switch to Q
Prevention of loops:
If Q has an outstanding switch request it will refuse any incoming switch request

Resources

•Distributed Systems – Tanenbaum
• **Differentiated Services : Wikepedia, Computer Networks: Forouzan**
•: End-Host Multicast Communication Using **Switch-Trees** Protocols