DISTRIBUTED COMPUTER SYSTEMS

MESSAGE ORIENTED COMMUNICATIONS

Dr. Jack Lange
Computer Science Department
University of Pittsburgh

Fall 2015

Outline

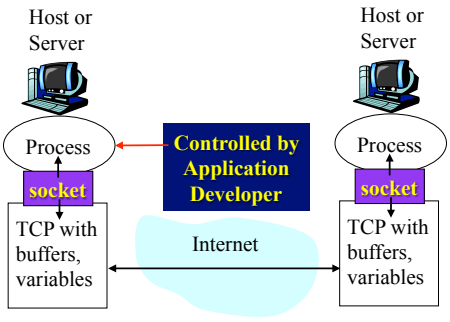
- Message Oriented Communication
  - Sockets and Socket API
  - Message Passing Interface – MPI
    - Architecture and Design Issues
  - Message Oriented Persistent Communication
    - Message Queues
    - IBM WebSphereMQ
- Stream Oriented Communication
  - Continuous Media Requirements
  - Flow Specification and QoS Support
  - Delay and Jitter Control
  - Packet Loss Control
- Conclusion

Message Oriented Communication

Sockets

Sockets

- A Socket is a communication end point
  - Processes can send and receive messages to and from its socket
- Abstraction over the communication end point that is by the local OS for a specific transport protocol
  - Sending process relies on transport infrastructure which brings message to socket at receiving process



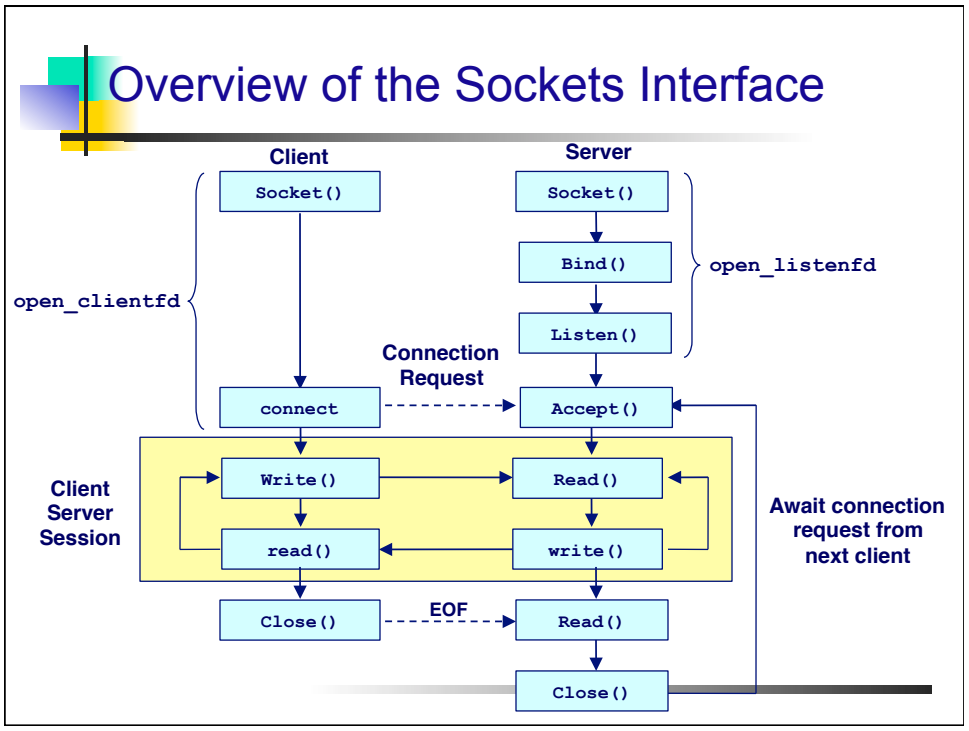
Socket Arguments – Protocol

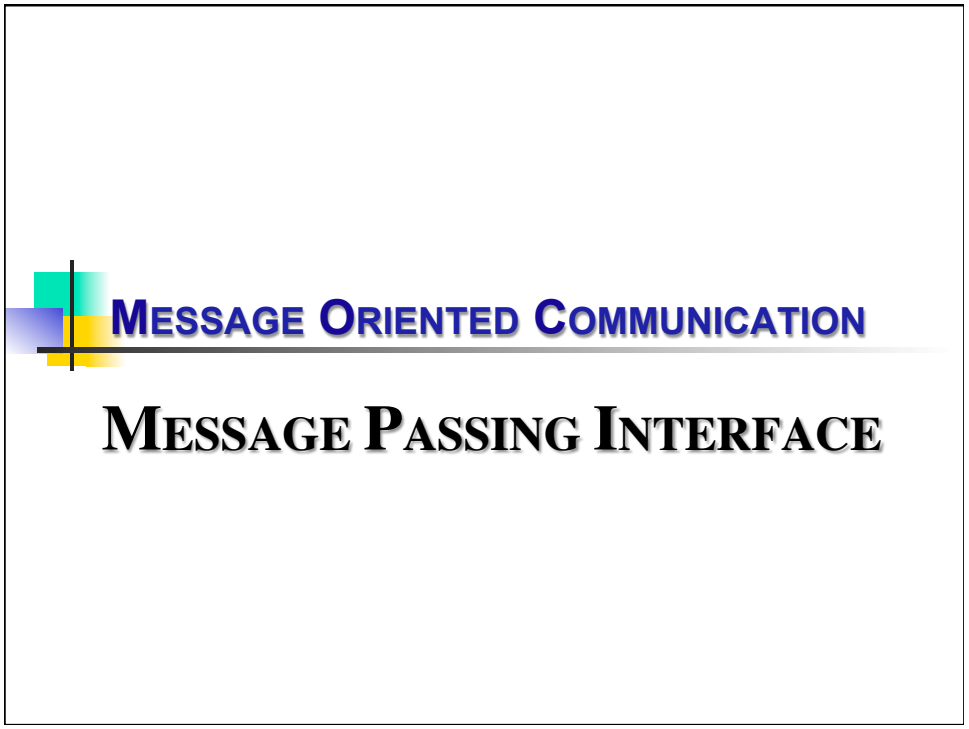
- **Function** `socket – int socket ( int family, int type, int protocol)`
  - `Return > 0 if success and -1 if error`

Family	Type	Protocol	Actual Protocol
AF_INET	SOCK_DGRAM	IPPROTO_UDP	UDP
AF_INET	SOCK_STREAM	IPPROTO_TCP	TCP
AF_INET	SOCK_RAW	IPPROTO_ICMP	ICMP
AF_INET	SOCK_RAW	IPPROTO_RAW	raw IP

Sockets Creation

- The system call socket() is used to create a new socket
  - Int sockfd = socket(PF_INET, SOCK_STREAM, 0);
- PF_INET = Protocol Family (Internet)
- SOCK_STREAM = TCP
  - TCP is a reliable protocol
- SOCK_DGRAM = UDP
  - UDP is a “best effort” protocol, no guaranteed reliability


Overview of the Sockets Interface
Client
Server
Socket()
Socket()
Bind()
Listen()
open_clientfd
open_listenfd
Connection Request
connect
Accept()
Write()
Read()
read()
write()
Client Server Session
Await connection request from next client
Close()
EOF
Read()
Close()

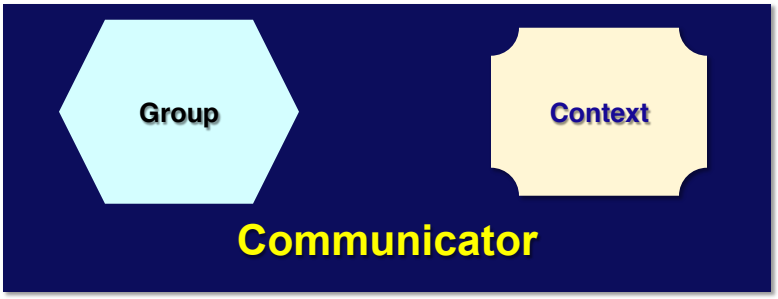

MESSAGE ORIENTED COMMUNICATION
MESSAGE PASSING INTERFACE

What is MPI?

- A message-passing library specifications for parallel computers, clusters, and heterogeneous networks
  - Extended message-passing model
- Communication modes
  - Standard, synchronous, buffered, and ready.
- Designed for parallel applications and tailored to **transient** communications
- Provides access to advanced parallel hardware
  - It assumes that serious failures are fatal and do not require automatic recovery
    - For example, process crashes or network disconnect

Group and Context

- MPI assumes communication takes place with a group of processes – (groupID, procID)
  - Use in lieu of a transport-level address
- MPI ties the concepts of **process group** and **communication context** into a **communicator**



MPI Abstractions

- A ***process group*** is high-level abstraction, visible to users
- A ***context*** is a system-defined object that uniquely identifies a communicator – Mechanism to isolate messages in distinct libraries and the user programs from one another
  - A message sent in one context can't be received in other contexts.
  - The communication context is low-level abstraction, not visible to users
- A ***communicator*** is a data object that specializes the scope of a communication.
  - MPI_COMM_WORLD is an initial communicator, which is predefined and consists of all the processes running when program execution begins

MPI Communication Semantics

- *Point-to-point communication* is the basic concept of MPI standard and fundamental for send and receive operations for typed data with associated message tag.
  - Using the point-to point communication, messages can be passed to another process with explicit message tag and implicit communication context.
  - Each process can carry out its own code in MIMD style, sequential or multi-threaded.
  - MPI is made *thread-safe* by not using global state.

MPI Communication Primitives

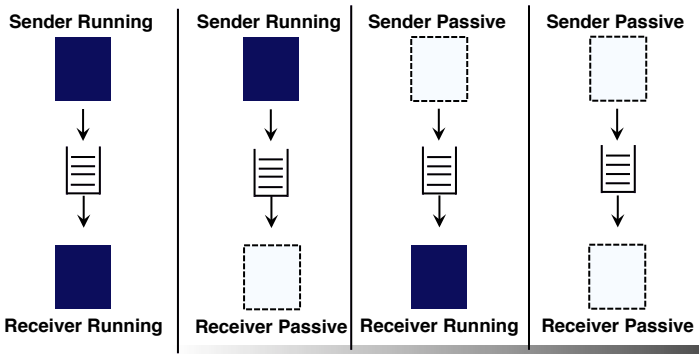
Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
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

MESSAGE ORIENTED COMMUNICATION

MESSAGE QUEUING SYSTEMS

Message-Queuing Model

- MQM, aka Message Oriented Middleware, provides support for **persistent asynchronous** communication
  - MOM offers intermediate-term message storage capacity, without requiring either the sender of receiver to be active



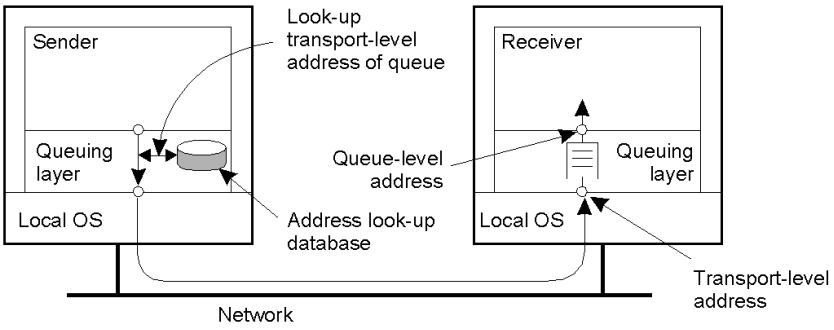
MOM Basic Semantics and Interface

- MOM supports **loosely-coupled in time** semantics
  - Senders are given only guarantees that message will eventually be inserted in the receiver's queue
  - No guarantee about **if** or **when** will messages be read
    - Recipient-behavior dependent
  - Basic interface to a queue in 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.

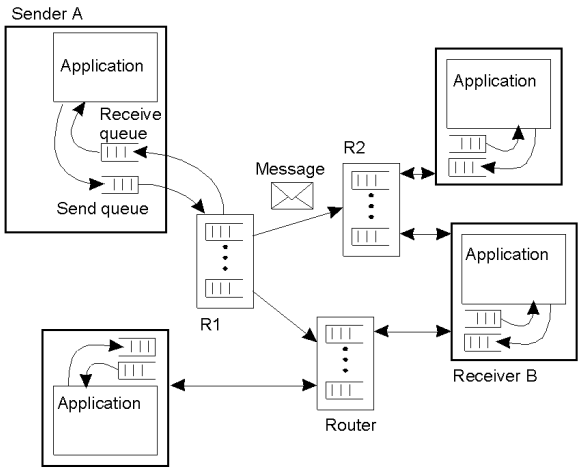
MQM General Architecture

- Source and destination queues are distributed across multiple machines
- The relationship between queue-level addressing and network-level addressing.



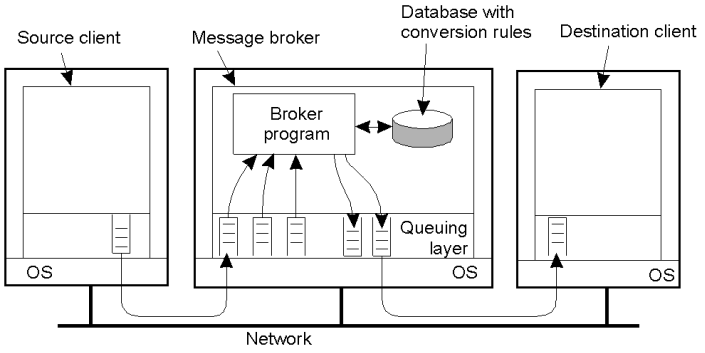
MQM Generalized Architecture – Routers

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



Message Brokers

- Common message format, typically used in traditional networks, does not generally work for MQM systems
  - Mostly due to the diversity of applications, which reduces the best common format to a sequence of bytes
- Brokers – Message broker in a message-queuing system.

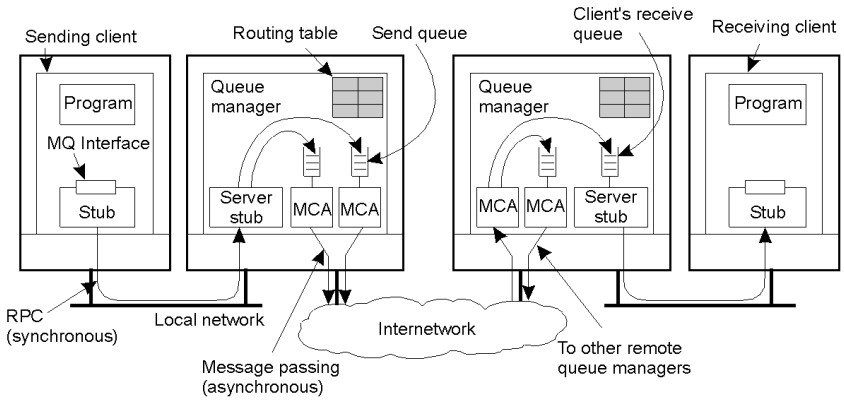


MQM Brokers Functionalities

- Message brokers act as an application-level gateway
  - Their main purpose is to convert incoming messages to a format that can be understood by the recipient
  - Properly match end-of-record delimiters between database applications
  - More advanced broker are designed to handle conversations between two different database applications
    - Difficult task, as information and semantic "mapping" between two different applications may not always be possible
- Brokers are commonly used in Enterprise Application Integration for ***mediation***
  - Publish/Subscribe is the typical communication model
  - Brokers "matches" messages to applications' interests

WebSphere MQ – MQ Example

- General organization of IBM's WebSpher MQ message-queuing system – Queue Managers (QM), Message Channels (MC), and Message Channel Agents (MCA)



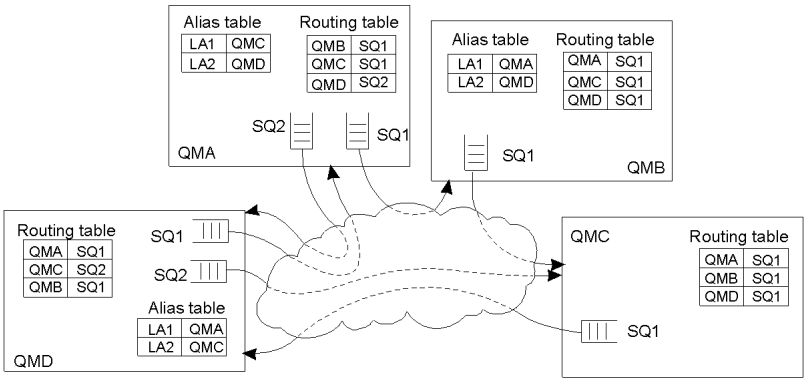
WebSphereMQ MCs and MCAs

- Message channels are an abstraction of transport level connections.
  - A MC is a unidirectional, reliable connection between a sending and a receiving QM
    - Internet-based MCs are implemented as TCP connections
- The two ends of a MC are managed by a MCA

Attribute	Description
Transport type	Determines the transport protocol to be used
FIFO delivery	Indicates that messages are to be delivered in the order they are sent
Message length	Maximum length of a single message
Setup retry count	Specifies maximum number of retries to start up the remote MCA
Delivery retries	Maximum times MCA will try to put received message into queue

WebSphereMQ Message Transfer

- Generalized organization queuing network using routing tables and aliases.
  - Routing Table Entry: <DestMQ, sendQ>, where DestMQ is the name of the destination QM and senQ is the name of the local send queue



Message Transfer Primitives

- WebSphereMQ Message Queue Interface Primitives

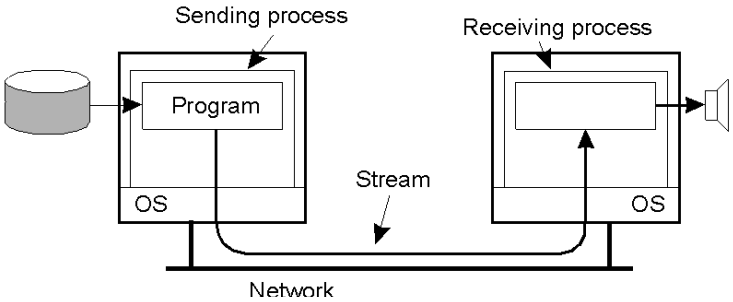
Primitive	Description
MQopen	Open a (possibly remote) queue
MQclose	Close a queue
MQput	Put a message into an opened queue
MQget	Get a message from a (local) queue

MESSAGE ORIENTED COMMUNICATION

STREAM ORIENTED COMMUNICATION

Stream Communication Model

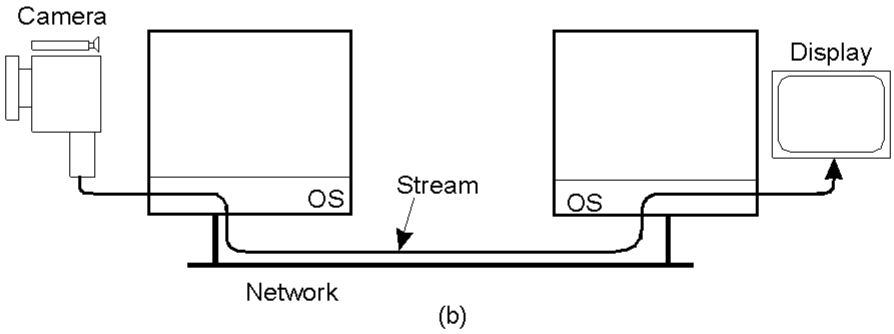
- Support for data streams, time-dependent information such as audio and video, requires communication models that go beyond those that support independent and complete units of information



General Architecture for Streaming Stored Media over a Network

Data Streams – Point-to-Point

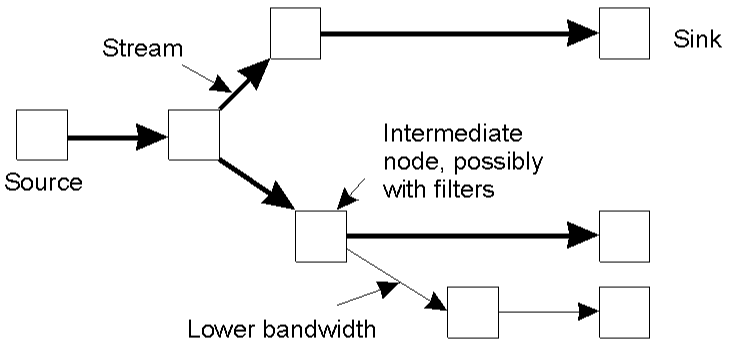
- Setting up a stream directly between two devices.



(b)

Data Streams -- Multicasting

- An example of multicasting a stream to several receivers.



Streams, flows and QoS

- The Quality of Service (QoS) of an application is the "look and feel" of how the stream will flow
- QoS Support depends on two components
  - Specification of the flow/stream
  - Enforcing the QoS requirements specification of the supported flows
- The QoS support implementation depends on many, many factors,
  - It may require the "Flow Admission Protocol"
    - Given a new flow specifications, and a set of currently supported flows, a decision has to be made whether the new flow is to be **accepted** or **rejected**

Streams and Quality of Service

- QoS Flow Specification Parameters
  - The required bit rate at which data should be transported.
  - The maximum delay until a session has been set up
  - The maximum end-to-end delay .
  - The maximum delay variance, or jitter.
  - The maximum round-trip delay.

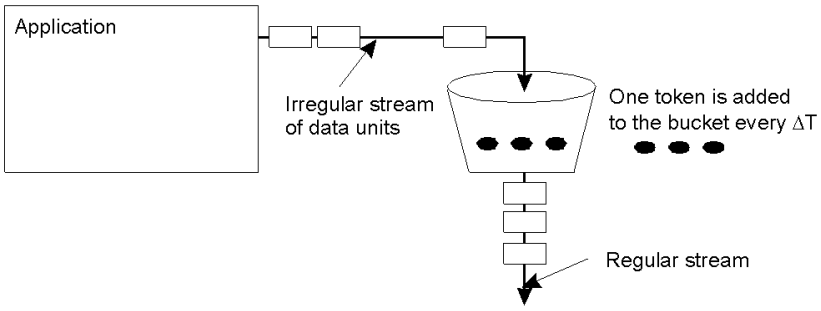
QoS Specification and Services

- A flow specification model and services

Characteristics of the Input	Service Required
▪ Maximum data unit size (bytes) ▪ Token bucket rate (bytes/sec) ▪ Token bucket size (bytes) ▪ Maximum transmission rate (bytes/sec)	▪ Loss sensitivity (bytes) ▪ Loss interval (μsec) ▪ Burst loss sensitivity (data units) ▪ Minimum delay noticed (μsec) ▪ Maximum delay variation (μsec) ▪ Quality of guarantee

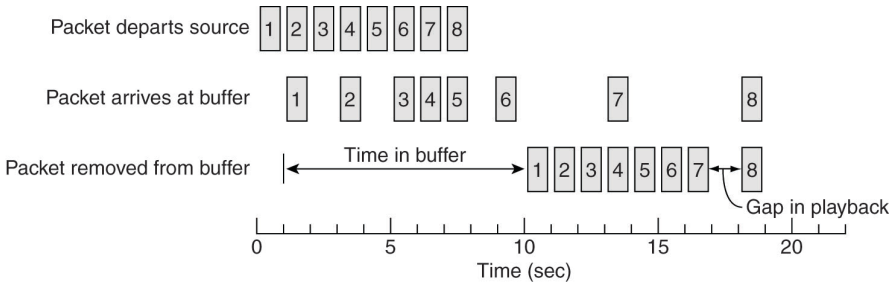
Specifying QoS – Token Bucket

- The principle of a token bucket algorithm.



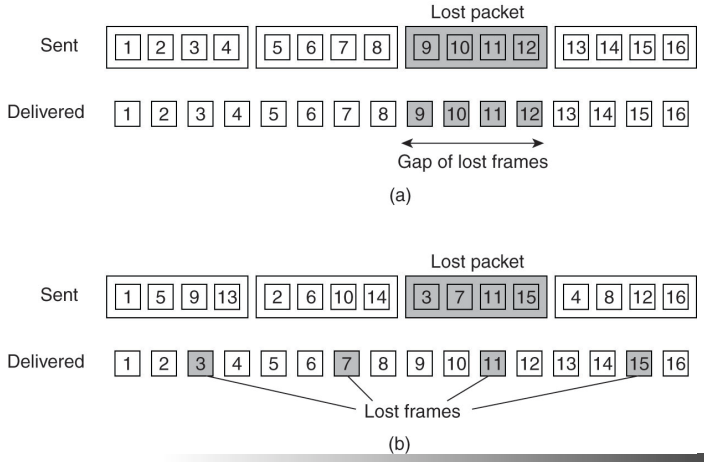
Enforcing QoS – Binding Jitter

- Buffers can be used to reduce jitter
  - Jitter represents the delay variation between consecutive packets
    - Typically measured as the variance of packet interarrival times

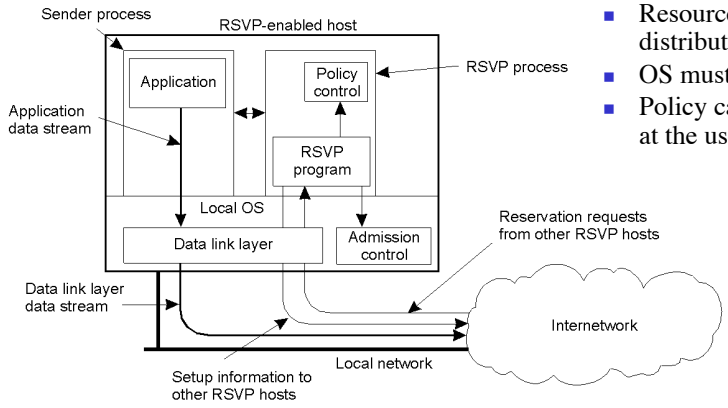


Enforcing QoS – Mitigating Packet Loss

- The effect of packet loss (a) non interleaved transmission and (b) interleaved transmission.



Setting Up a Stream

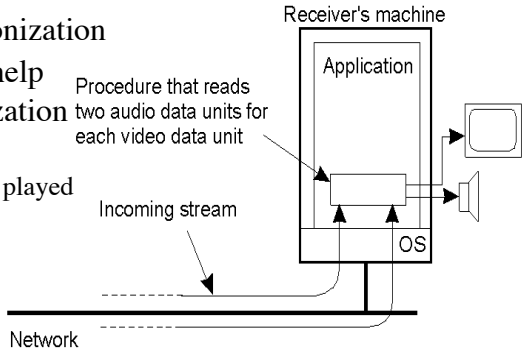


- Resource reservation in a distributed system
- OS must support mechanism
- Policy can be implemented at the user level

- Reservations
  - Statistical (good on average) vs guaranteed (good for important stuff)
- No standards, much research, no prevalent results

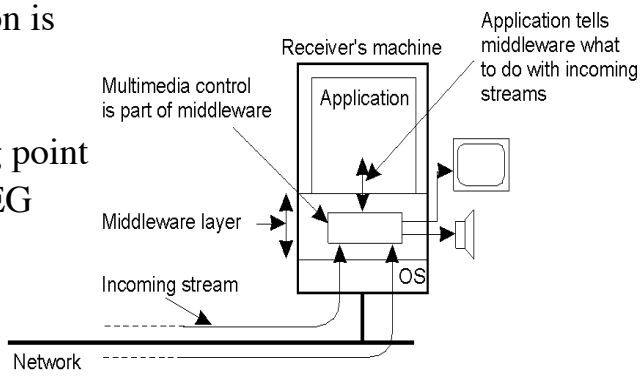
Synchronization Mechanisms

- Synchronization of data and of flows is difficult, mainly due to the unpredictability of computing systems
  - Interrupts may disturb media continuity
  - Priority inversion may occur – High priority task blocked by a low priority task
  - Different flows must be synchronized
- Application-level Synchronization
- Hardware can be used to help specially when synchronization requirements are strict
  - Separate streams should be played within 30 μ-seconds



Synchronization Mechanisms

- Synchronization supported by high-level interfaces.
- Middleware multiplexes all substreams into a single stream and demultiplex the streams at the receiver.
- Synchronization is handled at multiplexing/ demultiplexing point
- Example: MPEG



Conclusion

- Message Oriented Communication
  - Sockets and Socket API
  - Message Passing Interface – MPI
    - Architecture and Design Issues
  - Message Oriented Persistent Communication
    - Message Queues
    - IBM WebSphereMQ
- Stream Oriented Communication
  - Continuous Media Requirements
  - Flow Specification and QoS Support
  - Delay and Jitter Control and Reliability