CprE 450/550X
Distributed Systems and Middleware

Inter-process Communication

Yong Guan
3216 Coover
Tel: (515) 294-8378
Email: guan@ee.iastate.edu

Feb. 11, 2003

Selected Topics of Term Papers

- We will discuss the topics for term papers next class.
- I will list the topic list on the course web page tonight or tomorrow.
- Each student should finish his/her own term paper.

3

Readings for Today's Lecture

- References
  - **Chapter 2 of "Distributed Systems: Principles and Paradigms"**
  - **Chapter 11 of "Java Network Programming and Distributed Systems"**

4

Jave RMI

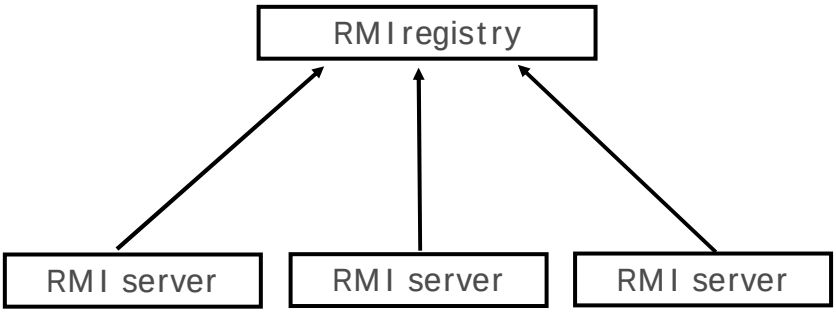
- RMI: A Java technology that allows one JVM to communicate with another JVM and have it execute an object method.

- RPC and RMI
  - **RPC supports multiple languages, whereas RMI only support Java**
  - **RMI deals with objects, but RPC does not support the notion of objects**
  - **RPC offers procedures (not associated with a particular object)**

5

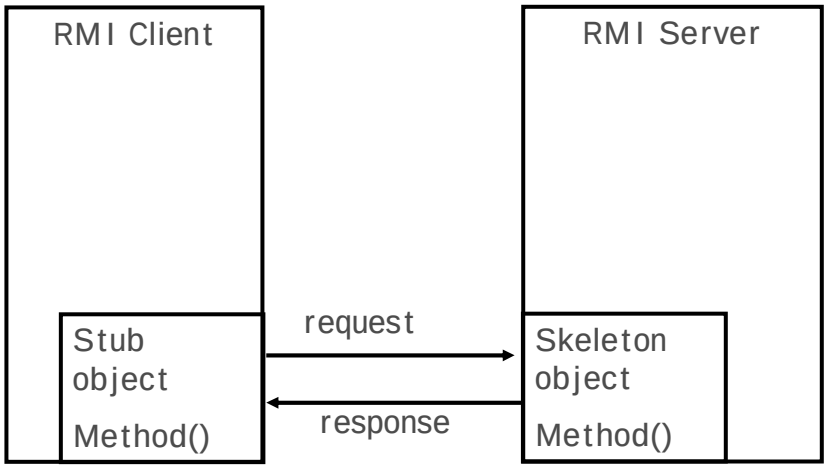
How RMI works

- The format used by RMI for representing a remote object reference: rmi://hostname:port/servicename



6

How RMI works



7

Define a RMI Service Interface

```
Public interface RMILightBulb extends java.rmi.Remote
{
   Public void on() throws java.rmi.remoteexecution;
   Public void off() throws java.rmi.remoteexecution;
   Public boolean ison() throws java.rmi.remoteexecution;
}
```

8

Implement a RMI Service Interface

```
Public class RMILightBulbImpl
     extends java.rmi.server.UnicastRemoteObject
     implements RMILightBulb
{
     Public RMILightBulbImpl () throws java.rmi.remoteexecution
     {setBulb(false);}

     Private boolean lighton;

     Public void on() throws java.rmi.remoteexecution
     {        setBulb(true); }

     Public void off() throws java.rmi.remoteexecution
     {        setBulb(false); }

     Public boolean ison() throws java.rmi.remoteexecution
     {return getBulb();}

     Public void setBulb(boolean value)
     {lighton = value;}

     Public void getBulb()
     {return lighton;}

}
```

9

Create Stub and Skeleton Classes

Rmic RMILightBulbImpl

Two files would be produced:

- **RMILightBulbImpl_Stub.class**
- **RMILightBulbImpl_Skeleton.class**

10

Create a RMI Server

```
import java.rmi.*;
Import java.rmi.server.*;

Public class LightBulbServer
{
    Public static void main(String args[])
    {
        Try{
                RMILightBulbImpl bulbService=new RMILightBulbImpl();
                RemoteRef location = bulbService.getRef();

                String registry = args[0];

                String registration = "rmi://"+registry+"/RMILightBulb";

                Naming.rebind(registration, bulbService);
        }
    }
}
```

11

Create a RMI Client

```
import java.rmi.*;

Public class LightBulbClient
{
    Public static void main(String args[])
    {
        Try{
            String registry = args[0];

            String registration = "rmi://"+registry+"/RMILightBulb";

            Remote remoteService = Naming.lookup(registration);

            bulbService.on();
            system.out.println(bulbService.isOn());

            bulbService.off();
            system.out.println(bulbService.isOn());


        }
    }
}
```

12

Running the RMI system

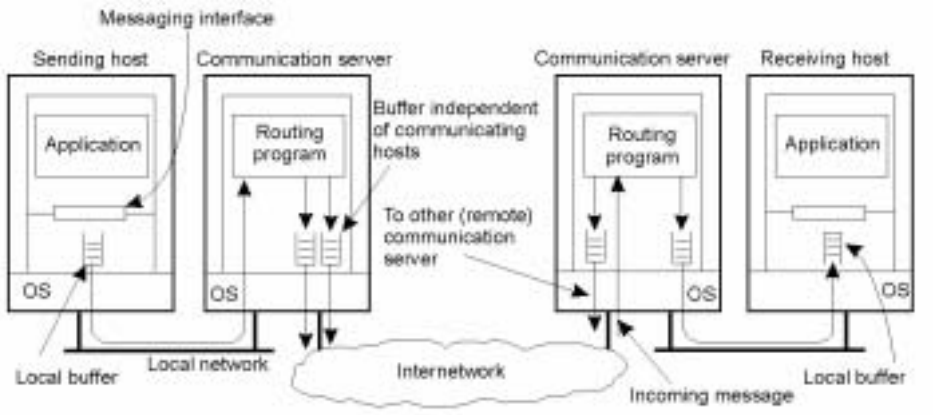
- Copy all necessary files to a directory on the local file system of all clients and the server.
- Change to the directory where the files are located, and run rmiregistry.
- In a separate console window, run the server with a hostname where rmiregistry is running.

  Java LightBulbServer hostname
- In a separate console window (another machine), run the client with a hostname where rmiregistry is running.

  Java LightBulbServer hostname

13

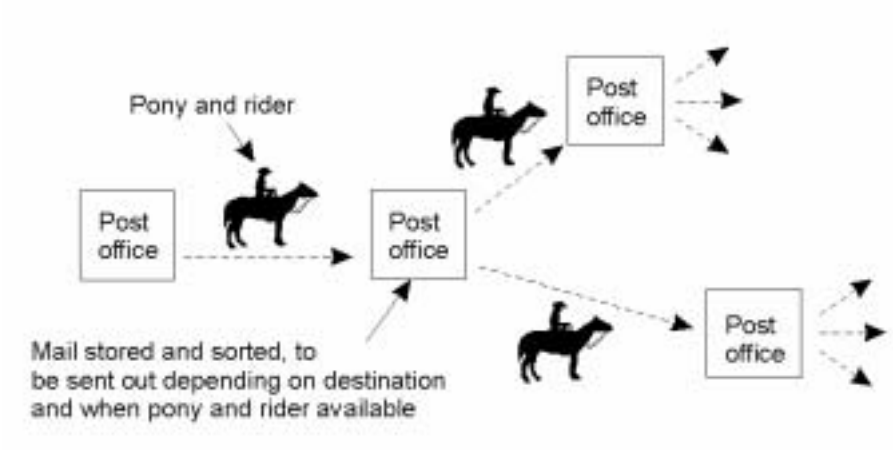
Persistence and Synchronicity in Communication (1)



- General organization of a communication system in which hosts are connected through a network

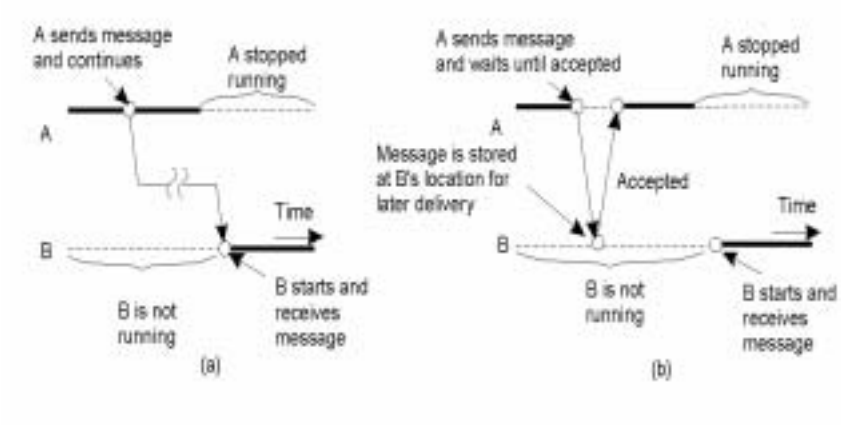
14

Persistence and Synchronicity in Communication (2)



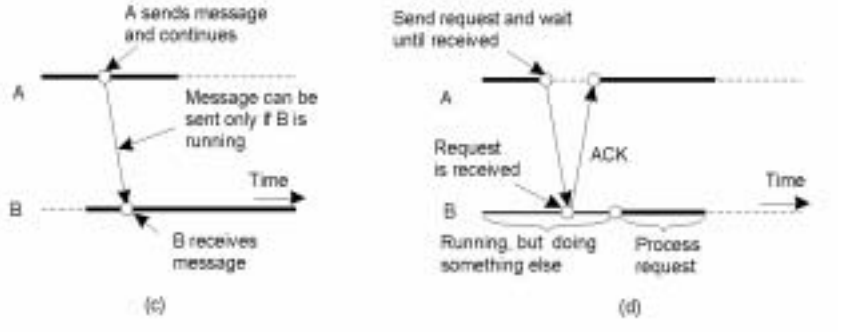
- Persistent communication of letters back in the days of the Pony Express.

Persistence and Synchronicity in Communication (3)



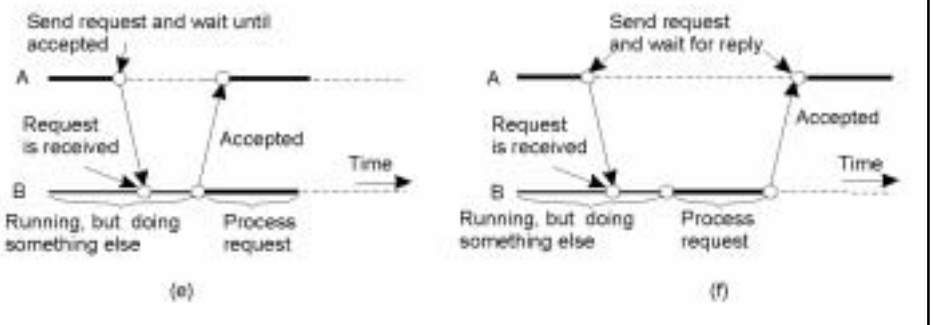
a) Persistent asynchronous communication
b) Persistent synchronous communication

Persistence and Synchronicity in Communication (4)



c) Transient asynchronous communication
d) Receipt-based transient synchronous communication

Persistence and Synchronicity in Communication (5)



e) Delivery-based transient synchronous communication at message delivery

f) Response-based transient synchronous communication

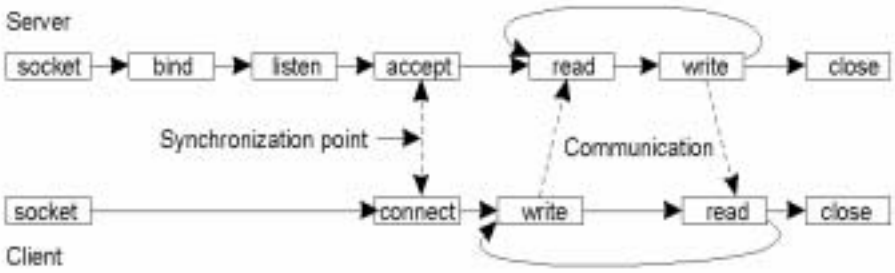
Berkeley Sockets (1)

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.

19

Berkeley Sockets (2)



- Connection-oriented communication pattern using sockets.

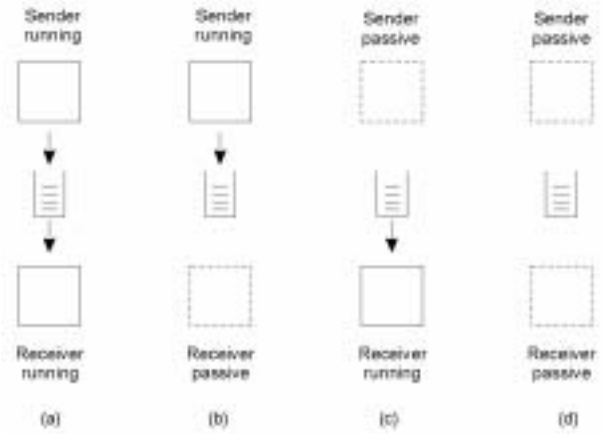
20

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.

Message-Queuing Model (1)



- Four combinations for loosely-coupled communications using queues.

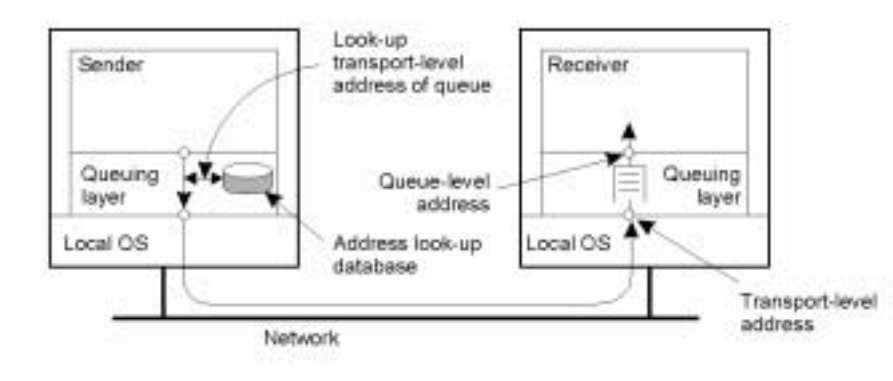
Message-Queuing Model (2)

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.

23

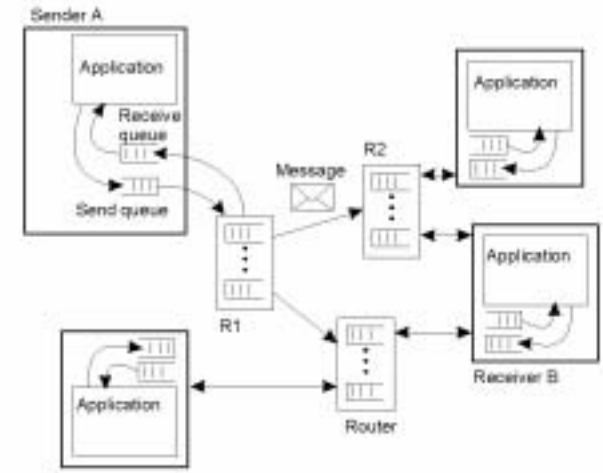
General Architecture of a Message-Queuing System (1)



- The relationship between queue-level addressing and network-level addressing.

24

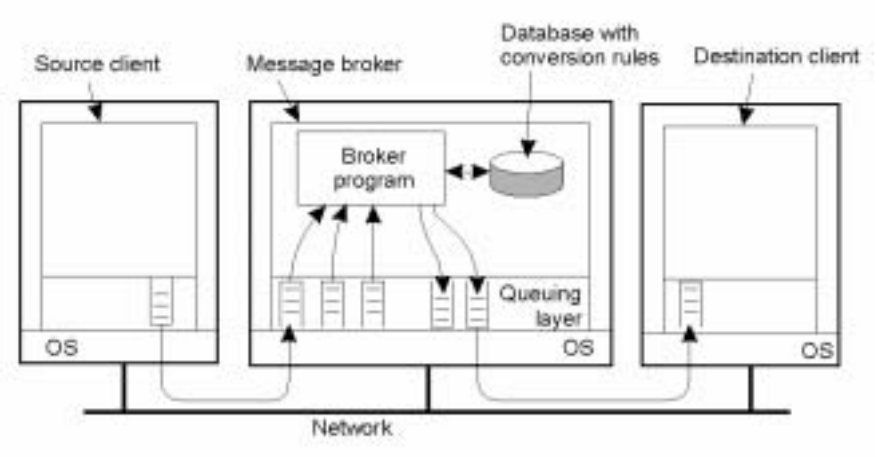
General Architecture of a Message-Queuing System (2)



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

25

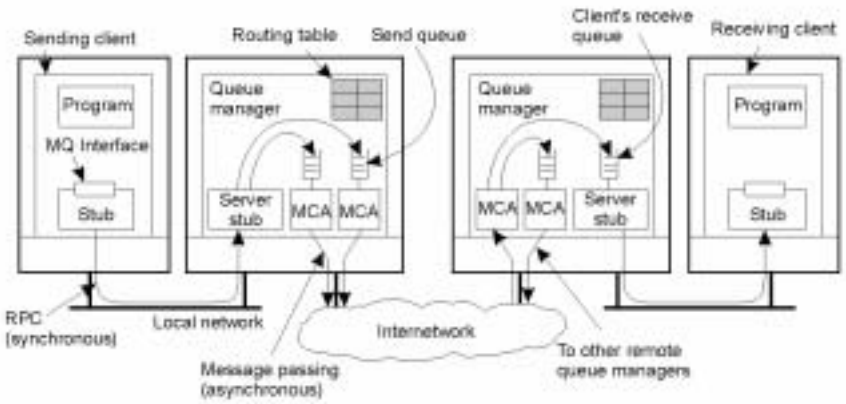
Message Brokers



The general organization of a message broker in a message-queuing system.

26

Example: IBM MQSeries



- General organization of IBM's MQSeries message-queuing system.

27

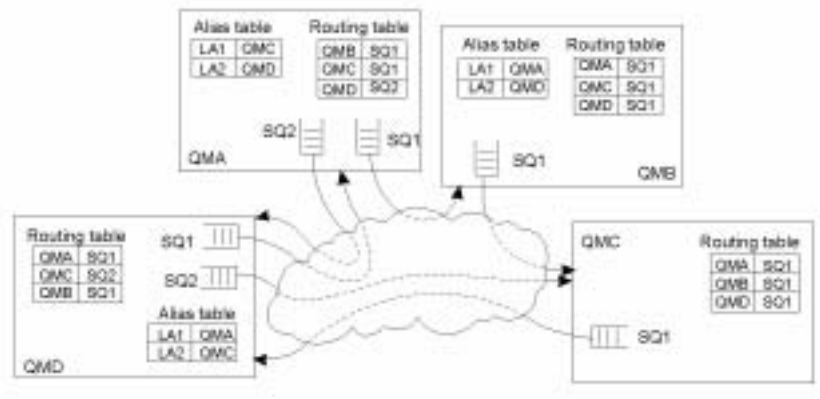
Channels

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

- Some attributes associated with message channel agents.

28

Message Transfer (1)



- The general organization of an MQSeries queuing network using routing tables and aliases.

29

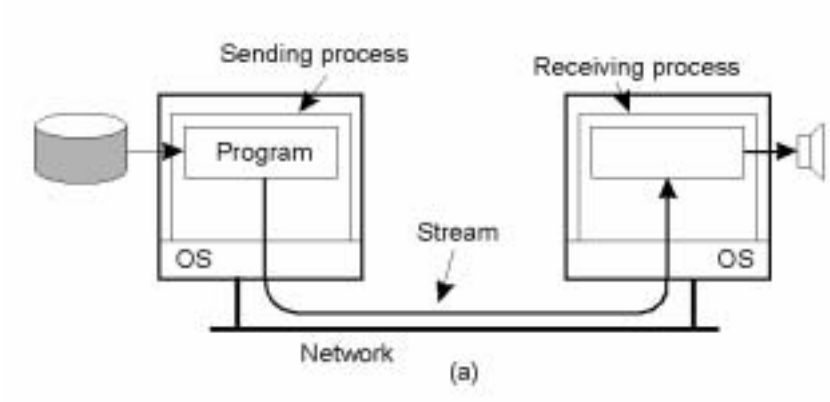
Message Transfer (2)

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

- Primitives available in an IBM MQSeries MQI

30

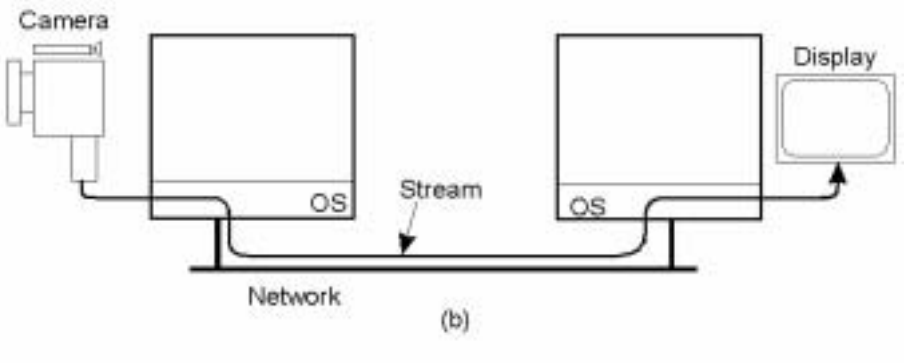
Data Stream (1)



- Setting up a stream between two processes across a network.

31

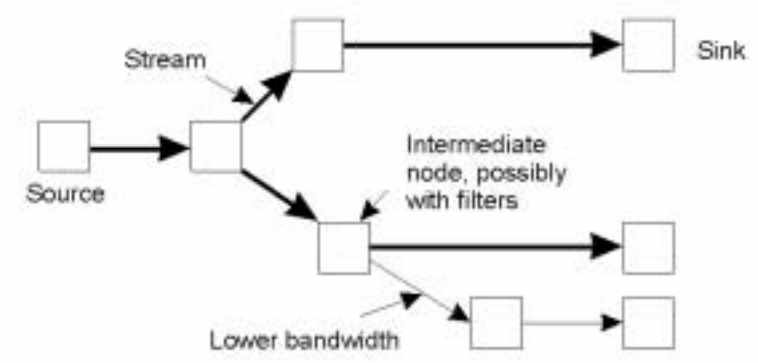
Data Stream (2)



- Setting up a stream directly between two devices.

32

Data Stream (3)



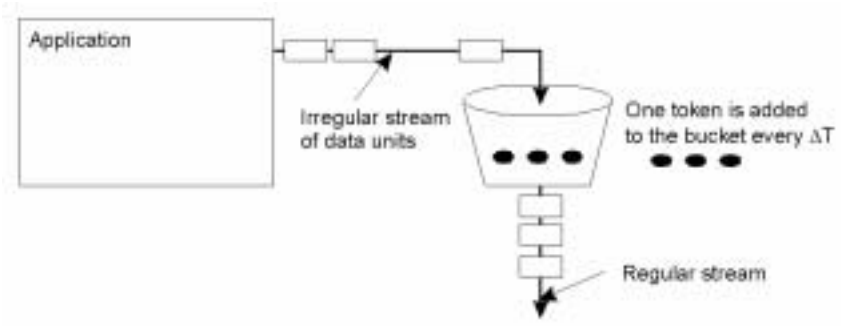
- An example of multicasting a stream to several receivers.

Specifying QoS (1)

Characteristics of the Input	Service Required
◆maximum data unit size (bytes) ◆Token bucket rate (bytes/sec) ◆Toke 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

- A flow specification.

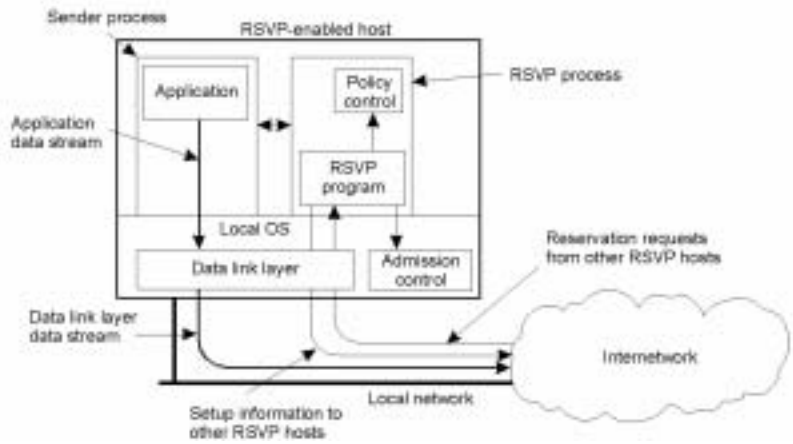
Specifying QoS (2)



- The principle of a token bucket algorithm.

35

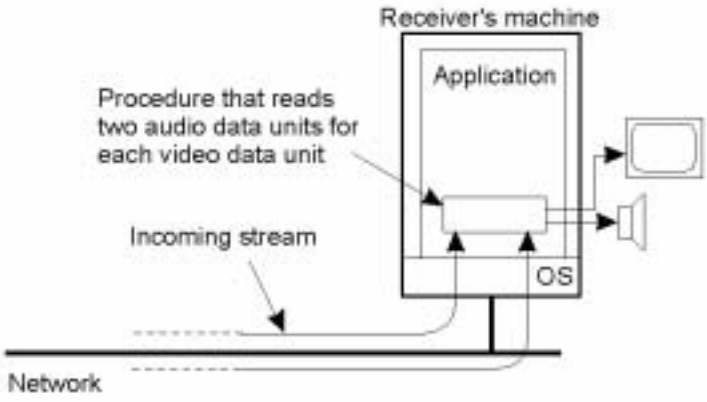
Setting Up a Stream



The basic organization of RSVP for resource reservation in a distributed system.

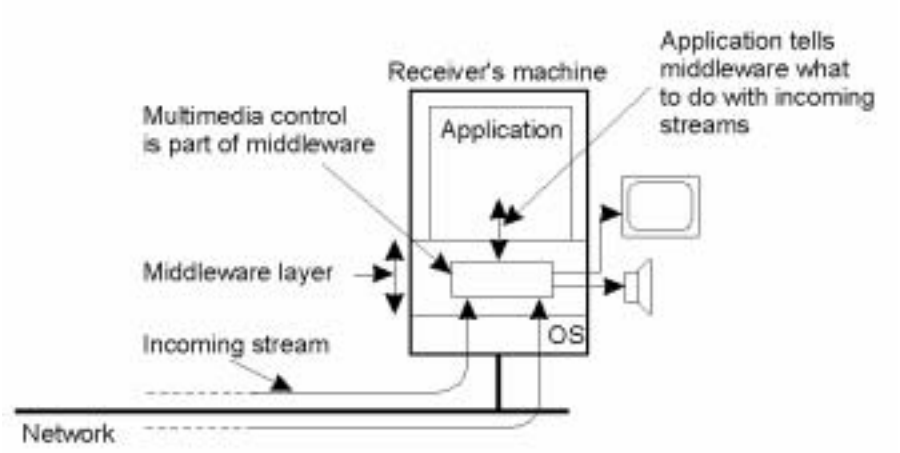
36

Synchronization Mechanisms (1)



- The principle of explicit synchronization on the level data units.

Synchronization Mechanisms (2)



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