

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 & 13, 2003

2

Potential Topics of Term Papers

The topic should be related to distributed systems in general, such as

- Group Communications,
- Peer-to-Peer systems,
- Overlay Networks,
- Grid Computing,
- Object Middleware,
- and others.

Potential Topics of Term Papers

- ◆ Resource Discovery and Management
- ◆ Security and Policy Management
- ◆ Resource Scheduling and Load Balancing
- ◆ Synchronization (e.g., clock synchronization, Election Algorithm, mutual exclusion, etc.)
- ◆ Consistency and replication
- ◆ Reliability and Survivability
- ◆ Performance Evaluation
- ◆ Anonymity
- ◆ Workload characterization
- ◆ Multi-cast Fingerprinting
- ◆ Anonymous authentication in dynamic group communications
- ◆ Data Replication strategies for Grid Computing systems
- ◆ Reputation-based resource scheduling for Grid Computing systems
- ◆ Middleware-based application design and development (e.g., real-time CORBA, fault-tolerant CORBA, etc.)
- ◆ You are welcome to propose your own topic!!!

Your Term Papers

- ◆ Every student is required to finish a term paper.
- ◆ Bi-weekly meeting: 20-30 minutes for each student. If you are interested in it, please make a schedule with me.
- ◆ Deadlines:
 - February 25 (Tuesday, 5:00pm), topic selection due (UG).
 - March 13 (Thursday, 5:00pm), 3-pages proposal (problem definition) due (UG).
 - March 27 (Thursday, 5:00pm), 5-pages solutions and drafted evaluation plan due (G).
 - April 3 (Thursday, 5:00pm), 5-pages summary of the papers you read and important issues you think (U)
 - April. 15 (Thursday 5:00pm), experimental results and improved solution due (G).
 - April. 25 (Thursday 5:00pm), 10-page literature survey (U) and 20-pages full term paper (G) due, including problem definition, solutions, experimental data, conclusion and future works.
- ◆ Paper format: Latex or WORD, IEEE transactions, please refer: <http://www.ieee.org/organizations/pubs/transactions/stylesheets.htm>

Readings for Today's Lecture

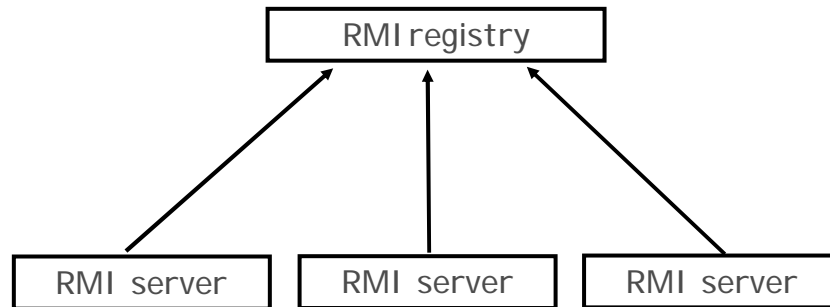
- References
 - Chapter 2 of "Distributed Systems: Principles and Paradigms"

Jave RMI

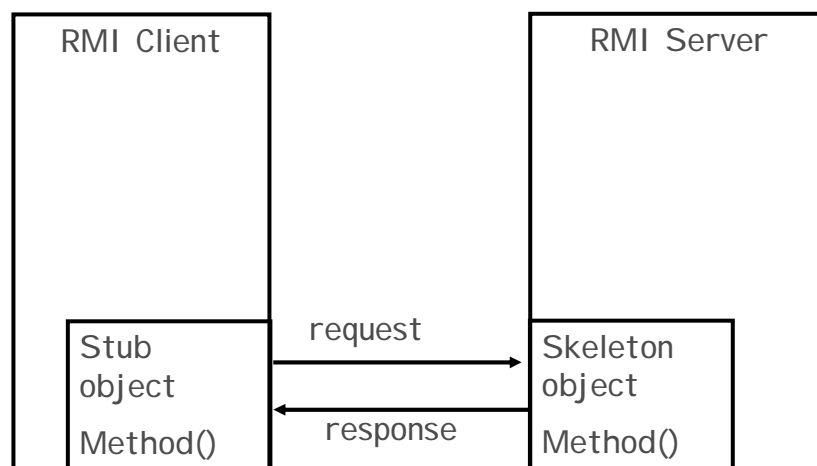
- RMI : A Java technology that allows one JVM to communicate with another JVM and have it execute an object method.
- RPC and RMI
 - RPC supports multiple languages, whereas RMI only support Java
 - RMI deals with objects, but RPC does not support the notion of objects
 - RPC offers procedures (not associated with a particular object)

How RMI works

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



How RMI works



Define a RMI Service Interface

```
Public interface RMI LightBulb 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;
}
```

Implement a RMI Service Interface

```
Public class RMI LightBulbImpl
    extends java.rmi.server.UnicastRemoteObject
    implements RMI LightBulb
{
    Public RMI LightBulbImpl () 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;}

}
```

Create Stub and Skeleton Classes

Rmic RMI LightBulbI mpl

Two files would be produced:

- RMI LightBulbI mpl_Stub.class
- RMI LightBulbI mpl_Skeleton.class

Create a RMI Server

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

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

            String registry = args[0];

            String registration = "rmi://" + registry + "/RMI LightBulb";

            Naming.rebind(registration, bulbService);

        }
    }
}
```

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 + "/RMI LightBulb";

            Remote remoteService = Naming.lookup(registration);

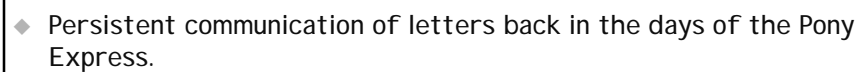
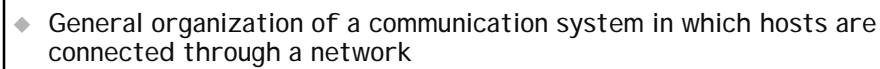
            bulbService.on();
            system.out.println(bulbService.isOn());

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

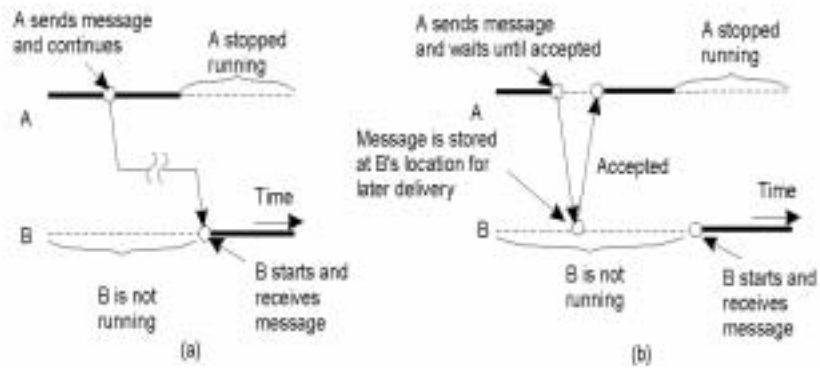
        }
    }
}
```

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 LightBulbClient hostname

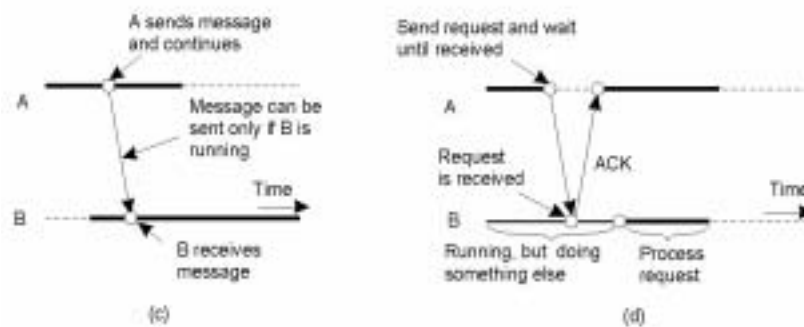


Persistence and Synchronicity in Communication (3)



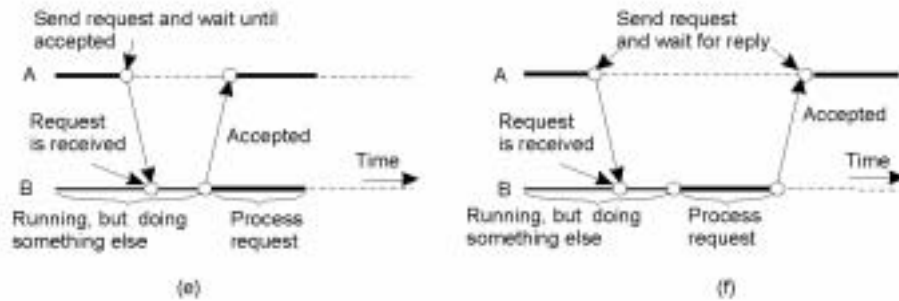
- a) Persistent asynchronous communication
- b) Persistent synchronous communication

Persistence and Synchronicity in Communication (4)



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

Persistence and Synchronicity in Communication (5)



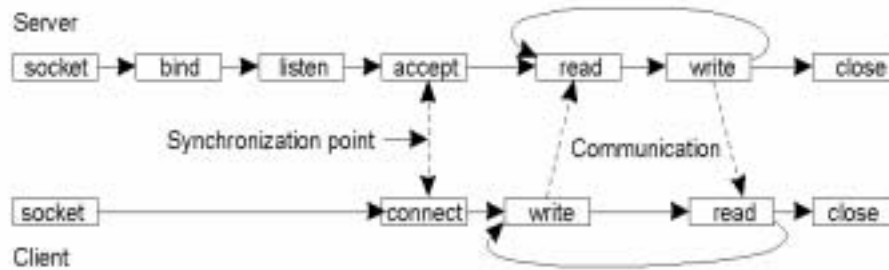
- e) Delivery-based transient synchronous communication at message delivery
- f) Response-based transient synchronous communication

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.

Berkeley Sockets (2)



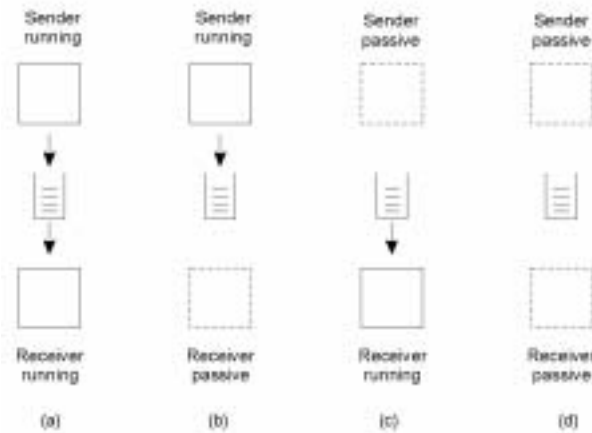
- ◆ Connection-oriented communication pattern using sockets.

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_issend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_rcv	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)



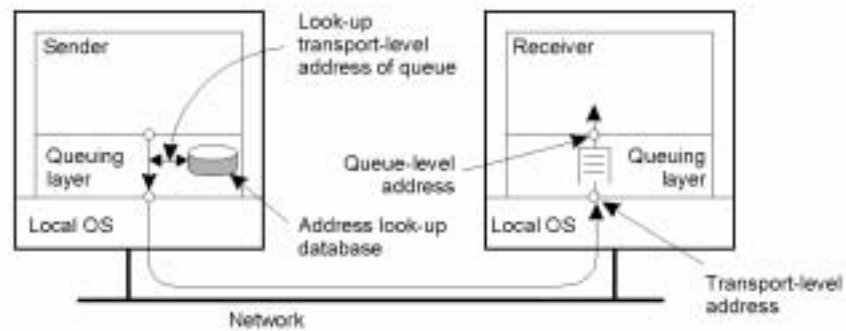
- ◆ Four combinations for loosely-coupled communications using queues.

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.

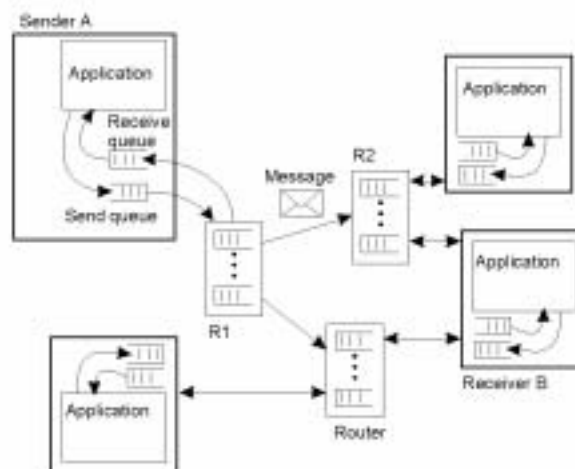
- ◆ Basic interface to a queue in a message-queuing system.

General Architecture of a Message-Queuing System (1)



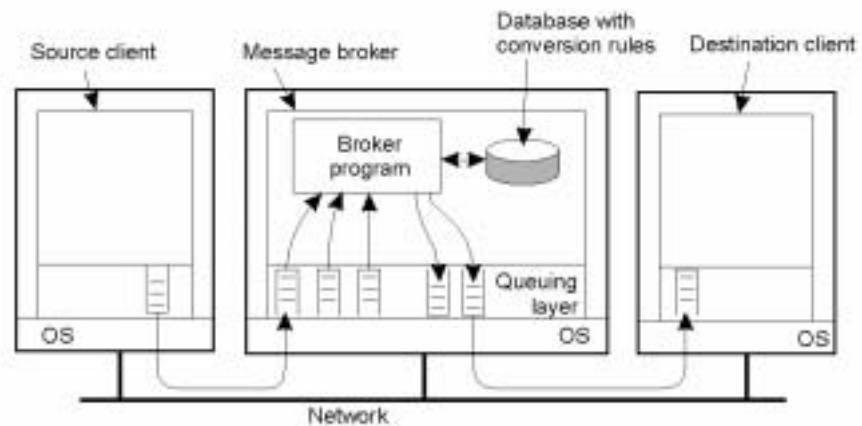
- ◆ The relationship between queue-level addressing and network-level addressing.

General Architecture of a Message-Queuing System (2)



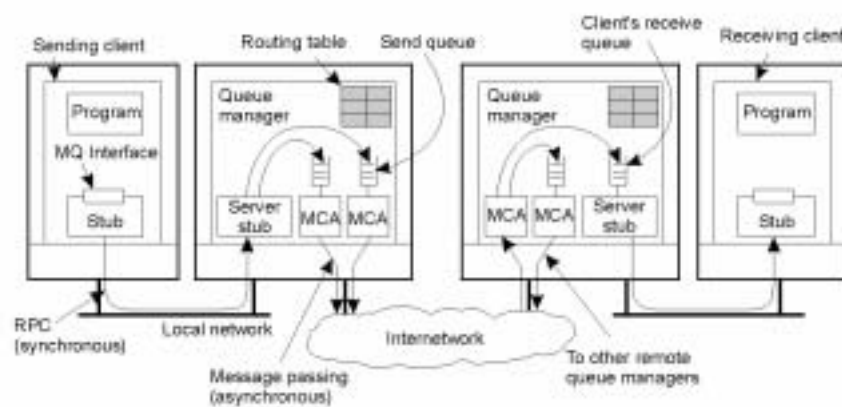
- ◆ The general organization of a message-queuing system with routers.

Message Brokers



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

Example: IBM MQSeries



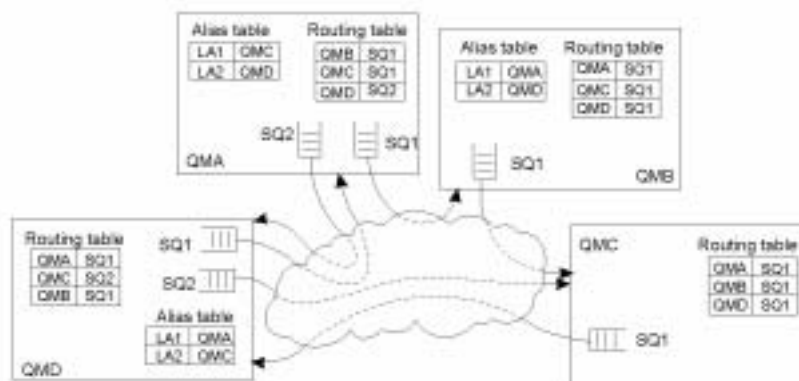
◆ General organization of IBM's MQSeries message-queuing system.

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.

Message Transfer (1)



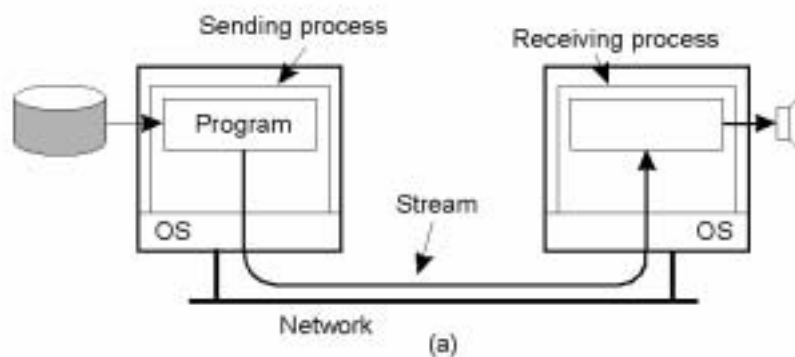
- ◆ The general organization of an MQSeries queuing network using routing tables and aliases.

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

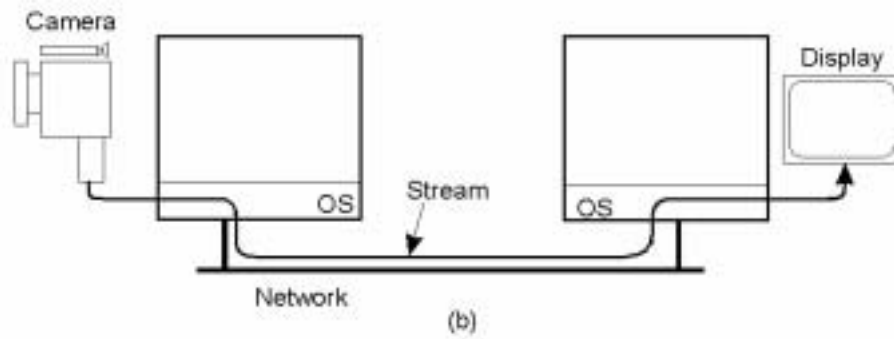
- ◆ Primitives available in an IBM MQSeries MQI

Data Stream (1)



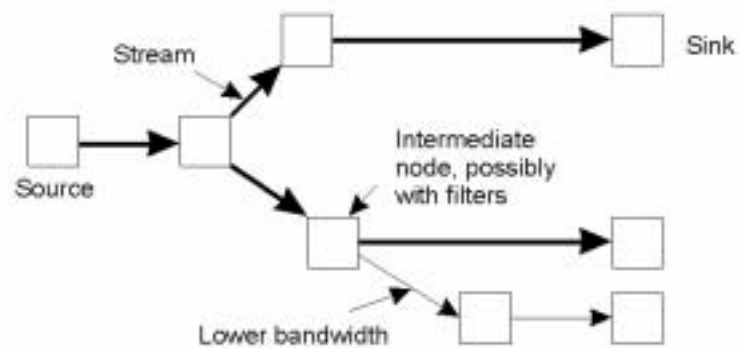
- ◆ Setting up a stream between two processes across a network.

Data Stream (2)



- ◆ Setting up a stream directly between two devices.

Data Stream (3)



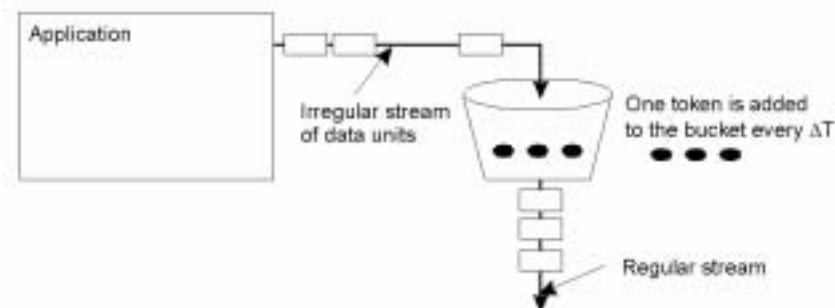
- ◆ An example of multicasting a stream to several receivers.

Specifying QoS (1)

Characteristics of the Input	Service Required
<ul style="list-style-type: none"> ◆ maximum data unit size (bytes) ◆ Token bucket rate (bytes/sec) ◆ Token bucket size (bytes) ◆ Maximum transmission rate (bytes/sec) 	<ul style="list-style-type: none"> ◆ Loss sensitivity (bytes) ◆ Loss interval (μsec) ◆ Burst loss sensitivity (data units) ◆ Minimum delay noticed (μsec) ◆ Maximum delay variation (μsec) ◆ Quality of guarantee

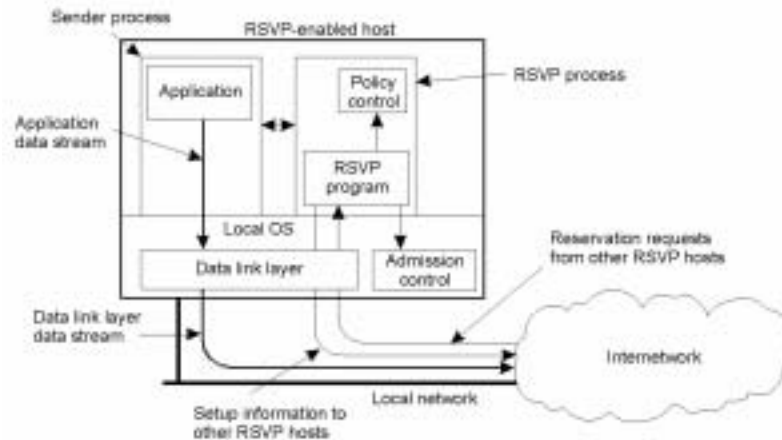
- ◆ A flow specification.

Specifying QoS (2)



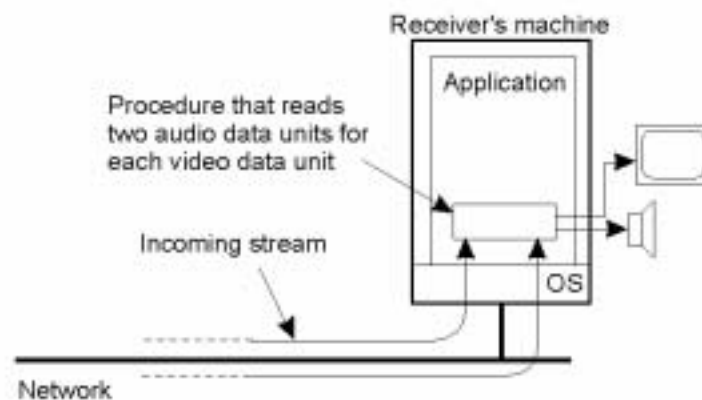
- ◆ The principle of a token bucket algorithm.

Setting Up a Stream



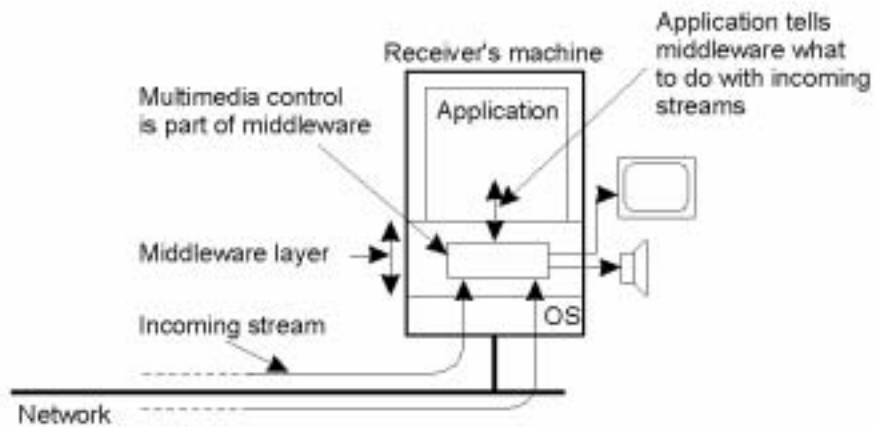
The basic organization of RSVP for resource reservation in a distributed system.

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.