Communication

Chapter 2
Communication

- No shared memory
- Message passing only

- Issues to be resolved at both ends
  - How many volts for a bit 0 / bit 1?
  - How can you tell the beginning/end of a message?
  - How long are numbers, strings, and others?

- Agreements are needed at variety of levels
  - On different issues. => layered model
Layered Protocols

- OSI model is designed to allow open systems to communicate.
- Open system:
  - One that is prepared to communicate with any other open system by using standard rules that govern the format, contents, and meaning of the messages sent and received.
- Protocols:
  - Formalization of the standard rules.
    - Connection-oriented
    - Connectionless
- Protocol suites (protocol stack)
  - A collection of protocols used in a particular system.
Layers, interfaces, and protocols in the OSI model.
Each layer deals with one specific aspect of the communication and provides an interface to the one above it
Layered Protocols (2)

Each layer adds its own header to the message.

Bits that actually appear on the network

A typical message as it appears on the network.
Physical Layer

- Concerned with transmitting the 0s and 1s.
  - How many vols to use for 0 and 1?
  - How many bits per second to transmit?
  - Can both ends transmit at the same time?

- Deals with standardizing the electrical, mechanical, and signaling interfaces.
  - RS-232-C
Data Link Layer

• Physical layer is only concerned with transmitting the 0s and 1s.
• What to do if there are errors?
  – Data link layer is needed to detect and correct them!!
• Data link layer group bits into units(frames) and checks.
  – Putting a special bit pattern at the start and end of the frame,
  – As well as checksums.
• The receiving end computes the checksum from the received frame and compare it with the attached checksum.
• Sender and receiver communicate using their header field.
  – Oops, something wrong, Retransmit!
  – I already did! No you did not!
  – Oh well, here it is again! O.K. I got it!
## Data Link Layer

Discussion between a receiver and a sender in the data link layer.

<table>
<thead>
<tr>
<th>Time</th>
<th>A</th>
<th>B</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Data 0</td>
<td></td>
<td>A sends data message 0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Data 0</td>
<td>B gets 0, sees bad checksum</td>
</tr>
<tr>
<td>2</td>
<td>Data 1</td>
<td>Control 0</td>
<td>A sends data message 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B complains about the checksum</td>
</tr>
<tr>
<td>3</td>
<td>Control 0</td>
<td>Data 1</td>
<td>Both messages arrive correctly</td>
</tr>
<tr>
<td>4</td>
<td>Data 0</td>
<td>Control 1</td>
<td>A retransmits data message 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B says: &quot;I want 0, not 1&quot;</td>
</tr>
<tr>
<td>5</td>
<td>Control 1</td>
<td>Data 0</td>
<td>Both messages arrive correctly</td>
</tr>
<tr>
<td>6</td>
<td>Data 0</td>
<td></td>
<td>A retransmits data message 0 again</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Data 0</td>
<td>B finally gets message 0</td>
</tr>
</tbody>
</table>
Network Layer

- **Routing:**
  - How to find the best path from the sender to the receiver?
- **IP (Internet Protocol)**
  - Connection less, most widely used network layer protocol
- **ATM Virtual Channel Protocol**
  - Connection-oriented
  - Unidirectional connection from a source to destination.
Transport Layer

- What if the whole packet get lost?
- Applications expect reliable communication.

**Sender side:**
- Upon receiving a message, transport layer
  - Break it into small pieces,
  - Assign each one a sequence number,
  - And sends them all.

**Receiver side:**
- Upon receiving the pieces, transport layer
  - Reorders them if necessary,
  - Combine them into original message,
  - And send it up to the application.
Client-Server TCP

(a) Normal operation of TCP.
(b) Transactional TCP.

Using a three-way handshake protocol, much of overhead
Middleware Protocols

An adapted reference model for networked communication.
Conventional Procedure 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

- By pushing the parameters onto the stack
- Indicating a file
- After callee finish, puts the return value in a register, removes the return address, and transfers control back to the caller.
Client and Server Stubs

- Packs the parameters into a message and requests that message to be sent to the server.
- Unpacks the parameter from the message, then performs its work and returns the result to the caller.

Principle of RPC between a client and server program.

마치 local에서 procedure call하듯이
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
Passing Value Parameters (1)

Steps involved in doing remote computation through RPC
Passing Value Parameters (2)

Little endian

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>I</td>
<td>J</td>
</tr>
</tbody>
</table>

Big endian, will find an integer $5 \times 2^{24}$ and a string “JILL”

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>J</td>
<td>I</td>
<td>L</td>
<td>L</td>
</tr>
</tbody>
</table>

Simply invert, now the integer is 5 and the string is “LLIJ”

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>I</td>
<td>J</td>
</tr>
</tbody>
</table>

Problem: Without additional information about what is a string and what is an integer

(a) Original message on the Pentium

(b) The message after receipt on the SPARC

(c) The message after being inverted. The little numbers in boxes indicate the address of each byte
Parameter Specification and Stub Generation

a) A procedure

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

(b) The client and server stub must agree the message format. But, it is not sufficient.

<table>
<thead>
<tr>
<th>foobar's local variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x</code></td>
</tr>
<tr>
<td><code>y</code></td>
</tr>
<tr>
<td><code>5</code></td>
</tr>
<tr>
<td><code>z[0]</code></td>
</tr>
<tr>
<td><code>z[1]</code></td>
</tr>
<tr>
<td><code>z[2]</code></td>
</tr>
<tr>
<td><code>z[3]</code></td>
</tr>
<tr>
<td><code>z[4]</code></td>
</tr>
</tbody>
</table>
Extended RPC Models

What if the caller and callee are on the same machine?

- Local IPC mechanism would be a lot more efficient.
- However, the mechanism would be different!

Door

- Same mechanism (RPC) on the same machine.
- A generic name for a procedure in the address space of a server process that can be called by processes collocated with the server.
- Originally designed for Spring OS.
Doors

Server
• Registers the door.
  – An ID is returned to be used to give the door a symbolic name.
  – A file name is used in Solaris.

Client
• Calls the door by means of the system call door_call.

OS
• Does upcall to the server process that registered the door.

Server
• Processes the request and returns through the system call door_return.
The principle of using doors as IPC mechanism.

IPC: Inter Process Communication, 기계 하나에 여러 process가 있는 경우, RPC를 같은 기계에서 쓴다.
Asynchronous RPC

In RPC, client blocks until a reply is returned.
- Not necessary if no result is return.
  - Transferring money from one to another account.
  - Adding entries to a database.
  - Starting remote services.

Asynchronous RPC
- Server immediately sends a reply as soon as the request is received.
- The reply acts as an acknowledgement that the server is going to process the RPC.
- Client continues without blocking.
Asynchronous RPC (1)

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

Sometimes, the client blocking is waste of time.

• The client wants to do something while waiting for the reply.

Client
• Calls the server and sends the request.
• After receiving the acknowledgement, continues processing.

Server
• Calls the client back to return the reply.

One-way RPC
• Does it necessary for the client to wait for the acknowledgement?
• Sometimes, it is not!
A client and server interacting through two asynchronous RPCs
DCE
(Distributed Computing Environment)

- A middleware system
  - It is designed to execute as a layer of abstraction between existing OS and distributed applications.
- Widespread acceptance
  - Starting with UNIX, ported to VMS, Windows NT, etc.
- Programming model is Client-server.
- Services
  - Distributed file service
  - Directory service
  - Security service
  - Distributed time service
Goal of DCE RPC

• Enable a client to access a remote service.
• Existing code can run in a distributed environment with few, if any, changes.
• Hide the details from the clients, and the servers.
  – Locate the server automatically,
  – Set up the communication (binding),
  – Handle the message transfer (fragmenting, reassemblying)
  – Automatically handle data type conversion.
    (marshalling)
• Clients and servers are highly independent.
  – Different language, architecture, network protocols…
Writing a Client and a Server

RPC는 IDL을 통해 표현한다
(Interface Definition Language)

The steps in writing a client and a server in DCE RPC.
Binding a Client to a Server

Server location is done in two steps:
1. Locate the server’s machine
2. Locate the server on that machine

It is necessary that the server be registered and prepared to accept incoming calls

Client-to-server binding in DCE.
RPC semantics

At-most-once operation
- No call is carried out more than once.
- If a server crashes during an RPC and then recovers, the client does not repeat the operation.

Idempotent operation
- Can be repeated multiple times without harm.
  - Reading a specified block.

Exactly-Once
Remote Object Invocation

Objects

- Good at hiding its internals from the outside world by means of a well-defined interface.
- Encapsulates data, called state, and the operations, the methods.

Can we utilize the RPC mechanism to invoke remote objects?

- Remote Method Invocation
Remote Object Invocation

Objects
• Good at hiding its internals from the outside world by means of a well-defined interface.
• Encapsulates data, called state, and the operations, the methods.

Can we utilize the RPC mechanism to invoke remote objects?
• Remote Method Invocation

Distributed Objects
• Separation between the interface and the object.
Distributed Objects

Marshaling: 두 machine이 다른 기종이라면, 양쪽이 다 이해할 수 있는 형태로 바꿔 전송해 줄 수 있어야 한다.
Unmarshaling: My data 형태로 바꾸기

Common organization of a remote object with client-side proxy.
Compile-time versus Runtime Objects

Compile-time Objects
- Related to language-level objects
- Defined as the instance of a class.
- A class is a description of an abstract type in terms of a module with data elements and operations on that data.
- Drawback: dependency on a particular programming language.

Runtime Objects
- Construct distributed objects during runtime independent of the programming language in which the application is written.
- Objects are solely defined in terms of the interfaces they implement.
- An implementation of an interface can then be registered at an adapter.
- Object adapter
- The adapter will take care of the invocation by calling the object.
Persistent and Transient Objects

**Persistent Object**
- Object that continues to exist even if it is currently not contained in the address space of a server process.
- Usually stored in secondary storage when the server exits.

**Transient Object**
- Exists only as long as the server that manages the object.
- As soon as the server exits, the object ceases to exist.

**Implicit vs. explicit binding**
Binding a Client to an Object

(a) Example with implicit binding using only global references

Distr_object* obj_ref; // Declare a systemwide object reference
obj_ref = ...; // Initialize the reference to a distributed object
obj_ref-> do_something(); // Implicitly bind and invoke a method

(b) Example with explicit binding using global and local references

Distr_object objPref; // Declare a systemwide object reference
Local_object* obj_ptr; // Declare a pointer to local objects
obj_ref = ...; // Initialize the reference to a distributed object
obj_ptr = bind(obj_ref); // Explicitly bind and obtain a pointer to the local proxy
obj_ptr -> do_something(); // Invoke a method on the local proxy

a) (a) Example with implicit binding using only global references
b) (b) Example with explicit binding using global and local references
Implementation of Object Reference

How DCE does it?

- **RPC endpoint = (IP, port, RPC_id)**
- Endpoint change can be handled by employing a daemon to listen to a well-known endpoint.
- **Object endpoint = (IP, port, obj_id)**

How to handle IP change?

- **Location server** to the rescue.
- **Server** will take care of the invocation by calling the object.

What if different protocols are used?

- Object reference can have such information.
- **Implementation handle** (agent?)
Static vs. Dynamic RMI

**Static invocation**
- Specify the object’s interface in an IDL.
- Interfaces of an object is known.
- `fobject.append(int)`

**Dynamic invocation**
- It is sometimes convenient to compose a method invocation at runtime.
- `Invoke(fobject, id(append), int)`
- 예) web browser에서 click했을때 수행되는 application
Parameter Passing

Easier than RPC

- Systemwide object reference
- If only distributed objects, call by reference.
- However, it could be inefficient.
  - Especially for small objects.

Different ways for local/remote objects

- Remote object reference
  - Is copied and passed as a value parameter.
  - The object is passed by reference.
- Local object reference
  - The object is copied as a whole and passed along with the invocation.
  - The object is passed by value.
Parameter Passing

The situation when passing an object by reference or by value.
To create an object, a client will have to issue a request at the server. Each class of dynamic objects has an associated create procedure.

a) Distributed dynamic objects in DCE.

b) Distributed named objects
Java RMI

In DCE, distributed objects have been added as a refinement.
In Java, distributed objects have been integrated into the language.

• Remote objects are treated like local objects as much as possible

Some differences

• Cloning is different.
  – Cloning a remote object is only done at server, without cloning the proxies as well.

• Blocking is different. (synchronized method)
  – A process can be blocked inside an object (monitor).
  – Where should it block? At the client or at the server?
  – At the proxy.
  – Distributed locking should be used to synchronize processes operating on different proxies.
Java Remote Object Invocation

Any primitive or object type can be passed as a parameter to an RMI.

- As long as they can be marshaled (serializable).
- Most can be except platform dependent objects,
  - File descriptors, sockets, etc..
- Even proxies are serializable.
  - They run on the same JVM, remember?
2.4 Message Oriented Communication

- 어느 한쪽이 준비가 되지 않았을 때에는 ??
  - 다른 종류의 communication paradigm 필요

- Persistence
  - A message submitted for transmission is stored as long as it takes to deliver it to the receiver

- Transient
  - A message is stored as long as the sending and receiving application are executing
  - All transport-level communication services typically offer only transient communication.

- Asynchronous

- Synchronous
Persistence communication of letters
(Pony Express)

Mail stored and sorted, to be sent out depending on destination and when pony and rider available
Persistence and Synchronicity

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

Server 있는 경우 persistence, 없다면 local buffer에 있다가 종료되면 사라짐
Combination of Persistency and synchrony

1. Persistent asynchronous
   - A message is either persistently stored in a buffer at the local host, or at the first communication server.
   - Electronic mail

2. Persistent synchronous
   - Messages can be persistently stored only at the receiving host.
   - A sender is blocked until its message is stored at the receiver’s buffer.
   - The receiving application is not necessarily executing.
Persistence and Synchronicity in Communication (3)

a) Persistent asynchronous communication

b) Persistent synchronous communication
(blocked until stored/delivered)
3. Transient asynchronous communication
   • A message is temporarily stored in a local buffer.
   • The sender continues without blocking.
   • The message is sent to the destination.
   • If the receiving party is not ready, it fails.
   • UDP

4. Receipt-based Transient synchronous communication
   • The sender blocks until message is stored in a local buffer at the receiver.
c. Transient asynchronous communication
   (store in the sending buffer and continue) UDP, asynchronous RPC

   d. Receipt-based transient synchronous communication
      (blocked until message is stored at the local buffer of the receiver)
Combination of Persistency and synchrony

5. Delivery-based Transient synchronous communication
   - The sender blocks until message is delivered to the receiver for further processing.
   - Similar to Asynchronous RPC.

6. Response-based Transient synchronous communication
   - The sender blocks until it receives a reply message.
   - RPC and RMI.
e) Delivery-based transient synchronous communication at message delivery (blocked until delivered)

f) Response-based transient synchronous communication (blocked until delivered, processed and replied) RPC, RMI
Message-Oriented Transient Communication

Berkeley Sockets
• Communication endpoints

X/Open Transport Interface (XTI)
• Formally, Transport Layer Interface (TLI)

Message Passing Interface (MPI)
• Provides High level of abstraction.
• For Parallel Programming
Message Oriented Transient Communication:

Berkeley Sockets (1)

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

Socket primitives for TCP/IP.
Berkeley Sockets (2)

Connection-oriented communication pattern using sockets.
The Message-Passing Interface (MPI)

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

Some of the most intuitive message-passing primitives of MPI.
Message-Oriented Persistent Communication

Message-queueing systems, or Message-Oriented Middleware (MOM)
- Persistent Asynchronous Communication.
- Usually deals messages with long latency.

Internet e-mail
- Most well-known, traditional system.
Both the sender and receiver are executing during the entire transmission of a message.

Only the sender is executing

Combination of a passive sender and an executing receiver

Sender and receiver are passive

Sender running

Receiver running

(a)

Sender running

Receiver passive

(b)

Sender passive

Receiver running

(c)

Sender passive

Receiver passive

(d)

Four combinations for loosely-coupled communications using queues.

A sender is generally given only the guarantee that the message will be inserted in the recipient’s queue eventually.
Basic interface to a queue

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put</td>
<td>Append a message to a specified queue</td>
</tr>
<tr>
<td>Get</td>
<td>Block until the specified queue is nonempty, and remove the first message</td>
</tr>
<tr>
<td>Poll</td>
<td>Check a specified queue for messages, and remove the first. Never block.</td>
</tr>
<tr>
<td>Notify</td>
<td>Install a handler to be called when a message is put into the specified queue.</td>
</tr>
</tbody>
</table>

Callback : handler, which is automatically invoked whenever a message is put into the queue.
Often implemented by means of a daemon.
Message-Queuing System

- Messages can be put only into queues that are local to the sender.
- Messages can be read only from local queues.
- Message-queuing system is responsible for transferring messages from the sender’s queue to the receiver’s.

Mapping between queue names and network locations.
- Like DNS for e-mail

Queue managers
- Manage queues,
- Relay messages.

Relays
- Network Management
- Scalability
- Multicasting
- gateways
General Architecture of a Message-Queuing System (1)

The relationship between queue-level addressing and network-level addressing.
The general organization of a message-queuing system with routers.
Message Brokers

How to integrate existing and new applications into a single, coherent distributed information system?

• Messages should be understood by all applications.
  – Standard message format?
  – Impractical, since applications have very little in common.

• Message Broker is used to make it easy to convert message formats.

Message broker

• May be a simple message re-formatter.
• Application level gateways.
  – Might lose some information in conversion.
• Need database of conversion rules.
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

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

Some attributes associated with message channel agents.
The general organization of an MQSeries queuing network using routing tables and aliases.
Message Transfer (2)

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQopen</td>
<td>Open a (possibly remote) queue</td>
</tr>
<tr>
<td>MQclose</td>
<td>Close a queue</td>
</tr>
<tr>
<td>MQput</td>
<td>Put a message into an opened queue</td>
</tr>
<tr>
<td>MQget</td>
<td>Get a message from a (local) queue</td>
</tr>
</tbody>
</table>

Primitives available in an IBM MQSeries MQI
2.5 Stream Oriented Communication

So far,
- Independent communication
- No timing requirement.

• Time dependent media support
  - Continuous media such as audio/video

• Data stream
  - Asynchronous transmission mode
  - Synchronous transmission mode
    • Maximum end-to-end delay
  - Isochronous transmission mode
    • Maximum as well as minimum end-to-end delay
Data Stream

A virtual connection between a source and a sink.

• The source or sync could be a process or a device.
• The source or sync could be single or multiple.
  – Multicast stream

Simple stream
  – Single sequence of data

Complex stream
  – Consists of several related simple streams.
  – Audio/video/subtitles…
Data Stream (1)

Setting up a stream between two processes across a network.

Sending process reading an audio file from disk, and sending it, byte for byte, through a network.

The receiving process retching the bytes as they come in, and passing then to the local audio device.
Data Stream (2)

Setting up a stream directly between two devices.

Setting up a direct connection between source and sink, be directly forwarded to a display device.
An example of multicasting a stream to several receivers.

multiparty communication: attaching multiple sinks to a stream, the data stream is multicast to several receivers
Streams and Quality of Service

A virtual connection between a source and a sink.

- The source or sync could be a process or a device.
- The source or sync could be single or multiple.
  - Multicast stream

Simple stream
  - Single sequence of data

Complex stream
  - Consists of several related simple streams.
  - Audio/video/subtitles...
Specifying Quality of Service

By flow specification containing bandwidth requirements

• Token bucket algorithm
  – Specifies how the stream will shape its network traffic.
  – When the bucket is full, tokens will simply be dropped.

• Service requirements

• What if the application has no idea?
  – Provide some default values,
    • Such as (audio, high/medium/low, …)
    • (video, mpeg1/mpeg2/mpeg4, …)
### Specifying QoS (1)

<table>
<thead>
<tr>
<th>Characteristics of the Input</th>
<th>Service Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>• maximum data unit size (bytes)</td>
<td>• Loss sensitivity (bytes)</td>
</tr>
<tr>
<td>• Token bucket rate (bytes/sec)</td>
<td>• Loss interval (µsec)</td>
</tr>
<tr>
<td>• Token bucket size (bytes)</td>
<td>• Burst loss sensitivity (data units)</td>
</tr>
<tr>
<td>• Maximum transmission rate (bytes/sec)</td>
<td>• Minimum delay noticed (µsec)</td>
</tr>
<tr>
<td></td>
<td>• Maximum delay variation (µsec)</td>
</tr>
<tr>
<td></td>
<td>• Quality of guarantee</td>
</tr>
</tbody>
</table>

A flow specification.
Specifying QoS (2)

The principle of a token bucket algorithm.

Each time the application wants to pass a data unit of size $N$ to the network, it will have to remove enough tokens from the bucket that jointly represent at least $N$ bytes.

Data is passed to the network at a relatively constant rate determined by the rate of generating tokens.
Setting up a Stream

Resources needed:

- **Bandwidth**
  - Ensuring data units are properly scheduled for transmission.
- **Buffers**
  - Should be allocated in routers and OSes.
- **Processing capacity**
  - Data units should be processed in time.
  - Scheduler, encoders/decoders, filters, etc.

No single best model exists

- For specifying QoS, describing resource and translating QoS parameters to resource usage.
Resource Reservation Protocol
Stream Synchronization

Important issue in multimedia systems

• Different streams should be synchronized.
  – Discrete data stream + continuous data stream
  – Continuous data stream + continuous data stream
• A difference of more than 20 microseconds is not allowed for stereo effect.
• Between audio and video, 40-80 miliseconds can be tolerated.
Synchronization Mechanisms (1)

The principle of explicit synchronization on the level data units.

A process that simply executes read and write operation on several simple streams, ensuring that those operations adhere to specific timing and synchronization constraints.
Synchronization Mechanisms (2)

Middleware offers a collection of interfaces for controlling audio and video streams.

Application tells middleware what to do with incoming streams.

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