Distributed Operating Systems
Communication (II)

Ewa Niewiadomska-Szynkiewicz

ens@ia.pw.edu.pl

Institute of Control and Computation Engineering
Warsaw University of Technology

Communication (II)

1. Message-oriented Communication
2. Stream-oriented Communication

Message-oriented Communication – Introduction

- √ When it cannot be assumed that the receiving side is executing at the time a request (in RPC or RMI) is issued, alternative communication services are needed.
- √ The inherent synchronous nature of RPCs and RMIs, by which a client is blocked until its request has been processed, sometimes has to be replaced.
- √ Message-oriented communication is proposed.

The message-passing paradigm is widely used approach for programming parallel machines, especially those with distributed memory.

Message-passing Paradigm – Attributes

Two key attributes characterizing the message-passing paradigm:

- √ it assumes a partitioned address space,
- √ it supports only explicit parallelization.

The logical view of a machine supporting the message-passing paradigm consists of **P** processes, each with its own exclusive address space.

Communication System (1)

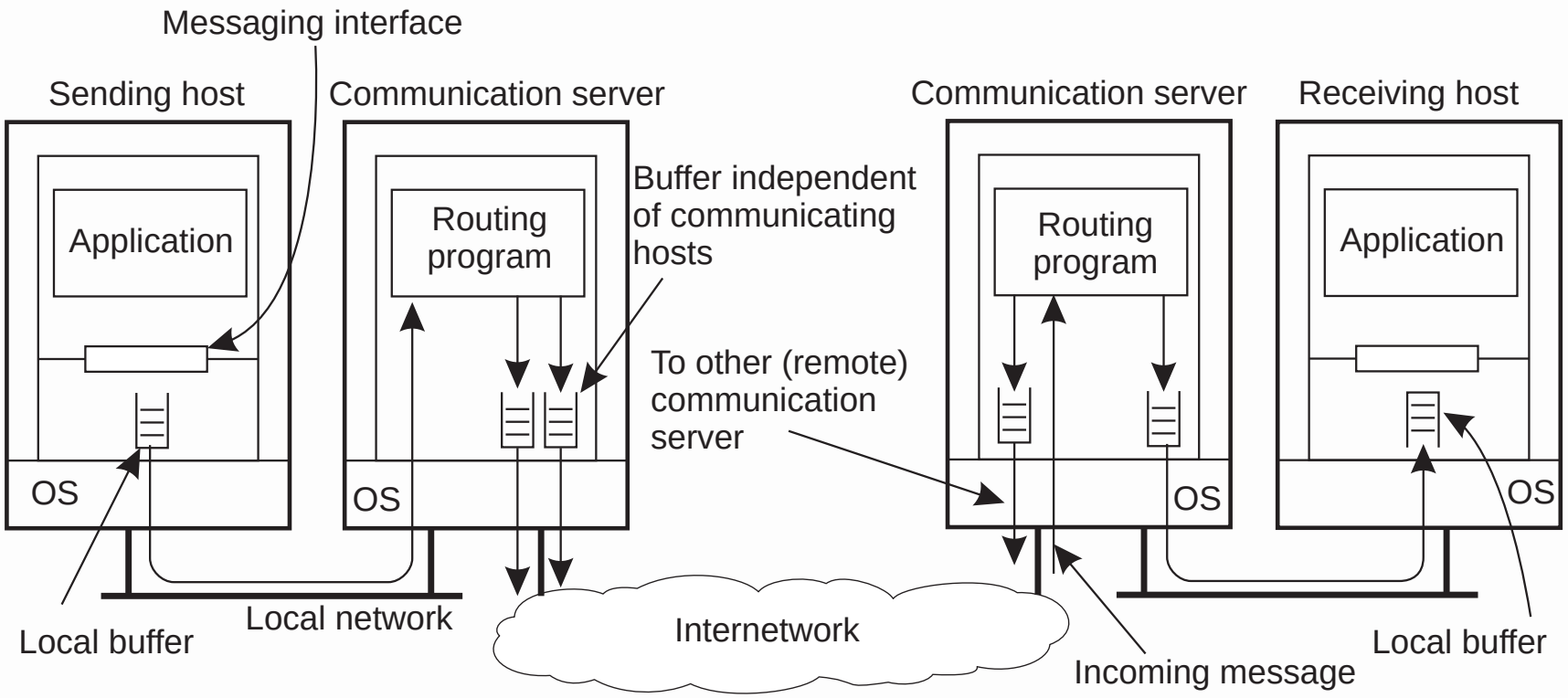
Assumption – communication system organized as follows:

- √ applications are executed on hosts,
- √ each host connected to one communication server,
- √ buffers may be placed either on hosts or in the communication servers of the underlying network,
- √ example: an e-mail system.

Types of communication:

- √ **persistent** vs **transient** communication,
- √ **synchronous** vs **asynchronous** communication.

Communication System (2)



Organization of a communication system, hosts connected through a network

- √ queued messages sent among processes,
- √ sender not stopped in waiting for immediate reply,
- √ fault tolerance often ensured by middleware.

Persistence vs. Transient Communication

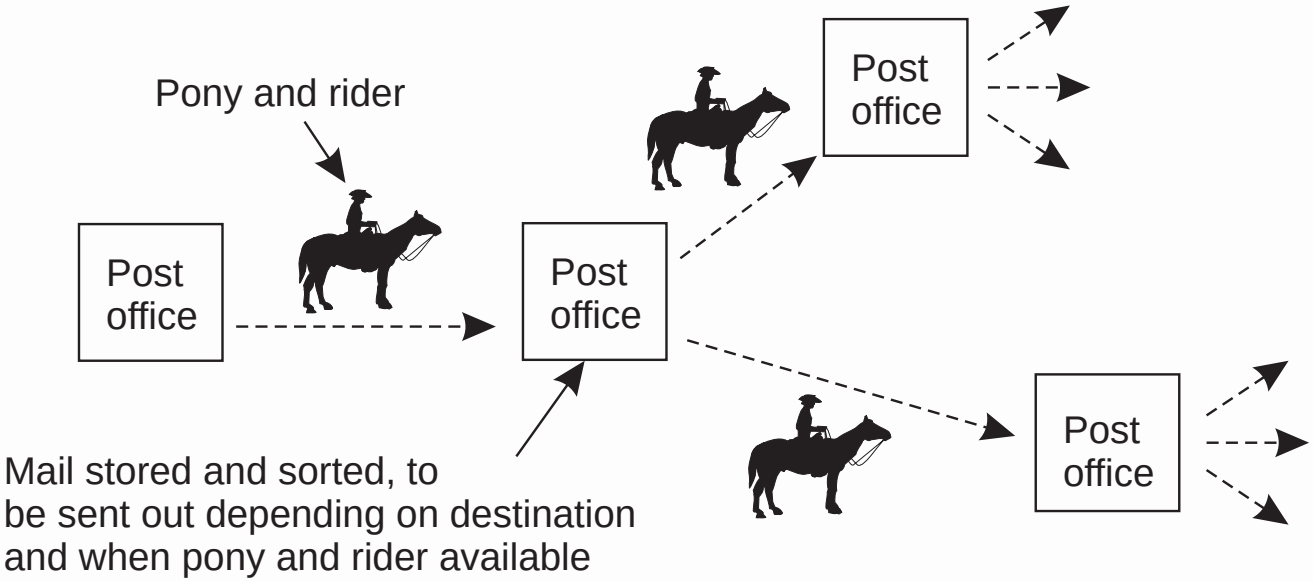
Persistent communication

A message is stored at a communication server as long as it takes to deliver it at the receiver.

Transient communication

A message is discarded by a communication server as soon as it cannot be delivered at the next server or at the receiver.

Persistence Communication – Example



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

Synchronous vs. Asynchronous Communication

Asynchronous communication

The sender continues immediately after it has submitted its message for transmission.

Synchronous communication

The sender is blocked until its message is stored in a local buffer at the receiving host or actually delivered to the receiver.

Synchronous Communication – Example

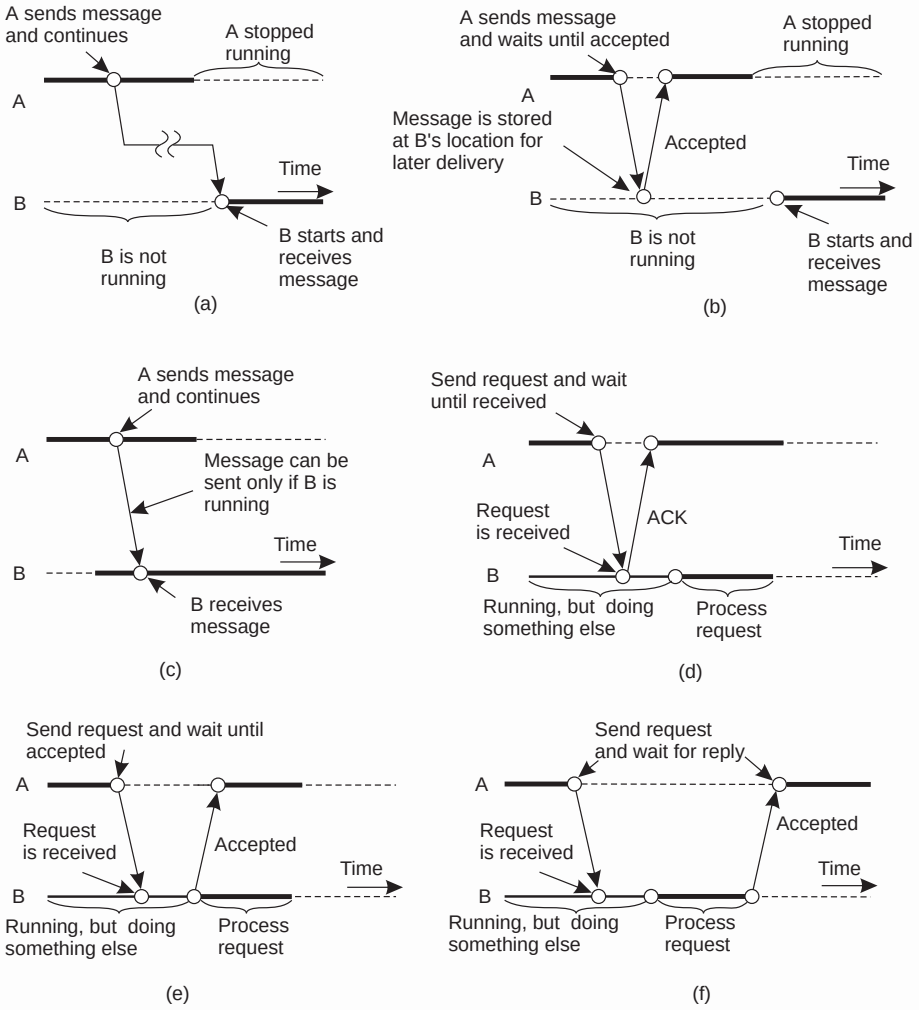
Client/server computing generally based on a model of **synchronous communication**:

- √ client and server to be active at the time of communication,
- √ client issues request and blocks until reply received,
- √ server essentially waits only for incoming requests and subsequently processes them.

Drawbacks of synchronous communication:

- √ client cannot do any other work while waiting for reply,
- √ failures to be dealt with immediately (the client is waiting),
- √ in many cases the model simply is not appropriate (mail, news).

Different Forms of Communication



Different forms of communication:

a. persistent asynchronous,
b. persistent synchronous,
c. transient asynchronous,
d. receipt-based transient synchronous,
e. delivery-based transient synchronous,
f. response-based transient synchronous,

Message-Oriented Transient Communication

- √ **Berkeley Sockets** – socket interface introduced in Berkeley UNIX,
- √ another transport layer interface: **XTI**, **X/Open Transport Interface**, formerly called the Transport Layer Interface (TLI), developed by AT&T
- √ **Message-Passing Interface** (MPI) – standard for message passing

Sockets and XTI are very similar in their model of network programming, but differ in their set of primitives.

Berkeley Sockets

Socket

Communication endpoint to which an application write data that are to be sent over the underlying network and from which incoming data can be read.

A socket forms an abstraction over the actual communication endpoint that is used by the local operating system for a specific transport layer.

Berkeley Sockets – Communication Types (1)

Connection-Oriented Communication

- √ When devices communicate, they perform handshaking to set up an end-to-end connection.
- √ The handshaking process may be as simple as synchronization such as in the transport layer protocol TCP, or as complex as negotiating communications parameters as with a modem.

Connection-Oriented systems can only work in bi-directional communications environments. To negotiate a connection, both sides must be able to communicate with each other.

Berkeley Sockets – Communication Types (2)

Connectionless Communication

- √ No effort is made to set up a dedicated end-to-end connection.
- √ Connectionless communication is usually achieved by transmitting information in one direction (source to destination) without checking to see if the destination is still there, or if it is prepared to receive the information.

Connectionless communication works fine when there is little interference, and plenty of speed available.
In environments where there is difficulty transmitting to the destination, information may have to be re-transmitted several times before the complete message is received.

Socket Primitives

When calling the socket **primitive**, the caller creates a new communication endpoint for a specific transport protocol.
Communication point creation – the local operating system reserves resources to accommodate sending and receiving messages for the specific protocol.

Operations:

- √ New communication endpoint (socket) creation
- √ A local address association with the new socket
- √ Allocation of resources in connection-oriented communication
- √ Connection establishment
- √ Sending and receiving data

The type of communication has to be decided when the socket is created (the same communication type for both client and server has to be established)

Socket Primitives for TCP/IP (1)

Create a new communication endpoint

```
int socket(int addr, int socktype, int protocol)
```

Attach a local address pointed by `name` to a socket `s`

```
int bind(SOCKET s, const struct sockaddr *name, int namelen)
```

Sending and receiving data – functions: `int sendto` and `int recvfrom`

Socket Primitives for TCP/IP (2)

Connection-oriented communication

Announce willingness to accept `backlog` connections

```
int listen(SOCKET s, int backlog)
```

Actively attempt to establish a connection (client)

```
int connect(SOCKET s, const struct sockaddr *addr, int
addrlen)
```

Block caller until a connection request arrives (server)

```
int accept(SOCKET s, const struct sockaddr *addr, int addrlen)
```

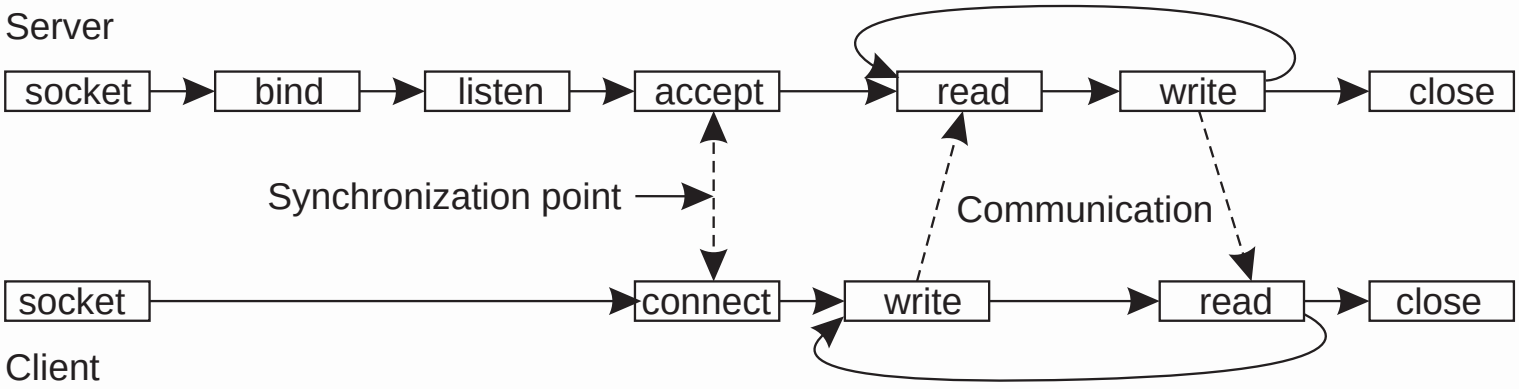
Send some data over the connection

```
int send(SOCKET s, const char *buf, int len, int flags)
```

Receive some data over the connection

```
int recv(SOCKET s, const char *buf, int len, int flags)
```

Sockets – Connection-oriented Communication



The general pattern followed by a client and server for connection-oriented communication using sockets.

Berkeley Sockets – Discussion

Good points:

- √ simple and effective data transmission in parallel and distributed systems,
- √ portable applications can be developed.

Sockets insufficient because:

- √ the wrong level of abstraction supporting only send and receive primitives,
- √ binary format of transmitted data (conversion is necessary),
- √ designed to communicate using general-purpose protocol stacks such as TCP/IP, not suitable in high-speed interconnection networks (with different forms of buffering and synchronization).

Message-Passing Interface (MPI) – Introduction (1)

MPI (Message-Passing Interface)

A widely used standard for writing message-passing programs developed by the MPI working group.
The MPI Forum was constituted since 1993
(`http://www.mpi-forum.org/`).

Group of message-oriented primitives (about 250 subroutines and functions) that would allow developers to easily write highly efficient applications in C, C++ and Fortran.

The interface is suitable for use by fully general MIMD (Multiple Instruction, Multiple Data) programs, as well as those written in style of SPMD (Single Program, Multiple Data).

In SPMD tasks are split up and run simultaneously on multiple processors with different input in order to obtain results faster.

Message-Passing Interface (MPI) – Introduction (2)

MPI assumptions:

- √ communication within a known group of processes,
- √ each group with assigned **ID**,
- √ each process within a group also with assigned **ID**,
- √ all serious failures (process crashes, network partitions) assumed as fatal and without any recovery,
- √ a (**groupID**, **processID**) pair used to identify source and destination of the message,
- √ only **receipt-based transient synchronous communication** not supported, other supported.
- √ group communication is provided.

MPI concepts

The concepts that MPI provides are as follows:

- √ Contexts of communication;
- √ Communicators;
- √ Groups of processes;
- √ Virtual topologies;
- √ Attribute caching.

Communicator

Communicator

Communicator specifies the communication context for the communication operation. Each communication context provides a separate *communication universe*.

Messages are always received within the context they were sent, and messages sent in differen contexts do not interfere.

Communicator – identified by the handle with type MPI_Comm.
MPI_COMM_WORLD – a predefined communicator in MPI (allows communication with all processes accessible after MPI initialization).

Selected MPI Primitives

Starting MPI library

```
int MPI_Init( int *argc, **argv[] )
```

Terminating MPI library

```
int MPI_Finalize( void )
```

The number of processes determination

```
int MPI_Comm_size( MPI_Comm comm, int *size )
```

The rank determination

```
int MPI_Comm_rank( MPI_Comm comm, int *rank )
```

The handle to the group

```
int MPI_Comm_group( MPI_Comm comm, MPI_Group *group )
```

Communication in MPI - Introduction (1)

Sending and receiving messages (point-to-point communication)

- √ **synchronous** communication – tasks synchronize to perform interactions,
- √ **asynchronous** communication – all concurrent tasks execute asynchronously (non-deterministic behavior due to a race conditions).

Three phases of the message transfer:

- √ data is pulled out of the send buffer and a message is assembled,
- √ a message is transferred to receiver,
- √ data is pulled from the incoming message and disassembled into the receive buffer.

Communication in MPI - Introduction (2)

Types of send/receive operations provided in MPI library:

- √ synchronous and asynchronous
  - ⋆ **synchronous** - send operation returns only after the receiver has received the data.
  - ⋆ **asynchronous** - send operation returns just after sending the message.

- √ blocking and non-blocking:
  - ⋆ **blocking** - send/receive operation blocks until it can guarantee that the semantics won't be violated on return irrespective of what happens in the program.
  - ⋆ **non-blocking** - send/receive operation returns before it is semantically safe to do so.

Communication in MPI – Functions (1)

In the message-passing paradigm the communication is initiated by the sender.

Blocking communication:

Asynchronous send (standard)

```
1  int MPI_Send ( void *sendbuf, int count,
2                 MPI_Datatype datatype,
3                 int dest, int tag, MPI_Comm comm )
```

Buffered send – function MPI_Bsend (required buffer allocation - function MPI_Buffer_attach.

Synchronous send – function MPI_Ssend

Communication in MPI – Functions (2)

Blocking receive (standard)

```
1 int MPI_Recv( void *recvbuf, int count,
2           MPI_Datatype datatype,
3           int source, int tag,
4           MPI_Comm comm, MPI_Status *status )
```

Non-blocking communication:

- √ asynchronous send: MPI_Isend
- √ buffered send: MPI_Ibsend
- √ synchronous send: MPI_Issend
- √ non-blocking receive: MPI_Irecv

Blocking and non-blocking testing: MPI_Probe and MPI_Iprobe.

Communication in MPI – Recommendations

- √ Asynchronous blocking send MPI_Send; blocking buffered send MPI_Bsend for large amount of transmitted data, or when more control is needed.
- √ Asynchronous blocking receive MPI_Recv, preceded by non-blocking testing MPI_Iprobe.
- √ Synchronization – barrier function MPI_Barrier.

Communication – Alternative Proposition

Alternative proposition – combined blocking send and receive.

```
1  int MPI_Sendrecv(void *sendbuf, int sendcount,
2                   MPI_Datatype sendtype, int dest,
3                   int sendtag,
4                   void *recvbuf, int recvcount,
5                   MPI_Datatype recvtype, int source,
6                   int recvtag,
7                   MPI_Comm comm, MPI_Status *status )
```

Collective Communication in MPI

All processes in the group call the communication routine, with matching arguments

- √ Synchronization – barrier function: MPI_Barrier
- √ Broadcast from one member to all members of a group: MPI_Bcast
- √ Scatter data from one member to all members of a group: MPI_Scatter, MPI_Scatterv
- √ Gather data from all members of a group to one member: MPI_Gather, MPI_Gatherv
- √ ”All to all” send: MPI_Alltoall
- √ Global reduction operations: MPI_Reduce

Stream-Oriented Communication – Introduction

Data stream – sequence of data units (can be applied to discrete and continuous media).

Types of streams:

a. **simple** – consists of only a single sequence of data,

b. **complex** – consists of several related simple streams.

Stream-Oriented Communication – exchange time-dependent information (such as: audio and video stream). Forms of communication in which timing plays a crucial role.

Stream-Oriented Communication – Example

Example:

- √ an audio stream built up as a sequence of 16-bit samples each representing the amplitude of the sound wave as it is done through PCM (Pulse Code Modulation),
- √ audio stream represents CD quality, i.e. 44100Hz,
- √ samples to be played at intervals 1/44100sec.

To reproduce the original sound, it is essential that the samples in the audio stream are played out in the order they appear in the stream and at intervals of exactly 1/44100sec.

Continuous Media vs. Discrete Media

Medium

A medium refers to the means by which the information is conveyed and represented (sound, image, text,...).

Continuous (representation) media

The temporal relationships between different data items are **fundamental** to correctly interpreting the data (sound, series of images,...).

Discrete (representation) media

The temporal relationships between different data items are **not fundamental** to correctly interpreting the data (text, files with code,...).

Which facilities a distributed system should offer to exchange time-dependent information such as audio and video streams? Support for:

- √ exchange of time-dependent data **continuous media**,
- √ multimedia communication (**continuous & discrete media**).

Transmission Modes

Asynchronous transmission mode

Data items in a stream are transmitted one after the other, but there are no further timing constraints on when transmission of items should take place.

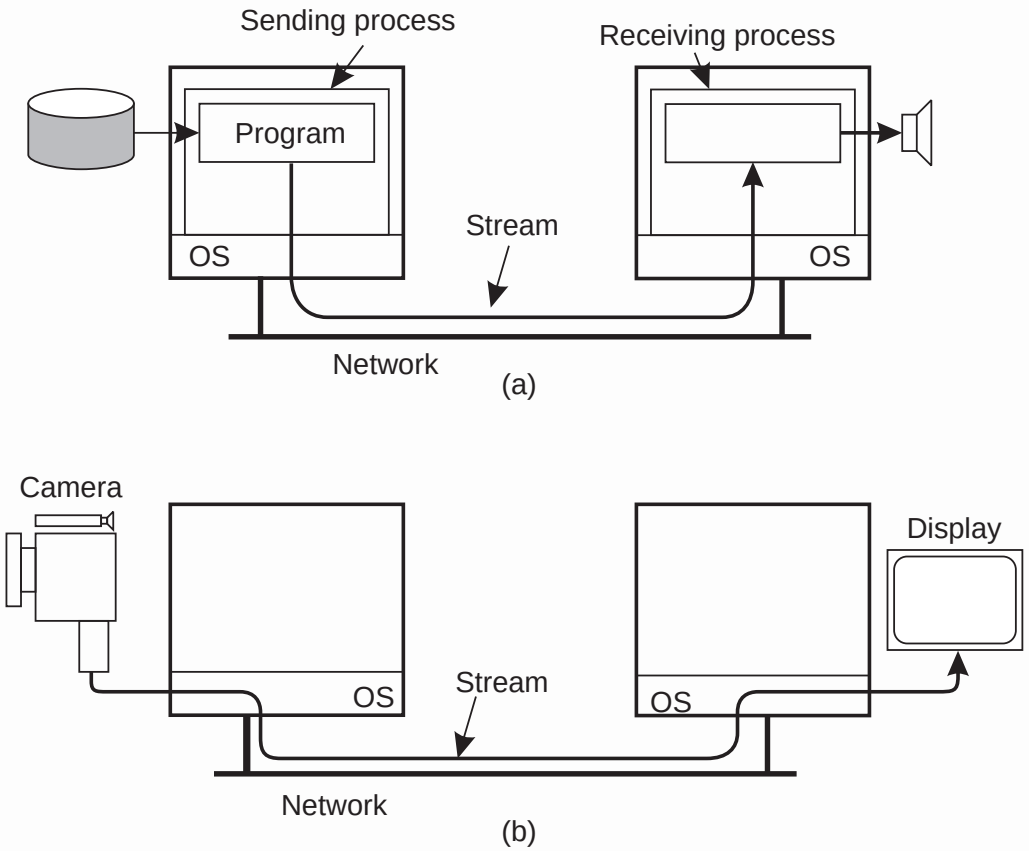
Synchronous transmission mode

Maximum end-to-end delay defined for each unit in a data stream.

Isochronous transmission mode

It is necessary that data units are transferred on time. Data transfer is subject to bounded (delay) jitter.

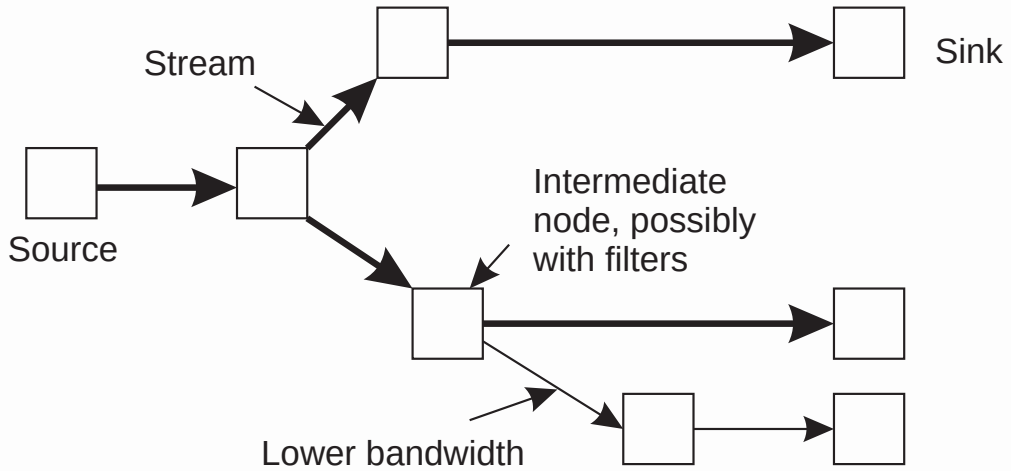
Stream Communication – System Architecture



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

b. Setting up a stream directly between two devices.

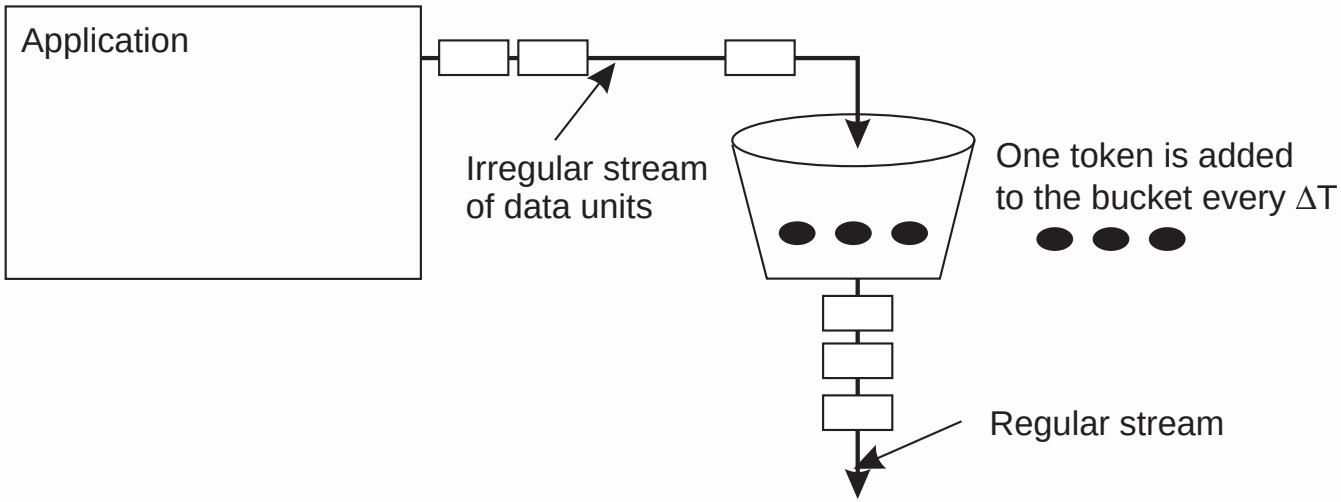
Stream Communication – Multicasting



An example of multicasting a stream to several receivers.

- √ problem with receivers having different requirements with respect to the quality of the stream,
- √ **filters** to adjust the quality of an incoming stream, differently for outgoing streams.

QoS – Token Bucket Algorithm



The principle of a token bucket algorithm

- √ tokens generated at a constant rate; each token represents a fixed number of **k** bytes of data,
- √ tokens buffered in a bucket which has limited capacity (when the bucket is full, tokens are dropped).

QoS – Specification

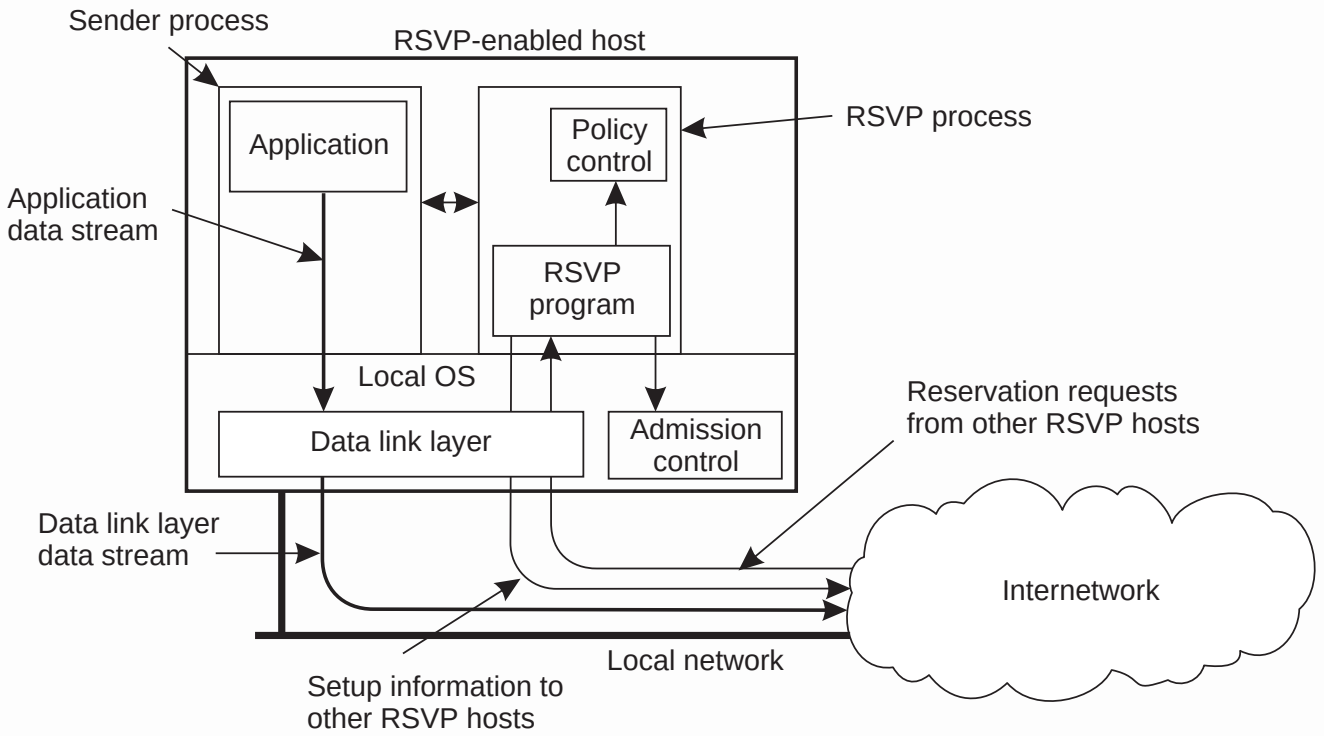
QoS (**Quality of Service**)

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

A flow specification.

Time-dependent requirements among other **Quality of Service (QoS)** requirements.

Setting Up a Stream (RSVP)

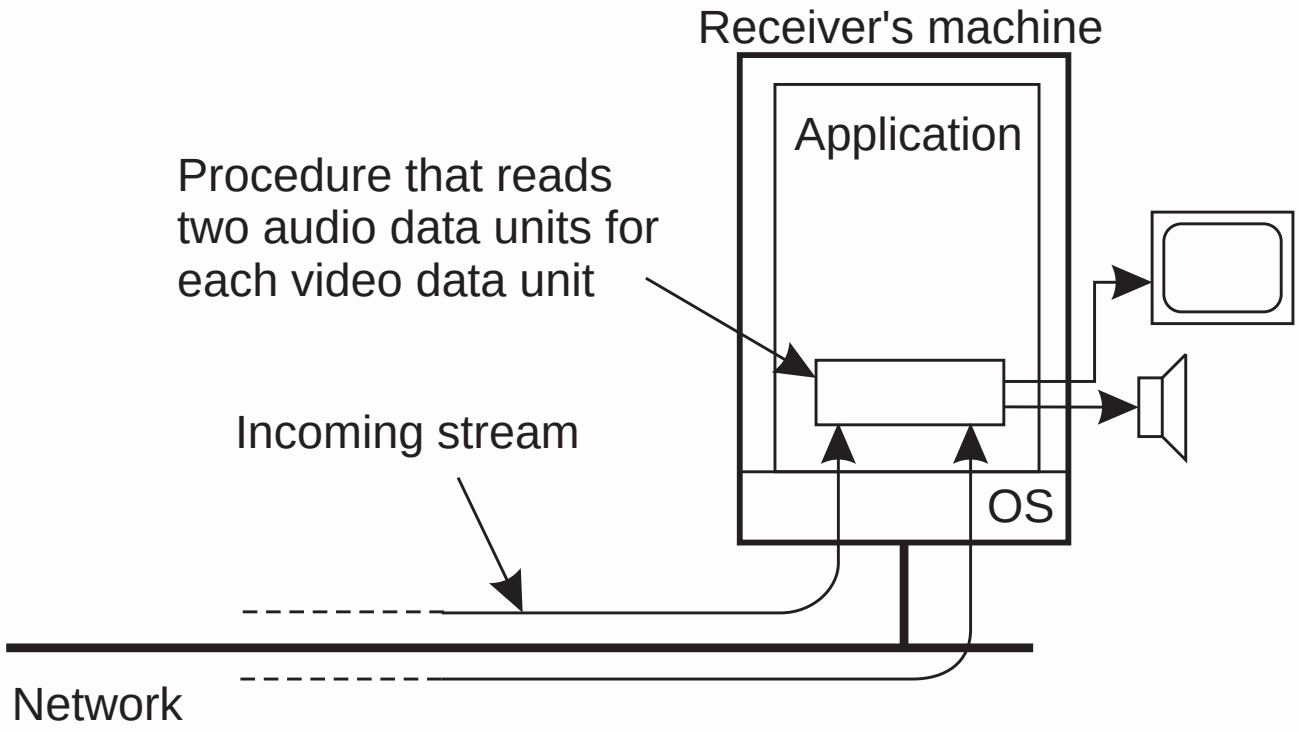


The basic organization of RSVP (*Resource reSerVation Protocol*), transport-level protocol for resource reservation in a distributed system.

Streams Synchronization (1)

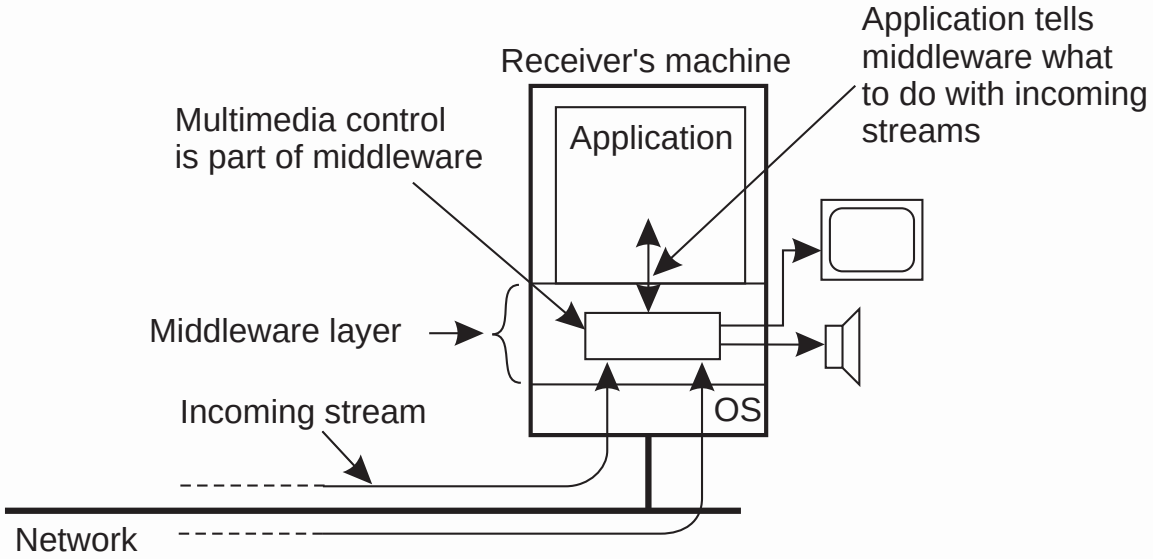
Two issues:

- √ the basic mechanisms for synchronizing two streams,
- √ the distribution of those mechanisms in a network.



Streams Synchronization (2)

Typical approach for many multimedia middleware systems.



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

Multiplex of all substreams into a single stream and demultiplexing at the receiver. Synchronization is handled at multiplexing/demultiplexing point (MPEG).