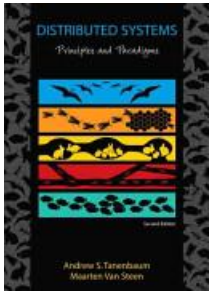
Chapter 4: COMMUNICATION
Parts 3-4-5

Communications in Distributed Systems

Message-oriented communication, Stream-oriented communication, Application-level multicast



Thanks to the authors of the textbook [**TS**] for providing the base slides. I made several changes/additions.
These slides may incorporate materials kindly provided by Prof. Dakai Zhu.
So I would like to thank him, too.

Turgay Korkmaz
korkmaz@cs.utsa.edu

Chapter 3: Communications

- FUNDAMENTALS
  - Layered Protocols
  - Grand tour of computer networking, the Internet
  - Socket Programming
- REMOTE PROCEDURE CALL
  - Basic RPC Operation
  - Parameter Passing
  - Asynchronous RPC
  - RMI
  - CORBA
- MESSAGE-ORIENTED COMMUNICATION
  - Transient and Persistent Communication
- STREAM-ORIENTED COMMUNICATION
  - Support for Continuous Media and Quality of Service
  - Stream Synchronization
- MULTICAST COMMUNICATION
  - Application-Level Multicasting
  - Gossip-Based Data Dissemination

Objectives

- To understand how processes communicate (the heart of distributed systems)
- To understand low-level message passing
  - sockets
- To learn higher-level communication mechanisms
  - RPC, RMI, CORBA
- To understand various forms of communications and their issues
  - Msg-, Stream-oriented communication, multicast, etc.

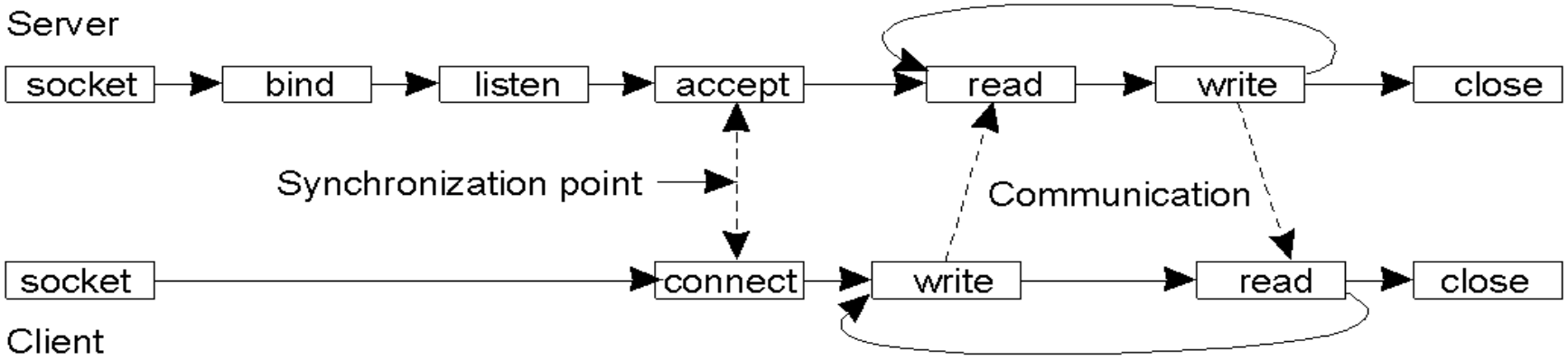
RPC (RMI) might not be appropriate in some cases, e.g., when sender and receiver are not running at the same time

MESSAGE-ORIENTED COMMUNICATION

Message-oriented Transient Communication

We already covered this when talking about Sockets

- Many distributed systems built on top of simple transient message-oriented model (TCP, UDP)



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

Message-oriented Transient Communication

Message-Passing Interface (MPI)

- Sockets designed for network communication (e.g., TCP/IP)
  - + Support simple send/receive primitives
  - - Abstraction not suitable for other protocols in clusters of workstations or massively parallel systems
    - Need an interface with more advanced primitives
- Large number proprietary libraries and protocols are provided
  - But they are incompatible and makes it hard to port an application
  - Need for a standard interface
- Message-passing interface (MPI) – platform independent
  - Designed for parallel applications (uses transient communication)
  - Directly uses the underlying communication facilities
  - Communications take place within a group
    - Each endpoint is a *(groupID, processID)* pair

MPI Primitives

Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer (async)
MPI_send	Send a message and wait until copied to local or remote buffer (sync)
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 (do not block, async)
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

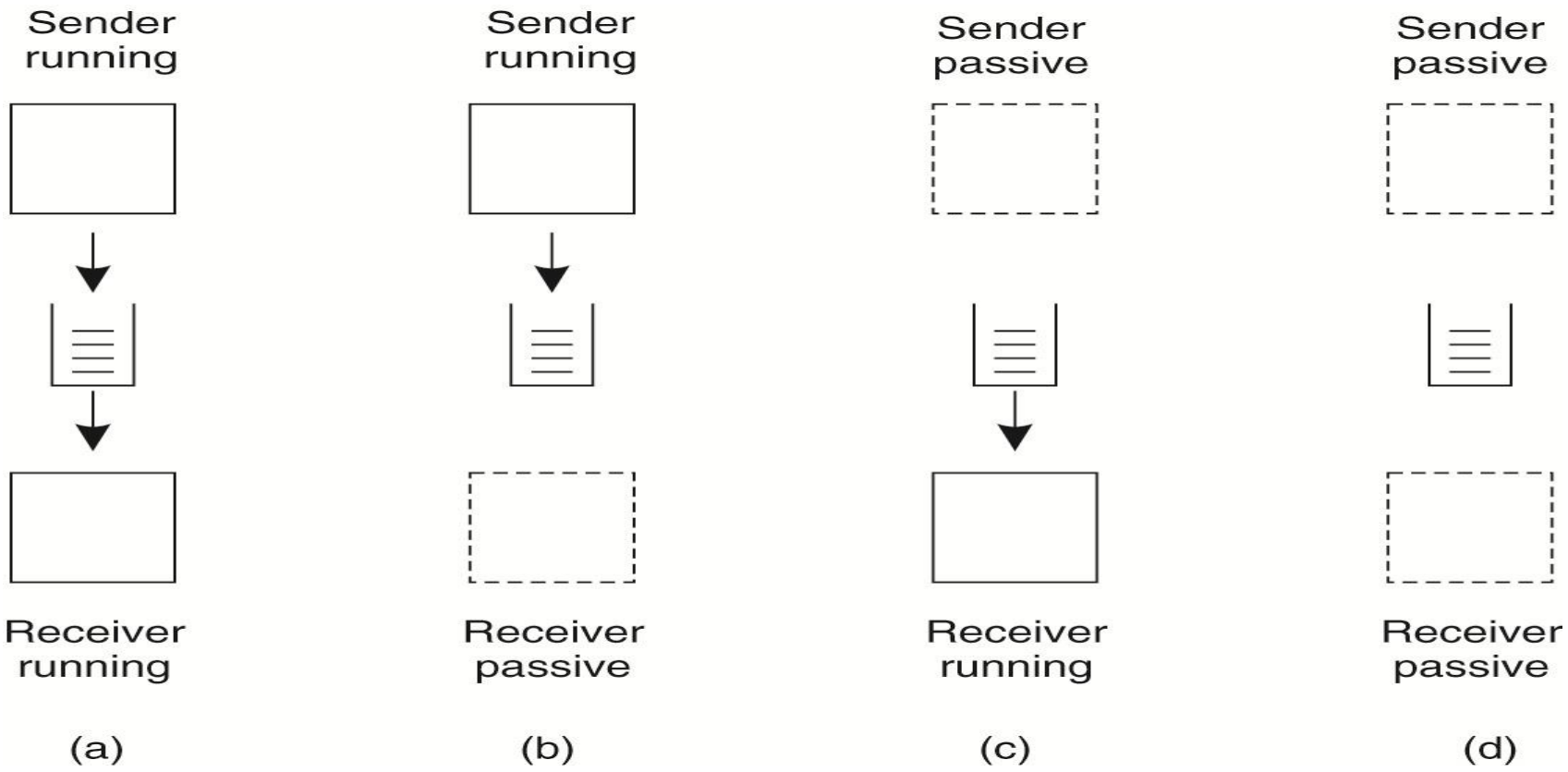
Different forms allow MPI implementers to optimize performance

Message-oriented Persistent Communication

Message-Queuing System (or MOM—Message-Oriented Middleware)

- Message queuing systems
  - Support **asynchronous persistent** communication through support of middleware-level **queues**.
  - Communicate by inserting messages in queues providing intermediate storage for message
  - Sender and receiver could be inactive (e.g., e-mail)
- Sender is only guaranteed that message will be eventually inserted in recipient's queue
  - No guarantees on when or if the message will be read
  - "Loosely coupled communication" with four combinations

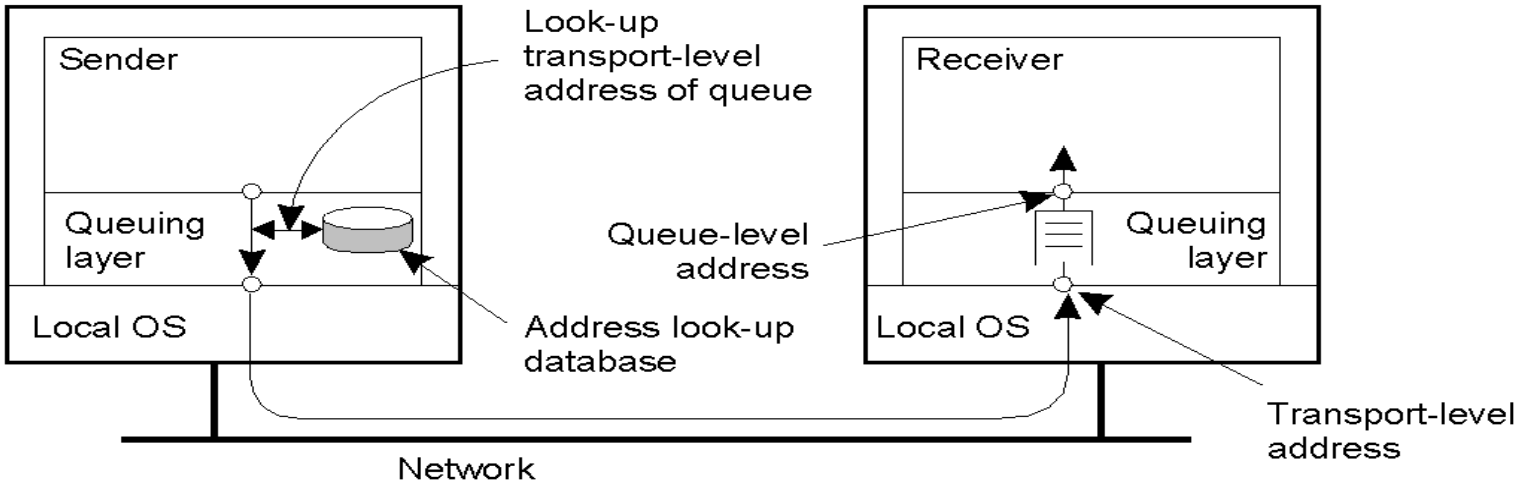
Message-Queuing Systems



Messages should have a system wide unique ID for destination…
Also we need

a common messaging protocol and
simple primitives to be able to send/receive messages…

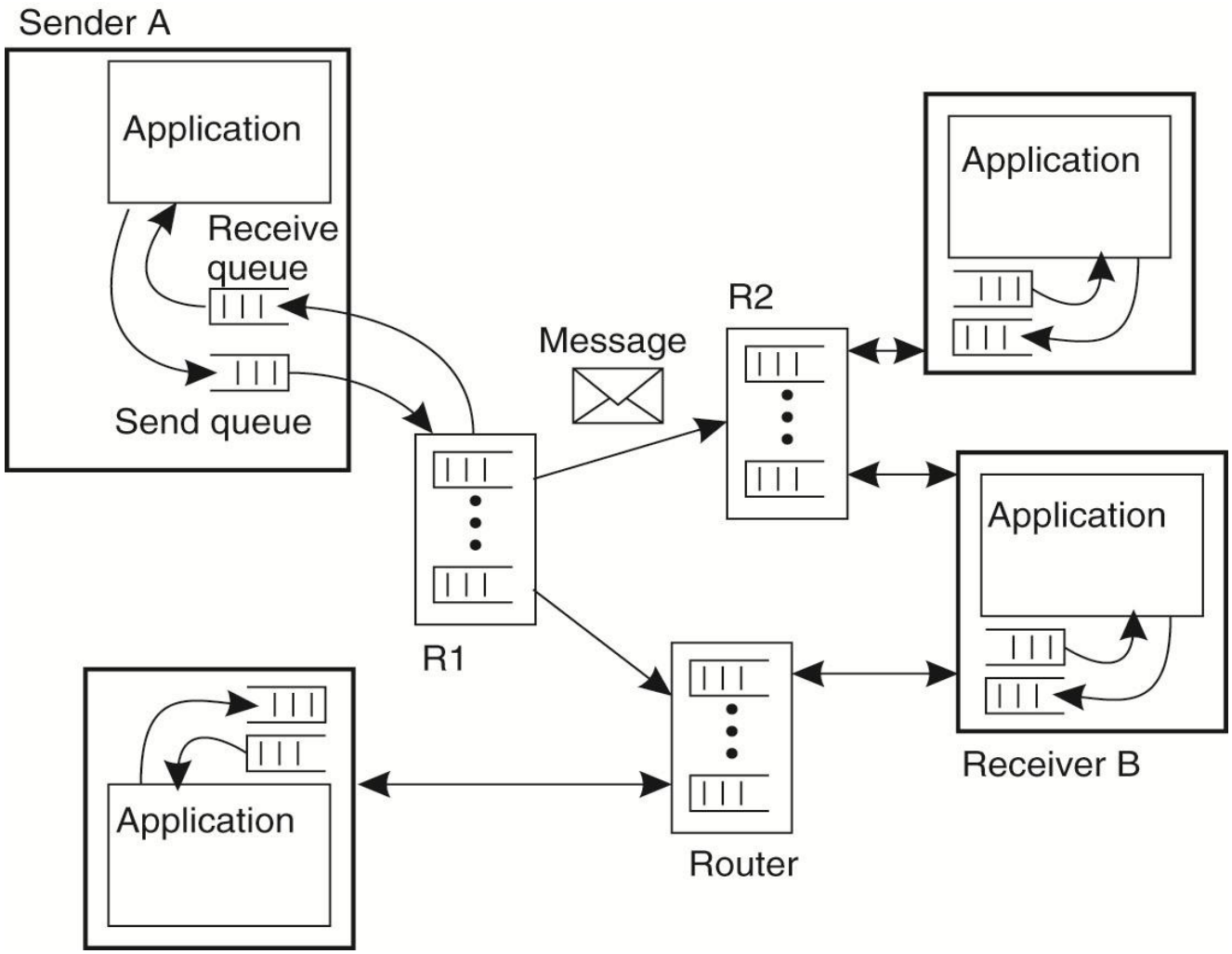
Message-Queuing System Architecture



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.

Message-queuing system should map queues to network locations…

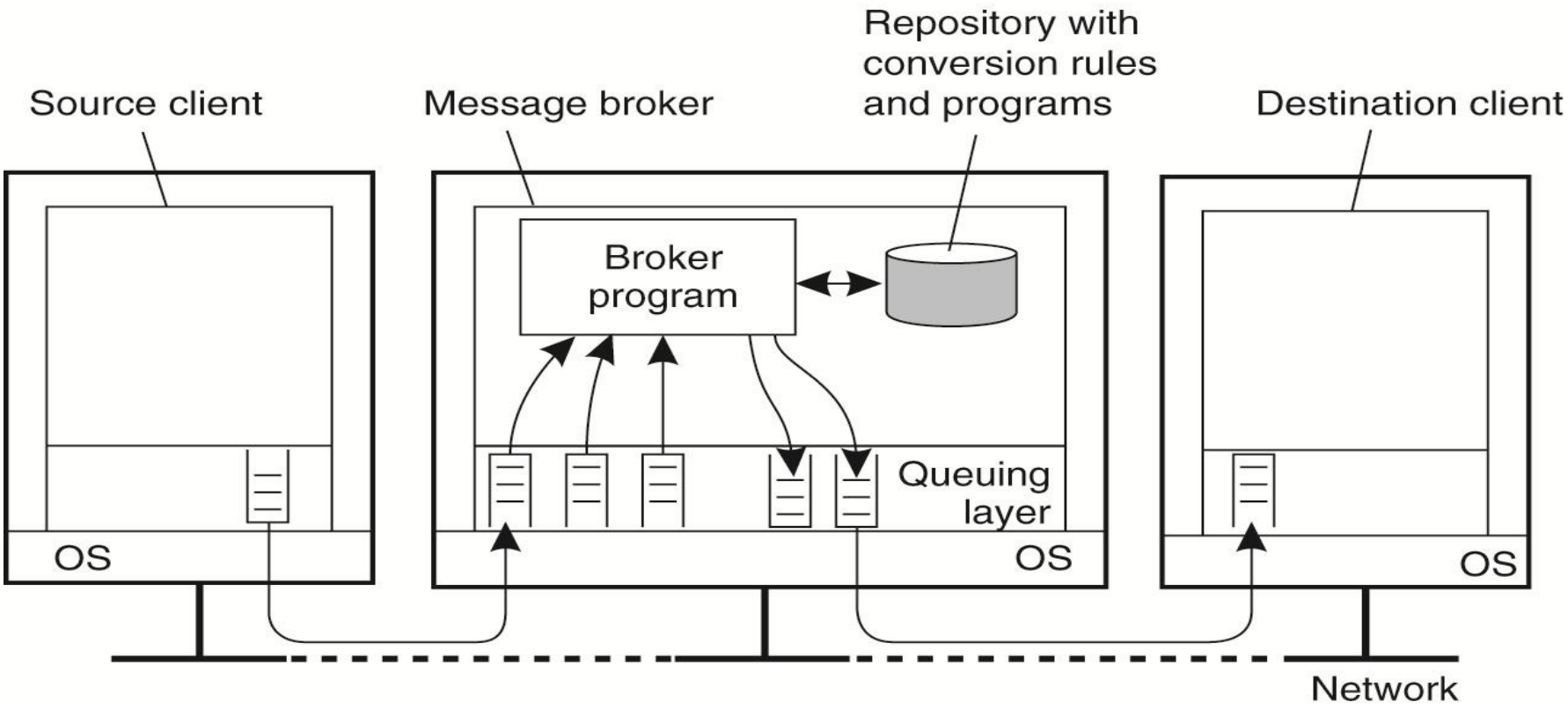
Message-Queuing System with Routers



Message Broker

Problem: Message queuing systems assume a common messaging protocol. But if a new application requires a separate format then all potential receivers need to be updated!

Solution: Learn to live with different formats! Accordingly, have a message broker that *transforms incoming messages to target format*



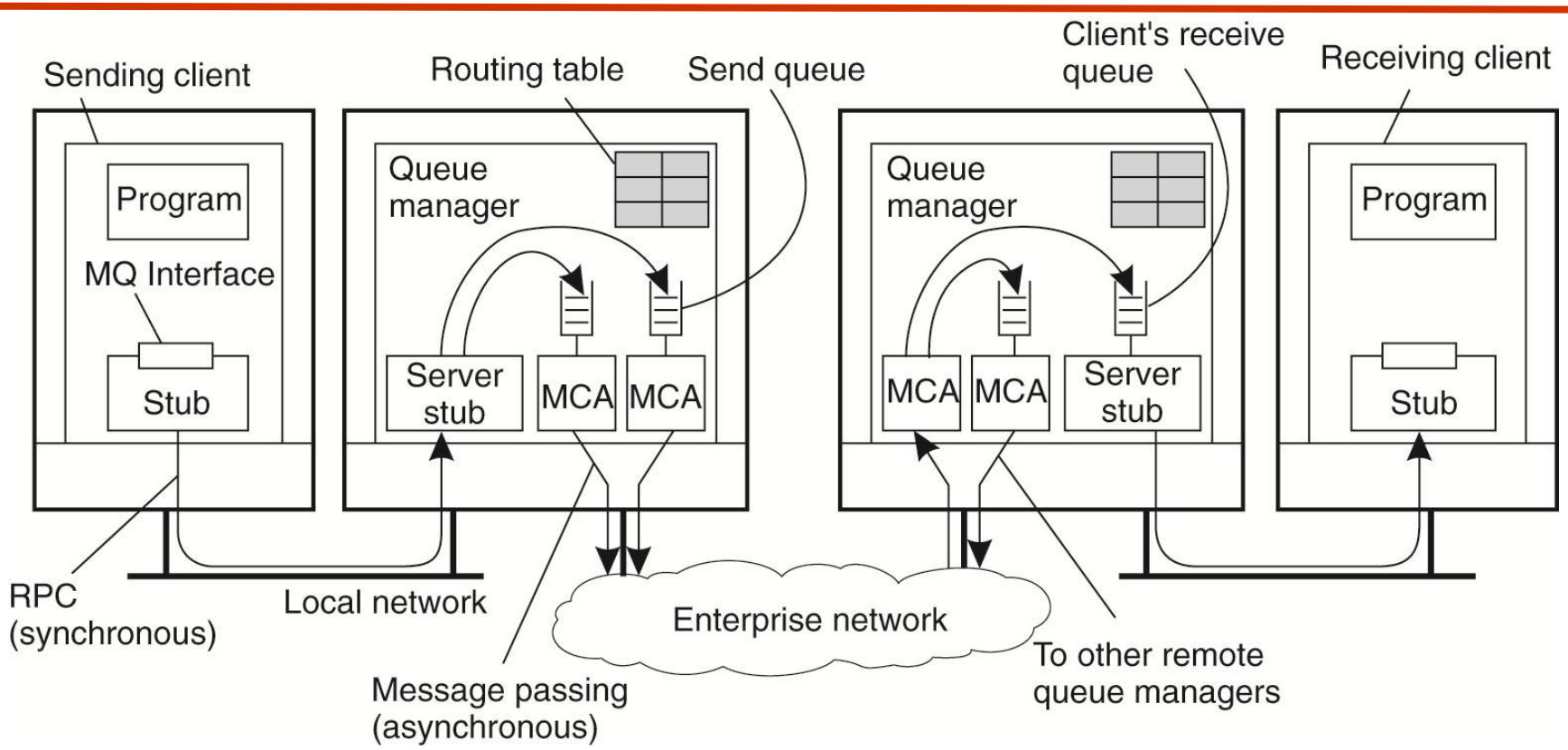
A note on MQS or MOM

- E-mail is a special MQS, which provides direct communication support for end users
- General MQS provides persistent communication between any processes and support various other applications such as workflow, groupware, batch processing
- Requires more advanced features than e-mail
  - Guarantee message delivery,
  - Message priorities
  - Logging facility,
  - Efficient multicasting, routing
  - Load balancing
  - Fault tolerance

Message Broker

Very often acts as an **application gateway**
May provide **subject-based** routing capabilities ⇒ **Enterprise Application Integration**

Example: IBM's WebSphere MQ



- Application-specific messages are put into, and removed from queues
- Queues reside under the regime of a queue manager
- Processes can put messages only in local queues, or through RPC
- Message transfer between different queues requires a channel

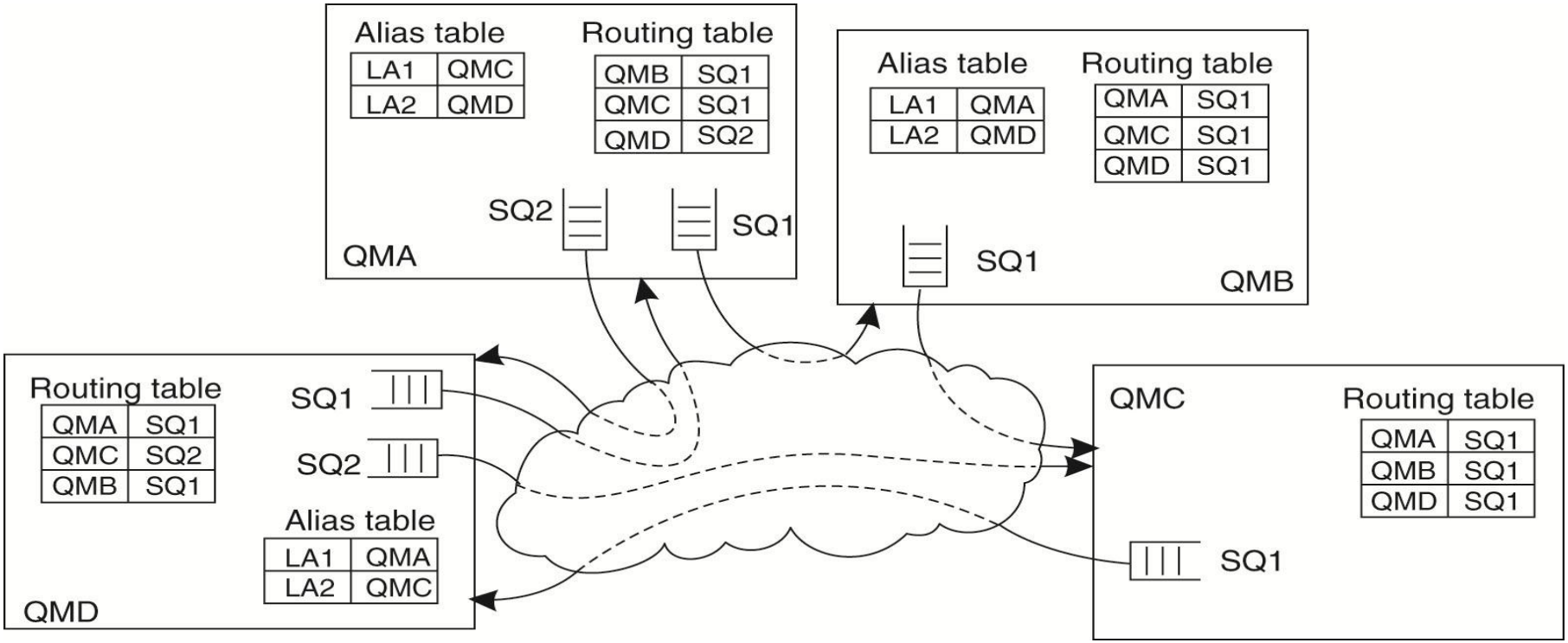
Example: IBM's WebSphere MQ: Channel

- There is a message channel agent (MCA) at each endpoint of a channel
- Message channel agents are responsible for:
  - Setting up channels using lower-level network communication facilities (e.g., TCP/IP)
  - (Un)wrapping messages from/in transport-level packets
  - Sending/receiving packets
- Overall behavior of a channel and MCA is controlled by various attributes

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

Example: IBM's WebSphere MQ: Routing

- By using logical names, in combination with name resolution to local queues, it is possible to put a message in a remote queue
- Routes are set up manually (system administration)



How to exchange time-dependent information (e.g., audio, video)?

STREAM-ORIENTED COMMUNICATION

Stream Oriented Communication

- So far the communication was discrete (complete unit of information is exchanged), timing has no effect on correctness…
- In some cases, timing plays a crucial role…
  - Examples: audio and video ("continuous media")
    - Video conference sends and receives 30 frames/s
  - Animations
  - Sensor data
- Push mode: no explicit requests for individual data units beyond the first "play" request

Transmission of Continuous Media

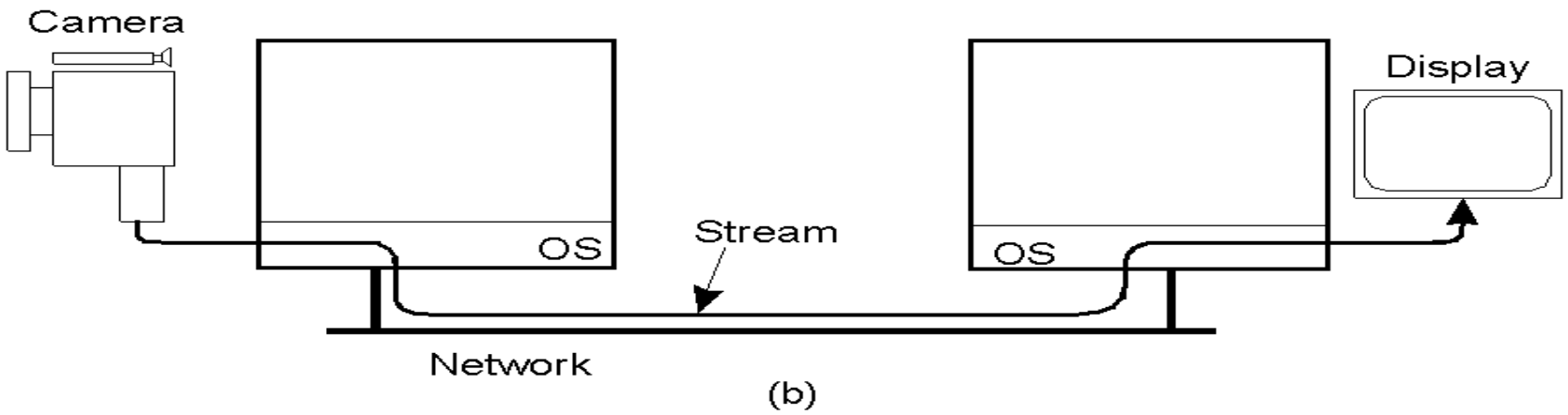
Different timing guarantees: 3 types of transmission

- **Asynchronous:** no restrictions with respect to **when** data is to be delivered
  - (e.g., a file can be transferred as a data stream)
- **Synchronous:** define a maximum end-to-end delay for individual data packets
  - (e.g., sensor data need to be send less than 1 sec, no harm if sent faster)
- **Isochronous:** define a maximum and minimum end-to-end delay (**jitter** is bounded)
  - (e.g., multimedia streams, audio, video streams)

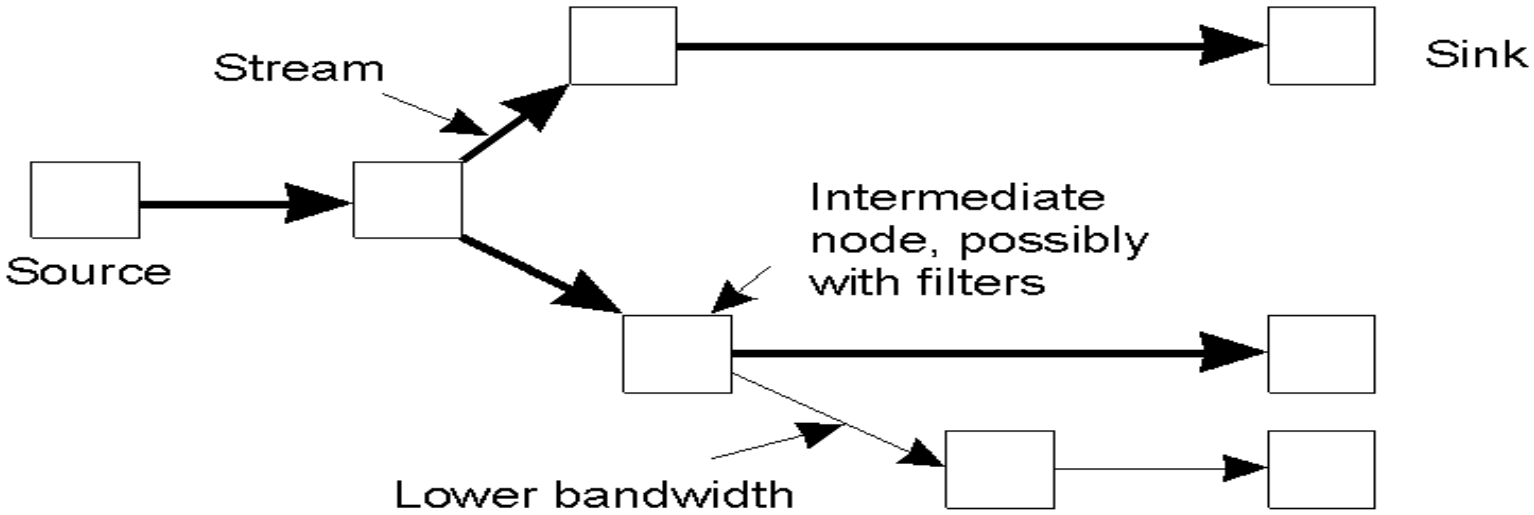
Stream

- **Definition:** A (continuous) data stream is a connection-oriented communication facility that supports **isochronous** data transmission
- Common stream characteristics
  - Streams are unidirectional unless interactive
  - There is generally a single **source**, and one or more **sinks**
  - Two types of streams:
    - **Simple**: single flow of data, e.g., audio or video
    - **Complex**: multiple data flows, e.g., stereo audio or combination audio/video
      - Synchronization of sub-streams is important!
  - Live streaming vs. Stored streaming
    - Less vs more opportunity for tuning

Examples


Camera
OS
Stream
OS
Display
Network

(b)

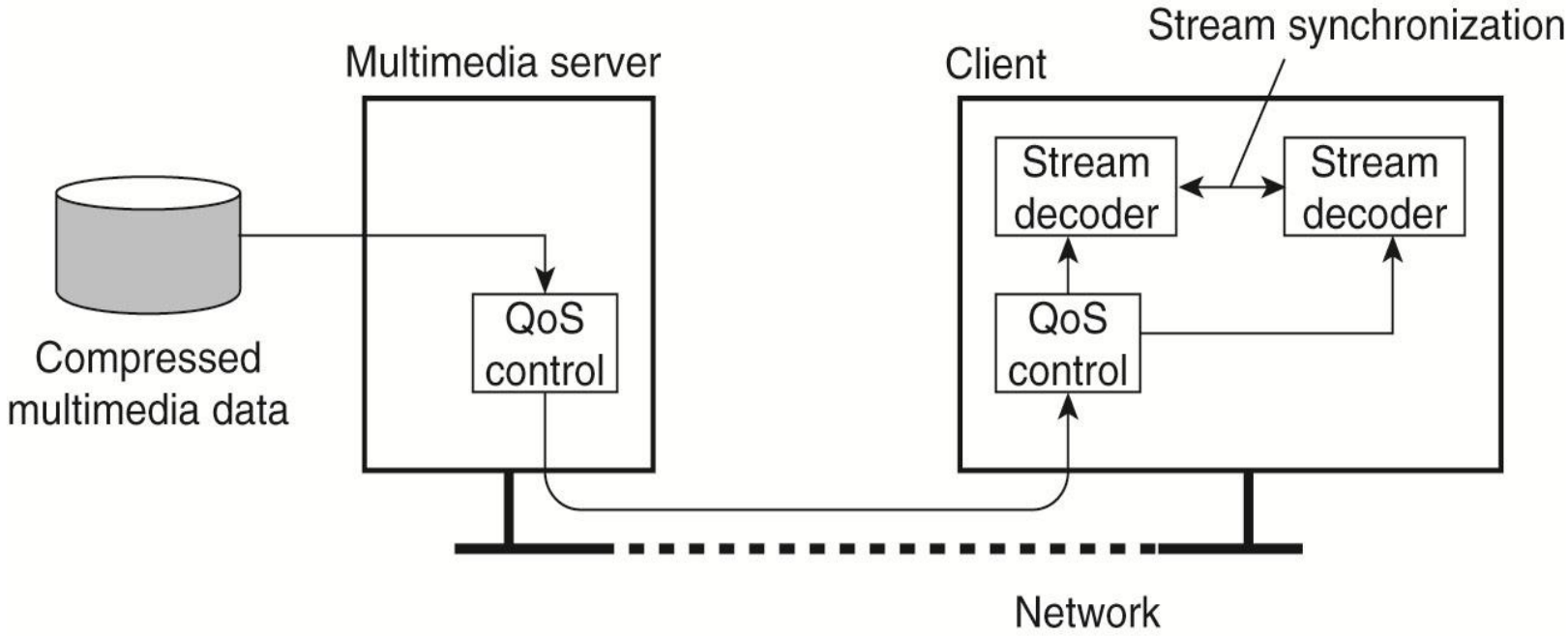

Stream
Sink
Source
Intermediate
node, possibly
with filters
Lower bandwidth

Quality of Service (QoS)

- Time-dependent and other nonfunctional requirements are specified as *Quality-of-Service (QoS)*
  - Requirements/desired guarantees from the underlying systems
  - Application specifies workload and requests a certain service quality
  - Contract between the application and the system

  - The required **bit rate** at which data should be transported.
  - The **maximum delay** until a session has been set up (i.e., when an application can start sending data).
  - The **maximum end-to-end delay** (i.e., how long it will take until a data unit makes it to a recipient).
  - The maximum delay variance, or **jitter**.
  - The **maximum round-trip delay**.

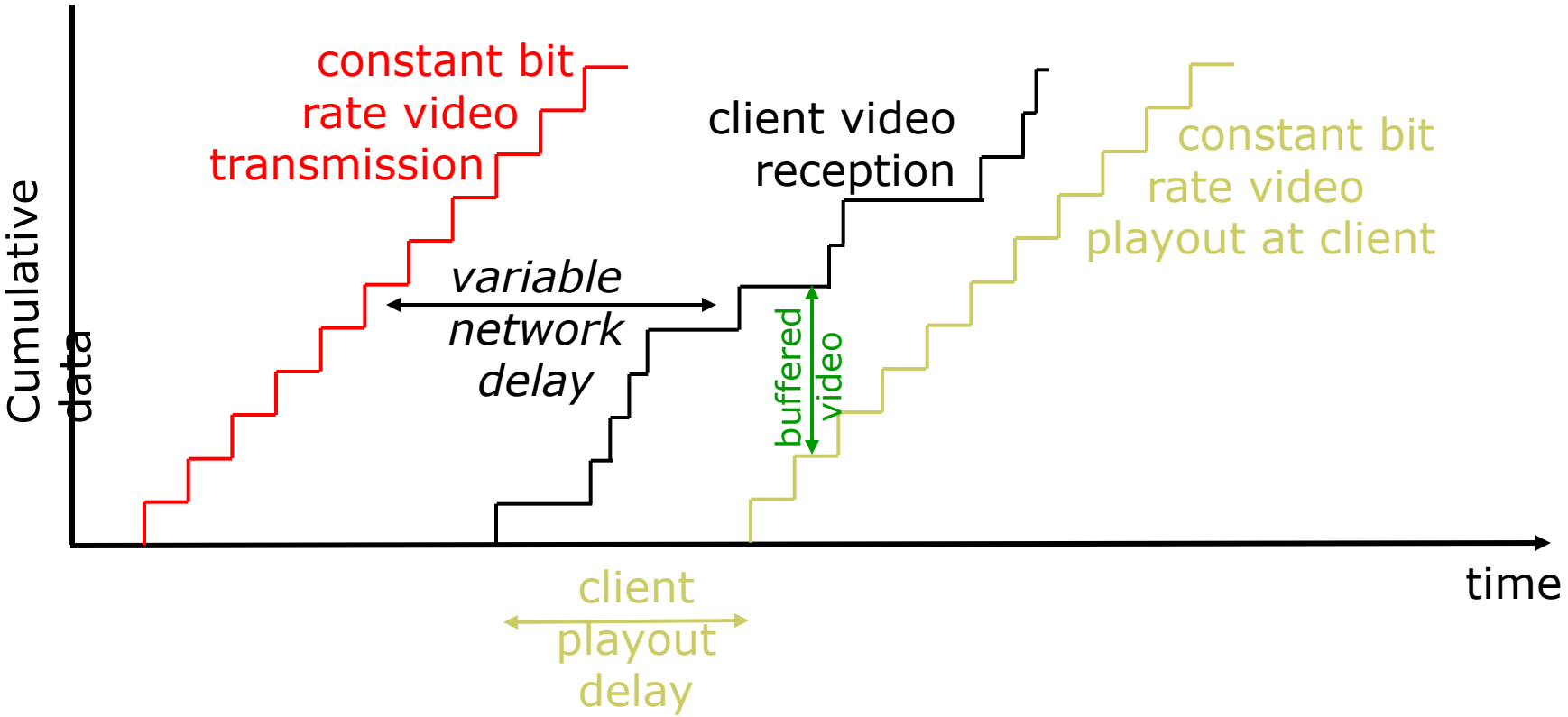
How to Enforce QoS



New network-level mechanisms and protocols
vs.
Application-level e2e mechanisms on the existing network (Internet)

Example: E2E QoS Mechanism

Client Buffering to reduce justter



- client-side buffering, playout delay compensate for network-added delay, delay jitter

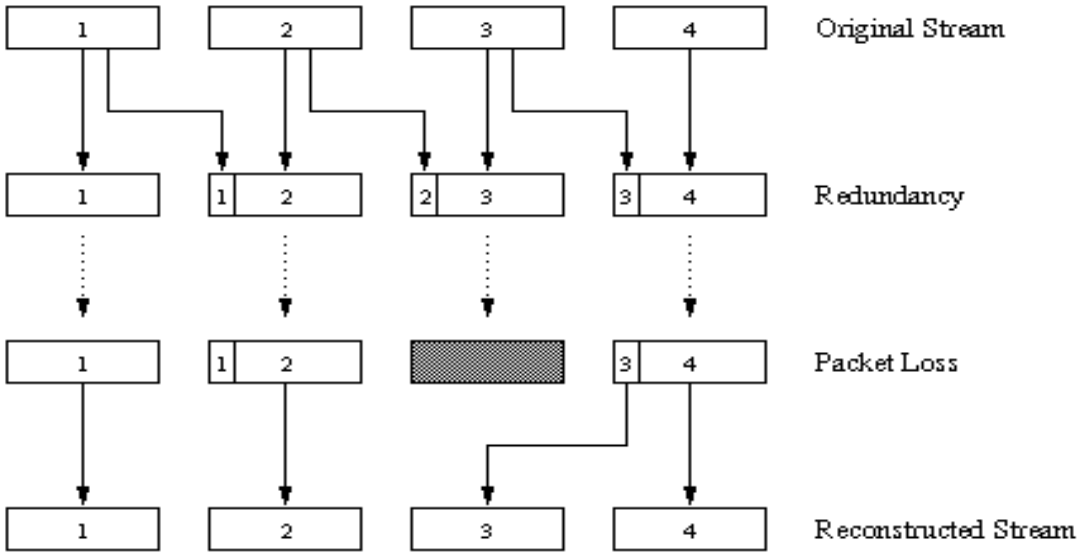
From Computer Networking by Kurose and Ross.

Example: E2E QoS Mechanism

to recovery from packet loss (1)

FEC scheme

- "piggyback lower quality stream"
- send lower resolution audio stream as redundant information
- e.g., nominal stream PCM at 64 kbps and redundant stream GSM at 13 kbps.

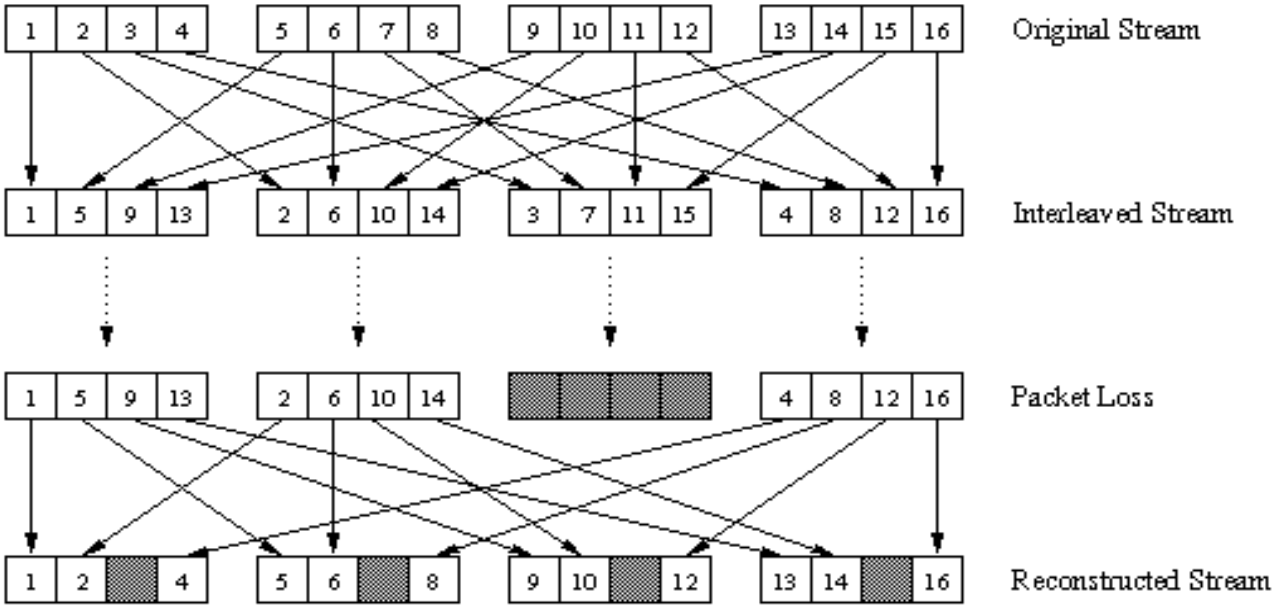


- whenever there is non-consecutive loss, receiver can conceal the loss.
- can also append (n-1)st and (n-2)nd low-bit rate chunk

From Computer Networking by Kurose and Ross.

Example: E2E QoS Mechanism

to recovery from packet loss (2)



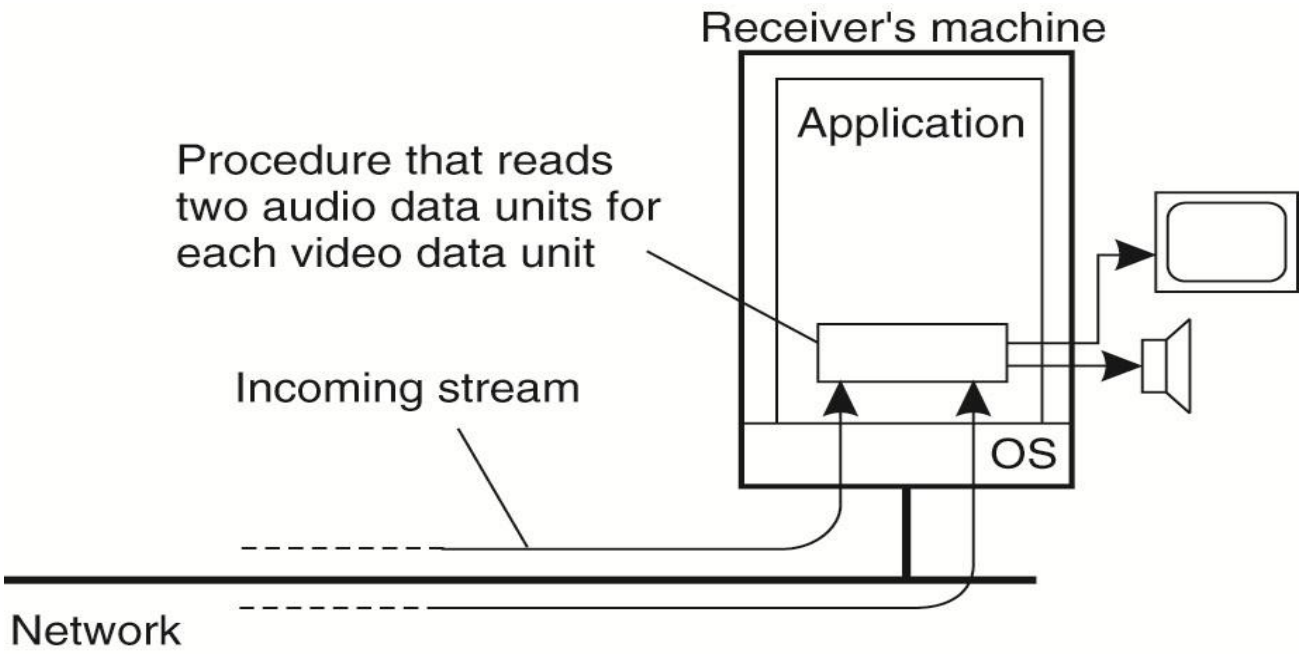
Interleaving

- chunks divided into smaller units
- for example, four 5 msec units per chunk
- packet contains small units from different chunks
- if packet lost, still have most of every chunk
- no redundancy overhead, but increases playout delay

From Computer Networking by Kurose and Ross.

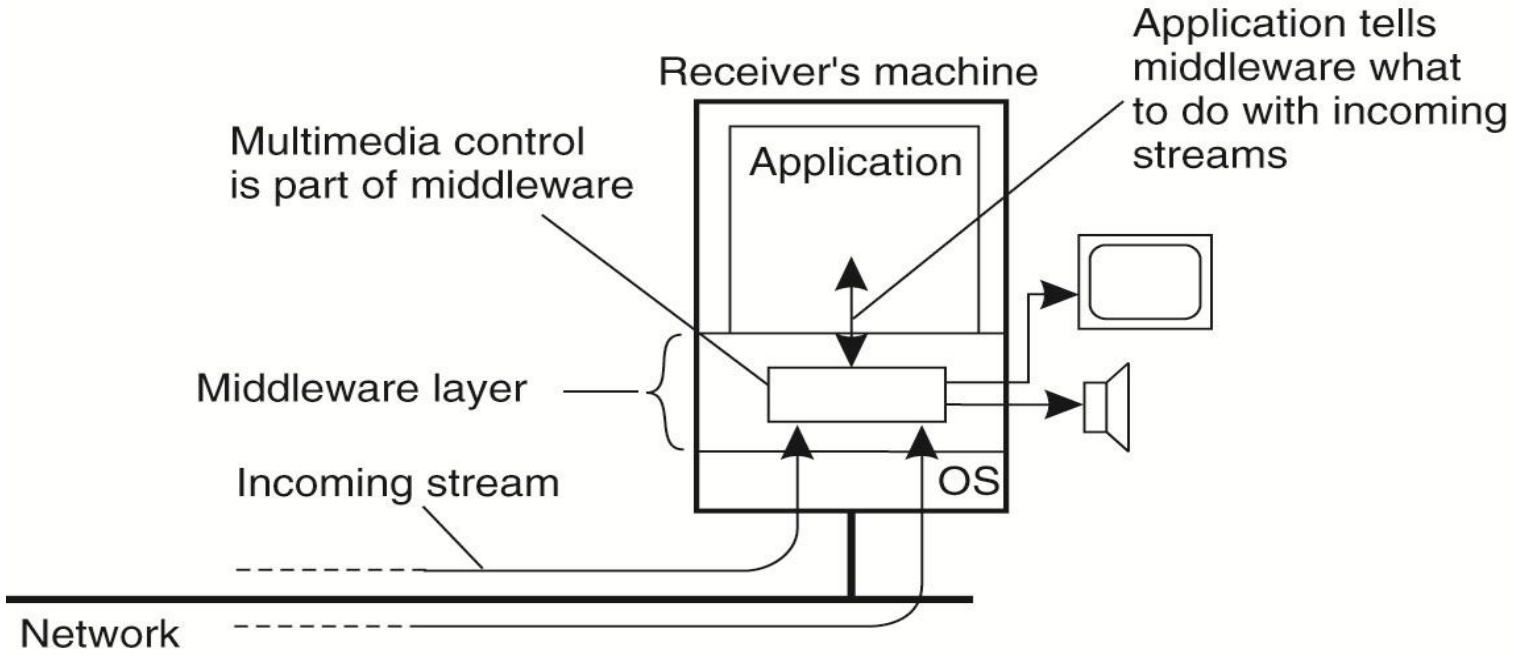
Stream Synchronization

- **Problem:** Given a complex stream, how do you keep the different sub-streams in synch?
  - Stereo sound: two channels are played together Difference should be less than 20–30 *μ*sec!
- A simple approach: let application be responsible
  - Alternate between two channels



Stream Synchronization (cont.)

- Alternative Solution: **high-level interfaces**
  - Multiplex all sub-streams into a single stream, and demultiplex at the receiver.
  - Synchronization is handled at multiplexing/demultiplexing point (MPEG).



Numeric example….

APPLICATION-LEVEL MULTICAST

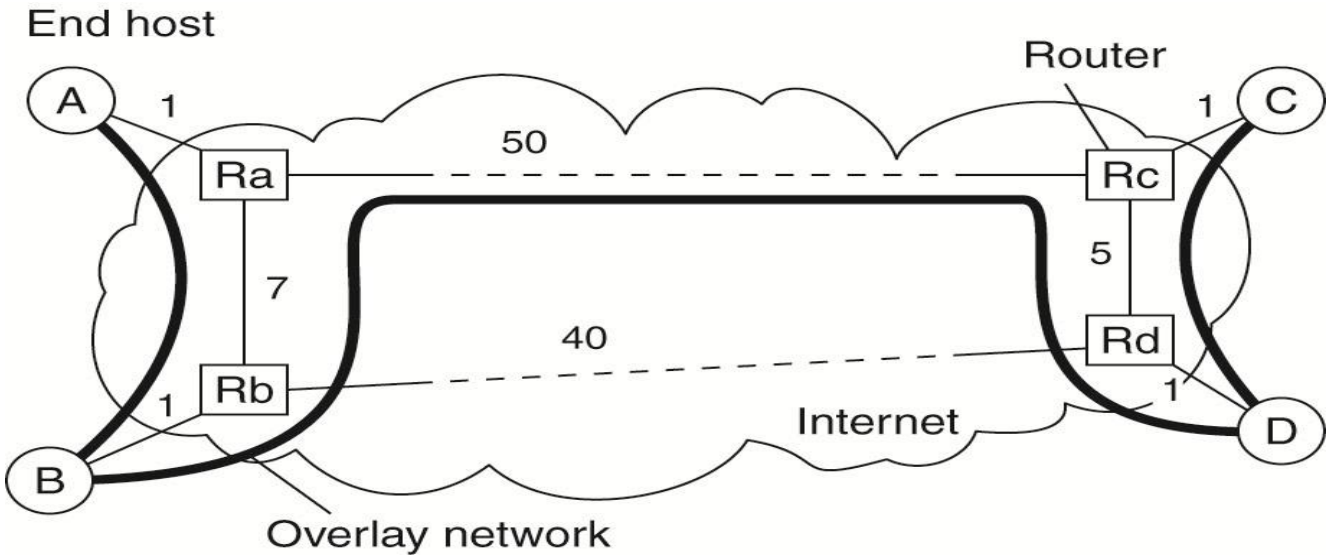
Application-level Multicast

- Application-level multicasting (ALM)
  - Organize nodes of a **distributed system** into an **overlay network** and use that network to disseminate data
- Epidemic-based data dissemination

ALM: Overlay Construction with Chord

- **Example:** Chord-based peer-to-peer system
- Initiator generates a **multicast identifier** *mid*
- Lookup succ(mid), the node responsible for mid.
- Request is routed to succ(mid) → the **root**
- If *P* wants to join, it sends a **join** request to the root
- When request arrives at *Q*:
  - *Q* has not seen a join request before ⇒ it becomes **forwarder**; *P* becomes child of *Q*. Join request continues to be forwarded.
  - *Q* knows about tree ⇒ *P* becomes child of *Q*. No need to forward join request anymore.

ALM: Some costs



- Link stress: How often does an ALM message cross the same physical link?

  Example: message from A to D needs to cross (Ra,Rb) twice.

- Stretch: Ratio in delay between ALM-level path and network-level path.

  Example: messages B to C follow path of length 71 at ALM, but 47 at network level → stretch = 71/47.

Epidemic Algorithms

- **Basic idea:** no write–write conflicts, single writer
  - Update operations are initially performed at a few replicas
  - A replica passes its updated state to a limited neighbors
  - Update propagation is **lazy**, i.e., not immediate
  - Eventually, each update should reach every replica
- Propagation models
  - **Anti-entropy:** Each replica regularly chooses another replica at random, and exchanges state differences, leading to identical states at both afterwards
  - **Gossiping:** A replica which has just been updated (i.e., has been **contaminated**), tells a number of other replicas about its update (contaminating them as well).

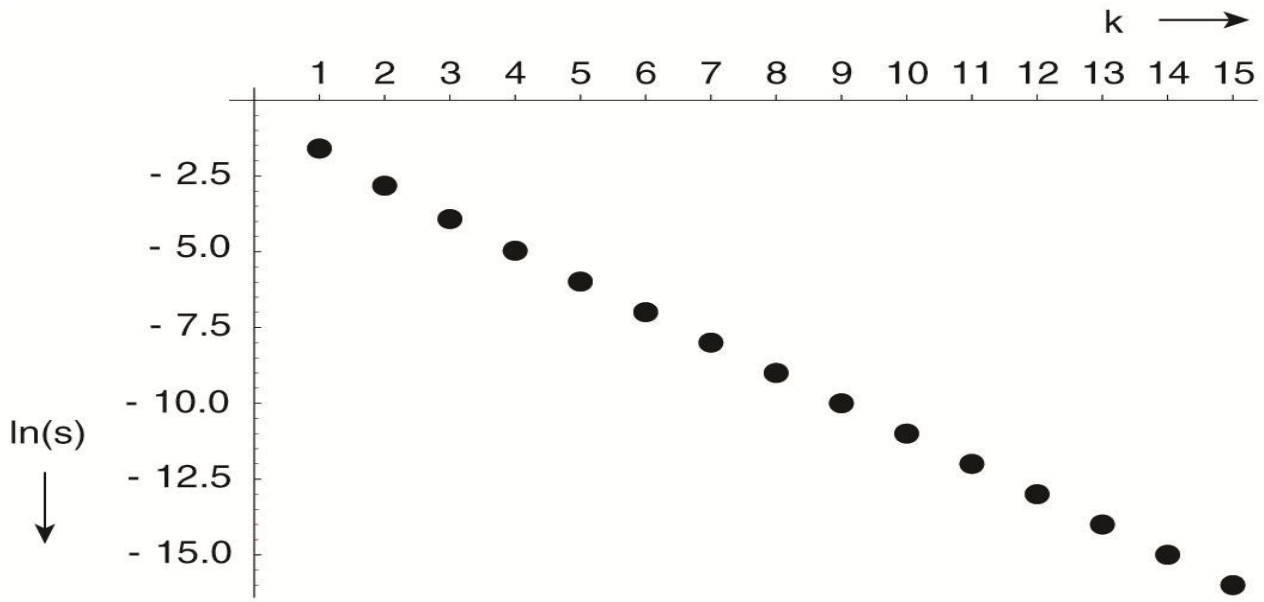
Anti-Entropy Propagation Model

- Node P selects node *Q* from the system at random
- Schemes to propagates updates
  - **Push**: *P* only sends its updates to *Q*
  - **Pull**: *P* only retrieves updates from *Q*
  - **Push-Pull**: *P* and *Q* exchange mutual updates (after which they hold the same information).

- **Observation:** for push-pull it takes **O(*log*(*N*))** rounds to disseminate updates to all *N* nodes (**round** = when every node as taken the initiative to start an exchange).

Gossiping Propagation Model

- A server *S* having an update to report, contacts other servers. If a server is contacted to which the update has already propagated, *S* stops contacting other servers with probability 1/*k*.
- Gossiping alone is not enough for full propagation



Deleting Values

- **Fundamental problem:** We cannot remove an old value from a server and expect the removal to propagate.
- **Solution:** Removal has to be registered as a special update (**Death certificate)**
- When to remove a death certificate?
  - Run a global algorithm to detect whether the removal is known everywhere, and then collect the death certificates (looks like garbage collection)
  - Assume death certificates propagate in finite time, and associate a maximum lifetime for a certificate (can be done at risk of not reaching all servers)

Example: Data dissemination

- Many variants of data dissemination
- **Aggregation:** every node *i* maintain a variable x_i.
- When two nodes gossip, they each reset their variable to: $x_a, x_b \rightarrow (x_a+x_b)/2$
- Result → in the end each node will have computed the average: $sum(x_i)/n$
- If $x_1=1$, and $x_i=0$ → size of network