Communication
Layered Protocols

* Layers, interfaces, and protocols in the OSI model.

```
1. Physical
   - Physical protocol

2. Data link
   - Data link protocol

3. Network
   - Network protocol

4. Transport
   - Transport protocol

5. Session
   - Session protocol

6. Presentation
   - Presentation protocol

7. Application
   - Application protocol
```
Layered Protocols

A typical message as it appears on the network.

- Data link layer header
- Network layer header
- Transport layer header
- Session layer header
- Presentation layer header
- Application layer header

Message

Data link layer trailer

Bits that actually appear on the network
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>
</tbody>
</table>
| 2    | Data 1 | Control 0 | A sends data message 1  
B complains about the checksum |
| 3    | Control 0 | Data 1 | Both messages arrive correctly          |
| 4    | Data 0 | Control 1 | A retransmits data message 0  
B says: "I want 0, not 1"  |
| 5    | Control 1 | Data 0 | Both messages arrive correctly          |
| 6    | Data 0 |        | A retransmits data message 0 again       |
| 7    |        | Data 0 | B finally gets message 0                 |
Middleware Protocols

- An adapted reference model for networked communication.
Types of Communications

- **Persistent communications**
  - Message that has been submitted is stored by the system as long as it takes to deliver it to its receiver

- **Transient communications**
  - Message is stored by the system only as long as the sending and receiving applications are executing

- **Synchronous communication**

- **Asynchronous communication**
Remote Procedure Call (RPC)

- Birrell and Nelson suggested that programs should be able to call procedures to executed on remote machines
Conventional Procedure Call

- Parameter passing in a local procedure call
  - (a) the stack before the call to read
  - (b) the stack while the called procedure is active

Passing parameters by value, reference, and copy/restore
Client and Server Stubs

- Principle of RPC between a client and server program.
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

Steps involved in doing remote computation through RPC

1. Client call to procedure
2. Stub builds message
3. Message is sent across the network
4. Server OS hands message to server stub
5. Stub unpacks message
6. Stub makes local call to "add"
Passing Value Parameters

a) Original message on the Pentium (Little Endian)
b) The message after receipt on the SPARC (Big Endian)
c) The message after being inverted. The little numbers in boxes indicate the address of each byte

```
(a)  0  0  0  5
    7  6  5  4
    L  L  I  J

(b)  0  1  2  3
    5  0  0  0
    4  5  6  7
    J  I  L  L

(c)  0  1  2  3
    0  0  0  5
    4  5  6  7
    L  L  I  J
```
Parameter Specification and Stub Generation

a) A procedure
   b) The corresponding message.

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

(a) (b)

foobars local variables

- x
- y
- 5
- z[0]
- z[1]
- z[2]
- z[3]
- z[4]
Asynchronous RPC

a) The interconnection between client and server in a traditional RPC

b) The interaction using asynchronous RPC

(a) Client Wait for result
   Call remote procedure
   Request
   Return from call
   Reply
   Call local procedure and return results
   Time

(b) Client Wait for acceptance
   Call remote procedure
   Request
   Accept request
   Return from call
   Call local procedure
   Time
Asynchronous RPC

A client and server interacting through two asynchronous RPCs (deferred synchronous RPCs)
Writing a Client and a Server

The steps in writing a client and a server in DCE RPC using the IDL compiler
Binding a Client to a Server in DCE

1. Register endpoint
2. Register service
3. Look up server
4. Ask for endpoint
5. Do RPC
Persistence and Synchronicity in Communication

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

General organization of a communication system in which hosts are connected through a network
Persistence and Synchronicity in Communication

a) Persistent asynchronous communication

b) Persistent synchronous communication

Diagram:
- A sends message and continues
- A stops running
- B starts and receives message
- B is not running

Diagram:
- A sends message and waits until accepted
- Message is stored at B's location for later delivery
- Accepted
- Time
- B starts and receives message
- B is not running
Persistence and Synchronicity in Communication

c) Transient asynchronous communication

d) Receipt-based transient synchronous communication
Persistence and Synchronicity in Communication

e) Delivery-based transient synchronous communication at message delivery

f) Response-based transient synchronous communication
**Berkeley Sockets**

Socket primitives for TCP/IP.

<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>
Berkeley Sockets

- Connection-oriented communication pattern using sockets.

![Berkeley Sockets Diagram]

- Server:
  - socket → bind → listen → accept → read → write → close

- Client:
  - socket → connect → write → read → close

  Synchronization point

  Communication
**The Message-Passing Interface (MPI)**

- MPI is suitable for parallel applications with transient communications where failures are fatal.
- Some of the most intuitive message-passing primitives of 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>
Message-Queuing Model

- Four combinations for loosely-coupled communications using queues.

Sender running  
Receiver running  
(a)

Sender running  
Receiver passive  
(b)

Sender passive  
Receiver running  
(c)

Sender passive  
Receiver passive  
(d)
Message-Queuing Model

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

<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>
The relationship between queue-level addressing and network-level addressing.
The general organization of a message-queuing system with routers
Message Brokers in Message-Queuing Systems
Example: IBM MQSeries

- General organization of IBM's MQSeries message-queueing system.
- Message Channel Agents (MCA) manage the message channel (pairwise reliable connection) between a sending and receiving Queue Manager.
**Message Channels**

Some attributes associated with message channel agents.

<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>
Message Transfer

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

Primitives available in an IBM MQSeries MQI

<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>
Communication for Continuous Media

- In continuous media the temporal relationships between the data elements is fundamental to correctly interpreting them (e.g. Audio and video)
  - Not so in discrete media (e.g. text, imagery)
- Data stream = sequence of data elements
  - Simple vs complex streams
- Timing is crucial in continuous data streams
- Transmission modes for data streams
  - Asynchronous
  - Synchronous
  - Isochronous
Continuous Data Streams

- Setting up an audio stream between two processes across a network
- Sampling rate of 44.4KHz with 16-bits per sample for Pulse Code Modulation (PCM)
Continuous Data Streams

- Setting up a stream directly between two devices.
Continuous Data Streams

- An example of multicasting a stream to several receivers.
## Specifying Quality of Service

A flow specification.

<table>
<thead>
<tr>
<th>Characteristics of the Input</th>
<th>Service Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. maximum data unit size (bytes)</td>
<td>1. Loss sensitivity (bytes)</td>
</tr>
<tr>
<td>2. Token bucket rate (bytes/sec)</td>
<td>2. Loss interval (µsec)</td>
</tr>
<tr>
<td>3. Token bucket size (bytes)</td>
<td>3. Burst loss sensitivity (data units)</td>
</tr>
<tr>
<td>4. Maximum transmission rate (bytes/sec)</td>
<td>4. Minimum delay noticed (µsec)</td>
</tr>
<tr>
<td></td>
<td>5. Maximum delay variation (µsec)</td>
</tr>
<tr>
<td></td>
<td>6. Quality of guarantee</td>
</tr>
</tbody>
</table>
**Specifying Quality of Service**

- **Typical QoS parameters**
  - Required bit rate
  - Maximum latency for sending data
  - Maximum end-to-end delay
  - Maximum delay variance (jitter)
  - Maximum round-trip delay

- **IP is a best-effort datagram service!**

- **Challenge:** provide support for continuous media using minimal support from the network/transport layers
Support for QoS

- Buffer enough data elements before delivering the 1st data element to the application, so that the QoS requirements are satisfied.

- Since IP is unreliable, packets many be lost, and since the media have high bit rates, request for retransmissions are not an option.
  - Use forward error correcting coding.
  - Use packet interleaving so that a single packet loss does not affect a long sequence of frames (data elements).
Synchronization Mechanisms

- The principle of explicit synchronization on the level data units.
- Video data unit = 1 frame, audio data unit = samples for duration of video frame

Diagram:
- Receiver's machine
- Application
- OS
- Procedure that reads two audio data units for each video data unit
- Incoming stream
- Network
Synchronization Mechanisms

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

- A group of nodes organize in an overlay network which they use to disseminate information to all its members.
- Example: Using CHORD construct a tree connecting the group using intermediate nodes as forwarders.
- Overlay construction is a graph embedding problem.
- Quality of embedding is measured by:
  - Link stress (congestion) = #overlay links that use a physical link
  - link dilation = length of physical path corresponding to overlay link
  - stretch = ratio of length of overlay path over the length of the corresponding physical path
  - Tree cost = total cost of physical links used by the links in the overlay tree
Overlay Construction

- Assume the existence of a rendezvous node that knows all the members of the group
- Each node links to the source
  - Stretch is 1, stress is high
- Switch-trees
  - A node can choose another parent (not one of its descendant) that is not overloaded
  - Some criteria for choosing new parent:
    - Reduce delay to source
    - New parent is closer than old parent
  - Node currently switching denies all other switch sequests
Gossip-based dissemination

Utilize ideas from spread of diseases to design protocols for disseminating updates to data elements

- Partition nodes into Infected (have data), susceptible (willing to spread data), and removed (non-cooperating) nodes

- Anti-entropy propagation model: each node P picks a random node Q and exchanges updates with Q
  - P only pushes updates to Q
  - P only pulls updates from Q
  - P pushes and pulls updates to/from Q
  - $O(\log N)$ rounds are sufficient to disseminate information to N nodes

Rumor spreading (gossiping)

- P pushes updates to Q; if Q was already infected, P stops being susceptible with some probability
Gossip-based dissemination

- Directional gossiping: nodes with few neighbors have higher probability of being selected (infected)
- Dealing with deletes
  - Utilize “death certificates”
- How do you select a node at random?
- Applications: computing the average
  - Suppose every node I has a value $x_i$
  - When gossiping nodes I and J replace their own values with the average of their values
  - After $O(\log N)$ rounds, everybody knows the average
  - Estimate the number of nodes in the system by having only a single node to have value 1, with all other nodes having value 0.