Foundations of Interprocess Communication

Layered Protocols

Due to the absence of shared memory, all communication in distributed systems is based on sending and receiving (low level) messages. When a process P wants to communicate with process Q, it first builds a message in its own address space. Then it executes a system call that causes the OS to send the message over the network to Q.

The OSI reference model

ISO = **International Standards Organization**
OSI = **Open Systems Interconnection**

Protocols that were developed as part of the OSI model were widely used and are essentially dead.

An open system is 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. These rules are formalized in what are called **communication protocols**. To allow a group of computers to communicate over the network, they must all agree on the protocols to be used. A protocol is said to provide a **communication service**.

Two types of service:

1. In **connection-oriented** service, before exchanging data the sender and receiver first explicitly establish a connection, and possibly negotiate specific parameters of the protocols they will use. When they are done, they release the connection.
2. With **connectionless** services, no setup in advance is needed. The sender just transmits the first message when it is ready. Dropping a letter in a mailbox is an example.

In OSI model, communication is divided into **seven layers** or levels. Each layer offers one or more specific communication services to the layer above it. In this way, problem of getting a message from A to B can be divided into manageable pieces, each of which can be solved independently of each other. Each layer provides an **interface** to the one above it.

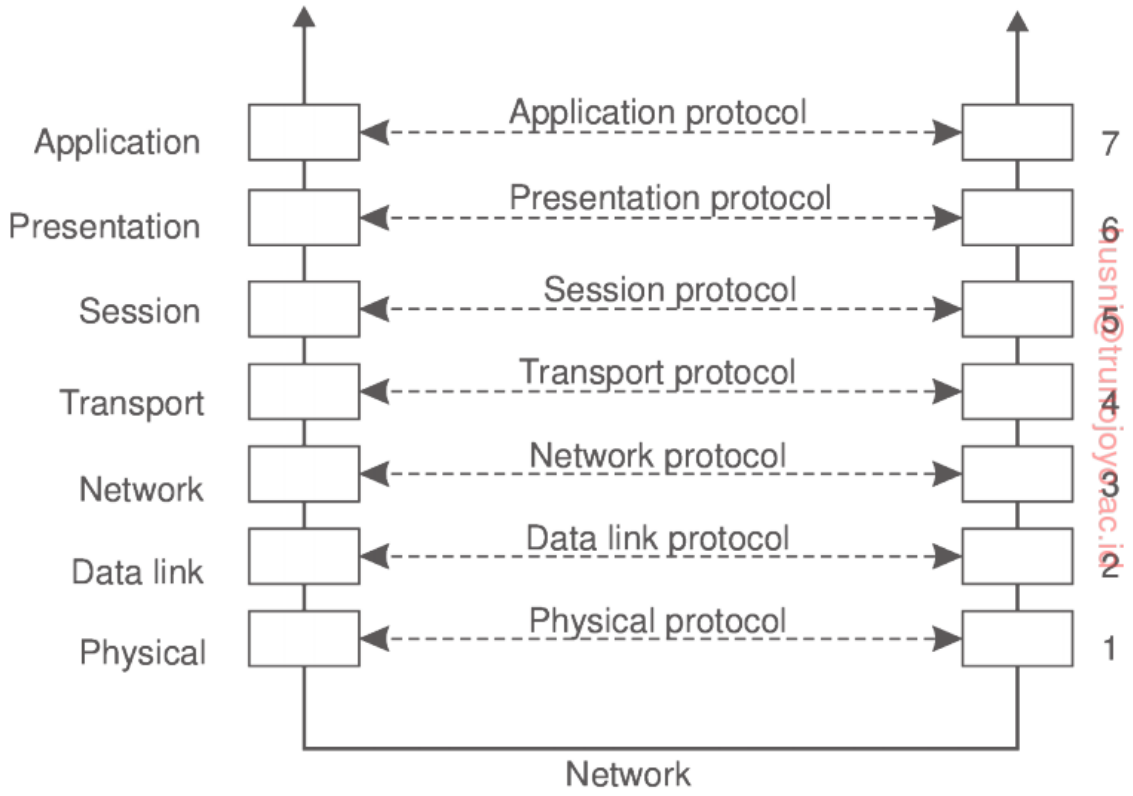


Figure 4.1: Layers, interfaces, and protocols in the OSI model.

The seven OSI layers are:

1. **Physical layer** Deals with standardizing how two computers are connected and how 0s and 1s are represented.
2. **Data link layer** Provides the means to detect and possibly correct transmission errors, as well as protocols to keep the sender and receiver in the same pace.
3. **Network layer** Contains the protocols for routing a message through a computer network, as well as protocols for handling congestion.
4. **Transport layer** Mainly contains protocols for directly supporting applications, such as those that establish reliable communication, or support real-time streaming of data.
5. **Session layer** Provides support for sessions between applications.
6. **Presentation layer** Prescribes how data is represented in a way that is independent of the hosts on which communication applications are running.
7. **Application layer** Essentially, everything else: e-mail protocols, Web access protocols, file-transfer protocols, and so on.

When a process P wants to communicate with a remote process Q, it builds a message and passes that message to the application layer as offered to it by means of an interface. This interface will typically appear in the form of a library procedure. The application layer software then adds a **header** to the front of the message and passes the resulting the message across the layer $6/7$ interface to the presentation. The presentation layer, in turn, adds it own header and passes the result down to the session layer, and so on. Some layers add not only a header to the front, but also a trailer to the end.

When it hits the bottom, the physical layer actually transmits the message by putting it to the physical transmission medium.

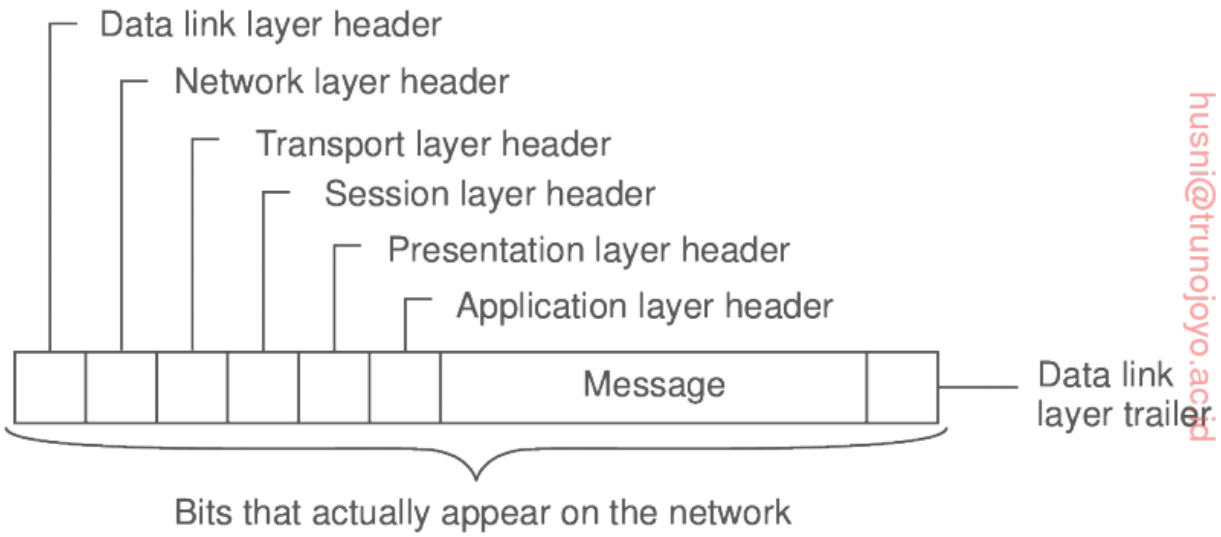


Figure 4.2: A typical message as it appears on the network.

When the message arrives at the remote machine hosting Q, it is passed upward, with each layer stripping off and examining its own header. Finally, the message arrives at the reciever, process Q, which may reply to it using the reverse path. the information in the layer-n header is used for the layer-n protocol.

The collection of protocols used in a particular system is called a **protocol suite** or **protocol stack**.

Middleware protocols

Middleware is an application that logically lives (mostly) in OSI application layer, but which contains many general-purpose protocols that warrant their own layers, independent of other, more specific applications. Mostly contains application-independent protocols.

Example: **DNS** is an application and therefore is logically placed in the application layer. However, it should be quite obvious that DNS is offering a general-purpose, application-independent service. Arguably, it forms a part of the middleware.

Other examples: Authentication protocols, distributed commit (transaction based) protocols providing **atomicity**, distributed locking protocols.

Adapted reference model

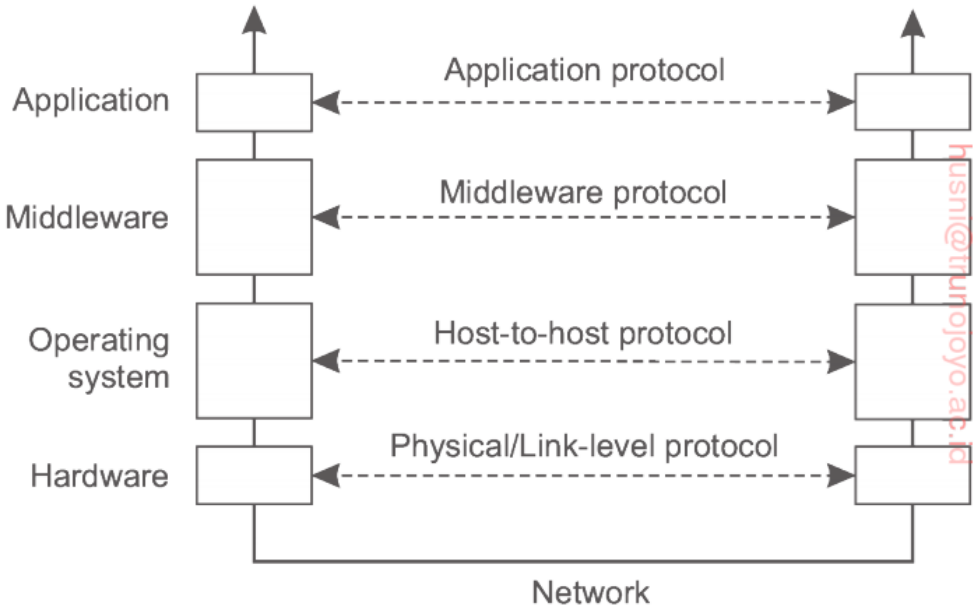
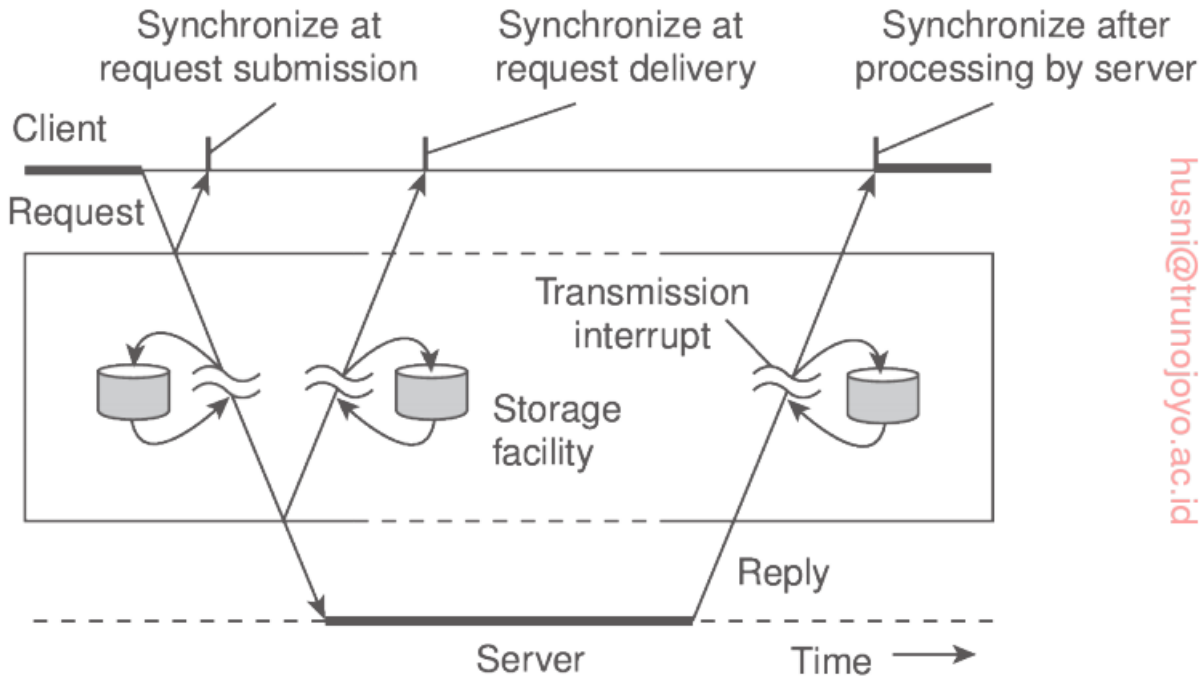


Figure 4.3: An adapted reference model for networked communication.

Types of Communication

With **persistent communication**, a message that has been submitted for transmission is stored by the communication middleware as long as it takes to deliver it to the receiver. The middleware will store the message at one or several of the storage facilities. As a consequence, it is not necessary for the sending application to continue executing after submitting the message. Likewise, recieving application need not be executing when the message is submitted.



In constrast, with **transient communication**, a message is stored by the communication system only as long as the sending and receiving application are **executing**. If the middleware cannot deliver a

message due to transmission interrupt, or because the recipient is currently not active, it will simply be discarded. Typcially, all transport-level communication services offer only transient communication. In this case, the communication system consists of traditional store-and-forward routers. If a router cannot deliver a message to the next one or the destination host, it will simply discard the message.

Besides being persistent or transient, communication can also be synchronous and asynchronous.

The characteristic feature of **asynchronous communication** is that a sender continues immediately after it has submitted its message for transmission. This means that the message is (temporarily) stored immediately upon the submission.

With **synchronous communication**, the sender is blocked until its request is known to be accepted. There are essentially 3 points where synchronous can take place:

- The sender may be blocked until the middleware notifies that it will over the transmission of the request.
- The sender may synchonize until its request has been delivered to the intended recipient
- Sender waits until its request has been fully processed, that is, up to the time the recepient returns a response.