Sockets

– Both forms of communication (UDP and TCP) use the socket abstraction, which provides an endpoint for communication between processes. Sockets originate from BSD UNIX but are also present in most other versions of UNIX, including Linux as well as Windows and the Macintosh OS. Interprocess communication consists of transmitting a message between a socket in one process and a socket in another process, as illustrated in Figure 1.

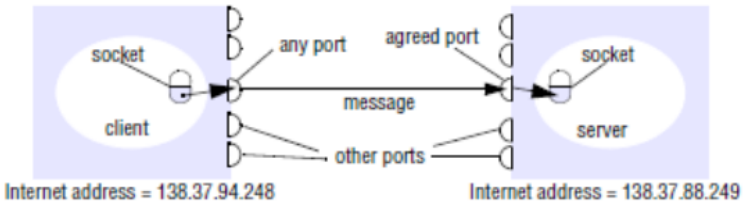

Figure 1: Sockets and ports

– For a process to receive messages, its socket must be bound to a local port and one of the Internet addresses of the computer on which it runs. Messages sent to a particular Internet address and port number can be received only by a process whose socket is associated with that Internet address and port number. Processes may use the same socket for sending and receiving messages. Each computer has a large number (2^{16}) of possible port numbers for use by local processes for receiving messages. Any process may make use of multiple ports to receive messages, but a process cannot share ports with other processes on the same computer. However, any number of processes may send messages to the same port. Each socket is associated with a particular protocol – either UDP or TCP.

– A socket passes through different states. Each state marks an event. It is the state of the socket that tells what a socket can do and it cannot do. Generally, a socket's lifecycle is described by eight primitives listed in Table 1 below:

Table 1: List of Typical Socket Primitives

Primitive	Description
Socket	Creates a socket, which is used by an application to serve as a communication end point.
Bind	Associates a local address to the socket. The local address includes an IP address and a port number. The port number must be a number between 0 and 65535. It should be unique for the protocol being used for the socket on the computer. For example, is a TCP socket uses port 12345, a UDP socket can also use the same port number 12345.
Listen	Defines the size of its wait queue for a client request. It is performed only by a connection-oriented server socket.
Accept	Waits for a client request to arrive. It is performed by a connection-oriented server socket.

Connect	Attempts to establish a connection to a server socket, which is waiting on an accept primitive. It is performed by a connection – oriented client socket.
Send/Sendto	Sends data. Usually send indicates a Send operation on a connection – oriented socket and Sendto indicates a send operation on a connectionless socket.
Receive/ReceiveFrom	Receives data. They are counterparts of Send and Sendto.
Close	Closes a connection

- There are two kinds of sockets, namely;

    i. A connection-oriented socket, and
    ii. A connection-less socket.

- A connection-oriented socket is also called a stream socket. A connectionless socket is also called a datagram socket. Note that data is always sent one datagram at a time from one host to another on the Internet using IP datagrams. Transmission Control Protocol (TCP), is one of the most widely used protocols to provide connection-oriented sockets. Whereas, the User Datagram Protocol (UDP), is one of the most widely used protocols to provide connectionless sockets.

- In a connection-oriented socket communication, the client and the server create a socket at their ends, establish a connection, and exchange information. TCP takes care of the errors that may occur during data transmission. TCP is also known as a reliable transport level protocol because it guarantees the delivery of the data. If it could not deliver the data for some reasons, it will inform the sender application about the error conditions. After it sends the data, it waits for an acknowledgment from the receiver to make sure that the data reached the destination. However, the reliability that TCP offers comes at a price. The overhead as compared to a connectionless protocol is much more significant, and it is slower. TCP makes sure that a sender sends the amount of data to the receiver, which can be handled by the receiver's buffer size. It also handles traffic congestion over the network. It slows down the data transmission when it detects traffic congestion. Java supports TCP sockets.

- User Datagram Protocol (UDP), which is used in a transport layer, is the most widely used protocol that provides a connectionless socket. It is unreliable, but much faster. It lets you send limited sized data—one packet at a time, which is different from TCP, which lets you send data as a stream of any size, handling the details of segmenting them in appropriate size of packets. Data delivery is not guaranteed when you send data using UDP. However, it is still used in many applications and it works very well. The sender sends a UDP packet to a destination and forgets about it. If receiver gets it, it gets it. Otherwise, there is no way to know—for the receiver—that there was a UDP packet sent to it. You can compare the communication used in TCP and UDP to the communication used in a telephone and mailing a letter. A telephone conversation is reliable and it offers acknowledgment between two parties that are communicating. When you mail a letter, you do not know when the addressee receives it, or if he received it at all. There is another important difference between UDP and TCP. UDP does not guarantee the ordering of data. That is, if you send five packets to a destination using UDP, those five packets may arrive in any order. However, TCP guarantees that packets will be delivered in the order they were sent. Java supports UDP sockets.

The TCP Stream Communication

- The API to the TCP protocol, which originates from BSD 4.x UNIX, provides the abstraction of a

stream of bytes to which data may be written and from which data may be read. The following characteristics of the network are hidden by the stream abstraction:

Message sizes: The application can choose how much data it writes to a stream or reads from it. It may deal in very small or very large sets of data. The underlying implementation of a TCP stream decides how much data to collect before transmitting it as one or more IP packets. On arrival, the data is handed to the application as requested. Applications can, if necessary, force data to be sent immediately.

Lost messages: The TCP protocol uses an acknowledgement scheme. As an example of a simple scheme (which is not used in TCP), the sending end keeps a record of each IP packet sent and the receiving end acknowledges all the arrivals. If the sender does not receive an acknowledgement within a timeout, it retransmits the message. The more sophisticated sliding window scheme [Comer 2006] cuts down on the number of acknowledgement messages required.

Flow control: The TCP protocol attempts to match the speeds of the processes that read from and write to a stream. If the writer is too fast for the reader, then it is blocked until the reader has consumed sufficient data.

Message duplication and ordering: Message identifiers are associated with each IP packet, which enables the recipient to detect and reject duplicates, or to reorder messages that do not arrive in sender order.

Message destinations: A pair of communicating processes establish a connection before they can communicate over a stream. Once a connection is established, the processes simply read from and write to the stream without needing to use Internet addresses and ports. Establishing a connection involves a connect request from client to server followed by an accept request from server to client before any communication can take place. This could be a considerable overhead for a single client-server request and reply.

– The API for stream communication assumes that when a pair of processes are establishing a connection, one of them plays the client role and the other plays the server role, but thereafter they could be peers. The client role involves creating a stream socket bound to any port and then making a connect request asking for a connection to a server at its server port. The server role involves creating a listening socket bound to a server port and waiting for clients to request connections. The listening socket maintains a queue of incoming connection requests. In the socket model, when the server accepts a connection, a new stream socket is created for the server to communicate with a client, meanwhile retaining its socket at the server port for listening for connect requests from other clients.

– The pair of sockets in the client and server are connected by a pair of streams, one in each direction. Thus each socket has an input stream and an output stream. One of the pair of processes can send information to the other by writing to its output stream, and the other process obtains the information

by reading from its input stream.

– When an application closes a socket, this indicates that it will not write any more data to its output stream. Any data in the output buffer is sent to the other end of the stream and put in the queue at the destination socket, with an indication that the stream is broken. The process at the destination can read the data in the queue, but any further reads after the queue is empty will result in an indication of end of stream. When a process exits or fails, all of its sockets are eventually closed and any process attempting to communicate with it will discover that its connection has been broken.

– The following are some outstanding issues related to stream communication:

Matching of data items: Two communicating processes need to agree as to the contents of the data transmitted over a stream. For example, if one process writes an int followed by a double to a stream, then the reader at the other end must read an int followed by a double. When a pair of processes do not cooperate correctly in their use of a stream, the reading process may experience errors when interpreting the data or may block due to insufficient data in the stream.

Blocking: The data written to a stream is kept in a queue at the destination socket. When a process attempts to read data from an input channel, it will get data from the queue or it will block until data becomes available. The process that writes data to a stream may be blocked by the TCP flow-control mechanism if the socket at the other end is queuing as much data as the protocol allows.

Threads: When a server accepts a connection, it generally creates a new thread in which to communicate with the new client. The advantage of using a separate thread for each client is that the server can block when waiting for input without delaying other clients. In an environment in which threads are not provided, an alternative is to test whether input is available from a stream before attempting to read it; for example, in a UNIX environment the select system call may be used for this purpose.

Failure model

– To satisfy the integrity property of reliable communication, TCP streams use checksums to detect and reject corrupt packets and sequence numbers to detect and reject duplicate packets. For the sake of the validity property, TCP streams use timeouts and retransmissions to deal with lost packets. Therefore, messages are guaranteed to be delivered even when some of the underlying packets are lost.

– But if the packet loss over a connection passes some limit or the network connecting a pair of communicating processes is severed or becomes severely congested, the TCP software responsible for sending messages will receive no acknowledgements and after a time will declare the connection to be broken. Thus TCP does not provide reliable communication, because it does not guarantee to deliver messages in the face of all possible difficulties.

– When a connection is broken, a process using it will be notified if it attempts to read or write. This has the following effects:

- The processes using the connection cannot distinguish between network failure and failure of

the process at the other end of the connection.

- The communicating processes cannot tell whether the messages they have sent recently have been received or not.

Use of TCP

– Many frequently used services run over TCP connections, with reserved port numbers. These include the following:

HTTP: The Hypertext Transfer Protocol is used for communication between web browsers and web servers;

FTP: The File Transfer Protocol allows directories on a remote computer to be browsed and files to be transferred from one computer to another over a connection.

Telnet: Telnet provides access by means of a terminal session to a remote computer.

SMTP: The Simple Mail Transfer Protocol is used to send mail between computers.