Problem Statement:

Write a program using TCP sockets for wired network to implement
a. Peer to Peer Chat
b. Multiuser Chat

Theory:

- **Introduction :**

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet protocol suite (IP), and is so common that the entire suite is often called TCP/IP. TCP provides reliable, ordered, error-checked delivery of a stream of octets between programs running on computers connected to a local area network, intranet or the public Internet. It resides at the transport layer. TCP provides a connection oriented service, since it is based on connections between clients and servers. TCP provides reliability. When a TCP client send data to the server, it requires an acknowledgement in return. If an acknowledgement is not received, TCP automatically retransmit the data and waits for a longer period of time. TCP is instead a byte-stream protocol, without any boundaries at all.

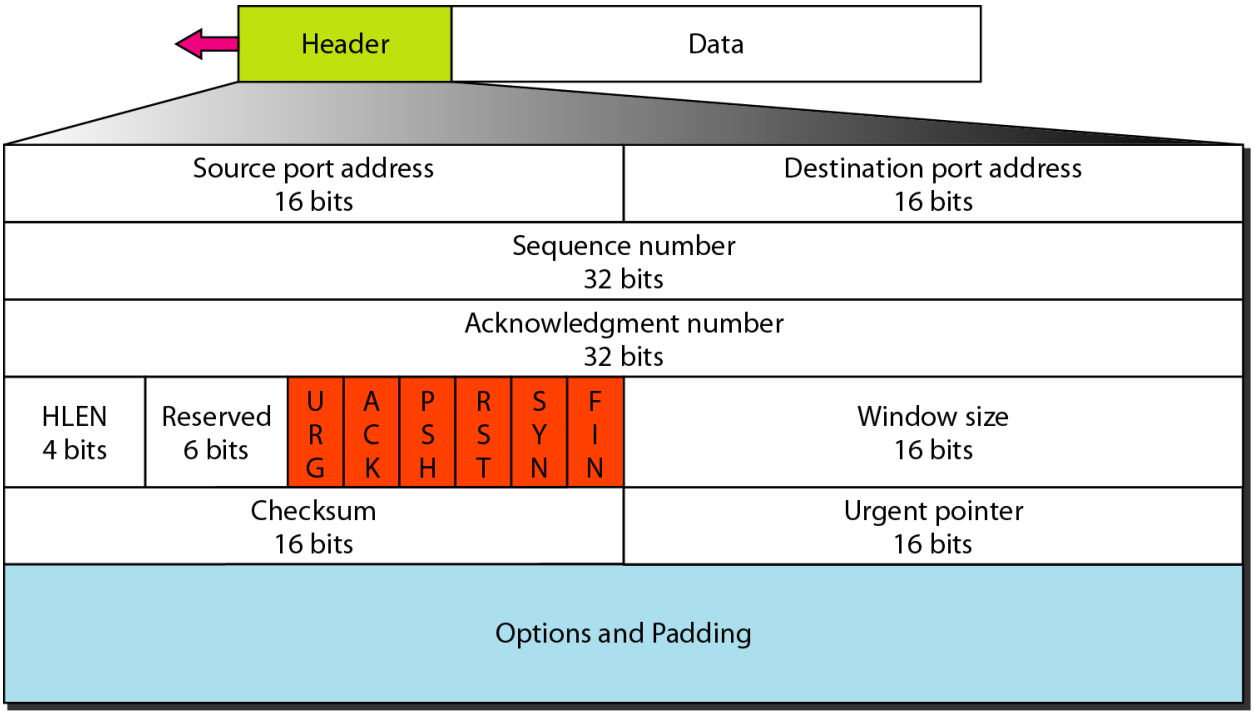

Figure 1: TCP header (Header size is 20 bytes if no options

Header size is 60 bytes if option present)

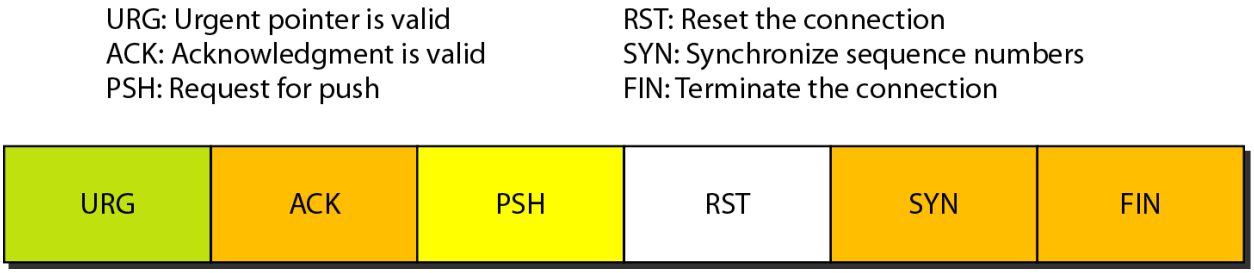

Figure 2: Control fields

Flag	*Description*
URG	The value of the urgent pointer field is valid.
ACK	The value of the acknowledgment field is valid.
PSH	Push the data.
RST	Reset the connection.
SYN	Synchronize sequence numbers during connection.
FIN	Terminate the connection.

- **TCP Features:**

1) Flow Control

TCP, unlike UDP, provides flow control. The receiver of the data controls the amount of data that are to be sent by the sender. This is done to prevent the receiver from being overwhelmed with data. The numbering system allows TCP to use a byte-oriented flow control.

2) Error Control

To provide reliable service, TCP implements an error control mechanism. Although error control considers a segment as the unit of data for error detection (loss or corrupted segments), error control is byte-oriented, as we will see later.

3) Congestion Control

TCP, unlike UDP, takes into account congestion in the network. The amount of data sent by a sender is not only controlled by the receiver (flow control), but is also determined by the level of congestion in the network.

- **Socket Programming**

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

The **java.net.Socket** class represents a socket, and the **java.net.ServerSocket** class provides a mechanism for the server program to listen for clients and establish connections with them.

The following steps occur when establishing a TCP connection between two computers using sockets −

1) The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
2) The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
3) After the server is waiting, a client instantiates a Socket object, specifying the server name and the port number to connect to.
4) The constructor of the Socket class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server.
5) On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.

- **Functions used:**

1) **public ServerSocket(int port) throws IOException**

Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.

2) **public Socket accept() throws IOException**

   Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely.

3) **public InputStream getInputStream() throws IOException**

   Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.

4) **public OutputStream getOutputStream() throws IOException**

   Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.

5) **static InetAddress getByAddress(byte[] addr)**

   Returns an InetAddress object given the raw IP address.

6) **static InetAddress getByName(String host)**

   Determines the IP address of a host, given the host's name.

7) **public void close() throws IOException**

   Closes the socket, which makes this Socket object no longer capable of connecting again to any server.

Conclusion: