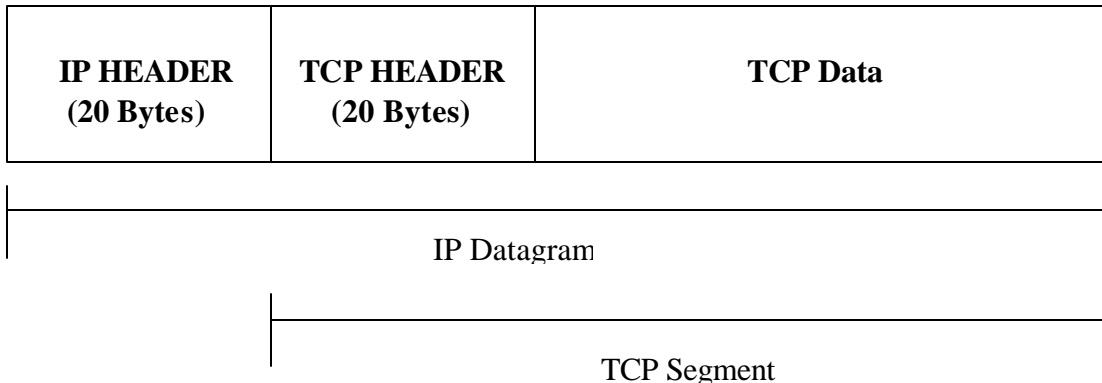


Transmission Control Protocol (TCP)



TCP provides a connection-oriented, reliable, byte stream service.

TCP vs UDP

- TCP provides automatic error control
- The IP layer gives no guarantee that datagrams will be delivered properly; it is up to TCP to timeout and retransmit as needed.
- TCP provides reliability of service; datagram packets that were received out of sequence will be reassembled in the right order to form the stream of bytes.
- TCP is used by applications such as Telnet, Rlogin, FTP, and SMTP.

TCP Service Model

- TCP service is obtained by creating end points called sockets. Each socket consists of a socket number and port number.
- A single daemon (called the *inetd* in Unix) waits on multiple ports for an incoming connection.

- Some well-known ports are 21(FTP), 23(telnet), 25(SMTP), 79(finger), and 80(HTTP).
- TCP connections are full-duplex and point-to-point.
- TCP connection is byte stream not message stream.
- TCP segment size is restricted by the network's Maximum Transfer Unit(MTU), generally 1500 bytes.
- Uses the sliding window protocol.

The TCP Header

Source Port				Destination Port				
Sequence Number								
Acknowledgement Number								
Header Length		U R G	A C K	P S H	P S T	S Y N	F I N	Window Size
Checksum				Urgent Pointer				
Options (or or more 32 bit words)								

Options (or or more 32 bit words)

Options (or or more 32 bit words)
--

- a port + IP address forms a unique 48-bit end-point
- Sequence number is used to mark the first byte; the acknowledgement number is the next byte that is expected. Note that each byte is marked with sequence number.
- Header Length tells how many 32-bit words are in the header.
- Following the Header Length is an unused 6 bit field
- URG is the urgent pointer (set to 1 if used). Indicates a byte offset from the current sequence number at which urgent data are to be found. Typically not used.

- ACK bit is set to 1 to indicate that the acknowledgment number is valid. The value 0 means don't use the acknowledgement number.
- PSH indicates PUSHed data; i.e. a request to the receiver to deliver the received data to the application and not buffer it.
- RST is used to reset the connection.
- SYN is used to establish connections. A connection request will typically have SYN=1 and ACK=0. A connection reply carries SYN=1 and ACK=1
- FIN is used to teardown a connection.
- The window size tells how many bytes may be sent starting at the byte acknowledged. Primarily this is used for flow control. Recall that TCP uses sliding window protocol.
- Checksum is used for error detection.
- The options are typically used for some other information not included in the standard header fields. Such as options may include information on how much TCP payload a host is willing to receive and how much to back track during retransmissions.

TCP Sockets in Java

- 1) `java.net.Socket` – provides the API for the client application
- 2) `java.net.ServerSocket` – provides the API for the server application

The java.net.Socket class

Constructor methods:

protected Socket() – creates an unconnected socket

Socket (***InetAddress address, int numPort***) throws java.io.IOException, java.lang.SecurityException – creates a socket connected to ***address*** in port ***numPort***.

Socket (***InetAddress address, int numPort, InetAddress localAddress, int localPort***) throws java.io.IOException, java.lang.SecurityException – creates a socket connected to address in port numPort and is bound to the specified local address in the local port.

Socket (***String hostName, int numPort***) throws java.net.UnknownHostException, java.io.IOException, java.lang.SecurityException – creates a socket connected to a specified hostname and port.

Socket (***String hostName, int numPort, InetAddress localAddress, int localPort***) throws java.net.UnknownHostException, java.io.IOException, java.lang.SecurityException – creates a socket connected to a specified hostName in port numPort and is bound to the specified local address in the local port.

Note: When a socket is instantiated, a connection is automatically established. Although, at times, the constructor method may block waiting for the connection to get through.

Socket class methods:

void close() throws java.io.IOException – closes the socket connection.

InetAddress getInetAddress()

InetAddress getLocalAddress()

– returns the remote (local) address that is connected to (used by) the socket.

InputStream getInputStream() throws java.io.IOException

OutputStream getOutputStream() throws

java.io.IOException

– returns an input (output) stream, which reads from (writes to) the application the socket is connected to.

int getLocalPort()

int getPort()

– returns the port number that the socket is bound (connected) to in the local (remote) machine.

int getReceiveBufferSize() throws java.net.SocketException

void setReceiveBufferSize(int size) throws

java.net.SocketException

– returns (sets) the receive buffer size used by the socket. It is advisable to use the `BufferedInputStream` or the `BufferedReader` to buffer incoming TCP data.

int getSendBufferSize() throws java.net.SocketException

void setSendBufferSize(int size) throws

java.net.SocketException

- returns (sets) the send buffer size used by the socket.

int getSoLinger() throws java.net.SocketException
void setSoLinger(boolean IFlag, int seconds) throws
java.net.SocketException, java.lang.IllegalArgumentException
- returns (sets) the value of the SO_LINGER socket option, which controls how long will the close() method blocks to allow the remaining data to be sent and acknowledged.

int getSoTimeout() throws java.net.SocketException
void setSoTimeout(int milliseconds) throws
java.net.SocketException
- returns (sets) the value of the SO_TIMEOUT socket option, which controls how long the call to read() will have block to get the necessary bytes of data.

boolean getTcpNoDelay() throws java.net.SocketException
void setTcpNoDelay(boolean onFlag) throws
java.net.SocketException
-returns (sets) the value of the TCP_NODELAY socket option. Value indicates whether Nagle's algorithm is enabled.

A Simple TCP Client Example

```
package NetProg;
import java.net.*;
import java.io.*;
public class TCPClient {
    public static void main(String [] args) {
        InputStreamReader in = new InputStreamReader(System.in);
        BufferedReader bufR = new BufferedReader(in);
        String message = "";
        while (true) {
            try {
                //create a socket and connect
                Socket socket = new Socket("205.174.40.142",4400);

                System.out.print("Enter a MESSAGE: ");
                message = bufR.readLine();

                socket.setSendBufferSize(message.length());

                //connect to a buffered reader
                BufferedOutputStream bos = new
                    BufferedOutputStream(socket.getOutputStream(),
                        socket.getSendBufferSize());
                for (int i=0; i<message.length(); i++){
                    bos.write((int) message.charAt(i));}

                bos.close();

                if (message.equals("bye")){
                    System.out.println("Closing.... goodbye!");
                    break;
                }

                socket.close();
            } //end of try
            catch (Exception e) {
                System.out.println("Error " + e.getMessage());
            }
        } //end of while
    } //end of main
} //end of class definition
```

A Simple TCP Server Example

```
package NetProg;
import java.net.*;
import java.io.*;
public class TCPServer {
    public static void main(String [ ] args) {
        String message;
        try {
            //create a socket and connect
            ServerSocket server=new ServerSocket(4400);
            while (true){
                Socket socket = server.accept();
                //connect to a buffered reader
                BufferedInputStream binst = new
                    BufferedInputStream(socket.getInputStream(),
                    socket.getReceiveBufferSize());
                System.out.print("\n RECEIVED MESSAGE: ");
                message = "";
                for (int i=0;i<socket.getReceiveBufferSize();i++) {
                    int data = binst.read();
                    if (data == -1) break;
                    else{
                        System.out.print ((char) data) ;
                        message += (char) (data);
                    }
                }
                //end of for
                message = message.trim();
                socket.close();
                if (message.equals("bye")){
                    System.out.println("\n Closing ... goodbye!");
                    break;
                }
            }
        }
        catch (Exception e) {
            System.out.println("Error " + e.getMessage());
        }
    }
}
```