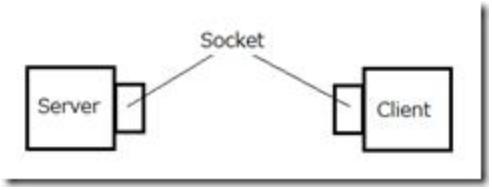
Sockets

Imagine a socket as a seaport that allows a ship to unload and gather shipping, whereas socket is the place where a computer gathers and puts data into the internet.



The Internet Connection basically connects two points across the internet for data sharing and other stuff. One process from computer C1 can communicate to a process from computer C2, over an internet connection. It has following properties:

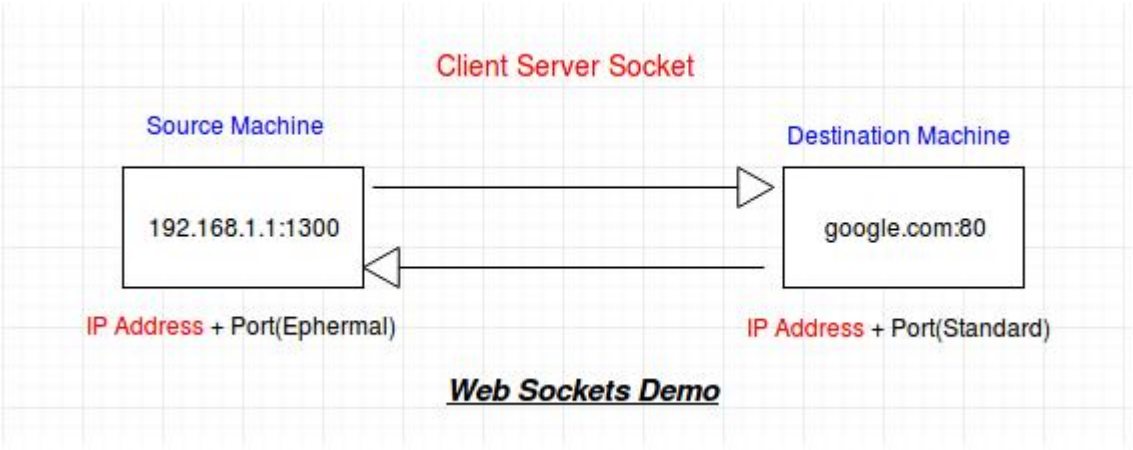
Reliable: It means until the cables connecting two computers are safe, data will be transferred safely.

Point-to-Point: Connection is established between 2 points.

Full-Duplex: It means transfer of information can occur in both ways i.e. from client to server as well as server to client simultaneously (at the same time).

Sockets are the endpoints of a bidirectional, point-to-point communication channel. Given an internet connection, say between client (a browser) and the server, we will have two sockets. A Client Socket and a Server Socket.

Socket acts on two parts: IP Address + Port Number



Socket Types

There are four types of sockets available to the users. The first two are most commonly used and the last two are rarely used.

Processes are presumed to communicate only between sockets of the same type but there is no restriction that prevents communication between sockets of different types.

Stream socket:
>Connection oriented
>>reliable connection
>>delivery is guaranteed
>Sequence is ensured
:practically data is not received in the same sequence; rather, each packet has a sequence number
>TCP

- **Stream Sockets** − Delivery in a networked environment is guaranteed. If you send through the stream socket three items "A, B, C", they will arrive in the same order − "A, B, C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries.

Datagram socket
>Connectionless
>>unreliable connection
>UDP

- **Datagram Sockets** − Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets − you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).

Raw socket
>used when wants to change any feature of the underlying protocol (ie. changing data size of TCP)

- **Raw Sockets** − These provide users access to the underlying communication protocols, which support socket abstractions. These sockets are normally datagram oriented, though their exact characteristics are dependent on the interface provided by the protocol. Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more cryptic facilities of an existing protocol.

- **Sequenced Packet Sockets** − They are similar to a stream socket, with the exception that record boundaries are preserved. This interface is provided only as a part of the Network Systems (NS) socket abstraction, and is very important in most serious NS applications.

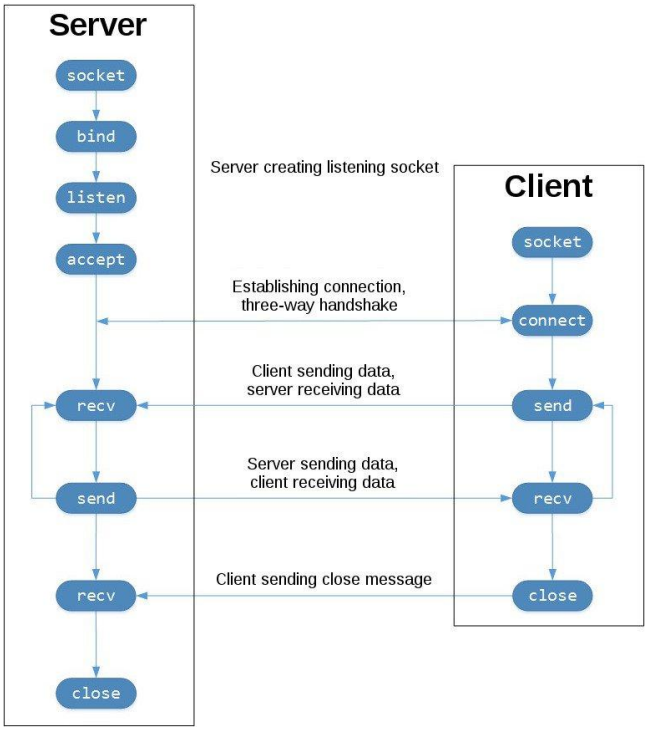
TCP Sockets

>high reliability
>transmission time is relatively less critical
>rearranges data packets in the order specified
>gives absolute guarantee that data transferred remains intact and arrives at the same order
>does flow control
>requires 3 packets to set up socket connection
>handles congestion control

>does error checking and error recovery
>erroneous packets are retransmitted from source to destination

If we are creating a connection between client and server using TCP then it has few functionality like, TCP is suited for applications that require high reliability, and transmission time is relatively less critical. It is used by other protocols like HTTP, HTTPs, FTP, SMTP, Telnet. TCP rearranges data packets in the order specified. There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. TCP does Flow Control and requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control. It also does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination.

In the diagram below, let's look at the sequence of socket API calls and data flow for TCP:



The left-hand column represents the server. On the right-hand side is the client.

Starting in the top left-hand column, note the API calls the server makes to setup a "listening" socket:

- socket()
- bind()
- listen()
- accept()

A listening socket does just what it sounds like. It listens for connections from clients. When a client connects, the server calls accept() to accept, or complete, the connection.

The client calls connect() to establish a connection to the server and initiate the three-way handshake. The handshake step is important since it ensures that each side of the connection is reachable in the network, in other words that the client can reach the server and vice-versa. It may be that only one host, client or server, can reach the other.

In the middle is the round-trip section, where data is exchanged between the client and server using calls to send() and recv().

At the bottom, the client and server close() their respective sockets.

The entire process can be broken down into following steps:

TCP Server –

1. using create(), Create TCP socket.
2. using bind(), Bind the socket to server address.
3. using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection
4. using accept(), At this point, connection is established between client and server, and they are ready to transfer data.
5. Go back to Step 3.

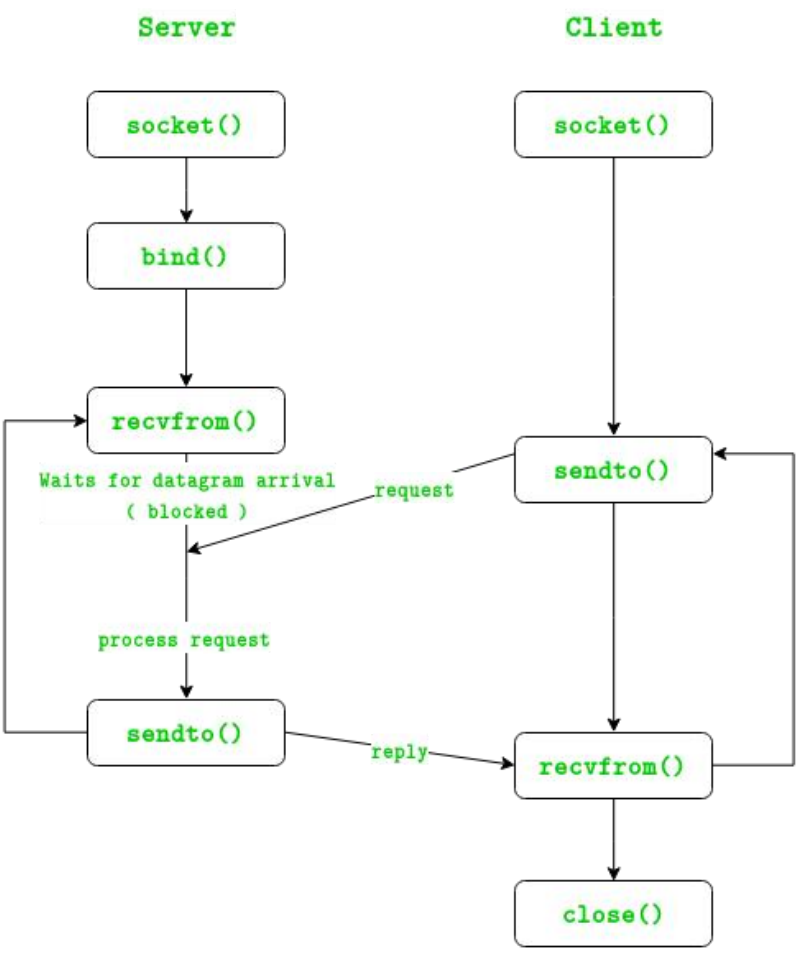
TCP Client –

1. Create TCP socket.
2. connect newly created client socket to server.

UDP

In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of sender which the server uses to send data to the correct client.

The entire process can be broken down into following steps :



UDP Server :

1. Create UDP socket.
2. Bind the socket to server address.
3. Wait until datagram packet arrives from client.
4. Process the datagram packet and send a reply to client.
5. Go back to Step 3.

UDP Client :

1. Create UDP socket.
2. Send message to server.
3. Wait until response from server is recieved.
4. Process reply and go back to step 2, if necessary.
5. Close socket descriptor and exit.

6.