Transport layer

The transport layer is the core of the Internet model. The application layer programs interact with each other using the services of the transport layer. Transport layer provides services to the application layer and takes services from the network layer.

**Duties of Transport Layer:-**

Transport layer is meant for the process to process delivery and it is achieved by performing a number of functions.

Following are the functions of transport layer:-

**1. Packetizing:**

* The transport layer creates packet out of the messages received from the application layer. Packetizing is a process if dividing a long message into smaller ones.
* These packets are then encapsulated into data field of the transport layer packet and headers are added.
* The length of the message which is to be divided can vary from several lines to several pages.

**2. Connection Control:**

* Transport layer protocols maybe divided into following two categories:
* *Connection oriented delivery-*A connection oriented transport layer protocol establishes a connection I.e. virtual path between sender and receiver. This is a virtual connection. The packet may travel out of order. The packets are numbered consecutively and communication is bidirectional.
* *Connectionless delivery-*A connectionless transport protocol will treat each packet independently. There is no connection between them. Each packet can take its own different route.

**3. Addressing:-**

* The client needs the address of the remote computer it wants to communicate with.
* Such a remote computer has a unique address so that it can be distinguished from the other computers.

4. **Providing reliability:-**

* For high reliability the flow of control and error control should be incorporated.
* Flow of control provided by transport layer and is used to perform end to end flow rather than across a single link.
* Error correction is generally achieved through retransmission.

**5. Flow control**

* The transport layer also responsible for the flow control mechanism between the adjacent layers of the TCP/IP model.
* It does not perform across a single link even it performs an end-to-end node.
* By imposing flow control techniques data loss can be prevented from the cause of the sender and slow receiver.
* For instance, it uses the method of sliding window protocol in this method receiver sends a window back to the sender to inform the size of the data is received.

**6. Error Control**

* Error Control is also performed end to end like the data link layer.
* In this layer to ensure that the entire message arrives at the receiving transport layer without any error (damage, loss or duplication). Error Correction is achieved through retransmission of the packet.
* The data has arrived or not and checks for the integrity of data, it uses the ACK and NACK services to inform the sender.

Sockets

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard UNIX file descriptors. In UNIX, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else.

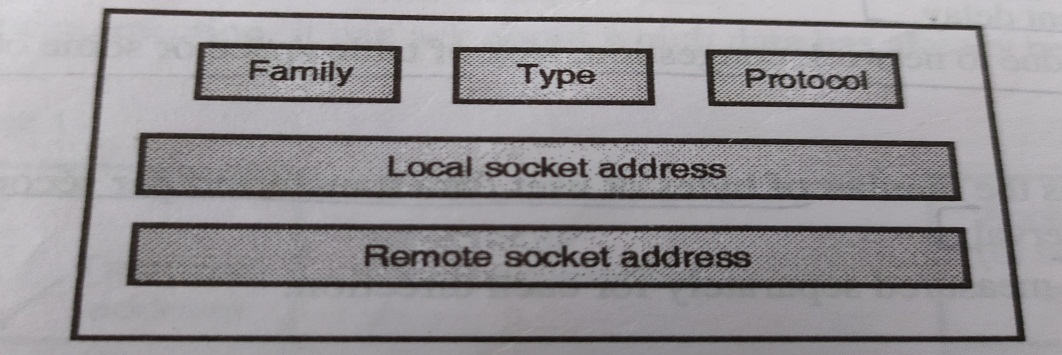
To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes.

Sockets were first introduced in 2.1BSD and subsequently refined into their current form with 4.2BSD. The sockets feature is now available with most current UNIX system releases.

Where is Socket Used?

A Unix Socket is used in a client-server application framework. A server is a process that performs some functions on request from a client. Most of the application-level protocols like FTP, SMTP, and POP3 make use of sockets to establish connection between client and server and then for exchanging data.

**Socket Structure**



**Family-** This field is used for defining the protocol group such as IPV4 or IPV6, UNIX domain protocol.

**Type-** This field is used for defining the types of socket such as stream socket, packet socket or raw socket.

**Protocol-**This field is usually set to zero for TCP and UDP.

**Local Socket address-** It is used for defining the local socket address. This address is a combination of local IP address and the port address.

**Remote Socket address-** It is used for defining the remote socket address.

Socket Types

There are three 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 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 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 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.

### Berkeley sockets

**Berkeley sockets** is an [application programming interface](https://en.wikipedia.org/wiki/Application_programming_interface) (API) for [Internet sockets](https://en.wikipedia.org/wiki/Internet_socket) and [Unix domain sockets](https://en.wikipedia.org/wiki/Unix_domain_socket), used for [inter-process communication](https://en.wikipedia.org/wiki/Inter-process_communication) (IPC). It is commonly implemented as a [library](https://en.wikipedia.org/wiki/Library_(computing)) of linkable modules. It originated with the 4.2BSD [UNIX](https://en.wikipedia.org/wiki/Unix) released in 1983.

A socket is an abstract representation ([handle](https://en.wikipedia.org/wiki/Handle_(computing))) for the local endpoint of a network communication path. The Berkeley sockets API represents it as a [file descriptor](https://en.wikipedia.org/wiki/File_descriptor) ([file handle](https://en.wikipedia.org/wiki/File_handle)) in the [Unix philosophy](https://en.wikipedia.org/wiki/Unix_philosophy) that provides a common interface for input and output to [streams](https://en.wikipedia.org/wiki/Standard_streams) of data.

Berkeley sockets evolved with little modification from a [*de facto* standard](https://en.wikipedia.org/wiki/De_facto_standard) into a component of the [POSIX](https://en.wikipedia.org/wiki/POSIX) specification. Therefore, the term **POSIX sockets** are essentially synonymous with *Berkeley sockets*. They are also known as *BSD sockets*, acknowledging the first implementation in the [Berkeley Software Distribution](https://en.wikipedia.org/wiki/Berkeley_Software_Distribution).

## Socket API functions

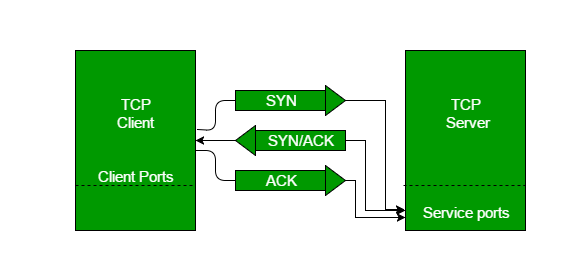
This list is a summary of functions or methods provided by the Berkeley sockets API library:

* socket() creates a new socket of a certain socket type, identified by an integer number, and allocates system resources to it.
* bind() is typically used on the server side, and associates a socket with a socket address structure, i.e. a specified local port number and IP address.
* listen() is used on the server side, and causes a bound TCP socket to enter listening state.
* connect() is used on the client side, and assigns a free local port number to a socket. In case of a TCP socket, it causes an attempt to establish a new TCP connection.
* accept() is used on the server side. It accepts a received incoming attempt to create a new TCP connection from the remote client, and creates a new socket associated with the socket address pair of this connection.
* send() and recv(), or write() and read(), or sendto() and recvfrom(), are used for sending and receiving data to/from a remote socket.
* close() causes the system to release resources allocated to a socket. In case of TCP, the connection is terminated.

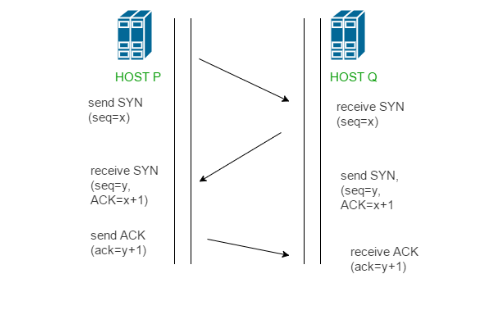
# TCP 3-Way Handshake Process

This could also be seen as a way of how TCP connection is established. Before getting into the details, let us look at some basics. TCP stands for **Transmission Control Protocol** which indicates that it does something to control the transmission of the data in a reliable way.

The process of communication between devices over the internet happens according to the current **TCP/IP** suite model (stripped out version of OSI reference model). The Application layer is a top pile of stack of TCP/IP model from where network referenced application like web browser on the client side establishes connection with the server. From the application layer, the information is transferred to the transport layer where our topic comes into picture. The two important protocols of this layer are – TCP, **UDP (User Datagram Protocol)** out of which TCP is prevalent (since it provides reliability for the connection established). However you can find application of UDP in querying the DNS server to get the binary equivalent of the Domain Name used for the website.



TCP provides reliable communication with something called **Positive Acknowledgement with Re-transmission (PAR)**. The Protocol Data Unit (PDU) of the transport layer is called segment. Now a device using PAR resend the data unit until it receives an acknowledgement. If the data unit received at the receiver’s end is damaged (It checks the data with checksum functionality of the transport layer that is used for Error Detection), then receiver discards the segment. So the sender has to resend the data unit for which positive acknowledgement is not received. You can realize from above mechanism that three segments are exchanged between sender (client) and receiver (server) for a reliable TCP connection to get established. Let us delve how this mechanism works:



* **Step 1 (SYN) :**In the first step, client wants to establish a connection with server, so it sends a segment with SYN(Synchronize Sequence Number) which informs server that client is likely to start communication and with what sequence number it starts segments with
* **Step 2 (SYN + ACK):**Server responds to the client request with SYN-ACK signal bits set. Acknowledgement(ACK) signifies the response of segment it received and SYN signifies with what sequence number it is likely to start the segments with
* **Step 3 (ACK) :**In the final part client acknowledges the response of server and they both establish a reliable connection with which they will start the actual data transfer

The steps 1, 2 establish the connection parameter (sequence number) for one direction and it is acknowledged. The steps 2, 3 establish the connection parameter (sequence number) for the other direction and it is acknowledged. With these, a full-duplex communication is established.

**UDP**

The User Datagram Protocol (UDP) is simplest Transport Layer communication protocol available of the TCP/IP protocol suite. It involves minimum amount of communication mechanism. UDP is said to be an unreliable transport protocol but it uses IP services which provides best effort delivery mechanism.

In UDP, the receiver does not generate an acknowledgement of packet received and in turn, the sender does not wait for any acknowledgement of packet sent. This shortcoming makes this protocol unreliable as well as easier on processing.

Requirement of UDP

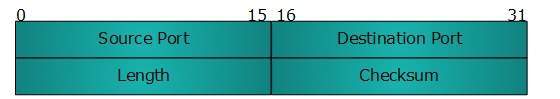
A question may arise, why do we need an unreliable protocol to transport the data? We deploy UDP where the acknowledgement packets share significant amount of bandwidth along with the actual data. For example, in case of video streaming, thousands of packets are forwarded towards its users. Acknowledging all the packets is troublesome and may contain huge amount of bandwidth wastage. The best delivery mechanism of underlying IP protocol ensures best efforts to deliver its packets, but even if some packets in video streaming get lost, the impact is not calamitous and can be ignored easily. Loss of few packets in video and voice traffic sometimes goes unnoticed.

Features

* UDP is used when acknowledgement of data does not hold any significance.
* UDP is good protocol for data flowing in one direction.
* UDP is simple and suitable for query based communications.
* UDP is not connection oriented.
* UDP does not provide congestion control mechanism.
* UDP does not guarantee ordered delivery of data.
* UDP is stateless.
* UDP is suitable protocol for streaming applications such as VoIP, multimedia streaming.

UDP Header

UDP header is as simple as its function.



UDP header contains four main parameters:

* **Source Port** - This 16 bits information is used to identify the source port of the packet.
* **Destination Port** - This 16 bits information, is used identify application level service on destination machine.
* **Length** - Length field specifies the entire length of UDP packet (including header). It is 16-bits field and minimum value is 8-byte, i.e. the size of UDP header itself.
* **Checksum** - This field stores the checksum value generated by the sender before sending. IPv4 has this field as optional so when checksum field does not contain any value it is made 0 and all its bits are set to zero.

UDP application

Here are few applications where UDP is used to transmit data:

* Domain Name Services
* Simple Network Management Protocol
* Trivial File Transfer Protocol
* Routing Information Protocol
* Kerberos

**TCP**

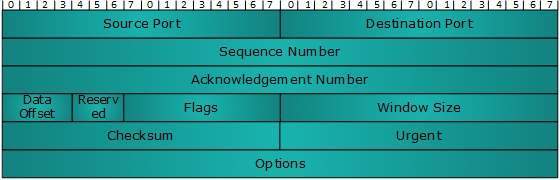
The transmission Control Protocol (TCP) is one of the most important protocols of Internet Protocols suite. It is most widely used protocol for data transmission in communication network such as internet.

## Features

* TCP is reliable protocol. That is, the receiver always sends either positive or negative acknowledgement about the data packet to the sender, so that the sender always has bright clue about whether the data packet is reached the destination or it needs to resend it.
* TCP ensures that the data reaches intended destination in the same order it was sent.
* TCP is connection oriented. TCP requires that connection between two remote points be established before sending actual data.
* TCP provides error-checking and recovery mechanism.
* TCP provides end-to-end communication.
* TCP provides flow control and quality of service.
* TCP operates in Client/Server point-to-point mode.
* TCP provides full duplex server, i.e. it can perform roles of both receiver and sender.

## Header

The length of TCP header is minimum 20 bytes long and maximum 60 bytes.



* **Source Port (16-bits)** - It identifies source port of the application process on the sending device.
* **Destination Port (16-bits)** - It identifies destination port of the application process on the receiving device.
* **Sequence Number (32-bits)** - Sequence number of data bytes of a segment in a session.
* **Acknowledgement Number (32-bits)** - When ACK flag is set, this number contains the next sequence number of the data byte expected and works as acknowledgement of the previous data received.
* **Data Offset (4-bits)** - This field implies both, the size of TCP header (32-bit words) and the offset of data in current packet in the whole TCP segment.
* **Reserved (3-bits)**  - Reserved for future use and all are set zero by default.
* **Flags (1-bit each)**
  + **NS** - Nonce Sum bit is used by Explicit Congestion Notification signaling process.
  + **CWR** - When a host receives packet with ECE bit set, it sets Congestion Windows Reduced to acknowledge that ECE received.
  + **URG** - It indicates that Urgent Pointer field has significant data and should be processed.
  + **ACK** - It indicates that Acknowledgement field has significance. If ACK is cleared to 0, it indicates that packet does not contain any acknowledgement.
  + **PSH** - When set, it is a request to the receiving station to PUSH data (as soon as it comes) to the receiving application without buffering it.
  + **RST** - Reset flag has the following features:
    - It is used to refuse an incoming connection.
    - It is used to reject a segment.
    - It is used to restart a connection.
  + **SYN** - This flag is used to set up a connection between hosts.
  + **FIN** - This flag is used to release a connection and no more data is exchanged thereafter. Because packets with SYN and FIN flags have sequence numbers, they are processed in correct order.
* **Windows Size** - This field is used for flow control between two stations and indicates the amount of buffer (in bytes) the receiver has allocated for a segment, i.e. how much data is the receiver expecting.
* **Checksum** - This field contains the checksum of Header, Data and Pseudo Headers.
* **Urgent Pointer**  - It points to the urgent data byte if URG flag is set to 1.
* **Options**  - It facilitates additional options which are not covered by the regular header. Option field is always described in 32-bit words. If this field contains data less than 32-bit, padding is used to cover the remaining bits to reach 32-bit boundary.

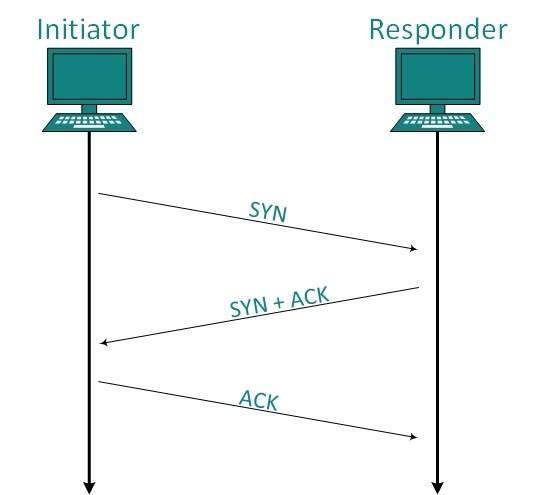
## Addressing

TCP communication between two remote hosts is done by means of port numbers (TSAPs). Ports numbers can range from 0 – 65535 which are divided as:

* System Ports (0 – 1023)
* User Ports ( 1024 – 49151)
* Private/Dynamic Ports (49152 – 65535)

## Connection Management

TCP communication works in Server/Client model. The client initiates the connection and the server either accepts or rejects it. Three-way handshaking is used for connection management.



### A port no. 5000 seq no. 1

C **port no. 5000 seq no. 1**

### Establishment

Client initiates the connection and sends the segment with a Sequence number. Server acknowledges it back with its own Sequence number and ACK of client’s segment which is one more than client’s Sequence number. Client after receiving ACK of its segment sends an acknowledgement of Server’s response.

### Release

Either of server and client can send TCP segment with FIN flag set to 1. When the receiving end responds it back by ACKnowledging FIN, that direction of TCP communication is closed and connection is released.

## ****TCP Timers-****

Timers used by TCP to avoid excessive delays during communication are called as TCP Timers. TCP uses several timers to ensure that excessive delays are not encountered during communications. Several of these timers are elegant, handling problems that are not immediately obvious at first analysis. Each of the timers used by TCP is examined in the following sections, which reveal its role in ensuring data is properly sent from one connection to another.

1. Time Out Timer
2. Time Wait Timer
3. Keep Alive Timer
4. Persistent Timer

**Time Out Timer-**

|  |
| --- |
| TCP uses a time out timer for retransmission of lost segments. |

* Sender starts a time out timer after transmitting a TCP segment to the receiver.
* If sender receives an acknowledgement before the timer goes off, it stops the timer.
* If sender does not receive any acknowledgement and the timer goes off, then [**TCP Retransmission**](https://www.gatevidyalay.com/tcp-retransmission-tcp-computer-networks/) occurs.
* Sender retransmits the same segment and resets the timer.
* The value of time out timer is dynamic and changes with the amount of traffic in the network.
* Time out timer is also called as **Retransmission Timer**.

**Time Wait Timer-**

|  |
| --- |
| TCP uses a time wait timer during connection termination. |

* Sender starts the time wait timer after sending the ACK for the second FIN segment.
* It allows resending the final acknowledgement if it gets lost.
* It prevents the just closed port from reopening again quickly to some other application.
* It ensures that all the segments heading towards the just closed port are discarded.
* The value of time wait timer is usually set to twice the lifetime of a TCP segment.

**Keep Alive Timer-**

|  |
| --- |
| TCP uses a keep alive timer to prevent long idle TCP connections. |

* Each time server hears from the client, it resets the keep alive timer to 2 hours.
* If server does not hear from the client for 2 hours, it sends 10 probe segments to the client.
* These probe segments are sent at a gap of 75 seconds.
* If server receives no response after sending 10 probe segments, it assumes that the client is down.
* Then, server terminates the connection automatically.

**Persistent Timer-**

* TCP uses a persistent timer to deal with a zero-widow-size deadlock situation.
* It keeps the window size information flowing even if the other end closes its receiver window.

|  |
| --- |
| **Explanation**    Consider the following situation-   * Sender receives an acknowledgment from the receiver with zero window size. * This indicates the sender to wait. * Later, receiver updates the window size and sends the segment with the update to the sender. * This segment gets lost. * Now, both sender and receiver keep waiting for each other to do something.   To deal with such a situation, TCP uses a persistent timer. |

* Sender starts the persistent timer on receiving an ACK from the receiver with a zero window size.
* When persistent timer goes off, sender sends a special segment to the receiver.
* This special segment is called as probe segment and contains only 1 byte of new data.
* Response sent by the receiver to the probe segment gives the updated window size.
* If the updated window size is non-zero, it means data can be sent now.
* If the updated window size is still zero, the persistent timer is set again and the cycle repeats.

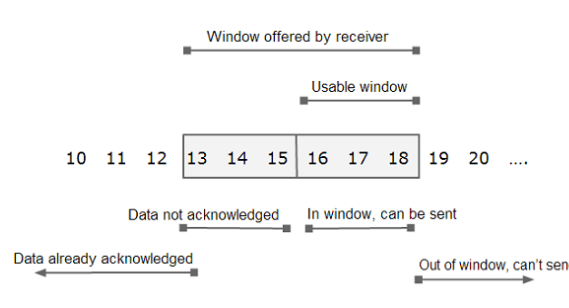
**TCP Flow Control**

TCP is the protocol that guarantees we can have a reliable communication channel over an unreliable network. When we send data from a node to another, packets can be lost, they can arrive out of order, the network can be congested or the receiver node can be overloaded. When we are writing an application, though, we usually don’t need to deal with this complexity, we just write some data to a socket and TCP makes sure the packets are delivered correctly to the receiver node. Another important service that TCP provides is what is called Flow Control.

### TCP Sliding Window

The process described in last section is good on receiver’s part but sender has to maintain a window on its side too. This window covers unacknowledged data and the data it can send keeping in mind the window size advertised by the receiver.

Following figure should give you an idea about how a sliding window looks like :



In the figure shown above:

* The available window advertised by the receiver is 6. This means that receiver can accept 6 bytes as of now.
* The window at sender side covers bytes ranging from 13 to 18 (I.e. 6 bytes in total).
* Out of this range, 13-15 are the bytes which have been sent but no acknowledgement is yet received for them.
* Bytes 16-18 are the bytes that sender can send as soon as possible.
* If sender starts receiving acknowledgement for bytes 13 to 15, the left end of the window starts closing in.
* The right end starts opening up as more and more window size is advertised by the receiver.
* This window slides towards right depending upon how fast receiver consumes data and sends acknowledgement and hence known as sliding window.

# TCP Congestion Control

TCP uses a congestion window and a congestion policy that avoid congestion. Previously, we assumed that only receiver can dictate the sender’s window size. We ignored another entity here, the network. If the network cannot deliver the data as fast as it is created by the sender, it must tell the sender to slow down. In other words, in addition to the receiver, the network is a second entity that determines the size of the sender’s window.

## How TCP slow start works

TCP slow start is one of the first steps in the congestion control process. It balances the amount of data a sender can transmit (known as the [congestion window](https://blog.stackpath.com/glossary-cwnd-and-rwnd/)) with the amount of data the receiver can accept (known as the [receiver window](https://blog.stackpath.com/glossary-cwnd-and-rwnd/)). The lower of the two values becomes the maximum amount of data that the sender is allowed to transmit before receiving an acknowledgment from the receiver.

**Step-by-step, here’s how slow start works:**

1. A sender attempts to communicate to a receiver. The sender’s initial packet contains a small congestion window, which is determined based on the sender’s maximum window.
2. The receiver acknowledges the packet and responds with its own window size. If the receiver fails to respond, the sender knows not to continue sending data.
3. After receiving the acknowledgement, the sender increases the next packet’s window size. The window size gradually increases until the receiver can no longer acknowledge each packet, or until either the sender or the receiver’s window limit is reached.

