Connection Oriented and Connectionless Services

These are the two services given by the layers to layers above them. These services are:

- Connection Oriented Service
- Connectionless Services

Connection Oriented Services

There is a sequence of operation to be followed by the users of connection oriented service. These are:

- Connection is established.
- Information is sent.
- Connection is released.

In connection oriented service we have to establish a connection before starting the communication. When connection is established, we send the message or the information and then we release the connection.

Connection oriented service is more reliable than connectionless service. We can send the message in connection oriented service if there is an error at the receivers end. Example of connection oriented is TCP (Transmission Control Protocol) protocol.

Connection Less Services

It is similar to the postal services, as it carries the full address where the message (letter) is to be carried. Each message is routed independently from source to destination. The order of message sent can be different from the order received.

In connectionless the data is transferred in one direction from source to destination without checking that destination is still there or not or if it prepared to accept the message. Authentication is not needed in this. Example of Connectionless service is UDP (User Datagram Protocol) protocol.

Difference between Connection-Oriented and Connectionless Service

.

Connection-Oriented Service

A connection-oriented service is a network service that was designed and developed after the telephone system. A connection-oriented service is used to create an end to end connection between the sender and the receiver before transmitting the data over the same or different networks. In connection-oriented service, packets are transmitted to the receiver in the same order the sender has sent them. It uses a handshake method that creates a connection between the user and sender for transmitting the data over the network. Hence it is also known as a reliable network service.

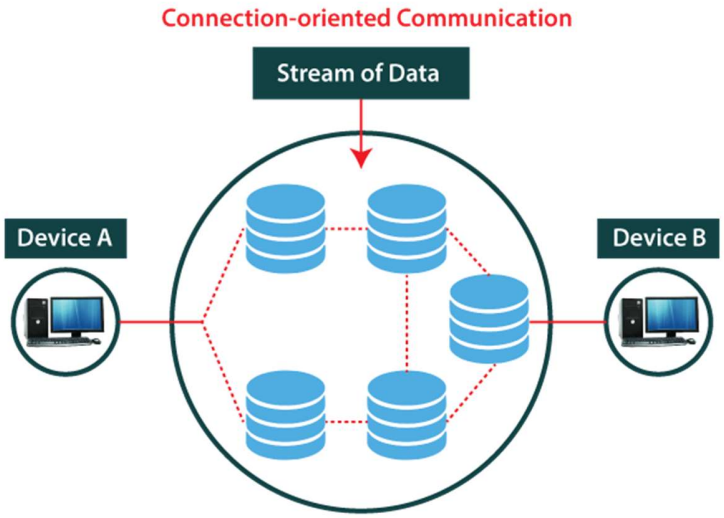

Figure 1: Connected – Oriented Communication

Connection-Oriented vs Connectionless Service

Suppose, a sender wants to send data to the receiver. Then, first, the sender sends a request packet to a receiver in the form of an SYN packet. After that, the receiver responds to the sender's request with an (SYN-ACK) signal/packets. That represents the confirmation is received by the receiver to start the communication between the sender and the receiver. Now a sender can send the message or data to the receiver.

Similarly, a receiver can respond or send the data to the sender in the form of packets. After successfully exchanging or transmitting data, a sender can terminate the connection by sending a signal to the receiver. In this way, we can say that it is a reliable network service.

Connectionless Service

A connection is similar to a postal system, in which each letter takes along different route paths from the source to the destination address. Connectionless service is used in the network system to transfer data from one end to another end without creating any connection. So it does not require establishing a connection before sending the data from the sender to the receiver. It is not a reliable network service because it does not guarantee the transfer of data packets to the receiver, and data packets can be received in any order to the receiver. Therefore we can say that the data packet does not follow a defined path. In connectionless service, the transmitted data packet is not received by the receiver due to network congestion, and the data may be lost.

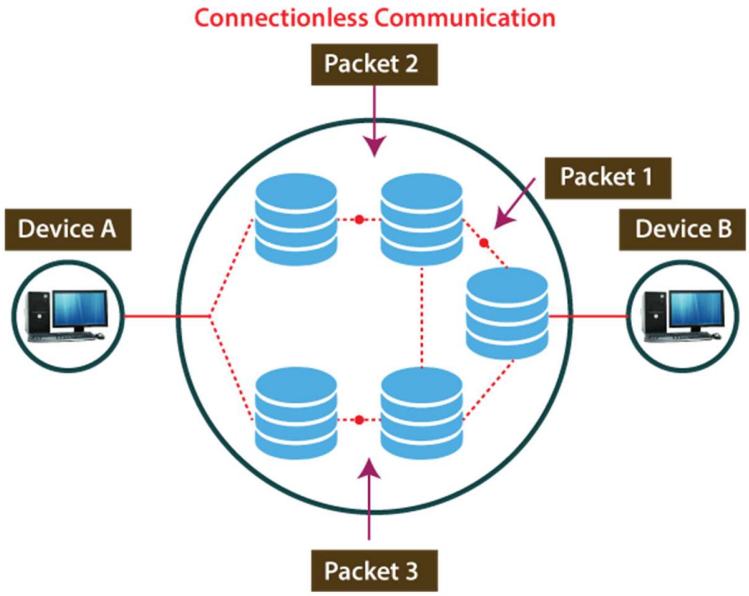

Figure 2: Connectionless Communication

S. No	Comparison Parameter	Connection-oriented Service	Connection Less Service
1.	Related System	It is designed and developed based on the telephone system.	It is service based on the postal system.
2.	Definition	It is used to create an end to end connection between the senders to the receiver before transmitting the data over the same or different network.	It is used to transfer the data packets between senders to the receiver without creating any connection.
3.	Virtual path	It creates a virtual path between the sender and the receiver.	It does not create any virtual connection or path between the sender and the receiver.
4.	Authentication	It requires authentication before transmitting the data packets to the receiver.	It does not require authentication before transferring data packets.
5.	Data Packets Path	All data packets are received in the same order as those sent by the sender.	Not all data packets are received in the same order as those sent by the sender.

6.	Bandwidth Requirement	It requires a higher bandwidth to transfer the data packets.	It requires low bandwidth to transfer the data packets.
7.	Data Reliability	It is a more reliable connection service because it guarantees data packets transfer from one end to the other end with a connection.	It is not a reliable connection service because it does not guarantee the transfer of data packets from one end to another for establishing a connection.
8.	Congestion	There is no congestion as it provides an end-to-end connection between sender and receiver during transmission of data.	There may be congestion due to not providing an end-to-end connection between the source and receiver to transmit of data packets.
9.	Examples	Transmission Control Protocol (TCP) is an example of a connection-oriented service.	User Datagram Protocol (UDP), Internet Protocol (IP), and Internet Control Message Protocol (ICMP) are examples of connectionless service.

Connection-Oriented and Connectionless Protocols

Generally, networking technologies are contrasted based on whether or not they use a dedicated path, or *circuit*, over which to send data, i.e circuit versus packet switching. Another way in which technologies and protocols are differentiated has to do with whether or not they use *connections* between devices.

Division of Protocols into Connection-Related Categories

Protocols are divided into two categories based on their use of connections:

- **Connection-Oriented Protocols:** These protocols require that a logical connection be established between two devices before transferring data. This is generally accomplished by following a specific set of rules that specify how a connection should be initiated, negotiated, managed and eventually terminated. Usually one device begins by sending a request to open a connection, and the other responds. They pass control information to determine if and how the connection should be set up. If this is successful, data is sent between the devices. When they are finished, the connection is broken.

- **Connectionless Protocols:** These protocols do not establish a connection between devices. As soon as a device has data to send to another, it just sends it.

Connection-oriented protocols are important because they enable the implementation of applications that require connections, over packet-switched networks that have no inherent sense of a connection. For example, to use the TCP/IP File Transfer Protocol, you want to be able to connect to a server, enter a login and password, and then execute commands to change directories, send or retrieve files, and so on.

This requires the establishment of a connection over which commands, replies and data can be passed. Similarly, the Telnet Protocol obviously involves establishing a connection—it lets you remotely use another machine. Yet, both of these work (indirectly) over the IP protocol, which is based on the use of packets.

Succinctly put, we can say, circuit-switched networking technologies are inherently connection-oriented, but not all connection-oriented technologies use circuit switching. Logical connection-oriented protocols can in fact be implemented on top of packet switching networks to provide higher-layer services to applications that require connections.

The TCP/IP Protocol suite has two main protocols that operate at the transport layer of the OSI Reference Model. One is the Transmission Control Protocol (TCP), which is connection-oriented; the other, the User Datagram Protocol (UDP), is connectionless. TCP is used for applications that require the establishment of connections (as well as TCP's other service features), prior to sending data. UDP is used by other applications that don't need connections or other features, but do need the faster performance that UDP can offer by not needing to make such connections before sending data.

Connection Oriented protocol like TCP should not be mistaken for Circuit switching. The reason is that the similarity ends at setting of connection to send data back and forth between devices, all that data is indeed still being sent as packets; there is no real circuit between the devices. This implies TCP must deal with all the potential pitfalls of packet-switched communication, such as the potential for data loss or receipt of data pieces in the incorrect order. Certainly, the existence of connection-oriented protocols like TCP doesn't obviate the need for circuit switching technologies (this can be contested).

TCP/IP Transmission Control Protocol (TCP)

If the Internet Protocol (IP), is the "workhorse" of the TCP/IP protocol suite, then the "worker" that rides on that horse would have to be the TCP/IP *Transmission Control Protocol (TCP)*. Like horse and rider, TCP and IP form a team that work together to make it possible for applications to easily run over an internetwork. The two are very important complements to each other, hence the reason why the TCP/IP protocol suite is named after them. While IP concerns itself with classic network-layer tasks such as addressing, datagram packaging and routing, which provide basic internetworking capabilities, TCP provides to applications a method of easily making use of IP, while filling in the capabilities that IP lacks. It allows TCP/IP devices to establish and manage connections and send data reliably, and takes care of handling all the potential issues that can occur during transmission so each application doesn't need to worry about such matters. To applications, TCP could thus be considered almost like a nice *user interface* to the fairly rudimentary capabilities of IP.

TCP is a connection-oriented, acknowledged, reliable, fully-featured protocol designed to provide applications with a reliable way to send data using the unreliable Internet Protocol. It allows applications to send bytes of data as a *stream* of bytes, and automatically packages them into appropriately-sized *segments* for transmission. It uses a special *sliding window acknowledgment system* to ensure that all data is received by its recipient, to handle necessary retransmissions, and to provide flow control so each device in a connection can manage the rate at which it is sent data.

TCP Functions

TCP's basic operation can be reasonably simplified by describing its primary functions. The following are what can be referred to as the five main tasks that TCP performs.

- **Addressing/Multiplexing:** TCP is used by many different applications for their transport protocol. Therefore, like its simpler sibling UDP, an important job for TCP is *multiplexing* the data received from these different processes so they can be sent out using the underlying network-layer protocol. At the same time, these higher-layer application processes are identified using TCP ports.

- **Connection Establishment, Management and Termination:** TCP provides a set of procedures that devices follow to negotiate and establish a TCP connection over which data can travel. Once opened, TCP includes logic for managing connections and handling problems that may result with them. When a device is done with a TCP connection, a special process is followed to terminate it.

- **Data Handling and Packaging:** TCP defines a mechanism by which applications are able to send data to it from higher layers. This data is then packaged into messages to be sent to the destination TCP software. The destination software unpackages the data and gives it to the application on the destination machine.

- **Data Transfer:** Conceptually, the TCP implementation on a transmitting device is responsible for the transfer of packaged data to the TCP process on the other device. Following the principle of layering, this is done by having the TCP software on the sending machine pass the data packets to the underlying network-layer protocol, which again normally means IP.

- **Providing Reliability and Transmission Quality Services:** TCP includes a set of services and features that allow an application to consider the sending of data using the protocol to be "reliable". This means that normally, a TCP application doesn't have to worry about data being sent and never showing up, or arriving in the wrong order. It also means other common problems that might arise if IP were used directly are avoided.

- **Providing Flow Control and Congestion Avoidance Features:** TCP allows the flow of data between two devices to be controlled and managed. It also includes features to deal with congestion that may be experienced during communication between devices.

Functions Not Performed By TCP

TCP has its limitations and certain areas that its designers specifically did not address. Among the notable functions TCP does not perform include:

- **Specifying Application Use:** TCP defines the transport protocol. It does not describe specifically how applications are to use TCP.

- **Providing Security:** TCP does not provide any mechanism for ensuring the authenticity or privacy of data it transmits. If these are needed they must be accomplished using some other means, such as IPSec, for example.

- **Maintaining Message Boundaries:** TCP sends data as a continuous stream, not as discrete messages. It is up to the application to specify where one message ends and the next begins.

- **Guaranteeing Communication:** *The question is : "isn't the whole point of TCP supposed to be that it guarantees data will get to its destination?"* Well, yes and no. TCP will detect unacknowledged transmissions and re-send them if needed. However, in the event of some sort of problem (network down, server failure etc) that prevents reliable communication, all TCP can do is “keep trying”. It can't make any guarantees because there are too many things out of its control. Similarly, it can attempt to manage the flow of data, but cannot resolve every problem

TCP Characteristics

The following are some ways to describe the Transmission Control Protocol and how it performs the functions described in the preceding topic:

- **Connection-Oriented:** TCP requires that devices first establish a connection with each other before they send data. The connection creates the equivalent of a circuit between the units, and is analogous to a telephone call. A process of negotiation occurs to establish the connection, ensuring that both devices agree on how data is to be exchanged.

- **Bidirectional:** Once a connection is established, TCP devices send data bidirectionally. Both devices on the connection can send and receive, regardless of which of them initiated the connection.

- **Multiply-Connected and Endpoint-Identified:** TCP connections are identified by the pair of sockets used by the two devices in the connection. This allows each device to have multiple connections opened, either to the same IP device or different IP devices, and to handle each connection independently without conflicts.

- **Reliable:** Communication using TCP is said to be *reliable* because TCP keeps track of data that has been sent and received to ensure it all gets to its destination. As earlier mentioned, TCP can't really “guarantee” that data will always be received. However, it ***can*** guarantee that all data sent will be checked for reception, and checked for data integrity, and then retransmitted when needed.

- **Acknowledged:** A key to providing reliability is that all transmissions in TCP are acknowledged (at the TCP layer—TCP cannot guarantee that all such transmissions are received by the remote application). The recipient must tell the sender “yes, I got that” for each piece of data transferred. This is in stark contrast to typical messaging protocols where the sender never knows what happened to its transmission. As we will see, this is fundamental to the operation of TCP as a whole.

- **Stream-Oriented:** Most lower-layer protocols are designed so that to use them, higher-layer protocols must send them data in blocks. IP is the best example of this; you send it a message to be formatted and it puts that message into a datagram. UDP is the same. In contrast, TCP allows applications to send it a continuous stream of data for transmission. Applications don't need to worry about making this into chunks for transmission; TCP does it.

- **Data-Unstructured:** An important consequence of TCP's stream orientation is that there are no natural divisions between data elements in the application's data stream. When multiple messages are sent over TCP, applications must provide a way of differentiating one message (data element, record, etc.) from the next.

- **Data-Flow-Managed:** TCP does more than just package data and send it as fast as possible. A TCP connection is *managed* to ensure that data flows evenly and smoothly, with means included to deal with problems that arise along the way.

TCP Connection Establishment

In TCP connections, the host client establishes the connection with the server using the three-way handshake process.

TCP THREE WAY HANDSHAKE

Before describing the three way handshake, it is important to types of message that control transitions between different states in TCP operations. These message types correspond to the TCP header flags that are set to indicate the function served. The three most popular ones are:

- ***SYN**:* A *synchronize* message, used to initiate and establish a connection. It is so named since one of its functions is to synchronizes sequence numbers between devices.
- ***FIN**:* A *finish* message, which is a TCP segment with the *FIN* bit set, indicating that a device wants to terminate the connection.
- ***ACK**:* An *acknowledgment*, indicating receipt of a message such as a *SYN* or a *FIN*.

There are other control messages, *RST*, *PSH* and *URG.* Please note only two of the three, the SYN and ACK is used in the three way handshake. The three way handshake is illustrated in Figure 3 below.

So, here are the main 3 steps, also known as TCP 3-way Handshake, which is required to fully establish the TCP connection:

In the first step, on the one side, either client or the server initiates the connection establishment by sending “synchronize” or **an SYN flag** to the other side. In this step, the client’s initial sequence number will be sent to the server to create a connection.

In return for the SYN flag, the TCP server will send an “Acknowledgement” or **ACK flag** along with its initial sequence number for the connection establishment.

And finally, with the client’s confirmation through **the ACK flag, in return to the server’s SYN flag**, the connection establishment will be activated.

Now, to understand the process clearly, here we will be discussing the further steps to establish the TCP connection down below:

In the first step, the client sends an SYN request to establish a connection, as the Packet SYN: to the server. After successfully receiving the SYN flag or the packet, now the server will send an ACK flag as the packet ACK: for the confirmation purpose to the client. In the process, the server can also send the data to the other side. For this, the server will be sending the data as the packet SYN: along to the client.

Finally, after receiving the data successfully, the client will then send an ACK flag as packet ACK: to the server as a confirmation.

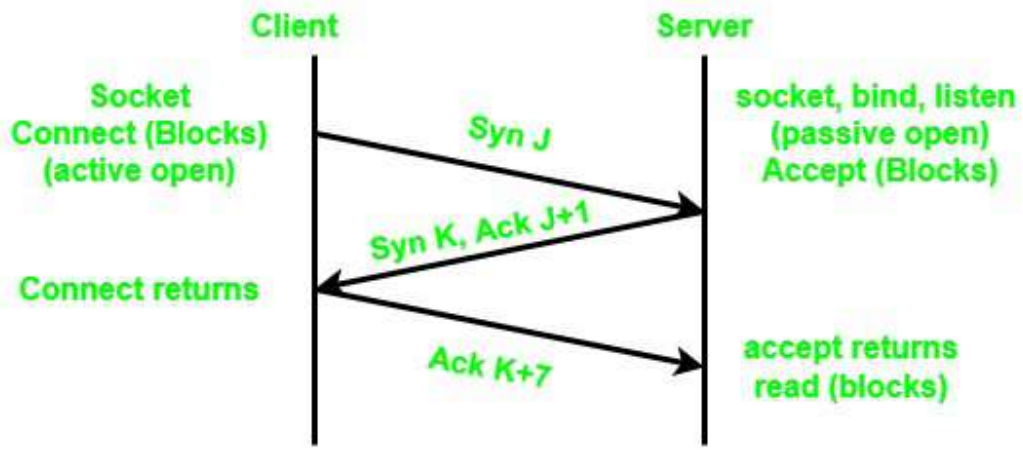

Figure 3: The TCP three way Handshake

TCP Connection Termination

To terminate an established TCP connection, the following 4 TCP packets are needed to be exchanged. Which you can understand with the statements below:

1. Host A → Host B: FIN flag set.
2. Host B → Host A: ACK flag set.
3. Host B → Host A: FIN flag set.
4. Host A → Host B: ACK flag set.

These 4 steps are known as a TCP 4-way handshake, which is necessary to terminate a TCP connection. For the termination of the established TCP connection, the following steps are necessary for the process. Which are given down below:

1. Firstly, from one side of the connection, either from the client or the server the FIN flag will be sent as the request for the termination of the connection.

2. In the second step, whoever receives the FIN flag will then be sending an ACK flag as the acknowledgment for the closing request to the other side.

3. And, at the Later step, the server will also send a FIN flag as the closing signal to the other side.

4. In the final step, the TCP, who received the final FIN flag, will be sending an ACK flag as the final Acknowledgement for the suggested connection closing.

Since the Four main steps are required to close an active connection, so, it is called a four-way handshake. For a better understanding, you can take a look at the diagram below:

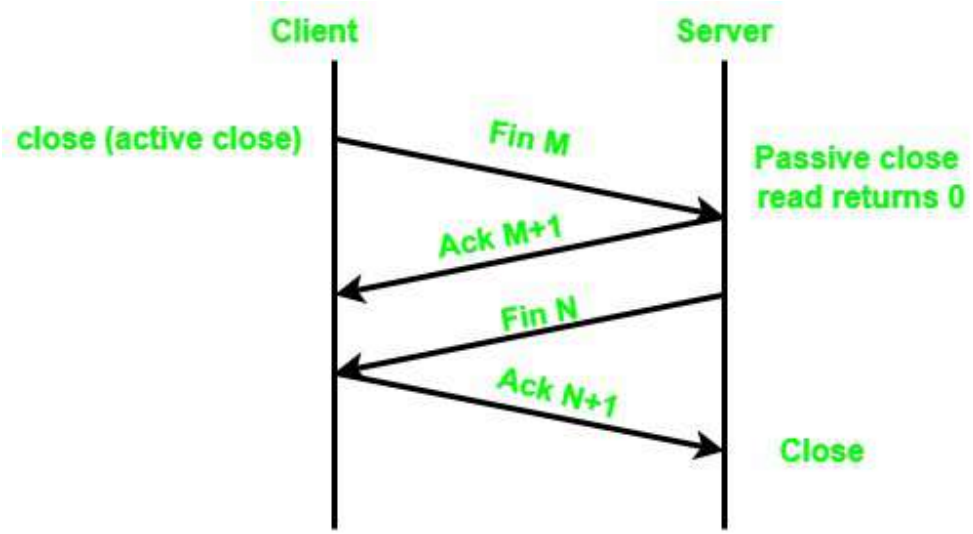

Figure 4: The Four-Way Handshake Process

Why Does TCP Connect Termination Need 4-way-Handshake?

Now the question is, why termination can not send ACK and FIN packets at once? By which one step can be reduced. And why the four-way handshake is necessary here?

Well, by looking at it carefully, it can be seen that the four-way Handshake is actually a set of two-way handshakes. However, it is not completely false that in some cases, the 2 and 3 can be set at the same packet.

The four-way handshake works as a pair of two-way Handshakes. Where, the first phase is, when the client sends the FIN flag to the server, and in return, the server sends the ACK flag as acknowledgment.

This can be understood by the statement below:

Client ------FIN-----> Server

Client <-----ACK------ Server

At this point, the client is in the waiting state, it is waiting for the FIN flag from the server. So, that the connection can be terminated. This state can be mentioned as FIN_WAIT_2.

Now as we know that it works as the full-duplex mode, so if the one-side connection is broken down, then no more data can be sent from that side of the connection. But it can still receive the data from another side.

Here, the server can send more data when the Client is in the FIN_WAIT_2 state. And once the server is done sending data, then the server will send the FIN flag to the client as the termination request, and then the client sends the ACK flag as the confirmation to terminate the connection.

As you can see in the statement below:

Client <-----FIN------ Server

Client ------ACK-----> Server

So, as explained, in this case, steps 2 and 3 cannot be sent as one package, since they belong to two different states.

Here four-way handshake is necessary in this case, as the first FIN flag, that is sent to the server by the client is a request for termination. And the first ACK, received by the client is just a response to FIN 1. Now here only the connection from the client is disconnected, but the server is still in a working position. This means it may still have some data to send. So, in this condition, the connection cannot be cut down suddenly. here, the other two steps are needed to be performed by the server.

Now the other reason is that it can become difficult for both sides to define why the peer does not respond. Also, not only the offline state can cause a packet lost. Alongside, the other exceptions in the server's processing can also lead to it. And there is another problem with this idea is that in this condition, the client will have to wait for a long time until the time is out. Here, the four-way handshake looks like a better and easier option to address these problems.

TCP Data Packaging: Segments

TCP is designed to have applications send data to it as a *stream* of bytes, rather than requiring fixed-size messages to be used. This provide maximum flexibility for a wide variety of uses, because applications don't need to worry about data packaging, and can send files or messages of any size. TCP takes care of packaging these bytes into messages called *segments* that are sized based on a number of different parameters. These segments are passed to IP, where they are encapsulated into IP datagrams and transmitted. The process is reversed by the receiving device: segments are removed from IP datagrams, then the bytes are taken from the segments and passed up to the appropriate recipient application protocol as a byte stream.

Byte number, Sequence number and Acknowledgement number:

All the data bytes that are to be transmitted are numbered and the beginning of this numbering is arbitrary. Sequence numbers are given to the segments so as to reassemble the bytes at the receiver end even if they arrive in a different order. The sequence number of a segment is the byte number of the first byte that is being sent. The acknowledgement number is required since TCP provides full-duplex service. The acknowledgement number is the next byte number that the receiver expects to receive which also provides acknowledgement for receiving the previous bytes.
Example:

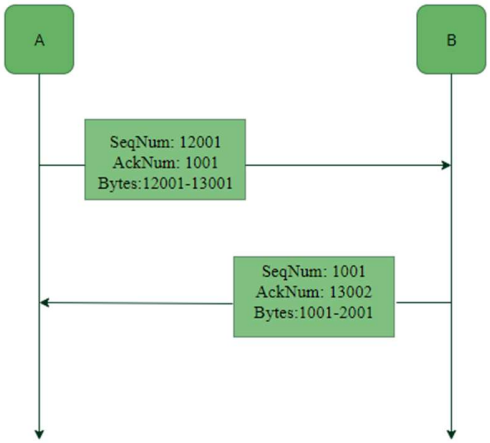

Figure 5: The TCP Numbering Process

In this example we see that A sends acknowledgement number1001, which means that it has received data bytes till byte number 1000 and expects to receive 1001 next, hence B next sends data bytes starting from 1001. Similarly, since B has received data bytes till byte number 13001 after the first data transfer from A to B, therefore B sends acknowledgement number 13002, the byte number that it expects to receive from A next.

TCP Segment structure

A TCP segment consists of data bytes to be sent and a header that is added to the data by TCP as shown:

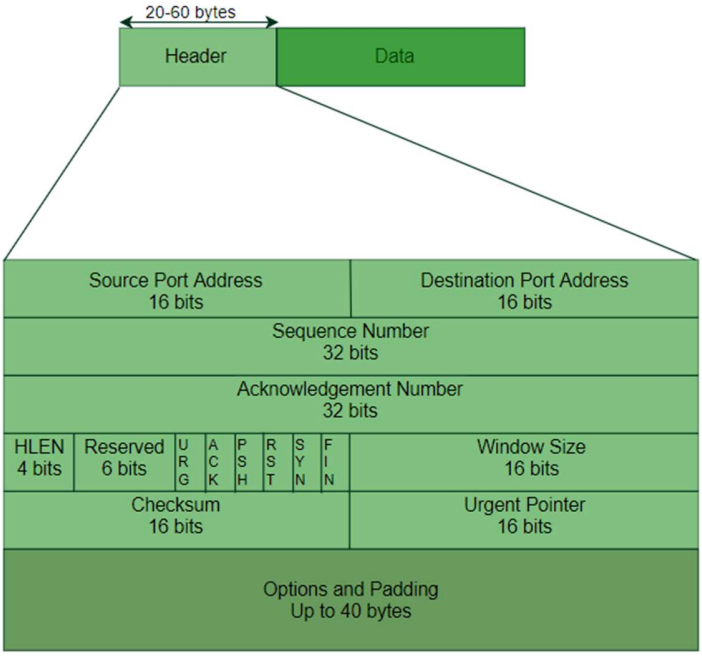

Figure 6: The TCP Segment Structure

The header of a TCP segment can range from 20-60 bytes. 40 bytes are for options. If there are no options, a header is 20 bytes else it can be of upmost 60 bytes.
Header fields:

- **Source Port Address –**
  A 16-bit field that holds the port address of the application that is sending the data segment.

- **Destination Port Address –**
  A 16-bit field that holds the port address of the application in the host that is receiving the data segment.

- **Sequence Number –**
  A 32-bit field that holds the sequence number, i.e, the byte number of the first byte that is sent in that particular segment. It is used to reassemble the message at the receiving end of the segments that are received out of order.

- **Acknowledgement Number –**
  A 32-bit field that holds the acknowledgement number, i.e, the byte number that the receiver expects to receive next. It is an acknowledgement for the previous bytes being received successfully.

- **Header Length (HLEN) –**
  This is a 4-bit field that indicates the length of the TCP header by a number of 4-byte words in the header, i.e if the header is 20 bytes(min length of TCP header), then this field will hold 5 (because 5 x 4 = 20) and the maximum length: 60 bytes, then it'll hold the value 15(because 15 x 4 = 60). Hence, the value of this field is always between 5 and 15.

- **Control flags –**
  These are 6 1-bit control bits that control connection establishment, connection termination, connection abortion, flow control, mode of transfer etc. Their function is:
    - URG: Urgent pointer is valid
    - ACK: Acknowledgement number is valid(used in case of cumulative acknowledgement)
    - PSH: Request for push
    - RST: Reset the connection
    - SYN: Synchronize sequence numbers
    - FIN: Terminate the connection
- **Window size –**
  This field tells the window size of the sending TCP in bytes.

- **Checksum –**
  This field holds the checksum for error control. It is mandatory in TCP as opposed to UDP.

Urgent pointer
This field (valid only if the URG control flag is set) is used to point to data that is urgently required that

needs to reach the receiving process at the earliest. The value of this field is added to the sequence number to get the byte number of the last urgent byte.

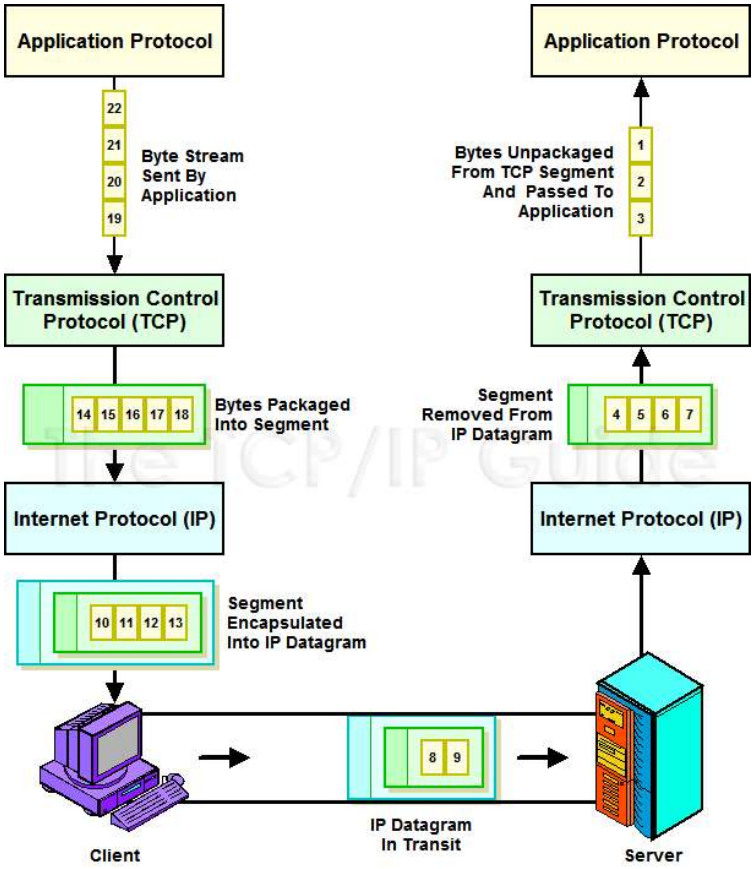

Figure 7: TCP Data Stream Processing and Segment Packaging

The TCP layer on a device accumulates data it receives from the application process stream. On regular intervals, it forms segments to be transmitted using IP. The size of the segment is controlled by two primary factors. The first issue is that there is an overall limit to the size of a segment, chosen to prevent unnecessary fragmentation at the IP layer. This is governed by a parameter called the maximum segment size (MSS), which is determined during connection establishment. The second is that TCP is designed so that once a connection is set up, each of the devices tells the other how much data it is ready to accept at any given time. If this is lower than the MSS value, a smaller segment must be sent. This is part of the sliding window system described later.

TCP Data Identification: Sequence Numbers

Since TCP is reliable, it needs to keep track of all the data it receives from an application so it can make sure it is all received by the destination. Furthermore, it must make sure the data is received in the order it was sent, and must retransmit any lost data. If data were conveyed to TCP in block-like messages, it would be fairly simple to keep track of the data by adding an identifier to each message. Since TCP is stream-oriented, however, that identification must be done for each byte of data! This may seem

surprising, but it is actually what TCP does, through the use of *sequence numbers*. Each byte of data is assigned a sequence number which is used to keep track of it through the process of transmission, reception and acknowledgment (though in practice, blocks of many bytes are managed using the sequence numbers of bytes at the start and end of the block). These sequence numbers are used to ensure that data sent in segments is reassembled into the original stream of data transmitted by the sending application. They are required to implement the sliding window system that enables TCP to provide reliability and data flow control

TCP Sliding Window Acknowledgment System For Data Transport, Reliability and Flow Control

The management of data in the connection oriented protocol called TCP is required to facilitate two key requirements of the protocol:

- **Reliability:** Ensuring that data that is sent actually arrives at its destination, and if not, detecting this and re-sending the data.
- **Data Flow Control:** Managing the rate at which data is sent so that it does not overwhelm the device that is receiving it.

To accomplish these tasks, the entire operation of the protocol is oriented around something called the *sliding window acknowledgment system*.

Providing Basic Reliability Using PAR

Before discussing the sliding window, there is need to discuss the basic reliability mechanism of TCP called ***Positive Acknowledgment with Retransmission (PAR).***

This is most easily done with a simple acknowledgment system. Device *A* sends a piece of data to Device *B*. Device *B*, receiving the data, sends back an *acknowledgment* saying, "Device *A*, I received your message". Device *A* then knows its transmission was successful.

Of course, since IP is unreliable, that message may in fact never get to where it is going. Device *A* will sit waiting for the acknowledgment and never receive it. Conversely, it is also possible that Device *B* gets the message from Device *A*, but the ***acknowledgment itself*** vanishes somehow. In either case, we don't want Device *A* to sit forever waiting for an acknowledgment that is never going to ever arrive.

To prevent this from happening, Device *A* starts a *timer* when it first sends the message to Device *B*, which allows sufficient time for the message to get to *B* and the acknowledgment to travel back, plus some reasonable time to allow for possible delays. If the timer expires before the acknowledgment is received, *A* assumes there was a problem and *retransmits* its original message. Since this method involves positive acknowledgments and a facility for retransmission when needed, it is commonly called *positive acknowledgment with retransmission (PAR)*, as shown in figure 8 below.

PAR is a technique that is used widely in networking and communications for protocols that exchange relatively small amounts of data, or exchange data infrequently. The basic method is functional, but it is ***inefficient*** for a protocol like TCP**.** There are two enhancements to the basic PAR scheme that addresses this. First, each message is given identification number; each can be acknowledged individually, so more than one can be in transit at a given time. Second, the receiver regularly communicates to the sender, a

send limit parameter, which restricts the number of messages the sender can have outstanding at once. The receiver can adjust this parameter to control the flow of data from the sender. See figure 9.

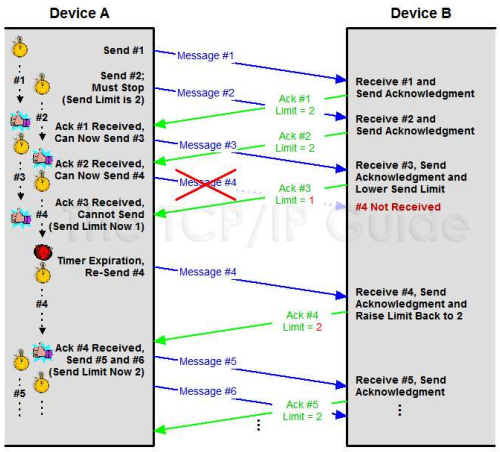

Figure 8: Basic PAR

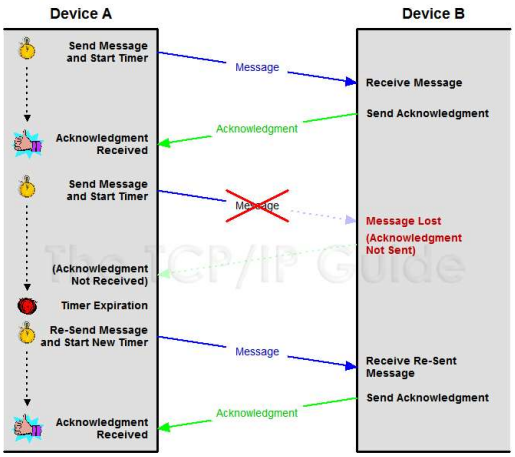

Figure 9: Enhanced PAR

The TCP sliding window uses a modified version of this technique but it does not transmit and acknowledge one byte at a time but instead deal with a range of bytes in a given segment (streams of data and not individual messages)

The Send Window and Usable Window

The key to the operation of the entire process is the number of bytes that the recipient is allowing the transmitter to have unacknowledged at one time. This is called the ***send window,*** or often, just the ***window***. The window is what determines how many bytes the sender is allowed to transmit. The term *usable window* is defined as the amount of data the transmitter is still allowed to send given the amount of data that is outstanding.

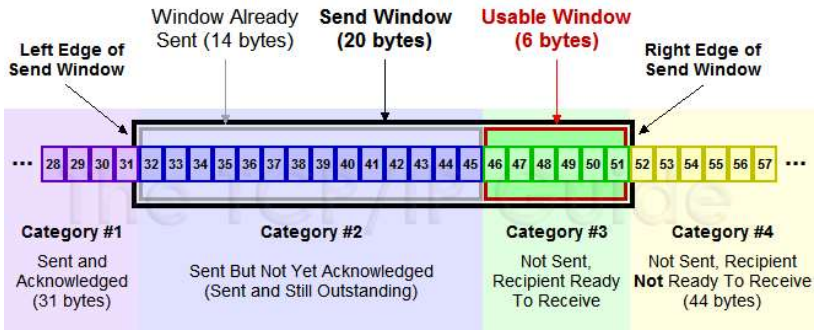

Figure 10: TCP Transmission Stream Categories and Send Window Terminology

The *send window* is the key to the entire TCP sliding window system. The usable window is the amount of the send window that the sender is still allowed to send at any point in time; it is equal to the size of the send window less the number of unacknowledged bytes already transmitted.

Processing Acknowledgments and Sliding the Send Window

Let's assume after some time, the destination device sends back a message to the sender providing an acknowledgment. If we assume five bytes have been acknowledged, and the window size didn't change, the sender is allowed to send five more bytes. In effect, the window shifts, or ***slides***, over to the right in the timeline. At the same time five bytes move from Category #2 to Category #1, five bytes move from Category #4 to Category #3, creating a new usable window for subsequent transmission. See figure 11.

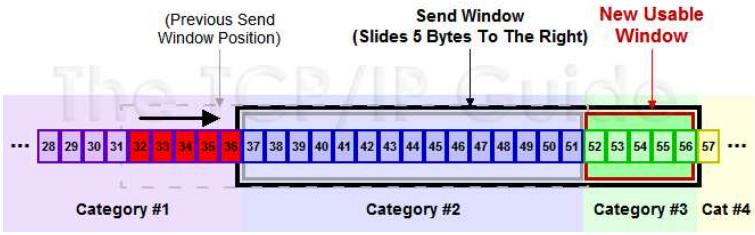

Figure 11: Sliding The TCP Send Window

User Datagram Protocol (UDP)

User Datagram Protocol (UDP) is a Transport Layer protocol. Unlike TCP, it is an unreliable and connectionless protocol. So, there is no need to establish a connection prior to data transfer. There are two main attributes of UDP: *simple* and *fast*. It is a ***simple*** protocol that uses a very straight-forward messaging structure that is similar to the message format used by many other TCP/IP protocols (in contrast to the more complex data structures—streams and segments—used by TCP). In fact, the only real goal of the protocol is to serve as an interface between networking application processes running at the higher layers, and the internetworking capabilities of IP. UDP is a ***fast*** protocol specifically because it doesn't have all the bells and whistles of TCP. This makes it unsuitable for use by many, if not most, typical networking applications. The UDP helps to establish low-latency and loss-tolerating connections over the network. Though Transmission Control Protocol (TCP) is the dominant transport layer protocol used with most of the Internet services; provides assured delivery, reliability, and much more but all these services cost us additional overhead and latency. But for some applications (that do not require reliability, acknowledgment or flow control features at the transport layer), this is exactly what they want from a transport layer protocol: something that takes their data and quickly shuffles it down to the IP layer with a minimum of fuss. For real-time services like computer gaming, voice or video communication, live conferences; we need UDP. Since high performance is needed, UDP permits packets to be dropped instead of processing delayed packets. There is no error checking in UDP, so it also saves bandwidth. User Datagram Protocol (UDP) is more efficient in terms of both latency and bandwidth. In choosing to use UDP, the application writer takes it upon himself or herself to take care of issues such as reliability and retransmissions, if they are needed. This can be a recipe for success or failure, depending on the application and how carefully UDP is used.

UDP Operation

UDP's only real task is to take data from higher-layer protocols and place it in UDP messages, which are then passed down to the Internet Protocol for transmission. The basic steps for transmission using UDP are:

1. **Higher-Layer Data Transfer:** An application sends a message to the UDP software.

2. **UDP Message Encapsulation:** The higher-layer message is encapsulated into the *Data* field of a UDP message. The headers of the UDP message are filled in, including the *Source Port* of the application that sent the data to UDP, and the *Destination Port* of the intended recipient. The checksum value may also be calculated.

3. **Transfer Message To IP:** The UDP message is passed to IP for transmission.

Of course, on reception at the destination device this short procedure is reversed.

UDP Header –

UDP header is an **8-bytes** fixed and simple header, while for TCP it may vary from 20 bytes to 60 bytes. The first 8 Bytes contains all necessary header information and the remaining part consist of data. UDP port number fields are each 16 bits long, therefore the range for port numbers is defined from 0 to 65535; port number 0 is reserved. Port numbers help to distinguish different user requests or processes.

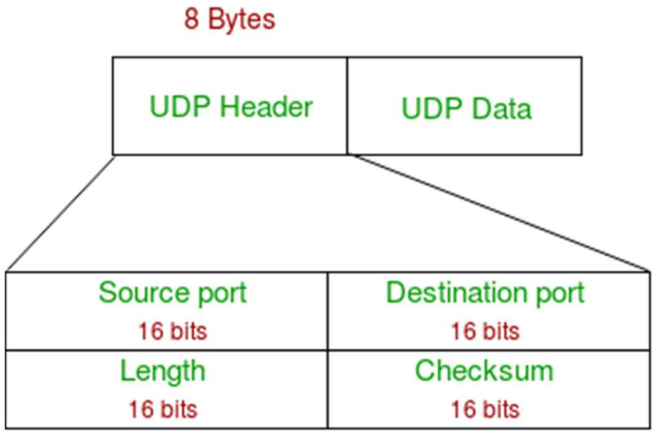

Figure 12: UDP Header

1. **Source Port:** Source Port is a 2 Byte long field used to identify the port number of the source.
2. **Destination Port:** It is a 2 Byte long field, used to identify the port of the destined packet.
3. **Length:** Length is the length of UDP including the header and the data. It is a 16-bits field.
4. **Checksum:** Checksum is 2 Bytes long field. It is the 16-bit one's complement of the one's complement sum of the UDP header, the pseudo-header of information from the IP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

Notes – Unlike TCP, the Checksum calculation is not mandatory in UDP. No Error control or flow control is provided by UDP. Hence UDP depends on IP and ICMP for error reporting. Also UDP provides port numbers so that is can differentiate between users requests.

What UDP Does Not

In fact, UDP is ***so*** simple, that its operation is very often described in terms of what it does ***not*** do, instead of what it does. As a transport protocol, some of the most important things UDP does not do include the following:

- UDP does not establish connections before sending data. It just packages it and… off it goes.

- UDP does not provide acknowledgments to show that data was received.
- UDP does not provide any guarantees that its messages will arrive.
- UDP does not detect lost messages and retransmit them.
- UDP does not ensure that data is received in the same order that they were sent.
- UDP does not provide any mechanism to manage the flow of data between devices, or handle congestion.

Applications of UDP:

- Used for simple request-response communication when the size of data is less and hence there is lesser concern about flow and error control.
- It is a suitable protocol for multicasting as UDP supports packet switching.
- UDP is used for some routing update protocols like RIP(Routing Information Protocol).
- Normally used for real-time applications which can not tolerate uneven delays between sections of a received message.
- UDP is widely used in online gaming, where low latency and high-speed communication is essential for a good gaming experience. Game servers often send small, frequent packets of data to clients, and UDP is well suited for this type of communication as it is fast and lightweight.
- Streaming media applications, such as IPTV, online radio, and video conferencing, use UDP to transmit real-time audio and video data. The loss of some packets can be tolerated in these applications, as the data is continuously flowing and does not require retransmission.
- VoIP (Voice over Internet Protocol) services, such as Skype and WhatsApp, use UDP for real-time voice communication. The delay in voice communication can be noticeable if packets are delayed due to congestion control, so UDP is used to ensure fast and efficient data transmission.
- DNS (Domain Name System) also uses UDP for its query/response messages. DNS queries are typically small and require a quick response time, making UDP a suitable protocol for this application.
- DHCP (Dynamic Host Configuration Protocol) uses UDP to dynamically assign IP addresses to devices on a network. DHCP messages are typically small, and the delay caused by packet loss or retransmission is generally not critical for this application.
- Following implementations uses UDP as a transport layer protocol:
    - NTP (Network Time Protocol)
    - DNS (Domain Name Service)
    - BOOTP, DHCP.
    - NNP (Network News Protocol)
    - Quote of the day protocol
    - TFTP, RTSP, RIP.
- The application layer can do some of the tasks through UDP-
    - Trace Route
    - Record Route
    - Timestamp
- UDP takes a datagram from Network Layer, attaches its header, and sends it to the user. So, it works fast.
- Actually, UDP is a null protocol if you remove the checksum field.
    1. Reduce the requirement of computer resources.
    2. When using the Multicast or Broadcast to transfer.
    3. The transmission of Real-time packets, mainly in multimedia applications.

Advantages of UDP:
1. Speed: UDP is faster than TCP because it does not have the overhead of establishing a connection and ensuring reliable data delivery.

2. Lower latency: Since there is no connection establishment, there is lower latency and faster response time.

3. Simplicity: UDP has a simpler protocol design than TCP, making it easier to implement and manage.

4. Broadcast support: UDP supports broadcasting to multiple recipients, making it useful for applications such as video streaming and online gaming.

5. Smaller packet size: UDP uses smaller packet sizes than TCP, which can reduce network congestion and improve overall network performance.

Disadvantages of UDP:
1. No reliability: UDP does not guarantee delivery of packets or order of delivery, which can lead to missing or duplicate data.

2. No congestion control: UDP does not have congestion control, which means that it can send packets at a rate that can cause network congestion.

3. No flow control: UDP does not have flow control, which means that it can overwhelm the receiver with packets that it cannot handle.

4. Vulnerable to attacks: UDP is vulnerable to denial-of-service attacks, where an attacker can flood a network with UDP packets, overwhelming the network and causing it to crash.

5. Limited use cases: UDP is not suitable for applications that require reliable data delivery, such as email or file transfers, and is better suited for applications that can tolerate some data loss, such as video streaming or online gaming.

UDP PSEUDO HEADER:
- the purpose of using a pseudo-header is to verify that the UDP packet has reached its correct destination
- the correct destination consist of a specific machine and a specific protocol port number within that machine

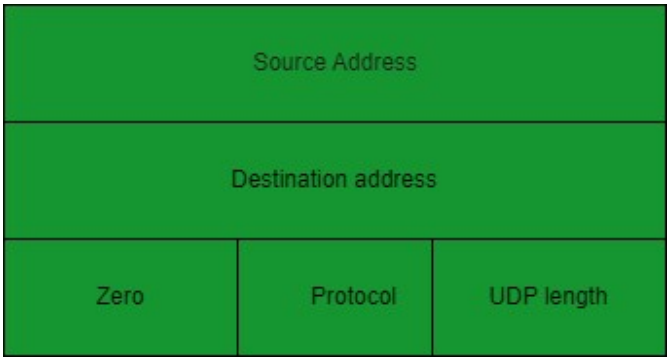

Figure 13: UDP Pseudo Header

UDP pseudo header details:

- the UDP header itself specify only protocol port number; thus, to verify the destination UDP on the sending machine computes a checksum that covers the destination IP address as well as the UDP packet.
- at the ultimate destination, UDP software verifies the checksum using the destination IP address obtained from the header of the IP packet that carried the UDP message.
- if the checksum agrees, then it must be true that the packet has reached the intended destination host as well as the correct protocol port within that host.

User Interface:

A user interface should allow the creation of new receive ports, receive operations on the receive ports that returns the data octets and an indication of source port and source address, and an operation that allows a datagram to be sent, specifying the data, source and destination ports and address to be sent.

IP Interface:

- the UDP module must be able to determine the source and destination internet address and the protocol field from internet header
- one possible UDP/IP interface would return the whole internet datagram including the entire internet header in response to a receive operation
- such an interface would also allow the UDP to pass a full internet datagram complete with header to the IP to send. the IP would verify certain fields for consistency and compute the internet header checksum.
- The IP interface allows the UDP module to interact with the network layer of the protocol stack, which is responsible for routing and delivering data across the network.
- The IP interface provides a mechanism for the UDP module to communicate with other hosts on the network by providing access to the underlying IP protocol.
- The IP interface can be used by the UDP module to send and receive data packets over the network, with the help of IP routing and addressing mechanisms.
- The IP interface provides a level of abstraction that allows the UDP module to interact with the network layer without having to deal with the complexities of IP routing and addressing directly.
- The IP interface also handles fragmentation and reassembly of IP packets, which is important for large data transmissions that may exceed the maximum packet size allowed by the network.
- The IP interface may also provide additional services, such as support for Quality of Service (QoS) parameters and security mechanisms such as IPsec.
- The IP interface is a critical component of the Internet Protocol Suite, as it enables communication between hosts on the internet and allows for the seamless transmission of data packets across the network.