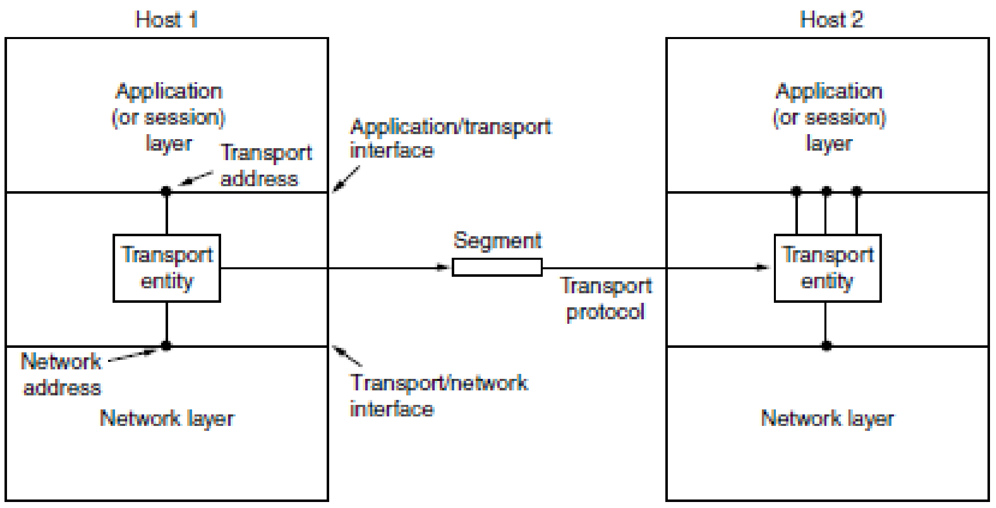
4. Transport Layer

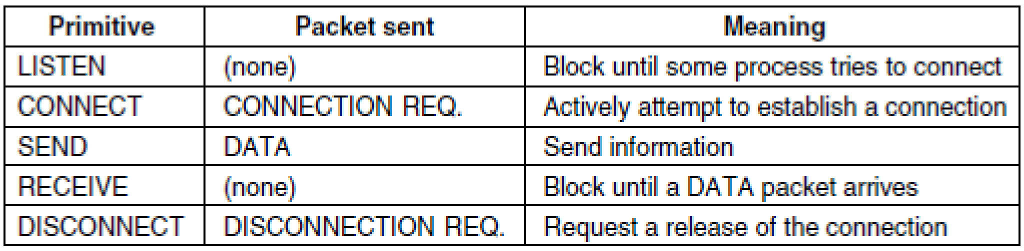
The transport layer facilitating the transmission of data between processes on different machines. It ensures an accurate level of reliability, regardless of the underlying physical networks being used. The primary objective of the transport layer is to provide a data transmission service that is efficient, reliable, and economically viable to its users, often processes located at the application layer. In order to do this, the transport layer utilises the services offered by the network layer. The component responsible for executing the necessary tasks inside the transport layer, including both software and hardware elements, is often referred to as the transport entity. The transport entity has the potential to reside inside the operating system kernel. The interconnection between the network, transport, and application layers is shown in the following diagram.



Similar to the existence of two distinct categories of network service, namely connection-oriented and connectionless, the transport service also encompasses two distinct kinds. The connection-oriented transport service has some similarities to the connection-oriented network service. In all scenarios, the process of establishing connections involves three distinct phases: establishment, data transmission, and release. Addressing and flow control exhibit similarities in both levels. Additionally, it is worth noting that the connectionless transport service has a striking resemblance to the connectionless network service.

Transport Service Primitives:

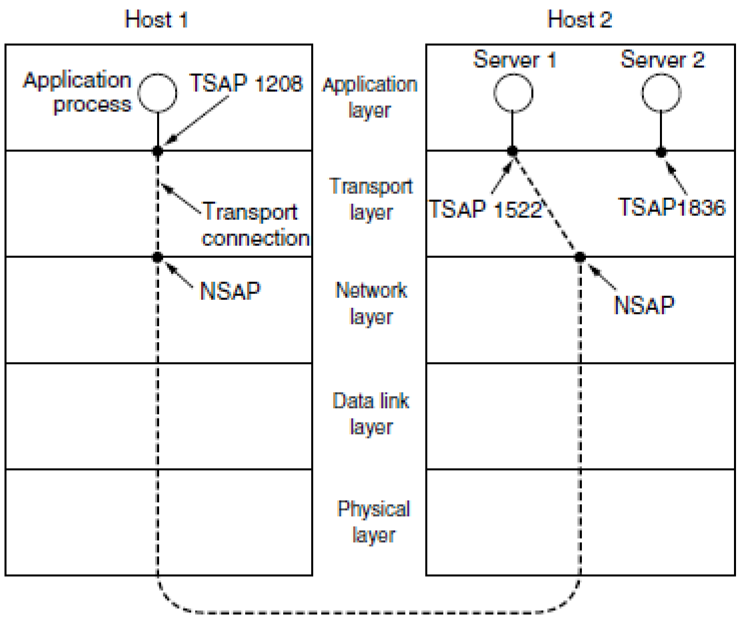
In order to facilitate user access to the transport service, the transport layer is required to provide a set of operations to application programmes, which is often referred to as a transport service interface. Every transport service has its own unique interface.To get insight into the potential utilisation of these fundamental components, let us contemplate an illustrative scenario including an application comprising a central server and several distributed clients. Initially, the server initiates the execution of a LISTEN primitive, often achieved by invoking a library method that triggers a system call. This system call causes the server to enter a blocked state until a client connection is established. When a client desires to establish communication with the server, it initiates the execution of a CONNECT primitive. The transportation entity does this rudimentary action by obstructing the caller's communication and dispatching a packet to the server.The execution of the client's CONNECT function results in the transmission of a CONNECTION REQUEST segment to the server. Upon arrival, the transport entity verifies if the server is in a blocked state on a LISTEN operation, indicating its readiness to handle incoming requests. If this condition is met, the server proceeds to unblock and transmits a CONNECTION ACCEPTED segment in response to the client. Upon the arrival of this particular section, the client becomes unblocked and a connection is successfully formed. The sharing of data is now facilitated by the use of the SEND and RECEIVE primitives. In its most basic manifestation, one party has the capability to execute a "blocking" RECEIVE operation, therefore awaiting the occurrence of a corresponding SEND operation by the other party. Upon the arrival of the section, the receiver becomes unblocked.When a connection becomes unnecessary, it must be withdrawn in order to allocate table space inside the two transport entities. There are two variations of disconnection: asymmetric and symmetric.



Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	Request a release of the connection

Addressing:

When an application process (like a user process) wants to connect to a remote application process, it has to say which one it wants to connect to. Setting up transport addresses that processes can listen to for connection requests is the usual way to do it. These places are known as ports on the Internet. The word "Transport Service Access Point" (TSAP) will be used to refer to a specific address in the transport layer. It's no surprise that the similar ends in the network layer, which are called network layer names, are called NSAPs. An example of an NSAP is an IP address. The NSAPs, TSAPs are connected in the way shown in the next figure.



This is one possible situation for a transport connection:

1. A mail server process joins TSAP 1522 on host 2 and waits for a call to come in. The networking model has nothing to do with how a process connects to a TSAP; it's all up to the local operating system. This could be done with a call like our LISTEN.

2. An application process on server 1 wants to send an email, so it connects to TSAP 1208 and sends a CONNECT request. It says that TSAP 1208 on host 1 is the source and TSAP 1522 on host 2 is the target. After this step is taken, a transport link is made between the application process and the server.

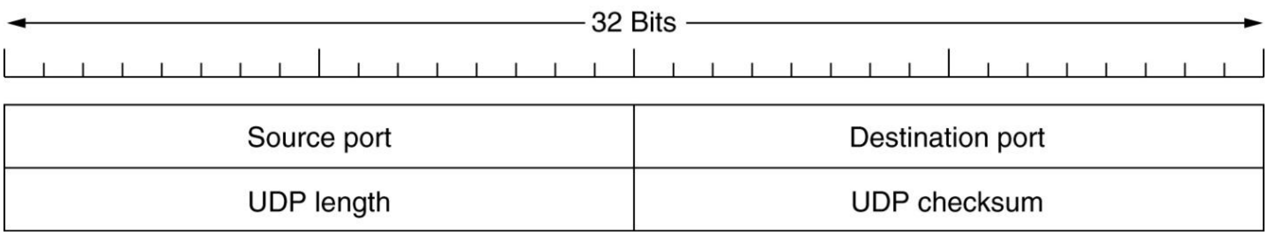
3. The entry process sends the email.

4. The mail service answers and says it will send the message.

5. The connection to the transport is broken down.

The Internet Transport Protocols: UDP

The UDP header:



A user datagram is the name for the packet that the UDP sends.

The source port address is the address of the process that sent the message.

With a destination port address, you can find the address of the process that will receive the message.

The total length field tells you how many bytes the whole user datagram is.

The check sum is a 16-bit number that is used to find mistakes.

An error can be found by UDP, and ICMP can then tell the source that a user datagram was damaged and should be thrown away. It probably makes sense to list some of the things that UDP doesn't do. It does not control the flow of data, fix errors, or send again after getting a bad portion. It does give you a way to talk to the IP protocol, and it also lets you stop multiple processes from using the ports at the same time.Client-server situations are one place where UDP is very useful. A client will often send a short request to the server and wait for a short response.

Remote Procedure Call:

There are similarities between delivering a message to a remote host and receiving a response, much as when you call a function in a programming language.

When a procedure on machine 2 is called by a process on machine 1, the calling process on machine 1 is halted, and the called procedure is executed on machine 2.

Data may go via the parameters from the caller to the callee and return as the procedure's outcome.

The programmer is not able to see any communication travelling. Known as RPC (Remote Procedure Call), this method is currently the foundation for a large number of networking applications.

The goal of remote procedure calls (RPCs) is to mimic local procedure calls as much as feasible.

In its most basic form, calling a remote procedure requires that the client programme be tied to a client stub—a tiny library process that runs in the client's address space and simulates the server method.

Similar to this, a process known as the server stub is connected to the server.

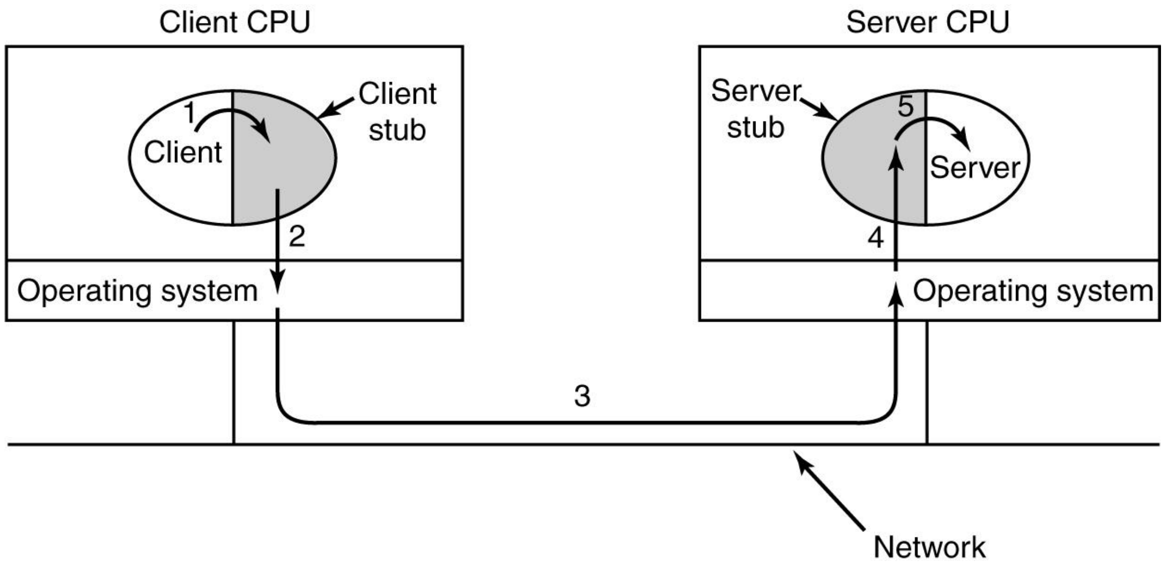


Figure illustrates the actual processes involved in creating an RPC.

• The client calls the client stub in step 1. The arguments for this call are being placed onto the stack in the standard manner; it is a local procedure call.

• The second step involves the client stub sending a system call to transmit the message after stuffing the arguments into it. Marshalling is the process of packing the parameters.

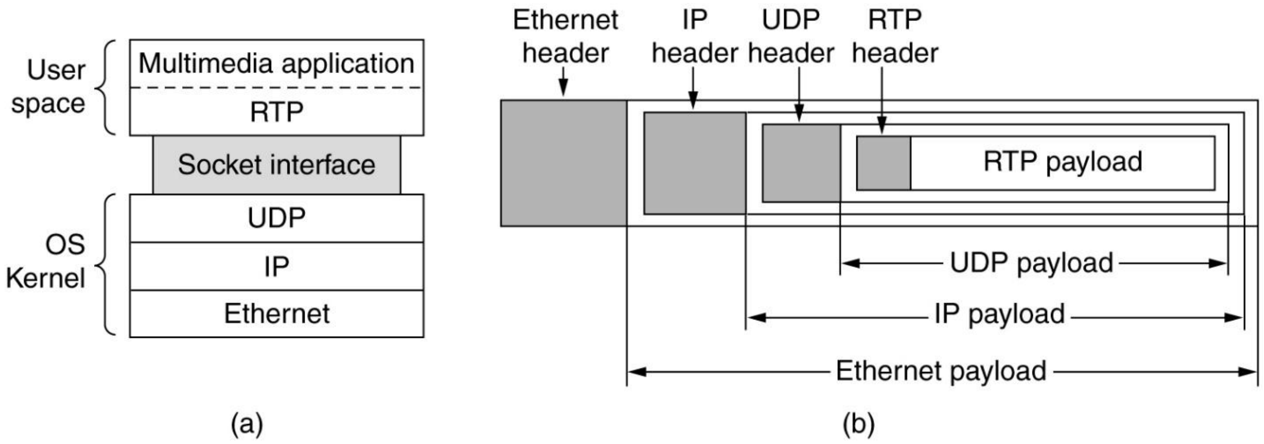
• The kernel transmits the message from the client computer to the server computer in step three.

• The arriving packet is sent to the server stub by the kernel in step four.

• The server stub in step 5 finally calls the server method using the unmarshaled arguments. The response follows the same route in the other direction.

The Real-Time Transport Protocol:

UDP is extensively used in two areas: client-server RPC and real-time multimedia applications, namely in Internet radio, Internet telephony, music-on-demand, videoconferencing, and other multimedia applications.Over time, it became evident that it would be beneficial to have a general real-time transport protocol for many applications.Real-time Transport Protocol (RTP) was thus created.



• As a result of this design, it might be difficult to determine which layer RTP is in.

• Multiplexing numerous real-time data streams onto single stream of UDP packets is the fundamental purpose of RTP.

• The UDP stream may be multicasting, or transmitted to many locations instead of only one (unicasting).

• RTP just utilises regular UDP, therefore unless certain standard IP quality-of-service features are set, routers do not handle its packets differently.

• Specifically, there are no further assurances about jitter, delivery, etc.

• The destination can identify if any packets are missing thanks to the numbering system, which assigns each packet delivered in an RTP stream a number one higher than its predecessor.

•The appropriate course of action for the destination in the event that a packet is missing is to use interpolation to estimate the missing data.

Retransmission is not a feasible solution since it is likely that the packet will arrive too late to be of any value.RTP lacks acknowledgements, flow management, error control, and a retransmission request mechanism as a result.

The RTP header:

There are three 32-bit words in it, along with maybe a few extensions.

• The Version field, which is now at 2, is included in the first word.

• The packet has been padded to a multiple of 4 bytes, as indicated by the P bit.

• The number of bytes added is indicated by the last padding byte.

• The presence of an extension header is indicated by the X bit.

• There is no standardised format or semantics for the extension header. The length of the extension is indicated by the first word in the document, which is the sole definition.

• The number of contributing sources is shown in the CC column. 0 to 15 may be used using it.

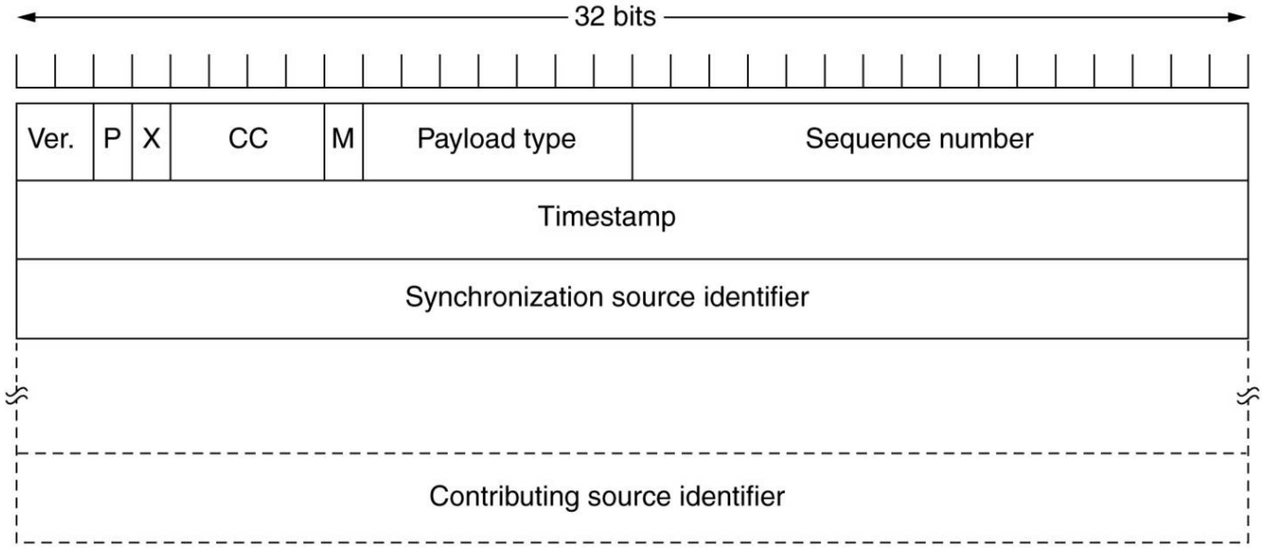
• The M bit is an application-specific marker that may be used to indicate the beginning of a word in an audio channel, the beginning of a video frame, or any other meaningful point for the application.

•Which encoding technique was used—MP3, uncompressed 8-bit audio, etc.—is disclosed in the Payload type field.

• All that the sequence number is, is a counter that is tallied with every transmitted RTP packet. It is used to find dropped packets.

• The stream's source generates the timestamp, which indicates the moment the packet's first sample was taken. This number may aid in jitter reduction.

• The stream to which the packet belongs is indicated by the Synchronisation source identity. It is a technique for multiplexing and demultiplexing many data streams into a single UDP packet stream.


32 bits
Ver.
P
X
CC
M
Payload type
Sequence number
Timestamp
Synchronization source identifier
Contributing source identifier

Real-time Transport Control Protocol:

• Real-time Transport Control Protocol, or RTCP, is the sisterprotocol of RTP. It is covered in RFC 3550, along with RTP, and it deals with synchronisation, feedback, and user interface.

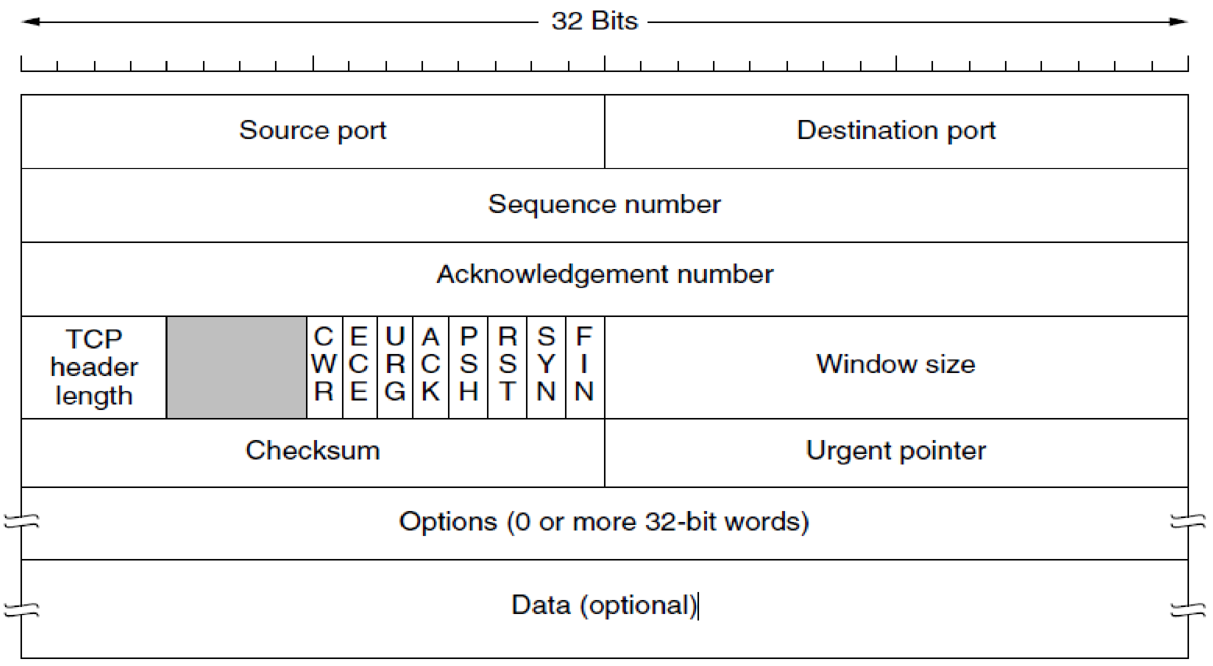
• No media samples are transported by it.

• Feedback on latency, jitter fluctuation in delay or bandwidth, congestion, and other network parameters may be sent to the sources via the first function.

• The encoding process may utilise this information to reduce the data rate during network outages and raise the data rate (and provide higher quality) during network outages.

• The encoding methods may be continually adjusted to deliver the greatest quality feasible given the present conditions by receiving continuous feedback.

TCP Segment Header:



The application programme on the source machine is defined by the source port address.

The application programme on the destination machine is defined by the target port address.

There is a possibility to split a data stream from the application programme into many TCP segments. The data's original data stream location is shown in the sequence number field.

The other communicating device's data is acknowledged by means of a 32-bit acknowledgment number.

Only when the ACK bit is set in the control field (as will be discussed later) is this number valid. Here, it specifies the predicted byte sequence number that comes next. The number of 32-bit (four-byte) words in the TCP header is indicated by the four-bit HLEN field.Up to 15 numbers may be defined by the four bits.The total amount of bytes in the header is obtained by multiplying this by 4. As a result, the header's maximum size is 60 bytes (4x15).40 bytes are available for the options section since the header must be at least 20 bytes in size.

• Set aside. A field with six bits is set aside for future usage.

• When ECN (Explicit Congestion Notification) is used, ECE is configured to signal an ECN-Echo to a TCP sender to instruct it to slow down when the TCP receiver receives a congestion indication from the network. • CWR and ECE are used to indicate congestion.

• CWR is configured to notify the TCP receiver of the Congestion Window Reduced so that it is aware of the sender's slowdown and may cease transmitting the ECN-Echo.

• If the Urgent pointer is being used, URG is set to 1. The urgent data should be located at a byte offset from the current sequence number, which is indicated by the Urgent pointer. This feature replaces interrupt messages.

• To show that the acknowledgement number is legitimate, the ACK bit is set to 1.

• The Acknowledgment number field is disregarded if ACK is 0, indicating that the segment does not include an acknowledgment.

• PUSHEd data is indicated by the PSH bit. Please provide the data to the application as soon as it arrives, rather than buffering it until a complete buffer has been received, as the receiver could normally do for efficiency.

• A connection that has gotten confused because of a host crash or another issue may be reset using the RST bit. Rejecting an invalid segment or an attempt to establish a connection are other uses for it.

• Connections are established using the SYN bit.

• SYN = 1 and ACK = 0 in the connection request indicate that the piggyback acknowledgment field is not being used.

• The connection reply has SYN = 1 and ACK = 1, indicating that it does include an acknowledgment.

• The ACK bit is used to differentiate between the two options that the SYN bit indicates: connection request and connection accepted.

• To cut a connection, use the FIN bit.

• It indicates that there are no more data for the sender to convey.

•Sequence numbers ensure that the SYN and FIN segments are handled in the proper order.

• Size of the window. The size of the sliding window is defined by the window, a 16-bit parameter.

• The checksum. A 16-bit field called the checksum is used for error detection.

• An urgent arrow. The header's last mandatory field is this one. Only when the URG bit in the control field is set is its value valid. In this instance, the sender is alerting the recipient to the urgent material included in the segment's data section. This marker indicates when urgent data ends and regular data begin.

• Padding and options. The optional fields are defined in the remaining TCP header. They are used for alignment or to provide the recipient with further information.

TCP Connection Establishment:

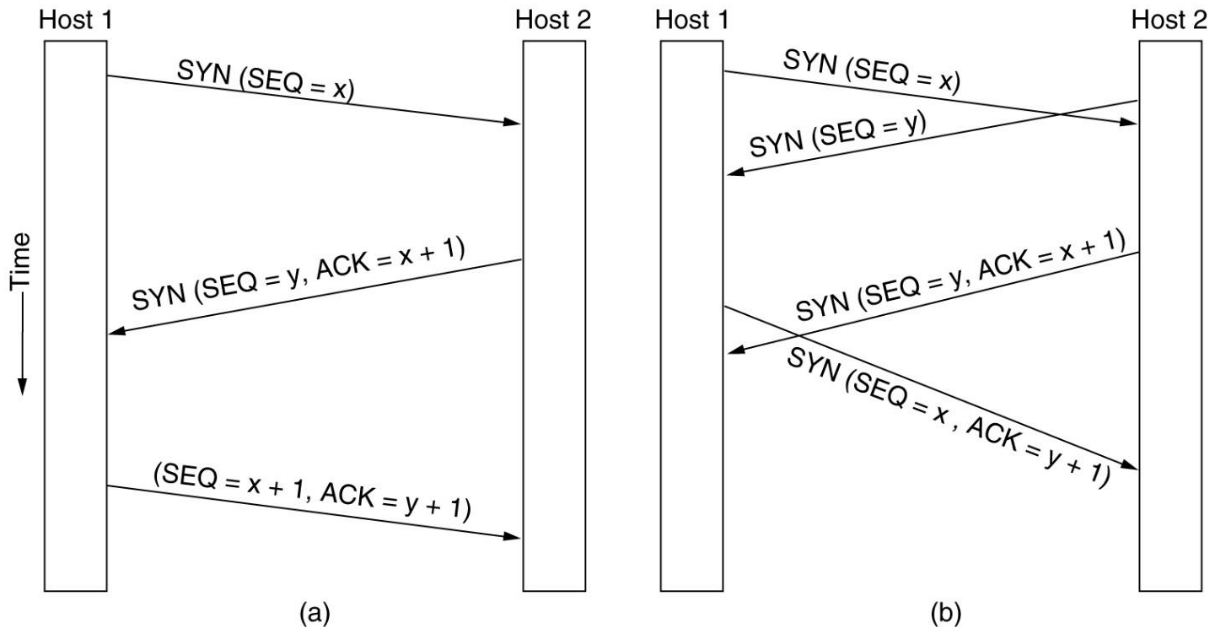
The three-way handshake is used in TCP to create connections.

• To create a connection, one side, says the server, uses the LISTEN and ACCEPT primitives to execute while waiting for an incoming connection. It may specify a specific source or it can accept connections from anybody.

• The other party, let's say the client, uses the CONNECT primitive to establish a connection. It provides the IP address and port it wishes to use, the largest TCP segment size it will allow, and optionally some user data (password, for example). The CONNECT primitive waits for a response after sending a TCP segment with the ACK bit off and SYN bit on.

• The TCP entity at the destination verifies whether a process has performed a LISTEN on the port specified in the Destination port field when this segment reaches there.

• If not, it rejects the connection by sending a reply with the RST bit set.


Host 1
Host 2
SYN (SEQ = x)
SYN (SEQ = y, ACK = x + 1)
(SEQ = x + 1, ACK = y + 1)
Time
(a)
Host 1
Host 2
SYN (SEQ = x)
SYN (SEQ = y)
SYN (SEQ = y, ACK = x + 1)
SYN (SEQ = x , ACK = y + 1)
(b)

(a) Establishing a TCP connection in the typical scenario.

(b) Both parties establishing the connection at the same time.

TCP Connection Release:

• Every single simplex connection is released apart from its siblings.

• Either side may send a TCP segment with the FIN bit set, indicating that it has finished transmitting data, to stop a connection.

• The direction is closed to fresh data after the FIN is recognised.

On the other hand, data may flow in the other way forever.

• The connection is relinquished when both directions have been cut off.

• Timer usage prevents the two-army dilemma.

• The sender of the FIN releases the connection if they do not get a response to their message within the allotted two maximum packet lifetimes. Eventually, the opposing side will also time out after realising that no one seems to be listening to it anymore.