Transport Layer responsibilities

Last Updated: 25-09-2019

Prerequisite – [Layers of OSI Model](https://www.geeksforgeeks.org/layers-osi-model/)  
Transport Layer is the second layer of the TCP/IP model. It is an **end-to-end** layer used to deliver messages to a host. It is termed as an end-to-end layer because it provides a point-to-point connection **rather than** hop-to- hop, between the source host and destination host to deliver the services reliably. The unit of data encapsulation in Transport Layer is a segment.

The standard protocols used by Transport Layer to enhance its functionalities are TCP(Transmission Control Protocol), UDP( User Datagram Protocol), DCCP( Datagram Congestion Control Protocol) etc.  
Various responsibilities of a Transport Layer –

* **Process to process delivery –**  
  While Data Link Layer requires the MAC address (48 bits address contained inside the Network Interface Card of every host machine) of source-destination hosts to correctly deliver a frame and Network layer requires the IP address for appropriate routing of packets , in a similar way Transport Layer requires a Port number to correctly deliver the segments of data to the correct process amongst the multiple processes running on a particular host. A **port number** is a 16 bit address used to identify any client-server program uniquely.
* **End-to-end Connection between hosts –**  
  The transport layer is also responsible for creating the end-to-end Connection between hosts for which it mainly uses TCP and UDP. TCP is a secure, connection- orientated protocol which uses a handshake protocol to establish a robust connection between two end- hosts. TCP ensures reliable delivery of messages and is used in various applications. UDP, on the other hand, is a stateless and unreliable protocol which ensures best-effort delivery. It is suitable for the applications which have little concern with flow or error control and requires to send the bulk of data like video conferencing. It is often used in multicasting protocols.
* **Multiplexing and Demultiplexing –**  
  Multiplexing allows simultaneous use of different applications over a network which is running on a host. The transport layer provides this mechanism which enables us to send packet streams from various applications simultaneously over a network. The transport layer accepts these packets from different processes differentiated by their port numbers and passes them to the network layer after adding proper headers. Similarly, Demultiplexing is required at the receiver side to obtain the data coming from various processes. Transport receives the segments of data from the network layer and delivers it to the appropriate process running on the receiver’s machine.
* **Congestion Control –**  
  Congestion is a situation in which too many sources over a network attempt to send data and the router buffers start overflowing due to which loss of packets occur. As a result retransmission of packets from the sources increases the congestion further. In this situation, the Transport layer provides Congestion Control in different ways. It uses **open loop** congestion control to prevent the congestion and **closed loop** congestion control to remove the congestion in a network once it occurred. TCP provides AIMD- additive increase multiplicative decrease, leaky bucket technique for congestion control.
* **Data integrity and Error correction –**  
  Transport layer checks for errors in the messages coming from application layer by using error detection codes, computing checksums, it checks whether the received data is not corrupted and uses the ACK and NACK services to inform the sender if the data has arrived or not and checks for the integrity of data.
* **Flow control –**  
  The transport layer provides a flow control mechanism between the adjacent layers of the TCP/IP model. TCP also prevents data loss due to a fast sender and slow receiver by imposing some flow control techniques. It uses the method of sliding window protocol which is accomplished by the receiver by sending a window back to the sender informing the size of data it can receive.

# DCN - Transmission Control Protocol

Advertisements

[Previous Page](https://www.tutorialspoint.com/data_communication_computer_network/transport_layer_introduction.htm)

[Next Page](https://www.tutorialspoint.com/data_communication_computer_network/user_datagram_protocol.htm)

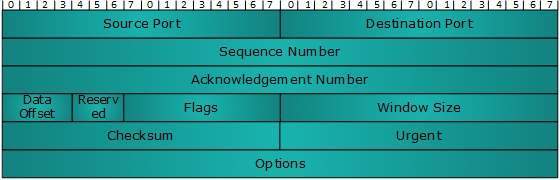
The transmission Control Protocol (TCP) is one of the most important protocols of Internet Protocols suite. It is most widely used protocol for data transmission in communication network such as internet.

## 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.
  + **ECE** -It has two meanings:
    - If SYN bit is clear to 0, then ECE means that the IP packet has its CE (congestion experience) bit set.
    - If SYN bit is set to 1, ECE means that the device is ECT capable.
  + **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.

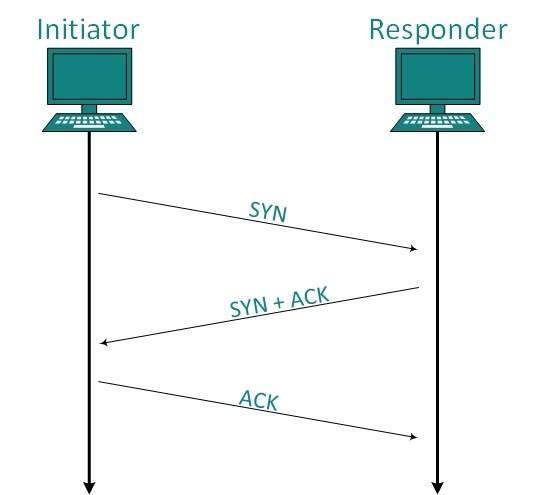
## Addressing

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.



### 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.

## Bandwidth Management

TCP uses the concept of window size to accommodate the need of Bandwidth management. Window size tells the sender at the remote end, the number of data byte segments the receiver at this end can receive. TCP uses slow start phase by using window size 1 and increases the window size exponentially after each successful communication.

For example, the client uses windows size 2 and sends 2 bytes of data. When the acknowledgement of this segment received the windows size is doubled to 4 and next sent the segment sent will be 4 data bytes long. When the acknowledgement of 4-byte data segment is received, the client sets windows size to 8 and so on.

If an acknowledgement is missed, i.e. data lost in transit network or it received NACK, then the window size is reduced to half and slow start phase starts again.

## Error Control &and Flow Control

TCP uses port numbers to know what application process it needs to handover the data segment. Along with that, it uses sequence numbers to synchronize itself with the remote host. All data segments are sent and received with sequence numbers. The Sender knows which last data segment was received by the Receiver when it gets ACK. The Receiver knows about the last segment sent by the Sender by referring to the sequence number of recently received packet.

If the sequence number of a segment recently received does not match with the sequence number the receiver was expecting, then it is discarded and NACK is sent back. If two segments arrive with the same sequence number, the TCP timestamp value is compared to make a decision.

## Multiplexing

The technique to combine two or more data streams in one session is called Multiplexing. When a TCP client initializes a connection with Server, it always refers to a well-defined port number which indicates the application process. The client itself uses a randomly generated port number from private port number pools.

Using TCP Multiplexing, a client can communicate with a number of different application process in a single session. For example, a client requests a web page which in turn contains different types of data (HTTP, SMTP, FTP etc.) the TCP session timeout is increased and the session is kept open for longer time so that the three-way handshake overhead can be avoided.

This enables the client system to receive multiple connection over single virtual connection. These virtual connections are not good for Servers if the timeout is too long.

## Congestion Control

When large amount of data is fed to system which is not capable of handling it, congestion occurs. TCP controls congestion by means of Window mechanism. TCP sets a window size telling the other end how much data segment to send. TCP may use three algorithms for congestion control:

* Additive increase, Multiplicative Decrease
* Slow Start
* Timeout React

## Timer Management

TCP uses different types of timer to control and management various tasks:

### Keep-alive timer:

* This timer is used to check the integrity and validity of a connection.
* When keep-alive time expires, the host sends a probe to check if the connection still exists.

### Retransmission timer:

* This timer maintains stateful session of data sent.
* If the acknowledgement of sent data does not receive within the Retransmission time, the data segment is sent again.

### Persist timer:

* TCP session can be paused by either host by sending Window Size 0.
* To resume the session a host needs to send Window Size with some larger value.
* If this segment never reaches the other end, both ends may wait for each other for infinite time.
* When the Persist timer expires, the host re-sends its window size to let the other end know.
* Persist Timer helps avoid deadlocks in communication.

### Timed-Wait:

* After releasing a connection, either of the hosts waits for a Timed-Wait time to terminate the connection completely.
* This is in order to make sure that the other end has received the acknowledgement of its connection termination request.
* Timed-out can be a maximum of 240 seconds (4 minutes).

## Crash Recovery

TCP is very reliable protocol. It provides sequence number to each of byte sent in segment. It provides the feedback mechanism i.e. when a host receives a packet, it is bound to ACK that packet having the next sequence number expected (if it is not the last segment).

When a TCP Server crashes mid-way communication and re-starts its process it sends TPDU broadcast to all its hosts. The hosts can then send the last data segment which was never unacknowledged and carry onwards

# What is Piggybacking in Networking?

[Computer Network](https://www.tutorialspoint.com/questions/category/Computer-Network)[Computer Engineering](https://www.tutorialspoint.com/questions/category/Computer-Engineering)[MCA](https://www.tutorialspoint.com/questions/category/MCA)

In reliable full - duplex data transmission, the technique of hooking up acknowledgments onto outgoing data frames is called piggybacking.

## Why Piggybacking?

Communications are mostly full – duplex in nature, i.e. data transmission occurs in both directions. A method to achieve full – duplex communication is to consider both the communication as a pair of simplex communication. Each link comprises a forward channel for sending data and a reverse channel for sending acknowledgments.

However, in the above arrangement, traffic load doubles for each data unit that is transmitted. Half of all data transmission comprise of transmission of acknowledgments.

So, a solution that provides better utilization of bandwidth is piggybacking. Here, sending of acknowledgment is delayed until the next data frame is available for transmission. The acknowledgment is then hooked onto the outgoing data frame. The data frame consists of an ack field. The size of the ack field is only a few bits, while an acknowledgment frame comprises of several bytes. Thus, a substantial gain is obtained in reducing bandwidth requirement.

## Working Principle

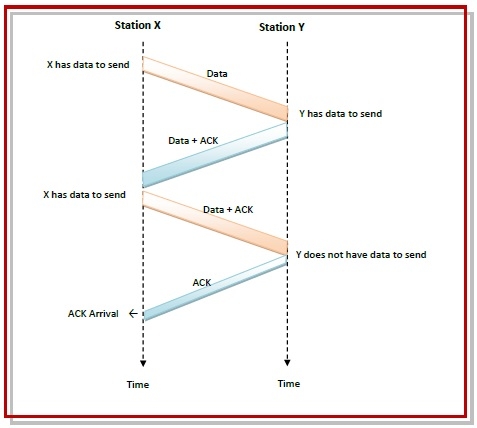
Suppose that there are two communication stations X and Y. The data frames transmitted have an acknowledgment field, ack field that is of a few bits length. Additionally, there are frames for sending acknowledgments, ACK frames. The purpose is to minimize the ACK frames.

The three principles governing piggybacking when the station X wants to communicate with station Y are −

* If station X has both data and acknowledgment to send, it sends a data frame with the ack field containing the sequence number of the frame to be acknowledged.
* If station X has only an acknowledgment to send, it waits for a finite period of time to see whether a data frame is available to be sent. If a data frame becomes available, then it piggybacks the acknowledgment with it. Otherwise, it sends an ACK frame.
* If station X has only a data frame to send, it adds the last acknowledgment with it. The station Y discards all duplicate acknowledgments. Alternatively, station X may send the data frame with the ack field containing a bit combination denoting no acknowledgment.

## Example

The following diagram illustrates the three scenario −



## How TCP Three-way handshake works (SYN, SYN-ACK, ACK)

Before the sending device and the receiving device start the exchange of data, both devices need to be synchronized. During the [TCP](https://www.omnisecu.com/tcpip/transmission-control-protocol-tcp.php) initialization process, the sending device and the receiving device exchange a few control packets for synchronization purposes. This exchange is known as Three-way handshake.

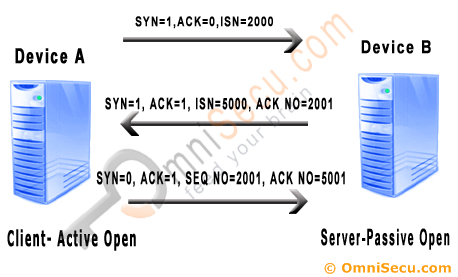
The Three-way handshake begins with the initiator sending a [TCP segment](https://www.omnisecu.com/tcpip/tcp-header.php) with the [SYN](https://www.omnisecu.com/tcpip/tcp-header.php) control bit flag set.

TCP allows one side to establish a connection. The other side may either accept the connection or refuse it. If we consider this from application layer point of view, the side that is establishing the connection is the client and the side waiting for a connection is the server.

TCP identifies two types of OPEN calls:

Active Open. In an Active Open call a device (client process) using TCP takes the active role and initiates the connection by sending a [TCP SYN message](https://www.omnisecu.com/tcpip/tcp-header.php)to start the connection.

Passive Open A passive OPEN can specify that the device (server process) is waiting for an active OPEN from a specific client. It does not generate any [TCP message segment](https://www.omnisecu.com/tcpip/tcp-header.php). The server processes listening for the clients are in Passive Open mode.



**TCP Three-way Handshake**

Step 1. Device A (Client) sends a [TCP segment](https://www.omnisecu.com/tcpip/tcp-header.php) with SYN = 1, ACK = 0, ISN (Initial Sequence Number) = 2000.

An Initial Sequence Number (ISN) is a random [Sequence Number](https://www.omnisecu.com/tcpip/tcp-header.php), allocated for the first packet in a new TCP connection.

The Active Open device (Device A) sends a segment with the[SYN flag](https://www.omnisecu.com/tcpip/tcp-header.php) set to 1, [ACK flag](https://www.omnisecu.com/tcpip/tcp-header.php) set to 0 and an Initial Sequence Number 2000 (For Example), which marks the beginning of the sequence numbers for data that device A will transmit. [SYN](https://www.omnisecu.com/tcpip/tcp-header.php) is short for SYNchronize. [SYN flag](https://www.omnisecu.com/tcpip/tcp-header.php) announces an attempt to open a connection.

Step 2. Device B (Server) receives Device A's [TCP segment](https://www.omnisecu.com/tcpip/tcp-header.php) and returns a [TCP segment](https://www.omnisecu.com/tcpip/tcp-header.php) with SYN = 1, ACK = 1, ISN = 5000 (Device B's [Initial Sequence Number](https://www.omnisecu.com/tcpip/tcp-header.php)), [Acknowledgment Number](https://www.omnisecu.com/tcpip/tcp-header.php) = 2001 (2000 + 1, the next [sequence number](https://www.omnisecu.com/tcpip/tcp-header.php) Device B expecting from Device A).

Step 3. Device A sends a [TCP segment](https://www.omnisecu.com/tcpip/tcp-header.php) to Device B that acknowledges receipt of Device B's ISN, With flags set as SYN = 0, ACK = 1, Sequence number = 2001, [Acknowledgment number](https://www.omnisecu.com/tcpip/tcp-header.php)= 5001 (5000 + 1, the next sequence number Device A expecting from Device B)

This handshaking technique is referred to as TCP Three-way handshake or SYN, SYN-ACK, ACK.

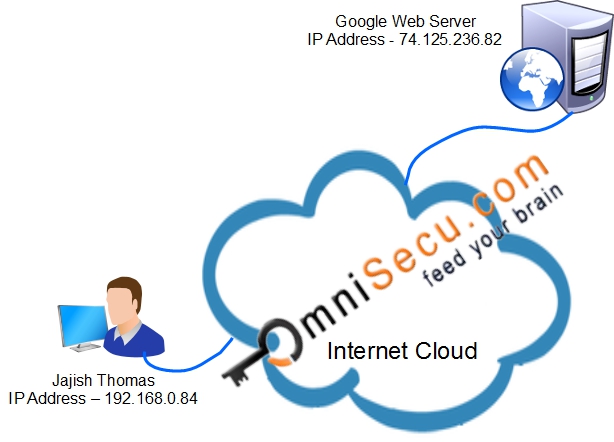
After the Three-way handshake, the connection is open and the participant computers start sending data using the agreed [sequence and acknowledge numbers](https://www.omnisecu.com/tcpip/tcp-header.php).

## TCP Three-way Handshake - A Real World Example

Let us dissect the process of TCP three-way handshake, using a real world example. I am including the Wireshark screen shots also, to understand the concepts more clearly.

I am sitting at my desk and I need to open Google web page for a search. The URL is https://www.google.com. Protocol used is HTTPS over TCP and Destination TCP Port number is TCP 443. I have entered the URL in my browser and hit the "Enter" key.

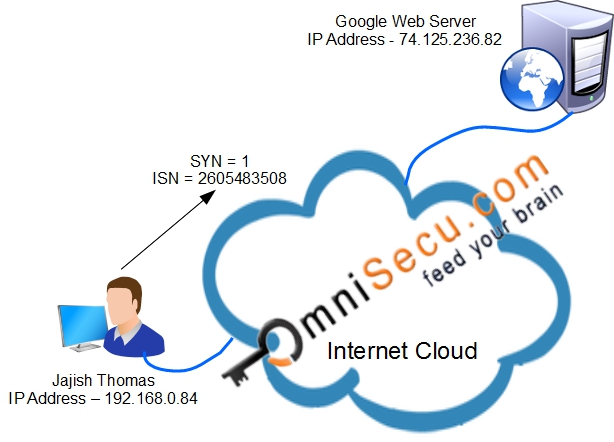
Topology and IP addresses for TCP Three-way handshake study are shown below.

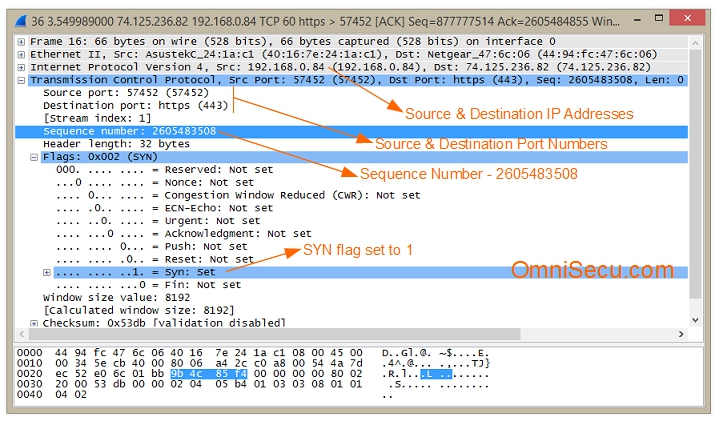


### Step 1 - TCP Three-way Handshake SYN

First step in establishing a reliable TCP connection (using Three-way handshake) between my computer and the Web Server is to send a [TCP segment](https://www.omnisecu.com/tcpip/tcp-header.php), with [SYN flag](https://www.omnisecu.com/tcpip/tcp-header.php) set to 1, to the Web Server. All other [TCP Flags](https://www.omnisecu.com/tcpip/tcp-header.php) are set to 0. The TCP Segment with SYN Flag set to 1, is informing the Web Server that My computer wants to open a TCP session with the Web Server.

The Initial Sequence Number (ISN) generated by the TCP/IP protocol stack in my computer is 2605483508.





Key values for the TCP Three-Way handshake SYN request (Sent from My Computer to Web Server) are shown in below table.

|  |  |
| --- | --- |
| Sequence Number | 2605483508 (My Initial Sequence Number) |
| SYN flag | 1 |
| ACK flag | 0 |
| Source IP Address | 192.168.0.84 (My IP Address) |
| Destination IP Address | 74.125.236.82 (Web Server's IP Address) |
| Source TCP Port Number | 57452 ([Private Port Number](https://www.omnisecu.com/tcpip/tcp-port-numbers.php), between 49152–65535, opened by the TCP/IP protocol stack running in my computer for this connection) |
| Destination TCP Port Number | 443 ([Well-known port number](https://www.omnisecu.com/tcpip/tcp-port-numbers.php) for HTTPS, where the Web Server is listening for incoming requests) |

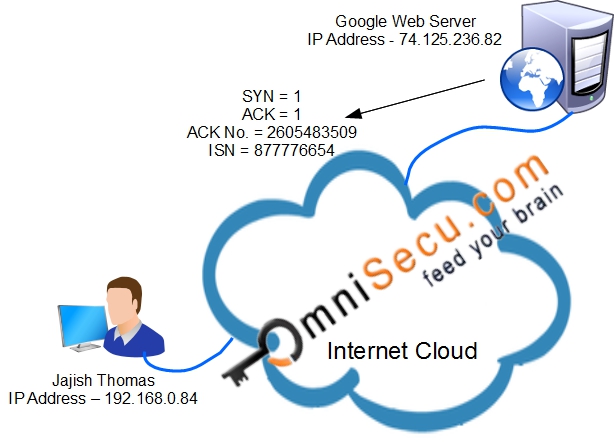
### Step 2 - TCP Three-way Handshake SYN-ACK

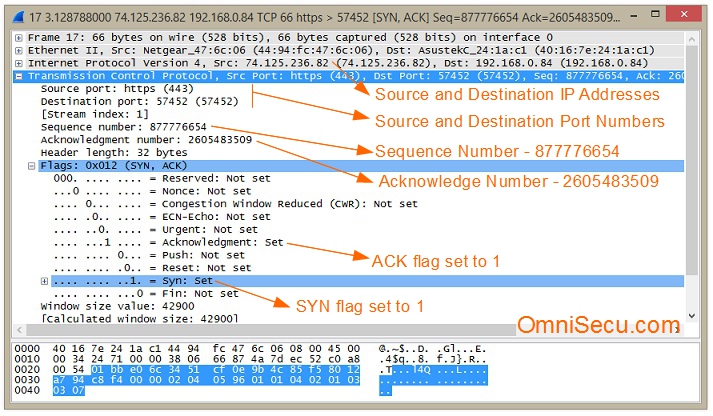
After receiving the SYN request from my computer, the Web Server replied back with a TCP SYN-ACK packet. In a TCP SYN-ACK packet, both SYN and ACK flags are set to 1 and the remaining TCP Flags are set to 0.

The SYN Flag set to 1 is to inform my computer that the Web Server is also willing to open a TCP session with my computer. The ACK Flag set to 1 is to Acknowledge previous TCP SYN reqest.

Initial Sequence Number (ISN) generated by the TCP/IP protocol stack running on the Web server is 877776654. The [Acknowledgement number](https://www.omnisecu.com/tcpip/tcp-header.php) 2605483509 is to inform My Computer that the previous data was received successfully. [Acknowledgement number](https://www.omnisecu.com/tcpip/tcp-header.php) poins that the next Sequence Number of the TCP segment from my computer to the Web Server should be 2605483509.

Note that the [Acknowledgment number](https://www.omnisecu.com/tcpip/tcp-header.php) is increased by 1 if [SYN, ACK or FIN flags](https://www.omnisecu.com/tcpip/tcp-header.php) are set in a received TCP packet. If the TCP packet is carrying data, the Acknowledgment number is increased according to the size of the data the packet is carrying.





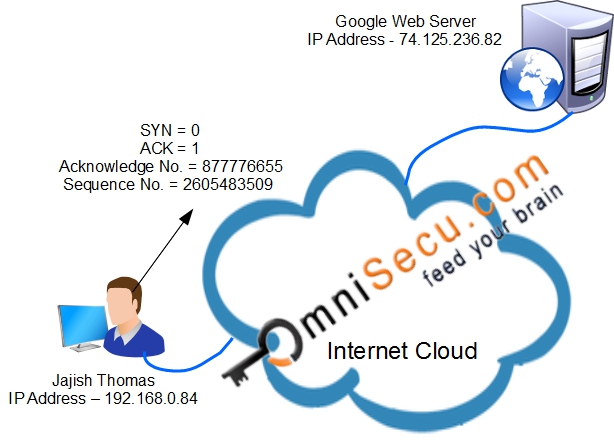
Key values for the TCP Three-Way handshake SYN-ACK message (Sent from Web Server to My Computer) are shown in below table.

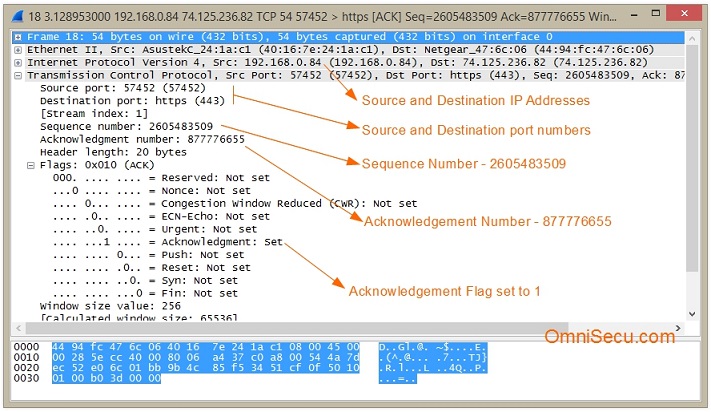
|  |  |
| --- | --- |
| Sequence Number | 877776654 (Web Server's Initial Sequence Number) |
| Acknowledgement number | 2605483509 (Web Server's Acknowledgement number) |
| SYN flag | 1 |
| ACK flag | 1 |
| Source IP Address | 74.125.236.82 (Web Server's IP Address) |
| Destination IP Address | 192.168.0.84 (My IP Address) |
| Source TCP Port Number | 443 ([Well-known port](https://www.omnisecu.com/tcpip/tcp-port-numbers.php) number for HTTPS, where the Web Server is listening for incoming requests) |
| Destination TCP Port Number | 57452 ([Private Port Number](https://www.omnisecu.com/tcpip/tcp-port-numbers.php), between 49152–65535, opened by the TCP/IP protocol stack running in my computer for this connection) |

### Step 3 - TCP Three-way Handshake ACK

The final step in establishing a TCP reliable connection using Three-Way handshake is to send back a TCP ACK packet to the Web Server, for the SYN-ACK packet we received in last step.

My Sequence number is 2605483509, as specified as the [Acknowledgement number](https://www.omnisecu.com/tcpip/tcp-header.php) in the previous SYN-ACK packet. My Acknowledgement number to the Web Server is 877776655.





Key values for the TCP Three-Way handshake ACK message (Sent from My Computer to Web Server) are shown in below table.

|  |  |
| --- | --- |
| Sequence Number | 2605483509 |
| Acknowledgement number | 877776655 |
| SYN flag | 0 |
| ACK flag | 1 |
| Source IP Address | 192.168.0.84 (My IP Address) |
| Destination IP Address | 74.125.236.82 (Web Server's IP Address) |
| Source TCP Port Number | 57452 ([Private Port Number](https://www.omnisecu.com/tcpip/tcp-port-numbers.php), between 49152-65535, opened by the TCP/IP protocol stack running in my computer for this connection) |
| Destination TCP Port Number | 443 ([Well-known port number](https://www.omnisecu.com/tcpip/tcp-port-numbers.php) for HTTPS, where the Web Server is listening for incoming requests) |

Once the TCP Three-way handshake ACK message is sent, TCP connection is Established and the computers can now start communicate reliably using TCP.

# TCP Connection Termination

Last Updated: 12-08-2019

In [TCP 3-way Handshake Process](https://www.geeksforgeeks.org/computer-network-tcp-3-way-handshake-process/) we studied that how connection establish between client and server in Transmission Control Protocol (TCP) using **SYN** bit segments. In this article we will study about how TCP close connection between Client and Server. Here we will also need to send bit segments to server which **FIN** bit is set to 1.

11

How mechanism works In TCP :

1. **Step 1 (FIN From Client) –** Suppose that the client application decides it wants to close the connection. (Note that the server could also choose to close the connection). This causes the client send a TCP segment with the **FIN** bit set to **1** to server and to enter the **FIN\_WAIT\_1** state. While in the **FIN\_WAIT\_1** state, the client waits for a TCP segment from the server with an acknowledgment (ACK).
2. **Step 2 (ACK From Server) –** When Server received FIN bit segment from Sender (Client), Server Immediately send acknowledgement (ACK) segment to the Sender (Client).
3. **Step 3 (Client waiting) –** While in the **FIN\_WAIT\_1** state, the client waits for a TCP segment from the server with an acknowledgment. When it receives this segment, the client enters the **FIN\_WAIT\_2** state. While in the **FIN\_WAIT\_2** state, the client waits for another segment from the server with the FIN bit set to 1.
4. **Step 4 (FIN from Server) –** Server sends FIN bit segment to the Sender(Client) after some time when Server send the ACK segment (because of some closing process in the Server).
5. **Step 5 (ACK from Client) –** When Client receive FIN bit segment from the Server, the client acknowledges the server’s segment and enters the **TIME\_WAIT** state. The **TIME\_WAIT** state lets the client resend the final acknowledgment in case the **ACK** is lost.The time spent by client in the **TIME\_WAIT** state is depend on their implementation, but their typical values are 30 seconds, 1 minute, and 2 minutes. After the wait, the connection formally closes and all resources on the client side (including port numbers and buffer data) are released.

In the below Figures illustrates the series of states visited by the server-side and also Client-side, assuming the client begins connection tear-down.In these two state-transition figures, we have only shown how a TCP connection is normally established and shut-down.

TCP states visited by ClientSide –  
https://media.geeksforgeeks.org/wp-content/uploads/CN-1.png

TCP states visited by ServerSide –  
https://media.geeksforgeeks.org/wp-content/uploads/CN-2.png

# TCP Congestion Control

Last Updated: 12-08-2019

**Prerequisites –** [Basic Congestion control knowledge](https://www.geeksforgeeks.org/computer-networks-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.

**Congestion policy in TCP –**

1. Slow Start Phase: starts slowly increment is exponential to threshold
2. Congestion Avoidance Phase: After reaching the threshold increment is by 1
3. Congestion Detection Phase: Sender goes back to Slow start phase or Congestion avoidance phase.

**Slow Start Phase : exponential increment –** In this phase after every RTT the congestion window size increments exponentially.

Initially cwnd = 1

After 1 RTT, cwnd = 2^(1) = 2

2 RTT, cwnd = 2^(2) = 4

3 RTT, cwnd = 2^(3) = 8

**Congestion Avoidance Phase : additive increment –** This phase starts after the threshold value also denoted as ssthresh. The size of cwnd(congestion window) increases additive. After each RTT cwnd = cwnd + 1.

Initially cwnd = i

After 1 RTT, cwnd = i+1

2 RTT, cwnd = i+2

3 RTT, cwnd = i+3

**Congestion Detection Phase : multiplicative decrement –** If congestion occurs, the congestion window size is decreased. The only way a sender can guess that congestion has occurred is the need to retransmit a segment. Retransmission is needed to recover a missing packet which is assumed to have been dropped by a router due to congestion. Retransmission can occur in one of two cases: when the RTO timer times out or when three duplicate ACKs are received.

* **Case 1 : Retransmission due to Timeout –** In this case congestion possibility is high.

(a) ssthresh is reduced to half of the current window size.  
(b) set cwnd = 1  
(c) start with slow start phase again.

* **Case 2 : Retransmission due to 3 Acknowledgement Duplicates –** In this case congestion possibility is less.

(a) ssthresh value reduces to half of the current window size.  
(b) set cwnd= ssthresh  
(c) start with congestion avoidance phase

User Datagram Protocol (UDP)

Last Updated: 12-08-2019

**User Datagram Protocol (UDP)** is a Transport Layer protocol. UDP is a part of Internet Protocol suite, referred as UDP/IP suite. Unlike TCP, it is **unreliable and connectionless protocol.** So, there is no need to establish connection prior to data transfer.

Though Transmission Control Protocol (TCP) is the dominant transport layer protocol used with most of Internet services; provides assured delivery, reliability and much more but all these services cost us with additional overhead and latency. Here, UDP comes into picture. For the realtime 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 save bandwidth.  
User Datagram Protocol (UDP) is more efficient in terms of both latency and bandwidth.

**UDP Header –**

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

1. **Source Port :** Source Port is 2 Byte long field used to identify port number of source.
2. **Destination Port :** It is 2 Byte long field, used to identify the port of destined packet.
3. **Length :** Length is the length of UDP including header and the data. It is 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, 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, 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.

**Applications of UDP:**

* Used for simple request response communication when size of data is less and hence there is lesser concern about flow and error control.
* It is 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.
* 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, OSPF.
* Application layer can do some of the tasks through UDP-
  + Trace Route
  + Record Route
  + Time stamp
* UDP takes datagram from Network Layer, attach its header and send it to the user. So, it works fast.
* Actually UDP is null protocol if you remove checksum field.

**When to use UDP?**

* + Reduce the requirement of computer resources.
  + When using the Multicast or Broadcast to transfer.
  + The transmission of Real-time packets, mainly in multimedia applications

User datagram protocol (UDP) operates on top of the Internet Protocol (IP) to transmit datagrams over a network. UDP does not require the source and destination to establish a three-way handshake before transmission takes place. Additionally, there is no need for an end-to-end connection.

Since UDP avoids the overhead associated with connections, error checks and the retransmission of missing data, it’s suitable for real-time or high performance applications that don’t require data verification or correction. If verification is needed, it can be performed at the application layer.

UDP is commonly used for Remote Procedure Call (RPC) applications, although RPC can also run on top of TCP. RPC applications need to be aware they are running on UDP, and must then implement their own reliability mechanisms.

## The benefits and downsides of UDP

UDP has a number of benefits for different types of applications, including:

* **No retransmission delays** – UDP is suitable for time-sensitive applications that can’t afford retransmission delays for dropped packets. Examples include Voice over IP (VoIP), online games, and media streaming.
* **Speed** – UDP’s speed makes it useful for query-response protocols such as DNS, in which data packets are small and transactional.
* **Suitable for broadcasts** – UDP’s lack of end-to-end communication makes it suitable for broadcasts, in which transmitted data packets are addressed as receivable by all devices on the internet. UDP broadcasts can be received by large numbers of clients without server-side overhead.

At the same time, UDP’s lack of connection requirements and data verification can create a number of issues when transmitting packets. These include:

* No guaranteed ordering of packets.
* No verification of the readiness of the computer receiving the message.
* No protection against duplicate packets.
* No guarantee the destination will receive all transmitted bytes. UDP, however, does provide a checksum to verify individual packet integrity.

**Differences between TCP and UDP**

| **TRANSMISSION CONTROL PROTOCOL (TCP)** | **USER DATAGRAM PROTOCOL (UDP)** |
| --- | --- |
| TCP is a connection-oriented protocol. Connection-orientation means that the communicating devices should establish a connection before transmitting data and should close the connection after transmitting the data. | UDP is the Datagram oriented protocol. This is because there is no overhead for opening a connection, maintaining a connection, and terminating a connection. UDP is efficient for broadcast and multicast type of network transmission. |
| TCP is reliable as it guarantees delivery of data to the destination router. | The delivery of data to the destination cannot be guaranteed in UDP. |
| TCP provides extensive error checking mechanisms. It is because it provides flow control and acknowledgment of data. | UDP has only the basic error checking mechanism using checksums. |
| Sequencing of data is a feature of Transmission Control Protocol (TCP). this means that packets arrive in-order at the receiver. | There is no sequencing of data in UDP. If ordering is required, it has to be managed by the application layer. |
| TCP is comparatively slower than UDP. | UDP is faster, simpler and more efficient than TCP. |
| Retransmission of lost packets is possible in TCP, but not in UDP. | There is no retransmission of lost packets in User Datagram Protocol (UDP). |
| TCP has a (20-80) bytes variable length header. | UDP has a 8 bytes fixed length header. |
| TCP is heavy-weight. | UDP is lightweight. |
|  |  |
| TCP doesn’t supports Broadcasting. | UDP supports Broadcasting. |
| TCP is used by HTTP, HTTPs, FTP, SMTP and Telnet. | UDP is used by DNS, DHCP, TFTP, SNMP, RIP, and VoIP. |

A short example to understand the differences clearly :

Suppose there are two houses, H1 and H2 and a letter have to be sent from H1 to H2. But there is a river in between those two houses. Now how can we send the letter?

Solution 1: Make a bridge over the river and then it can it delivered.

Solution 2: Get it delivered through a pigeon.

Consider the first solution as TCP. A connection has to made ( bridge ) to get the data (letter) delivered.

The data is reliable because it will directly reach to another end without loss in data or error.

And the second solution is UDP. No connection is required for sending the data.

The process is fast as compare to TCP, where we need to set up connection(bridge). But the data is not reliable: we don’t know whether the pigeon will go in the right direction, or it will drop the letter on the way or some issue is encountered in mid-travel.