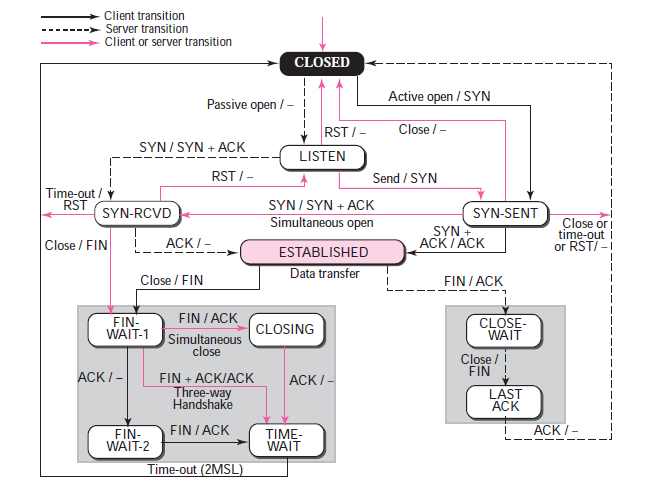
**TCP FSM**

* ***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***.



|  |  |  |
| --- | --- | --- |
| **TCP Finite State Machine (FSM) States, Events and Transitions** | | |
| **State** | **State Description** | **Event and Transition** |
| ***CLOSED*** | This is the default state that each connection starts in before the process of establishing it begins. The reason is that this state represents the situation where there is no connection between devices—it either hasn't been created yet, or has just been destroyed. If that makes sense.  | **Passive Open:** A server begins the process of connection setup by doing a passive open on a TCP port. At the same time, it sets up the data structure ([transmission control block or TCB](http://tcpipguide.com/free/t_TCPConnectionPreparationTransmissionControlBlocksT.htm)) needed to manage the connection. It then transitions to the ***LISTEN*** state. |
| **Active Open, Send *SYN*:** A client begins connection setup by sending a ***SYN*** message, and also sets up a TCB for this connection. It then transitions to the ***SYN-SENT*** state. |
| ***LISTEN*** | A device (normally a server) is waiting to receive a *synchronize* (*SYN*) message from a client. It has not yet sent its own ***SYN*** message. | **Receive Client *SYN*, Send *SYN+ACK*:** The server device receives a *SYN* from a client. It sends back a message that contains its own *SYN* and also acknowledges the one it received. The server moves to the ***SYN-RECEIVED*** state. |
| ***SYN-SENT*** | The device (normally a client) has sent a *synchronize* (*SYN*) message and is waiting for a matching *SYN* from the other device (usually a server). | **Receive *SYN*, Send *ACK*:** If the device that has sent its *SYN* message receives a *SYN* from the other device but not an *ACK* for its own *SYN*, it acknowledges the *SYN* it receives and then transitions to *SYN-RECEIVED* to wait for the acknowledgment to its *SYN*. |
| **Receive *SYN+ACK*, Send *ACK*:** If the device that sent the *SYN* receives both an acknowledgment to its *SYN* and also a *SYN* from the other device, it acknowledges the *SYN* received and then moves straight to the ***ESTABLISHED*** state. |
| ***SYN-RECEIVED*** | The device has both received a *SYN* (connection request) from its partner and sent its own *SYN*. It is now waiting for an ***ACK*** to its ***SYN*** to finish connection setup. | **Receive *ACK*:** When the device receives the *ACK* to the *SYN* it sent, it transitions to the ***ESTABLISHED*** state. |
| ***ESTABLISHED*** | The “**steady** **state**” of an open TCP connection. Data can be exchanged freely once both devices in the connection enter this state. This will continue until the connection is closed for one reason or another. | **Close, Send *FIN*:** A device can close the connection by sending a message with the *FIN* (*finish*) bit sent and transition to the ***FIN-WAIT-1***state. |
| **Receive *FIN*:** A device may receive a ***FIN*** message from its connection partner asking that the connection be closed. It will acknowledge this message and transition to the ***CLOSE-WAIT*** state. |
| ***CLOSE-WAIT*** | The device has received a close request (*FIN*) from the other device. It must now wait for the application on the local device to acknowledge this request and generate a matching request. | **Close, Send *FIN*:** The application using TCP, having been informed the other process wants to shut down, sends a close request to the TCP layer on the machine upon which it is running. TCP then sends a ***FIN*** to the remote device that already asked to terminate the connection. This device now transitions to ***LAST-ACK***. |
| ***LAST-ACK*** | A device that has already received a close request and acknowledged it, has sent its own *FIN* and is waiting for an *ACK* to this request. | **Receive *ACK* for *FIN*:** The device receives an acknowledgment for its close request. We have now sent our *FIN* and had it acknowledged, and received the other device's *FIN* and acknowledged it, so we go straight to the ***CLOSED*** state. |
| ***FIN-WAIT-1*** | A device in this state is waiting for an *ACK* for a *FIN* it has sent, or is waiting for a connection termination request from the other device. | **Receive *ACK* for *FIN*:** The device receives an **acknowledgment** for its close request. It transitions to the ***FIN-WAIT-2***state. |
| **Receive *FIN*, Send *ACK*:** The device does not receive an *ACK* for its own *FIN*, but receives a *FIN* from the other device. It acknowledges it, and moves to the ***CLOSING*** state. |
| ***FIN-WAIT-2*** | A device in this state has received an **ACK** for its request to terminate the connection and is now waiting for a matching ***FIN*** from the other device. | **Receive *FIN*, Send *ACK*:** The device receives a ***FIN*** from the other device. It acknowledges it and moves to the ***TIME-WAIT*** state. |
| ***CLOSING*** | The device has received a *FIN* from the other device and sent an *ACK* for it, but not yet received an *ACK* for its own *FIN* message. | **Receive *ACK* for *FIN*:** The device receives an acknowledgment for its close request. It transitions to the *TIME-WAIT* state. |
| ***TIME-WAIT*** | The device has now received a ***FIN*** from the other device and acknowledged it, and sent its own *FIN* and received an *ACK* for it. & it waits for 2 **MSL** & deletes **TCB**. | **Timer Expiration:** After a designated wait period, device transitions to the *CLOSED* state. |

**MSL:** Max Segment Lifetime

It is the Max time a segment can live on Internet(Usually 2 min). After a connection is “closed” it enters the state **TIME\_WAIT**, and will continue to occupy the port number for 2 MSL before it is actually removed.

**TCB:**

TCP uses a special data structure for this purpose, called a transmission control block **(TCB)**. The TCB contains all the important information about the connection, such as the **two socket numbers** that identify it and pointers to buffers where incoming and outgoing data are held. The TCB is also used to implement the sliding window mechanism. [It holds variables](http://www.tcpipguide.com/free/t_TCPSlidingWindowDataTransferandAcknowledgementMech.htm) that keep track of the **number of bytes received** and **acknowledged**, **bytes received and not yet acknowledged**, **current window size** and so forth. Of course**, each device maintains its own TCB for the connection.**

Before the process of setting up a TCP connection can begin, the devices on each end must perform some “prep work”. One of the tasks required to prepare for the connection is to set up the TCB that will be used to hold information about it. This is done right at the very start of the connection establishment process, when each device just transitions out of the CLOSED state.

**Why TIME\_WAIT??**

Firstly the **address** and **port** of each end point needs to be the same; which is normally unlikely as the client's port is usually selected for you by the operating system from the ephemeral port range and thus changes between connections. Secondly, the sequence numbers for the delayed segments need to be valid in the new connection which is also unlikely. However, should both of these things occur then **TIME\_WAIT** will prevent the new connection's data from being corrupted.

The second reason for the **TIME\_WAIT** state is to implement TCP's full-duplex connection termination reliably. Suppose that consider two endpoints endpoint1,endpoint2. Assume that endpoint2 initiates the **active close**. If the final **ACK** from end point 2 is dropped then the end point 1 will resend the final **FIN**.

**Best Practice:**

The best way to do this is to never initiate an **active close** from the server, no matter what the reason. If your peer times out, abort the connection with an **RST** rather than closing it. If your peer sends invalid data, abort the connection, etc. The idea being that if your server never initiates an active close it can never accumulate **TIME\_WAIT** sockets and therefore will never suffer from the scalability problems that they cause. As it as a Server needs to Process huge number of clients.

**Outbound connection:**

It is the Connection at client side I e here client initiates the Connection by initiating the Server IP, port.

**Inbound Connection:**

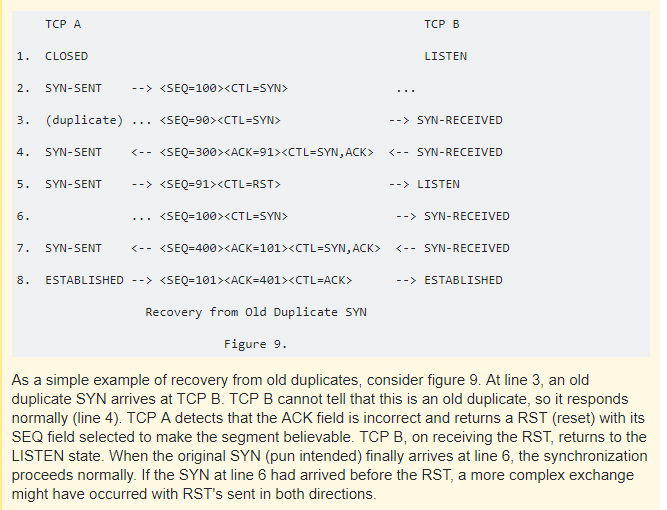
It is the Connection at Server side I e here Server Listens on particular IP, port for the clients to get connected.

**When ACK will be given by Receiver??**

Receiver generally gives ACK whenever it fills it’s Receiving Buffer with the Received data. There is no need of delivering it to the process. Process can read whenever it wants.

**Negative Scenarios:**

* Default timeout for packet is **0.6 sec** (It should not be same for all ).
* Next Retransmission time-out (RTO) = x\* current RTO ( generally X=2).
* For every packet in TCP Min retransmission count=3(default).
* TCP closes the connection if it reaches Max retransmission count(15).
* Incase of connection establishment, the device which initiates is responsible for verifying duplicate whether it is duplicate SYN’s ACK or original SYN’s ACK. Whenever initiator receives duplicate SYN’s ACK it Reset the connection by sending **RST** flag to other device.



**FIN\_WAIT2:**

many sockets which were connected to a specific remote application end up stuck in this state, it usually indicates that the remote application either always dies **unexpectedly** when in the **CLOSE\_WAIT** state or just fails to perform an active close after the passive close.

**CLOSE\_WAIT:**

If you see that connections related to a given process tend to always end up in the **CLOSE\_WAIT** state, it means that this process does not perform an active close after the passive close. When you write a program communicating over TCP, you should detect when the connection was closed by the remote host and close the socket appropriately. If you fail to do this the socket will stay in the **CLOSE\_WAIT** until the process itself disappears.