Computer Networks (CS425)

Instructor: Dr. Dheeraj Sanghi

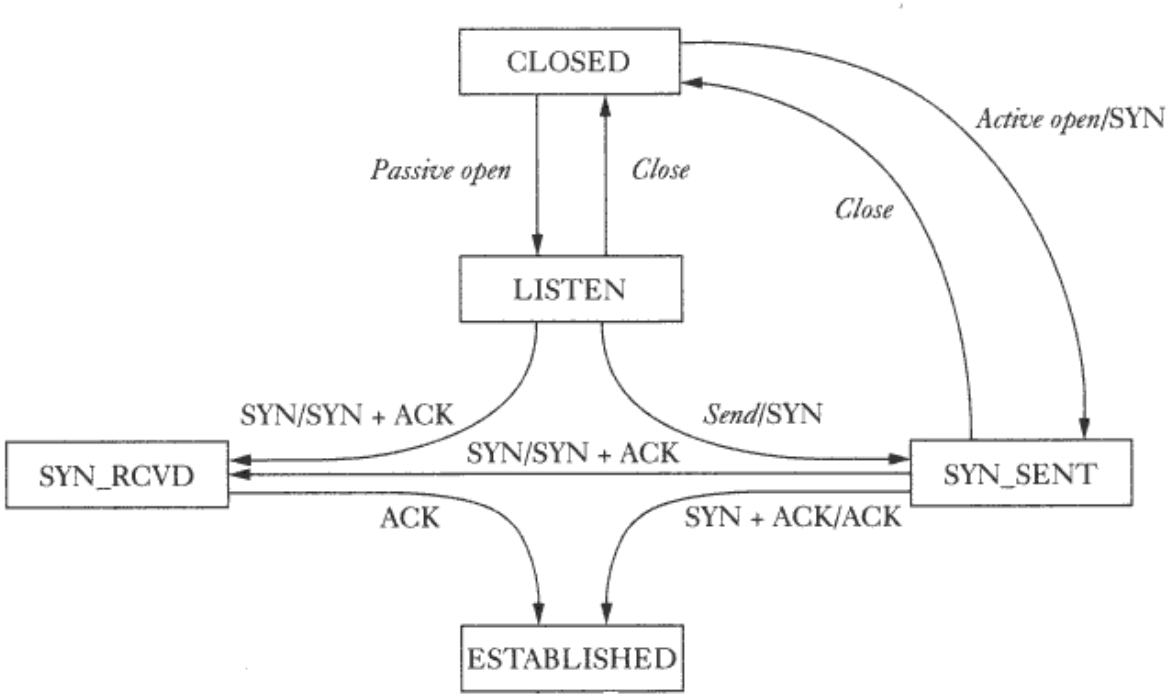
Prev| Next| Index

Transport Layer Protocol- Implementation Issues

In this class we discussed about the TCP from the implementation point of view and addressed various issues like state diagram and other details which TCP Standard does not define but supported by commercial implementations.

State Diagram

The state diagram approach to view the TCP connection establishment and closing simplifies the design of TCP implementation. The idea is to represent the TCP connection state, which progresses from one state to other as various messages are exchanged. To simplify the matter, we considered two state diagrams, viz., for TCP connection establishment and TCP connection closing.
Fig 1 shows the state diagram for the TCP connection establishment and associated table briefly explains each state.



TCP Connection establishment

The table gives brief description of each state of the above diagram.

State Description Table 1.

Listen	Represents the state when waiting for connection request from any remote host and port. This specifically applies to a Server. From this state, the server can close the service or actively open a connection by sending SYN.

Syn-Sent	Represents waiting for a matching for a connection request after having sent a connection request. This applies to both server and client side. Even though server is considered as the one with passive open, it can also send a SYN packet actively.
Syn_Rcvd	Represents waiting for a confirmation connection request acknowledgment after having both received and sent connection request.
Estab	Represents an open connection. Data transfer can take place from this point onwards.

After the connection has been established, two end-points will exchange useful information and terminate the connection. Fig. 2 shows the state diagram for terminating an active connection.

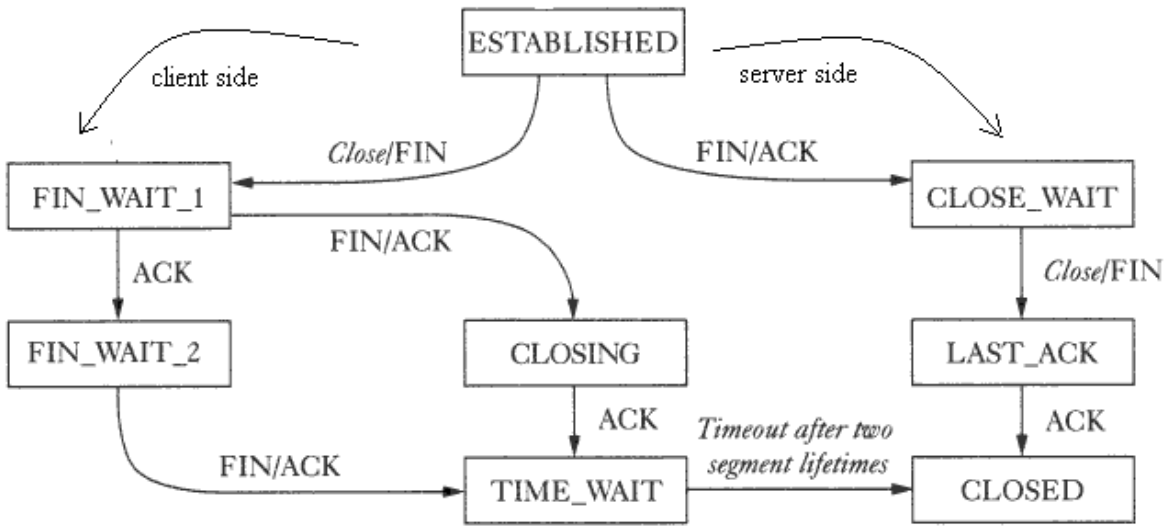


Fig 2. TCP Connection termination

State Description Table 2

FIN-WAIT-1	Represents connection termination request from the remote TCP peer, or an acknowledgment of the connection termination request previously sent. This state is entered when server issues close call.
FIN-WAIT-2	Represents waiting for a connection termination request from the remote TCP.

CLOSING	Represents connection termination request acknowledgment from the remote TCP.
TIME_WAIT	This represents waiting time enough for the packets to reach their destination. This waiting time is usually 4 min.
CLOSE_WAIT	Represents a state when the server receives a FIN from the remote TCP , sends ACK and issues close call sending FIN
LAST_ACK	Represents waiting for an ACK for the previously sent FIN-ACK to the remote TCP
CLOSE	Represents a closed TCP connection having received all the ACKs

Other implementation details

Quite Time

It might happen that a host currently in communication crashes and reboots. At startup time, all the data structures and timers will be reset to an initial value. To make sure that earlier connection packets are gracefully rejected, the local host is not allowed to make any new connection for a small period at startup. This time will be set in accordance with reboot time of the operating system.

Initial Sequence number :

Initial sequence number used in the TCP communication will be initialized at boot time randomly, rather than to 0. This is to ensure that packets from old connection should not interfere with a new connection. So the recommended method is to

- Initialize the ISN at boot time by a random number
- For every 500 ms, increment ISN by 64K
- With every SYN received, increment ISN by 64K

Maximum Request backlog at server

As we have seen in Unix Networking programming, *listen(sd,n),* sets a maximum to the number of requests to be obliged by the server at any time. So if there are already n requests for connection, and n+1 request comes, two things can be done.

- Drop the packet silently
- Ask the peer to send the request later.

The first option is recommended here because, the assumption is that this queue for request is a coincident and some time later, the server should be free to process the new request. Hence if we drop the packet, the client will go through the time-out and retransmission and server will be free to process it.

Also, Standard TCP does not define any strategy/option of knowing who requested

the connection. Only Solaris 2.2 supports this option.

Delayed Acknowledgment

TCP will piggyback the acknowledgment with its data. But if the peer does not have the any data to send at that moment, the acknowledgment should not be delayed too long. Hence a timer for 200 ms will be used. At every 200 ms, TCP will check for any acknowledgment to be sent and send them as individual packets.

Small packets

TCP implementation discourages small packets. Especially if a previous relatively large packet has been sent and no acknowledgment has been received so far, then this small packet will be stored in the buffer until the situation improves.

But there are some applications for which delayed data is worse than bad data. For example, in *telnet,* each key stroke will be processed by the server and hence no delay should be introduced. As we have seen in Unix Networking programming, options for the socket can be set as NO_DELAY, so that small packets are not discouraged.

ICMP Source Quench

We have seen in ICMP that ICMP Source Quench message will be send for the peer to slow down. Some implementations discard this message, but few set the **current window size** to 1.
But this is not a very good idea.

Retransmission Timeout

In some implementation (E.g.. Linux), RTO = RTT + 4 * delay variance is used to instead of constant 2.

Also instead of calculating RTT(est) from the scratch, cache will be used to store the history from which new values are calculated as discussed in the previous classes.

Standard values for Maximum Segment Life (MSL) will be between 0.5 to 2 minutes and Time wait state = f(MSL)

Keep Alive Time

Another important timer in TCP is keep alive timer. It is basically used by a TCP peer to check whether the other end is up or down. It periodically checks this connection. If the other end did not respond, then that connection will be closed.

Persist Timer

As we saw in TCP window management, when source sends one full window of packets, it will set its window size to 0 and expects an ACK from remote TCP to increase its window size. Suppose such an ACK has been sent and is lost. Hence source will have current window size = 0 and cannot send & destination is expecting next byte. To avoid such a deadlock, a Persist Timer will be used. When this timer goes off, the source will send the last one byte again. So we hope that situation has improved and an ACK to increase the current window size will be

received.

References

- http://www.ssfnet.org/Exchange/tcp/tcpTutorialNotes.html

---

back to top
Prev| Next | Index