International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-1 Issue-4, September 2012

Understanding Transmission Control Protocols: Basic Survey

Virendra Swarnkar, K. J. Satao

Abstract: ***Data transfer from one system to another has always been a challenging task. Multiple protocols have been developed to transmit data from one system to another, considering the security, convenience, and speed criteria. TCP (transmission control protocol) stands to be the most widely used and accepted protocol. In this paper we have discussed various commonly employed protocols for data transfer. Many algorithms and protocols have been stated which are capable in providing high speed data transfer along with security, especially in terms of congestions (little or no congestion is desirable). We have studied various transmission control protocols in this paper.***

Index Terms: TCP, HSTCP, Scalable TCP, SCTP

I. INTRODUCTION

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet Protocol Suite. TCP is one of the two original components of the suite, complementing the Internet Protocol (IP), and therefore the entire suite is commonly referred to as TCP/IP. *TCP provides reliable, ordered delivery of a stream of octets from a program on one computer to another program on another computer.* TCP is the protocol used by major Internet applications such as the World Wide Web, email, remote administration and file transfer. Other applications, which do not require reliable data stream service, may use the User Datagram Protocol (UDP), which provides a datagram service that emphasizes reduced latency over reliability. [1]

TCP is optimized for accurate delivery rather than timely delivery, and therefore, TCP sometimes incurs relatively long delays (in the order of seconds) while waiting for out-of-order messages or retransmissions of lost messages. It is not particularly suitable for real-time applications such as Voice over IP. For such applications, protocols like the Real-time Transport Protocol (RTP) running over the User Datagram Protocol (UDP) are usually recommended instead. [2]

II. TCP OPERATION

TCP protocol operations may be divided into three phases. Connections must be properly established in a multi-step handshake process (connection establishment) before entering the data transfer phase. After data transmission is completed, the connection termination closes established virtual circuits and releases all allocated resources. A TCP connection is managed by an operating system through a programming interface that represents the local end-point for communications, the Internet socket.

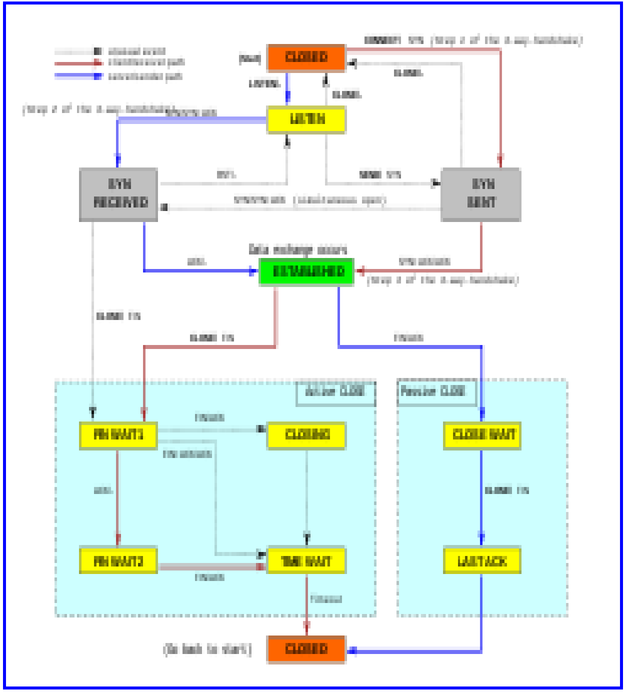

Fig 1: TCP State Diagram

During the lifetime of a TCP connection the local end-point undergoes a series of state changes:[3]

LISTEN: (server) represents waiting for a connection request from any remote TCP and port.

SYN-SENT: (client) represents waiting for a matching connection request after having sent a connection request.

SYN-RECEIVED: (server) represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request.

ESTABLISHED: (both server and client) represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.

FIN-WAIT-1: (both server and client) represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent.

FIN-WAIT-2: (both server and client) represents waiting for a connection termination request from the remote TCP.

CLOSE-WAIT: (both server and client) represents waiting for a connection termination request from the local user.

CLOSING: (both server and client) represents waiting for a connection termination request acknowledgment from the remote TCP.

LAST-ACK: (both server and client) represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request).

Revised Manuscript Received on 30 September 2012
*Correspondence Author(s)

Virendra Kumar Swarnkar*, M. Tech (SE) Scholar, CSVTU, RCET, Bhilai, India.
K. J. Satao, MCA, CSVTU, RCET, Bhilai, India.

TIME-WAIT: (either server or client) represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request. CLOSED: (both server and client) represents no connection state at all.

III. CONNECTION ESTABLISHMENT IN TCP

To establish a connection, TCP uses a three-way handshake. Before a client attempts to connect with a server, the server must first bind to and listen at a port to open it up for connections: this is called a passive open. Once the passive open is established, a client may initiate an active open. To establish a connection, the three-way (or 3-step) handshake occurs:

1. SYN: The active open is performed by the client sending a SYN to the server. The client sets the segment's sequence number to a random value A.
2. SYN-ACK: In response, the server replies with a SYN-ACK. The acknowledgment number is set to one more than the received sequence number (A + 1), and the sequence number that the server chooses for the packet is another random number, B.
3. ACK: Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value i.e. A + 1, and the acknowledgement number is set to one more than the received sequence number i.e. B + 1.

At this point, both the client and server have received an acknowledgment of the connection.

IV. CONNECTION TERMINATION IN TCP

The connection termination phase uses a four-way handshake, with each side of the connection terminating independently.

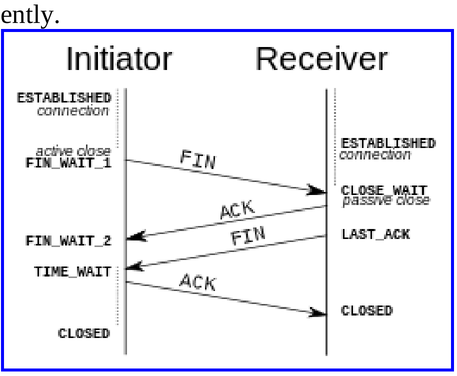

Fig 2: Connection Termination

When an endpoint wishes to stop its half of the connection, it transmits a FIN packet, which the other end acknowledges with an ACK. Therefore, a typical tear-down requires a pair of FIN and ACK segments from each TCP endpoint. After both FIN/ACK exchanges are concluded, the side which sent the first FIN before receiving one waits for a timeout before finally closing the connection, during which time the local port is unavailable for new connections; this prevents confusion due to delayed packets being delivered during subsequent connections.

A connection can be "half-open", in which case one side has terminated its end, but the other has not. The side that has terminated can no longer send any data into the connection, but the other side can. The terminating side should continue reading the data until the other side terminates as well.

It is also possible to terminate the connection by a 3-way handshake, when host A sends a FIN and host B replies with a FIN & ACK (merely combines 2 steps into one) and host A replies with an ACK.[4] This is perhaps the most common method.

It is possible for both hosts to send FINs simultaneously then both just have to ACK. This could possibly be considered a 2-way handshake since the FIN/ACK sequence is done in parallel for both directions.

Some host TCP stacks may implement a half-duplex close sequence, as Linux or HP-UX do. If such a host actively closes a connection but still has not read all the incoming data the stack already received from the link, this host sends a RST instead of a FIN (Section 4.2.2.13 in RFC 1122). This allows a TCP application to be sure the remote application has read all the data the former sent—waiting the FIN from the remote side, when it actively closes the connection. However, the remote TCP stack cannot distinguish between a Connection Aborting RST and this Data Loss RST. Both cause the remote stack to throw away all the data it received, but that the application still didn't read.

Some application protocols may violate the OSI model layers, using the TCP open/close handshaking for the application protocol open/close handshaking — these may find the RST problem on active close. As an example:

s = connect(remote);
send(s, data);
close(s);

For a usual program flow like above, a TCP/IP stack like that described above does not guarantee that all the data arrives to the other application. TCP uses a sliding window flow control protocol. In each TCP segment, the receiver specifies in the receive window field the amount of additionally received data (in bytes) that it is willing to buffer for the connection. The sending host can send only up to that amount of data before it must wait for an acknowledgment and window update from the receiving host. TCP sequence numbers and receive windows behave very much like a clock. The receive window shifts each time the receiver receives and acknowledges a new segment of data. Once it runs out of sequence numbers, the sequence number loops back to 0. When a receiver advertises a window size of 0, the sender stops sending data and starts the persist timer. The persist timer is used to protect TCP from a deadlock situation that could arise if a subsequent window size update from the receiver is lost, and the sender cannot send more data until receiving a new window size update from the receiver. When the persist timer expires, the TCP sender attempts recovery by sending a small packet so that the receiver responds by sending another acknowledgement containing the new window size.

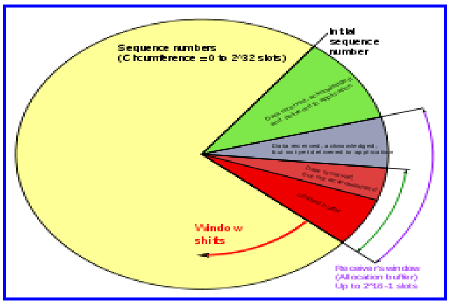

Fig 3: Shifting Of Receiving Window

If a receiver is processing incoming data in small increments, it may repeatedly advertise a small receive window. This is referred to as the silly window syndrome, since it is inefficient to send only a few bytes of data in a TCP segment, given the relatively large overhead of the TCP header. TCP senders and receivers typically employ flow control logic to specifically avoid repeatedly sending small segments. The sender-side silly window syndrome avoidance logic is referred to as Nagle's algorithm.

V. CONGESTION CONTROL

The final main aspect of TCP is congestion control. TCP uses a number of mechanisms to achieve high performance and avoid congestion collapse, where network performance can fall by several orders of magnitude. These mechanisms control the rate of data entering the network, keeping the data flow below a rate that would trigger collapse. They also yield an approximately max-min fair allocation between flows.

Acknowledgments for data sent, or lack of acknowledgments, are used by senders to infer network conditions between the TCP sender and receiver. Coupled with timers, TCP senders and receivers can alter the behavior of the flow of data. This is more generally referred to as congestion control and/or network congestion avoidance.

Modern implementations of TCP contain four intertwined algorithms: Slow-start, congestion avoidance, fast retransmit, and fast recovery (RFC 5681).

In addition, senders employ a retransmission timeout (RTO) that is based on the estimated round-trip time (or RTT) between the sender and receiver, as well as the variance in this round trip time. The behavior of this timer is specified in RFC 6298. There are subtleties in the estimation of RTT. For example, senders must be careful when calculating RTT samples for retransmitted packets; typically they use Karn's Algorithm or TCP timestamps (see RFC 1323). These individual RTT samples are then averaged over time to create a Smoothed Round Trip Time (SRTT) using Jacobson's algorithm. This SRTT value is what is finally used as the round-trip time estimate.

Enhancing TCP to reliably handle loss, minimize errors, manage congestion and go fast in very high-speed environments are ongoing areas of research and standards development. As a result, there are a number of TCP congestion avoidance algorithm variations.

VI. HIGH-SPEED TCP (HSTCP)

HSTCP is a new congestion control algorithm protocol defined in RFC 3649 for TCP. Standard TCP performs poorly in networks with a large bandwidth delay product. It is unable to fully utilize available bandwidth. HSTCP makes minor modifications to standard TCP's congestion control mechanism to overcome this limitation.

When an ACK is received (in congestion avoidance), the window is increased by $a(w)/w$ and when a loss is detected through triple duplicate acknowledgments, the window is decreased by $(1 - b(w))w$, where w is the current window size. When the congestion window is small, HSTCP behaves exactly like standard TCP so a(w) is 1 and b(w) is 0.5. When TCP's congestion window is beyond a certain threshold, a(w) and b(w) become functions of the current window size. In this region, as the congestion window increases, the value of a(w) increases and the value of b(w) decreases. This means that HSTCP's window will grow faster than standard TCP and also recover from losses more quickly. This behavior allows HSTCP to be friendly to standard TCP flows in normal networks and also to quickly utilize available bandwidth in networks with large bandwidth delay products.

HSTCP has the same slow start/timeout behavior as standard TCP.

Since only the congestion control mechanism is modified, HSTCP can be used with other TCP options like SACK. In real implementations, determining the increase and decrease parameters given a current window size is implemented as a lookup table. [5]

VII. NETWORK CONGESTION

In data networking and queuing theory, network congestion occurs when a link or node is carrying so much data that its quality of service deteriorates. Typical effects include queuing delay, packet loss or the blocking of new connections. A consequence of these latter two is that incremental increases in offered load lead either only to small increases in network throughput, or to an actual reduction in network throughput.

Network protocols which use aggressive retransmissions to compensate for packet loss tend to keep systems in a state of network congestion even after the initial load has been reduced to a level which would not normally have induced network congestion. Thus, networks using these protocols can exhibit two stable states under the same level of load. The stable state with low throughput is known as congestive collapse. [6]

Congestion avoidance can also efficiently be achieved by reducing the amount of traffic flowing into a network. When an application requests a large file, graphic or web page, it usually advertises a "window" of between 32K and 64K. This results in the server sending a full window of data (assuming the file is larger than the window). When there are many applications simultaneously requesting downloads, this data creates a congestion point at an upstream provider by flooding the queue much faster than it can be emptied. By using a device to reduce the window advertisement, the remote servers will send less data, thus reducing the congestion and allowing traffic to flow more freely. This technique can reduce congestion in a network by a factor of 40.

VIII. SCALABLE TCP

Scalable TCP is a simple change to the traditional TCP congestion control algorithm (RFC2581) which dramatically improves TCP performance in high-speed wide area networks. [7].

Scalable TCP changes the algorithm to update TCP's congestion window to the following:

cwnd := cwnd + 0.01

(for each ack received while not in loss recovery)

cwnd := 0.875 * cwnd

(on each loss event)

The idealized diagram below shows the main difference in the scaling properties of traditional and Scalable TCP. Traditional TCP probing times are proportional to the sending rate and the round trip time.

However Scalable TCP probing times are proportional only to the round trip time making the scheme scalable to high speed IP networks.[8]

To get a feel for what these properties mean in real life, suppose a TCP connection has a 1500 byte MTU and a round trip time of 200ms. The table below shows the approximate recovery times for traditional TCP congestion control and Scalable TCP congestion control at a variety of sending rates following a period of packet loss lasting less than one round trip time.

Rate	Standard TCP Recovery Time	Scalable TCP Recovery Time
1Mbps	1.7s	2.7s
10Mbps	17s	2.7s
100Mbps	2mins	2.7s
1Gbps	28mins	2.7s
10Gbps	4hrs 43mins	2.7s

Recovery Time for Standard and Scalable TCP

Whenever proposing an evolution of protocol standards it is important to consider the impact of its deployment on existing traffic. Scalable TCP has been designed from a strong theoretical base to ensure resource sharing and stability while maintaining agility to prevailing network conditions. The response curve for a traditional TCP connection and a Scalable TCP connection is shown below. For low loss rates TCP acheives has a smaller equilibrium window than Scalable TCP while for higher loss rates it has a larger equilibrium window.

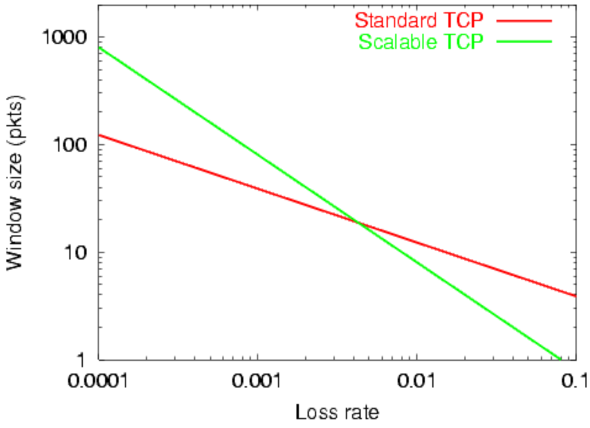

Fig 4: Response curves for Traditional and Scalable TCP

The Scalable TCP algorithm is only used for windows above a certain size. By chosing the point at which the response curves intersect good resource sharing with traditional connections can be ensured. This allows Scalable TCP to be deployed incrementally.

The control theoretic stability and variance of the scheme has been examined by Vinnicombe and Ott respectively. Furthermore the algorithm performs even better with the incremental deployment of ECN (RFC3168) at routers.

IX. STREAM CONTROL TRANSMISSION PROTOCOL

In computer networking, the Stream Control Transmission Protocol (SCTP) is a transport layer protocol, serving in a similar role to the popular protocols Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). It provides some of the same service features of both: it is message-oriented like UDP and ensures reliable, in-sequence transport of messages with congestion control like TCP.

The protocol was defined by the IETF Signaling Transport (SIGTRAN) working group in 2000,[9] and is maintained by the IETF Transport Area (TSVWG) working group. RFC 4960 defines the protocol. RFC 3286 provides an introduction.

In the absence of native SCTP support in operating systems it is possible to tunnel SCTP over UDP,[10] as well as mapping TCP API calls to SCTP ones[11].

Features

Features of SCTP include:

- Multihoming support in which one or both endpoints of a connection can consist of more than one IP address, enabling transparent fail-over between redundant network paths.
- Delivery of chunks within independent streams eliminate unnecessary head-of-line blocking, as opposed to TCP byte-stream delivery.
- Path selection and monitoring select a primary data transmission path and test the connectivity of the transmission path.
- Validation and acknowledgment mechanisms protect against flooding attacks and provide notification of duplicated or missing data chunks.
- Improved error detection suitable for Ethernet jumbo frames.

The designers of SCTP originally intended it for the transport of telephony (Signaling System 7) over Internet Protocol, with the goal of duplicating some of the reliability attributes of the SS7 signaling network in IP. This IETF effort is known as SIGTRAN. In the meantime, other uses have been proposed, for example, the Diameter protocol [12] and Reliable server pooling (RSerPool) [13].

X. CONCLUSION

This paper is made with an intension of surveying various available protocols managing data transmission between two systems. In our study we have analyzed TCP, HSTCP, Scalable TCP, SCTP. We have studied algorithm and concept of these protocols. There is scope for a new protocol that will eliminate the drawback of existing protocols.

XI. ACKNOWLEDGMENT

I express my sincere thanks to Mr. Santosh Rungta (Chairman, RCET, Bhilai), Mr. Saurabh Rungta (Director, Technical, RCET, Bhilai), Dr. Sipy Dubey (Dean – Research and Development), Prof. K. J. Satao (HOD-MCA) for their kind patronage and encouragement that made this paper possible. Working on this paper has been a great learning experience for me. There were moments of anxiety, when I could not solve a problem for several days and there were moments when I could solve a problem after struggling for several days, but I have enjoyed every moment of the process and are thankful to all people associated with us during this period.

REFERENCES

1. http://en.wikipedia.org/wiki/Transmission_Control_Protocol
2. Comer, Douglas E. (2006). Internetworking with TCP/IP:Principles, Protocols, and Architecture. 1 (5th ed.). Prentice Hall. ISBN 0-13-187671-6.
3. http://tools.ietf.org/html/rfc793
4. Tanenbaum, Andrew S. (2003-03-17). *Computer Networks* (Fourth ed.). Prentice Hall. ISBN 0-13-066102-3.
5. http://en.wikipedia.org/wiki/HSTCP
6. http://en.wikipedia.org/wiki/Network_congestion
7. http://www.deneholme.net/tom/scalable/
8. Tom Kelly, Scalable TCP: Improving Performance in Highspeed Wide Area Networks. Computer Communication Review 32(2), April 2003
9. RFC 2960 October 2000
10. http://tools.ietf.org/html/rfc2960
11. Ong, Lyndon; Randall R. Stewart; Qiaobing Xie (March 2000). Tunneling of SCTP over Single UDP Port. IETF. Retrieved 2011-07-15.
12. Bickhar, Ryan; Paul D. Amer; Randall R. Stewart (2007). "Transparent TCP-to-SCTP Translation Shim Layer" (PDF). Retrieved 2008-09-13.
13. "Transport". Diameter Base Protocol. IETF. sec. 2.1. RFC 3588. Retrieved 2012-05-18.
14. RFC 5351 Section 4.2
15. http://tools.ietf.org/html/rfc5351

AUTHOR PROFILE

Virendra Swarnkar, did his B.E from R.C.E.T., Bhilai, Dist: Durg Chhattisgarh, India. Currently he is pursuing M.Tech in Software Engineering from branch from Rungta College of Engineering & Technology, Bhilai, Chhattisgarh, India.

Prof. K. J. Satao is a Professor in Computer Science & Engineering at Rungta College of Engineering and Technology, Bhilai(C.G.). He has obtained his M.S. degree in Software Systems from BITS, Pilani (Rajasthan) in 1991. He has published more than 25 Papers in various reputed Journals, National & International Conferences. He is a Dean of the Computer Engineering & Information Technology in Chhattisgarh Swami Vivekanand Technical University, Bhilai. He is a member of the Executive Council and the Academic Council of the University.

He is a member of CSI and ISTE since 2000. He has worked in various other Engineering Colleges for about 24 Years and has over 4 Years industrial experience as well. His research & development work is equivalent to Ph.D. degree in Computer Science & Engineering. His area of research includes Operating Systems, Editors & IDEs, Information System Design & Development, etc.