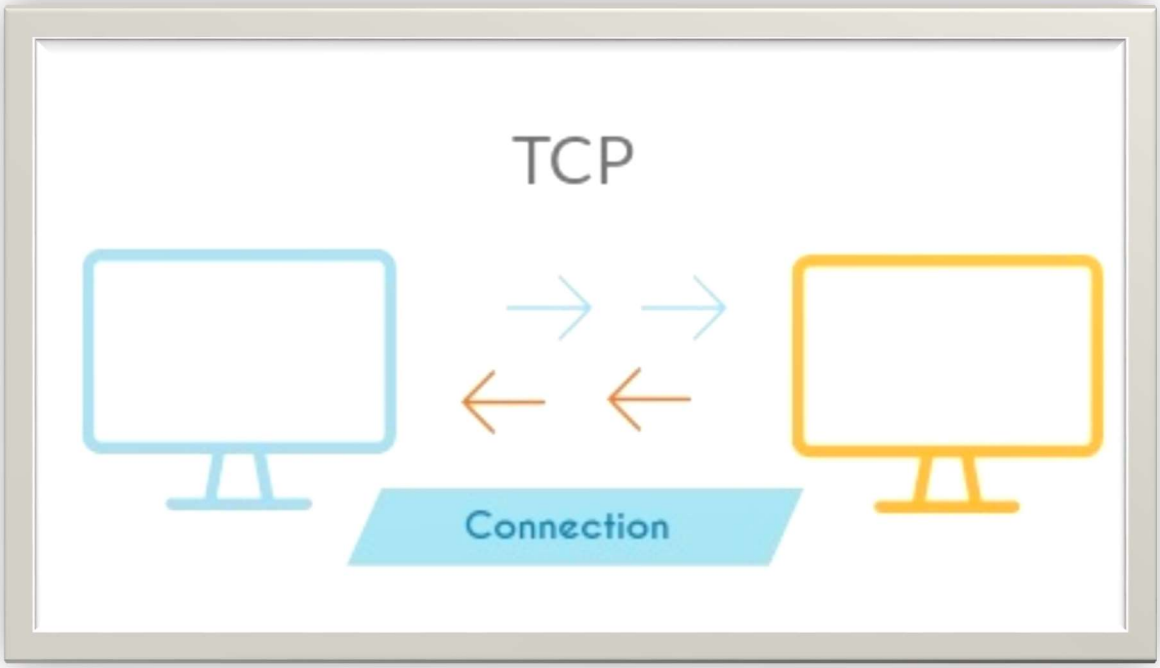
Computer Network Programming

Message borders with Own Markers for TCP Communication

Name: Mert DUMANLI

Student ID: 160315002 – NORMAL

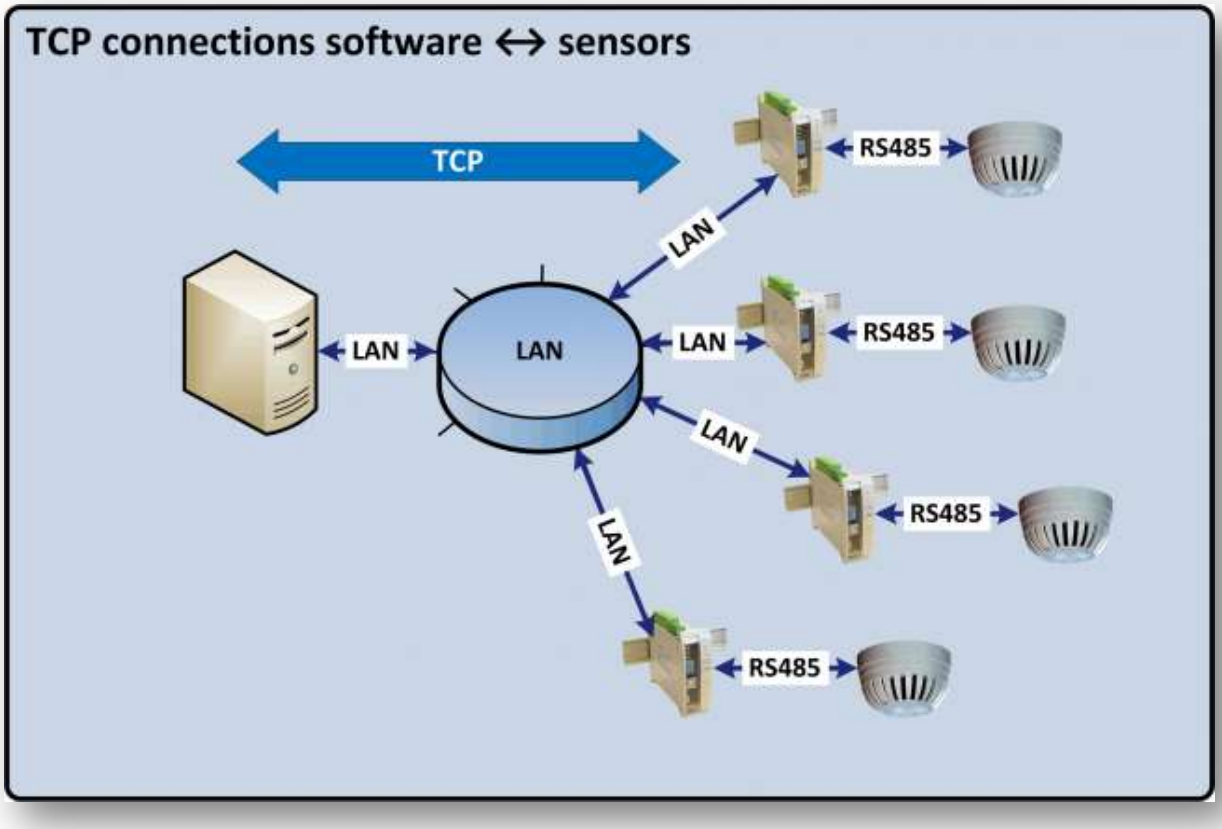


Transmission Control Protocol

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating via an IP network. Major internet applications such as the World Wide Web, email, remote administration, and file transfer rely on TCP, which is part of the Transport Layer of the TCP/IP suite. SSL/TLS often runs on top of TCP.

TCP is connection-oriented, and a connection between client and server is established (passive open) before data can be sent. Three-way handshake (active open), retransmission, and error-detection adds to reliability but lengthens latency. Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that prioritizes time over reliability. TCP employs network congestion avoidance. However, there are vulnerabilities to TCP including denial of service, connection hijacking, TCP veto, and reset attack. For network security, monitoring, and debugging, TCP traffic can be intercepted and logged with a packet sniffer.

Though TCP is a complex protocol, its basic operation has not changed significantly since its first specification. TCP is still dominantly used for the web, i.e. for the HTTP protocol, and later HTTP/2, while not used by latest standard HTTP/3.



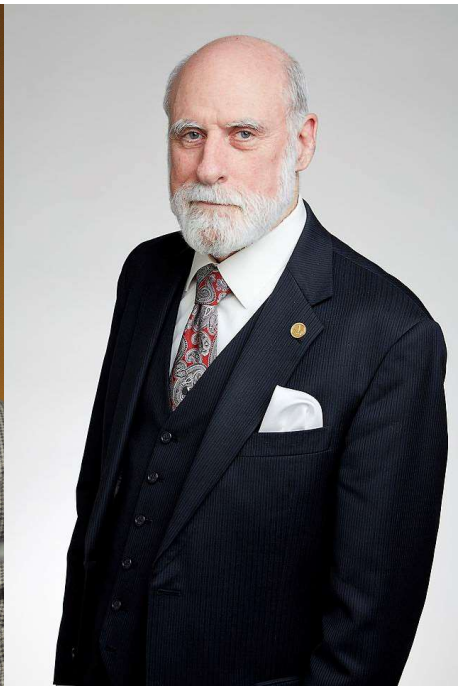
Historical Origin

In May 1974, Vint Cerf and Bob Kahn described an internetworking protocol for sharing resources using packet switching among network nodes. The authors had been working with Gérard Le Lann to incorporate concepts from the French CYCLADES project into the new network. The specification of the resulting protocol, RFC 675 (Specification of Internet Transmission Control Program), was written by Vint Cerf, Yogen Dalal, and Carl Sunshine, and published in December 1974. It contains the first attested use of the term Internet, as a shorthand for internetworking.

A central control component of this model was the Transmission Control Program that incorporated both connection-oriented links and datagram services between hosts. The monolithic Transmission Control Program was later divided into a modular architecture consisting of the Transmission Control Protocol and the Internet Protocol. This resulted in a networking model that became known informally as TCP/IP, although formally it was variously referred to as the Department of Defense (DOD) model, and ARPANET model, and eventually also as the Internet Protocol Suite.

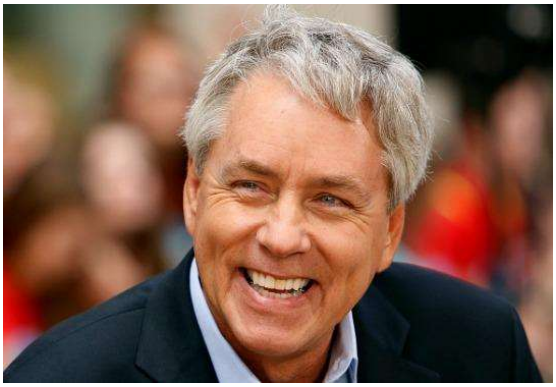
In 2004, Vint Cerf and Bob Kahn received the Turing Award for their foundational work on TCP/IP.

Bob Kahn



Vint Cerf

Gérard Le Lann *Yogen Dalal*

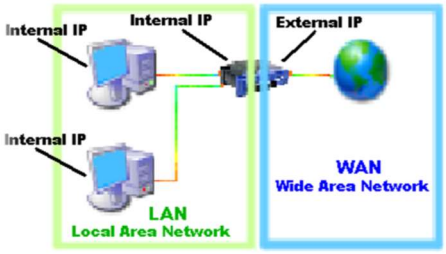


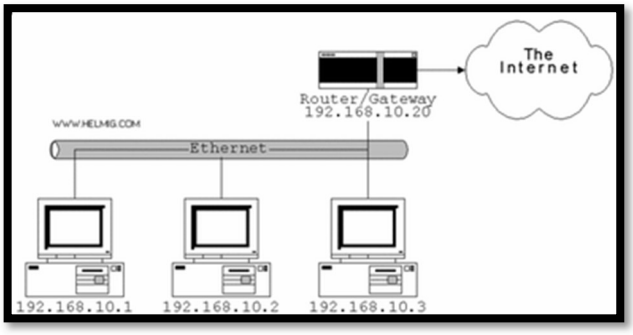
Carl sunshine

Network Function

The Transmission Control Protocol provides a communication service at an intermediate level between an application program and the Internet Protocol. It provides host-to-host connectivity at the transport layer of the Internet model. An application does not need to know the particular mechanisms for sending data via a link to another host, such as the required IP fragmentation to accommodate the maximum transmission unit of the transmission medium. At the transport layer, TCP handles all handshaking and transmission details and presents an abstraction of the network connection to the application typically through a network socket interface.

At the lower levels of the protocol stack, due to network congestion, traffic load balancing, or unpredictable network behaviour, IP packets may be lost, duplicated, or delivered out of order. TCP detects these problems, requests re-transmission of lost data, rearranges out-of-order data and even helps minimize network congestion to reduce the occurrence of the other problems. If the data still remains undelivered, the source is notified of this failure. Once the TCP receiver has reassembled the sequence of octets originally transmitted, it passes them to the receiving application. Thus, TCP abstracts the application's communication from the underlying networking details.





TCP is used extensively by many internet applications, including the World Wide Web (WWW), email, File Transfer Protocol, Secure Shell, peer-to-peer file sharing, and streaming media.

TCP is optimized for accurate delivery rather than timely delivery and can incur relatively long delays (on the order of seconds) while waiting for out-of-order messages or re-transmissions of lost messages. Therefore, 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) operating over the User Datagram Protocol (UDP) are usually recommended instead.

TCP is a reliable stream delivery service which guarantees that all bytes received will be identical and in the same order as those sent. Since packet transfer by many networks is not reliable, TCP achieves this using a technique known as positive acknowledgement with re-transmission. This requires the receiver to respond with an acknowledgement message as it receives the data. The sender keeps a record of each packet it sends and maintains a timer from when the packet was sent. The sender re-transmits a packet if the timer expires before receiving the acknowledgement. The timer is needed in case a packet gets lost or corrupted.

While IP handles actual delivery of the data, TCP keeps track of segments - the individual units of data transmission that a message is divided into for efficient routing through the network. For example, when an HTML file is sent from a web server, the TCP software layer of that server divides the file into segments and forwards them individually to the internet layer in the network stack. The internet layer software encapsulates each TCP segment into an IP packet by adding a header that includes (among other data) the destination IP address. When the client program on the destination computer receives them, the TCP software in the transport layer re-assembles the segments and ensures they are correctly ordered and error-free as it streams the file contents to the receiving application.

Protocol 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 resource that represents the local end-point for communications, the Internet socket. During the lifetime of a TCP connection, the local end-point undergoes a series of state changes:

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

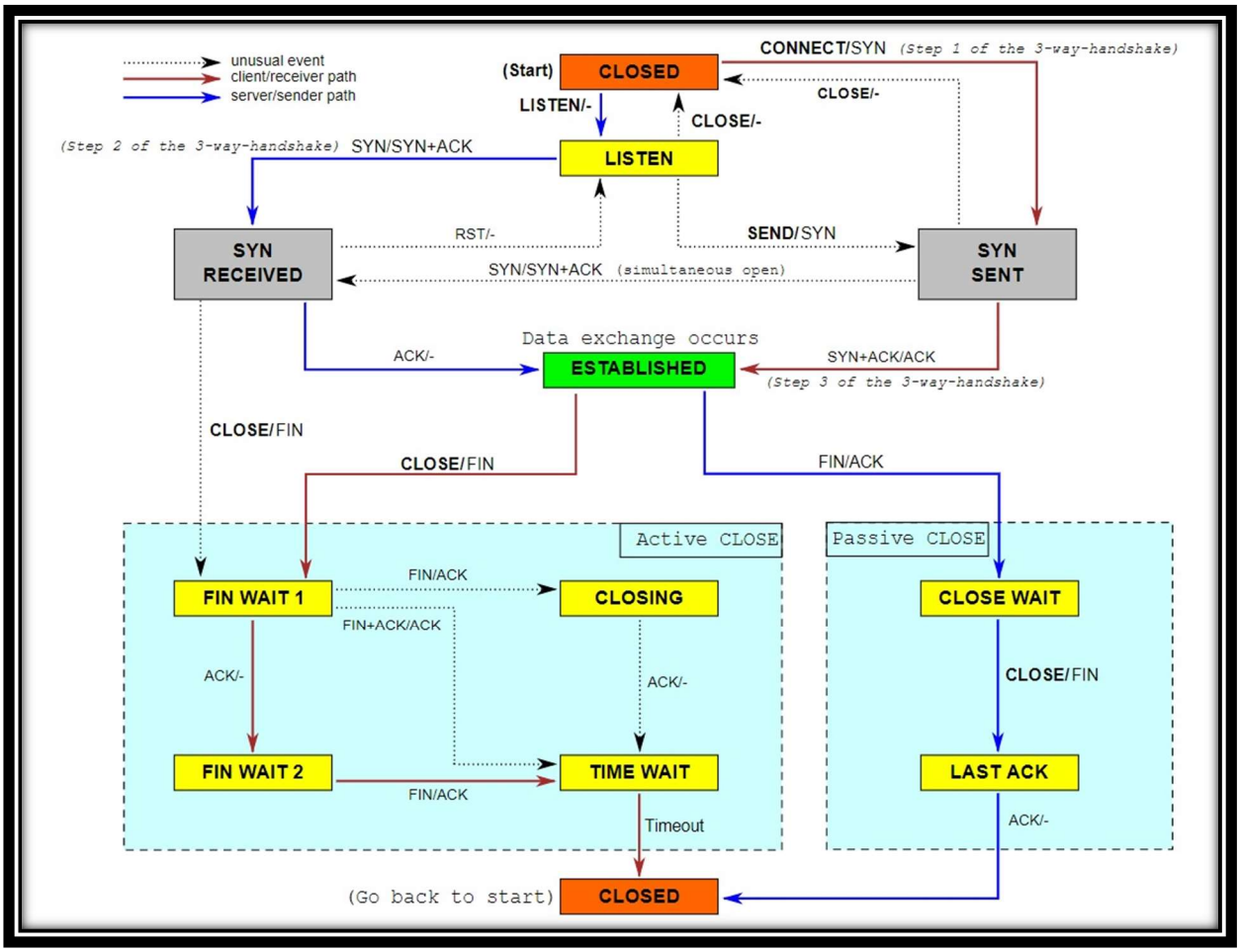
(Both server and client) represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request).

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. [According to RFC 793 a connection can stay in TIME-WAIT for a maximum of four minutes known as two maximum segment lifetime (MSL).]

CLOSED

(Both server and client) represents no connection state at all.

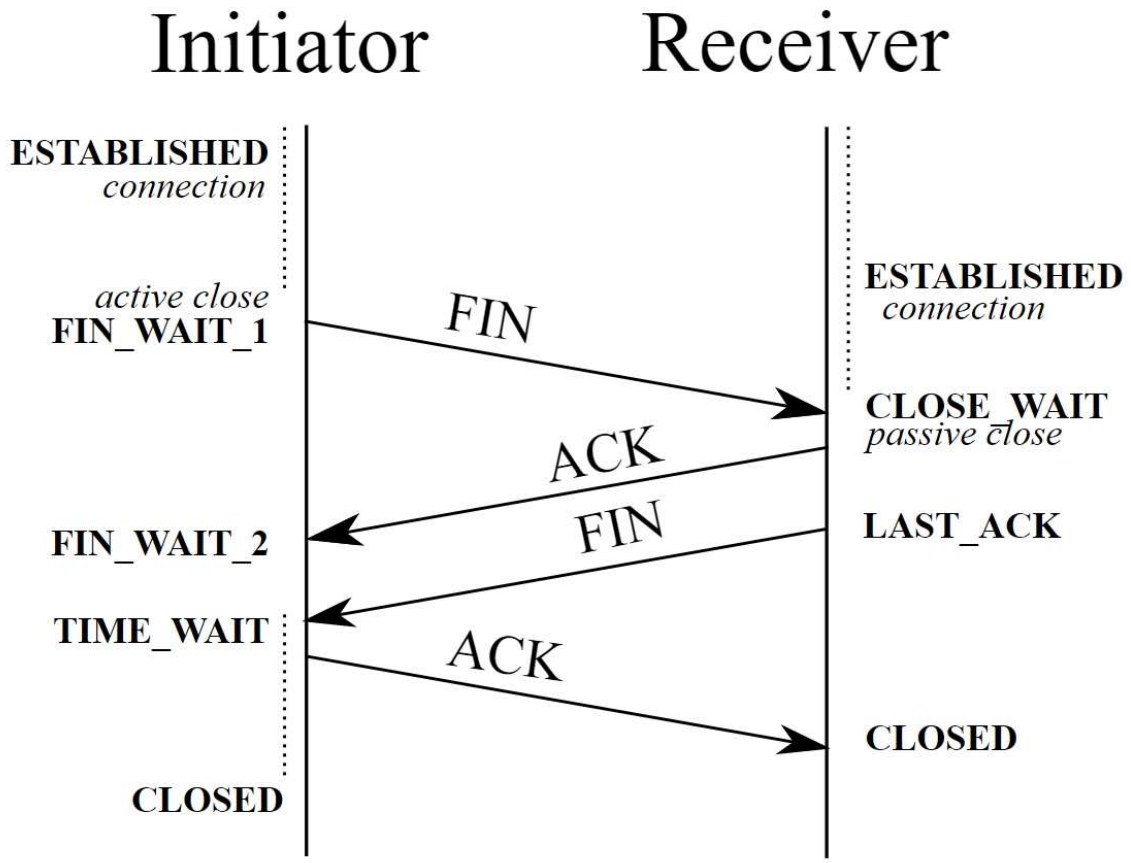


Connection establishment

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 i.e. 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. The steps 1, 2 establish the connection parameter (sequence number) for one direction and it is acknowledged. The steps 2, 3 establish the connection parameter (sequence number) for the other direction and it is acknowledged. With these, a full-duplex communication is established.



Connection termination

The connection termination phase uses a four-way handshake, with each side of the connection terminating independently. 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 the side that sent the first FIN has responded with the final ACK, it 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.

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 (losing the received data) 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. But the remote TCP stack cannot distinguish between a Connection Aborting RST and Data Loss RST. Both cause the remote stack to lose all the data received.

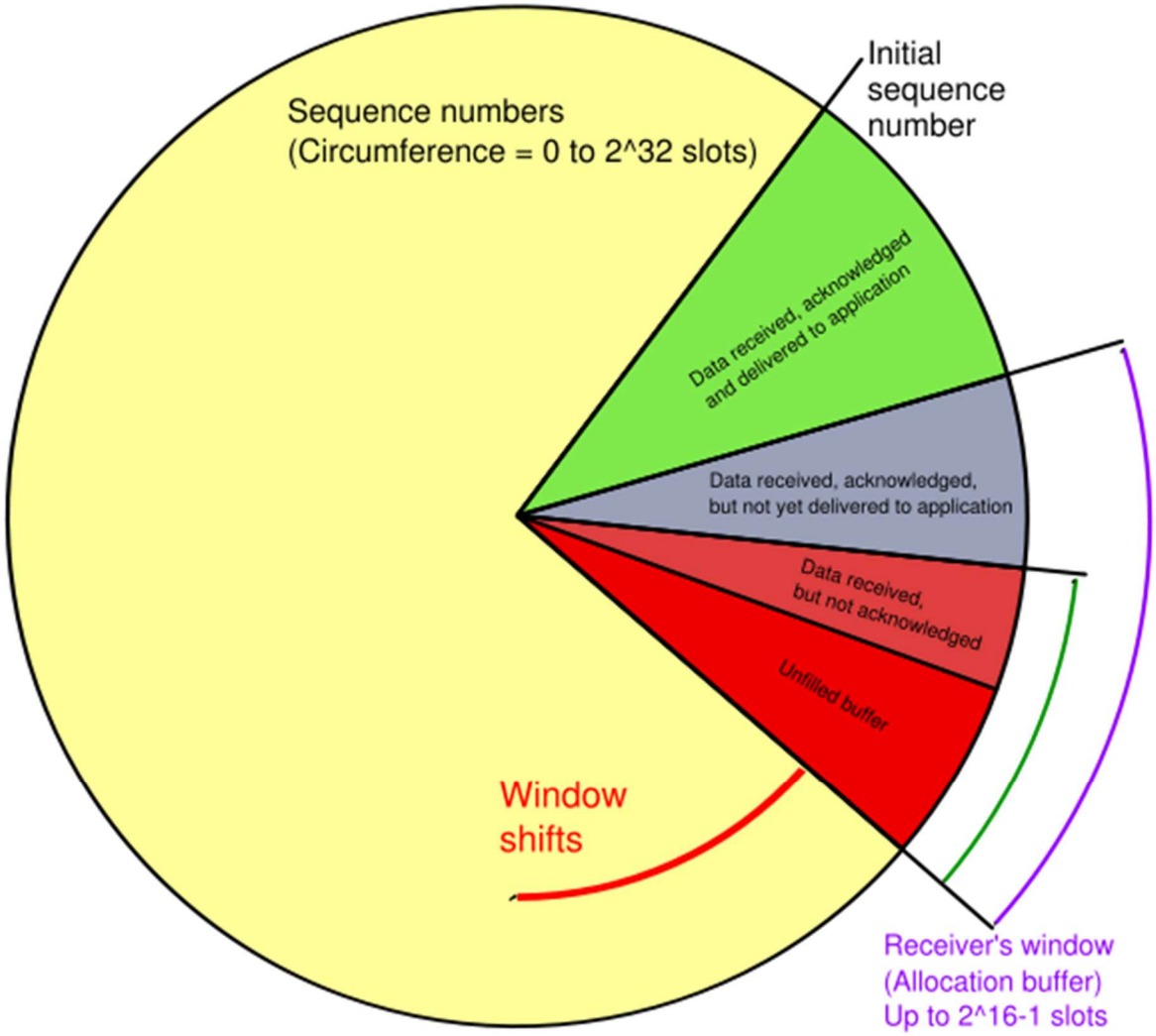
Some application protocols using the TCP open/close handshaking for the application protocol open/close handshaking may find the RST problem on active close. As an example:

s = connect(remote);

send(s, data);

close(s);

For a program flow like above, a TCP/IP stack like that described above does not guarantee that all the data arrives to the other application if unread data has arrived at this end.



C# Code's reading/rewriting and writing in Windows

I wrote with my friend the C# in visual studio in Windows operating system. Together we understood the codes, we tried to reach a solution.

C# code, which already exists as a source, the user sends a message and the server receives that message line by line. I made changes to the server code. I assigned the incoming message to the character index while reading the message continuously. I used the end of message for me as the "," character. (ASCII: 44) Wherever there is "," in the incoming message. It does it flawlessly.

References

- https://www.wikizeroo.org/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvVHJhbnNtaXNzaW9uX0NvbnRyb2xfUHJvdG9jb2w
- https://www.wikizeroo.org/index.php?q=aHR0cHM6Ly90ci53aWtpcGVkaWEub3JnL3dpa2kvVENQ
- C# Network Programming, Richard Blum
- *Interprocess Communications in Linux®: The Nooks & Crannies, By John Shapley Gray*
- *C# Network Programming by Richard Blum Sybex © 2003 (647 pages)*