NETWORK WITH TCP IP

MODULE 4

Introduction

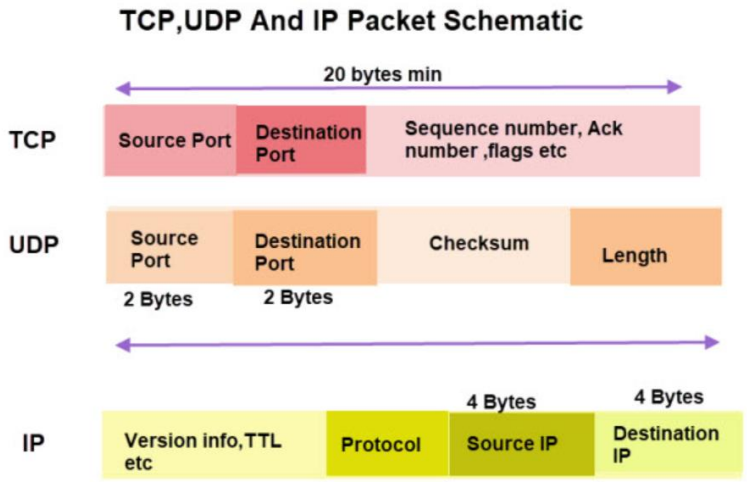
Ports and Sockets

The IP address identifies the device e.g. computer. However an IP address alone is not sufficient for running network applications, as a computer can run multiple applications and/or services. Just as the IP address identifies the computer, The network port identifies the application or service running on the computer. A port number uses 16 bits and so can therefore have a value from 0 to 65535 decimal.

Port numbers 0-1023 – Well known ports. These are allocated to server services by the Internet Assigned Numbers Authority (IANA). Ports 1024-49151- Registered Port -These can be registered for services with the IANA and should be treated as semi-reserved. User written programs should not use these ports.

Ports 49152-65535– These are used by client programs and you are free to use these in client programs. When a Web browser connects to a web server the browser will allocate itself a port in this range.

A connection between two computers uses a socket. A socket is the combination of IP address plus port Each end of the connection will have a socket.



INTRODUCTION OF UDP

TCP VS UDP

The TCP/IP protocol supports two types of port- TCP Port and UDP Port. TCP – is for connection orientated applications. It has built in error checking and will re transmit missing packets. UDP – is for connection less applications. It has no has built in error checking and will not re transmit missing packets. Applications are designed to use

either the UDP or TCP transport layer protocol depending on the type of connection they require.

Port Buffering

Instead of thinking of a process as the ultimate destination, we will imagine that each machine contains a set of abstract destination points called protocol ports. Each protocol port is identified by a positive integer. The local operating system provides an interface mechanism that processes use to specify a port or access it.

For example, if a process attempts to extract data from a port before any data arrives, the operating system temporarily stops (blocks) the process until data arrives. Once the data arrives, the operating system passes the data to the process and restarts it. In general, **ports are buffered**, so data that arrives before a process is ready to accept it will not be lost. To achieve buffering, the protocol software located inside the operating system places packets that arrive for a particular protocol port in a (finite) queue until a process extracts them.

The User Datagram Protocol

UDP uses the underlying Internet Protocol to transport a message from one machine to another, and provides the same unreliable, connectionless datagram delivery semantics as IP. It does not use acknowledgements to make sure messages arrive, it does not order incoming messages, and it does not provide feedback to control the rate at which information flows between the machines.

Thus, UDP messages can be lost, duplicated, or arrive out of order. Furthermore, packets can arrive faster than the recipient can process them

Definition

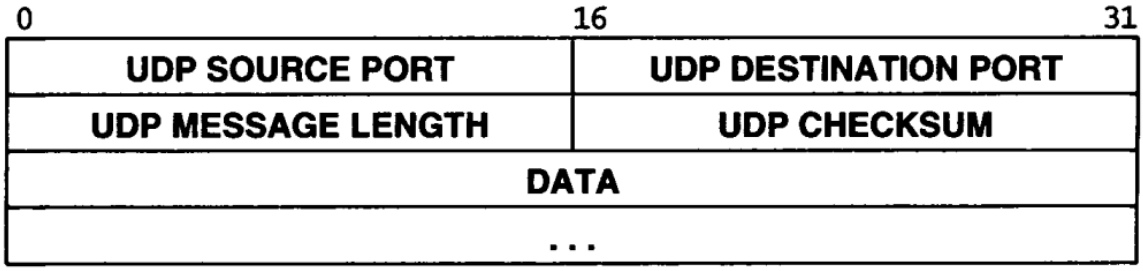
The User Datagram Protocol (UDP) provides an unreliable connectionless delivery service using IP to transport messages between machines. It uses IP to carry messages, but adds the ability to distin guish among multiple destinations within a given host computer.

An application program that uses UDP accepts full responsibility for handling the problem of reliability, including message loss, duplication, delay, out-of-order delivery, and loss of connectivity.

Format Of UDP Messages

Each UDP message is called a user datagram. Conceptually, a user datagram consists of two parts: a UDP header and a UDP data area.

As Figures hows, the header is divided into four 16-bit fields the port from which the message was sent, the port to which the message is destined, the message length, and a UDP checksum.



The SOURCE PORT and DESTINATION PORT fields contain the 16-bit UDP protocol port numbers. The SOURCE PORT is optional. When used, it specifies the port to which replies should be sent; if not used, it should be zero.

The LENGTH field contains a count of octets in the UDP datagram, including the UDP header and the user data. Thus, the minimum value for LENGTH is eight, the length of the header alone.

The UDP checksum is optional and need not be used at all; a value of zero in the CHECKSUM field means that the checksum has not been computed.

UDP Encapsulation And Protocol Layering

UDP provides our first example of a transport protocol. UDP lies in the layer above the Internet Protocol layer. Layering UDP above IP means that a complete UDP message, including the UDP header and data, is encapsulated in an IP datagram as it travels across an internet

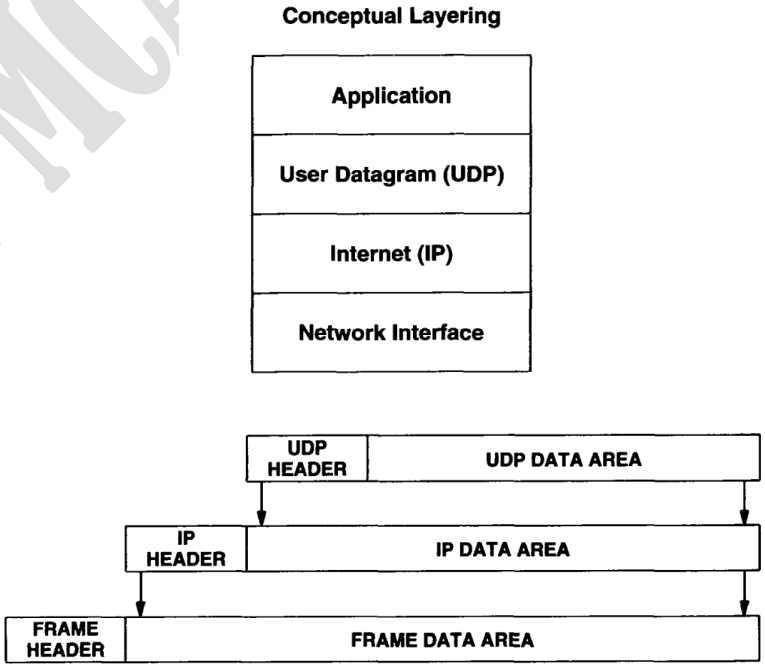


Figure 12.4 A UDP datagram encapsulated in an IP datagram for transmission across an internet. The datagram is further encapsulated in a frame each time it travels across a single network.

For the protocols, encapsulation means that UDP prepends a header to the data that a user sends and passes it to IP. The IP layer prepends a header to what it receives from UDP. Finally, the network interface layer embeds the datagram in a frame before sending it from one machine to another.

On input, a packet arrives at the lowest layer of network software and begins its ascent through successively higher layers. Each layer removes one header before passing the message on, so that by the time the highest level passes data to the receiving process, all headers have been removed.

The IP layer is responsible only for transferring data between a pair of hosts on an internet, while the UDP layer is responsible only for differentiating among multiple sources or destinations within one host.

Reliable Stream Transport Service (TCP)

The Need For Stream Deliver

At the lowest level, computer communication networks provide unreliable packet delivery. Packets can be lost or destroyed when transmission errors interfere with data, when network hardware fails, or when networks become too heavily loaded to accommodate the load presented. Networks that route packets dynamically can deliver them out of order, deliver them after a substantial delay, or deliver duplicates.

As a consequence, one goal of network protocol research has been to find general purpose solutions to the problems of providing reliable stream delivery, making it possible for experts to build a single instance of stream protocol software that all application programs use.

Properties Of The Reliable Delivery Service / TCP

- **Stream Orientation**. When two application programs (user processes) transfer large volumes of data, we think of the data as a stream of bits, divided into 8-bit octets, which are informally called bytes.

  The stream delivery service on the destination machine passes to the receiver exactly the same sequence of octets that the sender passes to it on the source machine.

- **Virtual Circuit Connection**. Making a stream transfer is analogous to placing a telephone call. Before transfer can start, both the sending and receiving application programs interact with their respective operating systems, informing them of the desire for a stream transfer.

- **Buffered Transfer.** Application programs send a data stream across the virtual circuit by repeatedly passing data octets to the protocol software.

  When transferring data, each application uses whatever size pieces it finds convenient, which can be as small as a single octet.

At the receiving end, the protocol software delivers octets from the data stream in exactly the same order they were sent, making them available to the receiving application program

- **Unstructured Stream**. It is important to understand that the TCPIIP stream service does not honor structured data streams.
  For example, there is no way for a payroll application to have the stream service mark boundaries between employee records, or to identify the contents of the stream as being payroll data.

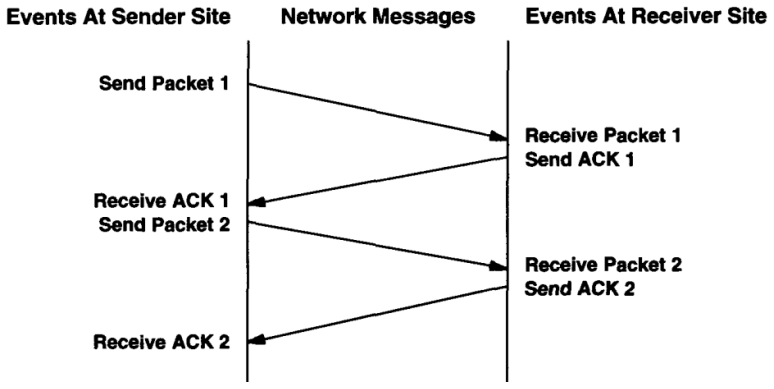
  Application programs using the stream service must understand stream content and agree on stream format before they initiate a connection.

- **Full Duplex Connection.**
  Connections provided by the TCPIP stream service allow concurrent transfer in both directions. Such connections are called full duplex. From the point of view of an application process, a full duplex connection consists of two independent streams flowing in opposite directions, with no apparent interaction.

Providing Reliability

Reliable stream delivery service guarantees to deliver a stream of data sent from one machine to another without duplication or data loss. Most reliable protocols use a single fundamental technique known as positive acknowledgement with retransmission.

The technique requires a recipient to communicate with the source, sending back an acknowledgement (ACK) message as it receives The sender keeps a record of each packet it sends and waits for an acknowledgement before sending the next packet.



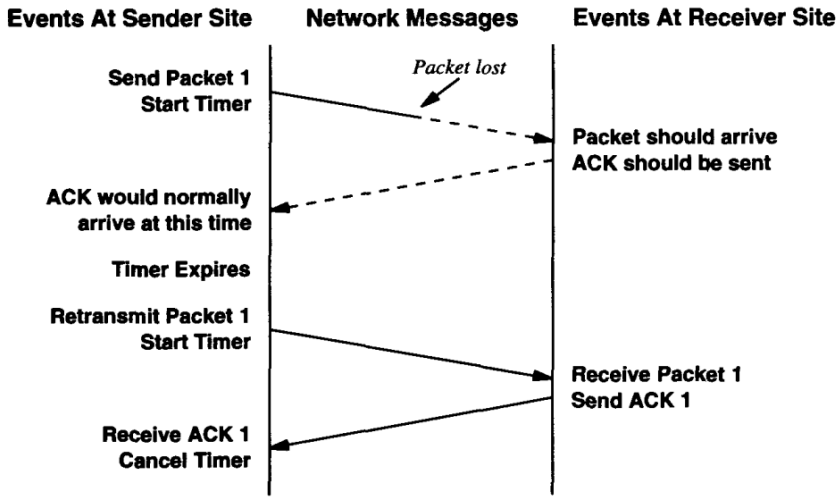


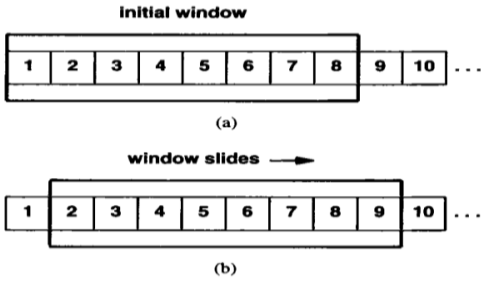
Figure 13.2 Timeout and retransmission that occurs when a packet is lost. The dotted lines show the time that would be taken by the transmission of a packet and its acknowledgement, if the packet was not lost

The sender also starts a timer when it sends a packet and retransmits a packet if the timer expires before an acknowledgement arrives.

The Idea Behind Sliding Windows

To achieve reliability, the sender transmits a packet and then waits for an acknowledgement before transmitting another. Data only flows between the machines in one direction at any time, even if the network is capable of simultaneous communication in both directions

A simple positive acknowledgement protocol wastes a substantial amount of network bandwidth because it must &lay sending a new packet until it receives an acknowledgement for the previous packet.



'igure 13.3 (a) A sliding window protocol with eight packets in the window, and (b) The window sliding so that packet *9* can be sent when an acknowledgement has been received for packet *1*. Only unacknowledged packets are retransmitted.

Sliding window protocols use network bandwidth better because they allow the sender to transmit multiple packets before waiting for an acknowledgement. The easiest

way to envision sliding window operation is to think of a sequence of packets to be transmitted as above Figure

We say that a packet is unacknowledged if it has been transmitted but no acknowledgement has been received. Technically, the number of packets that can be unacknowledged at any given time is constrained by the window size and is limited to a small, fixed number.

Because a well tuned sliding window protocol keeps the network completely saturated with packets, it obtains substantially higher throughput than a simple positive acknowledgement protocol.

The Transmission Control Protocol

TCP is a communication protocol, not a piece of software The protocol specifies the format of the data and acknowledgements that two computers exchange to achieve a reliable transfer, as well as the procedures the computers use to ensure that the data arrives correctly.

It specifies how TCP software distinguishes among multiple destinations on a given machine, and how communicating machines recover from errors like lost or duplicated packets. The protocol also specifies how two computers initiate a TCP stream transfer and how they agree when it is complete.

Segments, Streams, And Sequence Numbers

TCP views the data stream as a sequence of octets or bytes that it divides into segments for transmission. Usually, each segment travels across an internet in a single IP datagram. **TCP uses a specialized sliding window mechanism to solve two important problems: efficient transmission and flow control.**

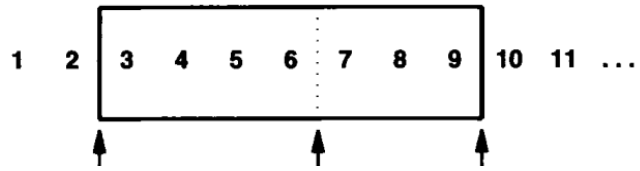
Like the sliding window protocol described earlier, the TCP window mechanism makes it possible to send multiple segments before an acknowledgement arrives. Doing so increases total throughput because it keeps the network busy.

The TCP form of a sliding window protocol also solves the end-to-end flow control problem, by allowing the receiver to restrict transmission until it has sufficient buffer space to accommodate more data

Operation of sling window

The TCP sliding window mechanism operates at the octet level, not at the segment or packet level. Octets of the data stream are numbered sequentially, and a sender keeps three pointers associated with every connection.

The pointers define a sliding window as Figure illustrates. The first pointer marks the left of the sliding window, separating octets that have been sent and acknowledged from octets yet to be acknowledged.



A second pointer marks the right of the sliding window and defines the highest octet in the sequence that can be sent before more acknowledgements are received.

The third pointer marks the boundary inside the window that separates those octets that have already been sent from those octets that have not been sent.

Variable Window Size And Flow Control

The advantage of using a variable size window is that it provides flow control as well as reliable transfer.

To avoid receiving more data than it can store, the receiver sends smaller window advertisements as its buffer fills. In the extreme case, the receiver advertises a window size of zero to stop all transmissions. Later, when buffer space becomes available, the receiver advertises a nonzero window size to trigger the flow of data again.

When intermediate machines become overloaded, the condition is called congestion, and mechanisms to solve the problem are called congestion control mechanisms. TCP uses its sliding window scheme to solve the end-to-end flow control problem

TCP Segment Format

The unit of transfer between the TCP software on two machines is called a segment. Segments are exchanged to establish connections, transfer data, send acknowledgements, advertise window sizes, and close connections.

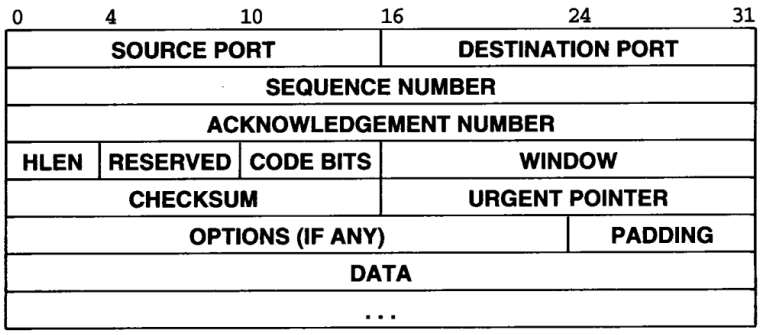


Figure 13.7 The format of a TCP segment with a TCP header followed by data. Segments are used to establish connections as well as to carry data and acknowledgements.

Fields SOURCE PORT and DESTINATION PORT contain the TCP port numbers that identify the application programs at the ends of the connection. The SEQUENCE NUMBER field identifies the position in the sender's byte stream of the data in the segment.

The ACKNOWLEDGEMENT NUMBER field identifies the number of the octet that the source expects to receive next

The HLENS field contains an integer that specifies the length of the segment header measured in 32-bit multiples. It is needed because the OPTIONS field varies in length, depending on which options have been included. Thus, the size of the TCP header varies depending on the options selected. The 6-bit field marked RESERVED is reserved for future use.

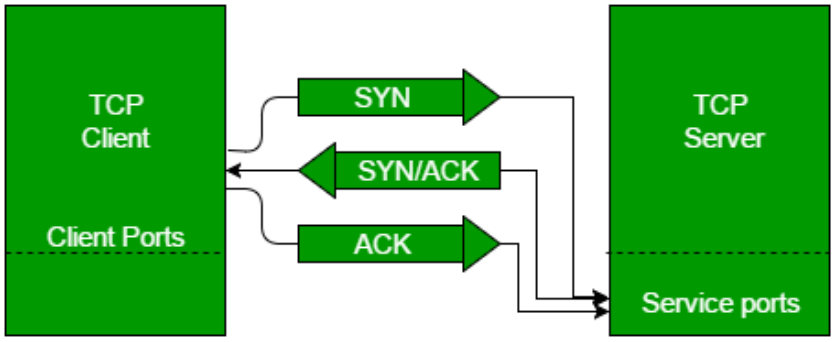
TCP software uses the 6-bit field labelled CODE BITS to determine the purpose and contents of the segment. TCP software advertises how much data it is willing to accept every time it sends a segment by specifying its buffer size in the WINDOW field.

Establishing A TCP Connection

Three way handshake process

TCP provides reliable communication with something called Positive Acknowledgement with Re-transmission(PAR). The Protocol Data Unit(PDU) of the transport layer is called a segment.

If the data unit received at the receiver's end is damaged, the receiver discards the segment. So the sender has to resend the data unit for which positive acknowledgement is not received.



You can realize from the above mechanism that three segments are exchanged between sender(client) and receiver(server) for a reliable TCP connection to get established.

Establishing A TCP Connection

- **Step 1 (SYN):** In the first step, the client wants to establish a connection with a server, so it sends a segment with SYN(Synchronize Sequence Number)
- **Step 2 (SYN + ACK):** Server responds to the client request with SYN-ACK signal bits set.
- **Step 3 (ACK):** In the final part client acknowledges the response of the server and they both establish a reliable connection with which they will start the actual data transfer

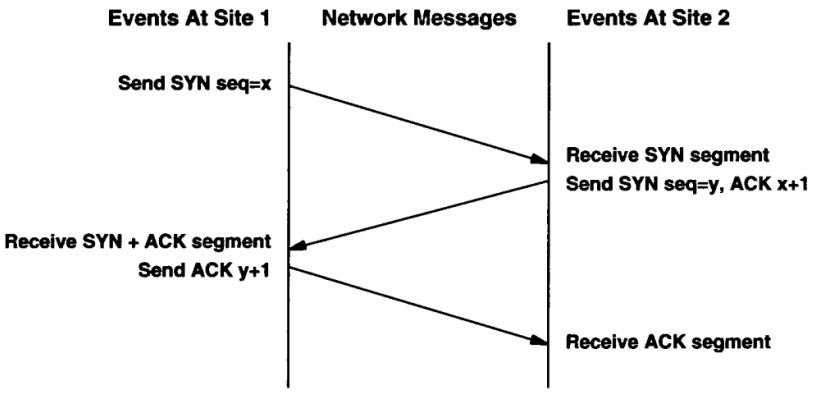


Figure 13.13 The sequence of messages in a three-way handshake. Time proceeds down the page; diagonal lines represent segments sent between sites. SYN segments carry initial sequence number information.

Usually, the TCP software on one machine waits passively for the handshake, and the TCP software on another machine initiates it. However, the handshake is carefully designed to work even if both machines attempt to initiate a connection simultaneously. Thus, a connection can be established from either end or from both ends simultaneously

Initial Sequence Numbers

The three-way handshake accomplishes two important functions. It guarantees that both sides are ready to transfer data (and that they know they are both ready), and it allows both sides to agree on initial sequence numbers.

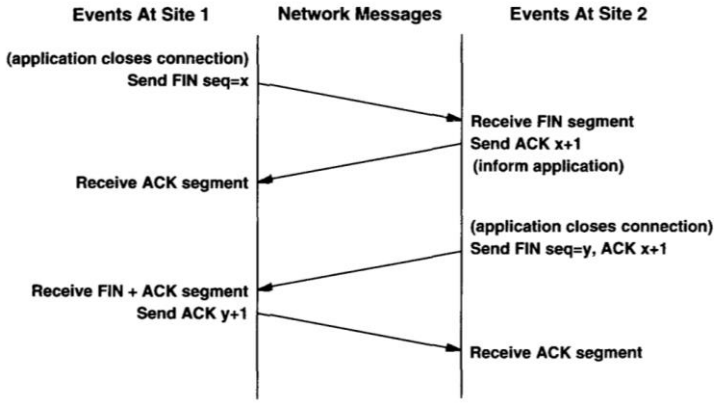
Sequence numbers are sent and acknowledged during the handshake. Each machine must choose an initial sequence number at random that it will use to identify bytes in the stream it is sending.

Closing a TCP Connection

Two programs that use TCP to communicate can terminate the **conversation gracefully** using the close operation. Internally, TCP uses a modified three-way handshake to close connections.

When an application program tells TCP that it has no more data to send, TCP will close the connection in one direction. To close its half of a connection, the sending TCP finishes transmitting the remaining data, waits for the receiver to acknowledge it, and then sends a segment with the FIN bit set. The receiving TCP acknowledges the FIN segment and informs the application program on its end that no more data is available

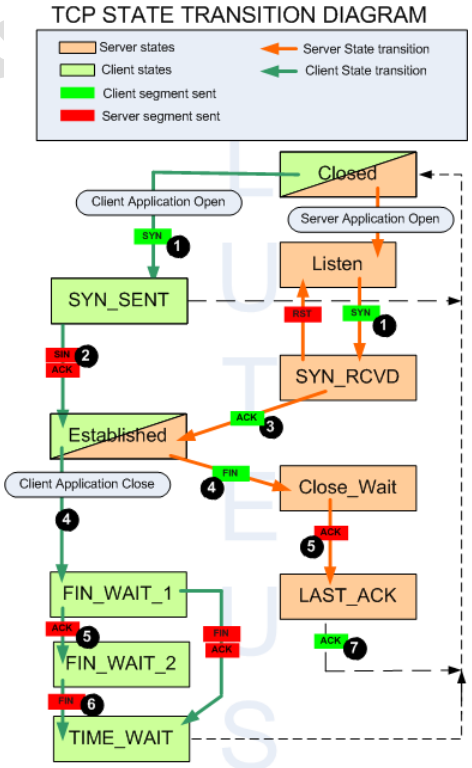
illustrates the procedure.



Of course, acknowledgements continue to flow back to the sender even after a connection has been closed. When both directions have been closed, the TCP software at each endpoint deletes its record of the connection.

TCP State Machine

Like most protocols, the operation of TCP can best be explained with a theoretical model called a finite state machine.



The TCP software at each endpoint begins in the CLOSED state. Application programs must issue either a passive open command (to wait for a connection from another machine), or an active open command (to initiate a connection).

An active open command forces a transition from the CLOSED state to the SYN SENT state. When TCP follows the transition, it emits a SYN segment. When the other end returns a segment that contains a SYN plus ACK, TCP moves to the ES TABLISHED state and begins data transfer

The TIMED WAIT state reveals how TCP handles some of the problems incurred with unreliable delivery. TCP keeps a notion of maximum segment lifetime (MSL), the maximum time an old segment can remain alive in an internet. To avoid having segments from a previous connection interfere with a current one, TCP moves to the TIMED WAIT state after closing a connection.

It remains in that state for twice the maximum segment lifetime before deleting its record of the connection. If any duplicate segments happen to arrive for the connection during the timeout interval, TCP will reject them.

However, to handle cases where the last acknowledgement was lost, TCP acknowledges valid segments and restarts the timer. Because the timer allows TCP to distinguish old connections from new ones, it prevents TCP from responding with a RST (reset) if the other end retransmits a FIN request.

Silly Window Syndrome

Silly Window Syndrome is a problem that arises due to poor implementation of TCP. It degrades the TCP performance and makes the data transmission extremely inefficient. The problem is called so because: ***It causes the sender window size to shrink to a silly value.***

The window size shrinks to such an extent where the data being transmitted is smaller than TCP Header.

The two major causes of this syndrome are as follows:

- Sender window transmitting one byte of data repeatedly.
- Receiver window accepting one byte of data repeatedly.

Cause-1: Sender window transmitting one byte of data repeatedly –

Suppose only one byte of data is generated by an application . The poor implementation of TCP leads to transmit this small segment of data. Every time the application generates

a byte of data, the window transmits it. This makes the transmission process slow and inefficient.

Cause-2: Receiver window accepting one byte of data repeatedly –

Suppose consider the case when the receiver is unable to process all the incoming data. In such a case, the receiver will advertise a small window size. The process continues and the window size becomes smaller and smaller. A stage arrives when it repeatedly advertises window size of 1 byte.

Cause 1 : avoidance

Nagle's algorithm suggests:

1. Sender should send only the first byte on receiving one byte data from the application.
2. Sender should buffer all the rest bytes until the outstanding byte gets acknowledged.
3. In other words, sender should wait for 1 RTT(Round Trip Time).

After receiving the acknowledgement, sender should send the buffered data in one TCP segment. Then, sender should buffer the data again until the previously sent data gets acknowledged.

Cause 2: Avoidance

Clark's solution suggests:

1. Receiver should not send a window update for 1 byte.
2. Receiver should wait until it has a descent amount of space available.
3. Receiver should then advertise that window size to the sender.