Socket : 1971

Network Programming

Introduction and TCP/IP:

→ Protocol: an agreement on how programs will communicate.

→ client / server.

→ Deciding that the client always initiates requests tends to simplify the protocol as well as the program themselves.

→ Some complex network applications require asynchronous callback communication, where a server initiates a message to the client.

→ clients normally communicate with one server at a time

→ From server's perspective, servers handles multiple clients at a time.

→ There are several different ways of server handling multiple clients.
- Multithreading.
- Socket programming.

→ The client application & server application may be thought of as communicating via a network protocol but actually multiple layers of network protocols are typically involved.

Network protocol: TCP/IP protocol suite (also called as Internet Protocol suite)

Ex: Web clients and servers communicate using the Transmission Control Protocol, or TCP.

TCP in turn, uses the Internet Protocol or IP and communicates with a datalink layer of some form.

(Figure 1.3)

→ Even though the client and server communicate using an application protocol, the transport layer communicate using TCP.

→ Actual flow of information between the client and server goes down the protocol stack on one side, across the network, and up the protocol stack on the other side.

→ Also note that client and server are typically user processes, while TCP and IP are normally part of the protocol stack within the kernel.

→ TCP and IP are not the only protocols that we will discuss. Some clients and servers use the User Datagram Protocol (UDP) instead of TCP.

→ IP (IP V4 used since early 1980s)
IPV4 will be replaced by IP V6 (developed during mid 1990s) by 2040

IP V4	IP V6
1. 32-bit address	128-bit address

(Explore few more differences.

→ All the time client and server are not connected to the same LAN. (fig (1.4))

→ Client and server on different LANs, with both LANs connected to Wide Area Network (WAN) using routers.

→ Routers are the building blocks of WANs.

Ex: The largest WAN is the internet

→ Many companies build their own WANs and these private WANs may or may not be connected to the internet.

→ To simplify the code, the wrapper functions are used.

→ These wrapper functions are used most of the time to check for an error, print an appropriate message, & terminate when an error occurs.

1.2 A Simple Daytime client: *

Socket API programming in Linux:

Note:

Client - Specific

— connect is an "active open" where a client knows about a server, and requests service

Server - Specific:

server needs to reserve a specific (ip, port) pair for receiving connections

Also known as "passive open"

(bind, listen, accept)

Two - way communication

— After the client connects to the server, there is nothing "special" about client vs server besides how the application layer protocol works.

— Both sides can send, recv and close whenever they want

(But they probably want to follow a well defined protocol if they want anything to work)

getaddrinfo:

Leverages DNS

- input: a dns hostname
- output: a list of potential IPs to connect to / listen on

socket:

creates a file descriptor, just like open

- But all it does is create it.
- doesn't even use the output of getaddrinfo!

Analogy: Building a door frame that is not yet connected to anything

connect:

- Given the file descriptor & an IP to connect to, create a connection
- uses TCP under the hood
- Abstracted away - as application programmers we don't need to know how it works, just that it does!
- Analogy: Install the door in a house. To use it, you don't need to know carpentry, just how to use keys & doorknobs

A simple Daytime client.

1. Client establishes a TCP connection with a server & the server sends back the current time & date in a human readable format

1. Include header files
2. Main function along with command line arguments.

   argc : argument count
   argv : argument values

argc : count of total command line arguments passed to executable on execution.

argv : array of character string of each command line argument passed to executable on execution.

3. Create TCP socket:

   sockfd = socket (AF_INET, SOCK_STREAM, 0);

→ returns a small integer descriptor that we can use to identify the socket in all future function calls.

→ The API that we are using is called the sockets API.

4 Specify server's IP address and port.

→ fill in an Internet socket address structure (sockaddr_in saddr) with the server's IP address & port number.

→ Socket address family to AF_INET (SH)

Set port no = 13 (day time server port on TCP/IP host)

→ set IP address to the value specified as the first command line argument (argv[1])

→ The IP address & the port number fields in the structure must be in specific formats.

ntohs: ("network to host short") used to convert a short (2-byte) integer from the standard network byte order to local host byte order.

htons: ("host to network short") to convert the binary port number

(5) Establish connection with server.

The connect() function, when applied to a TCP socket, establishes a TCP connection with the server specified by the socket address structure pointed to by the second argument.

We must also specify the length of the socket address structure as the third argument to connect, & for the Internet socket address structures, we always let the compiler calculate the length using C's sizeof operator.

(6) Read and display server's reply.

→ We read the server's reply & display the result using the standard I/O function.

→ TCP is a byte-stream protocol with no record boundaries.

→ The server's reply is normally a 26-byte string of the form

(7) Terminate program:

exit terminates the program.

Unix always closes all open descriptors when a process terminates, so our TCP socket is now closed.

perror() function displays the description of the error that corresponds to an error code stored in the system variable errno.

errno is a system variable which holds the error code that describes the error condition. This error condition is produced by a call to a library function.

1.3 Protocol Independence:

IPV4	IPV6
sockaddr_in	sockaddr_in6
AF_INET	AF_INET6
sin_family	sin6_family
sin_port	sin6_port

Protocol independent day time client using getaddrinfo (which is called tcp_connect).

1.4 Error handling: Wrapper functions

→ In any real-world program, it is essential to check every function call for an error return. (Errors from socket, connect, read & fputs)
Ex: err_quit, err_sys to print an error message and terminate the program.

→ sometimes, we want to do something other than terminate when one of these functions returns an error. i.e., we must check for an interrupted system call.

→ Since terminating on an error is the common case, we can shorten our programs by defining a wrapper function that performs the actual function call, tests the return value and terminates on an error.

→ The convention we use is to capitalize the name of the function as in

```
sockfd = Socket (AF_INET, SOCK_STREAM, 0);
```

→ Our wrapper function is as shown below.

```
int
Socket (int family, int type, int protocol)
    {
        int n;
        if ( (n = socket (family, type, protocol))
                                        < 0 )
            err_sys ("socket error");
        return (n);
    }
```

* Thread functions do not set the standard Unix errno variable when an error occurs. Instead the errno value is the return value of the function.

→ This means that every time we call one of the pthread_ functions, we must allocate a variable, save the return value in that variable, and then set errno to this value before calling err_sys.

```
int n;
if ( (n = pthread_mutex_lock (&fooxx_mutex)) != 0)
    errno = n, err_sys ("pthread_mutex_lock error");
```

→ Alternatively, we could define a new error function that takes the system's error number as an argument.

→ But we can make this piece of code much easier to read as just

Pthread_mutex_lock(&ndone_mutex);

by defining our own wrapper function as shown in figure below:

```
void
Pthread_mutex_lock(pthread_mutex_t *mptr)
{
    int n;
    if ((n = pthread_mutex_lock(mptr)) == 0)
        return;
    errno = n;
    err_sys("pthread_mutex_lock error");
}
```

→ With careful C coding, we could use macros instead of functions, providing a little run-time efficiency, but these wrapper functions are rarely the performance bottleneck of a program.

A macro is a name given to block of C statements as a pre-processor directive.

Being a preprocessor, the block of code is communicated to the compiler before entering into the actual coding [main() function]

→ A macro is defined with pre-processor directive. Macros are pre-processed which means that all the macros would be processed before your program compiles.

→ However wrapper functions are not preprocessed but compiled.

→ In macros, no type checking (incompatible operand, etc.) is done and thus use of macros can lead to errors.

→ However, this is not the case with functions.

→ Also, macros do not check for compilation error if any.

Using errno Value:

→ When an error occurs in a Unix function (such as one of the socket functions), the global variable errno is set to a positive value indicating the type of error & the function normally returns -1.

→ Our err_sys function looks at the value of errno & prints the corresponding error message string (ex: "connection timed out" if errno equals ETIMEDOUT).

→ The value of errno is set by a function only if an error occurs

→ Its value is undefined if the function does not return an error.

→ All the positive error values are constants with all uppercase names beginning with "E", and are normally defined in the <sys/errno.h> header.

→ No! error has a value of 0.

→ Storing errno in a global variable does not work with multiple threads that share all global variables.

1.5 A Simple Daytime Server.

(1) → Create a TCP socket
— identical to the client.

(2) Bind server's well-known port to socket
— The server's well known port (13 for the daytime service) is bound to the the socket by filling in an Internet socket address structure and calling bind.

— We specify the IP address as INADDR_ANY, which allows the server to accept a client on any interface, in case the server host

has multiple interfaces.

③ Convert socket to listening socket

- By calling listen, the socket is converted into a listening socket, on which incoming connections from clients will be accepted by the kernel.

- these three steps, socket, bind and listen are the normal steps for any TCP server to prepare listening descriptor.

④ Accept client connection, send reply

- Normally, the server process is put to sleep in the call to accept, waiting for client connection to arrive and be accepted.

- A TCP connection uses what is called three-way handshake to establish a connection.

- When this handshake completes, accept returns, and the return value from the function is a new descriptor (connectfd) that called the connected descriptor.

- This new descriptor is used for communication with the new client

- A new descriptor is returned by accept for each client that connects to our server.

- The current time & date are returned by the library function time, which returns the number of seconds since the Unix Epoch: 00:00:00 January 1, 1970, coordinated Universal Time (UTC).

→ The next library function ctime converts this integer value into a human readable string such as

Mon May 26 20:50:40 2003

→ A carriage return & linefeed are appended to the string by sprintf, & the result is written to the client by write.

(5) terminate connection.

- The server closes its connection with the client by calling close.

- This initiates the normal TCP connection termination sequence: a FIN is sent in each direction & each FIN is acknowledged by the other end.

- program is protocol dependent
- server handles only one client at a time
- If multiple client connections arrive at about the same time, the kernel queues them up to some limit & returns them to accept one at a time.

- two library function time & ctime are quite fast, but if they take more time then we may find some way to overlap the service of one client with another client.

- iterative server: it iterates through each client one at a time.

- concurrent server: one that handles multiple clients at the same time.
(Simplest technique to for a concurrent server is to call the Unix fork() function, creating one child process for each client. other techniques are to use threads instead of fork(); or to pre-fork a fixed number of children when the server starts.)

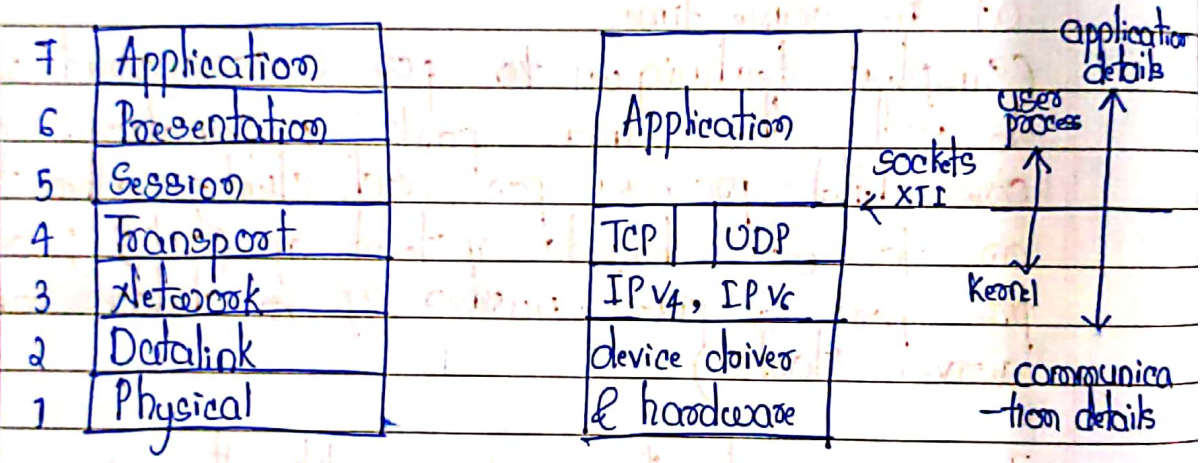
- If we want a server like this from a shell command line, we might want the server to run for a long time, since servers often run for as long as the system is up.

- This requires that we add code to the server to run correctly as a Unix daemon: a process that can run in the background, unattached to a terminal.

1.f OSI model:

→ A common way to describe the layers in a network is to use the International Organization for Standardization (ISO) Open Systems Interconnection (OSI) model for computer communications.

→ This is a seven-layer model as shown in figure below with appropriate mapping to the Internet protocol suite.



OSI Model Internet protocol suite

→ Datalink & physical layer corresponds to device driver & network hardware that are supplied with the system.

→ The network layer is handled by the IP v4 and IP v6 protocols.

→ The transport layer that we can choose from are TCP and UDP.

→ The Gap between TCP & UDP indicates that, an application may bypass the transport layer

& use IPv4 or IPv6 directly. This is called raw socket.

→ The upper three layers of the OSI model are combined into a single layer called the application.

→ This is the Web client (browser), Telnet client, Web server, FTP server, or any other applications that we are using.

→ With the internet protocols, there is rarely any discussion between the upper three layers of the OSI model.

→ Socket programming interfaces are interfaces from the upper three layers into the transport layer.

→ We focus on how to write applications using sockets that use either TCP or UDP.

→ Why do sockets provide the interface from the upper three layers of the OSI model into the transport layer?

1. The upper three layers handle all the details of the application (FTP, Telnet, or HTTP for example) & know little about the communication details.
   The lower four layers know little about the application, but handle all the communication details: sending data, waiting for acknowledgements, sequencing data that arrives out of order, calculating & verifying checksums & so on.

2. The second reason is that the upper three layers often form what is called a user process while the lower four layers are normally provided as part of the operating system kernel.

— Unix provides this separation between the user process and the kernel.

1.8 BSD Networking History (Self study)

1.9. Test Networks & Hosts.

— Fig 1.16 shows the various networks & hosts used in examples

— For each host the OS and the type of hardware is shown, since most of the operating systems run on more than one type of hardware.

— The name within each box is the hostname.

— Virtual Private Networks (VPNs) or secure shell (SSH) connections provide connectivity between these machines regardless of where they live physically.

— The notation "/24" indicates the number of consecutive bits starting from the left most bit of the address used to identify the network & subnet

Discovering Network Topology

- Two basic commands are provided by most UNIX systems and can be used to discover some details of network ; netstat and ifconfig

1. netstat -i : provides information on the interfaces. -n flag is used to print numeric addresses, instead of trying to find names for the networks.

→ This shows us the interfaces & their names.

(a) netstat -ni

- The loopback interface is called lo and the Ethernet is called eth0.

2. netstat -r : shows the routing table, which is another way to determine the interfaces.

(a) netstat -nr

3. Given the interface names, we execute ifconfig to obtain the details for each interface.

(a) ifconfig eth0
- This shows the IP address, subnet mask, & broadcast address.

- The MULTICAST flag is often an indication that the host supports multicasting

(-a) flag, which prints information on all configured interfaces.

IEC: International Electro Technical Commission.

4. One way to find the IP address of many hosts on the local network is to ping the broadcast address

(a) ping -b 206.168.112.127

1.10 Unix Standards

— Unix standardization was being done by the Austin Common Standards Revision Group (CSRG).
  - ISO/IEC 9945:2002,
  - IEEE Std 1003.1-2001
  - Single Unix Specification Version 3

— The POSIX Standard is considered throughout the discussion.
  - http://www.UNIX.org/version3

Background on POSIX:

— POSIX: is an acronym for Portable Operating System Interface

— Other standards: IEEE

— POSIX: adopted as international standards by ISO/IEC.

- IEEE Std 1003.1-1988:
  - first POSIX standard
  - Specified the 'C' language interface into a Unix like kernel.

- covered process primitives (fork, exec, signals & timers), the environment of a process (user IDs and process groups), files & directories (all the I/O functions), terminal I/O, system databases (password file and group file), and the tar and cpio archive formats.

- IEEE Std 1003.1-1990 :
  - known as ISO/IEC 9945-1:1990.
  - minimal changes to previous standard
  - Appended to the title "Part1: System Application Program Interface (API)
  - standard was 'C' language API

- IEEE Std 1003.2-1992
  - Title : "Part 2: Shell and utilities"
  - Defined shell (based on the System V Bourne shell)
  - About 100 utilities (programs normally executed from a shell, from awk and basename to vi and yacc.
  - referred to as POSIX.2

- IEEE Std 1003.1b-1993
  - known as IEEE P1003.4
  - included real-time extensions developed by P1003.4 working group.
  - following items were added:
    file synchronization, asynchronous I/O, semaphores, memory management (mmap & shared memory), execution scheduling, clocks & timers, & message queues

- IEEE Std 1003.1 1996 Edition [IEEE 1996]
  - included 1003.1-1990 (the base API), 1003.1b-1993 (real-time extensions), 1003.1c-1995 (pthreads), and 1003.1i-1995 (technical corrections to 1003.1b.
  - also called as ISO/IEC 9945-1:1996.
  - includes threads, thread synchronization (mutexes and condition variables), thread scheduling and synchronization scheduling
  - Referred to as POSIX.1.
  - This standard also contains a foreword stating that ISO/IEC 9945 consists of the following parts:

(Posix.1) Part 1: System API ('C' language)
(Posix.2) Part 2: Shell and Utilities
Part 3: System administration (under development)

- IEEE Std 1003.1g:
  - Protocol independent interfaces (PII) became an approved standard in 2000.
  - This is the networking API standard & it defines two APIs called as Detailed Network Interfaces (DNIs)

1. DNI/socket, based on the 4.4 BSD sockets API
2. DNI/XTI: based on the X/open XPG4 specification.

- referred to as POSIX.1g.
- available at : http://www.pasc.org/standing/sd11.html.

Self-Study:

- Background on The Open Group
- Unification of standards
- Internet Engineering Task Force (IETF)

1.11. 64-bit Architectures:

- During 1990s, the trend began toward 64-bit architectures and 64-bit software.
- One reason for larger addressing within a process (i.e., 64-bit pointers), which can address large amounts of memory (more than 2^{32} bytes).
- The common programming model for existing 32-bit Unix systems is called the ILP32 model, denoting that integers (I), long integers (L), and pointers (P) occupy 32-bits.
- The 64-bit Unix systems is called the LP64 model, meaning only long integers (L) and pointers (P) require 64-bits

- Figure below shows the comparison of these two models.

Datatype	ILP32 model	LP64 model
char	8	8
short	16	16
int	32	32
long	32	64
pointer	32	64

Fig

- From a programming perspective, the LP64 model means we cannot assume that a pointer can be stored in an integer.

- We must also consider the effect of the LP64 model on existing APIs.

- ANSI C invented the size_t datatype, which is used, for example, as the argument to malloc (the number of bytes to allocate), and as the third argument to read and write (the number of bytes to read or write)

- On a 32-bit system, size_t is a 32-bit value, but on a 64-bit system, it must be a 64-bit value to take advantage of the larger addressing model

- This means that 64-bit system will probably contain a typedef of size_t to be an unsigned long.

→ The networking API problem is that some drafts of POSIX.1g specified that function arguments containing the size of a socket address structure have the size_t datatype (ex: the third argument to bind and connect)

→ Some XTI structures also had members with a datatype of long (ex: the t_info and t_opthdr structures)

→ If these had been left as is, both would change from 32-bit values to 64-bit values when a Unix system changes from the ILP32 to the LP64 model.

→ In both instances, there is no need for a 64-bit datatype.

→ The length of the socket address structure is a few hundred bytes at most, and the use of long for the XTI structure members was a mistake.

→ The solution is to use datatypes designed specifically to handle these scenarios.

→ The socket API uses the socklen_t datatype for lengths of socket address structures, and XTI uses the t_scalar_t and t_uscalar_t datatypes.

→ The reason for not changing these values from 32-bits to 64 bits is to make it easier to provide binary compatibility on the new 64-bit systems for applications compiled under 32-bit systems.

The Transport Layer: TCP, UDP, and SCTP

→ Most client/server applications use either TCP or UDP.

→ SCTP (Stream Control Transmission Protocol) is a newer protocol designed for transport of telephony signaling across the Internet.

→ These transport protocols (TCP, UDP & SCTP) use the network-layer protocol IP, either IPV4 or IPV6.

→ UDP is a simple, unreliable datagram protocol, while TCP is sophisticated, reliable byte-stream protocol.

→ SCTP is similar to TCP as a reliable transport protocol, but it also provides message boundaries, transport-level support for multihoming, and a way to minimize head-of-line blocking.

→ We need to understand the services provided by these transport protocols to the application, so that we know what is handled by protocol & what we must handle in the application.

→ Why we need to understand the features of protocols?
- helps in writing robust clients & servers
- edges in debugging client & servers using tools like netstat.

→ Topics covered:
- TCP's three-way handshake · TCP's connection termination sequence & TCP's time wait state.

- SCTP's four-way handshake & SCTP's connection termination;
- SCTP, TCP and UDP buffering by the socket layer, and so on.

2.2 The Big Picture:

- Though the protocol suite is called "TCP/IP" there are more members of this family than just TCP and IP.
- Figure 2.1 shows an overview of these protocols (Text Book page no. 32)
- Both IPV4 and IPV6 are shown in figure.

IPV4 → AF_INET, sockaddr_in()
IPV6 → AF_INET6, sockaddr_in6()

i) tcp-dump: This application, tcpdump communicates directly with the datalink using either the BSD packet filter (BPF) or the data link provider interface (DLPI)

* except tcpdump other applications uses API which is normally sockets or XTL.

* There is an exception to this: Linux provides access to the datalink using a special type of socket called sock_PACKET

iii) trace-route: This program uses two sockets: one for IP and another for ICMP.

iii) IPv4: Internet Protocol v4:
- workhorse protocol of the IP suite since 1980s
- uses 32-bit addresses.
- provides packet delivery service for TCP, UDP, SCTP, ICMP, and IGMP.

iv) IPv6: Internet Protocol v6:
- designed in the mid-1990s as a replacement for IPv4.
- major change is larger address comprising 128 bits. to deal with the explosive growth of the internet in the 1990s.
- provides packet delivery service for TCP, UDP, SCTP and ICMPv6.

v) TCP: Transmission Control Protocol
- connection orient protocol.
- provides a reliable, full-duplex byte stream to its users.
- TCP sockets are an example of stream sockets.
- TCP takes care of details such as acknowledgements, timeouts, retransmission, and the like.
- most internet application programs use TCP.
- It use either IPv4 or IPv6.

vi) UDP: User Datagram Protocol
- UDP is a connectionless protocol, and UDP sockets are an example of datagram sockets.
- There is no guarantee that UDP

datagrams ever reach their intended destination
- uses either IP V4 or IP V6.

VII) SCTP: Stream Control Transmission Protocol.
- SCTP is a connection oriented protocol that provides a reliable, full-duplex ~~byte stream to its users~~ association.

- The word "association" is used when referring to a connection in SCTP because SCTP is multihomed, involving a set of IP addresses and a single port for each side of an association.

- SCTP provides a message service, which maintains record boundaries

- like TCP and UDP, SCTP can use either IPV4 or IPV6, but it can also use both IPV4 and IPV6 simultaneously on the same association.
-

VIII) ICMP: Internet Control Message Protocol.

- ICMP handles error and control information between routers and hosts

- These messages are normally generated by and processed by the TCP/IP networking software itself, not user processes, although we show the ping and traceroute programs, which use ICMP.

- It uses IPV4.

IX) IGMP: Internet Group Management Protocol.

- IGMP is used with multicasting which is optional with IP V4.

X) ARP: Address Resolution Protocol:

- ARP maps an IPV4 address int a hardware address (such as an Ethernet address)

- ARP is normally used on broadcast networks such as Ethernet, token ring, and FDDI, and is not needed on point-to-point networks.

XI) RARP: Reverse Address Resolution Protocol:

- RARP maps a hardware address into an IPV4 address.
- It is sometimes used when a diskless node is booting.

XII) ICMPV6: Internet control Message Protocol version 6.
ICMP6 combines the functionality of ICMP V4, IGMP, and ARP.

XIII) BPF: BSD Packet Filter:
The interface provides access to the datalink layer. It is normally found on Berkeley-derived kernels.

XIV) Datalink provider interface:

- The interface also provides access to the datalink layer. It is normally provided with SVR4

2.3 User Datagram Protocol (UDP)

- UDP is a simple transport-layer protocol.
- The application writes a message to a UDP socket, which is then encapsulated in a UDP datagram, which is then further encapsulated as an IP datagram, which is then sent to its destination.
- There is no guarantee that a UDP datagram will ever reach its final destination, that order will be preserved across the network, or that datagrams arrive only once.
- If a datagram reaches its final destination but the checksum detects an error or if the datagram is dropped in the network, it is not delivered to the UDP socket and is not automatically retransmitted.
- To ensure reliability: we can build lots of features into our application: acknowledgements from the other end, timeouts, retransmissions and so on.
- Each UDP datagram has a length. The length of the datagram is passed to the receiving application along with the data.
- UDP provides a connectionless service, as there need not be any long-term relationship between a UDP client and server.

— For example: a UDP client can create a socket and send a datagram to a given server and then send immediately send another datagram on the same socket to a different server.

— Similarly, a UDP server can receive several datagram on a single UDP socket, each from a different client.

2.4 Transmission Control Protocol (TCP)

— TCP provides connections between clients and servers.

— A TCP client establishes a connection with a given server, exchanges data with that server across the connection, and then terminates the connection.

— TCP also provides reliability. When TCP sends data to the other end, it requires an acknowledgement in return.

— If an acknowledgement is not received, TCP automatically retransmits the data and waits a longer amount of time.

— After some number of retransmissions, TCP will give up, with the total amount of time spent trying to send data typically between 4-10 minutes.

- TCP contains algorithms to estimate the round-trip time (RTT) between a client and server dynamically so that it knows how long to wait for an acknowledgement.
(Ex: RTT in LAN : ms ; RTT across WAN : seconds)

- TCP continuously estimates the RTT of a given connection, because the RTT is affected by variations in the network traffic.

- TCP also sequences the data by associating a sequence number with every byte that it sends.

- Example : Assume an application writes 2048 bytes to TCP socket, causing TCP to send two segments, the first containing the data with sequence numbers 1 - 1024 and the second containing the data with sequence numbers 1025 - 2048.

- A segment is the unit of data that TCP passes to IP.

- If the segments arrive out of order, the receiving TCP will reorder the two segments based on their sequence numbers before passing the data to the receiving application.

- If TCP receives duplicate data from its peer (say peer thought a segment was lost & retransmitted it when it wasn't really lost, the network was just overloaded), it can detect that the data has been duplicated (from the sequence numbers) & discard the duplicate data.

- TCP provides flow control.
  - TCP always tells its peer exactly how many bytes of data it is willing to accept from the peer at any one time. This is called the advertised window.
  - At any time, the window is the amount of room currently available in the receive buffer, guaranteeing that the sender cannot overflow the receive buffer.
- The window size changes dynamically over time. As data is received from the sender, the window size decreases, but as the receiving application reads data from the buffer, the window size increases.
- It is possible for the window to reach 0: When TCP's receive buffer for a socket is full & it must wait for the application to read data from the buffer, before it can take any more data from the peer.
- Finally, a TCP connection is full-duplex. This means that an application can send and receive data in both directions on a given connection at any time.
- This means that TCP must keep track of state information such as sequence numbers & window sizes for each direction of data flow: sending & receiving.

- After a full-duplex connection is established, it can be turned into a simplex connection if desired.

2.5 Stream Control Transmission Protocol: (SCTP)

- SCTP provides services similar to those offered by UDP and TCP.
- SCTP provides associations between clients & servers.
- Features of SCTP:
  - reliable
  - sequencing
  - flow-control
  - full duplex data transfer
- The word "association" is used in SCTP instead of "connection" to avoid connotation that a connection involves communication between only two IP addresses.
- An association refers to a communication between two systems, which may involve more than two addresses due to multihoming.
- Unlike TCP, SCTP is message-oriented. It provides sequenced delivery of individual records.
- Like UDP, the length of a record written by sender is passed to the receiving application.
- SCTP can provide multiple streams between connection endpoints, each with its own reliable sequenced delivery of messages.

→ A lost message in one of these streams does not block delivery of messages in any of the other streams.

→ This approach is in contrast to TCP, where a loss at any point in the single stream of bytes blocks delivery of all future data on the connection until the loss is repaired.

→ SCTP also provides a multihoming feature, which allows a single SCTP endpoint to support multiple IP addresses.

→ This feature can provide increased robustness against network failure.

→ An endpoint can have multiple redundant network connections, where each of these networks has a different connection to the Internet infrastructure.

→ SCTP can work around a failure of one network or path across the Internet by switching to another address already associated with the SCTP association.

2.6 TCP connection Establishment and Termination.

Objectives:

- To understand connect, accept and close functions
- To debug TCP applications using the netstat program.
- understand how TCP connections are established & terminated, and TCP's state transition diagram.

Three-Way Handshake:

The following scenario occurs when a TCP connection is established.

1. The server must be prepared to accept an incoming connection.
   This is normally done by calling socket, bind and listen and is called a passive open.

2. The client issues an active open by calling connect. This causes the client TCP to send a "synchronize" (SYN) segment, which tells the server the client's initial sequence number for the data that the client will send on the connection.
   Normally, there is no data sent with SYN; it just contains an IP header, a TCP header, and possible TCP options.

3. The server must acknowledge (ACK) the client's SYN and the server must also send its own SYN

containing the initial sequence number for the data that the server will send on the connection.

The server sends its SYN and the ACK of the client's SYN in a single segment.

4. The client must acknowledge the server's SYN.

→ The minimum number of packets required for this exchange is three; hence, this is called TCP's three way handshake.

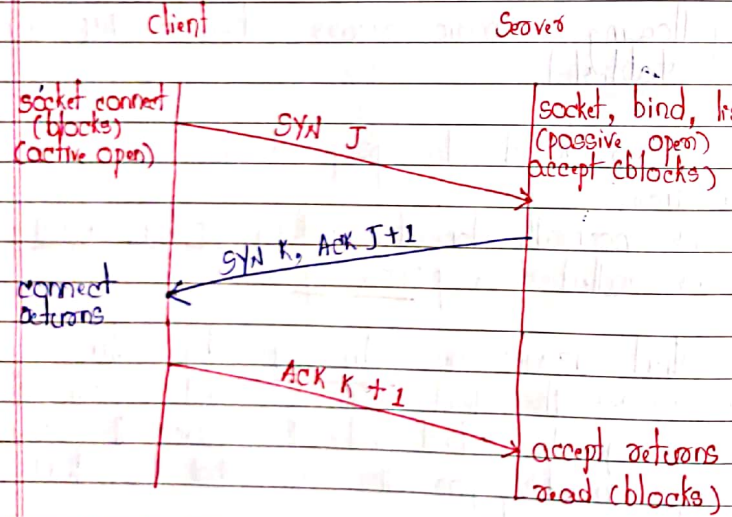


Figure: TCP three-way handshake.

→ It is shown that client's initial sequence number as J and the server's initial sequence number as K

→ The acknowledgement number in an ACK is the next expected sequence for the end sending the ACK.

→ Since a SYN occupies one byte of the sequence number space, the acknowledgement number in the ACK of each SYN is the initial sequence number plus one

→ Similarly, the ACK of each FIN is the sequence number of the FIN plus one.

TCP options:

→ Each SYN can contain TCP options. Commonly used options include the following:

a) MSS option:

→ With this option, the TCP sending the SYN announces its maximum segment size, the maximum amount of data that it is willing to accept in each TCP segment, on this connection.

→ The sending TCP uses the receiver's MSS value as the maximum size of a segment that it sends.

b) Window scale option:

→ The maximum window that either TCP can advertise to the other TCP is 65,535 bytes because the corresponding field in the TCP header occupies 16 bits

C.

Timestamp option: This option is needed for high-speed connections to prevent possible data corruption caused by old, delayed or duplicated segments.

TCP connection Termination:

→ Three segments to establish a connection.

→ Four segments to terminate a connection.

1. One application calls close first, & this end point performs the active close. This end's TCP sends a FIN segment, which means it is finished sending data.

2. The other end that receives the FIN performs the passive close. The received FIN is acknowledged by TCP.
The receipt of the FIN is also passed to the application as an end-of-file.
The receipt of the FIN means the application will not receive any additional data on the connection.

3. Sometime later, the application that received the end-of-file will close its socket. This causes its TCP to send a FIN.

4. The TCP on the system that receives this final FIN (the end that did the active close) acknowledge the FIN.

→ Since a FIN and ACK are required in each direction, four segments are normally required.

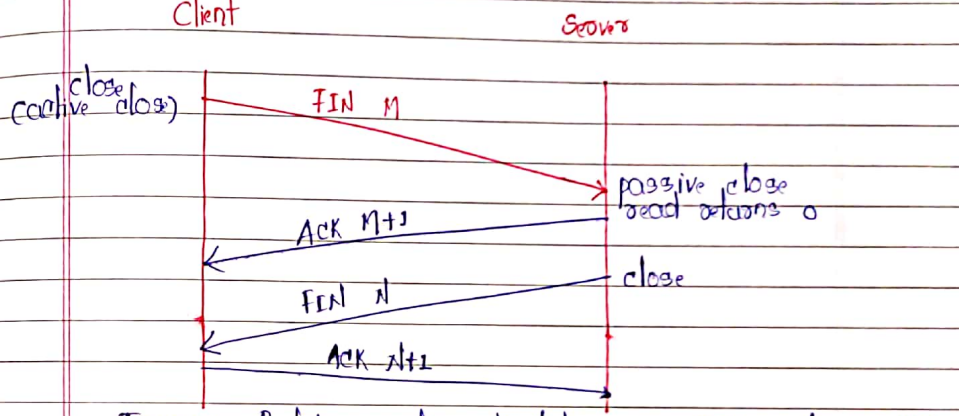


Figure: Packets exchanged when a TCP connection is closed.

TCP State Transition Diagram: