Unit – V

Transport Layer - Services - Connection Management - Addressing, Establishing and Releasing a Connection - Simple Transport Protocol - Internet Transport Protocols (ITP) - Network Security: Cryptography

5.1 Transport Layer

The network layer provides end-to-end packet delivery using data-grams or virtual circuits. The transport layer builds on the network layer to provide data transport from a process on a source machine to a process on a destination machine with a desired level of reliability that is independent of the physical networks currently in use. It provides the abstractions that applications need to use the network.

Transport Entity: The hardware and/or software which make use of services provided by the network layer, (within the transport layer) is called transport entity.

Transport Service Provider: Layers 1 to 4 are called Transport Service Provider.

Transport Service User: The upper layers i.e., layers 5 to 7 are called Transport Service User.

Transport Service Primitives: Which allow transport users (application programs) to access the transport service.

TPDU (Transport Protocol Data Unit): Transmissions of message between 2 transport entities are carried out by TPDU. The transport entity carries out the transport service primitives by blocking the caller and sending a packet the service. Encapsulated in the payload of this packet is a transport layer message for the server's transport entity. The task of the transport layer is to provide reliable, cost-effective data transport from the source machine to the destination machine, independent of physical network or networks currently in use.

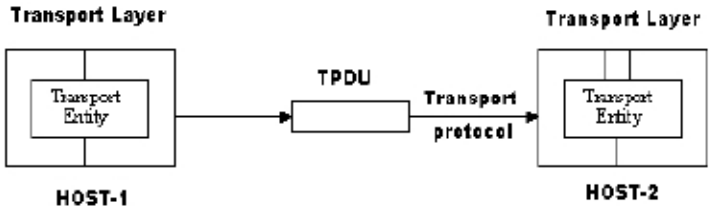

Figure 5.1: Transport Protocol Data Unit

5.2 Transport Services

1. Services Provided to the Upper Layers

The ultimate goal of the transport layer is to provide efficient, **reliable, and cost-effective data transmission** service to its users, normally processes in the application layer. To achieve this, the transport layer makes use of the **services pro-vided by the network layer.** The software and/or hardware within the transport layer that does the work is called the **transport entity.** The transport entity can be located in the operating system kernel, in a library package bound into network applications, in a separate user process, or even on the network interface card.

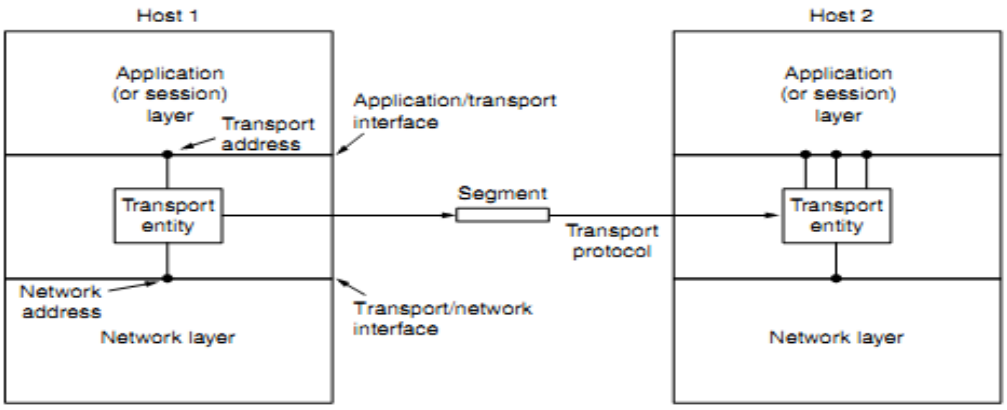

Figure 5.2: The network, Application and transport layer

There are two types of network service

- Connection-oriented
- Connectionless

Similarly, there are also two types of transport service. The connection-oriented transport service is similar to the connection-oriented network service in many ways.

In both cases, connections have three phases:

- Establishment
- Data transfer
- Release.

Addressing and flow control are also similar in both layers. Furthermore, the connectionless transport service is also very similar to the connectionless network service. The bottom four layers can be seen as the transport service provider, whereas the upper layer(s) are the transport service user.

2. Transport Service Primitives

To allow users to access the transport service, the transport layer must provide some operations to application programs, that is, a transport service interface. Each transport service has its own interface. The transport service is similar to the network service, but there are also some important differences.

The **main difference** is that the network service is intended to model the service offered by real networks. Real networks can lose packets, so the network service is generally **unreliable.**
The (connection-oriented) transport service, in contrast, is **reliable**

As an example, consider two processes connected by pipes in UNIX. They assume the connection between them is perfect. They do not want to know about acknowledgements, lost packets, congestion, or anything like that. What they want is a 100 percent reliable connection. Process A puts data into one end of the pipe, and process B takes it out of the other.

A **second difference** between the network service and transport service is **whom the services are intended for**. The network service is used only by the transport entities. Consequently, the transport service must be convenient and easy to use.

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

Figure 5.3 - The primitives for a simple transport service.

Eg: Consider an application with a server and a number of remote clients.

1. The server executes a “LISTEN” primitive by calling a library procedure that makes a System call to block the server until a client turns up.
2. When a client wants to talk to the server, it executes a “CONNECT” primitive, with “CONNECTION REQUEST” TPDU sent to the server.
3. When it arrives, the TE unblocks the server and sends a “CONNECTION ACCEPTED” TPDU back to the client.
4. When it arrives, the client is unblocked and the connection is established. Data can now be exchanged using “SEND” and “RECEIVE” primitives.
5. When a connection is no longer needed, it must be released to free up table space within the 2 transport entries, which is done with “DISCONNECT” primitive by sending “DISCONNECTION REQUEST”

TPDU: This disconnection can b done either by asymmetric variant (connection is released, depending on other one) or by symmetric variant (connection is released, independent of other one).

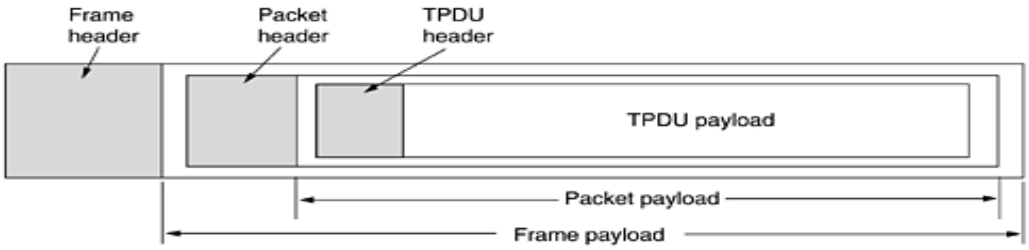

Figure 5.4 - Nesting of TPDUs, packets, and frames

- The term segment for messages sent from transport entity to transport entity.
- TCP, UDP and other Internet protocols use this term. Segments (exchanged by the transport layer) are contained in packets (exchanged by the network layer).
- These packets are contained in frames(exchanged by the data link layer).When a frame arrives, the data link layer processes the frame header and, if the destination address matches for local delivery, passes the contents of the frame payload field up to the network entity.
- The network entity similarly processes the packet header and then passes the contents of the packet payload up to the transport entity. This nesting is illustrated in Fig. 4.2.

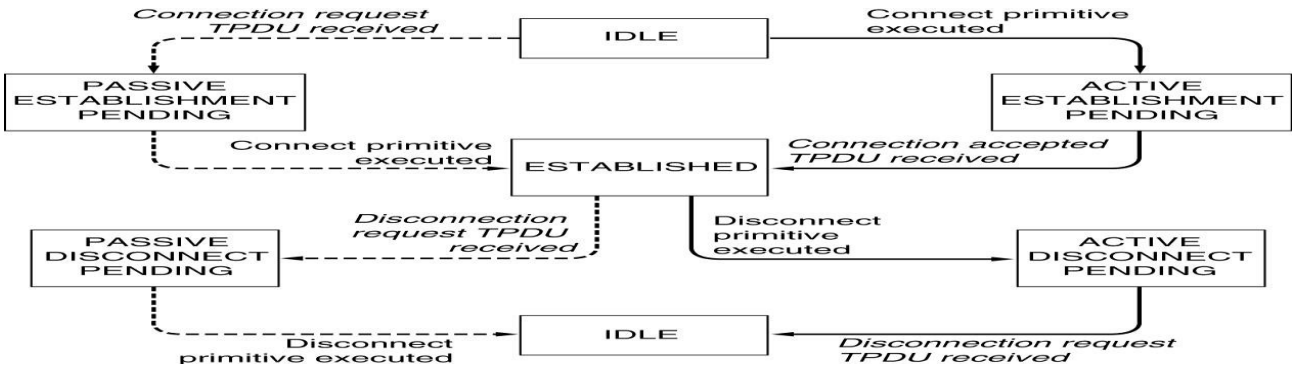

Figure 5.5 - A state diagram for a simple connection management scheme. Transitions labelled in italics are caused by packet arrivals. The solid lines show the client's state sequence. The dashed lines show the server's state sequence.

In fig. 4.3 each transition is triggered by some event, either a primitive executed by the local transport user or an incoming packet. For simplicity, we assume here that each TPDU is separately acknowledged. We also assume that a symmetric disconnection model is used, with the client going first. Please note that this model is quite unsophisticated. We will look at more realistic models later on.

3. Berkley Sockets

These primitives are socket primitives used in Berkley UNIX for TCP. The socket primitives are mainly used for TCP. These sockets were first released as part of the Berkeley UNIX 4.2BSD software distribution in 1983. They quickly became popular. The primitives are now widely used for Internet programming on many operating systems, especially UNIX -based systems, and there is a socket-style API for Windows called ‘‘**winsock**.’’

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

Figure 5.6 - The socket primitives for TCP.

The first four primitives in the list are executed in that order by servers.

1. The **SOCKET** primitive creates a new endpoint and allocates table space for it within the transport entity. The parameter includes the addressing format to be used, the type of service desired and the protocol. Newly created sockets do not have network addresses.
2. The **BIND** primitive is used to connect the newly created sockets to an address. Once a server has bound an address to a socket, remote clients can connect to it.
3. The **LISTEN** call, which allocates space to queue incoming calls for the case that several clients try to connect at the same time.
4. The server executes an **ACCEPT** primitive to block waiting for an incoming connection.

Some of the client side primitives are. Here, too, a socket must first be created

1. The **CONNECT** primitive blocks the caller and actively starts the connection process. When it completes, the client process is unblocked and the connection is established.
2. Both sides can now use **SEND** and **RECEIVE** to transmit and receive data over the full-duplex connection.
3. Connection release with sockets is symmetric. When both sides have executed a **CLOSE** primitive, the connection is released.

5.3 Elements of Transport Protocols

The transport service is implemented by a transport protocol used between the two transport entities. The transport protocols resemble the data link protocols. Both have to deal with error control, sequencing, and flow control, among other issues. The difference transport protocol and data link protocol depends upon the environment in which they are operated. These differences are due to major dissimilarities between the environments in which the two protocols operate, as shown in Fig 5.7.

At the data link layer, two routers communicate directly via a physical channel, whether wired or wireless, whereas at the transport layer, this physical channel is replaced by the entire network. This difference has many important implications for the protocols.

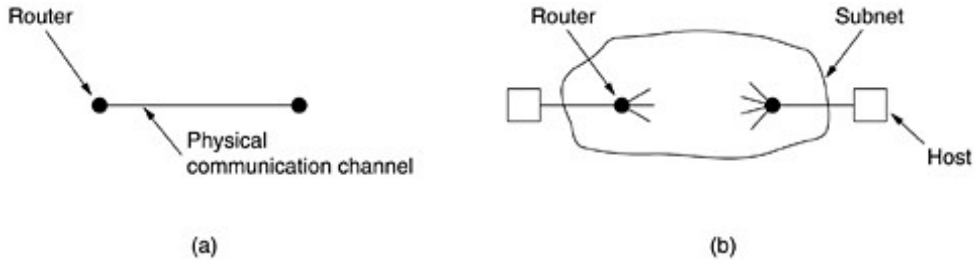

Figure 5.7 (a) Environment of the data link layer. (b) Environment of the transport layer.

In the data link layer, it is not necessary for a router to specify which router it wants to talk to. In the transport layer, explicit addressing of destinations is required.

In the transport layer, initial connection establishment is more complicated, as we will see. Difference between the data link layer and the transport layer is the potential existence of storage capacity in the subnet

Buffering and flow control are needed in both layers, but the presence of a large and dynamically varying number of connections in the transport layer may require a different approach than we used in the data link layer. The transport service is implemented by a transport protocol between the 2 transport entities.

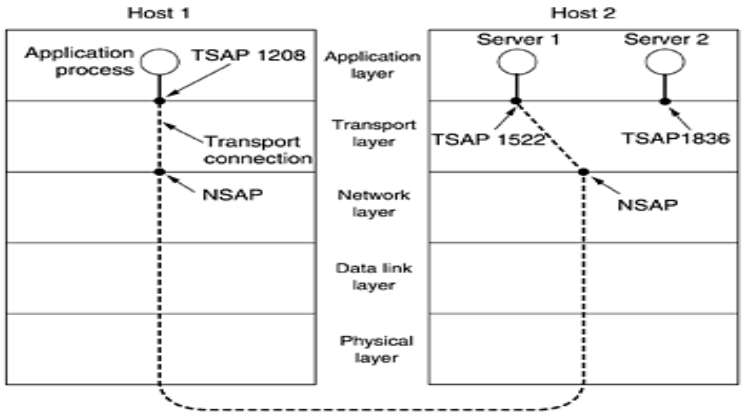

Figure 5.8 TSAP

Figure 5.8 illustrates the relationship between the NSAP, TSAP and transport connection. Application processes, both clients and servers, can attach themselves to a TSAP to establish a connection to a remote TSAP.

These connections run through NSAPs on each host, as shown. The purpose of having TSAPs is that in some networks, each computer has a single NSAP, so some way is needed to distinguish multiple transport end points that share that NSAP.

The elements of transport protocols are:

1. ADDRESSING
2. Connection Establishment.
3. Connection Release.
4. Error control and flow control
5. Multiplexing.

1. ADDRESSING

When an application (e.g., a user) process wishes to set up a connection to a remote application process, it must specify which one to connect to. The method normally used is to define transport addresses to which processes can listen for connection requests. In the Internet, these endpoints are called **ports**.

There are two types of access points.

TSAP (Transport Service Access Point) to mean a specific endpoint in the transport layer. The analogous endpoints in the network layer (i.e., network layer addresses) are not surprisingly called **NSAPs (Network Service Access Points).** IP addresses are examples of NSAPs.

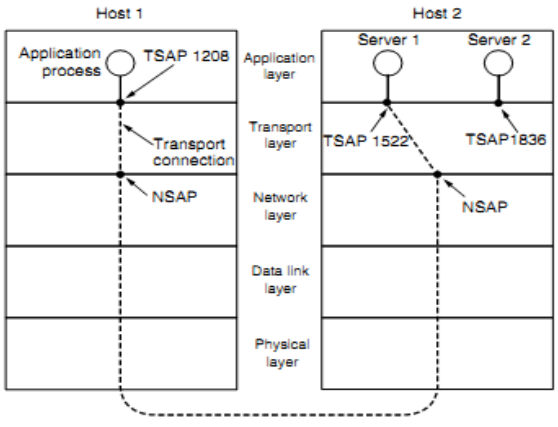

Figure 5.9 NSAP

Application processes, both clients and servers, can attach themselves to a local TSAP to establish a connection to a remote TSAP. These connections run through NSAPs on each host. The purpose of having TSAPs is that in some networks, each computer has a single NSAP, so some way is needed to distinguish multiple transport endpoints that share that NSAP.

A possible scenario for a transport connection is as follows:

1. A mail server process attaches itself to TSAP 1522 on host 2 to wait for an incoming call. How a process attaches itself to a TSAP is outside the networking model and depends entirely on the local operating system. A call such as our LISTEN might be used, for example.
2. An application process on host 1 wants to send an email message, so it attaches itself to TSAP 1208 and issues a CONNECT request. The request specifies TSAP 1208 on host 1 as the source and TSAP 1522 on host 2 as the destination. This action ultimately results in a transport connection being established between the application process and the server.
3. The application process sends over the mail message.
4. The mail server responds to say that it will deliver the message.
5. The transport connection is released

2. CONNECTION ESTABLISHMENT:

With packet lifetimes bounded, it is possible to devise a fool proof way to establish connections safely. Packet lifetime can be bounded to a known maximum using one of the following techniques:

- Restricted subnet design
- Putting a hop counter in each packet
- Time stamping in each packet

Using a 3-way hand shake, a connection can be established. This establishment protocol doesn’t require both sides to begin sending with the same sequence number.

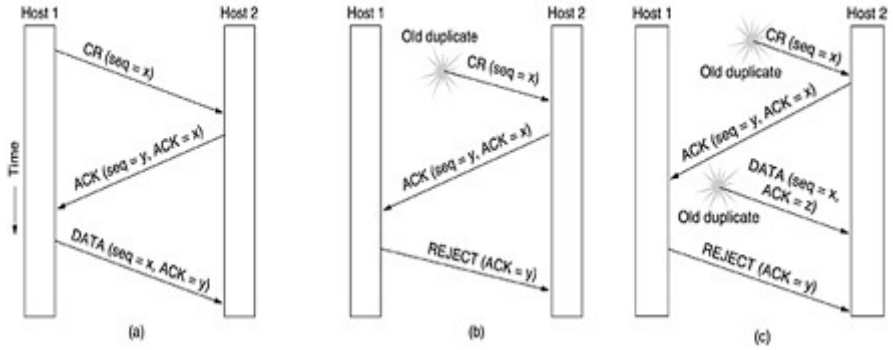

Figure 5.10: Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNEC TION REQUEST

(a) Normal operation. (b) Old duplicate CONNECTION REQUEST appearing out of nowhere. (c) Duplicate CONNECTION REQUEST and duplicate ACK .

The **first technique** includes any method that prevents packets from looping, combined with some way of bounding delay including congestion over the longest possible path. It is difficult, given that internets may range from a single city to international in scope.

The **second method** consists of having the hop count initialized to some appropriate value and decremented each time the packet is forwarded. The network protocol simply discards any packet whose hop counter becomes zero.

The **third method** requires each packet to bear the time it was created, with the routers agreeing to discard any packet older than some agreed-upon time.

In fig (A) Tomlinson (1975) introduced the three-way handshake.

This establishment protocol involves one peer checking with the other that the connection request is indeed current. Host 1 chooses a sequence number, x , and sends a CONNECTION REQUEST segment containing it to host 2. Host 2replies with an ACK segment acknowledging x and announcing its own initial sequence number, y.

Finally, host 1 acknowledges host 2's choice of an initial sequence number in the first data segment that it sends

In fig (B) the first segment is a delayed duplicate CONNECTION REQUEST from an old connection.

This segment arrives at host 2 without host 1's knowledge. Host 2 reacts to this segment by sending host1an ACK segment, in effect asking for verification that host 1 was indeed trying to set up a new connection.

When host 1 rejects host 2's attempt to establish a connection, host 2 realizes that it was tricked by a delayed duplicate and abandons the connection. In this way, a delayed duplicate does no damage.

The worst case is when both a delayed CONNECTION REQUEST and an ACK are floating around in the subnet.

In fig (C) previous example, host 2 gets a delayed CONNECTION REQUEST and replies to it.

At this point, it is crucial to realize that host 2 has proposed using y as the initial sequence number for host 2 to host 1 traffic, knowing full well that no segments containing sequence number y or acknowledgements to y are still in existence.

When the second delayed segment arrives at host 2, the fact that z has been acknowledged rather than y tells host 2 that this, too, is an old duplicate. The important thing to realize here is that there is no combination of old segments that can cause the protocol to fail and have a connection set up by accident when no one wants it.

3. CONNECTION RELEASE:

A connection is released using either asymmetric or symmetric variant. But, the improved protocol for releasing a connection is a 3-way handshake protocol.

There are two styles of terminating a connection:

1) Asymmetric release

Asymmetric release is the way the telephone system works: when one party hangs up, the connection is broken.

2) Symmetric release.

Symmetric release treats the connection as two separate unidirectional connections and requires each one to be released separately.

Fig-(a)	Fig-(b)	Fig-(c)	Fig-(d)

• One of the user sends a DISCONNECTION REQUEST TPDU in order to initiate connection release. • When it arrives, the recipient sends back a DR-TPDU, too, and starts a timer. • When this DR arrives, the original sender sends back an ACK-TPDU and releases the connection. • Finally, when the ACK-TPDU arrives, the receiver also releases the connection.	• Initial process is done in the same way as in fig-(a). • If the final ACK-TPDU is lost, the situation is saved by the timer. • When the timer is expired, the connection is released.	• If the second DR is lost, the user initiating the disconnection will not receive the expected response, and will timeout and starts all over again.	• Same as in fig-(c) except that all repeated attempts to retransmit the DR is assumed to be failed due to lost TPDUs. After 'N' entries, the sender just gives up and releases the connection.

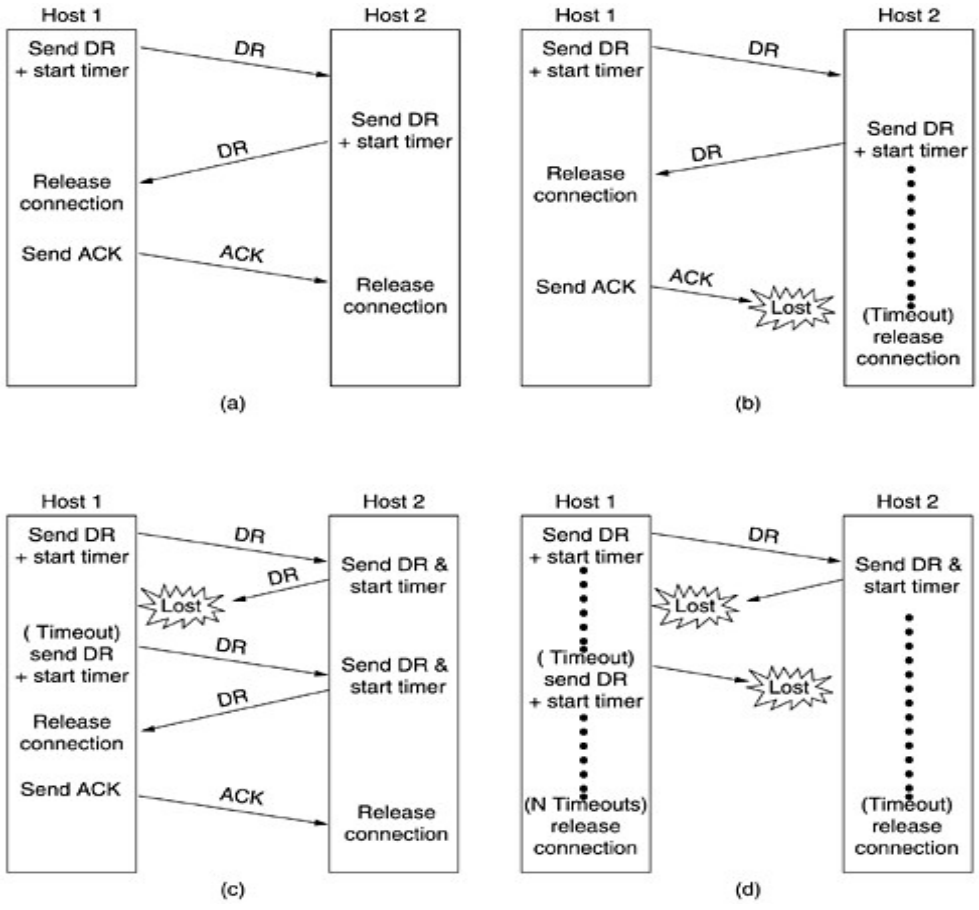

Figure 5.11: Connection Release

4. FLOW CONTROL AND BUFFERING:

Flow control is done by having a sliding window on each connection to keep a fast transmitter from over running a slow receiver. Buffering must be done by the sender, if the network service is unreliable. The sender buffers all the TPDUs sent to the receiver. The buffer size varies for different TPDUs.

They are:

(a). Chained Fixed-size Buffers:

If most TPDUs are nearly the same size, the buffers are organized as a pool of identical size buffers, with one TPDU per buffer.

(b). Chained Variable-size Buffers:

This is an approach to the buffer-size problem. i.e., if there is wide variation in TPDU size, from a few characters typed at a terminal to thousands of characters from file transfers, some problems may occur:

If the buffer size is chosen equal to the largest possible TPDU, space will be wasted whenever a short TPDU arrives.

If the buffer size is chosen less than the maximum TPDU size, multiple buffers will be needed for long TPDUs.

To overcome these problems, we employ variable-size buffers.

(c). One large Circular Buffer per Connection:

A single large circular buffer per connection is dedicated when all connections are heavily loaded.

1. Source Buffering is used for low band width bursty traffic
2. Destination Buffering is used for high band width smooth traffic.
3. Dynamic Buffering is used if the traffic pattern changes randomly.

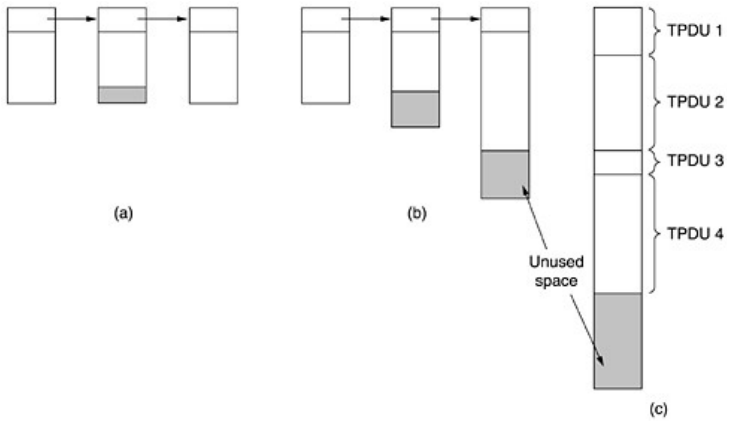

Figure 5.12: (a) Chained fixed-size buffers. (b) Chained variable-sized buffers. (c) One large circular buffer per connection.

5. MULTIPLEXING:

In networks that use virtual circuits within the subnet, each open connection consumes some table space in the routers for the entire duration of the connection. If buffers are dedicated to the virtual circuit in each router as well, a user who left a terminal logged into a remote machine, there is need for multiplexing. There are 2 kinds of multiplexing:

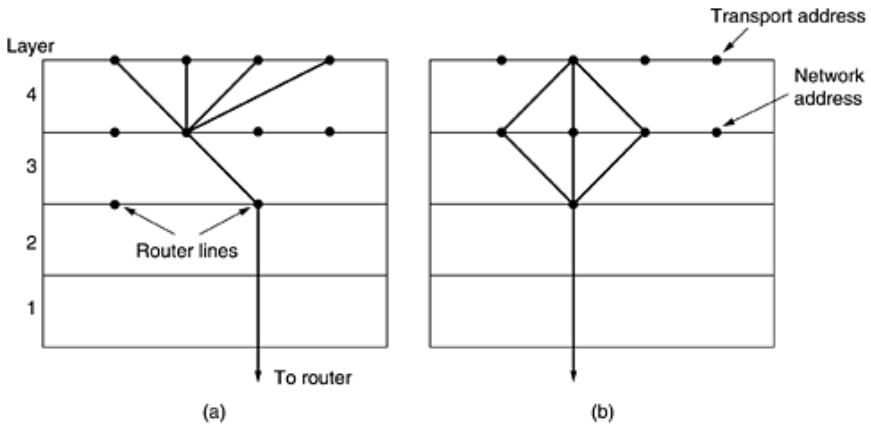

Figure 5.13: (a) Upward multiplexing. (b) Downward multiplexing

(a) Up-Ward Multiplexing:

In the above figure, all the 4 distinct transport connections use the same network connection to the remote host. When connect time forms the major component of the carrier's bill, it is up to the transport layer to group port connections according to their destination and map each group onto the minimum number of port connections.

(b) Down-Ward Multiplexing:

If too many transport connections are mapped onto the one network connection, the performance will be poor. If too few transport connections are mapped onto one network connection, the service will be expensive.

The possible solution is to have the transport layer open multiple connections and distribute the traffic among them on round-robin basis, as indicated in the above figure: With 'k' network connections open, the effective band width is increased by a factor of 'k'.

5.4 Transport Protocols

The Internet has two main protocols in the transport layer, **a connectionless protocol** and a **connection-oriented** one. The protocols complement each other. The connectionless

protocol is **UDP.** It does almost nothing beyond sending packets between applications, letting applications build their own protocols on top as needed.

The connection-oriented protocol is **TCP.** It does almost everything. It makes connections and adds reliability with retransmissions, along with flow control and congestion control, all on behalf of the applications that use it. Since UDP is a transport layer protocol that typically runs in the operating system and protocols that use UDP typically run in user s pace, these uses might be considered applications.

A. UDP (User Datagram Protocol)

i) Introduction to UDP

The Internet protocol suite supports a connectionless transport protocol called UDP (User Datagram Protocol). UDP provides a way for applications to send encapsulated IP datagrams without having to establish a connection.

UDP transmits segments consisting of an 8-byte header followed by the pay-load. The two ports serve to identify the end-points within the source and destination machines.

When a UDP packet arrives, its payload is handed to the process attached to the destination port. This attachment occurs when the BIND primitive. Without the port fields, the transport layer would not know what to do with each incoming packet. With them, it delivers the embedded segment to the correct application.

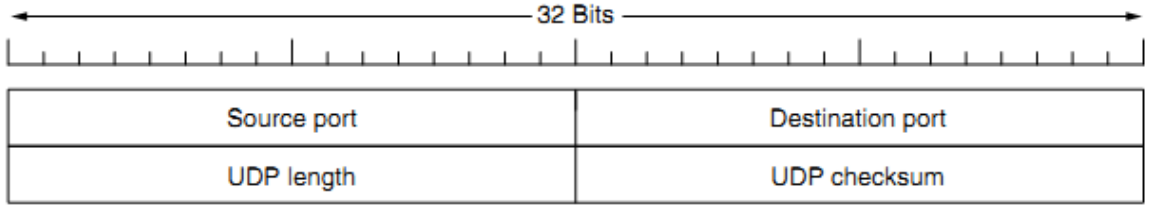

Figure 5.14: The UDP header

Source port, destination port: Identifies the end points within the source and destination machines.

UDP length: Includes 8-byte header and the data

UDP checksum: Includes the UDP header, the UDP data padded out to an even number of bytes if need be. It is an optional field

ii) Remote Procedure Call

In a certain sense, sending a message to a remote host and getting a reply back is like making a function call in a programming language. This is to arrange request-reply interactions on networks to be cast in the form of procedure calls.

For example, just imagine a procedure named *get IP address* (*host name*) that works by sending a UDP packet to a DNS server and waiting or the reply, timing out and trying again if one is not forthcoming quickly enough. In this way, all the details of networking can be hidden from the programmer.

RPC is used to call remote programs using the procedural call. When a process on machine 1 calls a procedure on machine 2, the calling process on 1 is suspended and execution of the called procedure takes place on 2. Information can be transported from the caller to the callee in the parameters and can come back in the procedure result. No message passing is visible to the application programmer. This technique is known as **RPC** (**Remote Procedure Call**) and has become the basis for many networking applications.

Traditionally, the calling procedure is known as the **client** and the called procedure is known as the **server.** In the simplest form, to call a remote procedure, the client program must be bound with a small library procedure, called the **client stub**,that represents the server procedure in the client's address space. Similarly, the server is bound with a procedure called the **server stub**. These procedures hide the fact that the procedure call from the client to the server is not local.

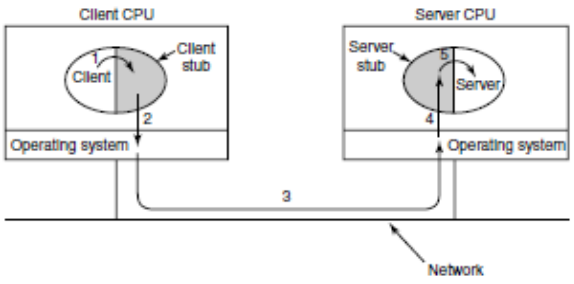

Figure 5.15: Remote Procedure Call

Steps in making a RPC

Step 1 is the client calling the client stub. This call is a local procedure call, with the parameters pushed onto the stack in the normal way.

Step 2 is the client stub packing the parameters into a message and making a system call to send the message. Packing the parameters is called **marshaling**.

Step 3 is the operating system sending the message from the client machine to the server machine.

Step 4 is the operating system passing the incoming packet to the server stub.
Step 5 is the server stub calling the server procedure with the **unmarshaled** parameters. The reply traces the same path in the other direction.

The key item to note here is that the client procedure, written by the user, just makes a normal (i.e., local) procedure call to the client stub, which has the same name as the server procedure. Since the client procedure and client stub are in the same address space, the parameters are passed in the usual way.

Similarly, the server procedure is called by a procedure in its address space with the parameters it expects. To the server procedure, nothing is unusual. In this way, instead of I/O being done on sockets, network communication is done by faking a normal procedure call. With RPC, passing pointers is impossible because the client and server are in different address spaces.

iii) Real-time Transport Protocols

It Described in RFC 3550. In widespread use for multimedia applications- Streaming Radio/Video, VOIP etc. Initially each application was having its own real time protocol, but they were all similar, but became apparent that a generic protocol that could handle multiple applications would be beneficial. It Has 2 parts. **First part** is the protocol for transporting audio and video data in packets. **Second part** is the processing that takes place- usually at the receiver to play the audio and video at the correct time.

a- The position of RTP in the protocol stack
B- Packet nesting

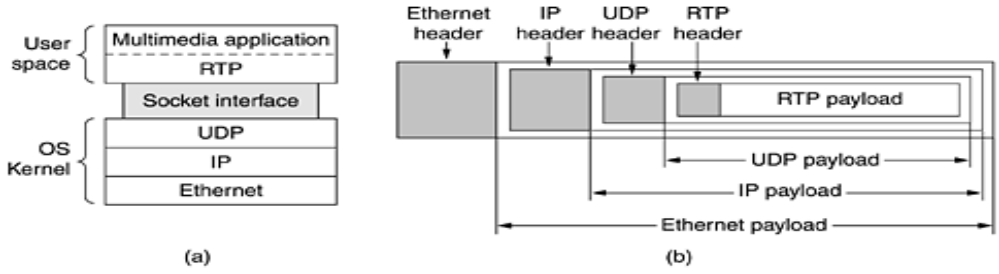

Figure 5.16: Real-time Transport Protocols

- The basic function is to multiplex several real time data streams onto a single stream of UDP packets which can then be sent to single (unicasting) or multiple (multicasting) destinations

- The UDP packets receive no special consideration except in the case of QOS features being enabled
- Each packet send is assigned a number 1 higher than the one before to determine if any are missing
- If a packet is not received then that data is skipped or approximated. There is not usually retransmission, nore is there an acknowledgement
- RTP paylods may contain multiple samples, each encoded multiple ways.
- RTP only provides the header field specifying the encoding only
- RTP supports time stamping
- Time stamping reduces effects of network delay and also allows multiple streams to be synchronized with each other

RTP Header

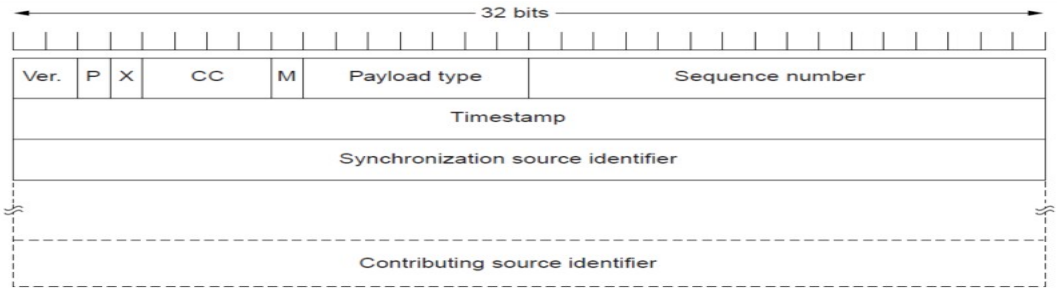

Figure 5.17: RTP Header

- Consists of three 32-bit words and possibly som extensions
- The first word contains the Version field
- The P bit indicates the packet has been padded to a multiple of 4 bytes. The final padding byte indicates how many bytes were added
- The X bit indicates an extension header is present. The only thing defined is that the first word of the extension gives the length
- The CC field tells how many contributing sources are present from 0 to 15
- The M bit is application specific marker bit. It is used to mark the start of something the application understand
- The Payload type tells which encoding algorithm has been used
- The Sequence number is the counter incremented on each RTP packet sent. Used to detect lost packets

- The Timestamp is produced by the source to indicate when the first sample was made. Only the differences between timestamps is significant.
- Synchronization source identifier identifies which stream the packet belongs to. This is the method used in multiplexing and demultiplexing multiple data streams into a single stream of UDP packets.
- Contributing source identifiers are used when mixers are present in the studio (if at all)

RTCP- Buffering and Jitter Control

- Jitter is caused by packets reaching the receiver at slightly different relatives times. Effects video and audio
- Buffering is the process of delaying playback a set amount of time so all the necessary packets for that period to arrive and be synchronized. This is a continuous process
- If a packet is delayed too long the media will either skip it or stop playing until the packet arrives. Usually in live applications such as VOIP calls, the packet is skipped.
- Streaming applications can use a larger buffer, but Live media applications typically use a smaller buffer because responsiveness is a priority

B. TCP (Transmission Control Protocol)

TCP was specifically designed to provide a reliable end-to end byte stream over an unreliable network. It was designed to adapt dynamically to properties of the inter network and to be robust in the face of many kinds of failures.

Each machine supporting TCP has a TCP transport entity, which accepts user data streams from local processes, breaks them up into pieces not exceeding 64kbytes and sends each piece as a separate IP datagram. When these datagrams arrive at a machine, they are given to TCP entity, which reconstructs the original byte streams. It is up to TCP to time out and retransmits them as needed, also to reassemble datagrams into messages in proper sequence.

The different issues to be considered are:

1. The TCP Service Model
2. The TCP Protocol
3. The TCP Segment Header
4. TCP Connection Establishment
5. TCP Connection Release

6. TCP Connection Management Modeling
7. TCP Sliding Window
8. TCP Timer Management
9. TCP Congestion Control

1. The TCP Service Model

TCP service is obtained by having both the sender and receiver create end points called **SOCKETS.** Each socket has a socket number(address)consisting of the IP address of the host, called a **"PORT"** (= TSAP)

To obtain TCP service a connection must be explicitly established between a socket on the sending machine and a socket on the receiving machine. All TCP connections are full duplex and point to point i.e., multicasting or broadcasting is not supported. A TCP connection is a byte stream, not a message stream i.e., the data is delivered as chunks

E.g.: 4 * 512 bytes of data is to be transmitted.

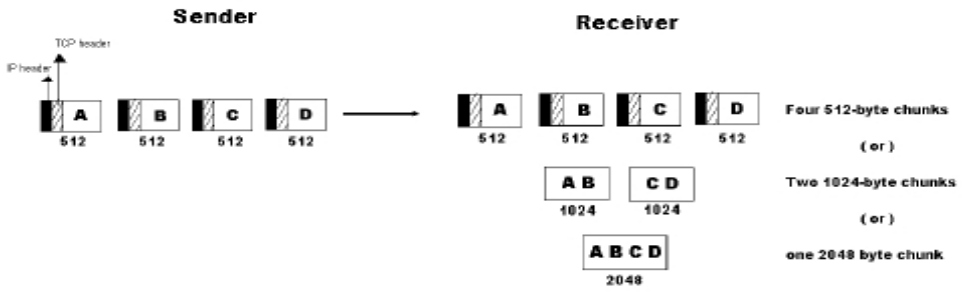

Figure 5.18: The TCP Service

Sockets:

A socket may be used for multiple connections at the same time. In other words, 2 or more connections may terminate at same socket. Connections are identified by socket identifiers at same socket. Connections are identified by socket identifiers at both ends. Some of the sockets are listed below:

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

Ports: Port numbers below 256 are called Well- known ports and are reserved for standard services.

***Eg*:**

PORT-21	To establish a connection to a host to transfer a file using FTP
PORT-23	To establish a remote login session using TELNET

2. The TCP Protocol

- A key feature of TCP, and one which dominates the protocol design, is that every byte on a TCP connection has its own 32-bit sequence number.
- When the Internet began, the lines between routers were mostly 56-kbps leased lines, so a host blasting away at full speed took over 1 week to cycle through the sequence numbers.
- The basic protocol used by TCP entities is the **sliding window protocol**.
- When a sender transmits a segment, it also starts a timer.
- When the segment arrives at the destination, the receiving TCP entity sends back a segment (with data if any exist, otherwise without data) bearing an acknowledgement number equal to the next sequence number it expects to receive.
- If the sender's timer goes off before the acknowledgement is received, the sender transmits the segment again.

3. The TCP Segment Header

Every segment begins with a fixed-format, 20-byte header. The fixed header may be followed by header options. After the options, if any, up to 65,535 - 20 - 20 = 65,495 data bytes may follow, where the first 20 refer to the IP header and the second to the TCP header. Segments without any data are legal and are commonly used for acknowledgements and control messages.

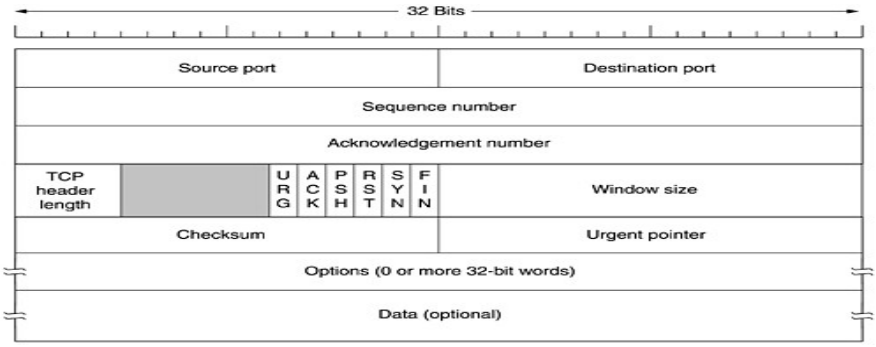

Figure 5.19: The TCP Header

Source Port, Destination Port : Identify local end points of the connections

Sequence number: Specifies the sequence number of the segment

Acknowledgement Number: Specifies the next byte expected.

TCP header length: Tells how many 32-bit words are contained in TCP header

URG: It is set to 1 if URGENT pointer is in use, which indicates start of urgent data.

ACK: It is set to 1 to indicate that the acknowledgement number is valid.

PSH: Indicates pushed data

RST: It is used to reset a connection that has become confused due to reject an invalid segment or refuse an attempt to open a connection.

FIN: Used to release a connection.

SYN: Used to establish connections.

4. TCP Connection Establishment

To establish a connection, one side, say, the server, passively waits for an incoming connection by executing the LISTEN and ACCEPT primitives, either specifying a specific source or nobody in particular.

The other side, say, the client, executes a CONNECT primitive, specifying the IP address and port to which it wants to connect, the maximum TCP segment size it is willing to accept, and optionally some user data (e.g., a password). The CONNECT primitive sends a TCP segment with the *SYN* bit on and *ACK* bit off and waits for a response.

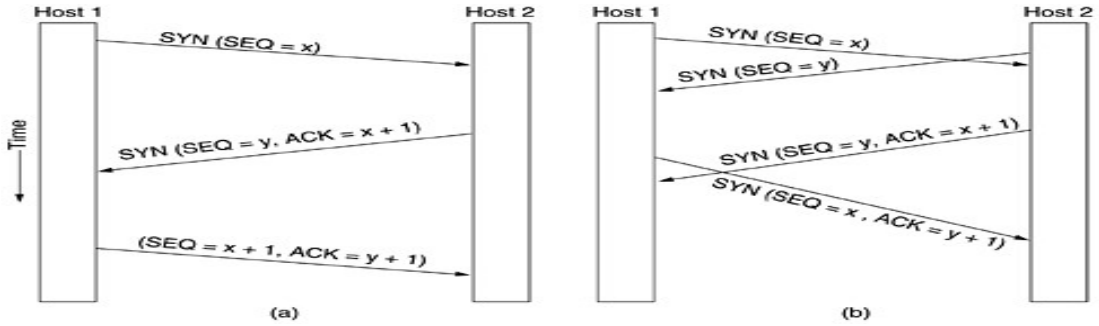

Figure 5.20: a) TCP Connection establishment in the normal case b) Call Collision

5. TCP Connection Release

- Although TCP connections are full duplex, to understand how connections are released it is best to think of them as a pair of simplex connections.
- Each simplex connection is released independently of its sibling. To release a connection, either party can send a TCP segment with the *FIN* bit set, which means that it has no more data to transmit.
- When the *FIN* is acknowledged, that direction is shut down for new data. Data may continue to flow indefinitely in the other direction, however.
- When both directions have been shut down, the connection is released.
- Normally, four TCP segments are needed to release a connection, one *FIN* and one *ACK* for each direction. However, it is possible for the first *ACK* and the second *FIN* to be contained in the same segment, reducing the total count to three.

6. TCP Connection Management Modeling

The steps required establishing and release connections can be represented in a finite state machine with the 11 states listed in Fig. 4.13. In each state, certain events are legal. When a legal event happens, some action may be taken. If some other event happens, an error is reported.

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

Figure 5.21: The states used in the TCP connection management finite state machine.

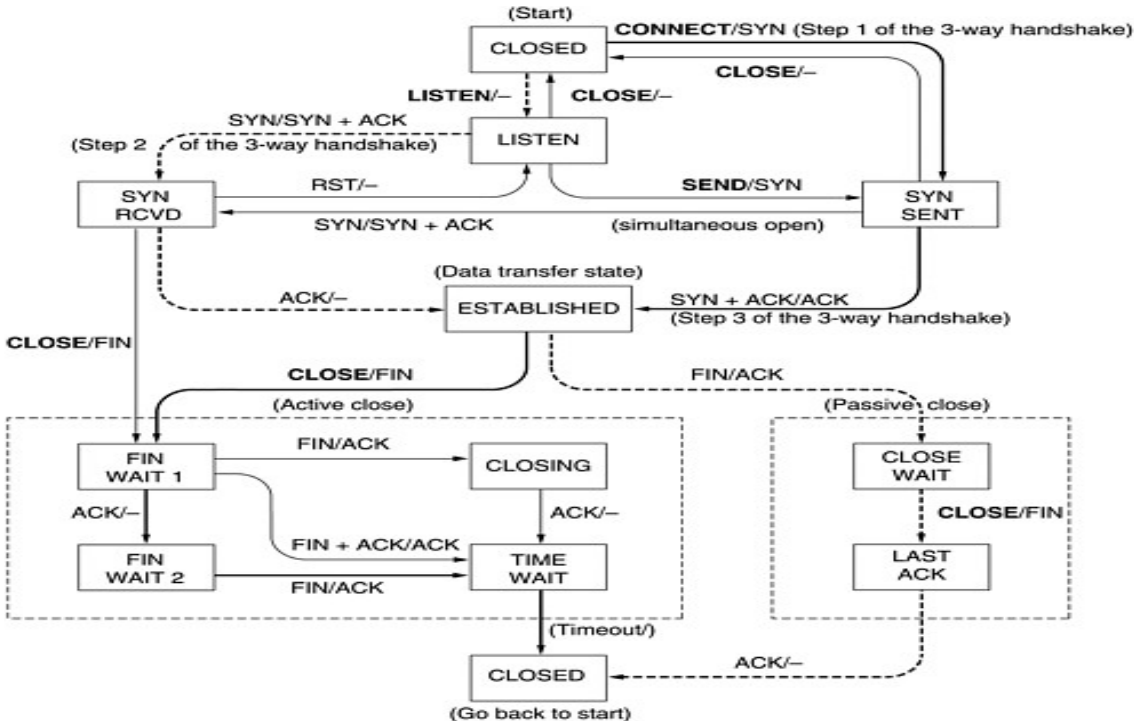

Figure 5.22: TCP connection management finite state machine.

TCP Connection management from server's point of view:

1. The server does a **LISTEN** and settles down to see who turns up.
2. When a **SYN** comes in, the server acknowledges it and goes to the **SYNRCVD** state
3. When the servers **SYN** is itself acknowledged the 3-way handshake is complete and server goes to the **ESTABLISHED** state. Data transfer can now occur.
4. When the client has had enough, it does a close, which causes a **FIN** to arrive at the server [dashed box marked passive close].
5. The server is then signaled.
6. When it too, does a **CLOSE**, a **FIN** is sent to the client.

7. When the client's acknowledgement shows up, the server releases the connection and deletes the connection record.

7. TCP Sliding Window

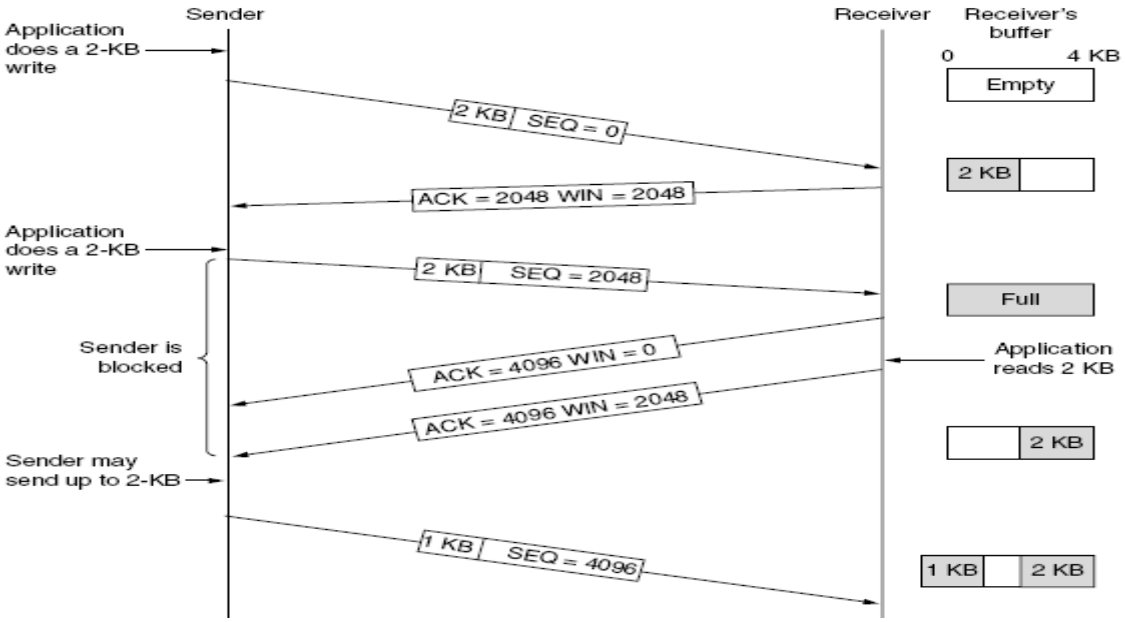

Figure 5.23: Window management in TCP.

1. In the above example, the receiver has 4096-byte buffer.
2. If the sender transmits a 2048-byte segment that is correctly received, the receiver will acknowledge the segment.
3. Now the receiver will advertise a window of 2048 as it has only 2048 of buffer space, now.
4. Now the sender transmits another 2048 bytes which are acknowledged, but the advertised window is'0'.
5. The sender must stop until the application process on the receiving host has removed some data from the buffer, at which time TCP can advertise a layer window.

i) Silly Window Syndrome:

This is one of the problems that ruin the TCP performance, which occurs when data are passed to the sending TCP entity in large blocks, but an interactive application on the receiving side reads 1 byte at a time.

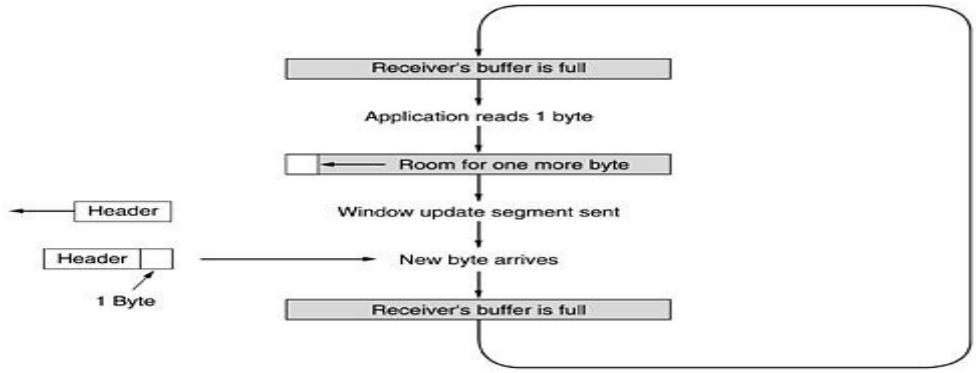

Figure 5.24: Silly Window Syndrome

Initially the TCP buffer on the receiving side is full and the sender knows this(win=0)

- Then the interactive application reads 1 character from tcp stream.
- Now, the receiving TCP sends a window update to the sender saying that it is all right to send 1 byte.
- The sender obligates and sends 1 byte.
- The buffer is now full, and so the receiver acknowledges the 1 byte segment but sets window to zero. This behavior can go on forever.

8. TCP Timer Management

TCP uses 3 kinds of timers:

1. Retransmission timer
2. Persistence timer
3. Keep-Alive timer.

1. Retransmission timer: When a segment is sent, a timer is started. If the segment is acknowledged before the timer expires, the timer is stopped. If on the other hand, the timer goes off before the acknowledgement comes in, the segment is retransmitted and the timer is started again. The algorithm that constantly adjusts the time-out interval, based on continuous measurements of n/w performance was proposed by JACOBSON and works as follows:

- For each connection, TCP maintains a variable RTT, that is the best current estimate of the round trip time to the destination inn question.
- When a segment is sent, a timer is started, both to see how long the acknowledgement takes and to trigger a retransmission if it takes too long.

- If the acknowledgement gets back before the timer expires, TCP measures how long the measurements took say M
- It then updates RTT according to the formula

$$\mathbf{RTT = \alpha RTT + (1-\alpha)\, M}$$

Where α = a smoothing factor that determines how much weight is given to the old value. Typically, **α =7/8**

Retransmission timeout is calculated as

$$\mathbf{D = \alpha\, D + (1-\alpha)\, |RTT - M|}$$

Where D = another smoothed variable, Mean RTT = expected acknowledgement value

M = observed acknowledgement value

$$\mathbf{Timeout = RTT + (4*D)}$$

2. Persistence timer:

It is designed to prevent the following deadlock:

- The receiver sends an acknowledgement with a window size of '0' telling the sender to wait later, the receiver updates the window, but the packet with the update is lost now both the sender and receiver are waiting for each other to do something
- When the persistence timer goes off, the sender transmits a probe to the receiver the response to the probe gives the window size
- If it is still zero, the persistence timer is set again and the cycle repeats
- If it is non zero, data can now be sent

3.Keep-Alive timer: When a connection has been idle for a long time, this timer may go off to cause one side to check if other side is still there. If it fails to respond, the connection is terminated.

9. TCP Congestion Control

TCP is the main network protocol in the Internet stack that's responsible for avoiding network congestion (IP simply drops packets).

TCP uses the window mechanism to control the data flow, and packet loss as the signal that there's congestion. It maintains a congestion window, separate from the current window value. The congestion window size is the number of bytes that the sender can have in

flight at a time. TCP stops sending traffic if either the congestion window or the window are full.

The initial congestion window is set to a small value of at most 4 segments. Each time an acknowledgement is received, the sender sends 1 extra segments worth of bytes. This algorithm is called **slow start**, and it exponentially increases the amount of data sent.

The sender keeps a slow start threshold for the connection. This is set high initially, so that it doesn't limit the connection. When packet loss is detected, the slow start threshold is set to half the congestion window. When the slow start threshold is reached, TCP switches to additive increase. The congestion window is increased by 1 segment each round trip time.

Instead of waiting for the retransmission timeout to detect packet loss, TCP also uses out-of-order packets as a signal that packets have been lost. When a receiver receives out of order packets, it sends duplicate acknowledgements to the sender so that the sender can retransmit the packet, which is presumed to be lost. This heuristic is known as **fast retransmission**.

Different congestion control algorithms used by TCP are:

- RTT variance Estimation.
- Exponential RTO back-off Re-transmission Timer Management
- Karn's Algorithm
- Slow Start
- Dynamic window sizing on congestion
- Fast Retransmit Window Management
- Fast Recovery

5.5 Cryptography

Cryptography is the study of secret (crypto) writing (graphy). The art of science encompassing the principles and methods of transforming an intelligible message into one that is intelligible, and then retransforming that message back to its original form.

Some of the encryption scheme are as follows

PLAINTEXT: This is the original intelligible message.

CIPHERTEXT: Transformed message.

ENCRYPTION ALGORITHMS: The encryption algorithm performs various substitutions and transformations on the plaintext.

SECRET KEY: Some critical information used by the cipher knows only to sender and receiver.

ENCIPHER(ENCODE): The process of converting plaintext to ciphertext.

DECIPHER(DECODE): The process of converting ciphertext back into plaintext.

Cryptography Systems are Characterised into three:

1) The type of operation used for transforming plaintext to ciphertext.

Two general principles

Substitution: In which each element in the plaintext (bit, letter, group or letter is mapped into another element.

Transposition: In which elements in plaintext are rearranged.

2) The numbers of keys used.

Both sender and receiver use same key as

- Symmetric
- Single key
- Secret key or conventional encryption.

Both sender and receiver use different key

- Asymmetric, two key or public key encryption.

3) The way in which plaintext is processed.

A Block cipher processes the input one block of elements at a time, producing an output block for each input block.(ie. Encrypt one bit/character at a time)

Eg: THIS IS EASY KEY(3)

WKLV LV HDVB.

A stream cipher processs the input elements continuously, producing output one element at a time.(ie. Break plaintext message in equal size blocks and encrypt each block as a unit).

Eg: THIS IS EASY

THI SIS EAS Y….. KEY(135)

UKN JWN……

1. Introduction to Cryptography

The messages to be encrypted, known as the plaintext, are transformed by a function that is parameterized by a key.

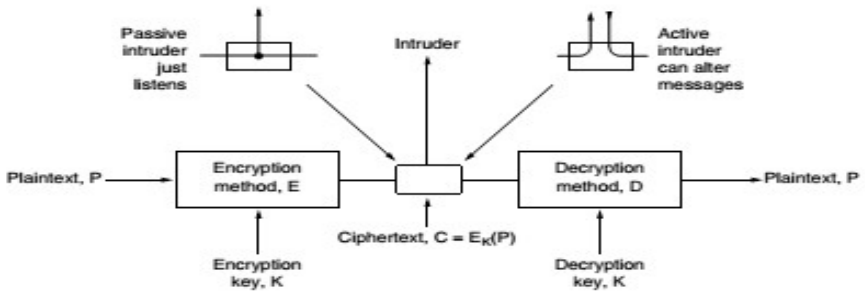

Figure 8-2. The encryption model (for a symmetric-key cipher).

Figure 5.25: Cryptography

The output of the encryption process, known as the ciphertext, is then transmitted, often by messenger or radio. It assumes that the enemy, or intruder, hears and accurately copies down the complete ciphertext. However, unlike the intended recipient, he does not know what the decryption key is and so cannot decrypt the ciphertext easily.

Sometimes the intruder can not only listen to the communication channel (passive intruder) but can also record messages and play them back later, inject his own messages, or modify legitimate messages before they get to the receiver(active intruder)

2. Substitution Techniques

The two basic building block of all encryption techniques are substitution and transposition. A substitution techniques is one in which the letters of plaintext are replaced by other letters or by number or symbols.

i. Caesar Cipher:

The earliest known use of a substitution cipher and the simplest, was by Julius Caesar. The Caesar cipher involves replacing each letter of alphabets with the letter standing three places further down the alphabets.

Eg: Plaintext: meet me after the toga party.

Ciphertext: PHHW PH DIWHU WKH WRJD SDUWB.

The letter following Z to A.

Plaintext: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Ciphertext: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

ii. Monoalphabetic Cipher

The general system of symbol for symbol substitution cipher. With the key being 26 letter string corresponding to the full alphabet.

Example: A->X, B->Y, C->Z, D->A…….Z->W

Example: starbucks at three

PQXOYRZHP XQ QEOBB

The basic attack takes advantage of the statistical properties of natural languages.

- In English, for example ,e is the most common letter, followed by t, o, a, n, i, etc.
- The most common two-letter combinations, or digrams, are th, in, er, re, it and an.
- The most common three-letter combinations, or trigrams, are the, ing, and, and ion.

3. Transposition Techniques

Substitution ciphers preserve the order of the plaintext symbols but disguise them. Transposition ciphers, in contrast, reorder the letters but do not disguise them. Figure 8-3 depicts a common transposition cipher, the columnar transposition. The cipher is keyed by a word or phrase not containing any repeated letters. In this example, **MEGABUCK is the key**. The purpose of the key is to order the columns, with column 1 being under the key letter closest to the start of the alphabet, and so on. The plaintext is written horizontally, in rows, padded to fill the matrix if need be. The ciphertext is read out by columns, starting with the column whose key letter is the lowest.

M	E	G	A	B	U	C	K	Plain Text
7	4	5	1	2	8	3	6	
p	l	e	a	s	e	t	r	Please transfer one million dollars to my Swiss bank account six two.
a	n	s	f	e	r	o	n	
e	m	i	l	l	i	o	n	**Cipher text**
d	o	l	l	a	r	s	t	**AFLLSKSOSELAWAIATOOSSC TCLNMOMANTESILYNRNNTSO WDPAEDOBUOERICXB**
o	m	y	s	w	i	s	s	
b	a	n	k	a	c	c	o	
u	n	t	s	I	x	t	w	
o	t	w	o	a	b	c	d	

Figure 5.26: Transposition Techniques

Some transposition ciphers accept a fixed-length block of input and produce a fixed-length block of output. These ciphers can be completely described by giving a list telling the order in which the characters are to be output.

WE ARE DISCOVERED SAVE YOURSELF would be written

```
W A E I C V R D A E O R E F
E R D S O E E S V Y U S L
```

or

```
W A E I C V R D A E O R E F E R D S O E E S V Y U S L .
```

To write the message in rectangle row by row and read the message off, column by column but permute the order of column. The order of column then becomes keyword AUTHOR and order the column by lexicographic order of the letters in the keyword.

Figure 5.27: Transposition Techniques

4. One-Time pad

One-time pad cipher is a type of Vignere cipher which includes the following features

- It is an unbreakable cipher.
- The key is exactly same as the length of message which is encrypted.
- The key is made up of random symbols.
- As the name suggests, key is used one time only and never used again for any other message to be encrypted.

Due to this, encrypted message will be vulnerable to attack for a cryptanalyst. The key used for a one-time pad cipher is called pad, as it is printed on pads of paper.

It is first described by Frank Miller in 1882. The one-time pad was re-invented in 1917 and patented a couple of years later. It is derived from the Vernam cipher, named after Gilbert Vernam, one of its inventors. Vernam's system was a cipher that combined a message with a key read from a punched tape. In its original form, Vernam's system was vulnerable because the key tape was a loop, which was reused whenever the loop made a full cycle. One-time use came later, when Joseph Mauborgne recognized that if the key tape were totally random, then cryptanalysis would be impossible.

Example:

Suppose Alice wishes to send the message "HELLO" to Bob. Assume two pads of paper containing identical random sequences of letters were somehow previously produced and securely issued to both. Alice chooses the appropriate unused page from the pad.

The way to do this is normally arranged for in advance, as for instance 'use the 12th sheet on 1 May', or 'use the next available sheet for the next message'. The material on the selected sheet is the key for this message. Each letter from the pad will be combined in a predetermined way with one letter of the message. It is common, but not required, to assign each letter a numerical value, e.g., "A" is 0, "B" is 1, and so on. In this example, the technique is to combine the key and the message using modular addition. The numerical values of corresponding message and key letters are added together, modulo 26. If key material begins with "XMCKL" and the message is "HELLO", then the coding would be done as follows:

H E L L O message

7 (H) 4 (E) 11 (L) 11 (L) 14 (O) message

+ 23 (X) 12(M) 2 (C) 10 (K) 11 (L) key

= 30 16 13 21 25 message + key

= 4 (E) 16 (Q) 13 (N) 21 (V) 25 (Z) message + key (mod 26)

E Q N V Z → ciphertext

If a number is larger than 25, then the remainder after subtraction of 26 is taken in modular arithmetic fashion. This simply means that if the computations "go past" Z, the sequence starts again at A.

The ciphertext to be sent to Bob is thus "EQNVZ". Bob uses the matching key page and the same process, but in reverse, to obtain the plaintext. Here the key is subtracted from the ciphertext, again using modular arithmetic:

E Q N V Z ciphertext

4 (E) 16 (Q) 13 (N) 21 (V) 25 (Z) ciphertext

- 23 (X) 12 (M) 2 (C) 10 (K) 11 (L) key

= -19 4 11 11 14 ciphertext – key

= 7 (H) 4 (E) 11 (L) 11 (L) 14 (O) ciphertext – key (mod 26)

H E L L O → message

Similar to the above, if a number is negative then 26 is added to make the number zero or higher.

The Security of the one-time pas is entirely due to the randomness of the key. If the Stream of characters that constitute the key is truly random, then the stream of characters that constitute the ciphertext will be truly random. Thus, there are no patterns or regularities that a cryptanalyst can use to attack the ciphertext.

The one-time pad offers complete security but, in practice, has two fundamental difficulties:

1. There is the practical problem of making large quantitied of random keys. Any heavily used system might requires millions of random character on a regular basis. Supply truly random characters in this volume is a significant task.

2. Even more daunting is the problem of key distribution and protection. For every message to be sent, a key of equal length is needed by both sender and receiver. Thus, a mammoth key distribution problem exists.

5. Two Fundamental Cryptographic Principles

i. Redundancy

The first principle is that all encrypted messages must contain some redundancy, that is, information not needed to understand the message. An example may make it clear why this is needed. Consider a mail-order company, The Couch Potato (TCP), with 60,000 products. Thinking they are being very efficient, TCP's programmers decide that ordering messages should consist of a 16- byte customer name followed by a 3-byte data field (1 byte for the quantity and 2 bytes for the product number). The last 3 bytes are to be encrypted using a very long key known only by the customer and TCP.

At first, this might seem secure, and in a sense it is because passive intruders cannot decrypt the messages. Unfortunately, it also has a fatal flaw that renders it useless. Suppose that a recently fired employee wants to punish TCP for firing her. Just before leaving, she takes the customer list with her. She works through the night writing a program to generate fictitious orders using real customer names. Since she does not have the list of keys, she just puts random numbers in the last 3 bytes, and sends hundreds of orders off to TCP.

When these messages arrive, TCP's computer uses the customers' name to locate the key and decrypt the message. Unfortunately for TCP, almost every 3- byte message is valid, so the computer begins printing out shipping instructions. While it might seem odd for a customer to order 837 sets of children's swings or 540 sandboxes, for all the computer knows, the customer might be planning to open a chain of franchised playgrounds. In this way, an active intruder (the exemployee) can cause a massive amount of trouble, even though she cannot understand the messages her computer is generating.

This problem can be solved by the addition of redundancy to all messages. For example, if order messages are extended to 12 bytes, the first 9 of which must be zeros, this attack no longer works because the ex-employee can no longer generate a large stream of valid messages. The moral of the story is that all messages must contain considerable redundancy so that active intruders cannot send random junk and have it be interpreted as a valid message.

However, adding redundancy makes it easier for cryptanalysts to break messages. Suppose that the mail-order business is highly competitive, and The Couch Potato's main competitor, The Sofa Tuber, would dearly love to know how many sandboxes TCP is selling so it taps TCP's phone line. In the original scheme with 3-byte messages, cryptanalysis was nearly impossible because after guessing a key, the cryptanalyst had no way of telling whether it was right because almost every message was technically legal. With the new 12-byte scheme, it is easy for the cryptanalyst to tell a valid message from an invalid one. Thus,

Cryptographic principle 1: Messages must contain some redundancy

In other words, upon decrypting a message, the recipient must be able to tell whether it is valid by simply inspecting the message and perhaps performing a simple computation. This redundancy is needed to prevent active intruders from sending garbage and tricking the receiver into decrypting the garbage and acting on the ‘‘plaintext.’’ However, this same

redundancy makes it much easier for passive intruders to break the system, so there is some tension here. Furthermore, the redundancy should never be in the form of n 0s at the start or end of a message, since running such messages through some cryptographic algorithms gives more predictable results, making the cryptanalysts' job easier.

ii. Freshness

The second cryptographic principle is that measures must be taken to ensure that each message received can be verified as being fresh, that is, sent very recently. This measure is needed to prevent active intruders from playing back old messages. If no such measures were taken, our ex-employee could tap TCP's phone line and just keep repeating previously sent valid messages. Thus,

Cryptographic principle 2: Some method is needed to foil replay attacks

One such measure is including in every message a timestamp valid only for, say, 10 seconds. The receiver can then just keep messages around for 10 seconds and compare newly arrived messages to previous ones to filter out duplicates. Messages older than 10 seconds can be thrown out, since any replays sent more than 10 seconds later will be rejected as too old.