COMPUTER NETWORK

B.TECH CSE

3rd Year – 1st Sem

UNIT – IV

Transport Layer

DEPARTMENT OF CSE

VIGNAN INSTITUTE OF TECHNOLOGY & SCIENCE
DESHMUKHI

THE TRANSPORT LAYER

In computer networking, the **transport layer** is a conceptual division of methods in the layered architecture of protocols in the network stack in the Internet protocol suite and the OSI model. The protocols of this layer provide end-to-end communication services for applications. It provides services such as connection-oriented communication, reliability, flow control, and multiplexing.

The transport layer builds on the network layer to provide data transport from a process on a **source** to **destination** machine with a desired level of reliability that is independent of the physical networks currently in use.

It is an end-to-end layer used to deliver messages to a host.

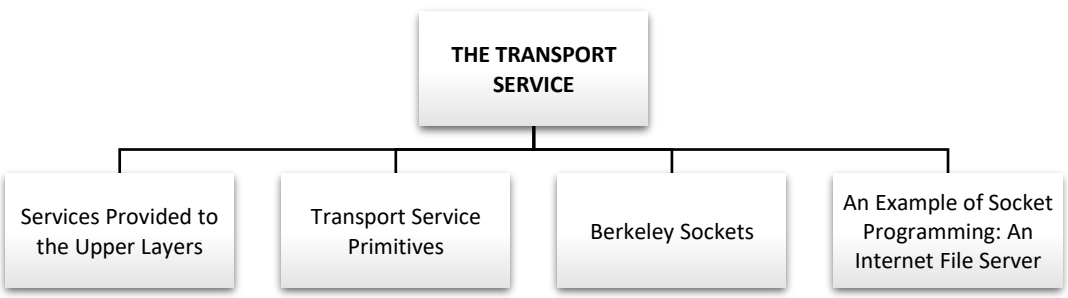
- ✓ Because it provides a point-to-point connection rather than hop-to-hop, between the source host and destination host to deliver the services reliably.
- ✓ It is responsible for ensuring that the data packets arrive accurately and reliably between sender and receiver.
- ✓ The transport layer most often uses TCP or User Datagram Protocol (UDP).
- ✓ In the TCP/IP network model, the transport layer comes between the application and network layers.

Responsibilities of a Transport Layer

- ✓ The Process to Process Delivery
- ✓ End-to-End Connection between Hosts
- ✓ Multiplexing and De-multiplexing
- ✓ Congestion Control
- ✓ Data integrity and Error correction
- ✓ Flow control

THE TRANSPORT SERVICE

What kind of service is provided to the application layer?



Services Provided to the Upper Layers

The main role of the transport layer is to provide the communication services directly to the application processes running on different hosts. The transport layer provides a logical communication between application processes running on different hosts. Although the application processes on different hosts are not physically connected, application processes use

the logical communication provided by the transport layer to send the messages to each other. The transport layer protocols are implemented in the end systems but not in the network routers. A computer network provides more than one protocol to the network applications. All transport layer protocols provide multiplexing/demultiplexing service. It also provides other services such as reliable data transfer, bandwidth guarantees, and delay guarantees. Each of the applications in the application layer has the ability to send a message by using TCP or UDP. The application communicates by using either of these two protocols. Both TCP and UDP will then communicate with the internet protocol in the internet layer. The applications can read and write to the transport layer. Therefore, we can say that communication is a two-way process.

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.

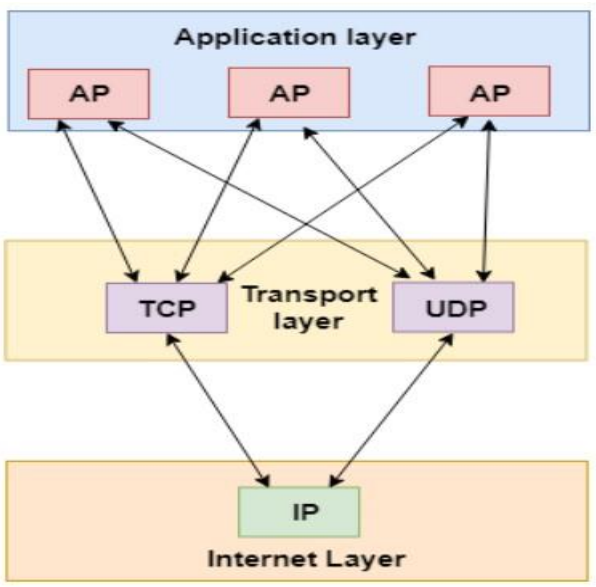
The software and/or hardware within the transport layer that does the work is called the **transport entity**.

The **connection-oriented transport service**

- ✓ Connections have three phases: **establishment**, **transfer** and **release**.

The **connectionless transport service**

- ✓ It can be difficult to provide a connectionless transport service on top of a connection-oriented network service, since it is inefficient to set up a connection to send a single packet and then tear it down immediately afterwards



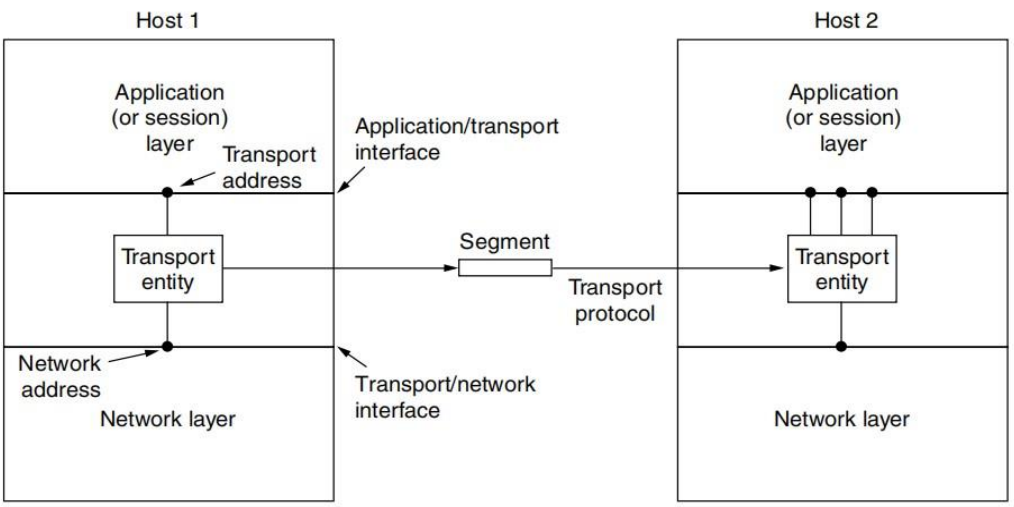


Figure 6-1. The network, transport, and application layers.

Transport Service Primitives

Transport service primitives play a crucial role in the field of networking and telecommunications. These primitives serve as the fundamental building blocks for the transport layer protocols, enabling reliable and efficient communication between network hosts

The primitives for a simple transport service:

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 main difference between network service and transport service is intended to model the service offered by real networks, warts and all. Real networks can l**ose packets**, so the network service is generally unreliable.

The **connection-oriented transport service**, in contrast, is reliable.

Of course, real networks are not error-free, but that is precisely the purpose of the transport layer — to provide a reliable service on top of an unreliable network.

There are **five types of service primitives**:

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.	Request a release of the connection

Figure 6-2. The primitives for a simple transport service.

- ✓ **LISTEN** : When a server is ready to accept an incoming connection it executes the **LISTEN** primitive. It blocks waiting for an incoming connection.
- ✓ **CONNECT** : It connects the server by establishing a connection. Response is awaited.
- ✓ **RECIEVE**: Then the **RECIEVE** call blocks the server.
- ✓ **SEND** : Then the client executes SEND primitive to transmit its request followed by the execution of RECIEVE to get the reply. Send the message.
- ✓ **DISCONNECT** : This primitive is used for terminating the connection.
  - ✓ After this primitive one can't send any message.
  - ✓ When the client sends **DISCONNECT** packet then the server also sends the **DISCONNECT** packet to acknowledge the client.
  - ✓ When the server package is received by client then the process is terminated.

➢ Connection Oriented Service Primitives

– There are 4 types of primitives for Connection Oriented Service :

- ✓ **CONNECT** : This primitive makes a connection
- ✓ **DATA, DATA-ACKNOWLEDGE, EXPEDITED-DATA** : Data and information is sent using thus primitive
- ✓ **DISCONNECT** : Primitive for closing the connection
- ✓ **RESET** : Primitive for reseting the connection

➢ Connectionless Oriented Service Primitives

• There are 2 types of primitives for Connectionless Oriented Service:

- ✓ **UNIDATA** : This primitive sends a packet of data
- ✓ **FACILITY, REPORT** : Primitive for enquiring about the performance of the network, like delivery statistics.

Berkeley Socket:

Berkeley sockets is an **application programming interface** (API) for Internet sockets and UNIX domain sockets. It is used for **inter-process communication (IPC)**. It is commonly implemented as a library of linkable modules. It originated with the 4.2BSD UNIX released in 1983.

Primitive	Meaning
SOCKET	Create a new communication endpoint
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection
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 6-5. The socket primitives for TCP.

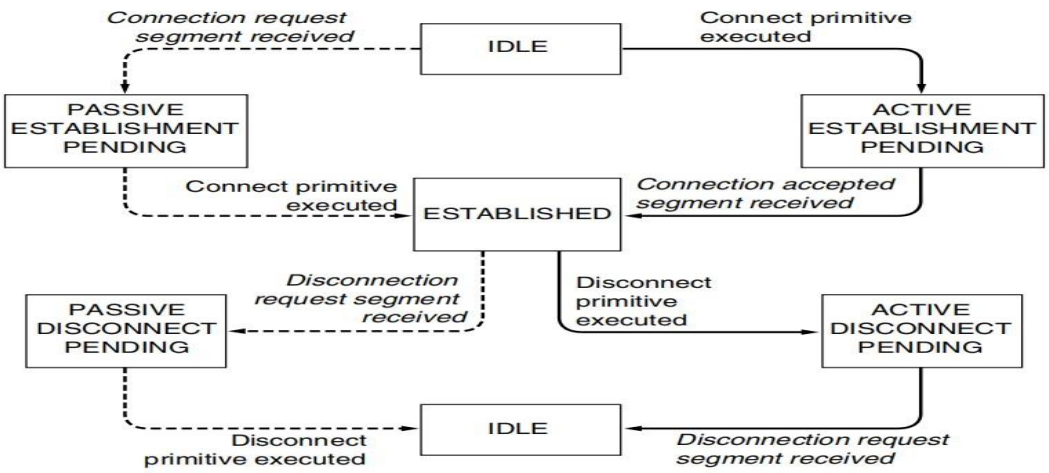


Figure 6-4. A state diagram for a simple connection management scheme. Transitions labeled in italics are caused by packet arrivals. The solid lines show the client's state sequence. The dashed lines show the server's state sequence.

Socket Programming:

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.

Server side:

- Server startup executes **SOCKET**, **BIND** & **LISTEN** primitives.
- ✓ **LISTEN** primitive allocate queue for multiple simultaneous clients.
- ✓ Then it use **ACCEPT** to suspend server until request.
- ✓ When client request arrives: **ACCEPT** returns.
- ✓ Start new socket (thread or process) with same properties as original, this handles the request, server goes on waiting on original socket.
- ✓ If new request arrives while spawning thread for this one, it is queued.

- ✓ If queue full it is refused.

Client side:

- ✓ It uses **SOCKET** primitives to create.
- ✓ Then use **CONNECT** to initiate connection process.
- ✓ When this returns the socket is open.
- ✓ Both sides can now **SEND**, **RECEIVE**.
- ✓ Connection not released until both sides do **CLOSE**.
- ✓ Typically client does it, server acknowledges.

An Example of Socket Programming: An Internet File Server

A ***socket*** is a communications connection point (endpoint) that can add name and address in a network.

- ✓ A **socket** is the endpoint used for connecting to a node.
- ✓ The processes that use a ***socket*** can reside on the same system or different systems on different networks.
- ✓ ***Sockets*** are useful for both stand-alone and network applications.

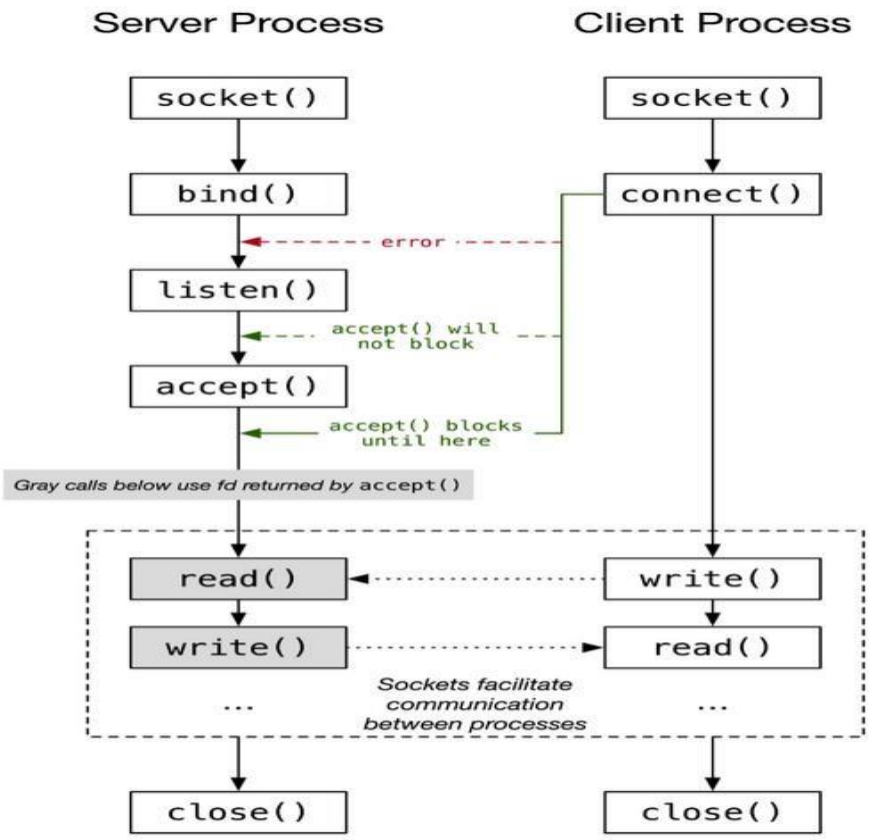
Socket programming is a way of connecting two nodes on a network to communicate with each other.

- ✓ A **node** represents a computer or a physical device with an internet connection.
- ✓ One socket(node) listens on a particular port at an **IP**, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.

Socket programming shows how to use socket APIs to establish communication links between **remote** and **local processes**.

- o The signals required to implement the connection between two nodes are sent and received using the sockets on each node respectively.

- ✓ The nodes are divided into two types, **server node** and **client node**.
- ✓ The client node sends the connection signal and the server node receives the connection signal sent by the client node.
- ✓ The **connection** between a server and client node is **established using the socket** over the transport layer of the internet.
- ✓ After a connection has been established, the client and server nodes can **share information** between them using the read and write commands.
- ✓ After sharing of information is done, the **nodes terminate the connection.**



ELEMENTS OF TRANSPORT PROTOCOLS

This describes various elements of transport protocol in transport layer of computer networks there are six elements of transport protocol namely: **Addressing, Connection establishment connection release, flow control and buffers, multiplexing and crash recovery**.

The ***transport service*** is implemented by a **transport protocol** used between the two transport entities.

- ✓ In some ways, transport protocols resemble the data link protocols.
- ✓ Both have to deal with **error control**, **sequencing**, and **flow control**, among other issues.

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.
In **data link layer** over **point-to-point** links such as wires or optical fiber, it is usually not necessary for a router to specify which router it wants to talk to — each outgoing line leads directly to a particular router.

In the **transport layer**, explicit addressing of estimations is required.

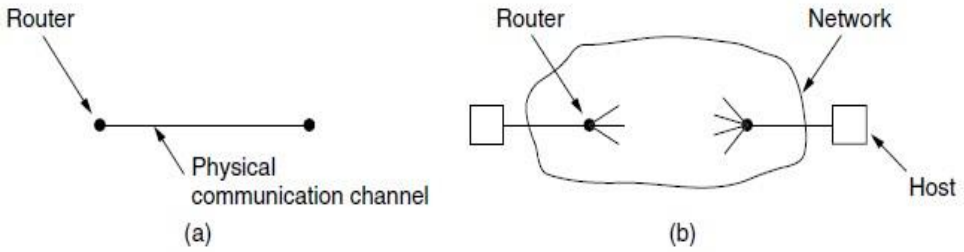
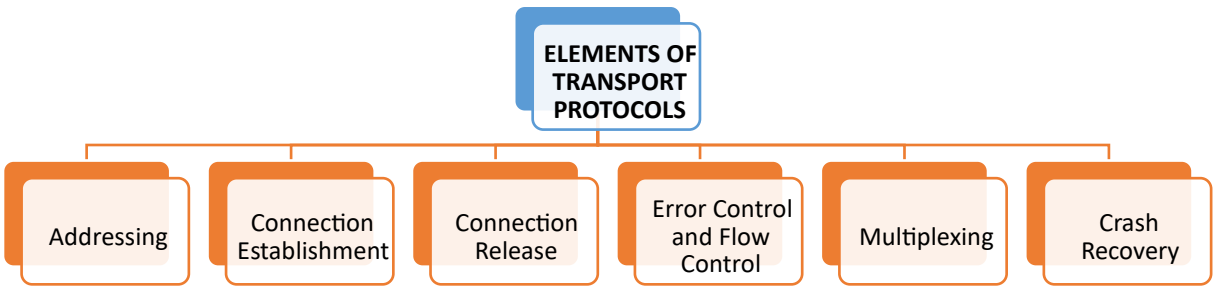


Figure 6-7. (a) Environment of the data link layer. (b) Environment of the transport layer.



Addressing

Addressing is the ability to communicate with the correct application on the computer. Addressing typically uses network ports to assign each sending and receiving application a specific port number on the machine. By combining the IP address used in the network layer and the port on the transport layer, each application can have a unique address.

- ✓ TransportLayer deals with **addressing** or **labelling** a frame.
- ✓ It also differentiates between a connection and a transaction.
- ✓ **Connection identifiers** are ports or sockets that label each frame, so the receiving device knows which process it has been sent from. This helps in keeping track of multiple-message conversations.
- ✓ Ports or sockets address multiple conservations in the same location.
- ✓ **Addressing** typically uses network ports to assign each sending and receiving application a specific port number on the machine.
- ✓ By combining the IP address used in the network layer and the port on the transport layer, each application can have a unique address.

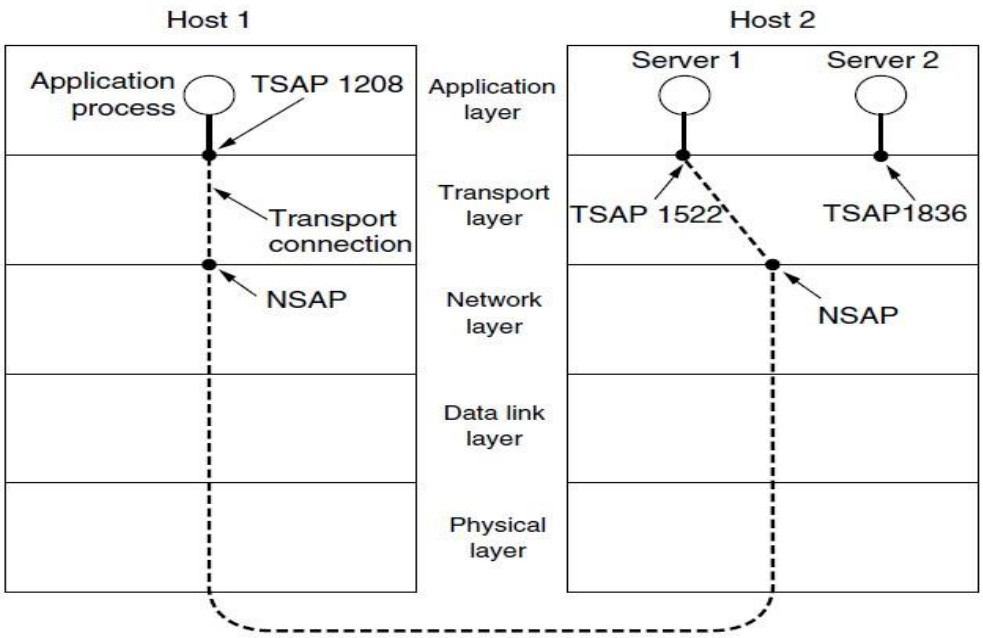
To define **transport addresses** to which processes can listen for connection requests.

In the Internet, these endpoints are called **ports**.

✓The generic term **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.



Connection Establishment

TCP connection establishment is a crucial phase in the lifecycle of a TCP connection, where the client and server establish a secure and robust link before data exchange begins. This process involves a series of steps, protocols, and handshakes that ensure seamless communication. The intricacies of TCP connection establishment, unraveling its significance and shedding light on the key aspects that make it an indispensable component of modern networking.

Establishing a connection sounds easy, but it is actually surprisingly tricky.

- ✓ At **first** glance, one transport entity to just send a **CONNECTION REQUEST** segment to the destination and wait for a **CONNECTION ACCEPTED** reply.
- ✓ The problem occurs when the network can lose, delay, corrupt, and duplicate packets. This behavior causes serious complications.
- ✓ To solve this specific problem, **Tomlinson (1975**) introduced the **three-way handshake**.
- ✓ This establishment protocol involves one peer checking with the other that the connection request is indeed current.

Three-way handshake

The 3-Way handshake is a TCP/IP network connection mechanism that connects the server and client. Before the real data communication process begins, both the client and server must exchange synchronization and acknowledgment packets.

The 3-way handshake mechanism is designed to allow both communicating ends to initiate and negotiate the network TCP socket connection parameters at the same time before data is transmitted. It allows you to transfer numerous TCP socket connections in both directions simultaneously.

TCP uses **three-way handshake** to establish connections.

- ✓ Within a connection, a timestamp is used to extend the 32-bit sequence number so that it will not wrap within the maximum packet lifetime, even for gigabit-per-second connections.
- ✓ This mechanism is a fix to TCP that was needed as it was used on faster and faster links.

✓ It is described in **RFC 1323** and called **PAWS** (**Protection Against Wrapped Sequence numbers**).

Three Way Handshake is a process used for establishing a TCP connection.

✓ Transmission Control Protocol (TCP) provides a secure and reliable connection between two devices using the ***3-way handshake*** process.

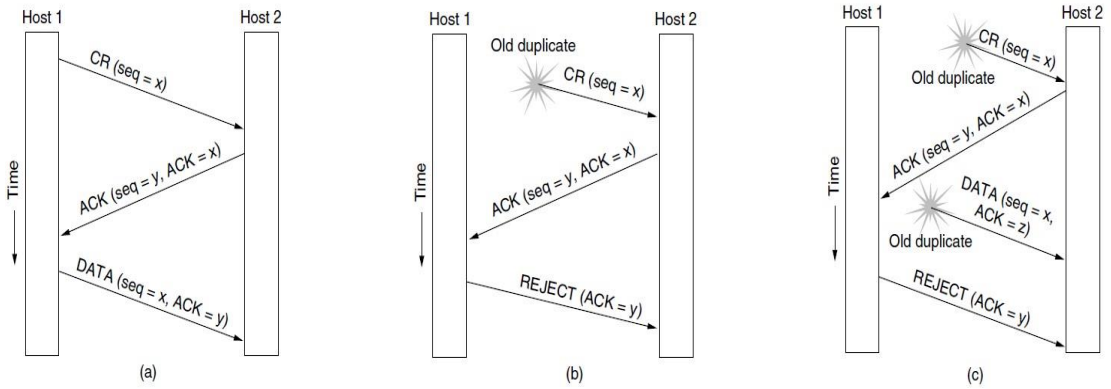


Figure 6-11. Three protocol scenarios for establishing a connection using a **three-way handshake**. CR denotes CONNECTION REQUEST. (a) Normal operation. (b) Old duplicate CONNECTION REQUEST appearing out of nowhere. (c) Duplicate CONNECTION REQUEST and duplicate ACK.

Connection Release

There are two styles of terminating a connection: asymmetric release and symmetric release

- ✓ **Asymmetric release** is the way the telephone system works: when one party hangs up, the connection is broken.
- ✓ **Symmetric release** treats the connection as two separate unidirectional connections and requires each one to be released separately.

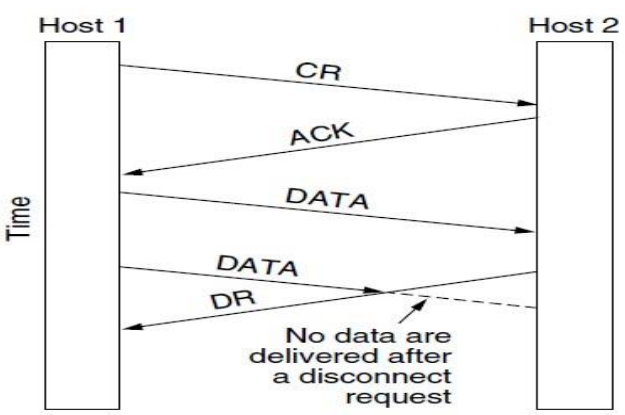


Figure 6-12. Abrupt disconnection with loss of data.

There is a famous problem that illustrates this issue. It is called the **two-army problem**.

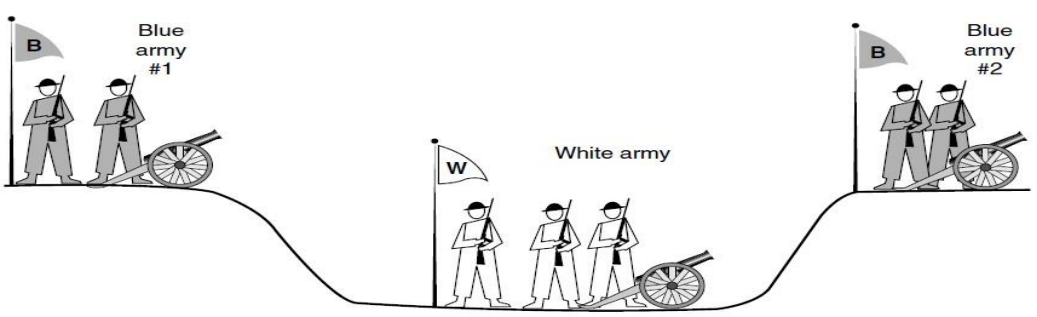


Figure 6-13. The two-army problem.

Error Control and Flow Control

Error Control is performed **end to end** in this layer to ensure that the complete message arrives at the receiving transport layer without any error. Error Correction is done through retransmission. network layer. Ensure correct delivery of data with efficiency.

Error Control: is performed end to end in this layer to ensure that the complete message arrives at the receiving transport layer without any error.

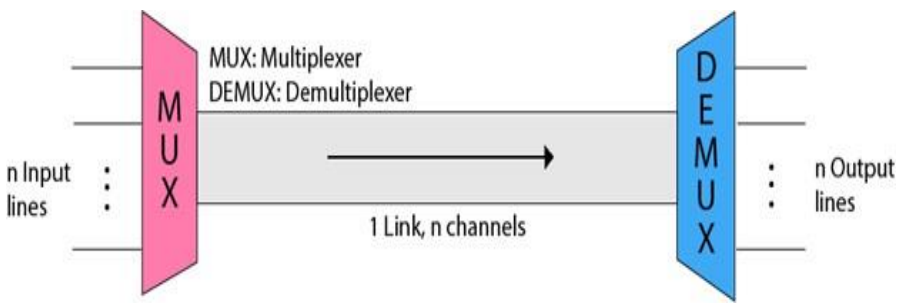
- ✓ Error Correction is done through retransmission.
- ✓ **Error control** is ensuring that the data is delivered with the desired level of reliability, usually that all of the data is delivered without any errors.
- ✓ **Flow Control:** In this layer, flow control is performed end to end.
- ✓ **Flow control** is keeping a fast transmitter from overrunning a slow receiver.

Multiplexing

Gathering data from **multiple application processes** of the sender, enveloping that data with a header, and sending them as a whole to the intended receiver is called **multiplexing**. Delivering received segments at the receiver side to the correct app layer processes is called demultiplexing.

Multiplexing involves **combining multiple data streams into a single transmission channel**.

- ✓ **Multiplexing** is a technique used to combine and send the multiple data streams over a single medium.
- ✓ The process of combining the data streams is known as multiplexing and **hardware** used for multiplexing is known as a **multiplexer**.



Advantages of Multiplexing:

- ✓ More than one signal can be sent over a single medium.
- ✓ The bandwidth of a medium can be utilized effectively.

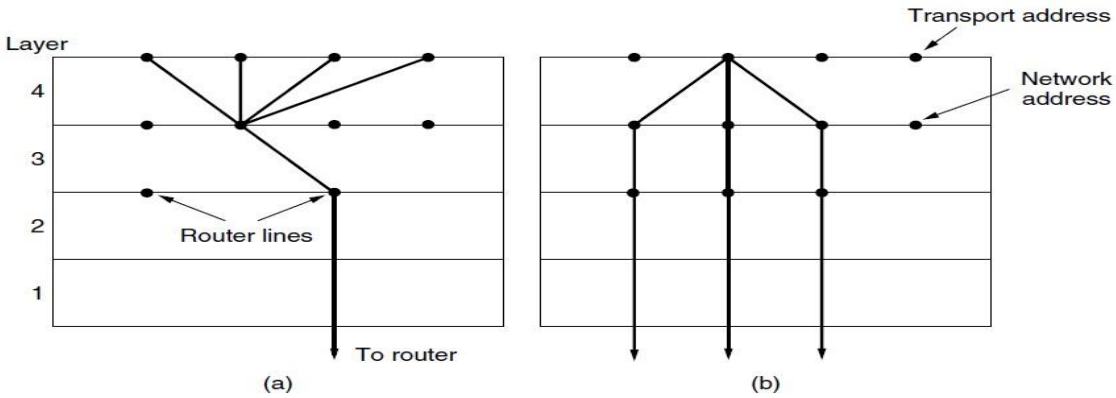


Figure 6-17. (a) Multiplexing. (b) Inverse multiplexing.

Crash Recovery

- ✓ If **hosts** and **routers** are subject to crashes or connections are long-lived (e.g., large software or media downloads), recovery from these crashes becomes an issue.
- ✓ If the **transport entity** is entirely within the hosts, **recovery** from network and router crashes is straightforward. • The **transport entities** expect lost segments all the time and know how to cope with them by using retransmissions

TCP is very reliable protocol.

- ✓ It provides sequence number to each of byte sent in segment.
- ✓ It provides the feedback mechanism i.e. when a host receives a packet, it is bound to ACK that packet having the next sequence number expected (if it is not the last segment).
- ✓ When a TCP Server crashes mid-way communication and re-starts its process it sends **TPDU** broadcast to all its hosts.
- ✓ The **hosts** can then send the last data segment which was never unacknowledged and carry onwards.

TCP and UDP protocols

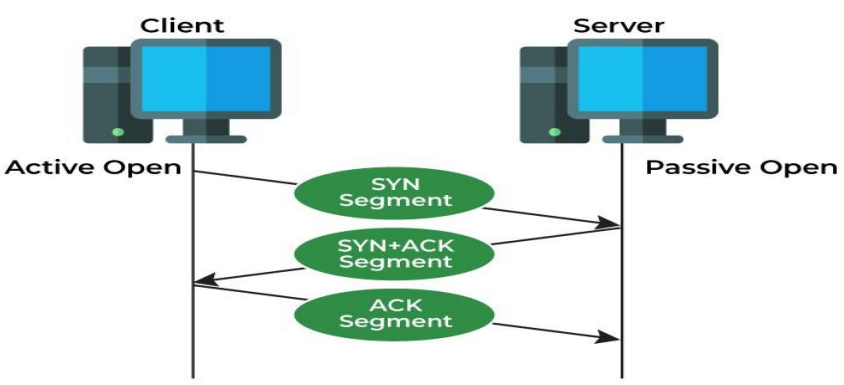
Transmission Control Protocol (TCP) and **User Datagram Protocol** (UDP) both are protocols of the Transport Layer.

TCP is a connection-oriented protocol where as UDP is a part of the Internet Protocol suite, referred to as the UDP/IP suite. UDP is an unreliable and connectionless protocol. They serve as the foundation for communication between applications over a network, but they differ in their features and use cases.

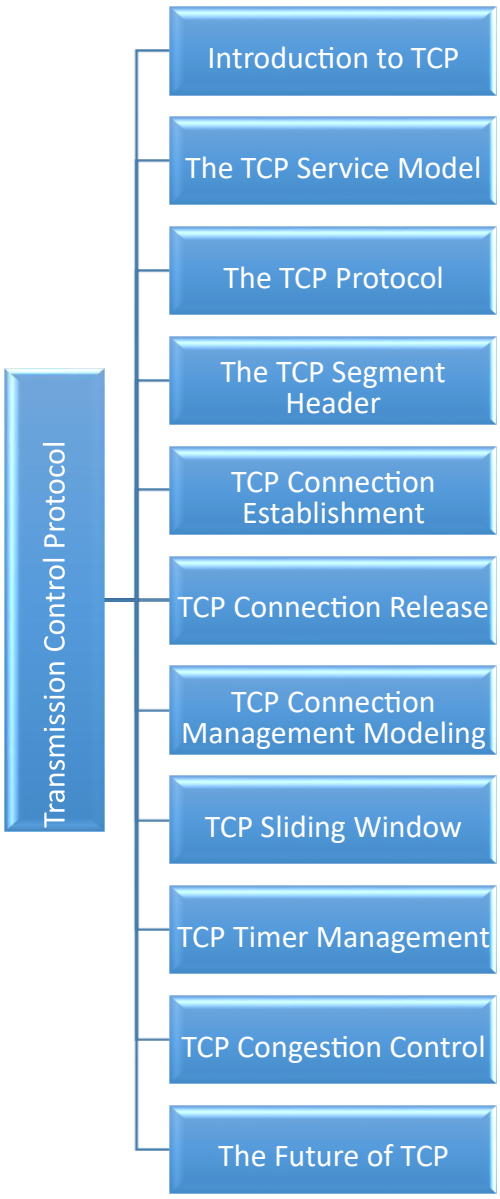
Transmission Control Protocol (TCP)

TCP (Transmission Control Protocol) is one of the main protocols of the Internet protocol suite. It lies between the Application and Network Layers which are used in providing reliable delivery services. It is a connection-oriented protocol for communications that helps in the exchange of messages between different devices over a network. The **Internet Protocol** (IP), which establishes the technique for sending data packets between computers, works with TCP.

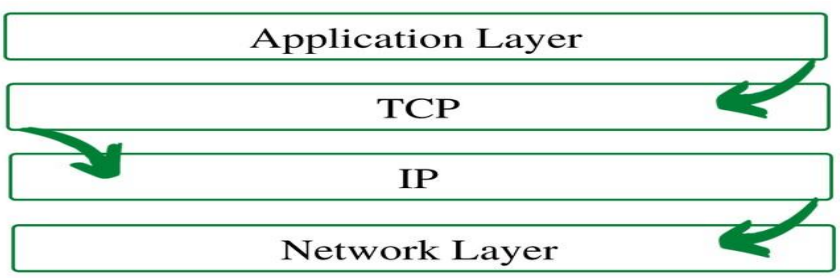
- ✦ **TCP** (**Transmission Control Protocol**) was specifically designed to provide a reliable **end-to-end** byte stream over an unreliable internetwork.
- ✦ TCP meaning Transmission Control Protocol, is a communications standard for delivering data and messages through networks. TCP is a basic standard that defines the rules of the internet and is a common protocol used to deliver data in digital network communications.



An internetwork differs from a single network because different parts may have wildly different topologies, **bandwidths**, **delays**, **packet sizes**, and other **parameters**.



Introduction to TCP (Transmission Control Protocol) is one of the main protocols of the Internet protocol suite. It lies between the Application and Network Layers which are used in providing reliable delivery services. It is a connection-oriented protocol for communications that helps in the exchange of messages between different devices over a network. The Internet Protocol (IP), which establishes the technique for sending data packets between computers, works with TCP



TCP Service Model

TCP provides several essential services to ensure reliable and ordered communication between applications over a network.

Both the sender and the receiver creating end-points, called sockets, obtain TCP service, \Each socket has a socket number (address) consisting of the IP address of the host and a 16-bit number local to that host, called a **port**. A port is the TCP name for a TSAP. For TCP service to be obtained, a connection must be explicitly established between a socket on one machine and a socket on another machine.

Port	Protocol	Use
20, 21	FTP	File transfer
22	SSH	Remote login, replacement for Telnet
25	SMTP	Email
80	HTTP	World Wide Web
110	POP-3	Remote email access
143	IMAP	Remote email access
443	HTTPS	Secure Web (HTTP over SSL/TLS)
543	RTSP	Media player control
631	IPP	Printer sharing

Figure 6-34. Some assigned ports.

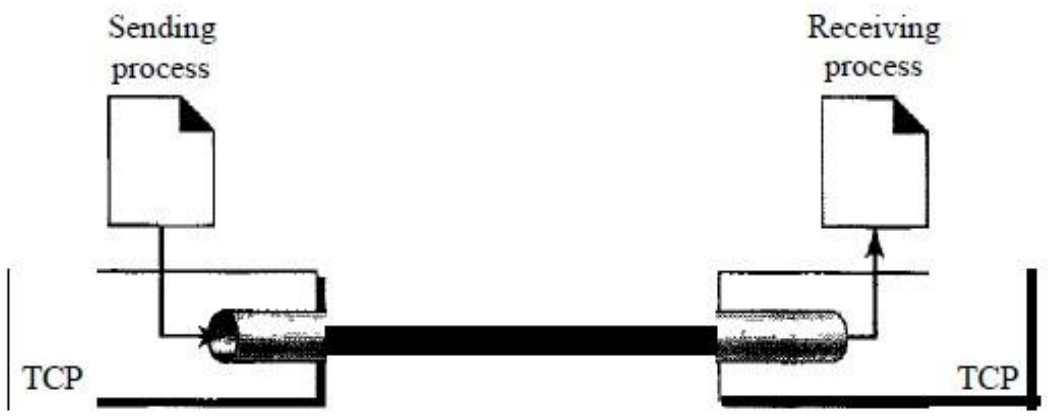
The various **services** provided by the TCP to the application layer:-

Process-to-Process Communication

- ✓ Process-to-process communication: To exchange of data and information between two applications or processes running on different devices within a network.
- ✓ The communication between these processes is facilitated by various protocols and mechanisms.
- ✓ TCP provides process-to-process communication using port numbers.

Connection-Oriented Service

- ✓ A **connection-oriented service** is a type of communication service provided by a network protocol, where a dedicated and reliable connection is established between two devices or processes before data transfer begins.
- ✓ This service ensures that data is delivered accurately, in order, and without errors.
- ✓ Connection-oriented services are commonly associated with protocols like Transmission Control Protocol (TCP) in the TCP/IP protocol suite.
    - ✦ The two TCPs establish a connection between them.
    - ✦ Data are exchanged in both directions.
    - ✦ The connection is terminated.



Stream Delivery Service

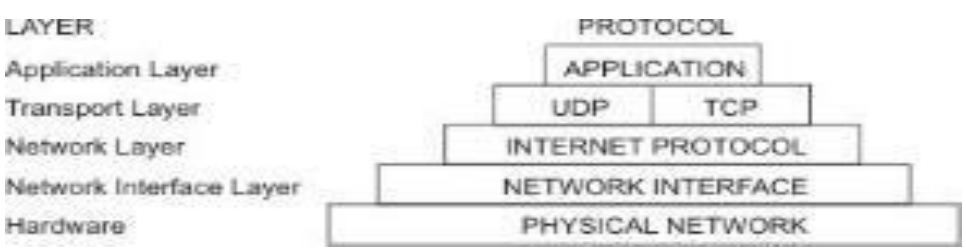
- ✓ A **stream delivery service** involves transmitting data as a continuous stream without explicit message boundaries.
- ✓ It is suitable for real-time applications like video and audio streaming, providing a constant and uninterrupted flow of data.
- ✓ TCP is a common protocol for stream delivery, ensuring reliable and ordered communication.

Reliable Service

- ✓ A **reliable service** in networking ensures accurate, ordered, and error-free data delivery between devices.
- ✓ It includes acknowledgment, retransmission, error detection, and flow control mechanisms. TCP is a common example of a reliable protocol, providing guarantees for data integrity and order.
- ✓ TCP is a reliable transport protocol. It uses an acknowledgment mechanism to check the safe and sound arrival of data.

The TCP Protocol

Protocols are sets of rules for message formats and procedures that allow machines and application programs to exchange information. Each machine involved in the communication in order for the receiving host to be able to understand the message must follow these rules. The TCP/IP *suite* of protocols can be understood in terms of layers (or levels). The **TCP/IP** protocol from the top: Application Layer, Transport Layer, Network Layer, Network Interface Layer, and Hardware. Transport Layer Protocols, either the **User Datagram Protocol** (**UDP**) or the **Transmission Control Protocol** (**TCP**).



TCP Segment Header

Each TCP segment contains a header and data. The TCP header contains many more fields than the UDP header and can range in size from to bytes, depending on the size of the options field. The TCP header shares some fields with the UDP header: source port number, destination port number, and checksum.

A packet in TCP is called a segment.

- ✓ The segment consists of a 20- to 60-byte header, followed by data from the application program. The header is 20 bytes if there are no options and up to 60 bytes if it contains options.
- ✓ 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

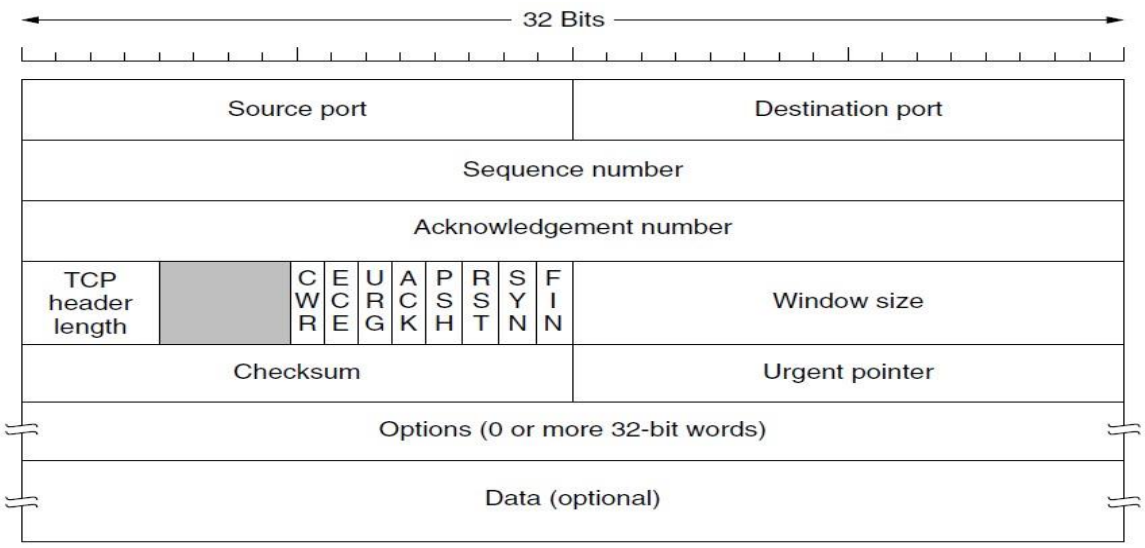


Figure 6-36. The TCP header.

TCP Segment Header

i. **Source port:** It defines the port of the application, which is sending the data. So, this field contains the source port address, which is 16 bits.

ii. **Destination port:** It defines the port of the application on the receiving side. So, this field contains the destination port address, which is 16 bits.

iii. **Sequence number:** This field contains the sequence number of data bytes in a particular session.

iv. **Acknowledgment number**: When the ACK flag is set, then this contains the next sequence number of the data byte and works as an acknowledgment for the previous data received. For example, if the receiver receives the segment number 'x', then it responds 'x+1' as an acknowledgment number.

v. **HLEN:** It specifies the length of the header indicated by the 4-byte words in the header. The size of the header lies between 20 and 60 bytes. Therefore, the value of this field would lie between 5 and 15.

vi. **Reserved:** It is a 4-bit field reserved for future use, and by default, all are set to zero.

vii. **Flags**

There are six control bits or flags:

a. **URG:** It represents an urgent pointer. If it is set, then the data is processed urgently.
b. **ACK:** If the ACK is set to 0, then it means that the data packet does not contain an acknowledgment.
c. **PSH:** If this field is set, then it requests the receiving device to push the data to the receiving application without buffering it.
d. **RST:** If it is set, then it requests to restart a connection.
e. **SYN:** It is used to establish a connection between the hosts.
f. **FIN:** It is used to release a connection, and no further data exchange will happen.

viii. **Window size :** It is a 16-bit field. It contains the size of data that the receiver can accept. This field is used for the flow control between the sender and receiver and also determines the amount of buffer allocated by the receiver for a segment. The value of this field is determined by the receiver.

ix. **Checksum :** It is a 16-bit field. This field is optional in UDP, but in the case of TCP/IP, this field is mandatory.

x. **Urgent pointer :** It is a pointer that points to the urgent data byte if the URG flag is set to 1. It defines a value that will be added to the sequence number to get the sequence number of the last urgent byte.

xi. **Options :** It provides additional options. The optional field is represented in 32-bits. If this field contains the data less than 32-bit, then padding is required to obtain the remaining bits.

TCP Connection Establishment

Connection establishment is performed by using the **three-way handshake** mechanism.

- ✓ A three-way handshake synchronizes both ends of a network by enabling both sides to agree upon original sequence numbers.
- ✓ To establish a connection, one side, say, the server, passively waits for an incoming connection by executing the LISTEN and ACCEPT primitives in that order, 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.

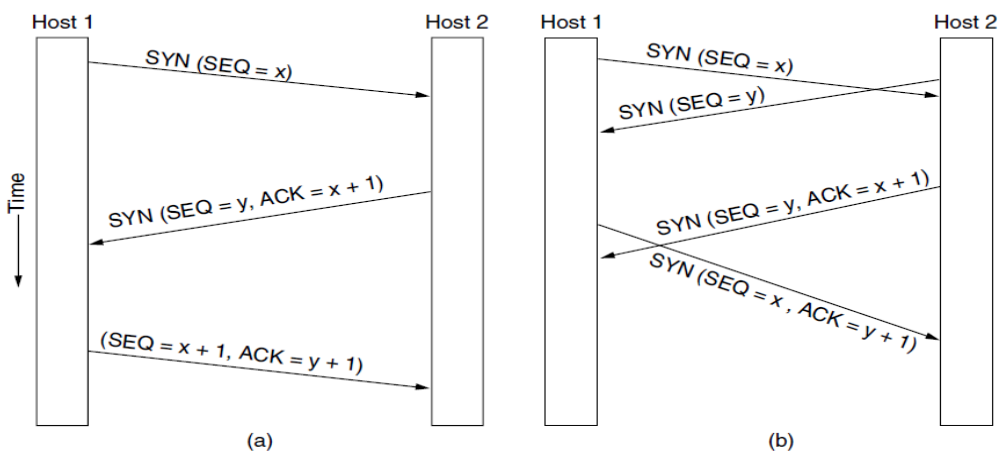


Figure 6-37. (a) TCP connection establishment in the normal case. (b) Simultaneous connection establishment on both sides.

TCP Connection Release

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.
- ✓ When both directions have been shut down, the connection is released.

TCP Connection Management Modeling

The steps required to establish and release connections can be represented in a finite state machine with the 11 states.

- ✓ 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.
- ✓ Each connection starts in the CLOSED state. It leaves that state when it does either a passive open (LISTEN) or an active open (CONNECT).

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
TIME 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 6-38. The states used in the TCP connection management finite state machine.

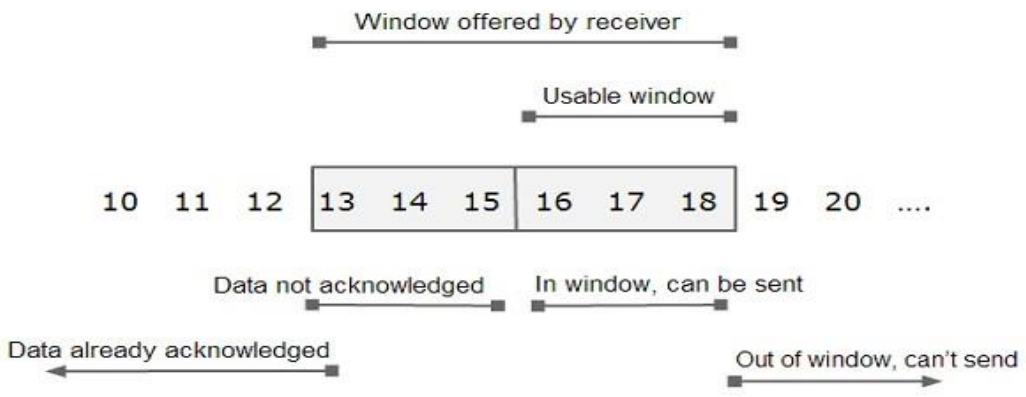
TCP Sliding Window

The TCP sliding window determines the number of unacknowledged bytes, x, that one system can send to another. Two factors determine the value of x: The size of the send buffer on the sending system. The size and available space in the receive buffer on the receiving system.

The sliding window is a technique that allows TCP to adjust the amount of data that can be sent or received at any given time.

> A sliding window is used to make transmission more efficient as weD as to control the flow of data so that the destination does not become overwhelmed with data. TCP sliding windows are byte-oriented.

- ✓ The sliding window is a variable-sized buffer that represents the available space in the sender's or the receiver's end of the connection.
- ✓ The sender can only send data that fits within the window, and the receiver can only accept data that fits within the window.
- ✓ The window size can change depending on the network congestion, the available bandwidth, and the feedback from the other end of the connection



Some points about TCP sliding windows:

- o The size of the window is the lesser of *rwnd* and *cwnd.* o The source does not have to send a full window's worth of data.
- ✓ The window can be opened or closed by the receiver, but should not be shrunk.
- ✓ The destination can send an acknowledgment at any time as long as it does not result in a shrinking window.

✓ The receiver can temporarily shut down the window; the sender, however, can always send a segment of 1 byte after the window is shut down.

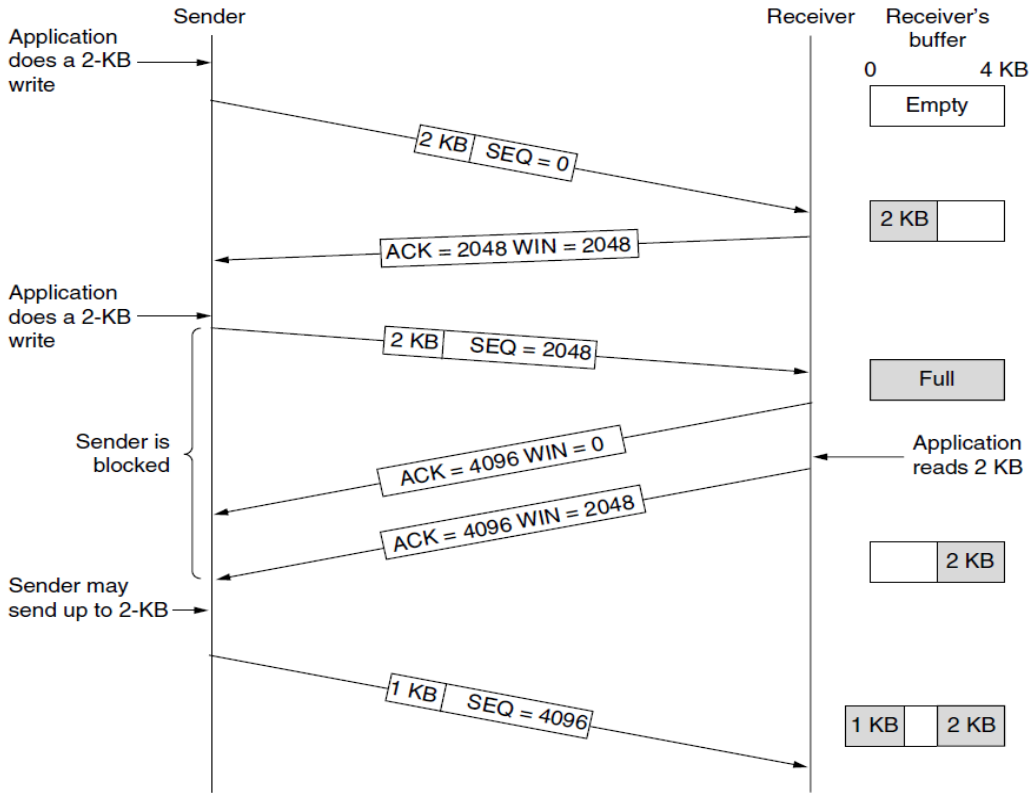


Figure 6-40. Window management in TCP.

TCP Features

Numbering System

There are two fields in TCP mainly **sequence number** and **acknowledgment number**.These two fields in the TCP mainly refers to Byte Number.

Byte Number

The bytes of data that are being transferred in each connection are numbered by TCP.

- ✓ The numbering mainly starts with a randomly generated number.
- ✓ TCP mainly numbers all the data bytes that are transmitted in a connection.

Sequence Number

After the numbering of bytes, the TCP makes the grouping of bytes in the form of "**segments**".

- ✓ A sequence is assigned to each segment that is being sent.
- ✓ The Sequence number of each segment is the number of the first byte that is carried in that segment.
- ✓ Thus the value in the sequence number field of the segment mainly defines the number of the first data byte that is contained in that segment.

The bytes of data being transferred in each connection are numbered by TCP.
The numbering starts with a randomly generated number.

Flow Control

The TCP provides the facility of Flow control. With the help of TCP, the receiver of the data control the amount of the data that are to be sent by the sender. The flow control is mainly done in order to prevent the receiver from being overwhelmed with the data. The numbering system also allows the TCP to use byte-oriented flow control.

TCP uses a sliding window to handle flow control. The sliding window protocol used by TCP, however, is something between the ***Go-Back-N*** and Selective Repeat sliding window.

- ✓ The window is *opened, closed,* or *shrunk.* These three activities, are in the control of the receiver (and depend on congestion in the network), not the sender.
- ✓ The sender must obey the commands of the receiver in this matter.
- ✓ Opening a window means moving the right wall to the right.
- ✓ Closing the window means moving the left wall to the right.
- ✓ Sluinking the window means moving the right wall to the left.

Error Control

As TCP provides reliable services, thus it implements an error control mechanism for this purpose. The Error control though considers the segment as the unit of data for error detection. Error control is byte-oriented in nature.

- ✓ TCP is a reliable transport layer protocol. This means that an application program that delivers a stream of data to TCP relies on TCP to deliver the entire stream to the application program on the other end in order, without error, and without any part lost or duplicated.
- ✓ TCP provides reliability using error control. Error control includes mechanisms for detecting corrupted segments, lost segments, out-of-order segments, and duplicated segments. Error control also includes a mechanism for correcting errors after they are detected. Error detection and correction in TCP is achieved through the use of three simple tools: checksum, acknowledgment, and time-out.

USER DATAGRAM PROTOCOL (UDP)

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.

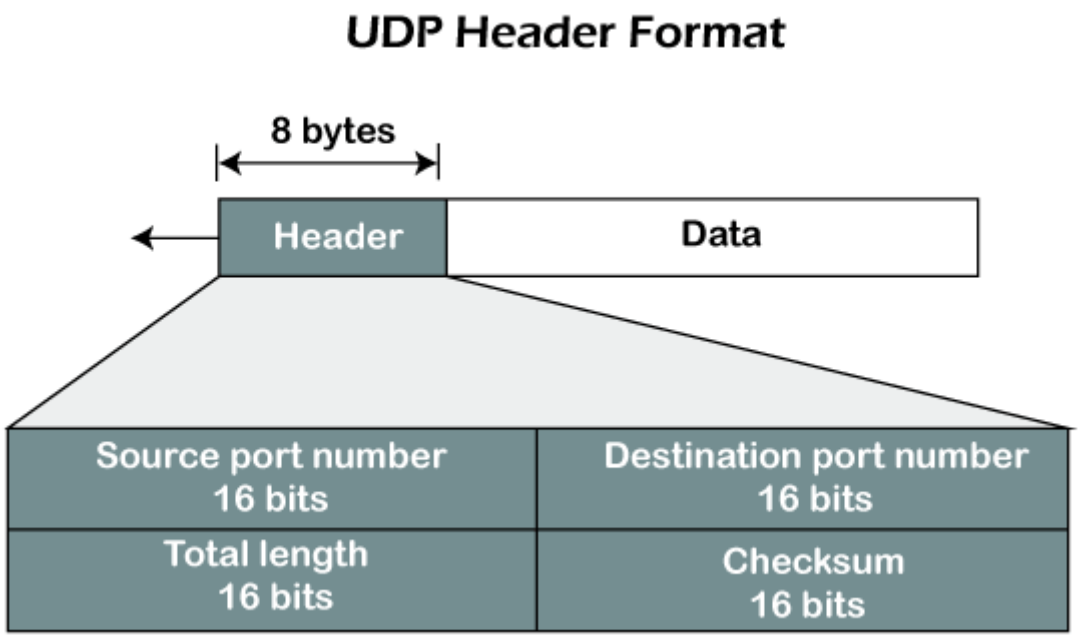
The Internet protocol suite supports a connectionless transport protocol called **UDP** (**User Datagram Protocol**). UDP provides a way for applications to send encapsulated IP datagrams without having to establish a connection.

The **User Datagram Protocol**, or UDP, is a communication protocol used across the Internet for especially time-sensitive transmissions such as video playback or DNS lookups. It speeds up communications by not formally establishing a connection before data is transferred.

The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to process communication instead of host-to-host communication. If a process wants to send a small message and does not care much about reliability, it can use UDP. Sending a small message by using UDP takes much less interaction between the sender and receiver than using TCP or SCTP.

UDP Header

In UDP, the header size is 8 bytes, and the packet size is upto 65,535 bytes. But this packet size is not possible as the data needs to be encapsulated in the IP datagram, and an IP packet, the header size can be 20 bytes; therefore, the maximum of UDP would be 65,535 minus 20. The size of the data that the UDP packet can carry would be 65,535 minus 28 as 8 bytes for the header of the UDP packet and 20 bytes for IP header.



UDP packets, called user datagrams, have a fixed-size header of 8 bytes. Figure 6-27 shows the format of a user datagram.

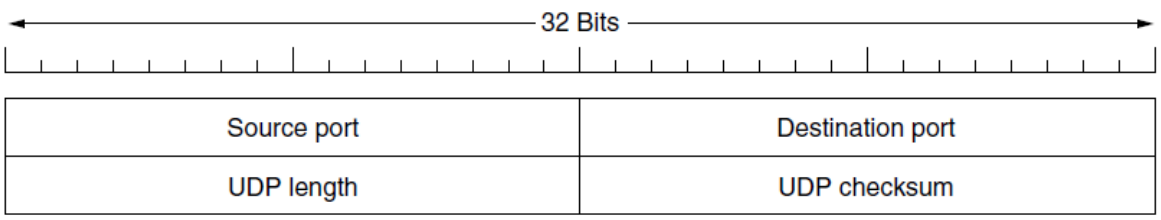


Figure 6-27. The UDP header.

$$\text{UDP length} = \text{IP length} - \text{IP header's length}$$

The fields in a UDP header are:

- **Source port** – The port of the device sending the data. This field can be set to zero if the destination computer doesn't need to reply to the sender.
- **Destination port** – The port of the device receiving the data. UDP port numbers can be between 0 and 65,535.
- **UDP Length** – Specifies the number of bytes comprising the UDP header and the UDP payload data. The limit for the UDP length field is determined by the underlying IP protocol used to transmit the data.
- **UDP Checksum** – The checksum allows the receiving device to verify the integrity of the packet header and payload. It is optional in IPv4 but was made mandatory in IPv6.

UDP Operation

UDP uses concepts common to the transport layer. These concepts will be discussed here briefly, and then expanded in the next section on the TCP protocol.

Connectionless Services

UDP provides a connectionless service. This means that each user datagram sent by UDP is an independent datagram. There is no relationship between the different user datagrams even if they are coming from the same source process and going to the same destination program. The user datagrams are not numbered. Also, there is no connection establishment and no connection termination, as is the case for TCP. This means that each user datagram can travel on a different path.

Flow and Error Control

UDP is a very simple, unreliable transport protocol. There is no flow control and hence no window mechanism. The receiver may overflow with incoming messages. There is no error control mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded.

Use of UDP

The following lists some uses of the UDP protocol:

- ✓ UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control. It is not usually used for a process such as FrP that needs to send bulk data.
- ✓ UDP is suitable for a process with internal flow and error control mechanisms. For example, the Trivial File Transfer Protocol (TFTP) process includes flow and error control. It can easily use UDP.
- ✓ UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but not in the TCP software.
- ✓ UDP is used for management processes such as SNMP.
- ✓ UDP is used for some route updating protocols such as Routing Information Protocol (RIP).

Compare TCP and UDP

Parameter	TCP	UDP
Name	Transmission Control Protocol	User Datagram Protocol or Universal Datagram Protocol
Connection	TCP is a connection-oriented protocol.	UDP is a connectionless protocol.
Usage	TCP is suited for applications that require high reliability, and transmission time is relatively less critical.	UDP is suitable for applications that need fast, efficient transmission, such as games. UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients.
Use by other protocols	HTTP, HTTPs, FTP, SMTP, Telnet	DNS, DHCP, TFTP, SNMP, RIP, VOIP.
Ordering of data packets	TCP rearranges data packets in the order specified.	UDP has no inherent order as all packets are independent of each other. If ordering is required, it has to be managed by the

		application layer.
Speed of transfer	The speed for TCP is slower than UDP.	UDP is faster because there is no error-checking for packets.
Header Size	TCP header size is 20 bytes	UDP Header size is 8 bytes.
Common Header Fields	Source port, Destination port, Check Sum	Source port, Destination port, Check Sum
Error Checking	TCP does error checking	UDP does error checking, but no recovery options.
Data Flow Control	TCP does Flow Control. TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.	UDP does not have an option for flow control
Reliability	There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent.	There is no guarantee that the messages or packets sent would reach at all.