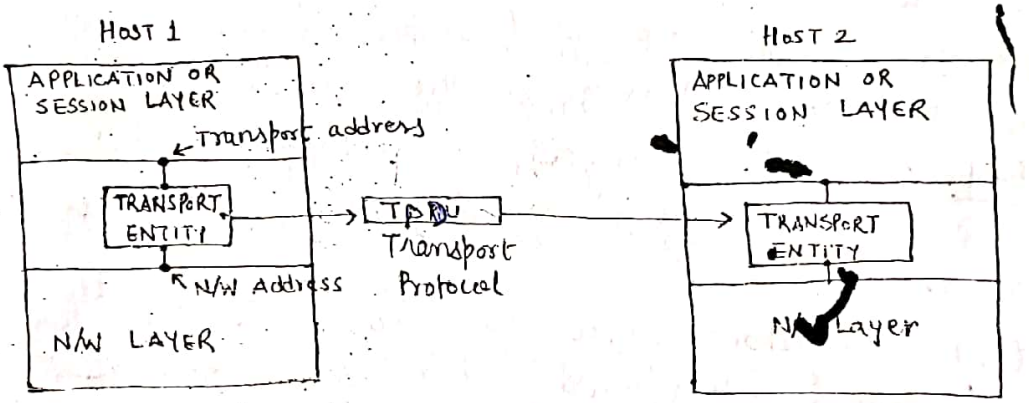
UNIT-3
Transport Layer

Transport Layer Services :- To achieve the goals of Transport layer services like efficiency, Reliability & cost effectiveness, transport layer makes use of the services provided by Network layer. The hardware/software that works in Transport layer is called Transport entity. The transport entity can be located in OS kernel, in separate user process, in library package or on Network Interface card (NIC). The relationship between Network, transport and application layer is illustrated in fig. below -



Transport Layer services are implemented by a transport protocol used b/w two transport entities. The services by Transport services are -

Addressing :- Different - different hosts runs different - different programs so transport layer header must therefore include a type of address called service point address or Port address which ensures that each packet approach to correct host and program running over it.

Segmentation & Reassembly :- message is divided into transmittable segments at the source machine, where each segment contains a sequence number. These sequence numbers provide a help to reassemble the segments at receiving side. Even this helps to find the lost packets.

Connection Control - Transport layer service provide two types (2) of connections -

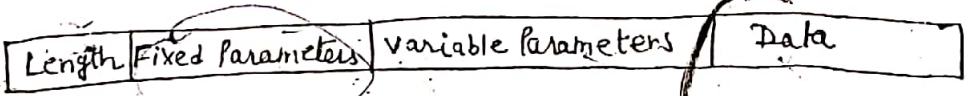
a) Connection-oriented service - In this a connection is established by handshake process means a path get made between source and destination. over this path, each packet is delivered to the transport layer of the destination machine. After message transmission the connection can be released. TCP is the transport layer protocol which provides connection oriented service in the internet.

b) Connectionless service : A connectionless transport protocol will treat each packet independently bcoz there is no connection b/w source & destination. Each packet can follow different route. UDP is transport layer protocol which provides connectionless service in the internet.

Flow Control :- Transport Layer is responsible for flow control. Flow control is performed end to end, rather then across a single link.

Error Control :- Transport layer is also responsible for error control. Error control is performed end to end. The sending transport layer makes sure that entire message get transmit without any damage, loss or duplication. If any error get arise, then error correcting methodologies gets taken.

Transport Protocol Data Unit (TPDU) :- ~~TPDP~~ TPDU is a kind of message that the protocols in the transport layer of the OSI model use for communication, which is then sent to Network layer. The format of TPDU is given below -



(i) Length : Length field indicates the total number of bytes in TPDU. It's of 1 byte.

(ii) Fixed Parameters :- fixed parameters contain certain factors like Code, Source reference, Destination reference, sequence number & credit allocation.

- code identifies the type of data unit, following codes are recognized by ISO.

CR - Connection Request	CC - Connection Confirm
DR - Disconnect Request	DC - Disconnect Confirm
DT - Data	ED - Expedited Data
AK - Acknowledgement	EA - Expedited Acknowledgement
RJ - Reject	ER - Error

urce & Destination Reference - The source & destination eference fields contain the addresses of the original sender ultimate destination of packet.

Sequence number - As the transmission is divided into smaller packets for transport, each segment is given a number that identifies its place in the sequence. Sequence numbers are used for acknowledgement, flow control & reordering of packets at the destination.

~~Sequence numbers~~ Credit allocation - Credit allocation enables a receiving station to tell the sender how many more data units may be sent before the sender must wait for an acknowledgement.

Variable parameters - The variable parameters section of a TPDU contain parameters that occur infrequently. These control codes are used mostly for management.

Data - The data section include a regular data or expedited data coming from the upper layers. Expedited data consists of a high priority message that must be handled out of sequence.

RANPORT Service Primitives - To allow users to access the transport service, the transport layer must provide a transport servia interface. Each transport service has its own interface.

There are 5 service primitives to avail the transport services -

LISTEN - The server executes a LISTEN primitive. In this library procedure makes a system call which blocks the server until a client turns up.

CONNECT - When a client wants to talk to the server, it executes a ~~CONNECT~~ CONNECT primitive. By this the client gets block & sends packet to the server. This CONNECT primitive has a packet of CONNECTION REQUEST TPDU sent to the server.

When the server is found to be in LISTEN mode, the CONNECT primitive awakes the server. The server get unblocked and a CONNECTION ACCEPTED TPDU get sent back to the client. When this TPDU arrives at client, the client gets unblocked assuring that connection has been established.

Data can now be exchanged using SEND & RECEIVE primitives. Every data pkt sent will be acknowledged.

→ Acknowledgements are managed by transport entities using network layer protocols.

→ Transport entities will need to worry about timers & retransmissions

When a connection is no longer needed, it must be released to free up tables space within the two transport entities. Disconnection is also of two types -

Asymmetric - Either transport user can issue a DISCONNECT primitive

Symmetric - When one side does a DISCONNECT, that means it has no more data to transfer but it is still willing to accept data from it's partner. In this model, a connection is released when both sides have done a DISCONNECT.

lements of Transport Protocols :-

some ways transport protocols resemble the data link protocols. ut DL layer is different from transport layer.
n the Data Link layer, it is not necessary for a router to specify which router it wants to talk to - each outgoing line uniquely specifies a particular router. In the transport layer, explicit addressing of destination is required.

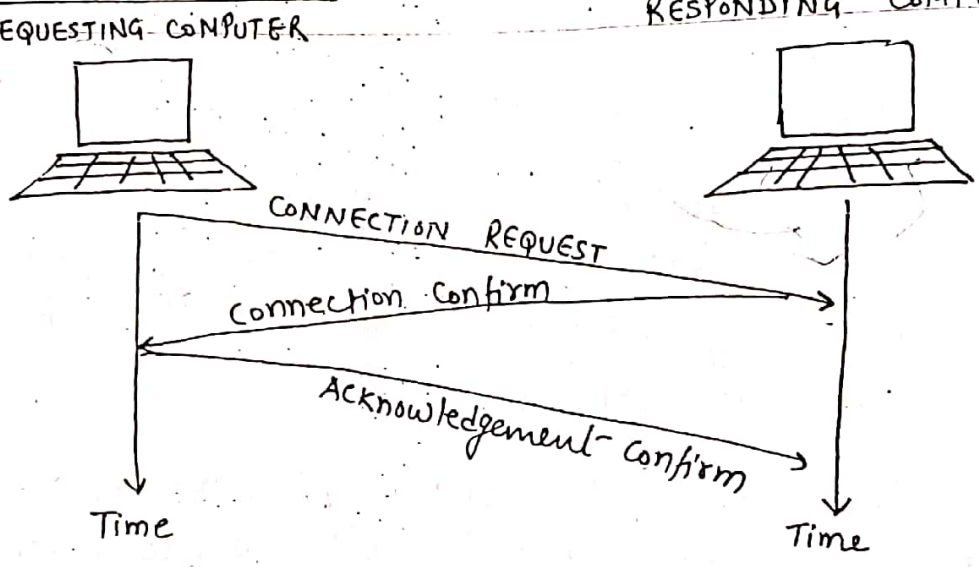
Buffering & flow control are needed in both layers, but the presence of a large & dynamically varying number of connections in the transport layer may require a different approach than we used in the data link layer.

Addressing :- many protocols consider combination of session, presentation & application level protocols into a single package called application. In there cases delivery to the session layer functions is, infact delivery to the application. So communication not just occur between one end machine to other end machine but from one end application to other end application means data generated by one application must be received by correct application on other machine.
In most cases we need communication of multiple applications between two end systems. But how does the Network identify which service access point on one host is communicating with which service access point on other host? To ensure accurate delivery from Service Access Point (SAP) to other SAP, we need another level of addressing

Connection :- To send message from one host to other host, we need to deploy (Implement करना) some path no matter connection oriented or connectionless. Connection Oriented paths are more reliable. Connection oriented transmissions has three stages -

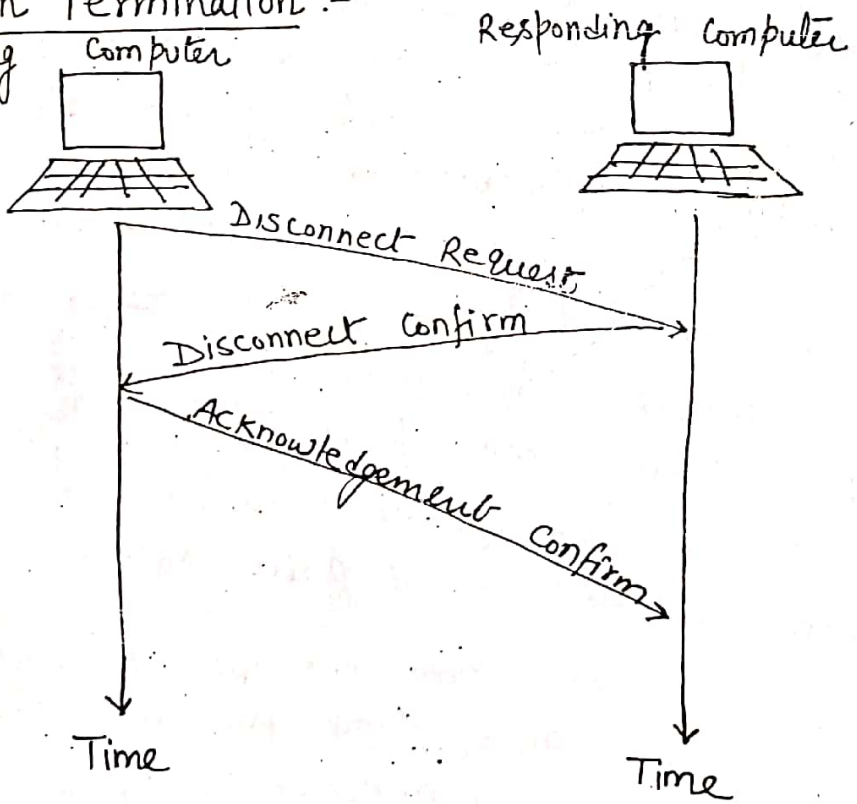
onnection establishment
ata transfer
onnection termination

Connection Establishment :- It's also called Handshake process

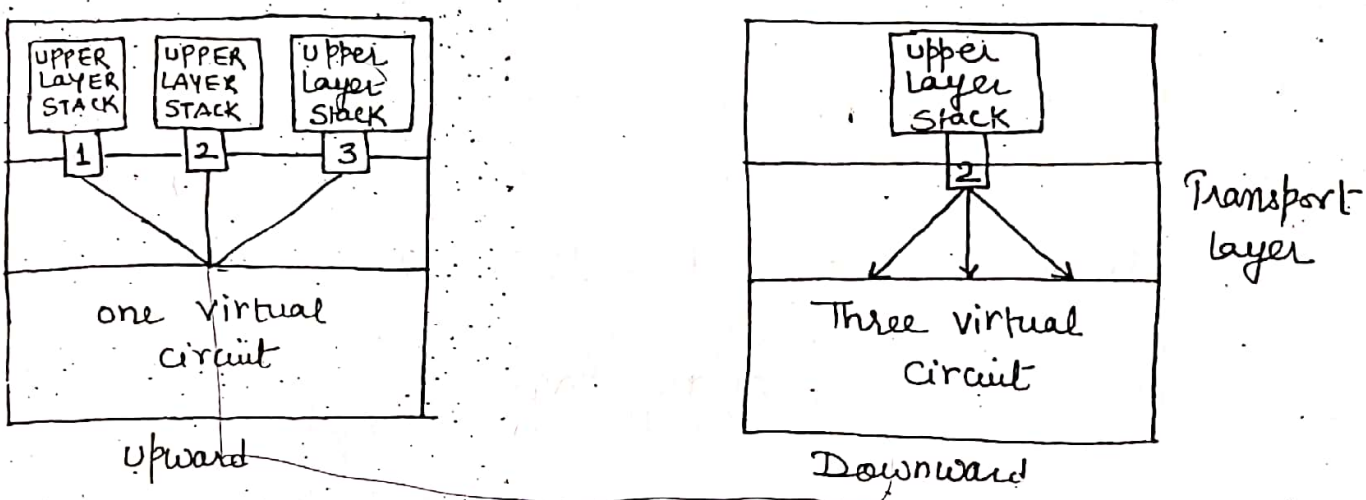


Data Transfer :- It's a simple known phenomena of data transfer.

Connection Termination :-



Multiplexing :- To improve transmission efficiency, the transport layer has the option of the multiplexing. Multiplexing at this layer occur in two ways : upward, meaning that multiple transport layer connections use the same N/W connection or downward, meaning that one transport layer connection uses multiple network connections.



ard Multiplexing :- Sometimes transport layer can send several nsmissions bound for the same destination along the same th by upward multiplexing. This means, if the underlying twork protocol has a high throughput, for example in the nge of 1 Gbps and the user can create data only in the nge of Mbps, then several users can ~~create~~ share on N/W nnection.

nward Multiplexing :- Downward multiplexing allows the transport er to split a single connection among several different paths improve throughput. This option is useful when the nderlying networks have a low or slow capacity.

Crash Recovery

In a case of a router crash, the two ansport entities must exchange information after e crash to determine which TPDUs were received d which were not. The crash can be recovered retransmitting the lost ones.

In the attempt to recover its rious status, the server might send a broadcast PDU to all other hosts, announcing that it had st crashed and requesting that its client nform it of the status of all open connections. ch client can be in one of the two states : e 'TPDU outstanding', S1, or 'no TPDUs outstanding', Based on only this state information, the nt must decide whether to retransmit the st recent TPDU or not. The client should etransmit if & only if it has an unacknowledged PDU outstanding, when it learns of the crash.

User Datagram Protocol (UDP)

The Internet makes two transport otocols available to the applications, UDP and TCP. The er Datagram Protocol (UDP) is a connectionless, unreliable ansport protocol. It does not add anything to the rvices of IP except to provide process-to-process

communication instead of host-to-host communication. Also, it performs very limited error checking. With the disadvantages come some advantages. UDP is a very simple protocol using a minimum of overhead. If a process wants to send a small message and does not care much about reliability, it can use UDP.

Packet Structure:-

UDP provides no guarantees to the upper layer protocol for message delivery and the UDP protocol layer retains no state of UDP message once sent. For this reason, UDP is sometimes referred to as Unreliable Datagram Protocol.

The UDP header consists of 4 fields, each of which is 2 bytes (16 bits). The use of two of those is optional in IPv4 i.e. source port number & UDP checksum. In IPv6, only the source port number is optional.

Source Port Number (16 bits)	Destination Port Number (16 bits)
Length	UDP Checksum
Application Data	

Source Port Number :- It not used, then it should be zero. If the source host is the server, the port number is likely to be a well-known port number.

Destination Port Number :- This field identifies the receiver's port and is required.

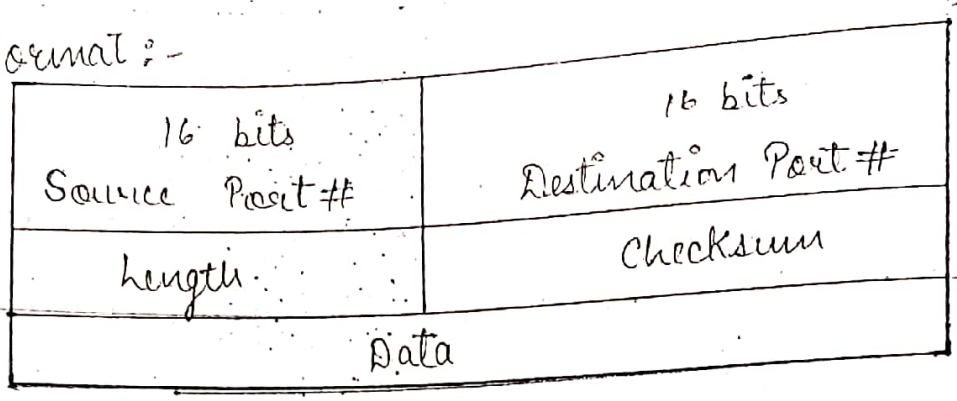
Length :- A field that specifies the length in bytes of the entire datagram: header and data. The minimum length is 8 bytes since that's length of the header.

Checksum :- The checksum field is used for error-checking of the header and data. If no checksum is generated by the transmitter, the field uses the value all-zeros. This field is not optional for IPv6.

UDP Checksum :-

A checksum is an error-detection method in which each transmitted message results in a numerical value based on the value of the bytes in a message. The transmitter places the calculated value in the message and then sends the value with the message. The receiver applies the same formula to each received message and check to make sure the accompanying numerical value is the same.

- UDP format :-



Source port - Source port field is of 16-bit field which specify the address of source.

Destination port - Destination port field is of 16-bit field which specify the address of destination.

Length - Length field specify the length of complete packet by 16-bit.

Checksum - Checksum field is of 16-bits which detects and corrects the located errors.

Data - Data field used to specify the actual data / information to be sent.

DP Operations :-

onnectionless Services :-

UDP provides a connectionless
ervice. This means that each user datagram sent by
DP is an independent datagram. Also, there is no
onnection establishment and no connection termination.
s is the case for TCP. This means that each user
datagram can travel on a different path.

low And Error Control :-

UDP is a very simple,
nreliable transport protocol. There is no flow control
nd hence no window mechanism. The receiver may
verflow with incoming messages. There is no error
ontrol mechanism in UDP except for the checksum.
his 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.

ncapsulation and Decapsulation :-

To send a message
rom one process to another, the UDP protocol encapsulates
nd decapsulates messages in an IP datagram.

Use of UDP :-

. UDP is suitable for a process that requires simple
request-response communication with little concern
for flow and error control.

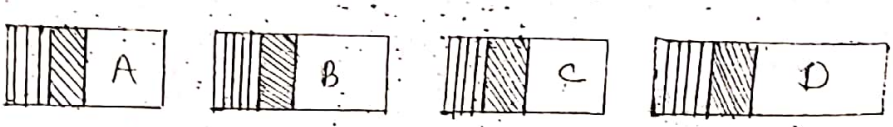
Introduction to TCP :-> (Transmission Control Protocol).

- TCP is said to be connection oriented because before one application can begin to send data each other, the two process must do the "hand shake" with each other, It means they must send some preliminary segments for estabelish the parameter to data transfer.
- TCP protocol runs only in the end system, not in the intermediate network elements. end to end communication
- A TCP provides a full duplex service, It means if there is a TCP connection between process A on one host and process B on another host, then application layer data can flow from process A to process B, at the same time data flows from process B to A
- A TCP entity accepts user data streams from local processes, breaks them into pieces not exceeding 64KB, and send each piece as a separate IP datagram.
- * The IP layer gives no guarantee that datagrams will be delivered properly, so it is up to TCP to time out and retransmit them as need.

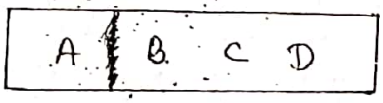
r. TCP Service Protocol :→ TCP service is obtained by both the sender and receiver creating end points, called sockets. Each socket has the socket number, consisting of the IP address of the host and 16 bit number local to the host, called port. port numbers 1024+ are called well known ports. and are reserved for standard services.

port	protocol	Use
21	FTP	File transfer
23	Telnet	Remote login
25	SMTP	E-mail
69	TFTP	Trivial File transfer protocol
79	finger	Lookup information about a user.
80	HTTP	World wide web.
110	POP-3	Remote email access
119	NNTP	USENT news

A TCP connection is a byte stream, not a message stream. message boundaries are not preserved end to end! For example, if the sending process does four 512 bytes write to a TCP stream, these data may be delivered to the receiving process as four 512 bytes chunks, two 1024 byte chunks, one 2048 byte chunk, or some other way.



Four 512 bytes segment sent as separate IP datagrams

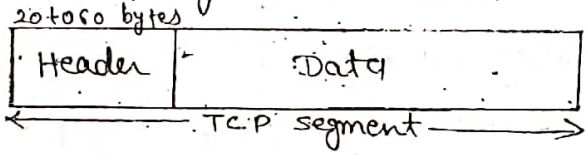


The 2048 bytes of data delivered to the application in a single READ call

TCP Services :→ The primary purpose of TCP is to provide a reliable logical circuit or connection service between pair of process. Reliability is ensured by

(1) connection-oriented service
(2) flow control using sliding window protocol
(3) Error detection using checksum.
(4) Error control using go-back-N ARQ technique.

TCP header & Segment structure :→



source port address (16 bits)								Destination Port address
Sequence number (32 bits)								
Acknowledgement numbers (32 bits)								
TCP header length (4 bits)	Reserved (6 bits)	URG	ACK	PSH	RST	SYN	FIN	Window size (16 bits)
Checksum (16 bits)								urgent pointer (16 bits)
options (0 or more 32 bits word)								
Data (optional)								

The diagram shows the layout of TCP segment. Every segment begins with a fixed format of 20 to 60 bytes, followed by data from the application layer.

1) Source Port Address :⇒ This is 16 bit field that defines the port number of the application program in the host that is sending the segment.

2) Destination Port address :⇒ This is a 16 bit field that defines the port number of the application program in the host that is receiving the segment.

3) Sequence number :⇒ This 32 bit field defines the number assigned to the first byte of data contained in the segment. The sequence number tells the destination. During connection establishment, each party uses a random number generator to create ISN (initial sequence no.)

4) Acknowledgement number :⇒ This 32 bit field defines the byte number that the receiver of the segment is receive from the other party. Acknowledge and data can be piggybacked together.

5) Header length :⇒ This 4 bit field indicates the number of 4 byte words in the TCP header. The length of header can be between 20 to 60 bytes.

(6) control field :⇒ This field defines 6 different control bits of flag.

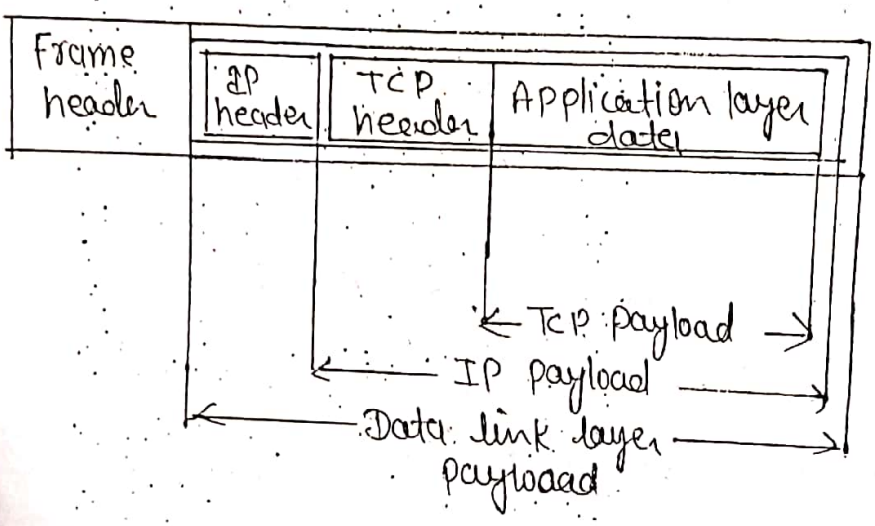
Flag	Description
URG	URG is set to 1 if Urgent Pointer is use. It is used to indicate a byte offset from the current sequence number.
ACK	The value of Acknowledgement field is valid. If ACK = 0, then the segment doesn't contain acknowledgement.
Psh	It indicates push data.
RST	This bit is used to resets the connection that has become confused due to host crash or some other reason.
SYN	The SYN bit is used to establish connection. If SYN=1, ACK=0 then piggybacked Acknowledgement is not in use, if SYN=1 ACK=1 then It bears an acknowledgement.
FIN	It is used to terminate the connection.

Window Size :→ This field defines the size of the window. The length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes.

Checksum :→ A checksum is also provided for extra reliability. The TCP checksum algorithm is simply to add up all the 16 bit world's in one's complement and then take 1's complement of sum.

9) URGENT Pointer :→ This is a 16 bit field, which is valid only if the ~~segment~~ urgent flag is set. It is used when the segment contains urgent data.

* Encapsulation :→



TCP CONNECTION :-

TCP is connection oriented. A connection oriented transport protocol establishes a virtual path between the source and destination. All the segments belonging to a message are then sent over this virtual path. Using a single virtual path for the entire message facilities the acknowledgment process as well as retransmission of damaged or lost frames.

virtual path provides facility for acknowledgement & retransmission

TCP which uses the services of IP, a connectionless protocol, can be connection oriented. The point is that a TCP connection is virtual, not physical.

In TCP, connection-oriented transmission requires three phases

(a) connection establishment.
(b) Data Transfer.
(c) connection termination.

(a) Connection establishment :-

→ TCP transmits the data in full duplex mode, when two TCP in two machines are connected they are able to sent segments to each other simultaneously.

→ It is done with a LISTEN-ACCEPT process, having listen and accept primitives, either specifying a specific source or nobody in particular.

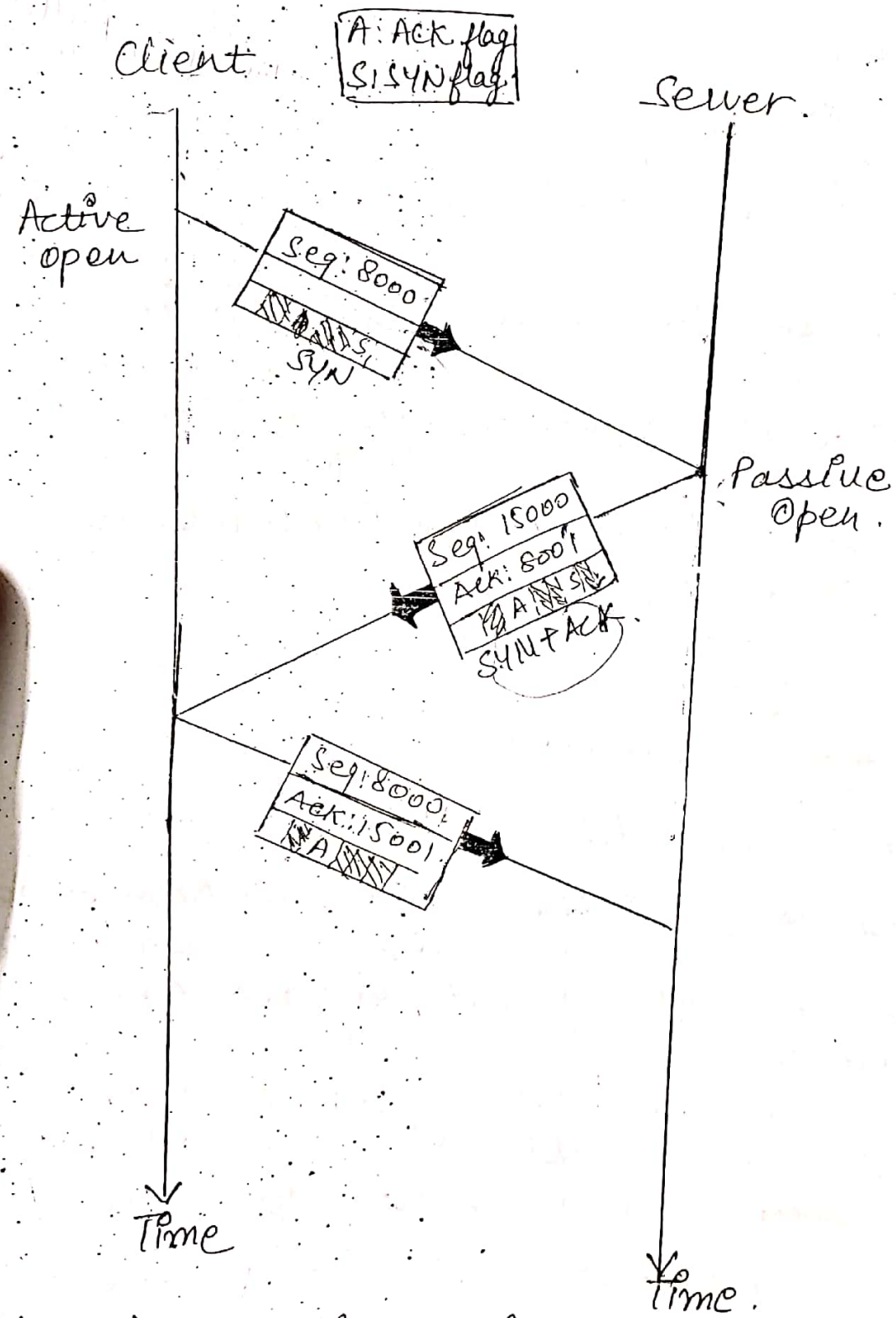
→ connection establishment serve three main purposes

(a) It allows each end to assure that the other exists.

(b) It allows exchange or negotiation of options parameters (eg. maximum segment size, maximum window size, quality of service). (8)

(c) It triggers allocation of transport entity resources (e.g. buffer space, entry in connection table).

→



Explanation of figure is on next page.

the given figure, the client and server both of them simultaneously attempt to establish connection between the same two sockets

ACK: The value of acknowledgment field is valid.

SYN: Terminate the connection.

ACTIVE OPEN:- The client program issues a request for an active open. A client that wishes to connect to an open server tells its TCP that it needs to be connected to the particular server. TCP can now start the three ways handshaking process.

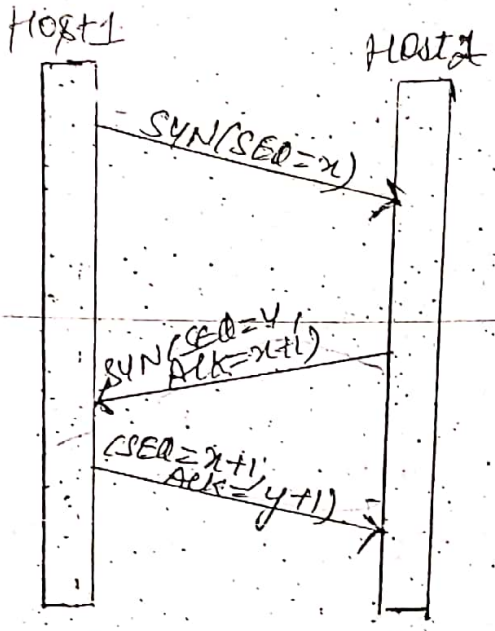
PASSIVE OPEN: The process starts with the server. The server program tells its TCP that it is ready to accept a connection. That is called a request for a passive open. Although the server TCP is ready to accept any connection from any machine in the world.

In this figure, the client send ~~this~~ first segment with a sequence number and requests for synchronization, here SYN flag gets set.

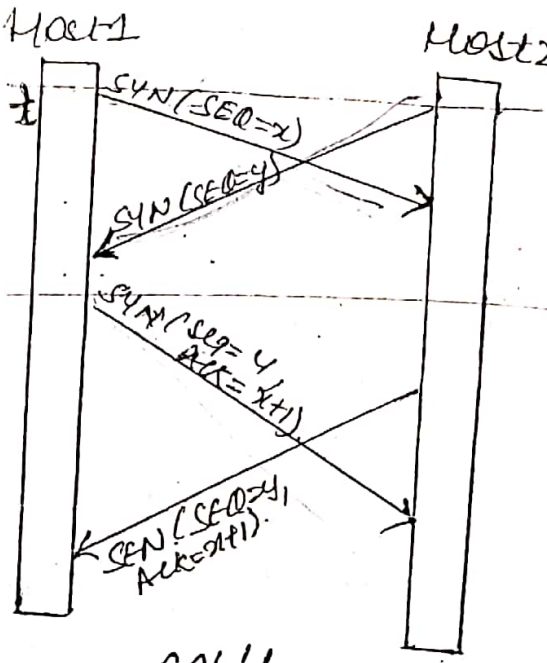
→ Further the server accepts the client's request and starts the communication by sending the acknowledgement. Here both SYN and Ack flag gets set.

→ Then client replies to server with an acknowledgement means giving the receipt of second segment, setting the Ack flag.

→ Sequence number in this segment is the same as the one in the SYN segment, this Ack segment does not consume any sequence number.



TCP connection establishments in the normal case.

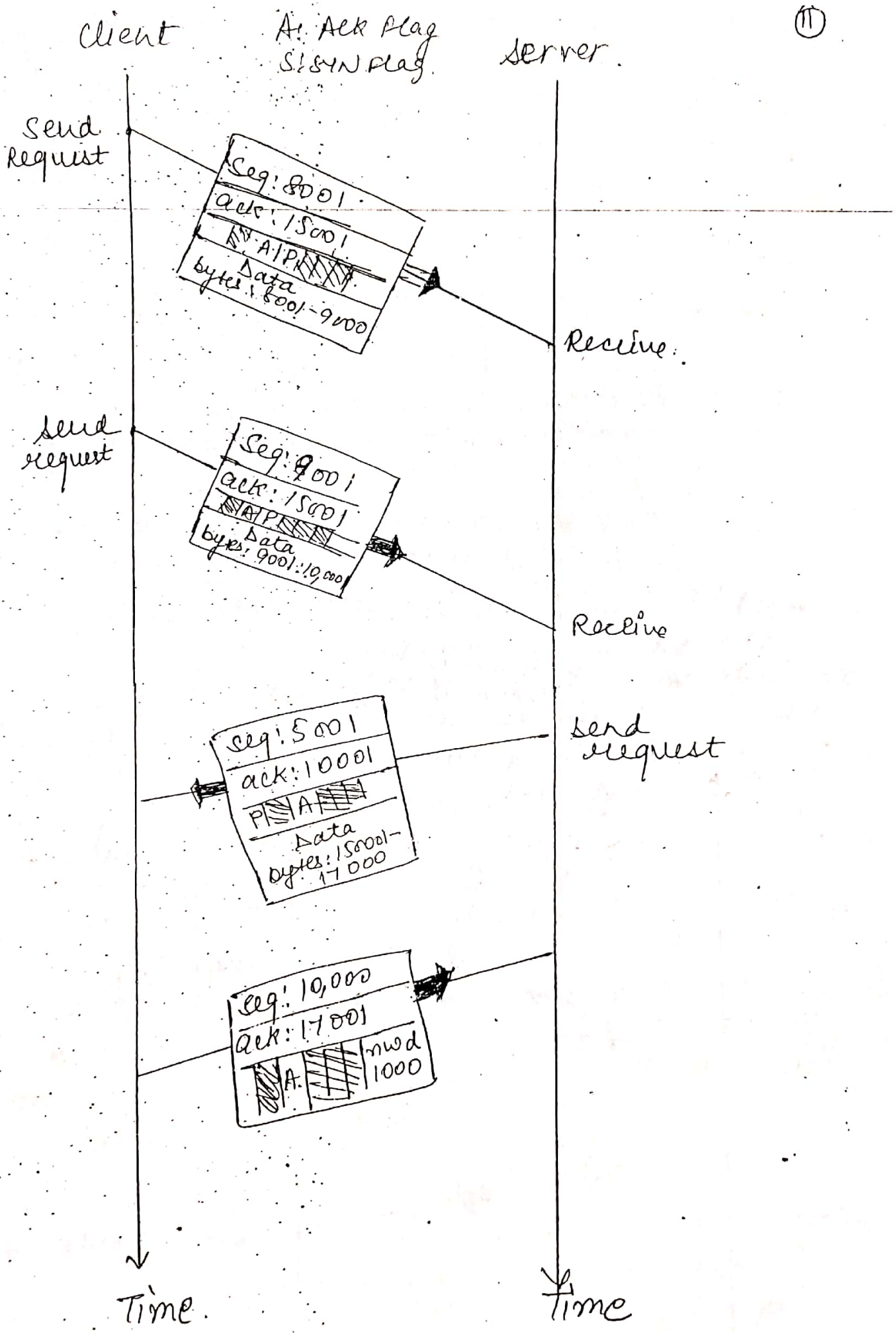


call collision.

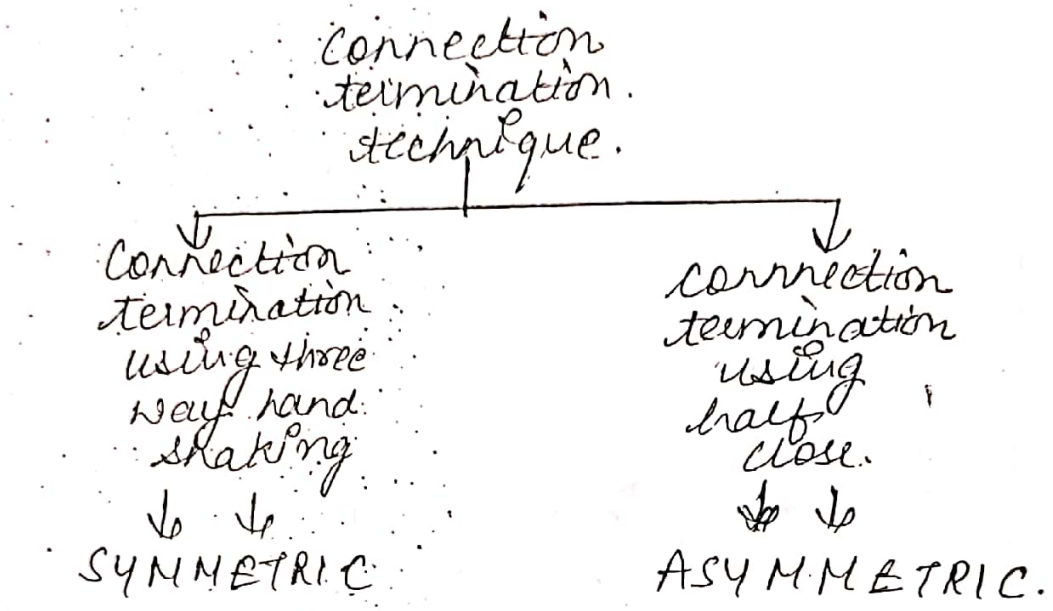
(a) simply Host 1 establishes a connection and Host 2 replies by sending the acknowledgment.

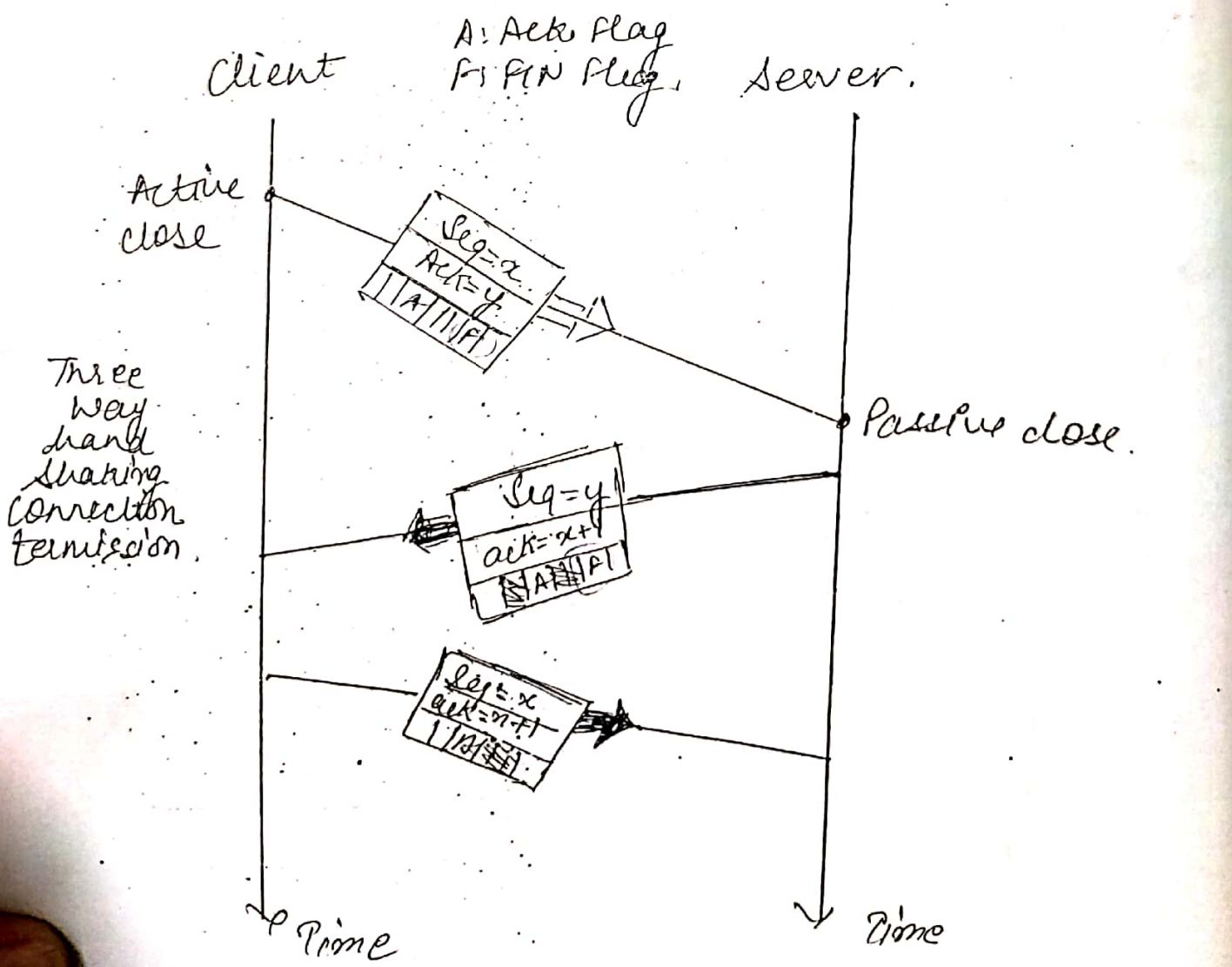
(a) In this, the two hosts simultaneously attempt to establish a connection between the same two sockets, ~~this sequence of~~, results in collision.

(b) Data Transfer :- After connection is established, bi directional data transfer we can take place. The client and server can both send data and acknowledgement. Data traveling in the same direction as an acknowledgement are carried on the same segment. The acknowledgement is piggy backed with the idea. (Diagram on next page)



c) Connection Termination or 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.





Explanation:- (Refer to connection establishment).

In the given figure, Active close sends (13) the acknowledgement and FIN, i.e to terminate the connection.

FIN segment can include the last chunk of data sent by the client or it can be just a control segment.

A FIN segment consumes one sequence number if it does not carry data.

Now server sends the confirmation that it had accepted the FIN and acknowledgement that FIN request is accepted.

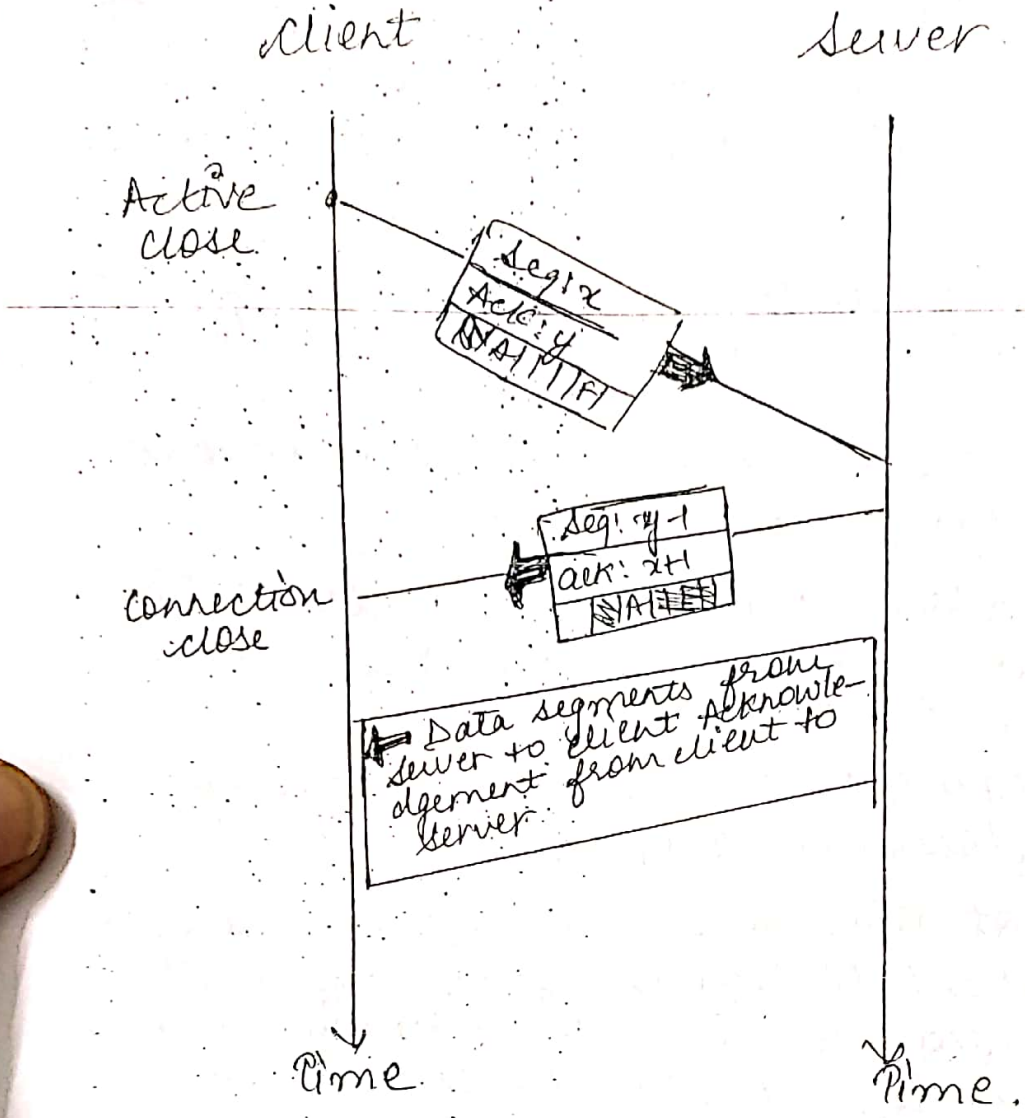
A FIN+ACK segment consumes one sequence number if it does not carry data.

Then the client also send the acknowledgement to receipt of the FIN segment from the TCP server. This segment contains the acknowledgement number, which is 1 plus the sequence number received in the FIN segment from the server.

This segment cannot carry data and consumes no sequence numbers.

HALF close:- In TCP one end can stop sending data while still receiving data. This is called half close OR Half duplex. Although either end can issue a half close, it is normally initiated by the client. It can occur when the server needs all the data before processing can begin.

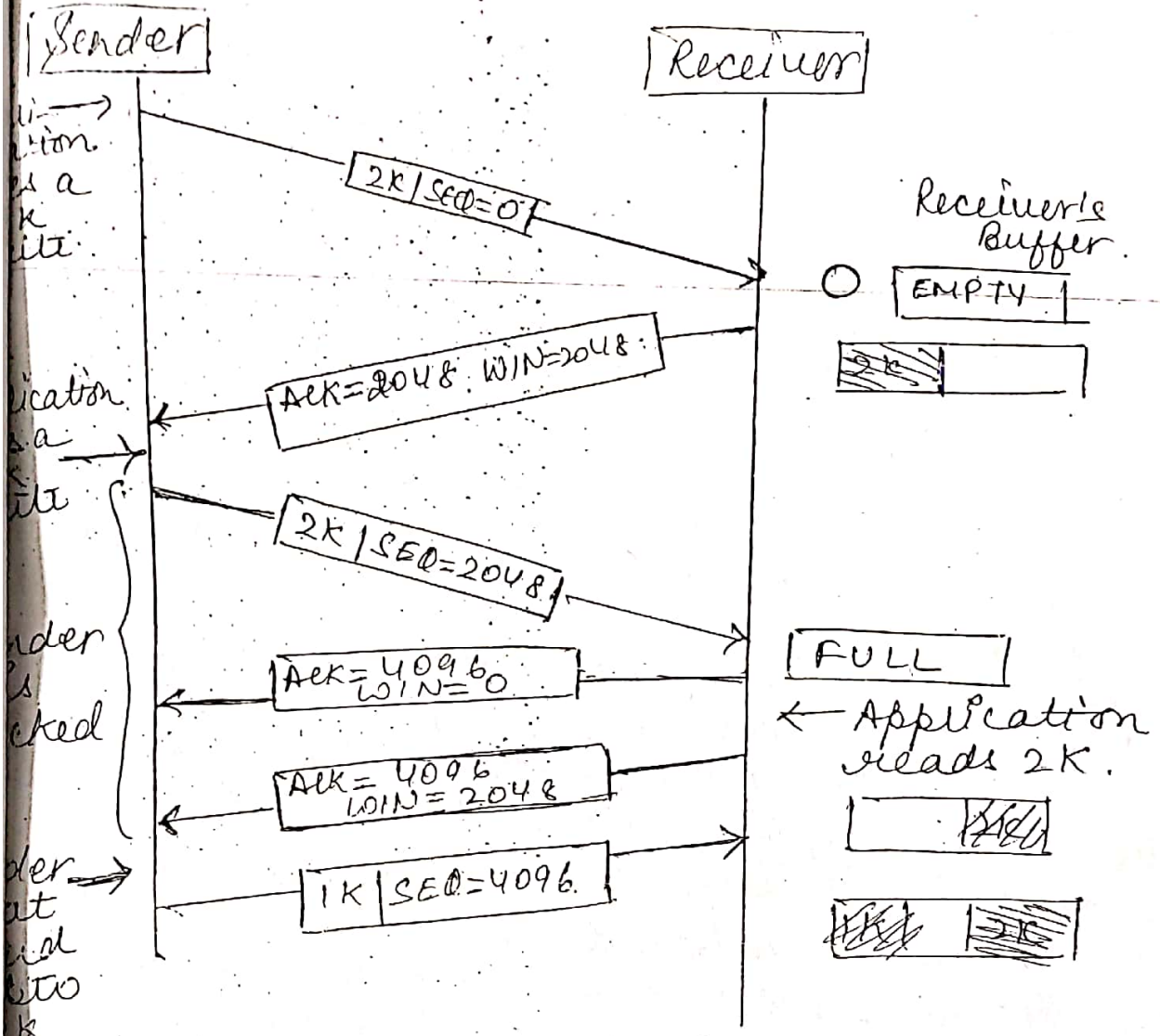
Half close → one host stops sending message. Receiver stops receiving message and after some time the connection is terminated.



→ In this figure, client has send the acknowledgement and request for termination.
→ Server is sending the acknowledgement but it has not send the request for termination.
→ One way communication is present here.
→ Here data segments are being send from server to client and acknowledgements are being send from client to server.

TCP TRANSMISSION POLICY :-

Transmission policy is basically used to describe the buffering mechanism when the speed of sender and receiver is not same.



→ In above figure, it is shown that receiver can receive data till that time till it has ~~emp~~ space and sender will get blocked till that time until some space gets empty at receiver sider.

→ Here sender is sending 2k data, this time receiver buffer is empty so it can accept the data as it has 4k space.

→ Receiver then sends the reciept of acknowledgement to sender.

→ Sender can now send ~~only~~ data till the limit of 2k as 2k is the size of buffer remaining at receiver side.

→ Receiver's Buffer is full when it receives 2k data from sender.

→ Now sending process is blocked.
→ The sender must stop until the application process on receiving host has removed some data from the buffer, at which time the TCP can advertise a larger window.
→ Now sender may send a 1-byte segment to make the receiver reannounce the next byte expected and window size.

(a) Nagle's algorithm :- If an application generates data one octet at a time, TCP will send the first octet immediately. However, until the ACK arrives, TCP will accumulate addition octet in its buffer. If the application is reasonably fast compared to the network successive segments will each contain many octets. If the application is slow compared to the network,

→ A way to reduce this usage is known as Nagle's algorithm, what Nagle suggested is simple: when data come into the sender one byte at a time, just send the first byte and buffer all the rest until the outstanding byte is acknowledged.
→ Nagle's algorithm is widely used by TCP implementations but there are times when it is better to disable it.
→ The outgoing TCP buffer begins with sufficient data for atleast one maximum size segment.
→ Nagle's algorithm additionally allows a new packet to be sent if enough data have trickled into fill the half of the window or a maximum speed.

Silly window Syndrome :-

Silly window syndrome (Clark 1982), this problem occurs when data are passed to the sending TCP entity in large blocks, but an interactive application on the receiving side reads data 1 byte at a time.

When a connection is first established, the receiving TCP allocates a buffer of K bytes and uses the WINDOW field in acknowledgment segments to advertise the available buffer size to the sender.

Syndrome created by the sender :- The sending TCP may create a silly window syndrome if it is serving an application program that creates data slowly.

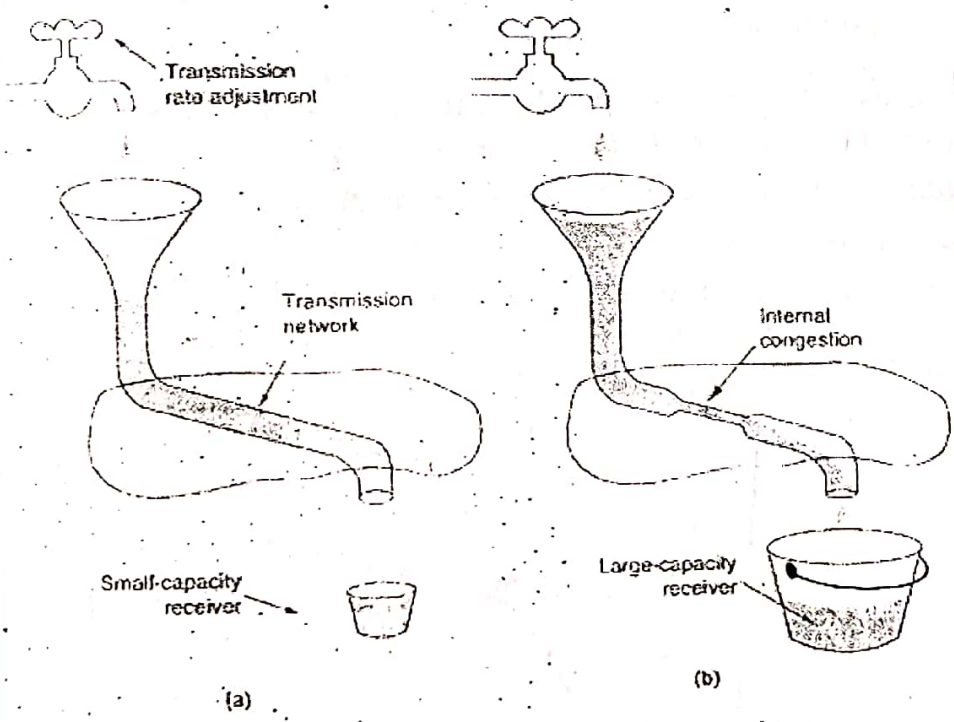
Syndrome created by the receiver :- The receiving TCP may create a silly window syndrome if it is serving an application program that consumes data slowly.

Avoiding Silly window Syndrome :- TCP specifications now include heuristics that prevent silly window syndrome. A heuristics used on the sending machines avoids transmitting a small amount of data in each segment. Another heuristics used on the receiving machine avoids sending small increments for window advertisements that can trigger small data packet.

TCP CONGESTION CONTROL

When the load offered to any network is more than it can handle, congestion builds up. The first step in managing congestion is detecting it. Earlier, A timeout caused by a loss of packet. Reason for packet loss could be
(1) noise on a transmission line
(2) packet discard at a congested router.

Nowadays, fibre optics have reduced packet loss due to transmission errors. In Fig-(a), we see a thick pipe leading to a small-capacity receiver. As long as the sender does not send more water than the bucket can contain, no water will be lost. In Fig-(b), the limiting factor is not the bucket capacity, but the internal carrying capacity of the network. If too much water comes in too fast, it will back up and some will be lost (in this case by overflowing the funnel).



(a) A fast network feeding a low-capacity receiver.
(b) A slow network feeding a high-capacity receiver.

Two problems are there: **Network capacity and Receiver capacity.** To solve these, each sender maintains two windows:

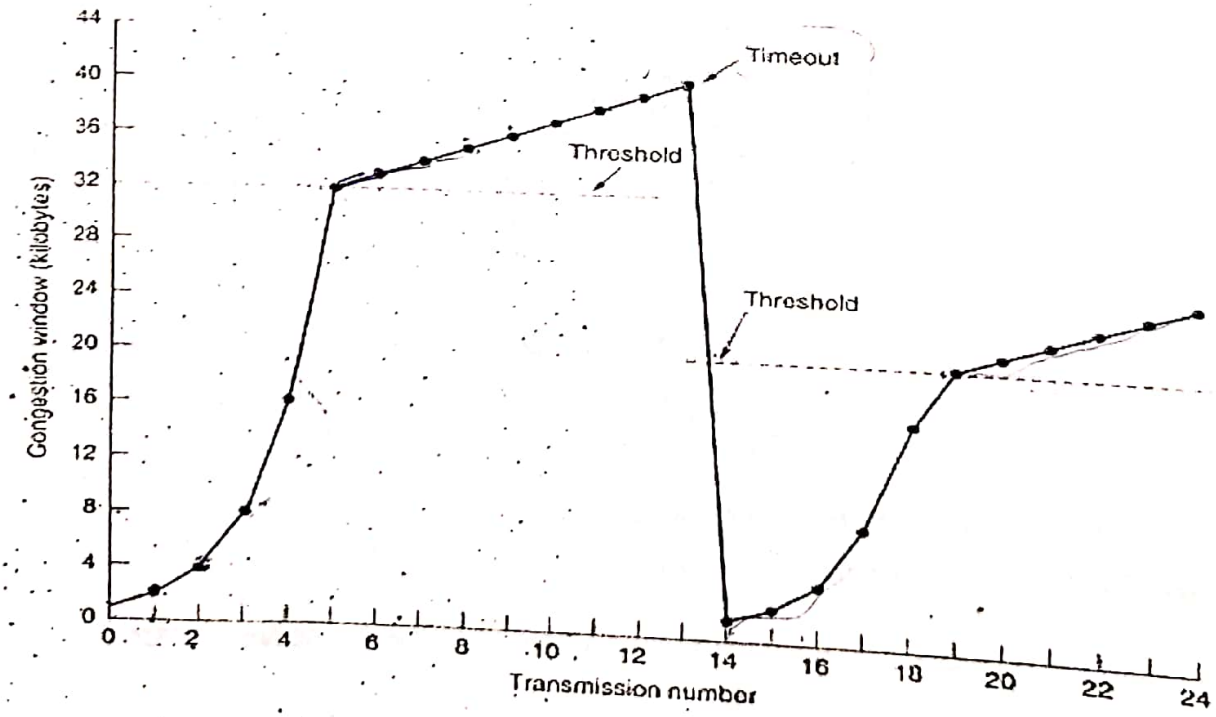
- Window granted by receiver.
- Congestion window.

The number of bytes that may be sent is the minimum of the two windows.

When a connection is established, the sender initializes the congestion window to the size of the maximum segment in use. It then sends one maximum segment. If this segment is acknowledged before the timer goes off, it adds one segment's number of bytes to the congestion window to make two maximum size segments and sends two segments. When the congestion window is **'n'** segments, if all *'n'* are acknowledged on time, the congestion window is increased by the bytes according to *'n'* segments.

The congestion window keeps growing exponentially. If segment of size, 1024, 2048, and 4096 bytes are working but a burst of 8192 bytes gives a timeout, the congestion window should be set to 4096 to avoid congestion. As long as the congestion window remains at 4096, no bursts longer than that will be sent, no matter how much window space the receiver grants. This algorithm is called **slow start**, but it is not slow at all. It is exponential.

"Congestion control algorithm" uses a third parameter, **threshold**, initially 64 KB. When a timeout occurs, the threshold is set to half of the current congestion window, and the congestion window is reset to one maximum segment.



The maximum segment size here is 1024 bytes. Initially, the congestion window was 64 KB, but a timeout occurred, so the threshold is set to 32 KB. The congestion window then grows exponentially until it hits the threshold (32 KB). Starting then, it grows linearly.

TCP TIMER MANAGEMENT

टाइमर की क्या value decide करे कि उस time period में acknowledgement आ जाए। और इसी टाइमर की value को फिक्स करने के process को timer management कहते है।

TCP uses multiple timers such as **retransmission timer**. When a segment is sent, a retransmission timer is **started**. If the segment is acknowledged before the timer expires, the timer is stopped. But, if the timer goes **off** before the acknowledgement comes in, the segment is retransmitted. The question that arises is: How long should the timeout interval be?

If the timeout is set too short, unnecessary retransmissions will occur. If it is set too long, retransmission delay, whenever a packet is lost.

The solution is to use a highly dynamic algorithm that constantly adjusts the timeout interval, based on continuous measurements of network performance. For each connection, TCP maintains a variable, *RTT*. 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. M is the time taken to receive the acknowledgement back. The formula

$$RTT = \alpha RTT + (1 - \alpha)M$$

$RTT(1-\alpha) = (1-\alpha)M$

$RTT = \frac{(1-\alpha)M}{(1-\alpha)}$

where α is a smoothing factor that determines how much weight is given to the old value. Typically $\alpha = 7/8$.
Normally, TCP uses βRTT, and β is kept constant which is not always true.

A new formula came which said that whenever an acknowledgement comes in, the difference between the expected and observed values, $|RTT - M|$, is calculated. A simple value of this is maintained in *D* by the formula

$$D = \alpha D + (1 - \alpha)|RTT - M|$$

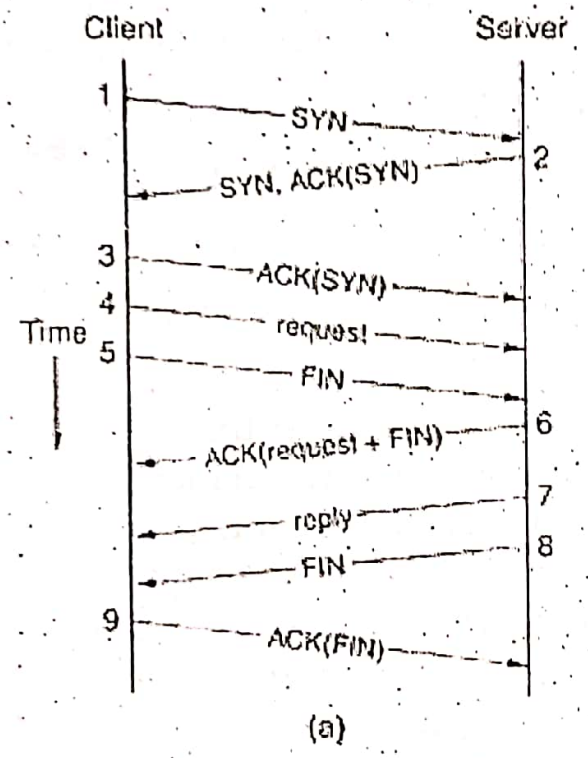
Most TCP implementations now use this algorithm and set the timeout interval to

Timeout = $RTT + 4 \times D$

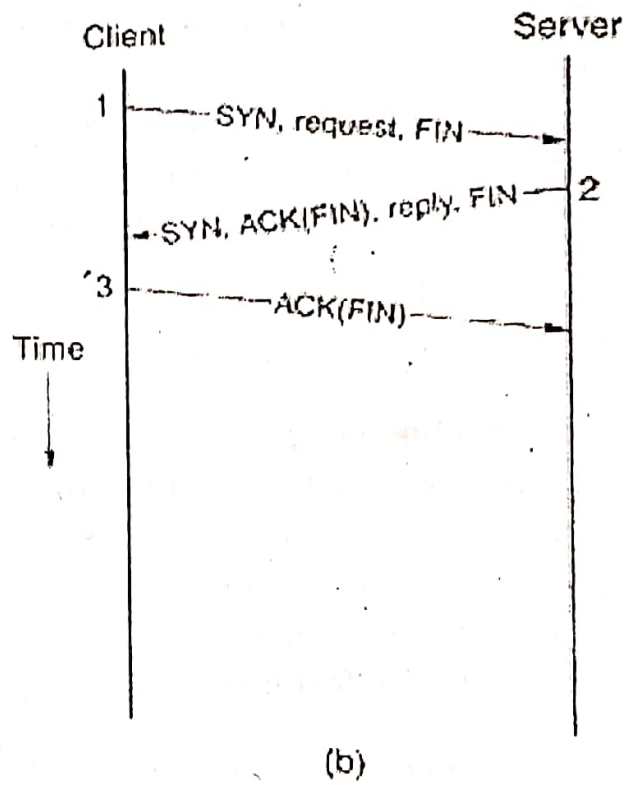
Transactional TCP

UDP can simply be used when, both the request and reply are small enough to fit into single packets. But, if the reply is large, then the pieces must be sequenced and a mechanism must be designed to retransmit lost pieces.

So TCP is to be used. The problem is the efficiency. (RPC-remote procedure call)



(a) RPC using normal TCP



(b) RPC using T/TCP

The nine packets are as follows:

- 1. The client sends a *SYN* packet to establish a connection.
- 2. The server sends an *ACK* packet to acknowledge the SYN packet.
- 3. The client completes the three-way handshake.
- 4. The client sends the actual request.
- 5. The client sends a *FIN* packet to indicate that it is done sending.
- 6. The server acknowledges the request and the *FIN*.
- 7. The server sends the reply back to the client.
- 8. The server sends a FIN packet to indicate that it is also done.

9. The client acknowledges the server's *FIN*.

Is there some way to combine the 'efficiency of UDP' with the 'reliability of TCP'? Almost, It can be done with an experimental TCP variant called **T/TCP (Transactional TCP)** it modify the standard connection setup sequence slightly to allow the transfer of data during setup.
Another proposal is **SCTP (Stream Control Transmission Protocol)**. Its features include message boundary preservation, multiple delivery modes and selective acknowledgements.

FAIRNESS

सबको बराबर बाँटना।

Consider 'K' TCP connections but all passing through the same link with rate 'R' bits-per-sec.

A congestion control mechanism is said to be **fair** if all connection's transmit rate is R/K i.e. each connection get equal share of link bandwidth.