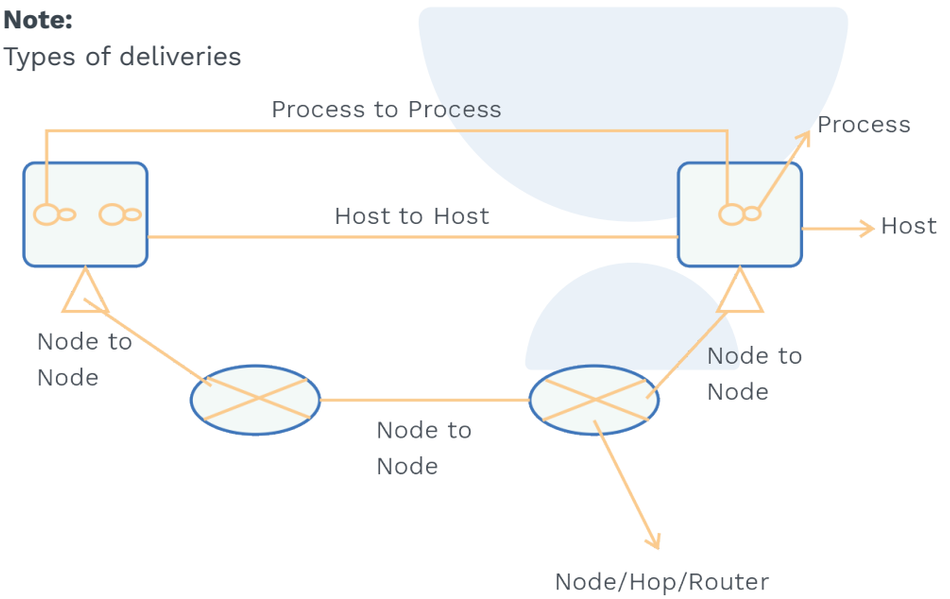
5 Transport Layer

5.1 TRANSPORT LAYER

- Main objective of the Transport layer is to deliver the packet from process to process.
- Transport layer protocol may be connectionless or connection-oriented depending on the how segments are treated.
- Transport layer may be responsible for error control and flow control, but these facilities are also provided by the data link layer, so what is the need here? The need is in data link layer error control and flow control is provided on the link only, but, here error and flow control is provided end to end.
- Transport layer protocols are UDP, TCP and SCTP.

Note:
Types of deliveries



- Data link layer is responsible for Node to Node delivery.
- Network layer is responsible for Host to Host delivery.
- Transport layer is responsible for Process to Process delivery.

Addressing mode at the transport layer is done using Port number, **Why the port number is needed, is IP address and MAC address are not sufficient?** Since at any Host, there may be many process running, in order to find out which process to reach, we need port number.

Rack Your Brain

How addressing is done at the data link layer and network layer?

In TCP header port number is 16 bit field. It ranges from 0 to 65535.

Port number	Protocol	Description
20	FTP, Data	File Transfer Protocol (data connection)
21	FTP, Control	File Transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol

Fig. 5.1 Port Number and Corresponding Protocols

PRACTICE QUESTIONS

Q1 **What is socket addressing?**

Sol: The combination of Port number and IP address is called socket addressing. Client socket identifies the client process uniquely and server socket identifies the server process uniquely.

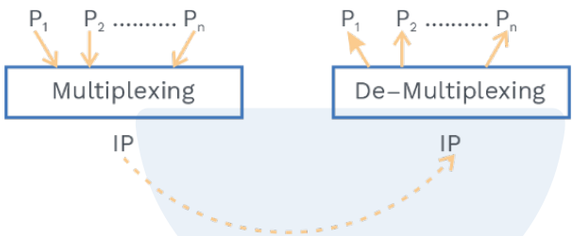
The IANA (Internet Assigned Number Authority) divides the port numbers into

- Well known,
- Registered, and
- Dynamic (or private)

Note:

Socket Address: IP Address + Port Number

Transport layer provides multiplexing and demultiplexing.

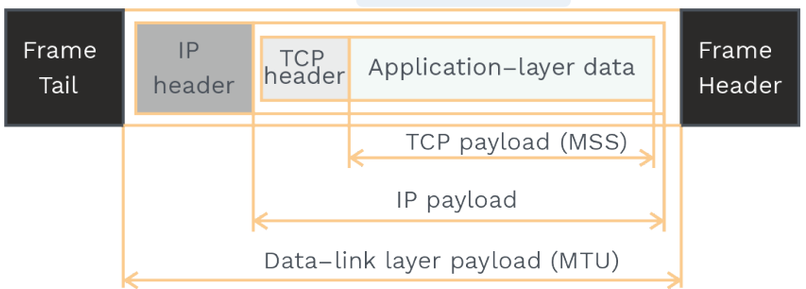


There are two types of major protocol in transport layer.

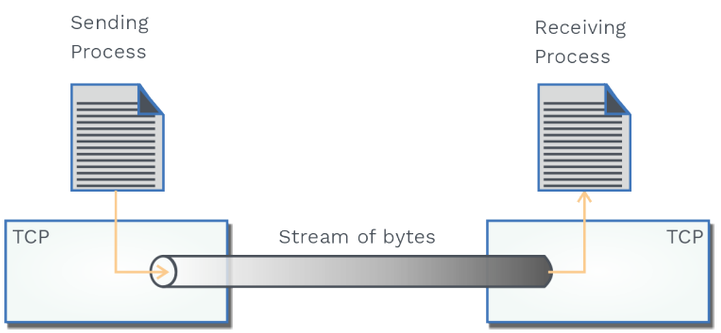
TCP and UDP

Lets see TCP,

Before going further lets see How TCP encapsulates.



- Transport layer provides stream delivery service.



TCP handles congestion (by reducing the sender window size)
How TCP Keep counts its segment,

1) Byte Number
2) Sequence Number
3) Acknowledgement Number

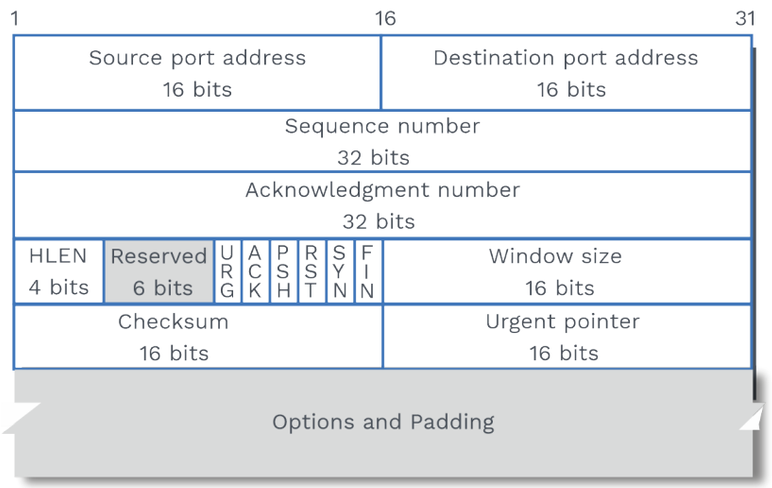
1) Byte number:
TCP count all the data bytes that needs to be transmitted.
Numbers can be started from any random number between 0 to 2^{32}-1.

Both are correct.
Data is sent to Transport Layer from Application Layer with no limitations. Now its Transport Layer duty to divide the data into chunks called as segments. Each segment is a collection of bytes.

2) Sequence number: For the first data byte of the segment, a number is given, where the same value is given in the field of the sequence number of in TCP header.

3) Acknowledgement number: The acknowledgement number defines the number of the next byte that the party expects to receive.
Lets See TCP segment format,

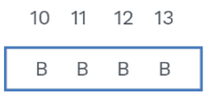
a. Segment



b. Header

Fig. 5.2 TCP Header Format

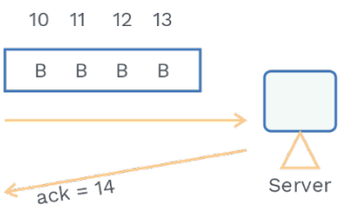
- **Source Port address:** It is a 16 bit address used for port for sending application.
- **Destination Port address:** It is a 16 bit address used for port for receiving application.
- **Sequence number:** It is 32 bit field, In order to ensure connectivity TCP need to make every byte which has to be transmitted as numbered, now the first byte number will be the sequence number.



Example: What will be the value at the sequence number field?

Sol: It is 1010 i.e 10

- **Acknowledgement number:** This field contains expected byte sequence number(i.e. expecting by receiver to get next from sender).



Why ack = 14, Why not 13?
By sending acknowledgement 14 server is saying I have taken bytes from 10 to 13 and now expecting the next segment whose the sequence number should be 14.

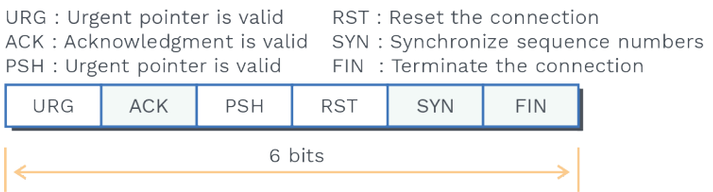
Header length:
- It has 4 bit.
- It defines length of TCP header.

Grey Matter Alert!

- We have minimum and maximum lengths of 20 and 60 bytes respectively.
- But at header length, we have only 4 bits i.e using 4 bits maximum we can go upto 15 bytes.
- It leads to the concept of scaling factor; in this case it is 4 bytes.
- Header length = Header length field value * 4 bytes.

Note:
What is the minimum and maximum length of the TCP header?
Minimum length = 20 Bytes, How? number of essential rows * size of each rows i.e 5 * 4 bytes
Maximum length = 60 bytes, How? Maximum size of options are 40 bytes, 20 + 40 = 60 bytes

- **Reserved bit:** These have 6 bit which is reserved for future use,



- **URG bit:** It tells there is some data which is urgent if the flag of the URG bit is set to 1.
  How one can know, How many bytes are urgent?
  For this reason we have an urgent pointer field.

- **ACK bit:** It tells about the validity of the acknowledgement number.
  Except for request segment, which is used in connection establishment, all the segment may have ACK = 1.

- **PSH bit:** For immediately pushing the entire buffer to receiver application, the PSH is used.

  Note:
  Difference between the URG bit and the PSH bit is that the PSH bit does not give any priority to the data and It just causes all the segments in the buffer to be pushed immediately to the receiving application.

- **RST bit:** It is used for resetting the application. It tells the receiver that something is wrong, please disconnect the resources and buffer.

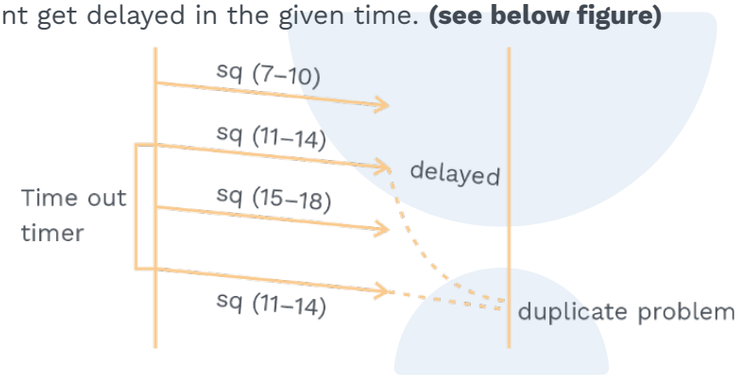
  When should I use the RST bit?
  When normal execution is not possible, there is a need for immediate termination.
  It may result in the loss of the data that are in transmission.

- **SYN bit:** This bit says about the TCP header sequence number field value to the receiver and indicates that the field value is the initial sequence number.

- **FIN bit:** For connection termination, it is used.
  Let's see **"What is wrap-around time"** and How sequence numbers deal with it !
  If the initial sequence number is chosen as S then the sequence number will be used from S to 2^{32}-1 and again from 0 to S.

The time taken to use all the sequence number is called Wrap-Around time.

- In modern computers, the lifetime of the TCP segment is 180 second.
- Entire concept lies in the fact that sequence number should be unique in a given time.
- Other wise otherwise duplicate segment problem will be there if Any segment get delayed in the given time. **(see below figure)**



PRACTICE QUESTIONS

Q2 **What will be the sequence number if we want no wrap around in the Life time of packet considering the bandwidth as B MBps.**

Sol: We know life time of packet is 180 sec.

2^x / B > 180 (let's say x is no. of bits used for sequence number)

2^x > 180 * B

Sequence Number = 180 * B

Number of bits needed for sequence number = $\log_2(180 * B)$

A wrap-around is always needed !! not at all, whenever we run out of sequence number then only wrap-around time is needed

- **Window size:** It is 16 bit field (0 to 65535)
  It defines the size of receiving window determined by the receiver.

- **Checksum:** It is 16 bit field.
  It checks for the error in the TCP header on the mandatory basis but in UDP the checksum is not mandatory.
  Receiver rejects the data that fails in CRC integrity.

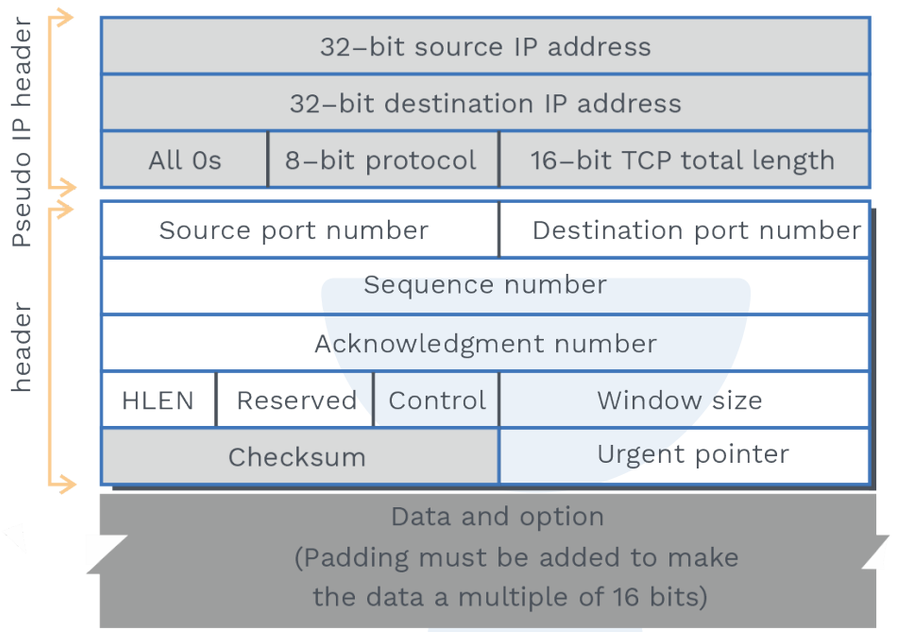


Fig. 5.3 TCP Header Format

Checksum at TCP can be done in 3 section TCP: pseudo IP header, TCP header, Data section (See above figure).

Urgent pointer: It is a 16 bit field.

Which is valid only when the urgent flag is set, it tells what number of byte is urgent in segment.

Note:
Number of urgent bytes = Urgent pointer + 1
End of urgent byte = Sequence number of the first byte in the segment + Urgent pointer

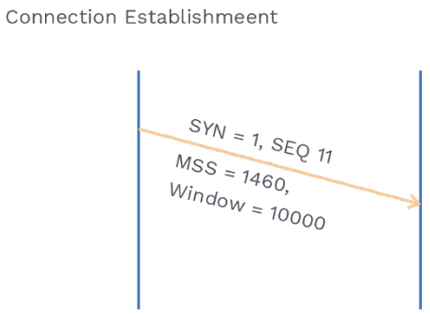
We know TCP is connection oriented, i.e. path are reserved.

TCP has 3 phases for connection oriented services:

- Connection establishment
- Data transfer
- Connection termination

Connection establishment:

Step 1



Why SYN = 1?

It is informing that the client wants to connect, this is why the SYN = 1.

Why SEQ = 11?

It is saying that the starting byte of this packet is 11, **Can it be any other number!** Yes it can be any random number between 0 to 2^{32}-1.

Why MSS = 1460?

This is the maximum segment size which the client can hold.

Why Window size = 29200?

This window size which the client can accommodate is 29200, Actually it can accommodate from 0 to 65535 and more (from options).

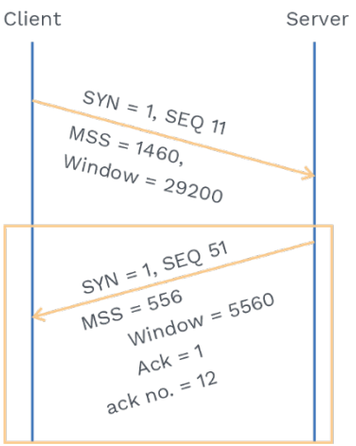
From where do we get MSS?

From option field

Note:

Though the SYN segment cannot carry data, it needs one sequence number.

Step 2



Why SYN = 1?
Now Server also wants to establish connection, that is why SYN = 1.

Why SEQ = 51?
Segment which server is sending its first-byte containing a sequence number = 51.

Why MSS = 556?
Server is saying to the client that I can accept a segment of 556 bytes only.

Why window size = 5560?
It basically tells client that my window size is 5560 bytes.

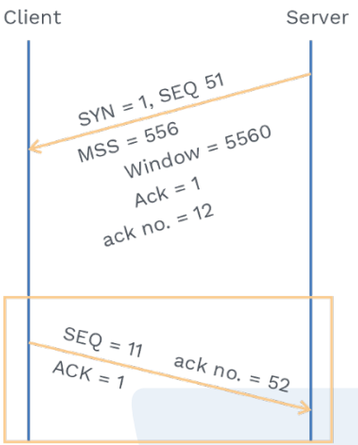
Why ACK = 1?
This means server is telling that it accepted the previous segment which client has sent.

Why is acknowledgement number 12?
Acknowledgement number always signifies the next byte, which is expected by the receiver.
It means the Server has accepted byte having sequence number 11 now expecting 12.

Note:
A SYN + ACK segment cannot carry data, but does consume one sequence number.

Note:
In any TCP segment
If SYN bit = 1 and ACK bit = 0, then it must be request segment.
If SYN bit = 1 and ACK bit = 1, then it must be reply segment.
If SYN bit = 0 and ACK bit = 1, then it must be pure acknowledgement or data segment.
If SYN = 0 and ACK = 0, then this is not possible.

Step 3



Why Seq = 11?
(Since Server ack no = 12, this means server was asking for that byte whose sequence no is 12.) Since, SIN = 0 and ACK = 1, it is a pure acknowledgment. Pure acknowledgment does not consume any sequence number. It will use previously used sequence number.

Why ack no = 52?
As client had taken 1 byte, when the server sent a segment having sequence number 51, the next sequence number that the client is expecting from server is 52.

Note:
If no data is carrying ,It consumes no sequence number by an Ack segment.

Data transfer:
Lets send some data:
Let's say the client has some data (11 bytes) to send, it will tell its sequence number as 12, and acknowledgement number as 52.

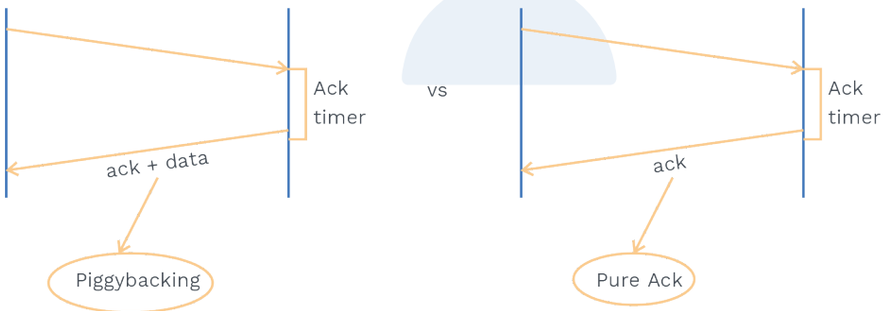
Now, the server has also some data (20 bytes) to send, it will send to the client and tells its seq no = 52 (next byte which client is expecting) and acknowledgement number as 23 (next byte which server is expecting)

Connection termination:

Any of the two parties can close the connection, although it's mainly dependent on the client-side.

Note:

Piggy backing and pure acknowledgement,

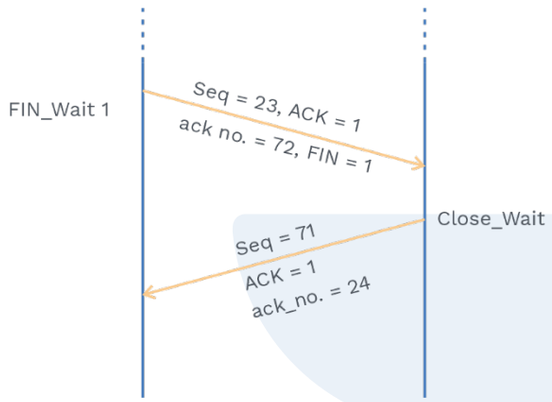


Note:

A TCP connection is terminated using the FIN segment where the FIN bit is set to 1.

Why Seq = 23?
Client is saying that my first byte starts with 23 only.

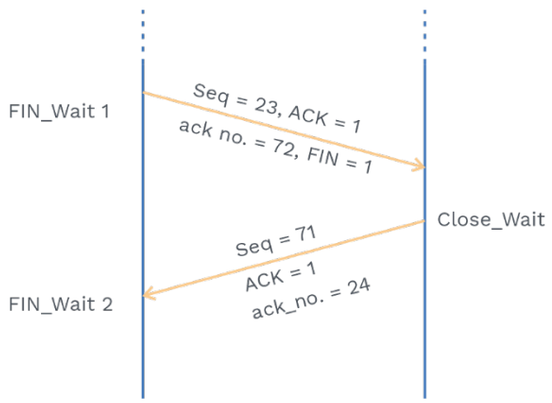
Why FIN = 1?
Client says that,It wants to end the connection, and it is going under FIN_WAIT 1 state.



Now the server sends an ACK to the client, and delivers outstanding data in its queue
to the application, and goes to the CLOSE-WAIT state.

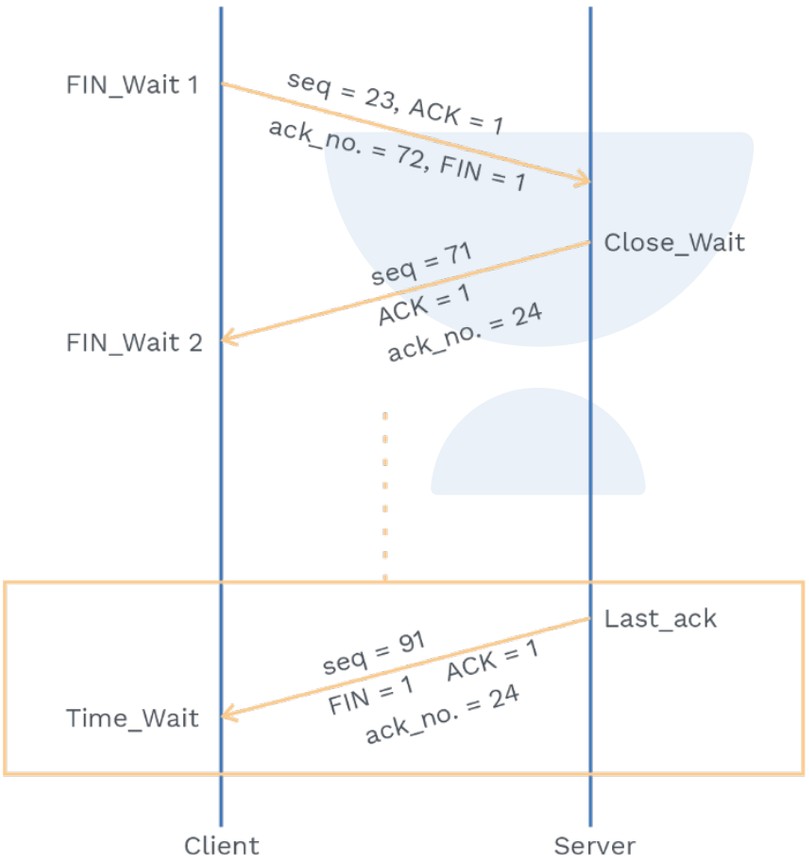
Why seq = 71, when the server is sending ACK?
Since this is pure acknowledgement. It does not use any sequence number.

Why acknowledgement no = 24?
Server is saying, if you want the next byte to be sent, then its sequence number is 24.
After receiving the acknowledgement from the server, the client enters the FIN_WAIT_2 state.



Note:
After Ack is received from server. Server releases its buffers. Though the client can still send pure acknowledgements to the server but not any data. But both data and ack server can send to client.

If server wants to close a connection, then it will send a FIN segment. After receiving FIN segment client goes into a Time wait state and sends ACK to Server saying that it is also freeing up my buffer.



Note:
The TIME_WAIT state allows the client to resend the final acknowledgement if it gets lost and the time spent by the client in TIME_WAIT maybe 2 min or 1 min it depends on the implementation of the protocol.

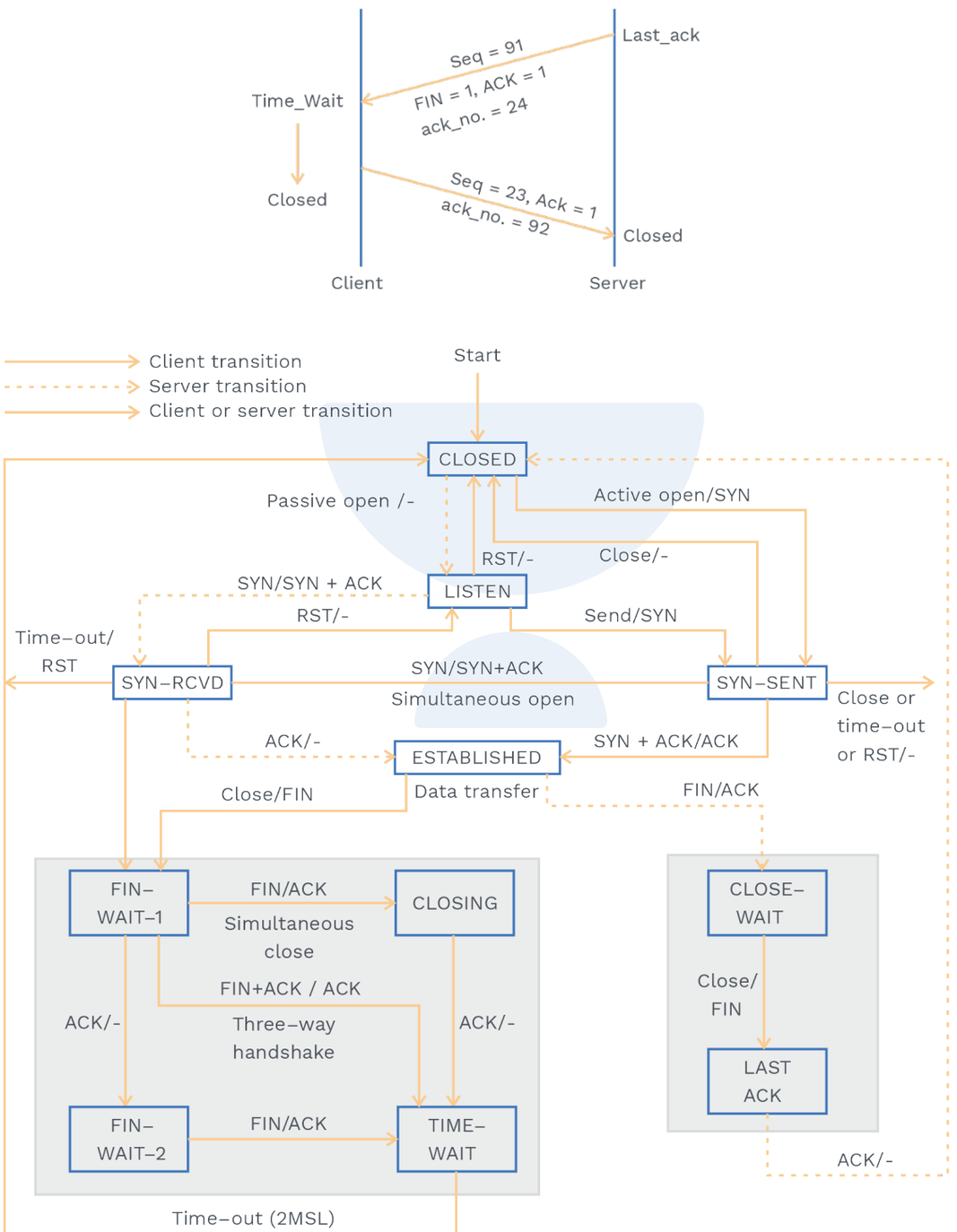


Fig. 5.4 TCP State Transition diagram

PRACTICE QUESTIONS

Q3 **What would be size of the window for host, if the value of receiver window is 1000 bytes and the value of cwnd is 999 bytes?**

Sol: Size of window for Host is min (rwnd, cwnd) => min(1000, 999) => 999.

rwnd → receiver window size

cwnd → congestion window size

Q4 **How Error control is provided by TCP?**

Sol: TCP provides reliability using error control by finding out what are segments which have been lost, by detecting corrupted segments, by finding out of order segments.

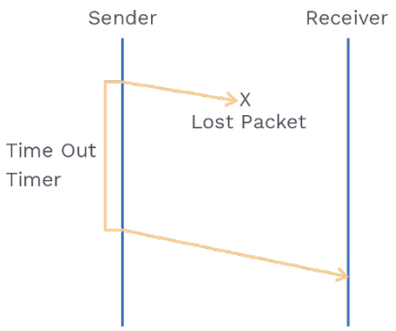
Note:
TCP has 3 main tools for detecting and correcting errors Checksum, acknowledgement and time out.

1) Checksum: If the checksum field is corrupted, then it means the segment has some error, and it will be discarded.
2) Acknowledgement: It is used for maintaining the confirmation of receiving the data segments.
3) For understanding the **Time out,** lets see how retransmission works in TCP.

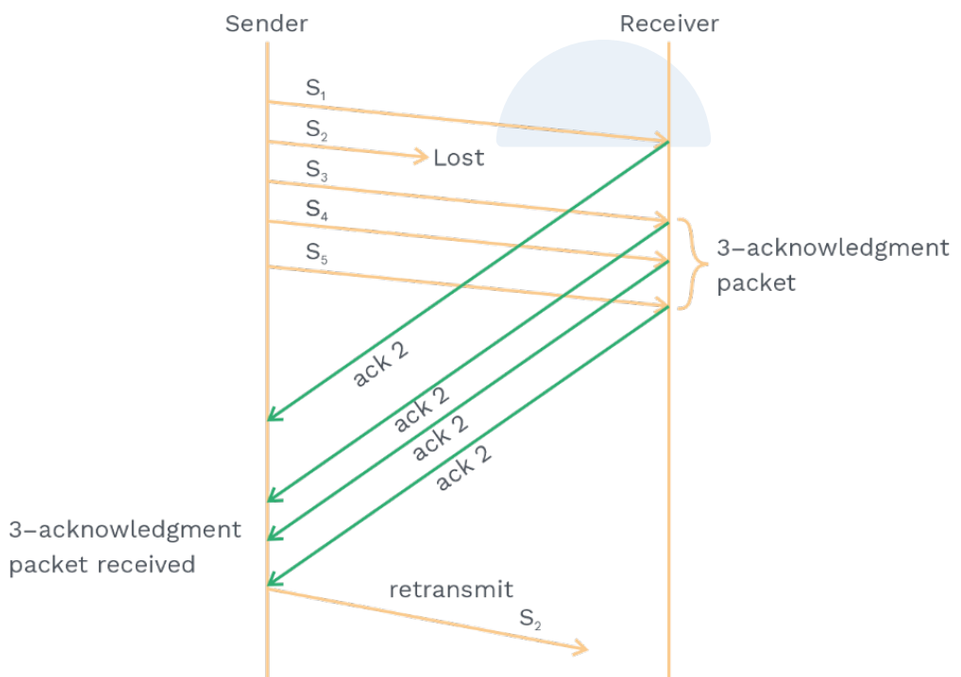
When the TCP segments get lost, then receiver needs to tell the sender using acknowledgement so that the sender can retransmit the packet.

Basically there are 2 cases in which the sender retransmits the packet.
1) **Timeout occurs**
2) **Three duplicates acknowledgements come back to the sender**

When Time Out occurs then sender will send the packet again.



- It leads to the possibility of strong congestion.
- When 3 duplicate acknowledgements come back to the sender, then it assumes that the corresponding segment is lost. Without waiting for the completion of time Out timer sender will retransmit the packet. (See the diagram below)
- After having the retransmitted S2, the Receiver sends the acknowledgement asking for S6 directly from the sender, and it will not ask for 3,4,5.
- It leads to the possibility of mild congestion.



Now, lets see congestion window.

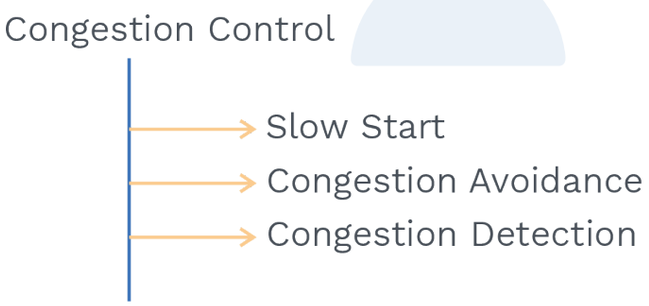
Sender has the information of rwnd size and cwnd size.

Note:
Actual swnd = min(rwnd size and cwnd size)
Swnd: Sender window
rwnd: receiver window
cwnd: congestion window

Congestion: It refers to the state of a system which slows down the network performance due to heavy traffic, and we can't avoid congestion completely.

Congestion control technique:
There is an assumption here, receiver window size (rwnd) is much larger than congestion window size (cwnd), so the sender window size is always equal to cwnd.

1) With this technique either we can prevent congestion before it happens.
2) We can remove congestion after it has happened.



Slow start phase:
The slow start algorithm is based on the idea that the size of the congestion window (cwnd) starts with 1 maximum segment size (MSS).

When will MSS be determined ! During the connection establishment phase.

- From where will we get MSS during the Connection establishment phase! From Options.
- The size of the window will increases 1 MSS each time one acknowledgement arrives.

Note:
Algorithm starts slowly, but grows exponentially.

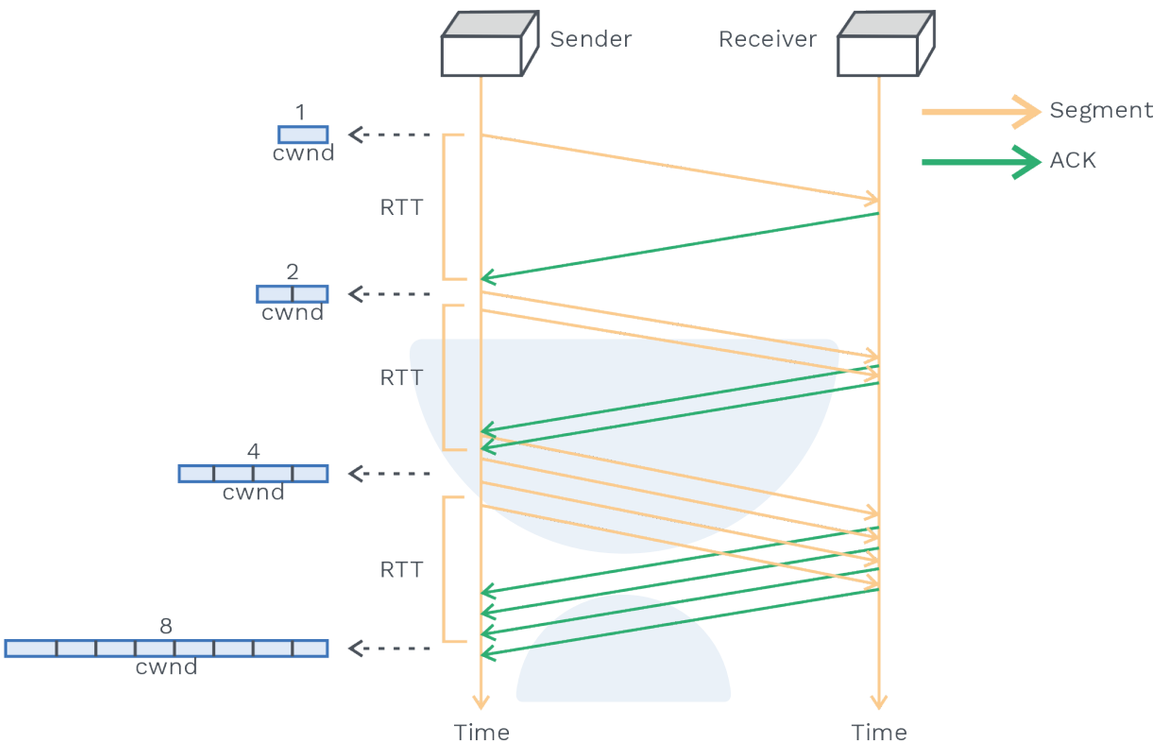


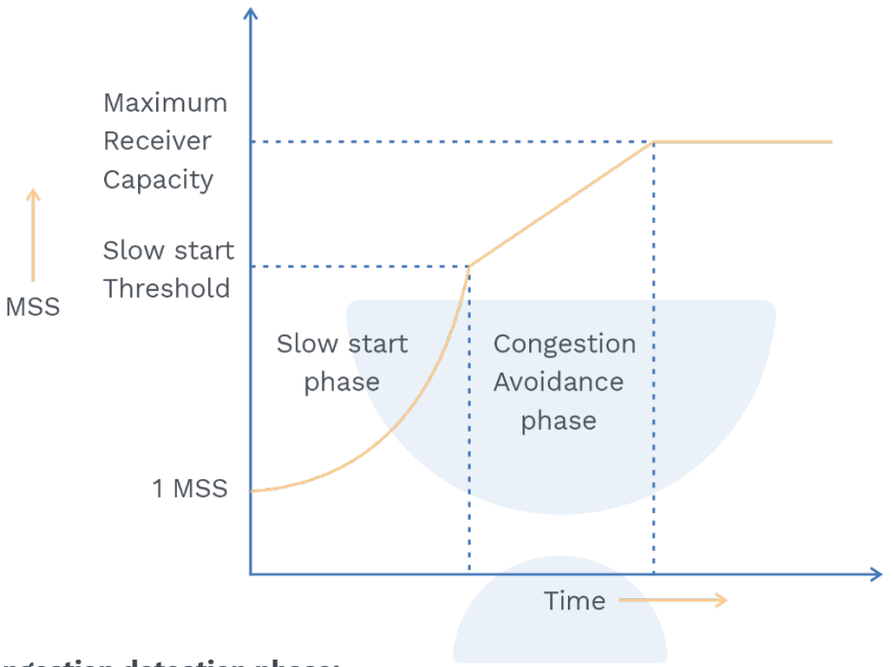
Fig. 5.5 Diagrammatic Representation of Congestion Control

Note:
In the slow start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.

Congestion avoidance phase:
We need to avoid congestion before it happens, it must slow down this exponential growth.
So, in this phase, the cwnd is incremented in a linear manner, i.e. after every ack receiving by the sender.
The sender increments the cwnd size by 1 mss.

cwnd: congestion window

Note:
This phase will go until the congestion window size becomes equal to the receiver window size or congestion is detected.



Congestion detection phase:
Case 1: When the timeout timer expires before receiving the acknowledgement for a segment. This case suggests a stronger possibility of congestion in the network.
What is the solution then!

- Sender in the slow start can set the half the size of current cwnd as the threshold.
- Decreasing the congestion window size to 1 MSS and Resume the slow start phase.

Case 2: When 3 duplicates acknowledgements are received. This case suggests the weaker possibility of congestion in the network.
What is the solution then!

- Sender should set the slow start threshold to half of the current congestion window size.
- Decreasing the congestion window size to slow the start threshold and resume the congestion avoidance phase.

USER DATAGRAM PROTOCOL

It is unreliable and connectionless protocol.

Why is UDP important?

When process communication depends on small message transfer and not on reliability, then UDP is important.

It does not guarantee in-order delivery, and it does not provide congestion control mechanism.

UDP header:

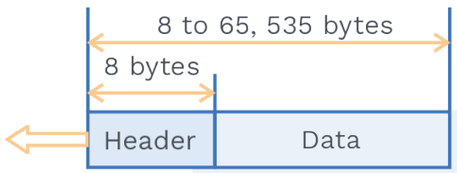


Fig. 5.5a. UDP user Datagram

Fig. 5.5b. Header Format

- **Source Port number:** It is the 16 bit field, It is the port number used by the process running on the source port.
  If the source host is the server, the port number, in most cases, is a well-known port number.
- **Destination Port number:** It is the 16 bit field, It identifies the port of the receiving application.
- **Total length:** It is the 16 bit field; It defines the total length.

  Total Length = Length of UDP Header + Length of data.
- **Checksum:** It is a 16 bit field. It is calculated on UDP Header, IP pseudo-header and data. Checksum calculation is not mandatory in UDP.

Previous Years' Question

Q) In an IPv4 datagram, the M bit is 0, the value of HLEN is 10, and the value of total length is 400, and the fragment offset value is 300. The position of the datagram, the sequence numbers of the first and the last bytes of the payload, respectively are:

a) Last fragment, 2400 and 2789
b) First fragment, 2400 and 2759
c) Last fragment 2400 and 2750
d) Middle fragment 300 and 689

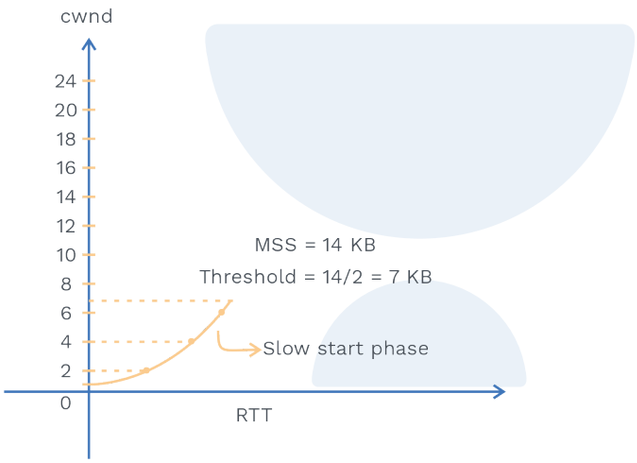
Sol: c) **(GATE-2013)**

Note:
Size of the UDP header is fixed.

PRACTICE QUESTIONS

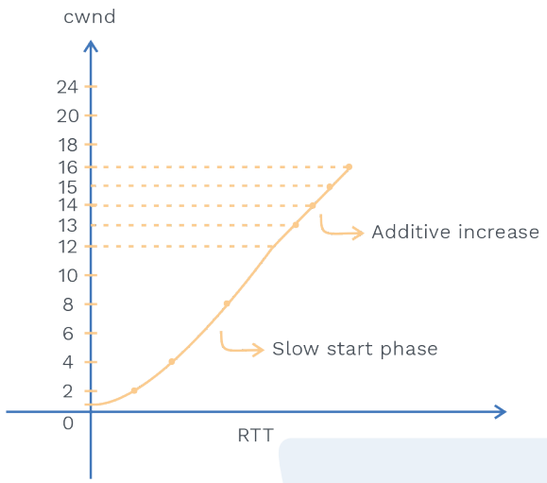
Q5 **Draw the graph if the congestion window size when time out occurs is 14KB in the slow start phase?**

Sol:



Q6 **Draw the graph to show how additive increase will behave if congestion window size when time out occurs is 24 KB?**

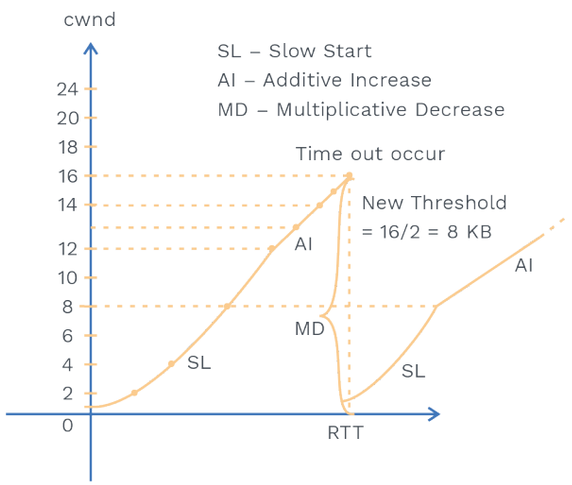
Sol: Threshold = 24 / 2 = 12. Till 12 KB it will be in slow start phase and after 12 KB it will be in additive increase.



In additive increase it will go to the MSS or Time Out occurs.

Q7 **Draw the graph to show how additive increase will behave if congestion window size is 24 KB and initial threshold is 12KB and time out occurs at 16KB?**

Sol: When Time out occurs at 16KB it will go into slow start phase.

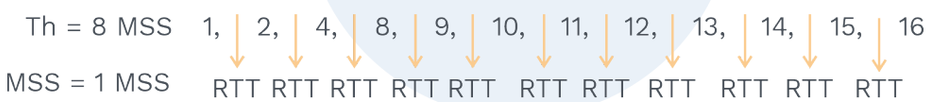


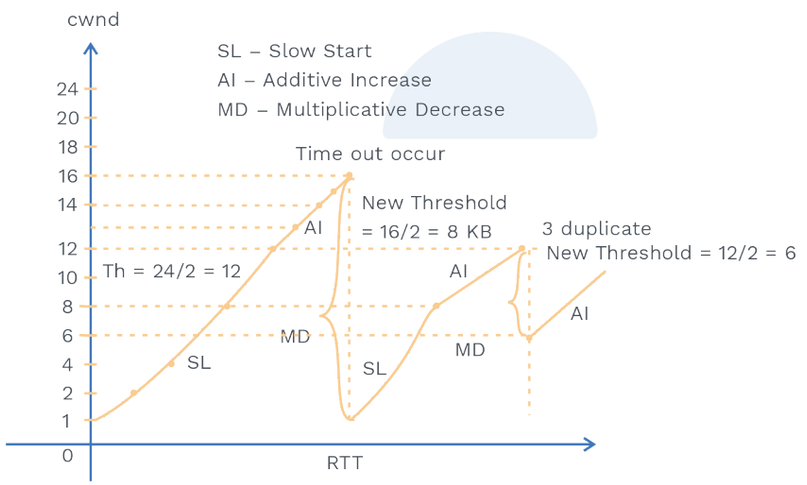
Q8 **Draw the graph to show how additive increase will behave if congestion window size when time out occurs is 24 KB, and Time out occurs at 16KB and after that show, what will happen when window size is 12KB and 3 acks event happens?**

Sol: When window size is 12 KB, and 3 ack event happens it will go into congestion avoidance phase.

Q9 **If congestion window size when time out occurs is 16 MSS, after how many RTT sender will reach 16 MSS?**

Sol: **after 11 RTT,**



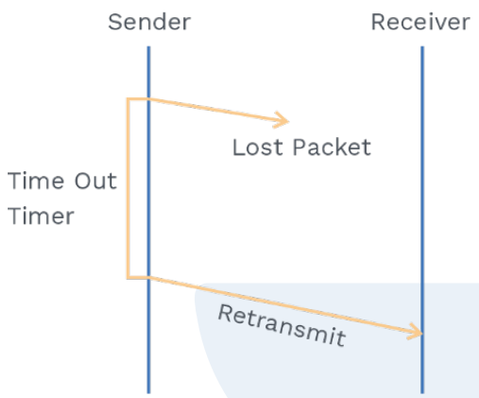


TCP timers:

There are types of timers,

- Retransmission timer
- Persistence timer
- Time wait timer
- Keep alive timer

Retransmission timer: It is basically used to retransmit lost segments. Sender starts the timer when it transmits the packet; the timer stops when the sender receives the acknowledgement.
Retransmission Timer is also called Time out timer.



How Time Out (TO) timer can be calculated.
In real scenario two cases may arise in which TO timer can be calculated,

1) High traffic (It may increase)
2) Low traffic (It may decrease)

There are 3 algorithm which can be used to calculate TO Dynamically.

1) Basic algorithm:

Lets see using example,
In TCP, the initial RTT is 12 msec. The acknowledgements for the first two segments are received in time 17 msec, 22 msec. Find the time out timer value for the first two segments using a basic algorithm. Use α = 0.6.
Here, α is called smoothing factor where $0 <= \alpha <= 1$
(value will be always given in questions)

For first segment:
Initial Round trip time (IRTT) = 12 msec.
Time out Timer (TO) = 2 * RTT = > 24msec.
Actual Round trip time (ARTT) for first segment = 17 msec.

For second segment:
Next round trip time (NRTT) =

$$NRTT = \propto IRTT + (1 - \propto) ARTT$$

$$= 0.6 * 12 + 0.4 * 17$$

= 14 msec

So, for second segment IRTT = 14msec

TO = 2 * 14 = 28msec

Actual Round trip time (ARTT) for second segment = 22 msec.
This is how basic algorithm computes Time out timer.

2) Jacobson's algorithm:

In TCP, the initial RTT is 12 msec, and the initial deviation is 7 msec. The acknowledgements for the first two segments are received in time 22 msec, 12 msec.
Find the time out timer value for the first two segments using Jacobson's Algorithm. Use α = 0.6.
For first segment,

IRTT = 12 msec

ID = 7 msec

TO = 4 * ID + IRTT

4 * 7 + 12

40 msec

ARTT = 22msec
Actual deviation = | ARTT - IRTT | = >| 22 - 12 | = 10msec

For second segment:

$$NRTT = \propto IRTT + (1- \propto) ARTT$$

$$= 0.6 * 12 + 0.4 * 22$$

= 16 msec

So for second segment, IRTT = 16msec

$$ND = \propto ID + (1- \propto) AD$$

$$= 0.6 * 7 + 0.4 * 10$$

= 8.2 msec

So for second segment, ID = 8.2msec

TO = 4 * ID + IRTT

4 * 8.2 + 16

48.8 msec

ARTT = 12msec
Actual deviation = | ARTT - IRTT | = >| 12 - 16 | = 4msec

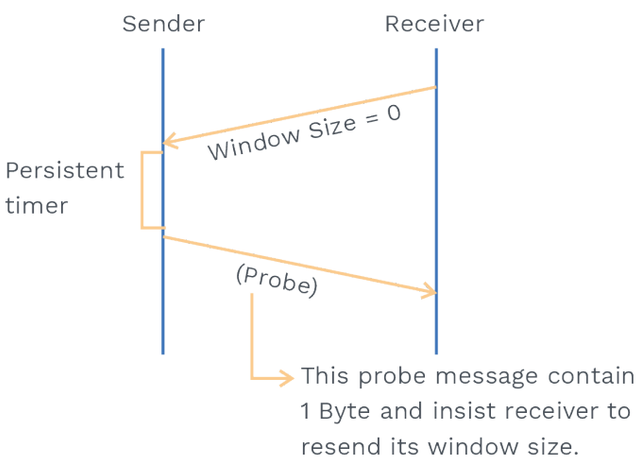
3) Karn's modification states:
Since actual RTT is not available, there is no need to find Time out timer we can double the time out timer (TOT) whenever the timer times out and make a retransmission.

Persistence timer:

- It is used to deal with a Zero window size situation.
- If the receivers announce that its window size is zero then the sender will stop transmitting and wait for an acknowledgement.

Note:
There is no retransmission if Ack segment is lost, and also the Ack segments are not acknowledged.

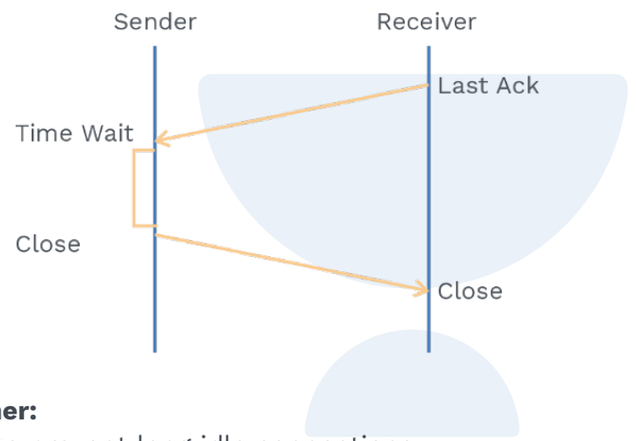
- There might be a case when the receiver has announced that it has a zero window size and because of this sender, was waiting and later when the receiver has gained a buffer for window and sent a segment to sender to update but this segment get lost now sender is still waiting for acknowledgement from the receiver, This is **deadlock state.**
- Inorder to overcome from deadlock situation, TCP uses persistence timer for each connection.



If the sender does not receive any message even after the probe message, it will double the persistent timer size.

Time wait timer:

- TCP uses a time wait timer during connection termination.
- Sender starts the time wait timer after sending the ACK for the second FIN segment, and It allows it to resend the final acknowledgement if it gets lost.
- What does it basically do? It prevents the just close segments to response it again to some other application.



Keep alive timer:

- It is used to prevent long idle connections.
- Client opens a TCP connection to a server, transfers some of the data, and becomes silent. There may be the possibility that the client has crashed. In this case, the connection remains open forever.
- In order to overcome this situation server will use Keep alive timer, so that it gets to know that whether client is down or not.

Improving quality of service using traffic shaping:

In traffic shaping two technique are used

1) Leaky bucket

2) Token bucket

Leaky bucket:

If the bucket has a small hole at the bottom, then the rate at which water flows out from the hole does not depend on how much rate water enters the bucket. It always flows out at a constant rate.

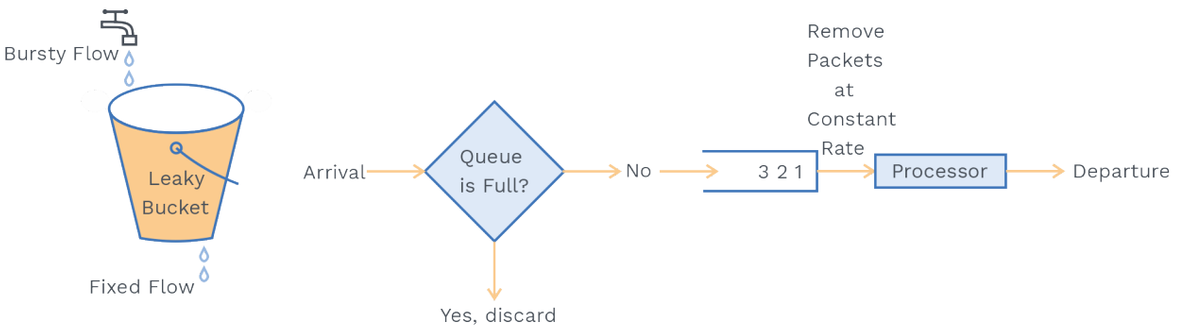


Fig. 5.6 Leaky Bucket Implementation

In a similar way, the leaky bucket algorithm can smooth the bursty traffic.

Token bucket:

- In the Leaky bucket, there is a chance when Host remains idle when there is no data to send.
- And when the host has some data then also it sends data at an average rate which affects the performance of system.
- The token bucket allows bursty traffic at a regulated maximum rate.

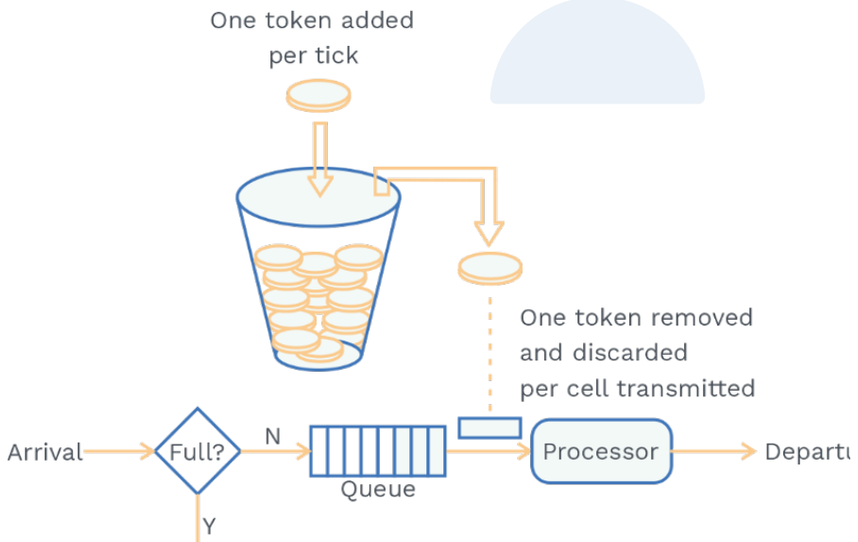


Fig. 5.7 Diagrammatic Representation of Token Bucket

Assume that there are 50 tokens, and the Host removes one token for every byte of data sent. The Host was idle for 50 units. In that time, the bucket collected 2500 Tokens, Now If the Host wants to consumes all these Token in one unit of time, then the rate should be 2500 byte/unit.

Chapter summary

- The objective of the Transport layer is to deliver the packet from process to process.
- Socket address is a combination of IP address and port address.
- Transport layer has two major protocol TCP and UDP.
- TCP Keep counts its segment Byte Number, Sequence Number and Acknowledgement Number.
- TCP count all the data bytes that needs to be transmitted.
- TCP has 3 phases for connection-oriented services:
  Connection Establishment
  Data transfer
  Connection termination
- In any TCP segment:
  If SYN bit = 1 and ACK bit = 0 , then it is request segment.
  If SYN bit = 1 and ACK bit = 1 , then it is reply segment.
  If SYN bit = 0 and ACK bit = 1 , then it is pure acknowledgement or data segment.
  If SYN = 0 and ACK = 0, then this is not possible.
- A SYN + ACK segment cannot carry data, but does consume one sequence number.
- A SYN segment cannot carry data, but it consumes one sequence number.
- If the server wants to close a connection, then it will sent a FIN segment.
- TCP has 3 main tools for detecting and correcting errors:
  Checksum, acknowledgement and time out.
- **Congestion:** It refers to the state of a system which slows down the network performance due to heavy traffic and we can't avoid congestion completely.
- Basically there are 2 cases in which the sender retransmits the packet:
  Either timeout occurs or
  3 duplicates acknowledgements come back to sender
- **UDP:** It is an unreliable, and connectionless protocol.
- **TCP Timers:**
  Retransmission timer
  Persistence timer
  Time wait timer
  Keep alive timer