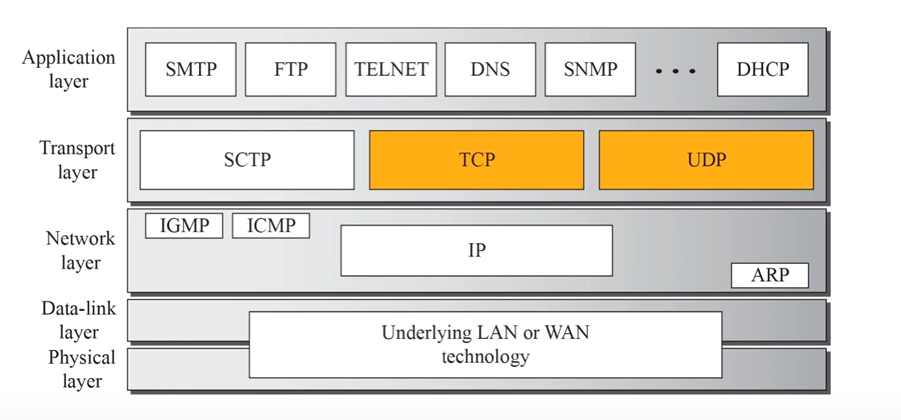
A1 wants to communicate with B1

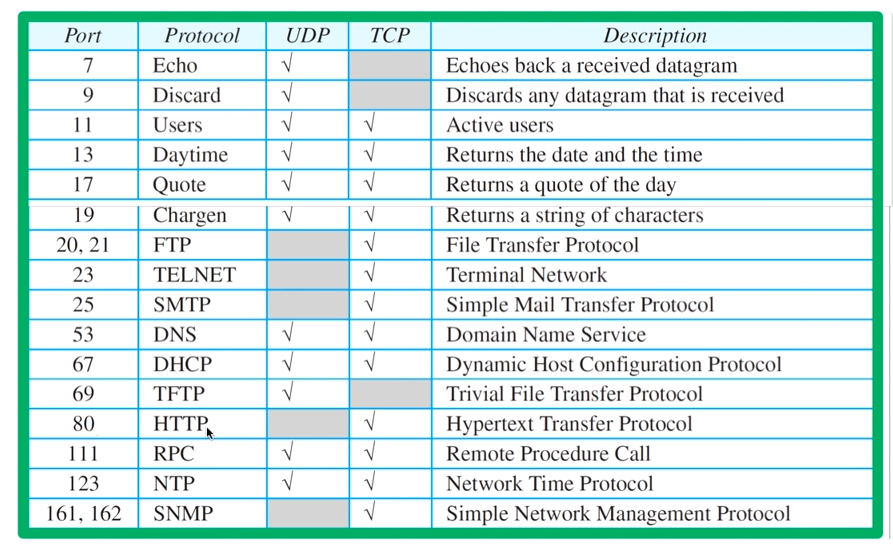
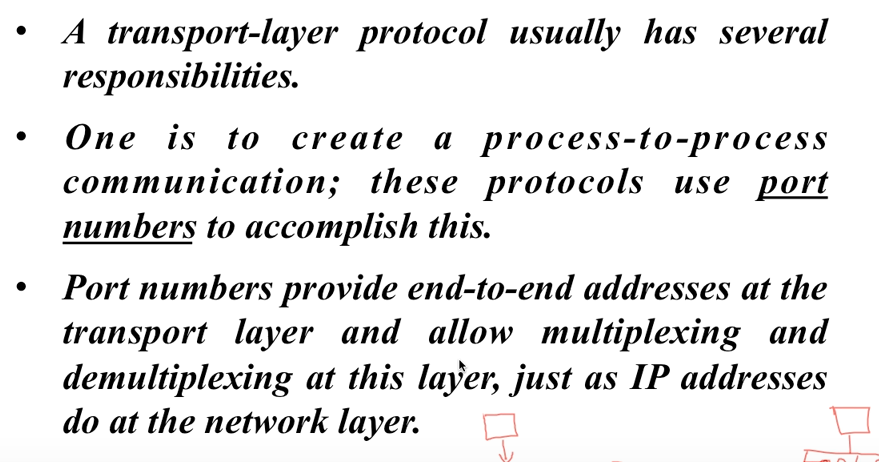


# Responsibilities of the data-link layer

# Responsibilities of the network layer

# Responsibilities of the transport layer

1. (End to end / Port to Port / Process to Process) delivery.
2. Reliability / In-order / No Loss of data.
3. Error Control (Checksum method)
4. Flow Control
5. Congestion Control
6. Multiplexing and Demultiplexing

**End-to-End**

**Reliability** 🡪 In Network Layer with IP addressing I am sending a message from one ip to another ip without any guarantee.  
If I send message With guarantee 🡪 TCP  
**In-order** 🡪 If I am sending 4 packets, then these 4 packets should be received exactly in the same order.

**Network Layer**  
In postal service, we will send a post and we are not at-all sure that the post card will reach the exact receiver. There may be loss of postal card / will not reach the exact destination.

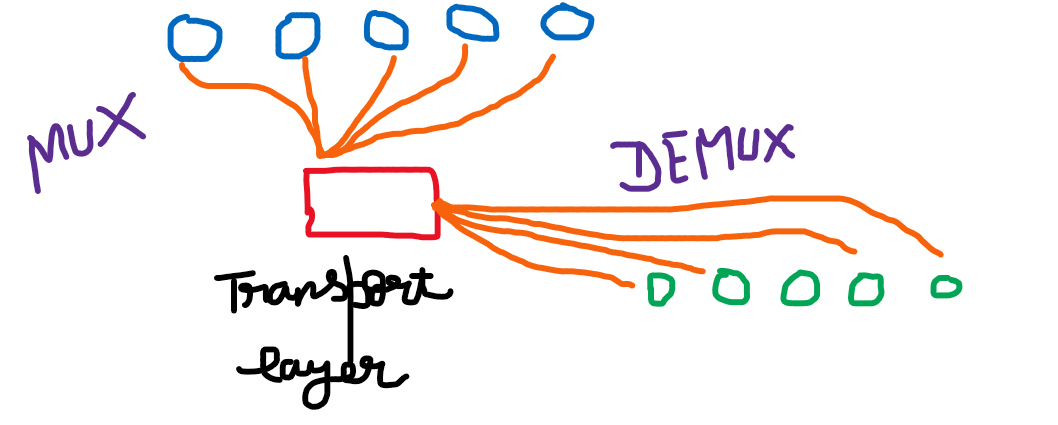
**Transport Layer**  
In telephone, first we will make sure the connection between two persons, and then we will start talking. So no loss of data. This is done with the help of TCP and UDP.

**Error Control**data link layer 🡪 node to node  
transport layer 🡪 A to B

**Flow control (Stop and wait, go back and arq, selective repeat request)**

Window(capacity) 🡪 Size of a single message.  
How many messages ??  
Speed of sending.

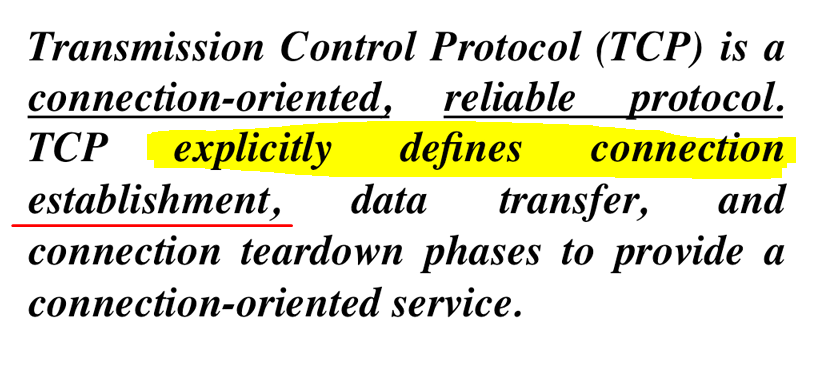
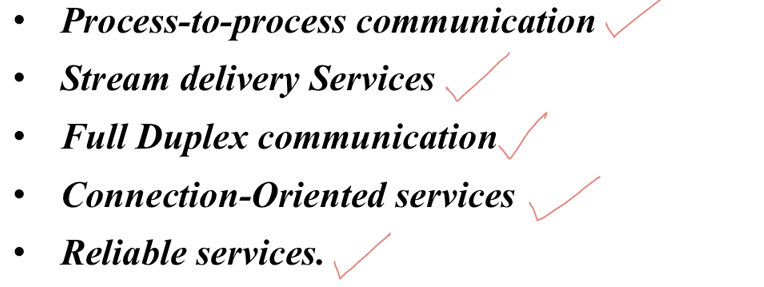
**Congestion control**  
If there is no flow control, congestion control happens.

**Multiplexing and Demultiplexing**

# Segmentation

Transport layer receives continuous bits from the application, presentation and session layer. These bits are separated into segments and then transferred to the network layer.

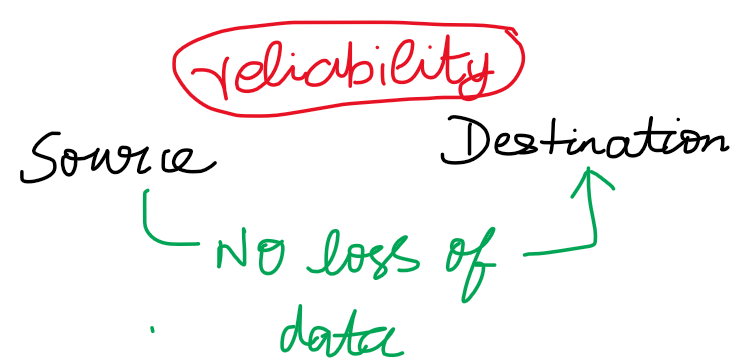
# **TCP Transfer Control Protocol (Connection oriented)**

  
After completion of the data-transfer the connection is terminated.  


* Byte Streaming
* Connection Oriented
* Full Duplex
* Piggybacking
* Error Control
* Flow Control
* Congestion Control

## **Byte Streaming**

## **Connection Oriented**



* Reliability must be ensured.
* If any loss of data, the entire data must be re-transmitted.
* 3-way handshaking protocol 🡪 to establish the connection.

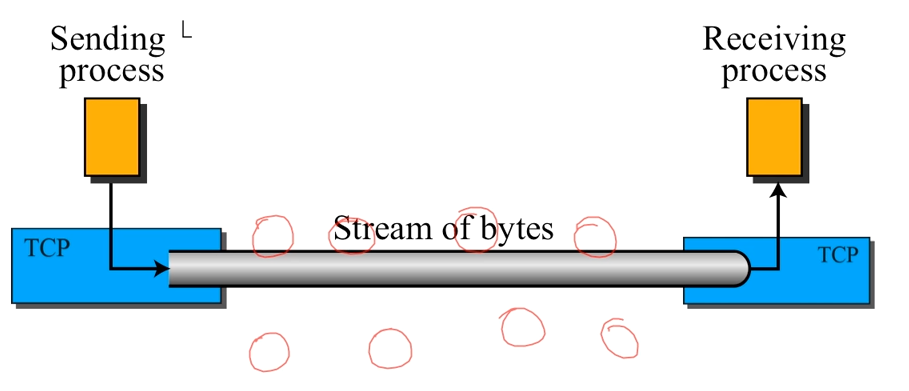
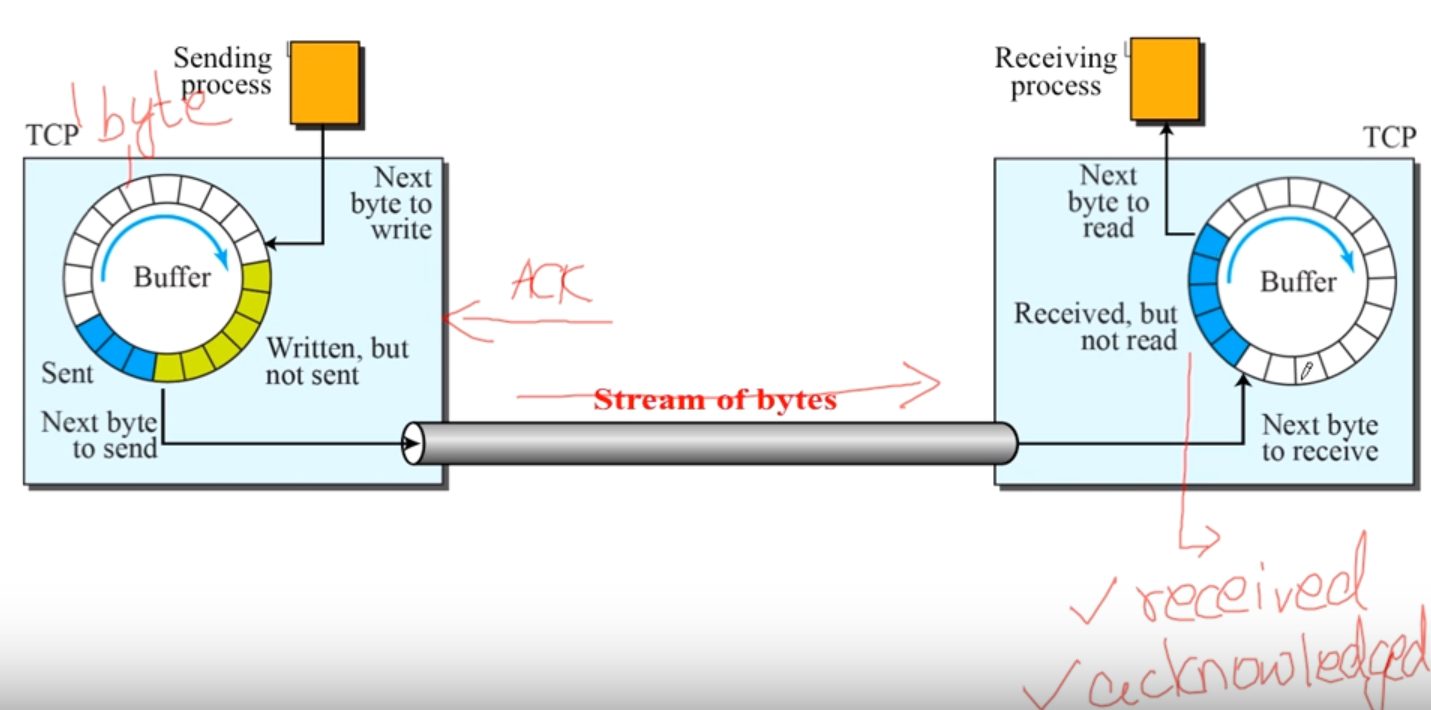
## **Full Duplex**

## **Piggybacking**

## **Error Control**

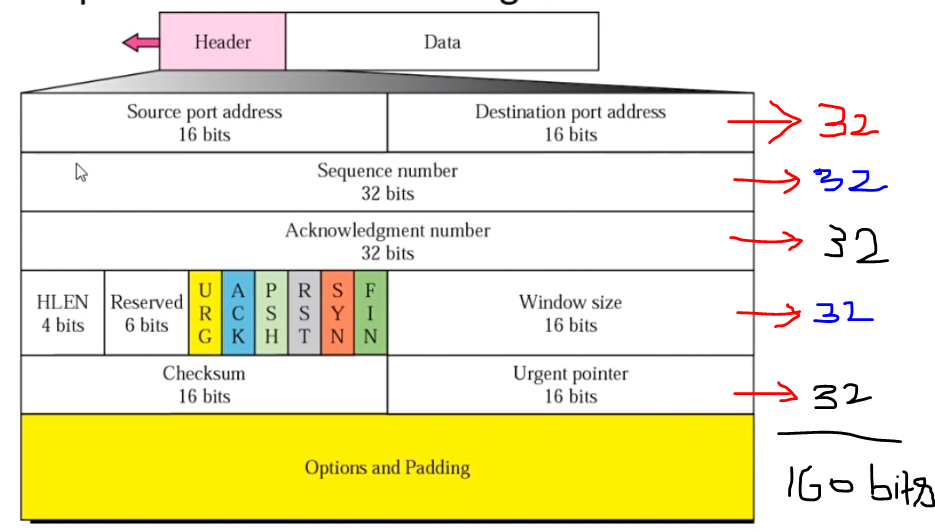
## **Flow Control Congestion Control**

# **Sending and Receiving buffers**

  
  
**Sender**White 🡪 bytes yet to be written.  
Green 🡪 bytes written and they are waiting for being sent.  
Blue 🡪 already sent. Why are they still residing in the buffer??  
Since they didn’t receive any acknowledgment. As soon as they receive the positive acknowledgement they will be removed from the buffer and gives the space the next byte to write (white blanks)  
**Receiver**Blue 🡪 These byte are received and acknowledged but not read.  
As soon as it is read by the receiver, these blue ones will be changed into white.

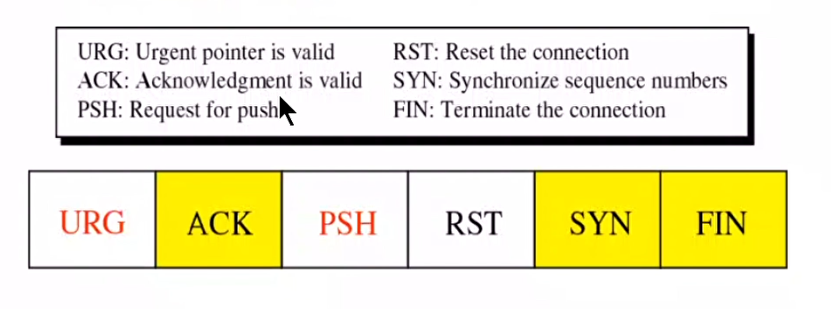
# **Numbering Bytes**

# **TCP Header structure**



Header 🡪 20 to 60 bytes.  
Header is going to guide the packet/segment/data/message to reach the desired destination.  
If we want to pass any segment, (32\*5 = 160 bits) are required.

Eg: Sending a message “Hi”  
Head(160 bits) + Data (“Hi” 🡪 16 bits) = 176 bits  
Totally 176 bits are required to send “Hi” message.   
176 bits are wasted, but there is no other way of sending the message without the header.  
In this case of small messages, UDP can be used.



Effectively controls the flow of segment from the sender to the receiver.

ACK,SYN and FIN 🡪 3 way

## **Source Port**

Defines the port number of the application program in the host that is **sending** the segment.

## **Destination Port**

Defines the port number of the application program in the host that is **receiving** the segment.

## **Sequence Number**

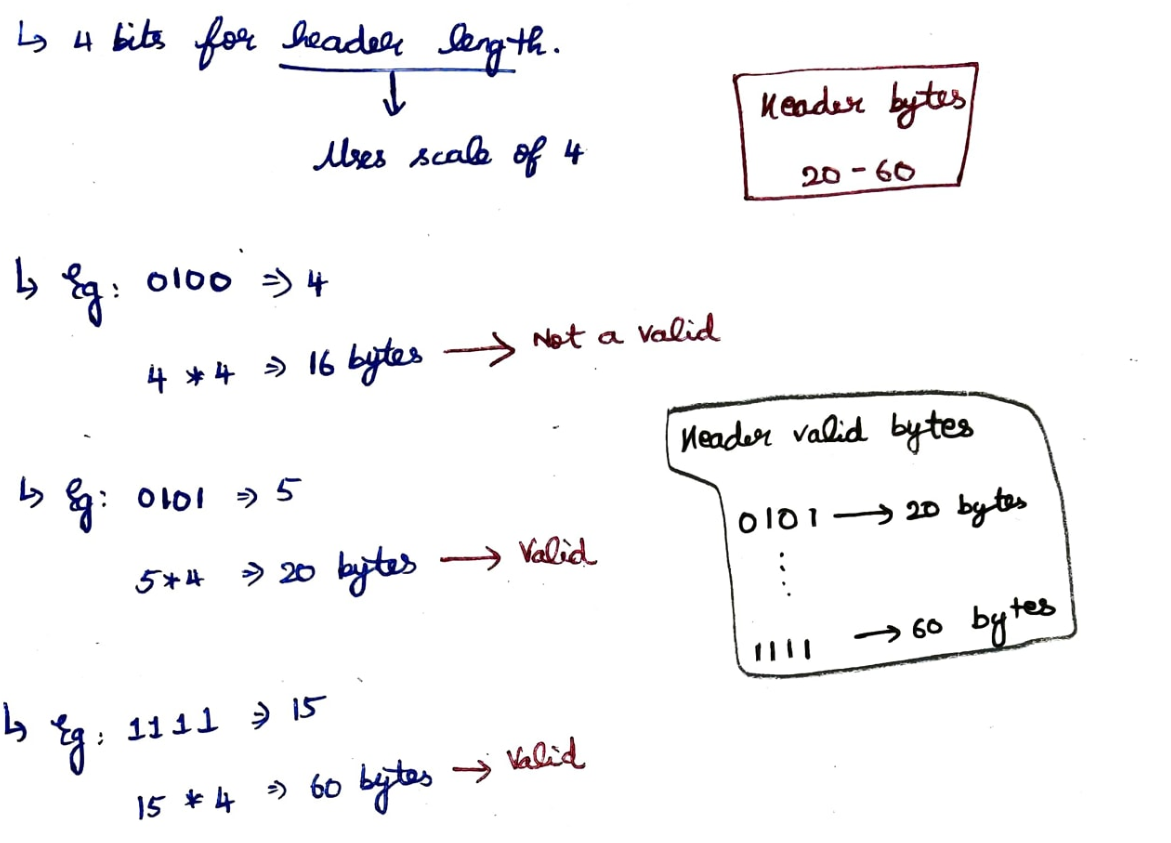
Defines the number assigned to the first byte of data contained in the segment.  
During the connection establishment, each sender uses a random number generator to create an Initial Sequence Number (ISN).

## **Acknowledgement Number**

If the source of the segment has successfully received the byte number **x** from the other sender, it defines the **x+1** as the acknowledgement number.

16 bits 🡪 2^16 = 65535 bits. (0 to 65535)  
Only 0 to 1023 are the well known address.

## **Header length (HLEN)**



* Will tell how many fields are there in the header.
* Indicating the number of 4-byte words in the TCP header.
* Based on the header length only the bits for **Options and Padding**
* 20 bytes as the header length, options and padding won’t be there.

## **Reserved Bits**

* 6 bits are used for future use.
* URG flag 🡪 Urgent Pointer (1-urgent , 0-not urgent )
* ACK flag 🡪
* PSH flag 🡪
* RST flag 🡪 reset the flag
* SYN flag 🡪 synchronization the flag
* FIN flag 🡪 finish

What is the difference between acknowledgment number and acknowledgment bit(i.e flag) ???  
Acknowledgment flags are just notifying that this segment is acknowledged.  
Acknowledgment number is the acknowledged number for the previous segment.

## **Window Size**

* Both the sender and the receiver defines the size of the window.
* Maximum window size 🡪 16 bits (0 to 2^16 -1) = 65,535 bytes.

## **Checksum**

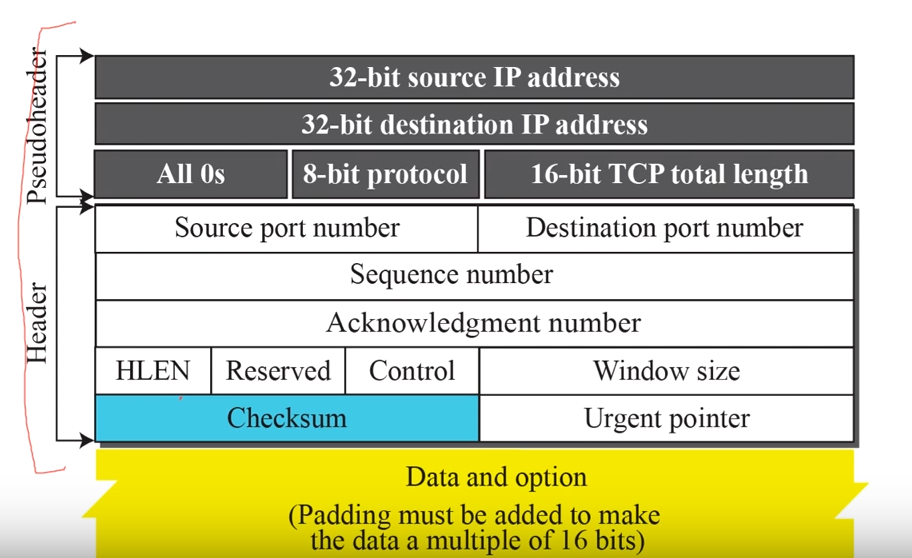
* Used in error control mechanism.
* Maximum checksum size 🡪 16 bits (0 to 2^16 -1) = 65,535 bytes.

## **Urgent pointer**

* Maximum urgent pointer size 🡪 16 bits (0 to 2^16 -1) = 65,535 bytes.
* Used when the segments contain the urgent data.
* Defines the number that must be added to the sequence number to obtain the number of the last urgent byte in the data section of the segment.
* Eg: text message and video call.   
  urgent pointer for video call (synchronise both video and audio signal) to have a smooth experience. And then text message can be sent since there is no sync for text message.

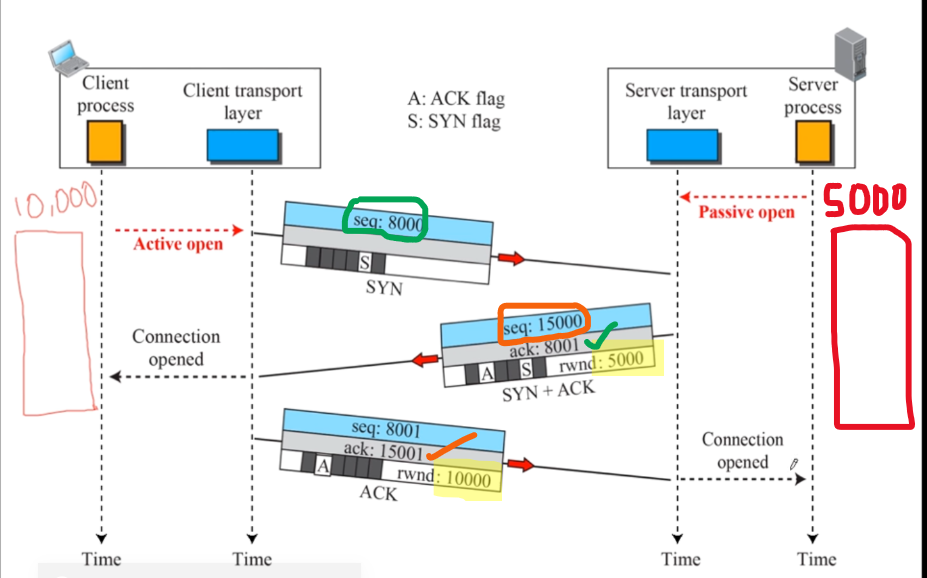
## **Options and Padding**

# **Pseudo header added to the TCP datagram**



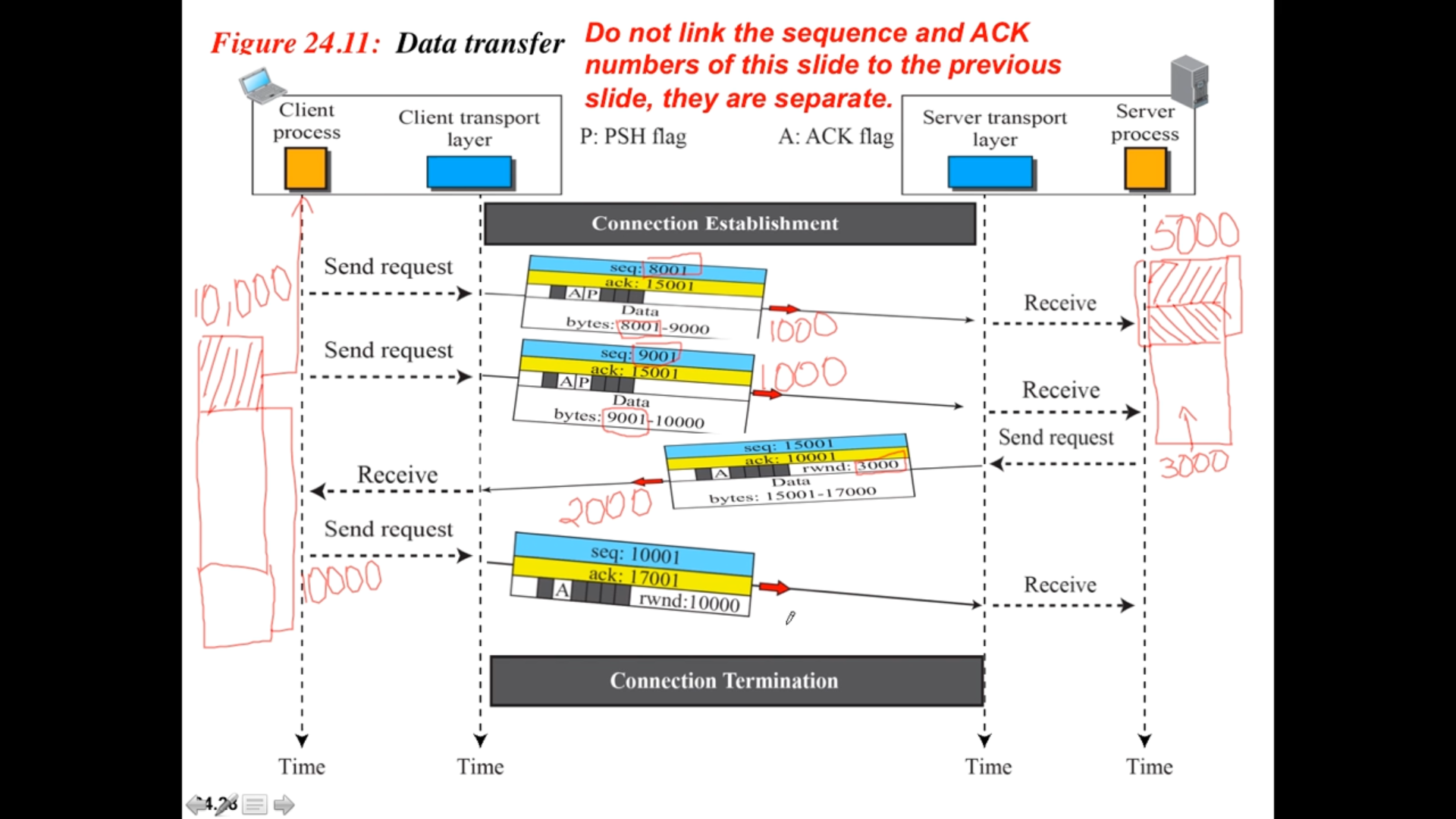
# **Connection Establishment using three-way handshaking**

* Set the connection and then start transfer the data. But in UDP, there is no connection required.
* Three packets transferred



1. Server will make the **passive open** ( server is ready to accept request from the client)
2. Client is ready to send the segments to the server (**Active open**)
3. Client sent (seq:8000) I am going to send the packet starting from the 8000th sequence number.
4. Server acknowledged (ack:8001) and (rwnd:5000) [ i.e Client can send up-to 5000 bytes to the server without any acknowledgement] and the **connection is opened** for the client.   
   Server sent (seq:15000) I am going to send the packet starting from the 15000th sequence number.
5. Client acknowledged (seq:8001) and (rwnd:10000) [ i.e Server can send up-to 10000 bytes to the client without any acknowledgement] and the **connection is opened** for the server.
6. Both the connections are opened, now the client and server both can send the packets/data to each-other.

# **Data Transfer phase**

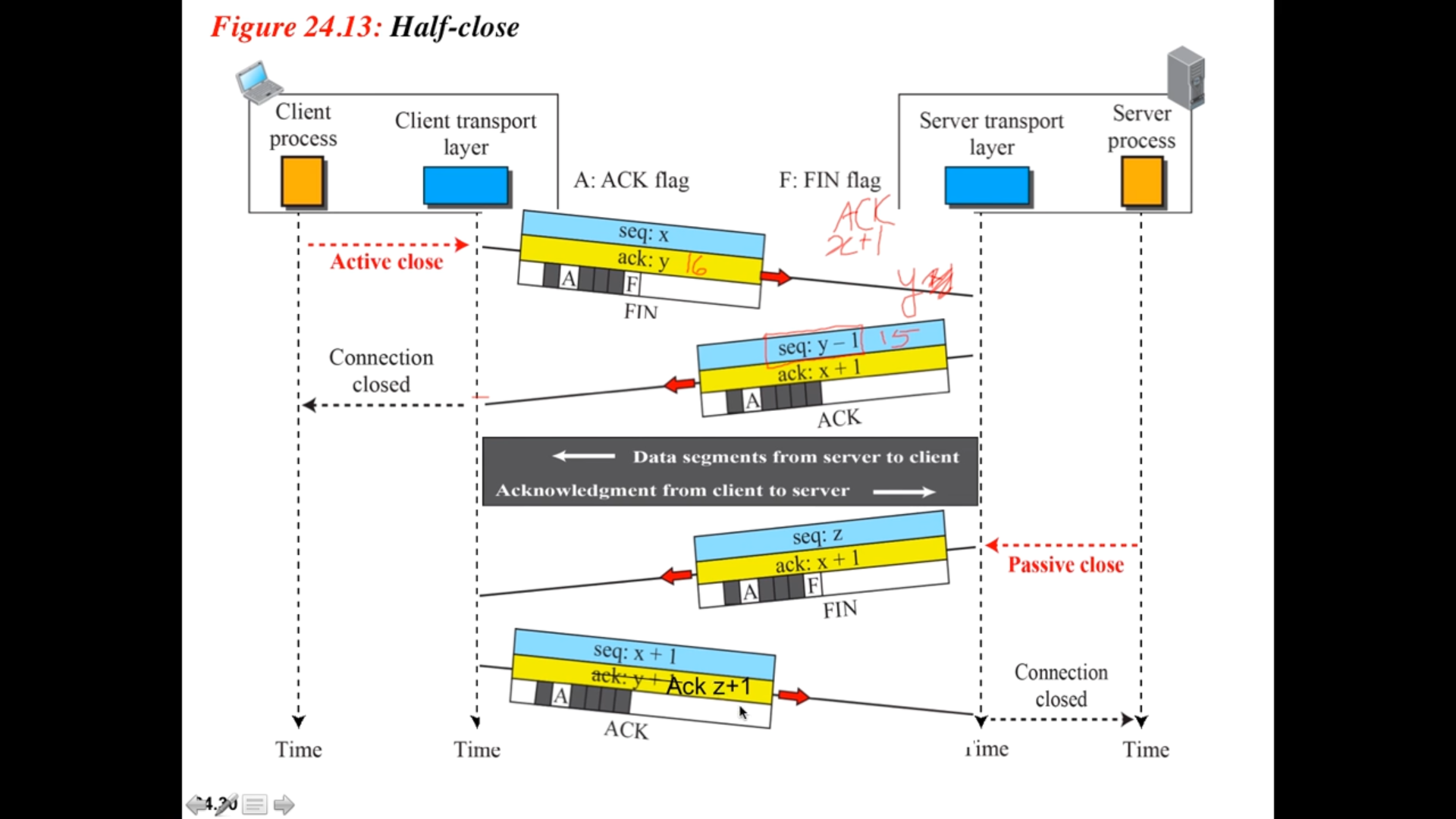


1. (seq:8001 and ack:15001) I want to send the data 1000 bytes (8001 – 9000).
2. (seq:9001 and ack:15001) I want to send another data of 1000 bytes (9001 – 10000).
3. (seq:15001 and ack:10001) I have received the data successfully from [ (8001 – 9000) and (9001-10000) ] bytes 🡪**collective acknowledgement**rwnd:3000 🡪 Only 3000 bytes are remaining in my buffer  
   data : 15001 – 17000.
4. (seq:10001 and ack:17001)   
   rwnd : 10000 ( Now I am having 10000 bytes free in the buffer (i.e the previously added items to the buffer (2000 bytes) have been pulled by the client and frees the buffer) Now the server can send 10000 bytes of data.

# **Connection Termination Phase (Full Close)**

1. (seq:x and ack:y) FIN flag to indicate that client has finished its task with the server. **Active close**
2. (seq:y and ack:x+1) FIN + ACK (acknowledgment for the client’s FIN) **Passive close,** for this client I am closing the connection and going to serve for another client. **Connection closed (for the client)**
3. (seq : x+1 and ack : y+1) Thank you very much for your server.

# **Connection Termination Phase (Half Close)**



# UDP (Connection-less)