Transport Layer

- between network layer and application layer
- responsible for providing services to appn layer and receiving from network layer
- provides **process-to-process communication**

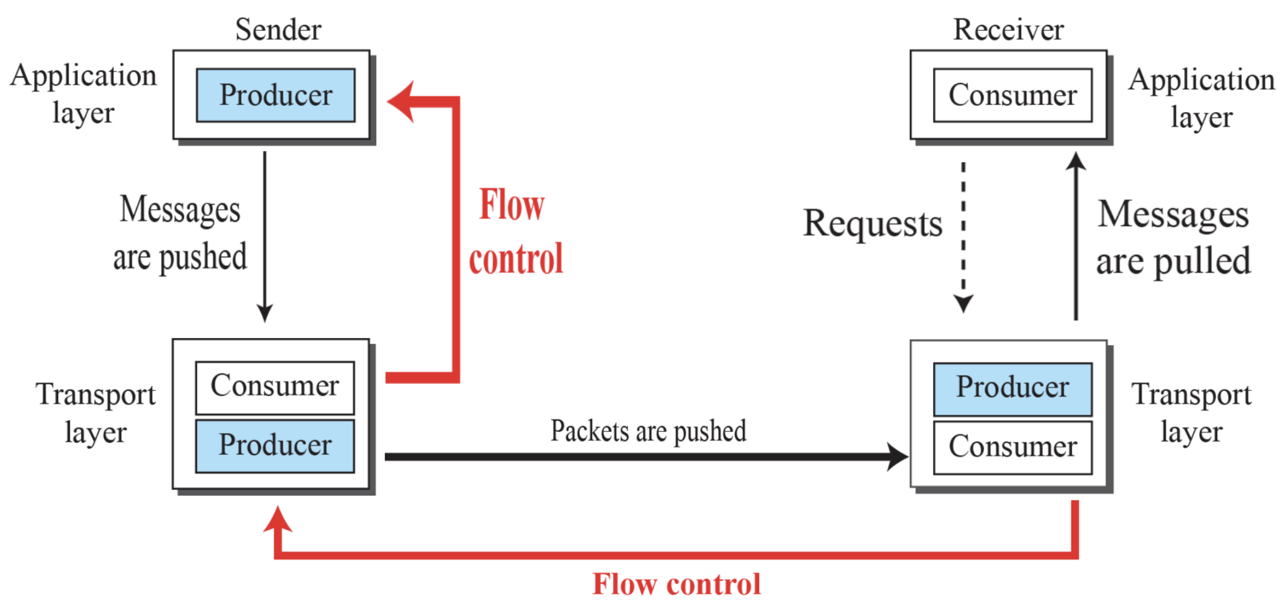
Addressing

- Just like hosts are identified by IP addresses, process inside them are identified by *Port Numbers*.
- Ranging from 0 to 65,535
    - 0 to 1023: well known (assigned and controlled by ICANN)
    - 1024 to 49151: registered (not assigned or controlled but can be registered)
    - 49152 to 65535: dynamic/private (neither assigned nor registered)

`Socket Address: [IP address, Port number]`

Flow Control

1. Pushing: Producer delivers to consumer then flow control done by consumer
2. Pulling: Consumer requests first and then Producer provides the message to consumer



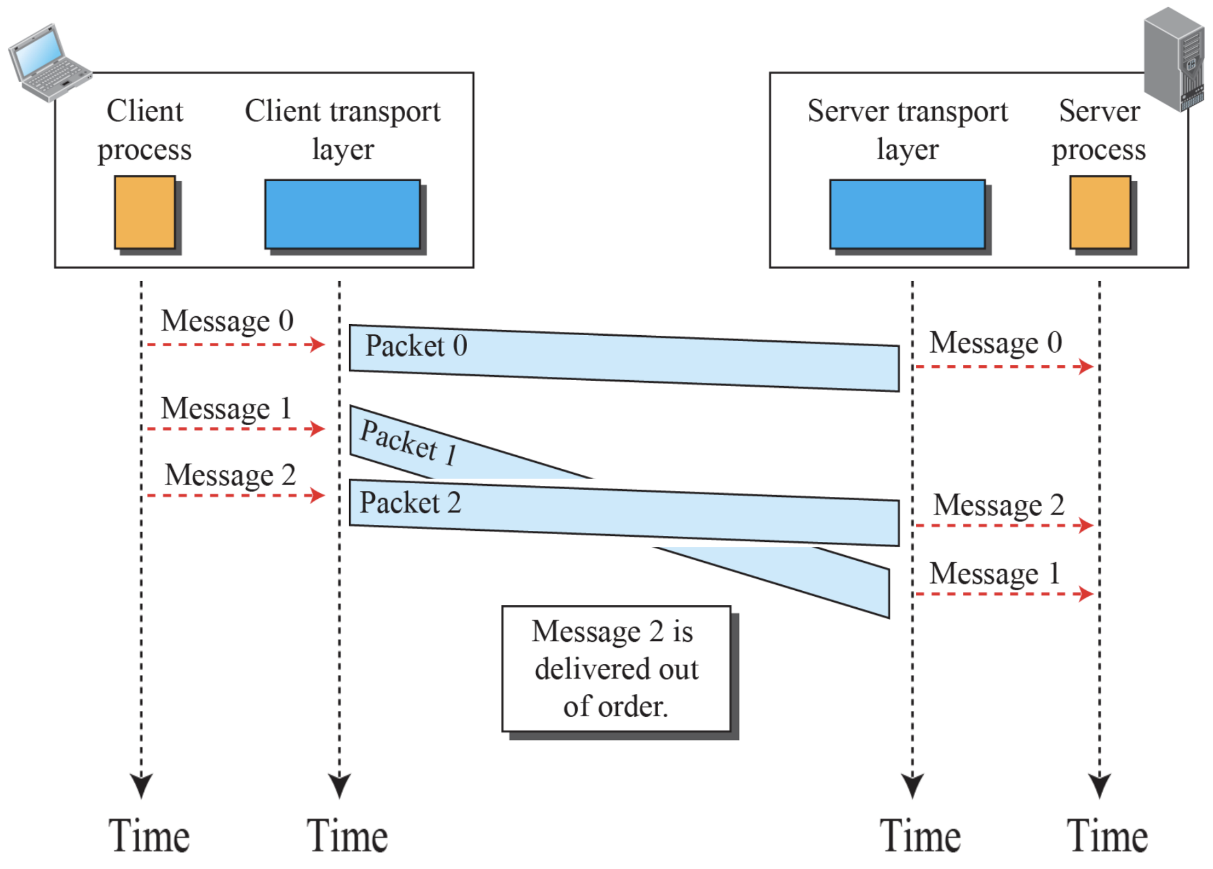
```
Note: Messages are transfered from process inside app layer to
transport layer and then transport layer encapsulates them into
packets that are transferred over logical channel.
```

Protocol types in Transport Layer

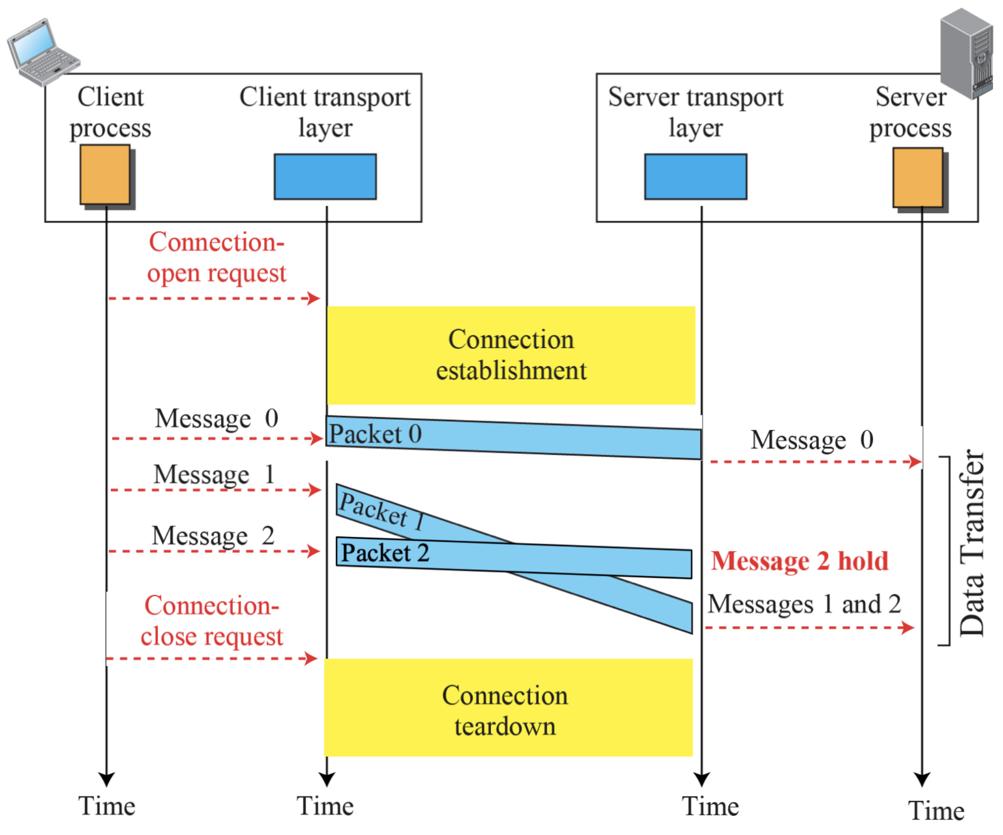
Transport layer provides two types of services:

1. Connectionless: the connection is always in established state between the two hosts.
   theres no acknowledgements fir received messages, nothing.
   but they are fast
   packets are lost then packets are lost
   ex: UDP
2. Connection-oriented: the connection has to be established explicitly
   there are acknlowdgements from reciever for everything
   compartitively slower, but retransmission of lost packets can be done
   ex: TCP

Connectionless service



Connection-oriented service



UDP

USER DATAGRAM PROTOCOL

- UDP is a connectionless, unreliable transport protocol
- There is no flow control mechanism
- There is no acknowledgment for received packets
- UDP, however, does provide error control to some extent
- If UDP detects an error in the received packet, it silently drops it

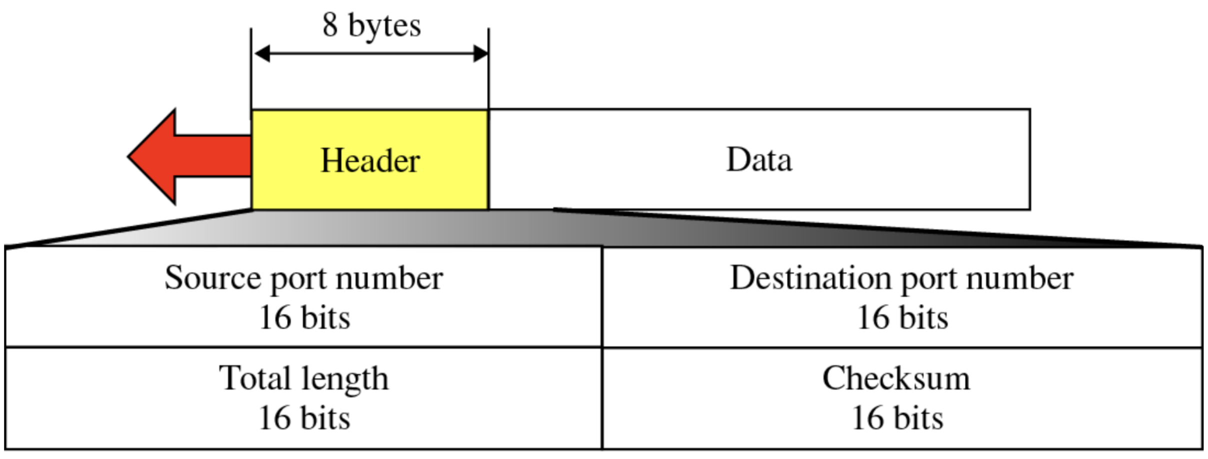
Why even use it then?

Glad you asked!

- 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
- Sending a small message using UDP takes much less interaction between the sender and receiver than using TCP

UDP Packets

...are called *User Datagrams* and they have a fixed header of **8 bytes**.



`UDP length = IP length - IP header length`

`UDP Data length = Total UDP length - 8 bytes (header length)`

UDP Services

1. Process to Process communication: UDP can help communicate between hosts' processes using socket addr (IP + Port)

2. Connectionless service: each user datagram is independently transmitted, there's no ack from receiver and no connection establishment takes place (unlike TCP)
3. No flow or error control: there's no flow control for datagrams, and only error control is done through checksum.
4. No congestion control: UDP assumes that the packets sent are small and sporadic, and cannot create congestion in the network.

TCP

TRANSPORT CONTROL PROTOCOL

- It is a **connection-oriented** protocol for communications that helps in the exchange of messages between the different devices over a network.
- The way it works is that TCP divides the message to be transmitted into several smaller packets and then uses IP to transfer them over the internet. They are then reassembled at the destination and ack for each segment is sent.
- Every TCP header is of fixed format size of **20 bytes**.
  TCP headers contain the following imp things:
  - Source port: used by receiver to reply
  - Destination port: to send to right place
  - Sequence number: to identify the sequence of the transferred segments
  - Ack number: this field contains the next number that the receiver expects to receive.
  - Checksum: used for error detection
  - **if host1 sends a SEQ of number x then the host 2 returns and ACK of x+1**.

TCP Connection Establishment

- uses a 3-way handshake to establish connection between client and server.
- first step is initialisation, then comes acceptance and third is data transfer
  Steps in detail:

1. First the server sends the client a SYN request describing the data and the rate at which that data will be received.

2. Another SYN is sent from client to server saying it's ready to receive the packets
3. Server sends the data

TCP Window Management

- the receiver may have a buffer storage in which it fills the data received by the sender. now the sender may send data of size that is smaller than the receiver's buffer size.
- In that case the buffer is filled with the data, and then in ack the sender is notified of the space left in the buffer
- once the buffer is full, receiver sends a request to sender asking to not send any more data as the buffer is full.
- after some data in the buffer is utilised another request asking for more data is sent