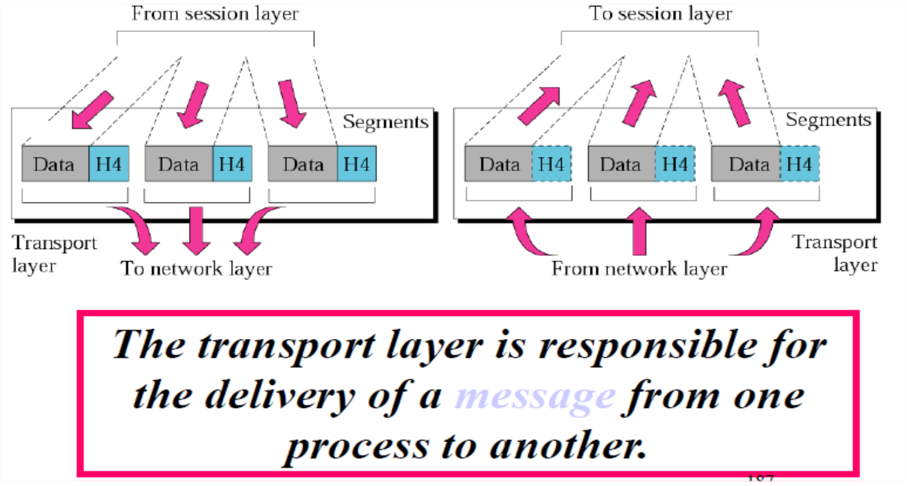
Transport services and protocols

Transport Layer Functions

The main functions of transport layer are:

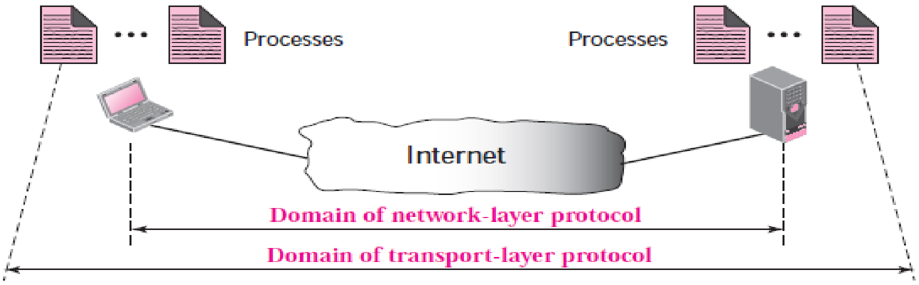
- Provides reliable data delivery (it's the TCP in TCP/IP).
- Receives data from upper layers and segments it into packets.
- Can provide error detection and correction.



The transport layer is responsible for the delivery of a message from one process to another.

The **Transport Layer** provide ***logical communication*** **between application processes** running on different hosts, Transport protocols run in the **end systems**

Network layer*:* logical communication **between hosts**
Transport layer*:* logical communication **between processes**


Processes
Processes
Internet
Domain of network-layer protocol
Domain of transport-layer protocol

<u>The Transport layer protocols:</u>

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: (UDP)
  - no-frills extension of "best-effort" IP

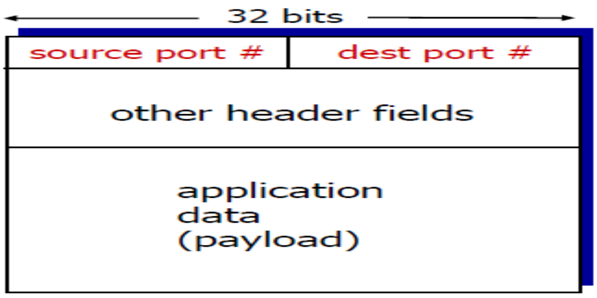
.

Services not available in transport layer protocol

- delay guarantees
- bandwidth guarantees

How Demultiplexing works

1. host receives IP datagrams
   - each datagram has source IP address, destination IP address
   - each datagram carries one transport-layer segment
   - each segment has source, destination port number
2. host uses **IP addresses & port numbers to direct segment to appropriate socket**
3. **There are two types of demultiplexing: Connectionless and Connection demultiplexing**



TCP/UDP segment format

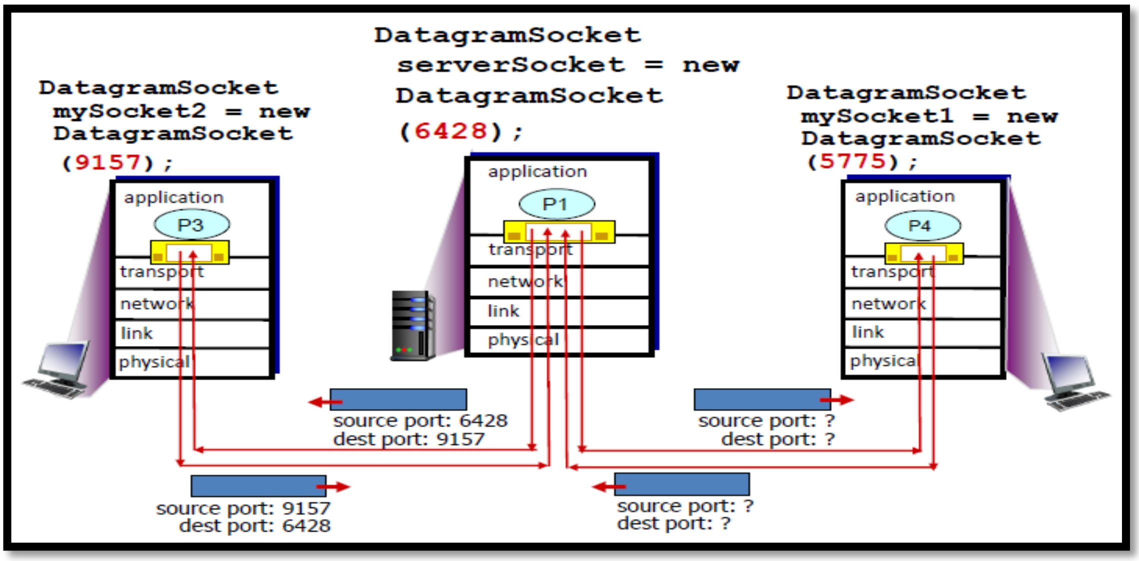
Connectionless Demultiplexing

1. **When creating datagram to send** into **UDP socket**, must specify
   - destination IP address
   - destination port #

2. **When host receives UDP segment:**
   - checks destination port # in segment
   - directs UDP segment to socket with that port #

3. IP datagrams with same dest. port #, but different source IP addresses and/or source port numbers will be directed to same socket at dest

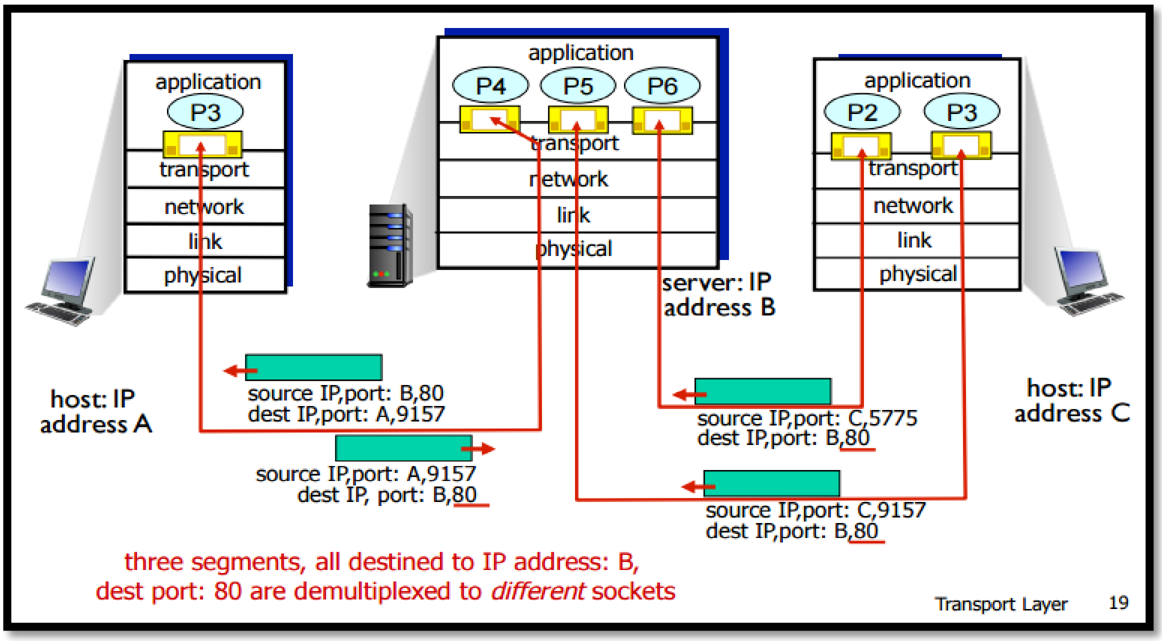
Connectionless Demultiplexing: example


DatagramSocket serverSocket = new DatagramSocket (6428);
DatagramSocket mySocket2 = new DatagramSocket (9157);
DatagramSocket mySocket1 = new DatagramSocket (5775);
application
P1
P3
P4
transport
network
link
physical
source port: 6428
dest port: 9157
source port: ?
dest port: ?
source port: 9157
dest port: 6428
source port: ?
dest port: ?

Connection-Oriented Demultiplexing

1. **TCP socket** identified by 4-tuple:
   - source IP address
   - source port number
   - destination IP address
   - destination port number
2. Demultiplexing: **receiver** uses all four values to **direct segment to appropriate socket**
3. **Server host** may support **many** simultaneous TCP sockets(each socket identified by its own 4-tuple)
4. Web servers have **different sockets** for each connecting client

Connection-oriented Demultiplexing: example



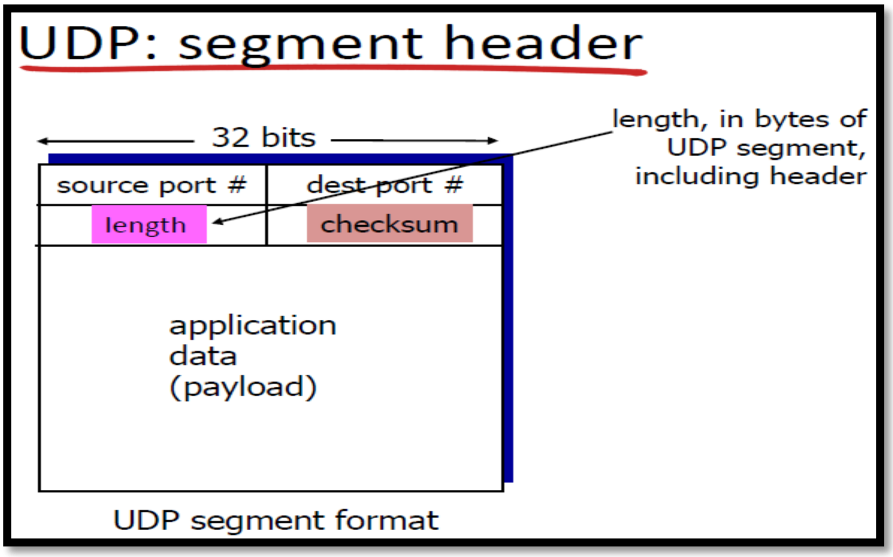
User Datagram Protocol (UDP)

User Datagram Protocol (UDP) is a transport layer protocol has the following features:

1. UDP is an **Internet transport protocol**.
2. There is **no flow control**.
3. **Fast data delivery**
4. **Unreliable service (**Best Effort**)** this is because UDP segments may be:
   - Lost
   - Delivered out-of-order to application (**datagrams are not numbered**)
5. **Connectionless:**
   - **No handshaking** between UDP sender, receiver
   - user datagrams sent by UDP is an **independent datagrams**
   - There is **no connection establishment and no connection termination**, this means that each user datagram **can travel on a different path**.
6. **Reliable transfer over UDP:**
   - Add reliability at application layer
   - Application-specific error recovery!

Q/ why is there a UDP?

- **No connection establishment** (which can add delay)
- **simple**: no connection state at sender, receiver
- **small header size**
- **No congestion control**



Using of UDP

The following lists some uses of the UDP protocol:

- streaming **multimedia apps** (loss tolerant, rate sensitive)
- Routing Information Protocol **(RIP)**
- Domain Name System **(DNS)**
- **SNMP**

Transmission Control Protocol (TCP)

1. **reliable,** in-order delivery (congestion control, flow control, connection setup)
2. TCP, **like** UDP, is a **process-to-process** (program-to-program) protocol. TCP, therefore, like UDP, **uses port numbers**.
3. **Unlike** UDP, TCP is a **connection oriented** protocol; it creates a virtual connection between two TCPs to send data.
4. In addition, TCP **uses flow and error control mechanisms** at the transport level.
5. **Reliable delivery**: no packet loss, error, duplication, disorder

Most of the user application protocols, such as **Telnet, SMTP, HTTP and FTP**, use TCP.

Comparison between UDP and TCP

UDP	TCP
1. **Connectionless** service, UDP datagrams are delivered independently.	**1.** **Connection-oriented**: connection establishment & termination
2. **Unreliable** delivery: packet loss, corruption, duplication, disorder.	**2.** **Reliable** delivery: no loss, no error, no duplication, no disorder.
3. **fast** as compared with TCP	**3.** **slow**: retransmission, flow control
4. No sequencing	**4.** Segment sequencing
5. No acknowledge	**5.** Acknowledge segment
6. Simple request-response communication without internal flow and error control	**6.** Connection overhead
7. RIP, DNS use UDP	**7.** **Telnet, SMTP,HTTP and FTP**, use TCP

Why Flow and Error Control

For **reliable** and **efficient** data communication a great deal of coordination is necessary between two machines. Some of these are necessary because both sender and receiver have:

1. Limited speed. (Receive, send process data).
2. Limited memory (storage) .

Requirements:

1. A fast sender should not **overwhelm** a slow receiver, which must perform a certain **amount of processing** before passing the data on to the higher-level software.
2. **If error occur** during transmission, it is necessary to create mechanism to correct it.

Flow Control

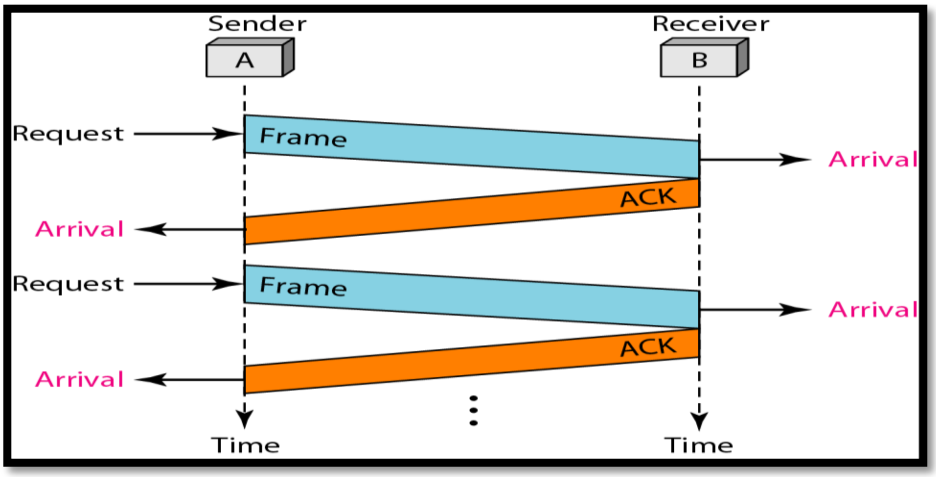
- Flow control is a **technique for speed-matching of transmitter and receiver**. Flow control ensures that a transmitting station **does not overflow** a receiving station with data.
- To control the flow of data, the receiver needs to send some **feedback** to the sender to inform the latter it is overwhelmed with data.
- **ACK** is a packet sent by one host in response to a packet it has received **Time out.**
- **Propagation delay** is defined as delay between transmission and receipt.
- **Propagation time** can be used to estimate time out period.

Flow Control Methods

There are two protocols developed for flow control, these are:

1. Stop-and-Wait

- This is the **simplest** form of flow control.
- After receiving the frame, the receiver indicates its willingness to accept another frame by sending back an **ACK frame** acknowledging the frame just received.
- The sender must **wait** until it receives the ACK frame **before** sending the next data frame



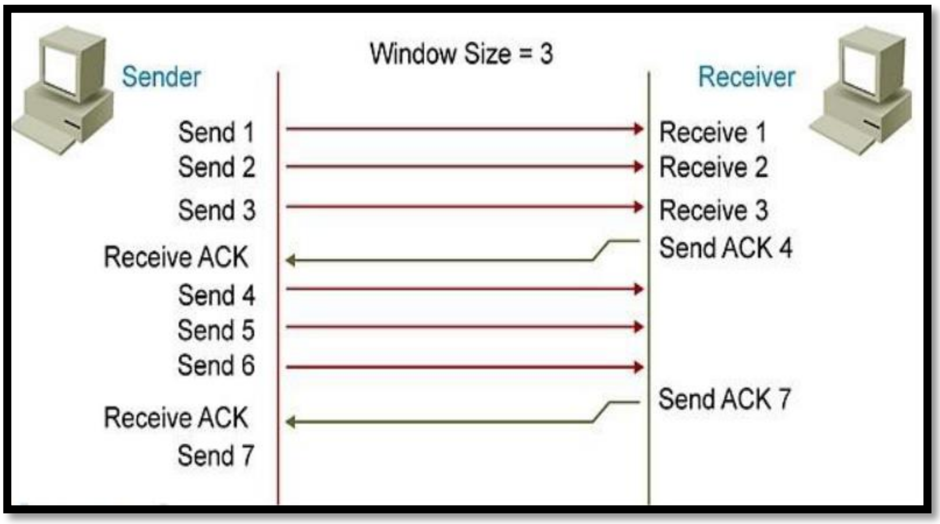
Disadvantage of Stop and Wait

- At any time there is only **one frame** that sent and waiting to be ACK1
- This is not good for **transmission media**.
- To improve efficiency **multiple frames should be transmit** while send is wait to one ACK this called sliding window

2. Sliding Window Flow Control

- With the use of multiple frames for a **single message**, the stop-and-wait protocol does not perform well. Only **one frame** at a time can be in transit.
- Efficiency can be greatly improved by allowing **multiple frames** to be in transit at the **same time**.

- To keep track of the frames, sender station sends **sequentially numbered frames**.
- **Window** meaning number of frames that sender can transmit in **same time**.
- Size of window can be **variable**.



Congestion Control: Reasons of congestions:

1. **Load** on the network.
2. The number of packets sent is greater than the **capacity** of the network.
3. **Low** Bandwidth.
4. **Slow** Processor.
5. **Results:**
   - lost packets (buffer overflow at routers)
   - long delays (queuing in router buffers)