

MODULE - IV

8 Hours

Transport Layer - Introduction and Transport-Layer Services: Relationship Between Transport and Network Layers, Overview of the Transport Layer in the Internet, Multiplexing and Demultiplexing: Connectionless Transport: UDP, UDP Segment Structure, UDP Checksum

Principles of Reliable Data Transfer: Building a Reliable Data Transfer Protocol, Pipelined Reliable Data Transfer Protocols, Go-Back-N, Selective repeat

Connection-Oriented Transport TCP: The TCP Connection, TCP Segment Structure, RoundTrip Time Estimation and Timeout. Reliable Data Transfer. Flow Control. TCP Connection Management.

TRANSPORT LAYER

- Data is represented in SEGMENTS
- Provides logical communication between the applications on different hosts
- Responsible for end to end message delivery
- Provide the acknowledgement for successful transmission of data,if any error occurred this layer is responsible to retransmit the data.
- Transport layer protocols are implemented at end system but not at routers
- **TCP :Transmission Control Protocol(TCP)**
- **UDP:User Datagram Protocol (UDP)**

Application Layer

Presentation Layer

Session Layer

Transport Layer

Network Layer

Data Link Layer

Physical Layer

TCP

- TCP is connection-oriented Protocol.
- TCP is reliable protocol.
- As TCP is connection-oriented protocol, so first the connection is established between two ends and then data is transferred and then the connection is terminated after all data being sent

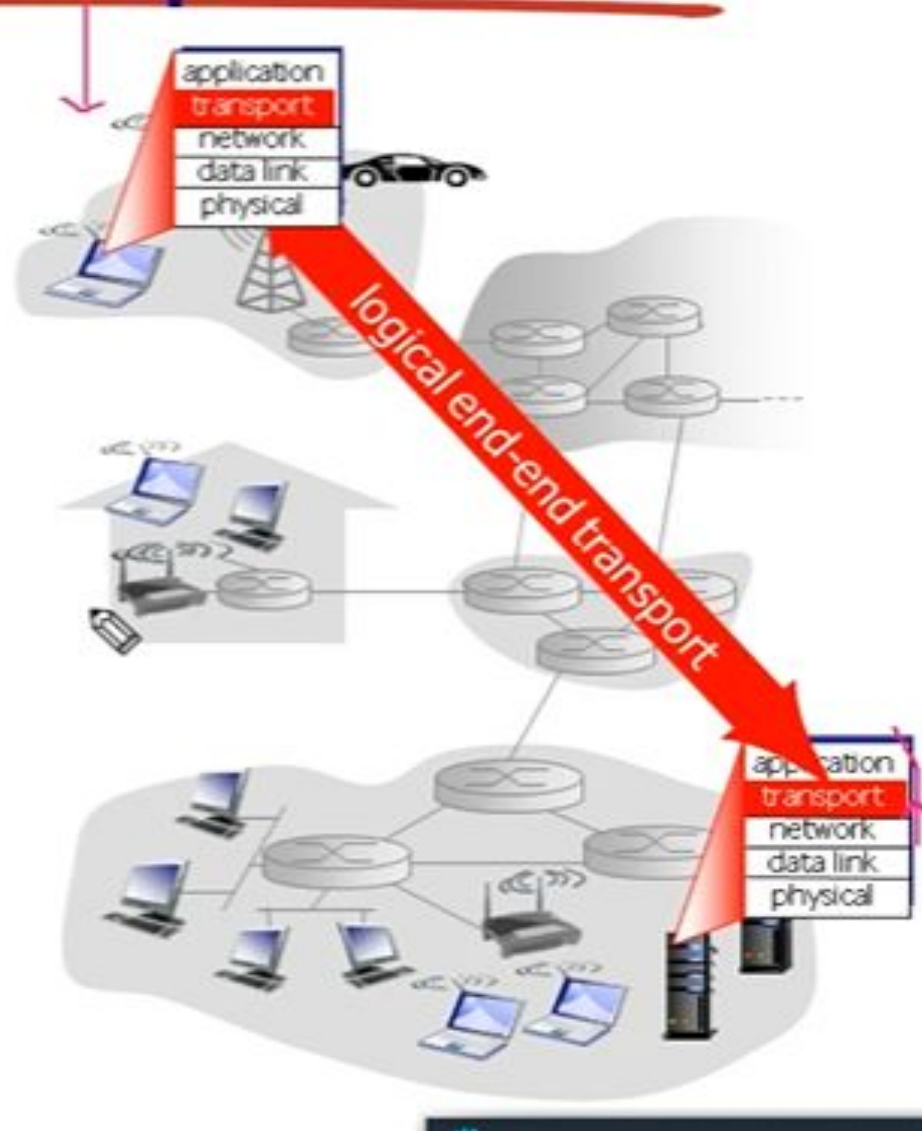
- **User Datagram Protocol (UDP)**

- UDP is not reliable protocol
- The protocol UDP is connectionless.

.

Transport services and protocols

- ❖ provide *logical communication* between app processes running on different hosts
- ❖ transport protocols run in end systems
 - send side: breaks app messages into *segments*, passes to network layer
 - rcv side: reassembles segments into messages, passes to app layer
- ❖ more than one transport protocol available to apps
 - Internet: TCP and UDP



Multiplexing and Demultiplexing


Multiplexing/demultiplexing


Demultiplexing at rcv host:

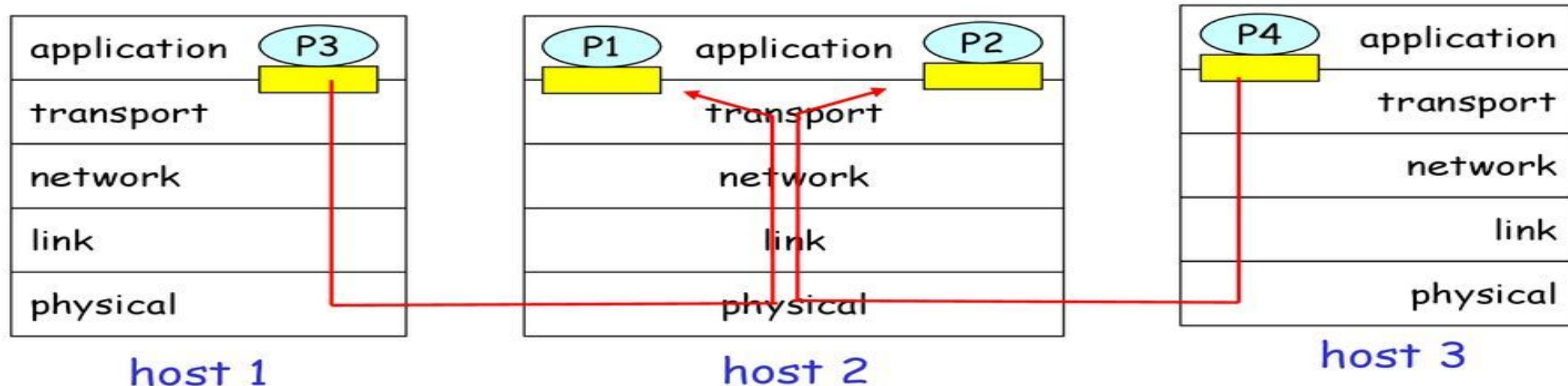
delivering received segments to correct socket

Multiplexing at send host:

gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)

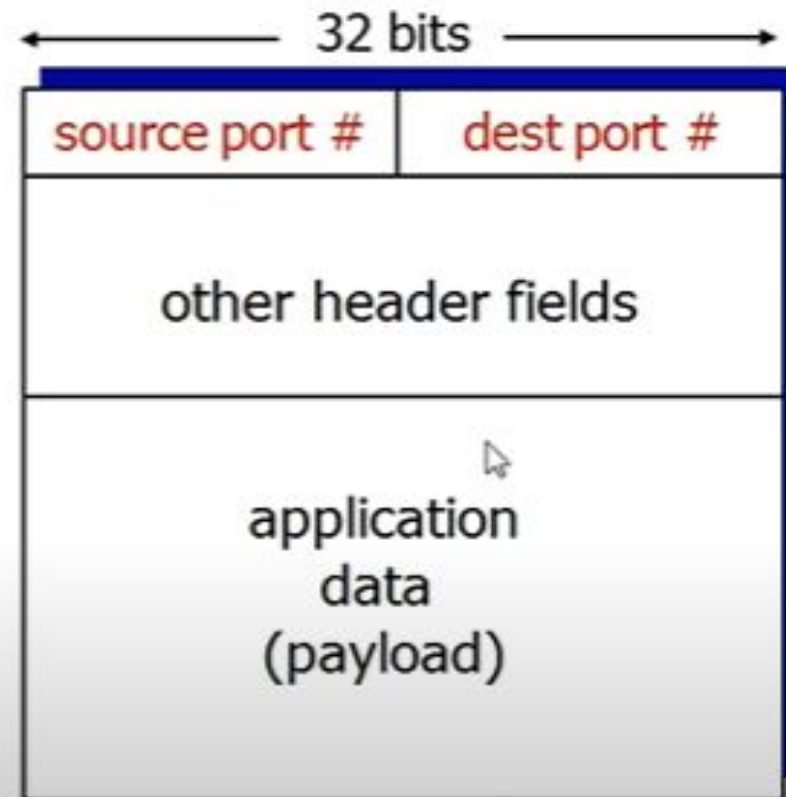
 = socket

 = process



How demultiplexing works

- ❖ 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
- ❖ host uses *IP addresses & port numbers* to direct segment to appropriate socket



TCP/UDP segment format

Connectionless demultiplexing

- ❖ *recall*: created socket has host-local port #:

```
DatagramSocket mySocket1  
= new DatagramSocket(12534) ;
```

- ❖ *recall*: when creating datagram to send into UDP socket, must specify

- destination IP address
- destination port #

- ❖ when host receives UDP segment:

- checks destination port # in segment
- directs UDP segment to socket with that port #



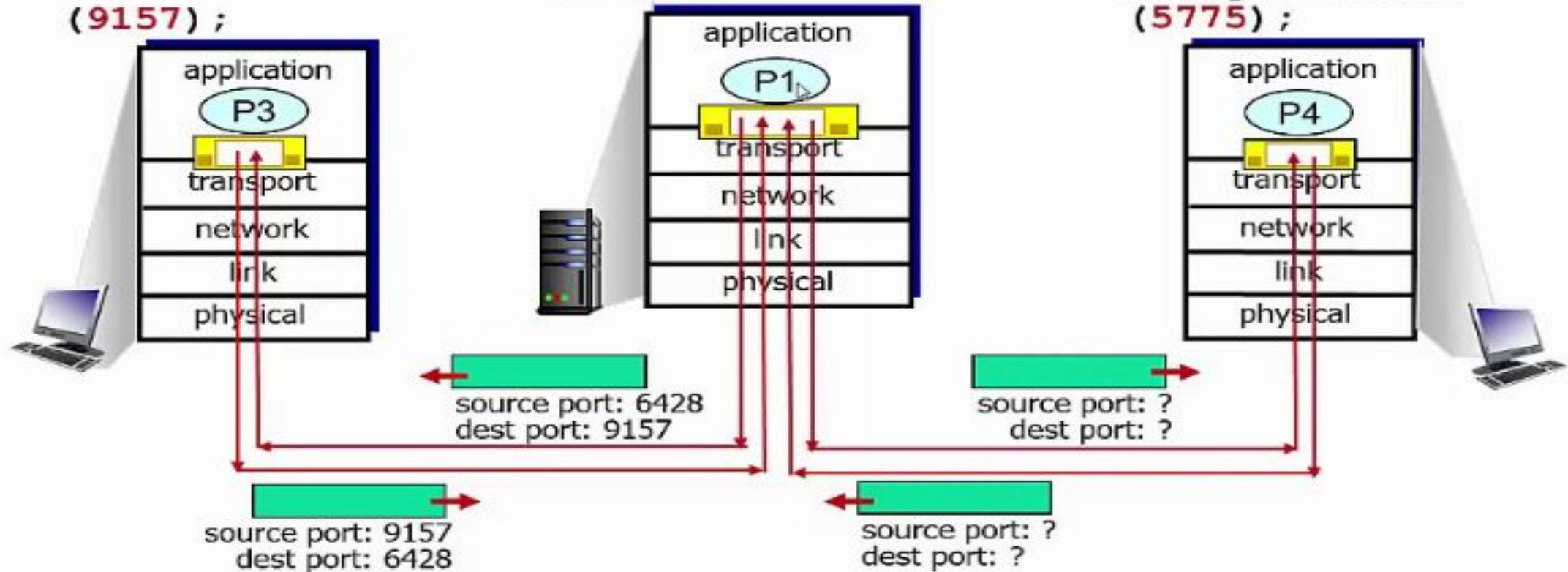
IP datagrams with *same dest. port #*, but different source IP addresses and/or source port numbers will be directed to *same socket* at dest

Connectionless demux: example

```
DatagramSocket  
mySocket2 = new  
DatagramSocket  
(9157);
```

```
DatagramSocket  
serverSocket = new  
DatagramSocket  
(6428);
```

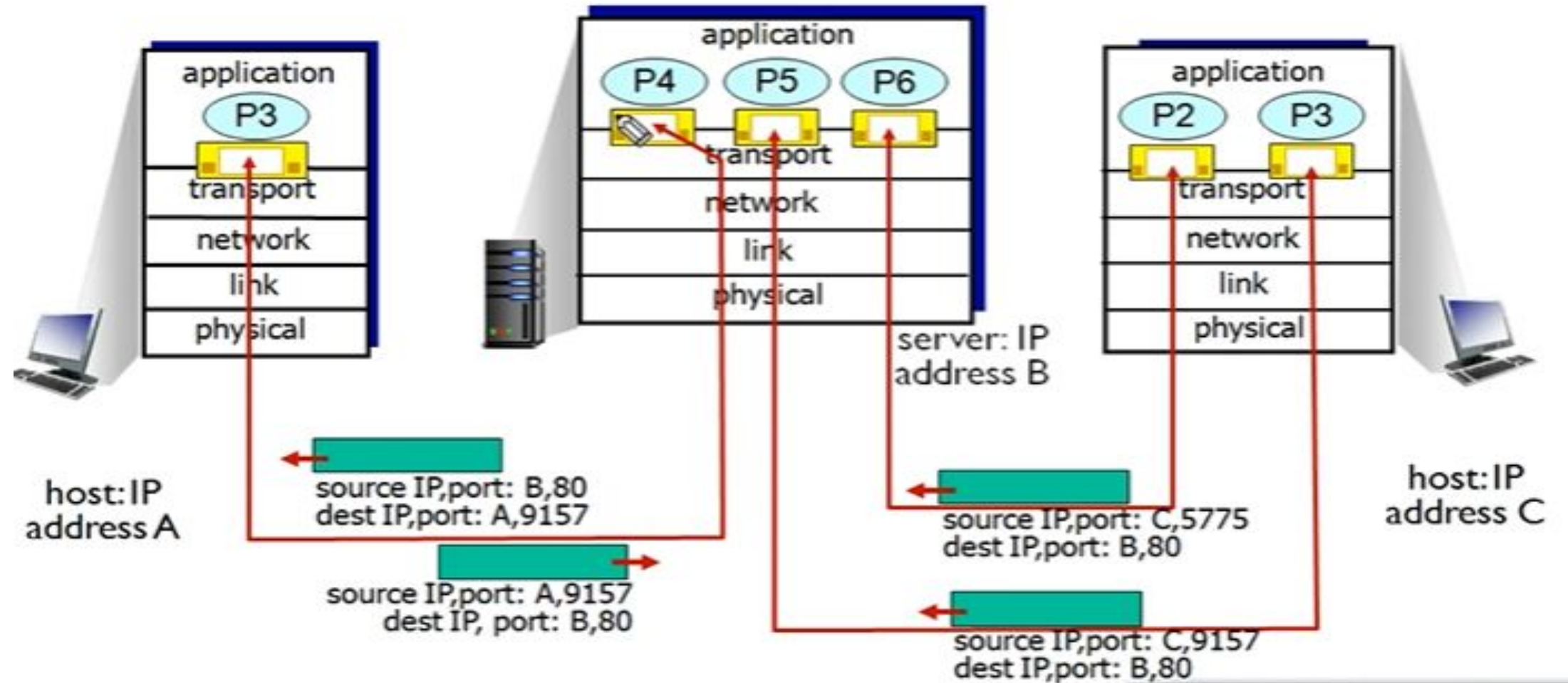
```
DatagramSocket  
mySocket1 = new  
DatagramSocket  
(5775);
```



Connection-oriented demux

- ❖ TCP socket identified by 4-tuple:
 - source IP address
 - source port number
 - dest IP address
 - dest port number
- ❖ demux: receiver uses all four values to direct segment to appropriate socket
- ❖ server host may support many simultaneous TCP sockets:
 - each socket identified by its own 4-tuple
- ❖ web servers have different sockets for each connecting client
 - non-persistent HTTP will have different socket for each request

Connection-oriented demux: example



Connectionless Transport :UDP

- The User Datagram Protocol (UDP) is called as a connection-less, unreliable protocol.
- UDP has a very limited error checking capability.
- It is a very simple protocol and it can be used with minimum overhead.
- The UDP can be used, when process needs to send a small message without any issue of reliability.
- UDP takes less time as compared to TCP (Transmission Protocol) or SCTP (Stream Control Transmission Protocol)

User Datagram Protocol

- User Datagram UDP packets are called as **user datagrams**, which contain the fixed-size header of 8-bytes.
- The important fields of user datagrams are:
 - 1. **Source Port Number**
 - It is used by the process, which is running on the source host.
 - Source Port Number is **16 bits long**
 - If the source host is a **client** and sends a request, the **port number** is **unknown**, which is requested by the process and then accepted by the UDP software.
 - If the **source host is a server** (server sending a reply), the port number is **known**.

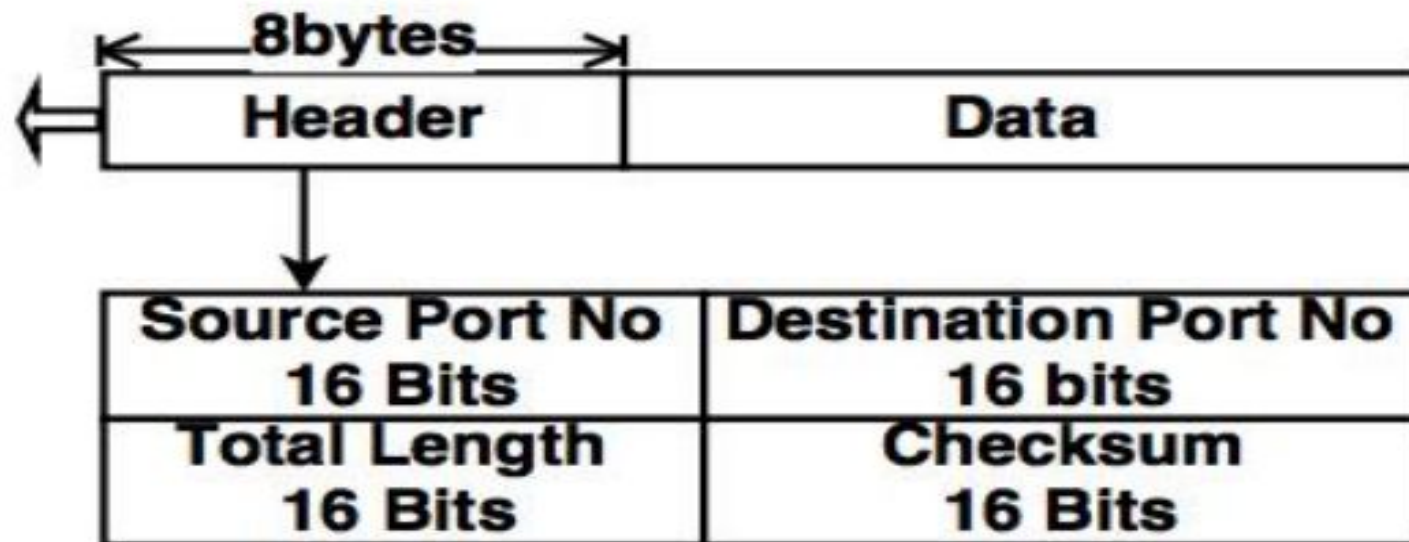
- 2. Destination Port Number

- It is used by the process, which is running on the destination host as well as source host.
- If the destination host is a **server** (server sending request), the **port number is known.**
- If the destination host is a **client** (client sending reply), the port number is **unknown.**

- 3. Length
- This is the total length of the user datagram and it is 16 bit (0 to 65535 bytes) long.
- A UDP user datagram can be stored in an IP datagram with a length of 65535 bytes.
- $\text{Total UDP length} = \text{IP length} - \text{IP headers length}.$

- 4. Checksum

Checksum is used to detect the errors in the user datagram.



User Datagram

Features of UDP Some features of UDP are as stated below:

- 1. Connectionless service UDP provides connectionless service.
- Each user datagram sent by the UDP is an independent datagram.
- There is no relationship between the different user datagrams even if they are coming from the same source and going to the same destination source.

Features of UDP Some features of UDP are as stated below:

- **2. Encapsulation and De-capsulation** While sending the messages from source process to destination, UDP encapsulates and de-capsulates the messages in an IP datagram.
- **3. Queuing** The Queues are associated with the ports in UDP

- Uses of UDP Some uses of UDP are as stated below
- UDP can be very useful to process the required simple request –response communication and less concern of error checking.
- UDP protocol is suitable for multitasking.
- UDP is also applicable in the management processes.
- For example: SNMP (Simple Network Management Protocol)
- UDP is used to implement RIP (Routing Information Protocol).

Transmission Control Protocol (TCP)

- Transmission Control Protocol (TCP)
- TCP is connection oriented protocol and process- to- process protocol.
- TCP uses flow and error control mechanism at the transport layer, hence TCP is reliable transport protocol.
- In TCP, a segment carries a data and control information

Connection Establishment

- TCP transmits data in both the directions (full duplex mode).
- When two TCPs are connected to each other, each TCP needs to initialize the communication (SYN) and approval(ACK) from each end to send the data

Three - Way Handshaking

- The Three - Way Handshaking protocol is used to establish connection between two TCPs. The steps are
- A client sends a SYN data packet to server. The purpose of this step is to see if the server is open for new connection.
- The server needs to keep all ports open to establish a new connection. When the server receives the SYN packet from the client, the client replies and returns the conformation SYN/ACK packet.
- The client receives the SYN/ACK packet and replies with ACK packet to establish the connection.

Three - Way Handshaking

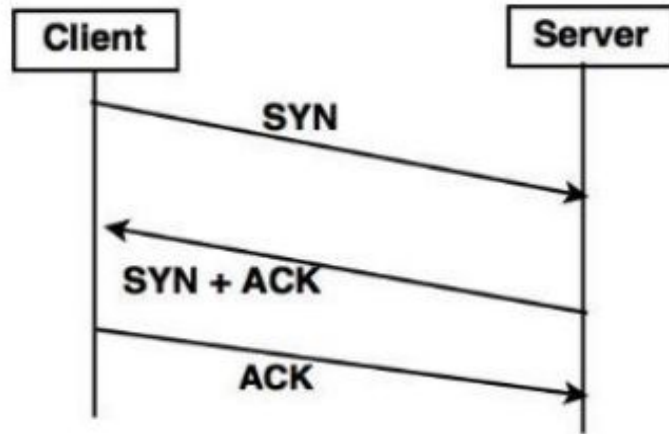
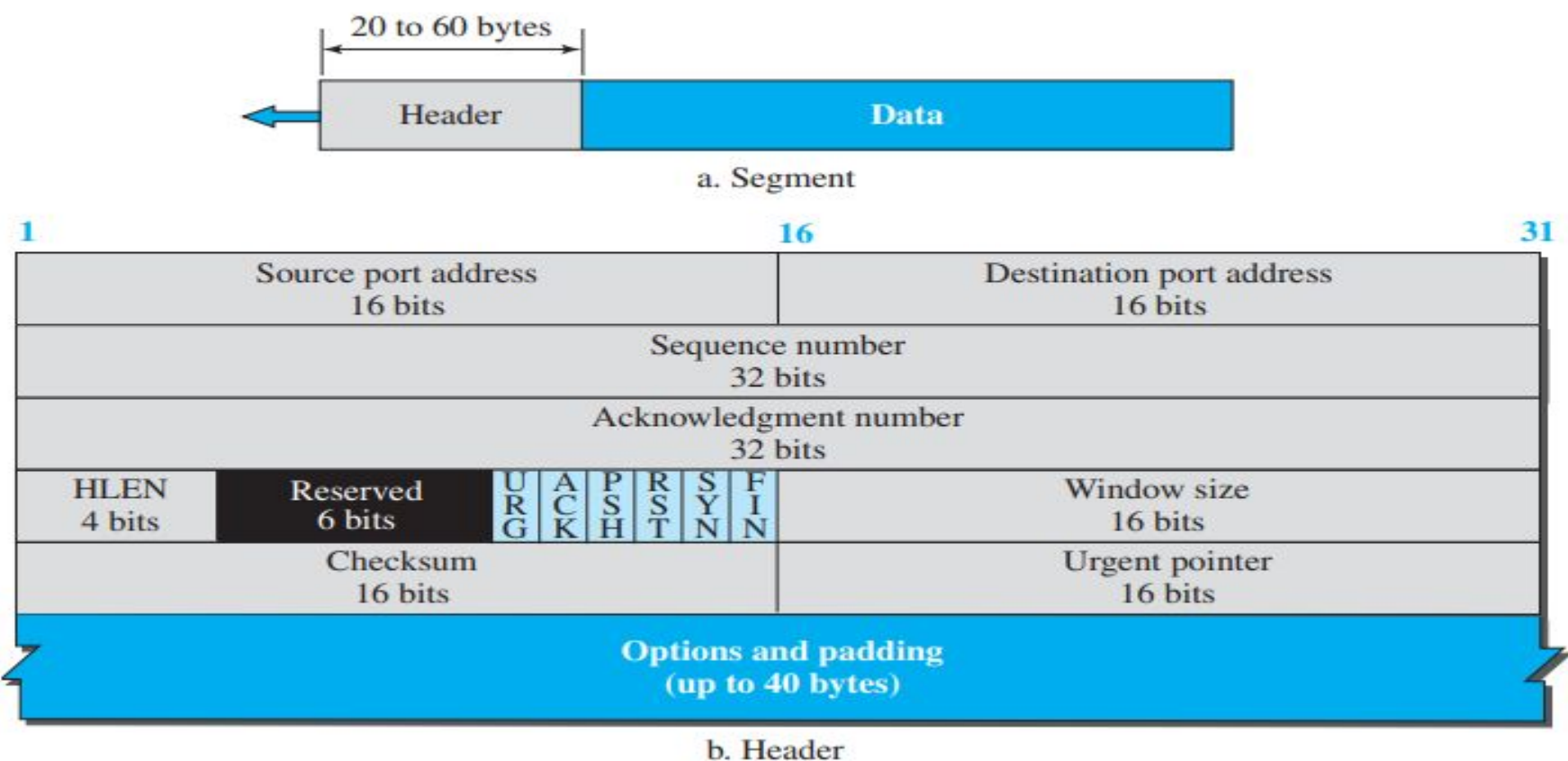


Fig: Three - Way Handshaking

TCP Segment Format

Figure 24.7 *TCP segment format*



TCP Segment Format

- The segment consists of a header of 20 to 60 bytes, followed by data from the application program.
- The header is 20 bytes if there are no options and up to 60 bytes if it contains options.

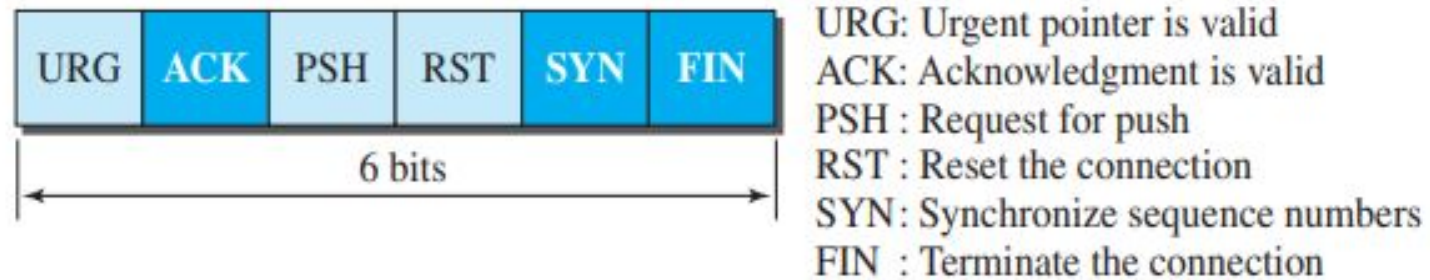
- **Source port address.** This is a 16-bit field that defines the port number of the application program in the host that is sending the segment.
- **Destination port address.** This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment.

- **Sequence number.** This 32-bit field defines the number assigned to the first byte of data contained in this segment.
- As we said before, TCP is a stream transport protocol.
- To ensure connectivity, each byte to be transmitted is numbered.
- The sequence number tells the **destination** which byte in this sequence is the **first byte** in the segment.
- During connection establishment (discussed later) each party uses a random number generator to create an initial sequence number (ISN), which is usually different in each direction

- **Acknowledgment number.** This 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party. If the receiver of the segment has successfully received byte number x from the other party, it returns $x + 1$ as the acknowledgment number.
- **Header length.** This 4-bit field indicates the number of 4-byte words in the TCP header. The length of the header can be between 20 and 60 bytes. Therefore, the value of this field is always between 5 ($5 \times 4 = 20$) and 15 ($15 \times 4 = 60$).

- **Control.** This field defines 6 different control bits or flags
- One or more of these bits can be set at a time. These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP.

Figure 24.8 *Control field*

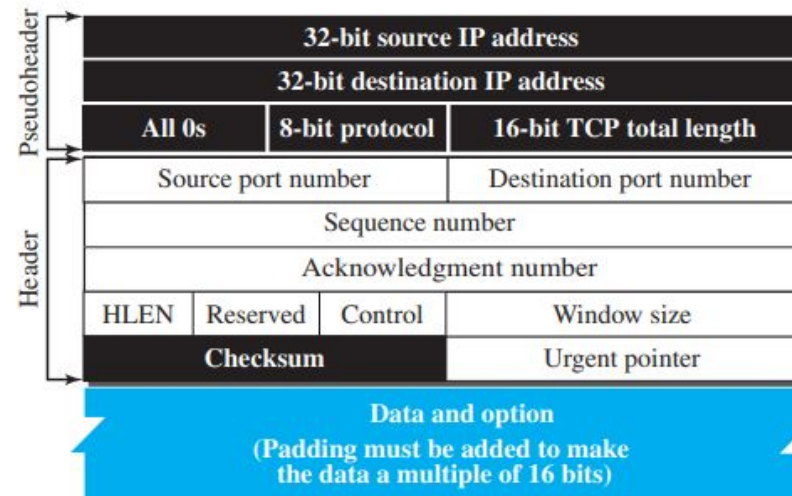


- **Window size.** This field defines the window size of the sending TCP in bytes. Note that the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes.
- This value is normally referred to as the receiving window (rwnd) and is determined by the receiver. The sender must obey the dictation of the receiver in this case.

- Checksum. This 16-bit field contains the checksum.
- use of the checksum in the UDP datagram is optional, whereas the use of the checksum for TCP is mandatory.

Pseudoheader added to the TCP datagram.

Figure 24.9 *Pseudoheader added to the TCP datagram*

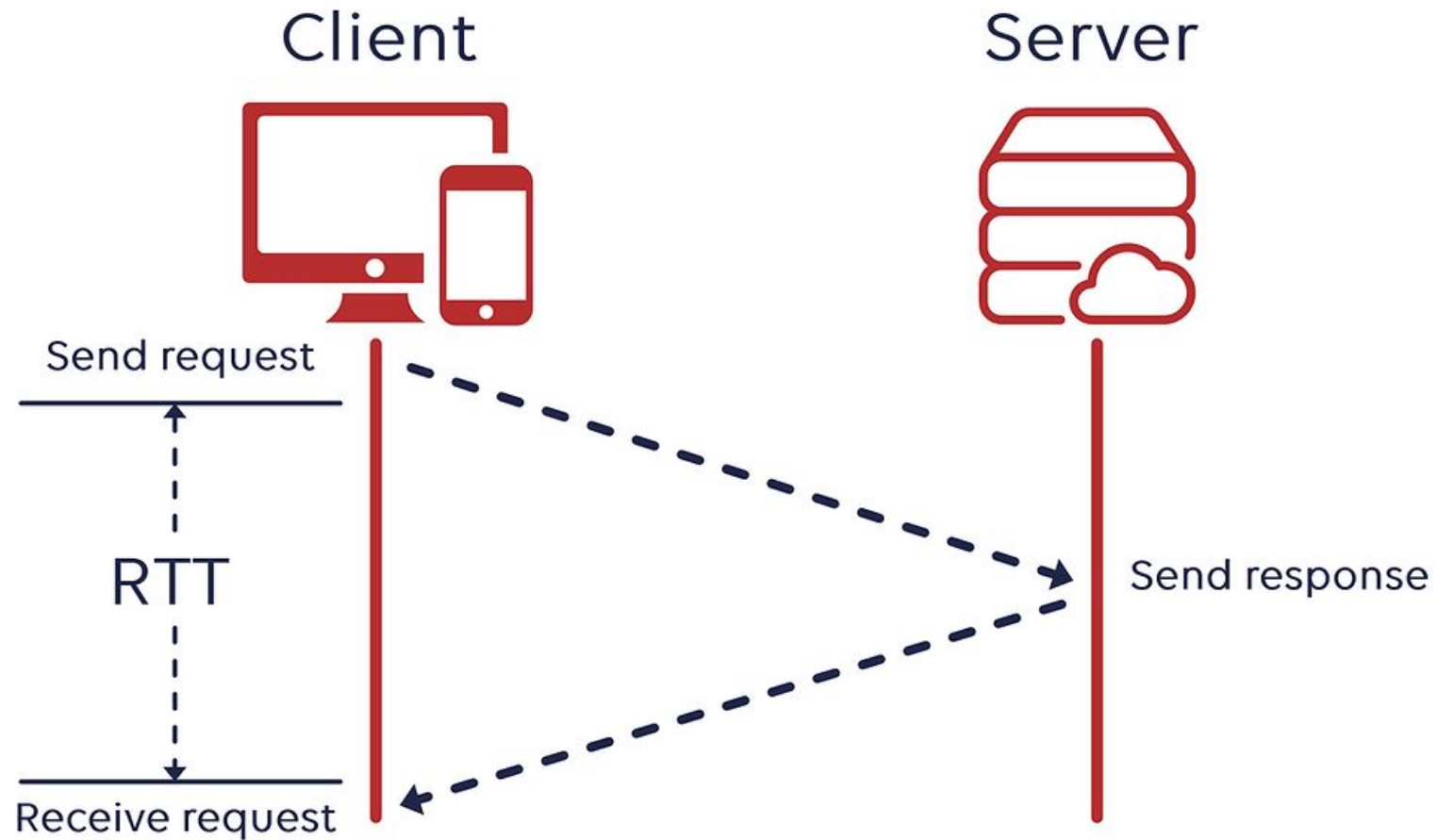


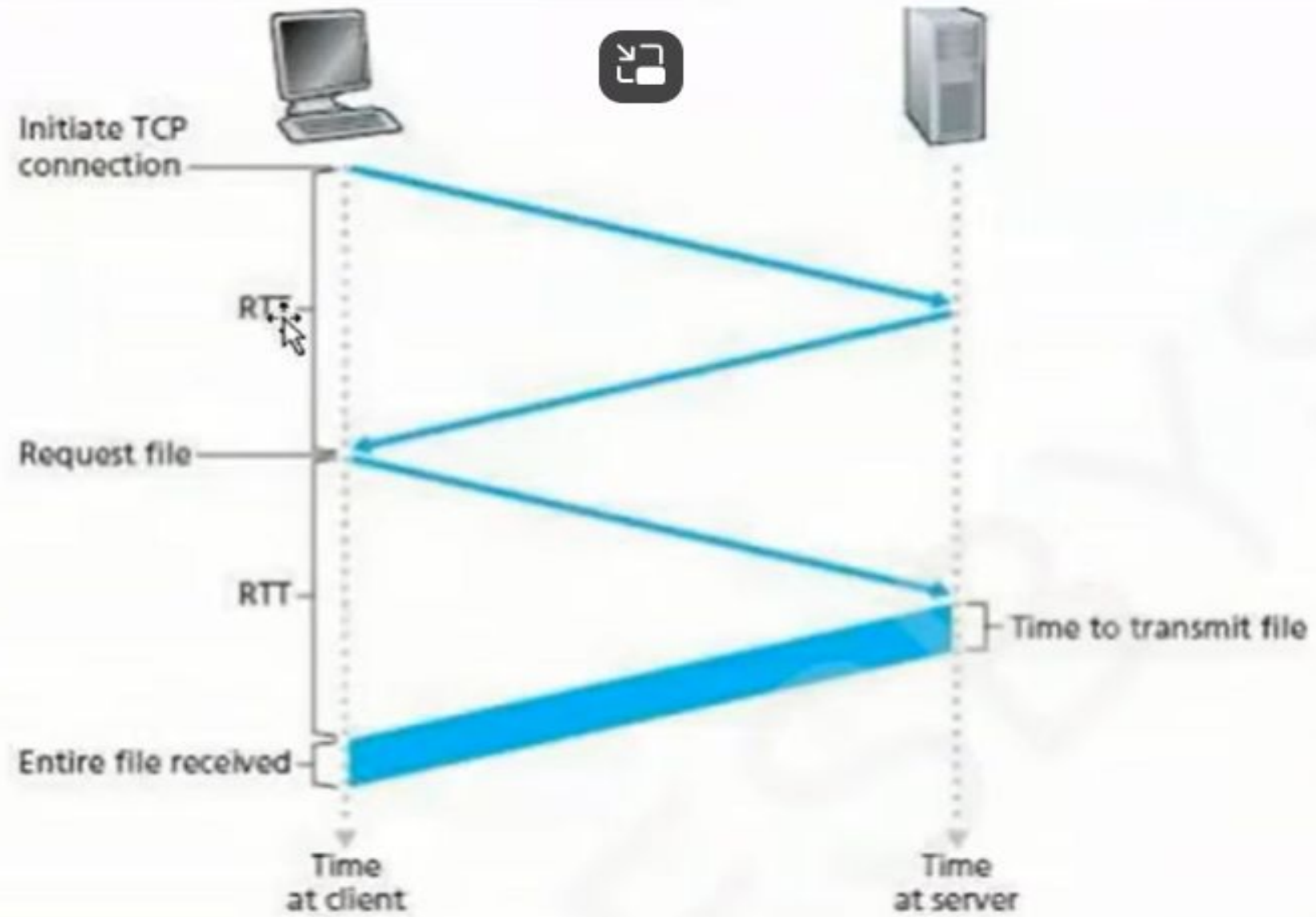
- Urgent pointer. This 16-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data.
- Options. There can be up to 40 bytes of optional information in the TCP header.

Round-trip time (RTT)

- Round-trip time (RTT) in networking is the time it takes to get a response after you initiate a network request.
- When you interact with an application, like when you click a button, the application sends a request to a remote data server.
- Then it receives a data response and displays the information to you. **RTT is the total time** it takes for the request **to travel over the network and for the response to travel back.**
- You can typically measure RTT in **milliseconds.**

Round-trip time (RTT)





ESTIMATING ROUND-TRIP TIME

2.5.3.1 Estimating the Round Trip Time

- SampleRTT is defined as

“The amount of time b/w when the segment is sent and when an acknowledgment is received.”

- Obviously, the SampleRTT values will fluctuate from segment to segment due to congestion.
- TCP maintains an average of the SampleRTT values, which is referred to as EstimatedRTT.

$$\text{EstimatedRTT} = (1 - \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT}$$

- DevRTT is defined as

“An estimate of how much SampleRTT typically deviates from EstimatedRTT.”

$$\text{DevRTT} = (1 - \beta) \cdot \text{DevRTT} + \beta \cdot |\text{SampleRTT} - \text{EstimatedRTT}|$$

- If the SampleRTT values have little fluctuation, then DevRTT will be small.
If the SampleRTT values have huge fluctuation, then DevRTT will be large.

2.5.3.2 Setting and Managing the Retransmission Timeout Interval

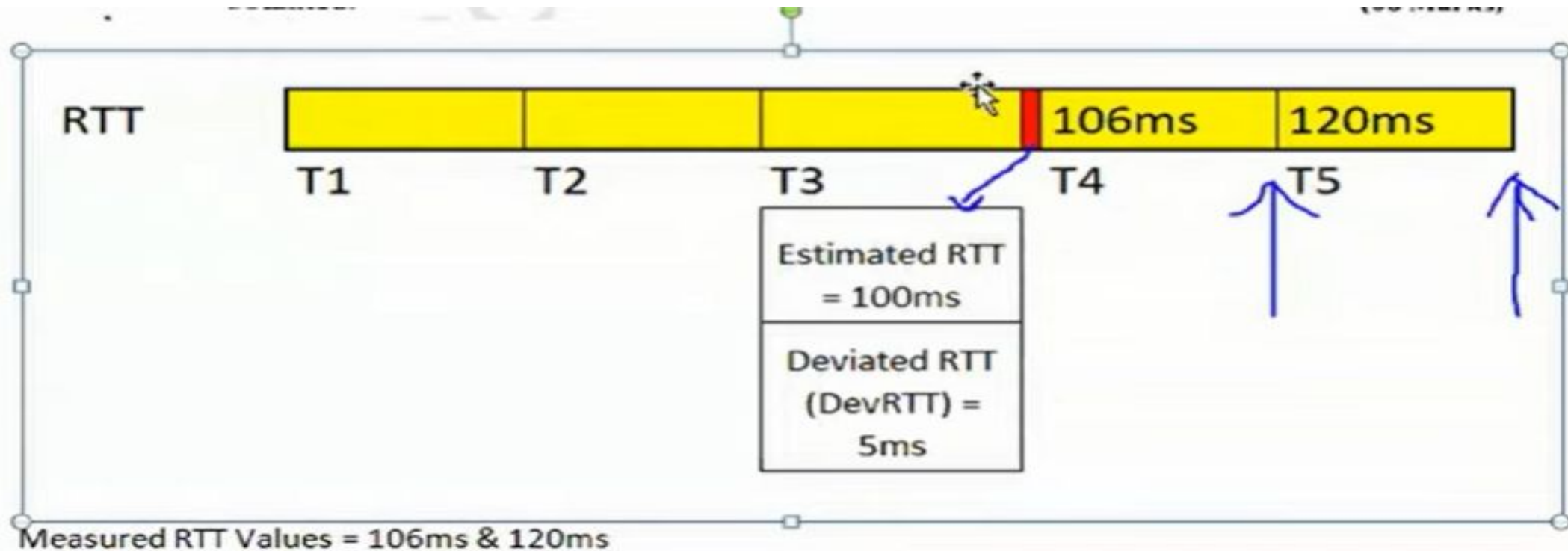
- What value should be used for timeout interval?
- Clearly, the interval should be greater than or equal to EstimatedRTT.
- Timeout interval is given by:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \cdot \text{DevRTT}$$

- 4 a. Suppose that two measured sample RTT values are 106ms and 120ms.
- i) Compute Estimated RTT after each of these Sample RTT value is obtained. Assume $\alpha = 0.125$ and Estimated RTT is 100ms. Just before first of the samples obtained.
 - ii) Compute DeVRTT. Assume $\beta = 0.25$ and DeVRTT is 5ms before first of the samples obtained.

(06 Marks)

- 4 a. Suppose that two measured sample RTT values are 106ms and 120ms.
- Compute Estimated RTT after each of these Sample RTT value is obtained. Assume $\alpha = 0.125$ and Estimated RTT is 100ms. Just before first of the samples obtained.
 - Compute DevRTT. Assume $\beta = 0.25$ and DevRTT is 5ms before first of the samples obtained.
- (06 Marks)



Measured RTT Values = 106ms & 120ms

Alpha = 0.125

Beta = 0.25

Estimated RTT = 100ms

DevRTT = 5ms

106ms

Estimated RTT = $(1 - \text{Alpha}) \times \text{Estimated RTT} + \text{Alpha} \times \text{Sample RTT}$

DevRTT = $(1 - \text{Beta}) \times \text{Dev RTT} + \text{Beta} \times |\text{SampleRTT} - \text{Estimated RTT}|$

Timeout = EstimatedRTT + $4 \times \text{DevRTT}$

$$\text{Estimated RTT} = (1-\text{Alpha}) * \text{Estimated RTT} + \text{Alpha} * \text{Sample RTT}$$

$$= (1-0.125) * \text{100} * 10^{-3} + 0.125 * 106 * 10^{-3}$$

$$= 0.875 * 100 * 0.001 + 0.125 * 106 * 0.001$$

$$= 0.0875 + 0.01325$$

$$= 0.10075 \text{ Sec}$$

$$\text{DevRTT} = (1 - \text{Beta}) * \text{Dev RTT} + \text{Beta} * |\text{SampleRTT} - \text{Estimated RTT}|$$

$$= (1 - 0.25) * 5 * 10^{-3} + 0.25 * |106 * 10^{-3} - 0.10075|$$

$$= 0.75 * 5 * 0.001 + 0.25 * |106 * 0.001 - 0.10075|$$

$$= 0.00375 + 0.25 * |0.106 - 0.10075|$$

$$= 0.00375 + 0.25 * 0.00525$$

$$= 0.00375 + 0.0013125$$

For sample value 106 ms, DevRTT = 0.0050625 Sec

Sample value 120 ms

Estimated RTT = (1-Alpha) * Estimated RTT + Alpha * Sample RTT

$(1-0.125) * 0.10075 + 0.125 * 120 * 0.001$

$0.088156 + 0.015 = 0.10316 \text{ sec}$

For sample value 120 ms, Estimated RTT = 0.10316 Sec

Deviated RTT

DevRTT = (1-Beta) * Dev RTT + Beta * |SampleRTT - Estimated RTT|

$= (1-0.25) * 0.0050625 + 0.25 * |120 * 10^{-3} - 0.10316|$

$= 0.003796875 + 0.25 * 0.01684$

$= 0.003796875 + 0.00421$

$= 0.008006875 \text{ Sec}$

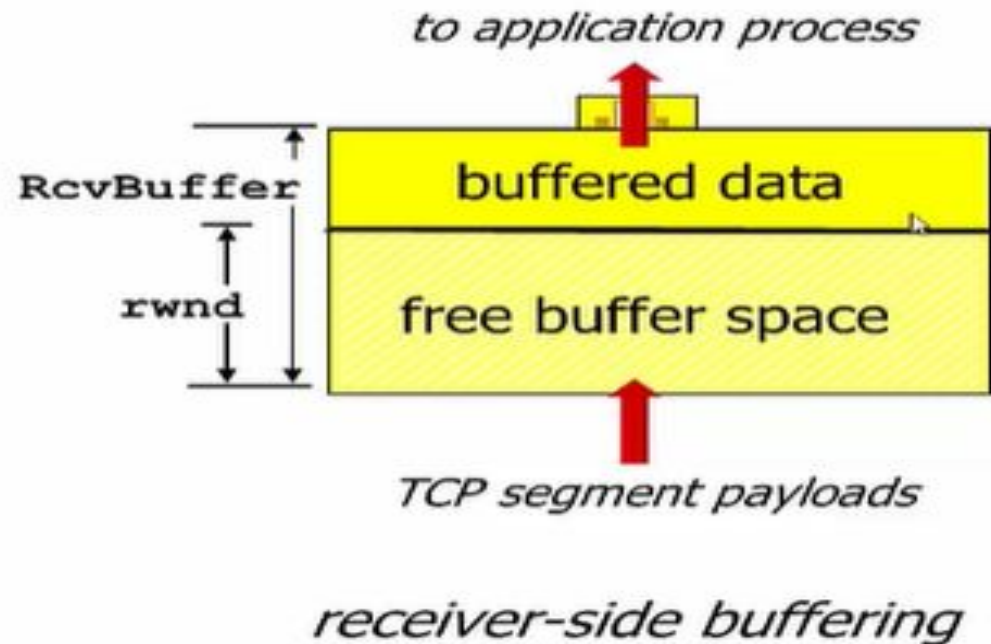
For sample value 120 ms, DevRTT = 8.007ms





TCP flow control

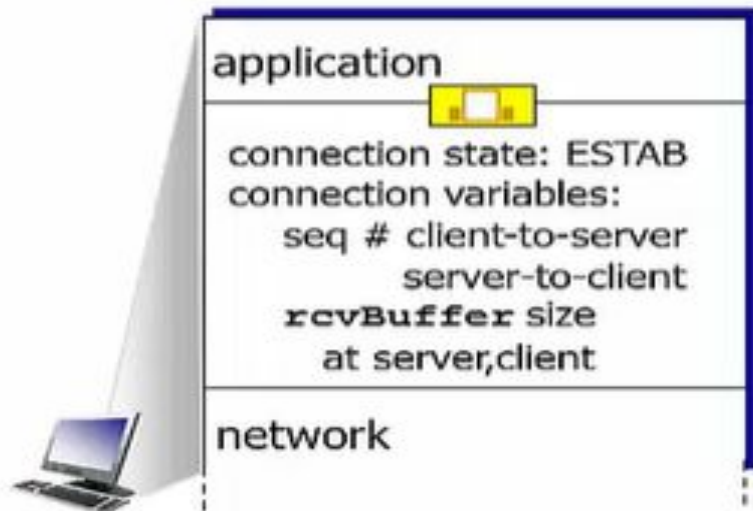
- ❖ receiver “advertises” free buffer space by including **rwnd** value in TCP header of receiver-to-sender segments
 - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
 - many operating systems autoadjust **RcvBuffer**
- ❖ sender limits amount of unacked (“in-flight”) data to receiver’s **rwnd** value
- ❖ guarantees receive buffer will not overflow



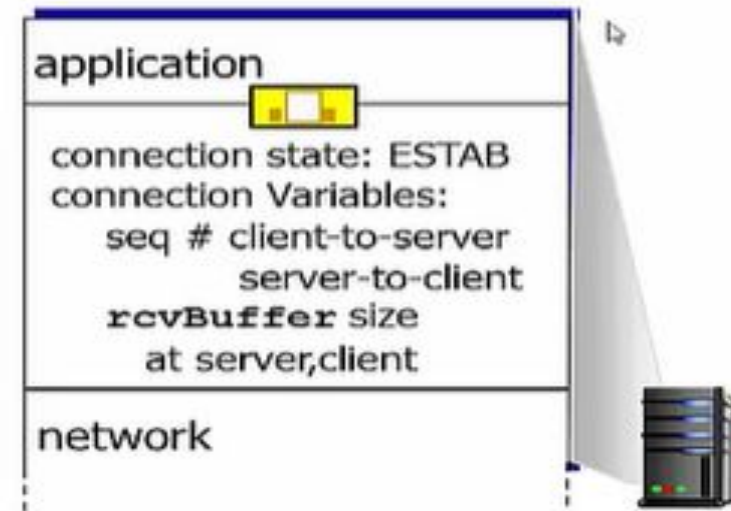
Connection Management

before exchanging data, sender/receiver “handshake”:

- ❖ agree to establish connection (each knowing the other willing to establish connection)
- ❖ agree on connection parameters

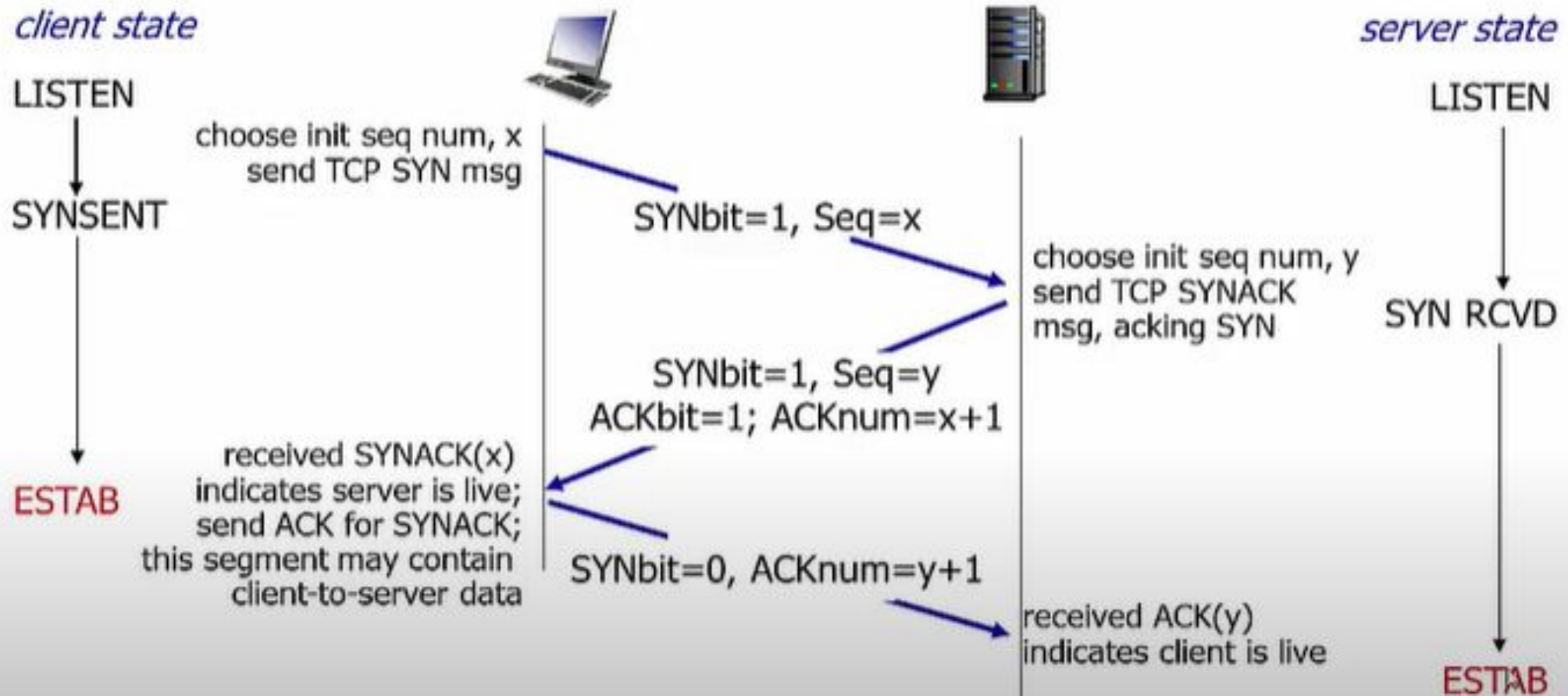


```
Socket clientSocket =  
    newSocket("hostname", "port  
    number");
```



```
Socket connectionSocket =  
    welcomeSocket.accept();
```


TCP 3-way handshake



TCP CONNECTION TERMINATION

