# Transport Layer

Unit- 10

# Introduction

- Data link layer → node-to-node delivery → delivery of frames between two neighboring nodes over a link.
- Network layer → host-to-host delivery → delivery of datagrams between two hosts.
- Transport layer → process-to-process delivery → delivery of data from one process to another.

- The ultimate goal of the transport layer is to provide efficient, reliable, and cost-effective services to the users, normally processes in the application layer.

# Process-to-process delivery

- A process is an application layer entity (running program) that use services of the transport layer.
- At a moment, several processes may be running on the source host and several on the destination host.
- A network layer protocol can deliver the message only to the destination computer. However, it is incomplete delivery. The message will still needs to be handled to the correct process and is done at the transport layer.
- Process-to-process communication can be achieved by **client/server paradigm**. i.e. a process on the **local host**, called a **client**, needs services form a process usually on the **remote host**, called a **server**.

# Addressing

- Whenever we need to deliver something to one specific destination among many, we need an address.
- At data link layer, we need MAC address to choose one node among several nodes.
- At network layer, we need IP address to choose one host among millions.
- At transport layer, we need a transport layer address, called a **port address**, to choose among multiple processes running on the destination host.
- Port number are the 16 bit integers between 0 and 65,535.
- The IANA (Internet Assigned Number Authority) has divided the port numbers into three ranges.
    1. Well-known port numbers
    2. Registered port numbers
    3. Dynamic port numbers

## Well-known port numbers

- These are the universal port numbers used by the server ranging form 0 to 1023.
- Every client process knows the well-known port number of the corresponding server process.

## Registered port numbers

- Port number ranges from 1024 to 49,151.
- These port numbers are assigned to user processes and or applications.

## Ephemeral/Dynamic/Private port numbers

- These are the port numbers defined the client program temporarily (for the duration of the request and its completion).
- They are chosen randomly by the transport layer software running on the client host when initiating a connection.
- Port numbers are greater then 49,151.

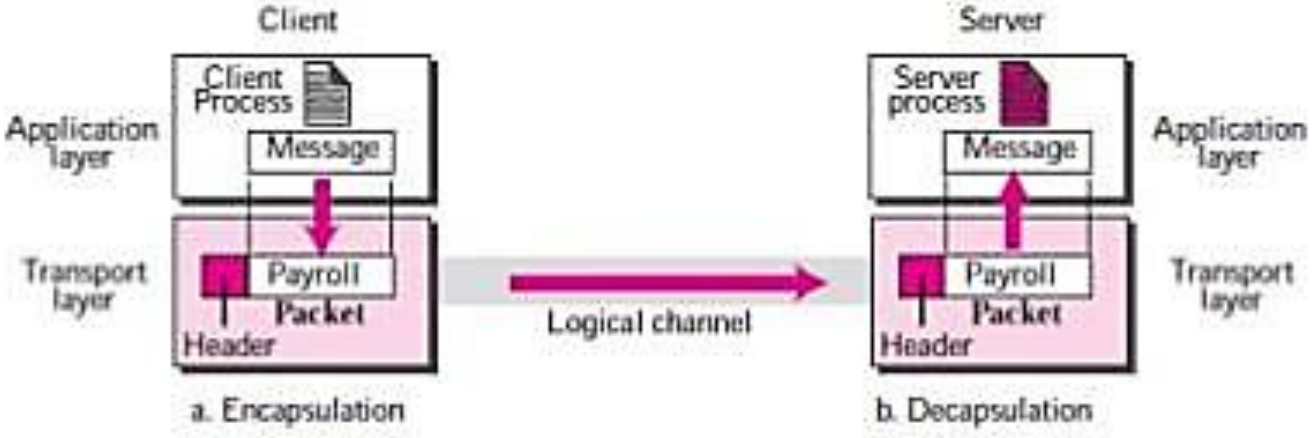
| Port number | Process name | Protocol used | Description |
|---|---|---|---|
| 20 | FTP-DATA | TCP | File transfer—data |
| 21 | FTP | TCP | File transfer—control |
| 22 | SSH | TCP | Secure Shell |
| 23 | TELNET | TCP | Telnet |
| 25 | SMTP | TCP | Simple Mail Transfer Protocol |
| 53 | DNS | TCP and UDP | Domain Name System |
| 69 | TFTP | UDP | Trivial File Transfer Protocol |
| 80 | HTTP | TCP and UDP | Hypertext Transfer Protocol |
| 110 | POP3 | TCP | Post Office Protocol 3 |
| 123 | NTP | TCP | Network Time Protocol |
| 143 | IMAP | TCP | Internet Message Access Protocol |
| 443 | HTTPS | TCP | Secure implementation of HTTP |

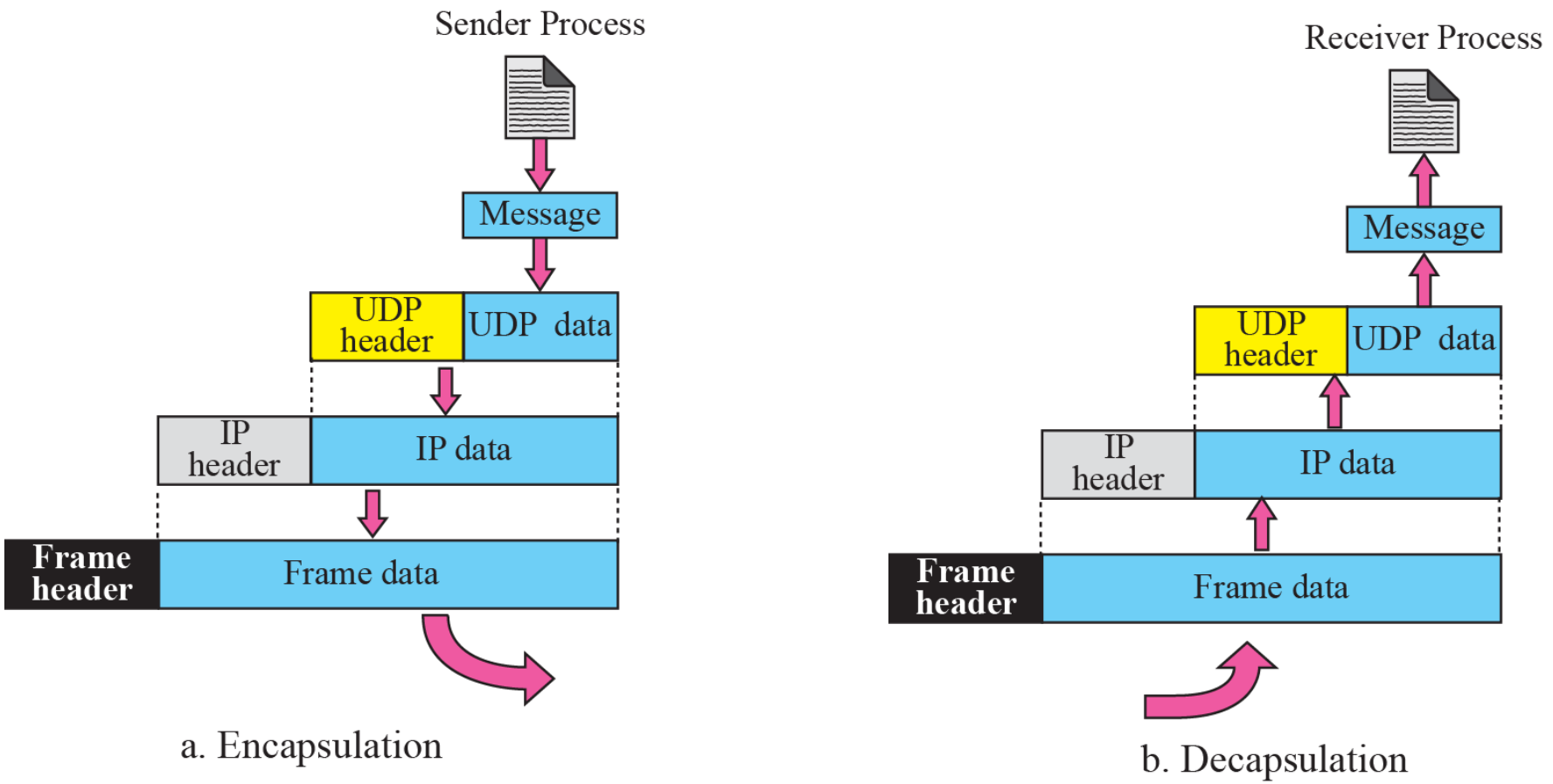
# Socket Address

- The transport layer protocol in the TCP/IP suite needs two identifiers: IP address and port numbers, at each end to make a make a process-to-process connection. The combination of an IP address and a port number is called a socket address.
- The client socket address defines the client process uniquely just as the sever socket address defines the server process uniquely.

# Encapsulation and Decapsulation

- When a process has message to send, it passes the message to the transport layer along with a pair of socket address. The transport layer receives the data and adds the transport layer header. The packet at the transport layer in the internet are called the user datagrams or segments.
- When the message arrives at the destination transport layer, the header is dropped and the transport layer divides the message to the process running at the application layer.




Sender Process
Message
UDP header
UDP data
IP header
IP data
Frame header
Frame data
a. Encapsulation
Receiver Process
Message
UDP header
UDP data
IP header
IP data
Frame header
Frame data
b. Decapsulation

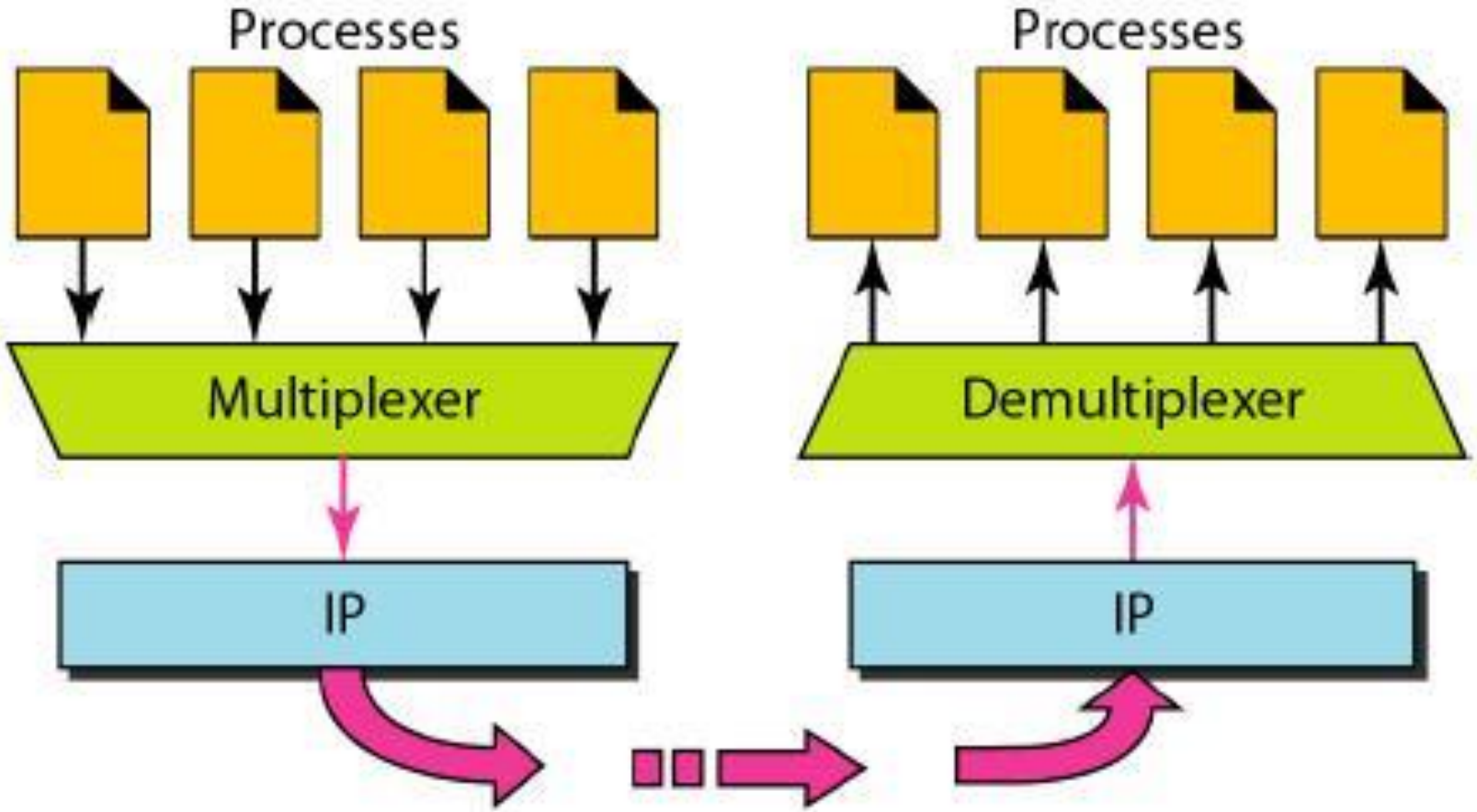
# Multiplexing and Demultiplexing

**Multiplexing**

- At the sender site, there may be several processes that need to send packets. However, there is only one transport layer protocol at any time. This is many-to-one relationship and requires multiplexing.
- The protocol accepts messages from different processes, differentiated by their assigned port numbers.
- After adding the header, the transport layer passes the packet to the network layer.

**Demultiplexing**

- At the receiver site, the relationship is one-to-many and requires demultiplexing.
- The transport layer receives datagrams from the network layer.
- After error checking and dropping of the header, the transport layer delivers each message to the appropriate process based on the port number,


Processes
Processes
Multiplexer
Demultiplexer
IP
IP

# Connectionless versus Connection-Oriented Service

- A transport layer protocol can either be connectionless or connection-oriented.

**Connection oriented Services**

- In connection oriented service, a connection is first established between the sender and receiver. Then data are transferred. At the end, the connection is released.
- In this type of transmission the receiving device sends an acknowledgment, back to the source after a packet or group of packet is received.
- Reliable but slower
- Example: TCP (Transmission control protocol)

**Connectionless Services**

- In this type of service, the packets are send from one party to another with no need for connection establishment and connection release.
- The packets are not numbered; they may be delayed or lost or arrived out of order.
- There is no acknowledgement either.
- Unreliable, but faster.
- Example: UDP (User datagram protocol)

# Differences:

| Connection Oriented Service | Connectionless Service |
|---|---|
| 1. Connection is established between sender and receiver prior to data transfer. | 1. No connection is established between sender and receiver prior to data transfer. |
| 2. Acknowledgment is sent to the sender after receiving the packet by the receiver. | 2. No acknowledgement |
| 3. Guarantee the delivery of packets. Loss packets are retransmitted. | 3. No guarantee of delivery of packets. Lost packets are not known or retransmitted. |
| 4. Reliable ; use flow and error control at transport layer | 4. Unreliable |
| 6. Slower and complex service | 6. Faster service |
| 7. Example: TCP, SCTP | 7. Example: UDP |

# UDP: User Datagram Protocol

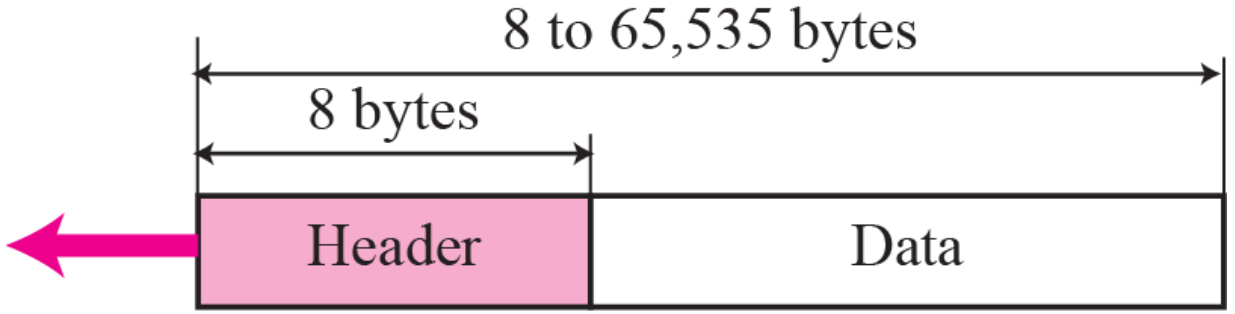
- UDP provides a connectionless service.
- Each user datagram sent by UDP is an independent datagram.
- These user datagram are not numbered.
- No connection is established and no connection is terminated, the user datagram can travel on a different path.

**Table 14.1** *Well-known Ports used with UDP*

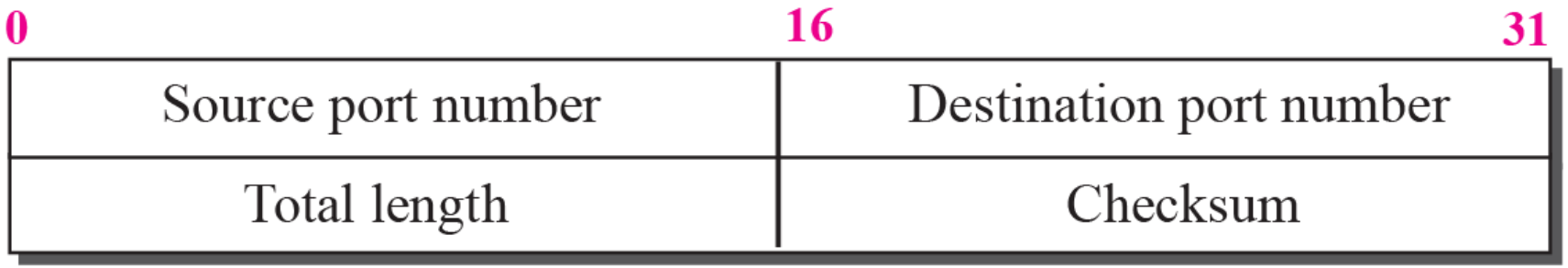
| *Port* | *Protocol* | *Description* |
|---|---|---|
| 7 | Echo | Echoes a received datagram back to the sender |
| 9 | Discard | Discards any datagram that is received |
| 11 | Users | Active users |
| 13 | Daytime | Returns the date and the time |
| 17 | Quote | Returns a quote of the day |
| 19 | Chargen | Returns a string of characters |
| 53 | Domain | Domain Name Service (DNS) |
| 67 | Bootps | Server port to download bootstrap information |
| 68 | Bootpc | Client port to download bootstrap information |
| 69 | TFTP | Trivial File Transfer Protocol |
| 111 | RPC | Remote Procedure Call |
| 123 | NTP | Network Time Protocol |
| 161 | SNMP | Simple Network Management Protocol |
| 162 | SNMP | Simple Network Management Protocol (trap) |

# User Datagram

- UDP packets, called user datagrams, have fixed-size header of 8 bytes.




a. UDP user datagram




b. Header format

- **Source Port:** This is the port number of the application that is originating the user data.
- **Destination Port:** This is the port number pertaining to the destination application.
- **Length:** This field describes the total length of the UDP datagram, including both data and header Information.
- **UDP Checksum:** optional field. Used to detect errors over the entire user datagram

# Flow and error control in UDP

- It is simple, unreliable transport protocol. There is no flow control and hence no window mechanism. The receiver may overflow with incoming messages.
- There is no error control mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded.

# Uses of UDP

- Suitable for a process that requires simple request-response communication such as **DNS** (Domain Name Service), an client-server application.
- Suitable for a process with internal flow and error control mechanism like **TFTP** (Trivial File Transfer Protocol).
- Suitable transport protocol for multicasting.
- Used for management processes such as **SNMP** (Simple Network Management Protocol).
- Used for some route updating protocols such as **RIP** (Routing Information Protocol)

# TCP: Transmission Control Protocol

- The **Transmission Control Protocol** (**TCP**) is a core protocol of the Internet Protocol Suite.
- TCP services:
  - Connection-oriented
  - Reliable
  - Full duplex communication
  - Byte stream delivery
  - segmentation

# Connection-oriented

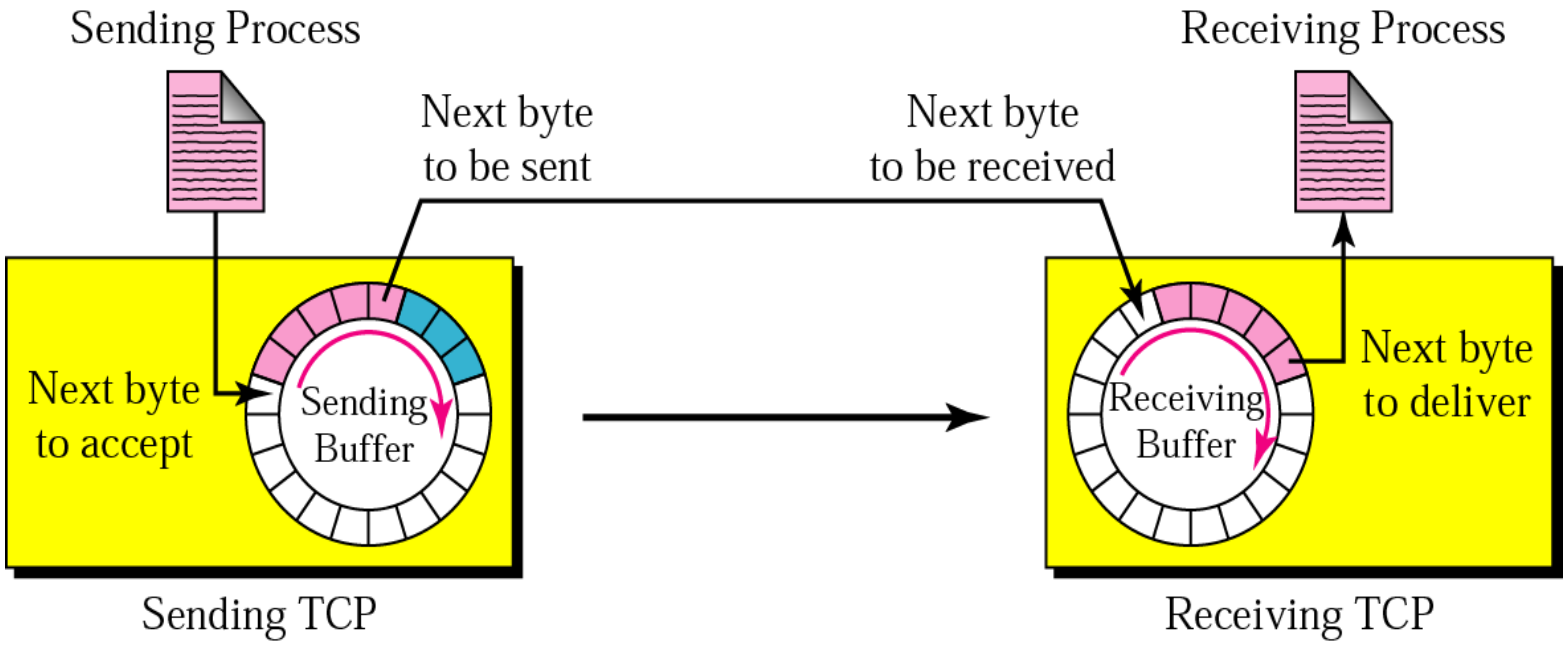
- *Connection oriented* means that a virtual connection is established before any user data is transferred. The following occurs:
  1. The two TCPs establish a connection between them.
  2. Data are exchanged in both direction.
  3. The connection is terminated.
- If the connection cannot be established - the user program is notified.
- If the connection is ever interrupted - the user program(s) is notified.

# Reliable

- *Reliable* means that every transmission of data is acknowledged by the receiver to check safe and sound arrival of data.
- If the sender does not receive acknowledgement within a specified amount of time, the sender retransmits the data

# Full Duplex Communication

- TCP offers full duplex communication, in which data can flow in both direction at the same time.
- Each TCP then has a sending and receiving buffer, and segments move in both direction.



# Byte stream delivery

- TCP, unlike UDP, is a **stream-oriented protocol**. That is, TCP allows the sending process to deliver data as stream of bytes and allows the receiving process to obtain as a stream of bytes.

# Segments

- The IP layer, as a service provider for TCP, needs to send data in packets, not as a stream of bytes.
- At transport layer, TCP groups a number of bytes together into a packet called a **segment**, and assign a **sequence number** to each segment.
- TCP adds a **header** to each segment and delivers the segment to the IP layer for transmission. The segments are encapsulated in IP datagrams and transmitted.

- Suppose a TCP connection is transferring a file of 5000bytes. The first byte is numbered 10,001. what are the sequence numbers for each segment if data are sent in five segments, each carrying 1000 bytes?

**Solution:**

The following shows the sequence number for each segment:

| | |
|---|---|
| Segment 1 | Sequence Number: 10,001 (range : 10,001 to 11,000) |
| Segment 2 | Sequence Number: 11,001 (range : 11,001 to 12,000) |
| Segment 3 | Sequence Number: 12,001 (range : 12,001 to 13,000) |
| Segment 4 | Sequence Number: 13,001 (range : 13,001 to 14,000) |
| Segment 5 | Sequence Number: 14,001 (range : 14,001 to 15,000) |

# Sequence Number and Acknowledgment Number

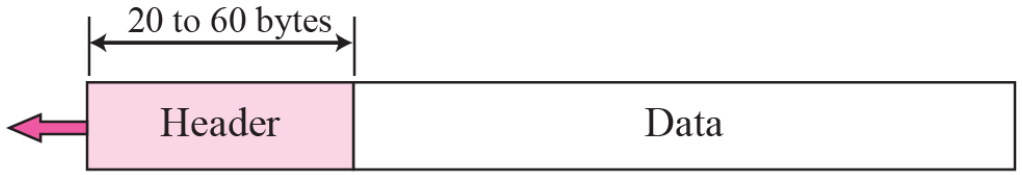
**Sequence Number**

- After the byes have been numbered, TCP assigns a sequence number to each segment that is being sent.
- The value in the sequence number field of a segment defines the number of the first data byte contained in that segment.

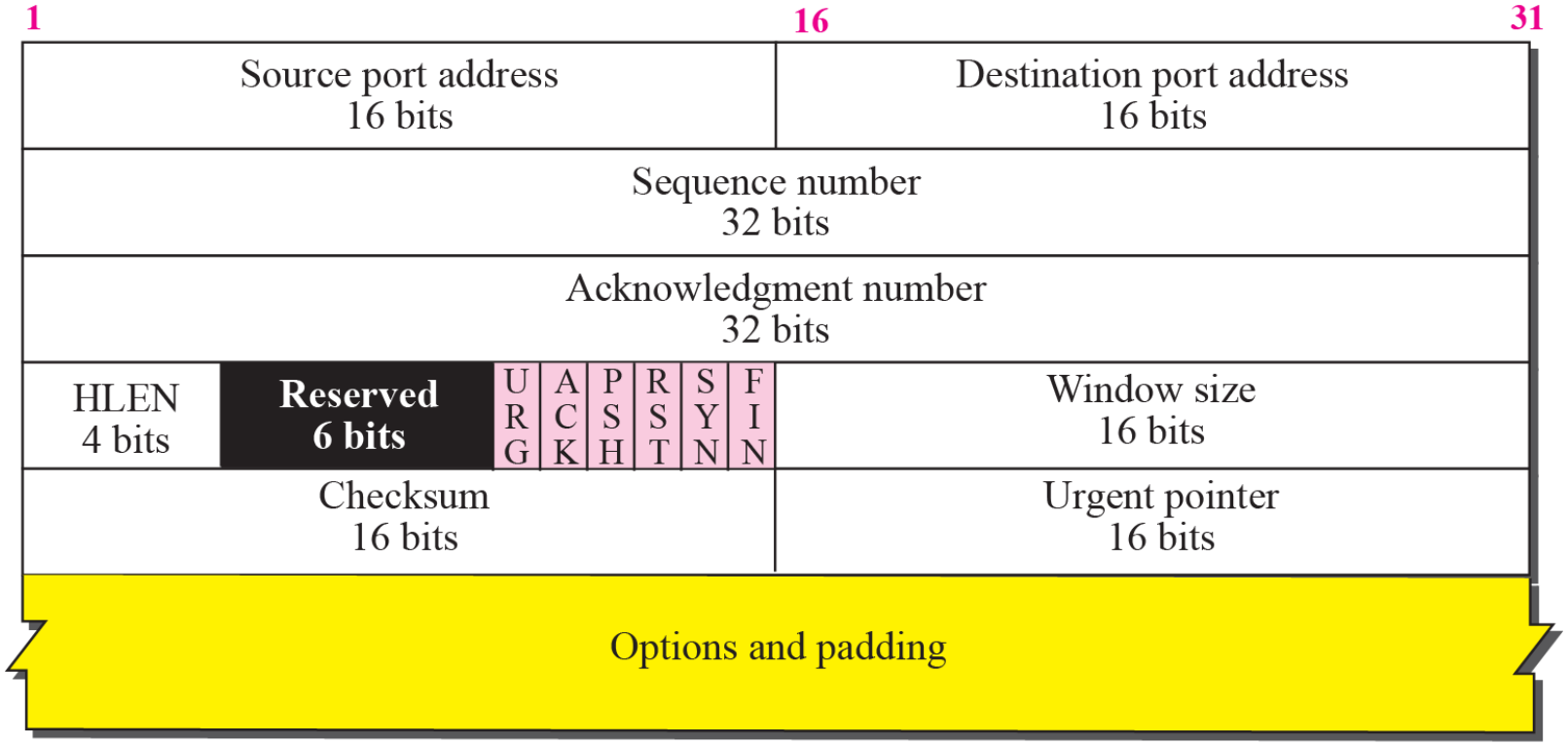
**Acknowledgment Number**

- After receiver receives the bytes, it sends acknowledgment number to the sender.
- However, the value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive.
- The acknowledgement number is cumulative. That is if a party uses 5643 as an acknowledgment number, it has received all bytes from the beginning up to 5642. however this does not mean the party has received 5642 bytes because the first byte does not necessarily have to start from 0.

# TCP Segment Format



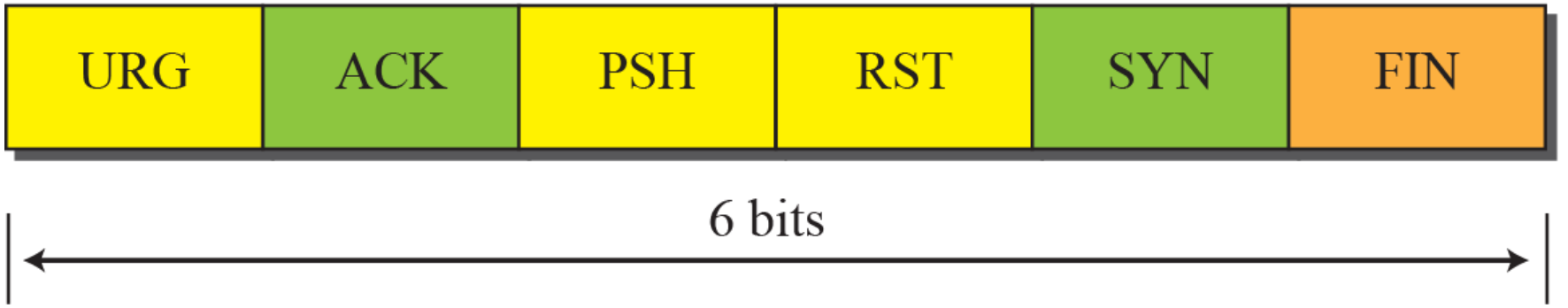

a. Segment




b. Header

# Control Field

URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection



# A TCP Connection

- TCP is connection-oriented protocol that requires three phases: connection establishment, data transfer and connection termination.
- It establishes a virtual path between source and destination. All the segments belongs to a message are then sent over this virtual path.

# Connection Establishment: Three-way Handshake

- TCP transmits data in full duplex mode.
- The connection establishment in TCP is called three-way handshaking.

**Three-way Handshake**

- A three-way-handshake is a method used in a TCP/IP network to create a connection between a local host/client and server.
- It is a three-step method that requires both the client and server to exchange SYN and ACK (acknowledgment) segment before actual data communication begins.
- A three-way-handshake is also known as a TCP handshake.

**The three Steps:**

**Step 1:**

- The client sends the first segment, a SYN segment, in which only the SYN flag is set.
- This segment is used for synchronization of a sequence number.
- It consumes one sequence number. It is incremented by 1 when data transfer starts. But, a SYN segment does not carry any data.
- The objective of this packet is to ask if the server is open for new connection.

**Step 2**:

- The target server must have open ports that can accept and initiate new connections. When the server receives the first SYN segment, then the server send the second segment, a SYN+ACK segment, with two flag bit set: SYN and ACK.
- A SYN+ACK segment cannot carry data, but consumes one sequence number.

**Step 3**:

- The client sends the third segment. This is just an ACK segment.
- It acknowledges the receipt of the second segment with the ACL flag and acknowledgment number field.
- An ACK segment,, if carrying no data, consumes no sequence number.

Upon completion of this process, the connection is created and the host and server can communicate.

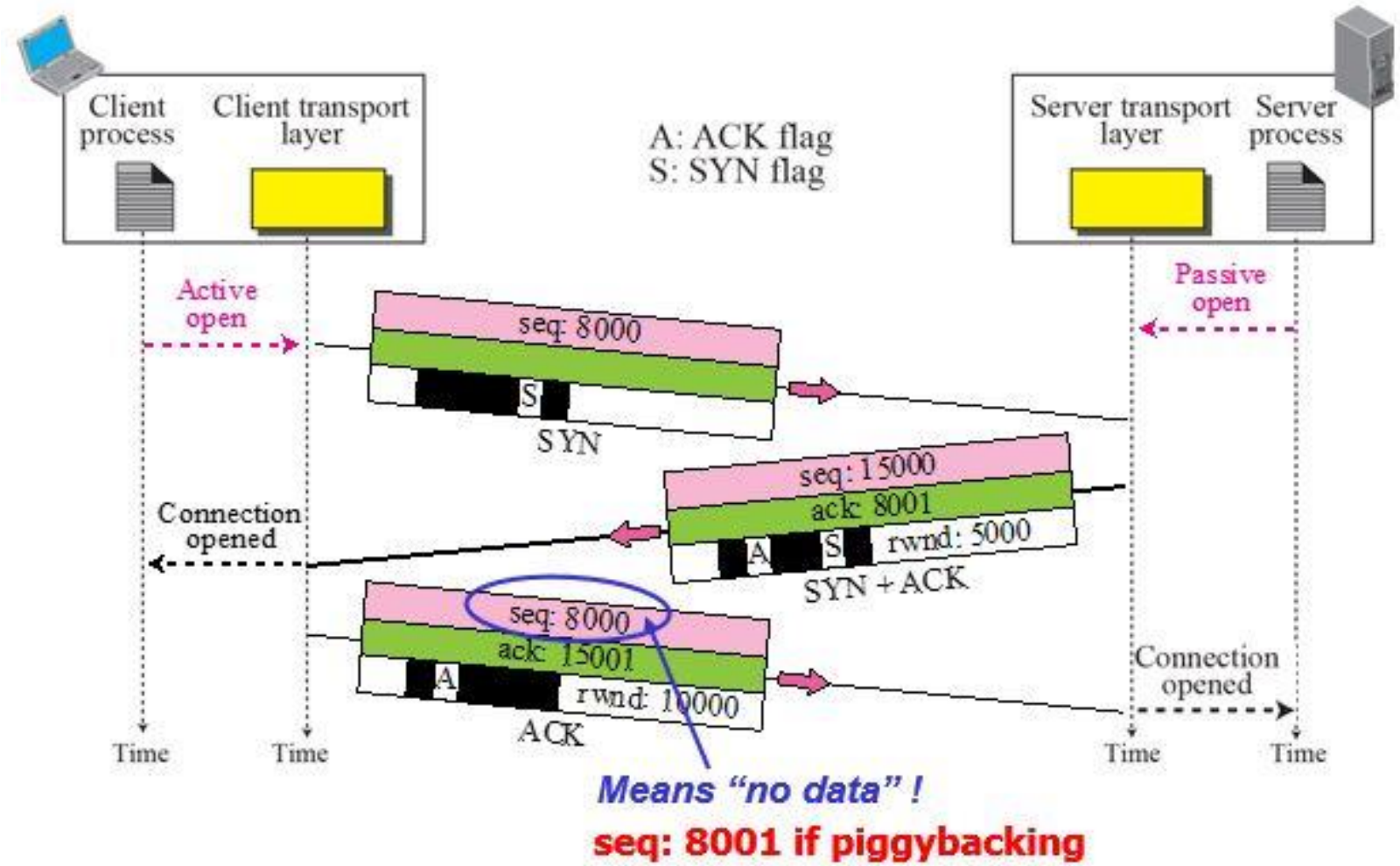
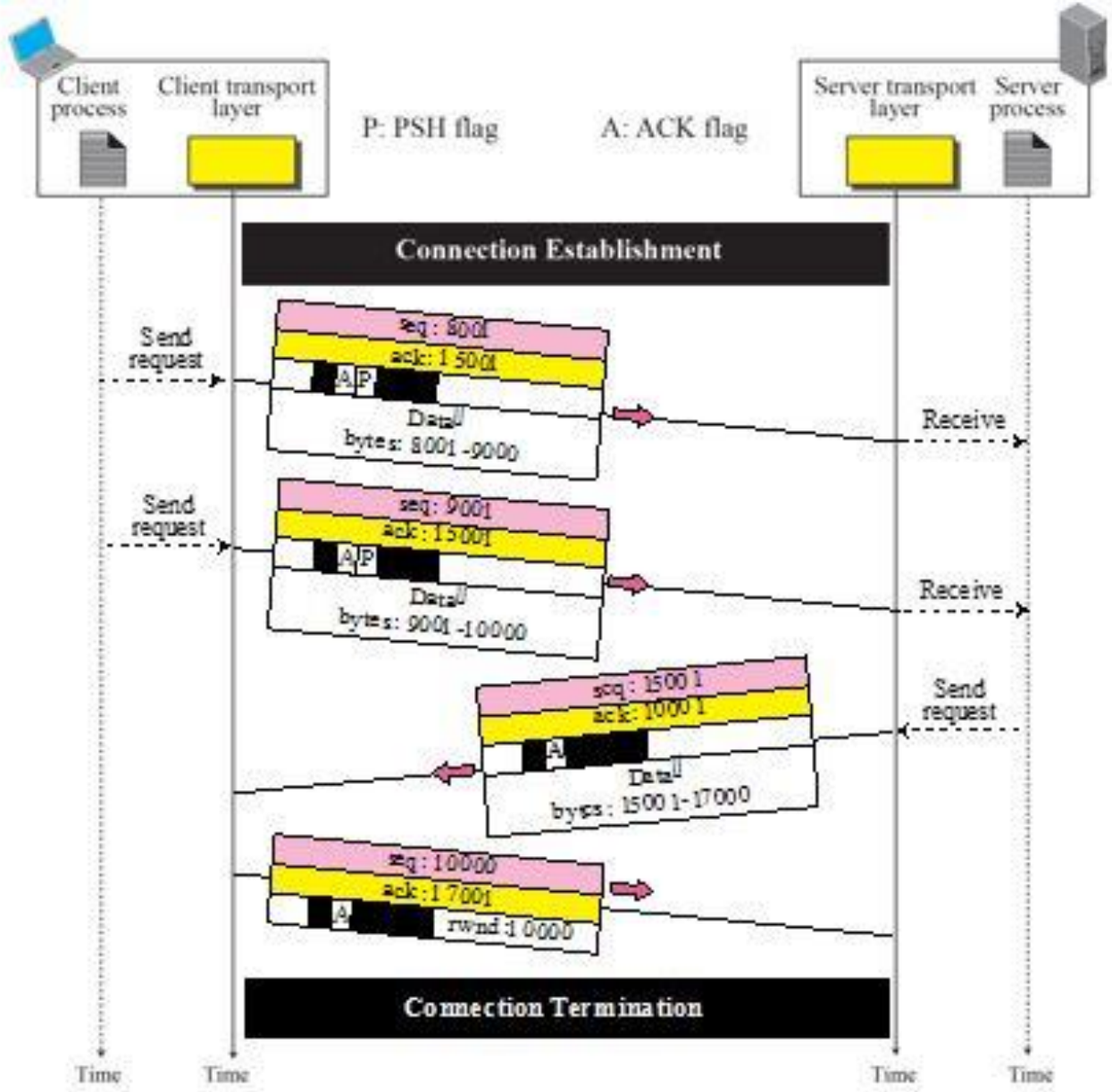



Figure: Connection establishment using three way handshaking

# Data Transfer

- After connection is established, bidirectional data takes place.
- The client and server can both send data and acknowledgments.
- The acknowledgment is piggybacked with data.


Client process
Client transport layer
P: PSH flag
A: ACK flag
Server transport layer
Server process
Connection Establishment
Send request
seq : 8001
ack: 1 5001
A P
Data
bytes: 8001 -9000
Receive
Send request
seq: 9001
ack: 15001
A P
Data
bytes: 9001 -10000
Receive
seq : 1500 1
ack: 1000 1
Send request
A
Data
byes : 1500 1- 17000
seq : 10000
ack : 1 7001
A
rwnd :1 0000
Connection Termination
Time
Time
Time
Time

- After connection is established, the client sends 2000 bytes of the data in two segment.
- The server sends 2000 bytes in one segment
- The client sends one more segment.
- The first three segments carry both data and acknowledgment, but the last segment carries only an acknowledgment because there are no more data to be sent.

- The data segment sent by the client have the PSH (push) flag set so that the server TCP knows to deliver data to the server process as soon as they are received.

# Connection Termination

- Either client or server can close the connection after exchange of data; but usually initiated by the client.
- Mostly it is implemented  with three-way handshaking.

**Step 1**:

- The client TCP, after receiving a close command from the client process, sends the first segment, a **FIN segment** in which the FIN flag is set.
- It can contain the last chunk of data sent by client.
- The FIN segment consumes one sequence number if it does not carry data.

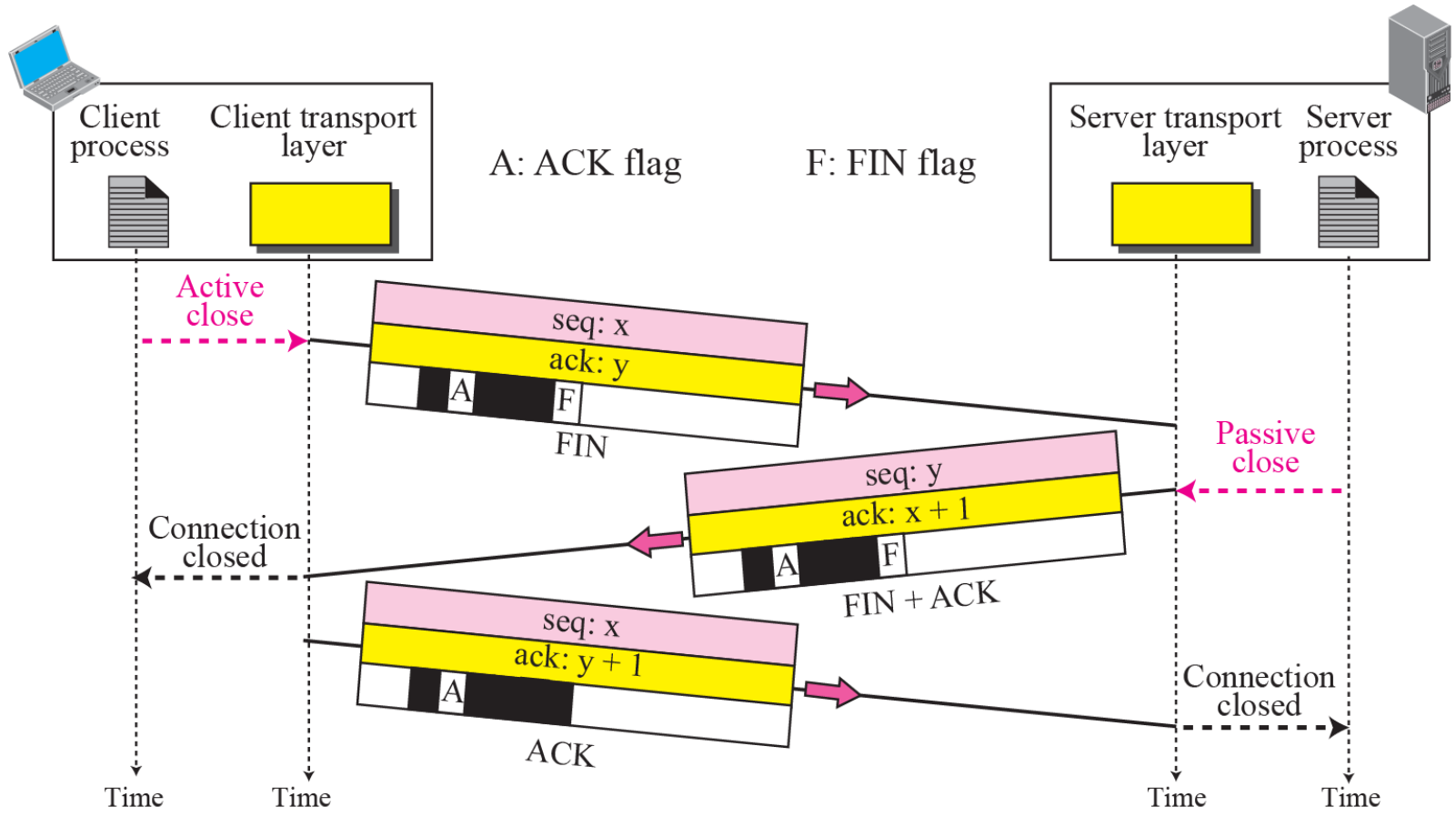
**Step 2**:

- The sever TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, **a FIN+ACK segment**, to confirm the receipt of the FIN segment from the client and at the same time to announce the closing of the connection
- This segment can also contained the last chunk of data from server.
- The FIN+ACK segment consumes one sequence number if it does not carry data.

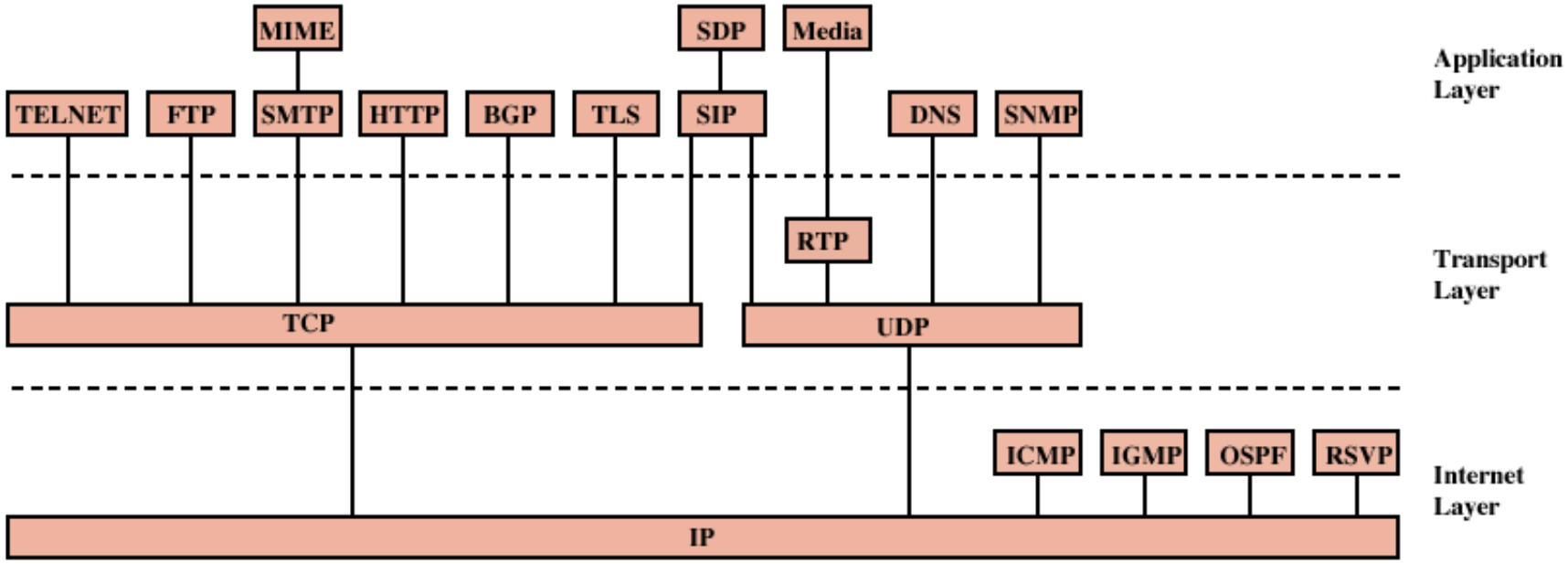
**Step 3**:

- The client TCP sends the last segment, an **ACK segment**, to confirm the receipt of the FIN segment from the TCP server.
- This segment cannot carry data and consumes no sequence numbers.

# Connection Termination



# Services in TCP/IP




BGP = Border Gateway Protocol
DNS = Domain Name System
FTP = File Transfer Protocol
HTTP = Hypertext Transfer Protocol
ICMP = Internet Control Message Protocol
IGMP = Internet Group Management Protocol
IP = Internet Protocol
MIME = Multi-Purpose Internet Mail Extension
OSPF = Open Shortest Path First
RSVP = Resource ReSerVation Protocol
RTP = Real-Time Transport Protocol
SDP = Session Description Protocol
SIP = Session Initiation Protocol
SMTP = Simple Mail Transfer Protocol
SNMP = Simple Network Management Protocol
TCP = Transmission Control Protocol
TLS = Transport Layer Security
UDP = User Datagram Protocol

# Well Known port in TCP

| *Port* | *Protocol* | *Description* |
|---|---|---|
| 7 | Echo | Echoes a received datagram back to the sender |
| 9 | Discard | Discards any datagram that is received |
| 11 | Users | Active users |
| 13 | Daytime | Returns the date and the time |
| 17 | Quote | Returns a quote of the day |
| 19 | Chargen | Returns a string of characters |
| 20 and 21 | FTP | File Transfer Protocol (Data and Control) |
| 23 | TELNET | Terminal Network |
| 25 | SMTP | Simple Mail Transfer Protocol |
| 53 | DNS | Domain Name Server |
| 67 | BOOTP | Bootstrap Protocol |
| 79 | Finger | Finger |
| 80 | HTTP | Hypertext Transfer Protocol |