# MODULE 4
# Transport Layer

Sneha M
AP(IT)

# Transport layer

- the transport layer is to provide services for the end to end communication for the various operating applications.
- TCP (transmission control protocol) is the most widely used transport protocol and so the internet protocol suite has been named after it i.e., the TCP/ IP.
- The ultimate goal of the transport layer is to provide efficient, reliable, and cost-effective data transmission service to its users.
- The software and/or hardware within the transport layer that does the work is called the transport entity

- The (logical) relationship of the network, transport, and application layers is illustrated

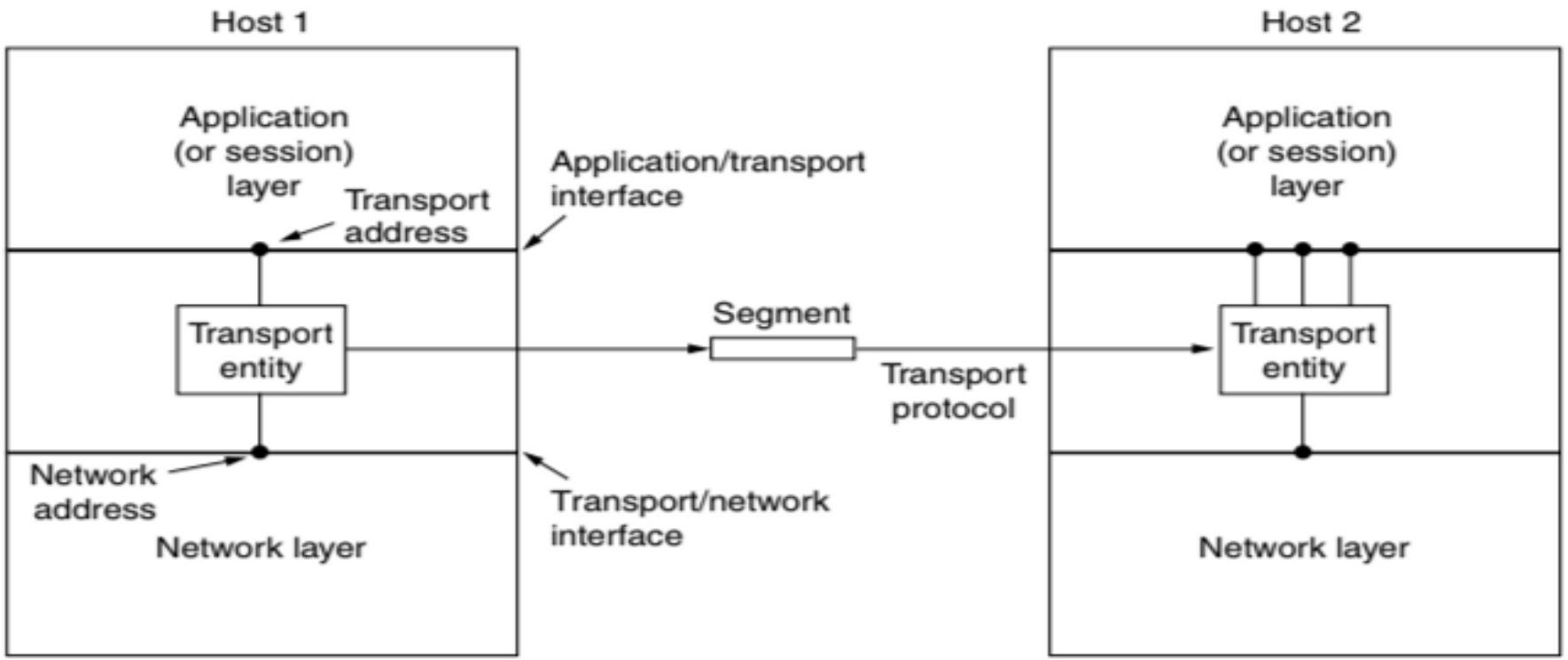
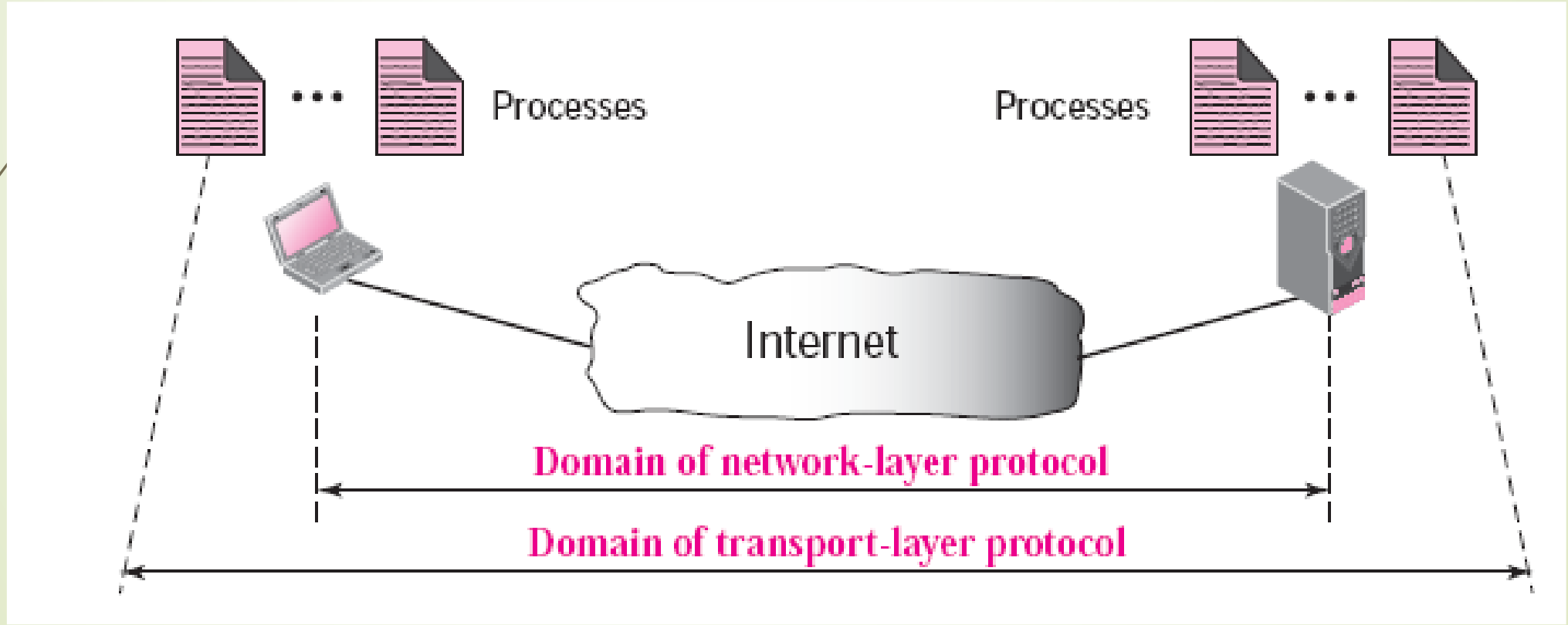



**Figure 6-1.** The network, transport, and application layers.

# Process-to-process communication

- A **transport layer** protocol is responsible for **delivery of the message** to the **appropriate process.**

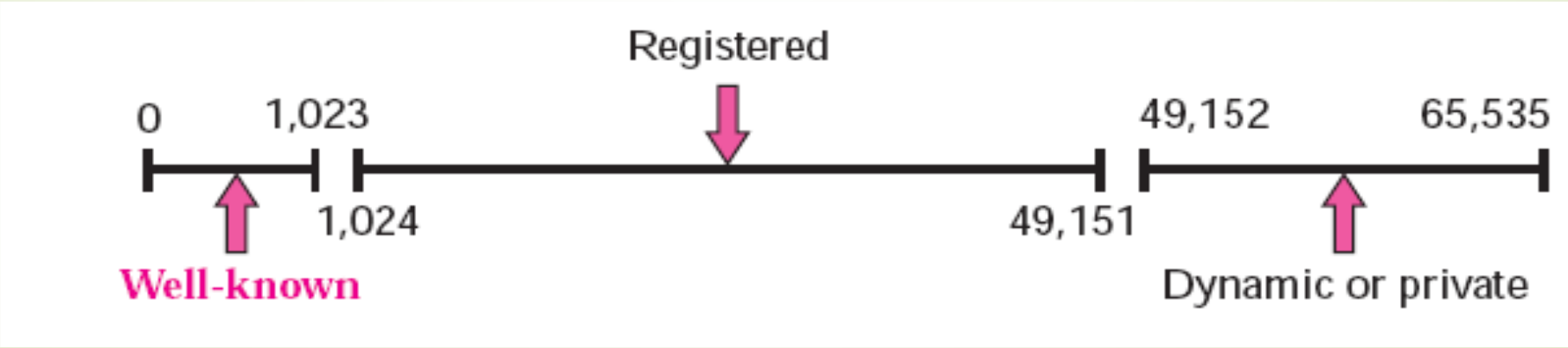


## Addressing: Port Numbers

- To define the processes, need second identifiers, called **port numbers**
- In the TCP/IP protocol suite, the port numbers are integers between 0 and 65,535 (16 bits).

# IANA and ICANN ranges

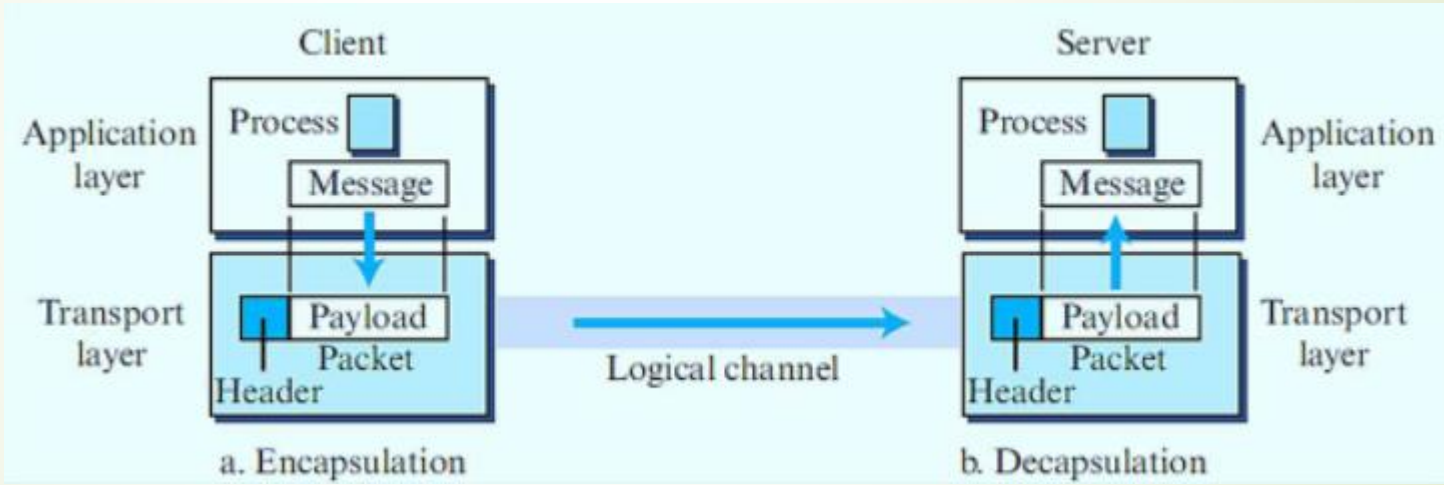
- The **Internet Assigned Numbers Authority (IANA)** and **Internet Corporation for Assigned Names and Numbers (ICANN)** are the central bodies that assigns and tracks port numbers.
- Divided the 65,535 numbers into three ranges : **well-known, registered** and **dynamic (or private)**



# Socket Address

- Connections between applications are made between {source IP address, source port} and {destination IP address, destination port} pairs
- **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 server socket address defines the server process uniquely.

# Encapsulation and Decapsulation



# Multiplexing and Demultiplexing

- Whenever an entity accepts items from more than one source, this is referred to as multiplexing (many to one).
- Whenever an entity delivers items to more than one source, this is referred to as demultiplexing (one to many).
- The transport layer at the source performs multiplexing; the transport layer at the destination performs demultiplexing.

# Flow Control

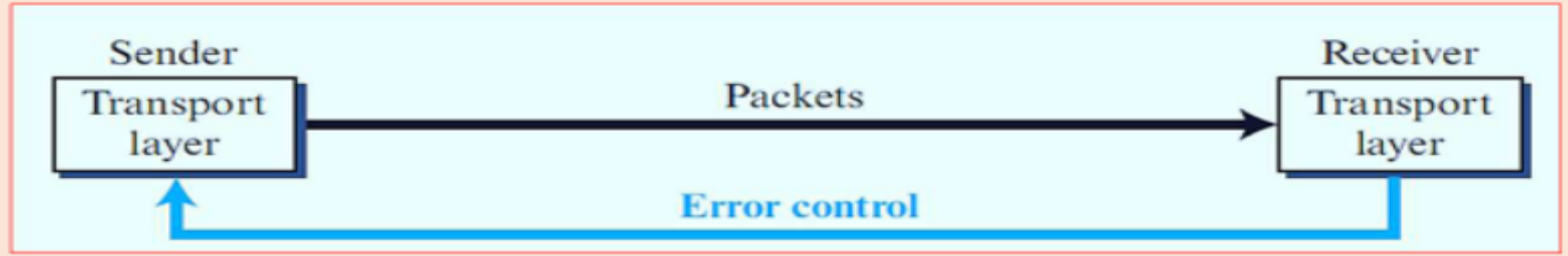
- **Delivery of items** from a producer to a consumer can occur in one of two ways: **pushing** or **pulling**
- If the sender delivers items whenever they are produced without a prior request from the consumer, the delivery is referred to as pushing.
- If the producer delivers the items after the consumer has requested them, the delivery is referred to as pulling.
- When the producer pushes the items, the consumer may be overwhelmed and there is a need for flow control, in the opposite direction, to prevent discarding of the items.
- The transport layer is responsible for flow control. It uses the sliding window protocol that makes the data transmission more efficient as well as it controls the flow of data so that the receiver does not become overwhelmed.

# Congestion avoidance

- Traffic entry in to the network can be controlled by means of congestion control by avoiding congestive collapse. The network might be kept in a state of congestive collapse by automatic repeat requests
- Congestive-Avoidance Algorithms (CAA) are implemented at the TCP layer as the mechanism to avoid congestive collapse in a network.

# Error Control

- Error control at the transport layer is responsible for
  - 1. Detecting and discarding corrupted packets.
  - 2. Keeping track of lost and discarded packets and resending them.
  - 3. Recognizing duplicate packets and discarding them.
  - 4. Buffering out-of-order packets until the missing packets arrive.



# Reliability:

- Packets may be lost during transport due to network congestion and errors. By means of an error detection code, such as a checksum, the transport protocol may check that the data is not corrupted, and verify correct receipt by sending an ACK or NACK message to the sender. Automatic repeat request schemes may be used to retransmit lost or corrupted data

# Transport Service primitives

| Primitive | Packet sent | Meaning |
|---|---|---|
| LISTEN | (none) | Block until some process tries to connect |
| CONNECT | CONNECTION REQ. | Actively attempt to establish a connection |
| SEND | DATA | Send information |
| RECEIVE | (none) | Block until a DATA packet arrives |
| DISCONNECT | DISCONNECTION REQ. | Request a release of the connection |

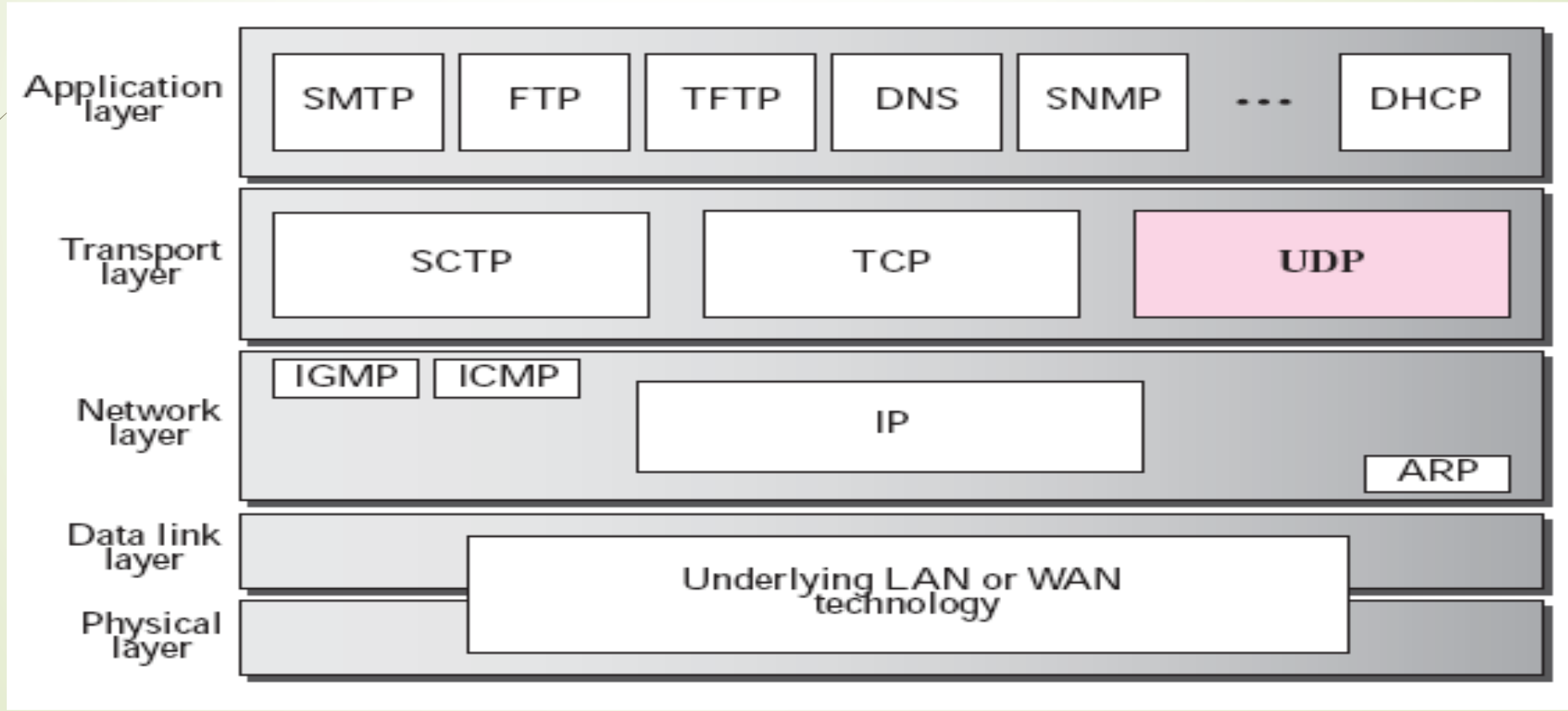
**Figure 6-2.** The primitives for a simple transport service.

- Transport service interface allow users to access the transport service by providing some operations to application programs.
- Each transport service has its own interface.
- Service primitives allows application programs to establish, use, and then release connections, which is sufficient for many applications.

- **Working**
- To start with, the server executes a LISTEN primitive, typically by calling a library procedure that makes a system call that blocks the server until a client turns up.
- When a client wants to talk to the server, it executes a CONNECT primitive.
- The transport entity carries out this primitive by blocking the caller and sending a packet to the server.
- Encapsulated in the payload of this packet is a transport layer message for the server's transport entity.

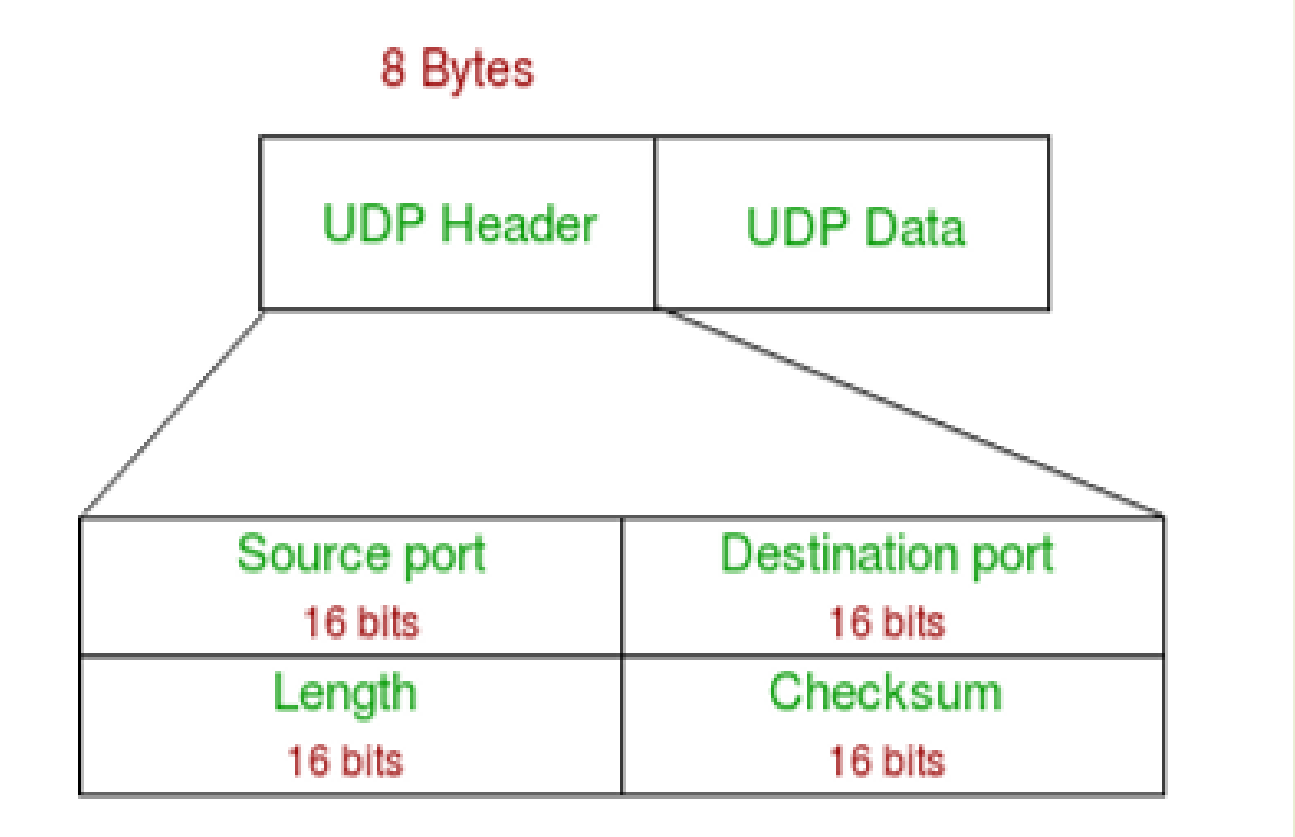
# The User Datagram Protocol (UDP)

- The User Datagram Protocol (UDP) provides an unreliable, best effort, connectionless delivery service using IP to transport messages between machines



- A connectionless, unreliable and unordered transport protocol
- Create process-to-process communication
- Allows an application to send user datagram to other application on the remote machine
- No flow control, No congestion control, No acknowledgement , No retransmission upon receipt of a bad segment.
- Delivery and duplicate detection are not guaranteed.
- Low overhead: faster than TCP
- For real time applications

# UDP segment structure



- **UDP Header-**

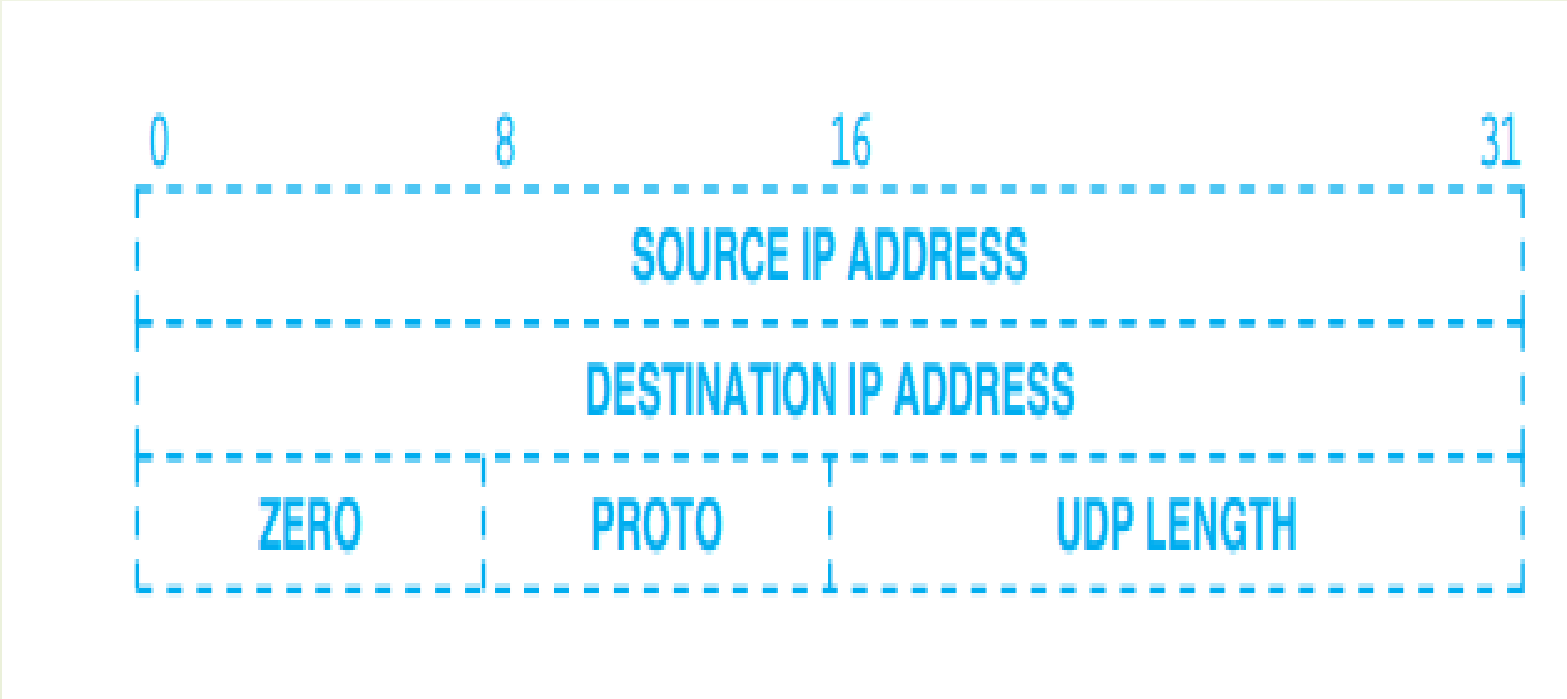
UDP header is an **8-bytes** fixed and simple header. The first 8 Bytes contains all necessary header information and the remaining part consist of data. UDP port number fields are each 16 bits long, therefore the range for port numbers is defined from 0 to 65535; port number 0 is reserved. Port numbers help to distinguish different user requests or processes.

- **Source Port:** Source Port is a 2 Byte long field used to identify the port number of the source.
- **Destination Port:** It is a 2 Byte long field, used to identify the port of the destined packet.
- **Length:** Length is the length of UDP including the header and the data. It is a 16-bits field. **UDP Length** : UDP header + data

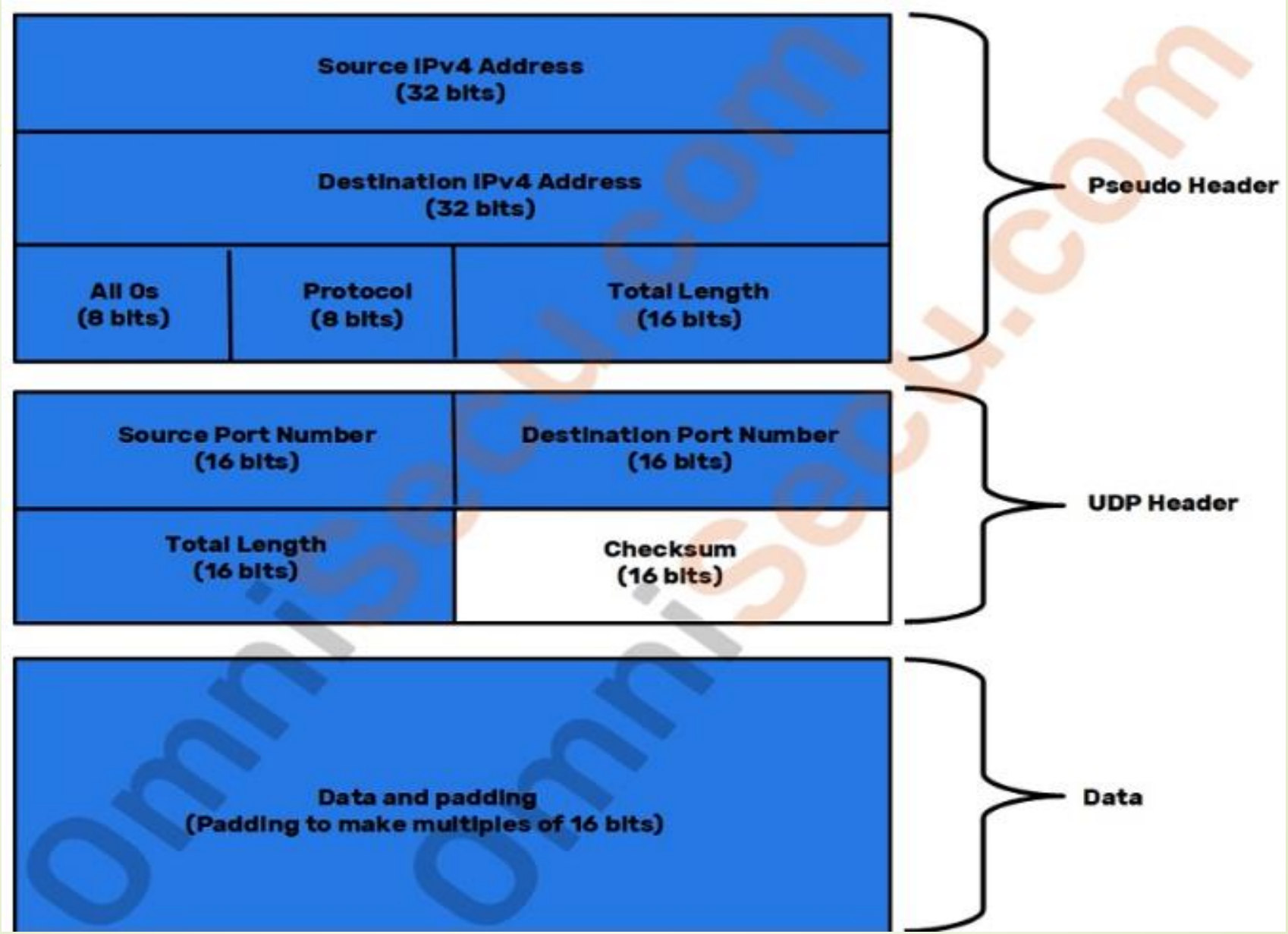
- **Checksum:** Checksum is 2 Bytes long field. ***optional*** 16 bit field – used to detect errors.
  - If 0, then there is no checksum, else it is a checksum calculation over **a pseudo header** + UDP header + data area

# PSEUDO HEADER

## -For CHecksum

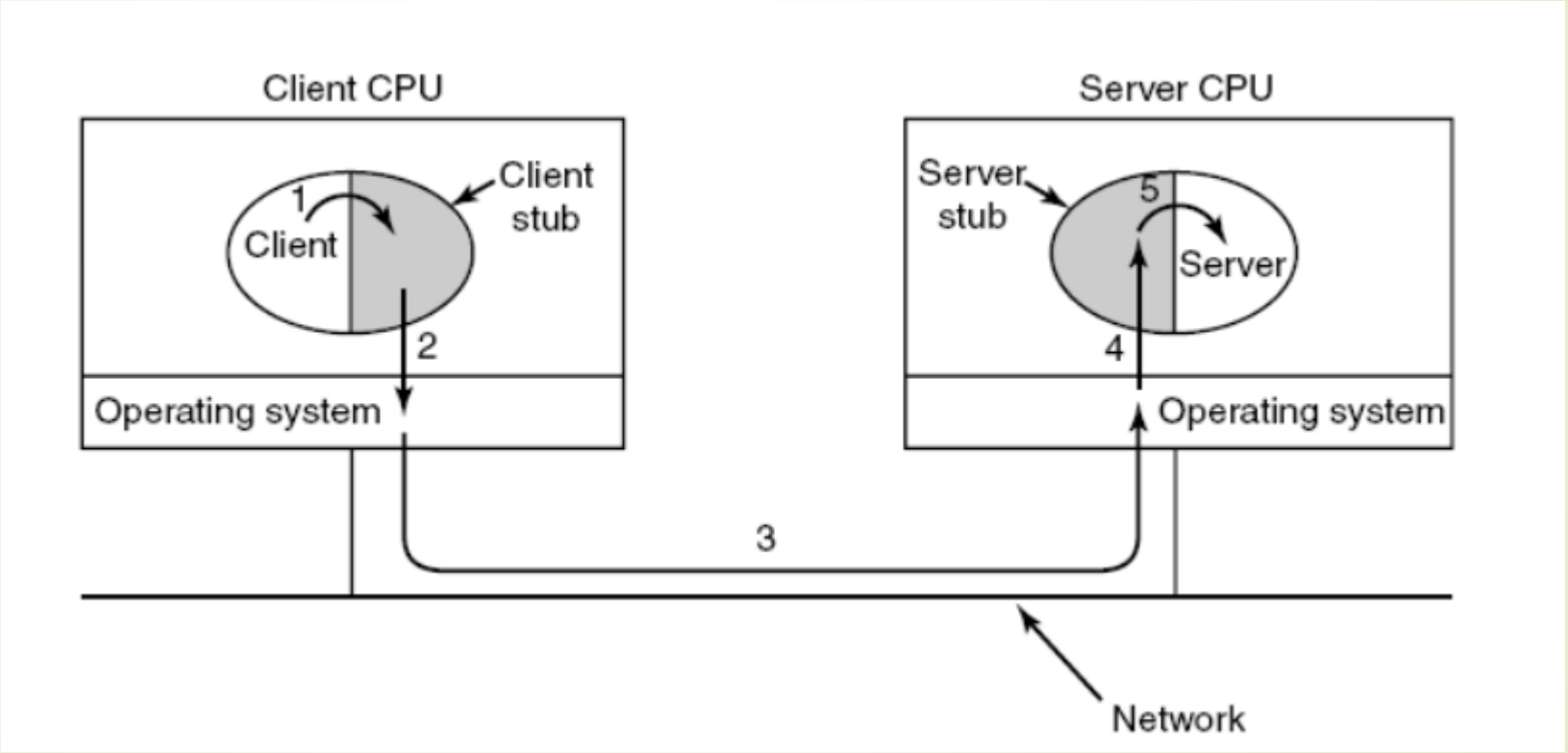


- UDP uses a concept called as "**pseudo header**".
- Checksum value is calculated after prepending a pseudo header to actual UDP header and data.
- Pseudo header **helps to find transfer bit errors**, **helps to detect misdelivered packets** and also to protect against other types of network errors like the **possibility of IP datagram reaching a wrong host.**

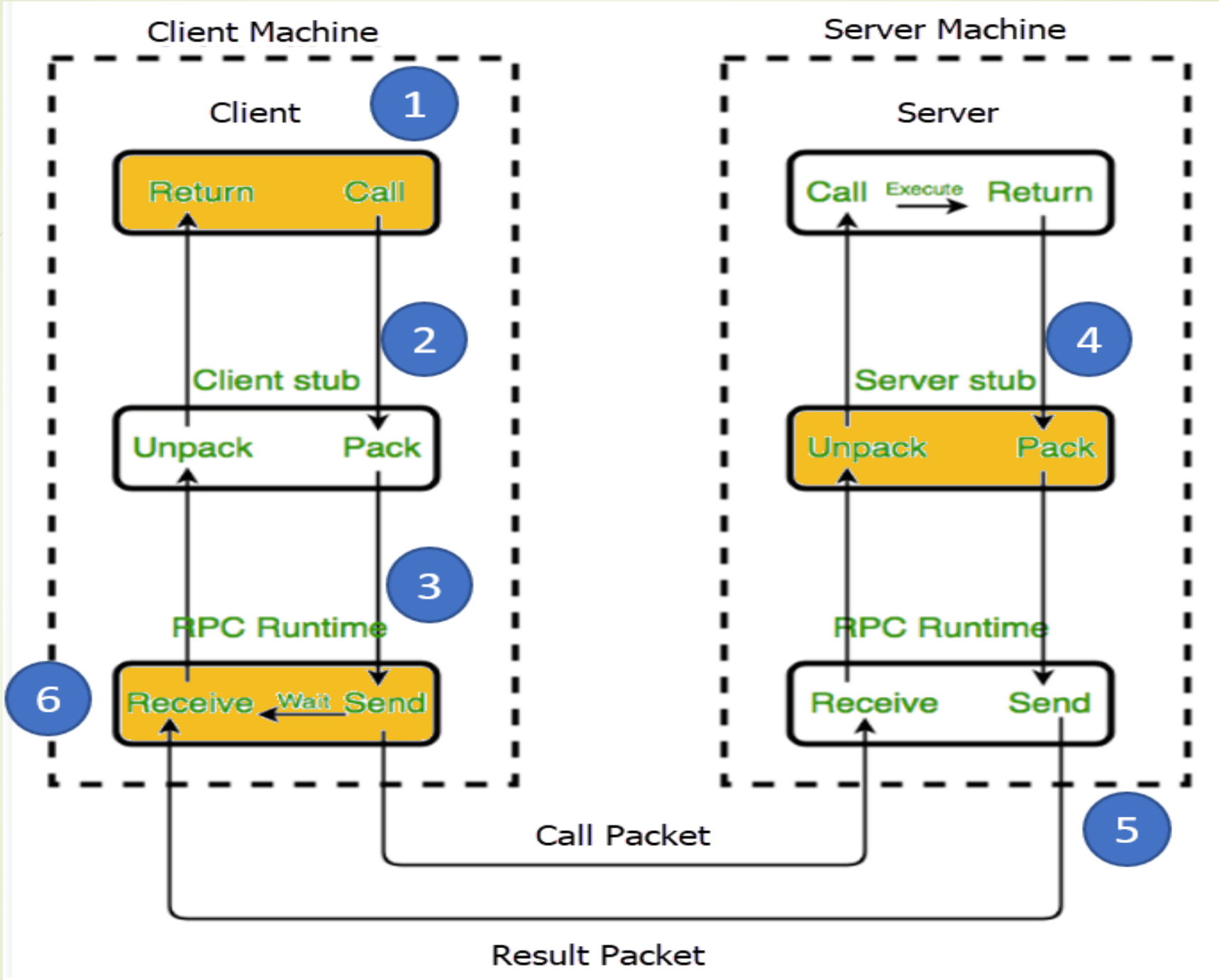

Source IPv4 Address
(32 bits)
Destination IPv4 Address
(32 bits)
All 0s
(8 bits)
Protocol
(8 bits)
Total Length
(16 bits)
Pseudo Header
Source Port Number
(16 bits)
Destination Port Number
(16 bits)
Total Length
(16 bits)
Checksum
(16 bits)
UDP Header
Data and padding
(Padding to make multiples of 16 bits)
Data

# Remote Procedure Call

- RPC is a protocol that **one program can use to request a service from a program located in another computer** on a network **without having to understand the network details.**
- RPC uses the **client-server model.**
- RPC spans transport layer as well as application layer.


Client CPU
Server CPU
Client
stub
1
Client
2
Server
stub
5
Server
4
Operating system
Operating system
3
Network

- It includes mainly five elements
  - **The client**
  - **The client stub** (Stub: piece of code used for converting parameters)
  - **RPC Runtime** (RPC communication Package)
  - **The server stub**
  - **The server**


Client Machine
Server Machine
Client
Server
1
Return
Call
Call
Execute
Return
2
4
Client stub
Server stub
Unpack
Pack
Unpack
Pack
3
RPC Runtime
RPC Runtime
6
Receive
Wait
Send
Receive
Send
5
Call Packet
Result Packet

## The Client :

- It is user process which initiates a RPC.
- The client makes a normal call that invokes a corresponding procedure in the client stub.

## The Client Stub :

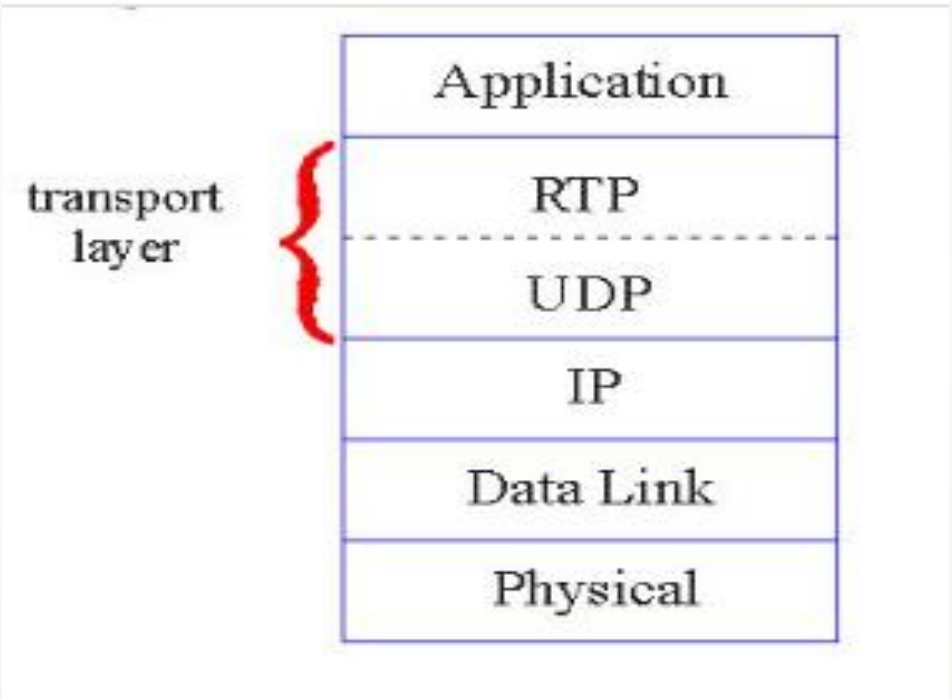
- To call a remote procedure, the client program must be bound with a small library procedure, called the **Client Stub**.
- On receipt of a request, it packs the parameters in the requirement into a message and asks RPC runtime to send.
- **Packing the parameters is called marshaling.**
- On receipt of a result it **unpacks the result** and passes it to the client – **Unmarshalling**

- RPC Runtime:
  - It handles the transmission of message between client and server
- The Server Stub :
  - Server bound with a procedure – **server stub**
  - It unpacks the call request – unmarshalling and make a normal call to the server
  - Marshalling the reply message
- The Server :
  - Executes an appropriate procedure and returns the result to packing

# Real-Time Transport Protocols

- A **protocol is designed to handle real-time traffic** (like audio and video) of the Internet
- RTP is used to transmit digitized audio or video signals over an IP internet
- Eg : real time multimedia applications like Internet radio, internet telephony, music-on-demand, video conferencing, video-on-demand etc
- RTP can provide an **end-to-end delivery services**
- Primarily designed for **multi-user multimedia applications**
- RTP **does not ensure timely delivery of data**
- **RTP run over UDP**

# Position of RTP in the protocol stack



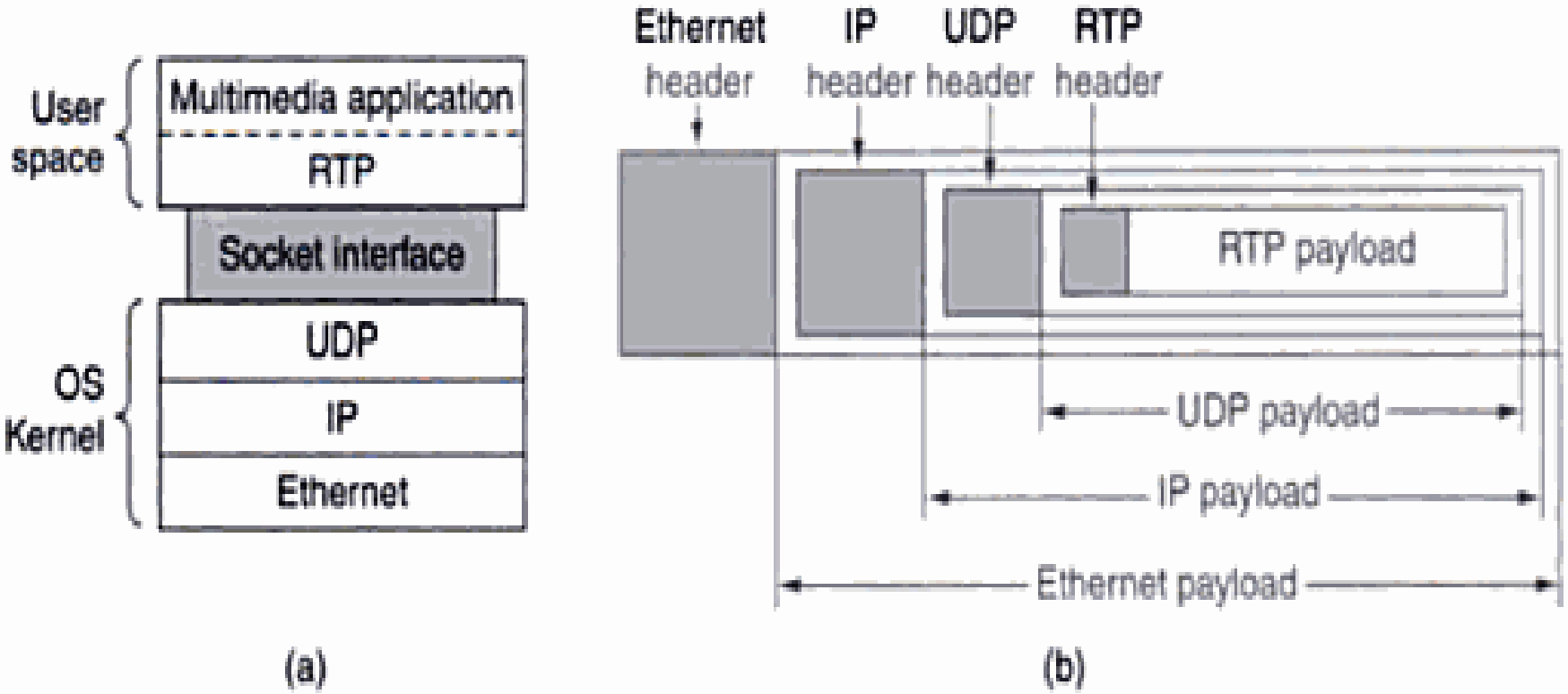
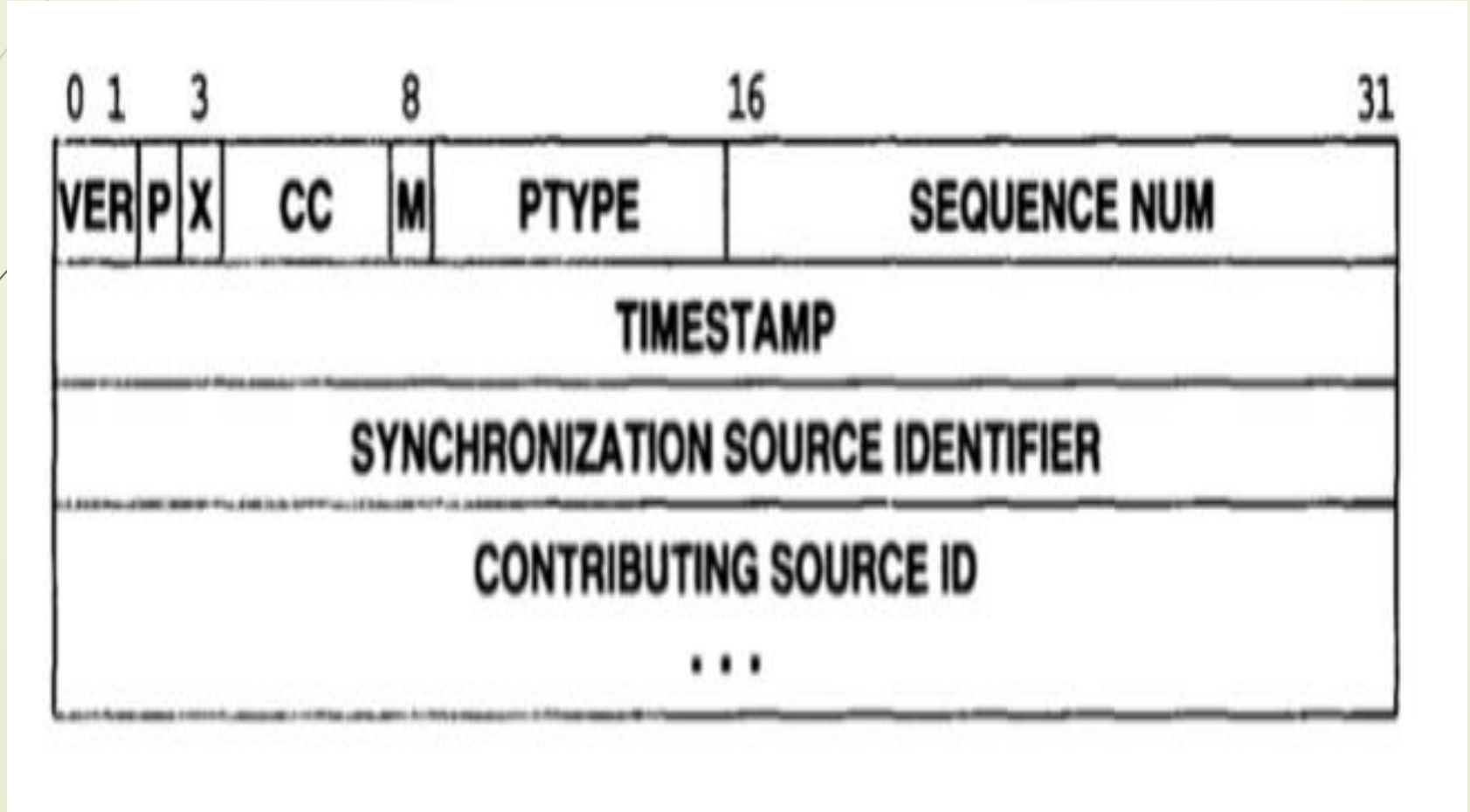



**Figure 6-25.** (a) The position of RTP in the protocol stack. (b) Packet nesting.

# RTP header


0 1 3 8 16 31
VER
P
X
CC
M
PTYPE
SEQUENCE NUM
TIMESTAMP
SYNCHRONIZATION SOURCE IDENTIFIER
CONTRIBUTING SOURCE ID
. . .

- RTP's header format is very simple and supports all real-time applications. RTP headers are made up of 32-bit words with data including Ver., P, X, CC, M, a Payload type, Sequence Number, Timestamp, Synchronization source identification, and Contributing source identity

**1. Version field**

It defines the protocol version.

**2. Padded bits**

The length of this field is 1-bit. If value is 1, then it denotes presence of padding at end of packet and if value is 0, then there is no padding.

**3. Extension header**

The length of this field is also 1 bit. If the value of this field is 1, it indicates that there is an extra extension header between the data and basic headers, and if the value is 0, there is no extra extension.

- **4. Contributor Count**

This 4-bit value represents the number of contributors. Because a 4-bit field can accept numbers ranging from 0 to 15, the maximum number of contributors is 15.

**5. Marker bit**

M in the header defines the marker bit, which is utilized to indicate a frame's start and end.

**6. Payload types**

This field is of length 7-bit to indicate type of payload.

**7. Sequence Number**

It indicates the quantity of RTP packets transmitted and grows by one value each time a packet is transmitted.

**8. Timestamp**

It has a length of 32 bits. It is utilized to determine the relationship between the timings of various RTP packets. The timestamp for the first packet is chosen at random, and the time stamp for subsequent packets is determined by adding the previous timestamp to the time required to produce the first byte of the current packet. The value of one clock tick differs depending on the app.

**9. Synchronization Source Identifier**

It describes the packet and the stream with which it is related.

**10. Contributing Source Identifier**

It is also a 32-bit variable that is utilized for source identification when many sources are present in the session. The mixer source is identified by the Synchronization source identity, while the Contributor identification identifies the remaining sources (up to 15).

# RTCP

- A companion protocol and integral part of RTP, known as the **RTP Control Protocol (RTCP)**, provides the needed control functionality
- It handles feedback, synchronization , monitor delivery and the user interface but does not transport any data
- The purpose of monitoring delivery is to determine whether RTP is providing the necessary Quality of Service (QoS) and to compensate for delays, if needed.
- RTCP is used in voice over IP (**VoIP**) and Internet Protocol Television (**IPTV**), **streaming media** and **video conferencing**.

# RTCP Operation

| Type | Meaning |
|---|---|
| 200 | Sender report |
| 201 | Receiver report |
| 202 | Source description message |
| 203 | Bye message |
| 204 | Application specific message |

**Figure 29.3** The five RTCP message types. Each message begins with a fixed header that identifies the type.

- The **sender report** is sent after a fixed interval by the active sender in a conference to report transmission as well as statistics of reception for all RTP packets transmitted during the time period. After receiving the RTP messages by the receiver, these details of absolute timestamps helps the receiver for synchronization process. And this is very much important in audio video transmission for finding the relative timestamp.

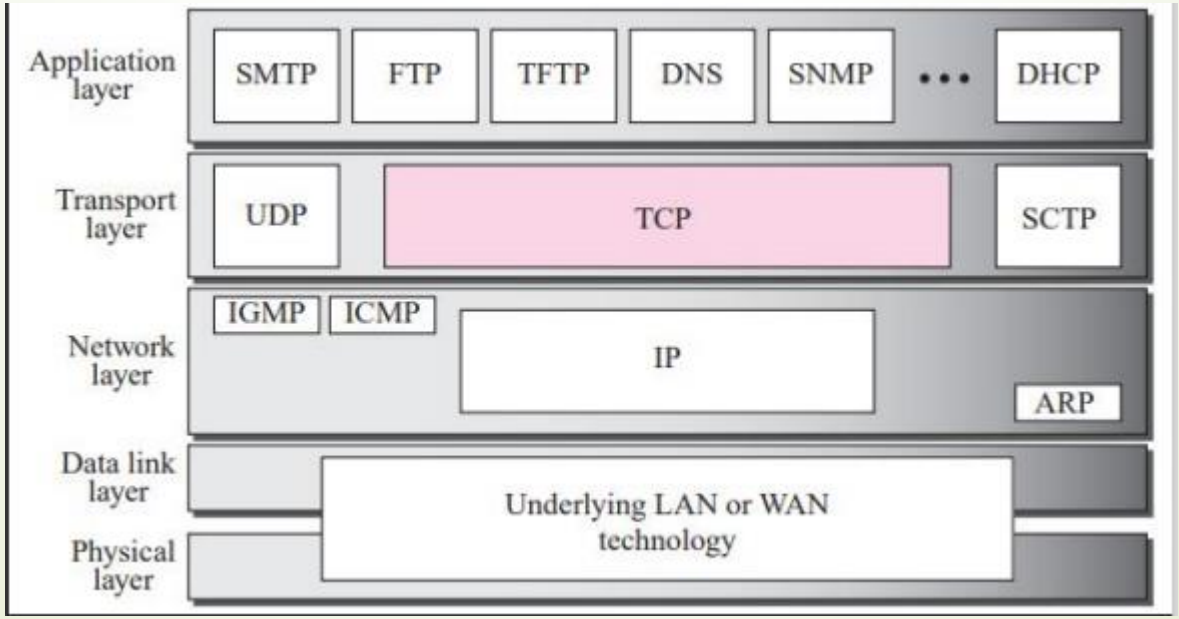
- **Receiver Report :** Passive participants are those participants that do not send RTP packets, and for them the Receiver report is used. This report is used to informs the sender and other receivers about the quality of service.
- **Source Description Message :** The source sends a source description message within a fixed interval to give some extra information about itself. It contains the details about the name of the source, its mail ID, contact number or source controller.

- **Bye Message :** To shut down a stream, a source sends a type of message which is known as Bye message. It is used by the source to announcing for leaving the conference. This message is a direct announcement for other sources about the absence of a source. It can be used for combining different media file.
- **Application-Specific Message :** If we want to make our application extensible then RTCP allows application-specific RTCP packets which is introduced by RTC 3611. It can be used to extend the type of application.

| TCP | UDP |
| --- | --- |
| Connection Oriented | Connectionless |
| Reliable | Less Reliable |
| Error Control | Error Detection |
| Slow Transmission | Fast Transmission |
| More Overhead | Less overhead |
| Flow Control | No Flow Control |
| Congestion Control | No Congestion Control |

# TCP

TCP (Transmission Control Protocol) is one of the main protocols of the Internet protocol suite. It lies between the Application and Network Layers which are used in providing reliable delivery services. It is a connection-oriented protocol for communications that helps in the exchange of messages between different devices over a network. The Internet Protocol (IP), which establishes the technique for sending data packets between computers, works with TCP.



# **services** provided by the TCP

- Process-to-Process Communication (using port numbers)
- Stream Delivery Service (segments)
- Full-Duplex Communication (Both direction)
- Multiplexing and Demultiplexing
- Connection-Oriented Service ( 3 step )
- Reliable Service

# process-to-process communication

- TCP provides process-to-process communication using port numbers
- TCP service is obtained by both the sender and the receiver creating endpoints, called **sockets**
- Each socket has a socket number (address) consisting of the IP address of the host and a 16-bit number local to that host, called a **port**
- For TCP service to be obtained, a connection must be explicitly established between a socket on one machine and a socket on another machine.
- Port numbers below 1024 are reserved for standard services that can usually only be started by privileged users. They are called **well-known ports**.

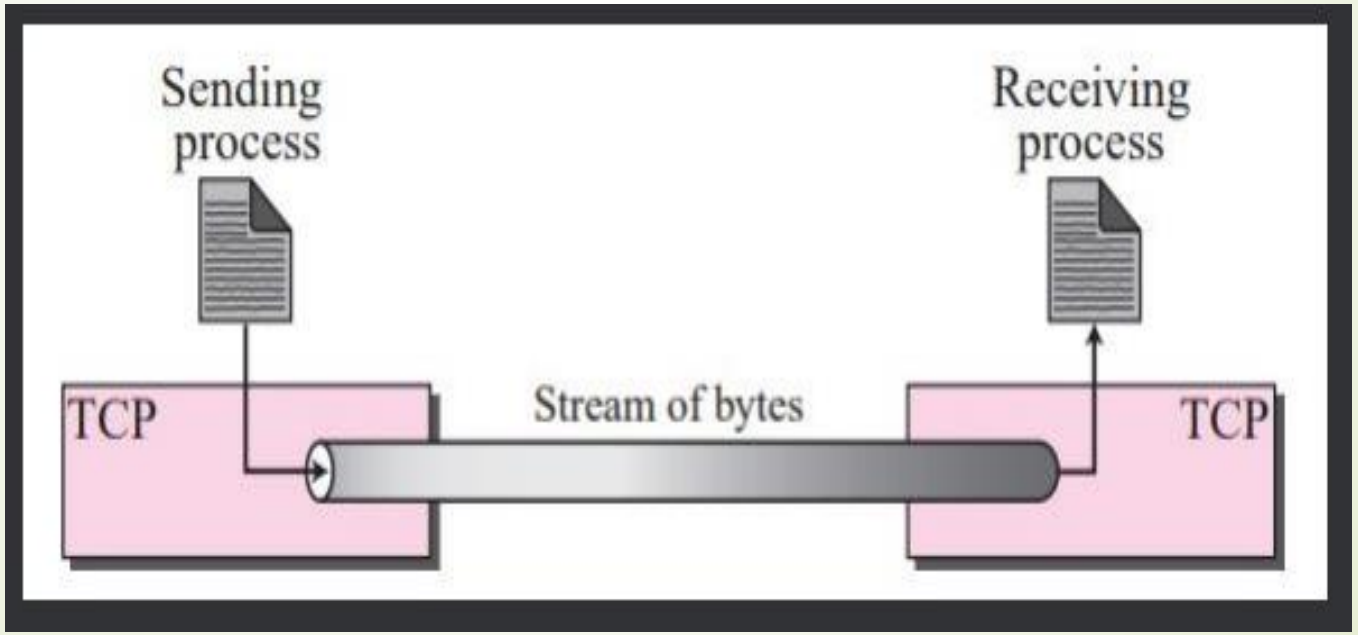
| Port | Protocol | Use |
|---|---|---|
| 20, 21 | FTP | File transfer |
| 22 | SSH | Remote login, replacement for Telnet |
| 25 | SMTP | Email |
| 80 | HTTP | World Wide Web |
| 110 | POP-3 | Remote email access |
| 143 | IMAP | Remote email access |
| 443 | HTTPS | Secure Web (HTTP over SSL/TLS) |
| 543 | RTSP | Media player control |
| 631 | IPP | Printer sharing |

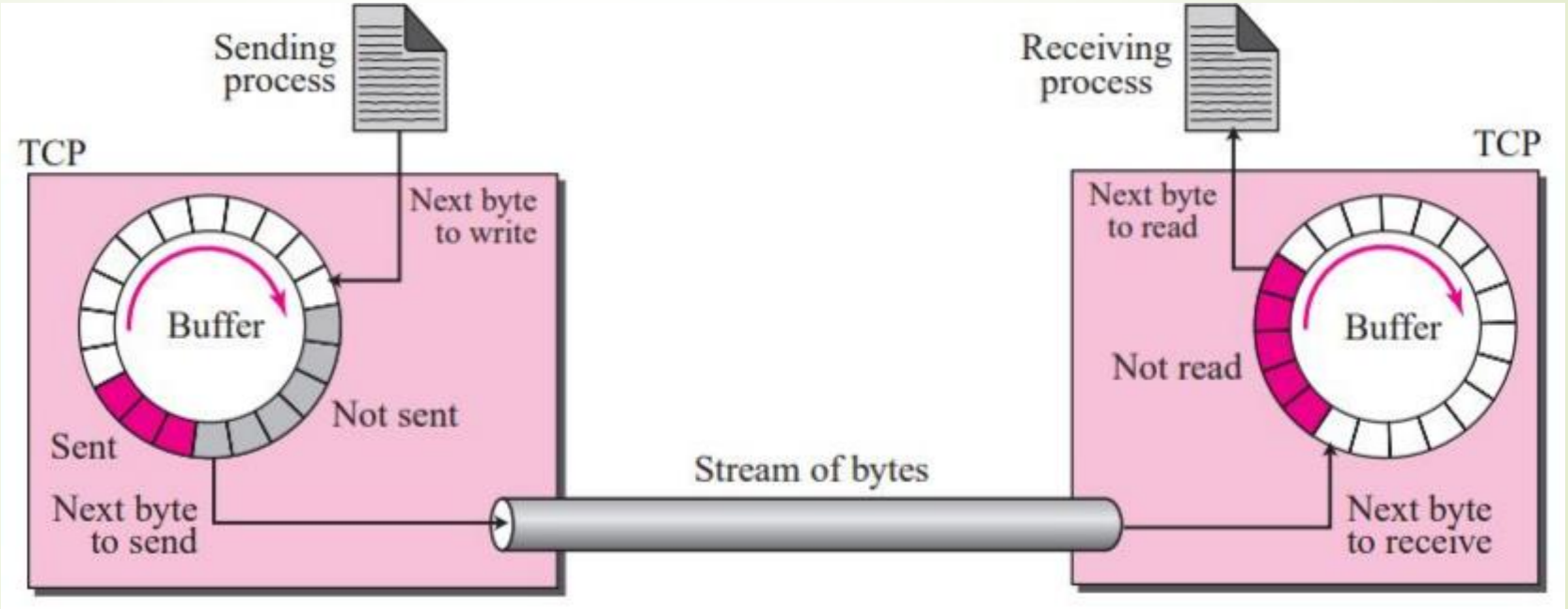
**Figure 6-34.** Some assigned ports.

# Stream Delivery Service

- TCP, unlike UDP, is a stream-oriented protocol.
- In UDP, a process sends messages with predefined boundaries to UDP for delivery,UDP adds its own header to each of these messages and delivers it to IP for transmission.
- TCP, on the other hand, allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes

- TCP creates an environment in which the two processes seem to be connected by an imaginary “tube” that carries their bytes across the Internet.




Sending process
Receiving process
TCP
TCP
Next byte to write
Next byte to read
Buffer
Buffer
Not sent
Not read
Sent
Stream of bytes
Next byte to send
Next byte to receive

# Full-Duplex Communication

- TCP offers full-duplex service, where data can flow in both directions at the same time.
- Each TCP endpoint then has its own sending and receiving buffer, and segments move in both directions

# Multiplexing and Demultiplexing

- Since TCP is a connection-oriented protocol, a connection needs to be established for each pair of processes.
- TCP does multiplexing and demultiplexing at the sender and receiver ends respectively as a number of logical connections can be established between port numbers over a physical connection

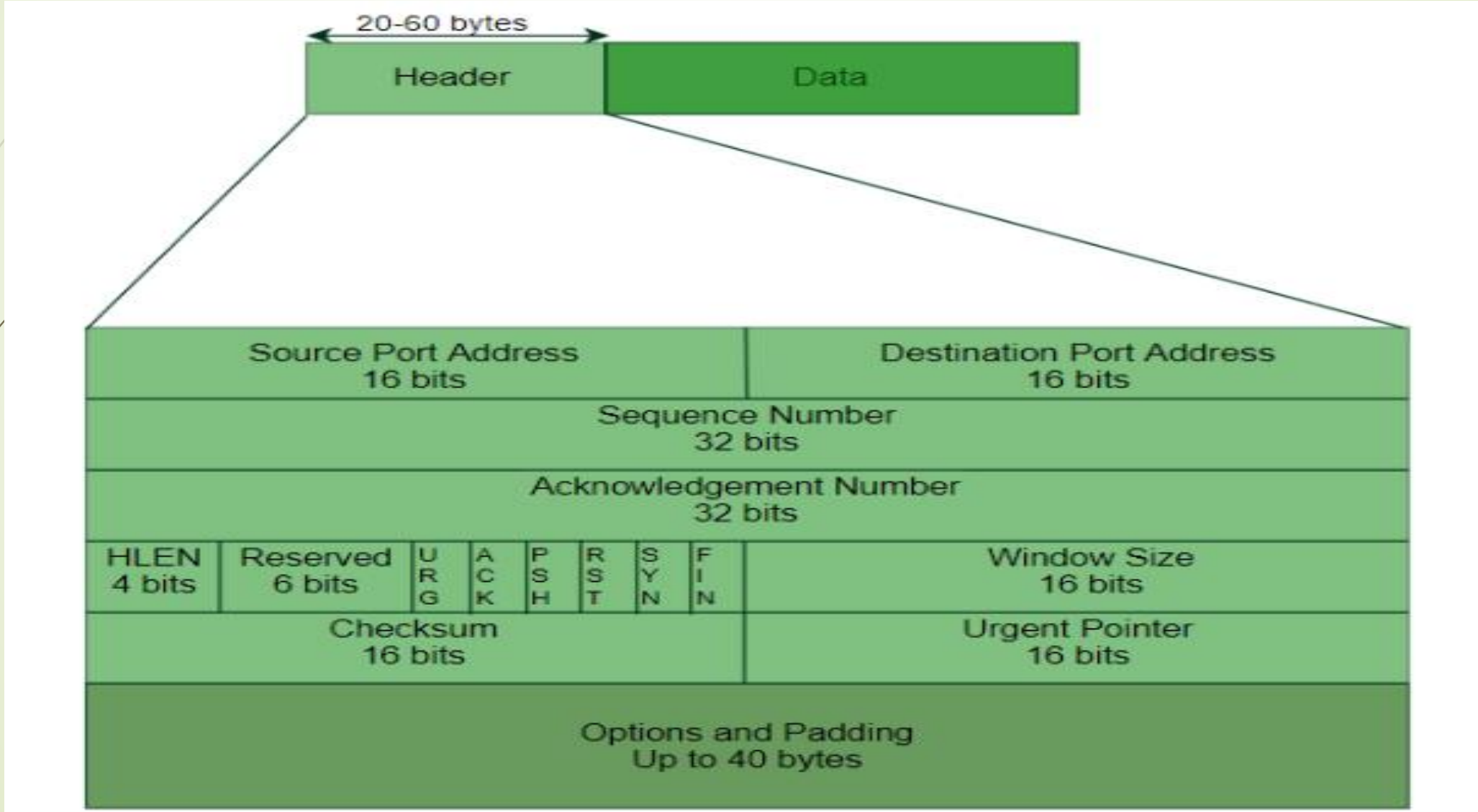
# Connection-Oriented Service

- When a process at site A wants to send to and receive data from another process at site B, the following three phases occur:
  1. The two TCPs establish a virtual connection between them.
  2. Data are exchanged in both directions.
  3. The connection is terminated.

# Reliable Service

- Although the TCP software keeps track of the segments being transmitted or received, there is no field for a segment number value in the segment header. Instead, there are two fields called the sequence number and the acknowledgment number.

# TCP Segment structure –



- **Source Port Address –**
  A 16-bit field that holds the port address of the application that is sending the data segment.

- **Destination Port Address –**
  A 16-bit field that holds the port address of the application in the host that is receiving the data segment.

- **Sequence Number –**
  This 32-bit field defines the number assigned to the first byte of data contained in this segment.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.

- **Acknowledgement 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. Acknowledgment and data can be piggybacked together
- **Header Length (HLEN) –**

  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 × 4 = 20) and 15 (15 × 4 = 60)

- Reserved: This is a 6-bit field reserved for future use.
- **Control flags –**
  These are 6 1-bit control bits that control connection establishment, connection termination, connection abortion, flow control, mode of transfer etc. Their function is:
  - URG: Urgent pointer is valid
  - ACK: Acknowledgement number is valid( used in case of cumulative acknowledgement)
  - PSH: Request for push
  - RST: Reset the connection
  - SYN: Synchronize sequence numbers
  - FIN: Terminate the connection

- **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.

- **Checksum –**
  This field holds the checksum for error control. It is mandatory in TCP as opposed to UDP

- **Urgent pointer –**
  This field (valid only if the URG control flag is set) is used to point to data that is urgently required that needs to reach the receiving process at the earliest. The value of this field is added to the sequence number to get the byte number of the last urgent byte.

# TCP - Connection establishment and Release

- TCP is connection-oriented, ie. establishes a virtual path between the source and destination.
- In TCP, connection-oriented transmission requires three phases:

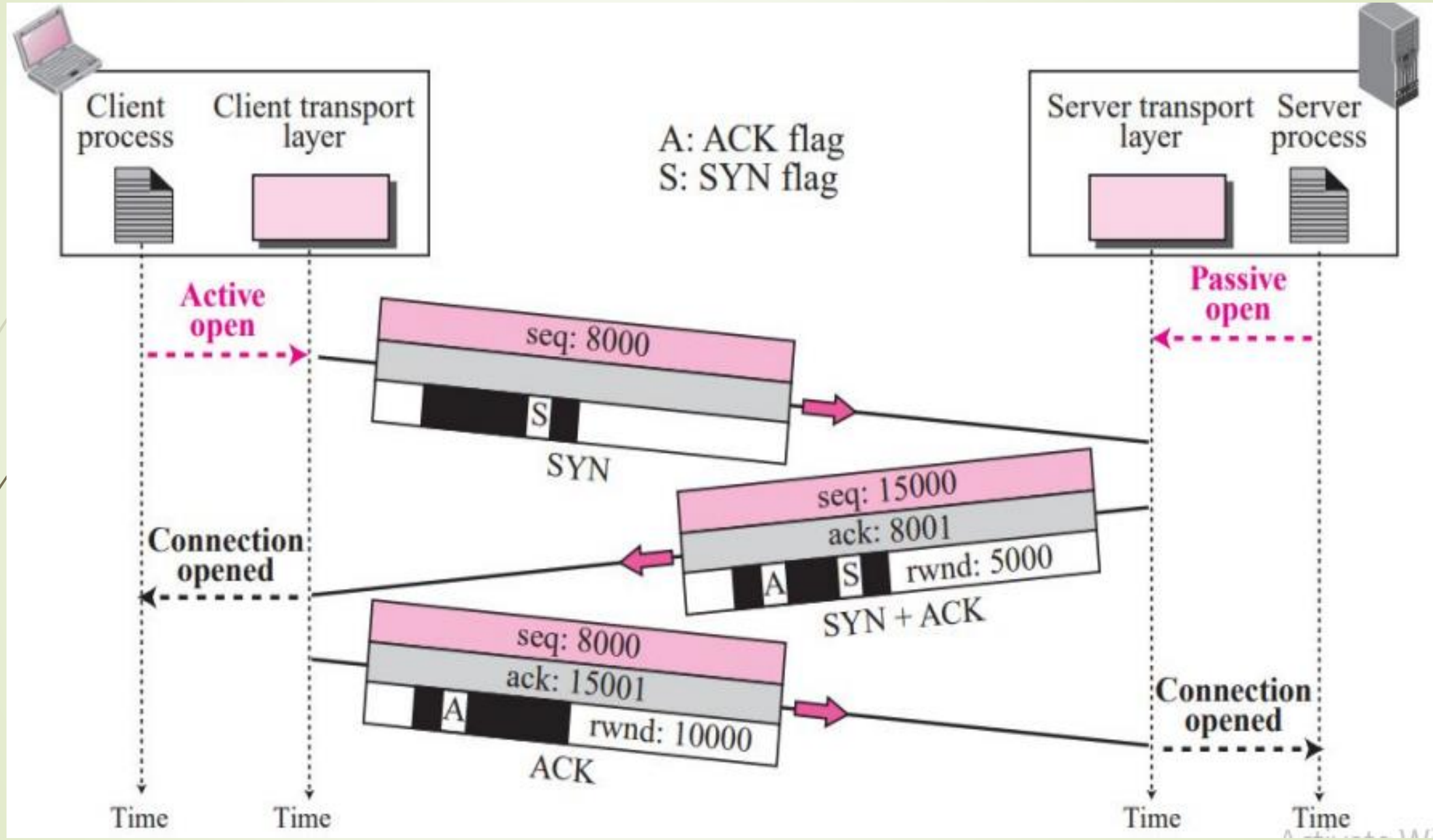
1. Connection establishment
2. Data transfer
3. Connection termination

# 1. Connection establishment

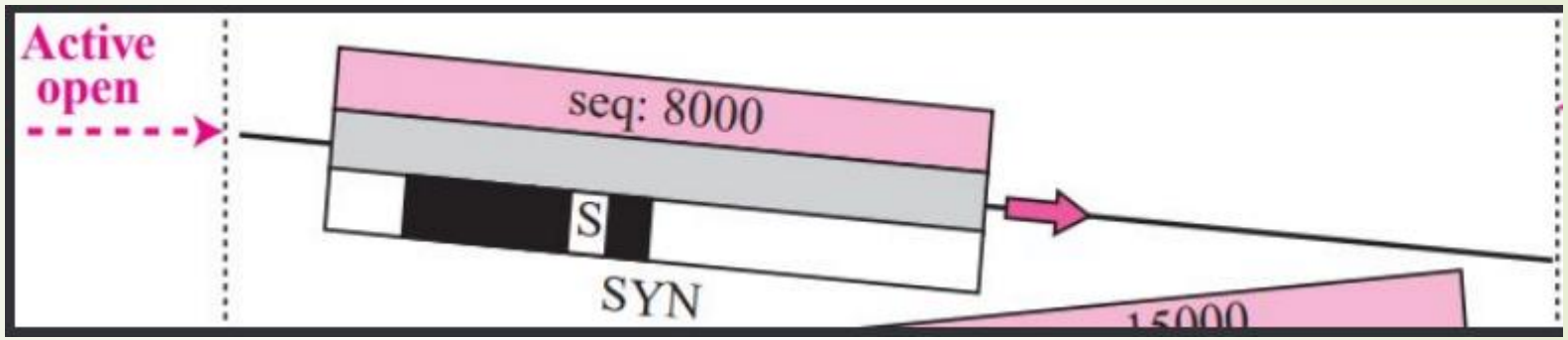
- The connection establishment in TCP is called three-way handshaking.

This process can be started using

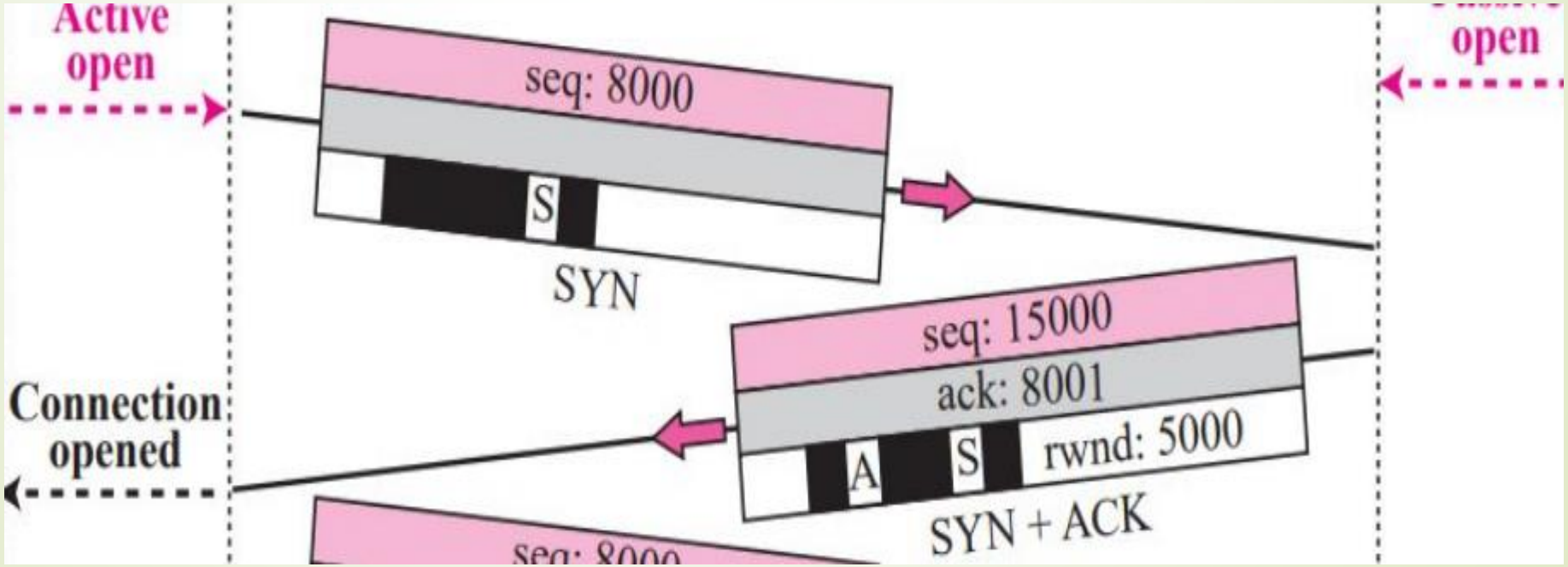
- Active open
- Passive Open


Client process
Client transport layer
A: ACK flag
S: SYN flag
Server transport layer
Server process
Active open
Passive open
seq: 8000
S
SYN
seq: 15000
ack: 8001
A
S
rwnd: 5000
SYN + ACK
Connection opened
seq: 8000
ack: 15001
A
rwnd: 10000
ACK
Connection opened
Time
Time
Time
Time

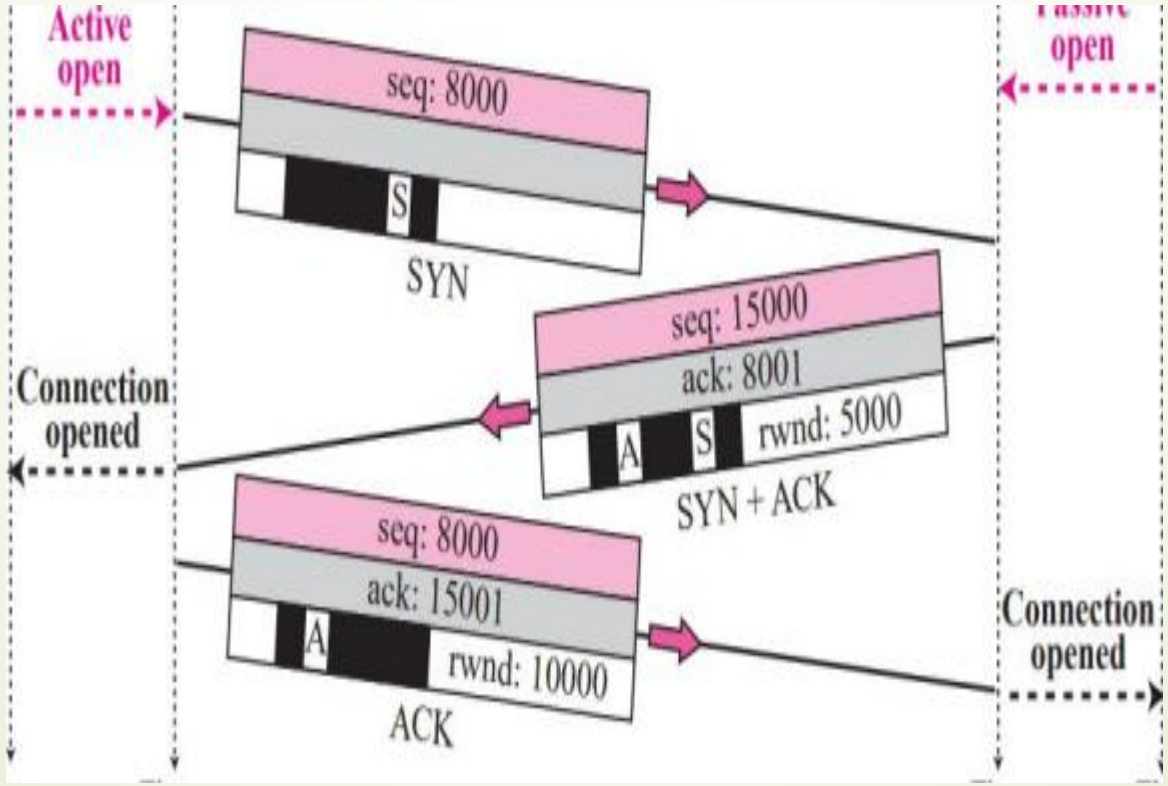
- The client sends the first segment, a SYN segment, in which only the SYN flag is set. The client in our example chooses a random number as the first
- sequence number and sends this number to the server. This sequence number is called the **initial sequence number (ISN).**
- Note that this segment does not contain an acknowledgment number. It does not define the window size either; a window size definition makes sense only when a segment includes an acknowledgment.



- The server sends the second segment, a SYN + ACK segment with two flag bits set: SYN and ACK. This segment has a dual purpose.
- First, it is a SYN segment for communication in the other direction. The server uses this segment to initialize a sequence number for numbering the bytes sent from the server to the client.
- Second ,The server also acknowledges the receipt of the SYN segment from the client by setting the ACK flag and displaying the next sequence number it expects to receive from the client. Because it contains an acknowledgment, it also needs to define the receive window size, rwnd (to be used by the client)



- The client sends the third segment. This is just an ACK segment. It acknowledges the receipt of the second segment with the ACK flag and
- acknowledgment number field. Note that the sequence number in this segment is the same as the one in the SYN segment; the ACK segment does not consume any sequence numbers. The client must also define the server window size.

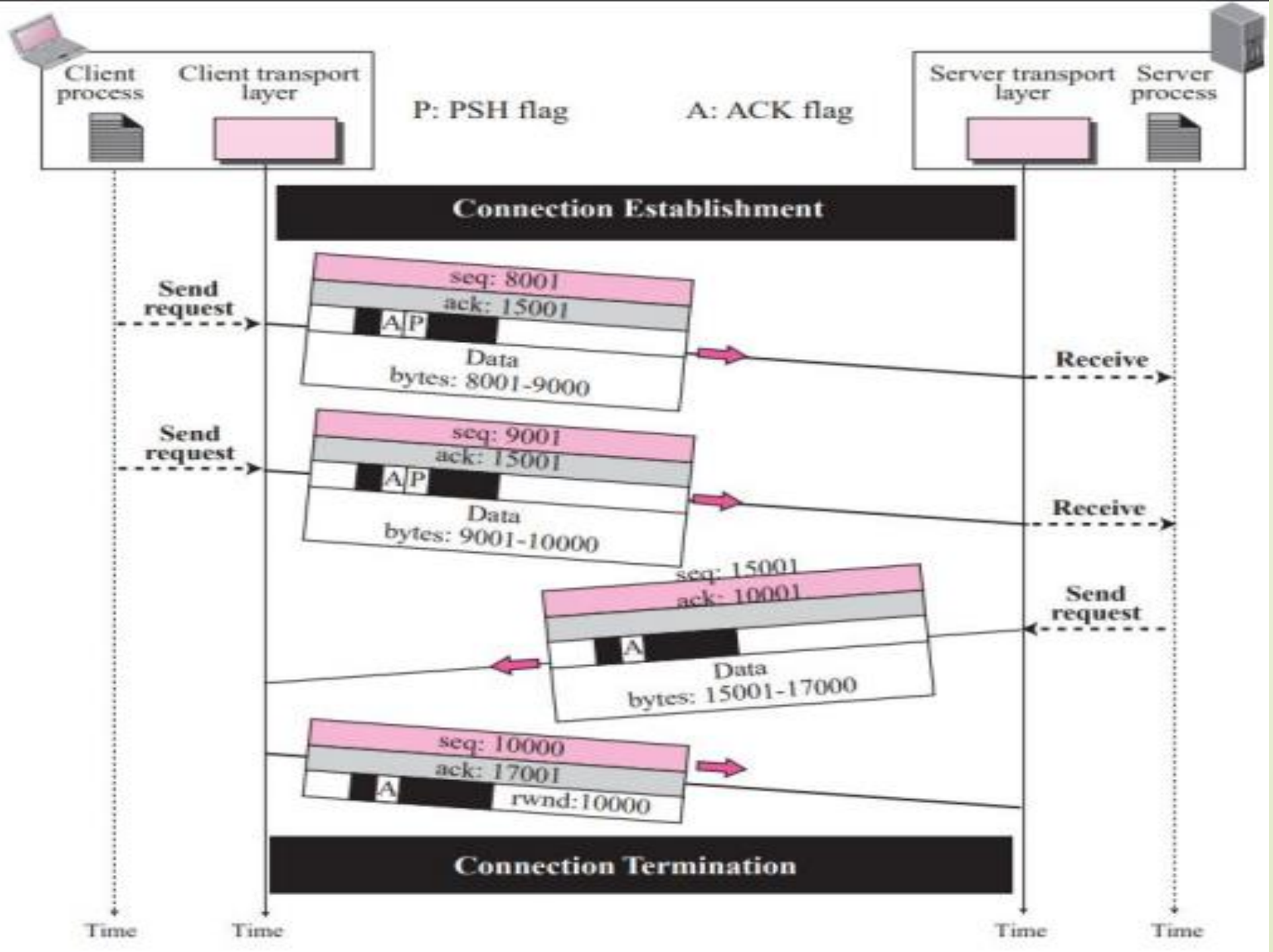


1. A SYN segment cannot carry data, but it consumes one sequence number.

2. A SYN + ACK segment cannot carry data, but does consume one sequence number.

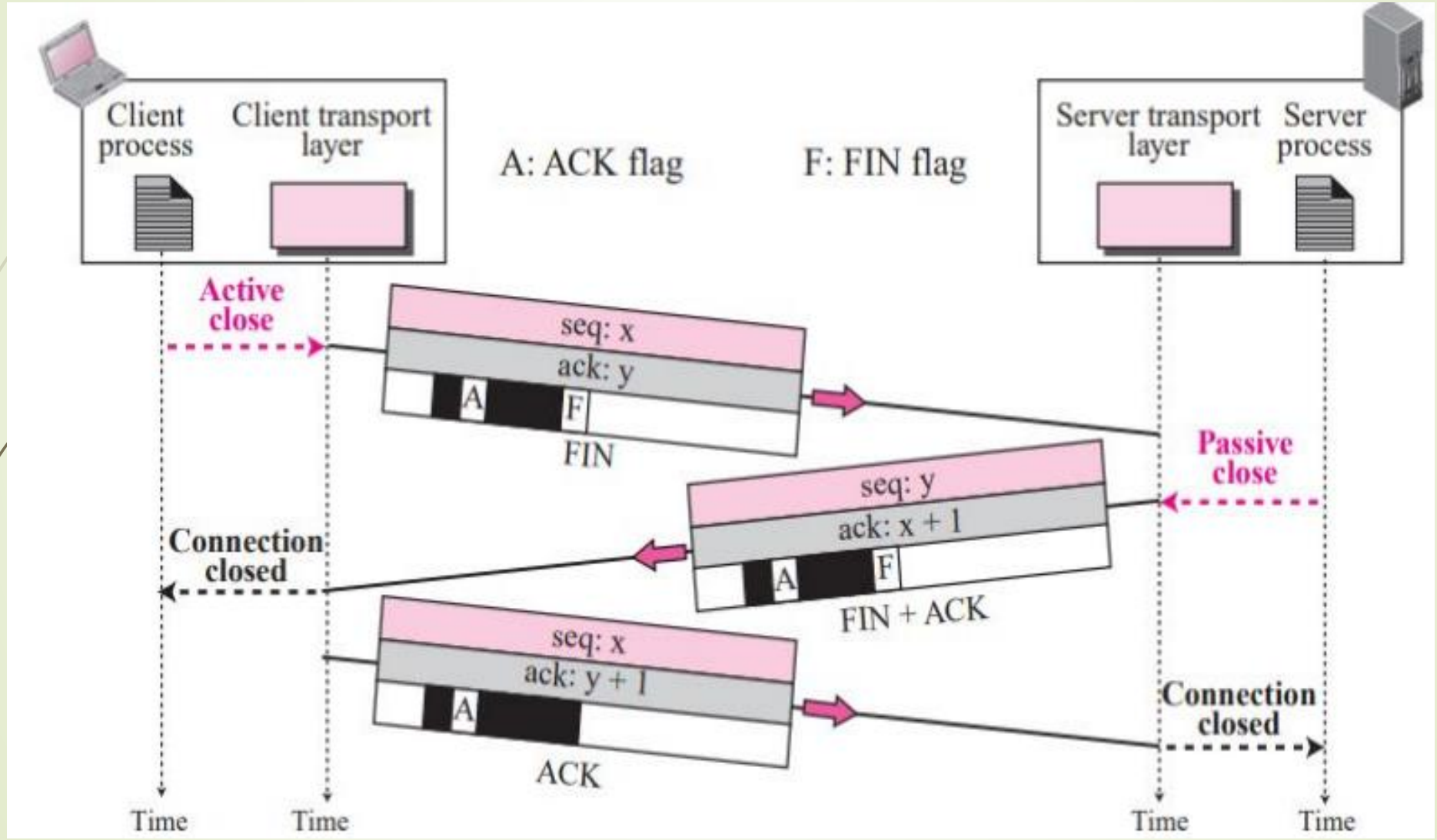
3. An ACK segment, if carrying no data, consumes no sequence number.

# Data transfer


Client process
Client transport layer
P: PSH flag
A: ACK flag
Server transport layer
Server process
Connection Establishment
Send request
seq: 8001
ack: 15001
A P
Data
bytes: 8001-9000
Receive
Send request
seq: 9001
ack: 15001
A P
Data
bytes: 9001-10000
Receive
seq: 15001
ack: 10001
Send request
A
Data
bytes: 15001-17000
seq: 10000
ack: 17001
A
rwnd:10000
Connection Termination
Time
Time
Time
Time

- After connection is established, bidirectional data transfer can take place. The client and server can send data and acknowledgments in both directions.
- Data traveling in the same direction as an acknowledgment are carried on the same segment. The acknowledgment is piggybacked with the data.

# Connection Termination



- Most implementations today allow three-way handshaking for connection termination:
- 1. In a common situation, 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. Note that a FIN segment can include the last chunk of data sent by the client or it can be just a control segment as shown in the figure. **If it is only a control segment, it consumes only one sequence number**

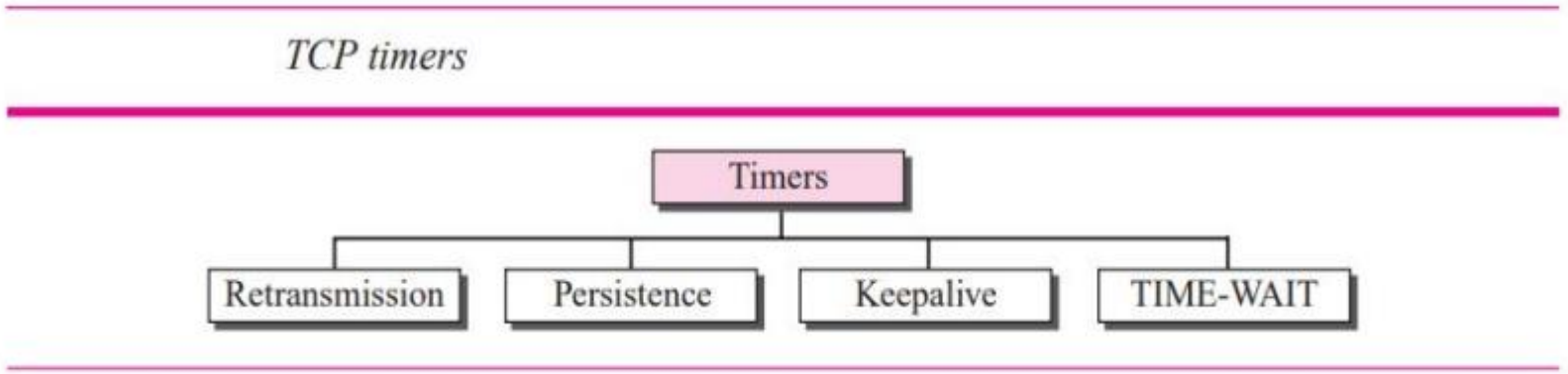
- 2. The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, a FIN+ACK segment, to confirm the receipt of the FIN segment from the client and at the same time to announce the closing of the connection in the other direction. This segment can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

- The client TCP sends the last segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server. This segment contains the acknowledgment number, which is one plus the sequence number received in the FIN segment from the server. **This segment cannot carry data and consumes no sequence numbers.**

# TCP Timer Management

- TCP uses multiple timers to do its work.
- **Timers can reduce the delay during communications.**

To perform its operation smoothly, most TCP implementations use at least four timers



# Retransmission timer

- When a segment is sent, a retransmission timer is started.
- If the segment is acknowledged before the timer expires, the timer is stopped.
- If, on the other hand, the timer goes off before the acknowledgement comes in, the segment is retransmitted and the timer started again.

# Persistent Timer

- To deal with a zero-window-size deadlock situation, TCP uses a persistence timer. When the sending TCP receives an acknowledgment with a window size of zero, it starts a persistence timer.
- When the persistence timer goes off, the sending TCP sends a special segment called a probe. This segment contains only 1 byte of new data. It has a sequence number, but its sequence number is never acknowledged;
- it is even ignored in calculating the sequence number for the rest of the data. The probe causes the receiving TCP to resend the acknowledgment which was lost.

# Keep Alive Timer –

- A keepalive timer is used to prevent a long idle connection between two TCPs. If a client opens a TCP connection to a server transfers some data and becomes silent (the client may crashed). In this case, the connection remains open forever. So a keepalive timer is used. Each time the server hears from a client, it resets this timer. The time-out is usually 2 hours. If the server does not hear from the client after 2 hours, it sends a probe segment.

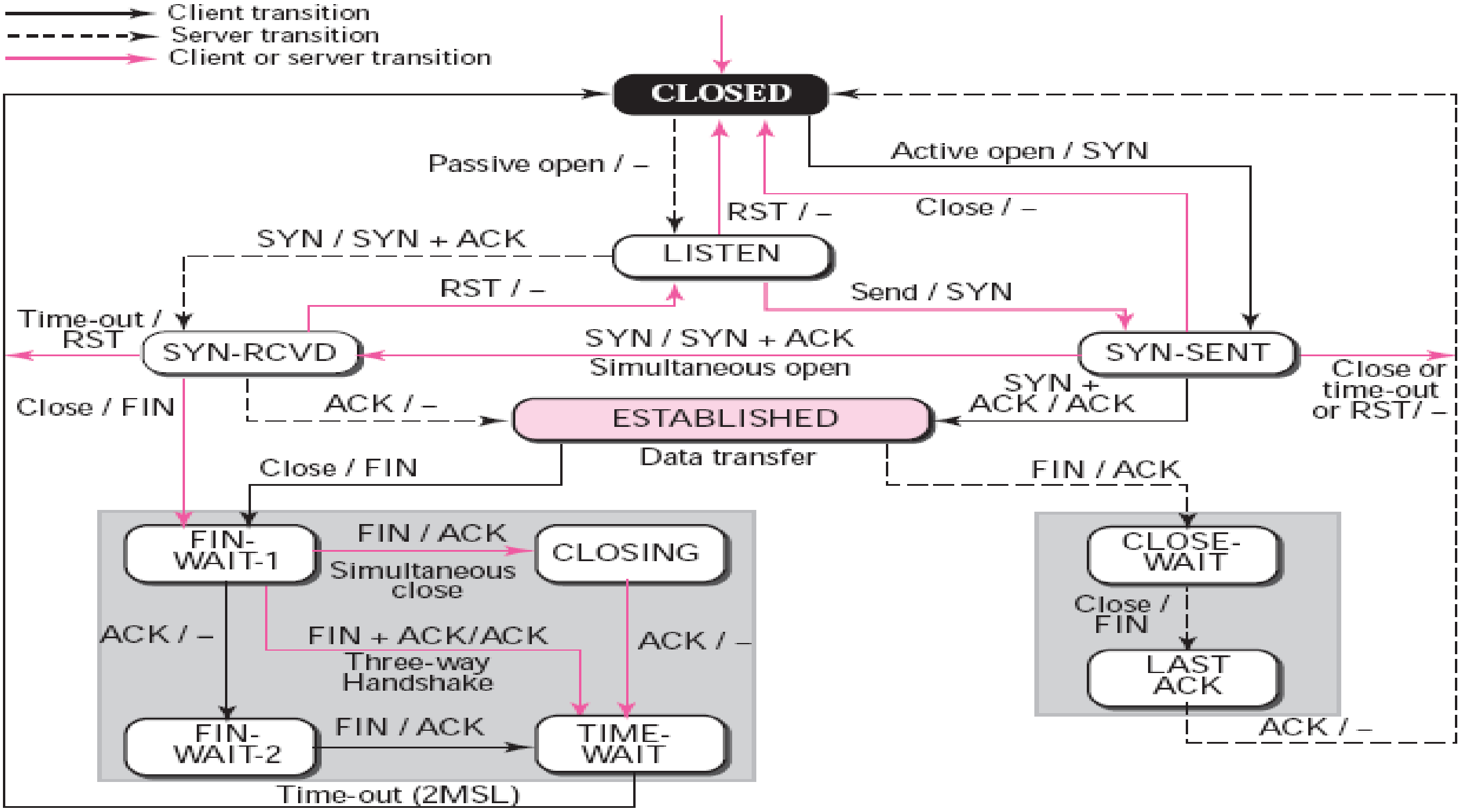
# Time wait Timer

- Timer used during TCP connection termination.
- Timer originates from last ACK and for two FIN and close the connection
- It runs to make sure that when a connection is closed, all packets created by it have died off.

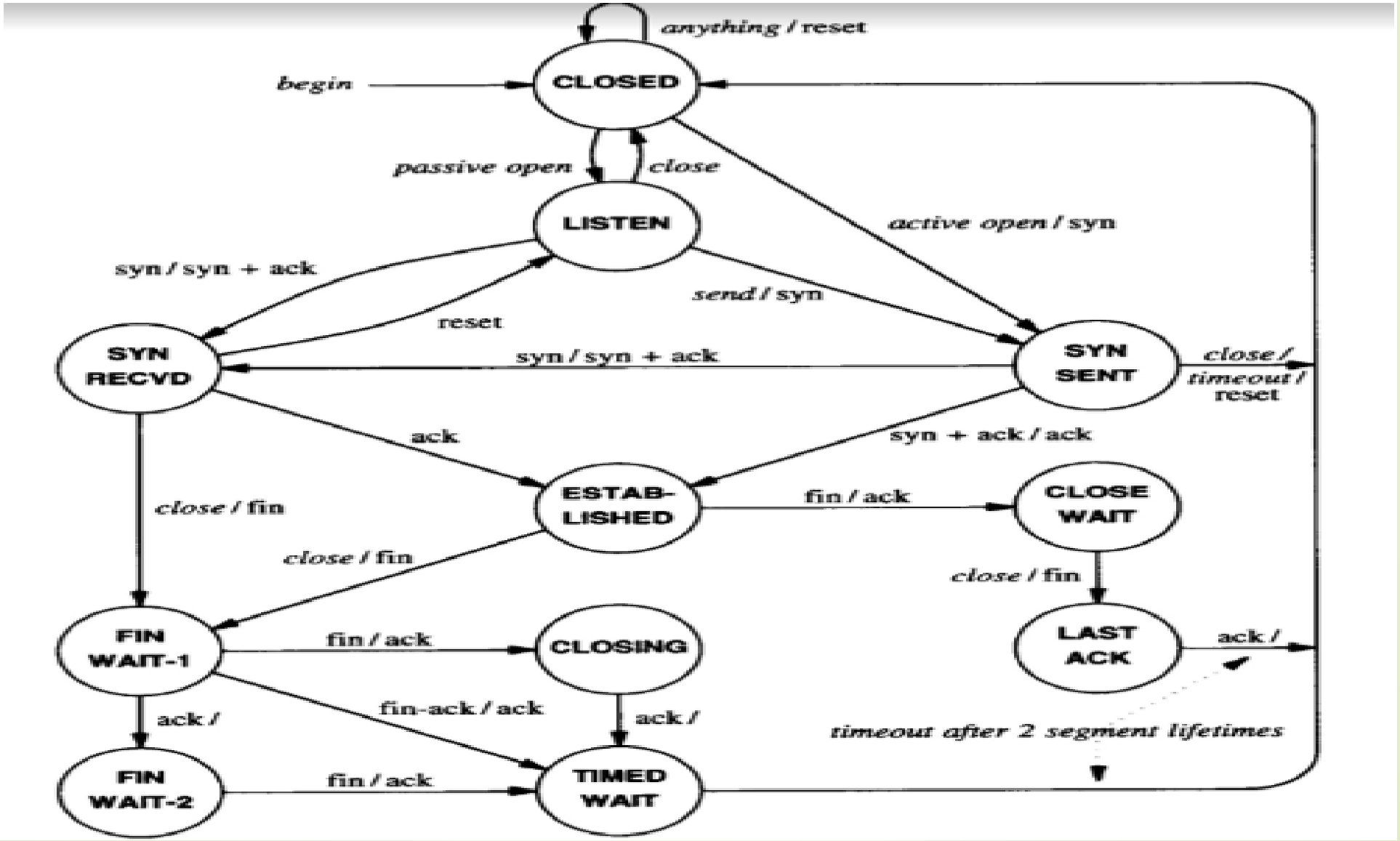
# State Transition Diagram

The steps required to establish and release connections can be represented in a finite state machine with the 11 states

| *State* | *Description* |
|---|---|
| CLOSED | No connection exists |
| LISTEN | Passive open received; waiting for SYN |
| SYN-SENT | SYN sent; waiting for ACK |
| SYN-RCVD | SYN+ACK sent; waiting for ACK |
| ESTABLISHED | Connection established; data transfer in progress |
| FIN-WAIT-1 | First FIN sent; waiting for ACK |
| FIN-WAIT-2 | ACK to first FIN received; waiting for second FIN |
| CLOSE-WAIT | First FIN received, ACK sent; waiting for application to close |
| TIME-WAIT | Second FIN received, ACK sent; waiting for 2MSL time-out |
| LAST-ACK | Second FIN sent; waiting for ACK |
| CLOSING | Both sides decided to close simultaneously |


Client transition
Server transition
Client or server transition
CLOSED
Passive open / –
Active open / SYN
RST / –
Close / –
SYN / SYN + ACK
LISTEN
RST / –
Send / SYN
Time-out / RST
SYN-RCVD
SYN / SYN + ACK
Simultaneous open
SYN-SENT
Close or time-out or RST / –
Close / FIN
ACK / –
ESTABLISHED
SYN + ACK / ACK
Data transfer
Close / FIN
FIN / ACK
FIN-WAIT-1
FIN / ACK
CLOSING
Simultaneous close
ACK / –
FIN + ACK/ACK
Three-way Handshake
ACK / –
FIN-WAIT-2
FIN / ACK
TIME-WAIT
Time-out (2MSL)
CLOSE-WAIT
Close / FIN
LAST ACK
ACK / –

# Finite state machine



- TCP software at each endpoint begins in the CLOSED state.
- Application programs must issue either a passive open command (to wait for a connection from another machine), or an active open command (to initiate a connection).
- An active open command forces a transition from the CLOSED state to the SYN SENT state.
- When TCP follows the transition, it emits a SYN segment. When the other end returns a segment that contains a SYN plus ACK, TCP moves to the ESTABLISHED state and begins data transfer.
- The TIMED WAIT state reveals how TCP handles some of the problems incurred with unreliable delivery. TCP keeps a notion of maximum segment lifetime (MSL), the maximum time an old segment can remain alive in an internet.

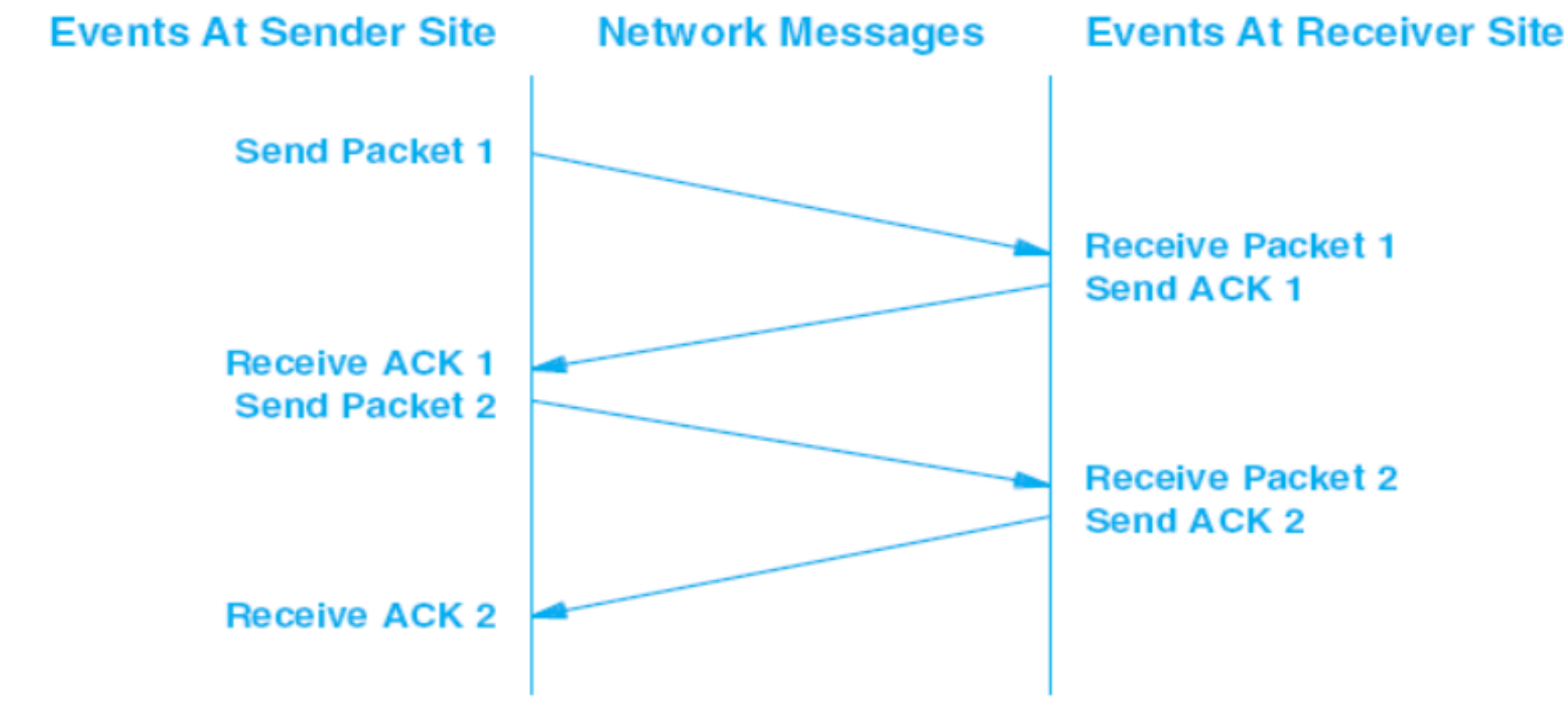
- To avoid having segments from a previous connection interfere with a current one, TCP moves to the TIMED WAIT state after closing a connection. It remains in that state for twice the maximum segment lifetime before deleting its record of the connection.
- If any duplicate segments happen to arrive for the connection during the timeout interval, TCP will reject them.
- However, to handle cases where the last acknowledgement was lost, TCP acknowledges valid segments and restarts the timer. Because the timer allows TCP to distinguish old connections from new ones, it prevents TCP from responding with a RST (reset) if the other end retransmits a FIN request.

# TCP Sliding Window

# Introduction

- TCP guarantees **reliable stream delivery service**.
- ie: **guarantees delivery** of a stream of data sent from one computer to another **without duplication or data loss.**
- When it sends a packet, the **sending software starts a timer.**
- If an acknowledgement arrives before the timer expires, the sender cancels the timer and prepares to send more data.
- If the timer expires before an acknowledgement arrives, the sender retransmits the packet.
- **The simplest possible retransmission scheme waits for a given packet to be acknowledged before it sends the next packet.**
- Known as **send-and-wait,** the approach can only send one packet at a time.

# The packet exchange for a basic send-and-wait protocol




- The left side of the figure lists events at the sending host, and the right side of the figure lists events at the receiving host.
- Each diagonal line crossing the middle shows the transfer of one packet or one ACK.

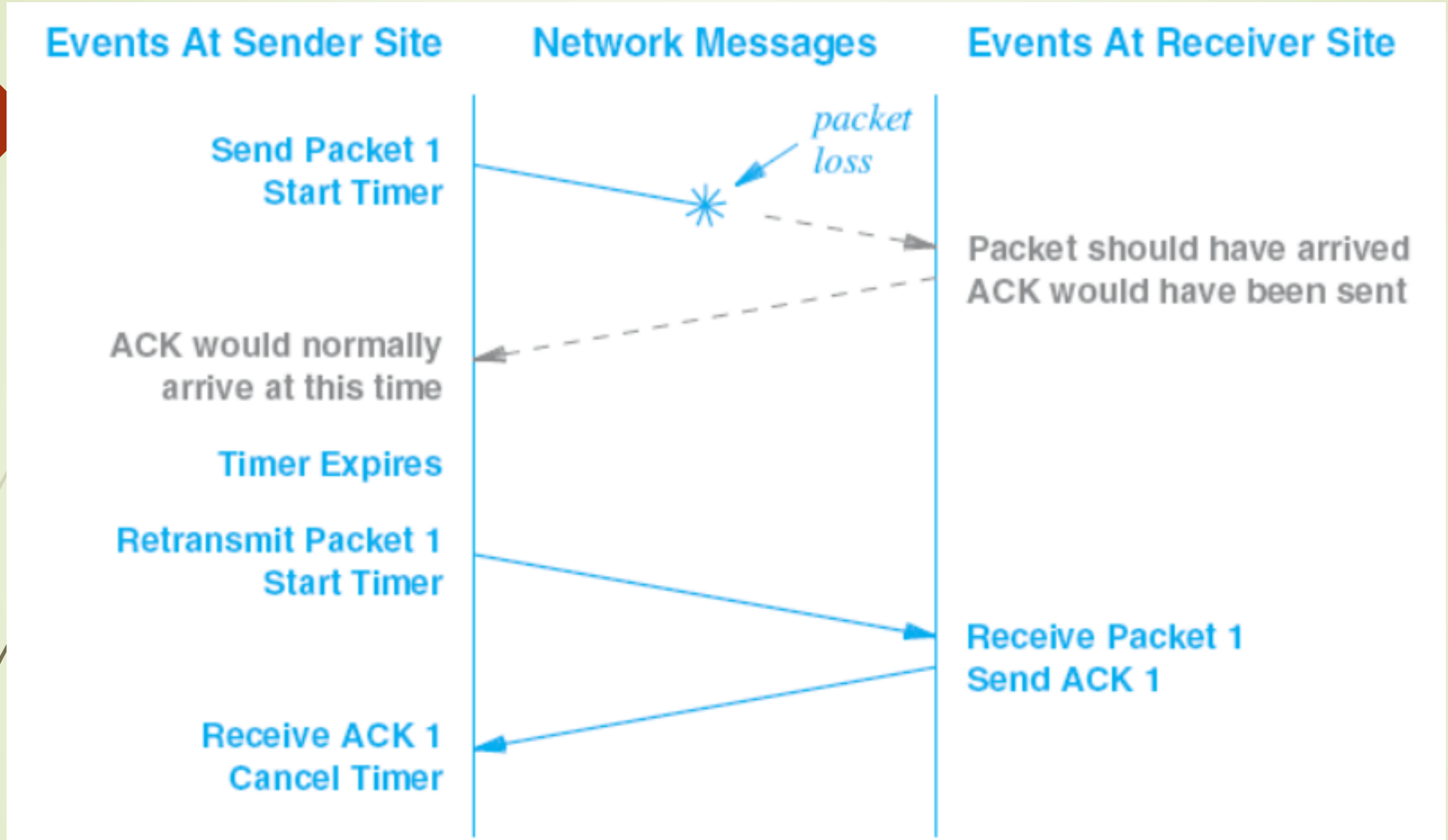
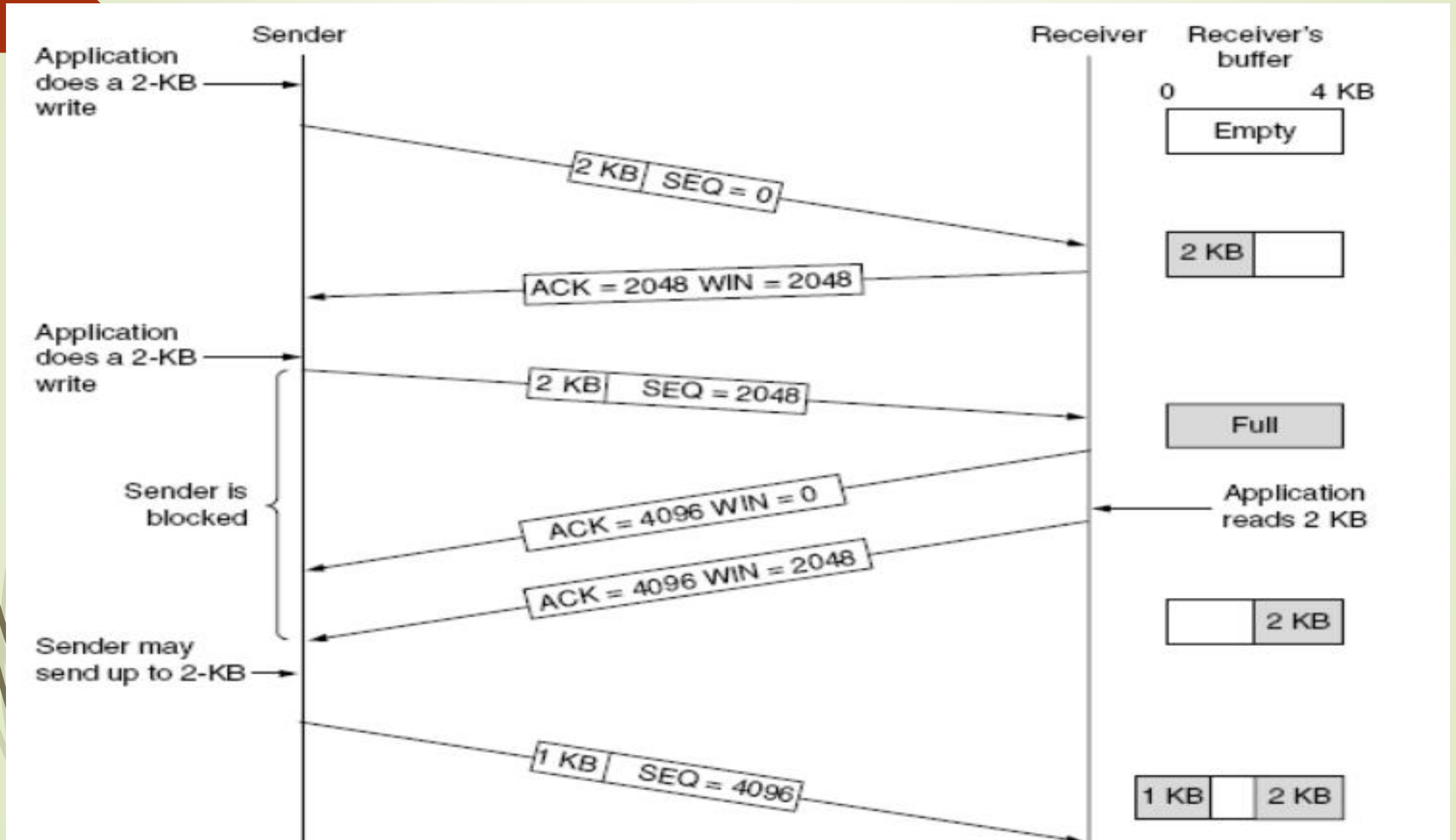



**Figure 11.2** Illustration of timeout and retransmission when a packet is lost.

# EXAMPLE - Window management in TCP



# Sliding Window

- **Additional mec**hanism that underlies reliable transmission.
- The mechanism **improves overall throughput**.
- To achieve reliability, the sender transmits a packet and then waits for an acknowledgement before transmitting another.
- As the previous figure shows, data flows between the machines **one packet at a time.**
- The **network will remain completely idle** until the acknowledgement returns.
- The sliding window technique uses a more complex form of acknowledgement and retransmission.
- The key idea is that a **sliding window allows a sender to transmit multiple packets before waiting for an acknowledgement**.
- The easiest way to envision a sliding window in action is to think of a sequence of packets to be transmitted.
- The protocol places a small, fixed-size ***window*** on the sequence and transmits all packets that lie inside the window.

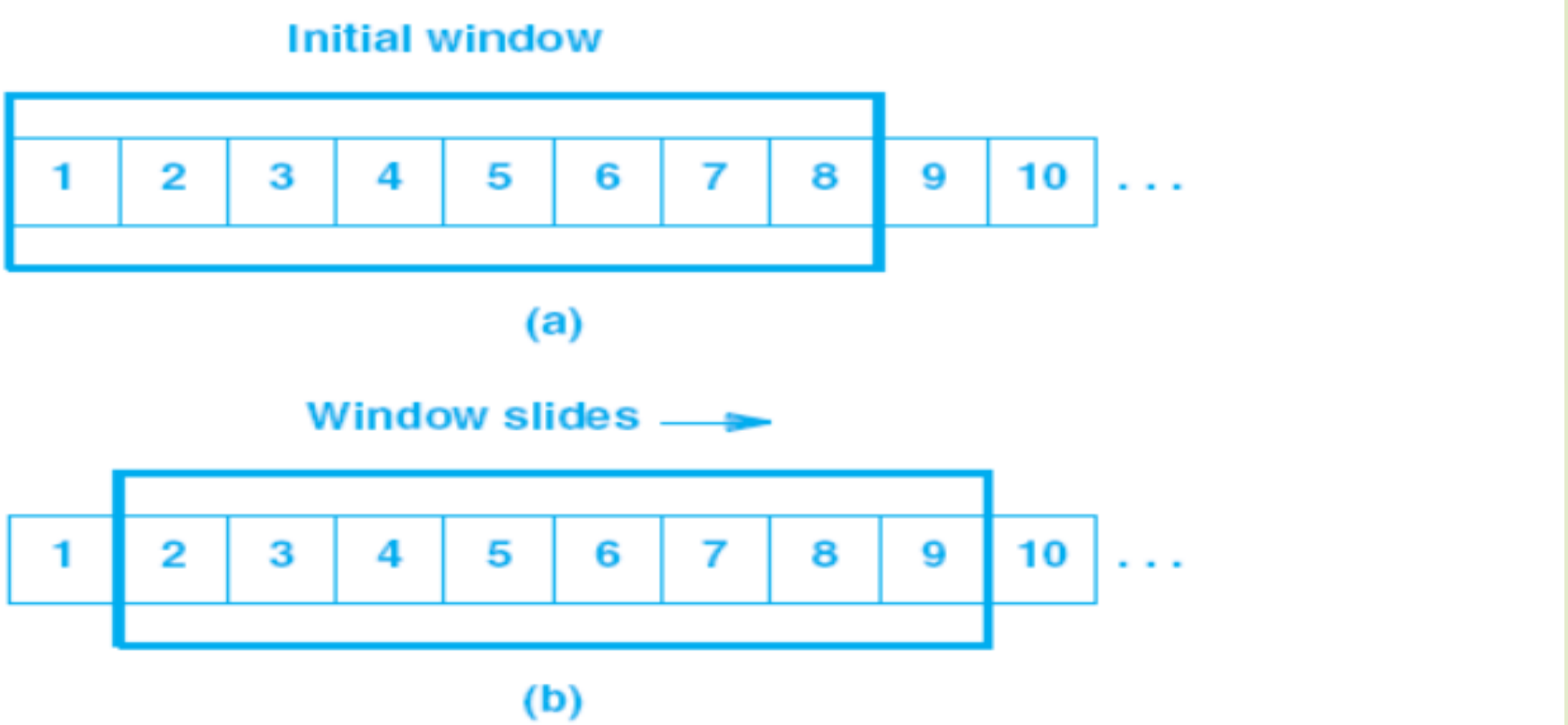
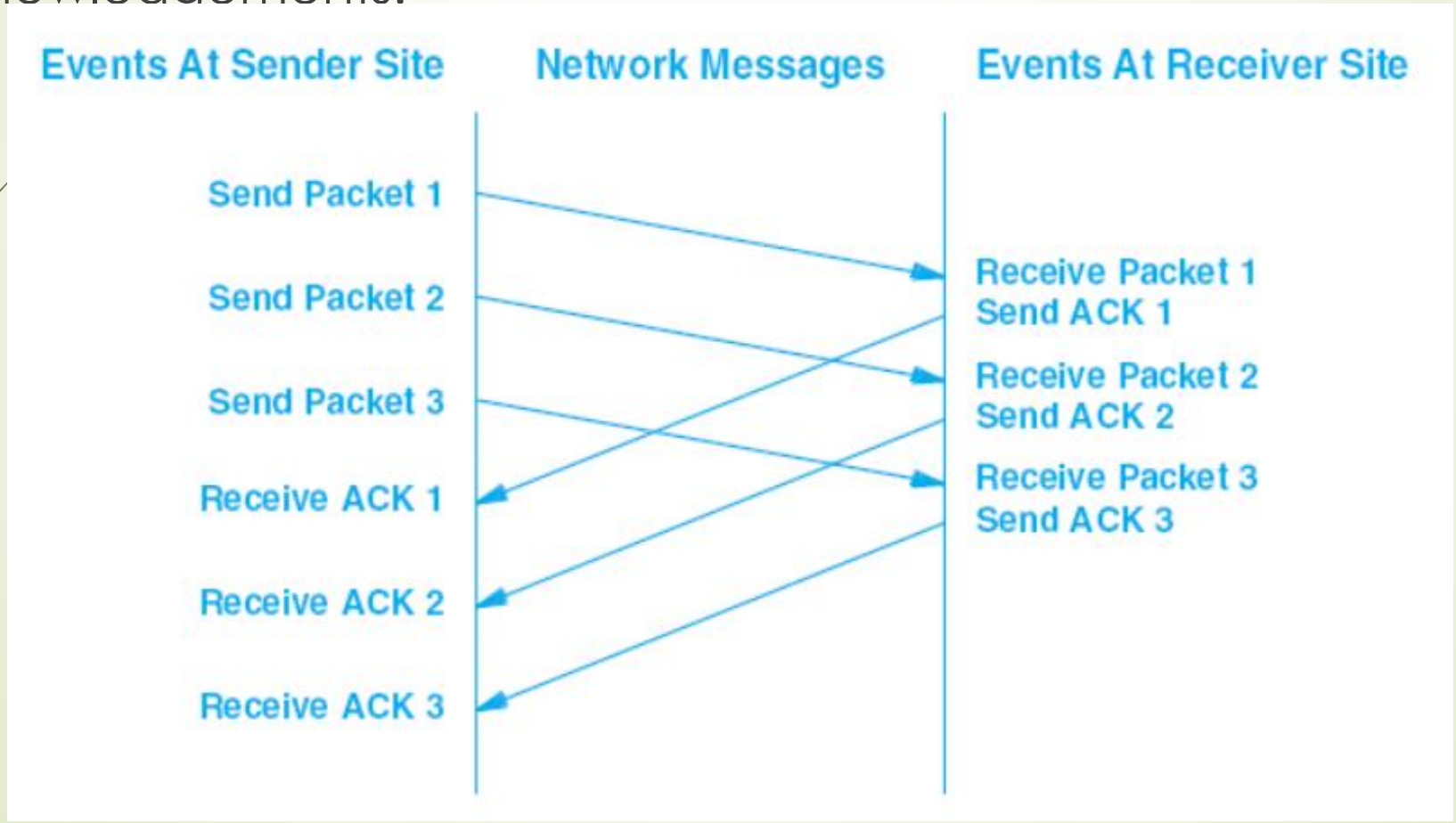



**Figure 11.3** (a) A sliding window with eight packets in the window, and (b) the window sliding so that packet *9* can be sent because an acknowledgement has been received for packet *1*.

- For example, in a sliding window protocol with window size 8, the sender is permitted to transmit 8 packets before it receives an acknowledgement.
- once the sender receives an acknowledgement for the first packet inside the window, it "slides" the window along and sends the next packet.
- **The window slides forward each time an acknowledgement arrives**.

- The performance of sliding window protocols **depends on the window size and the speed at which the network accepts packets.**
- Figure below shows an example of the operation of a sliding window protocol for a **window size of three packets.**
- Note that the sender transmits all three packets before receiving any acknowledgements.



- Conceptually, a sliding window protocol always remembers which packets have been acknowledged and keeps a separate timer for each unacknowledged packet.
- If a packet is lost, the timer expires and the sender retransmits that packet.
- When the sender slides its window, it moves past all acknowledged packets.
- At the receiving end, the protocol software keeps an analogous window, accepting and acknowledging packets as they arrive.
- Thus, the **window partitions the sequence of packets into three sets**:
  - Those packets to the left of the window have been successfully transmitted, received, and acknowledged;
  - those packets to the right have not yet been transmitted; and
  - those packets that lie in the window are being transmitted.

# TCP CONGESTION CONTROL

# TCP CONGESTION CONTROL - introduction

- **Congestion** in a network may occur if the load on the network – the number of packets sent to the network – is greater than the capacity of the network – the number of packets a network can handle.
- If every pair of communicating end systems continues to pump packets into the network as fast as they can, gridlock sets in and few packets are delivered to their destinations.
- When a router becomes congested, its buffers can overflow and packet loss can occur.
- The Internet avoids this problem by forcing end systems to diminish the rate at which they send packets into the network during periods of congestion.
- End systems are alerted to the existence of severe congestion when they stop receiving acknowledgments for the packets they have sent.

# TCP CONGESTION CONTROL

- TCP includes **congestion control mechanism**, a service for the general welfare of the Internet rather than for the direct benefit of the communicating processes.
- **Congestion Control** refers to the mechanisms and techniques to **control the congestion and keep the load below the capacity.**
  - Either prevent congestion before it happens, or remove congestion, after it has happened.
- The TCP congestion control mechanism throttles a process (client or server) when the network is congested.
- This mechanism forces each new TCP connection to initially transmit data at a relatively slow rate, but then allows each connection to ramp up to a relatively high rate when the network is uncongested.

# TCP CONGESTION CONTROL : self-clocking

The idea of TCP congestion control is for each source to determine how much capacity is available in the network, so that it knows how many packets it can safely have in transit.

- Once a given source has this many packets in transit, it uses the arrival of an ACK as a signal that one of its packets has left the network, and that it is therefore safe to insert a new packet into the network without adding to the level of congestion.
- By using ACKs to pace the transmission of packets, TCP is said to be *self-clocking*.

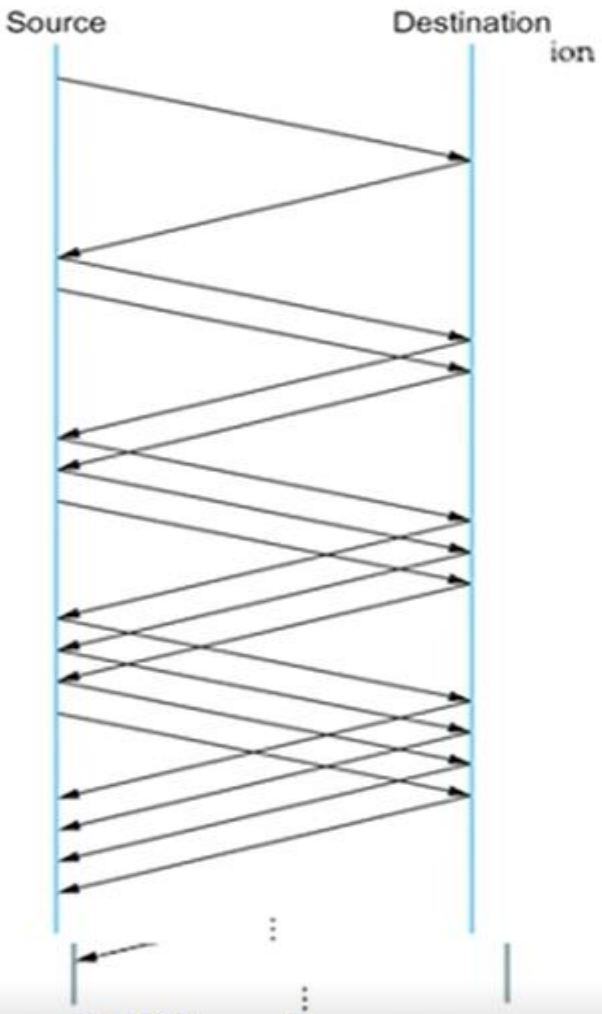
# TCP CONGESTION CONTROL : AIMD

- Additive Increase Multiplicative Decrease
  - TCP maintains a new state variable for each connection, called *CongestionWindow*, which is used by the source to limit how much data it is allowed to have in transit at a given time.
- The problem, of course, is how TCP comes to learn an appropriate value for CongestionWindow.
  - The answer is that the TCP source sets the CongestionWindow based on the level of congestion it perceives to exist in the network.
  - This involves decreasing the congestion window when the level of congestion goes up and increasing the congestion window when the level of congestion goes down. Taken together, the mechanism is commonly called *additive increase/multiplicative decrease (AIMD)*

# TCP CONGESTION CONTROL : AIMD

- The key question, then, is how does the source determine that the network is congested and that it should decrease the congestion window?
  - The answer is based on the observation that the main reason packets are not delivered, and a timeout results, is that a packet was dropped due to congestion. It is rare that a packet is dropped because of an error during transmission.
  - Therefore, TCP interprets timeouts as a sign of congestion and reduces the rate at which it is transmitting.
  - Specifically, each time a timeout occurs, the source sets CongestionWindow to half of its previous value. This halving of the CongestionWindow for each timeout corresponds to the "multiplicative decrease" part of AIMD.

# TCP CONGESTION CONTROL : AIMD




Packets in transit during additive increase, with one packet being added each RTT.

# TCP CONGESTION CONTROL : AIMD

- Although CongestionWindow is defined in terms of bytes, it is easiest to understand multiplicative decrease if we think in terms of whole packets.
  - For example, suppose the CongestionWindow is currently set to 16 packets. If a loss is detected, CongestionWindow is set to 8.
  - Additional losses cause CongestionWindow to be reduced to 4, then 2, and finally to 1 packet.
  - CongestionWindow is not allowed to fall below the size of a single packet, or in TCP terminology, the *maximum segment size (MSS).*

# TCP CONGESTION CONTROL : AIMD

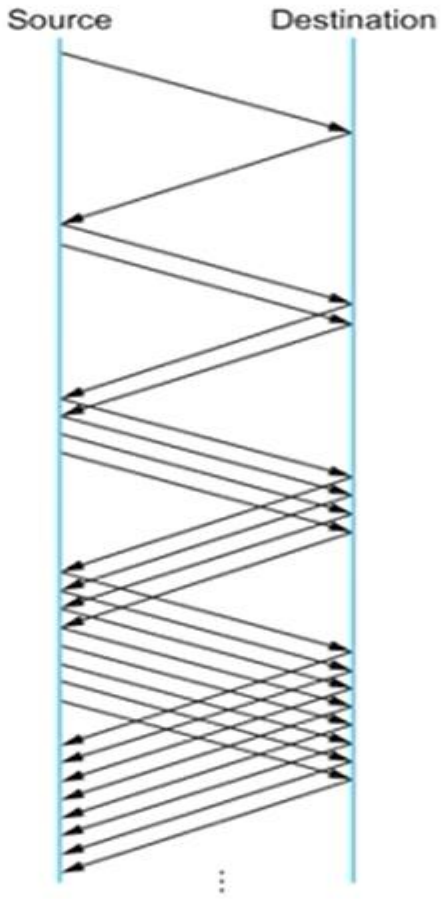
- A congestion-control strategy that only decreases the window size is obviously too conservative.
- We also need to be able to increase the congestion window to take advantage of newly available capacity in the network.
- This is the "additive increase" part of AIMD, and it works as follows.
  - Every time the source successfully sends a CongestionWindow's worth of packets—that is, each packet sent out during the last RTT has been ACKed—it adds the equivalent of 1 packet to CongestionWindow.

# TCP CONGESTION CONTROL : SLOW START

- Slow Start
  - The additive increase mechanism just described is the right approach to use when the source is operating close to the available capacity of the network, but it takes too long to ramp up a connection when it is starting from scratch.
  - TCP therefore provides a second mechanism, ironically called *slow start*, that is used to increase the congestion window rapidly from a cold start.
  - Slow start effectively increases the congestion windowexponentially, rather than linearly.

# TCP CONGESTION CONTROL : SLOW START

- Slow Start




Packets in transit during slow start.

# TCP CONGESTION CONTROL : SLOW START

- Specifically, the source starts out by setting CongestionWindow to one packet.
- When the ACK for this packet arrives, TCP adds 1 to CongestionWindow and then sends two packets.
- Upon receiving the corresponding two ACKs, TCP increments CongestionWindow by 2—one for each ACK—and next sends four packets.
- The end result is that TCP effectively doubles the number of packets it has in transit every RTT.

# TCP CONGESTION CONTROL

- There are actually two different situations in which slow start runs.
  - The first is at the very beginning of a connection, at which time the source has no idea how many packets it is going to be able to have in transit at a given time.
  - The second situation in which slow start is used is a bit more subtle; it occurs when the connection goes dead while waiting for a timeout to occur.

# Label switching, flows and MPLS

# Switching techniques

- **Circuit Switching**
  - A path is setup, and then data is sent down
- **Packet Switching**
  - Every packet is routed and forwarded hop-by-hop
- **Virtual Switching Technique**
  - Combine both circuit and packet switching; forwarding quickly
- **Label Switching**
  - Assigns and uses **labels** for switching instead IP address

# IP Routing Disadvantages

- Connectionless
  - Eg : No QoS
- Each router has to make independent forwarding decisions based on the IP address
- Larger IP headers
  - Atleast 20 bytes
- Routing in N/W layer slower than Switching
- Usually designed to obtain Shortest Path

# Label switching

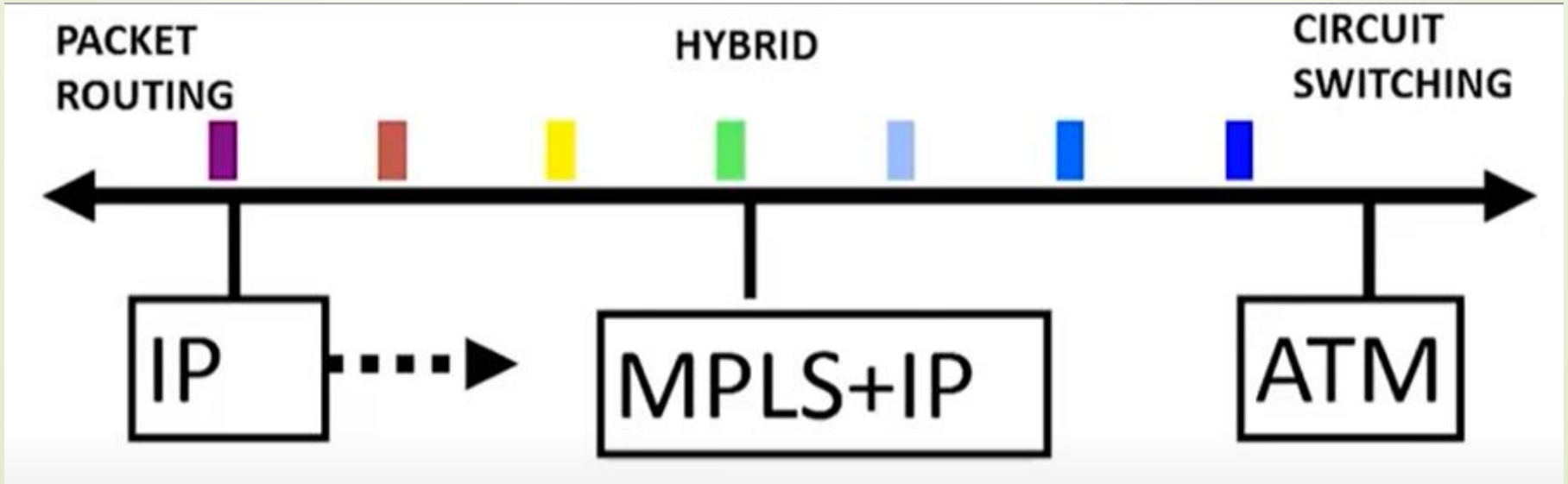
- Growth and evolution of the internet
- A virtual switching based technology, called ATM (Asynchronous Transfer Mode) had been proposed as an alternative to IP.
- Guaranteed superior performance.
- ATM switches were faster than routers as they used fixed length label lookup rather than the longest prefix match used by IP.
- Allow speed of L2 Switching at L3
- **'Label switching'** : an idea to evolve a mechanism to do L2-like or ATM-like switching at a router.

# LABEL SWITCHING

- Label switching is similar to virtual-circuit switching.
- The **path** for a flow of packets from a source to a destination is **identified before the packet transfer begins**.
- The **packets are given a fixed-size label.**
- It is this label that is used to switch the packets at each of the routers/switches.
- This gives a performance similar to L2 switching, while maintaining the flexibility of L3 routing.
- Also, it **provides a mechanism to support QoS** – as it provides a (virtual) connection-oriented framework.
- Label switching can work along with any other underlying protocol – IP or ATM, or any other protocol.

# MPLS

- **Multi Protocol Label Switching**
- **Protocol designed to work with label switching**
- This protocol tries to bring the **best of both worlds** (flexibility-oriented packet switching - IP and performance oriented circuit-switching) to work together.

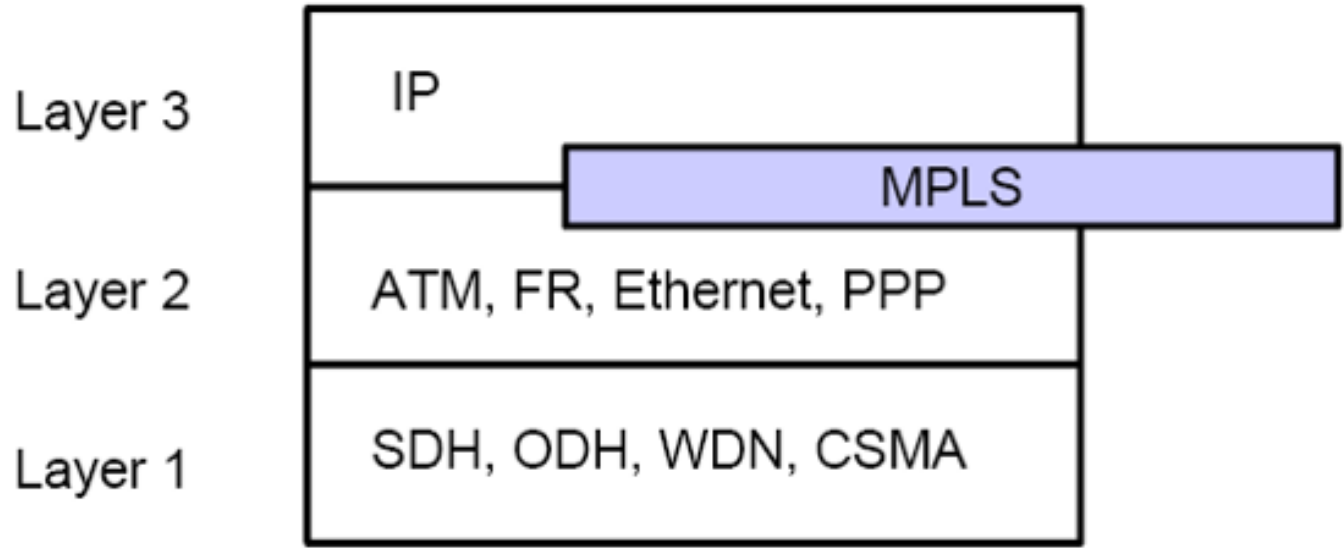


# MPLS

- MPLS **builds a connection-oriented service** on the IP network.
- Hence, MPLS comprises of a protocol to **establish an end-to-end path from source to the destination**, and a **hop-by-hop forwarding mechanism.**
- **Labels are used to set up the path.**
- A **protocol** is used to set up the labels along the path. (**LDP**)

# MPLS BAsics

- MPLS is arranged between Layer 2 and Layer 3 – in fact it is called a **Layer 2.5 protocol.**
- It is independent of Layer-2 an Layer-3 protocols.
- It **maps IP-addresses to fixed length labels**.
- It interfaces to existing routing protocols (RSVP, OSPF).
- Supports ATM, Frame Relay and Ethernet



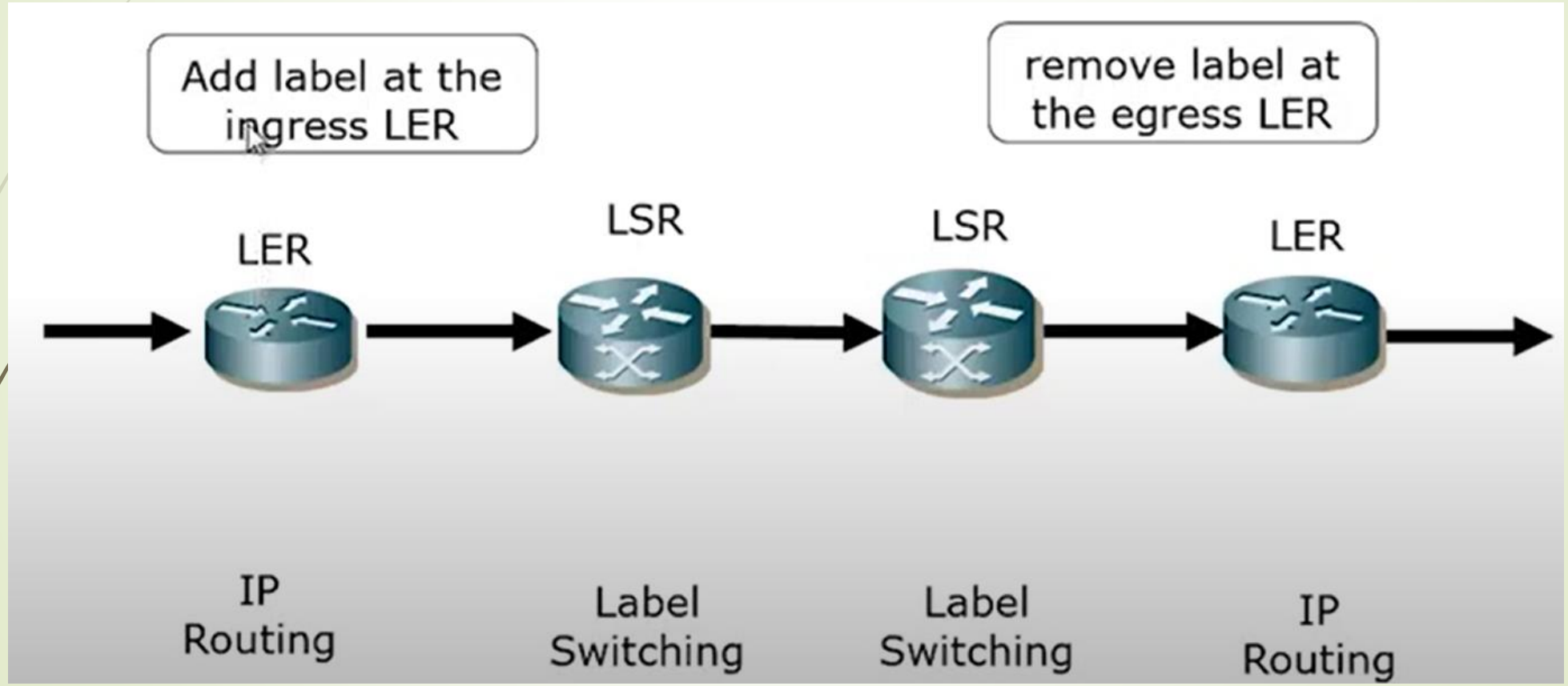
# MPLS Terminology

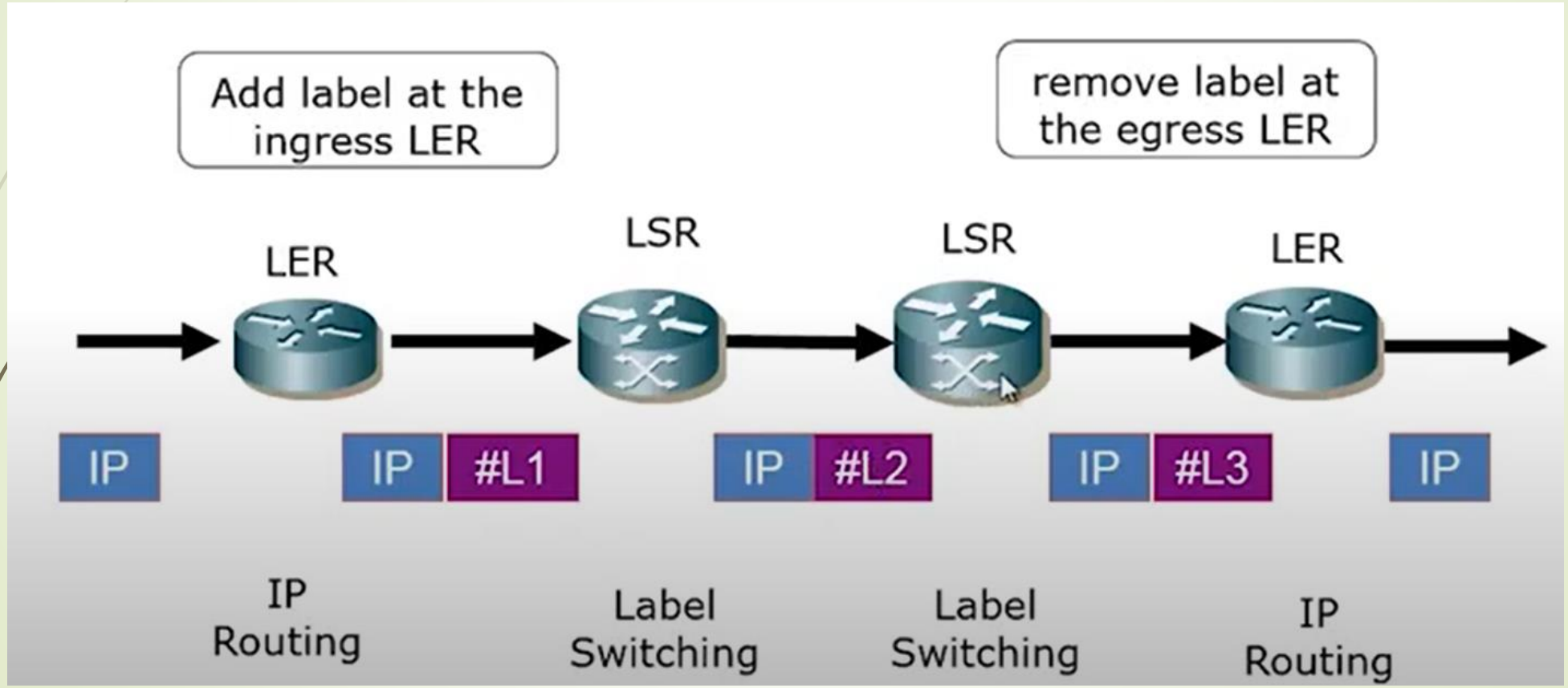
- **Label Switched Routers (LSR)**: These are routers that support MPLS.
- **Label Edge Router (LER)** – An LSR at the edge of the network is called Label Edge Router - also known as Edge LSR.
- **MPLS domain** : is the set of interconnected LSRs running the MPLS protocol.
- **Ingress LER**, at the incoming edge of an MPLS domain is responsible for adding labels to unlabelled IP packets.
- **Egress LER** at the outgoing edge, is responsible for removing the labels.
- **Label Switched Path (LSP)** – the path defined by the labels through LSRs between two LERs.

# MPLS Terminology

- **Label Forwarding Information Base (LFIB) –** a forwarding table (mapping) between labels and outgoing interfaces.
- **Forward Equivalent Class (FEC)** – All IP packets that follow the same path on the MPLS network and receive the same treatment at each node are said to belong to the same FEC.
- It can be equated to a **flow.**

# How does it work ?




Add label at the ingress LER
remove label at the egress LER
LER
LSR
LSR
LER
IP
IP
#L1
IP
#L2
IP
#L3
IP
IP Routing
Label Switching
Label Switching
IP Routing

# LABEL

- A label is an integer identifying an FEC (a flow).
- We cannot have globally or network- unique labels.
- Hence labels are switched at the routers.
- Therefore, labels are unique only between two nodes, and they change at each node as a packet traverses a path.
- Labels can be set manually, or dynamically using some label distribution mechanism.
- The label forwarding information base which provides a mapping between labels

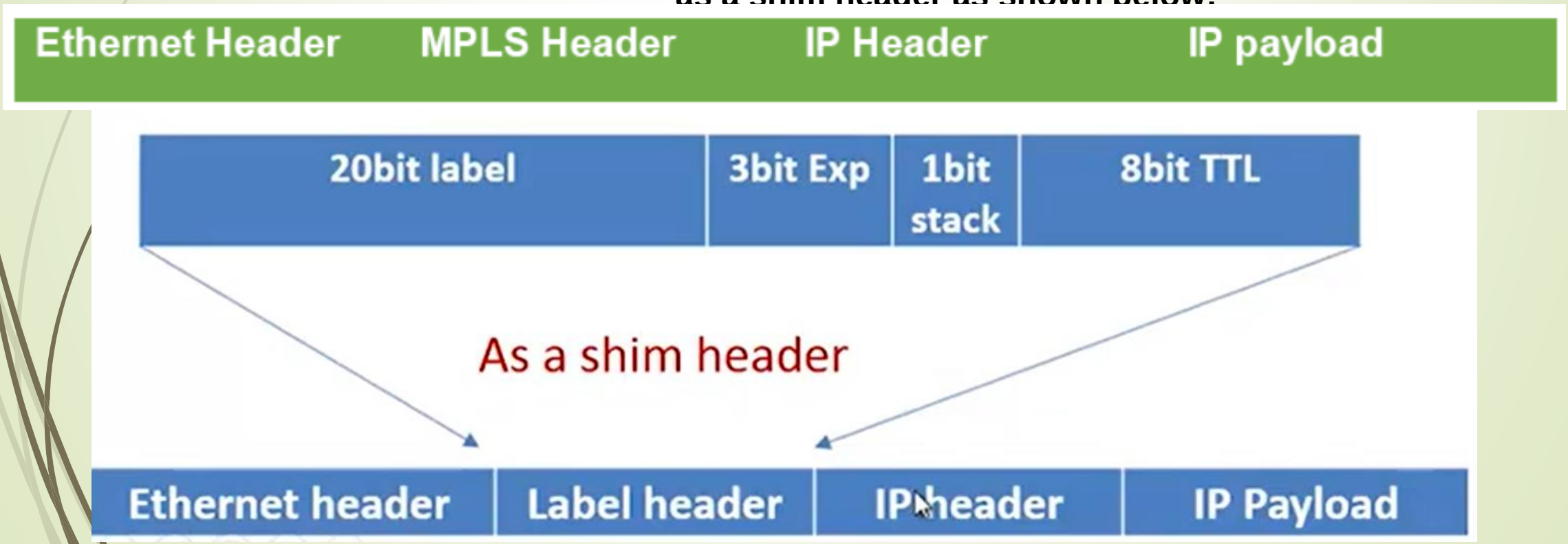
| Interface In | Label In | Destination | Interface Out | Label Out |
|---|---|---|---|---|
| 2 | 35 | 142.26 | 3 | 79 |

# Label format/mpls header

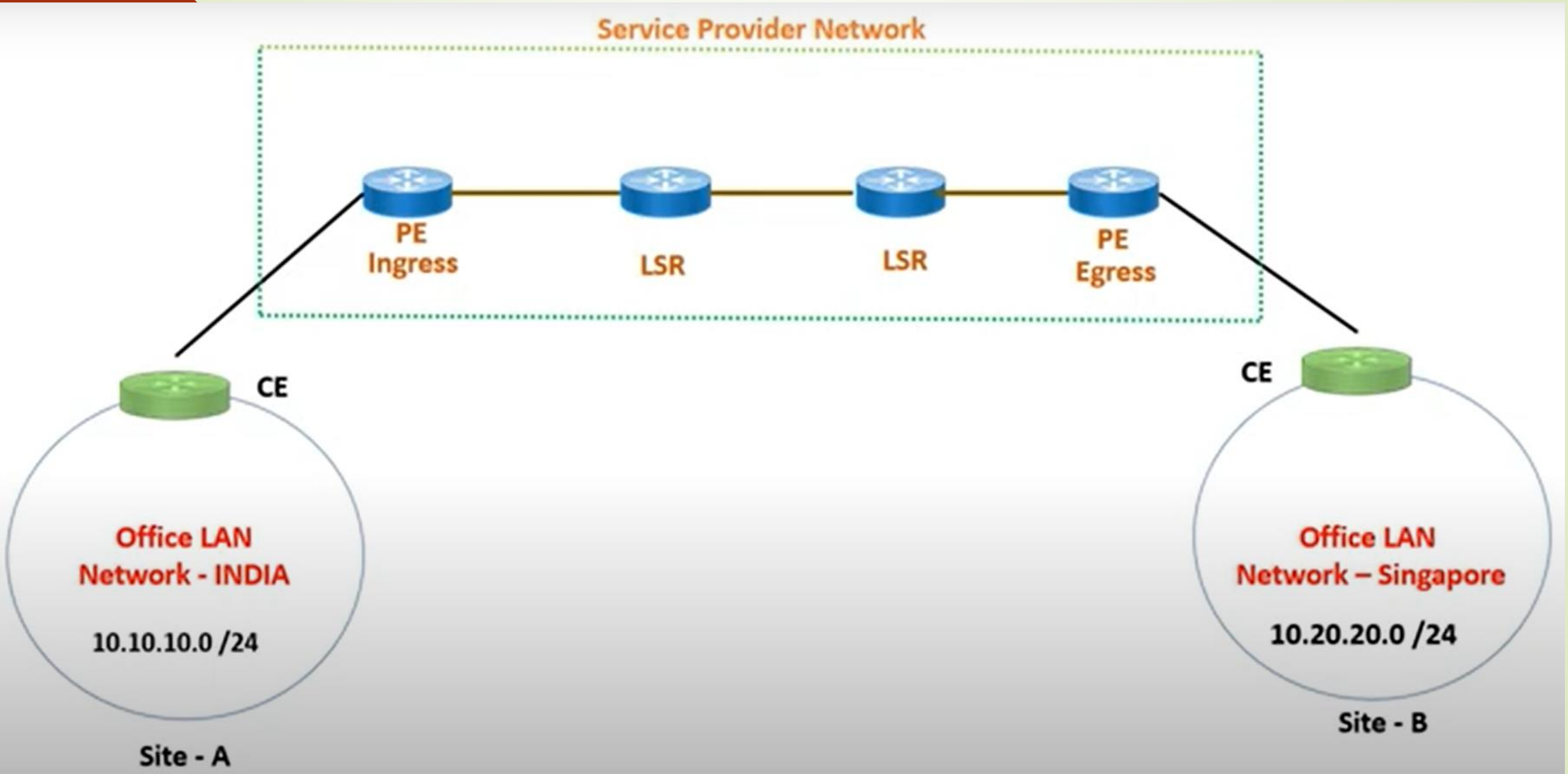
- Label: The actual label value used as the pointer for forwarding – represents the FEC.
- Exp: Experimental bits often used for QoS.
- S: Bottom of stack flag - for indicating whether the label is at the bottom of the label stack. 1 indicates that no label follows. This field is useful when there are multiple levels of MPLS labels.
- TTL: Time to live - This field has the same meaning as that for an IP packet.

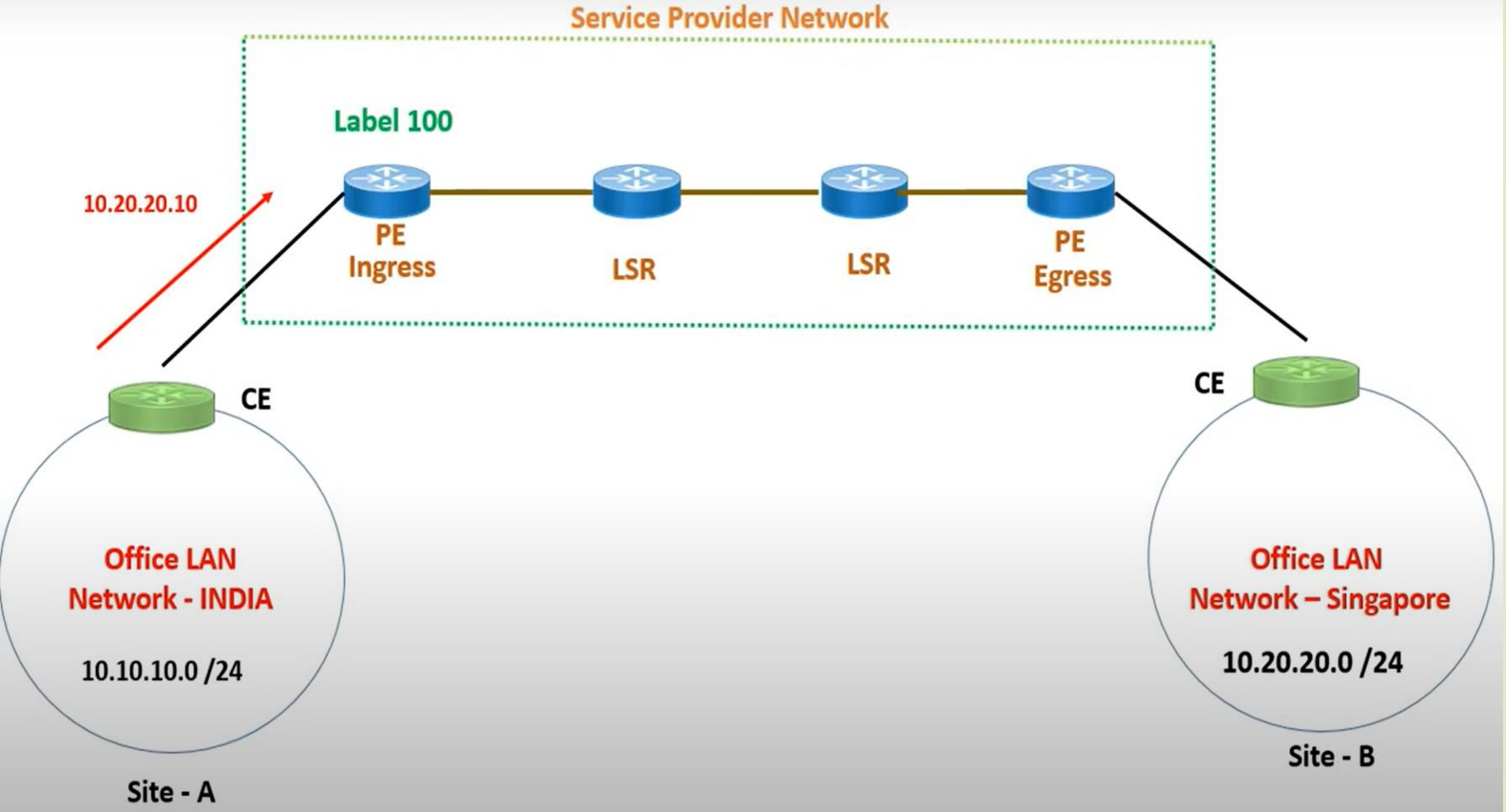
# Mpls encapsulation

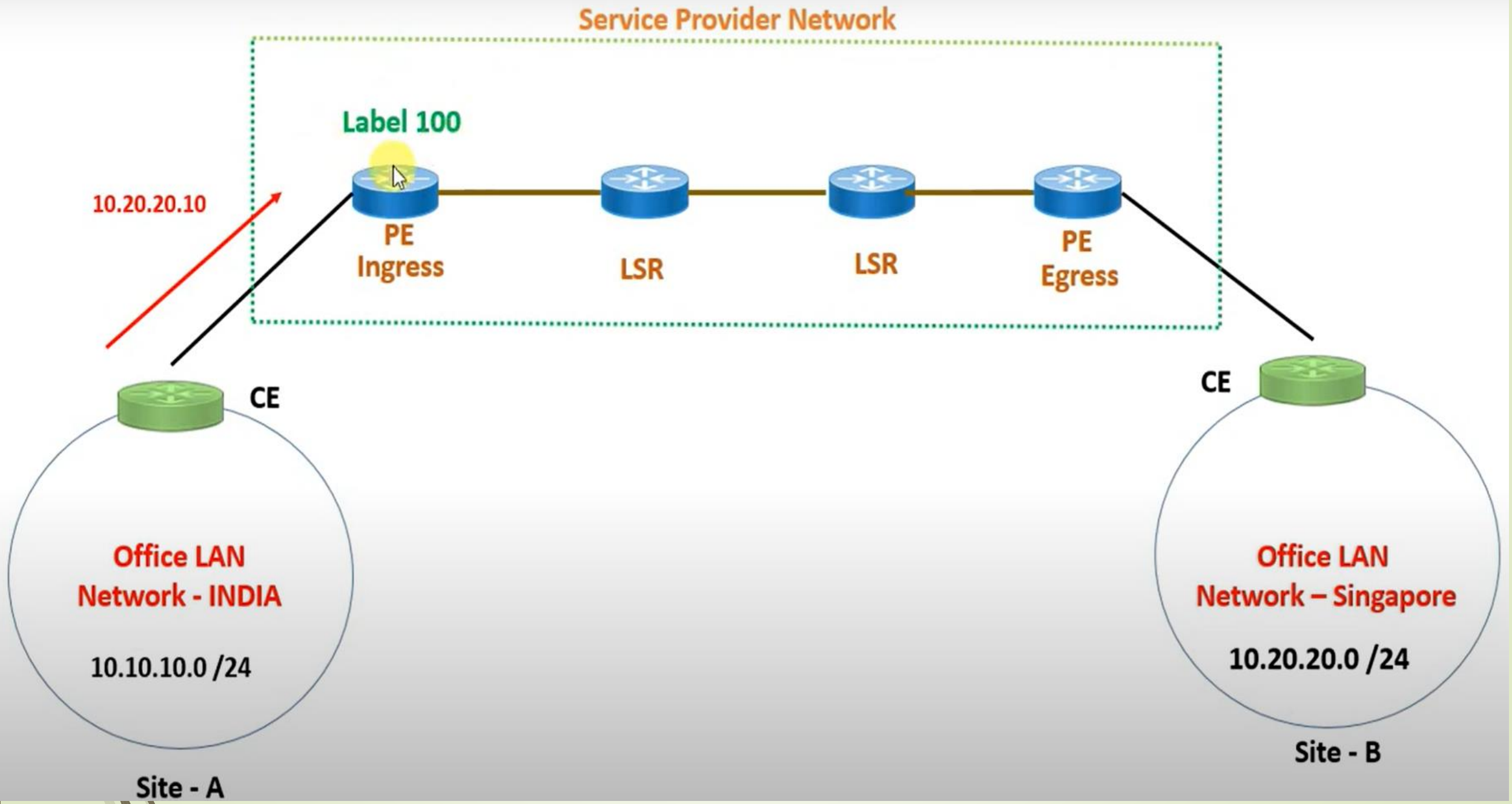
- MPLS is viewed as an L2.5 protocol.
- Accordingly, the MPLS header is also added between the L2 and L3 headers, as a shim header as shown below:

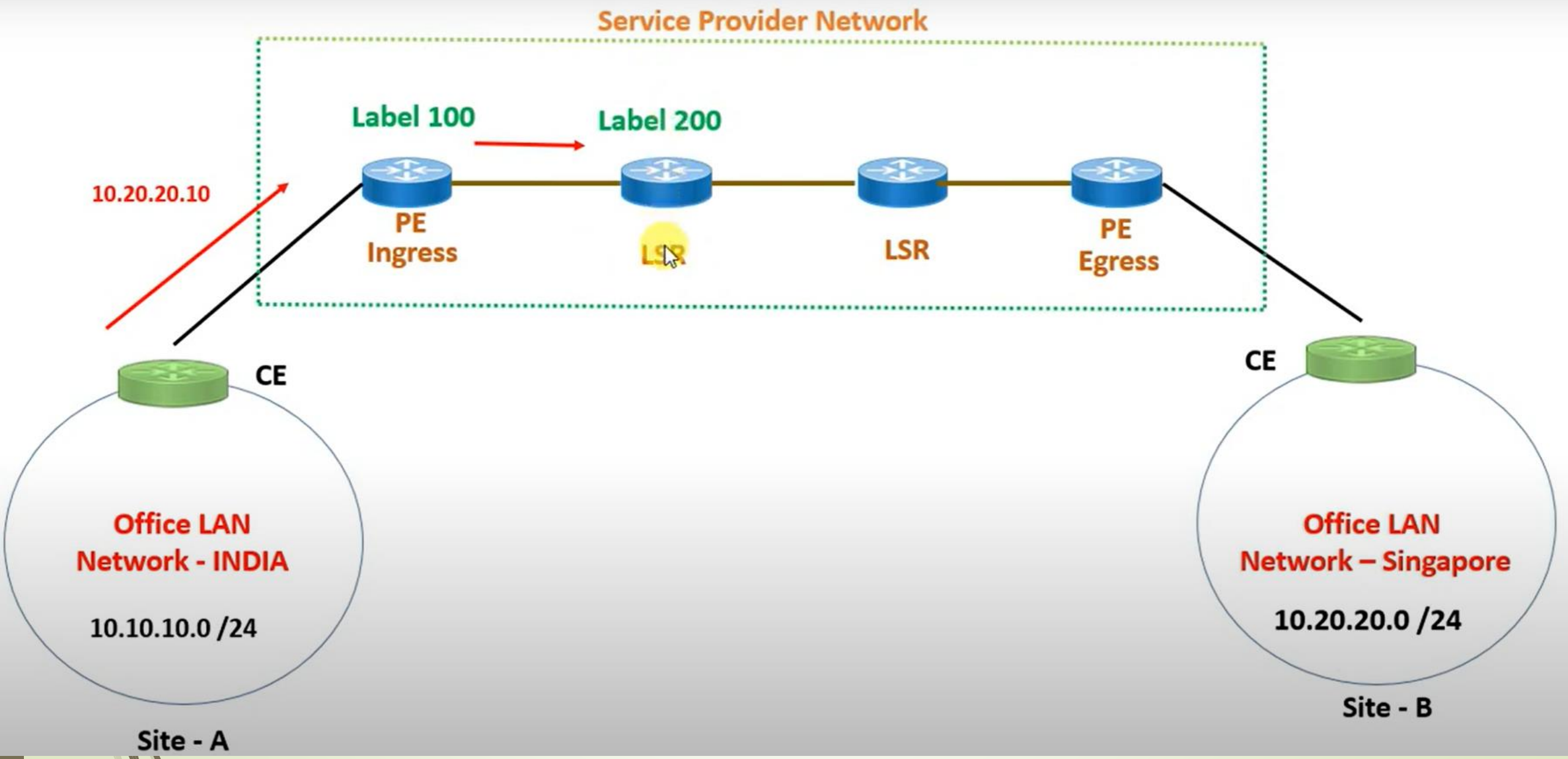


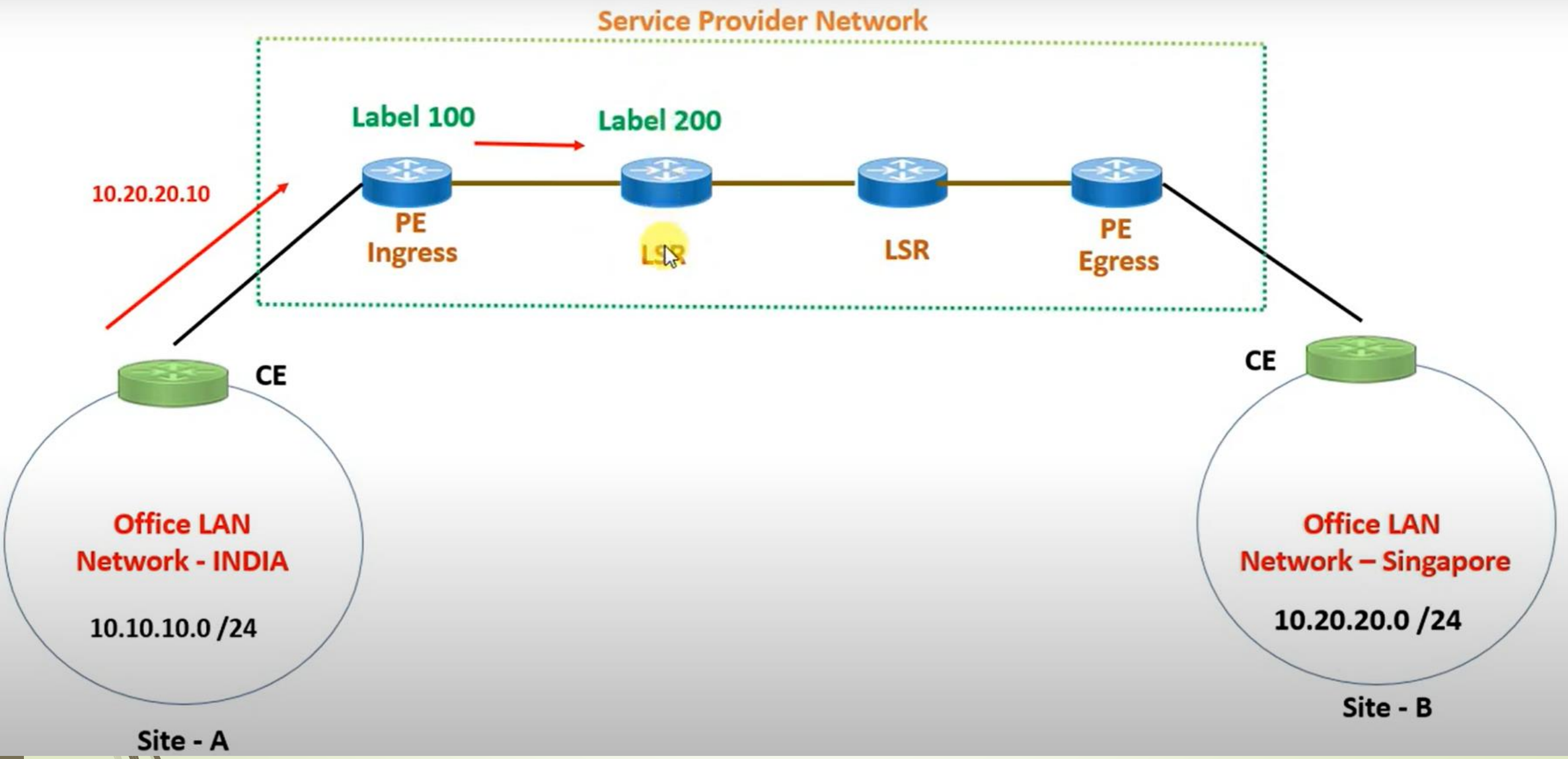
# MPLS WORKING – EXAMPLE




Service Provider Network
Label 100
10.20.20.10
PE
Ingress
LSR
LSR
PE
Egress
CE
CE
Office LAN
Network - INDIA
10.10.10.0 /24
Site - A
Office LAN
Network – Singapore
10.20.20.0 /24
Site - B


Service Provider Network
Label 100
10.20.20.10
PE
Ingress
LSR
LSR
PE
Egress
CE
CE
Office LAN
Network - INDIA
10.10.10.0 /24
Site - A
Office LAN
Network – Singapore
10.20.20.0 /24
Site - B


Service Provider Network
Label 100
Label 200
10.20.20.10
PE
Ingress
LSR
LSR
PE
Egress
CE
CE
Office LAN
Network - INDIA
10.10.10.0 /24
Site - A
Office LAN
Network – Singapore
10.20.20.0 /24
Site - B


Service Provider Network
Label 100
Label 200
10.20.20.10
PE
Ingress
LSR
LSR
PE
Egress
CE
CE
Office LAN
Network - INDIA
10.10.10.0 /24
Site - A
Office LAN
Network – Singapore
10.20.20.0 /24
Site - B

3 Major Operations of MPLS:

**PUSH** → Add Label to the packet, performed at the Ingress Router

**SWAP** → Replace the Label of the packet, performed by the LSR in between Ingress and Egress Router.

**POP** → Removing the label from the IP Packet, done at the Egress Router

# Network Address Translation (NAT)

# Special Address

- **Private Address**
  - A number of address are assigned for private use.
  - They are not recognized globally.

**Table 5.2** *Addresses for private networks*

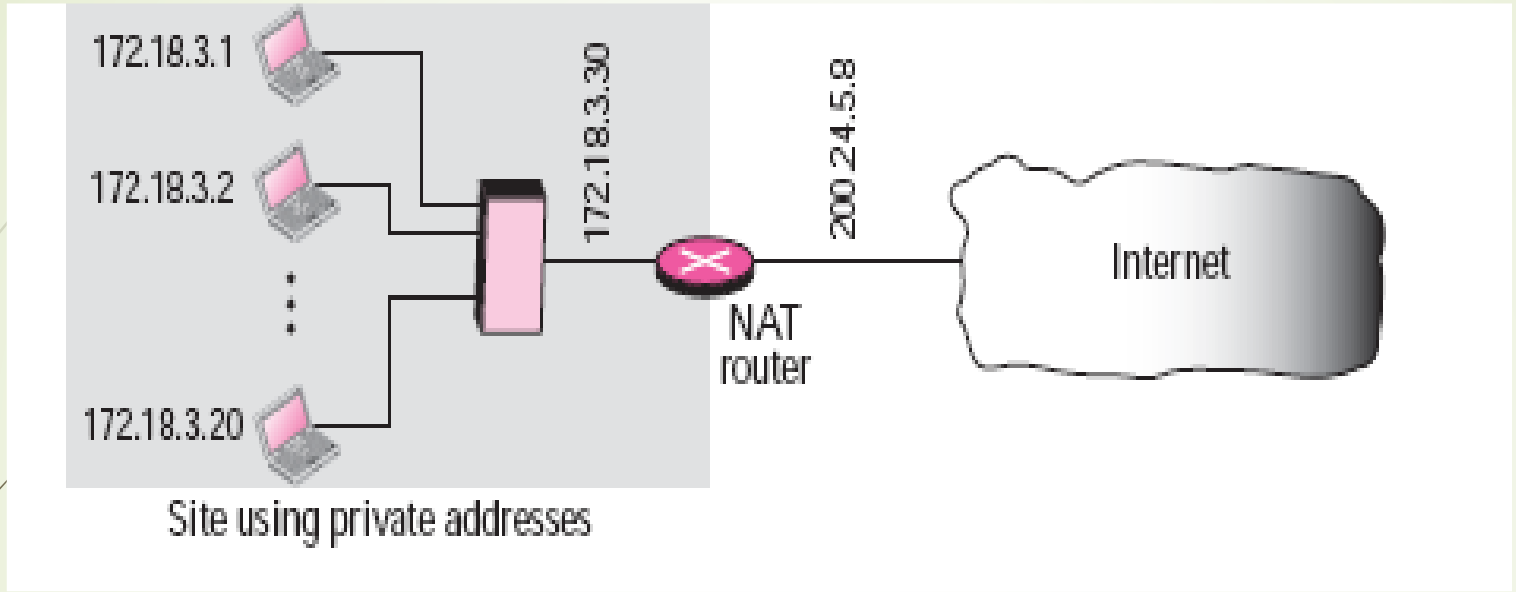
| *Block* | *Number of addresses* | *Block* | *Number of addresses* |
|---|---|---|---|
| 10.0.0.0/8 | 16,777,216 | 192.168.0.0/16 | 65,536 |
| 172.16.0.0/12 | 1,047,584 | 169.254.0.0/16 | 65,536 |

# NAT

- Technology that can provide the mapping between the private and universal addresses is **Network Address Translation (NAT)**
- It allows to **use a set of private addresses for internal communication** and **a set of global Internet addresses** for communication with the rest of the world.
- The site must have only one single connection to the global Internet through a **NAT-capable router** that runs **NAT software**.

# NAT

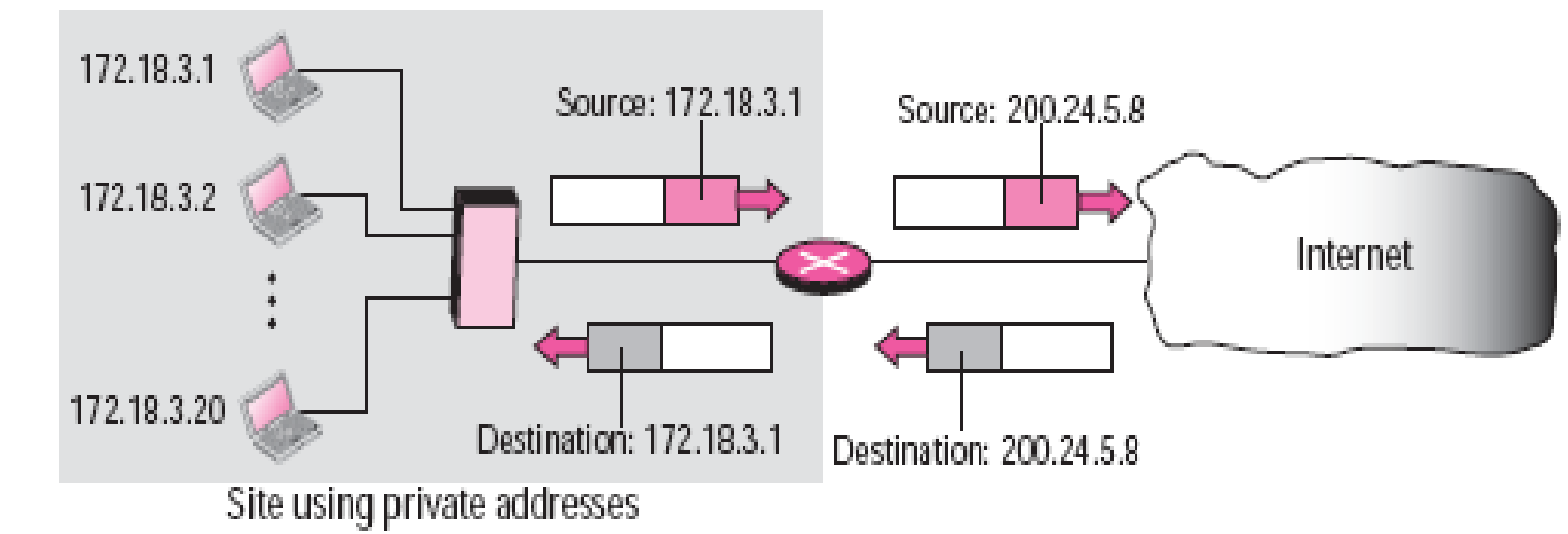





- The private network uses private addresses
- The **router** that connects the network to the global address **uses one private address and one global address**
- The private network is transparent to the rest of the internet
- The rest of the internet sees only the NAT router with address 200.24.5.8

# Address Translation

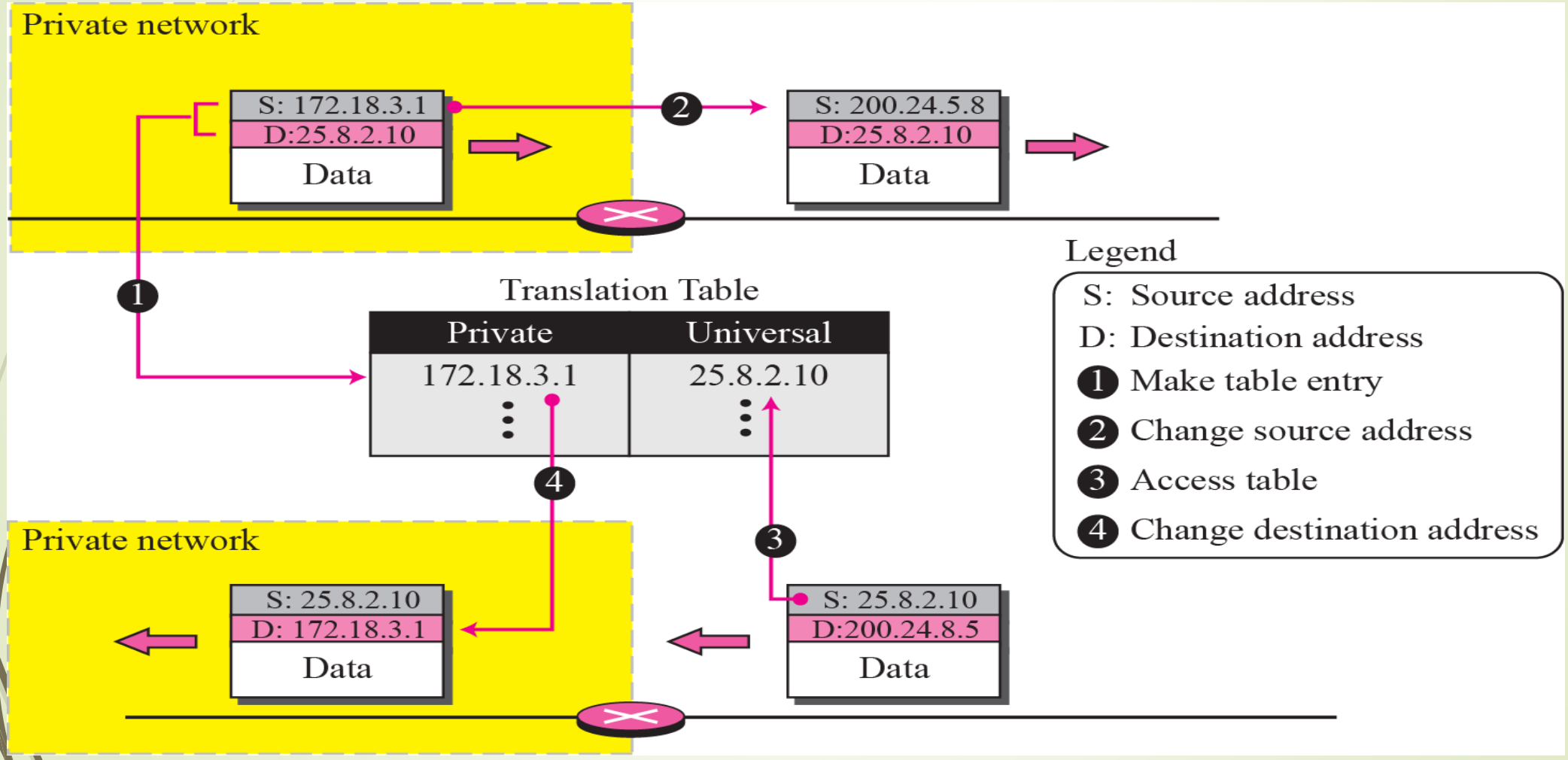





- All of the outgoing packets go through the NAT router, which replaces the ***source address*** in the packet with the global NAT address.
- All incoming packets also pass through the NAT router, which replaces the ***destination address*** in the packet with the appropriate private address.

# Translation Table





# NAT Table

- A basic translation table has only two columns: the private address and the external address (destination address of the packet).
- When the router translates the source address of the outgoing packet, it also makes note of the destination address— where the packet is going.
- When the response comes back from the destination, the router uses the source address of the packet (as the external address) to find the private address of the packet.
- Communication always initiated by the private network

# NAT : Pool of IP address

- For example, instead of using only one global address (200.24.5.8), the NAT router can use four addresses (200.24.5.8, 200.24.5.9, 200.24.5.10, and 200.24.5.11).
- In this case, four private-network hosts can communicate with the same external host at the same time.

# Translation table : Port address

- To allow a **many-to-many relationship** between private-network hosts and external server programs
- For example, **suppose two hosts inside a private network with addresses 172.18.3.1 and 172.18.3.2 need to access the HTTP server on external host 25.8.3.2.**
- The translation table has five columns, instead of two, that include the source and destination port addresses and the transport layer protocol

**Table 5.3** *Five-column translation table*

| *Private Address* | *Private Port* | *External Address* | *External Port* | *Transport Protocol* |
|---|---|---|---|---|
| 172.18.3.1 | 1400 | 25.8.3.2 | 80 | TCP |
| 172.18.3.2 | 1401 | 25.8.3.2 | 80 | TCP |
| ... | ... | ... | ... | ... |