20MCA104: Advanced Computer Networks

Module 2:

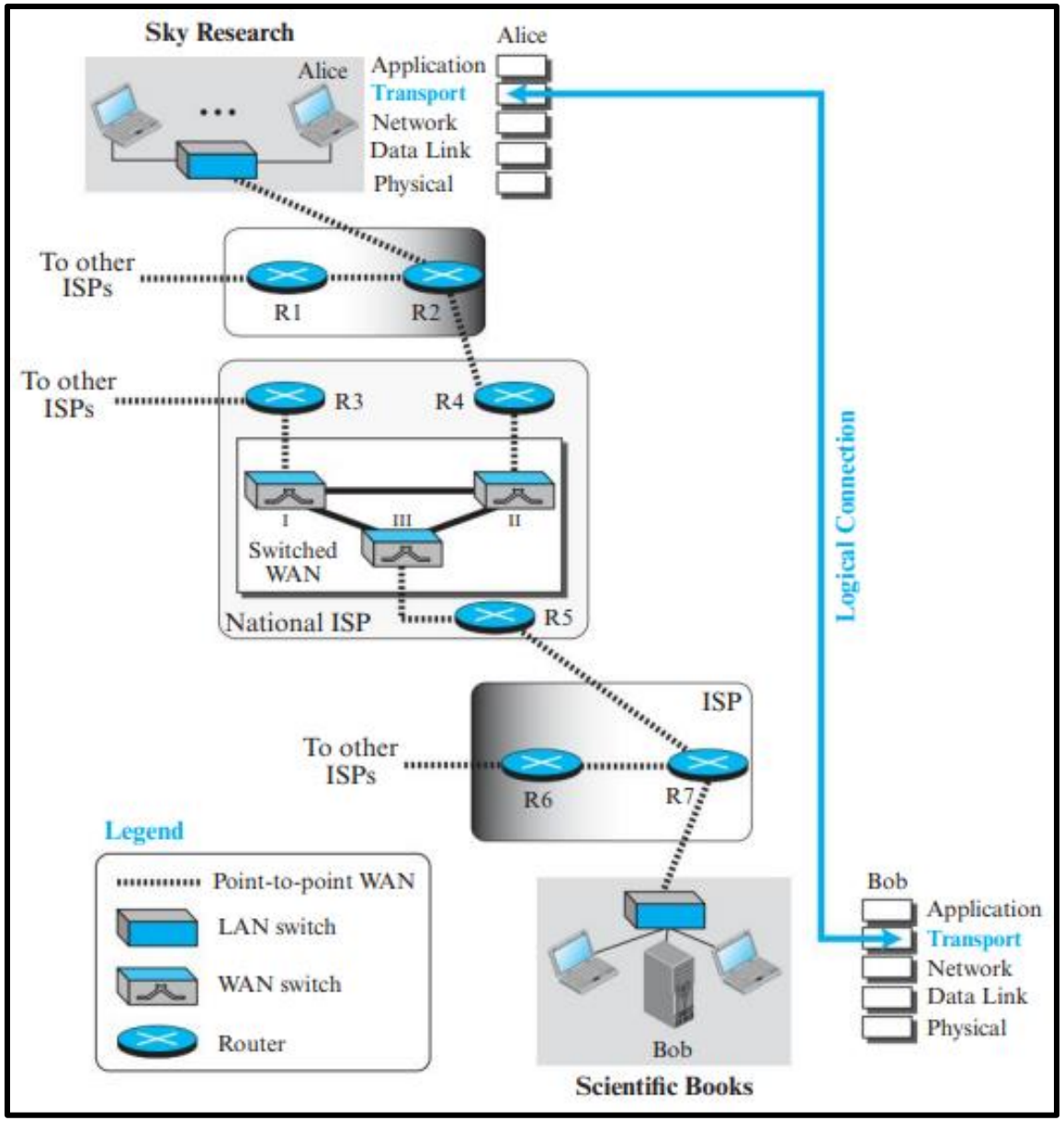
Transport Layer Protocols

Syllabus:

- **Transport Layer Protocols:**
  - Introduction to transport layer
  - Multiplexing and de-multiplexing
  - Principles of Reliable data transfer
    - Stop-and-wait and
    - Go- back- N
  - design and evaluation
  - Connection oriented transport TCP
  - Connectionless transport UDP
  - Principles of congestion control - efficiency and fairness

Transport Layer

- The transport layer in the TCP/IP suite is located between the application layer and the network layer.
- It provides services to the application layer and receives services from the network layer.
- Transport layer offers end-to-end connection and process-to-process communication between two processes on remote hosts.
- The data in the transport layer is called a Segment/Datagram.
- Transport protocols run in end systems.
  - Sender Side: breaks app messages into segments, passes to Network Layer.
  - Receiver Side: reassembles segments into messages, passes to application layer.



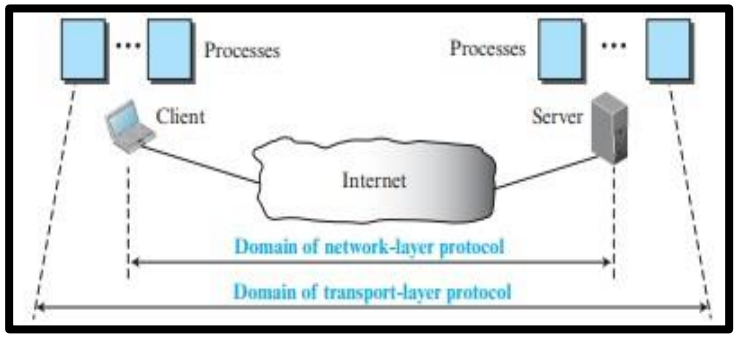
Logical Connection at the transport layer

Transport layer Services

1. Process to Process Communication
2. Addressing
3. Encapsulation and Decapsulation
4. Multiplexing and Demultiplexing
5. Error Control
6. Flow control
7. Congestion Control

Process-to-Process Communication:

- A process is an application-layer entity (running program) that uses the services of the transport layer.
- The network layer is responsible for communication at the computer level (host-to-host communication). A network-layer protocol can deliver the message only to the destination computer which is an incomplete delivery. The message still needs to be handed to the correct process.
- A transport-layer protocol is responsible for delivery of the message to the appropriate process.

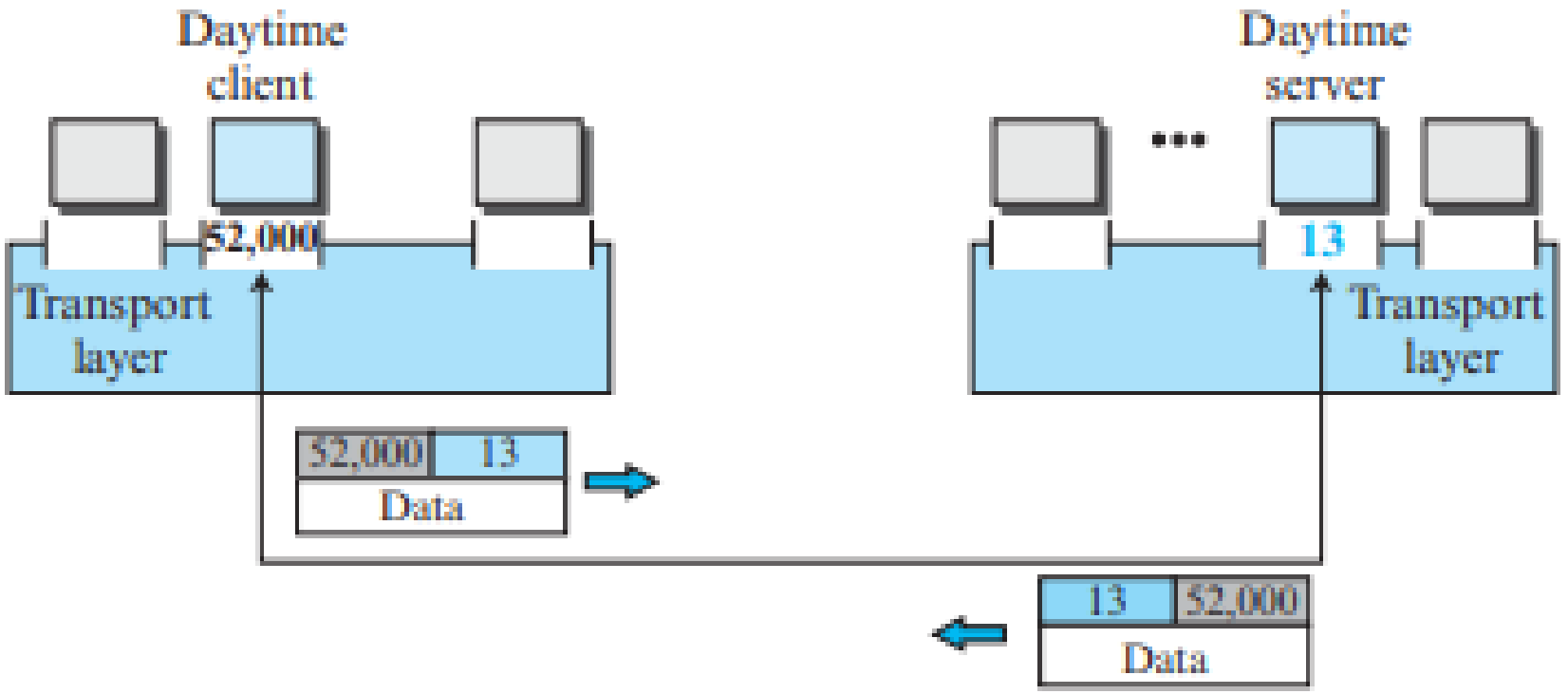


Addressing: Port Numbers

- To achieve process-to-process communication: client-server paradigm.
- A process on the local host, called a client, needs services from a process usually on the remote host, called a server.
- Both processes (client and server) have the same name.
- For example, to get the day and time from a remote machine, we need a daytime client process running on the local host and a daytime server process running on a remote machine.
- A remote computer can run several server programs at the same time, just as several local computers can run one or more client programs at the same time. For communication, we must define
    - local host
    - local process
    - remote host
    - remote process

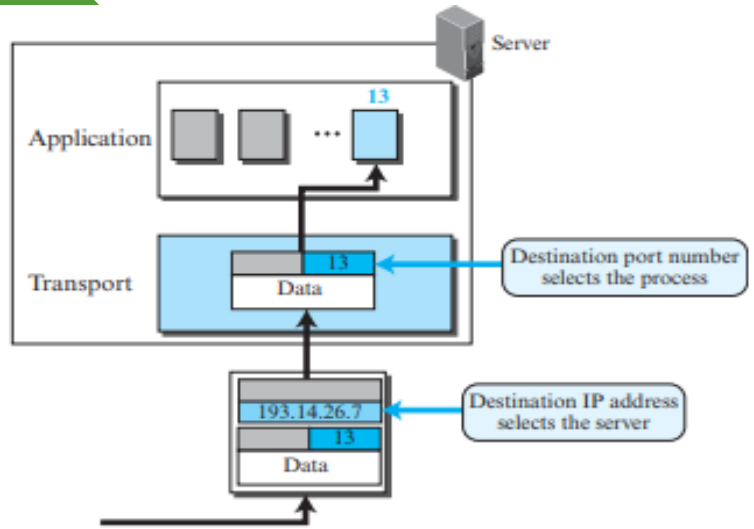
Addressing: Port Numbers

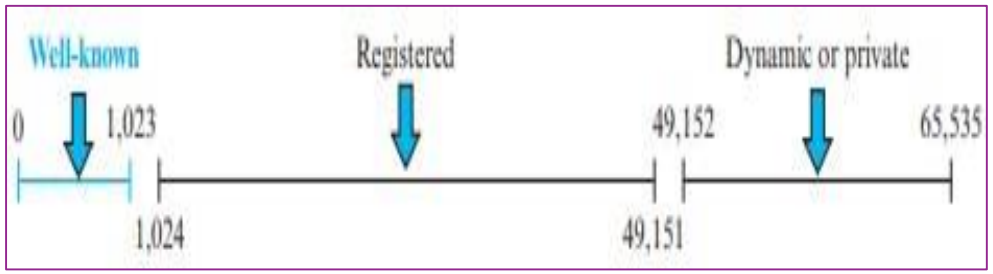
- The local host and the remote host are defined using IP addresses.
- To define the processes, we use second identifiers, called port numbers(In TCP/IP protocol
  suite, integers between 0 and 65,535 (16 bits)).
- The client program defines itself with a port number, called the ephemeral port number.
- An ephemeral port number is greater than 1,023 for some client/server programs.
- The server process must also define itself with a port number. TCP/IP has decided to use universal port numbers for servers; these are called well-known port numbers.
- IP addresses and port numbers play different roles in selecting the final destination of data. The destination IP address defines the host among the different hosts in the world. After the host has been selected, the port number defines one of the processes on this particular host.


Daytime client
Daytime server
52,000
13
Transport layer
Transport layer
52,000
13
Data
13
52,000
Data

Transport Layer:

IP addresses versus port numbers





ICANN Ranges

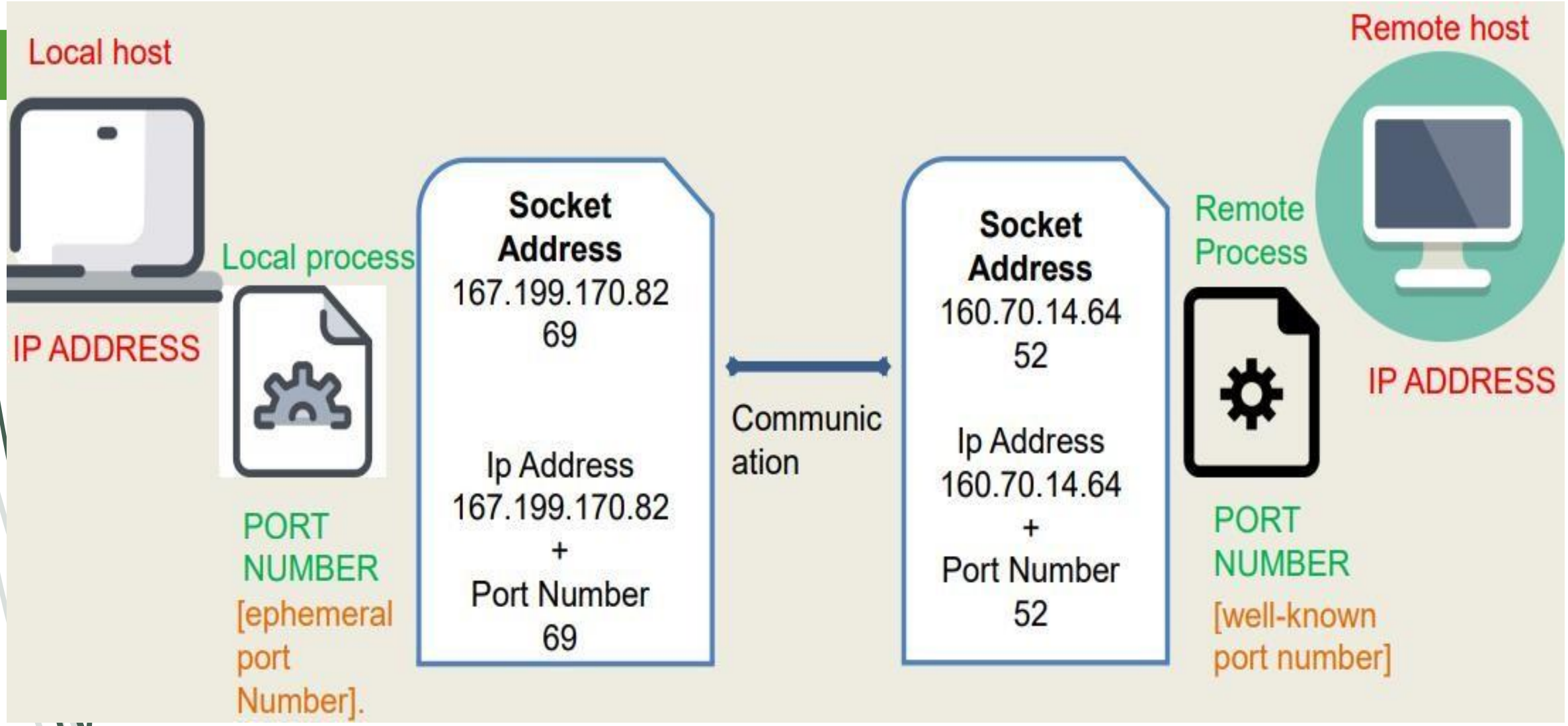
ICANN Ranges

- ICANN has divided the port numbers into three ranges:
- **Well-known ports.** The ports ranging from 0 to 1,023 are assigned and controlled by ICANN. These are the well-known ports.
- **Registered ports.** The ports ranging from 1,024 to 49,151 are not assigned or controlled by ICANN. They can only be registered with ICANN to prevent duplication.
- **Dynamic ports**. The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used as temporary or private port numbers.
- netsh int ipv4 show dynamicport tcp

Socket Addresses

- A transport-layer protocol in the TCP suite needs both the IP address and the port number, at each end, to make a 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 server socket address defines the server process uniquely.
- To use the services of the transport layer in the Internet, we need a pair of socket addresses: the client socket address and the server socket address.

Transport Layer:

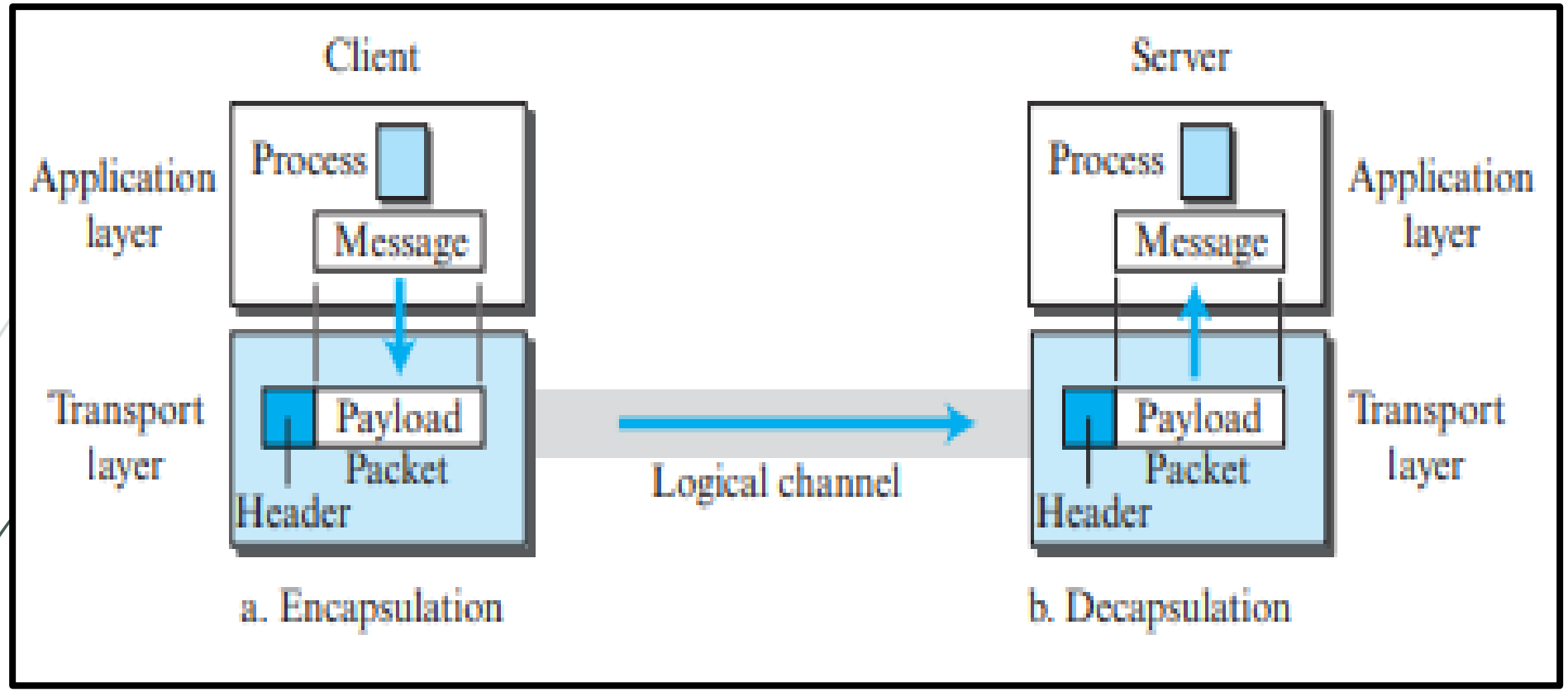


The Transport Layer is responsible for Process-to-Process communication

Encapsulation and Decapsulation

- To send a message from one process to another, the transport-layer protocol encapsulates and decapsulates messages.
- **Encapsulation** happens at the **sender site.** When a process has a message to send, it passes the message to the transport layer along with a pair of socket addresses and some other pieces of information, which depend on the transport-layer protocol.
- The transport layer receives the data and adds the transport-layer header. The packets at the transport layers in the Internet are called user datagrams, segments, or packets, depending on what transport-layer protocol we use.
- **Decapsulation** happens at the **receiver site**. When the message arrives at the destination transport layer, the header is dropped and the transport layer delivers the message to the process running at the application layer.
- The sender socket address is passed to the process in case it needs to respond to the message received.

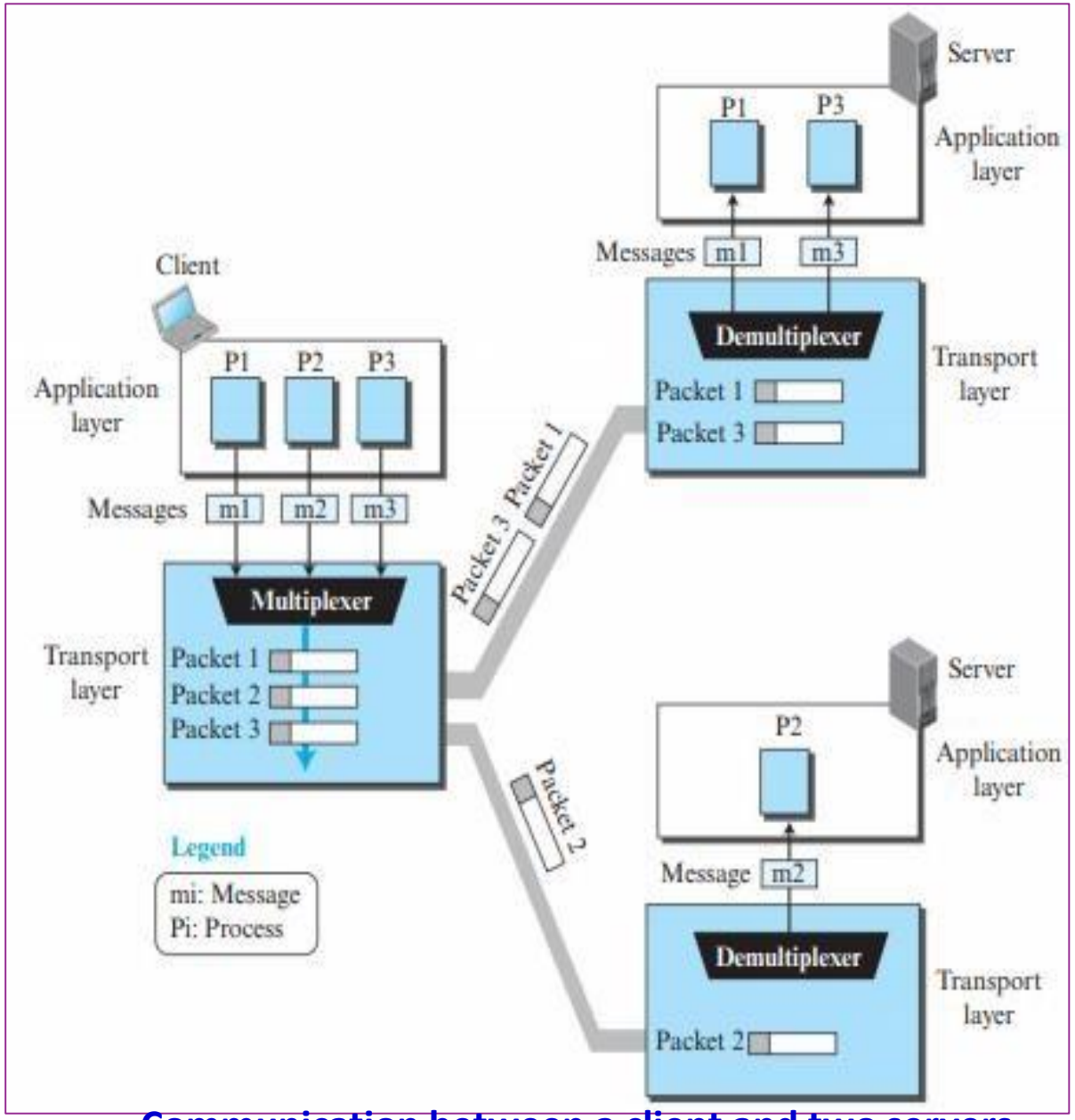
Transport Layer:



Encapsulation and decapsulation

Multiplexing and Demultiplexing

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



Communication between a client and two servers

Transport Layer:

- Three client processes are running at the client site, P1, P2, and P3.
- The processes P1 and P3 need to send requests to the corresponding server process running in a server.
- The client process P2 needs to send a request to the corresponding server process running at another server.
- The transport layer at the client site accepts three messages from the three processes and creates three packets. It acts as a multiplexer.
- The packets 1 and 3 use the same logical channel to reach the transport layer of the first server.
- When they arrive at the server, the transport layer does the job of a demultiplexer and distributes the messages to two different processes.
- The transport layer at the second server receives packet 2 and delivers it to the corresponding process.

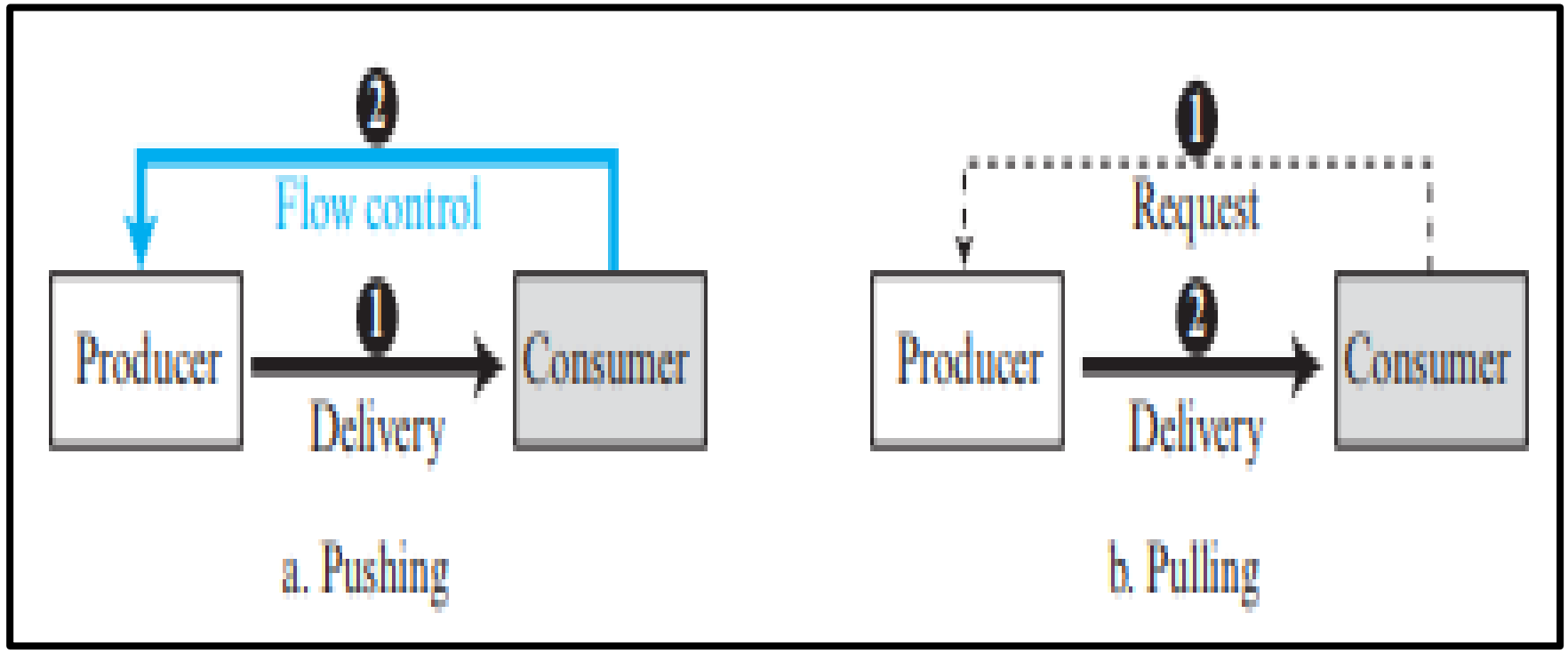
Flow Control

- If the items are produced faster than they can be consumed, the consumer can be overwhelmed and may need to discard some items. If the items are produced more slowly than they can be consumed, the consumer must wait, and the system becomes less efficient. Flow control is related to the first issue. We need to prevent losing the data items at the consumer site.
- Flow Control basically means that TCP will ensure that a sender is not overwhelming a receiver by sending packets faster than it can consume.
- The transport layer provides a flow control mechanism between the adjacent layers of the TCP/IP model.
- TCP also prevents data loss due to a fast sender and slow receiver by imposing some flow control techniques.
- It uses the method of sliding window protocol which is accomplished by the receiver by sending a window back to the sender informing the size of data it can receive.
-

Pushing or Pulling

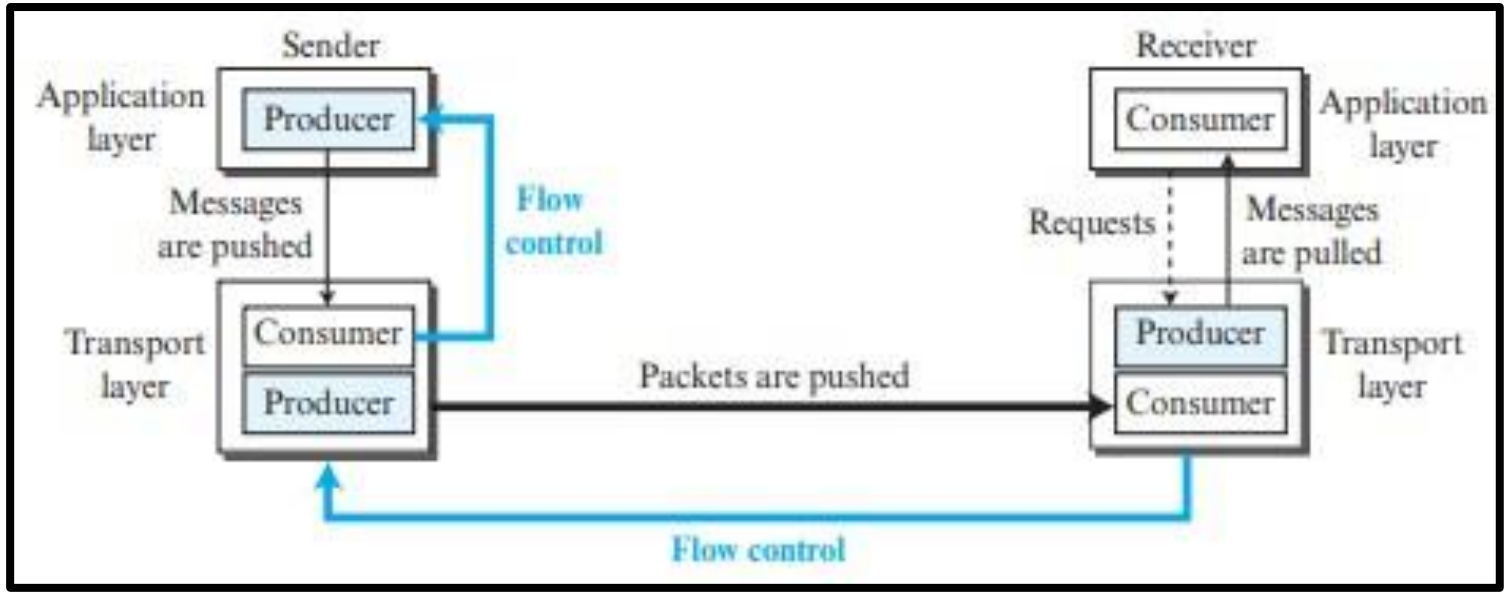
- Delivery of items from a producer to a consumer canoccur in two ways: pushing or pulling.
- If the sender delivers items whenever they are produced without a prior request from the consumer, the delivery is referred to as pushing.
- If the producer delivers the items after the consumer has requested 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.
- In other words, the consumer needs to warn the producer to stop the delivery and to inform it when it is ready again to receive the items.
- When the consumer pulls the items, it requests them when it is ready. So, there is no need for flow control.

Pushing or Pulling



Flow Control at Transport Layer

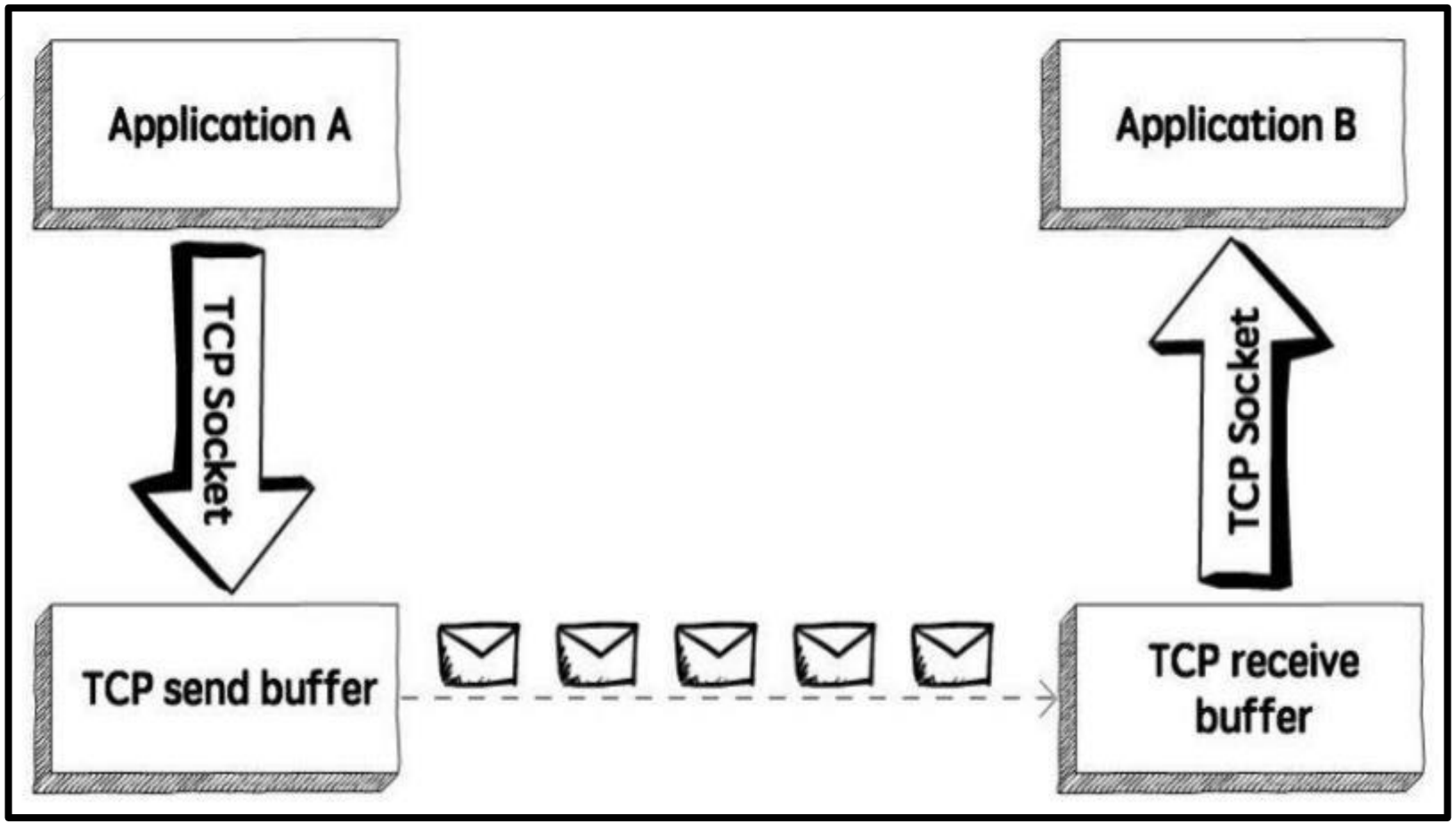
- In communication at the transport layer, dealing with four entities:
  - sender process
  - sender transport layer
  - receiver transport layer
  - receiver process



Buffers

- Flow control can be implemented in several ways, one of the solutions is normally to use two buffers:
  - one at the sending transport layer and the
  - other at the receiving transport layer.
- A buffer is a set of memory locations that can hold packets at the sender and receiver.
- The flow control communication can occur by sending signals from the consumer to the producer.
- When the buffer of the sending transport layer is full, it informs the application layer to stop passing chunks of messages;
- when there are some vacancies, it informs the application layer that it can pass message chunks again.
- When the buffer of the receiving transport layer is full, it informs the sending transport layer to stop sending packets.
- When there are some vacancies, it informs the sending transport layer that it can send packets again.

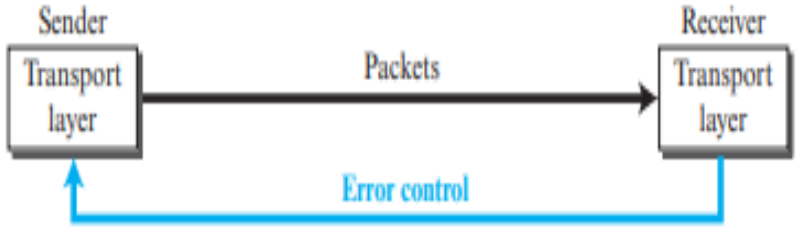
Buffers


Application A
Application B
TCP Socket
TCP Socket
TCP send buffer
TCP receive buffer

Error Control

- In the Internet, since the underlying network layer (IP) is unreliable, we need to make the transport layer Reliable if the application requires reliability.
- Reliability can be achieved to add error control services to the transport layer.
- 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



Sequence Numbers

- **Error control** requires that the sending transport layer knows which packet is to be resend and the receiving transport layer knows which packet is a duplicate, or which packet has arrived out of order.
- This can be done if the packets are numbered. So add a field to the transport-layer packet to hold the sequence number of the packet.
- When a packet is corrupted or lost, the receiving transport layer can somehow inform the sending transport layer to resend that packet using the sequence number.
- The receiving transport layer can also **detect duplicate** packets if two received packets have the **same sequence number**. The **out-of-order packets** can be recognized by observing gaps in the sequence numbers.
- **Packets** are **numbered sequentially**. To include the sequence number of each packet in the header, we need to **set a limit.** If the header of the packet allows **m bits** for the sequence number, the sequence numbers range from **0 to $2^m - 1$.**
- For **error control**, the sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits.

Acknowledgment

- We can use both positive and negative signals as error control.
- The receiver side can send a acknowledgment (ACK) for each of a collection of packets that have arrived safe and sound.
- The receiver can simply discard the corrupted packets.
- The sender can detect lost packets if it uses a timer.
- When a packet is sent, the sender starts a timer.
- If an ACK does not arrive before the timer expires, the sender resends the packet.
- Duplicate packets can be silently discarded by the receiver.
- Out-of-order packets can be either discarded (to be treated as lost packets by the sender), or stored until the missing ones arrives.

Congestion Control

- 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.
- Congestion control refers to the mechanisms and techniques that control the congestion and keep the load below the capacity.
- Congestion in a network or internetwork occurs because routers and switches have queues—buffers that hold the packets before and after processing.
- A router, for example, has an input queue and an output queue for each interface. If a router cannot process the packets at the same rate at which they arrive, the queues become overloaded and congestion occurs.
- Congestion at the transport layer is actually the result of congestion at the network layer, which manifests itself at the transport layer.

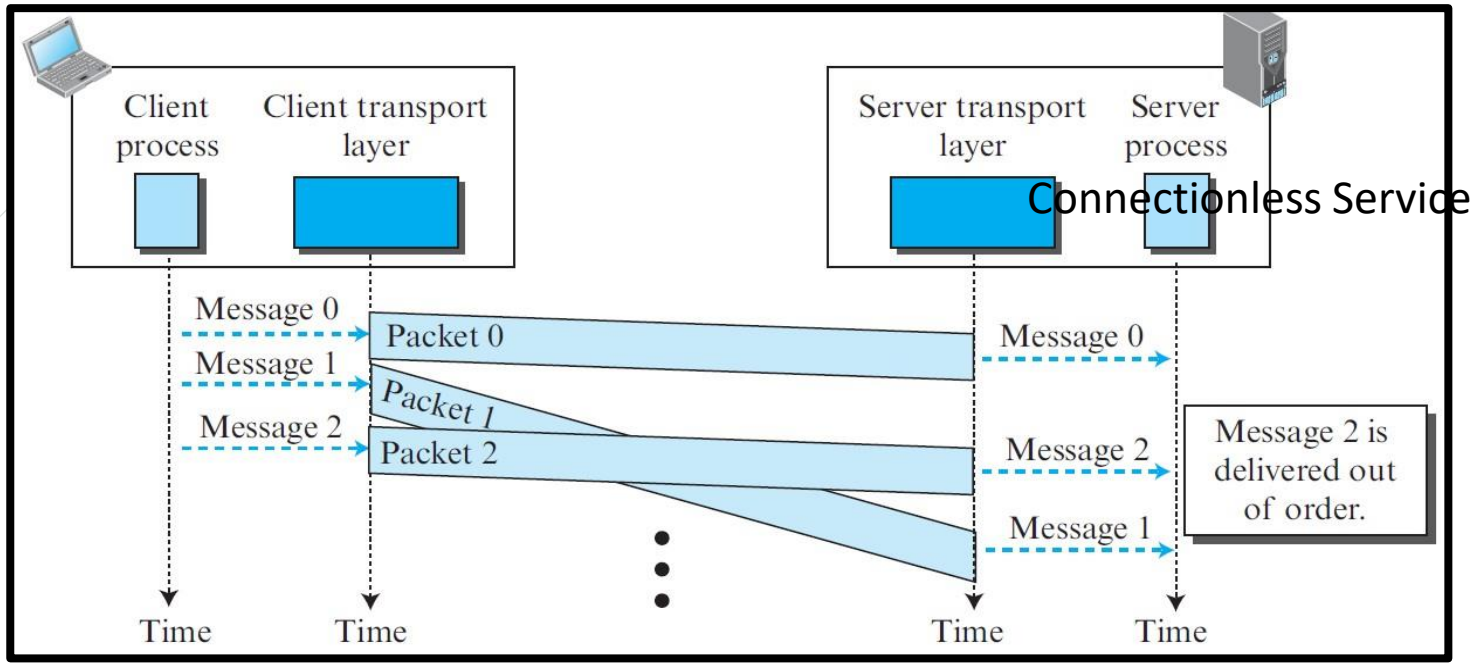
Connectionless and Connection-Oriented Services

A transport-layer protocol can provide two types of services: connectionless and connection-oriented.

❖ **Connectionless Service:**

- In a connectionless service, the source process (application program) needs to divide its message into chunks of data of the size acceptable by the transport layer and deliver them to the transport layer one by one.
- The transport layer treats each chunk as a single unit without any relation between the chunks. When a chunk arrives from the application layer, the transport layer encapsulates it in a packet and sends it.
- To show the independency of packets, assume that a client process has three chunks of messages to send to a server process. The chunks are handed over to the connectionless transport protocol in order.
- Since there is no dependency between the packets at the transport layer, the packets may arrive out of order at the destination and will be delivered out of order to the server process.

Transport Layer:



- The figure shows that at the client site, the three chunks of messages are delivered to the client transport layer in order (0, 1, and 2).
- Because of the **extra delay in transportation** of the second packet, the delivery of messages at the server is not in order (0, 2, 1). If these three chunks of data belong to the same message, the server process may have received a strange message.

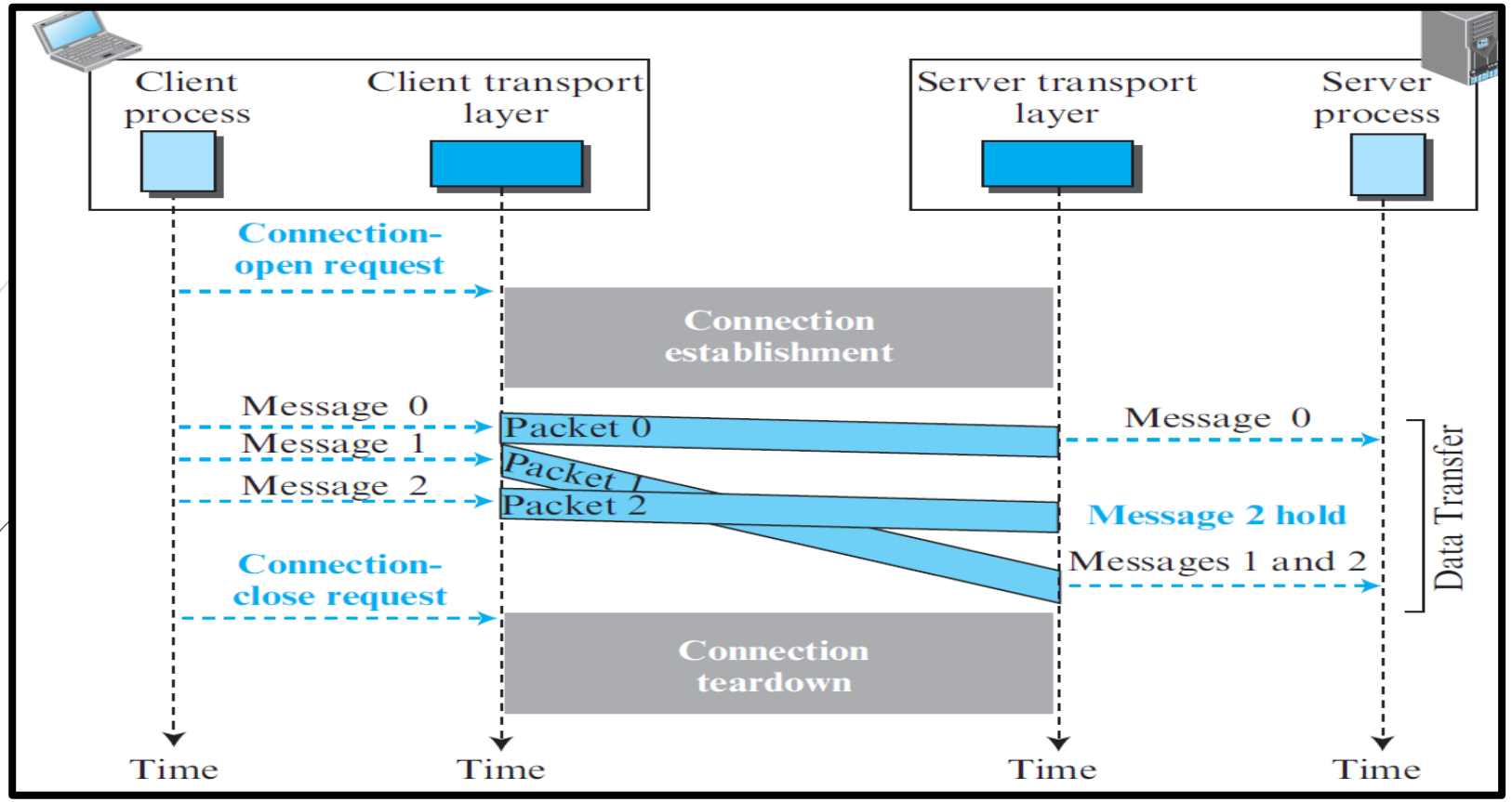
Connectionless and Connection-Oriented Services

- The situation would be worse if one of the packets were lost. Since there is no numbering on the packets, the receiving transport layer has no idea that one of the messages has been lost. It just delivers two chunks of data to the server process.
- These problems arise from the fact that the two transport layers do not coordinate with each other. The receiving transport layer does not know when the first packet will come nor when all of the packets have arrived.
- **No flow control, error control**, or **congestion control** can be effectively implemented in a connectionless service.

Connection-Oriented Service

- In a connection-oriented service, the client and the server first need to establish a logical connection between themselves.
- The data exchange can only happen after the connection establishment.
- After data exchange, the connection needs to be torn down.
- At the transport layer, connection-oriented service involves only the two hosts; the service is end to end.
- This means that we should be able to **make a connection-oriented protocol at the transport layer** over either a connectionless or connection-oriented protocol at the network layer.

Connection-Oriented Service



- Figure shows the connection establishment, data-transfer, and tear-down phases in a connection-oriented service at the transport layer.
- We can implement flow control, error control, and congestion control in a connection oriented protocol.

Transport Layer Protocols

Unidirectional Protocol:

- Simple Protocol
- ✓ Stop-and-Wait Protocol
- ✓ Go-Back-N Protocol
- Selective-Repeat Protocol

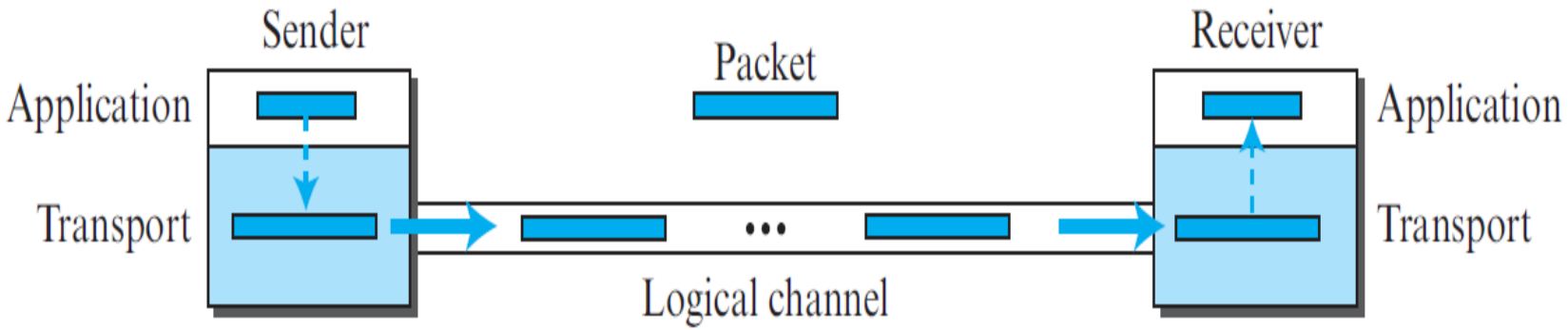
Bidirectional Protocol:

- Piggybacking

Simple Protocol

- This protocol is a simple connectionless protocol with neither flow nor error control.
- We assume that the receiver can immediately handle any packet it receives.
- In other words, the receiver can never be overwhelmed with incoming packets.
- The transport layer at the sender gets a message from its application layer, makes a packet out of it, and sends the packet.
- The transport layer at the receiver receives a packet from its network layer, extracts the message from the packet, and delivers the message to its application layer.
- The transport layers of the sender and receiver provide transmission services for their application layers.

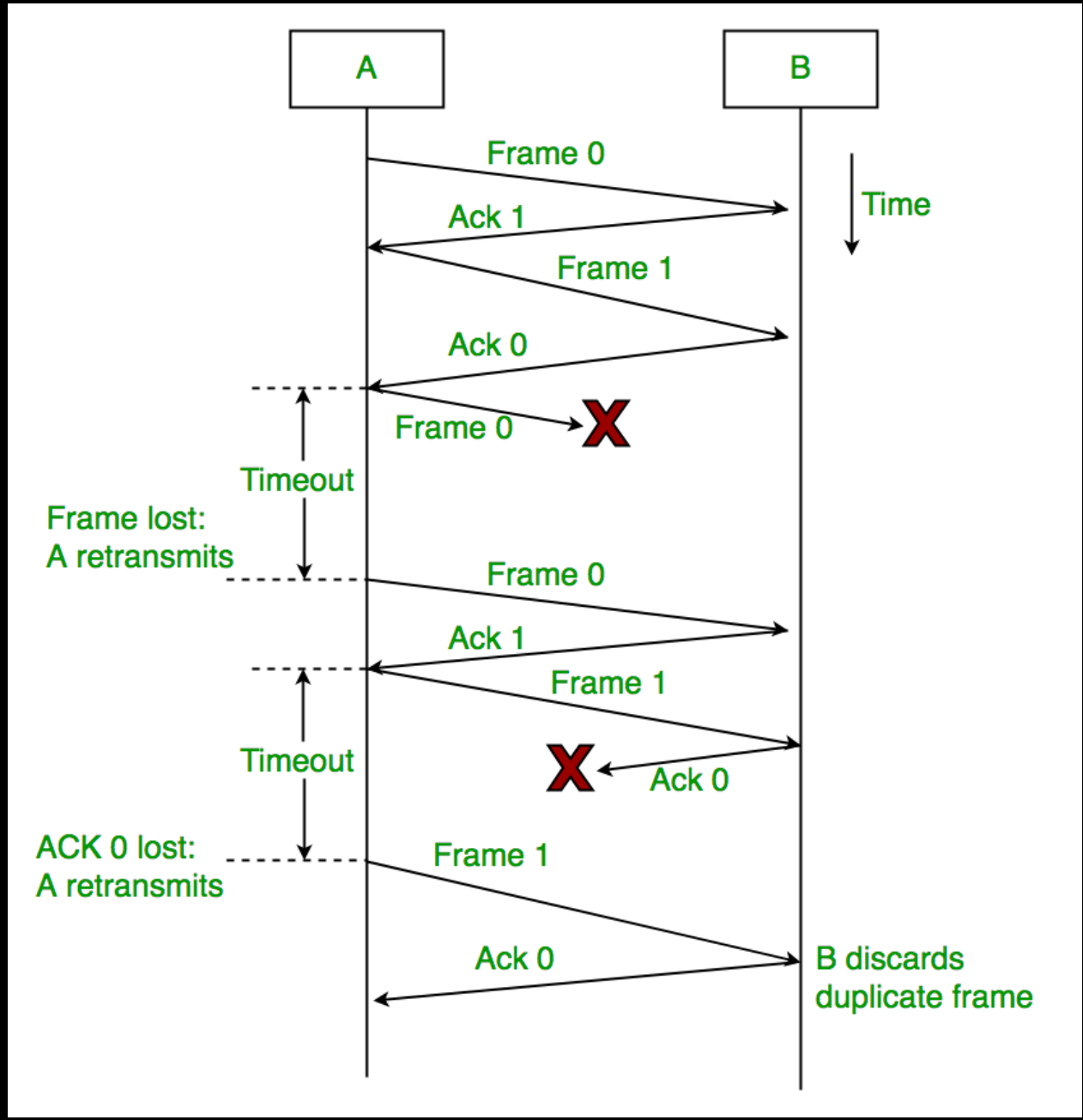
Simple Protocol



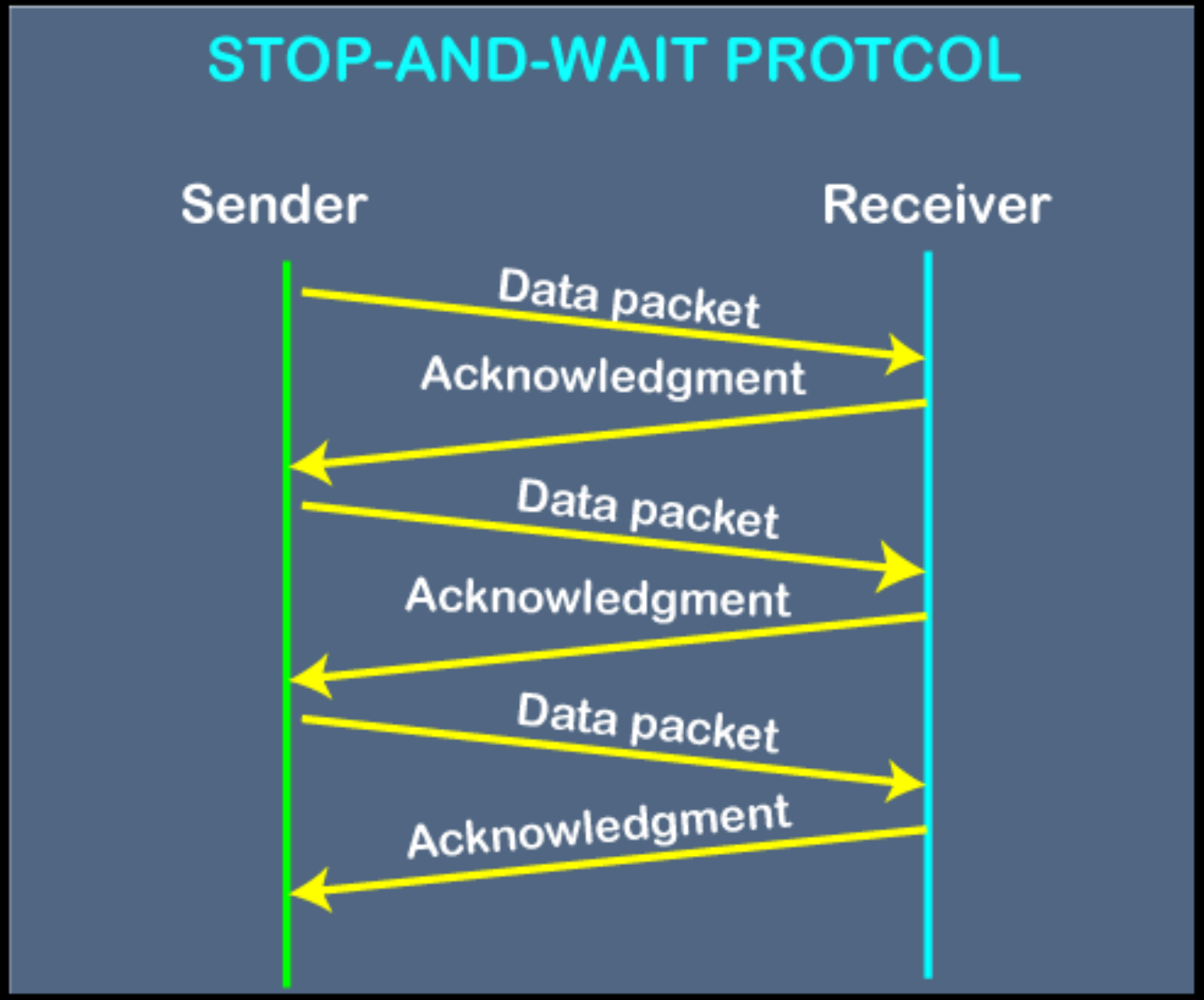
Stop-and-Wait protocol

- Connection-oriented protocol, uses both flow and error control.
- Both the sender and the receiver use a sliding window of size 1.
- The sender sends one packet at a time and waits for an acknowledgment before sending the next one.
- To detect corrupted packets, we need to add a checksum to each data packet.
- When a packet arrives at the receiver site, it is checked.
- If its checksum is incorrect, the packet is corrupted and silently discarded.
- The **silence of the receiver** is a signal for the sender that a packet was either **corrupted** or **lost.**
- Every time the **sender** sends a packet, it starts **a timer**.
- If an **acknowledgment arrives before** the timer expires, the timer is stopped and the sender **sends the next packet** (if it has one to send).

Stop-and-Wait protocol

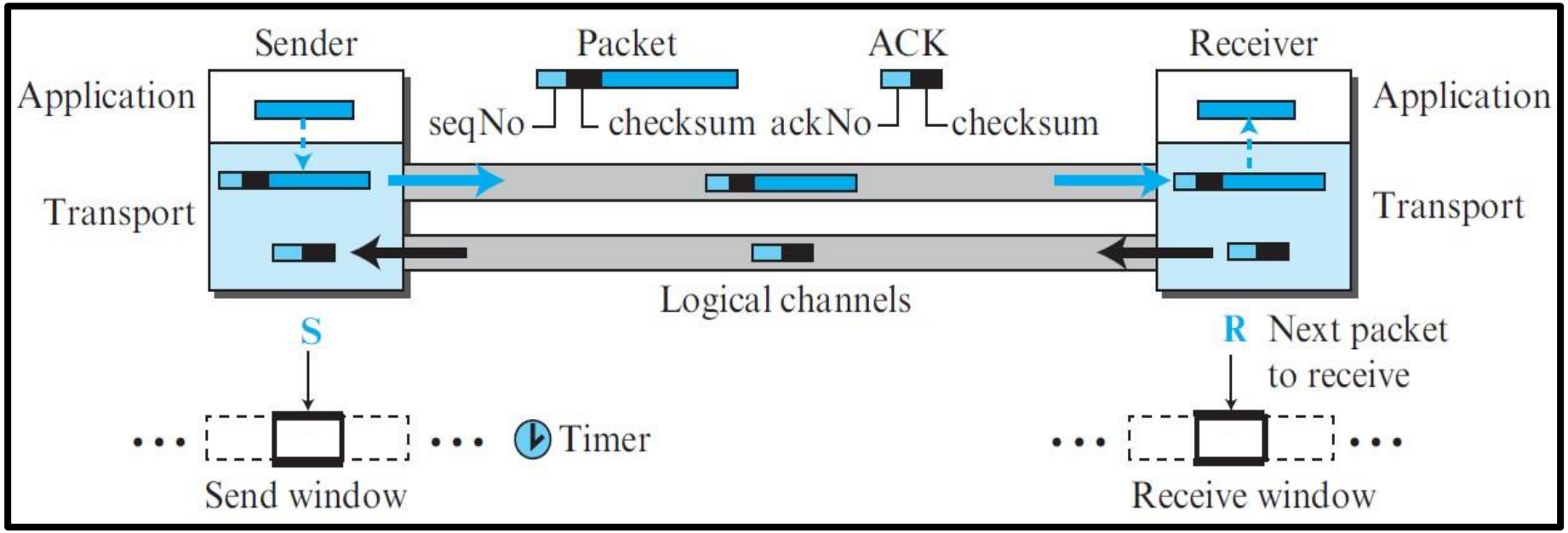


Transport Layer:



Stop-and-Wait protocol

- If the timer expires, the sender resends the previous packet, assuming that the packet was either lost or corrupted.
- This means that the sender needs to keep a copy of the packet until its acknowledgment arrives.
- Here only one packet and one acknowledgment can be in the channels at any time.



Sequence Numbers

- To prevent duplicate packets, the protocol uses sequence numbers and acknowledgment numbers. A field is added to the packet header to hold the sequence number of that number.
- We want to minimize the packet size, we look for the smallest range that provides unambiguous communication.

Acknowledgment Numbers

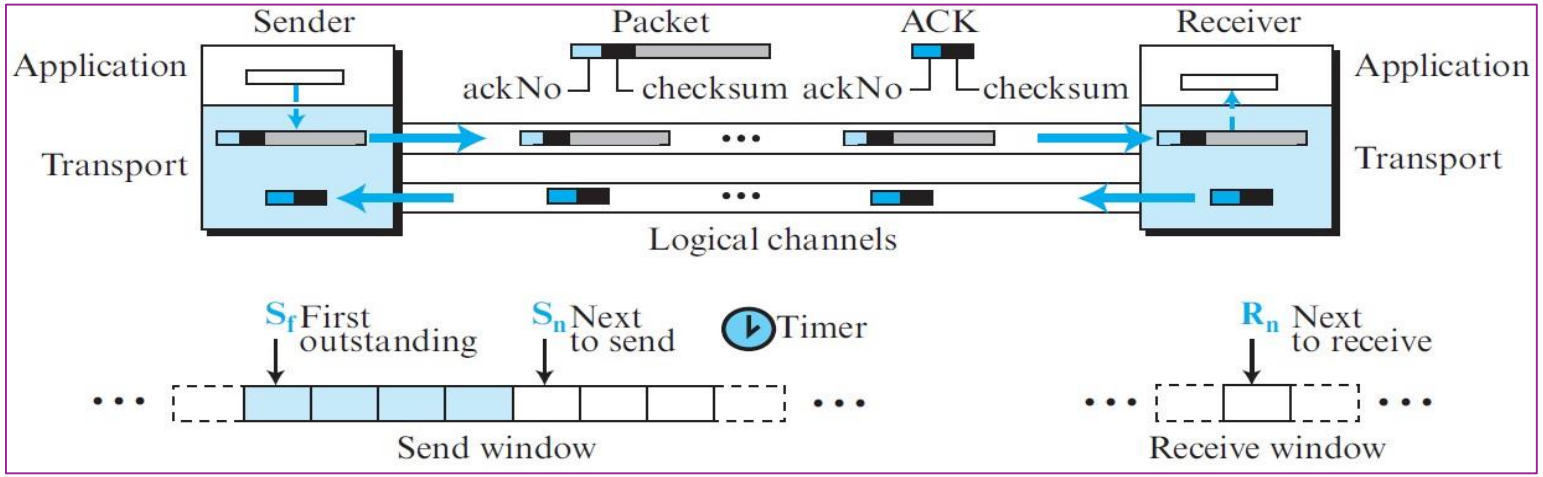
- Since the sequence numbers must be suitable for both data packets and acknowledgments, we use this convention:
- The acknowledgment numbers always announce the sequence number of the next packet expected by the receiver.
- For example, if packet 0 has arrived safe and sound, the receiver sends an ACK with acknowledgment 1 (meaning packet 1 is expected next). If packet 1 has arrived safe and sound, the receiver sends an ACK with acknowledgment 0 (meaning packet 0 is expected).

Transport Layer:

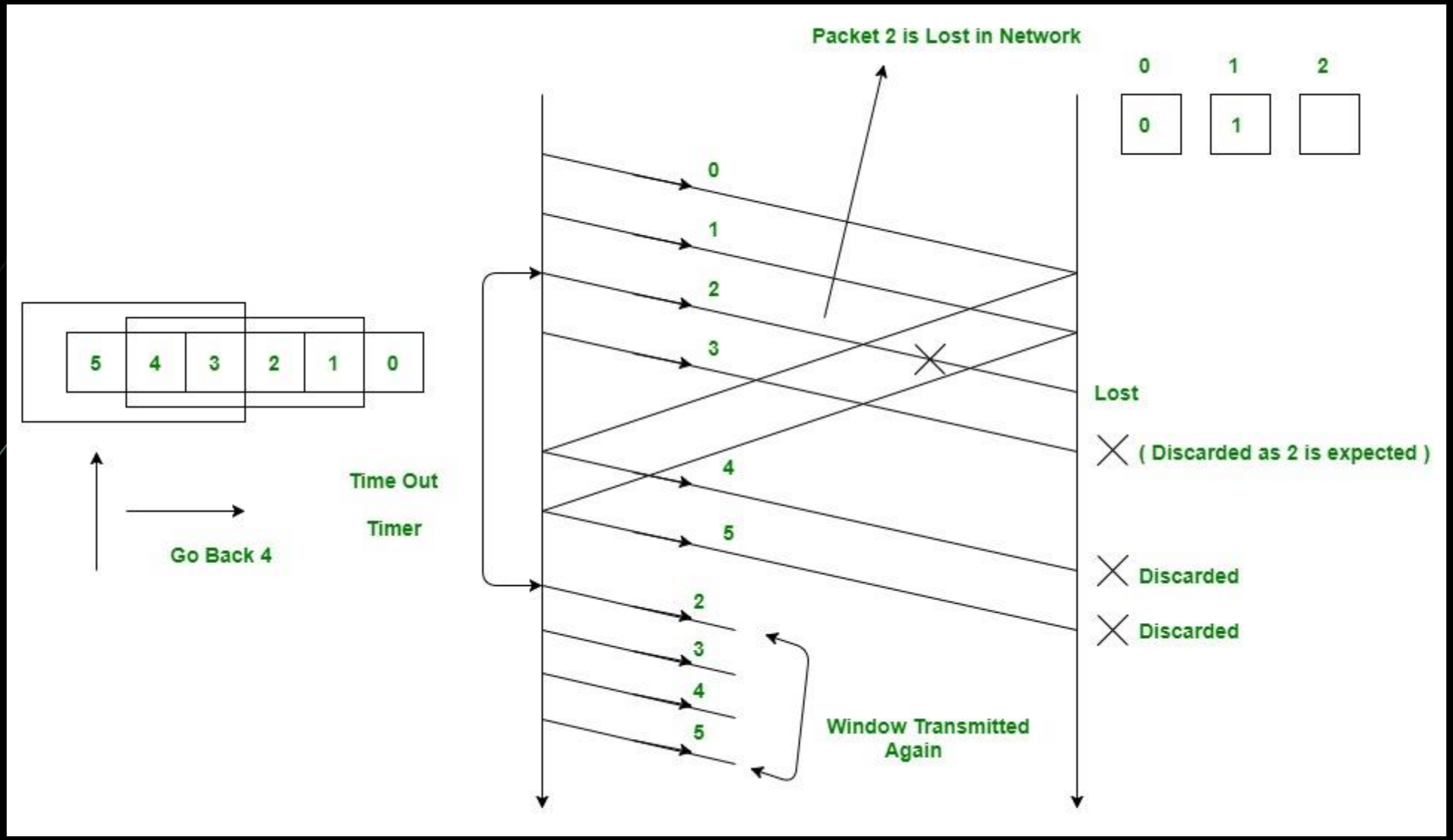
In the Stop-and-Wait protocol, the acknowledgment number always announces, in modulo-2 arithmetic, the sequence number of the next packet expected.

Go-Back-N Protocol (GBN)

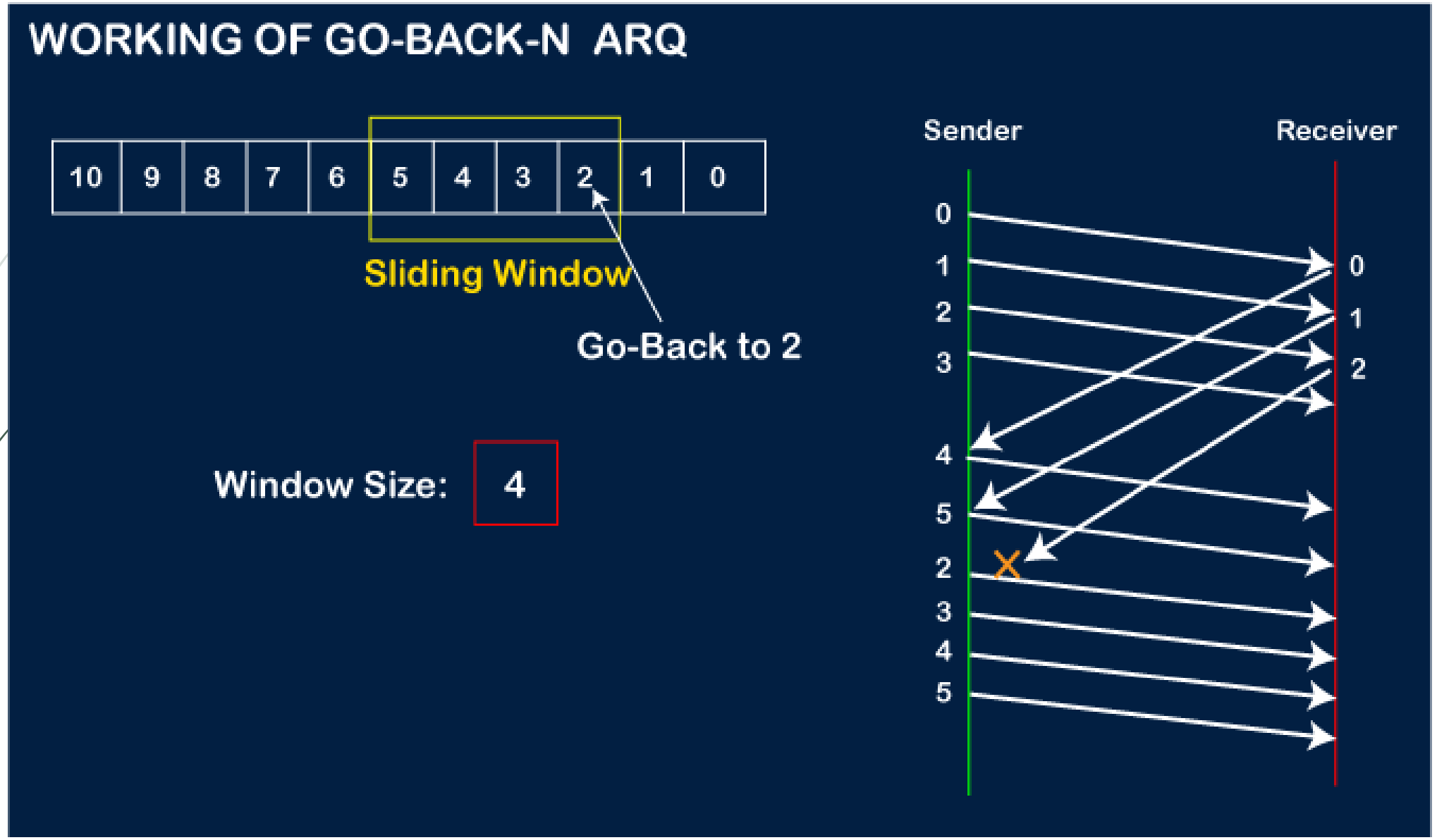
- To improve the efficiency of transmission (to fill the pipe), multiple packets must be in transition while the sender is waiting for acknowledgment.
- In other words, we need to let more than one packet be outstanding to keep the channel busy while the sender is waiting for acknowledgment.
- The key to Go-back-N is that we can send several packets before receiving acknowledgments, but the receiver can only buffer one packet.
- We keep a copy of the sent packets until the acknowledgments arrive.



Go-Back-N Protocol (GBN)



Go-Back-N Protocol (GBN)



Go-Back-N Protocol (GBN)

Sequence Numbers

- The sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits.

Acknowledgment Numbers

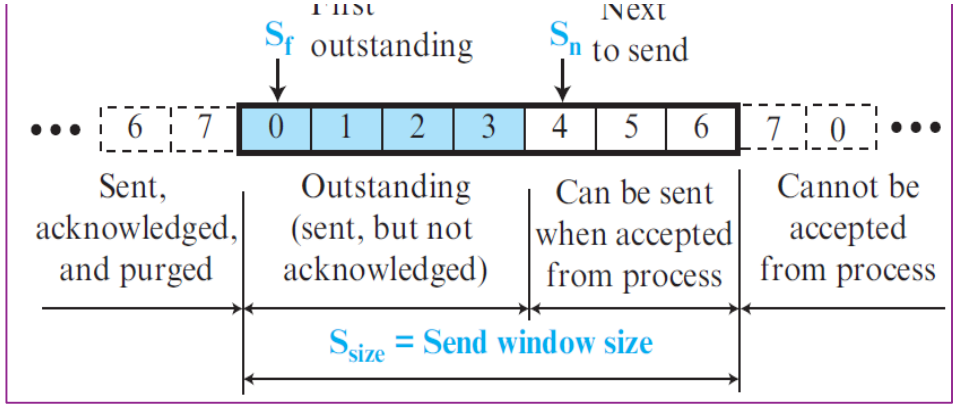
- An acknowledgment number in this protocol is cumulative and defines the sequence number of the next packet expected.
- If the acknowledgment number (ackNo) is 7, it means all packets with sequence number up to 6 have arrived, safe and sound, and the receiver is expecting the packet with sequence number 7.

Send Window

- The send window is an imaginary box covering the sequence numbers of the data packets that can be in transit or can be sent. In each window position, some of these sequence numbers define the packets that have been sent; others define those that can be sent.
- The maximum size of the window is $2^m - 1$.

Go-Back-N Protocol (GBN)

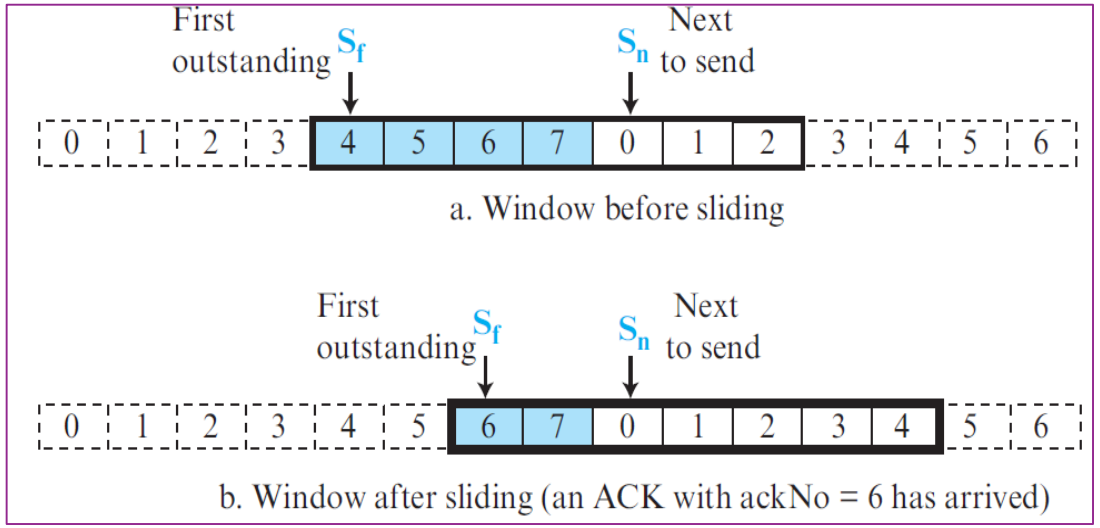
- The send window at any time divides the possible sequence numbers into **four regions**.
- The **first region**, left of the window, defines the sequence numbers belonging to packets that are already acknowledged. The sender does not worry about these packets and keeps no copies of them.
- The **second region**, colored, defines the range of sequence numbers belonging to the packets that have been sent, but have an unknown status. The sender needs to wait to find out if these packets have been received or were lost. We call these outstanding packets.
- The **third region**, white in the figure, defines the range of sequence numbers for packets that can be sent; however, the corresponding data have not yet been received from the application layer.
- The **fourth region**, right of the window, defines sequence numbers that cannot be used until the window slides.



Send window for Go-Back-N

Go-Back-N Protocol (GBN)

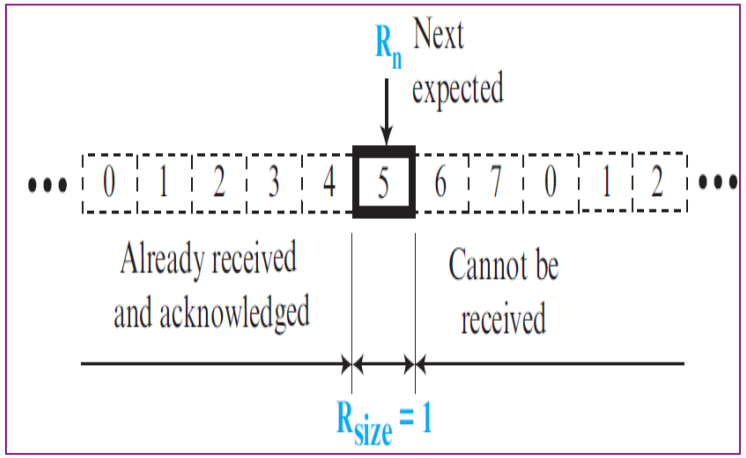
- The window itself is an abstraction; three variables define its size and location at any time.
- **S_f (send window, the first outstanding packet):** The variable S_f defines the sequence number of the first (oldest) outstanding packet.
- **S_n (send window, the next packet to be sent):** The variable S_n holds the sequence number that will be assigned to the next packet to be sent.
- **S_{size} (send window, size):** the variable S_{size} defines the size of the window, which is fixed in our protocol.



Go-Back-N Protocol (GBN)

Receive Window

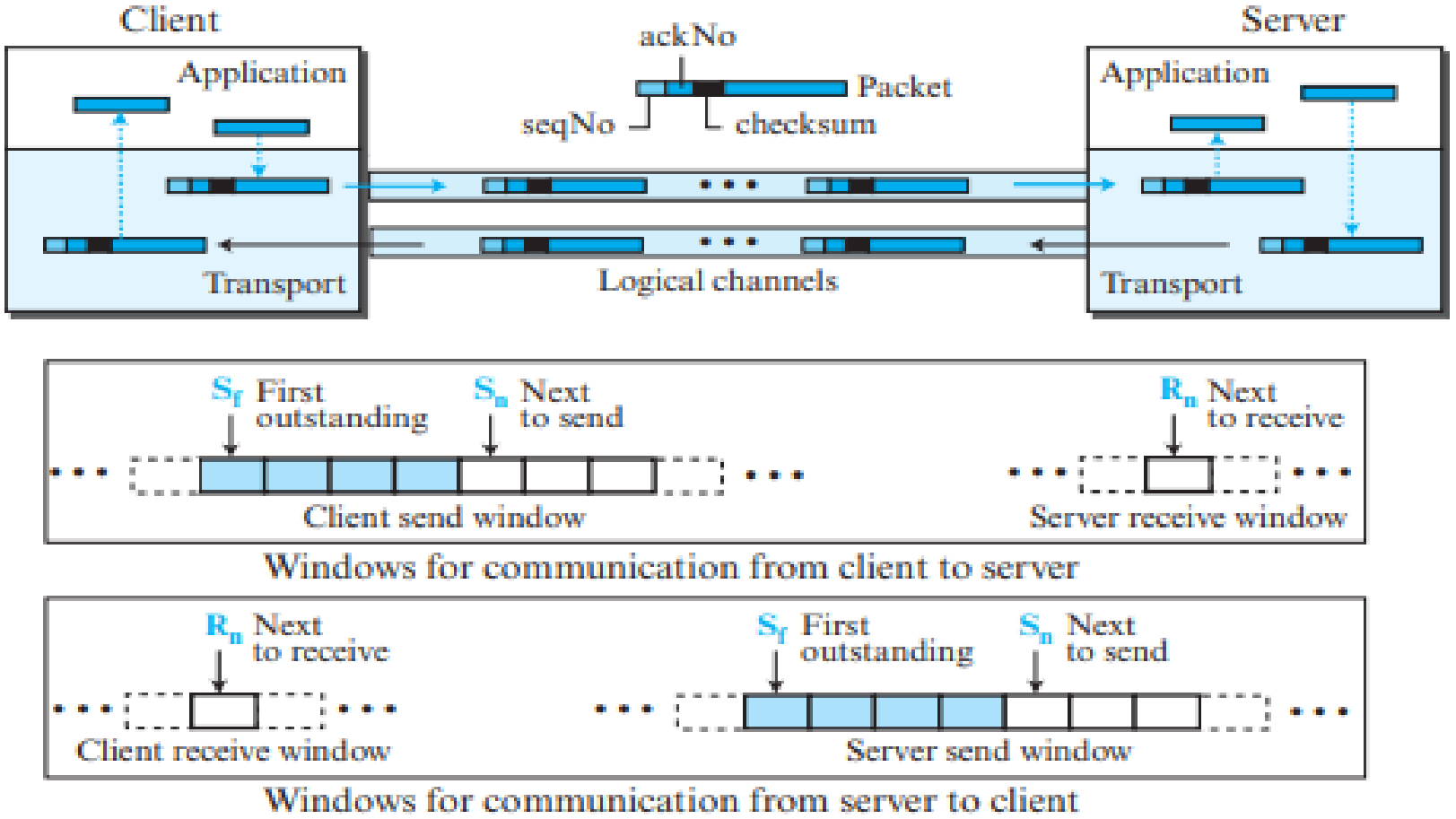
- The receive window makes sure that the correct data packets are received and that the correct acknowledgments are sent.
- In Go-back-N, the size of the receive window is always 1.
- The receiver is always looking for the arrival of a specific packet. Any packet arriving out of order is discarded and needs to be resent.
- Here we have one variable, R_n (receive window, next packet expected), to define this abstraction.
- The sequence numbers to the left of the window belong to the packets already received and acknowledged; the sequence numbers to the right of this window define the packets that cannot be received. Any received packet with a sequence number in these two regions is discarded.
- Only a packet with a sequence number matching the value of R_n is accepted and acknowledged.



Resending packets

- When the timer expires, the sender resends all outstanding packets.
- For example, suppose the sender has already sent packet 6 (Sn = 7), but the only timer expires. If Sf = 3, this means that packets 3, 4, 5, and 6 have not been acknowledged; the sender goes back and resends packets 3, 4, 5, and 6. That is why the protocol is called Go-Back-N.
- On a time-out, the machine goes back N locations and resends all packets.

- The Figure shows the layout for the GBN protocol implemented bidirectionally using piggybacking. The client and server each use two independent windows: send and receive.



Design of piggybacking in Go-Back-N

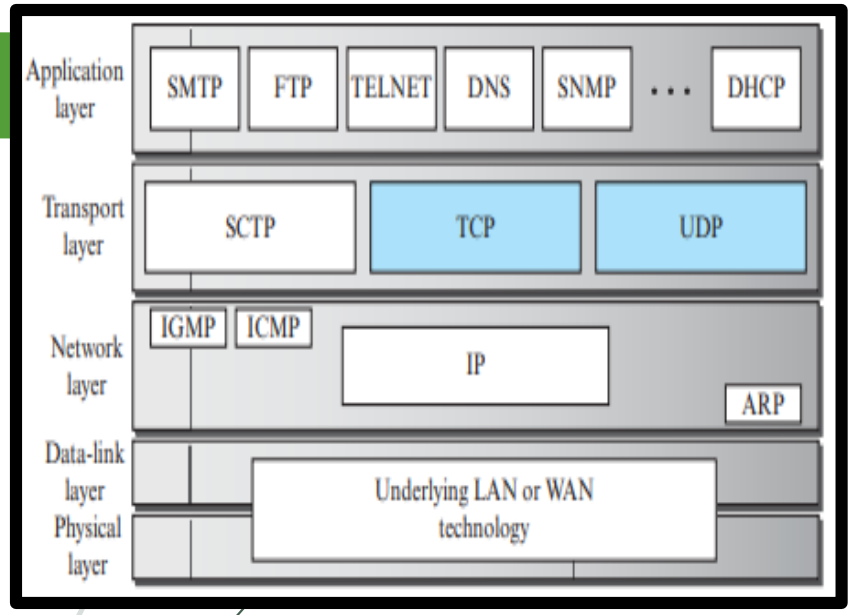
Transport Layer protocols:

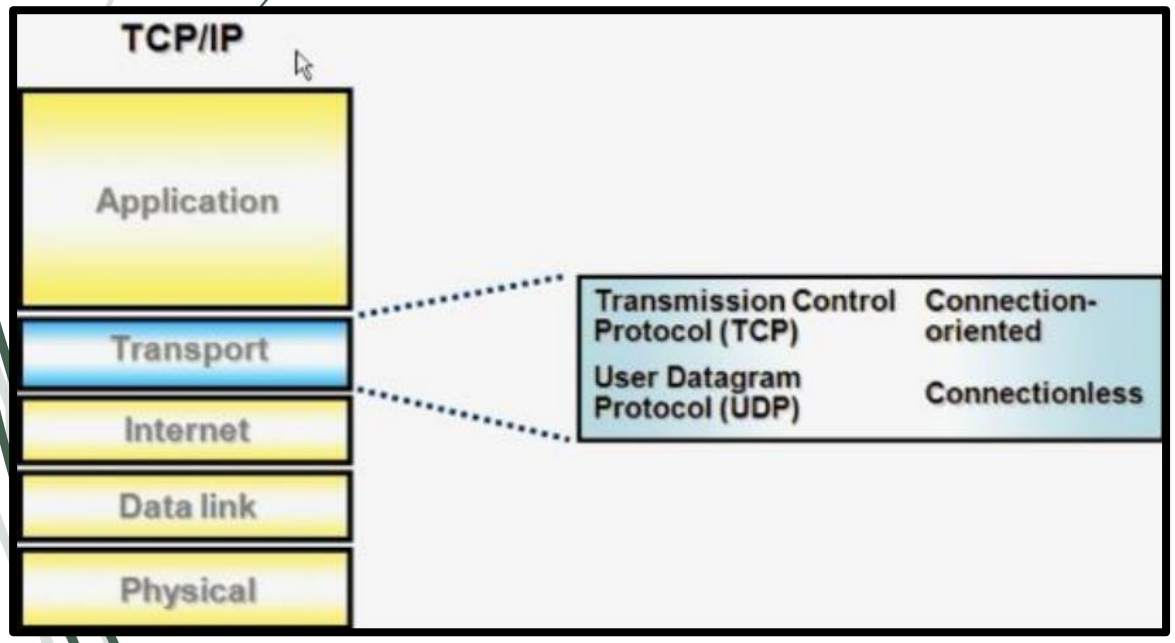
- Connection oriented transport TCP.
- Connectionless transport UDP.

Internet Transport-Layer Protocols: UDP and TCP

- These protocols are located between the application layer and the network layer and serve as the intermediary between the application programs and the network operations.
- UDP is an unreliable connectionless transport-layer protocol used for its simplicity and efficiency in applications where error control can be provided by the application-layer process.
- TCP is a reliable connection-oriented protocol that can be used in any application where reliability is important.
- There are other transport-layer protocols, such as SCTP.
- A transport-layer protocol usually has several responsibilities.
  - One is to create a **process-to-process communication**;
    - these protocols uses **port numbers** to accomplish this.

Position of transport-layer protocols in the TCP/IP protocol suite





Port	*Protocol*	*UDP*	*TCP*	*Description*
7	Echo	√		Echoes back a received datagram
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

Port	*Protocol*	*UDP*	*TCP*	*Description*
19	Chargen	√	√	Returns a string of characters
20, 21	FTP		√	File Transfer Protocol
23	TELNET		√	Terminal Network
25	SMTP		√	Simple Mail Transfer Protocol
53	DNS	√	√	Domain Name Service
67	DHCP	√	√	Dynamic Host Configuration Protocol
69	TFTP	√		Trivial File Transfer Protocol
80	HTTP		√	Hypertext Transfer Protocol
111	RPC	√	√	Remote Procedure Call
123	NTP	√	√	Network Time Protocol
161, 162	SNMP		√	Simple Network Management Protocol

User Datagram Protocol (UDP)

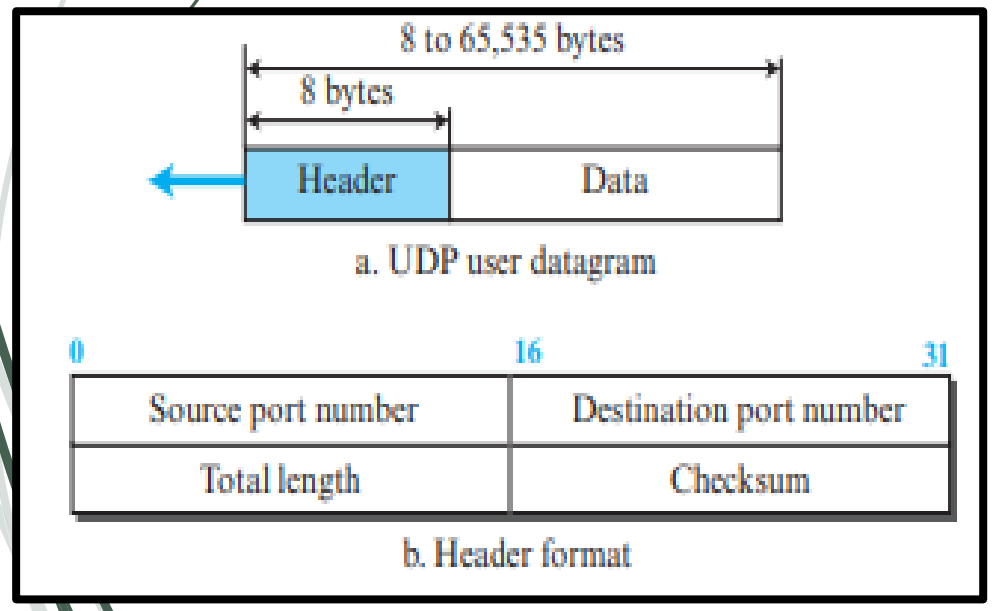
- The User Datagram Protocol (UDP) is a connectionless, unreliable transport protocol.
- It does not add anything to the services of IP except for providing process-to-process communication instead of host-to-host communication.
- UDP is a very simple protocol using a minimum of overhead.
- If a process wants to send a small message and does not care much about reliability, it can use UDP.
- Sending a small message using UDP takes much less interaction between the sender and receiver than using TCP.

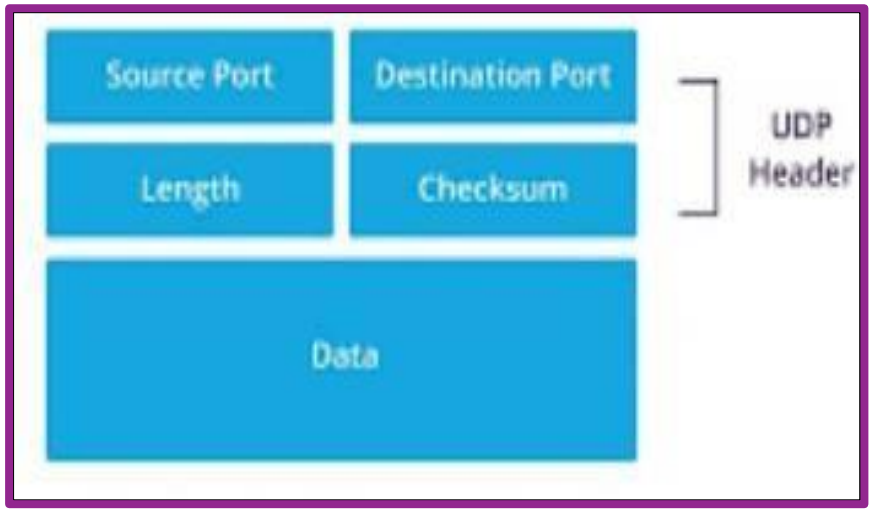
User Datagram

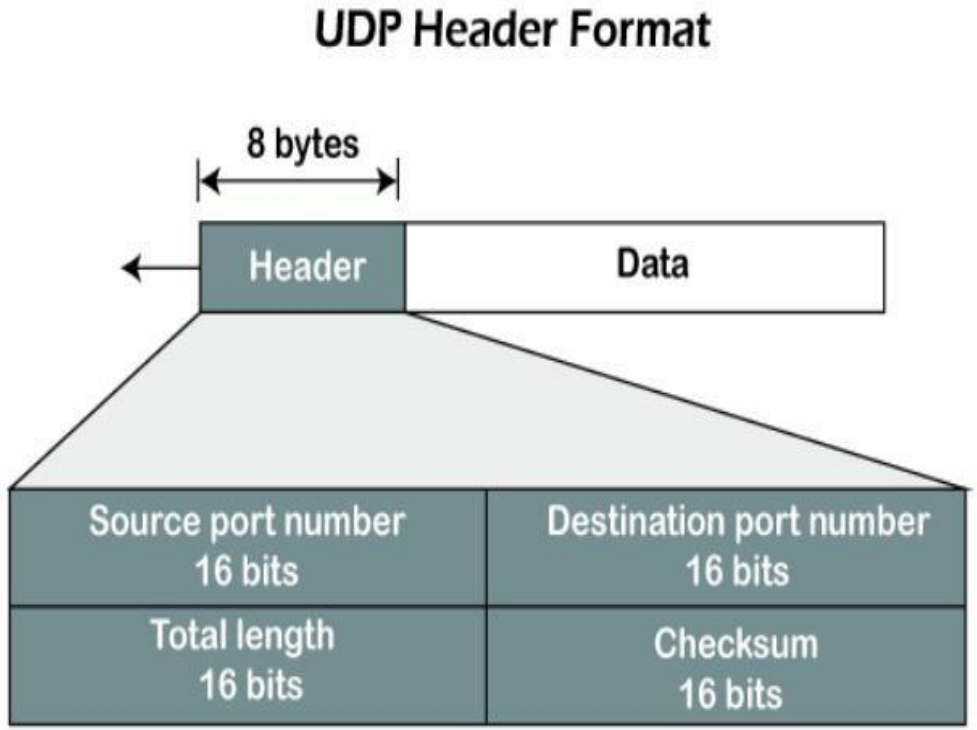
- **UDP packets**, called user datagrams, have a **fixed-size header of 8 bytes** made of four fields, each of 2 bytes (16 bits).

User Datagram Protocol (UDP)

- The Figure shows the format of a user datagram. The first two fields define the source and destination port numbers.
- The third field defines the total length of the user datagram, header plus data.
- The 16 bits can define a total length of 0 to 65,535 bytes.
- However, the total length needs to be less because a UDP user datagram is stored in an IP datagram with the total length of 65,535 bytes.
- The last field can carry the optional checksum.







- The UDP header contains **four fields:**

i. **Source port number:** It is 16-bit information that identifies which port is going to send the packet.

ii. **Destination port number:** It identifies which **port** is going to **accept the information.** It is **16-bit** information which is used to identify application level service on the destination machine.

iii. **Length: It is 16-bit field** that specifies the entire length of the UDP packet that includes the header also. The **minimum value** would be **8-byte** as the size of the header is 8 bytes.

iv. **Checksum**: It is a **16-bits field**, and it is an **optional field**. This checksum field **checks whether the information is accurate or not** as there is the possibility that the information can be **corrupted** while transmission. It is an optional field, which means that it depends upon the application, whether it wants to write the checksum or not. If it does not want to write the checksum, then all the 16 bits are zero; otherwise, it writes the checksum. In UDP, the checksum field is applied to the entire packet, i.e., header as well as data part whereas, in IP, the checksum field is applied to only the header field.

UDP Services

- **Process-to-Process Communication:**UDP provides process-to-process communication using **socket addresses**, a combination of IP addresses and port numbers.
- **Connectionless Services:** UDP provides a connectionless service. This means that each user datagram sent by UDP is an independent datagram. There is no relationship between the different user datagrams even if they are coming from the same source process and going to the same destination program. The user datagrams are not numbered. Also, unlike TCP, there is no connection establishment and no connection termination. This means that each user datagram can travel on a different path.
- **Flow Control:** UDP is a very simple protocol. There is no flow control, and hence no window mechanism. The receiver may overflow with incoming messages. The lack of flow control means that the process using UDP should provide for this service, if needed.

UDP Services

- **Error Control:** 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. The **lack of error control** means that the **process using UDP should provide** for this service, if needed.
- **Checksum:** UDP checksum calculation includes **three sections:**
  - a pseudoheader
  - the UDP header
  - the data coming from the application layer.
- The **pseudoheader** is the **part of the header of the IP packet** in which the user datagram is to be encapsulated with some fields **filled with 0s** .If the checksum does **not include the pseudoheader, a user datagram** may **arrive safe** and sound. However, if the IP header is corrupted, it may be delivered to the wrong host.
- The **protocol field** is added to ensure that the packet belongs to UDP, and not to TCP. The value of the protocol field for **UDP is 17.**
- If this value is changed during transmission, the checksum calculation n at the receiver will detect it and UDP drops the packet. It is not delivered to the wrong protocol.

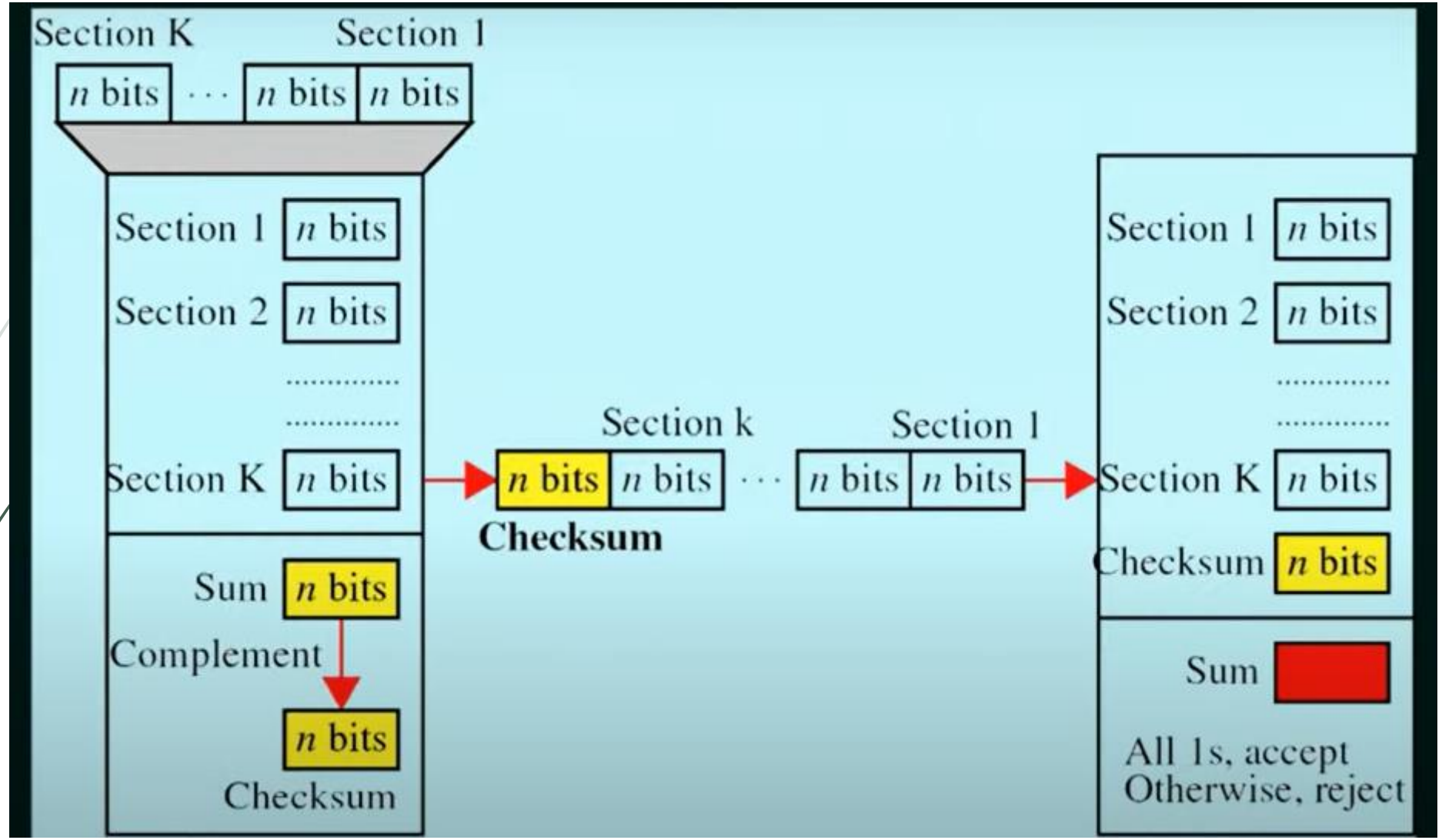
UDP Services

Checksum – Operation at sender side

1. Break the original message in to 'k' number of blocks with 'n' bits in each block.
2. Sum all the 'k' data blocks.
3. Add the carry to the sum, if any.
4. Do 1's complement to the sum = Checksum.

Obtained checksum is appended with the data and it is send

UDP Services



UDP Services

Checksum operation at the receiver side

- At the receiver side, the received data is divided into k blocks of n bits.
- Sum all the k data blocks
- Add the carry to the sum, if any
- If the obtained sum is all 1's accept, otherwise reject.

UDP Services

Checksum – Example

Consider the data unit to be transmitted is:

1001100111100010001001001000100

10011001	11100010	00100100	10000100

Checksum – Example

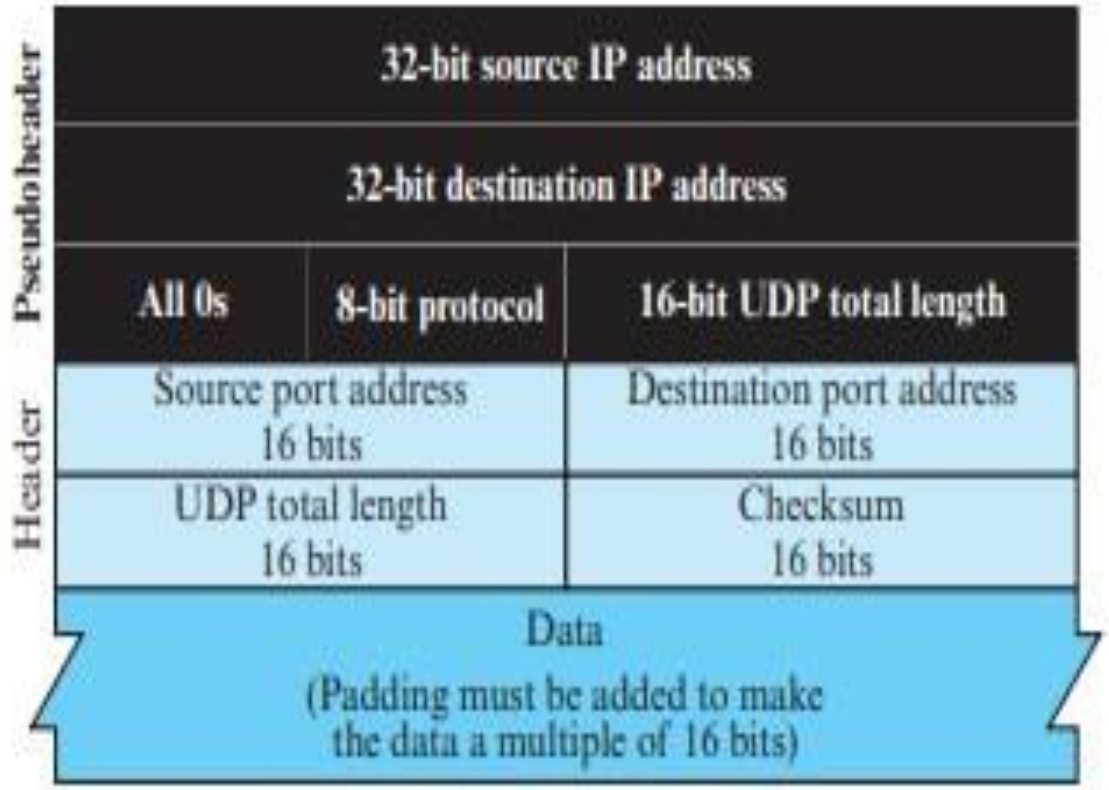
10011001	11100010	00100100	10000100

Carry		1	1	1	1	1			
		1	0	0	0	0	1	0	0
		0	0	1	0	0	1	0	0
		1	1	1	0	0	0	1	0
		1	0	0	1	1	0	0	1
1	0	0	0	1	0	0	0	1	1

Checksum – Example

10011001	11100010	00100100	10000100

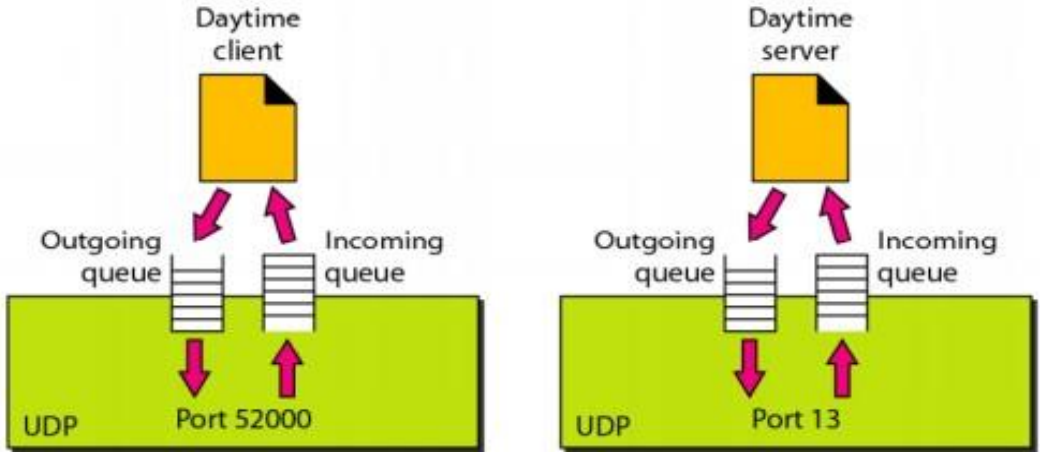
Carry	1	1	1	1	1			
	1	0	0	0	0	1	0	0
	0	0	1	0	0	1	0	0
	1	1	1	0	0	0	1	0
	1	0	0	1	1	0	0	1
	0	0	1	0	0	0	1	1
							1	0



Pseudoheader for checksum calculation

- **Optional Inclusion of Checksum:** The sender of a UDP packet can choose not to calculate the checksum. In this case, **the checksum field** is filled with **all 0s before being sent**. In the situation where the sender decides to calculate the checksum, but it happens that the result is all 0s, the checksum is changed to all 1s before the packet is sent. In other words, the sender complements the sum two times.

- **Congestion Control:** Since UDP is a connectionless protocol, it does not provide congestion control. UDP assumes that the packets sent are small and sporadic and cannot create congestion in the network. This assumption may or may not be true today, when UDP is used for interactive real-time transfer of audio and video.
- **Encapsulation and Decapsulation:** To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages.
- **Queuing:** In UDP, queues are associated with ports. At the client site, when a process starts, it requests a port number from the operating system. Some implementations create both an incoming and an outgoing queue associated with each process. Other implementations create only an incoming queue associated with each process.



- **Multiplexing and Demultiplexing:** In a host running a TCP/IP protocol suite, there is only one UDP but possibly several processes that may want to use the services of UDP. To handle this situation, UDP multiplexes and demultiplexes.

- **Lack of Error Control:** UDP does not provide error control; it provides an unreliable service.
- Most applications expect reliable service from a transport-layer protocol. Although a reliable service is desirable, it may have some side effects that are not acceptable to some applications.
- When a transport layer provides reliable services, if a part of the message is lost or corrupted, it needs to be resent. This means that the receiving transport layer cannot deliver that part to the application immediately; there is an uneven delay between different parts of the message delivered to the application layer.
- Some applications by nature do not even notice these uneven delays, but for some they are
very crucial.
- **Lack of Congestion Control:** UDP does not provide congestion control.UDP does not create additional traffic in an error-prone network. TCP may resend a packet several times and thus contribute to the creation of congestion or worsen a congested situation.
- Therefore, in some cases, lack of error control in UDP can be considered an advantage when congestion is a big issue.

Typical Applications:

Applications that can benefit more from the services of UDP than from those of TCP.

1. UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control. It is not usually used for a process such as FTP that needs to send bulk data .
2. UDP is suitable for a process with internal flow- and error-control mechanisms. For example, the Trivial File Transfer Protocol (TFTP) process includes flow and error control. It can easily use UDP.
3. UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but not in the TCP software.
4. UDP is used for management processes such as SNMP .
5. UDP is used for some route updating protocols such as Routing Information Protocol (RIP)
6. UDP is normally used for interactive real-time applications that cannot tolerate uneven delay between sections of a received message .

Transmission Control Protocol (TCP)

- A **connection-oriented** communications protocol that facilitates the exchange of messages between computing devices in a network. It is the **most common protocol** in networks that **use the Internet Protocol (IP)**; **together** they are sometimes **referred to as TCP/IP.**
- TCP explicitly defines **connection establishment, data transfer**, and **connection teardown** phases to provide a connection-oriented service.
- TCP uses a combination of GBN and SR protocols to **provide reliability**. To achieve this goal, TCP uses checksum (for error detection), **retransmission of lost or corrupted packets,** cumulative and **selective acknowledgments**, and **timers.**
- TCP takes messages from an application/server and **divides them into packets**, which can then be forwarded by the devices in the network – switches, routers, security gateways – to the destination. **TCP numbers each packet** and **reassembles them** prior to handing them off to the application/server recipient. Because it is connection-oriented, it ensures a connection is established and maintained until the exchange between the application/servers sending and receiving the message is complete.

Transmission Control Protocol (TCP)

TCP Services

- The Transmission Control Protocol works together with IP and provides a reliable transport service between processes using the network layer service provided by the IP protocol.
  - **Process-to-Process Communication**
  - **Stream delivery service**
  - **Multiplexing & De-multiplexing**
  - **Full duplex service**
  - **Connection oriented service**
  - **Reliability**

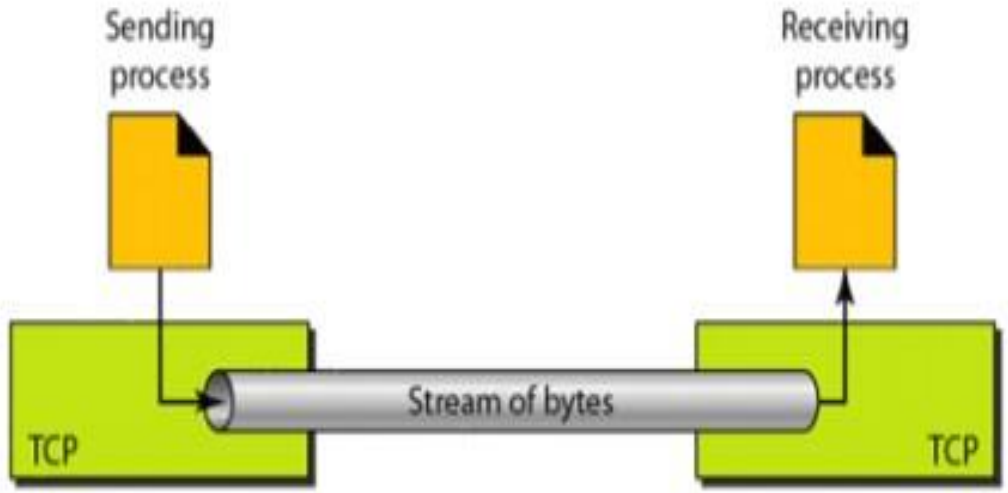
Transmission Control Protocol(TCP)

- **Process-to-Process Communication** – TCP provides process to process communication using port numbers or port addresses. Port numbers helps identify which process is sending or receiving data on a host.
- **Multiplexing & De-multiplexing** – Like UDP, TCP performs multiplexing at the sender and de-multiplexing at the receiver. However, since TCP is a connection- oriented protocol, a connection needs to be established for each pair of process.
- **Full duplex service –** 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.
- **Connection oriented service –** TCP, unlike UDP, is a connection-oriented protocol. When a process at site A wants to send to and receive data from another process at site B, the following **three phases occur**:
  i. Connection establishment
  ii. Data transfer
  iii. Connection termination

Transmission Control Protocol(TCP)

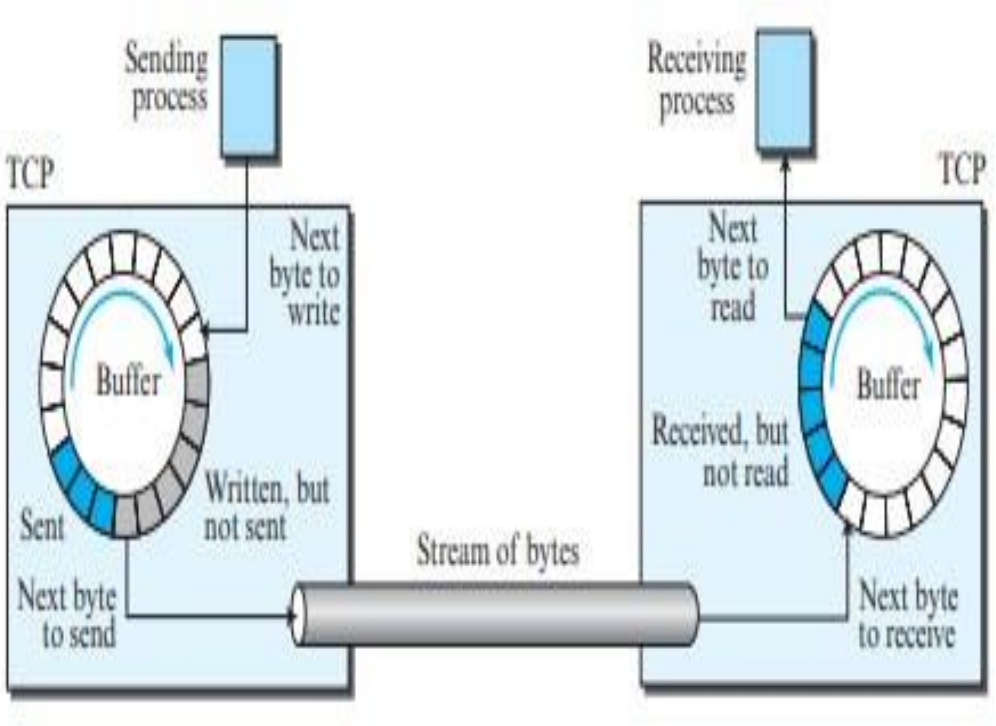
- **Reliability** – TCP is reliable as it uses checksum for error detection, attempts to recover lost or corrupted packets by re-transmission, acknowledgement policy and timers. It uses features like byte number and sequence number and acknowledgement number so as to ensure reliability. Also, it uses congestion control mechanisms.
- **Stream oriented** – TCP is a stream-oriented protocol. TCP 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.

This imaginary environment is depicted in Figure. The sending process produces (writes to) the stream and the receiving process consumes (reads from) it.



Sending and Receiving Buffers

- Because the sending and the receiving processes may not necessarily write or read data at the same rate, TCP needs buffers for storage.
- There are two buffers, the sending buffer and the receiving buffer, one for each direction.
- One way to implement a buffer is to use a circular array.



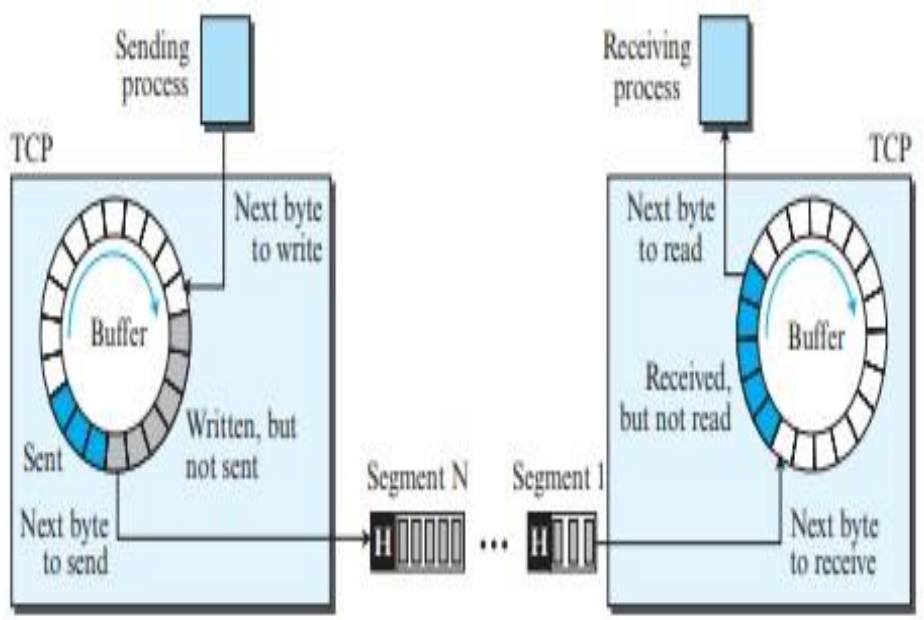
- **At the sender, the buffer has three types of chambers**.
- The white section contains empty chambers that can be filled by the sending process (producer).
- The colored area holds bytes that have been sent but not yet acknowledged. The TCP sender keeps these bytes in the buffer until it receives an acknowledgment.
- The shaded area contains bytes to be sent by the sending TCP.
- TCP may be able to send only part of this shaded section. This could be due to the slowness of the receiving process, or congestion in the network. Also, after the bytes in the colored chambers are acknowledged, the chambers are recycled and available for use by the sending process.

Transmission Control Protocol(TCP)

- The operation of the buffer at the receiver is simpler. The circular buffer is divided into two areas (shown as white and colored).
- The white area contains empty chambers to be filled by bytes received from the network.
- The colored sections contain received bytes that can be read by the receiving process. When a byte is read by the receiving process, the chamber is recycled and added to the pool of empty chambers.

Segments

The network layer, as a service provider for TCP, needs to send data in packets, not as a stream of bytes. At the transport layer, TCP groups a number of bytes together into a packet called a segment. TCP adds a header to each segment and delivers the segment to the network layer for transmission. The segments are encapsulated in an IP datagram and transmitted. This entire operation is transparent to the receiving process.



TCP segments

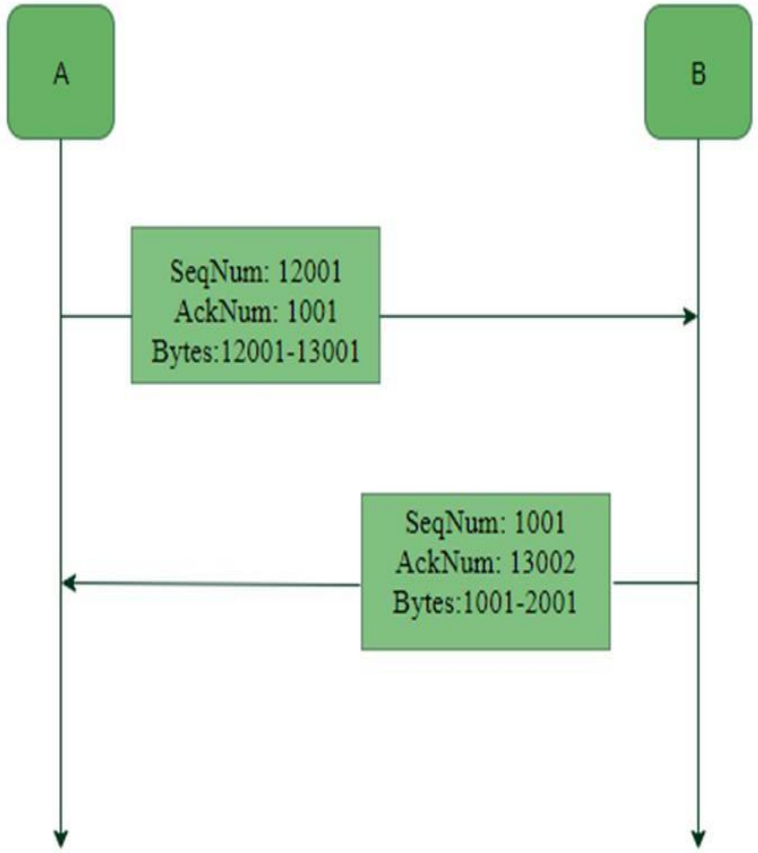
TCP Features

Sequence number

Sequence numbers are given to the segments so as to reassemble the bytes at the receiver end even if they arrive in a different order. Sequence number of a segment is the byte number of the first byte that is being sent.

Byte number

All the data bytes that are to be transmitted are numbered and the beginning of this numbering is arbitrary.



Acknowledgement number

Acknowledgement number is the next byte number that the receiver expects to receive which also provides acknowledgement for receiving the previous bytes.

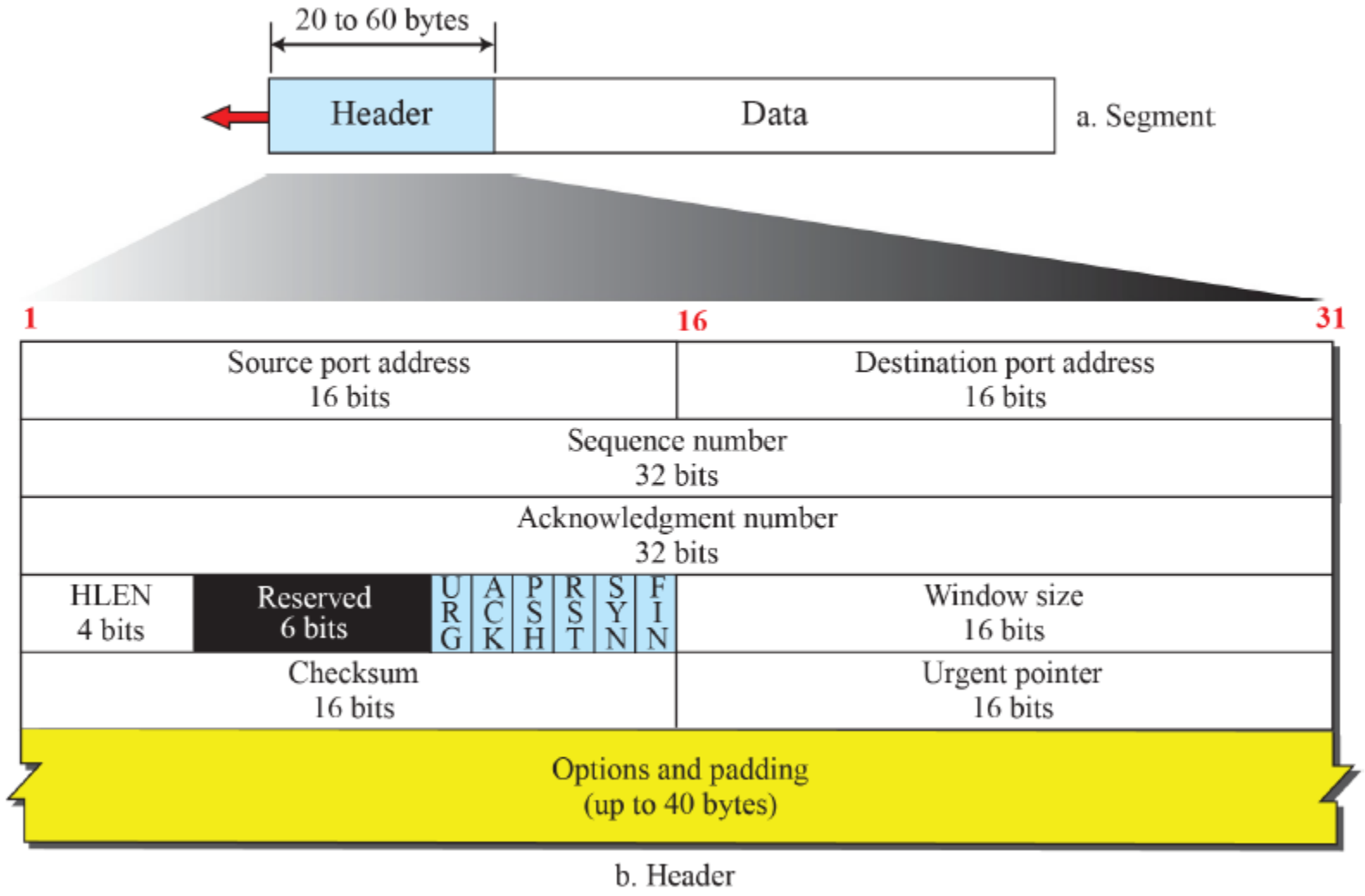
- A sends **acknowledgement number1001**, which means that it has received data bytes till byte number 1000 **and expects to receive 1001 next**, hence B next sends data bytes starting from 1001.

- Similarly, since B has received data bytes till byte number 13001 after the first data transfer from A to B, therefore B sends acknowledgement number 13002, the byte number that it expects to receive from A next.

TCP Segment

- At the transport layer, TCP **groups a number of bytes together** into a packet called a segment.
- TCP adds a header to each segment (for control purposes) and delivers the segment to the network layer for transmission.
- 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.

TCP segment format



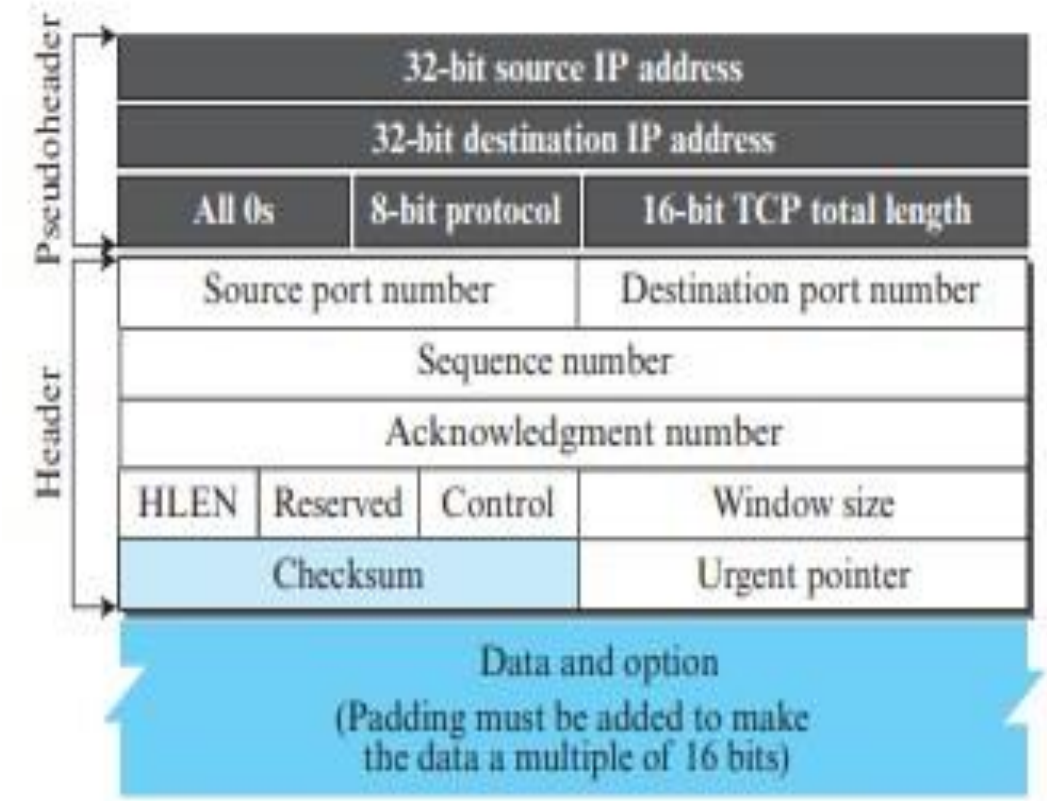
Segment

- **Source Port Address** – 16 bit field that holds the port address of the application that is sending the data segment.
- **Destination Port Address** – 16 bit field that holds the port address of the application in the host that is receiving the data segment.
- **Sequence Number** – 32 bit field that holds the sequence number, i.e, the byte number of the first byte that is sent in that particular segment. It is used to reassemble the message at the receiving end if the segments are received out of order.
- **Acknowledgement Number** – 32 bit field that holds the acknowledgement number, i.e, the byte number that the receiver expects to receive next. It is an acknowledgement for the previous bytes being received successfully.
- **Header Length (HLEN)** – This is a **4 bit field** that indicates the length of the TCP header by number of 4-byte words in the header, i.e, if the header is of 20 bytes(min length of TCP header), then this field will hold 5 (because 5 x 4 = 20) and the maximum length: 60 bytes, then it'll hold the value 15(because 15 x 4 = 60). Hence, the value of this field is always between 5 and 15.

Window size – This field tells **the window size of the sending TCP** in 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.

Transmission Control Protocol(TCP)

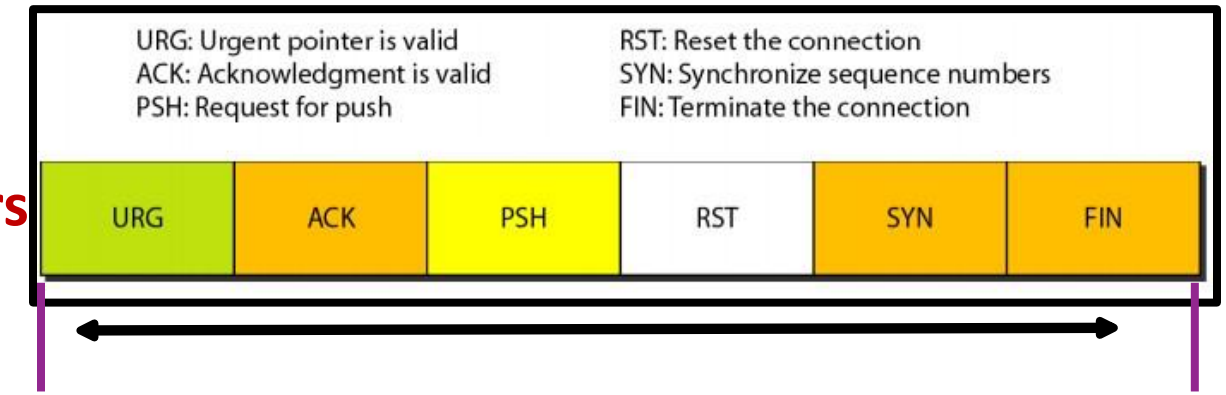
- **Checksum –** This field holds the checksum for error control. It is mandatory in TCP as opposed to UDP. . The same pseudoheader, serving the same purpose, is added to the segment. For the TCP pseudoheader, the value for the protocol field is 6.
- **Urgent pointer –** This field (valid only if the 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.



Pseudoheader added to the TCP datagram

Transmission Control Protocol(TCP)

- **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**



- **Options.** There can be up to 40 bytes of optional information in the TCP header

A TCP Connection

In TCP, connection-oriented transmission requires three phases:

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

Connection Establishment

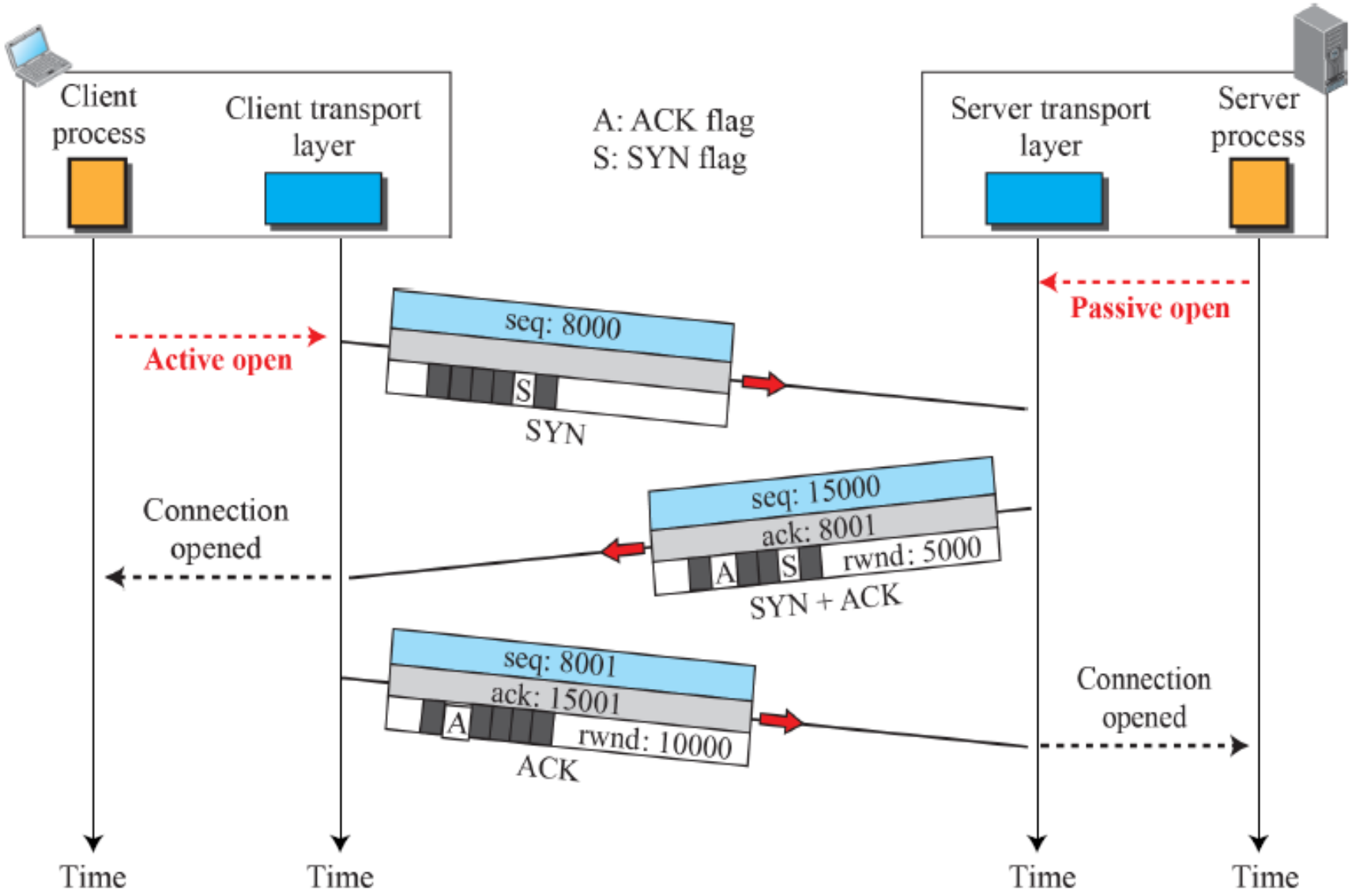
- TCP transmits data in full-duplex mode.
- When two TCPs in two machines are connected, they are able to send segments to each other simultaneously.
- This implies that each party must initialize communication and get approval from the other party before any data are transferred.

Three-Way Handshaking

- The connection establishment in TCP is called **three-way handshaking**.
- Eg:
- An application program, called the client, wants to make a connection with another application program, called the server, using TCP as the transport-layer protocol.
- The process starts with the server. The server program tells its TCP that it is ready to accept a connection. This request is called a **passive open.**
- Although the server TCP is ready to accept a connection from any machine in the world, it cannot make the connection itself. The client program issues a request for an **active open.**
- A client that wishes to connect to an open server tells its TCP to connect to a particular server.
- TCP can now start the three-way handshaking process.

Transmission Control Protocol(TCP)

Connection establishment using three-way handshaking



1

The **client** sends the first segment, a SYN segment, in which only the SYN flag is set.

This segment is for synchronization of sequence numbers.

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).**

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 segment can also include some options.

The **SYN segment** is a **control segment** and carries **no data**. However, it consumes one sequence number because it needs to be acknowledged. We can say that the SYN segment carries one imaginary byte.

2

The **Server** sends the second segment, a **SYN + ACK** segment with two flag bits set as: **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.

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 **acknowledgme**nt, it also needs to **define the receive window** size, **rwnd** (to be used by the client).Since this segment is playing the role of a SYN segment, it needs to be acknowledged. It, therefore, consumes one sequence number.

3

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.

The ACK segment does not consume any sequence numbers if it does not carry data, but some implementations allow this third segment in the connection phase to carry the first chunk of data from the client.

In this case, the segment consumes as many sequence numbers as the number of data bytes.

Transmission Control Protocol(TCP)

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

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

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

SYN Flooding Attack

- The connection establishment procedure in TCP is susceptible to a serious security problem called SYN flooding attack.
- This happens when one or more malicious attackers send a large number of SYN segments to a server pretending that each of them is coming from a different client by faking the source IP addresses in the datagrams.

- The server, assuming that the clients are issuing an active open, allocates the necessary resources, such as creating transfer control block (TCB) tables and setting timers.
- The TCP server then sends the SYN + ACK segments to the fake clients, which are lost.
- When the server waits for the third leg of the handshaking process, however, resources are allocated without being used.

Transmission Control Protocol(TCP)

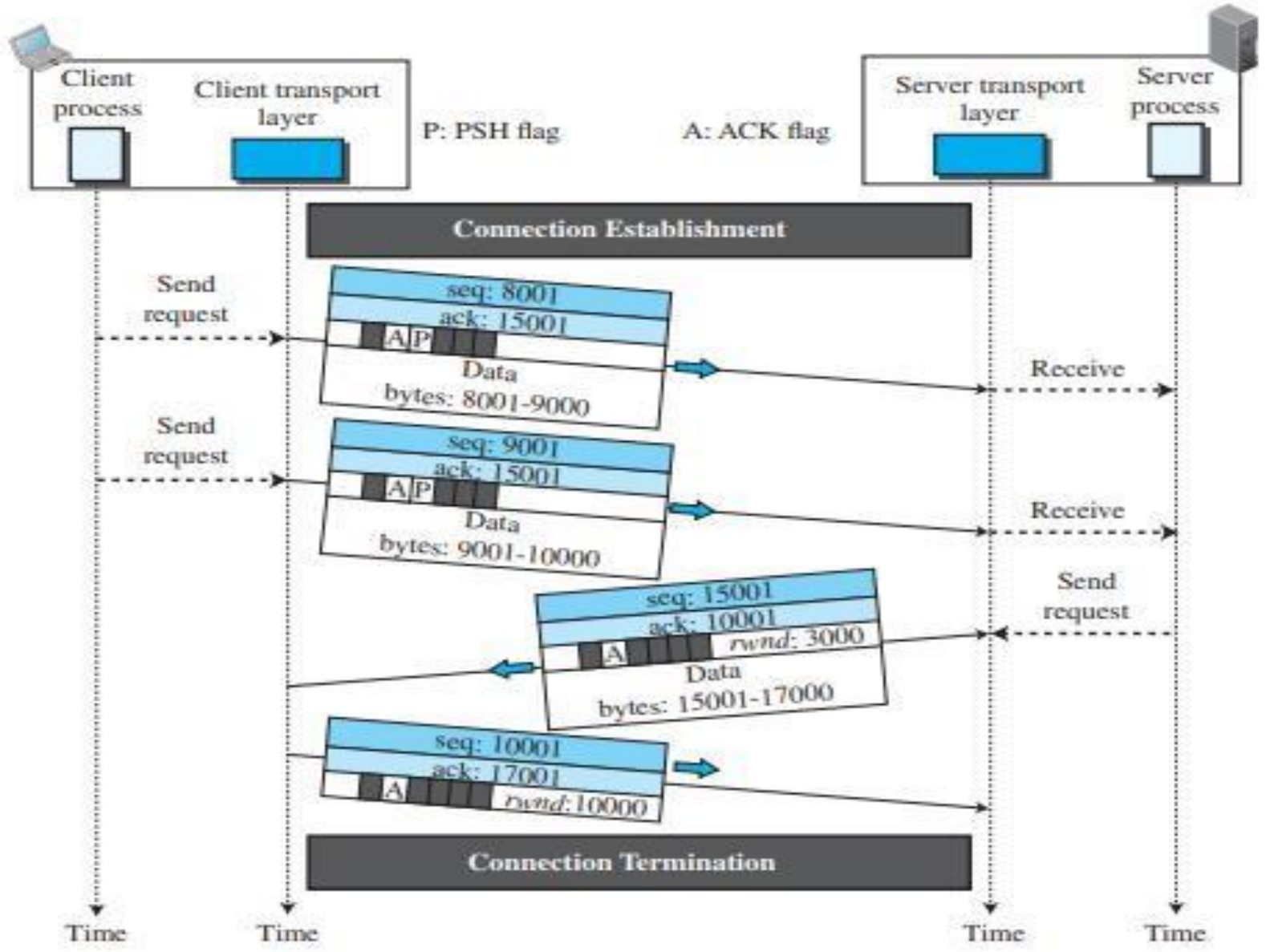
- If, during this short period of time, the number of SYN segments is large, the server eventually runs out of resources and may be unable to accept connection requests from valid clients. This SYN flooding attack belongs to a group of security attacks known as a denial of service attack, in which an attacker monopolizes a system with so many service requests that the system overloads and denies service to valid requests.

- Some implementations of TCP have strategies to alleviate the effect of a SYN attack. Some have imposed a limit of connection requests during a specified period of time. Others try to filter out datagrams coming from unwanted source addresses. One recent strategy is to postpone resource allocation until the server can verify that the connection request is coming from a valid IP address, by using what is called a cookie.
- SCTP, the new transport-layer protocol that uses this strategy

Data Transfer

- After connection is established, bidirectional data transfer can take place.
- The client and server can send data and acknowledgments in both directions.
- The acknowledgment is piggybacked with the data.
- For example, after a connection is established, the client sends 2,000 bytes of data in two segments.
- The server then sends 2,000 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 is no more data to be sent. Note the values of the sequence and acknowledgment numbers.
- The data segments 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.
- The segment from the server, on the other hand, does not set the push flag. Most TCP implementations have the option to set or not to set this flag.

Data transfer



Pushing Data

- The sending TCP uses a buffer to store the stream of data coming from the sending application program. The sending TCP can select the segment size.
- The receiving TCP also buffers the data when they arrive and delivers them to the application program when the application program is ready or when it is convenient for the receiving TCP. This type of flexibility increases the efficiency of TCP.
- However, there are occasions in which the application program has no need for this flexibility.
- *Eg: consider an application program that communicates interactively with another application program on the other end. The application program on one site wants to send a chunk of data to the application at the other site and receive an immediate response. Delayed transmission and delayed delivery of data may not be acceptable by the application program. TCP can handle such a situation.*

- The application program at the sender can request a push operation. This means that the sending TCP must not wait for the window to be filled. It must create a segment and send it immediately.

- The sending TCP must also set the push bit (PSH) to let the receiving TCP know that the segment includes data that must be delivered to the receiving application program as soon as possible and not to wait for more data to come. This means to change the byte oriented TCP to a chunk-oriented TCP, but TCP can choose whether or not to use this feature.

Urgent Data

- TCP is a stream-oriented protocol. This means that the data is presented from the application program to TCP as a stream of bytes. Each byte of data has a position in the stream.
- However, there are occasions in which an application program needs to send urgent bytes, some bytes that need to be treated in a special way by the application at the other end. The solution is to send a segment with the **URG bit set.**
- The sending application program tells the sending TCP that the **piece of data is urgent.** The sending TCP creates a segment and inserts the urgent data at the beginning of the segment. The rest of the segment can contain normal data from the buffer. The **urgent pointer field** in the **header** defines the end of the urgent data (the last byte of urgent data).

- For example, if the segment sequence number is 15000 and the value of the urgent pointer is 200, the first byte of urgent data is the byte 15000 and the last byte is the byte 15200.
- The rest of the bytes in the segment (if present) are non-urgent. It is important to mention that TCP's urgent data is neither a priority service nor an out-of-band data service as some people think.
- Rather, TCP urgent mode is a service by which the application program at the sender side marks some portion of the byte stream as needing special treatment by the application program at the receiver side.
- The receiving TCP delivers bytes (urgent or non-urgent) to the application program in order, but inform the application program about the beginning and end of urgent data. It is left to the application program to decide what to do with the urgent data

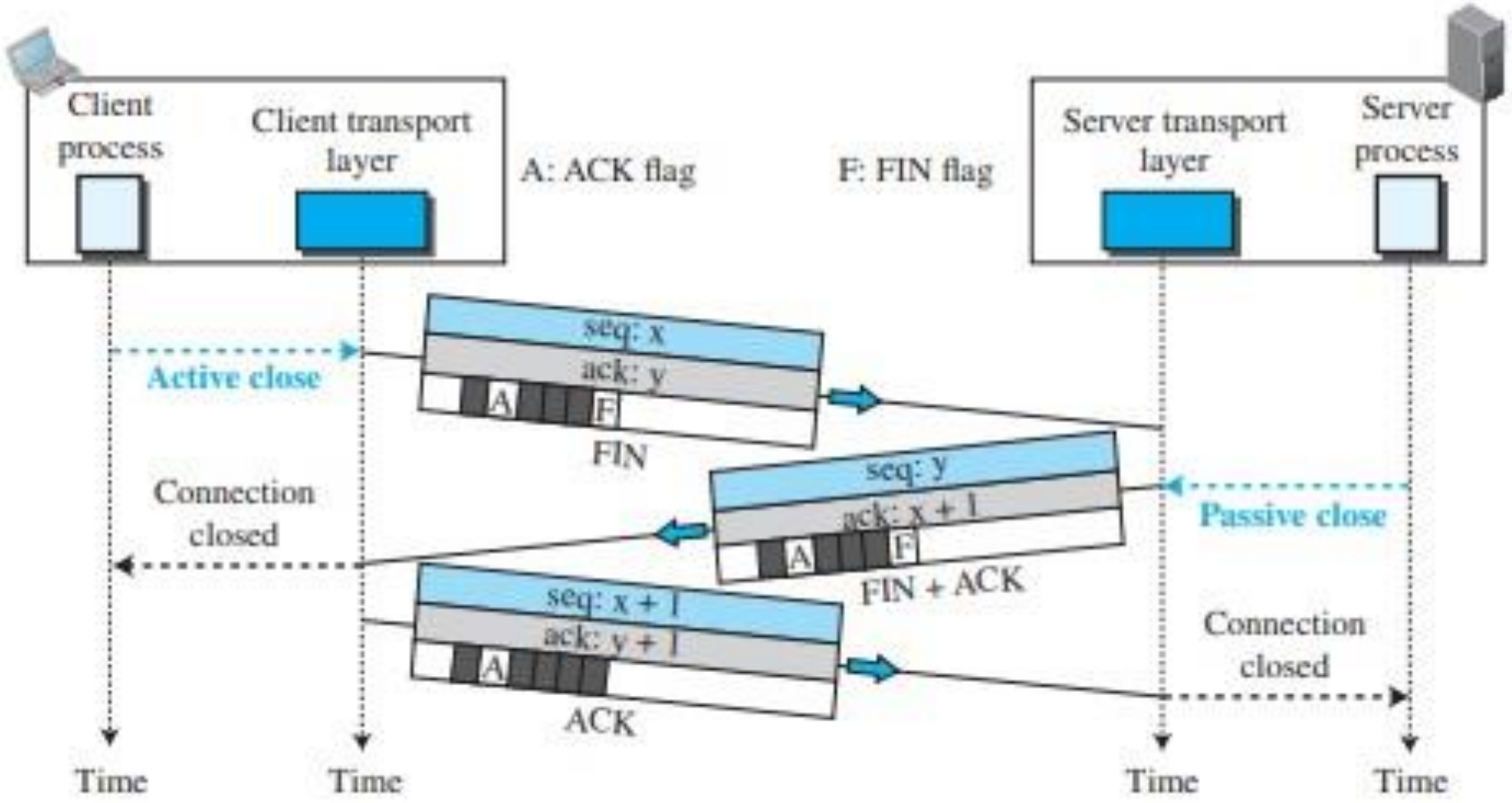
Connection Termination

- Either of the two parties involved in exchanging data (client or server) can close the connection, although it is usually initiated by the client.
- Most implementations today allow two options for connection termination:
  i. Three-way handshaking
  ii. Four-way handshaking with a half-close option.

Three-Way Handshaking

Most implementations today allow three-way handshaking for connection termination

Connection termination using three-way handshaking



Connection Termination

1

In this 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.

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 because it needs to be ackno.

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 because it needs to be acknowledged.

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 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.

Connection Termination

The FIN segment consumes one sequence number if it does not carry data.

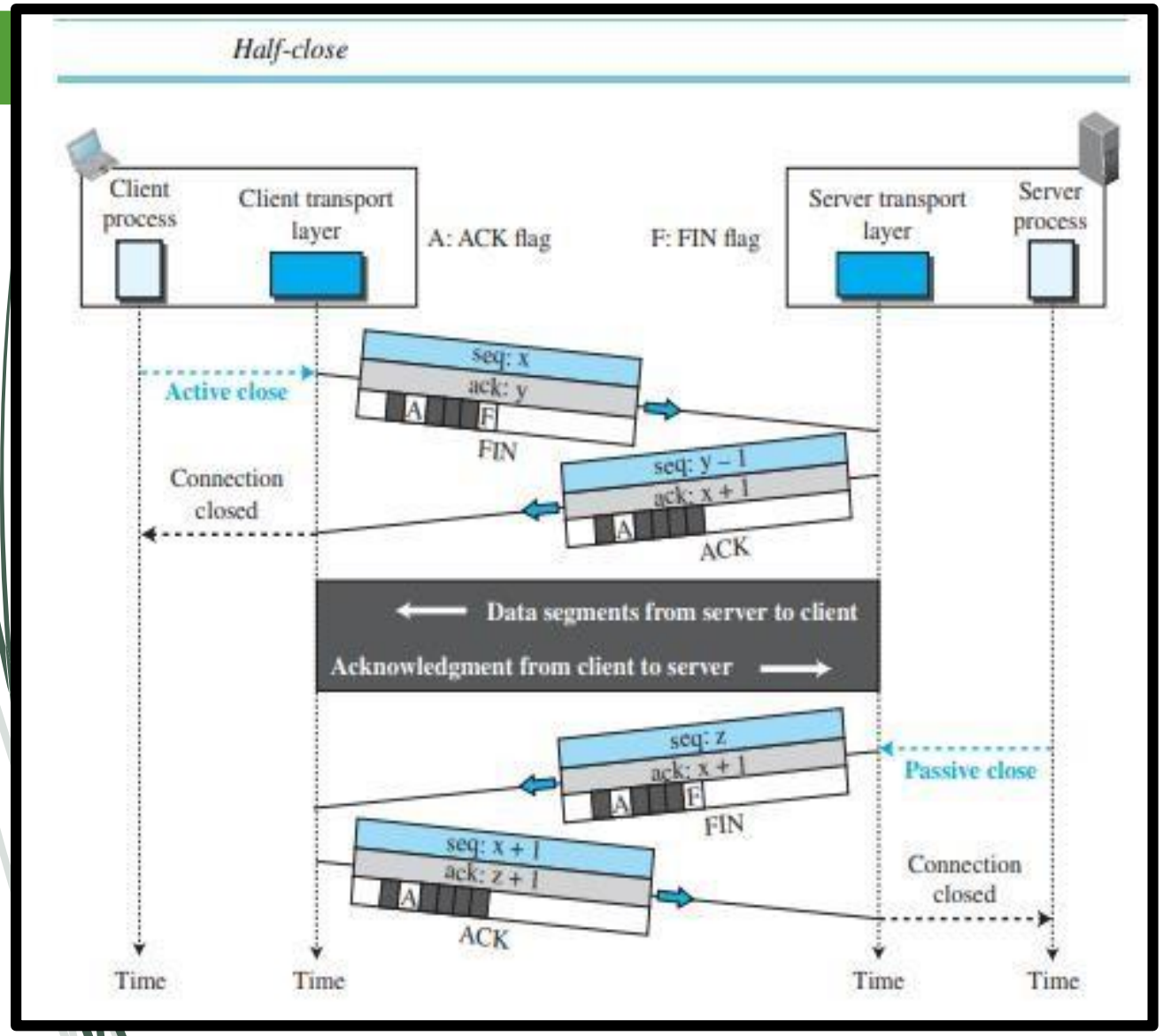
The FIN + ACK segment consumes one sequence number if it does not carry data.

Half-Close

- In TCP, one end can stop sending data while still receiving data. This is called a half close.
- Either the server or the client can issue a half-close request.
- It can occur when the server needs all the data before processing can begin.

- **Eg**: A good example is Sorting. When the client sends data to the server to be sorted, the server needs to receive all the data before sorting can start. This means the client, after sending all data, can close the connection in the client-to-server direction. However, the server-to-client direction must remain open to return the sorted data.

- The server, after receiving the data, still needs time for sorting; its outbound direction must remain open. Figure shows an example of a half-close.



- The data transfer from the client to the server stops. The client half-closes the connection by sending a FIN segment.
- The server accepts the half-close by sending the ACK segment. The server, however, can still send data. When the server has sent all of the processed data, it sends a FIN segment, which is acknowledged by an ACK from the client.
- After half closing the connection, data can travel from the server to the client and acknowledgments can travel from the client to the server. The client cannot send any more data to the server.

Connection Reset

- TCP at one end may deny a connection request, may abort an existing connection, or may terminate an idle connection.
- All of these are done with the **RST (reset) flag.**

TCP Congestion Control

TCP uses different policies to handle the congestion in the network.

- Congestion Window
- The size of the send window is controlled by the receiver using the value of rwnd, which is advertised in each segment traveling in the opposite direction.
- The use of this strategy guarantees that the receive window is never overflowed with the received bytes (no end congestion).
- This, however, does not mean that the intermediate buffers, buffers in the routers, do not become congested. A router may receive data from more than one sender. No matter how large the buffers of a router may be, it may be overwhelmed with data, which results in dropping some segments sent by a specific TCP sender.
- In other words, there is no congestion at the other end, but there may be congestion in the middle.
- TCP needs to worry about congestion in the middle(network) because many segments lost may seriously affect the error control. More segment loss means resending the same segments again, resulting in worsening the congestion, and finally the collapse of the communication.

TCP Congestion Control

- TCP is an end-to-end protocol that uses the service of IP.
- The congestion in the router is in the IP territory and should be taken care of by IP.
- However, IP is a simple protocol with no congestion control.
- TCP, itself, needs to be responsible for this problem.

TCP Congestion Control

- To control the number of segments to transmit, TCP uses another variable called a congestion window, cwnd, whose size is controlled by the congestion situation in the network.
- The cwnd variable and the rwnd variable together define the size of the send window in TCP.
- The first is related to the congestion in the middle (network);
- the second is related to the congestion at the end.
- The actual size of the window is the minimum of these two.

Actual window size = minimum (rwnd, cwnd);

Congestion Detection

- It is necessary to know how a TCP sender can detect the possible existence of congestion in the network. The TCP sender uses the occurrence of two events as signs of congestion in the network: time-out and receiving three duplicate ACKs.
- The first is the time-out: If a TCP sender does not receive an ACK for a segment or a group of segments before the time-out occurs, it assumes that the corresponding segment or segments are lost and the loss is due to congestion.
- Another event is the receiving of three duplicate ACKs (four ACKs with the same acknowledgment number):
  - when a TCP receiver sends a duplicate ACK, it is the sign that a segment has been delayed, but sending three duplicate ACKs is the sign of a missing segment, which can be due to congestion in the network. However, the congestion in the case of three duplicate ACKs can be less severe than in the case of time-out.
  - When a receiver sends three duplicate ACKs, it means that one segment is missing, but three segments have been received. The network is either slightly congested or has recovered from the congestion.

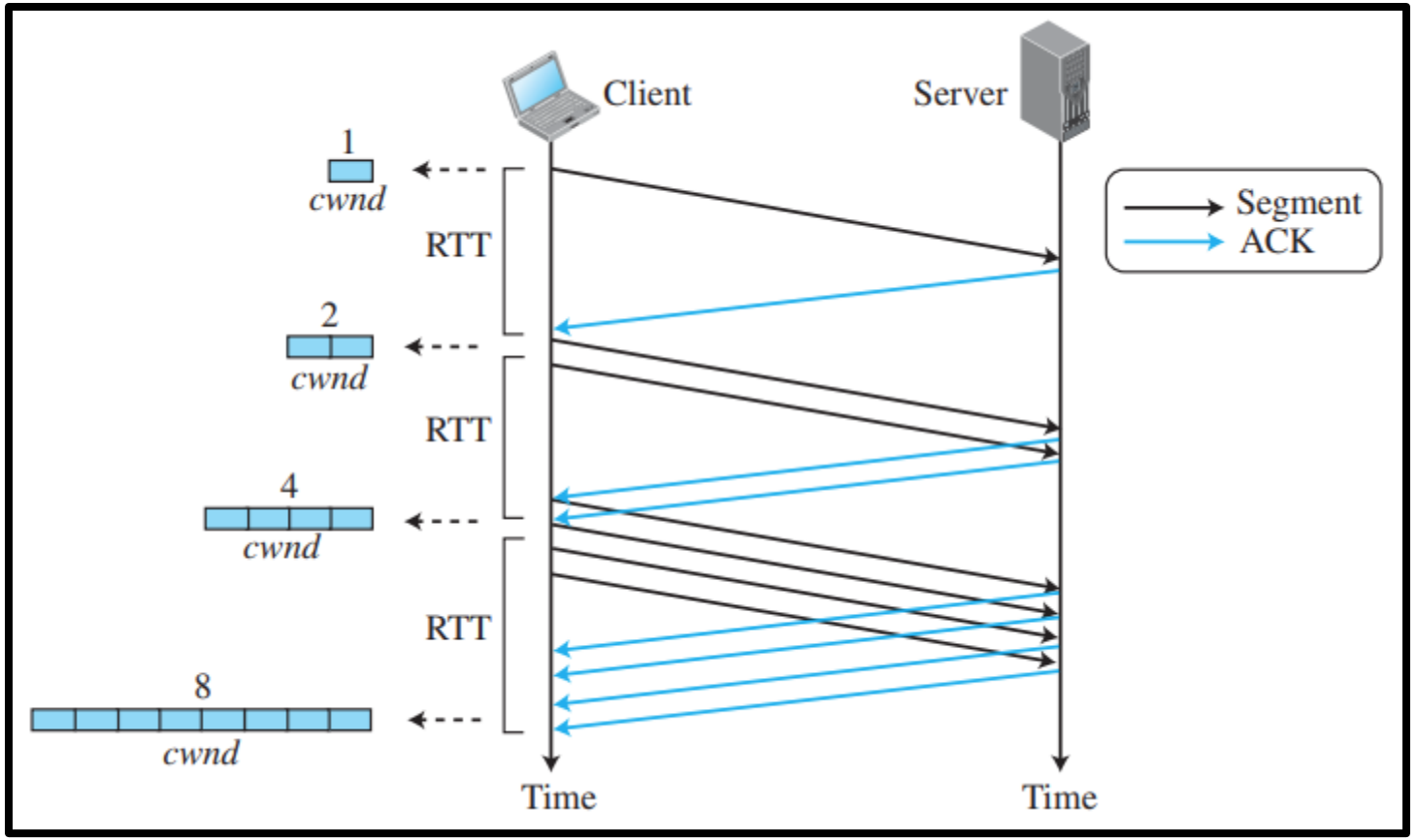
Congestion Policies

- TCP's general policy for handling congestion is based on three algorithms:
  i. Slow start
  ii. Congestion avoidance
  iii. Fast recovery.

Slow Start: Exponential Increase

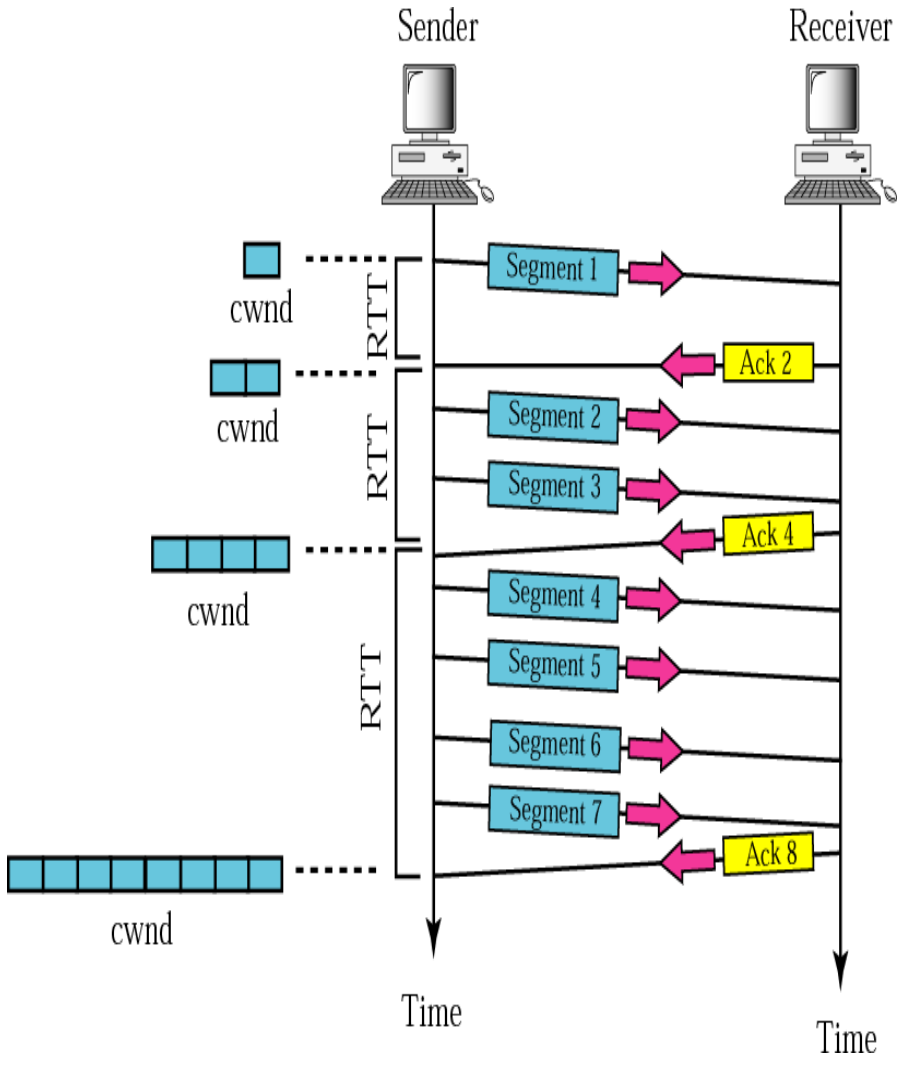
- The slow-start algorithm is based on the idea that the size of the congestion window (cwnd) starts with one maximum segment size (MSS), but it increases one MSS each time one acknowledgment arrives.
- In the slow start phase, the sender starts with a slow rate of transmission, but increases the rate rapidly to reach a threshold.
- When the threshold is reached, the rate of increase is reduced.
- Finally if ever congestion is detected, the sender goes back to the slow start or congestion avoidance phase, based on how the congestion is detected.

Slow start, Exponential increase



Congestion Policies

- The sender starts with cwnd = 1, means that the sender can send only one segment. After the first ACK arrives, the acknowledged segment is purged from the window, which means there is now one empty segment slot in the window.
- The size of the congestion window is also increased by 1 because the arrival of the acknowledgment is a good sign that there is no congestion in the network. The size of the window is now 2.
- After sending two segments and receiving two individual acknowledgments for them, the size of the congestion window now becomes 4, and so on.
- The size of the congestion window in this algorithm is a function of the number of ACKs arrived and can be determined as follows.
- If an ACK arrives, cwnd = cwnd + 1.



Congestion Policies

- If we look at the size of the cwnd in terms of round-trip times (RTTs), we find that the growth rate is exponential in terms of each round trip time, which is a very aggressive approach:

Start	$\rightarrow$	$cwnd = 1 \rightarrow 2^0$
After 1 RTT	$\rightarrow$	$cwnd = cwnd + 1 = 1 + 1 = 2 \rightarrow 2^1$
After 2 RTT	$\rightarrow$	$cwnd = cwnd + 2 = 2 + 2 = 4 \rightarrow 2^2$
After 3 RTT	$\rightarrow$	$cwnd = cwnd + 4 = 4 + 4 = 8 \rightarrow 2^3$

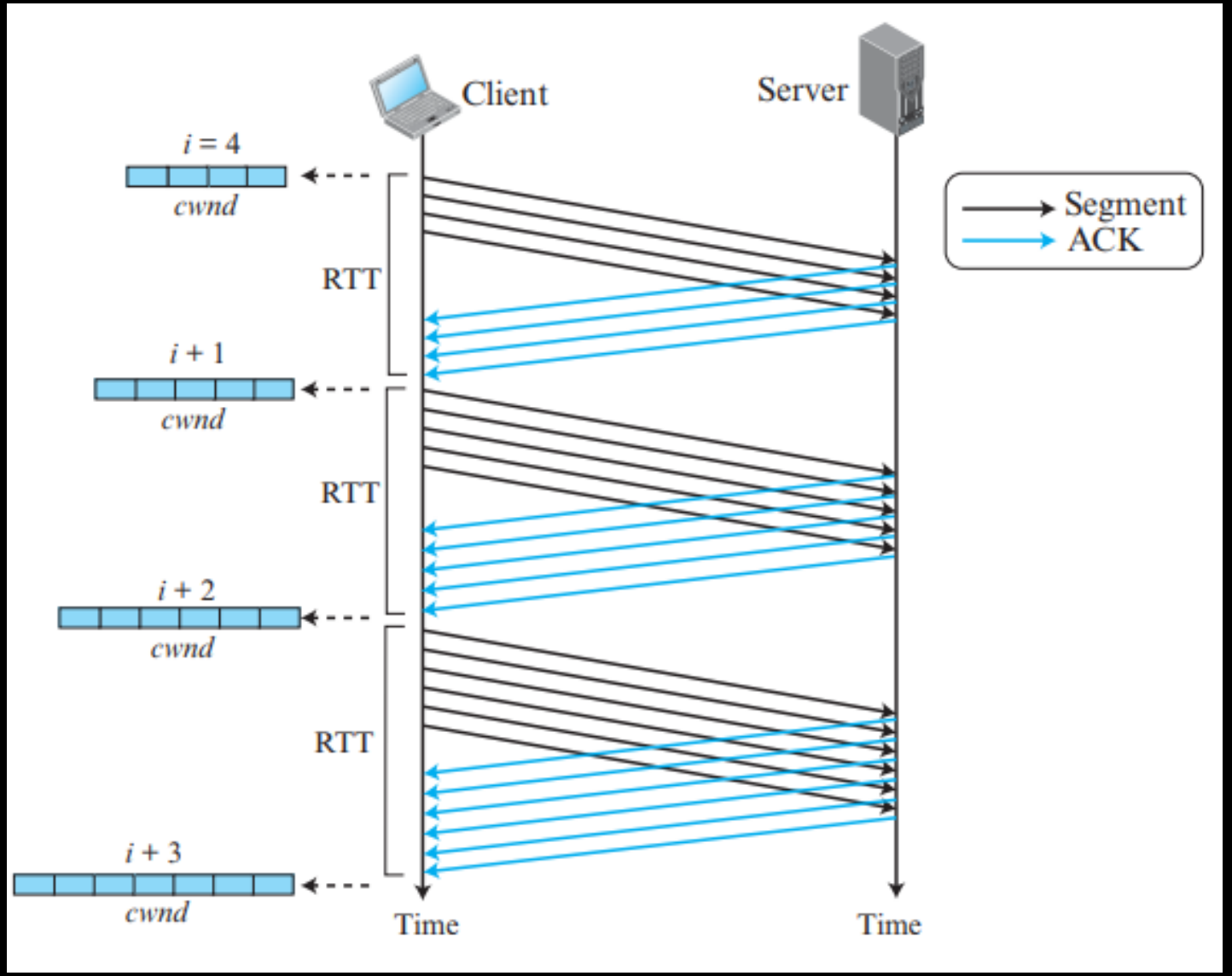
- A slow start cannot continue indefinitely. There must be a threshold to stop this phase.
- The sender keeps track of a variable named ssthresh (slow-start threshold).
- When the size of the window in bytes reaches this threshold, slow start stops and the next phase starts.

In the slow-start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.

Congestion Avoidance: Additive Increase

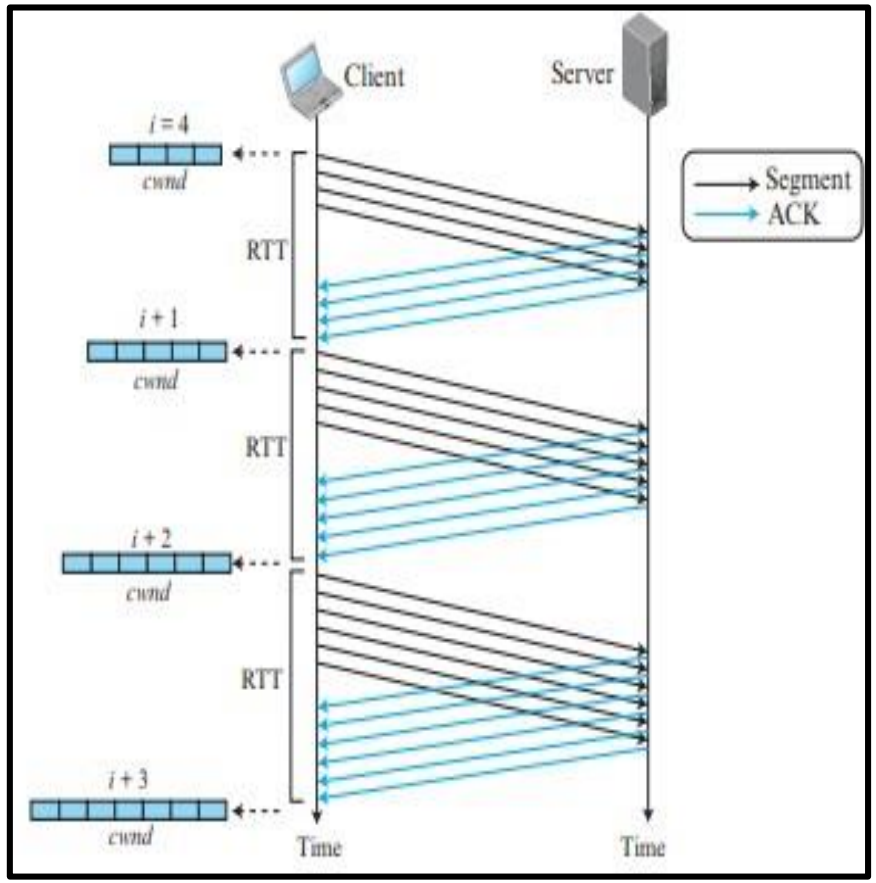
- If we continue with the slow-start algorithm, the size of the congestion window increases exponentially.
- To avoid congestion before it happens, one must slow down this exponential growth.
- TCP defines another algorithm called **congestion avoidance**, which increases the cwnd additively instead of exponentially.
- When the size of the congestion window reaches the slow-start threshold in the case where cwnd = i, the slow-start phase stops and the additive phase begins.
- In this algorithm, **each time the whole "window" of segments is acknowledged**, the size of the **congestion window is increased by one.**
- A window → number of segments transmitted during **RTT.**

Congestion Avoidance: Additive Increase



Congestion Policies

- The sender starts with cwnd = 4. This means that the sender can send only four segments. After four ACKs arrive, the acknowledged segments are purged from the window, which means there is now one empty segment slot in the window.
- The size of the congestion window is also increased by The size of window is now 5. After sending five segments and receiving five acknowledgments for them, the size of the congestion window now becomes 6. And so on.
- In other words, the **size of the congestion window** in this algorithm is also a **function of the number of ACKs** that have arrived and can be determined as follows:
- **If an ACK arrives, cwnd = cwnd + (1/cwnd).**



Congestion avoidance, additive increase

Congestion Policies

- In other words, the size of the window increases only 1/cwnd portion of MSS (in bytes).
- All segments in the previous window should be acknowledged to increase the window 1 MSS bytes.
- If we look at the size of the cwnd in terms of round-trip times (RTTs), we find that the growth rate is linear in terms of each round-trip time, which is much more conservative than the slow start approach.

Start	$\rightarrow$	$cwnd = i$
After 1 RTT	$\rightarrow$	$cwnd = i + 1$
After 2 RTT	$\rightarrow$	$cwnd = i + 2$
After 3 RTT	$\rightarrow$	$cwnd = i + 3$

In the congestion-avoidance algorithm, the size of the congestion window increases additively until congestion is detected.

Fast Recovery

- The fast-recovery algorithm is optional in TCP.
- The old version of TCP did not use it, but the new versions try to use it.
- It starts when three duplicate ACKs arrives that is interpreted as light congestion in the network.
- Like congestion avoidance, this algorithm is also an additive increase, but it increases the size of the congestion window when a duplicate ACK arrives (after the three duplicate ACKs that trigger the use of this algorithm).

If a duplicate ACK arrives, cwnd = cwnd + (1 / cwnd);

Fast Recovery

- Overall, Fast Recovery helps TCP to quickly recover from packet loss events by avoiding unnecessary timeouts and quickly retransmitting lost packets while still maintaining a conservative approach to congestion control.

Policy Transition

- We need to know about the three versions of TCP to determine the situations when TCP moves from one policy to another.
- Those three versions of TCP:
  1. **Taho TCP (early TCP)**
     - used only two different algorithms in their congestion policy: slow start and congestion avoidance.
  2. **Reno TCP: (**newer version of TCP**)**
     - added a new state to the congestion control FSM, called the fast-recovery state.
  3. **New Reno TCP**:

- **New Reno TCP**:
- A later version of TCP, called NewReno TCP, made an extra optimization on the Reno TCP. In this version, TCP checks to see if more than one segment is lost in the current window when three duplicate ACKs arrive.
- When TCP receives three duplicate ACKs, it **retransmits the lost segment** until a **new ACK** (**not duplicate**) arrives.
- If the new ACK defines the end of the window when the congestion was detected, TCP is certain that only one segment was lost.

Additive Increase, Multiplicative Decrease(AIMD)

- Out of the three versions of TCP, the Reno version is most common. In this version, most of the time the congestion is detected and taken care of by observing the three duplicate ACKs.
- Even if there are some time-out events, TCP recovers from them by aggressive exponential growth.
- In other words, in a long TCP connection, if we ignore the slow-start states and short exponential growth during fast recovery, the TCP congestion window is ;
- **cwnd = cwnd + (1/cwnd)** when an ACK arrives (congestion avoidance), and
- cwnd = cwnd / 2 when congestion is detected, as though SS does not exists and the length of FR is reduced to zero.
- The first is called additive increase; the second is called multiplicative decrease. This means that the congestion window size, after it passes the initial slow start state, follows a saw tooth pattern called **additive increase, multiplicative decrease (AIMD).**

TCP Throughput

Throughput = (0.75) Wmax / RTT;

Wmax → average of window sizes when the congestion occurs.