Unit III

Session layer and Transport layer

Session layer design issues,

Session Layer protocol - Remote Procedure Call (RPC),

Transport layer services,

Transport Layer Protocols: Simple Protocol, Stop-and-Wait Protocol, Go- Back-N Protocol (GBN), Selective- Repeat Protocol, Bidirectional Protocols: Piggybacking,

Internet Transport-Layer Protocols,

User Datagram Protocol: User Datagram, UDP Services, UDP Applications,

Transmission Control Protocol: TCP Services, TCP Features, Segment, A TCP Connection, State Transition Diagram, Windows in TCP, Flow Control, Error Control, TCP Congestion Control, TCP Timers, Options.

Session Layer

- The session layer is Layer 5 from the bottom in the OSI model.
- The job of the session layer is to control and maintain connections between systems to share data.
- It establishes, maintains, and ends sessions across all channels.
- In case of a network error, it checks the authenticity and provides recovery options for active sessions.
- It manages sessions and synchronizes data flow.
- Basically, this layer regulates when computers can send data and how much data they can send.
- Essentially it coordinates communication between devices.

Functions of session layer:

1. **Dialog Control** –
   Session layer allows two systems to enter into a dialog exchange mechanism which can either be full or half-duplex.
2. **Managing Tokens** –
   The communicating systems in a network try to perform some critical operations and it is Session Layer which prevents collisions which might occur while performing these operations which would otherwise result in a loss.
3. **Synchronization** –
   Checkpoints are the midway marks that are added after a particular interval during stream of data transfer. These points are also referred to as synchronization points. The Session layer permits process to add these checkpoints.
   For example, suppose a file of 400 pages is being sent over a network, then it is highly beneficial to set up a checkpoint after every 50 pages so that next 50 pages are sent only when previous pages are received and acknowledged.

Design Issues with Session Layer :

1. **Establish sessions between machines** – The establishment of session between machines is an important service provided by session layer. This session is responsible for creating a dialog between connected machines. The Session Layer provides mechanism for opening, closing and managing a session between end-user application processes, i.e. a semi-permanent dialogue. This session consists of requests and responses that occur between applications.
2. **Enhanced Services** – Certain services such as checkpoints and management of tokens are the key features of session layer and thus it becomes necessary to keep enhancing these features during the layer's design.
3. **To help in Token management and Synchronization** – The session layer plays an important role in preventing collision of several critical operation as well as ensuring better data transfer over network by establishing synchronization points at specific intervals. Thus it becomes highly important to ensure proper execution of these services.

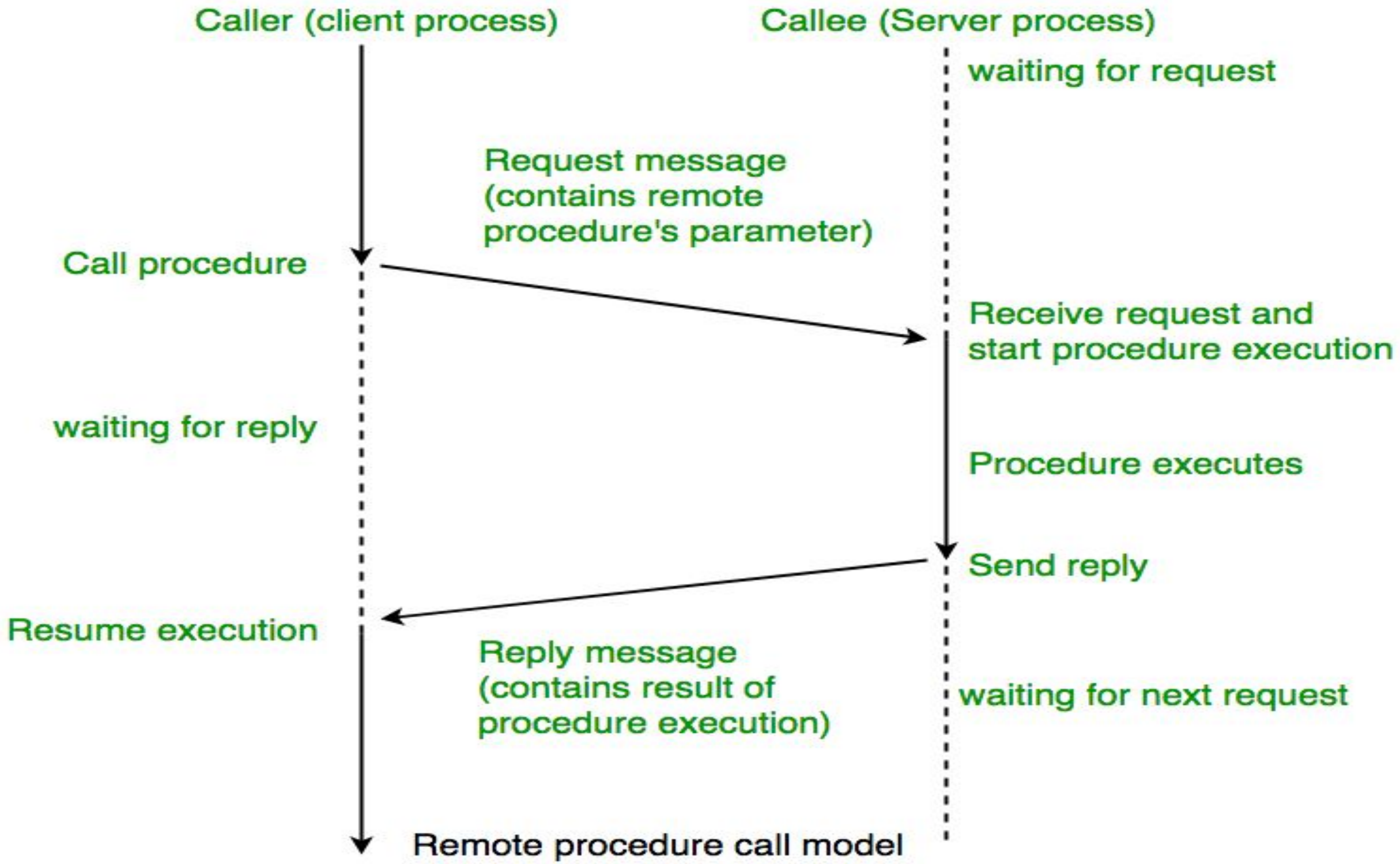
Remote Procedure Call (RPC)

- **Remote Procedure Call (RPC)** is a powerful technique for constructing **distributed, client-server based applications**.
- It is based on extending the conventional local procedure calling so that the **called procedure need not exist in the same address space as the calling procedure**.
- The two processes may be on the same system, or they may be on different systems with a network connecting them.

When making a Remote Procedure Call:

1. The calling environment is suspended, procedure parameters are transferred across the network to the environment where the procedure is to execute, and the procedure is executed there.

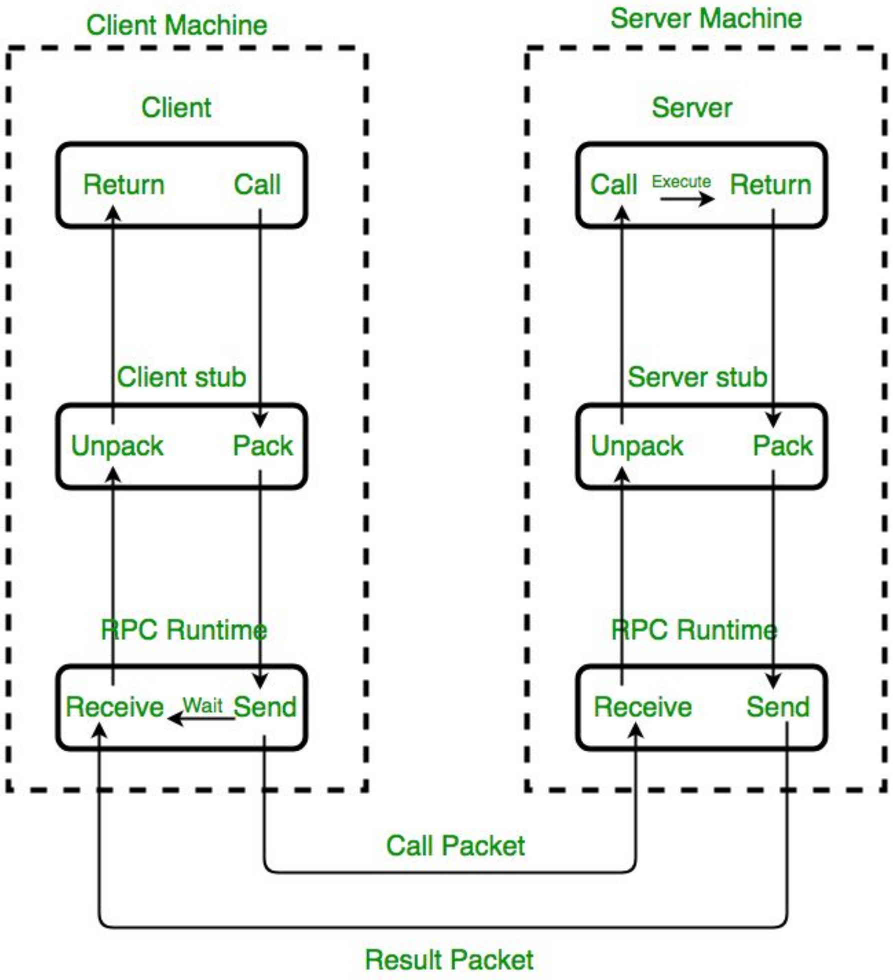
2. When the procedure finishes and produces its results, its results are transferred back to the calling environment, where execution resumes as if returning from a regular procedure call.



Remote procedure call model

The following steps take place during a RPC :

1. A client invokes a **client stub procedure**, passing parameters in the usual way. The client stub resides within the client's own address space.
2. The client stub **marshalls(pack)** the parameters into a message. Marshalling includes converting the representation of the parameters into a standard format, and copying each parameter into the message.
3. The client stub passes the message to the transport layer, which sends it to the remote server machine.
4. On the server, the transport layer passes the message to a server stub, which **demarshalls(unpack)** the parameters and calls the desired server routine using the regular procedure call mechanism.
5. When the server procedure completes, it returns to the server stub **(e.g., via a normal procedure call return)**, which marshalls the return values into a message. The server stub then hands the message to the transport layer.
6. The transport layer sends the result message back to the client transport layer, which hands the message back to the client stub.
7. The client stub demarshalls the return parameters and execution returns to the caller.



Implementation of RPC mechanism

Transport layer

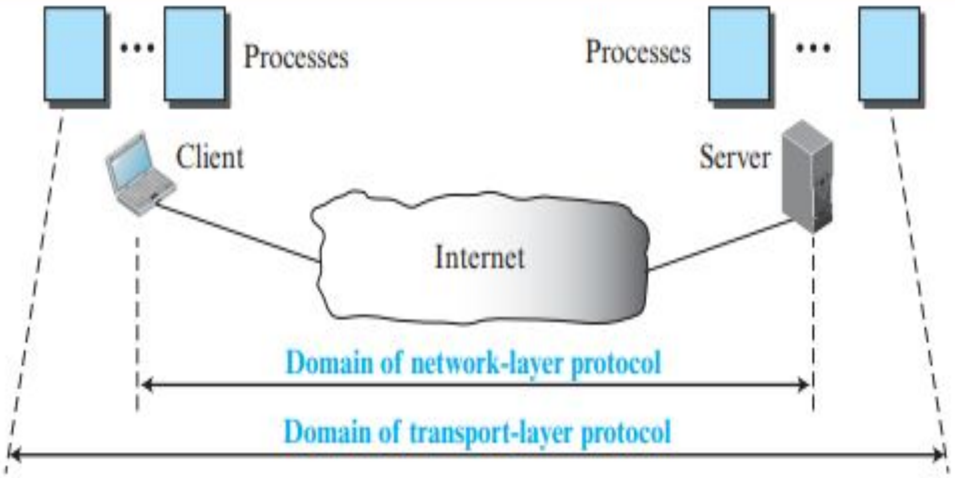
Transport layer services

- Process-to-Process Communication
- Encapsulation and Decapsulation
- Multiplexing and Demultiplexing
- Flow Control
- Error Control
- Congestion Control
- Connectionless and Connection-Oriented Services

Transport layer services

- **Process-to-Process Communication**
  - ❖ The first duty of a transport-layer protocol is to provide 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.
  - ❖ The message still needs to be handed to the correct process. This is where a transport-layer protocol takes over.
  - ❖ A transport-layer protocol is responsible for delivery of the message to the appropriate process.
  - ❖ Transport Layer requires a Port number to correctly deliver the segments of data to the correct process amongst the multiple processes running on a particular host.
  - ❖ A port number is a 16-bit address used to identify any client-server program uniquely.

Figure 3.2 *Network layer versus transport layer*


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

Addressing: Port Numbers

Transport layer services

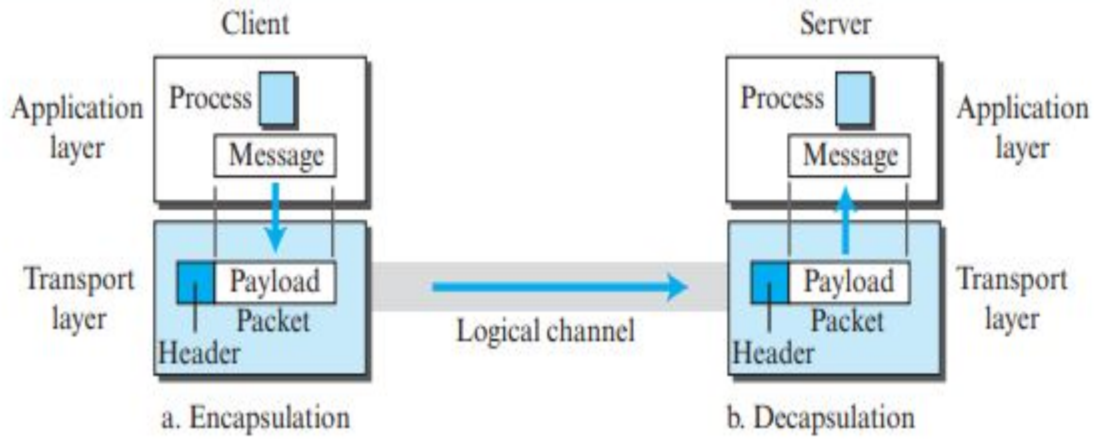
- **Encapsulation and Decapsulation**
  - ❖ To send a message from one process to another, the transport-layer protocol encapsulates and decapsulates messages.

  - ❖ **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 services

Figure 3.7 *Encapsulation and decapsulation*

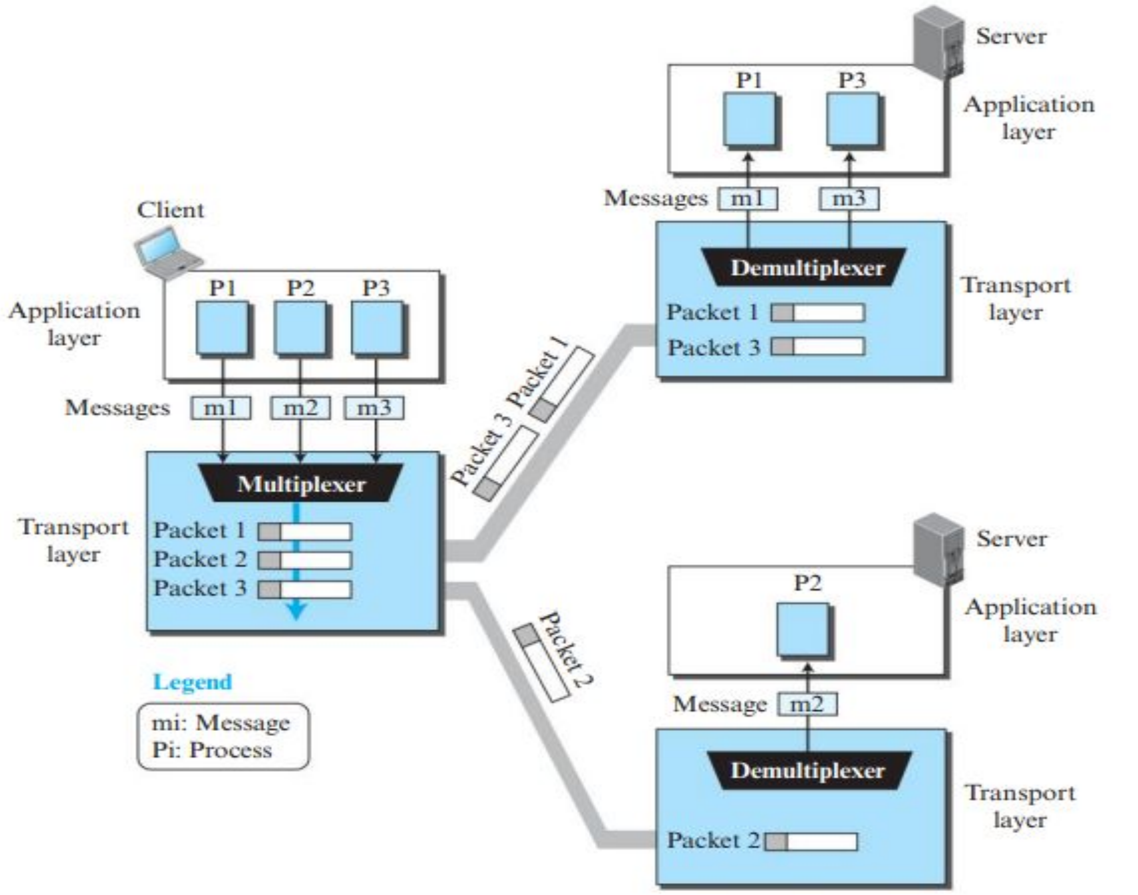


Transport layer services

- **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.

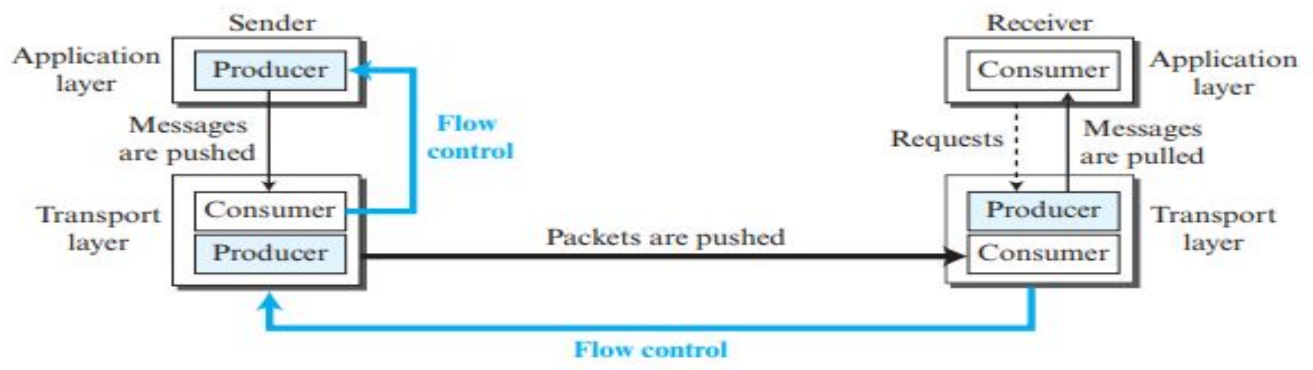
Figure 3.8 *Multiplexing and demultiplexing*



Transport layer services

- **Flow Control**
  - ❖ The transport layer provides a flow control mechanism between the adjacent layers of the TCP/IP model.
  - ❖ It 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.

Figure 3.10 *Flow control at the transport layer*



Transport layer services

- **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.

Transport layer services

- **Congestion Control**
  - ❖ An important issue in a packet-switched network, such as the Internet, is congestion.
  - ❖ 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.

- **Connectionless and Connection-Oriented Services**

Transport Layer Protocols:

Simple Protocol,

Stop-and-Wait Protocol,

Go- Back-N Protocol (GBN),

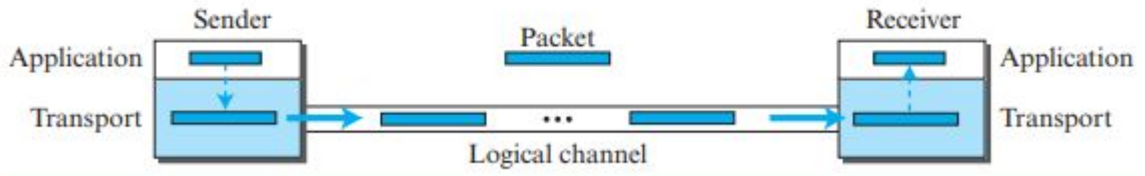
Selective- Repeat Protocol,

Bidirectional Protocols: Piggybacking, Internet Transport-Layer Protocols,

Simple Protocol

- Our first protocol is a simple connectionless protocol with neither flow nor error control.
- We assume that the receiver can immediately handle any packet it receives.
- 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.

Figure 3.17 *Simple protocol*

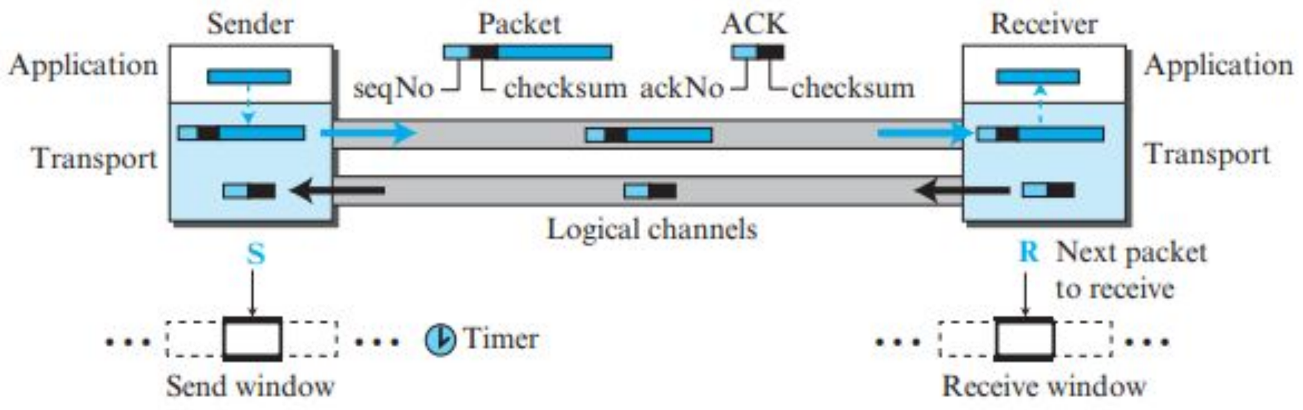


Stop-and-Wait Protocol

- Stop-and-Wait protocol is a connection-oriented protocol, which 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).
- 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.
- Only one packet and one acknowledgment can be in the channels at any time.

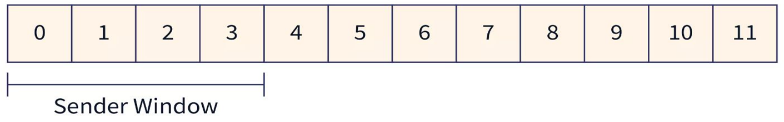
Stop-and-Wait Protocol

Figure 3.20 *Stop-and-Wait protocol*



Go- Back-N Protocol (GBN)

- Go Back N ARQ is a sliding window protocol which is used for flow control purposes. Multiple frames present in a single window are sent together from sender to receiver.
- Pipelining is allowed in the Go Back N ARQ protocol. Pipelining means sending a frame before receiving the acknowledgment for the previously sent frame.
- The sender window is a fixed-sized window that defines the number of frames that are transmitted from sender to receiver at once. The integer 'N' in the Go Back 'N' is the frame size.
- For example in Go Back 4 ARQ, the size of the sender window is 4.
- The Receiver window in the Go Back N ARQ protocol is always of size 1. This means that the receiver takes at most 1 frame at a single time.

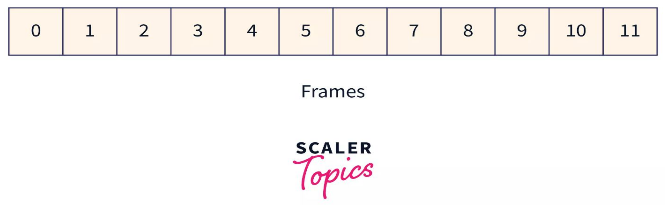


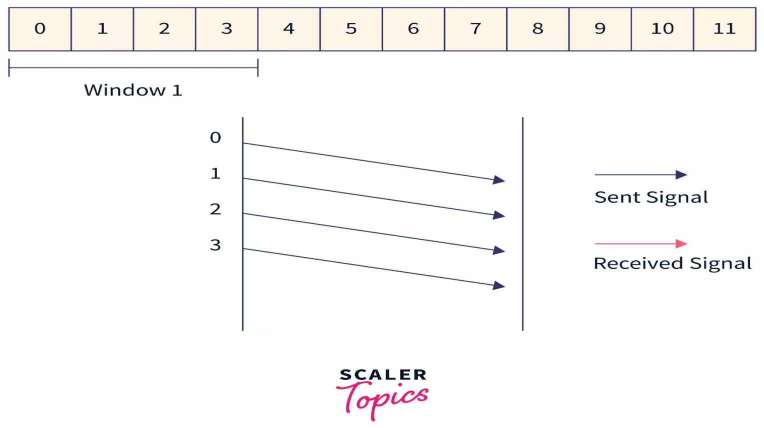
SCALER Topics

Working of Go-Back-N Protocol

1. Data packets are divided into multiple frames. Each frame contains information about the destination address, the error control mechanism it follows, etc. These multiple frames are numbered so that they can be distinguished from each other.

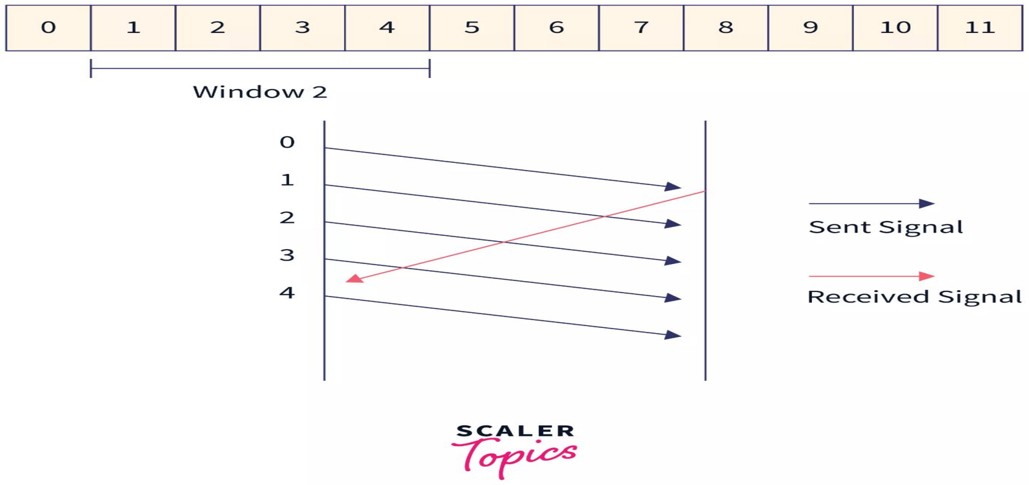
2. The integer 'N' in Go Back 'N' ARQ tells us the size of the window i.e. the number of frames that are sent at once from sender to receiver. Suppose the window size 'N' is equal to 4. Then, 4 frames (frame 0, frame 1, frame 2, and frame 3) will be sent first from sender to receiver.





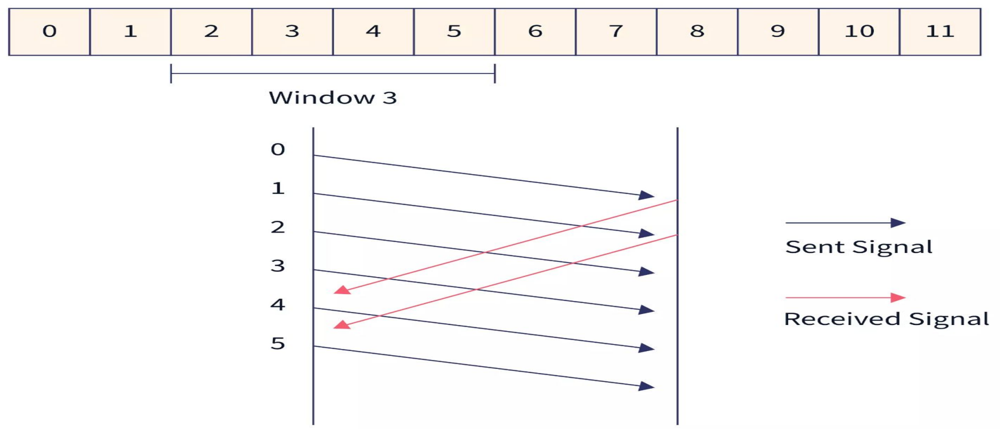
Working of Go-Back-N Protocol

3. The receiver sends the acknowledgment of frame 0. Then the sliding window moves by one and frame 4 is sent.



Working of Go-Back-N Protocol

4. The receiver sends the acknowledgment of frame 1. Then the sliding window moves by one and frame 3 is sent.

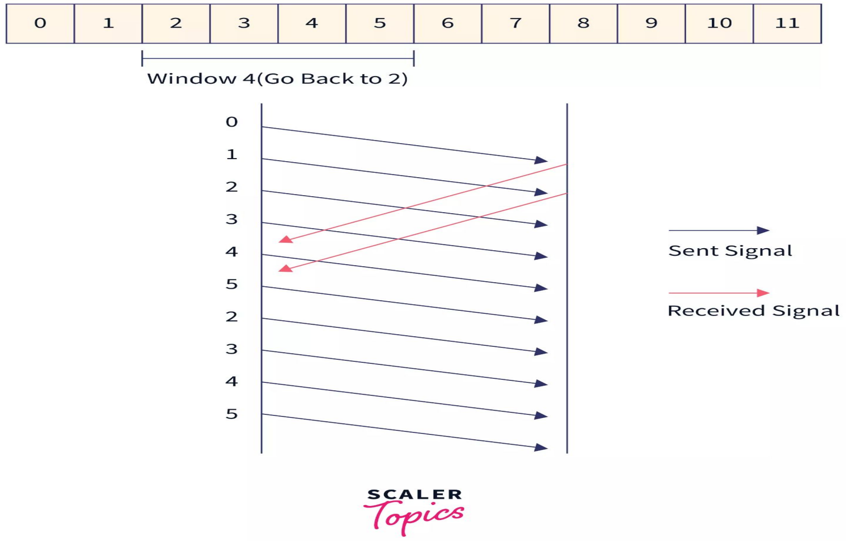


SCALER Topics

Working of Go-Back-N Protocol

5. The sender waits for the acknowledgment for a fixed amount of time. If the sender does not get the acknowledgment for a frame in time, it considers the frame to be corrupted. Then the sliding window moves to the start of the corrupted frame and all the frames in the window are retransmitted.

For example, if the sender does not receive the acknowledgment for frame 2, it retransmits all the frames in the windows i.e. frame



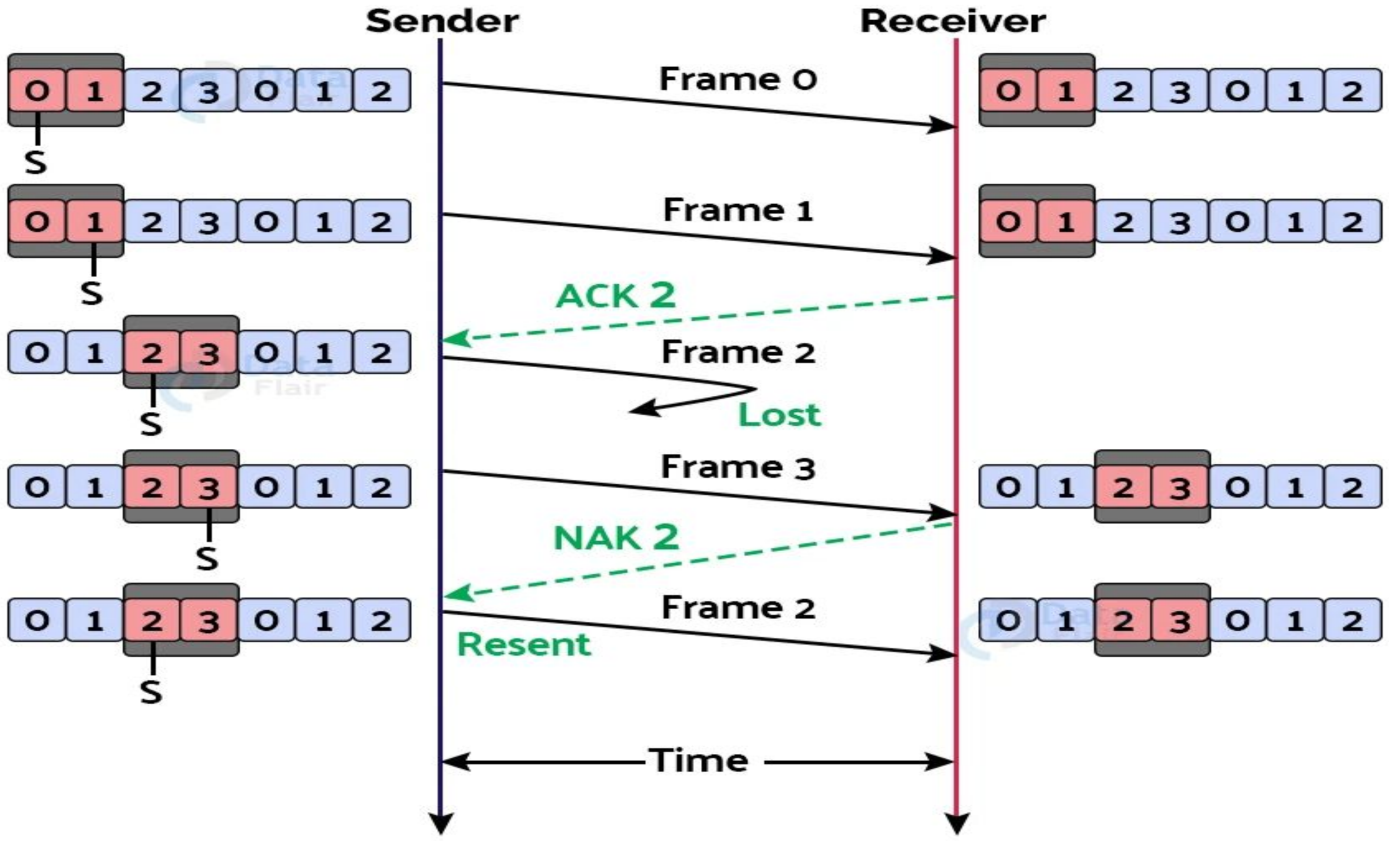
Selective- Repeat Protocol

- The Selective Repeat ARQ protocol is a type of error-control protocol used in data communication to ensure reliable delivery of data over a noisy channel.
- Unlike the Go-Back-N ARQ protocol which retransmits the entire window of packets, the Selective Repeat ARQ protocol retransmits only the packets that were not correctly received.
- In the Selective Repeat ARQ protocol, the sender transmits a window of packets to the receiver, and the receiver sends back an acknowledgment (ACK) to the sender indicating successful receipt of the packets.
- If the receiver detects an error in a packet, it sends a negative acknowledgment (NAK) to the sender requesting retransmission of that packet.
- In the Selective Repeat ARQ protocol, the sender maintains a timer for each packet in the window.
- If the sender does not receive an ACK for a packet before its timer expires, the sender retransmits only that packet.

Selective- Repeat Protocol

- At the receiver side, if a packet is received correctly, the receiver sends back an ACK with the sequence number of the next expected packet. However, if a packet is received with errors, the receiver discards the packet and sends back a NAK with the sequence number of the packet that needs to be retransmitted.
- Unlike Go-Back-N ARQ, in Selective Repeat ARQ, the receiver buffer is maintained for all packets that are not in sequence. When a packet with a sequence number different from the expected sequence number arrives at the receiver, it is buffered, and the receiver sends an ACK for the last in-order packet it has received.
- If a packet with a sequence number that the receiver has already buffered arrives, it is discarded, and the receiver sends an ACK for the last in-order packet it has received.

Selective- Repeat Protocol



Bidirectional Protocols: Piggybacking

- The four protocols we discussed earlier in this section are all unidirectional: data packets flow in only one direction and acknowledgments travel in the other direction.
- In real life, data packets are normally flowing in both directions: from client to server and from server to client. This means that acknowledgments also need to flow in both directions.
- A technique called **piggybacking** is used to improve the efficiency of the bidirectional protocols.
- When a packet is carrying data from A to B, it can also carry acknowledgment feedback about arrived packets from B; when a packet is carrying data from B to A, it can also carry acknowledgment feedback about the arrived packets from A.
- The client and server each use two independent windows: send and receive.

Internet Transport-Layer Protocols

- **UDP**
  - User Datagram Protocol:
  - User Datagram
  - UDP Services
  - UDP Applications
  -
- **TCP**
  - TCP Services
  - TCP Features
  - Segment
  - A TCP Connection
  - State Transition Diagram
  - Windows in TCP
  - Flow Control
  - Error Control
  - TCP Congestion Control
  - TCP Timers
  - Options

User Datagram Protocol (UDP):

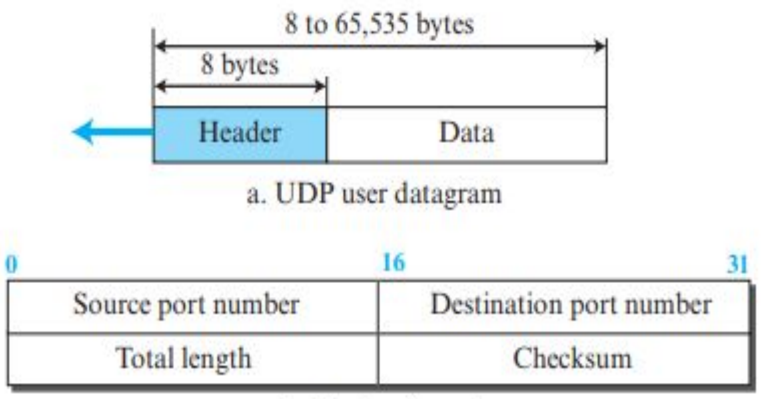
- The User Datagram Protocol (UDP) is a connectionless, unreliable transport protocol.
- It does not add anything to the services of IP except for providing process-to-process communication instead of host-to-host communication.
- UDP is a very simple protocol using a minimum of overhead.
- If a process 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).
- Figure 3.39 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.
- 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.

User Datagram

Figure 3.39 *User datagram packet format*



b. Header format

UDP Services

- **Process-to-Process Communication**
- **Connectionless Services**
- **Flow Control**
- **Error Control**
- **Congestion Control**
- **Encapsulation and Decapsulation**
- **Multiplexing and Demultiplexing**
- **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.

UDP Services

- **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 Applications

- 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.
- 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.
- UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but not in the TCP software.
- UDP is used for management processes such as SNMP.
- UDP is used for some route updating protocols such as Routing Information Protocol (RIP).
- UDP is normally used for interactive real-time applications that cannot tolerate uneven delay between sections of a received message.

Transmission Control Protocol (TCP)

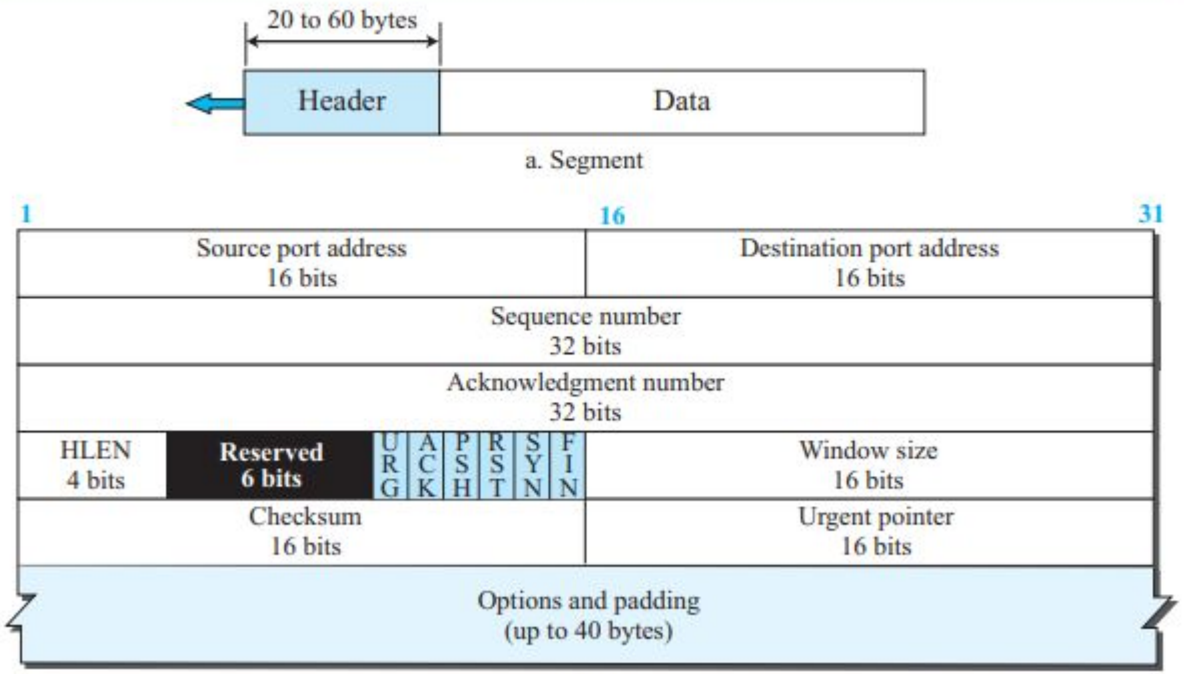
- Transmission Control Protocol (TCP) is a connection-oriented, reliable protocol.
- TCP explicitly defines connection establishment, data transfer, and connection teardown phases to provide a connection-oriented service.
- TCP uses a combination of GBN and SR protocols to provide reliability.
- To achieve this goal, TCP uses checksum (for error detection), retransmission of lost or corrupted packets, cumulative and selective acknowledgments, and timers.

TCP Services

- Process-to-Process Communication
- Stream Delivery Service
- Full-Duplex Communication
- Multiplexing and Demultiplexing
- Connection-Oriented Service
- Reliable Service

Segment:

Figure 3.44 *TCP segment format*



b. Header

Segment:

- **Source port:** It defines the port of the application, which is sending the data. So, this field contains the source port address, which is 16 bits.
- **Destination port:** It defines the port of the application on the receiving side. So, this field contains the destination port address, which is 16 bits.
- **Sequence number:** This field contains the sequence number of data bytes in a particular session.
- **Acknowledgment number:** When the ACK flag is set, then this contains the next sequence number of the data byte and works as an acknowledgment for the previous data received. For example, if the receiver receives the segment number 'x', then it responds 'x+1' as an acknowledgment number.
- **HLEN:** It specifies the length of the header indicated by the 4-byte words in the header. The size of the header lies between 20 and 60 bytes.
- **Reserved:** It is a 4-bit field reserved for future use, and by default, all are set to zero.

Segment:

- **Flags**

  There are six control bits or flags:

  1. **URG:** It represents an urgent pointer. If it is set, then the data is processed urgently.
  2. **ACK:** If the ACK is set to 0, then it means that the data packet does not contain an acknowledgment.
  3. **PSH:** If this field is set, then it requests the receiving device to push the data to the receiving application without buffering it.
  4. **RST:** If it is set, then it requests to restart a connection.
  5. **SYN:** It is used to establish a connection between the hosts.
  6. **FIN:** It is used to release a connection, and no further data exchange will happen.

- **Window size**

  It is a 16-bit field. It contains the size of data that the receiver can accept. This field is used for the flow control between the sender and receiver and also determines the amount of buffer allocated by the receiver for a segment. The value of this field is determined by the receiver.

Segment:

- **Checksum**

  It is a 16-bit field. This field is optional in UDP, but in the case of TCP/IP, this field is mandatory.
- **Urgent pointer**

  It is a pointer that points to the urgent data byte if the URG flag is set to 1. It defines a value that will be added to the sequence number to get the sequence number of the last urgent byte.
- **Options**

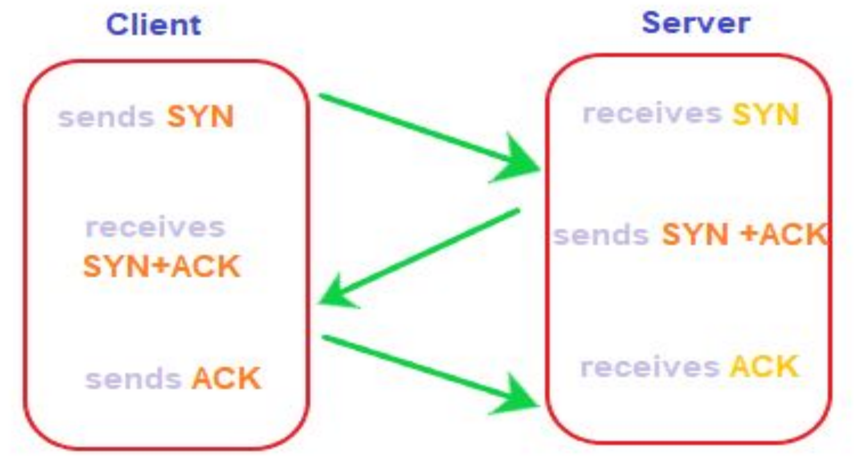
  It provides additional options. There can be up to 40 bytes of optional information in the TCP header.

A TCP Connection (A 3-way handshake)

- Handshake refers to the process to establish connection between the client and server.
- Handshake is simply defined as the process to establish a communication link.
- To transmit a packet, TCP needs a three way handshake before it starts sending data.
- The reliable communication in TCP is termed as **PAR** (Positive Acknowledgement Retransmission).
- When a sender sends the data to the receiver, it requires a positive acknowledgement from the receiver confirming the arrival of data. If the acknowledgement has not reached the sender, it needs to resend that data. The positive acknowledgement from the receiver establishes a successful connection.

A TCP Connection (A 3-way handshake)

- Here, the server is the server and client is the receiver.
- The diagram shows 3 steps for successful connection.
- A 3-way handshake is commonly known as SYN-SYN-ACK and requires both the client and server response to exchange the data.
- SYN means **synchronize Sequence Number** and ACK means **acknowledgment**.
- Each step is a type of handshake between the sender and the receiver.



A TCP Connection (A 3-way handshake)

The three handshakes are discussed in the below steps:

Step 1: SYN

SYN is a segment sent by the client to the server. It acts as a **connection request** between the client and server. It informs the server that the client wants to establish a connection. Synchronizing sequence numbers also helps synchronize sequence numbers sent between any two devices, where the same SYN segment asks for the sequence number with the connection request.

Step 2: SYN-ACK

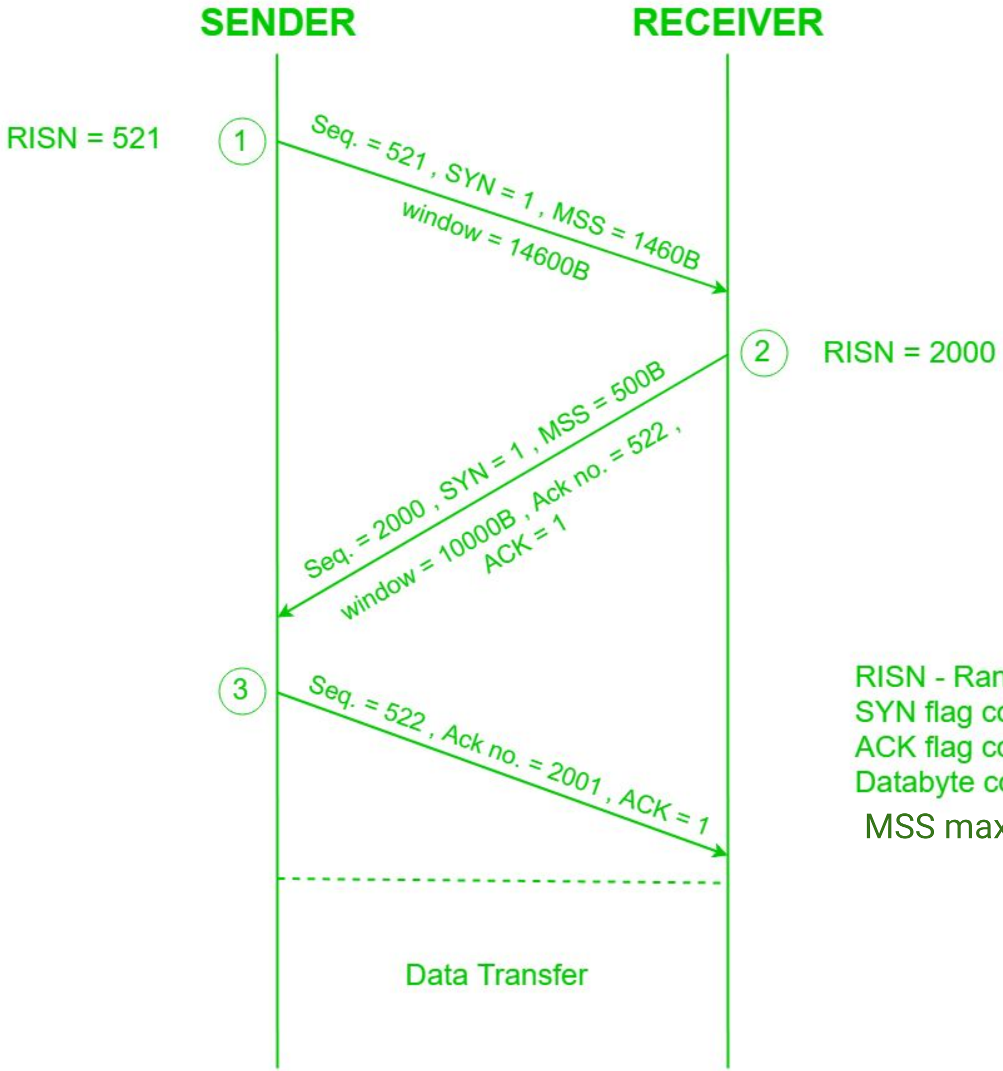
It is an SYN-ACK segment or an SYN + ACK segment sent by the server. The ACK segment informs the client that the server has received the connection request and it is ready to build the connection. The SYN segment informs the sequence number with which the server is ready to start with the segments.

A TCP Connection (A 3-way handshake)

Step 3: ACK

ACK (Acknowledgment) is the last step before establishing a successful TCP connection between the client and server. The ACK segment is sent by the client as the response of the received ACK and SN from the server. It results in the establishment of a reliable data connection.

After these three steps, the client and server are ready for the data communication process. TCP connection and termination are full-duplex, which means that the data can travel in both the directions simultaneously.

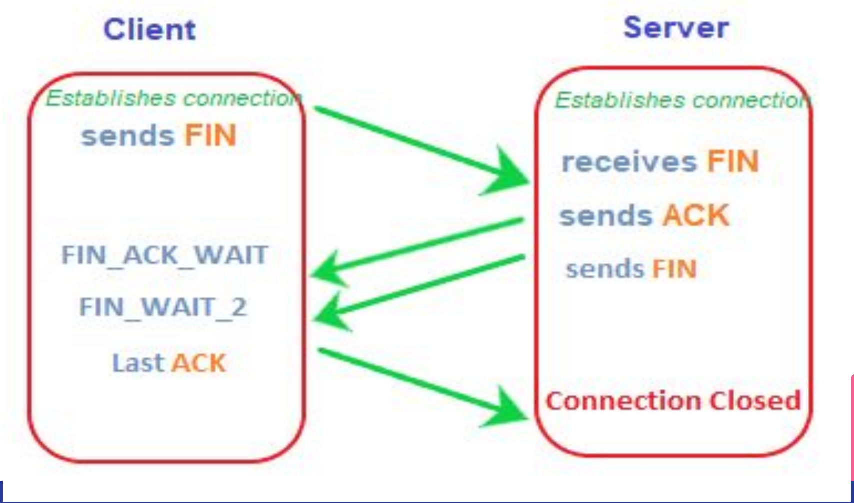

SENDER
RECEIVER
RISN = 521
1
Seq. = 521 , SYN = 1 , MSS = 1460B
window = 14600B
2
RISN = 2000
Seq. = 2000 , SYN = 1 , MSS = 500B
window = 10000B , Ack no. = 522 ,
ACK = 1
3
Seq. = 522 , Ack no. = 2001 , ACK = 1
Data Transfer

RISN - Random initial sequence number
SYN flag consumes 1 sequence number
ACK flag consumes 0 sequence number
Databyte consumes 1 sequence number

MSS maximum Segment size

TCP Termination (A 4-way handshake)

- To terminate or stop the data transmission, it requires a 4-way handshake.
- The segments required for TCP termination are similar to the segments to build a TCP connection (ACK and SYN) except the FIN segment.
- The **FIN segment** specifies **a termination request** sent by one device to the other.
- The client is the data transmitter and the server is a receiver in a data transmission process between the sender and receiver.



TCP Termination (A 4-way handshake)

Step 1: FIN

FIN refers to the **termination request** sent by the client to the server. The first FIN termination request is sent by the client to the server. It depicts the start of the termination process between the client and server.

Step 2: FIN_ACK_WAIT

The client waits for the ACK of the FIN termination request from the server. It is a **waiting state** for the client.

Step 3: ACK

The server sends the ACK (Acknowledgement) segment when it receives the FIN termination request. It depicts that the server is ready to close and terminate the connection.

TCP Termination (A 4-way handshake)

Step 4: FIN _WAIT_2

The client waits for the FIN segment from the server. It is a type of approved signal sent by the server that shows that the server is ready to terminate the connection.

Step 5: FIN

The FIN segment is now sent by the server to the client. It is a confirmation signal that the server sends to the client. It depicts the successful approval for the termination.

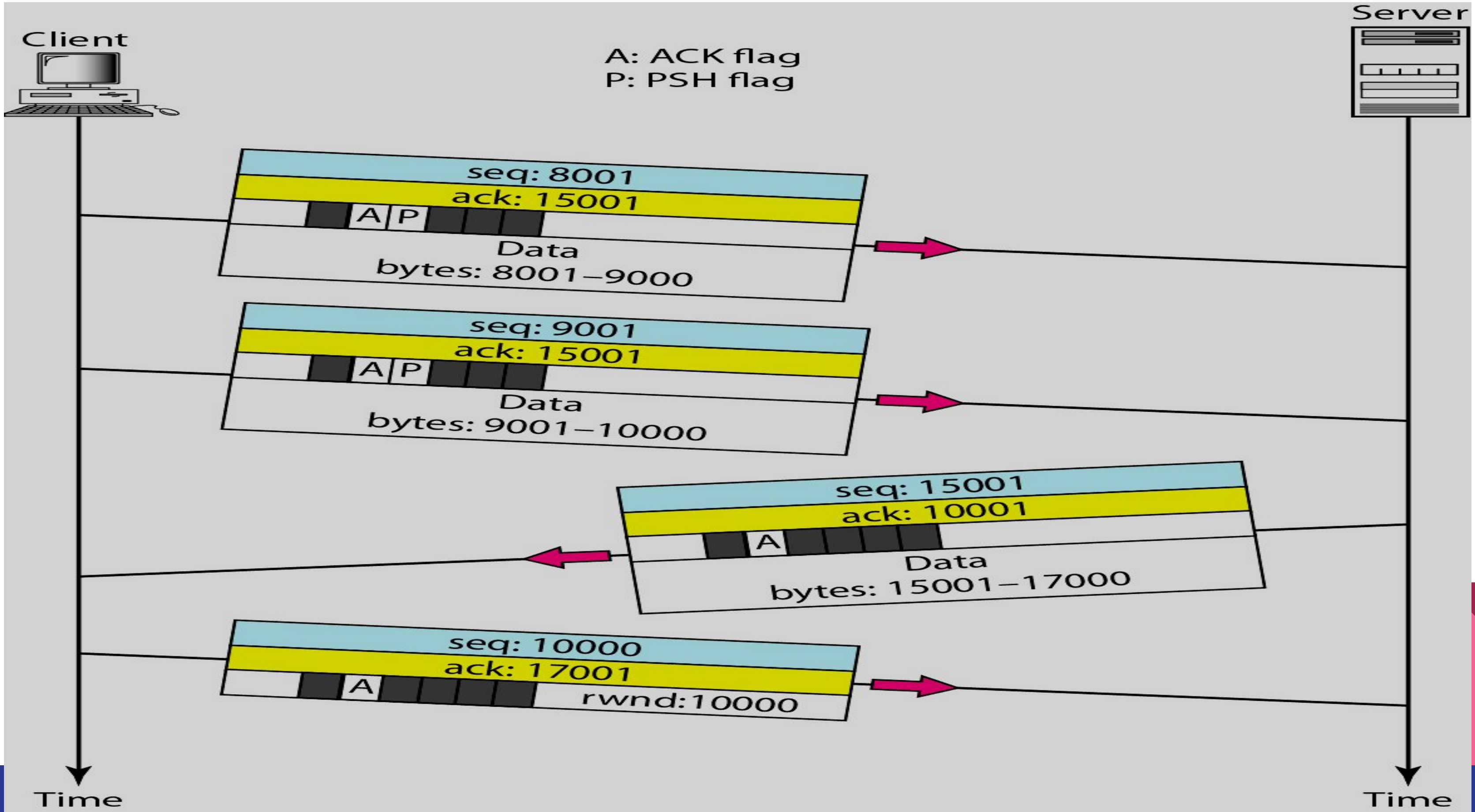
Step 6: ACK

The client now sends the ACK (Acknowledgement) segment to the server that it has received the FIN signal, which is a signal from the server to terminate the connection. As soon as the server receives the ACK segment, it terminates the connection.

Data Transfer

- The client and server can send data and acknowledgments in both directions.
- Data traveling in the same direction as an acknowledgement are carried on the same segment.
- The acknowledgement is piggybacked with the data. Next figure shows the example.
- In this example, after a connection is established, the client sends 2,000 bytes of data in two segments.
- The server then sends 2000 bytes in one segment.
- The client send one more segment. The first three segment carry both data and acknowledgement.
- Last segment carries only an acknowledgement because there is no more data to be sent.
- The data segment sent by client have **PSH (push) flag** to set so that the server TCP tries to deliver data to server process as soon as they are received.
- The segment from the **server, does not set the push flag**.
- Most TCP implementations have the option to set or not set this flag.

Data Transfer



Pushing Data

- 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.
- There are occasions in which the application program has no need for this flexibility.
- For example, an application program that communicates interactively with the another application program on the other end. The application program on one site wants to send a keystroke 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 situations. The application program at the sender can request a Push operation. This means that **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.**
- TCP can choose whether or not to use this features.

Connection Termination

- Any of the two parties involved in exchanging data (client and server) can close the connection, although it is usually initiated by the client.
- Most implementations today allow **two options** for connection termination:

1. Three way handshaking
2. Four way handshaking with half close option

Three-way Handshaking

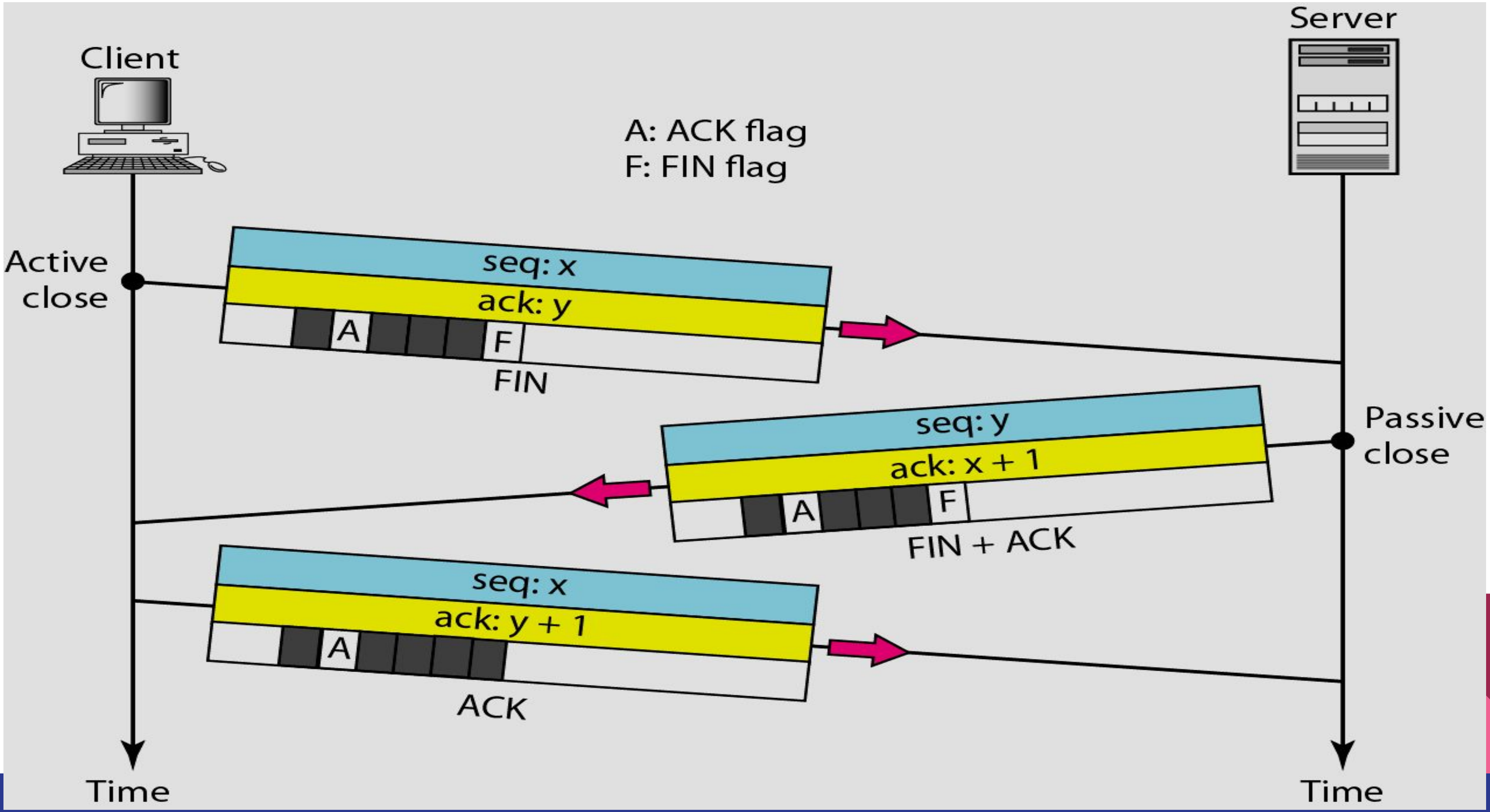
Step 1:

- ✔ In common situations, 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 figure.
- ✔ If it is only a control segment, it consumes only **one sequence number**.

Step 2:

- ✔ The server TCP, after receiving the FIN request, 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.
- ✔ At the same time **to announce the closing of the connection in the other directions**.
- ✔ 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**.

Three-way Handshaking



Three-way Handshaking

- **Step 3:**

✔ the client TCP sends the last segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server.

✔ This segment contains the acknowledgement number, which is one plus the sequence number received in FIN segment from the server.

✔ This segment cannot carry data and consumes no sequence numbers.

Windows in TCP

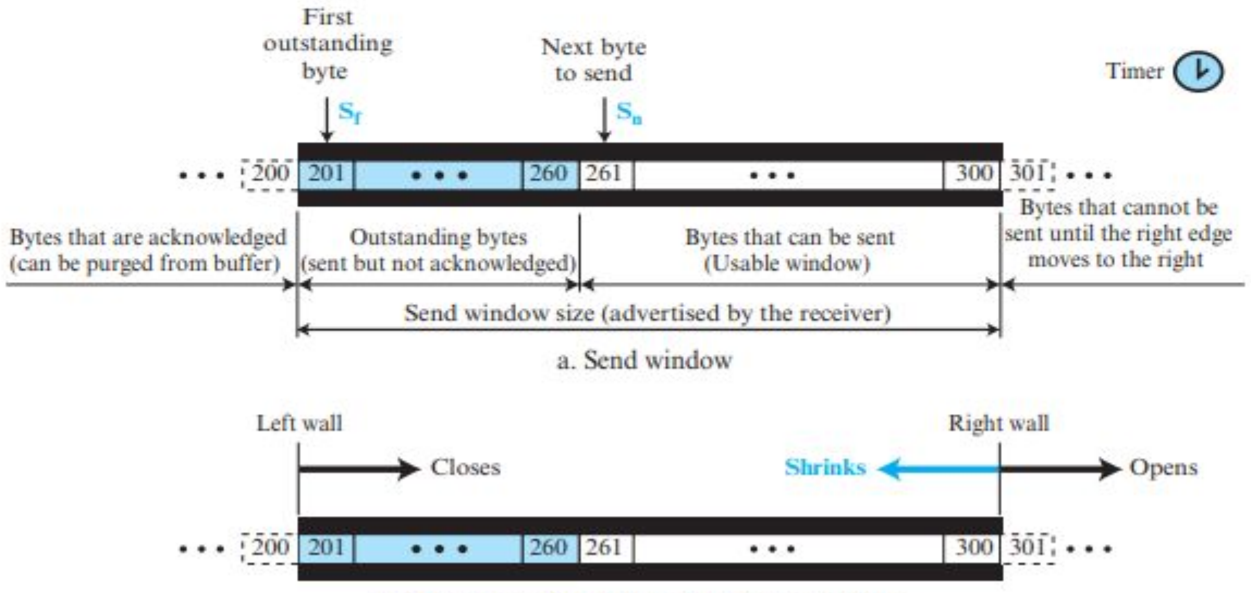
- TCP uses two windows (send window and receive window) for each direction of data transfer, which means four windows for a bidirectional communication.
- To make the discussion simple, we make an unrealistic assumption that communication is only unidirectional (say from client to server); the bidirectional communication can be inferred using two unidirectional communications with piggybacking.

Windows in TCP

Send Window

- The window size is 100 bytes but later we see that the send window size is dictated by the receiver (flow control) and the congestion in the underlying network (congestion control).
- The figure shows how a send window opens, closes, or shrinks.

Figure 3.54 *Send window in TCP*


First outstanding byte
S_f
Next byte to send
S_n
Timer
200
201
260
261
300
301
Bytes that are acknowledged (can be purged from buffer)
Outstanding bytes (sent but not acknowledged)
Bytes that can be sent (Usable window)
Bytes that cannot be sent until the right edge moves to the right
Send window size (advertised by the receiver)
a. Send window
Left wall
Closes
Right wall
Shrinks
Opens
200
201
260
261
300
301

b. Opening, closing, and shrinking send window

Windows in TCP

The send window in TCP is similar to one used with the Selective-Repeat protocol, but with some differences:

1. One difference is the nature of entities related to the window. The window size in SR is the number of packets, but the window size in TCP is the number of bytes. Although actual transmission in TCP occurs segment by segment, the variables that control the window are expressed in bytes.

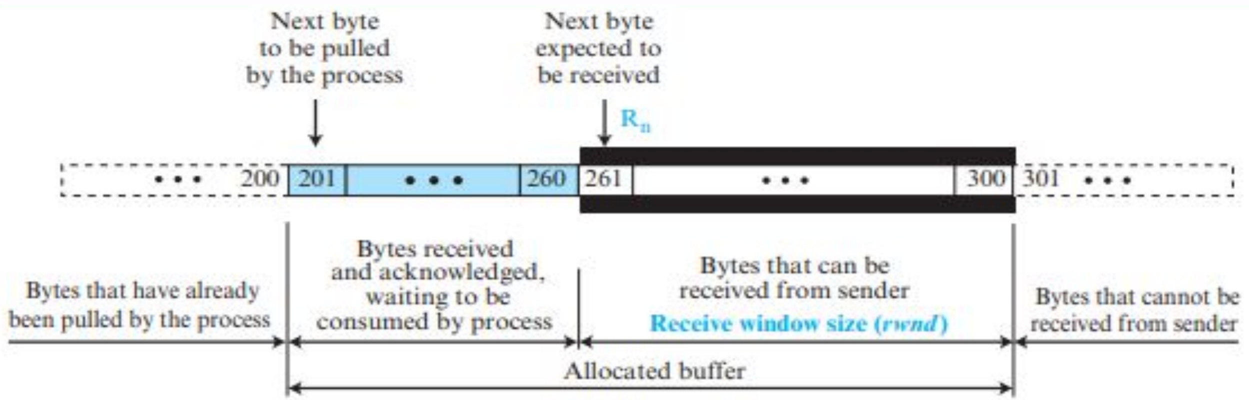
2. The second difference is that, in some implementations, TCP can store data received from the process and send them later, but we assume that the sending TCP is capable of sending segments of data as soon as it receives them from its process.

3. Another difference is the number of timers. The theoretical Selective-Repeat protocol may use several timers for each packet sent, but as mentioned before, the TCP protocol uses only one timer.

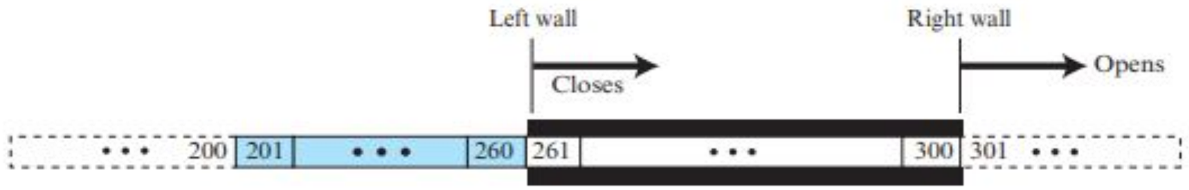
Windows in TCP

- **Receive Window**
- Figure 3.55 shows an example of a receive window.
- The window size is 100 bytes.
- The figure also shows how the receive window opens and closes; in practice, the window should never shrink.

Figure 3.55 *Receive window in TCP*


Next byte to be pulled by the process
Next byte expected to be received
R_n
200
201
260
261
300
301
Bytes that have already been pulled by the process
Bytes received and acknowledged, waiting to be consumed by process
Bytes that can be received from sender
Receive window size (*rwnd*)
Bytes that cannot be received from sender
Allocated buffer

a. Receive window and allocated buffer



b. Opening and closing of receive window

Windows in TCP

There are two differences between **the receive window** in TCP and the one we used for SR.

1. The first difference is that TCP allows the receiving process to pull data at its own pace.

This means that part of the allocated buffer at the receiver may be occupied by bytes that have been received and acknowledged, but are waiting to be pulled by the receiving process.

The receive window size is then always smaller or equal to the buffer size, as shown in Figure 3.55.

The receive window size determines the number of bytes that the receive window can accept from the sender before being overwhelmed (flow control). In other words, the receive window size, normally called rwnd, can be determined as:

Windows in TCP

2. The second difference is the way acknowledgments are used in the TCP protocol.

Remember that an acknowledgement in SR is selective, defining the uncorrupted packets that have been received.

The major acknowledgment mechanism in TCP is a cumulative acknowledgment announcing the next expected byte to receive (in this way TCP looks like GBN, discussed earlier).

The new version of TCP, however, uses both cumulative and selective acknowledgements; we will discuss these options on the book website.

TCP Timers

- TCP uses several timers to ensure that excessive delays are not encountered during communications.
- Several of these timers are elegant, handling problems that are not immediately obvious at first analysis.
- Each of the timers used by TCP is examined in the following sections, which reveal its role in ensuring data is properly sent from one connection to another.

TCP Timers

TCP implementation uses four timers –

Retransmission Timer –

- To retransmit lost segments, TCP uses retransmission timeout (RTO).
- When TCP sends a segment the timer starts and stops when the acknowledgment is received.
- If the timer expires timeout occurs and the segment is retransmitted.
- RTO (retransmission timeout is for 1 RTT) to calculate retransmission timeout we first need to calculate the RTT(round trip time).

RTT three types –

- **Measured RTT(RTTm)** – The measured round-trip time for a segment is the time required for the segment to reach the destination and be acknowledged, although the acknowledgement may include other segments.

TCP Timers

- **Smoothed RTT(RTTs) –**
  - It is the weighted average of RTTm.
  - RTTm is likely to change and its fluctuation is so high that a single measurement cannot be used to calculate RTO.

```
Initially -> No value

After the first measurement -> RTTs=RTTm

After each measurement -> RTTs= (1-t)*RTTs + t*RTTm

  Note: t=1/8 (default if not given)
```

TCP Timers

- **Deviated RTT(RTTd) –**
  - Most implementations do not use RTTs alone so RTT deviated is also calculated to find out RTO.

```
Initially -> No value

After the first measurement -> RTTd=RTTm/2

After each measurement -> RTTd= (1-k)*RTTd + k*(RTTm-RTTs)

  Note: k=1/4 (default if not given)
```

TCP Timers

- **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.

TCP Timers

- **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 will crash.
  - 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. If there is no response after 10 probes, each of which is 75 s apart, it assumes that the client is down and terminates the connection.

TCP Timers

- **Time Wait Timer –**
  - This timer is used during tcp connection termination.
  - The timer starts after sending the last Ack for 2nd FIN and closing the connection.
  - *After a TCP connection is closed, it is possible for datagrams that are still making their way through the network to attempt to access the closed port.*
  - *The quiet timer is intended to prevent the just-closed port from reopening again quickly and receiving these last datagrams.*
  - The **quiet timer** is usually set to twice the maximum segment lifetime (the same value as the Time-To-Live field in an IP header), ensuring that all segments still heading for the port have been discarded.