MODULE 5: TRANSPORT LAYER

Q.1) TCP VS UDP

Feature	TCP (Transmission Control Protocol)	UDP (User Datagram Protocol)
Connection Type	Connection-oriented; establishes a connection before data transfer.	Connectionless; sends data without establishing a connection.
Reliability	Reliable; ensures delivery through acknowledgments and retransmissions.	Unreliable; no delivery guarantees or acknowledgment of receipt.
Ordering	Maintains the order of packets; data is received in the same order it is sent.	No guarantee of packet order; packets may arrive out of sequence.
Error Checking	Provides extensive error checking; uses checksums and mechanisms for error recovery.	Basic error checking with checksums; no error recovery mechanisms.
Flow Control	Yes, uses techniques like windowing to control the rate of data transmission.	No flow control; sender can send packets at any time.
Congestion Control	Yes, implements congestion control algorithms to manage traffic and prevent network overload.	No congestion control; packets can be dropped if the network is congested.
Header Size	Larger header size (20 bytes or more) due to additional control information.	Smaller header size (8 bytes), leading to lower overhead.
Speed	Slower due to connection establishment, error checking, and recovery processes.	Faster because of minimal overhead and no connection setup.
Data Transmission	Stream-oriented; data is sent as a continuous stream of bytes.	Message-oriented; data is sent as discrete packets (datagrams).

Retransmission	Automatically retransmits lost packets based on acknowledgment from the receiver.	No retransmission; if packets are lost, they are not sent again.
Session Management	Manages sessions; maintains state information for ongoing connections.	No session management; each packet is treated independently.
Use Cases	Suitable for applications requiring reliability and ordered delivery (e.g., web browsing, file transfers, email).	Suitable for applications where speed is critical and some data loss is acceptable (e.g., live video streaming, online gaming, VoIP).
Network Utilization	Can lead to higher network overhead due to acknowledgments and error handling.	More efficient in terms of network utilization due to lower overhead.
Broadcast/Multicast Support	Limited broadcast/multicast support; primarily designed for point-to-point communication.	Strong support for broadcasting and multicasting; allows sending data to multiple receivers.
Header Information	Contains sequence numbers, acknowledgment numbers, and other control flags.	Contains source and destination ports, length, and checksum only.
Connection Termination	Requires a formal connection termination process (e.g., FIN and ACK packets).	No formal termination; packets can be sent and stopped at any time without a specific end.

Summary

- **TCP** is best for applications that require reliability, ordered delivery, and connection management, making it ideal for tasks like web browsing and file transfers.
- **UDP**, on the other hand, is suitable for applications that prioritize speed, efficiency, and can tolerate some data loss, such as live streaming and gaming.

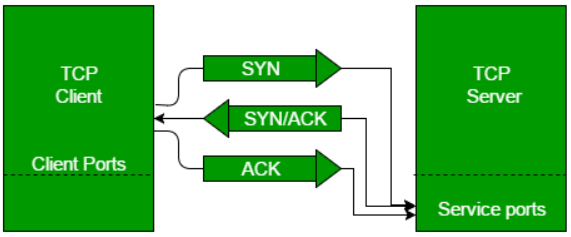
Q.2) TCP 3-Way Handshake Process

The **three-way handshake** is a fundamental process in the TCP (Transmission Control Protocol) used to establish a reliable connection between a client and a server.

This technique ensures that both parties are synchronized and ready to communicate before any data is transmitted.

It involves three steps: SYN (Synchronize), SYN-ACK (Synchronize-Acknowledge), and ACK (Acknowledge).

During the handshake, the client and server exchange initial sequence numbers and confirm the connection establishment.

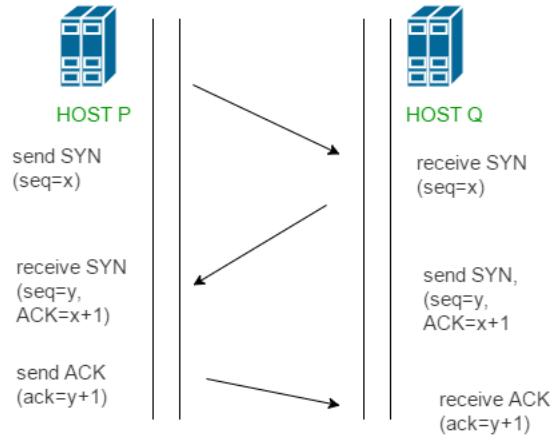


Steps of the Three-Way Handshake

1. **SYN (Synchronize)**:
    - The client initiates the connection by sending a **SYN** packet to the server.
    - This packet contains the client's initial sequence number (ISN) and indicates the client's intention to establish a connection.
    - The packet is marked with the SYN flag set to 1, signaling the server to prepare for a connection.
2. **SYN-ACK (Synchronize-Acknowledge)**:
    - Upon receiving the SYN packet, the server acknowledges the request by sending a **SYN-ACK** packet back to the client.
    - This packet serves two purposes:
        - It acknowledges the receipt of the client's SYN packet by sending an acknowledgment number (which is the client's ISN + 1).

- It also contains the server's initial sequence number (ISN) and sets both the SYN and ACK flags to 1.

3. **ACK (Acknowledge)**:
   - After receiving the SYN-ACK packet, the client sends an **ACK** packet back to the server.
   - This packet acknowledges the receipt of the server's SYN-ACK by including an acknowledgment number (which is the server's ISN + 1).
   - The ACK packet does not require any additional flags to be set other than the ACK flag.



Completion of Connection

Once the server receives the ACK packet from the client, the connection is considered established.

Both the client and server can now begin to exchange data reliably over the established connection.

Summary of Key Points

- **Reliability**: The three-way handshake ensures that both parties are ready to communicate and that their sequence numbers are synchronized.
- **Sequence Numbers**: Each party uses its initial sequence number (ISN) to keep track of the data being sent and received.

- **Flags**: The SYN and ACK flags indicate the purpose of each packet in the process.
- **Connection Establishment**: This process ensures a reliable and organized way to set up a connection before any data is transmitted.

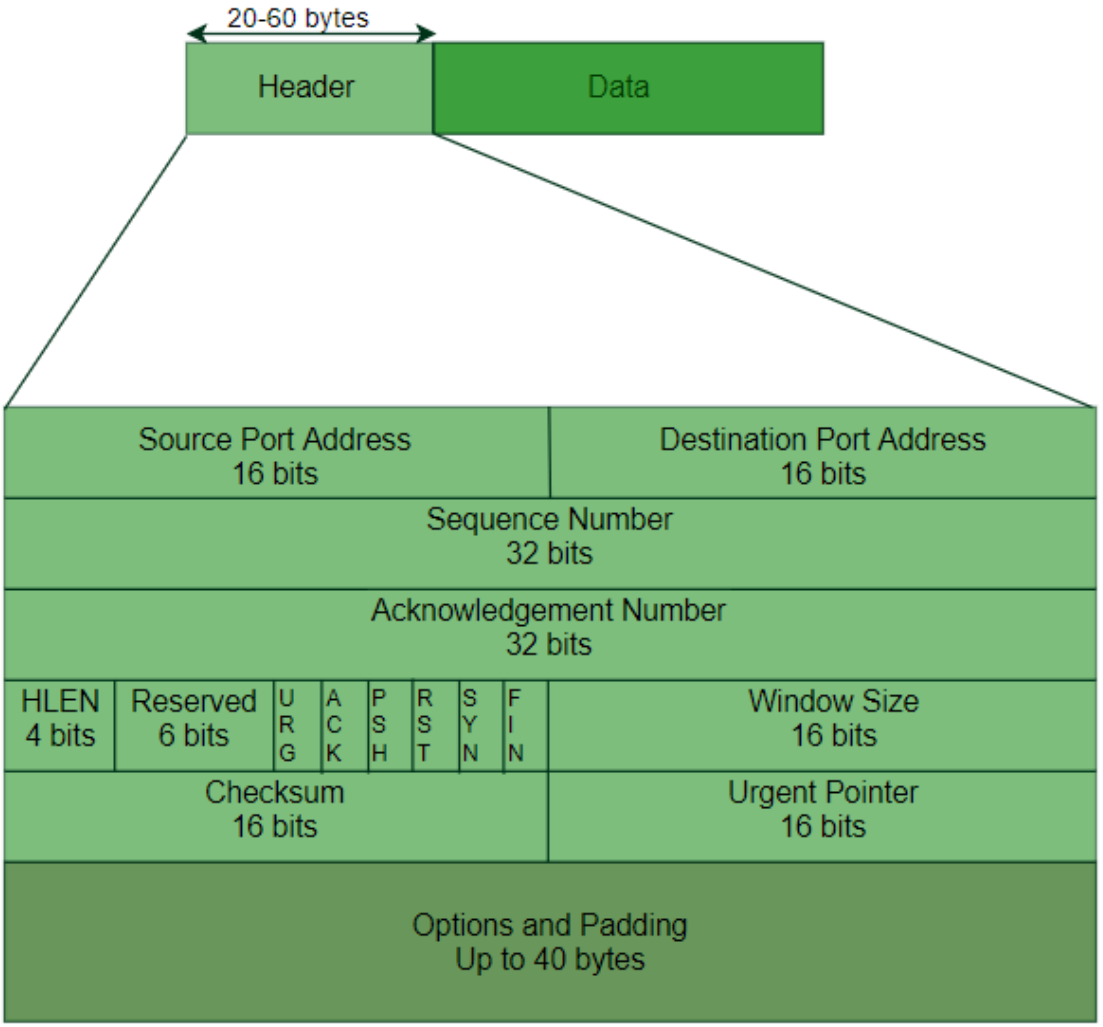
Conclusion

The three-way handshake is a crucial mechanism in TCP that ensures reliable connection establishment, allowing for effective and orderly communication between devices on a network. By synchronizing sequence numbers and confirming the readiness of both parties, TCP establishes a robust foundation for data transfer.

Q.3)DESCRIBE TCP HEADER WITH DIAGRAM

TCP Segment Structure

A TCP segment consists of data bytes to be sent and a header that is added to the data by TCP as shown:



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

The header of a TCP segment can range from 20-60 bytes. 40 bytes are for options. If there are no options, a header is 20 bytes else it can be of upmost 60 bytes. Header fields:

- **Sequence Number:** A 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 of the segments that are received out of order.
- **Acknowledgement Number:** A 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 a number of 4-byte words in the header, i.e if the header is 20 bytes(min length of **TCP header**), then this field will hold 5 (because 5 × 4 = 20) and the maximum length: 60 bytes, then it'll hold the value 15(because 15 × 4 = 60). Hence, the value of this field is always between 5 and 15.
- **Control flags:** These are 6 1-bit control bits that control connection establishment, connection termination, connection abortion, flow control, mode of transfer etc. Their function is:
  - URG: Urgent pointer is valid
  - ACK: Acknowledgement number is valid(used in case of cumulative acknowledgement)
  - PSH: Request for push
  - RST: Reset the connection
  - SYN: Synchronize sequence numbers
  - FIN: Terminate the connection
- **Window size:** This field tells the window size of the sending TCP in bytes.
- **Checksum:** This field holds the checksum for **error control** . It is mandatory in TCP as opposed to UDP.
- **Urgent pointer:** This field (valid only if the URG control flag is set) is used to point to data that is urgently required that needs to reach the receiving process at the earliest. The value of this field is added to the sequence number to get the byte number of the last urgent byte.

Q.4) EXPLAIN THE TCP CONNECTION ESTABLISHMENT AND CONNECTION RELEASE

⇒

TCP CONNECTION ESTABLISHMENT

To make the transport services, reliable TCP host must establish a connection oriented session with one another.

Connection establishment is performed by using three way handshake mechanism.

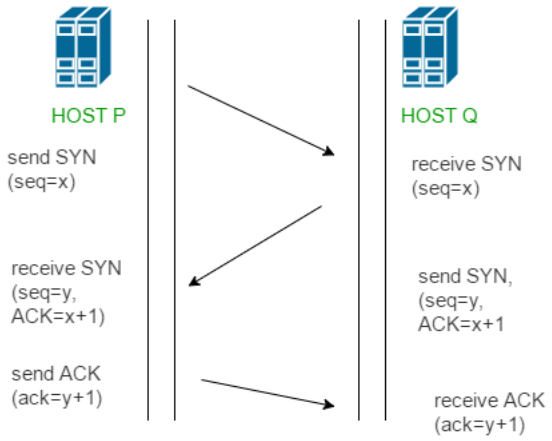
A three handshake synchronise both ends of connection by allowing both sides to agree upon initial sequence Numbers.

This mechanism also guarantees that both sites are ready to transmit data and know that the other side is ready to transmit it as well.

This is necessary, so that packets are not transmitted or transmitted during session establishment or after session termination.

Each host randomly choose a sequence number, used to track bites within the stream It is sending and receiving.

Then the three way handshake proceeds in the manner shown in the figure.



The requesting end (host P)sends a SYN segment specifying the port number of the server that the client wants to get connected to, and client's initial sequence number (x).

The server (host Q) response with its own SYN segment containing the servers initial sequence number (y)

The server also acknowledge the clients SY by acknowledging the clients SYN plus one (x+1). a SYN consumes one sequence number.

The client must acknowledge this SYN from the server by acknowledging the servers SYN plus one (SEQ= x+1, ACK = y+1).

This is how TCP connection is established.

CONNECTION RELEASE (CONNECTION TERMINATION)

The connection termination process in TCP is more flexible, allowing for **half-closed** connections where one side can finish sending data while the other side still receives.

The four-way handshake method is typically used to release the connection, ensuring an orderly and graceful close.

Steps in Connection Release:

1. **Step 1: FIN (Finish)**

- The side that wants to terminate the connection (usually the client) sends a **FIN** packet to signal that it has finished sending data.
- This packet includes the **FIN flag**, indicating no more data will be sent from the client.

2. **Step 2: ACK (Acknowledge)**

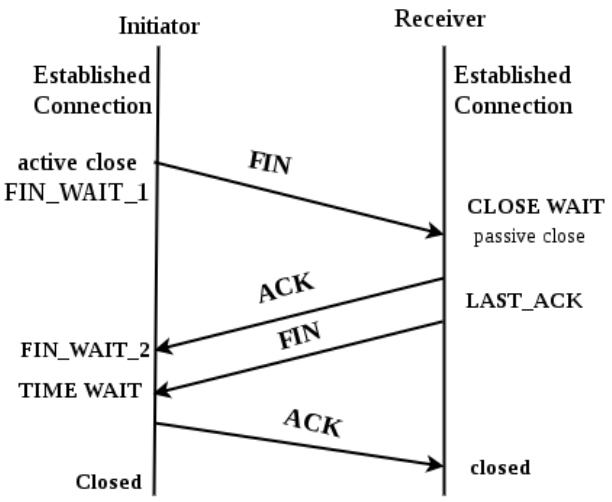
- The server receives the FIN packet, sends an **ACK** packet to confirm receipt, and acknowledges that it is aware of the client's intention to close.
- This ACK serves as confirmation but does not terminate the server's transmission to the client, so the connection remains **half-open**.

3. **Step 3: FIN (Finish)**

- When the server is ready to close its side of the connection, it sends its own **FIN** packet to the client.
- This FIN flag signals that the server has finished sending data.

4. **Step 4: ACK (Acknowledge)**

- The client responds with an **ACK** packet to confirm the server's FIN packet, fully closing the connection on both ends.
- This ACK marks the final step, fully releasing the connection.

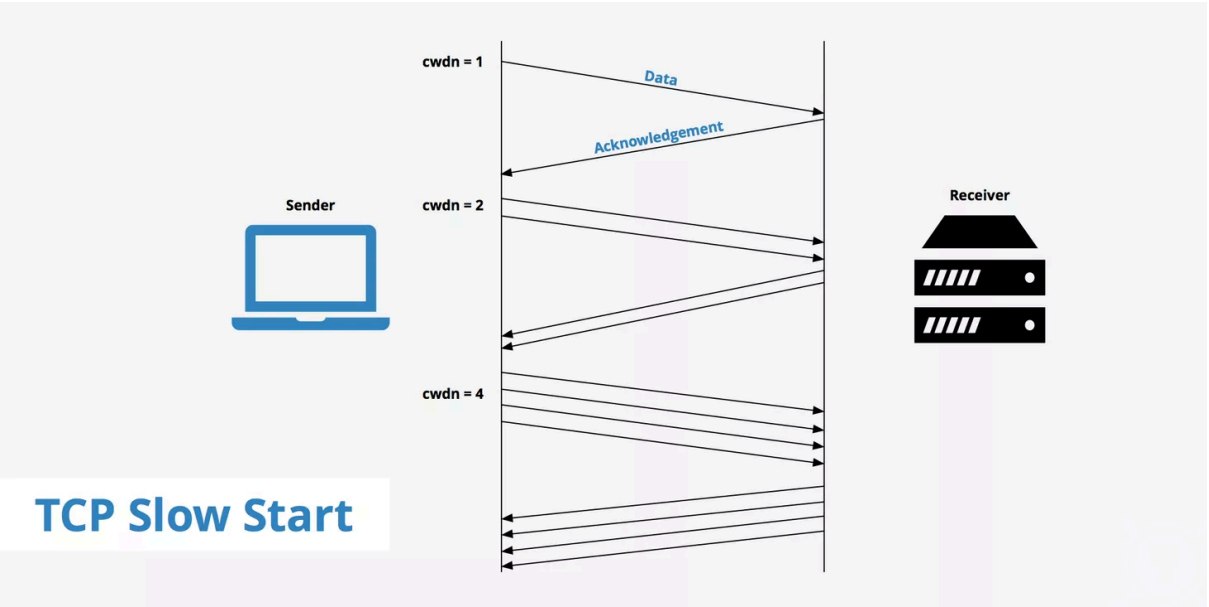


Q.5) SHORT NOTE ON SLOW START ALGORITHM

Slow Start in TCP

Slow Start is a crucial congestion control algorithm used in the Transmission Control Protocol (TCP) to manage network traffic and prevent congestion.

It is part of TCP's congestion control mechanism and helps ensure efficient use of network resources while minimizing packet loss.



Key Concepts

1. **Congestion Window (cwnd)**:
   - The **congestion window** is a TCP variable that determines the number of bytes a sender can transmit before needing an acknowledgment (ACK) from the receiver.
   - Initially, the **cwnd** is set to a small value (typically one or two Maximum Segment Sizes (MSS)).
2. **Exponential Growth**:
   - In Slow Start, the **cwnd** increases exponentially with each acknowledgment received. For every ACK received, the congestion window increases by one MSS.
   - This means that if the current window size is `n` MSS, the next window size will be `2n` MSS after receiving `n` ACKs.
   - This exponential growth continues until the congestion window reaches a predefined threshold (often called the **slow start threshold (ssthresh)**).
3. **Threshold**:
   - The **ssthresh** is a critical value that determines when the TCP connection transitions from Slow Start to the **Congestion Avoidance** phase.
   - When the **cwnd** exceeds the **ssthresh**, the growth rate changes from exponential to linear (increasing by one MSS per round-trip time (RTT)).
4. **Collision Detection**:
   - If packet loss is detected (often indicated by a timeout or duplicate ACKs), the TCP connection reduces the **cwnd** back to the initial size (or half of the previous size) and resets the **ssthresh**.
     This allows the connection to resume transmission while adapting to network conditions.

Steps in Slow Start

1. **Initialization**:
   - The **cwnd** is set to a low value (e.g., 1 MSS) and the **ssthresh** is set to a high value.

2. **Transmission**:
    - The sender transmits packets according to the current **cwnd**. For each received ACK, the **cwnd** is increased exponentially.
3. **Threshold Check**:
    - When the **cwnd** reaches the **ssthresh**, the TCP algorithm transitions from Slow Start to Congestion Avoidance.
4. **Congestion Event Handling**:
    - If a congestion event (packet loss) occurs, the **cwnd** is reduced, and the process restarts.

Advantages of Slow Start

- **Efficient Bandwidth Usage**: Slow Start allows TCP to probe the network capacity gradually, helping to find the optimal sending rate without overwhelming the network.
- **Adaptability**: It adjusts dynamically based on the current network conditions, allowing for efficient recovery from congestion.

Summary

Slow Start is a fundamental component of TCP's congestion control strategy that helps ensure reliable data transmission by gradually increasing the amount of data sent over the network. By using exponential growth of the congestion window, it effectively manages network traffic and minimizes the risk of congestion and packet loss.