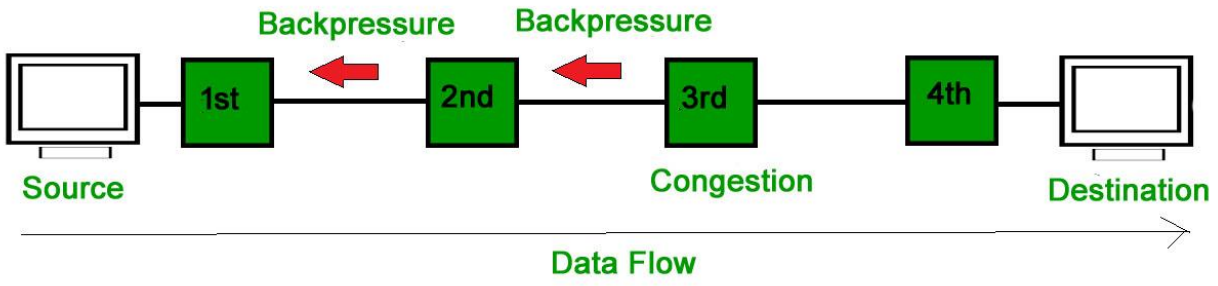
Question 1.

How does the Transport Layer handle flow control and congestion control? Explain the role of TCP's sliding window mechanism in flow control

The Transport Layer in the OSI and TCP/IP models is responsible for ensuring reliable data transfer between devices. Two critical functions it performs are flow control and congestion control

Flow Control:

Network flow control is the ability to regulate the speed at which the items are transferred between the communicating devices with a view of avoiding overwhelming the receiving device. It is crucial because the sender may be in a capacity to send data at a faster pace than the receiver can analyze it. If flow control is not done then one possible scenario is that the buffer at the receiver side gets filled up with data which are lost. In TCP (Transmission Control Protocol) , flow control is realized through the use of windowing; the sender can send a number of packets to the destination before getting an acknowledgement for the first one but the number of packets that can be in the network at any one time has a limit. The receiver determines the size of the window up through messages that report how much buffer capacity is left.



TCP's Sliding Window Mechanism in Flow Control:

TCP's sliding window mechanism works as follows:

1. **Window Size**: Each device has a window size, which defines the maximum number of bytes (or packets) that can be sent without receiving an acknowledgment from the receiver.
2. **Advertised Window**: The receiver informs the sender of how much space is available in its buffer via the TCP header field called the **advertised window**. This helps prevent the sender from overwhelming the receiver with too much data at once.
3. **Sliding the Window**: As the receiver acknowledges packets, the window "slides" forward, allowing the sender to send more data. If the receiver's buffer starts filling up, it can reduce the window size, signaling the sender to slow down. Conversely, when the buffer frees up, the window size can increase, allowing more data to be sent.

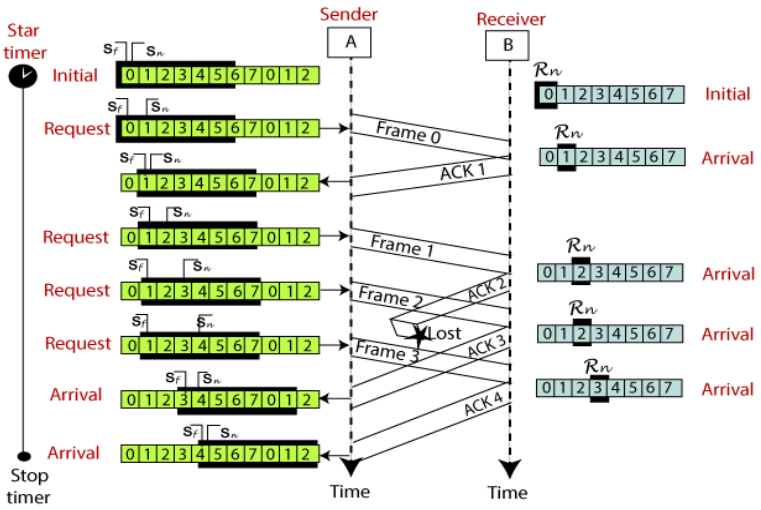
4. **Zero Window**: If the receiver's buffer is full, it can advertise a window size of zero, instructing the sender to stop sending data until space becomes available. The sender then periodically probes the receiver to check if it's ready to resume data transmission.

Congestion Control:

Congestion control is different from flow control as it deals with managing network congestion rather than individual sender-receiver pairs. It ensures that the sender does not flood the network with too much data, which could lead to packet loss and increased latency.

TCP uses a variety of algorithms for congestion control, such as:

- **Slow Start**: TCP begins by sending data at a low rate, increasing the rate exponentially until it detects packet loss.
- **Congestion Avoidance**: After slow start, TCP increases the rate more gradually (linearly) to avoid congestion.
- **Fast Retransmit and Fast Recovery**: If a packet is lost, TCP can quickly resend it without needing a full retransmission timeout.



Question 2.

What is the role of segmentation in the Transport Layer? How does the Transport Layer divide large data blocks into smaller segments and reassemble them at the destination?

Role of Segmentation in the Transport Layer

The **Transport Layer** plays a crucial role in managing data transmission between devices in a network. One of its core responsibilities is **segmentation**, which involves breaking down large data blocks into smaller, manageable pieces (called segments) for efficient transmission and reassembling them at the destination.

How the Transport Layer Divides and Reassembles Data:

1. Segmentation at the Sender:

When a sender has a large block of data (such as a file or message), the Transport Layer (specifically in protocols like TCP) divides this data into smaller units called **segments**.

Key steps in segmentation:

- **Determine Segment Size**: The Transport Layer calculates the size of each segment based on the **Maximum Segment Size (MSS)**, which is typically derived from the network's MTU.

  For example, if the MTU is 1500 bytes, the segment size might be slightly smaller (e.g., 1460 bytes), leaving space for headers.

- **Assign Sequence Numbers**: Each segment is assigned a **sequence number** that indicates its position within the entire data block. This ensures that segments can be reordered and reassembled correctly at the destination.
- **Add Headers**: Each segment is encapsulated with a Transport Layer header containing important information, including the sequence number, source and destination port numbers, and error-checking data (checksum).

2. Transmission Over the Network:

The segments are then sent across the network, typically using either TCP (for reliable, connection-oriented transmission) or UDP (for faster but unreliable, connectionless transmission).

3. Reassembly at the Receiver:

When the segments reach the destination, the Transport Layer at the receiving end reassembles them into the original data block.

Key steps in reassembly:

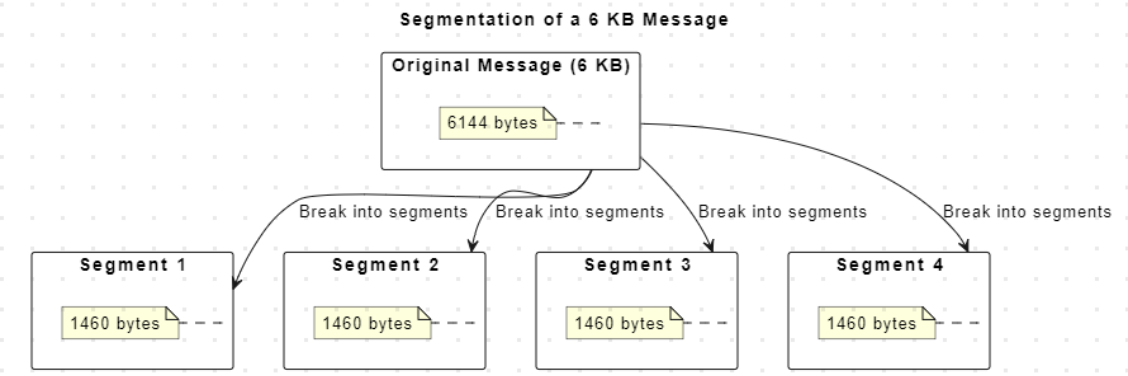
- **Sequence Numbering**: The Transport Layer uses the sequence numbers from the segment headers to arrange the segments in the correct order.
- **Handle Lost or Corrupted Segments**: If any segments are missing or corrupted, the Transport Layer requests retransmission of those specific segments (only in TCP, not in UDP).
- **Deliver Data to Application Layer**: Once all segments have been received and reassembled in the correct order, the Transport Layer passes the complete data block to the Application Layer.

Segmentation Example:

Let's say an application needs to send a 6 KB (6144 bytes) message, but the MTU is 1500 bytes. The Transport Layer breaks the message into four segments:

- Segment 1: 1460 bytes
- Segment 2: 1460 bytes
- Segment 3: 1460 bytes
- Segment 4: 1460 bytes (Headers will take up some of the 1500 bytes.)

The receiver will receive and reassemble these segments in the correct order using the sequence numbers provided in the headers.



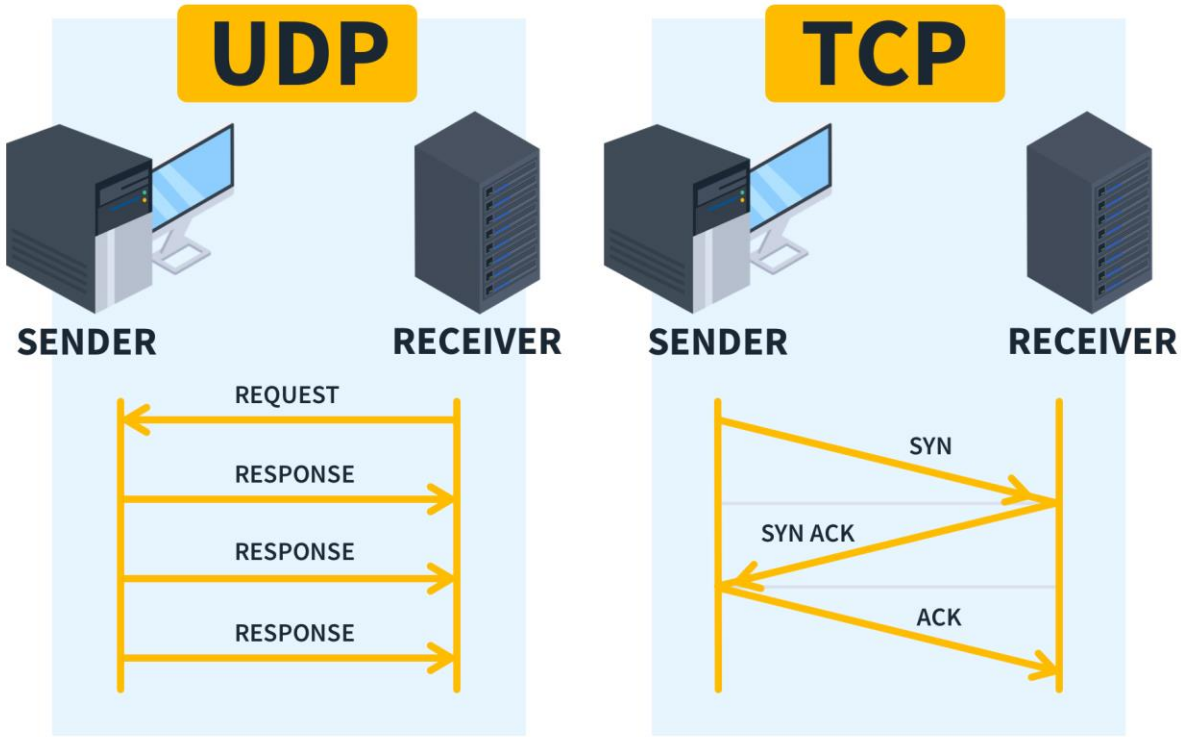
Question 3.

What are ports, and why are they important in the Transport Layer? Explain the difference between well-known, registered, and dynamic ports

What are Ports in the Transport Layer?

Ports are numerical identifiers used by the Transport Layer (e.g., TCP and UDP) to differentiate between multiple processes or services running on a single device. Ports allow a single computer to handle multiple network connections simultaneously by directing data to the correct application.

Each port is associated with a specific service or application on a device. When data is sent to or from a device, the Transport Layer uses port numbers to identify which application should handle the data.



Importance of Ports in the Transport Layer

1. **Multiplexing and Demultiplexing**: Ports enable the Transport Layer to **multiplex** (combine) data from multiple applications into a single stream for transmission and **demultiplex** (separate) incoming data back into the appropriate application on the receiving side. For example, web traffic (port 80 for HTTP or 443 for HTTPS) can be handled simultaneously with email traffic (port 25 for SMTP or 110 for POP3) on the same machine.
2. **Service Identification**: Each service running on a machine listens on a specific port, allowing the network to distinguish between different services. This ensures that incoming data is delivered to the correct application. For example, when a web browser requests a web page, the request is sent to port 80 or 443 on the web server.
3. **Security**: Ports also play a key role in **security**. Firewalls, for example, use port numbers to filter traffic by blocking or allowing data based on the service or application.

Port Number Ranges

Port numbers are divided into three ranges based on their purpose:

1. **Well-known Ports**:
    - **Range**: 0–1023
    - These are reserved for standard services and are typically used by system-level or widely used protocols.
    - Examples:
        - Port 80: HTTP (web traffic)
        - Port 443: HTTPS (secure web traffic)
        - Port 25: SMTP (email)
        - Port 21: FTP (file transfer)
    - Well-known ports are predefined and are often assigned by the **Internet Assigned Numbers Authority (IANA)**.
2. **Registered Ports**:
    - **Range**: 1024–49151
    - These ports are assigned by IANA to specific services or applications that may not be as universally used as those with well-known ports but still require a dedicated port.
    - Examples:
        - Port 3306: MySQL database
        - Port 8080: Alternative HTTP
        - Port 5432: PostgreSQL database
    - These ports are often used by user-defined applications or other specialized services that require a standard port assignment.
3. **Dynamic or Private Ports**:

- **Range**: 49152–65535
- These ports are not assigned to any specific service or protocol and are used for **temporary or dynamic connections**. They are often used by clients when initiating a connection to a server.
- For example, when your web browser opens a new connection to a website, it is assigned a dynamic port in this range to handle the outgoing request, while the server responds on the well-known port (like 80 or 443).
- Dynamic ports are typically chosen randomly and are not reserved for any particular application.

Summary of Port Differences

Port Range	Purpose	Examples
0–1023	**Well-known Ports**: Reserved for standard, widely-used protocols	HTTP (80), HTTPS (443), FTP (21)
1024–49151	**Registered Ports**: Assigned to specific applications or services	MySQL (3306), PostgreSQL (5432), Alternate HTTP (8080)
49152–65535	**Dynamic/Private Ports**: Used for temporary, client-side connections	Randomly assigned for outgoing connections

Question 4.

What is the process of connection establishment and termination in TCP? Explain the steps involved in the three-way handshake.

1. Connection Establishment (Three-Way Handshake)

The **three-way handshake** is used to establish a reliable connection between a client and a server. It ensures that both parties are synchronized and ready to exchange data.

- **Step 1: SYN (Synchronize)**
  - The client sends a **SYN** (synchronize) segment to the server. This segment includes a **sequence number**, which will be used to track the order of data transmission.
  - The client is effectively saying, "I want to start a connection and here's my sequence number."
  - **Example:**

        - Client → Server: `SYN, Seq = X`
- **Step 2: SYN-ACK (Synchronize-Acknowledgment)**
    - The server responds with a **SYN-ACK** segment. This acknowledges the client's SYN request and provides its own **sequence number**. The ACK part acknowledges the receipt of the client's SYN and the sequence number it sent.
    - The server is saying, "I received your request, here's my sequence number, and I acknowledge yours."
    - **Example:**
        - Server → Client: `SYN, Seq = Y, ACK = X + 1`
- **Step 3: ACK (Acknowledgment)**
    - The client responds with an **ACK** segment, which acknowledges the server's SYN-ACK. This final acknowledgment confirms that both sides are synchronized and ready to communicate.
    - The client is saying, "I received your sequence number and acknowledgment, now we're ready to communicate."
    - **Example:**
        - Client → Server: `ACK, Seq = X + 1, ACK = Y + 1`

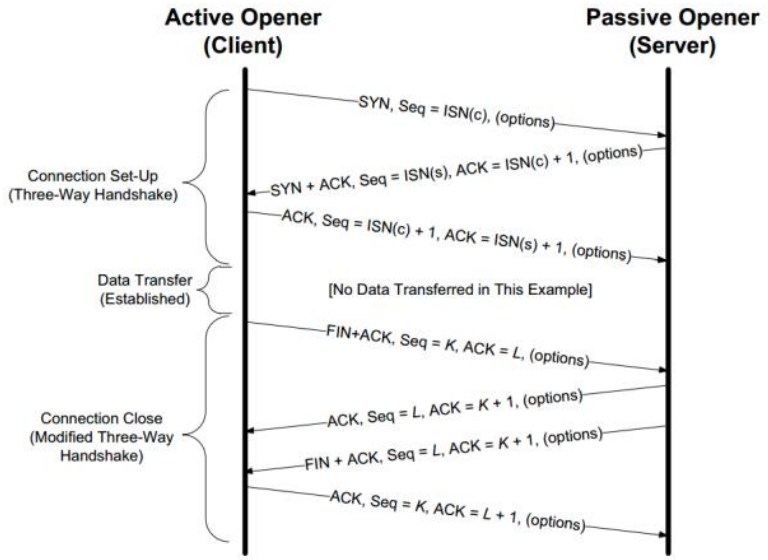
At this point, the connection is established, and data transmission can begin.

2. Connection Termination

To terminate a TCP connection, a four-step process occurs, involving the exchange of FIN (Finish) and ACK (Acknowledgment) segments. Either side (client or server) can initiate termination.

- **Step 1: FIN**
    - The side initiating the termination (client or server) sends a **FIN** segment, signaling that it has finished sending data.
    - **Example:**
        - Client → Server: `FIN, Seq = X`
- **Step 2: ACK**
    - The receiving side (server) responds with an **ACK** to acknowledge the receipt of the FIN. This means it received the termination request, but it may still need time to send any remaining data.
    - **Example:**
        - Server → Client: `ACK, Seq = Y, ACK = X + 1`
- **Step 3: FIN**
    - When the receiving side has finished sending its data, it sends a **FIN** segment to the initiator, signaling that it is done with the transmission as well.
    - **Example:**
        - Server → Client: `FIN, Seq = Z`
- **Step 4: ACK**
    - The initiator of the termination acknowledges the final FIN segment, completing the termination process.
    - **Example:**

- Client → Server: `ACK, Seq = X + 1, ACK = Z + 1`



Question 5.

Describe the concept of packet loss and how the Transport Layer handles it. How does TCP retransmit lost packets, and what techniques does it use to detect packet loss?

Packet Loss in TCP

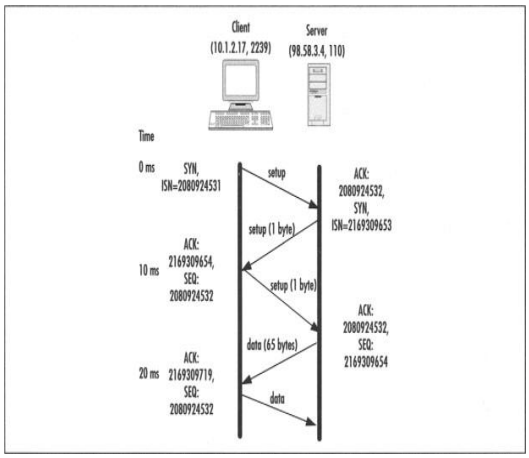
Packet loss occurs when data packets traveling across a network fail to reach their destination. This can happen due to network congestion, faulty hardware, signal interference, or other issues. TCP, being a reliable transport protocol, is designed to handle packet loss effectively, ensuring that all data is delivered accurately.

How TCP Handles Packet Loss

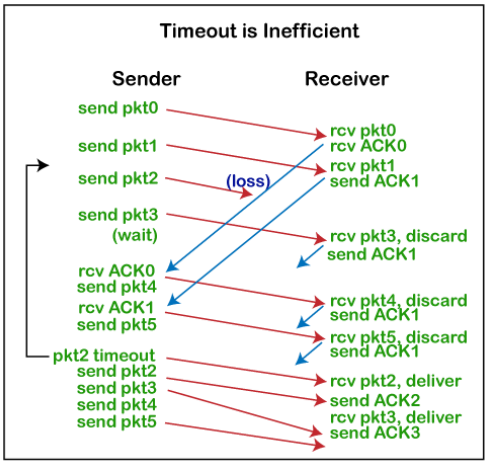
TCP uses several techniques to detect and recover from packet loss, ensuring reliable data delivery:

1. **Acknowledgments (ACKs) and Sequence Numbers**:
    - TCP numbers every byte of data sent and uses acknowledgments to confirm receipt. Each segment contains a sequence number that identifies the position of the first byte in the segment.

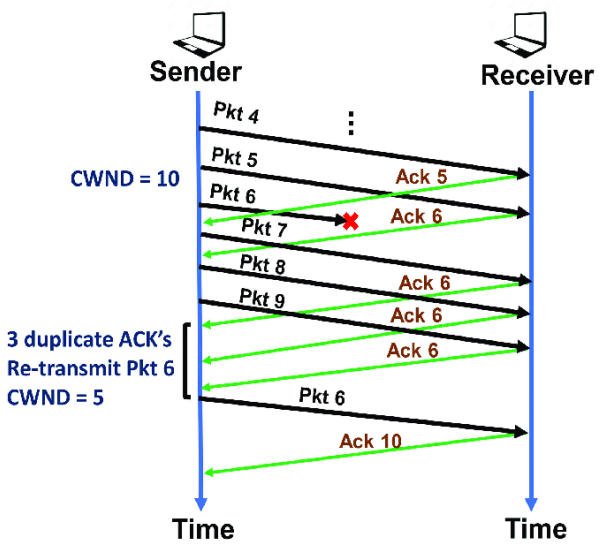
- When a packet arrives successfully, the receiver sends an acknowledgment (ACK) back to the sender with the next expected sequence number.
- If the sender does not receive an acknowledgment within a certain time frame, it assumes the packet was lost and retransmits it.



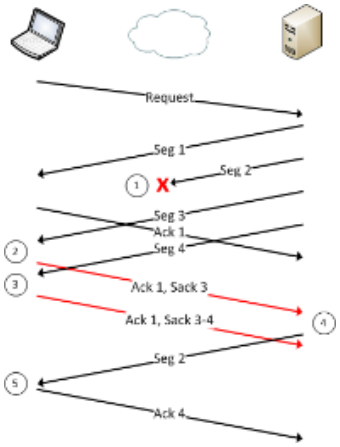
2. **Timeouts (Retransmission Timeout - RTO)**:
   - TCP uses a **retransmission timeout (RTO)** mechanism to detect packet loss. The sender starts a timer when it sends a packet and expects an acknowledgment (ACK) within a calculated timeout period.
   - If the ACK is not received before the timer expires, the packet is assumed to be lost, and TCP **retransmits** the lost packet.
   - The RTO is dynamically adjusted based on network conditions (using algorithms like the **exponential backoff**), so that TCP adapts to different levels of network congestion or delay.



3. **Duplicate Acknowledgments (DupACKs) and Fast Retransmit**:
    - When a packet arrives out of order at the receiver, it will acknowledge the last in-order packet it received, generating **duplicate ACKs** (DupACKs) for the sender.
    - For example, if packet 3 is lost, and packet 4 arrives, the receiver will send an acknowledgment for packet 2 (the last correctly received packet), even though it received packet 4. The sender will receive multiple ACKs with the same sequence number (DupACKs).
    - **Fast Retransmit** is triggered when the sender receives **three duplicate ACKs** for the same sequence number. Instead of waiting for a timeout, TCP immediately retransmits the missing packet, assuming it has been lost.



4. **Selective Acknowledgment (SACK) Option**:
    - The **SACK** mechanism allows the receiver to inform the sender which blocks of data have been received successfully, even if some packets in the middle were lost. This helps TCP retransmit only the missing packets instead of retransmitting a larger block of data.
    - Without SACK, the sender might retransmit all packets following a lost packet, even if some of them were already received. SACK improves efficiency by only retransmitting the lost segments.



Techniques Used by TCP to Detect Packet Loss

1. **Timeouts**: If an acknowledgment is not received within the retransmission timeout (RTO) period, the sender assumes that the packet was lost and retransmits it.
2. **Duplicate Acknowledgments**: The sender monitors for **three duplicate ACKs**, which indicate that a packet may have been lost. In this case, TCP uses the **fast retransmit** mechanism to immediately resend the missing packet.
3. **Selective Acknowledgments (SACK)**: TCP can use the SACK option to detect which specific segments were lost, helping to identify packet loss more accurately and avoid unnecessary retransmissions.