1. TCP Connection Establishment (Three-Way Handshake)

Purpose: To establish a reliable communication channel between the sender and receiver.

Steps:

1. **Sender Sends SYN:**
   - The sender sends a **SYN** (synchronize) message to the receiver to initiate a connection request.
2. **Receiver Responds with SYN-ACK:**
   - The receiver acknowledges the request by sending a **SYN-ACK** back to the sender.
3. **Sender Acknowledges with ACK:**
   - The sender sends an **ACK** back to the receiver to confirm the connection is established.

Outcome:

- A reliable connection is now established, and both sides are ready to exchange data.

---

2. TCP Data Transfer (Using Segments)

Purpose: To reliably transfer data between the sender and receiver.

Steps:

1. **Sender Divides Data into Segments:**
   - The sender breaks down large amounts of data into smaller **TCP segments**. Each segment contains a header (with sequence numbers, etc.) and data.
2. **Sender Sends Segments:**
   - The sender transmits these segments to the receiver.
3. **Receiver Acknowledges Segments:**
   - The receiver sends an **ACK** for the last successfully received byte.
4. **Flow Control:**
   - The receiver may adjust the **receive window size** to tell the sender how much more data it can send based on its buffer space.

Outcome:

- Data flows smoothly, with reliable acknowledgment and retransmission in case of packet loss.

---

3. TCP Flow Control

Purpose: To prevent the sender from overwhelming the receiver with too much data.

Steps:

1. **Receiver Advertises Buffer Space:**
   - The receiver sends the **receive window size** in each acknowledgment, indicating how much data it can still accept.
2. **Sender Adjusts Data Transmission:**
   - The sender uses this window size to adjust the rate at which it sends data. If the buffer is full, the sender will slow down or stop until the receiver processes some data.

Outcome:

- Prevents data overflow and ensures the receiver can handle the incoming data.

---

4. TCP Retransmission Due to Timeout

Purpose: To resend lost or unacknowledged data.

Steps:

1. **Sender Sends Data:**
   - The sender sends a TCP segment and starts a timer for that segment.
2. **No Acknowledgment Received:**
   - If the sender doesn't receive an acknowledgment within the timeout period, it assumes the segment is lost.
3. **Retransmit Data:**
   - The sender retransmits the lost segment.

Outcome:

- Lost segments are retransmitted to ensure that all data is received by the receiver.

---

5. Fast Retransmit

Purpose: To quickly recover lost packets without waiting for the timeout.

Steps:

1. **Receiver Sends Duplicate ACKs:**
   - If the receiver receives out-of-order segments, it sends **duplicate ACKs** for the last correctly received byte.
2. **Sender Receives Three Duplicate ACKs:**
   - After receiving three duplicate ACKs, the sender assumes the packet in question was lost.

3. **Retransmit the Lost Packet:**
    - The sender immediately retransmits the missing segment without waiting for the timeout.

Outcome:

- Faster recovery from packet loss, reducing delays.

---

6. TCP Connection Termination (Four-Way Handshake)

Purpose: To gracefully close a TCP connection once data transfer is complete.

Steps:

1. **Sender Sends FIN:**
    - The sender indicates it has finished sending data by sending a **FIN** (finish) segment.
2. **Receiver Acknowledges FIN:**
    - The receiver acknowledges the sender's **FIN** with an **ACK**.
3. **Receiver Sends FIN:**
    - The receiver sends its own **FIN** to indicate it is also finished sending data.
4. **Sender Acknowledges Receiver's FIN:**
    - The sender acknowledges the receiver's **FIN** with an **ACK**, completing the connection termination.

Outcome:

- The connection is gracefully closed without data loss, and resources are freed up.

---

7. Selective Acknowledgment (SACK)

Purpose: To improve efficiency by acknowledging specific blocks of received data.

Steps:

1. **Receiver Sends SACK Information:**
    - When the receiver detects packet loss, it uses **SACK** to inform the sender about which blocks of data have been successfully received.
2. **Sender Retransmits Only Missing Data:**
    - The sender retransmits only the missing segments (identified by the SACK) rather than retransmitting everything after the lost segment.

Outcome:

- More efficient retransmission of data, especially in networks with packet loss.

---

8. Congestion Control in TCP

Purpose: To prevent network congestion and optimize data transfer rate.

Steps:

1. **Slow Start:**
   - Initially, TCP sends data at a low rate and gradually increases the rate as the receiver successfully acknowledges the data.
2. **Congestion Avoidance:**
   - As data continues to flow, TCP detects signs of congestion (e.g., lost packets) and reduces the transmission rate to avoid further congestion.
3. **Fast Recovery:**
   - After detecting a lost packet (via duplicate ACKs), TCP quickly retransmits the lost data and resumes a higher transmission rate, without starting the slow start process again.

Outcome:

- Efficient data transfer with minimized congestion, ensuring smooth operation even under varying network conditions.

---

9. TCP Sequence Numbers and Acknowledgments

Purpose: To ensure data is delivered in the correct order and without duplication.

Steps:

1. **Sender Assigns Sequence Numbers:**
   - Each byte of data is assigned a **sequence number**, which helps the receiver reassemble the data in the correct order.
2. **Receiver Sends Acknowledgments:**
   - The receiver sends an **ACK** with the sequence number of the next expected byte to confirm successful receipt of data.
3. **Cumulative Acknowledgment:**
   - The receiver acknowledges all data up to the next expected byte, which includes acknowledging multiple segments at once.

Outcome:

- Ensures that data is received correctly and in order, with lost or out-of-order packets retransmitted.

---

10. Error Detection and Correction (Checksum)

Purpose: To ensure data integrity by detecting transmission errors.

Steps:

1. **Sender Computes Checksum:**
    - The sender computes a **checksum** over the data to verify integrity.
2. **Receiver Validates Checksum:**
    - The receiver calculates the checksum of the received data and compares it with the checksum sent in the segment header.
3. **Action on Error:**
    - If the checksums don't match, the receiver discards the segment, prompting a retransmission.

Outcome:

- Ensures the integrity of data during transmission by detecting and correcting errors caused by noise or corruption in the network.

---

11. Selective Repeat (ARQ - Automatic Repeat reQuest)

Purpose: An error correction protocol used to retransmit only the lost or corrupted packets, instead of retransmitting all the data after a loss.

Steps in Selective Repeat:

1. **Sender Divides Data:** The sender breaks down data into smaller packets and assigns **sequence numbers** to each packet.
2. **Sender Sends Packets:** The sender sends the packets, one after the other, to the receiver.
3. **Receiver Acknowledges Each Packet:** The receiver sends back an **ACK** for each successfully received packet, indicating the next expected packet's sequence number.
4. **Out-of-Order Packets:** If the receiver detects an out-of-order packet, it stores it in a buffer and sends an **ACK** for the last successfully received packet.
5. **Selective Retransmission:** When the sender gets duplicate ACKs (indicating a missing packet), it retransmits only the lost packet (not the entire window of data).
6. **Receiver Processes Packets:** As soon as the missing packet is received, the receiver can process all buffered data in the correct order.

Outcome: Only missing or lost packets are retransmitted, improving bandwidth efficiency and reducing unnecessary retransmissions.

---

12. Multiplexing

Purpose: The process of combining data from multiple applications into a single stream, which is sent over a single communication channel or connection.

Steps in Multiplexing:

1. **Multiple Applications Generating Data:** Different applications (e.g., HTTP, FTP) generate data to be transmitted.
2. **Adding Headers:** Each application's data is encapsulated in transport layer segments, where the segment is tagged with a **source port** and **destination port** (for identifying the application).
3. **Combining Data Streams:** The transport layer **combines** the data from all applications into a single stream and adds its own **headers**.
4. **Transmission:** The combined data stream is then passed to the **Network Layer** for delivery over a single network connection.
5. **Multiple Applications Share the Connection:** Both applications (HTTP, FTP, etc.) share the same transport layer connection, with each using a different port number.

Outcome: Allows multiple applications to share the same connection and optimizes network usage.

---

13. Demultiplexing

Purpose: The process of delivering incoming data to the correct application based on port numbers in the transport layer header.

Steps in Demultiplexing:

1. **Data Stream Arrives:** The receiver receives a data stream containing data from multiple applications (multiplexed data).
2. **Extract Transport Header:** The transport layer extracts the **destination port** from the header of the incoming segment.
3. **Identify Correct Application:** The transport layer uses the destination port to identify which application the data belongs to (e.g., HTTP uses port 80, FTP uses port 21).
4. **Forward Data:** The data is forwarded to the appropriate application (e.g., web browser, FTP client).
5. **Processing:** The application receives and processes the data (e.g., displaying a webpage, downloading a file).

Outcome: Demultiplexing ensures that data sent by different applications over the same transport connection is delivered to the correct application.