Contents

Slide Number: 1

Title: Introduction to the Transport Layer

Content Explanation:

The transport layer is responsible for **logical communication** between application processes running on different hosts. It serves as a bridge between the network layer and the application layer by breaking down messages from applications into smaller units called **segments** and sending them across the network.

Key Functions of the Transport Layer:

1. **Multiplexing and Demultiplexing:** Enables multiple processes to share the network connection by identifying data streams using ports.
2. **Reliable Data Transfer:** Ensures data is delivered accurately and in order, particularly in protocols like TCP.
3. **Flow Control:** Prevents sender applications from overwhelming the receiver by controlling the data transmission rate.
4. **Congestion Control:** Regulates traffic to prevent network congestion.

Protocols in the Transport Layer:

1. **TCP (Transmission Control Protocol):** Provides reliable, connection-oriented communication with features like error checking and retransmission.
2. **UDP (User Datagram Protocol):** A connectionless, lightweight protocol offering faster but less reliable communication.

Simple Explanation:

The transport layer is like a **delivery service** ensuring your message is packed properly, delivered to the right person, and handled if anything goes wrong during transit.

- If you want to be sure the package reaches safely and in order, you use **TCP**.
- If you want a quick delivery without checking much, you use **UDP**.

Examples:

1. **TCP Example:** When you send an email or access a website, you need reliable and in-order communication. TCP is used here.
2. **UDP Example:** For online video calls or live streaming, speed matters more than reliability. UDP is used in these cases.

Analogy:

The transport layer is like a **postal service:**

- TCP is a registered mail where you get confirmation upon delivery.
- UDP is like a regular post where you send it without tracking.

Slide Number: 2

Title: Transport Layer Services

Content Explanation:

The transport layer provides **core services** that enable communication between application processes on different devices. Two key services are **multiplexing** and **demultiplexing**.

1. Multiplexing and Demultiplexing:

- **Multiplexing:** At the sender's side, the transport layer collects data from multiple applications and combines them into segments. It uses **source port numbers** to identify which application the data came from.
- **Demultiplexing:** At the receiver's side, the transport layer examines the **destination port number** in each segment and directs the data to the correct application.

How it works:

- Each segment contains **port numbers** (source and destination) and **IP addresses**.
- The transport layer uses these details to ensure data reaches the correct **socket**, a unique endpoint for communication.

---

2. Distinction Between Transport and Network Layer Services:

- **Transport Layer Services:** Logical communication between **processes**. It uses ports to differentiate applications.
  - Example: Ensures a web browser and a video call app on the same machine receive their respective data.
- **Network Layer Services:** Logical communication between **hosts**. It identifies devices using IP addresses but does not differentiate between processes or applications.

Aspect	Transport Layer	Network Layer
Unit of Communication	Processes (applications)	Hosts (devices)
Identifiers	Port numbers	IP addresses

Aspect	Transport Layer	Network Layer
Protocols	TCP, UDP	IP
Responsibility	End-to-end communication	Packet routing

Simple Explanation:

- **Multiplexing** is like gathering multiple letters (data) from different people (applications) and putting them in separate envelopes with unique sender addresses (port numbers).
- **Demultiplexing** is like receiving a bundle of envelopes and distributing them to the correct recipients based on their addresses.

Examples:

1. **Multiplexing Example:**
   Sending a WhatsApp message and a file upload at the same time—both data streams are uniquely identified and packaged by the transport layer.
2. **Demultiplexing Example:**
   Receiving emails and browsing the web simultaneously—each data stream is directed to its correct application.

Analogy:

Imagine a **train station**:

- Multiplexing: Multiple trains (data streams) depart from the station, each with a unique train number (port number) and destination (IP address).
- Demultiplexing: At another station, each train is directed to the right platform (application) based on its number.

Slide Number: 3

Title: Connectionless Transport: UDP

Content Explanation:

UDP (User Datagram Protocol) is a **connectionless transport-layer protocol** known for its simplicity and efficiency. It provides basic transport-layer functions with minimal overhead.

1. Features of UDP:

- **Connectionless Communication:**
  - No need for establishing a connection before data transfer.

- Each segment is handled independently.
- **No Reliability:**
  - UDP does not guarantee delivery, order, or error recovery.
- **No Flow or Congestion Control:**
  - It assumes the application will handle such requirements, making UDP lightweight.
- **Low Overhead:**
  - UDP's simplicity makes it faster than TCP for applications that don't require reliability.

---

2. Use Cases of UDP:

- **DNS (Domain Name System):**
  - Fast, simple queries for domain name-to-IP address resolution.
- **SNMP (Simple Network Management Protocol):**
  - Used for monitoring and managing network devices.
- **Multimedia Applications:**
  - Live streaming and voice/video calls prioritize speed over reliability.

---

3. UDP Segment Structure and Checksum:

UDP segments consist of four main fields:

Field	Size (bits)	Description
Source Port	16	Identifies the sending application.
Destination Port	16	Identifies the receiving application.
Length	16	Total size of the UDP segment, including the header.
Checksum	16	Detects errors in the segment.
Payload	Variable	Contains the application data.

- **Checksum Functionality:**
  - Ensures integrity by detecting errors in the transmitted data.
  - Sender calculates the checksum based on segment content and adds it to the header.
  - Receiver recalculates the checksum to verify data integrity.

Simple Explanation:

UDP is like sending a **postcard**:

- You don't confirm if the recipient gets it.
- The message is simple and fast.
- You add a quick check (checksum) to ensure the content isn't damaged, but that's all.

Examples:

1. **DNS:**
   When you type "google.com" in your browser, UDP quickly resolves the domain name to an IP address.
2. **Multimedia:**
   Video streaming on YouTube uses UDP to ensure smooth playback, even if some packets are lost.
3. **SNMP:**
   Network administrators monitor router performance using SNMP over UDP.

Analogy:

UDP is like a **courier service** that delivers packages without tracking.

- It's fast and works well when speed matters more than ensuring the package arrives.

Title: Principles of Reliable Data Transfer

Content Explanation:

Reliable data transfer ensures that data is delivered **accurately**, **in sequence**, and **without loss or corruption** despite challenges like bit errors, packet loss, and network delays. It is a cornerstone of the transport layer, especially in protocols like TCP.

---

1. Overview of Reliable Data Transfer Mechanisms

- **Goal:** Ensure reliable communication over an unreliable network.
- **Key Mechanisms:**
  - **Error Detection:** Using checksums to detect corrupted data.
  - **Acknowledgements (ACKs):** Sender confirms receipt of data.
  - **Negative Acknowledgements (NAKs):** Receiver informs sender of corrupted or missing data.
  - **Retransmission:** Resending data when errors or losses occur.

---

2. Development of Reliable Data Transfer Protocols

Reliable data transfer protocols are developed incrementally, addressing increasing levels of network unreliability.

---

a. rdt1.0: Reliable Channel

- **Assumptions:**
  - The underlying channel is completely reliable (no errors or loss).
  - No special error-handling mechanisms are needed.
- **Process:**
  - The sender simply transmits data.
  - The receiver reads data and delivers it to the application layer.
- **Limitation:**
  - Unrealistic, as real-world channels are prone to errors and packet loss.

---

b. rdt2.0: Channel with Bit Errors

- **Assumptions:**
  - The channel may corrupt data (bit errors).
  - Acknowledgements (ACKs) and Negative Acknowledgements (NAKs) are used to ensure reliability.
- **Mechanisms:**
  - **Checksum:** Used to detect errors in transmitted data.
  - **ACK/NAK:** Receiver sends an ACK if the data is correct or a NAK if errors are detected.
  - On receiving a NAK, the sender retransmits the data.
- **Challenge:**
  - If ACK/NAK packets themselves are corrupted, the protocol fails to determine the next step.

---

c. rdt2.1: Handling Corrupted ACK/NAK

- **Improvements Over rdt2.0:**
  - Adds sequence numbers to distinguish new data from retransmitted data.
  - The sender maintains the current state (sequence number) to avoid delivering duplicate data.
  - The receiver discards duplicate packets by checking sequence numbers.
- **Mechanisms:**
  - If the receiver detects a corrupted packet, it sends the last correctly received ACK.
  - The sender retransmits data based on the sequence number.

---

d. rdt2.2: NAK-Free Protocol

- **Modification:**
  - Eliminates NAKs; the receiver sends duplicate ACKs instead to indicate issues.
  - This approach simplifies the protocol and reduces packet types.

---

e. rdt3.0: Channels with Errors and Packet Loss

- **Assumptions:**
  - The channel may lose packets entirely (data or ACKs).
  - Corruption is still possible.
- **Mechanisms:**
  - Implements a **timeout mechanism**:
    - The sender starts a timer after transmitting a packet.
    - If no ACK is received before the timeout, the packet is retransmitted.
  - Sequence numbers handle duplicates due to retransmissions.
  - Uses a **countdown timer** to determine when to retransmit data.
- **Challenge:**
  - Poor performance due to waiting for each ACK, especially on high-latency links.

---

3. Performance Issues in Stop-and-Wait Protocols

- In stop-and-wait, the sender must wait for an ACK before sending the next packet.

- **Utilization Formula**:

$$U_{\text{sender}} = \frac{L/R}{RTT + L/R}$$

  - L: Packet size in bits.
  - R: Transmission rate.
  - RTT: Round-Trip Time.

Problem:

  - Performance decreases drastically as RTT increases or the transmission rate is high.

---

4. Introduction to Pipelining

To improve performance, **pipelining** allows multiple packets to be "in-flight" (sent but not acknowledged). This reduces idle time for the sender.

---

a. Go-Back-N (GBN) Protocol

- The sender can transmit up to NNN packets without receiving an ACK.

- The receiver sends cumulative ACKs, acknowledging all packets up to a certain point.
- **On Timeout:** The sender retransmits all packets from the last unacknowledged one.

---

b. Selective Repeat (SR) Protocol

- The sender retransmits only the specific packets that were not acknowledged.
- The receiver buffers out-of-order packets until missing ones arrive.

Simple Explanation:

Reliable data transfer is like sending a **certified letter**:

1. You add tracking (checksum) to ensure it isn't damaged.
2. The recipient acknowledges receipt (ACK).
3. If something goes wrong (NAK or timeout), you resend the letter.

Examples:

1. **rdt1.0:** A perfect courier service where all packages arrive intact.
2. **rdt2.0:** Packages may be damaged; the recipient informs you to resend.
3. **rdt3.0:** Packages can also get lost, so you need to resend if you don't hear back in time.
4. **Pipelining:** Sending multiple packages at once instead of waiting for confirmation for each one.

Analogy:

Imagine sending books to a library:

- **Stop-and-Wait:** You send one book and wait for confirmation before sending another.
- **Pipelining:** You send multiple books, and the library informs you of missing or damaged ones.

Slide Number: 5

Title: Pipelined Protocols

Content Explanation:

Pipelined protocols allow the sender to transmit multiple packets without waiting for an acknowledgment (ACK) for each one. This approach improves **utilization** of the network, especially in high-latency scenarios.

---

1. Go-Back-N (GBN) Protocol:

- **Sender Operations:**
  - Maintains a **window of size NNN** for unacknowledged packets.
  - **Cumulative ACKs:** The receiver acknowledges all packets up to the last correctly received packet.
  - On **timeout:** The sender retransmits all packets starting from the last unacknowledged one.
  - Uses a **single timer** for the oldest unacknowledged packet.
- **Receiver Operations:**
  - Only processes packets in order.
  - Discards out-of-order packets.
  - Sends an ACK for the last correctly received in-order packet.

---

2. Selective Repeat (SR) Protocol:

- **Sender Operations:**
  - Maintains a **window of size NNN**.
  - Each packet has its own **timer** for retransmission.
  - Retransmits only the specific packets that were not acknowledged.
- **Receiver Operations:**
  - Buffers out-of-order packets until missing packets arrive.
  - Sends an ACK for each correctly received packet.

Feature	Go-Back-N (GBN)	Selective Repeat (SR)
Handling of Errors	Retransmits all unacknowledged packets.	Retransmits only the specific missing ones.
Receiver Buffering	No buffering of out-of-order packets.	Buffers out-of-order packets.
Timers	One timer for the oldest unacknowledged packet.	Individual timers for each packet.

Simple Explanation:

1. **GBN:** A strict teacher who asks the entire class to redo an assignment if one student makes a mistake.
2. **SR:** A lenient teacher who asks only the students with mistakes to redo their work.

Examples:

1. **GBN:** If packets 1-4 are sent and packet 3 is lost, the sender retransmits packets 3 and 4.
2. **SR:** If packet 3 is lost, the sender retransmits only packet 3.

Analogy:

Pipelined protocols are like **conveyor belts:**

- **GBN:** If one item falls off, the entire section is checked and restarted.
- **SR:** Only the missing item is replaced while others move forward.

Slide Number: 6

Title: Connection-Oriented Transport: TCP

Content Explanation:

TCP (Transmission Control Protocol) is a **connection-oriented** protocol designed for reliable communication. It provides features like in-order delivery, flow control, and congestion control.

---

1. Features of TCP:

- **Connection Establishment (Three-Way Handshake):**
  - **Step 1:** The client sends a SYN (synchronize) segment to the server.
  - **Step 2:** The server replies with a SYN-ACK (synchronize acknowledgment).
  - **Step 3:** The client responds with an ACK, and the connection is established.
  - This process ensures both parties are ready for communication.
- **Reliable, In-Order Delivery:**
  - TCP uses **sequence numbers** to identify the order of bytes in a data stream.
  - **ACKs** confirm successful receipt of data.
  - Lost or corrupted packets are retransmitted.
- **Flow Control:**
  - Ensures the sender does not overwhelm the receiver's buffer.
  - TCP uses a **sliding window mechanism** where the receiver advertises its available buffer space (window size) in the header.
- **Congestion Control:**
  - Adjusts the sender's data rate based on network congestion.

---

2. TCP Segment Structure:

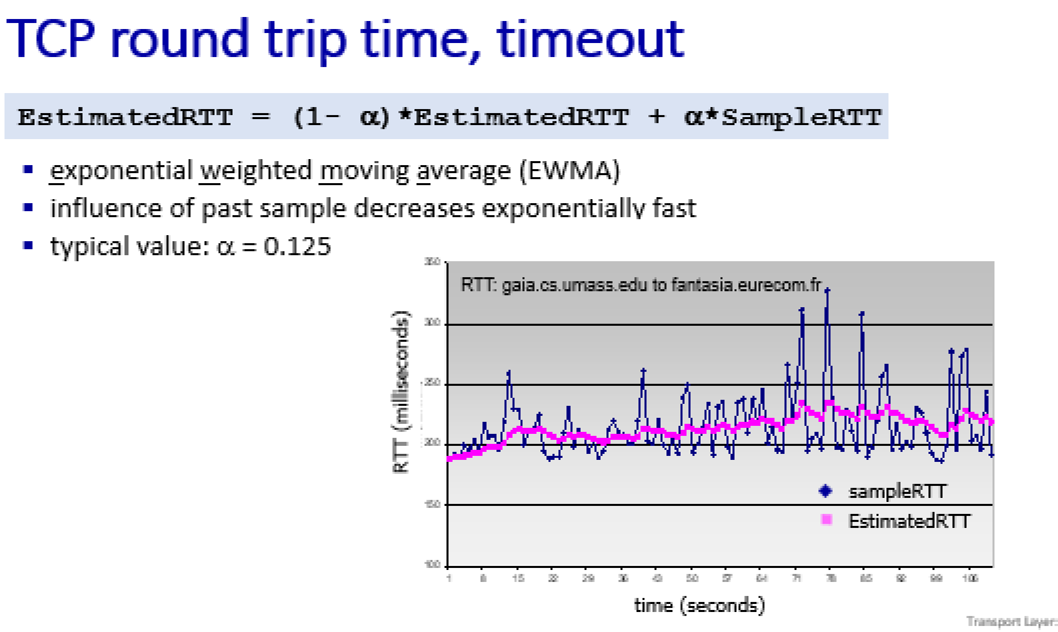
Field	Size (bits)	Description
Source Port	16	Identifies the sending application.
Destination Port	16	Identifies the receiving application.
Sequence Number	32	Position of the first byte in the segment.

Acknowledgment Number	32	Next expected byte from the sender.
Header Length	4	Size of the TCP header.
Flags	6	Control flags (e.g., SYN, ACK, FIN).
Window Size	16	Indicates the available buffer space at the receiver.
Checksum	16	Used for error detection.
Urgent Pointer	16	Points to urgent data (if any).

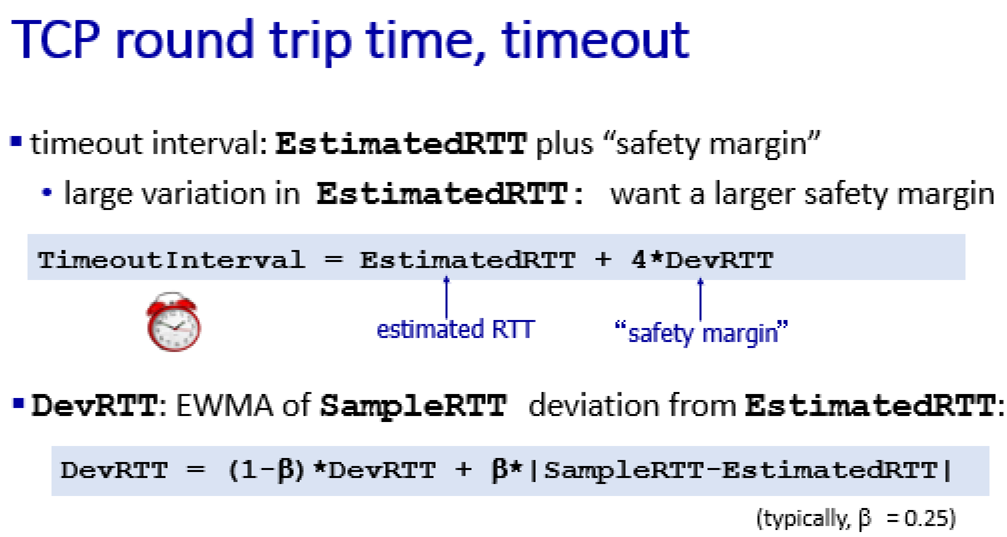
---

3. Round-Trip Time (RTT) Estimation:

- TCP dynamically adjusts its **timeout interval** to account for changing RTTs.
- **RTT Calculation:**



- **Timeout Interval:**



Simple Explanation:

TCP is like a **conversation** where:

1. You greet the other person (three-way handshake).
2. You take turns speaking (sliding window).
3. If the other person doesn't respond, you repeat yourself (retransmission).

Examples:

1. **Three-Way Handshake:** Establishing a connection for a video call.
2. **Sliding Window:** A sender pauses sending data if the receiver's buffer is full.

Analogy:

TCP is like a **registered mail service**:

- You ensure the recipient acknowledges receipt.
- You resend lost items, and you avoid overwhelming the recipient with too many packages.

Slide Number: 7

Title: TCP Congestion Control

Content Explanation:

TCP uses **congestion control mechanisms** to prevent overwhelming the network and ensure efficient data transmission.

1. Principles of Congestion Control:

- **Definition:** Congestion occurs when too much data is sent too quickly, causing delays, packet loss, and degraded performance.
- **Congestion Control Goals:**
  - Avoid overloading the network.
  - Ensure fair bandwidth allocation among multiple connections.

2. Algorithms Used in TCP:

a. Additive Increase, Multiplicative Decrease (AIMD):

- **Principle:** TCP probes for available bandwidth while reacting to network congestion.
- **Steps:**
  - **Additive Increase:** TCP gradually increases its sending rate (by 1 MSS per RTT) to probe for additional bandwidth.
  - **Multiplicative Decrease:** Upon detecting congestion (via packet loss), TCP cuts its congestion window size by half.
- **Behavior:** The **sawtooth pattern** where throughput increases linearly and drops sharply upon congestion.

b. Slow Start:

- **Purpose:** Quickly determine the available bandwidth at the start of a connection.
- **Mechanism:**
  - Starts with a small congestion window (1 MSS).
  - Doubles the congestion window every RTT until congestion is detected or it reaches the **slow start threshold (ssthresh)**.
- **Transition:** Switches to congestion avoidance once the window size exceeds ssthreshssthreshssthresh.

c. Congestion Avoidance:

- **Mechanism:** After slow start, the congestion window grows linearly to avoid congestion.
- **Behavior:**
  - Adds 1 MSS per RTT to the congestion window.
  - Reduces the sending rate when congestion is detected.

---

d. Modern Approaches:

- **TCP CUBIC:**
  - **Purpose:** Optimized for high-speed networks.
  - **Mechanism:**
    - Uses a cubic function to adjust the congestion window.
    - Increases rapidly when far from the congestion point and slows as it approaches.
  - **Benefits:** Provides better throughput in high-latency and high-bandwidth scenarios.
  - **Default in Linux:** TCP CUBIC is widely used in modern web servers and applications.

Simple Explanation:

TCP is like a driver on a highway:

- **Slow Start:** Start slow to assess the traffic.
- **AIMD:** Accelerate gradually but slow down sharply if you hit a traffic jam.
- **Congestion Avoidance:** Drive steadily when you're at a safe speed.
- **TCP CUBIC:** Adjust speed dynamically based on road and traffic conditions.

Examples:

1. **AIMD:** A file upload increases speed gradually, but when network congestion is detected, the speed is halved.
2. **TCP CUBIC:** Streaming high-definition video adapts dynamically to network conditions.

Analogy:

TCP congestion control is like a **water pipe**:

- AIMD ensures the flow increases slowly to avoid bursts.
- Slow start is like initially opening the faucet slightly and gradually increasing it.
- TCP CUBIC adjusts the flow dynamically based on pipe size and water pressure.

Slide Number: 8

Title: Advanced Topics in Transport Layer

Content Explanation:

---

1. Differences Between TCP and UDP:

Feature	TCP	UDP
Connection	Connection-oriented (requires handshake).	Connectionless (no handshake).
Reliability	Guarantees reliable delivery.	No reliability guarantees.
Data Ordering	Ensures in-order delivery.	No ordering of data.
Error Handling	Error detection and recovery.	Error detection only.
Use Cases	Web browsing, emails, file transfers.	Streaming, DNS, VoIP, gaming.

---

2. Introduction to QUIC (Quick UDP Internet Connections):

- **Purpose:** Designed to improve latency and reliability for modern web applications.
- **Key Features:**
  - Built on top of UDP but provides reliability and congestion control similar to TCP.
  - Reduces connection establishment time with a **single handshake** (integrates transport and security setup).
  - Multiplexes multiple streams over one connection, eliminating head-of-line blocking.
  - **Use Cases:** QUIC powers HTTP/3, used in applications like Google Chrome and YouTube.

---

3. Evolution of Transport-Layer Protocols for High-Speed Networks:

- Traditional transport protocols (like TCP) struggle with:
  - High-latency links.
  - Large bandwidth-delay products.
  - Frequent packet loss in wireless networks.
- **Modern Adaptations:**
  - **TCP Variants:** CUBIC, BBR (Bottleneck Bandwidth and RTT).
  - **QUIC:** Focuses on low latency and optimized performance for high-speed and high-latency networks.
  - **SCTP (Stream Control Transmission Protocol):** Supports multi-streaming and multi-homing for robust performance.

Simple Explanation:

1. **TCP vs. UDP:** TCP is reliable and ordered; UDP is faster but riskier.
2. **QUIC:** Combines the speed of UDP with the reliability of TCP.
3. **Modern protocols:** Tailored to handle today's high-speed networks and demanding applications.

Examples:

1. **TCP Use Case:** Downloading a file from the internet.
2. **UDP Use Case:** Watching live sports.
3. **QUIC Use Case:** Browsing modern websites powered by HTTP/3.

Analogy:

1. **TCP vs. UDP:** TCP is like sending a tracked package; UDP is like sending a regular letter.
2. **QUIC:** It's like sending a tracked package with instant setup and multiple compartments for different items.