Okay, let's move on to Chapter 6.

Chapter 6: The Transport Layer (Layer 4)

Note: The provided text indicates this chapter is incomplete and intended for completion in 2025 or beyond. The following notes are based on the content available as of May 2025.
The Transport Layer (Layer 4) acts as a "pipe" between communicating processes on different hosts across a network. Its primary role is to hide the complexities of the underlying network (Layers 1-3) from the application-oriented layers (Layers 5-7) and vice-versa.

Key Functions and Concepts:

1. Introduction

- **Process Identification (Addressing):**
  - Uses **ports** to identify the specific processes at the endpoints of the communication pipe.
  - **Well-known ports:** Standardized port numbers for common applications (e.g., port 80 for HTTP). A server registers with the OS to receive data on a specific port.
  - **Client-side ports:** A client typically requests an unused (dynamic) port from its OS to use as the source port. The server uses this port as the destination port for its replies.
- **Reliability of the Pipe:**
  - A key design decision is how reliable the communication should be.
  - **TCP (Transmission Control Protocol):** Designed for reliable transport.
  - **UDP (User Datagram Protocol):** Provides unreliable transport.
  - **QUIC (Quick UDP Internet Connections):** A newer protocol that provides reliable transport, often seen as an evolution of TCP, but it runs on top of UDP. It aims to be faster and incorporates security.

2. Reliability

Reliability in the transport layer context means:

1. Communication should ideally not fail.
2. If it does fail, the communicating parties should be made aware of the failure.

- **Potential Communication Failures:**
  - Data corruption (changes during transit).
  - Data loss (never arrives).
  - Data arrival rate too high for the receiver.
  - Data arriving out of sequence.
  - Duplicate data arrival.
- **Connection-Oriented Communication (e.g., TCP, QUIC):**
  - Establishes a connection, uses it for communication, and then terminates it.
  - Analogous to a telephone call.

    - **Reliable Connection Establishment and Termination (Handshakes):**
        - **Three-Way Handshake (TCP Example):**
            1. A -> B: SYN (Synchronize - Are you ready?)
            2. B -> A: SYN+ACK (Yes, I'm ready. Are you ready?)
            3. A -> B: ACK (Yes, I'm ready.)
        - **Termination (TCP Example - similar three-way process):**
            1. A -> B: FIN (Finish - Request to terminate)
            2. B -> A: FIN+ACK (Acknowledge termination request, also request to terminate from B's side)
            3. A -> B: ACK (Acknowledge B's termination request)
- **Connectionless Communication (e.g., UDP):**
    - No prior connection establishment or termination.
    - Analogous to the postal system (snail mail). Sender doesn't know if the recipient is ready or if the message will arrive.
- **Reliable Message Interchange Mechanisms:**
    - **Buffers:** Used at sender (to resend lost/damaged data) and receiver (to store incoming data, reorder out-of-sequence data, hold data if the application isn't ready).
    - **Flow Control:** Prevents sender from overwhelming the receiver or the network.
        - **Stop-and-Wait:** Sender sends one message, stops, and waits for an acknowledgement (ACK). If no ACK within a timeout, resends. Inefficient due to waiting times.
        - **Sliding Window:** Allows sender to transmit multiple messages (the "window") before receiving ACKs.
            - Acknowledged messages are removed from the sender's buffer and the window "slides" forward.
            - Numbered messages allow detection of duplicates and out-of-sequence arrivals.
            - **Go-Back-N ARQ:** If message 'N' is lost or NAKed (Negative Acknowledgement), sender resends message 'N' and all subsequent messages.
            - **Selective Repeat ARQ:** If message 'N' is lost or NAKed, sender only resends message 'N'. Receiver buffers out-of-order correct packets. TCP uses a variation of this.
        - **Unrestricted Flow:** No flow control is enforced (as in UDP).
        - **ARQ (Automatic Repeat reQuest):** General term for protocols using timers and ACKs for retransmission.
            - Requires packet timers (retransmit if expired) and acknowledgement timers (send ACK if no outgoing data to piggyback on).
    - **Piggybacking:** ACKs are often included in data messages going in the reverse direction to improve efficiency.
- **Reducing Transport Errors (Congestion Control):**
    - Transport errors can be caused by network congestion (too much traffic).

    Retransmitting data can worsen congestion.
  - **Congestion Control Strategy:**
    1. Decrease sending rate when errors (packet loss) occur.
    2. Increase sending rate cautiously when no errors occur.
    3. Aim for fair sharing of network resources in a decentralized manner.
  - Packet rates are often controlled by adjusting the **size of the sliding window** (or a separate congestion window). If a packet is lost, the window size might be reduced (e.g., halved).

3. Layer 4 Design Choices

- The choice isn't strictly binary (reliable vs. unreliable). The underlying network's reliability influences the checks needed at Layer 4.
- **ISO OSI Transport Protocol Classes (TP0-TP4):**
  - A range of protocols offering different levels of error control and features.
  - **TP0:** Minimal error control (assumes reliable network).
  - **TP4:** Assumes unreliable network and provides comprehensive reliability.
  - Features varied across classes: connection-oriented/less, resynchronization, explicit flow control, retransmission, checksums. (See Table 6.1 in the book for a summary).
  - These TPs are rarely used in practice; TCP, UDP, and QUIC dominate.

4. Ports

- Mechanisms to match incoming transport layer requests with the correct application process.
- A 16-bit number.
- **Port Number Ranges:**
  - **Well-Known Ports (0 to 1023):** For standard, well-known applications (e.g., HTTP: 80, FTP: 20/21, SMTP: 25, HTTPS: 443, POP3: 110). Managed by IANA.
  - **Registered Ports (1024 to 49151):** Can be registered for specific applications. Can be used by other applications if the registered one isn't running.
  - **Dynamic/Private Ports (49152 to 65535):** Cannot be registered. Used on an ad hoc basis, typically by clients for their source port.
- **Socket:** The combination of an IP address and a port number (e.g., xx.co.za:80).

5. TCP (Transmission Control Protocol)

- Reliable, connection-oriented protocol.
- TCP messages are called **segments**.
- **TCP Header (Figure 6.3):**
  - **Source Port (16 bits), Destination Port (16 bits):** Identify communicating processes.
  - **Sequence Number (32 bits):** Byte stream number of the first byte in this segment's data. Used for ordering and acknowledgements.
  - **Acknowledgement Number (32 bits):** If ACK flag is set, this is the sequence

    number of the *next* byte the sender of the ACK is expecting.
    - **Data Offset (4 bits):** Length of the TCP header in 32-bit words (minimum 5, for a 20-byte header).
    - **Reserved (6 bits):** For future use, should be zero.
    - **Flags (6 bits):**
        - URG: Urgent Pointer field is significant.
        - ACK: Acknowledgement field is significant.
        - PSH: Push function (data should be passed to application ASAP).
        - RST: Reset the connection.
        - SYN: Synchronize sequence numbers (used in connection setup).
        - FIN: No more data from sender (used in connection termination).
    - **Window (16 bits):** Number of data bytes the sender of this segment is willing to accept (flow control).
    - **Checksum (16 bits):** For error checking of header and data.
    - **Urgent Pointer (16 bits):** If URG flag is set, points to the byte following urgent data.
    - **Options (variable length):** e.g., Maximum Segment Size (MSS).
    - **Padding (variable length):** Ensures header ends on a 32-bit boundary.
- **TCP Flow Control:**
    - Uses sequence numbers (byte-oriented, not segment-oriented), acknowledgement numbers, and the window field.
    - **Example (Figure 6.4):**
        - SYN and FIN messages consume one sequence number even if they carry no data.
        - If segment has Seq=50 and 100 bytes of data, it contains bytes 50-149. Next segment will have Seq=150. ACK for this would be Ack=150.
        - Window size decreases as data is received into the buffer, and increases as the application consumes data from the buffer.
- **TCP Timers:**
    - **Retransmission Timer (RTO Timer):** Started when a segment is sent. If it expires before an ACK is received, the segment is assumed lost and retransmitted.
        - RTO value is dynamically calculated based on measured Round-Trip Times (RTTm).
    - **Persistence Timer:** Prevents deadlock if a receiver advertises a zero window and the subsequent window update is lost. If the timer expires, the sender sends a "probe" segment (1 byte, not part of data stream) to elicit an ACK with the current window size.
    - **Keepalive Timer:** Detects inactive or broken connections. If no data received for a long period (e.g., 2 hours), probes are sent. If no response, connection is abandoned.
    - **Quiet Timer (TIME-WAIT state timer):** After a connection closes, the endpoint that initiated the final ACK enters TIME-WAIT for twice the Maximum Segment Lifetime (2MSL). This ensures all segments from the old connection die off before

the port can be reused, preventing old duplicate segments from interfering with new connections.

- **TCP States (Figure 6.5 & 6.8 - State Diagram):**
  - CLOSED: No connection.
  - LISTEN: Server waiting for a connection request.
  - SYN-SENT: Client has sent SYN, waiting for SYN+ACK.
  - SYN-RECEIVED: Server has sent SYN+ACK, waiting for ACK.
  - ESTABLISHED: Connection is open, data can be transferred.
  - FIN-WAIT-1: Sent FIN, waiting for ACK or FIN from peer.
  - FIN-WAIT-2: Received ACK for its FIN, waiting for FIN from peer.
  - CLOSE-WAIT: Received FIN, local application is closing, will send FIN.
  - CLOSING: Sent FIN, received FIN, waiting for ACK of its FIN.
  - LAST-ACK: Sent FIN (after receiving FIN and local app closed), waiting for final ACK.
  - TIME-WAIT: Waiting for 2MSL after sending final ACK to ensure peer received it.
  - The netstat command can display TCP connection states (Figures 6.6, 6.7).
- **TCP Congestion Control:**
  - Aims to prevent network congestion by adjusting the sending rate.
  - **Congestion Window (cwnd):** Sender-side limit on the number of segments (or bytes) that can be in flight, independent of the receiver's advertised window.
  - **Slow Start:**
    1. cwnd typically starts small (e.g., 1 or 2 MSS).
    2. For each ACK received, cwnd increases (e.g., by 1 MSS). This leads to exponential growth of cwnd initially (1 -> 2 -> 4 -> 8...).
  - **Congestion Avoidance:** When packet loss occurs (signaling congestion):
    1. A threshold (ssthresh) is often set to cwnd/2.
    2. cwnd is reset (e.g., to 1 MSS).
    3. Slow start restarts.
    4. When cwnd reaches ssthresh, growth becomes linear (additive increase, e.g., cwnd increases by 1 MSS per RTT) to probe for more bandwidth more cautiously.
  - **Fast Retransmit/Fast Recovery:** Optimizations to avoid slow start after isolated packet losses (e.g., if multiple duplicate ACKs are received).

6. UDP (User Datagram Protocol)

- Connectionless, unreliable, best-effort transport protocol.
- No ACKs, no sequence numbers, no flow control, no congestion control provided by UDP itself.
- **UDP Header (Figure 6.9):** Very simple.
  - **Source Port (16 bits)**
  - **Destination Port (16 bits)**
  - **Length (16 bits):** Length of UDP header + UDP data in bytes. Minimum 8 bytes (header only).

    - **Checksum (16 bits):** Optional (can be all zeros if unused, though strongly recommended). Covers UDP header, UDP data, and a "pseudo-header" (containing IP addresses, protocol, UDP length from IP header).
- **Use Cases:**
    - Applications that don't need reliability or where TCP's overhead is too high (e.g., DNS lookups - simpler to resend query if no reply).
    - Streaming media where occasional loss is tolerable and retransmission delays are unacceptable.
    - Foundation for other protocols like QUIC.

7. QUIC (Quick UDP Internet Connections)

- Relatively new (standardized RFC 9000 in 2022, v2 in Dec 2023) reliable, connection-oriented transport protocol.
- Aims to be faster than TCP+TLS and more robust against network changes.
- **Key Characteristics:**
    - **Runs on UDP:** Uses UDP as the underlying datagram transport. Can be seen as Layer 4.1.
    - **Built-in Security:** Integrates TLS 1.3 (or later) encryption. No cleartext version.
    - **Multiplexed Connections via Streams:** A single QUIC connection can support multiple independent, parallel streams.
        - Reduces head-of-line blocking: Packet loss on one stream doesn't necessarily stall others.
        - Streams can be client-initiated or server-initiated, unidirectional or bidirectional.
    - **Connection IDs:**
        - Identifies connections independently of IP address and port.
        - Allows **connection migration** (e.g., mobile device switching from WiFi to cellular) without dropping the connection. Client sends a UDP datagram from new IP/port; QUIC layer at server notices and updates. Token mechanism authenticates migration. (Only client migration in v1/v2).
    - **Reduced Latency:** Aims for 0-RTT or 1-RTT connection setup for application data.
- **Cryptography Primer (relevant to QUIC):**
    - **Symmetric Encryption:** Same key for encryption and decryption. Fast, but key distribution is an issue.
    - **Public-Key (Asymmetric) Encryption:** Public key for encryption, private key for decryption (or vice-versa for digital signatures). Solves key distribution but is computationally slower.
    - **Digital Certificates & CAs (Certificate Authorities):** Bind public keys to identities.
    - **Session Keys:** Typically, public-key crypto is used to establish a secure channel, then a symmetric session key is negotiated/exchanged for faster bulk data encryption.

- **X25519 (Elliptic Curve Diffie-Hellman):** A key exchange mechanism where parties generate ephemeral key pairs. Each party combines its private key with the other's public key to arrive at the same shared secret, which then forms the basis for session keys. Used in TLS 1.3.
- QUIC mandates TLS 1.3+ and requires server authentication (client authenticates server via certificate).

- **QUIC Data Units:**
  - QUIC sends **packets** (which are payloads of UDP datagrams).
  - QUIC packets contain **frames**. A frame is a unit of data with a specific type and purpose.
  - **Packet Headers:**
    - **Long Header (Figure 6.10):** Used during connection establishment (Initial, 0-RTT, Handshake, Retry packets) and for Version Negotiation.
      - Header Form (1 bit): 1 for long.
      - Fixed Bit (1 bit): Always 1.
      - Packet Type (2 bits): e.g., Initial (QUICv1: 00, QUICv2: 01), Handshake (QUICv1: 10, QUICv2: 11).
      - Type-Specific Bits (4 bits).
      - Version (32 bits): e.g., 0x00000001 for v1, 0x6b3343cf for v2 (to combat ossification).
      - Destination Connection ID Length (8 bits) + ID (0-160 bits).
      - Source Connection ID Length (8 bits) + ID (0-160 bits).
      - Initial, Handshake, 0-RTT also have Packet Number (8-32 bits) and Length field (for packet number + payload).
    - **Short Header (Figure 6.11):** Used for 1-RTT (data) packets once connection is established.
      - Header Form (1 bit): 0 for short.
      - Fixed Bit (1 bit): Always 1.
      - Spin Bit, Reserved, Key Phase bits.
      - Packet Number Length (2 bits): Encodes length of Packet Number field.
      - Destination Connection ID (0-160 bits, length implied or from handshake).
      - Packet Number (8-32 bits).
- **Establishing QUIC Connections (Figures 6.12, 6.13):**
  - Combines transport handshake and cryptographic handshake (TLS 1.3).
  - **Packet Number Spaces:** Initial, Handshake, and Application Data packets have separate numbering spaces (each can start from 0).
  - **Protection Levels:** Initial packets have minimal protection, Handshake packets more, Application Data packets full TLS protection.
  - **Process (Simplified 1-RTT setup):**
    1. **Client -> Server (Initial Packet(s), t0):**
       - Contains CRYPTO frame(s) with TLS client hello (cipher suites, client's

        ephemeral public key for key exchange).
        - May also contain 0-RTT data if resuming a session.
    2. **Server -> Client (Initial + Handshake Packet(s), t1):**
        - Initial packet acknowledges client's initial.
        - Handshake packet(s) contain CRYPTO frame(s) with TLS server hello (selected cipher, server's ephemeral public key), server's certificate, signed data to authenticate server, TLS "finished" message.
        - Server can now derive session keys and send 1-RTT application data to client.
    3. **Client -> Server (Initial + Handshake Packet(s), t2):**
        - Initial packet acknowledges server's initial.
        - Handshake packet(s) contain CRYPTO frame(s) with client's TLS "finished" message.
        - Client can now derive session keys, decrypt any 1-RTT data from server, and send its own 1-RTT data.
  - **0-RTT Data:** Client sends early application data with its first flight of Initial packets, encrypted with session keys from a previous connection (if available and server supports it). Only for idempotent requests (e.g., GET, not POST that transfers money). HTTP status 425 (Too Early) may be used if server can't process 0-RTT data.
- **QUIC Frames (Examples):**
  - PING (Type 0x01): Confirms reachability.
  - ACK (Types 0x02, 0x03): Acknowledges received packets, can specify multiple ranges of received packet numbers and delays.
  - STREAM (Types 0x08-0x0f): Carries application data for a specific stream.
    - Fields: Type (encodes flags: Offset present, Length present, Final frame for stream), Stream ID, Offset (optional), Length (optional), Stream Data.
    - **Stream ID Encoding for Bidirectional/Unidirectional & Client/Server Initiated:**
      - Last bit: 0 for client-initiated, 1 for server-initiated.
      - Second to last bit: 0 for bidirectional, 1 for unidirectional.
      - **Example: Client-initiated bidirectional streams:** IDs are 0,4,8,12,... (general form 4n).
        - Stream 0: xxxx...xx00
        - Stream 4: xxxx...0100
  - CRYPTO (Type 0x06): Carries TLS handshake messages. Similar to STREAM but for cryptographic data.
  - **Variable-Length Integers:** Used for many QUIC fields (Type, Stream ID, Offset, Length). First 2 bits indicate length of the integer:
    - 00xxxxxx: 1-byte integer (6 data bits). Max value 26−1=63.
    - 01xxxxxx xxxxxxxx: 2-byte integer (14 data bits). Max value 214−1=16383.
    - 10xxxxxx ...: 4-byte integer (30 data bits). Max value 230−1≈109.
    - 11xxxxxx ...: 8-byte integer (62 data bits). Max value 262−1≈4×1018.

        - **Calculation Example for Frame Type 0x06 (CRYPTO):**
            - Decimal 6 is binary 110.
            - Smallest encoding (1-byte, prefix 00): 00000110 (Hex 06). This is required for frame types.
            - Valid variable-length encodings for the number 6 elsewhere:
                - 00000110 (Hex 06)
                - 01000000 00000110 (Hex 4006)
                - 10000000 00000000 00000000 00000110 (Hex 80000006)
                - 11000000 ... 00000110 (Hex C000...0006)
- **QUIC Packets Revisited (Example Figure 6.14):**
    - Client's first Initial packet often has Source Connection ID length 0 (omitted). Destination Connection ID is a temporary random value.
    - Payload can contain multiple frames (CRYPTO, PING, PADDING).
- **Tearing Down QUIC Connections:** Graceful termination similar to TCP's three-way handshake, or abnormal termination on timeouts/errors.
- **QUIC Reliability during Communication:**
    - Packets are numbered in their respective spaces (Initial, Handshake, Application Data).
    - Packets are ACKed; lost/damaged packets trigger retransmission of their *frames* in new packets with new packet numbers.
    - ACK frames provide detailed info about received packet ranges and timing, allowing accurate RTT calculation.
- **QUIC Flow Control:**
    - **Connection-level:** Endpoint advertises total buffer space (MAX_DATA frame).
    - **Stream-level:** Endpoint advertises buffer space per stream (MAX_STREAM_DATA frame).
    - Sender must not exceed these limits. Initial limits set during handshake.
    - DATA_BLOCKED / STREAM_DATA_BLOCKED frames signal sender is blocked by flow control limits.
- **Using Well-Known Ports in QUIC:**
    - Initial QUIC packets are sent to a well-known UDP port (e.g., UDP 443 for HTTPS).
    - Connection IDs are established during the handshake and then used for the duration of the connection, enabling migration.
    - Clients often initiate TCP and QUIC connection attempts in parallel; use QUIC if successful, fallback to TCP otherwise.
- **QUIC Updates and Ossification:**
    - **Protocol Ossification:** Inability to evolve a protocol because intermediate devices (middleboxes like firewalls) make assumptions about its wire format and break if it changes. TCP suffered from this.
    - QUIC aims to combat ossification:
        - Encrypts most of the transport header.
        - Uses unpredictable version numbers (e.g., v2 uses 0x6b3343cf not

0x00000002) and varies bit patterns for packet types between versions to prevent middleboxes from hardcoding assumptions.

- Designed for more frequent updates, potentially moving QUIC logic out of OS kernel into application space for easier tailoring.
- QUICv2 (RFC 9369) released shortly after QUICv1 mainly to exercise version negotiation and combat ossification vectors.