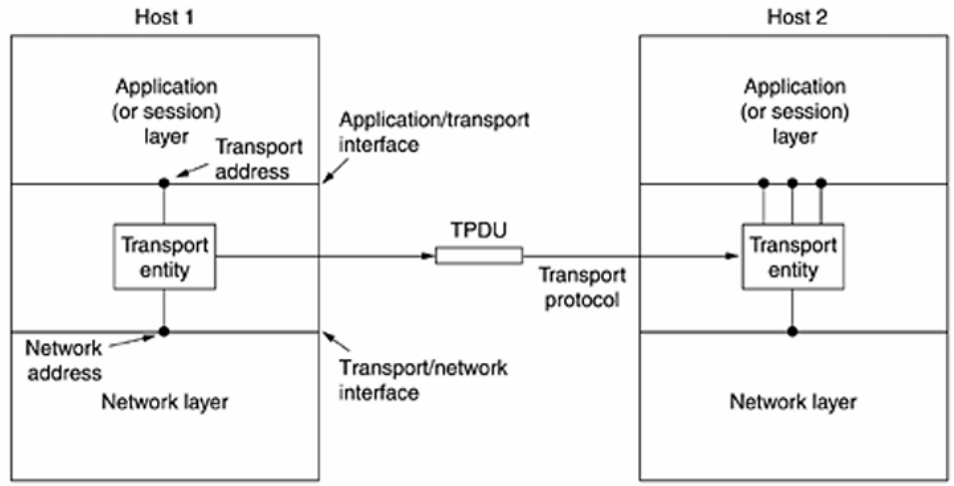
UNIT – 4

1. What is the Transport Layer?

The **Transport Layer** is a core part of network communication. Its main job is to provide reliable, cost-effective data transfer from the source to the destination, independent of the underlying physical network. It ensures that the data sent from one application reaches another application correctly and in order.

Figure 6-1. The network, transport, and application layers.



2. Why Do We Need the Transport Layer?

Even if the **Network Layer** (below the Transport Layer) can send data across the network, it may not be reliable. It could lose packets or face errors. The Transport Layer fixes these problems by:

- **Handling errors** (detecting lost or corrupted packets).
- **Ensuring reliable delivery** of data even if the network layer is unreliable.
- **Abstracting the network details** so that application developers don't need to worry about the underlying network complexities.

3. Services Provided by the Transport Layer

The Transport Layer offers two types of services:

1. **Connection-Oriented Service** (e.g., TCP):
    - Involves three phases: **Connection Establishment**, **Data Transfer**, and **Connection Release**.
    - Provides reliable data transfer with error checking, acknowledgments, and flow control.
2. **Connectionless Service** (e.g., UDP):
    - No need to establish a connection before sending data.
    - Faster but less reliable, as it doesn't guarantee packet delivery or order.

4. Key Functions of the Transport Layer

- **Reliable Data Transfer:** Ensures data reaches its destination correctly.
- **Error Control:** Detects and corrects errors in data transmission.
- **Flow Control:** Manages data flow to prevent the sender from overwhelming the receiver.
- **Multiplexing/Demultiplexing:** Allows multiple applications to use the network simultaneously by directing data to the correct application using port numbers.

5. Transport Service Primitives

To interact with the transport service, certain basic operations (primitives) are used:

- **LISTEN:** The server waits for a connection request.
- **CONNECT:** The client requests a connection to the server.
- **SEND:** Data is sent from one application to another.

- **RECEIVE:** The application waits and receives incoming data.
- **DISCONNECT:** Either side can end the connection when communication is finished.

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

6. Example Scenario (Client-Server Model):

1. **Server** calls **LISTEN**, waiting for clients.
2. **Client** calls **CONNECT** to initiate a connection.
3. **Server** accepts the connection, and both can **SEND** and **RECEIVE** data.
4. After communication, both call **DISCONNECT** to end the session.

Figure 6-3. Nesting of TPDUs, packets, and frames.

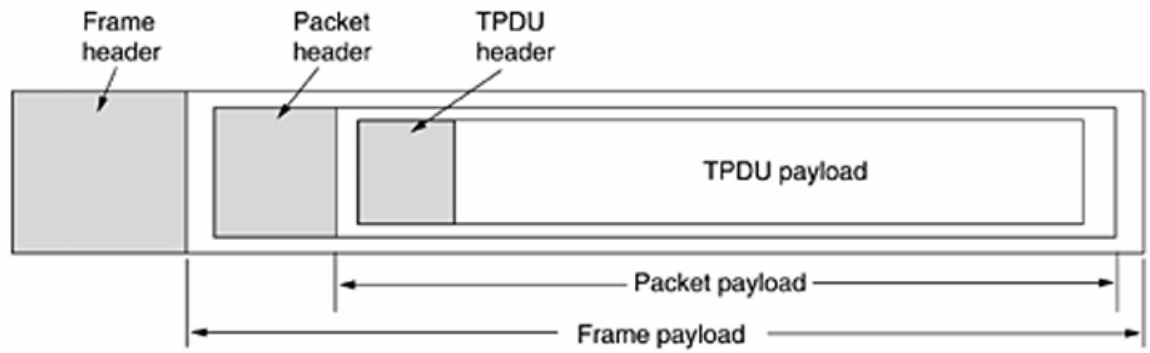
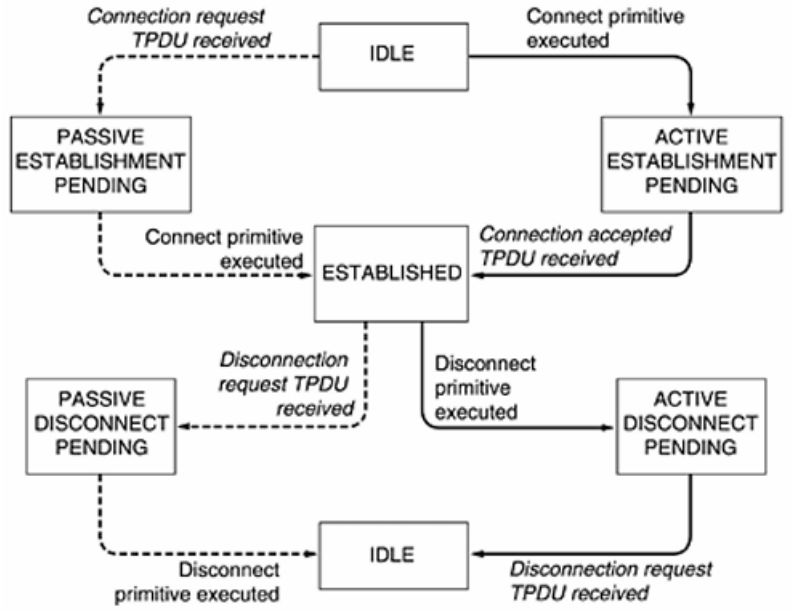


Figure 6-4. A state diagram for a simple connection management scheme. Transitions labeled in italics are caused by packet arrivals. The solid lines show the client's state sequence. The dashed lines show the server's state sequence.



A **socket** is an endpoint for sending or receiving data across a network.
It allows two devices (client and server) to communicate using TCP/IP.

- **Server Side**: Creates a socket, assigns an address (binds), listens for connections, and accepts clients.
- **Client Side**: Creates a socket and connects to the server.
- **Data Exchange**: Both use the socket to send and receive messages.

A **socket** is defined by two main components:

1. **IP Address**: Identifies the device on the network (e.g., 192.168.1.1).
2. **Port Number**: Identifies the specific application or service on the device (e.g., port 80 for HTTP, port 443 for HTTPS).

Together, the **IP address** and **port** form a unique **socket address** (e.g., 192.168.1.1:8080) that is used for network communication.

- **Server Side**: Binds to a specific IP and port to listen for incoming client requests.
- **Client Side**: Uses its own IP and a random port to connect to the server's IP and specified port.

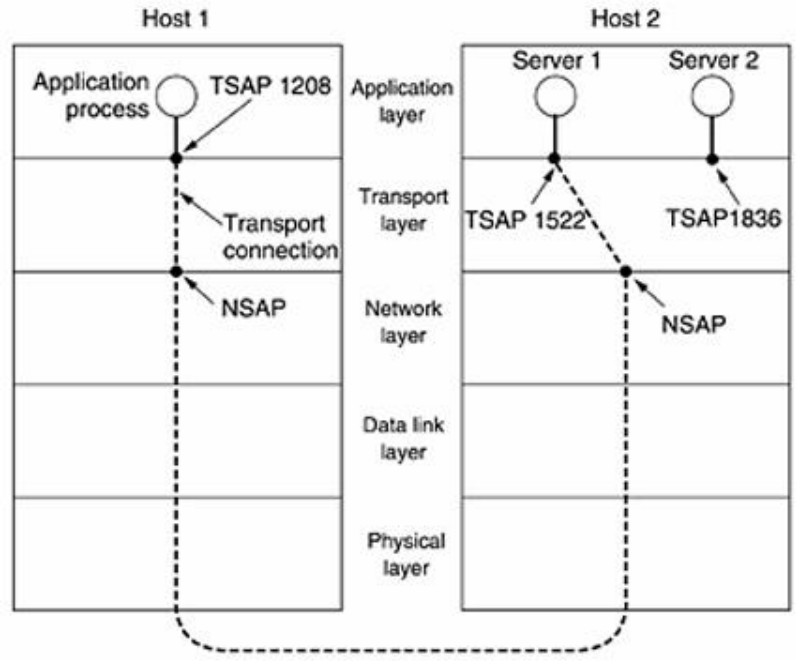
This combination ensures that data is sent to the correct application on the correct device.

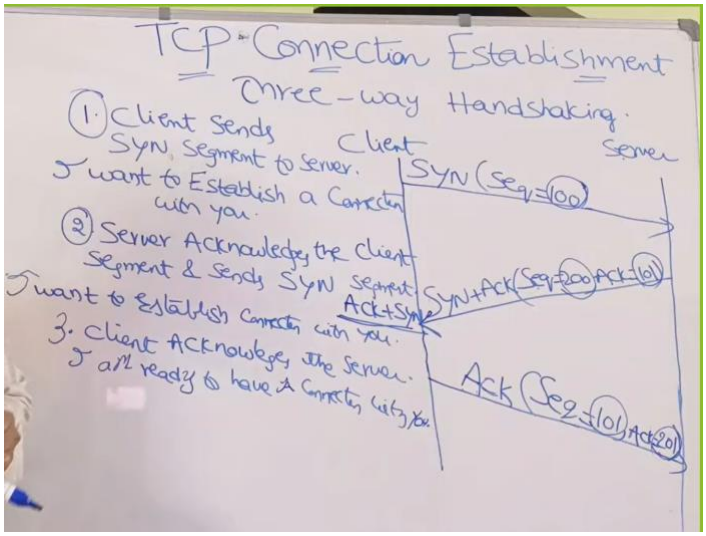
Key Terms:

1. **TSAP (Transport Service Access Point)**:
   - It's like a specific door number where an application listens for connections on a computer.
   - In the internet, this is called a **port** (e.g., port 80 for web servers).
2. **NSAP (Network Service Access Point)**:
   - It's the address of the computer in the network (like an IP address).

Together, **TSAP** helps identify specific programs on a computer, while **NSAP** identifies the computer itself.

Figure 6-8. TSAPs, NSAPs, and transport connections.





Connection Establishment in Transport Protocols (Detailed and Simple)

In computer networks, **connection establishment** is the process where two devices (usually called the client and the server) set up a reliable communication link to start exchanging data. This is crucial, especially in protocols like **TCP** (Transmission Control Protocol), which ensures that data is sent reliably and in the correct order.

Key Steps in Connection Establishment

1. **Addressing**:
    - Before establishing a connection, the client needs to know where to send the data. This involves:
        - **IP Address** (e.g., 192.168.1.1): Identifies the device (host).
        - **Port Number** (e.g., port 80): Identifies the specific application/service on the device, like a web server or an email server.

The combination of an IP address and port number is known as a **socket** (e.g., 192.168.1.1:80).

2. **Three-Way Handshake** (TCP Example):
    - TCP uses a method called the **Three-Way Handshake** to establish a reliable connection between the client and the server.
    - The three steps involved are:

Step 1: SYN (Synchronize)

- The **client** sends a **SYN** message to the server to indicate it wants to establish a connection.
- This message also includes a **sequence number**, which is used to track the order of packets.

Step 2: SYN-ACK (Synchronize-Acknowledgment)

- The **server** receives the SYN message and responds with a **SYN-ACK** message.
- The SYN-ACK message serves two purposes:
    - It acknowledges the client's SYN request.
    - It also includes the server's own sequence number to start its side of the connection.

Step 3: ACK (Acknowledgment)

- The **client** responds to the server's SYN-ACK with an **ACK** message.
- This final acknowledgment confirms that both parties are ready, and the connection is established.

After this handshake, the connection is established, and both client and server can start exchanging data.

3. **Data Transfer**:
    - Once the connection is established, the client and server can start sending and receiving data.
    - The data is divided into packets, each with sequence numbers to ensure they are received in the correct order.

4. **Connection Termination**:
    - When either the client or server finishes sending data, they need to close the connection using a **four-step process** known as the **Four-Way Handshake**.
    - This involves exchanging **FIN** (Finish) and **ACK** (Acknowledgment) messages to ensure both sides agree to close the connection safely.

Example Scenario

Imagine you want to access a website:

1. **Client** (Your Browser):
    - Sends a SYN message to the server of the website (e.g., www.example.com) to start a connection.
2. **Server**:
    - Receives the SYN message, responds with a SYN-ACK message to acknowledge the request.
3. **Client**:
    - Sends an ACK message to confirm, and now the connection is open.

Now, your browser can send an HTTP request (like asking for the homepage), and the server can respond with the webpage data.

Why Use a Handshake?

The handshake process ensures:

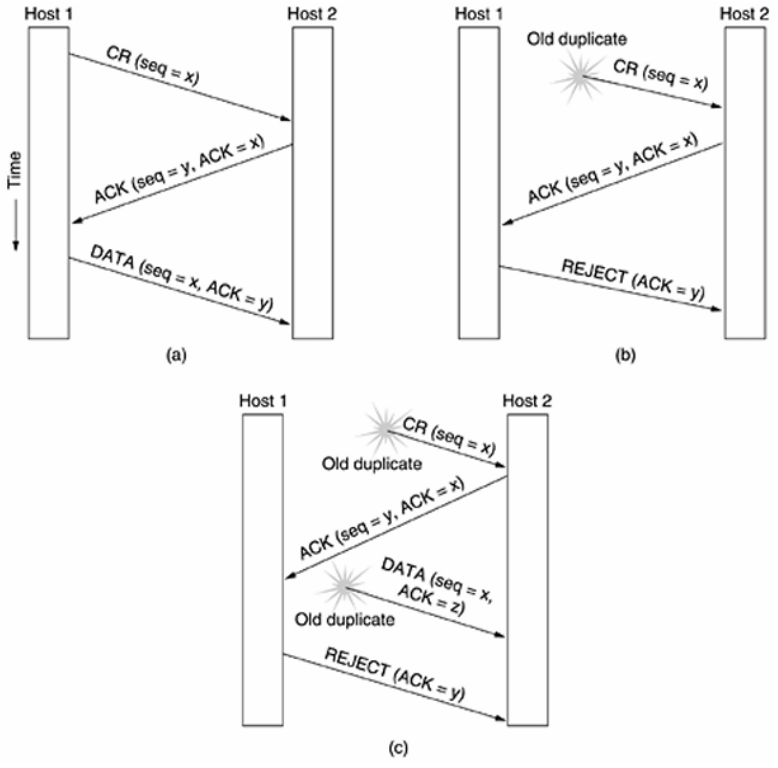
- **Reliability**: Both sides agree to communicate, reducing the chance of data loss.
- **Synchronization**: Sequence numbers help keep track of data packets, ensuring they are received in order.

In simpler protocols like **UDP** (User Datagram Protocol), this handshake is not used. UDP is faster but less reliable because it doesn't establish a connection before sending data.

The **Three-Way Handshake** in TCP is what makes it reliable for critical applications like web browsing, file transfers, and email, where data integrity is important.

Read notes too

Figure 6-11. Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST. (a) Normal operation. (b) Old duplicate CONNECTION REQUEST appearing out of nowhere. (c) Duplicate CONNECTION REQUEST and duplicate ACK.



Problem of Delayed Duplicate Messages

In real-world scenarios, sometimes old or delayed messages from previous connections can reappear due to network issues. This can potentially cause confusion if the old messages are mistaken as new connection requests. The three-way handshake solves this problem.

Handling Delayed Duplicate Messages

1. **Img - b: Old Connection Request (Fig. 6-11(b))**:
    - Suppose an old **CONNECTION REQUEST** (with sequence number x) from a previous session arrives unexpectedly at Host 2.
    - Host 2 responds with an **ACK** message but waits for Host 1 to confirm the connection.
    - Host 1, upon receiving this unexpected ACK message, rejects the request because it wasn't trying to establish a new connection.
    - Host 2 then realizes it was an old message and ignores it, preventing an accidental connection.
2. **Img - c: Old Connection Request and Old ACK (Fig. 6-11(c))**:
    - Sometimes, both an old **CONNECTION REQUEST** and its corresponding old **ACK** message can appear in the network.
    - When Host 2 receives the old CONNECTION REQUEST, it replies with an ACK using a new sequence number y.
    - If another delayed ACK from the old session arrives at Host 2, it checks if the sequence numbers match.
    - Since the old ACK would be using an outdated sequence number (z instead of y), Host 2 identifies it as a duplicate and ignores it.
    - This mechanism ensures that no accidental connection is established because of old, delayed messages.

Why is the Three-Way Handshake Important?

- It ensures that both hosts are synchronized and ready for data transfer.
- It prevents old, delayed messages from causing accidental or unwanted connections.
- The use of sequence numbers helps identify new and old messages, adding a layer of security and reliability.

Connection Release Explained Simply

Releasing a network connection (disconnecting) is usually simpler than establishing it, but it can still be tricky.

There are two main ways to release a connection: **asymmetric release** and **symmetric release**.

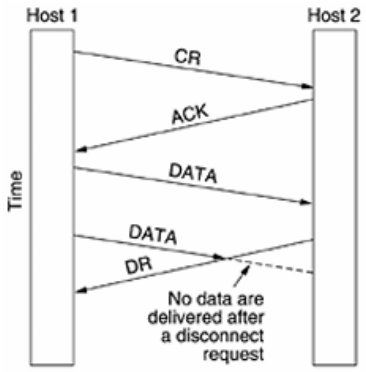
1. Asymmetric Release (like a phone hang-up)

- One side (e.g., Host 1) sends a disconnect message, and the connection is immediately broken.
- However, this can cause problems, like **data loss**. For example, if Host 1 sends some data and Host 2 disconnects before it receives that data, the data will be lost.

2. Symmetric Release (more controlled)

- In **symmetric release**, both sides release the connection independently, ensuring no data is lost.
- For example, Host 1 might send a disconnect request, but Host 2 can still receive data until it also decides to disconnect.

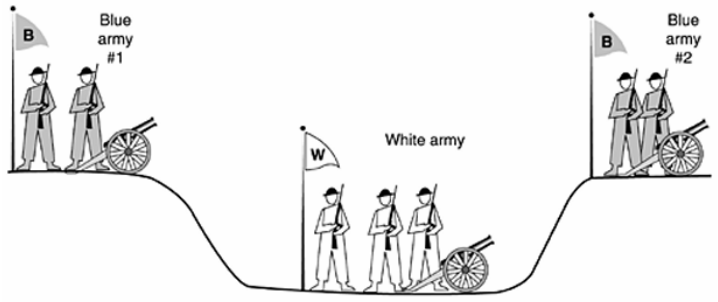
Figure 6-12. Abrupt disconnection with loss of data.



The Two-Army Problem

- This problem shows the challenge of coordinating actions (like disconnecting) when both sides need to be sure the other is ready. If neither side is sure the other is ready, neither will act, and the connection will never be released.
- In real-life situations, though, people are willing to take risks, so it's not as bad as the army problem.

Figure 6-13. The two-army problem.



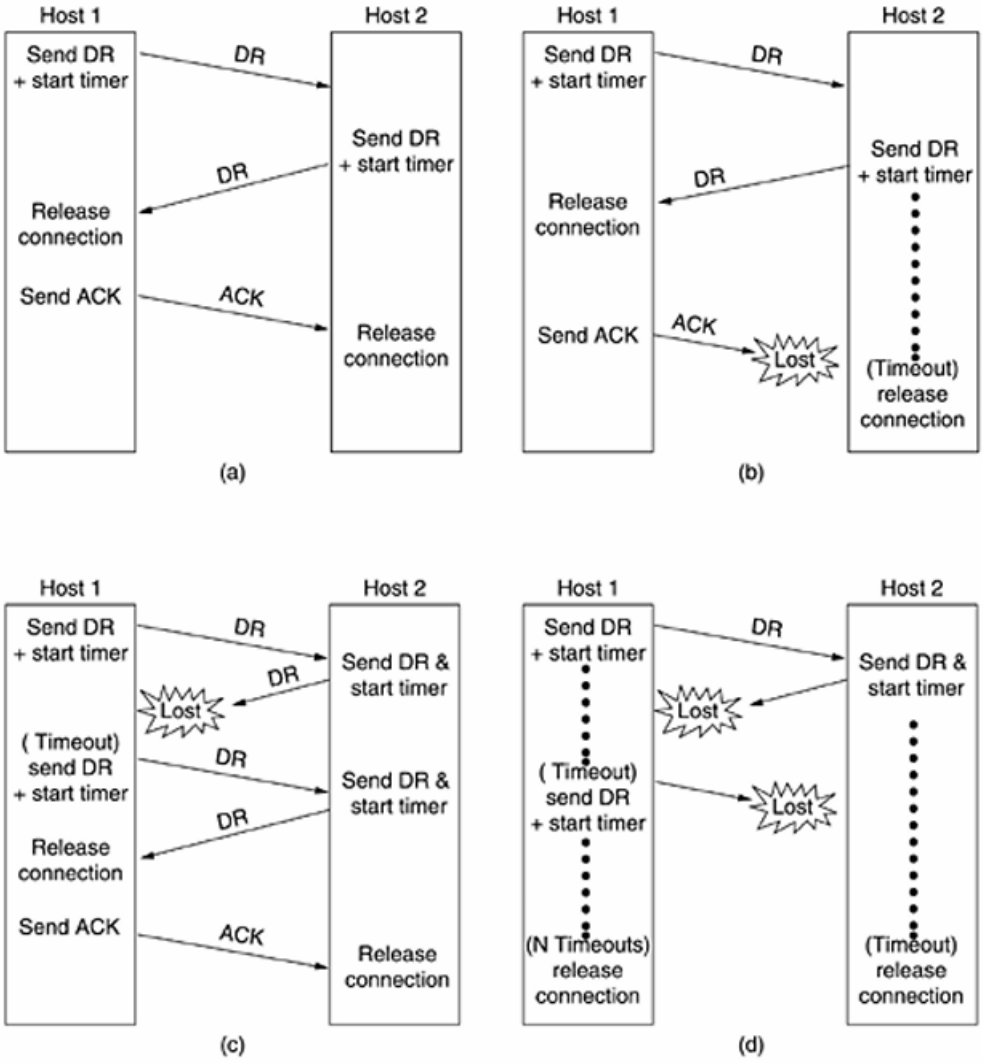
Three-Way Handshake for Disconnecting

- A **three-way handshake** is used for a reliable connection release:
    1. **Host 1** sends a **Disconnect Request (DR)** to Host 2.
    2. **Host 2** responds with its own DR, and starts a timer in case the message gets lost.
    3. **Host 1** acknowledges Host 2's DR, and once Host 2 gets the ACK, it knows the connection can be safely released.

Possible Issues During Connection Release:

- **Lost messages**: If any messages are lost, timers are used to resend or force the release after a timeout.
- **Repeated timeouts**: If the disconnection messages keep getting lost, the sender will eventually give up after several retries, but this can cause a "half-open" connection, where one side believes the connection is closed, but the other side doesn't know.
- **Automatic timeout**: To prevent half-open connections, a rule can be set where if no activity happens for a certain time, the connection is automatically closed.

Figure 6-14. Four protocol scenarios for releasing a connection. (a) Normal case of three-way handshake. (b) Final ACK lost. (c) Response lost. (d) Response lost and subsequent DRs lost.



Flow control and buffering

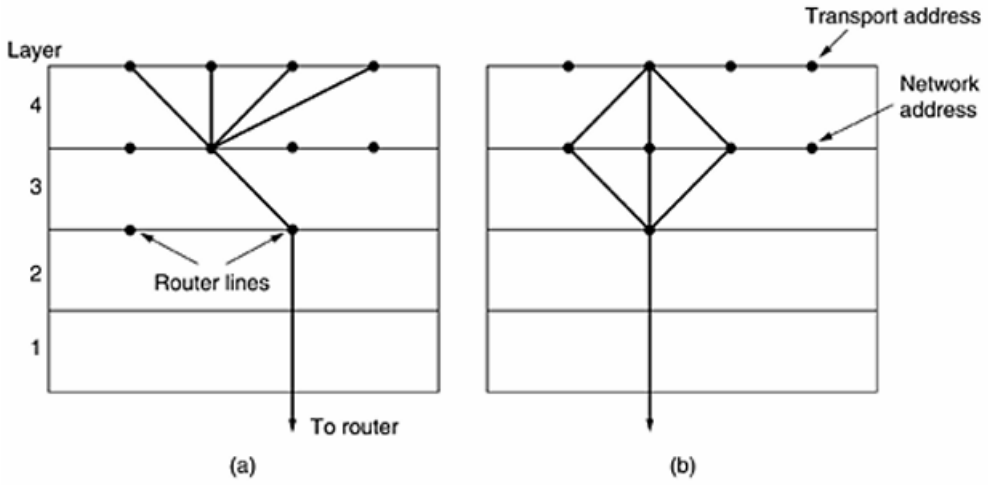
Flow control and buffering are essential for managing data transmission between a sender and a receiver. Here's a simplified breakdown:

1. **Flow Control**: It prevents a fast sender from overwhelming a slow receiver. Similar to data link layer flow control, it uses sliding windows to manage how much data is sent before waiting for an acknowledgment. However, in transport layer protocols, there can be many connections, so the strategies used in data link layers don’t always work well.
2. **Buffering**: Buffers store data temporarily while it is being sent or received. The sender might need to buffer data until it is acknowledged by the receiver. If the receiver is slow or its buffers are full, the sender may have to hold onto its data until the receiver is ready.
3. **Dynamic Buffer Allocation**: Buffers can be allocated dynamically, depending on the amount of data traffic. For high-bandwidth traffic (like file transfers), the receiver may dedicate buffers, while for bursty traffic (like interactive sessions), buffers may be managed dynamically.
4. **Buffer Size Management**: Different traffic types require different buffer sizes. Fixed-size buffers work for uniform data sizes, while variable-sized buffers are better for varying data sizes but are harder to manage.
5. **Deadlock Prevention**: If control messages (like buffer allocations) are lost, it can cause a deadlock, where the sender waits forever. To avoid this, periodic control messages are sent to maintain the flow.
6. **Network Congestion**: The flow control should also account for the network's capacity to handle data. If the sender sends too much data too fast, it can overwhelm the network. To avoid this, the sender adjusts its data flow based on the network’s capacity to handle the traffic.

Multiplexing:

- **Upward Multiplexing**: Multiple transport connections share the same network address, and the system decides which process gets the data.
- **Downward Multiplexing**: Multiple network connections are used to increase bandwidth (e.g., combining two ISDN lines for faster speeds).

Figure 6-17. (a) Upward multiplexing. (b) Downward multiplexing.



Crash Recovery in Networking :

1. **Network Crashes**:
   - If a network or router crashes, recovery is easier if the transport layer is handled by the host.
   - In a **datagram service**, lost packets are expected and retransmitted.
   - In a **connection-oriented service**, a new connection is made, and missing packets are requested.
2. **Host Crashes**:
   - When a server crashes during data transfer, it may forget where it left off, causing problems for clients trying to resend data.
3. **Client's Response**:
   - The server asks clients about the status of open connections.
   - Clients decide to resend data based on whether there is still unacknowledged data.
4. **Retransmission Issues**:
   - A crash can cause confusion about whether the data was received or not, leading to missing or duplicate data.
5. **No Perfect Fix**:
   - Different server and client strategies cause problems, and there's no foolproof way to ensure recovery works in all situations.
6. **End-to-End Acknowledgment Problem**:
   - Acknowledging receipt doesn’t guarantee that the task (like a database update) was completed, as crashes can happen at any point.
7. **Layered Recovery**:
   - If a crash happens in one layer (e.g., transport), recovery must be managed by higher layers (e.g., application).
   - The transport layer can only recover from failures if enough status information is saved.

Figure 6-18. Different combinations of client and server strategy.

Strategy used by sending host	Strategy used by receiving host: First ACK, then write			Strategy used by receiving host: First write, then ACK		
	AC(W)	AWC	C(AW)	C(WA)	W AC	WC(A)
Always retransmit	OK	DUP	OK	OK	DUP	DUP
Never retransmit	LOST	OK	LOST	LOST	OK	OK
Retransmit in S0	OK	DUP	LOST	LOST	DUP	OK
Retransmit in S1	LOST	OK	OK	OK	OK	DUP

OK = Protocol functions correctly
DUP = Protocol generates a duplicate message
LOST = Protocol loses a message

The Internet Transport Protocols: UDP

The Internet has two main protocols in the transport layer, a connectionless protocol and a connection-oriented one.

1. The connectionless protocol is UDP
2. The connection-oriented protocol is TCP.

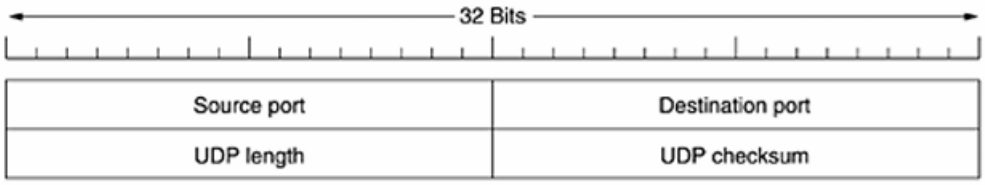
User Datagram Protocol (UDP)

What is UDP?

- UDP is a connectionless transport protocol used in computer networks.
- It allows data to be sent directly from one computer to another without establishing a connection, using small packets called datagrams.
- It is part of the internet protocol suite and is commonly used in applications where speed is critical, and a small amount of data loss is acceptable.
- The UDP (User Datagram Protocol) header is crucial for transmitting data between applications over a network.
- It is very simple and consists of only 8 bytes.
- The structure of the UDP header is designed to support the basic functionality of sending data from one machine to another using IP addresses, with the ability to deliver data to specific applications on the source and destination machines through the use of port numbers.

The UDP header contains the following fields:

Figure 6-23. The UDP header.



1. **Source Port (16 bits)**:
    - Identifies the port of the sending application (source).
    - This field is primarily useful when a response is expected, as the source port tells the receiver which port to send the response back to. If the source port is zero, the field is ignored.

2. **Destination Port (16 bits)**:
    - Identifies the port on the destination machine where the data is to be delivered. This allows the UDP protocol to direct the datagram to the correct process or application on the receiving machine. Without this, the transport layer wouldn't know how to handle the packet.
3. **Length (16 bits)**:
    - This field specifies the total length of the UDP packet, including the header and the data (payload). The minimum value for this field is 8 bytes, which accounts for the size of the header alone. The length provides the receiver with information on how much data is contained in the UDP datagram.
4. **Checksum (16 bits)**:
    - The UDP checksum is optional. It helps to verify the integrity of the data. When the checksum is used, it ensures that the data has not been corrupted during transmission. If the checksum is not computed, this field is set to zero (but stored as all 1s in the data field).
    - Although the checksum is optional, it is recommended to use it, particularly in cases where data integrity is important (such as with files, images, etc.). In cases like digitized speech, it may be omitted if minor data loss is acceptable.

UDP Header Structure (8 bytes total):

Field	Size (in bits)	Description
Source Port	16 bits	Identifies the source application port.
Destination Port	16 bits	Identifies the destination application port.
Length	16 bits	Specifies the length of the UDP datagram (header + data).
Checksum	16 bits	Optional field used to check data integrity. If not used, it's set to zero.

Key Features of UDP:

1. **No Connection Establishment**: UDP doesn't require setting up a connection before sending data, which makes it faster than connection-oriented protocols like TCP.
2. **Unreliable Transmission**: There is no error checking or recovery in UDP. If a packet is lost or corrupted, UDP doesn't retransmit it. Data delivery is not guaranteed.
3. **Low Overhead**: UDP has a small, simple 8-byte header and no need for connection management, resulting in low overhead and higher speed.
4. **Message Boundary Preservation**: UDP preserves message boundaries, meaning that even if data is sent in multiple packets, the receiving application will interpret them as a single message.
5. **No Flow or Congestion Control**: UDP does not manage the flow of data or react to network congestion, which could lead to packet loss in overloaded networks.

Advantages of UDP:

1. **Speed**: UDP is faster since it doesn't require connection setup, retransmissions, or acknowledgments.
2. **Simplicity**: UDP's simple header and lack of error recovery or flow control make it easier to implement and more efficient for small data transmissions.
3. **Real-Time Applications**: Ideal for applications that need fast delivery, such as video or voice streaming, where small data loss is tolerable but low latency is critical.

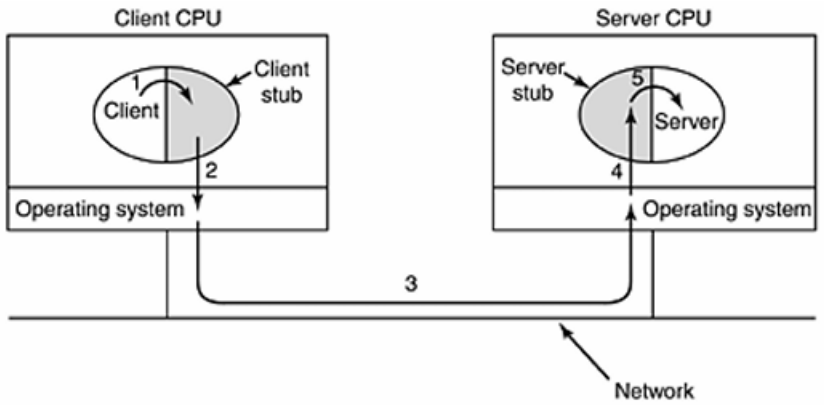
Use Cases of UDP:

1. **DNS (Domain Name System)**: DNS queries and responses use UDP due to its speed, as it requires only two messages—request and response—without the need for a connection.

2. **Real-Time Applications**: UDP is widely used in streaming services (RTP for video/audio), VoIP (Voice over IP), and online gaming, where data needs to be delivered quickly, and occasional packet loss is acceptable.
3. **Remote Procedure Calls (RPC)**: UDP is used for making quick requests where retransmission or acknowledgments are unnecessary.

   A **Remote Procedure Call (RPC)** is a way for a program on one computer (the client) to call a procedure on another computer (the server) as if it were a local function call. It hides the complexities of networking from the programmer. The client sends a request to the server, which processes it and sends a response back. This process makes network communication simpler and more like calling a function locally.

Figure 6-24. Steps in making a remote procedure call. The stubs are shaded.



Conclusion: UDP is perfect for applications that need quick, simple communication without the overhead of connection management or error checking. Although it doesn't guarantee data delivery or order, it is used in cases where speed is prioritized, such as real-time communications, DNS lookups, and streaming services.

RTP (Real-time Transport Protocol):
RTP is a protocol used to deliver audio, video, and other multimedia data over IP networks in real-time. It is commonly used in applications such as voice over IP (VoIP), video conferencing, and streaming media.

RTCP (Real-time Transport Control Protocol):

- **Purpose**: RTCP works alongside RTP (Real-time Transport Protocol) to manage feedback, synchronization, and user interface but does not transport any data itself.
- **Functions**:
  1. Provides feedback on network conditions (e.g., delay, jitter, bandwidth).
  2. Helps adjust encoding algorithms (like switching from MP3 to PCM) based on network conditions.
  3. Ensures synchronization between different streams, handling clock differences and drift.
- **Key Role**: Improves stream quality and synchronization by providing real-time feedback and control.

Internet Transmission Protocol - Transmission Control Protocol (TCP)

Transmission Control Protocol (TCP) is a reliable, connection-oriented transport layer protocol used for ensuring accurate data transmission between devices. It is widely used in applications where data integrity is crucial, such as web browsing, file transfers, and email.

Key Features of TCP:

1. **Connection-Oriented**:
    - Establishes a reliable connection using a **three-way handshake**:
        - **SYN**: Sender initiates the connection.
        - **SYN-ACK**: Receiver acknowledges the request.
        - **ACK**: Sender confirms the connection.
2. **Reliable Data Transmission**:
    - Uses **acknowledgments**, **retransmissions**, and **sequence numbers** to ensure data is delivered correctly and in order.
    - If a packet is lost, TCP **retransmits** it.
3. **Flow Control**:
    - Uses a **sliding window mechanism** to manage data flow and prevent congestion.
    - Adjusts the rate of data transmission based on the receiver's buffer capacity.
4. **Congestion Control**:
    - Adapts data transmission rate based on network conditions using algorithms like **Slow Start** and **Congestion Avoidance**.
5. **Error Detection**:
    - Incorporates a **checksum** to detect errors in transmitted segments, ensuring data integrity.
6. **Segmenting and Reassembly**:
    - Breaks large messages into smaller **segments** and reassembles them at the receiver using sequence numbers.
7. **Full-Duplex Communication**:
    - Supports simultaneous two-way data transmission between sender and receiver.
8. **Graceful Termination**:
    - Closes connections using a **four-step process** with FIN (Finish) packets, ensuring both sides complete their transmission.

Use Cases:

- **Web Browsing** (HTTP/HTTPS), **File Transfers** (FTP), and **Email** (SMTP, IMAP) rely on TCP for reliable data transmission.

Summary:

TCP ensures reliable, error-free, and in-order delivery of data, making it suitable for applications where data accuracy is essential. It manages congestion and flow control dynamically, providing efficient and stable communication over networks.

Timer Management in TCP:

- **Retransmission Timer**: TCP uses timers to detect lost segments. If the acknowledgment is not received before the timer expires, the segment is retransmitted.
- **Persistence Timer**: If the receiver’s window size is zero, the persistence timer ensures that the sender keeps sending a probe until the window size becomes non-zero.
- **Keepalive Timer**: Used to check if the connection is still active, especially during periods of inactivity. If no response is received, the connection is terminated.

Dynamic Timeout Estimation:

- TCP adjusts its retransmission timeout (RTO) dynamically based on round-trip time (RTT) measurements. The timeout is calculated using an algorithm that smooths the RTT and accounts for variations in network conditions.
- The timeout is adjusted with factors like RTT deviation and network congestion, ensuring that retransmissions occur without excessive delays or unnecessary retransmissions.

Karn's Algorithm:

- **Karn's Algorithm** prevents inaccurate RTT calculations when retransmitting a segment. The algorithm suggests not updating the RTT estimate when a segment is retransmitted.

Challenges in TCP:

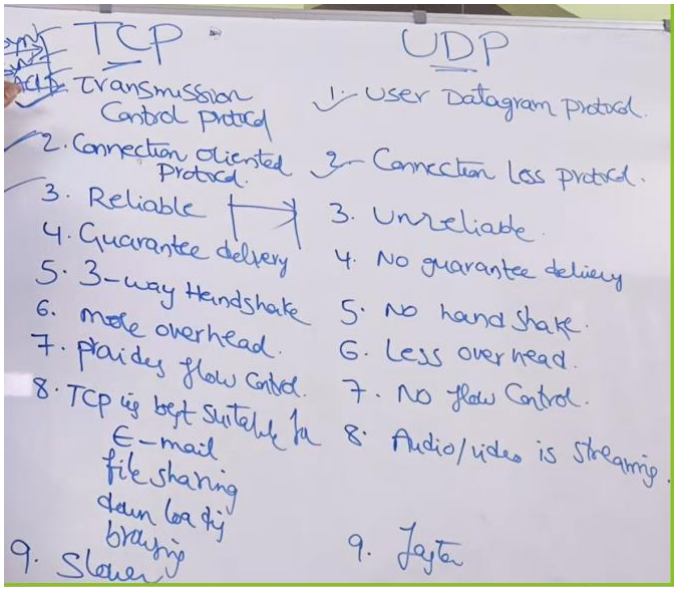
- **Wireless Networks**: TCP was originally optimized for wired networks, and it assumes that packet loss is due to congestion. However, in wireless networks, packet loss can occur due to unreliable transmission, which causes TCP to slow down unnecessarily.
- **Indirect TCP (I-TCP)**: To address this, an approach called indirect TCP splits the connection into two parts: one for the wired network and one for the wireless part, allowing each to be optimized separately.
- **Snooping TCP**: This variant improves wireless performance by allowing intermediate devices (like base stations) to retransmit lost packets without invoking the congestion control algorithm.

Use Cases of TCP:

- **Web Browsing**: TCP ensures that the data packets containing the HTML, images, and other content are delivered reliably and in order.
- **File Transfers**: FTP (File Transfer Protocol) uses TCP to ensure that files are transmitted accurately and without corruption.
- **Email**: SMTP and IMAP use TCP for reliable transmission of email data.

Summary:

TCP is a reliable, connection-oriented protocol that provides error-free, in-order delivery of data. It is best suited for applications where reliability and data integrity are more important than speed, such as file transfer, email, and web browsing. The dynamic adjustment of timeout and flow control mechanisms ensures efficient data transmission, even in the presence of network congestion or packet loss.



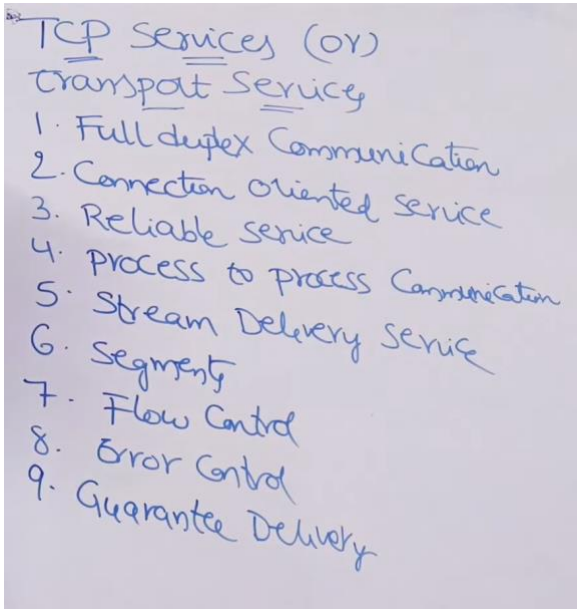


Figure 6-27. Some assigned ports.

Port	Protocol	Use
21	FTP	File transfer
23	Telnet	Remote login
25	SMTP	E-mail
69	TFTP	Trivial file transfer protocol
79	Finger	Lookup information about a user
80	HTTP	World Wide Web
110	POP-3	Remote e-mail access
119	NNTP	USENET news

TCP Header - Simple Explanation

The **TCP (Transmission Control Protocol) header** is a part of every TCP packet and contains important information for managing data transmission between two devices. It helps in establishing connections, ensuring data is transmitted reliably, and controlling the flow of data.

Structure of TCP Header (20-60 bytes)

The minimum size of a TCP header is **20 bytes** (160 bits), but it can extend up to **60 bytes** if optional fields are included.

Main Fields in TCP Header:

1. **Source Port (16 bits)**:
   - Identifies the port of the **sending application** on the source device.
2. **Destination Port (16 bits)**:
   - Identifies the port of the **receiving application** on the destination device.
3. **Sequence Number (32 bits)**:
   - Keeps track of the **order of data** sent. It indicates the starting byte number of the data in this packet.
4. **Acknowledgment Number (32 bits)**:
   - Used by the receiver to inform the sender of the **next expected byte**. This ensures reliable data transfer.
5. **Data Offset (4 bits)**:
   - Specifies the **length of the TCP header** in 32-bit words. This tells where the actual data begins.
6. **Reserved (3 bits)**:
   - These bits are **unused** and reserved for future use.
7. **Flags (9 bits)**: Also known as control bits, they indicate different states or controls for the connection:
   - **URG**: Urgent pointer field is significant.
   - **ACK**: Acknowledgment field is significant.
   - **PSH**: Push function (tells the receiver to process the data immediately).
   - **RST**: Reset the connection.
   - **SYN**: Synchronize sequence numbers to initiate a connection.
   - **FIN**: Finish, used to terminate a connection.
8. **Window Size (16 bits)**:
   - Specifies the **size of the receiver's buffer**, which is the amount of data the receiver can accept without acknowledgment.
9. **Checksum (16 bits)**:
   - Used for **error-checking** the TCP header and data. It ensures the data integrity during transmission.
10. **Urgent Pointer (16 bits)**:
    - Points to the **urgent data** in the packet, if the URG flag is set. This helps prioritize certain data.

11. **Options (Variable length, 0-40 bytes)**:
    - Optional field used for extra features, like setting the **Maximum Segment Size (MSS)**. This can extend the header size.
12. **Padding**:
    - Added to make the header size a **multiple of 4 bytes** (32 bits).

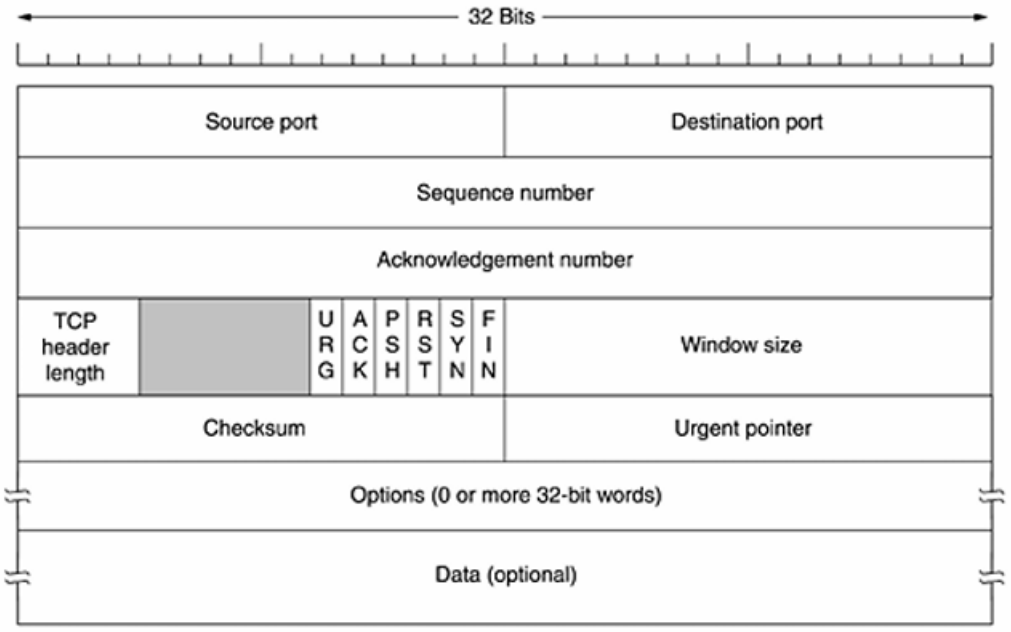
Summary of TCP Header Fields in Bits:

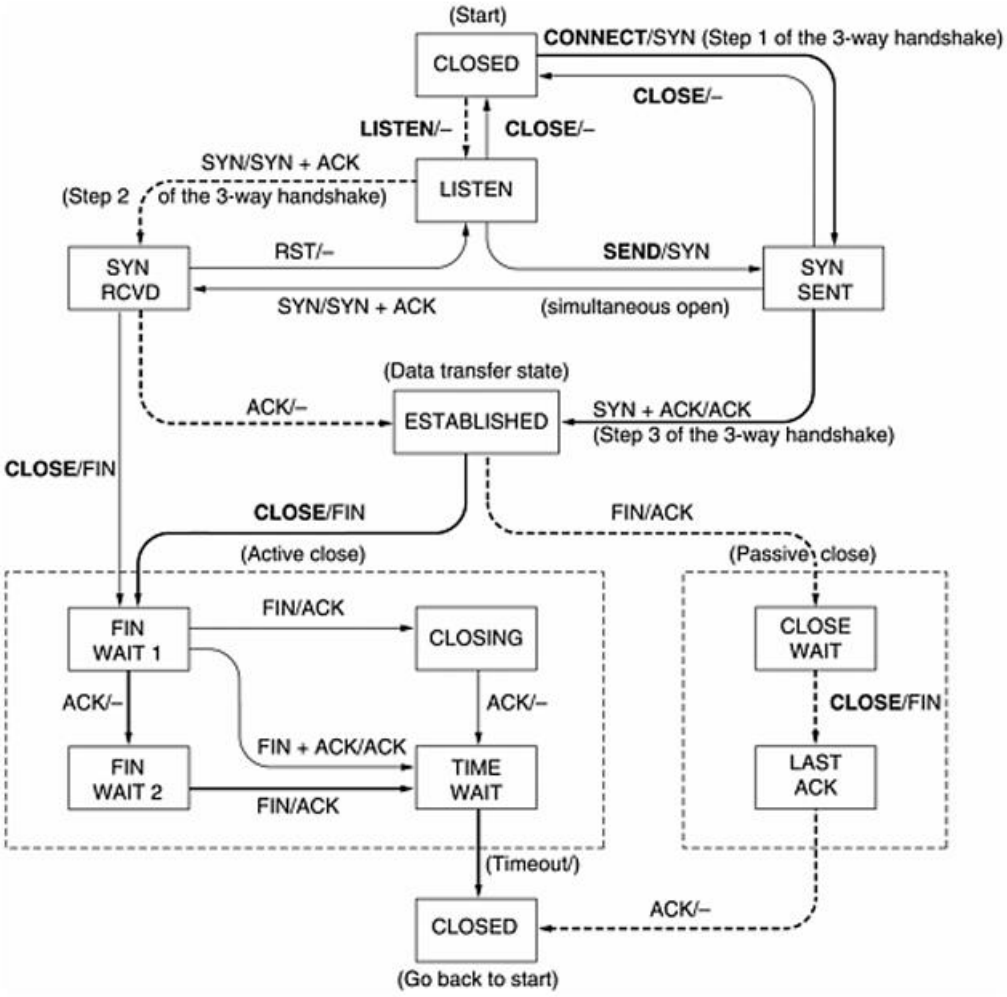
- **Source Port**: 16 bits
- **Destination Port**: 16 bits
- **Sequence Number**: 32 bits
- **Acknowledgment Number**: 32 bits
- **Data Offset**: 4 bits
- **Reserved**: 3 bits
- **Flags**: 9 bits
- **Window Size**: 16 bits
- **Checksum**: 16 bits
- **Urgent Pointer**: 16 bits
- **Options + Padding**: Variable (0-40 bytes)

Conclusion:

The TCP header plays a crucial role in ensuring **reliable data transmission** over the network by managing connection setup, maintaining the order of packets, and detecting errors. It is an essential component of the TCP/IP model used in most internet communications.

Figure 6-29. The TCP header.





TCP connection management finite state machine. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events. Each transition is labeled by the event causing it and the action resulting from it, separated by a slash

Domain Name System (DNS)

The **Domain Name System (DNS)** is a system that translates human-readable domain names (like www.example.com) into machine-readable IP addresses (like 192.0.2.1). It works as a distributed client-server system, where DNS clients (resolvers) send requests to DNS servers to map domain names to addresses or vice versa.

Key Components of DNS:

1. **DNS Hierarchy**:
   - **Root Level**: At the top of the DNS hierarchy.
   - **Generic Domains**: Includes labels like .com, .edu, .gov to define the type of organization.
   - **Country Domains**: Identified by two-letter country codes like .us for the United States, .in for India.
   - **Inverse Domain**: Used for mapping IP addresses to domain names (e.g., PTR queries).
2. **Resolution**:
   - **Name-to-Address Resolution**: The process where a domain name is mapped to its corresponding IP address.
   - **Address-to-Name Resolution**: When an IP address is mapped to a domain name, usually for security checks (PTR query).
   - **Recursive Resolution**: The client requests a final answer from the server, which may query other servers to resolve the query.
   - **Iterative Resolution**: The client is directed to other servers until the query is resolved.
3. **Caching**:
   - DNS servers cache responses to speed up subsequent queries. However, cached information has a time limit (TTL) to prevent outdated data.
4. **DNS Messages**:
   - **Query Messages**: Sent by clients to request information.
   - **Response Messages**: Sent by servers with the requested information, including question, answer, authoritative, and additional sections.
5. **DNS Records**:
   - **Question Records**: Used in queries to get information.
   - **Resource Records**: Contain the actual mapping data, such as IP addresses associated with domain names.
6. **Registrars**:
   - Organizations that manage the registration of domain names in the DNS system.
7. **Dynamic DNS (DDNS)**:
   - Allows automatic updates to DNS records (such as IP address changes), preventing the need for manual changes in DNS files.
8. **Encapsulation**:
   - DNS primarily uses **UDP** for quick responses (under 512 bytes), and **TCP** is used when the data size exceeds 512 bytes.

In simple terms, DNS is like the phonebook of the internet, where it helps to find the corresponding IP address for a given domain name or the domain name for an IP address. It involves various servers and processes to ensure that requests are resolved accurately and efficiently.

What is DNS?

- **DNS (Domain Name System)** translates human-readable domain names (like www.example.com) into IP addresses (like 192.168.1.1) that computers use to identify each other on the internet.
- It supports other applications like email programs by providing IP addresses for email recipients.

Why is DNS needed?

- Humans prefer easy-to-remember names over numeric IP addresses.
- Initially, mapping names to IP addresses was done using a **host file**, but as the internet grew, this method became inefficient and impractical due to size and update challenges.
- DNS solves this by distributing the mapping across many servers instead of relying on a single file or server.

DNS Name Space

DNS uses a hierarchical structure for organizing names:

1. **Flat Name Space**:
    - Names are simple and unstructured (e.g., "computer1").
    - Not practical for large systems because central control is needed to avoid duplicates.
2. **Hierarchical Name Space**:
    - Names are structured, like parts of an address (e.g., host.organization.type).
    - Example: challenger.deanza.edu
    - This allows decentralization, where each organization manages its part of the name.

Domain Name Space

- The structure of domain names resembles an **inverted tree**, with:
    - The **root** at the top.
    - **Labels** (subparts of names) forming branches.
- Example of a full domain name: challenger.atc.deanza.edu. (Ends with a dot indicating the root.)

Types of Domain Names

1. **FQDN (Fully Qualified Domain Name)**:
    - Includes the full path to the root (e.g., www.example.com.).
    - Used to uniquely identify hosts globally.
2. **PQDN (Partially Qualified Domain Name)**:
    - Partial names without the root (e.g., www or challenger).
    - Used within local networks where the missing parts can be filled automatically.

Distribution of DNS Data

- Storing all DNS data in one place would cause bottlenecks and single points of failure.
- DNS distributes data across **many servers**:
    1. **Root Servers**: At the top level, they delegate tasks to other servers.
    2. **Zones**: Each server is responsible for a specific section (zone) of the DNS tree.
        - If a server divides its domain further, it keeps references to lower-level servers.

DNS Servers

1. **Primary Server**:
    - Holds the original data for its zone.
    - Can create, maintain, and update its zone file.
2. **Secondary Server**:
    - Copies data from the primary server (via **zone transfer**).
    - Provides redundancy, ensuring data availability if the primary server fails.

DNS in the Internet

DNS (Domain Name System) organizes domain names into a tree-like structure, divided into **three sections**:

1. **Generic Domains**: Categories like .com, .edu, or .org, used based on the type of organization.
2. **Country Domains**: Two-letter codes like .us (United States) or .in (India) for countries, with subdivisions like states.
3. **Inverse Domain**: Used to map IP addresses back to domain names, useful for servers identifying clients.

Generic Domains

- The first level has specific labels to classify domains:
  - For example, .com for commercial, .edu for education, and .gov for government.
- Each label maps to an organization type (e.g., .biz for businesses, .aero for airlines).

Country Domains

- These use country codes (e.g., .ca.us for California in the United States).
- The domain structure can indicate subdivisions within countries.

Inverse Domain

- It maps **IP addresses** to **domain names**.
- For example, if a server gets an IP like 132.34.45.121, DNS flips it to 121.45.34.132.in-addr.arpa for lookup.

Name Resolution

This is the process of translating domain names to IP addresses or vice versa. It involves:

1. **Resolvers**: Clients (like your computer) send a query to the nearest DNS server.
2. **Recursive Resolution**: The DNS server contacts others until it finds the final answer, then returns it to the client.
3. **Iterative Resolution**: The DNS server gives the client another server's address to try, and the client repeats the query.

Caching

- DNS servers store results temporarily (caching) to speed up future queries.
- However, cached results might become outdated, so a **Time-to-Live (TTL)** is set to ensure old data is removed.

DNS Messages

There are two types of DNS messages:

1. **Query**: Sent by a client to ask for information.
2. **Response**: Sent by a server with the answer.

Both messages have sections like:

- **Header**: Contains IDs and flags.
- **Question Section**: What the client is asking.
- **Answer Section**: The response from the server.

Records

1. **Question Record**: The domain name queried by the client.
2. **Resource Record**: The information returned by the DNS server (e.g., IP address).

Registrars

- Registrars are companies that help register domain names (e.g., example.com) into the DNS database.

- ICANN oversees this process, ensuring the domain is unique.

Dynamic DNS (DDNS)

- DDNS automatically updates DNS records when changes occur (like adding new devices or updating IPs).
- It reduces manual effort and uses authentication for security.

DNS Communication

- **Port 53** is used for DNS queries.
- **UDP** is used for messages smaller than 512 bytes.
- **TCP** is used for larger messages or special cases, like transferring large sets of data.

E-mail

Electronic mail (e-mail) allows people to send and receive messages over the Internet. The architecture of an e-mail system includes several components, each serving a specific purpose. The three main components are:

1. **User Agent (UA):** This is the application used by the sender and receiver to compose and read messages. Examples include Outlook or Gmail.
2. **Message Transfer Agent (MTA):** This is responsible for sending and receiving messages between different systems over the Internet.
3. **Message Access Agent (MAA):** This component allows users to access and retrieve their messages from the server.

Here's a simplified explanation of four scenarios:

1. **Same System (Simple):** Both sender and receiver are on the same system. Alice writes a message and stores it in Bob's mailbox. Bob reads it later. No need for MTAs or MAAs.
2. **Different Systems (Across the Internet):** Alice and Bob are on different systems. Alice's MUA sends a message to an MTA, which transfers the message to Bob's system. Bob retrieves it from his mailbox using his MUA.
3. **Remote Systems (LAN/WAN):** Alice is on a different system (like a LAN or WAN) from Bob. Alice uses her MUA to send the message, which is transferred through MTAs. The message is stored on Bob's server, and he can read it later.
4. **Message Retrieval (Accessing Mail):** After a message reaches Bob's server, Bob uses an MAA to retrieve the message. The MAA pulls the message from the server to Bob's system, and Bob can read it using his MUA.

In the most common setup (the fourth scenario), there are:

- Two user agents (UA).
- Two sets of MTAs (client and server) to send the message.
- An MAA (client and server) to retrieve the message.

This architecture ensures that e-mail works smoothly across different systems and networks.

A **User Agent (UA)** is software used for sending, receiving, and managing email messages. It helps users with composing, reading, replying, forwarding, and organizing messages in mailboxes.

Services Provided by a User Agent:

1. **Composing Messages**: Helps create and edit emails, often with built-in spell and grammar checks.
2. **Reading Messages**: Displays received emails with summaries and provides options to view the full content.

3. **Replying to Messages**: Allows replies to senders or all recipients.
4. **Forwarding Messages**: Lets users send received messages to others.
5. **Handling Mailboxes**: Manages the inbox and outbox for storing received and sent emails.

Types of User Agents:

1. **Command-Driven**: Early email clients requiring keyboard commands (e.g., Mail, Pine).
2. **GUI-Based**: Modern clients with graphical interfaces (e.g., Outlook, Eudora).

Sending and Receiving Mail:

- **Sending Mail**: Composing an email includes the "envelope" (sender/receiver) and "message" (header with details like subject and body).
- **Receiving Mail**: The user agent alerts when new mail arrives and lets users view or delete messages.

Email Address:

- Consists of a **local part** (username) and a **domain name** (mail server).

Mailing List: Allows multiple recipients to be grouped under one name.

MIME (Multipurpose Internet Mail Extensions): A protocol that allows sending non-ASCII data like images, videos, and foreign languages by encoding them into ASCII format.

SMTP (Simple Mail Transfer Protocol): Protocol used for transferring email between servers. It includes commands like **HELO**, **MAIL FROM**, **RCPT TO**, and **DATA** to manage the email transfer process.

Mail Transfer Phases:

1. **Connection Establishment**: Setting up communication between servers.
2. **Mail Transfer**: The actual sending of the email.
3. **Connection Termination**: Closing the connection after email transfer is complete.

File Transfer Protocol (FTP) is a standard method used to transfer files between computers over the internet using TCP/IP. It uses two connections:

1. **Control Connection (Port 21):** Used to send commands and responses (e.g., to log in or request a file).
2. **Data Connection (Port 20):** Used to transfer the actual files.

FTP allows for different types of file transfers, such as:

- **Retrieving files** from the server to the client.
- **Storing files** from the client to the server.
- **Listing files** in a directory.

The data connection opens and closes each time a file is transferred, while the control connection remains open throughout the session.

Communication:

- On the control connection, commands and responses are sent in plain text using the 7-bit ASCII character set.
- On the data connection, files are transferred, and the file type (ASCII, EBCDIC, or binary) is specified to ensure correct formatting.

FTP can also be **anonymous**, meaning users can access publicly available files without needing a personal account. The common username is "anonymous" and password is "guest".

In summary, FTP is a reliable way to transfer files between systems, addressing differences in file formats, structures, and directories.

The **Simple Network Management Protocol (SNMP)** is a tool used to manage and monitor devices like routers, switches, and servers in a network. Here's a simplified explanation:

Concepts:

- **Manager and Agent**:
  - The **manager** is a central system (like a computer) that controls and monitors devices.
  - The **agent** is the device being managed (like a router).
- The **manager** can request information from the **agent**, update settings, or instruct the agent to take specific actions.

How SNMP Works:

1. **Monitoring**:
   The agent keeps track of important data (like how many packets it processes). The manager can ask for this data to check the agent's performance.
2. **Control**:
   The manager can send commands to the agent, such as resetting a device or adjusting its settings.
3. **Alerts**:
   If something unusual happens (like a problem or error), the agent can send a warning (called a **trap**) to the manager.

Components of SNMP:

SNMP works with two other protocols to manage the network:

1. **SMI (Structure of Management Information)**:
   - Sets the **rules** for naming and defining objects.
   - For example, it decides how variables are named and what type of data they can hold (e.g., numbers, text).
2. **MIB (Management Information Base)**:
   - A **database of objects** that the agent tracks.
   - It defines what information the agent provides, like the number of packets processed or errors detected.
3. **SNMP (Simple Network Management Protocol)**:
   - The protocol that handles communication between the manager and the agent.
   - It uses the rules from SMI and the objects defined in MIB to send and receive information.

Analogy:

Managing a network with SNMP is like writing a program:

- **SMI**: Defines the rules of the "language" (like variable names and data types).
- **MIB**: Declares the variables used in the program.
- **SNMP**: Runs the program by reading, updating, or interpreting the variables.

Example:

If the manager wants to know how many UDP packets an agent has processed:

1. **MIB** identifies the object (variable) that holds this information.
2. **SMI** ensures the object's name and format follow the rules.
3. **SNMP** sends a request to the agent, gets the data, and shows it to the manager.