Module - 5

1. **Introduction to Application Layer:**
   - Introduction, Client-Server Programming
2. **Standard Client Server Protocols:**
   - World Wide Web and HTTP
   - FTP
   - Electronic Mail
   - TELNET
   - Secure Shell (SSH)
   - Domain Name System (DNS)

1.1Introduction:

The Internet was originally designed for the purpose to provide service to users around the world. The protocols in this layer do not provide services to any other protocol in the suite; they only receive services from the protocols in the transport layer. This means that protocols can be removed from this layer easily. New protocols can be also added to this layer as long as the new protocols can use the services provided by one of the transport-layer protocols.

Standard and Nonstandard Protocols

Application-layer protocols that have been standardized and documented by the Internet authority, and that are used in interaction with the Internet are Standard Application-Layer Protocols. A programmer can create a nonstandard application-layer program by writing two programs that provide service to the user by interacting with the transport layer.

Application-Layer Paradigms;

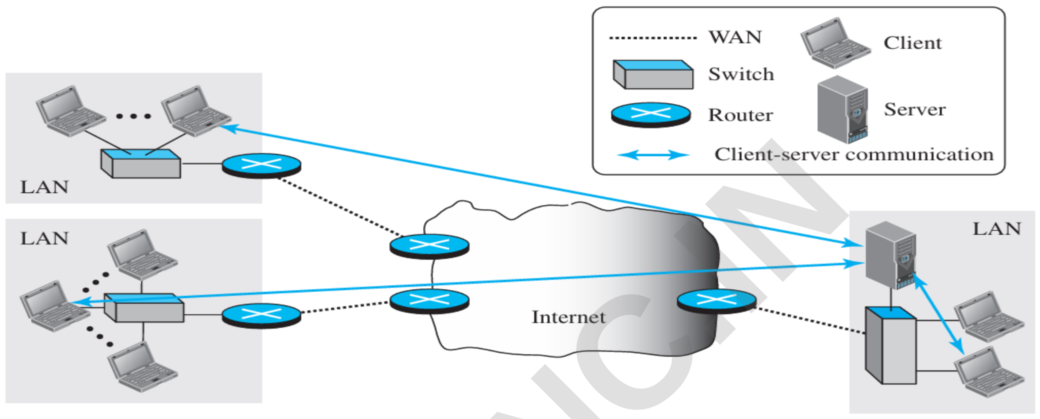
- Client-server paradigm
- Peer-to-peer paradigm

Client-server paradigm:

- The client-server paradigm is a traditional model where a server process provides services to client processes over the Internet.
- The server runs continuously, awaiting client requests, while the client process starts only when service is needed.
- This paradigm is comparable to real-world services, such as a telephone directory center, where the server (directory) is always available, and the client (caller) uses it as needed.
- The roles of the client and server are distinct, requiring separate application programs for each type of service.

- A key issue with this model is the centralized load on the server, which requires powerful infrastructure to handle multiple simultaneous client connections.
- Another challenge is the cost and maintenance of powerful servers, necessitating a return on investment for the service provider.
- Despite its limitations, this paradigm is still widely used in services like the World Wide Web (HTTP), FTP, SSH, and email.

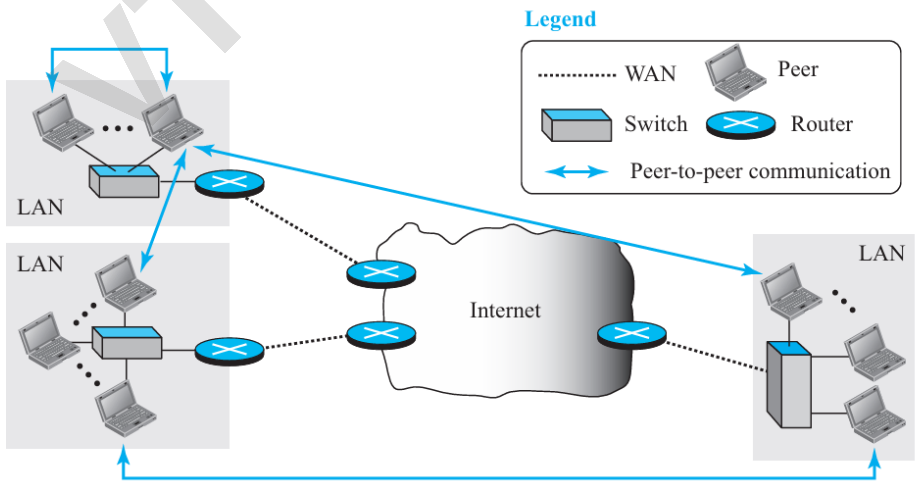
Figure 25.2 *Example of a client-server paradigm*



Peer-to-Peer:

- The peer-to-peer (P2P) paradigm eliminates the need for a central server by distributing responsibilities among peers.

Figure 25.3 *Example of a peer-to-peer paradigm*



- In this model, computers can act as both service providers and consumers, even simultaneously.
- Examples of P2P applications include Internet telephony (e.g., Skype) and file sharing (e.g., BitTorrent), where devices communicate directly without a dedicated server.

- The P2P paradigm is scalable and cost-effective, reducing the need for expensive, always-on servers.
- Security challenges arise as distributed communication is harder to secure than centralized server-based communication.
- Applicability issues exist since not all applications or users are suited for the P2P model. For instance, some users may resist adopting a peer-based implementation of the web.
- Despite challenges, P2P is widely used in modern applications like BitTorrent, Skype, IPTV, and Internet telephony.

1.2 Client-Server Programming

- In the client-server paradigm, communication occurs between two running application programs: client and server.
- The client initializes communication by sending a request, while the server waits to receive requests, processes them, and sends responses back to the client.
- The server must be continuously running to handle client requests, whereas the client program runs only when needed and stops after completing its tasks.
- For effective communication, the server program must start before the client program is executed.
- The server's lifetime is infinite, operating continuously, while the client's lifetime is finite, limited to the duration required to send requests and process responses.

Application Programming Interface

- **Client-server communication** requires a process at the application layer to interact with the lower layers of the TCP/IP protocol suite to establish a connection, exchange data, and terminate the connection.
- To achieve this, a **set of programming instructions** called an **Application Programming Interface (API)** is used. APIs act as an intermediary between the application layer process and the operating system.
- **APIs** provide the necessary commands to manage tasks such as opening connections, transmitting data, receiving responses, and closing connections.
- These APIs are integrated into the operating system, which implements the lower four layers of the TCP/IP protocol suite, enabling seamless communication between processes over the Internet.
- Some of the most common APIs for process communication include:
    - **Socket Interface**
    - **Transport Layer Interface (TLI)**
    - **STREAM**

These APIs simplify the development of networked applications by abstracting the complexities of lower-layer operations.

Socket:

- Sockets are treated like other sources as shown in figure 25.5 (e.g., keyboard, files) or sinks (e.g., monitor, files) in programming languages such as C, C++, or Java.
- They allow programs to send or receive data over a network just as they would with standard I/O operations.

Figure 25.4 *Position of the socket interface*

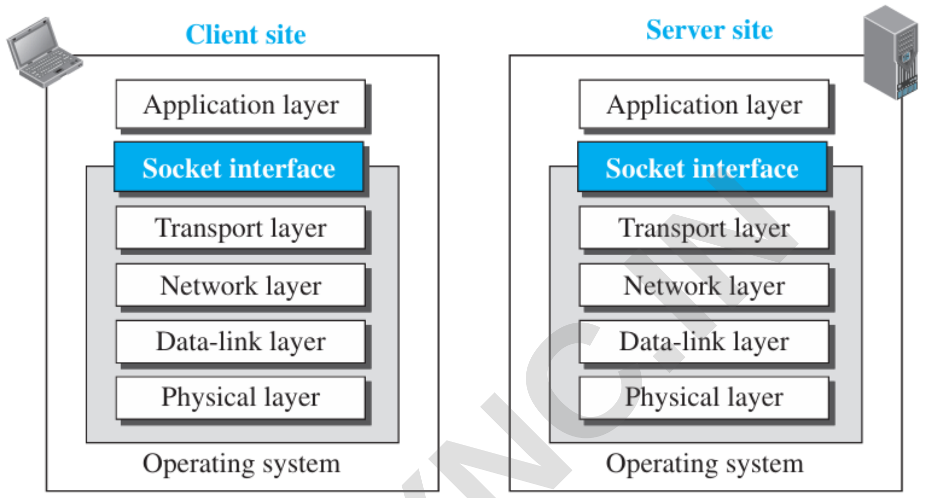


Figure 25.5 *Sockets used the same way as other sources and sinks*

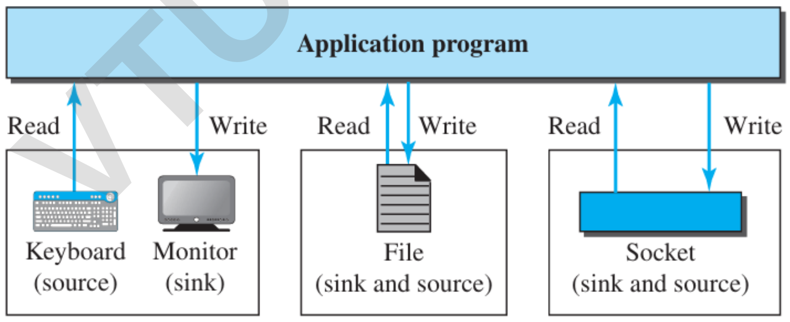
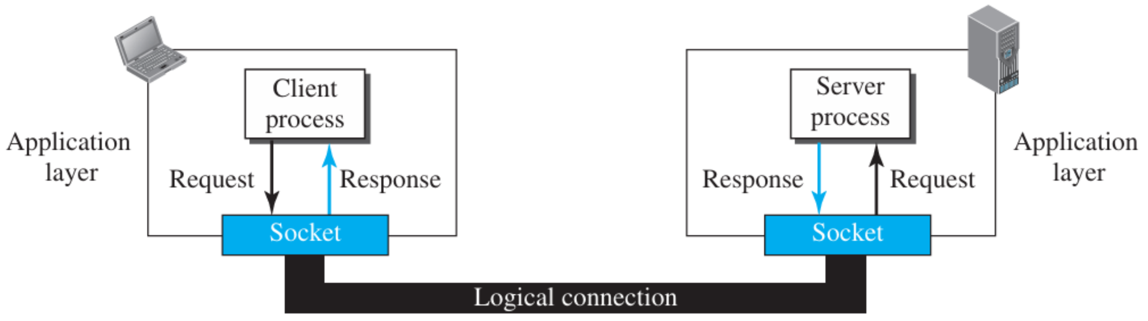


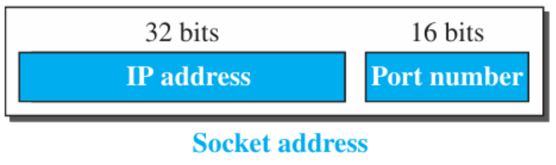
Figure 25.6 *Use of sockets in process-to-process communication*



- Sockets are not physical entities like files or terminals but are abstract objects created by application programs.
- Sockets act as communication endpoints for exchanging requests and responses between client and server processes.
- **Client's Perspective**: The socket appears as the entity that receives responses to its requests.
- **Server's Perspective**: The socket appears as the entity that sends requests requiring responses.
- Each socket must have source and destination addresses properly defined for communication.
- After setup, the operating system and the embedded TCP/IP protocol handle data transfer between the sockets.
- Sockets simplify programming by acting like terminals or files, enabling developers to use familiar instructions for reading and writing data.
- Communication occurs between two sockets, one on each end, as illustrated in diagrams like Figure 25.6.

Socket Addresses

Figure 25.7 *A socket address*



A socket address combines two identifiers:

- **IP Address**: Uniquely identifies a computer on the Internet.
- **Port Number:** Identifies the specific application process (client or server) on the computer.

Finding Socket Addresses

Server Site: The server needs a local (server) and a remote (client) socket address for communication.

Local Socket Address for Server:

- The operating system provides the server's local IP address.
- If the server process is standard, it uses a well-known port number (e.g., HTTP uses port 80).
- For non-standard servers, the designer can assign a port number within the range specified by the Internet authority.

Remote Socket Address for Server:

- The remote socket address corresponds to the client's address making the connection.
- It is determined dynamically from the client's request packet.
- The server's local socket address remains fixed, but the remote address changes with each client interaction.

Client Site The client also needs a local (client) and a remote (server) socket address for communication.

Local Socket Address for Client:

- The client's local IP address is provided by the operating system.
- A temporary (ephemeral) port number is assigned by the OS for each client process, ensuring it is unique.

Remote Socket Address for Client:

- The client must know the server's socket address to initiate communication.
- If the client knows both the server's IP address and port number (e.g., for testing custom programs), it can directly connect.
- If only the port number is known, the IP address is resolved using the Domain Name System (DNS), which maps server names (e.g., URLs) to their IP addresses.

DNS functions like a telephone directory, mapping server names to IP addresses, enabling clients to find servers for communication.

Using Services of the Transport Layer

Application layer processes communicate via transport layer protocols since there is no direct physical communication. Common transport-layer protocols in the TCP/IP suite are UDP, TCP, and SCTP.

UDP Protocol:

- Provides connectionless, unreliable, and message-oriented datagram service.
- Each message is independent, with no logical connection between packets.
- Suitable for applications prioritizing simplicity and speed over reliability, such as multimedia and management applications.

TCP Protocol:

- Provides connection-oriented, reliable, and byte-stream service.
- Uses a handshake to establish communication parameters, ensures data continuity through byte numbering, and supports flow and congestion control.
- Ideal for applications requiring long messages and reliability, but it lacks message-oriented boundaries.

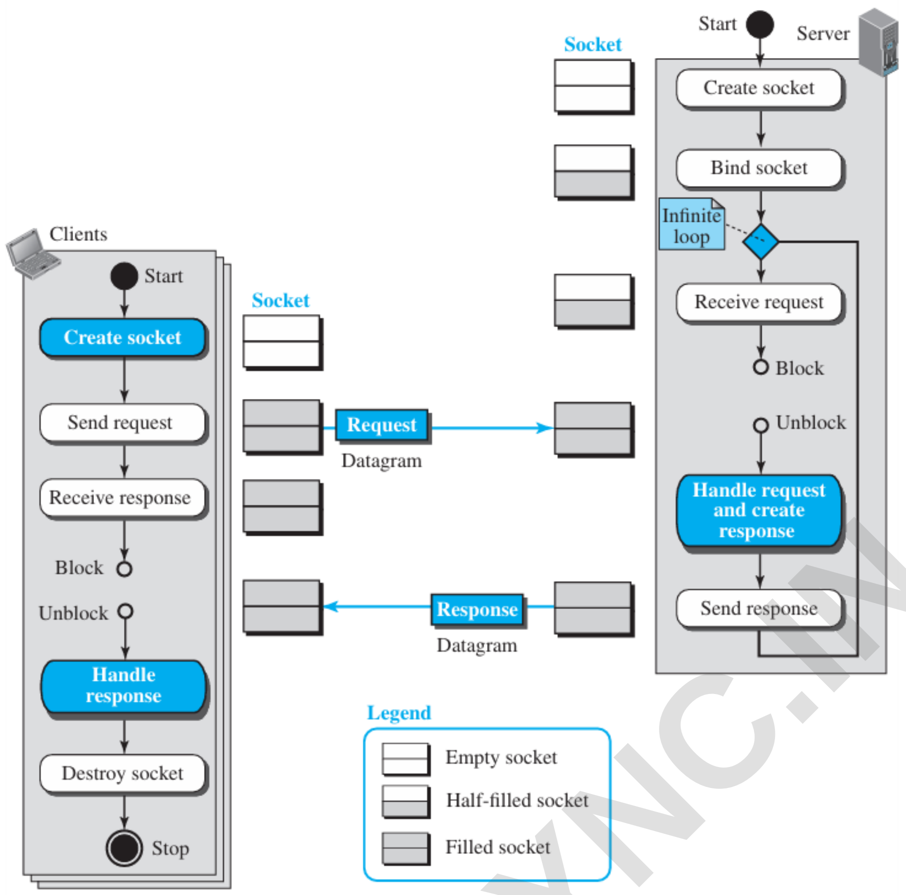
SCTP Protocol:

- Combines the benefits of TCP and UDP by offering connection-oriented, reliable, and message-oriented services.
- Supports multi-streaming for multiple network-layer connections, maintaining communication even during partial network failures.
- Suitable for applications needing reliability and multi-streaming capabilities**.**

Iterative Communication: Handles clients one at a time, in a serial manner. Iterative servers are simple and work well for short transactions, but can build up queues for longer transactions.

Concurrent Communication: Handles multiple clients simultaneously. Concurrent servers can create child processes for each client, allowing them to process requests without waiting for other transactions to complete.

- **Iterative Communication Using UDP**



Servers can respond iteratively (one client request at a time) or concurrently (multiple requests simultaneously). In an iterative server, requests are handled sequentially in a first-in, first-out (FIFO) manner.

- UDP provides connectionless communication, with no connection establishment or termination.
- Each client request is treated as a separate entity, even if sent by the. The socket cre ated at the server site lasts forever; the socket created at the client site is closed (destroyed) when the client process terminates. same client. In UDP communication, the client and server use only one socket each. Figure 25.8 shows the lifetime of the sockets in the server and client processes.
- Different clients use different sockets, but the server creates only one socket and changes only the remote socket address each time a new client makes a connection. This is logical, because the server does know its own socket address, but does not know the socket addresses of the clients who need its services; it needs to wait for the client to connect before filling this part of the socket address.
- There are multiple clients, but only one server. Each client is served in each iteration of the loop in the server. Note that there is no connection establishment or connection termination. Each client sends a single datagram and receives a single datagram. In other words, if a client wants to send two datagrams, it is considered as two clients for the server. The second datagram needs to wait for its turn. The diagram also shows the status of the socket after each action.

Server Process:

- A server makes a passive open, in which it becomes ready for the communication, but it waits until a client process makes the connection creating an empty socket bound to a well-known port.
- The server then issues a receive request command, which blocks until it receives a request from a client. The server waits for client requests, processes them, and sends responses in an infinite loop.
- After each iteration, the server's socket address resets partially, awaiting a new request.

Client Process:

- A client makes an active open, creating an empty socket and sending a request.
- The client waits (blocks) for the server's response, processes it, and then destroys the socket.

S**ocket Usage**:

- The server uses a single socket throughout its lifetime, dynamically updating the remote socket address for each new client.
- Each client creates and destroys its socket during communication.

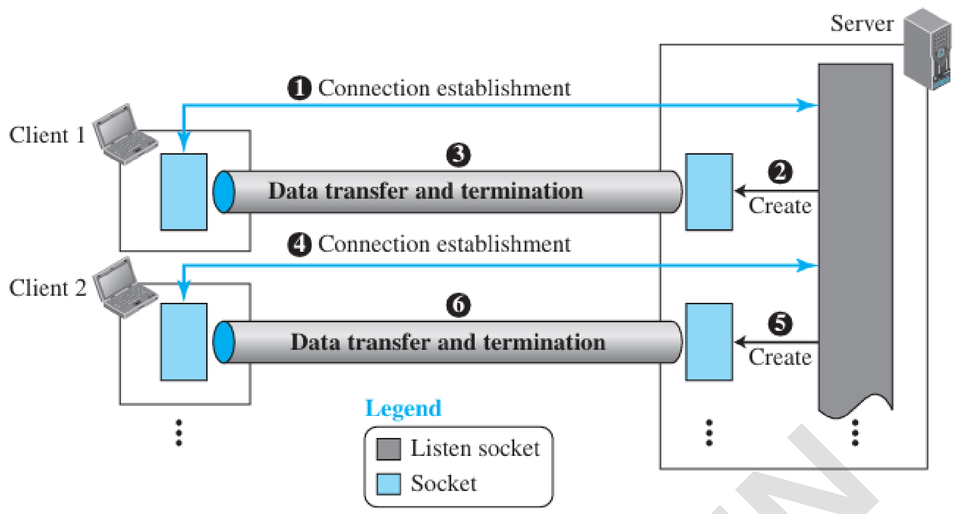
Iterative Communication Using TCP

Before sending or receiving data, a connection needs to be established between the client and the server. After the connection is established, the two parties can send and receive chunks of data as long as they have data to do so.

- **Sockets Used in TCP**

The TCP server uses two different sockets :one for connection establishment (***listen socket***) and the other for data transfer **(Socket).** The reason for having two types of sockets is to separate the connection phase from the data exchange phase. A server uses a listen socket to listen for a new client trying to establish connection. After the connection is established, the server creates a socket to exchange data with the client and finally to terminate the connection. The client uses only one socket for both connection establishment and data exchange (see Figure 25.10).

Figure 25.10 *Sockets used in TCP communication*


Server
Client 1
1 Connection establishment
3
Data transfer and termination
2
Create
4 Connection establishment
Client 2
6
Data transfer and termination
5
Create
Legend
Listen socket
Socket

Flow Diagram:

Figure 25.11 shows a simplified flow diagram for iterative communication using TCP. There are multiple clients, but only one server. Each client is served in each iteration of the loop.

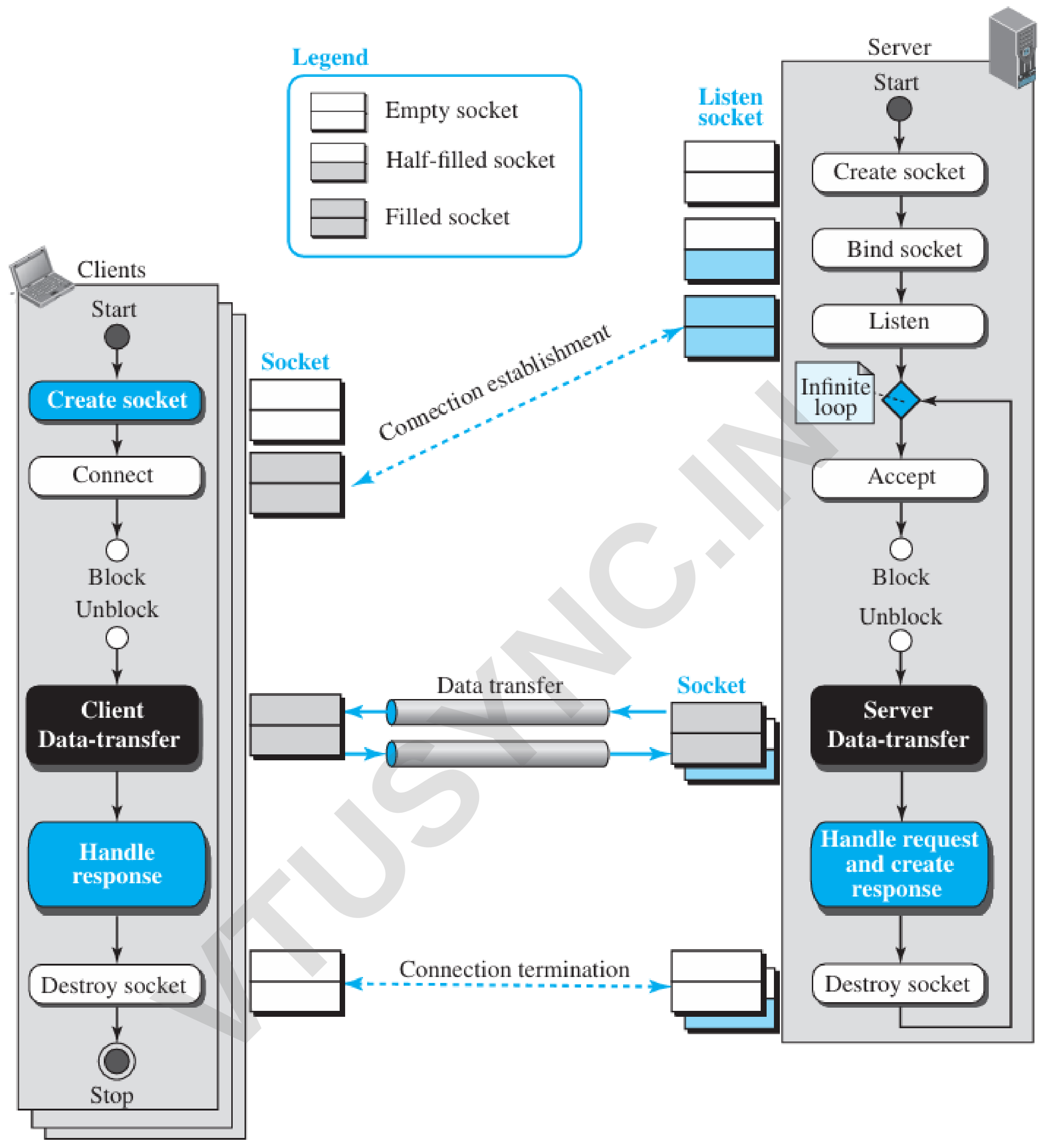
Server Process:

- In Figure 25.11, the TCP server process, like the UDP server process, creates a socket and binds it, but these two commands create the listen socket to be used only for the connection establishment phase.
- The server process then calls the listen procedure, to allow the operating system to start accepting the clients, completing the connection phase, and putting them in the waiting list to be served. The server process now starts a loop and serves the clients one by one.
- In each iteration, the server process issues the accept procedure that removes one client from the waiting list of the connected clients for serving. If the list is empty, the accept procedure blocks until there is a client to be served.
- When the accept procedure returns, it creates a new socket for data transfer. The server process now uses the client socket address obtained during the connection establishment to fill the remote socket address field in the newly created socket. At this time the client and server can exchange data.

Client Process:

The client flow diagram is almost similar to the UDP version except that the client data-transfer box needs to be defined for each specific case.

Figure 25.11 *Flow diagram for iterative TCP communication*



Application Layer overview:

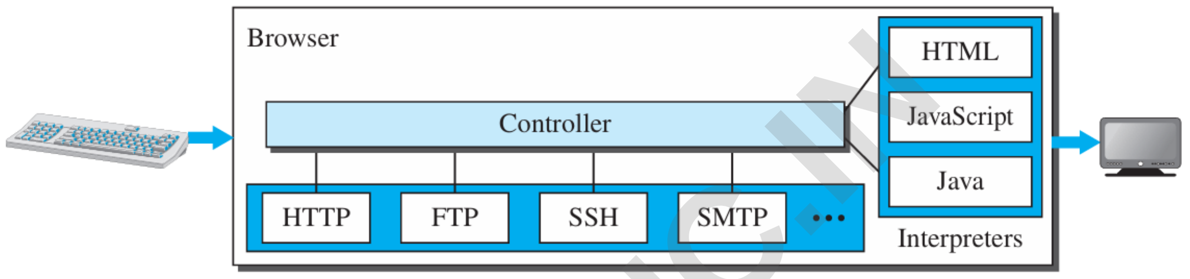
- Application layer determines how a specific user application should use network.
- Application layer is built on transport layer and provides network services to user application.
- Application layer defines and performs such applications as e-mail; file transfers, remote access to computers, real-time video conferencing.
- Application layer has its own software dependencies; when a new application is developed its software must run on multiple machines.

Standard Client Server Protocols:

1. **World Wide Web:** It is the repository of globally distributed web pages, connected by links. The two key aspects are:
    a. **Distributed**: Web servers worldwide can add and share web pages without central server overload.
    b. **Linked**: Hypertext enables one web page to refer to another anywhere in the world.

Web Client (Browser) : A web browser is an application for accessing websites. Each browser usually consists of three parts: a **controller**, **client protocols**, and **interpreters.**

Figure 26.2 *Browser*



The controller receives input from the keyboard or the mouse and uses the client programs to access the document. After the document has been accessed, the controller uses one of the interpreters to display the document on the screen. The client protocol can be one of the protocols described later, such as HTTP or FTP. The interpreter can be HTML, Java, or JavaScript, depending on the type of document. Some commercial browsers include Internet Explorer, Netscape Navigator, and Firefox.

Web Server: The web page is stored at the server. Each time a request arrives, the corresponding document is sent to the client. To improve efficiency, servers normally store requested files in a cache in memory; memory is faster to access than a disk. A server can also become more efficient through multithreading or multiprocessing. In this case, a server can answer more than one request at a time. Some popular web servers include Apache and Microsoft Internet Information Server.

Uniform Resource Locator (URL): A web page, as a file, needs to have a unique identifier to distinguish it from other web pages. To define a web page, we need three identifiers: **Protocol**, **host, port, and path.**

- **Protocol:** The first identifier is the abbreviation for the client-server program that we need in order to access the web page. Ex: HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol).
- **Host:** The host identifier can be the IP address of the server or the unique name given to the server. IP addresses can be defined in dotted decimal notation, the name is normally the domain name that uniquely defines the host, such as forouzan.com.

- **Port:** The port, a 16-bit integer, is normally predefined for the client-server application. For example, if the HTTP protocol is used for accessing the web page, the well-known port number is 80. However, if a different port is used, the number can be explicitly given.
- **Path**: The path identifies the location and the name of the file in the underlying operating system. The format of this identifier normally depends on the operating system. In UNIX, a path is a set of directory names followed by the file name, all separated by a slash. For example, /top/next/last/myfile is a path that uniquely defines a file named myfile, stored in the directory last, which itself is part of the directory next, which itself is under the directory top. In other words, the path lists the directories from the top to the bottom, followed by the file name.

Web Documents: Web documents are broadly classified into **static**, **dynamic**, and **active** documents.

- **Static Documents:** Fixed-content documents stored on a server.
  - **Characteristics**:
    - Content is determined at creation and cannot be altered by users.
    - A copy is sent to the client, viewable via a browser.
  - **Technologies Used**:
    - Created using languages like **HTML**, **XML**, **XSL**, and **XHTML**.

- **Dynamic Documents:** Created by the server in real-time upon a browser's request.
  - **Characteristics**:
    - Content varies with each request, e.g., displaying the current date/time.
    - The server runs a program or script to generate the content dynamically.
  - **Technologies Used**:
    - **Common Gateway Interface (CGI)** (historical).
    - Modern options:
      - **Java Server Pages (JSP)** – Java-based scripting.
      - **Active Server Pages (ASP)** – Microsoft's Visual Basic-based scripting.
      - **ColdFusion** – Embeds **SQL** queries in HTML.

- **Active Documents:** Require scripts or programs to run at the client site (browser).
  - **Characteristics**:
    - Used for applications like animation or user interaction.
    - Program/script is executed on the client's device.
  - **Technologies Used**:
    - **Java Applets**: Pre-compiled Java programs sent in binary (bytecode) format.
    - **JavaScript**: Downloaded and executed directly on the client browser.

2. **HyperText Transfer Protocol (HTTP):** The HyperText Transfer Protocol (HTTP) is used to define how the client-server programs can be written to retrieve web pages from the Web.

An HTTP client sends a request; an HTTP server returns a response. The server uses the port number 80; the client uses a temporary port number.

- **Hypertext and Multiple Requests:** Retrieving web page objects often involves multiple requests and responses, especially when objects are hosted on different servers.
- **TCP Connections**:
- For objects on different servers, a new TCP connection is required for each object.
- For objects on the same server, there are two approaches:
- **Nonpersistent Connection**: A new TCP connection for each object.
- **Persistent Connection**: A single TCP connection is used to retrieve all objects.
- **HTTP Versions**:
- HTTP (pre-1.1): Defaults to nonpersistent connections.
- HTTP 1.1 and beyond: Defaults to persistent connections, though users can modify this behavior.

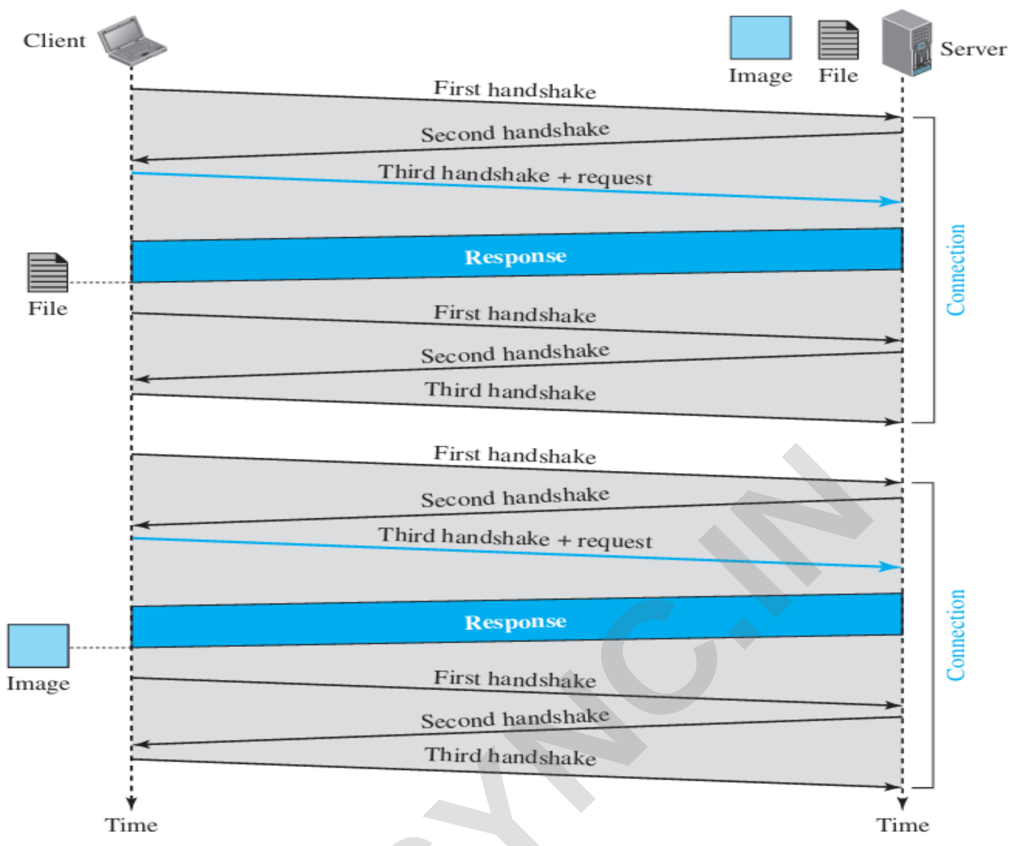
Nonpersistent Connections: In a nonpersistent connection, one TCP connection is made for each request/response. The following lists the steps in this strategy:

1. The client opens a TCP connection and sends a request.
2. The server sends the response and closes the connection.
3. The client reads the data until it encounters an end-of-file marker; it then closes the connection.

In this strategy, if a file contains links to N different pictures in different files (all located on the same server), the connection must be opened and closed N + 1 times. The nonpersistent strategy imposes high overhead on the server because the server needs N + 1 different buffers each time a connection is opened.

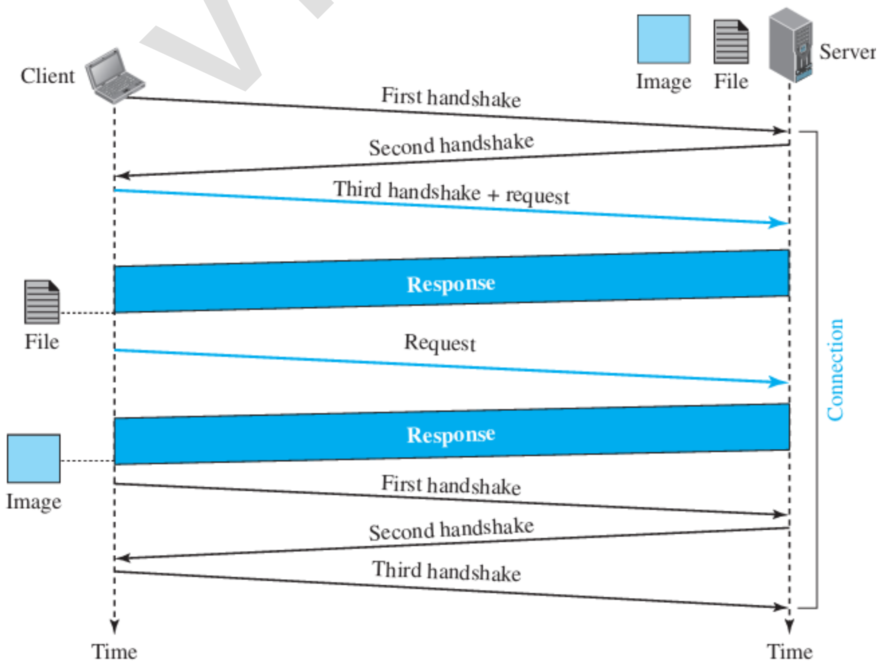
Figure 26.3 shows an example of a nonpersistent connection. The client needs to access a file that contains one link to an image. The text file and image are located on the same server. Here we need two connections. For each connection, TCP requires at least three handshake messages to establish the connection, but the request can be sent with the third one. After the connection is established, the object can be transferred. After receiving an object, another three handshake messages are needed to terminate the connection. This means that the client and server are involved in two connection establishments and two connection terminations. If the transaction involves retrieving 10 or 20 objects, the round trip times spent for these hand shakes add up to a big overhead. When we describe the client-server programming at the end of the chapter, we will show that for each connection the client and server need to allocate extra resources such as buffers and variables. This is another burden on both sites, but especially on the server site.

Figure 26.3 *Example 26.3*



Persistent Connections:

Figure 26.4 *Example 26.4*

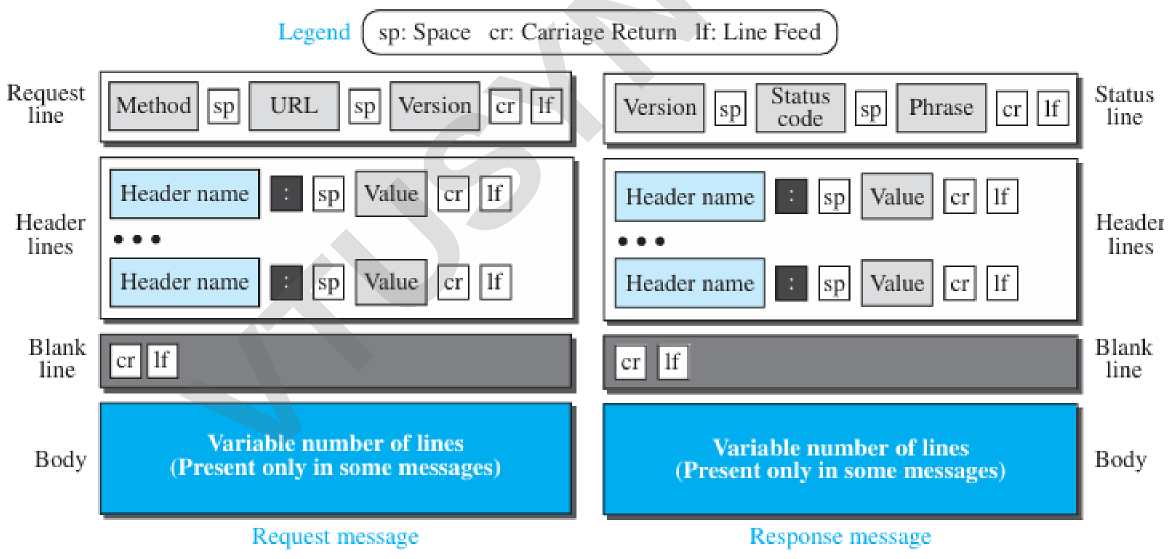


- The server keeps the connection open for additional requests after a response is sent.
- The connection is closed either on client request or after a timeout.
- Typically, the sender includes the data length in the response.
- When the data length is unknown (e.g., for dynamically generated documents), the server signals this by closing the connection after sending the data, indicating the end.
- Persistent connections save time and resources by reducing the need for multiple connection setups and terminations.
- Only one set of buffers and variables is maintained per connection, and the round-trip time for establishing connections is eliminated.

Figure 26.4 shows the same scenario as in Example 26.3, but using a persistent connection. Only one connection establishment and connection termination is used, but the request for the image is sent separately.

Message Formats

Figure 26.5 *Formats of the request and response messages*



The HTTP protocol defines the format of request and response messages, structured in four sections. In request messages, the first section is the request line, while in response messages, it is the status line. The other three sections share the same names in both message types but differ in content. These similarities are limited to naming conventions, with distinct details in each message type.

Request Message:

- The HTTP request message begins with a **request line** containing three fields: **method**, **URL**, and **version**, separated by spaces and terminated with carriage return and line feed.

- **Methods** define the request type:
    - **GET**: Retrieves data; body is empty.
    - **HEAD**: Retrieves metadata or tests URL validity; body is empty.
    - **PUT**: Uploads a new web page to the server (if permitted).
    - **POST**: Sends data to modify or add to a web page.
    - **TRACE**: Debugging; echoes the request back.
    - **DELETE**: Deletes a web page (if permitted).
    - **CONNECT**: Reserved for proxy server use.
    - **OPTIONS**: Retrieves properties of a web page.
- The **URL** specifies the address and name of the web page.
- The **version** indicates the HTTP protocol version, e.g., 1.1.
- **Header lines** may follow, providing additional information from the client to the server. Each header has a name, colon, space, and value.
- The **body** is optional and contains data or files to be sent, typically with PUT or POST methods.

Table 26.1 *Methods*

Method	*Action*
GET	Requests a document from the server
HEAD	Requests information about a document but not the document itself
PUT	Sends a document from the client to the server
POST	Sends some information from the client to the server
TRACE	Echoes the incoming request
DELETE	Removes the web page
CONNECT	Reserved
OPTIONS	Inquires about available options

Table 26.2 *Request header names*

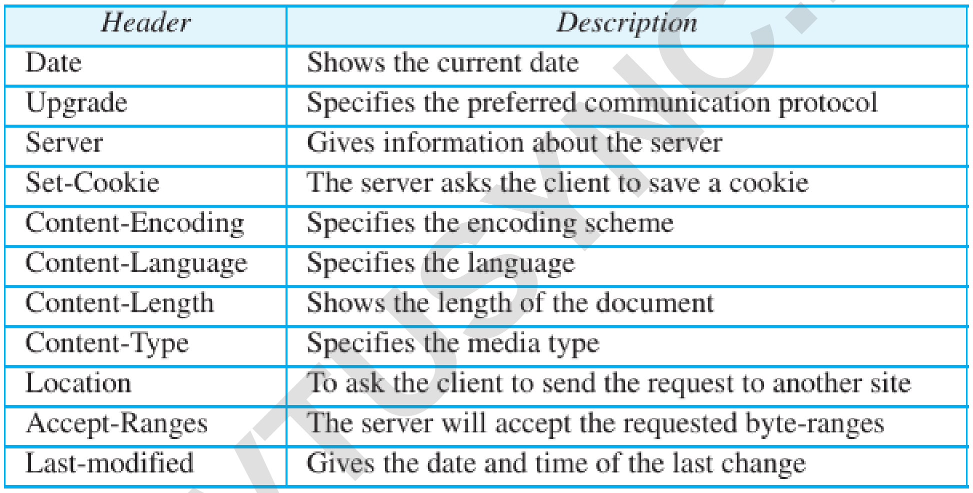
Header	*Description*
User-agent	Identifies the client program
Accept	Shows the media format the client can accept
Accept-charset	Shows the character set the client can handle
Accept-encoding	Shows the encoding scheme the client can handle
Accept-language	Shows the language the client can accept
Authorization	Shows what permissions the client has
Host	Shows the host and port number of the client
Date	Shows the current date
Upgrade	Specifies the preferred communication protocol
Cookie	Returns the cookie to the server (explained later)
If-Modified-Since	If the file is modified since a specific date

Response Message:

- The HTTP response message consists of a **status line**, header lines, a blank line, and sometimes a body.

- The **status line** includes three fields separated by spaces:
- **Version**: Specifies the HTTP protocol version (e.g., 1.1).
- **Status code**: A three-digit number indicating the request status:
    - **100 range**: Informational.
    - **200 range**: Success.
    - **300 range**: Redirection to another URL.
    - **400 range**: Client-side error.
    - **500 range**: Server-side error.
- **Status phrase**: A textual explanation of the status code.
- **Response header lines** provide additional information from the server to the client, formatted as a header name, colon, space, and value.
- The **body**, if present, contains the document or data being sent to the client; it is absent in error responses.

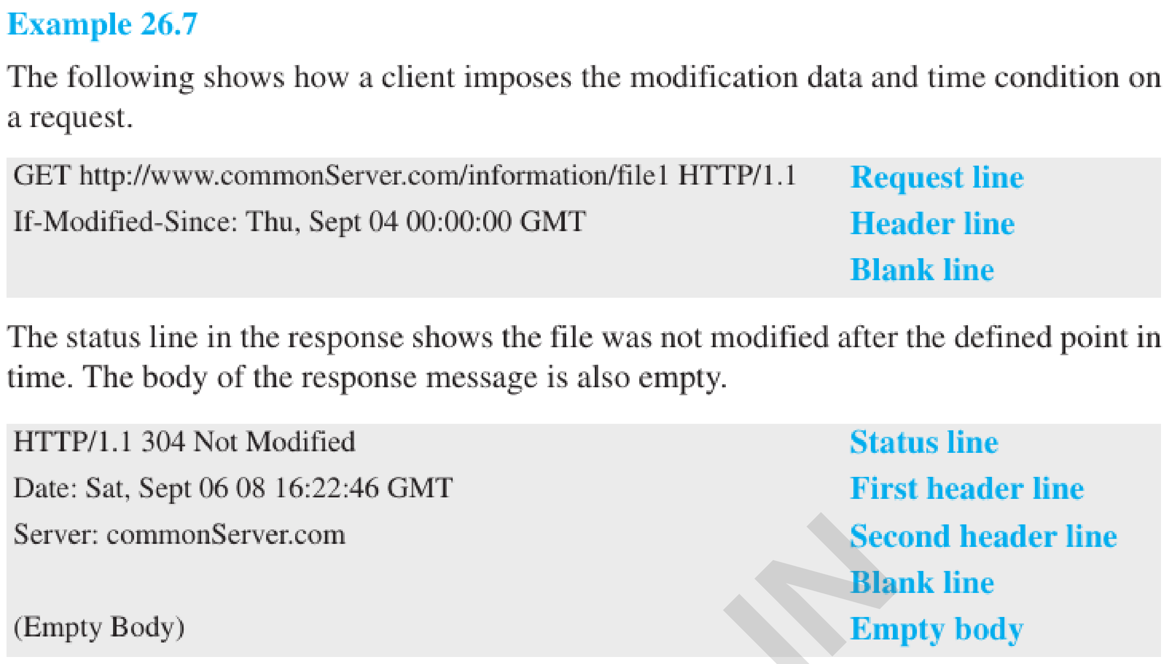
Table 26.3 *Response header names*



Header	*Description*
Date	Shows the current date
Upgrade	Specifies the preferred communication protocol
Server	Gives information about the server
Set-Cookie	The server asks the client to save a cookie
Content-Encoding	Specifies the encoding scheme
Content-Language	Specifies the language
Content-Length	Shows the length of the document
Content-Type	Specifies the media type
Location	To ask the client to send the request to another site
Accept-Ranges	The server will accept the requested byte-ranges
Last-modified	Gives the date and time of the last change

Conditional Request:

- A client can include conditions in its HTTP request.
- **Common Condition - Time/Date:** The client uses the **If-Modified-Since** header to request the page only if it has been modified after a specified date and time.
- This mechanism helps reduce unnecessary data transfer by avoiding the delivery of unmodified content.



Example 26.7

The following shows how a client imposes the modification data and time condition on a request.

GET http://www.commonServer.com/information/file1 HTTP/1.1	**Request line**
If-Modified-Since: Thu, Sept 04 00:00:00 GMT	**Header line**
	Blank line

The status line in the response shows the file was not modified after the defined point in time. The body of the response message is also empty.

HTTP/1.1 304 Not Modified	**Status line**
Date: Sat, Sept 06 08 16:22:46 GMT	**First header line**
Server: commonServer.com	**Second header line**
	Blank line
(Empty Body)	**Empty body**

Cookies: Cookies are small pieces of data stored on the user's device by a web browser. They are created by websites to store information about the user's session or preferences as HTTP is a **stateless protocol**.

Creating and Storing Cookies:

1. When a server receives a request from a client, it stores information about the client in a file or a string. The information may include the domain name of the client, the contents of the cookie (information the server has gathered about the client such as name, registration number, and so on), a timestamp, and other information depend ing on the implementation.
2. The server includes the cookie in the response that it sends to the client.
3. When the client receives the response, the browser stores the cookie in the cookie directory, which is sorted by the server domain name.

Functionality of Cookies:

- Cookies are stored on the client side and included in requests to the server if a match is found.
- Servers use cookies to identify returning clients without revealing their content to the browser or user.

Applications of Cookies:

- **E-commerce:**
    - Track shopping cart items (e.g., item details and prices) and update the cookie with new selections.
    - Retrieve the cookie for calculating the total charge at checkout.
- **Restricted Access:**
    - Sites send cookies to registered users during their first visit.

- On subsequent visits, only clients with valid cookies can access restricted content.

- **Personalized Web Portals:**
  - Cookies store user preferences for favorite pages.
  - On revisit, the server uses the cookie to personalize the user experience.
- **Advertising:**
  - Advertising agencies use cookies to track user interactions with banners across websites.
  - These cookies build a user profile for targeted ads and may be sold, raising privacy concerns.

Privacy Concerns:

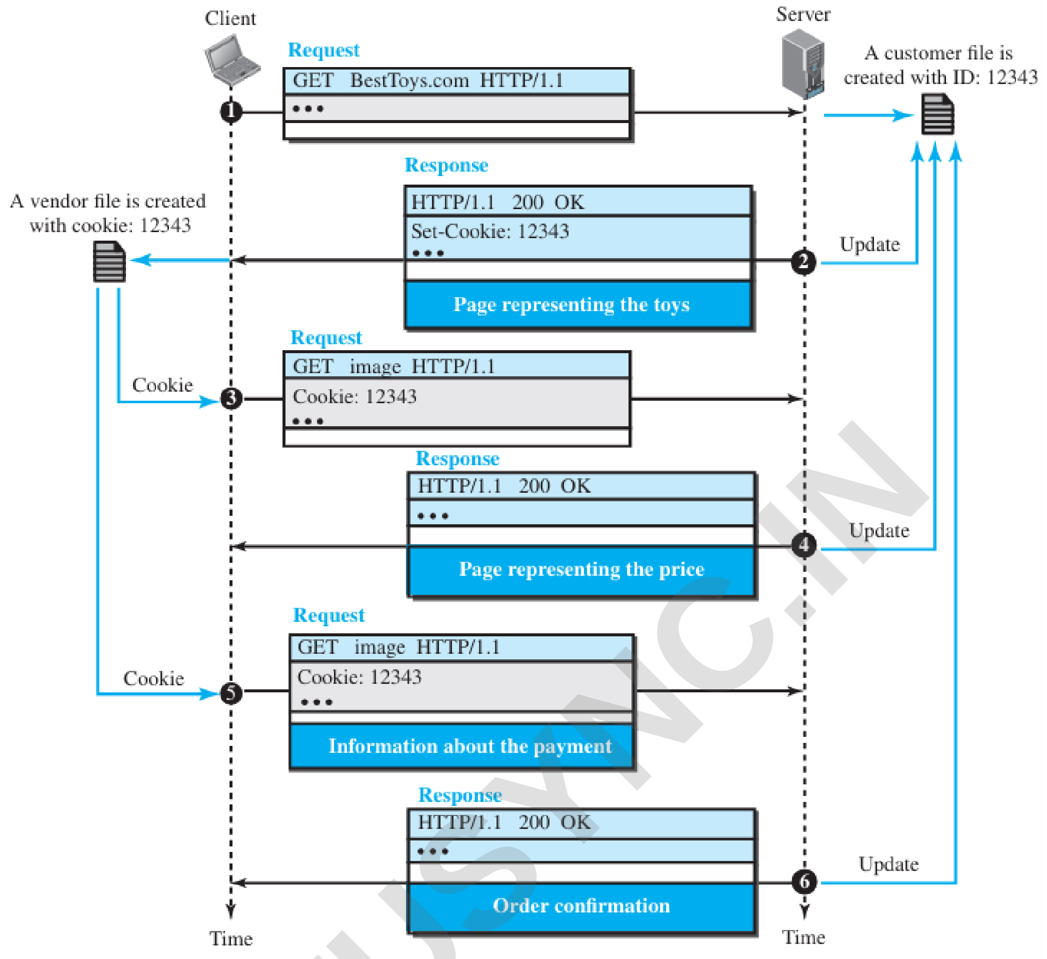
- Cookies used by advertising agencies for tracking user behavior are controversial.
- There is a need for regulations to protect user privacy.

Cookies help the server maintain continuity and provide personalized shopping experiences, even during intermittent interactions.

Use of Cookies in an Electronic Store (Scenario) as shown in daigram 26.8

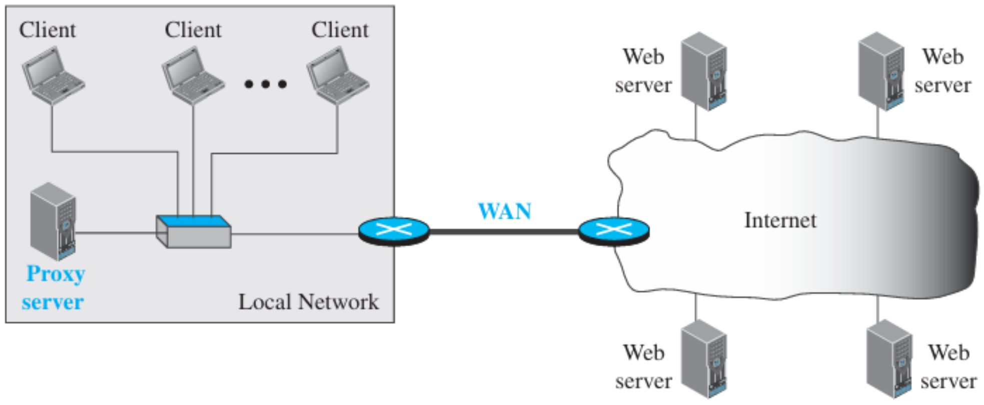
1. **Shopping Cart Creation:**
   - The store server creates an empty shopping cart (list) for the client and assigns a unique ID (e.g., 12343).
   - The server sends a response containing product images and links, along with a Set-Cookie header storing the cart ID.
2. **Cookie Storage on the Client:**
   - The browser saves the cookie (ID 12343) in a file named after the store (e.g., BestToys).
   - The cookie is not disclosed to the shopper.
3. **Toy Selection by the Shopper:**
   - When a toy is selected, the client sends a request to the server, including the cookie (ID 12343) in the Cookie header.
   - The server uses the cookie to identify the shopper and retrieves their shopping cart.
4. **Updating the Shopping Cart:**
   - The selected toy is added to the cart on the server-side.
   - The server sends a response with the updated cart details, including the total price.
5. **Payment Processing:**
   - The shopper provides payment details (e.g., credit card information) and sends a new request with the same cookie value (12343).
   - The server retrieves the shopping cart, processes the payment, and sends a confirmation response.
6. **Client Information Storage:**
   - Additional client information is stored on the server for future interactions.

Figure 26.8 *Example 26.8*



Web Caching: Proxy Servers:

Figure 26.9 *Example of a proxy server*



- A proxy server stores cached copies of responses to frequently accessed requests.
- The HTTP client sends its request to the proxy server first.
- The proxy server checks its cache:
  - If a response is found, it acts as a **server** and sends the response directly to the client.
  - If no response is found, it acts as a **client**, forwards the request to the target server, and stores the received response for future use.

Benefits of Proxy Servers:

- Reduces Load: Decreases the demand on the original web server.
- Improves Latency: Speeds up response time for cached requests.
- Minimizes Traffic: Limits the amount of data flowing between the client and the server.

Locations of Proxy Servers:

- **Client-Side Proxy:** Installed on individual computers for small-scale caching.
- **LAN Proxy:** Used within companies or organizations to manage requests for a local network.
- **ISP Proxy:** Installed by Internet Service Providers to handle requests for multiple customers.

Example Scenario (Figure 26.9):

- In a campus or company network:
  - HTTP requests from clients are routed to the proxy server.
  - If the proxy server has the cached page, it sends it directly to the client.
  - Otherwise, it forwards the request to the web server, caches the response, and delivers it to the client.

Cache Update :

- Proxy servers need strategies to determine how long to retain cached responses before deletion or replacement.
- One approach involves maintaining a list of sites that update content infrequently, such as a news agency updating its page daily.
- Proxy servers can retrieve such content once and keep it until the next expected update.
- Another method uses headers indicating the last modification time, allowing the proxy server to estimate the validity of the cached information.

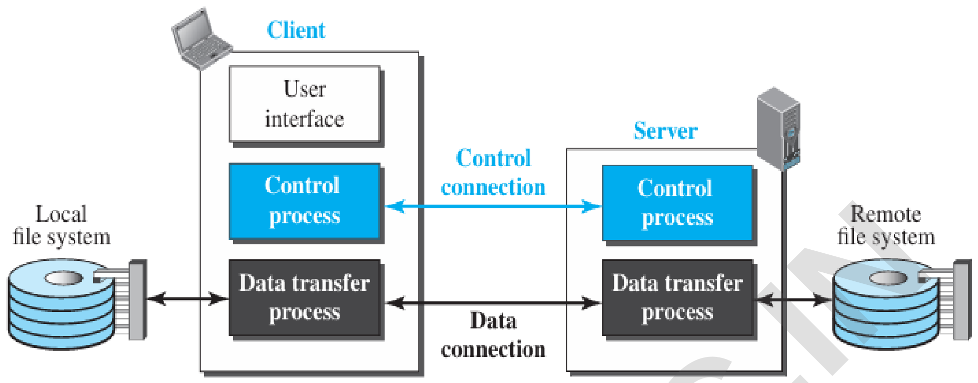
3. FTP

FTP (File Transfer Protocol) is a standard TCP/IP protocol for copying files between hosts, designed to handle challenges like differing file name conventions, data representations, and directory structures. FTP is more suitable than HTTP for transferring large files or files in various formats.

- FTP operates with two connections: **control connection** and **data connection**.
- Control connection uses TCP port 21 and remains active during the entire FTP session, handling commands and responses.

- Data connection uses TCP port 20, opening and closing for each file transfer, enabling efficient file handling.
- The client has three components: user interface, client control process, and client data transfer process.
- The server has two components: server control process and server data transfer process.

Figure 26.10 *FTP*



Control Connection:

- Communication occurs through ASCII-based commands and responses, similar to TELNET.
- Commands are sent from the client, and responses are sent by the server.
- Commands are uppercase ASCII strings, optionally followed by arguments.
- Responses have a numeric code and text explanation.

Data Connection:

- Data connection is initiated by the client with a passive open using an ephemeral port.
- The client uses the PORT command to notify the server of the port, and the server opens the connection using port 20.

Data Transfer:

- The data connection handles file transfers, with the following attributes defining transfer:
  - File Type: ASCII, EBCDIC, or image files.
  - Data Structure: File structure (stream of bytes), record structure (divided into records), or page structure (divided into pages with headers).
  - Transmission Mode: Stream mode (default continuous stream), block mode (data in blocks with headers), or compressed mode.

Efficiency:

- FTP separates commands (handled via control connection) and data transfers (handled via data connection) to enhance efficiency.
- File transfer includes retrieving files (server to client), storing files (client to server), and directory listing (server to client).

Table 26.4 *Some FTP commands*

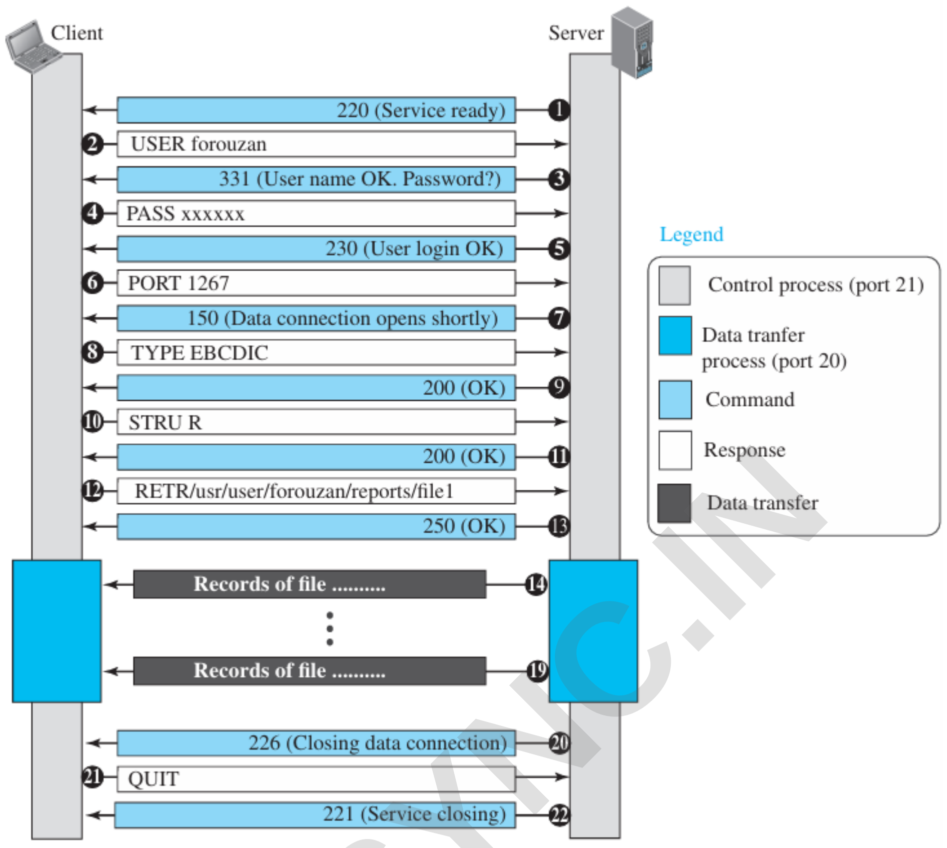
Command	*Argument(s)*	*Description*
ABOR		Abort the previous command
CDUP		Change to parent directory
CWD	Directory name	Change to another directory
DELE	File name	Delete a file
LIST	Directory name	List subdirectories or files
MKD	Directory name	Create a new directory
PASS	User password	Password
PASV		Server chooses a port
PORT	Port identifier	Client chooses a port
PWD		Display name of current directory
QUIT		Log out of the system
RETR	File name(s)	Retrieve files; files are transferred from server to client
RMD	Directory name	Delete a directory
RNFR	File name (old)	Identify a file to be renamed
RNTO	File name (new)	Rename the file
STOR	File name(s)	Store files; file(s) are transferred from client to server
STRU	**F**, **R**, or **P**	Define data organization (**F**: file, **R**: record, or **P**: page)
TYPE	**A**, **E**, **I**	Default file type (**A**: ASCII, **E**: EBCDIC, **I**: image)
USER	User ID	User information
MODE	**S**, **B**, or **C**	Define transmission mode (**S**: stream, **B**: block, or **C**: compressed

Table 26.5 *Some responses in FTP*

Code	*Description*	*Code*	*Description*
125	Data connection open	**250**	Request file action OK
150	File status OK	**331**	User name OK; password is needed
200	Command OK	**425**	Cannot open data connection
220	Service ready	**450**	File action not taken; file not available
221	Service closing	**452**	Action aborted; insufficient storage
225	Data connection open	**500**	Syntax error; unrecognized command
226	Closing data connection	**501**	Syntax error in parameters or arguments
230	User login OK	**530**	User not logged in

Example 26.10 Figure 26.11 shows an example of using FTP for retrieving a file. The figure shows only one file to be transferred. The control connection remains open all the time, but the data connection is opened and closed repeatedly. We assume the file is transferred in six sections. After all records have been transferred, the server control process announces that the file transfer is done. Since the client control process has no file to retrieve, it issues the QUIT command, which causes the service connection to be closed.

Figure 26.11 *Example 26.10*



Client
Server
220 (Service ready)
USER forouzan
331 (User name OK. Password?)
PASS xxxxxx
230 (User login OK)
PORT 1267
150 (Data connection opens shortly)
TYPE EBCDIC
200 (OK)
STRU R
200 (OK)
RETR/usr/user/forouzan/reports/file1
250 (OK)
Records of file ....
Records of file ....
226 (Closing data connection)
QUIT
221 (Service closing)
Legend
Control process (port 21)
Data tranfer process (port 20)
Command
Response
Data transfer

Example 26.11

The following shows an actual FTP session that lists the directories. The colored lines show the responses from the server control connection; the black lines show the commands sent by the client. The lines in white with black background show data transfer.

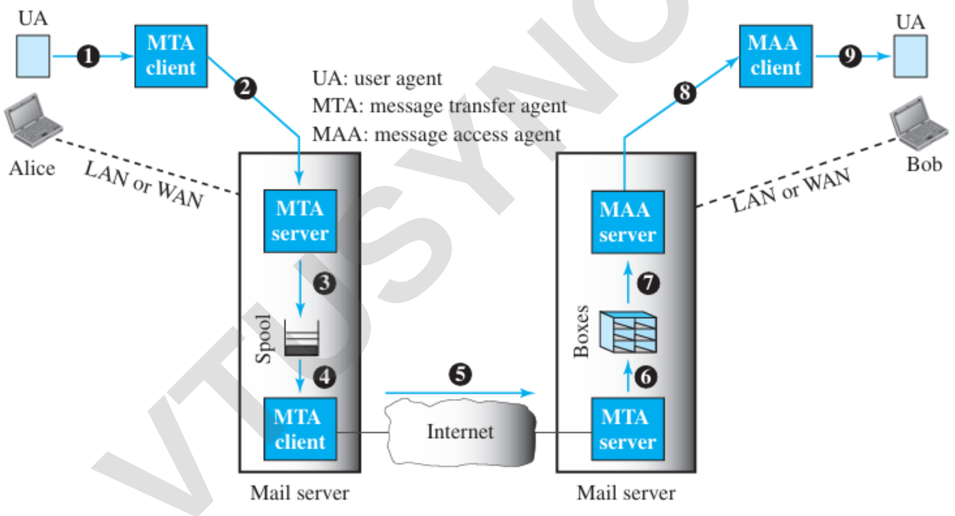
```
$ ftp voyager.deanza.fhda.edu
Connected to voyager.deanza.fhda.edu.
220 (vsFTPd 1.2.1)
530 Please login with USER and PASS.
Name (voyager.deanza.fhda.edu:forouzan): forouzan
331 Please specify the password.
Password:*********
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
227 Entering Passive Mode (153,18,17,11,238,169)
150 Here comes the directory listing.
drwxr-xr-x    2    3027    411    4096   Sep 24    2002   business
drwxr-xr-x    2    3027    411    4096   Sep 24    2002   personal
drwxr-xr-x    2    3027    411    4096   Sep 24    2002   school
226 Directory send OK.
ftp> quit
221 Goodbye.
```

Security for FTP: The FTP protocol was designed when security was not a big issue. Although FTP requires a password, the password is sent in plaintext (unencrypted), which means it can be intercepted and used by an attacker. The data transfer connection also transfers data in plain text, which is insecure. To be secure, one can add a Secure Socket Layer between the FTP application layer and the TCP layer. In this case FTP is called SSL-FTP.

4. ELECTRONIC MAIL

- E-mail is a one-way transaction; responses are optional and separate one-way transactions.
- Users run client programs on demand, while intermediate mail servers handle client-server interactions.
- **E-Mail Architecture:**

Figure 26.12 *Common scenario*



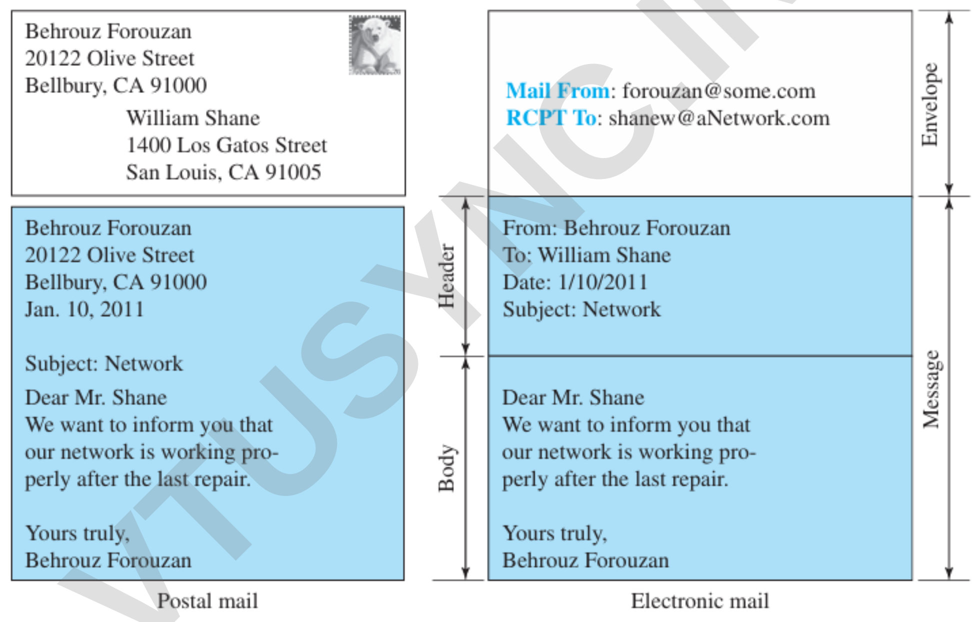
  - Alice (sender) and Bob (receiver) use a User Agent (UA) to send/receive messages via their respective mail servers.
  - Mail servers use Message Transfer Agents (MTAs) to send messages between them.
  - Bob retrieves messages from the server using a Message Access Agent (MAA).
  - Mailboxes (special files) store received messages, and queues (spools) temporarily hold messages waiting to be sent.
  - The electronic mail system needs two UAs, two pairs of MTAs (client and server), and a pair of MAAs (client and server).
- **Key Agents in E-Mail:**
  1. **User Agent (UA):**
     - Handles composing, sending, receiving, replying, and forwarding messages.
     - Types:

        - Command-driven (e.g., mail, pine).
        - GUI-based (e.g., Eudora, Outlook).

  2. **Message Transfer Agent (MTA):**
      - Handles sending messages between mail servers using the SMTP protocol.
      - Operates as a push mechanism, transferring messages automatically.
  3. **Message Access Agent (MAA):**
      - Handles retrieving messages from the mail server using a pull mechanism.

- **Mail Sending Process**:
    - Mail includes an envelope (sender/receiver addresses) and a message (header and body).
    - Header includes sender, recipient, subject, and additional details.
    - Body contains the actual content to be read.

Figure 26.13 *Format of an e-mail*



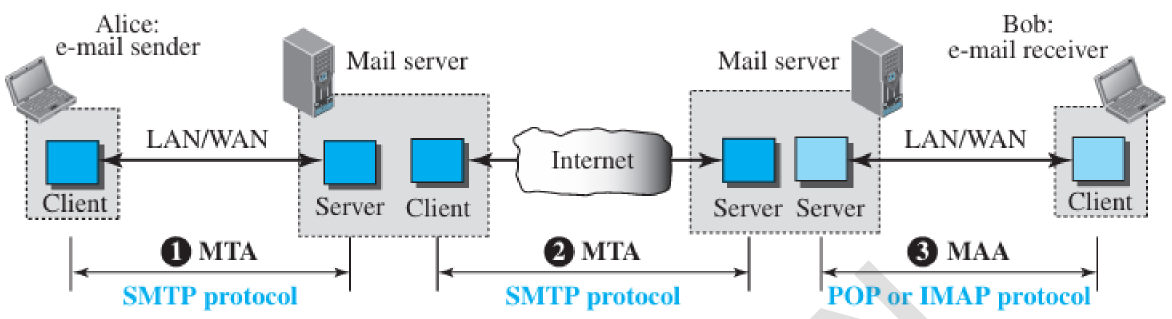
- **Mail Receiving Process:**
    - The UA informs the user about new mail and displays a summary of messages.
    - The user selects messages to view their content.
- **E-Mail Addresses:**
    - Composed of a local part (user mailbox) and a domain name, separated by @.
    - Domain names represent mail servers, defined by DNS or logical organizational names.
- Mailing Lists:
    - Alias names represent multiple addresses; messages are distributed to all addresses in the list.

SMTP (Simple Mail Transfer Protocol):

- Protocol for message transfer between MTAs:
    - Used between the sender's client and sender's mail server.

- Also used between sender and receiver mail servers.
- SMTP exchanges commands and responses, terminated by a two-character end-of-line token.

Figure 26.15 *Protocols used in electronic mail*



Commands and Responses : SMTP uses commands and responses to transfer messages between an MTA client and an MTA server. The command is from an MTA client to an MTA server; the response is from an MTA server to the MTA client. Each command or reply is terminated by a two character (carriage return and line feed) end-of-line token. Commands Commands are sent from the client to the server. The format of a command is shown below:

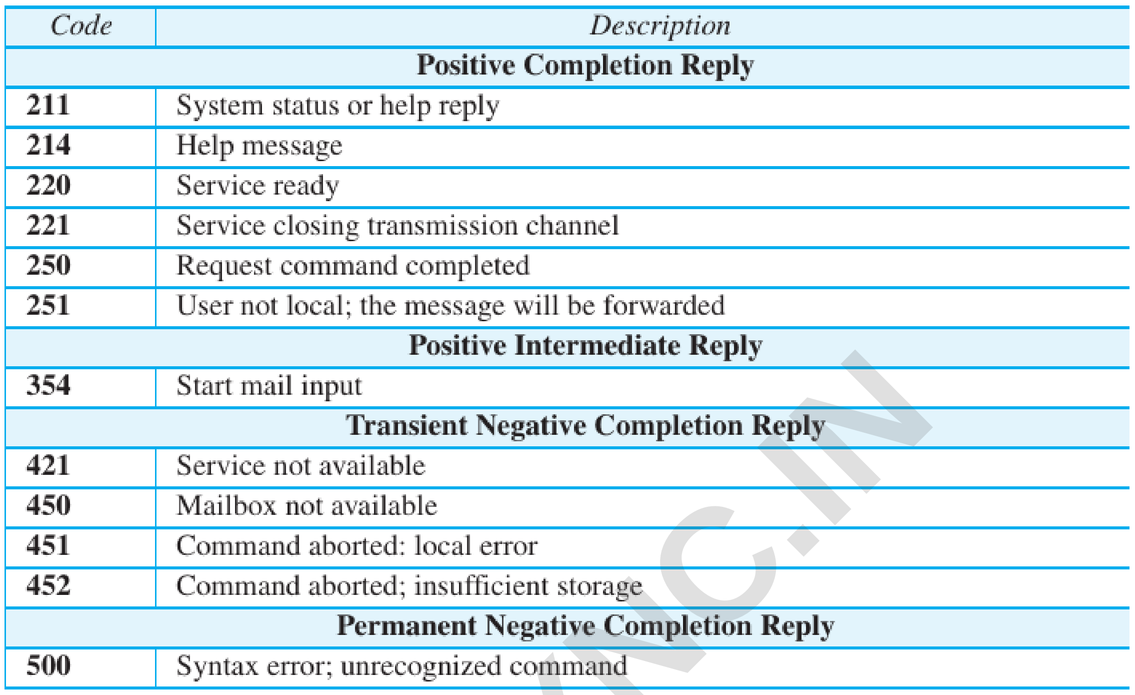
Keyword: argument(s)

Table 26.6 *SMTP commands*

Keyword	*Argument(s)*	*Description*
HELO	Sender's host name	Identifies itself
MAIL FROM	Sender of the message	Identifies the sender of the message
RCPT TO	Intended recipient	Identifies the recipient of the message
DATA	Body of the mail	Sends the actual message
QUIT		Terminates the message
RSET		Aborts the current mail transaction
VRFY	Name of recipient	Verifies the address of the recipient
NOOP		Checks the status of the recipient
TURN		Switches the sender and the recipient
EXPN	Mailing list	Asks the recipient to expand the mailing list
HELP	Command name	Asks the recipient to send information about the command sent as the argument
SEND FROM	Intended recipient	Specifies that the mail be delivered only to the terminal of the recipient, and not to the mailbox
SMOL FROM	Intended recipient	Specifies that the mail be delivered to the terminal *or* the mailbox of the recipient
SMAL FROM	Intended recipient	Specifies that the mail be delivered to the terminal *and* the mailbox of the recipient

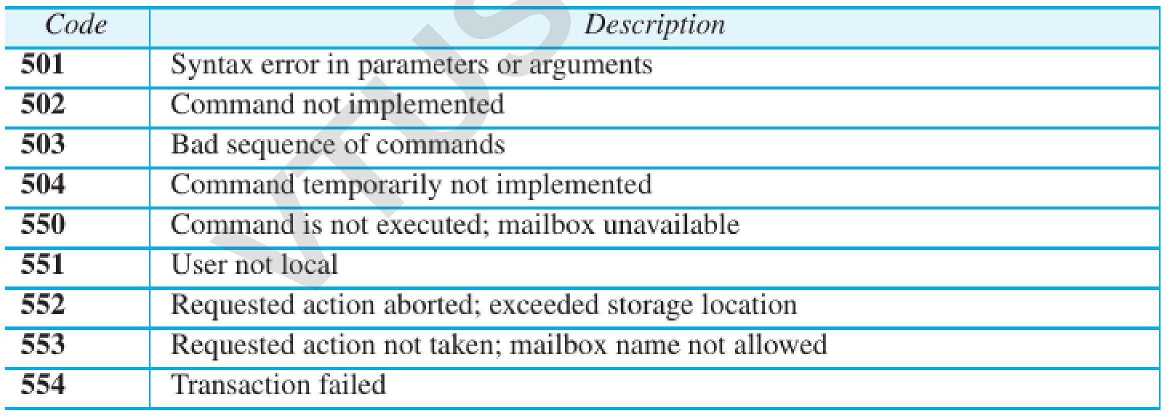
Responses : Responses are sent from the server to the client. A response is a three digit code that may be followed by additional textual information. Table 26.7 shows the most common response types.

Table 26.7 *Responses*



Code	Description
	Positive Completion Reply
211	System status or help reply
214	Help message
220	Service ready
221	Service closing transmission channel
250	Request command completed
251	User not local; the message will be forwarded
	Positive Intermediate Reply
354	Start mail input
	Transient Negative Completion Reply
421	Service not available
450	Mailbox not available
451	Command aborted: local error
452	Command aborted; insufficient storage
	Permanent Negative Completion Reply
500	Syntax error; unrecognized command

Table 26.7 *Responses (continued)*



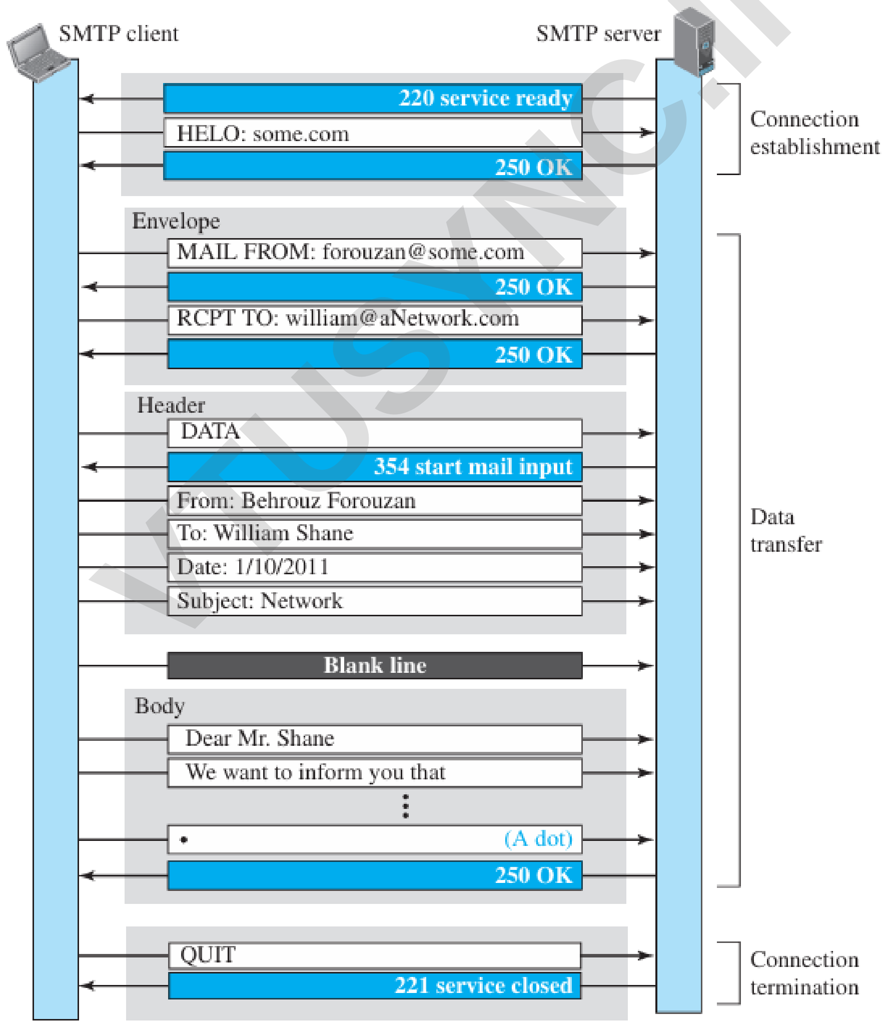
Code	Description
501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command temporarily not implemented
550	Command is not executed; mailbox unavailable
551	User not local
552	Requested action aborted; exceeded storage location
553	Requested action not taken; mailbox name not allowed
554	Transaction failed

- **Connection Establishment:**
  - **Step 1:** SMTP server sends code 220 (service ready) to indicate readiness. If unavailable, it sends 421 (service not available).
  - **Step 2:** Client sends HELO message with its domain name to identify itself.
  - **Step 3:** Server responds with code 250 (request command completed) or another status code.
- **Message Transfer:**
  - **Step 1:** Client sends MAIL FROM message with the sender's email address (mailbox and domain).
  - **Step 2:** Server responds with code 250 (OK) or another appropriate code.

- **Step 3:** Client sends RCPT TO message with the recipient's email address.
- **Step 4:** Server responds with code 250 (OK) or another appropriate code.
- **Step 5:** Client sends DATA message to start the actual message transfer.
- **Step 6:** Server responds with code 354 (start mail input) or another appropriate code.
- **Step 7:** Client sends the email content line by line, with each line ending in a two-character end-of-line token. The message ends with a line containing a single period (.).
- **Step 8:** Server responds with code 250 (OK) or another appropriate code.

Connection Termination : After the message is transferred successfully, the client ter minates the connection. This phase involves two steps.
1. The client sends the QUIT command.
2. The server responds with code 221 or some other appropriate code

Figure 26.16 *Example 26.12*



To show the three mail transfer phases, we show all of the steps described above using the information depicted in Figure 26.16. In the figure, we have separated the messages related to the envelope, header, and body in the data transfer section. Note that the steps in this figure are repeated two times in each e-mail transfer: once from the e-mail sender to the local mail server and once from the local mail server to the remote mail server. The local mail server, after receiving the whole e-mail message, may spool it and send it to the remote mail server at another time.

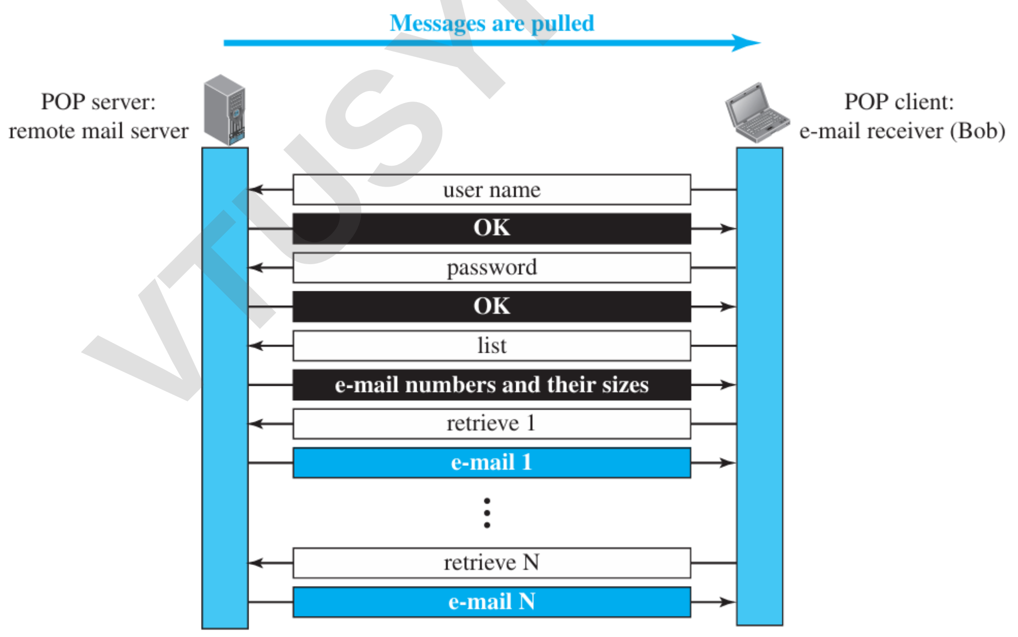
Message Access Agent: POP and IMAP :

Mail Delivery Protocols

1. **SMTP Usage:**
   - Used in the first and second stages of mail delivery.
   - Push protocol: sends messages from the client to the server.
2. **Third Stage Protocol:**
   - Requires a pull protocol: retrieves messages from the server to the client.
   - Involves a Message Access Agent.
   - Common protocols: POP3 and IMAP4.

POP3 (Post Office Protocol, Version 3)

Figure 26.17 *POP3*


Messages are pulled
POP server: remote mail server
POP client: e-mail receiver (Bob)
user name
OK
password
OK
list
e-mail numbers and their sizes
retrieve 1
e-mail 1
retrieve N
e-mail N

1. **Functionality:**
   - Simple but limited.
   - Client software connects to the mail server via TCP port 110.
   - User provides credentials to access the mailbox.
2. **Modes:**
   - Delete Mode: Mail is deleted after retrieval; used on permanent computers.
   - Keep Mode: Mail remains on the server; suitable for accessing mail on temporary devices.

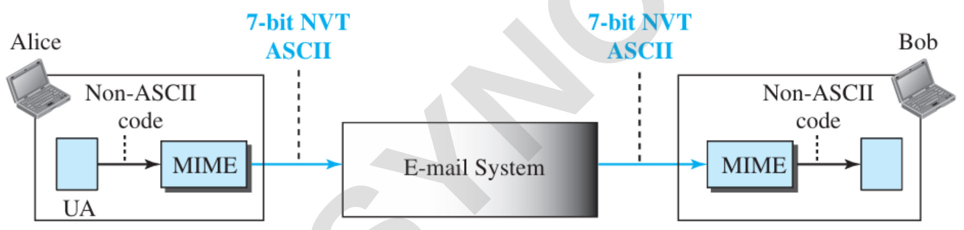
3. **Limitations:**
    - Cannot organize mail on the server (e.g., no folders).
    - Does not allow previewing or searching mail before downloading**.**

IMAP4 (Internet Mail Access Protocol, Version 4)

1. **Features:**
    - More powerful and complex than POP3.
    - Allows header preview and content search before downloading.
    - Supports partial downloads for bandwidth efficiency.
    - Enables mailbox organization:
        - Creating, deleting, and renaming mailboxes.
        - Hierarchical folder structure for mail storage.

MIME (Multipurpose Internet Mail Extensions)

Figure 26.18 *MIME*



1. **Purpose:**
    - Overcomes email limitations (e.g., ASCII-only format).
    - Supports non-ASCII data (e.g., different languages, binary files, multimedia).
2. **Functionality:**
    - Converts non-ASCII data to NVT ASCII for transmission.
    - Converts ASCII data back to original format at the receiving end.
3. **Applications:**
    - Enables sending multimedia and international language content through email.

MIME Headers : MIME defines five headers, as shown in Figure 26.19, which can be added to the origi nal e-mail header section to define the transformation parameters:
MIME-Version : This header defines the version of MIME used. The current version is 1.1. **Content-Type:** This header defines the type of data used in the body of the message. The content type and the content subtype are separated by a slash. Depending on the subtype, the header may contain other parameters. MIME allows seven different types of data, listed in Table 26.8.
Content-Transfer-Encoding

- Purpose: Specifies the encoding method for converting messages into binary (0s and 1s) for transport.
- Encoding Methods (Table 26.9): Five types, with focus on:

1. Base64 Encoding:
    - Divides data into 6-bit chunks.
    - Converts each chunk into one ASCII character (8 bits).
    - Results in a 25% overhead due to redundancy.
    - Suitable for transmitting binary or non-ASCII data.
2. Quoted-Printable Encoding:
    - Sends ASCII characters as is.
    - Encodes non-ASCII characters as three characters:
        - = followed by two hexadecimal digits representing the byte.
    - Example: A non-ASCII byte 9D16 is encoded as =9D.

Content-ID : Provides a unique identifier for the entire message in environments where multiple messages are present.

Table 26.8 *Data types and subtypes in MIME*

Type	*Subtype*	*Description*
Text	Plain	Unformatted
	HTML	HTML format (see Appendix C)
Multipart	Mixed	Body contains ordered parts of different data types
	Parallel	Same as above, but no order
	Digest	Similar to Mixed, but the default is message/RFC822
	Alternative	Parts are different versions of the same message
Message	RFC822	Body is an encapsulated message
	Partial	Body is a fragment of a bigger message
	External-Body	Body is a reference to another message
Image	JPEG	Image is in JPEG format
	GIF	Image is in GIF format
Video	MPEG	Video is in MPEG format
Audio	Basic	Single channel encoding of voice at 8 KHz
Application	PostScript	Adobe PostScript
	Octet-stream	General binary data (eight-bit bytes)

Table 26.9 *Methods for Content-Transfer-Encoding*

Type	*Description*
7-bit	NVT ASCII characters with each line less than 1000 characters
8-bit	Non-ASCII characters with each line less than 1000 characters
Binary	Non-ASCII characters with unlimited-length lines
Base64	6-bit blocks of data encoded into 8-bit ASCII characters
Quoted-printable	Non-ASCII characters encoded as an equal sign plus an ASCII code

Figure 26.20 *Base64 conversion*

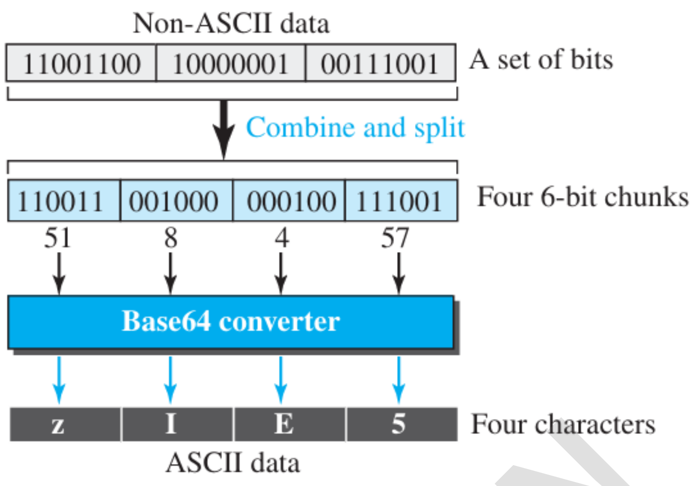
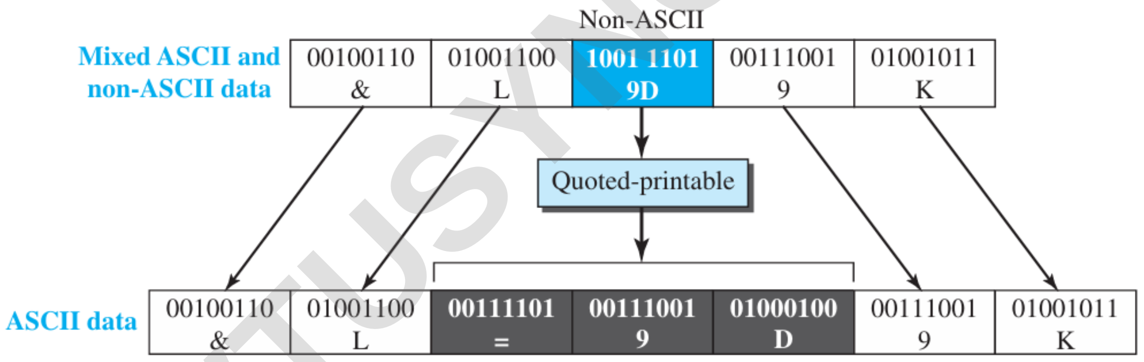


Figure 26.21 *Quoted-printable*



Web Based Mail :

- Web-based mail services (e.g., Hotmail, Yahoo, Gmail) allow users to send and receive emails via web browsers using **HTTP** instead of traditional email client protocols like **POP3** or **IMAP4**.
- Common use cases: Two scenarios based on the type of mail servers used by sender and receiver.

Case I: Traditional to Web-Based Mail Server

1. **Sender (Alice)**:
   - Uses a traditional mail server to send an email to the recipient's mail server via **SMTP**.
2. **Receiver (Bob)**:
   - Has an account on a web-based mail server.
   - Retrieves emails from the web server using **HTTP** instead of POP3 or IMAP4.
   - Workflow:
     - Bob sends an HTTP request to the website to log in.

- After authentication, the list of emails is sent in **HTML format**.
- Individual emails are retrieved using additional HTTP transactions.

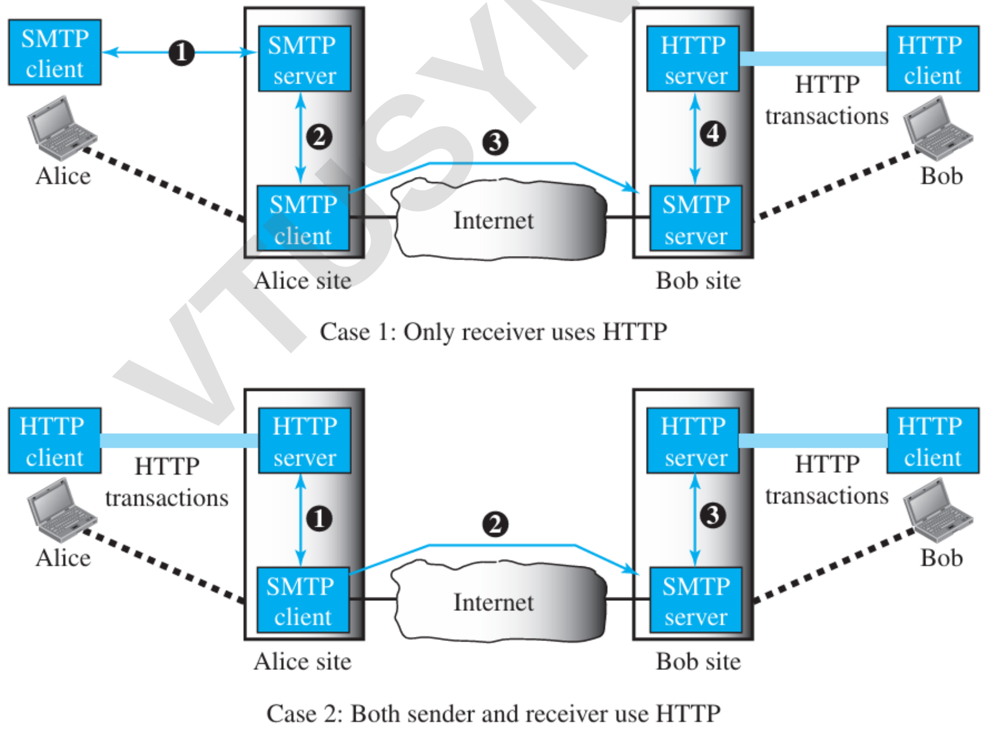
Case II: Web-Based Mail Servers on Both Ends

1. **Sender (Alice)**:
   - Uses HTTP to send an email to her web server.
   - The web server acts as an **SMTP client**, forwarding the email to the recipient's mail server (Bob's server) using **SMTP**.
2. **Receiver (Bob)**:
   - Retrieves emails from his web server using **HTTP** transactions.
3. **Key Protocols**:
   - SMTP: Transfers emails between the sender's and receiver's servers.
   - HTTP: Handles interactions between users and their respective web servers.

Key Differences from Traditional Mail

- **HTTP replaces POP3/IMAP4** for mail retrieval.
- **HTML format** is used for email presentation in the browser.

Figure 26.22 *Web-based e-mail, cases I and II*

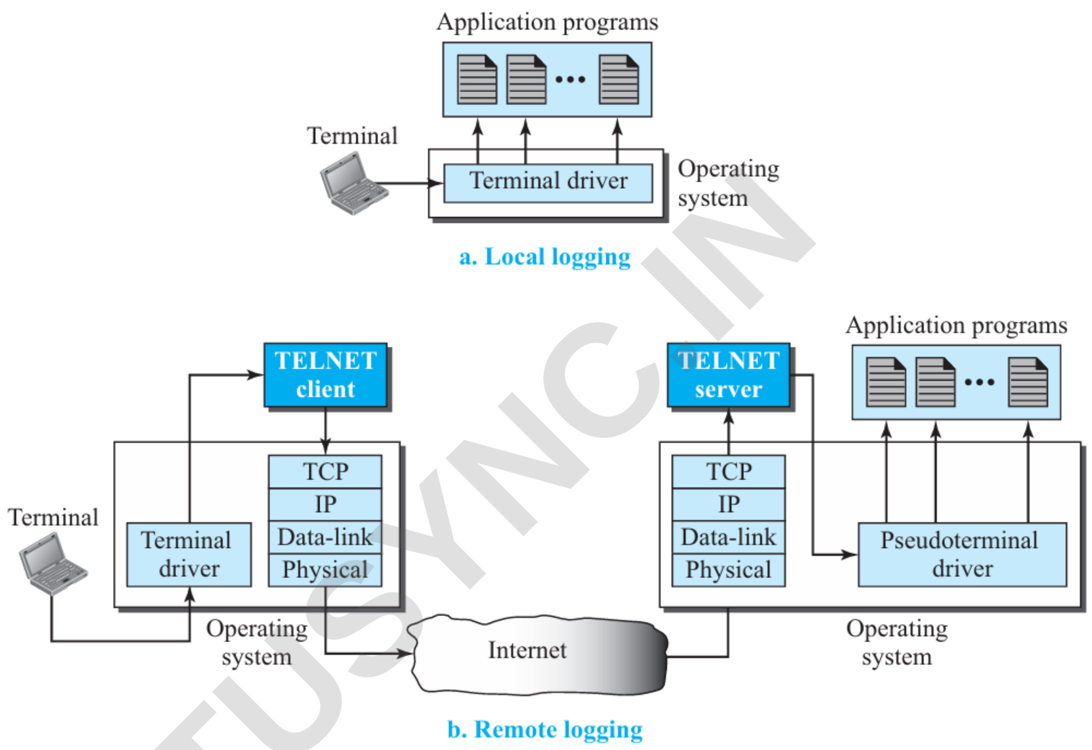


4. **TELNET**

• TELNET (Terminal Network) is a remote logging protocol enabling users to log into a remote server and access its services.

• Purpose: Provides a **generic client/server** pair for accessing remote resources without needing specialized client/server programs for every service (e.g., Java compiler usage on a university server).

Figure 26.23 *Local versus remote logging*



Features of TELNET

1. **Plaintext Communication:**
   - Data, including login credentials, is sent without encryption.
   - Vulnerable to hacking through eavesdropping.
2. **Authentication:**
   - Requires a login name and password to access the remote server.
3. **Use Cases:**
   - Historical: Once widely used for remote access to servers.
   - Current: Used by network administrators for diagnostic and debugging tasks despite security limitations**.**

Limitations

- Lack of encryption makes TELNET susceptible to unauthorized access.
- Replaced by Secure Shell (SSH), which offers encrypted communication.

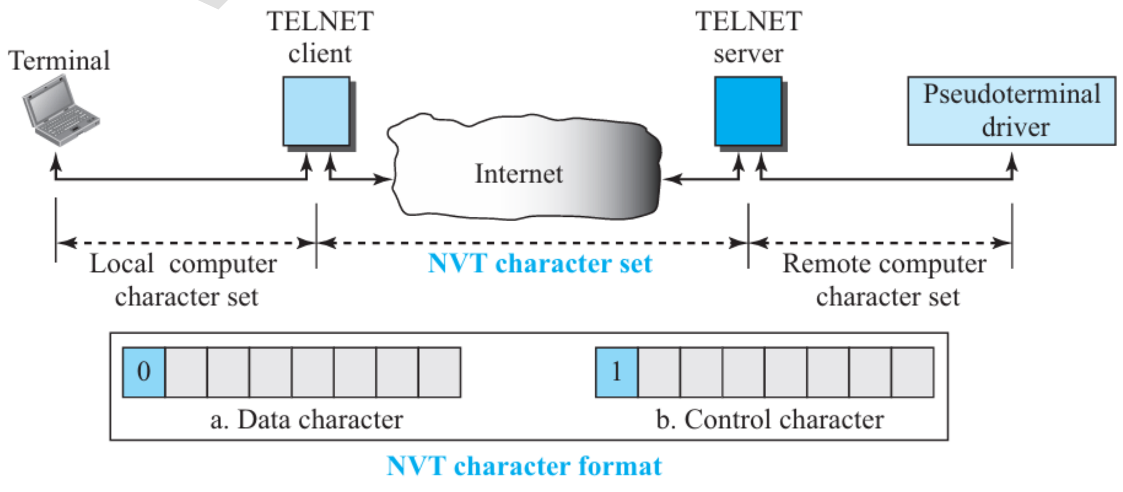
Local vs. Remote Logging

1. **Local Logging**:

   - User logs into a local system via a terminal or terminal emulator.
   - **Process**:
     - Keystrokes are processed by the terminal driver and passed to the OS.
     - OS interprets input and invokes appropriate programs.
2. **Remote Logging**:
   - User logs into a remote system to access applications or utilities.
   - **Process**:
     - Keystrokes are sent to the local OS but not interpreted.
     - Forwarded to TELNET client, converted to **NVT characters**, and transmitted over the Internet.
     - Remote TELNET server receives and converts the NVT characters into the format the remote OS understands.
     - A **pseudoterminal driver** simulates input from a physical terminal, enabling the OS to process the input.

Network Virtual Terminal (NVT) : The mechanism to access a remote computer is complex. This is because every computer and its operating system accepts a special combination of characters as tokens. For example, the end-of-file token in a computer running the DOS operating system is Ctrl+z, while the UNIX operating system recognizes Ctrl+d. We are dealing with heterogeneous systems. If we want to access any remote com puter in the world, we must first know what type of computer we will be connected to, and we must also install the specific terminal emulator used by that computer. TELNET solves this problem by defining a universal interface called the Network Virtual Terminal (NVT) character set. Via this interface, the client TELNET translates characters (data or commands) that come from the local terminal into NVT form and delivers them to the network. The server TELNET, on the other hand, translates data and commands from NVT form into the form acceptable by the remote computer.

Figure 26.24 *Concept of NVT*



1. **Purpose**:

  - Standardizes communication between heterogeneous systems by using a universal character set.
  - Eliminates the need to know the specifics of the remote system or install its terminal emulator.
2. **Functionality**:
  - TELNET client converts local characters/commands to **NVT form** and sends them to the network.
  - TELNET server converts **NVT form** characters/commands into the format used by the remote system.
3. **Character Sets**:
  - **Data Characters (NVT ASCII)**:
    - 8-bit characters; lower 7 bits follow US ASCII; highest bit is 0.
  - **Control Characters**:
    - 8-bit characters with the highest bit set to 1.

TELNET Options

1. **Negotiation**:
  - Client and server negotiate optional features before or during the session.
  - Supports sophisticated terminals with advanced features while providing defaults for simpler terminals.

User Interface

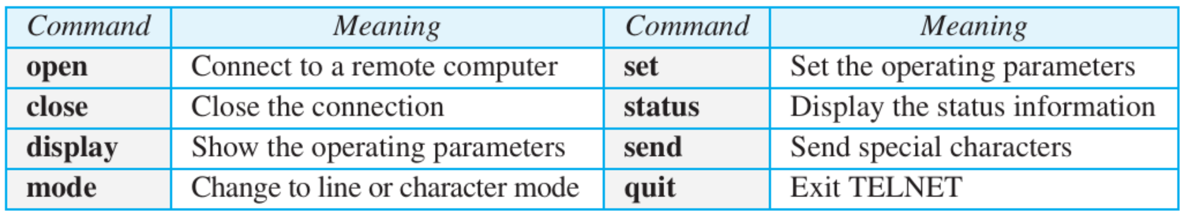
1. **Simplified Commands**:
  - Operating systems like UNIX provide user-friendly TELNET commands (e.g., as listed in Table 26.11).
  - Enables easy interaction with remote systems through a standardized interface.

Benefits of TELNET

- Provides access to remote systems without the need for specialized client/server pairs for each application.
- The **NVT character set** simplifies interaction with diverse operating systems and terminals.

User Interface ; The operating system (UNIX, for example) defines an interface with user-friendly commands. An example of such a set of commands can be found in Table 26.11.

Table 26.11 *Examples of interface commands*

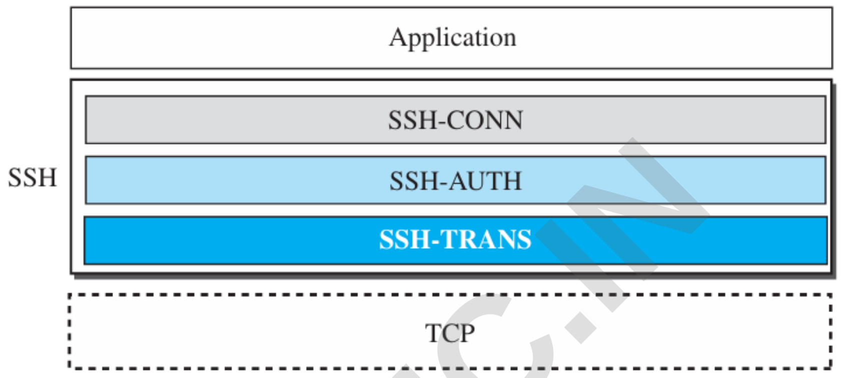


Command	*Meaning*	*Command*	*Meaning*
open	Connect to a remote computer	**set**	Set the operating parameters
close	Close the connection	**status**	Display the status information
display	Show the operating parameters	**send**	Send special characters
mode	Change to line or character mode	**quit**	Exit TELNET

SECURE SHELL (SSH)

Secure Shell (SSH) is a secure application program that can be used today for several purposes such as remote logging and file transfer, it was originally designed to replace TELNET. There are two versions of SSH: SSH-1 and SSH-2, which are totally incompatible. The first version, SSH-1, is now deprecated because of security flaws in it. In this section, we discuss only SSH-2. SSH is an application-layer protocol with three components, as shown in Figure 26.25.

Figure 26.25 *Components of SSH*



SSH Transport-Layer Protocol (SSH-TRANS) : SSH first uses a protocol that creates a secured channel on top of the TCP. This new layer is an independent protocol referred to as SSH-TRANS. When the procedure implementing this protocol is called, the client and server first use the TCP protocol to establish an insecure connection. Then they exchange several security parameters to establish a secure channel on top of the TCP.

1. Privacy or confidentiality of the message exchanged .
2. Data integrity, which means that it is guaranteed that the messages exchanged between the client and server are not changed by an intruder.
3. Server authentication, which means that the client is now sure that the server is the one that it claims to be.
4. Compression of the messages, which improves the efficiency of the system and makes attack more difficult.

SSH Authentication Protocol (SSH-AUTH) :

- Authenticates the **client** for the **server** after the secure channel is established.
- Similar to authentication in **Secure Socket Layer (SSL)**.
- The **client** sends a request message to the server containing:
    - **Username**.
    - **Server name**.
    - **Authentication method**.
    - Required authentication data.
- The **server** responds with either:
    - **Success message**: Confirms client authentication.
    - **Failed message**: Client must resend a new authentication request

SSH Connection Protocol (SSH-CONN):

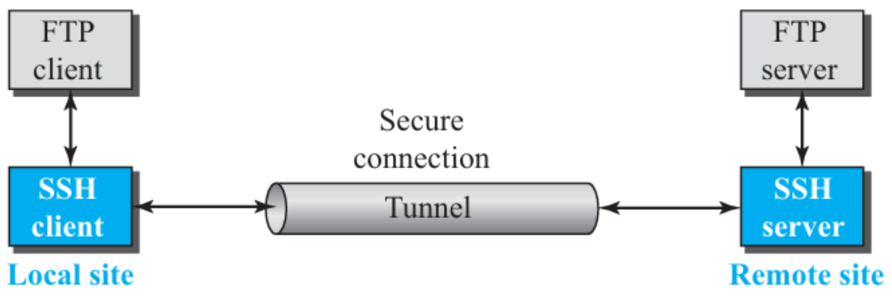
- Provides additional functionality after the secure channel is established and both client and server are authenticated.
- **Key Feature: Multiplexing**:
  - Allows multiple **logical channels** to be created over the single secure channel.
  - Each logical channel can serve different purposes, such as:
    - **Remote logging**.
    - **File transfer**.

Applications of SSH :

- **SSH for Remote Logging**
- **SSH for File Transfer** : One of the application programs that is built on top of SSH for file transfer is the Secure File Transfer Program (sftp). The sftp application program uses one of the channels pro vided by the SSH to transfer files. Another common application is called Secure Copy (scp). This application uses the same format as the UNIX copy command, cp, to copy files.
- **Port Forwarding** : One of the interesting services provided by the SSH protocol is port forwarding. We can use the secured channels available in SSH to access an application program that does not provide security services. Applications such as TELNET and Simple Mail Transfer Protocol (SMTP), can use the services of the SSH port forwarding mechanism.

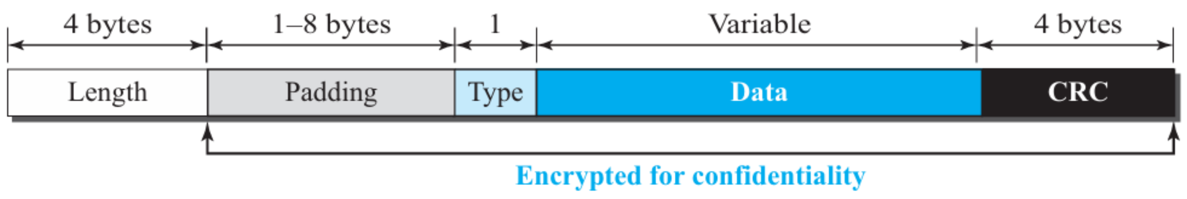
  The SSH port forwarding mechanism creates a tunnel through which the messages belonging to other protocols can travel. For this reason, this mechanism is sometimes referred to as SSH tunneling. Figure 26.26 shows the concept of port forwarding for securing the FTP application. The FTP client can use the SSH client on the local site to make a secure connection with the SSH server on the remote site. Any request from the FTP client to the FTP server is carried through the tunnel provided by the SSH client and server. Any response from the FTP server to the FTP client is also carried through the tunnel provided by the SSH client and server.

Figure 26.26 *Port forwarding*



Format of the SSH Packets :

Figure 26.27 *SSH packet format*



- The length field defines the length of the packet but does not include the padding.
- One to eight bytes of padding is added to the packet to make the attack on the security provision more difficult.
- The cyclic redundancy check (CRC) field is used for error detection.
- The type field designates the type of the packet used in different SSH protocols.
- The data field is the data transferred by the packet in different protocols.

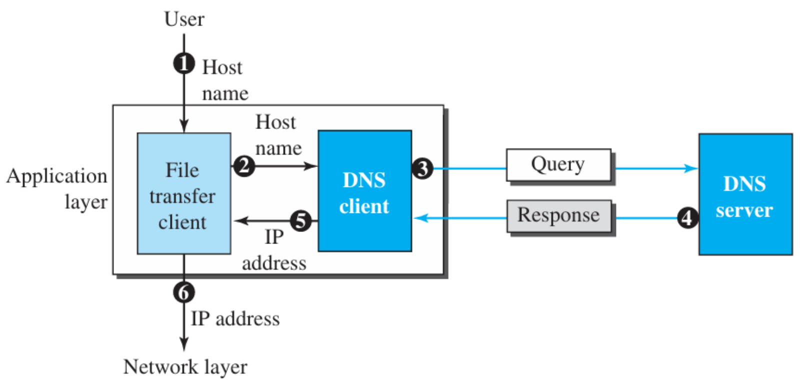
DOMAIN NAME SYSTEM (DNS)

Purpose of DNS

1. **Mapping Names to IP Addresses**:
   - Human-friendly names (e.g., *afilesource.com*) are mapped to machine-friendly IP addresses required for communication.
   - Similar to a phone directory mapping names to numbers.
2. **Why Not a Central Directory?**:
   - The Internet's scale makes a central directory impractical.
   - Central failure risks collapse of the entire system.
   - **Solution**: Distributed directory system, where multiple DNS servers share the information.

DNS Process

Figure 26.28 *Purpose of DNS*



Components:

- **DNS Client**: Resides on the user's computer and makes requests.
- **DNS Server**: Responds to mapping queries.

Steps to Map Host Name to IP Address:

1. User enters the hostname (e.g., afilesource.com) in the application (e.g., file transfer client).
2. File transfer client sends the hostname to the DNS client.
3. DNS client queries a known DNS server (its IP is pre-configured at boot).
4. DNS server responds with the corresponding IP address of the hostname.
5. DNS client passes the IP address to the file transfer client.
6. File transfer client uses the IP address to connect to the desired server.

Benefits of DNS

- **Scalability**: Distributed architecture avoids overloading a single server.
- **Resilience**: No single point of failure ensures continuity of service.
- **Convenience**: Users interact with familiar names instead of numeric addresses.

Name Space :

- Ensure unambiguous naming of machines by binding unique names to unique IP addresses.
- Names must be carefully managed to avoid duplication and ambiguity.

Types of Name Spaces

1. **Flat Name Space:**
   - Names are sequences of characters with no structure.
   - May have common sections, but they lack meaning.
   - Limitation: Unsuitable for large systems (e.g., the Internet) because it requires centralized control, which is impractical.
2. **Hierarchical Name Space:**
   - Names are composed of multiple parts, representing organizational structure.
     Example: Parts may define nature of the organization, name of the organization, and internal departments.
     - Central authority assigns the initial parts (e.g., organization type and name).
     - Organizations manage their internal parts, ensuring uniqueness within their namespace.
     - Scalable and avoids ambiguity, as names are distinct across organizations.

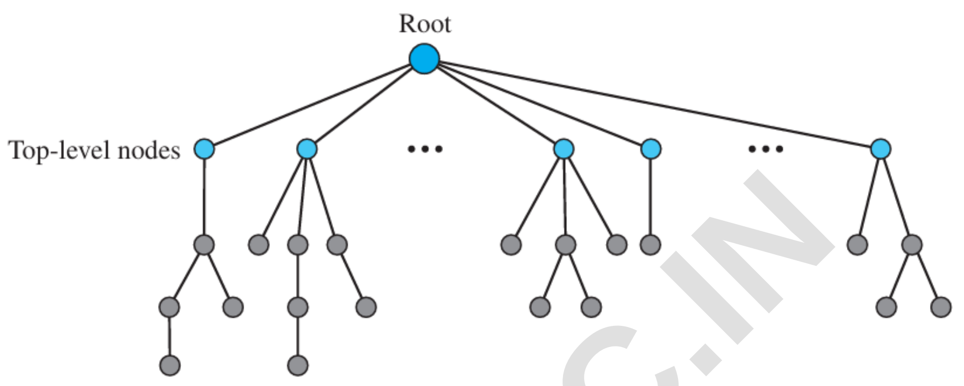
Example:

- Two organizations, first.com and second.com, both name a computer caesar.
- Final names are:
  - caesar.first.com
  - caesar.second.com

- Result: Unique and distinguishable names.

Domain Name Space : To have a hierarchical name space, a domain name space was designed. In this design the names are defined in an inverted-tree structure with the root at the top. The tree can have only 128 levels: level 0 (root) to level 127 (see Figure 26.29).

Figure 26.29 *Domain name space*



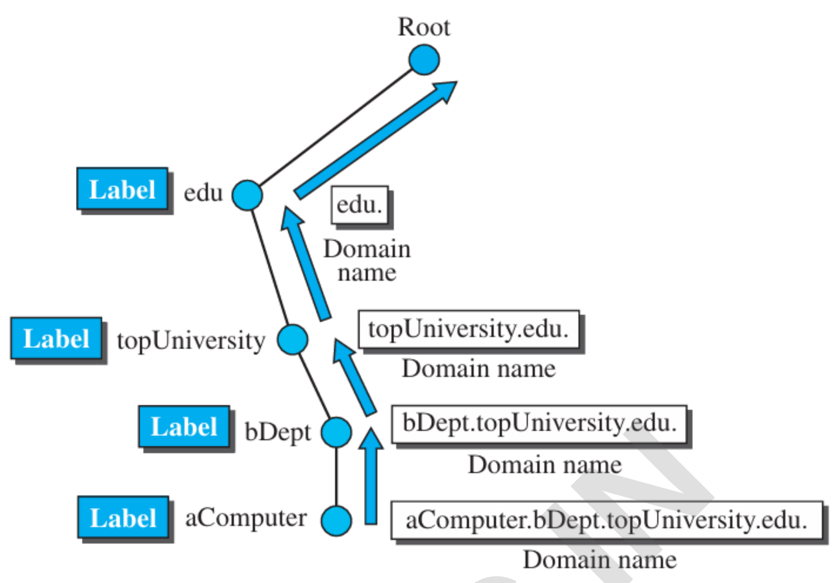
Label:

- Each node in the DNS tree has a label (string up to 63 characters).
- The **root label** is a null string (empty).
- **Uniqueness**: Children of the same node must have unique labels to ensure unique domain names.

Domain Name:

- Full domain names consist of labels separated by dots (.).
- Domain names are read from the node to the root, with the last label being the root (a null string, represented as a dot) as shown in figure 26.30.
- **Fully Qualified Domain Name (FQDN)**: Ends with a dot and represents the complete path to the root.
- **Partially Qualified Domain Name (PQDN)**: Does not reach the root and is resolved using a local suffix.

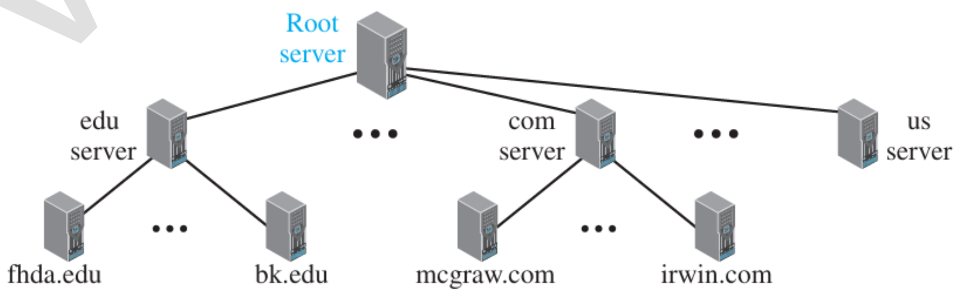
Figure 26.30 *Domain names and labels*



Hierarchy of Name Servers

- Centralized storage of the entire DNS hierarchy is inefficient and unreliable.
- Hierarchical Name Servers:
  - Divide the DNS hierarchy into domains and subdomains, with servers responsible for different zones.
  - Delegation ensures scalability and reliability**.**

Figure 26.32 *Hierarchy of name servers*



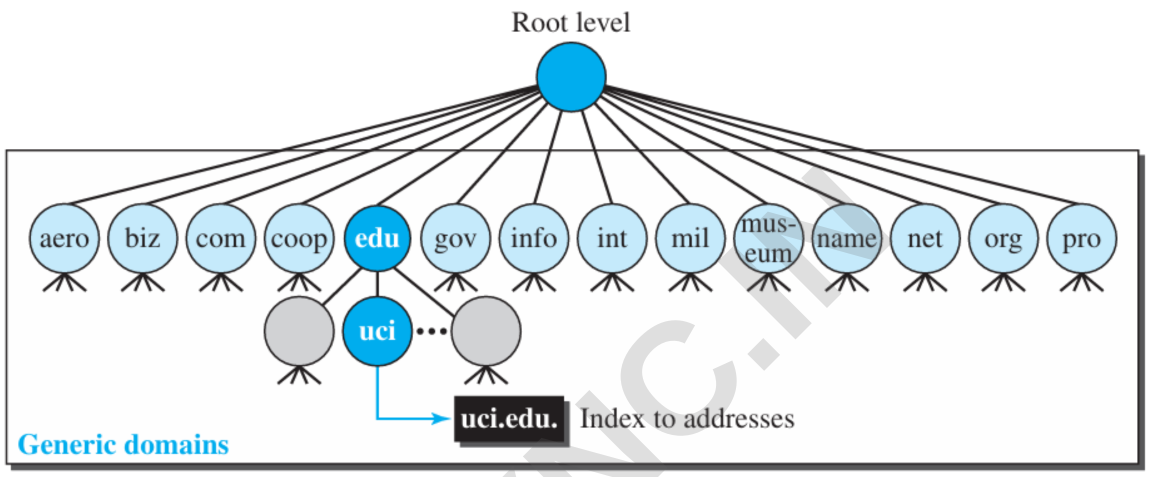
DNS Hierarchy in Practice

- A root server delegates to TLD servers (e.g., .com, .org), which delegate further to domain-level servers.
- Each level is responsible for its portion of the hierarchy, ensuring a scalable and fault-tolerant system.

DNS in the Internet: In the Internet, the domain name space (tree) was originally divided into three different sections: generic domains, country domains, and the inverse domains.

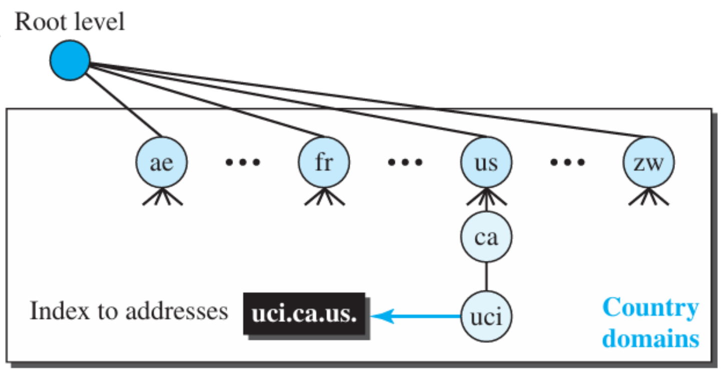
Generic Domains : The generic domains define registered hosts according to their generic behavior. Each node in the tree defines a domain, which is an index to the domain name space database (see Figure 26.34).

Figure 26.34 *Generic domains*



Country Domains : The country domains section uses two-character country abbreviations (e.g., us for United States). Second labels can be organizational, or they can be more specific national designations. The United States, for example, uses state abbreviations as a sub division of us (e.g., ca.us.). Figure 26.35 shows the country domains section. The address uci.ca.us. can be translated to University of California, Irvine, in the state of California in the United States.

Figure 26.35 *Country domains*



Resolution in DNS : Name-address resolution refers to the process of mapping a domain name to its corresponding IP address (or vice versa). This process is essential for network communication. DNS operates as a client-server application, where:

1. A DNS resolver (client) on the requesting host initiates the process.
2. The resolver interacts with a local or remote DNS server to obtain the mapping.

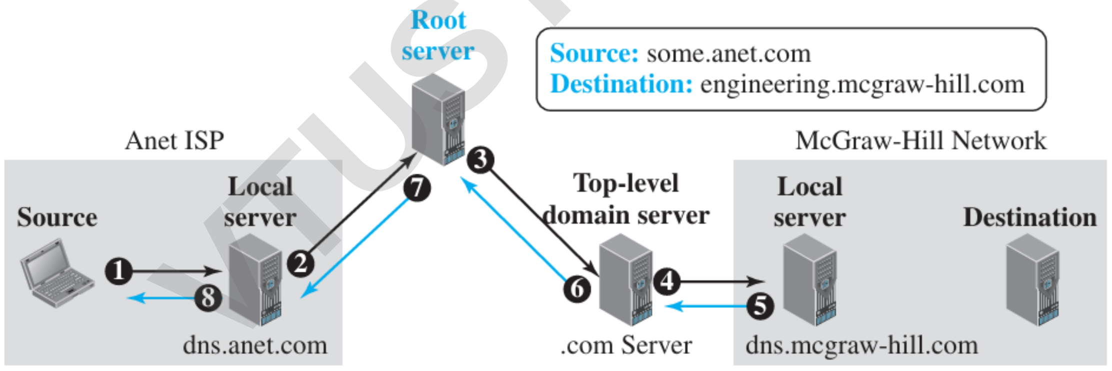
3. The process concludes either with a resolution (successful mapping) or an error message.

There are two types of resolution mechanisms: Recursive Resolution and Iterative Resolution.

Recursive Resolution

- **Process**:
    1. The resolver sends a query to the **local DNS server**.
    2. If the local server lacks the answer, it forwards the query to a **root DNS server**.
    3. The root server refers the query to a **top-level domain (TLD) server** (e.g., .com).
    4. The TLD server directs the query to the **authoritative DNS server** for the destination domain.
    5. The authoritative server resolves the query and sends the IP address back through the chain (TLD server → root server → local server → resolver).
    6. The local DNS server may cache the result for future requests.
- **Example**:
    - A host (some.anet.com) needs the IP of engineering.mcgraw-hill.com.
    - The recursive resolution flow:
        - Local DNS server → Root DNS server → TLD server (.com) → McGraw-Hill's DNS server → IP address returned via the same path.

Figure 26.36 *Recursive resolution*

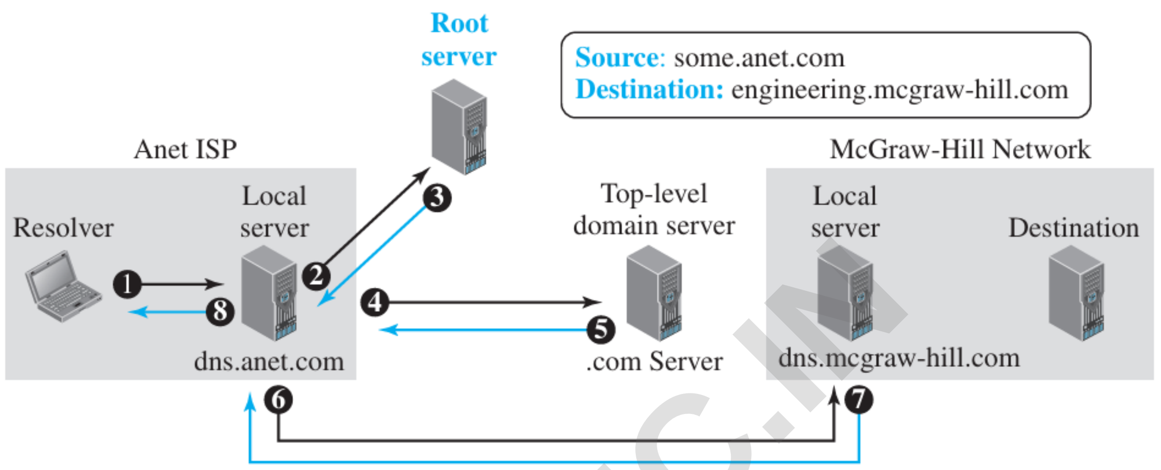


Iterative Resolution

- **Process**:
    1. The resolver queries the **local DNS server**.
    2. If the local server doesn't have the answer, it replies with the IP address of the next server (e.g., a root server).
    3. The resolver then queries the next server directly.
    4. This process repeats with referrals to subsequent servers (e.g., TLD server, authoritative server) until the resolver receives the final mapping.
- **Example**:
    - Using the same scenario:
        - The resolver first queries the local DNS server.
        - The local server provides the IP address of the root server.

- The resolver queries the root server, which provides the TLD server's IP.
- The resolver queries the TLD server, which refers to McGraw-Hill's DNS server.
- The resolver queries the authoritative server for the final IP address.

Figure 26.37 *Iterative resolution*



DNS Caching:

- DNS servers use caching to store mappings temporarily after resolving queries to reduce search time and improve efficiency.
- If the same or another client requests the same mapping, the server retrieves it from cache memory, marking the response as "unauthoritative" to indicate it's not from an authoritative source.
- Cached mappings may become outdated, leading to incorrect responses.
    - **Solution - Time to Live (TTL):**
        - Authoritative servers include a TTL value with each mapping, specifying how long it can be cached.
        - After the TTL expires, the mapping is invalidated, requiring a fresh query to the authoritative server.
- DNS servers maintain TTL counters for each cached mapping and periodically purge expired entries to prevent outdated information.

Resource Records : The zone information associated with a server is implemented as a set of resource records. In other words, a name server stores a database of resource records. A resource record is a 5-tuple structure, as shown below:

(Domain Name, Type, Class, TTL, Value)

The domain name field is what identifies the resource record. The value defines the information kept about the domain name. The TTL defines the number of seconds for which the information is valid. The class defines the type of network; we are only interested in the class IN (Internet). The type defines how the value should be interpreted.

DNS: distributed db storing resource records (RR)

RR format: `(name, value, type, ttl)`

type=A (address)

- `name` is hostname
- `value` is IP address

type=NS (Name server)

- `name` is domain (e.g., foo.com)
- `value` is hostname of authoritative name server for this domain

type=CNAME (canonical Name)

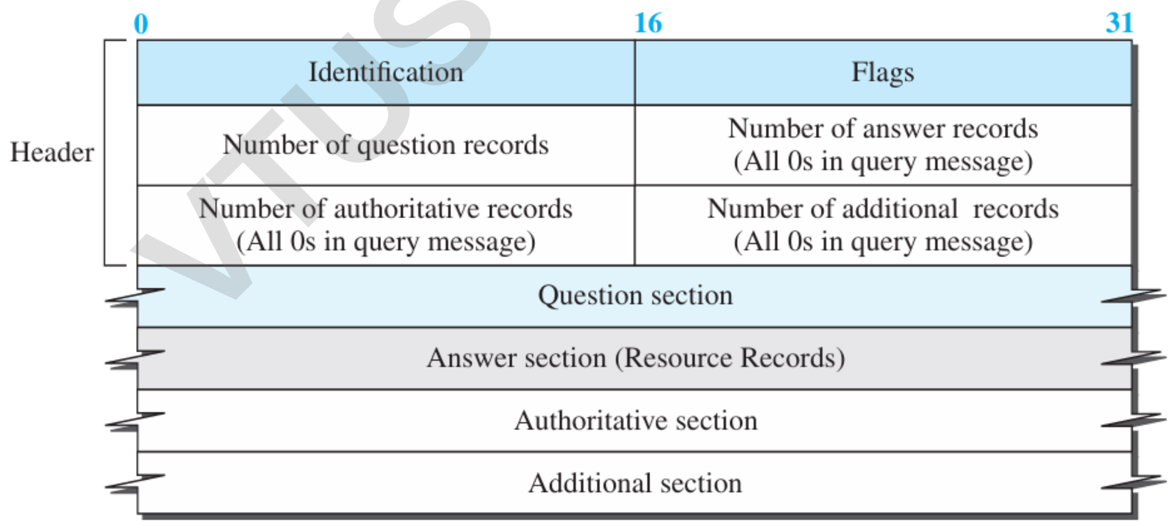
- `name` is alias name for some "canonical" (the real) name
- `www.ibm.com` is really `servereast.backup2.ibm.com`
- `value` is canonical name

type=MX (Mail Exchanger)

- `value` is name of mailserver associated with `name`

DNS Messages : To retrieve information about hosts, DNS uses two types of messages: query and response. Both types have the same format as shown in Figure 26.38.

Figure 26.38 *DNS message*


0
16
31
Header
Identification
Flags
Number of question records
Number of answer records (All 0s in query message)
Number of authoritative records (All 0s in query message)
Number of additional records (All 0s in query message)
Question section
Answer section (Resource Records)
Authoritative section
Additional section

Note:
The query message contains only the question section. The response message includes the question section, the answer section, and possibly two other sections.

- **Identification Field:** Used by the client to match responses with their corresponding queries.
- **Flag Field:** Indicates whether the message is a query or response and includes error status.
- **Header Fields:** Define the number of records of each type in the message.
- **Question Section:** Contains one or more question records; present in both query and response messages.

- **Answer Section:** Includes one or more resource records; present only in response messages.
- **Authoritative Section:** Provides information about authoritative servers for the query.
- **Additional Information Section:** Offers extra details to assist the resolver in processing the query.

Registrars :

- New domains are added to DNS through registrars, which are commercial entities accredited by ICANN.
- Registrars ensure the requested domain name is unique before adding it to the DNS database.
    - Required Information for Registration:
    - Name of the server.
    - IP address of the server.
    - For example, an organization "wonderful" with a server "ws" and IP "200.200.200.5" would provide this information to the registrar.

Domain name: ws.wonderful.com **IP address: 200.200.200.5**

Dynamic DNS (DDNS):

Dynamic DNS (DDNS) was developed to handle the increasing frequency of address changes on the Internet. Unlike traditional DNS, which requires manual updates to the master file for changes like adding or removing hosts or updating IP addresses, DDNS updates the DNS master file dynamically. When a binding between a name and an address is established, the information is sent (usually via DHCP) to a primary DNS server, which updates the zone. Secondary servers are notified of changes either actively (via messages) or passively (via periodic checks) and then request zone transfers.
To ensure security and prevent unauthorized changes, DDNS can incorporate authentication mechanisms.
DNS is critical to Internet infrastructure, supporting essential applications like web access and email. It is vulnerable to several attacks:

1. **Data Profiling:** Attackers can read DNS responses to analyze user behavior. Confidentiality can prevent this.
2. **Spoofing:** Attackers can intercept and modify DNS responses or create bogus ones, redirecting users. Message origin authentication and integrity mechanisms address this.
3. **Denial-of-Service (DoS):** Attackers can flood DNS servers, causing crashes. DNS caching helps mitigate this, though DNSSEC lacks specific DoS protections.

DNSSEC enhances security with digital signatures for authentication and integrity but does not provide confidentiality for messages.