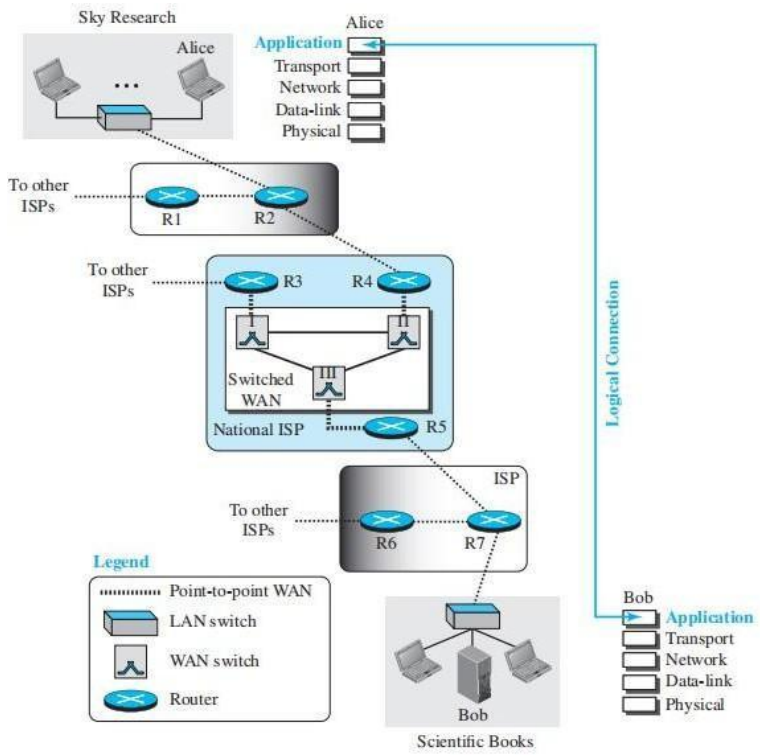
Module 5

Introduction to Application layer

Introduction

The application layer provides services to the user. Communication is provided using a logical connection, which means that the two application layers assume that there is an imaginary direct connection through which they can send and receive messages. Figure shows the idea behind this logical connection.



Providing Services

Before the Internet, communication networks like the telephone network were designed for specific services, such as voice communication. Over time, users adapted these networks to provide additional services, like fax, by adding extra hardware.

The Internet, similarly designed to serve global users, is more adaptable due to the layered structure of the TCP/IP protocol suite. Each layer in this suite can add or replace protocols, as

long as they work with the protocols in the layers below. This flexibility supports the evolving nature of the Internet.

The application layer, at the top, differs because it only receives services from the transport layer without needing to support other layers. This makes it easy to add or remove application protocols as long as they can work with the transport layer. This flexibility allows the Internet to continually expand with new application protocols, which has been a constant feature of its growth since its inception.

Standard and Nonstandard Protocols

For smooth Internet operation, the protocols in the first four layers of the TCP/IP suite need to be standardized and documented. These standard protocols are typically part of operating systems like Windows or UNIX. However, the application-layer protocols can be both standard and nonstandard for added flexibility.

Standard Application-Layer Protocols-Standard application-layer protocols are well-documented and standardized by Internet authorities. They are widely used in daily Internet interactions.

Nonstandard Application-Layer Protocols-Programmers can also create nonstandard (or proprietary) application-layer programs by developing two programs that provide services through the transport layer.

Application-Layer Paradigms

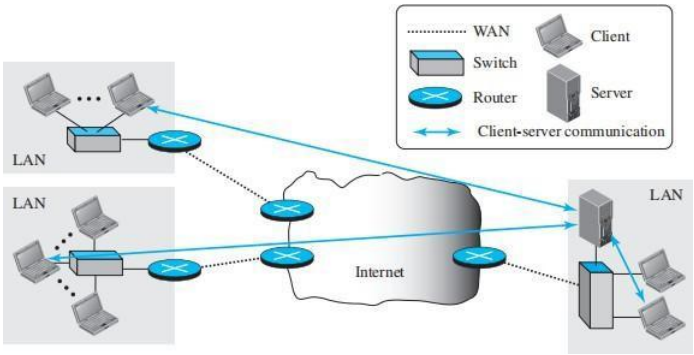
To use the Internet, two application programs must communicate: one on a computer somewhere in the world and the other on a different computer. These programs exchange messages over the Internet, but their relationship can vary. There are two main paradigms:

1. **Client-Server Paradigm**: One program (client) requests services, while the other (server) provides services.
2. **Peer-to-Peer (P2P) Paradigm**: Both programs can request and provide services.

Traditional Paradigm: Client-Server

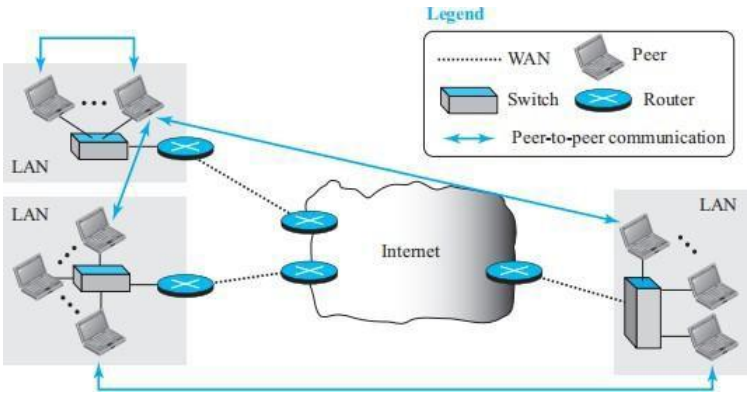
- ➔ In client-server architecture, there is an **always-on** host, called the **server**, which provides services when it receives requests from many other hosts, called **clients**.
- ➔ **Example:** In Web application Web server services requests from browsers running on client hosts. When a Web server receives a request for an object from a client host, it responds by sending the requested object to the client host.

- ➔ In client-server architecture, clients do not directly communicate with each other.
- ➔ The server has a fixed, well-known address, called an **IP address**. Because the server has a fixed, well-known address, and is always on, a client can always contact the server by sending a packet to the server's IP address.
- ➔ Some of the better-known applications with client-server architecture include the Web, FTP, Telnet, and e-mail.
- ➔ The most popular Internet services—such as search engines (e.g., Google and Bing), Internet commerce (e.g., Amazon and e-Bay), Web-based email (e.g., Gmail and Yahoo Mail), social networking (e.g., Facebook and Twitter)— employ one or more data centers.



New Paradigm: Peer-to-Peer

- ➔ In P2P architecture, there is minimal dependence on dedicated servers in data centers.
- ➔ The application employs direct communication between pairs of intermittently connected hosts, called peers.
- ➔ The peers are not owned by the service provider, but are instead desktops and laptops controlled by users, with most of the peers residing in homes, universities, and offices.
- ➔ Many of today's most popular and traffic-intensive applications are based on P2P architectures. These applications include file sharing (e.g., BitTorrent), Internet Telephony (e.g., Skype), and IPTV (e.g., Kankan and PPstream).



Mixed Paradigm

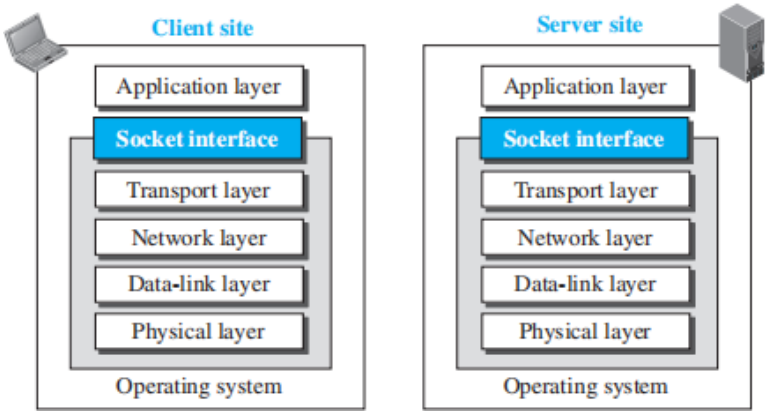
An application may choose to use a mixture of the two paradigms by combining the advantages of both. For example, a light-load client-server communication can be used to find the address of the peer that can offer a service. When the address of the peer is found, the actual service can be received from the peer by using the peer-to -peer paradigm.

CLIENT-SERVER PROGRAMMING

In a client-server paradigm, communication at the application layer is between two running application programs called ***processes***: a *client* and a *server*. The client is the program that initiates communication by sending a request, while the server is a program that waits for incoming requests, processes them, and returns a response. This setup requires that the server be running when a client request arrives, whereas the client only runs as needed. In practice, this means a client process can run on one computer and the server on another, but the server must be started first to be available for client requests.

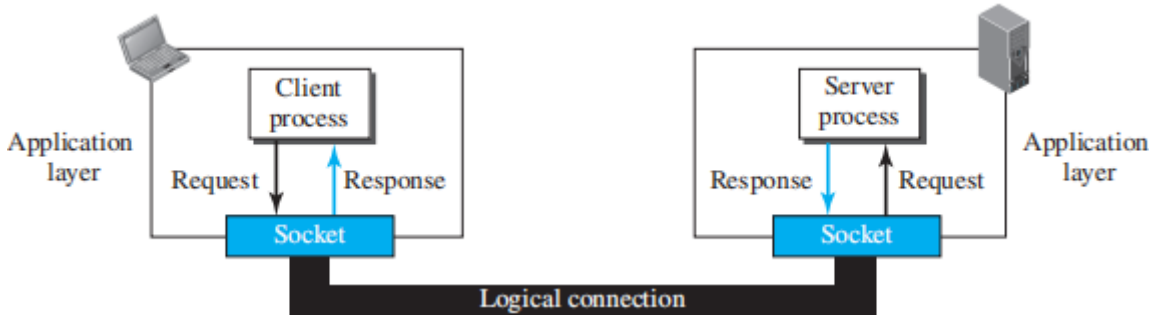
Application Programming Interface

The interface between the application layer and the transport layer within a host which is also referred to as the **Application Programming Interface (API).**
An API acts as a bridge between two parts: the application layer process (the program) and the operating system, which manages the first four layers of the TCP/IP protocol suite. Essentially, the computer's operating system includes these first four layers and an API so that applications can communicate through the Internet. This way, applications can easily send and receive messages by using the services of the operating system. te with the operating system when sending and receiving messages through the Internet. Several APIs have been designed for communication. Three among them are common: **socket interface, Transport Layer Interface (TLI),** and **STREAM**.



Sockets

A process sends messages into, and receives messages from, the network through a software interface called a **socket.** The application at the sending side pushes messages through the socket. At the other side of the socket, the transport-layer protocol has the responsibility of getting the messages to the socket of the receiving process



Socket Addresses

The interaction between a client and a server is two-way communication. In a two-way communication, we need a pair of addresses: local (sender) and remote (receiver). The local address in one direction is the remote address in the other direction and vice versa. Since communication in the client-server paradigm is between two sockets, we need a pair of **socket addresses** for communication: a local socket address and a remote socket address.

To identify the receiving process, two pieces of information need to be specified:
The address of the host
An identifier that specifies the receiving process in the destination host.
In the Internet, the host is identified by its **IP address**.
In addition to knowing the address of the host to which a message is destined, the sending process must also identify the receiving process running in the host.
A **destination port number** serves this purpose. Popular applications have been assigned specific port numbers.
For example, a Web server is identified by port number 80. A mail server process (using the SMTP protocol) is identified by port number 25.

Using Services of the Transport Layer

A pair of processes provide services to the users of the Internet, human or programs. A pair of processes, however, need to use the services provided by the transport layer for communication because there is no physical communication at the application layer. There are three common transport-layer protocols in the TCP/IP suite: UDP, TCP, and SCTP.

UDP Protocol

UDP is a connectionless, unreliable protocol that treats each message as an independent datagram with no connection between them. It doesn't guarantee delivery or retransmission of lost data but is fast and simple. UDP is ideal for applications that prioritize speed over reliability, like some management and multimedia apps.

TCP Protocol

TCP (Transmission Control Protocol) provides a connection-oriented, reliable, byte-stream service. It first establishes a logical connection between two ends through a handshaking process, setting parameters like packet size and buffer size. Data is then exchanged in segments, with byte numbering to ensure continuity. If bytes are lost or corrupted, TCP allows the receiver to request retransmission, making it reliable. TCP also supports flow and congestion control. However, TCP is not message-oriented and does not define boundaries in messages. Applications needing reliability and capable of handling long messages benefit from TCP.

SCTP Protocol

SCTP provides a service which is a combination of the two other protocols. Like TCP, SCTP provides a connection-oriented, reliable service, but it is not byte stream oriented. It is a message-oriented protocol like UDP. In addition, SCTP can provide multi-stream service by providing multiple network-layer connections. SCTP is normally suitable for any application that needs reliability and at the same time needs to remain connected, even if a failure occurs in one network-layer connection.

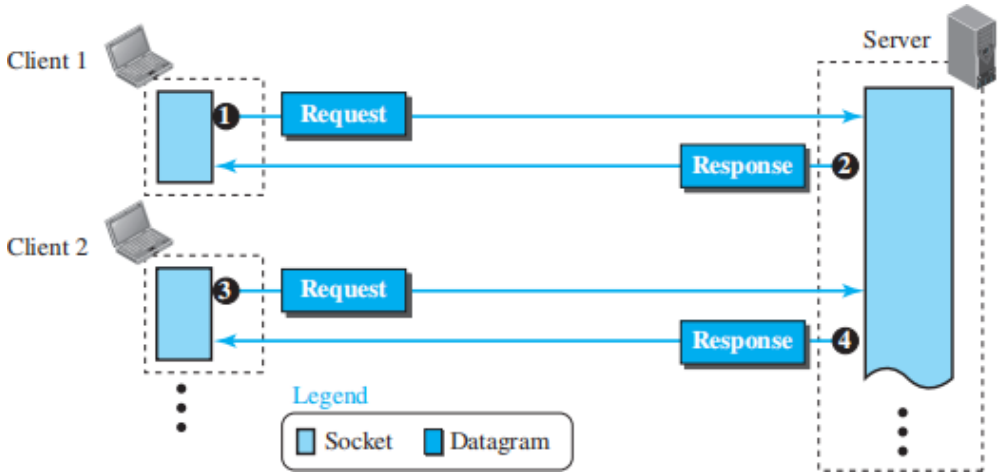
Iterative Communication Using UDP

Communication between a client program and a server program can occur iteratively or concurrently. Although several client programs can access the same server program at the same time, the server program can be designed to respond iteratively or concurrently. An iterative server can process one client request at a time; it receives a request, processes it, and sends the response to the requestor before handling another request. When the server is handling the request from a client, the requests from other clients, and even other requests from the same

client, need to be queued at the server site and wait for the server to be freed. The received and queued requests are handled in the first-in, first-out fashion.

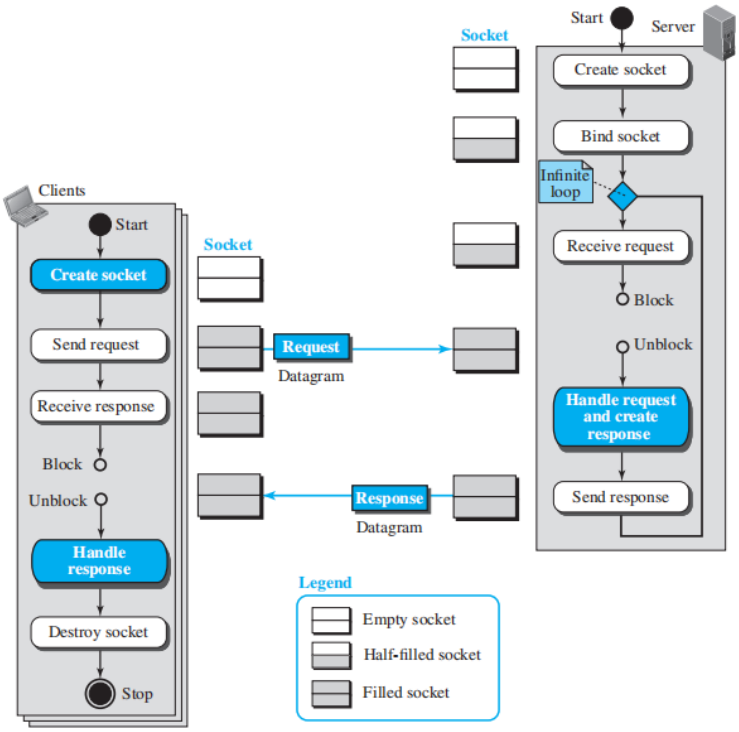
Sockets Used for UDP

In UDP communication, the client and server use only one socket each. The socket created at the server site lasts forever; the socket created at the client site is closed (destroyed) when the client process terminates. Figure shows the lifetime of the sockets in the server and client processes. 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.



Flow Diagram

UDP provides a connectionless service, in which a client sends a request and the server sends back a response. Figure shows a simplified flow diagram for iterative communication. There are multiple clients, but only one server. Each client is served in each iteration of the loop in the server. 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

The server makes a *passive open*, in which it becomes ready for the communication, but it waits until a client process makes the connection. It creates an empty socket. It then binds the socket to the server and the well-know port, in which only part of the socket (the server socket address) is filled (binding can happen at the time of creation depending on the underlying language). The server then issues a receive request command, blocking it from doing anything else until a client request arrives.

When a client request is received, the server fills in the remaining socket information (the client's address), processes the request, and sends the response back to the client. After this, the server clears the client's address in the socket and waits for the next request in a continuous loop. Each time, the socket is only fully filled when a new request comes in.

Client Process

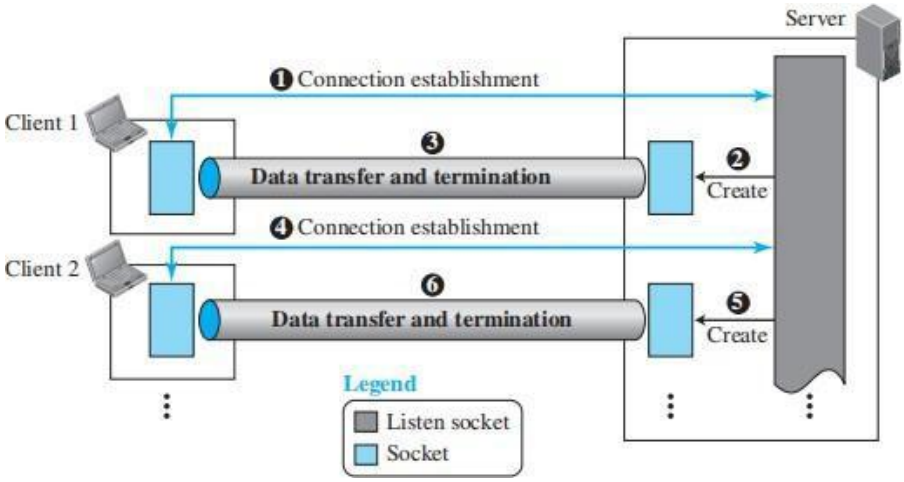
The client process makes an *active open*. In other words, it starts a connection. It creates an empty socket and then issues the send command, which fully fills the socket, and sends the request. The client then issues a receive command, which is blocked until a response arrives from the server. The response is then handled and the socket is destroyed.

Iterative Communication Using TCP

TCP is a connection-oriented protocol. 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.

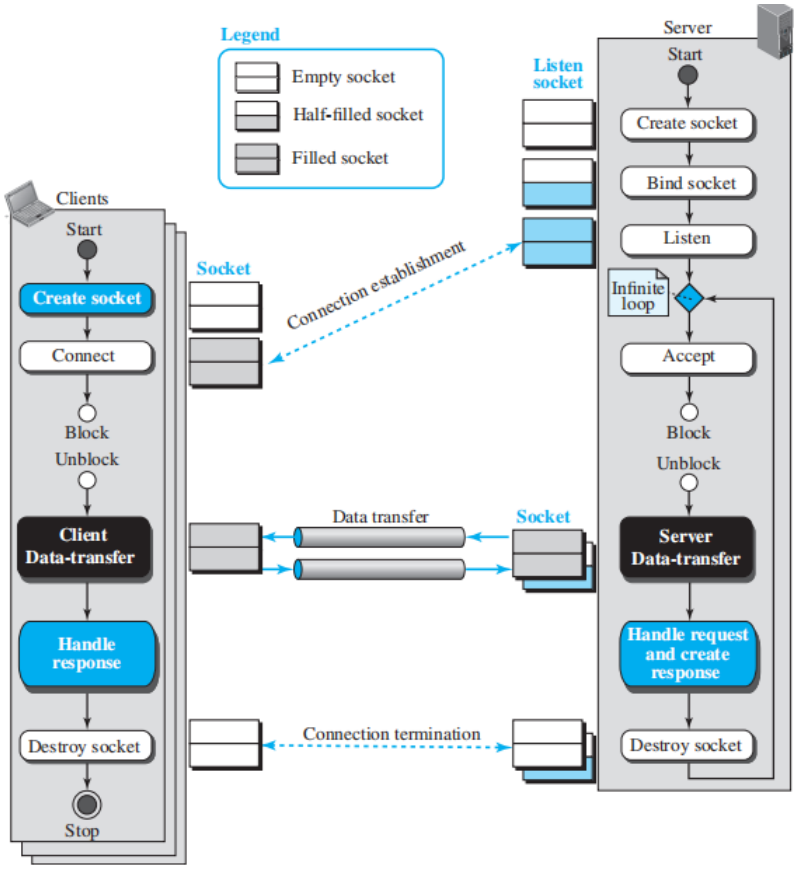
Sockets Used in TCP

The TCP server uses two different sockets, one for connection establishment and the other for data transfer. We call the first one the *listen socket* and the second the *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.



Flow Diagram

Figure below 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

- The TCP server creates a listening socket and binds it to a network address and port, used only for listening to incoming client connections.
- The server calls a "listen" function, allowing the operating system to accept connections and add them to a waiting list.
- The server enters a loop, handling each client one at a time.
- In each loop iteration, the server uses the "accept" function to pick a client from the waiting list; if no clients are waiting, the server blocks until a client arrives.
- Upon accepting a client, a new socket is created specifically for data exchange with that client.
- The server sets this new socket with the client's address for direct communication.
- The server and client can then exchange data using this client-specific socket.

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.

Standard Client-Server Protocols

WORLD WIDE WEB AND HTTP

World Wide Web

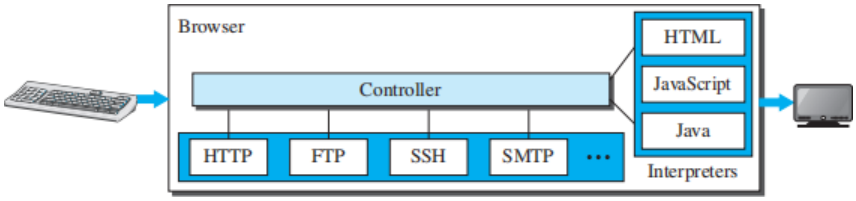
- The idea of the Web was first proposed by Tim Berners-Lee in 1989 at *CERN*†, the European Organization for Nuclear Research, to allow several researchers at different locations throughout Europe to access each other's' researches.
- The commercial Web started in the early 1990s.
- Today, the Web is a global information space, where documents (called web pages) are spread across the world and connected through links.
- The Web's growth and popularity are tied to two key ideas: "distributed" and "linked."
    - **Distributed**: Any web server worldwide can add new pages, which helps expand the Web without overwhelming a few servers.
    - **Linked**: Web pages can reference other pages on different servers, creating a network of interconnected information.
- This linking is done through **hypertext**, a concept from before the Internet. It allows a document to automatically access another linked document. The Web made this possible electronically by letting users click a link to retrieve a connected document.
- The term **hypertext** evolved to **hypermedia**, meaning web pages can include not only text but also images, audio, and video.
- The Web's purpose has grown beyond sharing documents; it now supports online shopping, gaming, and on-demand access to radio and TV programs.

Architecture

The WWW today is a distributed client-server service, in which a client using a browser can access a service using a server. However, the service provided is distributed over many locations called **sites**. Each site holds one or more **web pages**. A web page can be **simple or composite**. A simple web page has no links to other web pages; a composite web page has one or more links to other web pages. Each web page is a file with a name and address.

Web Client (Browser)

A variety of vendors offer commercial **browsers** that interpret and display a web page, and all of them use nearly the same architecture. Each browser usually consists of three parts: a controller, client protocols, and interpreters.



- 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, four main parts are needed:

1. **Protocol:** This is like choosing the "vehicle" or method to reach the web page. Commonly, this is HTTP (HyperText Transfer Protocol), which lets the browser access and display web pages. However, other methods, like FTP (File Transfer Protocol), can also be used for different types of access.
2. **Host:** The host tells us where the web page is stored. It can be written as an IP address (e.g., 64.23.56.17) or as a domain name, like "example.com," which makes it easier to identify servers.
3. **Port:** The port number is a specific “entry point” on the server. For example, port 80 is usually used for HTTP, while HTTPS uses port 443. Most of the time, the port is

standard and doesn't need to be written, but if a different port is used, it can be specified in the address.

4. **Path:** The path gives the exact location of the web page file within the server. It shows the series of folders and the file name, like "/folder1/folder2/filename." This tells the server exactly where to look to find and deliver the web page.

When we put these four elements together, we get a **URL (Uniform Resource Locator),** which is the complete address that uniquely identifies and fetches a specific web page. Each part is separated to create a clear structure for finding the page.

protocol://host/path	**Used most of the time**
protocol://host:port/path	**Used when port number is needed**

Example

The URL ***http://www.mhhe.com/compsci/forouzan/*** defines the web page related to one of the authors of this book. The string ***www.mhhe.com*** is the name of the computer in the McGraw-Hill company (the three letters ***www*** are part of the host name and are added to the commercial host). The path is ***compsci/forouzan/,*** which defines Forouzan's web page under the directory ***compsci*** (computer science).

Web Documents

The documents in the WWW can be grouped into three broad categories: static, dynamic, and active

Static Documents

Static documents are fixed-content documents that are created and stored in a server. When a client accesses the document, a copy of the document is sent. The user can then use a browser to see the document. Static documents are prepared using one of several languages: *HyperText Markup Language* (HTML), *Extensible Markup Language* (XML), *Extensible Style Language* (XSL), and *Extensible Hypertext Markup Language* (XHTML).

Dynamic Documents

A **dynamic document** is created by a web server whenever a browser requests the document. When a request arrives, the web server runs an application program or a script that creates the dynamic document. The server returns the result of the program or script as a response to the browser that requested the document. Because a fresh document is created for each request, the contents of a dynamic document may vary from one request to another. A very simple example of a dynamic document is the retrieval of the time and date from a server.

Active Documents

Active documents are files that require a program or script to run on the client's device to enable interactive or animated content on the user's screen. When a browser requests an active

document, the server sends a copy of the file to the client. The document is then run by the client's browser, allowing animations and interactions to occur locally on the user's device.
Active Documents are created using two methods

1. **Java Applets**: Written in Java and compiled into bytecode (binary format). Ready to run on the client's device.
2. **JavaScript**: Scripts that are downloaded from the server and executed directly by the client's browser.

These methods enable dynamic, interactive content directly on the client's device, making it possible for applications like animations and user interactions to work smoothly at the client side.

HyperText Transfer Protocol (HTTP)

- The Hyper Text Transfer Protocol (HTTP), the Web's application-layer protocol, is at the heart of the Web.
- HTTP is implemented in two programs: a **client** program and a **server** program. The client program and server program, executing on different end systems, talk to each other by exchanging HTTP messages. HTTP defines the structure of these messages and how the client and server exchange the messages.
- A Web page consists of **objects**. An object is simply a file like HTML file, a JPEG image, a Java applet, or a video clip—that is addressable by a single URL.
- Most Web pages consist of a base HTML file and several referenced objects.
- The base HTML file references the other objects in the page with the objects' URLs.
- HTTP defines how Web clients request Web pages from Web servers and how servers transfer Web pages to clients.
- When a user requests a Web page (for example, clicks on a hyperlink), the browser sends HTTP request messages for the objects in the page to the server. The server receives the requests and responds with HTTP response messages that contain the objects.
- HTTP uses TCP as its underlying transport protocol. The HTTP client first initiates a TCP connection with the server. Once the connection is established, the browser and the server processes access TCP through their socket interfaces

Non-Persistent and Persistent Connections

If Separate TCP connection is used for each request and response, then the connection is said to be **non-persistent**.

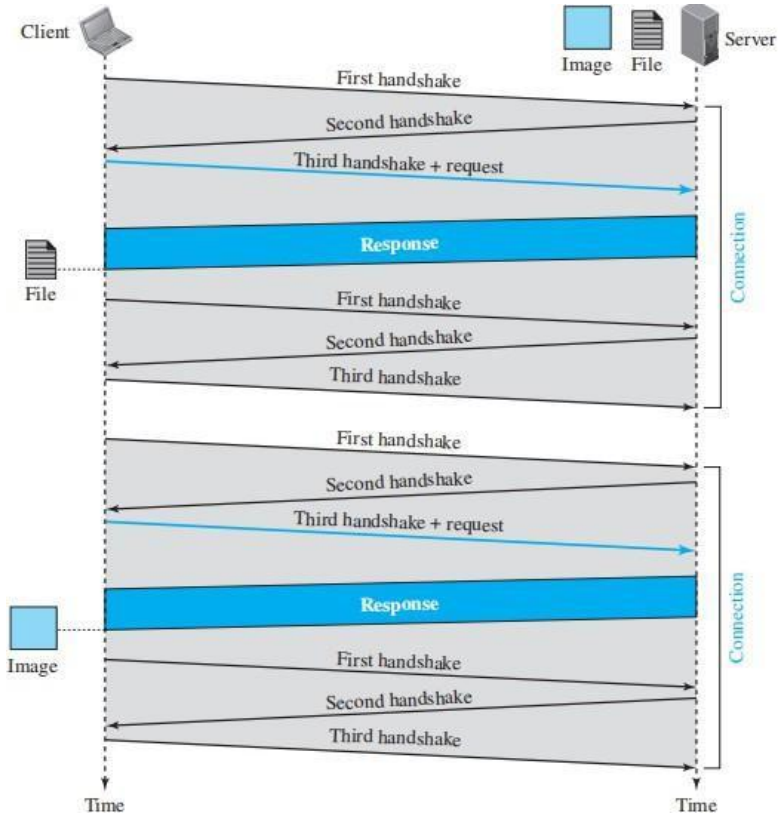
If same TCP connection is used for series of related request and response, then the connection is said to be **persistent.**

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.



- Round-trip time (RTT) is the time it takes for a small packet to travel from client to server and then back to the client.
- The RTT includes packet-propagation delays, packet queuing delays in intermediate routers and switches, and packet-processing delays.
- When a user clicks on a hyperlink, the browser initiates a TCP connection between the browser and the Web server; this involves a "three-way handshake"—the client sends a small TCP segment to the server, the server acknowledges and responds with a small TCP segment, and, finally, the client acknowledges back to the server.

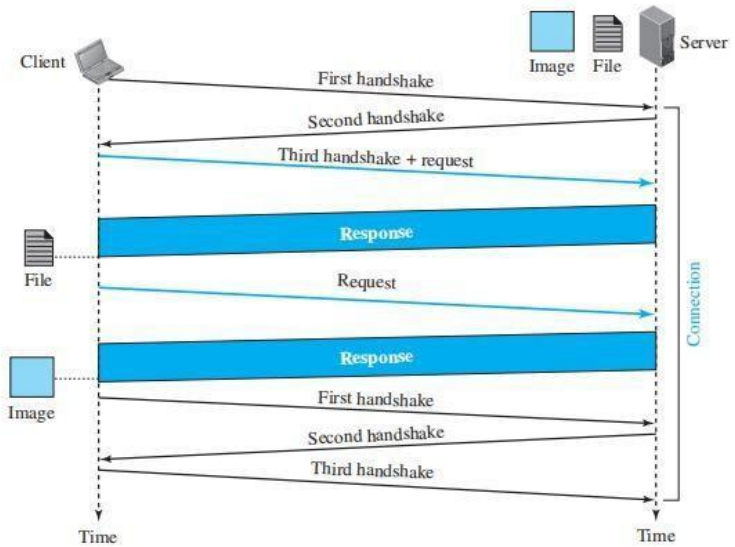
- The first two parts of the three-way handshake take one RTT.
- After completing the first two parts of the handshake, the client sends the HTTP request message combined with the third part of the three-way handshake (the acknowledgment) into the TCP connection.
- Once the request message arrives at the server, the server sends the HTML file into the TCP connection. This HTTP request/response eats up another RTT. Thus, roughly, the total response time is two RTTs plus the transmission time at the server of the HTML file.

Persistent Connections

HTTP version 1.1 specifies a persistent connection by default. Non-persistent connections have some shortcomings.

1. A brand-new connection must be established and maintained for each requested object. This can place a significant burden on the Web server, which may be serving requests from hundreds of different clients simultaneously.
2. Each object suffers a delivery delay of two RTTs— one RTT to establish the TCP connection and one RTT to request and receive an object.

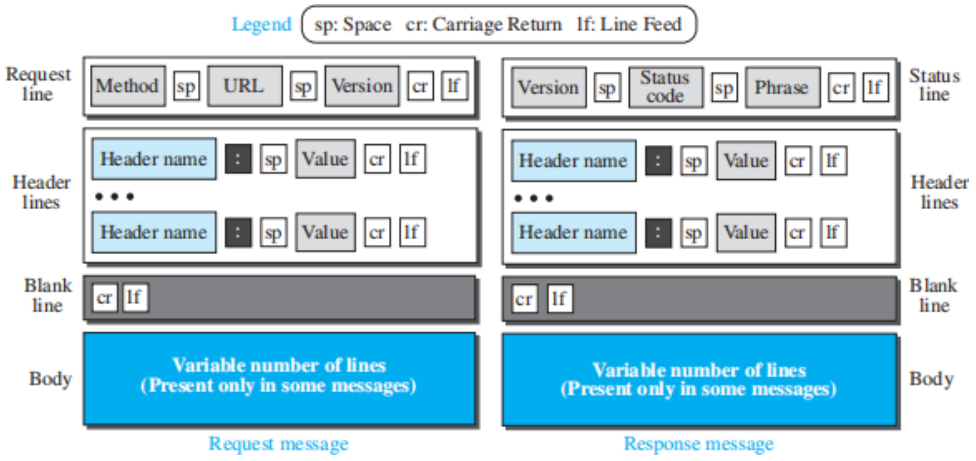
With persistent connections, the server leaves the TCP connection open after sending a response. Subsequent requests and responses between the same client and server can be sent over the same connection. In particular, an entire Web page can be sent over a single persistent TCP connection. Moreover, multiple Web pages residing on the same server can be sent from the server to the same client over a single persistent TCP connection.



Only one connection establishment and connection termination is used, but the request for the image is sent separately

Message Formats

The HTTP protocol defines the format of the request and response messages.



Request Message

Method: There are five HTTP methods:

- **GET**: The GET method is used when the browser requests an object, with the requested object identified in the URL field.
- **POST:** With a POST message, the user is still requesting a Web page from the server, but the specific contents of the Web page depend on what the user entered into the form fields. If the value of the method field is POST, then the entity body contains what the user entered into the form fields.
- **PUT:** The PUT method is also used by applications that need to upload objects to Web servers.
- **HEAD:** Used to retrieve header information. It is used for debugging purpose.
- **DELETE:** The DELETE method allows a user, or an application, to delete an object on a Web server.

URL: Specifies URL of the requested object

Version: This field represents HTTP version, usually HTTP/1.1

Header line: Ex:

Host: www.someschool.edu

Connection: close

User-agent: Mozilla/5.0

Accept-language: fr

The header line **Host:www.someschool.edu** specifies the host on which the object resides.

By including the **Connection:close** header line, the browser is telling the server that it doesn't want to bother with persistent connections; it wants the server to close the connection after sending the requested object.

The **User-agent:** header line specifies the user agent, that is, the browser type that is making the request to the server. Here the user agent is Mozilla/5.0, a Firefox browser.

The **Accept-language:** header indicates that the user prefers to receive a French version of the object, if such an object exists on the server; otherwise, the server should send its default version.

Response Message

EX

HTTP/1.1 200 OK

Connection: close

Date: Tue, 09 Aug 2011 15:44:04 GMT

Server: Apache/2.2.3 (CentOS)

Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT

Content-Length: 6821

Content-Type: text/html

(data data data data data ...)

The **status line** has three fields: the protocol version field, a status code, and a corresponding status message.

Version is HTTP/1.1

The status code and associated phrase indicate the result of the request. Some common status codes and associated phrases include:

1. 200 OK: Request succeeded and the information is returned in the response.
2. 301 Moved Permanently: Requested object has been permanently moved; the new URL is specified in Location: header of the response message. The client software will automatically retrieve the new URL.
3. 400 Bad Request: This is a generic error code indicating that the request could not be understood by the server.
4. 404 Not Found: The requested document does not exist on this server.
5. 505 HTTP Version Not Supported: The requested HTTP protocol version is not supported by the server.

Header fields:

- The server uses the **Connection: close** header line to tell the client that it is going to close the TCP connection after sending the message.
- The **Date:** header line indicates the time and date when the HTTP response was created and sent by the server.
- The **Server:** header line indicates that the message was generated by an Apache Web server; it is analogous to the User-agent: header line in the HTTP request message.

- The **Last-Modified:** header line indicates the time and date when the object was created or last modified.
- The **Content-Length:** header line indicates the number of bytes in the object being sent.
- The **Content-Type:** header line indicates that the object in the entity body is HTML text.

Conditional Request

A client can add a condition in its request. In this case, the server will send the requested web page if the condition is met or inform the client otherwise. One of the most common conditions imposed by the client is the time and date the web page is modified. The client can send the header line *If-Modified-Since* with the request to tell the server that it needs the page only if it is modified after a certain point in time.

Example

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

An HTTP cookie (also called web cookie, Internet cookie, browser cookie, or simply cookie) is a small piece of data sent from a website and stored on the user's computer by the user's web browser while the user is browsing. Cookies were designed to be a reliable mechanism for websites to remember stateful information (such as items added in the shopping cart in an online store) or to record the user's browsing activity (including clicking particular buttons, logging in, or recording which pages were visited in the past). They can also be used to remember arbitrary pieces of information that the user previously entered into form fields such as names, addresses, passwords, and credit card numbers.

It is often desirable for a Web site to identify users, either because the server wishes to restrict

user access or because it wants to serve content as a function of the user identity. For these purposes, HTTP uses cookies.

Creating and Storing Cookies

The creation and storing of cookies depend on the implementation; however, the principle is the same.

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 depending 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.

Using Cookies

When a client sends a request to a server, the browser looks in the cookie directory to see if it can find a cookie sent by that server. If found, the cookie is included in the request. When the server receives the request, it knows that this is an old client, not a new one. The contents of the cookie are never read by the browser or disclosed to the user. It is a cookie *made* by the server and *eaten* by the server.

- An e-commerce store uses a cookie to track client shopping. When an item is added to the cart, a cookie with the item number and unit price is sent to the browser. Each new selection updates the cookie. At checkout, the last cookie is retrieved to calculate the total charge.
- The site that restricts access to *registered clients* only sends a cookie to the client when the client registers for the first time. For any repeated access, only those clients that send the appropriate cookie are allowed.
- A web *portal* uses the cookie in a similar way. When a user selects her favourite pages, a cookie is made and sent. If the site is accessed again, the cookie is sent to the server to show what the client is looking for.
- Advertising agencies use cookies to track user behavior via banner ads on popular websites. When a user clicks a corporation's icon, a request goes to the agency, which sends the banner along with a cookie containing the user's ID. Future banner interactions build a profile of the user's web behavior. This data can be sold to third parties, making cookies controversial. New regulations are hoped to protect user privacy.

Ex:

Assume a shopper wants to buy a toy from an electronic store named BestToys.

Shopper initiates request: Shopper visits the BestToys website and requests to view toys.

Server creates cart: BestToys' server creates an empty shopping cart for this shopper, assigning a unique ID (e.g., 12343) to the cart.

Server sends response: Server sends a response with toy images, each with clickable links.

This response also includes a Set-Cookie header with the cart ID (12343), which is stored on the client's device in a file named BestToys.

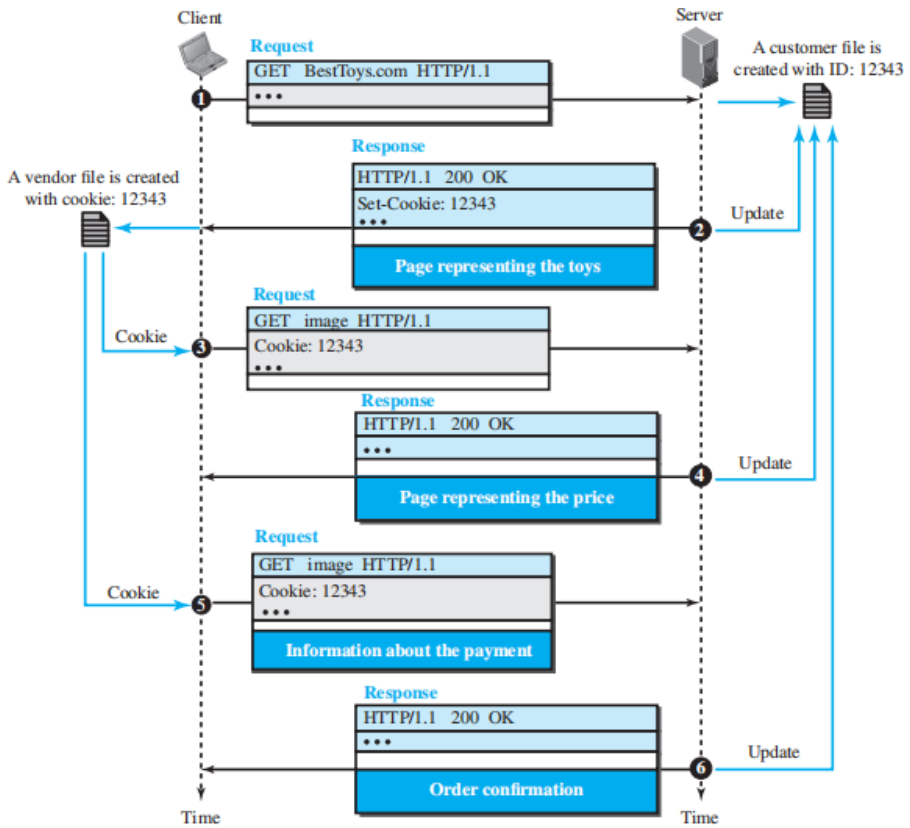
Shopper selects toy: Shopper clicks a toy to add to the cart; the client sends a request with the cookie (ID 12343) in the header, identifying the shopper.

Server retrieves cart: Server recognizes the returning ID 12343, retrieves the existing shopping cart, and adds the selected toy to the cart.

Server provides total price: Server responds with the updated total price and prompts the shopper for payment details.

Shopper makes payment: Shopper provides credit card information and submits it with the ID 12343 in the request.

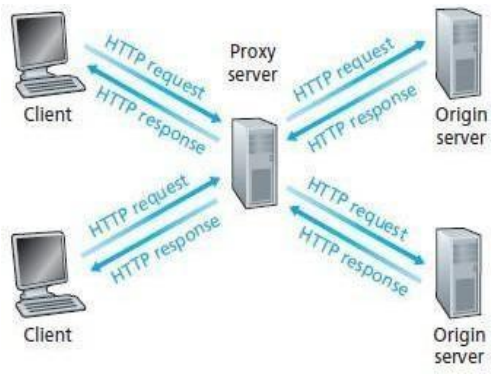
Order confirmation: Server confirms the ID, processes payment, completes the order, and sends a confirmation message to the shopper.



Web Caching

- A Web cache—also called a proxy server—is a network entity that satisfies HTTP requests on the behalf of an origin Web server.
- The Web cache has its own disk storage and keeps copies of recently requested objects in this storage.
- A user's browser can be configured so that all of the user's HTTP requests are first

directed to the Web cache.



Ex

Suppose a browser is requesting the object http://www.someschool.edu/campus.gif. Here is what happens:

1. The browser establishes a TCP connection to the Web cache and sends an HTTP request for the object to the Web cache.
2. The Web cache checks to see if it has a copy of the object stored locally. If it does, the Web cache returns the object within an HTTP response message to the client browser
3. If the Web cache does not have the object, the Web cache opens a TCP connection to the origin server, that *is, to www.someschool.edu. The Web cache then sends an HTTP request for the object* into the cache-to-server TCP connection.
4. After receiving this request, the origin server sends the object within an HTTP response to the Web cache.
5. When the Web cache receives the object, it stores a copy in its local storage and sends a copy, within an HTTP response message, to the client browser (over the existing TCP connection between the client browser and the Web cache).

HTTP Security

HTTP per se does not provide security. HTTP can be run over the Secure Socket Layer (SSL). In this case, HTTP is referred to as HTTPS. HTTPS provides confidentiality, client and server authentication, and data integrity.

FTP

FTP (File Transfer Protocol) is used for transferring file from one host to another host. Transferring files between systems isn't always simple, as systems can have different file name rules, data formats, or directory structures. FTP (File Transfer Protocol) addresses these issues smoothly, making it easier to move files between different systems. While HTTP can also transfer files, FTP is generally better for large files or when handling various file formats.

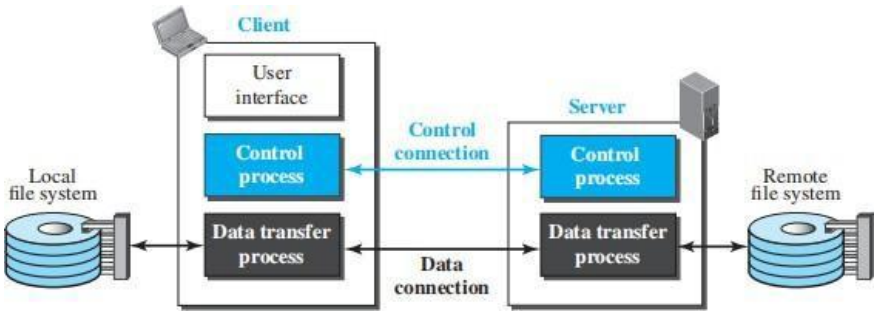


Figure shows the basic model of FTP. The client has three components: the user interface, the client control process, and the client data transfer process.
The server has two components: the server control process and the server data transfer process. The control connection is made between the control processes. The data connection is made between the data transfer processes.
Separation of commands and data transfer makes FTP more efficient. The control connection uses very simple rules of communication. We need to transfer only a line of command or a line of response at a time. The data connection, on the other hand, needs more complex rules due to the variety of data types transferred.

Two Connections
The two connections in FTP have different lifetimes. The control connection remains connected during the entire interactive FTP session. The data connection is opened and then closed for each file transfer FTP uses two well-known TCP ports: port 21 is used for the control connection, and port 20 is used for the data connection.

Control Connection
Control communication is achieved through commands and responses. This simple method is adequate for the control connection because we send one command (or response) at a time. Each line is terminated with a two-character (carriage return and line feed) end-of-line token. During this control connection, commands are sent from the client to the server and responses are sent from the server to the client. Commands, which are sent from the FTP client control process, are in the form of ASCII uppercase, which may or may not be followed by an argument. Some of the most common commands are shown in Table.

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

Every FTP command generates at least one response. A response has two parts: a three-digit number followed by text. The numeric part defines the code; the text part defines needed parameters or further explanations. The first digit defines the status of the command. The second digit defines the area in which the status applies. The third digit provides additional information. Table shows some common responses.

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

Data Connection

The data connection uses the well-known port 20 at the server site. However, the creation of a data connection is different from the control connection. The following shows the steps:

1. The client, not the server, issues a passive open using an ephemeral port. This must be done by the client because it is the client that issues the commands for transferring files.

2. Using the PORT command the client sends this port number to the server.

3. The server receives the port number and issues an active open using the well-known port 20 and the received ephemeral port number.

Communication over Data Connection

The purpose and implementation of the data connection are to transfer files through the data connection. The client must define the type of file to be transferred, the structure of the data, and the transmission mode.

Before sending the file through the data connection, we prepare for transmission through the control connection. The heterogeneity problem is resolved by defining three attributes of communication: file type, data structure, and transmission mode.

File Type

FTP can transfer one of the following file types across the data connection: ASCII file, EBCDIC file, or image file.

Data Structure

FTP can transfer a file across the data connection using one of the following interpretations of the structure of the data: *file structure, record structure,* or *page structure*.

Transmission Mode

FTP can transfer a file across the data connection using one of the following three transmission modes: *stream mode, block mode,* or *compressed mode*. The stream mode is the

default mode; data are delivered from FTP to TCP as a continuous stream of bytes. In the block mode, data can be delivered from FTP to TCP in blocks.

File Transfer

File transfer occurs over the data connection under the control of the commands sent over the control connection. However, we should remember that file transfer in FTP means one of three things: *retrieving a file* (server to client), *storing a file* (client to server), and *directory listing* (server to client).

Security for FTP

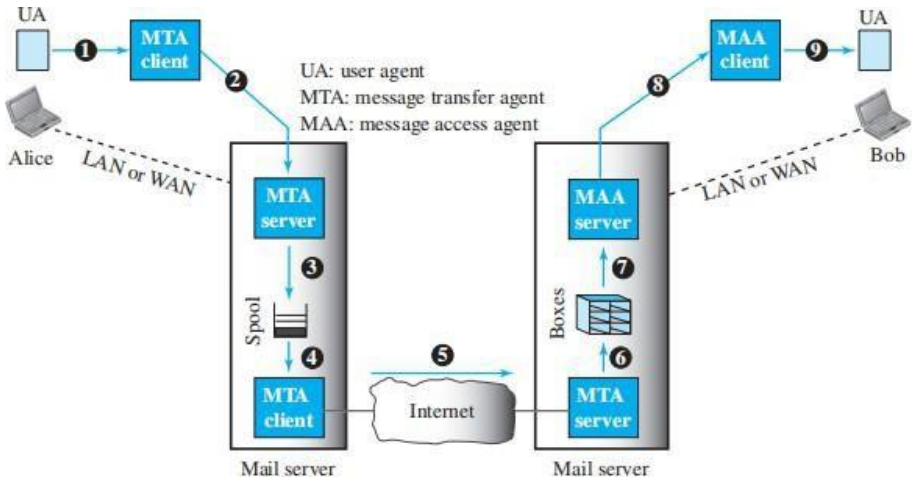
The FTP protocol was designed when security was not a big issue. Although FTP requires a password, the password is sent in plaintext (unencrypted), which means it can be intercepted and used by an attacker. The data transfer connection also transfers data in plain text, which is insecure. To be secure,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

ELECTRONIC MAIL

Electronic mail (or e-mail) allows users to exchange messages. In applications like HTTP or FTP, the server constantly awaits client requests to provide a response, whereas in email, messages are sent and stored until the recipient retrieves them. when Alice sends an email to Bob, a response is optional and, if given, is a separate transaction. Unlike typical client-server setups, Bob doesn't run a server to receive emails, as he might be offline. Instead, intermediate servers handle the client-server functions, allowing users to connect only when they wish to check or send messages.

Architecture

To explain the architecture of email, we present a typical scenario, as illustrated in Figure. An alternative scenario occurs when Alice or Bob is directly connected to their respective mail server, eliminating the need for a LAN or WAN connection; however, this variation does not affect the overall discussion.



In the common scenario, the sender and the receiver of the e-mail, Alice and Bob respectively, are connected via a LAN or a WAN to two mail servers. The administrator has created one mailbox for each user where the received messages are stored. A *mailbox* is part of a server hard drive, a special file with permission restrictions. Only the owner of the mailbox has access to it. The administrator has also created a queue (spool) to store messages waiting to be sent.
A simple e-mail from Alice to Bob takes nine different steps, as shown in the figure. Alice and Bob use three different *agents*: a **user agent (UA),** a **message transfer agent (MTA),** and a **message access agent (MAA).**

1. Alice composes an email using her User Agent (UA).
2. The UA sends the message to the Message Transfer Agent (MTA) client on Alice's mail server.
3. The MTA client forwards the message to the MTA server on the same mail server.
4. The email is placed in a spool (queue) on Alice's mail server, awaiting transfer.
5. The MTA client sends the email across the Internet to Bob's mail server.
6. The MTA server on Bob's mail server receives the email.
7. The email is stored in Bob's mailbox on his mail server.
8. Bob's Message Access Agent (MAA) server retrieves the message from the mailbox.
9. Bob uses his MAA client to access the message through his User Agent (UA).

There are two important points we need to emphasize here.

1. Bob cannot directly use the MTA server to receive messages, as this would require him to run the MTA server continuously, keeping his computer and network connection on all the time—an impractical setup, especially with a WAN connection.
2. Unlike the MTA client-server, which "pushes" messages to the server, Bob requires "pull" programs to retrieve messages. This is why he needs Message Access Agent (MAA) programs, allowing him to pull messages from the server when needed.

User Agent

The first component of an electronic mail system is the **user agent (UA).** A user agent is a software package (program) that composes, reads, replies to, and forwards messages. It also handles local mailboxes on the user computers.
There are two types of user agents: **command-driven and GUI-based**.
A command-driven user agent normally accepts a one-character command from the keyboard to perform its task. For example, a user can type the character *r*, at the command prompt, to reply to the sender of the message, or type the character *R* to reply to the sender and all recipients. Some examples of command driven user agents are *mail, pine,* and *elm*.
Modern user agents are GUI-based. They contain graphical user interface (GUI) components that allow the user to interact with the software by using both the keyboard and the mouse. Some examples of GUI-based user agents are *Eudora* and *Outlook***.**

Sending Mail

To send mail, the user, through the UA, creates mail that looks very similar to postal mail. It has an *envelope* and a *message*. The envelope usually contains the sender address, the receiver address, and other information. The message contains the *header* and the *body*. The header of the message defines the sender, the receiver, the subject of the message, and some other information. The body of the message contains the actual information to be read by the recipient.

Receiving Mail

The user agent is triggered by the user (or a timer). If a user has mail, the UA informs the user with a notice. If the user is ready to read the mail, a list is displayed in which each line contains a summary of the information about a particular message in the mailbox. The summary usually includes the sender mail address, the subject, and the time the mail was sent or received. The user can select any of the messages and display its contents on the screen.

Addresses

To deliver mail, a mail handling system must use an addressing system with unique addresses. In the Internet, the address consists of two parts: a *local part* and a *domain name,* separated by an @ sign.

The local part of an email address specifies the user's mailbox, while the domain part identifies the mail server or exchanger for sending and receiving mail.

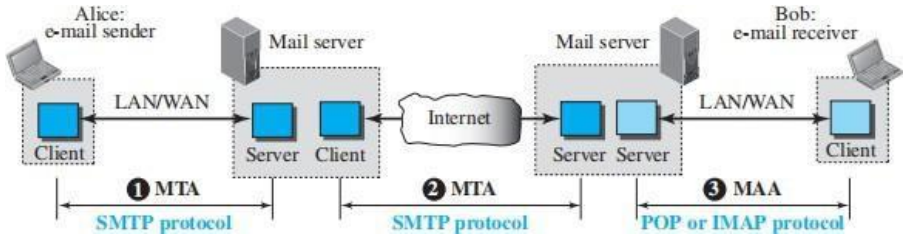
Mailing List or Group List

Electronic mail allows one name, an *alias,* to represent several different e-mail addresses; this is called a mailing list. Every time a message is to be sent, the system checks the recipient's name against the alias database; if there is a mailing list for the defined alias, separate messages, one for each entry in the list, must be prepared and handed to the MTA.

Message Transfer Agent: SMTP

An e-mail is an application that needs three uses of client-server paradigms to accomplish its task. It is important that we distinguish these three when we are dealing with e-mail. Figure shows these three client-server applications. We refer to the first and the second as Message Transfer Agents (MTAs), the third as Message Access Agent (MAA).

The formal protocol that defines the MTA client and server in the Internet is called ***Simple Mail Transfer Protocol (SMTP).***



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.

Mail Transfer Phases

The process of transferring a mail message occurs in three phases: connection estab lishment, mail transfer, and connection termination.

Connection Establishment

After a client has made a TCP connection to the well known port 25, the SMTP server starts the connection phase. This phase involves the following three steps:

1. The server sends code 220 (service ready) to tell the client that it is ready to receive mail. If the server is not ready, it sends code 421 (service not available).

2. The client sends the HELO message to identify itself, using its domain name address. This step is necessary to inform the server of the domain name of the client.

3. The server responds with code 250 (request command completed) or some other code depending on the situation.

Message Transfer

After connection has been established between the SMTP client and server, a single message between a sender and one or more recipients can be exchanged. This phase involves eight steps. Steps 3 and 4 are repeated if there is more than one recipient.

1. The client sends the MAIL FROM message to introduce the sender of the message. It includes the mail address of the sender (mailbox and the domain name). This step is needed to give the server the return mail address for returning errors and reporting messages.

2. The server responds with code 250 or some other appropriate code.

3. The client sends the RCPT TO (recipient) message, which includes the mail address of the recipient.

4. The server responds with code 250 or some other appropriate code.

5. The client sends the DATA message to initialize the message transfer.

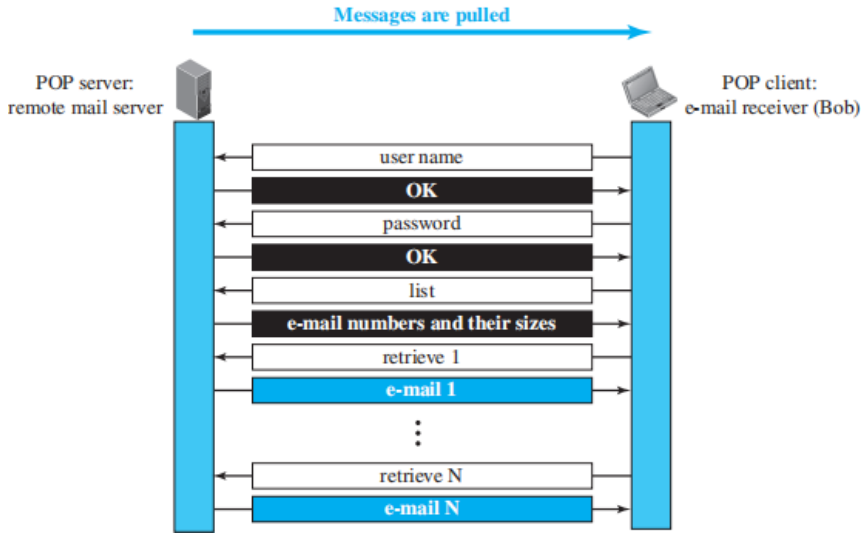
6. The server responds with code 354 (start mail input) or some other appropriate message.

7. The client sends the contents of the message in consecutive lines. Each line is terminated by a two-character end-of-line token (carriage return and line feed). The message is terminated by a line containing just one period.
8. The server responds with code 250 (OK) or some other appropriate code.

Connection Termination
After the message is transferred successfully, the client terminates 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.

Message Access Agent: POP and IMAP
SMTP protocol delivers the mail to the mail server (Push Protocol). To fetch the mail from mail server receiver used mail access protocols (Pull Protocol).There are currently a number of popular mail access protocols, including Post Office Protocol— Version 3 (POP3), Internet Mail Access Protocol (IMAP), and HTTP.

POP3
Post Office Protocol, version 3 (POP3) is simple but limited in functionality. The client POP3 software is installed on the recipient computer; the server POP3 software is installed on the mail server. Mail access starts with the client when the user needs to download its e-mail from the mailbox on the mail server. The client opens a connection to the server on TCP port 110. It then sends its user name and password to access the mailbox. The user can then list and retrieve the mail messages, one by one.



POP3 has two modes: the ***delete*** mode and the ***keep*** mode. In the delete mode, the mail is deleted from the mailbox after each retrieval. In the keep mode, the mail remains in the mailbox after retrieval.

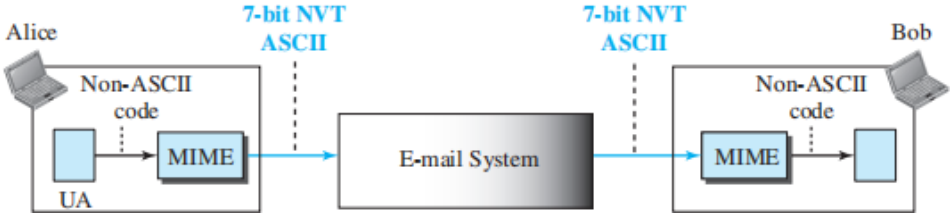
IMAP4

Another mail access protocol is **Internet Mail Access Protocol, version 4 (IMAP4).** IMAP4 is similar to POP3, but it has more features; IMAP4 is more powerful and more complex.
POP3 is deficient in several ways. It does not allow the user to organize her mail on the server; the user cannot have different folders on the server. In addition, POP3 does not allow the user to partially check the contents of the mail before downloading. IMAP4 provides the following extra functions:

- A user can check the e-mail header prior to downloading.
- A user can search the contents of the e-mail for a specific string of characters prior
- to downloading.
- A user can partially download e-mail. This is especially useful if bandwidth is limited and the e-mail contains multimedia with high bandwidth requirements.
- A user can create, delete, or rename mailboxes on the mail server.
- A user can create a hierarchy of mailboxes in a folder for e-mail storage.

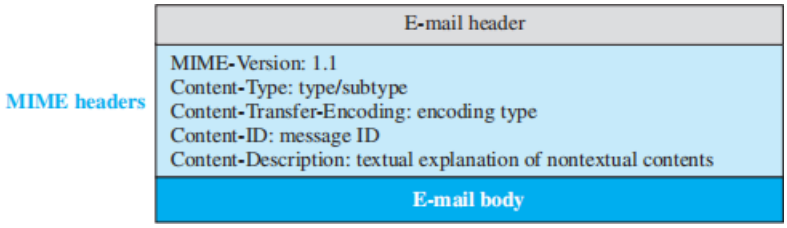
MIME

Email has a simple structure, but it is limited to sending messages in 7-bit ASCII format, which restricts its use for non-English languages and prevents sending binary files, video, or audio data.
Multipurpose Internet Mail Extensions (MIME) is a supplementary protocol that allows non-ASCII data to be sent through e-mail. MIME transforms non-ASCII data at the sender site to NVT ASCII data and delivers it to the client MTA to be sent through the Internet. The message at the receiving site is transformed back to the original data.
MIME is a set of software functions that transforms non-ASCII data to ASCII data and vice versa.



MIME Headers

MIME defines five headers, which can be added to the original 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

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)

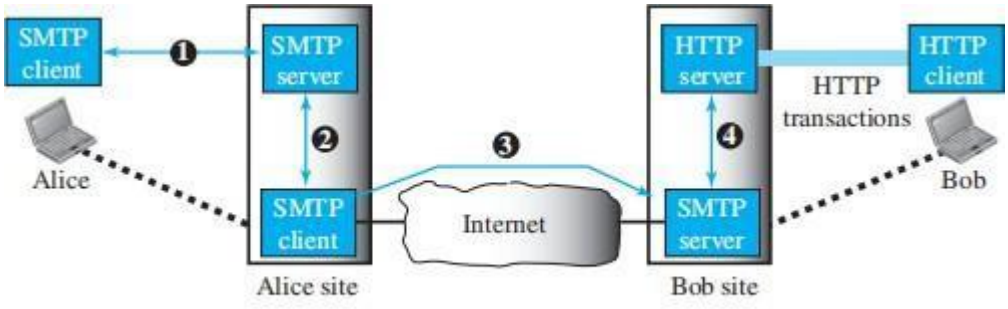
Content-Transfer-Encoding
This header defines the method used to encode the messages into 0s and 1s for transport. The five types of encoding methods are listed in Table.

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

Content-ID - This header uniquely identifies the whole message in a multiple message environment.

Content-Description -This header defines whether the body is image, audio, or video.

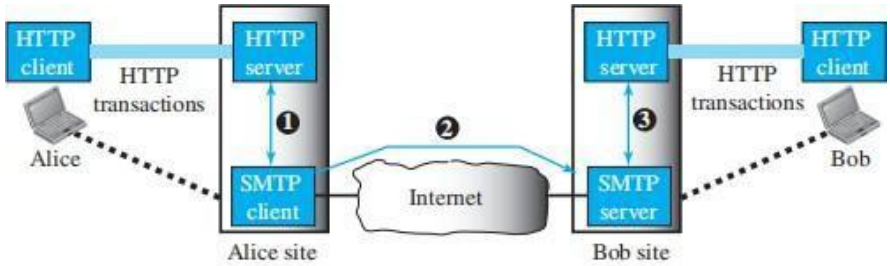
Web-Based Mail
E-mail is such a common application that some websites today provide this service to anyone who accesses the site. Three common sites are Hotmail, Yahoo, and Google mail. The idea is very simple. Figure shows two cases:



Case 1: Only receiver uses HTTP

Case I

In this scenario, Alice uses a traditional mail server to send an email to Bob, who has an account on a web-based server. The email is transferred from Alice's browser to her mail server via SMTP, and from the sending mail server to the receiving mail server through SMTP as well. However, when the message reaches Bob's web server, it is transferred to Bob's browser via HTTP, not POP3 or IMAP4. Bob requests his emails through HTTP by logging into the website (e.g., Hotmail). Once logged in, the web server sends a list of emails in HTML format, and Bob can browse and retrieve his emails using more HTTP transactions.



Case 2: Both sender and receiver use HTTP

Case II

In this case, both Alice and Bob use web servers, but not necessarily the same one. Alice sends her email to her web server using HTTP, with Bob's mailbox address as the URL. The Alice server then forwards the message to Bob's server using SMTP. Bob receives the email through HTTP transactions, but the transfer between servers still occurs via SMTP.

E-Mail Security

The email protocols do not include built-in security features. However, email exchanges can be secured using two application-layer security protocols, Pretty Good Privacy (PGP) and Secure/Multipurpose Internet Mail Extensions (S/MIME).

TELNET

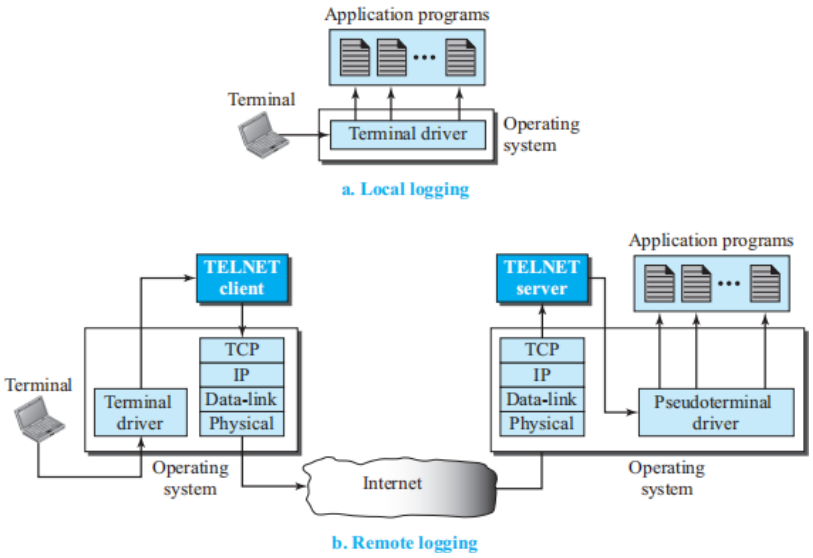
A specific client/server program is designed to provide a service, like FTP for file transfer. However, having a separate client/server pair for each service is not scalable. A solution is to use generic client/server programs that allow users to log into remote computers and use available services. For example, a student can log into a university server using a remote login program to access services like the Java compiler, without needing a separate client for each service. These generic programs are called **remote login applications**.

TELNET is an early remote login protocol that requires a username and password but sends all data, including the password, in plaintext, making it vulnerable to hacking. Due to security risks, TELNET has largely been replaced by **Secure Shell (SSH)**, which provides encrypted communication.

TELNET is briefly discussed because of its historical significance and for comparison with SSH.

1. The simple plaintext architecture of TELNET allows us to explain the issues and challenges related to the concept of remote logging, which is also used in SSH when it serves as a remote logging protocol.
2. Network administrators often use TELNET for diagnostic and debugging purposes.

Local versus Remote Logging



Local logging: When a user logs into their local system, keystrokes are processed by the terminal driver and passed to the operating system, which interprets them and invokes the appropriate application.

Remote logging: When accessing a program on a remote machine, TELNET is used. The user's keystrokes are sent to the terminal driver, but instead of being interpreted locally, they are passed to the TELNET client, which converts them into **Network Virtual Terminal**

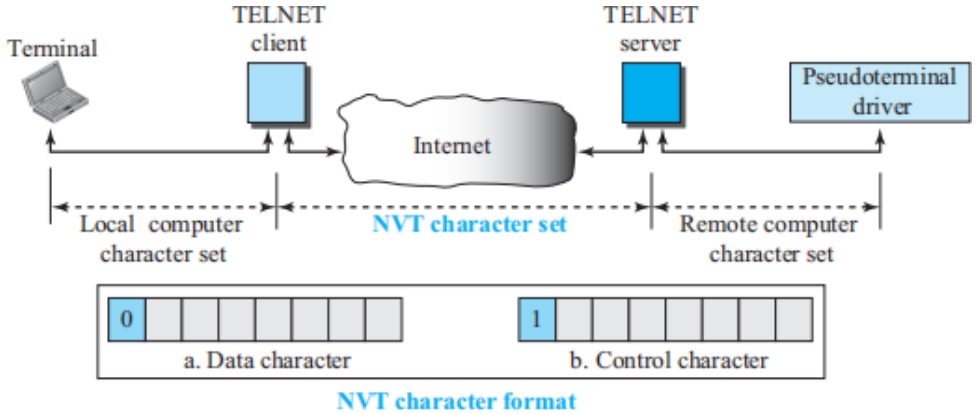
(NVT) characters. These NVT characters are sent through the network to the remote machine.

On the remote machine, the NVT characters are received by the TCP/IP stack and passed to the TELNET server. The server translates the NVT characters to ones the remote system can understand. Since the remote operating system can't directly process TELNET characters, a **pseudoterminal driver** is used to simulate a terminal input, allowing the characters to be passed to the correct application.

Network Virtual Terminal (NVT)

Accessing remote computers can be complex due to the different operating systems using unique character combinations as tokens (e.g., Ctrl+z for DOS and Ctrl+d for UNIX). To handle this complexity, TELNET provides a universal interface called the **Network Virtual Terminal (NVT)** character set.

- **NVT Character Set**: This allows TELNET to translate characters (data or commands) from the local terminal into NVT format for transmission across the network. On the remote side, TELNET translates the NVT characters back into a format understood by the remote computer.
- **Data and Control Characters**: NVT uses two sets of 8-bit characters:
  - **NVT ASCII**: For data, it uses a character set where the lowest 7 bits match US ASCII, and the highest bit is 0.
  - **Control Characters**: For control data, the highest bit is set to 1.



This system ensures that remote systems with different operating environments can communicate effectively.

Options

TELNET lets the client and server negotiate options before or during the use of the service. Options are extra features available to a user with a more sophisticated terminal. Users with simpler terminals can use default features.

User Interface

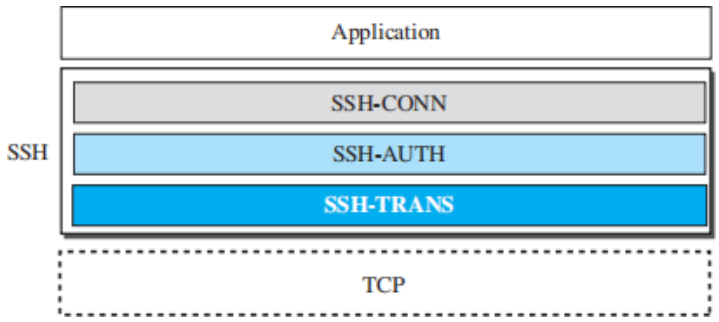
The operating system (UNIX, for example) defines an interface with user-friendly commands.

SECURE SHELL (SSH)

Although **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.

Components

SSH is an application-layer protocol with three components.



SSH Transport-Layer Protocol (SSH-TRANS)

Since TCP is not a secure transport-layer protocol, **SSH** uses an additional protocol called **SSH-TRANS** to create a secure channel on top of TCP. The process begins with the client and server establishing an insecure connection via TCP. They then exchange security parameters to establish a secure channel using SSH-TRANS.

The services provided by this protocol:

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)

After establishing a secure channel and authenticating the server, **SSH** proceeds to authenticate the client. The client sends a request to the server with the username, server name, authentication method, and required data. The server then responds with either a success message, confirming authentication, or a failure message, prompting the client to try again with a new request. This process is similar to client authentication in **SSL.**

SSH Connection Protocol (SSH-CONN)

Once the secure channel is established and both the server and client are authenticated, **SSH** uses the **SSH-CONN** protocol. One of its key features is **multiplexing**, which allows the client to create multiple logical channels over the secure channel. Each channel can be used for different purposes, such as remote logging or file transfer.

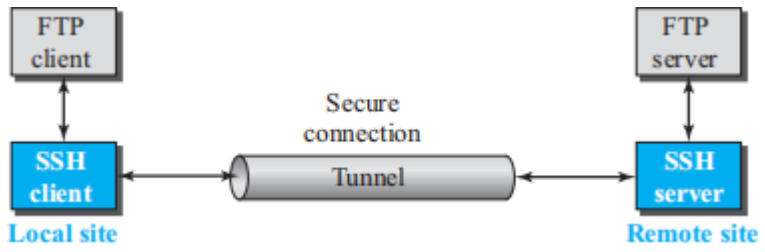
Applications

Although SSH is often thought of as a replacement for TELNET, SSH is, in fact, a general-purpose protocol that provides a secure connection between a client and server.

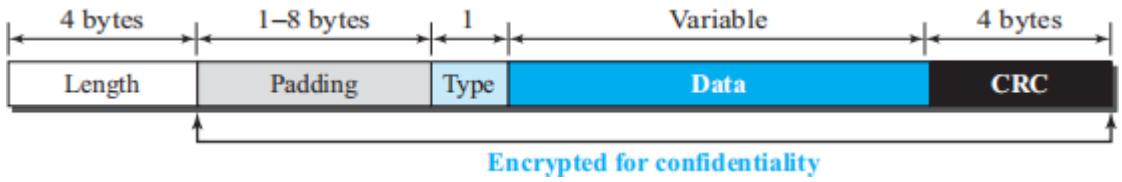
1. ***SSH for Remote Logging*** **-**Several free and commercial applications use SSH for remote logging.

2. ***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).

3. ***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 tunnelling.



Format of the SSH Packets

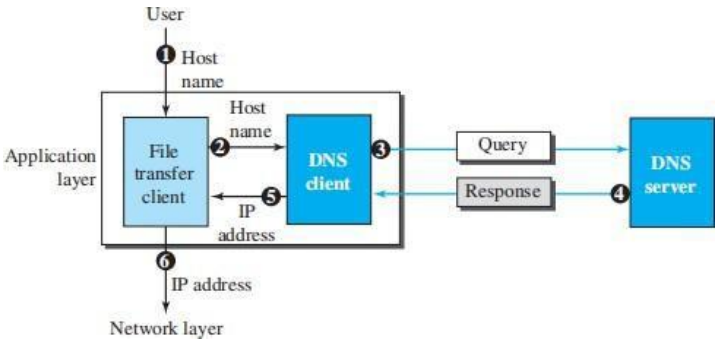


The length field defines the length of the packet but does not include the padding. One to eight bytes of padding is added to the packet to make the attack on the security provision more difficult. The cyclic redundancy check (CRC) field is used for error detection. The type field designates the type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

The Domain Name System (DNS) was developed to simplify access to Internet resources by mapping human-friendly names to IP addresses, which are needed for network identification. Similar to how a telephone directory helps map names to numbers, DNS serves as a directory for the Internet, allowing users to remember domain names rather than numeric IP addresses. A central directory for the entire Internet would be impractical due to its vast scale and vulnerability to failures. Instead, DNS information is distributed across multiple servers worldwide. When a host requires name-to-IP mapping, it contacts the nearest DNS server with the necessary information. This distributed structure enhances reliability and efficiency.

Figure shows how TCP/IP uses a DNS client and a DNS server to map a name to an address. A user wants to use a file transfer client to access the corresponding file transfer server running on a remote host. The user knows only the file transfer server name, such as *afilesource.com.* The TCP/IP suite needs the IP address of the file transfer server to make the connection.



The following six steps map the host name to an IP address:

1. The user passes the host name to the file transfer client.
2. The file transfer client passes the host name to the DNS client.
3. Each computer, after being booted, knows the address of one DNS server. The DNS client sends a message to a DNS server with a query that gives the file transfer server name using the known IP address of the DNS server.
4. The DNS server responds with the IP address of the desired file transfer server.
5. The DNS server passes the IP address to the file transfer client.
6. The file transfer client now uses the received IP address to access the file transfer server.

Name Space

To be unambiguous, the names assigned to machines must be carefully selected from a name space with complete control over the binding between the names and IP addresses. A name space that maps each address to a unique name can be organized in two ways: flat or hierarchical

Flat Name Space:

- Each name is a sequence of characters without inherent structure or meaning.
- Names might share common segments, but this doesn't imply any relationship.
- Centralized control is needed to avoid duplication, making a flat name space impractical for large systems like the Internet.

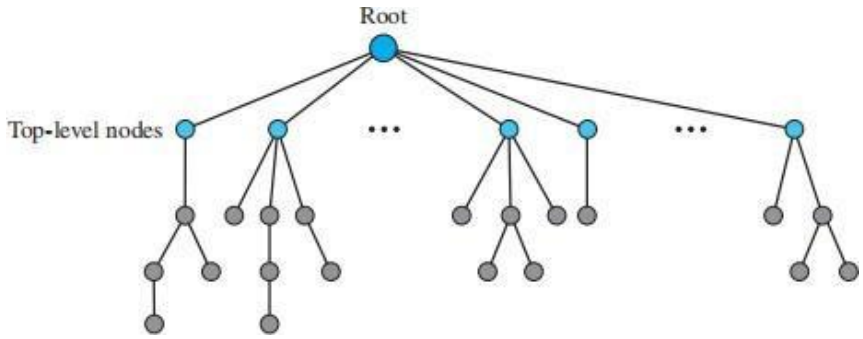
Hierarchical Name Space:

- Names consist of multiple parts, with each part representing a specific element, such as organization type, organization name, or department.
- Decentralization is possible since a central authority assigns only a portion of the name (e.g., organization type and name).
- Organizations manage the remaining parts, allowing unique identifiers even if different organizations use the same host name. For instance, "caesar.first.com" and "caesar.second.com" are distinct due to the organization names, ensuring *uniqueness* in large networks like the Internet.

This hierarchical structure is key to the *Domain Name System (DNS)*, enabling efficient, scalable, and decentralized name resolution across the Internet.

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 .

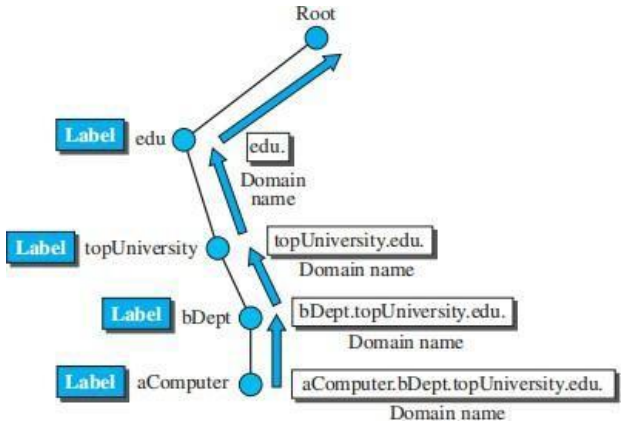


Label

Each node in the tree has a **label**, which is a string with a maximum of 63 characters. The root label is a null string (empty string). DNS requires that children of a node (nodes that branch from the same node) have different labels, which guarantees the uniqueness of the domain names.

Domain Name

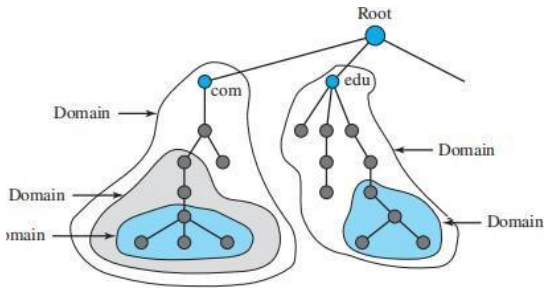
Each node in the tree has a domain name. A full **domain name** is a sequence of labels separated by dots (**.**). The domain names are always read from the node up to the root. The last label is the label of the root (null).



If a label is terminated by a null string, it is called a **fully qualified domain name (FQDN)**. The name must end with a null label, but because null means nothing, the label ends with a dot. If a label is not terminated by a null string, it is called a **partially** qualified **domain name (PQDN).** A PQDN starts from a node, but it does not reach the root. It is used when the name to be resolved belongs to the same site as the client. Here the resolver can supply the missing part, called the *suffix,* to create an FQDN.

Domain

A **domain** is a subtree of the domain name space. The name of the domain is the name of the node at the top of the subtree. Note that a domain may itself be divided into domains.

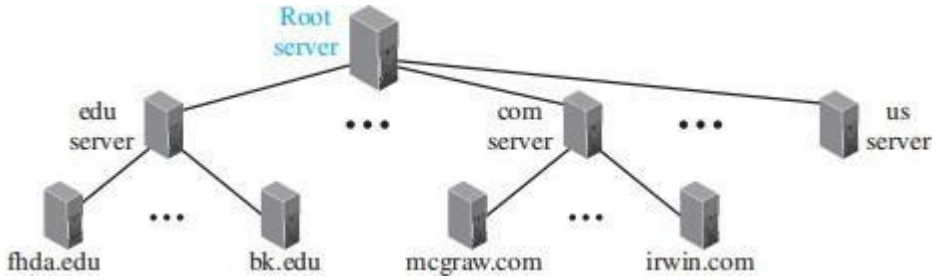


Distribution of Name Space

The information contained in the domain name space must be stored. However, it is very inefficient and also not reliable to have just one computer store such a huge amount of information. It is inefficient because responding to requests from all over the world places a heavy load on the system. It is not reliable because any failure makes the data inaccessible.

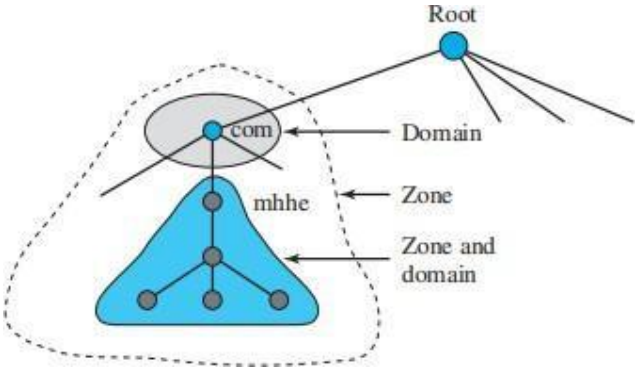
Hierarchy of Name Servers

The solution to these problems is to distribute the information among many computers called ***DNS servers.*** One way to do this is to divide the whole space into many domains based on the first level. In other words, we let the root stand alone and create as many domains (subtrees) as there are first-level nodes. Because a domain created this way could be very large, DNS allows domains to be divided further into smaller domains (subdomains). Each server can be responsible (authoritative) for either a large or small domain.



Zone

In DNS, a *zone* represents the part of the DNS hierarchy for which a server holds authority, facilitating distributed management. If a server is responsible for an entire domain without subdivisions, then the *zone* and *domain* are effectively the same, and the server keeps all relevant data in a *zone file*. However, if the server divides its domain into smaller *subdomains* and delegates authority for these to other servers, the *zone* differs from the *domain*. In this case, the main server retains responsibility for the broader domain but references lower-level servers that manage data for the specific subdomains. This hierarchical structure enables scalable and efficient distribution of DNS data across multiple servers.



Root Server

A **root server** is a server whose zone consists of the whole tree. A root server usually does not store any information about domains but delegates its authority to other servers, keeping references to those servers. There are several root servers, each covering the whole domain name space. The root servers are distributed all around the world.

Primary and Secondary Servers

DNS defines two types of servers: primary and secondary. A *primary server* is a server that stores a file about the zone for which it is an authority. It is responsible for creating, maintaining, and updating the zone file. It stores the zone file on a local disk.

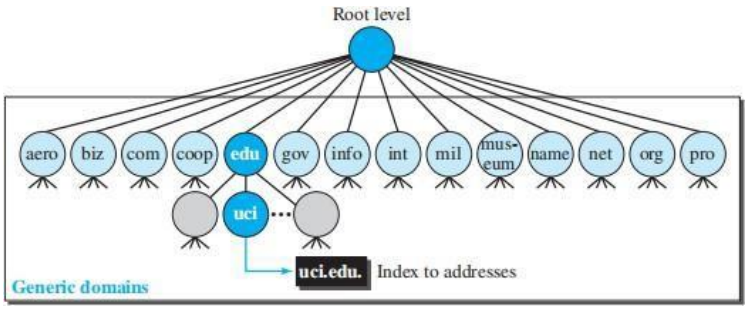
A *secondary server* is a server that transfers the complete information about a zone from another server (primary or secondary) and stores the file on its local disk. The secondary server neither creates nor updates the zone files. If updating is required, it must be done by the primary server, which sends the updated version to the secondary.

DNS in the Internet

DNS is a protocol that can be used in different platforms. In the Internet, the domain name space (tree) was originally divided into three different sections: generic domains, country domains, and the inverse domains. However, due to the rapid growth of the Internet, it became extremely difficult to keep track of the inverse domains, which could be used to find the name of a host when given the IP address. The inverse domains are now deprecated.

Generic Domains

The **generic domains** define registered hosts according to their generic behaviour. Each node in the tree defines a domain, which is an index to the domain name space database.



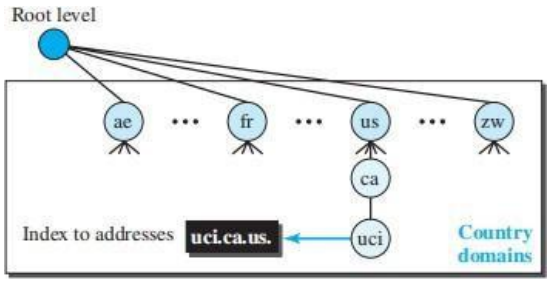
Looking at the tree, we see that the first level in the **generic domains** section allows 14 possible labels. These labels describe the organization types as listed in Table

Table 26.12 *Generic domain labels*

Label	*Description*	*Label*	*Description*
aero	Airlines and aerospace	**int**	International organizations
biz	Businesses or firms	**mil**	Military groups
com	Commercial organizations	**museum**	Museums
coop	Cooperative organizations	**name**	Personal names (individuals)
edu	Educational institutions	**net**	Network support centers
gov	Government institutions	**org**	Nonprofit organizations
info	Information service providers	**pro**	Professional organizations

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.). The address ***uci.ca.us.*** can be translated to University of California, Irvine, in the state of California in the United States.

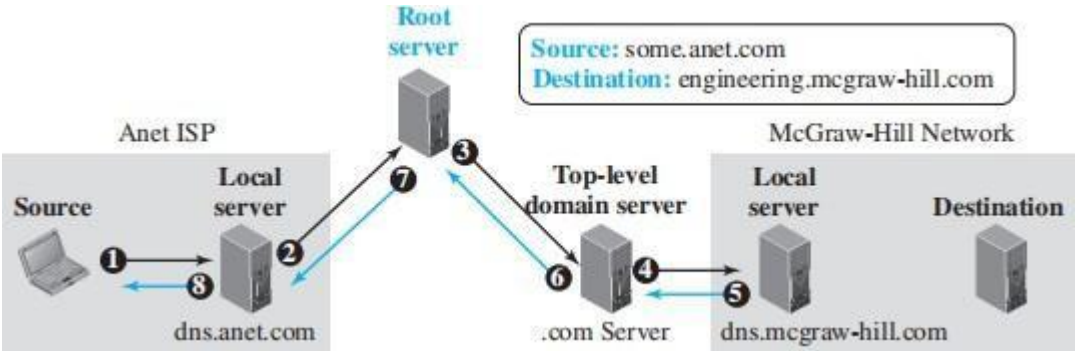


Resolution

Mapping a name to an address is called *name-address resolution.* DNS is designed as a client-server application. A host that needs to map an address to a name or a name to an address calls a DNS client called a ***resolver.*** The resolver accesses the closest DNS server with a mapping request. If the server has the information, it satisfies the resolver; otherwise, it either refers the resolver to other servers or asks other servers to provide the information. After the resolver receives the mapping, it interprets the response to see if it is a real resolution or an error, and finally delivers the result to the process that requested it. A resolution can be either recursive or iterative.

Recursive Resolution

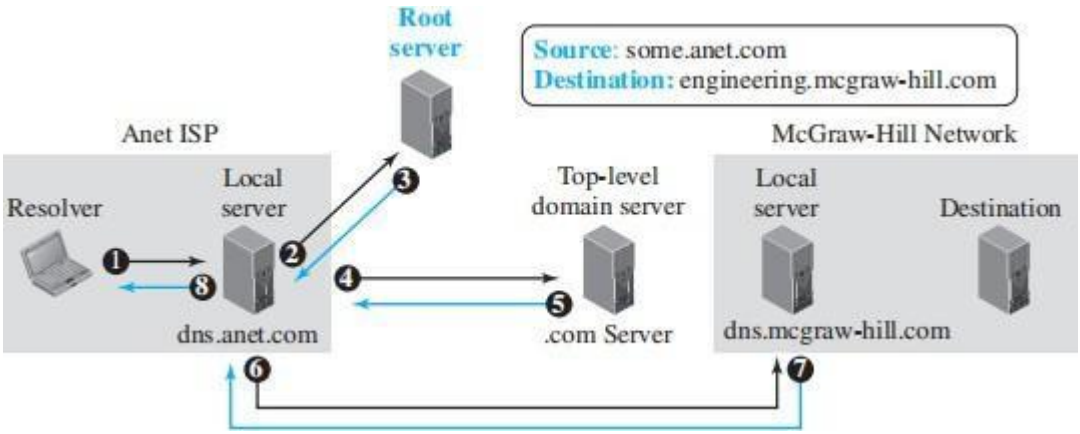
In a *recursive DNS resolution*, an application on a host (e.g., some.anet.com) needs the IP address of another host (e.g., engineering.mcgraw-hill.com) to send a message. The source host's *DNS resolver* (client) queries its local DNS server (dns.anet.com), which doesn't have the address and forwards the request to a root DNS server. The root server, lacking the specific mapping, knows the relevant top-level domain server (e.g., for *.com*) and forwards the query there. This server, not having the exact address, directs the query to McGraw-Hill's local DNS server (dns.mcgraw-hill.com), which finally provides the destination IP. This IP address is sent back step-by-step—from McGraw-Hill's DNS server to the top-level DNS server, then to the root server, the ISP's DNS server (which caches it), and ultimately back to the source host.



Iterative Resolution

In **iterative resolution,** each server that does not know the mapping sends the IP address of the next server back to the one that requested it. Figure shows the flow of information in an iterative resolution. Normally the iterative resolution takes place between two local servers; the original resolver gets the final answer from the local server. Note that the

messages shown by events 2, 4, and 6 contain the same query. However, the message shown by event 3 contains the IP address of the top-level domain server, the message shown by event 5 contains the IP address of the McGraw-Hill local DNS server, and the message shown by event 7 contains the IP address of the destination. When the Anet local DNS server receives the IP address of the destination, it sends it to the resolver (event 8).



Caching

To improve efficiency, DNS servers use *caching*. When a server requests a name-to-IP mapping from another server and receives a response, it stores the information in its *cache memory*. If the same or another client requests that mapping again, the server can quickly retrieve it from the cache, marked as *unauthoritative* to indicate it's not from the original source.

While caching speeds up DNS resolution, it can lead to issues if a mapping becomes outdated. To manage this, each authoritative server includes a *time-to-live (TTL)* value with the mapping, specifying how long the receiving server can cache it before it becomes invalid. DNS servers also keep a TTL counter for each cached mapping, regularly checking and purging entries with expired TTLs to maintain up-to-date data.

Resource Records

The zone information associated with a server is implemented as a set of *resource records*. A *resource record* is a 5-tuple structure, as shown below:

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

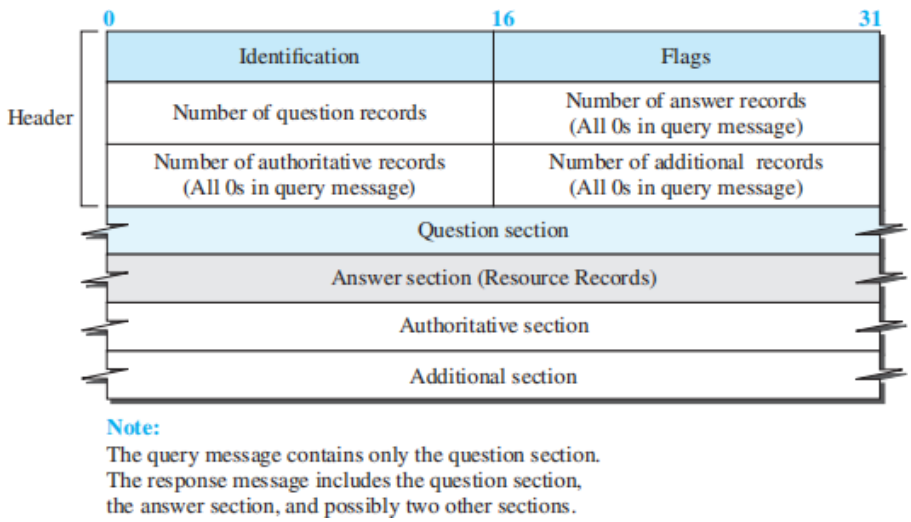
The domain name field is what identifies the resource record. The value defines the information kept about the domain name. The TTL defines the number of seconds for

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. Table lists the common types and how the value is interpreted for each type.

Type	*Interpretation of value*
A	A 32-bit IPv4 address (see Chapter 18)
NS	Identifies the authoritative servers for a zone
CNAME	Defines an alias for the official name of a host
SOA	Marks the beginning of a zone
MX	Redirects mail to a mail server
AAAA	An IPv6 address (see Chapter 22)

DNS Messages

To retrieve information about hosts, DNS uses two types of messages: *query* and *response*. Both types have the same format.



- The identification field is used by the client to match the response with the query.
- The flag field defines whether the message is a query or response. It also includes status of error.
- The next four fields in the header define the number of each record type in the message.
- The question section consists of one or more question records. It is present in both query and response messages.
- The answer section consists of one or more resource records. It is present only in response messages.

- The authoritative section gives information (domain name) about one or more authoritative servers for the query.
- The additional information section provides additional information that may help the resolver.

Registrars

New domains are added to the DNS through *registrars*, commercial entities accredited by *ICANN*. Registrars verify that the requested domain name is unique, enter it into the DNS database, and charge a fee. Today, there are many registrars; their names and addresses can be found at

http://www.intenic.net

To register, the organization needs to give the name of its server and the IP address of the server. For example, a new commercial organization named wonderful with a server named ws and IP address 200.200.200.5 needs to give the following information to one of the registrars:

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

DDNS

The *Dynamic Domain Name System (DDNS)* was created to automate DNS updates, accommodating frequent address changes on today's Internet. Instead of manual updates, DDNS allows changes like adding or removing hosts or updating IP addresses to be made dynamically. Typically, when a new name-to-address binding is established (e.g., via DHCP), it's sent to the *primary DNS server*, which updates the *zone*. *Secondary servers* learn of these changes through *active* or *passive notifications*—either the primary server sends an alert, or secondary servers periodically check for updates. After notification, secondary servers perform a *zone transfer* to receive updated information. To prevent unauthorized changes, DDNS can implement an *authentication mechanism*.

Security of DNS

DNS is one of the most important systems in the Internet infrastructure; it provides crucial services to Internet users. Applications such as Web access or e-mail are heavily dependent on the proper operation of DNS. DNS can be attacked in several ways including:

1. The attacker may read the response of a DNS server to find the nature or names of sites the user mostly accesses. This type of information can be used to find the user's profile. To prevent this attack, DNS messages need to be confidential

2. The attacker may intercept the response of a DNS server and change it or create a totally new bogus response to direct the user to the site or domain the attacker wishes the user to access. This type of attack can be prevented using message origin authentication and message integrity

3. The attacker may flood the DNS server to overwhelm it or eventually crash it. This type of attack can be prevented using the provision against denial-of-service attack.

To protect DNS, IETF has devised a technology named *DNS Security* (DNSSEC) that provides message origin authentication and message integrity using a security service called *digital signature* DNSSEC, however, does not provide confidentiality for the DNS messages. There is no specific protection against the denial-of service attack in the specification of DNSSEC. However, the caching system protects the upper-level servers against this attack to some extent.