Unit-2

19ECS-232

COMPUTER NETWORKS

B.SRIDEVI

ASSISTANT PROFESSOR

DEPARTMENT OF CSE

**SYLLABUS**

**Application Layer:** Principles of Network Applications, The Web and HTTP, Electronic Mail in the Internet, DNS - The Internet’s Directory Service, Socket Programming: Creating Network Applications

**APPLICATION LAYER**

The applications have been the driving force behind the Internet’s success, motivating people in homes, schools, governments, and businesses to make the Internet an integral part of their daily activities.

-Some of the Network applications are E-mail, instant messaging, P2P file sharing, multi-user network games, streaming stored video clips, internet telephone, real-time video conference, social networking.

**PRINCIPLES OF NETWORK APPLICATIONS**

-The core of network application development is writing programs that run on different end systems and communicate with each other over the network.

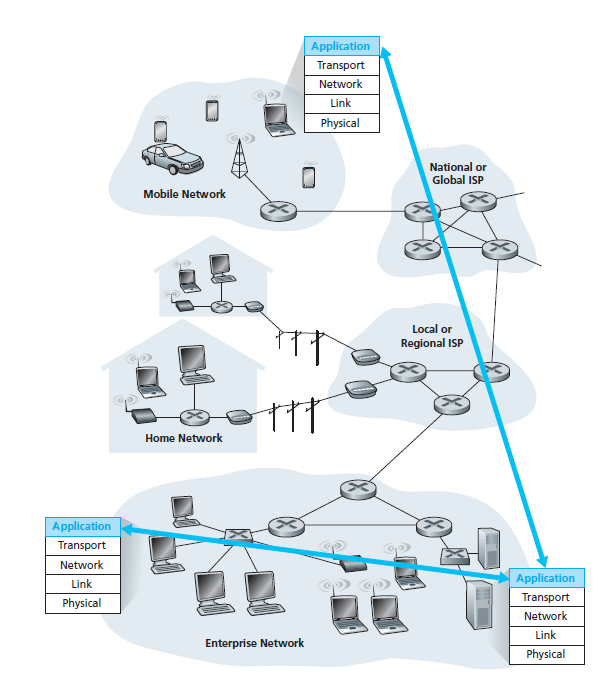
-Example, in the Web application there are two distinct programs that communicate with each other:

* The browser program running in the user’s host and
* The Web server program running in the Web server host.

- When developing a new application, the software could be written in any programming language that will run on multiple end systems.

-It is not needed to write software that runs on network core devices, such as routers or link-layer switches as network-core devices do not run user applications.

-Application software is confined to the end systems.



**Fig:** Communication for a network application takes place between end systems at the application layer

* **Network Application Architectures**

-An application’s architecture is distinctly different from the network architecture.

* The network architecture is fixed and provides a specific set of services to applications.
* The application architecture, on the other hand, is designed by the application developer and dictates how the application is structured over the various end systems.

-The two predominant architectural paradigms used in modern network applications are:

* The client-server architecture (or)
* The peer-to-peer (P2P) architecture.

**Client-Server Architecture**

-There is an always-on host, called the server, which provides services to requests from many other hosts, called clients.

-A classic example is the Web application for which an always-on Web server services requests from browsers running on client hosts.

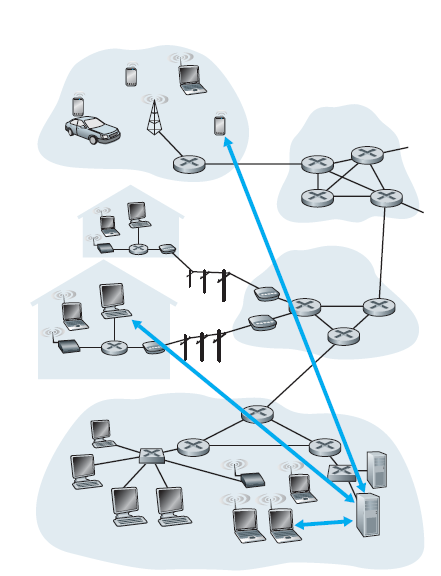
* When a Web server receives a request for an object from a client host, it responds by sending the requested object to the client host.
* In the client-server architecture, clients do not directly communicate with each

other.

-The client-server architecture is that the server has a fixed, well-known address, called an IP address.

-In a client-server application, a single-server host is incapable of keeping up with all the requests from clients.

* For this reason, a **data center**, housing a large number of hosts, is often used to create a powerful virtual server.
* A data center can have hundreds of thousands of servers, which must be powered and maintained.



**Fig:** Client-server architecture

**P2P Architecture**

-There is minimal or no reliance on dedicated servers in data centers.

-The application exploits direct communication between connected hosts, called peers.

-The peers are instead desktops and laptops controlled by users, with most of the peers residing in homes, universities, and offices.

-Because the peers communicate without passing through a dedicated server, the architecture is called peer-to-peer.

-Most popular and traffic-intensive applications are based on P2P architectures that include file sharing (e.g., BitTorrent), peer-assisted download acceleration, Internet Telephony (e.g., Skype), and IPTV.

-Some applications have hybrid architectures, combining both client-server and P2P elements.

-One of the most compelling features of P2P architectures is

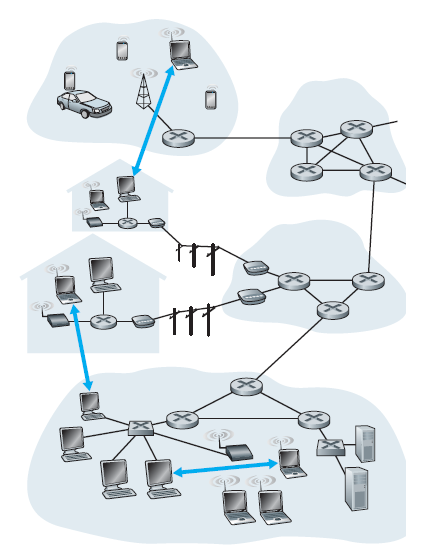
* **Self-scalability:** New peers bring new service capacity, as well as new service demands.
* P2P architectures are also cost effective, since they normally don’t require significant server infrastructure and server bandwidth.

-Future P2P applications face three major challenges:

* **ISP Friendly:** Most residential ISPs (including DSL and cable ISPs) have been

dimensioned for “asymmetrical” bandwidth usage, for much more downstream than upstream traffic. But P2P video streaming and file distribution applications shift upstream traffic from servers to residential ISPs, thereby future P2P applications need to be designed so that they are friendly to ISPs.

* **Security:** Because of their highly distributed and open nature, P2P applications can be a challenge to secure.
* **Incentives:** The success of future P2P applications also depends on convincing users to volunteer bandwidth, storage, and computation resources to the applications.



**Fig:** P2P architecture

* **Processes Communicating**

-In general, the programs, running in multiple end systems, communicate with each other. -In the jargon of operating systems, it is not actually programs but **processes** that communicate.

-A process can be thought of as a program that is running within an end system.

* When processes are running on the same end system, they can communicate with each other with inter-process communication, using rules that are governed by the end system’s operating system.

-Processes on two different end systems communicate with each other by exchanging **messages** across the computer network.

**Client and Server Processes**

-A network application consists of pairs of processes that send messages to each other over a network.

-Typically, one of the two processes is called as the **client** and the other process as the **server**.

* The process that initiates the communication (that is, initially contacts the other process at the beginning of the session) is labelled as the **client**.
* The process that waits to be contacted to begin the session is the **server**.

**The Interface Between the Process and the Computer Network**

-A process sends messages into, and receives messages from, the network through a software interface called a **socket**.

-A process is analogous to a house and its socket is analogous to its door.

* When a process wants to send a message to another process on another host, it

pushes the message out its door (socket).

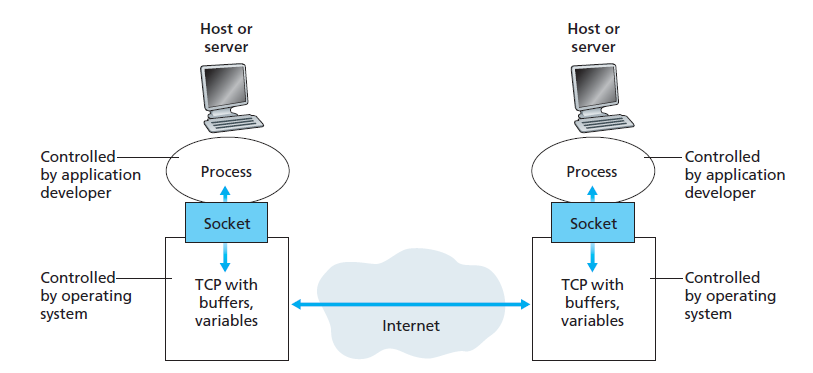
* This sending process assumes that there is a transportation infrastructure on the other side of its door that will transport the message to the door of the destination process.
* Once the message arrives at the destination host, the message passes through the receiving process’s door (socket), and the receiving process then acts on the message.

-Socket is the interface between the application layer and the transport layer within a host.

-It is also referred to as the **Application Programming Interface** **(API)** between the application and the network.

-The application developer has control of everything on the application-layer side of the socket but has little control of the transport-layer side of the socket such as

* The choice of transport protocol and
* The ability to fix a few transport-layer parameters such as maximum buffer and maximum segment sizes.



**Fig:** Application processes, sockets, and underlying transport protocol

**Addressing Processes**

-A process running on one host to send packets to a process running on another host, the receiving process needs to have an address.

-To identify the receiving process, two pieces of information need to be specified:

(1) The address of the host and

(2) An identifier that specifies the receiving process in the destination host.

-In the Internet, the host is identified by its **IP address**- a 32-bit quantity.

-In addition to knowing the address of the host, the sending process must also identify the receiving process running in the host, as a host could be running many network applications.

* A destination **port number** serves this purpose.

-Popular applications have been assigned specific port numbers.

* A Web server is identified by port number 80.
* A mail server process (using the SMTP protocol) is identified by port number 25.
* **Transport Services Available to Applications**

**-**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.

-The Internet, provides more than one transport-layer protocol.

-To choose transport-layer protocols the services provided by the available transport-layer protocols should be known.

-The possible services can be broadly classified along four dimensions:

* Reliable data transfer
* Throughput
* Timing
* Security.

**Reliable Data Transfer**

**-**The applications, has to be guaranteed that the data sent by one end of the application is delivered correctly and completely to the other end of the application.

-If a protocol provides such a guaranteed data delivery service, it is said to provide **reliable data transfer**.

* One important service that a transport-layer protocol can potentially provide to an application is process-to-process reliable data transfer.

-They are **loss-tolerant applications** which areacceptable, most notably multimedia applications such as conversational audio/video that can tolerate some amount of data loss.

* In these multimedia applications, lost data might result in a small glitch in the

audio/video—not a crucial impairment.

**Throughput**

-The context of a communication session between two processes along a network path, is the rate at which the sending process can deliver bits to the receiving process.

-Transport-layer protocol could provide, guaranteed available throughput at some specified rate.

-Applications that have throughput requirements are said to be **bandwidth-sensitive applications**.

* Many current multimedia applications are bandwidth sensitive.

**-Elastic applications** can make use of as much, or as little, throughput as happens to be available.

* Electronic mail, file transfer, and Web transfers are all elastic applications.

**Timing**

- A transport-layer protocol can also provide timing guarantees.

-With throughput guarantees, timing guarantees can come in many shapes and forms.

-A long delay between taking an action and seeing the response from the environment

makes the application feel less realistic.

* A service would be appealing to interactive real-time applications, such as Internet telephony, virtual environments, teleconferencing, and multiplayer games, all of which requires tight timing constraints on data delivery in order to be effective.

**Security**

-A transport protocol can provide an application with one or more security services.

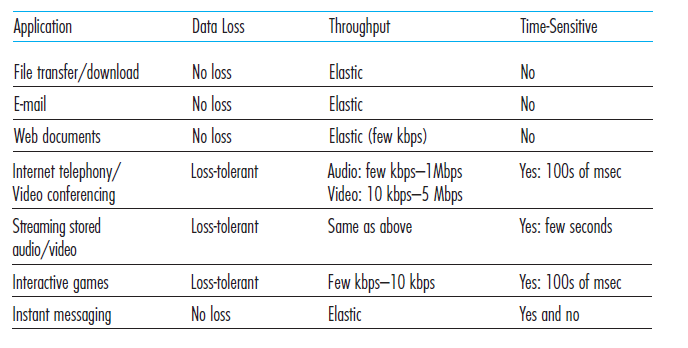
* In the sending host, a transport protocol can encrypt all data transmitted by the sending process, and in the receiving host, the transport-layer protocol can decrypt the data before delivering the data to the receiving process.

-A transport protocol can also provide other security services in addition to confidentiality, including data integrity and end-point authentication.

* **Transport Services Provided by the Internet**

-The Internet makes two transport protocols available to applications, UDP and TCP.

-The service requirements for some selected applications are:



**TCP Services**

-The TCP service model includes a connection-oriented service and a reliable data transfer service.

* + **Connection-oriented service:**
* TCP has the client and server which exchange transport layer control information with each other before the application-level messages begin to flow. This is called handshaking procedure.
* After the handshaking phase, a **TCP connection** is said to exist between the sockets of the two processes.
* The connection is a full-duplex connection in which the two processes can send messages to each other at the same time.
* When the application finishes sending messages, it must tear down the connection.
  + **Reliable data transfer service:**
* The communicating processes can rely on TCP to deliver all data sent without error and in the proper order.
  + TCP also includes a congestion-control mechanism. The TCP congestion-control mechanism throttles a sending process when the network is congested between sender and receiver.

**UDP Services**

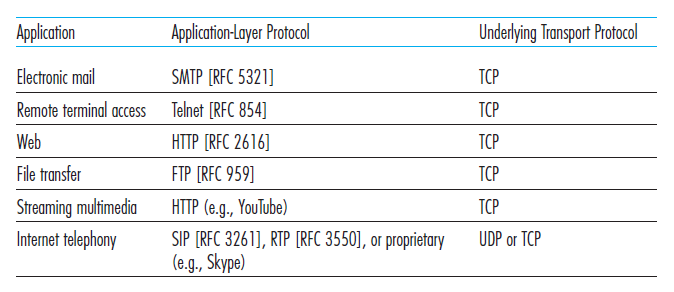
-UDP is a no-frills, lightweight transport protocol, providing minimal services.

-UDP is connectionless, so there is no handshaking before the two processes start to communicate.

-It provides an unreliable data transfer service—that is, when a process sends a message into a UDP socket, UDP provides no guarantee that the message will ever reach the receiving process.

-UDP does not include a congestion-control mechanism.

The transport protocols used by some popular Internet applications and their underlying transport protocol are:



* **­** **Application-Layer Protocols**

**-**An application-layer protocoldefines how an application’s processes, running on different end systems, pass messages to each other.

**-**In particular, an application-layer protocol defines:

* + The types of messages exchanged, for example, request messages and response

messages.

* + The syntax of the various message types, such as the fields in the message and

how the fields are delineated.

* + The semantics of the fields, that is, the meaning of the information in the fields
  + Rules for determining when and how a process sends messages and responds to

messages.

-It is important to distinguish between network applications and application-layer protocols.

-An application-layer protocol is only one piece of a network application.

* The Web application consists of many components, including a standard for document formats (that is, HTML), Web browsers (for example, Firefox and Microsoft Internet Explorer), Web servers (for example, Apache and Microsoft servers), and an application-layer protocol.

**THE WEB AND HTTP**

-The Internet was used primarily by researchers, academics, and university students to transfer files.

-In the early 1990s, a major new application arrived which called as the World Wide Web.

-The Web was the first Internet application that caught the general public’s eye which dramatically changed, and continues to change.

-It elevated the Internet from just one of many data networks to essentially the one and only data network.

-Web operates on demand.

* Users receive what they want, when they want it.
* **Overview of HTTP**

-The **Hyper Text Transfer Protocol (HTTP)**, the Web’s application-layer protocol,

is at the heart of the Web.

-It is defined in [RFC 1945] and [RFC 2616].

-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.

-A **Web page** consists of objects.

* An **object** is simply a file such as an HTML file, a JPEG image, a Java applet, or a video clip that is addressable by a single URL.
* Consider the page consists of a base HTML file and 10 JPEG images, and that all 11 of these objects reside on the same server.

-Each URL has two components: the hostname of the server and the object’s path name.

-For example, the URL http://www.someSchool.edu/someDepartment/picture.gif

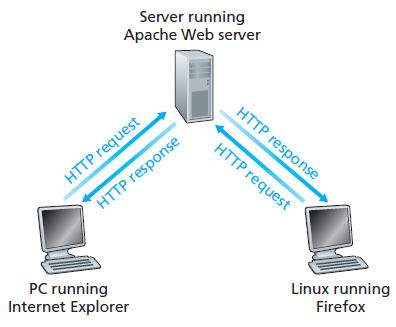
* www.someSchool.edu for a hostname and /someDepartment/picture.gif for a path name.

-HTTP defines how Web clients request Web pages from Web servers and how servers transfer Web pages to clients.

-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 client sends HTTP request messages into its socket interface and receives HTTP response messages from its socket interface.
* Similarly, the HTTP server receives request messages from its socket interface and sends response messages into its socket interface.

-An HTTP server maintains no information about the clients, HTTP is said to be a **stateless protocol**.



**Fig:** HTTP request-response behavior

* **Non-Persistent and Persistent Connections**

The client and server communicate for an extended period of time, with the client making a series of requests and the server responding to each of the requests.

**HTTP with Non-Persistent Connections**

-Each request/response pair is been sent over a separate TCP connection.

-Consider the URL for the base HTML file is

http://www.someSchool.edu/someDepartment/home.index

-Here what happens is:

1. The HTTP client process initiates a TCP connection to the server www.someSchool.edu on port number 80, which is the default port number for HTTP.

2. The HTTP client sends an HTTP request message to the server via its socket.

* The request message includes the path name /someDepartment/home.index.

3. The HTTP server process receives the request message via its socket, retrieves

the object /someDepartment/home.index from its storage, encapsulates the object in an HTTP response message, and sends the response message to the client via its socket.

4. The HTTP server process tells TCP to close the TCP connection.

5. The HTTP client receives the response message and the TCP connection terminates.

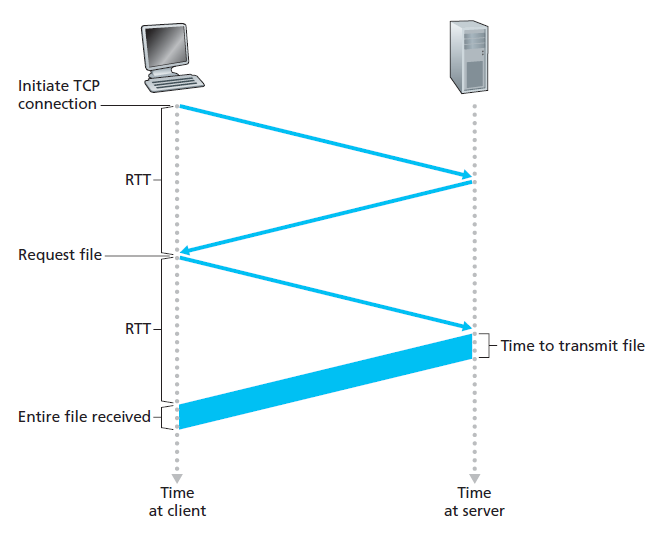
6. The first four steps are then repeated for each of the requested objects.

The use of non-persistent connections, where each TCP connection is closed after the server sends the object. TCP connection transports exactly one request message and one response message.

**Response Time**

**-**The **round-trip time (RTT)**, is defined as the time it takes for a small packet to travel from client to serverand then back to the client.

-The RTT includes packet-propagation delays, packet queuingdelays in intermediate routers and switches, and packet-processingdelays.



-To initiate a TCP connection between the browser and the Web server; itinvolves a “three-way handshake.

* 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.
* 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.

**HTTP with Persistent Connections**

-Non-persistent connections have some shortcomings.

* New connection must be established and maintained for each requested object.
* Firstly, for each of these connections, TCP buffers must be allocated.
* This can place a significant burden on the Web server, which may be serving requests from hundreds of different clients simultaneously.
* Secondly, 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.

-Multiple Web pages residing on the same server can be sent from the server to the same client over a single persistent TCP connection.

-The requests for objects can be made back-to-back.

-When the server receives the back-to-back requests, it sends the objects back-to-back.

-Typically, the HTTP server closes a connection when it isn’t used for a certain time.

* **HTTP Message Format**

-There are two types of HTTP messages

* Request messages
* Response messages

**HTTP Request Message**

-A typical HTTP request message:

GET /somedir/page.html HTTP/1.1

Host: [www.someschool.edu](http://www.someschool.edu)

Connection: close

User-agent: Mozilla/5.0

Accept-language: fr

-First, the message is written in ordinary ASCII text, so that an ordinary computer-literate human being can read it.

-Second, the message consists of five lines, each followed by a carriage return and a line feed.

-A request message can have many more lines or as few as one line.

-The first line of an HTTP request message is called the **request line** and the subsequent lines are called the **header lines**.

-The request line has three fields:

* The method field,
* The URL field, and
* The HTTP version field.
* The method field can take on several different values, including GET, POST, HEAD, PUT, and DELETE.
* The majority of HTTP request messages use the GET method.
* The GET method is used when the browser requests an object, the browser is requesting the object /somedir/page.html.

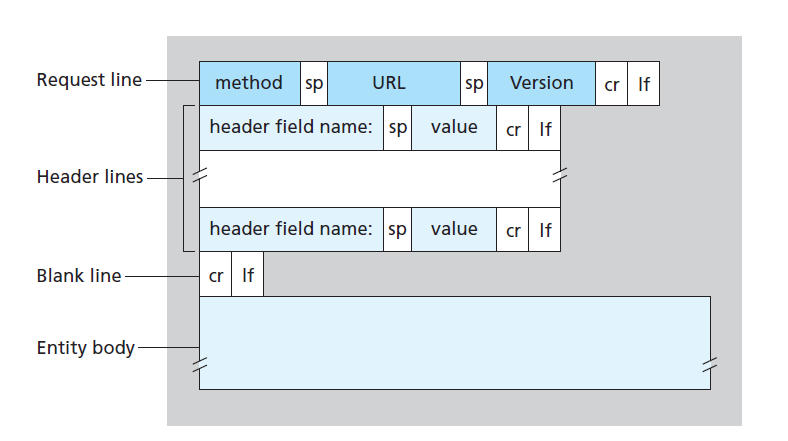
-The header lines

* Host: www.someschool.edu specifies the host on which the object resides.
* The Connection: close header line specifies 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
* 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 it will return; otherwise, the server sends its default version.

-The entity body is empty with the GET method, but is used with the POST method.

-An HTTP client often uses the POST method when the user fills out a form.



**Fig:** General format of an HTTP request message

-The HEAD method is similar to the GET method. When a server receives a request with the HEAD method, it responds with an HTTP message but it leaves out the requested object.

* Application developers often use the HEAD method for debugging.

-The PUT method allows a user to upload an object to a specific path (directory) on a specific Web server.

-The DELETE method allows a user, or an application, to delete an object on a Web server.

**HTTP Response Message**

-A typical HTTP response message:

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 response message has three sections:

* An initial **status line**
* Six **header lines**, and
* The **entity body**.

-The entity body is the message which contains the requested object itself (represented by data

data data data data ...).

-The status line has three fields:

* The protocol version field (HTTP/1.1)
* A status code (200)
* A corresponding status message (OK).

-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.

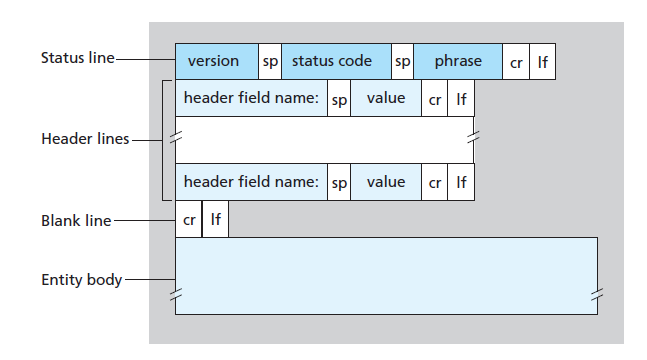
-The Last-Modified: header line indicates the time and date when the object was created or last modified.

-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.

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

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



**Fig:** General format of an HTTP response message

* **User-Server Interaction: Cookies**

-An HTTP server is stateless.

* However, if a web site wants 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.

-Cookies are defined in [RFC 6265], allow sites to keep track of users.

-Major commercial Web sites use cookies today.

-Cookie technology has four components:

* A cookie header line in the HTTP response message.
* A cookie header line in the HTTP request message.
* A cookie file kept on the user’s end system and managed by the user’s browser.
* A back-end database at the Web site.

-**Example:**

* Suppose a client visits Amazon.com for the first time.
* When the request comes into the Amazon Web server, the server creates a unique

identification number and creates an entry in its back-end database that is indexed by the identification number.

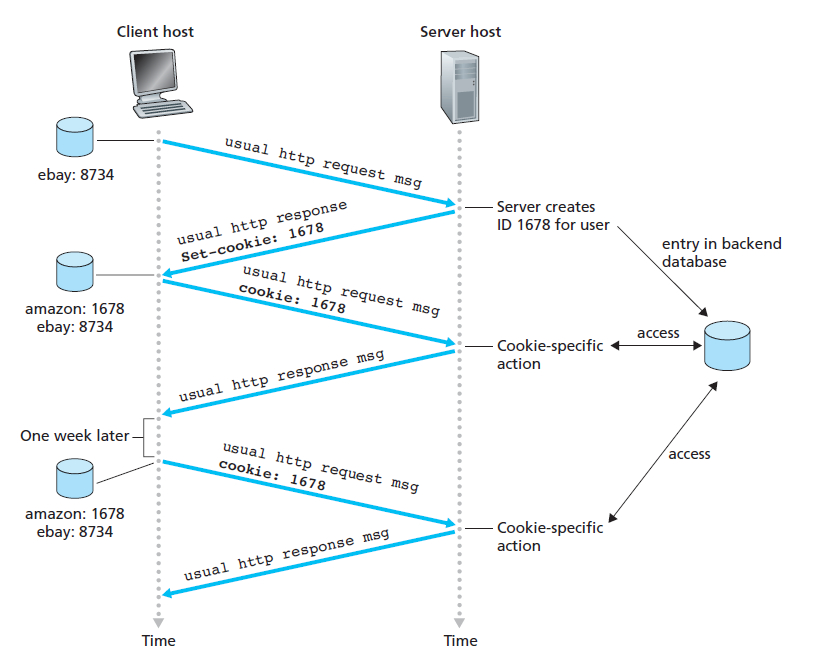
* The Amazon Web server then responds to client browser, including in the HTTP response a Set-cookie: header, which contains the identification number.
* When client browser receives the HTTP response message, it sees the Setcookie:

header.

* The browser then appends a line to the special cookie file that it manages.
* As client continues to browse the Amazon site, each time it requests a Web page, the browser consults the cookie file, extracts the identification number for this site, in the HTTP request.
* The Amazon server is able to track client’s activity at the Amazon site.
* Although the Amazon web site does not necessarily know client’s name, it

knows exactly which pages user visited, in which order, and at what times.

* If the client returns to Amazon’s site, even one week later, the browser will continue to put the header line (Cookie:1678) in the request messages.

****

**Fig:** Keeping user state with cookies

* **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.

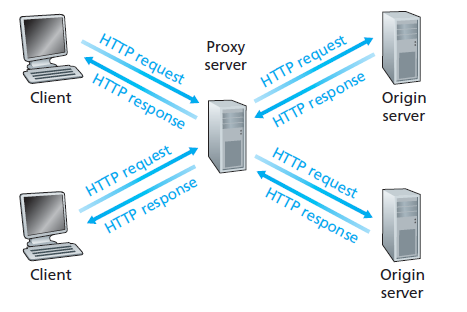
-The Web cache has its own disk storage and keeps copies of recently requested objects in this storage.

-Once a browser is configured, each browser request for an object is first directed to the Web cache and a copy is stored in the cache.

-**Example:**

* The browser establishes a TCP connection to the Web cache and sends an HTTP request for the object to the Web cache.
* 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.
* If the Web cache does not have the object, the Web cache opens a TCP connection to the origin server.
* The Web cache then sends an HTTP request for the object into the cache-to-server TCP connection.
* After receiving cache request, the origin server sends the object within an HTTP response to the Web cache.
* 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.

****

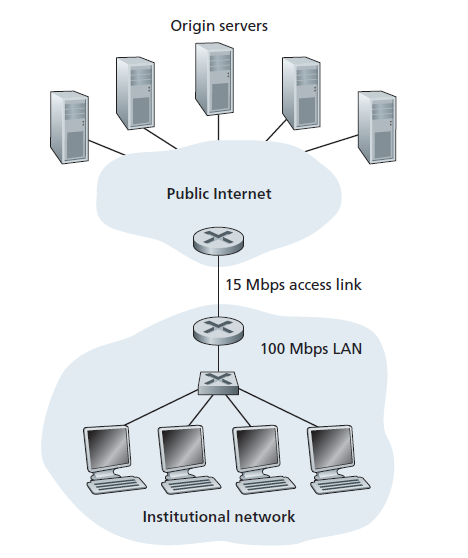
**Fig:** Clients requesting objects through a Web cache

-A cache is both a server and a client at the same time.

* When it receives requests from and sends responses to a browser, it is a server.
* When it sends requests to and receives responses from an origin server, it is a client.

-Web caching has seen deployment in the Internet for two reasons.

* First, a Web cache can substantially reduce the response time for a client.
* Second, as Web caches can substantially reduce traffic on an institution’s access link to the Internet.

****

**Fig:** Bottleneck between an institutional network and the Internet

-The institutional network is a high-speed LAN.

-A router in the institutional network and a router in the Internet are connected by a 15 Mbps link.

-The origin servers are attached to the Internet but are located all over the globe.

-Suppose the average object size is 1 Mbits and that the average request rate from the institution’s browsers to the origin servers is 15 requests per second.

-The total response time is, the time from the browser’s request of an object until its receipt of the object-is the sum of the LAN delay, the access delay (that is, the delay between the two routers), and the Internet delay.

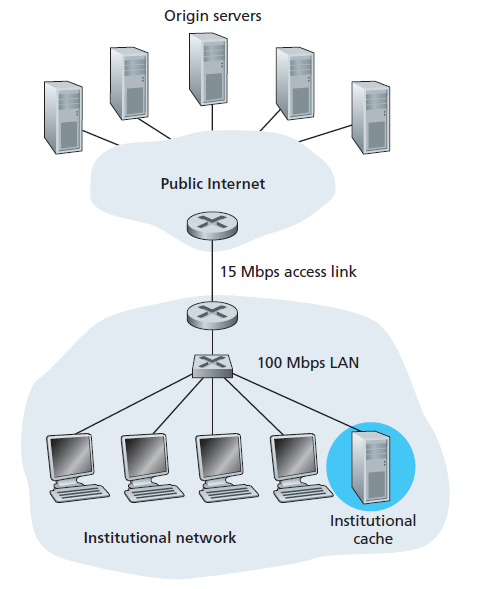
-The traffic intensity on the LAN is

(15 requests/sec) \_ (1 Mbits/request)/(100 Mbps) = 0.15

-The traffic intensity on the access link (from the Internet router to institution router) is

(15 requests/sec) \_ (1 Mbits/request)/(15 Mbps) = 1

-A traffic intensity of 0.15 on a LAN can be neglected and the traffic intensity approaches 1 as is the case of the access link the delay on a link becomes very large.

****

**Fig:** Adding a cache to the institutional network

-The solution is installing a Web cache in the institutional network.

-Through the use of **Content Distribution Networks (CDNs)**, Web caches are increasingly playing an important role in the Internet.

-A CDN company installs many geographically distributed caches throughout the Internet, thereby localizing much of the traffic.

* **The Conditional GET**

**-**Caching can reduce user perceived response times but it introduces a new problem

* The Web server may have been modified since the copy was cached at the client.

-To this, HTTP has a mechanism that allows a cache to verify that its objects are up to date.

* This mechanism is called the **conditional GET**.

-An HTTP request message so-called conditional GET message if

(1) The request message uses the GET method

(2) The request message includes an If-Modified-Since: header line.

-**Example:**

* First, on the behalf of a requesting browser, a proxy cache sends a request message to a Web server:

GET /fruit/kiwi.gif HTTP/1.1

Host: www.exotiquecuisine.com

* Second, the Web server sends a response message with the requested object to the

cache:

HTTP/1.1 200 OK

Date: Sat, 8 Oct 2011 15:39:29

Server: Apache/1.3.0 (Unix)

Last-Modified: Wed, 7 Sep 2011 09:23:24

Content-Type: image/gif

(data data data data data ...)

* The cache forwards the object to the requesting browser and also caches (maintains a copy) the object locally.
* This object may have been modified at the Web server in the week, the cache performs an up-to-date check by issuing a conditional GET to have data in the cache up-to-date.

GET /fruit/kiwi.gif HTTP/1.1

Host: www.exotiquecuisine.com

If-modified-since: Wed, 7 Sep 2011 09:23:24

-This conditional GET tells the server to send the object only if the object has been modified since the specified date.

* If the object has not been modified then, the Web server sends a response

message to the cache as the following with the entity body empty.

HTTP/1.1 304 Not Modified

Date: Sat, 15 Oct 2011 15:39:29

Server: Apache/1.3.0 (Unix)

(empty entity body)

**ELECTRONIC MAIL IN THE INTERNET**

-Electronic mail has been since the beginning of the Internet.

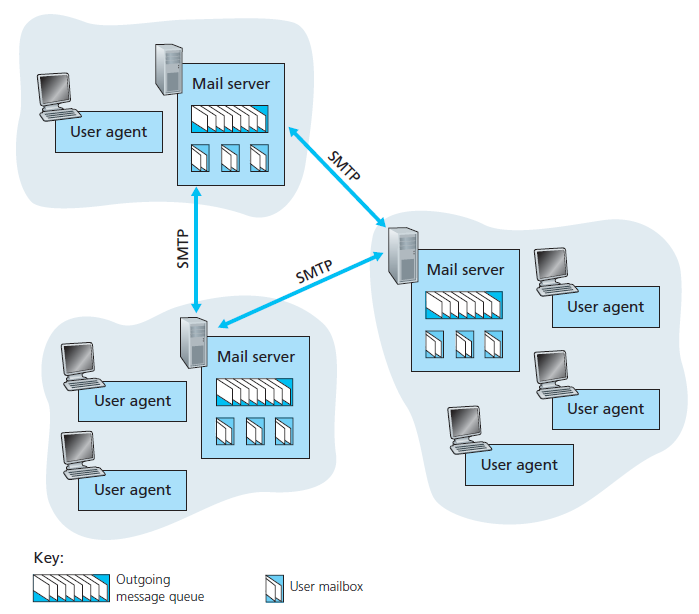
-It was the most popular application in the Internet.

* It remains one of the Internet’s most important and utilized applications.

-People send and read messages when it is convenient for them, without having to coordinate with other people’s schedules.

-Electronic mail is fast, easy to distribute, and inexpensive.

-Modern e-mail has many powerful features, including messages with attachments, hyperlinks, HTML-formatted text, and embedded photos.



**Fig:** A high-level view of the Internet e-mail system

-A high-level view of the Internet mail system; it has three major components:

* **User agents**
* **Mail servers**
* **Simple Mail Transfer Protocol (SMTP)**.

-User agents allow users to read, reply to, forward, save, and compose messages.

* Microsoft Outlook and Apple Mail are examples of user agents for e-mail.

-Mail servers form the core of the e-mail infrastructure which consists of outgoing message

Queue, **mailbox**.

-SMTP is the principal application-layer protocol for Internet electronic mail.

* It uses the reliable data transfer service of TCP to transfer mail from the sender’s mail

server to the recipient’s mail server.

* SMTP has two sides: a client side, which executes on the sender’s mail server, and a

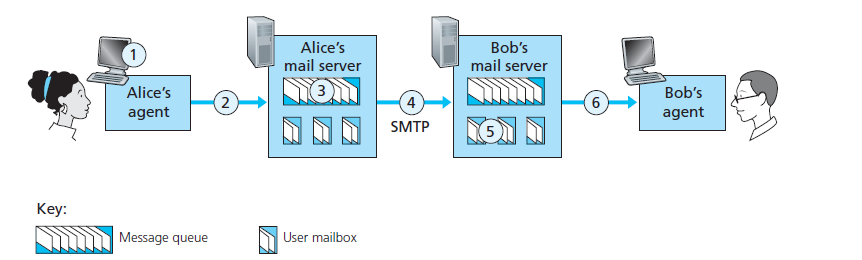
server side, which executes on the recipient’s mail server.

* Both the client and server sides of SMTP run on every mail server.
* **SMTP**

-SMTP defined in RFC 5321, is the heart of Internet electronic mail.

-SMTP transfers messages from senders’ mail servers to the recipients’ mail servers.

-Messages must be sent in 7-bit ASCII.



**Fig:** Alice sends a message to Bob

-The basic operation of SMTP, Suppose Alice wants to send Bob a simple ASCII message.

* + Alice invokes her user agent for e-mail, provides Bob’s e-mail address composes a message, and instructs the user agent to send the message.
  + Alice’s user agent sends the message to her mail server, and it is placed in a

message queue.

* The client side of SMTP, running on Alice’s mail server, sees the message in the message queue.
* It opens a TCP connection to an SMTP server, running on Bob’s mail server.
* After some initial SMTP handshaking, the SMTP client sends Alice’s message

into the TCP connection.

* At Bob’s mail server, the server side of SMTP receives the message.
* Bob’s mail server then places the message in Bob’s mailbox.
* Bob invokes his user agent to read the message at his convenience.

-SMTP does not normally use intermediate mail servers for sending mail, even when the two mail servers are located at opposite ends of the world.

A closer look at how SMTP transfers a message from a sending mail server to a receiving mail server. First, the client SMTP establishes a TCP connection to port 25 at the server SMTP (running on the receiving mail server host. During this SMTP handshaking phase, the SMTP client indicates the e-mail address of the sender (the person who generated the message) and the e-mail address of the recipient. Once the SMTP client and server have introduced themselves to each other, the client sends the message. SMTP can count on the reliable data transfer service of TCP to get the message to the server without errors. The client then repeats this process over the same TCP connection if it has other messages to send to the server; otherwise, it instructs TCP to close the connection.

* **Comparison with HTTP**

-HTTP transfers files from a Web server to a Web client.

-SMTP transfers files from one mail server to another mail server.

-HTTP is mainly a **pull protocol**-someone loads information on a Web server and users use HTTP to pull the information from the server at their convenience.

-SMTP is primarily a **push protocol**-the sending mail server pushes the file to the receiving mail server.

-SMTP requires each message, including the body of each message, to be in 7-bit ASCII format.

-HTTP data does not impose this restriction.

-HTTP encapsulates each object in its own HTTP response message.

-Internet mail places all of the message’s objects into one message.

* **Mail Message Formats**

-When an e-mail message is sent from one person to another, a header containing peripheral information precedes the body of the message itself.

-This peripheral information is contained in a series of header lines, which are defined in RFC 5322.

-The header lines and the body of the message are separated by a blank line.

-A typical message header looks like this:

From: alice@crepes.fr

To: bob@hamburger.edu

Subject: Searching for the meaning of life.

-After the message header, then the message body (in ASCII) follows.

* **Mail Access Protocols**

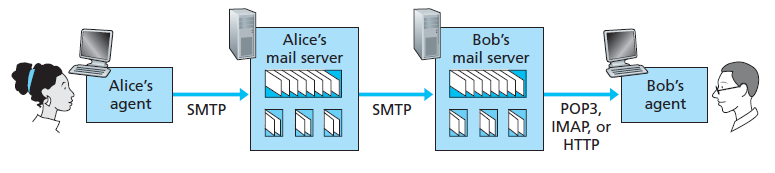
**-**SMTP has been designed for pushing e-mail from one host to another. But SMTP can’t be used to obtain the messages because obtaining the messages is a pull operation, whereas SMTP is a push protocol.

**-**There are currently a number of popular mail access protocols, including **Post Office**

**Protocol—Version 3 (POP3)**, **Internet Mail Access Protocol (IMAP)**, and HTTP.

SMTP is used to transfer mail from the sender’s mail server to the recipient’s mail

server; SMTP is also used to transfer mail from the sender’s user agent to the sender’s mail server. A mail access protocol, such as POP3, is used to transfer mail from the recipient’s mail server to the recipient’s user agent.



**Fig:** E-mail protocols and their communicating entities

**POP3**

-POP3 is an extremely simple mail access protocol.

-It is defined in [RFC 1939], which is short and quite readable.

-POP3 begins when the user agent opens a TCP connection to the mail server (the server) on port 110.

-With the TCP connection established, POP3 progresses through three phases:

* Authorization
* Transaction
* Update.

-During the first phase, authorization, the user agent sends a username and a password to authenticate the user.

* There are two possible responses: +OK used by the server to indicate that the previous

command was fine; and -ERR, used by the server to indicate that something was wrong with the previous command.

-During the second phase, transaction, the user agent retrieves messages; also, during this phase, the user agent can mark messages for deletion, remove deletion marks.

-The third phase, update, occurs after the client has issued the quit command, ending the POP3

session; at this time, the mail server deletes the messages that were marked for deletion.

-A user agent using POP3 can often be configured (by the user) to “download and delete” or to “download and keep.”

-In the download-and-delete mode, the user agent will issue the list, retr, and dele commands.

* In this download-and-delete mode the recipient cannot access his mail messages from multiple machines if want to access.

-In the download-and-keep mode, the user agent leaves the messages on the mail server after downloading them.

**IMAP**

-An IMAP server will associate each message with a folder

* When a message first arrives at the server, it is associated with the recipient’s INBOX folder.
* The recipient can then move the message into a new, user-created folder, read the message, delete the message, and so on.
* The IMAP protocol provides commands to allow users to create folders and move messages from one folder to another.
* IMAP also provides commands that allow users to search remote folders for messages matching specific criteria.

-Another important feature of IMAP is that it has commands that permit a user agent to obtain components of messages.

* This feature is useful when there is a low-bandwidth connection (for example, a slow-speed modem link) between the user agent and its mail server.

**Web-Based E-Mail**

-More and more users today are sending and accessing their e-mail through their Web

Browsers.

-The user communicates with its remote mailbox via HTTP rather than the POP3 or IMAP protocol.

**SOCKET PROGRAMMING: CREATING NETWORK APPLICATIONS**

-A typical network application consists of a pair of programs

* A client program and
* A server program which are residing on two different end systems.

-When these two programs are executed, a client process and a server process are created, and these processes communicate with each other by reading from, and writing to, sockets.

-The main task of developer is to write the code for both the client and server programs.

-There are two types of network applications.

* One type is an implementation whose operation is specified in a protocol standard, such as an RFC or some other standards document; such an application is sometimes referred to as “open”.
* The other type of network application is a proprietary network application.
* In this case the client and server programs employ an application-layer protocol that has not been openly published in an RFC or elsewhere.

-During the development phase, one of the first decisions that must be made is whether the application is to run over TCP or over UDP.

* **Socket Programming with UDP**

-The sending process can push a packet out the socket door, using UDP, it must first attach a destination address to the packet.

-After the packet passes through the sender’s socket, the Internet will use this destination address to route the packet through the Internet to the socket in the receiving process.

-As a host may be running many network applications processes, each with one or more sockets, it is also necessary to identify the particular socket in the destination host.

* When a socket is created, an identifier, called a **port number**, is assigned to it.

-Consider a simple client-server application to demonstrate socket programming for both UDP and TCP:

1. The client reads a line of characters (data) from its keyboard and sends the data

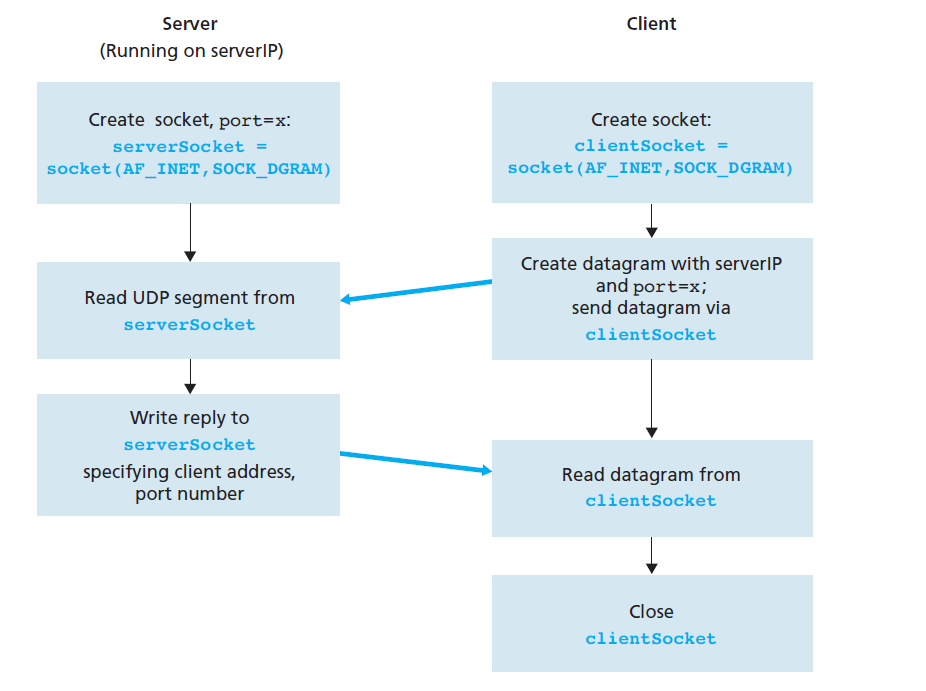
to the server.

2. The server receives the data and converts the characters to uppercase.

3. The server sends the modified data to the client.

4. The client receives the modified data and displays the line on its screen.

-The client program is called UDPClient.py, and the server program is called UDPServer.py.



**Fig:** The client-server application using UDP

**UDPClient.py**

Code for the client side of the application:

from socket import \*

serverName = ‘hostname’

serverPort = 12000

clientSocket = socket(socket.AF\_INET, socket.SOCK\_DGRAM)

message = raw\_input(’Input lowercase sentence:’)

clientSocket.sendto(message,(serverName, serverPort))

modifiedMessage, serverAddress = clientSocket.recvfrom(2048)

print modifiedMessage

clientSocket.close()

**UDPServer.py**

Code for the server side of the application:

from socket import \*

serverPort = 12000

serverSocket = socket(AF\_INET, SOCK\_DGRAM)

serverSocket.bind((*’’*, serverPort))

print *”The server is ready to receive”*

while 1:

message, clientAddress = serverSocket.recvfrom(2048)

modifiedMessage = message.upper()

serverSocket.sendto(modifiedMessage, clientAddress)

* **Socket Programming with TCP**

-TCP is a connection-oriented protocol which means that before the client and server can start to send data to each other, they first need to handshake and establish a TCP connection.

-The client creates its TCP socket and specifies the address of the welcoming socket in

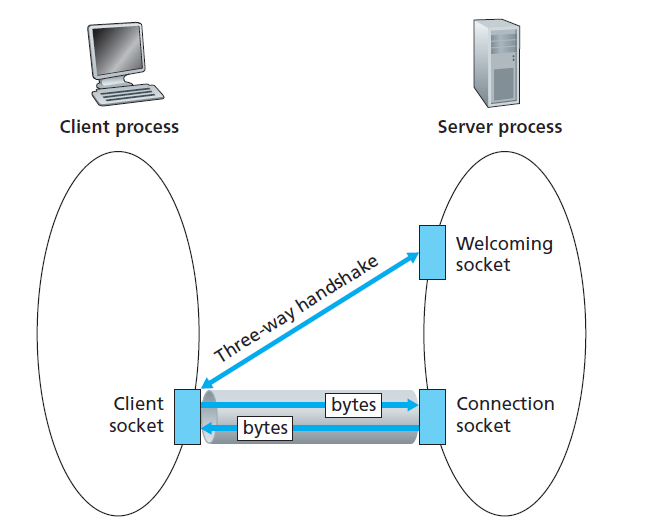
the server, namely, the IP address of the server host and the port number of the socket.

-Then the client initiates a three-way handshake and establishes a TCP connection with the server.

-During the three-way handshake, the server creates a **new**socket that is dedicated to that particular client.

-The client process can send bytes into its socket, and TCP guarantees that the server process will receive each byte in the order sent.

-TCP thus provides a reliable service between the client and server processes.



**Fig:** The TCP Server process has two sockets

**TCPClient.py**

Code for the client side of the application:

serverName = ’servername’

serverPort = 12000

clientSocket = socket(AF\_INET, SOCK\_STREAM)

clientSocket.connect((serverName,serverPort))

sentence = raw\_input(‘Input lowercase sentence:’)

clientSocket.send(sentence)

modifiedSentence = clientSocket.recv(1024)

print ‘From Server:’, modifiedSentence

clientSocket.close()

**TCPServer.py**

Code for the server side of the application:

from socket import \*

serverPort = 12000

serverSocket = socket(AF\_INET,SOCK\_STREAM)

serverSocket.bind((‘’,serverPort))

serverSocket.listen(1)

print ‘The server is ready to receive’

while 1:

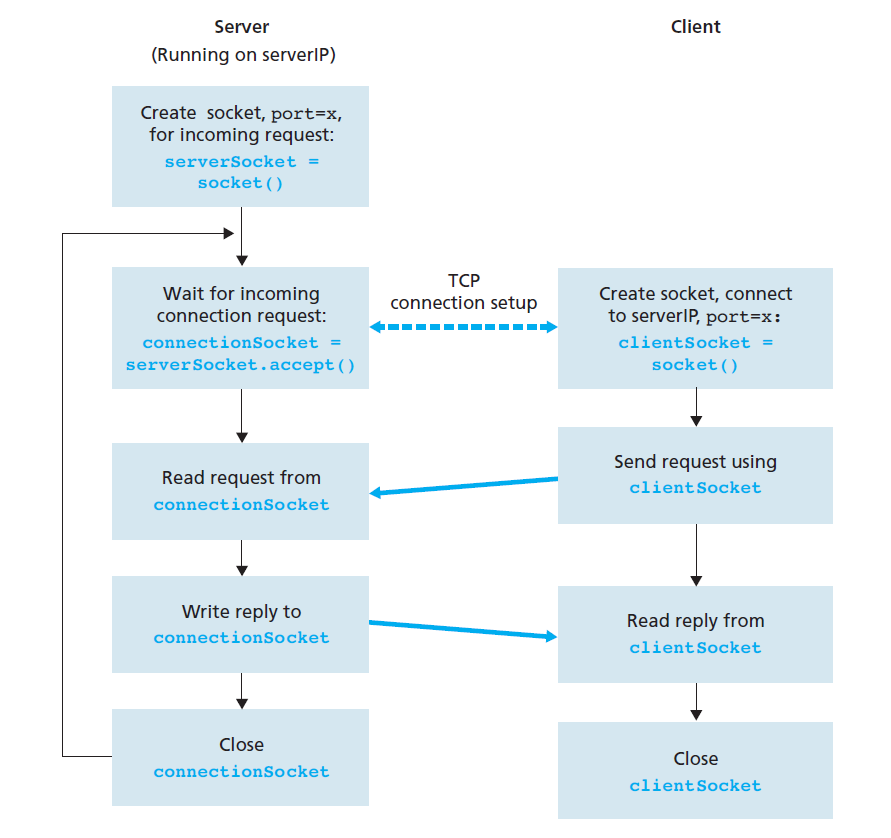
connectionSocket, addr = serverSocket.accept()

sentence = connectionSocket.recv(1024)

capitalizedSentence = sentence.upper()

connectionSocket.send(capitalizedSentence)

connectionSocket.close()



**Fig:** The client-server application using TCP