CHAPTER NO 2

APPLICATION LAYER

Network Application Architectures

From the application developer’s perspective, 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**. In choosing the application architecture, an application developer will likely draw on one of the two predominant architectural paradigms used in modern network applications: the **client-server architecture** or the **peer-to-peer (P2P)** architecture.

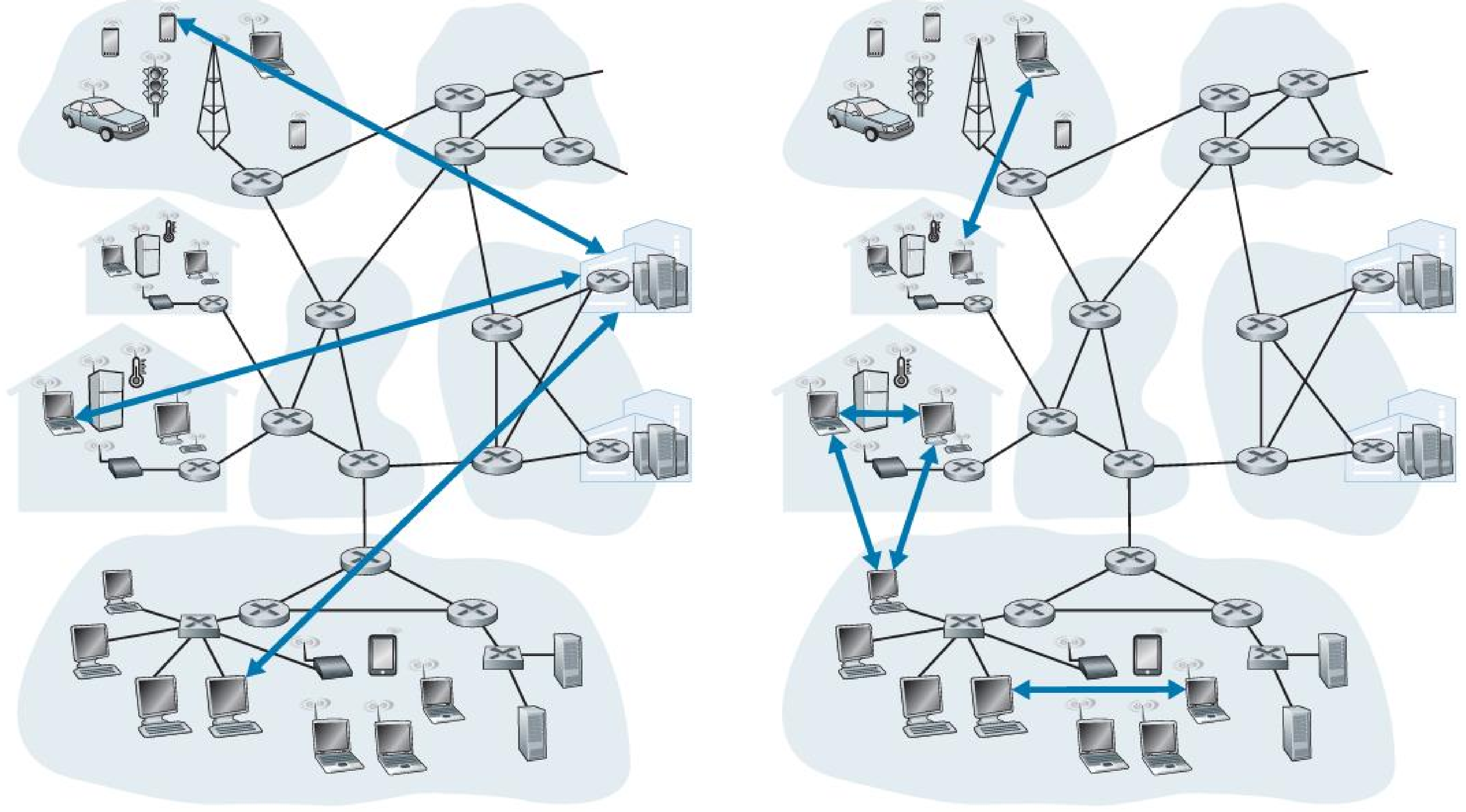
CLIENT-SERVER ARCHITECTURE:

- In a client-server architecture, there is an always-on host, called the server, which services 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.
- clients do not directly communicate with each other .for example, in the Web application, two browsers do not directly communicate.
- Client-server architecture is that the server has a fixed, well-known address, called an IP address. Because the server has a fixed, well-known address, and because the server 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 a client-server architecture include the Web, FTP, Telnet, and e-mail.
- a single-server host is incapable of keeping up with all the requests from clients. For example, a popular social-networking site can quickly become overwhelmed if it has only one server handling all of its requests. For this reason, a data center, housing a large number of hosts, is often used to create a powerful virtual server. The most popular Internet services—such as search engines (e.g., Google, Bing, Baidu), Internet commerce (e.g., Amazon, eBay, Alibaba), Web-based e-mail (e.g., Gmail and Yahoo Mail), social media (e.g., Facebook, Instagram, Twitter, and WeChat)—run in one or more data centers.

P2P architecture:

In a P2P architecture, there is minimal (or no) reliance on dedicated servers in data centers. Instead the application exploits 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.

An example of a popular P2P application is the file-sharing application BitTorrent.One of the most compelling features of P2P architectures is their self- scalability. For example, in a P2P file-sharing application, although each peer generates workload by requesting files, each peer also adds service capacity to the system by distributing files to other peers. P2P architectures are also cost effective, since they normally don’t require significant server infrastructure and server bandwidth (in contrast with clients-server designs with datacenters). However, P2P applications face challenges of security, performance, and reliability due to their highly decentralized structure.



a. Client-server architecture

b. Peer-to-peer architecture

Figure 2.2 ♦ (a) Client-server architecture; (b) P2P architecture

Processes Communicating

How 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 **interprocess communication**, using rules that are governed by the end system's operating system. But in this book, we are not particularly interested in how processes in the same host communicate, but instead in how processes running on different hosts (with potentially different operating systems) communicate. Processes on two different end systems communicate with each other by exchanging messages across the computer network. A sending process creates and sends messages into the network; a receiving process receives these messages and possibly responds by sending messages back.

Client and Server Processes

A network application consists of **pairs of processes** that send messages to each other over a network. For example, in the Web application a client browser process exchanges messages with a Web server process. In a P2P file-sharing system, a file is transferred from a process in one peer to a process in another peer. For each pair of communicating processes, **we typically label one of the two processes as the client and the other process as the server.** With the Web, a browser is a client process and a Web server is a server process. With P2P file sharing, the peer that is downloading the file is labeled as the client, and the peer that is uploading the file is labeled as the server.You may have observed that in some applications, such as in P2P file sharing, a process can be both a client and a server. Indeed, a process in a P2P file-sharing system can both upload and download files.

In the context of a communication session between a pair of processes, the process that initiates the communication (that is, initially contacts the other process at the beginning of the session) is labeled as the client. The process that waits to be contacted to begin the session is the server.

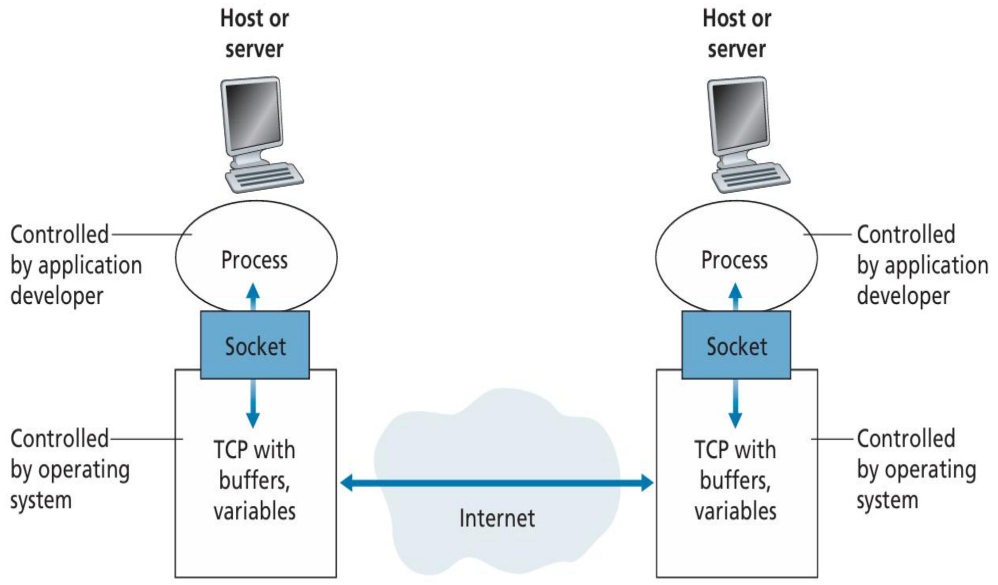
The Interface Between the Process and the Computer Network

Any message sent from one process to another must go through the underlying network. A process sends messages into, and receives messages from, the network through a software interface called a **socket.**

ANALOGY: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 shoves 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.

API (APPLICATION PROGRAMMING INTERFACE)

a 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, since the socket is the programming interface with which network applications are built. 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.The only control that the application developer has on the transport layer side is (1) the choice of transport protocol and (2) perhaps the ability to fix a few transport-layer parameters such as maximum buffer and maximum segment sizes


Host or server
Host or server
Controlled by application developer
Process
Process
Controlled by application developer
Socket
Socket
Controlled by operating system
TCP with buffers, variables
Internet
TCP with buffers, variables
Controlled by operating system

Addressing Processes

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

(1)the address of the host (IP address)

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

(Port Number)

In the Internet, the host is identified by its IP address. We'll discuss IP addresses n great detail in Chapter 4. For now, all we need to know is that an IP address is a 32-bit quantity that we can think of as uniquely identifying the host. In addition to knowing the address of the host to which a message is destined, the sending process must also identify the receiving process (more specifically, the receiving socket) running in the host. This information is needed because in general a host could be running many network applications. 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.

2.1.3 Transport Services Available to Applications

What are the services that a transport-layer protocol can offer to applications invoking it? We can broadly classify the possible services along four dimensions: **reliable data transfer, throughput, timing, and security.**

RELIABLE DATA TRANSFER: Packets can get lost within a computer network. For example, a packet can overflow a buffer in a router, or can be discarded by a host or router after having some of its bits corrupted. For many applications—such as electronic mail, file transfer, remote host access, Web document transfers, and financial applications—data loss can have devastating consequences Thus, to support these applications, something has to be done to guarantee 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.**When a transport-layer protocol doesn't provide reliable data transfer, some of the data sent by the sending process may never arrive at the receiving process. This may be acceptable for **loss-tolerant applications,** most notably multimedia applications such as conversational audio/video that can tolerate some amount of data loss.

Throughput:

the rate at which the sending process can deliver bits to the receiving process. Because other sessions will be sharing the bandwidth along the network path, and because these other sessions will be coming and going, the available throughput can fluctuate with time. These observations lead to another natural service that a transport-layer protocol could provide, namely, guaranteed available throughput at some specified rate. With such a service, the application could request a guaranteed throughput of r bits/sec, and the transport protocol would then ensure that the available throughput is always at least r bits/sec. Such a guaranteed throughput service would appeal to many applications.Applications that have throughput requirements are said to be bandwidth-sensitive applications. Many current multimedia applications are bandwidth sensitive.While bandwidth-sensitive applications have specific throughput requirements, **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. Of course, the more throughput, the better.

TIMING:A transport-layer protocol can also provide timing guarantees. As with throughput guarantees, timing guarantees can come in many shapes and forms. An example guarantee might be that every bit that the sender pumps into the socket arrives at the receiver's socket no more than 100 msec later. Such a service would be appealing to interactive real-time applications, such as Internet telephony, virtual environments, teleconferencing, and multiplayer games, all of which require tight timing constraints on data delivery in order to be effectiveong delays in Internet telephony, for example, tend to result in unnatural pauses in the conversation; in a multiplayer game or virtual interactive environment, a long delay between taking an action and seeing the response from The environment (for example, from another player at the end of an end-to-end connection) makes the application feel less realistic.

•

Security

Finally, a transport protocol can provide an application with one or more security services. For example, 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. Such a service would provide confidentiality between the two processes, even if the data is somehow observed between sending and receiving processes.

Application	Data Loss	Throughput	Time-Sensitive
File transfer/download	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Internet telephony/ Video conferencing	Loss-tolerant	Audio: few kbps–1 Mbps Video: 10 kbps–5 Mbps	Yes: 100s of msec
Streaming stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of msec
Smartphone messaging	No loss	Elastic	Yes and no

Figure 2.4 ♦ Requirements of selected network applications

2.1.4 Transport Services Provided by the Internet

TCP Services The TCP service model includes a connection-oriented service and a reliable data transfer service. When an application invokes TCP as its transport protocol, the application receives both of these services from TCP.

Connection-oriented service. TCP has the client and server exchange transportlayer control information with each other before the application-level messages begin to flow. This so-called handshaking procedure alerts the client and server, allowing them to prepare for an onslaught of packets. 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 that the two processes can send messages to each other over the connection 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. When one side of the application passes a stream of bytes into a socket, it can count on TCP to deliver the same stream of bytes to the receiving socket, with no missing or duplicate bytes.

TCP also includes a **congestion-control mechanism,** a service for the general welfare of the Internet rather than for the direct benefit of the communicating processes. The TCP congestion-control mechanism throttles a sending process (client or server) 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. UDP 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. Furthermore, messages that do arrive at the receiving process may arrive out of order.UDP does not include a congestion-control mechanism, so the sending side of UDP can pump data into the layer below (the network layer) at any rate it pleases.

in our brief description of TCP and UDP, conspicuously missing was any mention of throughput or timing guarantees— services not provided by today’s Internet transport protocols. **Does this mean that timesensitive applications such as Internet telephony cannot run in today’s Internet?** The answer is clearly **no**—the Internet has been hosting time-sensitive applications for many years. These applications often work fairly well because they have been designed to cope, to the greatest extent possible, with this lack of guarantee. Nevertheless, clever design has its limitations when delay is excessive, or the end-to-end throughput is Limited.

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 5321]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP 1.1 [RFC 7230]	TCP
File transfer	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (e.g., YouTube), DASH	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	UDP or TCP

2.1.5 Application-Layer Protocols

. An application-layer protocol defines 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

2.2 The Web and HTTP

The HyperText Transfer Protocol (HTTP), the Web's application-layer protocol, is at the heart of the Web. It is defined in [RFC 1945], [RFC 7230] and [RFC 7540]. 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.

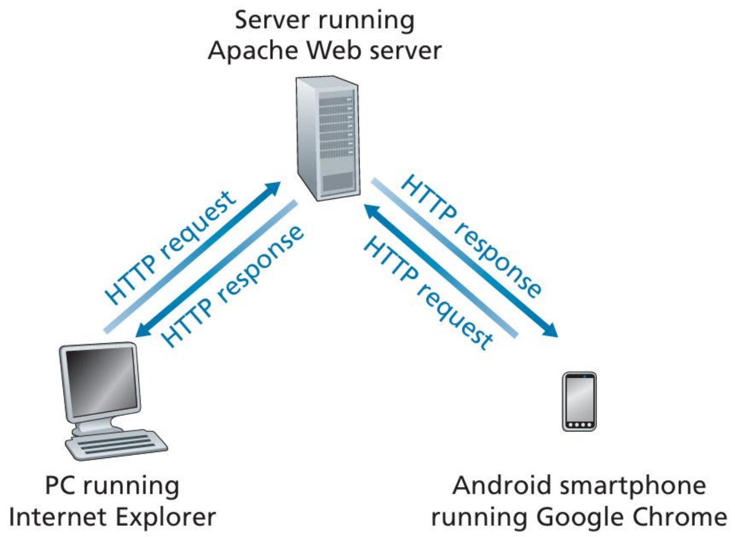
Web terminology.

A Web page (also called a document) consists of objects. An object is simply a file—such as an HTML file, a JPEG image, a Javascrpt file, a CCS style sheet file, or a video clip—that is addressable by a single URL. Most Web pages consist of a base HTML file and several referenced objects. For example, if a Web page contains HTML text and five JPEG images, then the Web page has Six objects: the base HTML file plus the five images.

```
http://www.someSchool.edu/someDepartment/picture.gif
```

has www.someSchool.edu for a hostname and /someDepartment/picture.
gif for a path name. Because Web browsers (such as Internet Explorer and Chrome) implement the client side of HTTP, in the context of the Web, we will use the words browser and client interchangeably. Web servers, which implement the server side of HTTP, house Web objects, each addressable by a URL. Popular Web servers include Apache and Microsoft Internet Information Server.

HTTP uses **TCP** as its underlying transport protocol (rather than running on top of UDP). 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., on the client side the socket inter
face is the door between the client process and the TCP connection; on the server side it is the door between the server process and the TCP connection. The client sends HTTP request messages into its socket interface and receives HTTP response messages from its socket interface. Because an HTTP server maintains
no information about the clients, HTTP is said to be **a stateless protocol.**



2.2.2 Non-Persistent and Persistent Connections

should each request/response pair be sent over a separate TCP connection, or should all of the requests and their corresponding responses be sent over the same TCP connection? In the former approach, the application is said to use non-persistentconnections; and in the latter approach, persistent connections.

HTTP can use both non-persistent connections and persistent connections. Although HTTP uses persistent connections in its default mode

round-trip time

(RTT), which 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, packetqueuing delays in intermediate routers and switches, and packet-processing delays.

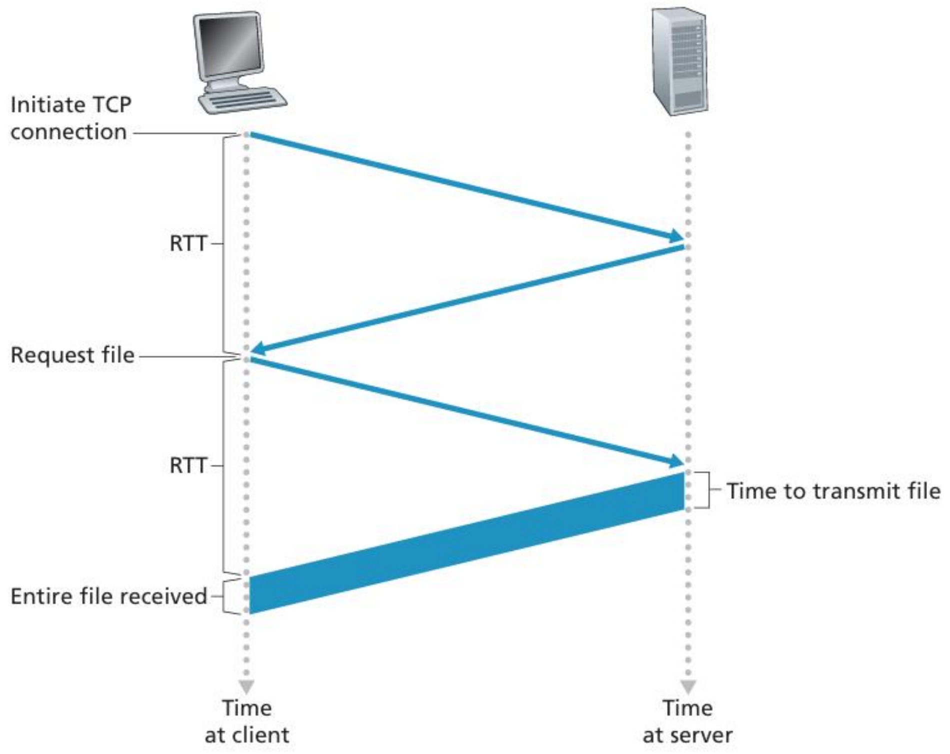


Figure 2.7 ♦ Back-of-the-envelope calculation for the time needed to request and receive an HTML file

HTTP WITH PERSISTENT CONNECTIONS:

With HTTP/1.1 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 (in the example above, the base HTML file and the 10 images) can be sent over a single persistent TCP connection. Typically, the HTTP server closes a connection when it isn’t used for a certain time (a configurable timeout interval). When the server receives the back-to-back requests, it sends the objects back-toback. The default mode of HTTP uses persistent connections with pipelining.

HTTP Request Message

Below we provide a typical HTTP request message:

- GET /somedir/page.html HTTP/1.1
- Host: www.someschool.edu
- Connection: close
- User-agent: Mozilla/5.0
- Accept-language: fr

HTTP RESPONSE MESSAGE:

Below we provide a typical HTTP response message. This response message could be the response to the example request message just discussed.

- HTTP/1.1 200 OK
- Connection: close
- Date: Tue, 18 Aug 2015 15:44:04 GMT
- Server: Apache/2.2.3 (CentOS)
- Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
- Content-Length: 6821
- Content-Type: text/html
- (data data data data data ...)

Let's take a careful look at this response message. It has three sections: an initial status line, six header lines, and then the entity body.

200 OK: Request succeeded and the information is returned in the response.

301 Moved Permanently: Requested object has been permanently moved

400 Bad Request: This is a generic error code indicating that the request could not be understood by the server.

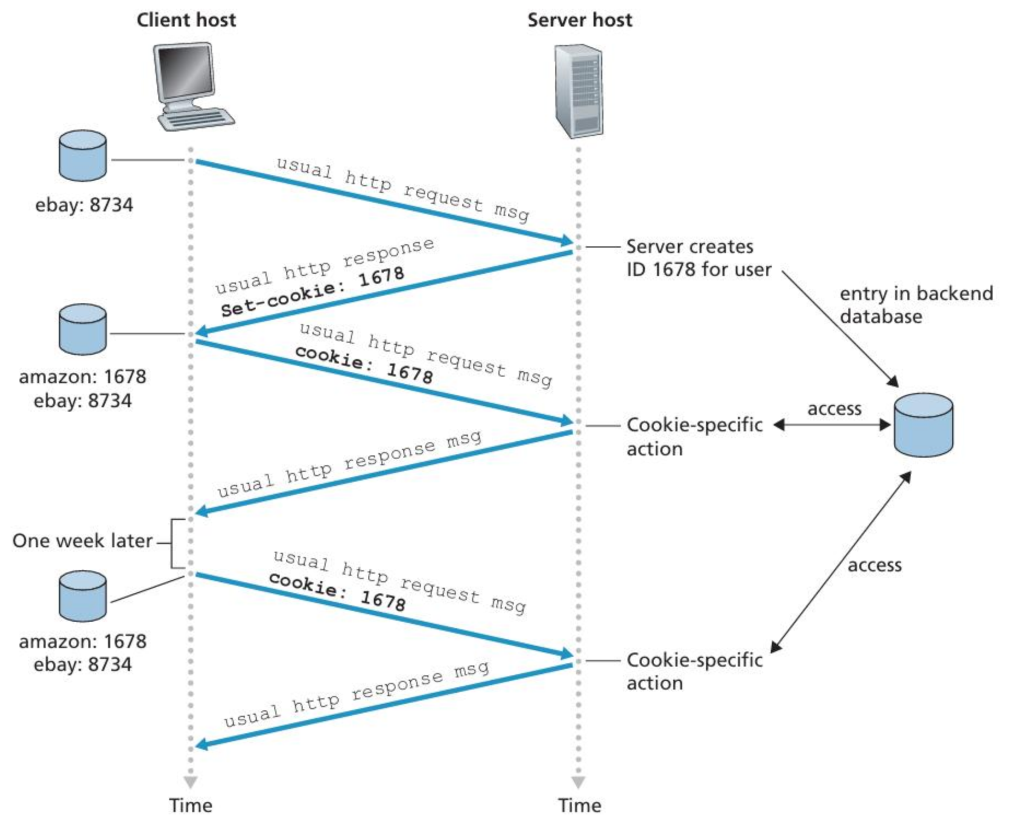
404 Not Found: The requested document does not exist on this server.

505 HTTP Version Not Supported: The requested HTTP protocol ver

sion is not supported by the server.

COOKIES:

HTTP uses cookies. Cookies, defined in [RFC 6265], allow sites to keep track of users. Most major commercial Web sites use cookies today.cookie technology has four components: (1) a cookie header line in the HTTP response message; (2) a cookie header line in the HTTP Request message; (3) a cookie file kept on the user's end system and managed by the user's browser; and (4) a back-end database at the Web site.



WEB-CACHING

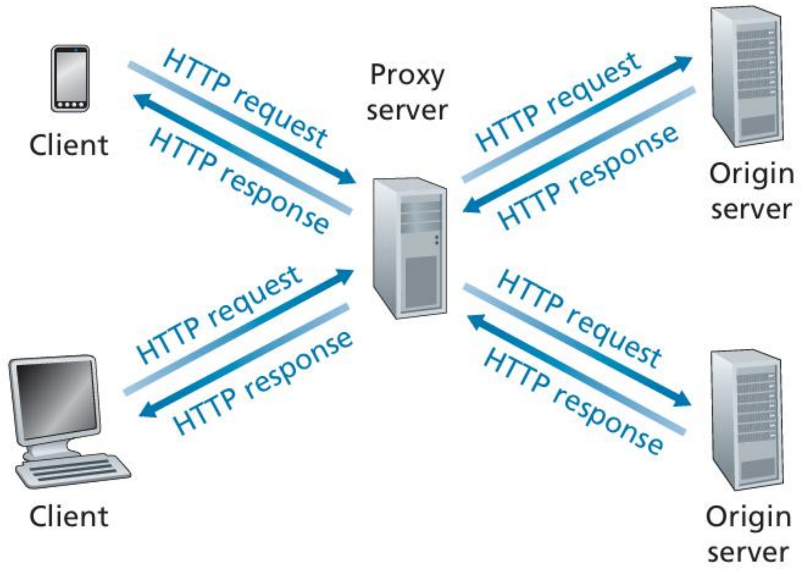
A Web cache—also called **a proxy server**—is a network entity that satisfies HTTP requests on the behalf of an origin Web server. The Web cache has its own disk storage and keeps copies of recently requested objects in this storage.

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. After receiving this request, the origin server sends the object within an HTTP response to the Web cache.

4. 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).



BENEFITS OF WEB CACHING:

First, a Web cache can substantially reduce the response time for a client request, particularly if the bottleneck bandwidth between the client and the origin server is much less than the bottleneck bandwidth between the client and the cache. If there is a high-speed connection between the client and the cache, as there often is, and if the cache has the requested object, then the cache will be able to deliver the object rapidly to the client. Second, as we will soon illustrate with an example, Web caches can substantially reduce traffic on an institution’s access link to the Internet. By reducing traffic, the institution (for example, a company or a university) does not have to upgrade bandwidth as quickly, thereby reducing costs.

THE CONDITIONAL GET

The Conditional GET

Although caching can reduce user-perceived response times, it introduces a new problem—the copy of an object residing in the cache may be stale. In other words, the object housed in the Web server may have been modified since the copy was cached at the client. Fortunately, HTTP has a mechanism that allows a cache to verify that its objects are up to date. This mechanism is called the conditional GET [RFC 7232]. An HTTP request message is a so-called conditional GET message if (1) the request message uses the GET method and (2) the request message includes an **If-Modified-Since: header line.**

GET /fruit/kiwi.gif HTTP/1.1

Host: www.exotiquecuisine.com

If-modified-since: Wed, 9 Sep 2015 09:23:24

HTTP/2

The primary goals for HTTP/2 are to reduce perceived latency by enabling request and response multiplexing over a single TCP connection, provide request prioritization and server push, and provide efficient compression of HTTP header fields. HTTP/2 does not change HTTP methods, status codes, URLs, or header fields. Instead, HTTP/2 changes how the data is formatted and transported between the client and server.

But developers of Web browsers quickly discovered that sending all the objects in a Web page over a single TCP connection has a **Head of Line (HOL) blocking problem. (LARGE VIDEO FILE).**

One of the primary goals of HTTP/2 is to get rid of (or at least reduce the number of) parallel TCP connections for transporting a single Web page. This not only reduces the number of sockets that need to be open and maintained at servers, but also allows TCP congestion control to operate as intended. But with only one TCP connection to transport a Web page, HTTP/2 requires carefully designed mechanisms to avoid HOL blocking

HTTP/2 FRAMING:

The ability to break down an HTTP message into independent frames, interleave them, and then reassemble them on the other end is the single most important enhancement of HTTP/2. The framing is done by the framing sub-layer of the HTTP/2 protocol. When a server wants to send an HTTP response, the response is processed by the framing sub-layer, where it is broken down into frames. The header field of the response becomes one frame, and the body of the message is broken down into one for more additional frames.The frames of the response are then interleaved by the framing sub-layer in the server with the frames of other responses and sent over the single persistent TCP connection. As the frames arrive at the client, they are first reassembled into the original response messages at the Framing sub-layer and then processed by the browser as usual. Similarly, a client's HTTP requests are broken into frames and interleaved.

Another feature of HTTP/2 is the ability for a server to send multiple responses for a single client request. That is, in addition to the response to the original request, the server can push additional objects to the client, without the client having to request each one. This is possible since the HTML base page indicates the objects that will be needed to fully render the Web page.