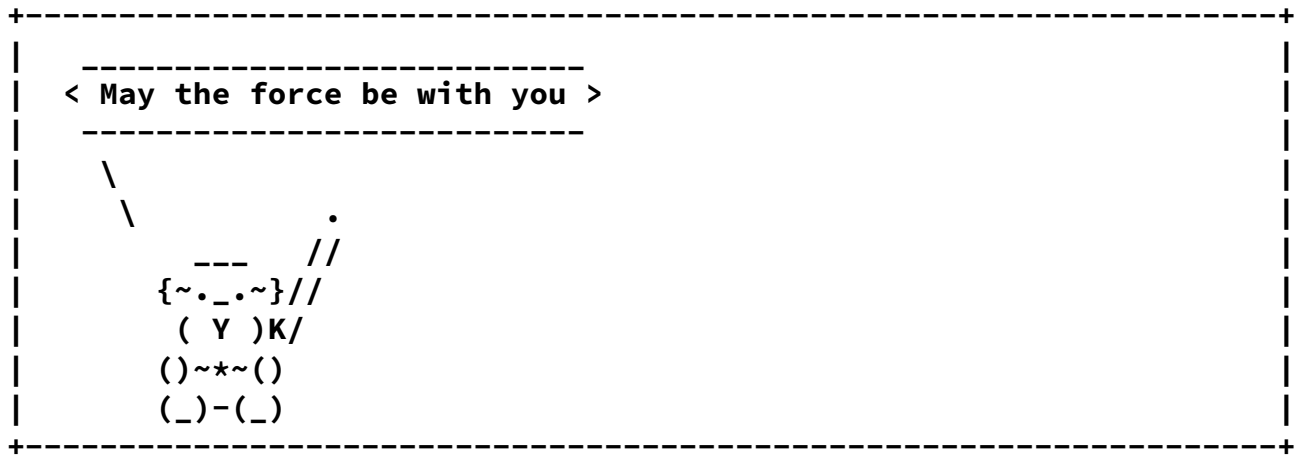

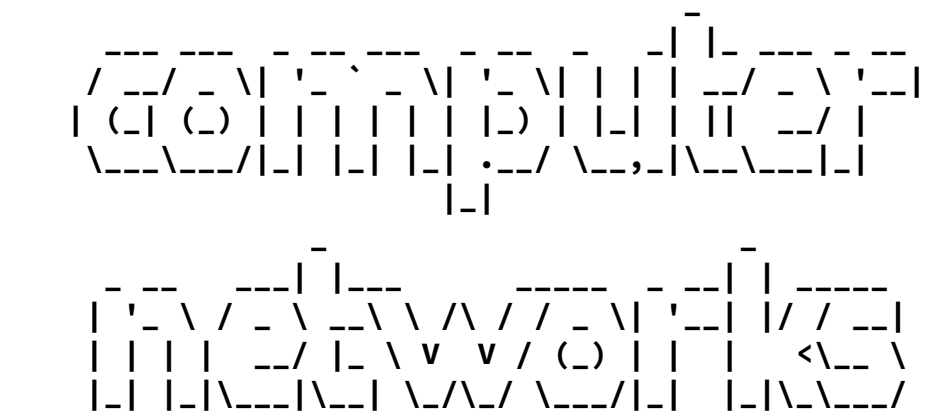

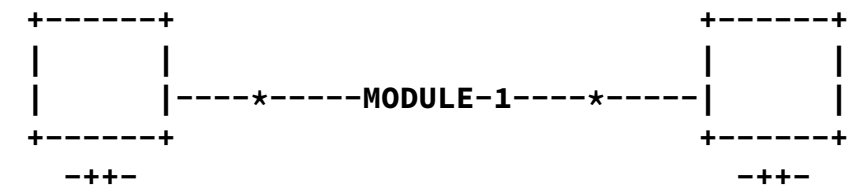

SOHAN JAIN

Computer Networks

Module-1

Sohan Jain

February 17, 2022

Contents

List of Figures

Part I

Question Papers

18CS52

Jan/Feb-2021

1. (a) What are the different transport services available to application? Explain. **(07)**

   (b) Explain HTTP request and response message format. **(08)**

   (c) Write a note on FTP and discuss about FTP command and replies. **(05)**

OR

2. (a) What are the steps involved between client and server in order to fetch 10 JPEG images. Which are residing in the same server by using non–persistent HTTP connection. The URL for base HTML file is `http://www.xyz.edu/department/base.index`. **(07)**

   (b) With a neat diagram explain how DNS server will interact to various DNS server hierarchically. **(05)**

   (c) Illustrate how *user1* can send mail to *user2*, and how *user2* receives the mail by using SMTP. **(08)**

17CS52

Aug/Sept-2020

1. (a) Compare client-server and peer-to-peer architecture. **(06)**

   (b) What are the different types of transport services provided by the internet? **(08)**

   (c) With a general format, explain the HTTP Request and HTTP Response messages. **(06)**

OR

2. (a) Explain FTP commands and replies. **(08)**

   (b) What are the services provided by DNS? **(04)**

   (c) Write a short note on: **(08)**

      i. Web caching

      ii. SMTP

17CS52

Dec/Jan-2020

1. (a) Which protocol can be used for fetching web pages? Explain its working with request and response message formats. **(10)**

   (b) Explain the services offered by DNS and also explain the DNS record and message format. **(10)**

OR

2. (a) Explain the working FTP along with its commands. **(08)**

   (b) Compare HTTP and SMTP. **(04)**

   (c) Illustrate how P2P architecture can be adopted in file sharing application like bit torrent. **(08)**

Part II

Question Bank

Chapter 1

Application Layer

1.1 Transport Services Available to Applications

Jan/Feb-2021 - 7M

We can broadly classify the possible services along four dimensions: Reliable data transfer, throughput, timing and security.

1. **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. When a transport protocol provides this service, the sending process can just pass its data into the socket and know with complete confidence that the data will arrive without errors at the receiving process.

2. **Throughput:**[1] 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.

3. **Timing:** 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.

4. **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.

[1] Throughput is the name given to the amount of data that can be sent and received within a specific timeframe.

1.2 Transport Services Provided by the Internet

Aug/Sept-2020 - 8M

The Internet makes two transport protocols available to applications, UDP and TCP. Each of these protocols offers a different set of services to the invoking applications.

Application	Data Loss	Throughput	Time-Sensitivity
File transfer	No loss	Elastic	No
Email	No loss	Elastic	No
Streaming stored audio/video	Loss-tolerant	Audio: few kbps-1Mbps Video: 10kbps-5Mbps	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps-10kbps	Yes: 100s of msec

Figure 1.1: Requirements of selected network applications

TCP Services 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 transport-layer control information with each other before the application-level messages begin to flow. This so-called handshaking procedure. After the handshaking phase, a TCP connection is said to exist between the sockets of the two processes. When the application finishes sending messages, it must tear down the connection.
- Reliable data transfer service: 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.

UDP 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.

Application	Application-layer Protocol	Transport Protocol
Electronic mail	SMTP	TCP
Web	HTTP	TCP
Internet Telephony	SIP, RTP, or proprietary	UDP or TCP
File Transfer	FTP	TCP

Figure 1.2: Popular Internet applications, their application-layer protocols, and their underlying transport protocols

1.3 HTTP Message Format

[Jan/Feb-2021] [Aug/Sept-2020] [Dec/Jan-2020]

1.3.1 HTTP Request Message

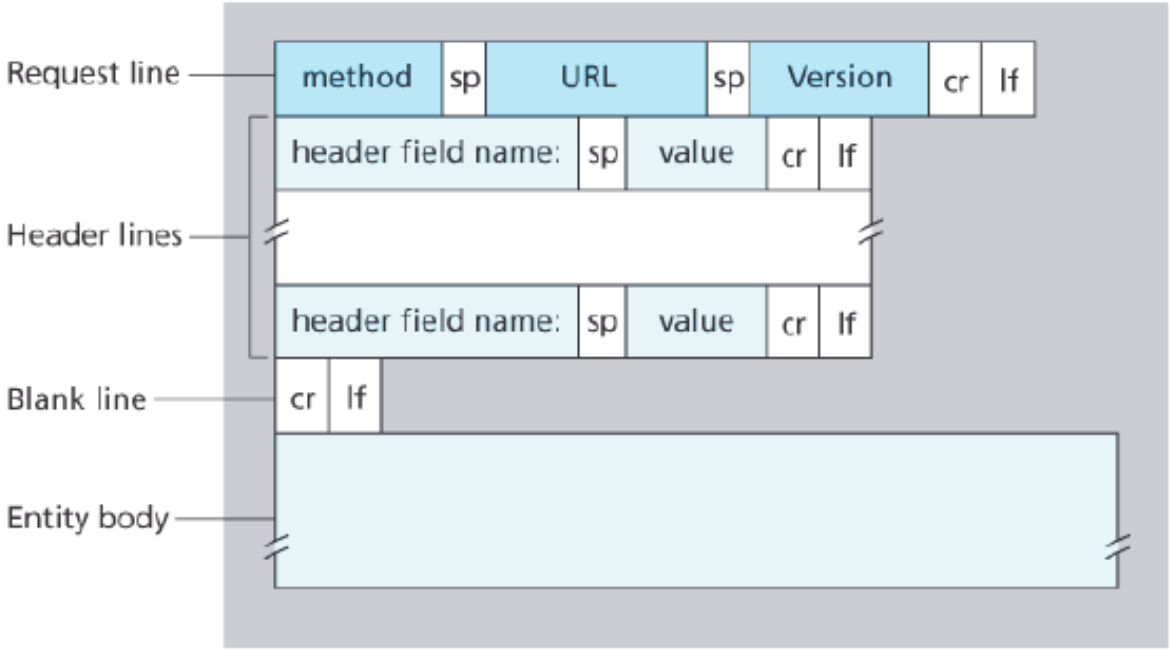

Figure 1.3: General format of an HTTP request message

Where sp – space, cr – carriage return and lf – line feed.
A typical HTTP request format:

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

Method:

- GET: The GET method is used when the browser requests an object, with the requested object identified in the URL field.
- POST: An HTTP client often uses the POST method when the user fills out a form—for example, when a user provides search words to a search engine.
- PUT: The PUT method is used by applications that need to upload objects to Web servers.
- HEAD: Used to retrieve header information.
- 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

1.3.2 HTTP Response Message

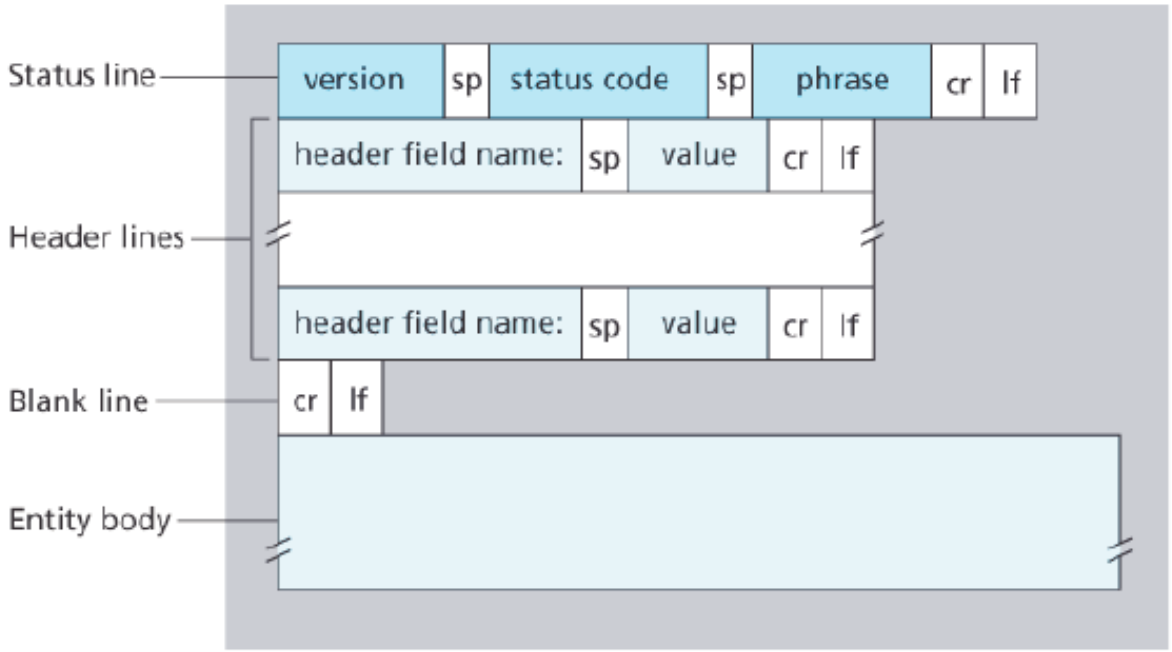

Figure 1.4: General format of an HTTP response message

A typical HTTP response message:

```
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 ...)
```

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

Version is HTTP/1.1

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.
- 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.

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.
- 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.

1.4 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.

As shown in Figure, a user's browser can be configured so that all of the user's HTTP requests are first directed to the Web cache.

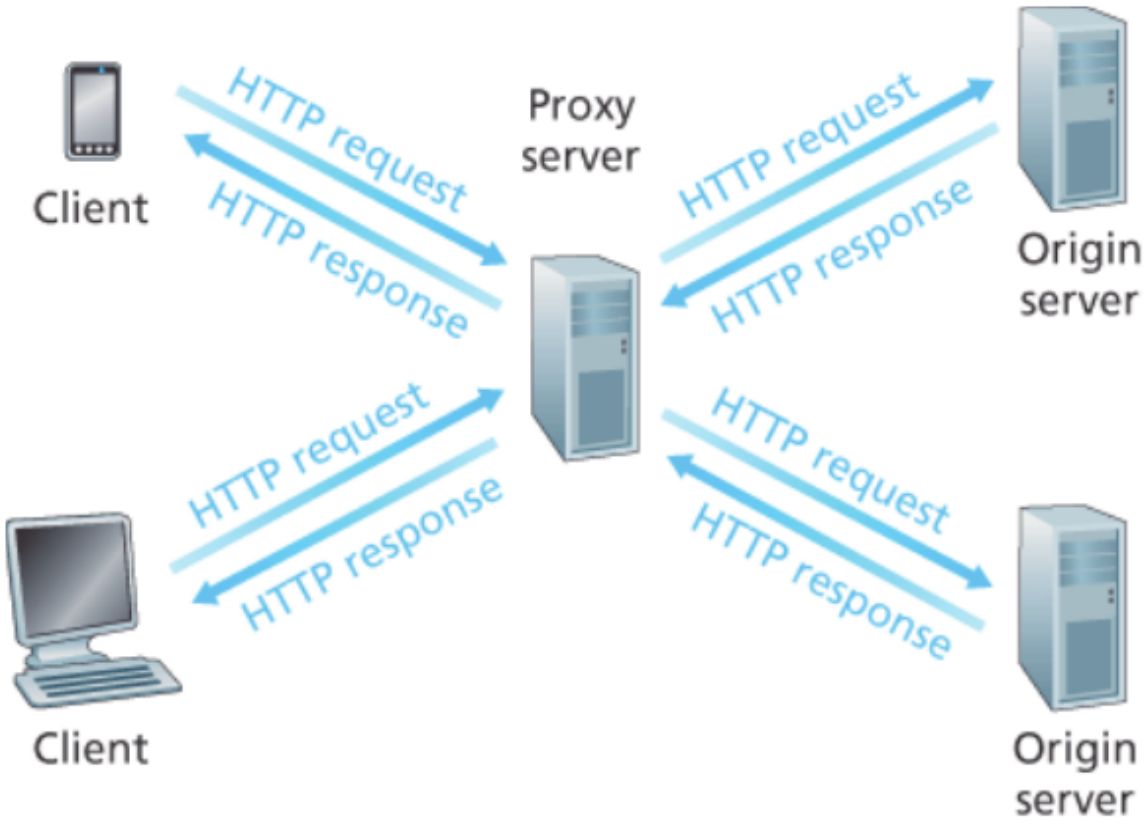

Figure 1.5: Clients requesting objects through a Web cache

As an example, 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. 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.

1.5 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.

- 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 is a so-called conditional GET message if the request message uses the *GET* method and 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, 3 Oct 2015 15:39:29
Server:  Apache/1.3.0 (Unix)
Last-Modified:  Wed, 9 Sep 2015 09:23:24
Content-Type:  image/gif

(data data data data data ...)
```

The cache forwards the object to the requesting browser but also caches the object locally and also stores the last-modified date along with the object.

One week later, another browser requests the same object via the cache, and the object is still in the cache.
Since this object may have been modified at the Web server in the past week, the cache performs an up-to-date check by issuing a conditional *GET*. Specifically, the cache sends:

```
GET /fruit/kiwi.gif HTTP/1.1
Host:  www.exotiquecuisine.com
If-modified-since:  Wed, 9 Sep 2015 09:23:24
```

This conditional GET is telling the server to send the object only if the object has been modified since the specified date. Suppose the object has not been modified since 9 Sep 2015 09:23:24.

Then, the Web server sends a response message to the cache:

```
HTTP/1.1 304 Not Modified
Date:  Sat, 10 Oct 2015 15:39:29
Server:  Apache/1.3.0 (Unix)

(empty entity body)
```

In the last response message has *304 Not Modified* in the status line, which tells the cache that it can go ahead and forward its cached copy of the object to the requesting browser.

1.6 FTP

[Jan/Feb-2021] [Aug/Sept-2020] [Dec/Jan-2020] (8M)

1.6.1 Working of FTP

FTP is used for transferring file from one host to another host.

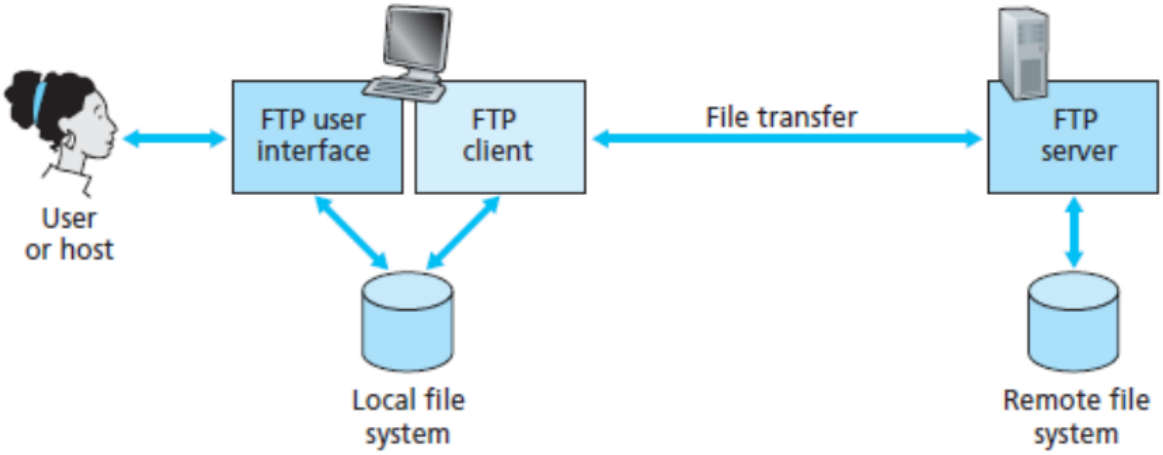

Figure 1.6: Working of FTP

- The user first provides the hostname of the remote host, causing the FTP client process in the local host to establish a TCP connection with the FTP server process in the remote host.
- The user then provides the user identification and password, which are sent over the TCP connection as part of FTP commands.

- Once the server has authorized the user, the user copies one or more files stored in the local file system into the remote file system (or vice versa).
- FTP uses two parallel TCP connections to transfer a file, a control connection and a data connection.
  - The control connection is used for sending control information between the two hosts.
  - The data connection is used to actually send a file.

Figure 1.7: FTP with two connections

1.6.2 FTP Commands and Replies

The commands, from client to server, and replies, from server to client, are sent across the control connection in 7-bit ASCII format. In order to delineate successive commands, a carriage return and line feed end each command. Each command consists of four uppercase ASCII characters, some with optional arguments.
Some of the common commands are given below:

- `USER username:` Used to send the user identification to the server.
- `PASS password:` Used to send the user password to the server.
- `LIST:` Used to ask the server to send back a list of all the files in the current remote directory.
- `RETR filename:` Used to retrieve a file from the current directory of the remote host.
- `STOR filename:` Used to store a file into the current directory of the remote host.

Each command is followed by a reply, sent from server to client. The replies are three–digit numbers, with an optional message following the number.

- `331 Username OK, password required`
- `125 Data connection already open; transfer starting`
- `425 Can't open data connection`
- `452 Error writing file`

1.6.3 Comparision with HTTP

HTTP	FTP
HTTP is used to access websites	FTP transfers file from one host to another
HTTP establishes data connection only.	FTP establishes two connection one for data and one for the control connection.
HTTP uses TCP's port number 80.	FTP uses TCP's port number 20 and 21.
HTTP does not require authentication.	FTP requires a password.
HTTP is efficient in transfering smaller files like web pages.	FTP is efficient in transfering larger files.

1.7 Electronic Mail in the Internet

E-mail has three major components: user agents, mail servers, and the Simple Mail Transfer Protocol SMTP).

1. **User agents** allow users to read, reply to, forward, save, and compose messages.
2. **Mail servers** form the core of the e-mail infrastructure. Each recipient has a mailbox located in one of the mail servers.
3. **SMTP** is the principal application-layer protocol for Internet electronic mail.

1.7.1 SMTP

SMTP transfers messages from senders' mail servers to the recipients' mail servers. It restricts the body of all mail messages to simple 7-bit ASCII.

For Example: Suppose Alice wants to send Bob a simple ASCII message:

1. 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.
2. Alice's user agent sends the message to her mail server, where it is placed in a message queue.
3. 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.
4. After some initial SMTP handshaking, the SMTP client sends Alice's message into the TCP connection.
5. 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.

6. Bob invokes his user agent to read the message at his convenience.

The scenario is summarized in Figure.

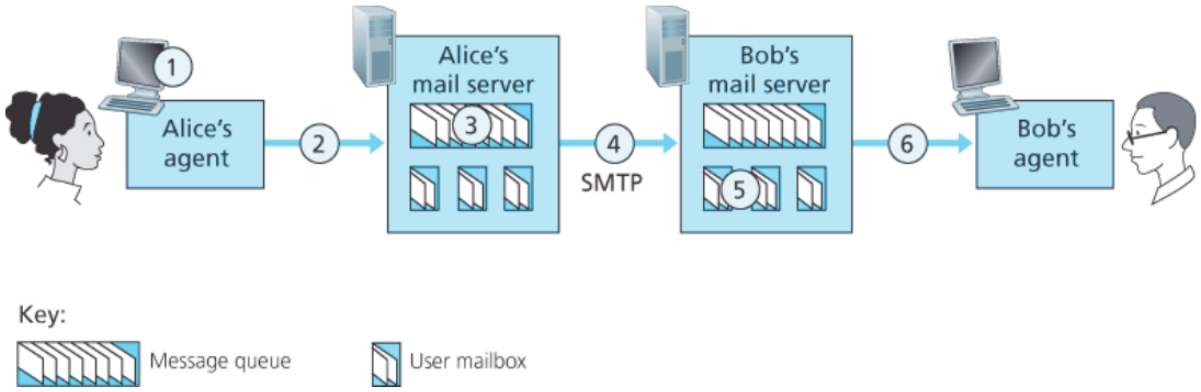

Figure 1.8: Alice sends a message to Bob

1.7.2 Comparision with HTTP

HTTP	SMTP
Pull Protocol–someone loads information on a Web server and users use HTTP to pull the information from the server at their convenience.	*Push Protocol*–the sending mail server pushes the file to the receiving mail server.
HTTP does not mandates data to be in 7-bit ASCII format.	SMTP requires each message, including the body of each message, to be in 7-bit ASCII format.
HTTP encapsulates each object in its own HTTP response message.	Internet mail places all of the message's objects into one message.

1.7.3 POP3

POP3 is an extremely simple mail access protocol.

- POP3 begins when the user agent opens a TCP connection to the mail server on port 110.
- With the TCP connection established, POP3 progresses through three phases: *authorization*, *transaction*, and *update*.
- During the *authorization* phase, the user agent sends a username and a password to authenticate the user.
- The *authorization* phase has two principal commands: `user <username>` and `pass <password>`.

```
user bob
+OK
pass hungry
+OK user successfully logged on
```

- During the *transaction* phase, the user agent retrieves messages; also during this phase, the user agent can mark messages for deletion, remove deletion marks, and obtain mail statistics.
- The *update* phase 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 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.
  Example:

```
C: list
S: 1 498
S: .
C: retr 1
S: (blah blah ...
S: ..................
S ...........  blah)
S: .
C: dele 1
C: quit
S: +OK POP3 server signing off
```

1.8 DNS

1.8.1 Services Provided by DNS

[Aug/Sept-2020]

In order for the user's host to be able to send an HTTP request message to the Web server `www.someschool.edu`, the user's host must first obtain the IP address of `www.someschool.edu`. This is done as follows:

1. The same user machine runs the client side of the DNS application.

2. The browser extracts the hostname from the URL and passes the hostname to the client side of the DNS application.
3. The DNS client sends a query containing the hostname to a DNS server.
4. The DNS client eventually receives a reply, which includes the IP address for the hostname.
5. Once the browser receives teh IP address from DNS, it can initiate a TCP connection to the HTTP server process located at port 80 at that IP address.

DNS provides a few other important services in addition to translating hostnames to IP addresses:

[Writing one point for each is enough]

- Host aliasing:
  - A host with complicated hostname can have one or more alias names.
  - For example, a hostname such as `relay1.west-coast.enterprise.com` could have, say, two aliases such as `enterprise.com` and `www.enterprise.com`. In this case, the hostname `relay1.west-coast.enterprise.com` is said to be *canonical hostname*.
- Mail server aliasing:
  - For obvious reasons, it is highly desirable that e-mail addresses be mnemonic.
  - If Bob has an account with Hotmail, Bob's e-mail address might be as simple as `bob@hotmail.com`. However, the hostname of the Hotmail server is more complicated and much less mnemonic than simply `hotmail.com`.
  - DNS can be invoked by a mail application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host.
- Load distribution:
  - DNS is also used to perform load distribution among replicated servers, such as replicated Web servers.

1.8.2 DNS Records and Messages

Dec/Jan-2020

The DNS servers that together implement the DNS distributed database store resource records (RRs).

A resource record is a four-tuple that contains the following fields:// `(Name, Value, Type, TTL)`

TTL is the time to live of the resource record; it determines when a resource should be removed from a cache.

The meaning of Name and Value depend on Type:

- If *Type=A*, then *Name* is a hostname and *Value* is the IP address for the hostname.
- If *Type=NS*, then *Name* is a domain and *Value* is the hostname of an authoritative DNS server that knows how to obtain the IP addresses for hosts in the domain.
- If *Type=CNAME*, then *Value* is a canonical hostname for the alias hostname Name. This record can provide querying hosts the canonical name for a hostname.
- If *Type=MX*, then *Value* is the canonical name of a mail server that has an alias hostname Name.

DNS Message

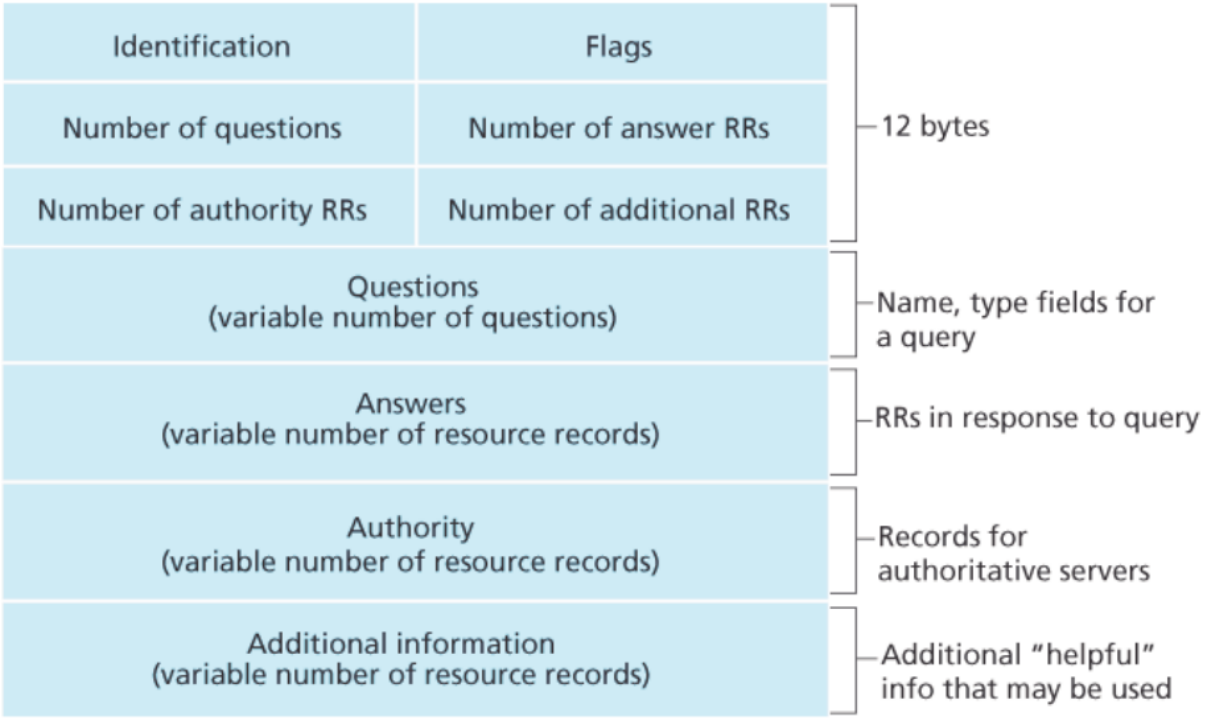

Figure 1.9: DNS Message Format

- The first 12 bytes is the header section.
- The first field is a 16-bit number that identifies the query.
- There are a number of flags in the flag field.
  - A 1-bit query/reply flag indicates whether the message is a query (0) or a reply (1).
  - A 1-bit authoritative flag is set in a reply message when a DNS server is an authoritative server for a queried name.
  - A 1-bit recursion-desired flag is set when a client desires that the DNS server perform recursion when it doesn't have the record.

  - A 1-bit recursion available field is sent in a reply if the DNS server supports recursion.

- The *question* section contains information about the query that is being made.
- In a reply from a DNS server, the answer section contains the resource records for the name that was originally queried.
- The *authority* section contains records of other authoritative servers.
- The *additional* section contains other helpful records.

1.9 Peer-to-Peer Applications

1.9.1 Working of BitTorrent for file distribution

The collection of all peers participating in the distribution of a particular file is called a *torrent*. Each *torrent* has an infrastructure node called a *tracker*.

When a peer joins a torrent, it registers itself with the tracker and periodically informs the tracker that it is still in the torrent.

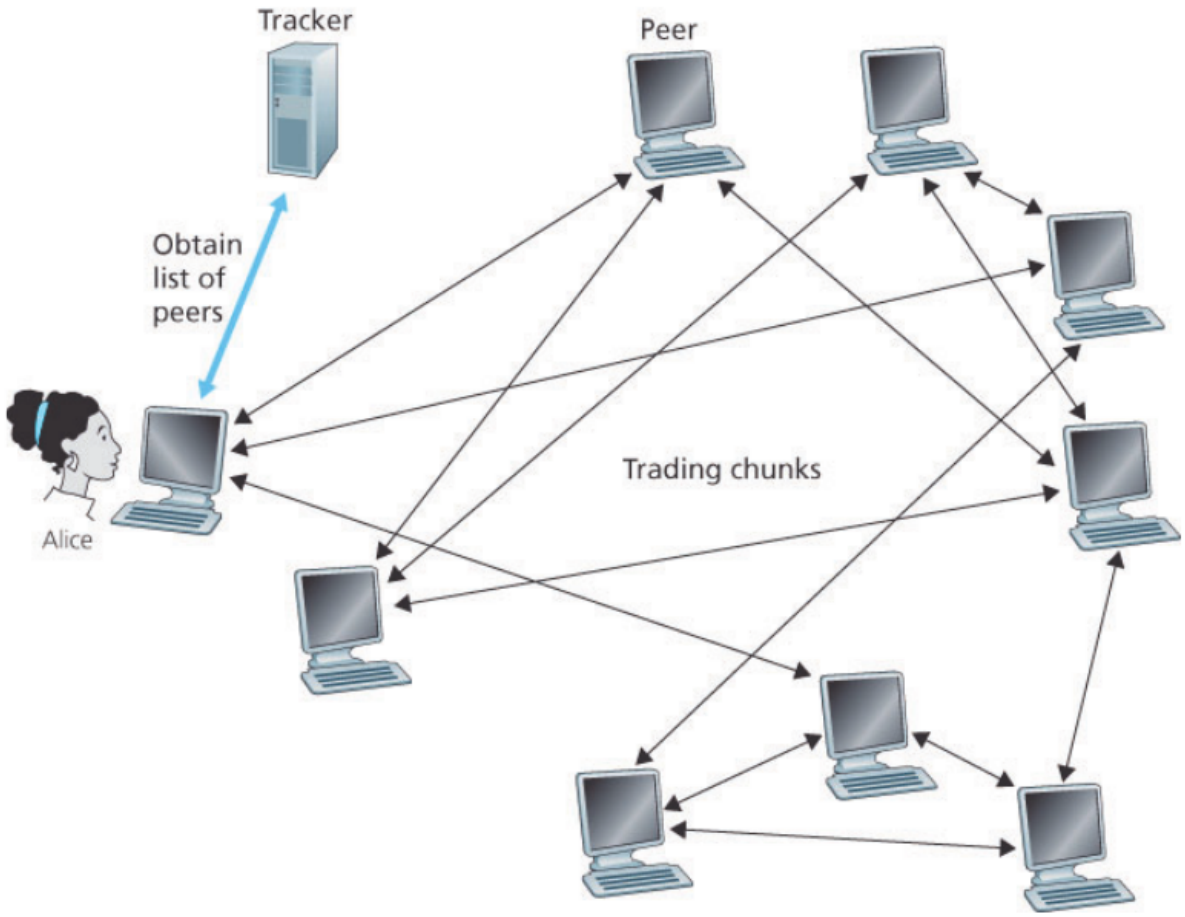

Figure 1.10: File distribution with BitTorrent

As shown in figure, when Alice joins the torrent, the tracker randomly selects a subset of peers and sends the IP address of these peers to Alice. Possessing this list of peers, Alice attempts to establish concurrent TCP connections with all the peers on this list.

At any given time, each peer will have a subset of chunks from the file, with different peers having different subsets. Periodically, Alice will ask each of her neighboring peers for the list of the chunks they have.

So at any given instant of time, Alice will have a subset of chunks and will know which chunks her neighbors have. Alice uses a technique called *rarest first*. The idea is to determine, from among the chunks she does not have, the chunks that are the rarest among her neighbors and then request those rarest chunks first. In this manner, the rarest chunks get more quickly redistributed.

To determine which requests she responds to, BitTorrent uses a clever trading algorithm. Alice continually measures the rate at which she receives bits and determines the four peers that are feeding her bits at the highest rate. She then reciprocates by sending chunks to these same four peers. Every 10 seconds, she recalculates the rates and possibly modifies the set of four peers.