COMPUTER NETWORKING

Study Notes

Zhijie Xia

Study Notes

github.com/zhijie-os

Contents

Chapter 1

Computer Networks and the Internet

1.1 The Internet

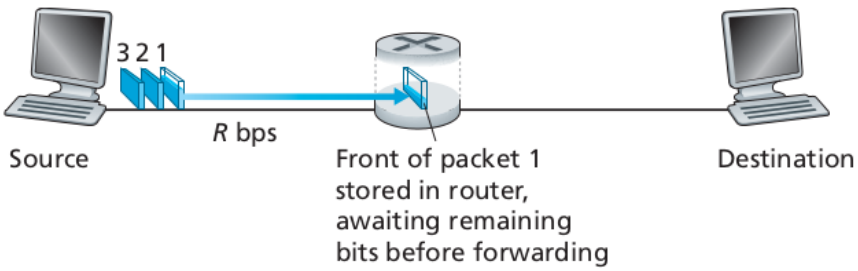
1.1.1 Terminology

1. ISP: Internet Service Provider
2. Host: End System
3. DSL: Digital Subscriber Line
4. FTTH: Fiber To The Home
5. LAN: Local Area Network
6. WAN: Wide Area Network

1.2 Packet Switching

packets :smaller chunks of data known as .

1.2.1 Store and Forward Transmission



1. The packet switch must receive the entire packet before it can transmit it.
2. Store: the switch buffer the packet's bits.
3. Forward: transmit the packet onto the outbound link.

1.2.2 Queuing Delays and Packet Loss

1. Each node has an output buffer/queue.
2. Each packet on the buffer wait for its turn to be transmitted.
3. Queuing Delay: The time that packet waits in the buffer.
4. Packet Loss: when buffer is completey full, the arriving packets will be dropped.

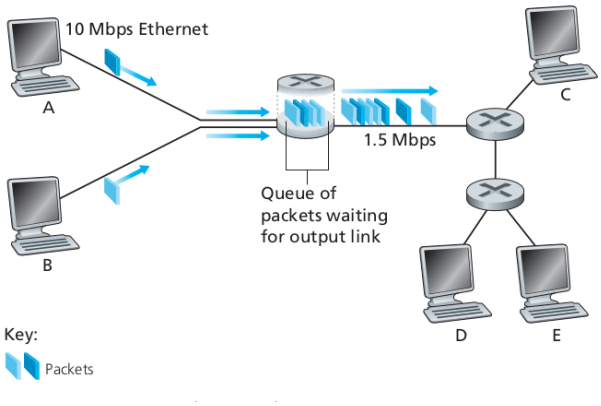


Figure 1.1: Packet Switching

1.2.3 Forwarding Tables and Routing Protocols

1. Each router has an IP address.
2. Each router has a **forwarding table**: Maps the destination addresses to outbound links.
3. End-to-end routing <=> taxi driver who doesn't use map but instead prefers to ask the directions.

1.3 Circuit Switching

Usage: traditional telephone networks.

1. Circuit Switching reverse resources.

1.3.1 Multiplexing in Circuit-Switched networks

1. FDM: frequency-division multiplexing
2. TDM: time-division multiplexing

FDM:frequency-division multiplexing

Frequency spectrum of a link is divided up among the connections established across the link.
For example, FM radio stations use FDM to share frequency spectrum between 88MHZ nd 108MHZ.

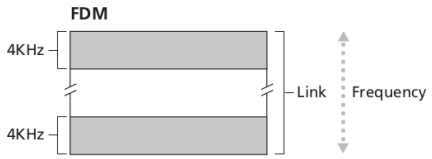


Figure 1.2: FDM

TDM: time-division multiplexing

Time is divided into frames of fixed duration, each frame is divided into fixed number of time slots. Each frame is like a round. At each frame, the use has guaranteed time slots to use.

1.3.2 Packet Switching Versus Circuit Switching

Advantage of Packet Switching

1. It offers better sharing of transmission capacity
2. Simpler, more efficient, and less costly to implement.

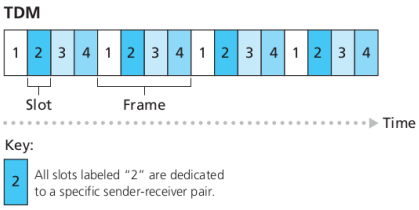


Figure 1.3: TDM

1.3.3 Why Packet Switching is more efficient?

Assumption: 90/10 rule: 90 percent of the time the user is idling.

Suppose users share a 1 Mbps link. Also suppose that each user alternates between peri- ods of activity, when a user generates data at a constant rate of 100 kbps, and periods of inactivity, when a user generates no data. Suppose further that a user is active only 10 percent of the time.

With Circuit Switching, 100 kbps must be reversed even the user is drinking coffee. Therefore only 1Mbps/100Kps = 10 users can share the link at the same time.

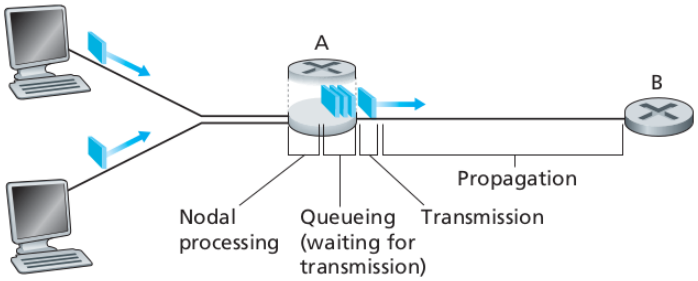
With Packet Switching, if there are 35 users, the probability that there are 11 or more users using the link simultaneously is less than 0.0004. Statistically, the packet switching is more efficient.

1.3.4 Advantage of Circuit Switching

1. Rate is guaranteed, suitable for real time services.

1.3.5 Delay, Loss, and Throughput in Packet-Switched Networks

Nodal delay = Processing Delay + Queuing Delay + Transmission Delay + Propagation Delay



1. Throughput: The amount of data per unit time that can be transferred between end hosts.
2. Processing Delay: The time required to examine the packet's header and determine where to direct packet is part of processing delay.
3. Queuing Delay: The time that the packet waits in the buffer.
4. Transmission Delay: The amount of time to put the complete packet onto the link.
5. Propagation Delay: The time required to propagate from the beginning of the link to the next node.

Processing Delay

Several parts of the processing delay:

1. The time required to examine the packet's header
2. The time required to check bit-level errors

Queuing Delay

The length of queuing delay is depended on the number of packets waiting in the buffer/queue. If the queue is empty and no other packet is currently being transmitted, the queuing delay will be zero.
Queuing can vary in a large scope. Queuing delays can be on the order of microseconds to milliseconds in practice.

Transmission Delay

Transmission delay depends on the physical medium of the link.
Transmission delay = the time the first bit is put on the link - the time the last bit is put on the link.
Denote the packet size L, and the transmission rate R.
The transmission delay will be $\frac{L}{R}$.

Propagation Delay

Propagation Delay depends on the physical medium of the link.
Denote the distanct between nodes d, and the speed of propagation (the speed of the link, usually a little less than the speed of light).
The propagation delay is d/s.
The progation delay can be negligible using fiber, but can be dominant if using satellite link.

1.3.6 Queuing Delay and Packet Loss

Unlike other delays, queuing delay can vary from packet to packet. For example 10 packets arrive at the same empty queue at the same time, the first packet will have zero queuing delay and the last packet will have relatively large queuing delay.
When is the queuing delay large and when is it insignificant? =¿ Depends on the rate of the arrving traffic(packets), the transmission rate of the link, and hether the traffic arrives periodically or arrives in bursts.

Let a denote the average rate at which packets arrive at the queue (a is in units of packets/sec). Recall that R is the transmission rate; that is, it is the rate (in bits/sec) at which bits are pushed out of the queue. Also suppose, for simplicity, that all packets consist of L bits.

Traffic instensity: $\frac{La}{R}$

If $\frac{La}{R} > 1$, the link is very busy and the queuing delay will continously increse and into infinite. If $\frac{La}{R} < 1$, is a necessary for reduce the queuing delay.

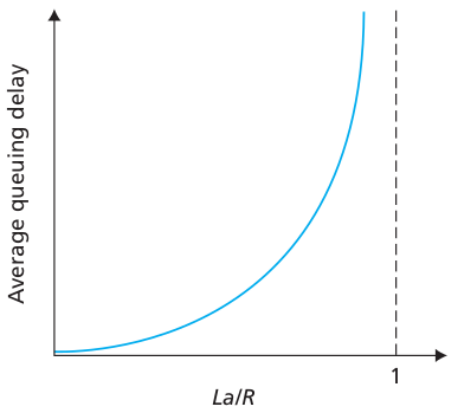


Figure 1.4: Dependence of average queuing delay on traffic intensity

1.3.7 End-to-End Delay

Suppose there are $N-1$ routers between the source host and the destination host. The transmission rate is fixed R, and the the distanct between each router is L. The end-to-end delay is:

$$d_{end-end} = \sum_{i=1}^{N} d_{proc} + d_{trans} + d_{prop} + d_{queu_i}$$

1.4 Protocol layering

Five-layer Internet Protocol Stack: Each layer provides services to the layer aboves it by 1)performing certain actions within that layer and by 2)using the services of the layer directly below it.

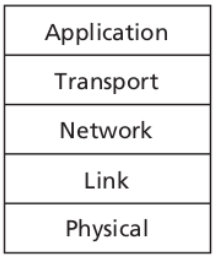


Figure 1.5: Five-layer Protocol Stack

1.4.1 Packet at different layers

1. Message : Application Layer
2. Segment : Transport Layer
3. Datagram : Network Layer
4. Frame : Link Layer

1.4.2 Application Layer

The Internet's application layer includes many protocols:

1. HTTP
2. SMTP
3. FTP
4. DNS
5. ...

1.4.3 Transport Layer

1. TCP
2. UDP

1.4.4 Networking Layer

Only one procotol is available at this layer =¿ **IP** (*Internet Protocol*).

1.4.5 Link Layer

Examples of linklayer protocols include:

1. Ethernet
2. WiFi
3. DOCIS

1.4.6 Physical Layer

The protocols in this layer are again link dependent and further depend on the actual transmission medium of the link For example, Ethernet has many physical-layer protocols:

1. twisted-pair copper wire
2. coaxial cable
3. fiber

need to be

In each case, a bit is moved across the link in a different way.

1.5 Encapsulation

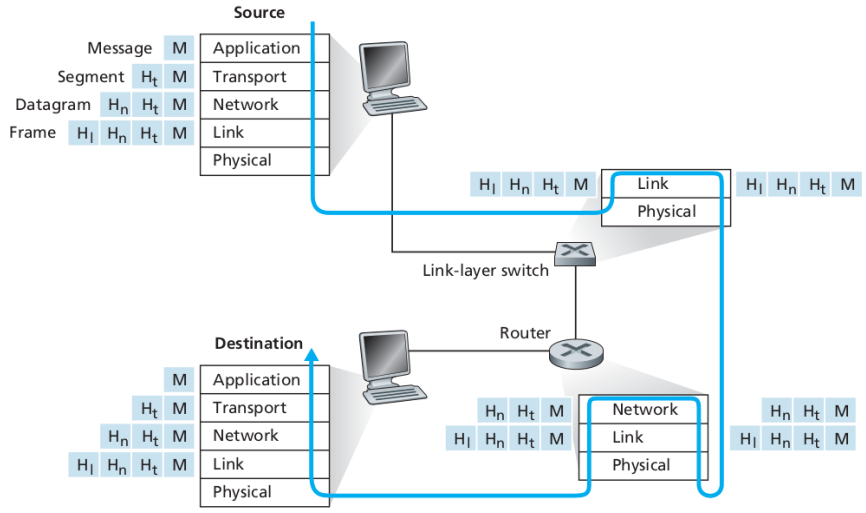


Figure 1.6: Encapsulation

Chapter 2

Application Layer

2.1 Principles of the Network Applications

2.1.1 Network Applicatoin Architectures

Application Architecture is designed by the application developer and *dictates how the application is structured* over various end systems.

Two predominant architectural paradigms:

1. client-server architecture
2. peer-to-peer(P2P) architecture

Client-Server

1. Server: the always-on host which services requests from many other hosts.
2. Client: the end host whic request services from the server.

Note that with the client-server architecture, clients do not directly communicate with each other;for example, in the Web application, two browsers do not directly communi- cate.
Another characteristic of the 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 well-known applications within client-server architecture:

1. Web
2. FTP
3. Telnet
4. e-mail

2.1.2 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.

Characteristics that P2P architecture has:

1. ***Self-Scalability***:although each peer generates workload by requesting files, each peer also adds service capacity to the system by distributing files to other peers.
2. Cost effictive: normally don't require significant server infrastructure and server bandwidth

The problems that P2P need to face:

1. ISP's asymmetrical bandwidth usage: ISPs are dimensioned for much more downstream than upstream traffic. But in p2p architecture, every residential ISPs will face a gigantic amount of upstream traffic.

2. Security: Because of their highly distributed and open nature, P2P applications can be a challenge to secure.

3. Incentives: Need users to volunteer bandwidth, storage, and computation resources to application.

2.1.3 Transport services available to applications

We can broadly classify the possible services along four dimensions:

1. Reliable data transfer

2. Throughput

3. Timing

4. Security

2.2 Socket

Most applications consist of pairs of communicating processes, with the two processes in each pair sending messages to each other.
Socket: the interface that a process sends messages into, and receives messages from.

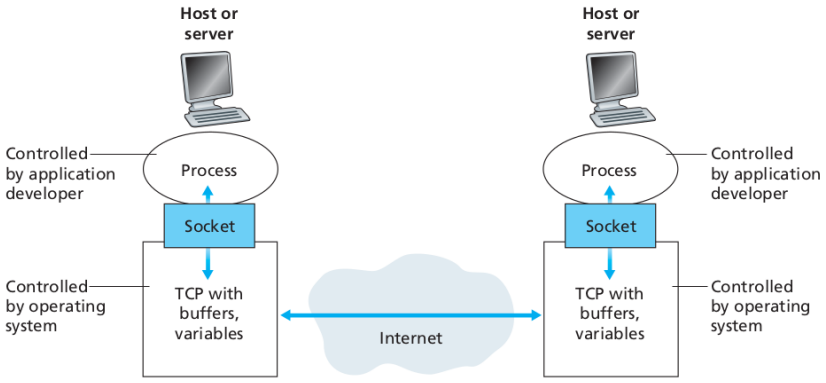


Figure 2.1: Socket

Socket is also referred as the API between the application and the network. The developer has control of everything on the application-layer side but only little control of the transport-layer side of the socket:

1. the choice of transport protocol
2. fix a few transport-layer parameters such as maximum buffer and maximum segment sizes.

2.2.1 Address Processes

Similarly, in order for a process running on one host to send packets to a process running on another host, the receiving process needs to have an address. In particular:

1. The address of the host: ***IP address***.
2. an identifier that specifies the receiving process in the destination host: ***Port number***.

Some ports are reserved by well-known applications. For example, a Web server has port number 80. A mail server process has port number 25.

2.3 The Web and HTTP

The Web operates on demand.

2.3.1 Overiew of HTTP

The Web's application-layer protocol: HTTP(HyperText Transfer Protocol). HTTP is implemented both in client program and server program. HTTP defines the structure of the messages and how the client and server exchange the messages.

Web browsers: Client side of HTTP.
Web servers: Server side of HTTP, addressable by a URL.

HTTP uses TCP

The flow:

1. The HTTP client initiates a TCP connection with the server.
2. The HTTP client sends request messages into its socketnd and waits to receive HTTP response messages from its socket interface.
3. The HTTP server receives request from its socket and sends response into its socket interface.
4. The HTTP client receives the response.

Another important thing is that HTTP is ***stateless protocol***.

2.3.2 HTTP with different connections

There are two connection types for HTTP:

1. Non-persistent connection: the application sends response using existing TCP connection.
2. Persistent connection: the appication sends each response with a new TCP connection.

RTT(Round Trip Time)
Non-persistent connection will take around two RTT for a single HTML file,and for each TCP established the TCP variables need to be stored in both client and server side. This can place a significant burden on the Web server.

Persistent connection will only take one RTT once the connection is established, and persistent connection with pipelining is the default mode of the HTTP.

2.3.3 HTTP Request Format

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

The first line is called: the ***request line***; The subsequent lines are called: the ***header lines***.
The request line has three fields:

1. The method field: including ***GET,POST,HEAD,PUT*** and ***DELETE***.
2. The URL field
3. The HTTP version field

The great majority of HTTP request messages use the GET method. The GET method is used when the browser requests an object, with the requested object iden tified in the URL field.

The header line ***Host: wwwwww.someschool.edu*** specifies the host on which the object resides. The header line ***Connection: close*** tells the server close the connection after sending the requested object. The ***User-agent*** and ***Accept-Language*** are self-explanatory.

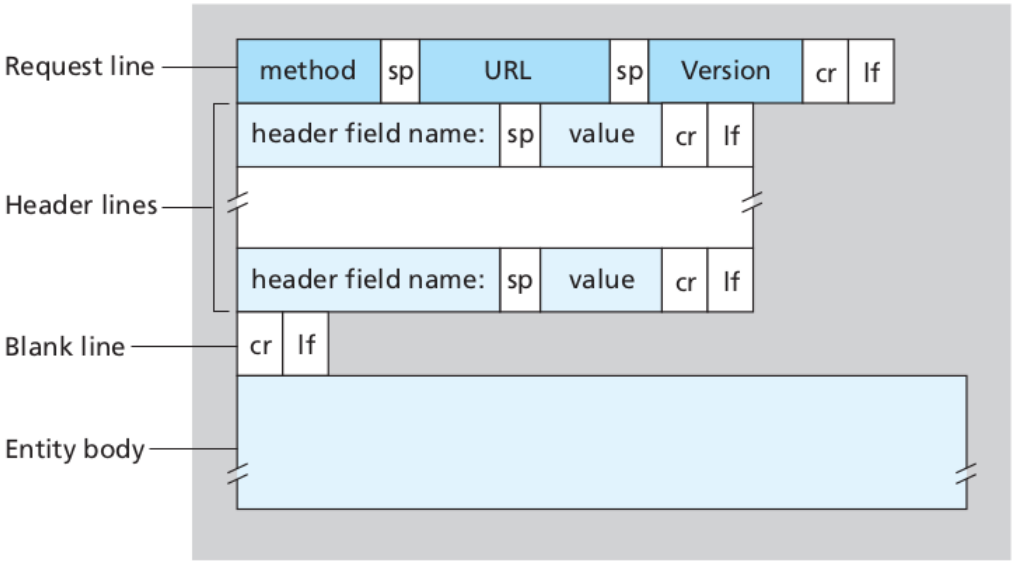


Figure 2.2: General format of an HTTP request message

The ***entity body*** is empty with ***GET*** method, but is used in ***POST*** method: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.
However HTML forms often use the ***GET*** method and include the inputted data in the request URL. For example, inputted data is banana and money, then the URL will look like *www.somesite.com/search?monkeys&bananas*.
The ***HEAD*** method only responds with an HTTP message but it leaves out the requested object(used to debug).
The ***PUT*** method allows a user to upload an object to specific path on specific Web server. It is also used by applications that need to upload objects to Web servers.
The ***DELETE*** method allows a user, or an application, to delete an object on a Web server.

2.3.4 HTTP Reponse Format

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 example has three sections: an initial ***status line***, six ***header line***, and then the ***entity body***.

The status line has three fields:

1. The protocol version field
2. A status code: 200 in the example
3. A corresponding status message.

The six header lines:

1. ***Connection: close*** header line: tell the client that it is going to close the TCP connection.
2. The ***Date:*** header line indicates the time and date when the HTTP response was created and sent by the server.
3. The ***Server:*** header line indicates that the message was generated by an Apache Web server.

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

General format of an HTTP response message

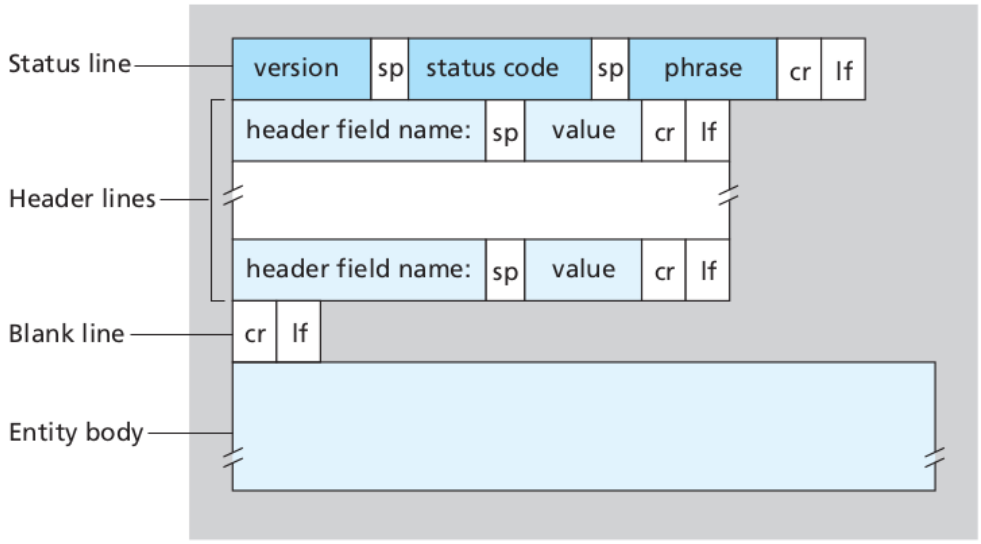


Figure 2.3: General format of an HTTP response message

Some common status codes and associated phrases include:

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

2.3.5 Cookies

HTTP use cookies to identify users.

Major cookies 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.
4. A back-end database at the Web site.

Flows of using cookies:

1. Alice send HTTP request message without a cookie header line.

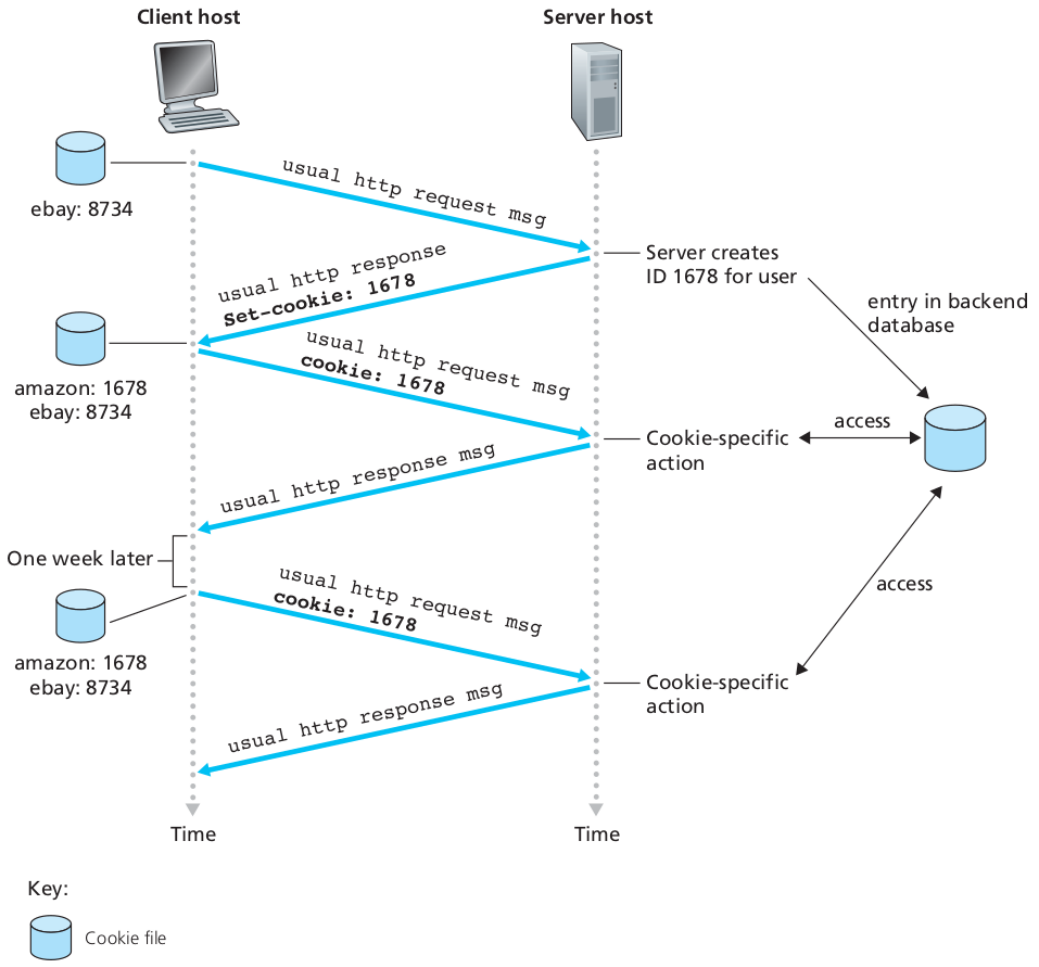


Figure 2.4: Use cookie to identify users

2. The server generates a new cookies number and stores it to the database.Then repsonses with Set-cookie: header ***Set-cookie: 1678***.

3. When Alice receives the response message, her browsers appends a line to the special cookie file that it manages.

4. In the following requests, Alice's browser will include the cookie for the server in the header lines. Like ***Cookie: 1678***.

5. The server will identify Alice with the cookie it receives in the header liens.

2.3.6 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 is both a client and a server*.

The web caching process:

1. The browser establishes TCP connection to the Web cache and sends 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 a copy of the object, it sends HTTP request for the object to the origin server.

4. The web cache receives the object from the origin server with HTTP response, it stores a copy of the object and sends back to the browser.

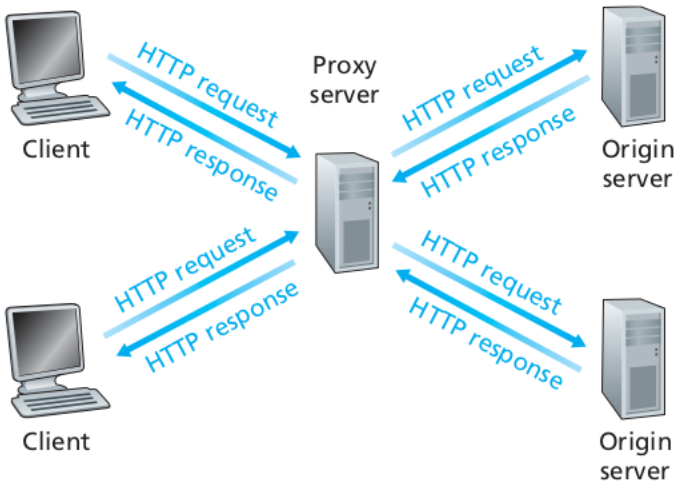


Figure 2.5: Web Caching

Typically a Web cache is purchased and installed by an ISP. For example, a uni- versity might install a cache on its campus network and configure all of the campus browsers to point to the cache.Or a major residential ISP (such as AOL) might install one or more caches in its network and preconfigure its shipped browsers to point to the installed caches.

Advantage of using a web cache/proxy server

1. 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 bot- tleneck bandwidth between the client and the cache.

2. Web caches can substantially reduce traffic on an institution's access link to the Internet. By reducing traffic, the institution does not have to upgrade bandwidth as quickly, thereby reducing costs.

3. Web caches can substantially reduce Web traffic in the Internet as a whole, thereby improving performance for all applications.

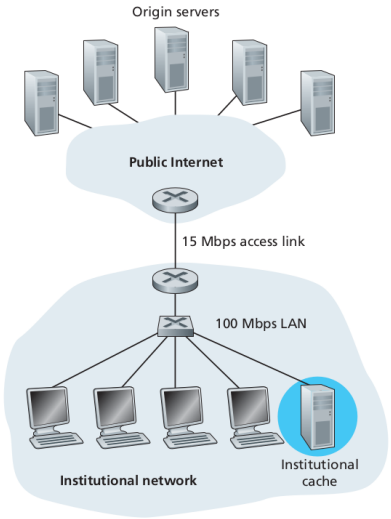


Figure 2.6: Adding a cache to the institutional network

Suppose that the average object size is 1Mbits and that the average request rate from instution's browsers to origin server is 15 requests per second.

The traffic intensity on the LAN is:

$$(15requests/sec) \cdot (1Mbits/request)/(100Mps) = 0.15 \tag{2.1}$$

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

$$(15requests/sec) \cdot (1Mbits/request)/(15Mps) = 1 \tag{2.2}$$

Which is really bad and goes to infinite.

However, when we use the Web cache in the Figure 2.6 when the hit rate is 0.4, we will a traffic intensity on the access link:

$$(15 * 0.6requests/sec) \cdot (1Mbits/request)/(15Mps) = 0.6 \tag{2.3}$$

Which is much better.

The conditional GET

The Web cache uses ***conditional GET*** to check whether the copy of the object is up to date. A HTTP request message is ***conditional GET*** if

1. The request message uses the ***GET*** method.
2. The request message includes an ***If-Modified-Since:*** header line.

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

GET /fruit/banana HTTP/1.1
HOST: www.walmart.ca

Second, the Web server receives the HTTP response from origin server:

HTTP/1.1 200 OK
Date: Sat, 8 Oct 2011 15:39:32
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 Web cache would forward the object to the requesting browser and store a copy of it. When the next time someone sends a same request message, the Web cache would send a ***conditional GET***:

GET /fruit/banana HTTP/1.1
HOST: www.walmart.ca
If-modified-since: Wed, 7 Sep 2011 09:23:24

Suppose the object is not updated after the last ***GET***, the origin server would response:

HTTP/1.1 304 Not Modified
Date: Sat, 15 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)
(empty entity body)

304 Not Modified indicates that the requested object is not modified since Wed, 7 Sep 2011 09:23:24. Therefore, it doesn't attach the data in the entity. The Web cache would forward the local copy of the object back to the browser.

Note that an HTTP request/response message is negligible. Therefore, a ***conditional GET*** does not cause delay to forward the copy of the object to the browser if there is no modification made.

2.4 FTP: File Transfer Protocol

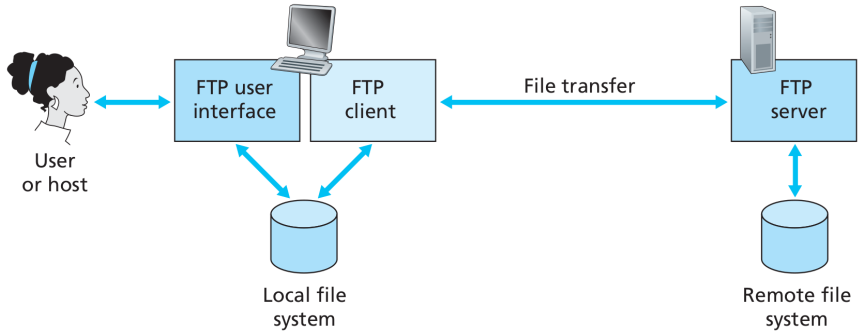


Figure 2.7: FTP

2.4.1 Overview of FTP

In Figure 2.7:

1. User interacts with FTP with an FTP user agent and provides the hostname of the remote host.
2. The process in the local host starts a TCP connection with the FTP server process.
3. The user provides identification and password via the established TCP connection.
4. The server authenticates and authorizes the user.
5. User can copy files from(into) the server(local).

2.4.2 Control connection and data connection

The ***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 processes - information such as user identification, password, commands to change remote directory, and commands to "put" and "get" files.

The ***data connection*** is used to actually send a file.
Because TCP uses a seperate control connection, FTP authenticateis said to send its control information ***out-of-band***. HTTP sends request and response header lines into the same TCP connection that carries the transferred file itself. Therefore, HTTP is said to send its control information ***in-band***.

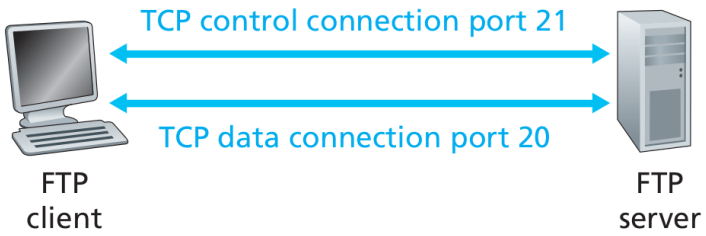


Figure 2.8: Control and data connections

1. The client process starts an FTP session with a remote host.
2. The client process initiate a control TCP connection with the server process on ***port 21***.
3. The client process sends user identification and password over the control connection.

4. The client process sends a command via control connection.
5. The server process receives a command for a file transfer over the control connection.
6. The server initiates a TCP data connection to client process on ***port 20***.

Note that FTP sends exactly one file over the data connection and closes the data connection. If, during the same session, the user wants to transfer another file, FTP opens another data connection. Thus, with FTP, the control connection remains open throughout the duration of the user session. New data connection is created for each file transferred within a session.

1. Data connection is non-persistent.
2. Control connection is persistent.

Also, the FTP server must maintain ***state*** about the user:

1. Associate the control connection with specific user account
2. Keep track of the user's current directory in the remote diretory tree.

Keeping track of this state information for each ongoing user session significantly constrains the total number of sessions that FTP can maintain simultaneously.

2.4.3 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. FTP commands are readable by people. Each command consists of four uppercase ASCII characters, some with optional arguments.

Some common commands:

1. USER username: Used to send the user identification to the server.
2. PASS password: Used to send the user password to the server.
3. LIST: Used to ask the server to send back a list of all the files in the current remote directory. The list of files is sent over a (new and non-persistent) data connection.
4. RETR filename: Used to retrieve (that is, get) a file from the current direc- tory of the remote host. This command causes the remote host to initiate a data connection and to send the requested file over the data connection.
5. STOR filename: Used to store (that is, put) a file into the current directory of the remote host.

Each command is followed by a reply, sent from server to client.

Some common replies:

1. 331 Username OK, password required
2. 125 Data connection already open; transfer starting
3. 425 Can't open data connection
4. Error writing file

2.5 Electronic Mail in the Internet

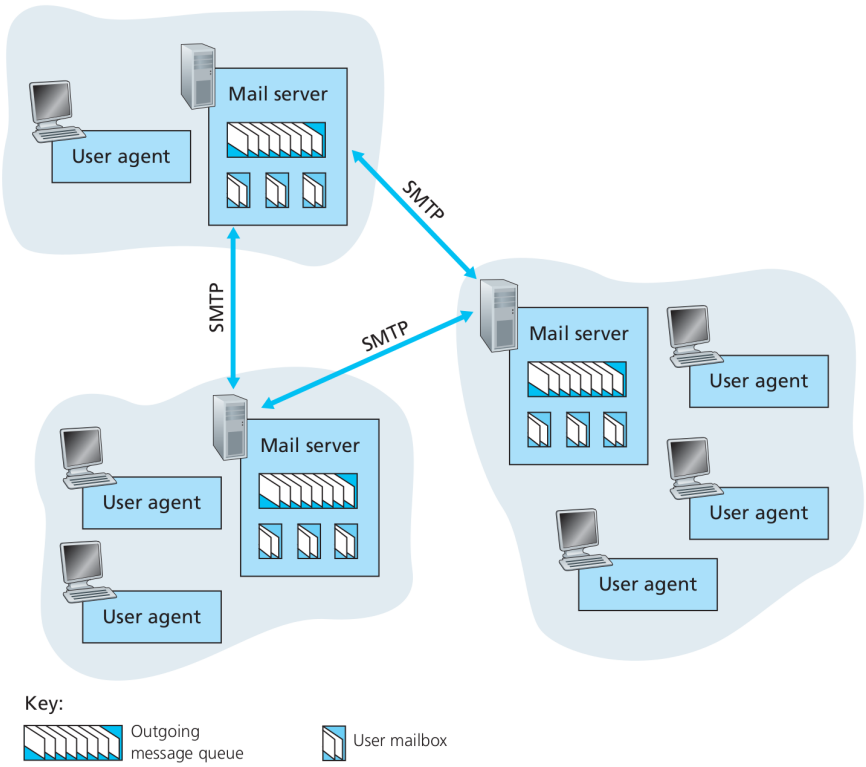


Figure 2.9: High-level view of email system

Three major components of in Figure 2.9:

1. User agents
2. Mail servers
3. SMTP: Simple Mail Transfer Protocol

Sender's view:

1. User agent allows the sender to compose the message.
2. The user agent sends the message to the email server.
3. The message is placed in the mail server's outgoing message queue.
4. The recipent retrieves the message from his/her mailbox in the mail server.

Mail servers form the core of the e-mail infrastructure. Each recipent has a ***maibox*** located in one of the mail servers. The mailbox manages and maintains the messages that have been sent to the recipent.

The typical message's journey:

1. Sender's user agent.
2. Sender's mail server.
3. Recipent's mail server.
4. Recipent's mailbox.

The sender mail server must also deal with failures in the recipent mail box: If the sender server can not delivery mail to the recipent server, sender server holds the message in a message queue and attempts to transfer the message later.

2.5.1 SMTP: Simple Mail Tranfer Protocol

SMTP uses TCP to transfer mail ***from the sender's mail server to the recipent's mail server***.

When a mail server sends mail to other mail servers, it acts as an SMTP client. When a mail server receives mail from other mail servers, it acts as an SMTP server.

SMTP restricts the body of all mail messages to simple 7-bit ASCII.

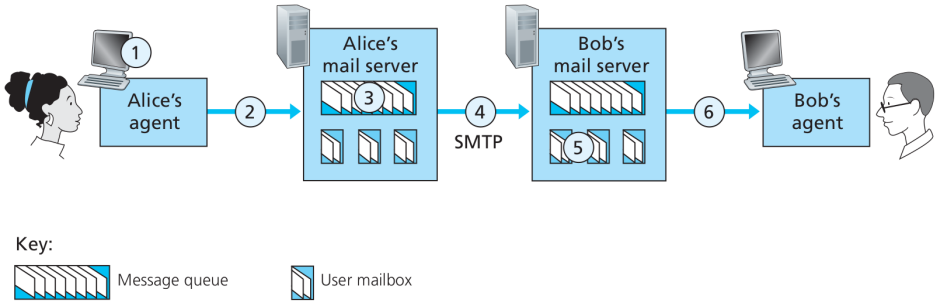


Figure 2.10: Alice sends a message to Bob

When Alice sends Bob an ASCII message:

1. Alice invokes her user agent for e-mail, provides Bob's e-mail address (for example, bob@someschool.edu), 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.

Detailed SMTP

SMTP does not use intermediate email servers for sending email. SMTP establishes a connection to port 25 at the server SMTP. Once the connection is established, the server and client perform some application layer-handshaking. SMTP client indicates the e-mail address of the sender and the e-mail address of the recipent. Once the handshaking is complete, the client sends the message and repeats if there are more messages to send to the same server.

Example:

1. S: 220 hamburger.edu
2. C: HELO crepes.fr
3. S: 250 Hello crepes.fr, pleased to meet you
4. C: MAIL FROM: ¡alice@crepes.fr¿
5. S: 250 alice@crepes.fr ... Sender ok
6. C: RCPT TO: ¡bob@hamburger.edu¿
7. S: 250 bob@hamburger.edu ... Recipient ok
8. C: DATA

9. S: 354 Enter mail, end with "." on a line by itself
10. C: Do you like ketchup?
11. C: How about pickles?
12. C: .
13. S: 250 Message accepted for delivery
14. C: QUIT
15. S: 221 hamburger.edu closing connection

Comparison with HTTP

Common: Both persistent HTTP and SMTP use persistent connections.

Difference:

1. The HTTP is maily a***pull protocol***; SMTP is primarily a ***push protocol***.
2. The message has to be encoded into 7-bit ASCII while using SMTP; HTTP doesn't impose this restriction.
3. HTTP encapsulates each object in its own HTTP response message; Internet mail places all of the message's objects into one message.

Mail message format

A typical message header looks like this:

From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Searching for the meaning of life.

Mail access protocol

Recall that a mail server manages mailboxes and runs the client and server sides of SMTP. If Bob's mail server were to reside on his local PC, then Bob's PC would have to remain always on, and connected to the Internet. Instead, a typical user runs a user agent on the local PC but accesses its mailbox stored on an always-on shared mail server.This mail server is shared with other users and is typically maintained by the user's ISP.

How can Bob retrieve his email from the mailbox?

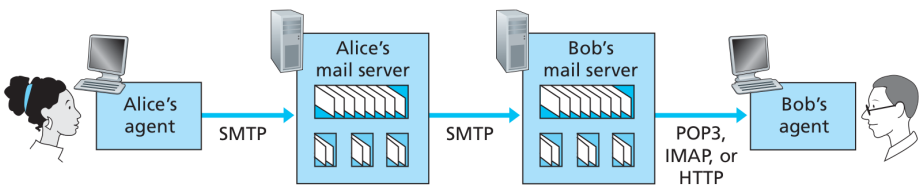


Figure 2.11: Retrieve mail from mailbox

Protocols like :***POP3*** (Post Office Protocol version 3.0), ***IMAP*** (Internet Mail Access Protocol), and ***HTTP*** are used to pull mail from mail server to the recipent's agent.

POP3

POP3 is extremly simple mail access protocol, and it is short and readable. Because the protocol is so simple, its functionality is rather limited.

POP3 begins when the user agent opens a TCP connection to mail server on ***port 110***.With the TCP connection estab- lished, POP3 progresses through three phases: authorization, transaction, and update.

During the first phase, authorization, the user agent sends a username and a password (in the clear) to authenticate the user.

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, and obtain mail statistics.

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.

In a POP3 transaction, the user agent issues commands, and the server responds to each command with a reply. There are two possible responses: ***+OK*** (sometimes followed by server-to-client data), 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.

The sequence of commands issued by a POP3 user agent depends on which of these two modes the user agent is operating in. In the download-and-delete mode, the user agent will issue the list, retr, and dele commands.

Transaction phrase example:

1. C: list
2. S: 1 498
3. S: 2 912
4. S: .
5. C: retr 1
6. S: (blah blah ...
7. S: .....
8. S: .....blah)
9. S: .
10. C: dele 1
11. C: retr 2
12. S: (blah blah ...
13. S: .....
14. S: .....blah)
15. S: .
16. C: dele 2
17. C: quit
18. S: +OK POP3 server signing off

2.5.2 IMAP: Internet Mail Access Protocol

The POP3 doesn't provide any means of a user to create remote folders and assign messages to folders. =¿ IMAP is introduced to solve this problem, but it is significantly more complex.

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.

Another important feature of IMAP is that it has commands that permit a user agent to obtain components of messages. For example, a user agent can obtain just the message header of a message or just one part of a multipart MIME message. This feature is useful when there is a low-bandwidth connection between the user agent and its mail server.

2.5.3 Web-Based E-mail

Hotmail introduced Web-based access in the mid 1990s.With this service, the user agent is an ordinary Web browser, and the user communicates with its remote mailbox via HTTP.

2.6 DNS: Domain Name System — The Internet's Directory service

Domain name vs Host name: hostname.domain.com

1. Hostname is the name given to the end-point(a machine).
2. Domain is the name given to the "network".

Internet hosts can be identified by different ways: by ***Hostname***, such as www.google.com or so-called ***IP address***.

2.6.1 Services Provided by DNS

From my perspective, DNS does one thing: search up IP address by hostname.
Note that ***DNS uses UDP and uses port 53***.
The DNS is

1. a distributed database implemented in a hierarchy of ***DNS servers***.
2. an application-layer protocol that allows host to query the distributed database.

DNS is commonly employed by other application-layer protocols—including HTTP, SMTP, and FTP — to translate user-supplied hostnames to IP addresses. 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, www.someschool.edu, 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 the IP address from DNS, it can initiate a TCP con- nection to the HTTP server process located at port 80 at that IP address.

DNS adds an additional delay (sometimes substantial) - to the Internet applications that use it. Fortunately, the desired IP address is often cached in a "nearby" DNS server, which helps to reduce DNS network traffic as well as the average DNS delay.
DNS provides a few other important services:

1. Host aliasing
2. Mail server aliasing
3. Load distribution: DNS is also used to perform load distribution among replicated servers, such as replicated Web servers. When the client makes a DNS query for a name mapped to a set of addresses, the server responds with the entire set of IP addresses, but rotates the ordering of the addresses within each reply.

2.6.2 Overview of How DNS Works

In order to deal with the issue of scale, the DNS uses a large number of servers, organized in a hierarchy fashion and distributed around the world. No single DNS server has all the mappings for all of the hosts in the Internet. Instead, the mappings are distributed across the DNS servers.

There are three classes of DNS servers — root DNS servers, top-level domain (TLD) DNS servers, and authoritative DNS servers — organized in a hierarchy in Figure 2.12.

When searching for www.amazon.com:

1. The client contacts one of the root servers.
2. The root server returns IP addresses for TLD servers for top-level domain com.

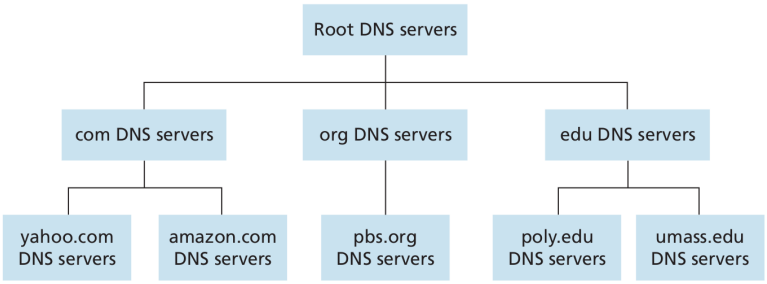


Figure 2.12: DNS Hierarchy

3. The client contacts one of the TLD servers.
4. The TLD server returns IP addresses of authoritative servers for amazon.com.
5. The client contacts one of the authoritative servers for amazon.com.
6. The authoritative server returns the IP for www.amazon.com

Root DNS servers: There are 13 root DNS servers in 2012:

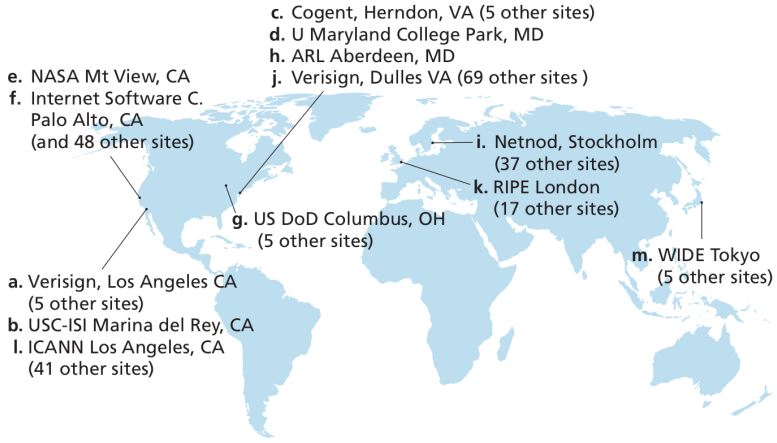


Figure 2.13: DNS Root Servers in 2012

Top-level domain(TLD) servers:
These servers are responsible of top-level domains suck as com, org, net, edu, and gov.

Authoritative DNS servers:
Every organization with publicly accessible hosts on the Internet must provide publicly accessible DNS records that map the names of those hosts to IP addresses.
Local DNS servers:
A local DNS does not strictly belong to the hierarchy of servers but it is nevertheless central to the DNS architecture. When a host makes a DNS query, the query is sent to the local DNS server, which acts a proxy, forwarding the query into the DNS server hierarchy.

Local DNS server services as a cache.The idea behind DNS caching is very simple. In a query chain, when a DNS server receives a DNS reply (containing, for example, a mapping from a host- name to an IP address), it can cache the mapping in its local memory.

The example in **??** makes use of both ***recursive queries*** and ***iterative*** queries. The query sent from cis.poly.edu to dns.poly.edu is a recursive query, since the query asks dns.poly.edu to obtain the mapping on its behalf. But the subsequent three queries are iterative since all of the replies are directly returned to dns.poly.edu. In theory, any DNS query can be iterative or recursive.

In the Figure 2.15 makes use of ***recursive queries***.

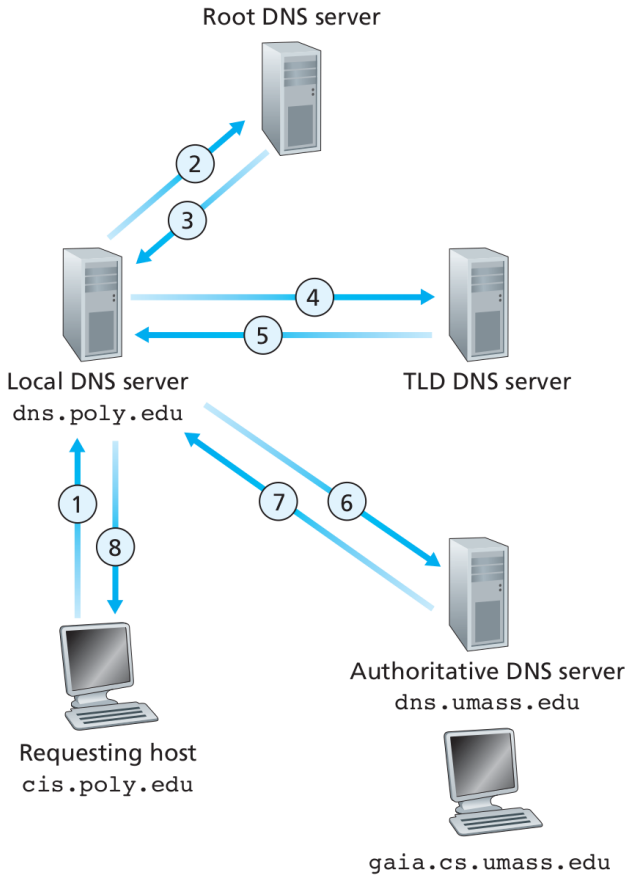


Figure 2.14: Interaction of the various DNS servers

2.6.3 DNS records and messages

The DNS servers that together implement the DNS distributed database store resource records (RRs), including RRs that provide hostname-to-IP address map- pings.

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.

Type field:

1. If Type = A, then Name is a hostname and Value is the IP address for the hostname.Example,(relay1.bar.foo.com, 145.37.93.126, A, TTL holder).
2. 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. Example,(foo.com, dns.foo.com, NS, TTL).
3. If Type = CNAME, then Name is an alias hostname and Value is the canonical name for the hostname. Example(foo.com, relay1.bar.foo.com, CNAME, TTL).
4. If Type = MX, then Name is an alias hostname of a mail server and the Value is the canonical name of the mail server. Example(foo.com, mail.bar.foo.com, MX, TTL).

2.6.4 DNS messages

There only two kinds of DNS messages: Query and Reply. Furthermore, both query and reply messages have the same format, as shown in Figure 2.16.

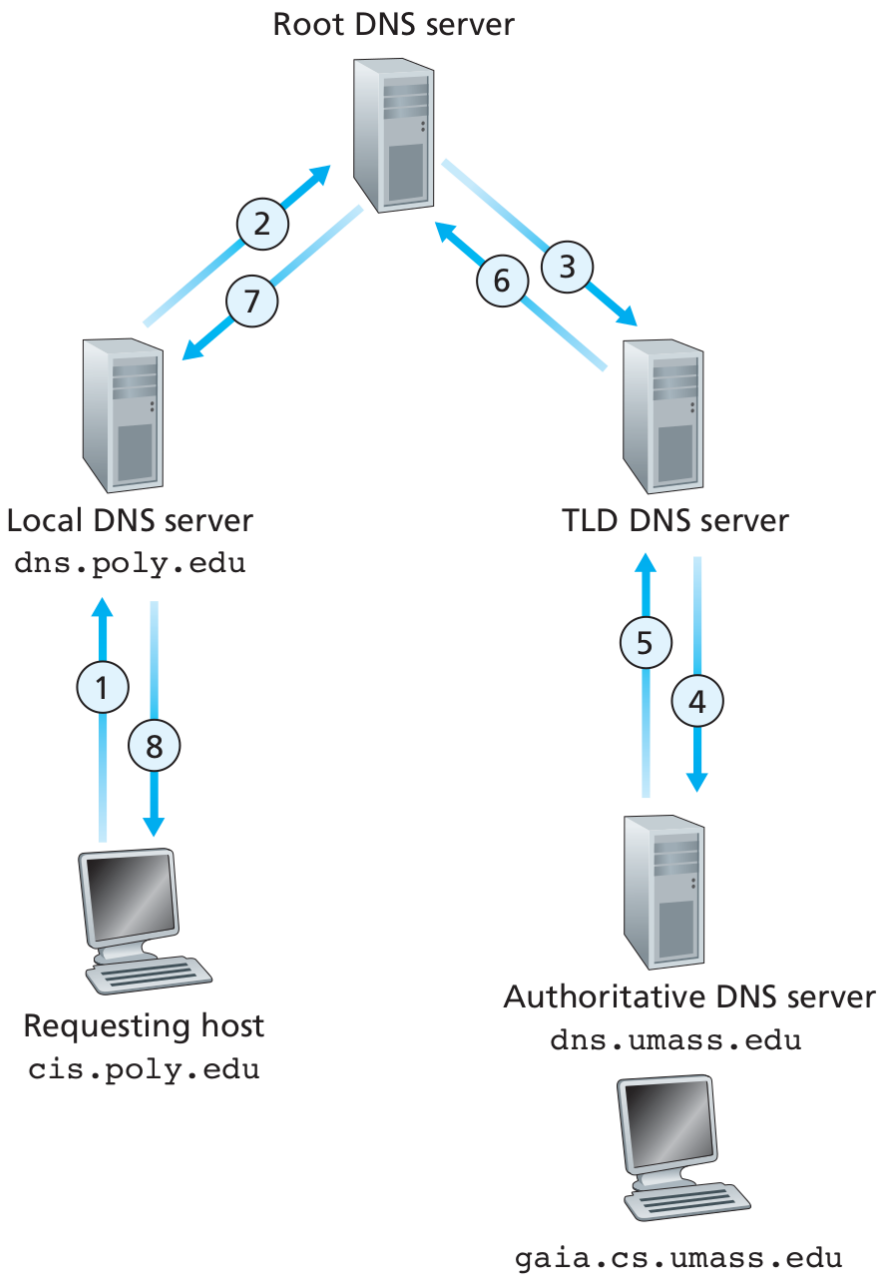


Figure 2.15: Recursive Queries

1. The first 12 bytes is the header section, which has a number of fields. The first field is a 16-bit number that identifies the query. This identifier is copied into the reply message to a query, allowing the client to match received replies with sent queries. 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 (host or DNS server) desires that the DNS server perform recursion when it doesn't have the record. A 1-bit recursion- available field is set in a reply if the DNS server supports recursion. In the header,there are also four number-of fields. These fields indicate the number of occurrences of the four types of data sections that follow the header.

2. The question section contains information about the query that is being made. This section includes (1) a name field that contains the name that is being queried, and (2) a type field that indicates the type of question being asked about the name—for example, a host address associated with a name (Type A) or the mail server for a name (Type MX).

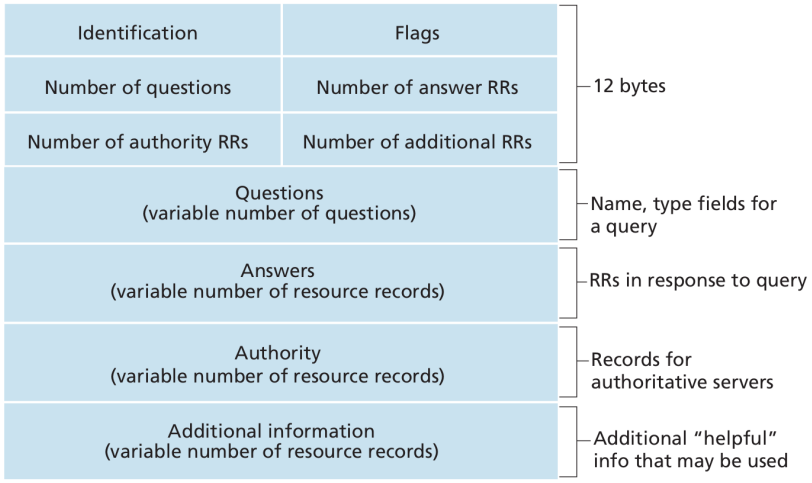


Figure 2.16: DNS message format

3. In a reply from a DNS server, the answer section contains the resource records for the name that was originally queried. Recall that in each resource record there is the Type (for example, A, NS, CNAME, and MX), the Value, and the TTL. A reply can return multiple RRs in the answer, since a hostname can have multi- ple IP addresses (for example, for replicated Web servers, as discussed earlier in this section).

4. The authority section contains records of other authoritative servers.

5. The additional section contains other helpful records. For example, the answer field in a reply to an MX query contains a resource record providing the canoni- cal hostname of a mail server. The additional section contains a Type A record providing the IP address for the canonical hostname of the mail server.

2.7 Peer-to-Peer Applications

With P2P architecture, there is minimal(or no) reliance on always-on infrastructure servers. Instead, pairs of intermittently connected hosts, called peers, communicate directly with other. The peers are not owned by a service provider, but are instead desktops and laptops controlled by users.

2.7.1 P2P File Distribution

Scalability of P2P Architecture

P2P is self-scaling because The number of "servers" would increase along the time.

Figure 2.17: Distribution time for P2P and client-server architectures

BitTorrent

BitTorrent is a popular P2P protocol for file distribution. In BitTor- rent lingo, the collection of all peers participating in the distribution of a particular file is called a ***torrent***.Peers in a torrent download equal-size chunks of the file from one another, with a typical chunk size of 256 KBytes. When a peer first joins a torrent, it has no chunks. Over time it accumulates more and more chunks. While it downloads chunks it also uploads chunks to other peers. Once a peer has acquired the entire file, it may leave the torrent, or (altruistically) remain in the torrent and continue to upload chunks to other peers. Also, any peer may leave the torrent at any time with only a subset of chunks, and later rejoin the 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. In this manner, the tracker keeps track of the peers that are participating in the torrent.

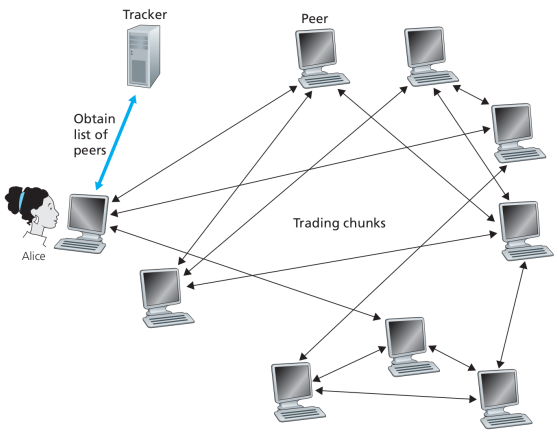


Figure 2.18: File distribution with BitTorrent

Say Alice joins the torrent, and the tracker randomly selects 50 peers and sender IP addresses of these 50 peers to Alice. Alice attempts to establish concurrent TCP connections with all peers on this list.

Neighboring Peers: The peers with which Alice succeeds in establishing a TCP connection.

As time evolves, some of these peers may leave and other peers may attempt to establish TCP connections with Alice. So a peer's neighboring peers will fluctuate over time.

Some strategies that BitTorrent uses

Q: "which chunks should she request first from her neighbors?" and "which of her neighbors should she send requested chunks?"
A: ***Rarest first***: The idea is to equalize the numbers of copies of each chunk in the torrent.

Q: Which requests to respond to?
A: The basic idea is that gives priority to the neighbors that are currently supplying data at the highest rate. In particular, a requesting peer A measures the rate at which it receives bits and determine the top four neighbors that are feeding her bits at the highest rate. Every 10 seconds, A recalculates the rates and possibly modifies the set of four peers. Importantly, every 30 seconds, A also picks one additional neighbor B at random and sends it chunks. Because A is feeding B data, A may become top four at B's list. B could also become one of the top Fortunately peers in A's list. If the two peers are satisfied with the trading, they will put each other in their top four lists and continue trading with each other until one of the peers finds a better partner.The effect is that peers capable of uploading at compatible rates tend to find each other. The random neighbor selection also allows new peers to get chunks, so that they can have something to trade. All other neighboring peers besides these five peers (four "top" peers and one probing peer) are "choked," that is, they do not receive any chunks from Alice.

2.7.2 Distributed Hash Tables (DHTs)

It is very straightforward to build a database with client-server architecture that stores all the (key, value) pairs in one central server. Instead P2P version of this database that will store the (key, value) pairs over millions of peers. In the P2P system, each peer will only hold a small subset of the totality of the (key, value) pairs. Any peer can query to the distributed database with a particular key and any peer can insert new key-value pairs into the database. Such a distributed database is called ***distributed hash table***(DHT).

Several approches to build a DHT

Approch 1: Randomly scatter the (key, value) pairs across all the peers and have each peer maintain a list of the IP addresses of all participating peers. In this design, the querying peer sends its query to all other peers, and the peers containing the (key, value) pairs that match the key can respond with their matching pairs.
Problem: The design is completely unscalable (brute force every query). Potientially too much traffic in the network.

Approch 2:Assign an identifier to each peer, where each identifier is an integer in the range [0, ,$2^n - 1$] for some fixed n. Given that each peer has an integer identifier and that each key is also an integer in the same range, a natu- ral approach is to assign each (key, value) pair to the peer whose identifier is the closest to the key.
This approach is more elegant. To implement such a scheme, need to define what is meant by "closet" for which many conventions are possible. =¿ "closest successor of the key"

But who to determine the peer that is closest to the key? If A were to keep track of all the peers in the system (peer IDs and corresponding IP addresses), A could locally determine the closest peer. But such an approach requires each peer to keep track of all other peers in the DHT—which is completely impractical for a large-scale system with millions of peers.

2.7.3 Circular DHT

In order for a peer to determine the peer that is closest to the key? Peers are organized into a circle. In this circular arrangement, each peer only keeps track of its immediate ***successor*** and immediate ***predecessor*** (modulo 2^n)

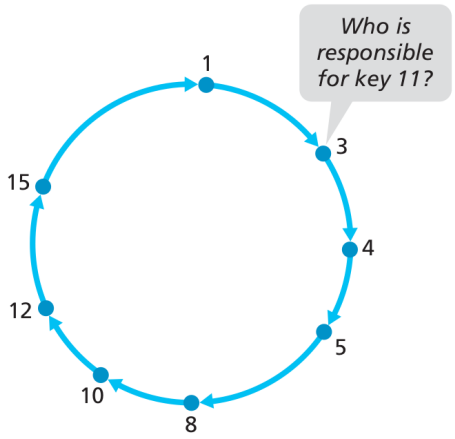


Figure 2.19: Peer 3 wants to determine who is responsible for key 11.

Some shortcuts can be added to speed up the query process.

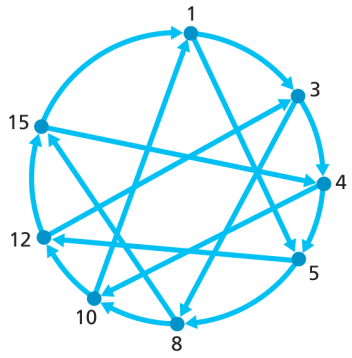


Figure 2.20: Circular DHT with shortcuts

.

2.8 Socket Programming: Create Network Applications

A typical network application consists of a pair of programs — a client program and a server program — residing in two different end systems.

2.8.1 Socket Programming With UDP

Chapter 3

Transport Layer

3.1 Introduction and Transport-Layer Services

A transport-layer protocl provides for ***logical communication*** between application processes running on different hosts. Transport-layer protocols are implemented in the end systems but not in network routers.

On sending side, the transport layer converts the ***message*** it receives into ***segments***. This is done by breaking the ***message*** into smaller chunks and adding a transport-layer header to each chunk to create the ***segment***. Then the transport layer passes the segment to the network-layer at the sending end system, and sent to the destination. It is important to note that network routers act only on the network-layer fields of the datagram; that is, they do not examine the fields of the transport-layer segment encapsulated with the datagram.

On receiving side, the network-layer extracts the transport-layer segment from the datagram and passes the segment up to the transport layer. The transport layer then processes the received segment, making the data in the segment available to the receiving application.

3.1.1 Relationship Between Transport and Network Layers

Transport-layer protocol provides logical communication between processes running on different hosts.

Network-layer protocol provides logical communication between hosts.

The possible services can be provided by a transport-layer protocol is constrained by the possible services that the network-layer protocol — the Internet Protocol. If the network-layer protocol cannot provide delay or bandwidth guarantees for transport-layer segments send between hosts, then the transport-layer protocol cannot provide delay or bandwidth guarantees for application messages sent between processes.

Nevertheless, certain services can be offered by a transport protocol even when the underlying network protocol doesn't offer the corresponding service at the net- work layer. For example, a transport protocol can offer reliable data transfer service even IP cannot. A transport protocol can use encryption even IP cannot.

3.1.2 Overiew of the Transport Layer in the Internet

The Internet, and more generally TCP/IP network, makes two distinct transport-layer protocols available to the application layer.

One of these protocol is ***UDP***(User Datagram Protocol), which provides an unreliable, connectionless service to the invoking application.

The second of these protocol is ***TCP***(Transmission Control Protocol), which provides a reliable, connection-oriented service to the invoking application.

The Internet's network-layer protocol is called IP(Internet Protocol), which provideslogical communication between hosts. The IP provides "best-effort delivery service".

The most fundamental responsibility of UDP and TCP is to extend IP's delivery service between two end systems to a delivery service between two processes running on the end systems.

Minimal transport-layer services:

1. Process-to-process data delivery:Extending host-to-host delivery(IP) to process-to-process delivery by transport-layer ***multiplexing*** and ***demultiplexing***.
2. Error checking: UDP and TCP also provide integrity checking by including ***error-detection fields*** in their segments' headers.

3.2 Multiplexing and Demultiplexing

Demultiplexing: Deliverying the data in a transport-layer segment to the correct socket.

Multiplexing: Gathering data chunks at the source host from different sockets, encapsulating each data chunk with header information to create segments, and passing the segments to the network layer.

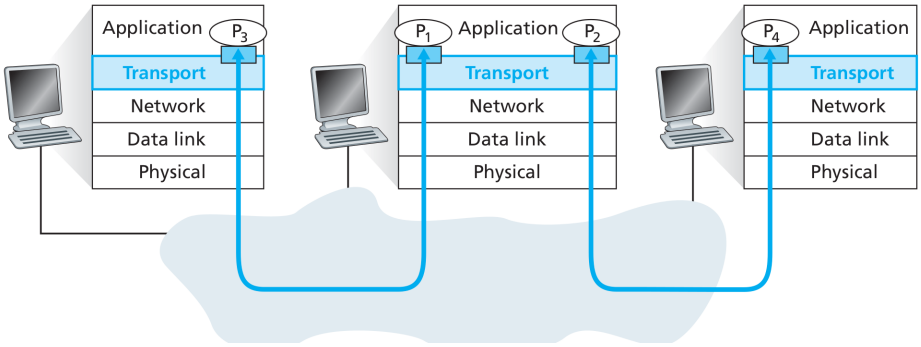


Figure 3.1: Transport-layer multiplexing and demultiplexing

3.2.1 How to achieve multiplexing and demultiplexing

Transport-layer multiplexing/demultiplexing requires

1. Source port: Unique Identifiers for sockets
2. Destination port: Each segment have special fields that indicate socket to which the segment is to be delivered.

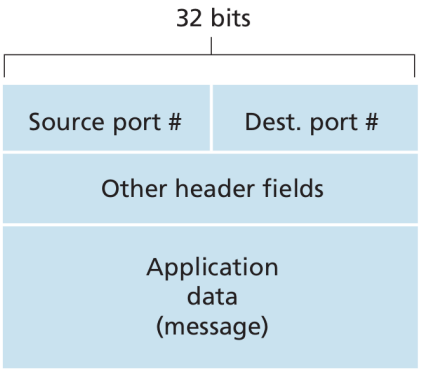


Figure 3.2: Source and destination port-number fields in a transport-layer segment

Each port number is a 16-bit number, ranging from 0 to 65535. The port numbers ranging ***from 0 to 1023*** are called ***well-know port numbers*** and ***are reserved*** for certain applications.

Connectionless(UDP) Multiplexing and Demultiplexing

UDP is fully identified by two-tuple consisting of

1. Destination IP address
2. Destination port number

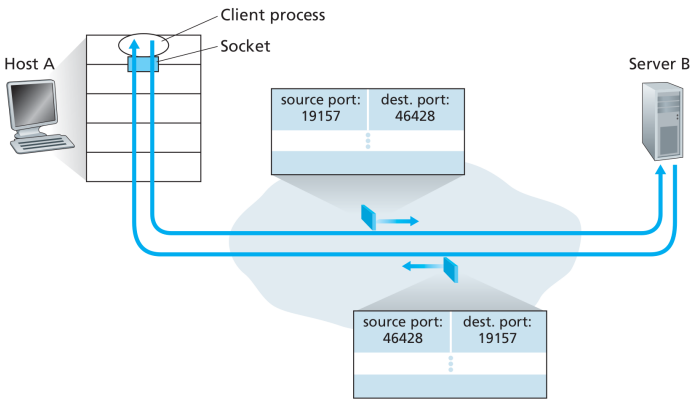


Figure 3.3: Connectionless Multiplexing and Demultiplexing

UDP server processes all clients via the same socket.

Connection-Oriented Multiplexing and Demultiplexing

TCP is identified by four-tuple consisting of

1. Source IP address
2. Source port number
3. Destination IP address
4. Destination port number

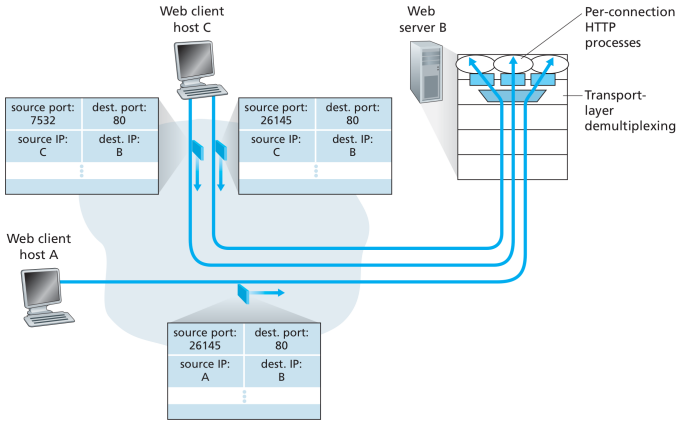


Figure 3.4: Connection-Oriented Multiplexing and Demultiplexing

TCP maintains a seperate socket for every connection. Why?

Because TCP implements reliability, it is easier with separate sockets. For example, when a message arrives at a socket, it is easy to simply see the source IP/port associated with that socket and send ACK. If you want to send a reply to a UDP message, you need to extract the source IP and port from the UDP header for every message.

3.3 Connectionless Transport: UDP

Basically, UDP pass application messages directly to the network layer after providing a multiplexing/demultiplexing service and some light error checking. UDP does just about as little as a transport protocol can do.

Why would one choose UDP rather than TCP?

1. Real-time applications cannot tolerate delay but can endure data loss.
2. Non-connection establishment: TCP connection-establishment delay is heavy.
3. No connection state: Can support many more active clients when the application runs over UDP rather than TCP.
4. Small packet header overhead: TCP only has 20 bytes of overhead. In comparison with TCP, UDP has much less header overhead — 8 bytes of overhead.

Why UDP is not ideal sometimes?

1. UDP has no congestion control: lead to a heavy congested network.

It is also possible for an application to have reliable data transfer when using UDP by building reliability into the application itself. But this is nontrivial task that would keep developers busy. Just use TCP if you need reliability.

3.3.1 UDP Segment Structure

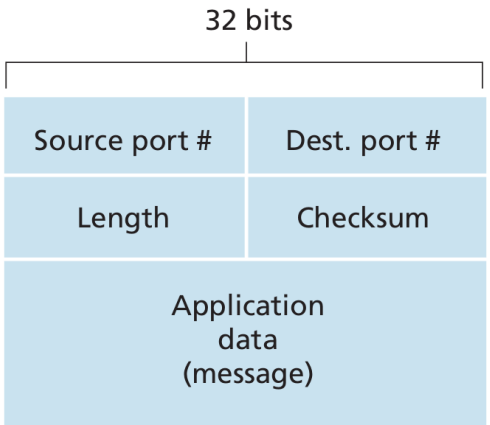


Figure 3.5: