COMPUTER NETWORKING

Study Notes

Zhijie Xia

Study Notes

github.com/zhijie-os

Contents

Chapter 1

Computer Networks and the Internet

1.1 The Internet

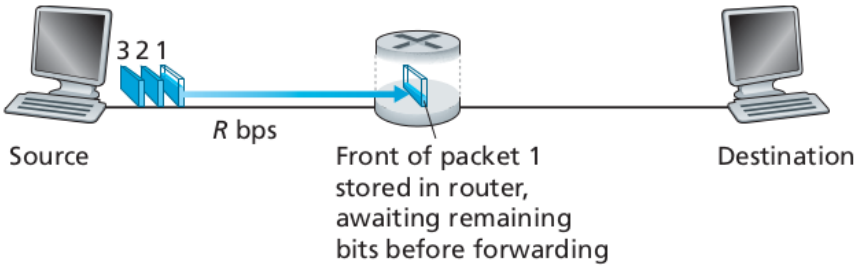
1.1.1 Terminology

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.

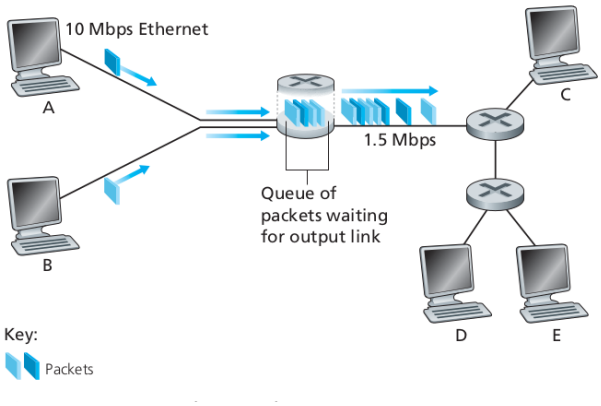


Figure 1.1: Packet Switching

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.

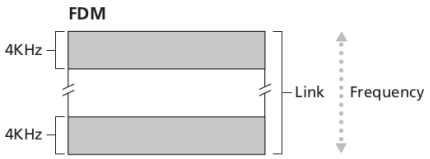


Figure 1.2: FDM

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.

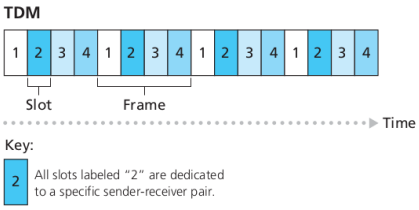


Figure 1.3: TDM

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.

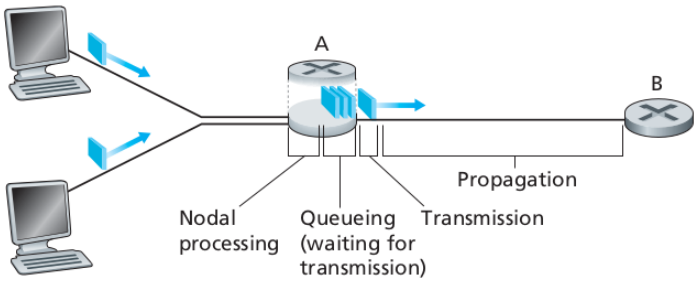
With Packet Switching, if there are 35 users, the probability that there are 11 or more users using the link simultaneously is less than 0.0004. Statistically, the packet switching is more efficient.

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.

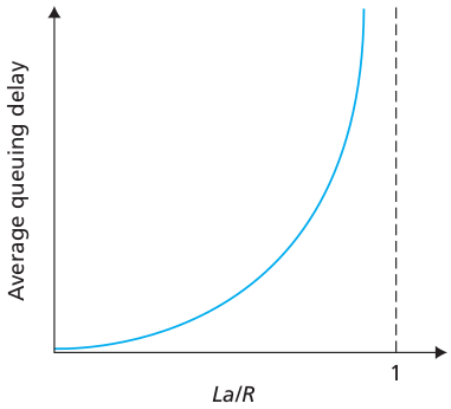


Figure 1.4: Dependence of average queuing delay on traffic intensity

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.

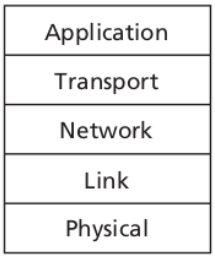


Figure 1.5: Five-layer Protocol Stack

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

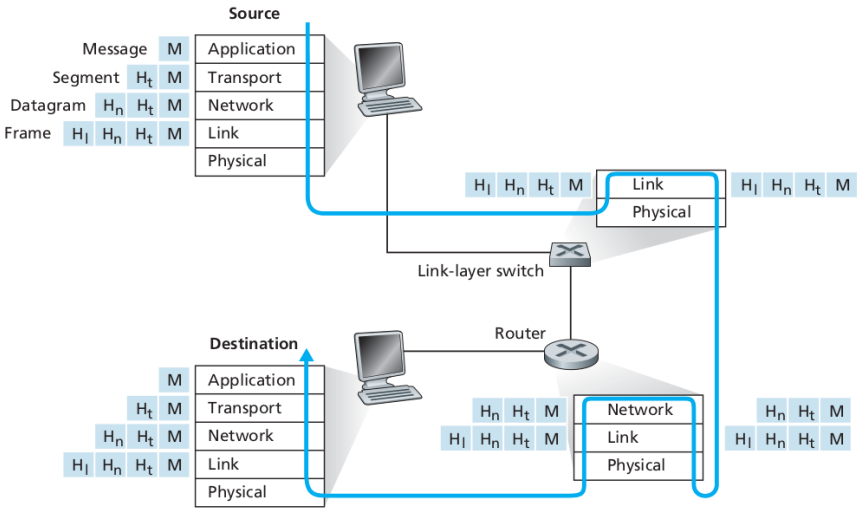


Figure 1.6: Encapsulation

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.

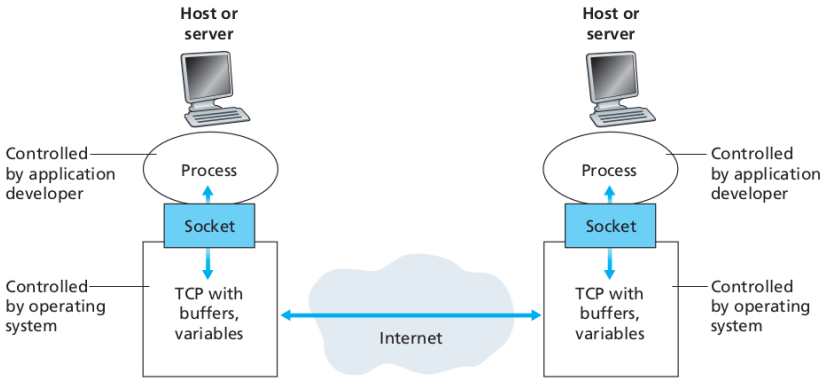


Figure 2.1: Socket

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: wwwww.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.

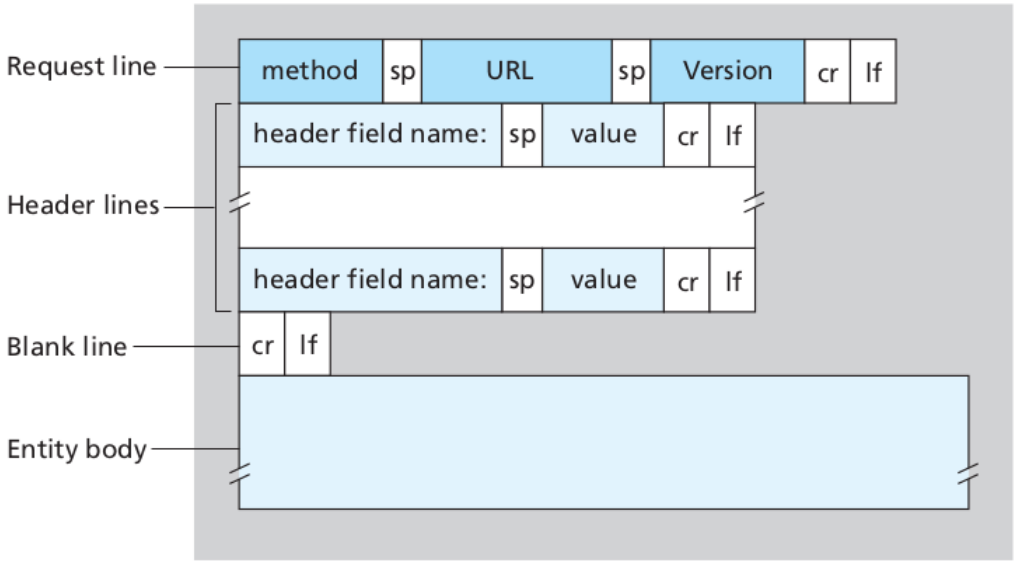


Figure 2.2: General format of an HTTP request message

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

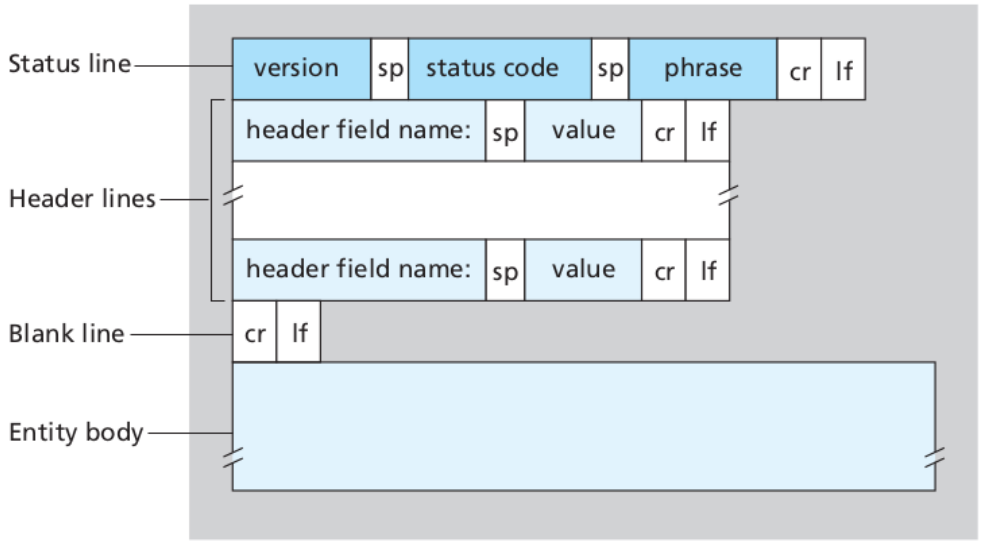


Figure 2.3: General format of an HTTP response message

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.

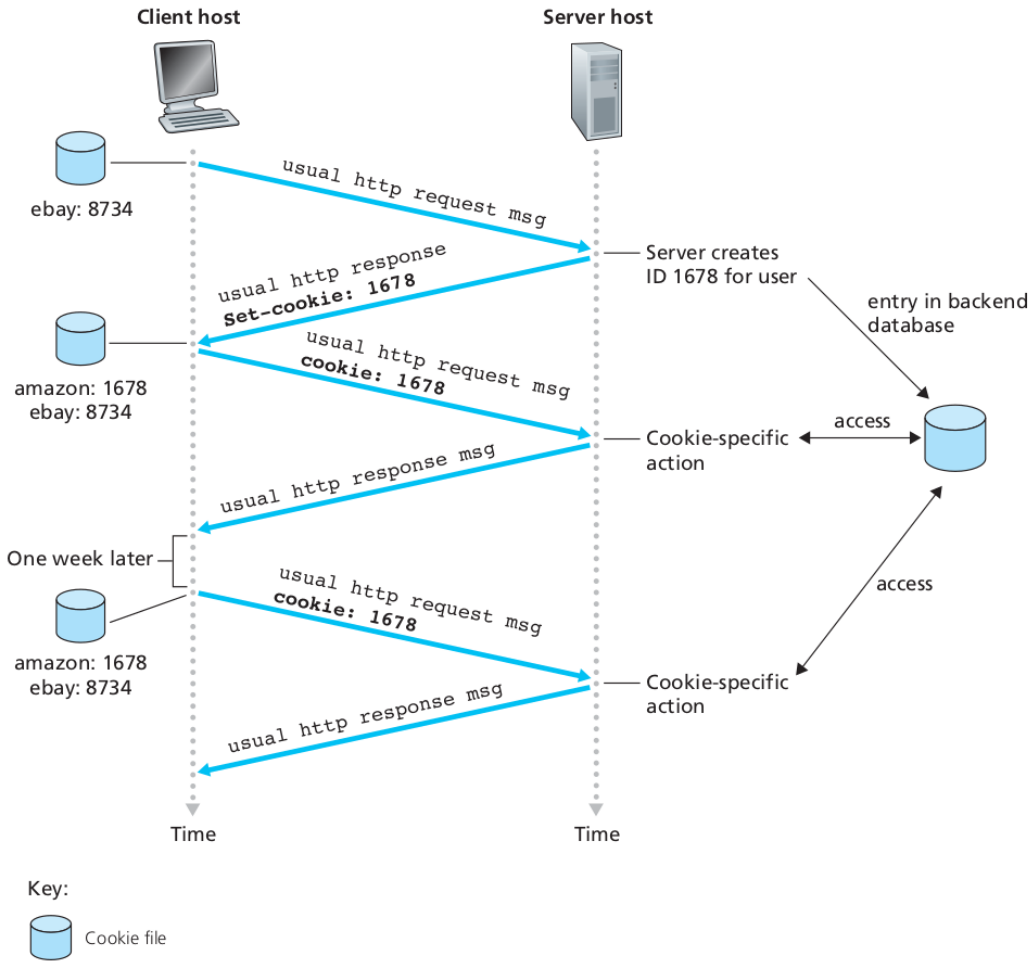


Figure 2.4: Use cookie to identify users

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.

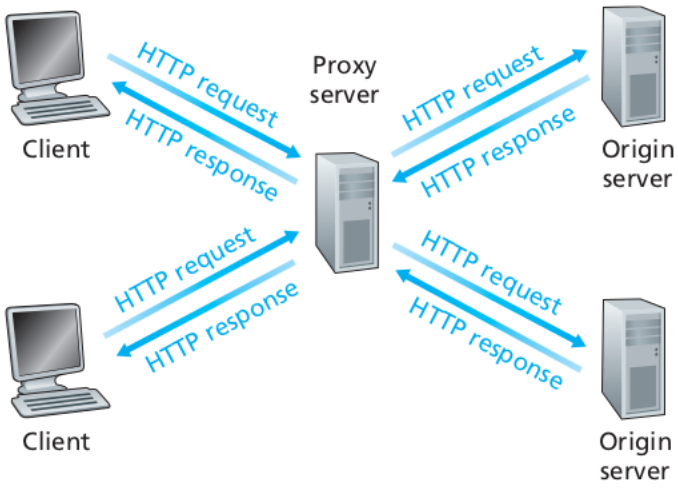


Figure 2.5: Web Caching

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.

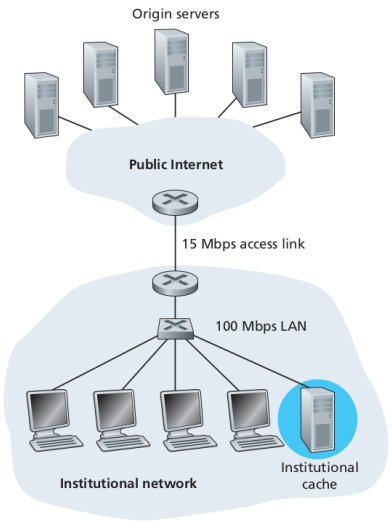


Figure 2.6: Adding a cache to the institutional network

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 ne