2.2 The Web and HTTP

world wide web:Tim Berners-Lee

2.2.1 Overview of HTTP

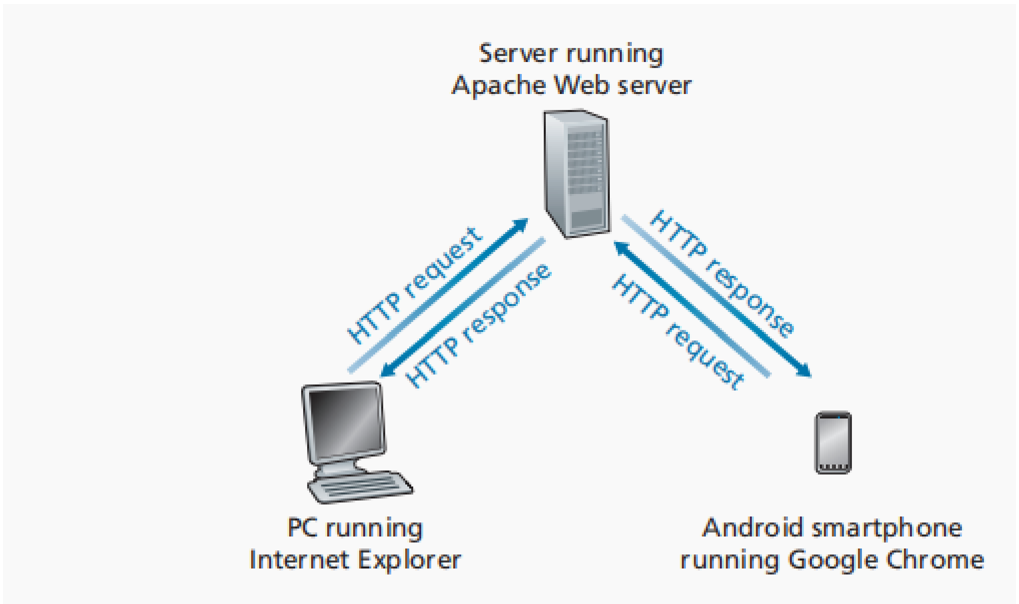
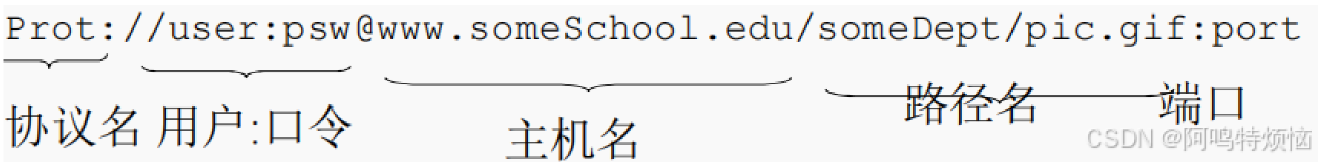


Figure 2.6 ♦ HTTP request-response behavior

- web page consists of objects, each of which can be stored on different Web servers
- web页由一些对象组成
- object can be HTML file, JPEG image, Java applet, audio file,...
- web page consists of base HTML-file which includes several referenced objects, each addressable by a URL, e.g.,
- web页含有一个基本的HTML文件，该基本HTML文件又包含若干对象的引用（链接）
- 通过URL(Uniform Resouce Locator)对每个对象进行引用
  - 访问协议，用户名，口令字，端口等
- URL（统一资源定位器）格式：



HTTP: hypertext transfer protocol超文本传输协议

- Web's application-layer protocol
- client/server model:

    - client: browser that requests, receives, (using HTTP protocol) and “displays” Web objects
    - server: Web server sends (using HTTP protocol) objects in response to requests
- HTTP uses TCP:
    - client initiates TCP connection (creates socket) to server, port 80
    - server accepts TCP connection from client
    - HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
    - TCP connection closed
- HTTP is “stateless”
    - server maintains no information about past client requests
- HTTP 1.0:RFC 1945
- HTTP 1.1:RFC 2068

protocols that maintain “state” are complex!

- past history (state) must be maintained
- if server/client crashes, their views of “state” may be inconsistent, must be reconciled

无状态的服务器能够支持更多的客户端

2.2.2 HTTP Connections:two types

Non-persistent HTTP

- 1.TCP connection opened
- 2.at most one object sent over TCP connection
- 3.TCP connection closed
  downloading multiple objects required multiple connections
  HTTP/1.0使用非持久连接

Persistent HTTP

- TCP connection opened to a server
- multiple objects can be sent over single TCP connection between client, and that server
- TCP connection closed
  HTTP/1.1默认使用持久连接

HTTP with Non-Persistent Connections

Non-persistent HTTP: example

User enters URL:www.someSchool.edu/someDepartment/home.index
(containing text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at
www.someSchool.edu on port 80
1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80 "accepts" connection, notifying client
2.HTTP client sends HTTP request message (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index
3.HTTP server receives request message, forms response message containing requested object, and sends message into its socket
4.HTTP server closes TCP connection.
5.HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects
6.Steps 1-5 repeated for each of 10 jpeg objects

Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back
往返时间RTT（round-trip time）：一个小的packet从客户端到服务器，再回到客户端的时间（传输时间忽略）

HTTP response time (per object):

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- object/file transmission time

Non-persistent HTTP response time = 2RTT+ file transmission time

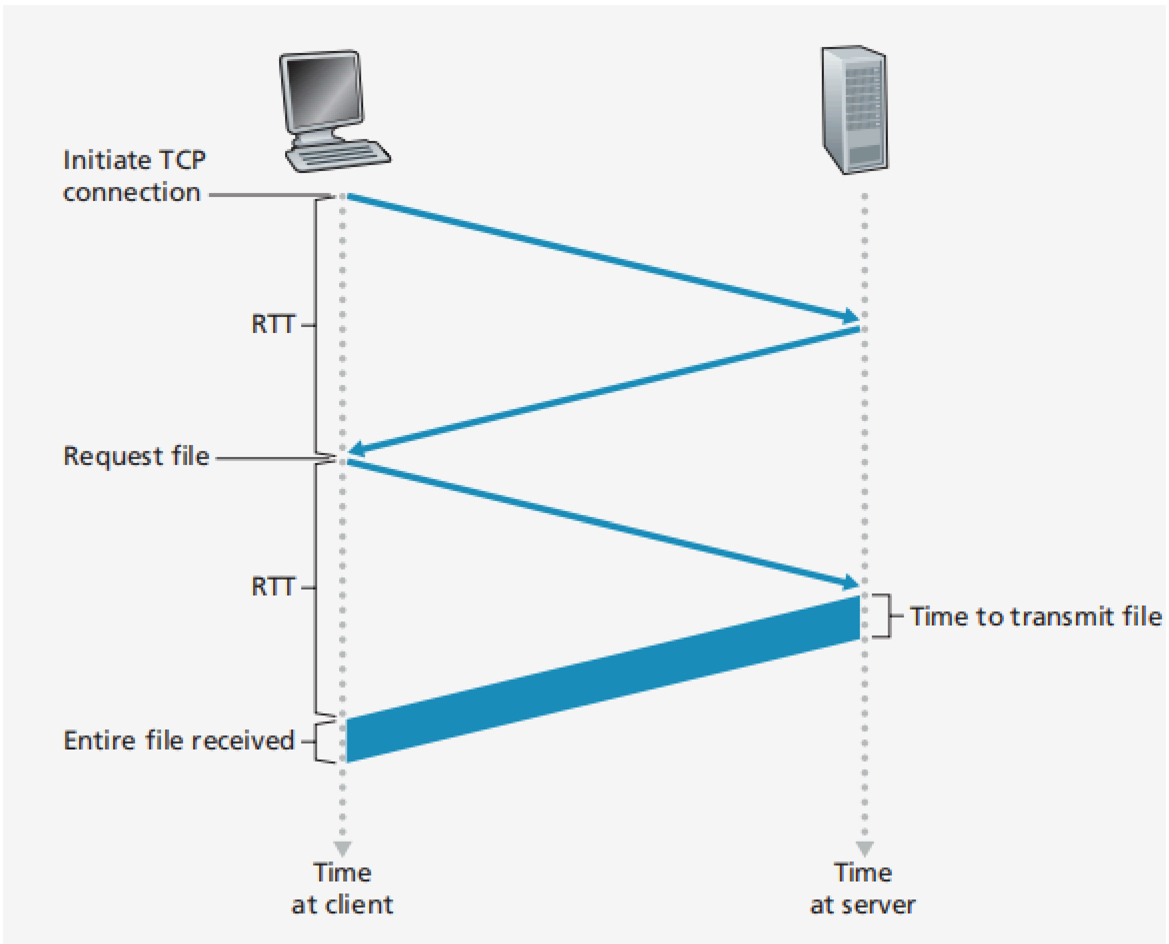


Figure 2.7 ♦ Back-of-the-envelope calculation for the time needed to request and receive an HTML file

HTTP with Persistent Connections

Non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for each TCP connection
- browsers often open multiple parallel TCP connections to fetch referenced objects in parallel

Persistent HTTP (HTTP1.1):

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects (cutting response time in half)对于所有引用的对象，只需一个RTT(将响应时间缩短一半)

非流水方式(without pipelining)的持久HTTP

- 客户端只能在收到前一个响应后才能发出新的请求
- 每个引用对象花费一个RTT

流水方式(pipelining)的持久HTTP

- HTTP/1.1的默认模式
- 客户端遇到一个引用对象就立即产生一个请求
- 所有引用（小）对象只花费一个RTT是可能的

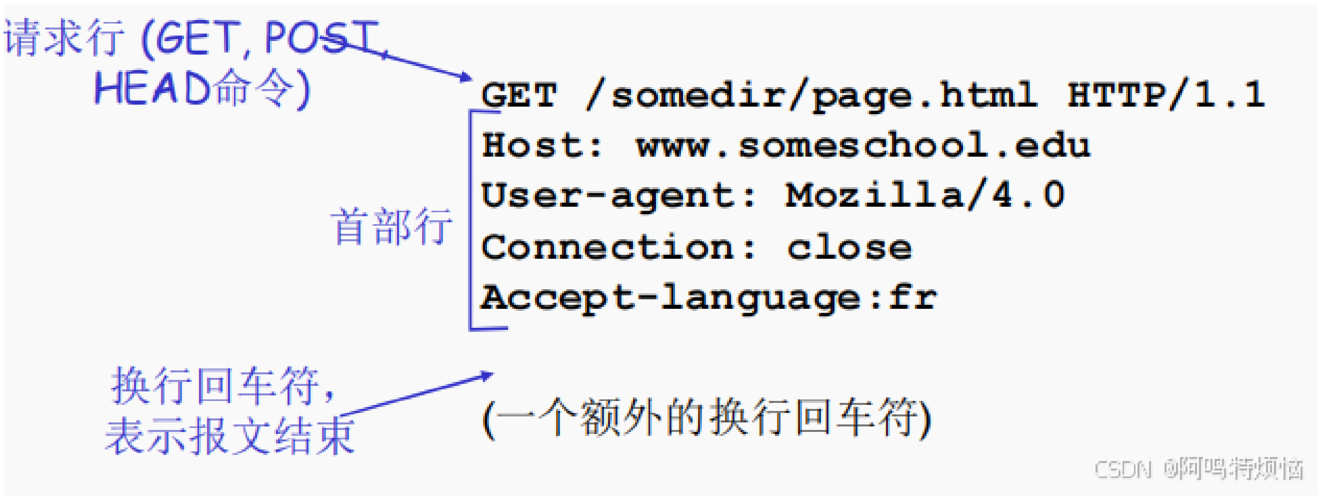
2.2.3 HTTP Message Format

HTTP Request Message

two types of HTTP messages: request, response

- HTTP request message:
- ASCII (human-readable format)

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



```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

request line (GET, POST,HEAD commands)
carriage return character
line-feed character
carriage return, line feed at start of line indicates end of header lines

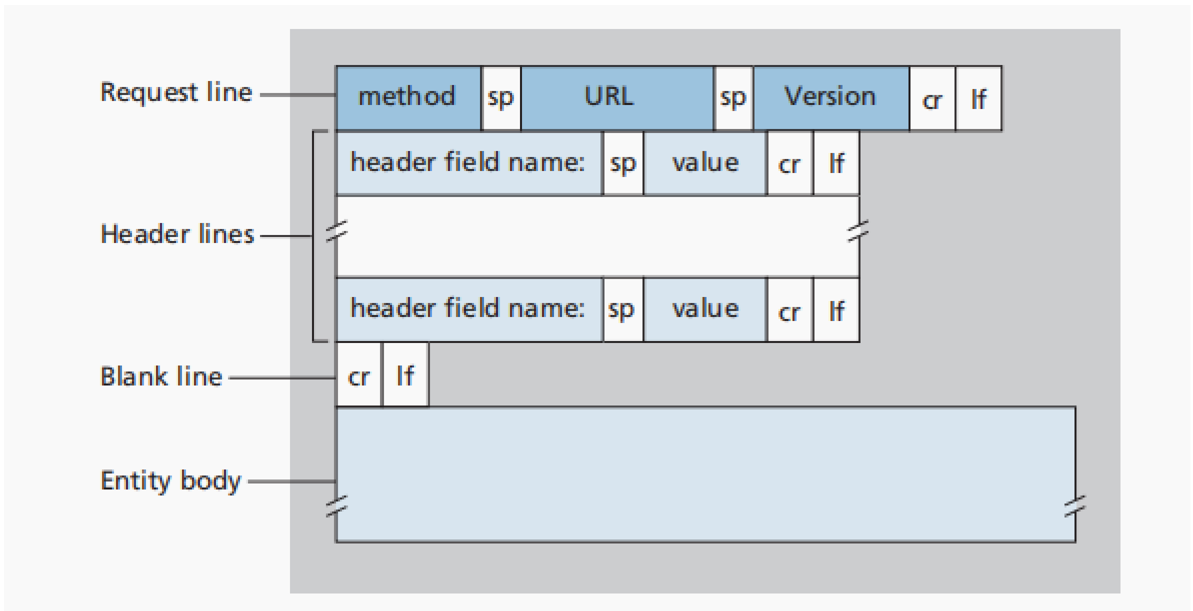


Figure 2.8 ♦ General format of an HTTP request message

POST method:

- web page often includes form input网页经常填写表格
- user input sent from client to server in entity body of HTTP POST request message
  GET method (for sending data to server):
- include user data in URL field of HTTP GET request message (following a '?'):
  www.somesite.com/animalsearch?monkeys&banana
  HEAD method:
- requests headers (only) that would be returned if specified URL were requested with an HTTP GET method.
- 请Server不要将所请求的对象放入响应消息中
  PUT method:
- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body of POST HTTP request message
- 将消息体中的文件上传到URL字段所指定的路径
  DELETE
- 删除URL字段所指定的文件

HTTP/1.0

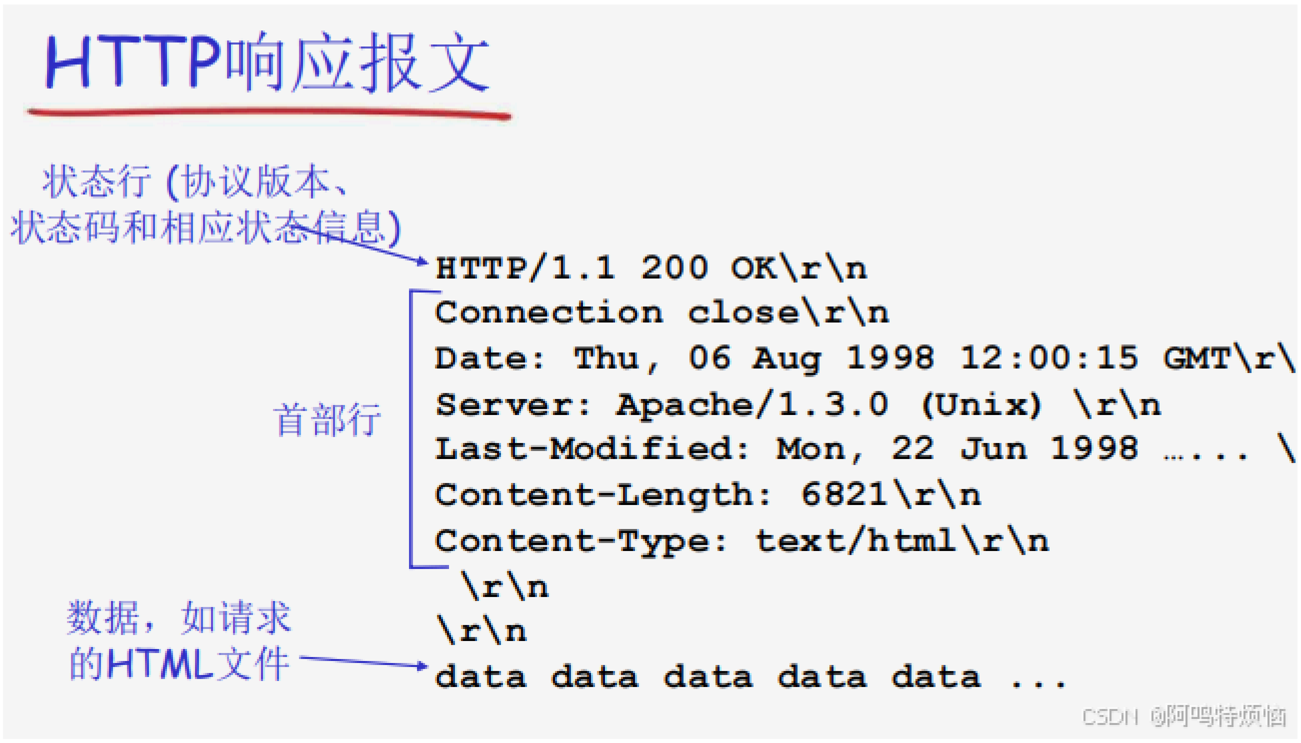
- GET
- POST
- HEAD
  HTTP/1.1
- GET
- POST

- HEAD
- PUT
- DELETE

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 ...)

status line (protocol status code status phrase)



```
HTTP/1.1 200 OK\r\n
Connection close\r\n
Date: Thu, 06 Aug 1998 12:00:15 GMT\r\
Server: Apache/1.3.0 (Unix) \r\n
Last-Modified: Mon, 22 Jun 1998 ...... \
Content-Length: 6821\r\n
Content-Type: text/html\r\n
 \r\n
\r\n
data data data data data ...
```

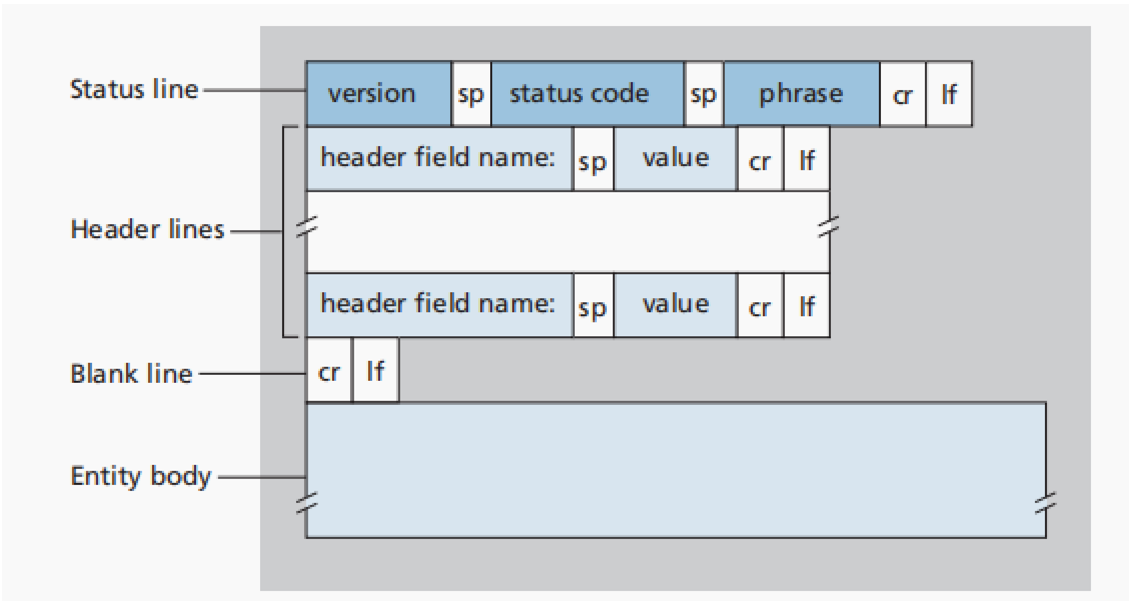


Figure 2.9 ♦ General format of an HTTP response message

HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:
  - 200 OK
    - request succeeded, requested object later in this message
  - 301 Moved Permanently
    - requested object moved, new location specified later in this message (in Location: field)
    - 客户端软件自动用新的URL去获取对象
  - 400 Bad Request
    - request msg not understood by server
  - 404 Not Found
    - requested document not found on this server
  - 505 HTTP Version Not Supported

2.2.4 User-Server Interaction: Cookies

Maintaining user/server state: cookies

Recall: HTTP GET/response interaction is stateless

- no notion of multi-step exchanges of HTTP messages to complete a Web "transaction"没有HTTP消息的多步交换以完成Web"事务"的概念。
  - no need for client/server to track "state" of multi-step exchange不需要客户端/服务器跟踪多步交换的"状态"
  - all HTTP requests are independent of each other所有的HTTP请求相互独立

  - no need for client/server to "recover" from a partially-completed-but-never-completely-completed transaction客户机/服务器不需要从部分完成但从未完全完成的事务中"恢复"

what happens if network connection or client crashes at t' ?

a stateful protocol: client makes two changes to X, or none at all

Web sites and client browser use cookies to maintain some state between transactions

four components:

1. cookie header line of HTTP response message
2. cookie header line in next HTTP request message
3. cookie file kept on user's host, managed by user's browser
4. back-end database at Web site

Example:

- Susan uses browser on laptop, visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID (aka "cookie")
  - entry in backend database for ID
- subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to "identify" Susan

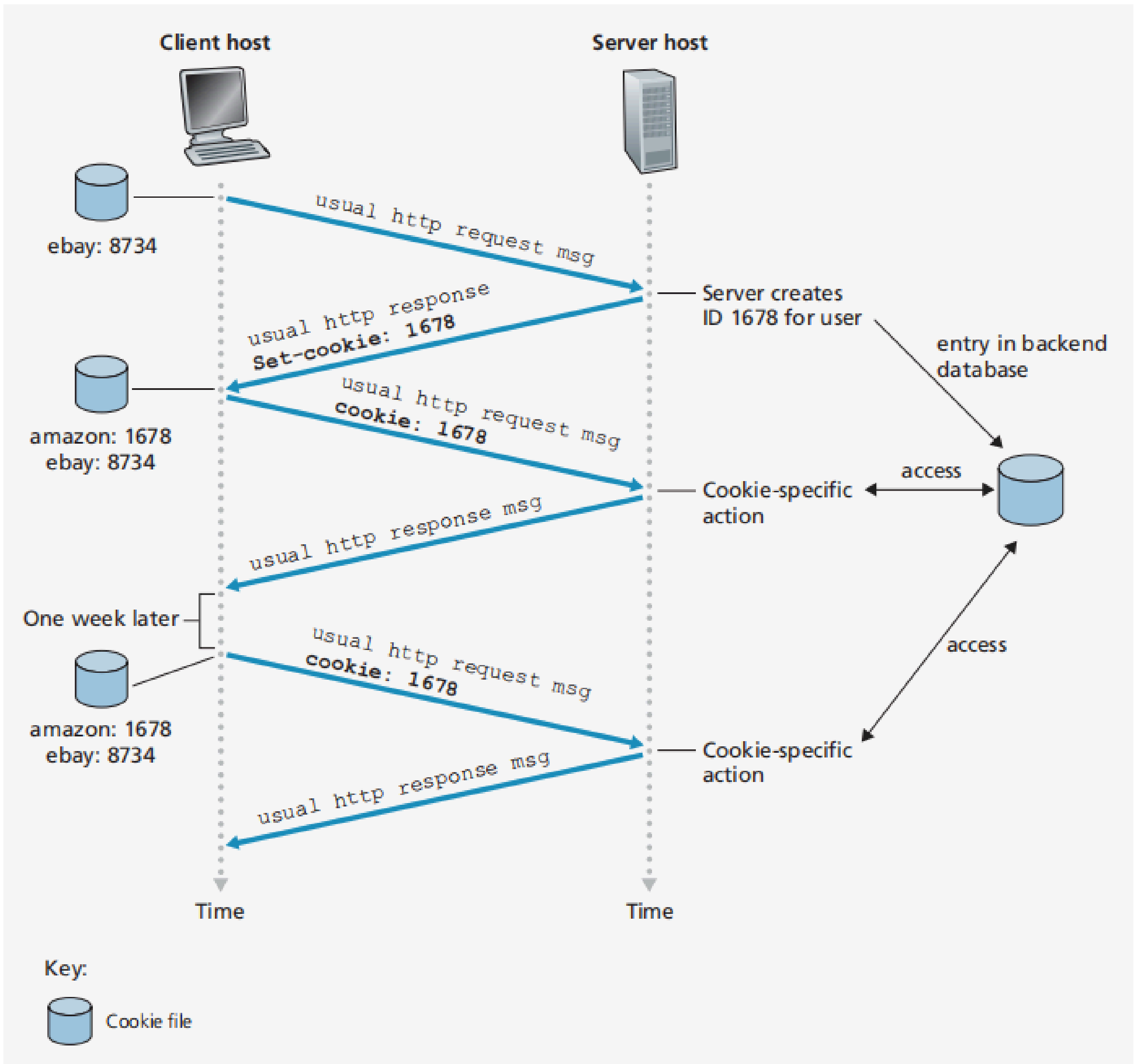


Figure 2.10 ♦ Keeping user state with cookies

HTTP cookies: comments

What cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)
  Challenge: How to keep state?
- at protocol endpoints: maintain state at sender/receiver over multiple transactions
- in messages: cookies in HTTP messages carry state
  cookies and privacy:
- cookies permit sites to learn a lot about you on their site.
- third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites第三方持久cookie(跟踪cookie)允许跨多

个网站跟踪公共身份(cookie值)

2.2.5 Web Caching(proxy server)

Goal: satisfy client requests without involving origin server

- user configures browser to point to a (local) Web cache
- browser sends all HTTP requests to cache
  - if object in cache: cache returns object to client
  - else cache requests object from origin server, caches received object, then returns object to client

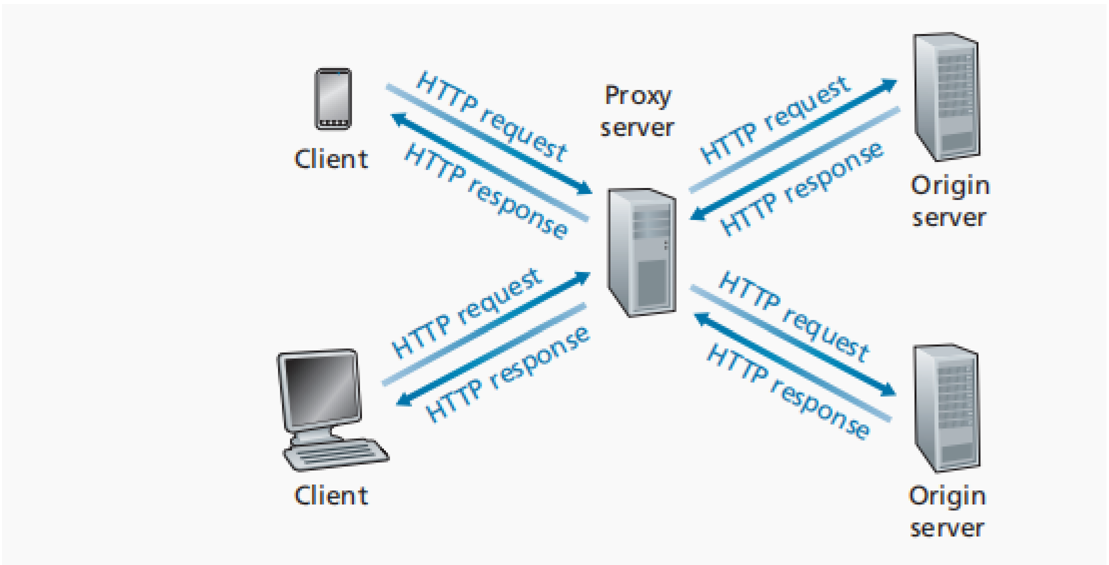


Figure 2.11 ♦ Clients requesting objects through a Web cache

Web caches (aka proxy servers)

- Web cache acts as both client and server
  - server for original requesting client
  - client to origin server
- server tells cache about object's allowable caching in response header:
  - Cache-Control: max-age=< seconds>
  - Cache-Control: no-cache

Why Web caching?

- reduce response time for client request
  - cache is closer to client
- reduce traffic on an institution's access link
- Internet is dense with caches
  - enables "poor" content providers to more effectively deliver content

Caching example
Scenario:

- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
    - avg data rate to browsers: 1.50 Mbps

Performance:

- access link utilization = .97
- LAN utilization: .0015
- end-end delay = Internet delay + access link delay + LAN delay = 2 sec + minutes + usecs
  problem: large queueing delays at high utilization!

Option 1: buy a faster access link
Cost: faster access link (expensive!)

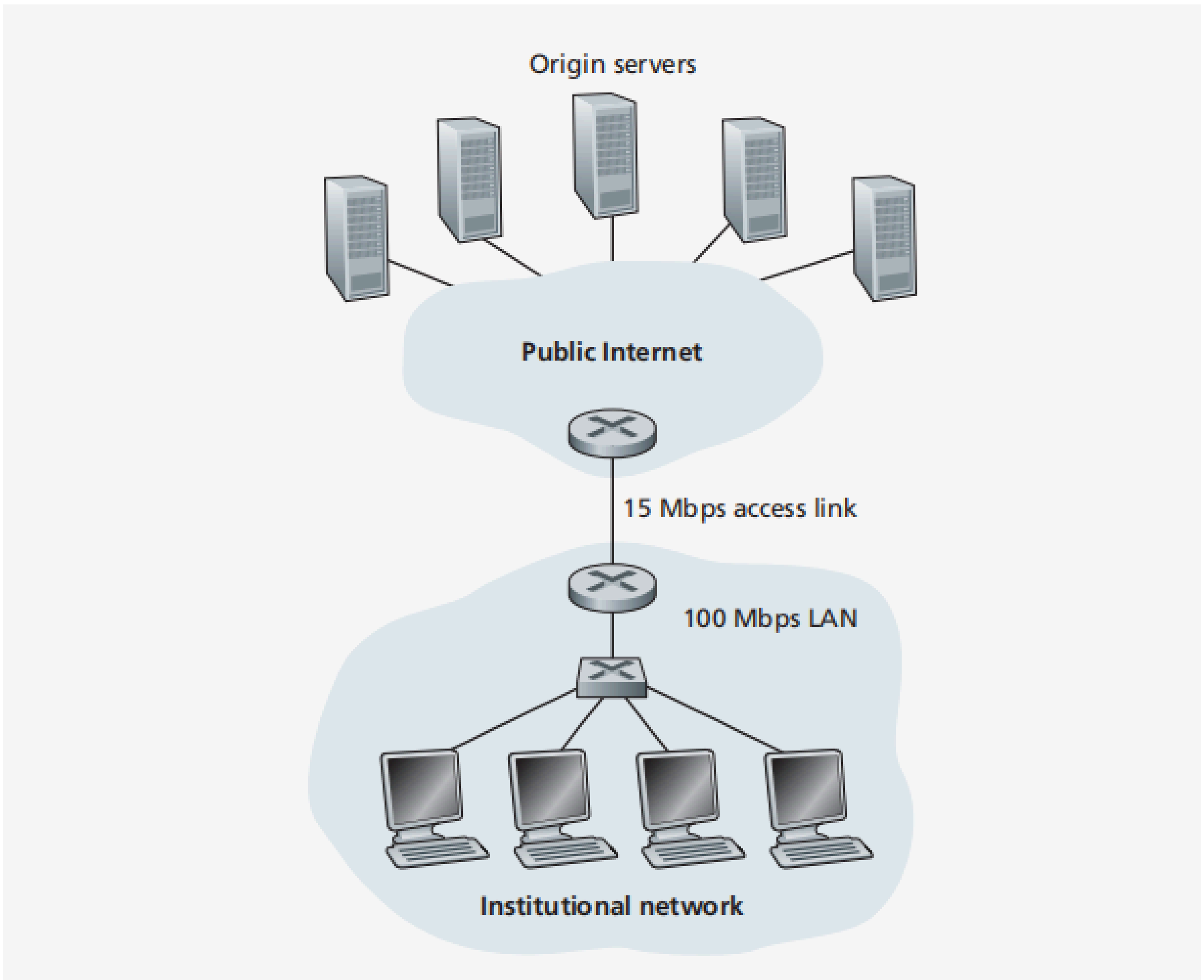


Figure 2.12 ♦ Bottleneck between an institutional network and the Internet

Option 2: install a web cache
How to compute link utilization, delay?

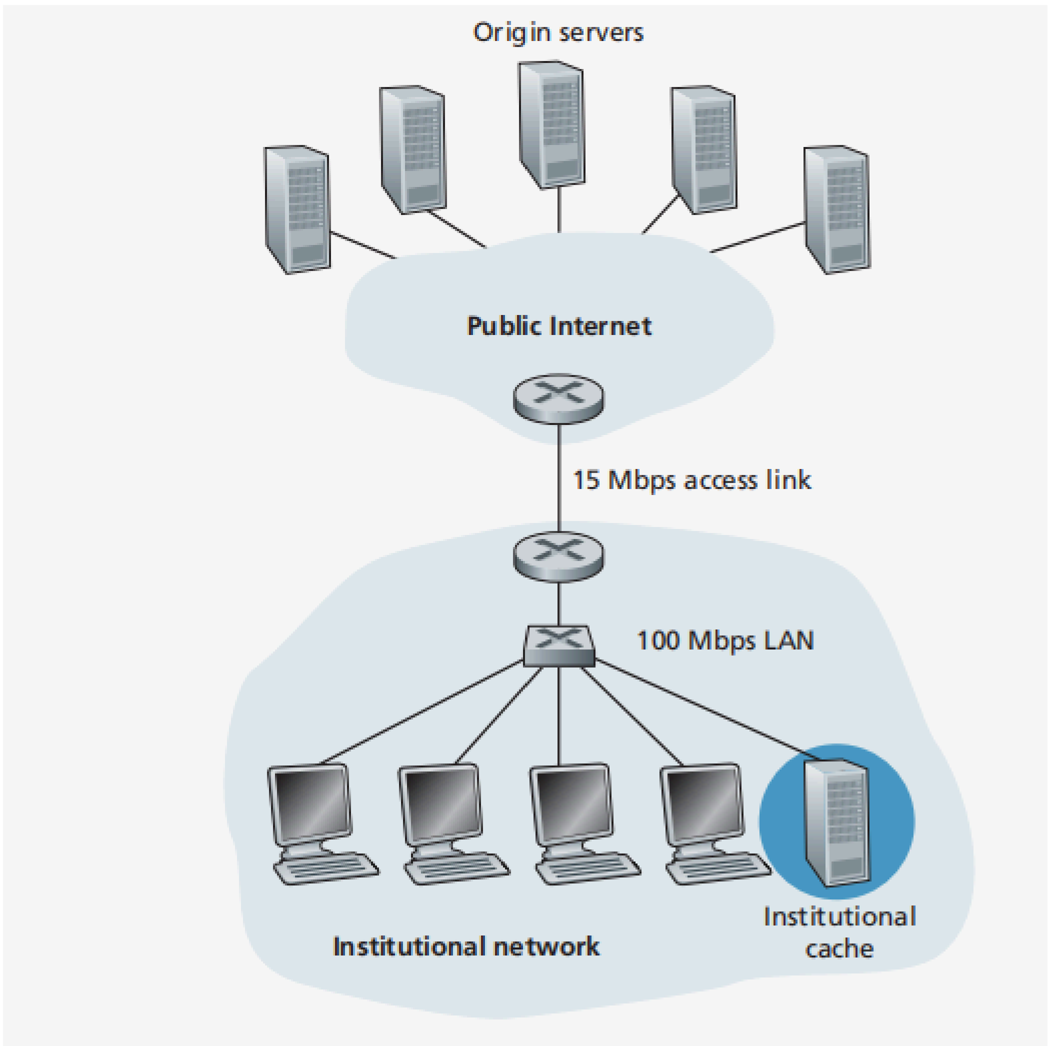
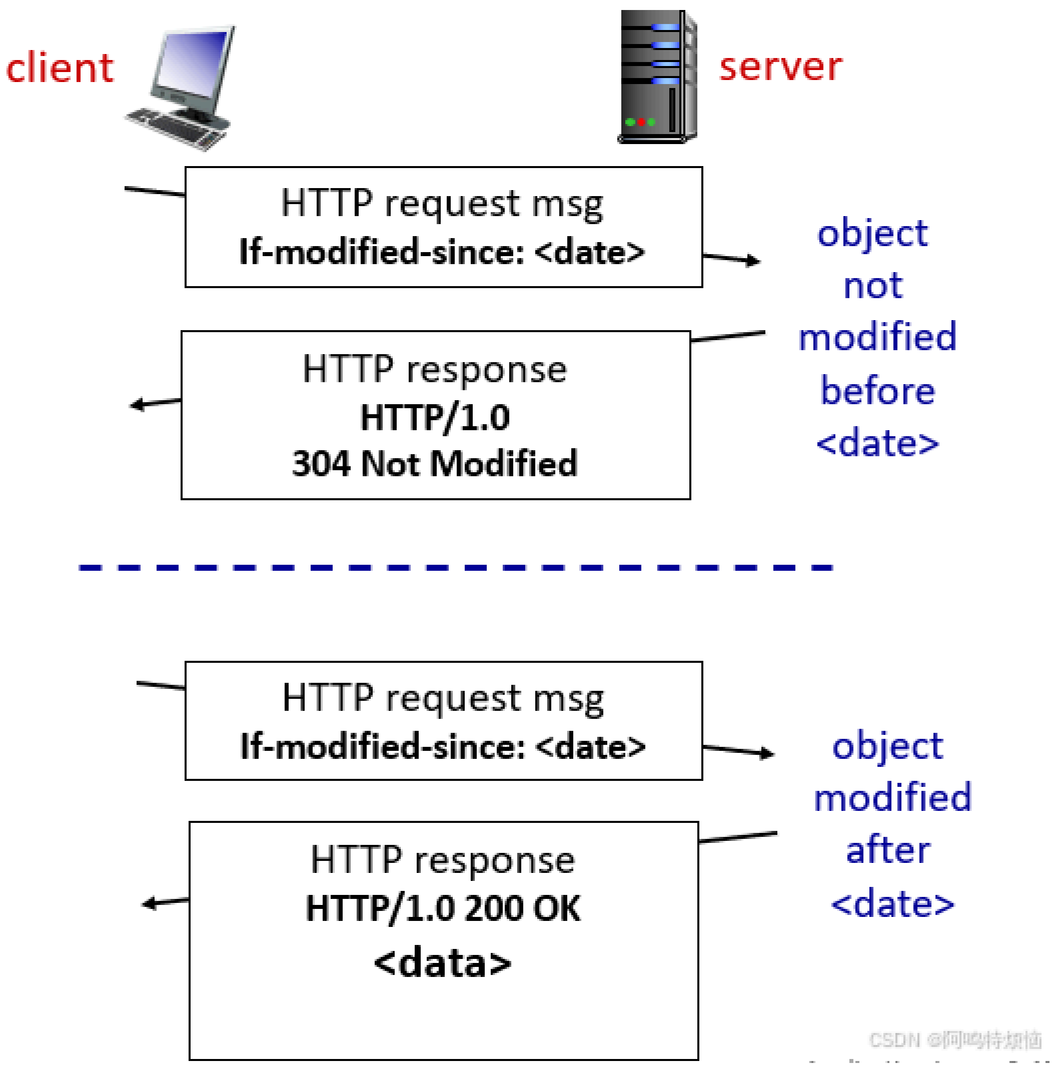


Figure 2.13 ♦ Adding a cache to the institutional network

The Conditional GET

Browser caching: Conditional GET

- Goal: don't send object if browser has up-to-date cached version
  - no object transmission delay (or use of network resources)
- client: specify date of browser-cached copy in HTTP request
  - If-modified-since: < date>
- server: response contains no object if browser-cached copy is up-to-date:
  - HTTP/1.0 304 Not Modified


client
server
HTTP request msg
If-modified-since: <date>
object not modified before <date>
HTTP response
HTTP/1.0
304 Not Modified
HTTP request msg
If-modified-since: <date>
object modified after <date>
HTTP response
HTTP/1.0 200 OK
<data>