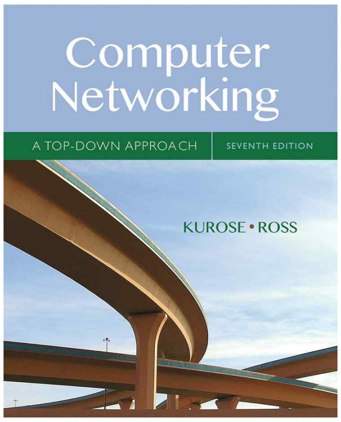
Chapter 2
Application Layer



Seongwook Youn
Department of Software
Korea National University of Transportation

The slides are adapted from the publisher's material

Computer Networking: A Top Down Approach
7th edition
Jim Kurose, Keith Ross
Addison-Wesley
April 2016

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 electronic mail
- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

네트워크 어플리케이션의 원리를 알아봅니다.
[1] 앱 아키텍처
[2] 앱 요구사항

Chapter 2: application layer

목표: network application protocols의 개념과 실행을 이해하자

our goals:

개념의, 구상의

이행, 실행

- Conceptual and implementation aspects of network application protocols
  - transport-layer service models
- Architectural paradigms

  전형적인
  - client-server paradigm
  - peer-to-peer paradigm
- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS

Some network apps

- e-mail
- web
- text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)

Hulu는 안 유명함

- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- …
- …

<요약>
[1] 다른 end systems가
[2] 네트워크 위에서 작동 합니다.
*네트워크 코어 장치는 필요 없습니다.

Message

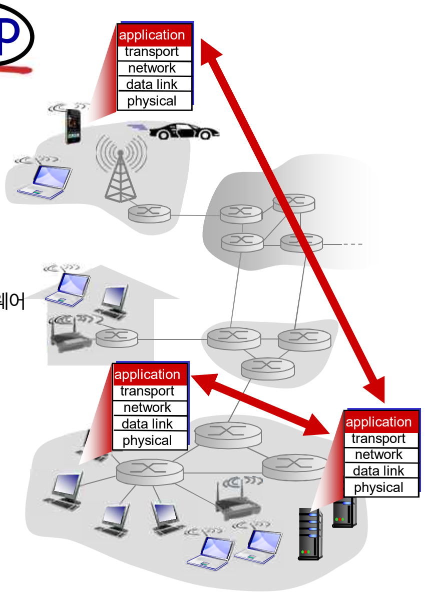
Creating a network app

write programs that:

- run on (different) *end systems* 계속되다
- communicate over network
- e.g., web server software communicates with browser software 브라우저와 의사소통하는 웹서버 소프트웨어
  1) web server <-> web server host
  2) user <-> host

no need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation



Communication for a network application takes place
Between end systems at the application layer

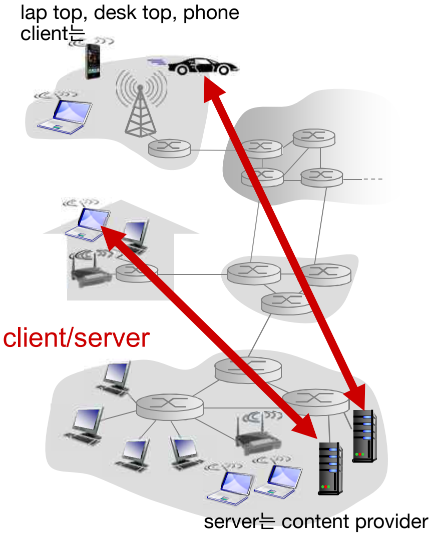
in the web application there are two distinguish program that communicate with each other

Application architectures

possible structure of applications:

- client-server 흔하게 일반적으로 사용하는 경우인데, 1) 웹사이트에 접속 2) 파일을 다운로드 받을 때 나 - client, 정보 주는 곳 - server
- peer-to-peer (P2P) --> 요즘 흔함
  다양한 peer들이 서로 정보를 1) 보내주거나 2) 받는 경우
  - Each host participates in the file-sharing community
  - The Programs in the various hosts may be similar or identical

Client-server architecture



server:

- always-on host
- 영구적인
  permanent IP address
- data centers for scaling

client는 항상 들어올 필요가 없습니다.
본인이 필요할 때만 컴퓨터를 키고 서버에 접속하면 됩니다.

clients:

- communicate with server
- 쉬엄 쉬엄
  may be intermittently connected
- 동적인, 활발한
  may have dynamic IP addresses
- do not communicate directly with each other

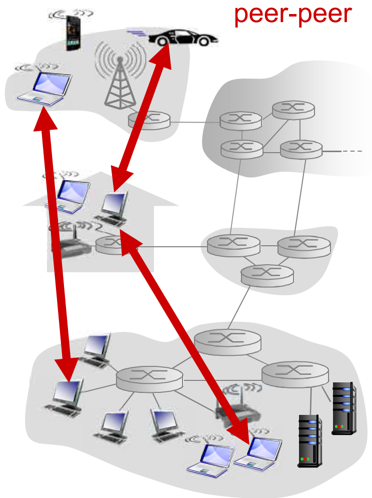
client는 필요할 때 서버를 통해 다운로드 받습니다.
client 끼리는 통신하지 않습니다.

새로운 peer은 새로운 서비스 요구/능력을 가져옵니다.

P2P architecture

- *no* always-on server
- 임의적인
  arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - 확장성
    self scalability – new peers bring new service capacity, as well as new service demands
- 쉬엄 쉬엄
  peers are intermittently connected and change IP addresses
  - complex management

Internet protocol Address
컴퓨터 네트워크 장치들이
서로를 인식하고 통신하기 위해
사용하는 특수한 번호



client-server는 server 쪽에만 엄청난 접속이 있지만
p2p는 개개인이 들어왔다 나갔다 하면서 정보를 주고 받기 때문에 엄청나게 복잡합니다.

호스트 개수에 따른 통신
[1] 하나의 컴퓨터 안 에서 두 개의 프로세스들이 inter-process communication을 통해 통신합니다.
[2] 두 개의 컴퓨터끼리 메시지를 교환을 통해 통신합니다.

프로세스의 개념을 알아 두세요 네트워크 어플리케이션은 네트워크를 통해 서로 다른 메시지가 교환 되는 프로세스들로 구성된다.

Processes communicating

호스트(컴퓨터) 내에서 실행 중인 프로그램

process: program running within a host

- within same host, two processes communicate using inter-process communication (defined by OS)
- processes in different hosts communicate by exchanging messages
- A network application consists of processes that send messages to each other over a network

clients, servers

통신을 시작시키는 프로세스

client process: process that initiates communication

연결을 기다리는 프로세스 --> 항상 대기

server process: process that waits to be contacted

- aside: applications with P2P architectures have client processes & server processes

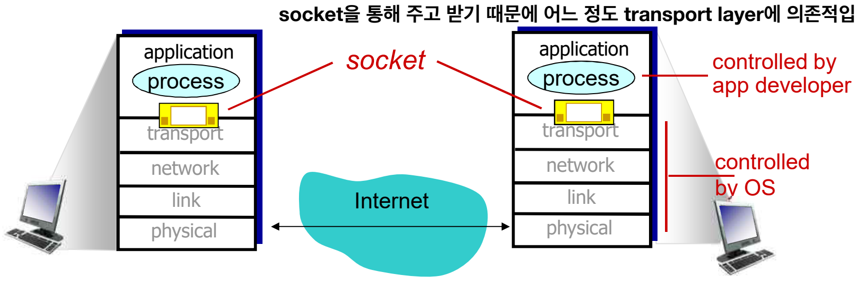
웹 어플리케이션에서 client browser 프로세스 -> web server 프로세스에서 메시지를 교환한다.
P2P 파일 쉐어링 시스템에서는 파일이 한 피어에 -> 다른 피어 프로세스로 파일이 전송된다.

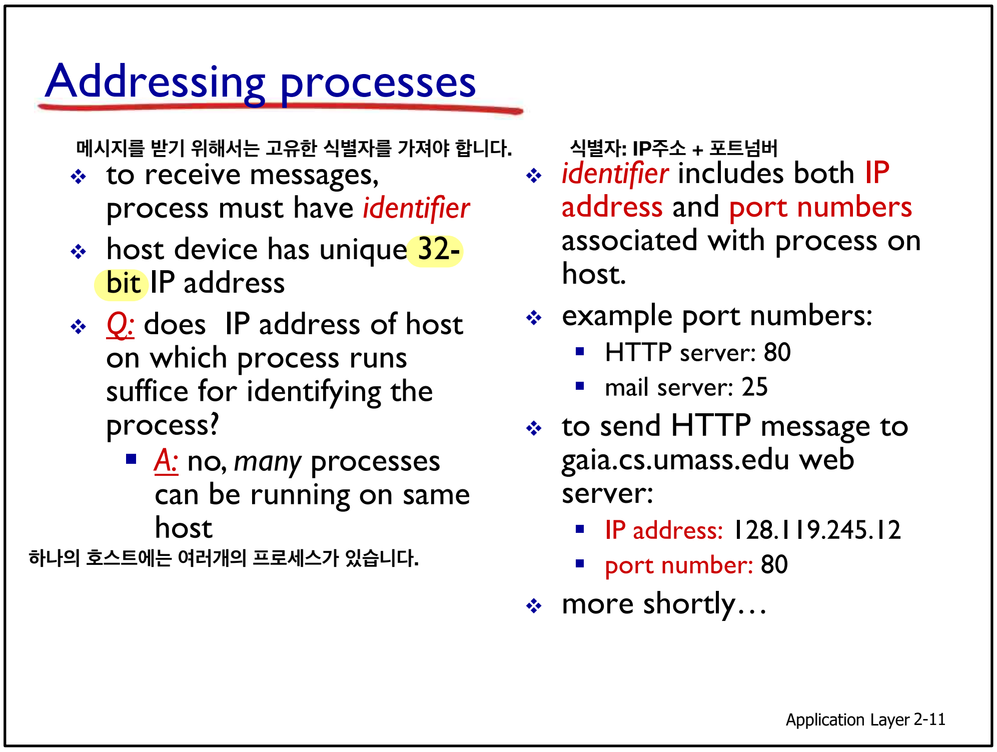
Sockets

프로세스는 메시지를 소켓을 통해 주고 받는다. --> 문과 같은 역할

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process

프로세스 자체는 application developer에서 관리가 되지만
메시지를 주고 받을 때는 application layer만 관리하는 것이 아닙니다.
socket을 통해 주고 받기 때문에 어느 정도 transport layer에 의존적입니다.





Addressing processes

메시지를 받기 위해서는 고유한 식별자를 가져야 합니다.

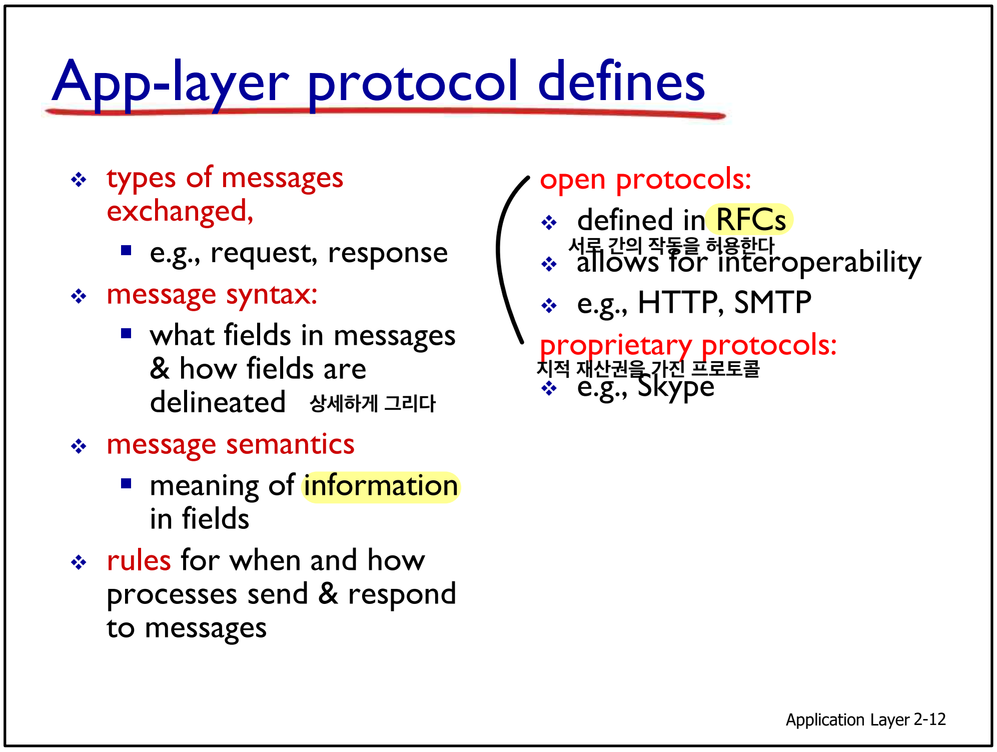
- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?
  - *A:* no, *many* processes can be running on same host

하나의 호스트에는 여러개의 프로세스가 있습니다.

식별자: IP주소 + 포트넘버

- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80
- more shortly…

Application Layer 2-11



App-layer protocol defines

- types of messages exchanged,
  - e.g., request, response
- message syntax:
  - what fields in messages & how fields are delineated 상세하게 그리다
- message semantics
  - meaning of information in fields
- rules for when and how processes send & respond to messages

open protocols:

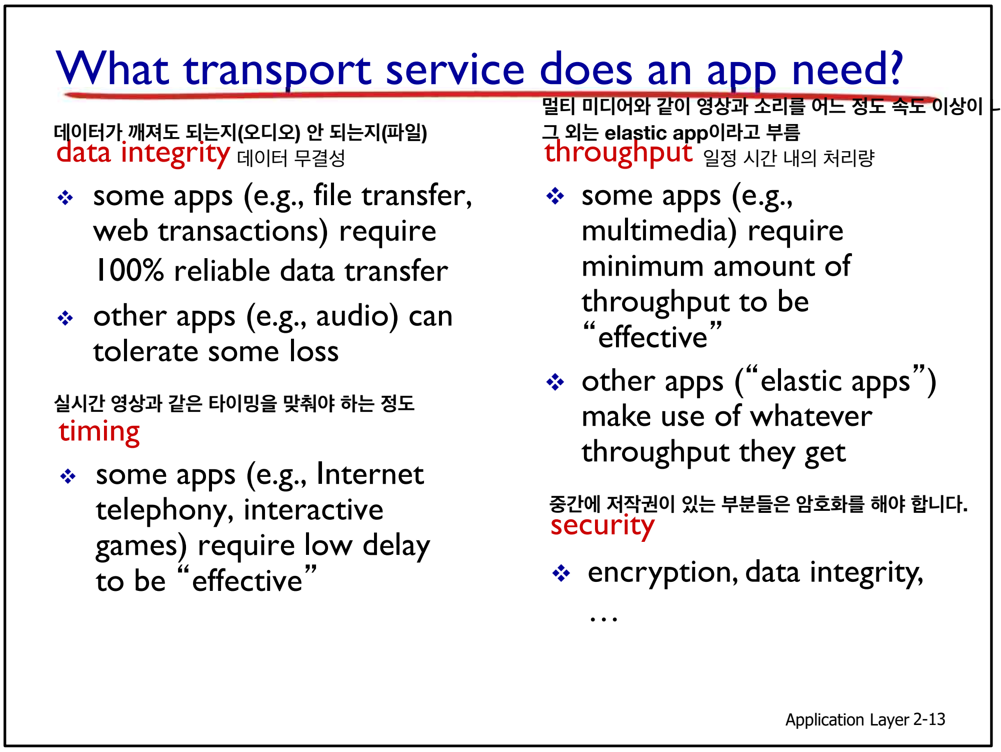
- defined in RFCs
- 서로 간의 작동을 허용한다
  allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

지적 재산권을 가진 프로토콜

- e.g., Skype

Application Layer 2-12



What transport service does an app need?

데이터가 깨져도 되는지(오디오) 안 되는지(파일)

data integrity 데이터 무결성

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

실시간 영상과 같은 타이밍을 맞춰야 하는 정도

timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

멀티 미디어와 같이 영상과 소리를 어느 정도 속도 이상이 나와야 볼 수 있는 것들

그 외는 elastic app이라고 부름

throughput 일정 시간 내의 처리량

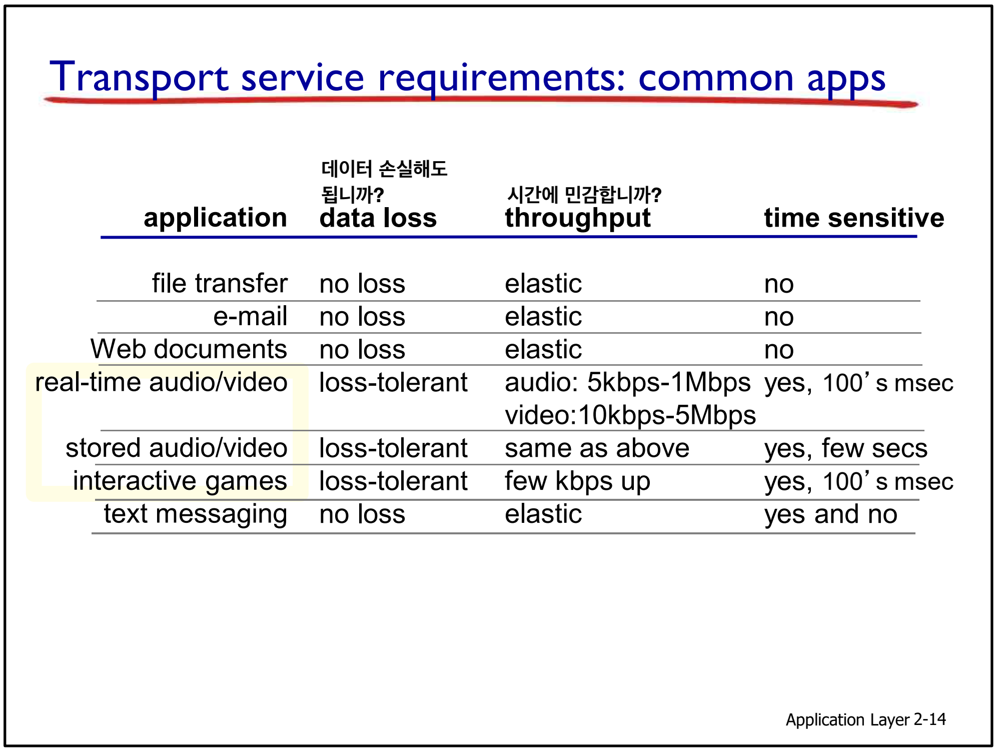
- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- other apps ("elastic apps") make use of whatever throughput they get

중간에 저작권이 있는 부분들은 암호화를 해야 합니다.

security

- encryption, data integrity, …

Application Layer 2-13



Transport service requirements: common apps

application	데이터 손실해도 됩니까? data loss	시간에 민감합니까? throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
text messaging	no loss	elastic	yes and no

Application Layer 2-14

이 둘의 차이를 이해해야 합니다.

Internet transport protocols services

TCP service:

- 믿을 수 있는 *reliable transport* between sending and receiving process
- 받는 쪽이 감당가능한 만큼만 보낸다. *flow control:* sender won't overwhelm receiver
- 조절하다, 낮추다 *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security
- *connection-oriented:* setup required between client and server processes

일일이 신경쓰는 타입
--> 대부분의 경우에서 사용함
loss된 것을 다시 보냅니다.

UDP service:

- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup,

보내놓고 신경 안 쓰는 타입
--> live streaming, 사용자가 많지 않은 경우 유리함

Application Layer 2-15

Internet apps: application, transport protocols

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

real-time 과 같은 경우 UDP를 사용 합니다.
사용자가 많지 않을 수록 유리합니다.

Application Layer 2-16

Securing TCP

기본적으로는 암호화가 없습니다.

TCP & UDP

- no encryption
- cleartext passwds sent into socket traverse 가로지르다 Internet in cleartext

SSL

Secure sockets layer
Web site <-> 브라우저 사이에
전송된 데이터를 암호화

- provides encrypted TCP connection
- data integrity
- end-point authentication

SSL is at app layer

- Apps use SSL libraries, which "talk" to TCP

SSL socket API

- cleartext passwds sent into socket traverse Internet encrypted

하도 문제가 생기니깐!! SSL를 사용하기 시작했습니다.
보내는 쪽 받는 쪽에서 인증이 필요합니다.
[0] 보내는 쪽에서 encrypted 를 하여 보냅니다.
[1] encrypted 된 채로 TCP에서 움직입니다.
[2] 받는 쪽에서 decrypted를 하여 받습니다.

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

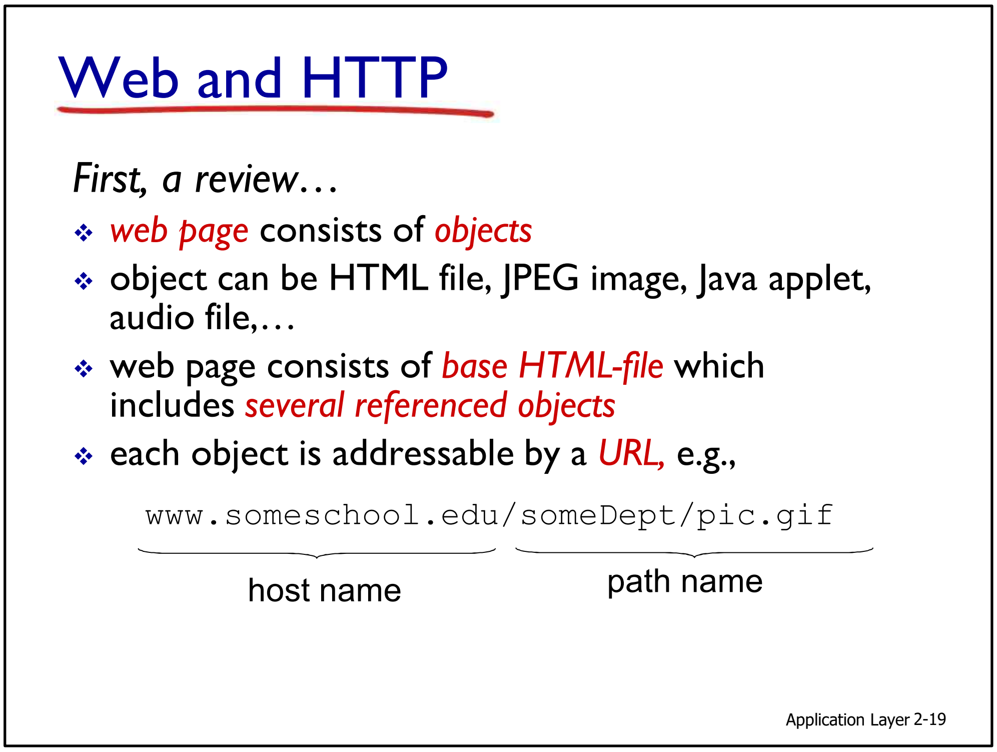
2.2 Web and HTTP

2.3 electronic mail
- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)



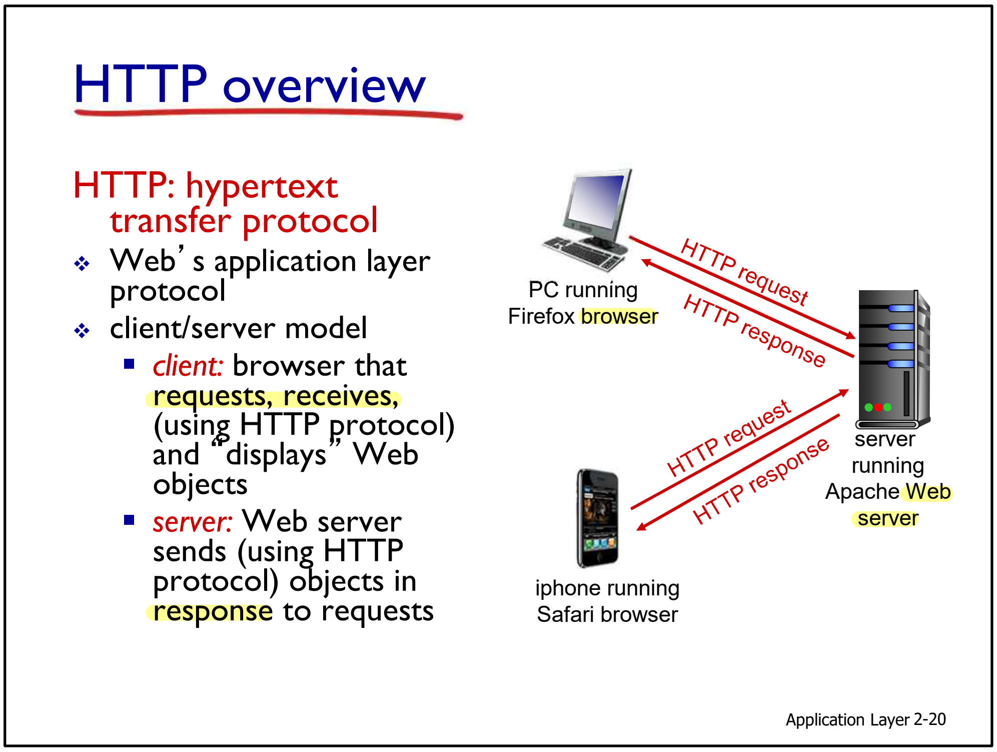
Web and HTTP

First, a review…

- *web page* consists of *objects*
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL,* e.g.,

$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$

Application Layer 2-19

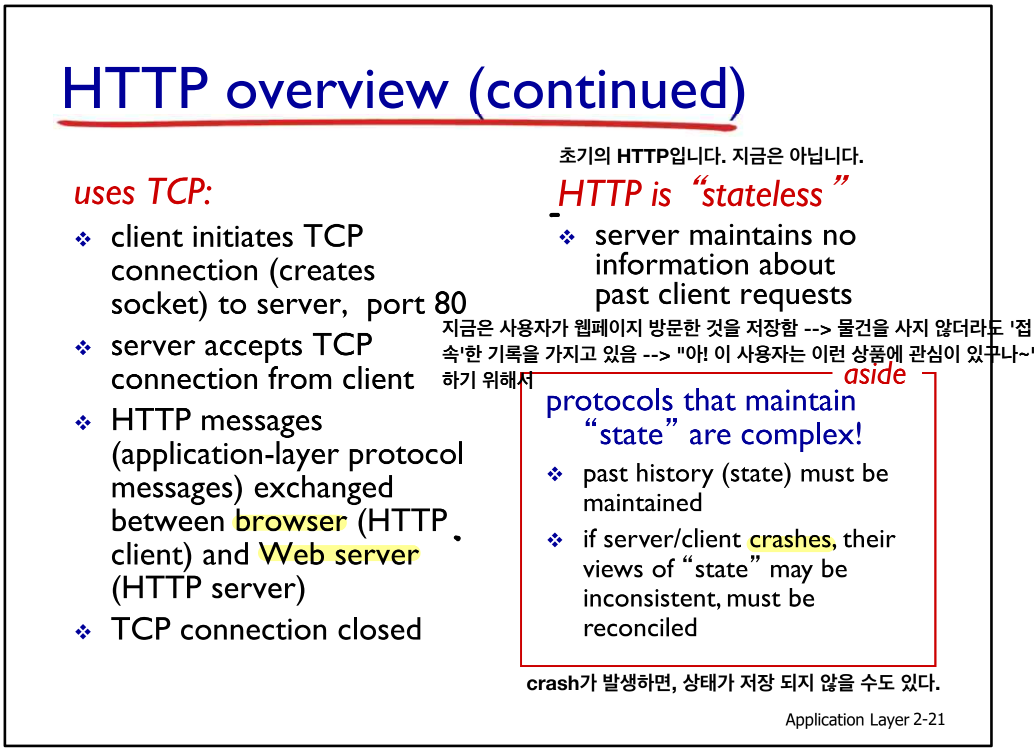


HTTP overview

HTTP: hypertext transfer protocol

- 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

Application Layer 2-20



HTTP overview (continued)

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

초기의 HTTP입니다. 지금은 아닙니다.

HTTP is "stateless"

- server maintains no information about past client requests

지금은 사용자가 웹페이지 방문한 것을 저장함 --> 물건을 사지 않더라도 '접속'한 기록을 가지고 있음 --> "아! 이 사용자는 이런 상품에 관심이 있구나~" 하기 위해서

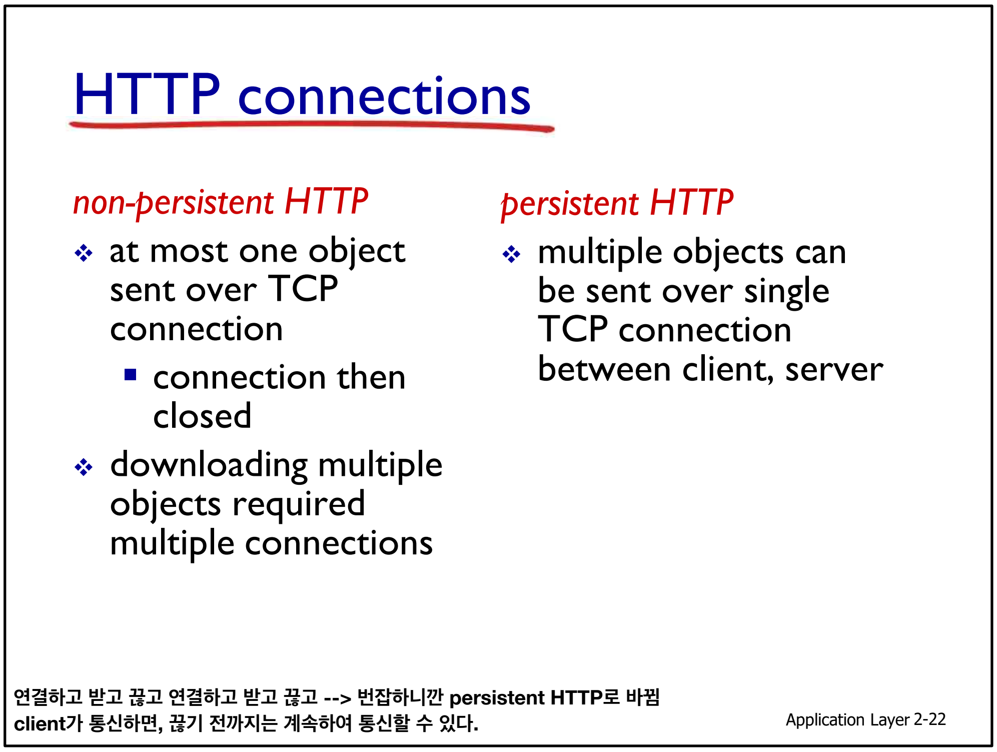
aside

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

crash가 발생하면, 상태가 저장 되지 않을 수도 있다.

Application Layer 2-21



HTTP connections

non-persistent HTTP

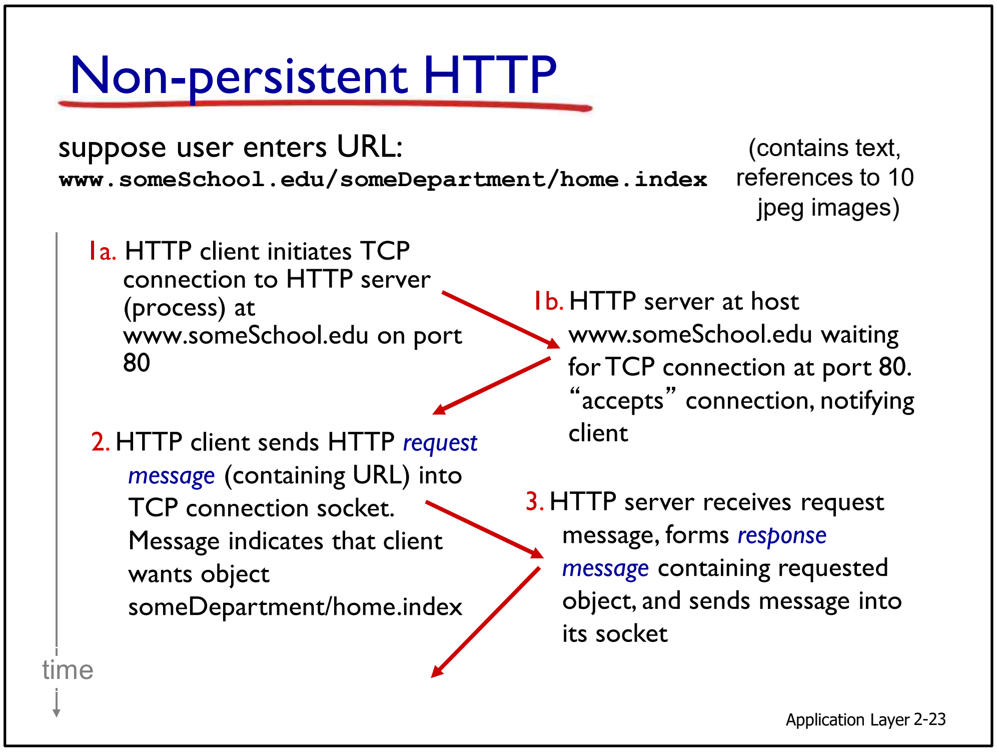
- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

- multiple objects can be sent over single TCP connection between client, server

연결하고 받고 끊고 연결하고 받고 끊고 --> 번잡하니깐 persistent HTTP로 바뀜
client가 통신하면, 끊기 전까지는 계속하여 통신할 수 있다.

Application Layer 2-22



Non-persistent HTTP

suppose user enters URL:
`www.someSchool.edu/someDepartment/home.index`

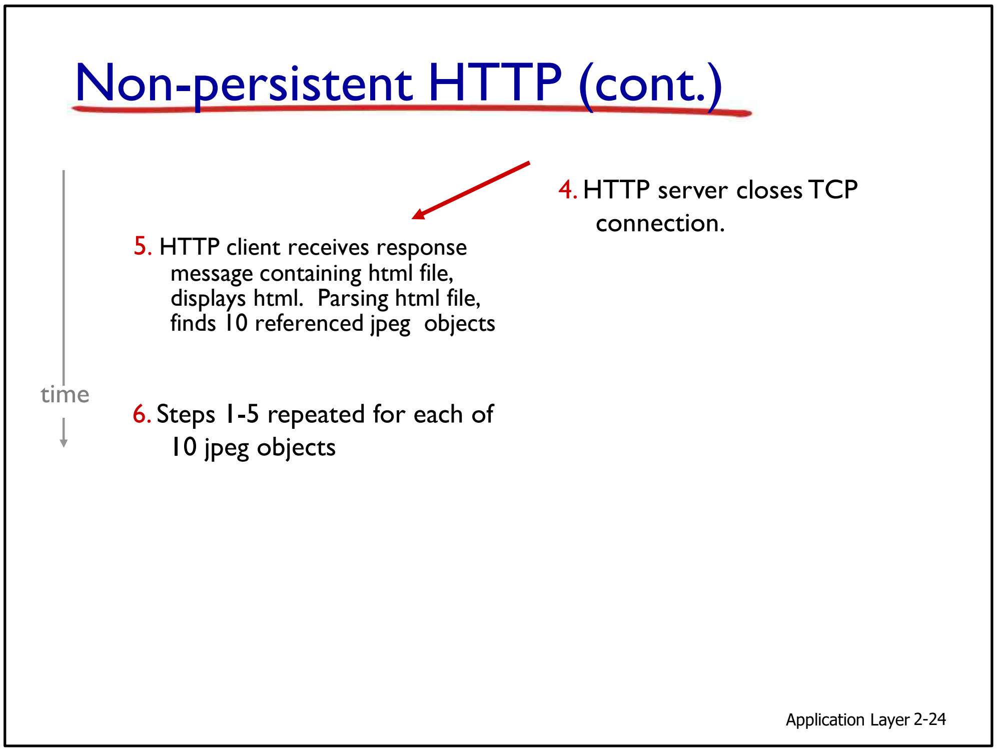
(contains text, references to 10 jpeg images)

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



Non-persistent HTTP (cont.)

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

우리가 고려해야 할 것입니다.

Non-persistent HTTP: response time

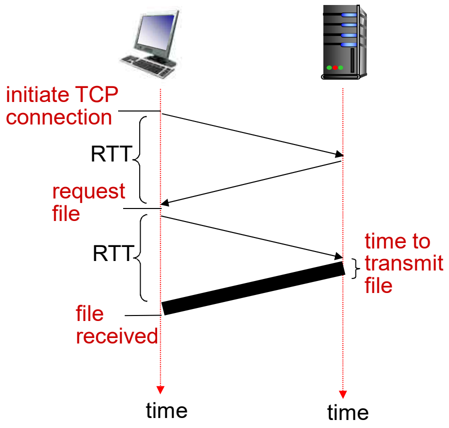
응답받은 하나의 패킷이 서버에서 돌아오는 시간을 입니다. RTT(round trip time)

RTT (definition): time for a small packet to travel from client to server and back

HTTP response time:

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

  2RTT+ file transmission time



Persistent HTTP

non-persistent HTTP issues:

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

안 좋다는 것임
시간도 많이 걸리고 오버헤드 있고 그렇다고 함

persistent HTTP:

- server leaves connection open after sending response
- 그 다음의, 차후의
  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

HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)

carriage return character

line-feed character

request line
(GET, POST,
HEAD commands)

실행될 요청

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

요청수행 (o.x)

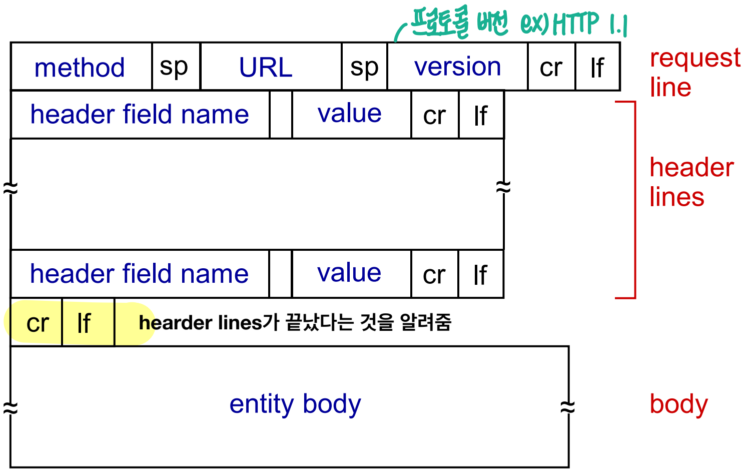
header lines

1)요청에 대한 설명
2)메시지 본문에 대한 설명

carriage return, line feed at start of line indicates end of header lines

서버와 클라이언트 사이의 데이터 교환 방식

HTTP request message: general format



hearder lines가 끝났다는 것을 알려줌

✓

Uploading form input

POST method:

- web page often includes form input
- input is uploaded to server in entity body

URL method:

- uses GET method
- input is uploaded in URL field of request line:

```
www.somesite.com/animalsearch?monkeys&banana
```

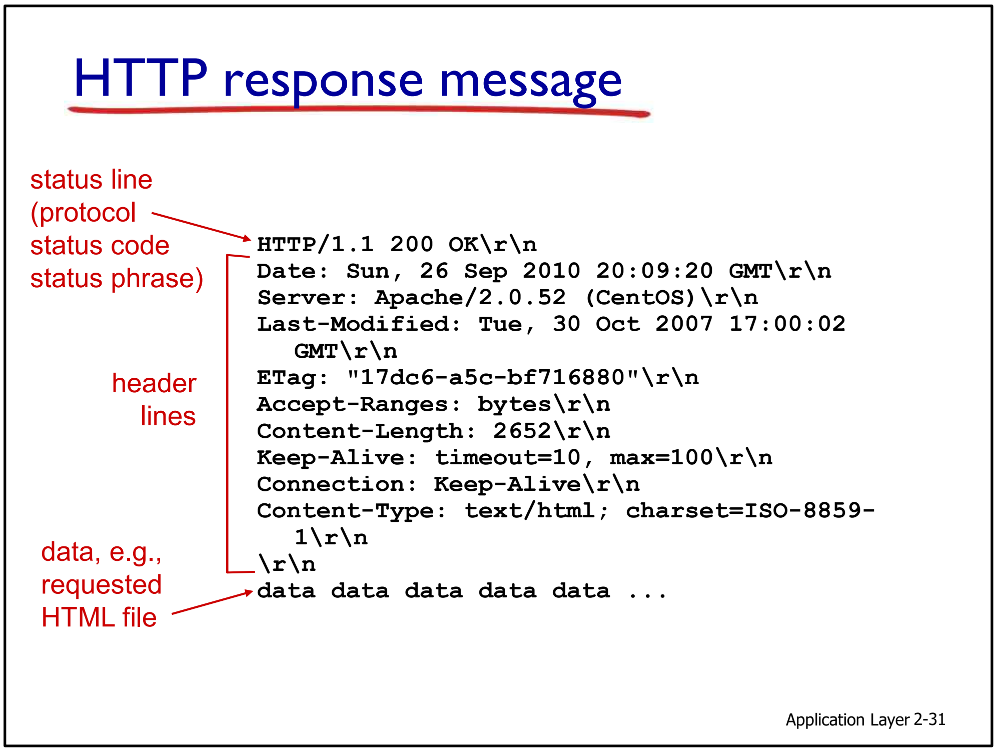
Method types

HTTP/1.0:

- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field



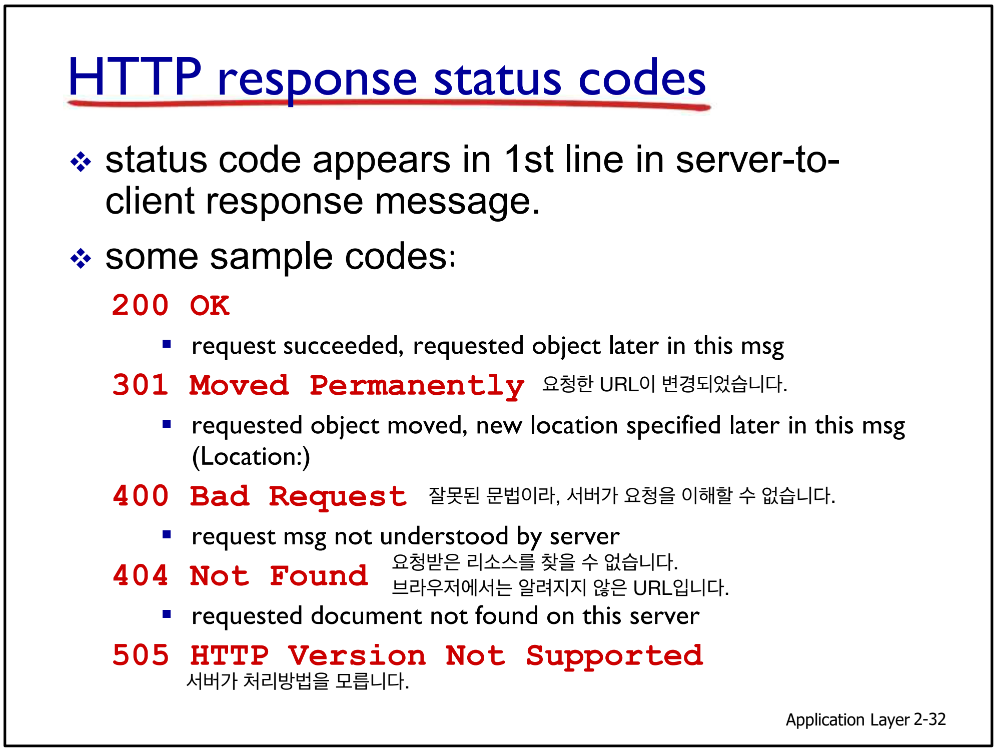
HTTP response message

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```



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 msg

301 Moved Permanently 요청한 URL이 변경되었습니다.

- requested object moved, new location specified later in this msg (Location:)

400 Bad Request 잘못된 문법이라, 서버가 요청을 이해할 수 없습니다.

- request msg not understood by server

404 Not Found 요청받은 리소스를 찾을 수 없습니다.
브라우저에서는 알려지지 않은 URL입니다.

- requested document not found on this server

505 HTTP Version Not Supported
서버가 처리방법을 모릅니다.

서버가 사용자의 웹 브라우저에 전송하는 작은 데이터 조각을 말합니다.
브라우저는 그 데이터 조각들을 저장해 놓았다가, 동일한 서버에 재 요청시 저장된 데이터를 함께 전송합니다.
두 요청이 동일한 브라우저에서 들어왔는지 판단할 때 사용합니다.

User-server state: cookies

원래는 stateless였지만, cookies를 사용하여 상태정보를 저장합니다.

many Web sites use cookies

four components:

```
HTTP/1.0 200 OK
```

1) cookie header line of HTTP *response* message

2) cookie header line in next HTTP *request* message

```
Content-type: text/html
Set-Cookie: yummy_cookie=choco
Set-Cookie: tasty_cookie=strawberry
```

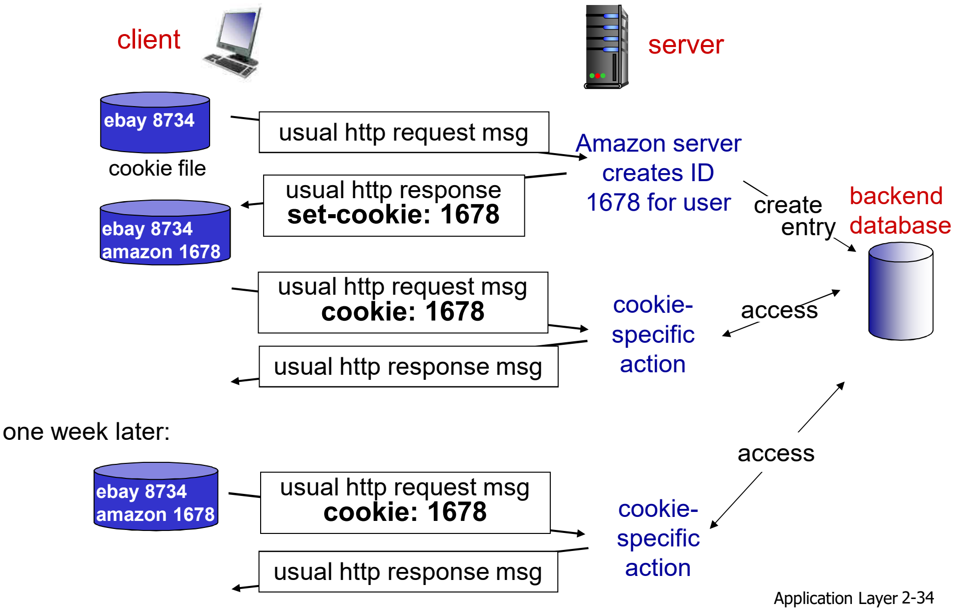
3) cookie file kept on user' s host, managed by user' s browser

4) back-end database at Web site

example:

- ❑ Susan access Internet always from same PC
- ❑ She visits a specific e-commerce site for first time 전자상거래
- ❑ When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID

Cookies: keeping "state" (cont.)



Cookies (continued)

what cookies can be used for:

- shopping carts
- recommendations
- Persistent logins
- user session state (Web e-mail)

aside

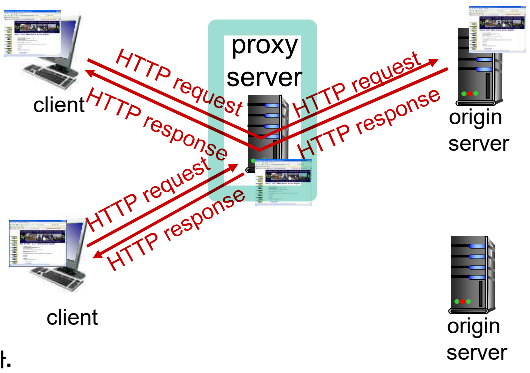
cookies and privacy:

- ❑ cookies permit sites to learn a lot about you
- ❑ you may supply name and e-mail to sites
- ❑ search engines use redirection & cookies to learn yet more
- ❑ advertising companies obtain info across sites

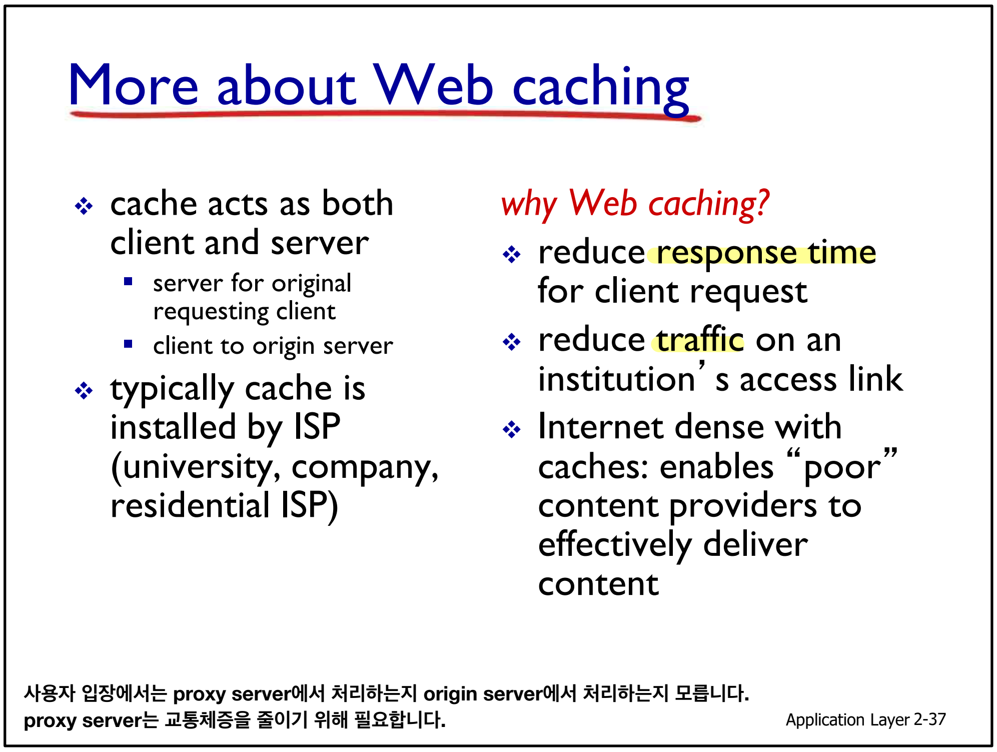
Web caches (proxy server)

goal: satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



구글 server는 엄청난 request를 처리하지 않습니다.
앞에 proxy server가 정리를 해줍니다.
처리 할 수 있는 웬만한 일들을 해준다고 생각하면 됩니다.
처리 할 수 없는 일들만 origin server로 보내서 처리합니다.



More about Web caching

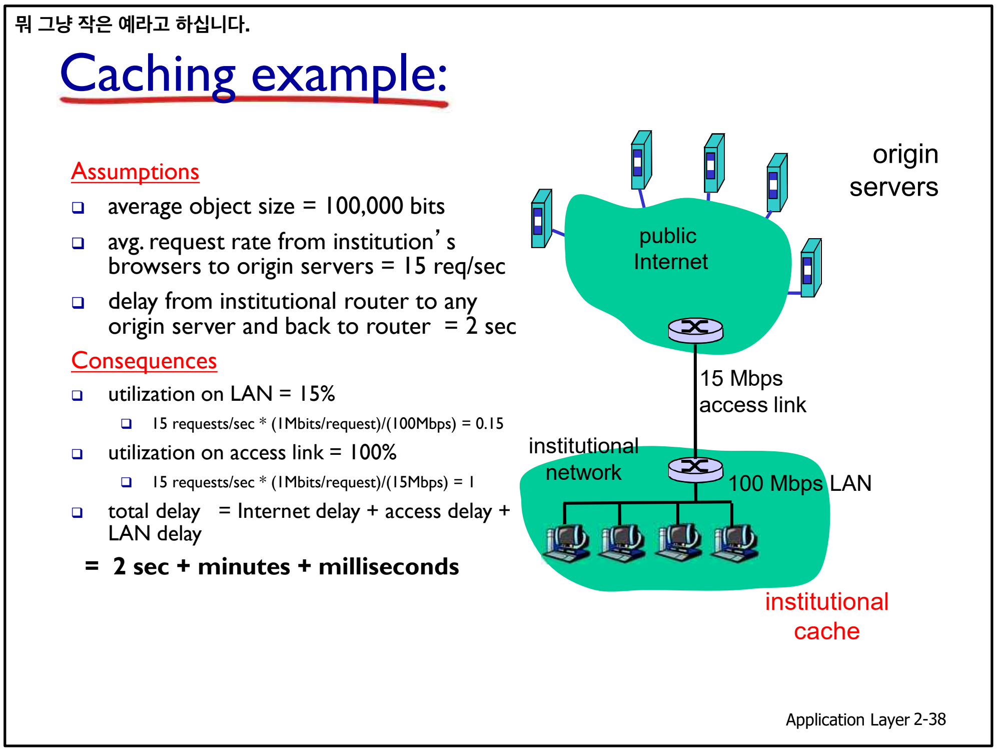
- cache acts as both client and server
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

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

사용자 입장에서는 proxy server에서 처리하는지 origin server에서 처리하는지 모릅니다.
proxy server는 교통체증을 줄이기 위해 필요합니다.

Application Layer 2-37



뭐 그냥 작은 예라고 하십니다.

Caching example:

Assumptions

- average object size = 100,000 bits
- avg. request rate from institution's browsers to origin servers = 15 req/sec
- delay from institutional router to any origin server and back to router = 2 sec

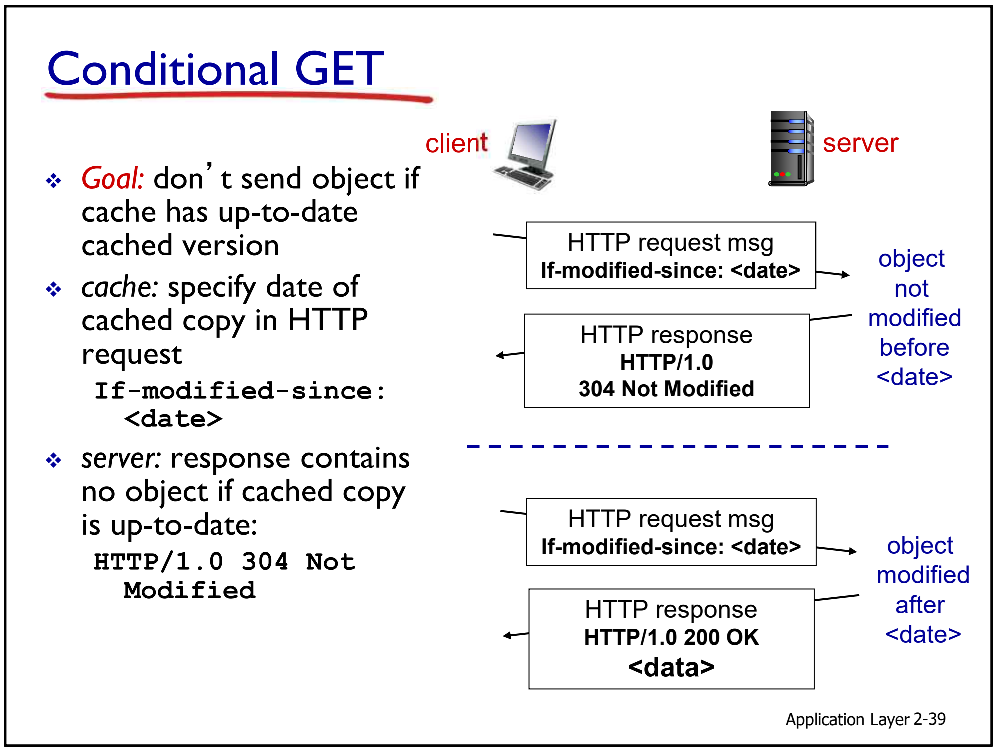
Consequences

- utilization on LAN = 15%
  - 15 requests/sec * (1Mbits/request)/(100Mbps) = 0.15
- utilization on access link = 100%
  - 15 requests/sec * (1Mbits/request)/(15Mbps) = 1
- total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + milliseconds

Application Layer 2-38

✓



Conditional GET

- *Goal:* don't send object if cache has up-to-date cached version
- *cache:* specify date of cached copy in HTTP request
  `If-modified-since: <date>`
- *server:* response contains no object if cached copy is up-to-date:
  `HTTP/1.0 304 Not Modified`

client

server

HTTP request msg
If-modified-since: <date>

HTTP response
HTTP/1.0
304 Not Modified

object not modified before <date>

HTTP request msg
If-modified-since: <date>

HTTP response
HTTP/1.0 200 OK
<data>

object modified after <date>

Application Layer 2-39