

컴퓨터 네트워크

- 애플리케이션 계층 1 -

순천향대학교 사물인터넷학과

목표

■ 네트워크 애플리케이션의 **개념과 구현** 측면 학습

- 애플리케이션 계층 프로토콜
- 클라이언트와 서버, P2P
- 트랜스포트 계층 서비스 모델
- 비디오 스트리밍 및 컨텐츠 분배 네트워크 **CDN**

■ 주요 애플리케이션 계층 **프로토콜** 학습

- **HTTP**
- **SMTP / POP3 / IMAP**
- **DNS**

■ 네트워크 애플리케이션 **프로그래밍**

- 소켓 API

애플리케이션 계층

2.1 네트워크 애플리케이션의 원리

2.2 웹과 HTTP

2.3 전자메일

- SMTP, POP3, IMAP

2.4 DNS - 인터넷 디렉토리 서비스

2.5 P2P 파일 분배

2.6 비디오 스트리밍과 컨텐츠 분배 네트워크

2.7 소켓 프로그래밍: 네트워크 애플리케이션 작성

네트워크 애플리케이션 예

- 소셜 네트워킹 서비스 social networking service
 - 웹 web
 - 텍스트 메시징 instant messaging
 - e-mail
 - 다중 사용자 네트워크 게임 multi-user network game
 - 저장 비디오 스트리밍 streaming stored video
 - YouTube, Netflix
 - 실시간 화상 회의 real-time video conferencing: Zoom
 - VoIP Voice over IP: Skype
 - P2P 파일 공유 P2P file sharing
 - 검색 search
 - 원격 로그인 remote login
-

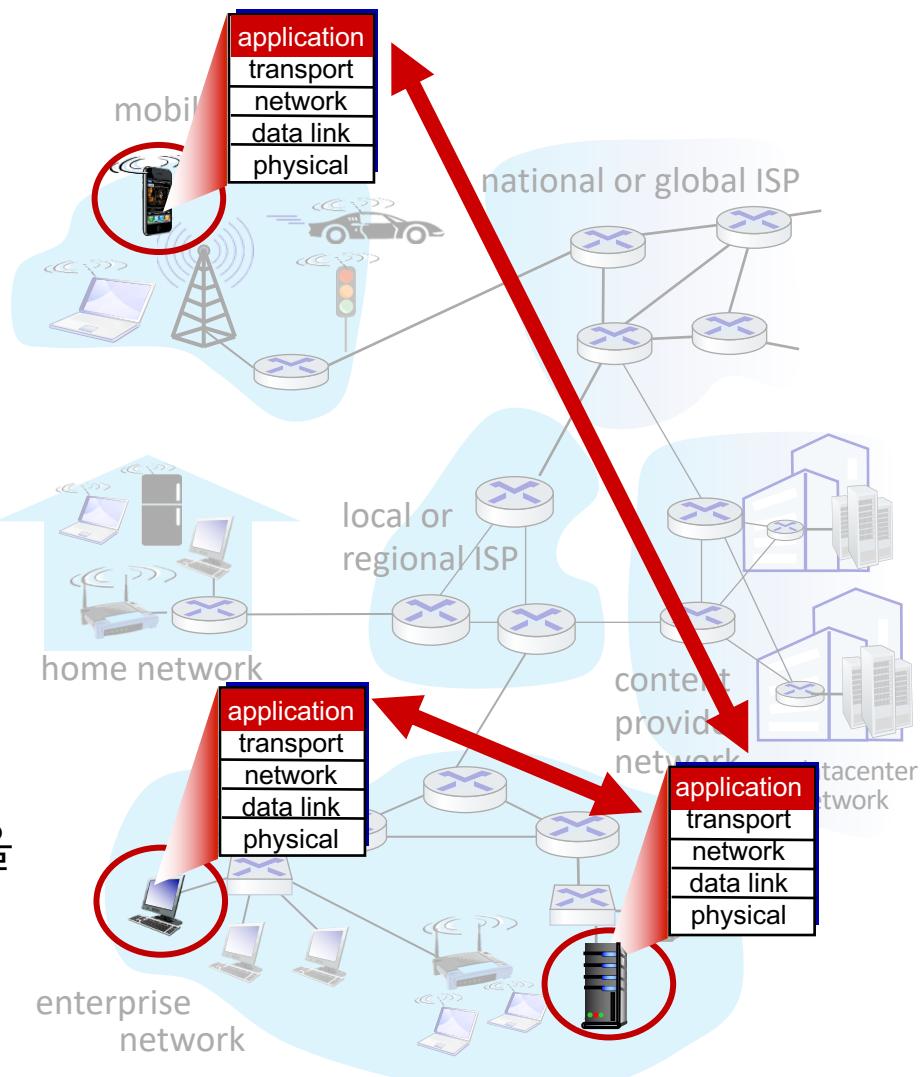
네트워크 애플리케이션 작성

■ 네트워크 애플리케이션

- 서로 다른 **종단 시스템**에서 실행됨
- 네트워크를 통해 **통신**
- 예) 웹 서버 프로그램은 웹 브라우저 프로그램과 통신

■ 네트워크 코어 장비를 위한 애플리케이션은 필요 없음

- 네트워크 코어 장비는 사용자 애플리케이션을 실행하지 않음
- 따라서, 종단 시스템의 애플리케이션을 빠르게 개발하고 적용할 수 있음



네트워크 애플리케이션 구조

■ 애플리케이션 구조 application architecture

- 개발자가 설계하는 애플리케이션이 다양한 종단 시스템에서 어떻게 조직되어야 하는지를 지시

■ 일반적으로 아래 2가지 구조가 사용됨

- 클라이언트/서버 구조 client-server architecture
- P2P 구조 Peer-to-Peer architecture

클라이언트/서버 구조

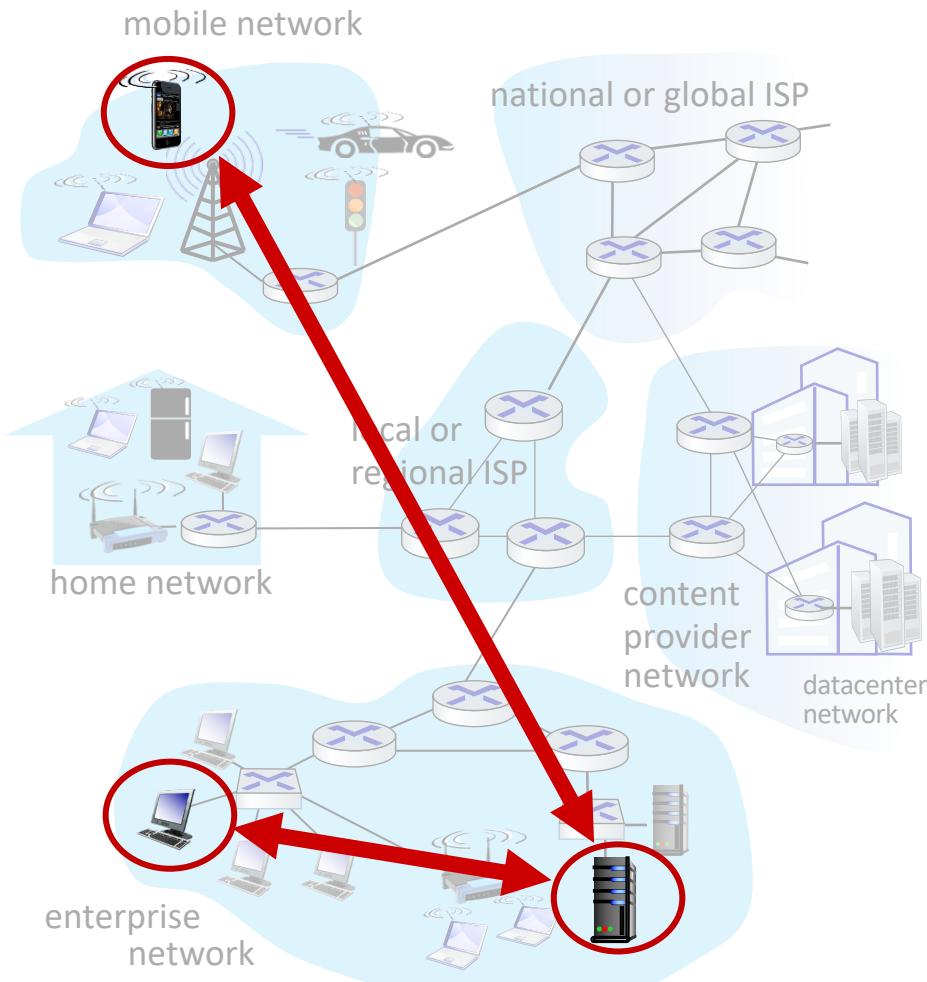
■ 서버 server

- 항상 켜져 있는(Always-on) 호스트
 - ✓ 서비스 제공
- 고정 IP 주소 할당
- 확장성을 위해
(때때로) 데이터 센터에 존재

■ 클라이언트 client

- 서버에 접속하여 통신
 - ✓ 서비스 요청
- 타 클라이언트와 직접 통신하지 않음
- 동적 IP를 가질 수 있음
- 항상 연결되어 있지 않고,
간헐적으로 통신할 수 있음

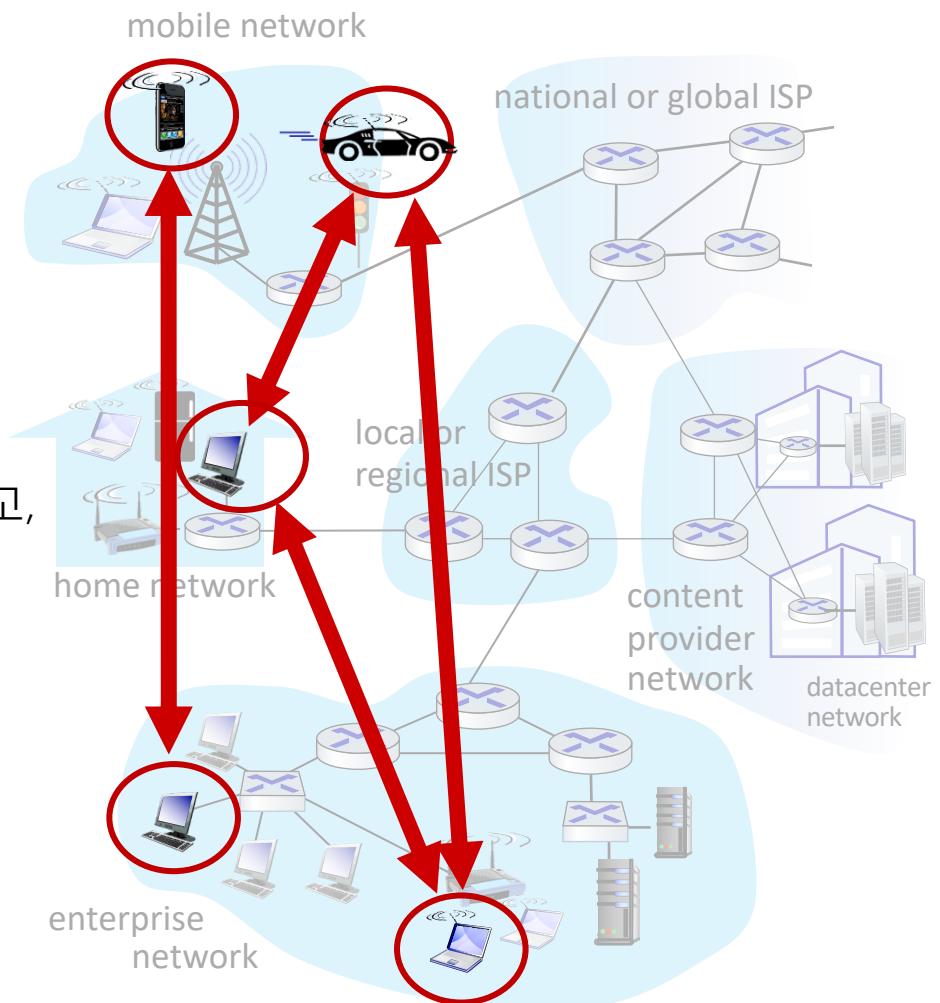
예) HTTP, IMAP, FTP



P2P 구조 Peer-to-Peer Architecture

■ P2P 구조

- 항상 켜져 있는 서버가 없음
- 임의의 종단 시스템들끼리 직접 통신함
- 각 **피어**^{Peer}들이 각각 서비스를 요청하고, 서비스를 제공함
 - ✓ 높은 자기 확장성(self scalability)
 - 새로운 피어는 새로운 서비스를 제공하고, 새로운 서비스를 요청
- 피어는 간헐적으로 인터넷에 연결되고, IP가 변할 수 있음
 - ✓ 관리가 어려움
- 예) P2P 파일 공유



프로세스 간 통신

■ 프로세스 process

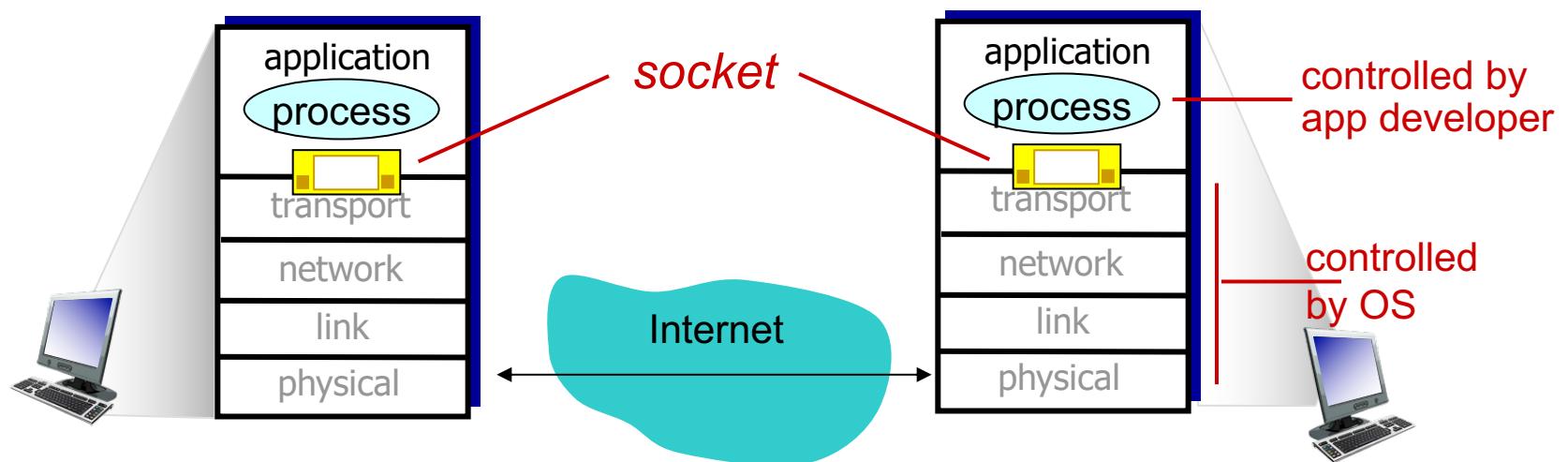
- 호스트에서 실행 중인 프로그램
- 동일 호스트 내에서, 2개의 프로세스는 OS에서 정의한 프로세스 간 통신^{IPC, inter-process communication}을 통해 통신 수행
- 서로 다른 호스트 사이에서, 2개의 프로세스는 메시지를 교환하여 통신

■ 클라이언트/서버 프로세스

- 클라이언트 프로세스는 두 프로세스 간의 통신세션을 시작(접속을 수행)하고 서비스를 요청하는 프로세스
- 서버 프로세스는 통신세션을 시작하기 위해 접속을 기다리는 프로세스
 - ✓ 예) 웹 브라우저가 대기 중인 웹 서버 프로세스와 접속을 수행하고 서비스를 요청
 - 웹 브라우저: 클라이언트 프로세스, 웹 서버: 서버 프로세스
- P2P 구조의 애플리케이션도 클라이언트 프로세스와 서버 프로세스를 가짐
 - ✓ 예) P2P 구조에서 피어 A가 피어 B에게 파일 전송을 요청하는 경우
 - 피어 A: 클라이언트 프로세스, 피어 B: 서버 프로세스

소켓 socket

- 프로세스는 소켓을 통해 메시지를 보내고 받음
 - 소켓은 호스트의 애플리케이션 계층과 트랜스포트 계층 간의 인터페이스
 - 프로세스는 “집”, 소켓은 “출입구”에 비유
 - ✓ 송신 프로세스는 출입구(소켓) 바깥 네트워크로 메시지를 밀어냄
 - ✓ 송신 프로세스는 수신 프로세스에 있는 소켓으로 메시지를 전달하기 위해 하위 계층의 서비스에 의존
 - 소켓은 애플리케이션과 네트워크 사이의 API Application Programming Interface



프로세스의 주소

- 메시지를 수신하기 위해서는 **식별자**^{Identifier}가 필요함
 - 호스트는 32비트 IP 주소를 가짐
 - Q. 호스트의 IP 주소로 프로세스를 식별할 수 있는가?
 - A. No.
 - ✓ 동일 호스트에서 많은 프로세스들이 동시에 실행될 수 있음
- 따라서, 호스트 상의 프로세스를 구분하기 위해서는 호스트의 IP 주소와 호스트에서 프로세스를 구분하는 **포트 번호**^{port number}가 필요함
 - 포트 번호 예제
 - ✓ HTTP 서버: 80, 메일 서버: 25
 - HTTP 메시지를 www.sch.ac.kr 웹 서버로 보내는 경우
 - ✓ IP 주소: 220.69.189.111
 - ✓ 포트 번호: 80

애플리케이션 계층 프로토콜

■ 애플리케이션 계층 프로토콜은 다음의 내용을 정의

- 교환되는 메시지 타입

- ✓ 예) 요청request 메시지, 응답response 메시지

- 메시지 문법syntax

- ✓ 예) 메시지 내에 어떤 필드가 있고, 각 필드의 길이는 얼마인지 정의

- 메시지 의미semantics

- ✓ 각 필드에 있는 정보의 의미

- 규칙rule

- ✓ 언제, 어떻게 프로세스가 메시지를 전송하고 메시지에 응답하는지를 결정

■ 개방형 프로토콜open protocols (예: HTTP, SMTP, FTP)

- RFC에 정의됨

- 상호운용성interoperability 제공

■ 독점적 프로토콜proprietary protocols (예: Zoom, 카카오톡)

애플리케이션이 필요로 하는 트랜스포트 서비스

■ 신뢰적 reliable 데이터 전송

- 이메일, 웹과 같은 애플리케이션은 100% 신뢰성 있는 데이터 전송이 필요
- 오디오와 같은 App은 약간의 손실을 감수할 수 있음
 - ✓ 손실 허용 애플리케이션 loss-tolerant application

■ 시간 timing

- 인터넷 전화, 인터넷 게임과 같은 App은 효율적인 서비스를 위해서 낮은 지연시간 low delay 을 요구함

■ 처리율 throughput

- 멀티미디어와 같은 App은 최소한의 처리율이 보장되어야 함
 - ✓ 대역폭 민감 애플리케이션 bandwidth-sensitive application
- 파일 전송과 같은 다른 애플리케이션은 처리율에 덜 민감함
 - ✓ 탄력적 애플리케이션 elastic application

■ 보안 security

- 암호화, 데이터 무결성, 인증 서비스 등

애플리케이션의 트랜스포트 서비스 요구사항

application	data loss	throughput	time sensitive?
file transfer/download	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, 10's msec
streaming audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	Kbps+	yes, 10's msec
text messaging	no loss	elastic	yes and no

인터넷 트랜스포트 프로토콜이 제공하는 서비스: TCP

■ TCP Transmission Control Protocol 서비스

- 신뢰성 있는 데이터 전송 reliable data transfer
 - ✓ 송수신 프로세스 간에 데이터의 손실 없이 올바른 순서로 전달
- 흐름 제어 flow control
 - ✓ 송신 프로세스가 수신 프로세스의 처리 속도에 맞추어 송신
- 혼잡 제어 congestion control
 - ✓ 네트워크가 혼잡해지면, 송신 프로세스의 전송 속도를 낮춤
- 연결지향형 connection-oriented
 - ✓ 클라이언트와 서버 프로세스들 간에 연결을 설정
 - ✓ 서로 간 전송 제어 정보를 교환하고 송수신 준비 (핸드쉐이킹 handshaking)
- 제공하지 않는 서비스
 - ✓ 저지연, 최소 처리율 보장, 보안

인터넷 트랜스포트 프로토콜이 제공하는 서비스: UDP

■ UDP User Datagram Protocol 서비스

- 비연결형

- ✓ 두 프로세스가 통신 전에 정보 교환(핸드쉐이킹)을 하지 않음

- 비신뢰적인 데이터 전송 서비스 unreliable data transfer

- 제공하지 않는 서비스

- ✓ 신뢰성 있는 데이터 전송, 흐름제어, 혼잡제어, 저지연, 최소 처리율 보장, 보안

■ Q. 그럼 도대체 UDP는 왜, 어디에 사용되는가?

- 실시간 애플리케이션들이 빠른 전송속도를 위해 UDP 사용

- TCP 혼잡제어와 패킷 오버헤드 문제 회피

- 많은 방화벽들이 UDP 트래픽을 차단하도록 설정되어 점차적으로 TCP 상에서 멀티미디어와 실시간 애플리케이션을 수행하도록 선택

인터넷 애플리케이션의 트랜스포트 프로토콜

application	application layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP 1.1 [RFC 7320]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7320], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

보안 TCP

■ TCP & UDP

- 암호화를 제공하지 않음
- 소켓을 통한 평문(암호화되지 않은 내용)의 비밀번호 등이 인터넷으로 전송됨 → 중간의 링크에서 훔쳐볼 수 있음

■ SSL Secure Socket Layer / TLS Transport Layer Security

- 암호화된 TCP 연결을 제공
- 데이터 무결성 data integrity
- 종단 간 인증 authentication
- SSL/TLS는 애플리케이션 계층 프로토콜
 - ✓ 애플리케이션은 SSL/TLS 라이브러리를 사용하여 암호화된 데이터를 TCP 소켓에 전달
- 상세 내용은 Chapter 8 참조!!!
- HTTPS Hypertext Transfer Protocol Secure
 - ✓ HTTP over SSL / HTTP over TLS
 - ✓ 상호 인증 및 데이터 암호화, 무결성 지원

애플리케이션 계층

2.1 네트워크 애플리케이션의 원리

2.2 웹과 HTTP

2.3 전자메일

- SMTP, POP3, IMAP

2.4 DNS - 인터넷 디렉토리 서비스

2.5 P2P 파일 분배

2.6 비디오 스트리밍과 컨텐츠 분배 네트워크

2.7 소켓 프로그래밍: 네트워크 애플리케이션 작성

웹과 HTTP

■ 웹 페이지는 객체 objects로 구성됨

- 객체는 HTML 파일, JPEG 이미지, 비디오 파일, 오디오 파일, ...

■ 웹 페이지는 기본 HTML 파일과 여러 참조 객체들로 구성

- 예) 웹 페이지가 HTML 텍스트와 5개의 JPEG 이미지로 구성
 - ✓ 1개의 기본 HTML 파일과 5개의 이미지 객체

■ 각 객체는 URL Uniform Resource Locator로 참조됨

ipsi.sch.ac.kr/schinfor/program04.html

host name

path name

www.someschool.edu/someDept/pic.gif

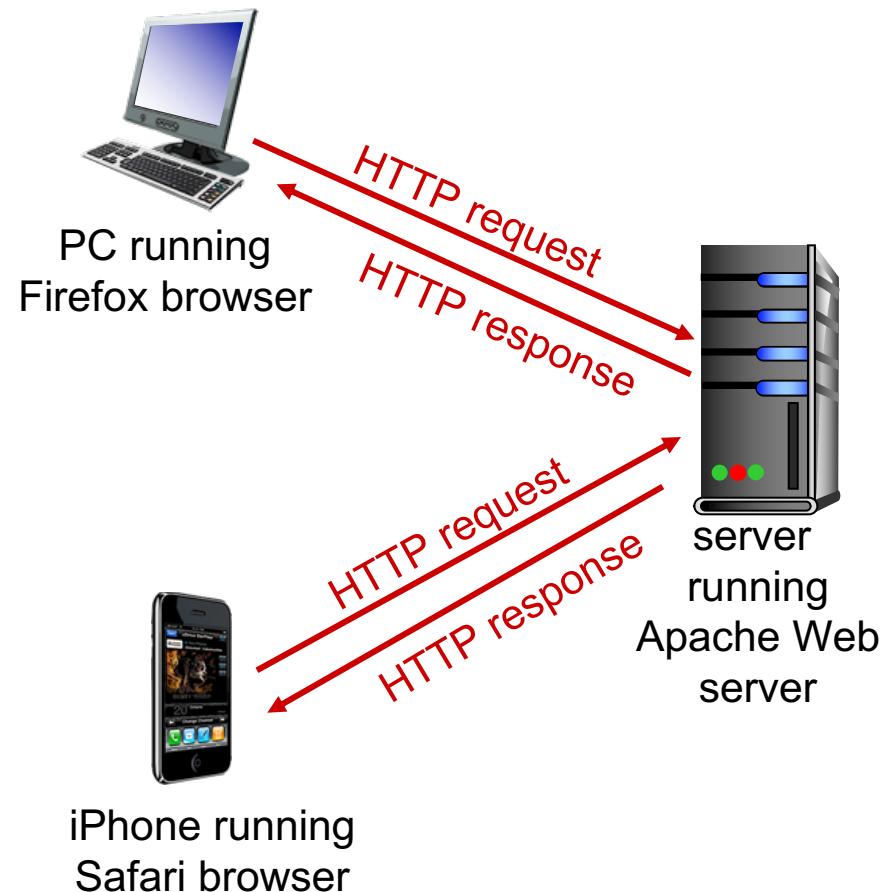
host name

path name

HTTP 개요

HTTP HyperText Transfer Protocol

- 웹의 애플리케이션 계층 프로토콜
- 클라이언트-서버 모델
 - 클라이언트
 - ✓ HTTP 프로토콜을 이용해서 웹 객체를 요청하고, 수신하여, 보여주는 웹 브라우저
 - 서버
 - ✓ HTTP 프로토콜을 이용해서 클라이언트의 요청에 응답하여 객체들을 보내는 웹 서버



HTTP 개요

■ TCP 사용

- 클라이언트는 포트 80번을 이용하여 서버로 TCP 연결을 시작
- 서버는 클라이언트로부터 TCP 연결을 수락
- 클라이언트(브라우저)와 서버(웹 서버) 간 HTTP 메시지(애플리케이션 계층 프로토콜 메시지) 교환
- TCP 연결 종료

■ HTTP는 “비상태 프로토콜(stateless protocol)”

- 서버는 클라이언트 과거 요청에 대한 정보(상태)를 저장하지 않음

참고

‘상태’를 저장하는 프로토콜은 복잡함

- 과거 히스토리(상태)가 유지되어야 함
- 서버나 클라이언트가 고장 나면, 상호 간의 상태 불일치가 발생하므로, 상태 일치 절차가 필요함

HTTP 연결 HTTP connections

■ HTTP 연결은 **비지속 연결**과 **지속 연결** 2가지 방식이 있음

■ **비지속 연결** non-persistent connection

- 요구/응답 쌍이 분리된 TCP 연결을 통해 송수신
- 하나의 TCP 연결로 하나의 객체만 전송
- 따라서, 여러 개의 객체를 다운받기 위해서는

여러 개의 연결이 필요

1. TCP 연결 열기
2. 1개 객체 전송
3. TCP 연결 닫기
4. 1~3 반복

■ **지속 연결** persistent connection

- 모든 요구/응답 쌍이 같은 TCP 연결 상에서 송수신
- 다수의 객체들이 하나의 TCP 연결로 전송
- 따라서, 여러 개의 객체를 다운받기 위해서는

1개의 연결이 필요

1. TCP 연결 열기
2. 객체 전송
3. 객체 전송
4. ...
5. TCP 연결 닫기

비지속 연결 HTTP Non-persistent HTTP

■ 비지속 연결 HTTP

- 1개의 TCP 연결에 1개의 객체가 전송됨
 - ✓ 객체 전송 후 TCP 연결 종료
- 여러 개의 객체를 다운받기 위해서는 **여러 개의 TCP 연결이 필요함**

■ 사용자가 아래 URL을 입력한다고 가정

- www.someSchool.edu/someDepartment/home.index
- 해당 사이트는 텍스트와 10개의 jpeg 이미지를 가짐



1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

time
↓

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. “accepts” connection, notifying client

비지속 연결 HTTP Non-persistent HTTP

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket.

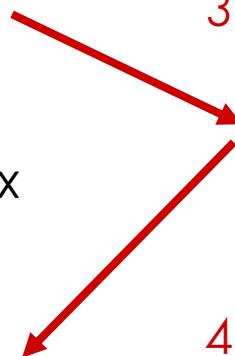
Message indicates that client wants object
someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

- time 5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

4. HTTP server closes TCP connection.

6. Steps 1-5 repeated for each of 10 jpeg objects



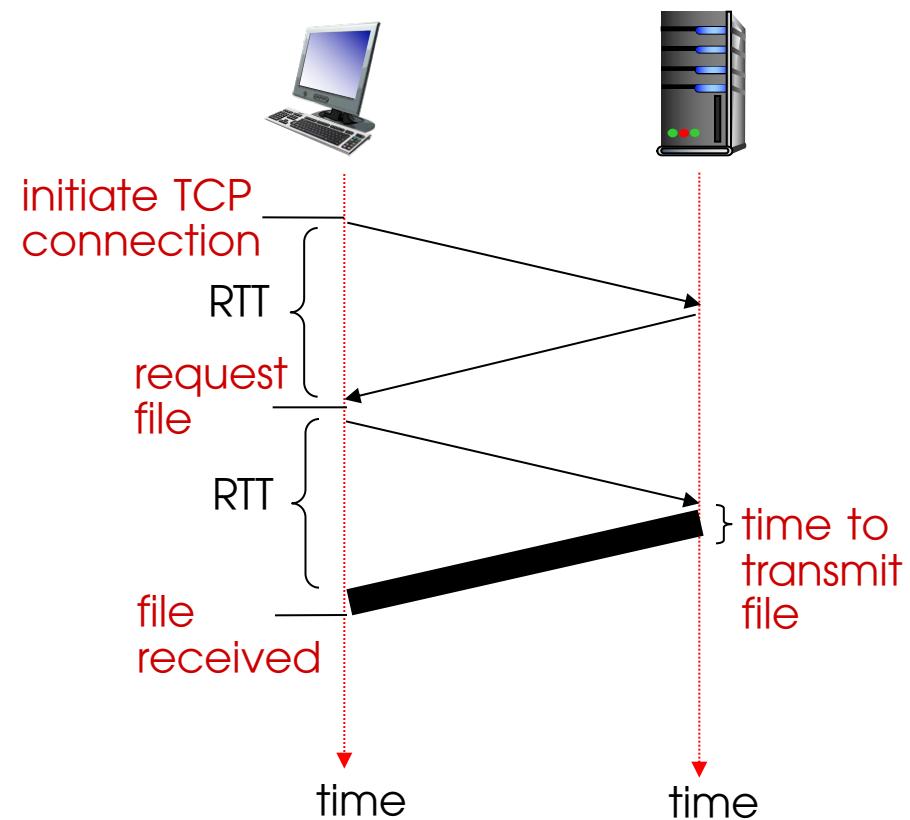
비지속 연결 HTTP : 응답 시간 Response Time

■ RTT Round Trip Time

- 클라이언트에서 송신된 작은 크기의 패킷이 서버까지 간 후 (그 응답이) 다시 클라이언트로 되돌아오는데 걸리는 시간

■ HTTP 응답 시간

- TCP 연결을 위해 1 RTT
- HTTP 요청을 하고, HTTP 응답으로 처음 몇 바이트를 받는데 필요한 시간 1 RTT
- 파일(객체) 전송 시간
- 비지속 연결 HTTP 응답 시간
= 2RTT + 파일 전송 시간



지속 연결 HTTP Persistent HTTP

■ 비지속 연결 HTTP의 단점

- 각 객체 당 2 RTT(+ 객체 전송시간)가 걸림
- 각 TCP 연결에 대한 OS 오버헤드
- 브라우저는 각 참조 객체를 가져오기 위해 종종 **병렬 TCP 연결**을 수행

■ 지속 연결 HTTP (HTTP1.1)

- 서버는 응답을 보낸 후에 TCP 연결을 그대로 유지
- 클라이언트/서버 간의 이후 HTTP 메시지들은 같은 연결을 통해 송수신
- 클라이언트는 참조 객체를 발견하면 바로 요청을 송신
- 전체 참조 객체에 대해 **1 RTT**만 필요함
 - ✓ 비지속 연결 HTTP에 비해 응답시간을 절반으로 줄임

HTTP 요청 메시지 | HTTP Request Message

■ 2가지 종류의 HTTP 메시지

- 요청(Request)
- 응답(Response)

■ HTTP 요청 메시지

- ASCII 텍스트 포맷

request line (GET, POST,
HEAD commands)

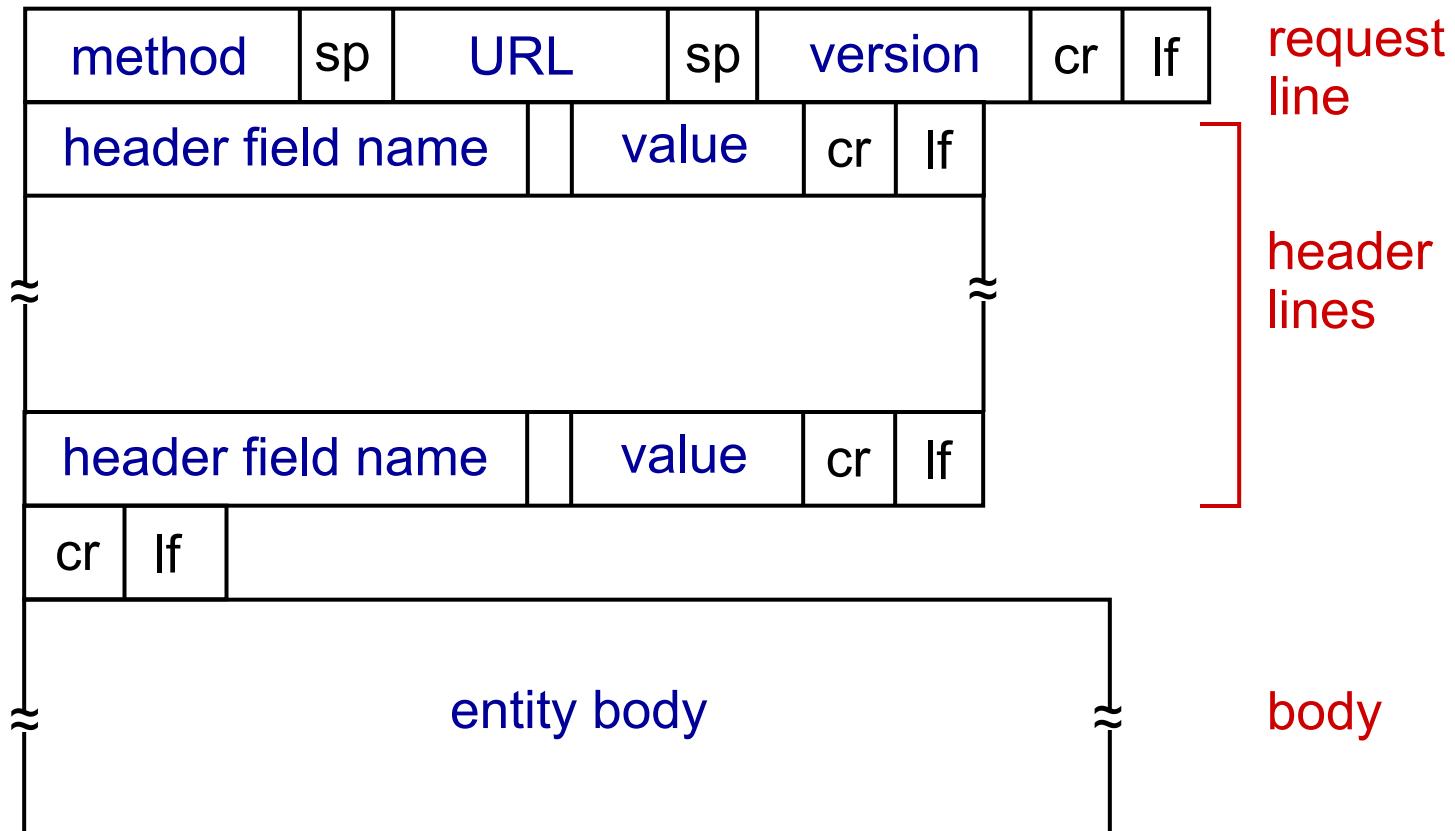
```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
```

header
lines

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

carriage return character
line-feed character

HTTP 요청 메시지: 형식 (format)



HTTP 요청 메시지: 메소드 종류 (method types)

■ GET

- 브라우저가 객체를 요청할 때 사용
- 가장 많이 사용됨
- URL에 사용자 입력을 포함시켜 서버로 전송할 수 있음

예) https://search.daum.net/search?w=tot&DA=YZR&t__nil_searchbox=bt&n&sug=&sugo=&q=iot

■ POST

- 웹 페이지는 폼 입력을 요구하는 경우가 있음
 - ✓ 로그인, 검색 등
- HTTP 요청 메시지의 **body(몸체)**에 사용자 입력이 저장되어 서버로 전달됨

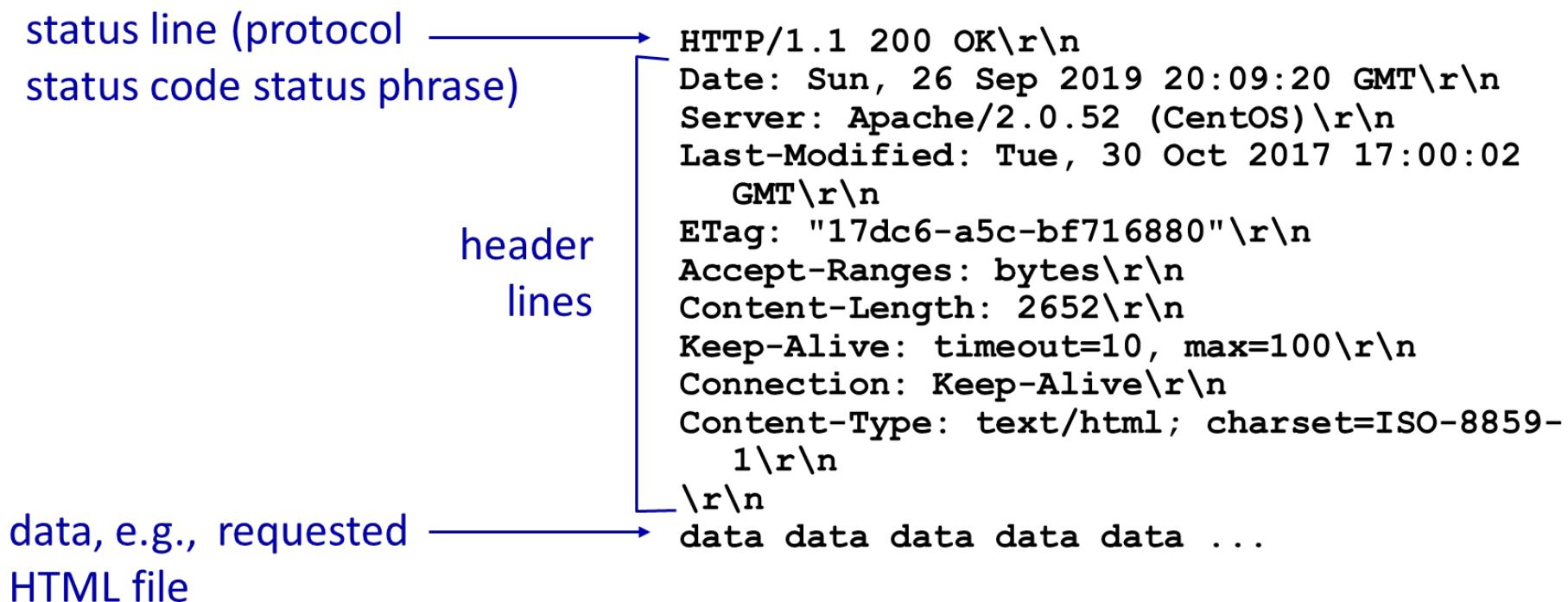
■ HEAD

- GET 메소드와 유사
- 서버가 응답 시 요청된 객체는 보내지 않음
- 주로 디버깅을 위해 사용됨

■ PUT

- 새로운 파일을 서버에 업로드
- URL 필드에 있는 경로에 있는 파일을 HTTP 요청 메시지의 body 안의 파일로 교체함

HTTP 응답 메시지 | HTTP Response Message



HTTP 응답 상태 코드

■ 상태 코드 Status code

- 응답 메시지의 1번째 라인(상태 라인 status line)에 표시됨

■ 주요 상태 코드

- 200 OK

- ✓ 요청이 성공되었고, 요청된 객체가 이 메시지로 보내짐

- 301 Moved Permanently

- ✓ 요청한 객체가 이동되었음
 - ✓ 새로운 위치가 메시지의 이후 부분에 기술됨 ("Location:" 헤더)

- 400 Bad Request

- ✓ 서버가 요청을 이해할 수 없다는 일반 오류 코드

- 404 Not Found

- ✓ 요청된 문서가 서버에 존재하지 않음

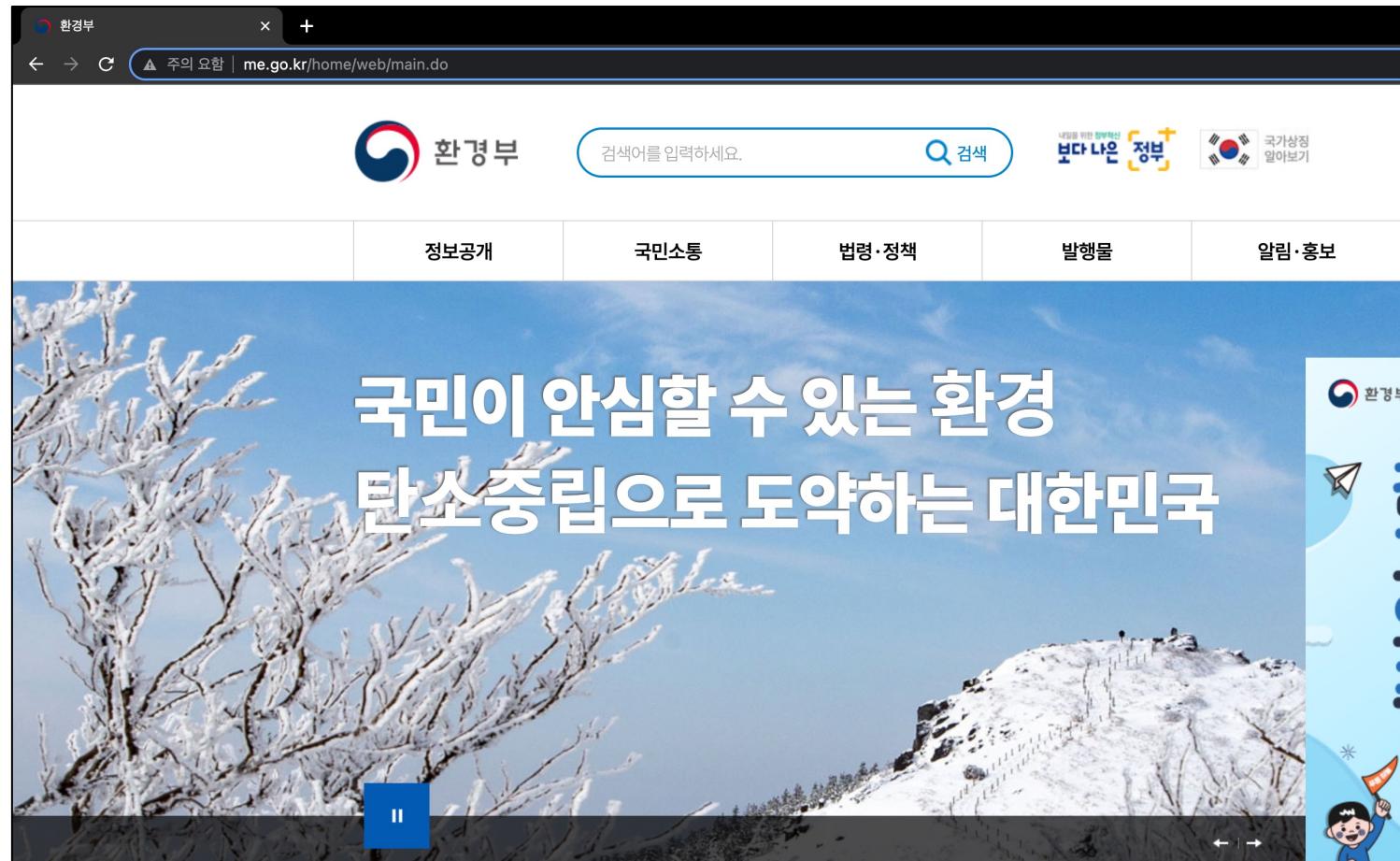
- 505 HTTP Version Not Supported

- ✓ 요청된 HTTP 프로토콜 버전을 서버가 지원하지 않음

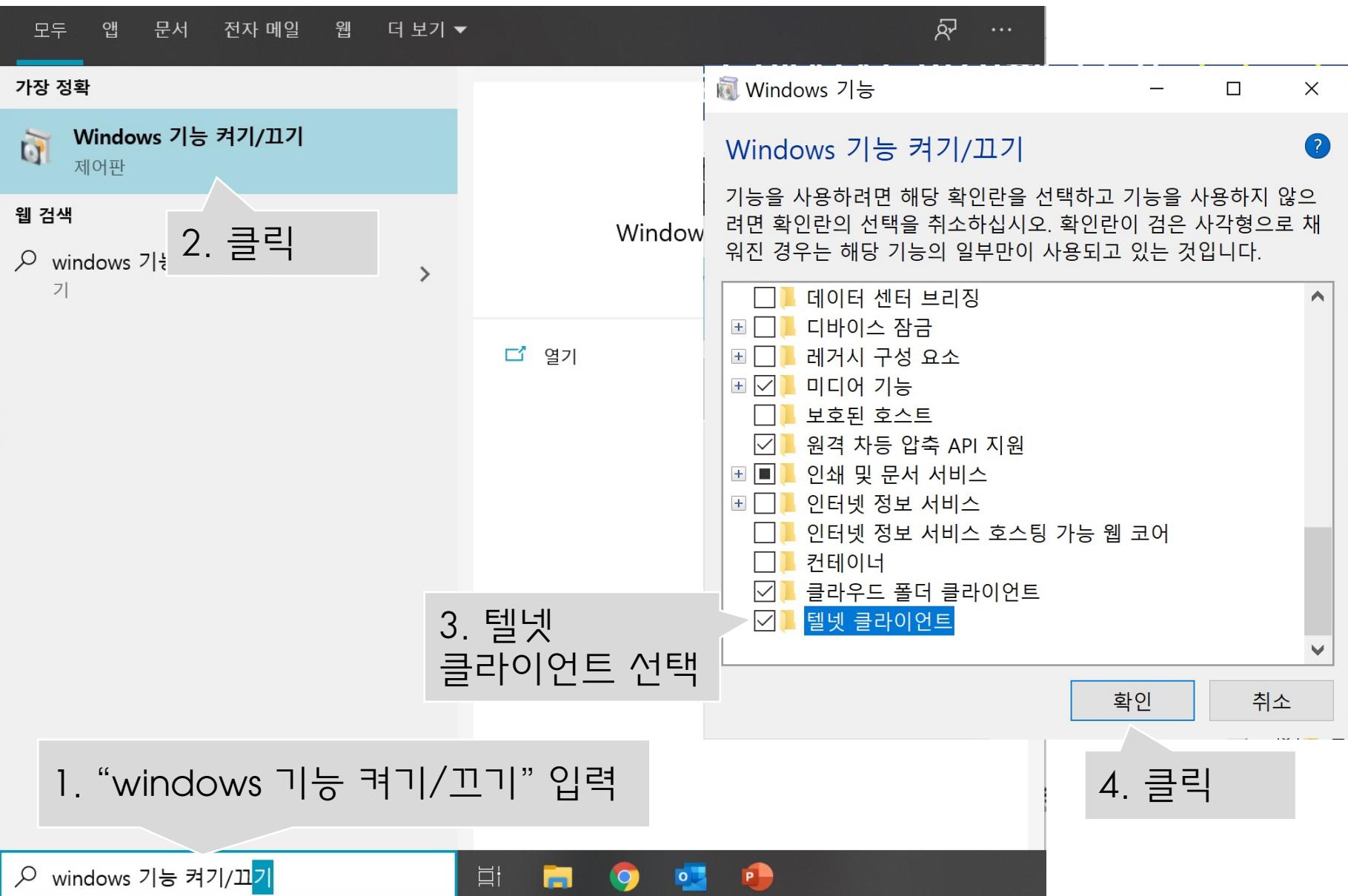
HTTP 응답 메시지 확인해 보기

■ 접속할 사이트

- <http://me.go.kr>



HTTP 응답 메시지 확인해 보기: telnet 이용



HTTP 응답 메시지 확인해 보기: telnet 이용

■ 원하는 웹 서버에 telnet 접속하기 (cmd 창에서 수행)

telnet me.go.kr 80

opens TCP connection to port 80
 (default HTTP server port) at me.go.kr
 anything typed in sent
 to port 80 at me.go.kr

■ GET HTTP 요청 입력

GET / HTTP/1.1

Host: me.go.kr [Enter 2번 입력]

by typing this in (hit carriage return twice), you send
 this minimal (but complete)
 GET request to HTTP server

■ HTTP 서버로부터 수신한 응답 메시지 확인

```
HTTP/1.1 200 OK
Date: Thu, 03 Mar 2022 11:34:57 GMT
Server: Apache
Set-Cookie: elevisor_for_j2ee_uid=554y9drkvr8y4; Expires=Fri, 03-Mar-2023 11:34:57 GMT; Path=/
Set-Cookie: JSESSIONID=Hp9P2gi29Ng0jk9Lm+p2W8nt.mehome2; Path=/
Content-Length: 602
Content-Type: text/html;charset=UTF-8

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ko" lang="ko">
<head>
<title>?罷黜遭□?罷黜?陁?</title>
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta charset="utf-8">
<meta name="description" content="">
<meta name="format-detection" content="telephone=no">
<meta name="viewport" content="width=device-width, height=device-height">
<script type="text/javascript" src="/jquery/jquery-3.1.1.js"></script>
<script type="text/javascript">
//<![CDATA[
location.href = "http://me.go.kr/home/web/main.do";
//]]>
</script>
</head>
<body>
</body>
```

HTTP 응답 메시지 확인해 보기: curl 이용

■ curl Client URL

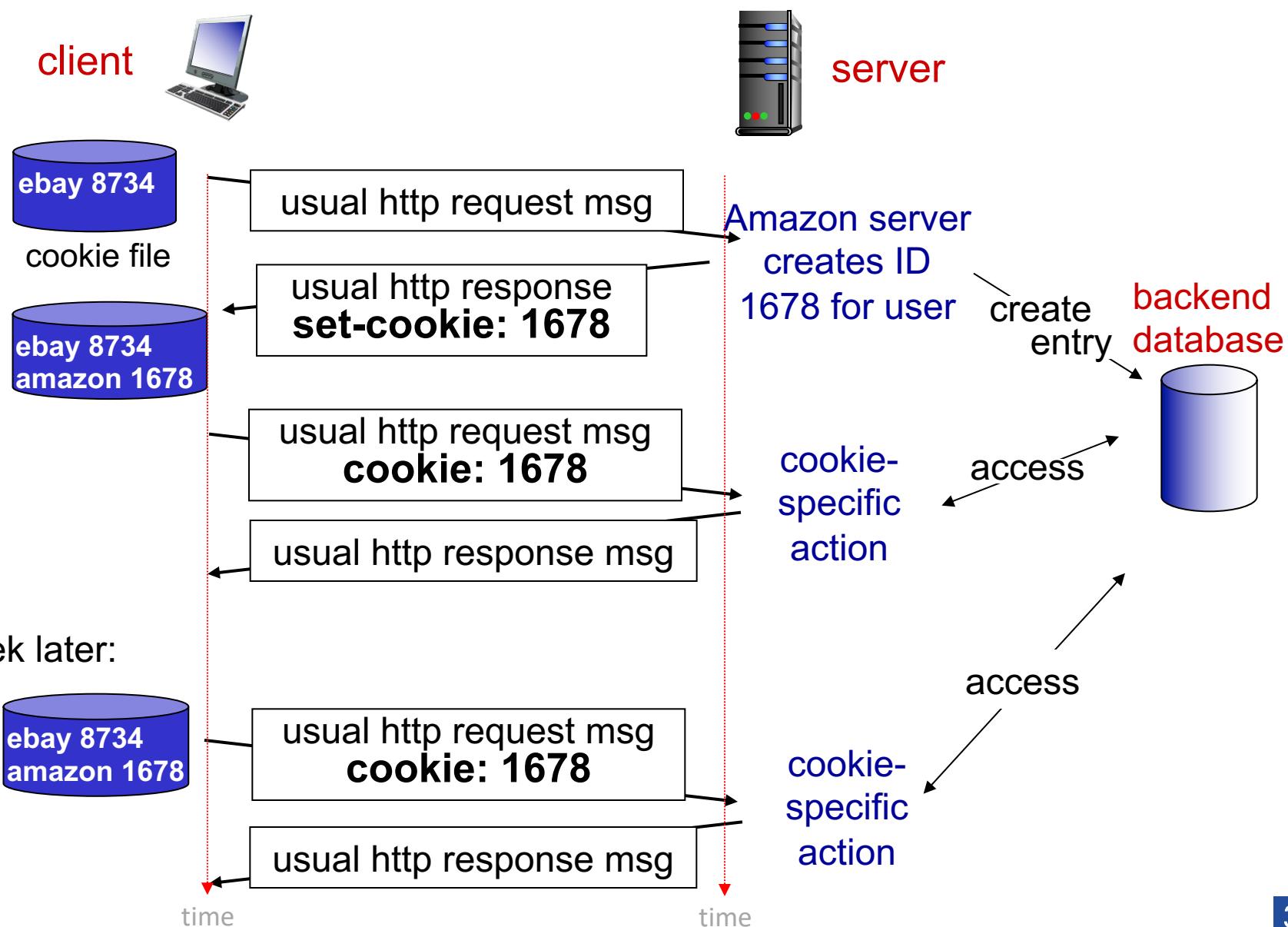
- 명령행 데이터 전송 도구 (command line data transfer tool)
- HTTP/HTTPS/FTP/TELNET/SMTP/POP3 등 주요 프로토콜을 지원
- C 언어 기반 libcurl 라이브러리도 제공

```
C:\Users\DAEHEE KIM>curl -v http://me.go.kr
*   Trying 27.101.216.200:80...
*   Connected to me.go.kr (27.101.216.200) port 80 (#0)
> GET / HTTP/1.1
> Host: me.go.kr
> User-Agent: curl/7.79.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Thu, 03 Mar 2022 11:38:41 GMT
< Server: Apache
< Set-Cookie: elevator_for_j2ee_uid=1dhgxptxs6m32; Expires=Fri, 03-Mar-2023 11:38:41 GMT; Path=/
< Set-Cookie: JSESSIONID=kMBmqUpY8CJbuKdIRVyONBpY.mehome1; Path=/
< Content-Length: 602
< Content-Type: text/html; charset=UTF-8
<
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ko" lang="ko">
<head>
<title>환경부 홈페이지</title>
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta charset="utf-8">
<meta name="description" content="">
<meta name="format-detection" content="telephone=no">
<meta name="viewport" content="width=device-width, height=device-height">
<script type="text/javascript" src="/jquery/jquery-3.1.1.js"></script>
<script type="text/javascript">
//<![CDATA[
location.href = "http://me.go.kr/home/web/main.do";</pre>
```

클라이언트-서버 간의 상태 유지하기: 쿠키 (cookie)

- 웹 사이트와 웹 브라우저는 “**쿠키**”를 사용하여 (약간의) 사용자 상태를 유지하고 관리함
- 쿠키의 4가지 구성 요소
 - HTTP 응답 메시지의 쿠키 헤더 라인
 - 다음 HTTP 요청 메시지의 쿠키 헤더 라인
 - 사용자 호스트에 저장되어 브라우저에 의해 관리되는 **쿠키 파일**
 - 웹 사이트의 백엔드(back-end) 데이터베이스
- 예제
 - 사용자 A는 랩탑에서 브라우저를 이용하여 특정 전자상거래 사이트에 최초 방문함
 - 최초 HTTP 요청이 들어오면, 서버는 다음을 생성
 - ✓ 유일한 **식별번호** (aka “cookie”)
 - ✓ 백엔드 데이터베이스에 식별번호(ID)에 대한 **엔트리**
 - 사용자 A로부터 이 사이트로 오는 **다음 HTTP 요청 메시지들은 cookie를 포함**하고, 서버는 이것을 통해 사용자 A를 식별함

클라이언트-서버 간의 상태 유지하기: 쿠키 (cookie)



쿠키

■ 쿠키의 활용

- 사용자 식별 확인 (권한 부여)
- 쇼핑 카트
- 제품 추천
- 사용자 세션 상태 (웹 기반 e-mail)

■ 상태 저장 방법

- 프로토콜 종단점
 - ✓ 여러 트랜잭션 동안 송신자와 수신자에 상태가 저장됨
- 쿠키: HTTP 메시지가 상태를 전송

참고

쿠키와 privacy

- 쿠키는 사이트가 사용자에 대해 많은 것을 알도록 함
- 쿠키 기반 사용자 식별을 통해 사용자의 다양한 개인 정보(쇼핑 정보, 검색 정보 등)가 노출될 수 있음

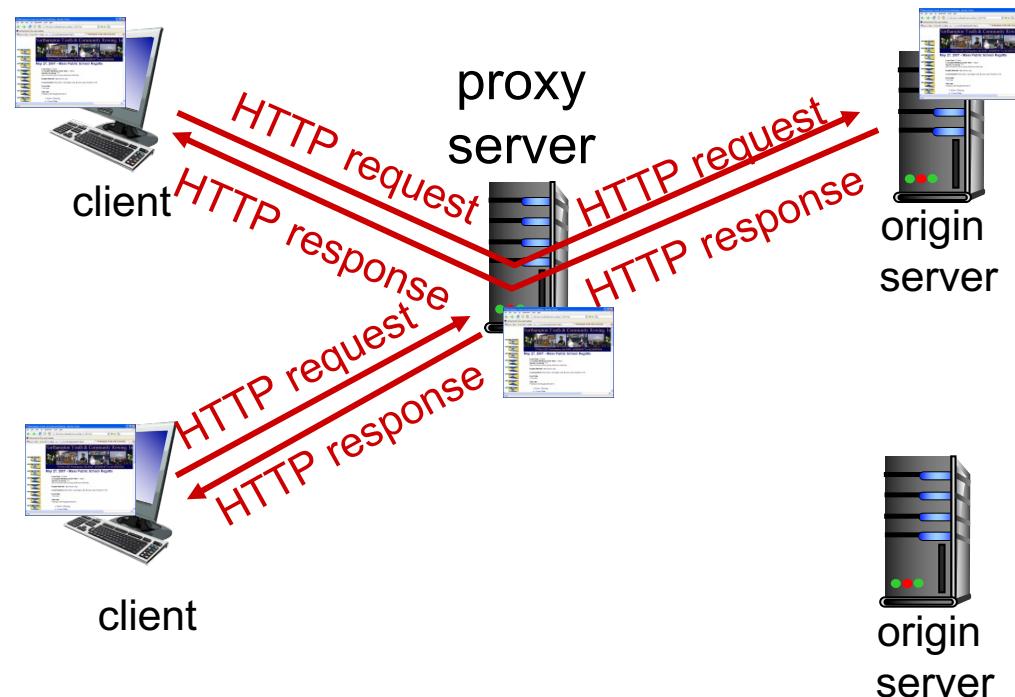
웹 캐시 (프록시 서버)

■ 웹 캐시 web cache는 원래 original 웹 서버의 관여없이 클라이언트의 HTTP 요청을 처리해주는 네트워크 개체

- 프록시 서버 proxy server 라고도 함
- 사용자는 웹 캐시를 사용하고 싶은 경우 브라우저에 설정하여야 함

■ 동작 방법

- 브라우저는 웹 캐시와 연결을 설정하고 웹 캐시에 모든 HTTP 요청을 전송
- 웹 캐시에 객체가 있으면 객체를 전송
- 없으면, 웹 캐시가 원래 서버에 객체를 요청하여 가져온 후 클라이언트에 전송하고 캐시에 저장



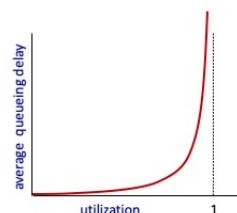
웹 캐시 (프록시 서버)

- 웹 캐시는 “**클라이언트**”이자 “**서버**”로 동작
 - 클라이언트(브라우저)에 대해서는 서버
 - 원래 서버에 대해서는 클라이언트
- 일반적으로, 웹 캐시는 ISP(대학, 회사, 기관)에 의해 설치됨
- 웹 캐싱 *web caching*의 이점
 - 클라이언트 요청에 대한 **응답 시간 감소**
 - ✓ 일반적으로 웹 캐시가 원래 서버보다 사용자에게 더 가까이 있으므로
 - 기관 액세스 링크의 **웹 트래픽 감소**
 - 인터넷은 많은 수의 웹 캐시를 가지고 있음
 - ✓ 컨텐츠 제공자가 저속도의 접속 회선을 가진 느린 서버에서 사이트를 운영하더라도 **효과적으로(빠르게) 컨텐츠를 전달**할 수 있게 해줌

웹 캐시 예제

■ 가정

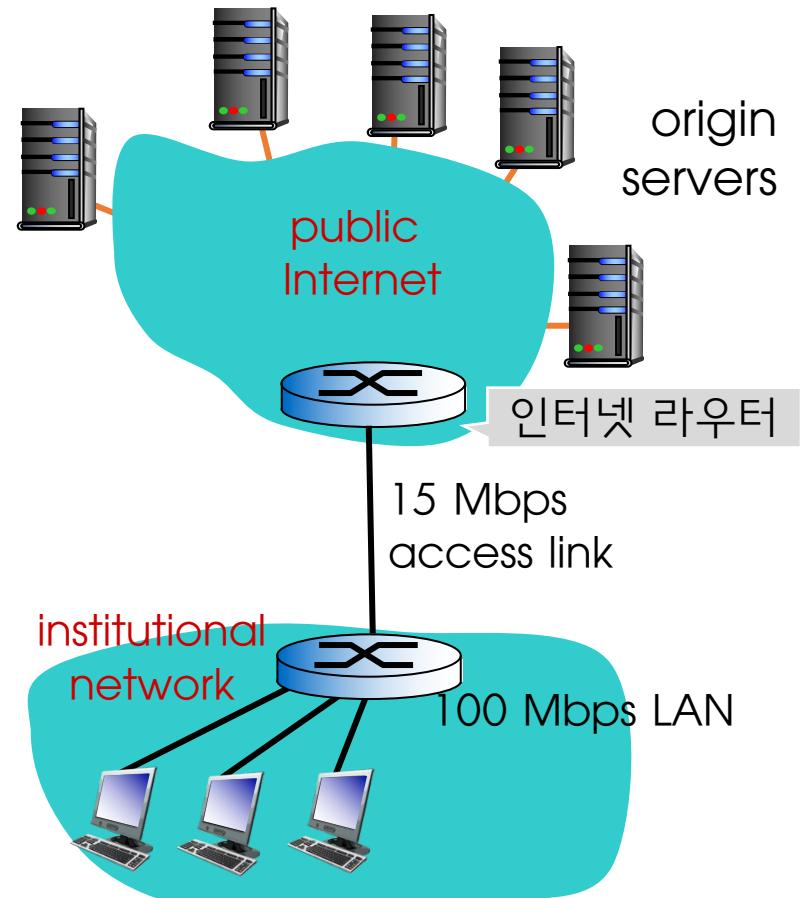
- 평균 객체 크기: 1 Mbits
- 평균 요청 속도: 15 요청/초
- 평균 전송 속도: 15 Mbps
- 인터넷 라우터에서 원래 서버 간 RTT : 2 sec
- 액세스 링크 속도: 15 Mbps



문제 발생

■ 결과

- LAN 이용율: 15%
- 액세스 링크 이용율 = 100% (100% is circled in blue)
- 전체 지연 = 인터넷 지연 + 액세스 링크 지연 + LAN 지연
 $= 2 \text{ 초} + \text{수 분(minutes)} + \text{수십 밀리미터초(msecs)}$



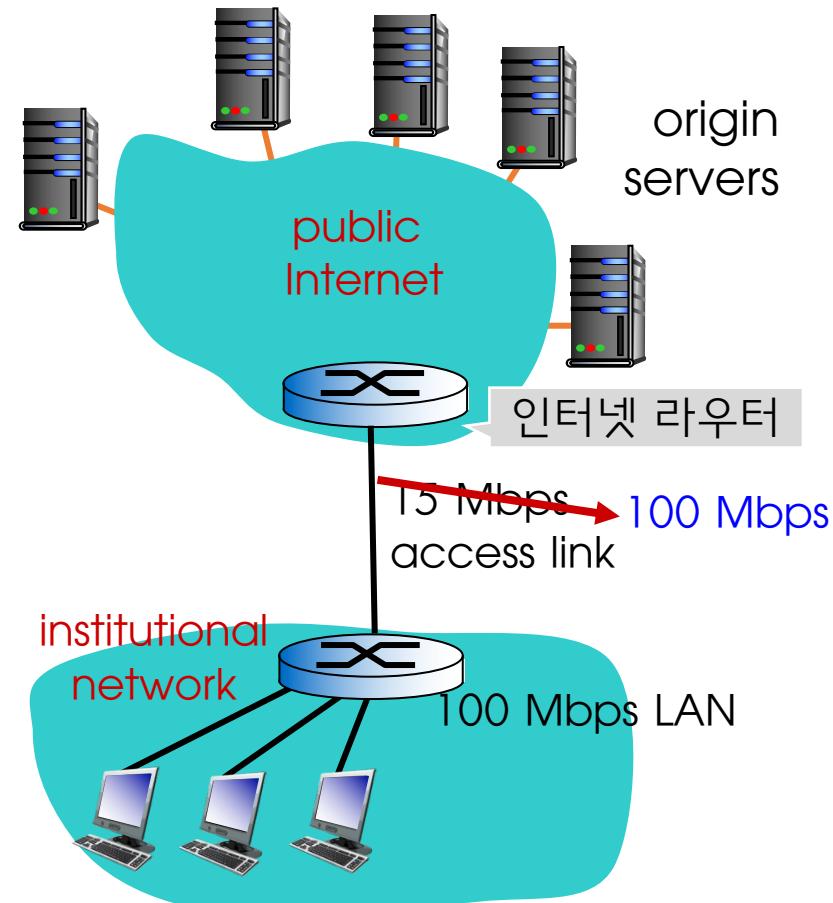
해결책 1: 액세스 링크 업그레이드

■ 가정

- 평균 객체 크기: 1 Mbits
- 평균 요청 속도: 15 요청/초
- 평균 전송 속도: 15 Mbps
- 인터넷 라우터에서 원래 서버 간 RTT : 2 sec
- 액세스 링크 속도: ~~15 Mbps~~ → 100 Mbps

■ 결과

- LAN 이용율: 15%
- 액세스 링크 이용율 = ~~100%~~ → 15%
- 전체 지연 = 인터넷 지연 + **액세스 링크 지연** + LAN 지연
 $= 2 \text{ 초} + \cancel{\text{수분}}(\text{minutes}) + \text{수십 밀리미터초}(msecs)$



비용

액세스 링크 속도 업그레이드
비용은 매우 비쌈!

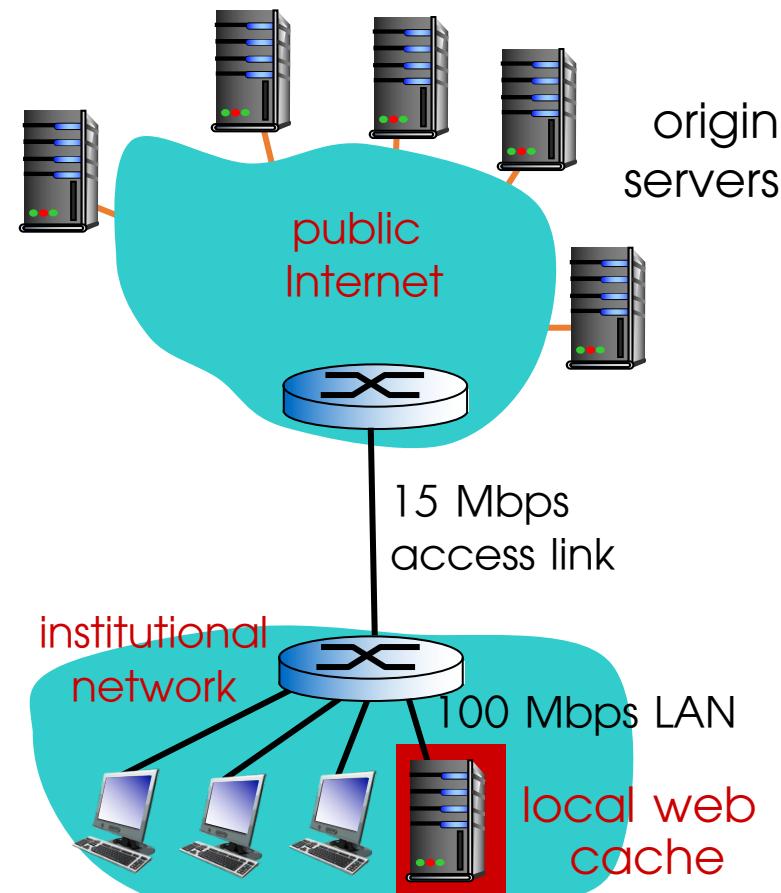
해결책 2: 웹 캐시 설치

■ 가정

- 평균 객체 크기: 1 Mbits
- 평균 요청 속도: 15 요청/초
- 평균 전송 속도: 15 Mbps
- 인터넷 라우터에서 원래 서버 간 RTT : 2 sec
- 액세스 링크 속도: 15 Mbps

■ 결과

- LAN 이용율: 15%
- 액세스 링크 이용율 = ?
- 전체 지연 = ?

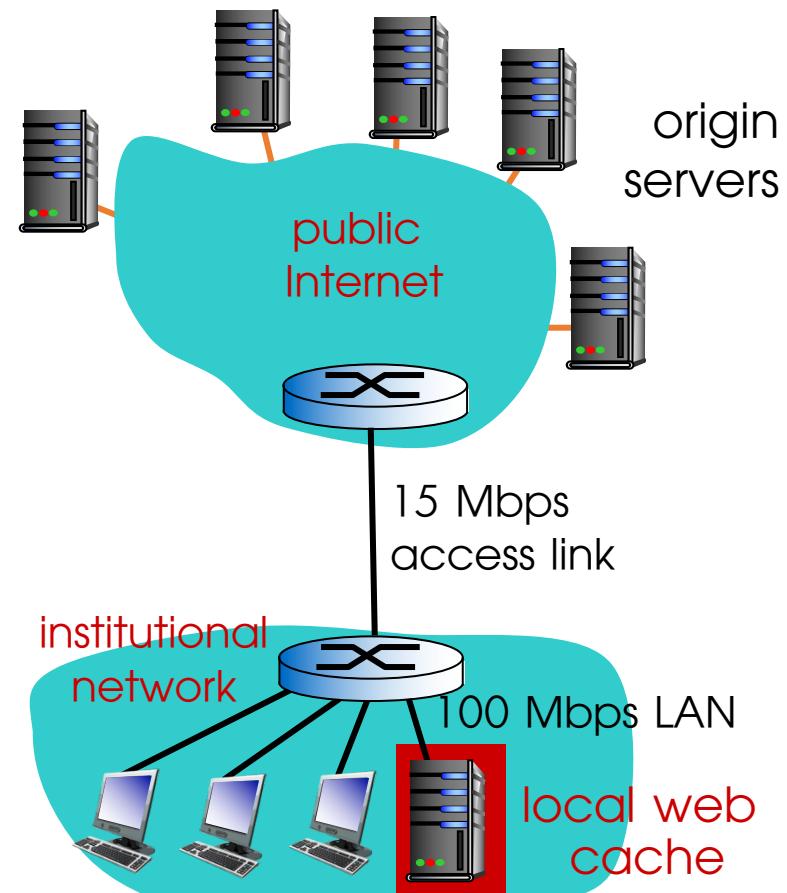


비용
웹 캐시 (쌈!)

해결책 2: 웹 캐시 설치

■ 액세스 링크 이용율, 지연

- 캐시 적중율 hit rate 을 0.4로 가정
 - ✓ 웹 캐시에서 40% 요청을 처리하고, 원래 서버에서 60%의 요청을 처리
- 액세스 링크 이용율
 - ✓ 60%의 요청이 액세스 링크 사용
 - ✓ 액세스 링크를 통한 데이터 속도
 - $0.6 * 15 \text{ Mbps} = 9 \text{ Mbps}$
 - ✓ 이용율 = $9/15 = 0.6$
 - 이 경우, 일반적으로 수십 ms의 지연 발생
 - 여기서는 10ms라 가정
- 전체 지연
 - ✓ $0.6 * (\text{원래 서버와의 지연}) + 0.4 * (\text{웹 캐시와의 지연})$
 $= 0.6 \times (2.01) + 0.4 \times (\sim \text{msecs})$
 $= \sim 1.2 \text{ secs}$
 - ✓ 100 Mbps로 액세스 링크 업그레이드한 것보다 지연 시간이 적음. (가격도 쌈!)



조건부 GET Conditional GET

- 웹 캐시의 객체들이 최신 버전이면 서버가 객체를 보내지 않음

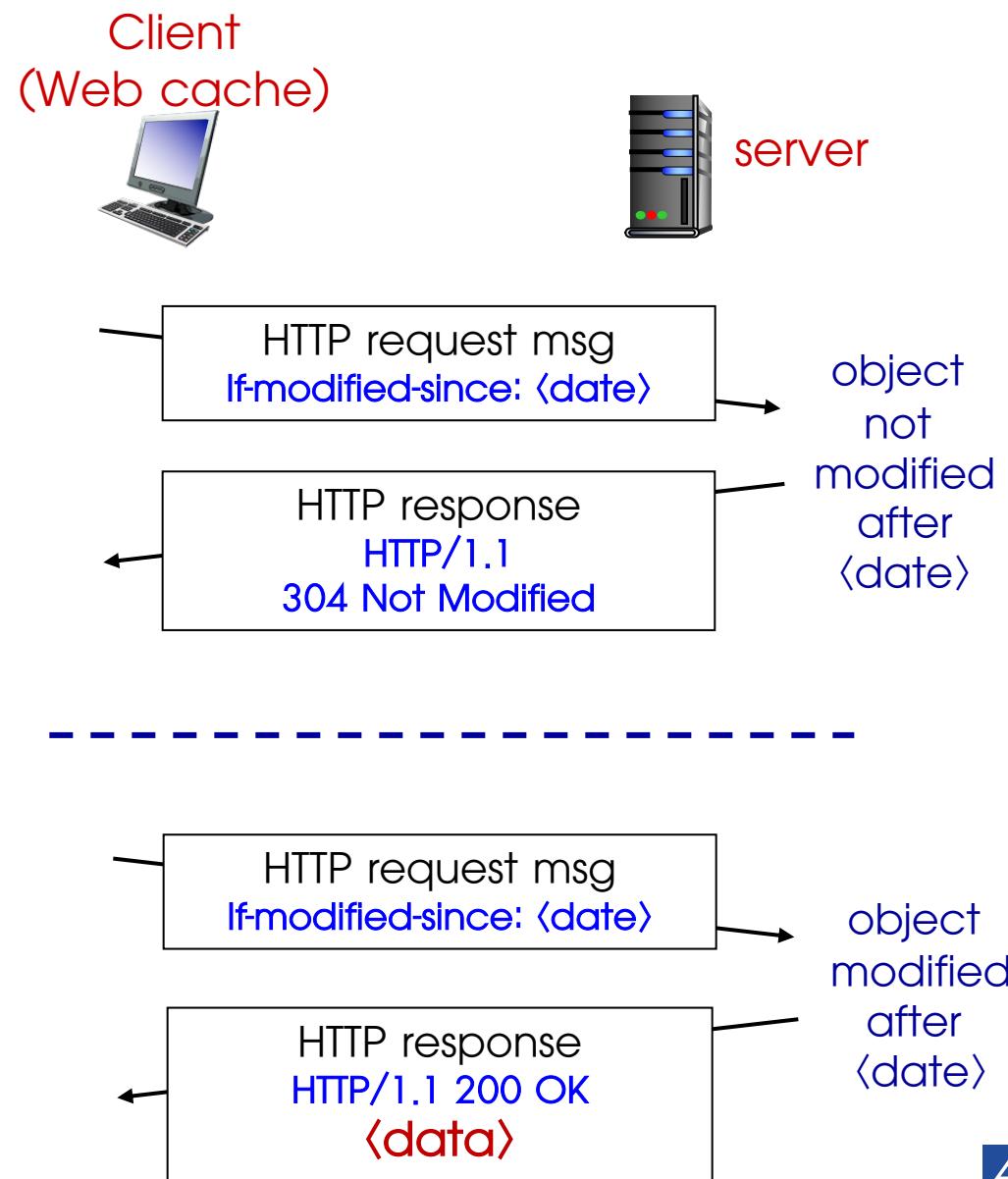
- 객체 전송 지연 시간이 필요 없음
- 링크 이용율을 낮춤

■ 클라이언트(웹 캐시)

- HTTP 요청에 캐시된 내용의 날짜를 기술
 - ✓ **If-modified-since: <date>**

■ 서버

- 클라이언트가 최신 객체를 가지고 있으면 객체가 생략된 응답 메시지를 전송
 - ✓ **HTTP/1.0 304 Not Modified**



HTTP/2

■ Goal

- 여러 객체를 가져오는 HTTP 요청 시 걸리는 시간을 최소화하는 것

■ HTTP/1.1

- 1개의 TCP 연결에서 여러 개의 연속된 GET을 사용하는 persistent HTTP 지원
- HTTP 요청 메시지에 대해 서버는 순서대로 응답 (FCFS)
 - ✓ FCFS: First-Come-First-Served 스케줄링
- 따라서, 큰 객체 뒤에 있는 작은 객체는 전송을 위해 기다려야 함
 - ✓ head-of-line(HOL) blocking이라고 함

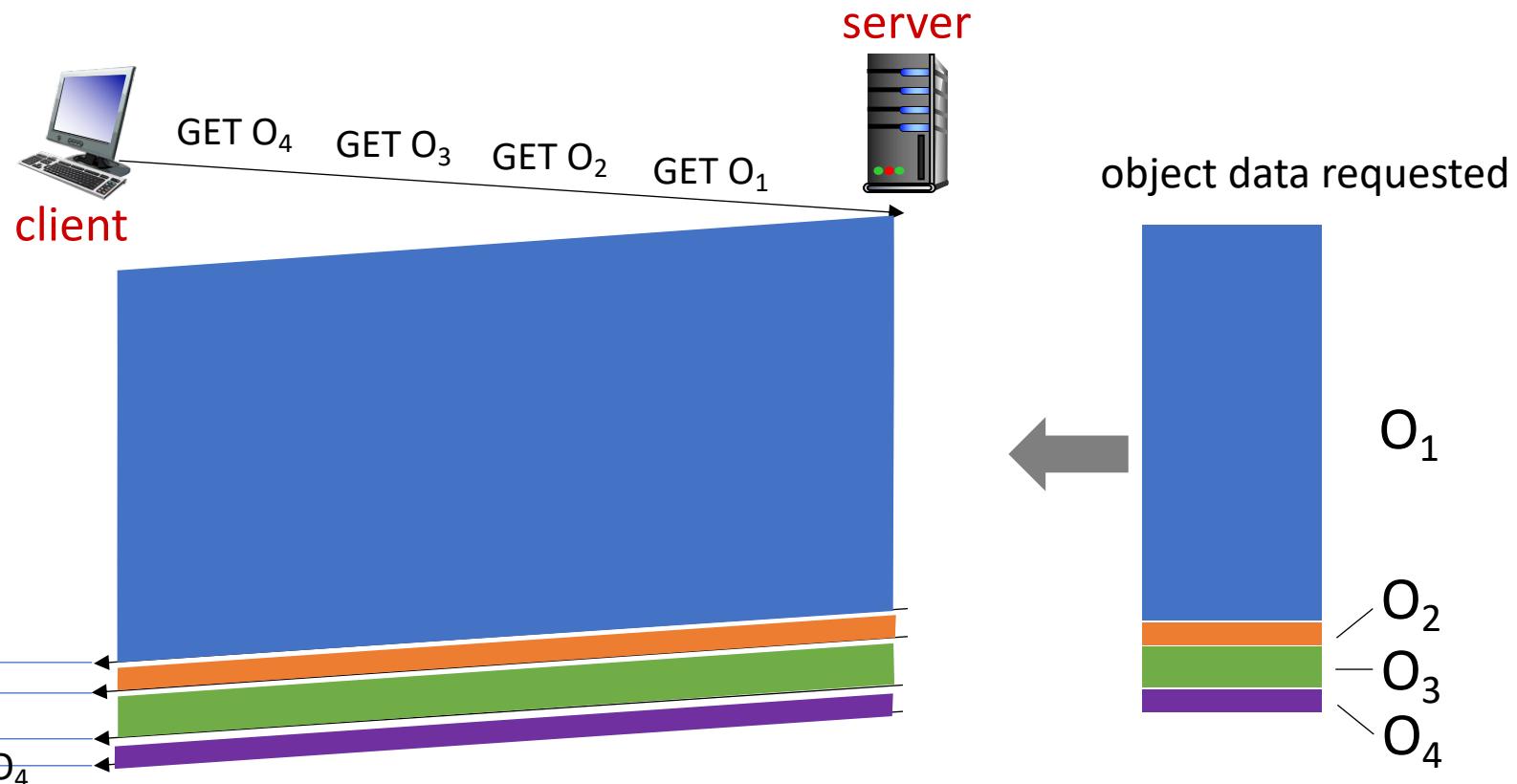
■ HTTP/2

- 서버에서 클라이언트로 객체 전송 시, 유연성 **flexibility**이 증가됨
- 메소드, 상태 코드, 헤더 등은 HTTP/1.1과 동일
- FCFS가 아니라 클라이언트의 요청 순서에 따라 객체 순서 변경 가능
- 클라이언트가 요청하지 않은 객체도 전송 가능
- 객체를 프레임 단위로 나누어 전송함으로써 HOL blocking을 줄일 수 있음

HTTP/2: HOL blocking 줄이기

■ HTTP/1.1

- 가정) 클라이언트가 1개의 큰 객체(비디오 파일)와 3개의 작은 객체를 요청

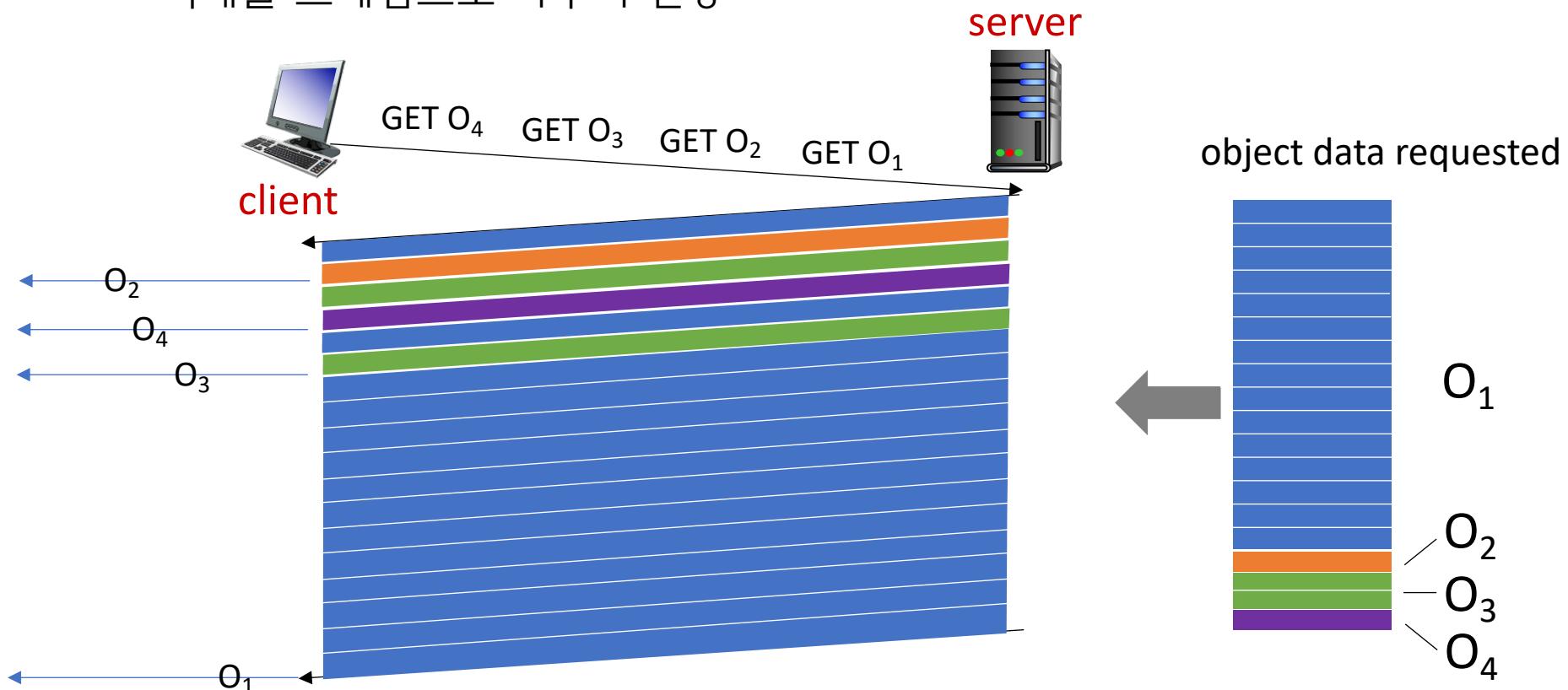


요청 순서대로 객체가 전송됨: O₁ 뒤에 O₂, O₃, O₄ 가 대기함

HTTP/2: HOL blocking 줄이기

■ HTTP/2

- 객체를 프레임으로 나누어 전송



O_2, O_3, O_4 는 빠르게 전송, O_1 는 약간 지연

요약

- 응용 프로그램 구조
 - “클라이언트-서버” vs “P2P”
- 응용 프로그램 서비스 요구사항
 - 신뢰성, 처리율, 지연시간
- 인터넷 전송 서비스 모델
 - 연결지향적, 신뢰성: TCP
 - 비신뢰성: UDP
- 대표적 응용 계층 프로토콜
 - HTTP