Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail
- SMTP, POP3, IMAP

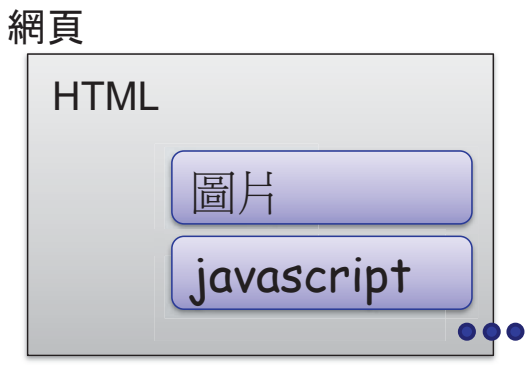
2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

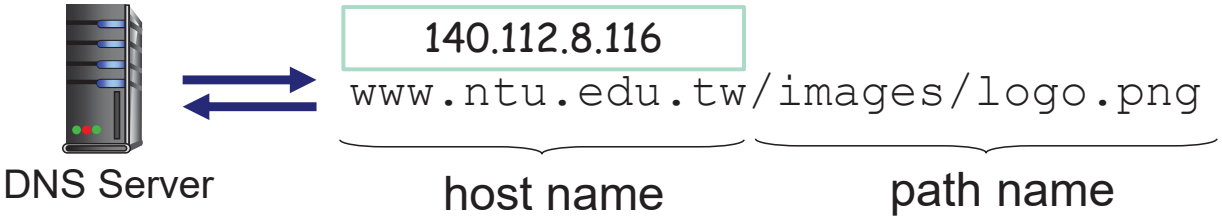
2.7 socket programming with UDP and TCP

Web and HTTP

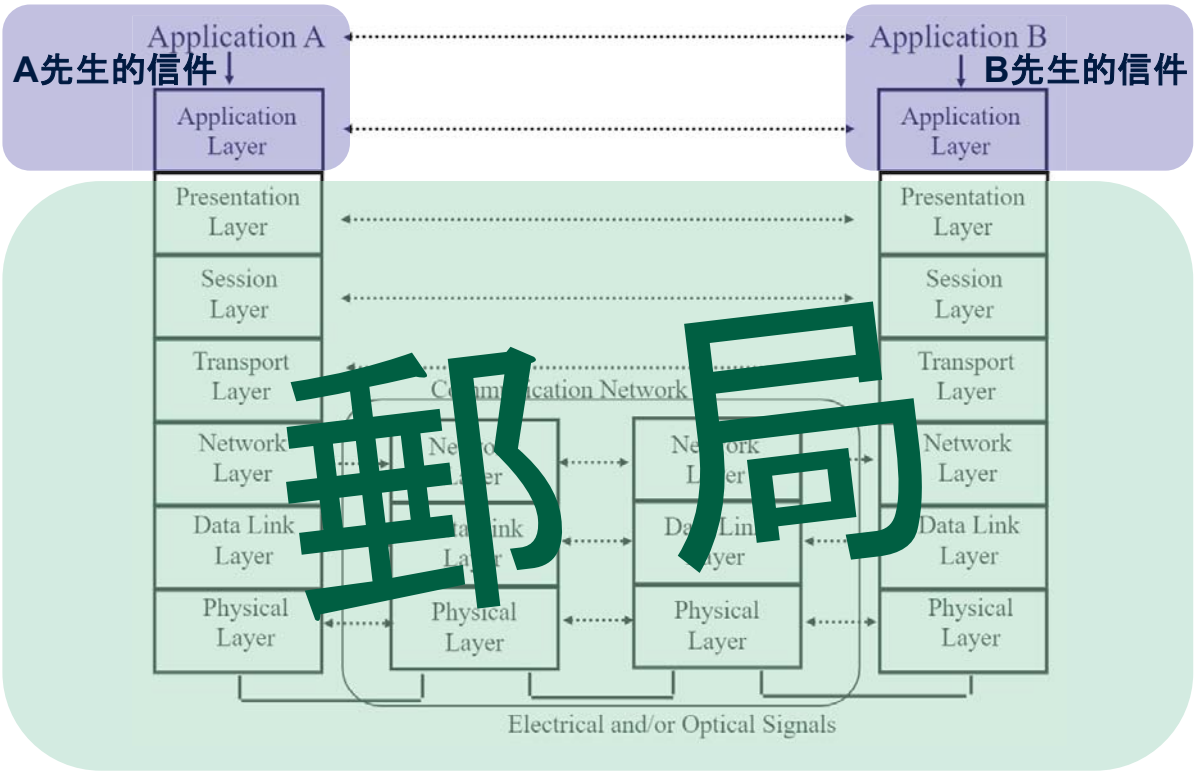


First, a review…

- *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.,



Protocol

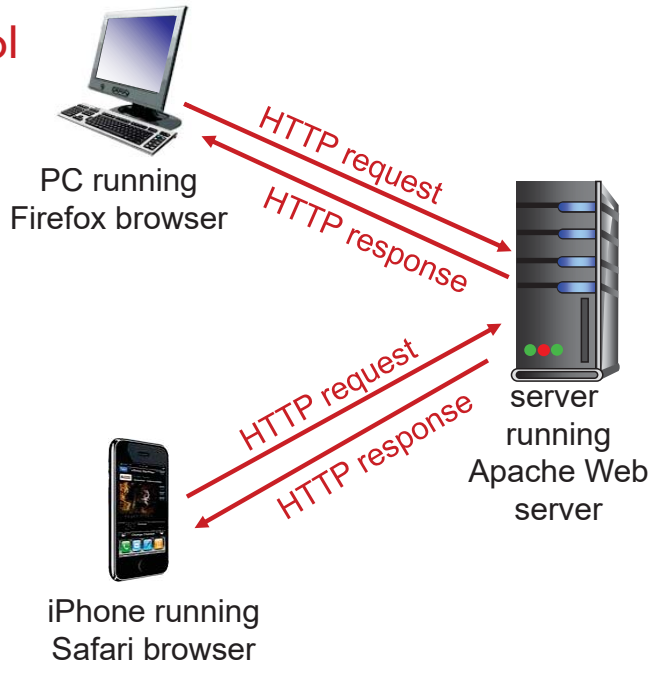


https://slideplayer.com/slide/8326489/

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



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

HTTP is "stateless"

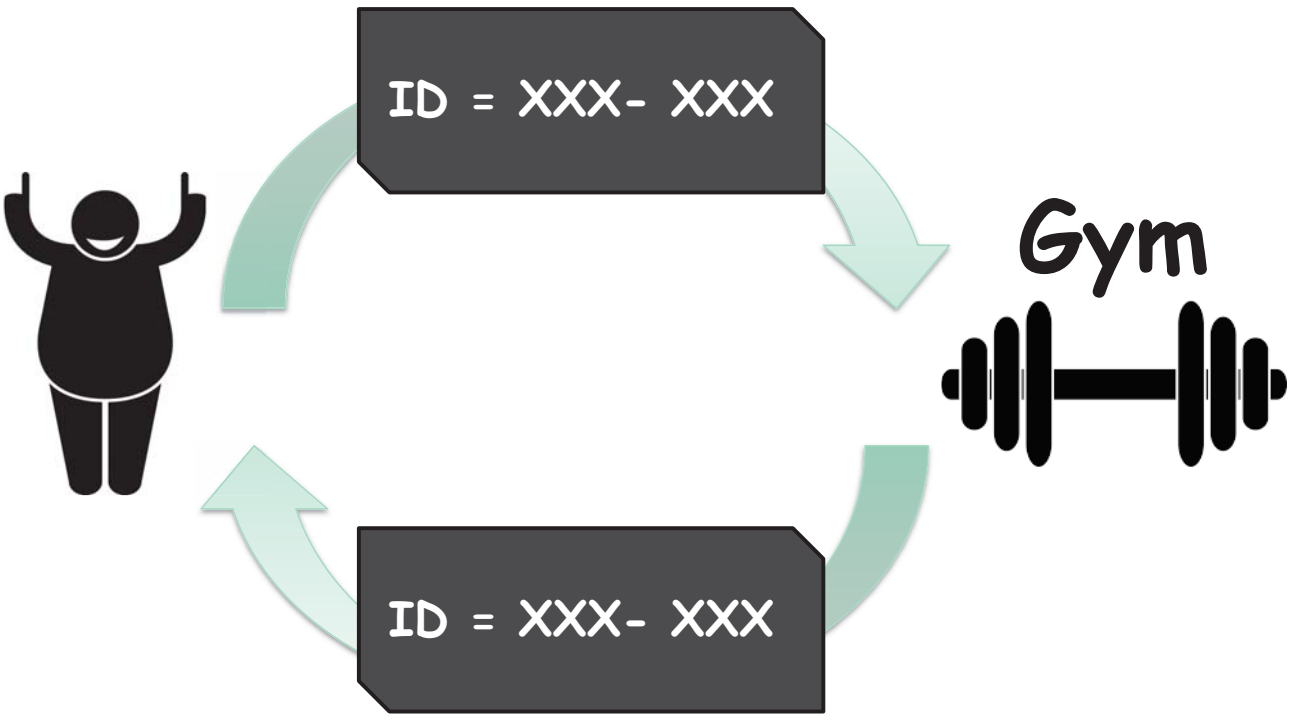
- server maintains no information about past client requests

aside

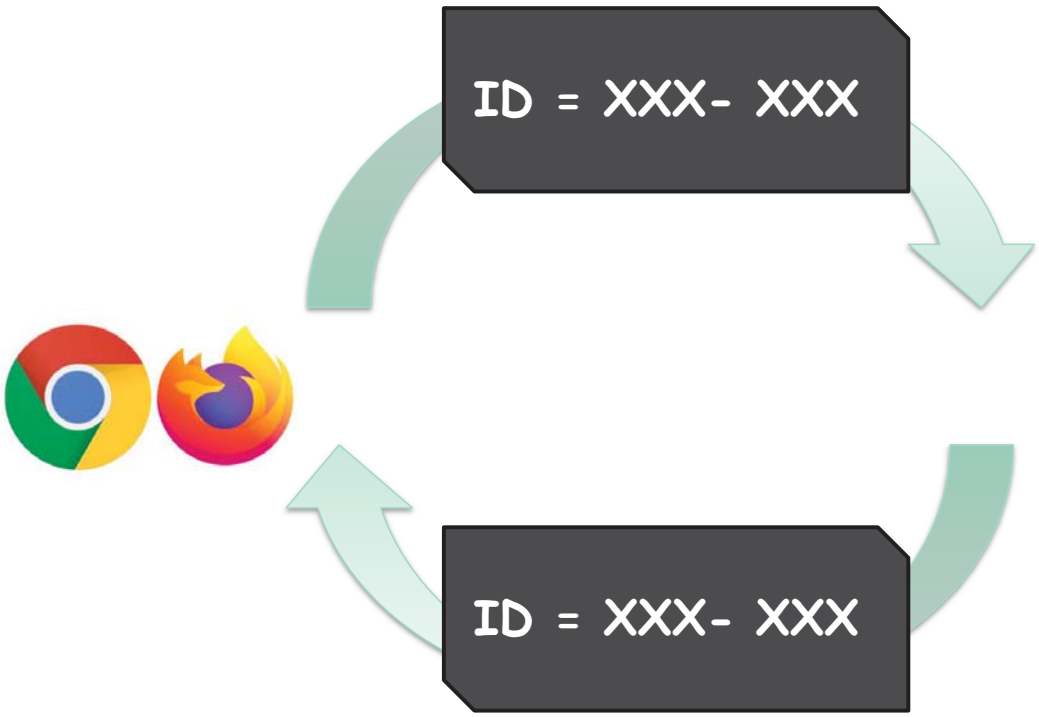
protocols that maintain "state" are complex!

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

Cookies and Session



Cookies and Session



運作流程

開啟 TCP connection

- 用來發送request和接受response。
- 一個到多個的連線。
- 可重複使用既有的連線。

接收
HTTP response

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

<h1>Hello, World!</h1>
```

開始 發送 接收 結束

發送
HTTP request

```
GET /example/ HTTP/1.1
Host: test.kaihao.top
Accept-Language: zh-TW
```

關閉連線
或 留著重複使用

Three-way Handshake耗時間，所以等等還要用的話可以先留著。

格式

接著將分別針對HTTP Request與HTTP Response做說明

HTTP Request

Path

欲存取的資源位置，

可由URL中擷取出來。

HTTP Method

GET：查詢資料。

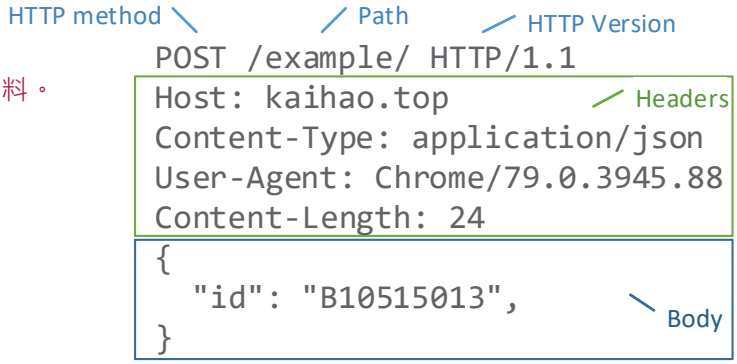
POST：新增資料。

PUT/PATCH：修改資料。

DELETE：刪除資料。

HTTP Version

標示所使用的HTTP版本



Headers

記錄一些資訊讓server知道

功能說明	範例
指定伺服器的domain及port。	`Host: kaihao.top:80`
指定可接受的response的自然語言。	`Accept-Language: en-US`
指定可接受的資料內容類型。	`Accept: text/plain`
標示Body的MIME類型。	`Content-Type: application/json`
標示發送請求者的相關資訊。	`User-Agent: Chrome/79.0.3945.88`

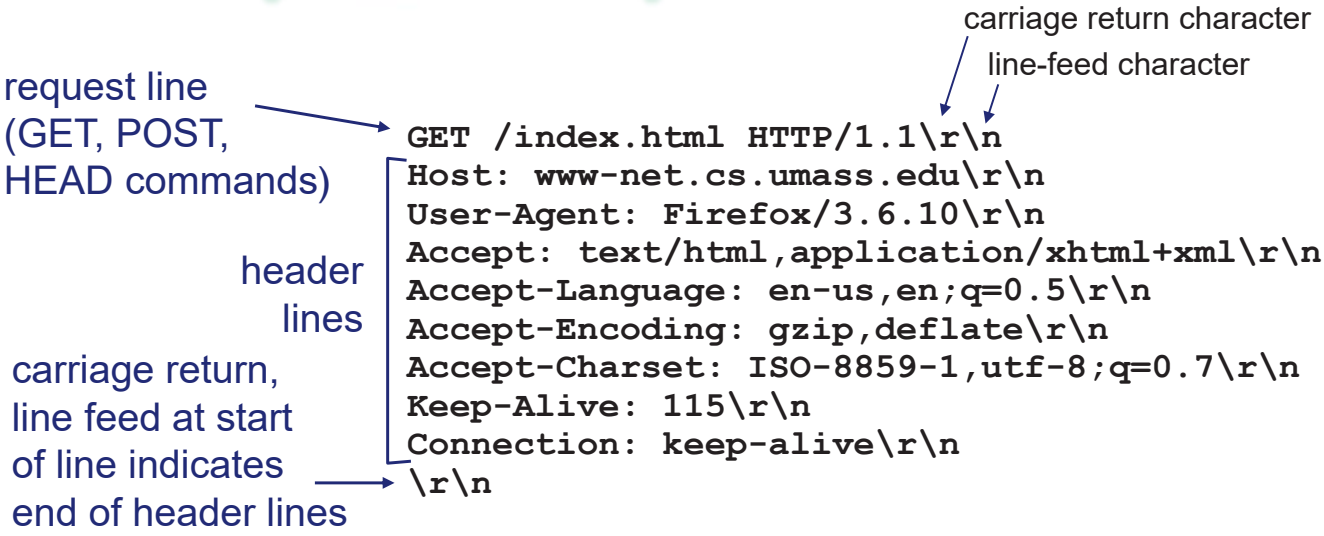
Body

讓某些HTTP method

可以傳遞資料

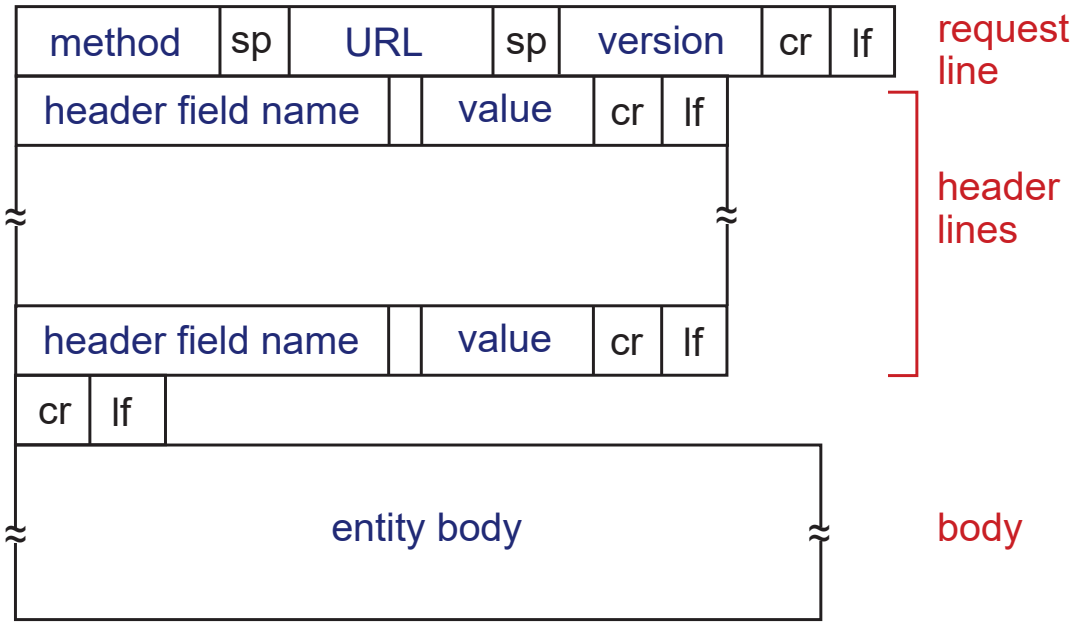
HTTP request message

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



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

HTTP request message: general format



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:

```
http://host.cc.ntu.edu.tw
        /sec/schinfo/epaper/article.asp?num=1448&sn=17618
```


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

Status Code

狀態碼	說明
1XX	資訊的通知與回應。
2XX	請求被成功的處理與回復。
3XX	重導向告知。
4XX	Client端的請求有誤。
5XX	Server端發生錯誤。

HTTP Version
標示所使用的HTTP版本

Status Message
一段對於狀態碼的簡短描述。

HTTP Version \ status code \ / status message

```
HTTP/1.1 200 OK                                   / Headers
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/10.0
Date: Sat, 04 Jan 2020 15:51:22 GMT
Content-Length: 3369

<!DOCTYPE html>                                   \ Body
  ...(剩下的HTML Document部分)
```

Headers
記錄一些資訊讓client知道

Body
放置client所請求的資源內容

功能說明	範例
紀錄這則回應的MIME類型。	Content-Type: text/html; charset=utf-8
伺服器的名稱。	Server: Apache/2.4.1 (Unix)
回應的Body共有幾個bytes。	Content-Length: 348
紀錄回應傳送的日期與時間。	Date: Sun, 05 Jan 2020 03:32:53 GMT

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 ...
```

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

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

400 Bad Request
- request msg not understood by server

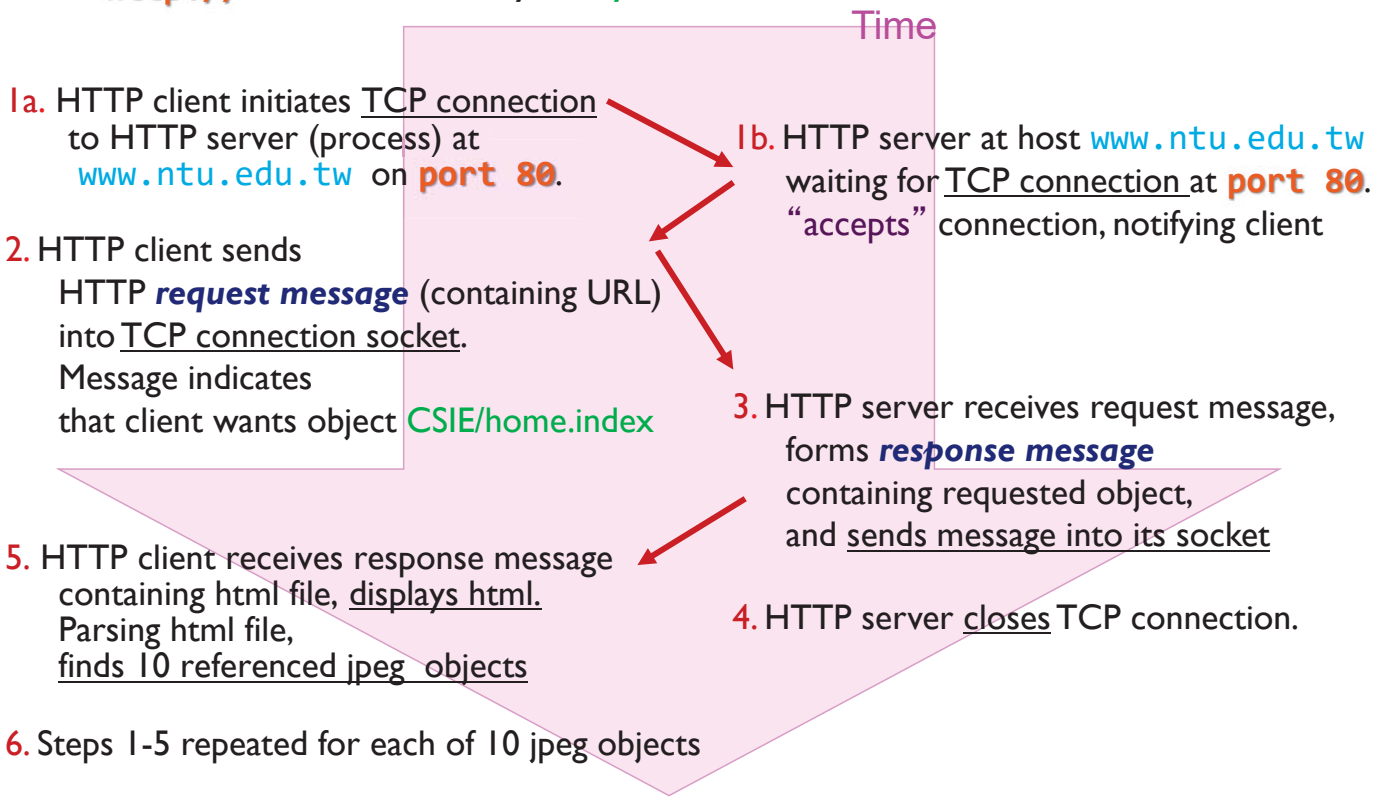
404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported
- the HTTP version used in the request is not supported by the server.

Non-persistent HTTP

suppose user enters URL: ↓ (contains text, references to 10 jpeg images)

`http://www.ntu.edu.tw/CSIE/home.index`



HTTP connections

non-persistent HTTP	persistent HTTP
at most one object sent over TCP connection (connection then closed) (downloading multiple objects required multiple connections)	multiple objects can be sent over single TCP connection between client, server

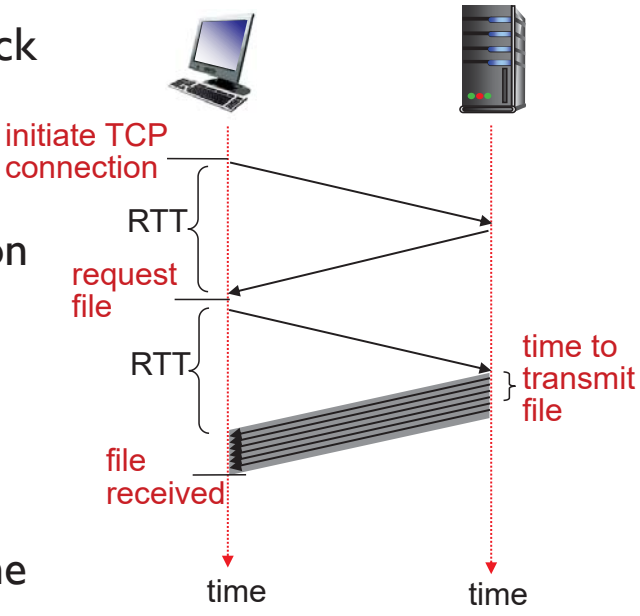
Non-persistent HTTP: **response time**

RTT (Round Trip Time)

def: 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

persistent HTTP:

- 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

Hint :

Non-persistent

Parallel

Persistent

Persistent+pipeline

...

實　作

本節會先使用現成的軟體做示範
接著利用 Socket 發送與接收 HTTP 訊息

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet gaia.cs.umass.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at gaia.cs.umass. edu. anything typed in will be sent to port 80 at gaia.cs.umass.edu

2. type in a GET HTTP request:

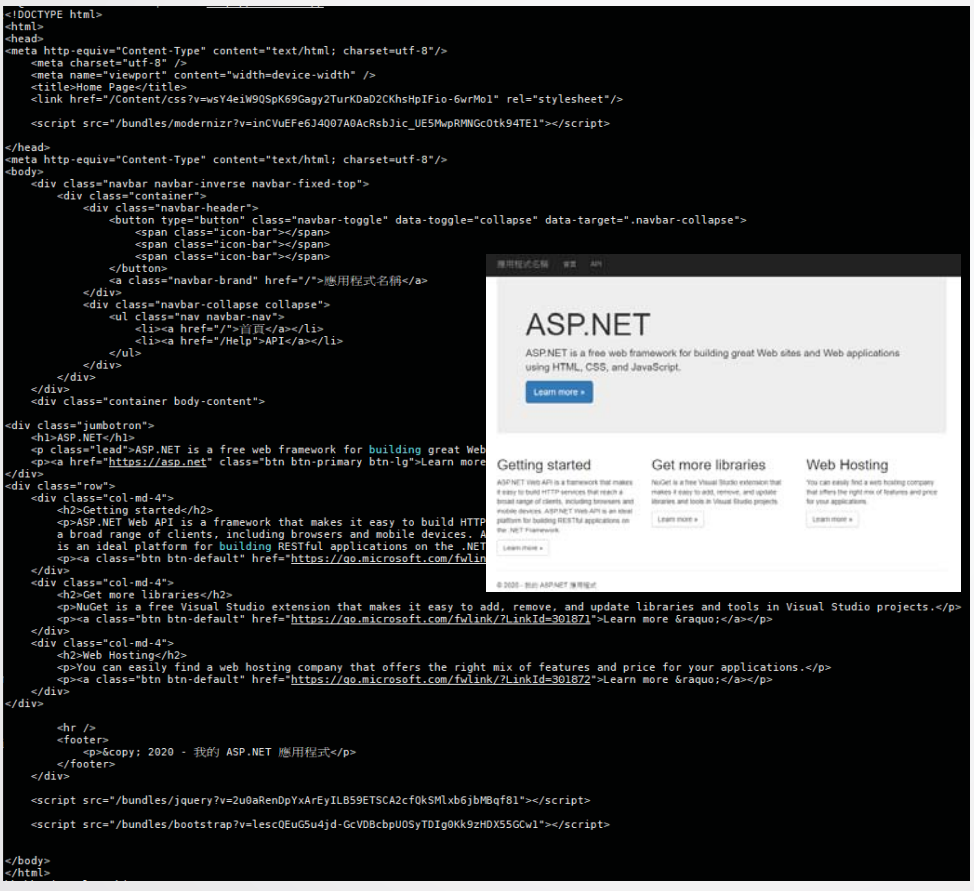
```
GET /kurose_ross/interactive/index.php HTTP/1.1
Host: gaia.cs.umass.edu
```

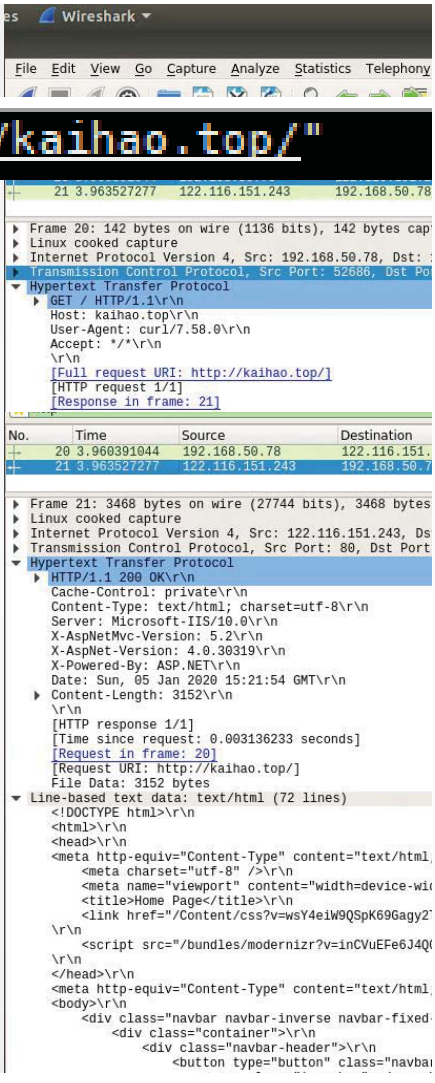
by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!
(or use Wireshark to look at captured HTTP request/response)

使用cURL

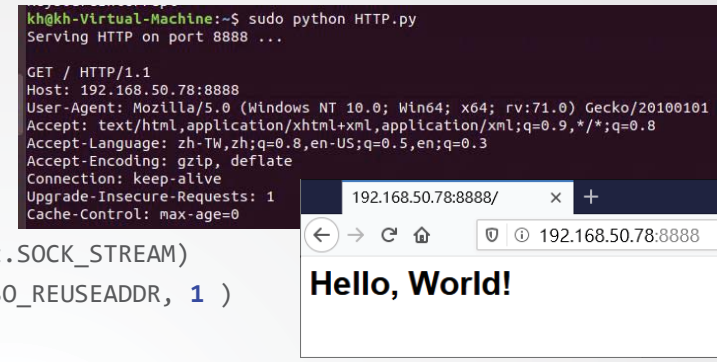
```
kh@kh-Virtual-Machine:~$ curl "http://kaihao.top/"
```

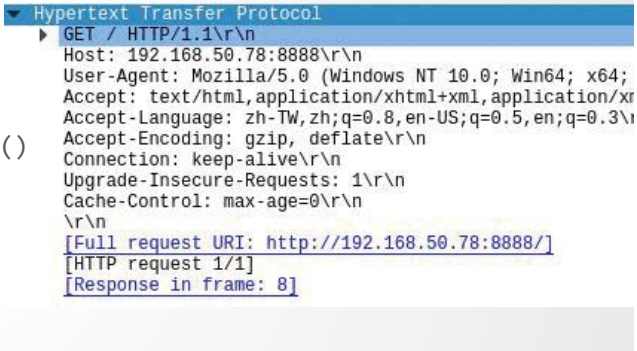


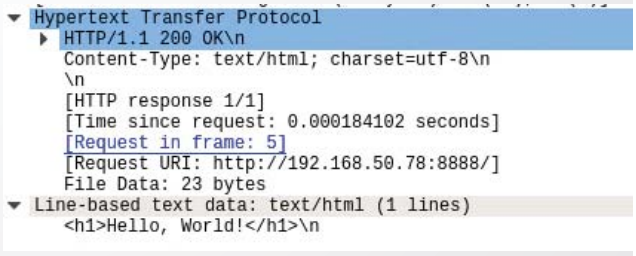


利用 Socket 接收 HTTP request

```
import socket

mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mySocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1 )
mySocket.bind( ('192.168.50.78',8888) )
mySocket.listen(1)
print('Serving HTTP on port 8888 ...\n')

while  True :
    clientConnection, clientAddress = mySocket.accept()
    request = clientConnection.recv( 1024 )
    print(request)

    http_response = """\
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

<h1>Hello, World!</h1>
"""
    clientConnection.sendall(http_response)
    clientConnection.close()
```







利用 Socket 發送 HTTP request

```
import socket

mySocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
mySocket.connect(('kaihao.top',80))
mySocket.send('''\
GET / HTTP/1.1
Host: kaihao.top

''')

data = mySocket.recv(1024)
while (len(data) > 0):
    print(data)
    data = mySocket.recv(1024)
```

```
khgkh-Virtual-Machine:~$ sudo python HTTP2.py
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/10.0
X-AspNetMvc-Version: 5.2
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Sun, 05 Jan 2020 16:48:43 GMT
Content-Length: 3152

<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>Home Page</title>
    <link href="/Content/css?v=wsY4eiW9QSpK69Gagy2TurKDaD2CKhsHp
    <script src="/bundles/modernizr?v=inCVuEFe6J4Q07A0AcRsbJic_U
</head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data
                    <sp
an class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="/">應用程式名稱</a
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li><a href="/">首頁</a></li>
                    <li><a href="/Help">API</a></li>
                </ul>
            </div>
```

```
▼ Hypertext Transfer Protocol
  ▶ GET / HTTP/1.1\n
    Host: kaihao.top\n
    \n
    [Full request URI: http://kaihao
    [HTTP request 1/1]
    [Response in frame: 29]
```

```
▼ Hypertext Transfer Protocol
  ▶ HTTP/1.1 200 OK\r\n
    Cache-Control: private\r\n
    Content-Type: text/html; charset=utf-8\r\n
    Server: Microsoft-IIS/10.0\r\n
    X-AspNetMvc-Version: 5.2\r\n
    X-AspNet-Version: 4.0.30319\r\n
    X-Powered-By: ASP.NET\r\n
    Date: Sun, 05 Jan 2020 16:48:43 GMT\r\n
  ▶ Content-Length: 3152\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.002988127 seconds]
    [Request in frame: 28]
    [Request URI: http://kaihao.top/]
    File Data: 3152 bytes
▼ Line-based text data: text/html (72 lines)
    <!DOCTYPE html>\r\n
    <html>\r\n
    <head>\r\n
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>\r\n
```

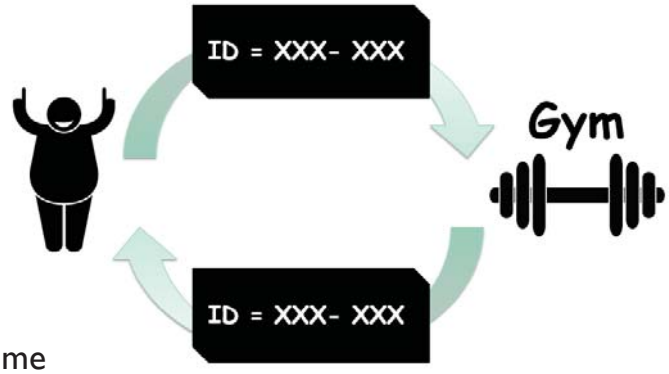
User-server state: cookies

many Web sites use cookies

four components:

1) cookie header line of HTTP *response* message
2) cookie header line in next HTTP *request* message

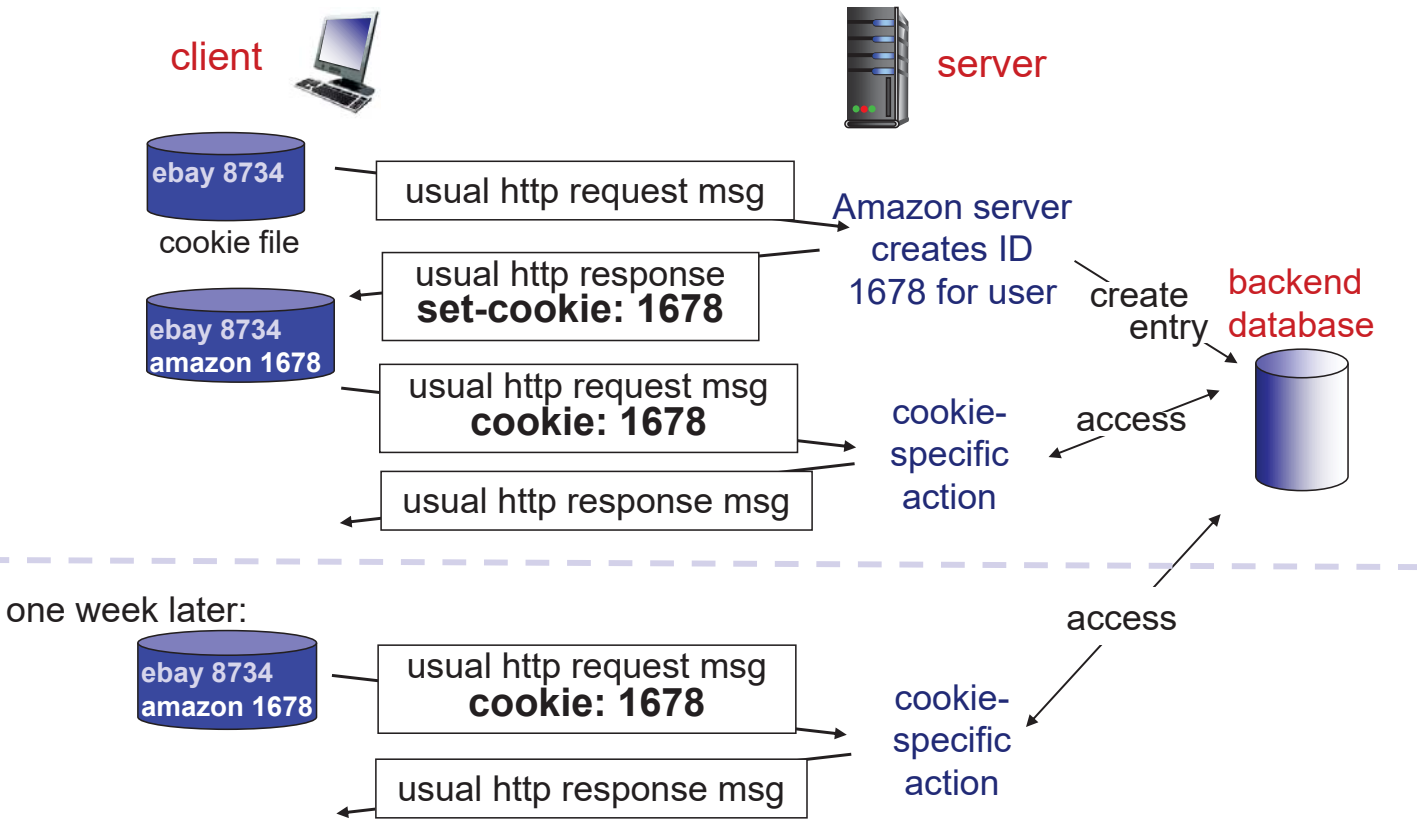
- cookie file kept on user's host, managed by user's browser
- back-end database at Web site



example:

- Susan always access Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping "state" (cont.)



延伸議題：同源政策(Same-origin policy)

Cookies (continued)

what cookies can be used for:

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

aside

cookies and privacy:

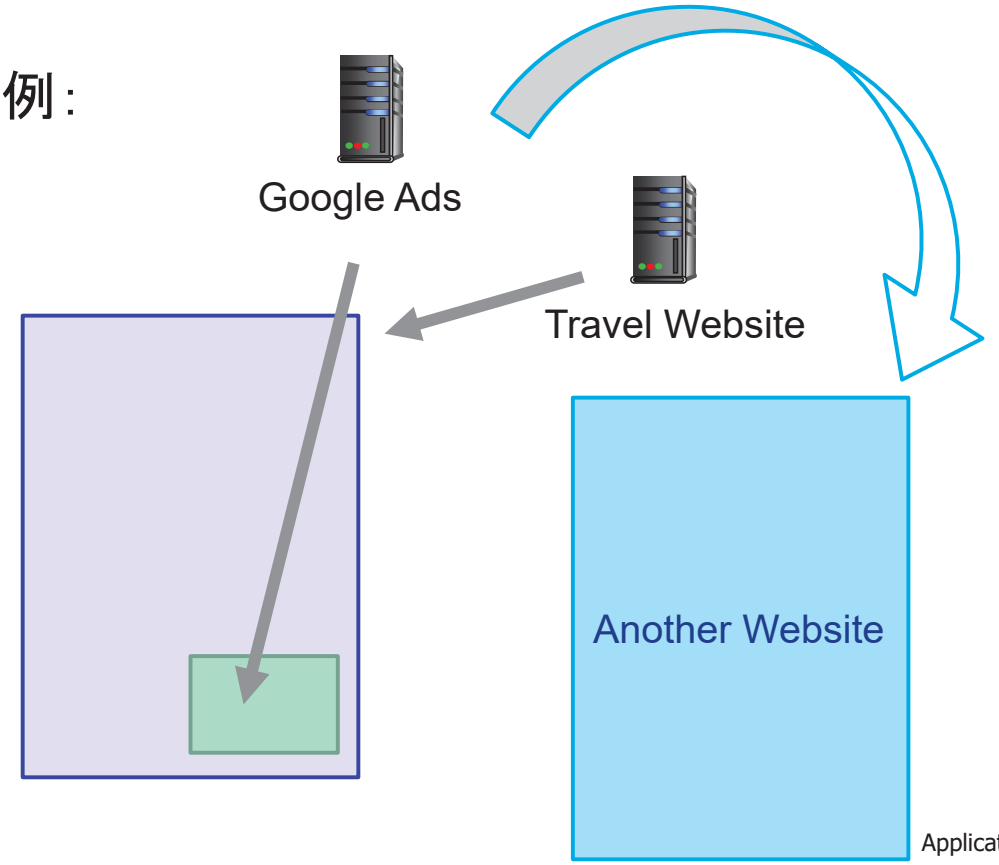
- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

how to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

第三方Cookie

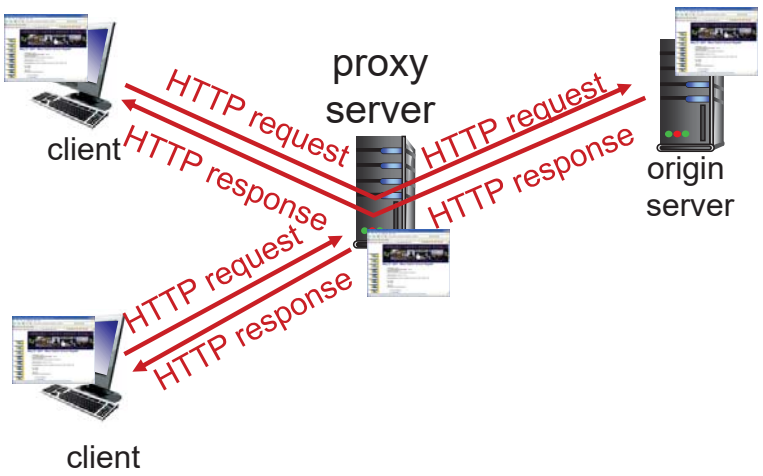
- 舉例：



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



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 (so too does P2P file sharing)
- Break through access restrictions

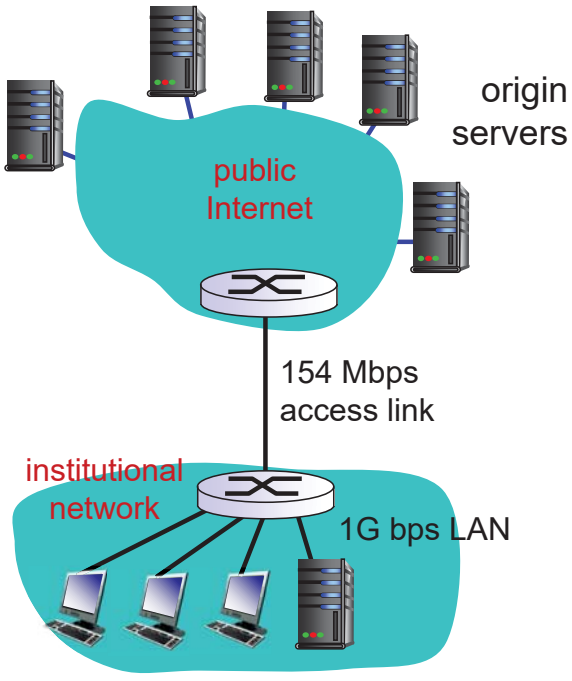
Caching example:

assumptions:

- avg object size: 10M bits
- avg request rate from browsers to origin servers: 15 request / sec
- avg data rate to browsers: 150 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 154 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = 99% *problem!*
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs



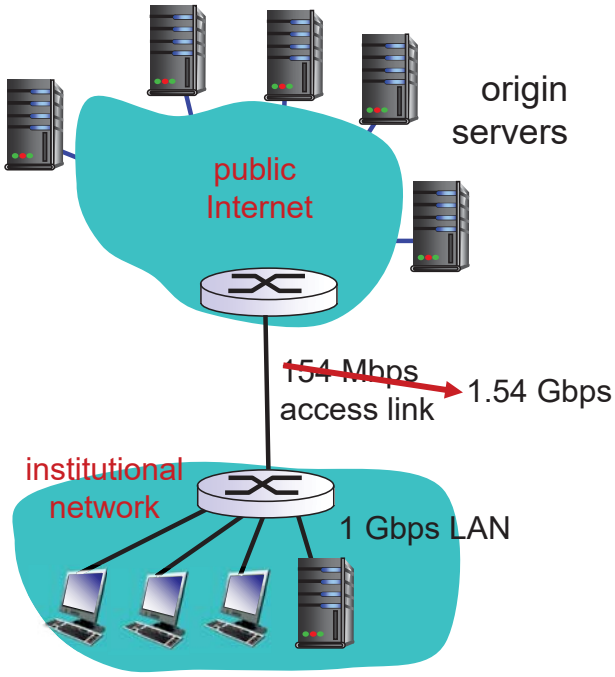
Caching example: fatter access link

assumptions:

- avg object size: 10 M bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 150 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: ~~154 Mbps~~ 1.54 Gbps

consequences:

- LAN utilization: 15%
- access link utilization = ~~99%~~ 9.9%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + ~~minutes~~ + usecs
  msecs



Cost: increased access link speed (not cheap!)

Caching example: install local cache

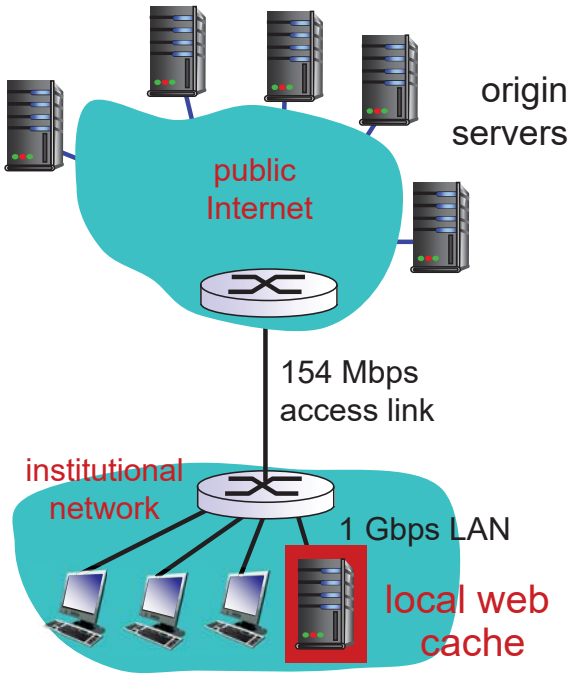
assumptions:

- avg object size: 10M bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 150 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 154 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ?
- total delay = ?

How to compute link utilization, delay?

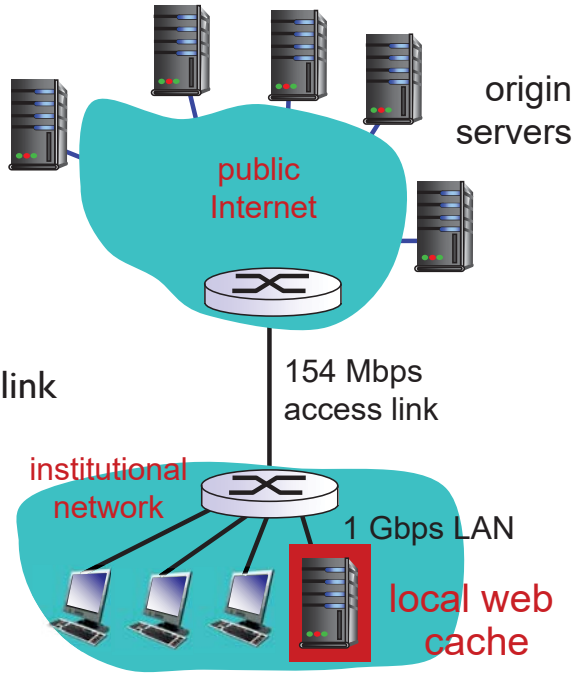


Cost: web cache (cheap!)

Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin
- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link
  = 0.6*150 Mbps = 90 Mbps
  - utilization = 90/154 = 58%
- total delay
  - = 0.6 * (delay from origin servers) + 0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs
  - less than with 1.54 Gbps link (and cheaper too!)



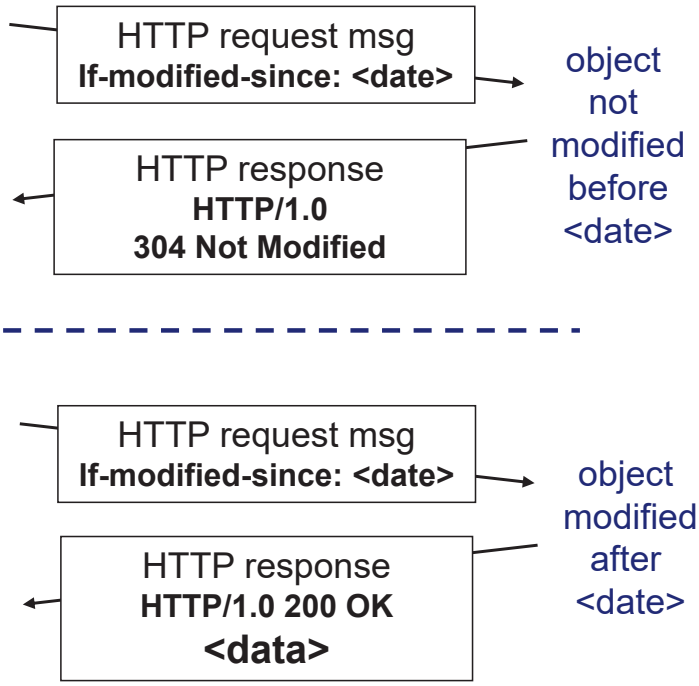
Conditional GET

- *Goal:* don't send object if cache has up-to-date cached version
  - no object transmission delay
  - lower link utilization
- *cache:* specify date of cached copy in HTTP request

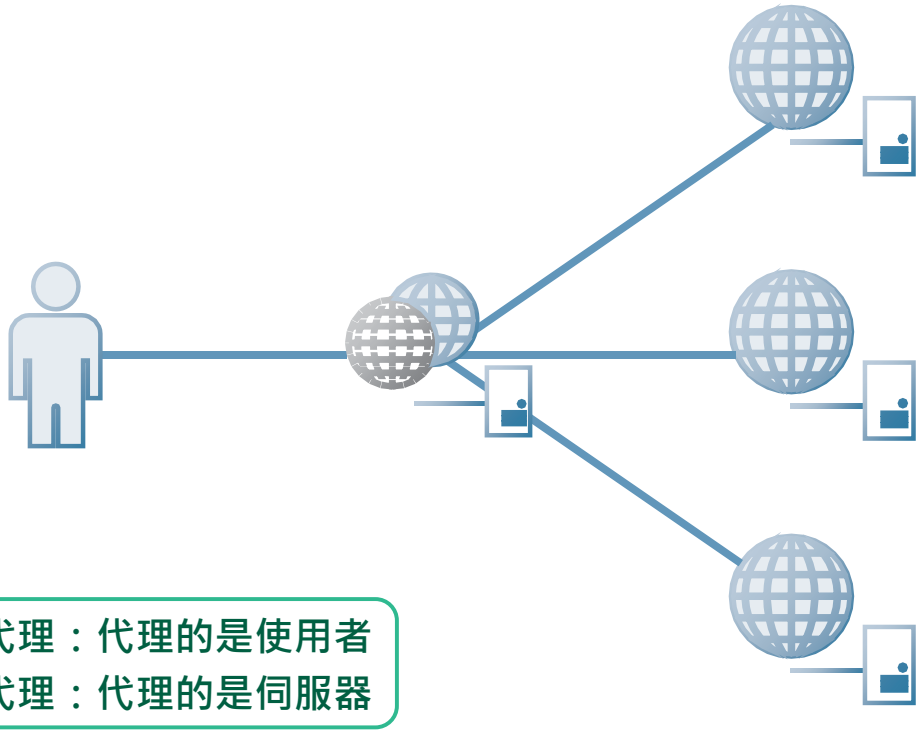
```
If-modified-since: <date>
```

- *server:* response contains no object if cached copy is up-to-date:

```
HTTP/1.0 304 Not Modified
```



反向代理 (Reverse proxy)



正向代理：代理的是使用者
反向代理：代理的是伺服器

相關議題

接著將針對 HTTP 的一些 Header 做討論

Referrer Header Leakage

```
GET /index.html HTTP/1.1
Host: abc.com
User-Agent: Chrome/65.0.3325.181
Referer: http://ex.tw?usr=kaihao&pwd=abcd1234
```

【 防 範 】

使用GET方法時，只單純做資料讀取和查詢，避免利用網址傳遞敏感資訊。

將所有Response header的Referrer-Policy選項設為no-referrer。

連線到外部網站時，一律先重新導向到某一頁面，再轉往目的頁面。

Content Security Policy

- 限制載入到此網站上的內容來源，
- 語法： Content-Security-Policy: <規範> <來源>[<來源>]*;

< 規範 >

名稱	說明
img-src	設定允許的圖片來源
style-src	設定允許的CSS來源
script-src	設定允許的JavaScript來源
frame-ancestors	設定該頁面允許被哪些網址嵌入
frame-src	設定該頁面允許嵌入哪些網址的內容

< 來源 >

- 直接指定網址（可包含萬用字元、通訊協定、埠號）。
- 特定關鍵字。（如 'none' 表示禁止任何來源，
  而 'self' 表示只允許同源的來源（scheme、domain、port 相同））

Clickjacking Attack

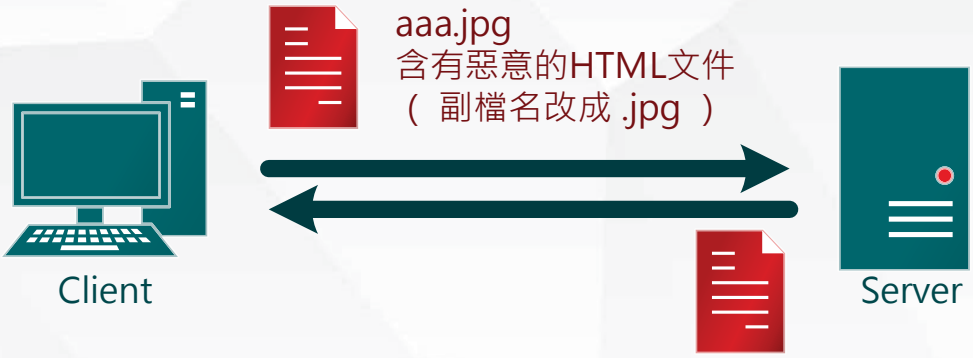
【 防 範 】

根本的解決方法是，讓自己的網站不能被隨意嵌入到其他網站裡頭。

```
X-Frame-Options: DENY | SAMEORIGIN | ALLOW-FROM <URL>
```

```
Content-Security-Policy: frame-ancestors 'none'
```

MIME Sniffing Attack



【 防 範 】

不應只以副檔名作為檔案內容的檢查依據。

```
X-Content-Type-Options: nosniff
```