Applications

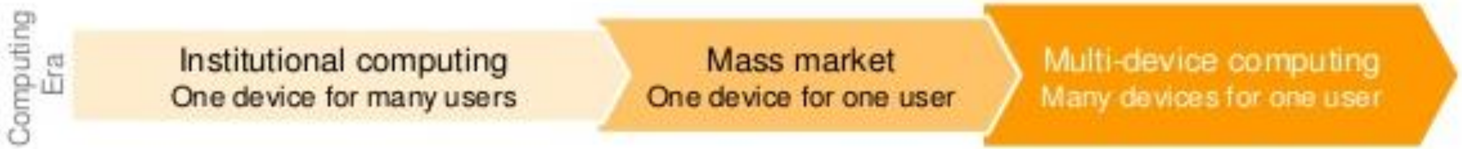
Our Goals:

- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
  - content distribution networks

- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- creating network applications
  - socket API

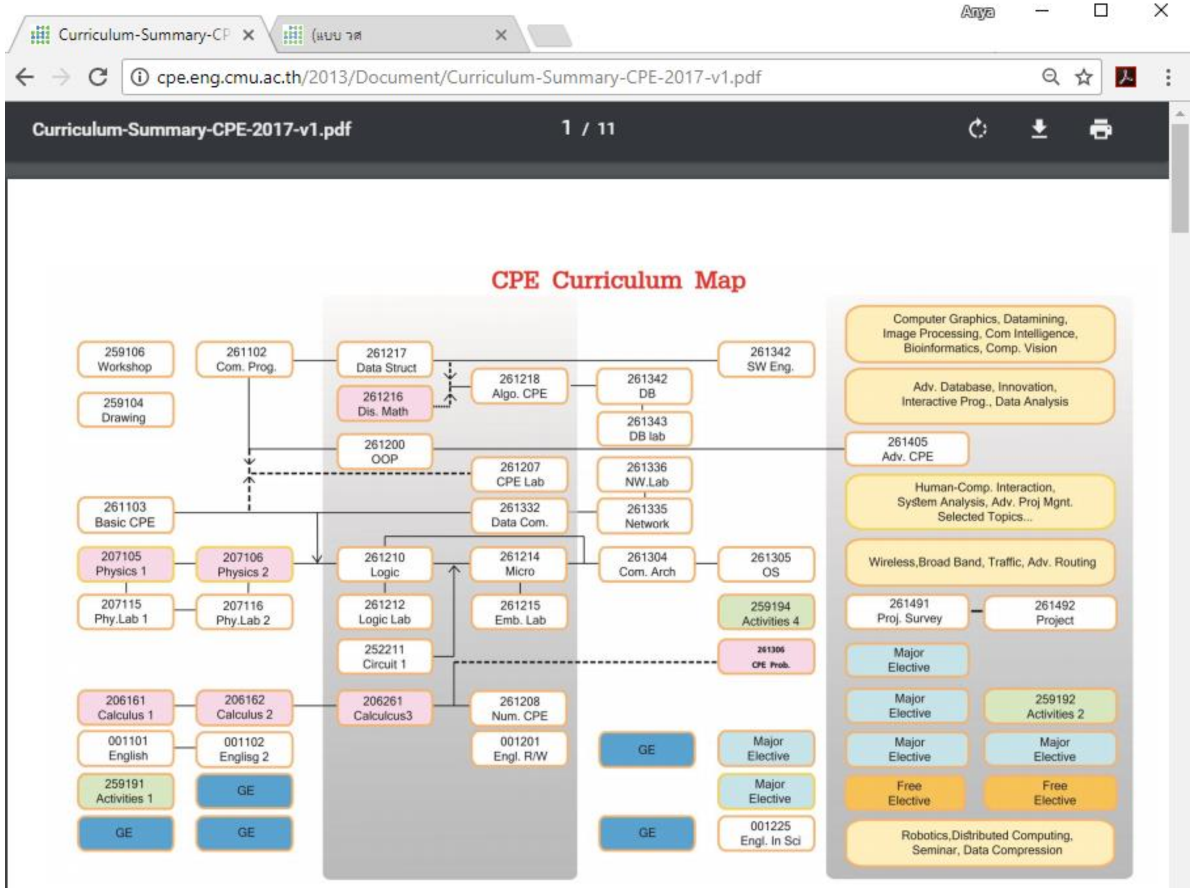
https://www.youtube.com/watch?v=uGI00HV7Cfw

The evolution of a computing technology



HTTP

Hypertext Transfer Protocol



HOW TO FETCH A WEBPAGE?

- When user type a URL…
- URL indicate hostname and path to "objects"
- Interaction between Server/Client

HTTP overview

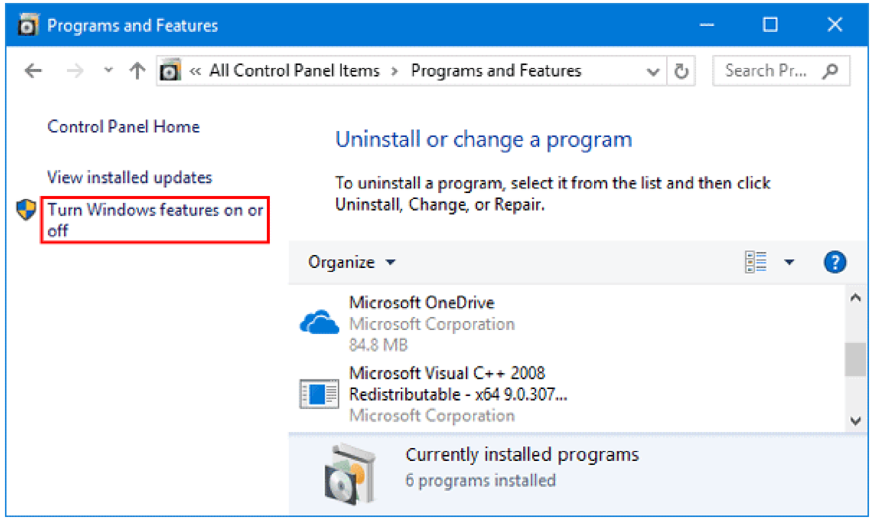
HTTP: hypertext transfer protocol

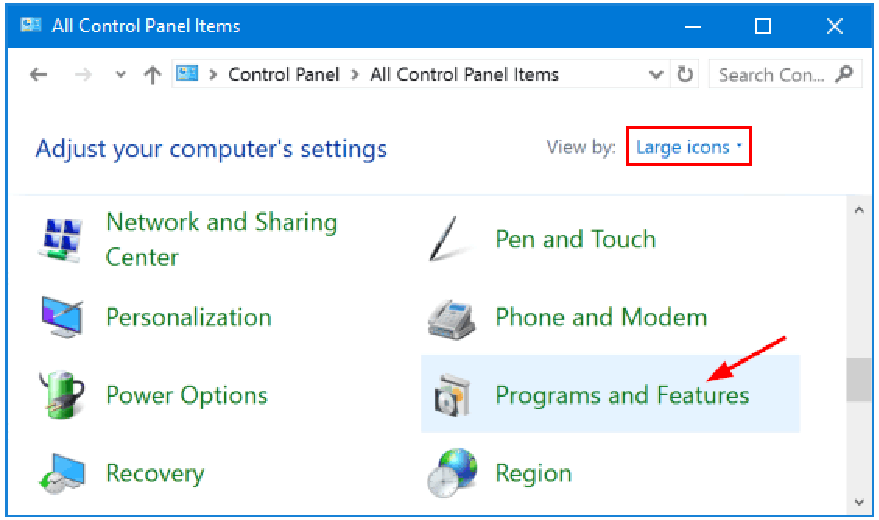
- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests

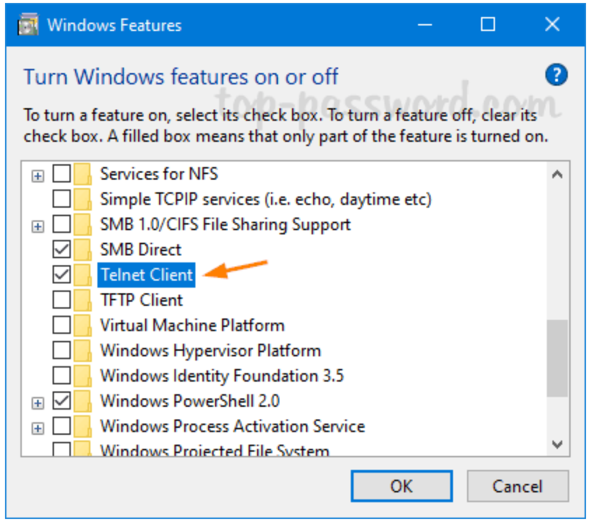
Web and HTTP

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.,



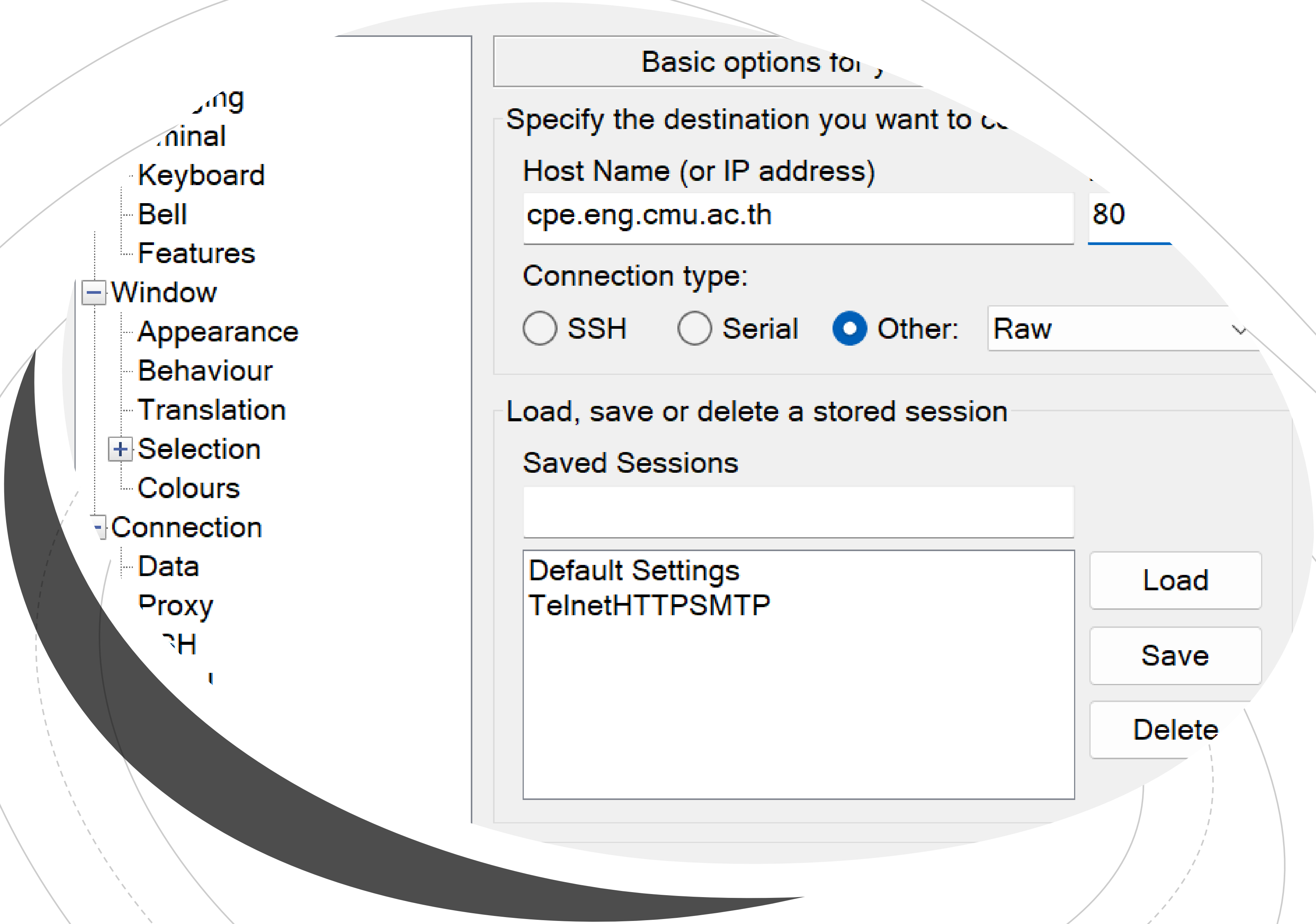




Telnet on Windows : Enable Telnet

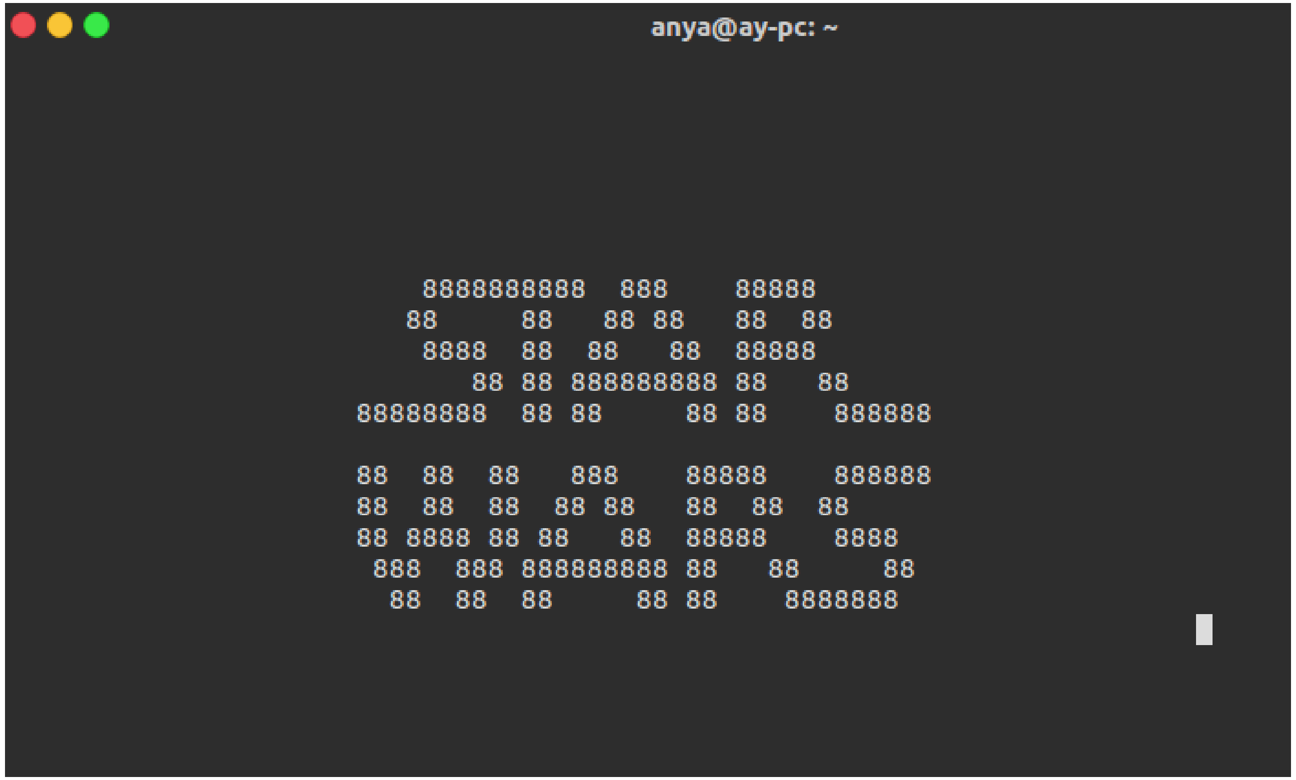
- Open Terminal with cmd command
- Enable Telnet on Windows
  - Open "Control Panel > Programs" and select "Turn Windows features on or off" under the category "Programs and Features".
  - In the Windows Features list, select "Telnet Client" to activate it, then apply the change.

Telnet with Putty



Telnet

telnet towel.blinkenlights.nl 23 ?



Telnet provides a bidirectional interactive text-oriented over the Transmission Control Protocol (TCP).

Results from TELNET

```
anya@ay-pc:~$ telnet www.cpe.eng.cmu.ac.th 80
Trying 202.28.24.139...
Connected to cpe.eng.cmu.ac.th.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.cpe.eng.cmu.ac.th

HTTP/1.1 200 OK
Date: Tue, 10 Oct 2017 07:00:26 GMT
Server: Apache
Last-Modified: Tue, 22 Oct 2013 06:07:54 GMT
ETag: "5ea04a-187-4e94e38a68280"
Accept-Ranges: bytes
Content-Length: 391
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<META HTTP-EQUIV="Refresh" CONTENT="0;URL=http://cpe.eng.cmu.ac.th/2013">
<title>Computer Engineering CMU</title>
</head>
<body>
</body>
</html>
Connection closed by foreign host.
anya@ay-pc:~$
```

```
GET / HTTP/1.1
Host: www.cpe.eng.cmu.ac.th
```

*enp3s0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http

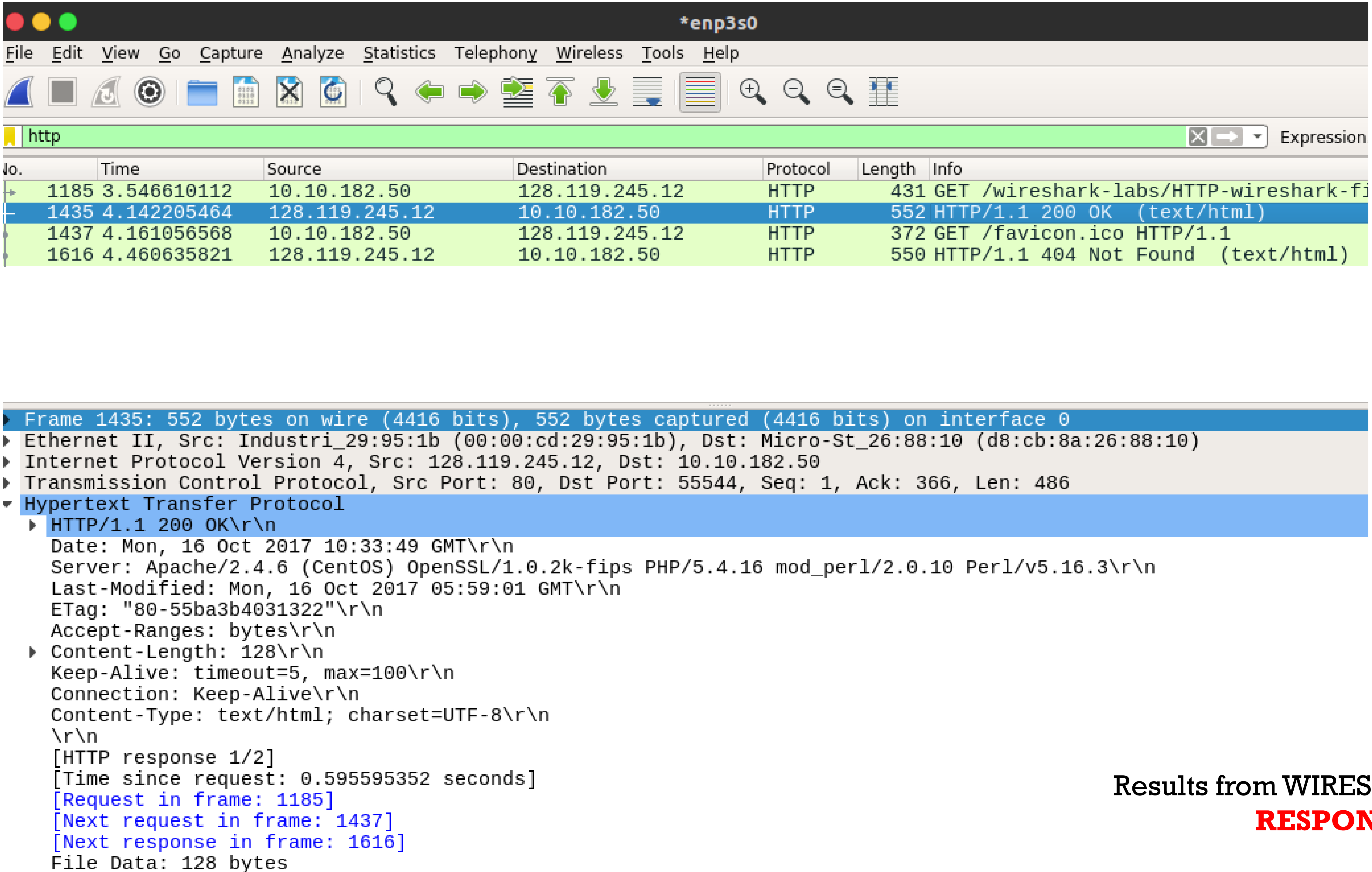
Expression...

No.	Time	Source	Destination	Protocol	Length	Info
1185	3.546610112	10.10.182.50	128.119.245.12	HTTP	431	GET /wireshark-labs/HTTP-wireshark-file1.
1435	4.142205464	128.119.245.12	10.10.182.50	HTTP	552	HTTP/1.1 200 OK (text/html)
1437	4.161056568	10.10.182.50	128.119.245.12	HTTP	372	GET /favicon.ico HTTP/1.1
1616	4.460635821	128.119.245.12	10.10.182.50	HTTP	550	HTTP/1.1 404 Not Found (text/html)

```
Frame 1185: 431 bytes on wire (3448 bits), 431 bytes captured (3448 bits) on interface 0
Ethernet II, Src: Micro-St_26:88:10 (d8:cb:8a:26:88:10), Dst: Industri_29:95:1b (00:00:cd:29:95:1b)
Internet Protocol Version 4, Src: 10.10.182.50, Dst: 128.119.245.12
Transmission Control Protocol, Src Port: 55544, Dst Port: 80, Seq: 1, Ack: 1, Len: 365
Hypertext Transfer Protocol
    GET /wireshark-labs/HTTP-wireshark-file1.html HTTP/1.1\r\n
    Host: gaia.cs.umass.edu\r\n
    User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:55.0) Gecko/20100101 Firefox/55.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    \r\n
    [Full request URI: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html]
    [HTTP request 1/2]
    [Response in frame: 1435]
    [Next request in frame: 1437]
```

Results from WIRESHARK

REQUEST



Results from WIRESHARK

RESPONSE

HTTP request message

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

request line (GET, POST, HEAD commands)

carriage return character

line-feed character

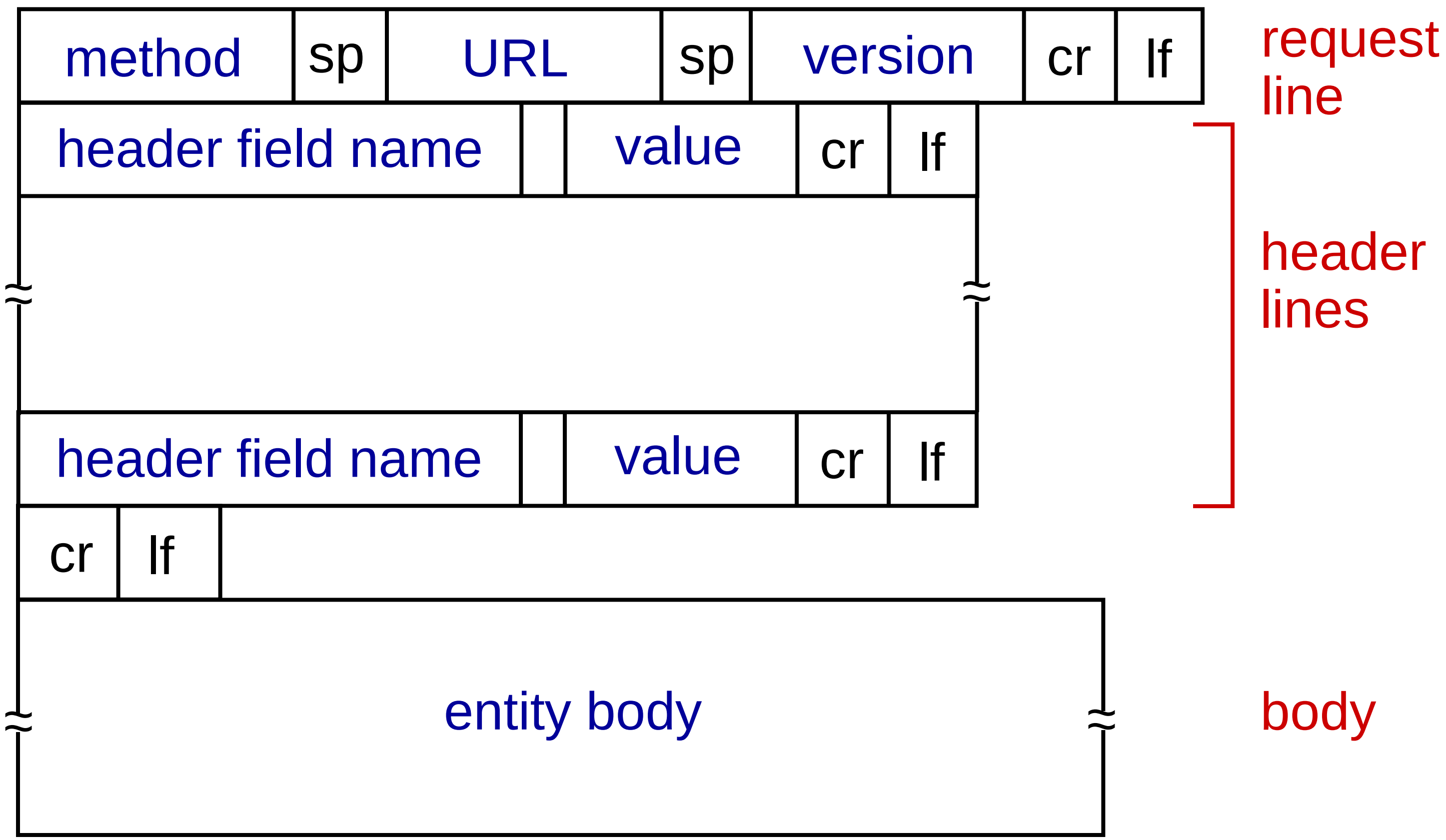
header lines

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

```
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
```

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

HTTP request message: general format



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 ...
```

* 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

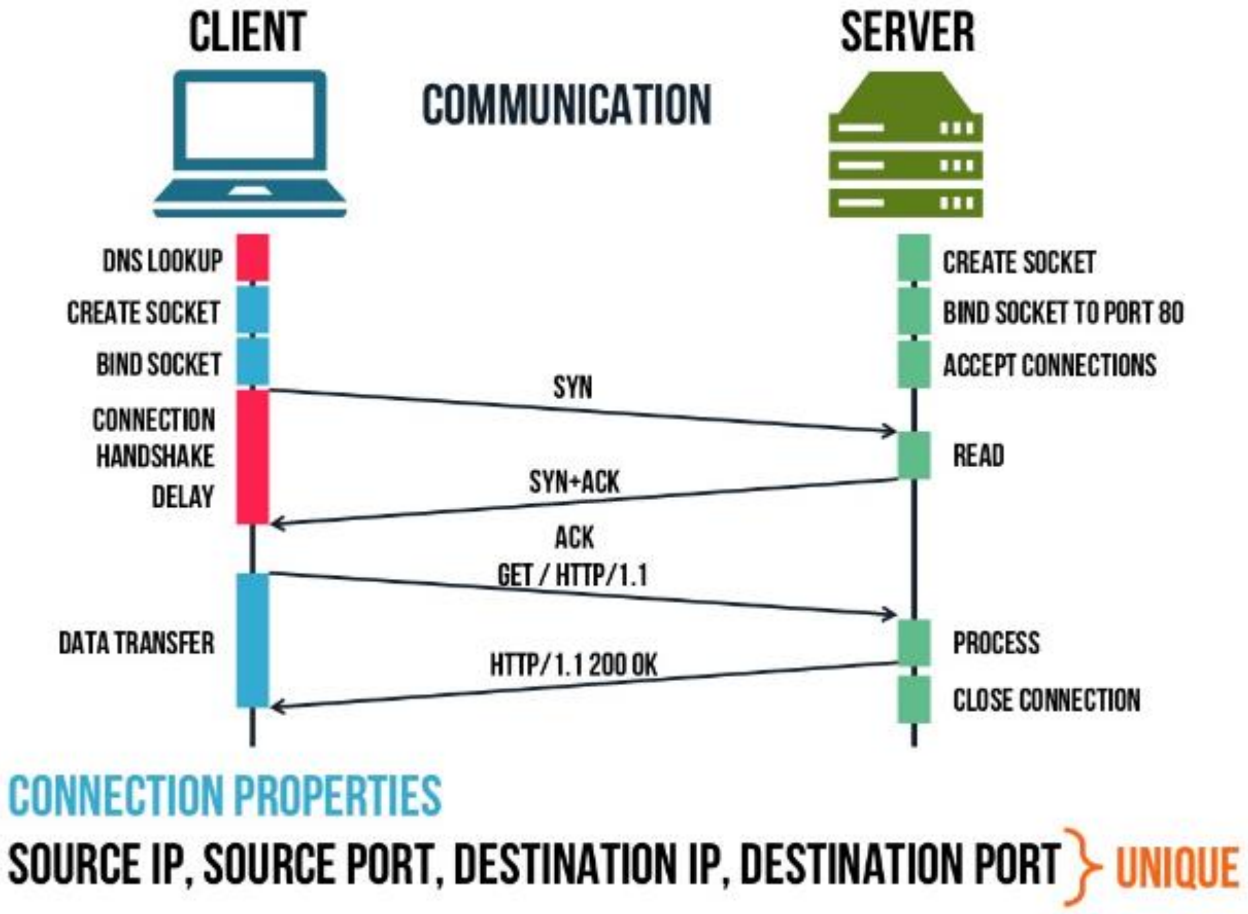
    404 Not Found

    - requested document not found on this server

    505 HTTP Version Not Supported

Can you write an HTTP protocol sequence diagram?

Can you write an HTTP protocol sequence diagram?



HTTP overview

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

Non-Persistent HTTP vs. Persistent HTTP

What if there are more than one objects?

per·sist·ent

/pərˈsist(ə)nt/

adjective

1. continuing firmly or obstinately in a course of action in spite of difficulty or opposition.
   "one of the government's most persistent critics"
   synonyms: tenacious, persevering, determined, resolute, purposeful, dogged, single-minded, tireless, indefatigable, patient, unflagging, untiring, insistent, importunate, relentless, unrelenting; More
2. continuing to exist or endure over a prolonged period.
   "persistent rain will affect many areas"
   synonyms: constant, continuous, continuing, continual, nonstop, never-ending, steady, uninterrupted, unbroken, interminable, incessant, unceasing, endless, unending, perpetual, unremitting, unrelenting, relentless, unrelieved, sustained More

Non-Persistent HTTP vs. Persistent HTTP

Example: an HTML file with 2 pictures in it, how many objects?

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

Non-Persistent HTTP

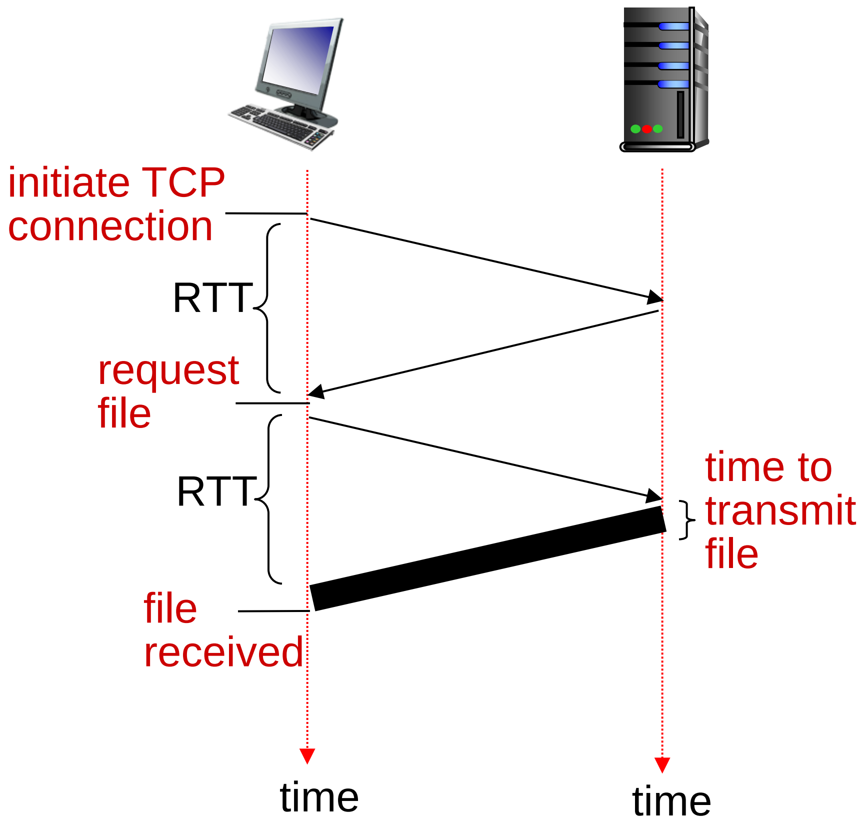
Non-persistent HTTP: response time

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: response time

non-persistent HTTP issues:

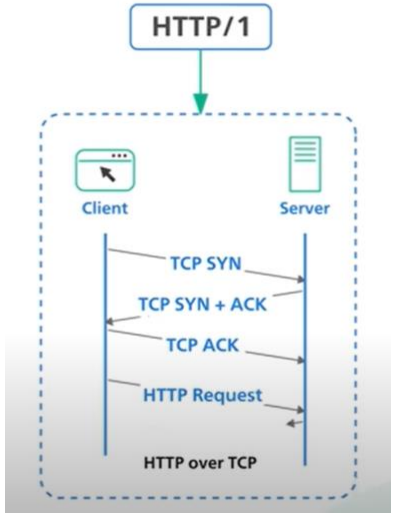
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

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

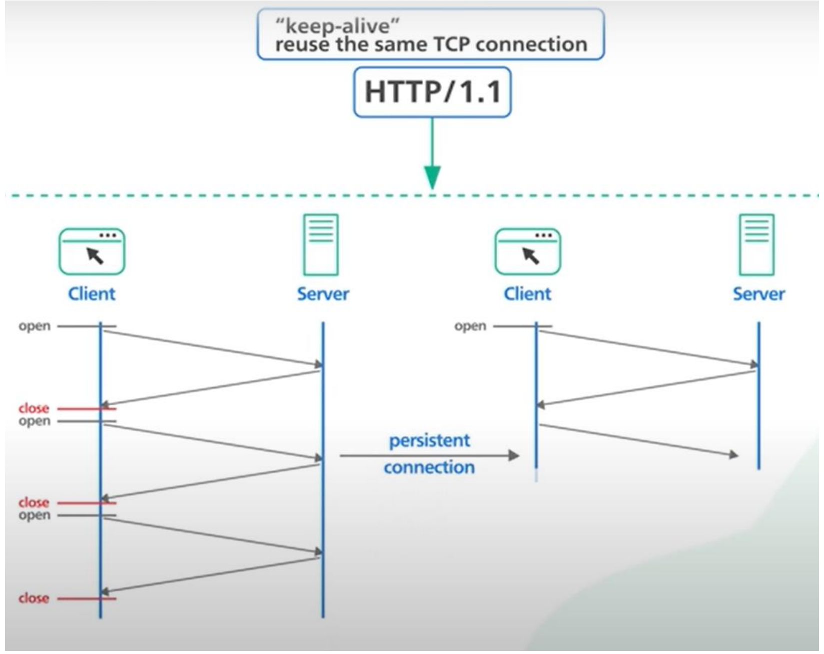
HTTP Evolution

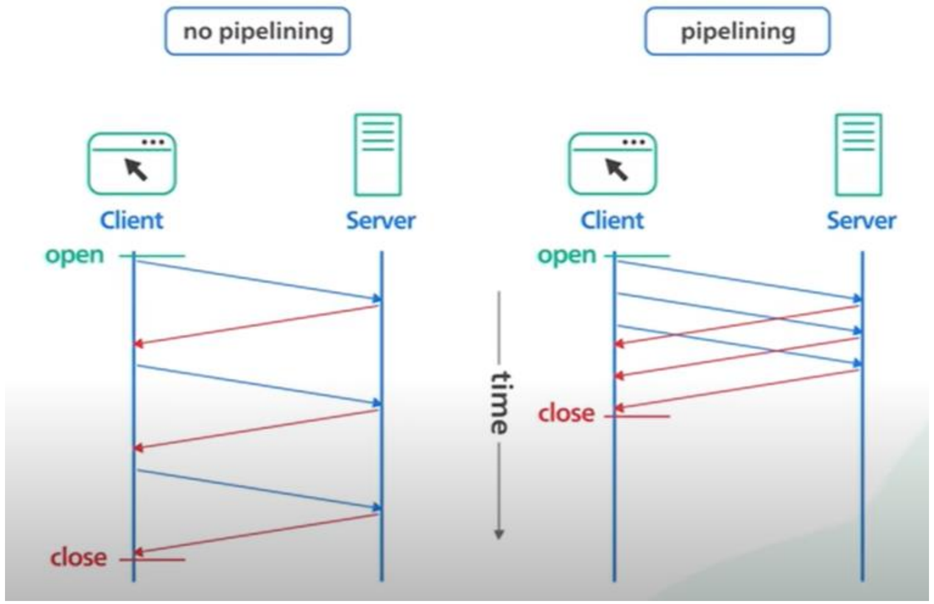
- **HTTP v.1**
  - HTTP v.1.0



1996

- HTTP v.1.1



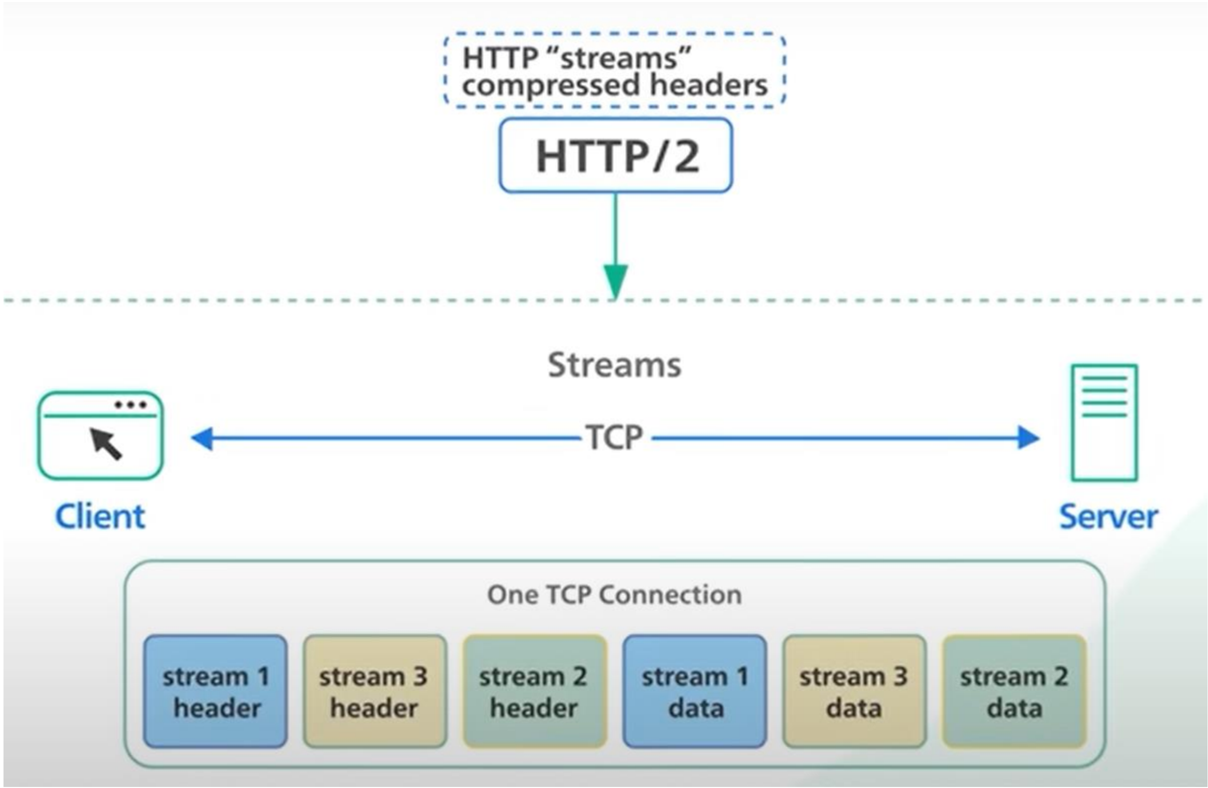


HTTP 1.1 in 1997
- Persistent connection
- Pipelining

https://www.youtube.com/watch?v=a-sBfyiXysI

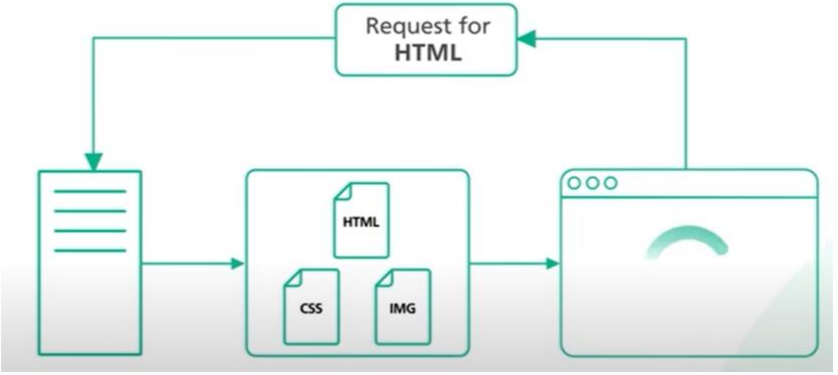
HTTP Evolution

- **HTTP v.2**



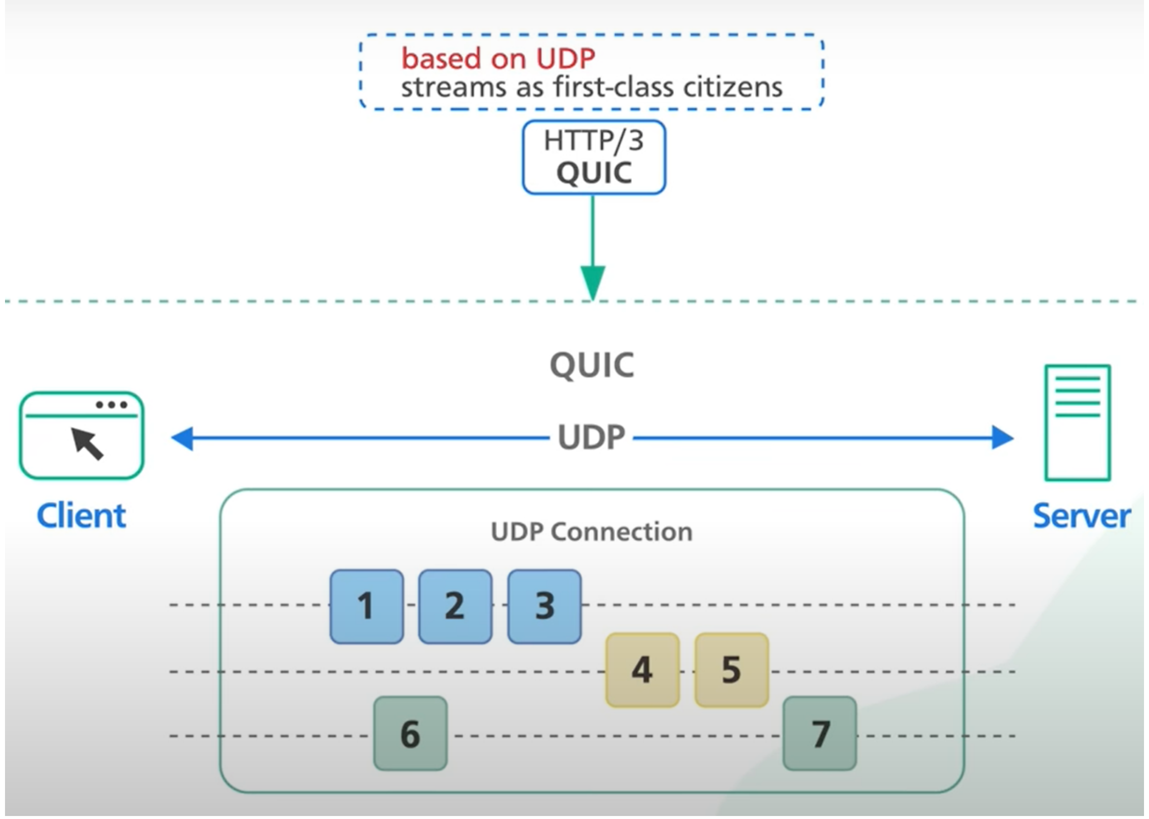
HTTP 2 in 2015

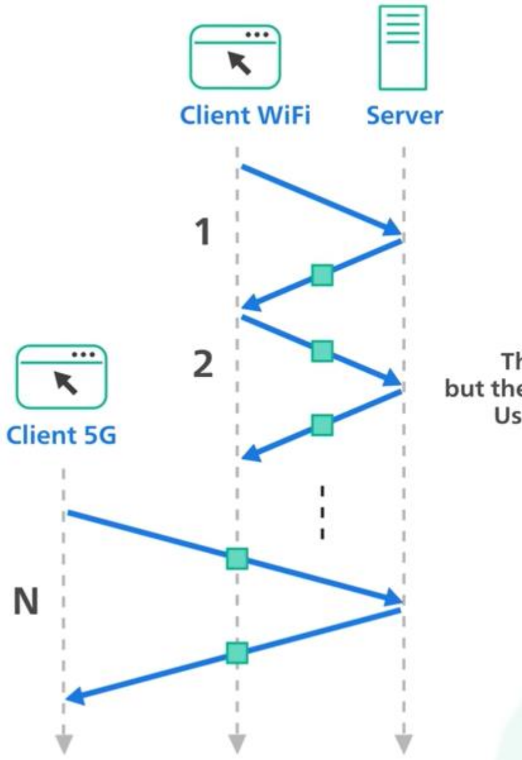
- HTTP stream
- Compressed Header
- Push method introduced : server sends update to client

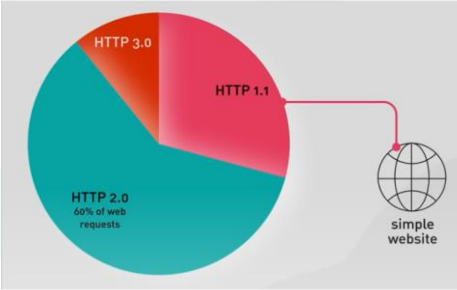


HTTP Evolution

- **HTTP v.3** HTTP 3 in 2022
  - QUIC protocol







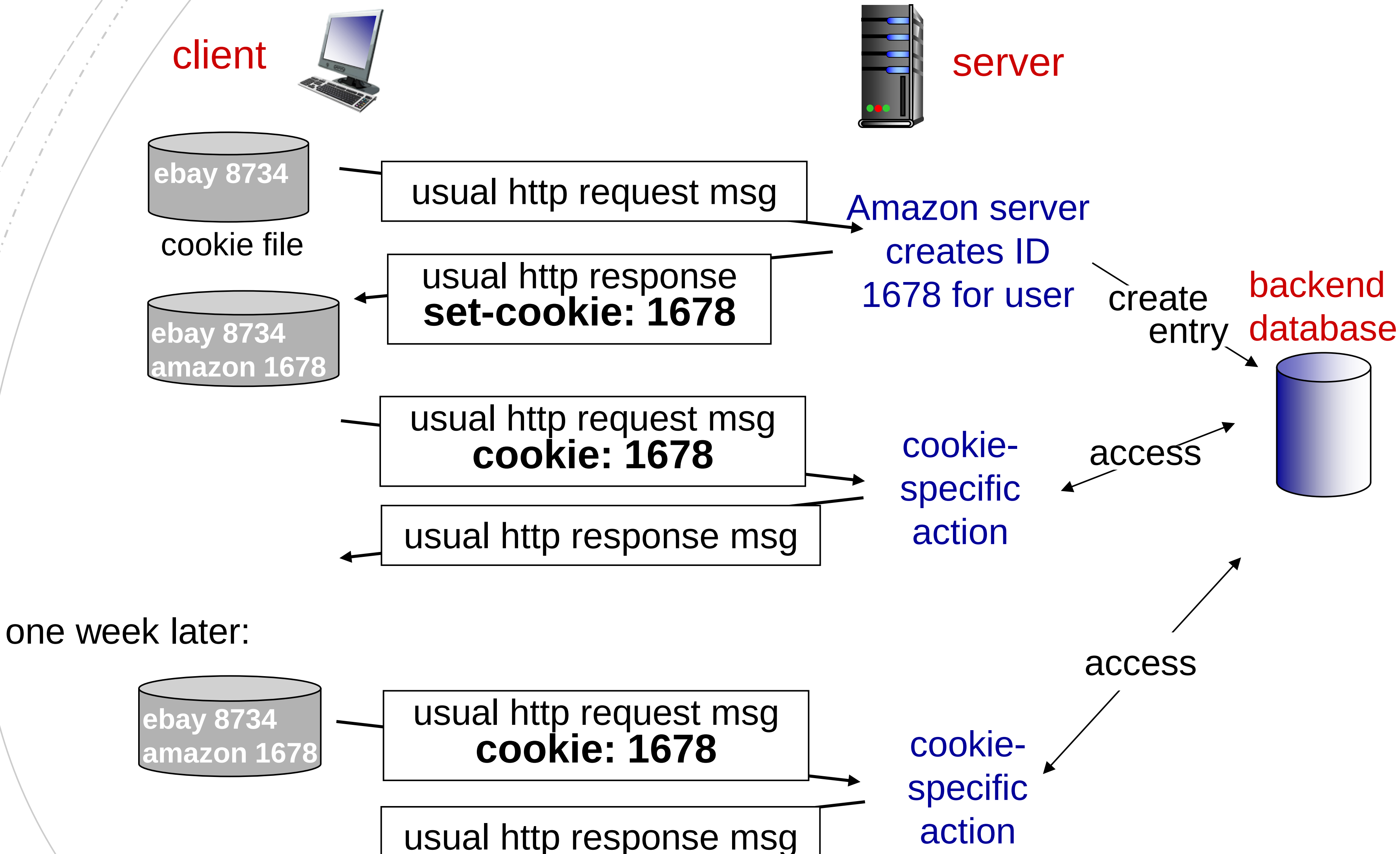
STATE
vs.
STATELESS

HTTP is stateless,
How to keep track of user information?

HTTP IS STATELESS!!!

We use cookies to keep state of information

Cookies: keeping "state"



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
3) cookie file kept on user's host, managed by user's browser
4) 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

User-server state: cookies

what cookies can be used for:

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

aside

cookies and privacy:

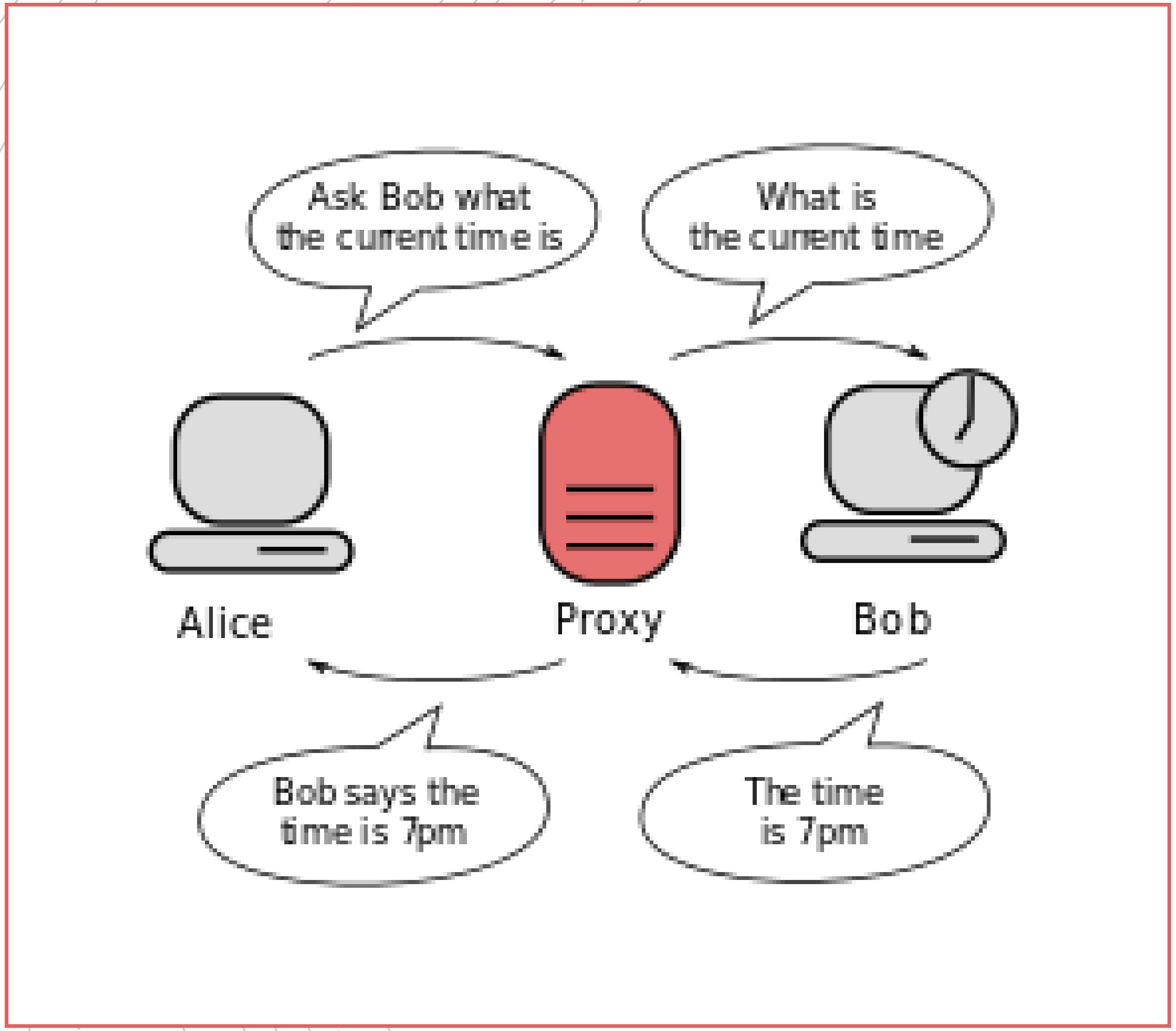
- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

how to keep "state":

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

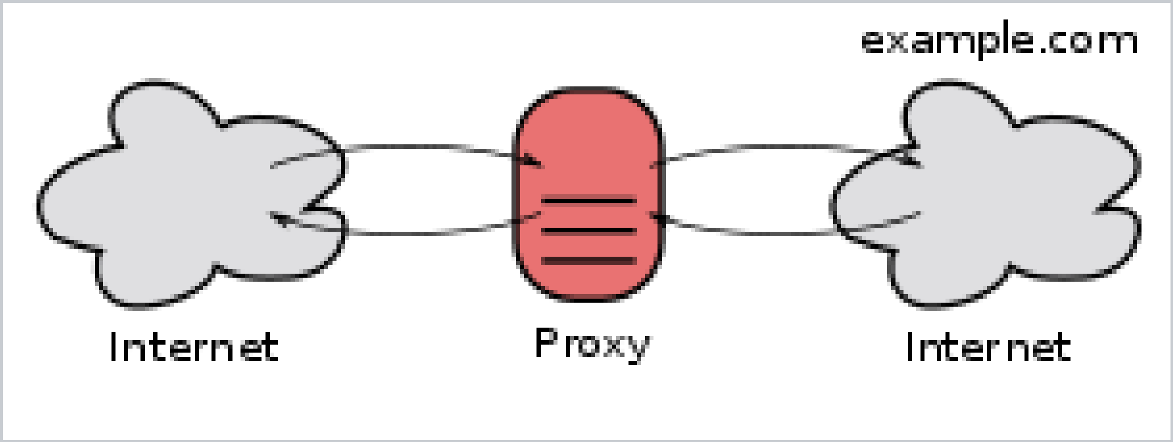
Web caches
or
proxy server

Can we FETCH a web page more efficiently?

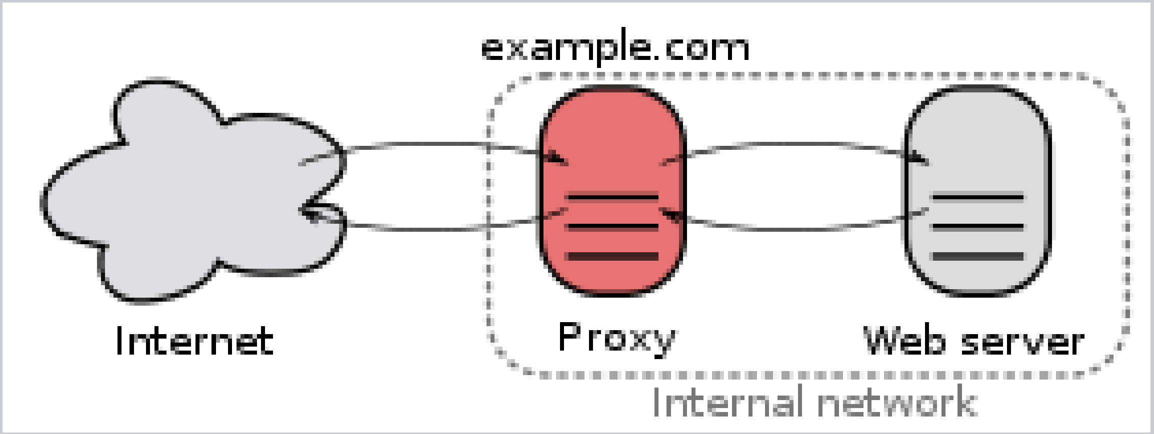


Can we FETCH a web page more efficiently?

- **A proxy server may reside on the user's local computer, or at various points between the user's computer and destination servers on the Internet.**
  - **A proxy server that passes requests and responses unmodified is usually called a gateway or sometimes a tunneling proxy.**
- - **A forward proxy** is an Internet-facing proxy used to retrieve from a wide range of sources (in most cases anywhere on the Internet).
- - **A reverse proxy** is usually an internal-facing proxy used as a front-end to control and protect access to a server on a private network. A reverse proxy commonly also performs tasks such as load-balancing, authentication, decryption or caching.



An open proxy forwarding requests from and to anywhere on the Internet.



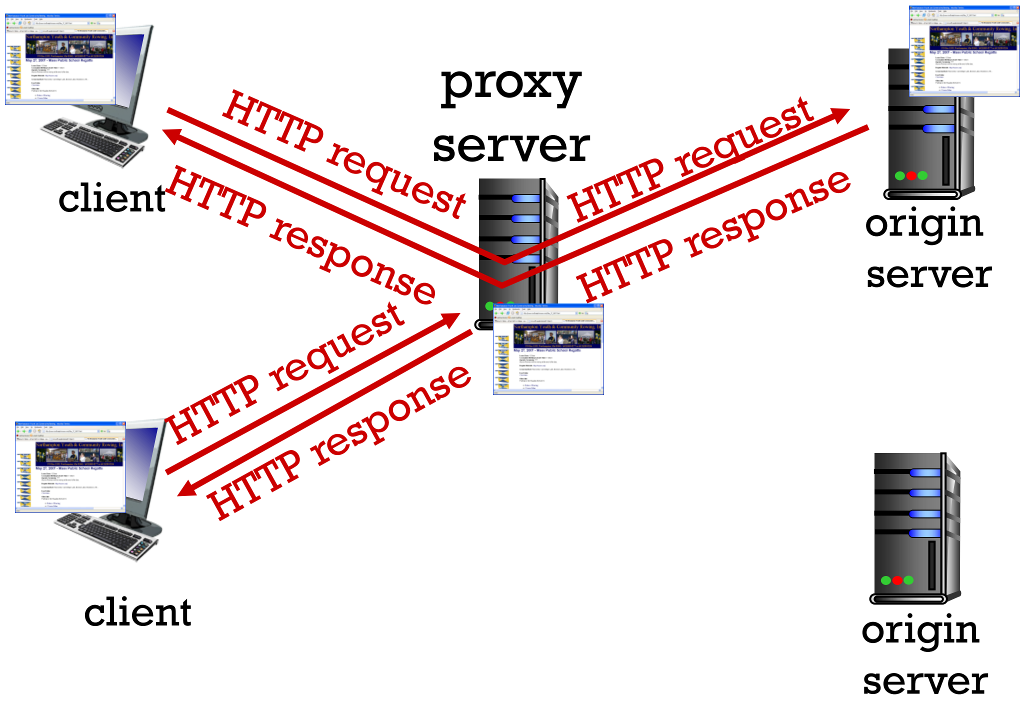
A reverse proxy taking requests from the Internet and forwarding them to servers in an internal network. Those making requests connect to the proxy and may not be aware of the internal network.

Proxy

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



Web caches (proxy server)

- 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)

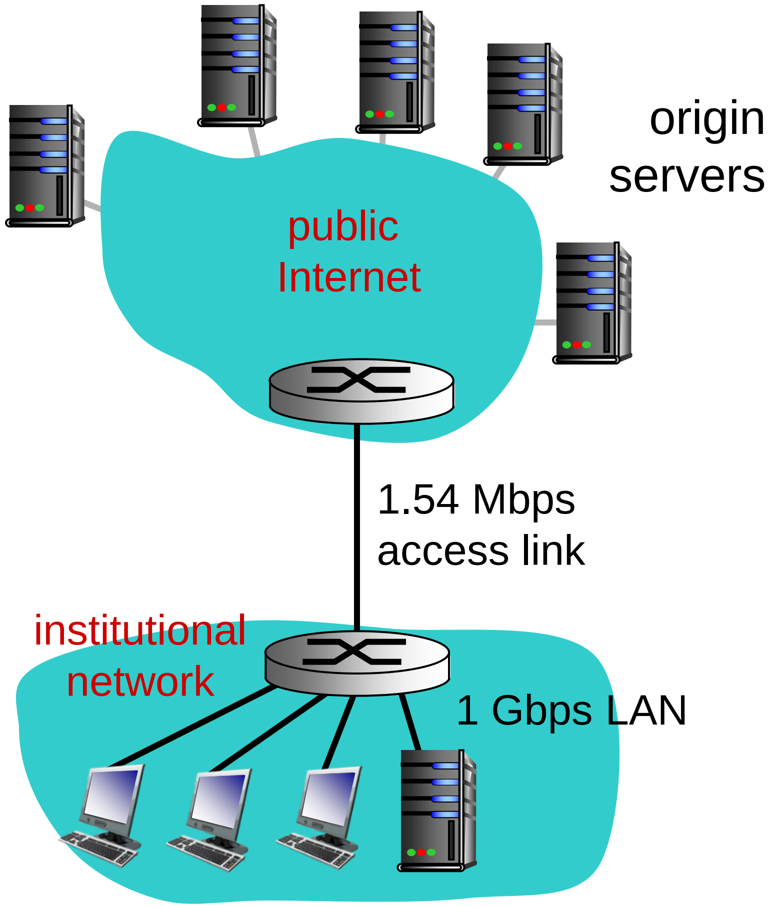
why Web caching?

- 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)

Caching example:

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

Can you compute LAN <u>utilization</u>, access link utilization, total delay?



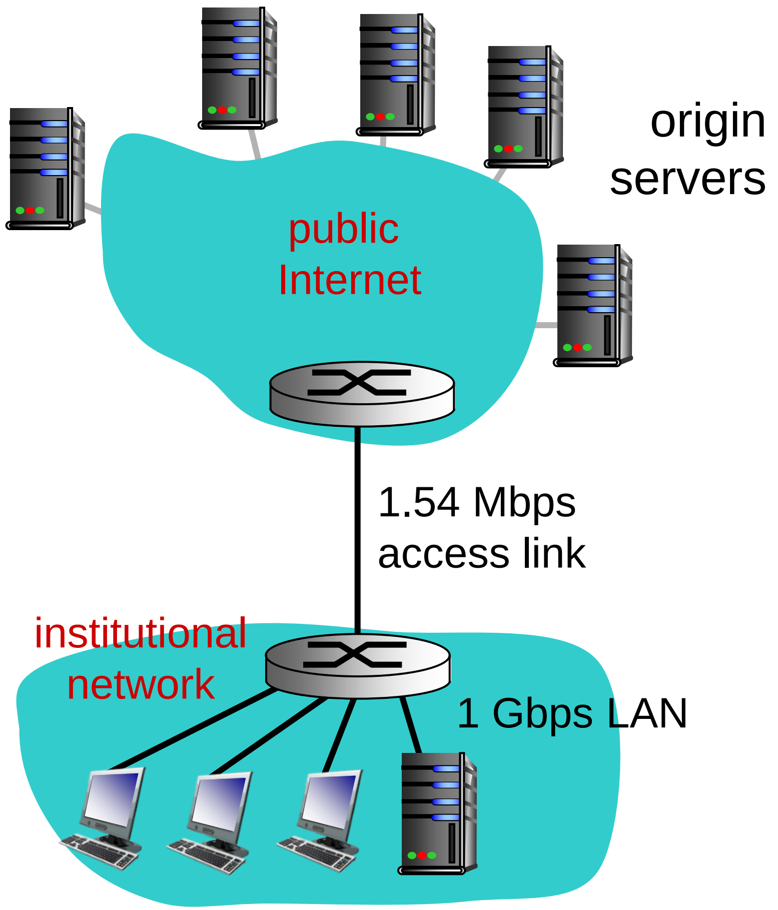
Caching example:

assumptions:

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

consequences:

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

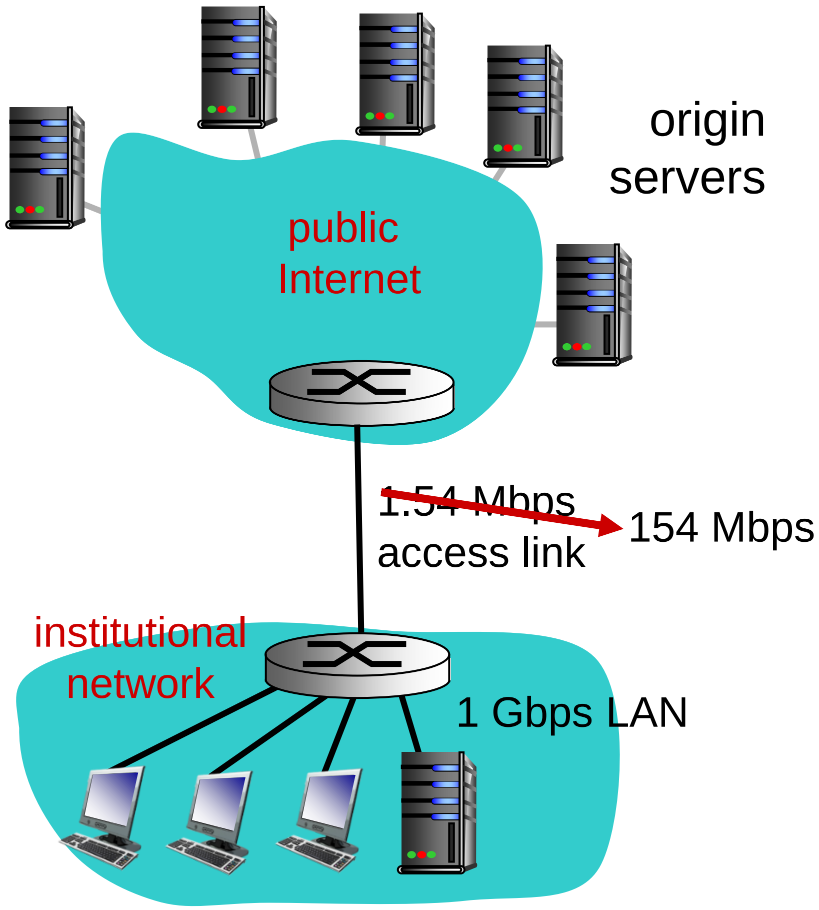


Caching example: fatter access link

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 154 Mbps

  If we change the access link to 154 Mbps, what will happen?

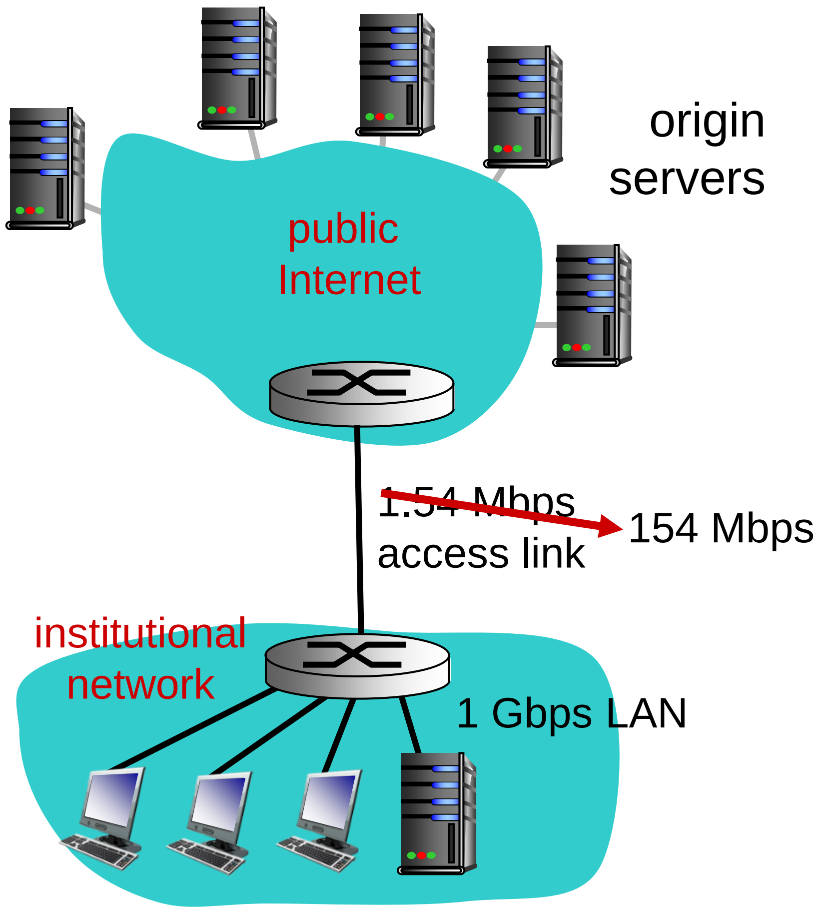


Caching example: fatter access link

assumptions:
- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 154 Mbps

consequences:
- LAN utilization: 15%
- access link utilization = 0.99%
- total delay = Internet delay + access delay + LAN delay
  = msecs

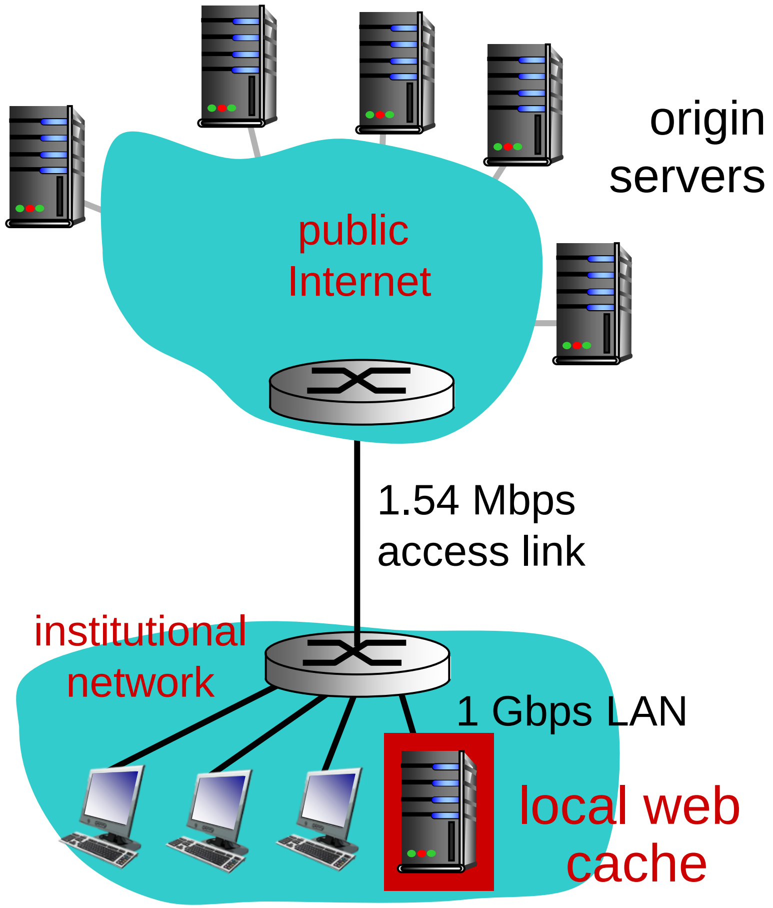
HOWEVER INCREASE ACCESS LINK IS EXPENSIVE !!!



Caching example: install local cache

assumptions:

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



Adding a catch with a hit-rate of 0.4!

How to compute link utilization, delay?

Cost: web cache (cheap!)

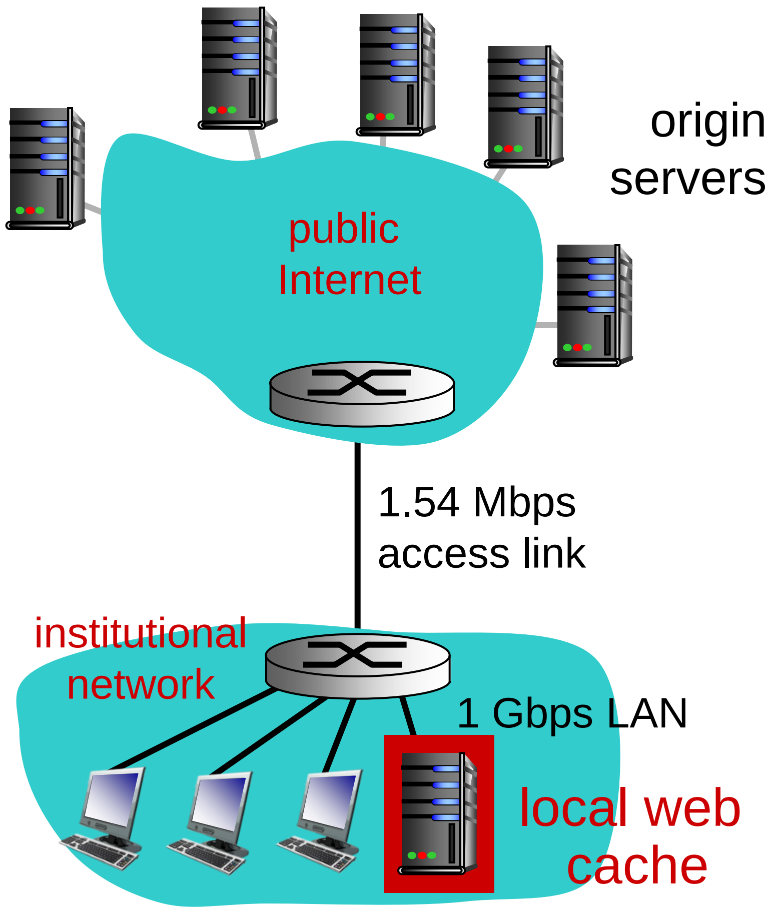
Caching example: install local cache

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*1.50 Mbps = .9 Mbps
  - utilization = 0.9/1.54 = .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 154 Mbps link (and cheaper too!)



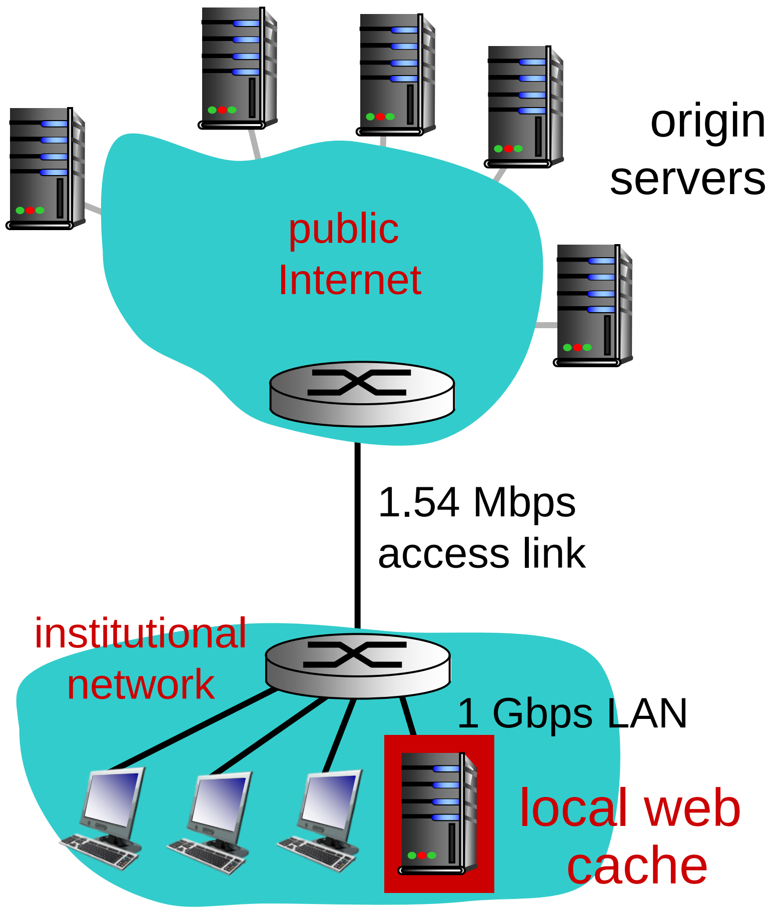
Caching example: install local cache

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*1.50 Mbps = .9 Mbps
  - utilization = 0.9/1.54 = .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 154 Mbps link (and cheaper too!)

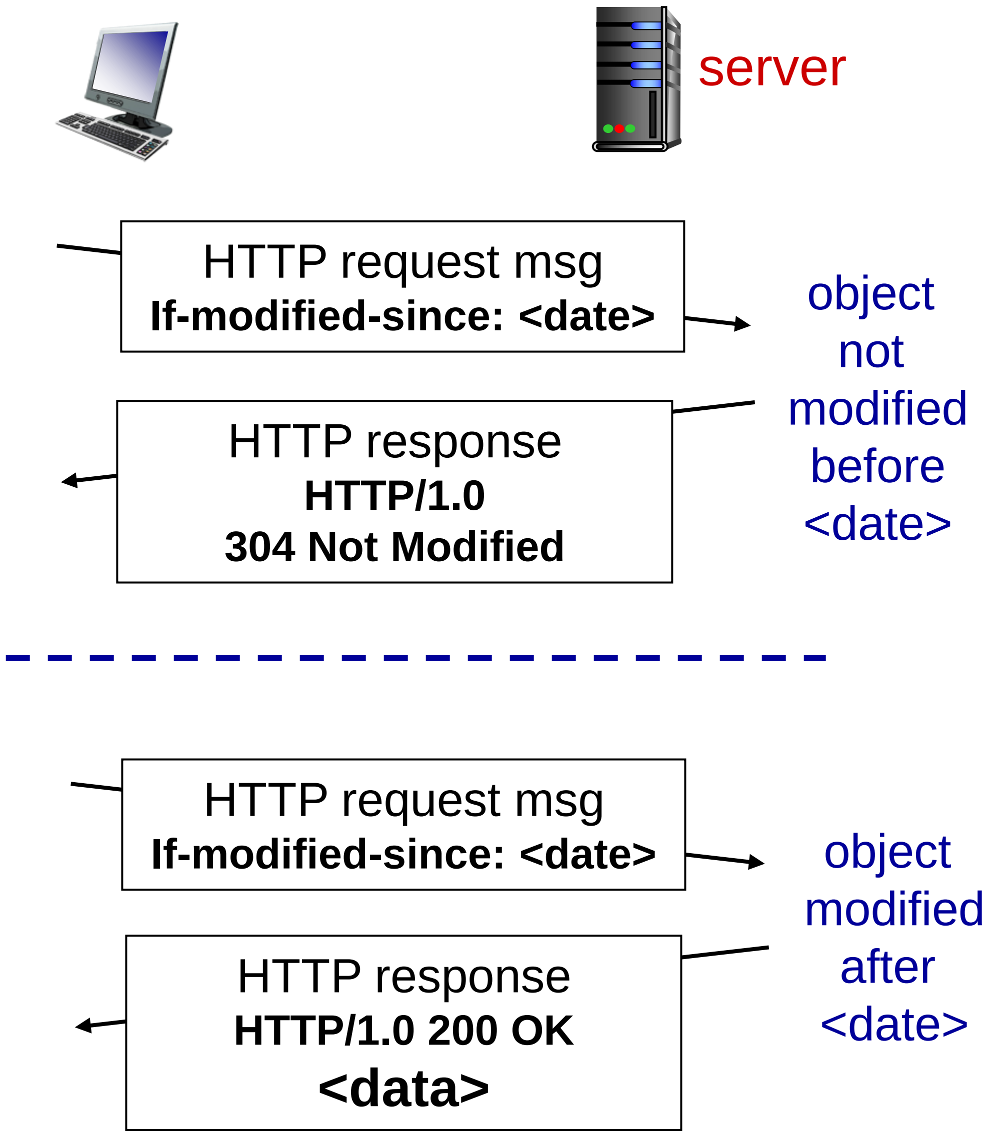


Conditional GET

- *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`



```
1770 4.073426582   128.119.245.12     10.10.182.50       TCP        74 80 → 55386 [SYN, ACK] Seq=0
1775 4.073525328   10.10.182.50       128.119.245.12     HTTP      613 GET /wireshark-labs/HTTP-wi
2103 4.670905143   128.119.245.12     10.10.182.50       HTTP      305 HTTP/1.1 304 Not Modified

Frame 2103: 305 bytes on wire (2440 bits), 305 bytes captured (2440 bits) on interface 0
Ethernet II, Src: Industri_29:95:1b (00:00:cd:29:95:1b), Dst: Micro-St_26:88:10 (d8:cb:8a:26:88:10)
Internet Protocol Version 4, Src: 128.119.245.12, Dst: 10.10.182.50
Transmission Control Protocol, Src Port: 80, Dst Port: 55382, Seq: 1, Ack: 548, Len: 239
Hypertext Transfer Protocol
  HTTP/1.1 304 Not Modified\r\n
  Date: Mon, 16 Oct 2017 10:21:57 GMT\r\n
  Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/5.4.16 mod_perl/2.0.10 Perl/v5.16.3\r\n
  Connection: Keep-Alive\r\n
  Keep-Alive: timeout=5, max=100\r\n
  ETag: "80-55ba3b4031322"\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.597379815 seconds]
  [Request in frame: 1775]
```