



PES
UNIVERSITY

CELEBRATING 50 YEARS

COMPUTER NETWORKS

TEAM NETWORKS

Department of Computer Science and Engineering

COMPUTER NETWORKS

Application Layer

Department of Computer Science and Engineering

Unit – 1 Application Layer

2.1 Principles of Network Applications

2.2 Web, HTTP and HTTPS

First, a quick review...

- web page consists of *objects*, each of which can be stored on different Web servers
- 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.,
- If a Web page contains HTML text and 5 JPEG images, then the Web page has 6 objects: the base HTML file plus the 5 images.

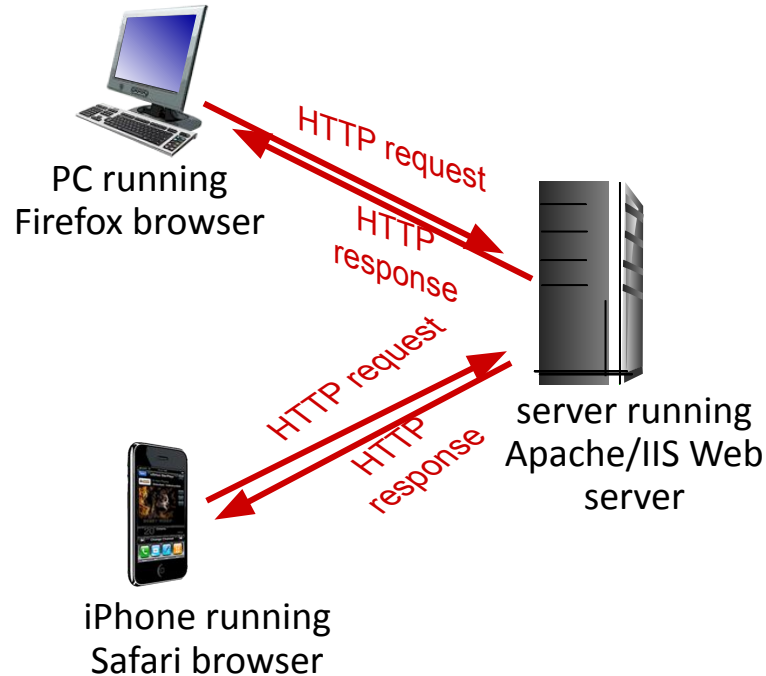
`www.someschool.edu/someDept/pic.gif`

host name

path name

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



Defined in RFC 1945; RFC 2616

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

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

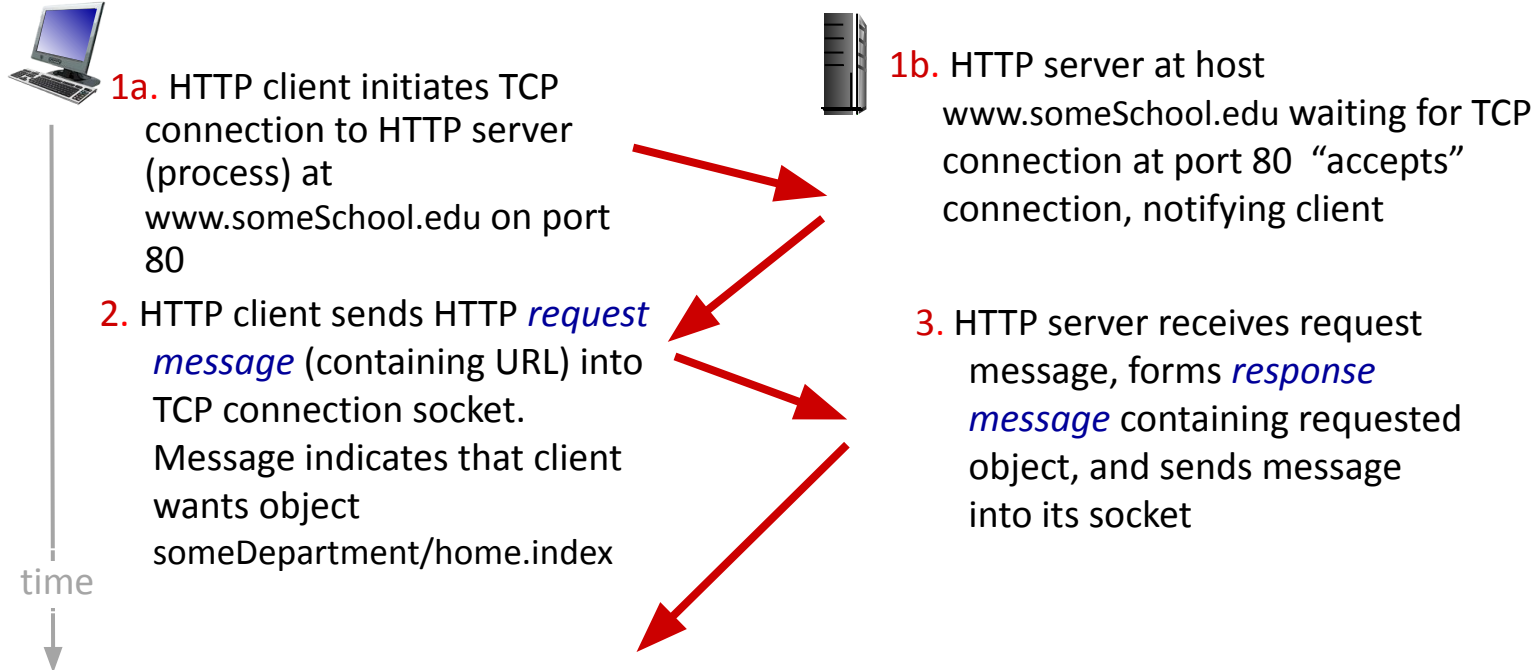
1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection closed

downloading multiple objects
required multiple connections

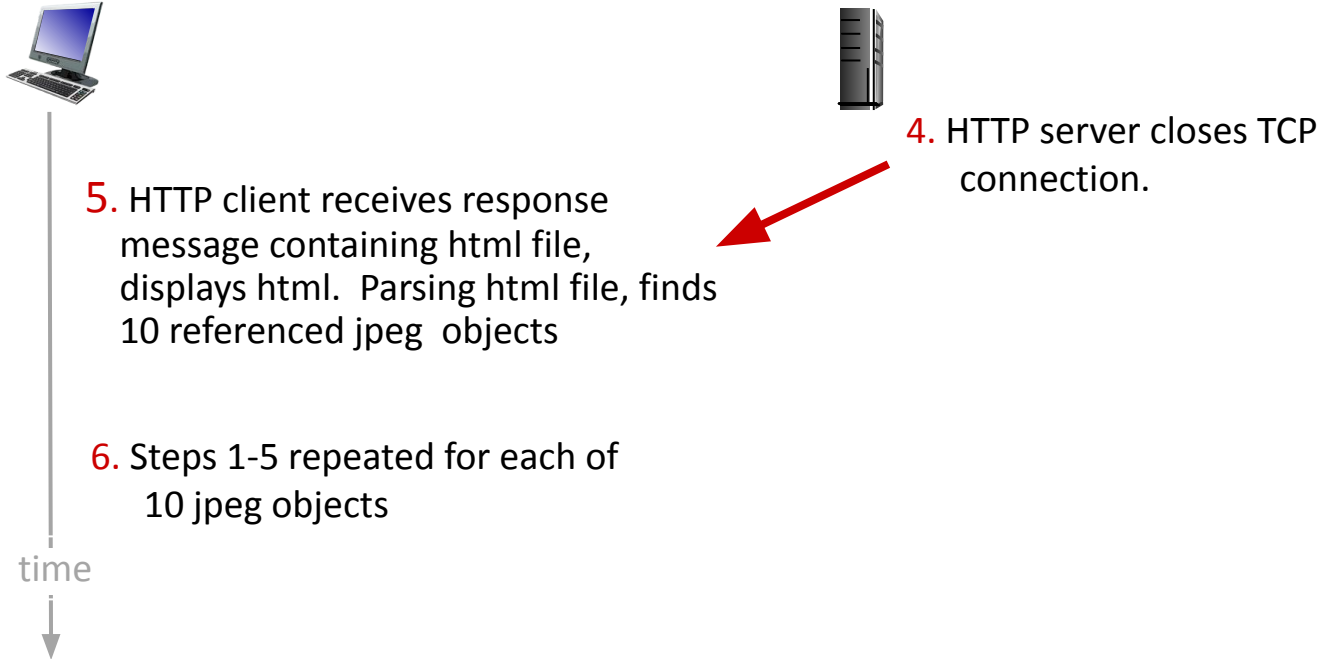
Persistent HTTP

- TCP connection opened to a server
- multiple objects can be sent over *single* TCP connection between client, and that server
- TCP connection closed

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



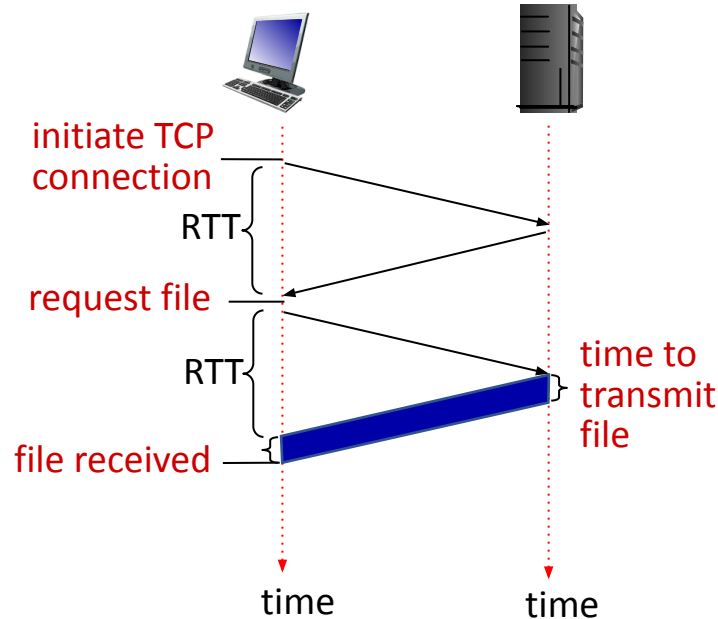
User enters URL: `www.someSchool.edu/someDepartment/home.index`
(containing text, references to 10 jpeg images)



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

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



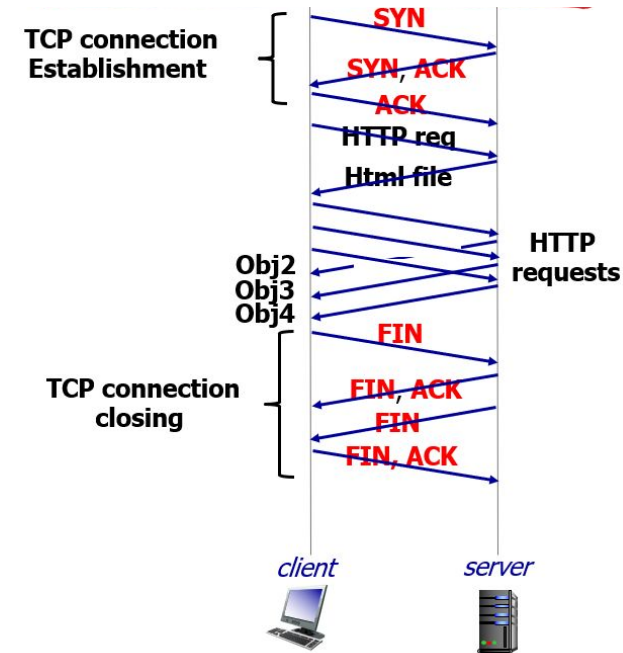
Non-persistent HTTP response time = $2RTT + \text{file transmission time}$

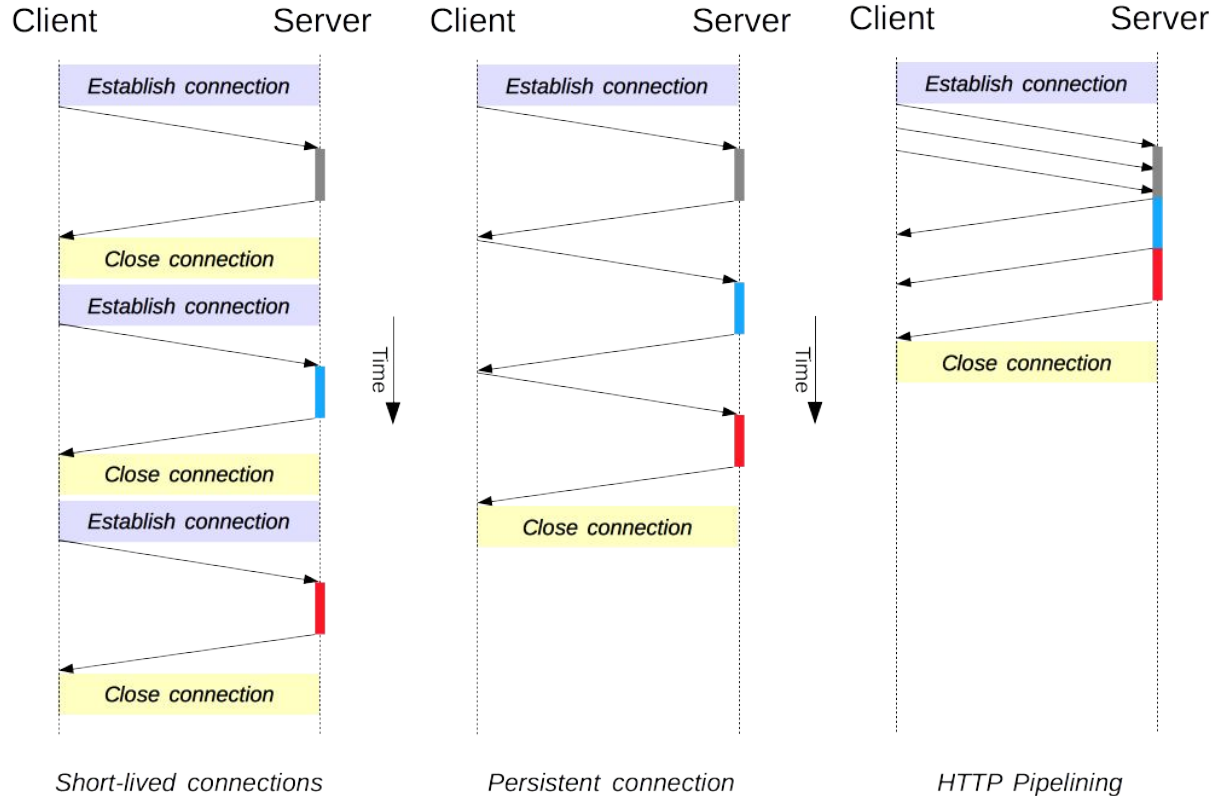
Non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection (TCP buffer and variables)
- 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)





Unit – 1 Application Layer

2.1 Principles of Network Applications

2.2 Web, HTTP and HTTPS

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

request line (GET, POST, HEAD commands) →

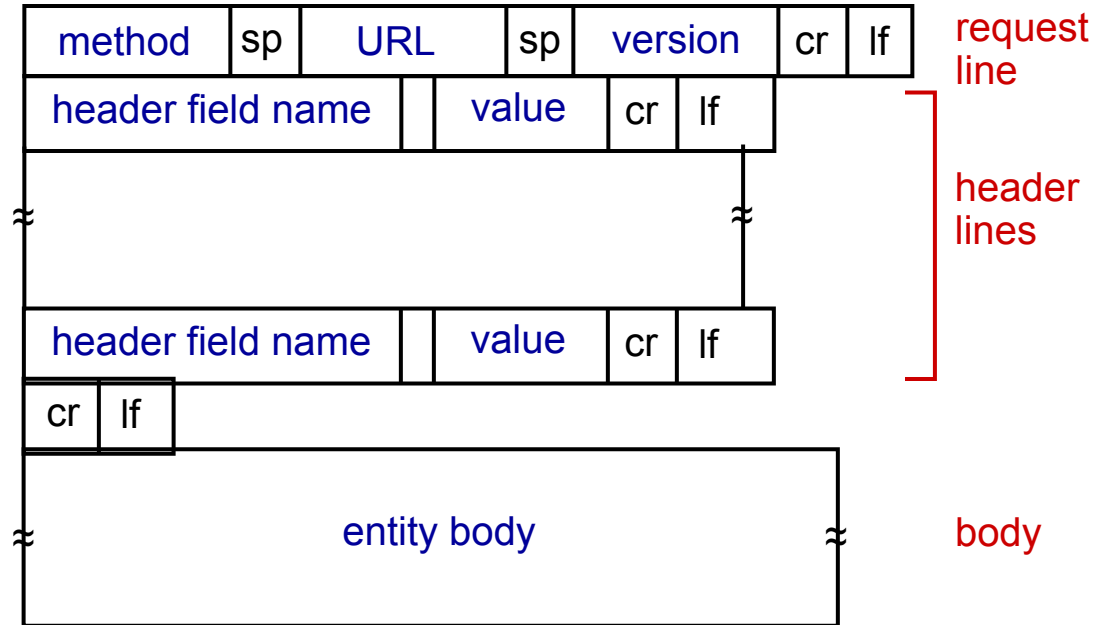
header lines →

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

carriage return character
line-feed character

```
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/



COMPUTER NETWORKS

HTTP Request Message – Wireshark Capture

Capturing from any

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
44	1.734232835	192.168.1.95	63.33.73.205	HTTP	677	GET /file1.html HTTP/1.1
50	1.831703556	63.33.73.205	192.168.1.95	HTTP	326	HTTP/1.1 200 OK (text/html)
52	1.874581455	192.168.1.95	63.33.73.205	HTTP	558	GET /favicon.ico HTTP/1.1
54	1.970307494	63.33.73.205	192.168.1.95	HTTP	326	HTTP/1.1 404 Not Found (application/json)

Frame 44: 677 bytes on wire (5416 bits), 677 bytes captured (5416 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 192.168.1.95, Dst: 63.33.73.205
Transmission Control Protocol, Src Port: 33862, Dst Port: 80, Seq: 1, Ack: 1, Len: 609
Hypertext Transfer Protocol
GET /file1.html HTTP/1.1\r\nHost: wireshark.grydeske.net\r\nConnection: keep-alive\r\nPragma: no-cache\r\nCache-Control: no-cache\r\nUpgrade-Insecure-Requests: 1\r\nUser-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.87 Safari/537.36\r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3\r\nReferer: http://localhost:8080/wireshark/wireshark-http.html\r\nAccept-Encoding: gzip, deflate\r\nAccept-Language: en-US,en;q=0.9,da;q=0.8\r\nCookie: __utma=231828553.1458339319.1536683537.1537430811.1537432942.3\r\n\r\n[Full request URI: http://wireshark.grydeske.net/file1.html]
[HTTP request 1/2]
[Response in frame: 50]
[Next request in frame: 52]

0000 00 04 00 01 00 06 9c eb e8 19 0a 4a 00 00 08 00J.....
0010 45 00 02 95 53 6b 40 00 40 06 9a 02 c0 a8 01 5f E...Sk@. @.....
0020 3f 21 49 cd 84 46 00 50 68 d6 3c f0 c3 38 b6 49 ?!I...F-P h<...8.I
0030 80 18 01 06 4d 7d 00 00 01 01 08 0a e0 c1 51 3b ...M}... ..Q;
0040 5f f3 8b d1 47 45 54 20 2f 66 69 6c 65 31 2e 68 ...GFT /file1.h

POST method:

- web page often includes form input
- user input sent from client to server in entity body of HTTP POST request message

HEAD method:

- requests headers (only) that would be returned *if* specified URL were requested with an HTTP GET method.

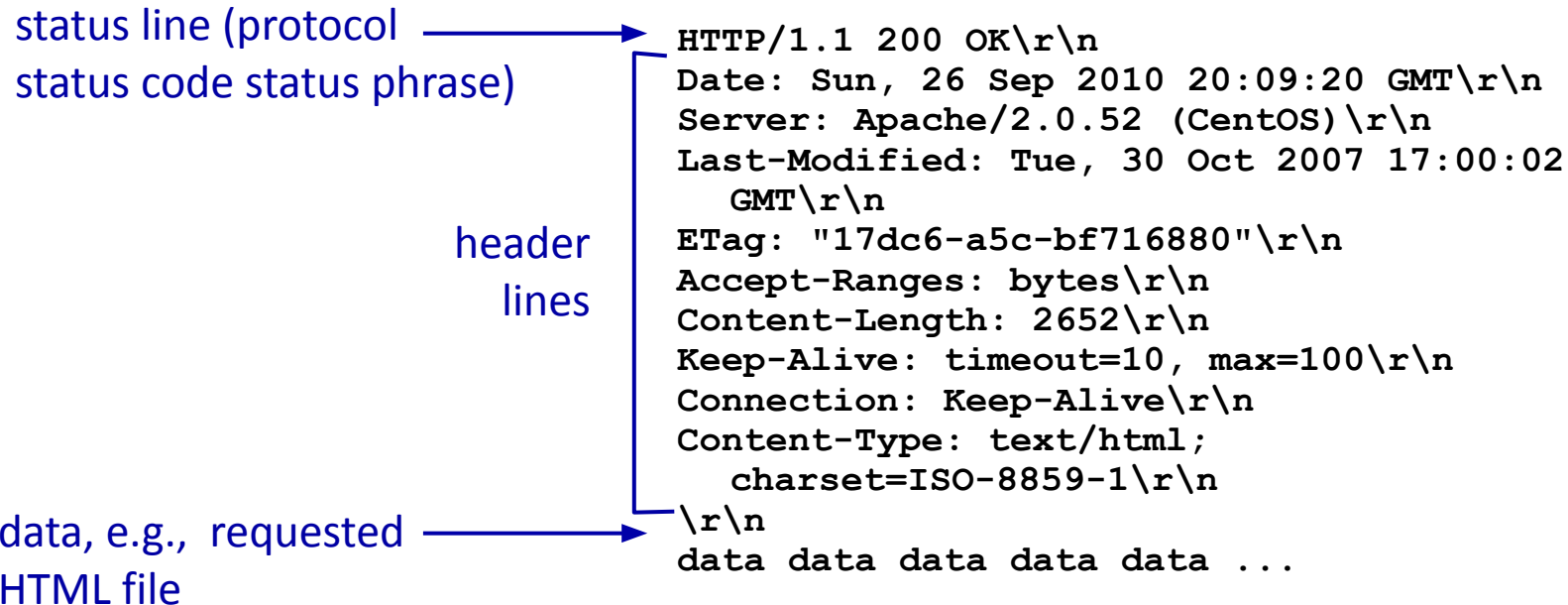
GET method (for sending data to server):

- include user data in URL field of HTTP GET request message (following a '?'):

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

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



COMPUTER NETWORKS

HTTP Response Message – Wireshark Capture

Capturing from Microsoft:\Device\NPF_{483C83F4-DCBA-4863-B523-3C4E1B03D06F} [Wireshark 1.8.5 (SVN Rev 47350 from /trunk-1.8)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: http Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
222	21:13:58.590670000	10.36.40.181	239.255.255.250	SSDP	528	NOTIFY * HTTP/1.1
223	21:13:58.590877000	Fe80::4195:59F3:544 ff02::c		SSDP	556	NOTIFY * HTTP/1.1
233	21:13:59.117254000	10.36.40.181	128.119.245.12	HTTP	473	GET /wireshark-labs/HTTP-wireshark-file3.html HTTP/1.1
241	21:13:59.150482000	128.119.245.12	10.36.40.181	HTTP	452	HTTP/1.1 200 OK (text/html)
242	21:13:59.190558000	10.36.40.181	239.255.255.250	SSDP	556	NOTIFY * HTTP/1.1
243	21:13:59.190758000	Fe80::4195:59F3:544 ff02::c		SSDP	584	NOTIFY * HTTP/1.1
245	21:13:59.443994000	10.36.40.181	128.119.245.12	HTTP	384	GET /favicon.ico HTTP/1.1
246	21:13:59.462702000	128.119.245.12	10.36.40.181	HTTP	532	HTTP/1.1 404 Not Found (text/html)
247	21:13:59.467414000	10.36.40.181	239.255.255.250	SSDP	542	NOTIFY * HTTP/1.1
248	21:13:59.467605000	Fe80::4195:59F3:544 ff02::c		SSDP	570	NOTIFY * HTTP/1.1

Frame 241: 452 bytes on wire (3616 bits), 452 bytes captured (3616 bits) on interface 0

- Ethernet II, Src: Cisco_4c:61:3f (00:1e:f7:4c:61:3f), Dst: HonHaiPr_0a:de:6b (cc:af:78:0a:de:6b)
- Internet Protocol Version 4, Src: 128.119.245.12 (128.119.245.12), Dst: 10.36.40.181 (10.36.40.181)
- Transmission Control Protocol, Src Port: http (80), Dst Port: 55990 (55990), Seq: 4381, Ack: 420, Len: 398
- [5 Reassembled TCP Segments (4778 bytes): #234(1423), #237(1460), #239(1460), #235(37), #241(398)]
- Hypertext Transfer Protocol
 - HTTP/1.1 200 OK\r\n
 - Date: wed, 27 Feb 2013 02:14:00 GMT\r\n
 - Server: Apache/2.2.3 (CentOS)\r\n
 - Last-Modified: wed, 27 Feb 2013 02:13:01 GMT\r\n
 - ETag: "d6c97-1194-50408540"\r\n
 - Accept-Ranges: bytes\r\n
 - Content-Type: text/html; charset=UTF-8\r\n
 - Content-Length: 4500\r\n
 - Connection: Keep-Alive\r\n
 - Age: 0\r\n
 - \r\n

0000 cc af 78 0a de 6b 00 1e f7 4c 61 3f 08 00 45 00 ..x.k.. .La?..E.
0010 01 b6 5f cd 00 00 3a 06 77 18 80 77 f5 0c 0a 24: w.w..\$.
0020 28 b5 00 50 da b6 70 e7 fd 49 a7 2b b3 a2 50 18 (.P..p..I.+..P..
0030 ff ff 16 ab 00 00 70 3e 3c 2f 70 3e 3c 70 3e 54p> </p><p>T
0040 68 65 20 65 6e 75 6d 65 72 61 74 69 6f 6e 20 69 he enum ration i
0050 6e 20 74 68 65 20 43 6f 6e 73 74 69 74 75 74 69 n the co nstituti
0060 6f 6e 2c 20 6f 66 20 63 65 72 74 61 69 6e 20 74 on of c ertain r

Frame (452 bytes) Reassembled TCP (4778 bytes)

Frame (frame), 452 bytes Packets: 336 Displayed: 40 Marked: 0 Profile: Default

- 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)

400 Bad Request

- request msg not understood by server

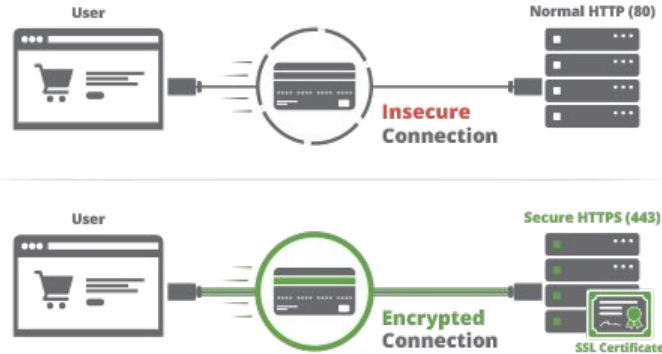
404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

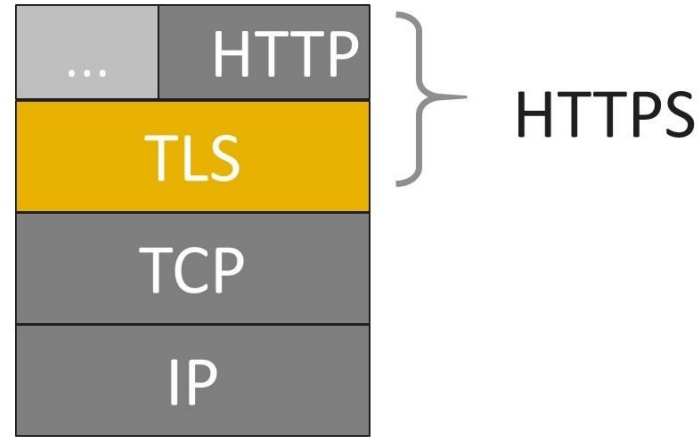


HTTP vs HTTPS



- HTTPS is HTTP with encryption – All communications between browser and server are encrypted (bi-directional).
- 'S' refers 'Secure' or HTTP over Secure Socket Layer.
- Uses TLS (SSL) to encrypt normal HTTP requests and responses.
- Attackers can't read the data crossing the wire and you know you are talking to the server you think you are talking too.

- HTTP + TLS -> Encrypted
- Uses port no. 443 for data communication.
- HTTPS is based on public/private-key cryptography.
 - The public key is used for encryption
 - The secret private key is required for decryption.
- SSL certificate is a web server's digital certificate issued by a third party CA.
 - Create an encrypted connection and establish trust.
- Is my certificate SSL or TLS?



Any message encrypted with Bob's public key can be only decrypted with Bob's private key.

- Step 1: Browser requests secure pages (HTTPS) from a server.
 - Step 2: Server sends its public key with its SSL certificate (digitally signed by a third party – CA).
 - Step 3: On receipt of certificate, browser verifies issuer's digital signature. (green padlock key)
 - Step 4: Browser creates a symmetric key (shared key), keeps one and gives a copy to server. Encrypts it using server's public key.
 - Step 5: On receipt of encrypted secret key, decrypts it using its private key and gets browser's secret key.
- Asymmetric and Symmetric key algorithms work together.
 - Asymmetric key algorithm – verify identity of the owner & its public key -> Establish trust.
 - Once connection is established, Symmetric key algorithm is used to encrypt and decrypt the traffic.

COMPUTER NETWORKS

How does SSL works?



Client messages server to initiate SSL/TLS communication



Server sends back an encrypted public key/certificate.



Client checks the certificate, creates and sends an encrypted key back to the server
(If the certificate is not ok, the communication fails)



Server decrypts the key and delivers encrypted content with key to the client

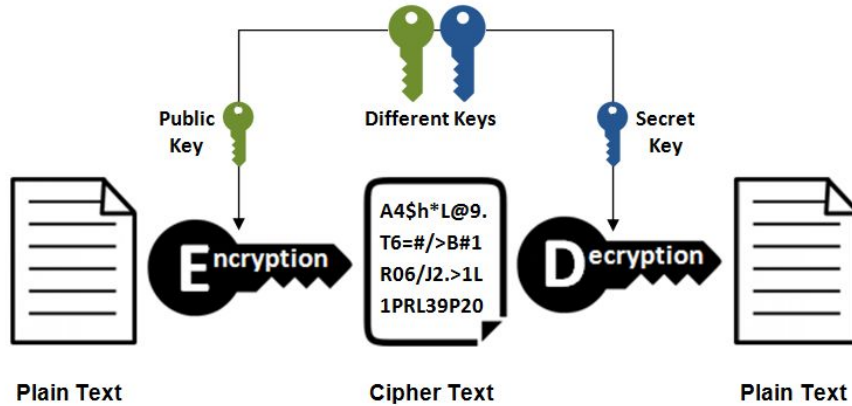


Client decrypts the content completing the SSL/TLS *handshake*

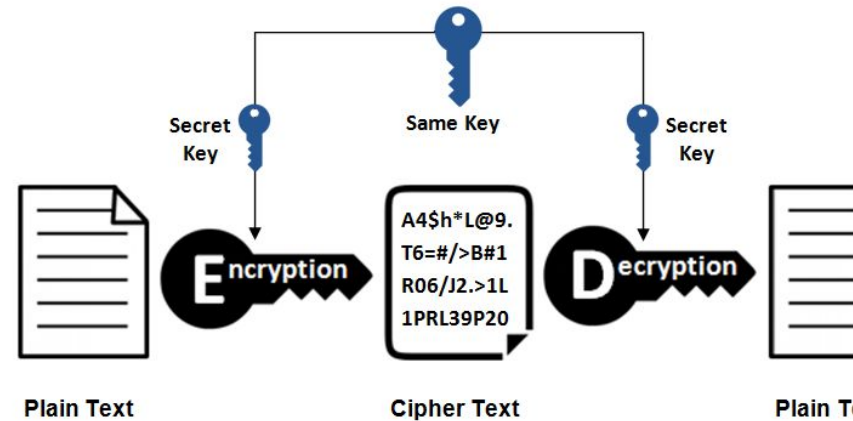


- Stronger Google ranking.
- Updated browser labels.
- Improved security.
- Increased customer confidence / safer experience.
- Build customer trust and improve conversions.

Asymmetric Encryption



Symmetric Encryption



- Website/HTTP/Internet cookies
- Piece of data from a specific website
- Stored on a user's computer
- Allows sites to keep track of users
- Eg: language selection



Cookies

This site uses cookies to offer you a better browsing experience. Find out more on [how we use cookies and how you can change your settings](#).

I accept cookies

I refuse cookies

This website uses cookies to ensure you get the best experience on our website.

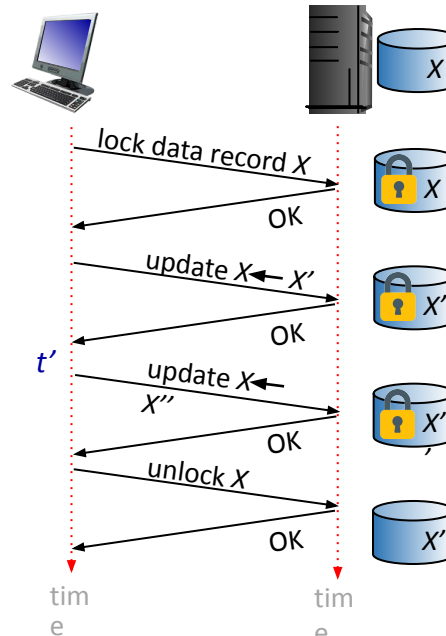
[Learn more](#)

Got it!

Recall: HTTP GET/response interaction is *stateless*

- no notion of multi-step exchanges of HTTP messages to complete a Web “transaction”
 - no need for client/server to track “state” of multi-step exchange
 - all HTTP requests are independent of each other
 - no need for client/server to “recover” from a partially-completed-but-never-completely-completed transaction

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



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

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

four components:

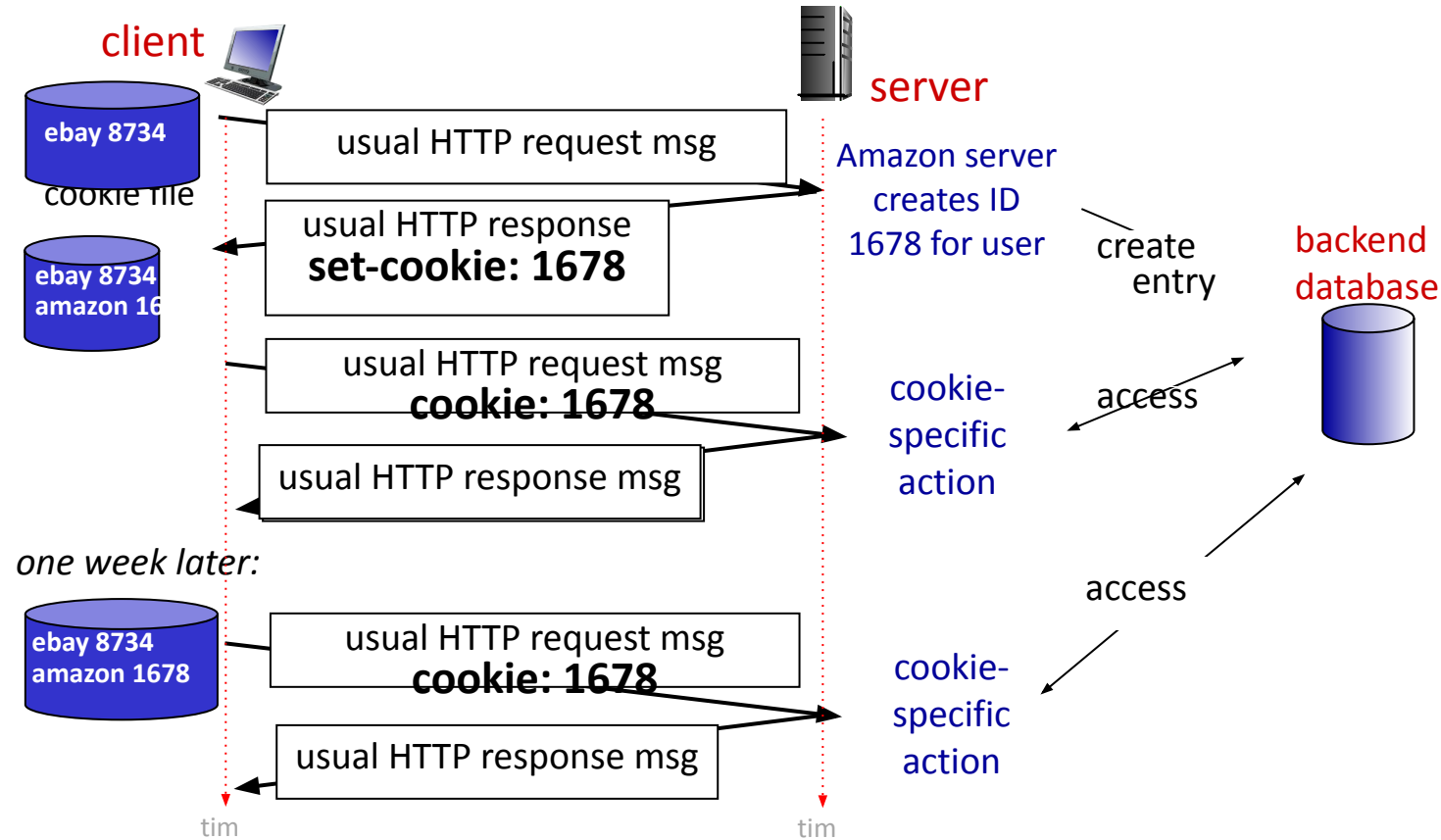
- 1) cookie header line of HTTP *response* message
- 2) cookie header line in next HTTP *request* message
- 3) cookie file kept on user's host, managed by user's browser
- 4) back-end database at Web site

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

COMPUTER NETWORKS

Maintaining user/server state: cookies (more)



What cookies can be used for:

- track user's browsing history
- remembering login details
- track visitor count
- shopping carts
- recommendations
- save coupon codes for you

Challenge: How to keep state:

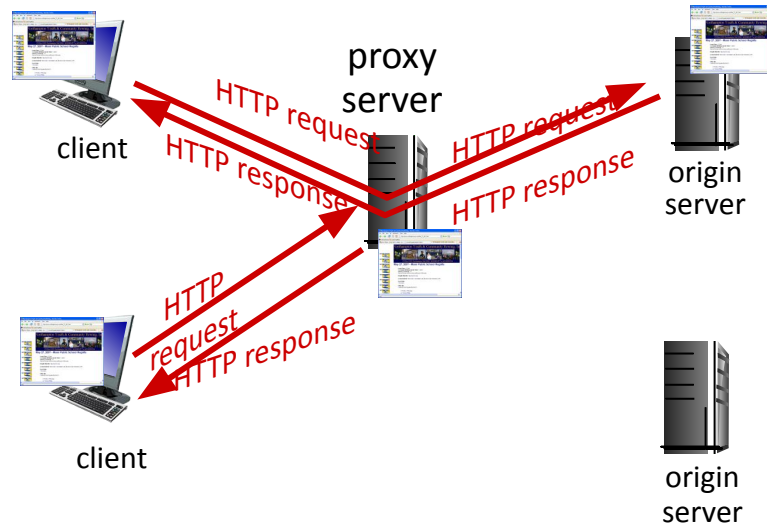
- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: HTTP messages carry state

cookies and privacy: aside

- 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

Goal: satisfy client request without involving origin server

- user configures browser to point to a *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



- Web 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 (speed)
 - cache is closer to client
- reduce traffic on an institution's access link (saves bandwidth)
- internet is dense with caches
 - enables “poor” content providers to more effectively deliver content
- privacy – surf the internet anonymously
- activity logging

COMPUTER NETWORKS

Caching example

$$(15 \text{ req/sec}) * (100 \text{ Kbits/req}) / (1.54 \text{ Mbps}) = 0.974$$

$$(15 \text{ req/sec}) * (100 \text{ Kbits/req}) / (1 \text{ Gbps}) = 0.0015$$

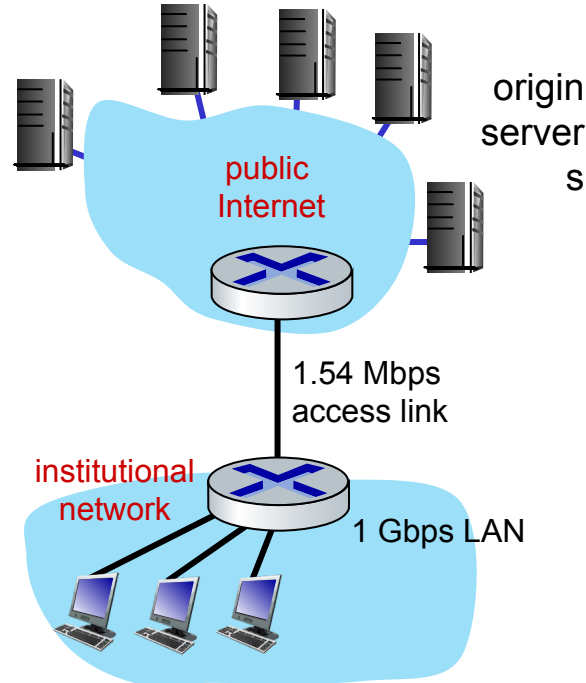
Scenario:

- 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
 - average data rate to browsers: 1.50 Mbps

Performance:

- LAN utilization: .0015
- access link utilization = .97
- end-end delay = Internet delay + access link delay + LAN delay
= 2 sec + minutes + usecs

problem: large
delays at high
utilization!



COMPUTER NETWORKS

Caching example: buy a faster access link

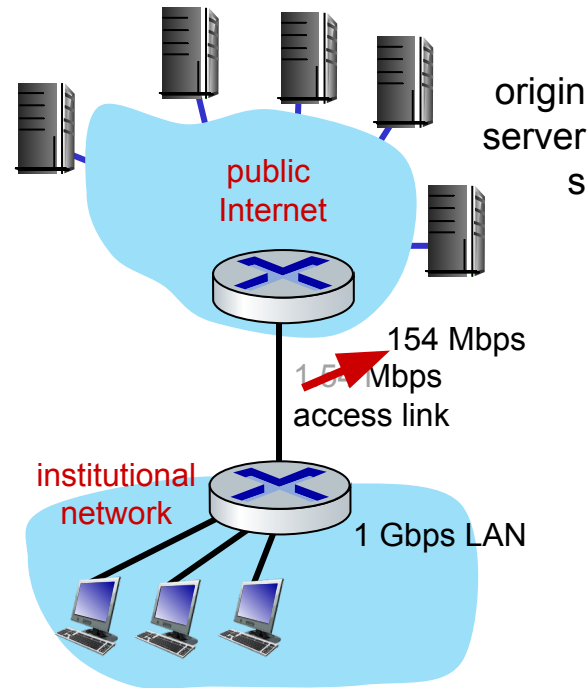
Scenario:

- access link rate: ~~1.54~~ ¹⁵⁴ Mbps
- RTT from institutional router to server: 2 sec
- Web object size: 100K bits
- Avg request rate from browsers to origin servers: 15/sec
 - avg data rate to browsers: 1.50 Mbps

Performance:

- LAN utilization: .0015
- access link utilization = ~~.97~~ ^{.0097}
- end-end delay = Internet delay + access link delay + LAN delay
= 2 sec + ~~minutes~~ ^{msecs}

Cost: faster access link (expensive!) ^{msecs}

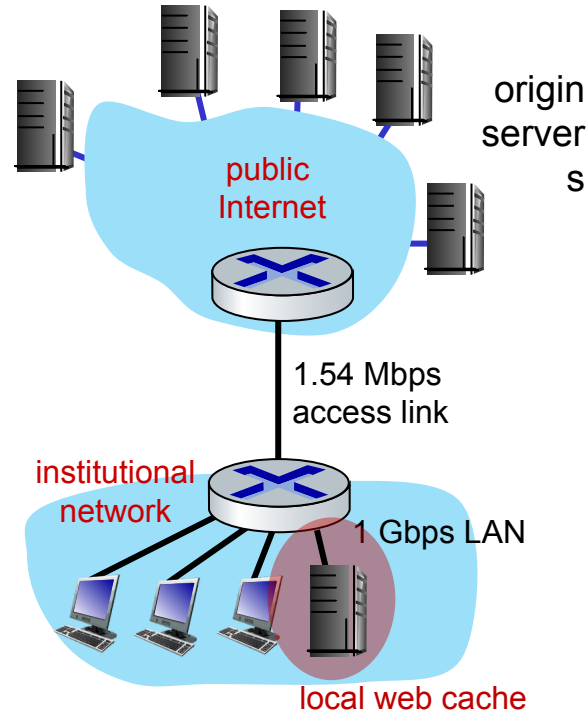


COMPUTER NETWORKS

Caching example: install a web cache

Scenario:

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



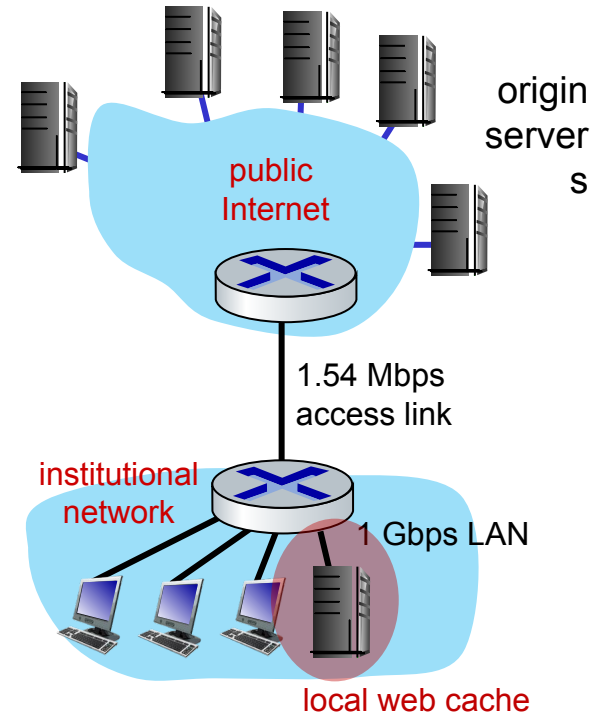
Performance: *How to compute link utilization, delay?*

- LAN utilization: .?
- access link utilization = ?
- average end-end delay = ?

Cost: web cache (cheap!)

Calculating access link utilization, end-end delay with cache:

- suppose cache hit rate is 0.4: 40% requests satisfied at cache, 60% requests satisfied at origin
- access link: 60% of requests use access link
- data rate to browsers over access link
 $= 0.6 * 1.50 \text{ Mbps} = .9 \text{ Mbps}$
- utilization $= 0.9 / 1.54 = .58$
- average end-end delay
 $= 0.6 * (\text{delay from origin servers})$
 $+ 0.4 * (\text{delay when satisfied at cache})$
 $= 0.6 (2.01) + 0.4 (\sim \text{msecs}) = \sim 1.2 \text{ secs}$



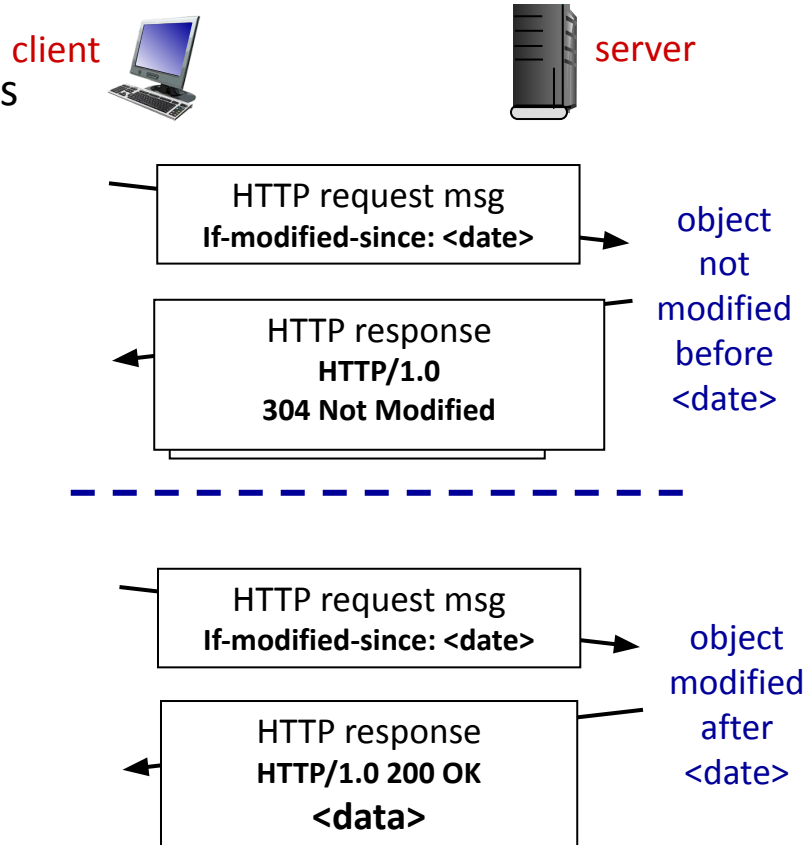
lower average end-end delay than with 154 Mbps link (and cheaper too!)

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



COMPUTER NETWORKS

Conditional Get (more)

Microsoft \Device\NPF_{483C83F4-DCBA-4863-B523-3C4E1B03D06F} [Wireshark 1.8.5 (SVN Rev 47350 from /trunk-1.8)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: http Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
4	20:33:03.198438000	10.36.40.181	149.152.32.102	HTTP	715	GET /ssp_director/p.php?a=UuFRxiQyPSeqYHTlPz04Jzu6ISs7PT4uNio4MTI%2BNjkmKy0gPScjKDonNz8Xa
321	20:33:03.427289000	149.152.32.102	10.36.40.181	HTTP	1514	[TCP out-of-order] HTTP/1.1 200 OK (JPEG JFIF image)
340	20:33:09.383079000	10.36.40.181	128.119.245.12	HTTP	473	GET /wireshark-labs/HTTP-wireshark-file2.html HTTP/1.1
341	20:33:09.407557000	128.119.245.12	10.36.40.181	HTTP	701	HTTP/1.1 200 OK (text/html)
343	20:33:09.677244000	10.36.40.181	128.119.245.12	HTTP	384	GET /favicon.ico HTTP/1.1
344	20:33:09.689986000	128.119.245.12	10.36.40.181	HTTP	532	HTTP/1.1 404 Not Found (text/html)
347	20:33:14.318343000	10.36.40.181	128.119.245.12	HTTP	586	GET /wireshark-labs/HTTP-wireshark-file2.html HTTP/1.1
348	20:33:14.331881000	128.119.245.12	10.36.40.181	HTTP	319	HTTP/1.1 304 Not Modified
349	20:33:14.410192000	10.36.40.181	128.119.245.12	HTTP	384	GET /favicon.ico HTTP/1.1
350	20:33:14.426443000	128.119.245.12	10.36.40.181	HTTP	532	HTTP/1.1 404 Not Found (text/html)

Ethernet II, Src: HonHaiPr_0a:de:6b (cc:af:78:0a:de:6b), Dst: Cisco_4c:61:3f (00:1e:f7:4c:61:3f)

Internet Protocol Version 4, Src: 10.36.40.181 (10.36.40.181), Dst: 128.119.245.12 (128.119.245.12)

Transmission Control Protocol, Src Port: 55404 (55404), Dst Port: http (80), Seq: 750, Ack: 1126, Len: 532

Hypertext Transfer Protocol

GET /wireshark-labs/HTTP-wireshark-file2.html HTTP/1.1\r\n

Host: gaia.cs.umass.edu\r\n

Connection: keep-alive\r\n

Cache-Control: max-age=0\r\n

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.22 (KHTML, like Gecko) Chrome/25.0.1364.97 Safari/537.22\r\n

Accept-Encoding: gzip,deflate,sdch\r\n

Accept-Language: en-US,en;q=0.8\r\n

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3\r\n

If-Modified-Since: wed, 27 Feb 2013 01:33:01 GMT\r\n

Full request URI: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html

01d0 20 49 53 4f 2d 38 38 35 39 2d 31 2c 75 74 66 2d ISO-8859-1,utf-8;q=0.7,*;q=0.3

01e0 38 3b 71 3d 30 2e 37 2c 2a 3b 71 3d 30 2e 33 0d .If-None-Match: "d6c96-1 73-c1336"

01f0 0a 49 66 2d 4e 6f 6e 65 2d 4d 61 74 63 68 3a 20 d40..If-Modified-Since: wed, 27 Feb 2013 01:33:01 GMT...

0200 22 64 36 63 39 36 2d 31 37 33 2d 63 31 33 33 36

0210 64 34 30 22 0d 0a 49 66 2d 4d 6f 64 69 66 69 65

0220 64 2d 53 69 6e 63 65 3a 20 57 65 64 2c 20 32 37

0230 20 46 65 62 20 32 30 31 33 20 30 31 3a 33 33 3a

0240 30 31 20 47 4d 54 0d 0a 0d 0a

Text item (text), 50 bytes

Packets: 367 Displayed: 10 Marked: 0 Dropped: 0

Profile: Default



PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

TEAM NETWORKS

Department of Computer Science and Engineering