

HTTP – Stateless Protocol

– server forgets about each client as soon as it delivers response ...

- **stateless behavior is issue when:**
 - server wants to have accurate count of site visitors
 - server wants to restrict user access, etc.
 - server wants to personalize pages for each client, or remember selections they made

Cookie Technology

– allows sites to keep track of users



WEBSITE
COOKIES

- (1) when a server identifies a new user, it adds **Set-Cookie header** to its response, containing an identifier for that user
- (2) the client is expected to store the info from the Set-Cookie header on its disk, and send this info back to the server by means of **Cookie header** in all subsequent requests made to the same server / Web domain ...

A cookie is a short piece of data, not code, which is sent from a web server to a web browser when that browser visits the server's site. The cookie is stored on the user's machine, but it is not an executable program and cannot **directly** harm the machine.

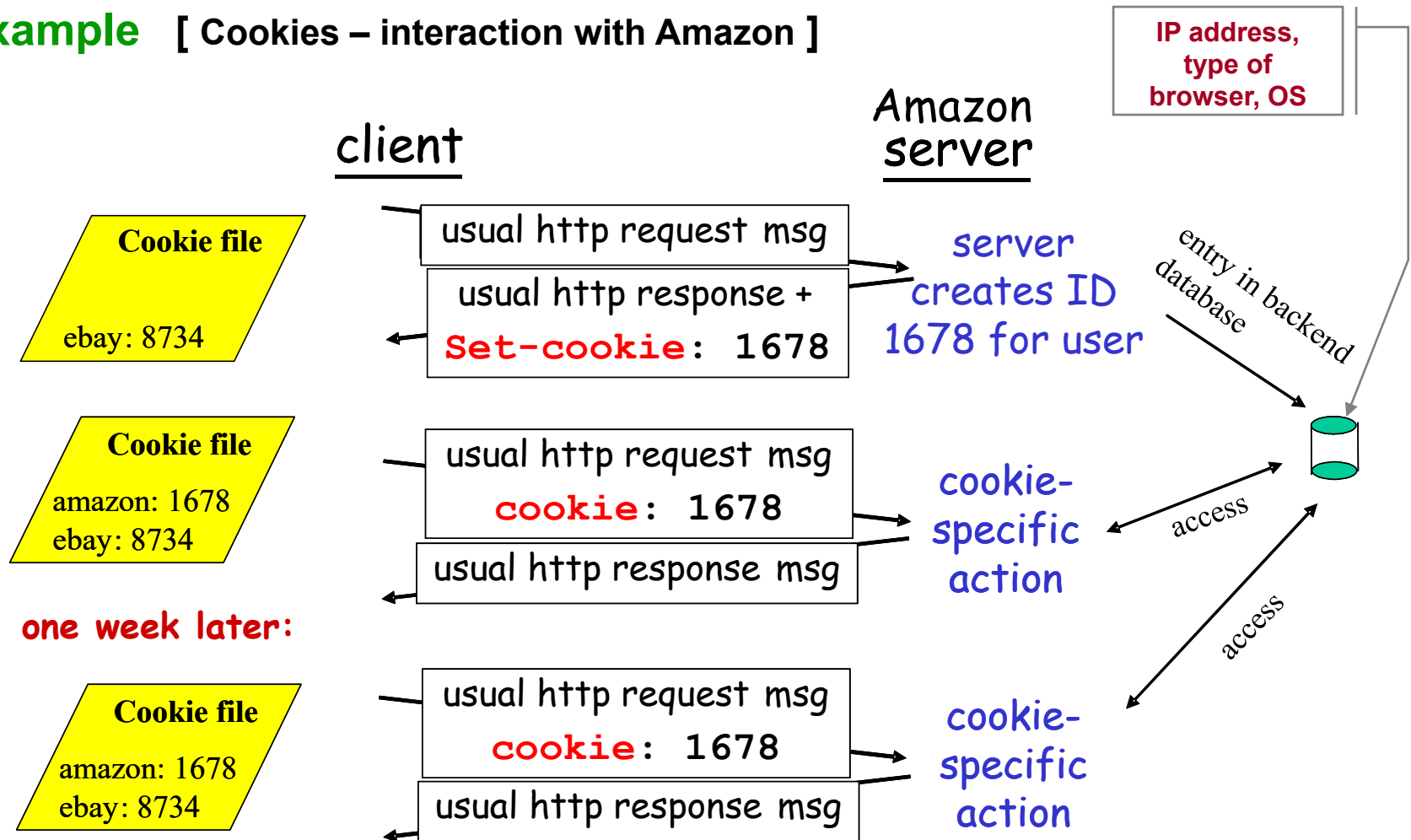
Set-Cookies & Cookies HTTP Headers

– the two headers are used to set and exchange cookies between client and server machines

- **Set-Cookies** is a 'Response Header' and is used by the server to set-up a new cookie on the client
- **Cookies** is a 'Request Header' and is used by the client to return the cookie(s) to the server



Example [Cookies – interaction with Amazon]



Not all sites use cookies, but major portals (e.g. Yahoo), e-commerce (e.g. Amazon), and advertising sites make extensive use of cookies.

Example [HTTP response and request containing cookies ...]

```
1 HTTP/2.0 200 OK
2 Content-type: text/html
3 Set-Cookie: yummy_cookie=choco
4 Set-Cookie: tasty_cookie=strawberry
5
6 [page content]
```

```
1 GET /sample_page.html HTTP/2.0
2 Host: www.example.org
3 Cookie: yummy_cookie=choco; tasty_cookie=strawberry
```

Example [Cookies – interaction with www.amazon.com]

The image shows a Wireshark packet capture of an HTTP interaction. The packet list shows a GET request for `/exec/obidos/subst` and a corresponding 302 response. The packet details pane for the response shows the following headers:

```
HTTP/1.1 302\r\n
  Response Code: 302
  Date: Mon, 14 Nov 2005 19:04:26 GMT\r\n
  Server: Server\r\n
  Set-Cookie: session-id-time=1132560000; path=/; domain=.amazon.com; expires=Monday, 21-M\r\n
  Set-Cookie: session-id=002-2690063-4102467; path=/; domain=.amazon.com; expires=Monday,\r\n
  Location: http://www.amazon.com/exec/obidos/subst/home/home.html/002-2690063-4102467\r\n
  Vary: Accept-Encoding,User-Agent\r\n
  Content-Encoding: gzip\r\n
  Connection: close\r\n
  Transfer-Encoding: chunked\r\n
  Content-Type: text/html\r\n
  \r\n
  HTTP chunked response
  Content-encoded entity body (gzip)
```

The packet bytes pane shows the raw data of the response, including the status line `HTTP/1.1 302` and the date `Mon, 14 Nov 2005 19:04:26 GMT`.

Example [Cookies – interaction with www.amazon.com (cont.)]

The image shows a Wireshark packet capture of an HTTP GET request to Amazon.com. The packet list shows a sequence of frames: a GET request (frame 44), a 302 redirect (frame 46), a continuation (frame 47), another GET request (frame 50), a 200 OK response (frame 54), and a continuation (frame 55). The packet details pane for frame 50 shows the HTTP request structure, including the request method, headers, and cookies. The cookies are highlighted with a red dashed box.

No.	Time	Source	Destination	Protocol	Info
44	2.107180	130.63.86.182	207.171.175.29	HTTP	GET /exec/obidos/subst
46	2.161473	207.171.175.29	130.63.86.182	HTTP	HTTP/1.1 302
47	2.161500	207.171.175.29	130.63.86.182	HTTP	Continuation
50	2.172205	130.63.86.182	207.171.175.29	HTTP	GET /exec/obidos/subst
54	2.696006	207.171.175.29	130.63.86.182	HTTP	HTTP/1.1 200 OK
55	2.697218	207.171.175.29	130.63.86.182	HTTP	Continuation

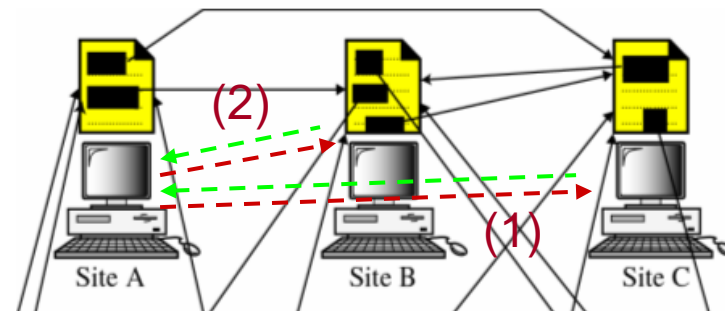
Frame 50 (551 bytes on wire, 551 bytes captured)
Ethernet II, Src: 00:0d:56:1f:4f:2e, Dst: 00:0c:cf:40:15:0a
Internet Protocol, Src Addr: 130.63.86.182 (130.63.86.182), Dst Addr: 207.171.175.29 (207.171.175.29)
Transmission Control Protocol, Src Port: 1684 (1684), Dst Port: http (80), Seq: 787, Ack: 92
Hypertext Transfer Protocol
GET /exec/obidos/subst/home/home.html/002-2690063-4102467 HTTP/1.1\r\n
Request Method: GET
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, app
Accept-Language: en-us\r\n
Accept-Encoding: gzip, deflate\r\n
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)\r\n
Host: www.amazon.com\r\n
Connection: Keep-Alive\r\n
Cookie: session-id-time=1132560000; session-id=002-2690063-4102467\r\n\r\n

0030 f7 51 59 ca 00 00 47 45 54 20 2f 65 78 65 63 2f .QY...GE T /exec/
0040 6f 62 69 64 6f 73 2f 73 75 62 73 74 2f 68 6f 6d obidos/s ubst/hom
0050 65 2f 68 6f 6d 65 2e 68 74 6d 6c 2f 30 30 32 2d e/home.h tml/002-
0060 32 36 39 30 30 36 33 2d 34 31 30 32 34 36 37 20 2690063- 4102467
0070 48 54 54 50 2f 31 2e 31 0d 0a 41 63 63 65 70 74 HTTP/1.1 ..Accept

Hypertext Transfer Protocol: P: 680 D: 429 M: 0

Issues with Cookies – although most of the time cookies are used for useful and benign purposes, there are ways to abuse them:

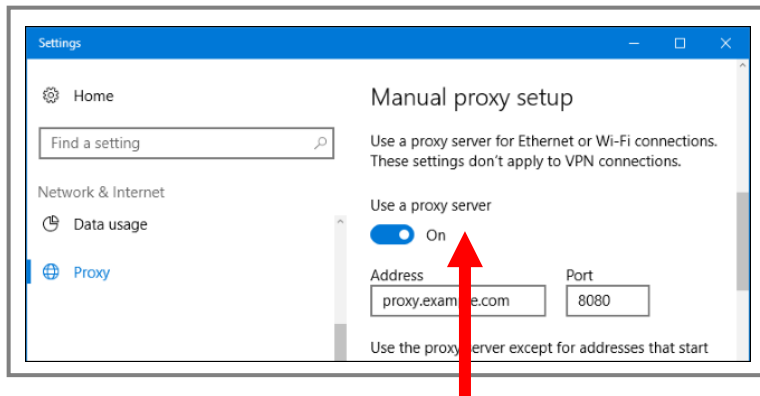
- (1) **Undesirable Cookies:** any server can set a cookie for any reason – since some Web browsers do not inform the user when a cookie is being set, he may not even be aware that this is happening
- (2) **Third-Party or Unintentional Cookies:** a cookie may be set by any server to which a request is sent, whether the user realizes it or not
 - by retrieving www.myfavoritesite.com/index.html that contains a reference to a tiny image at www.bigbrotherishere.com, the second site can set a cookie on your machine ...



indirect HTML referencing

Web Caching

Web Cache / Proxy Server – an intermediary entity that satisfies HTTP requests on the behalf of an origin Web server



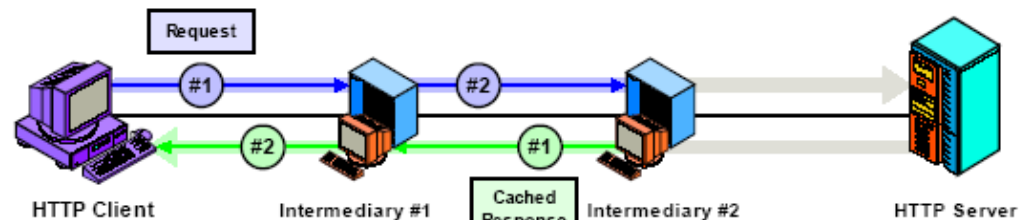
- users browser must be configured so that all requests are first directed to its LAN's Web cache
- Web cache checks if it has a copy of a requested object stored locally
 - if it does, it sends cached object back to client
 - if it does NOT, it requests object from the origin server, and then returns it to the client

- **advantages of caching:**

- (1) **reduce response time for clients' requests**
- (2) **reduce traffic on an institution's access link**

- **disadvantages of caching**

slower performance if object is not cached – extra layer is added



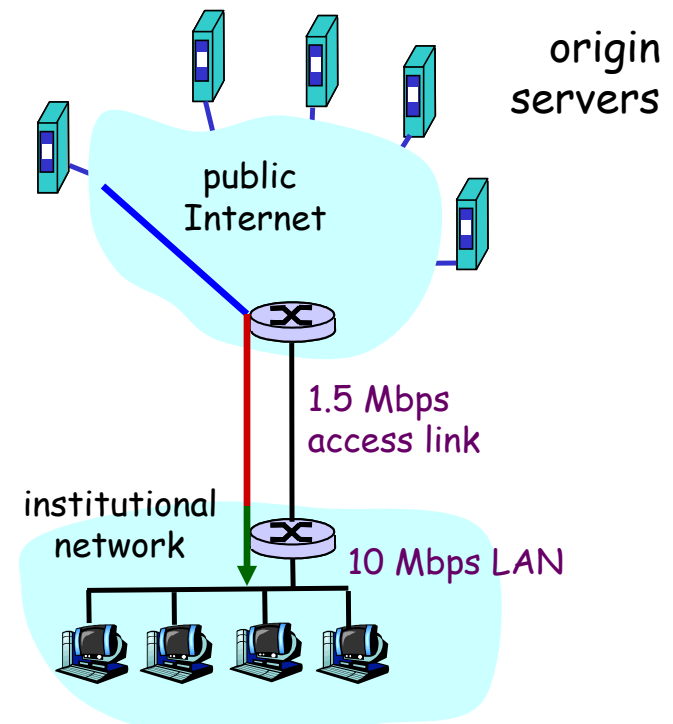
Example [caching]

Assumptions

- average object size = 100,000 bits
- avg. request rate from institution's browsers to origin servers = 15/sec

Consequences

- utilization on LAN =
$$= \frac{(15 \text{ req/sec} * 0.1 \text{ Mbit/req})}{10 \text{ Mbit/sec}} = 0.15 = 15\%$$
- utilization on access link =
$$= \frac{(15 \text{ req/sec} * 0.1 \text{ Mbit/req})}{1.5 \text{ Mbit/sec}} = 1 = 100\%$$
- total delay =
$$= \text{Internet delay} + \text{access delay} + \text{LAN delay} = 2 \text{ sec} + \text{minutes} + \text{milliseconds} = \text{several min}$$



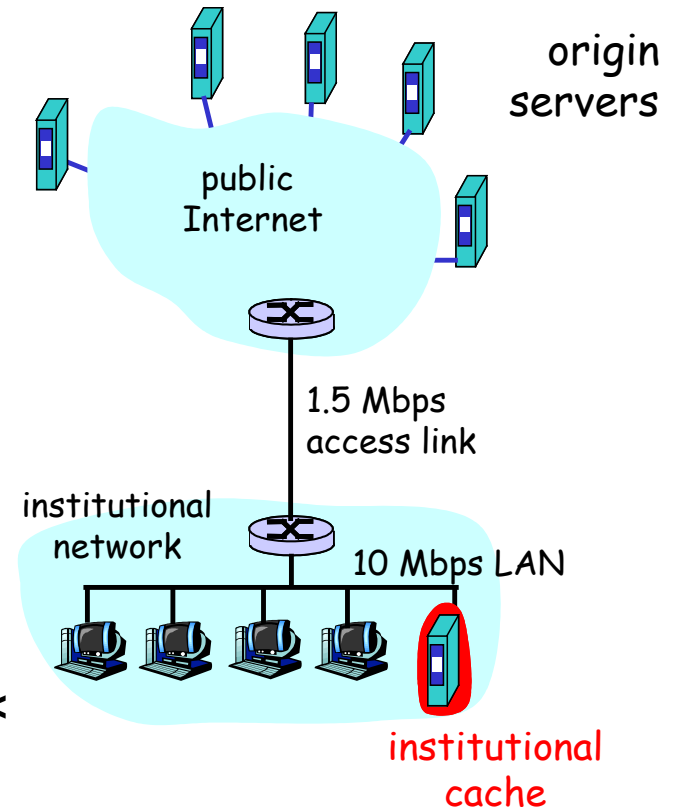
Possible solution: increase access rate – costly!!!

Install cache

- suppose hit rate (fraction of requests satisfied by cache) is 0.4 (typically 0.2 – 0.7)

Consequence

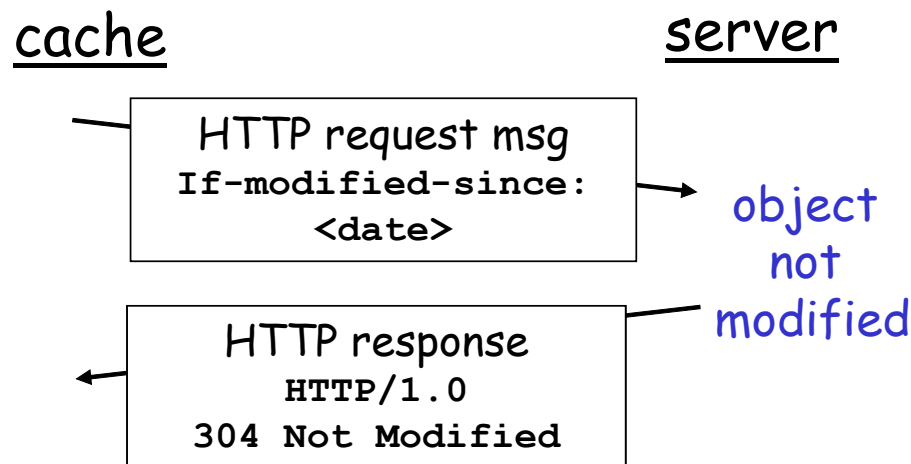
- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible access delay (say 10 msec)
- total average delay =
 $0.4 * \text{LAN access delay} + 0.6 * \text{WAN access delay} <$
 $< 0.4 * 0.01 \text{ sec} + 0.6 * (2 + 0.01 + 0.01) \text{ sec} \approx$
 $< \text{several seconds}$



For this solution, the institution has to (purchase and) install a Web cache. Many caches use public-domain software that runs on inexpensive PCs.

Web Cache Challenge – an object residing in the cache might be stale, i.e. the object may have been modified since the copy was cached

- goal: want to keep cache up-to-date, but don't send object if cache latest version
- solution: **conditional GET** – requires that GET method contains **If-Modified-Since** header line
 - server will include requested object in response only if object has been modified since specified date
 - this saves bandwidth and reduces user-perceived response time, especially if object is large



Example [If-Modified-Since HTTP Header]



https://developer.mozilla.org/en-US/docs/Web/HTTP/Conditional_requests