Caching and content distribution

TELE9751 lecture notes

19W*

Lecture outline

- Proxies
- Caches
  - Why
  - Where
  - How
- Content Distribution Networks

- Some themes of this course

Resources

Textbooks:

- Keshav: nothing
- Varghese: only caching applied to processor memories
- Tanenbaum: § 7.3.5: Caching, Content Delivery
- Kurose and Ross: § 2.2.5-6

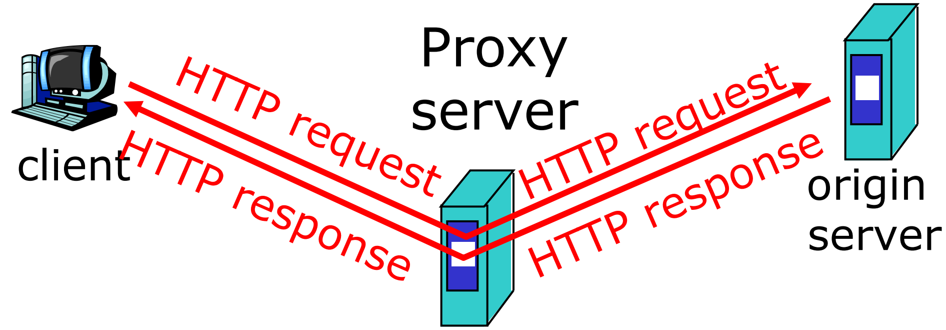
RFC 2616, Ch 13 (25 pages!), 14.9

G. Huston: "Web Caching" *Internet Protocol Journal,* 2(3):2-20

C. Deleuze: "Content Networks" *Internet Protocol Journal,* 7(2):2-11

RFCs aren't always entirely dry, e.g. RFC2616: "If the user has overridden the caching mechanisms in a way that would abnormally reduce the effectiveness of caches, the user agent SHOULD continually indicate this state to the user (for example, by a display of a **picture of currency in flames**)"
"Since the protocol normally allows the user agent to determine if responses are stale or not, this indication need only be displayed when this actually happens. The indication need not be a dialog box; it could be an icon (for example, **a picture of a rotting fish**)"

Proxies



Content (particularly stored, web content) is available from one source, but may also be available from other sources.

Terminology:

- Origin server: Original source of an object
- Proxy server: Provides object instead of origin server

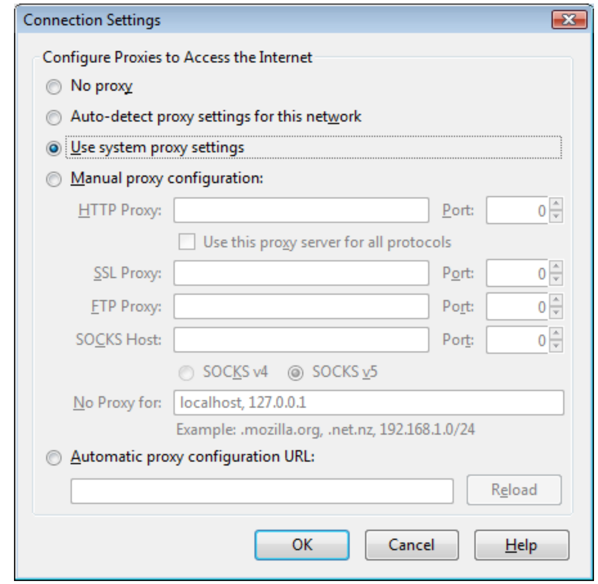
  Some (caches) are demand-driven: Acts as both a
  - server: responding to client's requests
  - client: forwarding requests that it cannot respond to towards the origin server

  Some are driven by supply *and* demand (e.g. proactive caching)

  Some are supply-driven: Content Distribution Networks

Figure based on Kurose & Ross

Client awareness of proxies



- Configured proxy: Client explicitly sends request to proxy, which forwards if necessary.
  - √ Gives client choice of proxy
  - √ May be chosen for reasons other than performance, e.g. privacy through an anonymising proxy
  - × Requires client configuration
- Transparent proxy: Intercepts requests from client and serves them if it can (using origin server's address).
  - √ Requires no configuration of client.
  - √ Service provider can install without coordination with clients.

  => Common today

  ("Transparent" in terms of needing no configuration. May be detectable in other ways, e.g. client sees same TTL on responses from servers varying number of hops away)

1N4*

Caching: Why

- Proxies
- Caches
  - Why
    - Caching principle
    - Caching in computer memories
    - Caching of content in networks
    - Caching example
    - Benefits of caching
    - Costs of caching
    - More on consistency
  - Where
  - How
- Content Distribution Networks

Caching principle

- Fast resources are scarce
  e.g. storage close to clients (low propagation delay & fast links)
- Aim to locate commonly used objects in fast resource, other objects can remain in slower resources.

Determining which objects are commonly accessed:
Accesses are often correlated, and are said to exhibit "locality".
Temporal locality: Information accessed at one *time* is likely to be accessed again in the near future.
Spatial locality: Information accessed at one *point* is likely to be accessed also by nearby points.

"Spatial locality" here refers to demand from nearby points, which differs from "spatial distribution" <DT] which refers to how far the traffic propagates through the network.

Link: Caching in computer memories

Program may repeat a loop several times.

Temporal locality: each instruction in the loop is accessed again in the near future. (c.f. instructions outside the loop).

Spatial locality: after one instruction in the loop is accessed, the next instruction in the loop will likely be accessed.

=> Improve performance by locating recently used and adjacent bytes of complete memory (e.g. DRAM or even disk) in a high-speed memory (e.g. SRAM on CPU).

Caching of content in networks

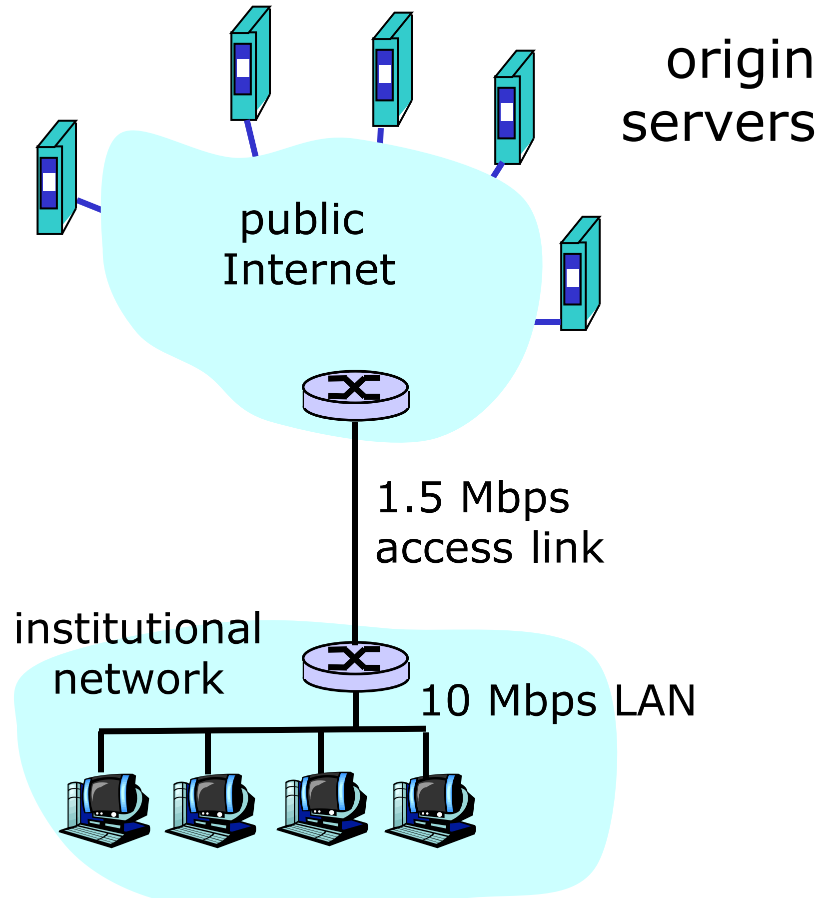
Temporal locality: Content used once is likely to be used again, in the near future, by that node.
- e.g. view UNSW logo one day; likely to view again the next day.

Spatial locality: Content used by one user is likely to also be used by nearby users.
- e.g. one student @ UNSW accesses www.facebook.com => likely that others will also

"In one study spanning more than a month, out of all the objects requested by individual users, on average close to 60 percent of those objects were requested more than once by the same user. … of the hits recorded in another caching study, up to 85 percent were the result of multiple users requesting the same object."

From B. Davison: "A Web caching primer", IEEE Internet Computing, 5(4):38-45. Referring to L. Tauscher and S. Greenberg, "How People Revisit Web Pages: Empirical Findings and Implications for the Design of History Systems," *Int'l J. Human Computer Studies*, 47(1):97-138
B. Duska et al: "The Measured Access Characteristics of World-Wide-Web Client Proxy Caches," *Proc. Usenix Symp. Internet Technologies and Systems (USITS)*, pp. 23-36.

Caching example (1)



Assumptions

- average object size = 100,000 bits
- avg. request rate from institution’s browser to origin servers = 15/sec
- delay from institutional router to any origin server and back to router = 2 sec

Consequences

- utilization on LAN = 15%
- utilization on access link = 100%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + milliseconds

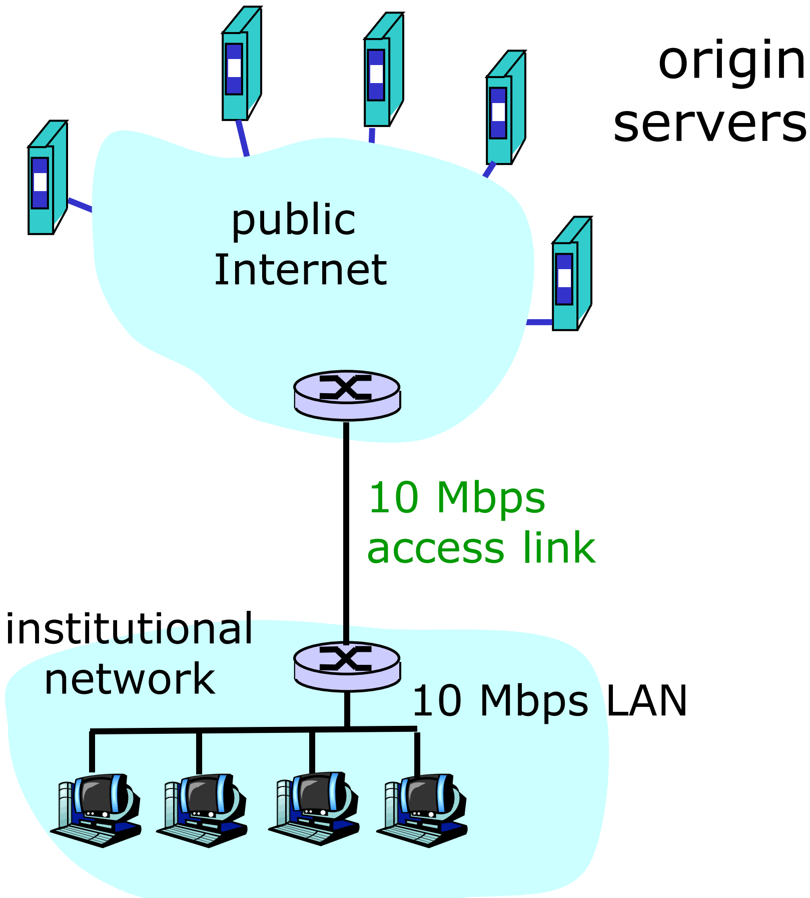
Caching example (2)

<u>Possible solution</u>

- increase bandwidth of access link to, say, 10 Mbps

<u>Consequences</u>

- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay
  = 2 sec + msecs + msecs
- often a costly upgrade



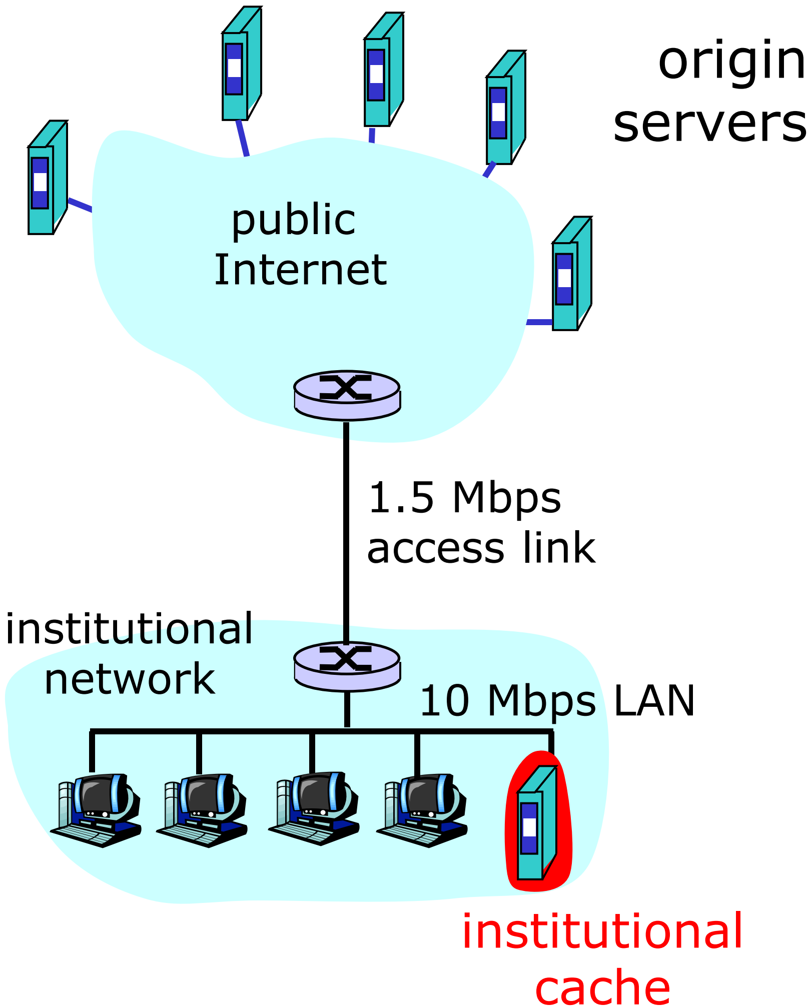
Caching example (3)

Install cache

- suppose hit rate is 0.4

Consequence

- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total delay = Internet delay + access delay + LAN delay
  = 0.6*2 sec + 0.6*0.01 secs + milliseconds† < 1.3 secs



† + millisecond LAN delays for 40% of traffic served by cache

Benefits of caching

Benefits: Eliminate the need (in many cases) to:

- Send request to origin server (reducing delay, and link use)
- Send full response from origin server (reducing link use)

Consequences:

√ Reduced delay
  - √ Directly benefits end-user.
  - √ May benefit service providers (ISPs or web servers) by making their service more popular to end-users.

√ Reduced traffic
  - √ Reduces load on network links
  - √ Reduces load on server
    e.g. reducing “flash crowds”

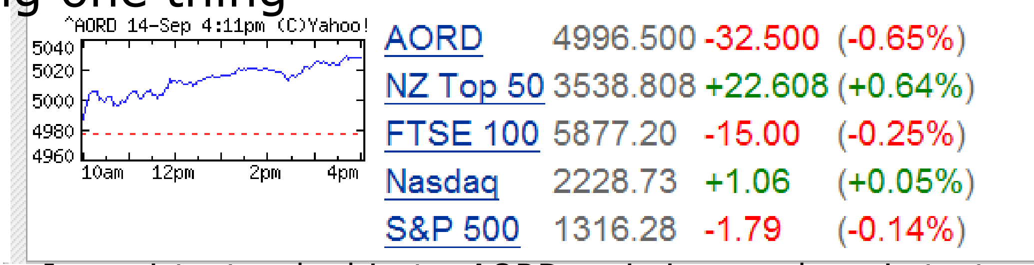
√ Mask unavailability of origin server
  e.g. when working offline, or during faults

Costs of caching

- Implementation cost: memory, processing, protocol for cache control directives
- Either
  - Client needs to be configured, or
  - Switch/router needs to forward requests to cache.
- Delay: for objects that
  - aren't cached: Must check cache *then* get from origin.
  - are cached: May want to "validate" them before using.
- Origin server loses control
  - Consistency: How to update replicas?
  - Monitoring: How track access to content? e.g. count impressions for charging advertisers

More on consistency

Consistency†: Ensure that multiple objects are in agreement
- Object in origin server vs object in cache
- Multiple objects describing one thing



Inconsistent web objects: AORD up in image, down in text.

If object at origin changes,
when/how do cached copies change?
- Validation: Cached copy used only after checking
- Invalidation: Origin may suggest expected expiry time. Origin can't readily inform caches of change:
  - × Hard when origin can't control copying => identify caches
  - × Scales poorly

Degrees of consistency:
- Strong: Never deliver inconsistent info
- Weak: Rarely deliver inconsistent info

† Called "semantic transparency" in RFC2616 about web caching.

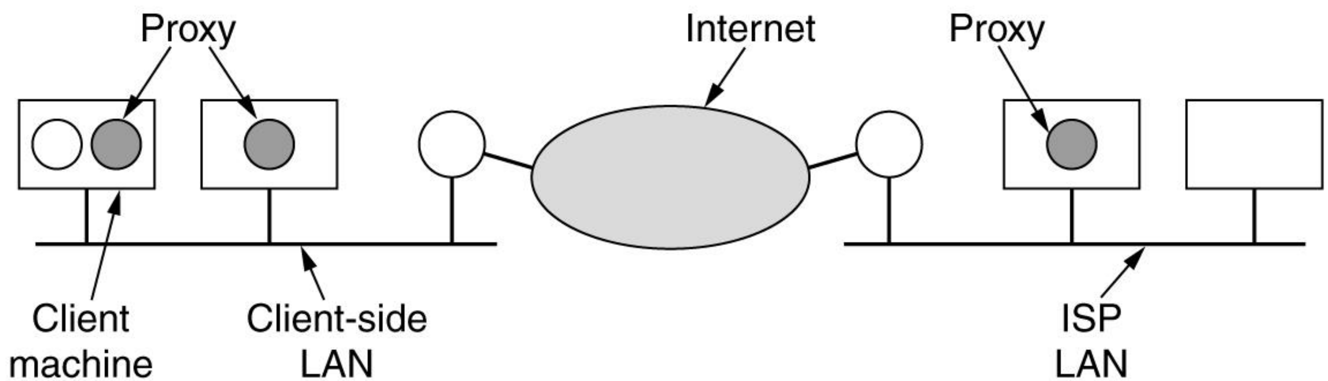
Outline

- Proxies
- Caches
  - Why
  - Where
    - Cache locations
    - Cache implementations
    - Cooperative caching
    - Client awareness of caches
  - How
- Content Distribution Networks

Cache locations

In client end-system

- Implemented in software
- Can cache private info



In network as a proxy cache

- Implemented in "hardware", e.g. router module or free-standing "appliance"
- Shared by multiple clients => can exploit "spatial" locality, but can't cache private info
- Client may be served by multiple proxies in different networks => "hierarchical caching"

 Tanenbaum Fig. 7-45

Cache implementations

When router classifies packets, identifies those suitable for cache (e.g. HTTP requests) and forwards them to cache rather than origin server.

Examples of Cisco Content Engines

- "Network modules" for 26-- 28-- 37-- 3800 routers

- Free-standing "appliances"

Images from http://www.cisco.com/en/US/prod/collateral/routers/ps274/images/09186a008061225c_guest-Cisco_2600@3600@3700_Series_Content_Engine_Modules-us-Product_Data_Sheet-en_2-1.jpg and http://www.cisco.com/en/US/prod/contnetw/ps5680/ps6474/prod_large_photo0900aecd8032afde.jpg

Cooperative caching

- If one cache doesn't have a usable copy, then rather than ask the origin server, perhaps try peer caches
- Cooperation effectively increases capacity of cache and user population (=> potential for spatial locality)

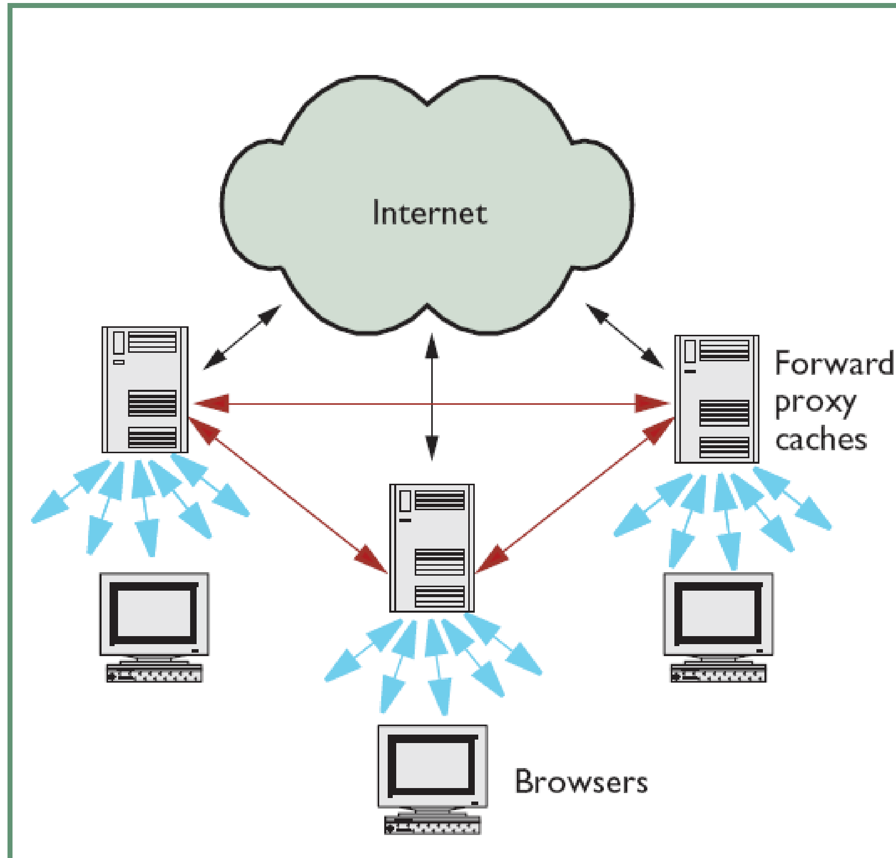


Figure 5. Cooperative caching. Caches communicate with peers before making requests over the Web.

From B. Davison: "[A Web caching primer]", IEEE Internet Computing, 5(4):38-45

Outline

- Proxies
- Caches
  - o Why
  - o Where
  - o How
    - Options for conveying directives
      - o HTML vs HTTP directives
      - o Sample HTTP header
    - Gross directives
    - Predictors
    - Validation
    - Practical issues
- Content Distribution Networks
- How to cache

HTML vs HTTP directives

Directives allow client/server to control how caching is used.

- HTML directives:
  e.g. `<meta http-equiv="pragma" content="no-cache">`
  HTML is close to users (e.g. author of web page) =>
  √ easy for authors to control
  × limited to HTML objects
- HTTP directives:
  e.g. `Cache-control:`, `Expires:`, `if-modified-since:`, header lines
  HTTP is lower level =>
  √ Easier for caches to locate =>
  √ More likely to be obeyed by caches
  √ Most popular directives => our focus
  Web server configuration determines HTTP directives used (e.g. in httpd.conf and .htaccess files)

Sample HTTP header

Request

Hypertext Transfer Protocol
    GET /tele9751/ HTTP/1.1\r\n
    Host: subjects.ee.unsw.edu.au\
    User-Agent: Mozilla/5.0 (Windo
    Accept: text/xml,application/xm
    Accept-Language: en-gb,en;q=
    Accept-Encoding: gzip,deflate\r
    Accept-Charset: ISO-8859-1,utf
    Keep-Alive: 300\r\n
    Connection: keep-alive\r\n
    \r\n

Response

Hypertext Transfer Protocol
    HTTP/1.1 200 OK\r\n
    Date: Mon, 25 Sep 2013 03:21:15 GMT\r\n
    Server: Apache/1.3.33 (Unix) mod_ssl/2.8.23 Op
    Last-Modified: Tue, 19 Sep 2013 21:02:33 GMT\r
    ETag: "28002d-730-45105ae9"\r\n
    Accept-Ranges: bytes\r\n
    Content-Length: 1840\r\n
    Keep-Alive: timeout=15, max=99\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html\r\n
    \r\n
Line-based text data: text/html
    <html>

    <head>
    <title>TELE 9751</title>
    </head>

    <body>
    <center><h1>TELE 9751: Switching Systems De

Outline of HTTP directives

Gross directives: Over-ride other controls

Predictors: Client can avoid another request and response if server previously predicted info would remain current.

- Lifecycle of an object
- Timeliness directives
- Expiry predicted by server

Validation: Client might avoid full response from server if it validates its copy.

- Versions of an object
- Validation by clients
- (re)validation directives
- Choosing whether to revalidate

Practical issues

- Replacement policies for caches with limited capacity
- Shared cache issues

e.g. accessing irtf.org

- Links to other sites (e.g. Facebook) that have been cached.
- Text examples omit irrelevant header fields
- Captured packets on course web page
- All traffic uses server port 80; examples identified by *client* port

Gross directives

Gross directives over-ride other cache controls, e.g. preventing or requiring use of cache.

In requests and responses:

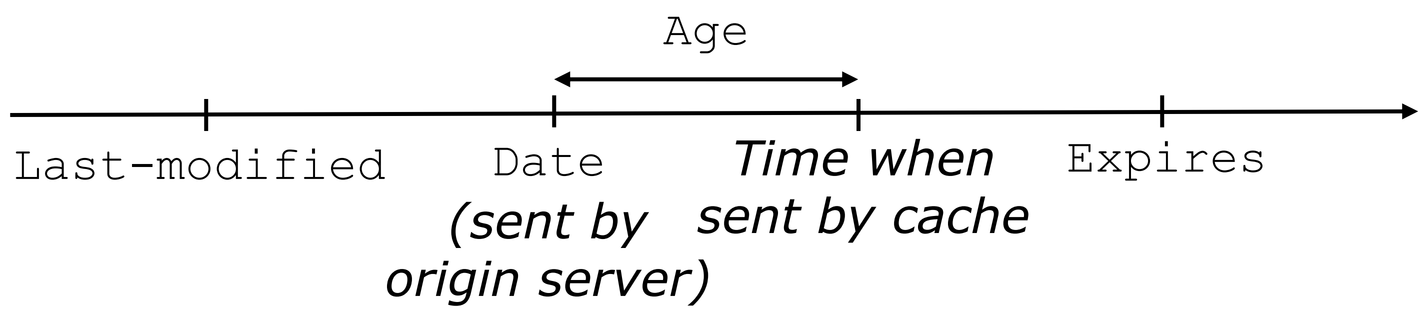
- **`no-store`**: prevents caching[†]: cache must not store (=> can't cache) either associated request or response.
  - intended "to prevent the inadvertent release or retention of sensitive information (for example, on backup tapes)."
- **`no-transform`**:
  - Context: Caches are generally permitted to convert the format of data, e.g. bitmap to jpeg for efficient storage.
  - this directive prevents that, e.g. for medical imaging.

In requests only:

- **`only-if-cached`**: Requires that cache be used. Cache responds with either cached response or 504 (Gateway Timeout) status.
  - Useful when origin server is unavailable & client does not want to wait for cache to timeout trying to validate object.

† A separate "`no-cache`" directive allows a cache to store an object, but requires validation before responding with that object.

Lifecycle of an object



Last-modified: When the object was last modified at the origin server.

- < `Date` of client's copy => OK to use copy
- `Date` – `Last-modified` suggests frequency of change

Date: When the object was sent by the origin server =>

- last time known to be fresh.
- + `Age` = Reference for checking for expiry

Expires: Server's prediction of when copies should be replaced.

Age: How long the object has spent in caches

Lifecycle examples

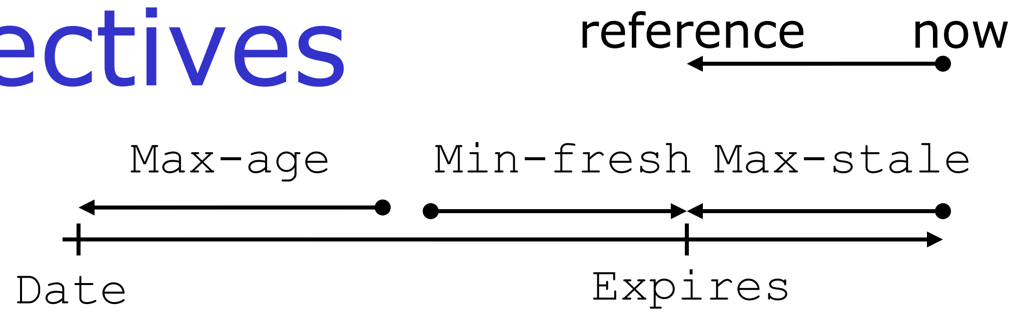
TCP port 49777

Icon showing how many have tweeted this page

GET /1/urls/count.json?url=http%3A%2F%2Firtf.org%2F&callback=twttr.receiveCount HTTP/1.1
Host: urls.api.twitter.com

HTTP/1.1 200 OK
Last-Modified: Tue, 24 May 2011 04:40:21 GMT
Cache-Control: must-revalidate, max-age=900
Expires: Tue, 24 May 2011 04:**55**:21 GMT
Date: Tue, 24 May 2011 04:40:21 GMT

Timeliness directives



Clients can request objects have certain timeliness:

- **Max-age**: Client may be unwilling to accept older info.
  - Note that this is wrt `Date` and not `Last-modified`, i.e. age measures time held in caches; not time since modified by server.
  - `Max-age` can also be carried in responses, limiting age in relative terms (e.g. seconds) c.f. absolute `Expires` date.
- **Min-fresh**: Client may seek to reuse object for some time; seeks reassurance that it won't expire before then. Prefers fresh copy from origin server to older copy from proxy cache.
- **Max-stale**: Client might accept info that is a bit stale. (By default, caches don't return stale info)

1MP*

Expiry predicted by server

Server can limit the lifetime of objects that it sends:

- **`Expires`**: When object becomes stale, in absolute terms.
  - Client/cache can determine relative time through `Expires` - `Date`
  - Useful for objects whose content is affected by calendar, e.g. update copyright dates each year.
- **`Cache-control: max-age`**:† Directly specifies maximum cache longevity (relative time)
  - Useful for specifying an interval (e.g. 1 day), without having to calculate absolute date.

e.g. set `max-age` high for corporate logo image, but low for news index page.

If server doesn't modify object before expiry time, then client can be sure of strong consistency.
But, expiry is a *prediction* by the server, and may be wrong -> "validation"

† There is also an `s-maxage` directive that overrides `max-age` for objects stored in ***s***hared caches.

Expiry examples

TCP port 49771

Facebook link to users who “Like” IRTF

GET /en_US/all.js HTTP/1.1
Host: connect.facebook.net
Referer: http://irtf.org/
If-None-Match: "8d8820ae13b1e6c701a6f8a11096eff7"

HTTP/1.1 200 OK
ETag: "0ee1df6dc294855b5cece50388a69773“
Cache-Control: public, max-age=777
Expires: Tue, 24 May 2011 04:53:18 GMT
Date: Tue, 24 May 2011 04:40:21 GMT

Versions of an object

Versions of an object can be identified by:

- `Last-modified` time:
  - × Requires origin server to run a clock
  - × One-second resolution

- "Entity† tags" (`Etag:`):
  - o Strings that distinguish different versions of an entity.
  - o e.g. created by hashing content of entity.
  - o e.g. ETag: "28002d-730-45105ae9"

† Entity is equivalent to "object" for our purposes.

Validation by clients

Validation = Check whether a copy is still usable.

Usually client seeks a usable copy if its copy isn't => "Conditional get":

1. Client sends GET request with
   - "`if-modified-since:` *Date of cached copy*"†, or
   - "`if-none-match:` *Etag of cached copy*"
2. Server responds with
   - object (if modified), or
   - response code "`HTTP/1.0 304 Not Modified`"

Validation takes time. Client must choose between delay and degree (strong/weak) of consistency.

† There is also an `If-Unmodified-Since` header field, used to continue accessing new ranges of an object that has previously been partially accessed.

Version & validation example

TCP port 49865

GET /img/ietf.png HTTP/1.1
Host: irtf.org
Referer: http://irtf.org/
If-Modified-Since: Fri, 06 May 2011 10:01:43 GMT
If-None-Match: "aa0b06-754-4a29893aa8fc0"
Cache-Control: max-age=0

HTTP/1.1 304 Not Modified
Date: Tue, 24 May 2011 05:21:29 GMT
ETag: "aa0b06-754-4a29893aa8fc0"
Expires: Tue, 31 May 2011 05:21:29 GMT
Cache-Control: max-age=604800

(re)validation directives

Server can force cache to revalidate:

- **`Cache-control: no-cache`**: cache must *never* reuse response (to satisfy a subsequent request) without revalidating it (with the origin server).
  - e.g. search results, in which URL appears the same but proper response depends on request.
- **`Cache-control: must-revalidate`**: must revalidate *if object appears stale*; irrespective of cache or client's acceptance of stale info

Another directive, `proxy-revalidate:` is like `must-revalidate:`, but only applies to shared caches (proxies).

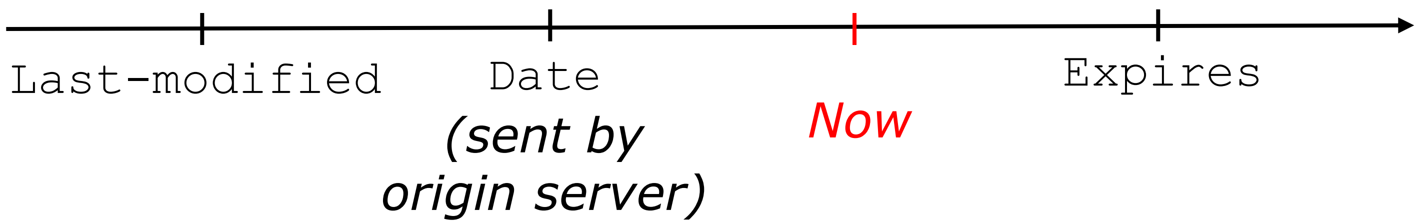
(Re)validation examples

TCP port 49775

GET
/plugins/like.php?channel_url=http%3A%2F%2Fstatic.ak.fbcdn.net%2Fconnect%2Fxd_proxy.php%3Fversion%3D2%23cb%3Df2e9ef73a74c7f8%26origin%3Dhttp%253A%252F%252Firtf.org
... HTTP/1.1
Host: www.facebook.com
Referer: http://irtf.org/

HTTP/1.1 200 OK
Cache-Control: private, no-cache, no-store, must-revalidate
Expires: Sat, 01 Jan 2000 00:00:00 GMT
Pragma: no-cache
Date: Tue, 24 May 2011 04:40:21 GMT

Choosing whether to revalidate



- Strong consistency requires revalidation for each use.
  - √ Cache can reduce traffic load (may not need to download)
  - × Propagation delays remain.
- Weak consistency provided by revalidating only when delay seems warranted by heuristics, e.g.
  - Revalidate if *Now* - `Date` >(`Date` - `Last-modified`)/2
  - Same heuristics suggest which cached objects can be replaced when cache is full.

Replacement policies for full cache

Cache capacity is limited. What should be replaced if cache is full?

Shouldn't cache dynamic content, since a past response will unlikely satisfy a new request.

How to identify dynamic content?

- Responses to POST requests
- Requests that include queries (indicated by "?" in URLs)
- URL suggests generated by a script (e.g. .php)
- Heuristics, e.g. `Date – Last-modified` < threshold

Cache objects that:

- Are likely to be useful in the future:
  - **demand side**: replace Least Recently Used object
  - **supply side**: keep objects that heuristics suggest are most likely to remain valid, e.g. `Expires` >> current time
- Have high cost to refetch
  - **Monetary**, e.g. UNSW used to cache traffic received through AARnet (pay-per-use) but not Grangenet (free, but now gone)
  - **Performance**, e.g. objects that took long to download from origin due to high delay or low throughput

Shared† cache issues

Some content may not be appropriate for other users that share a cache:

- *Personalised content*, e.g. depends on cookies supplied by client.
- *Private content*, e.g. responses to requests that included `Authorization` header

Server can set `cache-control:` directives:

- `private`: prevents sharing of an object
- `public`: overrides default treatment of private content

Directives that specifically control shared caches
(overriding directives that also apply to private caches)

- `s-maxage`: like `max-age`
- `proxy`†`-revalidate`: like `must-revalidate`: Shared cache can respond with cached object after revalidating it.

† "Shared caches" aka "public" / "proxy" caches

Sharing examples

See previous examples:
TCP port 49775: Private object
TCP port 49771: Public object

TCP port 49776 (Small GIF for Google Analytics):
GET /__utm.gif?utmwv=4.9.4&utms=1& ...
&utmdt=Internet%20Research%20Task%20Force%20(IRTF) ...
HTTP/1.1
Host: www.google-analytics.com
Referer: http://irtf.org/

HTTP/1.1 200 OK
Date: Wed, 18 May 2011 14:01:14 GMT
Pragma: no-cache
Expires: Wed, 19 Apr 2000 11:43:00 GMT
(Already expired => won't even be cached, let alone shared!)
Last-Modified: Wed, 21 Jan 2004 19:51:30 GMT
Cache-Control: private, no-cache, no-cache=Set-Cookie, proxy-revalidate

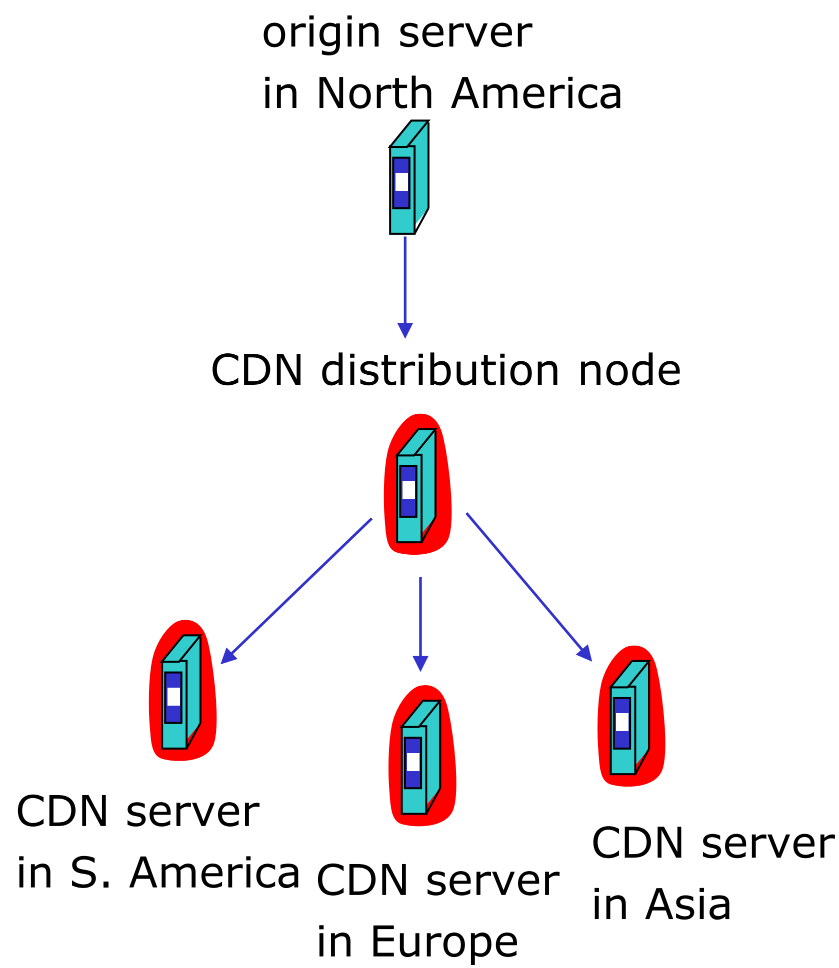
Outline

Content distribution networks (CDNs)†

The content providers are the CDN customers.

Content replication

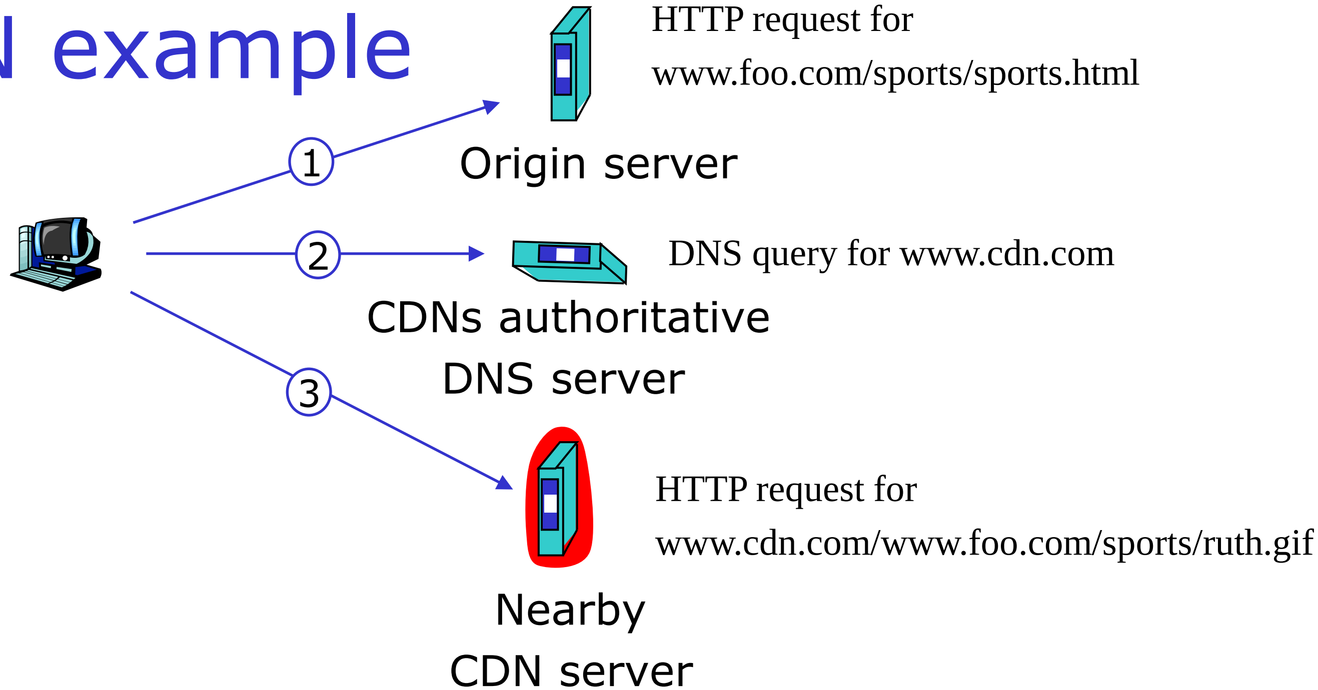
- CDN company installs hundreds of CDN servers throughout Internet
  - in lower-tier ISPs, close to users
- CDN replicates its customers' content in CDN servers. When provider updates content, CDN updates servers



Slide from Kurose and Ross

† aka Content *Delivery* Networks

CDN example



origin server

- www.foo.com
- distributes HTML
- Replaces:
  http://www.foo.com/sports.ruth.gif
  with
  http://www.cdn.com/www.foo.com/sports/ruth.gif

CDN company

- cdn.com
- distributes gif files
- uses its authoritative DNS server to route redirect requests

Slide from Kurose and Ross

[Server doesn't actually modify URL. Instead, domain name of their server points to CDN server - Tim]

More about CDNs

routing requests

- CDN creates a “map”, indicating distances from leaf ISPs and CDN nodes
- when query arrives at authoritative DNS server:
  - server determines ISP from which query originates
  - uses “map” to determine best CDN server

not just Web pages

- streaming stored audio/video
- streaming real-time audio/video
  - CDN nodes create application-layer overlay network

 Slide from Kurose and Ross

Caching/CDN conclusion

While switches historically just moved information across the network, they are becoming increasingly involved in supplying content and (not covered in this lecture) processing to provide application services in a distributed manner.

Things to think about

- Critical thinking: How might increasing end-to-end encryption (e.g. HTTPS) affect the usefulness of caches/CDNs?
- Engineering methods: Optimising for the common case is common across engineering, and exemplified by choosing what to cache.
- Links to other areas: We've seen caching elsewhere:
  - Packet classifiers: hybrids <12U], tries in cached RAM <1F0]
  - Bridge learning <DX] <PG]
- Independent learning: Read about Network Function Virtualisation which allows ISPs/carriers to use generic hardware/cloud-services to provide caching/CDNs, rather than specialised devices

The end!

- Please complete the **myExperience** Evaluation
  - We already know that Tim talks too fast
- **Final exam** covers topics from week 7 (packet classification) onwards
- **Consultation times** will be posted on the course web page
- **Good luck** with your careers as telecommunication engineers!

“You're still here? It's over. Go home. Go”