

18-345 – Fall 08

Lecture 23:

The Web

Peter Steenkiste

Electrical & Computer ENGINEERING

1

Outline

- HTTP review and details
- Persistent HTTP review
- HTTP caching
- Content distribution networks
- Cookies

Electrical & Computer ENGINEERING

2

HTTP Basics

- HTTP layered over bidirectional byte stream
 - Almost always TCP
- Interaction
 - Client sends request to server, followed by response from server to client
 - Requests/responses are encoded in text
- Stateless
 - Server maintains no information about past client requests

Electrical & Computer ENGINEERING

3

HTTP Request

- Request line
 - Method
 - GET – return URI
 - HEAD – return headers only of GET response
 - POST – send data to the server (forms, etc.)
 - URL
 - HTTP version
- Request headers
 - Authorization – authentication info
 - Acceptable document types/encodings
 - From – user email
 - If-Modified-Since
 - Referrer – what caused this page to be requested
 - User-Agent – client software
- Blank-line + Body

Electrical & Computer ENGINEERING

4

HTTP Request

method sp URL sp version cr lf request line

header field name : value cr lf } header lines

header field name : value cr lf

cr lf

Entity Body

Electrical & Computer ENGINEERING

5

HTTP Request Example

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5;
Windows NT 5.0)
Host: www.intel-iris.net
Connection: Keep-Alive
```

Electrical & Computer ENGINEERING

6

HTTP Response

- Status-line
 - HTTP version
 - 3 digit response code
 - 1XX – informational
 - 2XX – success
 - 200 OK
 - 3XX – redirection
 - 301 Moved Permanently
 - 303 Moved Temporarily
 - 304 Not Modified
 - 4XX – client error
 - 404 Not Found
 - 5XX – server error
 - 505 HTTP Version Not Supported
 - Reason phrase
- Headers
 - Location – for redirection
 - Server – server software
 - WWW-Authenticate – request for authentication
 - Allow – list of methods supported (get, head, etc)
 - Content-Encoding – E.g x-gzip
 - Content-Length
 - Content-Type
 - Expires
 - Last-Modified
- Blank-line + Body

7

HTTP Response Example

```

HTTP/1.1 200 OK
Date: Tue, 27 Mar 2001 03:49:38 GMT
Server: Apache/1.3.14 (Unix) (Red-Hat/Linux) mod_ssl/2.7.1
      OpenSSL/0.9.5a DAV/1.0.2 PHP/4.0.1pl2 mod_perl/1.24
Last-Modified: Mon, 29 Jan 2001 17:54:18 GMT
ETag: "7a11f-10ed-3a75ae4a"
Accept-Ranges: bytes
Content-Length: 4333
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
.....
  
```

8

Outline

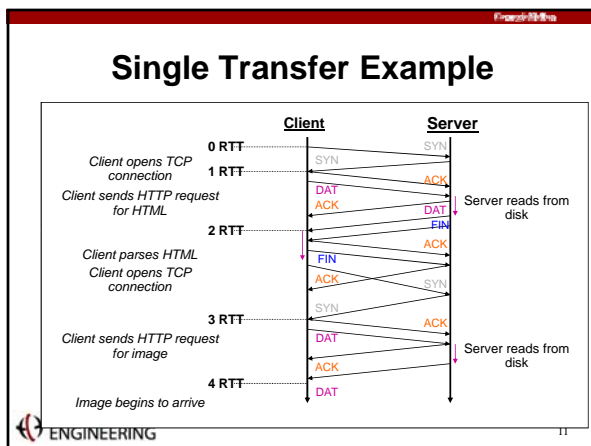
- HTTP intro and details
- **Persistent HTTP**
- HTTP caching
- Content distribution networks
- Cookies

9

HTTP 0.9/1.0

- One request/response per TCP connection
 - Simple to implement
- But: multiple (typically small) objects per page
- Many disadvantages
 - Multiple connection setups → three-way handshake each time
 - Several extra round trips added to transfer
 - Increases server state/processing
 - Short transfers are hard on TCP
 - Stuck in multiple slow start
 - Loss recovery is poor when windows are small

10

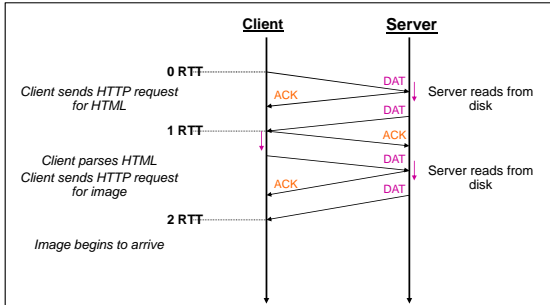


Persistent Connection Solution

- Multiplex multiple transfers onto one TCP connection
- How to identify requests/responses
 - Delimiter → Server must examine response for delimiter string
 - Content-length and delimiter → Must know size of transfer in advance
 - Block-based transmission → send in multiple length delimited blocks
 - Store-and-forward → wait for entire response and then use content-length
 - **Solution** → use existing methods and close connection otherwise

12

Persistent Connection Example



Persistent HTTP

Nonpersistent HTTP issues:

- Requires 2 RTTs per object
- OS must work and allocate host resources for each TCP connection
- But 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 are sent over connection

Persistent without pipelining:

- Client issues new request only when previous response has been received
- One RTT for each referenced object

Persistent with pipelining:

- Default in HTTP/1.1
- Client sends requests as soon as it encounters a referenced object
- As little as one RTT for all the referenced objects

Outline

- HTTP intro and details
- Persistent HTTP
- **HTTP caching**
- Content distribution networks
- Cookies

HTTP Caching

- Clients often cache documents
 - Challenge: update of documents
 - If-Modified-Since requests to check
 - HTTP 0.9/1.0 used just date
 - HTTP 1.1 has an opaque "entity tag" (could be a file signature, etc.) as well
- When/how often should the original be checked for changes?
 - Check every time?
 - Check each session? Day? Etc?
 - Use Expires header
 - If no Expires, often use Last-Modified as estimate
- But: many objects are not cacheable
 - Dynamic data, cgi scripts with parameters, etc.

Example Cache Check Request

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
If-Modified-Since: Mon, 29 Jan 2001 17:54:18 GMT
If-None-Match: "7a11f-10ed-3a75ae4a"
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Host: www.intel-iris.net
Connection: Keep-Alive
```

Example Cache Check Response

```
HTTP/1.1 304 Not Modified
Date: Tue, 27 Mar 2001 03:50:51 GMT
Server: Apache/1.3.14 (Unix) (Red-Hat/Linux)
mod_ssl/2.7.1 OpenSSL/0.9.5a DAV/1.0.2
PHP/4.0.1pl2 mod_perl/1.24
Connection: Keep-Alive
Keep-Alive: timeout=15, max=100
ETag: "7a11f-10ed-3a75ae4a"
```

Web Proxy Caches

- User configures 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

19

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 link delay + LAN delay = 2 sec + minutes + milliseconds

20

Caching Example (2)

Possible solution

- Increase bandwidth of access link to, say, 10 Mbps
- Often a costly upgrade

Consequences

- Utilization on LAN = 15%
- Utilization on access link = 15%
- Total delay = Internet delay + access link delay + LAN delay = 2 sec + msec + msec

21

Caching Example (3)

Install cache

- Suppose hit rate is 40%

Consequence

- 40% requests will be satisfied almost immediately (say 10 msec)
- 60% requests satisfied by origin server
- Utilization of access link reduced to 60%, resulting in negligible delays
- Weighted average of delays = $.4 \cdot 2 \text{ sec} + .6 \cdot 10 \text{ msec} < 1.3 \text{ secs}$

22

Content Distribution Networks (CDNs)

- The content providers are the CDN customers.

Content replication

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

23

Outline

- HTTP intro and details
- Persistent HTTP
- HTTP caching
- Content distribution networks**
- Cookies

24

Content Distribution Networks & Server Selection

- Replicate content on many servers
- Challenges
 - How to replicate content
 - Where to replicate content
 - How to find replicated content
 - How to choose among know replicas
 - How to direct clients towards replica

Server Selection

- Many metrics for server selection
 - Lowest load → to balance load on servers
 - Best performance → to improve client performance
 - Based on Geography? RTT? Throughput? Load?
 - Any alive node → to provide fault tolerance
- How to direct clients to a particular server?
 - As part of application → HTTP redirect
 - As part of naming → DNS

Application Based

- HTTP supports simple way to indicate that Web page has moved (30X responses)
- Server receives Get request from client
 - Decides which server is best suited for particular client and object
 - Returns HTTP redirect to that server
- Can make informed application-specific decision
 - Has access to all information in the GET request
- May introduce additional overhead → multiple connection setup, name lookups, etc.
 - HTTP redirect also has some design flaws

Naming Based

- Client does name lookup for service
- Name server chooses appropriate server address
 - A-record returned is “best” one for the client
- What information can name server base decision on?
 - Server load/location → must be collected
 - Information in the name lookup request
 - Name service client → typically the local name server for client

How Akamai Works

- Clients fetch html document from primary server
 - E.g. fetch index.html from cnn.com
- URLs for replicated content are replaced in html
 - E.g. `` replaced with ``
- Client is forced to resolve `aXYZ.g.akamaitech.net` hostname

How Akamai Works

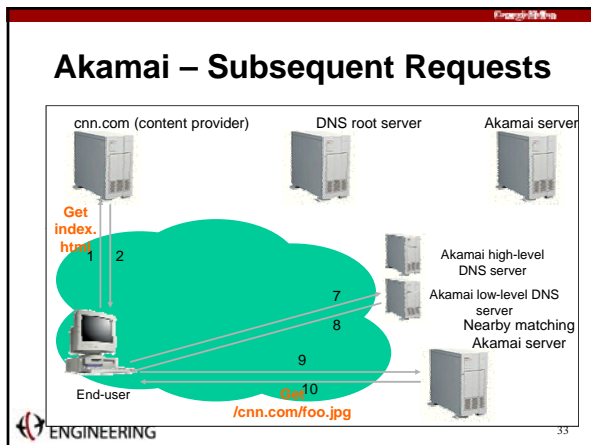
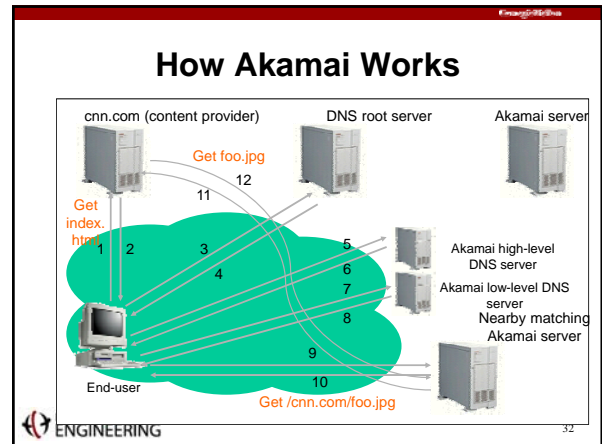
- How is content replicated?
- Akamai only replicates static content (*)
- Modified name contains original file name
- Akamai server is asked for content
 - First checks local cache
 - If not in cache, requests file from primary server and caches file

* (At least, the version we're talking about today. Akamai actually lets sites write code that can run on Akamai's servers, but that's a pretty different beast)

How Akamai Works

- Root server gives NS record for akamai.net
- Akamai.net name server returns NS record for g.akamaitech.net
 - Name server chosen to be in region of client's name server
 - TTL is large
- G.akamaitech.net nameserver chooses server in region
 - Should try to choose server that has file in cache - How to choose?
 - Uses aXYZ name and consistently map to a specific server
 - TTL is small → why?
- Significantly increases the load on DNS

31



Outline

- HTTP intro and details
- Persistent HTTP
- HTTP caching
- Content distribution networks
- Cookies

34

Cookies: Keeping “state”

Many major Web sites use cookies

Four components:

- 1) Cookie header line in the HTTP response message
- 2) Cookie header line in HTTP request message
- 3) Cookie file kept on user's host and managed by user's browser
- 4) Back-end database at Web site

Example:

- Susan access Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID

35

