Web Content Delivery

Outline

- HTTP review
- Persistent HTTP
- HTTP caching
- Proxying and content distribution networks
  - Web proxies
  - Hierarchical networks and Internet Cache Protocol (ICP)
  - Modern distributed CDNs (Akamai)

HTTP Basics (Review)

- 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 info about past client requests

HTTP Request

<table>
<thead>
<tr>
<th>method</th>
<th>sp</th>
<th>URL</th>
<th>sp</th>
<th>version</th>
<th>cr</th>
<th>if</th>
</tr>
</thead>
<tbody>
<tr>
<td>header field name</td>
<td>value</td>
<td>cr</td>
<td>if</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Entity Body
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

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

HTTP Response (cont.)

• 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

How to Mark End of Message?

• Content-Length
  – Must know size of transfer in advance
• Close connection
  – Only server can do this
• Implied length
  – E.g., 304 never have body content
• Transfer-Encoding: chunked (HTTP/1.1)
  – After headers, each chunk is content length in hex,
    CRLF, then body. Final chunk is length 0.
**Example: Chunked Encoding**

HTTP/1.1 200 OK 
Transfer-Encoding: chunked 
25 
This is the data in the first chunk 
1A 
and this is the second one 
0

- Especially useful for dynamically-generated content, as length is not a priori known
  - Server would otherwise need to cache data until done generating, and then go back and fill-in length header before transmitting

---

**Outline**

- HTTP review
- Persistent HTTP
- HTTP caching
- Proxying and content distribution networks
  - Web proxies
  - Hierarchical networks and Internet Cache Protocol (ICP)
  - Modern distributed CDNs (Akamai)

---

**Problems with simple model**

- Multiple connection setups
  - Three-way handshake each time
- Short transfers are hard on TCP
  - Stuck in slow start
  - Loss recovery is poor when windows are small
- Lots of extra connections
  - Increases server state/processing
  - Server forced to keep TIME_WAIT connection state
TCP Interaction: Short Transfers

- Multiple connection setups
  - Three-way handshake each time
- Round-trip time estimation
  - Maybe large at the start of a connection (e.g., 3 seconds)
  - Leads to latency in detecting lost packets
- Congestion window
  - Small value at beginning of connection (e.g., 1 MSS)
  - May not reach a high value before transfer is done
- Detecting packet loss
  - Timeout: slow
  - Duplicate ACK
    - Requires many packets in flight
    - Which doesn’t happen for very short transfers

Persistent HTTP

Non-persistent HTTP issues:
- Requires 2 RTTs per object
- OS must allocate 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 object

Persistent with pipelining:
- Default in HTTP/1.1 spec
- Client sends requests as soon as it encounters referenced object
  - As little as one RTT for all the referenced objects
- Server must handle responses in same order as requests

"Persistent without pipelining" most common

- When does pipelining work best?
  - Small objects, equal time to serve each object
  - Small because pipelining simply removes additional 1 RTT delay to request new content
- Alternative design?
  - Multiple parallel connections (typically 2-4). Also allows parallelism at server
    - Doesn’t have problem of head-of-line blocking like pipelining
      - Dynamic content makes HOL blocking possibility worse
  - In practice, many servers don’t support, and many browsers do not default to pipelining
Outline

• HTTP review
• Persistent HTTP
• HTTP caching
• Proxying and content distribution networks
  – Web proxies
  – Hierarchical networks and Internet Cache Protocol (ICP)
  – Modern distributed CDNs (Akamai)

HTTP Caching

• Clients often cache documents
  – When should origin be checked for changes?
  – Every time? Every session? Date?

• HTTP includes caching information in headers
  – HTTP 0.9/1.0 used: “Expires: <date>;” “Pragma: no-cache”
  – HTTP/1.1 has “Cache-Control”
    • “No-Cache”, “Private”, “Max-age: <seconds>”
    • “E-tag: <opaque value>”

• If not expired, use cached copy
• If expired, use condition GET request to origin
  – “If-Modified-Since: <date>”, “If-None-Match: <etag>”
  – 304 (“Not Modified”) or 200 (“OK”) response

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 (compat; 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, then returns to client

Caching Example (1)

Assumptions
- Average object size = 100K bits
- Avg. request rate from browsers to origin servers = 20/sec
- Delay from institutional router to any origin server and back to router = 2 sec

Consequences
- Utilization on LAN = 20%
- Utilization on access link = 100%
- Total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + milliseconds

Caching Example (2)

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

Consequences
- Utilization on LAN = 20%
- Utilization on access link = 20%
- Total delay = Internet delay + access delay + LAN delay
  = 2 sec + milliseconds

Caching Example (3)

Install Cache
- Support hit rate is 60%

Consequences
- 60% requests satisfied almost immediately (say 10 msec)
- 40% requests satisfied by origin
- Utilization of access link down to 53%, yielding negligible delays
- Weighted average of delays
  = .6*2 s + .4*.010 ms < 1.3 s
When a single cache isn’t enough

- What if the working set is > proxy disk?
  - Cooperation!

- A static hierarchy
  - Check local
  - If miss, check siblings
  - If miss, fetch through parent

- Internet Cache Protocol (ICP)
  - ICPv2 in RFC 2186 (& 2187)
  - UDP-based, short timeout

Problems

- Significant fraction (>50%) of HTTP objects uncachable

- Sources of dynamism?
  - Dynamic data: Stock prices, scores, web cams
  - CGI scripts: results based on passed parameters
  - Cookies: results may be based on passed data
  - SSL: encrypted data is not cacheable
  - Advertising / analytics: owner wants to measure # hits
    - Random strings in content to ensure unique counting
  - But...much dynamic content small, while static content large (images, video, .js, .css, etc.)

Content Distribution Networks (CDNs)

- Content providers are CDN customers

**Content replication**

- CDN company installs thousands of servers throughout Internet
  - In large datacenters
  - Or, close to users
- CDN replicates customers’ content
- When provider updates content, CDN updates servers

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 known replicas
  - How to direct clients towards replica
Server Selection

• Which server?
  – 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 routing: anycast, cluster load balancing
  – As part of application: HTTP redirect
  – As part of naming: DNS

Trade-offs between approaches

• Routing based (IP anycast)
  – Pros:
  – Cons:

• Application based (HTTP redirects)
  – Pros:
  – Cons:

• Naming based (DNS selection)
  – Pros:
  – Cons:
Trade-offs between approaches

• Routing based (IP anycast)
  – Pros: Transparent to clients, works when browsers cache failed addresses, circumvents many routing issues
  – Cons: Little control, complex, scalability, TCP can’t recover, ...

• Application based (HTTP redirects)
  – Pros: Application-level, fine-grained control
  – Cons: Additional load and RTTs, hard to cache

• Naming based (DNS selection)
  – Pros: Well-suited for caching, reduce RTTs
  – Cons: Request by resolver not client, request for domain not URL, hidden load factor of resolver’s population
    • Much of this data can be estimated “over time”

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. `<img src="http://cnn.com/af/x.gif">` replaced with `<img src="a73.g.akamai.net/7/23/cnn.com/af/x.gif>`
  – Or, cache.cnn.com, and CNN adds CNAME (alias) for cache.cnn.com → a73.g.akamai.net

• Client resolves aXYZ.g.akamaitech.net hostname
  – Maps to a server in one of Akamai’s clusters

Outline

• HTTP review

• Persistent HTTP

• HTTP caching

• Proxying and content distribution networks
  – Web proxies
  – Hierarchical networks and Internet Cache Protocol (ICP)
  – Modern distributed CDNs (Akamai)

How Akamai Works

• Akamai only replicates static content
  – At least, simple version. Akamai also lets sites write code that run on their servers, but that’s a pretty different beast

• Modified name contains original file name

• Akamai server is asked for content
  1. Checks local cache
  2. Check other servers in local cluster (via ICP)
  3. Otherwise, request from primary server and cache file

• CDN is a large-scale, distributed network
  – Akamai has ~25K servers spread over ~1K clusters world-wide
  – Why do you want servers in many different locations?
  – Why might video distribution architectures be different?
How Akamai Works

- Root server gives NS record for akamai.net
- This nameserver returns NS record for g.akamai.net
  - Nameserver chosen to be in region of client’s name server
  - TTL is large
- g.akamai.net nameserver chooses server in region
  - Should try to chose server that has file in cache (How?)
  - Uses aXYZ name and hash
  - TTL is small (Why?)
  - Small modification to before: (Why?)
    - CNAME cache.cnn.com → cache.cnn.com.akamaidns.net
    - CNAME cache.cnn.com.akamaidns.net → a73.g.akamai.net

How Akamai Works – Already Cached