Content Delivery on the Web: HTTP and CDNs

Brad Karp
UCL Computer Science
(slides contributed by Kyle Jamieson)

CS 3035/GZ01
9th December 2014
Outline

• The Web: HTTP and caching
  – The Hypertext Transport Protocol: HTTP
  – HTTP performance
    • Persistent and concurrent connections; pipelining
  – Web caching

• Content distribution networks (CDNs)
The World Wide Web

• **1990**: Tim Berners-Lee, British physicist working at CERN, trying to solve a real-world problem, remote access to scientific data

• “Can't we convert every information system so that it looks like part of some imaginary information system which everyone can read?”

• The *World Wide Web* (WWW): a distributed database of “pages” linked through the application-layer *Hypertext Transport Protocol* (HTTP)
Hypertext Markup Language (HTML)

• A Web page has several components
  – Base HTML file
  – Referenced objects (e.g., images)

• Hypertext Markup Language (HTML)
  – Representation of hypertext documents in ASCII format
  – Web browsers interpret HTML when rendering a page
  – Several functions:
    • Format text, reference images, embed hyperlinks (HREF)

• Straightforward to learn
  – Syntax easy to understand
  – Authoring programs can auto-generate HTML
  – Source almost always available
Hypertext Transfer Protocol (HTTP)

- **Client-server, application-layer** protocol for transferring information on the web

- Important properties:
  - Request-response protocol
  - Reliance on a global URI namespace
  - Resource *metadata* plays a key role
  - *Stateless*

- ASCII wire format:
  
  ```
  % telnet www.ucl.ac.uk 80
  GET /index.html HTTP/0.9
  <CRLF>
  ```
Steps in an HTTP request

1. HTTP client initiates TCP connection to server
2. HTTP client sends HTTP request to server
3. HTTP server processes and responds to request
4. HTTP client receives the request and parses response
5. TCP connection terminates

- How many RTTs for a client to complete a request?
Client-to-server communication

- **HTTP request message**
  - *Request line*: method, resource name, and protocol version
  - Request *methods* include:
    - **GET**: Return current value of resource, run program, ...
    - **HEAD**: Return the metadata associated with a resource
    - **POST**: Update resource, provide input to a program, ...
  - *Header lines*: provide information or modify request
  - *Body*: optional data (e.g., to “POST” data to the server)

**Request line:**
```
GET /index.html HTTP/1.1
Host: www.ucl.ac.uk
```

**Header lines:**
```
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
```

*Carriage return/line feed* (mandatory CR/LF)

indicates end of request message
Server-to-client communication

- **HTTP response message**
  - *Status line*: protocol version, status code, status phrase
  - *Header lines*: provide information
  - *Body*: optional data

**Status line (protocol, status code, status phrase):**

HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 2006 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 2006 ...
Content-Length: 6821
Content-Type: text/html

**Header lines:**

Data (*e.g.* requested HTML file)

(data data data data data data data ... )
Server-to-client communication

- **HTTP response message**
  - *Status line*: protocol version, status code, status phrase
  - *Header lines*: provide information
  - *Body*: optional data

<table>
<thead>
<tr>
<th>Code</th>
<th>Class</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xx</td>
<td>Informational</td>
<td>100 Continue</td>
</tr>
<tr>
<td>2xx</td>
<td>Success</td>
<td>200 OK</td>
</tr>
<tr>
<td>3xx</td>
<td>Redirection</td>
<td>304 Not Modified</td>
</tr>
<tr>
<td>4xx</td>
<td>Client error</td>
<td>404 Not Found</td>
</tr>
<tr>
<td>5xx</td>
<td>Server error</td>
<td>503 Service Unavailable</td>
</tr>
</tbody>
</table>
Web Server: Generating a response

• The web server has several options:
  
  – Return a file ("static content")
    • URL matches a file (e.g., /www/index.html)
    • Server returns the file’s contents as the response
    • Server generates appropriate response header
  
  – Generate a response dynamically
    • URL triggers a program on the server
    • Server runs program and sends output to client
  
  – Return metadata with no body
HTTP resource metadata

- Meta-data
  - Info about a resource, stored as a separate entity

- Examples:
  - Size of a resource, last modification time, etc.
  - Example: Type of the content
    - Data format classification (e.g., Content-Type: text/html)
    - Enables browser to automatically launch an appropriate viewer
    - From e-mail’s Multipurpose Internet Mail Extensions (MIME)

- Usage example: Conditional GET Request
  - Client requests object with “If-modified-since” header
  - If unchanged, server responds “HTTP/1.1 304 Not Modified”
  - No body in the server’s response, only a header
HTTP/1.1 bandwidth optimization

- **Data compression**
  - Image files pre-compressed, but much other content is not
  - Client uses Content-Encoding header in HTTP GET to signal which encodings it can decompress, and which it prefers

- **Range requests**
  - Problem: client wants to resume an incomplete download, view certain pages of a long document
  - Need to retrieve just a small section of a resource
  - Client uses Range header in HTTP GET to specify what portion of the resource to download
HTTP is stateless

- **Stateless protocol**
  - Each request-response exchange is treated independently
  - Servers are not required to retain state in the time between exchanges

- This is good: **Improves scalability** on the server-side
  - Don’t have to retain info across client requests
  - Can handle higher rate of client requests
  - Order of client requests doesn’t matter

- This is also a limitation: **Some apps need persistent state**
  - Need to uniquely identify user or store temporary info
  - *e.g.*, shopping cart, user preferences/profiles, usage tracking, ...
State in a stateless protocol: Cookies

- *Client-side* state maintenance
  - **Client stores** small state on behalf of server, **sends state** in future requests
  - At least two possibilities:
    - All state may be **encoded into cookie** stored at client
    - **Cookie indexes state** in a database on the server

- Used for authentication, customize user experience, provide shopping cart

---

```
HTTP/1.0 200 OK;
Set-cookie: 1678
```

One week later...

```
HTTP GET;
Cookie: 1678
```

```
HTTP/1.0 200 OK
```

```
Outline

• Content compression

• The Web
  – The Hypertext Transport Protocol: HTTP
  – HTTP performance
    • Persistent and concurrent connections; pipelining
    – Web caching

• Content distribution networks (CDNs)
TCP connection establishment delay

• Recall the TCP retransmit timeout: 
  \[ RTO = RTT + 4 \times \text{variance} \]
  – Initial RTO = 3 seconds
  – Double RTO on timeout

• Avoids unnecessary SYN retransmissions if RTT is large

• Frustrated users hit stop button (terminate TCP connections) and reload button
Effect of lowering initial RTO

- Today on the Internet
  - ≈ 2% TCP flows lose first SYN
    - Current proposal: Retransmit SYN after 1 sec
    - **Lowers the penalty for SYN loss from 3 seconds to 1 second**
  - ≈ 2% TCP flows’ RTT > 1 sec
    - Why? 802.11 MAC layer, slow dialup line, slow VPN connection
      - **Spurious SYN retransmission**

- Tradeoff between spurious SYN retransmission and user wait
HTTP/1.0 fetching items: Stop and wait

• Most Web pages have multiple objects (“items”), e.g., HTML file and a number of embedded images

• HTTP/1.0 uses stop and wait at the granularity of objects

• The result:
  – At least two RTTs per object
  – Depending on the size of the responses, each TCP connection may stay in slow start
Delay in the middle of a web transfer

- Now, the TCP connection is open and in slow start
- Recall TCP slow start: increase sender cwnd by a packet size for each received acknowledgement
  - Short HTTP transfers spend most of their time in slow start
- Small cwnd means that after a loss, receiver unlikely to generate three dup ACKs
  - Retransmission timeouts are more likely
HTTP/1.0 fetching items: Received sequence number plot

- Fetch an 8.5 Kbyte page with 10 embedded objects, most < 10 Kbyte
- **All TCP connections stay in slow start, except for the large object**
Improving HTTP Performance: 
Persistent connections

• Maintain one TCP connection across multiple HTTP requests
  – Including transfers subsequent to current page
  – Client or server can tear down connection

• Performance advantages:
  – Avoid overhead of connection set-up and tear-down
  – Allow TCP to learn more accurate RTT estimate
  – Allow TCP congestion window to increase
    • i.e., leverage previously discovered bandwidth

• Keep-Alive mechanism in HTTP/1.0, default in HTTP/1.1

GET /index.html HTTP/1.0
Connection: Keep-Alive
Persistent connections fetching items

- Still stop-and-wait at the granularity of objects, at the application layer
  - At least one RTT per object
Persistent connections avoid unnecessary slow starts

- Fetch an 8.5 Kbyte page with 10 embedded objects, most < 10 Kbyte
- Leave TCP connection open after server response, next HTTP request reuses it
- Only incur one slow start, but takes an RTT to issue next request
Improving HTTP Performance: Concurrent requests and responses

- Use multiple TCP connections *in parallel*
- Does not necessarily response order

- Client = 😊 Why?
- Server = 😊 Why?
- Network = 😞 Why?

**Is this *fair***?
- $N$ parallel connections use bandwidth $N$ times more aggressively than just one
- What’s a reasonable/fair limit as traffic competes with that of other users?
Improving HTTP performance:
Pipelined requests and responses

• **Pipeline** requests and responses
  – **Amortizes the RTT** across multiple objects retrieved
  – Reduce TCP connection overhead
  – Can reduce request overhead, packing multiple requests into one packet
  – **Small items in the response** (common) can also share TCP segments
  – Maintains order of responses
  – Item 1 always arrives before item 2

• **How is this different from concurrent requests/responses?**
Pipelined requests overlap RTTs

- Fetch an 8.5 Kbyte page with 10 embedded objects, most < 10 Kbyte
- Send multiple HTTP requests simultaneously
- **Overlaps RTTs of all requests**
Outline

• The Web: HTTP and Caching
  – The Hypertext Transport Protocol: HTTP
  – HTTP performance
    • Persistent and concurrent connections; pipelining
  – Web caching

• Content distribution networks (CDNs)
Improving HTTP Performance: Web caching

- Many clients transfer same information
  - Generates redundant server and network load
  - Clients experience unnecessary latency
Improving HTTP Performance: Caching: Why

• Motive for placing content closer to client:
  — User gets better response time
  — Content providers get happier users
    • Time is money, really!
  — Network gets reduced load

• Why does caching work?
  — Exploits locality of reference

• How well does caching work?
  — Very well, up to a limit
  — Large overlap in content
  — But many unique requests
Improving HTTP performance: Conditional GET

- **Goal**: Don’t send object if cache has up-to-date cached version
  - Saves server resources!

- **Client**: specifies the date of its cached copy in HTTP request
  - If-modified-since: <date>

- **Server**:
  - Compares If-modified-since date against “last modified” time of desired resource
  - Response contains no data if cached copy is up-to-date

Diagram:

- **Cache**
  - HTTP request
  - If-modified-since: <date>

- **Server**
  - HTTP response
  - HTTP/1.0 304 Not Modified

  - Object not modified

  - HTTP request
  - If-modified-since: <date>

  - HTTP response
  - HTTP/1.0 200 OK
  - <data>

  - Object modified
Improving HTTP Performance: 
Caching with Reverse Proxies

Cache data close to origin server $\rightarrow$ decrease server load
- Typically done by content providers
- Client thinks it is talking to the *origin server* (the server with content)
- *Does not work for dynamic content (e.g. content generated by a script running on the origin server).* Why?

![Diagram of Reverse Proxies and ISPs](image-url)
Improving HTTP Performance:
Caching with Forward Proxies

Cache data close to **clients** → reduce network traffic, decrease latency

- Typically done by ISPs or corporate LANs
- Client is **configured** to send HTTP requests to **forward proxy**
- Reduces traffic on ISP-1’s access link, origin server, and backbone ISP
Improving HTTP Performance: Caching and Replication

• Caching (pull)
  – Replicate content “on demand” after a request
  – Store the response message locally for future use

• Replication (push)
  – Planned replication of content in multiple locations
  – Update of resources handled outside of HTTP
  – Can replicate scripts that create dynamic responses
  – *e.g.*, content distribution networks, which we will discuss later
Outline

• The Web: HTTP and caching

• Content distribution networks (CDNs)
  – Hosting and load balancing
  – Akamai case study
Hosting: Multiple sites per machine

• Multiple web sites on a single machine
  – Hosting company runs the Web server on behalf of multiple sites (e.g., www.foo.com and www.bar.com)

• Problem: GET /index.html

• Solutions:
  1. Multiple server processes on the same machine
     • Have a separate IP address (or port) for each server
  2. Include site name in HTTP request
     • Single Web server process with a single IP address
     • Client includes Host header (e.g., Host: www.foo.com)
     • Required header with HTTP/1.1
Hosting: Multiple machines per site

• Replicate a popular Web site across multiple machines
  – Helps to handle the load
  – Places content closer to clients
  – Helps when content isn’t cacheable by proxies/CDNs

• **Problem:** Want to direct client to a *particular* replica.  *Why?*
  – **Balance load** across server replicas
  – Pair clients with *nearby* servers

• **Solution #1:** Manual selection by clients
  – Each replica has its own site name
  – A Web page lists the replicas (*e.g.*, by name, location), and asks clients to click on a hyperlink to pick
Hosting: Multiple machines per site

• **Solution #2:** Single IP address, multiple machines
  – Run multiple machines behind a single IP address

  – Ensure all packets from a single TCP connection go to the same replica

Load Balancer

64.236.16.20
Hosting: Multiple machines per site

- **Solution #3:** Multiple addresses, multiple machines
  - Same name but different addresses for all of the replicas
  - Configure DNS server to return different addresses

```
Internet

12.1.1.1
173.72.54.131
12.2.3.4
64.236.16.20
```
Content distribution networks

• Challenge ca. 2002: *How to reliably deliver large amounts of content to users worldwide?*
  
  – Flash crowds overwhelm web server, access link, or backend database infrastructure
  
  – More rich content: audio, video

• Possible solutions
  
  – Web caching: *dynamic content and diversity causes low proxy hit rates (25–40%)*
  
  – Multihoming for the server: **BGP takes minutes to converge upon route failure**
Improving HTTP Performance: Caching with Content Distribution Networks

• Integrate forward and reverse caching functionality
  – One overlay network (usually) administered by one entity
  – *e.g.*, Akamai

• Provide document caching
  – **Pull:** Direct result of clients’ requests
  – **Push:** Expectation of high access rate

• Also do some processing
  – Handle *dynamic* web pages
  – *Transcoding*
Content distribution networks

• Replicate content at the edge
  – Deploy thousands of content distribution network (CDN) edge servers at many ISPs
  – Push content to edge servers ahead of time
  – Avoid impairments (loss, delay) of using long paths
  – Solve “flash crowds” problem

• Which CDN server?
  1. Nearest server to client
  2. Likely to have content
  3. Available (load, bandwidth)
  4. Adapt choice quickly (secs.)
Improving HTTP Performance: CDN Example – Akamai

- Akamai creates **new domain names** for each client content provider
  - e.g., a128.g.akamai.net

- Akamai’s DNS servers are authoritative for the new domains

- The client content provider modifies its content so that **the URLs embedded in HTML pages** reference the CDN’s domains
  - e.g.: http://www.cnn.com/image-of-the-day.gif → http://a128.g.akamai.net/image-of-the-day.gif

- Client’s browser will issue HTTP GETs to the CDN instead of the origin server
Mapping requests to CDN servers

- Goal: find the CDN server **nearest to the client**
  - Establish BGP peering sessions with Internet border routers → coarse-grained AS map of Internet
  - Combine with live traceroute, loss measurement data between CDN servers
  - **Result: a map of the Internet**

- Finding an **available** CDN server
  - Server health, service requested, server load balancing (CPU, disk, network), and network condition
  - Agents simulate user behavior, downloading objects, and reporting failure rates and service times
DNS-based redirection

• Two levels of DNS indirection

1. Akamai top-level nameservers (TLNSs)
   • Locations: US (4), Europe (4), Asia (1)
   • TLNSs return eight LLNSs in three different regions
     – Chosen to be close to the requesting client (use the map)
     – Handles complete failure of any two particular regions

2. Akamai low-level nameservers (LLNSs)
   • Point to Akamai edge servers, which serve content
   • Do most of the load-balancing
DNS resolution

End user
DNS resolution

End user → Browser cache

1
DNS resolution
DNS resolution

1. End user
2. Browser cache
3. Local name server
DNS resolution

1. End user
2. Browser cache
3. Local name server
4. Generic Top-Level Domain Nameserver

[Slide: Bruce Maggs]
DNS resolution

1. **End user**
2. **Browser cache**
3. **Local name server**
4. **Generic Top-Level Domain Nameserver**
5. **10.10.123.5**

URL: `news.bbc.co.uk`
DNS resolution

1. End user
2. Browser cache
3. OS
4. Generic Top-Level Domain Nameserver
5. 10.10.123.5
6. ak.bbc.co.uk

bbc.co.uk’s nameserver

Local nameserver

news.bbc.co.uk
DNS resolution

1. End user
2. Browser cache
3. OS
4. news.bbc.co.uk
5. 10.10.123.5
6. ak.bbc.co.uk

bbc.co.uk’s nameserver

Local nameserver

Generic Top-Level Domain Nameserver

[Slide: Bruce Maggs]
DNS resolution

1. End user
2. Browser cache
3. OS
4. news.bbc.co.uk
5. 10.10.123.5
6. ak.bbc.co.uk
7. akamai.net
8. Generc Top-Level Domain Nameserver

bbc.co.uk’s nameserver

a212-g.akamai.net

Local name server

Generic Top-Level Domain Nameserver

[Slide: Bruce Maggs]
DNS resolution

- **End user**

- **Browser cache**

- **OS**

- **Local name server**

- **Generic Top-Level Domain Nameserver**

- **bbc.co.uk’s nameserver**

- **a212.g.akamai.net**

- **ak.bbc.co.uk**

- **news.bbc.co.uk**

- **10.10.123.5**

- **akamai.net**

- **15.15.125.6**

[Slide: Bruce Maggs]
DNS resolution

1. End user
2. Browser cache
3. Local nameserver
4. news.bbc.co.uk
5. akamai.net
6. ak.bbc.co.uk
7. a212.g.akamai.net
8. 10.10.123.5
9. 15.15.125.6
10. g.akamai.net

bbc.co.uk's nameserver

Generic Top-Level Domain Nameserver

Akamai TLNS

[Slide: Bruce Maggs]
DNS resolution

End user

browser cache

OS

Local name server

bbc.co.uk’s nameserver

a212.g.akamai.net

ak.bbc.co.uk

akamai.net

news.bbc.co.uk

10.10.123.5

15.15.125.6

g.akamai.net

Generic Top-Level Domain Nameserver

Akamai TLNS

Select cluster

[Slide: Bruce Maggs]
DNS resolution

1. End user
2. Browser cache
3. OS
4. news.bbc.co.uk
5. 10.10.123.5
6. a212.g.akamai.net
7. akamai.net
8. 15.15.125.6
9. 20.20.123.55
10. g.akamai.net
11. Akamai TLNS

Select cluster

[Slide: Bruce Maggs]
DNS resolution

- **End user**
- **Browser cache**
- **OS**
- **End user's nameserver**
  - CNAME: a212.g.akamai.net
  - ak.bbc.co.uk
- **Local nameserver**
  - 10.10.123.5
- **Generic Top-Level Domain Nameserver**
  - 20.20.123.55
  - 10.10.123.5
- **Akamai TLNS**
  - 20.20.123.55
  - 15.15.125.6
- **Akamai LLNS**
  - a212.g.akamai.net
  - g.akamai.net

**Select cluster**

[Slide: Bruce Maggs]
DNS resolution

End user

Browser cache

OS

bbc.co.uk’s nameserver

a212.g.akamai.net

ak.bbc.co.uk

CNAME

Generic Top-Level Domain Nameserver

10.10.123.5

akamai.net

15.15.125.6

20.20.123.55

a212.g.akamai.net

Akamai LLNS

Akamai TLNS

Select cluster

Select servers within cluster

[Slide: Bruce Maggs]
DNS resolution

Select cluster

Select servers within cluster

End user

Browser cache

OS

bbc.co.uk’s nameserver

CNAME

a212.g.akamai.net

ak.bbc.co.uk

4 news.bbc.co.uk

5 10.10.123.5

akamai.net

8 15.15.125.6

10 g.akamai.net

20.20.123.5

30.30.123.5

11 a212.g.akamai.net

12 Akamai TLNS

13 Akamai LLNS

Generic Top-Level Domain Nameserver

Local nameserver

[Slide: Bruce Maggs]
DNS resolution

End user

Browser cache

OS

Local nameserver

GTC

10.10.123.5

akamai.net

news.bbc.co.uk

20.20.123.5

a212.g.akamai.net

30.30.123.5

a212.g.akamai.net

20.20.123.5

End user

Browser cache

OS

Select cluster

Select servers within cluster

[Slide: Bruce Maggs]
Akamai DNS redirection

Web clients’ network

Any network

Edge server cluster

Select server

LLNS

TTL 20s

Content

End user

Content provider’s network

Network close to client

Content server

(news.bbc.co.uk)

HTML

Select cluster

TLNS

TTL 30–60m