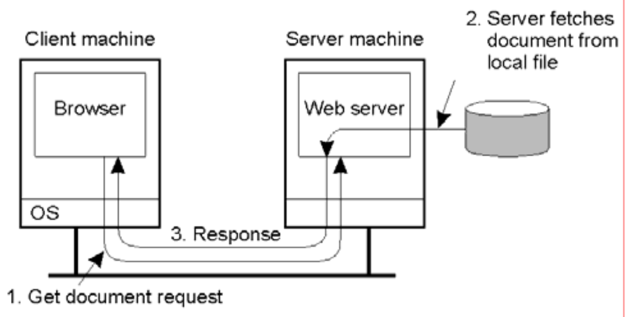
Distributed Computing Systems

The Web

The World Wide Web



- Huge client-server system
- Traditionally → Document-based
  - Referenced by "Uniform Resource Locator" (URL)

References

- [TS02] Andrew Tanenbaum and Marten van Steen, *Distributed Systems – Principles and Paradigms*, Prentice Hall, ©2002 (ch 11)
- [KR10] James F. Kurose and Keith W. Ross, *Computer Networking - A Top-Down Approach* (5th ed), Pearson, ©2010 (ch 2.2.4)
- [SAAF08] F. Schneider, S. Agarwal, T. Alpcan, and A. Feldmann. The New Web: Characterizing AJAX Traffic, In *Proceedings of the Passive and Active Measurement Conference (PAM)*, Cleveland, OH, USA, 2008

Outline

- Introduction (done)
- Document Model (next)
- Architecture
- Processes
- Caching
- Web 2.0

Document Model

- All information in documents
  - Typically in Hypertext Markup Language (HTML)
  - Different types: ASCII, scripts

```
<HTML>                                              <!- Start of HTML document     -->
<BODY>                                              <!- Start of the main body     -->
<H1>Hello World</H1>                                <!- Basic text to be displayed  -->
</BODY>                                             <!- End of main body           -->
</HTML>                                             <!- End of HTML section        -->

<HTML>                                              <!- Start of HTML document     -->
<BODY>                                              <!- Start of the main body     -->
<SCRIPT type = "text/javascript">                   <!- identify scripting language -->
  document.writeln ("<H1>Hello World</H1>);         // Write a line of text
</SCRIPT>                                           <!- End of scripting section   -->
</BODY>                                             <!- End of main body           -->
</HTML>                                             <!- End of HTML section        -->
```

- Scripts give you “mobile code” (more later)
- Can also have eXtensible Markup Language (XML)
  - Provides structure to document

XML Document Type Definition

(#PCDATA is primitive type, series of chars)

```
<!ELEMENT article (title, author+,journal)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (name, affiliation?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT affiliation (#PCDATA)>
<!ELEMENT journal (jname, volume, number?, month? pages, year)>
<!ELEMENT jname (#PCDATA)>
<!ELEMENT volume (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT month (#PCDATA)>
<!ELEMENT pages (#PCDATA)>
<!ELEMENT year (#PCDATA)>
```

- Definition above refers to journal article. Specifies type.
  - *Document Type Definition* (DTD)
  - Provides *structure* to XML documents → can test if XML is valid based on DTD

XML Document

Refers to previous dtd

```
<?xml = version "1.0">
<!DOCTYPE article SYSTEM "article.dtd">
<article>
   <title>Prudent Engineering Practice for Cryptographic Protocols</title>
   <author><name>M. Abadi</name></author>
   <author><name>R. Needham</name></author>
   <journal>
            <jname>IEEE Transactions on Software Engineering</jname>
            <volume>22</volume>
            <number>12</number>
            <month>January</month>
            <pages>6 – 15</pages>
            <year>1996</year>
   </journal>
</article>
```

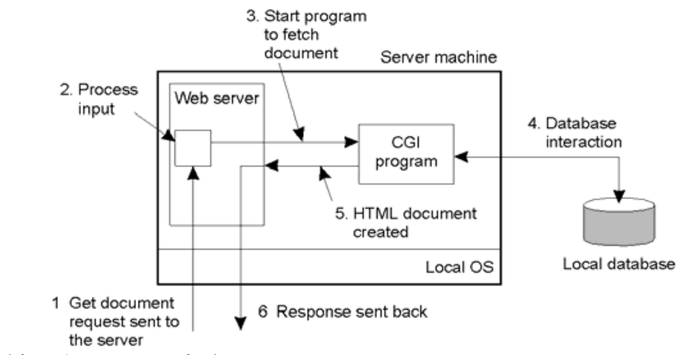
- (An XML document using XML definitions from previous slide)
- Formatting rules usually applied by embedding in HTML

Outline

- Introduction (done)
- Document Model (done)
- Architecture (next)
- Processes
- Caching
- Web 2.0

Architectural Overview

- Text documents typically "processed" on client
  - But can be done at server, too (e.g., Common Gateway Interface (CGI))



(often with user input , e.g., a form)

Server-Side Scripts

- Like Client, Server can execute JavaScript

```
(1)     <HTML>
(2)     <BODY>
(3)     <P>The current content of <pre>/data/file.txt</PRE>is:</P>
(4)     <P>
(5)     <SERVER type = "text/javascript");
(6)         clientFile = new File("/data/file.txt");
(7)         if(clientFile.open("r")){
(8)                 while (!clientFile.eof())
(9)                     document.writeln(clientFile.readln());
(10)                clientFile.close();
(11)        }
(12)    </SERVER>
(13)    </P>
(14)    <P>Thank you for visiting this site.</P>
(15)    </BODY>
(16)    </HTML>
```

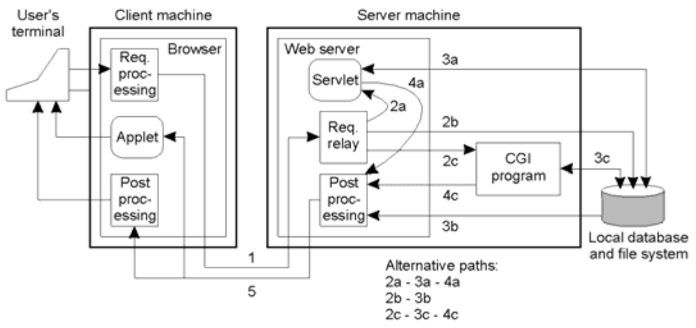
(The tag <SERVER…> is system specific)

- Server can also pass pre-compiled code → *applet*

<OBJECT codetype="application/java" classid="java.welcome.class">

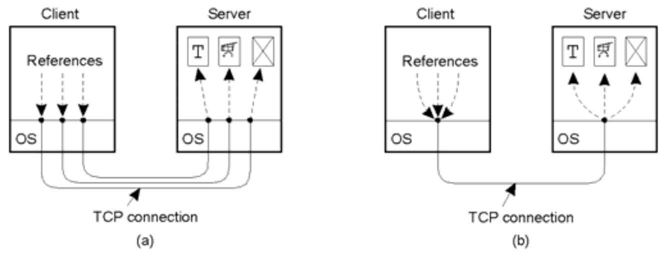
- *Servlets* are applets that run on server side (Architecture next slide)

Architectural Overview



HTTP Connections

- Communication based on Hypertext Transfer Protocol (HTTP)
  - Client request, server reply protocol
  - Uses TCP



a) Using *non-persistent* connections (HTTP 1.0)
- TCP connection setup expensive

b) Using *persistent* connections (HTTP 1.1)

- Can also have requests either *serially* or *in parallel*

User-server State: Cookies (1 of 3)

Web servers stateless

Many major Web sites use cookies to have state at server

Four components:

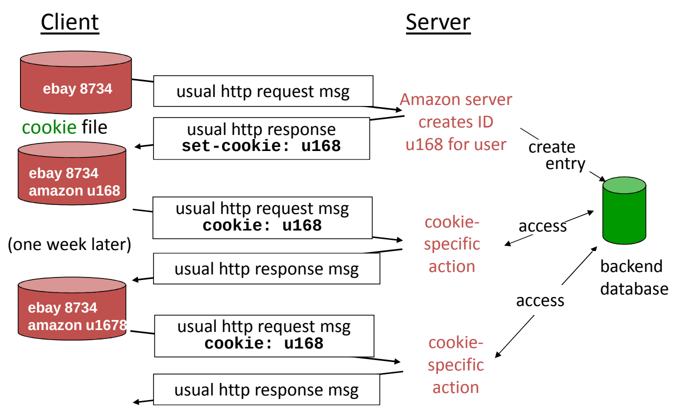
1) cookie header line of HTTP *response* message
2) cookie header line in HTTP *request* message
3) cookie file kept on user's PC, managed by user's browser
4) back-end database at Web site

Example:

- Susan always access Internet from same PC (e.g., in her home)
- Visits specific e-commerce site for first time (e.g., amazon.com)
- When initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

(see next slide for example)

User-server State: Cookies (2 of 3)



User-server State: Cookies (3 of 3)

What cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

Cookies and privacy:

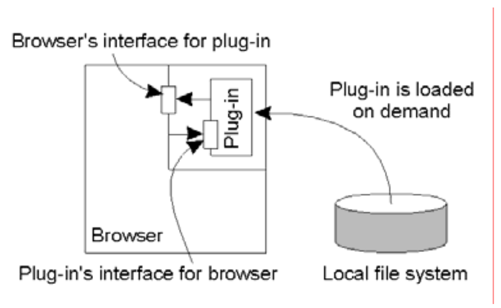
- Cookies permit some sites to learn lots about users
  - e.g., tracking visits by 3rd party sites
- Also, persist when session closed, so next user has
  - e.g., public computer
- Or stolen (sniffed), or sent to attacker if script embedded in primary Web page

Outline

• Introduction	(done)
• Document Model	(done)
• Architecture	(done)
• Processes	(next)
• Caching	
• Web 2.0	

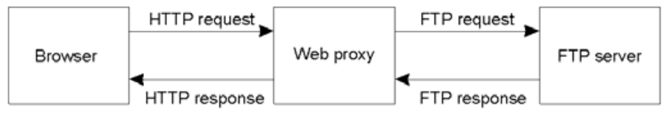
Client Process: Extensible Browser

- Make client browser extensible → do more than default
  - A plug-in
  - Associated with document type (MIME type)


Browser's interface for plug-in
Plug-in
Plug-in is loaded on demand
Browser
Plug-in's interface for browser
Local file system

Client-Side Process: Web Proxy

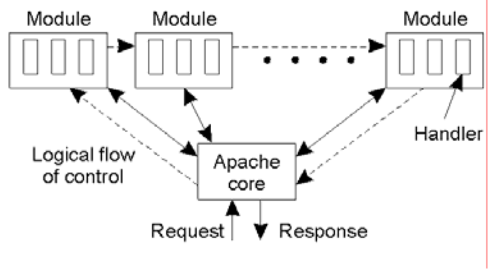
- Initially, handle connection when browser does not "speak" language



- Now, most browsers can handle, but proxies still used for common cache when many browsers (e.g., NZ) or to get "rights" of domain (e.g., `proxy.wpi.edu`)
- Can be useful for debugging Web session

Fiddler
Web Debugging Proxy

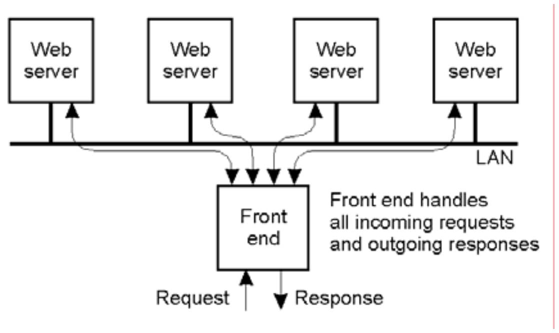
Servers



- Core invokes modules with data
  - Actual module path depends upon data type
- Phases:
  - authentication, response, syntax checking, user-profile, transmission
- Extend server to support different types (PHP)

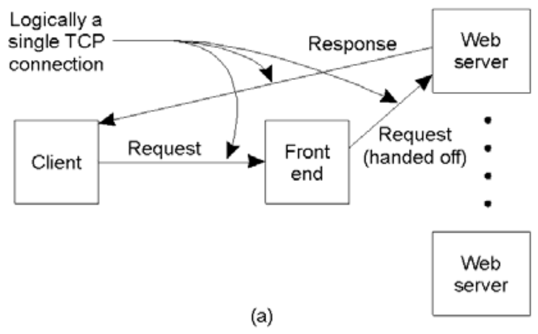
Server Clusters (1 of 3)

- Single server can become heavily loaded



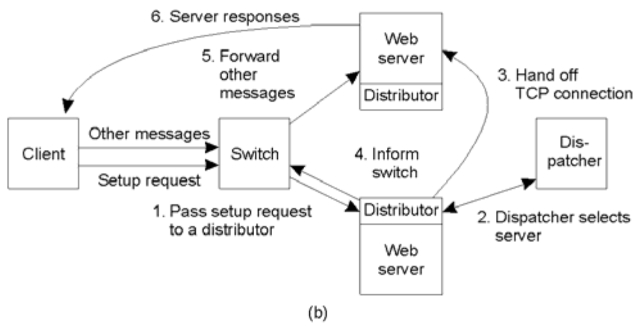
- Front-end replicates request to back-end (horizontal distribution)
  - But can become overloaded still since all connections through

Server Clusters (2 of 3)



- TCP handoff – most "work" on response, typically
  - But can't take advantage of document knowledge or caching
  - Or, higher-layer has to do more work, making front-end bottleneck

Server Clusters (3 of 3)



- Distributor talks to dispatcher initially, then hands off connection
- Front-end switch can stay at TCP layer, told where to send data

Outline

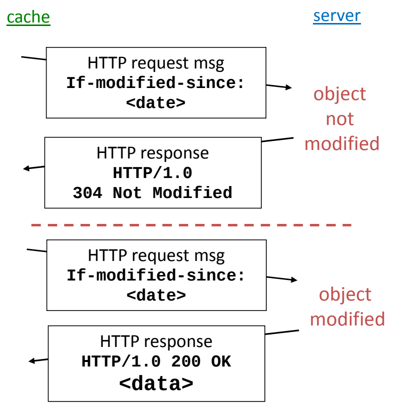
• Introduction	(done)
• Document Model	(done)
• Architecture	(done)
• Processes	(done)
• Caching	(next)
• Web 2.0	

Web Caching

- Browser keeps recent requests
  - Proxy can be valuable if *shared* interests
- Check cache first, server next
- Cache is full. How to decide replacement?
  - LRU (what is different than pages or disk blocks?)
  - e.g., *GreedyDual* (value divided by size)
- How consistent should client cache be to server content? What are tradeoffs?

Caching - Conditional GET

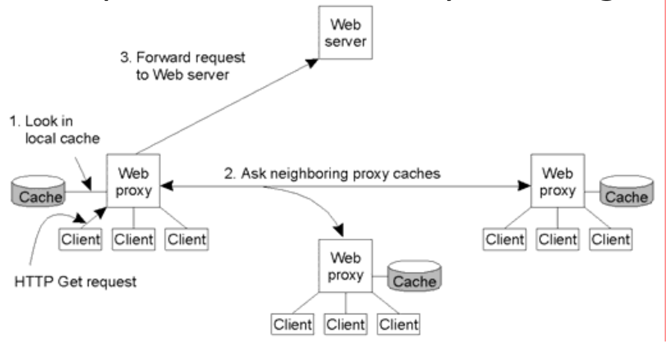
- Goal: don't send object if cache has up-to-date cached version
- *cache*: specify date of cached copy in HTTP request:
  `If-modified-since: <date>`
- *server*: response contains no object if cached copy is up-to-date. e.g.,
  `HTTP/1.0 304 Not Modified`



Cache Coherency

- Strong consistency
  - Validate each access
  - Server indicates if invalid
  - But requires request to server for *each* client request
- Weak consistency
  - Validate only when client clicks "refresh"
  - Or, using heuristic Time To Live (TTL)
    Squid $T_{expire} = \alpha(T_{cached} - T_{last\_modified}) + T_{cached}$
    $\alpha = 0.2$ (derived from practice)
- Why not have server *push* invalidation?
- In practice, cache hits low (50% max, only if really large)
  - Make "cooperative" caches

Cooperative Web Proxy Caching



- Proxy first checks neighbors before asking server
  - Shown effective for 10,000+ users
- But complicated, and often not clear win over single proxy

Andrew Tanenbaum and Marten van Steen, *Distributed Systems – Principles and Paradigms*, Prentice Hall, ©2002

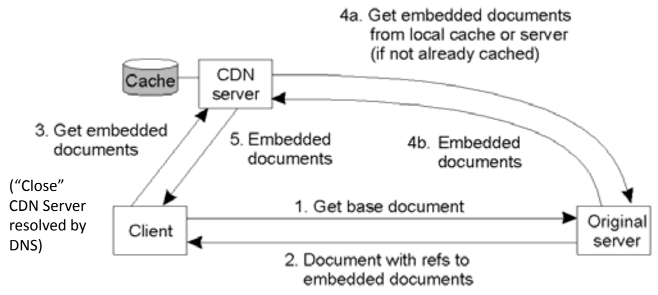
Misc Caching

- Static vs. Dynamic documents
  - Caching only effective for static docs (non CGI)
    - But Web increasingly dynamic (personalized)
    - Cookies used since server (mostly) stateless
  - Make proxies support active caching
    - Generate HTML
    - Need copies of server-side scripts/code
    - Accessing databases harder
- Caching large documents
  - Can only send changes from original
  - But in many cases, connection request is largest cost

Server Replication

- Clusters (covered)
- Deploy entire copy of Web site at another site (mirror)
  - Often done with file transfer networks (e.g., FTP servers)
  - But non-transparent (i.e., explicitly choose mirror)
- Content Delivery Network (CDN)
  - Have network of cooperative caches run by provider (next)

Akamai CDN


4a. Get embedded documents from local cache or server (if not already cached)
Cache
CDN server
3. Get embedded documents
5. Embedded documents
4b. Embedded documents
("Close" CDN Server resolved by DNS)
Client
1. Get base document
Original server
2. Document with refs to embedded documents

- Embedded documents have names that are resolved by Akamai DNS to local CDN server
  - Use Internet "map" to determine local server
- Local server gets copy from original server
- Akamai has many CDN servers "close" to clients

Andrew Tanenbaum and Marten van Steen, *Distributed Systems – Principles and Paradigms*, Prentice Hall, ©2002

Outline

• Introduction	(done)
• Document Model	(done)
• Architecture	(done)
• Processes	(done)
• Caching	(done)
• Web 2.0	(next)

The Web 1.0

- World Wide Web most popular application running over HTTP
- Users click on links embedded in documents to fetch additional documents
- Thus, in basic Web 1.0, HTTP request-response loads new page in response to each user request

The Web 2.0

- Shift away from class request-response, to *asynchronous* communication
- Web browsers request data *without* human intervention (e.g., without clicking on link/button)
- Such requests can mask latency or limited bandwidth
  - pre-fetching or
  - displaying before all data has arrived
- HTTP requests are becoming automated, rather than human generated
- AJAX is one such supporting technology

AJAX – What is it?

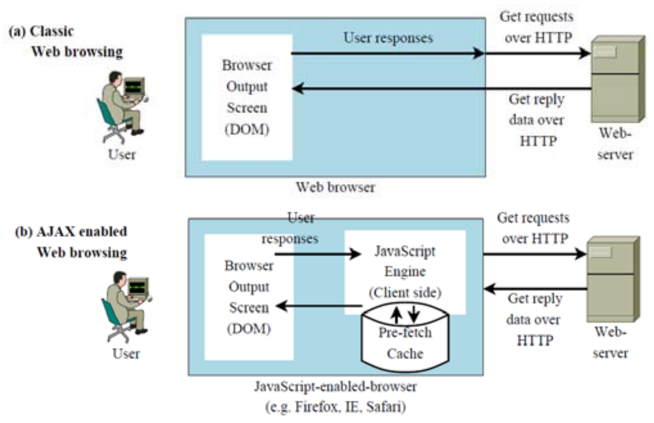
- AJAX – acronym for *Asynchronous JavaScript* and *XML*
  - Despite name, doesn't have to be XML (often *JSON*, a JavaScript text-based notation)
- Not stand-alone language or technology
- Methods for *client-side* code to create Web applications
- Clients send and receive data asynchronously, without interfering with display

AJAX Technologies

- AJAX uses:
  - Javascript (for altering page)
  - XML or JSON (for information exchange)
  - ASP or JSP (server side)
- Key is the XMLHttpRequest object
- All modern browsers support XMLHttpRequest object
  - Note: IE5 and IE6 uses an ActiveXObject.
- XMLHttpRequest object exchanges data with server asynchronously
  - Update parts of Web page, without reloading whole page

(illustration next)

AJAX versus Normal Web Browsing



Web 2.0 Applications

- Google Maps was early adopter of AJAX
  - Encouraged others to use for interactive apps
- Many Web-email offerings use AJAX technologies to rival desktop mail clients

The Problem

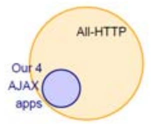
- Prediction algorithms may fetch data that is not needed → add extra data to Web browsing
- Fetching no longer limited by user read and click speed, but driven by JavaScript code logic
- Traditional Web traffic models (e.g., data rate for browsing, number of objects per page) may not hold

Web 2.0 Outline

- Introduction (done)
- Approach (next)
- Results
- Conclusions

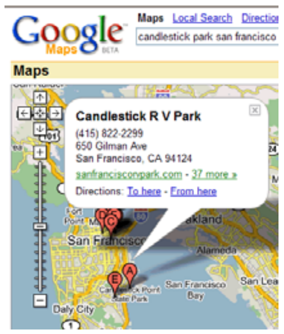
Approach

- Pick four popular, "representative", AJAX apps and study their traffic:
  - Google Maps
  - Google Mail
  - Gmx.de (popular German Webmailer)
  - Lokalisten.de (popular German social network)
- Traces from networks in Munich, Germany and Berkeley, USA
- Highlight differences in Web 2.0 traffic compared to traditional Web 1.0 traffic



Google Maps Internals

- Google Maps is AJAX application
  - Sometimes called the "canonical" AJAX app
- Prefetches tiles
- Opens multiple TCP connections to different servers
- Uses cookies to identify users
- HTTP 1.1 transfers data via persistence and pipelining
  - Mostly to fetch new image tiles
  - Also control, GUI-related pictures, user queries



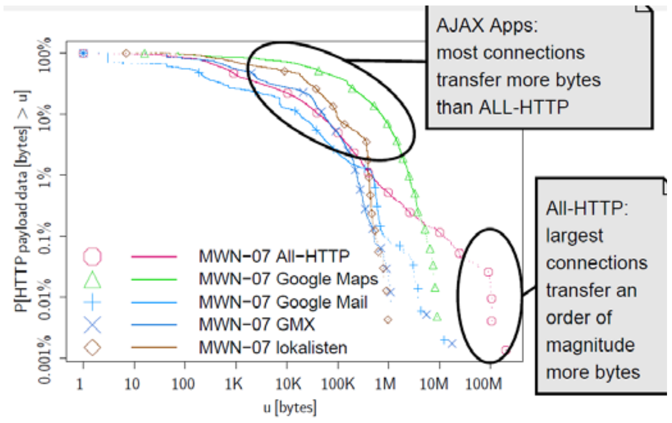
Analysis

- Extract standard HTTP request from all connections
  - Identified Google Maps via initial request to `maps.google.com` plus session cookies Google uses
- Studied:
  - Number of bytes transferred (HTTP payload)
  - Number of HTTP requests
  - Inter-request times
- Colors:
  - Red/Pink – All HTTP
  - Green – Google Maps

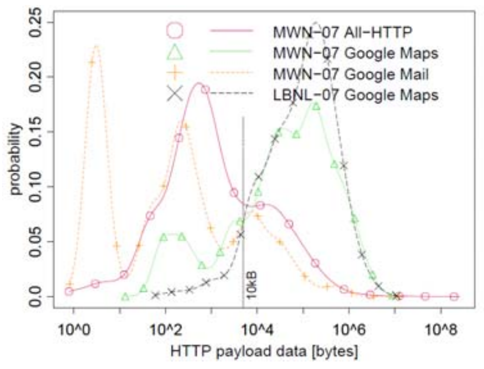
Web 2.0 Outline

• Introduction	(done)
• Approach	(done)
• Results	(next)
• Conclusions	

Bytes Per Connection (HTTP Payload)

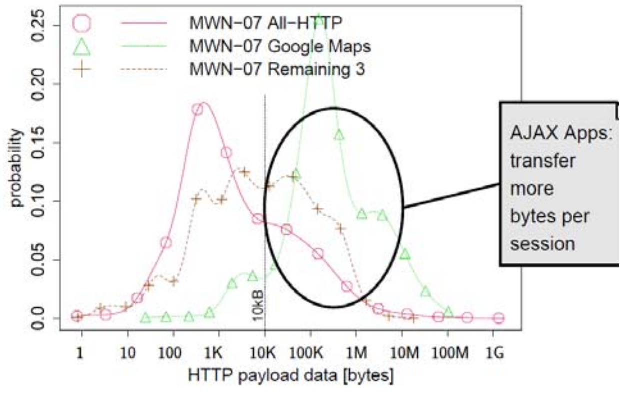


Bytes Per Connection (HTTP Payload)

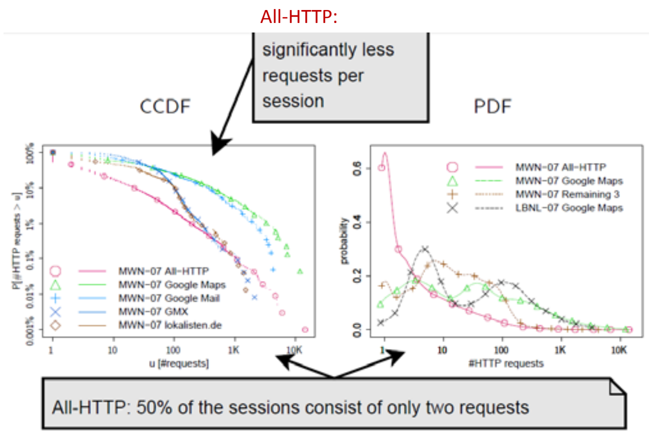


Compare HTTP 1.0, Maps to the right, Mail to the left
(HTTP 1.0 has not changed much since 1997)

Bytes Per Session (HTTP Payload)



Requests per Session



Conclusions

- On average, versus All-HTTP, AJAX apps
  - Transfer more bytes per connection and per session
  - Have more requests per session
  - Have significantly shorter inter-request times
- Larger sessions and burstier traffic with AJAX applications
- Existing Web traffic models still applicable, but need new parameterization