




Web

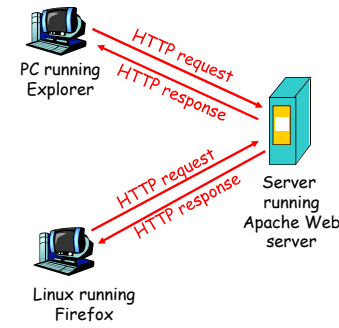
Kai Shen

1




Web and HTTP

- **Web:** the Internet application for distributed publishing and viewing of content
- **Client/server model**
 - **server:** hosts published content and sends the content upon request
 - **client:** requests, receives, and displays content
- **HTTP:** the comm. protocol supporting the web
 - request/response format




2



URLs

- Each web object managed by a server has a unique name called a URL (Universal Resource Locator)
- URL examples:
 - `http://www.rockester.edu:80/`
 - `http://www.rockester.edu/`
 - `http://www.rockester.edu`
 - Identifies an object called '/', managed by a Web server at `www.rockester.edu` that is listening on port 80.
- Include an authority component (server) and page component

3



Anatomy of an HTTP Transaction

```

unix> telnet www.rockester.edu 80
Trying 128.151.77.39...
Connected to www.rockester.edu.
Escape character is '^]'.
GET / HTTP/1.1
host: www.rockester.edu

HTTP/1.1 200 OK
Date: Tue, 14 Apr 2015 14:38:32 GMT
Server: Apache
Transfer-Encoding: chunked
Content-Type: text/html
...
Connection closed by foreign host.
unix>
  
```

*Client: open connection to server
Telnet prints 3 lines to the terminal*

*Client: request line
Client: required HTTP/1.1 HOST header
Client: empty line terminates headers.
Server: responds with web page*

*Lots of stuff
Server: closes connection
Client: closes connection and terminates*

4

HTTP Requests

- HTTP request is a *request line*, followed by zero or more *headers*
- Request line: **<method> <uri> <version>**
 - **<version>** is HTTP version of request (**HTTP/1.0** or **HTTP/1.1**)
 - **<uri>** is the full URL, or URL suffix (if the server is known).
 - **<method>**
 - **GET**: Retrieve content
 - Workhorse method (99% of requests)
 - **POST**: Retrieve dynamic content with arguments
 - Arguments for dynamic content are in the request body
 - **OPTIONS**: Get server or file attributes
 - **HEAD**: Like **GET** but no data in response body
 -
- Request headers: **<header name>: <header data>**
 - Provide additional information to the server.

5

HTTP Responses

- HTTP response is a *response line* followed by zero or more *response headers*.
- Response line: **<version> <status code> <status msg>**
 - **<version>** is HTTP version of the response.
 - **<status code>** is numeric status.
 - **<status msg>** is corresponding English text.
 - 200 OK Request was handled without error
 - 301 Moved Provide alternate URL
 - 403 Forbidden Server lacks permission to access file
 - 404 Not found Server couldn't find the file.
- Response headers: **<header name>: <header data>**
 - Provide additional information about response
 - **Content-Type**: MIME type of content in response body.
 - **Content-Length**: Length of content in response body.

6

How much to receive?

- Finish on close?
- Standard
 - Specify total length with content-length
 - Requires that program buffers entire message
- Chunked
 - Break into blocks
 - Prefix each block with number of bytes (Hex coded)

7

Chunked Encoding Example

```

HTTP/1.1 200 OK\r\n
Date: Sun, 31 Oct 2010 20:47:48 GMT\r\n
Server: Apache/1.3.41 (Unix)\r\n
Keep-Alive: timeout=15, max=100\r\n
Connection: Keep-Alive\r\n
Transfer-Encoding: chunked\r\n
Content-Type: text/html\r\n
\r\n
d75\r\n
<html>
<head>
<link href="http://www.cs.cmu.edu/style/calendar.css" rel="stylesheet"
type="text/css">
</head>
<body id="calendar_body">

<div id="calendar"><table width="100%" border="0" cellpadding="0"
cellspacing="1" id="cal">

. . .
</body>
</html>
\r\n
0\r\n
\r\n

```

First Chunk: 0xd75 = 3445 bytes

Second Chunk: 0 bytes (indicates last chunk)

8

HTTP Connection Persistency

TCP per-connection overhead:

- Connection establishment
- Congestion control: slow start

Non-persistent HTTP (1.0)

- At most one object is sent over a TCP connection.
- Pays TCP per-connection overhead for each object.

Persistent HTTP (1.1)

- Multiple objects can be sent over single TCP connection between the browser and web server.
- **Connection: Keep-Alive**

9

HTTP Versions

- Major differences between HTTP/1.1 and HTTP/1.0
 - Connection persistency
 - HTTP/1.1 supports *chunked encoding*
 - Transfer-Encoding: chunked
 - HTTP/1.1 requires **HOST** header
 - **Host: www.hosting-company.com**
 - Makes it possible to host multiple websites at single Internet host

10

Complexity of Web Server/Client Implementation

- Easy to implement a web server?
 - Tiny Web server described in text (226 lines of commented C code).
 - ~400 lines Java Web server on the web
 - Too simple for an assignment ☺
 - Complexity in performance, scalability, robustness, and security
- Web clients (browsers) are complex in user interactions

11

Proxies

- A *proxy* is an intermediary between a client and an *origin server*.
 - To the client, the proxy acts like a server.
 - To the server, the proxy acts like a client.

```

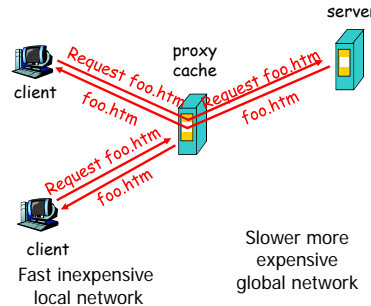
graph LR
    Client((Client)) -- "1. Client request" --> Proxy((Proxy))
    Proxy -- "2. Proxy request" --> OriginServer((Origin Server))
    OriginServer -- "3. Server response" --> Proxy
    Proxy -- "4. Proxy response" --> Client
  
```

- Can perform useful functions as requests and responses pass by
 - Examples: Caching, logging, anonymization, filtering, transcoding

12

Web Proxy Caching

- Cache is installed and shared by users (university, company, residential ISP)
- Goal:** satisfy client requests without involving the original server.
- Client sends all HTTP requests to proxy cache
 - object in cache: cache returns object
 - otherwise cache requests object from the original server, then returns object to client
- Benefits
 - bandwidth reduction
 - latency improvement



13

Cache Content Staleness

- Content providers lose direct control of cache content
 - retain some control through cached content staleness
- Is the cached page up-to-date?
 - using **If-modified-since** HTTP header.
 - removing pages that are too old.

14

Web Prefetching

- Prefetch a web page before client makes access
 - Save latency, but not bandwidth
- Prefetching heuristics?
 - Hyperlinks in the current page (assume client may click some of them)
 - Predict future accesses based on past browsing history
- Where to do prefetching?
- Our study: web prefetching is not effective
 - Non-prefetchable dynamic applications are increasingly dominant
 - Benefits of history-based prefetching is diminished by proxy caching

15

Mining of Cache Logs

- Web cache logs contain a wealth of information
 - List of who accessed what at when
- User privacy
- Aggregate statistics
 - Most popular web objects, distribution of web object popularity
 - General user access models (think time between accesses, pattern of page browsing sequence, ...)
 - ...

16



Assignment #5

- Implement a web proxy server
- Allow concurrency through multi-threading
- Reap allocated resources
 - Free dynamic memory, close connection sockets after use, reap zombie children processes
- Testing using telnet or some testing tools
- C or Java

17



Disclaimer

These slides were adapted from the CMU course slides provided along with the textbook of "Computer Systems: A programmer's Perspective" by Bryant and O'Hallaron. The slides are intended for the sole purpose of teaching the computer organization course at the University of Rochester.

18