Web Services

Kai Shen

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 communication protocol supporting the web
  - request/response format

**PC running Explorer**
**Server running Apache Web server**
**Linux running Firefox**

**URLs**

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

**Anatomy of an HTTP Transaction**

```bash
unix> telnet www.rochester.edu 80
Trying 128.151.57.101...
Connected to www.rochester.edu.
Escape character is '^]'.
GET /index.html HTTP/1.1
host: www.rochester.edu
GET /index.html HTTP/1.1
host: www.rochester.edu
GET /index.html HTTP/1.1
host: www.rochester.edu
HTTP/1.1 200 OK
Date: Tue, 17 Apr 2012 15:48:32 GMT
Server: Apache
Transfer-Encoding: chunked
Content-Type: text/html
...
Lots of stuff
Connection closed by foreign host.
unix>
```
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
    - POST: Retrieve dynamic content with arguments
    - 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.

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.

How much to receive?

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

Chunked Encoding Example

```
HTTP/1.1 200 OK
Date: Sun, 31 Oct 2010 20:47:48 GMT
Server: Apache/1.3.41 (Unix)
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

<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'>
... ...
</table></div>
</body></html>
```

First Chunk: 0xd75 = 3445 bytes

```
.. ...
</body>
</html>
```

Second Chunk: 0 bytes (indicates last chunk)
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

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
  - HTTP/1.1 adds additional support for caching

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

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.

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

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.

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

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, ...)
  - ...

![Diagram of Web Proxy Caching](image)
Assignment #6

- Implement a web proxy server
- Allow concurrency through multi-threading
- Testing using telnet
- C and Java

Security of Web Applications

- Interconnectedness exacerbates security threats
  - E.g., the buffer overflow problem would not have caused as much damage if it wasn’t in a network server (fingerd)
- Effective countermeasure needs cooperation between architecture/operating-system/networking/programming-language

Network Security Threat: Cross-Site Scripting

- Cross-site scripting:
  - duped to run script unintended by the original site
  - most significant vulnerability for web applications today
- Examples:
  - online message board FOOBAR displays user messages without much checking, attacker prepares a message that includes HTML tag and JavaScript code; a user who trusts FOOBAR views the message ...
  - attacker embeds attack strings in machine names (due to Simon Weber)

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.