Applications & Application-Layer Protocols:
The Web & HTTP

Dr. Michele Weigle
Department of Computer Science
Old Dominion University
mweigle@cs.odu.edu

http://www.cs.odu.edu/~mweigle/CS312-F11/

Application-Layer Protocols
Outline

◆ The architecture of distributed systems
  » Client/Server computing
  » P2P computing
  » Hybrid (Client/Server and P2P) systems

◆ Example client/server systems and their application-layer protocols
  » The World-Wide Web (HTTP)
  » E-mail (SMTP & POP)
  » Internet Domain Name System (DNS)
The Web & HTTP

Outline

◆ Terminology

◆ HTTP protocol
  » request message format
  » response message format
  » HTTP/1.0 – non-persistent connections
  » persistent connections

◆ Cookies

◆ Web caches

◆ Security

Application-Layer Protocols

The Web

◆ User agent (client) for the Web is called a browser
  » MS Internet Explorer
  » Mozilla Firefox
  » Apple Safari
  » Google Chrome

◆ Server for the Web is called a Web server
  » Apache (public domain)
  » MS Internet Information Server (IIS)
Application-Layer Protocols

Web terminology

- **Web page:**
  - Addressed by a URL
  - Consists of “objects”

- **Most Web pages consist of:**
  - Base HTML page
  - Embedded objects
Web Terminology
URLs (Universal Resource Locators)

Optional server port (Default = port 80)

www.someSchool.edu:8080/someDept/pic.gif

Server domain name
Object path name

◆ URL components
  » Server address
  » (Optional port number)
  » Path name

Web Terminology
The Hypertext Transfer Protocol (HTTP)

◆ Web’s application layer protocol

◆ Client/server model
  » client: browser that requests, receives, “displays” Web objects
  » server: Web server sends objects in response to requests
The Web & HTTP

Outline

Terminology

HTTP protocol
  » request message format
  » response message format
  » HTTP/1.0 – non-persistent connections
  » persistent connections

Cookies

Web caches

Security

The Hypertext Transfer Protocol

HTTP Overview

HTTP uses TCP
  » Browser initiates TCP connection to server (on port 80)

HTTP messages (application-layer protocol messages) exchanged between browser and Web server

HTTP/1.0: RFC 1945
  » One request/response interaction per connection

HTTP/1.1: RFC 2616
  » Persistent connections
  » Pipelined connections

HTTP is “stateless”
  » Server maintains no information about past browser requests

Aside

Protocols that maintain “state” are complex!
  » Past history (state) must be maintained
  » If server or client crashes, their views of “state” may be inconsistent and must be reconciled
The Hypertext Transfer Protocol

HTTP example

- User enters URL `www.someSchool.edu/someDept/home.index`
  - Referenced object contains HTML text and references 10 JPEG images
- Browser sends an HTTP “GET” request to the server `www.someSchool.edu`
- Server will retrieve and send the HTML file
- Browser will read the file and sequentially make 10 separate requests for the embedded JPEG images

HTTP 1.0 Example

URL `www.someschool.edu/someDept/home.index`

1) Browser initiates TCP connection to server at `www.someSchool.edu`.
   Port 80 is “well known” for server.
2) Server “accepts” connection
3) Client writes an HTTP GET request message (containing path) to TCP connection socket
4) Server reads request message, forms response message containing requested object, writes message to socket
5) Server closes TCP connection
HTTP 1.0 Example

URL www.someschool.edu/someDept/home.index

6) Browser reads response message containing the HTML file. Ten references to JPEG objects are found during the HTML parse.

7) Browser initiates TCP connection to server at www.someschool.edu.

TCP 3-way handshake

8) Server “accepts” connection.

The above steps are repeated for each of the 10 JPEG objects.

The Hypertext Transfer Protocol

HTTP message format

- Two types of HTTP message formats: request and response messages
  - ASCII (human-readable format)

- HTTP request message:
  - Request line
  - Optional header lines
  - Present only for some methods (e.g., POST)
HTTP Message Format
Google Chrome & Firefox request examples

◆ How does Chrome process:

http://www.cs.odu.edu/~mweigle/?

```
GET /~mweigle/ HTTP/1.1
Host: www.cs.odu.edu
Connection: keep-alive
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/13.0.782.220 Safari/535.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

```
GET /~mweigle/ HTTP/1.1
Host: www.cs.odu.edu
Connection: keep-alive
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6_8) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/13.0.782.220 Safari/535.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

```
GET /~mweigle/ HTTP/1.1
Host: www.cs.odu.edu
Connection: keep-alive
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6; en-US; rv: 1.9.2.3) Gecko/20100401 Firefox/3.6.3 GTB7.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
```
HTTP Message Format

General response message format

- Response messages
  - ASCII (human-readable format)

- Message structure:
  - Status line
  - Optional header lines
  - Requested object, error message, etc.

```plaintext
version <SP> code <SP> phrase <CR><LF>
"header field name "." value <CR><LF>

header field name "." value <CR><LF>
<CR><LF>

entity body
```

HTTP Message Format

Telnet example

- Connect to HTTP server and port
- Telnet output

  ```
  % telnet www.cs.odu.edu 80
  Trying 128.82.4.2...
  Escape character is '^]'.
  GET /~mweigle/files/foo.txt HTTP/1.0
  HTTP/1.1 200 OK
  Date: Fri, 17 Sep 2010 17:37:06 GMT
  Server: Apache/2.2.14 (Unix) DAV/2 PHP/5.2.11
  Last-Modified: Tue, 15 Sep 2009 18:14:46 GMT
  ETag: "921a6-65-473a1d91187"
  Accept-Ranges: bytes
  Content-Length: 101
  Connection: close
  Content-Type: text/plain
  ** This test file is stored in the UNIX file system at
  ** /home/mweigle/public_html/files/foo.txt
  Connection closed by foreign host.
  ```
### HTTP Message Format

#### Telnet example (2)

```
$ telnet www.cnn.com 80
Trying 157.166.226.26...
Escape character is '\['.
GET /notexist.html HTTP/1.0
HTTP/1.1 404 Object Not Found
Date: Fri, 17 Sep 2010 17:40:11 GMT
Server: Apache
Set-Cookie: CG=US:VA:Norfolk; path=/
Accept-Ranges: bytes
Content-Type: text/html
Content-Length: 22516
Connection: close

<HTML> <HEAD> . . .
   . . .
   Error type 404 - Object Not Found
</body> </html>
```

**Connect to HTTP server port**

**Telnet output**

**Type GET command plus blank line**

**HTTP response status line**

**HTTP response headers plus blank line**

**Object content**

**Telnet output**

### HTTP Message Format

#### HTTP response status codes

- **Sample response codes:**
  - **200 OK**
    - Request succeeded, requested object later in this message
  - **301 Moved Permanently**
    - Requested object moved, new location specified later in this message (Location:)
  - **400 Bad Request**
    - Request message not understood by server
  - **404 Not Found**
    - Requested document not found on this server
  - **505 HTTP Version Not Supported**
HTTP Live Headers
Firefox Plugin

- Allows you to view HTTP request and response headers in Firefox


HTTP Protocol Design
TCP Connections and RTT

- HTTP uses TCP as its transport protocol
- Must establish a TCP connection to each web server that is contacted
- TCP connection setup (handshake) takes 1 round-trip time (RTT)
HTTP Protocol Design

Non-persistent connections

- The default browser/server behavior in HTTP/1.0 is for the TCP connection to be closed after the completion of the request
  - Server parses request, responds, and closes TCP connection
  - The `Connection: keep-alive` header allows for persistent connections

HTTP Protocol Design

Non-persistent connections

- With non-persistent connections at least 2 RTTs are required to fetch every object
  - 1 RTT for TCP handshake
  - 1 RTT for request/response
HTTP Protocol Design
Persistent v. non-persistent connections

- **Non-persistent**
  - HTTP/1.0
  - Server parses request, responds, and closes TCP connection
  - At least 2 RTTs to fetch every object

- **Persistent**
  - Default for HTTP/1.1 (negotiable in 1.0)
  - Client sends requests for multiple objects on one TCP connection
  - Server, parses request, responds, parses next request, responds...
  - Fewer RTTs

Non-Persistent Connections

- Client initiates new TCP connection for each HTTP request
- At least 2 RTTs for each object
**Persistent Connections**

- Client sets up TCP connection once per server
- One RTT for connection setup
- At least one additional RTT for each embedded object

---

**Non-Persistent vs. Persistent Connections**  
**Performance Example**

- A base page with five embedded images located on the same web server

- How many TCP connections to download the page with non-persistent connections?
  - How many round-trip times (RTTs)?

- How many TCP connections to download the page with persistent connections?
  - How many RTTs?
Non-Persistent vs. Persistent Connections

Performance Example

◆ A base page with five embedded images located on the same web server

◆ How many TCP connections to download the page with *non-persistent* connections?
  » 1 TCP connection to download the base page
  » 1 TCP connection to download each of the 5 embedded images
  » Total: 6 TCP connections

◆ How many round-trip times (RTTs)?
  » Each TCP connection requires a handshake: 1 RTT
  » Once connection is setup, it takes 1 RTT to download an object (send HTTP request, receive HTTP response)
  » Can only download one object per connection (*non-persistent*), so 2 RTTs per connection
  » Total: 2 RTTs * 6 connections = 12 RTTs

Non-Persistent vs. Persistent Connections

Performance Example

◆ A base page with five embedded images located on the same web server

◆ How many TCP connections to download the page with *persistent* connections?
  » 1 TCP connection to download the base page
  » Since all 5 embedded images are at the same web server as the base page, no more TCP connections are needed
  » Total: 1 TCP connection

◆ How many round-trip times (RTTs)?
  » Each TCP connection requires a handshake: 1 RTT
  » Once connection is setup, it takes 1 RTT to download an object (send HTTP request, receive HTTP response)
  » There are 6 objects (base page + 5 images) to download
  » Total: 1 RTT (TCP handshake) + 6 RTTs (download all objects) = 7 RTTs
Non-Persistent vs. Persistent Connections

Your Turn

◆ A base page with 3 embedded images located on the same web server

◆ How many TCP connections with non-persistent HTTP connections?
  » How many RTTs?

◆ How many TCP connections with persistent HTTP connections?
  » How many RTTs?

Non-Persistent vs. Persistent Connections

Your Turn

◆ A base page with 3 embedded images located on a different web server than the base page

◆ How many TCP connections with non-persistent HTTP connections?
  » How many RTTs?

◆ How many TCP connections with persistent HTTP connections?
  » How many RTTs?
A base page with 3 embedded images located on the same web server
- RTT from client to web server is 50 ms
- slowest link on the path is 2 Mbps
  - other links are high-speed, so transmission delay is negligible
- base page is 1000 bytes
- each image is 3000 bytes

What is the total time (propagation + transmission delays) needed to download the entire web page with non-persistent HTTP connections?
The Web & HTTP

Outline

◆ Terminology
◆ HTTP protocol
  » request message format
  » response message format
  » HTTP/1.0 – non-persistent connections
  » persistent connections
◆ Cookies
◆ Web caches
◆ Security

HTTP User-Server Interaction

Cookies

◆ Server sends “cookie” to browser in response message
  \[\text{Set-cookie}: \text{<value>}\]

◆ Browser presents cookie in later requests to same server
  \[\text{cookie: <value>}\]

◆ Server matches cookie with server-stored information
  » Provides authentication
  » Client-side state maintenance (remembering user preferences, previous choices, …)
HTTP User-Server Interaction

Browser caches

- Browsers cache content from servers to avoid future server interactions to retrieve the same content

HTTP User-Server Interaction

The conditional GET

- If object in browser cache is “fresh,” the server won’t resend it
  » Browsers save current date along with object in cache

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

- Server’s response contains the object only if it has been changed since the cached date

- Otherwise server returns:
  HTTP/1.0 304 Not Modified

HTTP request
If-modified-since:<date>

HTTP response
HTTP/1.0
304 Not Modified

HTTP request
If-modified-since:<date>

HTTP response
HTTP/1.0
200 OK

...<data>
HTTP Message Format

Telnet example

% telnet www.cs.odu.edu 80
Trying 128.82.4.2...
Escape character is '^]'.
GET /~mweigle/files/foo.txt HTTP/1.0
HTTP/1.1 200 OK
Date: Fri, 17 Sep 2010 17:37:06 GMT
Server: Apache/2.2.14 (Unix) DAV/2 PHP/5.2.11
Last-Modified: Tue, 15 Sep 2009 18:14:46 GMT
ETag: "921a6-65-473a1c1d91187"
Accept-Ranges: bytes
Content-Length: 101
Connection: close
Content-Type: text/plain
** This test file is stored in the UNIX
** file system at
** /home/mweigle/public_html/files/foo.txt
Connection closed by foreign host.

HTTP User-Server Interaction

Cache Performance for HTTP Requests

◆ What is the average time to retrieve a web object?
  » \[ T_{\text{mean}} = \text{hit ratio} \times T_{\text{cache}} + (1 - \text{hit ratio}) \times T_{\text{server}} \]
  » where \( \text{hit ratio} \) is the fraction of objects found in the cache
  » Assume mean access time from a disk cache = 10 ms
  » Assume mean access time from the origin server = 1000 ms
◆ For a 60% hit ratio, the mean client access time is:
  \[(0.6 \times 10 \text{ ms}) + (0.4 \times 1,000 \text{ ms}) = 406 \text{ ms}\]
Cache Performance for HTTP Requests
What determines the hit ratio?

◆ Cache size
◆ Locality of references
  » How often the same web object is requested
◆ How long objects remain “fresh” (unchanged)
◆ Object references that can’t be cached at all
  » Dynamically generated content
  » Protected content
  » Content purchased for each use
  » Content that must always be up-to-date
  » Advertisements (“pay-per-click” issues)

The Web & HTTP
Outline

◆ Terminology
◆ HTTP protocol
  » request message format
  » response message format
  » HTTP/1.0 – non-persistent connections
  » persistent connections
  » pipelining
◆ Authentication
◆ Cookies
◆ Web caches
◆ Security
Security
Authentication vs. Encryption

◆ Authentication
  » to confirm the sender is who they say they are
  » using a digital certificate issued by a trusted third party

◆ Encryption
  » to prevent others from reading the message
  » using a cipher (encryption-decryption algorithm)
  » with symmetric or asymmetric (related public and private) keys

Security
Encryption and Decryption

◆ Symmetric cryptography
  » Same key for encryption and decryption
    ▶ would be efficient as long as the key is pre-agreed and secure
  » Not suitable for the Web

◆ Asymmetric-key cryptography (also called Public-key cryptography)
  » Using a pair of related public and private keys.
    Encryption and decryption are asymmetric.
  » Used in HTTPS
Security

HTTPS

◆ HTTP over Secure Socket Layer
  » Secure version of HTTP
  » Encrypts the session data
    ❖ Using either the SSL (Secure Socket Layer) protocol or the TLS (Transport Layer Security) protocol
  » SSL and TLS work above TCP but below application protocols (HTTP, SMTP, etc.)
◆ Transferred using HTTP, encrypted
  » with default TCP/IP port 443
◆ For Web pages, the URL begins with https://
◆ Provides server authentication and encrypted communication

Application-Layer Protocols

Outline

◆ The architecture of distributed systems
  » Client/Server computing
  » P2P computing
  » Hybrid (Client/Server and P2P) systems
◆ Example client/server systems and their application-layer protocols
  » The World-Wide Web (HTTP)
  » E-mail (SMTP & POP)
  » Internet Domain Name System (DNS)