CS 43: Computer Networks

Layering & HTTP

September 7, 2018
Last Class: Five-layer Internet Model

- **Application:** the application (e.g., the Web, Email)
- **Transport:** end-to-end connections, reliability
- **Network:** routing
- **Link (data-link):** framing, error detection
- **Physical:** 1’s and 0’s/bits across a medium (copper, the air, fiber)
Last Class: Five-layer Internet Model

- **Physical layer** (Cat 5, Coax, Air, Fiber Optics)
- **Link Layer** (Ethernet, WIFI, Cable)
- **Network Layer** (IP): Choose paths for data
- **Transport Layer**: share capacity, ordering, error checking, congestion control, …
- **Application Layer**
## OSI Seven-Layer Model

<table>
<thead>
<tr>
<th>Layer</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>the application (e.g., the Web, Email)</td>
</tr>
<tr>
<td>Presentation</td>
<td>formatting, encoding, encryption</td>
</tr>
<tr>
<td>Session</td>
<td>sockets, remote procedure call</td>
</tr>
<tr>
<td>Transport</td>
<td>end-to-end connections, reliability</td>
</tr>
<tr>
<td>Network</td>
<td>routing</td>
</tr>
<tr>
<td>Link (data-link)</td>
<td>framing, error detection</td>
</tr>
<tr>
<td>Physical</td>
<td>1’s and 0’s/bits across a medium (copper, the air, fiber)</td>
</tr>
</tbody>
</table>
Because of our layering abstractions, we can use any technology we want at any layer.

A. Always
B. Usually
C. Sometimes
D. Never
Internet Protocol Suite

The Hourglass Model

Applications
Transport

Data Link
Physical

The Hourglass Model

“Thin Waist”

FTP
HTTP
VoIP
TFTP

TCP
UDP

IP

NET_1
NET_2
...
NET_n
Putting this all together

- **ROUGHLY**, what happens when I click on a Web page from Swarthmore?

My computer

www.google.com
Web request (HTTP)

• Turn click into HTTP request

GET http://www.google.com/ HTTP/1.1
Host: www.google.com
...
Name resolution (DNS)

• Where is www.google.com?

My computer (132.239.9.64)

Local DNS server (132.239.51.18)

What’s the address for www.google.com

Oh, you can find it at 66.102.7.104
Data transport (TCP)

- Break message into packets (TCP segments)
- Should be delivered reliably & in-order

```
GET http://www.google.com HTTP/1.1
Host: www.google.com
...
```
Global Network Addressing

- Address each packet so it can traverse network and arrive at host

My computer (132.239.9.64)

www.google.com (66.102.7.104)
(IP) At Each Router

- Where do I send this to get it closer to Google?
- Which is the best route to take?
Link & Physical Layers

• Forward to the next node!

• Share the physical medium.

• Detect errors.
TCP/IP Protocol Stack

- Application Layer
  - HTTP
- Transport Layer
  - TCP
  - IP
- Network Layer
  - IP
  - Ethernet interface
  - SONET interface
- Link Layer
  - Ethernet interface
  - SONET interface
  - Ethernet interface
Today

• Application layer protocol: HTTP
• HTTP and the Web
• HTTP Requests: GET, POST
• HTTP Response message
• Cookies
Which of these is NOT a component of a typical HTTP transaction?

A. Request line

B. Header line

C. User name

D. Object/Entity body
Web services often use cookies, which are designed to...

A. Keep state for a user on a single website.

B. Keep state for a user across all websites they visit.

C. Ensure a user’s privacy while web browsing.
A Web cache…

A. Compresses web objects.

B. Encrypts web objects.

C. Saves downloaded objects closer to the user.
Creating a network app

write programs that:
• run on (different) end systems
• communicate over network
• e.g., web server software communicates with browser software

no need to write software for network-core devices
• network-core devices do not run user applications
• applications on end systems allows for rapid app development, propagation

Adapted from Kurose and Ross
What IS A Web Browser?
HTTP and the Web

First, a review...

- **web page** consists of objects
- object can be HTML file, JPEG image, Java applet, audio file,...
- web page consists of **base HTML-file** which includes several referenced objects
- each object is addressable by a **URL**, e.g.,

  www.someschool.edu/someDept/pic.gif

  - **host name**
  - **path name**
HTTP: Hypertext transfer protocol

- client/server model
  - **client**: browser that requests, receives, (using HTTP protocol) and “displays” Web objects
  - **server**: Web server sends (using HTTP protocol) objects in response to requests

Adapted from: Kurose and Ross
HTTP Overview

1. User types in a URL.
   http://some.host.name.tld/directory/name/file.ext
2. Browser establishes connection with server.
   Looks up “some.host.name.tld”
   Calls connect()
HTTP Overview

3. Browser requests the corresponding data.
   GET /directory/name/file.ext HTTP/1.0
   Host: some.host.name.tld
   [other optional fields, for example:]
   User-agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
   Accept-language: en
4. Server responds with the requested data.
HTTP/1.0 200 OK
Content-Type: text/html
Content-Length: 1299
Date: Sun, 01 Sep 2013 21:26:38 GMT
(Data data data data data...
HTTP Overview

5. Browser renders the response, fetches any additional objects, and closes the connection.
HTTP Overview

1. User types in a URL.
2. Browser establishes connection with server.
3. Browser requests the corresponding data.
4. Server responds with the requested data.
5. Browser renders the response, fetches other objects, and closes the connection.

It’s a document retrieval system, where documents point to (link to) each other, forming a “web”.

Slide 28
HTTP Overview (Lab 1)

1. User types in a URL.
2. Browser establishes connection with server.
3. Browser requests the corresponding data.
4. Server responds with the requested data.
5. Browser renders the response, fetches other objects, and closes the connection.

It’s a document retrieval system, where documents point to (link to) each other, forming a “web”.
Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

   `telnet demo.cs.swarthmore.edu 80`

   Opens TCP connection to port 80 (default HTTP server port) at example server. Anything typed is sent to server on port 80 at demo.cs.swarthmore.edu

2. Type in a GET HTTP request:

   ```
   GET / HTTP/1.1
   Host: demo.cs.swarthmore.edu
   ```

   (Hit carriage return twice) This is a minimal, but complete, GET request to the HTTP server.

3. Look at response message sent by HTTP server!
Example

$ telnet demo.cs.swarthmore.edu 80
Trying 130.58.68.26...
Connected to demo.cs.swarthmore.edu.
Escape character is '^]'.
GET / HTTP/1.1
Host: demo.cs.swarthmore.edu

HTTP/1.1 200 OK
Vary: Accept-Encoding
Content-Type: text/html
Accept-Ranges: bytes
ETag: "316912886"
Last-Modified: Wed, 04 Jan 2017 17:47:31 GMT
Content-Length: 1062
Date: Wed, 05 Sep 2018 17:27:34 GMT
Server: lighttpd/1.4.35

<html><head><title>Demo Server</title></head>
<body>
.....
</body>
</html>
Example

$ telnet demo.cs.swarthmore.edu 80
Trying 130.58.68.26...
Connected to demo.cs.swarthmore.edu.
Escape character is '^[].
GET / HTTP/1.1
Host: demo.cs.swarthmore.edu

HTTP/1.1 200 OK
Vary: Accept-Encoding
Content-Type: text/html
Accept-Ranges: bytes
ETag: "316912886"
Last-Modified: Wed, 04 Jan 2017 17:47:31 GMT
Content-Length: 1062
Date: Wed, 05 Sep 2018 17:27:34 GMT
Server: lighttpd/1.4.35
Example

$ telnet demo.cs.swarthmore.edu 80
Trying 130.58.68.26...
Connected to demo.cs.swarthmore.edu.
Escape character is '^]'.
GET / HTTP/1.1
Host: demo.cs.swarthmore.edu

<html><head><title>Demo Server</title></head>
<body>
....
</body>
</html>

Response headers

Response body
(This is what you should be saving in lab 1.)
HTTP request message

- two types of HTTP messages: request, response
- HTTP request message: ASCII (human-readable format)

```
GET /index.html HTTP/1.1
Host: web.cs.swarthmore.edu
User-Agent: Firefox/3.6.10
Accept: text/html,application/xhtml+xml
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7
Keep-Alive: 115
Connection: keep-alive
```

Request line (GET, POST, HEAD, etc. commands)
Why do we have these \r\n (CRLF) things all over the place?

A. They’re generated when the user hits ‘enter’.
B. They signal the end of a field or section.
C. They’re important for some other reason.
D. They’re an unnecessary protocol artifact.

GET /index.html HTTP/1.1\r\nHost: web.cs.swarthmore.edu\r\nUser-Agent: Firefox/3.6.10\r\nAccept: text/html,application/xhtml+xml\r\nAccept-Language: en-us,en;q=0.5\r\nAccept-Encoding: gzip,deflate\r\nAccept-Charset: ISO-8859-1,utf-8;q=0.7\r\nKeep-Alive: 115\r\nConnection: keep-alive\r\n\r\n
How else might we delineate messages?

A. There’s not much else we can do.

B. Force all messages to be the same size.

C. Send the message size prior to the message.

D. Some other way (discuss).
HTTP is all text…

• Makes the **protocol simple**
  – Easy to **delineate** message (\r\n)
  – (Relatively) human-readable
  – No worries about encoding or formatting data
  – Variable length data

• **Not the most efficient**
  – Many protocols use binary fields
    • Sending “12345678” as a string is 8 bytes
    • As an integer, 12345678 needs only 4 bytes
  – The headers may come in any order
  – Requires string parsing / processing

Slide 37
Wireshark

Filter: http

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>14:211168</td>
<td>130.58.68.164</td>
<td>130.58.68.137</td>
<td>HTTP</td>
<td></td>
<td>68 GET /-kwebb/ HTTP/1.1</td>
</tr>
<tr>
<td>16</td>
<td>14:268895</td>
<td>130.58.68.137</td>
<td>130.58.68.164</td>
<td>HTTP</td>
<td>5447</td>
<td>1 200 OK (text/html)</td>
</tr>
</tbody>
</table>

Frame 16: 5447 bytes on wire (43576 bits), 5447 bytes captured (43576 bits)
Internet Protocol Version 4, Src: 130.58.68.137 (130.58.68.137), Dst: 130.58.68.164 (130.58.68.164)
Transmission Control Protocol, Src Port: http (80), Dst Port: 35736 (35736), Seq: 1, Ack: 55, Len: 5381

Hypertext Transfer Protocol

HTTP/1.1 200 OK

[Expert Info (Chat/Sequence): HTTP/1.1 200 OK]
  Request Version: HTTP/1.1

Status Code: 200
  Response Phrase: OK
  Date: Mon, 02 Sep 2013 20:10:28 GMT
  Server: Apache/2.2.22 (Ubuntu)
  Last-Modified: Sat, 31 Aug 2013 20:44:44 GMT
  ETag: "c3f7c-1401-4e5446606210"
  Accept-Ranges: bytes

  Content-Length: 5121
  [Content length: 5121]
  Vary: Accept-Encoding
  Content-Type: text/html

Line-based text data: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>

0040  fe  fc  48 54  54 2f 20 32 30 30 30 30 30 30 20 4f  
0050  6e 2c  2e 0d 0a 32 30 33 31 32 30 33 31 33 30 33 30 
0060  33 30 33 31 33 30 33 31 33 30 33 31 33 30 33 31 33  
0072  30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 
0080  33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33  
0090  30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30  
0090  33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33  
0090  30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30  
0090  33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33  
0090  30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30  
0090  33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33  
0090  30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30  
0090  33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33  
0090  30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30  
0090  33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33  
0090  30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30  
0090  33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33  
0090  30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30  
0090  33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33  
0090  30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30  
0090  33 30 33 30 33 30 33 30 33 30 33 30 33 30 33 30 33  

HTTP Response Status Code (200): Packets: 20 Displayed: 2 Marked: 0 Load time: 0:00.000

Request Method Types ("verbs")

HTTP/1.0 (1996):
- GET
  - Requests page.
- POST
  - Uploads user response to a form.
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1 (1997 & 1999):
- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field
- TRACE, OPTIONS, CONNECT, PATCH
HTTP/1.0 (1996):

- **GET**
  - Requests page.

- **POST**
  - Uploads user response to a form.

- **HEAD**
  - Asks server to leave requested object out of response

HTTP/1.1 (1997 & 1999):

- **GET**, **POST**, **HEAD**

- **PUT**
  - Uploads file in entity body to path specified in URL field

- **DELETE**
  - Deletes file specified in the URL field

- **TRACE**, **OPTIONS**, **CONNECT**, **PATCH**

- (+) Persistent connections
Request Method Types ("verbs")

HTTP/1.0 (1996):
- GET
  - Requests page.
- POST
  - Uploads user response to a form.
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1 (1997 & 1999):
- **GET, POST, HEAD**
- **PUT**
  - uploads file in entity body to path specified in URL field
- **DELETE**
  - deletes file specified in the URL field
- **TRACE, OPTIONS, CONNECT, PATCH**
- **(+) Persistent connections**
Uploading form input

**GET (in-URL) method:**
- uses GET method
- input is uploaded in URL field of request line:
  ```
  www.somesite.com/animalsearch?monkeys&banana
  ```

**POST method:**
- web page often includes form input
- input is uploaded to server in request entity body
GET vs. POST

• GET can be used for idempotent requests
  – Idempotence: an operation can be applied multiple times without changing the result (the final state is the same)
GET vs. POST

• GET can be used for idempotent requests
  – Idempotence: an operation can be applied multiple times without changing the result (the final state is the same)

How many of the following operations are idempotent?

I. Incrementing a variable
II. Assigning a value to a variable
III. Allocating memory
IV. Compiling a program

A. None of them
B. One of them
C. Two of them
D. Three of them
E. All of them
GET vs. POST

• GET can be used for idempotent requests.
  – Idempotence: an operation can be applied multiple times without changing the result (the final state is the same)

• POST should be when...
  – A request changes the state of the server or DB
  – Sending a request twice would be harmful
    • (Some) browsers warn about sending multiple post requests
  – Users are inputting non-ascii characters
  – Input may be very large
  – You want to hide how the form works/user input
When might you use GET vs. POST?

<table>
<thead>
<tr>
<th>GET</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.  Forum post</td>
<td>Search terms, Pizza order</td>
</tr>
<tr>
<td>B.  Search terms, Pizza order</td>
<td>Forum post</td>
</tr>
<tr>
<td>C.  Search terms</td>
<td>Forum post, Pizza order</td>
</tr>
<tr>
<td>D.  Forum post, Search terms, Pizza Order</td>
<td></td>
</tr>
<tr>
<td>E.</td>
<td>Forum post, Search terms, Pizza Order</td>
</tr>
</tbody>
</table>
“REST” APIs

HTTP/1.0 (1996):

• GET
  – Requests page.

• POST
  – Uploads user response to a form.

• HEAD
  – asks server to leave requested object out of response

HTTP/1.1 (1997 & 1999):

• GET, POST, HEAD

• PUT
  – uploads file in entity body to path specified in URL field

• DELETE
  – deletes file specified in the URL field

• TRACE, OPTIONS, CONNECT, PATCH

• (+) Persistent connections
HTTP response message

HTTP/1.1 200 OK
Date: Sun, 26 Sep 2010 20:09:20 GMT
Server: Apache/2.0.52 (CentOS)
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT
ETag: "17dc6-a5c-bf716880"
Accept-Ranges: bytes
Content-Length: 2652
Keep-Alive: timeout=10, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1

data, e.g., requested HTML file

data data data data data data data ...
HTTP response status codes

- Status code appears in first line of server-to-client response message.
- Some common response codes:

  200 OK
  - Request succeeded, requested object later in this msg

  301 Moved Permanently
  - Requested object moved, new location specified later in this msg
    (Location:)

  400 Bad Request
  - Request msg not understood by server

  403 Forbidden
  - You don’t have permission to read the object

  404 Not Found
  - Requested document not found on this server

  505 HTTP Version Not Supported
HTTP response status codes

- Status code appears in first line of server-to-client response message.
- Many others too. Search “list of HTTP status codes”

420 Enhance Your Calm (twitter)
  - Slow down, you’re being rate limited

451 Unavailable for Legal Reasons
  - Censorship?

418 I’m a Teapot
  - Response from a teapot requested to brew a beverage (announced Apr 1)
State(less)

(XKCD #869, “Server Attention Span”)

Slide 51
State(less)

- Original web: simple document retrieval
- Server is not required to keep state between connections (often it might want to though)
- Client is not required to identify itself (server might refuse to talk otherwise though)
User-server state: cookies

Many web sites use cookies

*Four components:*

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

**Example:**

- Susan always accesses the Internet from her PC
- She visits specific e-commerce site for the first time
- When initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID
Cookies: keeping “state” (cont.)

one week later:

usual http request msg

usual http response msg

usual http request msg

cookie: 1678

usual http response msg

Amazon server creates ID 1678 for user

create entry

backend database

access

cookie-specific action
Cookies (continued)

What cookies can be used for:
• authorization
• shopping carts
• recommendations
• user session state (Web e-mail)

How to keep “state”:
• protocol endpoints: maintain state at sender/receiver over multiple transactions
• cookies: http messages carry state
Cookies: Teaching the Controversy

• Cookies permit sites to learn a lot about you

• You may supply name and e-mail to sites (and more!)

• 3rd party cookies (from ad networks, etc) can follow you across multiple sites.
  – Ever visit a website, and the next day ALL your ads are from them?

• You COULD turn them off
  – But good luck doing anything on the internet!
Next class

- HTTP Performance
- Persistent vs. non-persistent connections