The World Wide Web

TELE3118 lecture notes
© by Tim Moors, May 23, 2011
WWW.
Lecture outline

HyperText Transfer Protocol (HTTP)
• messages
  ◦ requests
    • methods
  ◦ responses
• state management
  ◦ persistent and non-persistent HTTP
  ◦ Cookies
• Replicating content
  ◦ Caches
  ◦ Content Distribution Networks
Review(?) of terms

“Web site” (e.g. www.example.com) consists of one or more web “pages” (e.g. index.html = default when no page specified) that often link to “objects” (e.g. images) not necessarily from same server (e.g. Google ads or analytics)

Uniform Resource Locators (URLs) locate objects

http URL = "http:" "//" host [ "":" port ] [ abs_path [ ""?"" query ]]


URI = similar but broader term

“Web browsers” (e.g. Firefox/Internet Explorer) provide the user interface to web sites, that are served from a “web server” (e.g. Apache software running on a computer)

HyperText Transfer Protocol (HTTP) used between browser & web server

May also interact with other types of servers, e.g. mailto: telnet:

Web server may interface with applications that respond to queries, e.g.

- cgi-bin interface to apps on web server, e.g. rfc-editor.org search
- Interface to other back-end server machines, e.g. databases
Why http://www.example.com?

“The Web’s Inventor Regrets One Small Thing”: //
Protocol: HTTP

Full name: HyperText Transfer Protocol
Purpose: Transfer web objects
Layer: Application
Uses: TCP

Identified by: port number 80 (mainly)
Also: 443 for HTTPS = HTTP over SSL/TLS;
     others include 8008 and 8080

Standards: RFC 1945 (v1.0), 2616 (v1.1)
HTTP messages

Contain
• header (in text, with CRLF framing), and
• body (optional & not necessarily text)

Client sends requests to server which responds.
• Server is stateless: Forgets request after responding.

generic-message = start-line
   *(message-header CRLF)
   CRLF
   [ message-body ]
start-line   = Request-Line | Status-Line
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
Status-Line  = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

e.g.
Client sends:
GET / HTTP/1.1 CRLF
Host: www.example.com CRLF
CRLF

Server responds like:
HTTP/1.1 200 OK CRLF
CRLF
<HTML>CRLF
<HEAD>CRLF
<TITLE>Example Web Page</TITLE>CRLF

CRLF = Carriage Return Line Feed
= ASCII codes 13 & 10
made visible here as blue text
Real HTTP messages

GET / HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.9.1.3) Gecko/20090824 Firefox/3.5.3 (.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://subjects.ee.unsw.edu.au/tele3118/

HTTP/1.1 200 OK
Date: Wed, 23 Sep 2009 00:37:50 GMT
Server: Apache/2.2.3 (Red Hat)
Last-Modified: Tue, 15 Nov 2005 13:24:10
ETag: "b80f4-1b6-80bfd280"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8

<html>
<head>
  <title>Example Web Page</title>
</head>
<body>
<p>You have reached this web page by typing &quot;example.com&quot;, &quot;example.net&quot;, or &quot;example.org&quot; into your web browser.</p>
<p>These domain names are reserved for use in documentation and are not available for registration. See &lt;a href="http://www.rfc-editor.org/rfc/rfc2606.txt"&gt;RFC 2606&lt;/a&gt;, Section 3.</p>
</body>
</html>
HTTP methods

Main methods:
• **GET**: Mainly to read from a server
  o URL gives info to the server, e.g. to supply search terms
• **POST**: Send info to a server
  o Especially for bulky files that are too large for GET URL
• **HEAD**: Returns headers as per GET, but not the object. e.g. to check whether object:
  o Still exists: For checking links
  o Has been updated
• **PUT/DELETE** (HTTP/1.1): Upload a new object to a server

Others include OPTIONS, TRACE, PROPFIND, (UN)LOCK, ...

Typing note: They are case sensitive => 501 “Method not implemented” error if you don’t capitalise
Status codes

= Numeric code + (arbitrary) textual description
Header may augment, e.g. Location: for 301

1xx Informational - Request received, continuing process (rare)

2xx Success, e.g. 200 OK

3xx Redirection, e.g. 301 Moved Permanently (to Location:)
Common, e.g. subjects.ee.unsw.edu.au/tele3118 -> tele3118/
Browsers limit # of redirections in case of loop,
e.g. Firefox: network.http.redirection-limit

304 Not Modified

4xx Client Error, e.g. 404 Not Found

401 Unauthorized (WWW-Authenticate: then Authorization:)

5xx Server Error, e.g. 500 Internal Server Error (e.g. back-end DB down)
503 Service Unavailable (e.g. due to maintenance)

SMTP&FTP use similar hierarchical numeric codes
Header lines

Over 50 standardised + proprietary lines
We’ll focus on common ones & those in example:

- Host:
- Referer:
- User-Agent:/Server:
- Accept*

State information:
- WWW-Authenticate:
  & Authorization:
- Cookie: & Set-Cookie:
- Connection: & Keep-Alive:

Replicas (caching)
- Date: & Last-Modified:

See also A. Wooster: “Fun With HTTP Headers”,
Host: & hosting

(Context: IP & TCP already deliver request to web server process on right machine.)

One machine may serve content for multiple web sites:

- **Web hosting service**: Web hosting business may use one IP address (e.g. server) to provide multiple sites
  - Organisations outsource hosting to specialist businesses that ensure availability, performance, protection against flash crowds and Denial Of Service attacks

- **Configured proxy** (see caching) may intermediate between clients & multiple origin servers

- **Multiple names for one site**, e.g. example.com and www.example.com

**Host**: allows client to specify which host on a service they wish to access
Why http://www.example.com?

- "www" technically redundant because implied by TCP port 80
- Created recognisable brand with 1990s users
- Search engines rank sites (in part) by number of inbound links => 1 identity better than 2
- http://example.com should redirect (HTTP status 301) to http://www.example.com

Configuration & troubleshooting:
- DNS should be configured for both (www.)example.com
- Client side "Domain guessing" (see DNS lecture) may try www.example.com after no result for example.com
- Web server should be configured for both (&redirect example.com)
- http://example.com may not work if you're lazy/efficient & omit "www"
Referer:† & analytics

Would like to track client’s path through web:
• How did they reach this web site?
• What path do they take through this site?
  => Optimise site design site (e.g. minimise steps to purchase, direct clients on desired path)
• Which pages have stale links to objects that are Not Found?

Referer: = URI of page containing link to object being requested
Referer: line omitted if object not requested via link, e.g. URL entered in browser address bar
Web site analytics businesses/software interprets referral paths
  ◦ May involve embedding links in pages
  ◦ e.g. Google Analytics

† The HTTP spec includes a typo (“Referer” not “Referrer”) which implementations must now perpetuate
Extract of typical access_log

Format: Client [date] Request Status Bytes Referer User-Agent discuss

"Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.5.30729; .NET CLR 3.0.30618)"
149.171.236.136 - - [26/Aug/2009:14:18:00 +1000] "GET /~tim/zoo/5d8baf3e.pcap HTTP/1.1" 200 10628
"http://uluru.ee.unsw.edu.au/~tim/zoo/index.html" "Mozilla..."
149.171.236.136 - - [26/Aug/2009:14:18:01 +1000] "GET /~tim/zoo/5d8baf3e.pcap HTTP/1.1" 206 10628
"http://uluru.ee.unsw.edu.au/~tim/zoo/" "Mozilla..."
"http://uluru.ee.unsw.edu.au/~tim/zoo/index.html" "Mozilla..."
"Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"
"Mozilla..."
"Mozilla..."
User-Agent: & Server:

Varied implementations of browsers/servers because of
• Choices made in implementing complicated standards (ambiguous or misinterpreted)
• Deliberate product differentiation, e.g. Microsoft-proprietary features in Internet Explorer

User-Agent: / Server: allow ends to identify/patch each other’s quirks

User-Agent: also identifies automated (non-browser) access, e.g. search engine crawler
  • Server may customise service, e.g.
    • Allow free search engine access, but paid user access
    • Map IP address to geographic location & customise content
Accept*: 

Accept*: indicates formats acceptable to client
  - Except Accept-Ranges: which indicates if server accepts requests for specific ranges of bytes in an object

- **Accept**: lists (and prioritises) acceptable formats, e.g.
  - JPEG, GIF, PNG images?
  - HTML or plain text?

- **Accept-Language**: Language preferences, e.g. (British) English, French, etc

- **Accept-Encoding**: compression formats, e.g. compress, gzip
  - Compression saves transmission delay & cost for client & server, at (small) expense of client decompression delay. (Server often stores objects in compressed form.)
HTTP is stateless

Server forgets request after serving it =>
√ Server is
  √ Scalable: Server doesn't have to remember clients
  √ Fault tolerant: Successive requests from one client can be served by different servers
× Client must repeatedly send context with each request (e.g. Authorization: or Cookie:)

HTTP originally (v1.0) extended statelessness to TCP connections: One request/response interaction per connection.
  e.g. consider reading page that references 2 images on that server. Page fits in 1MSS, images need 3MSS each
Non-persistent HTTP

Serial

Parallel connections
WWW-Authenticate: &
Authorization:

- To limit service to particular clients, server replies to request with
  ```
  HTTP/1.1 401 Authorization Required
  WWW-Authenticate: Basic realm="Hint of context for user"
  ```
- Client resends request with
  ```
  Authorization: Basic egwklyncreptdpsswd
  ```
- For stronger authentication, use Transport Layer Security (TLS), e.g. “HTTPS” = HTTP over TLS.
Persistent connections

HTTP 1.1 makes connections persistent (by default): Server retains connection some time after responding, in case client has another request.

=> Lower TCP overheads:

• **Time**: Lower latency: No delays from
  • Handshaking repeatedly
    • Even better: Requests/responses can be **pipelined**: Multiple flow over connection while waiting for other end. Like sliding window vs stop-and-wait
  • Ramping up speed with **Slow Start**
    • Better congestion control: no big bursts as many connections open & insert 1 segment into net

• **Bandwidth**: Less connection mgt packets

• **Memory & processing**: Less sockets for client/server to handle
Persistent HTTP

Serial

Pipelined
Connection: & Keep-Alive:

Give control of HTTP/1.1 persistence

• "Connection: Close": Indicates desire not to persist (??Might closing TCP connection disrupt flow??)

• "Connection: keep-alive": Desire to control persistence, e.g. with "Keep-Alive: 300" Persist for 300 sec, despite server default (15 sec for Apache)
Outline
State info for web access

- **User identity**
  - ✔ “shopping basket” of selected goods
  - ✔ Site personalisation, values for form fields etc
  - ✔ registration – have they paid for the service, do we know their email address, etc
  - ✗ tracking,
    - e.g. newspaper: what advertisements do people who read this sort of article read? → tune advertising for higher impact
    - can raise privacy concerns

- **Load sharing** – direct request to preferred server in server farm
Cookies†

Process:
1. Client requests information from server
   • Server responds, *including state information*
2. Client stores state information, associates it with server
3. Client includes state information with subsequent requests to server

Bottom line: Server doesn’t have to store state info.

† So named because like an edible cookie that leaves a trail of crumbs, electronic cookies record a trail of past actions, i.e. record state info. aka “handle”, “transaction ID”, or “token”
Cookies: keeping “state” (cont.)

client

usual http request
usual http response + Set-cookie: 1678

server

server creates ID 1678 for user

one week later:

Cookie file
amazon: 1678
ebay: 8734

Cookie file
amazon: 1678
ebay: 8734

Cookie file
amazon: 1678
ebay: 8734

usual http request msg cookie: 1678
usual http response msg

usual http request msg cookie: 1678
usual http response msg

cookie-specific action

cookie-specific action

entry in backend database

access

access

Slide from Kurose and Ross
Cookies when buying copies of Stevens

POST /shopping/BasketAdd.asp HTTP/1.1

... Cookie: BIGipServerdymocks-http=1038178763.20480.0000; ASPSESSIONIDQQQGGWFC=DHHLDGMCJCIFHDKNOCFEEDHJ

Data (47 bytes)
0000 50 72 6f 64 75 63 74 5f 49 44 3d 30 32 30 31 36
Product_ID=02016
0010 33 34 39 53 26 49 4d 41 47 45 31 2e 78 3d 32
ISBN 0201634953 ...

HTTP/1.1 100 Continue

Set-Cookie: BIGipServerdymocks-http=1038178763.20480.0000;

HTTP/1.1 302 Object moved

Set-Cookie: MSCSProfile=61E4CECF7275066FD87B9817DA5865CB01E8624F84600C0D...

POST /Shopping/BasketAdd.asp HTTP/1.1

Cookie: BIGipServerdymocks-http=1038178763.20480.0000; ASPSESSIONIDQQQGGWFC=DHHLDGMCJCIFHDKNOCFEEDHJ; MSCSProfile=61E4CECF7275066FD87B9817DA5865CB01E8624F84600C0D...

Data (33 bytes)
0000 50 72 6f 64 75 63 74 5f 49 44 3d 30 32 30 31 36
Product_ID=02016
0010 33 34 33 53 26 49 4d 41 47 45 31 2e 78 3d 32
ISBN 020163354X ...
Inefficient to repeatedly send from origin server to client:

- Clients could reuse previously downloaded content => **caching**
  - Browser can reuse objects downloaded earlier (e.g. images)
  - Proxy can share objects between multiple clients
  - Need to check currency of cached object
    - **Server reports Last-modified**: Cache makes GET request conditional by adding `If-Modified-Since:`. Server responds with status 200 & object (If modified), or 304 Not Modified (incurs delay, but saves bandwidth)

- **Content Distribution Networks** replicate content on servers closer to clients & direct client to closest server, e.g. www.abc.net.au

Figure based on Kurose & Ross
DIY: Distributing ABC content

- DNS aliases: www.abc.net.au → www.abc.net.au.edgesuite.net → a1632.g.akamai.net
- Client location affects address DNS gives for a1632...:
  - UNSW: 60.254.186.75 (whois reports owned by Akamai)
  - Optus: 198.142.23.104 (whois reports owned by Optus)
- Routes are shorter (=> better) for supplied addresses:

<table>
<thead>
<tr>
<th>From</th>
<th>UNSW</th>
<th>Optus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ms</td>
<td>192.168.1.1</td>
<td>0.251 ms eebu4s1.uwn.unsw.EDU.AU.92.171.149.i</td>
</tr>
<tr>
<td>7 ms</td>
<td>10.65.0.1</td>
<td>0.311 ms te-1-3.ombcr1.gw.unsw.edu.au</td>
</tr>
<tr>
<td>9 ms</td>
<td>riv3-ge2-1.gw.optusnet.com.au</td>
<td>0.349 ms te-7-2.unswbrl1.gw.unsw.edu.au</td>
</tr>
<tr>
<td>10 ms</td>
<td>mas1-ge13-0-0-811.gw.optusnet.com.au</td>
<td>0.525 ms gigabitethernet1.erl.unsw.cpe.aarnet.net.au</td>
</tr>
<tr>
<td>16 ms</td>
<td>mas4-unk8-1.gw.optusnet.com.au</td>
<td>0.698 ms ge-4-1-0.bbl1.a.sydney.aarnet.net.au</td>
</tr>
<tr>
<td>7 ms</td>
<td>a198-142-23-104.deploy.akamaitechnologies.com</td>
<td>0.892 ms tenge-2-1.pe2.c.sydney.aarnet.net.au</td>
</tr>
<tr>
<td></td>
<td>0.941 ms 202.167.228.102</td>
<td>0.928 ms 60.254.186.75</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supplied</th>
<th>Optus</th>
<th>UNSW</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ms</td>
<td>192.168.1.1</td>
<td>0.255 ms eebu4s1.uwn.unsw.EDU.AU.92.171.149.i</td>
</tr>
<tr>
<td>13 ms</td>
<td>10.65.0.1</td>
<td>0.308 ms te-1-3.ombcr1.gw.unsw.edu.au</td>
</tr>
<tr>
<td>7 ms</td>
<td>riv4-ge2-1.gw.optusnet.com.au</td>
<td>0.377 ms te-7-2.unswbrl1.gw.unsw.edu.au</td>
</tr>
<tr>
<td>8 ms</td>
<td>mas2-ge14-0-0-821.gw.optusnet.com.au</td>
<td>0.520 ms gigabitethernet1.erl.unsw.cpe.aarnet.net.au</td>
</tr>
<tr>
<td>7 ms</td>
<td>mas5-unk8-1.gw.optusnet.com.au</td>
<td>0.658 ms ge-4-1-0.bbl1.a.sydney.aarnet.net.au</td>
</tr>
<tr>
<td>8 ms</td>
<td>xe-3-1-0.sd42.optus.net.au</td>
<td>0.705 ms tenge-2-1.pe2.a.sydney.aarnet.net.au</td>
</tr>
<tr>
<td>11 ms</td>
<td>ge2-0-0-3926.ug2.optus.net.au</td>
<td>22.355 ms vlan843.o2gsc76f03.optus.net.au</td>
</tr>
<tr>
<td></td>
<td>Request timed out.</td>
<td>32.061 ms sun4-te9-2.gw.optusnet.com.au</td>
</tr>
<tr>
<td>13 ms</td>
<td>61.88.241.231</td>
<td>31.531 ms sun1-ge14-0.gw.optusnet.com.au</td>
</tr>
<tr>
<td>9 ms</td>
<td>internode.22rrc76f000.optus.net.au</td>
<td>29.562 ms mas1-ge13-0-0-811.gw.optusnet.com.au.a</td>
</tr>
<tr>
<td>11 ms</td>
<td>gil-47-133.cor2.sydney.internode.on.net</td>
<td>29.182 ms mas4-unk8-1.gw.optusnet.com.au</td>
</tr>
<tr>
<td>9 ms</td>
<td>150.101.197.205</td>
<td>29.202 ms a198-142-23-104.deploy.akamaitechnologies.com</td>
</tr>
</tbody>
</table>

* Data massaging: Measurements from Optus used Windows, from UNSW used Linux => differences in accuracy (Linux used -I option to use ICMP to match Windows)

Delay is for last of 3 probes to each hop. Windows often reported higher delay (e.g. 25ms) for 1st probe for each hop. Observed Oct. 2009.
Lecture summary

- Web objects transferred using HTTP
- Textual protocol
- Stateless: Server forgets request after responding
- Persistent connections minimise TCP overheads
- Client stores state & sends with each request, e.g. Authorization: & Cookies
Links

- Pipelining in HTTP just as sliding window provides better performance than stop-and-wait error control.
- HTTP uses spaces to frame URL (and URL uses characters such as : / and ? To frame parts) => % encoding to escape those characters if they appear within parts
- `Content-types` come from email MIME
- See DNS lecture for differences between `www.example.com` and `example.com`