Dynamic Web Development

Lecture 2 – The HTTP Protocol

Hypertext Transfer Protocol

A protocol for conveying hypertext pages across a network.

Invented by Tim Berners-Lee.

Development coordinated by W3C and IETF.

Latest standard is: HTTP v2.0,
defined in: RFC 7540 (May 2015)

A request / response protocol between clients and servers.

The client making the HTTP request is called the user agent. (Browser, spider, robot)

HTTP does not require TCP/IP. It can be used on top of any protocol which provides a reliable transport mechanism.

The HTTP Request

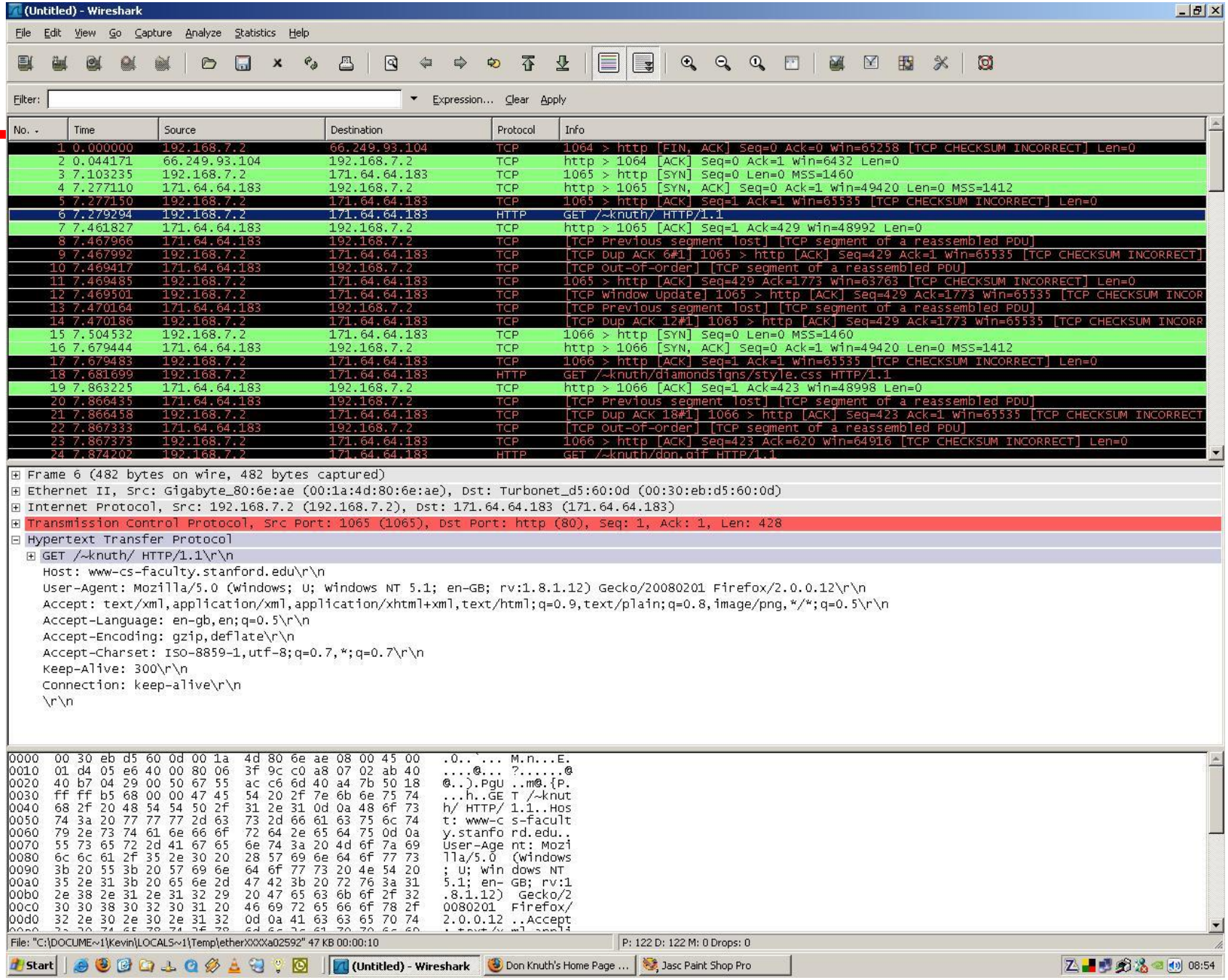
User types a URL into their browser. eg
http://www.example.com/index.html

An HTTP client initiates a request by establishing a TCP connection to a specified port on the host (Port 80 is the default).

Server listening on that port waits for an HTTP request message.

Browser sends the following request:

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Linux; X11; UTF-8)
Accept-Charset: UTF-8
```


(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: Expression... Clear Apply
No. Time Source Destination Protocol Info
1 0.000000 192.168.7.2 66.249.93.104 TCP 1064 > http [FIN, ACK] Seq=0 Ack=0 Win=65258 [TCP CHECKSUM INCORRECT] Len=0
2 0.044171 66.249.93.104 192.168.7.2 TCP http > 1064 [ACK] Seq=0 Ack=1 Win=6432 Len=0
3 7.103235 192.168.7.2 171.64.64.183 TCP 1065 > http [SYN] Seq=0 Len=0 MSS=1460
4 7.277110 171.64.64.183 192.168.7.2 TCP http > 1065 [SYN, ACK] Seq=0 Ack=1 Win=49420 Len=0 MSS=1412
5 7.277150 192.168.7.2 171.64.64.183 TCP 1065 > http [ACK] Seq=1 Ack=1 Win=65535 [TCP CHECKSUM INCORRECT] Len=0
6 7.279294 192.168.7.2 171.64.64.183 HTTP GET /~knuth/ HTTP/1.1
7 7.461827 171.64.64.183 192.168.7.2 TCP http > 1065 [ACK] Seq=1 Ack=429 Win=48992 Len=0
8 7.467966 171.64.64.183 192.168.7.2 TCP [TCP Previous segment lost] [TCP segment of a reassembled PDU]
9 7.467992 192.168.7.2 171.64.64.183 TCP [TCP Dup ACK 6#1] 1065 > http [ACK] Seq=429 Ack=1 Win=65535 [TCP CHECKSUM INCORRECT]
10 7.469417 171.64.64.183 192.168.7.2 TCP [TCP Out-Of-Order] [TCP segment of a reassembled PDU]
11 7.469485 192.168.7.2 171.64.64.183 TCP 1065 > http [ACK] Seq=429 Ack=1773 Win=63763 [TCP CHECKSUM INCORRECT] Len=0
12 7.469501 192.168.7.2 171.64.64.183 TCP [TCP Window Update] 1065 > http [ACK] Seq=429 Ack=1773 Win=65535 [TCP CHECKSUM INCOR
13 7.470164 171.64.64.183 192.168.7.2 TCP [TCP Previous segment lost] [TCP segment of a reassembled PDU]
14 7.470186 192.168.7.2 171.64.64.183 TCP [TCP Dup ACK 12#1] 1065 > http [ACK] Seq=429 Ack=1773 Win=65535 [TCP CHECKSUM INCORR
15 7.504532 192.168.7.2 171.64.64.183 TCP 1066 > http [SYN] Seq=0 Len=0 MSS=1460
16 7.679444 171.64.64.183 192.168.7.2 TCP http > 1066 [SYN, ACK] Seq=0 Ack=1 Win=49420 Len=0 MSS=1412
17 7.679483 192.168.7.2 171.64.64.183 TCP 1066 > http [ACK] Seq=1 Ack=1 Win=65535 [TCP CHECKSUM INCORRECT] Len=0
18 7.681699 192.168.7.2 171.64.64.183 HTTP GET /~knuth/diamondsigns/style.css HTTP/1.1
19 7.863225 171.64.64.183 192.168.7.2 TCP http > 1066 [ACK] Seq=1 Ack=423 Win=48998 Len=0
20 7.866435 171.64.64.183 192.168.7.2 TCP [TCP Previous segment lost] [TCP segment of a reassembled PDU]
21 7.866458 192.168.7.2 171.64.64.183 TCP [TCP Dup ACK 18#1] 1066 > http [ACK] Seq=423 Ack=1 Win=65535 [TCP CHECKSUM INCORRECT
22 7.867333 171.64.64.183 192.168.7.2 TCP [TCP Out-Of-Order] [TCP segment of a reassembled PDU]
23 7.867373 192.168.7.2 171.64.64.183 TCP 1066 > http [ACK] Seq=423 Ack=620 Win=64916 [TCP CHECKSUM INCORRECT] Len=0
24 7.874202 192.168.7.2 171.64.64.183 HTTP GET /~knuth/don.gif HTTP/1.1
Frame 6 (482 bytes on wire, 482 bytes captured)
Ethernet II, Src: Gigabyte_80:6e:ae (00:1a:4d:80:6e:ae), Dst: Turbonet_d5:60:0d (00:30:eb:d5:60:0d)
Internet Protocol, Src: 192.168.7.2 (192.168.7.2), Dst: 171.64.64.183 (171.64.64.183)
Transmission Control Protocol, Src Port: 1065 (1065), Dst Port: http (80), Seq: 1, Ack: 1, Len: 428
Hypertext Transfer Protocol
GET /~knuth/ HTTP/1.1\r\n
Host: www-cs-faculty.stanford.edu\r\n
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.8.1.12) Gecko/20080201 Firefox/2.0.0.12\r\n
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5\r\n
Accept-Language: en-gb,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 300\r\n
Connection: keep-alive\r\n
\r\n
0000 00 30 eb d5 60 0d 00 1a 4d 80 6e ae 08 00 45 00 .0..`... M.n...E.
0010 01 d4 05 e6 40 00 80 06 3f 9c c0 a8 07 02 ab 40 ....@... ?......@
0020 40 b7 04 29 00 50 67 55 ac c6 6d 40 a4 7b 50 18 @..).PgU ..m@.{P.
0030 ff ff b5 68 00 00 47 45 54 20 2f 7e 6b 6e 75 74 ...h..GE T /~knut
0040 68 2f 20 48 54 54 50 2f 31 2e 31 0d 0a 48 6f 73 h/ HTTP/ 1.1..Hos
0050 74 3a 20 77 77 77 2d 63 73 2d 66 61 63 75 6c 74 t: www-c s-facult
0060 79 2e 73 74 61 6e 66 6f 72 64 2e 65 64 75 0d 0a y.stanfo rd.edu..
0070 55 73 65 72 2d 41 67 65 6e 74 3a 20 4d 6f 7a 69 User-Age nt: Mozi
0080 6c 6c 61 2f 35 2e 30 20 28 57 69 6e 64 6f 77 73 lla/5.0 (Windows
0090 3b 20 55 3b 20 57 69 6e 64 6f 77 73 20 4e 54 20 ; U; Win dows NT
00a0 35 2e 31 3b 20 65 6e 2d 47 42 3b 20 72 76 3a 31 5.1; en- GB; rv:1
00b0 2e 38 2e 31 2e 31 32 29 20 47 65 63 6b 6f 2f 32 .8.1.12) Gecko/2
00c0 30 30 38 30 32 30 31 20 46 69 72 65 66 6f 78 2f 0080201 Firefox/
00d0 32 2e 30 2e 30 2e 31 32 0d 0a 41 63 63 65 70 74 2.0.0.12 ..Accept
File: "C:\DOCUME~1\Kevin\LOCALS~1\Temp\etherXXXXa02592" 47 KB 00:00:10
P: 122 D: 122 M: 0 Drops: 0
Start
(Untitled) - Wireshark
Don Knuth's Home Page ...
Jasc Paint Shop Pro
08:54

The HTTP Response

The server sends back something like:

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
 Server: Apache/1.3.27 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8

<html>
<body>
```

and the rest of the requested resource

HTTP Request Format

Request Line:

```
GET     /index.html     HTTP/1.1
```

Method URI Version No

Followed by a set of headers:

```
Host: www.example.com
User-Agent: Mozilla/5.0 (Linux; X11; UTF-8)
Accept-Language: en
```

Sometimes followed by an optional message body

HTTP Request Methods

GET

- Requests a copy of the resource stored at the specified URI.

POST

- Submits data to be processed (e.g. from an HTML form) to the specified URI. The data is included in the message body.

PUT

- Requests the server to store the data enclosed in the message body at the specified URI.

Some of the less common HTTP request methods are shown next.

Be aware that not all web servers implement all aspects of the HTTP protocol.

HTTP Request Methods

HEAD

- Same as a GET request, but tells the server to just send the response headers, not the message body.

DELETE

- Requests the server to delete the resource stored at the specified URI.

TRACE

- Requests that the server echoes back the received request, so that a client can see what intermediate servers are adding or changing in the request. Used for testing.

OPTIONS

- Returns the HTTP methods that the server supports. This can be used to check the functionality of a web server.

CONNECT

- Requests the server to set up an secure (usually encrypted) connection.

Safe Methods

GET and HEAD are conventionally regarded as safe methods.

They should only be used for data retrieval and should not be used to change the state of the server.

POST, PUT and DELETE can be used to change the state of the server.

Idempotent Methods

GET, HEAD, PUT and DELETE are defined as idempotent methods.

ie Multiple identical requests should have the same effect as a single request.

This allows a user agent to retry idempotent requests without informing the user, which may be useful when connecting to busy servers.

Which makes it more important to not use GET for anything other than document retrieval.

HTTP Response Format

Status Line:

```
HTTP/1.1    200        OK
```

Version Status Code Reason Phrase

Followed by a set of headers

```
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.27 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

and a message body, sometimes called an entity

```
<html>
<body>
and so on
```

HTTP Response Status Codes

1xx Informational

100 Continue
101 Switching Protocols

2xx Success

200 OK
201 Created
202 Accepted
203 Non-Authoritative Information
204 No Content
205 Reset Content
206 Partial Content

HTTP Response Status Codes

3xx Redirection

The client must take additional action to complete the request. This may be done by the user agent without notifying the user.

300	Multiple Choices
301	Moved Permanently
302	Found
303	See other
304	Not Modified
305	Use Proxy
307	Temporary Redirect

HTTP Response Status Codes

4xx Client Error

The request contains bad syntax or cannot be fulfilled. The server response should include a message entity containing an explanation of the situation.

400	Bad Request
401	Unauthorised
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
408	Request Timeout
414	Request URI Too Long

HTTP Response Status Codes

5xx Server Error

The server failed to fulfill an apparently valid request. The server response should contain a message entity containing an explanation of the error situation.

500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Temporarily Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported

Persistent Connections

In HTTP/0.9 and 1.0, the connection is closed after a single request/response pair.

In HTTP/1.1 a keep-alive-mechanism was introduced, where a connection could be reused for more than one request.

Such *persistent connections* reduce lag perceptibly, because the client does not need to re-negotiate the TCP connection after the first request has been sent.

HTTP Session State

HTTP is a stateless protocol.

Each request is treated as an independent transaction, unrelated to any other request.

This forces the use of alternative methods for maintaining a users' state, for example, when a host would like to customize content for a user who has visited before.

The common methods for solving this problem are:

- sending and requesting cookies.
- server side sessions.
- hidden variables (when current page is a form).
- URL encoded parameters (such as /index.php?userid=3).