Web Protocols

Resources
Request and Responses
Error Codes

Hypertext Transfer Protocol

- HTTP is an application/layer protocol on server port 80
- was created in 1993 by Tim Berbers-Lee, a physicist

  "The Hypertext Transfer Protocol (HTTP) is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems." RFC1945
- is one of the simplest protocols
  - Writing handler code is really easy

- has only two phases:
  - Request (Client → Server)
  - Response (Server → Client)
- is Stateless (as originally defined)
- Some REQ/RESP are idempotent, so optimal to cache
  - Every transaction has no memory of previous transactions

Hypertext Transfer Protocol

- HTTP had 4 main versions

v0.9.	Original release and simplest
v1.0	HTTP Sessions introduced as a series of request/response pairs
v1.1	Added partial downloads, better caching
v2	Significant upgrade, introduced in 2014

- Numerous optimisations have been added, but the basic protocol still works
  - Main goal in these optimisations was to improve Performance and reduce Latency
  - Generally only ver. 1.1 and 2 are now used.

Resource Retrieval using HTTP

Retrieval with the HTTP protocol has five main steps, usually implemented by the browser as client:

1. Map the server domain name to an IP address
2. Establish a TCP/IP connection to the web server
3. Transmit a **request** to the server, which includes a request method, and any additional information
4. Receive a **response** from the server, such as HTML text, an image, or other information
5. Server terminates the TCP/IP Connection (if timeout, then client terminates)

HTTP Example Using elnet

Normally telnet ←→telnetd port 23
ssh ←→ sshd port 22
http ←→ httpd port 80

```
% telnet titan.csit.rmit.edu.au 80
Trying 131.170.5.131...
Connected to titan.csit.rmit.edu.au.
Escape character is '^]'.
GET /

HTTP/1.1 200 OK
Date: Fri, 7 Feb 2018 17:35:20 GMT
Server: Apache/2.4.6 (Red Hat Enterprise Linux)
Last-Modified: Thu, 14 Jan 2016 02:26:01 GMT
ETag: "76-5294201ff0a4e"
Accept-Ranges: bytes
Content-Length: 118
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

<html>
<body>
 <iframe name="my_iframe" src="motd.txt" width=800 height=600
   align="centre"></frame>
</body>
</html>

Connection closed by foreign host.
```

← Type a blank Line (Two CRs in a row)
------- No HTTP Request Body --------

HTTP Response Header

← blank line between HTTP Response Header and content

Statelessness of HTTP

HTTP is a ***stateless*** protocol,

- there *no memory involved* in the transaction – all are alike
- so each HTTP transaction between a client and a server is an independent entity. That is, in our example, the request for the document body and the image are *unrelated* transactions.
- Does not use Session-layer state management

- With large numbers of transactions arriving at a web server, statelessness brings efficiency benefits, since a server does not need to track clients, sessions, or histories.

However if state is required, for example if a site uses a shopping cart, then other methods have to be implemented, for example cookies.

Idempotence

- An operation in a server is ***idempotent***, if that operation does not change the state of the server.
- This means:
  - You can do the operation many times and the result is always the same
    - You can change how you do it without affecting the server
    - You can test it without changing what clients see
  - You can assume what the result will be if you did it before (recently).
    - You can cache the operation
    - you can pretend to do it but really just return the result as if you had done it.
  - It allows all forms of virtualisation to be simplified

- Idempotence is an important performance NFA

HTTP Request & Response messages

- Request Header
  - Request line, starting with a Request Method
    - eg GET /path/to/resource
  - Other header lines
  - Blank line to separate header from content
  - Content
    - data to send to server (usually empty for GET)

- Response
  - Response line
    - return protocol + error code (eg HTTP 1.1 200 Ok)
  - Other header lines
  - Blank line to separate header from content
  - Content
    - returned data from server

HTTP Request Methods

- On the request line, there are 10 possible request methods that can be made to a HTTP server:

Command	Action
GET*	Return Document Contents
HEAD*	Return the document header only (Metadata)
PUT*	Replace the document with the following data
POST	Treat the document as a script and send data
OPTIONS*	Return a list of methods that work on this URI
CONNECT	Set up a TCP connection through HTTP (eg TLS)
Others:	CREATE, DELETE, TRACE*, PATCH

* Idempotent methods

HTTP Get Method

URL: http://server.com/path/to/resource?opt1=A&opt2=B

- GET retrieves a resource from the web server.
- GET can include additional information that will help the web server formulate a response.

- Format:

  GET /path/to/resource?options HTTP-version

- Examples

  GET /index.html
  - Get file index.html from the root directory

  GET /index.html HTTP/1.1
  - As above using HTTP/1.1 version

  GET /maps/showmap?lat=39&long=144&zoom=3
  - Get a map, using ?options to determine location and size

HTTP Head Method

URL: http://server.com/path/to/resource?opt1=A&opt2=B

- HEAD does NOT retrieves a resource from the web server.
- Instead it sets the same metadata that a GET would have set, but does not retrieve the resource
- Used for checking whether a server copy has been updated since last download.

- Format:

  HEAD /path/to/resource?options HTTP-version

- Examples

  HEAD /index.html HTTP/1.1

  – Like GET but without download resource

HTTP Put Method

URL: http://server.com/path/to/resource

- PUT sends a resource to the web server.
- typically used for file upload
- has security issues (e.g. XSS)

- Format:

  PUT /path/to/put/resource HTTP-version

- Examples

  PUT /index.html HTTP/1.1

  – As above using HTTP/1.1 version

  PUT /logs/locationdata.dat?lat=39&long=144&id=54642

  – Put log data, using ?options to determine location and ID

HTTP Post Method

URL: http://server.com/path/to/resource/resource?opt1=A&opt2=B

- POST sends data to the web server.
- typically used for data entry forms
- similar to GET except request content is not blank
  - instead it contains the options, that in GET would have been appended to the URL
  - can be compresses and/or encrypted, unlike HTTP headers.

- Format:

  POST /path/to/put/resource HTTP-version
  opt1=A
  opt2=B

- **Example**

  POST /logs/locationdata.dat HTTP/1.1
  lat=39
  long=144
  id=54642
  <blanks>

  - Put log data, using ?options to determine location and ID

Selected HTTP Status Codes

Code	Text	Description
2xx		Success
200	OK	URL found, Data Follows
202	Accepted	Request accepted for later processing
204	No Response	OK, but no data follows
206	Partial Content	GET partially succeeded (eg a PDF page)
3xx		Redirection
301	Moved	The URL has moved permanently
302	Found	The URL has moved temporarily
4xx		Client Errors
400	Bad Request	Syntax Error
401	Unauthorised	Authorisation Error
403	Forbidden	URL can never be retrieved
404	Not found	URL is not here
5xx		Server Errors
500	Internal Error	Something weird happened
501	Not Implemented	Feature not supported

HTTP/1.1

SPDY

Google was pushing SPDY, but now it is deprecated since May 2016

- SPDY ('speedy') acts as a tunnel/filter for HTTP. It:
  - Improves security
    - Usually requires SSL/TLS, but can work without it
  - Reduces page load latency
    - Compresses data stream & headers
      - (GZIP or DEFLATE)
    - Prioritises the data items sent (*Prioritised pipelining*)
      - items not necessarily sent in document order
    - Minimises the number of separate connections
  - Implements Server Push
    - Create an *imagined* (*virtual*) request, then respond to it.

SPDY sits in the Application Layer so it cannot be invisible and browsers need to know about it.

HTTP/2

- HTTP/2 (*Not HTTP 2.0, or HTTP/2.0*)
  - Complete superset of HTTP/1.1 (so fully supports it)
    - High level syntax is the same as HTTP/1.1
    - Low level signalling and transports are different
  - Based on SPDY, so same benefits as it
    - **Content Negotiation** mechanism (HTTP/1.1 /2 or other)
      - ALPN = Application Layer Protocol Negotiation
    - Improve page load latency like SPDY (with minor differences)
      - Header Compression, Server Push, Pipelining, Multiplexing
      - Fix "head of line" blocking issue
  - Encryption (defined for HTTP/HTTPS)
- The HTTP/2 standard is available from:
  - http://www.ietf.org/rfc/rfc7540.txt (2015)

SPDY sits in the Application Layer so it cannot be invisible and browsers need to know about it.

Criticisms of HTTP/2

- Flow Control
  - Handled by TCP as SYN/ACK pairs on separate connections
  - HTTP/2 Duplicates this using stream ID on a single connection
  - Breaks the protocol layering principles of TCP/IP

- Encryption
  - Some in Standards Committee tried to force encryption. Failed.
  - But browsers only use HTTP/2 over HTTPS,
    so de-facto succeeded

- HTTP/2 flow control is explained in detail at
  - http://qnimate.com/what-is-multiplexing-in-http2/#prettyPhoto

QUIC

(Quick UDP Internet Connections) - Google

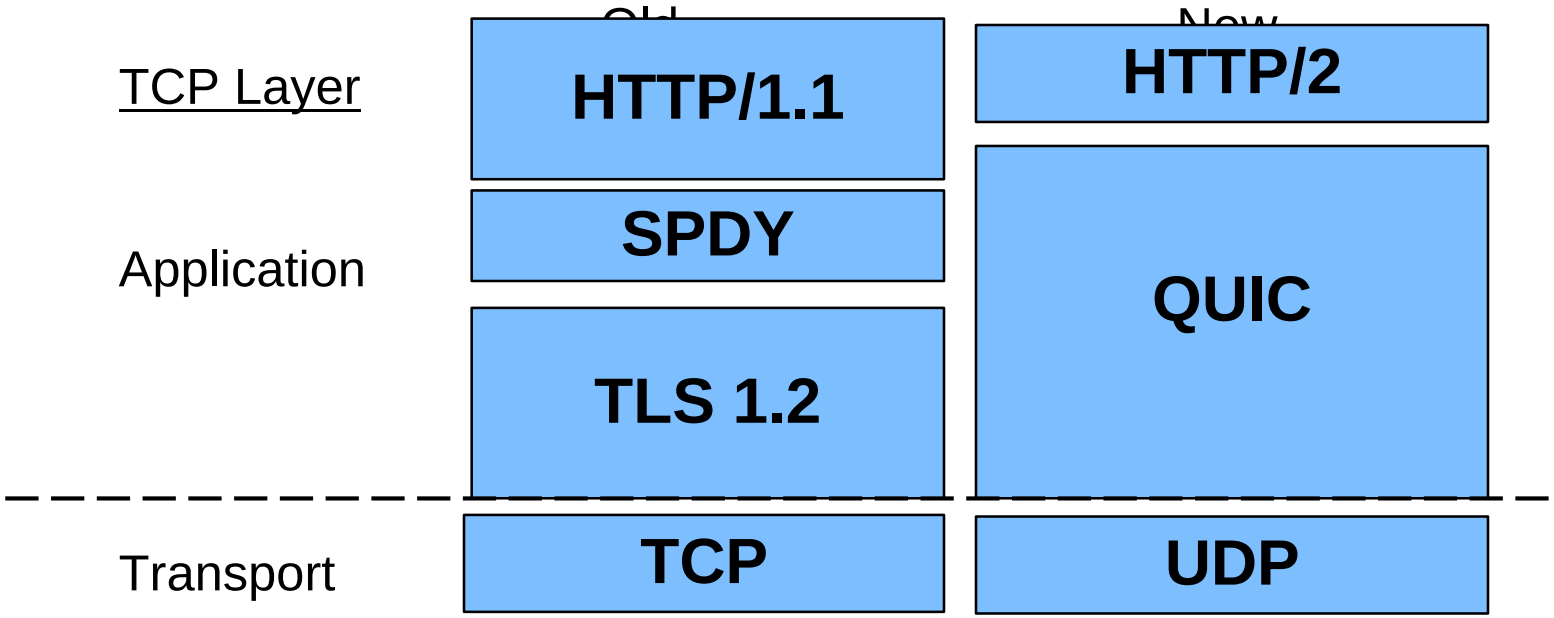
- aims to replace TCP
- reduce handshaking
  - TCP: 3 at the front, 2 at the end
- reduce RTT (round-trip-time)
  - by reducing number of RTs
- SSL/TLS-like security over UDP
- Allows stream multiplexing
    - (multi requests, receive responses in any order)
  - like SPDY
- Application layer congestion control which handles multiplexed connections
  - unlike TCP, which does not
- Fall back to TCP
  - if UDP is blocked

QUIC sits in the Transport Layer so it does not affect HTTP/1.1, but modifies signals for speed.

New Framework

- Overall design is to ***improve network latency***
  - improve by reducing ***number of packets*** sent
  - Header and body compression / encryption
  - Reduce duplication of headers and other information
  - Multiplex many parallel streams and manage them

more: https://ma.ttias.be/googles-quic-protocol-moving-web-tcp-udp/



Observations of New Framework

- These last few years showed how a major standard protocol can evolve.
  - Application Layer (a new HTTP?)
    - HTTP/2 is mostly HTTP/1.1 and SPDY combined
      - SPDY was temporary and is now implemented in HTTP/2
  - Transport Layer (a new TCP?)
    - QUIC is UDP with TLS, flow control and compression
      - QUIC is temporary and if no major issues arise, may well be incorporated into the next version of TCP

Summary

- HTTP
  - Definition
  - Resource Retrieval
  - Statelessness and Idempotence
  - Request and Response model
  - Request Methods and Error Codes
  - HTTP/2 / SPDY / QUIC
  - New Web Framework