Full Stack Web Development

HTTP

Outline

- HTTP
  - Overview
  - URLs & Resources
  - HTTP Messages
- TCP/IP
  - Overview
  - Connection Management?
- HTTP Architecture
  - Web Server
  - Proxies
  - Caching
  - Integration Points?
- HTTP Security
  - tbd.
- Identification, Authorization, and Security

HTTP

Overview

Web browsers, servers, and related web applications talk to each other through HTTP, the Hypertext Transfer Protocol.

HTTP: The Internet’s Multimedia Courier

- HTTP moves, the bulk of *resources* (e.g. JPEG images, HTML pages, text files, WAV audio files, …) shared through the Internet, quickly, conveniently and reliably from web servers around the world to web browsers, each and every day.
- HTTP uses reliable data-transmission protocols, guaranteeing data integrity (users) and reduced development effort (developer).

HTTP

Overview

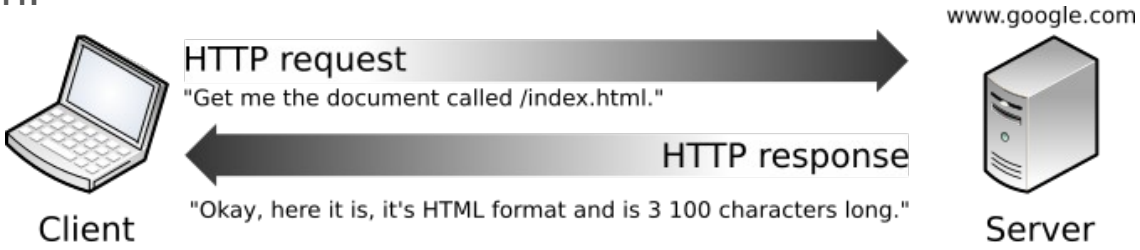
Web Clients and Servers

- Web content lives on web servers.
- Web servers speak the HTTP protocol, hence they are often referred to as *HTTP servers*.
- HTTP servers store the Internet’s data and provide the data when requested by *HTTP clients*.
- Clients send *HTTP requests* to servers, and servers return the requested data in *HTTP responses*.

HTTP

Overview

- Most common HTTP clients are web browsers, e.g. Mozilla Firefox, Google Chrome, Microsoft Edge, Apple Safari, …
- When browsing to “http://www.google.com”, your browser sends an HTTP request to the server *www.google.com*. The server tries to find the desired object (“/index.html”) and, if successful, sends the object to the client in an HTTP response, along with the object, the length of the object and other information.



HTTP

Overview

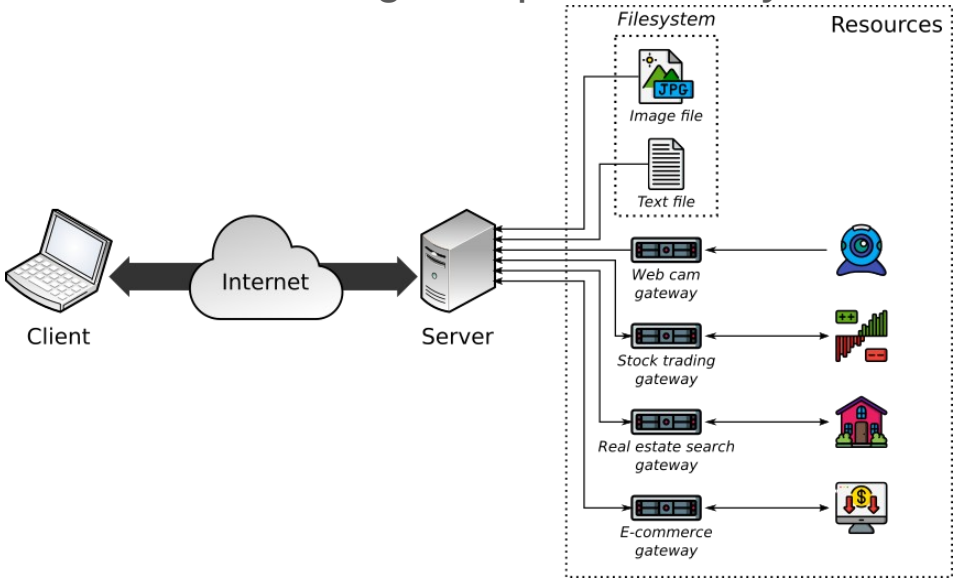
Resources

- Web servers host *web resources*. A web resource is the source of web content.
- The simplest kind of a web resource is a static file on the web server’s *filesystem* (e.g. text files, HTML files, Microsoft Word files, Webp image files, AVI movie files, etc.).
- Resources don’t have to be static files. Resources can also be software programs that generate content on demand, based on your identity, requested information, or on time of day. Resources can also be live image data from a camera, let you trade stocks, buy gifts from online stores, etc.

HTTP

Overview

- A resources is any kind of content source, e.g. company's forecast spreadsheet, Internet search engines, public library's shelves, etc.



HTTP

Overview

Media Types

- Because the Internet host a variety of different data types, HTTP carefully tags each object with a data format label called *MIME type*.
- MIME (Multipurpose Internet Mail Extensions) was originally designed to enable the transport of messages between different electronic mail systems and HTTP adopted it to describe and label its own multimedia content.
- Web servers attach a MIME type to all HTTP object data.
- Web clients, like web browsers, look at the associated MIME type to see if they know how to handle the object.

HTTP

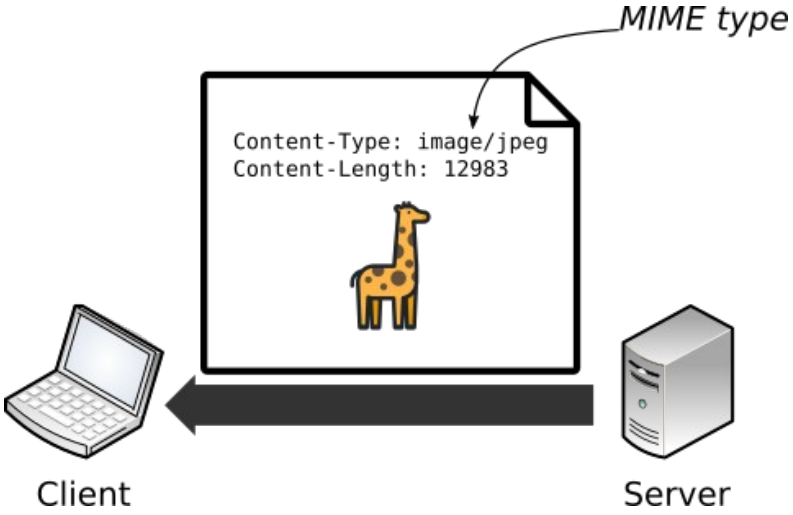
Overview

- A MIME type is a textual label, consisting of a primary object type and a specific subtype, separated by a slash:
  - HTML-formatted text documents: `text/html`
  - Plain ASCII text document: `text/plain`
  - JPEG version of an image: `image/jpeg`
  - GIF-format image: `image/gif`
  - Apple QuickTime movie: `video/quicktime`
  - Microsoft PowerPoint presentation: `application/vnd.ms-powerpoint`

HTTP

Overview

- Most web browsers can handle hundreds of popular object types: displaying image files, parsing and formatting HTML files, playing audio files, launching external plug-in software to handle special formats.



HTTP

Overview

URIs

- Web server resources have a name, so clients can point out what resources they are interested in.
- The server resource name is called a *uniform resource identifier*, or URI.
- URIs are kind of postal addresses of the Internet, uniquely identifying and locating information resources around the world.
- Example: https://www.technikum-wien.at/sites/default/files/logo-300x160_0.png
- URIs come in two flavors, called URLs and URNs.

HTTP

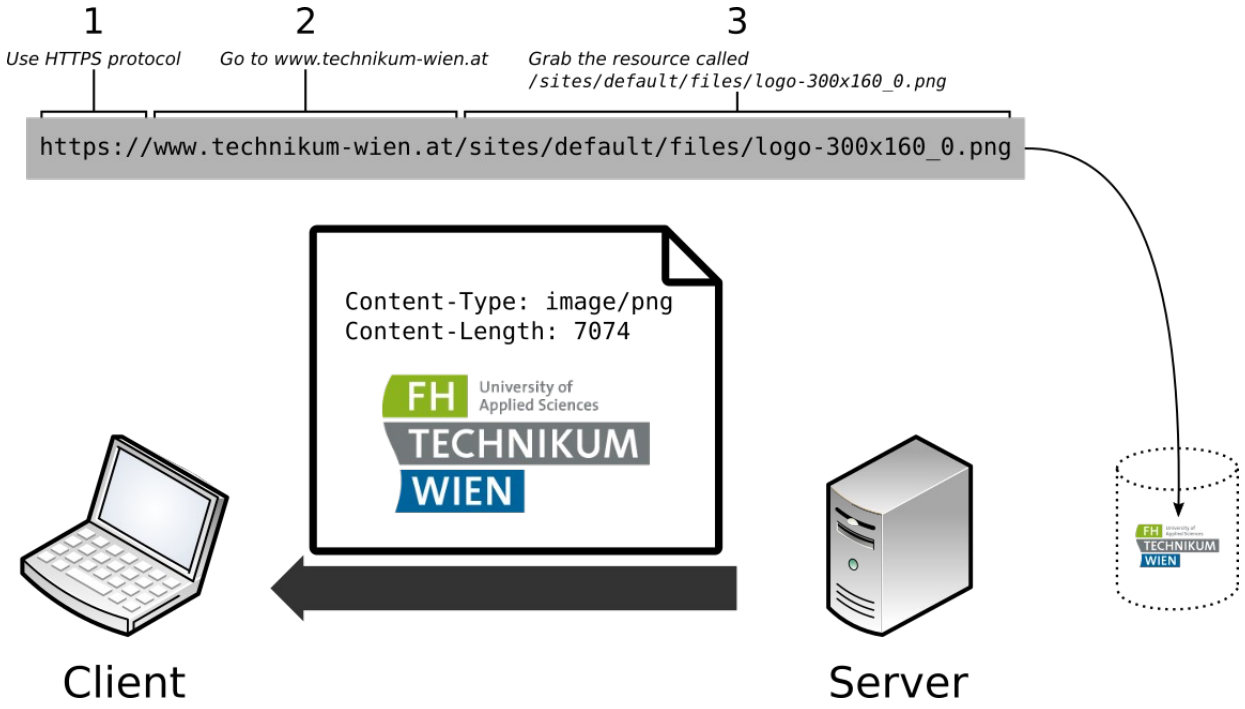
Overview

URLs

- The uniform resource locator (URL) is the most common form of resource identifier.
- URLs describe the specific location of a resource on a particular server. They tell you exactly how to fetch a resource from a precise, fixed location.
- Most URLs follow a standardized format of three main parts:
  - The first part is called the *scheme*, describing the protocol to use to access the resource.
  - The second part gives the server Internet address.
  - The rest names a resource on the web server.

HTTP

Overview



HTTP

Overview

URL	Description
https://www.google.com/index.html	
https://www.google.com/images/branding/googlelogo/1x/googlelogo_color_272x92dp.png	

HTTP

Overview

URNs

- The second type of URI is the *uniform resource name* (URN).
- A URN serves as a unique name for a particular piece of content, independent of its current location.
- URNs allow resource to move from place to place.
- URNs also allow resources to be accessed by multiple network access protocols while maintaining the same name.
- For example, following URN could be used to name the Internet standard document RFC 2141 regardless of where it resides:
  urn:ietf:rfc:2141

HTTP

Overview

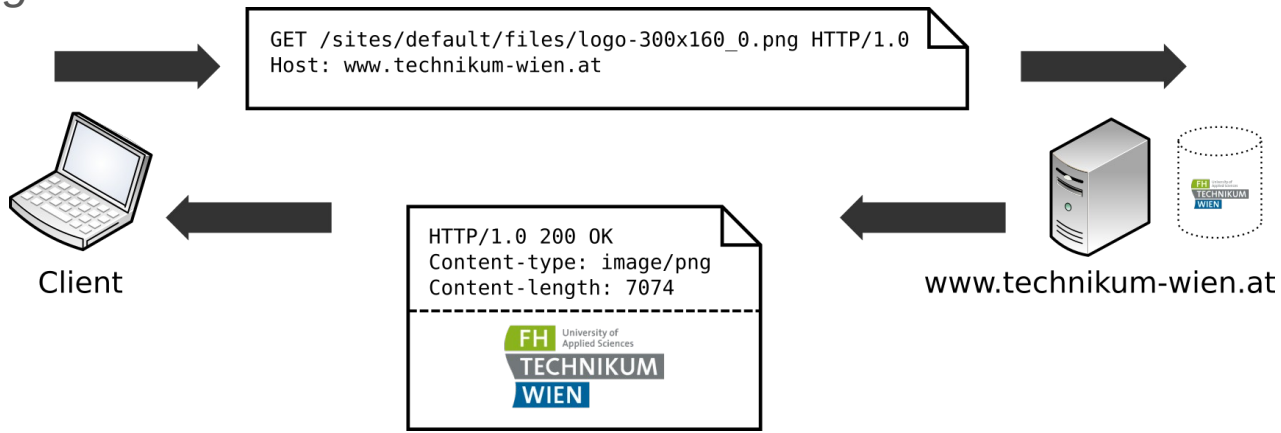
- URNs never gained widespread acceptance. The use of the terms *Uniform Resource Name* (URN) and *Uniform Resource Locator* (URL) has been deprecated in technical standards, since RFC 3986 in 2005, in favor of the term *Uniform Resource Identifier* (URI).
- Nowadays, URIs identify a name or resource. URIs are used in many Internet protocols to refer to and access information resources, perhaps uniquely and persistently, with some of them also being “locators” which are resolvable in conjunction with a specified protocol.
- The term URN continues now as one of more than hundred URI *schemes*, urn:, paralleling http:, ftp:, etc.

HTTP

Overview

Transactions

- HTTP transaction consists of a request command (sent from client to server), and a response result (sent from the server back to the client).
- This communication happs with formatted blocks of data called *HTTP messages*.



HTTP

Overview

Methods

- HTTP supports several different request commands, called *HTTP methods*.
- Every request message has a method, telling the server what action to perform (e.g. fetch a web page, delete a file, etc.)

HTTP method	Description
GET	Send named resource from the server to the client.
PUT	Store data from client into a named server resource.
DELETE	Delete the named resource from a server.
POST	Send client data into a server gateway application.
HEAD	Send just the HTTP headers from the response for the named resource.

HTTP

Overview

Status Codes

- Every HTTP response message comes back with a *status code*.
- The status code is a three-digit numeric code that tells the client if the request succeeded or not, or if other actions are required.

HTTP status code	Description
200	OK. Document returned correctly.
302	Redirect. Go someplace else to get the resource.
404	Not Found. Can’t find this resource.

HTTP

Overview

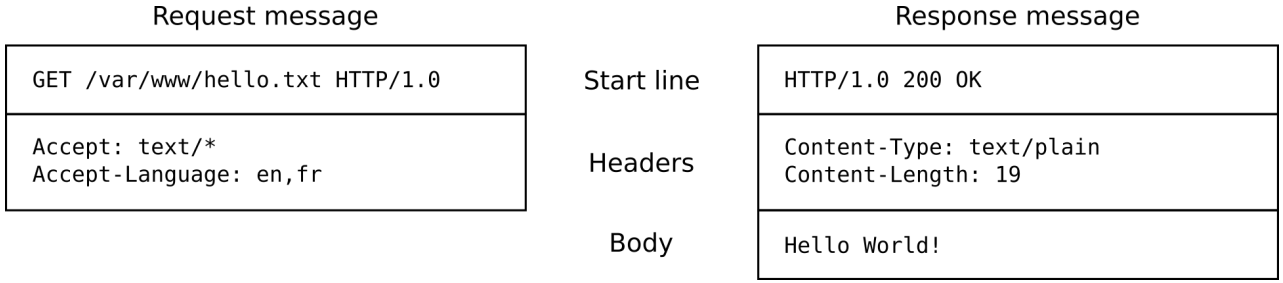
- HTTP also sends an explanatory textual “reason phrase” with each numeric status code.
- The textual phrase is included only for descriptive purposes; the numeric code is used for all processing. Following status codes and reason phrases are treated identically by HTTP software:
  - 200 OK
  - 200 Document attached
  - 200 Success
  - 200 All’s cool, dude

HTTP

Overview

Messages

- HTTP messages are simple, line-oriented sequences of characters.
- They are plain text, not binary, making them easy for humans to read and write.
- Messages sent from web clients to web servers are called *request messages*.
- Messages from servers to clients are called *response messages*.



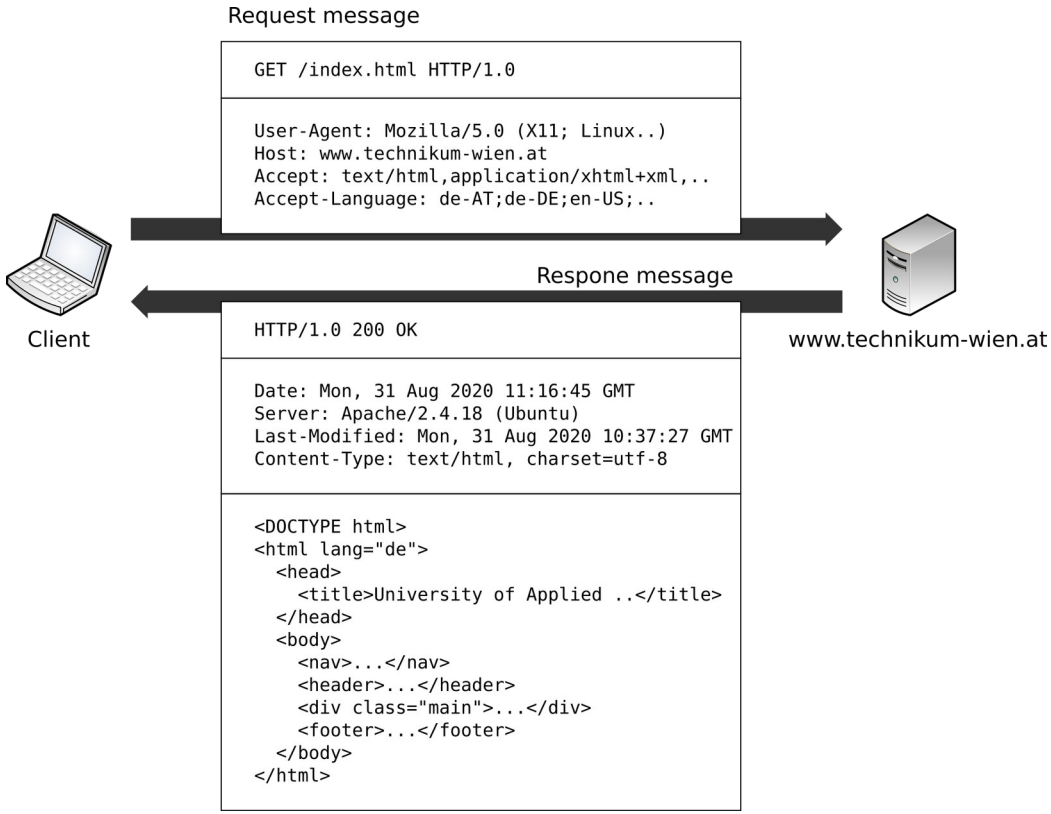
HTTP

Overview

- HTTP messages consist of three parts:
  - **Start Line**
    The first line of the message is the start line, indicating what to do for a request or what happened for a response.
  - **Header fields**
    Zero or more header fields follow the start line. Each header field consists of a name and a value, separated by a colon (:) for easy parsing. The headers end with a blank link. Adding a header field is as easy a adding another line.
  - **Body**
    After the blank linke is an optional body containing any kind of data. Request bodies carry data to the web server; response bodies carry data back to the client.

HTTP

Overview



HTTP

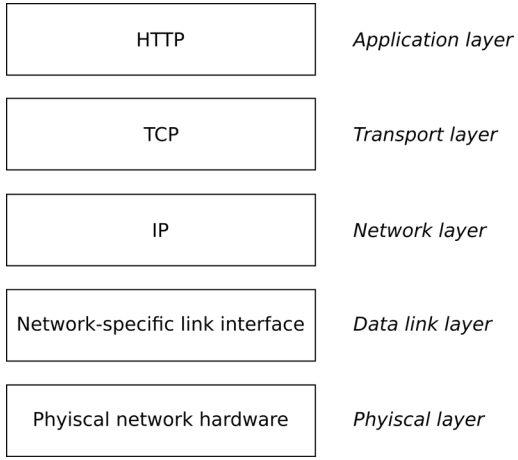
Overview

Connections

- HTTP is an application layer protocol, leaving the details of network communication to TCP/IP, the popular reliable Internet transport protocol.
- TCP/IP is a layered set of packet-switching network protocols spoken by computers and network devices around the world. TCP/IP hides the individual details of the networks and hardware, enabling medium-independent communication between participants.
- In networking terms, the HTTP protocol is layered over TCP. HTTP uses TCP to transport its message data. Likewise, TCP is layered over IP.

HTTP

Overview



- TCP provides:
  - Error-free data transportation
  - In-order delivery
  - Unsegmented data stream
- Once a TCP connection is established, messages exchanged between the client and server computers will never be lost, or received out of order.

HTTP

Overview

Connections, IP Addresses, and Port Numbers

- Before a HTTP client can send messages to a server, it needs to establish a TCP/IP connection between the client and server using *Internet protocol* (IP) addresses and port numbers.
- TCP needs the IP address of the server computer and the TCP port number associated with the specific software program running on the server.
- IP address and TCP port number are typically derived from URLs:
  - http://80.123.239.24/index.html
  - http://www.technikum-wien.at:80/index.html
  - http://www.google.at/index.html

HTTP

Overview

- URLs without a numeric IP address have a textual *domain name* or *hostname* (“www.technikum-wien.at”).
- Hostnames can be converted into IP addresses using the *Domain Name Service* (DNS).
- URLs without a specific port number are assumed to use a default values derived from the scheme (well-known port numbers; `http/80`, `https/443`).

BREAK!

Exercise

- Exercise 01
  - Open your browser and go to “https://www.google.com”
  - Open your browser’s developer tools and find out what file/resource you did request/receive
- Exercise 02
  - Connect to a HTTP server with telnet
  - Send HTTP request
  - Analyze HTTP response
- Exercise 03
  - Start a http server on your pc
  - Share content
  - Analyze traffic with Wireshark

HTTP

Overview

Protocol Versions

There are several versions of the HTTP protocol in use today. HTTP applications need to work hard to robustly handle different variations.

- HTTP/0.9
  HTTP/0.9 had only a single method (GET) and was designed to only fetch HTML (just text, no images).
  It does not support MIME typing, HTTP headers, or version numbers. This protocol contains many serious design flaws. Until 1995 around 18,000 servers were handling HTTP traffic on port 80 across the world.

HTTP

Overview

- HTTP/1.0
  HTTP/1.0 was the first version widely deployed. In 1996, the RFC 1945 codified HTTP/1.0, bringing a load of changes. The one page long specification of HTTP/0.9 grew to 60 pages with HTTP/1.0.
  HTTP/1.0 added version numbers, HTTP headers, additional methods, multimedia object handling, response codes, errors, content encoding (compression), and more. Though being a large leap from HTTP/0.9, HTTP/1.0 still had a number of known flaws, most notably, the inability to keep a connection open between requests, the lack of a mandatory Host header, and bare bones options for caching.

HTTP

Overview

- HTTP/1.0+
  Many popular web clients and servers rapidly added features to HTTP in the mid-1990s to meet the demands of a rapidly expanding, commercially successful World Wide Web.
  Many of these features, including long-lasting *keep-alive* connections, virtual hosting support, and proxy connection support, were added to HTTP and became unofficial, de facto standards.
- HTTP/1.1
  HTTP/1.1 focused on correcting architectural flaws in the design of HTTP by specifying semantics, introducing significant performance optimizations, and removing mis-features.

HTTP

Overview

- HTTP/1.1, cont.
  By making the Host header mandatory, it was now possible to perform *virtual hosting*, or serving multiple web properties on a single IP address.
  With using new connection directives, a web server is not required to close the connection after a response, allowing for an increase of performance and efficiency since the browser no longer needed to reestablish the TCP connection on every request.
  Pipelining, a feature allowing a client to send all of its requests at once, even if promising, had a couple of problems preventing its popularity. Servers still had to respond in order, resulting in *head of line blocking*.
  HTTP/1.1 was first defined in RFC 2068, then later replaced by RFC 2616, and finally revised in RFCs 7230 through 7235.

HTTP

Overview

- SPDY
  In 2009, Mike Belshe and Roberto Peon of Google proposed an alternative to HTTP, called SPDY (pronounced “speedy”). SPDY, although not the first proposal to replace HTTP, proved that there was a desire for something more efficient and, more importantly, a willingness to change. SPDY laid the groundwork for HTTP/2 and was responsible for proving out some of its key features such as multiplexing, framing, and header compression.
  It was integrated quickly, into Chrome and Firefox, and eventually would be adopted by almost every major browser. Similarly, the necessary support in servers and proxies came along at about the same pace.

HTTP

Overview

- HTTP/2
  In early 2012, the HTTP working group was rechartered to work on the next version of HTTP. After a initial call for proposals it was decided to use SPDY as a starting point for HTTP/2. In 2015, the RFC 7540 was published and HTTP/2 was official.
  The primary focus of HTTP/2 is on improving transport performance and enabling both lower latency and higher throughput.
  ...to be continued.

HTTP

Overview

Architectural Components of the Web

- Proxies
  HTTP intermediaries that sit between clients and servers.
  HTTP proxy servers are an important building block for web security, applications integration, and performance optimization.
- Caches
- Gateways
- Tunnels
- Agents