What is HTTP?: HTTP/2 in action notes(Barry Pollard)

Chapter 1. Web technologies and HTTP

1.1 How the Web Works

- Primary and first use of HTTP: to request web pages
- Suppose that you fire up a browser and go to www.google.com. Within a few seconds the following will have happened:
    1. The browser requests the real address of www.google.com from a Domain Name System (DNS) server, which translates the human-friendly name www.google.com to a machine-friendly IP address. If you think of an IP address as a telephone number, DNS is the telephone book. This IP address is either an older-format IPv4 address (which is nearly human-usable) or a new format IPv6 address. Due to the global nature of the internet, larger companies often have several servers around the globe. When you ask your DNS for the IP address, it often provides the IP address of the nearest server to make your internet browsing faster.
    2. The web browser asks your computer to open a Transmission Control Protocol (TCP) connection over IP to this address on the standard web port (port 80) or over the standard secure web port (port 443). IP is used to direct traffic through the internet, but TCP adds stability and retransmissions to make the connection reliable. As these two technologies are often used together, they're usually abbreviated as TCP/IP, and, together, they form the backbone of much of the internet. A server can be used for several services (such as email, FTP, HTTP, and HTTPS web servers), and the port allows different services to sit together under one IP address
    3. When the browser has a connection to the web server, it can start asking for the website. The web browser uses HTTP to ask the Google server for the Google home page
    4. Google server responds with whatever URL you asked for. Web page in HTML format is sent back. The webpage is divided into various sections defined by HTML tags and references other bits of information (CSS, JavaScript code, images, fonts, and so on)
    5. Web browser processes the returned request: browser starts to parse the HTML code and builds in memory the Document Object Model (DOM), which is an internal representation of the page
    6. Web browser requests any additional resources it needs. Each of these resources is requested in a similar manner, following steps 1-6 (those resources may in turn request other resources). This situation is one of the key things that makes web browsing slow and one of the key reasons for HTTP/2
    7. When the browser has enough of the critical resources, it starts to render the page onscreen. Choosing when to start rendering the page is a challenging task and not as simple as it sounds. If the web browser waits until all resources are downloaded, it would take a long time to show web pages. But if the web browser starts to render the page too soon, you end up with the page jumping around as more content downloads.
    8. After the initial display of the page, the web browser continues, in the background, to download other resources that the page needs and update the page as it processes them.

9. When the page is fully loaded, the browser stops the loading icon and fires the OnLoad JavaScript event, which JavaScript code may use as a sign that the page is ready to perform certain actions
10. At this point, the page is fully loaded, but the browser hasn't stopped sending out requests

1.2 What is HTTP?

- HTTP: Hypertext Transfer Protocol
- The Open Systems Interconnection (OSI) model
    1. Physical layer (such as Cable/WiFi/mobile)
    2. Data link layer (such as Ethernet)
    3. Network layer (Internet Protocol - IP)
    4. Transport and Session layer (Transmission Control Protocol)
    5. Session layer (Secure Sockets Layer - SSL / Transport Layer Security (TLS)
    6. Presentation layer (File format, such as ASCII, UTF-8, PNG, JPG, and so on...)
    7. Application layer (HTTP)
- HTTP is, at heart a request-and-response protocol
- The basic syntax of an HTTP request, after you open a connection: GET /page.html
- Web server needs only a TCP/IP connection o receive HTTP requests, you can emulate the browser by using a program such as Telnet
- Telnet is a simple program that opens a TCP/IP connection to a server and allows you to type text commands and view text responses (client used to try some simple HTTP commands
- For Windows, use the PuTTY software

1.3 The Syntax and History of HTTP

- Building upon HTTP/0.9, HTTP/1.0 added some key features including:
    - More request methods: HEAD and POST were added to the previously defined GET
    - Addition of an optional HTTP version number for all messages. HTTP/0.9 was assumed by default to aid in backward compatibility
    - HTTP headers, which could be sent with both the request and the response to provide more information about the resource being requested and the response being sent
    - A three-digit response code indicating (for example) whether the response was successful. This code also enabled redirect requests, conditional requests, and error status (404)
- The GET method stayed the same as under HTTP/0.9, though the addition of headers allowed a conditional GET (an instruction to GET only if this resource has changed since the last time the client got it; otherwise, tell the client that the resource hasn't changed and to carry on using that old copy). Also users could now get more than just hypertext media
- The HEAD method allowed a client to get all the meta information for a resource without downloading the resource itself. This can be useful because Google can check whether a resource has been modified and download it only if it has, thus saving resources and time
- The POST method allowed the client to send data to a web server. Rather than put a new HTML file directly on the server by using standard file-transfer methods, users could POST the file by using HTTP, provided that the web server was set up to receive the data and do something with it. POST isn't limited to whole files; it can be used for much smaller

parts of data. Forms on websites typically use POST. The POST method, therefore, allowed content to be sent from the client to the server as part of an HTTP request.

- A GET request is idempotent whereas a POST request is not, meaning that multiple GET requests to the same URL should always return the same result, whereas multiple POST requests to the same URL my not
- A typical HTTP/1.0 GET request is
  ```
  GET /page.html HTTP/1.0
  Accept: text/html, application/xhtml+xml, image/jxr/,*/*
  Accept-Encoding: gzip, deflate, br
  Accept-Language: en-GB, en-US;q=0.8, en;q=0.6
  Conection: keep-alive
  Host: www.example.com
  User-Agent: MyAwesomeWebBrowser 1.1
  ```
- This example tells the server what formats you can accept the response in (HTML, XHTML, XML, and so on), that you can accept various encodings (such as gzip, deflate, and brotli, which are compression algorithms used to compress data sent over HTTP), and what languages you prefer, and what browser you're using (MYAwesomeWebBrowser 1.1)
- A typical response from an HTTP/1.0 server is
  ```
  HTTP/1.0 200 OK
  Date: Sun, 25 Jun 2017 13:30:24 GMT
  Content-Type: text/html
  Server: Apache

  <!doctype html>
  <html>
  <head>
  ... Etc
  ```
- Some of the 3 digit responses can overlap. The response codes are designed to be broad categories, and its up to each application to use the status codes that fits best
- Changes that came with HTTP/1.1 (not fundamentally different from HTTP/1.0)
  - The host header was made mandatory
  - Added persistent connections
- Nowadays, many web servers host several sites on the same server (called virtual hosting), so it's important to tell the server which site you want as well as which relative URL you want on that site
- This feature was implemented by adding a Host header in the request:
  ```
  GET / HTTP/1.1
  Host: www.google.com
  ```
- This header is mandatory in HTTP/1.1. By not providing a Host header, the request is to be rejected by the server with a 400 response code, though most web servers are more forgiving and have a default host that is returned for such requests
- Requiring a host header allowed servers to make more us of virtual hosting and therefore allowing the enormous growth of the web without the complexity of adding individual web servers for each site. Additionally, the relatively low limit of IPv4 IP addresses would have been reached much sooner without this change.