Everything you need to know about HTTP

Source : https://cs.fyi/guide/http-in-depth

What is HTTP?

HTTP : TCP/IP-based / application layer communication protocol / standardizes how clients and servers communicate with each other

HTTP defines how content is requested(요청) and transmitted(전송) across the internet.

Application layer protocol is simply an abstraction layer that standardizes how hosts (clients and servers) communicate.

HTTP itself depends on TCP/IP to get requests and responses between the client and server. By default, TCP port 80 is used, but other ports can also be used. HTTPS, however, uses port 443.

HTTP/0.9 - The One Liner (1991)

The first documented version of HTTP was HTTP/0.9 which was put forward in 1991.

It only has a single method called GET.
If a client had to access some web page on the server, it would have made the simple request like below:
GET /index.html

And the response from server would have looked as follows:
(response body)
(connection closed)
That is, the server would get the request, reply with the HTML in response and as soon as the content has been transferred, the connection will be closed.

Features:

- No headers
- 'GET' was the only allowed method
- Response had to be HTML

HTTP/1.0 (1996)

In 1996, the next version of HTTP (HTTP/1.0) evolved that vastly(크게) improved over the original version.

Unlike HTTP/0.9 which was only designed for HTML response, HTTP/1.0 could now deal with other response formats like images, video files, plain text or any other content type as well.

Features:

- more methods (POST and HEAD)
- request/response formats got changed
- HTTP headers got added to both the request and responses
- status codes were added to identify the response
- character set support was introduced
- multi-part types : https://learn.microsoft.com/en-us/previous-versions/office/developer/exchange-server-2010/aa493937(v=exchg.140)
- authorization
- caching
- content encoding
- and more

Sample Request

Here is how a sample HTTP/1.0 request might have looked like:

```
GET / HTTP/1.0
Host: cs.fyi
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5)
Accept: */*
```

Alongside the request, the client has also sent its personal information, required response type etc. While in HTTP/0.9 clients could never send such information because there were no headers.

Sample Response

Example response to the request above may have looked like:

```
HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: 137582
Expires: Thu, 05 Dec 1997 16:00:00 GMT
Last-Modified: Wed, 5 August 1996 15:55:28 GMT
Server: Apache 0.84

(response body)
(connection closed)
```

In the first line of the response there is HTTP/1.0 (HTTP followed by the version number), then there is the status code 200 followed by the reason phrase (or description of the status code, if you will).

In this newer version, request and response headers were still kept as ASCII encoded, but the response body could have been of any type i.e. image, video, HTML, plain text or any other content type. So, now the server could send any content type to the client.

Connectionless and Drawback of HTTP/1.0

You couldn't have multiple requests per connection. That is, whenever a client will need something from the server, it will have to open a new TCP connection and after that single request has been fulfilled, the connection will be closed. And for any next requirement, it will have to be on a new connection.

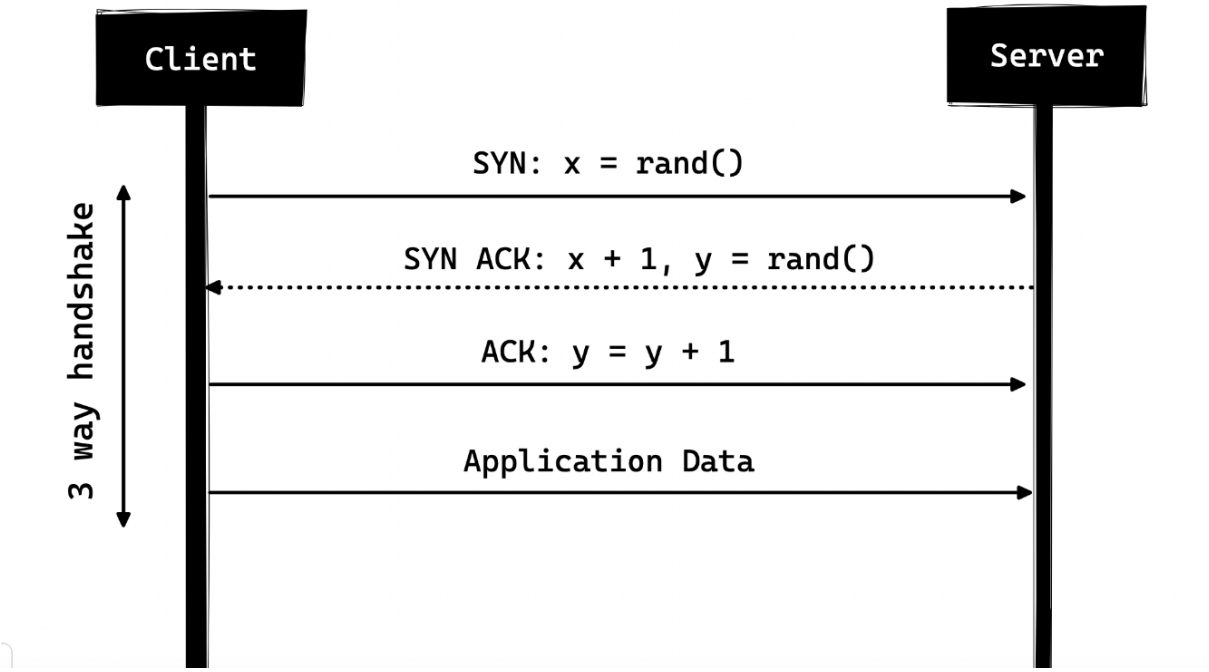
Why is it bad? Let's assume that you visit a webpage having 10 images, 5 stylesheets and 5 javascript files, totalling to 20 items that need to be fetched(가지고 오다) when a request to that web page is made. Since the server closes the connection as soon as the request has been fulfilled, there will be a series of 20 separate connections where each of the items will be served one by one on their separate connections. This large number of connections results in a serious performance hit as requiring a new TCP connection imposes(부과하다) a significant performance penalty because of three-way handshake followed by slow-start.

Three-way Handshake

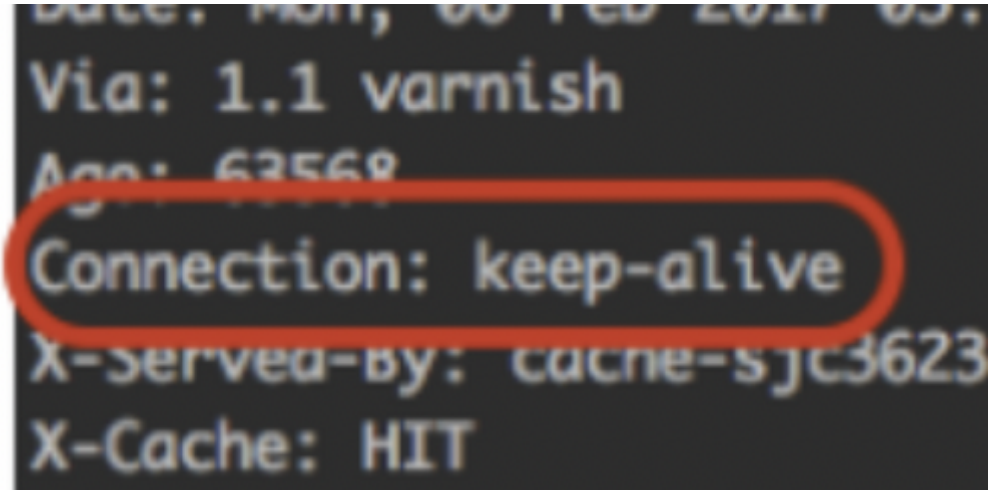
All the TCP connections begin with a three-way handshake in which the client and the server share a series of packets before starting to share the application data.

- SYN - Client picks up a random number, let's say x, and sends it to the server.
- SYN ACK - Server acknowledges(승인하다) the request by sending an ACK packet back to the client which is made up of a random number, let's say y picked up by server and the number x+1 where x is the number that was sent by the client
- ACK - Client increments the number y received from the server and sends an ACK packet back with the number y+1.

Once the three-way handshake is completed, the data sharing between the client and server may begin. It should be noted that the client may start sending the application data as soon as it dispatches(발송하다) the last ACK packet but the server will still have to wait for the ACK packet to be received in order to fulfill the request.



Connection: keep-alive



As we said before, a large number of connections with HTTP/1.0 can take a serious performance hit due to three-way handshake.

However, some implementations of HTTP/1.0 tried to overcome this issue by introducing a new header called Connection: keep-alive which was meant to tell the server to not close the connection. But still, it wasn't that widely supported and the problem still persisted(지속되다).

Stateless

HTTP is a stateless protocol. A server doesn't maintain the information about the client and so each of the requests has to have the information necessary for the server to fulfill the request on its own without any association with any old requests.

So it has to send some redundant(중복된) data on the wire causing increased bandwidth usage.

Stateless feature of HTTP is different from Connectionless.

HTTP/1.1 (1997)

After 3 years of HTTP/1.0, HTTP/1.1 was released in 1999.

Major Improvements of HTTP/1.1 over HTTP/1.0

Persistent Connections

HTTP/1.1 introduced the persistent connections i.e. connections weren't closed by default and were kept open which allowed multiple sequential requests. To close the connections, the header Connection: close had to be available on the request. Clients usually send this header in the last request to safely close the connection.

Pipelining

HTTP/1.1 introduced the support for pipelining, where the client could send multiple requests to the server without waiting for the response from the server on the same connection. And the server had to send the response in the same sequence in which requests were received.

But how does the client know that this is the point where the first response download completes and the content for the next response starts? To solve this, there must be a Content-Length header present which clients can use to identify where the response ends and it can start waiting for the next response.

Chunked Transfers

In case of dynamic content, when the server cannot really find out the Content-Length when the transmission starts, it may start sending the content in pieces (chunk by chunk) and add the Content-Length for each chunk when it is sent. In order to notify the client about the chunked transfer, server includes the header Transfer-Encoding: chunked

Others

New HTTP methods were added, which introduced PUT, PATCH, OPTIONS, DELETE

Hostname Identification : HTTP/1.0 host header wasn't required Hostname Identification, but HTTP/1.1 made it required

Caching / Byte Ranges / Character sets / Language negotiation / Client cookies / Enhanced compression support / New status codes / and more

SPDY (2009)

In 2009, Google announced SPDY, an alternative protocol to make the web faster and improve web security while reducing the latency of web pages.

Core Idea

bandwidth increases -> network performance increases

But because of latency, at some point there is not much of a performance gain.
If we keep dropping the latency, there is a constant performance gain.

core idea for performance gain behind SPDY : decrease the latency to increase the network performance.

Latency : how long it takes for data to travel between the source and destination (measured in milliseconds)
Bandwidth : the amount of data transferred per second

HTTP/2 (2015)

HTTP/2 was designed for low latency transport of content.

The key features or differences from the HTTP/1.1 are:

- Binary instead of Textual
- Multiplexing - Multiple asynchronous(비동기) HTTP requests over a single connection
- Header compression using HPACK
- Server Push - Multiple responses for single request
- Request Prioritization
- Security

Binary Protocol

HTTP/2 tends to address the issue of increased latency that existed in HTTP/1.x by making it a binary protocol. It's easier to parse but unlike HTTP/1.x it is not readable by the human eye.

Multiplexing

Since HTTP/2 is now a binary protocol and as I said above that it uses frames and streams for requests and responses, once a TCP connection is opened, all the streams are sent asynchronously through the same connection without opening any additional connections.

// 무슨 말인지? TCP가 binary protocol인가? 그래서 같은 binary protocol이기 때문에 sent asynchronously라고?

And in turn, the server responds in the same asynchronous way i.e. the response has no order and the client uses the assigned stream id to identify the stream to which a specific packet belongs.

This also solves the head-of-line blocking issue that existed in HTTP/1.x.
head-of-line blocking issue : the client will not have to wait for the request that is taking time and other requests will still be getting processed.

Pipelining vs Multiplexing :
https://stackoverflow.com/questions/34478967/what-is-the-difference-between-http-1-1-pipelining-and-http-2-multiplexing

Header Compression

When we are constantly accessing the server from the same client, there is a lot of redundant data that we are sending in the headers over and over, and sometimes there might be cookies increasing the headers size which results in bandwidth usage and increased latency. To overcome this, HTTP/2 introduced header compression.

Unlike request and response, headers are not compressed in gzip or compress etc formats.

There's a different mechanism in place for header compression, which is literal values are encoded using Huffman code and a headers table is maintained by the client and server, and both the client and server omit(생략하다) any repetitive header in the subsequent(후속) requests and reference them using the headers table maintained by both.

Is HTTP/2 a stateless protocol? HTTP/2 comes with a stateful header compression called HPACK : https://stackoverflow.com/questions/36178447/is-http-2-a-stateless-protocol

Server Push

The server, knowing that the client is going to ask for a certain resource, can push it to the client without even asking for it.

Server push allows the server to decrease the round trips(왕복) by pushing the data that is knows that client is going to demand. How it is done is, the server sends a special frame called PUSH_PROMISE notifying the client that, "Hey, I am about to send this resource to you! Do not ask me for it."

Request Prioritization

A client can assign(지정하다) a priority to a stream by including the prioritization information in the HEADERS frame by which a stream is opened. At any other time, the client can send a PRIORITY frame to change the priority of a stream.

Without priority information -> server processes the requests asynchronously
With priority information -> server decides how much of the resources need to be given to process which request

Security

Security through TLS is not mandatory(의무) for HTTP/2.

But most vendors stated that they will only support HTTP/2 when it is used over TLS.

So, it is not required by specs, but it has kind of become mandatory by default anyway.