Intro

Alexey Titov

Ariel University

December 25, 2020

Table of Contents

1. DNS
   - Introduction
   - Messages
   - Practice
2. Hypertext Transfer Protocol
   - Introduction
   - HTTP/2
   - HTTP/3
3. Transport Layer Security
4. DNS-over-HTTPS

Table of Contents

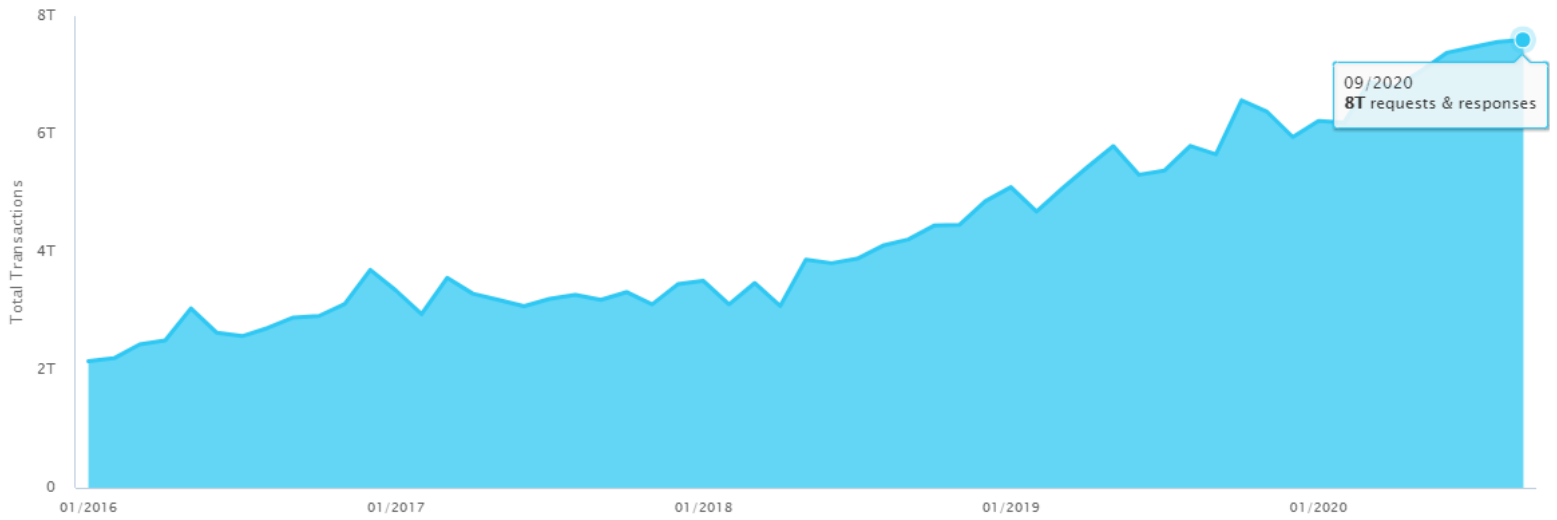
1. DNS
   - Introduction
   - Messages
   - Practice
2. Hypertext Transfer Protocol
   - Introduction
   - HTTP/2
   - HTTP/3
3. Transport Layer Security
4. DNS-over-HTTPS

DNS

The Domain Name System (DNS) is a distributed database system that translates hostnames to IP addresses and IP addresses to hostnames (e.g., it translates hostname miles.somewhere.net to IP address 192.168.244.34). DNS is also the standard Internet mechanism for storing and accessing several other kinds of information about hosts; it provides information about a particular host to the world at large.

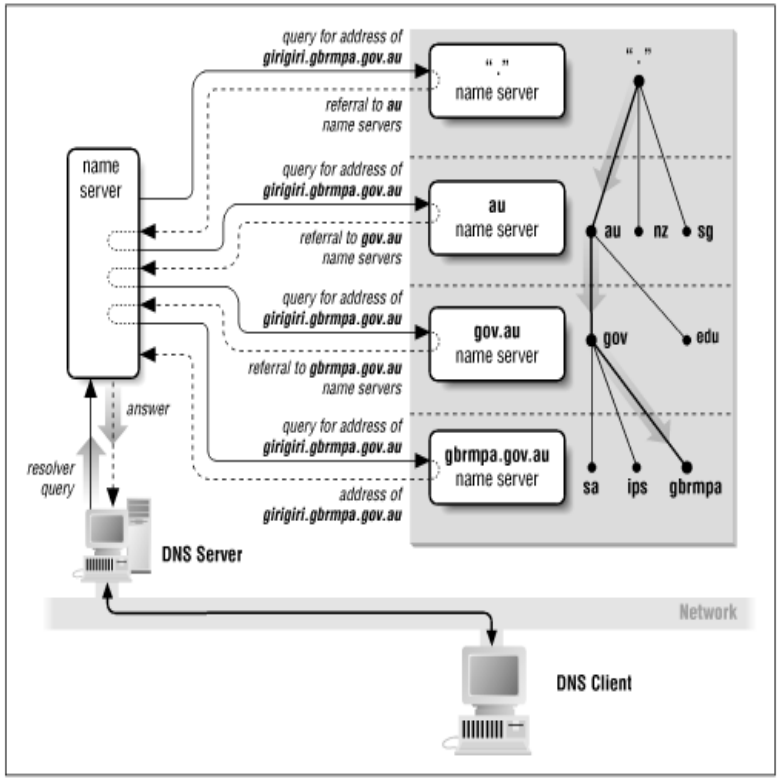
DNS

Only in the summer of 2020, more than 7 trillion DNS messages were exchanged per month.



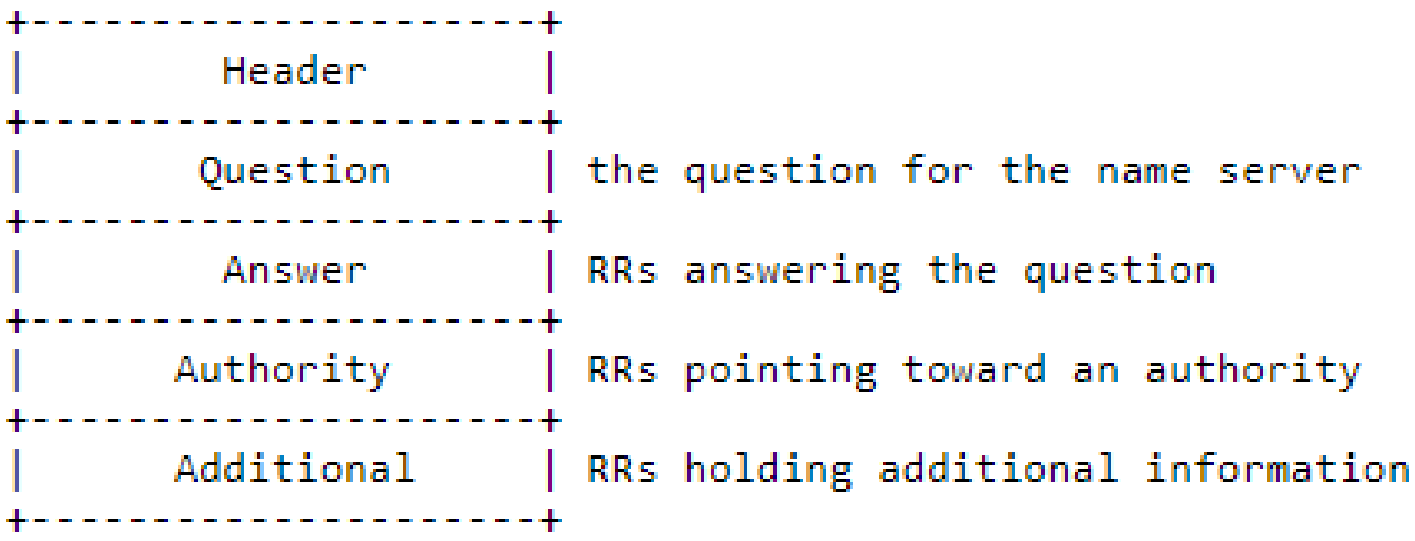
Link to statistics

DNS


query for address of girigiri.gbrmpa.gov.au
"."
name server
referral to au name servers
name server
query for address of girigiri.gbrmpa.gov.au
au
name server
referral to gov.au name servers
query for address of girigiri.gbrmpa.gov.au
gov.au
name server
referral to gbrmpa.gov.au name servers
answer
query for address of girigiri.gbrmpa.gov.au
gbrmpa.gov.au
name server
resolver query
address of girigiri.gbrmpa.gov.au
DNS Server
Network
DNS Client
"."
au
nz
sg
gov
edu
sa
ips
gbrmpa

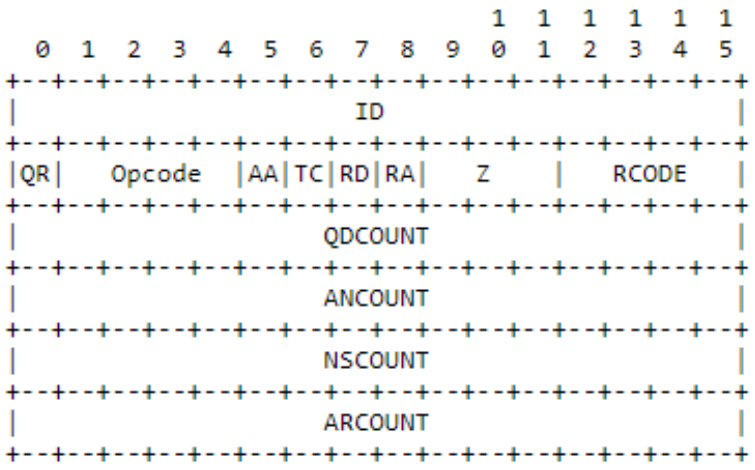
DNS

All communications inside of the domain protocol are carried in a single format called a message. The top level format of message is divided into 5 sections.



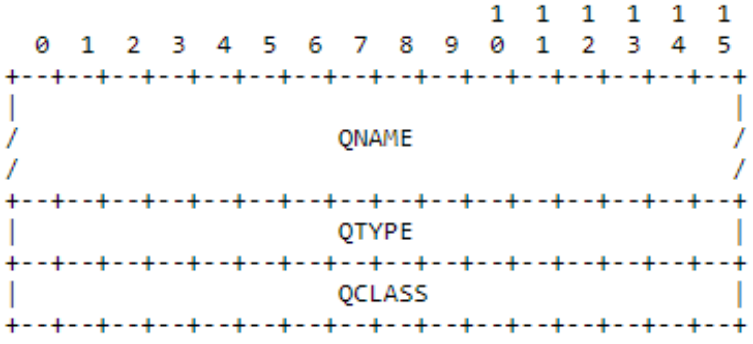
DNS

Header - The DNS header of the packet, 12 octets in length.



DNS

Question section - in this section, the DNS client sends requests to the DNS server informing about which name it is necessary to resolve the DNS record, as well as what type (NS, A, TXT, etc.). When the server responds, it copies this information and gives it back to the client in the same section.

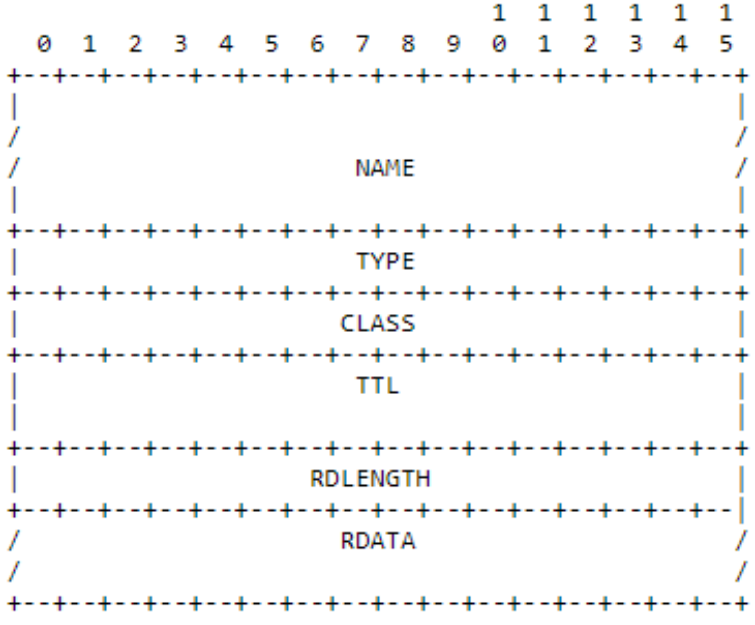


DNS

- **Answer section** - the server informs the client about the answer or several responses to the request, in which it reports the above data.
- **Authoritative section** - contains information about which authoritative servers were used to obtain information included in the DNS response section.
- **Additional record section** - additional records that are related to the request, but are not strictly answers to the question.

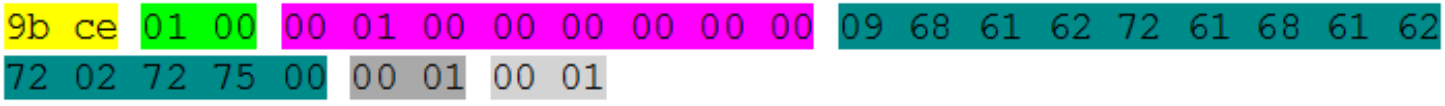
DNS

The answer, authority, and additional sections all share the same format: a variable number of resource records, where the number of records is specified in the corresponding count field in the header. Each resource record has the following format



DNS

Request


9b ce 01 00 00 01 00 00 00 00 00 00 09 68 61 62 72 61 68 61 62
72 02 72 75 00 00 01 00 01

DNS

- **Transaction ID** = 0x9bce.
- Next are the flags. 01 00 is represented as a binary value 0'0000'0'0'1'0'000'0000.
- **QR** = 0 means this packet is a request.
- **OPCODE** = 0000 - Standard request.
- **AA** = 0 - this field is meaningful only in DNS responses, therefore it is always 0.
- **TC** = 0 - this field is meaningful only in DNS responses, therefore it is always 0.

DNS

- **RD** = 1 - Please return only the IP address.
- **RA** = 0 - sent only by the server.
- **Z** = 000 - always zeros, reserved field.
- **RCODE** = 0000 - Everything went without errors.
- **QDCOUNT** = 00 01 - 1 record in the request section.
- **ANCOUNT** = 00 00 - The request always contains 0, section for responses.
- **NSCOUNT** = 00 00 - The request always contains 0, section for responses.
- **ARCOUNT** = 00 00 - The request always contains 0, section for responses.

DNS

Our first octet is 0x09, we represent it as a binary value 00'001001. The first two bits are 00, which means this is a normal label. The label is 9 bytes long (b001001). "68 61 62 72 61 68 61 62 72". These are 9 bytes. It says "habrahabr" (hexadecimal). Move on. Octet 0x02. The first two bits are 00, which means again a regular label with a length of 2 bytes. Here they are: "72 75". It is written "ru". Move on. Octet 0x00. Means the end of the host record. We got two words "habrahabr" and "ru". We combine them with a dot, we get "habrahabr.ru", this is the host that we requested.

DNS

- **QTYPE** = 0x0001 - Corresponds to type A (request for host address).
- **QCLASS** = 0x0001 - Corresponds to IN class.

DNS

Response

```
9b ce 81 80 00 01 00 01 00 00 00 00 09 68 61 62 72 61 68 61 62
72 02 72 75 00 00 01 00 01 09 48 41 42 52 41 48 41 42 52 c0 16
00 01 00 01 00 00 0c 90 00 04 b2 f8 ed 44
```

DNS

- **Transaction ID** = 0x9bce. It must be equal to the ID from the request.
- **QR** = 1 means this packet is a response.
- **OPCODE** = 0000 - Standard request.
- **AA** = 0 - Server is not authoritative for the domain.
- **TC** = 0 - All information fit into one package.
- **RD** = 1 - Please return only the IP address.
- **RA** = 1 - Server supports recursion.
- **Z** = 000 - always zeros, reserved field.
- **RCODE** = 0000 - Everything went without errors.

DNS

- **QDCOUNT** = 00 01 - 1 record in the request section.
- **ANCOUNT** = 00 01 - Now we have one record in the answer.
- **NSCOUNT** = 00 00 - The request always contains 0, section for responses.
- **ARCOUNT** = 00 00 - The request always contains 0, section for responses.
- **QTYPE** = 0x0001 - Corresponds to type A (request for host address).
- **QCLASS** = 0x0001 - Corresponds to IN class.
- **TTL** = 0x00000c90 - data actuality time 3216 seconds.
- **RDLENGTH** = 0x0004 - Data length 4 octets.
- **RDATA** = "b2 f8 ed 44".

Table of Contents

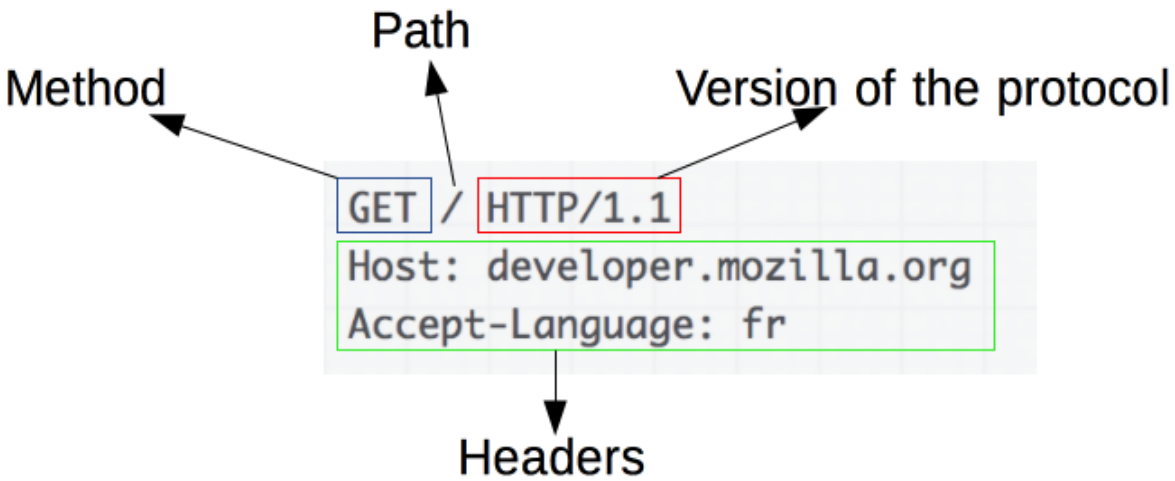
1. DNS
   - Introduction
   - Messages
   - Practice
2. Hypertext Transfer Protocol
   - Introduction
   - HTTP/2
   - HTTP/3
3. Transport Layer Security
4. DNS-over-HTTPS

HTTP

Hypertext Transfer Protocol (HTTP) is an application-layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers, but it can also be used for other purposes. HTTP follows a classical client-server model, with a client opening a connection to make a request, then waiting until it receives a response. HTTP is a stateless protocol, meaning that the server does not keep any data (state) between two requests. Though often based on a TCP/IP layer, it can be used on any reliable transport layer, that is, a protocol that doesn't lose messages silently like UDP does.

HTTP

Request



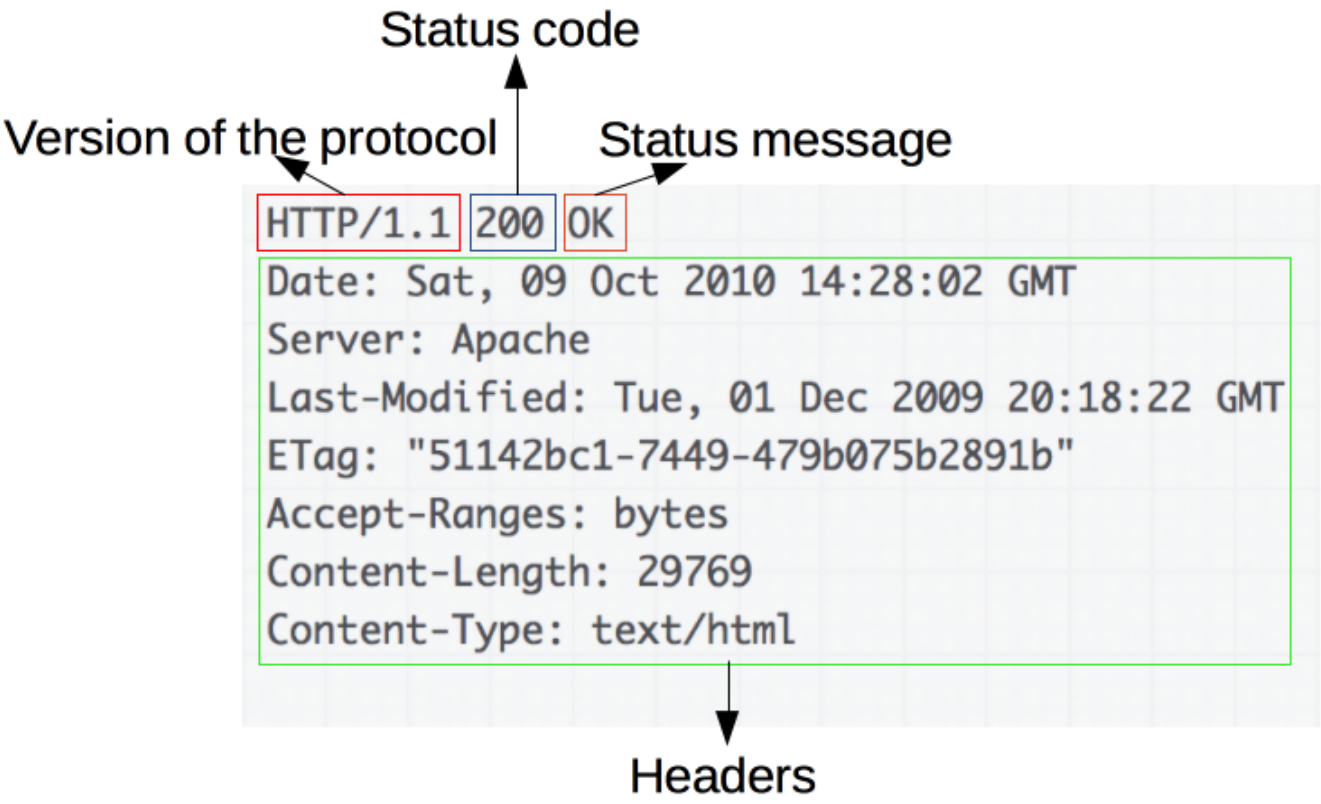
HTTP

Requests consists of the following elements:

- An HTTP method, usually a verb like GET, POST or a noun like OPTIONS or HEAD that defines the operation the client wants to perform.
- The path of the resource to fetch; the URL of the resource stripped from elements that are obvious from the context, for example without the protocol, the domain, or the TCP port.
- The version of the HTTP protocol.
- Optional headers that convey additional information for the servers.
- Or a body, for some methods like POST, similar to those in responses, which contain the resource sent.

HTTP

Response



HTTP

Responses consist of the following elements:

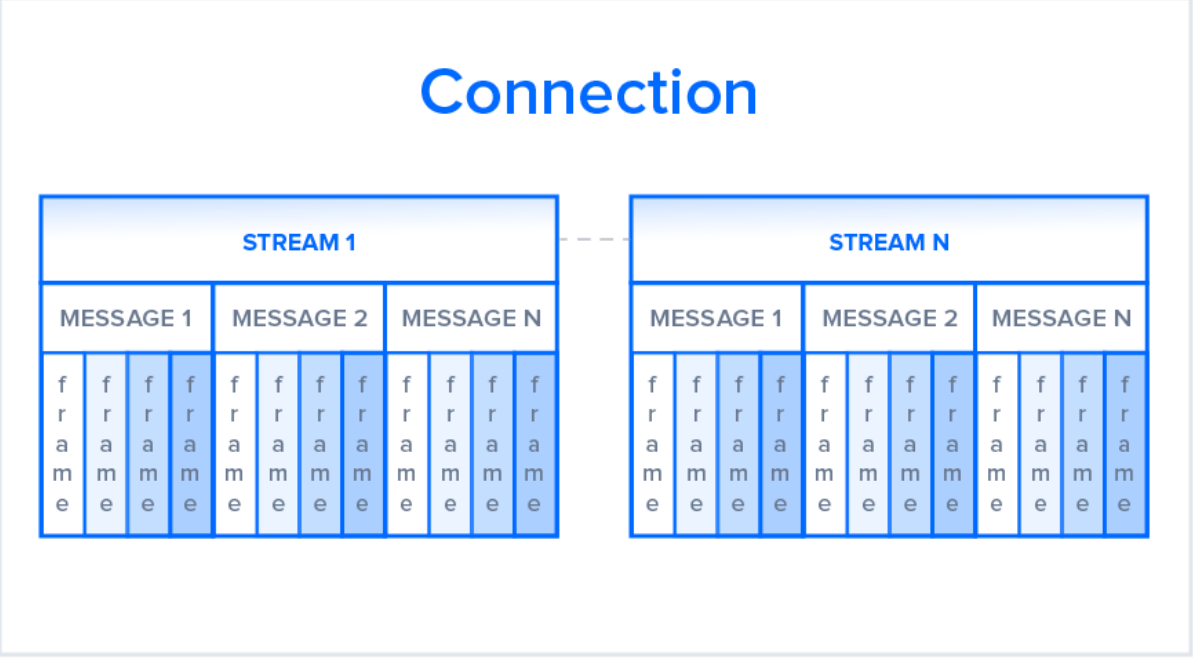
- The version of the HTTP protocol they follow.
- A status code, indicating if the request was successful, or not, and why.
- A status message, a non-authoritative short description of the status code.
- HTTP headers, like those for requests.
- Optionally, a body containing the fetched resource.

HTTP/2

HTTP/2 began as the SPDY protocol, developed primarily at Google with the intention of reducing web page load latency by using techniques such as compression, multiplexing, and prioritization.
From a technical point of view, one of the most significant features that distinguishes HTTP/1.1 and HTTP/2 is the binary framing layer, which can be thought of as a part of the application layer in the internet protocol stack.
The conversion of messages into binary allows HTTP/2 to try new approaches to data delivery not available in HTTP/1.1, a contrast that is at the root of the practical differences between the two protocols.

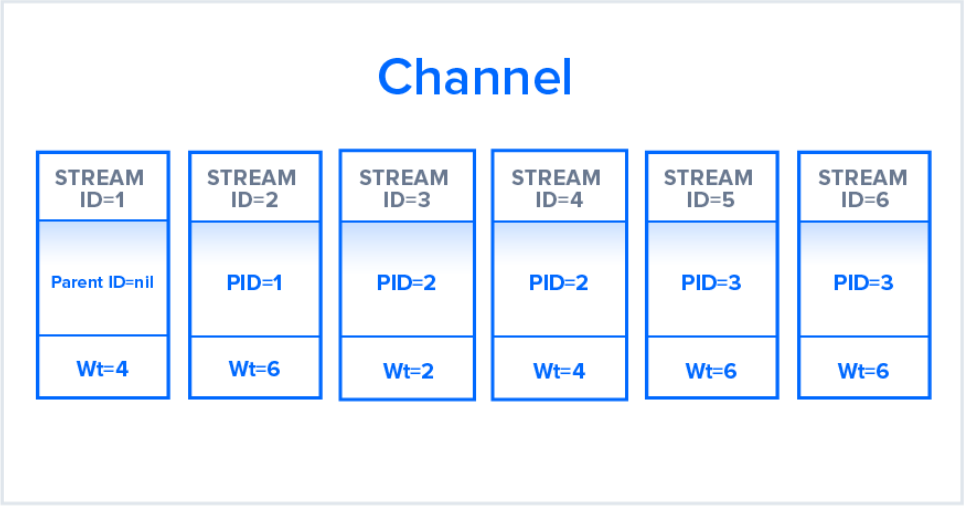
HTTP/2

In HTTP/2, the binary framing layer encodes requests/responses and cuts them up into smaller packets of information, greatly increasing the flexibility of data transfer.



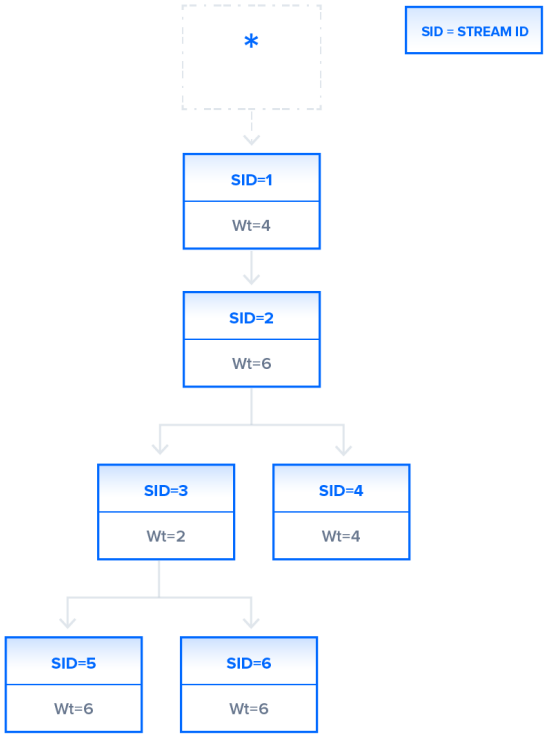
HTTP/2

When a client sends concurrent requests to a server, it can prioritize the responses it is requesting by assigning a weight between 1 and 256 to each stream. In addition to this, the client also states each stream's dependency on another stream by specifying the ID of the stream on which it depends. If the parent identifier is omitted, the stream is considered to be dependent on the root stream.



HTTP/2

The server uses this information to create a dependency tree, which allows the server to determine the order in which the requests will retrieve their data.



HTTP/2

- Since HTTP/2 enables multiple concurrent responses to a client's initial **GET** request, a server can send a resource to a client along with the requested HTML page, providing the resource before the client asks for it. This process is called *server push*.
- In HTTP/2, this process begins when the server sends a **PUSH_PROMISE** frame to inform the client that it is going to push a resource. This frame includes only the header of the message, and allows the client to know ahead of time which resource the server will push. If it already has the resource cached, the client can decline the push by sending a **RST_STREAM** frame in response.

HTTP/2

One of the themes that has come up again and again in HTTP/2 is its ability to use the binary framing layer to exhibit greater control over finer detail. The same is true when it comes to header compression. HTTP/2 can split headers from their data, resulting in a header frame and a data frame. The HTTP/2-specific compression program **HPACK** can then compress this header frame. This algorithm can encode the header metadata using Huffman coding, thereby greatly decreasing its size. Additionally, **HPACK** can keep track of previously conveyed metadata fields and further compress them according to a dynamically altered index shared between the client and the server.

HTTP/3

There are several differences between HTTP/2 and HTTP/3, but the main one is that HTTP/3 runs over a transport layer network protocol called QUIC that uses UDP as the transport layer protocol instead of TCP, resulting on performance and security improvements.

HTTP/3

Feature	HTTP/2	HTTP/3
Header compression algorithm	HPACK	QPACK
Handshake protocol	TCP + TLS	iQUIC
Handshake negotiation	At the certificate stage via ALPN=Application-Layer Protocol Negotiation (ALPN) protocol	After certificate negotiation via "Alt-Svc:" HTTP response header

HTTP/3

HTTP scheme	HTTP (not well adopted) / HTTPS	HTTPS
Prioritization	Yes	No, although HTTP/3 streams can have a "PRIORITY" frame to implement it

HTTP/3

- HTTP/3 will be performed using *HTTPS://* URLs. The world is full of these URLs and it has been deemed impractical and downright unreasonable to introduce another URL scheme for the new protocol.
- The added complexity in the HTTP/3 situation, that HTTP/3 is done over QUIC changes things in a few important aspects.
- Legacy, clear-text, *HTTP://* URLs will be left as-is and as we proceed further into a future with more secure transfers they will probably become less and less frequently used. Requests to such URLs will simply not be upgraded to use HTTP/3.

HTTP/3

The alternate service (Alt-svc:) header and its corresponding **ALT-SVC** HTTP/2 frame are not specifically created for QUIC or HTTP/3. They are part of an already designed and created mechanism for a server to tell a client: *"look, I run the same service on THIS HOST using THIS PROTOCOL on THIS PORT"*. An HTTP server includes an Alt-Svc: header in its response:

Alt-Svc: h3=":50781"

This indicates that HTTP/3 is available on UDP port 50781 at the same host name that was used to get this response.

HTTP/3

HTTP requests done over HTTP/3 use a specific set of streams.

- **HTTP/3 frames** - HTTP/3 means setting up QUIC streams and sending over a set of frames to the other end. There's but a small fixed number of known frames in HTTP/3. The most important ones are probably:
  - HEADERS, that sends compressed HTTP headers
  - DATA, sends binary data contents
  - GOAWAY, please shutdown this connection

HTTP/3

- **HTTP Request** - The client sends its HTTP request on a client-initiated bidirectional QUIC stream. A request consists of a single HEADERS frame and might optionally be followed by one or two other frames: a series of DATA frames and possibly a final HEADERS frame for trailers. After sending a request, a client closes the stream for sending.
- **HTTP Response** - The server sends back its HTTP response on the bidirectional stream. A HEADERS frame, a series of DATA frames and possibly a trailing HEADERS frame.
- **QPACK headers** -The HEADERS frames contain HTTP headers compressed using the QPACK algorithm. QPACK is similar in style to the HTTP/2 compression called HPACK, but modified to work with streams delivered out of order. QPACK itself uses two additional unidirectional QUIC streams between the two end-points. They are used to carry dynamic table information in either direction.

HTTP/3

- HTTP/3 server push is similar to what is described in HTTP/2, but uses different mechanisms.A server push is effectively the response to a request that the client never sent!
- Server pushes are only allowed to happen if the client side has agreed to them.
- In HTTP/3 the client even sets a limit for how many pushes it accepts by informing the server what the max push stream ID is. Going over that limit will cause a connection error.

HTTP/3

- If the server deems it likely that the client wants an extra resource that it hasn't asked for but ought to have anyway, it can send a **PUSH_PROMISE** frame (over the request stream) showing what the request looks like that the push is a response to, and then send that actual response over a new stream.
- Even when pushes have been said to be acceptable by the client before-hand, each individual pushed stream can still be canceled at any time if the client deems that suitable. It then sends a **CANCEL_PUSH** frame to the server.

Table of Contents

1 DNS
- Introduction
- Messages
- Practice

2 Hypertext Transfer Protocol
- Introduction
- HTTP/2
- HTTP/3

3 Transport Layer Security

4 DNS-over-HTTPS

TLS

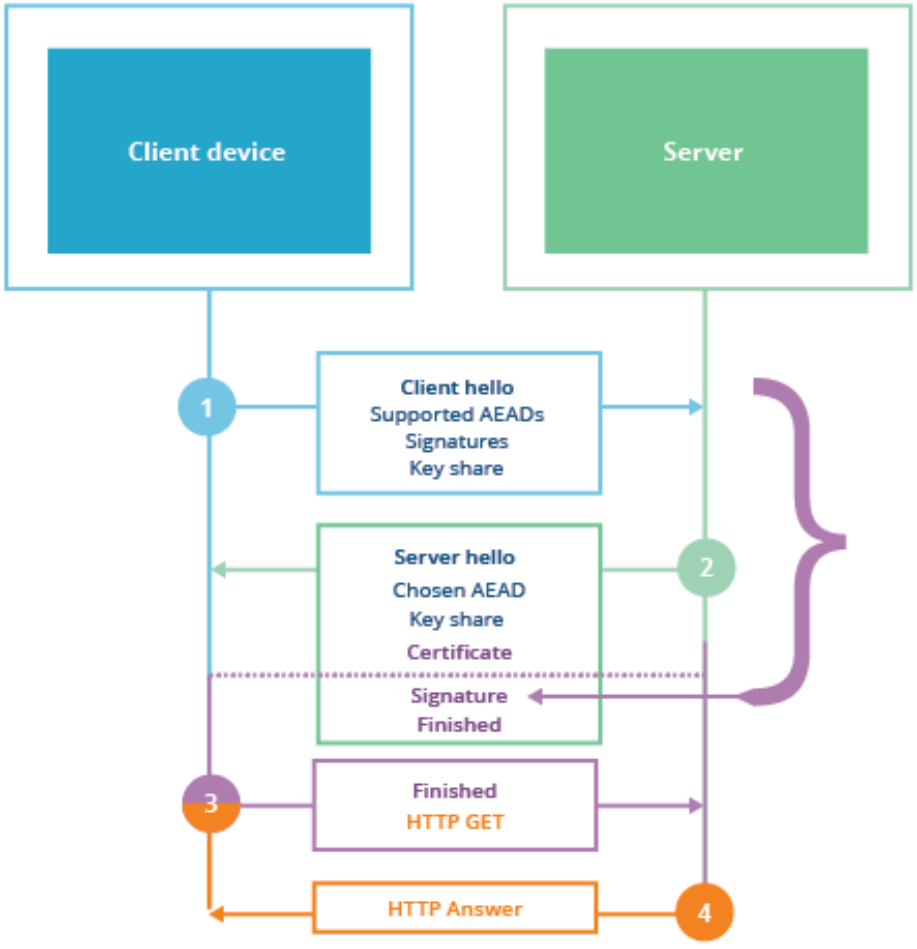
- Transport Layer Security, or TLS, is a cryptographic protocol that protects data exchanged over a computer network. TLS has become famous as the S in *HTTPS*.
- TLS was designed as a more secure alternative to its predecessor *Secure Sockets Layer* (SSL). Over the years, security researchers have discovered heaps of vulnerabilities affecting SSL, which motivated IETF to design TLS in an effort to mitigate them.

TLS

TLS versions:

- TLS 1.0 was published as RFC 2246 in 1996
- TLS 1.1 was published as RFC 4346 in 2006
- TLS 1.2 was published as RFC 5246 in 2008
- TLS 1.3 was published as proposed standard in RFC 8446 in 2018.

TLS


Client device
Server
1
Client hello
Supported AEADs
Signatures
Key share
2
Server hello
Chosen AEAD
Key share
Certificate
Signature
Finished
3
Finished
HTTP GET
HTTP Answer
4

TLS

TLS 1.3 security

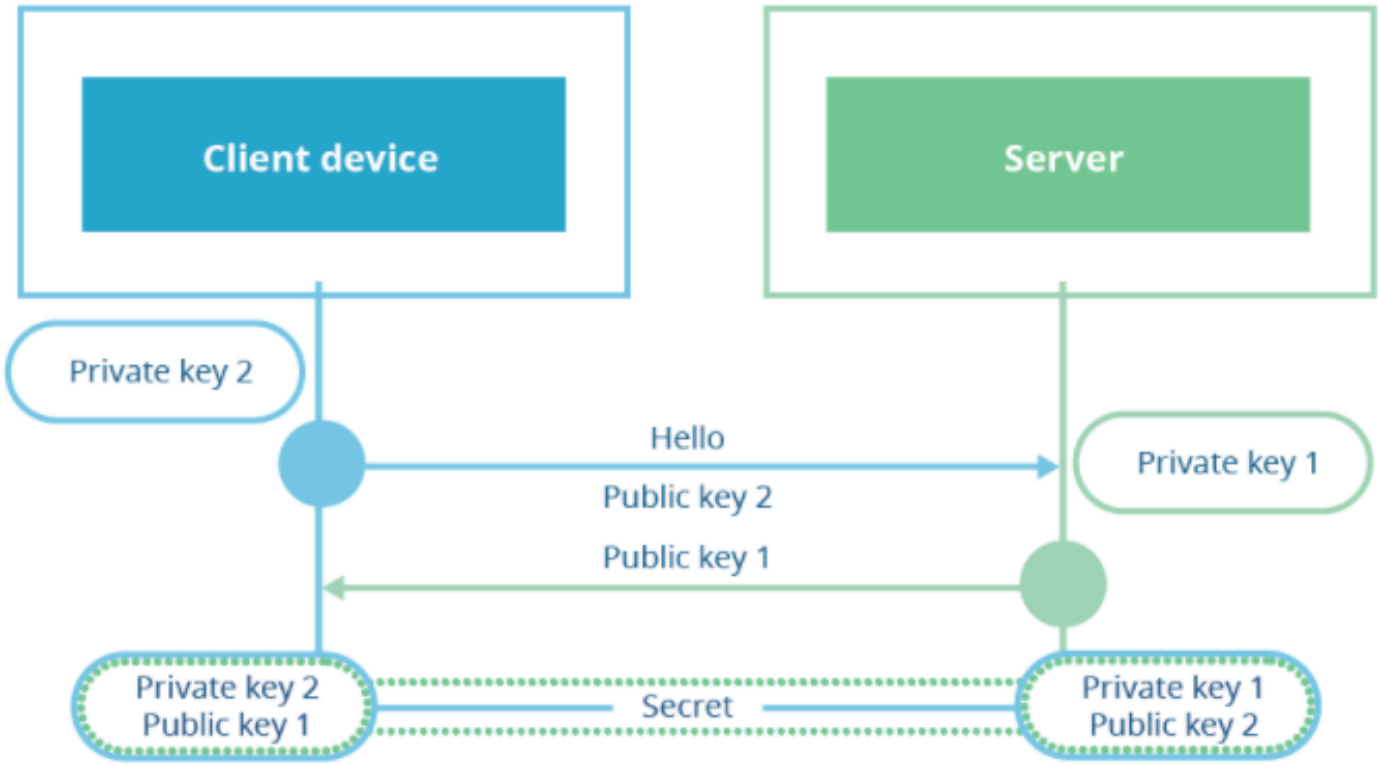
- In the new version, all key exchange algorithms, except the *Diffie-Hellman* (DH) key exchange, were removed.
- TLS 1.3 no longer supports unnecessary or vulnerable ciphers, such as CBC-mode and the RC4 cipher.
- TLS 1.3 requires servers to cryptographically sign the entire handshake, including the cipher negotiation, which prevents attackers from modifying any handshake parameters.
- The signatures themselves were also improved by implementing a new standard, called RSA-PSS.

TLS

TLS performance

- Besides improved security, the reduced set of parameters and the simplified handshake in TLS 1.3 also contributes to improving overall performance. Since there is only one key exchange algorithm and just a handful of supported ciphers, the absolute bandwidth required to set up a TLS 1.3 channel is considerably less than earlier versions.
- Furthermore, TLS 1.3 now supports a new handshake protocol, called *1-RTT mode*. In 1-RTT, the client can send DH key shares in the first handshake message, because it can be fairly certain of the parameters the server will use.

TLS


Client device
Server
Private key 2
Hello
Public key 2
Private key 1
Public key 1
Private key 2
Public key 1
Secret
Private key 1
Public key 2

TLS

- Instead of negotiating the parameters first and then exchanging keys or key shares, TLS 1.3 allows a client to set up a TLS channel with only one round-trip transaction.
- With another TLS 1.3 feature, called the *0-RTT Resumption mode*, when a browser visits a server for the first time and successfully completes a TLS handshake, both the client and the server can store a pre-shared encryption key locally.
- If the browser visits the server again, it can use this resumption key to send encrypted application data in its first message to the server.

TLS

TLS privacy

- Learning from past mistakes, TLS 1.3 now offers an extension that encrypts SNI information. When used correctly, this extension prevents attackers from leaking the remote server's domain name, so they have no method of tracking HTTPS user history. This feature provides greater privacy to Internet users than previous versions of TLS.

Table of Contents

1. DNS
   - Introduction
   - Messages
   - Practice
2. Hypertext Transfer Protocol
   - Introduction
   - HTTP/2
   - HTTP/3
3. Transport Layer Security
4. DNS-over-HTTPS

DoH

The **DNS-over-HTTPS** protocol is a recent invention. It was created a few years back and was proposed as an internet standard 2018 October. The protocol itself works by changing how DNS works. DoH encrypts DNS queries, which are disguised as regular HTTPS traffic - hence the DNS-over-HTTPS name. These DoH queries are sent to special DoH-capable DNS servers (called DoH resolvers), which resolve the DNS query inside a DoH request, and reply to the user, also in an encrypted manner.

DoH

The HTTP Request

- A DoH client encodes a single DNS query into an HTTP request using either the HTTP GET or POST method and the other requirements of this section. The DoH server defines the URI used by the request through the use of a URI Template.
- The URI Template is processed without any variables when the HTTP method is POST. When the HTTP method is GET the single variable "dns" is defined as the content of the DNS request, encoded with base64url.
- DoH servers MUST implement both the POST and GET methods. When using the POST method the DNS query is included as the message body of the HTTP request and the Content-Type request header field indicates the media type of the message. POST-ed requests are generally smaller than their GET equivalents.

DoH

```
:method = GET
:scheme = https
:authority = dnsserver.example.net
:path = /dns-query?dns=AAABAAABAAAAAAAAA3d3dwdleGFtcGxlA2NvbQAAAQAB
accept = application/dns-message
```

DoH

```
:method = POST
:scheme = https
:authority = dnsserver.example.net
:path = /dns-query
accept = application/dns-message
content-type = application/dns-message
content-length = 33

<33 bytes represented by the following hex encoding>
00 00 01 00 00 01 00 00  00 00 00 00 03 77 77 77
07 65 78 61 6d 70 6c 65  03 63 6f 6d 00 00 01 00
01
```

DoH

The HTTP Response

- The only response type is "application/dns-message", but it is possible that other response formats will be defined in the future.
- Different response media types will provide more or less information from a DNS response.
- The amount and type of information that a media type gives is solely up to the format, and not defined in this protocol.
- DNS response codes indicate either success or failure for the DNS query. A successful HTTP response with a 2xx status code is used for any valid DNS response, regardless of the DNS response code.
- For example, a successful 2xx HTTP status code is used even with a DNS message whose DNS response code indicates failure, such as **SERVFAIL** or **NXDOMAIN**.

DoH

```
:status = 200
content-type = application/dns-message
content-length = 61
cache-control = max-age=3709

<61 bytes represented by the following hex encoding>
00 00 81 80 00 01 00 01  00 00 00 00 03 77 77 77
07 65 78 61 6d 70 6c 65  03 63 6f 6d 00 00 1c 00
01 c0 0c 00 1c 00 01 00  00 0e 7d 00 10 20 01 0d
b8 ab cd 00 12 00 01 00  02 00 03 00 04
```

DoH

HTTP/2 is the minimum **RECOMMENDED** version of HTTP for use with DoH. The messages in classic UDP-based DNS are inherently unordered and have low overhead. A competitive HTTP transport needs to support reordering, parallelism, priority, and header compression to achieve similar performance. Those features were introduced to HTTP in HTTP/2.