ceedcv
CENTRE ESPECÍFIC
D´EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

UT 12.
NETWORK TOOLS AND PROTOCOLS

Computer Systems
CFGS DAW

Aarón Martín Bermejo
a.martinbermejo@edu.gva.es

2022/2023
Version:230313.1617

License

Nomenclature

Throughout this unit different symbols will be used to distinguish important elements within the content. These symbols are:

Important

Attention

Interesting

TABLE OF CONTENTS

UT 12. NETWORK TOOLS AND PROTOCOLS

1. INTRODUCTION

As we've studied before, computer networks are a huge part of the pillars of our actual highly technologically based society. Communications, finance, working remotely… almost every aspect of our daily life is connected to one of those aspects.

That's why we've studied networking features, classifications, architectures… In this topic we are going to get deeper into how computer networks have modeled the communication in order to make it efficient, safe and reliable under extreme conditions such as high availability, great stress, changing or failing infrastructure, etc. Those models of the communications are called **protocols** in computer networks.

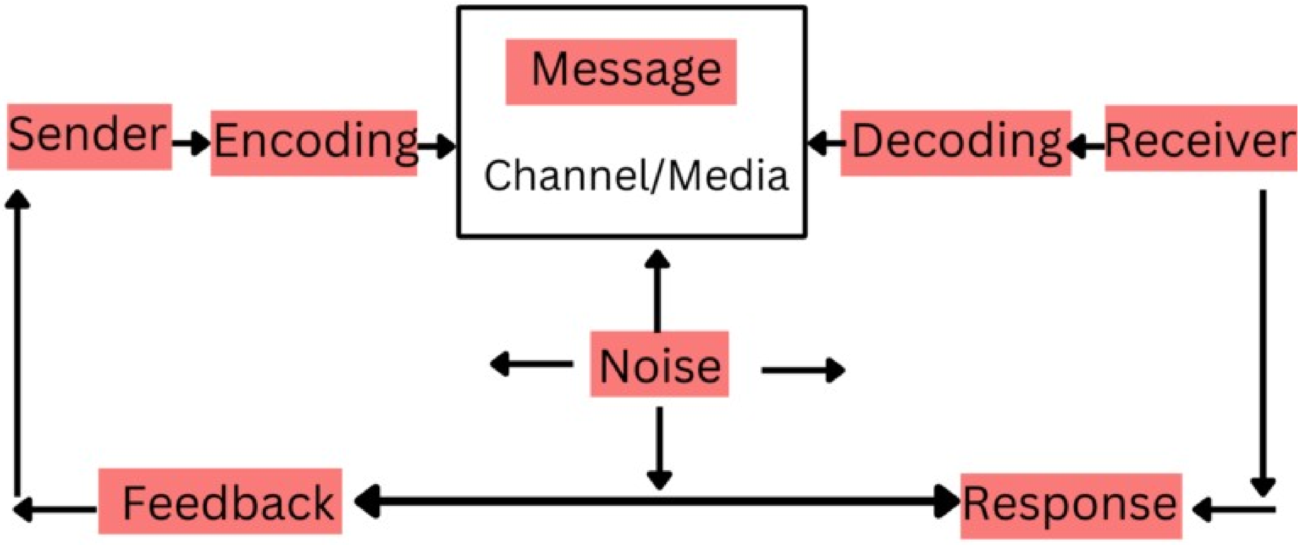
Also, we will cover some tools in order to build a toolset to work with those protocols and under computer networks.

2. PROTOCOLS

In computer networks, a definition for protocol can be a set of rules that specify how the messages should be formatted and how they should be sent, received and processed. Therefore, we have two parts of the definition of the protocol:

1. Format of the messages
2. How to process the data

The goal of a computer network protocol is to model the communication between computers. Communication is always based on the next elements:



1. *Communication elements: https://clearinfo.in/blog/elements-of-communication-process/*

In the figure we can see these basic elements:

1. Sender: the one sending the information
2. Receiver: the one receiving the information
3. Message: the information to be transmitted

These are the three basic elements of communication and, obviously, without them there won't be any communication. But there are more elements on the figure that are extremely important in communication, but specially in network computers:

- Channel/Media: through where or what the message is transmitted. Examples of channels are the air, a cable, paper...
- Encoding/Decoding: how the message is encoded and how it's decoded. Examples of encodings can be:
  - Any human language: spanish, english, sign language, body expression...
  - Machine "languages": binary, hexadecimal, ASCII...

It's any code that allows the sender to represent the information encoding it with some system and the receiver to understand it by decoding the information by the same system as the sender or some other system.

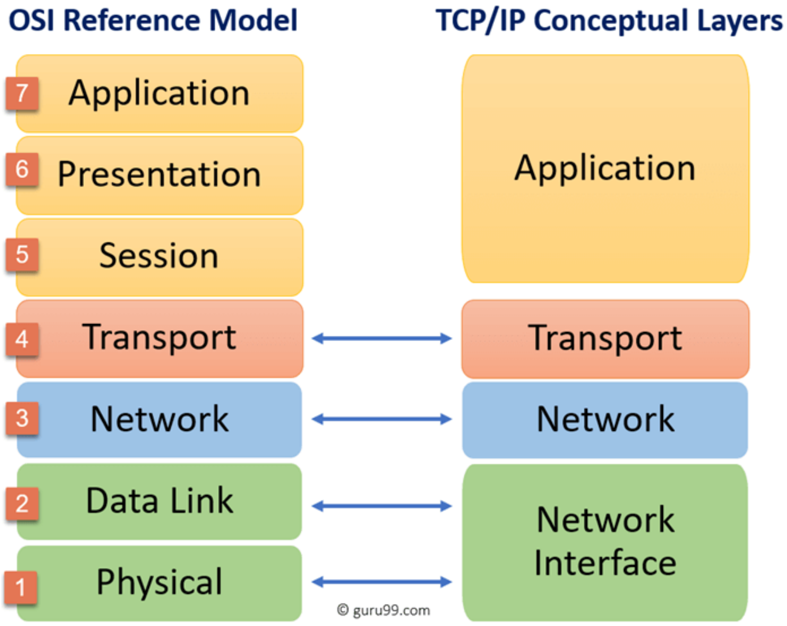
- Response/Feedback: it's information that the receiver send back to the sender. It can be a response to the originally sent message "hello – bye" or it can be any other kind of response or feedback like acknowledging the receipt of the message.
- Noise: the big enemy of communication. Noise speaks about any interference that affects the communication causing issues on it. It can be physical noise that does not allow two people to hear themselves or it can radio interferences, saturation of the Wifi radiofrequencies, etc.

These specific elements important in any communication but are of extreme importance in network computing, since they need to be solved in a way computers can process efficiently.

Computer network protocols will try to specify how computers need to address these communication elements in order to be able to send information.

Those protocols need to be standardized in order to allow computers to send and receive information so it can be understood. Exactly as in human language, if each person had its own language no one would be able to understand anything.

That's what OSI and TCP/IP try to solve. Remembering about OSI and TCP/IP was and which levels they define:



Different protocols intervene in the different layers of both models in order to solve the specific problems they require. In this unit we are going to focus

3. DNS

Domain Name System or DNS is the system that allows to translate from domain names[1] to IPs. While computers talk to each other by using IPs as the way of adressing each other, for humans remembering each IP number of each website or web service that we want to use would be unbearable. That's the reason of existence for DNS.

> 📢 Do you think you will access that much to aules if you needed to remember that its IP is https://213.0.87.122/ED

In order to ask for something using a domain name and being able to know to which IP that domain targets, some steps need to be performed to fulfill the DNS lookup or DNS resolution:

1. First, your computer will look into the **hosts file**. Hosts file is a plain text file that contains information about how hostnames are mapped IPs in your computer. In Linux hosts file is stored in /etc/hosts and in windows inside c:\Windows\System32\Drivers\etc\hosts. It typically contains the localhost redirection:

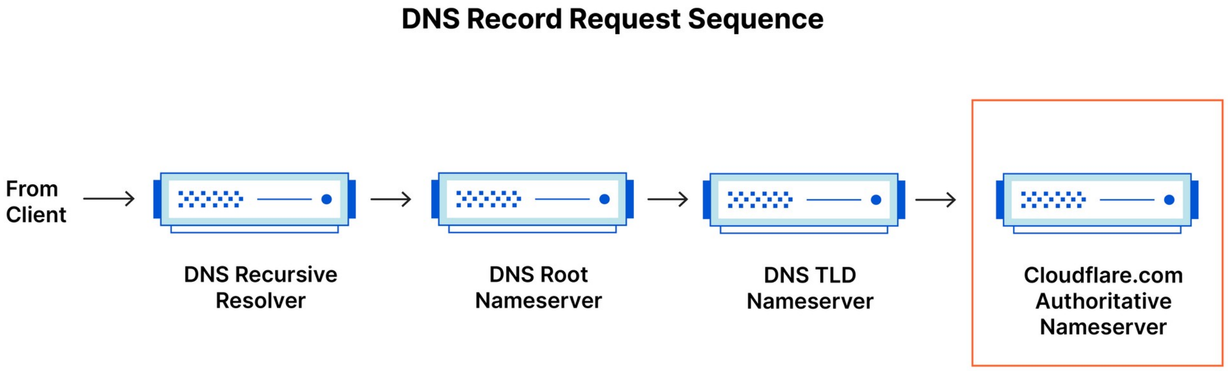
```
127.0.0.1   localhost
127.0.1.1   aamarber-MS-7996
```

   It can be edited to contain more hosts and it's super useful specially for developing, but also to manually configure the domain name resolution.

2. If the domain is not found in the hosts file, the **browser cache** will be checked in case that domain was resolved previously and that information can be reused.
3. Then, if not found neither at the hosts file or the browser cache, **DNS Resolver** is called to resolve the host's IP. DNS resolver will look into its cache just in case there was a previous request to solve that specific domain name. If not, it will redirect to the corresponding **Root Nameserver** based on the domain name**.**
4. **Root nameservers** are at the top of the DNS hierarchy, but are not the ones that hold the information on how to translate a domain to an IP. Instead, they have the information about where to look for it at the **TLD nameserver.**
5. **TLD nameservers** are responsible of managing the information of **top level domains (TLD)** like .com, .net, .org... and hold the information of the specific top level domain **Authoritative Nameserver**
6. The **authoritative nameserver** is the final step in the DNS lookup and are the servers that hold all the information about a domain, including its IP. It will be

[1] https://developer.mozilla.org/en-US/docs/Glossary/Domain_name

the server that will answer to the DNS lookup in the final step if no other step was able to.



Cloudflare: whatis DNS[2]

Even though it looks pretty complex, that process is performed millions and billions of times each day extremely fast, and it’s one of the pillars of the modern web.

But one important thing related to this “complex” process is that, as it has those steps and each one needs to talk with different servers with their own caches, changing the IP of a hostname can be a process that takes up until 24 hours.

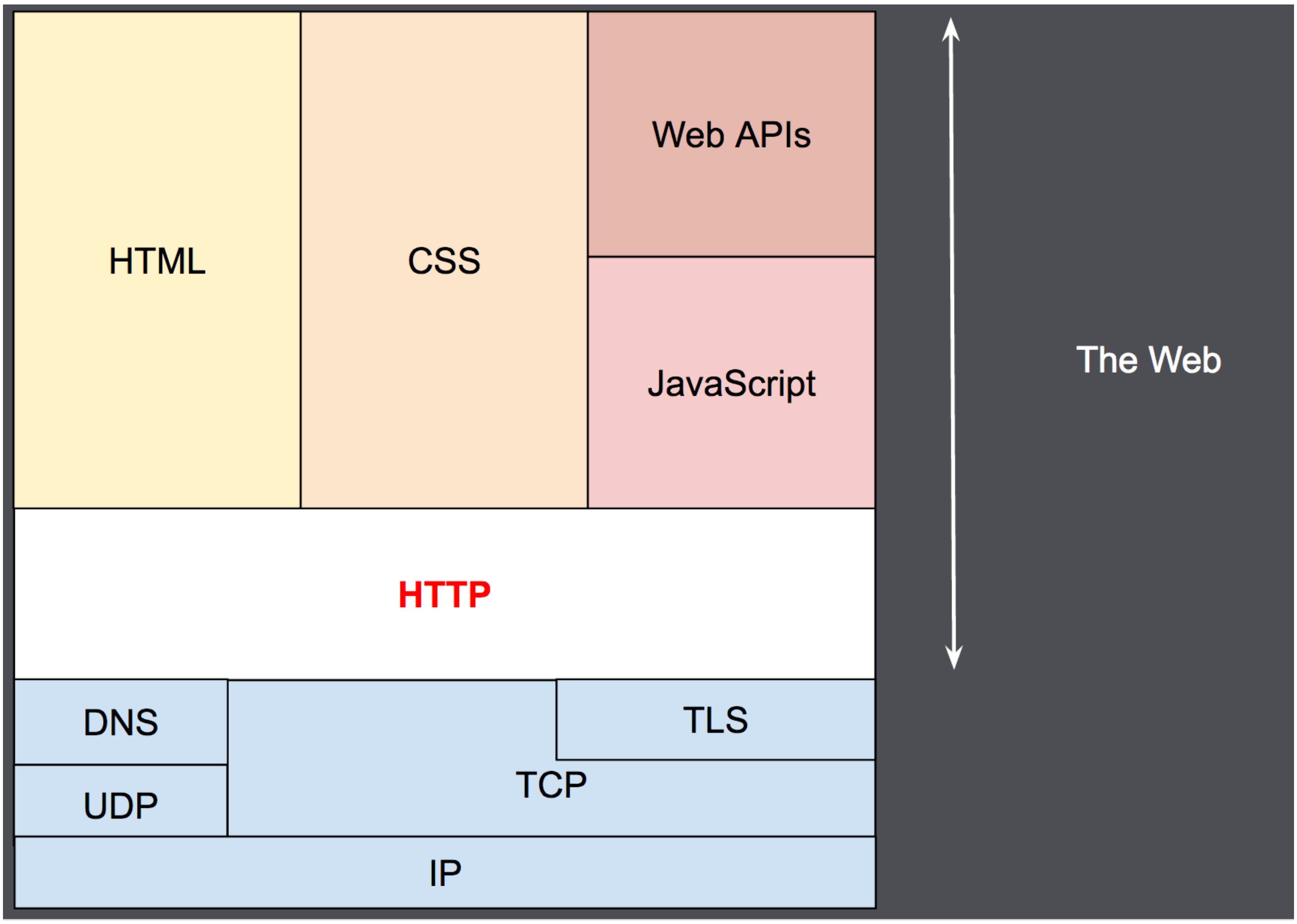
> 🗲 Remember that delay when you need to change your domain name authoritative nameserver, for instance

[2] https://www.cloudflare.com/learning/dns/what-is-dns/

4. HTTP

HTTP is a protocol that belongs in the application layer of the TCP/IP model and it's meant to fetch resources over the network. It is a client server protocol, which means that the client is the one initiating the communication and the server responds to it. It's basic for the web interchange of data.

In HTTP individual messages are interchanged. Client messages are called "requests" and server messages are called "responses", for obvious reasons.



https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

HTTP relies on different protocols that are under it, both of the same layer and of different layers obviously. Specially important is TCP and its encrypted version TLS, since HTTP communication is usually done through TCP. It can be done through UDP, but it's not the default communication protocol to use to send HTTP data.

HTTP has evolved since its creation in 1989-1991 by Tim Berners Lee until nowadays. First standard version was HTTP/1.1in 1997 and during 15 years it wqas the standard with some extensions to it. That amount of time without being radically modified means that HTTP was extremely stable and well defined and that it can answer to the needs of the web for a very long time. In May 2015 HTTP/2 was standardized with some major features to adapt to the current usage of the web and by January 2022 was being used by 46.9% websites[3] HTTP/3 came out in June 2022 with the major feature that it uses QUIC (another transport protocol) instead of TCP.

4.1 HTTP Messages

As we've spoken when talking about communication, one important part of it is the encoding and decoding of the messages. Therefore, HTTP specifies that encoding of the messages.

In HTTP, there are two types of messages: **requests** and **responses**.

4.1.1 REQUESTS

Requests are formatted as it follows:

1. **Start line**: HTTP messages are sent by the client to initiate an action on the server. They contain three items

   1.1. **HTTP method**: a verb or noun that describes the action to do. GET, POST, PUT, DELETE, OPTIONS are examples of valid HTTP methods.

   1.2. **Request target**: usually, the URL but it can be also an IP (and/or a port) or any other way to adress a server.

   1.3. **HTTP Version**: both for specifying the structure of the rest of the message and the expected HTTP version of the response.

   Some examples of the start line could be:

   GET www.google.es HTTP/1.1

   POST /register_user HTTP/2.0

   GET 127.0.0.1:8081/index.html HTTP/1.1

2. **Headers:** format of the headers is always a case insensitive string followed by a double dot. There are many of them but they can be generally classified among two general groups:

   2.1. Request headers: specify the request like stating which is the origin of the request (host: localhost), the user-agent or the encoding that the client will accept as a response (accept: text/html).

   2.2. Representation headers: describe the message, for instance any encoding applied or its content length.

[3] https://w3techs.com/technologies/details/ce-http2

3. **Body:** optional, since some request methods don't need a body (like GET). It's the data sent to the server if needed, like the data to POST or PUT.

4.1.2 RESPONSES

Responses are formatted as it follows:

1. **Status line**: first line of the HTTP response contains:

   1.1. Protocol version, as in the request.

   1.2. **Status code:** indicates success or failure of the request and usually why, specifying information about it. Common codes are:

   200: success

   404: not found

   401: not authorized

   All HTTP status codes are specified in the standard RFC 9110[4][5] but sometimes custom status codes are used.

2. Headers: http headers for responses follow the same format as in requests, a case insensitive string followed by double dot ":" and the value of the header. As in requests, HTTP response headers can be classified in two categories:

   1.1. Response headers: headers like "server" or "age"

   1.2. Representation headers: as in the request, describe the message, for instance any encoding applied or its content length.

   2.2. **Body**: some responses do not contain a body, because the response only requires a status code. But in most cases the response body is filled with the information asked in the request.

4.1.3 HTTP/1.x vs HTTP/2 messages

HTTP messages under HTTP/1.x are written in plain text (not encoded in any way) and are not splitted in any way. That means that HTTP/1.x messages bot requests and responses are sent fully in plain texts, which has the advantage that they can be read easily but they have great disadvantages:

- Can't be split easily so they can be multiplexed, since they are written in plain text.
- Can't be compressed easily or, at least, not in the most performant way.

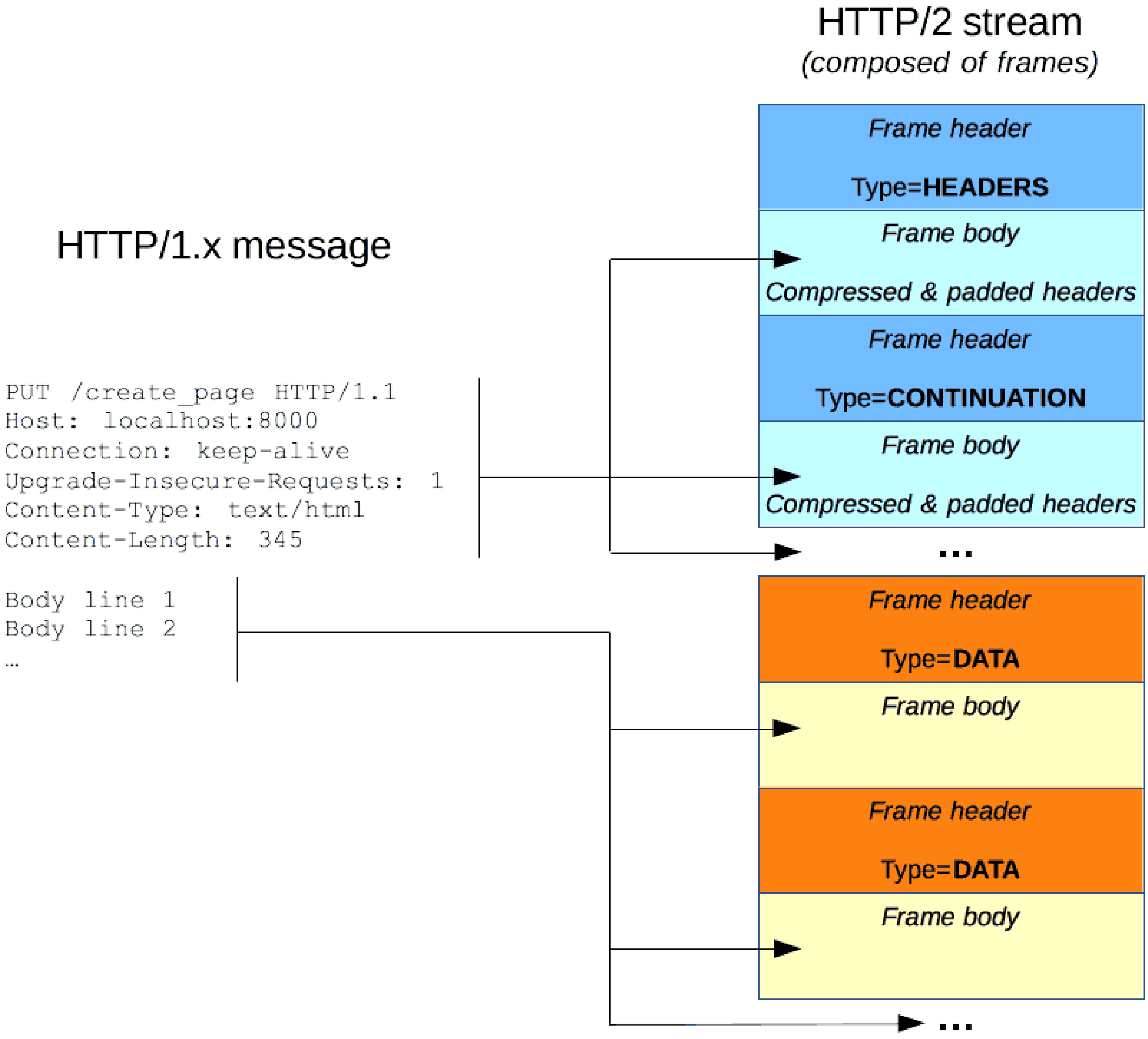
Those two disadvantages lead to some changes in the encoding of the messages in HTTP/2:

[4] https://httpwg.org/specs/rfc9110.html#overview.of.status.codes

[5] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

- HTTP/2 messages are sent in binary, instead of plain text
- HTTP/2 messages are split in frames and sent in a stream

Both changes have the purpose of improving the performance by compressing and multiplexing the interchange of information.



```
PUT /create_page HTTP/1.1
Host: localhost:8000
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: text/html
Content-Length: 345

Body line 1
Body line 2
...
```

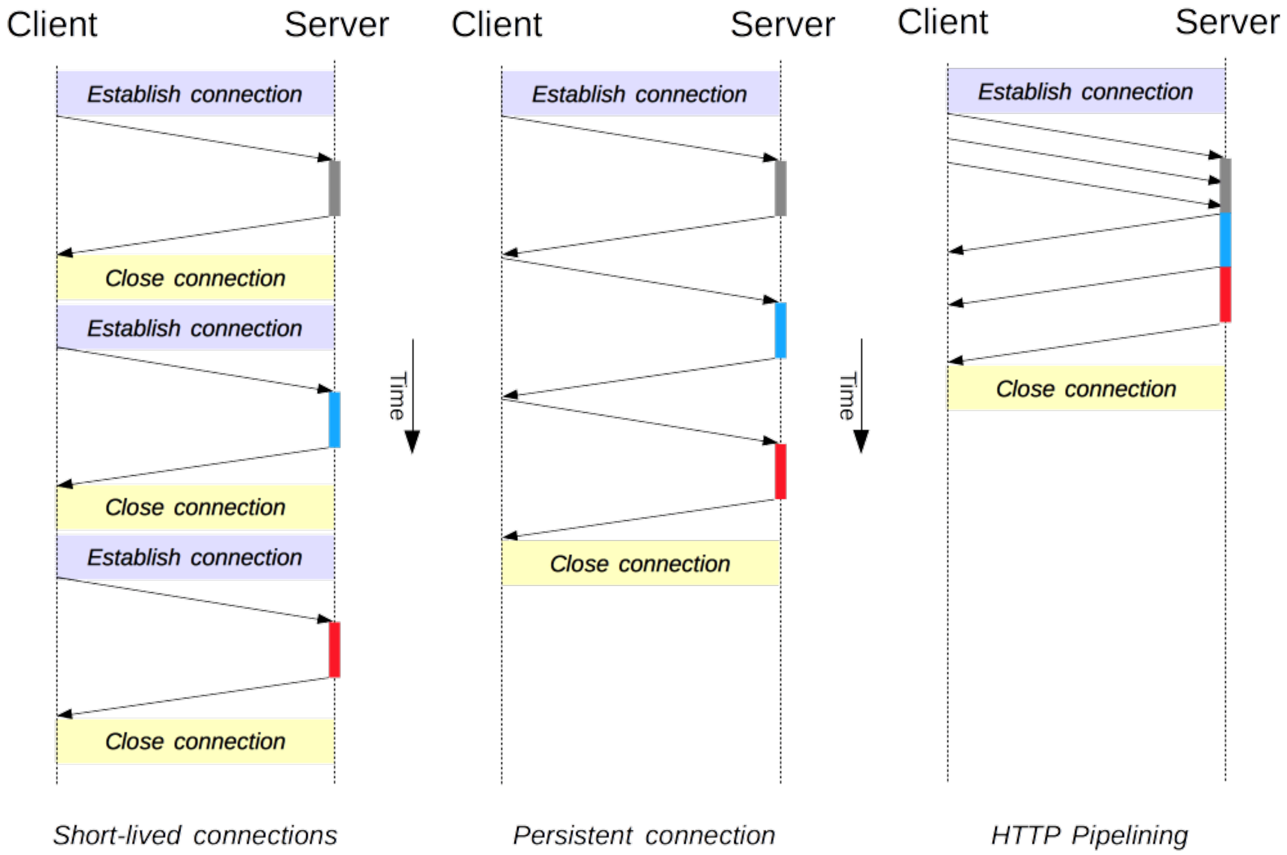
MDN: HTTP/1.x message vs HTTP/2 messages

4.2 HTTP Flow

The flow of communication in HTTP usually has three simplified steps:

1. The client establishes the connection with the server
2. Client sends the request
3. Server manages the request and sends back an answer

Those three basic steps that look so simple need to solve complex issues due to the high demand of the web requests world. Specially, managing the connection is a problem, since establishing a TCP connection consumes resources. That's why there are different strategies to do it:



MDN Managing connections[6]

- Short connections: means that each time a request needs to be done, a new connection is created and therefore the establishment of the connection and its closing need to be performed.
- Persistent connections: instead of establishing a new connection for each request, requests are persisted to avoid the establishment and closing of them. That way, when several requests need to be made they can skip those steps.
- HTTP pipelining: instead of waiting for each request to be made, all of them are sent through the same connection and the server will answer to them as it can.

HTTP/1.x and HTTP/2 have some differences in that connection management issue, optimizing that issue by allowing more requests to be made at the same time or by using multiplexing instead of pipelining as the algorithm to manage.

[6] https://developer.mozilla.org/en-US/docs/Web/HTTP/Connection_management_in_HTTP_1.x

4.3 HTTPS, TLS and SSL

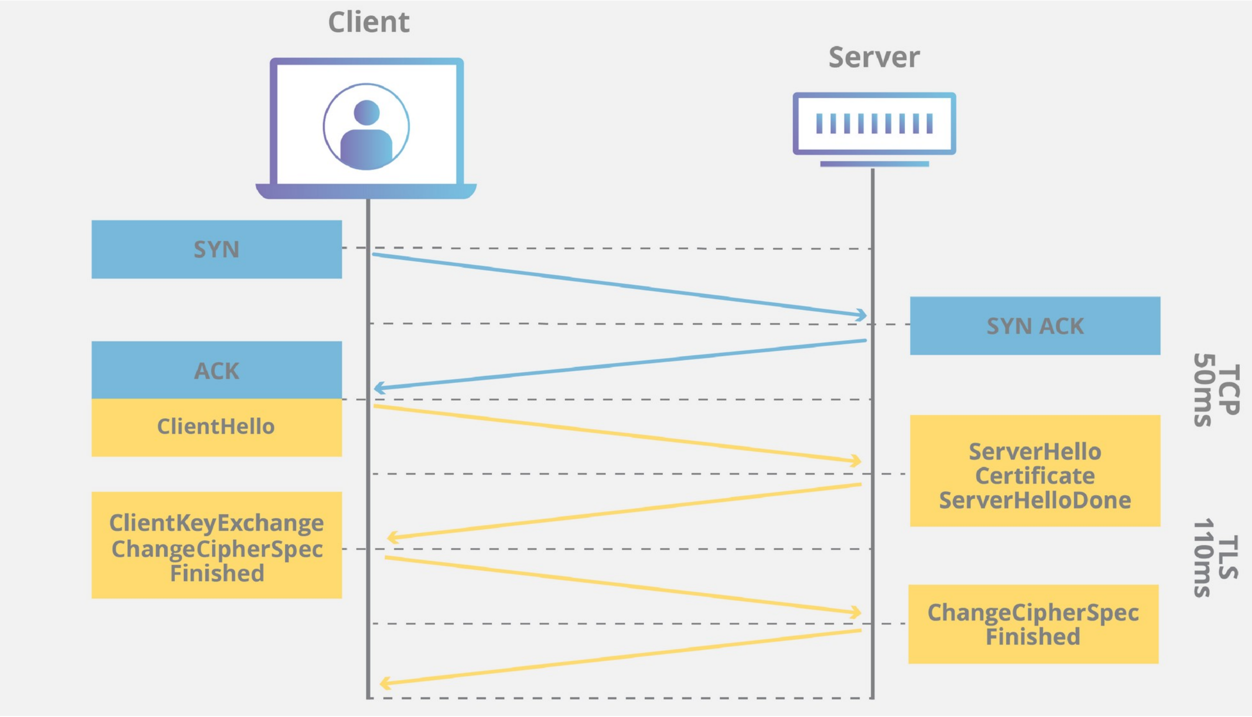
As in almost every aspect of computing, security is an important aspect of it. But for communication over networks, since the information sent can be pretty sensitive and that information travels through tons of unknown different computers that can intercept that information, giving a layer of security to it it's almost mandatory.

That's why SSL or Secure Sockets Layer was developed by Netscape as an encryption-based internet security protocol. It's meant to address problems like ensuring privacy, authentication and encryption of the information sent through networks, specially in HTTP. In fact, what HTTPS stands for is Hypertext Transfer Protocol **Secure** and means that the website implements SSL/TLS.

Wait… TLS? TLS or Transport Layer Security is a protocol developed by the IETF (Internet Engineering Task Force) as an evolvement of SSL. And as SSL was developed by a private company (Netscape) which was not involved in the development of the new standard, they changed the name. But as TLS was based in SSL both are usually confused.

TLS works as it follows:

1. First an authentication process between the client and the server is started. It's known as a **TLS handshake** and works as the following figure specifies:



TLS handshake[7]

2. After that, as both parts have the keys to cypher information, all communication will be cyphered between the client and the server,

[7] https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/

3. Also, all data is signed to verify that even though it’s encrypted, the data holds its integrity and it was not tainted.

In order for a website to implement TLS it needs a security certificate. A certificate has two important aspects:

1. Is like an ID Card that ensures that the server that we’re talking to is the one that it states.
2. It contains the **public** and the **private key**.
   - 2.1. The public key is exposed to any client, so they can cypher the information sent to the server and no one is able to peek at it.
   - 2.2. The private key is kept secret by the server and it’s the key that allows the server to decypher the information sent by the client.

There are also three types of security certificates:

1. Single-domain: can only be used for a specific domain
2. Wildcard: it’s tied to a single domain, but it can be used for subdomains like “aules.edu.gva.es” “portal.edu.gva.es” and so on.
3. Multidomain: issued for multiple domains

Most modern browsers mark non HTTPS websites as insecure and usually they are positioned way worse in search engines if security is not used. So it’s a must-do if you create or manage a modern website.

RESOURCES

1. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
2. https://www.cloudflare.com/learning/dns/what-is-dns/