Tecnologie e applicazioni web

HTTP

Filippo Bergamasco (filippo.bergamasco@unive.it)
http://www.dais.unive.it/~bergamasco/
DAIS - Università Ca'Foscari di Venezia

Academic year: 2023/2024

The web communication protocol

HTTP: **H**yper**T**ext **T**ransfer **P**rotocol is the principal communication protocol for the web

- Allowed the rapid development of the web
- Nowadays, it is used together with other protocols and standards (like WebSocket) but still plays a fundamental role

The web communication protocol

HTTP: **H**yper**T**ext **T**ransfer **P**rotocol is the principal communication protocol for the web.

Defines semantic, syntactic, and synchronization rules to exchange data

The web communication protocol

HTTP: **H**yper**T**ext **T**ransfer **P**rotocol is the principal communication protocol for the web.

Born to exchange hypertexts but not limited to this type of content

The web communication protocol

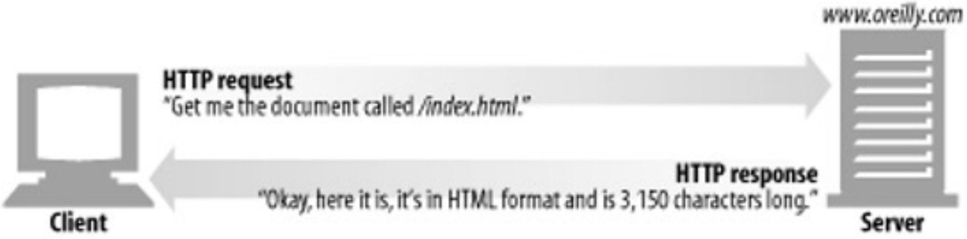
HTTP: **H**yper**T**ext **T**ransfer **P**rotocol is the principal communication protocol for the web.

Design to transfer content between client and server.

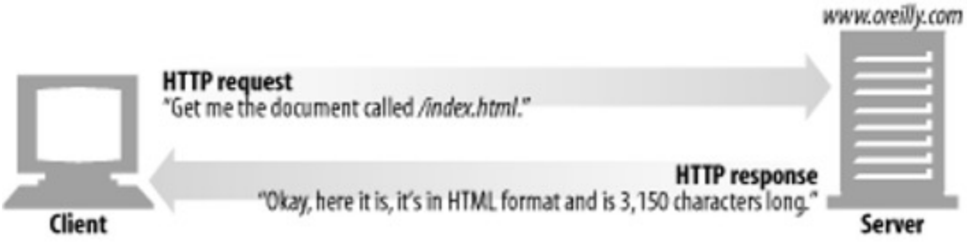
Base concepts

Client-Server model

- HTTP is a **request-response** protocol with a **client-server** communication model

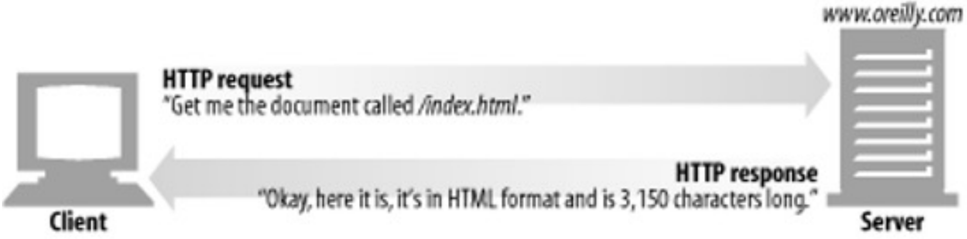

www.oreilly.com
HTTP request
"Get me the document called /index.html."
HTTP response
"Okay, here it is, it's in HTML format and is 3,150 characters long."
Client
Server

Client-Server model


www.oreilly.com
HTTP request
"Get me the document called /index.html."
HTTP response
"Okay, here it is, it's in HTML format and is 3,150 characters long."
Client
Server

A typical HTTP **client** is the web **browser**. The core browser functionality is to ask resources to the server (usually HTML pages) to visualize it on screen

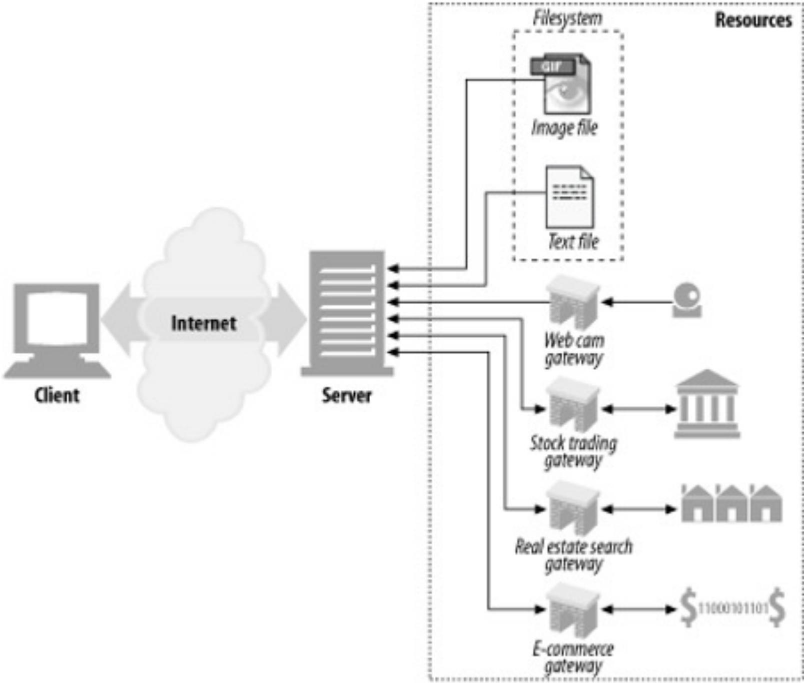
Client-Server model


www.oreilly.com
HTTP request
"Get me the document called /index.html."
HTTP response
"Okay, here it is, it's in HTML format and is 3,150 characters long."
Client
Server

An HTTP **server** is a software that responds to the requests performed by one or more clients.

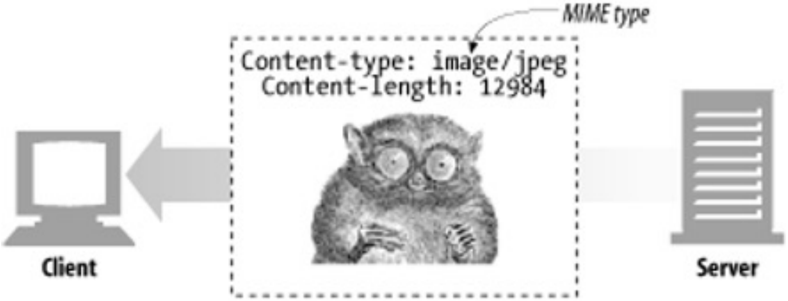
Resources

- The HTTP server hosts **resources**. Resources can be static or generated on-demands by other programs.



Resource type

- Every resource has an associated type, called **MIME** (Multipurpose Internet Mail Extension). The type describes the resource format (image, text, etc)



Resource type

Mime type is simply a string, formatted as:
`<type>/<subtype>`

For example:

- `text/html` is the mime type for an HTML document
- `text/plain` is the mime type for a simple text file with ASCII encoding
- `image/jpeg` is the mime type for a JPEG image
- `application/JSON` is the mime type for JSON strings
- etc.

Resource identifier

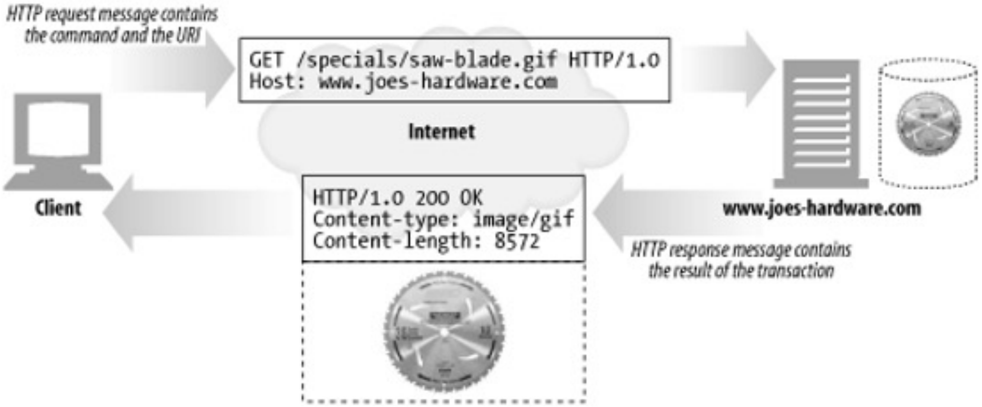
- Each resource has a name, called **URI** (Uniform Resource Identifier), to uniquely identify it on the web.

  Two kinds of URI: **URL** e **URN**

  - **U**niform **R**esource **L**ocator identifies a resource by specifying its location.
  - **U**niform **R**esource **N**ame identify a resource using a unique name regardless the location (not used in practice).

Request & Response

HTTP consists of a sequence of **transactions** comprising a **command** (client ➡️ server) called **request**, followed by a **response** (client ⬅️ server), formatted in an HTTP **message.**



```
GET /specials/saw-blade.gif HTTP/1.0
Host: www.joes-hardware.com
```

```
HTTP/1.0 200 OK
Content-type: image/gif
Content-length: 8572
```

HTTP request message contains the command and the URI
Internet
Client
www.joes-hardware.com
HTTP response message contains the result of the transaction

Methods

- HTTP supports different commands called **methods**
  - Every message sent to a server must specify a method
  - 5 common methods:
    - GET
    - PUT
    - DELETE
    - POST
    - HEAD

Status code

- Every response message contains a status code:
  - 3 digits number to notify the client if the request can be satisfied or more actions are needed
  - status code examples:
    - **200** (Success)
    - **302** (Redirect required)
    - **404** (The resource does not exist)

Protocol versions

- HTTP/0.9
  - First prototype released in 1991. Supports only the GET method and is bugged.
- HTTP/1.0
  - First version to be adopted on a large scale. Adds different methods and MIME types

Protocol versions

- HTTP/1.0+
  - Added support for keep-alive connections, virtual hosting and proxies
- HTTP/1.1
  - Corrects some bugs add adds many optimizations to improve the performance. It is the currently used version.

Protocol in detail

URL

A Uniform Resource Locator (URL) is a standard name to define and identify resources inside the web.

- Usually, it represents the “entry point” from which the users start browsing the web.
- Encodes the resource **name**, the **location** where to find it, and **how** the browser can retrieve it
- The general URL format comprises 9 distinct parts (not all parts are mandatory)

URL structure

```
<scheme>://<user>:<password>@<host>:<port>/<path>
                                      ;<params>?<query>#<frag>
```

Ex: http://www.joes-hardware.com/seasonal/index-fall.html

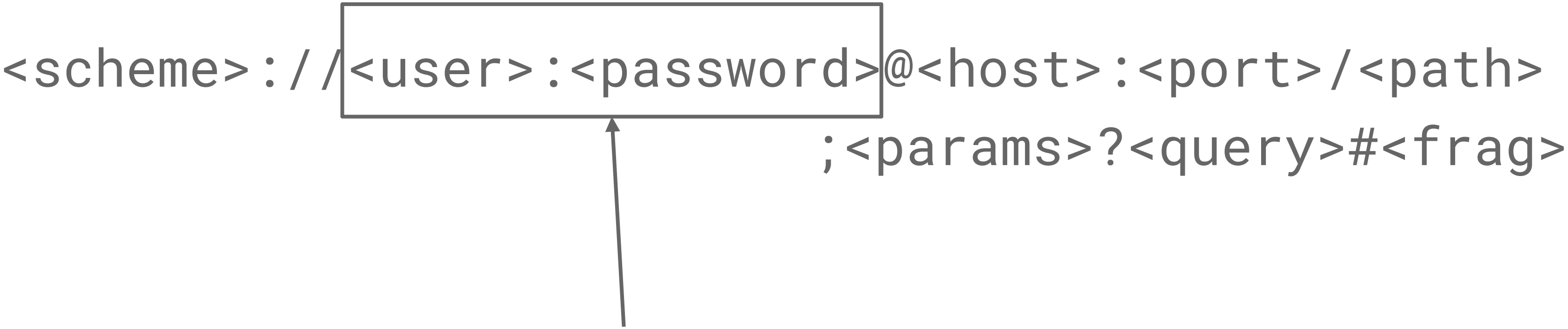
URL structure

```
<scheme>://<user>:<password>@<host>:<port>/<path>
                                    ;<params>?<query>#<frag>
```

The scheme specifies the protocol to use to retrieve the resource from the server

Ex: http, https, ftp, rtsp, etc.

URL structure

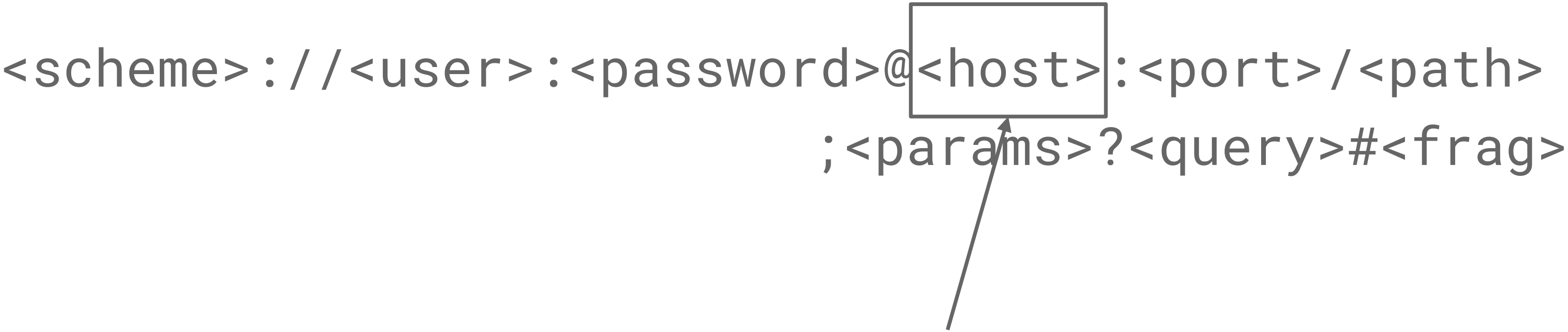


```
<scheme>://<user>:<password>@<host>:<port>/<path>
                                ;<params>?<query>#<frag>
```

User and password can sometimes be requested to authenticate with the server.

If not specified, the default user is “anonymous”

URL structure



```
<scheme>://<user>:<password>@<host>:<port>/<path>
                              ;<params>?<query>#<frag>
```

The server address can be a hostname (resolved via DNS) or an IP address. The host must always be specified.

URL structure

```
<scheme>://<user>:<password>@<host>:<port>/<path>
                              ;<params>?<query>#<frag>
```

The server port on which the server is listening for a connection. The port can be omitted depending on the scheme, and a default value is used.

Ex: HTTP default port is 80

URL structure

```
<scheme>://<user>:<password>@<host>:<port>/<path>
                                 ;<params>?<query>#<frag>
```

The (local) resource name hosted by the server. Path syntax depends on the scheme. In HTTP, the path can be divided into multiple segments separated by /. Ex: seg1/seg2/seg3

URL structure

```
<scheme>://<user>:<password>@<host>:<port>/<path>
                              ;<params>?<query>#<frag>
```

This component allows to specify a list of “name-value” pairs to a path segment. This is useful to provide additional information to the webserver depending on the resource requested.

URL structure

```
<scheme>://<user>:<password>@<host>:<port>/<path>
                              ;<params>?<query>#<frag>
```

Ex:

```
ftp://prep.ai.mit.edu/pub/gnu;type=d
http://www.aa.com/hammers;sale=false/index.html;graph
ics=true
```

URL structure

```
<scheme>://<user>:<password>@<host>:<port>/<path>
                              ;<params>?<query>#<frag>
```

Similar to <params>, <query> is frequently used to restrict the context of a certain resource

URL structure

```
<scheme>://<user>:<password>@<host>:<port>/<path>
                        ;<params>?<query>#<frag>
```

Ex: `http://ww.a.com/inventory.html?item=1234`

In this case, the resource inventory.html is restricted to a hypothetical item number 1234

URL structure

```
<scheme>://<user>:<password>@<host>:<port>/<path>
                              ;<params>?<query>#<frag>
```

Ex: `http://ww.a.com/inventory.html?item=1234&num=3`

Name-value pairs can be separated by the **&** character

<params> or <query>?

```
<scheme>://<user>:<password>@<host>:<port>/<path>
                              ;<params>?<query>#<frag>
```

What's the difference between <params> e <query>?

- <params> refer to a path segment, query to a resource
- <params> is helpful to the web server while <query> the gateway generating the resource.

URL structure

```
<scheme>://<user>:<password>@<host>:<port>/<path>
                                   ;<params>?<query>#<frag>
```

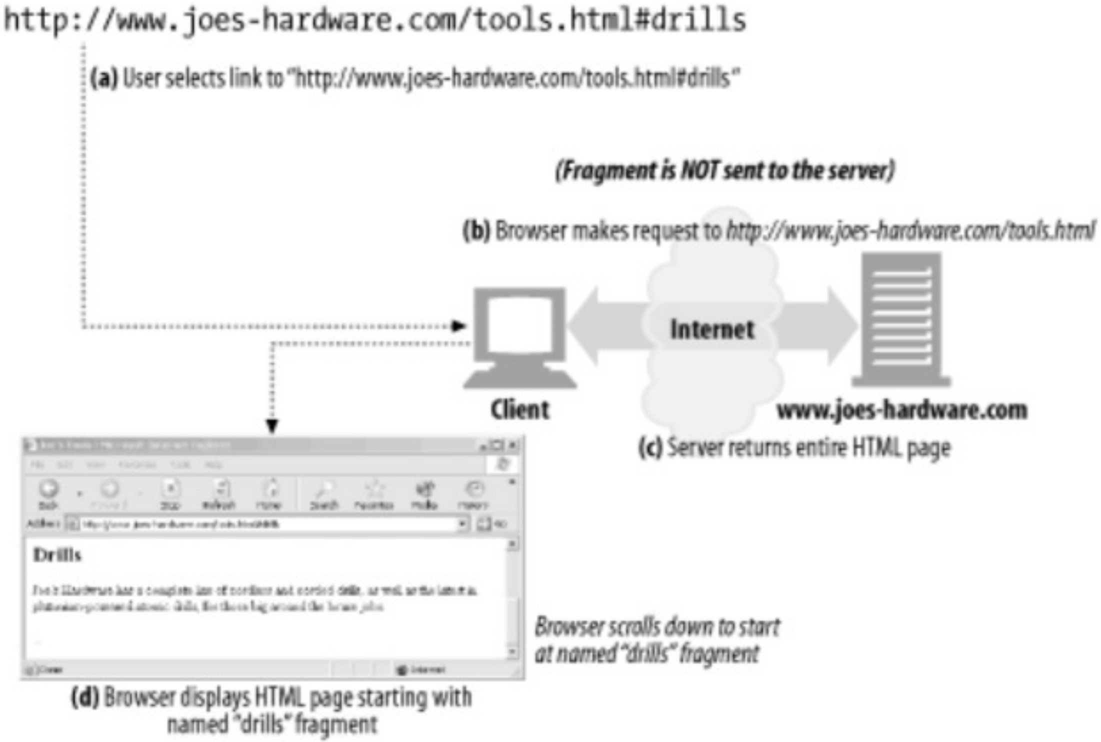
The fragment is used to identify a fragment inside the specified resource. For example, it can refer to a specific paragraph inside an HTML page

URL structure

```
<scheme>://<user>:<password>@<host>:<port>/<path>
                          ;<params>?<query>#<frag>
```

Since the server usually manages entire resources, not parts of them, the fragment section is not sent to the server during an HTTP request.

Resource fragments


http://www.joes-hardware.com/tools.html#drills
(a) User selects link to "http://www.joes-hardware.com/tools.html#drills"
(Fragment is NOT sent to the server)
(b) Browser makes request to http://www.joes-hardware.com/tools.html
Internet
Client
www.joes-hardware.com
(c) Server returns entire HTML page
Drills
Browser scrolls down to start at named "drills" fragment
(d) Browser displays HTML page starting with named "drills" fragment

Absolute and relative URLS

What we have seen so far are absolute URLs because they contain everything needed to access a specific resource.

Sometimes, it is simpler to use relative URLs (containing just the resource name) that can be automatically transformed into absolute URLs according to a specific **base** URL.

Relative URL

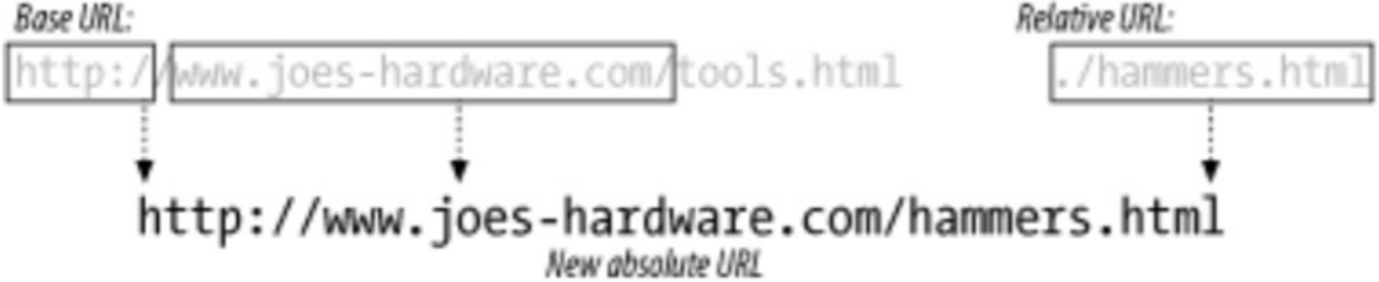
```
<html>
<head><title>Joe's Tools</title></head>
<body>
<h1> Tools Page  </h1>
<p> Joe's Hardware Online has the largest selection of <a
href="./hammers.html" />hammers
</body>
</html>
```

This URL is relative (there is no host and no schema specified)

Relative URL

If we assume the following base URL:

`http://www.joes-hardware.com/tools.html`


Base URL:
http://www.joes-hardware.com/tools.html
Relative URL:
./hammers.html
http://www.joes-hardware.com/hammers.html
New absolute URL

Base URL

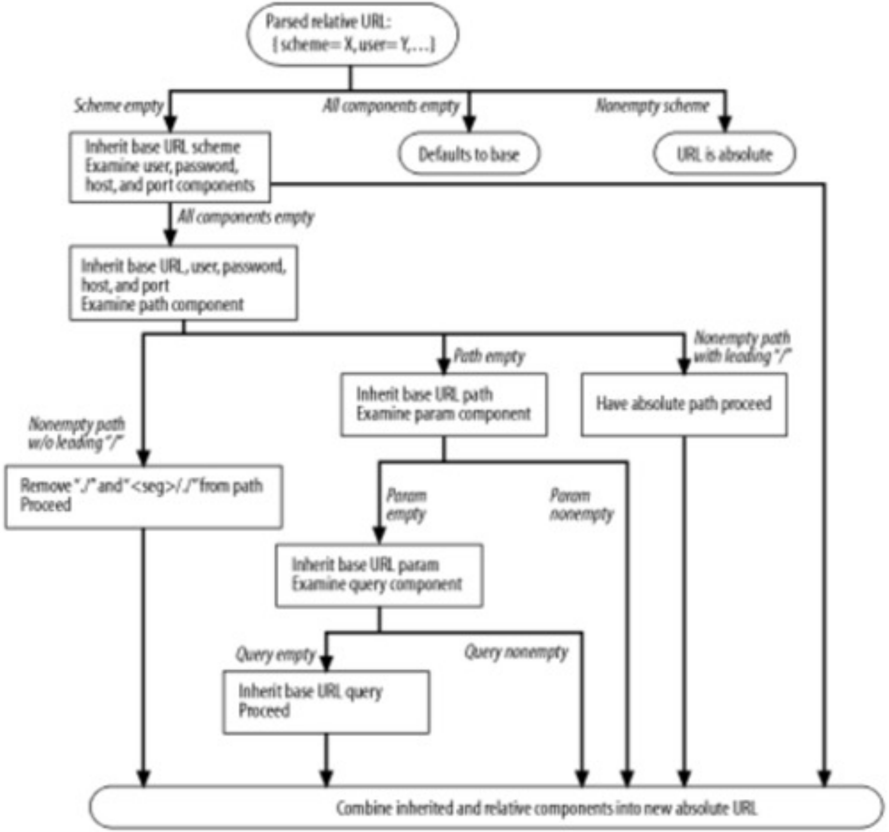
Base URL can be obtained in 2 ways:

1. By specifying it explicitly using the tag **`<base>`** in the document header

   Ex. **`<base href="http://www.mysite.com" />`**

2. Using the absolute URL of the resource (HTML page) in which the relative URL is contained (more common)

From relative to absolute



- Complete algorithm to obtain an absolute URL from a relative one
- Contained in RFC 2396

The URL character set

According to the HTTP protocol, URLs can only contain symbols from the standard **ASCII** character set.

If the resource name contains characters outside the ASCII set, it is necessary to escape the name with a set of valid characters.

Escaping

Character escaping in a URL is performed in the following way:

- % + <2 digits hex code>

The hex code to insert depend on the browser's character set. Default is UTF8

IRI

The **I**nternationalized **R**esource **I**dentifier (**IRI**) was defined in 2005 to extend the URL with all the characters contained in the universal character set (Unicode)

Backward-compatible via the standard URL escaping. Browsers designed to support IRI will visualize the correct characters

IRI

IRI example:

https://en.wiktionary.org/wiki/Ῥόδος

Corresponding escaped URL:

https://en.wiktionary.org/wiki/%E1%BF%AC%CF%8C%CE%B4%CE%BF%CF%82

IRI and phishing

One of the biggest problems of IRI is the Internationalized domain name (IDM) homograph attack.

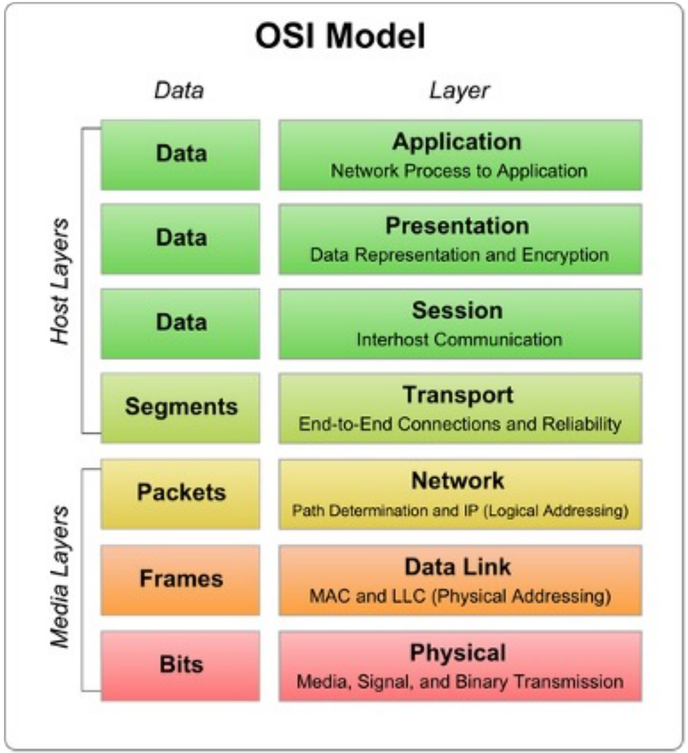
Idea:

Characters similar to common ASCII symbols are used to redirect a user to a malicious website.

Ex: paypɑl.it instead of paypal.it

HTTP connection

- HTTP is a protocol at the "application" level in the OSI model.
- Uses TCP/IP to exchange data between client and server
- Host address and port are defined inside the URL



OSI Model

	Data	Layer
Host Layers	Data	**Application** Network Process to Application
	Data	**Presentation** Data Representation and Encryption
	Data	**Session** Interhost Communication
	Segments	**Transport** End-to-End Connections and Reliability
Media Layers	Packets	**Network** Path Determination and IP (Logical Addressing)
	Frames	**Data Link** MAC and LLC (Physical Addressing)
	Bits	**Physical** Media, Signal, and Binary Transmission

TCP: Base concepts

TCP provides a bidirectional data stream

- Connection-oriented
- Reliable (data are either correctly received or an error is generated)
- Data arrive in the same order as they were generated
- Flow control is used to avoid congestion

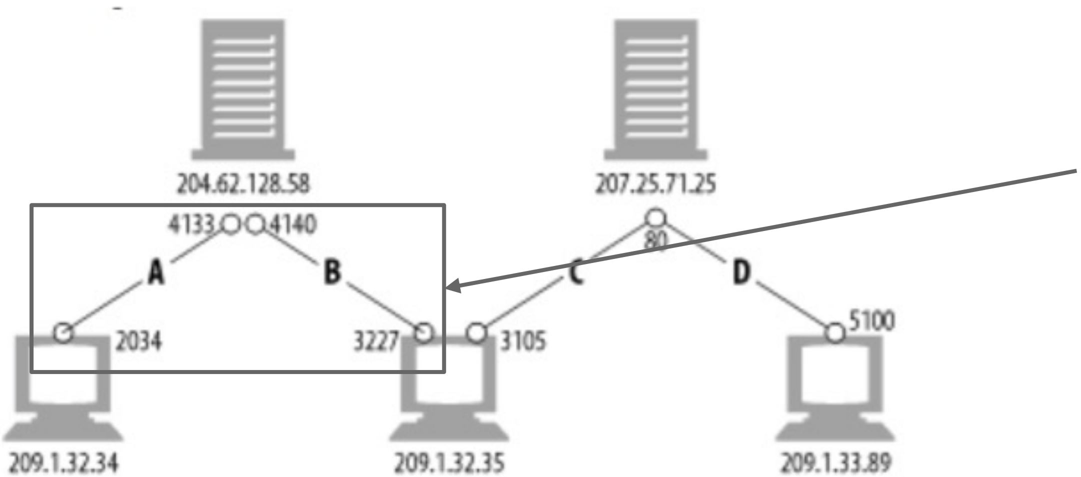
TCP: Base concepts

A TCP connection is uniquely defined by 4 values:

<source ip> <source port> <destination ip> <destination port >

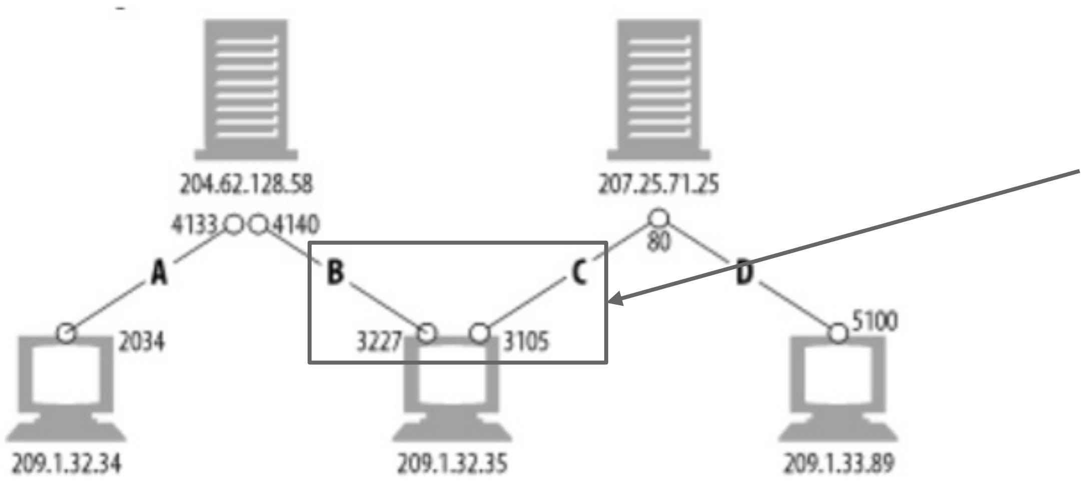
Multiple connections to/from the same IP or port are allowed. Every connection must have a different quadruple of values

TCP: Base concepts



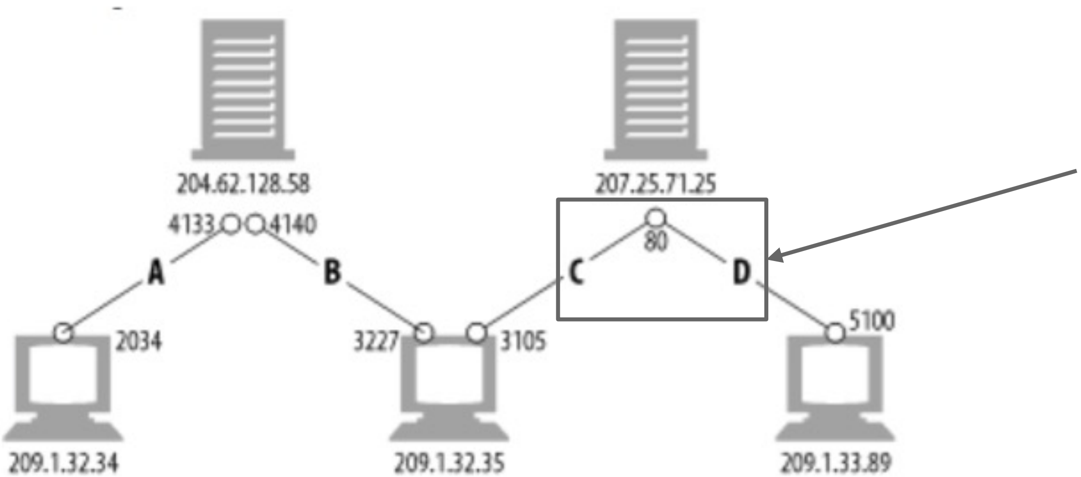
Different source addresses and ports

TCP: Base concepts



B and C share the same source address

TCP: Base concepts



C and D share the same destination address and port

HTTP messages

When the **client** establishes a TCP connection with the server, the HTTP protocol expects the exchange of at least 2 **messages:**

1. A **request** message (from the client to the server)
2. **A response** message (from the server back to the client)

Every message contains a text string describing its content and an optional payload

HTTP messages

A message is composed by 3 parts

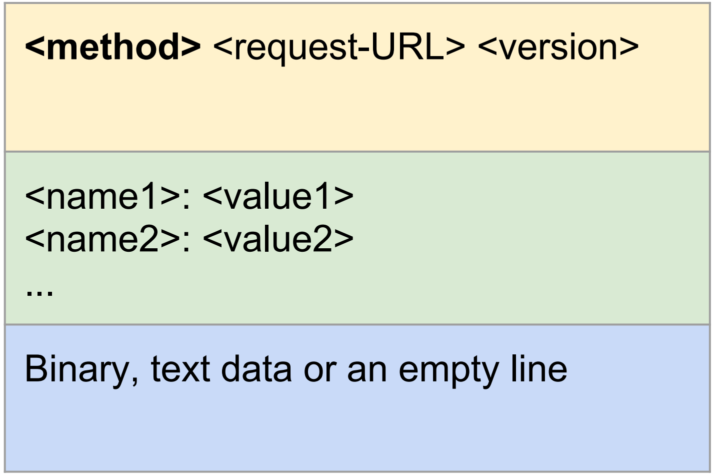
Start line	A string describing the message followed by a newline
Header	Multiple text lines to define options an properties of the message. The header section end with an empty newline.
Body	Generic data

Request

Every request message is composed as follows

Start line	<method> <request-URL> <version>
Header	<name1>: <value1> <name2>: <value2> ...
Body	Binary, text data or an empty line

Possible methods



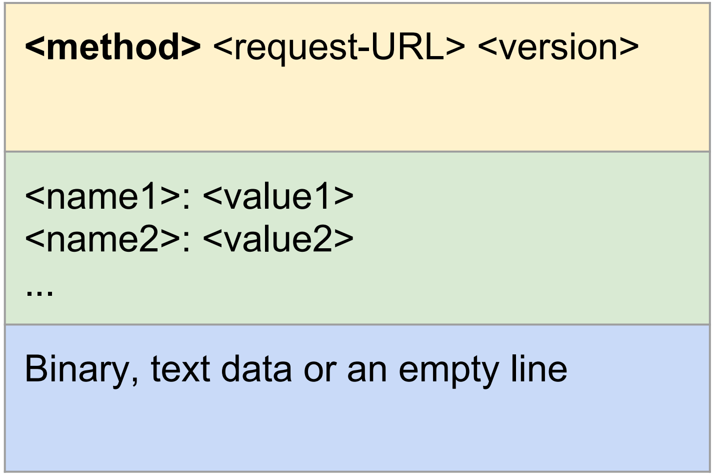
<method> <request-URL> <version>

<name1>: <value1>
<name2>: <value2>
...

Binary, text data or an empty line

GET	Ask the server to provide a certain resource
HEAD	Ask the server to provide only the resource header. Usages: ● Know the type of a resource ● Check if a resource exists ● Check if a resource has been modified
PUT	Ask the server to store a certain resource (the inverse of GET)

Possible methods



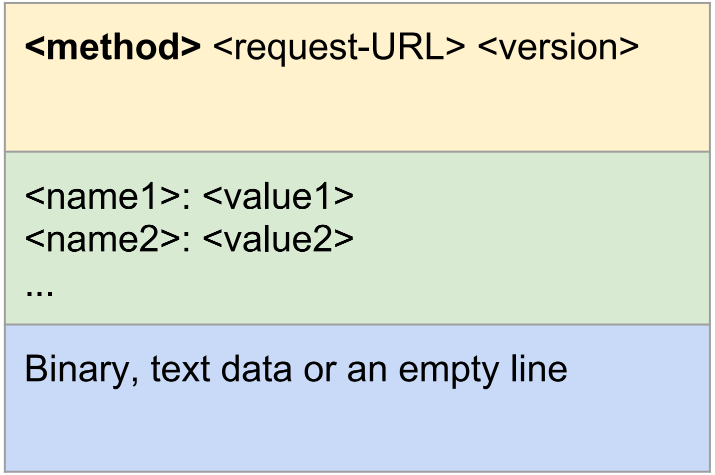
<method> <request-URL> <version>

<name1>: <value1>
<name2>: <value2>
...

Binary, text data or an empty line

POST	Used to send generic data to the server. Differently than PUT, data are not meant to be stored by the server but just used by different programs.
TRACE	Gives diagnostic data on the connection topology
OPTIONS	Ask the server for a list of supported functionalities.

Possible methods



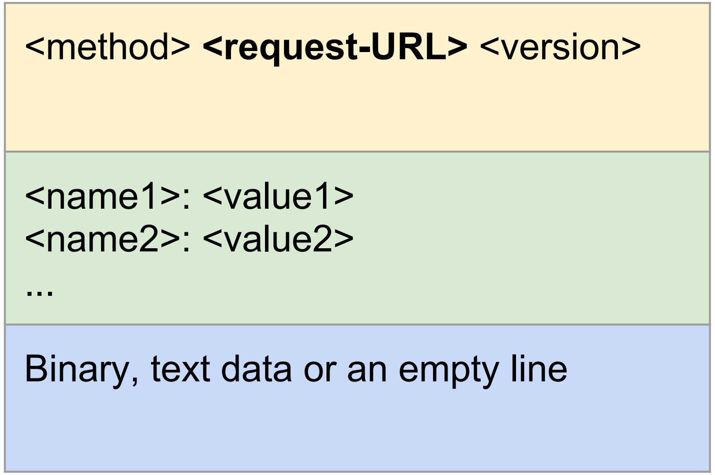
<method> <request-URL> <version>

<name1>: <value1>
<name2>: <value2>
...

Binary, text data or an empty line

DELETE	Used to ask the server to delete a resource. Usually requires authentication
LOCK, MKCOL, COPY, MOVE, etc	A server can support several other methods not included in HTTP/1.1 protocol (ex. WebDAV)

Request-URL



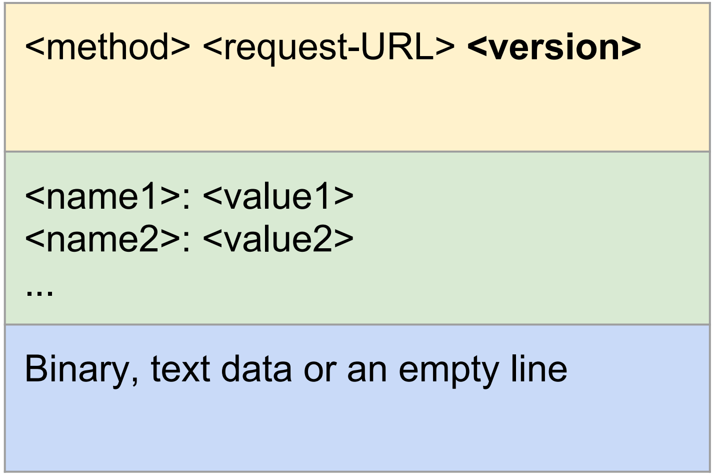
<method> **<request-URL>** <version>

<name1>: <value1>
<name2>: <value2>
...

Binary, text data or an empty line

Resource path

Version



<method> <request-URL> **<version>**

<name1>: <value1>
<name2>: <value2>
...

Binary, text data or an empty line

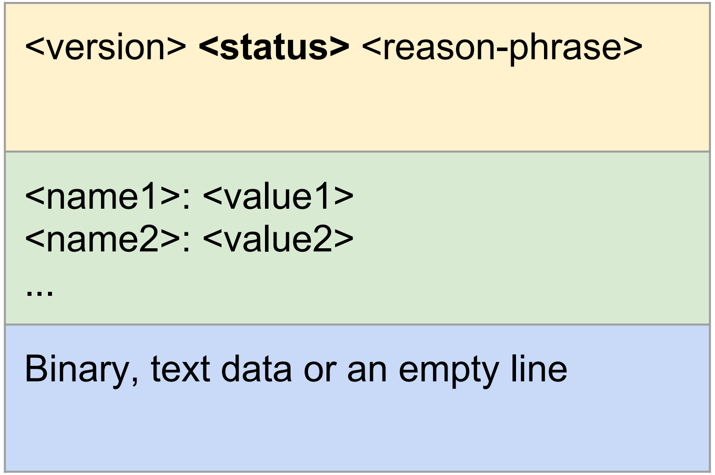
HTTP protocol version

Response

A response message is sent by the server to the client only **after a request message is received**.

Start line	<version> <status> <reason-phrase>
Header	<name1>: <value1> <name2>: <value2> ...
Body	Binary, text data or an empty line

Status



<version> **<status>** <reason-phrase>

<name1>: <value1>
<name2>: <value2>
...

Binary, text data or an empty line

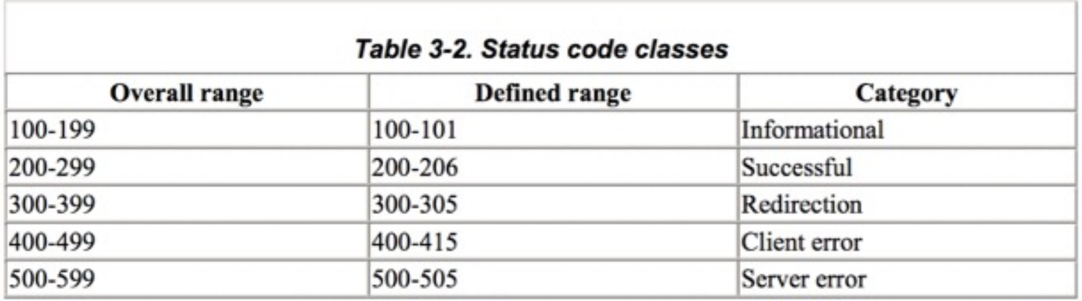
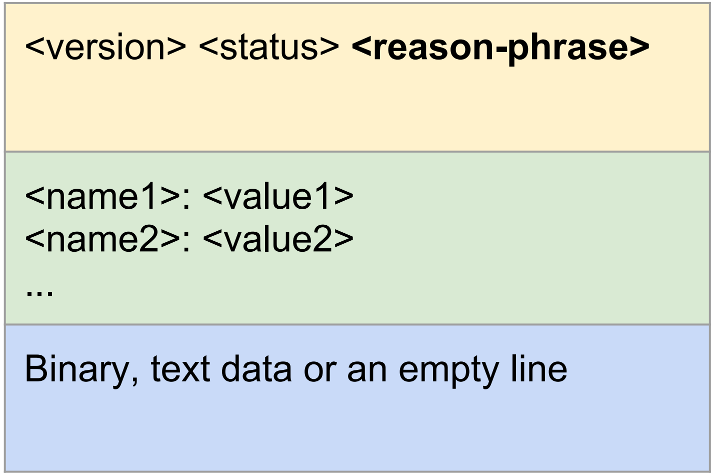


Table 3-2. Status code classes

Overall range	Defined range	Category
100-199	100-101	Informational
200-299	200-206	Successful
300-399	300-305	Redirection
400-499	400-415	Client error
500-599	500-505	Server error

Reason-phrase



<version> <status> **<reason-phrase>**

<name1>: <value1>
<name2>: <value2>
...

Binary, text data or an empty line

A text description of the status code

Headers

HTTP/1.1 define multiple headers according to the message type:

- General purpose
- Request headers
- Response headers
- Entity headers
- Protocol extensions

General purpose

Date	Message date and time
Upgrade	Specifies which protocol version the client wants to use
Cache-Control	Set caching directives
Trailer	Used for chunked transfers

Headers request

Client-IP	Client IP address
Host	Host IP address
Accept	Used to notify the server on the supported MIME types
Accept-Encoding	Used to specify the accepted encodings
If-Modified-Since	Ask the server to send the resource only if it has been modified after a certain date

Headers request

Authorization	Contains authentication data to access restricted resources
Cookie	Used by the client to send generic “token strings” called cookies

Headers response

WWW-Authenticate	Used to ask the client to provide authentication data
Set-Cookie	Used by the server to ask the client agent to store a generic “token string.”
Server	Web-server software name and version

Entity Headers

Allow	List of allowed methods on a certain resource
Content-Encoding	Resource encoding
Content-Length	Resource size
Content-Type	Resource MIME-type
Last-Modified	Last modified date of a resource

HTTP Security

HTTP protocol does not intrinsically provide functions to guarantee the **security** and **secrecy** of the exchanged data

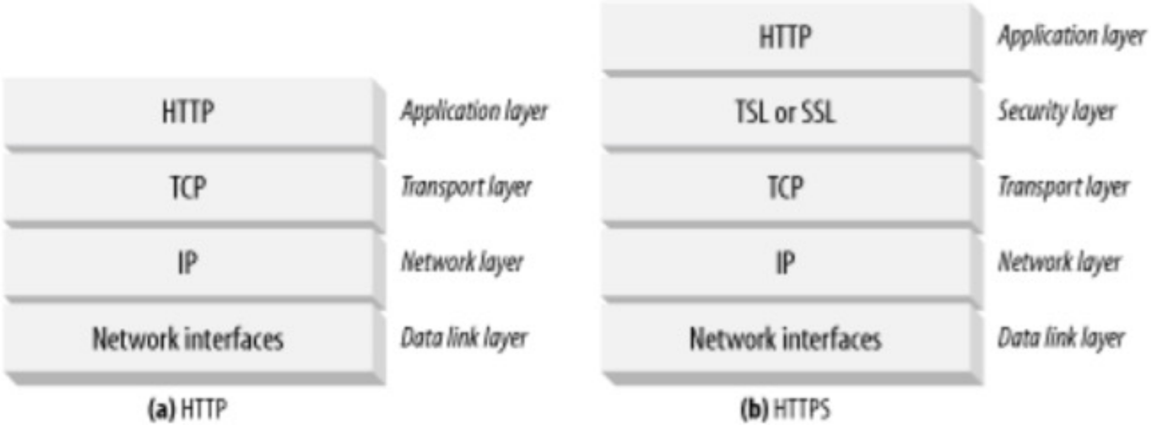
With the evolution of the modern web, we faced the need to use a secure protocol to manage bank transactions, authentication, etc.

HTTP Security

Anyone connected on the same network can potentially eavesdrop on the traffic generated by two entities to steal confidential data or maliciously alter the content (man-in-the-middle attack)

HTTP security is implemented by **inserting a new layer** below it. 2 options: SSL (Secure Socket Layer) or TLS (Transport Layer Security)

HTTPS



The advantage of using an additional layer is that the original HTTP protocol remains unchanged. The underlying connection is just established in a different way

HTTPS features

Server authentication

- A client can verify the identity of the server to ensure that is not connected to somebody else faking its identity

Client authentication

- Also, the server can verify the client regardless of the built-in HTTP authentication method.

HTTPS features

Transmission integrity

- It is possible to detect if the transmitted data has been tampered with by a third agent.

Data encryption

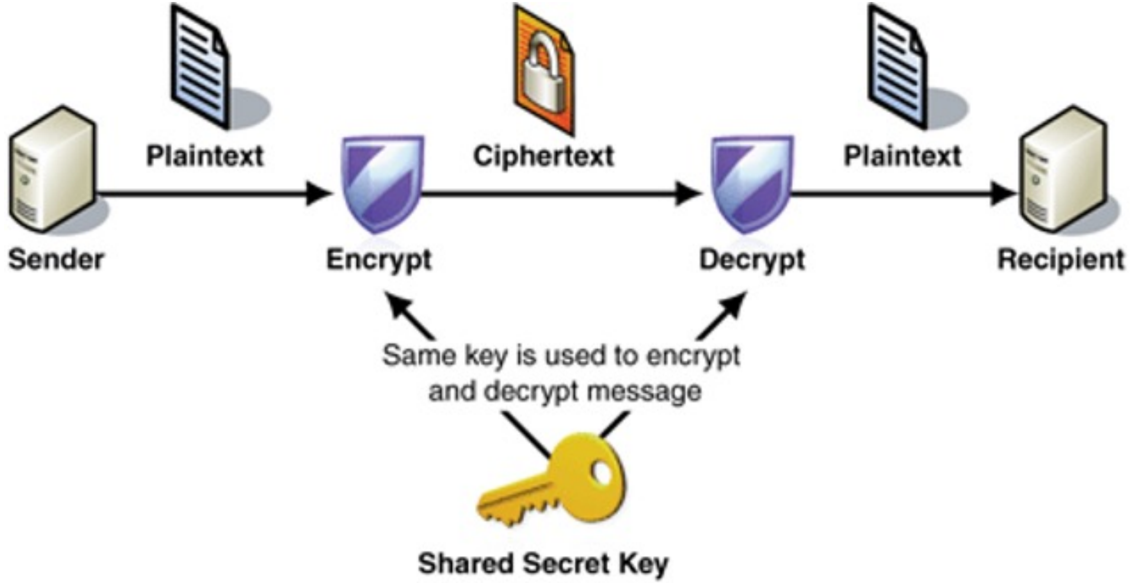
- The transmitted data are readable only by the client and server pair. Nobody can eavesdrop on the connection to steal sensible data.

HTTPS and cryptography

HTTPS is based on the following techniques:

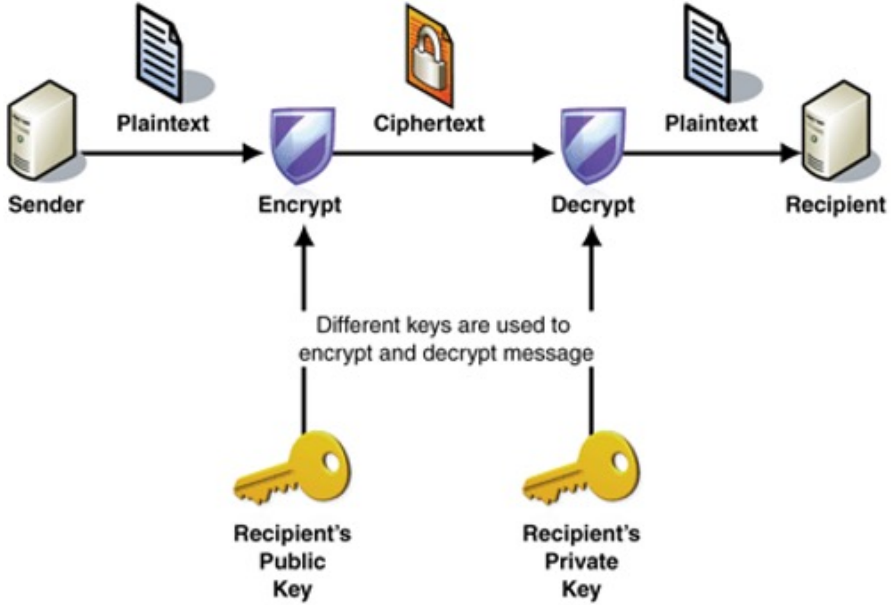
- Symmetric key cryptography
- Asymmetric key cryptography
- Digital signature
- Digital certificate

Symmetric key cryptography



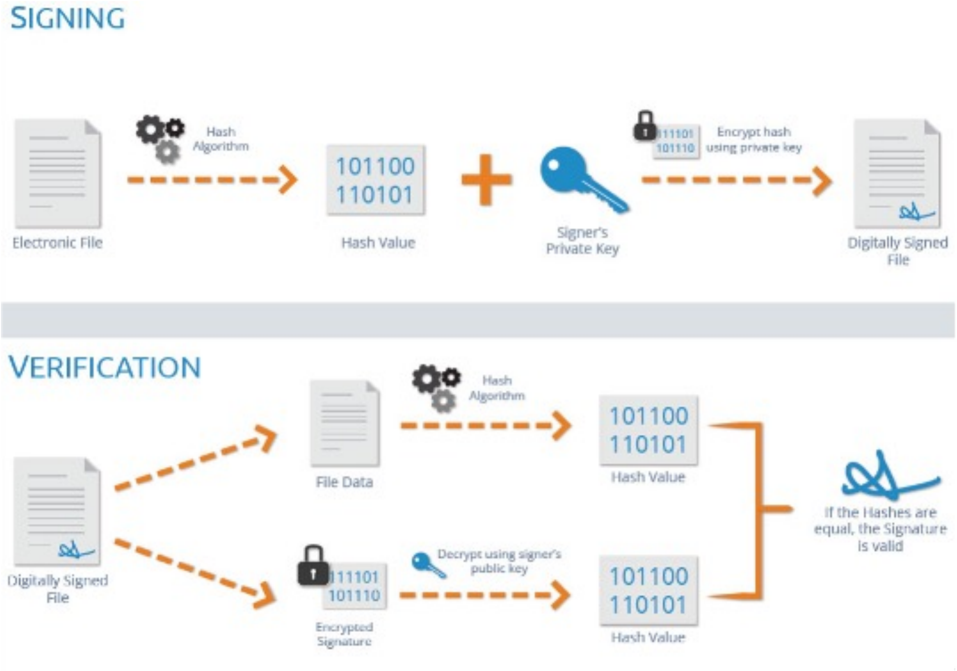
Problem: how to exchange keys?

Asymmetric key cryptography



Public key to crypt, private key to decrypt. Public key pairs can be freely exchanged.

Digital signature



Private key used to sign a document. Public key used to verify the signature

Digital certificate

Problem: Who guarantees that a specific public key really belongs to a certain owner?

"A digital certificate is an electronic document used to prove the ownership of a public key."

https://en.wikipedia.org/wiki/Public_key_certificate

Digital certificate

Usually contains:

- The subject name (person, server, organization, etc.)
- The expiration date
- Information on the issuer of the certificate
- The public key of the subject

Everything is signed by a trusted **signing authority** that has verified the correspondence between the subject and its public key

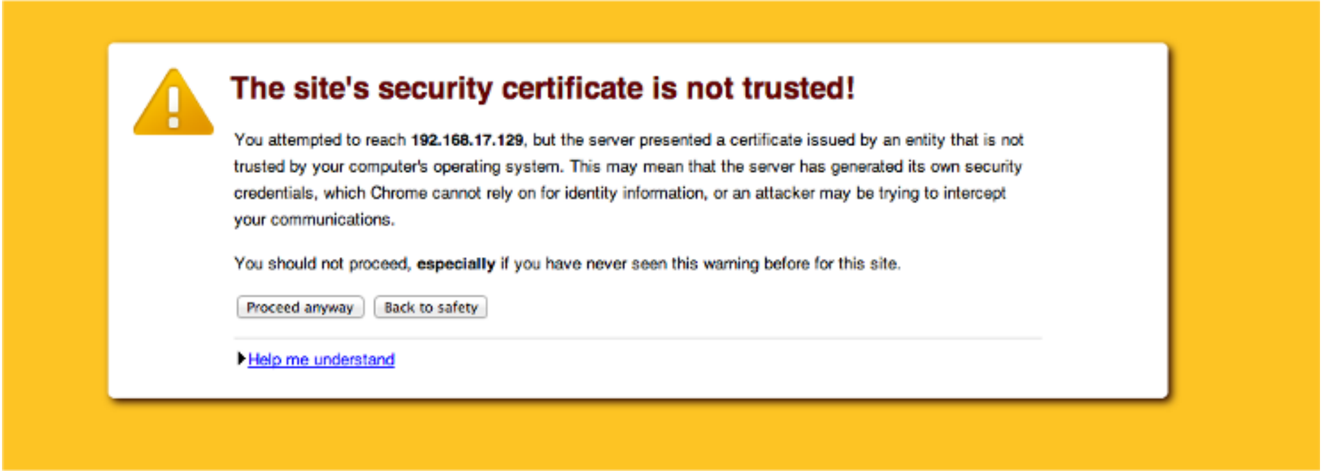
HTTPS

When the browser connects using HTTPS, the following operations are performed:

- The client requests the server's digital certificate
- The signing authority is read from the certificate
- If the signing authority is known and preinstalled in the browser, the digital signature is verified by using the public key of the signing authority.

HTTPS

If the signing authority is unknown (or the certificate is self-signed), the browser will warn the user that the server identity cannot be verified.



The site's security certificate is not trusted!

You attempted to reach **192.168.17.129**, but the server presented a certificate issued by an entity that is not trusted by your computer's operating system. This may mean that the server has generated its own security credentials, which Chrome cannot rely on for identity information, or an attacker may be trying to intercept your communications.

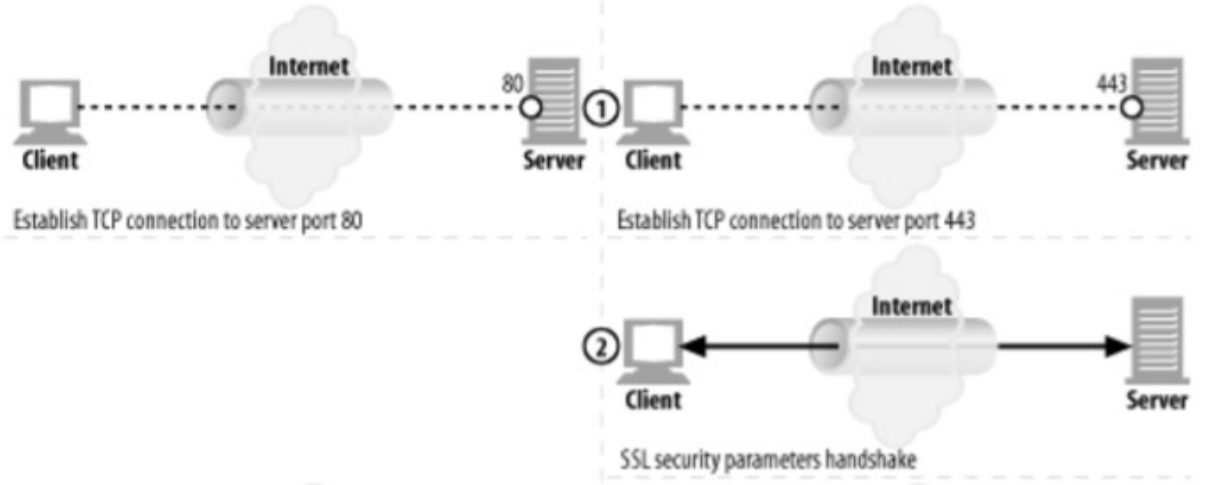
You should not proceed, **especially** if you have never seen this warning before for this site.

HTTPS

After the certificate check, the SSL/TLS layer negotiates the security parameters for use, and a secure encrypted channel is established.

After that, HTTPS works the same way as HTTP but operates on the secure channel

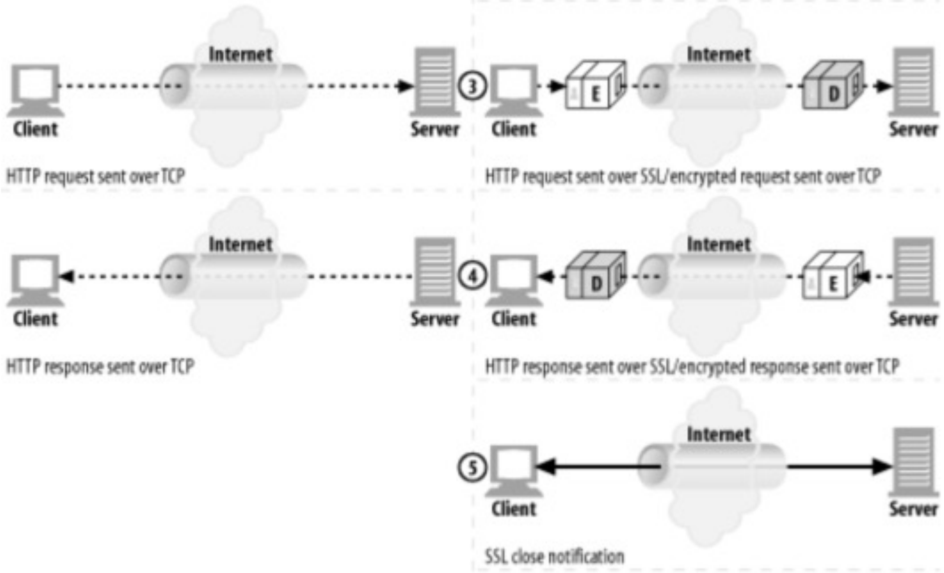
HTTP vs HTTPS


Internet
80
Client
Server
Establish TCP connection to server port 80
1
Client
Internet
443
Server
Establish TCP connection to server port 443
2
Client
Internet
Server
SSL security parameters handshake

- Client establishes a TCP connection on port 80

- Client establishes a TCP connection on port 443
- Security parameters handshake

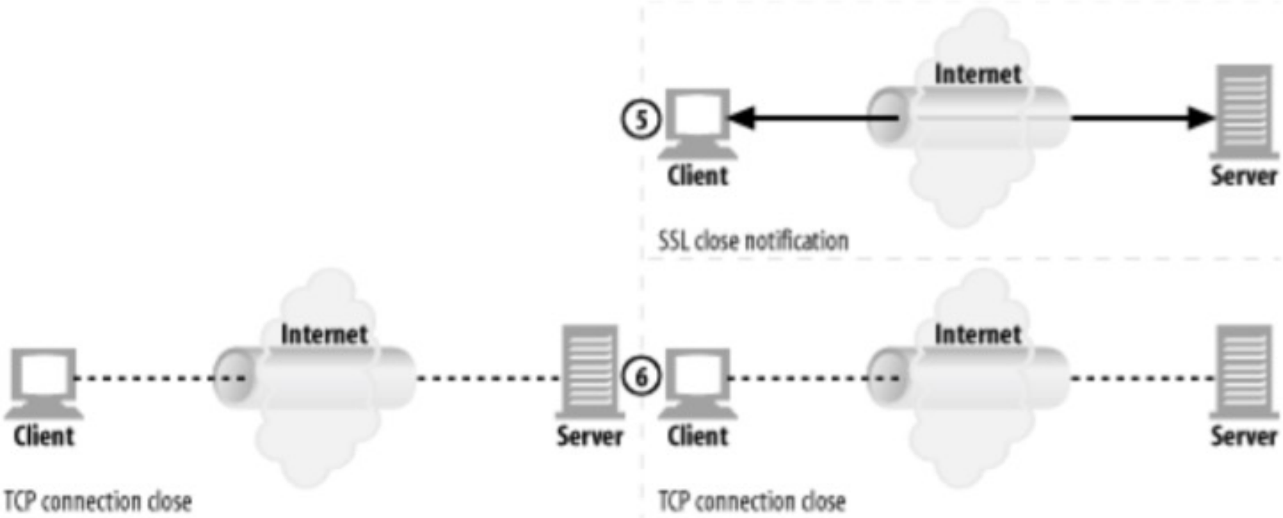
HTTP vs HTTPS


Internet
Client
Server
HTTP request sent over TCP
3
Client
E
D
Internet
Server
HTTP request sent over SSL/encrypted request sent over TCP
Internet
Client
Server
HTTP response sent over TCP
4
Client
D
E
Internet
Server
HTTP response sent over SSL/encrypted response sent over TCP
5
Client
Internet
Server
SSL close notification

- Request and response are exchanged on the TCP channel

- Request and response are exchanged securely via SSL/TLS

HTTP vs HTTPS



- TCP connection is closes

- SSL/TLS connection is closed
- TCP connection is closed