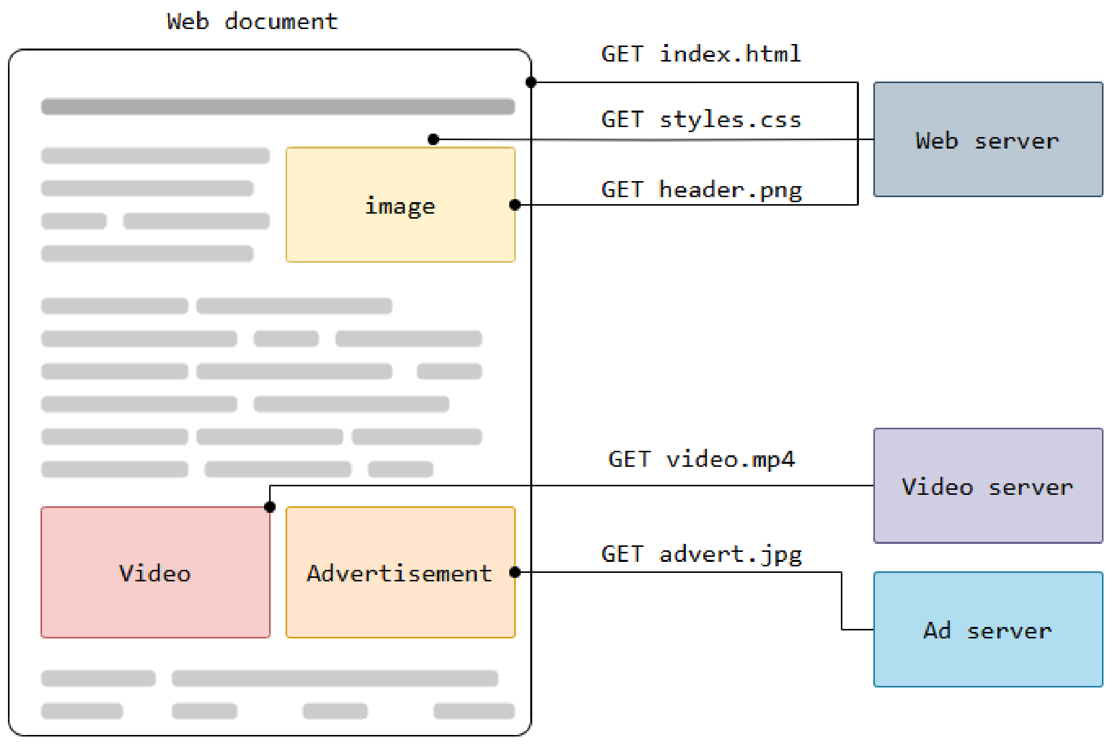
Overview of HTTP

HTTP is the foundation of data exchange on the Web. It is a **client-server protocol**, where the browser (client) sends requests to fetch resources like text, images, videos, scripts, and styles that together form a complete webpage.

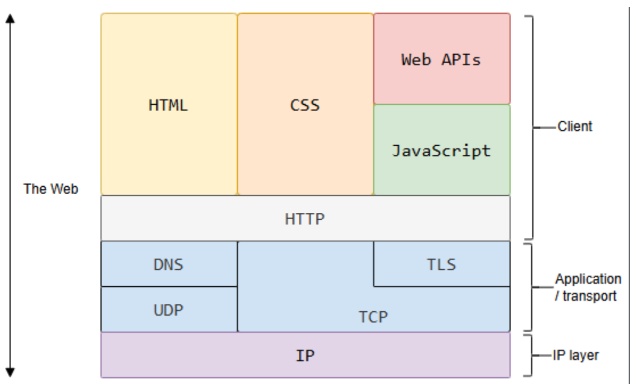


Clients and servers communicate

through **messages**:

the client sends **requests**, and the

server replies with **responses**.

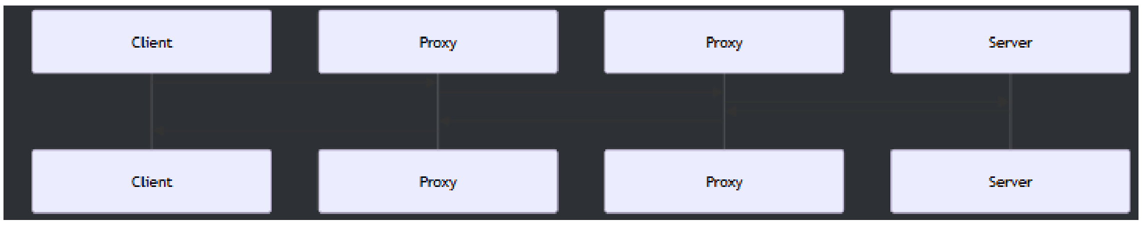


HTTP, created in the early 1990s, is an **extensible application layer protocol** that runs over TCP or encrypted TLS connections. It is used not only to fetch web documents, images, and videos but also to post data and update parts of webpages on demand.

Components of HTTP-based systems

HTTP is a client server protocol where the client (usually a web browser or a bot) sends requests, and the server sends responses.

Between the client and server, there are other entities called proxies. These proxies can act as gateways, caches, or perform other operations to help menage or optimize communication.



Between the browser and the server, there are many devices like routers and modems, but they are hidden by the layered structure of the web. These devices operate at the network and transport layers, while HTTP works at the application layer. Although these lower layers are useful for diagnosing network issues, they are mostly irrelevant to understanding http.

Client: the user-agent

The user-agents is any tool that acts on behalf of the user. It is usually a web browser but can also be developer tools or automated programs like web crawlers.

The browser is always the one that initiates HTTP requests. The requests. The server starts communication, although some technologies simulate server-initiated messages.

To display a web page, the browser first requests the HTML document, then fetches additional resources like CSS, scripts, images, and videos. It parses and combines these elements to render full page. Scripts can later request more data, updating the page dynamically.

A web page is Hypertext document, meaning it contains clickable links. When a user clicks a link, the browser sends a new HTTP request and processes the response to display the next page, enabling navigation across the web

The Web server

The Web Server responds to client requests by serving documents. Although it may appear as a single machine, it can actually be a group of servers working together (load balancing) or include other systems like caches, databases, or e-commerce platforms that generate content on demand.

Multiple server instances can run on the same physical machine, and witch HTTP/1.1 and the host header, they can even share the same IP address.

Proxies in Web Communication

A proxy is na intermediar server between a web browser (client) and a web server. While many devices in the network stack operate at lower levels (transport, network, physical), proxies function at the application layer, directly handling HTTP messages.

Types of Proxies

Transparent Proxies: Forward requests without modifying them.

Non-Transparent Proxies: Alter requests before forwarding (e.g., filtering or caching).

Functions of Proxies

Caching

Stores copies of frequently accessed content.

Can be public (Share across users) or private (like browser cache)
Filtering

Blocks or modifies content based on rules.

Examples: antivirus scanning, parental control.

Load Balancing

Distributes incoming traffic across multiple servers.

Improves performance and reliability.

Authentication

Controls access to resources.

Ensures Only authorized users can proceed.

Logging

Records request and response data.

Useful for analytic, debugging, and security audits.

Basic aspects of HTTP

Basic aspects of HTTP

HTTP is designed to be simple and readable by humans. Even which the added complexity of HTTP/2, witch uses frames to encapsulate messages, developers and newcomers can easily understand and test HTTP communications.

HTTP is extensible

Since HTTP/1.0, headers have made the protocol flexible and easy to expand. New features can be added through mutual agreements between clients and servers about the meaning of custom headers.

HTTP is stateless, but not sessionless

HTTP does not maintain a connection between requests, which can be problematic for interactive applications like shopping carts. However, cookies allow session management by preserving state across multiple requests using header extensions.

HTTP and connections

HTTP relies on reliable transport protocols like TCP, though it doesn't menage connections itself. HTTP/1.0 open a new TPC connection for each request, which is inefficient. HTTP/1.1 introduced persistent connections and pipelining, while HTTP/2 improved efficiency with multiplexing. New protocols like QUIC, built on UDP, are being developed to further enhance performance and reliability.

What can be controlled by HTTP

The extensibility of HTTP has enabled greater control over web functionality over time. Early features like caching and authentication were quickly adopted, while more advanced capabilities – such relaxing origin constraints- came later. HTTP allows servers and clients to manage various aspects of communication and behavior through headers and cooks.

Caching

HTTP lets servers control how documents are cached by clients and proxies, specifying what to store and for how long. Clients can also instruct proxies to ignore cached versions, ensuring fresh content.

Relaxing the origin constraint

Web browser enforce strict origin polices to protect user privacy, allowing only same-origin pages to access full content. HTTP headers can relax this restriction, enabling cross-origin resource sharing when needed even for security-related purposes.

Authentication

HTTP supports basic authentication using headers like www-autheticate, and more advanced session-based authentication through cookies. This ensures that only authorized users can access protected resources

Proxy and Tunneling

HTTP requests often pass through proxies, especially when clients or servers are on private networks. These proxies help route traffic and can support other protocols like FTP or SOCKS, wich operate at different layers.

Sessions

Although HTTP is stateless, cookies allow session management by linking requests to server-side state. This enables features like shopping carts and personalized user settings across multiple interactions.

HTTP flow

When a client wants to communicate witch a server, either the final server or a intermediate proxy, it performs the following steps

1 Open a TCP connection: The TCP connection is used to send a request, or several, and receive an answer. The client may open a new connection, reuse an existing connection, or open several TPC connections to the servers

2 Send an HTTP message: HTTP messages (before HTTP/2) are human-readable. With HTTP/2, these messages are encapsulated in frames, making them impossible to read directly, but the principles remain the same. For example:

HTTP Copy

```http
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

3 Read the response sent by the server, such as:

HTTP Copy

```http
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!doctype html>… (here come the 29769 bytes of the requested web page)
```

4 Close or reuse the connection for further requests.

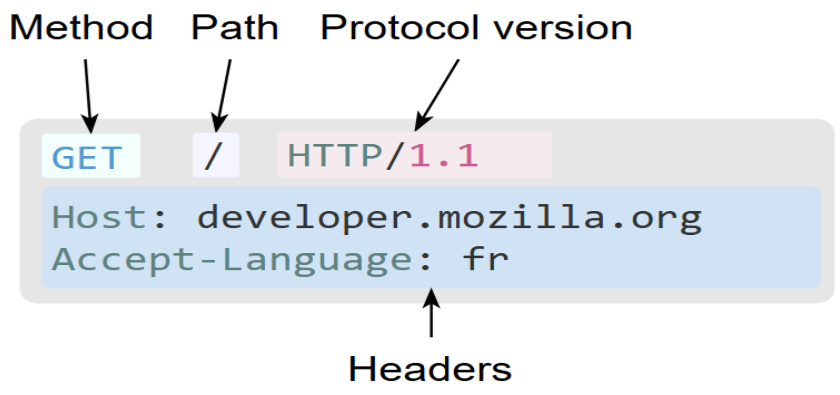
If HTTP pipelining is activated, several requests can be sent without waiting for the first response to be fully received. HTTP pipelining has proven difficult to implement in existing networks, where old pieces of software coexist witch modern version. HTTP pipelining has been superseded in HTTP/2 witch more robust multiplexing requests within a frame.

HTTP Messages

HTTP messages are either requests or responses. In HTTP/1.1 and earlier, they are human-readable, while in HTTP/2, they are embedded in binary frames for performance improvements like header compression and multiplexing. Despite the format change, the meaning of the messages remains the same, and clients reconstruct them as if they were HTTP/1.1 messages,

Requests

An example HTTP request:



```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

An HTTP Method, usually a verb like GET, POST, or a noun like OPTIONS or HEAD the defines the operation the client wants to perform. Typically, a client wants to fetch a resource (using GET) or post the value of an HTML form (using POST), though more operations may be needed in other cases.

The path of the resource to fetch: the URL of resource stripped from elements that are obvious from the context, for example without the protocol (http://), the domain (here developer.mozilla.org) or the TCP port (here, 80).

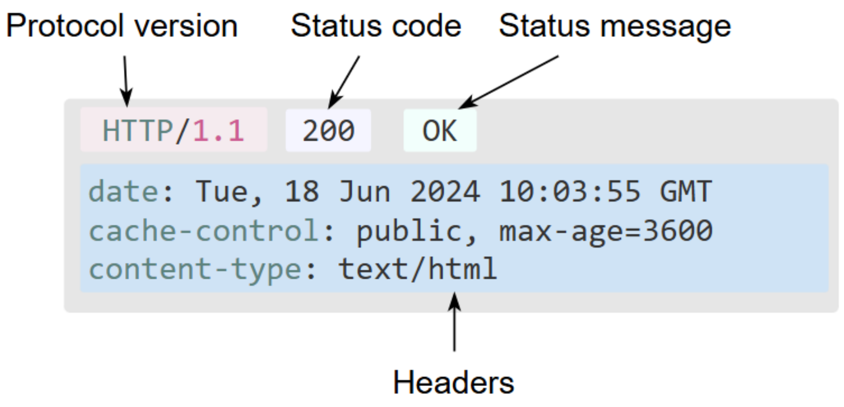
The version of the HTTP protocol

Optional headers that convey additional information for the servers.

A body, for some methods like POST, similar to those in response, which contain the resource sent.

Responses

An example response:



```
HTTP/1.1 200 OK
date: Tue, 18 Jun 2024 10:03:55 GMT
cache-control: public, max-age=3600
content-type: text/html
```

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.

APIs based on HTTP

The **Fetch API** is the most widely used HTTP-based API in JavaScript, replacing the older XMLHttpRequest. It allows developers to make HTTP requests easily and efficiently.

Another HTTP-based API is **Server-Sent Events (SSE)**, which enables one-way communication from the server to the client. Using the EventSource interface, the client opens a connection and listens for events. Incoming messages are automatically converted into Event objects and delivered to the appropriate event handlers in the browser.