Have you ever wondered what happens when you enter google.com on the address bar and press enter? They say computers understand machine language, 1’s and 0’s, and yet you may wonder, how does my computer understand ingles?

In this article, I will explain to you step by step what happens behind the scenes of google.com. I would however like to mention that the explanation may contain some ambiguous words. Not to worry, I have a well written thesaurus immediately after my explanation.

So, what happens behind the scenes of google.com?

On opening your browser, google for example, you are met with the words “Search Google or enter URL”. Let’s say you type "google.com" in your web browser's address bar and hit "Enter". All you see is a ‘line’ moving to the right and boom, you get most of what you wanted.

On hitting enter, a HTTP request is sent to the Google **server**. However, there is a small problem. Google servers don’t recognize any URL and here is why:

Computers communicate or access information from one another using **IP addresses.** The **domain** names like google.com are designed to be easily remembered by people, as we tend to find names more memorable than numerical addresses. However, computer systems use numerical addresses called IP addresses to identify websites on the internet.

When the request fails (you won't be notified), your operating system (Linux or Windows) tries to find google.com. First, it checks for the IP address in its cache then the file /etc/hosts. The cache is a high-speed data storage layer which stores a subset of data so that future requests for that data can be served up faster. /etc/hosts is a file that contains some IPv4 and IPv6 addresses including your loopback address(localhost).

If the OS fails to find the IP address, it calls the resolver. The resolver is your internet service provider.

My name is Google.com. I'm assuming all of you know me well
Windows OS
Root server
Resolver
You look familiar. Let me check my cache

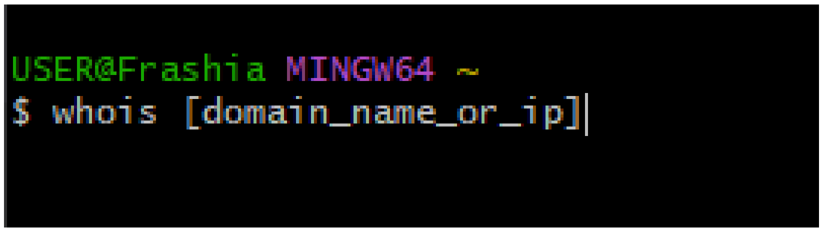
Step 1: The resolver checks its cache for this information. If this fails, it proceeds to the next step

All resolvers know where to locate the **root server.** This is where we involve the **DNS.** DNS It is a technology that translates human-adapted text-based domain names to machine-adapted, numerical-based IPs. The root server sits on top of the DNS hierarchy and is syntaxed: (letter). root.server.net. for example, a.root.server.net.

Step 2: The resolver requests the root server to find google.com. However, it as well doesn't know the IP address but knows where to locate the **TLD server. TLD** servers handle requests for the highest level of domains which include .com, .org, .net

Step 3: The resolver stores the information it's fed by the root server. Resolver now goes to the .com TLD server. Unfortunately, it also, can't provide the IP address but knows authoritative name servers for google.com. The .com TLD server points Resolver to the authoritative name servers. Why?

When a domain is purchased, the domain registrar reserves the name. After this, domain registrars communicate to the TLD registry concerning the authoritative name servers. Authoritative name servers are like ns1.google.com. These servers are normally 4 in number so as to split up tasks and incase one server fails, there is redundancy. To know who are the authoritative name servers for your domain, run a **WHOIS** query.



```
USER@Frashia MINGW64 ~
$ whois [domain_name_or_ip]
```

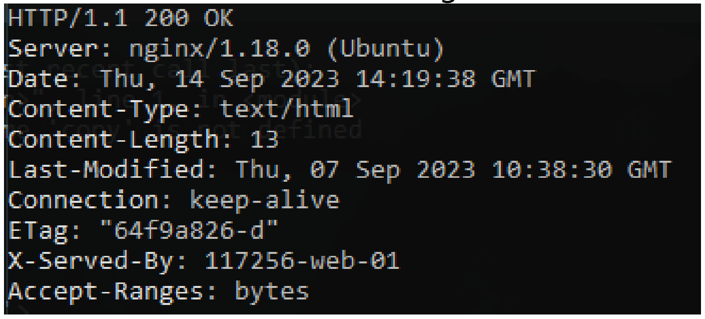
Step 4: Finally, the authoritative name server provides the IP address and the resolver saves this information. The resolver gives the OS the IP address so that it can save it and avoid future querying (repeating this lengthy process again). The IP address is stored at the local cache and not the file /etc/hosts. The local cache is managed by DNS resolver itself.

What's astonishing is that the whole above process happens within a second. Isn't that amazing!

Step 5: Well, after obtaining the IP address, our device (the client) will establish a TCP connection with the server. Then the browser will compose an HTTP request and that request will be sent to the server. This request will contain a request line, request headers and an optional request message body.

HTTP requests are not only used to fetch web pages. The request line contains a request method and by using different methods (GET, HEAD, PUT, POST, DELETE, ...) we can do a variety of things like post data on the server, ask the server to store data, ask the server to delete data and so on...To fetch a web page usually a GET or HEAD method is used.

The server will process this request and send an HTTP response. This response will also include a response (status) line, headers and an optional body. The response line contains a status code reflecting the outcome of the request. See the sample response



```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Thu, 14 Sep 2023 14:19:38 GMT
Content-Type: text/html
Content-Length: 13
Last-Modified: Thu, 07 Sep 2023 10:38:30 GMT
Connection: keep-alive
ETag: "64f9a826-d"
X-Served-By: 117256-web-01
Accept-Ranges: bytes
```

HTTP vs HTTPS

Indeed, this is a vast topic on its own. Nevertheless, I would just like to mention that HTTPS (hypertext transfer protocol secure) is just a secure version of HTTP.

DEFINITIONS

1. URL (A Uniform Resource Locator)- It is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. Examples: https://www.google.com/

2. Firewall - Firewalls are hardware or software security implementations that filter out incoming and outgoing network traffic and are usually put between a private network and the Internet.

3. TCP/IP (The Internet protocol suite)-A framework for organizing the set of communication protocols used in the Internet and similar computer networks according to functional criteria

4. HTTPS - The “https” portion refers to the protocol and must be followed by a semi-colon and two forward slashes. The Hypertext Transfer Protocol (HTTP) is one of the major protocols in the TCP/IP suite. It is a stateless client-server protocol used to

fetch web pages from the internet. HTTPS is a secure version of HTTP which uses the Secure Socket Layer (SSL) protocol for encrypting communication between the client and the server and securing user authentication.

5. Resolver - This is your ISP (internet service provider)

6. DNS- It is a technology that translates human-adapted text-based domain names to machine-adapted, numerical-based IPs.

7. IP address – A unique address that identifies a device on the internet or local network.

8. The "google.com" part is known as the domain name. A domain name is the website address. This is what users type in a browser search bar to directly access your website.

9. The ".com" is called the top-level domain. TLD is everything that follows the final dot of a domain name. For example, in the domain name 'google.com', '.com' is the TLD. The root server normally knows where to locate the .com TLD server. Other types of top-level domains are .ORG and .NET

10. The "Google" part is the second-level domain. A Second Level Domain (SLD) is part of the domain name that is located right before a Top-Level Domain (TLD)

11. The "www" part is known as the subdomain. Subdomains are part of a domain that comes before the main domain name and domain extension

12. The number at the end which is separated from the domain name by a colon is the port number. Ports are communication end-points identifying a specific process or type of network service enabling the use of multiple services from a single IP address. 443 is the port number associated with the HTTPS protocol.

<u>Common status codes to know are:</u>

1. 200 OK: request is fulfilled.

2. 301 Move Permanently: the requested resources have been moved permanently to a new location.

3. 302 Found and Redirect: the requested resources have been moved to a new location temporarily.

4. 404 Not Found: The requested resource cannot be found on the server.

5. 500 Internal Server Error: The server is confused, often caused by an error in the server-side program responding to the request.