The maximum length of domain name is 253

Port Number	Service/Room	Purpose
80	Web Server (HTTP)	Browsing websites
443	Secure Web (HTTPS)	Browsing secure sites
25	Mail Room (SMTP)	Sending emails
53	Reception Room (DNS)	Finding website addresses
22	Remote Access Room (SSH)	Logging into your system remotely

So if someone (another computer) wants to **send you an email**, they need to knock on **port 25**. If they want to **browse your website**, they knock on **port 80** or **443**.

A.) 🔍 Step 1: What Happens When You Type the Domain?

When you type **www.example.com** into your browser and hit **Enter**, **your computer needs to figure out where** that domain is located on the internet. This is how it works:

1. **Your computer doesn't understand "www.example.com" directly**.
   It only understands **IP addresses** (like **192.0.2.1**).
   So, the first thing your computer needs to do is **convert** the domain name into an **IP address**. This process is called **DNS resolution**.

🌐 Step 2: The Domain Name System (DNS)

Your computer uses the **Domain Name System (DNS)** to find the **IP address** associated with the domain name.

Here's how it works:

1. **DNS Request**: When you type **www.example.com**, your computer sends a request to a **DNS server** (like Google's DNS server at **8.8.8.8**).
2. **DNS Response**: The DNS server looks up **www.example.com** and sends back the **IP address** of the website's server (let's say it's **93.184.216.34**).

So now, instead of talking to **www.example.com**, your computer is now going to talk to the **IP address 93.184.216.34**.

The second level domain
is limited to 63 characters + TLD
can only use a-z 0-9 & hyphens
(cannot start or end with hyphens or have consecutive hyphens

"Port numbers are like room numbers in your house. They tell which specific room (service) should handle the request"

🔑 Step 3: Adding the Port Number

Now, your computer knows the **IP address** of the website's server, but there's still one more step: **finding the correct port**.

By default:

- **HTTP websites** use **Port 80**.
- **HTTPS (secure) websites** use **Port 443**.

How Does the Browser Know Which Port to Use?

- When you type **www.example.com**, your browser automatically **assumes** you want to visit the website using **Port 443** (because you're using **HTTPS**).
- This is **default behavior** — you don't need to manually add the port number to the URL!

So, when you visit **https://www.example.com**, **your browser** is really doing this:

1. Resolving **www.example.com** to the **IP address (e.g., 93.184.216.34)**.
2. Sending the request to **Port 443** of the server at **93.184.216.34**.

▌ Step 4: Connecting to the Server Through Ports

Now that your computer has the **IP address** and the **port** (Port 443 for HTTPS), it **connects** to the website's server:

1. Your computer sends a request through **Port 443** on the server (this is a **secure connection**).
2. The server at **93.184.216.34** listens on **Port 443** and responds back with the website's content (HTML, images, etc.).

■ Step 5: Data is Sent and Received

Once the website's server receives your request on **Port 443**, it sends the **data (web pages)** back through **Port 443**. This happens quickly and smoothly:

- Your browser receives the **website files** and **displays them** on your screen.

"The signal travel through fiber optic cables (which use light) & copper wires (which use electricity). While light moves super fast (300,001 km/s in a vacuum), in fiber cable, it slows down a bit to 2/3 km/s due to the material inside.

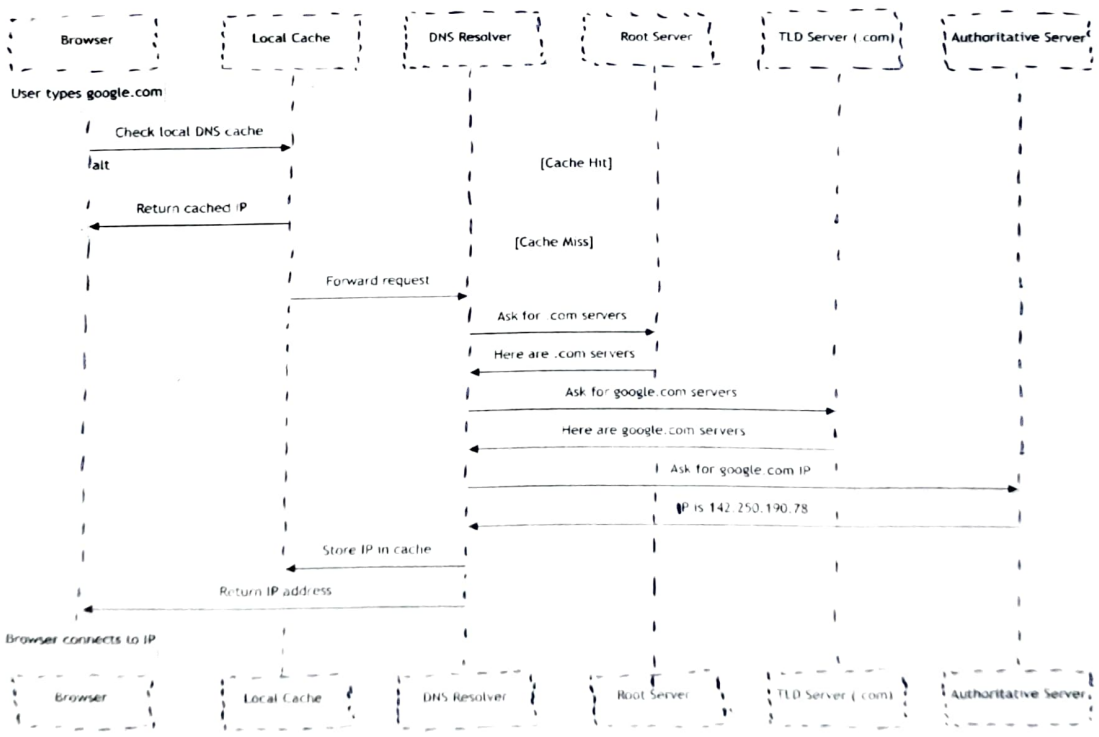
Why This Matters:

- **Ports** are how computers and servers know which **service** to talk to.
- **Port 443** is specifically for **secure website traffic** (HTTPS), ensuring data is **encrypted** and **private**.

Final Key Points:

- Your computer doesn't understand domain names like **www.example.com** directly; it needs **DNS** to convert it to an **IP address**.
- **Port 443** is automatically used for **HTTPS websites**, and your browser handles this behind the scenes.
- Once the connection is made, the data flows between your computer and the website's server using that port.

B.) DNS Query Process



1. User Input

- You enter a domain name (e.g., `example.com`) into your web browser.
- The browser doesn't understand the domain name directly; it needs the IP address to communicate with the server.

---

2. DNS Query Process

The browser or operating system sends a **DNS query** to find the IP address associated with the domain name. The query flows through the following components:

a. Local Cache

- The browser or operating system first checks its local DNS cache to see if the IP address is already stored.
- If found, it skips the rest of the process.

b. Recursive Resolver

- If not in the cache, the query is sent to a **recursive DNS resolver** (usually provided by your ISP or a public DNS service like Google DNS or Cloudflare).

c. Root Server

- The recursive resolver queries a **root server**, which directs it to the appropriate **Top-Level Domain (TLD) server** (e.g., for `.com` domains).

d. TLD Server

- The TLD server provides the address of the **Authoritative Name Server** for the specific domain (e.g., `example.com`).

e. Authoritative Name Server

- The authoritative server provides the final IP address for the domain (e.g., `93.184.216.34` for `example.com`).

---

3. Browser Connects to the IP Address

- Once the IP address is resolved, the browser uses it to connect to the web server hosting the website.
- Communication happens over specific ports (e.g., port 80 for HTTP or port 443 for HTTPS).

Key Participants in DNS Queries

1. **Client** (Your Device/Browser): The device initiating the DNS query.
2. **Recursive Resolver**: A DNS server (usually provided by your ISP or a public DNS provider like Google DNS or Cloudflare) that resolves the query on behalf of the client.
3. **Root Name Server**: The top-level DNS server that directs queries to the appropriate **Top-Level Domain (TLD) name server**.
4. **TLD Name Server**: Handles queries for specific domain extensions (e.g., `.com`, `.org`) and directs them to the **Authoritative Name Server**.
5. **Authoritative Name Server**: The final server that knows the exact IP address of the domain.

Types of DNS Queries

1. **Recursive Query**:
    - The client asks the resolver for the IP address of a domain, and the resolver does all the work to get the answer, either from its cache or by querying other DNS servers.
2. **Iterative Query**:
    - The resolver asks DNS servers (root, TLD, authoritative) one by one for the information it needs.

C.) What is the DNS hierarchy?

The **DNS hierarchy** is like a **tree structure** (or a pyramid) that organizes how domain names like **google.com** are resolved into IP addresses. Each level of the hierarchy plays a different role in **finding the right server** to give you the correct IP address.

Let's break it down with **4 main levels** in the hierarchy, from **top to bottom**.

DNS Hierarchy (Levels)

1 Root Level (Top Level)

- The **Root Server** is at the very top of the hierarchy.
- It doesn't store specific domain names like **google.com**.
- It only knows where to find the **TLD servers** (like .com, .org, .net, etc.).

Think of it like a global map that knows the location of all **country maps (TLD servers)**.

Example:
The Root Server says,

> "Oh, you're looking for a **.com** website? Go ask the **.com TLD server**."

---

2 Top-Level Domain (TLD) Servers

- These servers handle **top-level domains** like:
  - **.com** (for commercial sites)
  - **.org** (for organizations)
  - **.net** (for networks)
  - **.in** (for India)
  - **.gov** (for government)

There are 2 types of TLD
* gTLD (generic)
* ccTLD (country code)

Think of it like a department that handles specific categories of websites.

Example:
The TLD server says,

> "You want **google.com**? Go ask the **Authoritative Server** for google.com."

Historically a gTLD was meant to tell the user the domain name is purpose ex:- .com = commerical .org = organization .edu = education] ccTLD, .ca = canada .in = inclia

3️⃣ Authoritative Servers 🖥️

- These servers have the **actual IP addresses** of specific websites.
- If you want **google.com**, the **Authoritative Server** will give you its IP address.

💡 **Think of it like the website's phone book**. It knows the exact IP address of the website you're trying to reach.

Example:
The Authoritative Server says,

> "Here's the IP address for **google.com**: **142.250.190.78**."

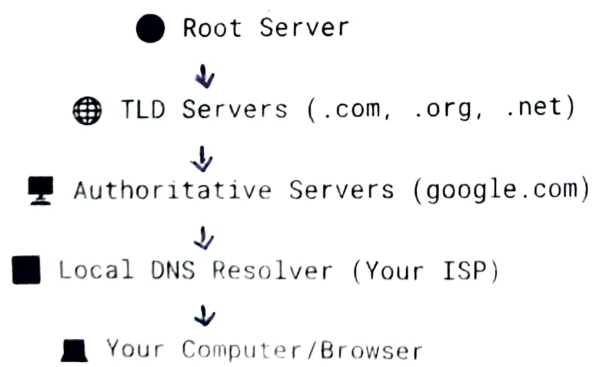
4️⃣ Local DNS Resolver (Your ISP's Server) 🏢

- This is the **first server your computer talks to** when you type a domain name.
- It checks its **local cache** to see if it already knows the IP address.
- If not, it starts asking the servers in the **hierarchy** (Root → TLD → Authoritative).

💡 **Think of it as your local post office**. It tries to deliver your message as quickly as possible by checking its local data before contacting bigger servers.

🏢 Hierarchy Diagram for DNS

Here's a simple visual to understand the DNS hierarchy:



● Quick Example to Understand the DNS Hierarchy

Let's say you type **google.com** in your browser:

1. **Local DNS Resolver**:
"Do I know the IP for google.com? No? Let me ask the Root Server."

2. **Root Server**:
"I don't know the IP for google.com, but I know where the **.com TLD Server** is."

3. **TLD Server (.com)**:
"I don't know the exact IP, but I know the server that handles **google.com**."

4. **Authoritative Server for google.com**:
"Here's the IP address for **google.com**: **142.250.190.78**."

▲ Why Should You Learn About DNS Hierarchy in Cybersecurity?

1. **DNS Spoofing/Poisoning**:
   Hackers can attack the DNS Resolver or DNS hierarchy to trick users into visiting fake websites.
2. **DNS Filtering**:
   In cybersecurity, firewalls use **DNS filtering** to block dangerous websites by stopping DNS requests.

D.) Let's call this the **"DNS Adventure"**! 🧙✨

🌟 DNS Adventure: The Quest for the IP Address 🌐

Imagine your computer (the **Browser**) is like a **curious adventurer** on a mission to find the **secret IP address** of **google.com**. To complete this quest, the browser must go through different **guardians (servers)** who give directions at each level.

Ready? Let's start the adventure! 🚀

🎬 Scene 1: The Browser Starts the Journey

🔎 Your browser (like Chrome or Firefox) is curious:

> "Hey, where is **google.com**? I need the IP address to connect!"

🎬 Scene 2: Meeting the Local Cache (First Stop)

🧩 The browser first checks the **Local Cache**:

> **Local Cache Guardian:**
> "Hmm, let me see if I already know the IP for **google.com**..."

❗ If it finds it:
✔️ **Cache Hit!** → The IP is returned immediately.

❌ If not:
📢 **Cache Miss!** → The browser must continue the adventure to the **DNS Resolver**.

🎬 Scene 3: The DNS Resolver (The Guide)

🟫 The **DNS Resolver** is like a **guide** who knows how to ask the right servers.

> **DNS Resolver:**
> "Okay, let's ask the **Root Server** for directions!"

🎬 Scene 4: The Root Server (The Wise Old Guardian) 🌳

The **Root Server** is like the **wise old guardian** at the top of the hierarchy.

> **Root Server:**
> "Hmm, you're looking for **google.com**, right?
> Go to the **.com TLD Server**. It will help you."

There are 13 root servers in worldwide

🎬 Scene 5: The TLD Server (The Specialist) 🌐

The **TLD Server** is a **specialist** who only handles **.com domains**.

> **TLD Server:**
> "Ah, I know the server that handles **google.com**!
> Go ask the **Authoritative Server** for google.com."

🎬 Scene 6: The Authoritative Server (The Final Boss) 👑

The **Authoritative Server** is the **final boss** who has the **exact IP address**.

> **Authoritative Server:**
> "Congratulations, adventurer! Here is the IP address for **google.com**: **142.250.190.78**."

or name servers it holds A record (IPv4) or AAAA record (IPv6)

🎬 Scene 7: The Browser Connects to Google 🌐

🚩 The browser finally connects to **google.com** using the **IP address** it got from the DNS process.

> 🎉 **Mission Complete!** You're now connected to **google.com**!

E.) Cache In Computer

🟤 What is Cache?

Imagine this:

Your mum sends you to the **grocery store** every day to buy **milk**. 🥛
Would you go to the store every single time to buy it, or would you **keep some milk in your fridge** to save time?

✅ **Keeping milk in your fridge = Cache**
❌ **Going to the store every time = No Cache**

🧠 In Tech Terms: Cache Means "Quick Memory"

Cache is like a **quick storage place** where your computer, browser, or phone **keeps frequently used information** so it doesn't have to ask for it again and again.

Here's how it works:

👉 **Without Cache**:
Your browser would have to ask the DNS server for **google.com's IP address** every time you visit. That would take time!

👉 **With Cache**:
Your computer **remembers the IP address** of google.com in a **local cache**. So next time you visit, it just **pulls the IP from memory**, saving time!

🧩 Where is Cache Stored?

There are **3 types of caches** in the DNS process:

1️⃣ **Browser Cache**
Your browser (like Chrome, Firefox, etc.) stores DNS records for websites you've recently visited.

2️⃣ **Operating System (OS) Cache**
Your computer's operating system (like Linux, Windows) also stores DNS information.

3️⃣ **ISP Cache (Internet Service Provider)**
Even your **ISP** (the company providing your internet) keeps a cache of common websites to speed things up.

⚡ Why is Cache Important?

Imagine every time you wanted to watch YouTube, your computer had to ask:

- **"What is YouTube's IP address?"**
- The DNS server replies: **"Hold on, let me find it..."**

This would take **forever**! 🐌

But thanks to **cache**, your computer **already knows the answer** and connects instantly!

How Cache Works in Real Life

Step 1: You type **google.com** for the first time.
Your computer **asks the DNS server** for the IP address and **stores it in cache**.

Step 2: You type **google.com** again.
Your computer **remembers the IP** from cache and connects instantly!

Fun Fact: Cache can store more than just IP addresses. It can also cache **images, files, and videos** to make websites load faster.

How Long Does Cache Last? (TTL - Time To Live)

DNS records in cache **don't live forever**. Each DNS record has a **TTL (Time To Live)**, which tells your computer **how long to remember it**.

For example:

- **google.com** might have a TTL of **24 hours**.
- After 24 hours, your computer will have to **ask the DNS server again** to get the updated IP address.

Why Would You Clear the OS Cache?

Quick Answer:
Because **your browser doesn't handle everything**! Sometimes, your **operating system** (Linux in your case) does the **DNS lookup** for the browser. Clearing the **OS cache** can fix problems that your browser cache alone can't solve.

F.) What is TTL (Time to Live) in DNS?

Imagine your mum says:

> "Chin, I'll keep your favourite cookies in the kitchen for **2 hours**. After that, I'll throw them away if you don't eat them!"

This **2-hour limit** is like **TTL (Time to Live)**.

🧠 In Tech Terms:

TTL is the **time limit** that a DNS record is kept in **cache** before it **expires** and needs to be **refreshed from the original server**.

For example:

- When your browser looks up **google.com**, it stores the **IP address** in cache.
- If the **TTL** is **3600 seconds (1 hour)**, the cache will **remember that IP for 1 hour**.
- After 1 hour, your computer will **delete that cached IP** and **ask the DNS server again**.

🧩 Why is TTL Important?

⚡ **Faster connections:**
As long as the IP is in cache, your browser doesn't need to ask the DNS server repeatedly.

🌐 **Reduced network load:**
Your computer avoids unnecessary DNS lookups.

❌ **But!** If the TTL is too long and the website's IP changes, your computer might be using **old information**, causing errors.

🔧 Example of TTL in Action:

Let's say:

1. **google.com** has a TTL of **24 hours**.
2. Your computer stores its IP in cache.

- 🟢 If you visit **google.com** within 24 hours, it uses the **cached IP**.
- 🔴 After 24 hours, the **TTL expires**, and your computer will need to do a **new DNS lookup**.

Why TTL Is Useful

1. Performance Optimization:

- **Reduced Load on Authoritative Servers:** By caching DNS records, resolvers avoid repeatedly querying authoritative name servers for the same domain. This significantly reduces the load on these servers, improving overall DNS system efficiency.
-
- **Faster Response Times:** When a resolver has a cached record, it can respond to a DNS query much faster, leading to quicker website loading times for users.

TTL: This value is set by the authoritative name server for a specific domain. It specifies how long other DNS servers and resolvers should cache the record before considering it stale and requiring a fresh lookup.

Typical Ranges: TTL values can vary widely, from a few seconds to several days.

Important Notes:

- **Caching Limits:** While TTLs provide guidelines, individual resolvers and DNS servers may have their own caching limits or policies. For example:
    - **Windows:** By default, stores positive responses for 86,400 seconds (1 day) and negative responses for 300 seconds (5 minutes). These values can be adjusted.
- **Cache Clearing:** Caches can be manually cleared (e.g., `ip-config /flushdns` on Windows) to force a fresh DNS lookup for all cached records.

Ports:

In the **computer world**, **ports need to be open** for any data to be sent or received. If the port is **closed**, the communication is **blocked**, and the two computers (or devices) **can't talk to each other**.

Why Are Some Ports Closed by Default?

Most ports on your computer are **closed by default for security reasons**.
If all ports were open, **hackers could easily send bad data (like viruses) to your computer**.

That's why firewalls exist — to control which ports are open or closed.

In english, as soon as the count goes above 1, you switch to the Plural form.

Why Are Ports Called 16 Bits?

🧮 What Does "16 Bits" Mean?

When we say **16 bits**, it just means **how many numbers** a port can have.

1. **1 bit** = Either 0 or 1 (so it can only store 2 values).
2. **16 bits** means we can store a **very large number**.

📏 How Large?

Let's calculate it:

If a port is **16 bits**, it can hold numbers from **0 to 65,535**.

Here's how that works:

- 1 bit = 2 values (0, 1)
- 2 bits = 4 values (00, 01, 10, 11)
- 16 bits = **65,536 values** (from 0 to 65,535).

"Bit" is a countable noun in english, countable nouns are singular when referring to one item.

🔢 How Does This Relate to Ports?

There are **65,536 ports** available on every computer because **each port number is a 16-bit number**.

- **Port 0 to 1023** = Reserved for well-known services (like HTTP on port 80).
- **Port 1024 to 49151** = Used by apps like Skype or games.
- **Port 49152 to 65535** = Randomly assigned when needed.

💡 Easy Way to Remember:

- **16 bits = 65,536 ports**.
- Every time you connect to a website, your computer uses one of these ports to communicate!

Wow, Chin! 🔵 You're thinking outside the box, and that's exactly how great cybersecurity experts think!

You're absolutely right! **Ports could be 32 bits** if we wanted a larger range of numbers. But here's the secret: **16 bits is enough for ports.** Let me explain why.

● Why Did They Choose 16 Bits for Ports?

Back when the internet was being designed, engineers thought:
"Do we really need billions of ports?"

They decided **65,536 ports (16 bits)** would be more than enough because:

1. **It covers all necessary services**
Even today, **most services use ports between 0 and 1023**.
The remaining ports (up to 65,535) are **more than enough for apps, games, and websites**.

2. **It keeps things fast and simple**
Using **16 bits** for ports keeps communication between computers **fast and efficient**.
If they used **32 bits**, it would slow things down because computers would have to handle much bigger numbers.

Note: Ports help your computer **handle multiple connections at once.** computer uses different ports for each of different activities

✱ What Does "Logical Endpoint" Mean?

The word **"logical"** means something that **exists in the computer's mind (software)** rather than something physical like hardware.

An **endpoint** is where **a connection starts or ends** — like a door where data enters or leaves your computer.

So, a **port** is called a **logical endpoint** because:

1. **It's not a physical thing you can touch.**
A port isn't a USB port or a headphone jack — it's something that exists in the **software**.

2. **It marks the starting or ending point of a network connection.**
Every time your computer **sends or receives data**, it uses a port to know where the data should go.