DNS

DNS = Domain Name System = Phonebook of internet. It translates Domain Names to IP Addresses and managed by ICANN (Internet Corporation for Assigned Names and Numbers) which is a non-profit organization.

The reason why we need Domain names because remembering numbers especially IP addresses are not easy for humans but remembering names are comparatively easy.

How DNS works?

When you visit www.codersgyan.com, the process of finding the website's IP address is similar to asking a librarian where a book is located.

Step–1 :– Checking Local Cache

Before making a request to an external server, your system first checks its local storage:

Browser Cache: If you've visited the website before, your browser may have already stored its IP address.
OS-Level Cache (DNS Client): If the browser doesn't have the record, the operating system looks into its DNS cache to find a match.

Step–2 :– DNS Resolver/Recursor

If the local cache doesn't have the record, the request is sent to a DNS Resolver—like a librarian who helps find the right section.

By default, your Internet Service Provider (ISP) provides a resolver.
You can also use third-party resolvers like Google's (8.8.8.8) or Cloudflare's (1.1.1.1) for potentially faster or more secure lookups.

Step–3 :– Root Nameservers

The DNS Resolver doesn't know the exact location but knows where to start looking. It asks the Root Nameserver, which acts like an index in a library. The Root Nameserver directs the resolver to the correct Top-Level Domain (TLD) server based on the domain extension (e.g., .com).

Step–4: TLD Nameserver

The .com TLD nameserver acts like a section in the library dedicated to .com domains. It knows where to find codersgyan.com and directs the resolver to the Authoritative Nameserver.

Step–5: Authoritative Nameserver

The Authoritative Nameserver holds the final answer. It provides the exact IP address for codersgyan.com, much like a bookshelf contains the book you were looking for.

In this case, codersgyan.com is stored at 104.21.16.1.

Now that the resolver has the IP address, it sends it back to your browser, which then loads the website.

Why not direct communication with ICANN

Our request doesn't go directly to ICANN (the organization managing domain names). Instead, it interacts with DNS Resolvers, which handle communication with the Root, TLD, and Authoritative Nameservers. This distributed system makes the process efficient and scalable.

Why Do DNS Updates Take Time?

DNS changes take time to propagate globally because Root, TLD, and Authoritative Nameservers around the world need to update their records. This delay is due to the Time-to-Live (TTL) settings, which determine how long DNS records are cached before refreshing.

DNS Caching

To speed up the process, DNS caching is used at multiple levels:

Browser Cache: The browser stores previously visited domain records to avoid repeated lookups.

OS-Level Cache (DNS Client): The operating system keeps a temporary DNS cache.

DNS Resolver Cache: ISPs and major DNS providers (e.g., Google 8.8.8.8, Cloudflare 1.1.1.1) maintain large caches to optimize domain lookups and reduce DNS traffic.

This caching system ensures that websites load faster and reduces the overall burden on DNS infrastructure

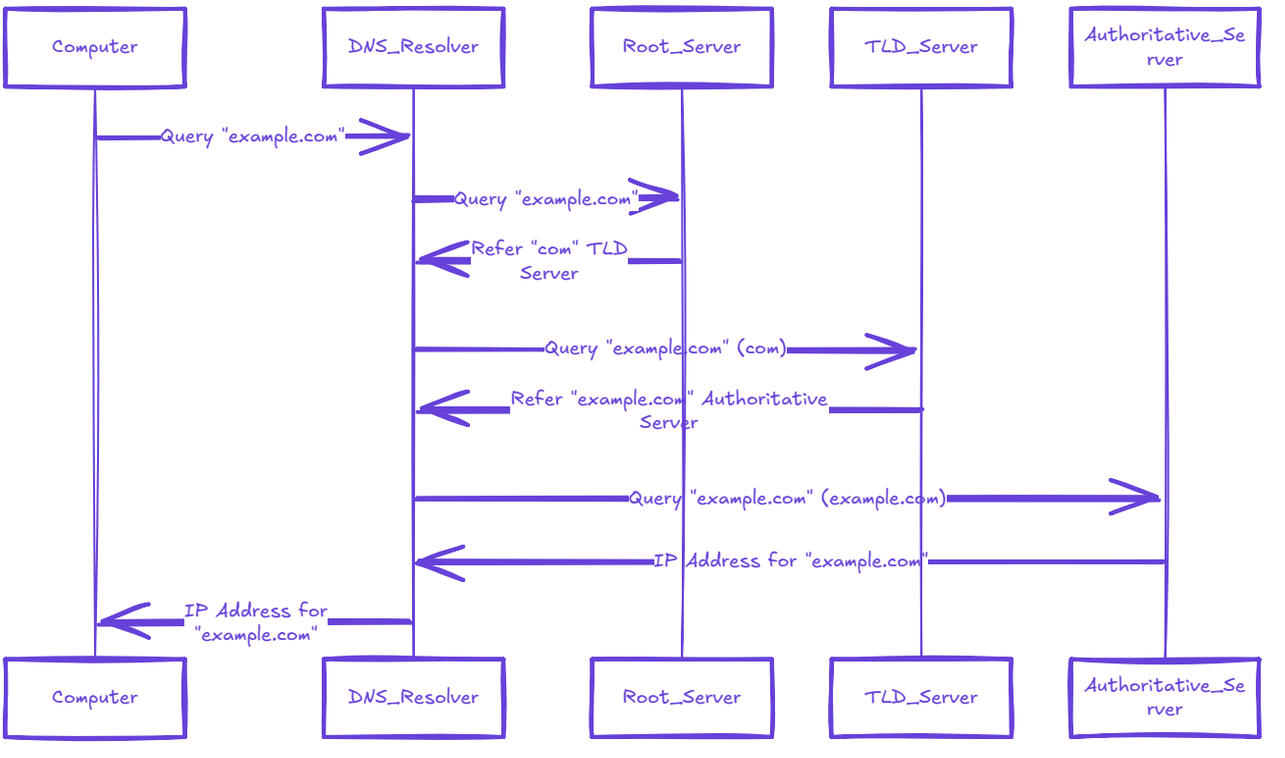


Figure DNS resolution sequence

TCP Protocol

TCP (Transmission Control Protocol) is one of the main protocols of the TCP/IP suite. It lies between the Application and Network Layers which are used in providing reliable delivery services.

- It is a connection-oriented protocol
- It breaks down the data into small bundles and afterward reassembles the bundles into the original message on the opposite end to make sure that each message reaches its target location intact.
- These bundles may travel along multiple routes if one route is jammed but the destination remains the same.

Why do we use TCP?

Reliability:

TCP's acknowledgment and retransmission mechanism ensures that if a segment is lost or corrupted, it will be resent. This is critical for applications where data integrity is a priority (e.g., file transfers, emails).

Ordered Data Delivery:

TCP guarantees that the data will be received in the same sequence in which it was sent, preventing misordered data issues common in lower-level protocols.

Congestion Control and Flow Control:

By adjusting the rate of data transmission based on network conditions and receiver capacity, TCP optimizes throughput and avoids overwhelming the network or the receiving host.

Universality:

TCP is a fundamental protocol used by many well-known applications and services (HTTP, SMTP, FTP, etc.). Adopting TCP means benefiting from widespread support and a mature, stable protocol.

Error Detection and Correction:

Every segment has a checksum that the receiver can use to detect transmission errors, prompting a retransmission if necessary.

TCP Headers:

Every segment in the TCP is being attached a TCP header. The size of TCP header is can vary from 20 - 60 bytes. A standard TCP Header with data looks like this.

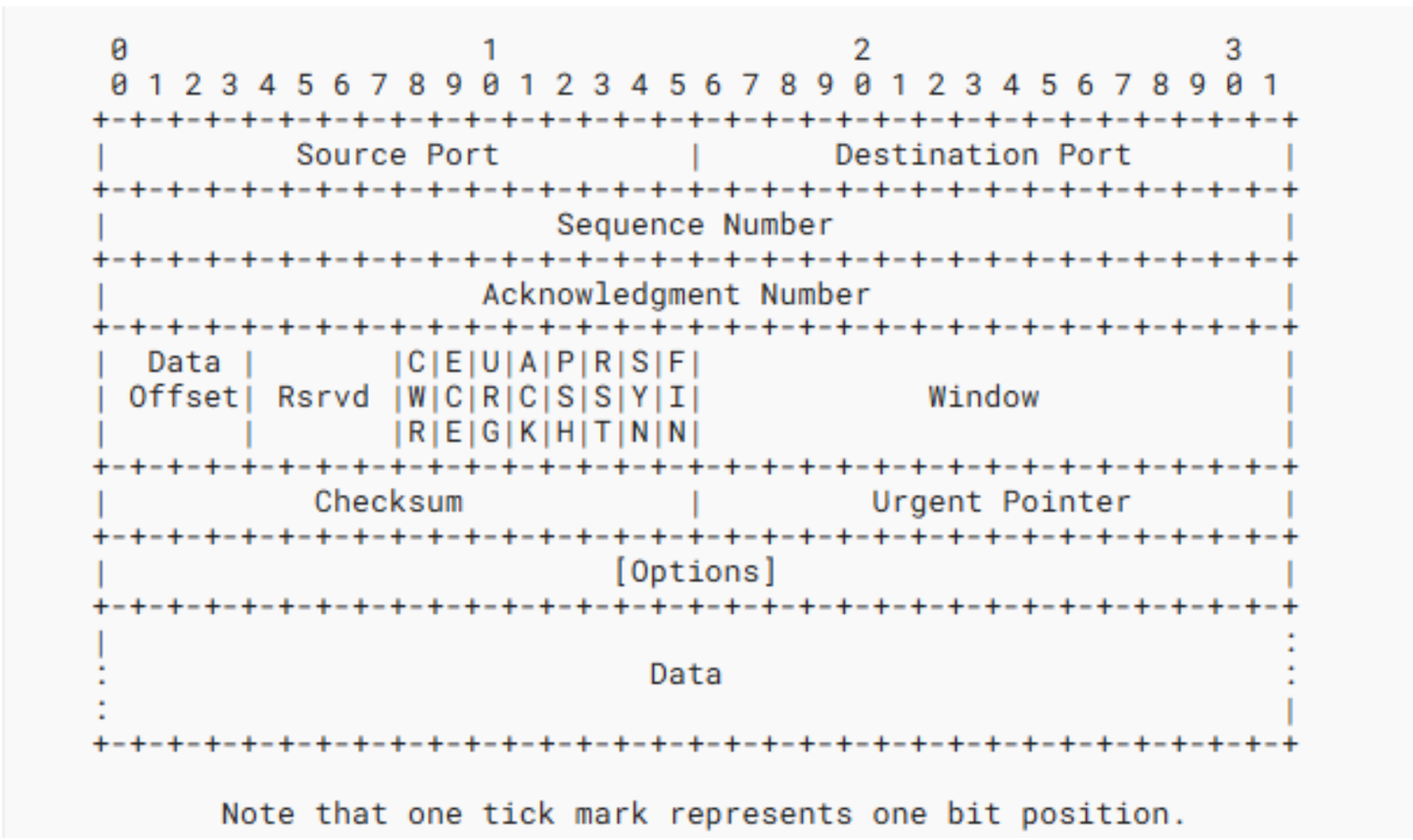


Figure. TCP Header format

The port number of the sending application on the sender's side

Destination Port:

The port number of the receiving application on the receiver's side

Sequence Number

Indicates the byte position of the first byte of data in the current segment

Acknowledgement Number

The sequence number of the next byte expected by the receiver (used for acknowledgment of receipt).

Checksum

A value used for error checking to ensure the data is not corrupted during transmission.

3-Way Handshake

The TCP Three-Way Handshake is the process used to establish a reliable connection between a client and a server before data transfer begins. Which means it comes even before the application layer to establish the connection.

Step 1: Client Sends SYN

Step	Sender → Receiver	Seq No	Ack No	Flags
1❑	Client → Server	1000	0	SYN

The client wants to start a TCP connection
It sends a SYN (Synchronize) packet to the server
The SYN packet contains an initial sequence number (ISN) of 1000 (randomly chosen for security reason)
The acknowledgment number is 0 because it's the first message in the handshake.

Step 2: Server Sends SYN-ACK

Step	Sender → Receiver	Seq No	Ack No	Flags
2️⃣	Server → Client	5000	1001	SYN-ACK

The server receives the SYN from the client and responds with SYN-ACK (Synchronization + Acknowledgment).It sends a SYN (Synchronize) packet to the server

The server chooses its own Initial Sequence Number (ISN) = 5000.
It acknowledges the client's request by setting Ack No = 1001 (Client's ISN + 1).

The SYN-ACK packet signals that the server is ready to communicate.

✅ Server acknowledges the connection request and sends its own SYN.

Step 3: Step 3: Client Sends ACK

Step	Sender → Receiver	Seq No	Ack No	Flags
3️⃣	Client → Server	1001	5001	ACK

The client receives the SYN-ACK from the server.

It sends an ACK (Acknowledgment) packet back to the server.

The sequence number remains 1001 (unchanged).

The acknowledgment number = 5001 (Server's ISN + 1), confirming the receipt of the SYN-ACK.

✅ Now, both client and server have agreed on starting the communication. The connection is established! 🎉

3 way TCP handshake diagram

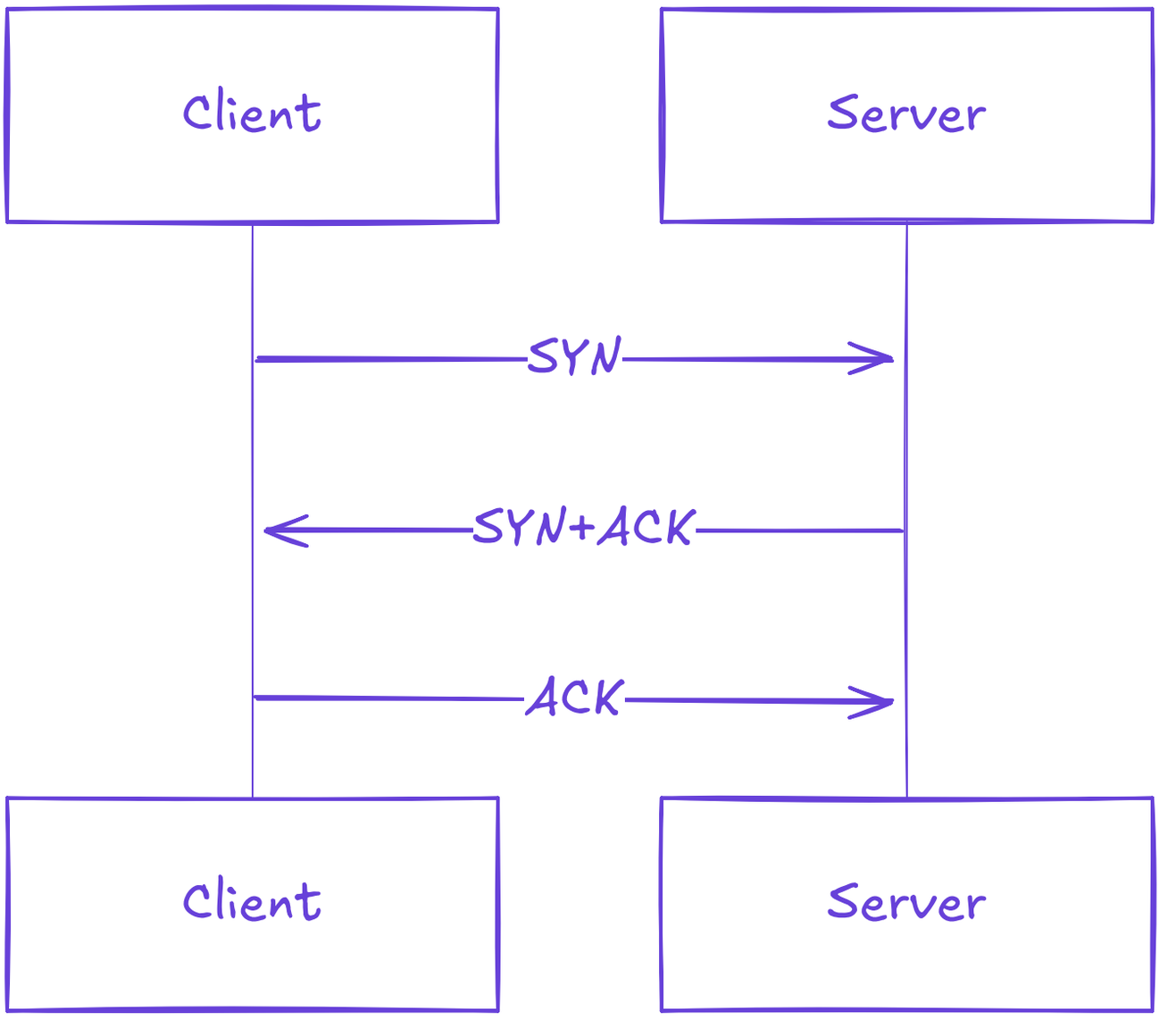


Figure. Showing TCP 3 way handshake connection establishment

Practical Explanation

Let's imagine we need to transfer an image of 3 MB from our computer to server. Before sending anything a 3-way handshake will happen to establish a TCP Connection.

Step–1: 3–Way handshake

Step Sender → Receiver Seq No Ack No Flags
1⃣ Client → Server 1000 0 SYN
2⃣ Server → Client 5000 1001 SYN-ACK
3⃣ Client → Server 1001 5001 ACK

✅ Connection Established. Now the client can start sending data.

Step–2: Creating Segments

Total file size = 3,145,728 bytes (3MB)
MSS (Max Segment Size) = 1460 bytes per segment
Total segments required = 2155 segments (since 3,145,728 ÷ 1460 ≈ 2155)

Step–3: send first segment

Step Sender → Receiver Seq No Ack No Payload

4⃣ Client → Server 1001 5001 1460B

5⃣ Server → Client 5001 2461 ACK

✅ Server acknowledges receipt of first 1460 bytes.

Step–4: send second segment

Step Sender → Receiver Seq No Ack No Payload
6⃣ Client → Server 2461 5001 1460 bytes
7⃣ Server → Client 5001 3921 ACK

✅ Server acknowledges up to byte 3921.

The process repeats until all 3MB of data is sent and acknowledged.
If any

TCP Retransmission Retries

When a packet is lost or checksum fails on server then no acknowledgment (ACK) is received, TCP will retry (retransmit) the packet after a timeout period. This timeout is called the Retransmission Timeout (RTO)

The initial timeout (RTO) is dynamically calculated based on network conditions.
Typical initial values:
Linux: ~200ms to 3 seconds
Windows: ~300ms to 3 seconds
RFC 6298 recommends 1 second as the default

TCP uses an exponential backoff strategy:
After each failed retry, the timeout doubles.
Example: 1s → 2s → 4s → 8s → 16s → ...

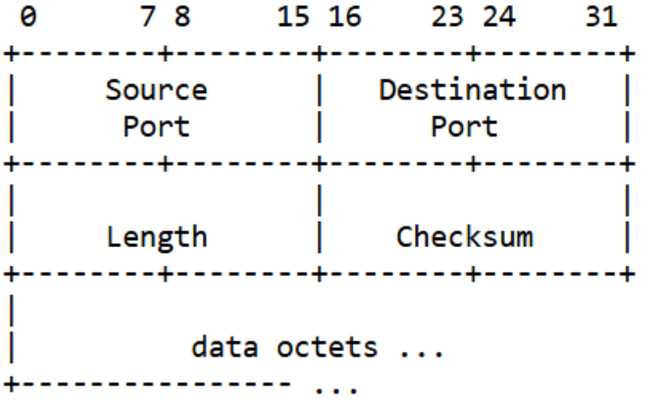
Linux: Default is 15 retries (can take ~15-30 minutes).
Windows: Default is 5 retries (~255 seconds).
MacOS: Similar to Linux (~15 retries).

UDP Protocol (User datagram Protocol)

UDP is one of the other protocols that is used in the Transport layer. It also breaks the data that is received from application to segments so that they be sends from different routes to destination.

Like TCP, it also breaks the data in to chunks which are knows as segments. To rearrange these segments on destination, side a checksum is maintained similar to TCP.

Source port and Destination port is assigned to each segment here.
Unlike TCP, UDP does not ensures guarantee of complete data transfer which means it does not have acknowledgement number.
Since there is no retrying mechanism for lost packets so UDP is fast as compared to TCP.
It is used in Video Calling (WebRTC), Online Games, DNS etc.



User Datagram Header Format

Field	Size (bits)	Size (bytes)
Source Port	16 bits	2 bytes
Destination Port	16 bits	2 bytes
Length	16 bits	2 bytes
Checksum	16 bits	2 bytes
Total	64 bits	8 bytes