NETWORKING AND COMMUNICATION

IP

An IP address is a unique address that identifies a device on the internet or a local network. IP stands for "Internet Protocol", which is the set of rules governing the format of data sent via the internet or local network.

In essence, IP addresses are the identifier that allows information to be sent between devices on a network. They contain location information and make devices accessible for communication. The internet needs a way to differentiate between different computers, routers, and websites. IP addresses provide a way of doing so and form an essential part of how the internet works.

Versions

Now, let's learn about the different versions of IP addresses:

IPv4

The original Internet Protocol is IPv4 which uses a 32-bit numeric dot-decimal notation that only allows for around 4 billion IP addresses. Initially, it was more than enough but as internet adoption grew, we needed something better.

Example: 102.22.192.181

IPv6

IPv6 is a new protocol that was introduced in 1998. Deployment commenced in the mid-2000s and since the internet users have grown exponentially, it is still ongoing.

This new protocol uses 128-bit alphanumeric hexadecimal notation. This means that IPv6 can provide about ~340e+36 IP addresses. That's more than enough to meet the growing demand for years to come.

Example: 2001:0db8:85a3:0000:0000:8a2e:0370:7334

Types

Let's discuss types of IP addresses:

Public

A public IP address is an address where one primary address is associated with your whole network. In this type of IP address, each of the connected devices has the same IP address.

Example: IP address provided to your router by the ISP.

Private

A private IP address is a unique IP number assigned to every device that connects to your internet network, which includes devices like computers, tablets, and smartphones, which are used in your household.

Example: IP addresses generated by your home router for your devices.

Static

A static IP address does not change and is one that was manually created, as opposed to having been assigned. These addresses are usually more expensive but are more reliable.

Example: They are usually used for important things like reliable geo-location services, remote access, server hosting, etc.

Dynamic

A dynamic IP address changes from time to time and is not always the same. It has been assigned by a Dynamic Host Configuration Protocol (DHCP) server. Dynamic

IP addresses are the most common type of internet protocol address. They are cheaper to deploy and allow us to reuse IP addresses within a network as needed.

Example: They are more commonly used for consumer equipment and personal use.

OSI Model

The OSI Model is a logical and conceptual model that defines network communication used by systems open to interconnection and communication with other systems. The Open System Interconnection (OSI Model) also defines a logical network and effectively describes computer packet transfer by using various layers of protocols.

The OSI Model can be seen as a universal language for computer networking. It's based on the concept of splitting up a communication system into seven abstract layers, each one stacked upon the last.

Why does the OSI model matter?

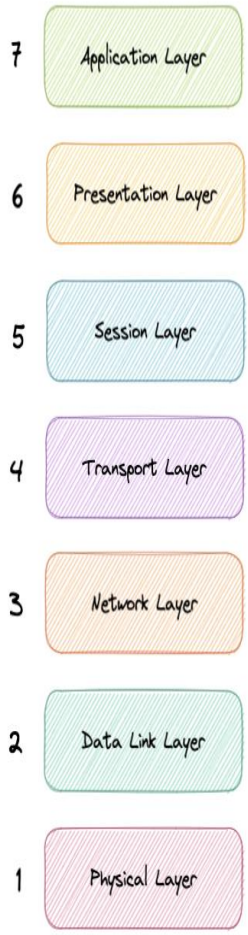
The Open System Interconnection (OSI) model has defined the common terminology used in networking discussions and documentation. This allows us to take a very complex communications process apart and evaluate its components.

While this model is not directly implemented in the TCP/IP networks that are most common today, it can still help us do so much more, such as:

- Make troubleshooting easier and help identify threats across the entire stack.
- Encourage hardware manufacturers to create networking products that can communicate with each other over the network.
- Essential for developing a security-first mindset.
- Separate a complex function into simpler components.

Network Layers

The seven abstraction layers of the OSI model can be defined as follows, from top to bottom:



---

1. Application

This is the only layer that directly interacts with data from the user. Software applications like web browsers and email clients rely on the application layer to initiate communication. But it should be made clear that client software applications are not part of the application layer, rather the application layer is responsible for the protocols and data manipulation that the software relies on to present

meaningful data to the user. Application layer protocols include HTTP as well as SMTP.

2. Presentation

The presentation layer is also called the Translation layer. The data from the application layer is extracted here and manipulated as per the required format to transmit over the network. The functions of the presentation layer are translation, encryption/decryption, and compression.

3. Session

This is the layer responsible for opening and closing communication between the two devices. The time between when the communication is opened and closed is known as the session. The session layer ensures that the session stays open long enough to transfer all the data being exchanged, and then promptly closes the session in order to avoid wasting resources. The session layer also synchronizes data transfer with checkpoints.

4. Transport

The transport layer (also known as layer 4) is responsible for end-to-end communication between the two devices. This includes taking data from the session layer and breaking it up into chunks called segments before sending it to the Network layer (layer 3). It is also responsible for reassembling the segments on the receiving device into data the session layer can consume.

5. Network

The network layer is responsible for facilitating data transfer between two different networks. The network layer breaks up segments from the transport layer into smaller units, called packets, on the sender's device, and reassembles these packets on the receiving device. The network layer also finds the best physical path for the data to reach its destination this is known as routing. If the two devices communicating are on the same network, then the network layer is unnecessary.

6. Data Link

The data link layer is very similar to the network layer, except the data link layer facilitates data transfer between two devices on the same network. The data link layer takes packets from the network layer and breaks them into smaller pieces called frames.

7. Physical

This layer includes the physical equipment involved in the data transfer, such as the cables and switches. This is also the layer where the data gets converted into a bit stream, which is a string of 1s and 0s. The physical layer of both devices must also agree on a signal convention so that the 1s can be distinguished from the 0s on both devices.

OSI (Open Systems Interconnection) 7 Layer Model

Layer	Application/Example		Central Device/ Protocols	
Application (7) Serves as the window for users and application processes to access the network services.	**End User layer** Program that opens what was sent or creates what is to be sent Resource sharing • Remote file access • Remote printer access • Directory services • Network management		**User Applications** SMTP	
Presentation (6) Formats the data to be presented to the Application layer. It can be viewed as the "Translator" for the network.	**Syntax layer** encrypt & decrypt (if needed) Character code translation • Data conversion • Data compression • Data encryption • **Character Set Translation**		JPEG/ASCII EBDIC/TIFF/GIF PICT	
Session (5) Allows session establishment between processes running on different stations.	**Synch & send to ports** (logical ports) Session establishment, maintenance and termination • Session support - perform security, name recognition, logging, etc.		**Logical Ports** RPC/SQL/NFS NetBIOS names	
Transport (4) Ensures that messages are delivered error-free, in sequence, and with no losses or duplications.	**TCP** Host to Host, Flow Control Message segmentation • Message acknowledgement • Message traffic control • Session multiplexing	PACKET FILTERING	TCP/SPX/UDP	
Network (3) Controls the operations of the subnet, deciding which physical path the data takes.	**Packets** ("letter", contains IP address) Routing • Subnet traffic control • Frame fragmentation • Logical-physical address mapping • Subnet usage accounting		**Routers** IP/IPX/ICMP	
Data Link (2) Provides error-free transfer of data frames from one node to another over the Physical layer.	**Frames** ("envelopes", contains MAC address) [NIC card — Switch — NIC card] (end to end) Establishes & terminates the logical link between nodes • Frame traffic control • Frame sequencing • Frame acknowledgment • Frame delimiting • Frame error checking • Media access control		**Switch** **Bridge** **WAP** PPP/SLIP	Land Based Layers
Physical (1) Concerned with the transmission and reception of the unstructured raw bit stream over the physical medium.	**Physical structure** Cables, hubs, etc. Data Encoding • Physical medium attachment • Transmission technique - Baseband or Broadband • Physical medium transmission Bits & Volts		**Hub**	

Hypertext transfer protocol (HTTP)

HTTP is a method for encoding and transporting data between a client and a server. It is a request/response protocol: clients issue requests and servers issue responses with relevant content and completion status info about the request. HTTP is self-contained, allowing requests and responses to flow through many intermediate routers and servers that perform load balancing, caching, encryption, and compression.

A basic HTTP request consists of a verb (method) and a resource (endpoint). Below are common HTTP verbs:

Verb	Description	Idempotent*	Safe	Cacheable
GET	Reads a resource	Yes	Yes	Yes
POST	Creates a resource or trigger a process that handles data	No	No	Yes if response contains freshness info
PUT	Creates or replace a resource	Yes	No	No
PATCH	Partially updates a resource	No	No	Yes if response contains freshness info
DELETE	Deletes a resource	Yes	No	No

*Can be called many times without different outcomes.

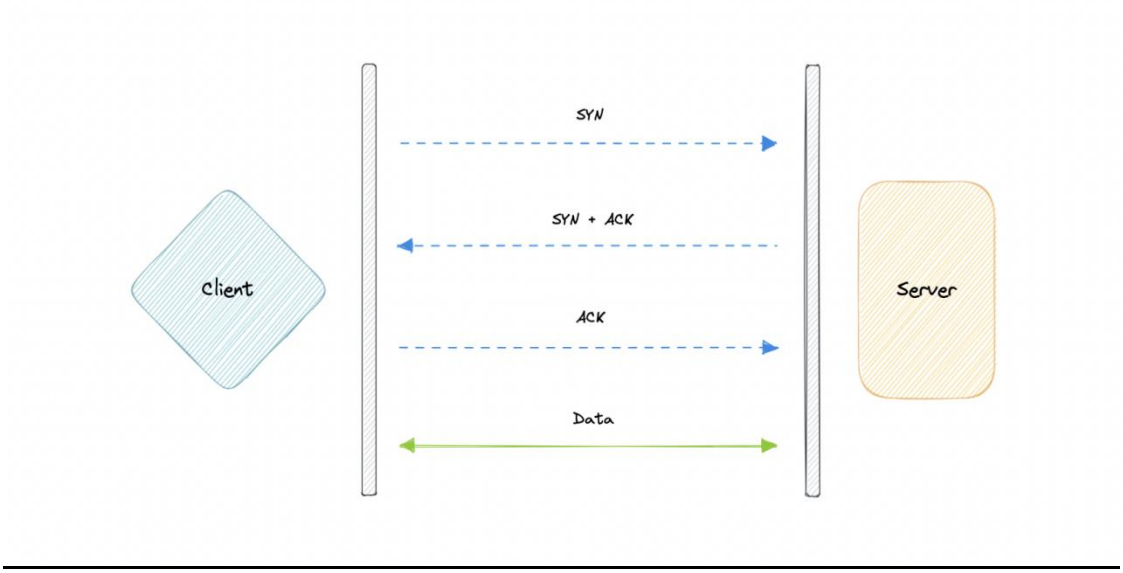
HTTP is an application layer protocol relying on lower-level protocols such as **TCP** and **UDP**.

Sources and further reading: HTTP

- What is HTTP?
- Difference between HTTP and TCP
- Difference between PUT and PATCH

TCP

Transmission Control Protocol (TCP) is connection-oriented, meaning once a connection has been established, data can be transmitted in both directions. TCP has built-in systems to check for errors and to guarantee data will be delivered in the order it was sent, making it the perfect protocol for transferring information like still images, data files, and web pages.



TCP is a connection-oriented protocol over an IP network. Connection is established and terminated using a handshake. All packets sent are guaranteed to reach the destination in the original order and without corruption through:

- Sequence numbers and checksum fields for each packet
- Acknowledgement packets and automatic retransmission

If the sender does not receive a correct response, it will resend the packets. If there are multiple timeouts, the connection is dropped. TCP also implements flow control and congestion control. These guarantees cause delays and generally result in less efficient transmission than UDP.

To ensure high throughput, web servers can keep a large number of TCP connections open, resulting in high memory usage. It can be expensive to have a large number of open connections between web server threads and say, a

memcached server. Connection pooling can help in addition to switching to UDP where applicable.

TCP is useful for applications that require high reliability but are less time critical. Some examples include web servers, database info, SMTP, FTP, and SSH.

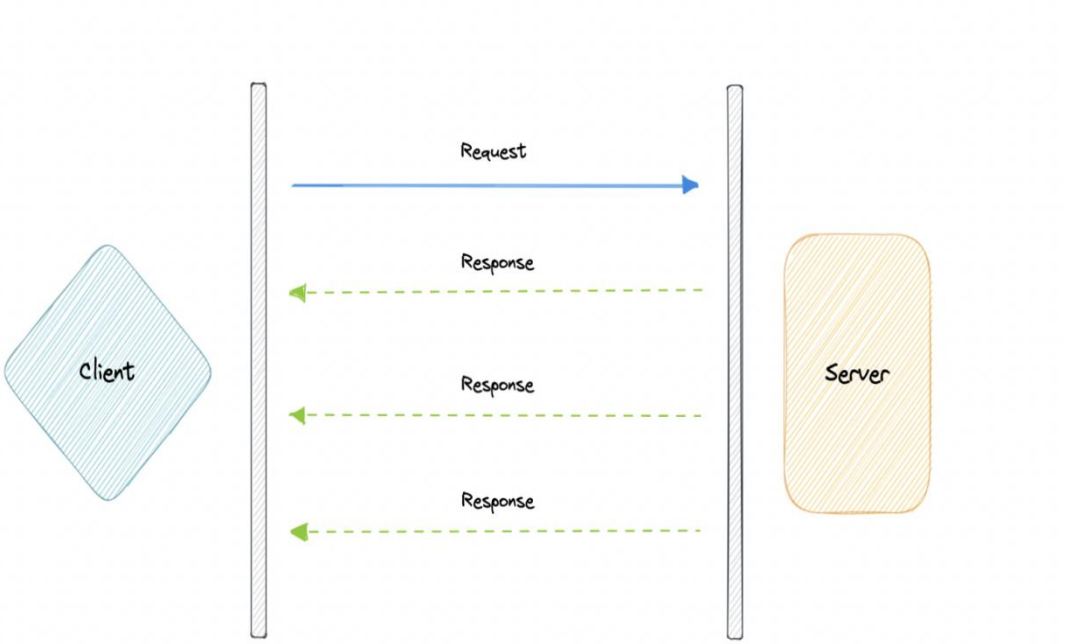
Use TCP over UDP when:

- You need all of the data to arrive intact
- You want to automatically make a best estimate use of the network throughput

But while TCP is instinctively reliable, its feedback mechanisms also result in a larger overhead, translating to greater use of the available bandwidth on the network.

UDP

User Datagram Protocol (UDP) is a simpler, connection less internet protocol in which error-checking and recovery services are not required. With UDP, there is no overhead for opening a connection, maintaining a connection, or terminating a connection. Data is continuously sent to the recipient, whether or not they receive it.



UDP is connectionless. Datagrams (analogous to packets) are guaranteed only at the datagram level. Datagrams might reach their destination out of order or not at all. UDP does not support congestion control. Without the guarantees that TCP support, UDP is generally more efficient.

UDP can broadcast, sending datagrams to all devices on the subnet. This is useful with DHCP because the client has not yet received an IP address, thus preventing a way for TCP to stream without the IP address.

UDP is less reliable but works well in real time use cases such as VoIP, video chat, streaming, and realtime multiplayer games.

It is largely preferred for real-time communications like broadcast or multicast network transmission. We should use UDP over TCP when we need the lowest latency and late data is worse than the loss of data.

Use UDP over TCP when:

- You need the lowest latency
- Late data is worse than loss of data
- You want to implement your own error correction

Sources and further reading: TCP and UDP

- Networking for game programming
- Key differences between TCP and UDP protocols
- Difference between TCP and UDP
- Transmission control protocol
- User datagram protocol
- Scaling memcache at Facebook

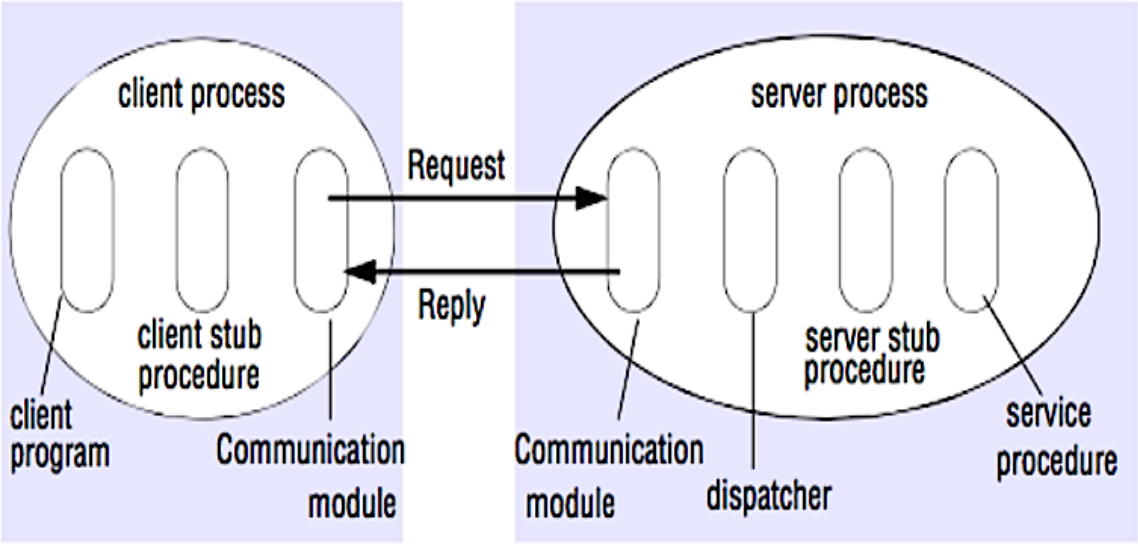
TCP vs UDP

TCP is a connection-oriented protocol, whereas UDP is a connectionless protocol. A key difference between TCP and UDP is speed, as TCP is comparatively slower than UDP. Overall, UDP is a much faster, simpler, and more efficient protocol, however, retransmission of lost data packets is only possible with TCP.

TCP provides ordered delivery of data from user to server (and vice versa), whereas UDP is not dedicated to end-to-end communications, nor does it check the readiness of the receiver.

Feature	TCP	UDP
Connection	Requires an established connection	Connectionless protocol
Guaranteed delivery	Can guarantee delivery of data	Cannot guarantee delivery of data
Re-transmission	Re-transmission of lost packets is possible	No re-transmission of lost packets
Speed	Slower than UDP	Faster than TCP
Broadcasting	Does not support broadcasting	Supports broadcasting
Use cases	HTTPS, HTTP, SMTP, POP, FTP, etc	Video streaming, DNS, VoIP, etc

Remote procedure call (RPC)



In an RPC, a client causes a procedure to execute on a different address space, usually a remote server. The procedure is coded as if it were a local procedure call, abstracting away the details of how to communicate with the server from the client program. Remote calls are usually slower and less reliable than local calls so it is

helpful to distinguish RPC calls from local calls. Popular RPC frameworks include Protobuf, Thrift, and Avro.

RPC is a request-response protocol:

- **Client program** - Calls the client stub procedure. The parameters are pushed onto the stack like a local procedure call.
- **Client stub procedure** - Marshals (packs) procedure id and arguments into a request message.
- **Client communication module** - OS sends the message from the client to the server.
- **Server communication module** - OS passes the incoming packets to the server stub procedure.
- **Server stub procedure** - Unmarshalls the results, calls the server procedure matching the procedure id and passes the given arguments.
- The server response repeats the steps above in reverse order.

Sample RPC calls:

```
GET /someoperation?data=anId
```

```
POST /anotheroperation
{
   "data":"anId";
   "anotherdata": "another value"
}
```

RPC is focused on exposing behaviors. RPCs are often used for performance reasons with internal communications, as you can hand-craft native calls to better fit your use cases.

Choose a native library (aka SDK) when:

- You know your target platform.
- You want to control how your "logic" is accessed.
- You want to control how error control happens off your library.
- Performance and end user experience is your primary concern.

HTTP APIs following **REST** tend to be used more often for public APIs.

Disadvantages: RPC

- RPC clients become tightly coupled to the service implementation.
- A new API must be defined for every new operation or use case.
- It can be difficult to debug RPC.
- You might not be able to leverage existing technologies out of the box. For example, it might require additional effort to ensure RPC calls are properly cached on caching servers such as Squid.

Representational state transfer (REST)

REST is an architectural style enforcing a client/server model where the client acts on a set of resources managed by the server. The server provides a representation of resources and actions that can either manipulate or get a new representation of resources. All communication must be stateless and cacheable.

There are four qualities of a **RESTful interface**:

- **Identify resources (URI in HTTP)** - use the same URI regardless of any operation.
- **Change with representations (Verbs in HTTP)** - use verbs, headers, and body.
- **Self-descriptive error message (status response in HTTP)** - Use status codes, don't reinvent the wheel.
- HATEOAS **(HTML interface for HTTP)** - your web service should be fully accessible in a browser.

Sample REST calls:

```
GET /someresources/anId
```

```
PUT /someresources/anId

{"anotherdata": "another value"}
```

REST is focused on exposing data. It minimizes the coupling between client/server and is often used for public HTTP APIs. REST uses a more generic and uniform method of exposing resources through URIs, representation through headers, and actions through verbs such as GET, POST, PUT, DELETE, and PATCH. Being stateless, REST is great for horizontal scaling and partitioning.

Disadvantages: REST

- With REST being focused on exposing data, it might not be a good fit if resources are not naturally organized or accessed in a simple hierarchy. For example, returning all updated records from the past hour matching a particular set of events is not easily expressed as a path. With REST, it is likely to be implemented with a combination of URI path, query parameters, and possibly the request body.
- REST typically relies on a few verbs (GET, POST, PUT, DELETE, and PATCH) which sometimes doesn't fit your use case. For example, moving expired documents to the archive folder might not cleanly fit within these verbs.
- Fetching complicated resources with nested hierarchies requires multiple round trips between the client and server to render single views, e.g. fetching content of a blog entry and the comments on that entry. For mobile applications operating in variable network conditions, these multiple roundtrips are highly undesirable.
- Over time, more fields might be added to an API response and older clients will receive all new data fields, even those that they do not need, as a result, it bloats the payload size and leads to larger latencies.

RPC and REST calls comparison

Operation	RPC	REST
Signup	**POST** /signup	**POST** /persons

Resign	**POST** /resign
	{
	"personid": "1234"
	}
		DELETE /persons/1234

Read a person	**GET** /readPerson?personid=1234	**GET** /persons/1234
Read a person's items list	**GET** /readUsersItemsList?personid=1234	**GET** /persons/1234/items

Add an item to a person's items	**POST** /addItemToUsersItemsList
	{
	"personid": "1234";
	"itemid": "456"
	}
		POST /persons/1234/items
		{
		"itemid": "456"
		}

Update an item	**POST** /modifyItem
	{
	"itemid": "456";
	"key": "value"
	}
		PUT /items/456
		{
		"key": "value"
		}

Delete an item	**POST** /removeItem
	{
	"itemid": "456"
	}
		DELETE /items/456

Sources and further reading: REST and RPC

- Do you really know why you prefer REST over RPC
- When are RPC-ish approaches more appropriate than REST?
- REST vs JSON-RPC

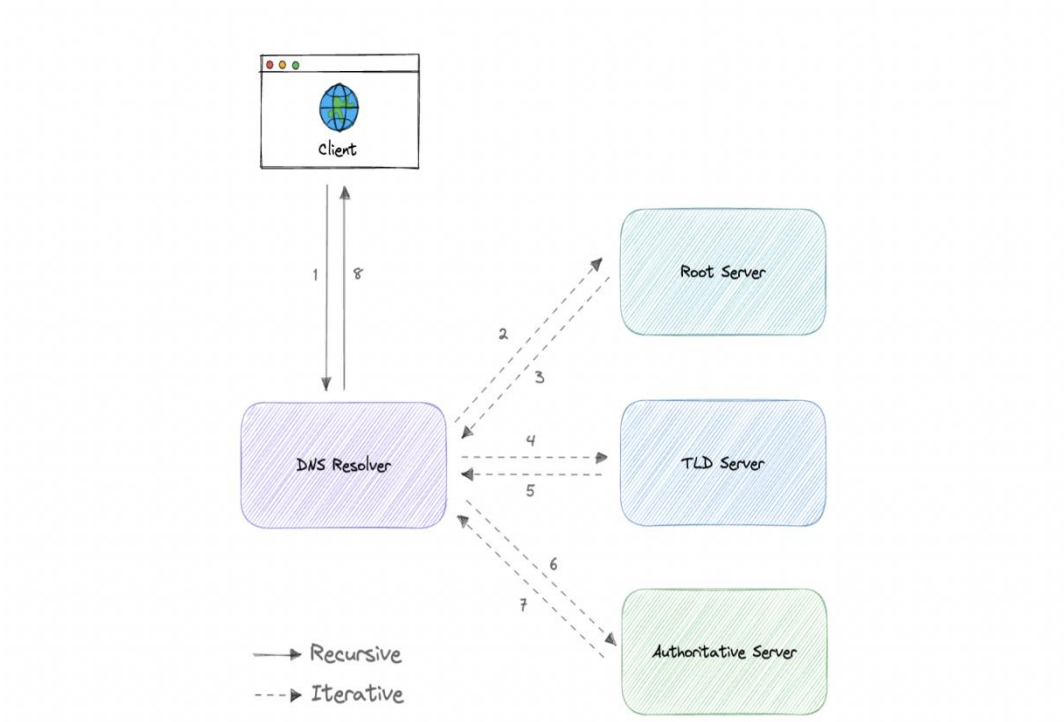
- Debunking the myths of RPC and REST
- What are the drawbacks of using REST
- Crack the system design interview
- Thrift
- Why REST for internal use and not RPC

Domain Name System (DNS)

Earlier we learned about IP addresses that enable every machine to connect with other machines. But as we know humans are more comfortable with names than numbers. It's easier to remember a name like google.com than something like 122.250.192.232.

This brings us to Domain Name System (DNS) which is a hierarchical and decentralized naming system used for translating human-readable domain names to IP addresses.

How DNS works

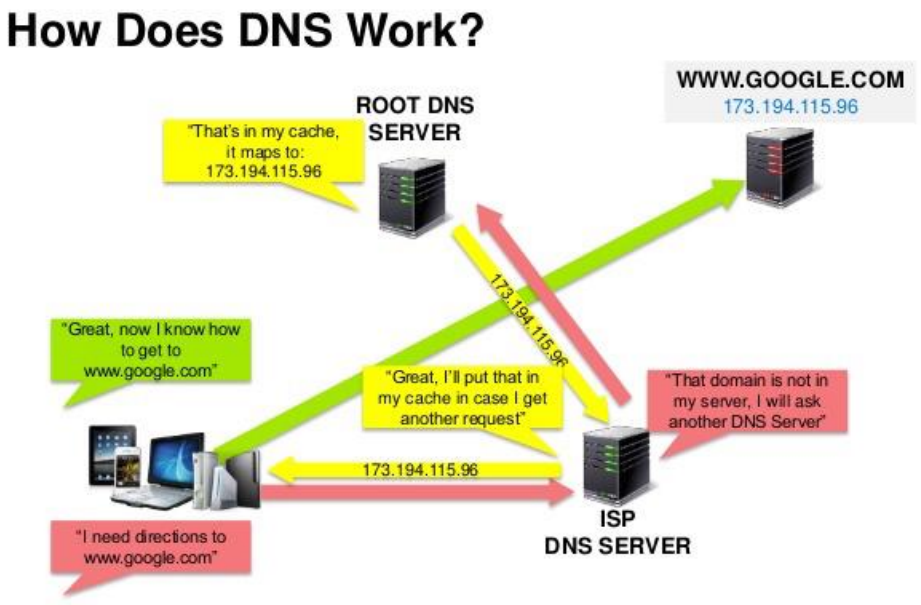


DNS lookup involves the following eight steps:

1. A client types example.com into a web browser, the query travels to the internet and is received by a DNS resolver.
2. The resolver then recursively queries a DNS root nameserver.

3. The root server responds to the resolver with the address of a Top-Level Domain (TLD).
4. The resolver then makes a request to the .com TLD.
5. The TLD server then responds with the IP address of the domain's nameserver, example.com.
6. Lastly, the recursive resolver sends a query to the domain's nameserver.
7. The IP address for example.com is then returned to the resolver from the nameserver.
8. The DNS resolver then responds to the web browser with the IP address of the domain requested initially.

Once the IP address has been resolved, the client should be able to request content from the resolved IP address. For example, the resolved IP may return a webpage to be rendered in the browser.



Server types

Now, let's look at the four key groups of servers that make up the DNS infrastructure.

DNS Resolver

A DNS resolver (also known as a DNS recursive resolver) is the first stop in a DNS query. The recursive resolver acts as a middleman between a client and a DNS nameserver. After receiving a DNS query from a web client, a recursive resolver will either respond with cached data, or send a request to a root nameserver, followed by another request to a TLD nameserver, and then one last request to an authoritative nameserver. After receiving a response from the authoritative nameserver containing the requested IP address, the recursive resolver then sends a response to the client.

DNS root server

A root server accepts a recursive resolver's query which includes a domain name, and the root nameserver responds by directing the recursive resolver to a TLD nameserver, based on the extension of that domain (.com, .net, .org, etc.). The root nameservers are overseen by a nonprofit called the Internet Corporation for Assigned Names and Numbers (ICANN).

There are 13 DNS root nameservers known to every recursive resolver. Note that while there are 13 root nameservers, that doesn't mean that there are only 13 machines in the root nameserver system. There are 13 types of root nameservers, but there are multiple copies of each one all over the world, which use Anycast routing to provide speedy responses.

TLD nameserver

A TLD nameserver maintains information for all the domain names that share a common domain extension, such as .com, .net, or whatever comes after the last dot in a URL.

Management of TLD nameservers is handled by the Internet Assigned Numbers Authority (IANA), which is a branch of ICANN. The IANA breaks up the TLD servers into two main groups:

- **Generic top-level domains**: These are domains like .com, .org, .net, .edu, and .gov.
- **Country code top-level domains**: These include any domains that are specific to a country or state. Examples include .uk, .us, .ru, and .jp.

Authoritative DNS server

The authoritative nameserver is usually the resolver's last step in the journey for an IP address. The authoritative nameserver contains information specific to the domain name it serves (e.g. google.com) and it can provide a recursive resolver with the IP address of that server found in the DNS A record, or if the domain has a CNAME record (alias) it will provide the recursive resolver with an alias domain, at which point the recursive resolver will have to perform a whole new DNS lookup to procure a record from an authoritative nameserver (often an A record containing an IP address). If it cannot find the domain, returns the NXDOMAIN message.

Query Types

There are three types of queries in a DNS system:

1. Recursive

In a recursive query, a DNS client requires that a DNS server (typically a DNS recursive resolver) will respond to the client with either the requested resource record or an error message if the resolver can't find the record.

2. Iterative

In an iterative query, a DNS client provides a hostname, and the DNS Resolver returns the best answer it can. If the DNS resolver has the relevant DNS records in its cache, it returns them. If not, it refers the DNS client to the Root Server or another Authoritative Name Server that is nearest to the required DNS zone. The DNS client must then repeat the query directly against the DNS server it was referred.

3. Non-recursive

A non-recursive query is a query in which the DNS Resolver already knows the answer. It either immediately returns a DNS record because it already stores it in a local cache, or queries a DNS Name Server which is authoritative for the record, meaning it definitely holds the correct IP for that hostname. In both cases, there is no need for additional rounds of queries (like in recursive or iterative queries). Rather, a response is immediately returned to the client.

Record Types

DNS records (aka zone files) are instructions that live in authoritative DNS servers and provide information about a domain including what IP address is associated with that domain and how to handle requests for that domain.

These records consist of a series of text files written in what is known as DNS syntax. DNS syntax is just a string of characters used as commands that tell the DNS

server what to do. All DNS records also have a "TTL", which stands for time-to-live, and indicates how often a DNS server will refresh that record.

There are more record types but for now, let's look at some of the most commonly used ones:

- **A (Address record)**: This is the record that holds the IP address of a domain.
- **AAAA (IP Version 6 Address record)**: The record that contains the IPv6 address for a domain (as opposed to A records, which stores the IPv4 address).
- **CNAME (Canonical Name record)**: Forwards one domain or subdomain to another domain, does NOT provide an IP address.
- **MX (Mail exchanger record)**: Directs mail to an email server.
- **TXT (Text Record)**: This record lets an admin store text notes in the record. These records are often used for email security.
- **NS (Name Server records)**: Stores the name server for a DNS entry.
- **SOA (Start of Authority)**: Stores admin information about a domain.
- **SRV (Service Location record)**: Specifies a port for specific services.
- **PTR (Reverse-lookup Pointer record)**: Provides a domain name in reverse lookups.
- **CERT (Certificate record)**: Stores public key certificates.

Subdomains

A subdomain is an additional part of our main domain name. It is commonly used to logically separate a website into sections. We can create multiple subdomains or child domains on the main domain.

For example, blog.example.com where blog is the subdomain, example is the primary domain and .com is the top-level domain (TLD). Similar examples can be support.example.com or careers.example.com.

DNS Zones

A DNS zone is a distinct part of the domain namespace which is delegated to a legal entity like a person, organization, or company, who is responsible for maintaining the DNS zone. A DNS zone is also an administrative function, allowing for granular control of DNS components, such as authoritative name servers.

DNS Caching

A DNS cache (sometimes called a DNS resolver cache) is a temporary database, maintained by a computer's operating system, that contains records of all the recent visits and attempted visits to websites and other internet domains. In other words, a DNS cache is just a memory of recent DNS lookups that our computer can quickly refer to when it's trying to figure out how to load a website.

The Domain Name System implements a time-to-live (TTL) on every DNS record. TTL specifies the number of seconds the record can be cached by a DNS client or server. When the record is stored in a cache, whatever TTL value came with it gets stored as well. The server continues to update the TTL of the record stored in the cache, counting down every second. When it hits zero, the record is deleted or purged from the cache. At that point, if a query for that record is received, the DNS server has to start the resolution process.

Reverse DNS

A reverse DNS lookup is a DNS query for the domain name associated with a given IP address. This accomplishes the opposite of the more commonly used

forward DNS lookup, in which the DNS system is queried to return an IP address. The process of reverse resolving an IP address uses PTR records. If the server does not have a PTR record, it cannot resolve a reverse lookup.

Reverse lookups are commonly used by email servers. Email servers check and see if an email message came from a valid server before bringing it onto their network. Many email servers will reject messages from any server that does not support reverse lookups or from a server that is highly unlikely to be legitimate.

Note: Reverse DNS lookups are not universally adopted as they are not critical to the normal function of the internet.

Examples

These are some widely used managed DNS solutions:

- Route53
- Cloudflare DNS
- Google Cloud DNS
- Azure DNS
- NS1

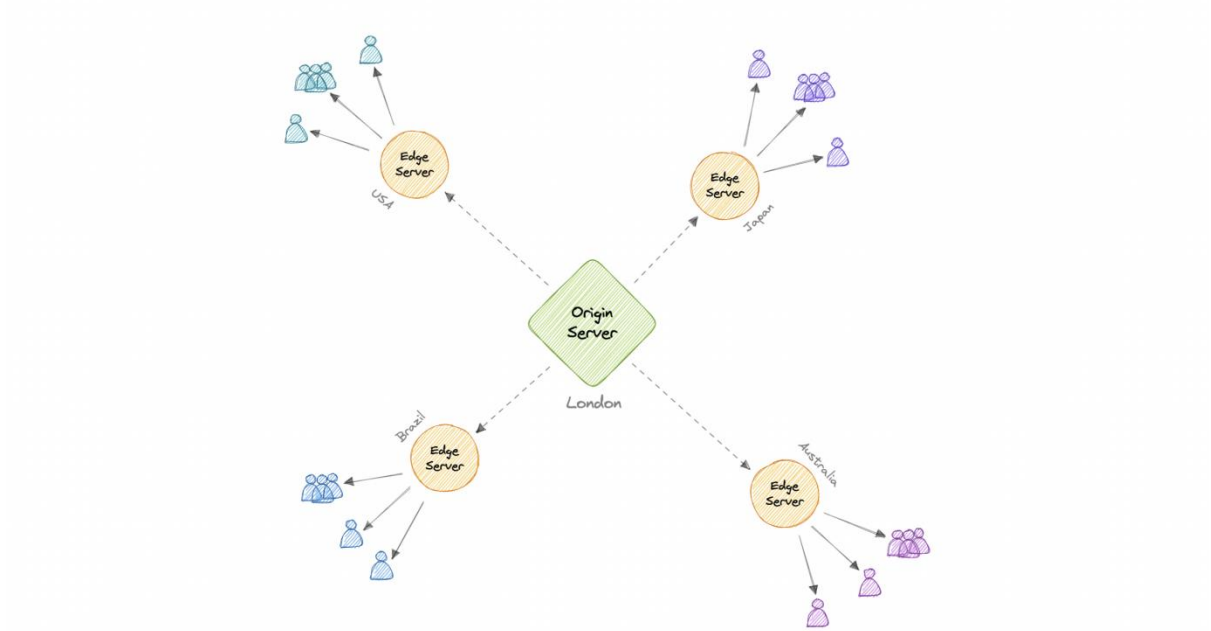
Content Delivery Network (CDN)

A content delivery network (CDN) is a geographically distributed group of servers that work together to provide fast delivery of internet content. Generally, static files such as HTML/CSS/JS, photos, and videos are served from CDN.

Why use a CDN?

Content Delivery Network (CDN) increases content availability and redundancy while reducing bandwidth costs and improving security. Serving content from CDNs can significantly improve performance as users receive content from data centers close to them and our servers do not have to serve requests that the CDN fulfills.

How does a CDN work?



In a CDN, the origin server contains the original versions of the content while the edge servers are numerous and distributed across various locations around the world.

To minimize the distance between the visitors and the website's server, a CDN stores a cached version of its content in multiple geographical locations known as edge locations. Each edge location contains several caching servers responsible for content delivery to visitors within its proximity.

Once the static assets are cached on all the CDN servers for a particular location, all subsequent website visitor requests for static assets will be delivered from these edge servers instead of the origin, thus reducing the origin load and improving scalability.

For example, when someone in the UK requests our website which might be hosted in the USA, they will be served from the closest edge location such as the London edge location. This is much quicker than having the visitor make a complete request to the origin server which will increase the latency.

CDN Types

CDNs are generally divided into two types:

1. Push CDNs

Push CDNs receive new content whenever changes occur on the server. We take full responsibility for providing content, uploading directly to the CDN, and rewriting URLs to point to the CDN. We can configure when content expires and when it is updated. Content is uploaded only when it is new or changed, minimizing traffic, but maximizing storage.

Sites with a small amount of traffic or sites with content that isn't often updated work well with push CDNs. Content is placed on the CDNs once, instead of being re-pulled at regular intervals.

2. Pull CDNs

In a Pull CDN situation, the cache is updated based on request. When the client sends a request that requires static assets to be fetched from the CDN if the CDN doesn't have it, then it will fetch the newly updated assets from the origin server and populate its cache with this new asset, and then send this new cached asset to the user.

Contrary to the Push CDN, this requires less maintenance because cache updates on CDN nodes are performed based on requests from the client to the origin server. Sites with heavy traffic work well with pull CDNs, as traffic is spread out more evenly with only recently-requested content remaining on the CDN.

Disadvantages

As we all know good things come with extra costs, so let's discuss some disadvantages of CDNs:

- **Extra charges**: It can be expensive to use a CDN, especially for high-traffic services.

- **Restrictions**: Some organizations and countries have blocked the domains or IP addresses of popular CDNs.
- **Location**: If most of our audience is located in a country where the CDN has no servers, the data on our website may have to travel further than without using any CDN.

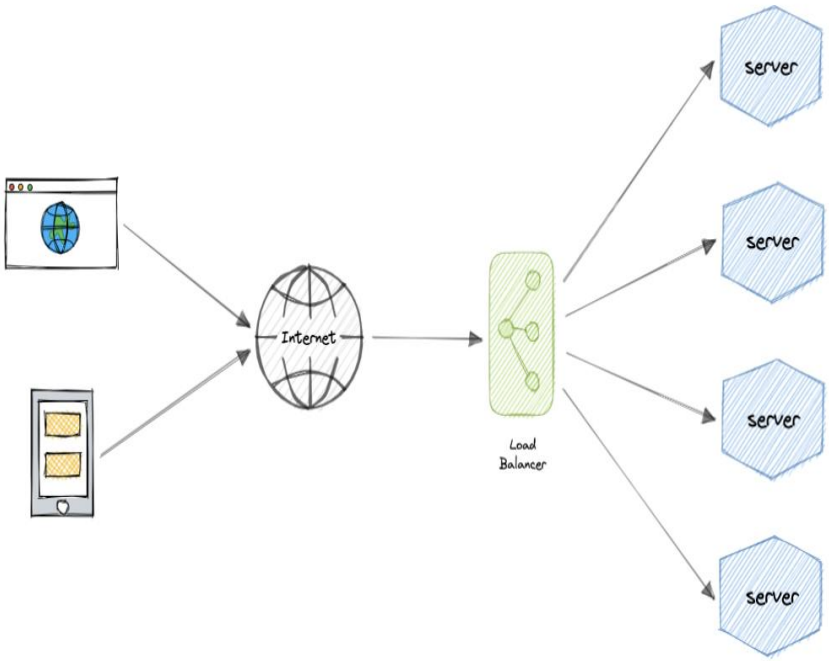
Examples

Here are some widely used CDNs:

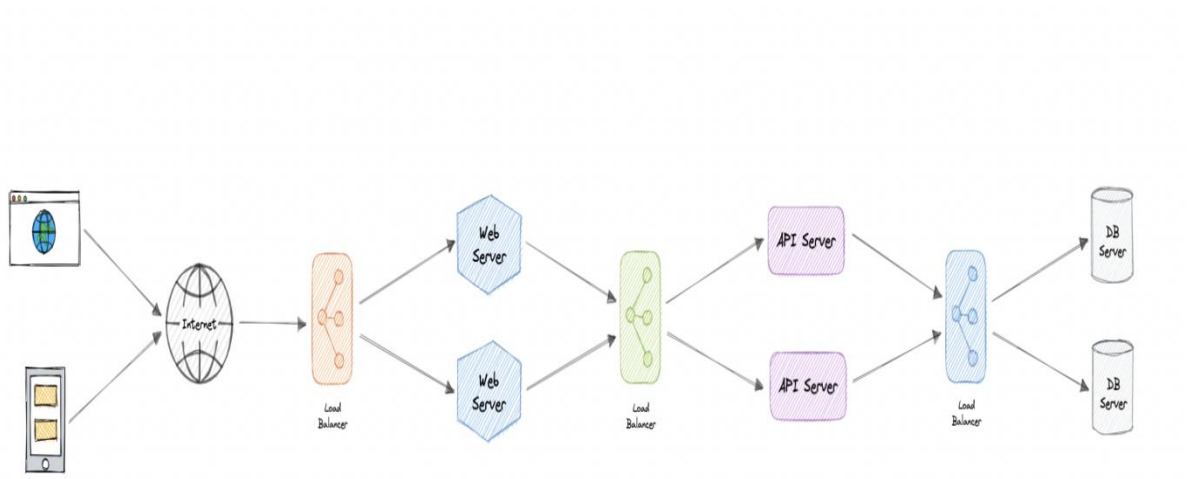
- Amazon CloudFront
- Google Cloud CDN
- Cloudflare CDN
- Fastly

Load Balancing

Load balancing lets us distribute incoming network traffic across multiple resources ensuring high availability and reliability by sending requests only to resources that are online. This provides the flexibility to add or subtract resources as demand dictates.



For additional scalability and redundancy, we can try to load balance at each layer of our system:



But why?

Modern high-traffic websites must serve hundreds of thousands, if not millions, of concurrent requests from users or clients. To cost-effectively scale to meet these high volumes, modern computing best practice generally requires adding more servers.

A load balancer can sit in front of the servers and route client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization. This ensures that no single server is overworked, which could degrade performance. If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts sending requests to it.

Workload distribution

This is the core functionality provided by a load balancer and has several common variations:

- **Host-based**: Distributes requests based on the requested hostname.

- **Path-based**: Using the entire URL to distribute requests as opposed to just the hostname.
- **Content-based**: Inspects the message content of a request. This allows distribution based on content such as the value of a parameter.

Layers

Generally speaking, load balancers operate at one of the two levels:

1. Network layer

This is the load balancer that works at the network's transport layer, also known as layer 4. This performs routing based on networking information such as IP addresses and is not able to perform content-based routing. These are often dedicated hardware devices that can operate at high speed.

2. Application layer

This is the load balancer that operates at the application layer, also known as layer 7. Load balancers can read requests in their entirety and perform content-based routing. This allows the management of load based on a full understanding of traffic.

Load Types

Let's look at different types of load balancers:

1. Software

Software load balancers usually are easier to deploy than hardware versions. They also tend to be more cost-effective and flexible, and they are used in conjunction with software development environments. The software approach gives us the flexibility of configuring the load balancer to our environment's specific needs. The boost in flexibility may come at the cost of having to do more work to set up the

load balancer. Compared to hardware versions, which offer more of a closed-box approach, software balancers give us more freedom to make changes and upgrades.

Software load balancers are widely used and are available either as installable solutions that require configuration and management or as a managed cloud service.

2. Hardware

As the name implies, a hardware load balancer relies on physical, on-premises hardware to distribute application and network traffic. These devices can handle a large volume of traffic but often carry a hefty price tag and are fairly limited in terms of flexibility.

Hardware load balancers include proprietary firmware that requires maintenance and updates as new versions, and security patches are released.

3. DNS

DNS load balancing is the practice of configuring a domain in the Domain Name System (DNS) such that client requests to the domain are distributed across a group of server machines.

Unfortunately, DNS load balancing has inherent problems limiting its reliability and efficiency. Most significantly, DNS does not check for server and network outages, or errors. It always returns the same set of IP addresses for a domain even if servers are down or inaccessible.

Routing Algorithms

Now, let's discuss commonly used routing algorithms:

- **Round-robin**: Requests are distributed to application servers in rotation.
- **Weighted Round-robin**: Builds on the simple Round-robin technique to account for differing server characteristics such as compute and traffic

handling capacity using weights that can be assigned via DNS records by the administrator.

- **Least Connections**: A new request is sent to the server with the fewest current connections to clients. The relative computing capacity of each server is factored into determining which one has the least connections.
- **Least Response Time**: Sends requests to the server selected by a formula that combines the fastest response time and fewest active connections.
- **Least Bandwidth**: This method measures traffic in megabits per second (Mbps), sending client requests to the server with the least Mbps of traffic.
- **Hashing**: Distributes requests based on a key we define, such as the client IP address or the request URL.

Advantages

Load balancing also plays a key role in preventing downtime, other advantages of load balancing include the following:

- Scalability
- Redundancy
- Flexibility
- Efficiency

Redundant load balancers

As you must've already guessed, the load balancer itself can be a single point of failure. To overcome this, a second or N number of load balancers can be used in a cluster mode.

And, if there's failure detection and the active load balancer fails, another passive load balancer can take over which will make our system more fault-tolerant.

Proxy

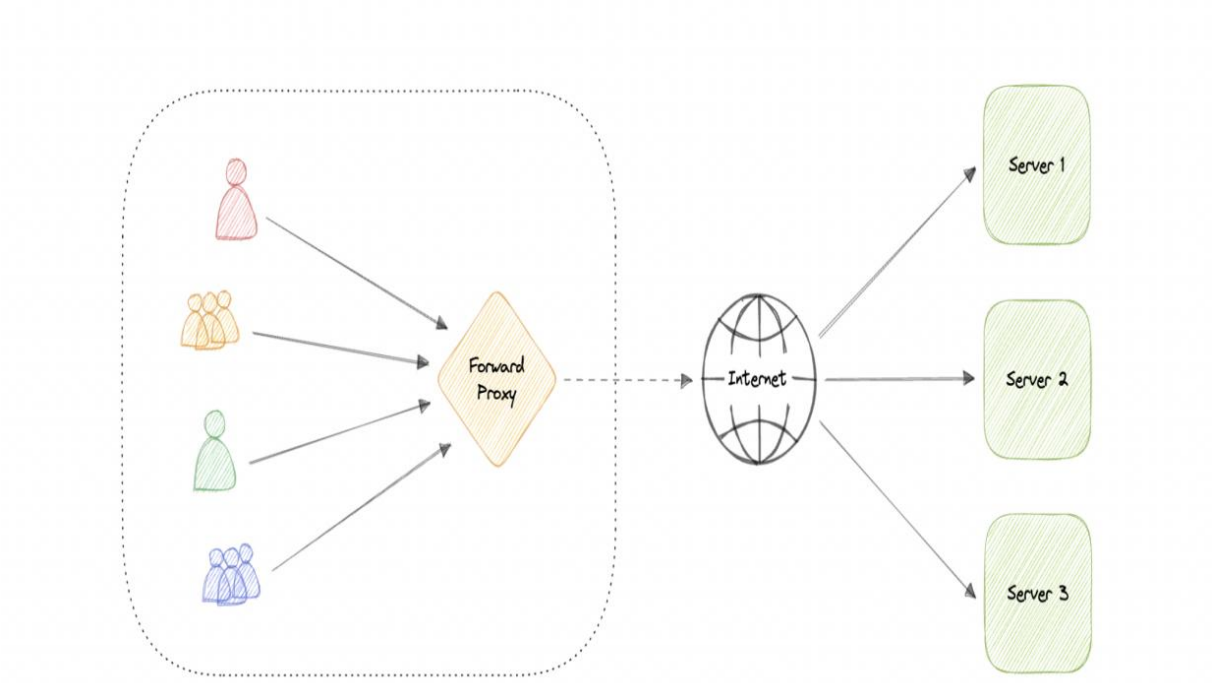
A proxy server is an intermediary piece of hardware/software sitting between the client and the backend server. It receives requests from clients and relays them to the origin servers. Typically, proxies are used to filter requests, log requests, or sometimes transform requests (by adding/removing headers, encrypting/decrypting, or compression).

Types of proxies

There are two types of proxies:

1. Forward Proxy

A forward proxy, often called a proxy, proxy server, or web proxy is a server that sits in front of a group of client machines. When those computers make requests to sites and services on the internet, the proxy server intercepts those requests and then communicates with web servers on behalf of those clients, like a middleman.



Advantages of forward Proxy

Here are some advantages of a forward proxy:

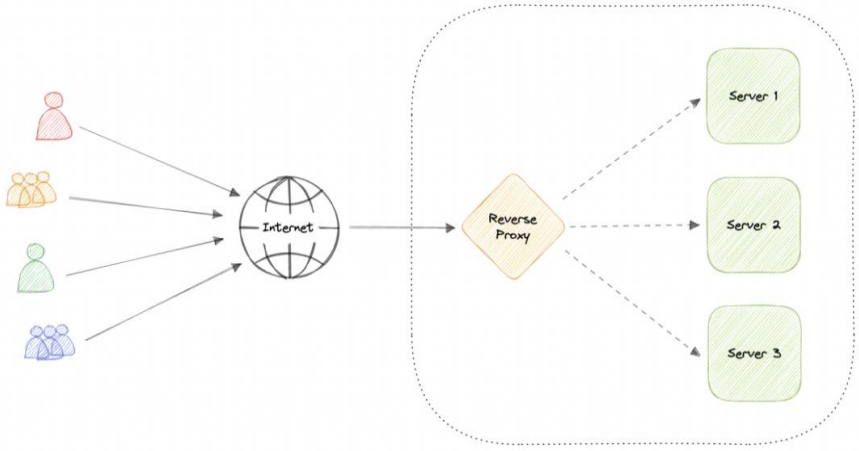
- Block access to certain content
- Allows access to geo-restricted content
- Provides anonymity
- Avoid other browsing restrictions

Although proxies provide the benefits of anonymity, they can still track our personal information. Setup and maintenance of a proxy server can be costly and requires configurations.

2. Reverse Proxy

A reverse proxy is a server that sits in front of one or more web servers, intercepting requests from clients. When clients send requests to the origin server of a website, those requests are intercepted by the reverse proxy server.

The difference between a forward and reverse proxy is subtle but important. A simplified way to sum it up would be to say that a forward proxy sits in front of a client and ensures that no origin server ever communicates directly with that specific client. On the other hand, a reverse proxy sits in front of an origin server and ensures that no client ever communicates directly with that origin server.



Introducing reverse proxy results in increased complexity. A single reverse proxy is a single point of failure, configuring multiple reverse proxies (i.e. a failover) further increases complexity.

Advantages

Here are some advantages of using a reverse proxy:

- Improved security
- Caching
- SSL encryption
- Load balancing
- Scalability and flexibility

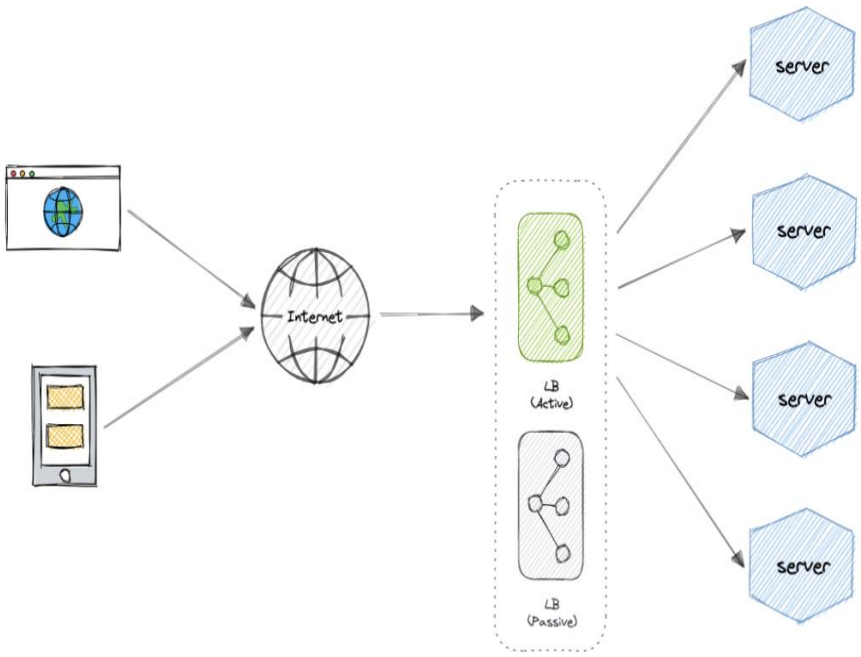
Load balancer vs Reverse Proxy

Wait, isn't reverse proxy similar to a load balancer? Well, no as a load balancer is useful when we have multiple servers. Often, load balancers route traffic to a set of servers serving the same function, while reverse proxies can be useful even with just one web server or application server. A reverse proxy can also act as a load balancer but not the other way around.

Examples

Below are some commonly used proxy technologies:

- Nginx
- HAProxy
- Traefik
- Envoy



Features of load balancers:

Here are some commonly desired features of load balancers:

- **Autoscaling**: Starting up and shutting down resources in response to demand conditions.
- **Sticky sessions**: The ability to assign the same user or device to the same resource in order to maintain the session state on the resource.
- **Healthchecks**: The ability to determine if a resource is down or performing poorly in order to remove the resource from the load balancing pool.
- **Persistence connections**: Allowing a server to open a persistent connection with a client such as a WebSocket.
- **Encryption**: Handling encrypted connections such as TLS and SSL.
- **Certificates**: Presenting certificates to a client and authentication of client certificates.
- **Compression**: Compression of responses.

- **Caching**: An application-layer load balancer may offer the ability to cache responses.
- **Logging**: Logging of request and response metadata can serve as an important audit trail or source for analytics data.
- **Request tracing**: Assigning each request a unique id for the purposes of logging, monitoring, and troubleshooting.
- **Redirects**: The ability to redirect an incoming request based on factors such as the requested path.
- **Fixed response**: Returning a static response for a request such as an error message.

Examples of load balancing

Following are some of the load balancing solutions commonly used in the industry:

- Amazon Elastic Load Balancing
- Azure Load Balancing
- GCP Load Balancing
- DigitalOcean Load Balancer
- Nginx
- HAProxy