



Let's start with a question: If millions of users worldwide use our data-intensive applications, and our service is deployed in a single data center to serve the users' requests, what possible problems can arise?

The following problems can arise:

- **High latency:** The user-perceived latency will be high due to the physical distance from the serving data center. User-perceived latency has many components, such as transmission delays (a function of available bandwidth), propagation delays (a function of distance), queuing delays (a function of network congestion), and nodal processing delays. Therefore, data transmission over a large distance results in higher latency. Real-time applications require a latency below 200 milliseconds (ms) in general. For the Voice over Internet Protocol (VoIP), latency should not be more than 150 ms, whereas video streaming applications cannot tolerate a latency above a few seconds.
- **Data-intensive applications:** Data-intensive applications require transferring large traffic. Over a longer distance, this could be a problem due to the network path stretching through different kinds of ISPs. Because of some smaller Path message transmission unit (MTU) links, the throughput of applications on the network might be reduced. Similarly, different portions of the network path might have different congestion characteristics. The problem multiplies as the number of users grows because the origin servers will have to provide the data individually to each user. That is, the primary data center will need to send out a lot of redundant data when multiple clients ask for it. However, applications that use streaming services are both data-intensive and dynamic in nature.

- **Scarcity of data center resources:** Important data center resources like computational capacity and bandwidth become a limitation when the number of users of a service increases significantly. Services engaging millions of users simultaneously need scaling. Even if scaling is achieved in a single data center, it can still suffer from becoming a single point of failure when the data center goes offline due to natural calamity or connectivity issues with the Internet.

→ solution to all the problem is CDN.

CDN is a group of geographically distributed proxy servers. A proxy server is an intermediate server between client & origin server.

We can bring data close to the user by placing a small data center near the user and storing copies of the data there.  
CDN mainly stores two types of data: static and dynamic

Let's look at the different ways CDN solves the discussed problems:

- **High latency:** CDN brings the content closer to end users. Therefore, it reduces the physical distance and latency.
- **Data-intensive applications:** Since the path to the data includes only the ISP and the nearby CDN components, there's no issue in serving a large number of users through a few CDN components in a specific area. As shown below, the origin data center will have to provide the data to local CDN components only once, whereas local CDN components can provide data to different users individually. No user will have to download their own copy of data from the origin servers.

**Note:** Various streaming protocols are used to deliver dynamic content by the CDN providers. For example, CDNs use the Real-time Messaging Protocol (RTMP), HTTP Live Streaming (HLS), Real-time Streaming Protocol (RTSP), and many more to deliver dynamic content.

- **Scarcity of data center resources:** A CDN is used to serve popular content. Due to this reason, most of the traffic is handled at the CDN instead of the origin servers. So, different local or distributed CDN components share the load on origin servers.



Note: A few well-known CDN providers are Akamai, StackPath, Cloudflare, Rackspace, Amazon CloudFront, and Google Cloud CDN.



Does a CDN cache all content from the origin server?

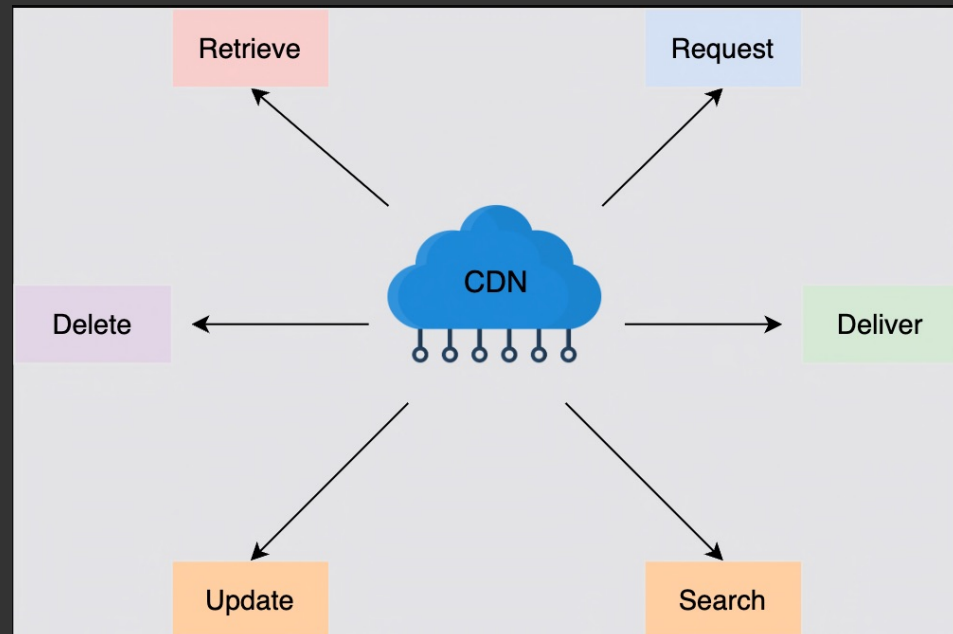
Hide Answer ^

Not likely. A CDN caches a considerable portion of the content depending on its capabilities, and it mostly caches the static content.

It also depends on the size of the content. For example, it might be possible for Netflix to store more than 90% of its movies in the CDN, while this might not be feasible for a service like YouTube due to the humongous volume of content.

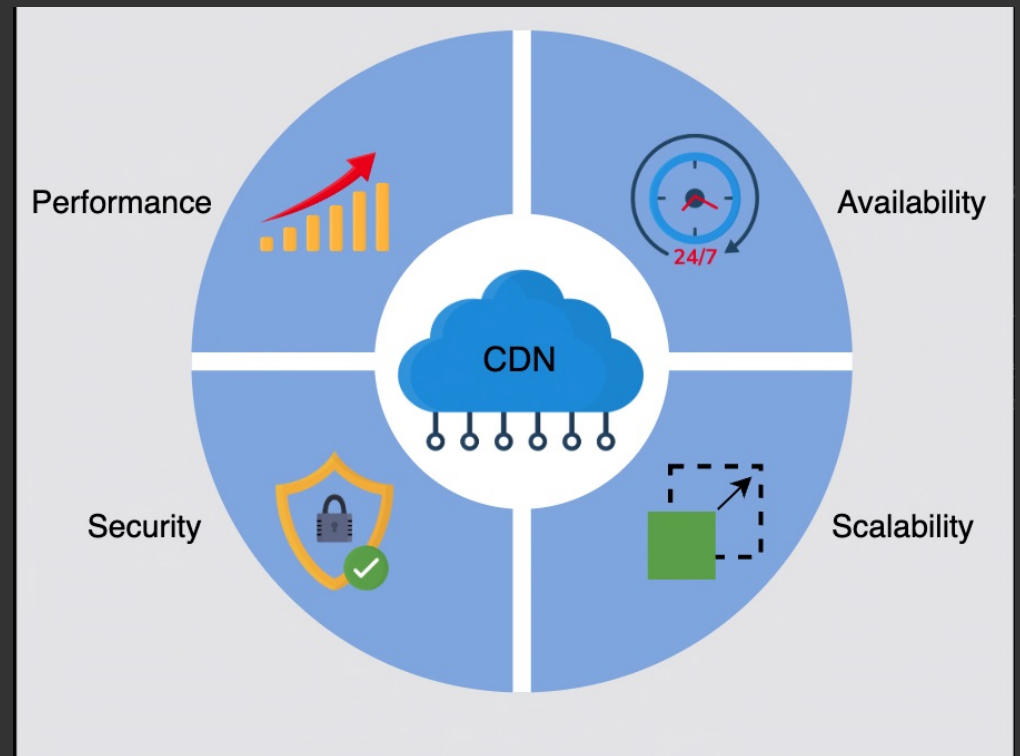
Requirements

Functional Requirements

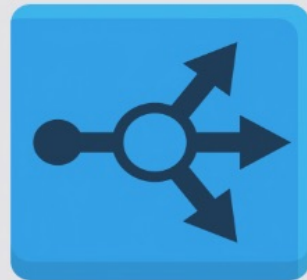


## Non functional

- Performance → minimum latency
- Availability → high
- Scalability → horizontally scale
- Reliability & security → NSPOF ✓



DNS



Load  
balancer

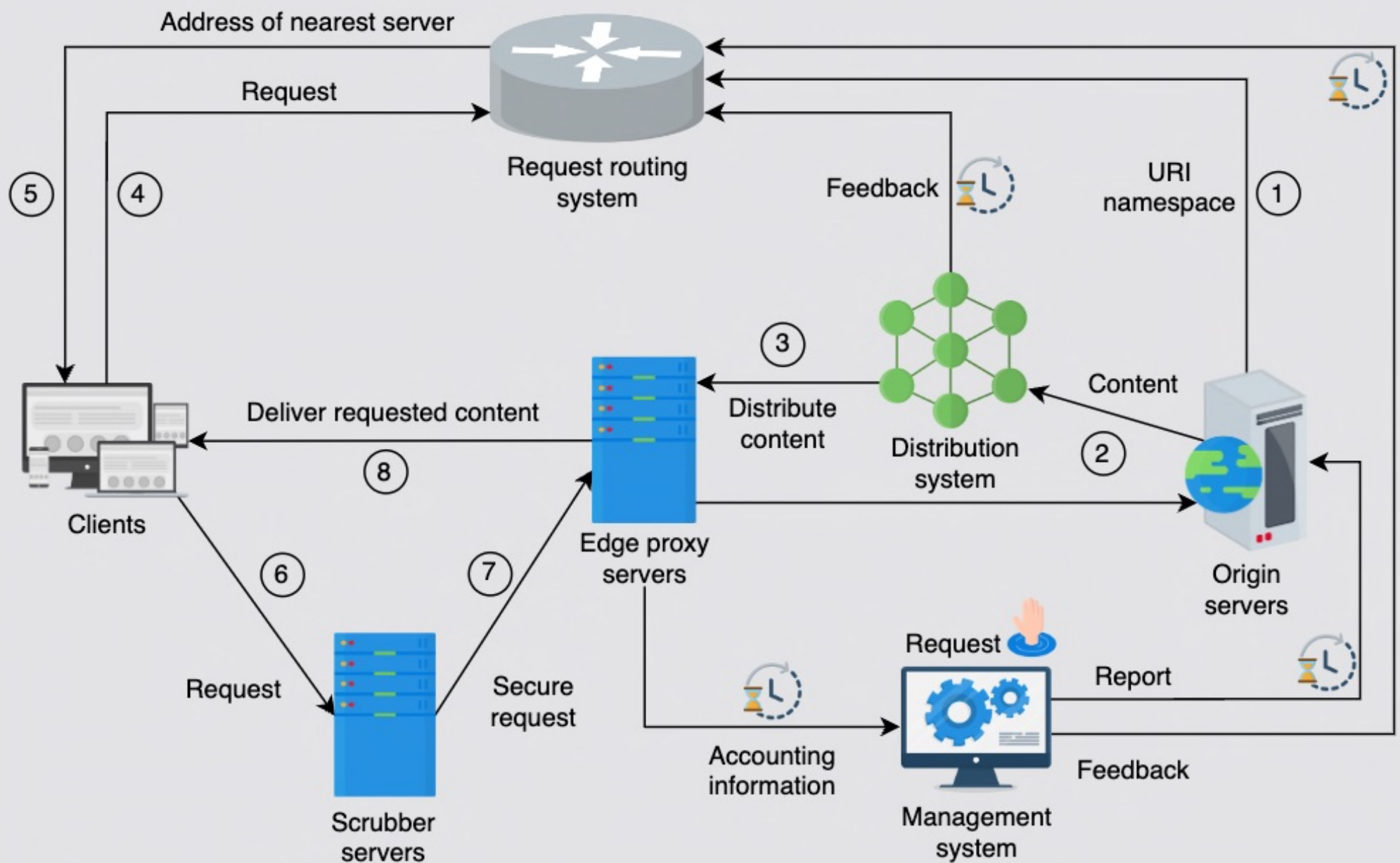
## Building blocks of CDN


- DNS
- Load balancer.


## Components

- **Clients:** End users use various clients, like browsers, smartphones, and other devices, to request content from the CDN.
- **Routing system:** The routing system directs clients to the nearest CDN facility. To do that effectively, this component receives input from various systems to understand where content is placed, how many requests are made for particular content, the load a particular set of servers is handling, and the URI (Uniform Resource Identifier) namespace of various contents. In the next lesson, we'll discuss different routing mechanisms to forward users to the nearest CDN facility.
- **Scrubber servers:** Scrubber servers are used to separate the good traffic from malicious traffic and protect against well-known attacks, like DDoS. Scrubber servers are generally used only when an attack is detected. In that case, the traffic is scrubbed or cleaned and then routed to the target destination.
- **Proxy servers:** The proxy or edge proxy servers serve the content from RAM to the users. Proxy servers store hot data in RAM, though they can store cold data in SSD or hard drive as well. These servers also provide accounting information and receive content from the distribution system.
- **Distribution system:** The distribution system is responsible for distributing content to all the edge proxy servers to different CDN facilities. This system uses the Internet and intelligent broadcast-like approaches to distribute content across the active edge proxy servers.
- **Origin servers:** The CDN infrastructure facilitates users with data received from the origin servers. The origin servers serve any unavailable data at the CDN to clients. Origin servers will use appropriate stores to keep content and other mapping metadata. Though, we won't discuss the internal architecture of origin infrastructure here.
- **Management system:** The management systems are important in CDNs from a business and managerial aspect where resource usage and statistics are constantly observed. This component measures important metrics, like latency, downtime, packet loss, server load, and so on. For third-party CDNs, accounting information can also be used for billing purposes.





 represents periodic activity

 represents operation when content is not in the proxy server and content needs to be fetched from the origin servers.

# Workflow

The workflow for the abstract design is given below:

1. The origin servers provide the URI namespace delegation of all objects cached in the CDN to the request routing system.
2. The origin server publishes the content to the distribution system responsible for data distribution across the active edge proxy servers.
3. The distribution system distributes the content among the proxy servers and provides feedback to the request routing system. This feedback is helpful in optimizing the selection of the nearest proxy server for a requesting client. This feedback contains information about which content is cached on which proxy server to route traffic to relevant proxy servers.
4. The client requests the routing system for a suitable proxy server from the request routing system.
5. The request routing system returns the IP address of an appropriate proxy server.
6. The client request routes through the scrubber servers for security reasons.
7. The scrubber server forwards good traffic to the edge proxy server.
8. The edge proxy server serves the client request and periodically forwards accounting information to the management system. The management system updates the origin servers and sends feedback to the routing system about the statistics and detail of the content. However, the request is routed to the origin servers if the content isn't available in the proxy servers. It's also possible to have a hierarchy of proxy servers if the content isn't found in the edge proxy servers. For such cases, the request gets forwarded to the parent proxy servers.

# API Design

## 1. Retrieve Content

```
retrieveContent(proxyserver_id, content_type, content_version, description)
```

Parameter	Description
<code>proxyserver_id</code>	This is a unique ID of the requesting proxy server.
<code>content_type</code>	This data structure will contain information about the requested content. Specifically, it will contain the category (audio, video, document, script, and so on), the type of clients it's requested for, and the requested quality (if any).
<code>content_version</code>	This represents the version number of the content. For the <code>/retrieveContent</code> API, the <code>content_version</code> will contain the current version of the content residing in the proxy server. The <code>content_version</code> will be <code>NULL</code> if no previous version is available at the proxy server.
<code>description</code>	This specifies the content detail—for example, the video's extension, resolution detail, and so on if the <code>content_type</code> is video.

## 2. Deliver (origin server to proxy server)

```
deliverContent(origin_id, server_list, content_type, content_version, description)
```

Parameter	Description
<code>origin_id</code>	This recognizes each origin server uniquely.
<code>server_list</code>	This identifies the list of servers the content will be pushed to by the distribution system.
<code>content_version</code>	This represents the updated version of the content at the origin server. The proxy server receiving the content will discard the previous version.

### 3. Request (clients & proxy servers)

```
requestContent(user_id, content_type, description)
```

Parameter	Description
<code>user_id</code>	This is the unique ID of the user who requested the content.

### 4. Search (proxy servers & peer proxy servers)

```
searchContent(proxyserver_id, content_type, description)
```

### 5. Update (Proxy server to peer proxy server)

```
updateContent(proxyserver_id, content_type, description)
```

Parameter	Description
<code>proxyserver_id</code>	This recognizes the proxy server uniquely in the PoP to update the content.

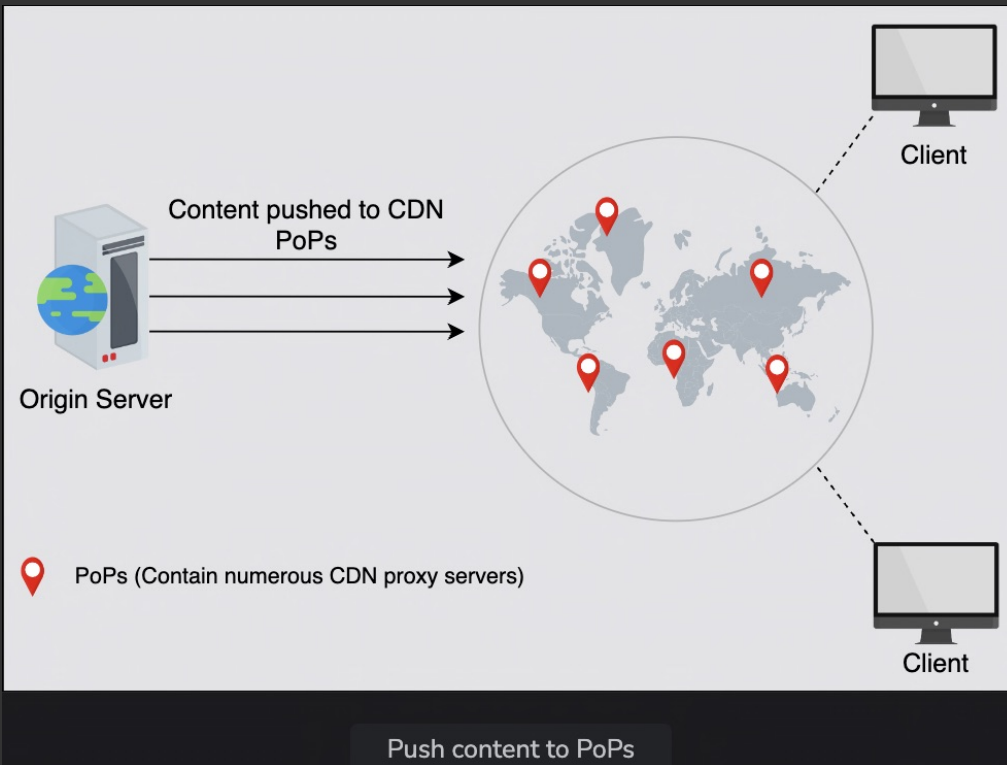


# In-depth Investigation Part - 1

## Content caching strategies in CDN

### Push CDN

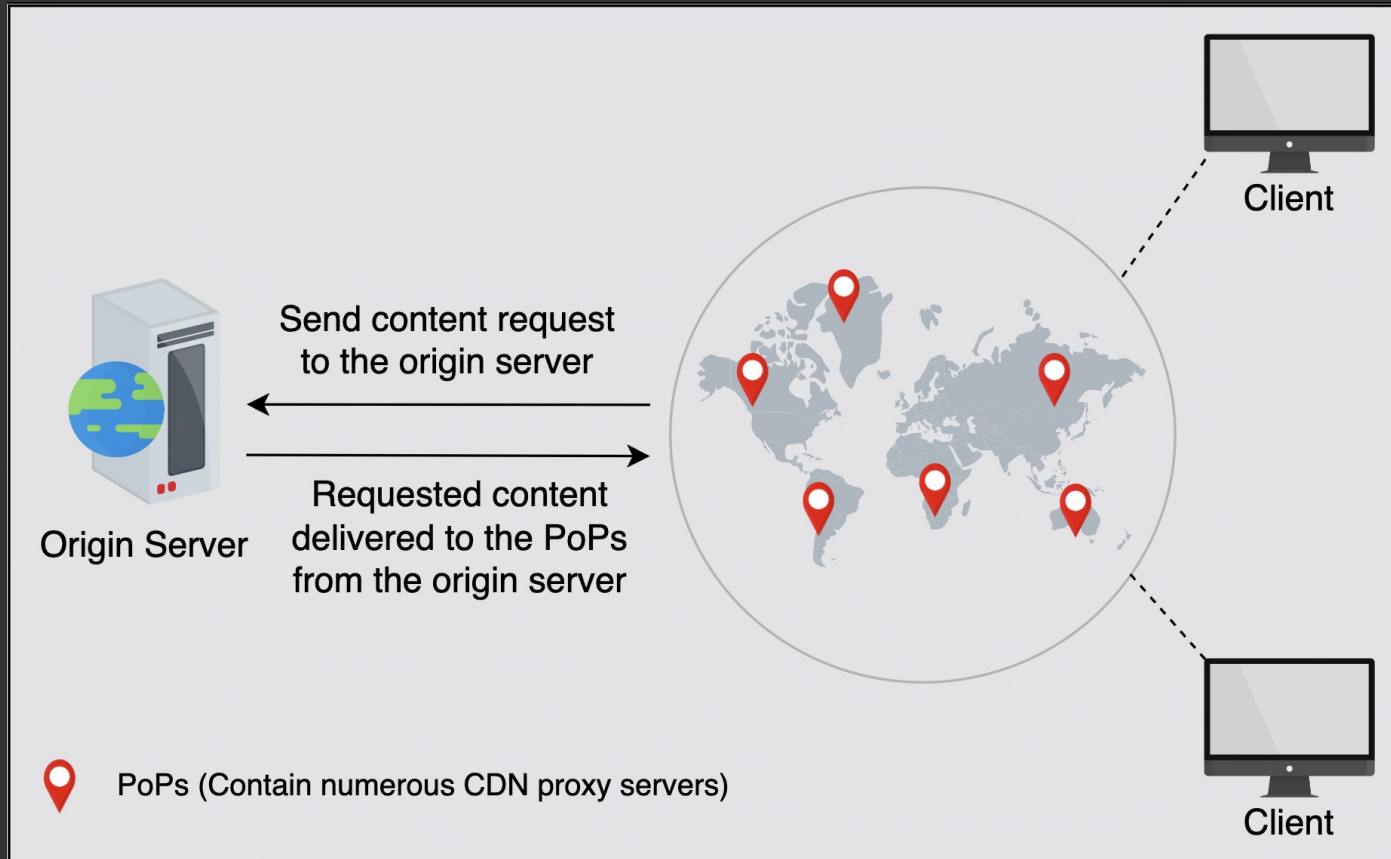
Content gets sent automatically to the CDN proxy servers from the origin server in the push CDN model. The content delivery to the CDN proxy servers is the content provider's responsibility. Push CDN is appropriate for static content delivery, where the origin server decides which content to deliver to users using the CDN. The content is pushed to proxy servers in various locations according to the content's popularity. If the content is rapidly changing, the push model might struggle to keep up and will do redundant content pushes.



## Pull CDN

A CDN pulls the unavailable data from origin servers when requested by a user. The proxy servers keep the files for a specified amount of time and then remove them from the cache if they're no longer requested to balance capacity and cost.

When users request web content in the pull CDN model, the CDN itself is responsible for pulling the requested content from the origin server and serving it to the users. Therefore, this type of CDN is more suited for serving dynamic content.



### Dynamic content caching optimization

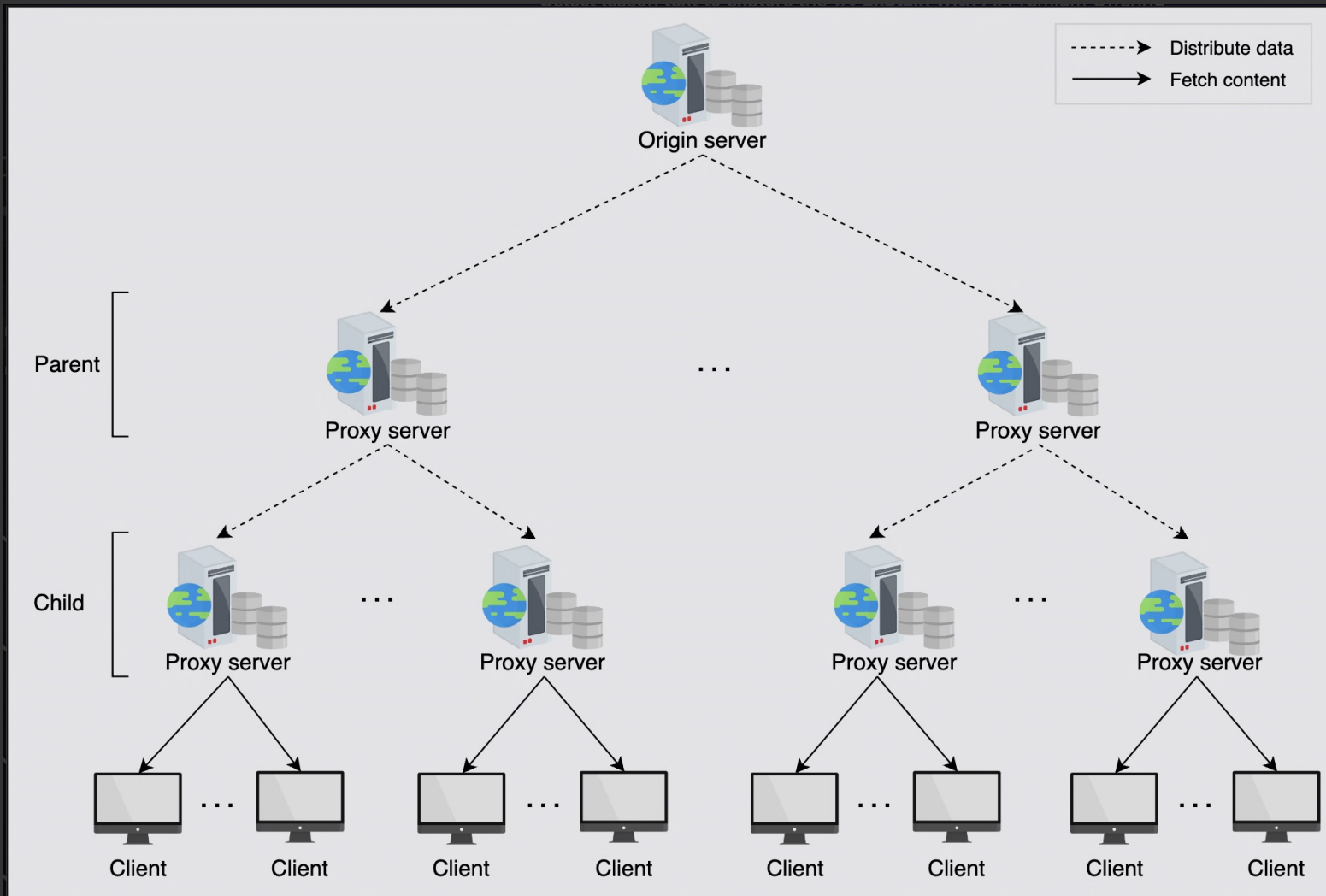
Since dynamic content often changes, it's a good idea to cache it optimally. This section deals with the optimization of frequently changing content.

Certain dynamic content creation requires the execution of scripts that can be executed at proxy servers instead of running on the origin server. Dynamic data can be generated using various parameters, which can be beneficial if executed at the proxy servers.

Another popular approach for dynamic data compression is Edge Side Includes (ESI) markup language. Usually, a small portion of the web pages changes in a certain time. It means fetching a full web page on each small change contains a lot of redundant data.

## Multi-tier CDN architecture

The content provider sends the content to a large number of clients through a CDN. The task of distributing data to all the CDN proxy servers simultaneously is challenging and burdens the origin server significantly. CDNs follow a tree-like structure to ease the data distribution process for the origin server.



What happens if a child or parent proxy server fails or if the origin server fails?

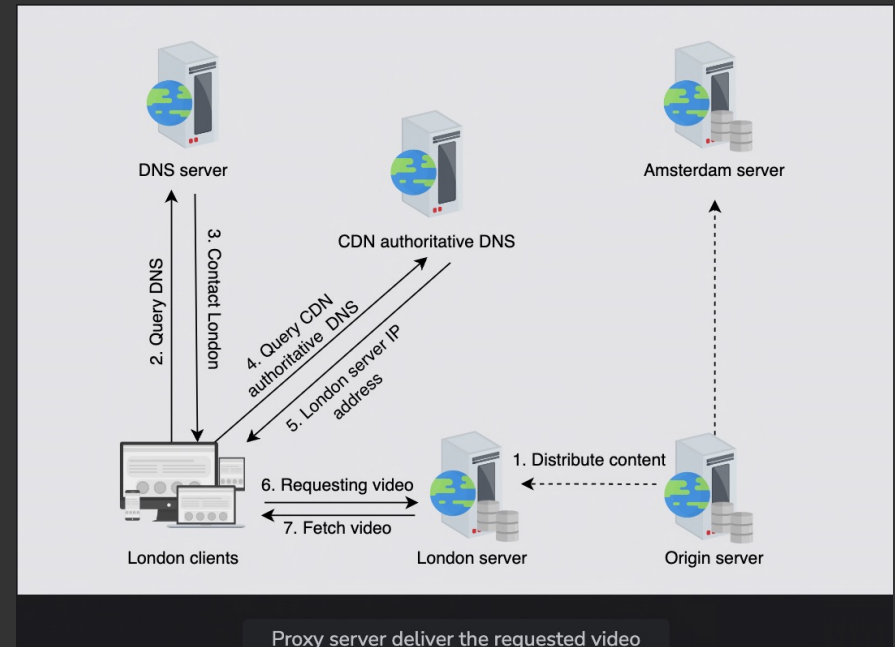
Each PoP contains a collection of CDN proxy servers. When any child proxy server stops working due to any failures, DNS can route clients to a different child-level proxy server. Each child proxy server knows many upper-layer parent servers, and if one fails, it can go to the other one. The origin server is a set of servers with a hot backup(s), and content is in replicated store. If any of the origin servers fail, other servers take the load.

Find the nearest proxy server to fetch the data.

Network distance between the user and the proxy server is crucial. This is a function of the following two things:  
The first is the length of the network path.  
The second is the capacity (bandwidth) limits along the network path.

Requests load refers to the load a proxy server handles at any point in time. If a set of proxy servers are overloaded, the request routing system should forward the request to a location with a lesser load. This action balances out the proxy server load and, consequently, reduces the response latency.

→ DNS Redirection





## Anycast

Anycast is a routing methodology in which all the edge servers located in multiple locations share the same single IP address. It employs the Border Gateway Protocol (BGP) to route clients based on the Internet's natural network flow. A CDN provider can use the anycast mechanism so that clients are directed to the nearest proxy servers for content.

## Client Multiplexing

Client multiplexing involves sending a client a list of candidate servers. The client then chooses one server from the list to send the request to. This approach is inefficient because the client lacks the overall information to choose the most suitable server for their request.

## HTTP Redirection

HTTP redirection is the simplest of all approaches. With this scheme, the client requests content from the origin server. The origin server responds with an HTTP protocol to redirect the user via a URL of the content.

# Consistency in CDN

## Periodic polling

Using the pull model, proxy servers request the origin server periodically for updated data and change the content in the cache accordingly. When content changes infrequently, the polling approach consumes unnecessary bandwidth. Periodic polling uses time-to-refresh (TTR) to adjust the time period for requesting updated data from the origin servers.

## Time-to-live (TTL)

Because of the TTR, the proxy servers may uselessly request the origin servers for updated data. A better approach that could be employed to reduce the frequency of refresh messages is the time-to-live (TTL) approach. In this approach, each object has a TTL attribute assigned to it by the origin server. The TTL defines the expiration time of the content. The proxy servers serve the same data version to the users until that content expires. Upon expiration, the proxy server checks for an update with the origin server. If the data is changed, it gets the updated data from the origin server and then responds to the user's requests with the updated data. Otherwise, it keeps the same data with an updated expiration time from the origin servers.

## Leases

The origin server grants a lease to the data sent to a proxy server using this technique. The lease denotes the time interval for which the origin server agrees to notify the proxy server if there's any change in the data. The proxy server must send a message requesting a lease renewal after the expiration of the lease. The lease method helps to reduce the number of messages exchanged between the proxy and origin server.

# Placement of Proxy Servers

## Placement of CDN proxy servers

The CDN proxy servers must be placed at network locations with good connectivity. See the options below:

On-premises represents a smaller data center that could be placed near major IXPs.

Off-premises represents placing CDN proxy servers in ISP's networks.

What benefits could an ISP get by placing the CDN proxy servers inside their network?

A CDN node cuts down the amount of external bandwidth that ISPs need and pay for. The SLA defined by ISPs with the proxy server providers may benefit the ISPs economically.

A CDN node improves responsiveness for the ISP's customers, which makes the ISP's customers happier. This gives them a competitive advantage over ISPs that don't have a CDN node.

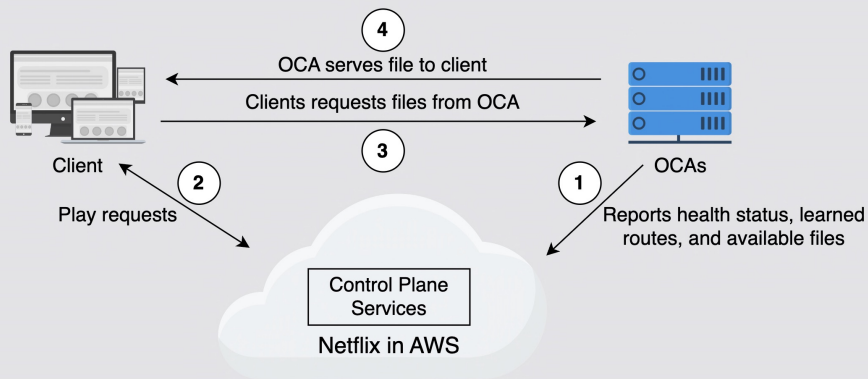
Bringing content closer to the users reduces data on the Internet core.

#### CDN as a service

Most companies don't build their own CDN. Instead, they use the services of a CDN provider, such as Akamai, Cloudflare, Fastly, and so on, to deliver their content. Similarly, players like AWS make it possible for anyone to use a global CDN facility.

#### Specialized CDN

We've discussed that many companies use CDN as a service, but there are cases where companies build their own CDN. A number of reasons factor into this decision. One is the cost of a commercial CDN service. A specialized CDN consists of points of presence (PoPs) that only serve content for their own company. These PoPs can be caching servers, reverse proxies, or application delivery controllers. Although a specialized CDN has high costs at its first setup, the costs eventually decrease with time. In essence, it's a buy versus build decision.



Why Netflix built its CDN

As Netflix became more popular, it decided to build and manage its own CDN for the following reasons:

The CDN service providers were scuffling to expand their infrastructure due to the rapid growth in customer demand for video streaming on Netflix.

With the increasing volume of streaming videos, the expense of using CDN services increased.

Video streaming is the main business and a primary revenue source for Netflix. So, protecting the data of all the videos on the platform is critical. Netflix's OCA manages potential data leakage risks in a better way.

To provide optimal streaming media delivery to customers, Netflix needed to maximize its control over the user's video player, the network between the user, and the Netflix servers.

Netflix's OCA can use custom HTTP modules and TCP connection algorithms to detect network problems quickly and troubleshoot any issues in their CDN network.

Netflix wanted to keep popular content for a long time. This wasn't entirely possible while operating with a public CDN due to the high costs that would be incurred to keep and maintain it.





### 3. Scalability

It brings content closer to the user and removes the requirement of high bandwidth, thereby ensuring scalability.

Horizontal scalability is possible by adding the number of reading replicas in the form of edge proxy servers.

The limitations with horizontal scalability and storage capacity of an individual proxy server can be dealt with using the layered architecture of the proxy servers we described above.

### 4. Reliability & Security

A CDN ensures no single failure point by carefully implementing maintenance cycles and integrating additional hardware and software when required. Apart from failures, the CDN handles massive traffic loads by equally distributing the load to the edge proxy servers. We can use scrubber servers to prevent DDoS attacks and securely host content. Moreover, we can use the heartbeat protocol to monitor the health of servers and omit faulty servers.