### 1. System Design: The Content Delivery Network (CDN)

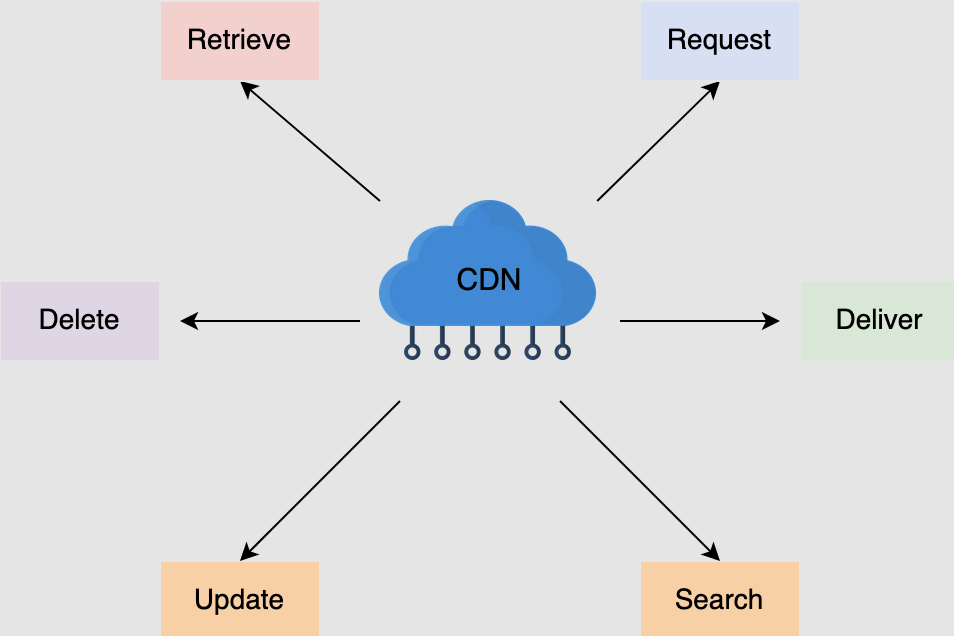
This document provides an in-depth look at the design principles and architecture of a Content Delivery Network (CDN). Key sections include:

* **Architecture Overview**: Describes the core components of a CDN, including edge servers, origin servers, and caching layers. Emphasizes the importance of geographically distributed edge servers to reduce latency and improve access speed.
* **Proxy Servers**: Details the role of proxy servers in caching and delivering content. Explains how proxy servers store frequently accessed data to reduce load on origin servers and speed up content delivery.
* **Cache Hierarchy**: Discusses the use of multi-tier caching, where edge caches handle local requests, and regional or parent caches handle misses from edge caches. This hierarchical approach ensures content is available closer to users.
* **Request Routing**: Explores various request routing techniques, such as DNS-based routing, Anycast routing, and IP Anycast. These methods help direct user requests to the nearest or most optimal edge server.
* **Fault Tolerance and Redundancy**: Describes strategies for ensuring high availability, such as redundant data storage, failover mechanisms, and load balancing across multiple servers. This ensures continuous service even if some servers fail.
* **Scalability**: Details methods for scaling the CDN horizontally by adding more edge servers and vertically by enhancing server capacities. Discusses the importance of elastic scaling to handle traffic spikes.

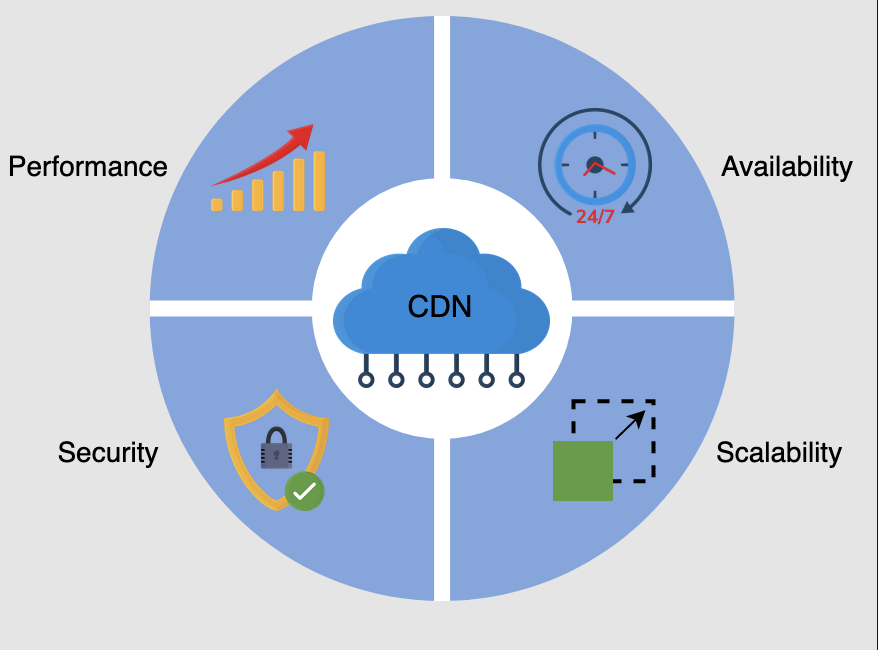
### 2. Introduction to a CDN

This introductory document sets the stage for understanding CDNs by formalizing their requirements and proposed solutions. Key topics include:

* **Proposed Solution**: Presents a high-level overview of a CDN's architecture and its components. Explains the benefits of using a CDN for content delivery.
* **Functional Requirements**: Lists essential functions such as content retrieval, updating, and efficient delivery. Emphasizes the need for the CDN to handle large volumes of data and support real-time updates.



* **Non-Functional Requirements**: Outlines critical attributes like performance, availability, scalability, and security. Details how the CDN must deliver content quickly, handle high traffic loads, scale efficiently, and ensure data security.



* **Building Blocks**: Introduces foundational components like the Domain Name System (DNS), which helps route requests to the nearest edge servers, and load balancers, which distribute traffic evenly across servers to prevent bottlenecks.
* **Performance Metrics**: Discusses key metrics for evaluating CDN performance, such as latency, throughput, and error rates. Highlights the importance of monitoring these metrics to ensure optimal CDN operation.

### 3. Design of a CDN

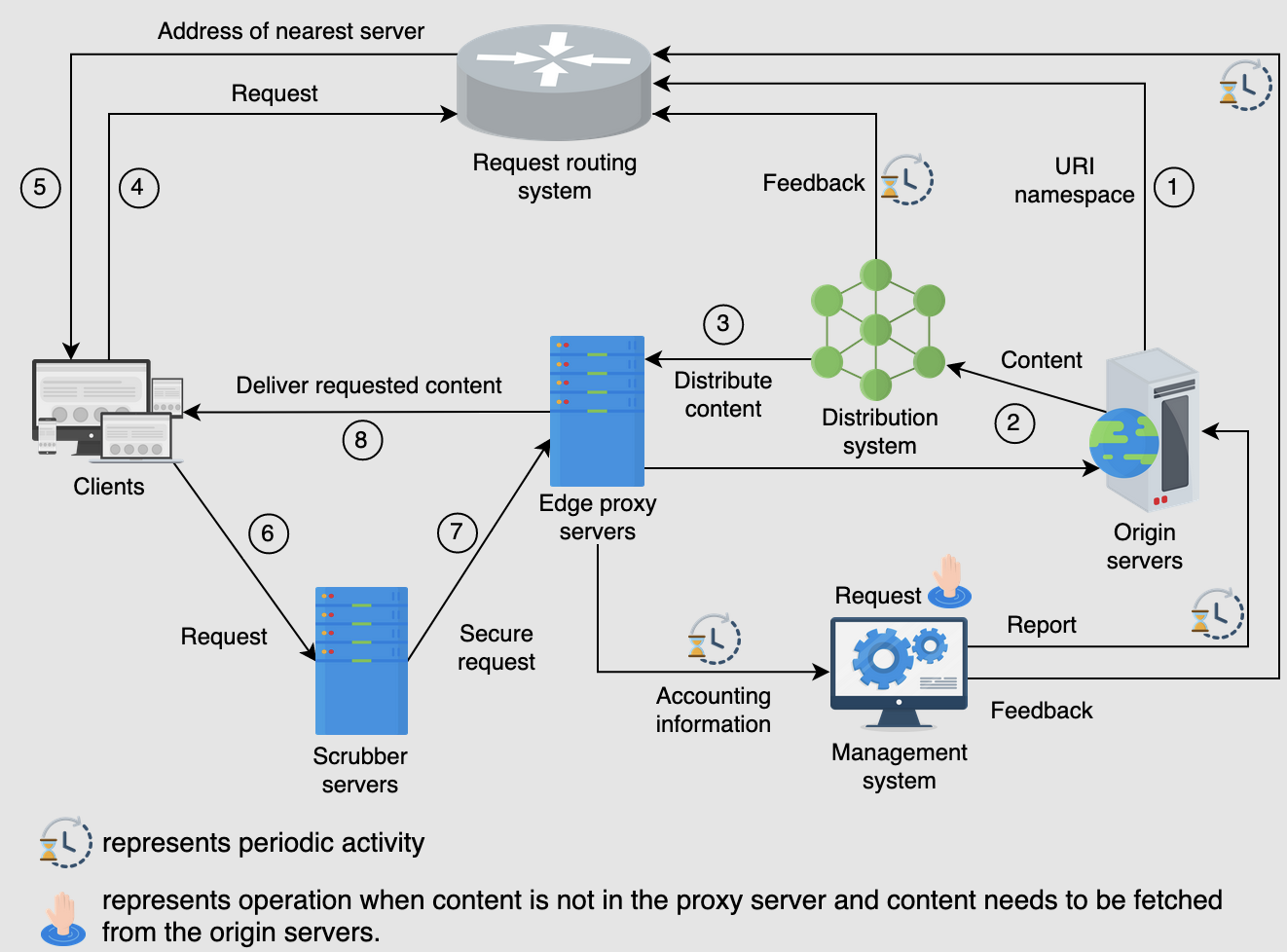
#### Introduction

The document provides an in-depth exploration of the design of a Content Delivery Network (CDN). It covers the fundamental components, workflow, and API design necessary to build and manage a functional CDN.

#### CDN Components

The document identifies and explains the key components that constitute a CDN:

1. **Clients**: End users who access content through various devices such as browsers, smartphones, and other internet-enabled gadgets.
2. **Routing System**: This system directs user requests to the nearest CDN facility, optimizing content delivery based on factors like content location, request load, and server performance. Different routing mechanisms are discussed to forward users to the appropriate CDN facility.
3. **Scrubber Servers**: These servers filter out malicious traffic, protecting the CDN from attacks like DDoS. They only activate during an attack, scrubbing the traffic before routing it to the target destination.
4. **Proxy Servers**: Also known as edge proxy servers, they serve cached content from RAM to users. They store hot data in RAM for fast access and cold data in SSDs or hard drives. Proxy servers also gather accounting information and receive content from the distribution system.
5. **Distribution System**: Responsible for distributing content to all edge proxy servers across different CDN facilities. It uses the internet and intelligent broadcast-like methods to disseminate content efficiently.
6. **Origin Servers**: These servers provide the original content that the CDN caches. If the CDN does not have the requested content, the origin servers serve it directly to the clients.
7. **Management System**: This system oversees the CDN’s performance and resource usage, monitoring metrics like latency, downtime, packet loss, and server load. For third-party CDNs, it also handles accounting and billing.



#### Workflow

The workflow section describes how the CDN components interact to deliver content efficiently:

1. **URI Namespace Delegation**: Origin servers provide the URI namespace of cached objects to the request routing system.
2. **Content Distribution**: Origin servers publish content to the distribution system, which then distributes it to edge proxy servers. The distribution system provides feedback to the routing system about the cached content locations.
3. **Request Routing**: Clients request a suitable proxy server from the routing system, which returns the IP address of an appropriate server.
4. **Security Scrubbing**: Client requests pass through scrubber servers to filter malicious traffic.
5. **Content Serving**: Edge proxy servers serve the requested content to clients and periodically forward accounting information to the management system. If content is not available at the edge server, the request is forwarded to parent proxy servers or the origin server.

#### API Design

The API design section outlines the interfaces for various CDN functionalities:

1. **Retrieve Content**: Proxy servers request content from origin servers using the /retrieveContent API. Parameters include proxyserver\_id, content\_type, content\_version, and description.
2. **Deliver Content**: Origin servers deliver content to proxy servers using the /deliverContent API. Parameters include origin\_id, server\_list, content\_type, content\_version, and description.
3. **Request Content**: Clients request content from proxy servers using the /requestContent API. Parameters include user\_id, content\_type, and description.
4. **Search Content**: Proxy servers search for content in peer proxy servers using the /searchContent API. Parameters include proxyserver\_id, content\_type, and description.
5. **Update Content**: Proxy servers update content in peer proxy servers using the /updateContent API. Parameters include proxyserver\_id, content\_type, and description.
6. **Delete content**

### 4. In-depth Investigation of CDN: Part 1

The document, "In-depth Investigation of CDN: Part 1," part of the Grokking Modern System Design course, explores complex aspects of CDN models and their application in modern web traffic management. It primarily focuses on CDN caching strategies, architecture, and dynamic content optimization, breaking down these elements to provide a deep understanding of their functions and benefits.

#### CDN Models: Push and Pull

The document starts by delineating the two primary CDN models: Push and Pull.

* **Push CDN**: This model involves preemptively sending content from the origin servers to CDN proxy servers. It is best suited for static content that doesn't change often. The push strategy allows content providers to manage content distribution to ensure availability across geographically distributed proxy servers. However, it may lead to inefficiencies like redundant content pushes if not managed correctly.
* **Pull CDN**: Contrarily, the Pull model reacts to user requests by retrieving content from the origin servers only when needed. This model is more efficient for dynamic content that changes frequently. It minimizes storage requirements on proxy servers and reduces costs by caching content only when there is actual demand.

The document emphasizes that many content providers leverage both models to maximize efficiency and cost-effectiveness.

#### Dynamic Content Caching Optimization

Dynamic content presents unique challenges for caching due to its ever-changing nature. The document details strategies to optimize caching for such content:

* **Execution of Scripts at Proxy Servers**: By processing certain operations at the edge, closer to the user, CDNs can significantly reduce latency and server load. For instance, adapting content based on user location or other real-time data can be more efficiently handled at the proxy server rather than the origin server.
* **Compression Techniques**: To further reduce data load and improve transmission speeds, advanced compression techniques such as Cloudflare's Railgun or Edge Side Includes (ESI) are used. ESI, for example, allows partial updates to web pages which prevents redundant data transfer by caching unchanged parts of the page.

#### Multi-tier CDN Architecture

The document explains the hierarchical, tree-like structure of CDNs, designed to efficiently distribute content:

* **Edge and Parent Proxy Servers**: In a multi-tier CDN, content flows from the origin server to parent proxy servers and then to edge servers closer to the end users. This tiered approach helps scale the CDN to handle large volumes of data and user requests without overloading the origin server.
* **Data Distribution**: The process involves distributing data across a network of proxy servers using a controlled replication strategy to ensure that content is available near users and to reduce latency.

#### Proximity and Server Selection

Key techniques for directing user requests to the nearest and most appropriate proxy server include:

* **DNS Redirection**: Utilizes DNS lookups to direct requests to the nearest CDN node based on factors like geographic location and server load, enhancing content delivery speed and reducing latency.
* **Anycast Routing**: Shares a single IP address among multiple edge servers, routing users to the nearest server based on network topology and conditions, thereby optimizing response times and balancing server loads.
* **HTTP Redirection**: Involves redirecting client requests at the HTTP level, allowing for flexible and dynamic content delivery paths based on real-time server availability and performance metrics.

### 5. In-depth Investigation of CDN: Part 2

This document is part of a series on Content Delivery Networks (CDNs) and focuses on advanced topics concerning the deployment and maintenance of CDN infrastructure, particularly on achieving content consistency and strategic proxy server deployment.

#### Content Consistency Mechanisms

Maintaining data consistency across CDN proxy servers and origin servers is critical to ensure users receive the most current content without encountering stale data. The document discusses several mechanisms to achieve this:

* **Periodic Polling**: Proxy servers periodically query the origin server to check for updates. This method is bandwidth-intensive and less efficient for infrequently updated content. It utilizes a "time-to-refresh" (TTR) parameter to manage polling intervals.
* **Time-to-Live (TTL)**: Each data object is assigned a TTL that dictates how long it can be served before it must be revalidated. This method reduces unnecessary updates by allowing content to remain in cache until the TTL expires, at which point the proxy server checks for updates.
* **Leases**: Origin servers grant leases for data, indicating a timeframe during which the proxy server can assume the content will not change. Upon lease expiration, the proxy server must check back with the origin server for possible updates. This method minimizes the messaging overhead between proxy servers and the origin, with dynamic adjustments based on observed load.

#### Deployment of CDN Proxy Servers

The document provides guidance on strategic deployment of proxy servers, essential for optimizing CDN performance and resource utilization:

* **Location and Quantity**: Deciding where to place proxy servers involves evaluating network connectivity and user density. Tools like ProxyTeller help determine optimal placements by analyzing metrics such as hit ratios, network bandwidth, and client response times.
* **On-premises vs. Off-premises**: The choice between deploying proxy servers on-premises (near user aggregation points like IXPs) or off-premises (within ISP networks) depends on specific business needs and technical constraints.
* **Technology Use Cases**: The example of Google using split TCP connections illustrates how maintaining persistent, high-capacity connections at IXPs can significantly reduce latency perceived by users by circumventing the slow-start and handshake delays typical of remote server communications.

#### Specialized CDN Services

The document highlights how some companies choose to build and operate their own CDNs instead of relying on third-party providers:

* **Netflix's Open Connect**: A prime example where Netflix uses its custom CDN to handle its massive video streaming traffic. This proprietary CDN allows Netflix to control data flow efficiently, manage network health, and optimize content delivery by keeping popular content longer and managing network issues more directly.
* **Advantages of Private CDNs**: While more expensive initially, private CDNs can offer long-term savings and better control over content delivery, especially for services with specific requirements or massive scale.

### 6. Evaluation of CDN's Design

#### Performance

The document outlines how CDNs achieve high performance primarily through the reduction of latency. Key strategies include:

* **Content Serving from RAM**: Proxy servers primarily serve content directly from RAM, which reduces access times significantly.
* **Strategic Proxy Server Placement**: By placing proxy servers near users or within Internet Service Provider (ISP) networks, CDNs can provide faster access to content and handle high traffic more efficiently.
* **Optimized Request Routing**: The routing system within a CDN directs user requests to the nearest proxy server, further reducing latency.
* **Layered Proxy Servers**: If content is not available on one layer of proxy servers, the request is passed up to the next layer. This hierarchy ensures efficient content retrieval and maintains performance even under high load.

#### Availability

Availability in CDNs is maintained through several mechanisms:

* **Redundancy**: By replicating data across multiple proxy servers, CDNs prevent a single point of failure.
* **Load Balancing**: Load balancers distribute incoming requests to different proxy servers based on their current load and geographical location, enhancing the robustness of the network.

#### Scalability

CDNs are designed to scale seamlessly in response to increasing demand:

* **Horizontal Scaling**: Additional proxy servers can be added to the network to handle increased traffic without significant redesign of the network structure.
* **Content Distribution Efficiency**: By bringing content closer to the user, CDNs minimize the need for high bandwidth connections back to origin servers.

#### Reliability and Security

CDNs enhance their reliability and security with various measures:

* **Maintenance and Upgrades**: Regular updates and maintenance cycles ensure that the system remains robust against failures.
* **Scrubber Servers**: These are used to mitigate Distributed Denial of Service (DDoS) attacks, protecting the network from malicious traffic.
* **Health Monitoring**: Heartbeat protocols and other monitoring tools help detect and isolate faulty servers, ensuring continuous service.