Content and API Acceleration Using Content Delivery Networks

Tom Maximilian von Allwörden, Markus Sosnowski*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: tom.von-allwoerden@tum.de, sosnowski@net.in.tum.de

***Abstract*—Modern web services are accessed all over the world by potentially many users. Content Delivery Networks (CDNs) play an integral role in lowering the latency for end-users that are using these services by caching objects like a website's source code and images close to the user. Traditionally, CDNs have been used for static content, but with the rise of personalized user experiences and dynamically changing content, new requirements for CDNs emerged. APIs are the most common way such content is accessed, so an effort is made to accelerate these.**

This paper will give a brief introduction into how CDNs work, how virtual network optimize routing and then examine how dynamic content and APIs can benefit from CDNs via different invalidation techniques and what tradeoffs to consider. Furthermore the paper collects diverse approaches in the field of edge computing for moving the application logic itself close to end users.

***Index Terms*—cdn, internet, api acceleration, dynamic content, virtual network, edge computing**

1. Introduction

Content Delivery Networks (CDNs) bring web-services like websites or APIs closer to the end-user by deploying various globally distributed replica- or edge servers that cache the content of the origin server. Such locations are called Point of Presences (PoPs) and typically consist of Internet Exchange Points or locations inside of an Internet Service Provider (ISP)'s networks [1]. Sitaraman et al. [2] show the main business incentive: high load times of a website can avert potential users away from a commercial site. A large CDN provider like Akamai therefore deploys over 170 000 edge servers in about 1300 networks to accommodate todays needs [3,4], especially in times where latency is the single biggest influence on web performance, according to Ilya Grigorik [5].

When a user tries to access a web-service, instead of visiting the origin server directly, they get directed to a geographically close edge server by the CDN's request-routing system. If the content is cached, it is then immediately returned, thus lowering the latency; however if the content is not cached the CDN must fetch it from the origin [6]. This can be done effectively using a CDN's virtual overlay network [2].

Beheshti [7] mentions that caching is especially effective for static content like images or script files, but dynamic content and APIs are traditionally harder to cache because simple caching could lead to the user getting stale, outdated content.

This paper is structured as follows: Section 3 presents an overview of different request routing methods. We take a closer look at before mentioned virtual networks and how connections to the origin are accelerated in Section 4. Section 5 is about handling dynamic content with a focus on a technique called invalidation. Lastly, in the umbrella edge computing fall various approaches that improve a site's performance, which we discuss in Section 6.

2. Related Work

Two of the focal points of this paper are cache invalidation and consistency. We will not touch on the protocols that different CDNs use to implement their invalidation strategies, such as leases. This is discussed in more detail by Ninan et al. in [8]. Wingerath et al. describe their approach for handling dynamic content in [9]. They focus on the refresh procedure to detect stale content and afterwards invalidate it automatically. We, on the other hand, will look at manual invalidation, where we invalidate instantly when events cause data to become stale.

3. Request Routing

Katz-Bassett et al. [10] state that the request-routing system is responsible for directing a user from the origin server to one of the CDN edge servers. There are three common approaches CDNs use: Domain Name System (DNS)-based, IP anycast-based and HTTP 304-based redirecting, where the user first contacts the origin, which then redirects the user to a CDN server. We take a look at the former two in more detail.

Gang Peng [6] presents more advanced methods, such as those based on Peer-2-Peer systems, but these are beyond the scope of this paper.

3.1. Domain Name System

With the DNS-based approach, which is covered by Hung et al. [11], the administrator of a domain origin places a CNAME entry pointing to the CDN's DNS server entry in its domain name server.

When a user requests a website, their local DNS (LDNS) resolver, most commonly provided by their ISP, will recursively query the domain name and contact the CDN's DNS server. This server then selects the edge server PoP based on the LDNS IP-address [11]. This is usually a good indicator for the approximate geographical location of a user when not using a public resolver [10].

 doi: 10.2313/NET-2023-11-1_11

Other metrics, such as the network load and previous edge server choices are also taken into account when selecting the edge server within a PoP [2].

Notice that with this scheme the end user's IP-address is not directly used for selecting an edge server. Sitaraman et al. [12] and [10] discuss the proposed EDNS-Client-Subnet DNS extension that allows the LDNS server to send the user's /24 IP-address-prefix along with the request, thus allowing the CDN to make a better decision.

This approach is widely popular. Examples of CDNs using DNS-based approaches include Akamai [2], Edgenext [13] and AWS Cloudfront [14].

3.2. IP Anycast

With the IP Anycast approach, every edge server is assigned the same IP-address. This method makes use of the fact that when a network router gets the same route announced from different interfaces, it chooses only the one with the lowest hop count. A disadvantage with this is that CDNs have less control over which edge server the client connects to, but studies have shown that the method leads to the optimal edge server in 80% of the time [10].

Popular CDNs, which use this approach include Cloudflare [10], Microsoft Azure Front Door [15] and Google Cloud CDN [16].

4. Virtual Network

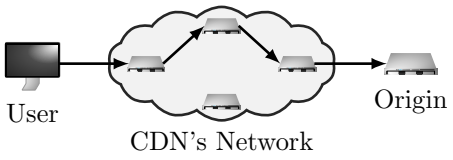

Figure 1: Routing using a CDN's virtual network

Classical internet routing has several disadvantages: The Border-Gateway Protocol that ISPs use to exchange routes is known to lead to sub-optimal routing since it has no topology information and solely uses hop-count as its main metric [2]. The propagation of failed routes can also take a significant amount of time. Furthermore, ISPs are also driven by financial incentives and could prefer routes over cheaper but slower peering-partner compared to costly Customer-Provider (C2P) connections [2].

To mitigate this, CDNs influence the routing path of a package by adding their servers as intermediate hops, resulting in an overall more optimal route. They do this by continuously measuring latency and package drop between their edge servers and taking the topology information of their server locations into account [2].

Some big players like AWS Cloudfront [17] and Microsoft Azure [18] come with their own backbone infrastructure between different PoPs, circumventing suboptimal internet routing even further. Common optimizations when using the virtual network include:

Long-Term TCP& SSL connections A user's TCP & SSL connections are terminated close to them at the edge. From there on, the edge server uses pre-established TCP & SSL connections between the origin and various edge servers. As the respective handshakes require several round-trips, the saved cost quickly accumulates [2,19].

No slow start TCP normally begins in a "slow start" phase with small window sizes. CDNs skip this phase and choose larger window sizes by taking the previously mentioned network measurements into account [2,19].

Data Compression : Images and other objects can be compressed on the edge close to the origin server, and thereby loaded quicker [2,19].

Abundant packages : CDNs can try to send the same packages over multiple routes and then take the one that arrives first. This also helps against package loss, but can cause more congestion [2].

Virtual networks thus accelerate the content fetch from the origin in case of a cache-miss and are therefore important for uncacheable and dynamic content. It was shown, that in Asia the use of a CDN's more optimized routing can lead to a 30-50% performance improvement [2].

5. Caching of Dynamic Content

In the last section we saw a general way to speed up requests to the origin, but we did not make use of a CDN's caching capability yet. This section introduces dynamic content and discuss techniques of caching it.

Dynamic content is content that changes over time but might be the same for all users while personalized content refers to dynamic content that is different for each user. For example, a site presenting the top 10 most viewed news articles of the day is dynamic content, while a user's shopping cart is personalized content. As the storage requirements for the latter scale with the number of users with each entry only relevant for a single user, caching it might be of little value for a service, but we will see a way of handling this content in Section 6.

Lawrence et al. [20] highlights that not all dynamic content is the same: some content is valid for a long time, while other might change with every request made. The update trigger might be an external event, such as a new blog post release, or it might simply be after a certain period of time has passed. The latter would be easier to handle, as one could just choose this period as the cache's Time-To-Live (TTL).

The main problem we face with caching is consistency, as we have multiple, distributed copies of an object that might become stale at any time. But even for dynamic content that normally changes with every request, a site owner might choose to intentionally give up on consistency and decide to cache it for a brief period, a practice referred to as micro-caching, as this alone can take a considerable load off the origin [9]. This is especially true for large services that deal with hundreds to thousands of requests per second.

Another way CDNs can benefit APIs is by accumulating multiple requests over a short period of time and sending them bundled to the origin. Therefore, taking load and overhead off of the origin [21].

As APIs usally interface with a database at the origin, there are efforts that try to map the cache objects to this underlying relational data. The analysis is typically done by observing the traffic between web listeners, databases

and web services, but the analysis of this mapping is too expensive and unreliable to use in practice [20].

Therefore, we will focus on the most common way dynamic content is cached, namely, fine-grained cache control and invalidation.

5.1. Cache Control and Invalidation

In comparison to the difficult and expensive task of maintaining and modeling data dependencies between cache objects, invalidation is a cheap and simple alternative [20]. We want to cache dynamic content for as long as possible and invalidate it as soon as it becomes stale. The most common way invalidation occurs is simply when the set livetime of a cached item expires. This livetime can be controlled via the "Expires" header and is passed in the response from the origin [22]. Another option is to invalidate the cash manually. This can be done via the CDN's API, also refered to as *purging* [23,24].

We generally want to invalidate as little as necessary so that we reduce the number of cache misses. Fine-grained invalidation is generally more difficult to do and costs the CDN more resources. CDNs commonly provide different purging methods that vary in their level of granularity [23,24]:

By URL Invalidates the cache object that is associated with the URL.

By Prefix Invalidate all objects with a given URL prefix

By Tag The origin Server can associate cache objects with a tag by adding a "Cache-Tag" header to the response. This way many possibly by path unrelated endpoints can be purged with a single request.

By Geo Only invalidate cache objects at certain PoPs.

Even with purging, users could still get stale content if the CDN only provides weak consistency. Weak consistency means that the cached objects on the different PoPs only *eventually* get invalidated instead of *instantly* as it is the case with strong consistency [25]. Even CDNs such as Fastly that advertise a "instant purge" feature do not have strong consistency across different PoPs [26]. Although this can already be enough for services, where users requesting the same content tend to be geographically close together. When we look at a typical restful API, there are generally two types of requests [24]:

state requesting GET
state changing POST, PUT, DELETE

The former can usually be cached, while the latter are commonly passed through to the origin [24]. In response to a state changing request, the corresponding cache entries need to be invalidated. Let us explore how invalidation mechanics can be used in practice by examining the following example of a video watching & commenting platform with the following endpoints:

- `PUT /api/<videoid>/comment`: users can leave comments with their username attached.
- `GET & PUT /api/<userid>/profile`: users can retrieve or change their profiles e.g. their username.

In the event that a user changes their username, we want to invalidate all comments made by them to reflect this change. In this scenario, invalidating every object separately is a costly task. Instead, we use tags to associate each comment and possibly other user-related endpoints with the user's id and purge them with one purge call [24,27,28]. Another possibility is to handle purging and other logic on the edge servers themselves. We highlight this in more detail in Section 6.

Further techniques to to improve invalidation include:

Cache-keys The index into the cache to associate an URL with a cached object is called the cache-key. Modern CDNs allow one to include or exclude certain parts and parameters of the URL and request headers into the key to avoid unnecessary cache-misses [29].

Auto revalidation A CDN might try to fetch the newest state of an object once the TTL has expired. If the content was invalidated via a message from the origin, the origin could pass the newest version of that object along [25].

Invalidation order Monitoring the popularity of objects allows a CDN to invalidate the popular ones first as these affect the most users [20].

In recent years, GraphQL APIs have become a modern alternative to RESTful APIs. Wundergraph and Hygraph are examples of services that specialize in GraphQL API management and include CDN-like caching capabilities. These can handle invalidation automatically to some extent, but use relatively simple approaches: Wundergraph allows the developer to define dependencies between APIs but not the data dependencies within an API [5,30]. It caches all objects for 10 seconds. Hygraphs supports invalidation and does this based on the GraphQL schema and content changes, but only with weak consistency [31].

6. Edge Computing

Invalidation is not perfect for applications with a high number of uncacheable requests or whenever we have a cache-miss, as the origin is the sole producer of fresh content and needs to be contacted.

The idea behind edge computing is to move as much application logic as possible close to the user onto edge servers. When a request can be completely handled on the edge, no additional overhead is needed to reach out to the origin server, thus decreasing latency and taking load of the origin. For this, applications are usually split into an edge and an origin component [21].

One popular approach is the deployment of *edge functions*. Edge functions run in response to incomming requests. These can then modify the request, interact with the CDN's caching system, or implement entire API-endpoints on edge, and thus behave similarly to an API-gateway [2]. For example, the invalidation logic discussed in the previous section can be implemented as edge functions. Most CDNs offer such functions, for example AWS Lambda@Edge [32] or Cloudflare Worker [33].

Applications that only need a static database to function are prime candidates to be moved completely onto the edge. For example encyclopedias, dictionaries or product catalogues of e-commerce businesses with a fixed number of products [21]. In the following sections, we will discuss various techniques in more detail.

6.1. Normalization

URL normalization is applied to the HTTP path and parameters of incoming requests and ensures the same

 doi: 10.2313/NET-2023-11-1_11

encoding for all paths in compliance with RFC3986 [34]. As the path is a common component of the cache key, different encodings of the same object can lead to unnecessary cache misses.

For example, these two URLs refer to the same object: "example.com/api/./user/青 沼" and "example.com/api/user/%E9%9D%92%E6%B2%BC". Since these would hit in different cache keys, normalization is applied to map the former path into the latter.

6.2. Edge Site Includes

Personalized sites are created explicitly for the user requesting them, and it thus provides little value for a CDN to cache the site as is, especially because the presented content might become stale fast anyway [20]. But looking at how personalized sites are constructed one might notice that they are often built using shared fragments [20]. For example, Figure 2 shows different fragments of the bing.com website with fragments such as recent news or weather that could be shared across users of the same city.

Edge-Site-Includes (ESI) allow for the creation of dynamic, fragmented websites by assembling the website on the edge. Here, a website consists of an HTTP template with special ESI tags, that describe the type and location of the fragments that should be included in the final site. The fragments are shared between different templates and can each be individually cached and have their own TTL [2,20]. The ESI environment allows one to encapsulate personalized information into a fragment that itself can be cached [20]. A edge server might also speculatively assemble a site beforehand, based on the user's last visited sites [35].

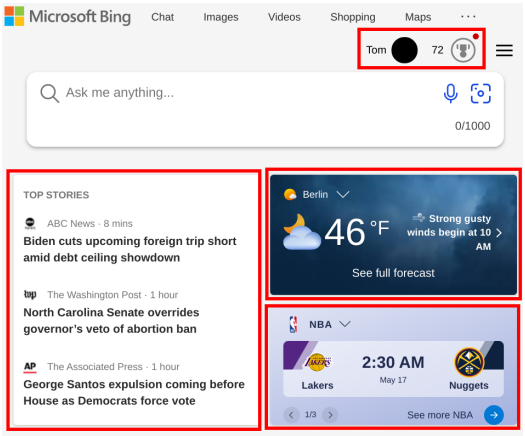

Figure 2: Sample fragmentation of bing.com

6.3. Validation & Authorization

Before forwarding a request to the origin it can be checked for ill-formatted content [2]. A certain kind of validation is authorization, where we check if the user has the necessary access rights. JSON Web Tokens are a popular method to authenticate users. We assume that the user got a API token from an issuer (most likely the origin) in advance. The client then includes this token in the "Authentication" header within each request. The edge which is provided a JSON Web Key Set by the Issuer, checks the user token against this set [36].

Other authorization schemes like OAuth or openid connect can also be handled on edge [37,38].

6.4. Aggregation

Some simple API endpoints might only consist of requests to other external services like cloud databases or weather APIs. These requests can be moved from origin to edge servers and format the responses on edge [2]. Additionally, the responses of those endpoints might be cached on the edge server. More advanced handling can also be achieved using edge functions. Even transactional tasks can benefit from solely exchanging raw and compressed data between the origin and edge servers, where the final product is then assembled on edge [21].

6.5. Session Handling

Websites commonly keep sessions with clients to recognize users and keep temporary data across multiple requests. Often, the returned websites embed this temporary information based on the session token. For example, current shopping cart items can be associated with the session. Edge servers can be used to replace placeholders in a generic HTML site with the content acquired by the state. The other way around is also possible: If no session token is found, the edge can directly return the default site for anonymous users [20].

7. Conclusion and Future Work

CDNs are an essential part of accelerating web services and APIs through the caching of both static and dynamic content. They rely on a request-routing system to connect a user to an edge server and thus lower the main factor that contributes to a site's loading time: latency. We looked at the concept of virtual networks, which lower latency between the edge and origin servers and ways to accelerate dynamic content and APIs. The different methods of invalidation and cache control are tricky to facilitate correctly and demand careful consideration from developers with respect to the application build.

This paper also laid out how edge computing can be used to move different aspects of applications onto edge, like authentication, site assembly or endpoints based on accumulative requests. To conclude, these approaches can not only take load off the origin, lowering the bandwidth needed, but also make end user's experience better through faster loading times, especially when their request can completely be handled on edge.

To lessen the design complexity of edge functions for custom purging, research should be conducted to look at how invalidation can be done more automatically by using assumptions about data dependencies in the cached objects and API endpoints, and how unified API Guidelines can help in this endeavour.

References

[1] "Cloudflare Glossary - Internet exchange points," https://www.cloudflare.com/de-de/learning/cdn/glossary/internet-exchange-point-ixp/, [Online; accessed 28-March-2023].

[2] J. S. Erik Nygren, Ramesh K. Sitaraman, "The Akamai Network: A Platform for High-Performance Internet Applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, Aug. 2010. [Online]. Available: https://doi.org/10.1145/1842733.1842736

[3] B. M. Maggs and R. K. Sitaraman, "Algorithmic Nuggets in Content Delivery," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 3, pp. 52–66, Jul. 2015. [Online]. Available: https://doi.org/10.1145/2805789.2805800

[4] "Why Akamai," https://www.akamai.com/why-akamai, [Online; accessed 28-March-2023].

[5] I. Grigorik, *High Performance Browser Networking: What every web developer should know about networking and web performance*. "O'Reilly Media, Inc.", 2013.

[6] G. Peng, "CDN: Content Distribution Network," 2004. [Online]. Available: https://arxiv.org/abs/cs/0411069

[7] H. Beheshti, "Fastly - Leveraging your CDN to cache "uncacheable" content," https://www.fastly.com/blog/leveraging-your-cdn-cache-uncacheable-content, [Online; accessed 28-March-2023].

[8] A. Ninan, P. Kulkarni, P. Shenoy, K. Ramamritham, and R. Tewari, "Cooperative Leases: Scalable Consistency Maintenance in Content Distribution Networks," in *Proceedings of the 11th International Conference on World Wide Web*, ser. WWW '02. New York, NY, USA: Association for Computing Machinery, 2002, pp. 1–12. [Online]. Available: https://doi.org/10.1145/511446.511448

[9] W. Wingerath, F. Gessert, E. Witt, H. Kuhlmann, F. Bücklers, B. Wollmer, and N. Ritter, "Speed Kit: A Polyglot and GDPR-Compliant Approach For Caching Personalized Content," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020, pp. 1603–1608. [Online]. Available: https://doi.org/10.1109/ICDE48307.2020.00142

[10] M. Calder, A. Flavel, E. Katz-Bassett, R. Mahajan, and J. Padhye, "Analyzing the Performance of an Anycast CDN," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 531–537. [Online]. Available: https://doi.org/10.1145/2815675.2815717

[11] Z. Wang, J. Huang, and S. Rose, "Evolution and challenges of DNS-based CDNs," *Digital Communications and Networks*, vol. 4, no. 4, pp. 235–243, 2018. [Online]. Available: https://doi.org/10.1016/j.dcan.2017.07.005

[12] F. Chen, R. K. Sitaraman, and M. Torres, "End-User Mapping: Next Generation Request Routing for Content Delivery," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 167–181, Aug. 2015. [Online]. Available: https://doi.org/10.1145/2829988.2787500

[13] "EdgeNext CDN Introduction," https://home.console.edgenext.com/#/doc/content/cdn/Product%20Introduction/Product%20overview, [Online; accessed 28-March-2023].

[14] "How CloudFront delivers content," https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/HowCloudFrontWorks.html, [Online; accessed 28-March-2023].

[15] "Microsoft Azure Front Door - Traffic acceleration," https://learn.microsoft.com/en-us/azure/frontdoor/front-door-traffic-acceleration, [Online; accessed 28-March-2023].

[16] "Google Cloud CDN - Choose a CDN Product," https://cloud.google.com/cdn/docs/choose-cdn-product, [Online; accessed 28-March-2023].

[17] "Amazon CloudFront Key Features," https://aws.amazon.com/cloudfront/features/?nc1=h_ls&whats-new-cloudfront.sort-by=item.additionalFields.postDateTime&whats-new-cloudfront.sort-order=desc, [Online; accessed 28-March-2023].

[18] "How Microsoft builds its fast and reliable global network," https://azure.microsoft.com/en-us/blog/how-microsoft-builds-its-fast-and-reliable-global-network/, [Online; accessed 28-March-2023].

[19] "Microsoft CDN - Dynamic Site Acceleration," https://learn.microsoft.com/en-us/azure/cdn/cdn-dynamic-site-acceleration, [Online; accessed 28-March-2023].

[20] J. Anton, L. Jacobs, X. Liu, J. Parker, Z. Zeng, and T. Zhong, "Web Caching for Database Applications with Oracle Web Cache," in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '02. New York, NY, USA: Association for Computing Machinery, 2002, pp. 594–599. [Online]. Available: https://doi.org/10.1145/564691.564762

[21] A. Davis, J. Parikh, and W. E. Weihl, "Edgecomputing: Extending Enterprise Applications to the Edge of the Internet," in *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers and Posters*, ser. WWW Alt. '04. New York, NY, USA: Association for Computing Machinery, 2004, pp. 180–187. [Online]. Available: https://doi.org/10.1145/1013367.1013397

[22] "Amazon CloudFront - Managing how long content stays in the cache," https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Expiration.html, [Online; accessed 31-March-2023].

[23] "Cloudflare CDN - Purge Cache," https://developers.cloudflare.com/cache/how-to/purge-cache/, [Online; accessed 31-March-2023].

[24] "Fastly - API Caching, Part 1," https://www.fastly.com/blog/api-caching-part-i, [Online; accessed 30-March-2023].

[25] M. Hossein Sheikh Attar and M. Tamer Özsu, "Alternative Architectures and Protocols for Providing Strong Consistency in Dynamic Web Applications," *World Wide Web*, vol. 9, no. 3, pp. 215–251, Oct. 2006. [Online]. Available: https://doi.org/10.1007/s11280-006-8563-1

[26] "Fastly - Purging," https://developer.fastly.com/reference/api/purging/, [Online; accessed 31-March-2023].

[27] "Fastly - API Caching, Part 2," https://www.fastly.com/blog/api-caching-part-ii, [Online; accessed 30-March-2023].

[28] "Fastly - API Caching, Part 3," https://www.fastly.com/blog/api-caching-part-iii, [Online; accessed 30-March-2023].

[29] "Amazon CloudFront - Controlling the Cache Key," https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/controlling-the-cache-key.html, [Online; accessed 31-March-2023].

[30] "Wundergraph Architecture - Manage API Dependencies explicitly," https://docs.wundergraph.com/docs/architecture/manage-api-dependencies-explicitly, [Online; accessed 31-March-2023].

[31] "Hygraph - Caching," https://hygraph.com/docs/api-reference/basics/caching, [Online; accessed 31-March-2023].

[32] "AWS Lambda Features," https://aws.amazon.com/lambda/features/, [Online; accessed 31-March-2023].

[33] "Cloudflare - How Workers work," https://developers.cloudflare.com/workers/learning/how-workers-works/, [Online; accessed 01-April-2023].

[34] T. Berners-Lee, R. T. Fielding, and L. M. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," RFC 3986, Jan. 2005. [Online]. Available: https://doi.org/10.17487/RFC3986

[35] Suresha and J. R. Haritsa, "On Reducing Dynamic Web Page Construction Times," in *Advanced Web Technologies and Applications*, J. X. Yu, X. Lin, H. Lu, and Y. Zhang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 722–731. [Online]. Available: https://doi.org/10.1007/978-3-540-24655-8_78

[36] "Google Cloud CDN - Using JWT to authenticate users," https://cloud.google.com/api-gateway/docs/authenticating-users-jwt, [Online; accessed 31-March-2023].

[37] J. T. Zhao, S. Y. Jing, and L. Z. Jiang, "Management of API Gateway Based on Micro-service Architecture," *Journal of Physics: Conference Series*, vol. 1087, no. 3, p. 032032, sep 2018. [Online]. Available: https://doi.org/10.1088/1742-6596/1087/3/032032

[38] "Wundergraph - OpenID Connect-Based Authentication," https://docs.wundergraph.com/docs/features/openid-connect-based-authentication, [Online; accessed 31-March-2023].