The Potential of QUIC for Web Proxies Accelerating REST APIs

Kevin Fischer, Markus Sosnowski*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: kevin.fischer@tum.de, markus.sosnowski@tum.de

***Abstract*—The increasing demand for faster and more secure web services have driven significant advancements in networking protocols. This paper explores the potential of QUIC, a modern transport protocol, to accelerate web proxies and REST APIs, which are critical components in today's Internet architecture. We provide an overview of the REST (Representational State Transfer) paradigm and web proxies. Aswell, we explore basic and in-development features of QUIC and related work. The paper then analyses the problems, requirements and challenges of REST APIs, Web Proxies, and Content Delivery Networks and maps features of QUIC to them to show their potential for acceleration. The paper finds that QUIC can enhance the performance, security and reliability of REST APIs served through web proxies and CDNs. Still, there are connections between QUIC, REST APIs, and web proxies that need further research.**

***Index Terms*—quic, content delivery networks, rest, web proxies**

1. Introduction

In an era where web traffic is growing rapidly, the need for reliable, fast, and secure networking protocols has never been more critical. The most widely adopted solution for ensuring secure web communication is the Hypertext Transfer Protocol Secure (HTTPS), which integrates the Transmission Control Protocol (TCP) with Transport Layer Security (TLS). While HTTPS was and still is the standard for secure web traffic, its reliance on TCP has introduced several challenges and problems, particularly with the increasing and changing needs of modern web applications. To address these challenges, Google proposed QUIC in 2016 for standardization, among other things, with the goal of enhancing the performance and security of web communications beyond what TCP could traditionally offer [1].

When considering mechanisms for client-server communication, REST (Representational State Transfer) or RESTful APIs have become the predominant paradigm. REST outlines architectural requirements that APIs (Application Programming Interfaces) must follow [2]. In many implementations, requests to REST APIs are routed through web proxies. These web proxies can provide several advantages, such as improved security, reduced latency, and enhanced performance [3]. Given these benefits, the role of web proxies in optimizing the performance of REST APIs is of significant interest.

This paper explores the potential of QUIC to accelerate REST APIs, specifically in scenarios in which web proxies are involved. We aim to analyze how the features of QUIC can address the challenges and requirements of REST APIs when served by web proxies, thereby offering a more efficient and secure solution.

The remainder of this paper is structured as follows: Section 2 provides background information about QUIC, REST, Web Proxies, and CDNs (Content Delivery Networks), Section 3 presents related work and sets this paper apart from others. In Section 4, we analyze the problems, challenges, and requirements of REST APIs served by Web Proxies. In Section 5 we then map the features of QUIC to the problems that we found in Section 4. The paper concludes with a summary of our findings in Section 6.

2. Background

This section gives an overview of the relevant concepts for the following problem analysis and mapping of features to the problems.

2.1. QUIC Protocol

QUIC is a transport protocol, which was developed by Google and later standardized by the Internet Engineering Taskforce (IETF) [1]. Unlike traditional protocols like TCP in combination with TLS, QUIC is built on top of the User Datagram Protocol (UDP) and contains features that aim to reduce latency, enhance security and improve the overall efficiency of web communication.

2.1.1. Key Features of QUIC. QUIC introduces several features that differentiate it from the traditional TCP+TLS combination:

- **1-RTT and 0-RTT Handshakes:** While TCP requires at least one round-trip time (1-RTT) for the handshake and TLS needs 1-RTT as well, QUIC combines these steps [4]. Figure 1 shows how QUIC establishes a secure connection with a single round-trip. Moreover, QUIC supports 0-RTT handshakes for repeated connections, thereby data can be sent immediately [5].
- **Built-in Encryption:** QUIC mandates encryption by default and uses TLS 1.3 for secure communications [6]. From the start of the communication, the built-in encryption provides confidentiality, integrity, and authenticity. As a result, QUIC does

 doi: 10.2313/NET-2025-05-1_02

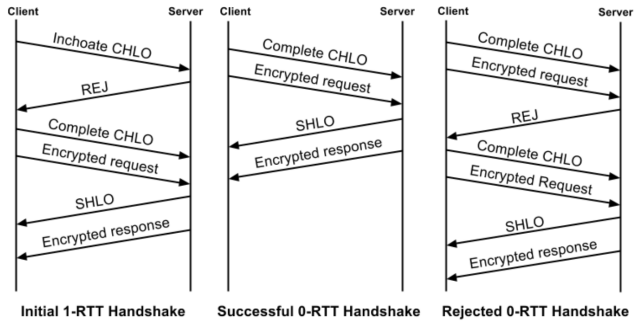

Figure 1: Timeline of QUIC's initial 1-RTT handshake, a subsequent successful 0-RTT handshake, and a rejected 0-RTT handshake [5].

not need an extra encryption layer like TLS on top of TCP.

- **Connection Migration:** QUIC allows to migrate a connection and continue it seamlessly when a client's IP address changes e.g. when switching from Wi-Fi to cellular data. While TCP uses a 4-tuple, QUIC utilizes a connection ID to identify a connection from a client to a server [7].
- **Stream Multiplexing:** QUIC can handle multiple streams of data within a single connection without the issue of head-of-line blocking, which occurs in TCP when packet loss in one stream can delay the delivery of data in other streams [8]. By using different `StreamIDs`, packet loss does not affect all streams, but only streams that are in the lost packet [9]. Figure 2 illustrates how QUIC behaves in comparison to TCP when encountering packet loss while having multiple streams.

The following two are extensions to QUIC that are particularly relevant for scenarios with a web proxy layer and CDNs. There is still ongoing work and development on these extensions:

- **MASQUE (Multiplexed Application Substrate over QUIC Encryption):** MASQUE is an extension for QUIC which allows to multiplex traffic of different applications with a single QUIC connection. It can create a secure tunnel to a proxy and because of that, it is particularly useful for applications like VPNs or proxy services with multiple streams, since the streams can be handled securely within one connection [10], [11]. An example scenario with a proxy server is demonstrated in Figure 3.
- **QUIC-Aware Proxying:** QUIC-Aware Proxying allows clients to tunnel QUIC connections and adds a new "forwarded" mode as stated in [12]. It allows to use special-purpose transforms rather than full re-encapsulation and re-encryption of QUIC packets when they are forwarded by a proxy.

2.2. REST APIs

REST is a paradigm that outlines principles for building scalable and stateless web services. It was first introduced by Roy Fielding in 2000 in [2], since then REST has become the dominant framework for designing APIs [14].

2.2.1. Key Principles of REST. As defined by Roy Fielding in [2], REST is based on the following principles:

- **Stateless:** Each request from a client to a server has to contain all necessary information, as the server does not memorize the client state between requests.
- **Client-Server Architecture:** REST separates concerns by defining distinct roles for clients (handling user interaction) and servers (managing data and logic).
- **Uniform Interface:** A consistent and standardized interface is used. Typically HTTP methods (GET, POST, PUT, DELETE) are used to interact with resources.
- **Code on Demand:** Web servers can send executable programs to clients. Oftentimes communication is needed in advance to assure that the client is able to process the offered ressource.
- **Layered System:** REST systems are layered, allowing intermediaries like proxies to manage requests without clients being aware.
- **Cache:** Responses can be marked as cacheable to enable reuse of responses for identical requests, thereby improving performance.

2.3. Web Proxies and HTTP Caching

Web proxies act as intermediaries between clients and servers. They forward requests from clients to servers and vice versa. While offering several benefits, such as enhanced security, traffic filtering and more, proxies can optimize network performance by caching frequently requested (static) content [15], [16]. Thereby proxies can reduce the load on the origin server and decrease response times.

The concept of HTTP caching enables web proxies to store copies of previously fetched content. When a client requests the same resource (e.g. a static web page), the proxy can serve it from the cache instead of sending another request to the origin server, which improves speed and efficiency. Caching mechanisms can be controlled by HTTP headers like `Cache-Control` and `Expires`, which can for example configure if you want your data to be cached or not [17]. Since caching eliminates the need for repetitive requests to the server, it significantly improves loading times and bandwidth usage [3].

2.4. Content Delivery Networks

A Content Delivery Network (CDN) is a network of distributed servers around the world that deliver web content to users. They deliver content based on their geographic location, the origin of the webpage and a content delivery server [18]. The main goal of a CDN is to serve content to clients fast and reliable. Oftentimes CDNs are used to deliver static content like images, stylesheets, and scripts, but also dynamic content like API responses. When requesting content from a website using a CDN, the request is redirected to the nearest CDN server (edge

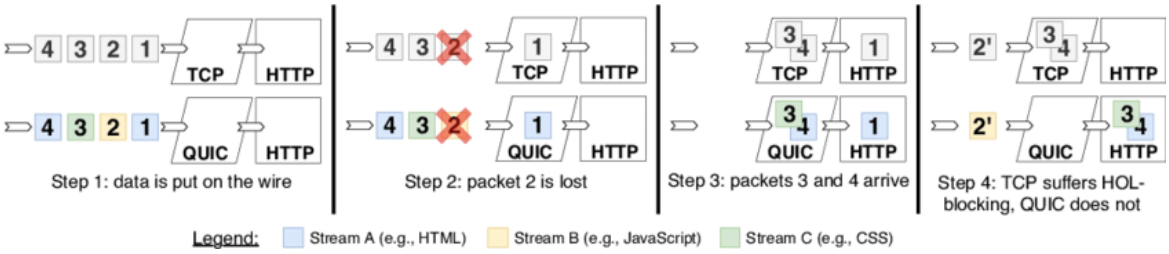

Figure 2: Head-Of-Line blocking in TCP vs QUIC [8].

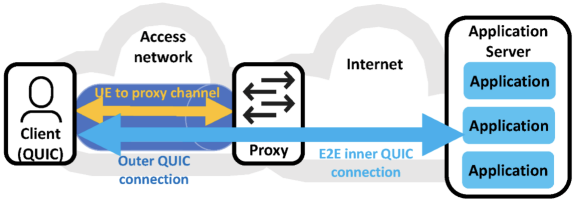

Figure 3: Proxy setup using MASQUE for QUIC-based tunneling [13], [10].

server) [19]. This redirection is typically achieved by performing AnyCast or a DNS resolution, where the domain name is mapped to the IP address that is closest to the edge server. This architecture that serves content from servers that are geographically closer to the client results in lower latency. By caching content on edge servers, CDNs can not only improve response times, but also reduce the load that the origin server has to handle [20].

3. Related Work

There are several papers that have explored the performance of QUIC's specific features. For instance, Kühlewind et al. [10] analyzed implications of MASQUE, while Cook et al. [21] examined how 0-RTT connection establishment of QUIC effects certain performance metrics in different conditions, like mobile networks. Other research provides general analyses of QUIC. Megyesi et al. [22] evaluated and compared QUIC's performance with other variants. This paper aims to provide a conceptual overview of how QUIC's features can address and improve the specific challenges of a web proxy to accelerate REST APIs.

4. Problem Analysis

This Section analyses the problems and challenges of REST APIs. These challenges come from limitations in the underlying transport protocols (most of the time TCP+TLS) or constrains of the REST paradigm.

4.1. Latency Due to Connection Establishment

REST APIs normally use HTTP over TCP, which results in a three-way handshake to establish a connection [23]. When adding TLS for encrypting data, additional two RTTs are needed for the handshake process before any data can be sent, although TLS 1.3 provides new features that allow a faster connection establishment [24]. This sequential handshake procedure results in latency, particularly bad for applications that require quick/real-time interactions. These circumstances imply an increased response time, where each connection results in a delay that affects user experience and general application performance. In scenarios with a web proxy layer, this additional layer could potentially increase latency and impact performance of the REST API.

4.2. Head-of-Line Blocking

HTTP/2 over TCP suffers from Head-of-Line Blocking due to in-order delivery of packets as illustrated in Figure 2 and explained in [21]. This phenomenon can cause delays in delivery of data, hence affecting the performance of a REST API that relies on punctual responses. In certain scenarios this phenomenon implies reduced throughput, longer wait times of API responses and especially has an impact on concurrent requests where multiple API requests are sent over a single TCP connection.

4.3. Connection Migration Challenges

REST APIs are frequently accessed by mobile clients that are likely to switch their network interfaces from time to time, such as from Wi-Fi to cellular data. Because of that the client's IP address can change during active sessions. Traditional TCP connections, identify a connection via the client's IP address and a port number, hence any change in the network interface leads to the termination of the TCP connection. If a client wants to request again from a REST API, it is necessary to establish a new connection. This process introduces additional latency due to the required TCP handshake and TLS negotiation. In scenarios with a web proxy layer or CDN is, this issue is further extended due to the additional layers of network traversal and new connections that need to be managed. Vargas et al. [25] highlights that a significant portion of API requests in CDNs consist of small, frequent JSON messages from mobile clients. The inability of TCP to handle connection migration efficiently leads to an increased latency and lower performance for these clients. This phenomenon implies reduced throughput and longer wait times for API responses.

4.4. Security Overhead

REST APIs typically use TLS to encrypt data, thereby ensuring confidentiality and integrity [23]. The overhead of encrypting and decrypting data consumes CPU resources of both clients and servers [4]. This can lead

to increased response times and reduced throughput in scenarios with many concurrent connections and high load. This overhead is particularly significant for REST APIs with frequent and small requests, where the relative cost of encryption / decryption is high compared to the actual payload size. Proxies that inspect or cache content often terminate TLS connections and act as intermediaries between client and server. This requires separate TLS handshakes for each leg, introducing additional latency and computational load.

4.5. Inefficient Multiplexing

REST APIs often utilize HTTP/2 to enhance performance through multiplexing, which allows multiple requests and responses to be sent concurrently over a single TCP connection, as described in [8]. But since HTTP/2 operates over TCP, packets must be delivered in order. This leads to head-of-line blocking, where the loss of a single packet delays all other packets until the lost packet is retransmitted. This limitation reduces the effectiveness of multiplexing and can cause delays in delivering API responses.

4.6. Caching

Cache validation mechanisms which use headers like `Cache-Control` and `ETag` often require additional RTTs for clients to confirm data freshness, increasing latency [26]. The statelessness of REST can lead to redundant data transmission (all meta informations need to be transfered) [27]. Encryption adds further complications, since the TLS connection between client and server is terminated at the proxy or CDN. This requires the proxy to decrypt and inspect the data, which can be computationally expensive and introduce additional latency.

5. Mapping QUIC Features to Problems

In this section, we map the features of QUIC introduced in Section 2 to the problems and challenges of Section 4. By mapping QUIC's features to the challenges of REST APIs served through web proxies and CDNs, we show how QUIC can enhance performance, security, and reliability.

5.1. Reducing Latency Due to Connection Establishment

QUIC combines the transport protocol and encryption layer. It reduces the number of round trips required to establish a secure connection. For new connections, QUIC can establish a secure connection in 1-RTT, and for repeat connections, it can achieve 0-RTT connection establishment, allowing data to be sent directly [1], [5]. By reducing the connection establishment latency, QUIC improves the responsiveness of REST APIs, especially in situations where short-lived connections to APIs via web proxies or CDNs are made.

5.2. Eliminating Head-of-Line Blocking

QUIC eliminates head-of-line blocking by implementing stream multiplexing at the application layer over UDP, which does not enforce in-order delivery. Streams of QUIC are independent, which means that the loss of packets in one stream does not affect packets in other streams [8]. As a result, REST APIs are able to handle multiple paralell requests more efficiently, with reduced latency and improved throughput.

5.3. Handling Connection Migration Challenges

QUIC connections are identified by a `ConnectionID` instead of a 4-tuple consisting of both IP addresses and ports [7]. This abstraction allows the connection to stay active even if the network address of the client changes. As a result, clients can seamlessly continue their sessions without the overhead of reconnecting and renegotiating cryptographic parameters. This feature is useful for improving the reliability and performance of REST APIs that are accessed by clients with mobile devices.

5.4. Addressing Security Overhead

By mandating encryption and including it into the transport layer, as described in [6], QUIC simplifies the security model. Additionally, QUIC's handshake process contributes to faster and more efficient secure connections. This improves security for REST APIs served via web proxies or CDNs. Apart from that, QUIC encrypts most of the header information, hence there are less possibilities for attacks, where attackers try to learn from the header information.

5.5. Multiplexing Efficiency

QUIC stream multiplexing allows REST APIs to handle multiple paralell requests with a single connection [9]. The independent streams prevent delays because of packet loss in one stream from affecting others, resulting in higher throughput and lower latency. This is beneficial for web proxies and CDNs that manage high volumes of simultaneous API requests.

5.6. Caching and QUIC

QUIC over HTTP/3 supports HTTP caching mechanisms. Eventhough QUIC encrypts most of the transport layer headers for security, HTTP/3 allows necessary header fields to remain accessible to intermediaries for caching [28]. Nevertheless, the encrypted nature of QUIC shows challenges for transparent caching proxies that rely on inspecting unencrypted headers.

5.7. CDN Integration with QUIC

CDNs can be improved with QUIC's features, such as connection migration. This persistence can reduce latency and improve the user experience. QUIC's handling of multiple streams and reduced connection establishment times allow CDNs to deliver content more effectively [20].

For instance, QUIC allows edge servers send content with lower latency and higher throughput, which is especially beneficial for REST APIs that rely on CDN infrastructure to reach a global audience. MASQUE and QUIC-Aware Proxying can also improve CDNs, since these functions are only partially available with the traditional TCP+TLS stack.

5.8. MASQUE and QUIC-Aware Proxying for Improved Proxy Performance

MASQUE allows clients to tunnel to proxies using QUIC and enable multiplexed traffic over a single QUIC connection [11]. This is useful for things like VPNs or proxy services that handle multiple streams within one connection. QUIC-Aware Proxying allows clients to tunnel QUIC connections or forward packets through an HTTP/3 proxy without terminating the connection at the proxy [12]. By using the extended CONNECT-UDP method over HTTP/3, the proxy can forward QUIC packets between the client and the target server transparently. The result is a reduced overhead associated with terminating and re-establishing connections at the proxy, but also the inability of CDNs to optimize things like caching.

6. Conclusion

This paper explored the conceptual potential of QUIC to accelerate REST APIs served through web proxies and CDNs. The paper identified key challenges / problems of REST APIs that mostly stem from traditional TCP-based protocols. Among these problems were latency from connection establishment, head-of-line blocking and connection migration issues. We mapped these problems to QUIC's features like including reduced handshake times, built-in encryption and connection migration.The paper demonstrated how QUIC can enhance performance, security and reliability. Adopting QUIC for REST APIs offers improvements, paving the way for faster and more secure web services that meet the evolving demands of today's internet infrastructure. Nevertheless there are still connections between QUIC, REST and web proxies that need further research, for example weither QUIC can be beneficial for certain properties of REST like uniform interfaces and statelessness.

References

[1] M. Bishop, "HTTP/3 and QUIC: Past, Present, and Future," https://www.akamai.com/blog/performance/http3-and-quic-past-present-and-future, 2021, [Online; accessed 29-August-2024].

[2] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[3] R. T. Fielding, M. Nottingham, and J. Reschke, "HTTP Caching," 2022, [Online; accessed 14-September-2024]. [Online]. Available: https://www.rfc-editor.org/info/rfc9111

[4] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The Cost of the "S" in HTTPS," 2014.

[5] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," 2017.

[6] M. Thomson and S. Turner, "Using to Secure QUIC," https://www.rfc-editor.org/info/rfc9001, 2021, [Online; accessed 29-August-2024].

[7] L. Tan, W. Su, Y. Liu, X. Gao, N. Li, and W. Zhang, "Proactive Connection Migration in QUIC," 2021.

[8] R. Marx, T. De Decker, P. Quax, and W. Lamotte, *Resource Multiplexing and Prioritization in HTTP/2 over TCP Versus HTTP/3 over QUIC*, 11 2020, pp. 96–126.

[9] T. Viernickel, A. Froemmgen, A. Rizk, B. Koldehofe, and R. Steinmetz, "Multipath QUIC: A Deployable Multipath Transport Protocol," in *2018 IEEE International Conference on Communications (ICC)*, 2018.

[10] M. Kühlewind, M. Carlander-Reuterfelt, M. Ihlar, and M. Westerlund, "Evaluation of QUIC-based MASQUE proxying," 2021.

[11] D. Schinazi, "The MASQUE Proxy," Tech. Rep., 2024. [Online]. Available: https://datatracker.ietf.org/doc/draft-schinazi-masque-proxy/04/

[12] T. Pauly, E. Rosenberg, and D. Schinazi, "QUIC-Aware Proxying Using HTTP," Tech. Rep., 2024. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-masque-quic-proxy/03/

[13] Z. Krämer, M. Kühlewind, M. Ihlar, and A. Mihály, "Cooperative performance enhancement using QUIC tunneling in 5G cellular networks," 2021.

[14] RapidAPI, "2021 State of APIs Report," 2021, accessed: Month Day, Year. [Online]. Available: https://rapidapi.com/uploads/WP_2021_Developer_Survey_So_AP_Is_Report_f3832520be.pdf

[15] A. Luotonen and K. Altis, "World-Wide Web proxies," 1994.

[16] C. BasuMallick, "What Is a Proxy Server? Working, Types, Benefits, and Challenges," 2023, [Online; accessed 14-September-2024]. [Online]. Available: https://www.spiceworks.com/tech/data-center/articles/proxy-server/

[17] "HTTP caching," https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching, [Online; accessed 14-September-2024].

[18] R. Buyya, M. Pathan, and A. Vakali, *Content Delivery Networks*. Springer Berlin Heidelberg, 2008.

[19] E. Nygren, R. Sitaraman, and J. Sun, "The Akamai Network: A Platform for High-Performance Internet Applications," 2010.

[20] Cloudflare, "What is a CDN?" https://www.cloudflare.com/de-de/learning/cdn/what-is-a-cdn/, [Online; accessed 16-September-2024].

[21] S. Cook, B. Mathieu, P. Truong, and I. Hamchaoui, "QUIC: Better for what and for whom?" in *2017 IEEE International Conference on Communications (ICC)*, 2017.

[22] P. Megyesi, Z. Krämer, and S. Molnár, "How quick is QUIC?" in *2016 IEEE International Conference on Communications (ICC)*, 2016.

[23] E. Rescorla, "RFC2818: HTTP Over TLS," 2000.

[24] ——, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8446

[25] S. Vargas, U. Goel, M. Steiner, and A. Balasubramanian, "Characterizing JSON Traffic Patterns on a CDN," in *Proceedings of the Internet Measurement Conference*, 2019.

[26] R. T. Fielding, M. Nottingham, and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Caching," RFC 7234, 2014, [Online; accessed 29-August-2024].

[27] M. Massé, *REST API Design Rulebook*. O'Reilly Media, Inc., 2012.

[28] M. Bishop, "HTTP/3," RFC 9114, 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9114