Multi-End QUIC: A Transport Protocol to Enable One-to-Many Communications

El Mehdi Makhroute
UCLouvain
Louvain-la-Neuve, Belgium
el.makhroute@uclouvain.be

Quentin De Coninck
UMONS
Mons, Belgium
quentin.deconinck@umons.ac.be

Tom Barbette
UCLouvain
Louvain-la-Neuve, Belgium
tom.barbette@uclouvain.be

Abstract

The Internet has shifted from an end-to-end paradigm to an end-to-end**s** paradigm, i.e. to serve a provider's content (e.g. a web page, an app feed, ...) the client must connect to many ends to fetch various types of resources like web documents, pictures, scripts, news feed, videos, etc... However, even the recent QUIC transport protocol failed to catch up with this change of paradigm, and the user is forced to waste many round-trip times to fetch the entire content from multiple servers. In this paper, we propose Multi-End QUIC, an extension of Multipath QUIC that enables the establishment of sub-streams directly to backend servers or third-party servers. Multi-End QUIC alleviates the need for proxies in the edge that re-encode and delay those sub-streams, that is able to bypass the relays entirely. This leads to a ~50% latency improvement in a preliminary experiment.

ACM Reference Format:
El Mehdi Makhroute, Quentin De Coninck, and Tom Barbette. 2025. Multi-End QUIC: A Transport Protocol to Enable One-to-Many Communications. In *Proceedings of the CoNEXT'25 Student workshop (CoNEXT-SW '25), December 1–4, 2025, Hong Kong, Hong Kong.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3769700.3771696

1 Introduction

The modern Internet has evolved from simple point-to-point communications to one-to-many communications: e.g. a single web page access typically requires fetching many resources from different sources. Similarly, novel workloads like distributed ML inference involves data transfer across multiple compute nodes.

Current solutions rely on intermediate components (middleboxes, proxies, or load balancers) that terminate the client connections and forward requests to backend servers, as shown in Figure 1(a). Such intermediate components become bottlenecks and introduce extra hops, latencies and inefficiency in resources utilization as many round-trip times are consumed to fetch the complete distributed content.

Efforts have been spent on developing the QUIC transport protocol [3] to address two major limitations of TCP: head-of-line blocking and the dependence on a single network path. The Multipath extension [1] exploits QUIC's design principle of decoupling a connection from a fixed network address pair to use multiple network paths within a single connection. Based on these two properties of QUIC, we propose Multi-End QUIC, an extension that enables a single client connection to use multiple paths, each mapped to a distinct endpoint with per-path transport state, while ensuring consistent transport-level behavior.

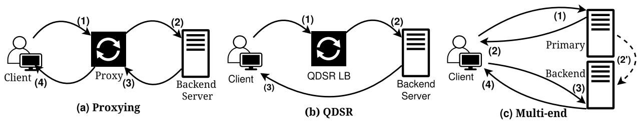

Figure 1: Proxying vs QDSR vs Multi-End

Recent works [5, 6] attempted to improve proxy-based schemes by leveraging QUIC properties. QDSR [6] uses Direct Server Return and QUIC multiplexed streams to split a QUIC connection across multiple backend servers which can send data simultaneously to the client, bypassing the load balancer (LB) (see Figure 1(b)). QDSR showed that eliminating proxy relaying for downstream traffic can boost throughput and reduce latency. Yet, all upstream traffic must pass via the LB, and thus it still creates a single point of failure and potential bottleneck. TrafficGrinder [5] uses the 0-RTT feature of QUIC to implement a load-balancer, allowing a handoff to a single secondary server. The authors show that TrafficGrinder attains near-optimal load balancing and faster completion times, compared to LeastLoaded and PowerOfTwoChoices policies, making it especially beneficial for short/ latency-sensitive flows. However, TrafficGrinder still binds each QUIC connection to a single server. In contrast, Multi-End's main advantage is its multi-server, multi-path use of a single connection with multiplexed streams. By splitting one QUIC connection across several backend servers, Multi-End effectively maintains the same logical connection while using multiple paths for each backend server (see Figure 1(c)). This "one-to-many" setup eliminates not only the redundant downlink but also uplink relay through the proxy.

2 Design And Implementation

Multi-End QUIC is built on top of the Multipath QUIC (MPQUIC), but uses it to reach different servers instead of having different paths to the same host. From the client's perspective, Multi-End QUIC is a Multipath connection. In reality, each address is a distinct server.

Servers Synchronization To make these servers appear as one, shared state is required. We designate one server as the primary and the others as backend servers. The primary server handles the handshake. When content must be fetched by the client on backend servers, the primary distributes the common connection state, which includes encryption keys, the client's connection-level flow-control window, and the set of available and used Connection

IDs. Other state can remain local to each server: e.g., the stream sending state, the application data number space, path-specific recovery/congestion control state.

Client support The advantage of our proposal is that the client only needs be able to open streams directly on a specific path. The client otherwise relies entirely on Multipath QUIC, currently in the last stages of standardization[4]. In future work, we envision the use of a specific URL, similar to MASQUE to indicate the need to fetch a resource on a particular path. Using a single connection with multiple streams avoids the inefficiencies of multiple connections, such as handshake overhead, independent congestion control, and higher resource usage.

Implementation Our prototype implementation is based on Cloudflare's open-source Quiche library with Multipath QUIC support. After the standard QUIC handshake and like MPQUIC, the client and the primary server exchange new Connection Identifiers (CIDs) for different paths. Each path corresponds to a specific backend server address. Meanwhile, the primary server sends the shared connection state to the backend server over a TCP channel established with all backend servers. After that, the client can perform path validation with a selected backend server and sends data directly to it. The backend server decrypts the received data using the shared keys, processes streams and can push back ACKs and responses directly to the client.

3 Performance Evaluation

We evaluate the performance of Multi-End QUIC using a virtual testbed as illustrated in Figure 2. Our testbed consists of two servers interconnected via a switch and one client connected through a router. All components (hosts and switches) are implemented using virtual machines. The baseline is a classical proxy-based scheme where the first server acts as a proxy, implemented using NGINX (nginx/1.16.1) with Cloudflare's Quiche patch [2] and fetches content from the backend server on the client's behalf.

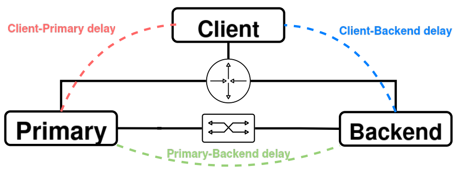

Figure 2: Testbed

In proxy mode, the end-to-end latency between the client and the backend server is determined by the sum of the Client-Primary RTT and the Primary-Backend RTT. However, Multi-End QUIC eliminates the additional relay overhead, and the latency during data transfer depends only on the Client-Backend RTT. Our experimental results in Figure 3(a) show a first scenario under identical network delays of 1ms between each host, Multi-End achieves lower latencies for different file sizes transfer. We then consider a scenario where the primary request is made to a frontend server, but the bulk of the content can be served by a closer edge server. Introducing additional virtual latency of around 10ms between the client and primary server shows an even greater advantage as reported in Figure 3(b).

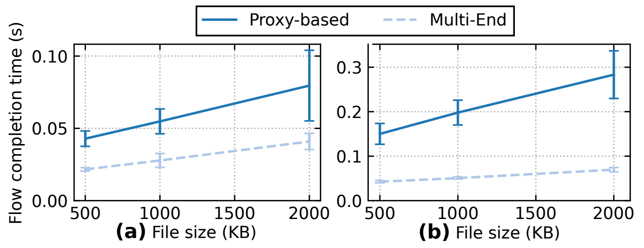

Figure 3: Latency comparison between Multi-End and proxy-based approach

We also evaluate the CPU usage improvement using Multi-End compared to the proxy-based approach. As the data is sent directly from the backend server to the client, we do not waste primary CPU cycles to forward it. Our experimental results show that with 8 concurrent clients making 100 requests for 500 KB file, the primary server in the Multi-End approach consumes fewer CPU cycles (9%) compared to the proxy-based approach (92%). At the same time, the CPU utilization of the back-end server increases from 26% to 49%, but the overall CPU consumption between both servers remains lower than that of the proxy-based approach. The increase of CPU load of the backend server can be explained by the additional encryption computational overhead. In addition, the lower primary CPU consumption enables the first server to act as a load-balancer for many backend servers.

4 Conclusion

In this paper, we introduced Multi-End QUIC, a new extension of QUIC protocol that splits a single QUIC connection across multiple servers. Our work extends QUIC from point-to-point communication to one-to-many communication. Unlike proxy-based schemes, Multi-End QUIC eliminates proxy induced delays which results in lower latency and efficient resource utilization. In future work we will evaluate more complete use cases including real-world experiments and focus more on security aspect.

Acknowledgments

This project has been funded by the Fonds National de la Recherche Scientifique - FNRS through the MIS grant 40020886 and UCLouvain FSR. This research was partly supported by the European Union and Wallonia in the framework of the program "Wallonia 2021-2027"

References

[1] Q. De Coninck and O. Bonaventure. 2017. Multipath QUIC: Design and Evaluation. In *Proceedings of the 13th international conference on emerging networking experiments and technologies* (Incheon, Republic of Korea). ACM, 160–166.
[2] Alessandro Ghedini. 2019. Experiment with HTTP/3 using NGINX and quiche. https://blog.cloudflare.com/experiment-with-http-3-using-nginx-and-quiche/. Cloudflare Blog.
[3] A. et al. Langley. 2017. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proc. of SIGCOMM'17*. ACM, 183–196.
[4] Yanmei Liu, Yunfei Ma, Quentin De Coninck, Olivier Bonaventure, Christian Huitema, and Mirja Kühlewind. 2025. *Multipath Extension for QUIC*. Internet-Draft. Internet Engineering Task Force.
[5] Robert J Shahla, Reuven Cohen, and Roy Friedman. 2024. Trafficgrinder: A 0-rtt-aware quic load balancer. In *2024 IEEE 32nd International Conference on Network Protocols (ICNP)*. IEEE, 1–11.
[6] Z. et al. Wei. 2024. QDSR: accelerating layer-7 load balancing by direct server return with QUIC. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)* (Santa Clara, CA, USA). 16 pages.