MASQUE-based Performance Enhancing Proxies

Patrick Bokelmann, Daniel Petri*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: patrick.bokelmann@tum.de, petriroc@net.in.tum.de

***Abstract*—Transport-layer protocols like TCP were originally designed to operate end-to-end, providing reliable and congestion-controlled transport. However, with the growth of the global Internet and the integration of wireless links, this characteristic has been suppressed by Performance Enhancing Proxies (PEPs) in an effort to boost transmission performance. Yet, the widespread use of these middleboxes has hindered the advancement of the TCP protocol by effectively requiring network traffic to conform to their standards. Novel transport protocols like QUIC combat this ossification with extensive header encryption, forcing the discontinuation of these transparent middleboxes. As the demand for these performance enhancements persists, this paper proposes the implementation of PEPs for QUIC connections using successive Multiplexed Application Substrate over QUIC Encryption (MASQUE) tunnels, allowing the tunneled connection to disable its congestion control mechanisms.**

***Index Terms*—transport protocols, performance-enhancing proxies, middleboxes, MASQUE, QUIC**

1. Introduction

In previous decades, the global network has departed from the transport layer's end-to-end design in favor of in-network performance enhancement functions. Operating on multiple network layers, Performance Enhancing Proxies (PEPs) have transparently altered and optimized network traffic. For instance, at the transport layer, middleboxes can filter or space out TCP acknowledgements, retransmit packets that were lost on a path segment, send localized acknowledgements, or terminate TCP connections and insert themselves as a man-in-the-middle [1]. Other use-cases include the implementation of tunneling, compression, or prioritized connection multiplexing.

These practices are particularly beneficial in wireless networks. For example, (geostationary) satellite links suffer from long round-trip times (RTTs) due to their distance from the Earth, which in turn cause long feedback loops. Congestion events on the network path after the satellite link may negatively affect the sending behaviour on the satellite link, effectively underutilizing the link [1]. In contrast, W-LAN links are prone to latency variations or packet losses caused by layer-two retransmits and handovers. By caching TCP segments, using local retransmit timers, suppressing duplicate acknowledgements, or outright TCP connection splitting, PEPs may isolate the wireless and wired path segment.

However, the rise of widespread encryption has increasingly hindered the operation of traditional PEPs. Novel transport layer protocols like QUIC authenticate the endpoint of a connection, preventing or impeding the impersonation of endpoints. With transparent middleboxes' reliance on specific protocol definitions, which significantly hinders the evolution of new protocol iterations, most of QUIC's header fields are encrypted. While this is effective in combating protocol ossification, it further restricts the deployment of transparent PEPs. Nevertheless, with a congestion control scheme similar to TCP, the protocol is still susceptible to the same issues mitigated by PEPs.

Necessitated by the departure from these transparent enhancements, there seems to be a general consensus that connection endpoints should select network functions and provide them with the necessary information that is otherwise protected through encryption.

This paper explores a novel approach to realizing similar features for QUIC connections by encapsulating packets using MASQUE, an HTTP/3 proxy. Our design proposes a sequence of MASQUE tunnels to provide reliable transport with split congestion control loops, which allows the congestion control algorithm of the inner connection to be disabled. The remainder of this work is organized as follows: Section 2 outlines the mechanisms of the QUIC protocol and provides an overview of the MASQUE proxying. Section 3 discusses related work. We then detail our proposed approach in Section 4. Finally, we conclude the paper and identify points for future work in Section 5.

2. Background

The *QUIC* protocol [2] is a novel transport protocol built as a successor to TCP and used as the underlying transport protocol for HTTP/3. Built on top of UDP, it departs from port numbers and IP addresses to connection IDs to identify a connection. These allow the connection to migrate across different network paths. Furthermore, it integrates TLS to provide a confidential and authenticated connection. This not only results in a reduced connection establishment duration in comparison to TLS on TCP, but also directly authenticates the server at the transport layer, ensuring an end-to-end connection. Additionally, it introduced two different header types: one for packets used to establish a connection, needing more header fields during that phase. The second header type is used in 1-RTT packets after the connection has been established, which reduces the overhead of unused header fields. As

 doi: 10.2313/NET-2025-11-3_12

explained in Section 1, most header fields in a QUIC packet are encrypted in an effort to combat protocol ossification. During the connection establishment process, QUIC enables the use of 0-RTT packets to transmit application data from a previous connection using an already existing cryptographic context. While the encryption of 0-RTT packets does not guarantee perfect forward secrecy, this mechanism may be exploited to accelerate the establishment of tunnelled end-to-end connections.

The contents of a QUIC packet are further organised into different frames, which are used to carry control information and application data. These frames allow QUIC to support multiplexing multiple data streams, eliminating head-of-line blocking issues commonly found in TCP. Streams are used to transmit data reliably and are subject to congestion and flow control. In addition to reliable transport, the datagram extension [3] introduces support for unreliable payloads contained in datagram frames. While subject to congestion control, they do not have to be retransmitted in case of a packet loss and are not subject to flow control limitations.

MASQUE is a group of protocols that define the tunneling of UDP, IP, or Ethernet packets over an HTTP/3 connection. The *CONNECT-UDP* method [4] allows the proxying of UDP packets inside an HTTP connection. The client connects to a MASQUE proxy and issues a CONNECT-UDP upgrade request, providing the server's IP address and port number. Packets are then encapsulated in HTTP datagrams, which directly translate to QUIC datagrams [5], and sent through the QUIC connection to the proxy. A packet the proxy receives is then decapsulated and forwarded to the server. UDP packets with a QUIC packet payload can be sent in a MASQUE connection, tunneling the end-to-end connection, as shown in Figure 1.

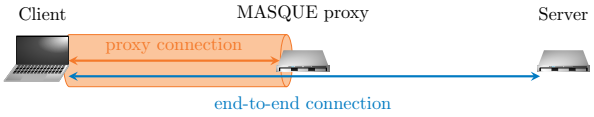

Figure 1: MASQUE Proxy [6]

Should the underlying transport protocol not support unreliable data transmission, HTTP datagrams can instead be enclosed in HTTP Capsules [5], which are then transmitted reliably in a QUIC stream. This mechanism also enables support for legacy protocols like TCP, thus maintaining backward compatibility with previous HTTP versions such as HTTP/1.1 or HTTP/2. HTTP Capsules can be re-encoded into HTTP Datagrams in transit. However, this is discouraged in [5] to avoid disturbances in the packet flow. While MASQUE was originally conceived as a measure to improve user privacy, Krämer et al. [7] voiced a proposal for usage in a performance-enhancing context, using reliable transmissions to counteract the effects of an unreliable local link.

3. Related Work

Kosek et al. [8] utilized QUIC's connection migration capabilities in combination with reworked cryptographic mechanisms to split QUIC connections at a proxy while maintaining an encrypted payload. The authors derive additional keying material during the handshake to introduce an additional layer of encryption for the application data beyond the built-in QUIC transport layer encryption. This enables the entire connection state, except for the additionally derived keys, to be shared with a middlebox while maintaining the application data's end-to-end cryptographic protections. To split an existing connection at a so-called SMAQ middlebox, the client opens an additional QUIC connection to the proxy, instructs it to act as a SMAQ proxy, and shares the connection state. The proxy then duplicates the connection state to gain a separate state for each endpoint. Next, it sends a PING to both the client and server using the new connection states, thereby initiating a connection migration. Posing as the other endpoint for both the server and client, the proxy is thus able to split the connection and insert itself in the middle while forwarding all traffic between the endpoints. Before sending application data, the endpoints encrypt it using the additional keying material. This largely maintains end-to-end privacy, integrity, and authenticity, despite sharing the QUIC connection keys with a third party. This design notably resembles TCP connection splitting with an underlying TLS connection.

Yuan et al. [9] employ acknowledgements sent by a proxy to simulate the behaviour of split congestion control loops at the client. The proxy computes packet identifiers of encrypted packets, which it then bundles into acknowledgements sent to a connection endpoint. Leveraging these early acknowledgements provided by the middlebox, the packet's sender can determine the path segments the packet has successfully traversed and adjust its congestion control algorithm accordingly. The authors observe that the sum of the congestion windows of split TCP connections using the QUBIC congestion control algorithm should equal the congestion window of a single, end-to-end congestion control loop. If a loss occurs on a given path segment in the split scenario, only the congestion window of that path would be decreased. Thus, the sum only decreases proportionally to the path segment's share of the congestion window. To emulate this behaviour in an end-to-end congestion control loop with access to out-of-band acknowledgements over a path segment, the congestion window is decreased proportionally to the path segment's share of the overall path.

4. Design

To achieve split congestion control loops, we establish a sequence of MASQUE tunnels for the QUIC connection over the entirety of the network path, as depicted in Figure 2. Using HTTP capsules enables endpoints to delegate the responsibility for end-to-end congestion and flow control to the proxy connections on each path segment. As each packet is tunneled using streams, it is transmitted reliably to the next proxy. Furthermore, the path segment to the (next) proxy is congestion-controlled by the connection to the proxy, effectively realizing split congestion

 doi: 10.2313/NET-2025-11-3_12

control loops. Note that this design relies on the server either being able to accept a MASQUE connection or to be able to receive incoming connections through a MASQUE tunnel as proposed in [10]. Furthermore, middleboxes must not convert HTTP capsules into QUIC datagrams, as this would break end-to-end connection reliability. While endpoints can disable their inner congestion control mechanisms, they should still adhere to the flow control limitations. Furthermore, acknowledgements should still be processed in case a packet loss occurs at the server in transit from the MASQUE tunnel connection to the end-to-end connection. The frequency of the acknowledgements, and the retransmit timers can be set higher than usual, and acknowledgements can be bundled together.

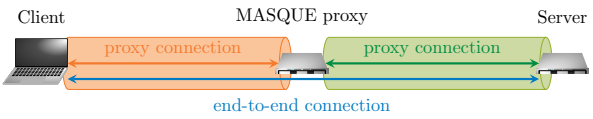

Figure 2: End-to-end tunnel with chained MASQUE connections

4.1. Connection Establishment

The connection establishment procedure is depicted in Figure 3. We assume that no previous cryptographic context is available from previous connections, i.e., no 0-RTT packets can be used. First, the client establishes the connection to the proxy. Then, it upgrades the HTTP connection and includes the initial end-to-end handshake packet encapsulated in an HTTP capsule. Upon receiving the Connect request, the proxy extracts the server address and port and initiates a connection to the server. After issuing a Connect request to the server and establishing another MASQUE tunnel, the proxy forwards the initial end-to-end handshake packet to the server. Finally, the server responds with the end-to-end handshake packet, which is tunneled through both MASQUE connections using HTTP capsules. The connection is established as soon as the client receives the server handshake, and data can be exchanged end-to-end.

4.2. Performance Overhead

A conventional QUIC connection requires one RTT to establish the cryptographic keys to send protected data. Assuming the MASQUE proxy is on the direct network path, establishing the connection through a series of MASQUE tunnels requires an additional handshake for each path segment. Because the HTTP upgrade token containing the server destination address is application data, it can only be transmitted once the handshake is complete. Thus, establishing the tunnels can not be parallelized. The tunnel for the next path segment can only start to be established after the handshake for the previous tunnel has been completed, and the first 1-RTT packet has been sent. Assuming that the first initial packet of the end-to-end connection is already carried in the first 1-RTT packet of each MASQUE connection, it reaches the server after 1.5 consecutive RTTs on each path segment. With the MASQUE tunnels fully established after the first end-to-end packet, the server's handshake response does not experience similar delays. In total, this setup requires at least two end-to-end RTTs to establish the end-to-end connection. Note that these calculations do not account for potential path asymmetries and processing delays in the proxies, and only serve to provide an understanding of the minimal latency overhead. Furthermore, 0-RTT packets, or using already-established MASQUE tunnels, have the potential to reduce this overhead significantly.

The latency of an established connection may be further affected by the internal stream buffers of a proxy. In particular, buffering stream data may lead to repeatedly dis- and reassembling packets in the proxy, resulting in a processing overhead. Additionally, Kühlewind et al. [6] demonstrated that internal stream scheduling algorithms may lead to an increase in packet losses for a proxied connection if the MASQUE tunnel is shared with multiple end-to-end connections.

Importantly, encapsulating QUIC packets in an HTTP connection incurs an overhead in the number of bits needed to transmit a packet. This decreases the available bandwidth, as more bytes of the MTU are needed to send the additional QUIC and HTTP headers, leaving less space for the payload in each packet. When using HTTP capsules to tunnel QUIC packets, the bit overhead, including the end-to-end header, is approximately 10 % for a packet size of 1440 B, whereas the overhead for a packet without tunneling and a packet size of 1380 B is approximately 3 % [6]. It should be noted that the overhead when using a MASQUE tunnel may change in the future, with efforts to introduce the capability to forward packets instead of tunneling [11], reducing the overhead induced by the additional headers of the outer QUIC connection.

4.3. Comparison to Existing Approaches

By migrating the existing connection to the proxy, Kosek et al. [8] avoid the tunneling overhead incurred by additional headers of the outer connection, as discussed in Section 4.2. However, with packets' contents consisting largely of application data, their approach should exhibit similar processing overheads at the endpoints and middleboxes. Both approaches doubly encrypt the application data as well as rebuild the outer encryption layer at the proxy. However, our proposal does not necessitate the connection to be established end-to-end first and allows established tunnels to be used by parallel connections, reducing the connection establishment latency. Furthermore, it avoids having to rework QUIC's cryptographic handshake and exposing privacy-sensitive connection details to a third party.

In contrast, Yuan et al. [9] limit the role of the proxy to providing additional information to an endpoint in the connection. As the proxied packet flow is only observed but not modified by the middlebox, the additional information necessitates a communication side-channel. Additionally, this requires the proxy to operate on-path.Furthermore, the endpoint receiving the acknowledgements needs to implement a congestion control algorithm that can capitalize on them. Even then, such algorithms will only be able to

 doi: 10.2313/NET-2025-11-3_12

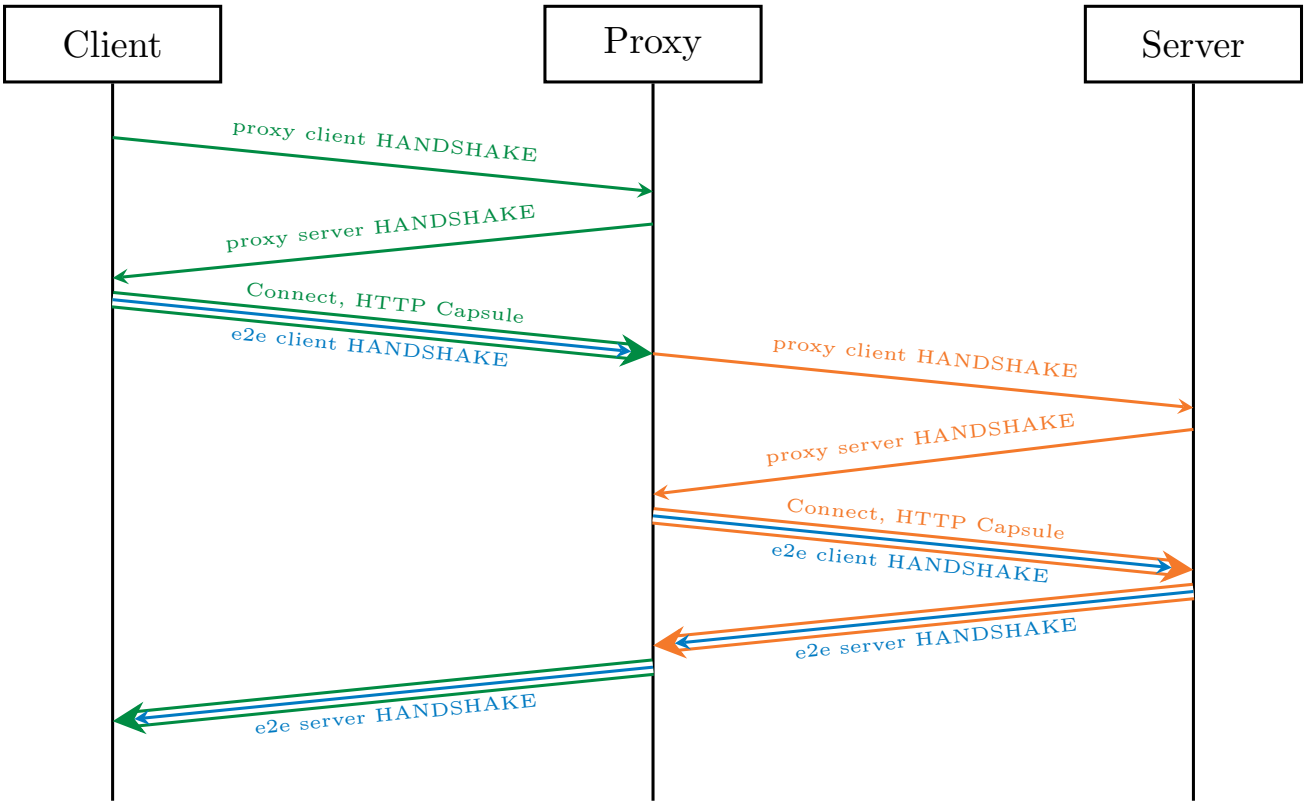

Figure 3: Connection Establishment

approximate the behavior of split congestion control loops. Unless endpoints also exchange packet identifiers with a middlebox that buffers sent packets, packets will always have to be retransmitted end-to-end in case of a loss.Such designs have not been investigated by academia as of the writing of this paper. However, unlike the solutions proposed in this paper and by Kosek et al., operating such a proxy does not require support from both endpoints. Finally, determining whether calculating and distributing packet identifiers or de- and reencrypting packets requires more computational resources remains an open question.

5. Conclusion and Future Work

In this paper, we motivated the continued relevance of Performance Enhancing Proxies for post-TCP transport protocols like QUIC and revisited the general mechanisms of the QUIC protocol and MASQUE proxies. Our main contribution is putting forward a proposal to realise PEPs with split connection control loops for QUIC connections. We further discussed the performance overheads of the solution and compared alternative approaches.

Future work could empirically examine the presented approach and evaluate its performance in comparison with the alternative approaches discussed in Section 3. Moreover, finding alternative solutions to establish a reliable and congestion-controlled tunnel between the last middlebox and the server could eliminate a possible obstacle for adopting the proposed scheme.

References

[1] J. Griner, J. Border, M. Kojo, Z. D. Shelby, and G. Montenegro, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," RFC 3135, Jun. 2001. [Online]. Available: https://www.rfc-editor.org/info/rfc3135

[2] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9000

[3] T. Pauly, E. Kinnear, and D. Schinazi, "An Unreliable Datagram Extension to QUIC," RFC 9221, Mar. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9221

[4] D. Schinazi, "Proxying UDP in HTTP," RFC 9298, Aug. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9298

[5] D. Schinazi and L. Pardue, "HTTP Datagrams and the Capsule Protocol," RFC 9297, Aug. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9297

[6] M. Kühlewind, M. Carlander-Reuterfelt, M. Ihlar, and M. Westerlund, "Evaluation of QUIC-based MASQUE proxying," in *Proceedings of the 2021 Workshop on Evolution, Performance and Interoperability of QUIC*, ser. EPIQ '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 29–34. [Online]. Available: https://doi.org/10.1145/3488660.3493806

[7] Z. Krämer, M. Kühlewind, M. Ihlar, and A. Mihály, "Cooperative performance enhancement using quic tunneling in 5g cellular networks," in *Proceedings of the 2021 Applied Networking Research Workshop*, ser. ANRW '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 49–51. [Online]. Available: https://doi.org/10.1145/3472305.3472320

[8] M. Kosek, B. Spies, and J. Ott, "Secure middlebox-assisted quic," in *2023 IFIP Networking Conference (IFIP Networking)*, 2023, pp. 1–9.

[9] G. Yuan, M. Sotoudeh, D. K. Zhang, M. Welzl, D. Mazières, and K. Winstein, "Sidekick: In-Network assistance for secure End-to-End transport protocols," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. Santa Clara, CA: USENIX Association, Apr. 2024, pp. 1813–1830. [Online]. Available: https://www.usenix.org/conference/nsdi24/presentation/yuan

[10] Y. Rosomakho, “Reverse HTTP CONNECT for TCP and UDP,” Internet Engineering Task Force, Internet-Draft draft-rosomakho-masque-reverse-connect-00, Apr. 2025, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-rosomakho-masque-reverse-connect/00/

[11] T. Pauly, E. Rosenberg, and D. Schinazi, “QUIC-Aware Proxying Using HTTP,” Internet Engineering Task Force, Internet-Draft draft-ietf-masque-quic-proxy-05, Mar. 2025, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-masque-quic-proxy/05/

 doi: 10.2313/NET-2025-11-3_12