Cascades of Nested Acknowledgments in Multi-Hop MASQUE

Kathrin Elmenhorst
kelmenhorst@uos.de
RWTH Aachen University
Ericsson Research
Osnabrück University

Mirja Kühlewind
mirja.kuehlewind@ericsson.com
Ericsson Research

Ike Kunze
kunze@comsys.rwth-aachen.de
RWTH Aachen University

Constantin Sander
sander@comsys.rwth-aachen.de
RWTH Aachen University

Klaus Wehrle
wehrle@comsys.rwth-aachen.de
RWTH Aachen University

ABSTRACT

MASQUE proxying is a mechanism for tunneling traffic over QUIC, which can help obstruct IP-based tracking when used in a nested multi-hop fashion as, e.g., done with Apple's Private Relay. Yet, multi-hop MASQUE tunneling incurs an encapsulation overhead and we find further performance issues: because tunneled ACKs become ACK-eliciting, nested MASQUE tunnels can trigger a cascade of unnecessary ACKs of ACKs that amplifies with each hop. We theoretically derive that the number of tunneled ACKs increases exponentially with the number of proxies, but that the growth factor can be small when ACK aggregation is used. Our testbed measurements reveal that practical ACK handling flattens the worst-case cascade, but the ACK byte overhead still reaches up to 13 % when using four proxies. To reduce this impact, we finally evaluate two mitigation approaches which decrease overhead with little to no performance impact.

CCS CONCEPTS

• **Networks** → **Network performance analysis**.

KEYWORDS

MASQUE, QUIC, HTTP, Proxy, Multi-hop, Nested ACK

ACM Reference Format:
Kathrin Elmenhorst, Mirja Kühlewind, Ike Kunze, Constantin Sander, and Klaus Wehrle. 2025. Cascades of Nested Acknowledgments in Multi-Hop MASQUE. In *Applied Networking Research Workshop (ANRW 25), July 22, 2025, Madrid, Spain.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3744200.3744773

ANRW 25, July 22, 2025, Madrid, Spain

ACM ISBN 979-8-4007-2009-3/25/07
https://doi.org/10.1145/3744200.3744773

1 INTRODUCTION

Proxy-based privacy tools, such as VPNs [17] or Tor [19], see increasing use as they address the growing privacy awareness of users. Yet, VPNs usually only operate with a single proxy hop, i.e., provide limited privacy [18], while the multi-hop-by-design Tor has performance issues [2]. In contrast, the MASQUE [16, 21] standardization initiative enables secure multi-hop systems with high performance. For this, MASQUE proxies traffic over QUIC [9] which features built-in encryption, fast connection setup, and stream-based multiplexing. First major MASQUE deployments demonstrate its relevance: Apple's "Private Relay" [3] proxies all Safari traffic with MASQUE, and a proposal suggests the same for Google Chrome [6]. These deployments use two-proxy setups to ensure that no relay can see both end-to-end (e2e) IP addresses. Increasing the number of proxies and decentralizing their operation would further enhance privacy because all proxies need to collude to link sender and receiver IP addresses.

In light of this potential, we aim to understand the performance implications of multi-hop MASQUE deployments. MASQUE uses QUIC DATAGRAM frames, a special *unreliable* yet *ACK-eliciting* frame type, i.e., the receiver must respond with an ACK frame within the specified maximum ACK delay [9]. While packets that only contain ACK frames are not ACK-eliciting, they also get encapsulated in QUIC datagrams when tunneled through MASQUE, thereby becoming (unnecessarily) ACK-eliciting for the next-hop proxy. This effect cascades at each additional hop such that increasing the number of proxies to enhance privacy increasingly worsens the impact of this phenomenon. As this overhead has not yet been studied, in this paper we focus on its extent and performance impact. Our contributions are as follows:

- We formalize the impact of cascading ACKs in multi-hop MASQUE with variable ACK ratio, finding that ACKs increase exponentially with the number of proxies.
- We evaluate the practical impact of cascading ACKs for a simple scenario in a Docker-based testbed with two

MASQUE implementations, showing that while the stacks' ACK handling reduces the worst-case impact, ACKs represent a substantial part of MASQUE protocol overhead.
- To reduce this overhead, we a) use ACK aggregation by configuring different ACK delays as defined by QUIC, and b) propose a novel non-ACK-eliciting DATAGRAM frame, and demonstrate in our testbed that these mitigations can reduce ACK overhead.

2 BACKGROUND: QUIC-BASED MASQUE

MASQUE's connect-udp [21] method allows proxying UDP datagrams through QUIC, as shown in Fig. 1. The client instructs the MASQUE proxy to open a UDP socket to a remote UDP target [21]. Thereafter, the QUIC connection between client and proxy acts as a tunnel for carrying e2e UDP payload which is forwarded by the proxy. During tunneling, e2e messages are encapsulated inside QUIC packets using DATAGRAM frames which are not retransmitted when lost [14, 22]. This form of unreliable transfer avoids known performance issues due to nested retransmissions [7, 23]. The QUIC-aware extension [15] provides a performance optimization called *forwarded mode*, which allows proxies to omit the tunnel encapsulation and re-encryption in QUIC-over-QUIC scenarios by directly forwarding packets between incoming and outgoing UDP ports. Because e2e packets are merely forwarded without dedicated QUIC tunnels, forwarded mode entirely avoids the overhead discussed in the remainder of this paper. Apple's Private Relay uses this optimization on the first hop, however, forwarded mode might not fit every use case, as it exposes clients to additional deanonymization risks [15].

Multi-hop use of MASQUE. As MASQUE can tunnel UDP payloads, it can also tunnel the UDP-based QUIC and, thus, also MASQUE traffic, enabling the creation of proxy chains for multi-hop tunneling. Such proxy chains help hiding e2e connection information from participating proxies as any proxy in the chain at most sees one e2e host. However, when chaining MASQUE connections, the overheads accumulate: every time the UDP payload is encapsulated in DATAGRAM frames and put inside an encrypted QUIC packet, a minimum overhead of at least 30 B is created that scales linearly with the number of proxies. Additionally, QUIC DATAGRAM frames must be acknowledged which triggers additional ACKs in multi-hop scenarios, as we detail next.

3 ANALYZING ACKS-OF-ACKS

Multi-hop MASQUE proxying encapsulates tunneled e2e packets in several nested QUIC DATAGRAM frames as we illustrate for $n = 3$ proxies in the first line of Fig. 2. Each DATAGRAM frame needs to be acknowledged on every respective QUIC connection which means that there are n "direct" ACKs triggered by a single e2e data packet. In addition

Figure 1: MASQUE's connect-udp method.

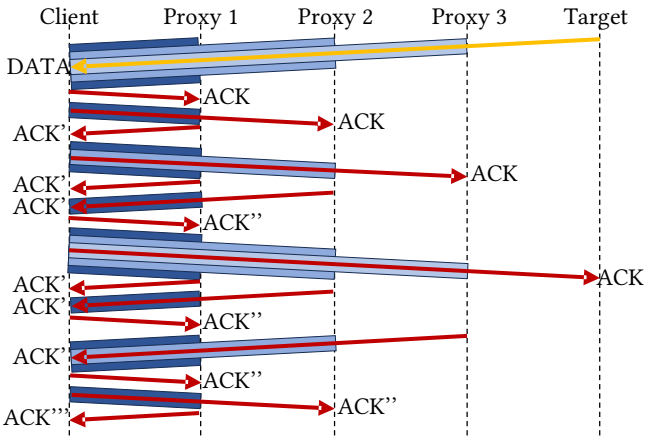

Figure 2: Data on multi-hop MASQUE connections triggers direct ACKs and ACKs-of-ACKs (ACK'-ACK''')

to these ACKs, we observe that multi-hop MASQUE produces additional ACK'-ACK'''. This phenomenon is triggered by MASQUE encapsulating ACK frames in DATAGRAM frames that are themselves ACK-eliciting. Thus, encapsulating non-ACK-eliciting QUIC packets makes them ACK-eliciting, and ACKs-of-ACKs occur, causing cascades of ACKs.

Formalizing ACK overhead. We will now count the cascading MASQUE ACKs triggered by a single e2e data packet, and formalize this overhead for a variable ACK ratio r, i.e., sending an ACK frame for every r ack-eliciting packets. In particular, we define the worst-case by assuming that each ACK frame is sent in a separate QUIC packet.

We start with the innermost e2e connection that transmits $\frac{1}{r}$ ACK for each original DATA packet. The outer tunnel connection needs to acknowledge both the encapsulated DATA (direct ACK) as well as – in reverse direction – the encapsulated ACK (indirect ACK), which leads to $\frac{1}{r} \cdot (1 + \frac{1}{r})$ additional ACKs. The next tunnel layer again adds $\frac{1}{r}$ direct ACK, and $\frac{1}{r}$ indirect ACKs for each inner ACK. This pattern repeats for every additional tunnel, so we can model the number of ACKs on all layered connections as a recursive sequence with $a_r(0) = \frac{1}{r}$ and $a_r(n) = a_r(n-1) + \frac{1}{r} \cdot (1 + a_r(n-1))$ which can be resolved to $\hat{f}_r(n) = \frac{1}{r} \cdot \sum_{i=0}^{n} (1 + \frac{1}{r})^i$. Finally, we subtract the $\frac{1}{r}$ innermost e2e ACK as it is not considered MASQUE overhead, so that

$$f_r(n) = \frac{1}{r} \cdot \sum_{i=1}^{n} (1 + \frac{1}{r})^i.$$

Without ACK aggregation, i.e., $r = 1$, we get $f_1(n) = \sum_{i=1}^{n} 2^i$.

Thus, MASQUE ACKs increase exponentially with the number of proxies. Yet, ACK aggregation reduces the exponential base (< 2) so that the growth rate is slowed down.

MASQUE tunnel endpoints cannot reliably avoid sending ACKs-of-ACKs without additional aids since encrypted QUIC packets do not reveal whether they contain ACKs. Thus, reducing unintentional ACKs-of-ACKs requires measures on the QUIC or MASQUE level which we explore next.

4 MITIGATING ACK OVERHEAD

The large theoretical impact of ACKs-of-ACKs stems from (i) *instantly* acknowledging (ii) *non-ACK-eliciting packets becoming ack-eliciting*. Thus, we study two mechanisms that reduce ACK overhead by (i) *delaying* or (ii) *avoiding* ACKs.

4.1 Delayed ACKs (DACKs)

We first leverage an established QUIC-level ACK reduction technique, namely delayed ACKs (DACKs), showing how the QUIC configuration of MASQUE endpoints impacts MASQUE overhead. DACKs allow delaying the sending of ACKs until they can be piggybacked with other data or a timeout occurs. RFC 9000 [9] defines a default "max_ack_delay" of 25 ms for QUIC, which can be adjusted during the handshake. We focus on delaying ACKs on the *MASQUE connections* with the aim to reduce the number of ACK packets between client and proxies. Since tunneled ACK packets trigger further ACKs, as described above, the reduction of ACK packets on an inner MASQUE connection directly reduces the number of ACKs on the outer MASQUE connection. This allows to interrupt and reduce the ACK cascades without modifying the e2e connection. ACK feedback should still be sent at least once per round-trip time (RTT) such that congestion control can correctly estimate the connection's state [8].

4.2 Non-ACK-eliciting datagram (No ACK)

The crucial ACKs-of-ACKs occur because non-ACK-eliciting packets becoming ACK-eliciting when encapsulated in DATAGRAM frames. To avoid this, we propose introducing a new type of DATAGRAM frames which is *non-ACK-eliciting*. Using non-ACK-eliciting datagrams for tunneling should abolish ACKs-of-ACKs and decrease ACKs in general, as the tunnels mainly carry DATAGRAM frames after setup. One challenge is that the congestion control of the proxies relies on frequent ACK feedback, i.e., sending no ACKs would starve the tunnels. Thus, using non-ack-eliciting datagrams requires to disable congestion control on the MASQUE tunnels. This is allowed by the MASQUE specification as long as the e2e connection is congestion controlled [21].

Another aspect is that traffic is often server-driven while clients mainly send ACKs such that ACK-only packets are much more likely to be sent by the client. Optimizing for this scenario, we experiment with a unidirectional mode which only uses non-ACK-eliciting datagrams when sending in the client-to-proxy direction, and thus only disables congestion control at the MASQUE client. We evaluate our two mitigation strategies in a practical testbed as we detail next.

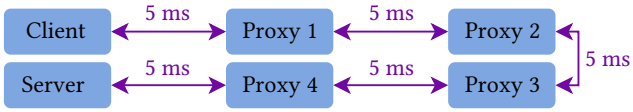

Figure 3: Docker-based testbed setup for 4 proxies.

5 TESTBED MEASUREMENTS

We assess the practical impact of cascading tunneled ACKs in a Docker-based testbed as shown in Fig. 3. In particular, we spawn multi-hop MASQUE connections to tunnel a QUIC connection between a client and a server, using separate containers for isolating the individual QUIC endpoints and proxies. Our test cases are designed to compare a diverse range of proxy configurations across a focused set of network scenarios: Specifically, we deploy 0-4 intermediate proxies, and vary their ACK strategies as described below. Using Linux tc, we enforce two different bandwidths by limiting the client's link to 10 or 50 Mbps, size the bottleneck buffer to 1.5×BDP, and configure a fixed e2e base delay of 25 ms: starting at the client, each of the first $n-1$ hops has a 5 ms delay; the last hop fills up the difference to 25 ms. Packet loss is not considered. For each test case, we run 50 measurements of the client downloading a 2 MB file. During each measurement, we capture traffic using tcpdump and collect qlog [12, 13] files from every endpoint and proxy.

5.1 QUIC Stacks

We implement multi-hop MASQUE client and proxies using two stacks with MASQUE support, aioquic [1] (Python) and Chromium QUIC [5] (C++). Both stacks use Cubic congestion control. For the e2e server, we always use Chromium.

Implementation of mitigations. While Chromium by default uses an ACK delay of 0.25×RTT, aioquic barely "delays" by the smallest possible interval of the internal timer, i.e., 1 ms. To implement varying DACKs, we make the ACK delay per connection configurable via command line parameters. Since ACK delays during slow start can negatively impact throughput, the client starts delaying after receiving 100 packets. To enforce even higher delays, we relax Chromium's ACK constraints by a) allowing to exceed the recommended limit of 25 ms [9], b) allowing to delay even if piggybacking ACKs is possible, and c) allowing to acknowledge more than 2 packets per ACK. For our non-ACK-eliciting datagram variant, we mark DATAGRAM frames as non-ACK-eliciting and disable congestion control on the proxies, while the e2e connection remains congestion-controlled.

5.2 Metrics

Besides evaluating established performance metrics, e.g., completion time, we define two focused overhead metrics for multi-hop MASQUE which are zero if no proxies are used:

Total MASQUE overhead. From pcap traces, we deduce the total MASQUE (byte) overhead by subtracting the number of e2e bytes from the total number of transmitted bytes T[1], and then dividing by T.

MASQUE ACK overhead. To focus on the overhead incurred by cascading ACKs, we define the MASQUE ACK overhead as the percentage of transmitted bytes that are contributed by *MASQUE-induced ACKs*. This explicitly excludes e2e ACKs as they do not constitute overhead. To this end, we calculate the sum of all "ACK-only" packets (see below), add the frame sizes of additional piggybacked ACKs, subtract the e2e connection's ACK bytes, and finally divide by T.

Detecting "ACK-only" packets, i.e., tunnel packets only containing ACK or DATAGRAM frames on every encapsulation layer, *requires knowledge about the (encapsulated) frame types of a given tunnel packet.* Instead of assuming access to key material of all connections to inspect the full content of *nested* packets in pcap, we developed a simple packet matching algorithm based on qlog files. Based on the known per-packet tunneling overhead and the recorded packet sizes as well as arrival orders, we can match an "outer" packet to the "inner" packet(s) starting with the first proxy qlog and going hop-wise down to the e2e qlog.

5.3 Limitations

Our methodology is designed to provide a first view on multi-hop MASQUE ACK overhead and compare different ACK overhead mitigation strategies in a simple scenario. As such, our evaluation in this paper does not intent to represent a realistic Internet scenario, e.g. with cross traffic or lossy links, but a controlled setup where we focus on those parameters that impact ACK aggregation: the Bandwidth Delay Product (BDP) which is varied by changing the link capacity, and the implementations' ACK handling. Note that link loss does not significantly impact MASQUE ACK overhead as DATAGRAM frames are not retransmitted. Still, the generalizability of our results is limited by the stable latency profile used in the testbed. Further, since ACK algorithms vary between QUIC stacks [4], our focus on two stacks likely does not cover the entire behavior spectrum of QUIC implementations.

6 RESULTS

In this section, we present our results on the practical impact of ACK cascades and evaluate the effectiveness of our mitigation approaches.

[1]Since all traffic needs to pass through the first hop, we deduce transmitted bytes from the first hop's traces.

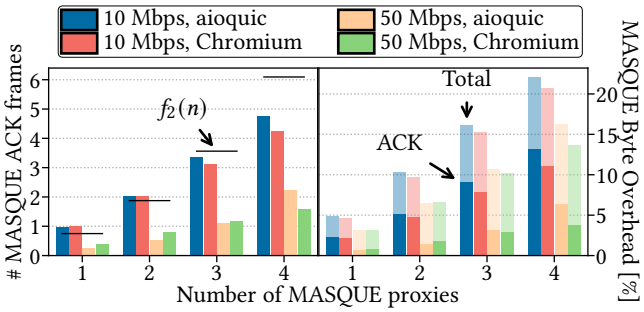

Figure 4: MASQUE ACK overhead described as number of frames per e2e data packet (left), and bytes (right).

6.1 ACK Frames and Byte Overhead

We first focus on the effective impact of cascading ACKs in a *naïve* test case where we configure a minimal ACK delay of 1 ms (aioquic's default) on the tunnel connections and do not apply any mitigation technique. This impact is presented by quantifying the number of MASQUE-generated ACKs and their byte overhead. Fig. 4 (left) depicts the median number of non-e2e ACK frames for each e2e data packet for the different stacks and bandwidth scenarios, and shows the theoretical overhead $f_2(n)$, i.e., using an ACK ratio of 2 as recommended by QUIC [9]. In the 10 Mbps scenario, the number of MASQUE ACKs resembles but does not perfectly fit $f_2(n)$ which is because the effective ACK ratio resulting from the 1 ms delay is not consistently 2, and because the theoretical overhead does not account for the case where multiple frames are carried by the same QUIC packet. Strikingly, the number of ACKs is much lower in the higher-bandwidth scenario, e.g., when using four proxies: while MASQUE tunneling produces 4.2-4.8 ACK frames per e2e data packet in the 10 Mbps scenario, we only observe 1.6-2.2 ACK frames for 50 Mbps. This difference occurs because the higher bandwidth leads to more ACK-eliciting packets arriving within the 1 ms ACK delay such that the ACK ratio is higher and more frames can be coalesced into the same QUIC packet.

This bandwidth-dependent ACK aggregation also affects the ACK *byte* overhead, as shown in Fig. 4 (right): At 10 Mbps, ACKs can make up more than 13 % of all transmitted bytes, which corresponds to 60 % of the total overhead (e.g., for four aioquic proxies), while the impact is roughly halved at 50 Mbps. In contrast, the non-ACK-related overhead is not affected by the bandwidth and only increases with the number of proxies, i.e., the number of encapsulations.

Finally, we observe that Chromium shows lower ACK byte overhead than aioquic. We attribute this observation to Chromium's ACK packetization, which is strongly optimized to piggyback ACK frames with other frames [5] such that these frames share the encapsulation overhead.

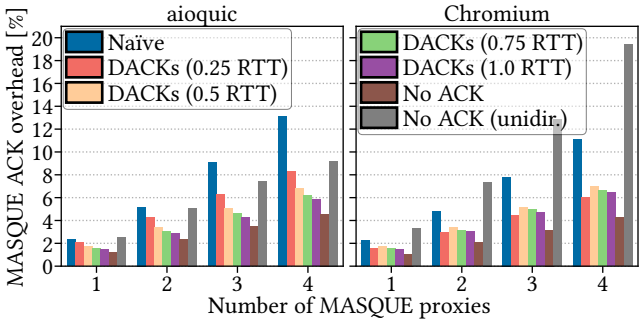

Figure 5: MASQUE ACK overhead for naïve ACKs and mitigation schemes.

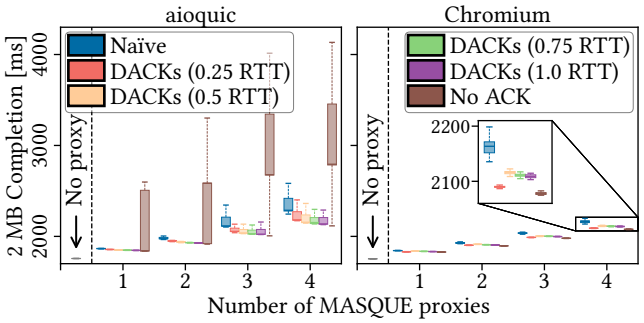

Figure 6: Download completion times for naïve ACKs and mitigation schemes.

6.2 ACK Overhead Mitigation Approaches

Having seen that the naïve ACK aggregation with a 1 ms delay is already effective in reducing the worst-case exponential growth of ACK overhead, we next study our proposed mitigation schemes that aim to further decrease overhead. Specifically, we compare the overhead of the naïve variant to the non-ACK-eliciting datagrams variant (No ACK) and to DACKs with different delays while focusing on the 10 Mbps case where the overhead was highest before. Fig. 5 shows the measured MASQUE ACK overhead as defined in Sec. 5.2, while Fig. 6 shows the download completion times of all variants for aioquic (left) and Chromium (right).

Non-ACK-eliciting datagrams. As can be seen in Fig. 5, No ACK is most aggressive in reducing the overhead as it disables ACKs of DATAGRAM frames altogether such that there are almost no ACK frames on the actual MASQUE connections. The only ACKs transmitted are e2e ACKs, and due to their encapsulation there is still ACK overhead, namely around 1 % for one proxy with linearly increasing overheads to around 4.5 % for the four-proxy case (both stacks).

Unexpectedly, the unidirectional variant fails to reduce ACK overhead and, in the case of Chromium, even results in an increase. We trace this observation to the client – which is now only sending non-ACK-eliciting packets – no longer receiving regular feedback. This leads to ever-growing ACK ranges [9] sent by the client because no ACK frame is ever acknowledged. Closer investigation shows that the aioquic client compensates this by sending a PING frame for every 8th ACK packet in order to actively trigger RTT feedback. Such PING frames are then causing the same cascading effects as before. Chromium does not send PING frames which leads to lengthy ACK ranges for most of the connection's duration. Consequently, even though there are less ACK-only packets, the average ACK-only packet size explodes to up to 400 B, leading to a 19 % ACK byte overhead when using four proxies. Due to these unintended effects, we disregard the unidirectional variant from further evaluation.

In terms of performance, our bidirectional No ACK option improves completion times by up to 81 ms with respect to the naïve case in Chromium as shown in Fig. 6. For aioquic, however, it leads to strongly increased time variance and median values. This is explained by single packet losses during the encapsulated MASQUE handshake which defer the connection setup by up to 800 ms. The losses are likely caused by the lack of pacing due to the disabled congestion control in aioquic (cf. Sec. 5.1).

Varying ACK delays. Our initial analysis in Sec. 6.1 indicated that delaying ACKs by 1 ms, the default for aioquic, already reduces overhead. We next evaluate the impact of varying the ACK delays from 0.25 to 1×RTT, considering that congestion control normally requires ACK feedback at least once per RTT.

As can be seen in Fig. 5, in aioquic, a relative ACK delay of 0.25×RTT reduces the ACK overhead for three and four proxies by 3 to 5 percent points w.r.t. the naïve approach. Further increasing the delay to 1×RTT reduces the overhead by another 2 percent points, i.e., it more than halves the overhead of the naïve approach. Chromium's ACK byte overhead is similarly reduced by 0.5-1×RTT DACKs, however, we observe the lowest overhead for an ACK delay of 0.25×RTT, indicating that higher ACK delays may decrease the number of ACK frames but not necessarily the number of ACK *packets*. In particular, ACK *packetization* largely impacts the effective byte overhead since a QUIC packet header consumes more bytes than a standard ACK frame.

Performance-wise, all DACK configurations improve the download times compared to the naïve case as shown in Fig. 6. We observe the biggest difference for four proxies: Chromium completion time improves by 73 ms (0.25×RTT delay), and aioquic's even by 150 ms (1×RTT delay), which corresponds to goodput increases from 7.0 to 7.5 Mbps, and 7.4 to 7.7 Mbps respectively. Packet loss rates (not shown) are slightly increasing with larger delays compared to the naïve 1 ms delay as endpoints are more likely to (falsely)

label packets as lost if the corresponding ACK was delayed. However, the differences to the naïve case are at most 2.6 percent points with four proxies and, as seen above, there is no negative impact on transmission time.

Takeaway. *The extent of MASQUE ACK overhead depends on implementation factors, namely the configured local ACK delay and ACK packetization strategy. When using a time-based ACK delay, e.g.,* 1 ms*, faster transmissions yield less ACK overhead. Delayed ACKs as well as non-ACK-eliciting datagrams can significantly reduce the ACK overhead of multi-hop MASQUE. Non-ACK-eliciting datagrams improve overhead by up to* 65 %*, yet require standard modification. Delayed ACKs can nearly approach this efficiency and are already supported by QUIC.*

7 DISCUSSION

Having seen the theoretical impacts and practical implications of cascading ACKs, we next discuss our results w.r.t. the deployment of multi-hop MASQUE and in comparison to one public deployment, namely Apple Private Relay (PR).

Deployment Considerations From the deployment perspective, our mitigation approaches work without changing the MASQUE-oblivious e2e traffic, because they are only applied to the *tunnel* QUIC connections between the client and proxies. However, the No ACK scheme - which disables proxy congestion control - should only be deployed if the e2e traffic is congestion controlled. While this assumption cannot be made for all use cases and is difficult to verify [11], it is reasonable for web browsing - which is the primary use case for multi-hop MASQUE today (see Apple's PR [3]). Whereas No ACK needs support by both MASQUE client and proxy, DACKs can be applied on individual MASQUE endpoints only. The most effective place to apply ACK aggregation is the first hop since most ACKs are generated on the outermost tunnel (see Sec. 3). Thus, it might be worthwhile to focus ACK reduction on this connection.

Real World Impact Our main experiments focus on measurements in a local testbed due to the limited deployment of multi-hop MASQUE in the wild. However, we also investigated the behavior of PR, the one known dual-proxy MASQUE architecture, using a PR-enabled iCloud account and MacOS Sonoma to download a 1 MB file from a self-hosted web server through PR. Though using a 1 Gbps Internet connection, the small download size induced a median throughput close to the 10 Mbps Docker scenario. Evaluating the pcap traces extracted at the client and server, as we have no access to the PR infrastructure itself, we found a total byte overhead of 7.2 % and an estimated ACK overhead of 4.1 %. Based on these overhead values – which are lower than in the 2-proxy Docker setup – and the observed packet sizes, we suspect that PR uses forwarded mode (cf. Sec. 2) on the first proxy which essentially produces a single-proxy ACK overhead because there is only one tunnel.

8 RELATED WORK

To the best of our knowledge, there is no published work on generic multi-hop MASQUE with related work focusing on single-hop MASQUE and PR. Kühlewind et al. [10], e.g., investigate the performance of single-hop MASQUE settings by comparing different MASQUE modes and analyzing the impact of nested QUIC connections and congestion control. They find that larger packet sizes reduce packet overhead and transfer times, and that MASQUE's unreliable tunneling avoids problems with nested congestion control that occur when nesting reliable connections. Additional studies investigate the dual-hop MASQUE architecture PR: Sattler et al. [20] map out PR's topology with respect to proxy locations. Trevisan et al. [24] conduct active performance measurements of PR and find median speed impairments of up to 87 % and 63 % for download and upload throughput when using PR. Finally, Zohaib et al. [25] investigate the effectiveness of traffic analysis attacks against PR and find that PR can be deanonymized using attacks that also work against Tor.

9 CONCLUSION

This paper identifies and analyzes the phenomenon of cascading ACKs on nested QUIC connections. Focusing on multi-hop MASQUE as a use case of nested QUIC, we show that this cascading effect is caused by ACKs becoming ACK-eliciting after their encapsulation. Our theoretical analysis shows that ACK frames increase exponentially with the number of proxies, while the growth factor depends on the ACK ratio. Controlled multi-hop MASQUE measurements of two QUIC stacks reveal that the effective overhead amounts to up to 13 % when using four proxies and that stack-dependent ACK aggregation flattens the increase rates. By comparing the naïve case to dedicated mitigation schemes, we demonstrate that delayed ACKs and non-ACK-eliciting datagrams can reduce ACK overhead with little to no negative impact on the performance of multi-hop MASQUE.

In future work, our measurements could be extended to evaluate multi-hop MASQUE overhead under more diverse and realistic network conditions and load scenarios, considering additional metrics, e.g., CPU and memory cost. With respect to mitigation, it would be interesting to investigate the effect of ACK delays on delay-based congestion control algorithms. Another promising angle is analyzing the impact of tunneled ACKs on privacy: large numbers of small ACK packets could create a distinct traffic pattern that might make multi-hop MASQUE vulnerable to trivial protocol analysis attacks. Overall, the large-scale adoption of multi-hop MASQUE, e.g., in Apple Private Relay, calls for a closer investigation of its various performance and privacy implications.

Ethics. This work does not raise any ethical issues.

REFERENCES

[1] aiortc. 2024. aioquic. https://github.com/aiortc/aioquic.

[2] Mashael Alsabah and Ian Goldberg. 2016. Performance and Security Improvements for Tor: A Survey. *ACM Computing Surveys (CSUR)* 49, 2 (2016). https://doi.org/10.1145/2946802

[3] Apple. 2021. iCloud Private Relay Overview. https://www.apple.com/privacy/docs/iCloud_Private_Relay_Overview_Dec2021.PDF.

[4] Ana Custura, Tom Jones, Raffaello Secchi, and Gorry Fairhurst. 2022. Reducing the acknowledgement frequency in IETF QUIC. *International Journal of Satellite Communications and Networking* 41, 4 (2022). https://doi.org/10.1002/sat.1466

[5] Google. 2024. QUICHE. https://github.com/google/quiche.

[6] GoogleChrome. 2024. IP Protection. https://github.com/GoogleChrome/ip-protection.

[7] Osamu Honda, Hiroyuki Ohsaki, Makoto Imase, Mika Ishizuka, and Junichi Murayama. 2005. Understanding TCP over TCP: Effects of TCP Tunneling on End-to-End Throughput and Latency. In *Performance, Quality of Service, and Control of Next-Generation Communication and Sensor Networks III*, Vol. 6011. International Society for Optics and Photonics, SPIE. https://doi.org/10.1117/12.630496

[8] Jana Iyengar and Ian Swett. 2022. *QUIC Acknowledgement Frequency*. Internet-Draft draft-ietf-quic-ack-frequency-02. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-quic-ack-frequency/02/ Work in Progress.

[9] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000. https://doi.org/10.17487/RFC9000

[10] Mirja Kühlewind, Matias Carlander-Reuterfelt, Marcus Ihlar, and Magnus Westerlund. 2021. Evaluation of QUIC-based MASQUE proxying. In *Proceedings of the 2021 Workshop on Evolution, Performance and Interoperability of QUIC (EPIQ '21)*. Association for Computing Machinery. https://doi.org/10.1145/3488660.3493806

[11] Ike Kunze, Constantin Sander, Lars Tissen, Benedikt Bode, and Klaus Wehrle. 2024. SpinTrap: Catching Speeding QUIC Flows. In *NOMS 2024-2024 IEEE Network Operations and Management Symposium*. IEEE/IFIP. https://doi.org/10.1109/NOMS59830.2024.10575719

[12] Robin Marx, Luca Niccolini, and Marten Seemann. 2022. *Main logging schema for qlog*. Internet-Draft draft-ietf-quic-qlog-main-schema-03. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-quic-qlog-main-schema/03/ Work in Progress.

[13] Robin Marx, Maxime Piraux, Peter Quax, and Wim Lamotte. 2020. Debugging QUIC and HTTP/3 with qlog and qvis. In *Proceedings of the 2020 ACM/IRTF Applied Networking Research Workshop (ANRW)*. https://doi.org/10.1145/3404868.3406663

[14] Tommy Pauly, Eric Kinnear, and David Schinazi. 2022. An Unreliable Datagram Extension to QUIC. RFC 9221. https://doi.org/10.17487/RFC9221

[15] Tommy Pauly, Eric Rosenberg, and David Schinazi. 2024. *QUIC-Aware Proxying Using HTTP*. Internet-Draft draft-ietf-masque-quic-proxy-03. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-masque-quic-proxy/03/ Work in Progress.

[16] Tommy Pauly, David Schinazi, Alex Chernyakhovsky, Mirja Kühlewind, and Magnus Westerlund. 2023. Proxying IP in HTTP. RFC 9484. https://doi.org/10.17487/RFC9484

[17] Reethika Ramesh, Leonid Evdokimov, Diwen Xue, and Roya Ensafi. 2022. VPNalyzer: Systematic Investigation of the VPN Ecosystem. In *Network and Distributed System Security*. The Internet Society. https://dx.doi.org/10.14722/ndss.2022.24285

[18] Reethika Ramesh, Anjali Vyas, and Roya Ensafi. 2023. All of them claim to be the best": multi-perspective study of VPN users and VPN providers. In *Proceedings of the 32nd USENIX Conference on Security Symposium (SEC '23)*. USENIX Association.

[19] M.G. Reed, P.F. Syverson, and D.M. Goldschlag. 1998. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications* 16, 4 (1998). https://doi.org/10.1109/49.668972

[20] Patrick Sattler, Juliane Aulbach, Johannes Zirngibl, and Georg Carle. 2022. Towards a tectonic traffic shift?: investigating Apple's new relay network. In *Proceedings of the 22nd ACM Internet Measurement Conference (IMC '22)*. ACM. http://dx.doi.org/10.1145/3517745.3561426

[21] David Schinazi. 2022. Proxying UDP in HTTP. RFC 9298. https://doi.org/10.17487/RFC9298

[22] David Schinazi and Lucas Pardue. 2022. HTTP Datagrams and the Capsule Protocol. RFC 9297. https://doi.org/10.17487/RFC9297

[23] Olaf Titz. 2001. Why TCP Over TCP Is A Bad Idea.

[24] Martino Trevisan, Idilio Drago, Paul Schmitt, and Francesco Bronzino. 2023. Measuring the Performance of iCloud Private Relay. In *Passive and Active Measurement: 24th International Conference, PAM 2023, Virtual Event, March 21–23, 2023, Proceedings*. Springer. https://doi.org/10.1007/978-3-031-28486-1_1

[25] Ali Zohaib, Jade Sheffey, and Amir Houmansadr. 2023. Investigating Traffic Analysis Attacks on Apple iCloud Private Relay. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security (ASIA CCS '23)*. Association for Computing Machinery. https://doi.org/10.1145/3579856.3595793