

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Leonard Walter 223531

**EVALUATING THE USE OF MASQUE PROXIES FOR
ACHIEVING DNS PRIVACY**

Master's Thesis

Supervisor: Shaymaa Mamdouh Khalil
MSc.

Co-supervisor: Silver Saks
MSc.

Tallinn 2024

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Leonard Walter 223531

**MASQUE VAHESERVERITE KASUTAMISE HINDAMINE DNS
PÄRINGUTE PRIVAATSUSE KAITSEKS**

Magistritöö

Juhendaja: Shaymaa Mamdouh Khalil
MSc.

Kaasjuhendaja: Silver Saks
MSc.

Tallinn 2024

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

A handwritten signature in black ink, appearing to read 'L. Walter', written in a cursive style.

Author: Leonard Walter

12.05.2024

Abstract

The Domain Name System (DNS) is a crucial part of the internet. Current internet infrastructure relies on unencrypted DNS, exposing user privacy and raising security concerns. Multiple DNS solutions have been standardized in recent years, that encrypt DNS messages to hide them from observers on the network. More recently the trend has also moved towards concealing the user's IP address to avoid being identifiable by the resolver. QUIC-based MASQUE proxies were first brought to the spotlight because of Apple's Private Relay. Here they were employed to achieve a similar anonymization of the user. This thesis explores a novel approach utilizing MASQUE proxies to achieve a balance between strong privacy and security while minimizing performance penalties.

The first part of the thesis evaluates existing encrypted DNS solutions, highlighting the inherent trade-off between enhanced privacy and security, and the associated performance impact. Following this, a novel DNS system based on MASQUE proxy technology is implemented. This system aims to maintain high levels of security and privacy while achieving better performance compared to existing solutions. The final evaluation confirms the feasibility of reducing performance costs in the novel DNS system. However, maintaining the same level of privacy and security as existing solutions proved challenging in the current prototype.

The thesis is written in English and is 82 pages long, including 8 chapters, 45 figures and 5 tables.

List of Abbreviations and Terms

ALPN Application Layer Protocol Negotiation

APNIC Asia-Pacific Network Information Centre

AS Autonomous System

CDN Content Delivery Network

CID Connection Identifiers

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

DoH DNS over HTTPS

DoQ DNS over QUIC

DoT DNS over TLS

ECH Encrypted Client Hello

E2E end-to-end

FEC Forward Error Correction

HoL Head-of-Line

HPKE Hybrid Public Key Encryption

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IETF Internet Engineering Task Force

ISP Internet Service Provider

LDNS Local DNS Resolver

MASQUE Multiplexed Application Substrate over QUIC Encryption

NAT Network Address Translation

ODoH Oblivious DNS over HTTPS

RFC Request for Comments

RTT Round-Trip Time

SNI Server Name Indication

TLS Transport Layer Security

VPN Virtual Private Network

Table of Contents

1	Introduction	10
2	Background and Related Technologies	12
2.1	DNS	12
2.2	Encrypting DNS	14
2.3	The QUIC Protocol	16
2.4	DNS over QUIC	18
2.5	Client Anonymity	19
2.6	Oblivious DNS over HTTPS	20
2.6.1	Oblivious HTTP	23
2.7	MASQUE	23
3	State of the Art and Related Work	26
3.1	DNS over QUIC	26
3.1.1	QUIC and 0-RTT	27
3.2	Oblivious DoH	28
3.3	MASQUE	29
3.3.1	MASQUE Prototype Implementations	30
3.3.2	Commercial applications of MASQUE	30
4	Research Methods	33
4.1	Phase 1 - DNS Protocol and Tool Selection	33
4.1.1	DNS protocol selection	33
4.1.2	Evaluation Tools	34
4.2	Phase 2 - Defining Evaluation Metrics	34
4.2.1	Defining Performance Metrics	35
4.2.2	Defining Privacy Metrics	36
4.2.3	Defining Security Metrics	36
4.2.4	Summarizing Metrics for the Evaluation	37
4.3	Phase 3 - Testbed Implementation	37
4.3.1	Testbed Stage 1	38
4.3.2	Testbed Stage 2	39
4.4	Phase 4 - Evaluation and comparison	39
4.4.1	Evaluating Performance	40
4.4.2	Evaluating Security and Privacy	40

5	Evaluating Existing Encrypted DNS Systems	42
5.1	Analyzing DoH	42
5.1.1	DoH Performance	43
5.1.2	DoH Security and Privacy	43
5.2	Analyzing DoQ	46
5.2.1	0-RTT and Session Resumption for DNS over QUIC	48
5.2.2	Security concerns with 0-RTT DoQ	54
5.3	Oblivious DoH evaluation	56
5.3.1	Oblivious DoH performance	57
5.3.2	Oblivious DoH privacy and security	59
6	Implementing a Novel DNS System using MASQUE and DoQ	62
6.1	Masque Proxy implementation	62
6.1.1	MASQUE operational modes	62
6.2	Benefits of a custom MASQUE proxy	64
7	Evaluating a Novel DNS System using MASQUE and DoQ	65
7.1	Evaluating MASQUE proxied DoQ performance	65
7.2	Evaluating MASQUE proxied DoQ security	67
7.3	Summary of the Evaluation	67
7.4	Comparison to Oblivious DNS over HTTPS (ODoH)	68
7.5	Future improvements	69
8	Summary	71
	References	73
	Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis	78
	Appendix 2 – Configuration Files and Code Modifications	79

List of Figures

1	Example of DNS traffic intercepted with Wireshark.	13
2	Example of DNS eavesdropping.	13
3	Simplified Oblivious DNS Proxy Stream Overview.	21
4	ODoH HTTP POST message to proxy.	22
5	ODoH HTTP POST message to resolver.	22
6	Stream Nesting for Encapsulation.	24
7	HTTP CONNECT-UDP message	25
8	Simplified Private Relay Dual Hop Architecture	31
9	dig used for DNS lookup on taltech.ee	38
10	Simplified testbed architecture for testing DNS systems.	39
11	Simplified testbed architecture for testing DNS systems with a proxy.	40
12	Wireshark Flow Graph of a DNS request over UDP.	40
13	Wireshark capture of a DNS request over UDP with important fields highlighted.	41
14	DoH lookup using Q.	42
15	Flow chart of a DoH request with TLS1.3.	43
16	Intercepted encrypted DoH request packet.	44
17	Intercepted encrypted DoH TLS Client Hello packet.	45
18	Traffic capture of a DNS over QUIC (DoQ) request to nextdns DoQ resolver.	46
19	Traffic capture of a DoQ request to nextdns DoQ resolver.	47
20	TLS Client Hello from captured DoQ request to nextdns DoQ resolver.	47
21	Traffic capture of a DoQ request to nextdns DoQ resolver.	48
22	Protocol stack used for 0-RTT testing.	49
23	Message flow for DNS over HTTP/3 with Routedns as client.	50
24	Triggering a DoH request manually.	50
25	Message flow for DNS over HTTP/3 with Chromium as client.	51
26	0-RTT flow for DNS over HTTP/3 with the updated Routedns client.	51
27	Updated setup with dnsproxy.	52
28	0-RTT flow for DoQ with the updated Routedns client and dnsproxy resolver.	53
29	0-RTT rejected by Adguards public resolver.	53
30	Intercepted 0-RTT message.	54
31	0-RTT replay attack response.	55
32	Traffic capture during 0-RTT replay on the resolver.	55

33	Simplified Message Flow for an ODoH Request.	56
34	Console Screenshot of an ODoH request.	57
35	Traffic capture of an oblivious DoH resolution with a loaded certificate, captured on the ODoH client.	58
36	Flow graph of an oblivious DoH resolution with a loaded certificate, cap- tured on the ODoH proxy.	58
37	Traffic capture of an oblivious DoH resolution captured on the ODoH proxy.	59
38	Decrypted ODoH query message on the proxy.	60
39	Traffic capture of Oblivious DoH messages between proxy and target resolver.	60
40	DNS traffic sent over a MASQUE tunnel captured on the server.	64
41	DoQ request to nextdns resolver over a MASQUE proxy.	65
42	Setup for testing oblivious DoQ using a MASQUE proxy.	66
43	Sending 0-RTT DoQ queries over a MASQUE proxy.	66
44	Traffic capture of a MASQUE tunnel establishment.	67
45	Traffic capture from the MASQUE proxy filtered to show only the for- warded DoQ traffic.	68

List of Tables

1	Defining Evaluation metrics for different DNS protocols.	37
2	Evaluation metrics for DNS over HTTPS.	45
3	Evaluation metrics for DNS over QUIC.	53
4	Evaluation metrics for Oblivious DNS over HTTPS.	61
5	Evaluation metrics for all evaluated DNS protocols.	68

1. Introduction

The DNS protocol plays a crucial role in the functioning of the internet by facilitating the resolution of domain names to their corresponding IP addresses. Instead of having to remember an IP address such as 89.58.42.177, DNS allows users to access webpages based on user-friendly names like example.com. DNS is almost as old as the internet protocol itself. In the 1980s, during the initial introduction of DNS [1], security and privacy were not considered. By default, DNS queries and responses are exchanged without encryption, and can thus be easily read and/or manipulated by third parties. A disruption can result in significant costs, and the manipulation of DNS data can serve as a starting point for attacks. In recent years, there have been multiple pushes to make DNS more secure, mostly by encrypting the DNS queries. The DNS resolver can still keep track of the user's requests.

There have been multiple approaches to encrypting DNS. During the late 2010s, a new combination of technologies was introduced that allowed tunneling DNS messages inside an HTTP stream called DNS over HTTPS (DoH). In 2022 the Internet Engineering Task Force (IETF) released an experimental Request for Comments (RFC) [2] that defines how a proxy can be added between a DoH client and a resolver, aiming to improve the client's anonymity level. The protocol is called Oblivious DNS over HTTPS (ODoH). So far research has shown that this helps with anonymization but adds significant latency to the DNS resolution [3].

Advancements in the networking sector, mainly through the introduction of the QUIC protocol, demonstrated that it is possible to achieve faster connections over the traditional TCP-based internet transport. Kosek et al. [4] identified that QUIC's performance benefits also translate to DoQ. Currently, the industry is preparing for the adoption of MASQUE proxies, for example, Apple's Private Relay, that offer advantages over conventional VPN or proxy solutions. Sengupta et al. [5] propose in their work to add a MASQUE proxy to the DoQ system to tighten the privacy properties, and to counter DNS fingerprinting.

This thesis aims to assess the current landscape of encrypted DNS protocols. Further, we will take a look at the new MASQUE proxy standard and will evaluate if it can be used to achieve a setup similar to ODoH in functionality, but using MASQUE and DoQ. Particularly, leveraging the advantages of the QUIC protocol and its latency benefits. Eventually, the goal is to combine a DoQ prototype with the recently standardized MASQUE proxy as a proof-of-concept. This demonstration aims to show the potential for a faster, more

secure, and privacy-preserving DNS than the currently available options.

The first and main research question is:

- Is it possible to build an oblivious DoQ prototype using MASQUE with the current state of technology?

If it is possible to build a testbed, further research will contain a comparison to figure out if a MASQUE proxy for DNS over QUIC (DoQ)

- can keep up in performance with other encrypted or anonymized DNS solutions?
- can provide the same level of privacy as other encrypted DNS solutions, with a special focus on comparing it to Oblivious DNS over HTTPS (ODoH)?

If the performance analysis shows better results than regular ODoH, this novel setup could provide a faster alternative for users who currently use ODoH.

For this thesis, the decision was made to split the background chapter into two. The second chapter provides an introduction to the network protocols used in this research and gives some background on currently faced issues with the existing state of the technologies. In the third chapter, existing implementations of the protocols are showcased and results from past research on encrypted DNS technologies are evaluated. The fourth chapter shows what methods were used to answer the research questions and define metrics for the evaluation. In the fifth chapter, the evaluation of existing DNS protocols takes place. The sixth chapter details the implementation process for the MASQUE-based proxy setup. Finally, in the seventh chapter, the novel DNS system is evaluated and compared to the results from chapter five. In chapter 8 the findings are summarized.

2. Background and Related Technologies

This chapter delves into the technologies and protocols that form the foundation of this thesis. Additionally, it explores the historical evolution of the internet's approach to privacy, particularly as it relates to the Domain Name System (DNS). By examining the history of DNS protocols, we can gain valuable context for understanding the challenges we face with contemporary DNS security and privacy.

2.1 DNS

The DNS is a crucial part of every connected network and is also sometimes referred to as the human bridge to the Internet [3]. It serves as a directory that helps users locate websites and services on the internet, making it easier to access online resources. DNS can be imagined as the phonebook of the Internet that helps us every time we access an online service. RFC 1035 [1] describes how the DNS can be used to translate human-readable domain names (e.g., `www.taltech.ee`) into IP addresses that are required to build a network connection between peers.

If someone wants to visit `taltech.ee` on their devices, the browser or underlying operating system automatically produces a DNS request to find the associated IP address of the TalTech infrastructure. The DNS request for `taltech.ee` is then sent to a DNS resolver. A DNS resolver maintains a substantial lookup table that allows it to correlate domain names with their respective IP addresses. If it does not know the IP, it will create a new recursive request to a DNS resolver which is higher in hierarchy, up to the root server. With the response from the recursive lookup, it can now answer the request and update its table, but this process is outside of the scope of this thesis. The focus of this work is limited to the communication between the client and its immediate peer DNS resolver.

Regular DNS requests that follow RFC 1035 [1] are predominantly transmitted using UDP (DoUDP) or in some cases also over TCP (DoTCP). Neither of these protocols offers encryption. RFC 9076 [6] provides an overview on privacy considerations with the DNS system. The RFC states that the lack of encryption enables manipulation, redirection, and surveillance.

Figure 1 shows captured DNS traffic from wiretapping and the information that can be

```

ludp-stream eq 22
No.    Time           Source            Destination       Protocol  Length  Info
... 5.591164    10.20.254.40    1.0.0.1          DNS       70      Standard query 0x3fee A taltech.ee
... 5.600091    1.0.0.1         10.20.254...    DNS       86      Standard query response 0x3fee A taltech.ee A 81.21.253.51

> Frame 462: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface \Device\NPF_{2D66E1DF-A
> Ethernet II, Src: Fortinet_9a:6e:2c (e0:23:ff:9a:6e:2c), Dst: Micro-St_48:b6:fd (04:7c:16:48:b6:fd)
> Internet Protocol Version 4, Src: 1.0.0.1, Dst: 10.20.254.40
> User Datagram Protocol, Src Port: 53, Dst Port: 59006
> Domain Name System (response)
  Transaction ID: 0x3fee
  Flags: 0x8180 Standard query response, No error
  Questions: 1
  Answer RRs: 1
  Authority RRs: 0
  Additional RRs: 0
  Queries
  > taltech.ee: type A, class IN
  Answers
  > taltech.ee: type A, class IN, addr 81.21.253.51
  [Request In: 455]
  [Time: 0.008927000 seconds]

```

Figure 1. Example of DNS traffic intercepted with Wireshark.

openly gathered from it. This information includes the client’s IP address, the DNS resolver’s address used to resolve the query, and the query itself.

Figure 2 shows how plaintext DNS reveals what websites a user is requesting and how it allows an attacker to modify the traffic. Any man-in-the-middle between a client and the DNS resolver can inspect such cleartext DNS traffic. This allows an observer not only to learn the browsing and application usage behavior but also to identify the types of devices that are in use. Together this allows creating user profiles solely based on DNS traffic [7]. Furthermore, manipulation of responses can also be common. A paper from 2011 identified how some Internet Service Providers (ISPs) manipulated DNS responses to inject advertisements [8]. It can be concluded, that solely relying on cleartext DNS poses a big threat to one’s privacy.

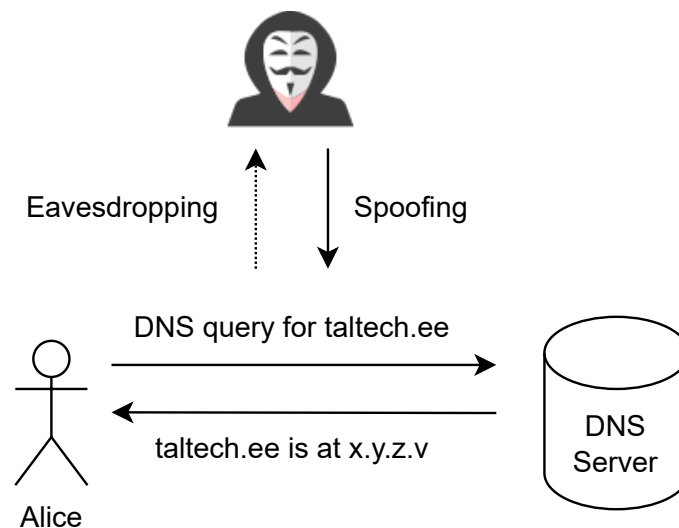


Figure 2. Example of DNS eavesdropping.

An example from recent history can be shown by how some regimes have used DNS-based censorship methods to block access to websites that do not align with their preferences. One example would be how Turkey blocked access to YouTube and Twitter in 2014 after a video of a confidential conversation leaked that caused nationwide protests [9]. In an attempt to stop the leaked video from spreading further, DNS requests for YouTube and Twitter were simply filtered.

DNS filters are also increasingly used to block sites that host copyright-infringing material. Germany for example has recently started employing DNS filters for internet censorship. German ISPs were ordered to filter DNS requests from their clients to prevent users from accessing streaming websites [10].

In the battle against such forms of "light" censorship, one strategy involves the reconfiguration of personal devices to utilize an alternative independent DNS resolver. While it is indeed possible to manually configure your device to use a specific DNS resolver, most users typically do not undertake this level of customization. Instead, by default, the DNS resolver is dynamically assigned by the ISP each time a new IP address is allocated to the user's device via the Dynamic Host Configuration Protocol (DHCP) protocol. While switching to a different DNS resolver may seem like a solution, it doesn't always guarantee complete privacy. Authoritarian regimes often implement filtering mechanisms at their exchange points, allowing them to intercept DNS requests and responses regardless of the chosen resolver.

So why are we then still mostly using plaintext DNS, one might ask? It is mostly for practical and performance reasons. DNS resolution times can have a big impact on the overall responsiveness and load times of web services [4]. Slow DNS resolutions can drastically influence the browsing experience. If DNS resolutions have a high delay, it becomes a bottleneck, causing delays in loading websites. Latency and load times are shown to have a direct impact on user's behavior and their retention rate [11]. A study found that users who were confronted with slow web searches reported feeling significantly more tensed, tired, terrible, frustrated and sluggish [12].

2.2 Encrypting DNS

It's been over ten years since Edward Snowden revealed a widespread surveillance initiative conducted by the US National Security Agency (NSA), which involved the extensive collection of data via the Internet. This and many other events have accelerated the move towards encryption. In 2014 the Internet Architecture Board released an open letter [13] to encourage privacy by design. As encryption technologies have become more prevalent

and people have grown increasingly conscious of privacy issues, the demand for encrypted DNS has surged as well. To safeguard user privacy, the networking community has recently suggested combining DNS with established encryption technologies like Transport Layer Security (TLS).

A paper from Lyu, Gharakheili, and Sivaraman [14] provides a great overview of possible attacks on DNS. Although their main focus lies on detecting malicious encrypted DNS, which is not directly related to this work, they still provide great insight into the development of DNS encryption. They summarized different attacks on plaintext DNS and highlighted what can be mitigated through the use of encryption.

DNS over TLS (DoT) was published in 2016. It was the first IETF standardized form of encrypted DNS using a TCP/TLS stack to transport DNS requests [14]. Doan, Tsareva, and Bajpai [15] conducted a study in which they analyzed DNS lookups over one week in 2021. From over 90M requests, only 0.4% were encrypted with TLS. They found that there are also negative effects on performance caused by DoT encryption. Furthermore, encrypted requests were more likely to fail and had higher latency compared to plaintext DNS. Failed requests often are caused by hardware that doesn't support the protocol and firewalls blocking traffic on the dedicated DoT port [14].

DoH was standardized in 2018 to address the issues with DoT. HTTPS is supported by almost all devices. Tunneling DNS traffic through it can mask it as HTTPS traffic, and mix DNS requests with regular web traffic. DoH has seen wider adoption in recent years and is now supported by most browsers [14]. The performance in comparison to plaintext DNS is still weaker, as it adds more delay due to the slow TCP/TLS handshake that is part of HTTPS.

Nevertheless, it is pretty evident that encrypted DNS has not yet replaced the plaintext version. A big reason for this is that most private networks stay with the default DNS resolver provided by the ISP.¹ In the referenced dataset, "sameas" represents DNS resolutions that happen in the same Autonomous System (AS), usually though the resolvers provided by the ISP. ISPs often prioritize raw DNS speed over encryption, reasoning that as long as traffic stays within their network, security risks are minimal.

Kosek et al. [4] found that both DoT and DoH are limited by the round trips required for the handshakes of the underlying TCP. Depending on the distance to the DNS resolver, additional round trips can have a drastic effect on the speed. DNS latency affects page load times and this in turn influences the user's behavior and use time on the webpage.

¹<https://stats.labs.apnic.net/rvrs>

According to the Asia-Pacific Network Information Centre (APNIC) DoT and DoH together make up about 22% of DNS traffic worldwide while in Europe and America, the number is closer to 15% as of April 2024.² It has to be added that for some countries their sample sizes are very small and it is hard to make precise measurements [16]. Some authoritarian countries such as Belarus or Myanmar show oddly high usage of encrypted DNS, this is most probably because these countries block DNS messages from leaving their country. This results in a situation where only encrypted DNS can connect to the resolvers that conduct the measurements. What supports this theory, is that traffic from such authoritarian countries is mostly DoH. DoT can be easily blocked because it uses a non-standard port. DoH on the other hand can be mistaken as regular HTTPS traffic since it uses port 443. The APNIC measurements from Western nations show that DoT is more common than DoH. For example, Germany has a 12% DoT share while DoH only accounts for 4.5%.

Another reason why encrypted DNS is not yet more common is that it often creates a headache for security professionals. This encryption, while beneficial for user privacy, allows malware to better mask its activities by hiding the accessed websites [14]. Traditional security tools that rely on inspecting the content of DNS traffic become much less effective when faced with this encryption. This is also a reason why many companies have not yet adopted it in their networks. The same also accounts for home networks, an example could be if parents want to restrict their kid's access to age-restricted content. Most tools that support such filtering are DNS-based and rely on cleartext DNS. Passive monitoring solutions will not be able to monitor domain names if encrypted DNS is used. This issue has not yet been properly addressed. On one side standardization is pushing toward strengthening security and privacy of modern DNS systems. On the other side, research is increasingly focusing on machine learning and AI-based filtering of encrypted DNS [17, 18].

2.3 The QUIC Protocol

QUIC is a modern connection-oriented transport protocol built on UDP. It was initially introduced by Google to address the inherent limitations of TCP [19]. QUIC's use cases started as mainly a transport protocol for Hypertext Transfer Protocol Secure (HTTPS) but over time it evolved to become a general-purpose transport protocol. It was standardized as such by the IETF in 2021 as RFC 9000 [20].

Since the late 2010s, the QUIC protocol made significant strides, revolutionizing the land-

²<https://stats.labs.apnic.net/edns>

scape of network protocols with its innovative approaches and substantial enhancements over TCP, as noted in [5]. This initiative stemmed from the realization that TCP, being a foundational protocol conceived in the earliest stages of the internet, has encountered significant challenges in adapting to modern networking demands.

Similar to DNS, security and privacy weren't primary concerns when early networking protocols, such as TCP, were designed in the 1980s. Despite its longstanding prominence, TCP has reached a point where augmenting its capabilities proves as a slow and tedious process that is in parts also constrained by legacy hardware. As a result, Google initiated the development of QUIC as a comprehensive substitute, that allows for swift evolution and seamless integration of novel functionalities [19].

QUIC boasts several key features that enhance security and performance. First, it integrates built-in TLS encryption, ensuring data privacy throughout the connection. Furthermore, QUIC minimizes exposed data on sent packets by encrypting most header fields (*wire image* in QUIC terminology).

Additionally, TLS 1.3, which is integrated into QUIC, supports new methods for faster connection establishment. Using Early Data in HTTP is standardized in RFC 8470[21]. These new TLS 1.3 handshakes can already be used with HTTP/2 but they are mostly associated with QUIC and HTTP/3. Compared to the conventional TCP/TLS stack that requires multiple exchanges to set up a channel and share cryptographic keys, QUIC combines communication and encryption handshakes. RFC 9001[22] describes how TLS is used in QUIC. These new connection establishment methods are also known as 0-RTT and 1-RTT. Round-Trip Time (RTT) relates to how many trips it takes until data can be transmitted.

Establishing a QUIC connection with 0-RTT allows sending encrypted data in the first message [20, 21]. This is only possible if the client knows the exchanged parameters such as keys from a previous connection. It enables clients to reconnect to servers using stored session information, avoiding the need to renegotiate session parameters and cryptographic keys. Another speedup is brought by TLS Session Resumption which allows a client and server to reuse previously established session parameters and cryptographic keys. With this feature, it is possible to quickly resume a past session without needing to exchange all parameters again during the handshake, reducing connection setup time and latency. 0-RTT and Session resumption can be combined but even if 0-RTT is not supported, many servers still support the session resumption feature.

Additional performance improvements in QUIC are made possible by the added support

for stream-aware multiplexing which mostly eliminates Head-of-Line (HoL) blocking [23]. In TCP, all data for a connection is treated as a single, ordered sequence. This can lead to HoL blocking, where a delay in receiving one part of the data stream stalls the delivery of all subsequent data in that stream, even if other parts have already arrived.

QUIC, on the other hand, allows for multiplexing data streams within a single connection. Each stream functions independently, carrying its own data and acknowledgments. This enables parallel processing of data streams. In a web page download scenario: one stream could carry the HTML content, another the images, and a third JavaScript files. If there's a delay in receiving an image (stream 2), it won't prevent the delivery and rendering of the HTML content (stream 1) or the execution of JavaScript (stream 3). This stream-aware multiplexing eliminates HoL blocking in QUIC, leading to potentially faster and more responsive connections.

QUIC also offers improvements in flexibility and pacing and adds features such as Forward Error Correction (FEC) and connection migration. This feature allows a QUIC connection to seamlessly transition between different network environments without disrupting data transfer. This is particularly valuable for mobile devices that frequently switch between cellular and Wi-Fi networks. Unlike TCP which uses the IP and Port to identify a connection, QUIC has unique Connection Identifiers (CIDs). With the CID, the client can inform the server about a network switch while maintaining the connection. The server recognizes the CID and updates its records, enabling uninterrupted data flow on the new network.

The QUIC datagram mode is another important feature of the protocol that allows disabling the reliability and retransmission features for a stream. This mode deviates from the core QUIC functionality by enabling the transmission of unreliable datagrams over an established QUIC connection. With QUIC datagrams, it is possible to serve both static elements (e.g. images, files) and unreliable content like WebRTC (Real-Time Communication) over the same QUIC connection.

2.4 DNS over QUIC

HTTP/3 and QUIC's swift rise to prominence captured much interest and prompted a movement to extend its advantages to various other technologies [24]. As a result, providing DNS services over the QUIC transport protocol represents the natural progression. Currently there exist two different protocol stacks that provide encrypted DNS functionality with QUIC. Since QUIC's main use case is HTTP/3, it was easy to adapt DoH to a protocol stack by swapping out the TCP/TLS-based HTTP/2 with a QUIC-based HTTP/3 stack.

A second protocol stack was also introduced to reduce the complexity by not using HTTP. DoQ as described in RFC 9250 [25] is the latest form of DNS encryption. As mentioned in section 2.2, encrypted versions of DNS had higher latencies due to the increased complexity in connection establishment. DoQ facilitates the exchange of encrypted DNS queries and responses over a QUIC connection to minimize latency while matching the privacy levels of DoH and DoT.

AdGuard is a company that specializes in privacy-centric DNS solutions. They were among the pioneers in offering DoQ to customers. In a blog post, they outline a list of advantages that DoQ has when compared to DoH [26]. According to them, DoQ provides the same level of encryption as DoH since QUIC also uses TLS 1.3 [25], but boasts significantly better performance. This stems from QUIC's latency-reducing features such as 0-RTT connection establishment and elimination of head-of-line blocking [26]. Furthermore, it offers advantages in the privacy area since it uses less information compared to DoH. HTTPS always comes with additional headers and cookies that leave traces and allow third parties to conduct fingerprinting.

0-RTT for DNS requests can also lead to privacy concerns as outlined in the DoQ RFC [25]. A possible scenario exists, where a man in the middle can replay a 0-RTT DoQ request, which might prompt the recursive resolver to query authoritative resolvers. By observing the outgoing traffic of the recursive resolver at certain times, adversaries could determine the queried name from the 0-RTT data. The RFC recommends disabling 0-RTT by default and it should only be enabled if the associated risks are understood.

2.5 Client Anonymity

Encrypting DNS can eliminate threats of tampering and eavesdropping by a man-in-the-middle. Then again, the DNS resolver, which is usually provided by the ISP, is still able to collect a lot of sensitive data from its clients. They see the decrypted traffic and client IP addresses which enables them to create profiles based on requests linked to IP addresses. The profiles could then allow for some degree of inference regarding the client's identity.

While encrypting DNS requests with DoH, DoQ, or DoT (see section 2.2) offers significant security benefits, it also concentrates user data with the chosen provider. This, coupled with the fact that encrypted DNS infrastructure is currently controlled by a few large players, raises concerns about privacy, competition, and potential single points of failure[5].

So far discussed technologies only covered a threat model of third parties that listen in on the communication. Recently the network community also moved towards supporting

client anonymity by updating the threat model to also include the resolver. The internet relies on the Internet Protocol for communication. Each device has a unique IP address that acts like its physical mailing address. This inherent feature allows for efficient routing of data packets but also presents a privacy concern. Every sent packet has the client's IP address on it. Monitoring traffic and analyzing the source IP addresses makes it easy to track the behavior of users.

Client anonymity in DNS is achieved by keeping the user's identity separate from the actual DNS query content. To achieve increased privacy when using the IP protocol, a common approach is to route traffic through an intermediary or proxy. When a user sends a request through a proxy, the proxy forwards the request with its own IP address, masking the user's identity from the website. This makes it appear as if the user is connecting from the proxy's location. However, it's important to note that proxy servers don't guarantee complete anonymity. The proxy still knows the client's IP address, so there is some trust involved in the proxy not sharing that information. Furthermore, the user still has to be careful with sent data, as there could be identifying information in it or attached to the metadata.

There already exist many well-established technologies that solve this problem. Virtual Private Network (VPN)'s and proxies are used by many people to conceal their identities while browsing the internet. These same technologies can also be used to achieve DNS client anonymity but often lack flexibility and are easy to detect and often blocked by governments [27].

There also exist methods to hide the query itself from the resolver[28]. Such private information retrieval techniques offer strong privacy guarantees by hiding the queried information from the server. However, this often comes at a significant cost in terms of computational overhead and increased latency. Thus it was considered out of scope for this thesis.

2.6 Oblivious DNS over HTTPS

Another new and still experimental enhancement to the DNS ecosystem is ODoH, sometimes also referred to as ODNs. Oblivious DoH has been defined in RFC 9230 [2] since June 2022. The goal of this new protocol is to further enhance the privacy of DNS by hiding the client's IP address from the DNS resolver through the use of a proxy.

Tanya Verma from Cloudflare, who is also a co-author on the Oblivious DNS RFC describes in a blog post in great detail how the technology operates [29]. They claim, that by

employing an independent proxy, the DNS resolver exclusively interacts with the proxy’s IP address, ensuring that the client’s IP address remains undisclosed. Additionally, the proxy remains uninformed of the requested domains, thereby precluding any possibility of interference or eavesdropping on DNS requests. Consequently, only the DNS resolver retains the capability to access and interpret the content of these requests. It is important to note here that the proxy and resolver have to be two independent entities, if they cooperate and share their knowledge, the benefits of using the proxy are diminished.

Schmitt, Edmundson, and Feamster [30] showed that using unencrypted DNS exposes a spectrum of information, encompassing web browsing patterns and the variety of devices present in a user’s residence. They were the first to recommend using an independent third party that proxies DNS requests to hide the client’s identity. By employing such a proxy in front of the recursive resolvers for encrypted DNS queries, they were able to show that it is possible to hide the IP addresses of the clients that initiated these queries.

ODoH builds upon DoH and extends it by adding another encryption layer to the DNS query inside the DoH stream. This new encryption of the DNS messages works with the Hybrid Public Key Encryption (HPKE) scheme defined in RFC 9180[31]. HPKE is a streamlined encryption method designed for ease of use, adaptability, and long-lasting security. It’s more lightweight than using a full end-to-end (E2E) TLS connection. Cloudflare provides a good overview of its goals, workings, and how it is used in ODoH[32].

The scheme in context with ODoH can be explained as follows: Alice in this case first needs to know the public key of the DNS resolver. For example, she could randomly choose a resolver from a big list but such a mechanism is out of scope for this thesis. Alice then encrypts the URL where she wants to do the DNS query with the DNS resolver’s public key. She then creates an HTTPS session with the proxy and adds the encrypted query, as well as the IP address of the DNS resolver to a POST request. Upon reception of the request, the proxy terminates the first encrypted HTTPS session and forwards the HPKE encrypted message over a new HTTPS session. The response follows the same pattern just in reverse. Figure 3 shows a simplified overview of an ODoH connection and the encryption layers that secure the DNS message exchange.

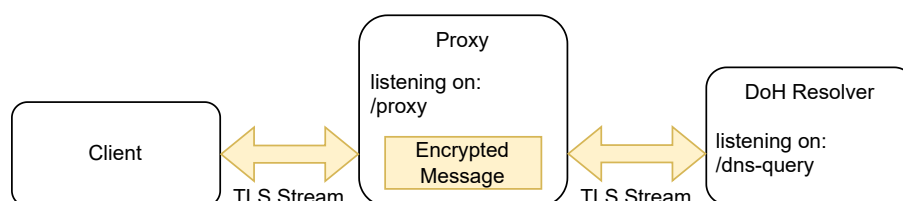


Figure 3. Simplified Oblivious DNS Proxy Stream Overview.

Both POST requests can be seen in the following Figure 4 and Figure 5. The first POST request shows the communication between the client and proxy. The second POST is the forwarded request from the proxy to the DNS resolver according to the specification [2]. The proxy is listening on **https://dnsproxy.example/proxy** for HTTP POST requests.

```
:method = POST
:scheme = https
:authority = dnsproxy.example
:path = /proxy?targethost=dnstarget.example/dns-query
accept = application/oblivious-dns-message
content-length = 106
query = dGFsdGVjaC5lZQo=
```

Figure 4. ODoH HTTP POST message to proxy.

When the proxy receives a request, it will open a new HTTPS session with the specified target resolver. The query in this case is then forwarded to **https://dnstarget.example/dns-query**. Upon reception of a request, the DNS resolver uses his private key to decrypt the query. A symmetric key is derived from the plaintext DNS query, which is subsequently used to encrypt the response.

```
:method = POST
:scheme = https
:authority = dnstarget.example
:path = /dns-query
accept = application/oblivious-dns-message
content-length = 106
query = dGFsdGVjaC5lZQo=
```

Figure 5. ODoH HTTP POST message to resolver.

Cloudflare in their paper from 2020, also conducted tests to measure the performance impact by adding a proxy to the DNS request [29]. They claim that the performance impact, that comes from the additional encryption between the client and proxy is marginal. But they also acknowledge that there is additional latency introduced by the proxy, which can range between 100 to 200ms compared to regular DoH requests.

To sum it up, with ODoH, users can increase their privacy levels compared to regular encrypted DNS by adding a proxy to the DNS message flow. The drawback is that this added complexity impacts the responsiveness and speed of the DNS resolution. Furthermore, there are centralization concerns. If the proxy and resolver cooperate or both their security is breached, an attacker can trivially link the request to the corresponding client.

Achieving the best privacy results with ODoH relies on free choice from a big pool of openly available proxies.

2.6.1 Oblivious HTTP

ODoH introduced a new form of so-called indirection of traffic. This oblivious idea has since been extended, not limiting it to DNS. Oblivious HTTP works on the same principle and allows forwarding anonymized HTTP requests. In January of 2024, the oblivious HTTP RFC [33] was released by the IETF. The Oblivious HTTP RFC standardizes a mechanism to send HTTP messages while preserving end-user privacy. The standardization of the oblivious protocols shows that there is a bigger move towards supporting client anonymity.

2.7 MASQUE

Multiplexed Application Substrate over QUIC Encryption (MASQUE) is the name of a working group at IETF. They are working on the standardization of mechanism(s) that allow configuring and concurrently running multiple proxied stream- and datagram-based flows inside an HTTP connection.

Currently, there exist a variety of different solutions that already offer anonymization through proxy forwarding, so why is MASQUE needed? Regular TCP/UDP proxies are limited to a static 5-tuple. Source and destination IP + Port and the protocol. SOCKS proxies reveal the target address in cleartext and VPN tools often require administrative privileges. Furthermore, most of the conventional proxy and VPN solutions use protocols and ports that are easy to detect and block.

The HTTP CONNECT method was already introduced in HTTP/2. An HTTP CONNECT request contains the address and port of a remote server. It is sent to a proxy and the proxy then initiates a connection to the specified remote location. All further traffic that is sent to the proxy will then be immediately forwarded to the remote server. The traffic looks like regular HTTP traffic to observers and can thus not be easily blocked. In HTTP/2 this proxy mechanism was limited to TCP traffic.

MASQUE is aiming to extend this by also allowing the proxying of datagrams. To achieve this a new HTTP method was introduced: CONNECT-UDP. The primary goal of the MASQUE mechanism(s) is to allow "configuring and concurrently running multiple proxied stream- and datagram-based flows encapsulated inside an HTTP connection" as described in the working groups charter [34]. MASQUE builds upon the HTTP CONNECT

method and its extensions, that allow for a simple proxy configuration. An overview of all MASQUE-related technologies can be found on the IETF datatracker [35].

The CONNECT-UDP method initiates an unreliable QUIC datagram connection between the client and the proxy server, which is crucial for encapsulating a wide variety of protocols [36]. The stacking of reliable and congestion-controlled protocols leads to suboptimal behavior and severely affects performance. For example, suppose a packet is lost during transmission and two reliable streams are stacked. In that case, both streams will trigger a resend of the lost packet, leading retransmission timeouts to drift apart, which can stall the entire connection. In TCP this phenomenon is often referred to as TCP Meltdown. Furthermore, doubled retransmissions send unnecessary duplicate traffic thus increasing congestion.

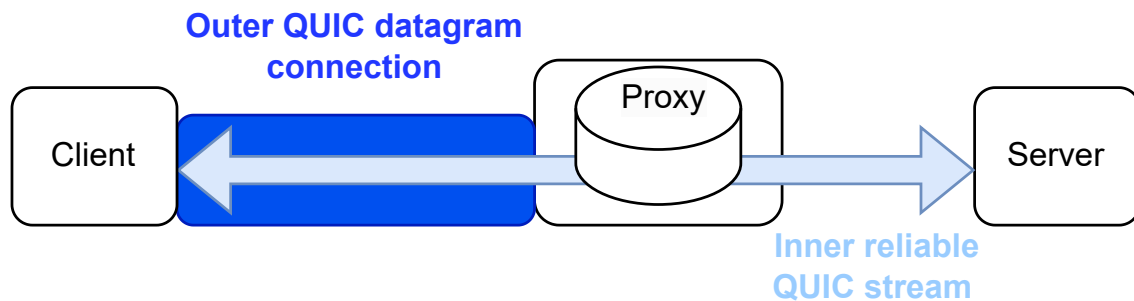


Figure 6. Stream Nesting for Encapsulation.

MASQUE can avoid such a meltdown by using the QUIC datagram mode [37] as shown in Figure 6. Here, QUIC datagram frames of the outer MASQUE stream, are sent without reliability guarantees. This means that if a QUIC sender deems a datagram frame as lost, it will not be retransmitted. MASQUE is made possible by QUIC datagrams [38] which lay the base for the CONNECT-UDP method. In Figure 6 the inner stream is marked as reliable, but this does not necessarily have to be true, as any traffic can be routed through a MASQUE proxy.

Oblivious DNS terminates the TCP and TLS sessions at the proxy and only forwards the encrypted ODoH message. In contrast, HTTP Connect-based proxies such as MASQUE establish an E2E stream between client and target. Such proxies offer more flexibility and are not limited to a specific application.

The following snippet shows the fields from a CONNECT-UDP HTTP request as specified in the RFC 9298[36] and RFC 9484, [39]. A client wishing to access a target web server at 192.0.2.6:443 using MASQUE first sends a CONNECT request to the MASQUE proxy.

Upon receiving the CONNECT request, the MASQUE proxy responds with a 200 OK

```
:method = CONNECT
:protocol = connect-udp
:scheme = https
:path = /.well-known/masque/udp/192.0.2.6/443/
:authority = example.org
capsule-protocol = ?1
```

Figure 7. HTTP CONNECT-UDP message

message if successful. It then opens a new UDP socket towards the target web server. With the connection established, the client can now send any UDP-based data, such as an HTTP/3 GET request, to the MASQUE proxy. The proxy will then forward this data to the specified target web server (192.0.2.6:443) on behalf of the client.

The MASQUE proxy has also gained some new further extensions with two new methods: connect-IP (RFC 9484[39]) and connect-ethernet (currently still an IETF draft [35]) that allow proxying of arbitrary traffic.

In section 2.6 it was already briefly mentioned that Apple uses ODoH in Private Relay. This raises the question, why are they using two separate systems for HTTP and DNS traffic, wouldn't it be possible and beneficial to send DNS traffic over the same MASQUE relay?

3. State of the Art and Related Work

This chapter aims to provide insight into the current state of available technologies and existing research in the field that are important to answer this thesis's research questions. The main focus is examining whether the required technology stack to build an oblivious version of DoQ using MASQUE exists.

3.1 DNS over QUIC

Despite the publication of the DoQ RFC [25] in 2022, widespread adoption has not yet happened. Various open-source DNS implementations offer support for DoQ and according to [privacyguides](https://www.privacyguides.org/en/dns/)¹ a few of the big DNS providers already support DoQ. Still, DoQ makes out a rather small part of all DNS requests. According to Adguard, DoQ traffic only represents about 1% of the total DNS traffic to their resolvers [40]. None of the big web browsers support it yet.

Kosek et al. [41] focused in their research on the adoption and performance of DoQ. They found more than 1000 DoQ resolvers while mapping the entire IP range with ZMAP. Kosek et al. [41] also analyzed DoQ's response times in comparison to DoT and DoH. Their paper is titled with the Question if DoQ is the "One to Rule them All". This emphasizes how much impact DoQ might have in the future. They focused on a large group of publicly available DoQ servers and repeatedly sent requests to measure the response time over an entire week. Approximately 40% of their measurements still experience significantly slower response times than anticipated. This delay is attributed to the enforcement of QUIC's traffic amplification limit and not using 0-RTT.

In a follow-up paper, Kosek et al. [4] repeated the measurements. This time with 0-RTT session resumption. It was found that this change was able to significantly boost DoQ's speed, outperforming DoT and DoH by 33%. Therefore, DoQ renders encrypted DNS notably more attractive than DoH. It only lags behind DoUDP by approximately 50%, whereas DoT and DoH exhibit increased delay of around 66% for single queries. Furthermore, for more complex websites where a multitude of domain names have to be looked up, DoQ even catches up to regular DNS over UDP. With multiple lookups, the added latency cost of encryption amortized and DoQ was measured only roughly 2% slower than DoUDP.

¹<https://www.privacyguides.org/en/dns/>

3.1.1 QUIC and 0-RTT

0-RTT is one of the key features of QUIC that allows sending encrypted data in the first message, thus reducing connection establishment time to zero. Sengupta et al. [5] did the first extensive analysis on using DoQ with 0-RTT. They analyzed cross-layer interactions of DoQ while web browsing. They evaluated multiple DNS protocols and their impact on a full web page request from the first DNS lookup up to the moment the page has finished loading. Their results showed that 0-RTT handshakes offer substantial performance benefits compared to the traditional TCP/TLS stack used in DoH. In their measurements, 0-RTT DoQ was able to half the load times in the best-case scenario.

As mentioned in section 2.4 using the current form of 0-RTT messages unfortunately also introduces some negative side effects. Encrypted data in QUIC 0-RTT packets does not have forward secrecy. This introduces the risk of potential replay attacks. An attacker can resend captured 0-RTT data to the server, potentially tricking it into performing unauthorized actions. The RFC provides a good overview [25] on privacy and security considerations with 0-RTT and session resumption.

The DoQ RFC suggests disabling 0-RTT data by default and only enabling it for users who understand the associated risks. This recommendation is further supported by research by Kosek et al. [4], who found that none of the public DoQ resolvers in 2022 supported 0-RTT.

While 0-Round Trip Time (0-RTT) in QUIC offers faster connection establishment by reusing session information, it also introduces security concerns to HTTP/3. Since the server hasn't fully validated the client yet, sending sensitive data or performing actions that can modify the server state (like POST, PUT, DELETE) with 0-RTT could be exploited by malicious actors. To mitigate this risk, 0-RTT should be restricted to safe HTTP operations like GET, HEAD, OPTIONS, and TRACE. These methods primarily retrieve information from the server without modifying its state, reducing the potential for misuse of 0-RTT and ensuring a balance between performance and security.

DNS requests can change the server state, in that they might trigger a recursive lookup, thus it is recommended to disable it by default and only enable it for clients that are aware of the associated risks. There is also research in this area to mitigate these risks mentioned above with 0-RTT. There are multiple approaches to secure 0-RTT messages. Göth et al. [42] show that Bloom-Filter Key Encapsulation Mechanisms can be used to achieve forward secrecy while keeping added cost from computational complexity to a minimum. They claim that previous attempts at securing 0-RTT communication were either too slow or not

thoroughly tested for real-world use.

3.2 Oblivious DoH

Oblivious DNS has not gained too much attention in the recent past. When the idea was still developed, around 2020, there was also a lot of research. Since then, the attention on ODoH has declined.

Singanamalla et al. [3, 29] show the privacy benefits of ODoH. They performed latency and response time measurements on the technology and compared it to other existing encrypted DNS solutions. They took wide-scale measurements and collected data from DNS resolvers on multiple continents. In their testing, they showed that ODoH can outperform a setup where DNS is tunneled over the Onion Router (TOR) network, but trails behind regular DoH when it comes to resolve times.

Kumar and Bustamante [43] acknowledges the privacy benefits of ODoH but criticizes its higher cost and increased chance of client-LDNS mismatch. Local DNS Resolver (LDNS) mismatch hurts performance by directing users to distant servers. This can happen when visiting a site hosted by a Content Delivery Network (CDN). The ODoH resolver does not know the client's IP and can thus also not make assumptions on its location. Further, he can not decide which is the closest node of the CDN and might select a long-distance and high-latency server for the response.

When it comes to software implementations and support, the available open-source ODoH implementations are currently limited, with many appearing outdated or explicitly labeled as prototypes. The DNSCrypt project [44] and their implementation supports a wide array of DNS protocols. They also have support for Oblivious DoH in their client and server, but they lack the support for ODoH relaying. The Cloudflare ODoH implementation has not been updated since the release of the corresponding paper [3] in 2020.

Apple offers ODoH support as part of their Private Relay.² Private Relay lets users surf with increased privacy by routing traffic through two separate internet relays. ODoH queries are sent encrypted through the first relay, similar to the HTTP requests but are then routed directly to the resolver. [45]

²https://www.apple.com/icloud/docs/iCloud_Private_Relay_Overview_Dec2021.pdf

3.3 MASQUE

Cloudflare provides a comprehensive overview of MASQUE's goals on their blog page [46] that is closely tied to a paper by the same authors [47]. The same engineers are also heavily involved in the standardization process and are part of the MASQUE working group. In their paper [47], they focus a lot on the performance of tunneled traffic under various circumstances. They tested reliable and unreliable transmission and evaluated the effects of nested congestion control. Furthermore, they expect that MASQUE will prove to be more resilient over time. Connections default through port 443, which for both TCP and UDP blends in well with general HTTP/3 traffic and is less susceptible to blocking than other proxying/tunneling solutions.

Since work started on MASQUE standardization, few papers have been published, assessing its benefits. In an early paper from 2021 [47], the researchers used a MASQUE setup built with aioquic³ and tested in a simulated environment. While the setup introduces overhead and requires careful stream scheduling, it offers potential performance benefits, especially for mobile networks. However, the researchers pointed out that further research is needed due to limitations in the current implementation and testing environment.

At the time of writing, work is still ongoing on MASQUE features and some features are still in the IETF draft phase. Nevertheless, MASQUE is already starting to see some adoption. Apple introduced iCloud private relay, which offers their customers an extra layer of anonymization while browsing. Other tech companies such as Google⁴, Cloudflare[48], or Fastly [49] also work on their MASQUE-based proxying services.

If this positive trend of MASQUE adoption continues, it has the potential to carve out a significant niche within the privacy tunnel technology field. MASQUE offers similar benefits to conventional VPN and proxy solutions while offering more flexibility and stealth. QUIC's performance benefits make it attractive for many applications and its share of the internet traffic is steadily increasing. Sooner or later tunneling technologies will want to adopt this as well and currently MASQUE seems very promising. Similar to a VPN, MASQUE proxies reduce the network provider's ability to observe traffic, that information is transferred to the proxy operator.

Furthermore, the growing prevalence of QUIC, with its inherent performance gains, makes it an increasingly attractive choice for various applications. It's only a matter of time before tunneling technologies embrace this evolution, and MASQUE currently stands out as a

³<https://github.com/aiortc/aioquic>

⁴<https://github.com/GoogleChrome/ip-protection>

highly promising candidate to deliver the backbone technology.

3.3.1 MASQUE Prototype Implementations

At the point of writing, most QUIC/HTTP3 implementations don't yet support the MASQUE features such as the connect-udp method. Generally, there is a low abundance of open-source implementations of the MASQUE features and not all MASQUE implementations support all features from the specification.

Google offers a MASQUE prototype that is featured in their QUICHE QUIC chromium experimental section⁵. The code can also be viewed on github⁶. It is one of the more feature-rich and up-to-date implementations. It supports connect-udp, connect-ip and connect-ethernet. Generally, it seems well-maintained and receives regular updates as the standardization progresses.

Other openly available prototypes are INVISV masque⁷, masque-go⁸ and masquerade⁹ which is written in Rust. INVISV only open-sourced their client and is lacking a server implementation. The latter two implement an early MASQUE version and have not seen updates in a long time. Work on integrating the MASQUE features to quic-go¹⁰ just started in late April of 2024. it currently does not feature a working version.

3.3.2 Commercial applications of MASQUE

Apple is one of the most prominent early MASQUE adopters as of 2024. Apple has been offering iCloud Private Relay for most of their iOS devices since late 2021. Their marketing claim says that it prevents websites and network providers from creating detailed profiles about you while the browsing experience remains fast. iCloud Private Relay is designed to enhance user privacy and security by encrypting internet traffic and routing it through two separate internet relays, also known as dual-hop architecture.

These devices are known as ingress relay and egress relay. While the network and ingress relay can access the client's IP address, the server name remains encrypted, rendering it invisible to them. This encrypted information is then transmitted from the ingress to the egress relay, which must be provided by independent infrastructure partners such as

⁵<https://www.chromium.org/quic/playing-with-quic/>

⁶<https://github.com/google/quiche/tree/main/quiche/quic/masque>

⁷<https://github.com/Invisv-Privacy/masque>

⁸<https://github.com/marten-seemann/masque-go>

⁹<https://github.com/jromwu/masquerade>

¹⁰<https://github.com/quic-go/masque-go>

Fastly, Cloudflare or Akamai. The third party then carries out traffic forwarding to the designated target server. The egress relay will only be aware that the sender is utilizing iCloud Private Relay, but it won't have access to the real client's IP address. Apple markets it as an internet privacy service that helps its users conceal their location, IP address, and DNS records ¹¹. According to Reuters[50], it is not available in Belarus, Colombia, Egypt, Kazakhstan, Saudi Arabia, South Africa, Turkmenistan, Uganda, and the Philippines [50]

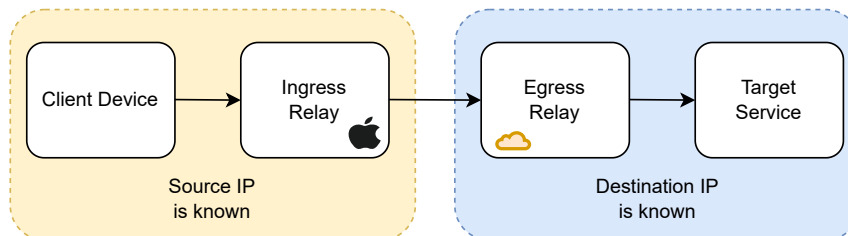


Figure 8. Simplified Private Relay Dual Hop Architecture

The connection to the ingress proxy defaults to MASQUE, with an HTTP/2 CONNECT fallback mechanism in case of QUIC failures or blockage. Notably, the handshake employs raw public keys for authentication instead of the standard TLS certificate approach [45]. Interestingly, Apple claims that for connections to websites supporting TLS or QUIC, the initial handshake messages are bundled with the proxy request, eliminating the need for separate responses from the proxies. This approach minimizes additional latency introduced by the private relay. Apple most likely achieves this by establishing a permanent MASQUE tunnel between user end devices and the private relay network.

Apple's iCloud Private Relay leverages a proprietary implementation of the MASQUE framework. While several research papers have analyzed the technology [51, 52], Apple adheres to its typical closed-source approach, leaving the inner workings largely obscure. Trevisan et al. [51] evaluated Apple's iCloud private relay and its effects on web page load times and download throughput. They found that it limits bandwidth and introduces increased latency, which negatively affects page load times. Although it has to be noted that their findings are based on limited results from tests in just three different locations. [52] Kiesel criticized the node assignment mechanism in a conference talk. Apart from switching Private Relay on or off, users can only specify a location mode. Users can select either to maintain their general location or to receive a random location that will be selected from the same country and timezone. All other variables in the system, such as the ingress and egress nodes are chosen by Apple. In his talk, he stated that users have to trust Apple to choose an independent egress node.

Cloudflare is also working on integrating MASQUE into its commercial privacy proxy

¹¹https://www.apple.com/icloud/docs/iCloud_Private_Relay_Overview_Dec2021.pdf

implementation called WARP [48]. The first beta version is supposed to be released in the second quarter of 2024. So far there is not too much known about it. INVISV is another operator that already today offers a MASQUE-based privacy relay for customers. They market their privacy relay as a better VPN alternative [53] and offer an app for Android.

4. Research Methods

This chapter outlines the methodology employed to evaluate the performance and privacy characteristics of different DNS protocols. A controlled experimental approach was chosen, utilizing a dedicated testbed to evaluate various DNS technologies.

These are the primary goals of this thesis:

- **Comparative Analysis of Established DNS Technologies:** This objective focuses on comparing the performance and privacy aspects of selected existing encrypted and anonymized DNS solutions. This analysis will highlight the capabilities and limitations of these established protocols.
- **Exploration of Oblivious DoQ Prototype:** It will be attempted to develop a prototype for an Oblivious DoQ solution utilizing the MASQUE proxy protocol. While the feasibility of building this prototype is considered promising based on current technology, its success will be determined during the final stages of the research. Regardless of the prototype's development outcome, the core value of this research lies in the comparative analysis mentioned above.

To achieve the set-out goals, four stages were defined to better structure the research process. The following sections describe the research progress in each phase.

4.1 Phase 1 - DNS Protocol and Tool Selection

The initial phase of our research focused on selecting the most suitable DNS protocols and evaluation tools for the investigation.

4.1.1 DNS protocol selection

Existing research was evaluated and the choice was made to restrict the analysis on DNS over HTTPS (DoH), DNS over QUIC (DoQ), and Oblivious DNS over HTTPS (ODoH). The selected protocols represent the latest advancements in secure DNS protocols. Focusing on these ensures that this evaluation reflects the current state-of-the-art.

By focusing on just these three protocols, it is possible to delve deeper into their specific functionalities, performance characteristics, and security properties. This allows for a more

nuanced and informative comparison.

Other candidates that are not considered are DNS over TLS and DNSCrypt. DNS over TLS was not evaluated because it has similar characteristics to DoH but is more prone to blocking. DNSCrypt was not evaluated because it lacks official standardization, potentially hindering widespread adoption.

4.1.2 Evaluation Tools

In parallel with the protocol exploration, potential tools and frameworks were identified that could aid in the research process. This involved searching relevant research literature and online resources to understand the tools used in previous studies and the methodologies employed for DNS protocol evaluation. It was found that many of the DNS client and resolver implementations used in past research were custom-built for their specific needs. Most of the time, these custom tools that were published with the associated paper, are left abandoned and become outdated. Instead, the decision was made to use popular, up-to-date DNS implementations where possible. The selection of tools was heavily influenced by the number of features they offered.

The analysis will take place on Linux systems, as these offer the best support for custom DNS clients and resolvers. The test machine that functions as a client was set up on a Debian virtual machine. Generally, most tools were installed in docker containers. This allows an easy way to distribute and host them on a server. Linux networking tools such as iptables are used to define routing rules and for isolating DNS traffic.

Traffic captures from the DNS systems is done with tcpdump and Wireshark. Both tools can listen on physical and virtual network interfaces to record data streams. The recordings are evaluated using Wireshark, which includes many helper tools that assist with the visualization and packet inspection of captured traffic.

4.2 Phase 2 - Defining Evaluation Metrics

The evaluation of the technologies is split into three segments.

- Performance metrics compare the complexity of the DNS protocols and how they impact resolution times.
- The privacy metrics aim to determine the amount of information leaked about the client from a DNS query.

- Security metrics help conceal the connection and aim to harden the protocols against blocking or manipulation.

4.2.1 Defining Performance Metrics

Extensive research and benchmarking tools exist for analyzing DNS server implementations, including CPU load, scalability, and resistance to Denial-of-Service (DoS) attacks. Common tool in this category are DNSperf¹ or DNS-Shotgun.² However, this work delves deeper into comparing the underlying protocols while minimizing the influence of external factors like hardware, software implementation, and network environment.

Time measurements of the duration for DNS lookups will not play a key role in the evaluation of this thesis. It is influenced by too many factors that are hard to control, thus it is common practice to measure on a wide scale and over a longer period to get averaged results. Furthermore, for this thesis, the testing involves many prototypes that can have different properties varying from implementation to implementation. Instead, it was decided that the focus is more on evaluating the differences in protocols and not on comparing the various available DNS implementations.

Another important decision was made to evaluate non-persistent connections. All the encrypted DNS protocols rely on connection-oriented protocols. An application can keep a connection open for a long time and reuse the same connection with the same encryption keys for multiple DNS requests. For example, Firefox operates in this way. It keeps open the TLS connection to a DoH server and sends a keepalive packet roughly every 50 seconds if no DNS requests are made. This is so the connection does not time out. Instead, it was decided to evaluate non-persistent connections to the DNS resolver and focus on the differences in connection establishment.

When analyzing the protocols, one could also consider accounting for the cryptographic complexity, for example, if there are multiple TLS sessions. However, according to Tanya Verma[29] who analyzed ODoH, the impact of TLS encryption on the latency is minimal.

Recursive lookups are out of scope and it is assumed that the resolver already possesses the answer for the requested domain. Under these conditions, the DNS lookup time is primarily influenced by three key factors:

The **number of round trips** it takes for the DNS resolution. Ideally, it only takes a

¹<https://www.dnsperf.com/>

²<https://dns-shotgun.readthedocs.io/en/stable/>

single request and response, that is also why plaintext DNS over UDP is still so common. Additional roundtrips might be necessary for certain protocols because encryption keys need to be exchanged first.

Round-Trip Time (RTT) is affected by the network's latency, which is mostly influenced by the physical distance between two computer systems. The RTT describes how long it takes for data from the client to reach the server and back.

Resolver latency. High volumes of DNS requests can slow down resolvers. This can be amplified by the hardware, software and protocol's complexity.

The RTT is largely limited by the physical properties of the network, it can only be lowered by selecting a resolver that is at a closer distance to the client. The resolver latency is what makes a fair comparison of measured latencies complicated. It is affected by a lot of different influences and is often very unpredictable.

Thus, as already mentioned above, the evaluation will only measure the number of round trips it takes to resolve a query. It is the only metric that is directly impacted by the choice of protocol.

4.2.2 Defining Privacy Metrics

A full comparison is very hard due to the large amount of properties that can affect the protocols. Furthermore, one has to take into consideration that there are many different threat models for the privacy metrics. It was decided to focus on the following questions to define the privacy metrics:

- Is the client's identity exposed to the DNS resolver?
- Is the content of the DNS query exposed to observers on the wire?
- Does the connection reveal additional metadata about the client that are unrelated to the DNS query?

4.2.3 Defining Security Metrics

Encryption plays a crucial role in safeguarding DNS queries. When data is encrypted, it becomes significantly more difficult to manipulate or tamper with the content. However, even with encryption, an adversary can still block the DNS messages if he can detect them as such. The aim is to prevent the message from being identifiable as encrypted DNS. The

following properties of encrypted traffic are commonly used for filtering traffic and should be checked:

- Is the resolvers’s hostname visible?
- Is the resolvers’s IP visible
- Is the protocol visible?

If an attacker can observe the destination IP address and hostname of the resolver in the network traffic, they could potentially block DNS requests altogether. This scenario is particularly relevant in certain countries where governments restrict outbound DNS traffic and limit citizens to using state-controlled resolvers. In such cases, even encrypted DNS traffic might be identifiable and blocked.

4.2.4 Summarizing Metrics for the Evaluation

The following table will be used to provide a broad overview of the findings for every analyzed DNS protocol. It features the metrics discussed in this section.

Category	Metric
Performance	Number of Round Trips
Privacy	Exposed Client Identity Exposed DNS Query Exposed Metadata
Security	Risk of Tampering Risk of DNS Server IP Blocking Risk of Destination Hostname Blocking Risk of Protocol Blocking

Table 1. Defining Evaluation metrics for different DNS protocols.

4.3 Phase 3 - Testbed Implementation

Building upon the chosen DNS protocols, evaluation tools, and metrics, a dedicated testbed environment was set up to facilitate the evaluation.

A public server with a static IP was rented from netcup³. This enables conducting performance measurements not limited to a simulated environment. Due to the physical distance between Estonia and the server located in Germany, the RTT was measured to be around 30 milliseconds. This is not a big issue though, as latency measurements don’t play a key role in this evaluation.

³<https://www.netcup.de/>

The server functioned as a flexible platform, allowing deployment of proxy implementations and acting as a DNS resolver in specific test scenarios. This self-hosted server also facilitated data collection through traffic captures and logs, enriching the evaluation process. The testbed was grouped into two stages. Stage one of the testbed was used for testing and evaluating DNS protocols without a proxy, while stage two added a proxy between client and resolver.

4.3.1 Testbed Stage 1

The following protocols were tested in stage 1:

- DNS over HTTPS (DoH)
- DNS over QUIC (DoQ)

The dig tool can send regular DNS over UDP requests to the local stub resolver. By default, applications rely on the system configuration to resolve domain names. On Debian-based Linux systems, the DNS resolver is configured in `/etc/resolv.conf`. Linux offers native support for DNS over UDP and TCP, readily facilitating these protocols. With minimal system configuration adjustments, DNS over TLS can also be integrated. Other DNS protocols, however, require the installation of a dedicated client.

A typical dig DNS resolution can look as follows: The output of the dig command shown in Figure 9 reveals the answer for the DNS query but also provides some additional information. In the example, it took 27ms to resolve the IP of `taltech.ee` (81.21.253.51). Here, it can be seen that Google's public DNS resolver was used as indicated by the server IP (8.8.8.8) in the response.

```
user@debian:~$ dig taltech.ee
; <<>> DiG 9.18.24-1-Debian <<>> taltech.ee
;; Got answer:
;; -->HEADER<<- opcode: QUERY, status: NOERROR, id: 6982
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;taltech.ee.                IN      A

;; ANSWER SECTION:
taltech.ee.                60      IN      A      81.21.253.51

;; Query time: 27 msec
;; SERVER: 8.8.8.8#53(8.8.8.8) (UDP)
;; WHEN: Thu Apr 18 15:50:43 EEST 2024
;; MSG SIZE rcvd: 55
```

Figure 9. dig used for DNS lookup on `taltech.ee`

A stub resolver was used on the host machine for many of the evaluated scenarios. A DNS stub resolver acts as an intermediary for applications and DNS servers. When an application needs to translate a domain name into an IP address, it sends the request to the stub resolver. The stub resolver runs as a separate process on the same machine and handles all the DNS processing. Stub resolvers can be used to encrypt communication with DNS servers for added security. Figure 10 shows the topology used for the three DNS setups that don't rely on a proxy.

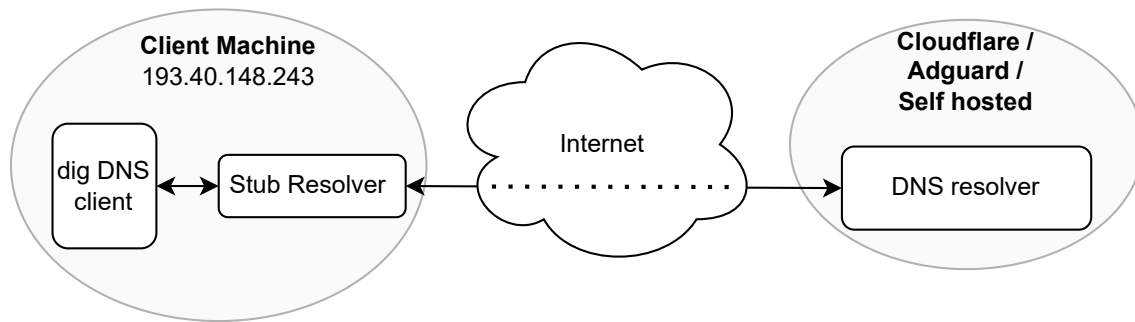


Figure 10. Simplified testbed architecture for testing DNS systems.

To generate encrypted DNS traffic, a stub resolver such as Routedns⁴ can be set up. It is an open-source DNS stub resolver and proxy implementation written in go. Routedns was set up in a docker container listening on port 5355. The dig tool allows specifying a custom DNS resolver with the following syntax: `dig @172.17.0.2 -p 5355 taltech.ee`. Here the IP address of the local routedns stub resolver's docker container is specified with the @ symbol.

4.3.2 Testbed Stage 2

To have control over the proxy and resolver functionality, a second server was added for the final testbed. A local client was set up to work with the proxy server for further measurements. The following DNS protocols rely on a proxy. They were analyzed in the testbed shown in Figure 11.

- Oblivious DNS over HTTPS (ODOH)
- DoQ over a MASQUE Proxy

4.4 Phase 4 - Evaluation and comparison

This section describes how the evaluation on the captured data from different DNS systems is done. A single test to gather traffic captures usually consists of the following steps:

⁴<https://github.com/folbricht/routedns>

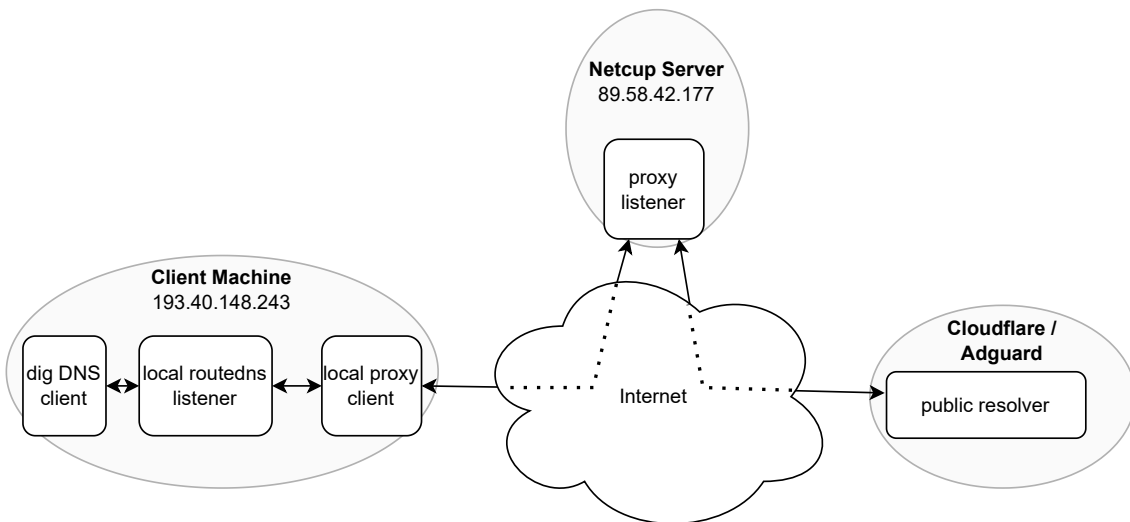


Figure 11. Simplified testbed architecture for testing DNS systems with a proxy.

Preparing the testbed and setting up Wireshark on one or more interfaces if testing includes a proxy. On the client machine, the DNS request is triggered with a DNS client, in many cases the dig tool was used. After receiving the DNS response, the traffic capture is stopped and can be analyzed with Wireshark.

4.4.1 Evaluating Performance

This section shows how the performance of different DNS protocols is evaluated based on some examples.

Wireshark flow graphs can be used to figure out the number of handshakes it takes to resolve a DNS query. Figure 12 shows that it only takes a single round trip to resolve a DNS query over UDP.

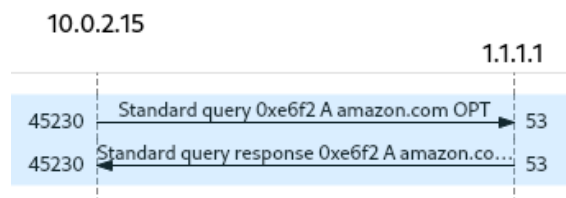


Figure 12. Wireshark Flow Graph of a DNS request over UDP.

4.4.2 Evaluating Security and Privacy

Packet capture analysis plays a big role in covering security and privacy metrics. Network traffic is captured during DNS resolution with different setups. Wireshark is used to identify exposed data. From the captures, most of the metrics can easily be identified.

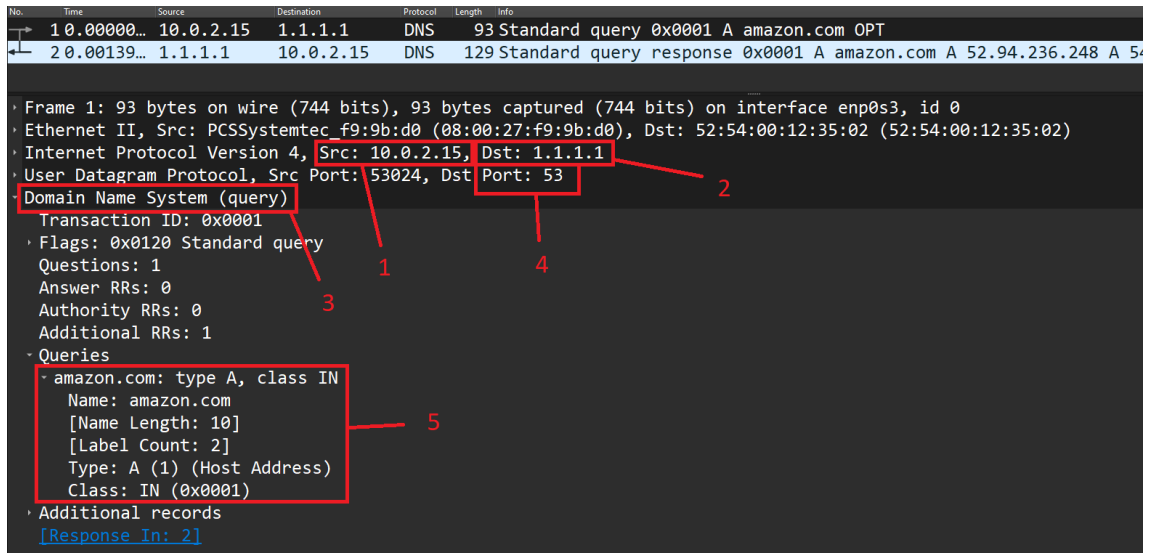


Figure 13. Wireshark capture of a DNS request over UDP with important fields highlighted.

The information from the highlighted fields in Figure 13 reveals information about the client's IP address (1) and the resolver's IP address (2). The captured traffic can easily be recognized as DNS (3), this is because Wireshark automatically analyzes the packet's contents. In this case, it is also easy to determine the protocol as port 53 (4) is used, which is the default port for DNS. The queried address and type is shown in the highlighted area (5). In this case, type A reveals that the query asks for the IPv4 address of amazon.com. There is a lot of information revealed from the DNS query, the same is also true for the DNS response, which is not shown in detailed view in this screenshot.

5. Evaluating Existing Encrypted DNS Systems

This chapter delves into the evaluation of the selected protocols based on the metrics established in section 4.2. A concise overview of the test setup configuration for each protocol is provided, followed by a comprehensive analysis presented in table format. These tables summarize the performance of each protocol according to the defined metrics, allowing for a clear and comparative assessment.

5.1 Analyzing DoH

For analyzing DoH Routedns was first used as a local stub resolver. Unfortunately, it does not support exporting the TLS secrets, so instead the Q¹ client was used. Figure 14 shows a screenshot of a name resolution with Q using Cloudflare resolvers (<https://1.1.1.1:443/dns-query>). By specifying the resolver with the @ symbol and providing an HTTPS address, Q is automatically switching to DoH.

```
root@d1a4d519a0e4:/go# q taltech.ee A @https://1.1.1.1:443/dns-query -S
taltech.ee. 43s A 81.21.253.51
Stats:
Received 54 B from https://1.1.1.1:443/dns-query in 15.3ms (14:52:46 05-02-2024 UTC)
Opcode: QUERY Status: NOERROR ID 54377: Flags: qr rd ra (1 Q 1 A 0 N 0 E)
```

Figure 14. DoH lookup using Q.

Q allows specifying a `-tls-client-key=` parameter to store TLS secrets. With Wireshark, it was then possible to load and decrypt the TLS data packets carrying the DoH request and response. Depending on the combination of client and server, different behaviors were observed. Q defaults to the HTTP GET method while the Routedns client uses the POST method for DoH queries. Both clients can be configured to use a different HTTP method.

The DNS query in the HTTP GET contains the base64 encoded domain name. In this case for taltech.ee it looks like this:

```
GET /dns-query?dns=pEcBAAABAAAAAAAAAB3RhbHRlY2gCZWUAAAEAAQ
```

After some experimentation with different public resolvers, it was decided to self-host a Routedns DoH resolver on the rented server. This way, it was possible to analyze resolver

¹<https://github.com/natesales/q>

logs and have full control over the caching mechanism to avoid reverse lookups.

5.1.1 DoH Performance

Figure 15 shows a DoH request triggered with Q to the self-hosted DoH listener with Routedns. The listener is configured to make recursive lookups but it is equipped with a cache for DNS responses. The DNS lookup took three round trips, the same behavior was also observed on other public resolvers.

The first, gray message exchange is the TCP handshake. In the second handshake, the TLS client hello message is sent. This is part of the TLS handshake and key exchange, to set up the encrypted channel. Finally, in the third handshake, the green highlighted HTTP messages indicate the DNS query and response.

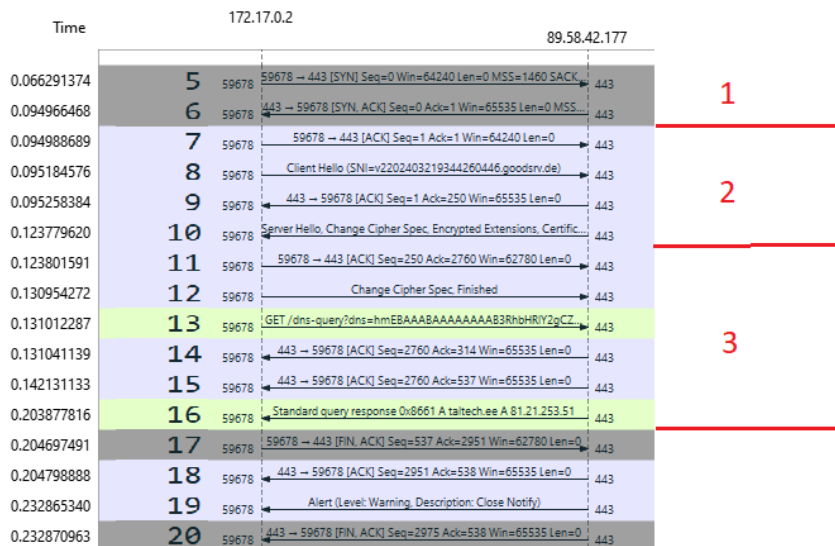


Figure 15. Flow chart of a DoH request with TLS1.3.

5.1.2 DoH Security and Privacy

The following screenshot shows the encrypted request packet with the exposed information highlighted.

The DNS request packet has packet number 13 in the capture also seen in Figure 15. Without access to the TLS secrets, an observer can only see encrypted application data. Nevertheless, it is still possible to gather a lot of information from the captured traffic. The source and target IP are visible. In this scenario, the source IP is a local IP, because the traffic was captured on the client machine’s interface. As soon as the packet leaves

No.	Time	Source	Destination	Protocol	Length	Info
5	0.06629...	172.17.0.2	89.58.42.177	TCP	74	59678 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM
6	0.09496...	89.58.42.1...	172.17.0.2	TCP	58	443 → 59678 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
7	0.09498...	172.17.0.2	89.58.42.177	TCP	54	59678 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
8	0.09518...	172.17.0.2	89.58.42.177	TLSv1.3	303	Client Hello (SNI=v2202403219344260446.goodsrv.de)
9	0.09525...	89.58.42.1...	172.17.0.2	TCP	54	443 → 59678 [ACK] Seq=1 Ack=250 Win=65535 Len=0
10	0.12377...	89.58.42.1...	172.17.0.2	TLSv1.3	28...	Server Hello, Change Cipher Spec, Application Data, Applic
11	0.12380...	172.17.0.2	89.58.42.177	TCP	54	59678 → 443 [ACK] Seq=250 Ack=2760 Win=62780 Len=0
12	0.13095...	172.17.0.2	89.58.42.177	TLSv1.3	118	Change Cipher Spec, Application Data
13	0.13101...	172.17.0.2	89.58.42.177	TLSv1.3	277	Application Data
14	0.13104...	89.58.42.1...	172.17.0.2	TCP	54	443 → 59678 [ACK] Seq=2760 Ack=314 Win=65535 Len=0
15	0.14213...	89.58.42.1...	172.17.0.2	TCP	54	443 → 59678 [ACK] Seq=2760 Ack=537 Win=65535 Len=0
16	0.20387...	89.58.42.1...	172.17.0.2	TLSv1.3	245	Application Data
17	0.20469...	172.17.0.2	89.58.42.177	TCP	54	59678 → 443 [FIN, ACK] Seq=537 Ack=2951 Win=62780 Len=0
18	0.20479...	89.58.42.1...	172.17.0.2	TCP	54	443 → 59678 [ACK] Seq=2951 Ack=538 Win=65535 Len=0
19	0.23286...	89.58.42.1...	172.17.0.2	TLSv1.3	78	Application Data
20	0.23287...	89.58.42.1...	172.17.0.2	TCP	54	443 → 59678 [FIN, ACK] Seq=2975 Ack=538 Win=65535 Len=0

```

Frame 13: 277 bytes on wire (2216 bits), 277 bytes captured (2216 bits) on interface docker0, id 0
Ethernet II, Src: 02:42:ac:11:00:02 (02:42:ac:11:00:02), Dst: 02:42:7a:d2:db:2f (02:42:7a:d2:db:2f)
Internet Protocol Version 4, Src: 172.17.0.2, Dst: 89.58.42.177
Transmission Control Protocol, Src Port: 59678, Dst Port: 443, Seq: 314, Ack: 2760, Len: 223
Transport Layer Security
  TLSv1.3 Record Layer: Application Data Protocol: Hypertext Transfer Protocol
    Opaque Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 218
    Encrypted Application Data [truncated]: b1f5282238f25808cecf0c98580297628ed7d84331446ed1dd2db22c14ce2dfe7
    [Application Data Protocol: Hypertext Transfer Protocol]

```

Figure 16. Intercepted encrypted DoH request packet.

the home network it will pass through a Network Address Translation (NAT) router and receive a public one.

All DoH messages are transmitted through the TLS session as encrypted data and masked as HTTP. An observer can only see that HTTPS messages were exchanged over port 443. One downside of TLS is that it exposes extra information. Figure 17 presents a screenshot of a TLS client hello message captured during session establishment. This message includes the Server Name Indication Server Name Indication (SNI) extension, which reveals the intended DNS server in plain text. While not always present, the SNI extension is frequently used when multiple websites share a single public IP address. As previously mentioned, this can be exploited for techniques like SNI-based blocking.

```

- Handshake Protocol: Client Hello
  Handshake Type: Client Hello (1)
  Length: 240
  Version: TLS 1.2 (0x0303)
  Random: 1c9e90903992ad4b0609e381c6dc0b59bac9dc04edf6d52a95761edc75553683
  Session ID Length: 32
  Session ID: 12977faa17f58fe0bdeeb9b39f292d8ed215e95463dd5f9892642f9574d6eb66
  Cipher Suites Length: 6
  Cipher Suites (3 suites)
  Compression Methods Length: 1
  Compression Methods (1 method)
  Extensions Length: 161
  Extension: server_name (len=36) name=v2202403219344260446.goodsrv.de
    Type: server_name (0)
    Length: 36
  Server Name Indication extension
    Server Name list length: 34
    Server Name Type: host_name (0)
    Server Name length: 31
    Server Name: v2202403219344260446.goodsrv.de

```

Figure 17. Intercepted encrypted DoH TLS Client Hello packet.

Other metadata that is exposed during the handshake is the supported cipher suites or the session ID as illustrated. All this information might be used to infer some information about the client.

Category	Metric	Result
Performance	Round Trips	3
Privacy	Exposed Client Identity	Yes
	Exposed DNS Query	No
	Exposed Metadata	Yes
Security	Risk of Tampering	No
	Risk of DNS Server IP Blocking	Yes
	Risk of Destination Hostname Blocking	Yes
	Risk of Protocol Blocking	No

Table 2. Evaluation metrics for DNS over HTTPS.

Table 2 provides an overview of the findings for DoH. Figure 15 shows that the DNS lookup took 3 handshakes. Figure 16 showed that the client’s IP is exposed. The DNS query is encrypted and can only be seen as Encrypted Application Data. With an encrypted query, the risk of tampering with the DNS content is also eliminated, as the client won’t be able to decrypt the response as soon as a single bit is changed. This is because of the avalanche effect property of TLS encryption. Figure 17 reveals some metadata that is present in the TLS client hello. In Figure 16, it is highlighted, that the resolver’s IP address and hostname are visible, which can be used for filtering. The DNS protocol is not revealed by the intercepted traffic, observers can only see the HTTP protocol.

5.2 Analyzing DoQ

The DoQ testing was conducted in a similar setting as for DoH. Again, Q was used as the DoQ client because it allows exporting the TLS secrets for debugging purposes. Testing DoQ requires finding a different resolver though, since Cloudflare does not yet support DoQ on their DNS servers. For the following traffic captures, DNS queries were sent to nextdns.

```
q taltech.ee A @quic://51252d.dns.nextdns.io:853 -S \
--reuse-conn --tls-key-log-file=out.key
```



Figure 18. Traffic capture of a DoQ request to nextdns DoQ resolver.

Figure 18 shows the flow graph of a DoQ lookup to nextdns. The TLS key from the DoQ client was added to Wireshark for more insight into the message exchange. With the decrypted data, it can be seen that the query was resolved in two round trips. Q supports TLS 1.3 and is using QUIC's 1-RTT connection setup by default. The TLS handshake is already done in the first round trip, unlike with DoH which required a separate TCP handshake first.

The QUIC traffic is mostly encrypted. The recorded traffic in Figure 19 shows details of packet 16 from the same DNS request seen in Figure 18. From the encrypted DNS request, it is possible to learn the client's IP as well as the server's IP and port. Other than that, there is not much more information exposed here.


```

No.    Time           Source            Destination      Protocol  Length  Info
-----
5 0.050518398  10.0.2.15       185.87.111...   QUIC     12..    Initial, DCID=ab95a26aa1aef2c6065ac7f2b609, PKN: 0, PADDING, CRYPTO
6 0.250980971  10.0.2.15       185.87.111...   QUIC     12..    Initial, DCID=ab95a26aa1aef2c6065ac7f2b609, PKN: 1, PADDING, CRYPTO
7 0.251018385  10.0.2.15       185.87.111...   QUIC     12..    Initial, DCID=ab95a26aa1aef2c6065ac7f2b609, PKN: 2, PADDING, CRYPTO
8 0.258924454  185.87.111...  10.0.2.15       QUIC     12..    Handshake, SCID=76f25007
9 0.258924528  185.87.111...  10.0.2.15       QUIC     12..    Handshake, SCID=76f25007
10 0.258973174  185.87.111...  10.0.2.15       QUIC     12..    Handshake, SCID=76f25007
11 0.258973195  185.87.111...  10.0.2.15       QUIC     185     Protected Payload (KP0)
12 0.259199967  10.0.2.15       185.87.111...   QUIC     12..    Initial, DCID=76f25007, PKN: 3, ACK, PADDING
13 0.259214269  10.0.2.15       185.87.111...   QUIC     78     Handshake, DCID=76f25007
14 0.268052092  10.0.2.15       185.87.111...   QUIC     142    Protected Payload (KP0)
15 0.268065503  10.0.2.15       185.87.111...   QUIC     71     Protected Payload (KP0)
16 0.268076305  10.0.2.15       185.87.111...   QUIC     97     Protected Payload (KP0)
17 0.275720743  185.87.111...  10.0.2.15       QUIC     296    Protected Payload (KP0)
18 0.286966803  185.87.111...  10.0.2.15       QUIC     109    Protected Payload (KP0)
19 0.287051874  10.0.2.15       185.87.111...   QUIC     70     Protected Payload (KP0)

* Frame 16: 97 bytes on wire (776 bits), 97 bytes captured (776 bits) on interface enp0s3, id 0
* Ethernet II, Src: PCSSystemtec f9:9b:d0 (08:00:27:f9:9b:d0), Dst: 52:54:00:12:35:02 (52:54:00:12:35:02)
* Internet Protocol Version 4, Src: 10.0.2.15, Dst: 185.87.111.218
* User Datagram Protocol, Src Port: 56236, Dst Port: 853
* QUIC IETF
  * QUIC Connection information
    [Packet Length: 55]
  * QUIC Short Header
    0... .. = Header Form: Short Header (0)
    .1... .. = Fixed Bit: True
    ..0... .. = Spin Bit: False
  Remaining Payload: f049a6c20b1baee9d04b07b74614d014d169fbf16cd6afa4695aa446f8cd0dec1aeb0152e227fb8cbf2be937df153cab2a48c8bf1226

```

Figure 19. Traffic capture of a DoQ request to nextdns DoQ resolver.

The protected payload packets do not expose what domain was resolved, but the QUIC TLS handshake still exposes some information as shown in the following screenshots. During traffic captures, the name of the DNS server can be seen in the SNI field and the Application Layer Protocol Negotiation (ALPN) field reveals the protocol, in this case, DoQ.

```

* TLSv1.3 Record Layer: Handshake Protocol: Client Hello
  * Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 267
    Version: TLS 1.2 (0x0303)
    Random: 30c4d373654cde60fdf450af8afbc07bfb681f8a94ba9b2126717fdb64dbd3b7
    Session ID Length: 0
    Cipher Suites Length: 6
    Cipher Suites (3 suites)
    Compression Methods Length: 1
    Compression Methods (1 method)
    Extensions Length: 220
    Extension: server_name (len=26) name=51252d.dns.nextdns.io
    Extension: status_request (len=5)
    Extension: supported_groups (len=10)
    Extension: ec_point_formats (len=2)
    Extension: signature_algorithms (len=26)
    Extension: renegotiation_info (len=1)
    Extension: extended_master_secret (len=0)
    Extension: application_layer_protocol_negotiation (len=6)
      Type: application_layer_protocol_negotiation (16)
      Length: 6
      ALPN Extension Length: 4
    - ALPN Protocol
      ALPN string length: 3
      ALPN Next Protocol: doq

```

Figure 20. TLS Client Hello from captured DoQ request to nextdns DoQ resolver.

With this information, an observer can block the traffic by filtering DoQ handshakes or by blocking based on the nameservers specified in the handshake. Figure 21 shows the QUIC transport parameters that are also exposed during the initial handshake. This is a lot of metadata that an observer might use to identify clients.

```
- Extension: quic_transport_parameters (len=55)
  Type: quic_transport_parameters (57)
  Length: 55
  Parameter: GREASE (len=0)
  Parameter: initial_max_stream_data_bidi_local (len=4) 524288
  Parameter: initial_max_stream_data_bidi_remote (len=4) 524288
  Parameter: initial_max_stream_data_uni (len=4) 524288
  Parameter: initial_max_data (len=4) 786432
  Parameter: initial_max_streams_bidi (len=2) 100
  Parameter: initial_max_streams_uni (len=2) 100
  Parameter: max_idle_timeout (len=4) 30000 ms
  Parameter: max_udp_payload_size (len=2) 1452
  Parameter: max_ack_delay (len=1) 26
  Parameter: disable_active_migration (len=0)
  Parameter: active_connection_id_limit (len=1) 4
  Parameter: initial_source_connection_id (len=0)
```

Figure 21. Traffic capture of a DoQ request to nextdns DoQ resolver.

5.2.1 0-RTT and Session Resumption for DNS over QUIC

Sengupta et al. [5] showed in their paper that it is possible to drastically speed up the DNS resolution process for DoQ by employing 0-RTT. They used a complex setup to extract TLS session tickets from a custom Chromium build and then reused them with their DNS client. They used DNSperf, an open-source tool². The tool has not seen any updates though in the last two years and it had some broken dependencies, thus it was not further evaluated.

Many different DoQ implementations were evaluated by reviewing their documentation and code. Most don't even mention support for 0-RTT and don't use EarlyData structures in the code. The EarlyData [21] functions enable sending encrypted 0-RTT data in the first TLS message.

Routedns claims to support 0-RTT for the QUIC client when paired with a resolver that also supports 0-RTT. With a code analysis, it was confirmed that indeed the connection setup is using EarlyData functions. To study the properties of DoQ, a local DNS stub resolver was set up using Routedns. Routedns is configured to listen locally on UDP port 5355 and to forward DoQ requests to the selected public resolvers. To allow decrypting the traffic, Routedns's client code was modified to enable writing the TLS secrets to a file.

In a first experiment, DoQ queries were made to multiple public resolvers. More specifically nextdns, adguard and freedns. Repeated queries were triggered after 30 seconds of the initial DoQ query, but no 0-RTT packets were detected in Wireshark. This confirms the findings of Kosek et al. [4] who could not find any public DoQ server that offered support for 0-RTT data.

The next step was to set up a custom server that supports 0-RTT. For this, a reverse proxy

²<https://github.com/mgranderrath/dnsperf>

is needed that terminates the QUIC session and only forwards the DNS traffic to the Routedns listener. There are only a few HTTP servers that offer support for 0-RTT packets in HTTP/3 but this will not work with DoQ since it is not using HTTP. There was no good candidate found that offers QUIC reverse proxy functionality for other traffic than HTTP.

To not run out of options, it was decided to try to use DNS over HTTP/3. Both Routedns and Q can use this mode of DoH by changing the client configuration. Caddy is a popular HTTP server that can also be used as a reverse proxy to handle TLS connections. Since version v2.7.3 (August 2023) it supports 0-RTT for HTTP/3 traffic ³. Using the latest version requires building it from source code. Most Linux packet managers offer an older version of Caddy.

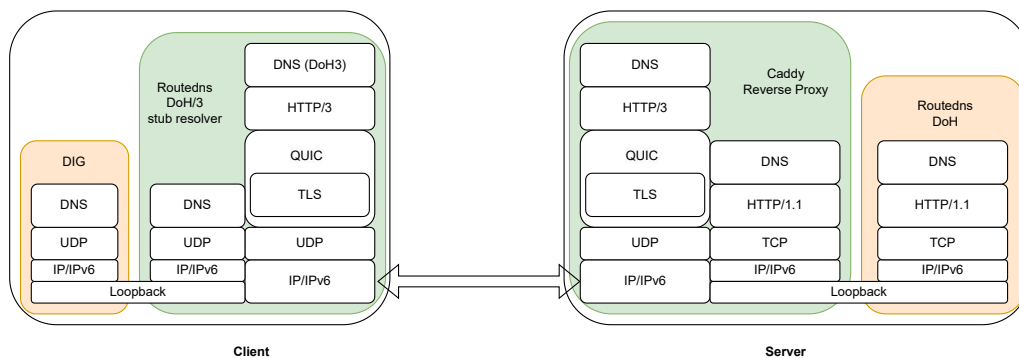


Figure 22. Protocol stack used for 0-RTT testing.

The Routedns listener was configured to DoH over cleartext HTTP. This use case is only recommended for such reverse-proxy scenarios, where TLS is handled by a different application. Caddy was set up in reverse proxy mode to forward the DoH3 packets as cleartext HTTP to the local Routedns listener as shown in Figure 22.

To confirm whether 0-RTT EarlyData was used for the exchange, traffic captures were taken on the servers's public as well as loopback interface. During the test, multiple queries were requested, to ensure TLS secrets could be exchanged beforehand. Unfortunately, the captured results showed that no 0-RTT packets were sent.

In Figure 23 the blue highlighted packet carries the DoH query. The QUIC stream was decrypted with the TLS secrets from Routedns. There were no 0-RTT packets captured. This was strange because it was previously confirmed that the caddy server supports 0-RTT. A static website was hosted and while repeatedly accessing that website in chromium, 0-RTT were observed in Wireshark.

Routedns does not have documentation on the 0-RTT support and there are also no other

³<https://github.com/caddyserver/caddy/releases/tag/v2.7.3>

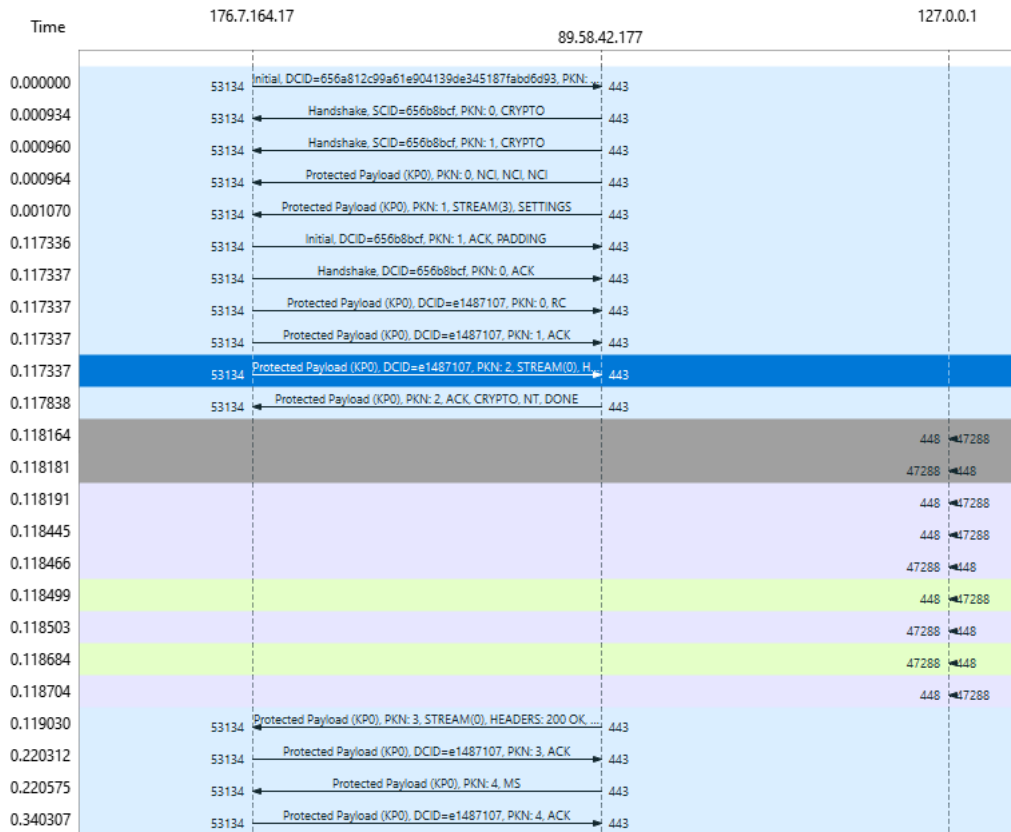


Figure 23. Message flow for DNS over HTTP/3 with Routedns as client.

mentions about 0-RTT on the GitHub page except that one statement. It seems like it either is not fully supported or some other measures or configuration changes are needed to get it to work. In the next attempt, Chromium was used as client for DoH/3. DoH requests can be triggered manually by reusing the HTTP GET requests from a decrypted message. Chromium does not understand how to handle the response but it works well enough in triggering a request.

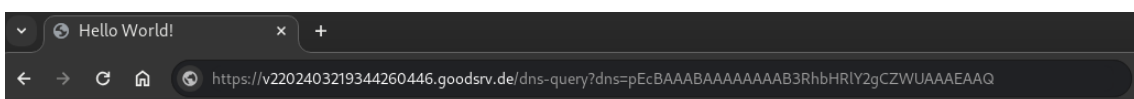


Figure 24. Triggering a DoH request manually.

The following flow graph shows how a successful 0-RTT DNS request over HTTP/3 can return a response in one single round trip.

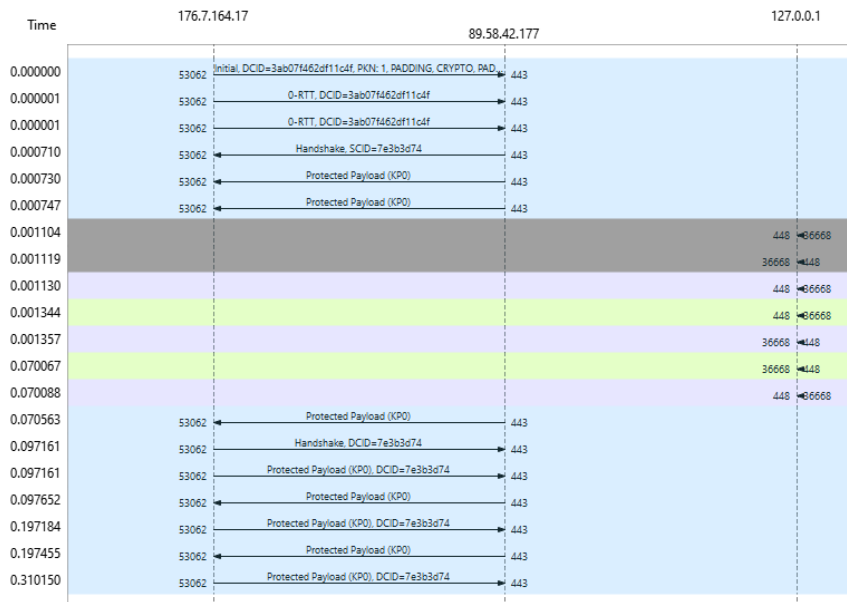


Figure 25. Message flow for DNS over HTTP/3 with Chromium as client.

After it was confirmed that 0-RTT can work with the server setup, the goal was to also get the Routedns client to support it as well. This would then allow proper handling of the query responses. After some debugging and testing, it was found that the client was missing a TLS session ticket cache and was not using the right HTTP Get0RTT method. These options are necessary for session resumption and 0-RTT to work. After some experimenting, it was possible to modify the client and a pull request was opened on GitHub to integrate the changes ⁴.

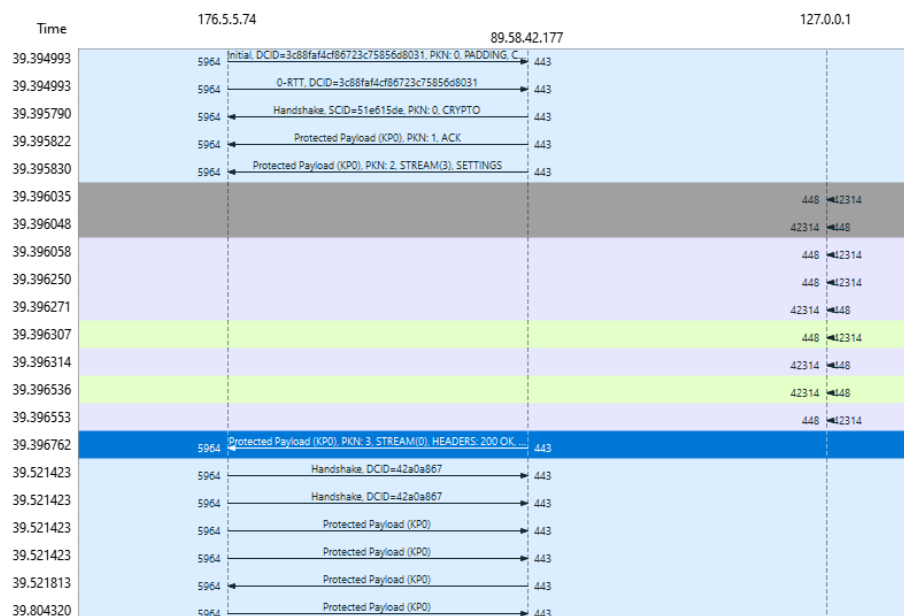


Figure 26. 0-RTT flow for DNS over HTTP/3 with the updated Routedns client.

⁴<https://github.com/folbricht/routedns/issues/385>

With the updated client, Routedns can now also use the 0-RTT TLS handshake. Unfortunately, the 0-RTT packets themselves can not be decrypted with the extracted TLS secrets. From the captured files it can be observed that the 0-RTT is never sent on its own. The regular handshake "initial" packet is also sent, which contains the full TLS client hello message. This is done as a fallback mechanism if the server decides to drop or reject the 0-RTT session resumption packet. The 0-RTT packet itself is encrypted and does not reveal information about its contents.

The 0-RTT mode analyzed in this section was tested with DNS over HTTP/3. DoQ's properties are very similar, it just eliminates the HTTP/3 layer on top of QUIC. The QUIC and TLS message exchanges are very similar for both protocols.

Due to a coincidence while analyzing the code of Routedns's DoQ client, it was found, that Adguard's dnsproxy client⁵ also supports 0-RTT. It is not mentioned in the documentation, but there are many functions that handle EarlyData in the tool's code. dnsproxy is similar to Routedns in that it supports multiple DNS protocols and can act as client, server, or, as the name implies, DNS proxy.

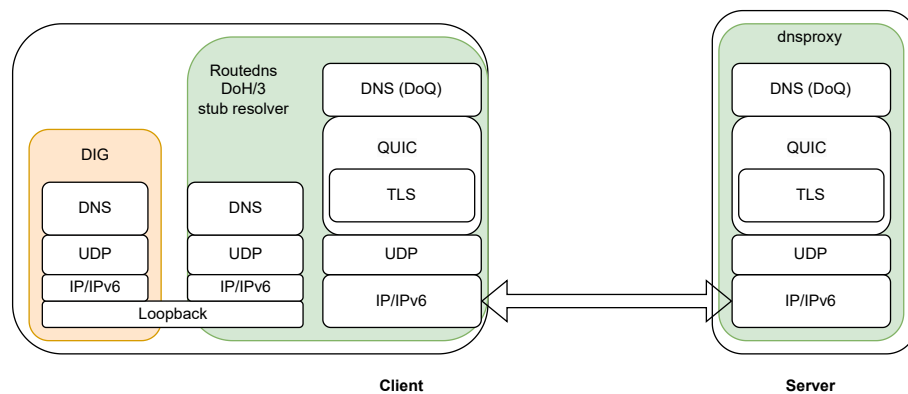


Figure 27. Updated setup with dnsproxy.

The dnsproxy was installed on the proxy server and configured to act as a resolver for DoQ, similar to Routedns used in the previous sections. With dnsproxy acting as the DoQ resolver, the setup was simplified a lot. It was equipped with a cache and configured to forward DNS queries to Cloudflare's resolvers as shown by the following command.

```
./dnsproxy -l 0.0.0.0 -q 8853 -u 1.1.1.1:53 -p 0 \
  --cache --cache-min-ttl=600 --tls-crt=$path.crt \
  --tls-key=$path.key -v
```

⁵<https://github.com/AdguardTeam/dnsproxy>

It was then tested with the updated Routedns client and successfully confirmed that 0-RTT was also working for DoQ. Figure 28 shows 0-RTT traffic between a Routedns client and dnsproxy resolver. The traffic was decrypted with TLS secrets exported from the client. The 0-RTT message can not be decrypted with the secrets, thus only the DNS query response is shown in the capture file.

No.	Time	Source	Destination	Protocol	Length	Info
6	0.202425475	172.17.0.2	89.58.42.177	QUIC	1294	Initial, DCID=e83ad173780d7832585666, PKN: 0, PADDING, CRYPTO
7	0.202453927	172.17.0.2	89.58.42.177	QUIC	135	0-RTT, DCID=e83ad173780d7832585666
8	0.331037317	89.58.42.177	172.17.0.2	QUIC	1294	Handshake, SCID=e5a8e01a, PKN: 0, CRYPTO
9	0.331089436	89.58.42.177	172.17.0.2	QUIC	67	Protected Payload (KP0), PKN: 1, ACK
10	0.331225014	89.58.42.177	172.17.0.2	DNS	120	Standard query response 0x0000 A taltech.ee A 81.21.253.51 OPT

Figure 28. 0-RTT flow for DoQ with the updated Routedns client and dnsproxy resolver.

The updated client was also tested with public DoQ resolvers but it was again confirmed that none of them support 0-RTT. Adguards DoQ server creates and shares session tickets that can be used for 0-RTT requests but even after multiple trials, it was always rejected as the server responds with a QUIC retry message.



Figure 29. 0-RTT rejected by Adguards public resolver.

Table 3 provides an overview of the findings for DoQ. 0-RTT introduces some new privacy concerns that will be analyzed in the next section and that don't appear in this table. As shown in this section, DoQ has similar properties to DoH but because it does not rely on HTTP, its protocol and port are visible to an observer. This makes it easier to block the protocol.

Category	Metric	Result
Performance	Round Trips	1-2
Privacy	Exposed Client Identity	Yes
	Exposed DNS Query	No
	Exposed Metadata	Yes
Security	Risk of Tampering	No
	Risk of DNS Server IP Blocking	Yes
	Risk of Destination Hostname Blocking	Yes
	Risk of Protocol Blocking	Yes

Table 3. Evaluation metrics for DNS over QUIC.

5.2.2 Security concerns with 0-RTT DoQ

As previously mentioned, there exist some issues with 0-RTT messages. They can be captured and later used for a replay attack. Tools like scapy⁶ can be used to replay intercepted traffic.

In a small experiment with the 0-RTT DoQ setup, it was tried to reproduce the replay attack, by intercepting 0-RTT traffic. First, a regular DoQ query was sent to the resolver to establish a TLS session. Then the network interface on the client machine was deactivated to simulate intercepting traffic. While the network connectivity was down, a new DoQ request was made to the server and captured with Wireshark.

No.	Time	Source	Destination	Protocol	Length	Info
20.0004...	172.17.0.2	89.58.42.177	QUIC	1294	Initial, DCID=702b07190d0a5308125d5398e8, PKN:	
30.0004...	172.17.0.1	172.17.0.2	ICMP	590	Destination unreachable (Network unreachable)	
40.0004...	172.17.0.2	89.58.42.177	QUIC	134	0-RTT, DCID=702b07190d0a5308125d5398e8	
50.0004...	172.17.0.1	172.17.0.2	ICMP	162	Destination unreachable (Network unreachable)	

Figure 30. Intercepted 0-RTT message.

The interface was activated again and the capture file was loaded in scapy. The following Python commands were then used to resend the 0-RTT QUIC packets for the attack.

```
from scapy.all import *
packets = rdpcap("../test.pcapng")
p = packets[1][UDP].payload
p2 = packets[3][UDP].payload
new_p = IP(dst="89.58.42.177")/UDP(sport=36491, dport=8853)/p
new_p2 = IP(dst="89.58.42.177")/UDP(sport=36491, dport=8853)/p2
for packet in [new_p, new_p2]:
    send(packet)
```

The crafted packets can be observed with Wireshark as it is sent to the server. If everything was done correctly and the 0-RTT ticket was still valid, the resolver will send a response back. Figure 31 shows a capture of the packets sent with scapy and the response by the server that was sent back.

The attacker in this case has no access to the victim's TLS secrets, so he can not decrypt the response from the server to learn what the query was. The problem arises if the attacker can monitor or infer information from the resolver's outgoing traffic.

⁶<https://github.com/secdev/scapy>

No.	Time	Source	Destination	Protocol	Length	Info
30	0.016328...	10.0.2.15	89.58.42.177	QUIC	1294	Initial, DCID=702b07190d0a5308125d5398e8, P
40	0.064617...	10.0.2.15	89.58.42.177	QUIC	134	0-RTT, DCID=702b07190d0a5308125d5398e8
50	0.165827...	89.58.42.177	10.0.2.15	QUIC	1294	Protected Payload (KP0)
60	0.165866...	10.0.2.15	89.58.42.177	ICMP	590	Destination unreachable (Port unreachable)
70	0.199627...	89.58.42.177	10.0.2.15	QUIC	66	Protected Payload (KP0)
80	0.199641...	10.0.2.15	89.58.42.177	ICMP	94	Destination unreachable (Port unreachable)

Figure 31. 0-RTT replay attack response.

No.	Time	Source	Destination	Protocol	Length	Info
1434	92.6739...	176.4.146.248	89.58.42.177	QUIC	1294	Initial, DCID=702b07190d0a5308125d5398e8, PKN: 0, PADDING, CRYPTO
1435	92.6748...	89.58.42.177	176.4.146.248	QUIC	1294	Handshake, SCID=d33a5f9f
1437	92.7062...	176.4.146.248	89.58.42.177	QUIC	134	0-RTT, DCID=702b07190d0a5308125d5398e8
1438	92.7065...	89.58.42.177	176.4.146.248	QUIC	66	Protected Payload (KP0)
1439	92.7068...	89.58.42.177	1.1.1.1	DNS	90	Standard query 0x0000 A test.fi OPT
1442	92.7913...	89.58.42.177	176.4.146.248	QUIC	1294	Initial, SCID=d33a5f9f, PKN: 1, PADDING, CRYPTO
1443	92.7914...	89.58.42.177	176.4.146.248	QUIC	1294	Initial, SCID=d33a5f9f, PKN: 2, PADDING, CRYPTO
1447	92.9058...	89.58.42.177	176.4.146.248	QUIC	237	Handshake, SCID=d33a5f9f
1448	92.9058...	89.58.42.177	176.4.146.248	QUIC	237	Handshake, SCID=d33a5f9f

Figure 32. Traffic capture during 0-RTT replay on the resolver.

In this case, the replay attack triggered the resolver to do a recursive lookup with DNS to Cloudflare’s resolvers. Of course, real DNS servers that handle encrypted DNS should not be configured to use cleartext DNS for recursive lookups. For this experiment, it was used to showcase this one scenario.

The attack is partially mitigated by reducing the observability of this recursive traffic with encryption. But it is not eliminated. In another attack scenario, if the attacker has access to the recursive resolver, he could select a time frame with low traffic to isolate the replayed DoQ query and guarantee a reverse lookup. By controlling the recursive solver, he can now decrypt the traffic from the original receiving resolver of the 0-RTT request, thus he can learn the query.

Session ticket linkability with session resumption is another privacy concern. If session tickets are repeatedly used it can lead to easy fingerprinting for any observer. The DoQ RFC 9250[25] already specifies recommendations to minimize the risks involved with session resumption. It is recommended that clients should use resumption tickets only once and by default, clients should not use session resumption if the IP address has changed.

5.3 Oblivious DoH evaluation

To establish a baseline for comparison, an Oblivious DNS setup was additionally deployed within the testbed. However, similar to the other protocols, the ODoH standard currently has limited implementation options. The available implementations are also still considered experimental.

The Cloudflare ODoH relay server⁷ and client⁸ were installed in the testbed. The open-source implementation was also used in their paper [3] that focused on ODoH performance.

While the Cloudflare ODoH implementation holds promise, it has not received any updates since 2021. One key challenge is the confusing mechanism for clients to acquire the resolver's keys. In the existing version of the ODoH client, this is by default carried out through an additional handshake process. The client initiates a handshake with the resolver to obtain its HPKE public key, before making the ODOH request. This means that the server is in direct contact with the client in this implementation. This approach is meant to be used for testing the ODoH mechanism, and a more robust and scalable solution for key distribution is needed for ODoH to reach its full potential.

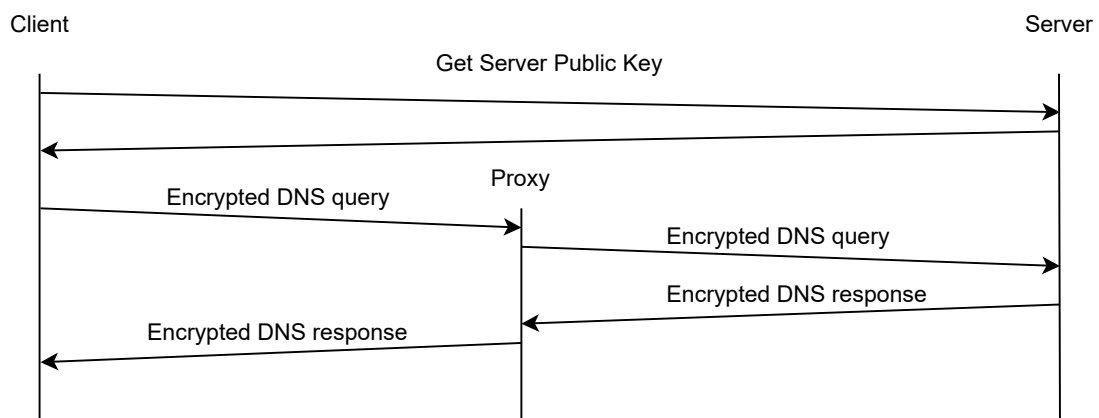


Figure 33. Simplified Message Flow for an ODoH Request.

In this scenario, although upstream servers do not directly receive DNS queries from the client, they still acquire knowledge of the client's IP address during the preceding key query. It first shows a connection to the target server for the key exchange and only after that the connection to the ODoH proxy. Ideally, there should be public servers that collect public ODoH targets and regularly update their public keys. Users can then get a full list of available ODoH targets and proxies to choose randomly from.

⁷<https://github.com/cloudflarearchive/odoh-server-go>

⁸<https://github.com/cloudflare/odoh-client-go>

Only after studying the code of the client implementation, an undocumented "--certificate" option was discovered that allows skipping the key exchange for every new DNS resolution. In Figure 33 this translates to decoupling the first transaction from the message exchange. The public key can be gathered with the odohconfig-fetch functionality of the client. Gathering the key currently also works with DoH. A DNS request can also request the server's TXT records. For ODoH, the resolver shares his public keys this way.

```

root@91e02482a309:/go/odoh-client-go# ./odoh-client odohconfig-fetch --target odoh.cloudflare-dns.com > cloudflare.conf
root@91e02482a309:/go/odoh-client-go# echo $(< cloudflare.conf)
002c00010028002000010001002096c1f14851e7f44e2b96c4c0f8398c69af3727e52a30b9db28a62de60df79a70
root@91e02482a309:/go/odoh-client-go# ./odoh-client odoh --dnstype A --domain taltech.ee --target odoh.cloudflare-dns.com --proxy v2
02403219344260446.goodsrv.de --config $(< cloudflare.conf)
;; opcode: QUERY, status: NOERROR, id: 58863
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;taltech.ee.      IN      A

;; ANSWER SECTION:
taltech.ee.      60     IN      A       81.21.253.51

Resolution time:
149.572983ms
root@91e02482a309:/go/odoh-client-go# S

```

Figure 34. Console Screenshot of an ODoH request.

Figure 34 shows how a domain name query is made with the Cloudflare ODoH client. Required parameters include the domain and type to look up, the target resolver URL, the proxy URL, and the target's public key config.

5.3.1 Oblivious DoH performance

Again, the source code of the client was modified to store the TLS secrets. Figure 35 shows a decrypted traffic capture from the client. There is a big time gap between the green request and response packets that contain the ODoH query. This is because the traffic in this capture only shows the communication between the client and proxy. The proxy has to wait for the resolver's response.

To get a better picture of what is happening, traffic from the proxy was also evaluated. For decrypting all ODoH messages, it is necessary to have access to the proxy's TLS secrets, since the proxy opens its own TLS session with the target server. The proxy only forwards the encrypted query string.

The proxy's source code was also modified to log TLS secrets. With that done, it is now possible to analyze all the exchanged messages in Wireshark. Traffic captures were taken on both the client and proxy's public-facing interface. Figure 36 shows the message flow graph from the capture on the proxy.

The flow chart illustrates that both connections, client-to-proxy and proxy-to-resolver,

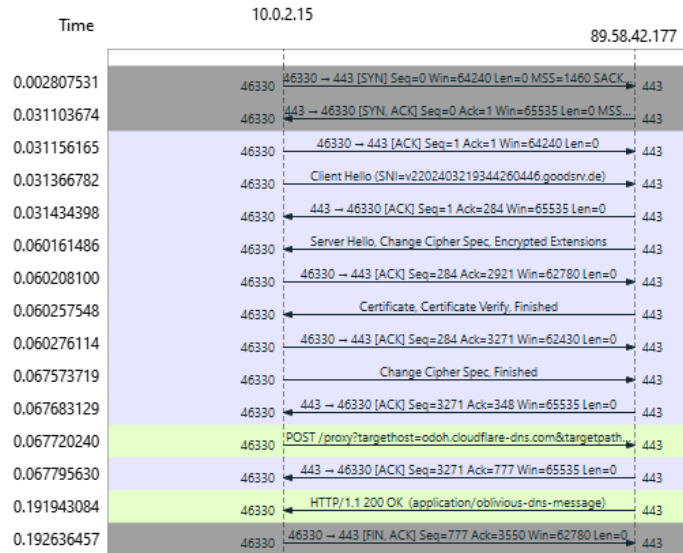


Figure 35. Traffic capture of an oblivious DoH resolution with a loaded certificate, captured on the ODoH client.

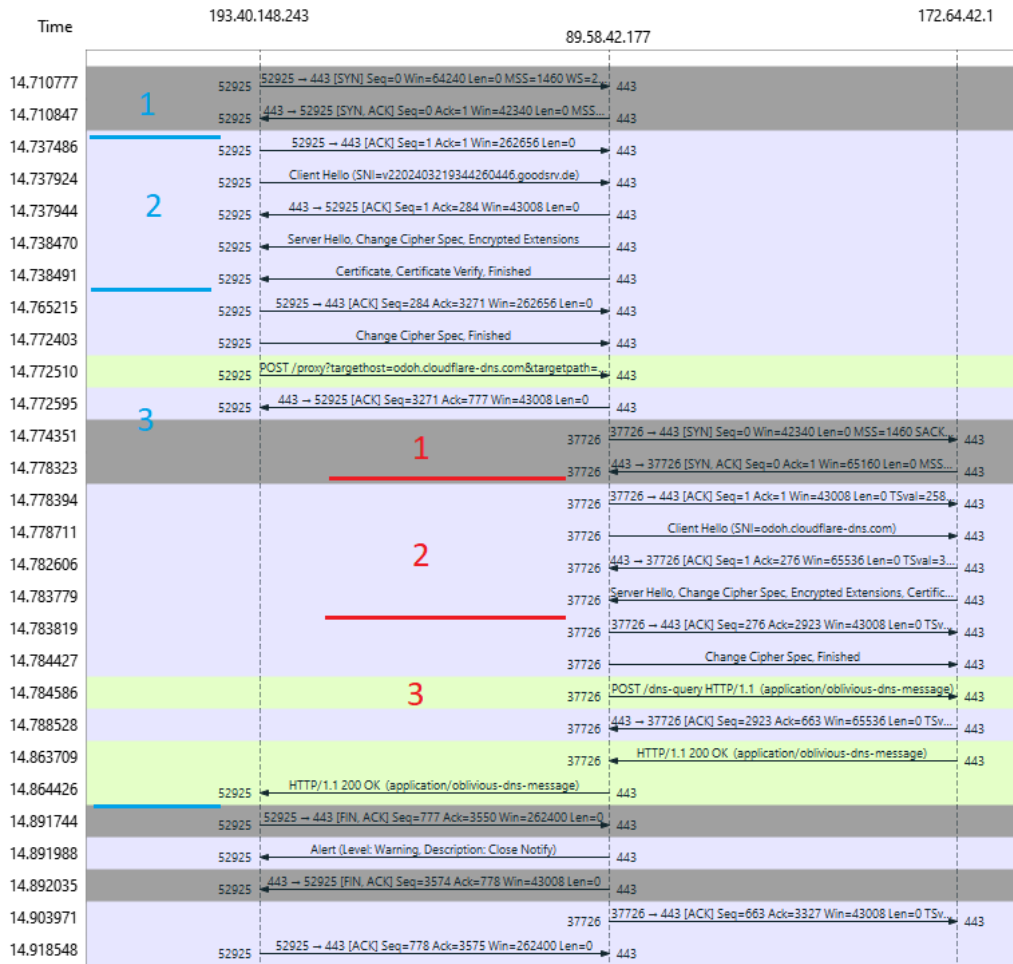


Figure 36. Flow graph of an oblivious DoH resolution with a loaded certificate, captured on the ODoH proxy.

require a full TCP/TLS 1.3 handshake with two round trips before transmitting the application data. This initial handshake secures the channel for the application payload, in this case, the HPKE encrypted ODoH query.

The ODoH query resolution takes 3 round trips between client and proxy and a further 3 round trips between proxy and target resolver. Furthermore, one has to consider that this is only possible if the target's public key is known beforehand. Otherwise, it would require another 3 round trips to gather the key from a server that lists ODoH public keys.

5.3.2 Oblivious DoH privacy and security

The ODoH protocol is very similar to DoH in its properties, with the additional bonus that it conceals the client's identity from the target resolver. To analyze the full scope of the connection, one must consider two observers. One that listens to traffic between the client and proxy and one that listens to traffic between the proxy and the target resolver.

The observer between the client and proxy can observe a message stream that has identical properties to DoH. He sees a HTTPS stream to a public IP. From the observed traffic he can not directly trace back that DNS is tunneled inside the HTTPS stream. Furthermore, he should not be able to tell if the target IP is functioning as a proxy server or as an HTTPS server.

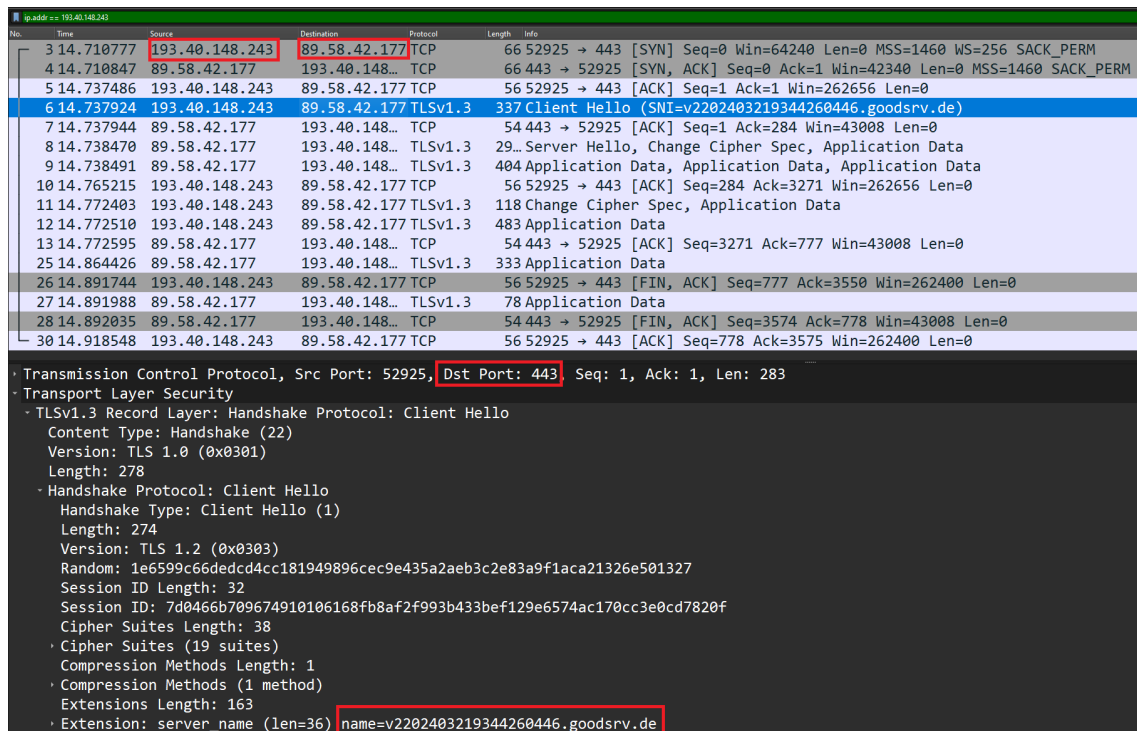


Figure 37. Traffic capture of an oblivious DoH resolution captured on the ODoH proxy.

To the observer, the client hello message provides the largest source of information. Again, there is a lot of metadata and information such as the SNI, but the vast majority of the exchanged messages are encrypted, as shown in Figure 37

The TLS stream is terminated at the proxy, thus it is able to read the application data. Figure 38 shows the decrypted post message, that provides the proxy the forwarding instructions. The file data that can be seen is the HPKE encrypted DNS query. The proxy can not decrypt it.

```

Hypertext Transfer Protocol
POST /proxy?targethost=odoh.cloudflare-dns.com&targetpath=%2Fdns-query HTTP/1.1\r\n
Host: v2202403219344260446.goodsrv.de\r\n
User-Agent: Go-http-client/1.1\r\n
Content-Length: 117\r\n
Accept: application/oblivious-dns-message\r\n
Content-Type: application/oblivious-dns-message\r\n
Accept-Encoding: gzip\r\n
\r\n
[Full request URI: https://v2202403219344260446.goodsrv.de/proxy?targethost=odoh.cloudflare-dns.com&targetpath=%2Fdns-query]
[HTTP request 1/1]
[Response in frame: 25]
File Data: 117 bytes
Media Type
Media type: application/oblivious-dns-message (117 bytes)

```

Figure 38. Decrypted ODoH query message on the proxy.

For the second scenario, where an observer listens in on traffic between the proxy and the target resolver, the following capture file with traffic between proxy and target can be analyzed.

No.	Time	Source	Destination	Protocol	Length	Info
14	14.774351	89.58.42.177	172.64.42.1	TCP	74	37726 → 443 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM
15	14.778323	172.64.42.1	89.58.42.177	TCP	74	443 → 37726 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1400
16	14.778394	89.58.42.177	172.64.42.1	TCP	66	37726 → 443 [ACK] Seq=1 Ack=1 Win=43008 Len=0 TSval=2587014
17	14.778711	89.58.42.177	172.64.42.1	TLSv1.3	341	Client Hello SNI=odoh.cloudflare-dns.com
18	14.782606	172.64.42.1	89.58.42.177	TCP	66	443 → 37726 [ACK] Seq=1 Ack=276 Win=65536 Len=0 TSval=34649
19	14.783779	172.64.42.1	89.58.42.177	TLSv1.3	29...	Server Hello, Change Cipher Spec, Application Data
20	14.783819	89.58.42.177	172.64.42.1	TCP	66	37726 → 443 [ACK] Seq=276 Ack=2923 Win=43008 Len=0 TSval=25
21	14.784427	89.58.42.177	172.64.42.1	TLSv1.3	130	Change Cipher Spec, Application Data
22	14.784586	89.58.42.177	172.64.42.1	TLSv1.3	389	Application Data
23	14.788528	172.64.42.1	89.58.42.177	TCP	66	443 → 37726 [ACK] Seq=2923 Ack=663 Win=65536 Len=0 TSval=34
24	14.863709	172.64.42.1	89.58.42.177	TLSv1.3	470	Application Data
29	14.903971	89.58.42.177	172.64.42.1	TCP	66	37726 → 443 [ACK] Seq=663 Ack=3327 Win=43008 Len=0 TSval=25

```

Frame 17: 341 bytes on wire (2728 bits), 341 bytes captured (2728 bits)
Ethernet II, Src: 4a:27:82:43:56:8c (4a:27:82:43:56:8c), Dst: IETF-VRRP-VRID_01 (00:00:5e:00:01:01)
Internet Protocol Version 4, Src: 89.58.42.177, Dst: 172.64.42.1
Transmission Control Protocol, Src Port: 37726, Dst Port: 443, Seq: 1, Ack: 1, Len: 275
Transport Layer Security
TLSv1.3 Record Layer: Handshake Protocol: Client Hello

```

Figure 39. Traffic capture of Oblivious DoH messages between proxy and target resolver.

An observer can only identify the proxy as the source of the traffic. In this case, the SNI is hinting that the traffic might be ODoH. The other properties remain the same as for DoH traffic.

The resolver can decrypt the traffic, as well as the HPKE encrypted ODoH query. He does not learn any information about the client though as he only learns the proxy's IP address.

An advantage of this setup is also the elimination of metadata that could reveal the client

at the resolver. This is because the TLS session is only affected by the proxy’s properties. In other words, even if many different clients with various TLS properties use this ODoH proxy, to the server, all connections will look the same. For maximum privacy, a ODoH proxy should be used by many different clients, and clients should also rotate the proxies frequently.

Category	Metric	Result
Performance	Round Trips	3 + 3
Privacy	Exposed Client Identity	No
	Exposed DNS Query	No
	Exposed Metadata	No
Security	Risk of Tampering	No
	Risk of DNS Server IP Blocking	No
	Risk of Destination Hostname Blocking	No
	Risk of Protocol Blocking	No

Table 4. Evaluation metrics for Oblivious DNS over HTTPS.

Table 4 summarizes the findings for the evaluation of Oblivious DoH. In the table, it can be seen that there are no security or privacy concerns found with the defined metrics. Yet, the cost for this is that the handshake is long. Figure 36 showed that it takes a total of 3 handshakes between client and proxy and another 3 handshakes to establish the second TLS session between proxy and ODoH resolver.

With the added proxy the privacy metrics are better, compared to DoH and DoQ. It was possible to conceal the client’s identity and eliminate any metadata. Furthermore, since the destination IP is not visible to an observer between client and proxy, the risk of DNS server IP or hostname-based blocking is also eliminated.

6. Implementing a Novel DNS System using MASQUE and DoQ

This chapter describes the implementation process of a MASQUE proxy and the follow-up steps required to forward DNS traffic over it.

6.1 Masque Proxy implementation

The go programming language is used for many DNS and QUIC applications. Cloudflares ODoH implementation uses go, many of the DNS clients and resolvers are written in go, for example, Adguards dnsproxy, Routedns, q, and also the caddy web server is written in go. Most of these applications rely on the quic-go¹ library. It is one of the more feature-rich implementations of the QUIC and HTTP/3 protocol. Unfortunately, it doesn't yet support MASQUE. Work on integrating the MASQUE features to quic-go² just started in late April of 2024.

Google's MASQUE implementation currently offers the most features out of the available implementations. Furthermore, it offers both a client and server implementation. However, it's important to note that even this implementation is labeled as a not production-ready prototype. It was used for further testing.

Installing the tools was not straightforward, since the MASQUE implementation is part of Chromium, it requires building Chromium from source.³

6.1.1 MASQUE operational modes

Google's MASQUE implementation features three different HTTP methods: connect-udp, connect-ip, connect-ethernet. The MASQUE server was set up to listen on port 443. It can be configured only to allow a specific tunneling mode but the default setting supports all modes.

The MASQUE client can be operated in three ways making use of the different HTTP

¹<https://github.com/quic-go/quic-go>

²<https://github.com/quic-go/masque-go>

³https://chromium.googlesource.com/chromium/src/+/main/docs/linux/build_instructions.md

methods. The following section will analyze the behavior of the modes.

```
masque_client myproxy.ee:443 https://quic.nginx.org/
```

HTTP/3 requests can be made by specifying a target URL. This style of proxying uses connect-udp to tunnel the requested webpage over HTTP/3. In this mode, the proxy session will only remain active for the duration it takes to load the target webpage. Because this mode currently only supports HTTP/3 websites it can not be used for DoQ without some modifications to the client and server applications.

The second method can be used by setting the `-bring_up_tun` flag and it doesn't require specifying a target. This option makes use of the connect-ip method and creates a virtual interface TUN device on the host machine. A TUN device is a virtual network device in the Linux kernel.

```
masque_client myproxy.ee:443 --bring_up_tun  
> Bringing up tun with address 10.1.1.2
```

The MASQUE server in this tunnel mode does not forward incoming packets by default. Routing was configured to allow forwarding and masking traffic from the proxy. Once established, a simple ping towards the tunnel will confirm the flow of QUIC packets to the proxy. Figure 40 shows a proxied DNS query to Cloudflare's DNS resolver. This method can be used to conceal the source IP of the DNS query, but it doesn't eliminate the identity leak entirely. Instead, the trust is now shifted to the proxy as it has the full insight of the client, the query and the destination.

This method is most likely also what Apple does with their private relays. In subsection 3.3.2 it was already mentioned that Apple claims to support 0-RTT forwarding in private relay. This raises two possibilities. Either they use a permanent MASQUE tunnel, that is activated as soon as the private relay service is turned on. Alternatively, Apple might have implemented its own 0-RTT secure authentication method that builds upon the HTTP CONNECT concept but addresses the security concerns associated with 0-RTT in this context.

The `-bring_up_tap` mode makes use of the connect-ethernet method. It works similar to the TUN mode but uses TAP devices that can support ethernet frames. TUN devices were chosen as the preferred method because all traffic evaluated in this thesis is IP-based and TUN devices work just fine for that.

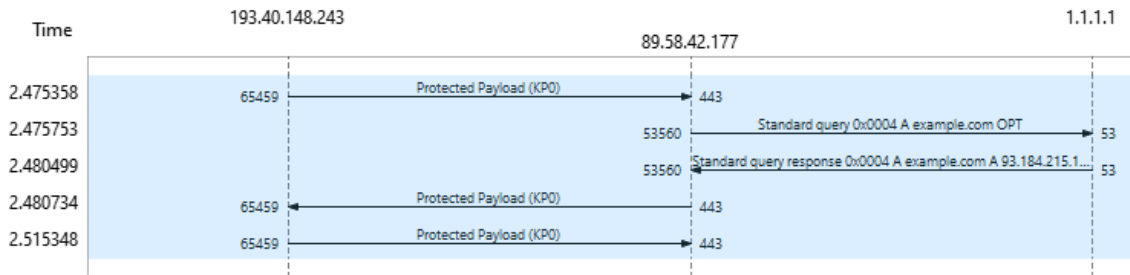


Figure 40. DNS traffic sent over a MASQUE tunnel captured on the server.

6.2 Benefits of a custom MASQUE proxy

The implementation provided by Google already offers a full IP tunnel. This supports all DNS protocols, but can also be used for every other type of internet traffic. Modifying the proxy to only support a specific DNS protocol, for example, DoQ, could limit possible cases of abuse.

Due to the limited timeframe of this thesis, it was decided to stick to the provided MASQUE client and server. For future research, it might be interesting to build a custom, encrypted DNS-only proxy implementation.

7. Evaluating a Novel DNS System using MASQUE and DoQ

In this chapter the prototype built in chapter 6 will be evaluated based on the same evaluation process that was applied in chapter 5.

7.1 Evaluating MASQUE proxied DoQ performance

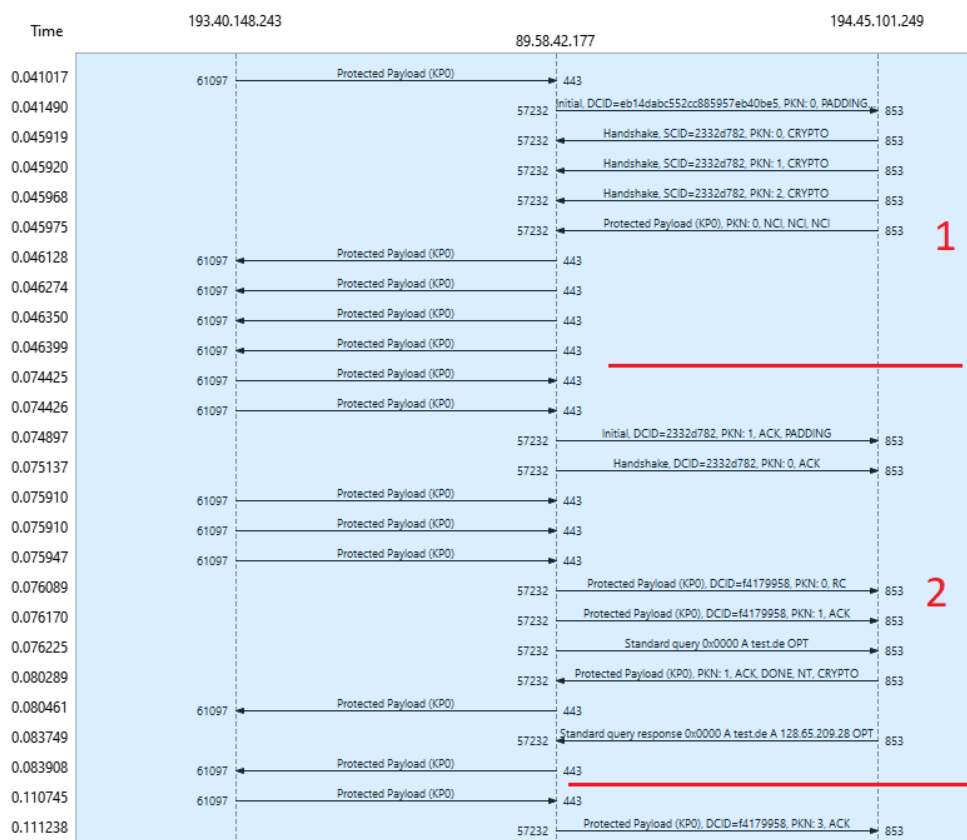


Figure 41. DoQ request to nextdns resolver over a MASQUE proxy.

The lessons learned from evaluating DoQ helped while implementing the novel testbed. A lot of the tools could be reused, but some additions were also required. For the first tests, a public resolver from nextdns was chosen.

Figure 41 shows DoQ traffic captured on the MASQUE proxy. It can be seen that the proxy server immediately forwards the received packets.

The DoQ queries then leverage this established connection for resolution, typically requiring two additional handshakes with the DoQ resolver. This answers the first research

question. It is possible to effectively anonymize DoQ queries with the use of a MASQUE proxy.

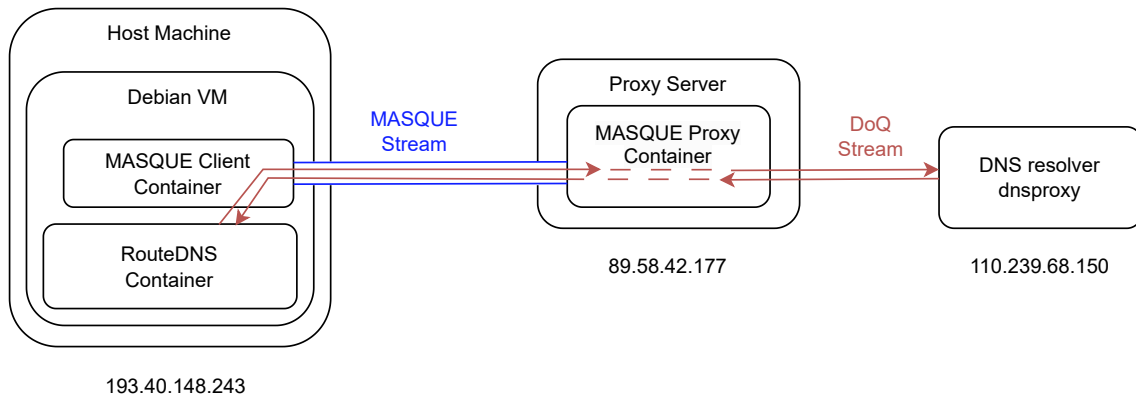


Figure 42. Setup for testing oblivious DoQ using a MASQUE proxy.

Another objective involved exploring the integration of 0-RTT functionality with the MASQUE proxy for DoQ. However, during the evaluation of DoQ solutions, no publicly available resolvers were identified that supported 0-RTT. To address this limitation and test the interaction between MASQUE and 0-RTT DoQ, a secondary server was set up utilizing the previously mentioned dnsproxy tool. This time, the physical distance was bigger, as the DoQ resolver is hosted in Jakarta. Nevertheless, since resolution times are not evaluated, this does not play a big role. Figure 42 shows the full setup used to proxy DoQ with MASQUE.

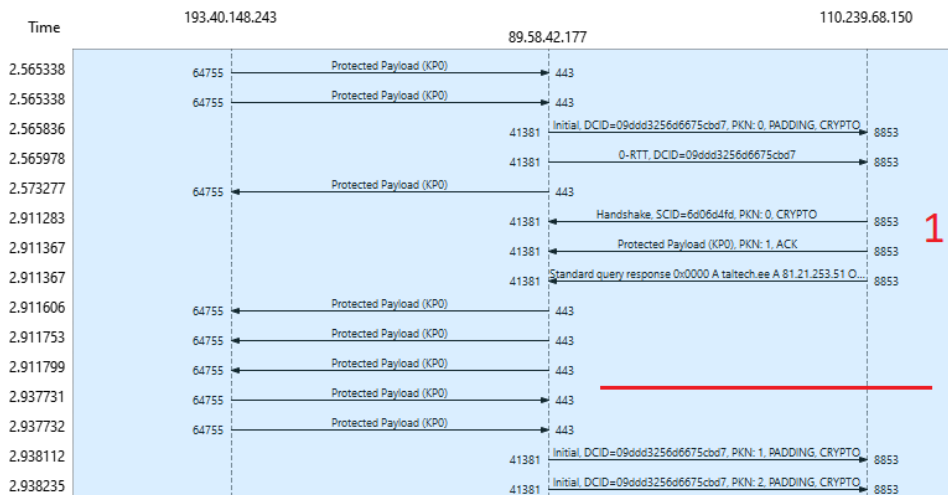
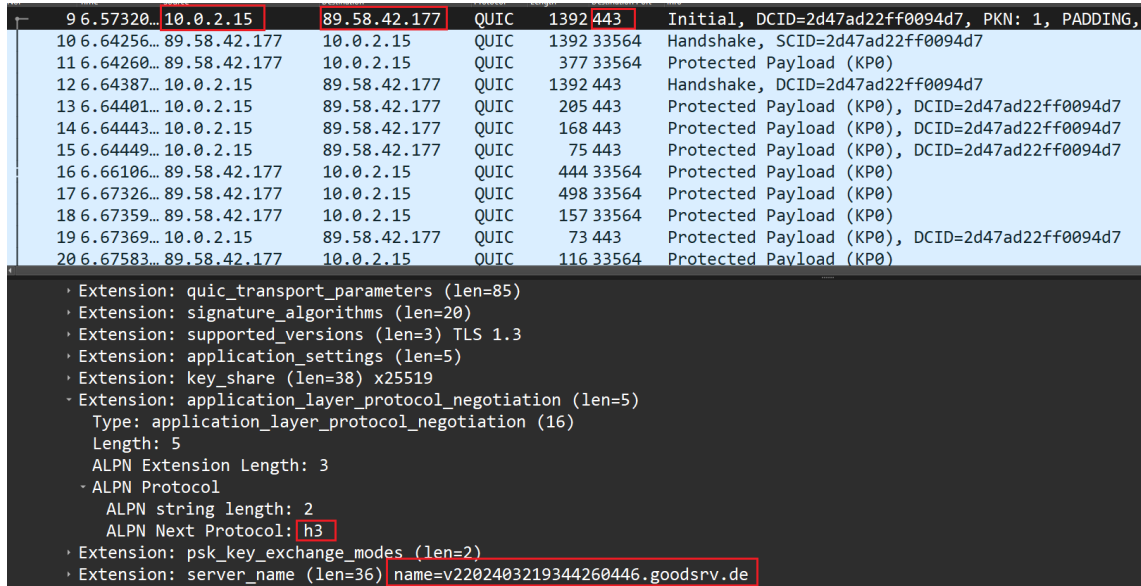


Figure 43. Sending 0-RTT DoQ queries over a MASQUE proxy.

Utilizing the self-hosted dnsproxy resolver with 0-RTT support significantly reduced the number of messages exchanged. The DoQ stream only needs one handshake to resolve the query. This translates to a more efficient process. If a proxy is chosen that can be reached in low latency, then the additional hop will have close to no effect on the overall latency.

7.2 Evaluating MASQUE proxied DoQ security

Similar to the ODoH evaluation, there are also two observers considered here. One that observes traffic between client and proxy and one between proxy and the target resolver. The QUIC traffic from Figure 44 shows the MASQUE tunnel establishment from traffic captures taken in the testbed.



Time	Source IP	Destination IP	Protocol	Length	Details
9.6.57320...	10.0.2.15	89.58.42.177	QUIC	1392 443	Initial, DCID=2d47ad22ff0094d7, PKN: 1, PADDING,
10.6.64256...	89.58.42.177	10.0.2.15	QUIC	1392 33564	Handshake, SCID=2d47ad22ff0094d7
11.6.64260...	89.58.42.177	10.0.2.15	QUIC	377 33564	Protected Payload (KP0)
12.6.64387...	10.0.2.15	89.58.42.177	QUIC	1392 443	Handshake, DCID=2d47ad22ff0094d7
13.6.64401...	10.0.2.15	89.58.42.177	QUIC	205 443	Protected Payload (KP0), DCID=2d47ad22ff0094d7
14.6.64443...	10.0.2.15	89.58.42.177	QUIC	168 443	Protected Payload (KP0), DCID=2d47ad22ff0094d7
15.6.64449...	10.0.2.15	89.58.42.177	QUIC	75 443	Protected Payload (KP0), DCID=2d47ad22ff0094d7
16.6.66106...	89.58.42.177	10.0.2.15	QUIC	444 33564	Protected Payload (KP0)
17.6.67326...	89.58.42.177	10.0.2.15	QUIC	498 33564	Protected Payload (KP0)
18.6.67359...	89.58.42.177	10.0.2.15	QUIC	157 33564	Protected Payload (KP0)
19.6.67369...	10.0.2.15	89.58.42.177	QUIC	73 443	Protected Payload (KP0), DCID=2d47ad22ff0094d7
20.6.67583...	89.58.42.177	10.0.2.15	QUIC	116 33564	Protected Payload (KP0)

```
· Extension: quic_transport_parameters (len=85)
· Extension: signature_algorithms (len=20)
· Extension: supported_versions (len=3) TLS 1.3
· Extension: application_settings (len=5)
· Extension: key_share (len=38) x25519
· Extension: application_layer_protocol_negotiation (len=5)
  Type: application_layer_protocol_negotiation (16)
  Length: 5
  ALPN Extension Length: 3
  ALPN Protocol
    ALPN string length: 2
    ALPN Next Protocol: h3
· Extension: psk_key_exchange_modes (len=2)
· Extension: server_name (len=36) name=v2202403219344260446.goodsrv.de
```

Figure 44. Traffic capture of a MASQUE tunnel establishment.

To an observer, it looks similar to HTTP/3 traffic. The traffic is encrypted and doesn't reveal much information about what is transmitted. The initial handshake TLS client hello message is the only cleartext message transmitted. All further data and DNS requests are fully encrypted. The difference now compared to the DoQ request is that the application protocol is recognized as h3 in the ALPN field, which is the abbreviation for HTTP/3. This can avoid blocking based on the application layer protocol field. The proxy's hostname is still visible.

The forwarded traffic has all the properties of DoQ as shown in Figure 45. The protocol can be identified by the port and the ALPN field which are exposed during the client hello. The destination resolver's hostname is also visible. The key property introduced by the MASQUE proxy is that the client's IP address is replaced with that of the proxy, thus the resolver will only see the request coming from the proxy. The proxy itself knows the client and resolver but has no access to the DNS query.

7.3 Summary of the Evaluation

The following table summarizes all evaluated DNS protocols.

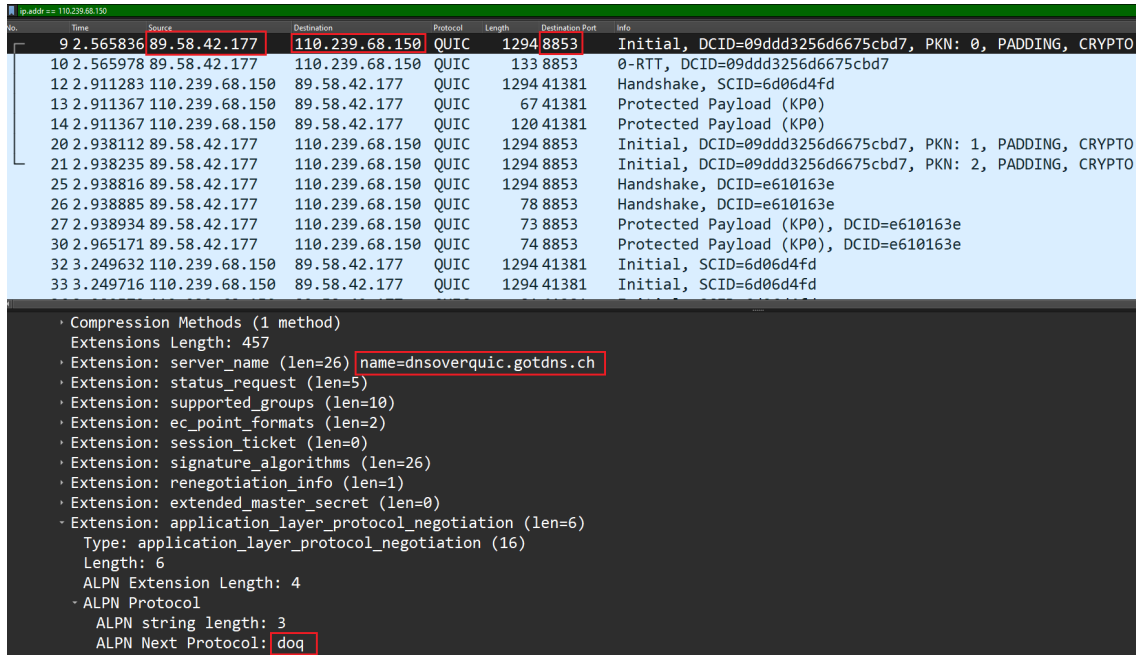


Figure 45. Traffic capture from the MASQUE proxy filtered to show only the forwarded DoQ traffic.

Category	Metric	DoH	DoQ	ODoH	DoQ+M
Performance	Round Trips	3	1-2	3 + 3	1-2 + 1-2
Privacy	Exposed Client Identity	Yes	Yes	No	No
	Exposed DNS Query	No	No	No	No
	Exposed Metadata	Yes	Yes	No	Yes
Security	Risk of Tampering	No	No	No	No
	Risk of DNS Server IP Blocking	Yes	Yes	No	No
	Risk of Destination Hostname Blocking	Yes	Yes	No	No
	Risk of Protocol Blocking	No	Yes	No	No

Table 5. Evaluation metrics for all evaluated DNS protocols.

7.4 Comparison to ODoH

The demonstrated setup has many similarities to ODoH. There is a similar split of information, the proxy knows the identity and the resolver the query. It also relies on non-collusion between the proxy and the resolver.

Coupling a MASQUE proxy with DoQ reduces the number of handshakes required to resolve a DNS query. This optimization can be further enhanced by utilizing 0-RTT resume with DoQ. However, it's important to note that 0-RTT DoQ is susceptible to replay attacks even if this attack is not straightforward as demonstrated in subsection 5.2.2. Fortunately, combining 0-RTT DoQ with a MASQUE proxy effectively counters this vulnerability to a big degree. The MASQUE proxy replaces the client's IP address in the initial 0-RTT

DoQ request which prevents attackers from linking the request back to the client and thus rendering the replay attack ineffective. It has to be noted that with the current state of technology, the risk of a replay attack by the proxy itself remains.

One downside of this new combination of protocols is that potentially more metadata is exposed since the QUIC and TLS session is not terminated and exists between the client and resolver. ODoH streams are not E2E, instead the proxy works on the application layer and only forwards the encrypted DNS query. MASQUE proxied streams on the contrary don't get intercepted. Depending on the specific MASQUE method, the proxy will forward the full Ethernet, IP, or UDP packet as explained in subsection 6.1.1 which makes it an E2E stream. Due to this factor, there is an exchange of transport and security parameters between the client and resolver. This information could potentially be used for fingerprinting on the resolver. Thus it is recommended to use default TLS settings and minimize information present in TLS extensions and QUIC transport parameters.

Furthermore, when DoQ is used, it can easily be identified and blocked after the proxy based on the port and the ALPN header in the TLS handshake. Here, DoH/3 could be used instead of DoQ. DoH/3 is transmitted over the regular HTTP/3 port and also identifiable as HTTP/3 from the acALPN header which helps to conceal the DNS query.

7.5 Future improvements

ODoH adoption has been slow so far and the protocol requires a specialized setup. MASQUE on the contrary has gained a lot of attention recently and is already being adopted by many of the big players in the tech industry. MASQUE is a more flexible upcoming technology with a lot of attention focused on it. Many of the big tech companies are currently working on their own MASQUE-based proxy frameworks. The industry is moving towards MASQUE. Eventually encrypted DNS will also move to MASQUE. MASQUE can support many different applications, and thus there will probably also be a large number of MASQUE proxies available. This may also benefit privacy, a bigger choice of relays allows users to frequently rotate between proxies. This, in turn, makes consequences less severe in the case of a compromised resolver and proxy.

subsection 5.2.2 explains some of the security concerns with 0-RTT. There are already multiple approaches implemented in TLS [54] to limit the threat of replay attacks. Göth et al. [42] go even further by proposing new cryptographic methods that can achieve forward secrecy for 0-RTT without adding much additional cost.

Encrypted Client Hello (ECH) is another major advancement for network technology

[55]. With ECH the sensitive data in the TLS client hello message can be concealed. Cloudflare called it the last puzzle piece to privacy [56] in a blog post. ECH is already part of Chromium ¹ and Firefox. ECH is made possible by two big new technologies that have a lot of resemblance to ODoH. It introduces a new method to access and distribute public keys via DNS [57]. These public keys of the server can then be used with the same lightweight HPKE encryption scheme that is part of oblivious DNS to encrypt the Client Hello Messages in new connections. This is an important step for user privacy. With ECH, fingerprinting, tracking, and SNI based filtering becomes much harder to implement, but it does change the fact that the target can still see all the information.

Another great feature of MASQUE is that proxies can easily be chained. Adding a second proxy hop to the DNS resolution request can further split up the trust between parties. This in turn, makes it harder to reverse the connection from an observer's perspective. This dual-hop architecture is also used by Apple in their private relay service.

One MASQUE-related feature which is still actively being worked on in standardization, proposes optimizations for tunneling QUIC in HTTP/3 [58]. The draft is titled QUIC-aware proxying using HTTP/3. It proposes adding new signaling so that double encryption can be avoided. This would simplify the tunneling of encrypted protocols such as QUIC in HTTP/3 MASQUE. This proposal is still in discussion though as there have been multiple IETF participants voicing criticism on it.²

¹<https://chromestatus.com/feature/6196703843581952>

²<https://www.youtube.com/watch?v=pj6ufe9r3Hc>

8. Summary

The technologies of the internet are evolving towards encrypting all traffic. Sending cleartext messages that can be deciphered by any man in the middle observer is slowly being phased out. DNS is currently one of the last major technologies that is still mostly used in cleartext form.

In this thesis, an analysis of existing DNS protocols was performed. More specific, the initial handshakes up to the first query resolution were analyzed. An experimental setup was chosen and metrics were defined for the analysis. Multiple encrypted DNS solutions were evaluated based on their performance, privacy and security metrics.

From there it was possible to determine the differences between the protocols. The analysis showed that there is a tradeoff between performance and security + privacy. The ODoH protocol was shown to provide great privacy to users while the protocol blends in well with regular HTTP traffic. It introduces a new form of client anonymity at the cost of a more extensive handshake.

Leveraging recent advancements in internet protocols, this work explores MASQUE, a novel technology offering client anonymity similar to existing solutions. The implemented system, combining a MASQUE proxy with DNS over QUIC, achieves similar anonymity to ODoH while requiring a significantly shorter handshake. However, the impact of this on the overall DNS query resolution time needs to be analyzed in future research.

One major challenge of this thesis was the limited amount of available implementations. Moreover, the implementations that exist often lack critical features, are usually poorly documented, and are often labeled as not production-ready. Another challenge was the rapid evolution of the technologies and new releases are constantly reshaping the field. Multiple papers, implementations and standards were released or received big updates during the writing of this thesis.

In conclusion, this thesis demonstrated that a MASQUE proxy offers vast performance benefits compared to ODoH while providing a similar level of privacy. The proposed new combination of protocols can be an option for users who are looking for an upgraded version of ODoH.

One key drawback that stems from MASQUE proxies properties, is that the TLS sessions are not terminated. This thesis does not provide a deep analysis of the privacy impacts this can have. Not using an E2E connection is a big feature of ODoH. Thus the logical next step following this research would be to evaluate an ODoH version that supports HTTP/3 and does proxying on the application layer. As it was discovered that a proxy can mitigate some of the security concerns with using 0-RTT for DNS, an Oblivious version of DoH/3 should also be analyzed with 0-RTT.

References

- [1] P. Mockapetris. *Domain names - implementation and specification*. RFC 1035. Nov. 1987. DOI: 10.17487/RFC1035. URL: <https://www.rfc-editor.org/info/rfc1035>.
- [2] Eric Kinnear et al. *Oblivious DNS over HTTPS*. RFC 9230. June 2022. DOI: 10.17487/RFC9230. URL: <https://www.rfc-editor.org/info/rfc9230>.
- [3] Sudheesh Singanamalla et al. “Oblivious dns over https (odoh): A practical privacy enhancement to dns”. In: *arXiv preprint arXiv:2011.10121* (2020).
- [4] Mike Kosek et al. “DNS privacy with speed? Evaluating DNS over QUIC and its impact on web performance”. In: *Proceedings of the 22nd ACM Internet Measurement Conference*. 2022, pp. 44–50.
- [5] Jayasree Sengupta et al. “On Cross-Layer Interactions of QUIC, Encrypted DNS and HTTP/3: Design, Evaluation and Dataset”. In: *arXiv preprint arXiv:2306.11643* (2023).
- [6] Tim Wicinski. *DNS Privacy Considerations*. RFC 9076. July 2021. DOI: 10.17487/RFC9076. URL: <https://www.rfc-editor.org/info/rfc9076>.
- [7] Georgios Kambourakis and Georgios Karopoulos. “Encrypted DNS: The good, the bad and the moot”. In: *Computer Fraud & Security* 2022.5 (2022).
- [8] Nicholas Weaver, Christian Kreibich, and Vern Paxson. “Redirecting {DNS} for Ads and Profit”. In: *USENIX Workshop on Free and Open Communications on the Internet (FOCI 11)*. 2011.
- [9] The Guardian. *Turkey blocks YouTube amid 'national security' concerns*. Mar. 27, 2014. URL: <https://www.theguardian.com/world/2014/mar/27/google-youtube-ban-turkey-erdogan> (visited on 11/04/2023).
- [10] Nathalia Sautchuk-Patricio. *Content Blocking at the DNS Level in Germany*. Nov. 8, 2021. URL: <https://circleid.com/posts/20211108-content-blocking-at-the-dns-level-in-germany> (visited on 11/04/2023).
- [11] Miguel Barreda-Ángeles et al. “Unconscious physiological effects of search latency on users and their click behaviour”. In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2015, pp. 203–212.

- [12] Ioannis Arapakis, Souneil Park, and Martin Pielot. “Impact of response latency on user behaviour in mobile web search”. In: *Proceedings of the 2021 Conference on Human Information Interaction and Retrieval*. 2021, pp. 279–283.
- [13] Internet Architecture Board. *IAB Statement on Internet Confidentiality*. Nov. 13, 2014. URL: <https://datatracker.ietf.org/doc/statement-iab-statement-on-internet-confidentiality/> (visited on 05/10/2024).
- [14] Minzhao Lyu, Hassan Habibi Gharakheili, and Vijay Sivaraman. “A Survey on DNS Encryption: Current Development, Malware Misuse, and Inference Techniques”. In: *ACM Comput. Surv.* 55.8 (2022). ISSN: 0360-0300. DOI: 10.1145/3547331. URL: <https://doi.org/10.1145/3547331>.
- [15] Trinh Viet Doan, Irina Tsareva, and Vaibhav Bajpai. “Measuring DNS over TLS from the edge: adoption, reliability, and response times”. In: *Passive and Active Measurement: 22nd International Conference, PAM 2021, Virtual Event, March 29–April 1, 2021, Proceedings 22*. Springer. 2021, pp. 192–209.
- [16] Sebastián García et al. “Large scale measurement on the adoption of encrypted DNS”. In: *arXiv preprint arXiv:2107.04436* (2021).
- [17] Guannan Hu and Kensuke Fukuda. “Privacy Leakage of DNS over QUIC: Analysis and Countermeasure”. In: *2024 International Conference on Artificial Intelligence in Information and Communication (ICAIC)*. IEEE. 2024, pp. 518–523.
- [18] Dmitrii Vekshin, Karel Hynek, and Tomas Cejka. “Doh insight: Detecting dns over https by machine learning”. In: *Proceedings of the 15th International Conference on Availability, Reliability and Security*. 2020, pp. 1–8.
- [19] Péter Megyesi, Zsolt Krämer, and Sándor Molnár. “How quick is QUIC?” In: *2016 IEEE International Conference on Communications (ICC)*. IEEE. 2016, pp. 1–6.
- [20] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. May 2021. DOI: 10.17487/RFC9000. URL: <https://www.rfc-editor.org/info/rfc9000>.
- [21] Martin Thomson, Mark Nottingham, and Willy Tarreau. *Using Early Data in HTTP*. RFC 8470. Sept. 2018. DOI: 10.17487/RFC8470. URL: <https://www.rfc-editor.org/info/rfc8470>.
- [22] Martin Thomson and Sean Turner. *Using TLS to Secure QUIC*. RFC 9001. May 2021. DOI: 10.17487/RFC9001. URL: <https://www.rfc-editor.org/info/rfc9001>.
- [23] Constantin Sander. *HTTP/3 and QUIC — prioritization and head-of-line blocking*. Nov. 30, 2022. URL: <https://blog.apnic.net/2022/11/30/http-3-and-quic-prioritization-and-head-of-line-blocking/>.

- [24] Robin Marx. *Why HTTP/3 is Eating the World*. June 2023. URL: <https://pulse.internetsociety.org/blog/why-http-3-is-eating-the-world>.
- [25] Christian Huitema, Sara Dickinson, and Allison Mankin. *DNS over Dedicated QUIC Connections*. RFC 9250. May 2022. DOI: 10.17487/RFC9250. URL: <https://www.rfc-editor.org/info/rfc9250>.
- [26] Vasily Bagirov. *AdGuard DNS-over-QUIC*. Dec. 15, 2020. URL: <https://adguard-dns.io/en/blog/dns-over-quic.html> (visited on 11/04/2023).
- [27] Aurelija Einorytė. *VPN bans: How do they work and who initiates them?* Feb. 2024. URL: <https://nordvpn.com/de/blog/vpn-ban/>.
- [28] Yunming Xiao et al. “PDNS: A Fully Privacy-Preserving DNS”. In: *Proceedings of the ACM SIGCOMM 2023 Conference*. 2023, pp. 1182–1184.
- [29] Sudheesh Singanamalla Tanya Verma. *Improving DNS Privacy with Oblivious DoH in 1.1.1.1*. Dec. 8, 2020. URL: <https://blog.cloudflare.com/oblivious-dns> (visited on 11/04/2023).
- [30] Paul Schmitt, Anne Edmundson, and Nick Feamster. “Oblivious DNS: Practical privacy for DNS queries”. In: *arXiv preprint arXiv:1806.00276* (2018).
- [31] Richard Barnes et al. *Hybrid Public Key Encryption*. RFC 9180. Feb. 2022. DOI: 10.17487/RFC9180. URL: <https://www.rfc-editor.org/info/rfc9180>.
- [32] Christopher Wood. *HPKE: Standardizing public-key encryption (finally!)* May 25, 2022. URL: <https://blog.cloudflare.com/hybrid-public-key-encryption> (visited on 11/04/2023).
- [33] Martin Thomson and Christopher A. Wood. *Oblivious HTTP*. RFC 9458. Jan. 2024. DOI: 10.17487/RFC9458. URL: <https://www.rfc-editor.org/info/rfc9458>.
- [34] Martin Duke. *Multiplexed Application Substrate over QUIC Encryption WG*. Feb. 2023. URL: <https://datatracker.ietf.org/doc/charter-ietf-masque/>.
- [35] Alejandro Sedeño. *Proxying Ethernet in HTTP*. Internet-Draft draft-ietf-masque-connect-ethernet-02. Work in Progress. Internet Engineering Task Force, Apr. 2024. 14 pp. URL: <https://datatracker.ietf.org/doc/draft-ietf-masque-connect-ethernet/02/>.
- [36] David Schinazi. *Proxying UDP in HTTP*. RFC 9298. Aug. 2022. DOI: 10.17487/RFC9298. URL: <https://www.rfc-editor.org/info/rfc9298>.

- [37] Tommy Pauly, Eric Kinnear, and David Schinazi. *An Unreliable Datagram Extension to QUIC*. RFC 9221. Mar. 2022. DOI: 10.17487/RFC9221. URL: <https://www.rfc-editor.org/info/rfc9221>.
- [38] David Schinazi and Lucas Pardue. *HTTP Datagrams and the Capsule Protocol*. RFC 9297. Aug. 2022. DOI: 10.17487/RFC9297. URL: <https://www.rfc-editor.org/info/rfc9297>.
- [39] Tommy Pauly et al. *Proxying IP in HTTP*. RFC 9484. Oct. 2023. DOI: 10.17487/RFC9484. URL: <https://www.rfc-editor.org/info/rfc9484>.
- [40] SIDN. *New DNS over QUIC protocol makes encrypted DNS traffic faster and more efficient*. Aug. 25, 2022. URL: <https://www.sidn.nl/en/news-and-blogs/new-dns-over-quic-protocol-makes-encrypted-dns-traffic-faster-and-more-efficient> (visited on 04/14/2024).
- [41] Mike Kosek et al. “One to rule them all? a first look at dns over quic”. In: *International Conference on Passive and Active Network Measurement*. Springer. 2022, pp. 537–551.
- [42] Christian Göth et al. “Optimizing 0-RTT Key Exchange with Full Forward Security”. In: *Proceedings of the 2023 on Cloud Computing Security Workshop*. 2023, pp. 55–68.
- [43] Rashna Kumar and Fabián E Bustamante. “Reclaiming Privacy and Performance over Centralized DNS”. In: *arXiv preprint arXiv:2302.13274* (2023).
- [44] *DNSCrypt proxy implementation*. URL: <https://github.com/DNSCrypt/dnscrypt-proxy/wiki/Oblivious-DoH> (visited on 03/18/2024).
- [45] Patrick Sattler et al. “Towards a tectonic traffic shift? investigating Apple’s new relay network”. In: *Proceedings of the 22nd ACM Internet Measurement Conference*. 2022, pp. 449–457.
- [46] Christopher Wood Lucas Pardue. *Unlocking QUIC’s proxying potential with MASQUE*. Mar. 20, 2022. URL: <https://blog.cloudflare.com/unlocking-quic-proxying-potential> (visited on 11/04/2023).
- [47] Mirja Kühlewind et al. “Evaluation of QUIC-Based MASQUE Proxying”. In: *Proceedings of the 2021 Workshop on Evolution, Performance and Interoperability of QUIC*. EPIQ ’21. Virtual Event, Germany: Association for Computing Machinery, 2021, pp. 29–34. ISBN: 9781450391351. DOI: 10.1145/3488660.3493806. URL: <https://doi.org/10.1145/3488660.3493806>.
- [48] Dan Hall. *Zero Trust WARP: tunneling with a MASQUE*. June 22, 2023. URL: <https://blog.cloudflare.com/zero-trust-warp-with-a-masque> (visited on 03/27/2024).

- [49] Simon Kuhn. *Enabling privacy on the Internet with Oblivious HTTP*. Mar. 23, 2023. URL: <https://www.fastly.com/blog/enabling-privacy-on-the-internet-with-oblivious-http> (visited on 04/07/2024).
- [50] Stephen Nellis and Paresh Dave. “Apple’s new “private relay” feature will not be available in China”. In: *Reuters* (June 2021). URL: <https://www.reuters.com/world/china/apples-new-private-relay-feature-will-not-be-available-china-2021-06-07/>.
- [51] Martino Trevisan et al. “Measuring the performance of icloud private relay”. In: *International Conference on Passive and Active Network Measurement*. Springer. 2023, pp. 3–17.
- [52] Heiko Kiesel. *TrustMeRelay? Investigating Apple’s iCloud Private Relay*. URL: https://media.ccc.de/v/camp2023-57214-trustmerelay_investigating_apple_s_icloud_private_relay (visited on 03/28/2024).
- [53] INVISV. *Why VPNs are Wrong and MPRs are Right*. Sept. 15, 2022. URL: <https://invisv.com/articles/relay> (visited on 05/10/2024).
- [54] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446. URL: <https://www.rfc-editor.org/info/rfc8446>.
- [55] Eric Rescorla et al. *TLS Encrypted Client Hello*. Internet-Draft draft-ietf-tls-esni-18. Work in Progress. Internet Engineering Task Force, Mar. 2024. 51 pp. URL: <https://datatracker.ietf.org/doc/draft-ietf-tls-esni/18/>.
- [56] Rushil Mehra Alessandro Ghedini Christopher Wood. *Encrypted Client Hello - the last puzzle piece to privacy*. Sept. 29, 2023. URL: <https://blog.cloudflare.com/encrypted-client-hello> (visited on 05/07/2024).
- [57] Benjamin M. Schwartz, Mike Bishop, and Erik Nygren. *Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records)*. RFC 9460. Nov. 2023. DOI: 10.17487/RFC9460. URL: <https://www.rfc-editor.org/info/rfc9460>.
- [58] Tommy Pauly, Eric Rosenberg, and David Schinazi. *QUIC-Aware Proxying Using HTTP*. Internet-Draft draft-ietf-masque-quic-proxy-00. Work in Progress. Internet Engineering Task Force, Aug. 2023. 24 pp. URL: <https://datatracker.ietf.org/doc/draft-ietf-masque-quic-proxy/00/>.

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis¹

I Leonard Walter

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “evaluating the use of masque proxies for achieving dns privacy”, supervised by Shaymaa Mamdouh Khalil and Silver Saks
 - 1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
 - 1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons’ intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

12.05.2024



¹The non-exclusive licence is not valid during the validity of access restriction indicated in the student’s application for restriction on access to the graduation thesis that has been signed by the school’s dean, except in case of the university’s right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 - Configuration Files and Code Modifications

Caddy server config

```
v2202403219344260446.goodsrv.de {
    root * /var/www/html/
    file_server
    reverse_proxy /dns-query 127.0.0.1:448 { transport http }
    log { output file /var/log/access.log }
}
```

Routing setup for the MASEQUE proxy server

```
iptables -I FORWARD 1 --in-interface tun0 -o eth0 -j ACCEPT
iptables -I FORWARD 2 --in-interface eth0 -o tun0 -j ACCEPT
iptables -t nat -I POSTROUTING -p all -s 10.1.1.2 -j SNAT \
    --to-source 89.58.42.177
```

Routing setup to only forward traffic from Routedns docker to the MASQUE tunnel

```
ip rule add from 172.17.0.2 table 1
ip route add 10.1.1.0/24 dev tun0 scope link table 1
ip route add default via 10.1.1.1 dev tun0 table 1
```

Diff file for ODOH client TLS secrets logging and query time measurements

```
odoh-client-go/commands/request.go
15a16,17
> "os"
> "time"
148a149,160
>
> fmt.Println("TLS secret logging enabled")
> keyLogFile, err2 := os.OpenFile("/var/tls.key", \
    os.O_WRONLY|os.O_APPEND|os.O_CREATE, 0666)
> if err2 != nil {
>     fmt.Println("unable to open TLS key log file: %s", err2)
```

```

>   }
>   tlsConfiguredTransport := &http.Transport{
>       TLSClientConfig: &tls.Config{KeyLogWriter: keyLogFile},
>   }
>   client.Transport = tlsConfiguredTransport
>
171a184
>   start := time.Now()
202a216
>   fmt.Println("Resolution time: \n", time.Since(start))

```

Diff file for ODOH proxy TLS secrets logging

```

odoh-server-go/proxy.go
31a32,33
>   "os"
>   "crypto/tls"
46a49,60
>   var tlsConfig *tls.Config
>   tlsConfig = new(tls.Config)
>   keyLogFile, err1 := os.OpenFile("/var/tls.key", \
>       os.O_WRONLY|os.O_APPEND|os.O_CREATE, 0666)
>   if err1 != nil {
>       log.Println("unable to open TLS key log file: %s", err1)
>   }
>
>   tlsConfig.KeyLogWriter = keyLogFile
>   transport := &http.Transport{TLSClientConfig: tlsConfig}
>   client.Transport = transport

```

Diff file for Routedns DoH TLS secrets logging and 0-RTT

```

routedns/dohclient.go
8a9
>   "os"
184c185,190
<   req, err := http.NewRequestWithContext(ctx, "GET", u, nil)
---
>   method := http.MethodGet

```

```

> if d.opt.Transport == "quic" {
> method = http3.MethodGet0RTT
> }
>
> req, err := http.NewRequestWithContext(ctx, method, u, nil)
270a277,286
>
> Log.Debug("TLS secret logging enabled")
> keyLogFile, err := os.OpenFile("/var/tls.key", \
    os.O_WRONLY|os.O_APPEND|os.O_CREATE, 0666)
> if err != nil {
>     Log.Errorf("unable to open TLS key log file: %s", err)
> }
> tlsConfig.KeyLogWriter = keyLogFile
>
> // enable TLS session caching for session resumption and 0-RTT
> tlsConfig.ClientSessionCache = tls.NewLRUClientSessionCache(100)
424a441
> Log.Debug("using dial early")

```

Diff file for Routedns DoQ TLS secrets logging and 0-RTT

```

7a8
> "os"
64a67,76
>
> Log.Debug("TLS secret logging enabled")
> keyLogFile, err := os.OpenFile("/var/tls.key", \
    os.O_WRONLY|os.O_APPEND|os.O_CREATE, 0666)
> if err != nil {
>     Log.Errorf("unable to open TLS key log file: %s", err)
> }
> tlsConfig.KeyLogWriter = keyLogFile
>
78a91,97
> // enable TLS session caching for session resumption and 0-RTT
> tlsConfig.ClientSessionCache = tls.NewLRUClientSessionCache(100)
>
212c228,242

```

```

< s.EarlyConnection, s.udpConn, err = quicDial(context.TODO(), \
    s.hostname, endpoint, s.lAddr, s.tlsConfig, s.config)
---
> s.udpConn, err = net.ListenUDP("udp", nil)
> if err != nil {
>     log.WithError(err).Debug("couldn't create UDP connection")
>     return nil, err
> }
>
> // Resolve the UDP address for the endpoint
> udpAddr, err := net.ResolveUDPAddr("udp", endpoint)
> if err != nil {
>     log.WithError(err).Debug("couldn't resolve UDP address for \
    endpoint [" + endpoint + "]")
>     return nil, err
> }
>
> // Use quic.DialEarly to attempt to use 0-RTT DNS queries
> s.EarlyConnection, err = quic.DialEarly(context.TODO(), \
    s.udpConn, udpAddr, s.tlsConfig, s.config)

```