

# QUIC-Exfil: Exploiting QUIC’s Server Preferred Address Feature to Perform Data Exfiltration Attacks

Thomas Grübl

Communication Systems Group CSG, Department of  
Informatics IfI, University of Zurich UZH  
Zürich, Switzerland  
gruebl@ifi.uzh.ch

Jan von der Assen

Communication Systems Group CSG, Department of  
Informatics IfI, University of Zurich UZH  
Zürich, Switzerland  
vonderassen@ifi.uzh.ch

Weijie Niu

Communication Systems Group CSG, Department of  
Informatics IfI, University of Zurich UZH  
Zürich, Switzerland  
niu@ifi.uzh.ch

Burkhard Stiller

Communication Systems Group CSG, Department of  
Informatics IfI, University of Zurich UZH  
Zürich, Switzerland  
stiller@ifi.uzh.ch

## Abstract

The QUIC protocol is now widely adopted by major tech companies and accounts for a significant fraction of today’s Internet traffic. QUIC’s multiplexing capabilities, encrypted headers, dynamic IP address changes, and encrypted parameter negotiations make the protocol not only more efficient, secure, and censorship-resistant, but also practically unmanageable by firewalls. This opens up doors for attackers that may exploit certain traits of the QUIC protocol to perform targeted attacks, such as data exfiltration attacks. Whereas existing data exfiltration techniques, such as TLS and DNS-based exfiltration, can be detected on a firewall level, QUIC-based data exfiltration is more difficult to detect, since changes in IP addresses and ports are inherent to the protocol’s normal behaviour.

To show the feasibility of a QUIC-based data exfiltration attack, we first introduce a novel method which leverages the server preferred address feature of the QUIC protocol and, thus, allows an attacker to exfiltrate sensitive data from an infected machine to a malicious server, disguised as a server-side connection migration. The attack is implemented in the form of a proof of concept tool in Rust. We evaluated the performance of five anomaly detection classifiers – Random Forest, Multi-Layer Perceptron, Support Vector Machine, Autoencoder, and Isolation Forest – trained on datasets collected from three distinct network traffic scenarios. The classifiers were trained on ~ 700K benign and malicious QUIC packets and 786 connection migration events, but were unable to effectively detect the data exfiltration attempts. Furthermore, post-analysis of the traffic captures did not reveal any identifiable fingerprint. As part of our evaluation, we also interviewed five leading firewall vendors and found that, as of today, no major firewall vendor implements functionality capable of distinguishing between benign and malicious QUIC connection migrations.

## CCS Concepts

• **Security and privacy** → **Network security**; *Intrusion/anomaly detection and malware mitigation*; • **Networks** → **Transport protocols**.

## Keywords

QUIC, Network Security, Data Exfiltration, Anomaly Detection

## ACM Reference Format:

Thomas Grübl, Weijie Niu, Jan von der Assen, and Burkhard Stiller. 2025. QUIC-Exfil: Exploiting QUIC’s Server Preferred Address Feature to Perform Data Exfiltration Attacks. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS ’25)*, August 25–29, 2025, Hanoi, Vietnam. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3708821.3733872>

## 1 Introduction

The QUIC protocol, originally developed by [39], is an IETF standardized, connection-oriented, UDP-based transport protocol that serves as a replacement to TLS over TCP [24]. Its main benefits include One Round Trip Time (1-RTT) handshakes, Zero Round Trip Time (0-RTT) connection re-establishments when prior communication has taken place, seamless network path migration, and state-of-the-art security [19]. As of today, QUIC has been widely deployed by major tech companies such as Google, Meta, Apple, and Cloudflare. The popularity of QUIC has led to some application-layer protocols adopting it as their main transport protocol, such as DNS-over-QUIC [17] or HTTP/3 [4]. QUIC plays a fundamental role within HTTP/3 [4], making it ubiquitous in today’s Internet traffic. In 2023, HTTP/3 already accounted for around 30% of HTTP requests, with adoption expected to continue growing as more web servers and browsers integrate QUIC support [3].

QUIC packets are encapsulated in UDP datagrams. Each packet has a QUIC header that contains details such as the flags indicating a certain packet type and the Connection ID (CID). The CID is a randomly generated, variable-length identifier that endpoints use to demultiplex network traffic sent between them [19]. Since the QUIC protocol encrypts the entire communication, including a significant part of the handshake, it is inherently difficult to analyze the underlying traffic. Although QUIC was primarily designed for performance improvements, QUIC’s properties also enhance privacy and make it more resilient to network interference through



features such as the frequent rotation of CIDs and the concurrent transmission of packets over multiple streams [29].

While these features improve the security and privacy for end-users, they pose challenges for firewall vendors. The general consensus among firewall vendors and administrators appears to be that QUIC-based traffic should be blocked because it renders Deep Packet Inspection (DPI) and Intrusion Detection/Prevention Systems (IDS/IPS) useless (*cf.* Section 6). Typically, the communication between client and server is then forced to fall back to TLS over TCP, allowing enterprise firewalls to perform HTTPS DPI [6]. As of now, firewall vendors do not provide any specifics on how enterprise-level firewalls can filter QUIC traffic effectively.

Some inherent traits of the QUIC protocol, such as the connection migration capability and the encrypted handshake, make QUIC-based data exfiltration techniques arguably more devious than TLS or DNS-based data exfiltration because middleboxes are not able to differentiate between benign and spoofed connection migration attempts. It is, therefore, crucial to understand how the underlying properties of the QUIC protocol can impact the success of data exfiltration attacks.

The goal of data exfiltration is to covertly extract sensitive data from an infected host. There exist many methods to exfiltrate data — depending on the underlying protocol used, some are more effective than others. According to the MITRE ATT&CK framework [31], data exfiltration attacks can be broadly divided into network-based and physical exfiltration methods. The former leverages widely-used network protocols, such as HTTP, HTTPS, or DNS, to send data to a target device owned by a malicious actor. As part of the latter, adversaries may use removable storage, such as external hard drives or USB drives, to exfiltrate data. Effectiveness in this context can be seen as a combination of data throughput and covertness of the method. The severity of data exfiltration attacks becomes apparent when considering the financial impact of data breaches. The average cost of data breaches has been consistently rising over the last years, with current estimations reaching a global all-time high of 4.88 million USD per data breach. The top three industries to experience the highest data breach costs are healthcare, financial, and industrial [18].

In this paper, we specifically focus on QUIC’s server-side connection migration feature. We show that an attack can be designed that replicates the behavior of this feature to covertly exfiltrate data to a target server. The attack can be leveraged by adversaries that have already infected a host machine. From the perspective of a middlebox, the exfiltration payload packets are not distinguishable from packets that originate from a benign server-side connection migration. While we initially explored classifiers as a potential defense strategy to detect QUIC-based data exfiltration traffic, our findings indicate that machine learning-based classifiers are ineffective in reliably distinguishing the attack from legitimate traffic. To the best of our knowledge, this paper provides the first analysis of QUIC-based data exfiltration attacks and potential mitigations. The **contributions** of the paper can be summarized as follows:

- ✓ We develop a QUIC-based data exfiltration method that exploits the QUIC connection migration feature by mimicking a server-side path migration procedure to exfiltrate sensitive data from an infected host machine. The method does not

require any changes to the QUIC client or server applications and is designed based on the version-independent properties of IETF QUIC.

- ✓ We implement a PoC prototype in Rust, which comprises a packet sniffer and a custom QUIC parser and mimics benign packet features such as payload length, entropy, time differences between outgoing packets as well as the server-side connection migration packets.
- ✓ We show that due to the limited number of visible features available in QUIC traffic, it is difficult to differentiate between benign and malicious QUIC traffic. Our custom detectors — Random Forest, Multi-Layer Perceptron, Support Vector Machine, Autoencoder, and Isolation Forest — were trained on over 700K QUIC packets and 786 server-side connection migration events, collected across three distinct network traffic scenarios.
- ✓ In addition, we conducted unstructured interviews with five leading firewall vendors to gain insights into the current state of their QUIC traffic filtering capabilities.

First, we begin with the background (Section 2) and the problem statement (Section 3). Section 4 presents the data exfiltration method and the threat model. Section 5 describes the PoC implementation written in Rust. Section 6 presents the experimental setup, the anomaly detection results and the survey of leading firewall vendors, followed by a brief discussion of additional mitigation strategies in Section 7, the related work in Section 8, and the conclusion in Section 9.

## 2 Background

This section briefly introduces the main characteristics of the QUIC protocol, focusing on its header structures and the connection migration features for both client-side and server-side migrations since those are the most relevant aspects for the attack. The description of the packet headers mainly relies on the version-independent properties of QUIC, defined in RFC 8999 [33], and connection migrations are described based on RFC 9000 [19].

### 2.1 A QUIC Overview

QUIC packets generally come in two different forms: long header and short header packets. Long header packets are sent as part of the handshake messages before 1-RTT keys are established. Thereafter, the QUIC endpoints switch to sending short header packets [19]. Long header packets include version-specific bits, the version number, CIDs for both source and destination as well as their respective lengths, other version-dependent data, and the payload itself. In comparison, short header packets only contain version-specific bits, the Destination Connection ID (DCID), version-dependent data, and the payload, meaning that the length of the CID cannot be determined from a short header packet alone. Without observing the initial long header packets, the necessary context for understanding short header packets, *e.g.*, which connection they belong to, cannot be established. All QUIC endpoints, therefore, need to keep track of the CIDs in use [33].

## 2.2 Connection Migration

Compared to traditional TCP and UDP connections, where each connection is uniquely identifiable by its 5-tuple (source IP address, destination IP address, source port, destination port, protocol), a QUIC connection may change its underlying 5-tuple without establishing a new connection using a handshake. This is referred to as connection migration or (network) path migration. The primary benefit of QUIC connection migrations lies in the reduction of Round Trip Times (RTT) since it eliminates performing handshakes repeatedly whenever an endpoint changes IP address. This feature is particularly time-saving when a QUIC endpoint migrates multiple times [25].

Client-side connection migration is a feature within the IETF QUIC standard that allows clients to change their source IP address while retaining an existing connection with a QUIC server. This process is presented in Fig. 1. The CM Initiator (*i.e.*, client) detects a source IP address change and sends a *PATH\_CHALLENGE* frame to the CM Responder (*i.e.*, server). The server responds with a *PATH\_RESPONSE* frame, confirming receipt of the challenge and the viability of the new path. The number of probing packet round trips is version-dependent. Once the *PATH\_RESPONSE* is received by the client, the client may use the new source IP address for all further (non-probing) communication with the server. Client-side connection migrations are beneficial in cases where a QUIC endpoint moves from Wi-Fi to the cellular network or vice versa [19].

Server-side connection migration, like client-side migration, ensures that a QUIC connection can continue seamlessly whenever the server’s IP address changes mid-connection. The server preferred\_address transport parameter is a data structure within the quic\_transport\_parameters extension, which defines a secondary (preferred) server address. It is sent from a QUIC server to a client as part of the handshake. The parameter contains fields for an IPv4 Address, IPv4 Port, IPv6 Address, IPv6 Port, CID Length, CID, and the Stateless Reset Token (*cf.* Fig. 2). A server may specify a preferred IPv4 and/or IPv6 address during the initial handshake, that

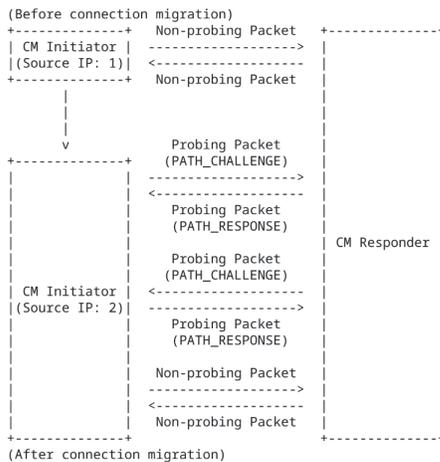


Figure 1: Client-side QUIC Connection Migration [26]

the client can use to communicate with the server at any time after a successful connection establishment to the primary server address. To initiate a server-side connection migration, the client simply sends a path validation packet, containing a *PATH\_CHALLENGE* frame to the secondary address and waits for the server’s acknowledgment [19]. It is important to note that both client-side and server-side connection migrations are initiated by the client. The server-side connection migration can, for example, be used in microservice deployment at the network edge to seamlessly migrate a container hosting a QUIC-based service from one server address to a different one with minimum latency [35]. Another use case for server-side connection migration is splitting network traffic mid-session across different network paths to increase privacy. Wang et al. [45] introduce Connection Migration Powered Splitting (CoMPS), which helps to reduce the risk of traffic analysis attacks by network-level adversaries.

RFC 9000 [19] recommends reducing the linkability of QUIC connections by, inter alia, using a new CID and source port when migrating to another IP address. Consequently, a middlebox is unable to tell whether a client performed a connection migration, or if a second client is starting to communicate [15].

The prevalence of QUIC connection migrations has been shown by [5], who found that the top providers that support connection migrations are Cloudflare, AWS, Hostinger, Akamai and Google. However, connection migration is not yet supported by other major providers.

## 3 Problem Statement

In this section, we describe how server-side connection migrations can be mimicked to exfiltrate sensitive data from a device. We compare this approach to existing client-side request forgery attacks and highlight the shortcomings of current firewall technologies in analyzing QUIC traffic.

### 3.1 Connection Migration

As specified in RFC 9000 [19], “the use of a connection ID allows connections to survive changes to endpoint addresses (IP address and port), such as those caused by an endpoint migrating to a new network.” Consequently, both a client and a server can use different IP addresses to communicate while retaining an existing connection [25].

Further, a “server might receive a packet addressed to its preferred IP address at any time after it accepts a connection” [19]. Concretely, a client can choose to migrate the connection to a preferred server

```

Preferred Address {
  IPv4 Address (32),
  IPv4 Port (16),
  IPv6 Address (128),
  IPv6 Port (16),
  Connection ID Length (8),
  Connection ID (...),
  Stateless Reset Token (128),
}
    
```

Figure 2: Format of the Server Preferred Address Transport Parameter [19]

destination IP address and initiate the path migration by sending a QUIC packet encapsulating a *PATH\_CHALLENGE* frame to the new (preferred) server IP address. This feature can be exploited by an adversary who wants to mimic a QUIC server-side connection migration by changing the destination IP address of an outgoing QUIC packet and thus sending an illegitimate packet to the adversary’s server. The payload of such a packet can be arbitrarily modified so that it encapsulates sensitive data from the victim’s machine. As specified in [19], the preferred server address is sent from the server to the client as part of the encrypted handshake. Hence, it remains unknown to a middlebox. Therefore, a middlebox is unable to differentiate between legitimate preferred server addresses and malicious ones. It is important to note that migrating to a new path does not require a dedicated handshake [25, 41]. The path validation process is solely initiated by the client sending a “*PATH\_CHALLENGE frame containing an unpredictable payload on the path to be validated*” [19].

Similarly to [12], who evaluated the effectiveness of **client-side** Connection Migration Request Forgery (CMRF) attacks, our method shows that the **server-side** connection migration feature can be exploited to exfiltrate data from an infected client to a target server. RFC 9000 [19] describes “Request Forgery with Spoofed Migration” (*i.e.*, CMRF) attacks, where clients can spoof the source address of a QUIC packet as part of an apparent connection migration. This tricks the server into sending datagrams to the spoofed address [19]. However, [19] fails to mention that spoofing the destination address using the `preferred_address` transport parameter of a QUIC packet can be misused to covertly exfiltrate data from a QUIC endpoint to a malicious server — disguised as a server-side connection migration.

As per RFC 9000 [19], the `preferred_address` transport parameter may store a single secondary address for each address family (IPv4 and IPv6). There are Internet draft specifications that aim to increase the number of additional addresses that a server can use, such as the Multipath Extension for QUIC by [25]. According to [32], a QUIC frame can be specified which securely advertises additional server IP addresses that a client may use to communicate with the server. Such additional addresses are transmitted inside an `ADDITIONAL_ADDRESSES` frame, which is also encrypted with the rest of the handshake and thus remains invisible to a middlebox. Therefore, a malicious exfiltration program may specify an exfiltration server IP address as the destination IP, and a middlebox has to assume that this is a previously advertised server IP address.

### 3.2 Firewalls

It is inherent to the QUIC protocol that CID renegotiation happens in an encrypted way — out of sight of potential path-based network filtering devices such as firewalls. In order to keep track of a QUIC connection, a stateful firewall would need to store the Source Connection ID (SCID) during the handshake [11] and subsequently match the DCID of an outgoing packet with the previously stored SCID. However, changes in DCID are not visible to a middlebox, as renegotiation of CIDs can occur post-handshake inside the encrypted “Protected Payload” packets. QUIC does not expose any more unencrypted information on the connection migration process. Since path validation packets are not preceded by a dedicated

handshake and the connection can be migrated to another endpoint on both the client and the server-side at any time [19], firewalls would need to treat QUIC packets without a prior handshake as connection migration attempts, yielding UDP 5-tuple checks and CID tracking unfeasible.

If no full decryption of the QUIC traffic is done, it is almost impossible for a middlebox to differentiate between a legitimate connection migration and a spoofed one by only analyzing unencrypted packet headers. Currently, firewalls simply filter QUIC traffic based on the handshake packets, but may not be able to perform further stateful inspection after the handshake. Therefore, to further motivate this research, we have conducted unstructured interviews with five leading firewall vendors, who support our findings regarding the challenges of effectively analyzing QUIC traffic. They emphasize the difficulty in distinguishing between legitimate and spoofed connection migrations without decrypting QUIC traffic.

In summary, connection migration — a feature that enhances QUIC’s performance — can also be misused by the data exfiltration method proposed in this paper. In the following section, we present a methodology and the PoC implementation for demonstrating how QUIC’s connection migration feature can be exploited and we show that QUIC-based data exfiltration attacks are difficult to differentiate from the normal protocol behavior.

## 4 Methodology

In the following, we provide a comprehensive description of the proposed data exfiltration attack. After presenting an overview of the threat model and the underlying assumptions, we introduce the methodology, which comprises a sniffing phase, spoofed path validation, and a continued exfiltration phase.

### 4.1 Threat Model & Assumptions

We consider a scenario in which an attacker has already infected a victim’s machine and aims to covertly exfiltrate sensitive data to a target server owned by the attacker. The attack is performed as per the MITRE ATT&CK framework sub-technique “Exfiltration Over Alternative Protocol: Exfiltration Over Symmetric Encrypted Non-C2 Protocol” [31]. Circumventing on-device anomaly detection systems is out of the scope of this paper, as we only consider on-path middleboxes and additional on-path traffic analysis software to be the main defenders.

The underlying assumptions can be summarized as follows:

- (1) It is assumed that the adversary has successfully gained access to a compromised machine within the internal network of a victim. The compromised machine is *firewalled*, meaning it is located behind a host-based and/or enterprise-level network-based stateful firewall.
- (2) The adversary is also capable of secretly deploying the data exfiltration software onto the compromised machine and has gained elevated privileges to successfully execute the attack.
- (3) We assume that a defender (*e.g.*, the firewall administrator) does not enforce a strict packet inspection policy (*i.e.*, full decryption of traffic) and does not outright block UDP ports 80 and 443.

- (4) We further assume that the defender knows how to perform (encrypted) traffic analysis, which encompasses the analysis of packet lengths, time deltas between adjacent packets, payload entropy, and the analysis of all the cleartext information sent during the handshake.

### 4.2 Data Exfiltration Method

We base our method on the version-independent properties of the IETF QUIC protocol [33] and the connection migration feature described in RFC 9000 [19], in which a QUIC client has prior knowledge of the alternative server addresses to which it can choose to migrate its connections. During connection establishment, the server notifies the client about its alternative addresses by populating the `preferred_address` transport parameter with one or more alternative addresses. If a packet gets lost during the communication, the client can assume that the server has migrated to a new address and the client can choose to validate the new network path by sending a probing packet to the new server address [35]. Alternatively, the client may initiate a server-side connection migration at any time by sending a path validation packet to the preferred server address.

Fig. 3 presents the data exfiltration methodology, which is divided into three distinct phases — a sniffing phase, a path validation phase, and the main exfiltration phase. The involved entities are (i) the client (victim), whose machine is infected with the exfiltration tool, (ii) a middlebox, whose task is to detect and block suspicious network traffic, (iii) a benign server, and (iv) the malicious exfiltration server, which accepts all incoming connections and sends the respective response messages. During the initial sniffing phase, the data exfiltration software collects all outgoing QUIC short header packets and stores relevant information such as IP, UDP, and QUIC headers, as well as timestamps and the QUIC payload lengths. This

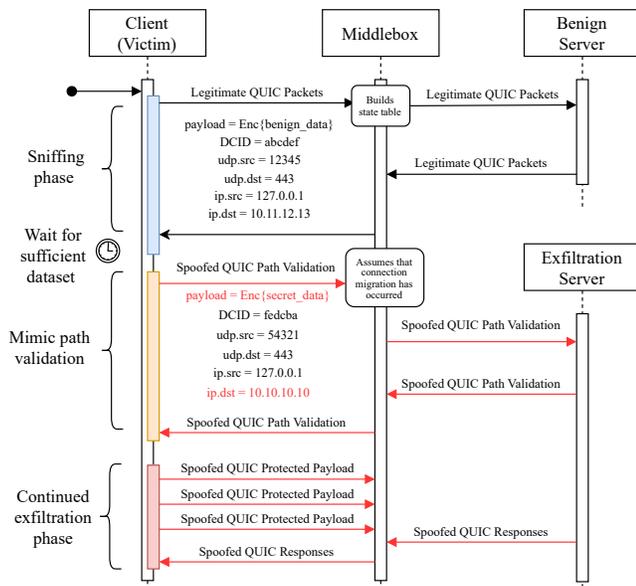


Figure 3: Proposed Data Exfiltration Attack.

information is reused at a later stage to mimic the structure of benign packets.

As part of the sniffing phase, the exfiltration tool waits for existing connections to retire their CID by continuously probing whether the socket that binds the currently used source port can be re-bound. Once the CID has been retired, the exfiltration tool can immediately emulate a server-side connection migration, without having to fear that benign traffic continues to be sent and causing suspicious overlaps of benign and malicious traffic originating from the same QUIC connection.

The first step of a connection migration is the path validation, which is mimicked by sending a spoofed path validation request to the malicious exfiltration server. The spoofed path validation packet may already contain the first bytes of sensitive data since a benign path validation packet is also sent via an encrypted channel and is, therefore, not visible to a middlebox. As per [19], a benign path validation request includes an 8-byte `PATH_CHALLENGE` frame, which in turn expects an 8-byte `PATH_RESPONSE` frame. An endpoint must use datagram sizes of at least 1200 bytes to transmit `PATH_CHALLENGE` and `PATH_RESPONSE` frames, as this ensures that the path is able to handle datagrams of this size [19]. Therefore, the spoofed path validation datagrams are also designed to have a size of at least 1200 bytes. When the middlebox first encounters the path validation packet, it must either assume that a QUIC connection migration is occurring or that an arbitrary UDP packet is sent to a destination address that has not been seen before. In both cases, middleboxes generally allow connections initiated from the inside (trusted) network to the outside.

Lastly, the main exfiltration phase begins by continuously sending packets to the spoofed exfiltration server address and receiving the appropriate spoofed responses from the exfiltration server, to mimic a “healthy” connection between a QUIC client and a QUIC server. Thereafter, the malicious connection may then be retired at any time and a new suitable connection can be selected to mimic another connection migration.

It is important to note that the method does not attempt to transmit any handshake packets. Therefore, middleboxes and/or fingerprinting software cannot detect nor block an illegitimate connection establishment attempt based on a handshake packet (cf. Section 6.3).

### 4.3 Increasing Stealthiness

In order to increase stealthiness, the data exfiltration tool can store a range of exfiltration server destination IP addresses; it may only use an address once to mimic a single connection migration and discards the address afterwards.

Furthermore, to minimize the payload size variance between benign and malicious traffic, the malicious payload lengths are sampled from a distribution of benign payload lengths, making a statistical analysis of packet lengths less feasible. Similarly, the time deltas between QUIC packets, which can be considered a unique fingerprint of the application, are also mimicked (cf. Section 6.2). The exfiltration payloads are encrypted with an AES-256 key stored in the data exfiltration executable. This key is not used to establish secrecy, but rather to increase the entropy of the data exfiltration

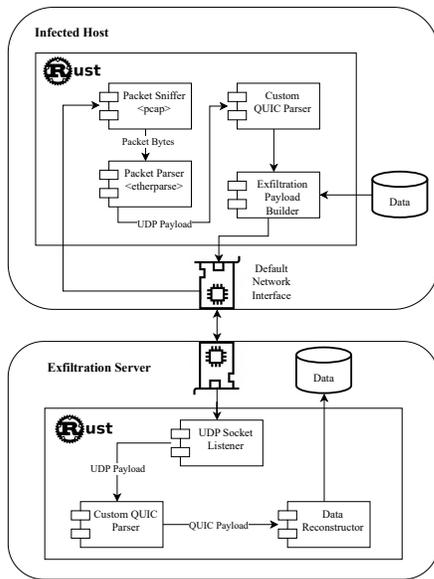


Figure 4: Proof-of-Concept Prototype Architecture.

payloads, in order to resemble the entropy of benign QUIC protected payload packets.

Data exfiltration is only attempted when a legitimate connection is retired, *i.e.*, when its DCID changes, or stops being used by the QUIC endpoint. Additionally, our method’s exfiltration throughput is dependent on the user’s activity, and only exfiltrates data while the user is actively browsing the web.

#### 4.4 Domain Registration Records

Detecting exfiltration traffic based on mismatches between domain registration records of old and new destination IP addresses is possible. However, as shown in [27], the WHOIS records may not be publicly accessible under certain jurisdictions. The EU General Data Protection Regulation (GDPR) requires certain WHOIS data to be redacted, which could make this attack more difficult to detect.

Furthermore, an attacker could host an exfiltration server on Amazon Web Services (AWS), resulting in domain registration records in which the organization’s name appears identical to those of legitimate Amazon.com traffic. Similarly, YouTube traffic IP addresses have WHOIS records pointing to Google LLC. Data exfiltration servers hosted on Google Cloud would have similar WHOIS records, making it challenging to identify them as malicious based only on WHOIS records.

Checking published IP address ranges specific to a cloud provider, which can be obtained directly from a cloud provider’s documentation, is one way to identify exfiltration attempts. For instance, when a QUIC connection to *google.com* migrates to a new destination server hosted in a Google Cloud, with identical domain registration records, it may be possible to cross-check against public IP address ranges of Google Cloud services to identify spoofed connection migration attempts. However, as of now, there is no indicator that leading firewall vendors offer such functionality in their products

to identify connection migrations and differentiate between benign and malicious ones (*cf.* Section 6.4).

## 5 Implementation

The PoC prototype was implemented in Rust as depicted in Fig. 4 and the client’s source code has been released<sup>1</sup>. It consists of a client-side data exfiltration executable, which comprises a packet sniffer, packet parser, custom QUIC parser, and exfiltration payload builder module. The server-side executable consists of a UDP socket listener, a custom QUIC parser and a data reconstruction module.

Every exfiltration task is performed in a new thread, meaning that multiple malicious connection migrations can be performed in parallel for different QUIC connections, while the outgoing benign traffic is continuously monitored for suitable connections that can be used to mimic new connection migrations. The outgoing QUIC packets are sniffed using the Rust *pcap* library [9] (version 1.2.0) and parsed using the Rust *etherparse* library [40] (version 0.14.2). It is assumed that QUIC uses port 443 over UDP. Every outgoing UDP packet’s payload is parsed as per the version-independent properties of the QUIC protocol [33]. The header structure of short header packets does not specify where the DCID ends nor does it specify the length of the DCID. Hence, observing a short header packet alone, does not give any indication to where the DCID bytes end and where the payload bytes start. If the parser identifies a QUIC long header packet as part of a handshake, it extracts its DCID, stores it, and subsequently discards the packet. The DCID is then used to map succeeding short header packets to their preceding handshake. This is required to (i) replicate the exact length of the original DCID in spoofed packets, as QUIC endpoints usually stick to a certain CID length, and (ii) mimic the lengths of the original payloads. Special cases, such as packets that contain a long and a short header at the same time or packets that contain a zero-length DCID, are discarded.

Monitoring all outgoing packets on the default network interface can result in a loop, as the exfiltration client itself also transmits packets over this interface. Sniffing and parsing those packets would result in an unnecessary performance overhead and would dilute the observed features of benign packets. To prevent this, the exfiltration client maintains a blacklist to ignore all malicious packets. This blacklist contains the SHA-3-256 hashes of all previously sent malicious packets, against which every outgoing packet is checked.

## 6 Evaluations

This section first introduces the experimental setup and the three different user activity scenarios that were considered. Subsequently, this section focuses on identifying unique features of QUIC connection migrations and training anomaly detection classifiers to try to detect the proposed attack. The remainder of this section discusses the feasibility of detecting the proposed attack using fingerprinting tools, followed by the results of a survey of leading firewall vendors, who were asked to assess the QUIC-filtering capabilities of their firewall products as part of unstructured interviews.

<sup>1</sup><https://github.com/thomasgruebl/quic-exfil>

**Table 1: Network Traffic Generation and Device Roles in the Experimental Testbed.**

Traffic Type	Device Number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
QUIC Traffic	●	●	●	●	●	●	●	●	○	○	○	○	○	○	○	○
Benign Migration	●	●	●	●	●	●	●	●	○	○	○	○	○	○	○	○
Exfiltration	○	○	○	○	○	○	●	●	●	●	○	○	○	○	○	○
Non-QUIC Traffic	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

**Table 2: Number of Benign and Malicious Connection Migrations per Scenario.**

Scenario	Benign	Malicious
Mixed	245	27
YouTube	371	34
Noise	98	11
Total	714	72

## 6.1 Experimental Setup

In our experimental setup, we simulate a small network of devices to generate QUIC traffic, including benign and malicious server-side connection migrations. The network consists of 16 Docker containers running Ubuntu 18.04 LTS with the Xfce desktop environment and VNC/noVNC servers to provide remote access [1]. Our machine acting as a firewall is a virtual machine (VM) with 32 GB of RAM, running Ubuntu 22.04 LTS. It uses iptables with the conntrack module for stateful packet filtering and connection tracking. Packet capturing is performed on the inward-facing interface of the firewall (*i.e.*, the Docker virtual Ethernet bridge adapter). As shown in Table 1, the network generates multiple types of traffic. Devices 1–8 generate QUIC traffic, including benign QUIC connection migrations to simulate real-world user-initiated network traffic. Devices 9–16 represent other backend (non user-controlled) machines, which do not generate QUIC traffic. Devices 7–10 are infected with the data exfiltration software and generate malicious connection migrations as well as QUIC-based data exfiltration traffic. All devices (1–16) also generate other types of non-QUIC network traffic. To emulate QUIC server-side connection migrations, we used a modified version of Cloudflare *quiche* v0.23.2 [7]. The quiche server is running on the firewall, the quiche clients are running on devices 1–8 and trigger QUIC connection migrations at random time intervals every 0–30 minutes. This allows us to collect benign connection migration fingerprints. The total number of generated connection migrations, benign and malicious, are presented in Table 2. Every single malicious connection migration in the dataset, that is followed by at least one exfiltration payload packet, constitutes a data exfiltration attempt.

As mentioned in Section 4.3, to increase the stealthiness of the method, the exfiltration throughput depends on the network activity of the infected hosts. Therefore, the following evaluation does not demonstrate that a certain throughput can be achieved. Instead,

it considers three different user scenarios in which the exfiltration tool attempts to send data to a target server and evaluates the performance of the anomaly detection classifiers. The following user activity scenarios were considered:

- (1) General Network Activity (24h of Mixed Traffic): A simulated small company network consisting of 16 Docker containers, each representing a host that collectively generates web traffic over a 24-hour period. This includes browsing across various domains (*e.g.*, youtube.com, google.com, facebook.com, instagram.com, cloudflare.com, amazon.com, chatgpt.com) and thus creating a number of different overlapping QUIC connections.
- (2) Isolated Streaming Scenario (24h of YouTube Traffic): The same simulated network, but now restricted to generating primarily YouTube video traffic over a 24-hour period. This scenario serves as a controlled test case to contrast with the more diverse traffic patterns of the general network activity.
- (3) Background Noise Traffic (24h Idle Mode): A subset of devices generating low-interaction background traffic for 24 hours, simulating idle devices with existing open QUIC connections (*e.g.*, open browser tabs, apps, etc.) but minimal active user interaction.

Across all three scenarios, we gathered a total of 710,690 outgoing QUIC short header packets, of which 690,101 are benign and 20,589 are spoofed. Scenario 1 (*i.e.*, mixed traffic) generated 427,644 outgoing QUIC short header packets, 416,961 of which are benign and 10,683 of which are spoofed. The total achieved exfiltration volume is 6.34 MB. Scenario 2 (*i.e.*, YouTube traffic) generated 255,649 outgoing QUIC short header packets, 247,624 of which are benign and 8,025 of which are spoofed, resulting in a total data exfiltration volume of 2.97 MB. Scenario 3 (*i.e.*, noise traffic) generated a total of 27,397 packets, including 25,516 benign and 1,881 malicious ones and achieved a total volume of 1.10 MB.

It is evident that the data exfiltration throughput is highly dependent on the victim’s activity. In cases where the user browses websites that do not require large amounts of data to be downloaded, only infrequent QUIC acknowledgment packets are being sent back to the server. YouTube traffic generates many outgoing QUIC acknowledgments with small payload sizes. Data uploads to a cloud provider typically generate a high number of outgoing QUIC packets using the maximum payload size. Mimicking a connection migration in the latter case enables the attacker to exfiltrate large amounts of sensitive data.

**Table 3: Anomaly Detection Features.**

Category	Feature	Considered	Justification
Handshake	TLS Client Hello, Server Hello, etc.	✗	Both legitimate and spoofed connection migrations cannot be reliably mapped to a preceding handshake.
Short Header Packets	Packet Length	✗	The length of the entire packet is dependent on features that may also differ across benign packets (e.g., IP header structure).
	SCID/DCID	✗	Both the length of the CIDs and the CIDs themselves have no informational content and can vary across benign and malicious traffic.
	QUIC Payload Length	✓	The payload length may be used as an indicator of irregular traffic patterns. Unusual payload sizes can indicate data exfiltration attempts.
	Connection Migration Payload Length	✓	The payload length of a connection migration packet (i.e., a packet containing a <i>PATH_CHALLENGE</i> frame) can be used as an indicator of unusual connection migration attempts.
	Time $\Delta$ between two outgoing packets	✓	The data exfiltration tool may have different processing times compared to a benign QUIC endpoint.
	Time $\Delta$ between two ingoing packets	✗	This feature is similar to the previous one, since the exfiltration server’s QUIC endpoint would be functioning identically to the QUIC endpoint running on the client, therefore only one of the two features need to be considered.
	Time $\Delta$ between a request-response pair	✗	The RTT latency can be arbitrary for both legitimate and spoofed addresses, depending on the physical location of the destination server.
	Payload Entropy	✗	The entropy of payloads can be imitated by encrypting the exfiltration payload using a cryptographic key stored in the exfiltration software.
	Latency Spin Bit	✗	The latency spin bit is used by on-path observers to measure the per-round-trip latency. For every received packet from the server, the client simply flips the bit, which does not provide enough informational content to differentiate benign from malicious traffic.
	Fixed Bit	✗	As the name suggests, the fixed bit has a constant value and serves as a way to differentiate QUIC traffic from other UDP-based traffic. It can simply be adopted by malicious packets.
Packet number	✗	In short header packets, the packet number, as well as its length, are cryptographically obfuscated and thus not visible to middleboxes.	
Other QUIC-version dependent bits	✗	A spoofed packet can simply adopt these bits from a benign packet.	

## 6.2 Anomaly Detection

In this evaluation section, we analyze the proposed data exfiltration method by assessing the ability of machine learning-based anomaly detectors to detect this kind of malicious network behaviour. Since existing anomaly detection classifiers that detect network-based exfiltration attempts are commonly trained on handshake metadata [49], and QUIC connection migrations do not require handshakes, a new feature set needs to be defined for QUIC-based data exfiltration traffic.

Table 3 presents a range of different features that can be considered in the context of QUIC traffic analysis. Most features can be easily imitated – only three features have been identified which require some level of sophistication to imitate. Therefore, our objective was to mimic the following features with the help of our PoC implementation: ① The connection migration payload, ② the “normal” QUIC packet payload length, and ③ the time delta between two outgoing packets of the same QUIC flow.

Feature ① is a binary feature that captures connection migration attempts in outgoing QUIC packets. More specifically, it observes the first packet of a connection migration and stores its payload length. We classify every observed migration attempt accordingly, using this binary label as a potential discriminator between “normal” protected payload packets and packets containing a *PATH\_CHALLENGE* frame.

Feature ②, payload length, can be imitated by first observing benign traffic during the sniffing phase depicted in Fig. 3. After a sufficiently sized dataset of benign QUIC payload lengths has been collected, the data exfiltration tool randomly samples from this dataset – essentially replicating the payload sizes of previously seen traffic of the same QUIC flows. For instance, when mimicking

a connection migration of a YouTube server, the exfiltration tool continues to use the same payload sizes as those observed in benign acknowledgment packets sent from the client to the YouTube server. We heuristically chose to gather a dataset of 1,000 packets per QUIC connection prior to the first exfiltration attempt.

To replicate the payload sizes of benign QUIC flows, we define  $\mathcal{D}$  to be the entire dataset that is created over the course of an exfiltration process. Since the dataset changes over time, we denote the subset of the dataset at time  $t$  by  $\mathcal{D}_t$  which is used to choose the next QUIC payload size. We then randomly sample a value  $x$  from the dataset  $\mathcal{D}$  at time  $t$ ,

$$x_t \sim \mathcal{D}_t$$

and use this value  $x_t$  to set the length of the next malicious payload.

Feature ③, the time delta between two outgoing packets, is computed as per Algorithm 1. The intervals in which the packets are placed “on the wire” can be considered a unique fingerprint of an application. From the attacker’s perspective, this is the most difficult feature to mimic among the three, because it requires dynamic adjustments of inter-arrival times on a per-flow basis. To mimic feature ③, Algorithm 2 was implemented.

Algorithm 1 monitors the time deltas of benign QUIC connections and stores them in a dictionary. During data exfiltration, Algorithm 2 randomly samples from this distribution to determine the appropriate sleep time before sending the next packet. The base rate of exfiltration was determined by running a single thread of the tool, measuring the time deltas, and averaging them. Since the standard deviation is low, the average can be reliably used. The base rate is typically very close to zero (median: 7 ms, average: 58 ms), meaning the required sleep times closely match the observed

---

**Algorithm 1** Compute Time Deltas per DCID
 

---

**Require:** Pre-filled hashmap  $H$  ▶ Key: DCID, Value: Sorted list of timestamps

- 1: Initialize an empty hashmap  $\Delta T$  ▶ Stores time deltas for all DCIDs
- 2: **for** each  $(DCID, T_{list})$  in  $H$  **do**
- 3:   **if**  $|T_{list}| > 1$  **then**
- 4:     **for**  $i \leftarrow 1$  to  $|T_{list}| - 1$  **do**
- 5:       Compute time delta:  
                    $\Delta T_i = T_{list}[i + 1] - T_{list}[i]$
- 6:       Append  $\Delta T_i$  to  $\Delta T[DCID]$
- 7:     **end for**
- 8:   **end if**
- 9: **end for**
- 10: **return**  $\Delta T$

---



---

**Algorithm 2** Mimic Observed Time Deltas
 

---

**Require:**  $\Delta T = \{DCID_1 : [dt_1, dt_2, \dots], DCID_2 : [dt_1, dt_2, \dots]\}$  ▶ Observed time deltas

**Require:**  $BR$  ▶ Base rate of packet sending

**Require:**  $n$  ▶ Number of packets to exfiltrate

**Ensure:** Adjusted sleep times for mimicking the observed rate

- 1: **for**  $i \leftarrow 1$  to  $n$  **do**
- 2:    $\Delta T_i =$  Get random sample from  $\Delta T[DCID]$
- 3:   Compute sleep time:  
                    $S_i = \Delta T_i - BR$
- 4:   **if**  $S_i > 0$  **then**
- 5:     Sleep for  $S_i$  milliseconds
- 6:   **else**
- 7:     No sleep needed
- 8:   **end if**
- 9:   Send packet
- 10: **end for**

---

time deltas. This low base rate can be traced back to the fact that the exfiltration tool bypasses common rate-limiting mechanisms such as flow control and congestion control, making it faster than most other applications.

We do not consider time intervals between adjacent pairs of requests and responses, because these are dependent on the location of the destination server – which can also significantly differ across benign servers performing connection migrations. Nor do we consider the time deltas between two ingoing packets, since this feature’s expressiveness is identical to ③.

Given the inherent challenges of anomaly detection in realistic highly imbalanced network traffic datasets where data exfiltration attempts constitute a small fraction (~3%) of the overall traffic, we evaluated the performance of five distinct anomaly detection classifiers. These classifiers use both supervised methods – Random Forest (RF), Multi-Layer Perceptron (MLP), and Support Vector Machine (SVM) – and unsupervised methods – Autoencoder (AE) and Isolation Forest (IF). Supervised methods, such as RF, MLP, and SVM, typically perform well in scenarios where clear decision

**Table 4: Comparison of the Classification Performance Across Three Scenarios**

Classifier	Metric	Scenario		
		Mixed	Youtube	Noise
RF	F1-Score	0.35	0.18	0.47
	Recall	0.31	0.15	0.45
	Precision	0.40	0.22	0.50
	Accuracy	0.97	0.96	0.94
MLP	F1-Score	0.07	0.10	0.20
	Recall	0.44	0.34	0.45
	Precision	0.04	0.06	0.13
	Accuracy	0.72	0.81	0.78
SVM	F1-Score	0.02	0.05	0.08
	Recall	0.03	0.07	0.08
	Precision	0.02	0.04	0.08
	Accuracy	0.93	0.92	0.89
AE	F1-Score	0.00	0.01	0.00
	Recall	0.01	0.01	0.00
	Precision	0.00	0.01	0.01
	Accuracy	0.92	0.92	0.93
IF	F1-Score	0.06	0.08	0.05
	Recall	0.23	0.21	0.08
	Precision	0.03	0.05	0.04
	Accuracy	0.82	0.84	0.83

RF → Random Forest, MLP → Multi-Layer Perceptron, SVM → Support Vector Machine, AE → Autoencoder, IF → Isolation Forest

boundaries can be learned from labeled examples [49]. Unsupervised methods, like AE and IF, are designed to identify anomalies without explicit labels, by learning the characteristics of normal data and flagging deviations. We selected these classifiers to assess the robustness of feature-based anomaly detection against the data exfiltration technique designed to mimic benign traffic patterns.

We evaluated the detection performance across the three user activity scenarios. Table 4 presents a comparative analysis of the classifiers, which focuses on the following metrics: Precision, Recall, and F1-Score for the anomaly class, and overall Accuracy. Due to the class imbalance, accuracy alone is not a reliable indicator of performance, as high accuracy can be trivially achieved by classifiers that predominantly predict the majority class (*i.e.*, benign traffic).

Across all scenarios, the results presented in Table 4 reveal that all five classifiers are unable to effectively detect the data exfiltration attempts. Despite achieving high accuracy scores in some scenarios, these scores are misleading due to the imbalanced datasets. The more important metrics for the anomaly class – F1-Score, Recall, and Precision – consistently show very low values across all classifiers and scenarios.

In the YouTube scenario, for instance, the RF classifier achieves a relatively high accuracy of 96%. However, its F1-Score for detecting data exfiltration is 0.18, with a Recall of only 0.15 and a Precision of 0.22. This indicates that while RF correctly classifies the majority of benign traffic, it fails to identify most actual data exfiltration attempts and produces a high number of false negatives. Similar poor performance is observed for the MLP and SVM classifiers in the YouTube scenario, with even lower F1-Scores of 0.10 and

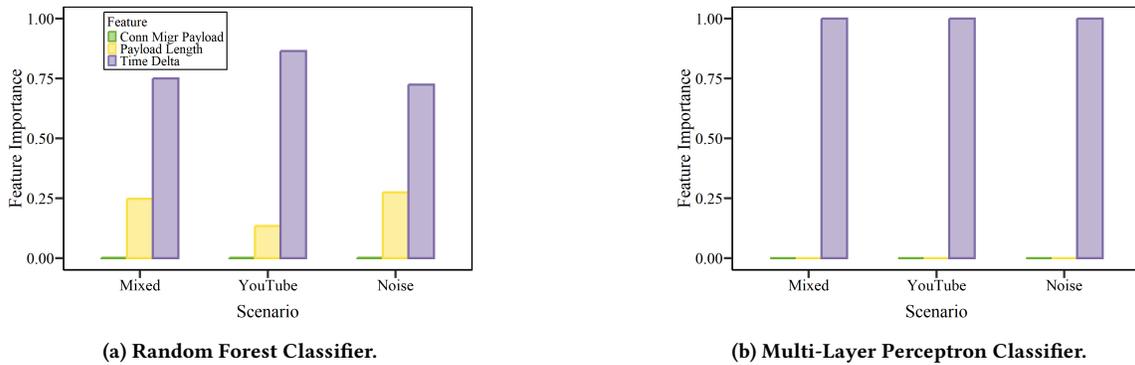


Figure 5: Normalized Feature Importance Across Three Scenarios.

0.05 respectively. The AE yields a near-zero F1-Score (0.01) and IF achieves a slightly higher but still very low F1-Score of 0.08.

In the Noise scenario, RF achieves a high accuracy of 96%, while its F1-Score for anomaly detection remains low at 0.47. MLP performs even worse with an F1-Score of 0.20 in this scenario. SVM and AE again show very poor anomaly detection performance, with near-zero F1-Scores. IF exhibits a slightly improved F1-Score of 0.05 compared to AE and SVM.

In the Mixed traffic scenario, the anomaly detection performance of all classifiers is low. RF achieves the highest F1-Score among the classifiers at 0.35, but still fails to differentiate benign from exfiltration traffic. MLP, SVM, AE, and IF all exhibit very low F1-Scores, ranging from 0.00 to 0.07.

We have also conducted a feature importance analysis for both the RF and MLP detectors. We used the *Gini* importance metric for RF, a measure of how much each feature contributes to the homogeneity of nodes in decision trees. For the MLP detector, we employed the Sharpley Additive Explanation (SHAP) method [28], a game-theoretical approach that attributes the model output to each feature based on their marginal contributions. Fig. 5a reveals that feature ③ is the most important across all three scenarios. Our analysis also indicates that for the MLP, the time delta feature contributes 100% to its classification decisions (*cf.* Fig. 5b). However due to the successful mimicking strategy of time delta feature ③, it is still difficult to detect malicious attempts.

### 6.3 Fingerprinting Software

Network metadata fingerprinting tools collect information from packet headers (*e.g.*, IP addresses, port numbers, TLS-specific metadata) to create a “fingerprint” that uniquely identifies a network packet flow. This fingerprint can be used not only for various purposes, such as network performance optimization and device management, but also to make inferences about the underlying application or website that is visited. The features used for fingerprinting can be extracted by a passive eavesdropper, such as an on-path observer [13]. Fingerprinting tools, such as the open-source tools Cisco Mercury [30] or FATT [20], extract a fingerprint from a QUIC handshake and try to map non-handshake packets to their preceding handshakes. The mapping between the handshakes and the corresponding payload packets is done using the 5-tuple. Packets

with modified 5-tuples and no preceding handshakes (*i.e.*, QUIC connection migration packets) are therefore not mapped to a specific fingerprint.

Wireshark [46] offers the ability to follow QUIC streams. When encountering unknown QUIC packets that cannot be associated with a handshake, the packets are labeled as “*Unknown QUIC connection. Missing Initial Packet or migrated connection?*”, if the protected payload packet contains a CID that was not previously seen as part of the handshake packets. In our current PoC implementations, we reuse existing DCIDs. Even when new DCIDs are used, Wireshark’s labeling process cannot distinguish between benign and malicious connection migrations.

As part of the evaluation, a post-analysis of the captured PCAP files has been performed using Cisco Mercury [30]. The preceding handshake for both benign and malicious connection migrations was detected by Cisco Mercury, however, the tool fails to map the traffic after a migration (including the migration event itself) to the original handshake. This means, that the tool is unable to correlate the post-migration traffic with the initial connection, hence, treating it as an entirely new flow rather than a continuation of the original session. As a result, since the new UDP flow lacks handshake data, the malicious destination IP address of the exfiltration server cannot be found in the Cisco Mercury fingerprinting results.

### 6.4 Survey of Leading Firewall Vendors

As part of this study, unstructured interviews with leading firewall vendors were conducted. The qualitative approach of surveying vendors was intended to replace a quantitative evaluation because it directly addresses the core question: whether modern firewalls even possess the necessary features to reliably detect QUIC connection migrations. We contacted eight firewall vendors listed in [10], five of which agreed to share their perspectives on the QUIC protocol for research purposes. The interviewees’ roles included “Systems Engineer”, “Cyber Security Specialist”, “Senior Sales Engineer”, “Consulting Systems Engineer”, “Senior Solutions Engineer” and “Systems Engineering Manager”. The main questions asked, presented in Table 5, revolved around the capabilities offered by their firewall products with regard to handling QUIC traffic. Specifically, the survey sought to determine whether the vendors (*i*) recommend blocking QUIC entirely, (*ii*) include information about the

**Table 5: Perspectives on the QUIC Protocol from Leading Firewall Vendors**

Firewall vendor ...	A	B	C	D	E
(i) recommends blocking QUIC entirely.	✓	✓	✓	✓	✗
(ii) builds a state table based on QUIC connection IDs.	✗	✗	✗	✗	✗
(iii) can differentiate QUIC traffic from other types of traffic.	✓	✓	✓	✓	✓
(iv) can perform basic filtering of QUIC traffic (allow / deny).	✓	✓	✓	✓	✓
(v) currently offers functionality to decrypt QUIC traffic on the firewall.	✗	✗	✓	✓	✓
(vi) has HTTP/3 Deep Packet Inspection (DPI) capabilities.	✗	✗	✓	✓	✓
(vii) can recognize QUIC connection migrations.	✗	✗	✗	✗	✗
(viii) plans on / is currently developing new QUIC-related firewall features.	✓	✓	✓	~	✓
(ix) believes that the competitors struggle with identical challenges.	✓	✓	~	✓	✓

~ → Is unsure / cannot provide clear answer.

QUIC protocol into their firewall state tables, (iii) can differentiate QUIC traffic from other types of traffic, and (iv) can perform basic filtering of QUIC traffic. Additionally, it was assessed whether the firewalls (v) currently offer functionality to decrypt QUIC traffic, which is similar to HTTPS decryption on a technical level, and (vi) can perform HTTP/3 DPI. Full decryption of QUIC traffic is a pre-requisite for performing HTTP/3 DPI, which three out of five vendors are currently offering as part of their product suite. DPI refers to analyzing the contents of the packet after decrypting it, although it is also sometimes referred to as header analysis of the unencrypted QUIC handshake portions, such as the TLS Server Name Indication (SNI) field. Full decryption means that the firewall performs a man-in-the-middle inspection on QUIC traffic, making all embedded contents (e.g., HTTP/3) readable to the firewall and enabling fine-grained content filtering. No vendor mentioned that their firewall solutions can reliably identify QUIC connection migrations – neither client-side nor server-side (vii).

Almost all surveyed vendors are (viii) planning on developing or currently actively developing new QUIC-related features. However, as of now, they still recommend that their clients block the QUIC protocol entirely. Interestingly, all participants shared that QUIC traffic analysis features are rarely requested by clients, which suggests that there may be a lack of awareness or understanding about the importance and benefits of QUIC traffic analysis. Additionally, the firewall vendors were asked for their perspective on how their competitors are dealing with the increase in QUIC traffic. The general consensus among four out of five participants was that QUIC traffic analysis seems to be, as expected, an industry-wide challenge (ix).

## 7 Discussion

As of today, modern firewalls are not tracking QUIC connections in state tables, meaning that state tables have to treat every outgoing UDP packet as a new connection attempt [11]. Firewalls generally operate under the assumption that outgoing traffic is safe, as it originates from within the trusted network. When an outgoing UDP packet arrives at the firewall, the firewall checks whether the state table contains an existing entry corresponding to the packet’s 5-tuple. If a matching entry is found, the packet is processed according to the pre-established rules for that connection. If no matching entry exists, the firewall may create a new entry in the state table or take other actions based on its configuration.

Although there are ways to perform stateful treatment of QUIC traffic based on its few observable features, such stateful treatment requires trade-offs with the confidentiality and censorship-resistance of the protocol. RFC 9312 [22] discusses the manageability of the protocol and analyzes ways to perform stateful treatment of QUIC traffic. Apart from observing the cleartext parts of the handshake and implementing custom QUIC extensions that un-conceal more information, there are limited options available to reliably track connections. Using the CID as a stateful identifier is not possible, since the CIDs can be renegotiated at any time within the encrypted channel. Stateful firewalls cannot even rely on the detection of end-of-flow signals to terminate a connection, as end-of-flow signals are not visible to an on-path observer [22]. Therefore, a QUIC-aware firewall would have to rely on timer-based state removals.

In the context of the proposed data exfiltration method, this means that, as of today, most enterprise-level network-based firewalls cannot detect a potential QUIC connection migration, nor can they differentiate between migrations and request forgery attacks.

### 7.1 Mitigation Strategies

The risk of the proposed attack can be partly mitigated through the following countermeasures:

- A client may disable QUIC transport parameters, like active connection migration using the `disable_active_migration` (0x0c) flag, or remove the server preferred\_address field from the QUIC implementation. Any connection migration attempt can then be flagged as anomalous activity.
- A modification to QUIC’s implementation so that the preferred\_address transport parameter field is part of the unencrypted QUIC handshake. As a result, custom middle-box software can be developed that can recognize connection migration attempts by monitoring QUIC packets for changes in their source or destination IP addresses and/or ports. These changes can then be checked against the contents of the preferred\_address parameter and domain registration records of the new IP addresses to verify their legitimacy.
- Another approach is full decryption of QUIC traffic on a firewall level. Although not recommended for privacy reasons, as it defeats the purpose of the QUIC protocol, it allows for in-depth HTTP/3 inspection. Even in such a case,

firewall vendors would still need to implement the aforementioned custom detection mechanisms to verify that the `preferred_address` address field contains a legitimate address.

## 8 Related Work

### 8.1 QUIC Request Forgery Attacks

QUIC traffic is inherently difficult to differentiate from “normal” UDP traffic. The encryption of QUIC headers and payloads obfuscates the traffic, which prevents easy inspection and classification. Even advanced network sniffing tools such as Wireshark cannot reliably detect QUIC if the handshake phase has not been observed. In addition to obfuscation challenges, there are various attacks targeting the QUIC protocol, for instance, request forgery attacks.

RFC 9000 [19] describes different types of request forgery attacks, including “Request Forgery with Client Initial Packets”, “Request Forgery with Preferred Addresses”, “Request Forgery with Spoofed Migration”, and “Request Forgery with Version Negotiation”. Although [19] mentions a request forgery attack on the DCID field in packets sent to a preferred address, it fails to point out that a packet’s destination IP may be spoofed to redirect traffic to a malicious IP address. The RFC suggests no specific countermeasures beyond generic security recommendations.

Gbur and Tschorsch [12] performed an analysis of the feasibility of client-side request forgery attacks. They focused on client-side Server Initial Request Forgery (SIRF), Version Negotiation Request Forgery (VNRF), and Connection Migration Request Forgery (CMRF). Their CMRF analysis only encompassed forging client-side connection migration events, which aim to trick a legitimate QUIC server into sending a QUIC packet to a spoofed address. It did not cover the forging of server-side connection migration events.

One example of spoofing connection migration events with benevolent intent is MIMIQ [15], a privacy-enhancing system that aims to prevent traffic analysis by a middlebox. The system allows clients to maintain anonymity by frequently rotating their source IP address without disrupting connections. It prevents adversaries from identifying the client or associating multiple flows with it. Connections are broken into smaller flows and migration times are strategically chosen, making it harder for adversaries to analyze traffic and gather information about a particular client.

### 8.2 Data Exfiltration

The field of data exfiltration is vast, encompassing various attack vectors such as exfiltration over web services, physical media, network media, and alternative [network] protocols [31]. We therefore limited the review to papers that developed attacks related to the MITRE ATT&CK framework technique “Exfiltration over Alternative Protocol” [31], which includes all network protocols not being used as the main C2 channel.

There have been efforts in QUIC-based data exfiltration, such as the prototype presented in [48], which embeds data within a legitimate QUIC connection. However, this approach does not attempt to hide the exfiltration, leaves a noticeable fingerprint due to the handshake, and does not adjust packet features like payload length.

Sudhan and Kulkarni [43] introduced a method to establish a covert channel between two QUIC endpoints using the latency spin

bit. The spin bit is an optional QUIC protocol feature that allows for passive on-path network latency monitoring. The proposed method requires two QUIC endpoints to establish a legitimate connection, making it less feasible for malicious data exfiltration attempts. Furthermore, only one bit of data can be exchanged per QUIC packet, which makes the method impractical for exfiltrating large amounts of data.

Zhan et al. [49] proposed a method to detect DNS-over-HTTPS (DoH) based data exfiltration by analyzing TLS-fingerprints and training Boosted Decision Trees, Random Forest, and Logistic Regression classifiers on flow-based features, achieving detection accuracies of over 99%.

Vaccari et al. [44] exploit the Message Queue Telemetry Transport (MQTT) protocol, commonly used within IoT networks, to exfiltrate sensitive data from a private network. Their method successfully exfiltrated payloads up to 3000 bytes over the MQTT protocol, and simultaneously, they were able to achieve detection accuracies of up to 99% using Random Forest classifiers.

Klein [21] introduced a data exfiltration method that exploits stateful IPv4 IDs, TCP ISNs and IPv6 flow labels on popular server operating systems. The study demonstrated the feasibility of exfiltrating data from *firewalled* networks using the global protocol states to establish covert channels. In addition, it explored cross-protocol attacks and measured exfiltration bandwidth, which was sufficiently high to extract secret key material within a few hours.

## 9 Conclusion

This paper analyzes the feasibility of covert data exfiltration attacks using the QUIC transport protocol. We find that adversaries can make use of QUIC’s server-side connection migration feature to exfiltrate data from a trusted network to a target server. We show that, because of the inherent traits of the QUIC protocol, QUIC-based data exfiltration techniques are difficult to differentiate from normal protocol behaviour, and not even custom anomaly detection classifiers are able to detect such data exfiltration attempts. Some mitigation strategies include outright disabling server-side connection migration, sending the `preferred_address` parameter as part of the unencrypted handshake, or implementing custom firewall software that checks the `preferred_address` parameter against domain registration entries. From the lack of QUIC traffic analysis capabilities offered by leading firewall vendors, one can infer that, as of today, firewalls cannot effectively handle the complexities of QUIC. Our contribution lies not in demonstrating the success of ML-based detection, but rather in revealing the limitations of current ML methods against advanced mimicking attacks. Future work may include developing heuristics-based QUIC traffic inspectors that can be deployed on middleboxes.

## Acknowledgments

This work was partially supported by (a) the University of Zürich UZH, Switzerland, and (b) the Horizon Europe Framework Program’s project Certify, Grant Agreement No. 101069471, funded by the Swiss State Secretariat for Education, Research, and Innovation SERI, under Contract No. 22.00165.

## References

- [1] accetto. 2024. *Headless Ubuntu/Xfce container with VNC/noVNC and Firefox (G2)*. <https://hub.docker.com/r/acchetto/xubuntu-vnc-novnc-firefox>
- [2] Alibaba. 2024. *xquic*. <https://github.com/alibaba/xquic>
- [3] David Belson and Lucas Pardue. 2023. *Examining HTTP/3 usage one year on*. <https://blog.cloudflare.com/http3-usage-one-year-on>
- [4] M. Bishop. 2022. *HTTP/3*. RFC 9114. IETF. <https://www.ietf.org/rfc/rfc9114.txt>
- [5] Aurélien Buchet and Cristel Pelsser. 2024. An Analysis of QUIC Connection Migration in the Wild. *arXiv preprint arXiv:2410.06066* (2024).
- [6] Cisco. 2023. *Encrypted Visibility Engine: An overview of Cisco Secure Firewall's Encrypted Visibility Engine (EVE)*. <https://secure.cisco.com/secure-firewall/v7.3/docs/encrypted-visibility-engine-73>
- [7] Cloudflare. 2024. *quiche*. <https://github.com/cloudflare/quiche/blob/master/quiche/src/lib.rs>
- [8] devsisters. 2024. *libquic*. <https://github.com/devsisters/libquic>
- [9] ebfll and Kozłowski, Wojciech. 2024. *pcap: A packet capture API around pcap-w-pcap*. <https://crates.io/crates/pcap>
- [10] Gartner. 2024. *Network Firewalls Reviews and Ratings*. <https://www.gartner.com/reviews/market/network-firewalls>
- [11] Konrad Yuri Gbur and Florian Tschorsch. 2021. A quic (k) way through your firewall? *arXiv preprint arXiv:2107.05939* (2021).
- [12] Konrad Yuri Gbur and Florian Tschorsch. 2023. QUICforge: Client-side Request Forgery in QUIC. In *Network and Distributed System Security (NDSS) Symposium*.
- [13] I. Goldberg, T. Wang, and C.A. Wood. 2020. *Network-Based Website Fingerprinting*. Technical Report. pearg. <https://www.ietf.org/archive/id/draft-irtf-pearg-website-fingerprinting-00.html>
- [14] Google. 2024. *Google quiche*. <https://github.com/google/quiche>
- [15] Yashodhar Govil, Liang Wang, and Jennifer Rexford. 2020. {MIMIQ}: Masking {IPs} with Migration in {QUIC}. In *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*.
- [16] HAProxy. 2024. *haproxy*. <https://www.haproxy.org/>
- [17] Christian Huitema, Sara Dickinson, and Allison Mankin. 2024. . RFC 9250. IETF. <https://www.rfc-editor.org/rfc/rfc9250.txt>
- [18] IBM. 2024. *Cost of a Data Breach Report 2024*. <https://www.ibm.com/reports/data-breach>
- [19] Jana Iyengar and Martin Thomson. 2021. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. IETF. <https://www.ietf.org/rfc/rfc9000.txt>
- [20] Karimi, Adel. 2019. *FATT: fingerprint all the things!* <https://github.com/0x4D31/fatt>
- [21] Amit Klein. 2022. Subverting Stateful Firewalls with Protocol States. In *Network and Distributed System Security (NDSS) Symposium*.
- [22] M. Kühlewind and B. Trammell. 2022. *Manageability of the QUIC Transport Protocol*. RFC 9312. IETF. <https://www.ietf.org/rfc/rfc9312.txt>
- [23] Lainé, Jeremy . 2024. *aiquic*. <https://github.com/aiortc/aiquic>
- [24] Adam Langley, Alistair Ridcho, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the conference of the ACM special interest group on data communication*. 183–196.
- [25] Yanmei Liu, Yunfei Ma, Quentin De Coninck, Olivier Bonaventure, Christian Huitema, and Mirja Kühlewind. 2024. *Multipath Extension for QUIC*. Work in Progress. IETF. <https://datatracker.ietf.org/doc/draft-ietf-quic-multipath/>
- [26] Tan Lizhuang, Gao Xiaochuan, Su Wei, Li Na, and Zhang Wei. 2020. *Connection Migration in QUIC*. Work in Progress. IETF. <https://datatracker.ietf.org/doc/html/draft-tan-quic-connection-migration-00>
- [27] Chaoyi Lu, Baojun Liu, Yiming Zhang, Zhou Li, Fenglu Zhang, Haixin Duan, Ying Liu, Joann Qionga Chen, Jinjin Liang, Zaifeng Zhang, et al. 2021. From WHOIS to WHOAS: A Large-Scale Measurement Study of Domain Registration Privacy under the GDPR. In *Network and Distributed System Security (NDSS) Symposium*.
- [28] Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 4765–4774. <http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf>
- [29] Beatrice Martini and Niels ten Oever. 2019. *QUIC Human Rights Review*. Technical Report. HRP Human Rights Review Team. <https://datatracker.ietf.org/meeting/102/materials/slides-102-hrpe-slides-hrreview-quic-00>
- [30] McGrew, David and Enright, Brandon and Anderson, Blake and Messenger, Lucas and Weller, Adam and Chi, Andrew and Acharya, Shekhar and Antonyk, Anastasiia-Mariia and Stepanov, Oleksandr and Viswanathan, Vigneshwari and Raj, Apoorv. 2020. *Cisco Mercury: network metadata capture and analysis*. <https://github.com/cisco/mercury>
- [31] MITRE. 2024. MITRE ATT&CK Framework. Exfiltration Over Alternative Protocol. <https://attack.mitre.org/techniques/T1048/>
- [32] Maxime Piraux and Olivier Bonaventure. 2024. *Additional addresses for QUIC*. Work in Progress. IETF. <https://datatracker.ietf.org/doc/draft-piraux-quic-additional-addresses/>
- [33] T. Pornin. 2021. *Version-Independent Properties of QUIC*. RFC 8999. IETF. <https://www.ietf.org/rfc/rfc8999.txt>
- [34] private-octopus. 2024. *picoquic*. <https://github.com/private-octopus/picoquic>
- [35] Carlo Puliafito, Luca Conforti, Antonio Virdis, and Enzo Mingozzi. 2022. Server-side QUIC connection migration to support microservice deployment at the edge. *Pervasive and mobile computing* 83 (2022), 101580.
- [36] Puliafito, Carlo and Conforti, Luca and Virdis, Antonio and Mingozzi, Enzo. 2022. *Server-side QUIC connection migration to support microservice deployment at the edge*. [https://github.com/kruviser/aiquic-explicit\\_UniPisa](https://github.com/kruviser/aiquic-explicit_UniPisa)
- [37] quic-go. 2024. *quic-ggo*. <https://github.com/quic-go/quic-go>
- [38] quinn-rs. 2024. *quinn*. <https://github.com/quinn-rs/quinn>
- [39] Jim Roskind. 2012. *Quick UDP internet connections: Multiplexed stream transport over UDP*. <https://www.ietf.org/proceedings/88/slides/slides-88-tsvarea-10.pdf>
- [40] Schmid, Julian. 2024. *etherparse*. <https://crates.io/crates/etherparse>
- [41] Cherie Shi. 2019. *QUIC Connection Migration*. Technical Report. IETF. <https://datatracker.ietf.org/doc/slides-104-maprg-quic-connection-migration-cherie-shi/>
- [42] Randall R. Stewart, Michael Tüxen, and karen Nielsen. 2022. Stream Control Transmission Protocol. RFC 9260. <https://doi.org/10.17487/RFC9260>
- [43] H. H. Sudhan S. and Sameer G. Kulkarni. 2024. Security and Service Vulnerabilities with HTTP/3. In *2024 16th International Conference on COMMunication Systems & NETWORKS (COMSNETS)*. IEEE, 55–60.
- [44] Ivan Vaccari, Sara Narteni, Maurizio Aiello, Maurizio Mongelli, and Enrico Cambiaso. 2021. Exploiting Internet of Things protocols for malicious data exfiltration activities. *IEEE Access* 9 (2021), 104261–104280.
- [45] Mona Wang, Anunay Kulshrestha, Liang Wang, and Prateek Mittal. 2022. Leveraging strategic connection migration-powered traffic splitting for privacy. *arXiv preprint arXiv:2205.03326* (2022).
- [46] Wireshark Foundation. 2024. *Wireshark*. <https://www.wireshark.org>
- [47] Yamamoto, Kazu. 2024. *IETF QUIC implementation in Haskell*. <https://hackage.haskell.org/package/quic>
- [48] ytsif. 2024. *PyExfil: Stress Testing Detection & Creativity*. <https://github.com/ytsif/PyExfil/blob/master/USAGE.md>
- [49] Mengqi Zhan, Yang Li, Guangxi Yu, Bo Li, and Weiping Wang. 2022. Detecting DNS over HTTPS based data exfiltration. *Computer Networks* 209 (2022), 108919.

All links above were last accessed on June 3, 2025.

## A Evaluation of Open-Source Implementations

Since the proposed attack relies on client-initiated server-side migrations, it is vital to understand the actual adoption of this feature. Thus, this section reviews several open-source client- and server-side implementations of the QUIC protocol. Each solution (*cf.* Table 6) is statically analyzed to infer whether it supports server-side connection migrations and/or Multipath QUIC [25]. Furthermore, since libraries differ in terms of maturity and adoption, a non-exhaustive set of dependents of each library (*i.e.*, other libraries, clients, or applications) is enumerated.

Currently, 8 out of the 11 reviewed libraries implement the feature, whereas the remaining 3 only implement client-side connection migrations. For example, *quic-go* is a widely-used library that does not actively support server-side connection migration. *aiquic* parses the `preferred_address` field, but does not support active migration. Similarly, *Cloudflare quiche* does not support server-side connection migration. However, from the source code, it can be inferred that it is planned to implement the feature. On the other hand, several libraries already allow active server-side connection migrations and/or QUIC Multipath connections. For example, *picoquic*, *libquic*, *quinn*, *aiquic\_pisa*, *Google quiche*, *haproxy*, *Haskell quic*, and Alibaba's *xquic* support the feature. These implementations are used by several operating systems and user-space applications, including the Google Chrome browser.

Thus, based on these observations, it can be argued that server-side connection migration is not a hypothetical feature of QUIC but instead a relevant feature of the protocol. Therefore, the previously mentioned attack is exploiting a well-established feature for several implementations. However, it must be mentioned that there is no empirical evidence of the feature's prevalence in network traffic.

**Table 6: Overview of Statically Analyzed Implementations**

Implementation	Implemented	Notable Dependents
<i>quic-go</i> [37]	✗	cloudflared, caddy, synthing
<i>libquic</i> [8]	✓	goquic, chromium
<i>Cloudflare quiche</i> [7]	✗*	Cloudflare, NGINX
<i>picoquic</i> [34]	✓	Picotls, RIOT OS
<i>quinn</i> [38]	✓	h3-quinn, nestri, EasyTier
<i>aiquic</i> [23]	✗	airbyte, mitmproxy, envoy
<i>aiquic_pisa</i> [36]	✓	–
<i>Google quiche</i> [14]	✓	Google, Chromium
<i>haproxy</i> [16]	✓	Instagram, Airbnb, pfSense
<i>Haskell quic</i> [47]	✓	hprox, warp-quic, http3
<i>Alibaba xquic</i> [2]	✓	Taobao Mobile

\*Implementation of Server-Side Migration Planned

## B Wireshark Filters

The following Wireshark filters were used to extract the datasets for training the anomaly detectors:

1) Matching all outgoing QUIC Protected Payload packets in the testbed:

```
ip.src == 172.19.0.0/16 && quic && quic.
  header_form == "short header" && quic.
  header_form != "long header" && quic.dcid !=
  ""
```

**Listing 1: Wireshark Filter Example 1**

2) Matching all outgoing QUIC Protected Payload packets as well as all benign connection migration attempts (triggered using Cloudflare *quiche*):

```
ip.src == 172.19.0.X && ((quic && quic.header_form
  == "short header" && quic.header_form != "
  long header" && quic.dcid != "") || (udp &&
  udp.length == 1358))
```

**Listing 2: Wireshark Filter Example 2**

## C Additional Considerations

### C.1 Potential Drawbacks of Establishing New QUIC Connections

One of the primary challenges in executing a covert data exfiltration attack using the QUIC protocol arises from the transparency during the connection establishment phase. Middleboxes and fingerprinting tools typically filter connections based on the initial handshake packets. This initial packet exchange, which includes the “Initial” QUIC packet or potentially the “0-RTT” packet, serves as a clear indicator of new connection establishment.

These packets contain cleartext details, such as the TLS Client Hello and TLS Server Hello messages, that can expose a certain fingerprint. Given the amount of metadata within the handshake packets, any attempts to establish a new connection to exfiltrate data would likely increase the visibility of the attack. In particular, fingerprinting tools (*cf.* Section 6.3) filter exclusively based on handshakes and therefore increase the risk of the attack being identified and blocked at an early stage.

### C.2 Comparison with TLS and DNS-based Data Exfiltration

Defense systems specifically look for TLS Client Hello or TCP SYN packets to identify connection establishments. TCP-based connections typically require a new handshake to establish a valid connection when an underlying IP address changes. Similarly to QUIC, which allows connection migrations that change the underlying IP address *without* requiring a new handshake, there are further exceptions, such as the Stream Control Transmission Protocol (SCTP) [42], which can reconfigure IP addresses mid-connection using the *Set Primary* instruction. However, SCTP packets in non-telecom networks are not as prevalent as QUIC traffic, and, as a result, QUIC-based data exfiltration that mimics legitimate connection migrations potentially poses a greater security concern.

DNS-based data exfiltration may raise suspicion when multiple standalone DNS queries are produced, and not followed by a TCP and/or TLS handshake after an IP address has been resolved. This makes high-throughput DNS-based data exfiltration practically impossible without attracting attention. QUIC, since it inherently anticipates changes in the underlying IP header, may make data exfiltration appear less anomalous compared to other types of data exfiltration. Additionally, due to the high adoption of the QUIC protocol in popular web services [3], a QUIC-based data exfiltration attack may achieve high throughput without raising suspicion.

### C.3 Additional Insights from Leading Firewall Vendors

Firewall Vendor C mentioned that most requests regarding the QUIC protocol are coming from researchers, with only very little coming from industry. This indicates a divergence between academic interest in the protocol’s potential and the industry’s current level of adoption or need for it. Firewall Vendor D argues that the small performance gain through a reduced RTT does not justify the manageability challenges and potential security risks it entails. The vendors also correctly point out that tracking QUIC connections via a CID state table is not feasible, since the privacy-preserving mechanisms in QUIC may change CIDs at any time to prevent middleboxes from uniquely identifying a connection. A set of usable CIDs is negotiated as part of the encrypted QUIC handshake, and thus remains hidden from middleboxes.