Department of Informatics
Technical University of Munich

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

MASTER'S THESIS IN INFORMATICS

Analyzing QUIC in the Wild

Philippe Buschmann

Technical University of Munich

Department of Informatics

Master's Thesis in Informatics

Analyzing QUIC in the Wild

Analyse von QUIC im Internet

Author:	Philippe Buschmann
Supervisor:	Prof. Dr.-Ing. Georg Carle
Advisor:	Johannes Zirngibl, Patrick Sattler, Benedikt Jaeger
Date:	May 15, 2021

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Garching, May 15, 2021

Location, Date

Signature

Abstract

QUIC is a secure transport protocol originally developed by Google in 2012 and has been a draft of the Internet Engineering Task Force (IETF) since 2016. This draft defines QUIC as an alternative to TCP/TLS and as default for the Hypertext Transfer Protocol Version 3. Multiple companies such as Akamai, Cloudflare, Facebook and Fastly already adopted the IETF QUIC protocol and Google uses its derivation Google QUIC (gQUIC) for its services. At the time of writing, the IETF QUIC protocol is about to become the QUIC standard.

To assess the current state of IETF QUIC in the wild, we need a tool that scans for IETF QUIC. However, there is currently no tool for scanning IETF QUIC, since these tools either do not support QUIC at all or only support gQUIC.

In this work, we measure the support of IETF QUIC among IPv4 and IPv6 targets and collect supported versions, QUIC transport parameters, TLS parameters and TLS certificates of QUIC-capable servers. For this, we update an existing scanner to support IETF QUIC and implement a stateful scanner which is able to capture more information about a QUIC-capable target.

Our measurements show that most of the QUIC traffic originates from AS13335 (40.46%, Cloudflare), AS15169 (19.96%, Google), AS20940 (12.64%, Akamai) and AS54113 (11.53%, Fastly) with a strong correlation between Autonomous Systems (ASes) and supported versions. In addition, IETF QUIC is more widely supported than gQUIC (77.61% - 94.52% IETF QUIC, 36.22% - 51.24% gQUIC) but regardless, the support of both protocols is growing. Furthermore, the results of the stateful scans show a correlation between error messages and ASes. In general, stateful scans with Server Name Indication (SNI) were less error prone but also less divers in terms of AS origins because most of the traffic originates from AS13335 (Cloudflare). Additionally, we find that around 50% of the QUIC configurations (based on QUIC transport parameters) in combination with ASes have a low entropy and are suitable for fingerprinting. Furthermore, more than 99% of the scanned targets preferred the TLS cipher suite `0x1301`.

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Dr.-Ing. Carle who provided the topic. Further thanks go to my advisors Johannes Zirngibl, Patrick Sattler and Benedikt Jaeger who were always reachable for questions and provided great feedback. Additionally, I want to thank Juliane Aulbach who although not explicitly listed as an advisor participated in our weekly meetings and provided valuable feedback. Next, I would like to express my thanks to the Chair of Network Architectures and Services which provided additional scans of other protocols than QUIC and resources like a virtual machine.

Last but not least, I would like to thank my family and friends for their emotional support. Specifically, I want to thank my partner, who had my back and helped me putting my thoughts together.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

QUIC is the name of a transport protocol originally created by Jim Roskind at Google in 2012 [30]. At first, QUIC was an acronym for "Quick UDP Internet Connection" [30] which was later discarded after the Internet Engineering Task Force (IETF) QUIC working group started to work on QUIC in 2016 [18] and decided to use "QUIC" as the name for the protocol. Since then, more than 34 drafts have been published by the IETF group with the latest draft being version 34.

Since Google and the IETF worked simultaneously on QUIC [10], the IETF draft of QUIC differs from the Google QUIC (gQUIC) implementation. In this thesis, we focus on the IETF QUIC draft and refer to it as QUIC.

According to the IETF draft of QUIC [19], QUIC is an alternative to Transmission Control Protocol (TCP)+Transport Layer Security (TLS) and is going to be used as transport protocol in the Hypertext Transfer Protocol Version 3 (HTTP/3) [6]. At the time of writing, the QUIC protocol is still a draft of the IETF working group and not yet a RFC standard. However, this may change in the near future. The latest draft (version 34) defines the official QUIC version [19] and developer should wait with the implementation of this draft until it becomes an IETF standard.

Moreover, QUIC and its derivation gQUIC are already widely in use. For example, gQUIC carries over a third of the traffic of Google [10]. Facebook reports that QUIC is responsible of over 75% of Facebook's traffic [20]. Furthermore, Content Delivery Networks (CDNs) like Cloudflare [26], Akamai [34, 5] and Fastly [17] provide support for IETF QUIC in their services. In contrast, originally only Akamai and Google supported QUIC [31].

On the 16th April 2021, Firefox officially announced QUIC and HTTP/3 support in Firefox Nightly and Beta [9]. Before that, Chrome already officially announced gQUIC and IETF QUIC support for their browser [10]. These adoptions of QUIC simplify the use of QUIC and depending on the configuration, enable QUIC by default if supported by the server.

With the official standard of QUIC about to be released, we want to assess the state of QUIC in the wild. However, it is not possible to measure QUIC at the time of writing since tools either do not support IETF QUIC yet [3] or were only released for gQUIC [31].

Our main goal is to measure the support of QUIC among a selected target group e.g., Alexa Top 1M, and collect the supported versions of servers which are QUIC-capable. In addition, we want to collect information about the configuration of a QUIC server.

This thesis provides the implementation of a stateless scanner and a stateful scanner. The stateless scanner leverages a version negotiation and collects QUIC-capable targets and their supported versions. The stateful scanner establishes a connection with the QUIC-capable targets and collects the QUIC transport parameters, TLS parameters and TLS certificates.

We scanned targets in IPv4 and IPv6. The scans yielded the following results:

- Most QUIC-capable targets originate from Autonomous Systems (ASes) of Cloudflare, Google, Akamai, Fastly and Facebook (largest to smallest share). In addition, there is a strong correlation between ASes and supported versions.
- IETF QUIC is more widely supported than gQUIC with Akamai changing its support from only gQUIC to gQUIC and IETF QUIC. In total, the use of both protocols increases.
- Stateful scans with Server Name Indication (SNI) are less error prone than scans without SNI. In addition, SNI scans have a more dominant share of Cloudflare while scans without SNI are more divers.
- Half of the QUIC configurations derived of QUIC transport parameters can be used to infer the AS of a target.
- The majority of targets only prefer the mandatory cipher suite of TLS 1.3.

In this thesis, we introduce QUIC and provide an overview of its design in Section 2. Next, we describe our stateless and stateful scanner as well as our measurement setup in Section 3. Then, we present and evaluate the results of our scans in Section 4. In Section 5, we put our results in the context of related work. Last, we conclude our work and describe future work in Section 6.

Chapter 2

Background

In this chapter, we give an overview of QUIC, its functionality and available parameters which we use to evaluate the deployment of QUIC in the wild. Since QUIC is an alternative to TCP and TLS, we start with a comparison between TCP and QUIC. Then, we describe QUIC in more detail according to its current draft [19]. We cover the versioning, the structure of packets and frames, the handshake, the encryption and the header protection.

2.1 QUIC as Alternative to TCP+TLS

Hypertext Transfer Protocol Version 2 (HTTP/2) uses TCP as transport protocol and TLS as security layer. Instead of re-using the previous standard, HTTP/3 introduces a new stack and uses User Datagram Protocol (UDP) with QUIC. We compare the International Organization for Standardization (ISO)-Open Systems Interconnection (OSI) layers of a HTTP/3 and HTTP/2 stack to give an overview of the differences of both in Figure 2.1.

In the HTTP/2 stack on the left, each layer has its own function: TCP handles congestion and ensures a reliable data stream, TLS encrypts and decrypts data, while HTTP/2 enables stream multiplexing. In contrast, the HTTP/3 stack on the right changes the approach to split functionality into layers. By using UDP, the HTTP/3 stack uses a smaller UDP datagram instead of a larger TCP segment but loses e.g., the ability to ensure a reliable data transfer. Therefore, QUIC needs to perform these operations such as controlling congestion and ensuring reliable data streams. Additionally, QUIC incorporates the TLS protocol and can handle multiple data streams over one connection. Thus, HTTP/3 does not implement stream multiplexing like HTTP/2.

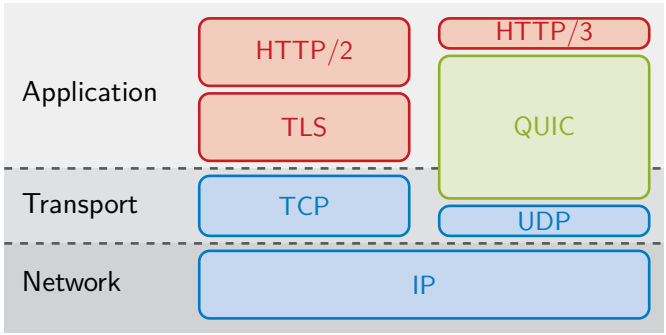


Figure 2.1: Simplified Representation of a HTTP/2 and HTTP/3 Stack

In addition, the QUIC implementation resides in user space instead of the kernel space like TCP and TLS. Therefore, QUIC may lack kernel optimizations in the future but can be updated frequently and specialized for a use-case [31].

2.2 QUIC Versions

The implementation of QUIC in user space enables frequent updates to be rolled out without updating the kernel. Therefore, each release of new changes to the QUIC protocol introduces a new version. These versions may be specific for a capability, the developing company, the library used or a draft of the protocol. In addition, versions might not be compatible with each other. For example, Google started with the development of QUIC and continued working on their own version of it, even after the IETF working group began to standardize it [10]. This version named gQUIC is not compatible with the QUIC protocol developed by the IETF working group.

The distribution of QUIC versions is first-come, first-serve [15] and recorded in the Github repository of the IETF working group [15]. This list is constantly being expanded. Table 2.1 shows an overview of the registered versions and our acronyms for each version. We use these acronyms in our results in Chapter 4. According to the IETF specification [19], the versions from `0x00000001` to `0x0000ffff` are reserved for the IETF QUIC standard and from `0xff000000` to `0xffffffff` for the IETF draft. Most versions reserved by Google are used for different iterations of gQUIC. Other versions like `0xfaceb000` indicate the company (in this case Facebook) and/or the library which uses them (in this case mvfst).

Any version that matches the pattern `0x?a?a?a?a` with `?` representing any single-digit hexadecimal number from `0` to `f` is only used for the QUIC version negotiation. A client may use these versions in an initial request to a server to force a version negotiation, while servers include these versions in their list of supported versions to ensure a working

Version	Owner	Notes	Acronym
0x00000000	n/a	Reserved as invalid	invalid
0x0000[0001-ffff]	IETF	Reserved for standard	ietf-[01-ff]
0xff[000000-ffffff]	IETF	Reserved for draft	draft-[01-ff]
0x?a?a?a?a	IETF	Version negotiation	
0x51303[0-5]3[1-9]	Google	gQUIC v1-59	Q0[01-59]
0x51303939	Google	gQUIC v99	Q099
0x5430343[8-9]	Google	gQUIC+TLS v48-49	T0[48-49]
0x5430353[0-9]	Google	gQUIC+TLS v50-59	T0[50-59]
0x54303939	Google	gQUIC+TLS v99	T099
0x50524f58	Google	Proxied QUIC	prox
0xfaceb00[0-f]	Facebook	Library mvfst	mvfst-[0-f]
0xabcd000[0-f]	Microsoft	Library MsQuic	msq-[0-f]
0xf123f0c[0-f]	Mozilla	Library MozQuic	mozq-[0-f]
0x454747[00-ff]	NetApp	Library Quant	netapp-[00-ff]
0x51474f[00-ff]	quic-go	Library quic-go	quicgo-[00-ff]
0x91c170[00-ff]	quicly	Library quicly	quicly-[00-ff]
0x50435130	Private Octopus	Library picoquic	picoquic
0xf0f0f1f[0-f]	Telecom Italia	Measurability experiments	tcit-[0-f]
0xf0f0f0f[0-f]	ETH Zürich	Measurability experiments	ethzue-[0-f]

Table 2.1: Overview of QUIC Versions and Chosen Acronyms [15]

version negotiation [19]. We use the forced version negotiation but we do not define an acronym for these versions because we exclude them from our results.

From 2019 to 2021, the IETF working group released seven to nine drafts/versions per year. At the time of writing, the IETF QUIC standard is not yet in use and the most up to date draft is version 34. We always refer to the draft version 34 when explaining QUIC if not mentioned otherwise.

2.3 QUIC Packets and Frames

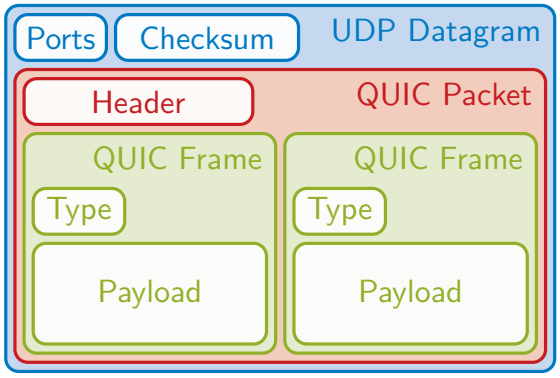


Figure 2.2: Overview of a QUIC Packet with a Header and Multiple Frames

In this section, we describe the QUIC packet structure. According to the IETF QUIC draft [19], the payload of an UDP datagram is called a (QUIC) packet. A packet consists of a header and one or more frames. Each frame has a type and contains a payload. The general structure can be seen in Figure 2.2.

QUIC differentiates between long and short header packets. The long header specifies whether a packet is of type *Initial*, *0-RTT*, *Handshake*, *Retry* or *Version Negotiation*. QUIC uses long header packets for key and connection establishment as well as version negotiation. Only after negotiating the *1-RTT* keys, QUIC can use short header packets which can only be of type *1-RTT*. We describe the use of headers and packet types in Section 2.3.1.

Each header contains at least one connection ID, the destination connection ID, to keep track of the shared state of a QUIC connection between two endpoints. QUIC uses these connection IDs to relay the packet to the other endpoint even after an underlying protocol changes the IP address or UDP port [19].

The type of header and packet limits the selection of frame types. For example, *Version Negotiation* packets cannot contain any frames, while *1-RTT* packets can contain every

Packet Type	Frame Type
Version Negotiation	None
Retry	None
Initial	ACK, CONNECTION_CLOSE, CRYPTO, PADDING, PING
Handshake	ACK, CONNECTION_CLOSE, CRYPTO, PADDING, PING
0-RTT	CONNECTION_CLOSE, CRYPTO, PADDING, PING, STREAM (..)
1-RTT	ACK, CONNECTION_CLOSE, CRYPTO, PADDING, PING, STREAM (..)

Table 2.2: Permitted Combination of Frames and Packets

possible frame type of the specification. *Initial* and *Handshake* packets can only contain ACK, CRYPTO, PADDING, PING and CONNECTION_CLOSE frames which we explain in detail in Section 2.3.2. Therefore, *Initial* and *Handshake* packets can only acknowledge packets (ACK), negotiate keys (CRYPTO), contain padding (PADDING), ping an endpoint (PING) and close connections (CONNECTION_CLOSE). Table 2.2 shows an overview of the allowed frame types for each packet type. In our research, we focus on *Version Negotiation*, *Initial* and *Handshake* packets.

QUIC uses *Initial* and *Handshake* packets with CRYPTO frames to negotiate *1-RTT* keys and establish a secure connection. For this, QUIC incorporates TLS to generate keys and encrypt its payload. In addition to encrypting a packet, QUIC protects the header and the payload of a packet. We examine the encryption and protection in Section 2.3.3.

2.3.1 Packet Types

The packet type depends on the type specified in its header. The type of a header is specified by the Header Form (HF) bit, which is the most significant bit of a QUIC packet. HF is equal to one for long header packets and zero for short header packets. Long header packets contain type-specific bits with which the endpoint differentiates between *Initial*, *0-RTT*, *Handshake* and *Retry* packets. *Version Negotiation* packets also use a long header but do not specify the type-specific bits.

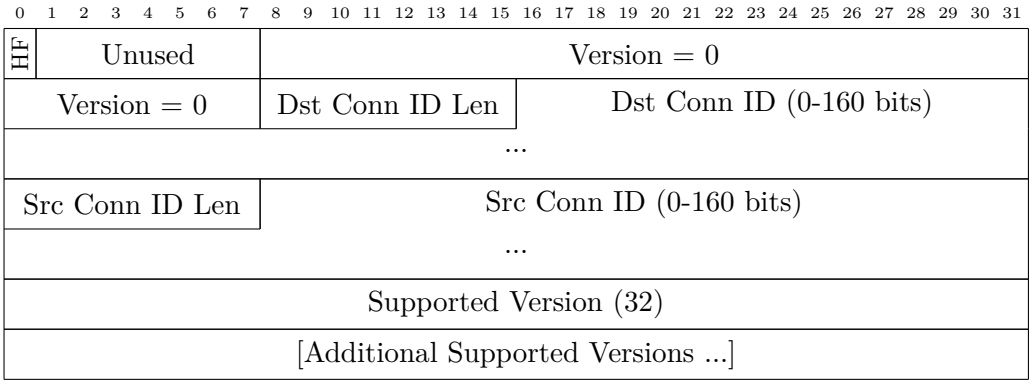


Figure 2.3: *Version Negotiation* Packet Structure

First, we describe *Version Negotiation* packets, followed by *Initial* and *0-RTT* packets. Then, we explain the use and structure of *Handshake* and *Retry* packets. Last, we analyze *1-RTT* packets.

Version Negotiation Packets

QUIC implements a version negotiation using *Version Negotiation* packets [19], because the supported versions of a client and a server may differ. When a client initiates a new connection with a version which is not supported by the server, the server responds with a *Version Negotiation* packet. The server can limit the number of *Version Negotiation* responses and should respond with an unsupported version even if the server is not able to decrypt the packet and its payload. A server may not respond if a packet is not sufficiently large (minimum 1200 bytes as explained in Section 2.3.1 - *Initial* packets). An endpoint is not allowed to respond to a *Version Negotiation* packet with another *Version Negotiation* packet to avoid infinite loop of *Version Negotiation* packets. A version negotiation can be forced by using the QUIC version `0x?a?a?a?a` with `?` representing any single-digit hexadecimal number from `0` to `f` as seen in Section 2.2. In our research, we use a forced version negotiation to capture all of the supported version of a server.

Figure 2.3 shows the structure of a *Version Negotiation* packet. The *Version Negotiation* packet does not include a version in the version field which is therefore set to zero. However, a server responding with a version negotiation includes all its supported version as additional fields at the end of the packet.

Initial Packets

To initiate a connection, QUIC uses *Initial* packets with a CRYPTO frame containing a TLS Client Hello [19]. Since the specification limits the frame types used by an *Initial* packet, it can only ping an endpoint, negotiate keys, acknowledge packets and

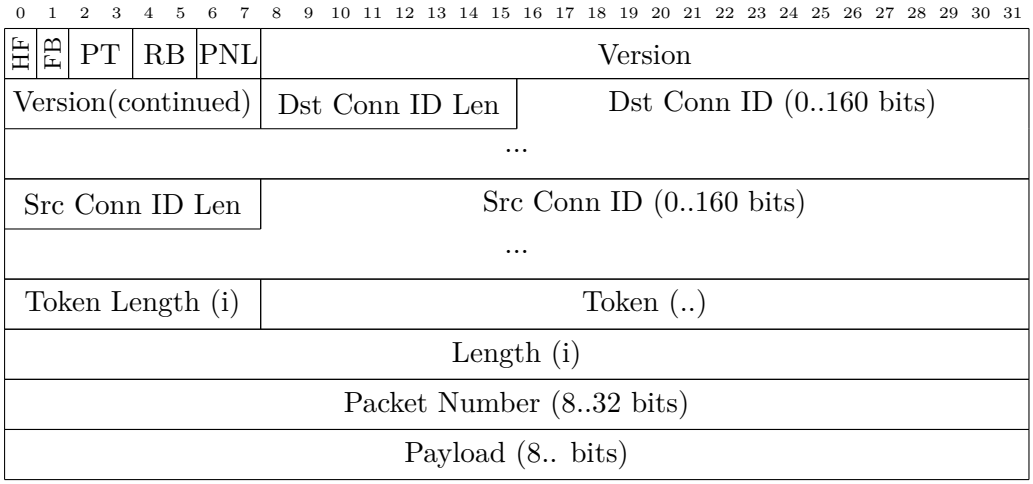


Figure 2.4: *Initial* Packet Structure

close connections. Without handshake keys, a client must increase the size of an *Initial* packet to at least 1200 bytes with a PADDING frame. This reduces the effectiveness of amplification attacks and ensures a reasonable Path Maximum Transmission Unit [19].

Figure 2.4 depicts the structure of an *Initial* packet. Because of its long header the Header Form (HF) and Fixed Bit (FB) are set to one. To allow QUIC to coexist with other protocols, the FB is set to one; otherwise, the packet must be discarded. For *Initial* packets the Packet Type (PT) is set to zero. The Reversed Bits (RB) must be zero before header protection. The Packet Number Length (PNL) specifies the $length + 1$ of the Packet Number field in bytes. In other words, if the PNL is zero, the Packet Number field has a length of one byte. Version defines the QUIC version the endpoint wants to use.

The length of QUIC connection IDs is variable. For that reason, Dst Conn ID Len (destination connection ID length) and Src Conn ID Len (source connection ID length) specify the length of Dst Conn ID (destination connection ID) and Src Conn ID (source connection ID) in bytes. The value of the length field of both connection IDs can be zero depending on the packet type but must not exceed 20. The last premise may change in future QUIC versions [19]. If the length of the connection ID is 0, the connection ID field length is 0 and the field is therefore removed from the header.

On connection establishment, the QUIC server might send a *Retry* packet with a token and has to discard all subsequent *Initial* packets without a token to validate the client's address. If a server requests a token, a client needs to set Token Length and Token to the according values. If the server does not provide a token, Token Length is set to zero and the Token field can be removed from the header.

Length specifies the length of the payload in bytes including the length of Packet Number. QUIC uses this number to acknowledge packets with ACK frames.

0-RTT Packets

0-RTT packets are almost interchangeable to *Initial* packets. A client can initiate a connection by sending an *Initial* packet or a *0-RTT* packet. In contrast to *Initial* packets, *0-RTT* packets use the Packet Type (PT) 0x1 and include early data to the server before TLS handshake completion. Therefore, *0-RTT* can e.g., also contain STREAM frames in comparison to *Initial* packets. QUIC uses STREAM frames to transfer data between endpoints, e.g., the HTTP request and response. To establish a *0-RTT* connection, we need to store previously negotiated keys.

However, we do not have access to these keys on a first request to the server. In addition, QUIC server may buffer *0-RTT* packets even if no connection could be established. For this reason and because of our focus on connection establishment as opposed to data transmission, we do not cover *0-RTT* packets in detail.

Handshake Packets

Handshake packets are structurally equal to *Initial* packets but without the Token Length and Token field. *Handshake* packets use the Packet Type (PT) 0x2. After a client receives its first *Handshake* packet from a server which usually contains a CRYPTO frame with encrypted extensions or certificates, it responds with *Handshake* packets. *Handshake* packets must adhere to the same limits as *Initial* packets when using frame types [19]. Thus, only ACK, CRYPTO, PADDING, PING, and CONNECTION_CLOSE frame types are allowed. QUIC server use *Handshake* packets to transmit CRYPTO frames including encrypted extensions, certificates and/or handshake finished messages after a Server Hello message.

Retry Packets

Figure 2.5 shows the structure of a *Retry* packet. It is structurally similar to an *Initial* packet and uses the Packet Type (PT) 0x3. In comparison to an *Initial* packet, the *Retry* packet does not use reserved bits and a packet number. Additionally it does not use header protection and only secures the integrity of the Retry Token by generating a Retry Integrity Tag. A server may respond to an *Initial* packet with a *Retry* packet to perform address validation. If a client receives a *Retry* packet, it must use the Token in all subsequent *Initial* packets.

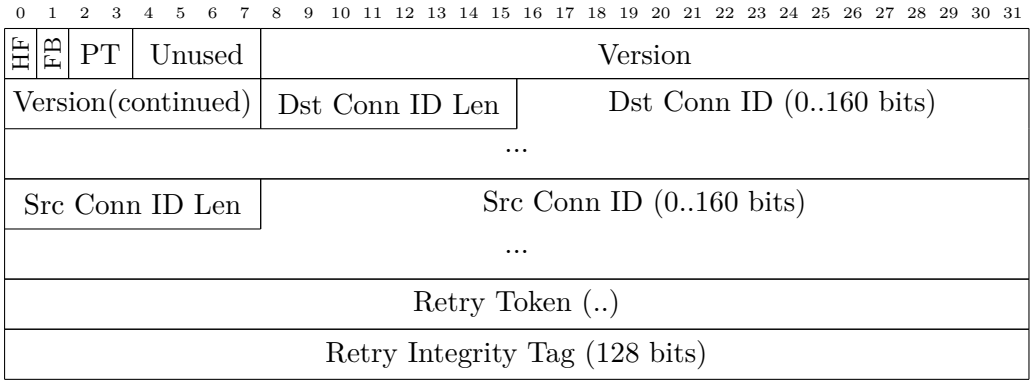


Figure 2.5: *Retry* Packet Structure

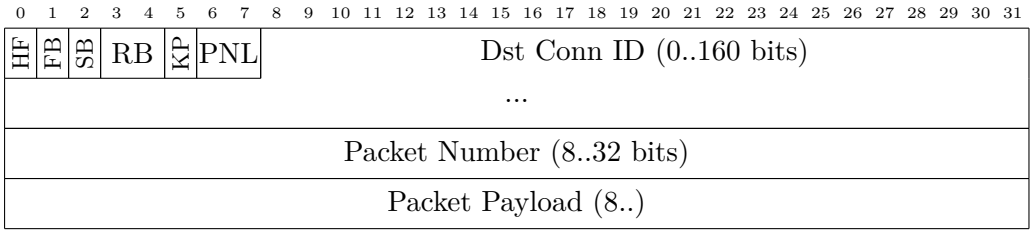


Figure 2.6: Short Header Structure

1-RTT Packets

1-RTT packets are short header packets which are available after establishing the *1-RTT* keys. To negotiate these keys, at least one round-trip between two endpoints is necessary: the current connection establishment for *Initial* packets or the last connection for *0-RTT* packets. Therefore, QUIC defines these packets as *1-RTT* packets.

The structure can be seen in Figure 2.6. In contrast to the long header, the short header starts with the Header Format (HF) set to zero. Next, the Fixed Bit (FB) is set to one and the Reserved Bits (RB) are set to zero. The short header introduces the latency Spin Bit (SB), which usage is optional but allows latency monitoring by observation endpoints [11]. Additionally, short headers identify changed packet protection keys with the Key Phase (KP). KP is initially set to zero and then toggled to indicate a key update. When using *1-RTT* packets, endpoints used and specified their destination connection ID and its length in previous *0-RTT*, *Initial* and *Handshake* packets. For this reason, short headers only have the destination connection ID field and are missing the length of the destination connection ID.

2.3.2 Frame Types

The frame depends on and is limited by the type of packet. Since we focus on the connection establishment and not on data transfer, we only shortly describe the function

of ACK, CRYPTO, PADDING, PING and CONNECTION_CLOSE frames. A frame in general contains a type and a payload depending on the type.

The receiver of a packet uses ACK frames to signal the sender which packets were received. ACK frames use the value of the packet number to identify packets. Thus, *Version Negotiation* and *Retry* packets cannot be directly acknowledged (but implicitly via the next *Initial* packet sent).

QUIC endpoints use CRYPTO frames to transmit cryptographic handshake messages. In other words, CRYPTO frames contain TLS messages like Client or Server Hello, Encrypted Extensions and Certificates.

PADDING frames have no content (filled with zeros) and only serve the purpose to increase the packet size to at least 1200 bytes.

To ensure that the peer is still alive, an endpoint can send PING frames. As response, the other endpoint should acknowledge the previously sent PING packet.

By sending CONNECTION_CLOSE frames, an endpoint signals a peer that the connection is being closed. Then, the sender enters a closing state and the receiver enters a draining state. These states should last at least three times the probe timeout and allow connections to discard any delayed or reordered packets. After the closing or the draining state ends, an endpoint should discard all state related to the connection.

2.3.3 Encryption and Protection

As mentioned in Section 2.1, the TCP+TLS stack separates the functionality into layers. This way, TCP provides a reliable connection between endpoints and TLS provides a secure channel over an untrusted medium. In contrast to TCP+TLS, the QUIC stack incorporates TLS handshake and alert messages directly and bears the responsibility of the TLS record layer [19]. In other words, QUIC uses parts of the TLS interface version 1.3 or higher, while TLS uses the reliable data connection and the record layer of QUIC.

As explained in Section 2.3.1, QUIC can use previously gathered information about a connection to send a *0-RTT* packet. This packet includes encrypted application data protected by Early Data (*0-RTT*) keys. Therefore, *0-RTT* bare the risk of a replay attack. Since we do not focus on *0-RTT* packets, we do not describe this process in detail. However, the QUIC protocol defines this process in its specifications [19, 33].

Our focus is on *Initial*, *Handshake*, *Retry* and *Version Negotiation* packets and their payload and header protection. Each packet type has its specific encryption key. *Version Negotiation* packets do not use any cryptographic protection. *Retry* packets use e.g., the AEAD_AES_128_GCM algorithm [33] to generate a 128 bit long Retry Integrity

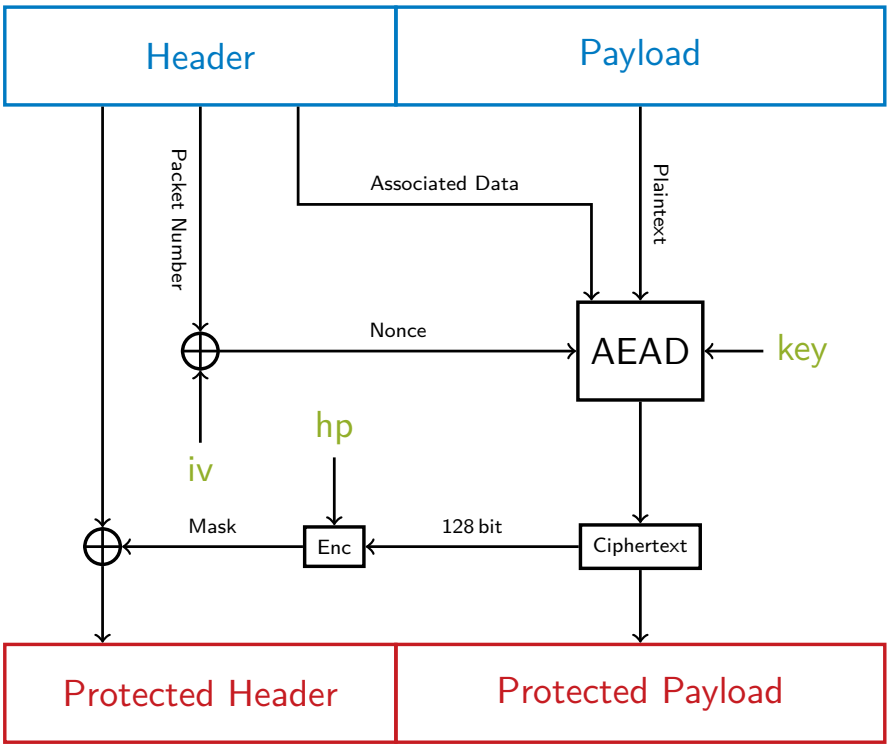


Figure 2.7: Overview of the Header and Payload Protection in QUIC

Tag and to protect the integrity of the Retry Token. QUIC protects *Initial* packets with initial secrets and encrypts *Handshake* packets with the negotiated TLS handshake keys. QUIC uses the resulting *1-RTT* keys to encrypt application data in short header packets.

QUIC encrypts the payload/frames of a QUIC packet with the *1-RTT* keys if available. Additionally to payload encryption, QUIC protects part of the header. The fields are specific for the type of header. If defined in the header, QUIC protects the Reserved Bits (RB), the Key Phase (KP), the Packet Number and the Packet Number Length (PNL). The header protection should prevent the correlation of a packet number with an activity [19]. In addition, it should prevent the injection of packets with valid packet numbers by attackers which cannot observe the network [19].

The encryption and protection is displayed in Figure 2.7. First, the nonce is calculated by XOR'ing the Packet Number and the initialization vector (`iv`). Then, QUIC uses the Authenticated Encryption with Associated Data (AEAD) algorithm described in RFC5116 [24] with the payload as plaintext, the header as associated data, the nonce and the encryption `key` to calculate the cipher text. This cipher text is the resulting protected payload. Next, the first 128 bits of the ciphertext are encrypted with the

header protection key (`hp`) using the Advanced Encryption Standard (AES) cipher in Electronic Code Book (ECB) mode resulting in a mask. This mask is then used to receive the protected header by XOR'ing the mask with the unprotected header fields (RB, [KP], PNL, Packet Number).

We show the calculation of the variables `iv`, `key` and `hp` in Algorithm 1. The QUIC specification defines the different labels and the salt. While the values of the labels currently stay constant throughout the QUIC versions, the salt should change in between the version. In this case, the salt `0xafbfec289993d24c9e9786f19c6111e04390a899` is valid for QUIC from **draft-1d** to **draft-20**. **draft-21** and **draft-22** use the salt `0x38762cf7f55934b34d179ae6a4c80cadccbb7f0a`. QUIC uses the version-specific salt, the connection ID and the HMAC-based Extract-and-Expand Key Derivation Function (HKDF) [22] to generate secrets with `hkdf_extract` and `hkdf_expand`.

Algorithm 1 Calculating `iv`, `key` and `hp` for *Initial* packet

Require: Connection ID `connection_id`

```
salt = 0xafbfec289993d24c9e9786f19c6111e04390a899
label_client_in = 0x00200f746c73313320636c69656e7420696e00
label_quic_key = 0x00100e746c7331332071756963206b657900
label_quic_iv = 0x000c0d746c7331332071756963206976 97600
label_quic_hp = 0x00100d746c7331332071756963206876 87000
initial_secret = hkdf_extract(salt, connection_id)
client_initial_secret = hkdf_expand(initial_secret,
info=label_client_in, length=32)
key = hkdf_expand(client_initial_secret, info=label_quic_key,
length=16)
iv = hkdf_expand(client_initial_secret, info=label_quic_iv,
length=12)
hp = hkdf_expand(client_initial_secret, info=label_quic_hp,
length=16)
```

Ensure: `iv`, `key`, `hp`

2.4 QUIC Connection

To put the previous sections into context, we give an overview of the QUIC connection establishment and teardown by showing an example of a QUIC connection in Figure 2.8. Since we focus on connection establishment and not data transfer, the example starts with and uses *Initial* packets instead of *0-RTT* packets.

The first two packet transmissions are not mandatory but occur if a client tries to use a version in an *Initial* packet which is unsupported by a server. The client then

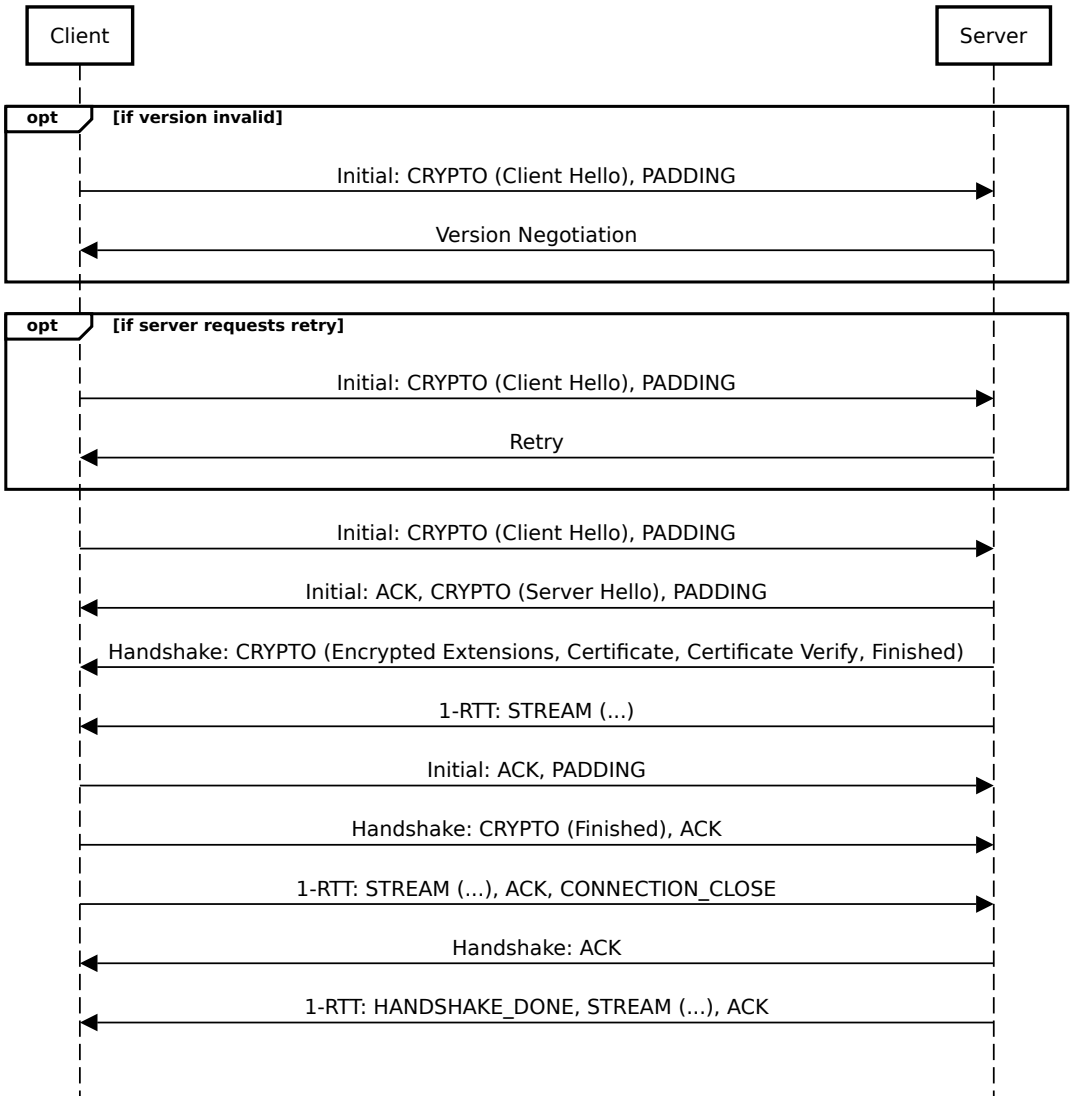


Figure 2.8: QUIC Handshake in Detail

receives a *Version Negotiation* packet including supported versions to choose from. If the client does not support any version included in the *Version Negotiation* packet, the connection establishment fails. This should be stateless but it also depends on the implementation of QUIC used by server. If the client supports a version of the *Version Negotiation* packet, it can use this version in an *Initial* packet to retry the connection establishment.

After sending an *Initial* packet with a valid and supported version, the server may respond with a *Retry* packet to perform address validation. A client that receives a *Retry* packet, must include the received Token in subsequent *Initial* packets.

Then, the client sends a valid *Initial* packet with a CRYPTO frame including a TLS Client Hello and PADDING frame. The PADDING frame ensures that the *Initial* packet is at least 1200 bytes long. If the *Initial* packet is shorter, an endpoint should not reply according to the specification [33].

Next, the server responds with an *Initial* packet. This packet contains an ACK frame to acknowledge the client *Initial* packet, a CRYPTO frame including a Server Hello and a PADDING frame. In addition, the server sends one or more *Handshake* packets with CRYPTO frames containing the following TLS messages: Encrypted Extensions, Certificate, Certificate Verify and Finished. Furthermore, the server may start to send an *1-RTT* packet including data (STREAM frame).

As a response, the client sends an *Initial* packet which acknowledges the *Initial* packet of the server. Additionally, the client sends a *Handshake* packet with an acknowledgement of the previous *Handshake* packet and a CRYPTO frame including a TLS Finished message. Further, it may sent an *1-RTT* packet with an acknowledgement (ACK frame) and data (STREAM frame). In our case, we want to close the connection without sending more data. This is done by including a CONNECTION_CLOSE frame inside the *1-RTT* packet.

Last of all, the server responds with an acknowledgement of the previous *Handshake* packet and the previous CONNECTION_CLOSE.

Since we control the messages of the client and want to decrease the load on the server as much as possible, we try to terminate the initiated session as soon as possible. The current draft of QUIC [19] describes three ways to terminate a connection: idle timeout, immediate close and stateless reset.

An idle timeout occurs if one peer stays idle for longer than the established maximum idle timeout. This maximum idle timeout is advertised by either the server or the client as the transport parameter `max_idle_timeout` in the TLS extensions, if the parameter

is neither omitted or zero. If activated by at least one endpoint, both select the minimum valid (non-zero) value of the `max_idle_timeout` as their maximum idle timeout. Upon receiving and processing a packet successfully or sending an ack-eliciting packet, the endpoint resets its timeout timer. If an endpoint advertises a maximum idle timeout, it commits to terminating the connection with an immediate close prior to the idle timeout.

To clear the state of an endpoint and to trigger an immediate close of a connection, an endpoint can send the `CONNECTION_CLOSE` frame which closes all streams. The sending endpoint enters the closing state, which means that it stores enough information to respond with a `CONNECTION_CLOSE` frame to incoming packets of the same, already closed connection. The receiving endpoint enters the draining state, which is identical to the closing state but the endpoint can only send at maximum one packet containing a `CONNECTION_CLOSE` frame before entering the state and must not send any further packets after entering the state. The `CONNECTION_CLOSE` frame should use the highest level of packet protection available to ensure that the endpoint receives and processes the packet correctly.

The third possibility to close a connection is the stateless reset. The stateless reset should only be used if an endpoint is not able to process packets of a connection due to an outage or crash and is not able to send a `CONNECTION_CLOSE` frame to clear the state of its peer. Otherwise, the endpoint must use a `CONNECTION_CLOSE` frame.

In our case, we decide against only using an idle timeout since this would leave the server waiting for the timeout of our connection. Since we want to solely terminate the connection for which we have the connection information, we must not use a stateless reset. Otherwise, we would break the requirements of the QUIC specification. This leaves us with the only choice of an intermediate close.

Chapter 3

Methodology

In this thesis, we aim to identify QUIC-capable servers on the Internet and collect information about them. First, we describe which details we collect about a scanned server and how these details can be retrieved. Then, we introduce the stateless and the stateful scanner, their functionality, and their underlying implementation. After that, we explain the challenges of both scanners. Last but not least, we describe the ethical guidelines for our scans.

3.1 Collected Information

We capture the following information about a server:

1. QUIC capability
2. Supported versions of QUIC
3. Transport parameters of QUIC
4. Configuration of TLS
5. TLS Certificates

First, we identify QUIC-capable server. In Section 2.2 and 2.3.1, we describe the use of an invalid version (`0x?a?a?a?a`) with an *Initial* packet to trigger the QUIC version negotiation. After studying the specification [19], we identified that this approach uses the fewest number of requests and has the smallest request size necessary to receive an answer from a QUIC-capable server.

If the server receives an *Initial* packet, the server must process the version first, before verifying the packet protection and the payload of the packet [19]. As a result, we do not have to protect the header and payload of the *Initial* packet and the server does not have to decrypt the packet to read the version negotiation version. This reduces the load on the server and should be stateless since the server does not need to store information about the connection for later requests. However, a server could potentially store additional information to limit the number of responses to the same client. This depends on the implementation and we describe this behavior in Section 3.4.

We send one *Initial* packet with a version negotiation version (`0x?a?a?a?a`) per target in a subnet for each scan. If a server responds to this request, it might not necessarily be QUIC-capable. It can also be an echo server, which mirrors our request. Since these servers could lead to false positives, we drop each response with a size of less than size of the *Version Negotiation* header and each packet that does not use our destination connection ID. Additionally, we check if the version of the *Version Negotiation* packet is `0x00000000` which is necessary according the QUIC specification [19].

Next, we collect the supported versions of a server. Since we receive a *Version Negotiation* packet from a server to verify if it supports QUIC, we already receive the supported versions as payload of the packet as explained in Section 2.3.1. Therefore, we do not need to make additional requests to collect this data.

We can only collect the transport parameters of QUIC, the TLS configuration and TLS certificates if we establish a QUIC connection. On successful establishment, we receive an *Initial* packet with a TLS Server Hello message and one or multiple *Handshake* packets with encrypted extensions and certificates. From the Server Hello message, we extract the QUIC transport parameters of a server and the TLS configuration. The TLS configuration includes e.g., the available cipher suites and the signing algorithms. From the *Handshake* packet, we store the certificates as is. To reduces the load on the server, we avoid additional HTTP/3 GET requests and close the connection with a CONNECTION_CLOSE frame.

This approach is stateful since the server has to store the information about the connection, e.g., the length of the connection ID for *1-RTT* packets, the encryption and decryption keys. In contrast to the stateless approach, the server uses encryption and header protection as explained in Section 2.3.3 and thus, needs more resources to communicate with its peer. Overall, the traffic volume of the stateful approach is larger for each target.

To avoid any unnecessary load on the server and to be as resource-saving as possible, we split the stateless and stateful approach into a stateless and a stateful scanner. The

stateless scanner uses the forced version negotiation and the stateful scanner establishes and terminates a connection to known targets.

3.2 Stateless Scanner

The stateless scanner is built on ZMap. ZMap is a fast single packet network scanner that is capable of scanning the entire IPv4 address space in under five minutes given a 10 Gigabit Ethernet Internet connection [1]. ZMap uses probe modules to scan a specific protocol and offers e.g., stateless TCP and UDP scans. We extend its functionality by adding a probe module which is capable of forcing a version negotiation.

3.2.1 Implementation

Originally, Rüth et al. [31] created a probe module for ZMap that detects QUIC-capable server. This module sends packets with the first version of gQUIC (acronym `Q001`), a Client Hello message and padding. Rüth et al. wrote this probe module for a version of QUIC which was then still developed by Google and is now known as gQUIC. Therefore, the packets resemble *1-RTT* packets which cannot be used to initiate a connection to a server. As a result, the module fails to detect IETF QUIC-capable servers. We use the structure of this module and update it in accordance to the IETF QUIC specification [19].

Specifically, our scanner forces a version negotiation to obtain a *Version Negotiation* packet as response. The forced version negotiation uses an *Initial* packet with version `0x1a1a1a1a`, padding to increase the size of the packet to 1200 bytes, no encryption and no Client Hello message. The encryption and Client Hello message are not necessary according to the QUIC specification [19]. We created an additional probe module with the encryption and a TLS Client Hello message. We tested both scanners with test servers and verified that the responses were the same. As explained in Section 3.1, we filter incoming responses based on their length, their structure, their version and their destination connection ID and drop all invalid responses.

As input, the ZMap QUIC probe module needs a port, an address and a netmask. Additionally, the module includes an option to deactivate the padding of the *Initial* packet to scan for servers that do not adhere to the specification [19] and respond to *Initial* packets with a size of less than 1200 bytes. Furthermore, we can choose the rate of transmitted packets per second. As a result, we receive the IP addresses of QUIC-capable servers and their supported versions. In this thesis, we also refer to the set of supported versions as version group.

Date	IP Version	Port	Padding	Total Targets	Packet Rate
2021-01-21	IPv4	443	yes	2.88B	5000
2021-02-03	IPv4	443	yes	2.90B	8000
2021-02-17	IPv4	443	yes	2.91B	10 000
2021-03-03	IPv4	443	yes	2.91B	10 000
2021-03-09	IPv4	443	no	2.15M	10 000
2021-03-17	IPv4	443	yes	2.91B	10 000
2021-04-06	IPv4	443	yes	2.90B	15 000
2021-04-14	IPv4	443	yes	2.94B	15 000
2021-04-20	IPv4	443	yes	2.97B	15 000
2021-04-24	IPv4	8443	yes	3.02B	15 000
2021-03-12	IPv6	443	yes	12.04M	8000
2021-04-16	IPv6	443	yes	4.09M	2000
2021-04-24	IPv6	443	yes	21.47M	2000

Table 3.1: Configuration of Stateless Scans

3.2.2 Test Setup

Before using the stateless scanner for measurements in the wild, we tested the stateless scanner with a local server and the test servers listed on the repository of the IETF working group [14]. Some of the public test servers were missing domain name entries and were not available, but we successfully received the *Version Negotiation* packets from the available servers.

3.2.3 Configurations of Stateless Scans

Table 3.1 shows the configurations of our measurements in the wild. Most of the times, we scan on port 443 with padding. In this context, padding and no padding describes whether the *Initial* packet includes enough padding to have a size of at least 1200 bytes. To check if servers comply with the QUIC specification, we scan without padding on 2021-03-09.

We expect the highest response rate of QUIC servers on port 443, since HTTP/3 uses QUIC as underlying protocol [6] and there is a strong convention to transfer encrypted Hypertext Transfer Protocol (HTTP) over port 433. However, we found that some QUIC services use port 8443, if not configured otherwise [32]. Therefore, we also scanned port 8443 on 2021-04-24.

We refer to port 443 with padding as the default configuration, to port 8443 with padding as “alternative port scan” and to port 443 without padding as “no padding scan”. Our results contain eleven scans with the default configuration, one no padding

scan on 2021-03-09 and one alternative port scan on 2021-04-24. In total, we analyze ten scans of IPv4 and three scans of IPv6.

In addition, Table 3.1 depicts the packet rate for each scan. We used a packet rate of 5000 to 15 000 packets per second in IPv4 scans and a rate of 2000 to 8000 packets per second in IPv6 scans.

As input for the IPv4 stateless scans with the default and alternative port configuration, we use targets generated locally from the Border Gateway Protocol (BGP). Therefore, the total number of targets varies between 2.88B and 3.02B for IPv4. As input for the no padding scan, we use the targets with successful responses of the scan on 2021-03-03. The reason is that we only want to test how many QUIC-capable targets respond to an invalid *Initial* packet with a version negotiation.

The total number of targets for IPv6 varies between 4.09M and 21.47M. As input for the first IPv6 scan on 2021-03-12, we used the targets of the IPv6 hitlist [13]. The IPv6 hitlist excludes targets e.g., originating from Cloudflare from its result to remove bias [13]. On 2021-04-16, we used the IPv6 addresses of the DNS scans of our Chair. These DNS scans mostly contain targets originating from Cloudflare as visualized in Section 4.1.4. For the last scan on 2021-04-24, we combined the IPv6 hitlist and the DNS scans to flatten the difference in bias of each scan.

3.3 Stateful Scanner

To collect more details than the supported versions, we implement a stateful scanner from scratch.

3.3.1 Implementation

The stateful scanner is written in Golang and uses forks of the libraries `quic-go`[1] and `qtls`[2]. `quic-go` implements the versions 29, 32 and 34 of the IETF QUIC draft in Go [8]. `qtls` itself is a fork of the official Go TLS library. We alter both libraries to expose more details of the TLS messages and the QUIC session.

The stateful scanner establishes a connection with a target as described in Section 2.4 and stores the QUIC transport parameter, TLS parameter and certificates. In comparison to the stateless scanner, a target of the stateful scanner consists of an IP address

[1] `https://github.com/lucas-clemente/quic-go` (last visited: 02.05.2021)

[2] `https://github.com/marten-seemann/qtls-go1-16/` (last visited: 02.05.2021)

and a port. Additionally, a target may have a hostname. We differentiate two types of stateful scans: SNI and no SNI scans. SNI scans use targets with hostnames while no SNI scans only use targets with an IP address and port. The scanner uses SNI to differentiate between multiple virtual servers on one IP address [7].

Additionally, the stateful scanner terminates the connection to its peer by sending a CONNECTION_CLOSE frame. In comparison to the idle timeout, this termination stops the server from idling and frees its resources and resets its state.

Furthermore, the scanner includes an option to log the encryption and protection keys. When scanning, we can store the QUIC traffic in Packet Capture (PCAP) files. With these keys, tools like Wireshark can decrypt these files which allows us to inpect the individual packets.

As input for stateful scans, we use the latest result of the stateless scanner filtered by the supported versions of `quic-go`. For SNI scans, we extended the IP addresses with their corresponding hostnames from the Domain Name System (DNS) scans of the Chair of Network Architectures and Services. The hostnames result from scans of top level domains (TLDs) like `com`, `net`, `org`, of lists like Alexa top 1 million [16] and the Centralized Zone Data Service (CZDS) [2]. In addition, we added an option to limit the rate of connections establishments by using a token bucket.

The stateful scanner establishes a connection to the targets and stores start and finish time of the connection establishment and TLS handshake, the QUIC transport parameters, TLS parameters and extensions, TLS certificates and possible error messages of TLS or QUIC.

3.3.2 Test Setup

Similar to the stateless scan, we used a local server and the available test servers of the IETF working group [14] to test the stateful scanner. Since some test server are not up to date, we only received 20 successful responses of 23 servers. We failed to connect to two servers due the choice of the Application-Layer Protocol Negotiation (ALPN) in our TLS Client Hello. We analyze this error in Section 4.2.1. The third server responded with error of an incompatible QUIC version. In other words, the server was not capable of using a QUIC version supported by quic-go. We describe this challenge in Section 3.4.

3.3.3 Configurations of Stateful Scans

Table 3.2 shows the configuration of the stateful scan in the wild. In total, half of the scans use SNI and the other half do not use SNI. Of the four scans, two scans use IPv4 targets and the other two scans use IPv6 targets. Each scan uses a token bucket with

Date	IP Version	SNI	Bucket Size	Bucket Refill Dur.
2021-04-18	IPv4	no	500	2 ms
2021-04-22	IPv4	no	500	2 ms
2021-04-18	IPv4	yes	500	2 ms
2021-04-23	IPv4	yes	500	2 ms
2021-04-16	IPv6	no	500	2 ms
2021-04-24	IPv6	no	500	2 ms
2021-04-16	IPv6	yes	500	2 ms
2021-04-24	IPv6	yes	500	2 ms

Table 3.2: Configuration of Stateful Scans

a size of 500 and a refill duration of 2ms. This corresponds to a rate of 500 targets per second.

3.4 Challenges

Since QUIC requires a minimum *Initial* packet size of 1200 bytes [19], the probe module of the stateless scanner generates a higher traffic volume than e.g., the ZMap TCP probe module. To avoid overloading the scanned network or parts of it, we limit the packet rate to a maximum of 15 000 packets per second. With an shared bandwidth of 1 Gbit/s, we are capable of scanning 3 million targets in under 56 hours. As a result, the scanning speed of the QUIC probe module is slower than the maximum possible speed of ZMap but adheres to ethical guidelines (see Section 3.6).

Additionally, we discovered in a local test that a server might not respond to the second of two consecutive request sent in a time frame of around 10 seconds. This is probably due to the client still trying to use the same invalid version after receiving a *Version Negotiation* packet. In comparison, a real client would either try to change its version or avoid sending further packets. The specification defines that a server should respond with a version negotiation even if it is not able to decrypt the payload but is allowed to limit these responses [19]. Therefore, we cannot determine the exact amount of time and requests necessary because it depends on the implementation used by the server. However, we do not expect to encounter this issue since we scan a larger amount of addresses (e.g., the IPv4 address space) without scanning the same IP address twice.

On the other hand, the challenge of the stateful scanner is its support of versions. The stateful scanner has only access to the versions supported by its underlying library quic-go. Therefore, we cannot scan older drafts than draft-29 as well as might not be able

to scan newly released drafts immediately. In addition, the scanner is not compatible with gQUIC. Nevertheless, we identified the most supported version with the collected data from the stateless scans in Section 4.1.5 which is at the time of writing the 29th version of the IETF quic draft.

3.5 Measurement Setup

For all stateless and stateful scans, we use Debian Buster on a system with 40GiB of Random Access Memory (RAM) and an Intel(R) Core(TM) i7 CPU 965 with a clock speed of 3.20GHz. All scans use a 1Gbit/s uplink which is shared with other scans of the Chair of Network Architectures and Services. To map an IP address to its AS, we use the University of Oregon Route Views Archive Project [25] to download the latest AS prefixes announced by the BGP. Then, we use the CIDR Report [4] to map AS numbers to AS names. Additionally, we removed all version negotiation versions `0x?a?a?a?a` from the supported versions in all responses in order to eliminate the uniqueness of version groups.

3.6 Ethical Considerations

Network measurements like active scans on the Internet can be intrusive and interpreted as attack. To avoid complications, we follow this process: First, we test the stateless and stateful scanners locally and with test servers provided by the IETF on Github [14] before using it on other targets on the Internet. We verify, that both scanners act according to the specification [19] and use as few resources of the server as possible e.g., terminating the connection, to release the server-side state and resources. Second, we go through an internal approval process of the Chair of Network Architectures and Services before scanning. Third, we provide a website on the scanning addresses. This website contains information about the measurements of the chair and a way to contact us for questions or to exclude targets or subnets from future scans. Fourth, we use a blocklist to exclude targets from our scans. In addition, we limit the packet rate to avoid overloading parts of a network e.g., routers on the path to endpoints or endpoints themselves.

Chapter 4

Results and Evaluation

We split our results and their evaluation into stateless and stateful scans. We start with the examination and evaluation of the stateless scans. Then, we analyze the results of the stateful scans.

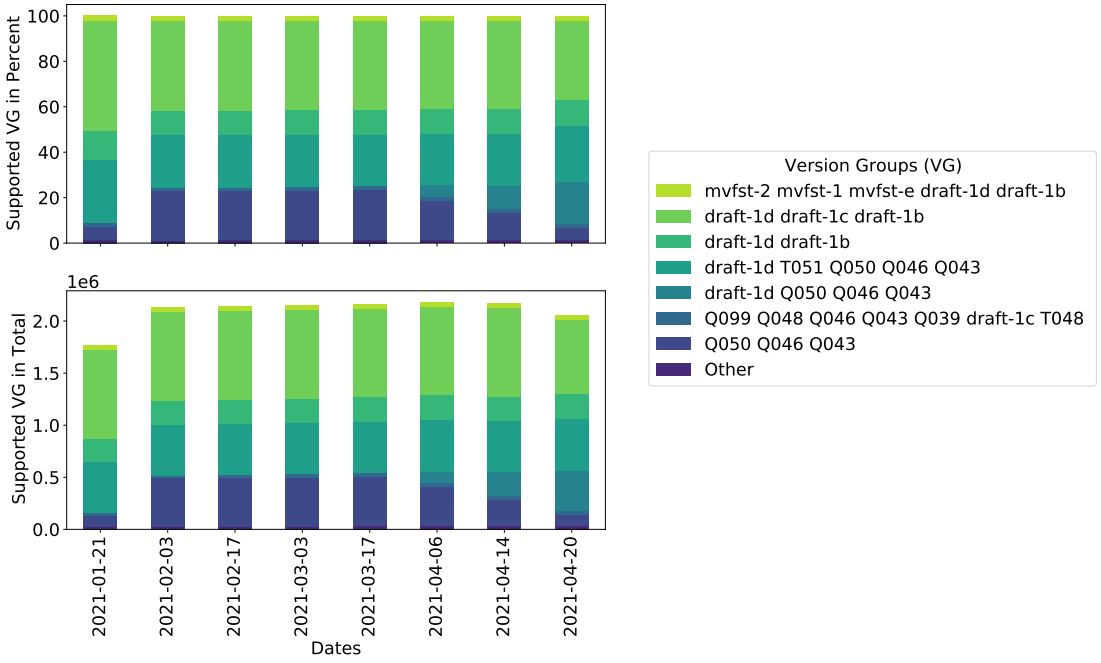
4.1 Stateless Scans

First, we analyze the version groups and ASes for each configuration and IP version. We start with the default configuration scans of IPv4. Next, we describe and evaluate the results of the no padding and the alternative port scans. Then, we examine the default configuration scan of IPv6. After that, we describe the overall version distribution and explicitly the share of IETF QUIC and gQUIC. Last, we summarize our findings.

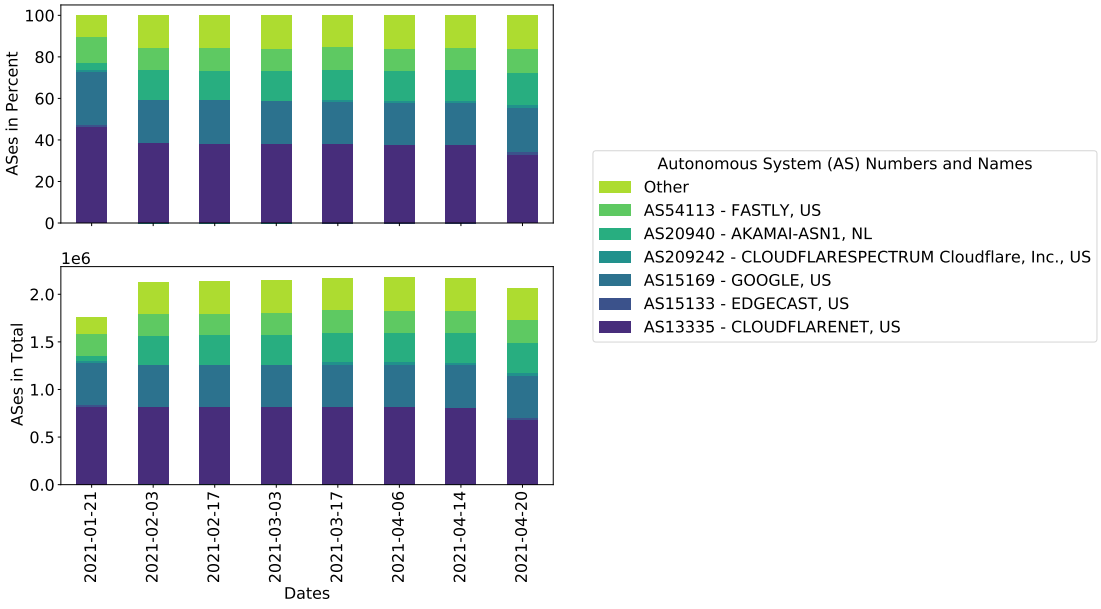
4.1.1 IPv4 Default Configuration

Firstly, we examine the results of the IPv4 scan with the default configuration. The amount of successful responses of each scan in Table 4.1 increases from 2021-01-21 to 2021-04-06 and decreases afterwards. Overall, the number of successful responses is around 2.1M with an exception in January (2021-01-21) where the number is approximately 1.76M. In total, only 0.06% to 0.08% of the scanned IPv4 targets supported QUIC. In comparison, scans of the Chair of Network Architectures and Services indicate a higher usage of TCP+TLS; the TCP+TLS support is around 1.78% across the same amount of IPv4 targets. This is around 20 to 30 times more than QUIC.

Figure 4.1 depicts the distribution of version groups and the distribution of ASes for each scan. The charts at the bottom of Figure 4.1a and 4.1b show the distribution in total numbers, while the charts at the top show the distribution in percent.



(a) Distribution of Version Groups



(b) Distribution of Autonomous Systems

Figure 4.1: Distribution of Version Groups and Autonomous Systems of IPv4 Stateless Scans with Default Configuration

Date	Total Targets	Successful Responses	Share
2021-01-21	2.88B	1.76M	0.06%
2021-02-03	2.90B	2.13M	0.07%
2021-02-17	2.91B	2.14M	0.07%
2021-03-03	2.91B	2.15M	0.07%
2021-03-17	2.91B	2.16M	0.07%
2021-04-06	2.90B	2.18M	0.08%
2021-04-14	2.94B	2.17M	0.07%
2021-04-20	2.97B	2.06M	0.07%

Table 4.1: Total Targets and Successful Responses of the IPv4 Stateless Scans with the Default Configuration

We group version groups with a share smaller than 1.0% into "Other". "Other" contains 47 different version groups such as **picoquic draft-1d**, **draft-1d draft-1e draft-1f draft-20** and **quicgo-ff**. Some version groups indicate the used library e.g., **picoquic**, **quicgo-ff**, **msq-0**, **netapp-22**. Additionally, the "Other" group contains version groups that are unique combinations of multiple IETF draft versions and gQUIC versions regardless of their order.

Based on Figure 4.1a, the increase in total successful responses between 2021-01-21 (1.76M) and 2021-02-03 (2.13M) is caused by the increased usage of the version group **Q050 Q046 Q043** (increase from 54.2k to 306.0k). In combination with Figure 4.1b, we identify that this increase originates from AS20940 (Akamai). As a result, this indicates that either Akamai enabled the QUIC protocol in more of its services or that we could not reach parts of AS20940 on the 2021-01-21. Since we do not have complete scans before 2021-01-21, we cannot determine the cause.

Furthermore, we notice a decrease in successful responses on the 2021-04-20 which is caused by less support for the version group **draft-1d draft-1c draft-1b** (682.6k instead of 813.5k). We identify that this version group originates from and is mostly used by AS13335 (Cloudflare). This decrease only occurs in the last scan (2021-04-20) and should be analyzed in subsequent scans as future work.

In addition to the total increase and decrease in successful responses, Figure 4.1a visualizes a change and adoption in version groups. While the total number of version groups **Q050 Q046 Q043** and **draft-1d Q050 Q046 Q043** combined stays roughly the same over the last three scans (about 314.3k), we notice an increase of version group **draft-1d Q050 Q046 Q043** (to 262.8k) and a decrease of version group **Q050 Q046 Q043** (to 57.8k). We discovered that both version groups originate from AS20940 (Akamai).

This indicates that Akamai is slowly adopting **draft-1d** of the IETF QUIC protocol and that this trend likely continues in future scans.

Besides of these three major changes, the overall distribution remains unchanged. The top ASes are AS13335 (38.3%, Cloudflare), AS15169 (21.2%, Google), AS20940 (13.4%, Akamai) and AS54113 (11.0%, Fastly). This distribution influences the distribution of version groups. The largest version groups are **draft-1d draft-1c draft-1b** (39.8%, mostly AS13335, Cloudflare), **draft-1d T051 Q050 Q046 Q043** (23.6%, mostly AS15169, Google), **Q050 Q046 Q043** (16.3%, mostly AS20940, Akamai), **draft-1d Q050 Q046 Q043** (4.3%, mostly AS20940, Akamai), and **mvfst-2 mvfst-1 mvfst-e draft-1d draft-1b** (1.9%, half from AS32934, Facebook).

In "Other", we discovered version groups that contain nothing but multiple version negotiation versions (`0x?a?a?a?a`) (0.3%). According to the specification, these versions force a version negotiation and cannot be used for a valid connection establishment [19]. The targets with only version negotiation versions mostly originate from AS15169 (93.8%), AS24424 (5.1%) and AS396982 (0.6%) which are all owned by Google.

Additionally, we found the version group **Q050 Q049 Q048 Q046 Q043 draft-1b draft-19 T050 4e303433** (88.9k). This version group contains the version `0x4e303433` which is not listed and reserved on the GitHub page of IETF working group [15]. This version group originates from AS4837 (21.8%), AS9808 (19.2%), AS4134 (10.6%), AS56046 (2.5%) and more whose companies are mainly located in Asia with the majority of them located in China.

Furthermore, "Other" contains seven version groups (1.9k targets) with the first official version of the IETF standard (`0x00000001` or **ietf-01**) as it is documented in the draft version 34 [19]. However, this version is not yet officially released. Over half of theses version groups originate from AS47541 (24.7%), AS47542 (19.5%) and AS28709 (10.5%) which belong to a Russian social media platform called vk.com.

Across all responses, the oldest gQUIC versions are **Q039** and **T048** and the newest are **Q099** and **T051**. The oldest IETF QUIC version is **draft-16** and the newest is **draft-22** or respectively the unreleased version **ietf-01**.

In general, the top ten ASes are responsible for around 90% of QUIC support as illustrated in Figure 4.2. AS13335 (Cloudflare) is responsible for 33.13% to 46.48% of successful responses. The top three ASes are responsible for 70.28% to 84.48% of the results. In total, the results contain 4.30k to 4.69k ASes.

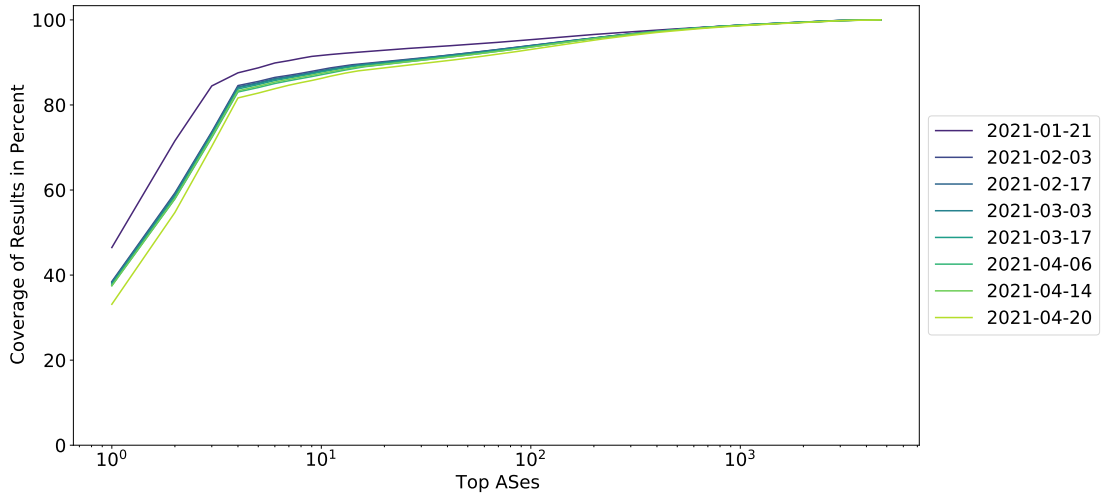


Figure 4.2: Cumulative Percentage of ASes Ranked by Most Frequently Used ASes

4.1.2 IPv4 No Padding

Next, we analyze the results of the no padding scan. The no padding scan on 2021-03-09 has a smaller amount of targets (2.15M) because it uses the successful responses of the previous IPv4 default configuration scan as input. 11.3% or 242.8k targets responded to an *Initial* packet without padding. The QUIC specification defines that *Initial* packets with a size of less than 1200 bytes should be dropped [19]. The specification also specifies that a server should process the version of an *Initial* packet first [19]. Although not intended, libraries could possibly check the version of an *Initial* packet before its length and may respond to invalid *Initial* packet.

231.6k targets use the version group **draft-1d draft-1b**. This is a share of over 95.4% of the total number of responses and originates from AS54113 which is owned by Fastly.

Similar to the default configuration scan results, the results contain version groups with only version negotiation versions. Around 9.1k targets responded with this type of version group; all of which originate from AS15169 (Google).

Other version groups include e.g., the versions of the IETF draft, **mvfst-1**, **picoquic** and **netapp-22** which combined leads to a total share of 0.59% of successful responses. In total, the majority of responses originate of AS54113 (95.4%, Fastly), AS15169 (3.8%, Google) and AS16509 (0.3%, Amazon).

Furthermore, we tested if it is possible to establish a connection with these servers using an *Initial* packet without padding. We chose 9 targets at random from the top three ASes (3 from Fastly, 3 from Google, 3 from Amazon). We checked their availability by sending an *Initial* packet with a version negotiation version and no padding. After we

verified that these servers were reachable and still responded to the no padding scan, we created an *Initial* packet with valid content (version **draft-1d** and TLS Client Hello) but without padding and tried to establish a connection. However, we received no response to these packets from all servers. Then, we tried to establish a connection with valid packets including padding, but we only got responses from Fastly.

To summarize, no server establishes a connection with *Initial* packets without padding, some servers (Google, Amazon) only responded to forced version negotiations and Fastly server responded to forced version negotiations and valid *Initial* packets. This indicates a faulty configuration of the tested servers of Google and Amazon and a invalid implementation or configuration of QUIC used by the servers of Fastly.

We tried to reproduce this behavior of Fastly and used their open source library `quicly` [28] to created a server. We tested the server by sending the same packets. However, we could not even reproduce the response to a forced version negotiation with no padding.

4.1.3 IPv4 Alternative Port

Additionally to the scans on port 443, we scanned targets with the alternative port 8443 on 2021-04-24. Although the number of total targets is similar to the scans with the default configurations (3.02B), this scan has fewer successful responses with a number of only 703.3k. As a result, only 0.02% of the scanned IPv4 targets instead of 0.06% to 0.08% (default configuration) support QUIC on port 8443 .

99.98% (703.1k) of the scanned targets used the version group **draft-1d draft-1c draft-1b**. Other version groups are mostly permutations of the gQUIC and IETF versions and have a share of combined 0.02%. The majority of the responding targets originates from AS13335 (96.0%) and AS209242(3.3%) which are both owned by Cloudflare.

Although we drop invalid incoming responses as explained in Section 3.2.1, one target responded with a valid *Version Negotiation* header and the invalid version group "2021-04-26T20:17:08.223+0200". This target seems to mimic the QUIC version negotiation but responds with a timestamp. It originates from AS20473 which is owned by the data center company Constant.

4.1.4 IPv6 Default Configuration

Next, we analyze IPv6 scans with the default configuration. Table 4.2 shows that the successful responses increase from 76.7k to 210.0k. However in combination with the varying amount of scanned targets, we can neither deduce a trend nor a correlation. However, the results on 2021-04-24 indicate that the QUIC-capable targets of the DNS scan (2021-04-16) and the IPv6 hitlist (2021-03-12) are mutually exclusive.

Date	Total Targets	Successful Responses	Share
2021-03-12	12.04M	76.7k	0.64%
2021-04-16	4.09M	133.5k	3.26%
2021-04-24	21.47M	210.0k	0.98%

Table 4.2: Total Targets and Successful Responses of the IPv6 Stateless Scans with the Default Configuration

To compare QUIC with TCP+TLS, we use the scans of the Chair of Network Architectures and Services. These scans use the same target group as our IPv6 stateless scan on 2021-04-16. Across these targets, TCP+TLS has a coverage of around 73.3% which is around 22 times higher than the QUIC coverage of 3.26%.

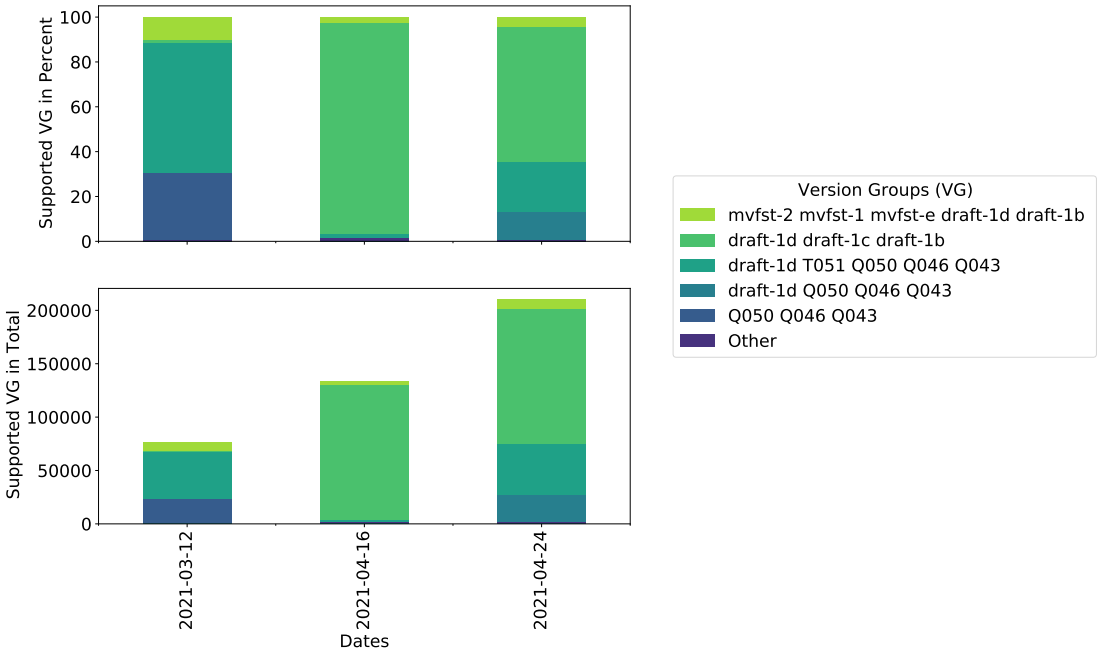
Figure 4.3 shows the distribution of version groups and ASes of the default configuration scan of IPv6. As in Table 4.2, we notice an increase in successful responses in Figure 4.3 over time but cannot conclude a trend due to the varying targets used for the scans.

In comparison to IPv4, we identify a similar distribution of ASes and version groups for the last two scans in Figure 4.3b. The largest AS is AS13335 (Cloudflare), followed by AS15169 (Google) and AS20940 (Akamai) which is exactly the same order as in IPv4 scans. As a result, the version group distribution of the last two scans in Figure 4.3a is also similar to IPv4 with the most widely used version group being **draft-1d draft-1c draft-1b**.

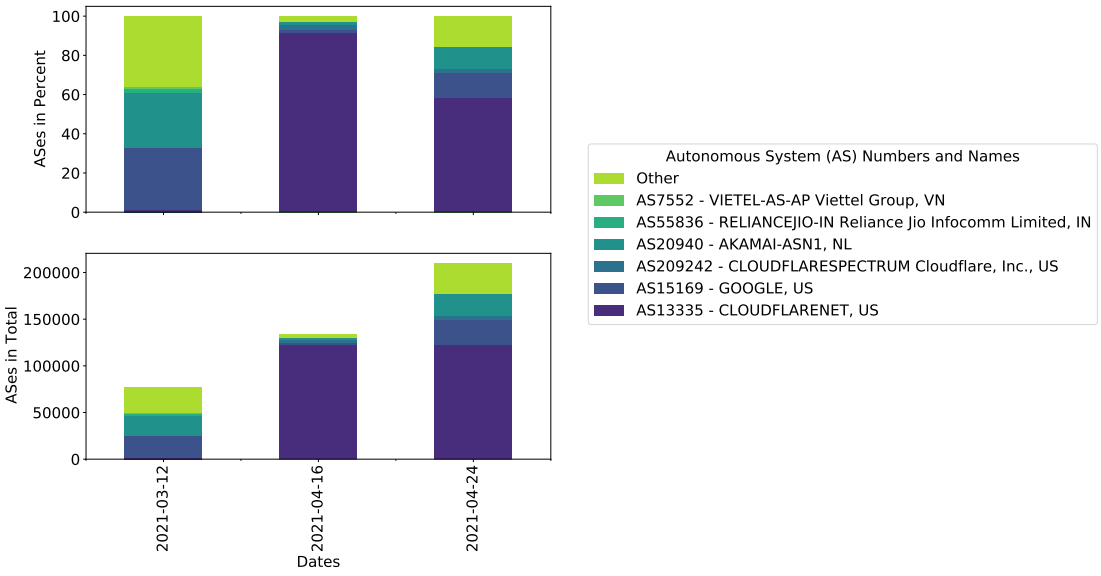
On the other hand, the first scan of Figure 4.3a on 2021-03-12 mostly consists of the version groups **draft-1d T051 Q050 Q046 Q043** (57.9%), **Q050 Q046 Q043** (30.1%), **mvfst-2 mvfst-1 mvfst-e draft-1d draft-1b** (9.9%) and **draft-1d draft-1c draft-1b** (1.5%). Although these version groups are similar to IPv4, we notice the smaller share of version group **draft-1d draft-1c draft-1b** (39.8% for IPv4) caused by the lack of targets originating from Cloudflare. In terms of ASes, we notice the same tendency as in IPv4 but without the dominant share of Cloudflare. The majority of responses of the scan on 2021-03-12 originate from AS15169 (31.7%, Google), AS20940 (28.0%, Akamai), AS55836 (1.8%, Reliance Jio Infocomm), AS7552 (1.3%, Viettel Group) and AS13335 (1.2%, Cloudflare).

4.1.5 Overall Version Distribution

To analyze the adoption of the IETF QUIC draft, we divided the version groups into IETF QUIC and gQUIC versions. For example, if one target supports multiple versions of IETF QUIC, we only count this target once. Since targets may support both versions



(a) Distribution of Version Groups



(b) Distribution of Autonomous Systems

Figure 4.3: Distribution of Version Groups and Autonomous Systems of IPv6 Stateless Scans with Default Configuration

Date	Scan Type	IP Version	IETF Support	gQUIC Support
2021-01-21	default	IPv4	93.72%	36.22%
2021-02-03	default	IPv4	77.89%	47.04%
2021-02-17	default	IPv4	77.80%	47.09%
2021-03-03	default	IPv4	77.76%	47.23%
2021-03-17	default	IPv4	77.61%	47.39%
2021-04-06	default	IPv4	82.40%	47.66%
2021-04-14	default	IPv4	88.06%	47.54%
2021-04-20	default	IPv4	94.52%	51.24%
2021-03-12	default	IPv6	69.83%	88.32%
2021-04-16	default	IPv6	99.42%	2.89%
2021-04-24	default	IPv6	99.93%	35.31%
2021-03-09	no padding	IPv4	96.01%	0.00%
2021-04-24	alternative port	IPv4	99.99%	0.00%

Table 4.3: IETF QUIC and gQUIC Support in Stateless Scans

at the same time, the percentages are not mutually exclusive and can add up to more than 100% in Table 4.3.

Table 4.3 shows the shares of the IETF QUIC and gQUIC versions. We notice a disparity between the IPv4 default configuration scans and other scans. For example, both the no padding and alternative port scan only support a negligible share of gQUIC while the shares of gQUIC in IPv6 scans vary from 2.89% to 88.32% (mainly due to the Cloudflare share). For this reason, we focus on the IPv4 default configuration scan.

In each scan, between 77.61% and 94.52% of targets support the IETF QUIC protocol. In comparison, the support of gQUIC is lower with a share of 36.22% to 51.24% per scan.

Over the time of the analysis, we notice a decrease in the support of IETF QUIC from 2021-01-21 (93.72%) to 2021-03-17 (77.61%) which is due the introduction of the version group **Q050 Q046 Q043** in AS20940 (Akamai). This decrease is followed by an increase from 2021-03-17 (77.61%) to 2021-04-20 (94.52%). This major change is due to the adoption of version **draft-1d** in version group **draft-1d Q050 Q046 Q043** also originating from AS20940 (Akamai). This indicates an increased adoption of IETF QUIC. Nevertheless, this increased adoption does not influence the share of gQUIC. The results show an overall increase in gQUIC support with a minor deviation of -0.12% on 2021-04-14.

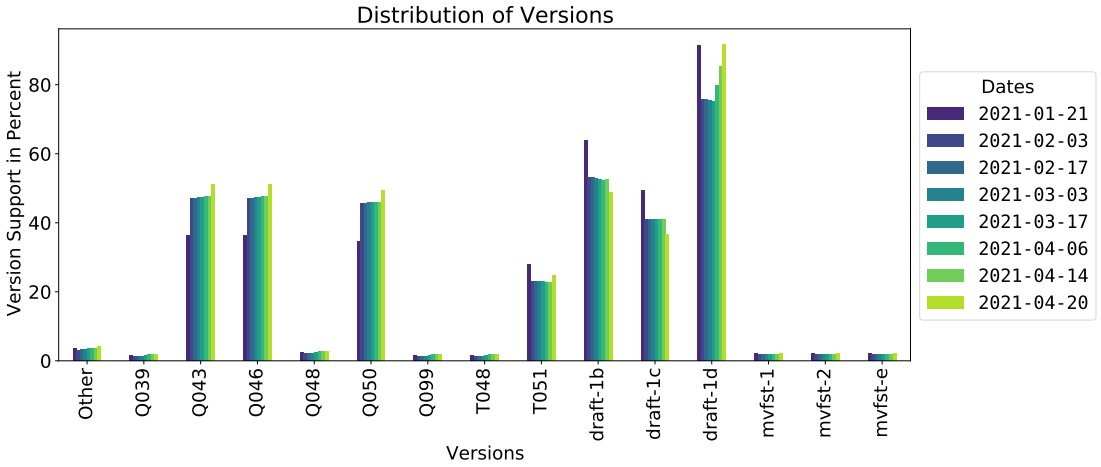


Figure 4.4: Supported Versions of IPv4 Stateless Scan with Default Configuration

Figure 4.4 depicts the individual versions in percent for the IPv4 scan with default configuration. The most supported version is **draft-1d**, followed by **draft-1b**, **Q043**, **Q046**, **Q050**. Next is the version **draft-1c**, followed by **T051**. We notice a trend towards fewer support for versions of **draft-1c** and **draft-1b** and an increased support for **draft-1d**. The increase of gQUIC support is visible across the versions **Q039**, **Q043**, **Q046**, **Q048**, **Q050** and **Q099** as well as for version **T048**. Apart from the spike on 2021-04-20, we identify an overall decrease of support for version **T051**.

4.1.6 Summary

The stateless scans show that AS13335 (40.46%, Cloudflare), AS15169 (19.96%, Google), AS20940 (12.64%, Akamai) and AS54113 (11.53%, Fastly) prevail among the all successfully scanned targets. The most supported version groups are **draft-1d draft-1c draft-1b** (42.07%, mostly Cloudflare), **draft-1d T051 Q050 Q046 Q043** (22.37%, mostly Google and Akamai), **Q050 Q046 Q043** (15.18%, mostly Akamai) and **draft-1d draft-1b** (11.54%, mostly Fastly). Therefore, the scan results of the stateless scanner show a strong correlation between version groups and ASes.

Our results indicate that the IETF QUIC is more widely supported than gQUIC; especially, in IPv6, no padding or alternative port scans. Nevertheless, the support of both protocols mostly increases over the time of our measurements. In addition, we also notice an adoption of IETF QUIC (**draft-1d**) by AS20940 (Akamai). The version **draft-1d** is the most supported version of QUIC-capable targets.

Our scans contain responses which only include version negotiation versions and responses with versions that are either not reserved (`0x4e303433`) or not officially released

Date	IP Version	SNI	Total Targets	Successful Responses	Share
2021-04-18	IPv4	no	1.85M	146.2k	7.88%
2021-04-22	IPv4	no	1.89M	147.0k	7.79%
2021-04-16	IPv6	no	132.6k	7.5k	5.73%
2021-04-24	IPv6	no	208.5k	57.9k	27.76%
2021-04-18	IPv4	yes	41.32M	37.81M	91.51%
2021-04-23	IPv4	yes	41.55M	38.01M	91.49%
2021-04-16	IPv6	yes	5.94M	5.26M	88.55%
2021-04-24	IPv6	yes	5.96M	5.26M	88.31%

Table 4.4: Total Number of Targets and Number of Successful Responses of the Stateful Scans

yet (`0x00000001`). Furthermore, we discover that 11.3% of QUIC-capable servers in IPv4 respond to forced version negotiations without padding. However, we cannot establish a connection to these servers with *Initial* packets without padding. In addition, our results show that the majority of IPv4 traffic on port 8443 originates from AS13335 (Cloudflare).

In general, we determine that the QUIC support is growing but that the coverage of TCP+TLS is 20 to 30 times higher in IPv4 and IPv6.

4.2 Stateful Scans

Table 4.4 depicts the total number of targets and successful responses of each stateful scan. Due to multiple hostnames per IP address, we scanned a higher number of targets with SNI than without SNI. In both SNI and no SNI scans, we scanned more targets in IPv4 than IPv6. Nevertheless, we notice a significant difference in successful responses for SNI with 88.31% to 91.51% in comparison to no SNI with 5.72% to 27.76%.

We exclude the results from the no SNI scan on 2021-04-08, because we encountered a limit of file descriptors that we were allowed to open on our operation system. This lead to fewer responses than we would have actually captured.

We use the QUIC-capable servers of the stateless scanner which supported the versions of our library as input for our stateful scanner, which tries to establish connections with each of them. Since there are many points of failure in a complete connection establishment, we capture the error messages if a connection establishment was not successful. We refer to responses without error message as successful responses and to responses with error messages to unsuccessful responses.

For the no SNI scan, we examined each target which consists of an IP address and a port. However, we cannot use the same approach for SNI scans since we scanned targets with the same IP address multiple times with different hostnames. This would lead to a bias towards IP addresses with a larger amount of hostnames. Therefore, we only used IP addresses once if at least one hostname resulted in a successful or unsuccessful response. In other words, the following results use a set of unique IP addresses for SNI and no SNI if not stated otherwise. We refer to this dataset as filtered targets and to all targets with bias towards the hostnames as unfiltered targets.

First, we analyze the type of responses and their corresponding ASes. Then, we focus on the QUIC transport parameters of the successful responses. After that, we evaluate the results of the QUIC transport parameters, TLS versions and TLS cipher suites.

4.2.1 Responses

First, we start with the analysis of our responses. Before analyzing the AS distribution of successful responses, we examine responses which include an error message. We group ASes with a share of less than 1.5% into "Other".

Unsuccessful Responses and Error Messages

The most common errors of unfiltered targets are:

- CRYPTO_ERROR (0x128): TLS handshake failure 40 (72.99%)
- timeouts (15.41%)
- no compatible QUIC versions found (we support [draft-29], server offered [0x51303530 gQUIC 46 gQUIC 43]) (10.95%)

The QUIC specification defines CRYPTO_ERROR with the code 0x128 as a generic CRYPTO_ERROR for handshake failures. Additionally, TLS handshake failure 40 "indicates that the sender was unable to negotiate an acceptable set of security parameters given the options available" [29]. This error message is generic and does not specify which security parameters are the issue. Therefore, we picked two servers with this error message at random and tried customizing all available TLS parameters, changing the order of the ALPN and using SNI. We did not receive any successful response from the servers. However, we cannot make any general conclusions since our sample group only consisted of two targets.

The same CRYPTO_ERROR occured twice in our test setup as explained in Section 3.3.2. There, we were able to resolve the error by reordering the values of our

Date	IP Version	SNI	Total Errors
2021-04-18	IPv4	no	1.70M
2021-04-22	IPv4	no	1.74M
2021-04-16	IPv6	no	125.1k
2021-04-24	IPv6	no	150.7k
2021-04-18	IPv4	yes	229.3k
2021-04-23	IPv4	yes	227.3k
2021-04-16	IPv6	yes	106.6k
2021-04-24	IPv6	yes	107.4k

Table 4.5: Total Number of Unsuccessful Responses of the Stateful Scans with Filtered Targets

ALPN. We discovered that these test servers only checked the first value of the ALPN which indicates a faulty configuration or incorrect implementation of QUIC.

Among the unfiltered targets, CRYPTO_ERRORs mainly originate from AS13335 (90.77%, Cloudflare), AS15169 (7.16%, Google), AS209242 (0.81%, Cloudflare) and AS20940 (0.60%, Akamai).

The error messages containing timeouts indicate either an unreachable target (No recent network activity) or a reachable target which does not respond to QUIC in a timely manner (Handshake did not complete in time). The latter only occured five times in total. All other timeouts are caused by an unreachable target. Most unreachable targets of the unfiltered targets originate from AS54113 (63.22%, Fastly), AS20940 (23.84%, Akamai) and AS13335 (2.50%, Cloudflare).

Although we filter the input file according to the supported versions of QUIC, we receive the incompatible QUIC versions error. We tested five servers with this error messages with our stateless scan and discovered that all servers responded with the version group **draft-1d Q043 Q046 Q050 T051**. However, when we tried to establish a connection, all servers responded with a *Version Negotiation* packet including the version group **Q043 Q046 Q050**. These results were reproducible for results of the SNI and no SNI scan. Most servers of the unfiltered targets that responded with this error message originated from AS15169 (96.94%) and AS396982 (3.03%) both owned by Google. In our stateless scan, 563.0k IP addresses support version group **draft-1d Q043 Q046 Q050 T051**. Out of these addresses, 182.4k addresses (32.40%) incorrectly claim to support this version.

Next, we explain the shares of unsuccessful responses and their AS origins of the filtered targets. We differentiate between IPv4 and IPv6 scans (no SNI and SNI combined), and between SNI and no SNI scans (IPv4 and IPv6 combined).

Table 4.5 shows the total amount of unsuccessful responses of the filtered targets. As in Table 4.4, we notice a significant difference in the amount of unsuccessful targets for IPv4 without SNI (1.70M - 1.74M) in comparison to IPv4 with SNI (227.3k - 229.3k). However, we notice a smaller difference (18.5k - 43.3k) with SNI in regards to IPv6.

Figure 4.5a shows the distribution (accumulated over dates) of unsuccessful responses without the use of SNI in IPv4 (3.45M targets) and IPv6 (275.8k targets). The majority of unsuccessful targets originate from AS13335 (Cloudflare) for both IPv4 (43.03%) and IPv6 (86.10%) with a more dominant share in IPv6. Additionally, we receive more error messages from AS15169 (Google) and AS54113 (Fastly) in IPv4 (22.80%, 13.54%) than in IPv6 (1.28%, 0.26%). The share of error messages originating from AS20940 (Akamai) is similar for both IP protocols (12.22% of IPv4, 9.00% of IPv6).

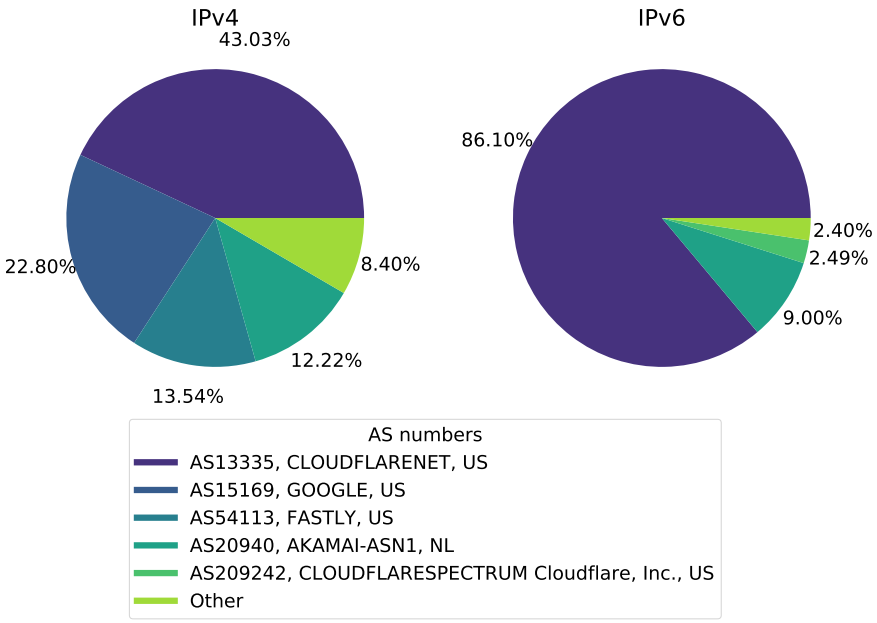
Figure 4.5b depicts the distribution (accumulated over dates) of unsuccessful responses with the use of SNI in IPv4 (456.6k targets) and IPv6 (214.0k targets). As in the no SNI results, the majority of unsuccessful targets originate from AS13335 (Cloudflare) for both IPv4 (56.87%) and IPv6 (96.96%) also with a more dominant share in IPv6. However, both shares are larger than in no SNI scans. In contrast to IPv6, 40.65% of unsuccessful targets originate from AS15169 (Google) in IPv4.

Successful Responses

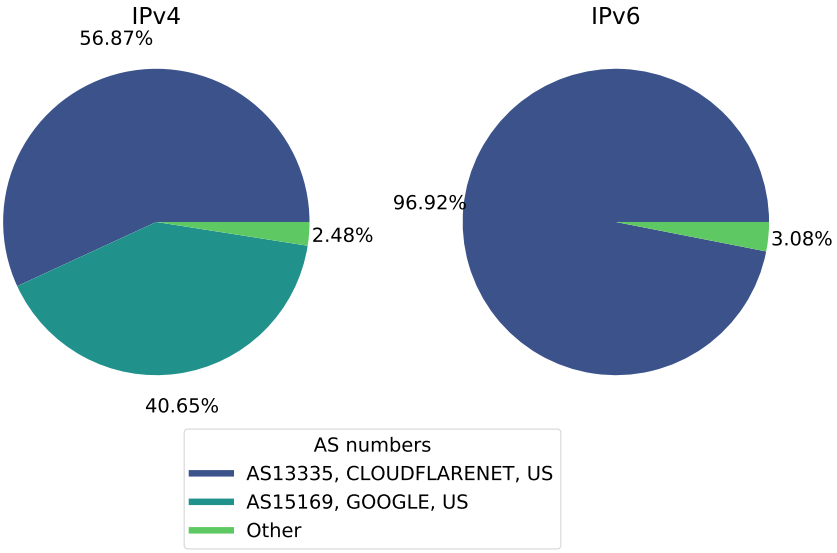
Next, we analyze the successful responses of SNI and no SNI scans and their AS distribution for filtered targets.

Figure 4.6a shows the distribution of ASes of SNI and no SNI scans in IPv4. The responses of the no SNI scans mainly originate from AS15169 (31.17%, Google), AS13335 (4.40%, Cloudflare) and AS12824 (3.45%, home.pl). The remaining share of the SNI scan consists of ASes in "Other" (60.99%). "Other" contains for example targets originating from AS55836 (3.36k, Reliance Jio Infocomm Limited) or AS32934 (3.19k, Facebook). In total, "Other" contains around 4.3k distinct ASes. In IPv4 scans with SNI, the successful responses mostly originate from Cloudflare with AS13335 (84.50%) and AS209242 (4.54%), followed by AS12824 (3.79%, home.pl) and AS15169 (1.70%, Google).

Figure 4.6b depicts a similar distribution of no SNI and SNI scans in IPv6 as in IPv4. However, fewer targets without SNI originate from "Other" (45.03% instead of 60.99%) and more targets originate from AS13335 (11.43% instead of 4.40%, Cloudflare). The share of AS15169 (Google) in IPv6 scans without SNI is larger with 39.37% instead of 31.17%. SNI scans of IPv6 have a larger share of AS13335 (91.94%, Cloudflare) than in IPv4 (84.50%).

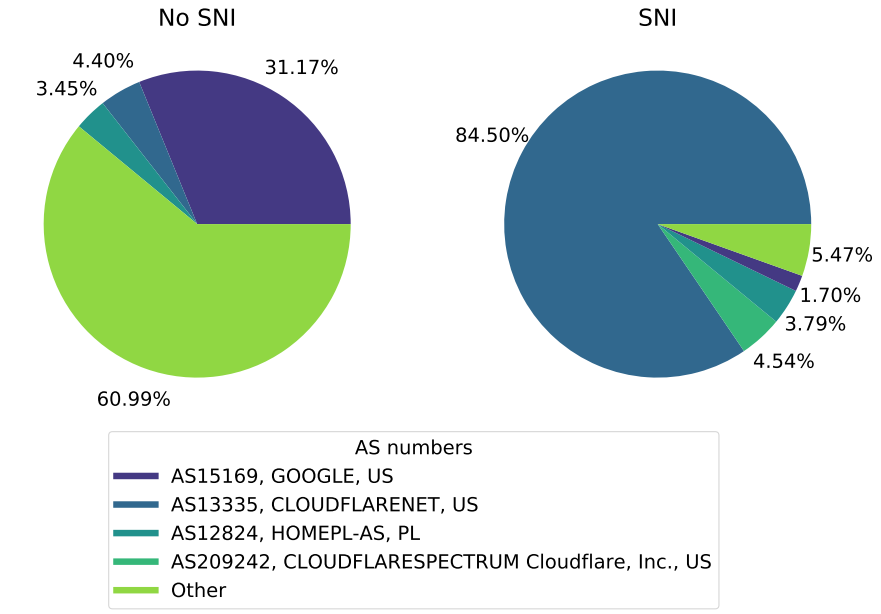


(a) No SNI

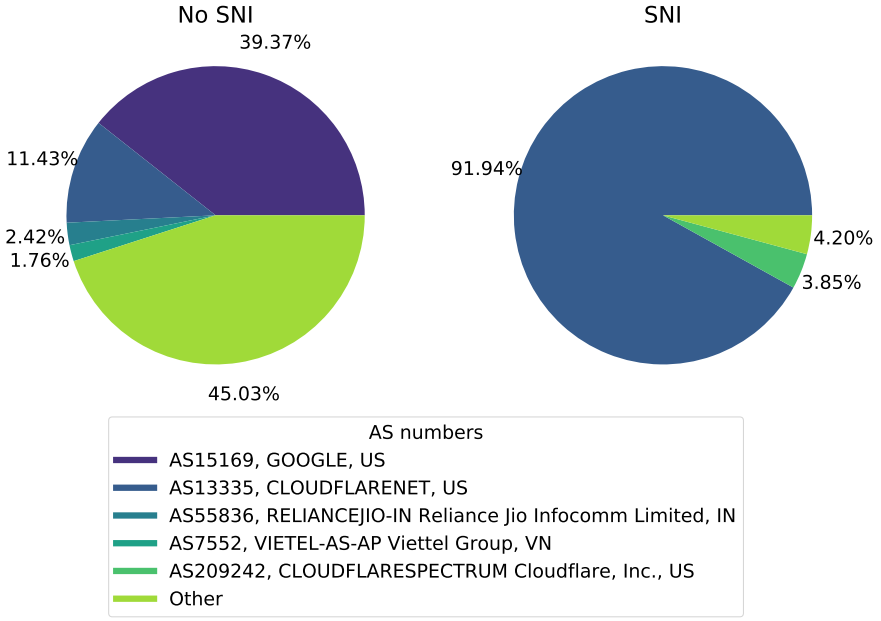


(b) SNI

Figure 4.5: IPv4 and IPv6 AS Distribution of Unsuccessful Filtered Targets of Stateful Scans



(a) IPv4



(b) IPv6

Figure 4.6: No SNI and SNI AS Distribution of Successful Filtered Targets of Stateful Scans

4.2.2 QUIC Transport Parameters

In addition to the successful and unsuccessful responses, we captured QUIC transport parameters from successful responses of our filtered targets. All parameters are set by the server and extracted from the TLS Server Hello message. Some of the QUIC transport parameters are chosen at random for every new connection establishment like the destination connection ID. For this reason, we filter for session specific QUIC transport parameters and only used the following parameters for fingerprinting:

- max_udp_payload_size
- initial_max_data
- initial_max_stream_data_bidi_local
- initial_max_stream_data_bidi_remote
- initial_max_stream_data_uni
- initial_max_streams_bidi
- initial_max_streams_uni
- ack_delay_exponent
- max_ack_delay
- disable_active_migration
- active_conn_id_limit

The purpose of these parameters is explained in the IETF QUIC specification [19]. For each target, we concatenated the parameters in the order of the list above and hashed them to use the hash as identifier for the configurations. Table A.1 depicts the hashes and the parameters in order of the preceding list. In total, we found 38 unique hashes over IPv4, IPv6, no SNI and SNI combined.

Table 4.6 shows the number of distinct configurations available for each stateful scan. We notice a weak correlation between the amount of targets scanned and the amount of unique configuration. In general, IPv4 scans resulted in more unique configurations than IPv6 scans. 22 configurations are used in both SNI and no SNI scans. Only a few are specific to SNI such as:

- `ca0608af548c52ce538ebcfe7a8291ed3c2ef3200a382cb065234a316f78803d`

The top configurations are used by 389.0k (51.26%), 121.8k (16.06%), 105.8k (13.95%) and 65.8k (8.67%) of filtered targets. Most of these targets originate from AS13335

Date	IP Version	SNI	Configurations	Entropy	Low Entropy Configurations
2021-04-18	IPv4	no	32	1.20	18
2021-04-22	IPv4	no	32	1.20	18
2021-04-16	IPv6	no	15	0.41	8
2021-04-24	IPv6	no	16	1.27	7
2021-04-18	IPv4	yes	29	0.61	16
2021-04-23	IPv4	yes	28	0.61	15
2021-04-16	IPv6	yes	20	0.46	13
2021-04-24	IPv6	yes	21	0.47	13

Table 4.6: Total Number of Unique Configurations and Entropy of the Stateful Scans

(371.8k, 49.00%, Cloudflare), AS15169 (121.8k, 16.06%, Google), AS12824 (18.5k, 2.44%, home.pl), and AS209242 (17.0k, 2.24%, Cloudflare).

To get the informativeness of the results on each date, we calculated the Shannon Entropy [23] over the probability of the configurations. Table 4.6 shows that the entropy is higher for IPv4 no SNI scans (1.20) than for IPv4 SNI scans (0.61). This means that the configurations of no SNI are less predictable than the configurations of SNI scans. In addition, we notice a drop in entropy on the first no SNI scan of IPv6 on 2021-04-16.

For IPv4 SNI scans, the top three configurations are:

1. `f8190cf2df70d8456154e7685ea4b00ec279f695558b8b43a10d43bd1a2d9f02`
2. `3db4ec1a7142968f7d8a6982ef10e63bf6eadb01b89167b28702c0711b53daaf`
3. `e1419b37eba377c0614e576146acaac0e6d42a1e1091d5fde707cb7854ebda56`

To improve readability, we now refer to the top three configurations as *alpha*, *beta* and *gamma*. Most targets that use *alpha* originate from AS13335 (94.88%, Cloudflare) and AS209242 (5.10%, Cloudflare). *beta* targets originate from AS12824 (88.89%, home.pl) and AS29222 (1.54%, Infomaniak Network). Targets which use *gamma* are distributed more evenly and the largest percentage of targets originate from AS32934 (6.87%, Facebook).

The results from the top three configurations show that each has a different expressiveness and information content. Therefore, we grouped the configurations and ASes and calculated the Shannon Entropy [23] for each group.

Figure 4.7 shows number of ASes and the entropy of each distinct configuration for IPv4 SNI scans. The entropy of each AS distribution shows what we have shown so far: The

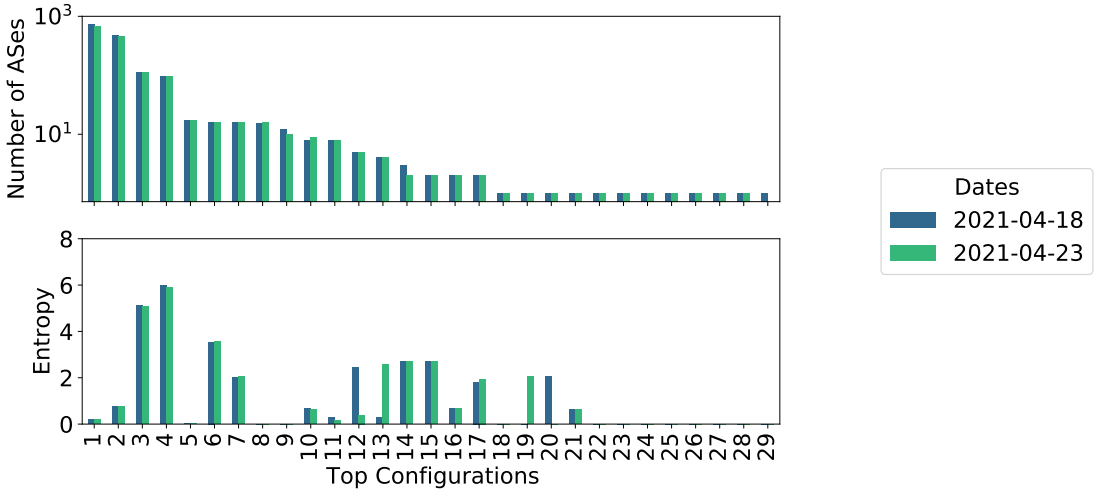


Figure 4.7: Entropy and Number of ASes per Configuration of IPv4 SNI Scans

first two out of the top three configurations (*alpha* and *beta*) have both a relatively low entropy (0.20 and 0.77) compared to *gamma* (5.09) which is mostly due to the diversity of ASes.

Compared with IPv4 SNI scans, Figure 4.8 visualizes a similar pattern of high and low entropy for some configurations of the IPv4 no SNI scans.

We notice similarities in entropy distribution (high and low entropy configurations) when comparing Figures 4.7 and 4.8 (IPv4) and Figures 4.9 and 4.10 (IPv6).

Table 4.6 lists the number of configurations with low entropy (less than 0.5) for each scan. Low entropy means that we have a higher chance to find the AS number of a target with only the configuration given. Therefore, we know that over half of the configurations could be used in theory to fingerprint the origins.

In our case, this use is more of a proof of concept since we can identify the target's AS number by using the target's IP address. However, we could calculate the entropy of the configurations combined with their implementations/libraries identified by other data (e.g., HTTP headers) in the future. This could be used to identify the implementation of targets which do not disclose it if the entropy of the combination (configuration and implementation) is low.

4.2.3 TLS Version and Cipher Suites

We analyzed the TLS versions and cipher suites used in each stateful scan. Our results of the unfiltered targets show that all successful responses used TLS version 1.3. According

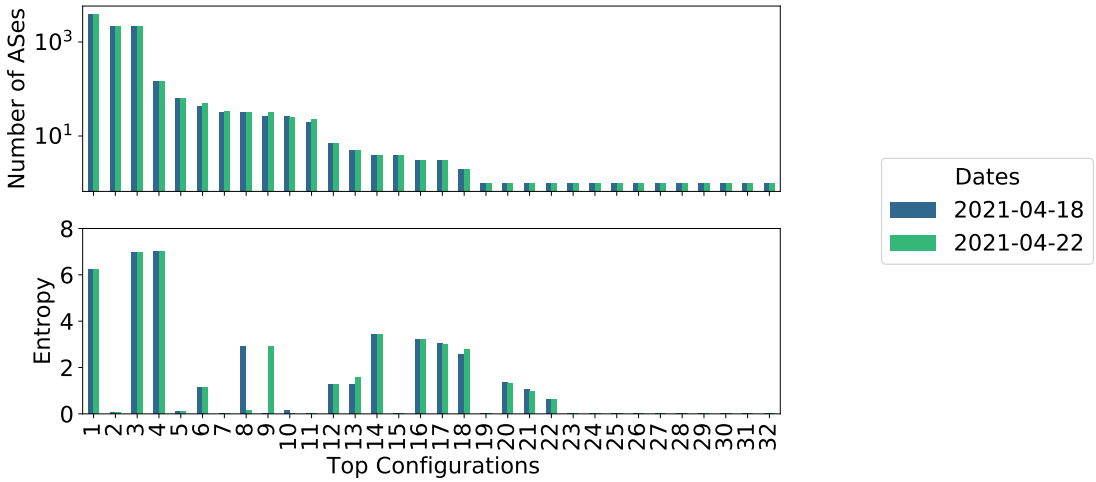


Figure 4.8: Entropy and Number of ASes per Configuration of IPv4 no SNI Scans

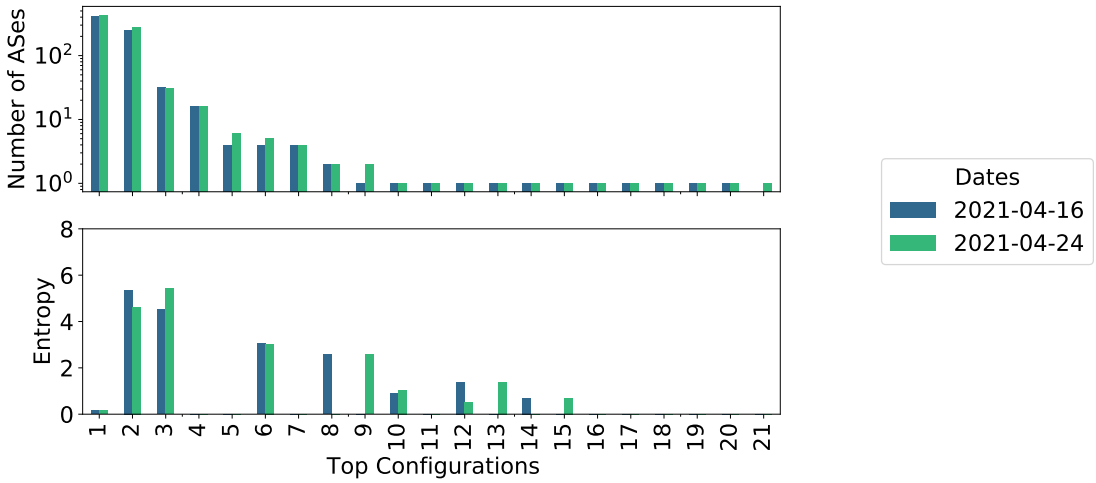


Figure 4.9: Entropy and Number of ASes per Configuration of IPv6 SNI

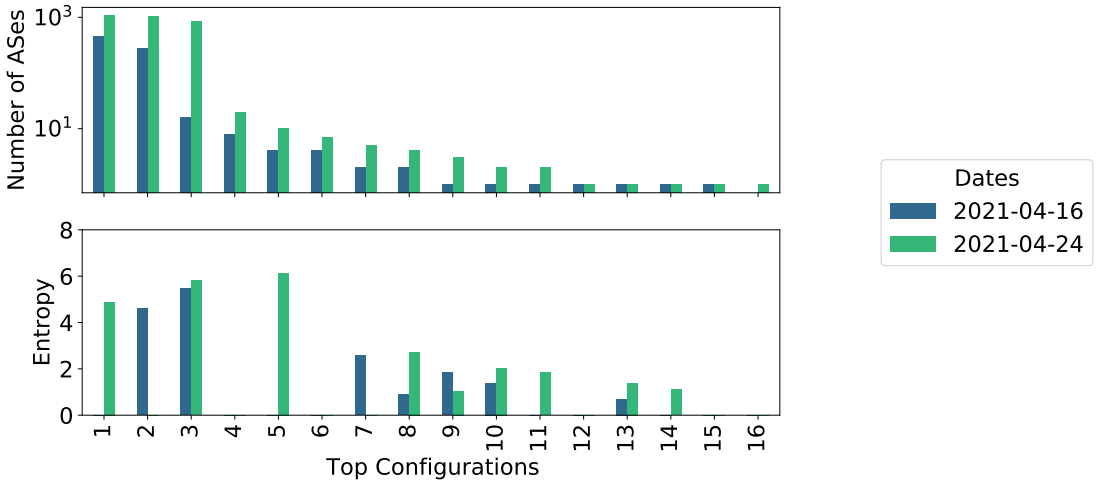


Figure 4.10: Entropy and Number of ASes per Configuration of IPv6 no SNI

			Cipher Suite		
Date	IP Version	SNI	0x1301	0x1302	0x1303
2021-04-18	IPv4	no	99.85%	0.00%	0.15%
2021-04-22	IPv4	no	99.85%	0.00%	0.15%
2021-04-16	IPv6	no	99.99%	0.00%	0.01%
2021-04-24	IPv6	no	99.93%	0.00%	0.07%
2021-04-18	IPv4	yes	100.0%	0.00%	0.00%
2021-04-23	IPv4	yes	100.0%	0.00%	0.00%
2021-04-16	IPv6	yes	100.0%	0.00%	0.00%
2021-04-24	IPv6	yes	100.0%	0.00%	0.00%

Table 4.7: Distribution of Supported Cipher Suites in Stateful Scans

to the QUIC specification, all endpoints should terminate the connection if a version lower than TLS 1.3 is used [33].

TLS 1.3 only supports the use of five cipher suites [29]:

- `TLS_AES_128_GCM_SHA256` (0x1301)
- `TLS_AES_256_GCM_SHA384` (0x1302)
- `TLS_CHACHA20_POLY1305_SHA256` (0x1303)
- `TLS_AES_128_CCM_SHA256` (0x1304)
- `TLS_AES_128_CCM_8_SHA256` (0x1305)

According to the specification of TLS 1.3 [29], an endpoint must support cipher suite 0x1301 and should support cipher suites 0x1302 and 0x1302. According to the specification of QUIC, cipher suite 0x1305 must not be used. The stateful scanner supports cipher suites 0x1301, 0x1302 and 0x1303 and includes them in the TLS Client Hello message of the *Initial* packet. Then, the server select its preferred cipher suite and responds with a TLS Server Hello message.

Table 4.7 depicts the distribution of preferred cipher suites. The scans show that targets mostly select cipher suite 0x1301 (99.85% - 100.0%) and 0x1303 (0.00% - 0.15%). The share of cipher suite 0x1302 is negligibly low (5-143 targets).

4.2.4 Summary

We analyzed successful and unsuccessful connection establishments and found three types of error messages that prevail: TLS handshake failures (mostly AS13335, Cloudflare), timeouts (mostly AS54113, Fastly) and incompatible versions (mostly AS15169, Google).

The last error invalidates parts of the results of the stateless scan. 32.40% of addresses that support version group **draft-1d Q043 Q046 Q050 T051** incorrectly claim to support **draft-1d** and **T051**.

In total, we notice that with filtered targets stateful scans with SNI are less error prone and less divers than without SNI. In terms of unsuccessful responses, SNI leads to a shift towards a higher share of AS13335 (Cloudflare) and AS15169 (Google) in IPv4 and to a more prominent share of only AS13335 (Cloudflare) in IPv6. The use of SNI leads to a higher amount of successful responses from AS13335 (Cloudflare) while scans without SNI are generally more divers in terms of AS origins.

Furthermore, we identified QUIC configurations based on the QUIC transport parameters. Some configurations in combination with the AS had a lower entropy because mostly only one AS used this configuration. These low entropy configurations can be used to predict the AS if only given the configuration or respectively the QUIC parameters. In future work, this could be used in combination with e.g., HTTP headers to fingerprint the implementation of QUIC.

Last but not least, we determined that most targets preferred TLS cipher suite 0x1301. The share of other cipher suites is negligible.

Chapter 5

Related Work

Next, we put this thesis into the context of similar research of the QUIC transport protocol.

First of all, we need to mention the main influence on this thesis namely the work of Rüth et al. [31]. Rüth et al. use a probe module in ZMap to scan QUIC-capable server and supported version. In addition, they use a stateful scanner to capture and analyze TLS certificates. They focus on gQUIC (since QUIC was still developed by Google at that time) and identify that mainly Google and Akamai support QUIC. We use the same but updated (IETF QUIC compatibility, no padding option) ZMap probe module. We find that the origins of QUIC traffic shifted to mostly Cloudflare followed by Google and Akamai. In addition, we focus on the QUIC transport parameter and their potential use as fingerprint.

In contrast to measurements on the Internet, Gagliardi et al. [12] test implementations of IETF QUIC. They focus on version 23 of the draft (which was at that time the latest version). Gagliardi et al. send modified *Initial* packets to 10 of the public test servers of QUIC [14] and analyze their responses. These modifications consist of increasing and decreasing the size of a packet, removing QUIC transport parameters or ALPN, and setting the version to a future version. Gagliardi et al. determine that two server respond to the smaller (invalid) *Initial* packet and some server only respond to the request with missing QUIC parameters (especially ALPN). In our thesis, we also discover that ALPN can lead to an unsuccessful connection establishment. Additionally, we use a similar approach to generate no padding packets but we extend the measurements to reachable IPv4 and IPv6 targets and test it on all known QUIC-capable servers on the Internet.

In contrast to Gagliardi et al. , we discover that we can force a version negotiation at some servers but no connection establishment.

Similar to Gagliardi et al. , Piraux et al. [27] focus on testing and provide a test suite to check for invalid behavior in implementations. To verify the functionality of their test suite, Piraux et al. use a similar approach as Rüth et al. to identify QUIC-capable server. Then, they try to connect to these servers with their test suite. The result is either success, failure or error. Failure is when a server is not QUIC conform and error is return in case of a server error. They visualize an error rate around 10% to 20%, a failure rate of around 20% to 30% and a success rate of around 50% to 70%. Compared to our results, we determine a similar share of error messages for SNI scans (8.49% - 11.69%) but a significantly higher share for no SNI scans (72.24% - 94.27%). Like Piraux et al. , we notice that some server do not adhere to the QUIC specification (ALPN).

Other work related to QUIC is in areas of performance. While performance is not the main focus of our work, we shortly describe the related topic.

Kakhki et al. [21] compare the performance of QUIC and TCP. For this, they test the fairness between the cubic congestion control protocols of both transport protocols. They discover that QUIC shares the bottleneck bandwidth fairly with other QUIC connections but uses a much higher share when used simultaneously with TCP and is therefore unfair to TCP. In addition, they show that QUIC outperforms TCP in page load times for all scenarios but loading a large number of small files. However, Kakhki et al. use gQUIC versions 25 to 34 which are not supported anymore according to our results. Therefore, this behavior may have changed.

Chapter 6

Conclusion and Future Work

Although Google and Akamai were among the first companies to support QUIC, our results show that companies like Fastly have caught up to them and that Cloudflare even surpassed them in the total number of QUIC-capable targets. In addition, we discovered that targets originating from Google sometimes respond with faulty QUIC versions or claim to support more version than they are capable of using.

The share of IETF QUIC is significantly larger than the share of gQUIC with most targets supporting the IETF QUIC version **draft-1d**. In regards to the upcoming official release of the IETF QUIC, we found that version `0x00000001` is already in use, even though it should not be the case yet.

Furthermore, we suggest using SNI for future stateful scans to reduce the amount of error messages. However, this leads to successful responses that are biased towards Cloudflare.

In general, QUIC as alternative to TCP+TLS is widely supported across CDNs like Cloudflare, Akamai and Fastly but in comparison the support of TCP+TLS is 20 times larger in both IPv4 and IPv6.

The results show a trend towards an increasing adoption of the **draft-1d** in ASes from Akamai. This trend should be studied in future work. Moreover, future work may be able to clarify which reason is behind the decrease of targets from Cloudflare in the last scan; for example, if it is or is not a fluctuation in reachability from our side.

We already implemented a basic HTTP HEAD request and capture the response, specifically the HTTP headers with our stateful scanner. In our scans, we did not include the

scanning of these headers, but the analysis of them is a desirable future work and might reveal the underlying implementation of a QUIC-capable target.

The implementation of HTTP headers may also be used in combination with the QUIC configurations to calculate the entropy of this combination. With low entropy combinations, future analysis can predict or fingerprint the implementation even if it is not exposed in the HTTP headers.

Chapter A

Appendix

All Possible Hashes	Parameters in Order of the List in Section 4.2.2											
f8190cf2df70d8456154e7685ea4b00ec279f695558b8b43a10d43bd1a2d9f02	65527	10485760	0	1048576	1048576	256	3	3	25	false	0
270e04f2d01595bd9d2848ecc1da8294460335986398d9f8aba1c674dd659177	1472	196608	131072	131072	131072	100	103	3	25	true	0
a1cf673bc1a6b51fc9c7627a8704f9cb62f8c6da8a1d3d97464e3c900d7b17e6	1472	196608	131072	131072	131072	100	103	3	25	false	0
3db4ec1a7142968f7d8a6982ef10e63bf6eadb01b89167b28702c0711b53daaf	65527	10485760	0	1048576	1048576	128	3	3	25	false	0
e1419b37eba377c0614e576146acaac0e6d42a1e1091d5fde707cb7854ebda56	1404	1048576	67584	67584	67584	100000	100000	3	25	false	0
a97efb4a17437de674a746ac0f93242e68a6760b95ef0267bd479548e1638f74	1500	1048576	67584	67584	67584	100000	100000	3	25	false	0
58f04fc114f6466c3bc8096d20af6a4c99811b24a739f6833cbd547e25a182f2	1404	10485760	10485760	10485760	10485760	100000	100000	3	25	false	0
c3af1b56bf0eda20e304b2e11e1893211147768ef58e924cb3f183d8350d9cb9	1452	786432	524288	524288	524288	100	100	3	26	true	4
63e3c4b3611ad98c92bed3c961c0eb55ed8e56a600d8d3e1462c2b2d7b08caeb	1472	1048576	65536	65536	65536	100	103	3	25	false	0
dfda9c00d33528f41363387dbdfb0061ba67e8d17d1fe01df1fe5142d5e5494f	1500	10485760	10485760	10485760	10485760	100000	100000	3	25	false	0
6a194a5ac055cfb2e6e531bbe8bbab1402714d7b9b19fdceb2b96d4b4e88efff	1472	16777216	1048576	32768	1048576	100	10	10	25	false	4
fe3d5a3a520cebf65d8770c4a430c88962a0b12fe018c53afb290a32c9f0b66c	1452	786432	524288	524288	524288	1000	100	3	26	true	4
6d6bd50c8b417a1e09f5031d141dc6902b016dab8d43b345c79aaf1fd328ca61	65527	16777216	1048576	1048576	1048576	16	3	3	25	false	2
92909c91afd3e3c854dee6532defdadeced9f55212d989510dd22b7513773c8b	65527	1048576	65536	65536	65536	16	16	3	25	false	2
b3ac794dfbe021852fa9d5abbfe584f0af60a2be9d18b904278ee270b97612df	1472	1048576	1048576	1048576	1048576	100	3	3	25	true	0
4dc1a31f321a3eb21979d6c6e8695280223e529a93728c76c9e70894f60017bc	1452	786432	524288	524288	524288	256	256	3	26	true	4
2ea099cf6b4141e6de5822d8336c5f22e19a890380cd765cacf15f1c06d57f4d	65527	1048576	65536	65536	65536	16	3	3	25	false	2
522e3bf2c77d999c1bfa5974d03f43c52ab5c5bf44c224880d5f811f92918c5d	4611686018427387903	1048576	65536	65536	65536	16	16	3	25	false	2
f043c83232f83907d140e2920a124b9057b3d1494bc91c62490288b51c419d6d	1472	196608	131072	131072	131072	10	13	3	25	true	0
b55ff0f5ed4773dc71f629e027fd8ab2ac4ea9da192ba13a5a7b6d5f5e943d21	1472	196608	131072	131072	131072	30	33	3	25	true	0
68c1026e55b4ca77ff9eba324034efc9503709325e357121ff69c31d25bd0e02	65527	10485760	0	1048576	1048576	256000	3	3	25	false	0
73f2f476e521478f37578ef0764da275193a7522950248a3c84c88d6dbd742b0	65527	10240	0	1024	1024	128	3	3	25	false	0
a0483a57ad60e7d2bf088deef2dd4d455a9066bb2283712e5099a45650738f7b	65527	10485760	0	1048576	1048576	12800	10000	3	25	false	0
ca0608af548c52ce538ebcfe7a8291ed3c2ef3200a382cb065234a316f78803d	1480	16777216	1048576	1048576	1048576	96	32	3	0	false	5
9e9eb3623b121dc07a4384aa866dc8a1ef1bcd44a43749193ca48942d9985892	1472	16777216	32768	32768	32768	100	3	8	41	false	4
a01356da5b3021765fa5b27fc923f6e9aa86a22a83aeafbf562ff64fdde712c9	1350	8192	1000000	1000000	1000000	32	100	3	25	true	0
56b2350023d8815f5f77a094686602ab48aec240736571a28d9f0f64ec351466	65508	1048576	1250000	1250000	1250000	32	32	3	0	false	5
505288d26d1db782dfaf56f05d1a66f2cf4249ba0ed154b9aa64950308737411	1472	196608	131072	131072	131072	10000	10003	3	25	true	0
9bac3985f264a78053bd4f4579d00ef9e47d95aab8f0d9388313b5c508e8e32f	65527	16777216	1048576	1048576	1048576	16	16	3	25	false	2
a347199b7dd21aa1fa1ba7a9660a90211b179fc8cff337449826708a56d2fe7e	65527	1048576	65536	65536	65536	100	16	3	25	false	2
831812fe766038c5d500e70e97e27c01d9b7755eecd4d738c9beba704802a7de	1500	1048576	66560	66560	66560	2048	2048	3	25	false	0
b7d0f79f5a0ef10415160bac16e27ca8d5e77b2a43ad302c8ad64eaaec85f71d	65527	1048576	65536	65536	65536	100	3	3	25	false	2
6baea98cdbe567c598e51d8964ef527bee325e077a2da15571ba8e04860bf9c6	65527	2097152000000	0	1048576000000	1048576000000	256	3	3	25	false	0
c86c946d870169f18fb0e9c7b0139e5cd6f92b12cdb63cdf7ed624ad1a0ac948	1500	10000000	1000000	1000000	1000000	100	10	0	25	false	2
849e2c3b8754607b711175c21ae37af072e2c6e2d9365db6ca1ecb83da1dac46	1472	16777216	1048576	1048576	1048576	100	10	10	25	false	4
08789993ffffc411f29c746767df4f8bda00f3f094ed9c73fe024b5729cb45fa	4611686018427387903	65536	0	4096	4096	100	100	3	25	false	4
38c3331e05362b7794717cea11dc617669cb508e7d28d1eb33c998e5cc0e188c	1350	10000000000	1000000000	1000000000	1000000000	100	100	3	25	true	0
7eec69484be49a552e7356581295a0d442580d67c515dee15197cd431af29c0a	1480	4611686018427387903	1250000	1250000	1250000	100	0	3	0	false	5

Table A.1: Configuration Hashes and Parameter

Chapter B

List of Acronyms

AEAD	Authenticated Encryption with Associated Data.
AES	Advanced Encryption Standard.
ALPN	Application-Layer Protocol Negotiation.
AS	Autonomous System.
BGP	Border Gateway Protocol.
CDN	Content Delivery Network.
CSV	Comma-Separated Values.
DNS	Domain Name System.
ECB	Electronic Code Book.
gQUIC	Google QUIC.
HKDF	HMAC-based Extract-and-Expand Key Derivation Function.
HTTP/2	Hypertext Transfer Protocol Version 2.
HTTP/3	Hypertext Transfer Protocol Version 3.
HTTP	Hypertext Transfer Protocol.
IETF	Internet Engineering Task Force.

ISO	International Organization for Standardization.
OSI	Open Systems Interconnection. (Reference model for layered network architectures by the OSI.)
SNI	Server Name Indication.
TCP	Transmission Control Protocol. (Stream-oriented, reliable, transport layer protocol.)
TLS	Transport Layer Security. (Transport Layer Security)
UDP	User Datagram Protocol. (Datagram-oriented, unreliable transport layer protocol.)

Bibliography

[1] David Adrian et al. „Zippier ZMap: Internet-Wide Scanning at 10 Gbps“. In: *8th USENIX Workshop on Offensive Technologies (WOOT 14)*. San Diego, CA: USENIX Association, Aug. 2014. URL: https://www.usenix.org/conference/woot14/workshop-program/presentation/adrian.

[2] Internet Corporation for Assigned Names and Numbers. *Centralized Zone Data Service*. 2020. URL: https://czds.icann.org/ (visited on 05/02/2021).

[3] Justin Bastress. *Module to scan for gQUIC support*. 2018. URL: https://github.com/zmap/zmap/pull/494 (visited on 05/12/2021).

[4] Tony Bates, Philip Smith, and Geoff Huston. *CIDR REPORT*. 2021. URL: https://www.cidr-report.org/as2.0/autnums.html (visited on 05/09/2021).

[5] Mike Bishop. *HTTP/3: Ready to Land*. 2020. URL: https://blogs.akamai.com/2020/11/http3-ready-to-land.html (visited on 05/12/2021).

[6] Mike Bishop. *Hypertext Transfer Protocol Version 3 (HTTP/3)*. Internet-Draft draft-ietf-quic-http-34. Work in Progress. Internet Engineering Task Force, Feb. 2021. 75 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34.

[7] Simon Blake-Wilson et al. *Transport Layer Security (TLS) Extensions*. RFC 3546. June 2003. DOI: 10.17487/RFC3546. URL: https://rfc-editor.org/rfc/rfc3546.txt.

[8] Lucas Clemente. *A QUIC implementation in pure Go*. https://github.com/lucas-clemente/quic-go. 2021.

[9] Dragana Damjanovic. *QUIC and HTTP/3 Support now in Firefox Nightly and Beta*. 2021. URL: https://hacks.mozilla.org/2021/04/quic-and-http-3-support-now-in-firefox-nightly-and-beta/ (visited on 05/12/2021).

[10] David Schinazi and Fan Yang and Ian Swett. *Chrome is deploying HTTP/3 and IETF QUIC*. Oct. 2020. URL: https://blog.chromium.org/2020/10/chrome-is-deploying-http3-and-ietf-quic.html (visited on 05/09/2021).

[11] Piet De Vaere et al. „Three Bits Suffice: Explicit Support for Passive Measurement of Internet Latency in QUIC and TCP“. In: *Proceedings of the Internet Measurement Conference 2018*. IMC '18. Boston, MA, USA: Association for Computing Machinery, 2018, 22–28. ISBN: 9781450356190. DOI: 10.1145/3278532.3278535. URL: https://doi.org/10.1145/3278532.3278535.

[12] Eva Gagliardi and Olivier Levillain. „Analysis of QUIC Session Establishment and Its Implementations“. In: *Information Security Theory and Practice*. Ed. by Maryline Laurent and Thanassis Giannetsos. Cham: Springer International Publishing, 2020, pp. 169–184. ISBN: 978-3-030-41702-4.

[13] Oliver Gasser et al. „Clusters in the Expanse: Understanding and Unbiasing IPv6 Hitlists“. In: *Proceedings of the Internet Measurement Conference 2018*. IMC '18. Boston, MA, USA: Association for Computing Machinery, 2018, 364–378. ISBN: 9781450356190. DOI: 10.1145/3278532.3278564. URL: https://doi.org/10.1145/3278532.3278564.

[14] IETF QUIC Working Group. *Implementations*. 2021. URL: https://github.com/quicwg/base-drafts/wiki/Implementations (visited on 05/02/2021).

[15] IETF QUIC Working Group. *QUIC Versions*. 2021. URL: https://github.com/quicwg/base-drafts/wiki/QUIC-Versions (visited on 01/10/2021).

[16] Alexa Internet. *Alexa - Top Sites*. 2021. URL: https://www.alexa.com/topsites (visited on 05/02/2021).

[17] Jana Iyengar. *The state of QUIC and HTTP/3 2020*. 2020. URL: https://www.fastly.com/blog/state-of-quic-and-http3-2020 (visited on 05/12/2021).

[18] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Internet-Draft draft-ietf-quic-transport-00. Work in Progress. Internet Engineering Task Force, Nov. 2016. 45 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-00.

[19] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Internet-Draft draft-ietf-quic-transport-34. Work in Progress. Internet Engineering Task Force, Jan. 2021. 207 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-34.

[20] Matt Joras and Yang Chi. *How Facebook is bringing QUIC to billions*. 2020. URL: https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/ (visited on 05/12/2021).

[21] Arash Molavi Kakhki et al. „Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols“. In: *Proceedings of the 2017 Internet Measurement Conference*. IMC '17. London, United Kingdom: Association for Computing Machinery, 2017, 290–303. ISBN: 9781450351188.

DOI: 10.1145/3131365.3131368. URL: https://doi.org/10.1145/3131365.3131368.

[22] Dr. Hugo Krawczyk and Pasi Eronen. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. RFC 5869. May 2010. DOI: 10.17487/RFC5869. URL: https://rfc-editor.org/rfc/rfc5869.txt.

[23] J. Lin. „Divergence measures based on the Shannon entropy“. In: *IEEE Transactions on Information Theory* 37.1 (1991), pp. 145–151. DOI: 10.1109/18.61115.

[24] David McGrew. *An Interface and Algorithms for Authenticated Encryption*. RFC 5116. Jan. 2008. DOI: 10.17487/RFC5116. URL: https://rfc-editor.org/rfc/rfc5116.txt.

[25] University of Oregon. *Route Views Archive Project*. 2021. URL: http://www.routeviews.org/routeviews/ (visited on 05/09/2021).

[26] Lucas Pardue. *A Last Call for QUIC, a giant leap for the Internet*. 2020. URL: https://blog.cloudflare.com/last-call-for-quic/ (visited on 05/12/2021).

[27] Maxime Piraux, Quentin De Coninck, and Olivier Bonaventure. „Observing the Evolution of QUIC Implementations“. In: *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*. EPIQ'18. Heraklion, Greece: Association for Computing Machinery, 2018, 8–14. ISBN: 9781450360821. DOI: 10.1145/3284850.3284852. URL: https://doi.org/10.1145/3284850.3284852.

[28] *quicly*. 2021. URL: https://github.com/h2o/quicly (visited on 05/09/2021).

[29] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446. URL: https://rfc-editor.org/rfc/rfc8446.txt.

[30] Jim Roskind. *Quick UDP internet connections: Multiplexed stream transport over UDP*. 2012. URL: https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/ (visited on 01/25/2021).

[31] Jan Rüth et al. „A First Look at QUIC in the Wild“. In: *Passive and Active Measurement*. Ed. by Robert Beverly, Georgios Smaragdakis, and Anja Feldmann. Cham: Springer International Publishing, 2018, pp. 255–268. ISBN: 978-3-319-76481-8.

[32] Amazon Web Services. *Enable the QUIC UDP transport protocol*. 2021. URL: https://docs.aws.amazon.com/dcv/latest/adminguide/enable-quic.html (visited on 05/09/2021).

[33] Martin Thomson and Sean Turner. *Using TLS to Secure QUIC*. Internet-Draft draft-ietf-quic-tls-34. Work in Progress. Internet Engineering Task Force, Jan. 2021. 66 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-quic-tls-34.

[34] Medhat Yakan and Akhil Jayaprakash. *Introducing QUIC for Web Content*. 2018. URL: https://developer.akamai.com/blog/2018/10/10/introducing-quic-web-content (visited on 05/12/2021).