A Quic(k) Security Overview: A Literature Research on Implemented Security Recommendations

Stefan Tatschner
Fraunhofer Institute AISEC
Garching bei München, Germany
University of Limerick
Limerick, Ireland
stefan.tatschner@aisec.fraunhofer.de

Sebastian N. Peters
Fraunhofer Institute AISEC
Garching bei München, Germany
sebastian.peters@aisec.fraunhofer.de

David Emeis
Fraunhofer Institute AISEC
Garching bei München, Germany
david.emeis@aisec.fraunhofer.de

John Morris
University of Limerick
Limerick, Ireland
john.morris@ul.ie

Thomas Newe
University of Limerick
Limerick, Ireland
thomas.newe@ul.ie

ABSTRACT

Built on top of UDP, the relatively new QUIC protocol serves as the baseline for modern web protocol stacks. Equipped with a rich feature set, the protocol is defined by a 151 pages strong IETF standard complemented by several additional documents. Enabling fast updates and feature iteration, most QUIC implementations are implemented as user space libraries leading to a large and fragmented ecosystem. This work addresses the research question, "if a complex standard with a large number of different implementations leads to an insecure ecosystem?". The relevant RFC documents were studied and "Security Consideration" items describing conceptional problems were extracted. During the research, 13 popular production ready QUIC implementations were compared by evaluating 10 security considerations from RFC9000. While related studies mostly focused on the functional part of QUIC, this study confirms that available QUIC implementations are not yet mature enough from a security point of view.

CCS CONCEPTS

• **Security and privacy** → **Web protocol security**; *Security requirements*; • **Networks** → *Web protocol security*; Transport protocols.

KEYWORDS

QUIC, RFC9000, security considerations, web

ACM Reference Format:
Stefan Tatschner, Sebastian N. Peters, David Emeis, John Morris, and Thomas Newe. 2023. A Quic(k) Security Overview: A Literature Research on Implemented Security Recommendations. In *The 18th International Conference on Availability, Reliability and Security (ARES 2023), August 29–September 01, 2023, Benevento, Italy*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3600160.3605164

This work is licensed under a Creative Commons Attribution International 4.0 License.

ARES 2023, August 29–September 01, 2023, Benevento, Italy

© 2023 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-0772-8/23/08.
https://doi.org/10.1145/3600160.3605164

1 INTRODUCTION

QUIC is a general purpose transport layer network protocol which was designed by Jim Roskind at Google in 2012 and publicly announced as an experiment in 2013. While being built on top of the connectionless but efficient User Datagram Protocol (UDP) [1], it was submitted to the Internet Engineering Task Force (IETF) for standardization in 2015 and finally published as RFC9000 [14], which is supported by RFC8999 [27], RFC9001 [29], and RFC9002 [13] in 2021. As specified in RFC9001, QUIC uses the Transport Layer Security (TLS) [21] protocol which enables authentication of peers and provides confidentiality and integrity protection for messages exchanged by endpoints.

Originally developed in the 1970s, the Transmission Control Protocol (TCP) [10] nowadays suffers from some limitations, e.g. slow connection setup times, especially when it is used with TLS. For instance with TCP+TLS, a client must go through both the TCP three-way and the TLS handshake. QUIC was created to address some of these restrictions and in doing so, has earned it the nickname "TCP/2". The most common case for connection establishment requires four roundtrips for TCP+TLS and one roundtrip for QUIC [8].

The reason for the increasing adoption of QUIC is its incorporation into the web protocol stack as the Hypertext Transfer Protocol (HTTP) is used by more than five billion people for accessing the internet [25]. Facebook announced in 2020 that more than 75% of their Internet traffic uses QUIC and HTTP/3 [7]. The current version HTTP/3 is designed to delegate features provided by the HTTP/2 framing layer to native QUIC features. A good example are HTTP/2 [28] streams which were multiplexed over a single TCP connection in the previous HTTP version. In HTTP/3, streams are provided by the QUIC protocol stack [4].

In order to allow rapid iteration of the protocol, the designers intended the protocol stack to be located in user space. Consequently, multiple implementations for different programming languages exist today. Being a basic building block of the current version of the most used internet protocol, different QUIC implementations are built into major browsers[1], such as Chromium, Firefox, or Safari.

[1] https://caniuse.com/http3

Due to its dissemination, QUIC carries responsibility for internet users in terms of privacy and security. For instance, there have been multiple attempts to compromise the security of the internet. In the past, there were reports about undersea cable tapping [3] or attempts to weaken the security parameters of TLS [9].

In RFC documentation, sections called "Security Consideration" are used to discuss conceptional weaknesses in published network protocols according to the attacker model explained in RFC3552 [22]. If necessary, follow-up RFC documents are published to keep the older documents up-to-date. Unfortunately, the current approach of the IETF has a downside. For instance, QUIC has a documented known vulnerability called opt-ack which is already known from TCP and might cause a wide-spread congestion collapse [24]. Countermeasures for opt-ack are well known, but for TCP, developers decided not to implement them due to performance degradation reasons. In the QUIC RFC, there are suggestions for a mitigation strategy but the relevant sections are tagged with the RFC2119 keywords MAY or SHOULD [6] turning them into *optional* countermeasures.

The research question this paper addresses is: "Does the complexity of the RFC standard on the one hand and the large number of different QUIC implementations on the other hand lead to an insecure QUIC ecosystem?". Initially, the conceptional weaknesses of QUIC, documented by the designers and their mitigation strategies are investigated. Subsequently, exploratory Internet research on existing QUIC implementations is conducted. From these results, 13 libraries were chosen for a deeper analysis. As a next step, the selected QUIC implementations are analyzed to determine if they implemented the suggested mitigation strategies. Finally, the results are compared and discussed. In this study the scope was intentionally limited to the QUIC protocol, additionally incorporated protocols, such as TLS, were not considered.

The presented paper aims to give an update of the available QUIC implementations by conducting an analysis of the QUIC environment. Further, the available analysis of vulnerabilities is complemented by comparing QUIC implementations with the security recommendations from the RFC documents.

2 RELATED WORK

Sherwood et al. and Adamsky et al. both show in their studies from 2005 and 2012 how misbehaving receivers could have a major impact on whole network infrastructures [24, 2]. To overcome such emerging problems in their standards, the IETF publishes documents updating already established RFCs. For instance, there is RFC5961 [26] specifying a modification of the TCP inbound segment handling, reducing the chances of a successful, known attack. On the downside, this modification is also formulated as an optional item and is not specified as a mandatory TCP update, thereby increasing the number of possible ways of implementing the protocol. The reason for specifying updates as optional is often for backwards compatibility. However, there are other approaches like flag days known from the Domain Name System (DNS) which help to move the ecosystem forward[2].

Piraux et al. created a QUIC test suite in 2018 that interacts with public QUIC servers to verify their conformance with key features of the IETF specification [20]. The authors compared 15 different QUIC implementations revealing that their grade of conformance to the IETF specification differs largely. Additionally, this paper provides an overview of the evolution of RFC2119 keywords. Approximately half of all tracked keywords in the QUIC specification – at the time of this study – were SHOULD or SHOULD NOT, respectively. In other words, the QUIC specification contains a significant amount of optional requirements. This work demonstrates that the complexity and optional keyword nature of the QUIC protocol leads to many implementations behaving differently for key features.

Marx et al. in 2018 proposes a standardized logging and visualization format for debugging and analyzing QUIC [18]. This proposal led to a RFC draft describing a high-level schema for a standardized logging format called qlog, which is currently at revision 5 [19]. The goal is a standardized logging format which is implemented across multiple different implementations that provides an opportunity for comparing them at runtime.

In a subsequent work, Marx et al. analyzed and discussed 15 different QUIC implementations in 2020 [17]. They used their previously presented approach based on the qlog format to debug and compare the QUIC stacks. The authors concluded that even though these stacks all implemented exactly the same RFCs, their low-level implementation choices led to large differences in behavior between them. These differences may be caused by the large number of optional items, as outlined in [20]. It might be expected that many implementations eventually develop into a smaller set of best practices rather than the observed approaches.

In 2023, Chatzoglou et al. presented a comprehensive overview of 18 QUIC implementations and they analyzed the QUIC security through the lens of relevant literature [7]. The authors revealed several practical zero-day vulnerabilities and came to the conclusion that the available fragmented production-level implementations might not yet be mature enough.

3 METHODOLOGY

This research consists of multiple, consecutive working steps. The results were collected in separate tables including additional metadata, e.g. keywords for search or further references. As these tables form the database for the evaluation, they will be published as additional data to this paper. From a conceptional point of view, the methodology is structured as follows:

(1) **Identification of relevant RFC documents**: The relevant RFC documents are presented on the QUIC working group's homepage[3]. There are a few documents which (at the time of writing) only exist as draft documents; draft documents were not considered. During this research it turned out that only RFC9000 [14] contains relevant *optional* measures. Other documents, such as RFC9001 [29], RFC9308 [15], or RFC9312 [16] contain Security Considerations but they are either not applicable to the present evaluation or they just reference RFC9000.
(2) **Analysis of "Security Considerations"**: The "Security Considerations" sections were analyzed in order to find concrete, documented problems with proposed solutions. The relevant RFC2119 keywords were extracted to outline

[2] https://www.isc.org/blogs/dns-flag-day-2020/

[3] https://quicwg.org/

Project	⋆	Backed by	Language	TLS Stack	References
quic-go	8.0k	Google	Go	modified Go stdlib	Syncthing, caddy, cloudflared, traefik
cloudflare/ quiche	7.4k	Cloudflare	Rust	Boring SSL	Android (DNS), curl, nginx
MsQuic	3.3k	Microsoft	C	SChannel	HTTP/3 and SMB Stack on Windows
Quinn	2.7k	Instant Domains, Inc	Rust	multiple (default: rustls)	hyper/reqwest
Neqo	1.6k	Mozilla	C, C++	NSS	Firefox
XQUIC	1.4k	Alibaba	C	BoringSSL or BabaSSL	tengine
mvfst	1.3k	Facebook	C++	Fizz	internally at Facebook
aioquic	1.2k	Jeremy Lainé	Python	custom	hypercorn, httpx
LSQUIC	1.2k	LiteSpeed Technologies Inc	C	BoringSSL	internally at LiteSpeed
ngtcp2	0.9k	Tatsuhiro Tsujikawa	C++	multiple	curl
s2n-quic	0.9k	Amazon Web Services	Rust	s2n-tls or rustls	Amazon Web Services
quicly	0.6k	Fastly, DeNA Co. Ltd.	C	BoringSSL	H2O webserver
google/ quiche	0.4k	Google	C, C++	BoringSSL	Chromium

Table 1: Overview of identified production ready QUIC implementations, sorted by Github stars. The full version of this table is available in the supplementary material.

whether the proposed solution is considered mandatory or optional. For the result of this work step, see Section 4.

(3) **Online search for QUIC implementations**: The number of libraries for this analysis was limited by the following constraints: Only implementations that consider themselves as production-ready or implementations that are already used by popular applications, such as curl[4]. Only Free and open-source software (FOSS) projects, because an important point are publicly available discussions on development decisions. Support for QUIC v1. The selection should cover major programming languages, including C, C++, Go, Python, and Rust. For the result of this work step, see Table 1. For further reference, an extended version of this table, also including all omitted implementations, is available online[5].

(4) **Evaluation of implemented measures**: The identified implementations were reviewed, if the referenced 'Security Considerations' were implemented. For this purpose, a manual approach was chosen: for Github-based projects, the source code, issue tracker, and pull requests were searched for relevant keywords. For non Github-based projects, at least the source code was searched; additional related resources were considered if available. For the result of this work step, see Table 2. Additional information, such as the used search keywords, are published online[5].

4 SECURITY CONSIDERATIONS

In their documents, the IETF highlights possible conceptual weaknesses in standardized sections, called "Security Considerations". The RFC documents outlined by the QUIC working group were investigated and relevant sections were extracted. Some text passages define optional (security related) components of the QUIC protocol that are not required by the standard to be implemented. Indicators for optional sections are the RFC 2119 [6] keywords SHOULD (NOT) and MAY (NOT); SHOULD indicates an item is recommended, or in other words, there are valid reasons to consider a particular item. MAY indicates that an item is truly optional. Furthermore, there are mandatory items marked with the keyword MUST (NOT). The following conceptional weaknesses were identified by analyzing the Security Considerations sections in the relevant RFC documents.

Table 2 represents the results of this evaluation. Each following section cites the relevant part in the RFC and gives an explanation about what the authors searched for in each implementation to add a ✓ mark in the table. For a few results, the authors needed to simplify the criteria, otherwise the analysis would go beyond the scope of this paper.

4.1 Amplification

This attack is described in [14, Sec. 21.3]. A peer can be tricked into sending packets to a victim by abusing address validation tokens. The proposed mitigation strategy is limiting the lifetime of the mentioned tokens. The RFC 2119 keyword SHOULD is used in the referenced section. The ✓ mark was applied if the project showed indications that at least basic anti-amplification mechanisms are implemented.

4.2 Optimistic ACK

This attack is described in [14, Sec. 21.4]: "An endpoint that acknowledges packets it has not received might cause a congestion controller to permit sending at rates beyond what the network supports." The proposed strategy to detect this attack is skipping packet numbers. The RFC 2119 keyword MAY is used in the referenced section. The ✓ mark was applied if the project has implemented a mechanism to skip packet numbers.

[4] https://curl.se
[5] https://rumpelsepp.org/projects/quic-overview

4.3 Slowloris

This attack is described in [14, Sec. 21.6]: The attacks commonly known as Slowloris tries to keep many connections to the target endpoint open and hold them open for as long as possible. The proposed strategy is choosing appropriate limits for a QUIC connection, for instance limiting the maximum number of clients, or a minimum transfer speed. The RFC 2119 keyword SHOULD is used in the referenced section. The ✓ mark was applied if the project showed indications that Slowloris is explicitly addressed or a fine-grained limiting system is available.

4.4 Stream Fragmentation and Reassembly

This attack is described in [14, Sec. 21.7]. This attack can become arbitrary complex; to summarize, an attacker tries to force a peer to allocate lots of memory by modifying packet numbers, for example. The proposed mitigations could consist of avoiding over committing memory, limiting the size of tracking data structures, delaying reassembly of STREAM frames, implementing heuristics based on the age and duration of reassembly holes, or some combination of these. The RFC 2119 keyword SHOULD is used in the referenced section. The ✓ mark was applied if the project showed indications that one of the proposed mitigations is present or that the issue is explicitly addressed.

4.5 Stream Commitment

This attack is described in [14, Sec. 21.8]: "An adversarial endpoint can open a large number of streams, exhausting state on an endpoint". The proposed mitigation is to set appropriate limits. There is an own section which explains how these limits are determined [14, Sec. 4.6]. The RFC 2119 keyword MUST is used in the referenced section. The ✓ mark was applied if the project implemented a system to limit stream resources.

4.6 Peer Denial of Service

This attack is described in [14, Sec. 21.9]: "QUIC and TLS both contain frames or messages that have legitimate uses in some contexts, but these frames or messages can be abused to cause a peer to expend processing resources without having any observable impact on the state of the connection." The proposed technique for detecting this attack is "track cost of processing relative to progress". Implementations should then react with an appropriate error, for instance with terminating the connection. The RFC 2119 keywords SHOULD and MAY are used in the referenced section. The ✓ mark was applied if the project has implemented a system to track connection-specific statistics.

4.7 Explicit Congestion Notification

This attack is described in [14, Sec. 21.10]: "An on-path attacker could manipulate the value of ECN fields in the IP header to influence the sender's rate. RFC3168 [11] discusses manipulations and their effects in more detail." The proposed mitigation is ignoring the Explicit Congestion Notification (ECN) field in the IP header *unless* the packet passes a validation step which is described in [14, Sec. 13.4.2]. There are no RFC2119 keywords used in this section and ECN is an optional IP feature [11]. The ✓ mark was applied if the project has support for the ECN field and showed indications that a validation step is implemented.

4.8 Stateless Reset Oracle

A Stateless Reset is a response to a packet that cannot be associated with an active connection and is a possibility to stop peers continuing to send packets. This attack is described in [14, Sec. 21.11]: "Stateless resets create a possible denial-of-service attack analogous to a TCP reset injection." This attack depends on a special condition of connection IDs and reset tokens. Implementations must avoid this special condition. The RFC 2119 keywords MUST and MUST NOT are used in the referenced section. The ✓ mark was applied if the project explicitly addressed the stateless reset vulnerability.

4.9 Version Downgrade

This attack is described in [14, Sec. 21.12]: "Future versions of QUIC that use Version Negotiation packets MUST define a mechanism that is robust against version downgrade attacks." At the time of writing, a mechanism for QUIC version negotiation exists as a IETF draft [23]. A few QUIC libraries still implement draft revisions of QUIC and may be vulnerable to downgrade attacks. The RFC 2119 keyword MUST is used in the referenced section. The ✓ mark was applied if the project explicitly addressed the version downgrade problem - not necessarily by implementing the brand new QUIC draft already.

4.10 Traffic Analysis

This attack is described in [14, Sec. 21.14]: "The length of QUIC packets can reveal information about the length of the content of those packets." Implementations can use PADDING frames to obfuscate the length of packets. There are no RFC2119 keywords used in this section. The authors are aware that hiding all information in network traffic is a complex task on its own, cf. [5]. Therefore, the ✓ mark was applied if PADDING can be controlled via an Application Programming Interface (API) from a particular application.

5 EVALUATION

The results of the conducted evaluation can be obtained from Table 2. For the table, different symbols were chosen:

- ✓ The mitigation according to the defined criteria in Section 4 is addressed.
- **[✓]** A special case of the security consideration is implemented, therefor the defined criteria in Section 4 are only partially addressed.
- ✗ The mitigation according to the defined criteria in Section 4 is *not* addressed.
- **n/a** The mitigation is out of scope and not available for the project, since a required underlying feature is not implemented.
- * The project is not developed in public but behind closed doors. Such projects have very little public information available, such as code reviews or discussions about the design decisions. The results of these projects need to be taken with a grain of salt.

Security Consideration	*quic-go*	*c/quiche*	*MsQuic*	*Quinn*	*Nego*	*XQUIC**	*mvfst**	*aioquic*	*LSQUIC*	*ngtcp2*	*s2n-quic*	*quicly*	*g/quiche**
Amplification (cf. Sec. 4.1)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Optimistic ACK (cf. Sec. 4.2)	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✓	✗
Slowloris (cf. Sec. 4.3)	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗
Stream Fragmentation and Reassembly (cf. Sec. 4.4)	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
Stream Commitment (cf. Sec. 4.5)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Peer Denial of Service (cf. Sec. 4.6)	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	[✓]	✗	✗
Explicit Congestion Notification (cf. Sec. 4.7)	n/a	n/a	✓	✓	n/a	n/a	n/a	n/a	n/a	✓	✓	n/a	✓
Stateless Reset Oracle (cf. Sec. 4.8)	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Version Downgrade (cf. Sec. 4.9)	✓	✗	✓	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓
Traffic Analysis (cf. Sec. 4.10)	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

Table 2: Evaluation Results. ✓ means the mitigation is available as defined in Section 4. [✓] means that a special case of this mitigation is implemented. ✗ means the mitigation is not available. The abbreviation n/a indicates that a required underlying feature is not available. Projects with a star * seem to be developed behind closed doors and the results need to be taken with a grain of salt. The full version of this table is available in the supplementary material.

Three projects, mvfst, google/quiche, and XQUIC, have public available source code but the development process itself is not visible. The XQUIC project was very difficult to analyze with the chosen approach. The project contains over 60k lines of C code but the code history is structured into only 78 commits. Since there is no further documentation or metadata in the form of discussions or issues available, it was required to analyze the source code of XQUIC and rely on comments or function names.

All projects implement measures against Amplification attacks. The security considerations Optimistic ACK (implemented by three), Slowloris (implemented by three), and Version Downgrade (implemented by six) were addressed by a few projects. These special weaknesses were relatively easy to spot, because they were often documented in the source code or in the issue tracker. Stream Fragmentation is addressed by LSQUIC. The project even offers documentation about buffer handling which is claimed to be not vulnerable to stream fragmentation attacks by its authors. Stream Commitment was addressed by all projects. Stream Commitment was found to be similar to the Peer Denial of Service consideration that no project has addressed. The difference between those two is that mitigating Peer Denial of Service requires a more fine grained tracking of connection resources and limits. Quinn and quicly were the only projects that implement a fine-grained connection statistics system. However, this statistics system is only used for meta information and not for triggering certain actions, such as terminating a connection. There is one exception, though: s2n-quic implements a measure against reflection attacks[6], which falls into the category of Peer Denial of Service but it is a special case. Explicit Congestion Notification requires an underlying feature to be present. Only four projects have implemented this feature. If the underlying feature was implemented, the relevant security consideration was addressed as well. Traffic Analysis is the only item, which is not addressed by any of the QUIC implementations.

The authors of the s2n-quic chose a commendable and transparent approach. They extracted RFC2199 keywords from all involved RFCs and created a machine readable compliance overview. Each RFC2199 item was assigned a status (e.g. completed, missing tests, …) and if available a tracking issue on Github. Unfortunately, most items relevant for this study (cf. Section 4) had status set to "unknown". The weblink to the compliance report is very large due to the usage of hash values in the URL. Therefore, it is not appropriate for this paper's layout. However, the weblink is included in the supplementary material.

During the evaluation, a reference[7] to a later standard, *DNS over Dedicated QUIC Connections* [12] was noticed. This standard includes a section that references RFC9000 according to traffic analysis. This section is specified with the keyword MUST: "Implementations MUST protect against the traffic analysis attacks described in Section 7.5 by the judicious injection of padding. This could be done either by padding individual DNS messages using the EDNS(0) Padding Option (…) or by padding QUIC packets (…)." [12, Sec. 5.4].

During the evaluation, it was further noticed that an IETF draft *Compatible Version Negotiation for QUIC* [23] is intended to be published as RFC9368 but has not yet been approved by one of its authors[8]. Interestingly enough, LSQUIC seems to be the first project claiming RFC9368 support. This (potential) RFC9368 requires downgrade protection with the keyword MUST.

6 DISCUSSION

According to Section 3, this study used a manual approach for evaluating the implemented measures. In contrast to presented related studies in Section 2, there is no test suite or automated tooling available. A manual approach is more sensitive to human error than a machine-aided approach; especially because the relevant data needs to be collected by hand from different sources, such as code

[6] https://github.com/aws/s2n-quic/issues/1259

[7] https://github.com/ngtcp2/ngtcp2/issues/626

[8] https://www.rfc-editor.org/auth48/rfc9368

comments, Github issues and documentation. This approach does *not endorse* the correctness of an implemented countermeasure, it only shows that the developers are aware of certain problems that need to be addressed. The authors are aware that some problems, for instance Slowloris, Stream Commitment, or Peer Denial of Service, are expected to be addressed by similar looking countermeasures. However, a manual approach enables accidental findings like RFC9250 where certain optional security considerations from RFC9000 are required with the keyword MUST. Because the basic standard lacks keywords for Traffic Analysis, cf. Section 4.10, no project has implemented suitable APIs up to now. Consequently, this study confirms the results presented in related work [7, 18, 20]: The QUIC ecosystem suffers from being fragmented with many implementations that have unique differences.

It was observed that only one project implements a mitigation strategy for Stream Fragmentation (cf. Section 4.4) and no project implements Peer Denial of Service mitigation. One possible explanation for this is that our search technique used is not suitable for this kind of mitigation. Since Stream Fragmentation and Peer Denial of Service are complex topics, the text in the RFC document for the former only suggests using generic techniques, such as avoiding over-committing memory or limiting the size of data structures, rather than specifying a particular mitigation technique. For the latter, tracking the cost for each connection is suggested and then appropriate actions should be triggered in case certain limits are reached. Due to the generic nature of these mitigations, it is assumed that perhaps they were not recognized by the manual approach. No project seems to address Traffic Analysis (cf. Section 4.10). The authors have formed the impression that Traffic Analysis was not on the radar of most developers. There are Github issues[9] [10] asking for APIs to specify PADDING, but no one has written the required code yet. It is expected that this will change and suitable APIs will start to appear, because the recently published RFC9250 explicitly requires the implementation of Traffic Analysis.

While related studies mostly focused on the functional part of QUIC, this study found that the hypothesis of a fragmented ecosystem is also true for the security related parts. The RFC authors' decision to locate the QUIC protocol stack into user space, led to separate implementations from each of the big players, such as Cloudflare, Facebook, Google, and Microsoft. For instance, Google maintains two distinct QUIC implementations: quic-go and google/quiche which is shipped with Google Chrome.

During the analysis, it was also noticed that the governance of the analyzed projects differs strongly. There are projects like quic-go, quiche, or s2n-quic which are developed in public with a large community and a lot of searchable information on Github. On the contrary, there are projects like mvfst, XQUIC, or google/quiche where the code is publicly available but development seems to happen behind closed doors. This is demonstrated by the fact that there is little public information or discussion about design decisions available online. The mvfst and google/quiche projects use commit messages that carry extra information for internal use by Facebook or Google respectively. In the XQUIC project, there are very few commits in the repository (currently 78), but major features with plus 12k lines of code changes were merged without public review[11]. Additionally, there are unanswered issues about testing[12] and concerns about easy to guess address tokens[13]. Since the source code is used primarily for collecting information, such projects are very difficult to analyze using the chosen approach. Nevertheless, multiple implementations help testing the robustness, standards compliance, and interoperability.

In order to improve the security, sustainability and maintainability of the QUIC ecosystem as a whole, the authors suggest creating a test suite (or extend an existing one, cf. [20]) which covers security consideration tests at runtime. At best, passing these tests should become a requirement to identify as a compliant QUIC implementation. Good examples for existing test suites in different working areas are xfstests[14] for filesystems and litmus[15] for WebDAV. For tracking the standards compliance, the approach of Amazon is a good example and is worth including into the proposed test suite. Furthermore, the authors suggest proposing QUIC for inclusion in standard libraries or even operating system kernels. Being part of a larger software project ensures long term maintenance and simplifies usage and research. Both approaches seem to be a good and sustainable solution that is already being worked on[16] [30].

Despite the fact that the ecosystem is fragmented by the existence of many different but inter-operable QUIC implementations, an open question remains: "Does the lack of implemented security considerations have an impact on the operational security of QUIC?" A future study examining QUIC projects at runtime for actual vulnerabilities caused by the lack of particular security considerations and possible obstacles in the development might be worthwhile.

7 CONCLUSION

This research paper confirmed the results of previous studies associated with the QUIC ecosystem. From a security point of view, the QUIC ecosystem is fragmented and no mature general purpose implementation is available. However, there are well maintained implementations available that are used in production software, for instance quic-go is used in the Caddy web server[17]. Big players tend to create their own implementations and ship them with their software rather than using and contributing to already available implementations. For inexplicable reasons these software stacks are available as Open Source software, but the development happens behind closed doors.

Answering the formulated research question: "Does the complexity of the RFC standard on the one hand and the large number of different QUIC implementations on the other hand lead to an insecure QUIC ecosystem?", the authors would answer this question with an emphatic *yes*. The complex RFC standard with a lot of optional items and the lack of QUIC being available in major software stacks led to a large number of different implementations that addressed the same problems differently. The study confirmed

[9] https://github.com/ngtcp2/ngtcp2/issues/626
[10] https://github.com/mozilla/neqo/issues/784
[11] https://github.com/alibaba/XQUIC/pull/287
[12] https://github.com/alibaba/XQUIC/issues/265
[13] https://github.com/alibaba/XQUIC/issues/266
[14] https://git.kernel.org/pub/scm/fs/xfs/xfstests-dev.git/about/
[15] http://webdav.org/neon/litmus/
[16] https://github.com/golang/go/issues/44886
[17] https://caddyserver.com

that a lot of known and documented security considerations are not addressed by most QUIC implementations. Reducing the number of available QUIC implementations and including those into major software stacks like standard libraries or operating system kernels might help to improve the ecosystem.

8 DATA AVAILABILITY

We provide supplementary material under the CC0 ("No Rights Reserved") license at https://rumpelsepp.org/projects/quic-overview. Among others, the material provides more detailed versions, also including timestamps and weblinks, of Table 1 and Table 2.

ACKNOWLEDGMENTS

Many thanks to Daniel and Florian from the segfault.fm podcast (https://segfault.fm) for inspiring this paper in episode 0x1a. The authors also would like to thank Michael Heinl for providing his Overleaf account for writing this paper. This work was partially supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 16KIS1847 and partially by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) under Grant No. 13I40V010A.

REFERENCES

[1] 1980. User Datagram Protocol. RFC 768. https://doi.org/10.17487/RFC0768

[2] Florian Adamsky, Syed Ali Khayam, Rudolf Jäger, and Muttukrishnan Rajarajan. 2012. Security Analysis of the Micro Transport Protocol with a Misbehaving Receiver. In *2012 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*. 143–150. https://doi.org/10.1109/CyberC.2012.31

[3] Richard J Aldrich and Athina Karatzogianni. 2020. Postdigital war beneath the sea? The Stack's underwater cable insecurity. *Digital War* 1, 1 (2020), 29–35. https://doi.org/10.1057/s42984-020-00014-x

[4] Mike Bishop. 2022. HTTP/3. RFC 9114. https://doi.org/10.17487/RFC9114

[5] Konstantin Böttinger, Dieter Schuster, and Claudia Eckert. 2015. Detecting Fingerprinted Data in TLS Traffic. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security* (Singapore, Republic of Singapore) *(ASIA CCS '15)*. Association for Computing Machinery, New York, NY, USA, 633–638. https://doi.org/10.1145/2714576.2714595

[6] Scott O. Bradner. 1997. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119. https://doi.org/10.17487/RFC2119

[7] Efstratios Chatzoglou, Vasileios Kouliaridis, Georgios Karopoulos, and Georgios Kambourakis. 2023. Revisiting QUIC attacks: a comprehensive review on QUIC security and a hands-on study", journal="International Journal of Information Security. 22, 2 (01 4 2023), 347–365. https://doi.org/10.1007/s10207-022-00630-6

[8] Shan Chen, Samuel Jero, Matthew Jagielski, Alexandra Boldyreva, and Cristina Nita-Rotaru. 2021. Secure Communication Channel Establishment: TLS 1.3 (over TCP Fast Open) versus QUIC. *Journal of Cryptology* 34, 3 (24 May 2021), 26. https://doi.org/10.1007/s00145-021-09389-w

[9] Xavier de Carné de Carnavalet and Paul C. van Oorschot. 2023. A Survey and Analysis of TLS Interception Mechanisms and Motivations. *ACM Comput. Surv.* (1 2023). https://doi.org/10.1145/3580522

[10] Wesley Eddy. 2022. Transmission Control Protocol (TCP). RFC 9293. https://doi.org/10.17487/RFC9293

[11] Sally Floyd, Dr. K. K. Ramakrishnan, and David L. Black. 2001. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168. https://doi.org/10.17487/RFC3168

[12] Christian Huitema, Sara Dickinson, and Allison Mankin. 2022. DNS over Dedicated QUIC Connections. RFC 9250. https://doi.org/10.17487/RFC9250

[13] Jana Iyengar and Ian Swett. 2021. QUIC Loss Detection and Congestion Control. RFC 9002. https://doi.org/10.17487/RFC9002

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

[15] Mirja Kühlewind and Brian Trammell. 2022. Applicability of the QUIC Transport Protocol. RFC 9308. https://doi.org/10.17487/RFC9308

[16] Mirja Kühlewind and Brian Trammell. 2022. Manageability of the QUIC Transport Protocol. RFC 9312. https://doi.org/10.17487/RFC9312

[17] Robin Marx, Joris Herbots, Wim Lamotte, and Peter Quax. 2020. Same Standards, Different Decisions: A Study of QUIC and HTTP/3 Implementation Diversity. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC* (Virtual Event, USA) *(EPIQ '20)*. Association for Computing Machinery, New York, NY, USA, 14–20. https://doi.org/10.1145/3405796.3405828

[18] Robin Marx, Wim Lamotte, Jonas Reynders, Kevin Pittevils, and Peter Quax. 2018. Towards QUIC Debuggability. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC* (Heraklion, Greece) *(EPIQ'18)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3284850.3284851

[19] Robin Marx, Luca Niccolini, Marten Seemann, and Lucas Pardue. 2023. *Main logging schema for qlog*. Internet-Draft draft-ietf-quic-qlog-main-schema-05. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-quic-qlog-main-schema/05/ Work in Progress.

[20] Maxime Piraux, Quentin De Coninck, and Olivier Bonaventure. 2018. Observing the Evolution of QUIC Implementations. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC* (Heraklion, Greece) *(EPIQ'18)*. Association for Computing Machinery, New York, NY, USA, 8–14. https://doi.org/10.1145/3284850.3284852

[21] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. https://doi.org/10.17487/RFC8446

[22] Eric Rescorla and Brian Korver. 2003. Guidelines for Writing RFC Text on Security Considerations. RFC 3552. https://doi.org/10.17487/RFC3552

[23] David Schinazi and Eric Rescorla. 2022. *Compatible Version Negotiation for QUIC*. Internet-Draft draft-ietf-quic-version-negotiation-14. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-quic-version-negotiation/14/ Work in Progress.

[24] Rob Sherwood, Bobby Bhattacharjee, and Ryan Braud. 2005. Misbehaving TCP Receivers Can Cause Internet-Wide Congestion Collapse. In *Proceedings of the 12th ACM Conference on Computer and Communications Security* (Alexandria, VA, USA) *(CCS '05)*. Association for Computing Machinery, New York, NY, USA, 383–392. https://doi.org/10.1145/1102120.1102170

[25] Statista. 2023. Number of internet users worldwide from 2005 to 2022. https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/

[26] Randall R. Stewart, Mitesh Dalal, and Anantha Ramaiah. 2010. Improving TCP's Robustness to Blind In-Window Attacks. RFC 5961. https://doi.org/10.17487/RFC5961

[27] Martin Thomson. 2021. Version-Independent Properties of QUIC. RFC 8999. https://doi.org/10.17487/RFC8999

[28] Martin Thomson and Cory Benfield. 2022. HTTP/2. RFC 9113. https://doi.org/10.17487/RFC9113

[29] Martin Thomson and Sean Turner. 2021. Using TLS to Secure QUIC. RFC 9001. https://doi.org/10.17487/RFC9001

[30] Peng Wang, Carmine Bianco, Janne Riihijärvi, and Marina Petrova. 2018. Implementation and Performance Evaluation of the QUIC Protocol in Linux Kernel. In *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems* (Montreal, QC, Canada) *(MSWIM '18)*. Association for Computing Machinery, New York, NY, USA, 227–234. https://doi.org/10.1145/3242102.3242106