An Empirical Approach to Evaluate the Resilience of QUIC Protocol against Handshake Flood Attacks

Benjamin Teyssier, Y A Joarder, Carol Fung
Concordia Institute for Information
Systems Engineering (CIISE)
Concordia University
Montreal, Canada
Email: {benjamin.teyssier, ya.joarder, carol.fung}@concordia.ca

***Abstract*—QUIC is a new transport protocol aiming to enhance web connection performance and security. It was gaining popularity quickly in recent years and has been adopted by a number of prominent tech companies, including Facebook, Amazon, and Google. However, the resilience of QUIC Protocol against various cyber attacks has not been fully tested yet. In this paper, we investigate the resilience of QUIC Protocol against handshake flood attacks. We conducted comprehensive experiments to evaluate the resource consumptions of both the attacker and the target during incomplete handshake attacks, including CPU, memory, and bandwidth. The DDoS amplification factor was measured and analyzed based on the results. We compared the results against TCP Syn Cookies under Syn flood attacks. We show that the QUIC Protocol design has a much larger DDoS amplification factor compared to the TCP Syn Cookies, which means QUIC is more vulnerable to handshake DDoS attacks. Also, the CPU resource of QUIC servers is most likely the bottleneck during the handshake flood attacks. To the best of our knowledge, this is the first study to thoroughly investigate resilience of QUIC to handshake DDoS attacks.**

***Index Terms*—QUIC, TCP, DDoS, Amplification Factor, Syn Cookies, Syn Flood Attacks**

I. Introduction

The emergence of the QUIC protocol represents a significant milestone in Internet communication, which offers enhanced performance and reduced latency compared to the traditional TCP protocol [1]. Its capability of facilitating secure and efficient data transmission across unreliable networks has attracted much attention from both academia and industry [2]. However, as with any innovative technology, it is imperative to scrutinize its potential vulnerabilities. One such area that deserves rigorous investigation is the resilience against handshake flood attacks [3]. The handshake flood attack is a type of Denial-of-Service (DoS) attack, which poses a significant threat to network infrastructure [4]. The handshake flood attack exploits the handshake mechanism inherent in many protocols, enabling attackers to amplify their traffic and potentially trigger service outages and resource exhaustion [5]. While the implications of handshake flood attacks have been extensively studied in protocols such as the Domain Name System (DNS) [6] and Network Time Protocol (NTP) [7], a comprehensive understanding of their impact within the QUIC protocol remains absent in the current body of literature. Given the growing use of QUIC including many well-known tech companies such as Google and Amazon, this knowledge gap is alarming. Our paper addresses this critical gap in our understanding by providing a practical analysis of QUIC's susceptibility to handshake flood attacks. To address the critical need for a thorough security assessment of the QUIC protocol, we experimentally explored QUIC's resilience against handshake flood attacks.

We constructed a realistic testbed with a QUIC server and a QUIC attacker client that generates spoofed handshaking requests. We conducted exhaustive experiments to measure the resource consumption of both the attacker and the targeted server during incomplete handshake attacks, containing CPU, memory, and bandwidth. Our results reveal that the QUIC protocol design has a significantly larger DDoS amplification factor compared to TCP Syn Cookies [8], indicating a higher vulnerability to handshake DDoS attacks. In contrast to previous studies [9]–[12], we have identified the CPU resource of QUIC servers as the most likely bottleneck during handshake flood attacks. Our findings will provide valuable insights for future QUIC protocol development.

The major contributions of this work can be summarized as following: 1) We are the first to investigate the QUIC protocol resource consumption during handshake flood attack; 2) We compared the resource consumption of QUIC protocol with TCP Syn Cookies under handshake flood attacks; 3) We measured the DDoS amplification factors of QUIC protocol and revealed a critical vulnerability of the QUIC protocol design. The remainder of this paper is organized as follows: Section II delves into the existing body of research pertinent to our study. After that, we discuss QUIC and TCP handshake process, packet structure and others in section III. Subsequently, Section IV presents the empirical findings of our investigation, focusing on aspects such as CPU and Memory Usage. We then discuss the broader implications of our research and its contributions to the field in Section V. Finally, we conclude our discourse with concluding remarks in Section VI.

978-3-903176-59-1 ©2023 IFIP

II. Related Works

QUIC has emerged as a promising alternative to TCP, gaining significant attention due to its design for low latency, security, and reliability over UDP [3], [13]. The protocol's design emphasizes a quick initial connection setup, mitigating state exhaustion and amplification-related attacks [5], [10], [14]. In this section, we briefly go over some related works including QUIC handshakes attacks and mitigation techniques.

A. QUIC Protocol Handshake Attacks and Mitigation

The QUIC handshake incorporates Transport Layer Security (TLS) within its protocol handshake, which begins with a client sending an Initial, and the server responding with its own Initial and a Handshake packet, all of which can be coalesced into a single UDP datagram [3]. To protect against state exhaustion, servers can utilize RETRY packets, although this practice is not commonly implemented [3]. To mitigate the risk of handshake flood attacks, QUIC limits the server's response data until the client's IP address is verified [15]. As specified by RFC 9000, the server can only send three times the data bytes received from the client's Initial, inclusive of padding and resent bytes [13]. This three-fold data limit is minimal compared to the amplification potential of other protocols [5], [14]. In addition, Nawrocki et al. [3] introduce QUICsand, a novel congestion control algorithm for network traffic management. The algorithm dynamically adjusts the pacing rate and congestion window size to optimize latency and throughput. Their experimental results demonstrate QUICsand's superior performance over other congestion control algorithms across different network scenarios.

B. QUIC Performance and Potential Security issues

Despite these security measures, QUIC has demonstrated excellent performance and even surpassed TCP in some cases [16]–[19]. However, security compromises have been made to enhance latency [20], and handshake latency may be an issue if there's a version disagreement between the client and server [21]. QUIC incorporates TLS 1.3 to ensure authenticated confidentiality and integrity [22], [23]. The TLS 1.3 handshake commences with a client relaying its supported cipher suites, key parameters, and other metadata via the first Initial message. The server reciprocates with its parameters and an X.509 certificate to authenticate its identity [23], [24]. Deployment studies have primarily focused on QUIC service availability. The protocol's adoption started before its standardization [25]–[27], with major tech companies leading the charge [28], [29]. However, DNS over QUIC has seen lacklustre adoption and weak handshakes due to large certificates if Session Resumption is not employed [6], [30].

This paper is the first to systematically study the influence of TLS certificates on QUIC performance, drawing upon comprehensive measurements. Furthermore, it is also the first to investigate the resilience of QUIC Protocol against handshake flood attacks, providing a comprehensive analysis of the resource consumption of both the attacker and the target during incomplete handshake attacks, including CPU, memory, and bandwidth.

III. Handshake Processes of QUIC and TCP

This section provides an overview of the handshake processes for TCP and QUIC protocols. Knowing these handshake processes is crucial for understanding their vulnerabilities to handshake flood attacks. We start with an introduction of QUIC handshake process, followed by TCP handshaking process. A comparison of the QUIC and TCP handshake procedures is shown in Figure 1.

A. QUIC Handshake Process

The QUIC protocol, designed to enhance web connection performance and security, combines the transport and TLS 1.3 handshakes into a single step, thereby reducing latency [13]. As shown in Figure 1 (a), the QUIC handshake begins with the client sending an Initial packet containing the *Client's Hello* message. The server responds with its Initial packet, which includes the *Server Hello* message and other necessary information.

Following this, the client sends a client finished packet to indicate it has completed the handshake. Note that the first data packet is sent out at the same time to save the roundtrip time. The server responds with its first data response packet to confirm the validity of the handshake process. Note that, only one additional round trip time is needed for the handshake process. That is the reason it is called QUIC 1-RTT handshake process [23].

It is worth noting that QUIC facilitates an even faster connection establishment with a 0-RTT (zero Round Trip Time) mechanism. This feature leverages the client's previous communication with the same server, allowing the client to send data to the server in the first round trip of the protocol handshake. This is accomplished by the client storing some information about the server after the first connection, which can then be used to establish subsequent connections without needing a full handshake.

As shown in Figure 2 (a), a QUIC packet structure comprises a UDP packet header and a QUIC packet header containing various fields. The UDP packet header includes the source port, destination port, checksum, and length. The QUIC packet header contains flags, a connection ID, and a packet number, all encrypted for security. The QUIC payload, which is also encrypted, contains a flow control frame, a stream frame header, an ACK frame, and a stream frame payload. The QUIC Packet has a simpler structure, with a connection ID (0 to 20 bytes), packet number (1 / 2 or 4 bytes), version (4 bytes), and payload (variable size). The connection ID identifies the connection, the packet number orders the packets, and the version indicates the version of the QUIC protocol used. The payload carries the actual data, which is organized into frames. Importantly, it is important to note that the packet structures can vary based on the specific protocols versions and the

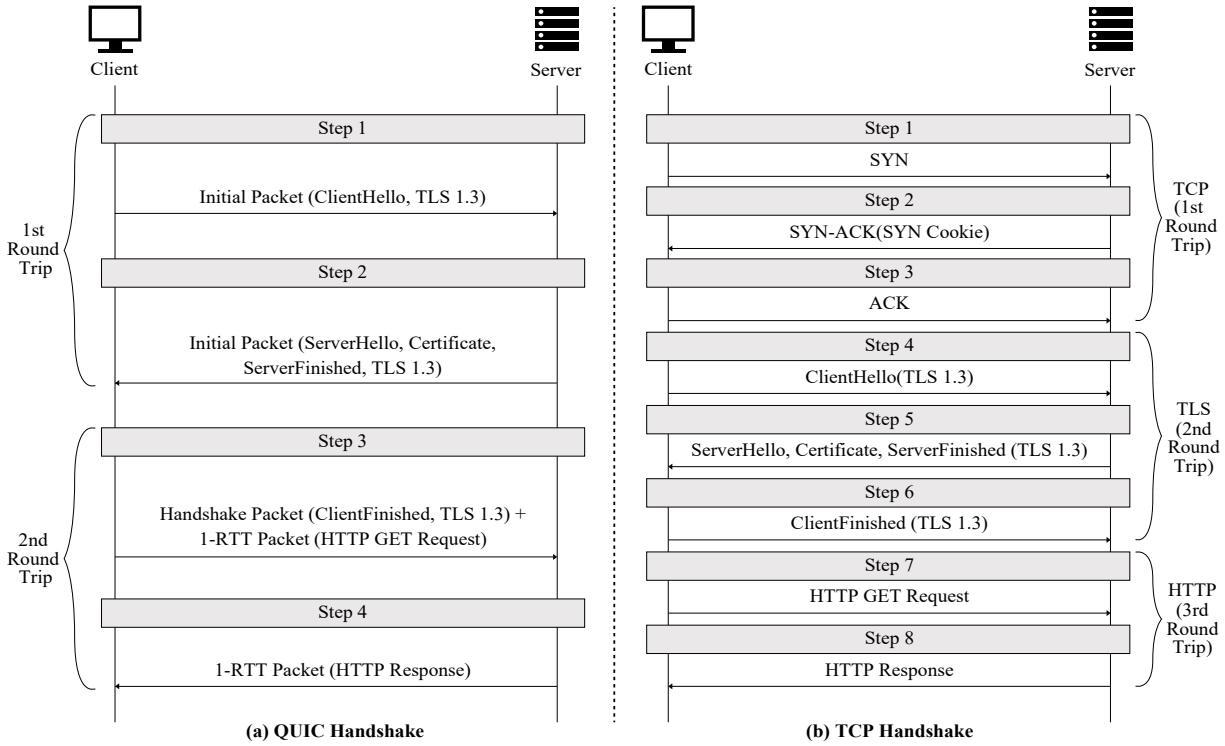

Fig. 1: Overall Handshaking Process of (a) QUIC and (b) TCP

connections' configurations. Two types of structures of QUIC packets are given below:

1) QUIC ClientHello Packet Structure: The *ClientHello* packet contains several fields, including Version, Connection ID, Length, Packet Number, and the Payload containing the client's public key for key exchange, and the HTTP/2 SETTINGS frame for the connection. The *client hello* packet is typically larger as it includes a list of cipher suites and extensions that the client supports and can be around 200-300 bytes.

2) QUIC ServerHello Packet Structure: The *ServerHello* packet contains Version, Connection ID, Length, Packet Number, and the Payload containing the server's public key for key exchange, and the HTTP/2 SETTINGS frame that outlines the server's preferences and constraints for the connection. The*ServerHello* packet is generally smaller than *ClientHello*, as it only needs to include the chosen cipher suite and compression method, and may be approximately 100-200 bytes.

B. TCP Handshake Process

As shown in Figure 1 (b), The TCP handshake, on the other hand, is a three-step process. The client sends a SYN packet to the server to initiate the connection. The server responds with a SYN-ACK packet, acknowledging the request. The client then sends an ACK packet to confirm the connection. This is followed by multiple rounds of communication for TLS and HTTP requests, which can add to the connection's latency.

- **SYN**: It is the first step in establishing a reliable connection between the client and the server. The SYN packet has a randomly generated sequence number (S_c), the SYN flag turned to 1, and no payload.
- **SYN-ACK**: Upon receiving the SYN packet, the server responds with a SYN-ACK packet. This packet serves two purposes. Firstly, it acknowledges the receipt of the SYN packet from the client. It authenticates itself by setting the acknowledge number equivalent to the sequence number of the SYN packet ($A_s = S_c$). Secondly, it carries a new Sequence Number S_s. The server generates S_s randomly in a stateful handshake, and stores the number for the verification of the subsequent ACK packet. However, under SYN flood attacks, the server's backlog fills up quickly and cannot accept more SYN requests, which leads to the denial of service.

  To address this issue, the server can use a stateless handshake solution called *SYN Cookies* [31], where the new sequence number S_s is specially generated based on the client's IP address, port number, and other information. It enables the server to verify the legitimacy of the final ACK in the handshake without storing every sequence number it generates. This mechanism does not require any backlog on the server and therefore enhances its resilience against TCP SYN flood attacks.
- **ACK**: After receiving the SYN-ACK packet from the server, the client sends back an ACK packet. This packet acknowledges the receipt of the SYN-ACK packet from

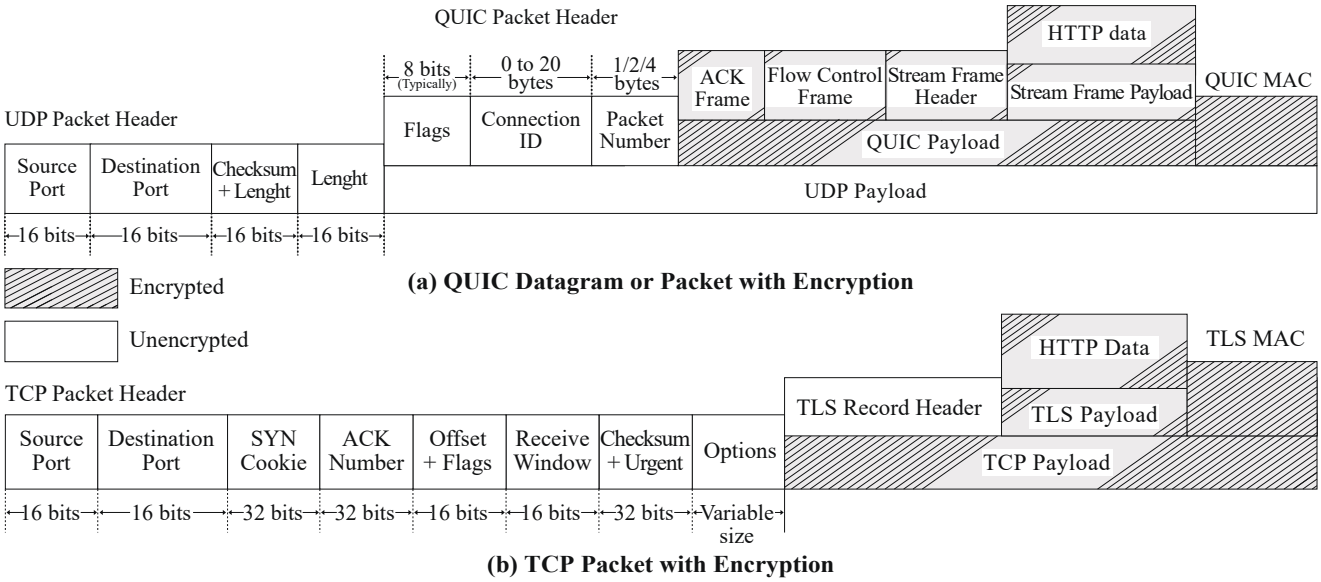

Fig. 2: Illustration of the packet structures, highlighting the size and Encrypted areas

the server by sending back the received sequence number as an acknowledgment number $A_c = S_s$. The server verifies the validity of the ACK packet by comparing the received acknowledgment number with the expected number. A TCP connection is established when the ACK packet passes the verification process.

As shown in Figure 2 (b), A TCP packet structure consists of a TCP packet header and a TCP payload. The TCP packet header includes the source port, destination port, sequence number, ACK number, receive window length, flags, checksum, and the option field. The TCP payload contains encrypted HTTP data, a TLS MAC, and a TLS payload.

1) TCP SYN Packet Structure: A TCP SYN packet typically includes Source Port, Destination Port, Sequence Number, Acknowledgment Number, offset, flags (with the SYN flag set to 1), Receive Window, Checksum and Urgent Pointer. A TCP SYN packet is typically 40 bytes long.

2) TCP SYN-ACK Packet Structure: A TCP SYN ACK packet typically includes Source Port, Destination Port, Sequence Number (SYN Cookie), Acknowledgment Number, offset, flags (with the SYN flag and ACK flag set to 1), Receive Window, Checksum and Urgent Pointer. A TCP SYN ACK packet is typically 40 bytes long.

3) Types of TCP Handshaking: In TCP, there are two types of handshaking: stateful and stateless TCP handshaking. Stateful and stateless handshakes refer to whether or not a server maintains state information during the TCP handshake process. In a stateful TCP handshake, the server retains information about each client's connection requests during the handshake process. For instance, after receiving a SYN packet from a client, the server keeps a record of the pending connection before it receives the final ACK packet. This record, known as a Transmission Control Block (TCB), includes the client's IP address and port number, the server's sequence number, and more. The TCB allows the server to keep track of each client's connection status. However, maintaining this state information for every connection request requires significant memory resources, especially for servers handling an important number of connections.

On the other hand, in a stateless TCP handshake, the server does not keep a record of each client's connection requests. Rather, the server uses mechanisms such as SYN Cookies to manage connections. A SYN Cookie, included in the SYN-ACK packet sent by the server, is a cryptographic token derived from the client's IP address, port number, and other data. When the server receives the final ACK packet from the client, it uses this SYN Cookie to verify the legitimacy of the client's connection request. This stateless approach mitigates the risk of SYN flood attacks, in which an attacker overwhelms the server with many SYN packets but never completes the handshake. The stateless TCP handshake significantly reduces the memory resources required on the server, enhancing its ability to handle a high volume of connections and resist DoS attacks.

IV. Evaluation

In this section, we evaluate the resilience of QUIC protocol against handshake flood attacks. We first present our experimental design, followed by the evaluation results on CPU, memory, and bandwidth.

A. Experimental setup

The experiments were performed between two basic Linux devices, one acting as a client, the other acting as the server

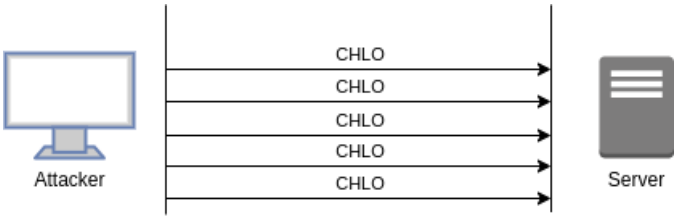

Fig. 3: Experimental Setup

as shown on Figure 3. One device was a Raspberry PI Model 3B having 1GB of RAM available and a Quad Core 1.2GHz CPU, the other was a laptop with 16GB of RAM and Intel i7 1.8GHz CPU. The roles of server and client were reversed between every measurement to ensure no dependence on the hardware for our results. Following the same idea, the base resource usage of processes present on the machine should not influence the measure. To that effect, the measures were conducted on the process running the server and client to isolate the resource consumption.

The goal of the experiments is to evaluate the resilience of QUIC Protocol against handshaking flooding attacks. We compare its resource consumption under attack with another protocol. TCP being the most popular transport protocol, it is interesting to use it as a comparison. However, to ensure a fair comparison in terms of resilience and resource usage, it was necessary to deploy the stateless version : TCP SYN Cookies [31]. Then we estimated the amplification factor of both protocols to assess the lever an attacker has on its victim. The different resources monitored were the Memory Usage, CPU Usage and Bandwidth. Note that amplification factor is the ratio between the resource consumption of the server to process the packets and the needs of the attacker to craft those. The version of QUIC used is *aioquic* [32], it is implemented in Python. The attack code is a modified version of *aioquic* itself. It only contains the code to create a packet to be as lightweight as possible, the part leading the attack was written from scratch. The TCP SYN Cookie code is implemented in Python to have a more relevant comparison. It was written from scratch but following the guidelines given in TCP SYN Cookies Linux implementation.

B. CPU Usage

Due to the stateless characteristic of both QUIC and TCP SYN Cookie, the server does not store the status of each connection request. Therefore the storage need is not impacted by the attack volume. However, in order to verifiy the legitimacy of the incoming connection initialization requests, both QUIC and TCP servers need to do some calculations as specified in Section III. To measure the CPU consumption of QUIC server, we increase the number of hello-client packets from 1 packet/second to 50 packet/second and observe the CPU usage on the server side. The CPU usage over time was also estimated to monitor the behavior of the server during an attack. The CPU consumption was obtained with Linux built-in tools *ps* to measure the resources used by a specific process.

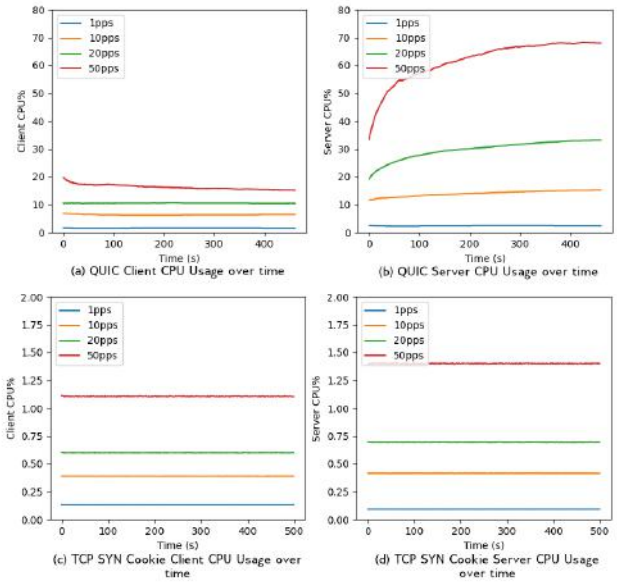

Fig. 4: CPU Usage over time

The experiment was conducted multiple times and the result plotted is the mean of all the values. It also contains the 95% confidence interval to show the variability from one row of experiment to the other.

1) Usage over time: We observed the resource consumption and the server behavior over time during handshake attacks. This experiment was conducted to observe the CPU Usage over time during attacks at different rates. An attack was started at different rates and the CPU percentage was taken every second. The experiment was performed 3 times with the different rates to ensure a smooth output. However, it was pretty stable and the results were very close from one row of experiment to the other.

On the QUIC server-side of Figure 4 (b), we can observe the increase of CPU shows inertia before stabilizing. It also appears that higher rates create more inertia than lower one. On the QUIC client-side, as shown in Figure 4 (a), a slight decrease happens in the first seconds. Then the CPU usage is quickly stabilized to its final value which is also close to the one found previously. The inertia is caused by the queue of packets. At first the server proceeds to the verifications needed to process the packet. It checks the different values of the flags corresponding to the parameters of the connection inside the header. It also needs to verify if the address was validated. Then it starts preparing the *Server Hello* [13] packet. During the first minutes it falls behind, it is the time for the server to allocate more resources and process the different packets. This inertia can be a double-edged characteristic for the system. It gives more time to a potential DDoS Detection system to notice the attack and to potentially prevent further harm. However, this stack filling up also means that potential clients might encounter additional delay during this period of time.

The experiment was performed under the same conditions for

TCP SYN Cookie. The resource consumption is instantly the one it stabilizes to. Both on the server-side and the client-side, we can observe the same values around the beginning of the experiment as in Figure 4 (c)(d). The CPU usage is low compared to the one obtained with QUIC.
This stability is mostly explained by the low resource consumption. It never goes above 1.5% of the total device capacity. The performance of this protocol is obviously a benefit and a key feature of the SYN Cookies.

In the rest of the experiment in this paper, we use the stablized resource usage values to represent the usage on each data rate.

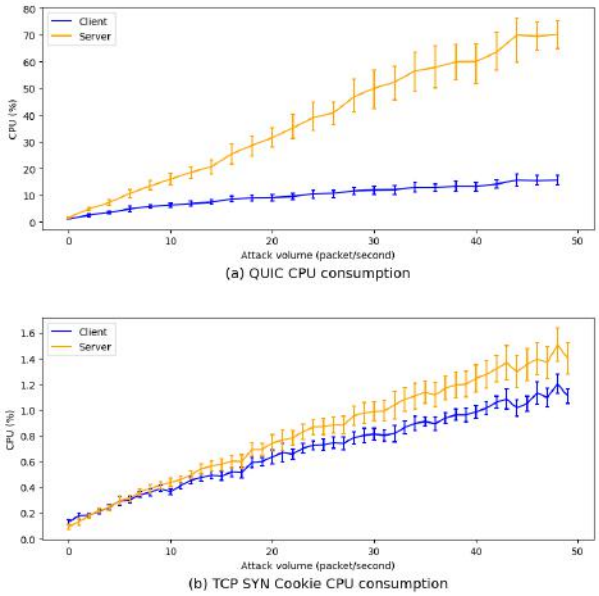

Fig. 5: QUIC and TCP SYN Cookie CPU Usage

2) CPU Consumption: Figure 5 (a) shows the average and confidence interval of additional CPU usage of the QUIC server and attacker client when the attacker increases the attack volume. The x-axis corresponds to the attack rate in packets per second. We can see that under the QUIC handshake flood attack, the CPU consumption for both the client and server increase almost linearly with the attack volume. The server's CPU usage reaches its capacity when the attack volume is close to 50 packet/second. Note that 50 packets per seconds is not the rate limit of the DDoS attack volume since attacker still has a lot of room to increase its attack volume. However, 50 packets was the limit of our server as it reaches 73% of CPU usage (almost 100% with the base usage). Higher values are not relevant as the overloaded server causes noise in the measures. The experiment shows that during an QUIC-Flooding attack, CPU is the bottleneck. During the QUIC handshake process, the client uses much less CPU resource than the server.
Figure 5(b) was obtained by leading the same experiment as Figure 5 (a) with the TCP SYN Cookie protocol. First of all, it appears that the CPU consumption of sending/handling the same number of TCP handshake requests is significantly lower than QUIC both for server and client. For example,

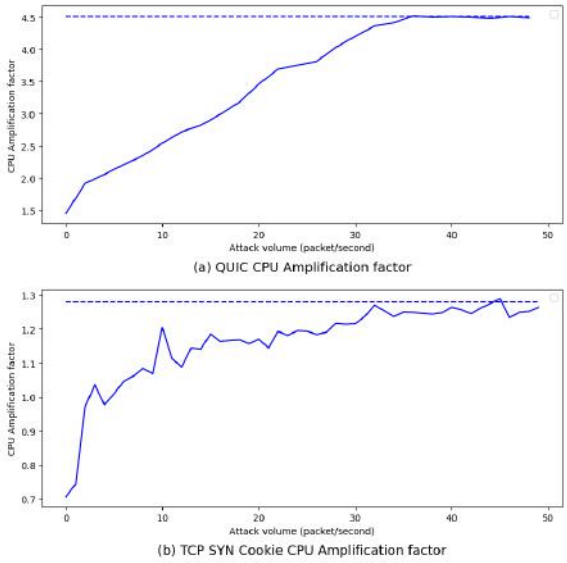

Fig. 6: QUIC and TCP SYN Cookie CPU Amplification Factor

the TCP server only consumes less than 1% CPU resource compared to the 30% consumed by QUIC server when attack volume is 20 packet/second. This is mainly due to the design complexity of QUIC packets and verifications compared to TCP SYN Cookies. Actually TCP SYN Cookie is very similar to usual TCP where the connection ID is just a verifiable hash instead of being a fixed number stored on the server. On the other hand, QUIC packets have a lot of fields that are used for verification purposes. The size of the packets and the verifications themselves are costful in terms of resource consumption.

3) Amplification Factor: This experiment highlights the CPU amplification factor for both QUIC and TCP SYN Cookie. The attacker crafts packets in the most economical way to ensure an optimal amplification. He does not wait for the server responses and does not process those, he also does not close the handshake as it would be useless resource usage.
The QUIC amplification factor plotted on Figure 6 (a) seems to stabilize around 4.6 meaning that for every unit of computation needed by the attacker, the server will deploy 4.6 times more resources to process the packets. This high amplification factor can be explained by multiple aspect of QUIC design. First the server has to validate the parameters provided by the client. If not, the server might send a Retry Packet. Then the server has to craft the Handshake packet depending on the information contained in the header of the client's one. An amplification factor over 4 is potentially a serious vulnerability. With the adequat infrastructure, an attacker can take down systems with more than 4 times its computing power reducing drastically the complexity and the cost of an attack. The amplification factor of TCP SYN Cookie presented in Figure 6 (b) is notably

lower, it stabilizes around 1.28 meaning the server has to do 28% more work than the client to process the packet. It is mainly due to the design of the protocol. During the handshake, the server only has to compute a hash based on informations from the incoming packet and the time when it received it. This hash will be the cookie sent back to the client to identify the connection without storing any connection ID. Siphash [33] is the pseudorandom function used to hash these informations. It was designed specifically for TCP SYN Cookie with performance in mind to make sure the server is able to process the incoming packets without overloading the computation resources.
This low amplification reduces the lever an attacker has on its victim. It means an attacker must have at least 80% of its victim resources to ensure the sucess of its attack.

C. Memory Usage

The second resource we measure is the memory usage. During this experiment, we applied the same conditions as in the measure of CPU Amplification and CPU usage over time. The *ps* tool used to get the CPU also allows to monitor memory so the same protocols were followed to optimize relevance of the results and enhance comparability.

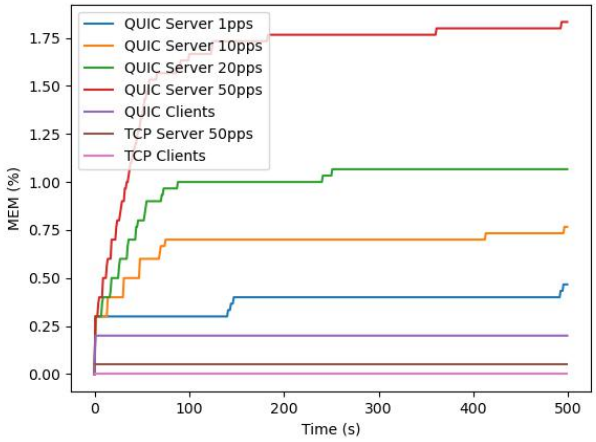

Fig. 7: Memory usage over time

1) Usage over time: Even if it was established that memory would not be the bottleneck when the system reaches stability, it is important to study memory over time to make sure there is no behavior that could be a vulnerability for the server. For instance, an important memory peak caused by the attacker in a short amount of time before reaching stability could be harmful to the system.
We can observe that the memory follows the behavior of the CPU for the QUIC protocol. There is inertia in the first seconds but the memory usage increases before reaching the stabilized value. There is no resource consumption peak at the beginning of the attack. TCP SYN Cookie was also plotted on this graph as a comparison but memory usage is so low the measure were not accurate on the client-side and on the server-side for rates

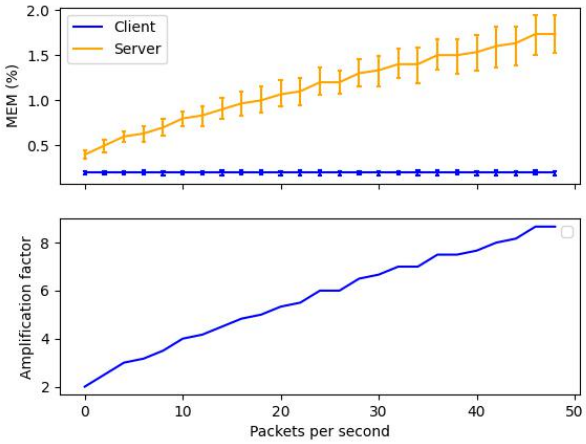

Fig. 8: QUIC Memory Usage and Amplification Factor

under 50pps. The highest usage reached is 0.06% of the device total capacity. The small jumps in the graphs can be explained by the very low usage and the accuracy of the tool when the memory usage is below 1%.
These results confirms the fact that memory is less likely the bottleneck under handshake flood attacks. It also highlights how low the memory usage for TCP SYN Cookie is.
2) Memory Consumption: As shown in Figure 8, on the QUIC client-side, memory usage remains stable when the attack rate increases. The confidence interval is very tight meaning the results obtained during the different rounds of the experiment provided very similar results. The value is low around 0.1% of the device total capacity. On the server-side, there is an increase in memory usage along with the attack rate. The confidence interval is also close to the curve meaning the impact of this type of attack on resource is relatively predictable. The highest value reached is below 2% which is low compared to the CPU usage for a similar attack rate. Compared to CPU usage, the memory usage of QUIC server is much lower and CPU is much more likely to be exhausted first under handshake flood attack.
3) Memory usage and Amplification Factor: As shown in Figure 8, the memory amplification factor increases along with the attack rate following the drift of the server memory usage reaching 8.6. It is worth noting that the amplification factor may keep on increasing with the attack rate. However, since the CPU resource on the QUIC server has already reached its capacity, a larger attack rate is not meaningful.
We can see that the server memory usage increases linearly with the number of packets because of the asynchronous functions used to handle multiple packets and connections at a time. However once the packets are processed the memory usage falls down to 0 as everything is dumped. Even if the amplification reaches high values, these metrics are not very relevant considering the fact that CPU is a bottleneck before memory could be an issue. In addition to that, it is easy to add memory to an infrastructure but adding CPU is more difficult

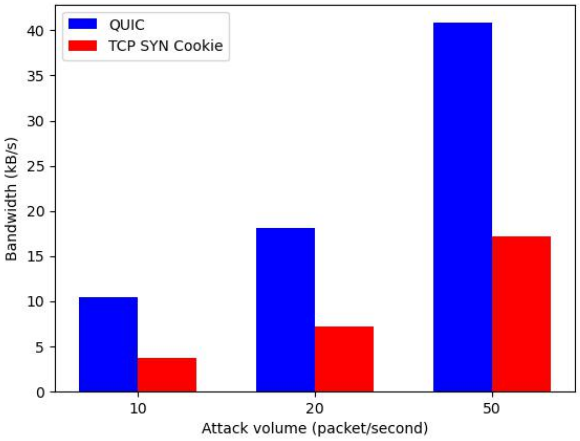

Fig. 9: Bandwidth usage comparison

and certainly more costful.

D. Bandwidth

TCP SYN Flooding could be a threat for the stateful TCP servers in terms of memory because it stores the Sequence Number to verify connections. However, it can also be the target of attacks on the bandwidth. This experiment was conducted to monitor the bandwidth usage under flooding attacks with QUIC and TCP SYN Cookie. As shown in Figure 9, for both protocols, the incoming and outgoing traffic are close. This is mainly due to the design of both protocols to ensure a low bandwidth amplification factor with symetrical sized packets.

Following the previous experiment protocols, we isolated the bandwidth of the process from the base usage. We can observe a more significant bandwidth consumption for QUIC than for TCP SYN Cookies. This can be explained by the numerous fields in a QUIC packet. As explained in Section III, the QUIC Handshake contains the TLS one. The key exchange is done during the 1-RTT handshake with the server certificate embedded in the *Server Hello* message.

The symmetrical aspect of these designs ensure a bandwidth amplification factor of 1. The client sends a packet with some padding and the server has to send the exact same amount of data by padding the packet as much as necessary. This feature slightly affects performance but significantly reduces the vulnerabilities regarding bandwidth.

V. Discussion

In our experiment, our CPU was the resource that was drained under QUIC Flooding Attack. It is intuitive as the server needs a lot of computations to handle the packets according to the process described in RFC9000 [13]. It could cause complete service disruption as the server would not be able to process any other incoming packet and would not have enough resources available to run its other processes. The memory also presents a high amplification factor but the usage is much lower so that it is unlikely a threat for most of the systems. Moreover, adding more CPU typically costs more than adding memory. The bandwidth use is not a problem with these stateless protocols as they were design with security in mind to ensure neutral amplification factor. In addition to that, the stateless design of the QUIC protocol avoids the full backlog problem encountered during the TCP SYN flood attack to stateful TCP servers.

The 0-RTT feature of QUIC is a good way to enhance the performance of the server as it significantly reduces the amount of handshakes necessary. 0-RTT is always optional for the client and he can always request a new handshake if he wants. Therefore it does not change the attacker perspective who can still lead the same attack. However, it allows the server to detect an attack more easily. Indeed, a client repeating multiple complete 1-RTT handshake in a row while 0-RTT is available would be suspicious. If a client has already done the handshake recently, he still has the certificate and 1-RTT handshake would only represent a loss of perfomance and additional latency. Nevertheless, 0-RTT exposes the server to other vulnerabilities inherent to this feature.

Even though QUIC has multiple RFC documents [13] [23] [15] to provide guidelines, there are multiple different implementations [34] [35] [36]. The difference in efficiency in this variety of solutions might have an impact on the results we found. It would be interesting to try the vulnerability of different implementations. Our implementation was using Python, a more resource-efficient language as C could reduce the amplification factor. We found a high amplification factor that can be used by an attacker as a lever on its victim. An attacker can take down an infrastructure which has 4 times its own computing power. This is a serious issue which was not present in TCP SYN Cookie. The resilience of TCP SYN Cookie is mainly due to the simplicity of its design. The server only computes the cookie and compare it to the one from the packet. There are no additional mechanisms. TCP SYN Cookie [31], on the other hand seems to have only one legitimate version, the one of its designer written for Linux and adapted for other systems. It makes the deployments of this protocols more homogeneous. Our work was conducted on low to medium resources devices. The results are relevant but could use comparison to the architecture of already deployed QUIC servers of larger scales. Finally, the address validation mechanism present in RFC9000 is high level and not very specific. Therefore depending on the implementation, the attack might need to send an acknoledgment once in a while not to get blocked by the server. A comparison of the different address validation mechanisms would be interesting.

VI. Conclusion

In conclusion, this paper presented an empirical evaluation of QUIC against Flooding Attack. It evaluated the resource consumption of the attacker and the victim server during an attack. The results showed that the CPU is the main bottleneck during the attack. The memory presents a high amplification but remains at low level compared to CPU not making it

a primary concern. Bandwidth is symmetrical on the client and server-side making the amplification neutral and reducing significantly the interest of an attacker into reflection attacks. TCP SYN Cookie presents interesting capabilities in terms of performance during the handshake. However the embedded TLS handshake in QUIC provides faster RTT and therefore better performances.

REFERENCES

[1] Y. Cui, T. Li, C. Liu, X. Wang, and M. Kühlewind, "Innovating Transport with QUIC: Design Approaches and Research Challenges," *IEEE Internet Computing*, vol. 21, pp. 72–76, Mar. 2017.

[2] Y. A. Joarder and C. Fung, "A survey on the security issues of quic," in *2022 6th Cyber Security in Networking Conference (CSNet)*, pp. 1–8, 2022.

[3] M. Nawrocki, R. Hiesgen, T. C. Schmidt, and M. Wählisch, "Quicsand: Quantifying quic reconnaissance scans and dos flooding events," in *Proceedings of the 21st ACM Internet Measurement Conference*, (Virtual Event), pp. 283–291, ACM, 2021.

[4] J. Mirkovic and P. Reiher, "Internet denial of service: Attack and defense mechanisms," 2004.

[5] C. Rossow, "Amplification Hell: Revisiting Network Protocols for DDoS Abuse," in *Proceedings 2014 Network and Distributed System Security Symposium*, (San Diego, CA), Internet Society, 2014.

[6] M. Kosek, T. V. Doan, M. Granderath, and V. Bajpai, "One to Rule them All? A First Look at DNS over QUIC," Mar. 2022. arXiv:2202.02987 [cs].

[7] M. Kührer, C. Rossow, and T. Holz, "Hell of a handshake: Abusing tcp for reflective amplification ddos attacks," in *USENIX Security Symposium*, pp. 463–478, 2014.

[8] D. Shah and V. Kumar, "TCP SYN Cookie Vulnerability," July 2018. arXiv:1807.08026 [cs].

[9] M. Nawrocki, P. F. Tehrani, R. Hiesgen, J. Mücke, T. C. Schmidt, and M. Wählisch, "On the interplay between TLS certificates and QUIC performance," in *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*, CoNEXT '22, (New York, NY, USA), pp. 204–213, Association for Computing Machinery, Nov. 2022.

[10] K. Wolsing, J. Rüth, K. Wehrle, and O. Hohlfeld, "A Performance Perspective on Web Optimized Protocol Stacks: TCP+TLS+HTTP/2 vs. QUIC," June 2019.

[11] C. Rossow, "Amplification hell: Revisiting network protocols for ddos abuse," in *Proceedings of NDSS*, p. 15, Internet Society, 2014.

[12] F. J. Ryba, M. Orlinski, M. Wählisch, C. Rossow, and T. C. Schmidt, "Amplification and drdos attack defense – a survey and new perspectives," Tech. Rep. 1505.07892, arXiv, 2015.

[13] "QUIC: A UDP-Based Multiplexed and Secure Transport," Tech. Rep. RFC9000, RFC Editor, May 2021.

[14] F. J. Ryba, M. Orlinski, M. Wählisch, C. Rossow, and T. C. Schmidt, "Amplification and DRDoS Attack Defense – A Survey and New Perspectives," May 2015.

[15] J. Iyengar and I. Swett, "QUIC Loss Detection and Congestion Control," Request for Comments RFC 9002, Internet Engineering Task Force, May 2021.

[16] P. Biswal and O. Gnawali, "Does QUIC Make the Web Faster?," *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, Dec. 2016.

[17] G. Carlucci, L. De Cicco, and S. Mascolo, "HTTP over UDP: an experimental investigation of QUIC," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, (New York, NY, USA), pp. 609–614, Association for Computing Machinery, Apr. 2015.

[18] S. Cook, B. Mathieu, P. Truong, and I. Hamchaoui, "QUIC: Better for what and for whom?," *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6, May 2017.

[19] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, "Taking a long look at QUIC: an approach for rigorous evaluation of rapidly evolving transport protocols," in *Proceedings of the 2017 Internet Measurement Conference*, (London United Kingdom), pp. 290–303, ACM, Nov. 2017.

[20] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru, "How Secure and Quick is QUIC? Provable Security and Performance Analyses," *2015 IEEE Symposium on Security and Privacy*, pp. 214–231, May 2015.

[21] E. Gagliardi and O. Levillain, "Analysis of QUIC Session Establishment and Its Implementations," vol. 12024, (Cham), pp. 169–184, Springer International Publishing, 2020.

[22] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Request for Comments RFC 8446, Internet Engineering Task Force, Aug. 2018.

[23] M. Thomson and S. Turner, "Using TLS to Secure QUIC," Request for Comments RFC 9001, Internet Engineering Task Force, May 2021.

[24] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," Request for Comments RFC 5280, Internet Engineering Task Force, May 2008.

[25] M. D, T. L, M. J, B. J, and B.-J. J, "Analyzing the Adoption of QUIC from a Mobile Development Perspective," *EPIQ 2020 - Proceedings of the 2020 Workshop on the Evolution, Performance, and Interoperability of QUIC*, 2020.

[26] "Website Fingerprinting in the Age of QUIC | MIT CSAIL."

[27] M. Trevisan, D. Giordano, I. Drago, M. M. Munafò, and M. Mellia, "Five Years at the Edge: Watching Internet From the ISP Network," *IEEE/ACM Transactions on Networking*, vol. 28, pp. 561–574, Apr. 2020.

[28] J. Mücke, M. Nawrocki, R. Hiesgen, P. Sattler, J. Zirngibl, G. Carle, T. C. Schmidt, and M. Wählisch, "Waiting for QUIC: On the Opportunities of Passive Measurements to Understand QUIC Deployments," Sept. 2022. arXiv:2209.00965 [cs].

[29] J. Rüth, I. Poese, C. Dietzel, and O. Hohlfeld, "A First Look at QUIC in the Wild," vol. 10771, pp. 255–268, 2018. arXiv:1801.05168 [cs].

[30] M. Kosek, L. Schumann, R. Marx, T. V. Doan, and V. Bajpai, "DNS Privacy with Speed? Evaluating DNS over QUIC and its Impact on Web Performance," in *Proceedings of the 22nd ACM Internet Measurement Conference*, pp. 44–50, Oct. 2022. arXiv:2305.00790 [cs].

[31] W. Eddy, "TCP SYN Flooding Attacks and Common Mitigations." RFC 4987, Aug. 2007.

[32] J. Lainé, "aioquic." https://github.com/aiortc/aioquic, 2019.

[33] J.-P. Aumasson and D. J. Bernstein, "Siphash: a fast short-input prf." Cryptology ePrint Archive, Paper 2012/351, 2012. https://eprint.iacr.org/2012/351.

[34] C. Huitema, "picoquic." https://github.com/private-octopus/picoquic, 2019.

[35] F. incubator, "mvfst." https://github.com/facebookincubator/mvfst, 2019.

[36] Microsoft, "Msquic." https://github.com/microsoft/msquic, 2019.