Zangetsu: a censorship resistant QUIC VPN

Dhruv Patel, Amit Phabba, Karthikeya Sharma,
Bhagyashree Ramananda, and Gaurav Chandrabhan

College of Computing, Georgia Institute of Technology

Abstract

The essence of censorship circumvention is encryption protocols providing data confidentiality, integrity, and authenticity. Many governmental, ISP, and other entities employ shallow and deep packet inspection to detect such traffic. Virtual Private Network (VPNs) proxies have become indispensable for safeguarding sensitive data and ensuring secure communication over any public or private network. However, many popular VPN protocols (namely IPSec, OpenVPN, and Wireguard) have distinct headers, making throttling and/or outright blocking easy by censoring entities. QUIC (Quick UDP Internet Connection) is a protocol devised by Google to achieve decreased latency by bundling various layers and utilizing multiple streams over UDP. It is the transport layer protocol of the new HTTP/3 standard and has built-in congestion control and packet loss remediation. In this paper, we implement a VPN proxy utilizing QUIC to mask all data as normal web traffic and reduce connection latency. We found QUIC to be a very effective protocol to use for proxies. QUIC had the best download bandwidth, second best upload bandwidth, average ping latencies around the world, and second best connection setup (handshake) latency, with the potential for "0 RTT" handshakes to further reduce the time. Built with security in mind (and for the purpose of HTTPS/3 and TLS 1.3), we also found that the encryption standards used in QUIC are very secure in comparison to the other VPN protocols. Tunneling over QUIC increases potential collateral damage should censoring entities wish to block or throttle it, furthering utility in evading censorship. Based on our observed speeds, obfuscation properties, security-oriented, and novel "future-proof" protocol design, we believe that a QUIC-based VPN can be a way forward to enforce data privacy in censored environments.

Introduction

QUIC (Quick UDP Internet Connections) spearheads advancements in modern transport layer protocols, presenting a transformative impact on web communication. Developed by Google to surmount the constraints of traditional TCP and UDP protocols, QUIC prioritizes speed, security, and efficiency in internet data transfer. Its distinctive feature lies in seamlessly merging connection setup and encryption processes, significantly reducing latency. With support for multiplexing, enabling concurrent data streams within a single connection, and features like 0-RTT for rapid connection establishment, QUIC stands as a robust solution catering to the dynamic demands of contemporary web applications.

The QUIC connection establishment process starts with the client initiating a handshake by sending a Client Hello (CHLO) to the server. The server responds with a Reject (REJ) containing vital information including server configuration, certificate chain, and a source address token for client identity verification. After authenticating the server's configuration, the client sends a complete CHLO with its Diffie-Hellman (DH) value, allowing both parties to derive a shared key, alongside an encrypted request. The server responds with a final encrypted Server Hello (SHLO), completing the handshake. The client can retain server information for subsequent "0-RTT" handshakes, ensuring a secure and efficient connection establishment.

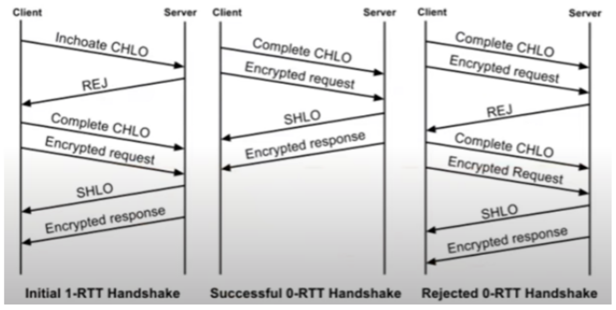

Figure 1: *0-RTT and 1-RTT handshake process between client and server employing QUIC*

Figure 1 gives an overview of 0-RTT handshake and 1-RTT handshake for client and server employing QUIC protocol. QUIC employs unique congestion and loss recovery mechanisms, assigning each packet a new number and utilizing acknowledgments that explicitly encode delay for precise Round-Trip Time (RTT) estimation. Wireshark detection of QUIC traffic involves Ethertype, Protocol fields, and heuristic dissectors. While QUIC's encryption ensures privacy, detection challenges arise in discerning VPN traffic from legitimate users. Censorship-resistant VPNs employ obfuscation techniques, and QUIC's active probing resistance adds complexity for censorship systems. Circumvention techniques include packet length padding and dynamic port allocation, but known weaknesses like TCP Fast Open and TLS over TLS present potential security compromises. Due to distinct characteristics in protocols like IPSec, WireGuard, and OpenVPN, QUIC VPN, resembling normal HTTPS traffic, offers privacy with encrypted packet numbers thwarting correlation by passive on-path attackers.

Our goal is to harness QUIC's latency improvements and address the current absence of standard VPNs utilizing the QUIC protocol in the market. Building upon prior work in the realm of QUIC, such as the development of encrypted performance-enhancing proxies (PEPs), we aim to create a QUIC-based VPN that combines the advantages of reduced latency and enhanced privacy through encryption. Our focus extends beyond theoretical frameworks, aiming to provide practical solutions for secure data transmission. We explore the viability of a QUIC-based, comparing and contrasting to the three most popular VPN protocols: IPSec, Wireguard, and OpenVPN. We compare effective speeds and security levels, as well as obfuscation ability. In the next section we cover related work, followed by our methodology and results. We end with a discussion of our findings. We aim to contribute to the evolution of secure and high-performance VPN solutions in the ever-evolving landscape of digital communication.

Related Work

There has been a plethora of past work in the field of QUIC. Many researchers have analyzed QUIC for flaws and Google's use of it for more than 50 percent of their traffic. Experiments have been conducted to compare the performance of QUIC and TCP in terms of three key aspects – communication quality, compatibility fairness and user experience [1]. It was found that QUIC exhibited significantly lower connection establishment latency compared to TCP, with a factor of 5x the speed. QUIC uses a congestion control algorithm that is more responsive to network conditions, thus less susceptible to packet loss than TCP. QUIC outperformed TCP in terms of throughput and fairness, particularly in high network congestion conditions. Using the Cubic congestion control algorithm, page load times decreasing by up to 37.11% compared to TCP.

Several attacks on QUIC have been identified [1]. The attacks exploit the protocol's vulnerabilities in its handshake and encryption. QUIC-Reflection, QUIC-Amplification, and 0-RTT DoS attacks are some common attacks that can be mitigated by implementing IP spoofing protection, packet number encryption, and connection throttling mechanisms. One of the most significant challenges is the lack of a mechanism against version downgrade attacks. In a version downgrade attack, an attacker can force a client to downgrade to an older, less secure version of the QUIC protocol. This can allow the attacker to exploit vulnerabilities in the older protocol. QUIC's stateless handshake and lack of explicit security negotiation have also raised concerns. Ongoing research focuses on developing more robust security mechanisms for QUIC, such as improved handshake protocols and stronger encryption algorithms. Some delve into the security implications of QUIC's design, particularly its use of an initial key to establish connections and its lack of forward secrecy [2]. QUIC uses an initial key to exchange data before a final session key is set. This can be exploited by attackers to prevent QUIC from achieving minimal latency advantages. If an attacker can compromise a long-term key, they will be able to decrypt all the data that has been exchanged using that key, even if the key is later revoked.

QPEP, a QUIC-based encrypted proxy for satellite broadband, addresses some of these concerns by providing an additional layer of security through encryption [3]. Traditional proxies and VPNs can introduce significant overhead to satellite broadband connections when using TCP. QPEP utilizes encryption to protect all communication between the client and the proxy, ensuring the confidentiality and integrity of sensitive data. It employs AES-GCM and ChaCha20-Poly1305 encryption algorithms, along with Perfect Forward Secrecy, Certificate Authority validation, and a secure handshake mechanism to safeguard against eavesdropping and tampering. QPEP was tested in a simulated environment, and it was found that it outperforms both unencrypted proxies and traditional VPNs, reducing latency by up to 50% and increasing throughput by up to 30%. QPEP has the potential to make satellite broadband more secure for a wide range of applications. An assessment of QUIC performance in satellite Internet access revealed that its encrypted transport layer headers render it impervious to alterations by Performance Enhancement Proxies (PEPs) [4]. The results demonstrated a superior performance of QUIC compared to HTTP over UDP tunnels, especially when utilized with Operator C, although some performance variations were observed.

Another study verified implementation of the QUIC record layer and evaluated its security and performance [5]. The researchers developed a reference implementation of QUIC utilizing the verified record layer, along with an example server and client for secure file transfers. The prototype comprises three modules: TLS handshake, QUIC record layer, and QUIC protocol logic, with the latter verified for safety. The performance evaluation of the record layer demonstrates throughput of 1.98 GB/s for typical packet sizes. The overall file transfer performance of the reference implementation was assessed, showing competitive results compared to an unverified baseline, with a 21 percent lag on larger file sizes attributed to a coarse-grained locking strategy. The study emphasizes the achievement of a provably-safe and secure QUIC implementation, providing insights into the verification methodology and performance characteristics. Hence, QUIC security issues arise from implementation issues rather than an actual protocol flaw.

Methodologies

QUIC Client & Server

We implemented a VPN software using QUIC as the transport and security protocol. The client side starts with pushing network traffic to a user space. This is typically conducted by a daemon started by the VPN application on the client system. A TUN/TAP Device (a virtual network interface) is used. The system recognizes this as any other network interface and forwards the network packets to it based on the rules set in the system's firewalls and forwarding rules (IP tables in Linux). We modify the rules to make the new interface the default gateway for all traffic, providing full encryption. Another process reads what is going out of the virtual interface and subsequently encrypts the data before forwarding over the physical network interface towards a VPN server. Since the packet is encrypted, the original destination is masked by the encryption and only the VPN server headers are visible in the packet to any onlooker. [6]

On the server side, network packets from the client are received and are decrypted over the physical network interface. The server has the packets that the client intended for various destinations. If the server forwards these packets to the Internet as it is, the destination servers will see the client's source IP or the private IP of the VPN subnet. The server process will change the src IP in the packet to the server's IP and a custom server port. The server must then maintain a record of this matching so that when it receives a packet intended for the server IP and that particular port, it can identify that this packet is for the client and forward it over the encrypted protocol accordingly (much like NAT gateways). This feature is called Masquerading and can be enabled for the server's network interface. Since we forward the packets from the physical interface to the virtual network interface (and vice versa), the firewall must be adjusted to enable forwarding of network packets in both of these interfaces on both the client and the server.

In the actual implementation, we stick very closely to the outline above. We start with the creation of the TUN/TAP Device. We do this by using the python-pytun package which is a Python wrapper for a C library. We use a bash script which has the requisite iptables commands to modify them. We start a separate thread to read the packets from the device and then transmit it over the established QUIC connection. The QUIC protocol implementation we used was the aioquic library [7]. This library gives us an interface to implement for the client and server and simplifies the initial handshake, as well as providing functions and objects needed to transmit data subsequently. The necessary parameters are server IP, port, server certificate, and client session tickets. On the server side, we use the same libraries to establish the connection and a thread to read and transmit data over to the client side. We can use the same class with certain tweaked parameters for both the client and the server. Improvements can be made to our implementation, include support for a multi-client architecture, robust authentication, and converting the tunnel read to a future so that it can be completely asynchronous (increasing multi-client performance).

Measurements

We compare and contrast four VPN protocols: IPSec, OpenVPN, Wireguard, and our QUIC VPN, alongside a baseline (control) of no VPN usage. We do a latency experiment to assess the latency incurred compared to the baseline. We focus on measuring the round-trip time (RTT) for a 10-packet ping from a client to five destination web servers located across the world. The chosen destinations include eecs.berkeley.edu (UC Berkeley), gov.za (government of South Africa), cam.ac.uk (Cambridge University), jpx.co.jp (Tokyo stock exchange), and fearp.usp.br (Sao Paulo University), providing a diverse geographic spread. Latencies from multiple trials were averaged out. As time of day affects global Internet congestion, each trial was held at different times of the day (Eastern Standard Time).

In the bandwidth and jitter test, we evaluate the impact of various VPN protocols on bandwidth and throughput. This is accomplished through the use of the SpeedTest CLI to assess both upload and download bandwidths. We also investigated bandwidth jitter. Similar to latency, we did trials throughput the day and averaged the results. The same SpeedTest server used in all trials and was geolocated in Atlanta.

The handshake latency test focuses on quantifying the time it takes to establish a connection between the client and server for different VPN protocols. Unique handshakes are employed for each VPN protocol, and the nature of the handshake (1 RTT or 2+ RTT) was investigated. The methodology employs Wireshark to identify key handshake events, enabling precise measurement of connection establishment latency. Multiple trials were averaged. We consider handshake latency to be the time between the first connection setup packet being sent and the first data packet being sent.

In conducting the longer-term throughput testing, we implemented a continuous speed test using the ipFail tool, running multiple 120-second tests for each VPN protocol. The experiments were executed with a single download thread over the specified duration. The ipFail tool was selected for its reliability in measuring network performance, and the extended duration of the tests allowed us to capture a more comprehensive view of the protocols' sustained throughput. Graphs generated from the results will be analyzed to assess any observed jitter, providing valuable insights into the stability and consistency of the VPN protocols over an extended testing period.

Findings

The tests conducted on the VPN protocols reveal that QUIC stands out in certain areas and has room for improvement in others.

In terms of latency – a measure of time delay while using the VPN – QUIC VPN was never the fastest across all the tested locations, including Berkeley, Cambridge, Cape Town, Tokyo, and Sao Paulo (Figure 2). This suggests that QUIC

VPN's performance varies based on location and might be affected by factors like distance or network infrastructure. Note that latency measurements will always be subject to global network congestion and destination's time of day.

The download speeds achieved by the QUIC VPN were the second highest among the protocols tested, signifying its strength in receiving data (Figure 4). The upload bandwidth was second highest (Figure 3).

In the jitter test, which measures the consistency of the connection, QUIC VPN had a moderate performance, indicating occasional fluctuations in speed (Figure 5).

During the handshake latency test, which measures the time it takes to establish a connection, QUIC VPN came in second (Figure 6). It's important to note that Wireguard, the protocol with the fastest handshake time, used a technique that allows instant connection setup, known as "0 RTT", which the QUIC protocol is designed to utilize but did not in this particular test. Thus, Wireguard's handshake latency was simply the time it took to craft and send the data packet after the first connection setup packet was sent.

We also observe qualitative information regarding the underlying security of the different VPN protocols (Table 1).

Wireguard achieved the highest upload bandwidth and lowest handshake latency. QUIC achieved the highest download bandwidth. IPSec achieved lowest bandwidth jitter. And no one protocol showed the lowest ping latency consistently: to Berkeley, Cambridge, and Cape Town was OpenVPN and to Tokyo and Sao Paulo was IPSec.

Discussion & Analysis

We found QUIC to have the highest download bandwidth and second highest upload bandwidth (trading spots with Wireguard). This is caused by the multistreaming property of QUIC connections, eliminating head-of-line blocking as data can be delivered on an alternate stream if one gets blocked. This data is then reassembled at the destination. Congestion control is moved to the user space in QUIC and packet loss recovery is implemented by the protocol as well, essentially allowing TCP-level reliability but over multiple UDP streams. Despite the additional overhead of servicing multiple streams and separately managing flow, we saw that QUIC achieved average latencies with pings across the world. We observed it to be second best to Cambridge & Tokyo, third best to Berkeley, and worst to Cape Town & Sao Paulo. Seeing as QUIC did not perform the worst across the board and noting that the latencies were all relatively close to one another, we reason that the overhead of using this protocol alongside performing encryption does not significantly negatively impact the performance. That is to say that the stream management in QUIC does not slow down its real-world usage in comparison to other popular VPN protocols. We also acknowledge the fact that Internet usage at certain times will have varying levels of congestion, especially when accessing resources around the world. The round trip of going across the ocean is significantly higher than any performance overhead, and having no outlier latency values attests to this fact. QUIC performed third best in bandwidth jitter, with Wireguard narrowly beating it by 1.8 ms. Similar to latencies, bandwidth jitter is very dependent on current local and public Internet conditions and congestion levels. Not observing a worse value (or an outlier value) when compared to the other VPN protocols shows the somewhat consistent performant nature of QUIC. Finally, we observed the second best handshake latency with QUIC. Handshake latencies were measured as the time between sending the first handshake packet and sending the first data packet. Wireguard uses a special "0 RTT" handshake, resulting in much lower times than QUIC's 1 RTT handshake. However, we were doing authentication through username/password. The QUIC protocol has a built-in "0 RTT" handshake should the client remember the server's configuration. Thus, the QUIC handshake latency can be further reduced to that of "0 RTT" with the use of configuration files or session tokens. In such cases, the encrypted data packet is sent immediately after the complete Client Hello packet is sent, resulting in immediate data transfer.

The QUIC-based VPN provides the same pro-

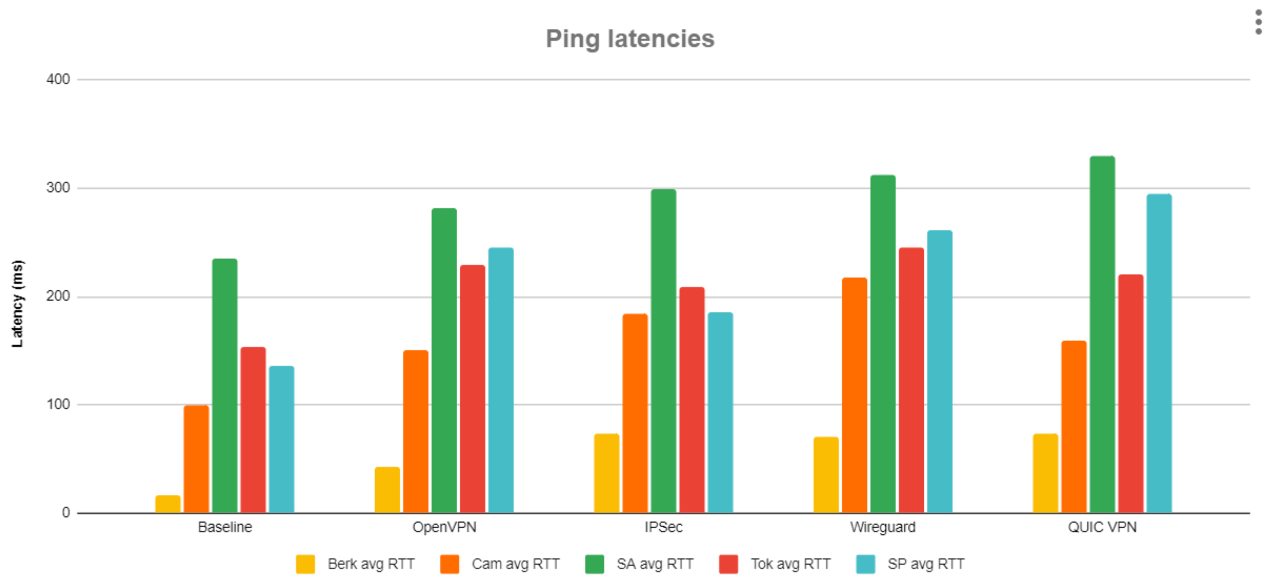

Figure 2: *Observed latencies of different VPN protocols*

	OpenVPN	IPSec	Wireguard	Zangetsu
Encryption scheme	AES 256	AES 128 CBC	ChaCha20	AES 128 GCM
Hashing scheme	SHA 256	SHA 256	Blake2s	SHA 256
Bits of entropy	256	128	256	128
Do headers reveal VPN usage?	Yes	Yes	Yes	No

Table 1: *Security schemes of different VPN protocols*

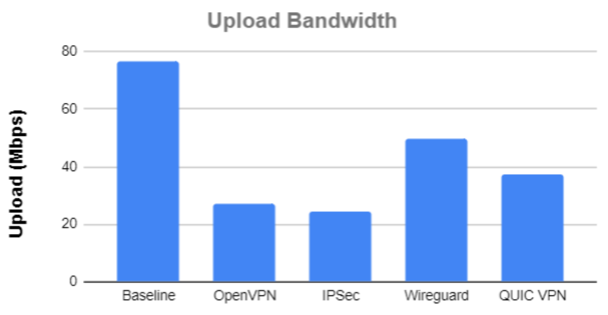

Figure 3: *Upload bandwidth of different VPN protocols*

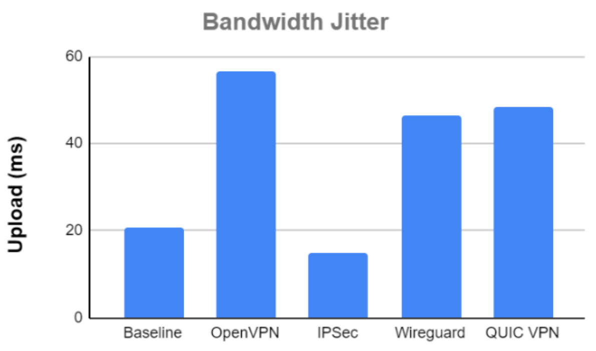

Figure 5: *Bandwidth jitter of different VPN protocols*

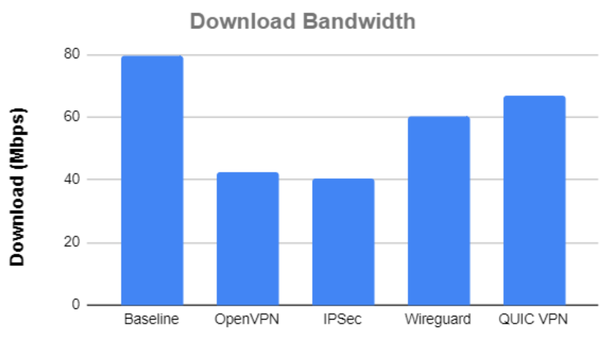

Figure 4: *Download bandwidth of different VPN protocols*

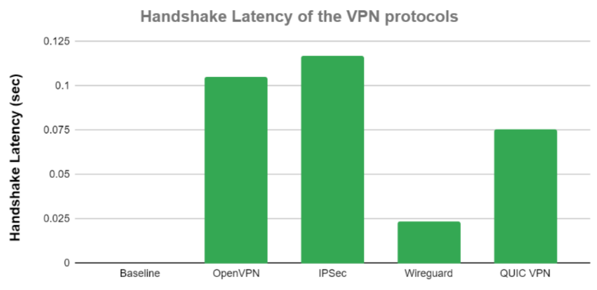

Figure 6: *Handshake times of VPN protocols*

tections as any VPN, namely confidentiality and integrity on any network. This prevents MITM from snooping traffic. The biggest advantage of QUIC over IPSec, OpenVPN, and Wireguard is that it looks like normal web traffic. QUIC is already used by Google for over 50% of traffic to Google servers and is the transport layer protocol for the upcoming HTTPS/3 web protocol [8]. Many companies (including big names such as Meta) are also adopting it, leading to 30% of all HTTP traffic being HTTPS/3 [9]. Look like normal web traffic allows the QUIC VPN traffic to bypass most firewalls and network address translators. Choosing a growing protocol that will eventually be used for most/all web traffic is perfect for censorship circumvention. While a traditional HTTPS/2 proxy can be used to tunnel traffic, it does not have the speed improvements of QUIC's multi-streamed low-latency, security-oriented design, or "future-proofness". IPSec (ESP packets) are easily sniffed due to their header format. OpenVPN and WireGuard both suffer the same issue. Even when using traditional TLS with these existing VPN protocols, the handshake itself can expose the fact that a VPN is being used, leading to blocking or throttling of the connection. QUIC does not have to deal with these issues. It also encrypts packet numbers as part of its protocol design, stopping passive on-path attackers from correlating users' activity over multiple network paths. Contrary to most VPN protocols, the default port should be used for the QUIC VPN to avoid suspicion, as web traffic is expected to go through port 443. Alternative port usage might trigger alarms in censorship systems, highlighting the need for subtlety in traffic patterns.

Our QUIC VPN would perform well under the known heavy censorship of China. The Great Firewall of China is known to use deep packet inspection, looking at multiple heuristics when determining VPN usage before throttling suspected connections including: fingerprints of known protocols, entropy tests using the fraction of set bits, and the fraction, position, and max contiguous count of printable ASCII characters [10]. Utilizing full encryption from the start (such as encrypting the 0 RTT handshake with a predetermined configuration) and providing fake data to active probing attempts can both thwart attempts to detect VPN usage. Entropy tests would yield same result as TLS encryption (used in all HTTPS). Currently, action is taken in a white-list style where if zero of the heuristics are matched then a block is made, however our QUIC VPN matches more than one of the heuristics. Finally, China's Great Firewall is currently limited to TCP, but QUIC uses UDP, creating much greater complexity if support is added. If an entity attempted to block or throttle observed QUIC traffic on suspicion that it is a VPN, then it would lead to large collateral damage (high false positive rate). If an entity is willing to take such collateral damage or control the end device itself (through software or through an installed root CA), then not much can be done to prevent blocking of QUIC or of any other VPN. Thus, the QUIC VPN likely performs well under those conditions. Further research is to be done on practical operation of the QUIC VPN.

Issues & Future Work

Several strategic areas present themselves as avenues for further refinement and innovation. One significant aspect involves the implementation of Zero Round-Trip Time (0 RTT), a feature designed to enhance connection latency. This entails the maintenance or storage of session tickets on the server, facilitating faster connection setups and substantially reducing latency, thereby optimizing user experience. Expanding the capabilities of QUIC-based VPNs to accommodate multi-client support is paramount for scalability and versatility beyond hobbyist use. Future development efforts should focus on optimizing system architecture to securely handle concurrent connections, catering to the diverse needs of an expanding user base. In parallel, strengthening authentication mechanisms emerges as a priority. Some may consider utilizing more robust authentication protocols, such as multi-factor or biometric authentication, to fortify defenses. An additional avenue for improvement lies in the implementation of dynamic routing and load balancing capabilities. Moreover, exploring advanced encryption standards, including the option for AES 256-bit encryption, addresses concerns about the vulnerability of lower-bit encryption methods, offering users the choice of stronger encryption. Lastly, given the evolving landscape of cryptography, research into quantum-resistant cryptography algorithms continues to be prevalent. This forward-looking approach involves future-proofing the VPN against emerging threats.

Conclusion

Well-known VPN protocols like IPSec, OpenVPN, and Wireguard are susceptible to detection and interference due to their distinctive headers. This paper introduces Zangetsu, a QUIC-based VPN, as a promising alternative. QUIC was originally designed by Google to minimize latency and bundled with various layers, making it a robust transport layer protocol. The implementation of a VPN using QUIC demonstrated notable advantages, including superior download bandwidth, competitive upload bandwidth, and efficient connection setup latency. Notably, QUIC's encryption standards prove to be secure, positioning it as a viable option for evading censorship. The observed speeds, obfuscation properties, and forward-looking protocol design suggest that a QUIC-based VPN holds potential for enhancing data privacy in censored environments. Being disguised as normal web traffic, QUIC gains strong censorship-resistant properties. While AES 256 theoretically offers a larger key space and may withstand brute force attacks slightly better, the practical difference in security between AES 128 and AES 256 is often negligible for most applications. AES 128 also offers advantages in terms of computational efficiency and resource utilization. QUIC's role in VPN applications focuses on minimizing latency and increasing speed, rendering it an optimal choice. Its adept resistance to probing proves advantageous in regions with censorship, enabling the VPN to operate discreetly. We identify avenues for future research, such as multi-client support, robust authentication mechanisms, and the implementation of session tickets for 0 RTT handshakes, showing the promising trajectory of QUIC in the realm of VPN technology. A link to the code can be found here: `github.com/Blastguy/Zangetsu-VPN` [11]

References

[1] E. Chatzoglou et al. "Revisiting QUIC attacks: a comprehensive review on QUIC security and a hands-on study". In: *Int. J. Inf. Secur.* 22 (2023), pp. 347–365. DOI: 10.1007/s10207-022-00630-6.

[2] Robert Lychev et al. "How Secure and Quick is QUIC? Provable Security and Performance Analyses". In: (2015), pp. 214–231. DOI: 10.1109/SP.2015.21.

[3] J. Pavur et al. "QPEP: A QUIC-Based approach to encrypted Performance enhancing proxies for High-Latency Satellite Broadband". In: (2020). DOI: 10.48550/arXiv.2002.05091.

[4] J. Deutschmann, Hielscher K. S., and R. German. "Satellite Internet Performance Measurements". In: *International Conference on Networked Systems* (2019), pp. 1–4. DOI: 10.1109/NetSys.2019.8854494.

[5] A. Delignat-Lavaud et al. "A Security Model and Fully Verified Implementation for the IETF QUIC Record Layer". In: *IEEE Symposium on Security and Privacy* (2021), pp. 1162–1178. DOI: 10.1109/SP40001.2021.00039.

[6] Frank Denis. *dsvpn*. GitHub. github.com/jedisct1/dsvpn. 2020.

[7] Jeremey Laine. *aioquic*. GitHub. github.com/aiortc/aioquic. 2023.

[8] Frederic Lardinois. "Google Wants To Speed Up The Web With Its QUIC Protocol". In: (2015). URL: techcrunch.com/2015/04/18/google-wants-to-speed-up-the-web-with-its-quic-protocol.

[9] D. Belson and L. Pardue. "Examining HTTP/3 usage one year on". In: (2023). URL: blog.cloudflare.com/http3-usage-one-year-on.

[10] M. Wu et al. "How the Great Firewall of China Detects and Blocks Fully Encrypted Traffic". In: *USENIX Security Symposium* (2023), pp. 2653–2670.

[11] *Zangetsu*. GitHub. github.com/Blastguy/Zangetsu-VPN. 2023.