Accelerating QUIC's Connection Establishment on High-Latency Access Networks

Erik Sy, Tobias Mueller, Moritz Moennich and Hannes Federrath

University of Hamburg

Hamburg, Germany

Abstract—A significant amount of connection establishments on the web require a prior domain name resolution by the client. Especially on high-latency access networks, these DNS lookups cause a significant delay on the client's connection establishment with a server. To reduce the overhead of QUIC's connection establishment with prior DNS lookup on these networks, we propose a novel QuicSocks proxy. Basically, the client delegates the domain name resolution towards the QuicSocks proxy. Our results indicate, that colocating our proxy with real-world ISP-provided DNS resolvers provides great performance gains. For example, 10% of our 474 sample nodes distributed across ISP's in Germany would save at least 30 ms per QUIC connection establishment. The design of our proposal aims to be readily deployable on the Internet by avoiding IP address spoofing, anticipating Network Address Translators and using the standard DNS and QUIC protocols. In summary, our proposal fosters a faster establishment of QUIC connections for clients on high-latency access networks.

Index Terms—QUIC Transport Protocol, SOCKS Proxy, DNS, QuicSocks Proxy

I. Introduction

For U.S. households latency is the main web performance bottleneck for broadband access networks exceeding a throughput of 16 Mbit/sec [1]. Depending on the user's location and the deployed access technology like cable, fiber, Digital Subscriber Line (DSL), Long-Term Evolution (LTE), or satellite, users may experience significant network latencies. High-latency links reduce the user's quality of experience during web browsing [2] and negatively impact the per-user revenue of online service provider [3]. Thus, optimizing the web performance on such existing high-latency network links is an important task. In this paper, we focus on improving the time to first byte which contributes up to 21% of the page load time for popular websites [1]. In detail, we improve the delay of QUIC's connection establishment with prior DNS lookup on high-latency links. QUIC replaces the TLS over TCP protocol stack within the upcoming HTTP/3 version [4]. As the web is built upon the Hypertext Transfer Protocol (HTTP) and the standardization of QUIC receives widespread support, the QUIC protocol is expected to be widely deployed on the Internet in the forthcoming years.

Our proposal assumes, that Internet Service Providers (ISP) aim to improve their clients' quality of experience during web browsing. This assumption is substantiated by ISPs providing recursive DNS resolvers to accelerate their client's DNS lookups. In this work, we propose ISP-provided proxies to reduce the delay of their client's QUIC connection establishments. Instead of conducting a DNS lookup and waiting for the response, this design allows the client to directly send its initial QUIC messages to the novel QuicSocks proxy. Upon receiving these messages, the proxy resolves the domain name and forwards the messages to the respective QUIC server. After the end-to-end encrypted connection between the client and the server is established, the connection is seamlessly migrated to the direct path between these peers.

These novel QuicSocks proxies can accelerate the client's connection establishment to a server, if they perform faster DNS lookups and/or have a lower network latency to the QUIC server compared to the client. A favorable network topology would place a QuicSocks proxy colocated with the ISP-provided DNS resolver in an on-path position between client and the server.

Our proposal can be applied to many high-latency links. Within the next years, enterprises like SpaceX, OneWeb, and Telesat plan to launch thousands of satellites for global broadband connectivity aiming to provide Internet access to millions of people [5]. This presents a well-suited application area for our proposal because of the significant latencies of about 30 ms between the client and the ISP's ground station [6].

In summary, this paper makes the following contributions:

- We propose the novel QuicSocks design that allows clients to send initial handshake messages without a prior resolution of the domain name. The name resolution is conducted by a QuicSocks proxy from a more favorable position in the ISP's network to accelerate the connection establishment.
- We evaluate our proposal by assuming a colocation of the ISP-provided DNS resolver with the QuicSocks proxy. Results based on our analytical model indicate for a QUIC connection establishment accelerations between 33% and 40% on a U.S. mobile LTE network. Furthermore, our measurements of real-world network topologies indicate the feasibility of significant performance gains for clients on high-latency access networks. For example. 10% of the investigated clients save at least 30 ms to complete their QUIC handshake.
- We implemented a prototype of our proposal to demonstrate its real-world feasibility. Our results indicate, that the computations of the QuicSocks proxy itself are lightweight and contribute less than 1.2 ms to a QUIC connection establishment.

The remainder of this paper is structured as follows: Section II introduces the QUIC and the SOCKS protocol and describes the performance problem that we aim to solve. Section III summarizes the proposed QuicSocks design and evaluation results are presented in Section IV. Related work is reviewed in Section V, and Section VI concludes the paper.

II. BACKGROUND AND PROBLEM STATEMENT

In this section, we first describe the QUIC protocol which is deployed in HTTP version 3. Subsequently, we review the SOCKS protocol used to exchange network packets between a client and a server through an intermediate proxy. Then, we investigate the performance problem of DNS queries on subsequent connection establishments.

A. QUIC transport protocol

In this paper, we refer to QUIC's draft version 20 of the Internet Engineering Task Force (IETF) as the QUIC protocol [7]. The QUIC transport protocol will replace TLS over TCP in the upcoming HTTP/3 network protocol [4], which is closely tied to the world wide web. Thus, deployment of HTTP/3 on the web will significantly contribute to QUIC's adoption on the Internet in the forthcoming years. Compared to TLS over TCP, the UDP-based QUIC protocol allows for faster connection establishments [8], mitigates head-of-line blocking [9], and can be extended because of a lower interference through middleboxes [10].

In the following, we provide details on two mechanisms of the QUIC protocol that our proposed QuicSocks approach makes use of. These are a challenge-response mechanism in QUIC's connection establishment known as stateless retry and QUIC's connection migration that allows transferring an established connection to a new endpoint address.

a) Stateless retry: The stateless retry mechanism can be optionally used by QUIC servers to validate the source address claimed by a client before proceeding with the cryptographic connection establishment. As shown in Figure 1, the server responds to the client's initial connection request with a retry message that contains a source address token. This token is opaque to the client and contains information about the client's source address. Subsequently, the client returns this token together with its previously sent ClientHello message. Upon receiving this message from the client, the server first validates the presented token. A token is valid for a connection request if the client's claimed source address matches the address encoded in the token. In this case, the server assumes that the client can receive packets at the claimed address and proceeds with the cryptographic connection establishment. A stateless retry presents a performance limitation as it adds a round-trip time to the connection establishment. However, it supports QUIC servers to protect against denial-of-service attacks. Therefore, QUIC servers are likely to use these optional stateless retries when experiencing many connection requests from source addresses with unresponsive clients.
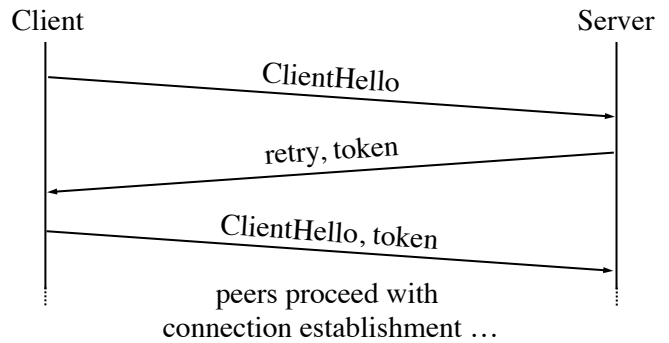

Fig. 1. Schematic of QUIC's stateless retry mechanism.

b) Connection migration: QUIC connections are associated with connection IDs that allow the identification of a connection independently of the used source address and port number. Connection IDs allow connections to survive an endpoint's change of the IP address and/or port number which might occur because of NAT timeouts and rebinding [11], or clients changing their network connectivity. Only QUIC clients can initiate connection migrations to a different endpoint's IP address and/or port number. However, the client must wait until the handshake is completed and forward secure keys are established before initiating the connection migration. The endpoints might use multiple network paths simultaneously during the connection migration. The peers can optionally probe a new path for peer reachability before migrating a connection to it. However, when a connection is migrated to a new path the server must ensure the client is reachable via this path before sending large amounts of data. Furthermore, the peers need adapting their sending rate to the new path by resetting their congestion controller and round-trip time estimator.

B. SOCKS protocol

RFC 1928 describes the current version of the SOCKS protocol [12]. Usages of SOCKS proxies include the traversal of network firewalls [12], the translation between IPv6 and IPv4 address space [13], and privacy-enhancing technologies such as TOR onion routing [14]. Figure 2 provides a schematic of a connection between client and server through a SOCKS proxy. To begin with, the client establishes a TCP connection to the proxy's port 1080. This connection is used by the client and the SOCKS proxy to exchange control messages. For example, the client can use this control channel for authentication or to request a new connection to a server. The SOCKS protocol supports the exchange of UDP datagrams between the client and server. As a result, QUIC connections can be established via default SOCKS proxies. For this purpose, the client sends a UDP associate request to the socks proxy. If the client's request is approved, the SOCKS proxy responses with a source address and port number to which the client can send the UDP datagrams to be relayed to the server. Subsequently, the client attaches a SOCKS request header to its UDP datagrams and sends them to the indicated port number/ IP address. Upon
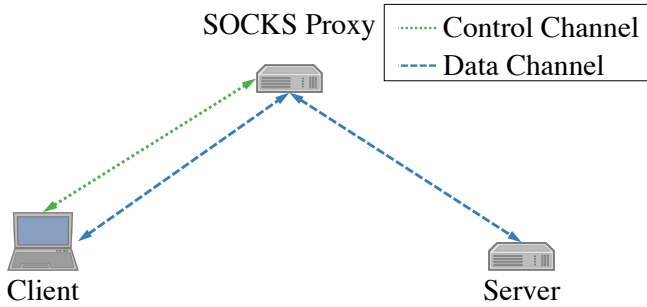
Fig. 2. Schematic of a network packet exchange between client and server employing an intermediate SOCKS proxy.

Fig. 3. Schematic of a complete DNS query using a recursive resolver.

receiving these UDP datagrams, the proxy will remove the request header and send them from its own source address to the server. The server will send its response to the proxy server, which will then relay it to the client. Note, that the SOCKS protocol allows clients to delegate the task of DNS name resolution. For this purpose, the client includes the domain name within its request header. Subsequently, the SOCKS proxy resolves this domain name and relays the packets to the corresponding destination.

C. Delays caused by high latencies to recursive DNS resolvers

The Domain Name System (DNS) is responsible for resolving domain names into IP addresses. Many operating systems or web browsers have a local DNS cache. However, between 12.9% and 20.4% of a user's established TCP connections directly follow a DNS query [15]. A popular website requires connections to about 20 different hostnames [16]. Hence, the user conducts on average between 2.6 and 4.1 fresh DNS queries per website retrieval. Each of these DNS queries delays the subsequent connection establishment to the server serving the queried hostname. Furthermore, websites usually have a nested hierarchy of requests to different hostnames [16]. If such nested requests to different hostnames require each a DNS query by the client, then the website loading is delayed by the time required for these sequential DNS queries.

In this paper, we assume that the client can resolve either a domain name using its local cache or needs to query recursively its DNS resolver as shown in Figure 3. If the recursive resolver has a cache miss for the queried domain name, it starts an iterative query. The arrows two to seven in Figure 3 indicate such a complete iterative query involving the DNS root server, Top Level Domain (TLS) server, and finally the authoritative nameserver of the respective domain name. DNS recursive resolvers make extensive use of caching with reported hit rates larger than 80% [15]. Thus, the round-trip time (RTT) between the client and the recursive resolver can present a significant source of delay for a DNS query. Studies of home routers in the US indicate typical RTT between 5 ms and 15 ms to their ISP-provided DNS resolver [1]. However, a fraction of about 5% of the users experience a RTT longer than 20 ms [17]. Studies with the popular third-party resolver Google Public DNS indicate a median RTT of 23 ms, while between 10%
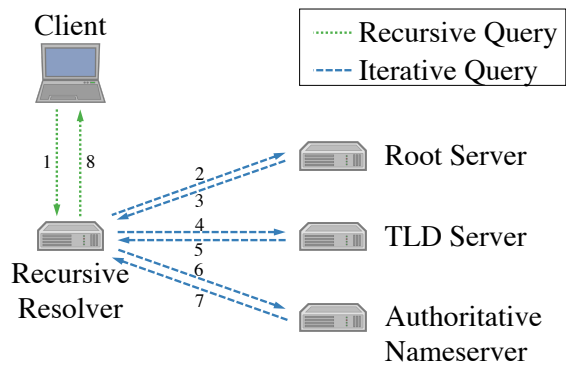
and 25% of the measurement nodes experienced RRTs longer than 50 ms [17]. For users having a downstream throughput of more than 16 Mbits/sec, the page load time highly depends on their network latency and DNS query time compared to their available throughput [1]. As a result, especially clients having a high network latency to their resolver require technological improvements to reduce their experienced DNS query time to achieve faster website retrievals.

III. QUICSOCKS

In this section, we introduce the QuicSocks design. This novel approach improves the latency of QUIC's connection establishments that directly follow a DNS lookup. First, we summarize our design goals, before we present QuicSocks. Finally, we describe the implementation of our QuicSocks prototype.

A. Design goals

We aim to develop a solution that supports the following goals:

1) Deployable on today's Internet which excludes approaches requiring changes to middle-boxes, kernels of client machines, the DNS protocol, or the QUIC protocol.
2) Reduces the latency of QUIC's connection establishments that require a prior DNS lookup.
3) Does not make use of IP address spoofing as this practice conflicts with RFC 2827 [18].
4) Supports clients behind Network Address Translators (NAT).
5) Guarantees confidentiality by assuring end-to-end encryption between the client and the web server.
6) Limits the consumption of the proxy's bandwidth.
7) Privacy assurances similar to using a recursive DNS resolver.

B. Design

In this section, we present the protocol flow of a connection establishment using a QuicSocks proxy.

First, the client needs to establish a control channel with the QuicSocks proxy. The client learns via the control channel

which port of the QuicSocks proxy can be used for its connection establishments. A single control channel can be used to establish several QUIC connections via the proxy. Furthermore, the control channel is used by the proxy to validate the client's claimed source address.

Subsequently, the establishment of a single QUIC connection follows the protocol flow shown in Figure 4. Note, that each UDP datagram exchanged between the client and server is encapsulated and carries a request header as common in the SOCKS protocol [12]. To begin the connection establishment, the client sends its QUIC ClientHello message to the Quic-Socks proxy indicating the domain name of the destination server in the SOCKS' request header. Upon receiving this message, the proxy authenticates the client based on the data-grams' encapsulation and caches this message. Subsequently, the proxy does a DNS lookup for the presented domain name and forwards the ClientHello message to the destination's server IP address. Next, the proxy forwards also the obtained DNS response to the client. Note, that the QuicSocks proxy sends all forwarded datagrams from its own source address. Upon receiving the DNS response from the proxy, the client starts probing the direct path to the respective web server to prepare a seamless connection migration to this new path.

Upon receiving the forwarded ClientHello, the server can optionally conduct a stateless retry as shown in Figure 4. In this case, the server returns a retry message and an address validation token to the proxy. Receiving such a request for a stateless retry, the proxy resends the cached ClientHello message and along with the received address validation token. This challenge-response mechanism allows the QUIC server to validate the claimed source address before proceeding with the cryptographic connection establishment. Following the default QUIC handshake, the server proceeds by sending messages including the ServerHello and the FIN, which signals that the server established forward-secure encryption keys. Receiving these messages of the cryptographic connection establishment, the proxy forwards them to the client. Based on these messages, the client validates the server's identity and computes its forward-secure encryption keys. To complete the handshake, the client sends its FIN message via the proxy to the server. Subsequently, the client migrates the established connection towards the direct path between client and server. The connection migration reduces the system utilization of the QuicSocks proxy and possibly leads to shorter round-trip times between client and server.

C. Implementation

The implementation of our proposal aims to demonstrate its real-world feasibility. Our implemented prototype is capable of establishing a connection via the default SOCKS protocol and subsequently migrate the connection to the direct path between QUIC server and client. Our modified client is written in about 350 lines of Rust code and make use of the rust-socks (v0.3.2) and quiche (v0.1.0-alpha3) libraries. Rust-socks provides an abstraction of a SOCKS connection with an interface that is similar to the operating system's UDP sockets and allows to
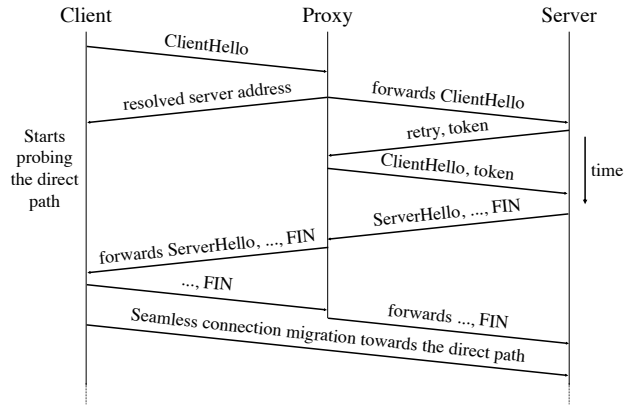

Fig. 4. Protocol flow of a QUIC handshake via the proposed QuicSocks proxy, who resolves the server's IP address. The handshake includes an optional stateless retry initiated by the server. After the server and client established forward secure keys and exchanged FIN messages, the client initiates the connection migration towards the direct path.

transparently use a SOCKS connection. Quiche is an experimental QUIC implementation that separates protocol messages from socket operations which accommodates our use-case of switching between SOCKS sockets and the operating system's UDP sockets within the same QUIC connection. In detail, we modified quiche's example client implementation to perform a QUIC handshake through a rust-socks socket. Once the connection establishment is completed, we switch to a new operating system UDP socket to communicate with the QUIC server over the direct path. Note, that the server's IP address required to conduct this switching is provided by the datagram header of the default SOCKS protocol.

Furthermore, we adapted our client implementation to measure the required time for a connection establishment. The time is measured from the request to establish a connection until the QUIC handshake is completed. In total, we implemented these time measurements for three different connection situations. The first connection situation includes additionally the overhead required to establish the connection with the SOCKS proxy. The second connection situation assumes an established SOCKS connection and measures only the time required to establish a QUIC handshake employing a SOCKS proxy. Finally, the last connection situation conducts a time measurement for a plain QUIC connection establishment without using a SOCKS proxy.

Note, that our prototype does not provide a complete QuicSocks implementation because we did not apply changes to the used SOCKS proxy. As a result, the used proxy does not support the stateless retry mechanism as proposed. Furthermore, our proxy does not provide the client with the resolved QUIC server address directly after the DNS lookup. Instead, within our test setup, the client retrieves the QUIC server address from the SOCKS encapsulation of the forwarded server response. Hence, our client implementation does not start validating the direct path between client and server before migrating the connection.

IV. EVALUATION

In this section, we evaluate the proposed connection establishment via QuicSocks proxies. To begin with, we investigate feasible performance improvements of our proposal compared to the status quo via an analytical model. Then, we conduct latency measurements between clients, servers, and DNS resolvers to approximate real-world delays for QuicSocks proxies that are colocated with the respective DNS resolver. Finally, we present performance measurements using our QuicSocks prototype.

A. Analytical evaluation

The performance benefit of employing a QuicSocks proxy for the connection establishment depends on the network topology. For reasons of clarity, we assume in our analytical model a colocation of the DNS resolver and the QuicSocks proxy (see Figure 5). Furthermore, our model is reduced to the network latency between the involved peers. As shown in Figure 5, we denote the round-trip time between client and DNS resolver/ QuicSocks proxy as RTT_{DNS}. RTT_{direct} and RTT_{Server} describe the round-trip time between server and client, and between server and QuicSocks proxy, respectively.

TABLE I
COMPARISON OF THE REQUIRED NETWORK LATENCY TO RESOLVE A DOMAIN NAME AND ESTABLISH A QUIC CONNECTION USING THE STATUS QUO AND THE PROPOSED QUICSOCKS PROXY.

| Stateless | Latency to establish the connection (incl. DNS) | |
retry	Status quo	Proposal
w/o	$RTT_{DNS} + RTT_{direct}$	$RTT_{DNS} + RTT_{Server}$
with	$RTT_{DNS} + 2 * RTT_{direct}$	$RTT_{DNS} + 2 * RTT_{Server}$

Table I presents the evaluation results for our analytical model. A connection without stateless retry requires RTT_{DNS} to resolve the domain name and subsequently RTT_{direct} to establish the connection between client and QUIC server using the status quo. However, to establish the same connection via a QuicSocks proxy the sum of RTT_{DNS} and RTT_{Server} is required. Note, that we define a connection to be established when the client and the server computed their forward-secure encryption keys and are ready to send application data. Thus, we may count a connection as established before the client's FIN message has been processed by the server. With respect to stateless retries, we observe that the delay of the connection establishments increase for the status quo and our proposal by a RTT_{direct} and a RTT_{Server}, respectively. In total, our analytical model indicates our proposal outperforms the current status quo if RTT_{Server} is smaller than RTT_{direct}. In this case, we find that the reduced delay of the connection establishment without stateless retry is equal to the difference between RTT_{Server} and RTT_{direct}. Moreover, the benefit of our proposal is doubled if the connection establishment requires a stateless retry.

Our proposal achieves its worst performance when the client is colocated with the server. However, the best performance can be realized when the DNS resolver, the QuicSocks proxy, and the server are colocated.
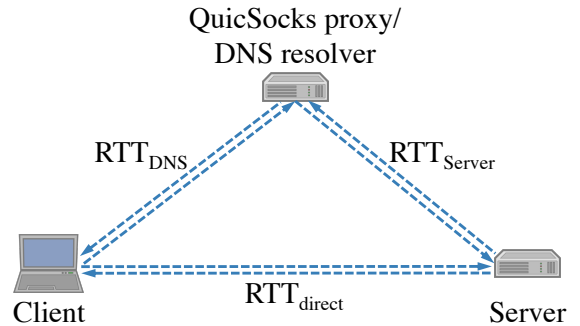

Fig. 5. Network topology with colocation of DNS resolver and QuicSocks proxy. Furthermore, this overview marks the round-trip time (RTT) between different peers.

In the following, we assume an ISP provides a DNS resolver/ QuicSocks proxy half-way, on-path between client and server. Note, that the client's latency to the first IP hop (last mile latency) contributes between 40% and 80% of a typical RTT_{direct} [19]. A typical RTT_{direct} in the LTE mobile network of the U.S. is 60 ms to reach popular online services [20]. For this example, we assume RTT_{DNS} and RTT_{Server} to be each 30 ms, while RTT_{direct} is 60 ms.

TABLE II
COMPARISON OF THE REQUIRED NETWORK LATENCY TO RESOLVE A DOMAIN NAME AND ESTABLISH A QUIC CONNECTION USING THE STATUS QUO AND THE PROPOSED QUICSOCKS PROXY VIA AN AVERAGE U.S. MOBILE LTE CONNECTION.

| Stateless | Latency to establish connection (incl. DNS) | |
retry	Status quo	Proposal
w/o	90 ms	60 ms
with	150 ms	90 ms

Table II provides the results for this example. We find, that our proposal accelerates the connection establishment by 30 ms and 60 ms depending on the requirement of a stateless retry. In summary, the total delay overhead of the connection establishment is reduced by up to 40% and at least 33.3%. Note, that the absolute benefit of our proposal is even higher for 3G networks where RTT_{direct} in the U.S. is on average between 86 ms and 137 ms depending on the mobile network provider [20].

B. Real-world network topologies

Our analytical evaluation indicates, that our proposal can significantly reduce the latency of a QUIC connection establishments with a prior DNS query if the QuicSocks proxy has a favorable position in the network topology. In this section, we investigate real-world network topologies to approximate the feasible performance benefit of QuicSocks proxies when they are colocated with ISP-provided DNS resolvers. We begin by describing our applied methodology to measure real-world network topologies. Subsequently, we evaluate the QuicSocks proposal based on our collected data.

a) Data collection: Our data collection aims to measure RTT_{DNS}, RTT_{Server}, and RTT_{direct} for different real-world clients. We use nodes of the RIPE Atlas network [21] to represent our clients. These RIPE Atlas nodes allow us to conduct custom ping measurements and DNS queries. The selected nodes are in different autonomous systems all over Germany including home networks and data centers. Furthermore, also our test server is in a data center in Germany operated by the Hetzner Online GmbH. The aim of this test setup is to be representative for a typical Internet connections in countries with a similar infrastructure like Germany.

To measure the RTTs between the involved peers, we require the IP address of each peer to conduct corresponding ping measurements. While we have access to the IP address of our clients and the test server, we cannot look up the address of the client's locally configured DNS resolver. Furthermore, the DNS resolvers might use an anycast service for its IP address [22] that may return different physical endpoints when pinged from the client and the server, respectively. We used message 6 in Figure 3, where the recursive resolver sends a request to the authoritative nameserver to learn the IP address of the recursive DNS resolver. In detail, we announced a DNS authority section at our test server for a subdomain such as dnstest.example.com. Then, we conducted a DNS query from the client to a random subdomain in our authority section such as foobar.dnstest.example.com. At the same time, we captured the network traffic on the server and found a DNS query for this subdomain foobar.dnstest.example.com. We reasoned that the sender's address of this DNS query is resolving the client's DNS query. Depending on the DNS setup, the IP address of locally configured DNS resolver might mismatch the address sending the query to the authoritative nameserver. For these cases, we assume that both DNS resolvers are colocated yielding about the same RTT_{DNS} and RTT_{Server} with respect to our measurements.

In total, we used 800 RIPE Atlas nodes in Germany to conduct our data collection on the 13th of June 2019. A successful measurement includes RTT_{DNS}, RTT_{Server}, and RTT_{direct} for the nodes, where we used an average over five ping measurements to determine the respective RTT. In our data collection, we obtained successful results for 650 nodes. Failures can be mainly attributed to DNS resolver that did not respond to ping measurements. However, a small fraction of measurements experienced also failures during DNS measurements.

To focus our data collection on ISP-provided DNS resolvers, we investigated the autonomous system numbers of the observed IP addresses. We assume, that an ISP-provided DNS resolver uses an IP address from the same autonomous system as the node does. This approach allows us to sort out configured public DNS resolvers such as Google DNS which will usually operate from an IP address assigned to a different autonomous system compared to the node. In total, our data collection successfully obtained measurements from 474 nodes in Germany using each an ISP-provided DNS resolver.

b) Results: To accelerate a connection establishment via our proposal, we require RTT_{Server} to be smaller than RTT_{direct}.
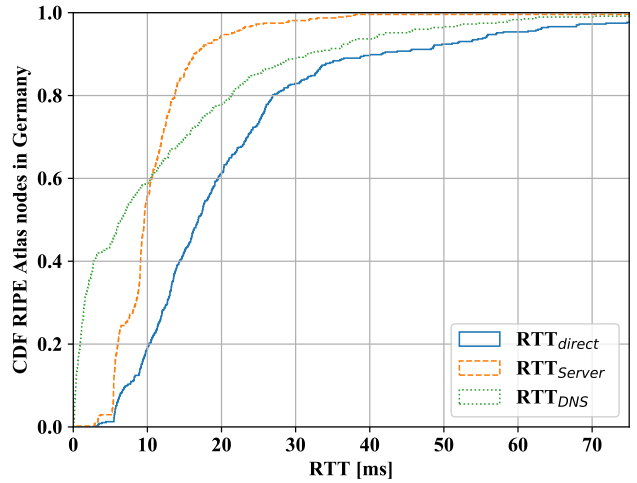

Fig. 6. Cumulative distribution of the RIPE Atlas nodes in Germany using an ISP-provided DNS resolver over RTT_{DNS}, RTT_{Server}, and RTT_{direct}.

Figure 6 provides a cumulative distribution of the RIPE Atlas nodes in Germany using an ISP-provided DNS resolver over the corresponding RTT's. The plot shows RTT_{direct}, RTT_{Server}, and RTT_{DNS} as solid, dashed, and dotted lines, respectively. Our results indicate for almost all clients RTT_{Server} is significantly smaller than RTT_{direct}. For 51% of the considered RIPE Atlas nodes, RTT_{Server} is at least 5 ms smaller than RTT_{direct}. Furthermore, 36.7% of the nodes experience RTT_{Server} to be at least 10 ms smaller than RTT_{direct}. As can be observed in Figure 6, almost no nodes experiences RTT_{Server} to be longer than 40 ms, while a tail of 10% of the respective RIPE Atlas nodes observe a longer RTT_{direct}. In this long tail, we find 7.2% and 3.8% of the nodes to have a RTT_{Server} that outperforms RTT_{direct} by at least 40 ms and 50 ms, respectively. Furthermore, Figure 6 provides a plot of RTT_{DNS}. We find, that 60% of the nodes have a RTT with their ISP-provided DNS resolver of less than 10 ms. Moreover, RTT_{DNS} is almost always smaller than RTT_{direct} for a specific node. This can be explained through RIPE Atlas nodes that are located towards the periphery of the Internet compared to their ISP-provided DNS resolvers holding a position closer to the core of the Internet.

To evaluate our proposal compared to the status quo, we combine the equations provided in Table I with the measured RTT. Figure 7 plots these results as a cumulative distribution of the RIPE Atlas nodes in Germany using an ISP-provided DNS resolver over the required network latency to complete the QUIC connection establishment. In total, Figure 7 contains four plots. In the scenario of a QUIC connection establishment using a stateless retry, the solid and dashed line represent the status quo and our proposed solution, respectively. In the scenario of a QUIC handshake without stateless retry, the status quo and our proposal are marked as dash-dotted and dotted lines, respectively.

In total, our results indicate our proposal accelerates the connection establishment for the great majority of investigated
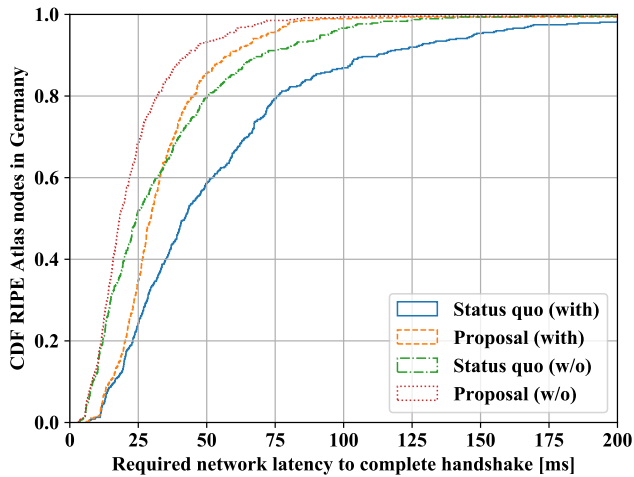
Fig. 7. Cumulative distribution of the RIPE Atlas nodes in Germany using an ISP-provided DNS resolver over the required network latency of the QUIC handshake. The plot compares the status quo versus our proposal for handshakes with and without stateless retry.

RIPE Atlas nodes. Furthermore, we observe the trend that performance improvements are higher for nodes with longer required network latency to complete the handshake. For example, we find that approximately 10% of the nodes save at least 30 ms establishing the connection without stateless retry and 60 ms with a stateless retry. Moreover, 24.3% of the investigated nodes save at least 15 ms without and 30 ms with a stateless retry during the connection establishment. Note, that approximately a third of the nodes experience a faster connection establishment using our proposal in a stateless retry connection establishment than having a status quo handshake without stateless retry.

C. Prototype-based Measurements

In this section, we compare the delay of a default QUIC connection establishment with handshakes using our proposal. However, the performance of our proposal significantly depends on the network topology of our test setup. This measurement neglects network topologies and investigates the delay caused by the computational overhead of introducing a QuicSocks proxy on a network link.

a) Data collection: For our test setup, we use a publicly accessibly QUIC server, a Dante SOCKS proxy (v1.4.2) and our implemented prototype to represent the client. Our proto-type and the Dante SOCKS proxy are run on the same virtual machine. The virtual machine is equipped with 1 vCPU and 0.6 GB RAM and runs Debian 9.9 (Stretch). The colocation of our client implementation with the proxy ensures that measurements using the proxy make use of the same network path as measurements conducted without the proxy. In detail, we conduct three different types of measurements on the 25th of June 2019 of which we repeat each measurement 1 000 times. The *default* measurements do not employ our proxy and investigate the required time to establish a QUIC connection with the server. The *cold start* measurements include the

time required to establish the SOCKS connection and the subsequent QUIC handshake via the proxy. Note, that a single SOCKS connection can be used to establish several QUIC connections. The *warm start* measurement includes the time to establish a QUIC connection via our proxy but excludes the delay incurred by establishing the SOCKS connection.

b) Results: Our data collection provided us with 1 000 values for each of the three measurement types. To evaluate our collected data, we retrieve the minimum and the median value of each measurement type. The *default* measurement has a minimum of 49.145 ms and a median of 51.309 ms. The *warm start* measurement has a minimum of 49.708 ms and a median of 52.471 ms. These values are between 1.1% and 2.3% higher than the *default* measurement. This can be explained by the additional overhead caused by the interaction with the proxy. Furthermore, these values indicate an absolute overhead of using a SOCKS proxy of less than 1.2 ms for the median value, if the SOCKS connection is already established. The *cold start* measurement yields a minimum value of 52.073 ms and a median of 54.772 ms. Comparing both measurements using the SOCKS proxy, we can attribute an additional overhead of about 2.3 ms in our test setup to establish the SOCKS connection. As a result, we recommend clients to early establish their SOCKS connection and to use the *warm start* approach to reduce the delays during their QUIC connection establishments.

V. RELATED WORK

There is much previous work on accelerating connection establishments on the web. For example, Google launched in 2019 its feature Chrome Lite Pages [23]. Lite Pages runs a proxy server that prefetches a website and forwards a compressed version of it to the client. This approach leads to significant performance improvements for clients experiencing high network latencies as they do only need to establish a single connection to the proxy server to retrieve the website. However, as major disadvantages compared to our proposal this leads to a significant load on the proxy server and breaks the principle of end-to-end transport encryption between the client and the web server.

Furthermore, Miniproxy [24] can be used to accelerate TCP's connection establishment. This approach places a proxy between the client and the web server, which doubles the number of required TCP handshakes. Miniproxy can provide a faster TCP connection establishment in case of a favorable network topology and significant RTTs between client and web server. The QUIC protocol includes computationally expensive cryptographic handshakes causing a significant delay compared to TCP's handshake [25]. Therefore, this approach seems less feasible when QUIC is used.

The ASAP [26] protocol piggybacks the first transport packet within the client's DNS query and the DNS server forwards it to the web server after resolving the IP address. However, this approach requires the DNS server to spoof the clients IP address which leads to a violation of the Best Current Practice RFC 2827 [18]. Furthermore, a deployment

of ASAP requires significant infrastructural changes to the Internet because it uses a custom transport protocol.

Further possible performance improvements can be achieved by sending replicated DNS queries to several DNS resolvers and occasionally receiving a faster response [27]. Another DNS-based mechanism aiming to reduce latency uses Server Push [28] where the resolver provides speculative DNS responses prior to the client's query. In total, these approaches tradeoff a higher system utilization versus a possibly reduced latency.

VI. CONCLUSION

We expect high-latency access networks to remain a web performance bottleneck for a significant number of users throughout the forthcoming years. The QUIC protocols aims to reduce the delay of connection establishments on the web. However, our measurements across a wide variety of access networks in Germany indicates a tail of users is affected by significant delays beyond 100 ms to complete a DNS lookup with a subsequent QUIC connection establishment. Our proposal exploits the fact that ISP-provided DNS resolvers are typically located further into the core Internet than clients. We find, that colocating a proxy with the ISP-provided DNS resolver provides significant performance gains for clients on high-latency access networks. For example, a client can delegate the task of DNS lookups to the proxy in a more favorable network position. Furthermore, the QUIC protocol provides features such as connection migration or the concept of stateless retries that allow further performance-optimizations when employing a proxy. We hope that our work leads to an increased awareness of the performance problems experienced by a significant tail of users on high-latency access networks and spurs further research to reduce this web performance bottleneck.

REFERENCES

[1] S. Sundaresan, N. Feamster, R. Teixeira, and N. Magharei, "Measuring and mitigating web performance bottlenecks in broadband access networks," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 213–226.

[2] M. Varvello, J. Blackburn, D. Naylor, and K. Papagiannaki, "EYEORG: A Platform For Crowdsourcing Web Quality Of Experience Measurements," ser. CoNEXT '16. New York, NY, USA: ACM, 2016, pp. 399–412.

[3] S. Souders. (2009) Velocity and the Bottom Line. [Online]. Available: http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html/

[4] M. Bishop, "Hypertext Transfer Protocol Version 3 (HTTP/3)," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-http-20, Apr. 2019, work in Progress.

[5] J. O'Callaghan. (2019) Blue Origin To Launch Satellites For Company Battling SpaceX And Others For Space Internet Supremacy. [Online]. Available: https://tinyurl.com/y2damy3m

[6] K. Bode. (2017) OneWeb's Low-Latency Satellite Broadband Plan Gets FCC Approval. [Online]. Available: https://www.dslreports.com/shownews/OneWebs-LowLatency-Satellite-Broadband-Plan-Gets-FCC-Approval-139833

[8] E. Sy, "Surfing the Web quicker than QUIC via a shared Address Validation," 2019.

[7] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-transport-20, Apr. 2019, work in Progress.

[9] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar *et al.*, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 183–196.

[10] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?" ser. SIGCOMM '11. ACM, 2011, pp. 181–194.

[11] S. Hätönen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, and M. Kojo, "An experimental study of home gateway characteristics," ser. SIGCOMM '10. ACM, 2010, pp. 260–266.

[12] M. D. Leech, "SOCKS Protocol Version 5," RFC 1928, Mar. 1996.

[13] H. Kitamura, "A SOCKS-based IPv6/IPv4 Gateway Mechanism," RFC 3089, Apr. 2001.

[14] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," Naval Research Lab Washington DC, Tech. Rep., 2004.

[15] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS performance and the effectiveness of caching," *IEEE/ACM Transactions on networking*, vol. 10, no. 5, pp. 589–603, 2002.

[16] E. Sy, M. Moennich, T. Mueller, H. Federrath, and M. Fischer, "Enhanced Performance for the encrypted Web through TLS Resumption across Hostnames," 2019.

[17] L. Zhu, Z. Hu, J. Heidemann, D. Wessels, A. Mankin, and N. Somaiya, "Connection-oriented DNS to improve privacy and security," in *2015 IEEE symposium on security and privacy*. IEEE, 2015, pp. 171–186.

[18] D. Senie and P. Ferguson, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," RFC 2827, May 2000.

[19] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè, "Broadband Internet Performance: A View from the Gateway," ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 134–145.

[20] OpenSignal. (2018) USA Mobile Network Experience Report January 2019 . [Online]. Available: https://www.opensignal.com/reports/2019/01/usa/mobile-network-experience

[21] Reseaux IP Europeens Network Coordination Centre. (2019) RIPE Atlas – Internet measurement network. [Online]. Available: https://atlas.ripe.net/

[22] W. Milliken, T. Mendez, and D. C. Partridge, "Host Anycasting Service," RFC 1546, Nov. 1993.

[23] Greenstein, Ben and Gao, Nancy. (2019) Chrome Lite Pages - For a faster, leaner loading experience. [Online]. Available: http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

[24] G. Siracusano, R. Bifulco, S. Kuenzer, S. Salsano, N. B. Melazzi, and F. Huici, "On the Fly TCP Acceleration with Miniproxy," in *Proceedings of the 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, ser. HotMIddlebox '16. New York, NY, USA: ACM, 2016, pp. 44–49.

[25] E. Sy, T. Mueller, C. Burkert, H. Federrath, and M. Fischer, "Enhanced Performance and Privacy for TLS over TCP Fast Open," 2019.

[26] W. Zhou, Q. Li, M. Caesar, and P. B. Godfrey, "ASAP: A Low-latency Transport Layer," ser. CoNEXT '11. New York, NY, USA: ACM, 2011, pp. 20:1–20:12.

[27] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low Latency via Redundancy," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. New York, NY, USA: ACM, 2013, pp. 283–294.

[28] P. E. Hoffman and P. McManus, "DNS Queries over HTTPS (DoH)," RFC 8484, Oct. 2018.