Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Enhanced Performance and Privacy for Core Internet Protocols

Kumulative Dissertation

zur Erlangung des akademischen Grades

Dr. rer. nat.

an der Fakultät für Mathematik, Informatik und Naturwissenschaften der Universität Hamburg

eingereicht beim Fach-Promotionsausschuss Informatik von

Erik Sy
aus Pasewalk

November 2019

Gutachter:

Prof. Dr.-Ing. Hannes Federrath

Prof. Dr-Ing. Claudia Diaz

Tag der Disputation:

18. August 2020

Danksagung

Diese Dissertation ist zwischen 2016 und 2019 am Arbeitsbereich Security and Privacy der Universität Hamburg entstanden. An dieser Stelle möchte ich mich bei allen bedanken, die zu ihrem Gelingen beigetragen haben.

Besonderer Dank gilt meinem Doktorvater Hannes Federrath, der mich bei der Erstellung dieser Dissertation begleitet hat. Er hat mit mir die Zuversicht geteilt, dass ich nach meinem Studium der Physik erfolgreich in der IT-Sicherheit forschen werde.

Mein Dank gilt ferner Christian Burkert, Mathias Fischer, Dominik Herrmann, Moritz Mönnich, Tobias Müller, und Matthias Marx für die tolle Zusammenarbeit beim Verfassen von Forschungsbeiträgen.

Ich bedanke mich auch bei meinen Kollegen die ich während meiner Promotionszeit kennen und schätzen gelernt habe: Maximilian Blochberger, Britta Böhm, Doganalp Ergenc, Steffen Haas, Malte Hamann, Stefanie Jasser, David Jost, Jens Lindemann, Sadaf Momeni, Johanna Nehring-Ansohn, Tom Petersen, Dimitra Pons, Henning Pridöhl, Eugen Ruppert, Monina Schwarz, Nurefsan Sertbas, Bahareh Shojaie, Marius Stübs, Florian Wilkens und Ephraim Zimmer.

Schließlich bedanke ich mich bei meiner Familie, die mich nach Kräften unterstützt und bestärkt hat. Mein ganz besonderer Dank gilt meiner Partnerin Lisa und meinen Kindern Malou und Bela. Sie haben mich intensiv während des Entstehens dieser Arbeit begleitet und mich, immer wenn es nötig war, wieder auf andere Gedanken gebracht.

Zusammenfassung

Der Datenverkehr im Internet besteht zu einem Großteil aus kurzen Datenübertragungen, welche die Protokolle Domain Name System (DNS), Transmission Control Protocol (TCP), Transport Layer Security (TLS) oder QUIC nutzen. Im Rahmen dieser Dissertation werden Möglichkeiten untersucht, wie diese Protokolle im Hinblick auf Datenschutz, Ressourcenverbrauch sowie Latenz für den Verbindungsaufbau optimiert werden können.

Zu den Ergebnissen dieser Dissertation im Bereich DNS gehört die Vorstellung eines Designs zur Auflösung von Domainnamen in IP-Adressen, bei dem der Nutzer relevante DNS-Einträge nicht nur von DNS-Resolvern sondern ebenfalls direkt von Webservern erhalten kann. Dieser Ansatz kann dem Nutzer traditionelle DNS-Anfragen ersparen und damit den Aufruf von Websites erheblich beschleunigen. Ferner, ermöglicht dieser Ansatz einen besseren Schutz von Nutzerdaten, weil der verwendete DNS-Resolver deutlich weniger DNS-Anfragen vom Nutzer einsehen kann.

Die Protokolle TCP, TLS und QUIC wurden im Rahmen dieser Dissertation auf Datenschutzprobleme untersucht. Jedes dieser Protokolle erlaubt ein mehrtägiges Tracking von Onlineaktivitäten eines Internetnutzers. Im Fall der Fast Open Erweiterung des TCP-Protokolls sind die Datenschutzprobleme besonders gravierend, weil der ausgenutzte Trackingmechanismus im Kernel der Betriebssysteme verankert ist und sich der Kontrolle durch Nutzeranwendungen weitgehend entzieht. Als Reaktion auf dieses Datenschutzproblem wurde u.a. im Browser Firefox die Unterstützung für diese TCP-Protokollerweiterung eingestellt. Zur konstruktiven Lösung der identifizierten Datenschutzprobleme wurden jeweils Verbesserungsmöglichkeiten vorgestellt, evaluiert und gegebenenfalls implementiert. Als Gegenmaßnahme zum erwähnten Problem von TCP Fast Open wurde das TCP Fast Open Privacy Protokoll eingeführt, welches unter anderem einen TLS-verschlüsselten Datenkanal einsetzt, um Tracking auf Basis von beobachteten Datenverkehr zu verhindern.

Zur Steigerung der Performanz von TCP, TLS und QUIC werden in dieser Dissertation zahlreiche Forschungsbeiträge vorgestellt. Beispielsweise konnte für den Aufbau von TLS-Verbindungen während des Aufrufs einer durchschnittlichen Website ein Einsparpotential von 44% der benötigten CPU-Berechnungen und bis zu 30,6% der benötigten Zeit aufgezeigt werden. Diese erhebliche Performanzsteigerung resultiert aus dem Umstand, dass oft TLS-Verbindungen zu anderen Hostnamen mit einem optimierten Verbindungsaufbau basierend auf TLS Session Resumptions durchgeführt werden können.

Insgesamt stellt diese Dissertation mehrere Forschungsbeiträge vor, um dem Ziel von datenschutzfreundlichen und performanten Datenübertragungen im Internet etwas näher zu kommen.

Abstract

Most flows on the Internet are short transfers using the protocols Domain Name System (DNS), Transmission Control Protocol (TCP), Transport Layer Security (TLS), or QUIC. This thesis investigates opportunities to improve these protocols regarding their privacy protections, the consumed resources, and the delay caused by the connection establishment.

The results of this thesis comprise in the scope of DNS the presentation of a design allowing web servers to push relevant DNS records to their clients. This novel approach can save the client traditional DNS lookups, which would add additional delay to the resolving of domain names into IP addresses. Concerning privacy protections, this design improves the privacy posture of the client with respect to the traditional DNS resolver because it reduces the number of DNS lookups that can be observed by the DNS resolver.

Furthermore, this thesis studies privacy threats in the protocols TCP, TLS, and QUIC that each can be exploited to track a user's online activities for several days. In the case of the Fast Open extension of the TCP protocol the identified privacy threats are especially serious because this tracking mechanism is implemented in the kernel of the operating systems and user applications have difficulties to restrict it. As a reaction to this detected tracking mechanism, the Mozilla Foundation deactivated this protocol extension in all their products including the popular Firefox browser. To mitigate the presented privacy problems, this thesis introduces, evaluates and, where appropriate, implements countermeasures. The issues of TCP Fast Open are addressed by the proposed TCP Fast Open Privacy protocol, which uses for this purpose a cross-layer approach combining TLS and TCP. Thus, a transport-encrypted data channel is used to prevent tracking based on traffic analysis.

Moreover, in the scope of this thesis performance-optimizations for TCP, TLS, and QUIC are investigated. For example, savings of 44% of the required CPU computations and up to 30.6% of the required elapsed time are demonstrated for the TLS connection establishments during the retrieval of an average website. This improvement becomes feasible when efficient TLS session resumption handshakes are used across different hostnames.

In summary, this thesis provides several contributions towards the goal of secure, privacy-friendly and well-performing transactions on the Internet.

Contents

1 Introduction

The Web exhibits a growing adoption of Hypertext Transfer Protocol Secure (HTTPS) traffic [6]. HTTPS encrypts the content of a communication and thus ensures a basic level of security and privacy. After the widespread online surveillance by the National Security Agency (NSA) came to public attention [11], efforts arose to improve the protection of communications on the Internet. For example, browsers such as Firefox and Google Chrome started deploying DNS over HTTPS (DOH) [9] to securely perform their DNS resolution. This thesis aims to contribute to these efforts by investigating core Internet protocols. It focuses on the protection of the client's privacy against mass surveillance and online tracking. Furthermore, to enhance the user experience on the web, this thesis studied performance limitations of the investigated protocols.

1.1 Problem Statement

This thesis investigates the design and implementation of core Internet protocols to identify existing privacy leaks and performance problems with the aim to mitigate these issues. To illustrate the privacy and performance problems at hand, this section introduces in the following three different use cases. The first two use cases focus on the real-world exploitation of privacy leaks in core Internet protocols, while the last use case describes the negative impact of performance limitations in the investigated protocols.

Use Case 1: Monitoring User Activities by Online Services and DNS Resolvers

Many online services are collecting as many information on their users as possible. The number of visits, the rate of new visitors, and the time spent by users at an online service are examples for metrics that allow to evaluate the popularity of an online service. However, to analyze the audience of an online service, a mechanism is required that allows distinguishing visits from different users and identifying visits from the same user. Privacy leaks of investigated network protocols can allow this identification of users without the user's consent.

Furthermore, online services and DNS resolvers may use such tracking mechanisms in the deployed protocols to create profiles of users' online activities. Related work indicates that some online advertising networks aggregate user profiles across several websites to personalize the displayed advertisements [4]. For example, Google's advertising network can link a user's visits across almost 80% of the Alexa Top Million Websites [13]. Tracking mechanisms within core Internet protocols can facilitate this user tracking and may enable various entities to obtain detailed insights into a user's online activities.

A single tracking mechanism within the HTTPS protocol stack can be sufficient to enable user tracking on the web. In this thesis, it is assumed that the users mitigate well known tracking mechanisms such as HTTP cookies or browser fingerprinting [2]. For such users novel tracking mechanisms in standard Internet protocols pose a serious thread to their privacy as there is no opt-out.

Previous studies [5, 16] indicate that some online trackers are early adopters of new tracking mechanisms. For example, scripts for browser fingerprinting in the wild have been found to contain information leaks of new browser APIs such as the Battery API or AudioContext API [5, 16]. To protect the user's privacy, browsers such as Firefox and Safari increasingly deploy methods to limit the feasibility of known tracking mechanisms such as HTTP cookies [8]. As a result, online trackers are incentivized to deploy new tracking mechanisms to continue their user tracking.

Use Case 2: Network Forensics on Internet Traffic

While the content of HTTPS traffic is encrypted, the analysis of this traffic can provide metadata such as the identities of the communications partners and their communication behavior. Thus, based upon such an analysis entities with access to this network traffic such as local network administrators, the Internet Service Provider (ISP), or law enforcement agencies can gain detailed insights into a user's online activities. Previous studies indicate that some ISPs aggregate user profiles based upon observed network traffic for advertising purposes [33, 34]. Furthermore, the Snowden revelations documented a worldwide wiretapping of network traffic by the NSA [11].

To protect the user's privacy against these threats, the information leakage of the HTTPS protocol stack needs to be minimized. For that, these protocols should transmit sensitive information with transport encryption whenever this is feasible. For example, TLS is progressing in this direction by encrypting the host name of a requested website [21]. Therefore, it becomes more difficult to collect meaningful data from analyzing HTTPS traffic.

Closing these potential information leaks of the HTTPS protocol stack becomes more relevant as also DNS is moving towards an encrypted data transmission [36, 9]. Consequently, the DNS can also protect information like the requested host name. However, the encryption of this information does not protect against active attacks like website fingerprinting that identify the requested website based upon its characteristic network traffic patterns [17].

Use Case 3: Performance Issues of Core Internet Protocols

The time required to retrieve an online resource can vary significantly, especially if the corresponding clients are distributed across the globe. Reasons for this include the network latency between the communicating peers and possible cache misses at the configured recursive DNS resolver of clients DNS. Significant delays impact the user's quality of experience during web browsing [35] and have a negative impact on the per-user revenue of online service provider [22]. Thus, it is an important goal for the Internet community to optimize the performance of web browsing.

1.2 Research Questions

This thesis addresses several research questions, which are relevant with respect to the described problem statement. The first two areas of research questions aim to identify and subsequently mitigate tracking mechanisms based on core Internet protocols. The third area of research questions is concerned with performance-optimizations for the investigated protocols.

1: Questions on User Tracking via Core Internet Protocols

To what extent can core Internet protocols such as DNS, TCP, TLS, QUIC or their extensions be exploited by an adversary to link several connections to the same user? What are the privacy risks originating from the identified tracking mechanisms? What are the limitations of these tracking mechanisms? Does the default configuration of popular web browsers or operating systems enable user tracking based on the investigated mechanisms?

2: Questions on Privacy Protections for Core Internet Protocols

What measures are necessary to restrict and potentially completely avoid user tracking via core Internet protocols? Are these measures applicable under real-world constraints? Do these countermeasures impact the security or the performance of the corresponding protocol? How effectively do the considered countermeasures prevent the linking of several connections to the same user?

3: Questions on Enhancing the Performance of Core Internet Protocols

What measures can be applied to accelerate connection establishments on the web? How do the identified measures impact the security and privacy properties of the investigated protocols? What are the limitations of these performance enhancements?

This thesis addresses these research questions within several publications. The deployed research methodology to provide answers to these research questions is presented in the next section.

1.3 Research Methodology

To enhance the performance and privacy of the investigated Internet protocols this thesis mainly applies the research methods privacy and security analysis as well as prototyping and simulation experiments. In the following, these research methods are briefly introduced.

Privacy and Security Analysis

In this thesis, I used this research method to analyze vulnerabilities of and threats to communication protocols and information systems [1]. For this purpose, I defined an attacker model for the investigated communication protocol or information system. Subsequently, I determined the level of privacy and security risks the investigated system is exposed to by reviewing the threats to and vulnerabilities of this system. This step includes the review of the standards describing the investigated protocols. Furthermore, I conducted experiments with implementations of these protocols to further assess investigated risk. The conducted experiments include browser measurements, measurements of kernel implementations and scans of server implementations deployed on the Internet. Beyond this, I analyzed also behavioral patterns in real-world DNS traffic to better understand the resulting privacy risks of an investigated threat. A threat presents a great risk for the user's privacy, if it allows a unique identification of a peer, enables long tracking periods, and is persistent. Furthermore, the risk assessment of a novel threat/ vulnerability includes a comparison to known risks. In the field of privacy research, this includes a comparison to known tracking mechanisms. The result of such a risk assessment can be used to identify feasible countermeasures.

Prototyping and Experiments via Simulations and Measurements

In this thesis, the research methods of prototyping and experiments via simulations and measurements are deployed to investigate the performance properties of the proposed protocol designs [19]. In the scope of this thesis, a prototype of the proposed TCP Fast Open Privacy protocols and the QuicSocks proxy has been implemented. Following this, performance measurements have been conducted comparing these demonstrators versus the status quo. A prerequisite of a computer simulations is a formal model of the investigated problem. Within the context of this thesis, several computer simulations are used to simulate the performance impact of proposals on connection establishments using TCP, TLS, or the QUIC protocol. Some of these simulations required prior empirical measurements to derive an adequate formal model. For example, I investigated first the real-world server load-balancing of hostnames supporting the TFO protocol. Once this investigation was concluded, it was possible to use this characteristic load-balancing behavior to evaluate the performance of the proposed TCP Fast Open Privacy protocol in such a scenario. The aim of conducting these simulations was to compare the proposals versus the status quo.

1.4 Contributions

This section summarizes the contributions of this thesis. The contributions are grouped according to the three identified research questions. First, I present the novel tracking mechanisms that have been investigated within the scope of this thesis. Second, I summarize contributions on enhanced privacy protections for DNS, TCP, TLS, and QUIC. Finally, I introduce identified performance improvements for the investigated core Internet protocols.

1.4.1 Investigation of Novel Tracking Mechanisms

The first research question (see Sec. 1.2) addresses the identification of novel tracking mechanisms using core Internet protocols. In the scope of this thesis, tracking mechanisms within the Fast Open extension of TCP, within TLS and within QUIC have been identified and investigated. In the following of this section, a description of the investigated tracking mechanisms is provided.

TCP: In the context of this thesis tracking via Fast Open cookies is described [30]. Network-based observers and online services can use these cookies to track a user's online activities. Tests of this tracking mechanisms with popular browsers such as Chrome, Firefox and Opera indicate that tracking periods with this mechanism are unrestricted. Moreover, the conducted measurements suggest that tracking is feasible across private browsing modes, browser restarts and different applications on the same host. Online tracker can deploy this mechanism as a third-party to track users across different websites.

TLS: This thesis reports tracking via TLS session resumption mechanisms [26]. This approach enables network-based observers and online services to link a user's online activities via TLS session identifiers and session tickets in TLS versions prior to 1.3. TLS 1.3 [20] mitigates user tracking by network-based observers. However, online services can still conduct user tracking via the provided session resumption mechanism. Conducted browser measurements suggest that browsers restrict the lifetime of these TLS session resumption mechanisms. This thesis presents a prolongation attack that allows extending tracking periods beyond the session resumption lifetime of the user agent. Based upon these measurements, online tracker can utilize these mechanisms for third-party tracking against most popular browsers.

QUIC: This thesis presents two independent tracking mechanisms within Google's QUIC protocol [27]. Tracking via QUIC's *server config* enables network-based observer and online services to link a users online activities. The second tracking mechanism deploys QUIC's *source-address token*. User tracking via the latter mechanism is only feasible for online services. Browser measurements indicate, that both tracking mechanisms are unrestricted with respect to feasible tracking periods. Furthermore the conducted tests suggest, that online trackers can deploy these mechanisms for third-party tracking to link user activities across different websites.

In summary, a main contribution of this thesis is the investigation of tracking mechanism in two popular HTTPS protocol stacks as shown in Figure 1.1.

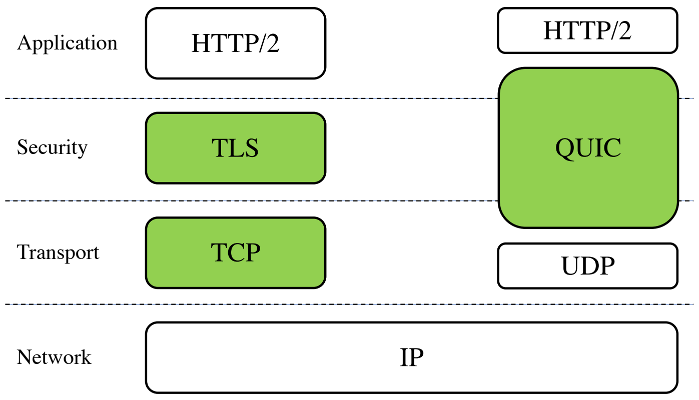


Figure 1.1: Overview of popular HTTPS protocol stacks. The green colored protocols contain tracking mechanisms that have been investigated within the scope of this thesis.

1.4.2 Design, Implementation and Evaluation of Privacy Protections

The second research question of this thesis is concerned with privacy protections for core Internet protocols. To address this question, mitigations against the investigated tracking mechanisms are provided. Furthermore, this thesis presents privacy enhancements for the DNS protocol that improve the user's privacy posture towards its' DNS resolver. The contributions presented in this thesis for the privacy protections in DNS, TCP, TLS, and QUIC are summarized in the paragraphs below.

DNS: The DNS lookups of a client allow the DNS resolver to monitor a large fraction of the client's online activities. To protect clients against this monitoring, this thesis presents resolver-less DNS [23]. Here, web servers push relevant DNS records to their clients via HTTP response header. Subsequently, the client can save a traditional DNS lookup, if the DNS record from the web server is used instead. This approach reduces the number of DNS lookups that can be monitored by the client's DNS resolver and therefore improves the privacy posture of the client. Note that resolver-less DNS requires occasionally a fallback DNS lookup to bootstrap the first connection to a web server and to validate a fraction of DNS records retrieved from a web server. Thus, this proposed privacy protection still enables a fallback DNS resolver to monitor a diminished fraction of the client's online activities.

TCP: To address tracking via TCP Fast Open, the novel TCP FOP protocol is presented in the scope of this thesis [30]. TCP FOP is a cross-layer protocol and deploys an encrypted TLS channel to send Fast Open cookies from the server to the client. In combination with a policy to not reuse Fast Open cookies to establish several connections, this countermeasure prevents network-based observers to link online activities to the same user based upon the observed Fast Open cookies. With respect to tracking by online services a performance versus privacy tradeoff exists. Thus, the most effective privacy protection is achieved by not reusing Fast Open cookies, which prevents abbreviated handshakes. To balance both, a restricted lifetime of cached Fast Open cookies and restrictive scenarios for their reuse can be considered. The impact of the lifetime of these tracking mechanisms on the rate of abbreviated handshakes when revisiting a website is studied with a DNS dataset to derive real-world browsing behavior. The results of this analysis indicate that a lifetime of 10 minutes does achieve a rate of 27.7% of abbreviated handshakes, while a lifetime of 60 minutes leads to 48.3% of abbreviated handshakes. These results suggest that short lifetimes for this tracking mechanism can contribute to significant rates of abbreviated connection establishments. One example for a restrictive scenario can be that hosts which are not allowed to use legitimate tracking methods like HTTP cookies are also restricted with respect to the reuse of Fast Open cookies.

TLS: A first step to mitigate user tracking via TLS session resumption, presents the deployment of the latest TLS version 1.3 [26]. Compared to previous TLS versions TLS 1.3 uses single-use session tickets that mitigate user tracking based on traffic analysis. However, the problem of tracking by online services remains a privacy threat even for this TLS version. TLS 1.3 provides a zero round-trip time (0-RTT) and a 1-RTT mode to resume previous TLS sessions. The initial TLS 1.3 connection establishment requires a latency of 1-RTT and does not enable user tracking via TLS session resumption. However, the resuming of a previous session causes less computational costs than an initial connection establishment [29]. Consequently, a privacy versus computational cost tradeoff takes place when the 1-RTT resumption handshake is used instead of an initial connection establishment. The 0-RTT resumption handshake saves a round-trip time and has reduced computational costs compared to the TLS 1.3 initial handshake. However, the 0-RTT mode has lower security guarantees by being vulnerable to replay attacks and not providing forward secrecy. Thus, a performance versus privacy plus security tradeoff happens between the 0-RTT resumption handshake and the TLS 1.3 initial handshake. To balance both tradeoffs at hand, a restricted lifetime of the session resumption mechanism and restrictive scenarios for the use of these mechanisms can be deployed.

QUIC: QUIC enables user tracking via two independent mechanisms [27]. Source-address tokens enable online services to track a user's online behavior while they allow to save one round-trip time during the connection establishment. Therefore, this mechanism presents a performance versus privacy tradeoff which can be balanced by restricting the lifetime of these cached token and limiting the scenarios that permit the usage of these cached tokens. QUIC's *server config* can be regarded as a public information. Therefore, the mitigation of this tracking mechanism can be achieved by requiring a group of users to utilize the same server config. These groups of users make up an anonymity set [18] and the larger these groups are, the smaller is probability to link a user visit to the correct user. To ensure that a large user group utilizes the same *server config*, a system that logs all *server configs* which users observe from a specific hosts can be used. Note that this presents a similar approach as the established Certificate Transparency Logs [12]. Subsequently, tracking by an online service can be identified as this service issues a large number of *server configs* within short periods. Thus, as a countermeasure against these tracking online services, a user may disable abbreviated handshakes when connecting to these hosts which are known to conduct tracking via this mechanism.

1.4.3 Enhancing the Performance of Core Internet Protocols

The third research question of this thesis aims to improve the performance of core Internet protocols. In detail, this thesis includes proposals on performance improvements for DNS, TCP, TLS, and QUIC. A common goal of these proposals presents the reduction of the delay caused by connection establishments on the web. A summary of the contributions to each investigated protocols is described in the paragraphs below.

DNS: About 25% of the average DNS transactions cause delays between 10 ms and 1 s [3]. Thus, DNS lookups can significantly impact the performance of short web flows. In the scope of this thesis, resolver-less DNS is proposed to address this performance problem [23]. A client receiving a DNS record via resolver-less DNS can directly start a connection establishment to the provided destination without a prior DNS lookup. Hence, the performance gain of resolver-less DNS can be as large as the delay caused by a DNS lookup of the given host using a fallback DNS mechanism. Experiments conducted within the context of this thesis indicate that the performance gain is at least 80 ms per DNS lookup for the one percent of users with the highest network latency towards their DNS resolvers.

TCP: Measurements presented in this thesis indicate that for the TCP Fast Open (TFO) protocol in average 40% of the first reconnection to the same domain name fail to conduct an abbreviated handshake. The reason for this is that TFO does not anticipate load balancing across different IP addresses. To overcome this real-world performance limitations of TCP Fast Open, the TCP Fast Open Privacy (FOP) protocol is introduced in this thesis [30]. Compared to TFO, the novel TCP FOP protocol is aware of domain names as they are required for the server authentication in the context of TLS. This enables TCP FOP to cache a Fast Open cookie with respect to the domain name in which context the cookie was retrieved by the client. Subsequently, to set up a fresh connection to a host with the same domain name, the client utilizes the cached Fast Open cookie in an abbreviate handshake attempt independently of the server IP address. This practice allows to save a round-trip time upon every reconnection to hosts with the same domain name and therefore overcomes the described performance limitation of TFO. To support this performance enhancement, servers that provide resources for the same domain name need to mutually accept their issued Fast Open cookies.

TLS: In the scope of this thesis performance limitations related to the TLS session resumption mechanism are investigated [29]. In detail, TLS 1.3 recommends to conduct session resumptions only for reconnections to the same hostname. However, conducted measurements investigating popular websites indicate that session resumptions are feasible across different hostnames sharing the same TLS certificate. This work includes a description of a new TLS extension enabling the server to inform the client about the possibility of resuming the current connection state with hinted other hostnames. The evaluation of this extension for the Alexa Top Thousand Sites yields, that this proposal reduces the number of required full handshakes by 58.7% upon the first website visit. Due to the fact, that resumed connection establishments have a lower computational overhead, this saves on average about 44% of the required CPU time to establish the connections required to retrieve a website. Moreover, resumed handshakes are faster established than initial connections. The results presented in this thesis suggest, that this extension saves up to 30.6% of the time required to establish the website's TLS connections compared to the current status quo.

QUIC: This thesis contains three papers [24, 28, 32] dedicated to accelerating QUIC's connection establishments. Two of these papers [24, 28] focus on improvements of QUIC's address validation mechanism. The third paper [32] contributes a novel QuicSocks proxy design that allows accelerating connection establishment on high-latency access networks. QUIC's address validation mechanism uses address validation tokens to avoid the delay overhead of an additional round-trip during the connection establishment. Currently, it presents a performance limitation that a QUIC server serving *example.com* cannot issue an address validation token to the client to be used for its trusted subdomain *static.example.com*. To overcome this limitation, a new transport parameter for the QUIC protocol is proposed that allows a server to indicate other hostnames where an issued address validation token can be used to perform an abbreviated handshake. The second paper focuses on the performance problem that the first connection to a QUIC server still requires an additional round-trip to conduct the address validation because the client has not yet retrieved an address validation token. To address this problem, out-of-band validation tokens are proposed that can be issued by trusted entities and are validated by the consuming QUIC server during the connection establishment. A proposed example for such trusted entities are DNS resolvers that issue the respective out-of-band validation tokens during the client's DNS lookup. In total, the proposed approaches regarding QUIC's address validation mechanism allow saving a round-trip time for almost all connection establishments enforcing a strict address validation. The third paper investigates the performance of QUIC's connection establishment on high-latency access networks such as satellite links. It proposes to colocate a novel QuicSocks proxy with the ISP-provided DNS resolver and assumes that this network topology leads to shorter network latencies between QUIC server and QuicSocks proxy compared to the latency between client and server. The client can now send initial handshake messages without a prior domain name resolution to the QuicSocks proxy because the proxy conducts the respective DNS lookup on behalf of the client. Subsequently the proxy forwards the received initial messages to the corresponding QUIC server. This practice accelerates the connection establishment because the client does not need to conduct a prior DNS lookup before sending its initial handshake messages. A QUIC server may respond to the clients initial messages with a retry message together with an address validation token. This retry message instructs the client to resend its initial handshake messages together with the received address validation token. The design of the QuicSocks proxy allows this proxy to directly respond to the retry message without forwarding this message to the client. This contributes to further reductions of the delay of the connection establishment. After the connection establishment is completed, the client uses QUIC's feature of connection migration to migrate the QUIC connection to the direct path between client and server. The conducted performance evaluation of the proposal indicates that 10% of the 474 test nodes distributed across ISP's in Germany saved at least 30 ms per QUIC connection establishment by using the proposed design.

1.5 Structure of this Thesis

This thesis is divided into six chapters, as shown in Figure 1.2. The core of this cumulative thesis are the Chapters 2 to 9, each focusing on performance and privacy enhancements for the Internet protocols DNS, TCP, TLS and QUIC, respectively.

The focus of Chapter 2 lies on DNS. In detail, resolver-less DNS is introduced that allows web servers to push DNS records to their clients. This approach allows clients to save the DNS lookup using their traditional DNS resolver. As a result, this design allows saving the delay caused by DNS lookups and strengthens the user's privacy posture towards the traditional DNS resolver.

Chapter 3 focuses on TCP. It introduces the Fast Open extension of TCP and reports performance and privacy deficits in this protocols. To mitigate the identified problems, the new Fast Open Privacy extension is proposed, implemented and evaluated.

Findings related to TLS are presented in Chapters 4 to 5, which consist of two publications. The first publication introduces the session resumption mechanism of TLS and describes and evaluates the privacy problems caused by this mechanism. Subsequently, countermeasures are reviewed and their feasibility is evaluated based on real-world user traffic. The second publication addresses performance limitations of the TLS session resumption mechanism. It introduces a new TLS extension to enable the efficient use of resumed connection establishments across different hostnames. Afterward, it evaluates the performance benefit of the proposed TLS extension based on the page loading behavior of the Alexa Top Thousand websites.

The Chapters 6 to 9 summarize results related to the QUIC protocol and consists of four publications. The first publication introduces QUIC's connection establishment and analyzes user tracking via QUIC's server config and source-address token. Subsequently, it reviews feasible countermeasures against these tracking mechanisms. The second and third publication within this chapter focus on performance optimizations concerning the source address validation of QUIC servers using the stateless retry mechanism. The last publication in this chapter aims to provide faster connection establishments for clients on high-latency access networks using a proposed QuicSocks proxy.

Chapter 10 concludes this thesis and provides an outlook on possible future work in the performance and privacy research on Internet protocols.

1.6 List of Publications

In the context of my research, ten research papers have already been produced. The research questions of this thesis are answered in a frame of eight publications. The following six publications have been accepted by international journals or proceedings of international conferences which pursue a strict peer-review:

1. Erik Sy, Tobias Mueller, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2019. Enhanced Performance and Privacy for TLS over TCP Fast Open. *Proceedings on Privacy Enhancing Technologies 2020*, 2 (2020), Accepted for publication.

2. Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2018. Tracking Users Across the Web via TLS Session Resumption (**Outstanding Paper Award**). In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC '18).* ACM, New York, NY, USA, 289–299. https://doi.org/10.1145/3274694.3274708

3. Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2019. A QUIC Look at Web Tracking. *Proceedings on Privacy Enhancing Technologies 2019*, 3 (2019), 255 – 266. https://content.sciendo.com/view/journals/popets/2019/3/article-p5.xml

4. Erik Sy. 2019. Surfing the Web quicker than QUIC via a shared Address Validation (**Best Student Paper Award**). In *The 27th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2019).* Split, Croatia.

5. Erik Sy, Christian Burkert, Tobias Mueller, Hannes Federrath, and Mathias Fischer. 2019. QUICker Connection Establishment with Out-Of-Band Validation Tokens. In *2019 IEEE 44th Conference on Local Computer Networks (LCN) (LCN 2019).* Osnabrück, Germany.

6. Erik Sy, Tobias Mueller, Moritz Moennich, and Hannes Federrath. 2019. Accelerating QUIC's Connection Establishment on High-Latency Access Networks. In *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC) (IPCCC 2019).* London, United Kingdom (Great Britain).

Additionally, this thesis includes the following two articles that are currently not published by international journals or proceedings of international conferences.

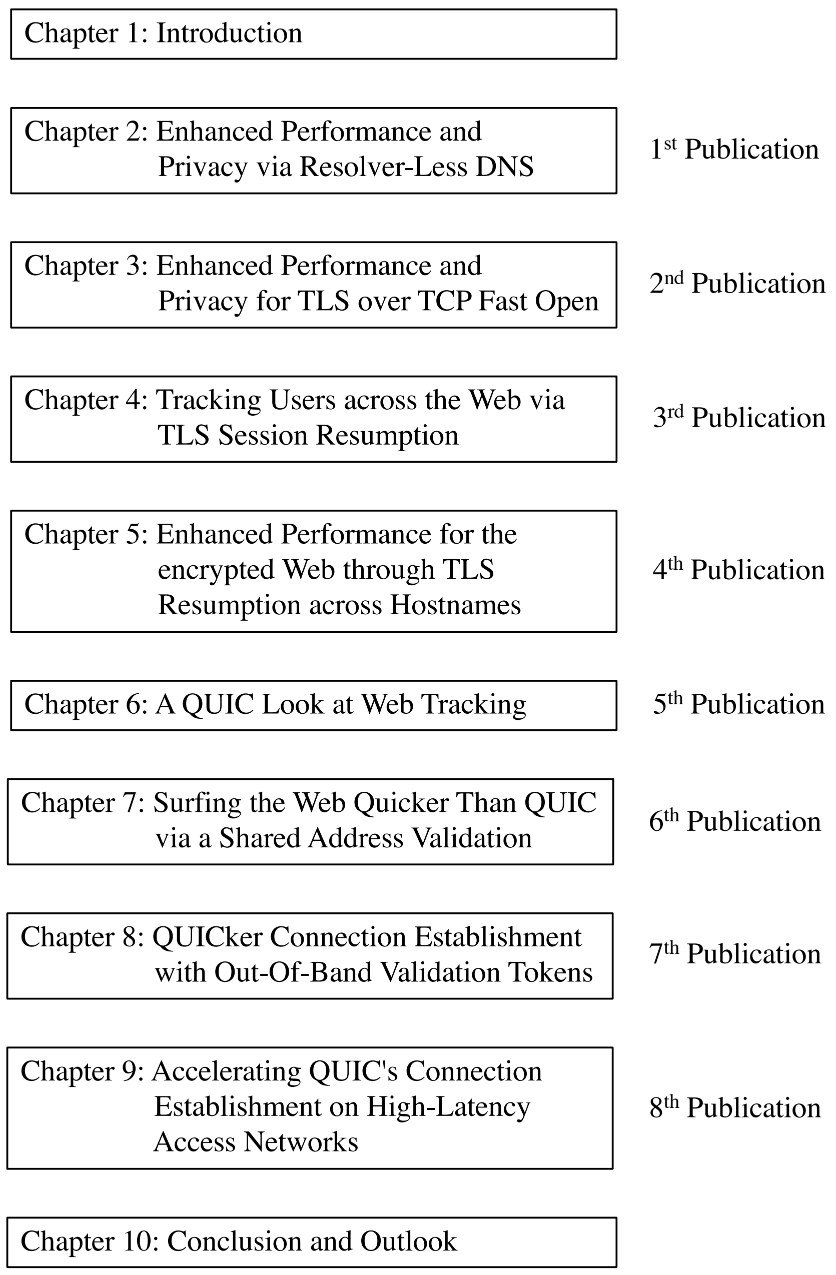

Chapter 1: Introduction
Chapter 2: Enhanced Performance and Privacy via Resolver-Less DNS
1st Publication
Chapter 3: Enhanced Performance and Privacy for TLS over TCP Fast Open
2nd Publication
Chapter 4: Tracking Users across the Web via TLS Session Resumption
3rd Publication
Chapter 5: Enhanced Performance for the encrypted Web through TLS Resumption across Hostnames
4th Publication
Chapter 6: A QUIC Look at Web Tracking
5th Publication
Chapter 7: Surfing the Web Quicker Than QUIC via a Shared Address Validation
6th Publication
Chapter 8: QUICker Connection Establishment with Out-Of-Band Validation Tokens
7th Publication
Chapter 9: Accelerating QUIC's Connection Establishment on High-Latency Access Networks
8th Publication
Chapter 10: Conclusion and Outlook

Figure 1.2: Structure of this thesis

1. Erik Sy. 2019. Enhanced Performance and Privacy via Resolver-Less DNS. *Preprint arXiv:1908.04574* (2019). http://arxiv.org/abs/1908.04574

2. Erik Sy, Moritz Moennich, Tobias Mueller, Hannes Federrath, and Mathias Fischer. 2019. Enhanced Performance for the encrypted Web through TLS Resumption across Hostnames. *Preprint arXiv:1902.02531* (2019). http://arxiv.org/abs/1902.02531

Furthermore, I contributed to the following two research papers that are not included in this thesis:

1. Erik Sy, Tobias Mueller, Matthias Marx, and Dominik Herrmann. 2018. AppPETs: A Framework for Privacy-preserving Apps. In *Proceedings of the 33rd Annual ACM*

Symposium on Applied Computing (SAC '18). ACM, New York, NY, USA, 1179–1182. https://doi.org/10.1145/3167132.3167415

2. Matthias Marx, Erik Sy, Christian Burkert, and Hannes Federrath. 2018. *Anonymity Online – Current Solutions and Challenges*. Springer International Publishing, Cham, 38–55. https://doi.org/10.1007/978-3-319-92925-5_4

2 Enhanced Performance and Privacy via Resolver-Less DNS

Summary of this publication

Citation	Erik Sy. 2019. Enhanced Performance and Privacy via Resolver-Less DNS. *Preprint arXiv:1908.04574* (2019). http://arxiv.org/abs/1908.04574
Status	Unpublished
Type of paper	Research paper
Aim	This paper aims to improve the performance of resolving domain names into IP addresses during web browsing. Furthermore, this work aims to improve the client's privacy posture towards the traditional recursive DNS resolver.
Methodology	This work includes a performance, privacy and security review of the traditional DNS using recursive resolvers. The empirical evaluation includes a measurement of the network latencies between 650 test clients and their recursive resolvers. Furthermore, this work reports measurements on the expiration time of popular DNS records as well as popular configurations for EDNS client subnet.
Contribution	This article presents the first description of an HTTP response header field allowing web servers to push relevant DNS records to their clients. This novel resolver-less DNS approach allows clients to save DNS lookups which can significantly delay connection establishments on the web. To approximate the performance gains enabled by resolver-less DNS, a measurement of the round-trip time between 650 test clients distributed over different access networks in Germany and their pre-configured recursive DNS resolver is provided. The results indicate that five percent of our test clients experience a DNS lookup time of at least 49 ms. The novel resolver-less DNS design allows avoiding the delays caused by traditional DNS lookups contributing to significant performance improvements during web browsing. Furthermore, it is found that the proposal contributes to a reduced number of DNS queries sent towards the recursive DNS resolver. As a result, recursive resolver can observe only a diminished share of the user's browsing activities, which fosters the user's privacy posture towards the resolver. Furthermore, an investigation of the number of additional DNS lookups required by a web server to support the proposal is conducted. The results suggest that the expiration time of DNS records and support for EDNS client subnet can negatively impact the number of required DNS lookups. Furthermore, evaluations indicate that the median investigated

	hostname has an expiration time for its DNS record of five minutes and does not support EDNS client subnet. Thus, for this median hostname the number of required DNS lookups to serve valid DNS records can be as low as one lookup within five minutes. To protect the client against malicious DNS records retrieved via resolver-less DNS, the proposal includes a validation strategy. This approach ensures that the client sends only application data to unauthenticated servers when also the traditional recursive DNS resolver has provided a malicious DNS records. Furthermore, it is found that resolver-less DNS improves the user's resilience against censorship by DNS resolvers.
Co-authors' contribution	This article presents an independent work of the author without contributions by co-authors.

Abstract

The domain name resolution into IP addresses can significantly delay connection establishments on the web. Moreover, the common use of recursive DNS resolvers presents a privacy risk as they can closely monitor the user's browsing activities. In this paper, we present a novel HTTP response header allowing web server to provide their clients with relevant DNS records. Our results indicate, that this resolver-less DNS mechanism allows user agents to save the DNS lookup time for subsequent connection establishments. We find, that this proposal saves at least 80 ms per DNS lookup for the one percent of users having the longest round-trip times towards their recursive resolver. Furthermore, our proposal decreases the number of DNS lookups and thus improves the privacy posture of the user towards the used recursive resolver. Comparing the security guarantees of the traditional DNS to our proposal, we find that resolver-less DNS achieves at least the same security properties. In detail, it even improves the user's resilience against censorship through tampered DNS resolvers.

Keywords: DNS, Performance, HTTP response header

2.1 Introduction

The Internet community aims to improve the performance of connection establishments on the web with means such as the QUIC transport protocol and TLS 1.3 zero Round-Trip Time (0-RTT) handshakes. However, many connection establishments on the web require a prior resolution of the domain name into IP addresses. The Domain Name System (DNS) is responsible for this task. Thus, DNS lookups can become a performance bottleneck on the web. Reducing the time required to resolve domain names into IP addresses benefits the page load time of websites. Following this, it leads to an improved user experience during web browsing [27] and increases the per-user revenue of online service provider [22].

To further improve the performance of resolving domain names into IP addresses, we propose a novel HTTP response header. This new header field *DNS-Record* allows web server to provide their clients the resolved IP addresses for relevant domain names. Subsequently, the client can directly begin with the connection establishment saving the DNS lookup time. Furthermore, this approach improves user privacy as only a smaller fraction of a client's browsing activities can be observed by the used DNS resolver.

In summary, this paper makes the following contributions:

- We propose resolver-less DNS allowing web servers to push DNS records to their clients via HTTP response header.
- We demonstrate the performance improvements gained by our proposal and investigate the resulting DNS overhead for the web server. Our results indicate that our proposal allows clients to save the DNS lookup time. We find, that the DNS lookup accounts for at least 80 ms for the one percent of users having the longest round-trip times towards their DNS resolver.
- We review the privacy and security impact of our proposal. We observe that resolver-less DNS significantly improves the user's privacy posture towards its recursive DNS resolver. Furthermore, we find that our proposed design provides at least the security guarantees of the traditional DNS.

The remainder of this paper is structured as follows: Section 2.2 provides an overview of the traditional DNS and introduces its performance, privacy and security problems that we aim to mitigate. Section 2.3 summarizes the proposed resolver-less DNS design. Evaluation results and a discussion of our proposal are presented in Section 2.4 and 2.5. Related work is reviewed in Section 2.6. Section 2.7 concludes the paper.

2.2 DNS Overview and Problem Statement

In this section, we first describe a popular DNS deployment using recursive DNS resolvers. Subsequently, we review performance, privacy and security problems of this approach to resolve domain names into IP addresses.

2.2.1 DNS Overview

The Domain Name System (DNS) is responsible for resolving domain names into IP addresses and presents an essential component of our Internet infrastructure. Real-world DNS traffic indicates that median users conduct about 1384 DNS lookups per day to 372 different hostnames [8]. Popular operating systems and web browsers have a local DNS cache to reduce the number of required DNS lookups. In total, between 12.9% and 20.4% of the average user's TCP connections directly follow a DNS query to the recursive resolver [12]. Figure 2.1 provides a schematic of a DNS lookup using a recursive resolver. DNS recursive resolvers are usually provided by the Internet Service Provider (ISP) or the client may use a publicly accessible recursive resolver such as Google DNS. The DNS lookup starts with the user's query containing the domain name it aims to resolve, shown as arrow 1 in Figure 2.1. Upon receiving this query, the recursive resolver can either directly resolve the domain name using its DNS cache or it experiences a cache miss. In case of a cache miss, the recursive resolver investigates the authoritative nameserver of the respective domain name. This can be done via an iterative query along the DNS hierarchy involving the DNS root server and the Top Level Domain (TLD) server, as can be seen in Figure 2.1. Finally, the recursive resolver queries the authoritative nameserver to resolve the respective domain name into an IP address. Receiving this response from the authoritative nameserver, the recursive resolver forwards it towards the client as indicated by arrow 8 in Figure 2.1. Note, that a recursive resolver may cache information about TLD servers

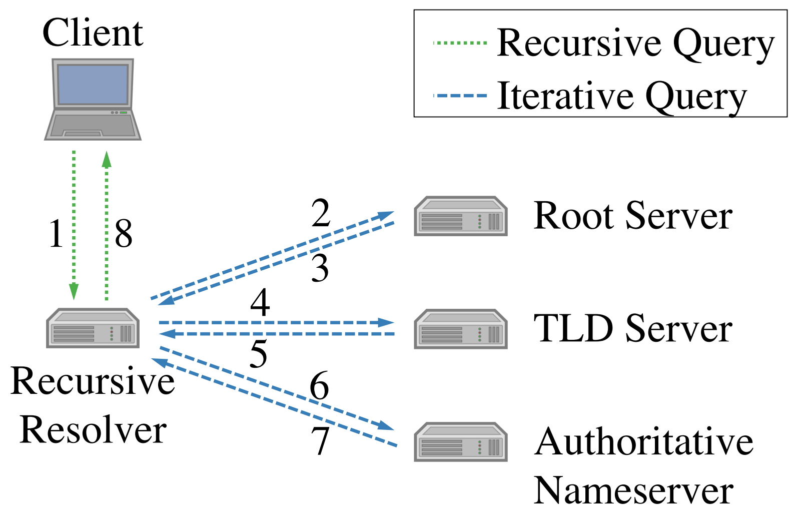


Figure 2.1: Schematic of a DNS lookup using a recursive resolver. The numbers indicate the sequence of the data flows.

and authoritative nameservers allowing it to skip steps of this complete query flow in future queries. For recursive resolvers cache hit rates larger than 80% are reported [12]. Furthermore, studies suggest that recursive resolvers can often perform faster iterative queries compared to most of their users due to their advantageous position in the network topology [25]. Thus, it is more common for clients to resolve domain names into IP addresses using recursive resolvers compared to conducting the iterative queries themselves.

2.2.2 Problem Statement

In this section, we describe performance, privacy and security problems of the traditional DNS that we aim to solve with our proposal.

Performance Problem

A major motivation for clients to conduct DNS lookups is that they aim to send network packets towards the resolved IP address. In these cases, the DNS lookup is an intermediate step delaying the start of the communication between client and server. The DNS lookup time depends amongst others on the round-trip time between client and recursive resolver. This round-trip time varies depending on the access network between a few milliseconds for wired connections and more than a hundred milliseconds for cellular 3G networks [9]. Another contribution to the lookup time presents the time required by the resolver to respond to the request. For DNS cache hits these transactions complete in under 1 ms. However, about 25% of the average DNS transactions require between 10 ms and 1 second to complete [3]. In total, the DNS lookup time can significantly increase the page loading time and therefore reduce the user-perceived quality of web experience [18].

Privacy Problem

There are several privacy risks associated with traditional DNS. Here, we are focusing on the problems related to the recursive resolver. Resolvers are in a privileged position that allows them to reconstruct a significant fraction of the user's browsing session based on the observed queries. Furthermore, patterns within the observed DNS queries allow the recursive DNS resolver to identify the same user across several sessions with high probability [8]. A study of the privacy policies of popular DNS resolvers indicates that many resolvers collect users' DNS traffic and subsequently monetize this data via advertising [2]. Users are often unaware of the privacy risks associated with using a recursive resolver [13]. Thus, an improved DNS would restrict the impact of recursive resolvers on user privacy.

Security Problem

In traditional DNS, the client has difficulties to detect that a recursive resolver modified the requested DNS records or even censored its response. Measurements on DNS manipulations by recursive resolvers indicate that this presents a real-world problem enabling Internet censorship [19]. As a consequence, the censored online resources become unavailable to users. To protect the user's ability to access online resources, an improved DNS should be more resilient against censorship. Note, that a DNS security extension called *DNSSEC* exists aiming to detect manipulations of DNS records. However, *DNSSEC* faces several deployment issues including a significant share of DNSSEC-enabled domains providing an incomplete or incorrect record preventing a validation [4]. Furthermore, the used DNSSEC Key Signing Keys (KSKs) are often weak and the majority of DNSSEC-enabled domains does not rotate these KSKs [4]. Thus, DNSSEC does not represent a feasible solution to the described security problem.

2.3 Resolver-less DNS

In this section, we introduce the resolver-less DNS design. This novel approach allows web servers to provide DNS records to their clients to accelerate their page loading time. First, we summarize our design goals, before we present resolver-less DNS.

2.3.1 Design Goals

We aim to develop an approach to resolve hostnames into IP addresses that supports the following goals:

1. Deployable on today's Internet which excludes approaches requiring changes to middleboxes, kernels of client machines, or the DNS protocol.
2. Reduces the time required for the client to resolve a hostname and thus improves the webpage loading time.
3. Does not require additional Internet infrastructure such as proxies or dedicated DNS resolvers to be deployed.

4. Reduces the number of DNS queries sent to the DNS resolver. Thus, the resolver observes a diminished fraction of the user's browsing activities improving the privacy posture of the user.

5. Supports the performance-optimization EDNS client subnet.

6. Supports clients behind Network Address Translators (NAT).

7. Guarantees confidentiality accordingly to the client's connection with the web server providing the DNS records.

8. Security assurances are not weaker than using a recursive DNS resolver.

9. Provides an improved resilience to the threat of Internet censorship based on DNS.

2.3.2 Design

To describe the design of the resolver-less DNS, we start by introducing a novel HTTP response header field of the type *DNS-Record*. Then, we summarize changes required to the client and server behavior to support the resolver-less DNS proposal.

The *DNS-Record* HTTP Response Header Field

Our design introduces a novel HTTP response header field with the name *DNS-Record*. This header field contains the following key-value pairs:

- *hostname* indicating the record's hostname,
- *A* providing the 32-bit IPv4 address of the hostname,
- *A_TTL* providing the time-to-live in seconds of the A address record,
- *AAAA* providing the 128-bit IPv6 address of the hostname,
- *AAAA_TTL* providing the time-to-live in seconds of the AAAA address record.

An example response header can look like this: *DNS-Record: hostname=a.com; A=1.1.1.1; A_TTL=299*. To provide the client with a list of DNS records within the same HTTP response, the server can append further DNS records using a comma for separation. Note, that HTTP provides a *Link* response header field allowing the server to hint resources to the user agent that should be preloaded [17]. Thus, our proposal extends this existing HTTP functionality to hint resources by providing additionally the corresponding DNS records.

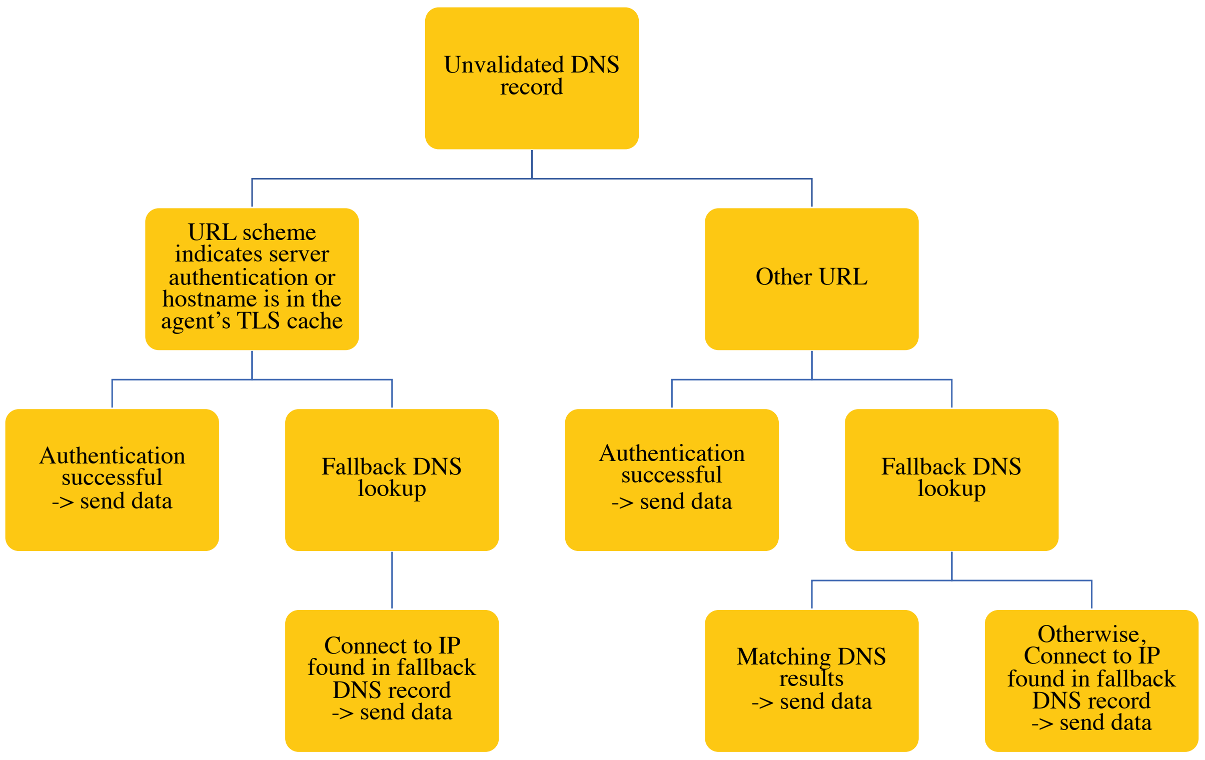

Unvalidated DNS record
URL scheme indicates server authentication or hostname is in the agent's TLS cache
Other URL
Authentication successful -> send data
Fallback DNS lookup
Authentication successful -> send data
Fallback DNS lookup
Connect to IP found in fallback DNS record -> send data
Matching DNS results -> send data
Otherwise, Connect to IP found in fallback DNS record -> send data

Figure 2.2: Overview of the validation strategy for unvalidated DNS records.

Client Behavior

A user agent such as a web browser receiving the *DNS-Record* HTTP response header field saves the content within its DNS cache and marks this DNS record as retrieved via resolver-less DNS. Subsequently, when the user agent intends to connect to this hostname, it retrieves the DNS record from its cache and validates that the record has not yet expired. The client can use these DNS records to directly connect to the hostname without conducting a DNS lookup. However, the client has usually no mechanism available to validate the integrity of these DNS records [4]. To protect against web server serving malicious DNS records, we restrict the usage of resolver-less DNS records as follows.

First, clients are only allowed to send application data over an established connection if they validated the server's identity via a server authentication mechanism or a fallback DNS lookup. Figure 2.2 presents the validation strategy applied by user agents on DNS records retrieved via resolver-less DNS. Following this validation strategy, the user agent is allowed to directly start the connection establishment of protocols such as TCP, QUIC, and TLS based on the IP address in the unvalidated DNS record. However, the client must only send application data over these connections, if it is encrypted in a manner that only the correct server entity is capable to decrypt this data. TLS session resumption mechanisms [23] are an example for these permitted encryption schemes. On the left branch of the validation strategy, the user agent has an indication that the server can authenticate itself by providing an identity proof. For example, the hostname has a record in the user agent's TLS cache or the scheme of the Uniform Resource Locator (URL) indicates server authentication as the scheme *https* does. Subsequently, the client attempts to authenticate the server's identity. If the authentication is successful, this validates the used DNS record and the client can send application data over this connection. Otherwise, the client needs to conduct a fallback DNS lookup to query the correct IP address of the hostname.

Then, the client connects and sends application data to the server at the IP address presented in the fallback DNS record. Note, that the web is moving towards an encrypted ecosystem where today about 80% of a user's HTTP requests use the secure HTTPS variant [10]. Furthermore, the upcoming HTTP version 3 uses always server authentication and transport encryption to exchange application data between client and server [1]. On the right branch, the user agent has no indication that the server is capable to authenticate itself. However, in case the server authenticates it's identity during the connection establishment, then the client can directly send application data. Otherwise, the client is required to conduct a fallback DNS lookup on the corresponding hostname. In case of matching results between the fallback and the resolver-less DNS record, the established connection can be used to transmit application data. Otherwise, the client uses the address within the fallback DNS record to establish a new connection and exchange application data. Note, that a privacy versus performance tradeoff exists concerning the moment when the client starts the fallback DNS lookup. The best performance is achieved, when the client always directly starts to validate the resolver-less DNS record via a fallback DNS lookup. However, the client achieves the best privacy protection if the user agent only conducts necessary fallback DNS lookup based on the validation strategy. In the following, we assume that the user agent conducts the fallback DNS lookup on the left branch of the validation strategy only when it becomes necessary. However, the user agent will conduct the fallback DNS lookup on the right branch of the validation strategy as early as possible.

Second, there is the risk of web servers providing false DNS records leading to a negative performance impact when the user agent visits other websites. As a result, we recommend user agents to restrict the contexts in which they trust the DNS records retrieved via resolver-less DNS. For example, web browsers can restrict the usage of such DNS records within the context of the same browser tab or the same website. In case of a context switch leading to reduced trust in these DNS records, we recommend a similar practice as described above. In detail, the user agent starts the connection establishment towards the IP address found in the resolver-less DNS record and validates this information via a fallback DNS lookup. For matching IP addresses within both DNS records, the user agent is permitted to send application data over the established connection. Otherwise, the client prefers the name resolution of its fallback DNS mechanisms and uses a connection established to this IP address to exchange application data.

It seems reasonable to assume that websites provided by Google yield the same records as a DNS lookup to Google Public DNS. Similar, the DNS records of websites served by Cloudflare can be trusted as much as a lookup to Cloudflare's DNS service. Thus, we recommend to include a whitelist mechanism within user agents that does not apply the described restrictions if the client trusts a websites to provide authentic DNS records. As Google and Cloudflare use a dedicated certificate authority to issue certificates for their hosted websites, this information could be used to associate websites to certain whitelisted organizations.

Server Behavior

Websites are primarily composed of HTML documents. During a website visit, the user agent retrieves these documents along with other content such as images or videos from a web server using HTTP. In the following, we use the term application server to refer to the application building the HTML documents. The HTTPS server is tasked to communicate with the client and aims to serve whenever possible cached content to achieve a scalable system design. Thus, the

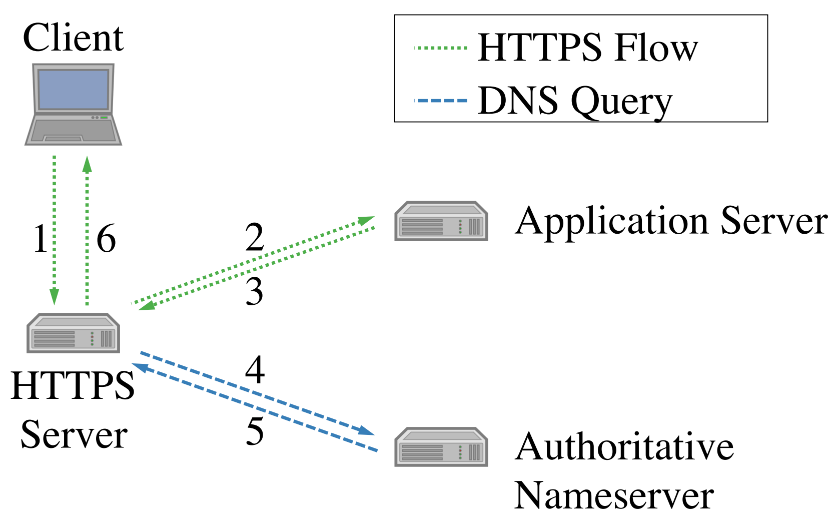


Figure 2.3: Overview on the resolver-less DNS design where the HTTPS server provides the DNS record.

HTTPS server occasionally forwards requests to the application server to fetch updated versions of the cached content or to retrieve dynamic content directly from the application.

In the proposed design of a resolver-less DNS, we assign the task of DNS resolution towards the HTTPS server. For example, if a website is served via a Content Delivery Network (CDN), it would be their task to provide the name resolution. However, we want the application server to indicate the relevant hostnames as this prevents the HTTPS server from parsing HTML documents to learn the relevant hostnames. We assume, that the HTTPS server and the application server communicate via HTTP with each other and the HTTP response header provided by the application will be merged into the server's HTTPS response to the client. Thus, the application server could parse the HTML resource hints [7] such as instructions for DNS prefetching and writes for relevant hostnames *DNS-Records* without resolution results into the HTTP response header. Upon receiving such responses in step three of Figure 2.3, the HTTPS server parses the HTTP response header and recognizes the necessity to include additional information into the *DNS-Record* header field.

Subsequently, the HTTPS server conducts a DNS lookup of the respective hostnames and uses the retrieved DNS records to complete the *DNS-Record* header field of the response. The response can now be forwarded to the client as indicated by step 6 of Figure 2.3. Furthermore, the HTTPS server can now cache the response including the *DNS-Record* header fields. Upon serving the cached response to other clients, the server must dynamically recompute the remaining time-to-live for these *DNS-Record* header fields. Moreover, the HTTPS server should refresh the DNS records in the cached HTTP response before they expire.

Note, that HTTPS server can also decide to remove *DNS-Record* header fields for some relevant hostnames from their response sent to the client. Reasons for this practice include for example, HTTPS server facing temporary resource constraints may want to deactivate resolver-less DNS during these periods. Furthermore, HTTPS server may not want to forward DNS records of relevant hostnames with a very short expiration time such as a few seconds.

Authoritative nameserver may want to provide a different DNS response to users within different locations. This approach allows nameserver to chose the IP address of a server having a rather low network latency towards the client. To support the client in communicating with an advantageous server, the HTTPS server can support EDNS client subnet (ECS) [5]. Here, the HTTPS server queries the nameserver with a truncated IP address of the client and receives a DNS record valid for a scope of IP addresses. The HTTPS server can cache this DNS record and serve all clients matching the defined scope of IP addresses the same DNS record similar as described above. In case the client's source address does not match the scope of the cached DNS record, the HTTPS server is required to conduct another DNS lookup with the client subnet information to retrieve an adequate DNS record.

2.4 Performance Evaluation

In this section, we approximate the performance impact of the proposed resolver-less DNS. To begin with, we present a measurement of the round-trip time between typical clients and their resolvers. This RTT presents a lower boundary for the time required by a client to conduct DNS lookups. Subsequently, we investigate the expiration time of DNS records of popular hostnames and support for EDNS client subnet because these attributes impact the number of required DNS lookups by the web server.

2.4.1 Lower Boundary of the DNS Lookup Time

As the main benefit of our proposal, it allows clients to save the DNS lookup time and therefore reduces the webpage loading time. To approximate this performance enhancement for real-world clients, we investigate the round-trip time between clients and their pre-configured DNS resolver. Note, that this RTT presents only a lower boundary for the DNS lookup time as we neglect the time required by the resolver to process the query and to conduct possibly necessary iterative queries to other servers to provide the client with its response.

Data Collection

To obtain the real-world RTT between a broad range of clients and their pre-configured recursive resolvers, we make use of the RIPE Atlas network [20]. The nodes of this RIPE Atlas network are distributed across several autonomous systems using various access technologies. Using this network, we can task nodes to conduct custom DNS queries or ping measurements. For this measurement, we selected a set of 800 RIPE Atlas nodes located in Germany. Thus, we aim to obtain a realistic representation of typical Internet accesses in countries similar to Germany concerning their infrastructure.

To measure the corresponding RTTs from different selected nodes, we use these nodes to ping their pre-configured recursive DNS resolver. To ensure that resolvers using anycast services always return the same physical server and provide consistent RTTs, we start by investigating the IP address of the used recursive resolvers. For this purpose, we control an authoritative nameserver for a subdomain similar to dnstest.a.com. Subsequently, we send a query for a random subdomain in our DNS zone to the pre-configured recursive resolver such as random.dnstest.a.com. Concurrently, we capture the network traffic of our authoritative nameserver and observe a DNS query for exactly the subdomain random.dnstest.a.com. Thus, we reason that the sender's IP address of the observed DNS query in our network traffic is resolving the client's DNS query and is therefore associated with the node's recursive DNS resolver. In cases, where this observed IP address does not match the locally configured DNS resolver within the node, we assume that these both IP addresses are colocated and yield about the same RTT.

We conducted our data collection on the 13th of June 2019 with five ping measurements between each node and its pre-configured DNS resolvers. We computed the RTT as the average of these five ping measurements and obtained in total results for 650 nodes. The main cause for failures can be attributed to resolvers not responding to our ping measurements. Additionally, we observed also a small number of nodes experiencing failures during their DNS measurements.

Furthermore, we derived the autonomous system numbers of the IP addresses of all nodes and their resolvers. For matching autonomous systems (AS) between the node and its recursive resolver, we assume that the resolver is provided by the node's Internet Service Provider. This approach allows us to compare the performance of resolvers within the same AS versus the performance of resolvers belonging to a different AS as it would be likely for public DNS resolvers such as Goggle DNS. In total, our data collection successfully obtained measurements from 650 nodes in Germany of which 474 nodes had at least one pre-configured DNS resolver within the same AS. 298 of the investigated RIPE Atlas nodes had at least one pre-configured DNS resolver not matching their own AS.

Results

The traditional DNS has a performance overhead compared to our proposal as it requires clients to retrieve DNS records from their resolvers. In this section, we evaluate the round-trip time between our 650 test nodes and their pre-configured DNS resolver. This round-trip time represents the lower boundary of the performance overhead of the traditional DNS. The round-trip time is a good approximation of the time required to conduct the DNS lookup if the resolver has a cached response for the client's query. However, in case of a cache miss, the DNS lookup time can be significantly longer as the resolver conducts additionally iterative queries to the authoritative nameserver of the queried hostname. Thus, a fraction of about 25% of real-world DNS lookups require between 10 ms and 1 second to complete [3].

Figure 2.4 plots a cumulative distribution of the round-trip time between 650 RIPE Atlas nodes located in Germany and their pre-configured DNS resolvers. The solid, blue plot shows the corresponding distribution taking all DNS resolvers into account. The dashed, orange plot focuses on a subset of DNS resolvers that match the ASN of our test node conducting the DNS lookup. Here, we assume that this DNS resolver is provided by the client's ISP. Whereas the dash-dotted, green plot indicates DNS lookups where the client and resolver are located in different autonomous systems as it can be expected for client's using a public DNS resolver. Our results indicate that the investigated round-trip times are significantly faster when the client and the used DNS resolver share the same AS. For example, 59% of the tested nodes experienced less than 10 ms round-trip time when the resolver was located within the same AS. However, only 17% of the clients completed the round-trip within this duration when the peers did not share the same AS. Moreover, we observe that a tail of clients experiences significantly longer round-trip times with their DNS resolver. For the solid, blue plot the 95th percentile and the 99th percentile have a value of 49 ms and 80 ms, respectively.

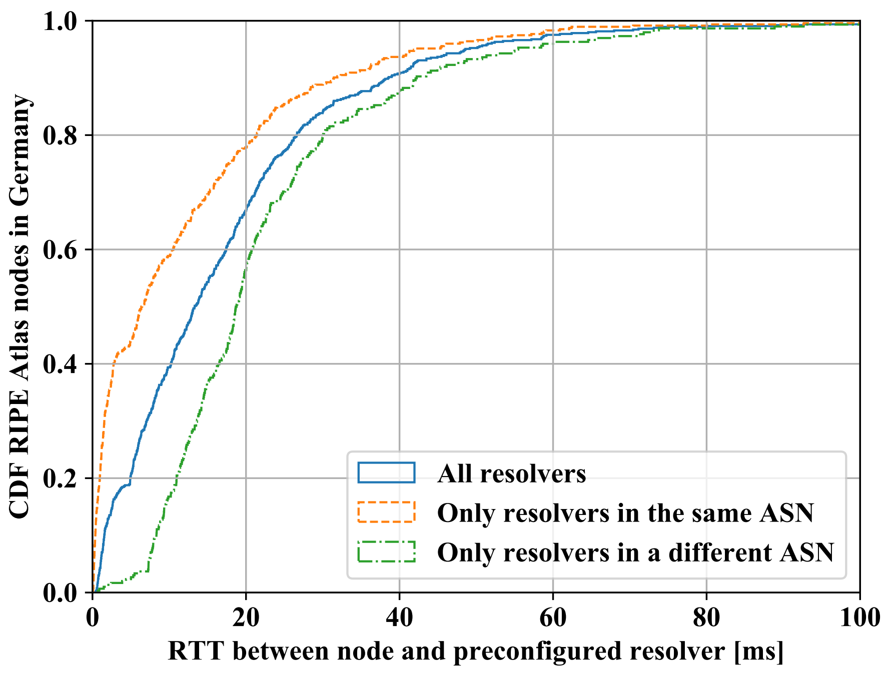


Figure 2.4: Cumulative distribution of the Round-Trip Time (RTT) between the preconfigured DNS resolver and 650 RIPE Atlas nodes located in Germany.

In total, our results indicate that our proposal significantly improves the performance of DNS resolutions even when the DNS resolver can provide the client with a cached response. We find, that the long tail of clients experiencing high network latencies with their DNS resolver benefits the most from the proposed resolver-less DNS.

2.4.2 Expiration Time of DNS Records

The retrieval of an average website requires about 20 connections to different hostnames [24]. Thus, we assume that a web server supporting our proposal requires a fresh cache of about these 20 hostnames per website. To approximate the number of required DNS lookups by the web server, we investigate in the following the expiration time of DNS records for popular websites. We start by describing our methodology used to collect these DNS records, before we summarize our results.

Data Collection

For this measurement, we retrieve fresh DNS records directly from the authoritative nameserver of a given hostname. In detail, we identified the authoritative nameserver of the Alexa Top Ten Thousand websites [11] using the DNS lookup tool *dig*. Then, we used the provided nameserver to retrieve a fresh DNS record of the type A and AAAA for the corresponding hostname. Subsequently, we retrieved the expiration time presented in the DNS record. We conducted this measurement on the 5th of August 2019. In total, we successfully retrieved the expiration time for 2401 AAAA and 9362 A DNS records. The failures are mainly caused by nameservers not supporting the requested DNS record type and connection failures such as timeouts.

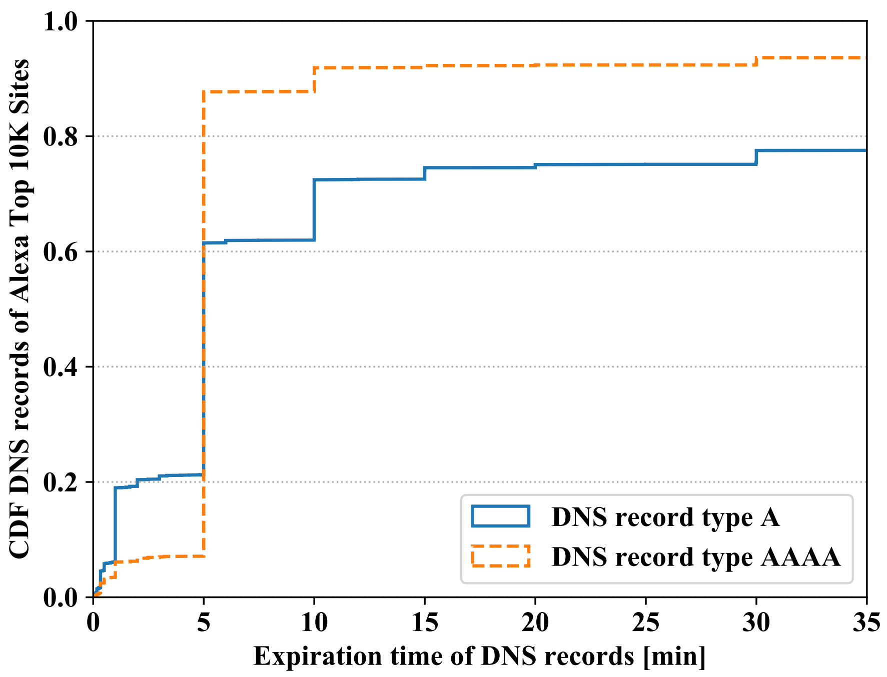


Figure 2.5: Cumulative distribution of the collected DNS records of the Alexa Top 10K Sites over their expiration time.

Results

Figure 2.5 plots a cumulative distribution of the retrieved DNS records of the Alexa Top Ten Thousands Sites over their expiration time. The solid, blue and the dashed, orange plot represent the expiration times of A and AAAA DNS record types, respectively. Our results indicate that an expiration time of five minutes is very popular within the investigated DNS records. We find that 80.6% of the AAAA and 40.2% of the A DNS records use exactly this configuration. Furthermore, A DNS records often use one and ten minutes expiration time with a share of 12.8% and 10.5%, respectively. We observe, that only a tail of 12.5% of the A records and 6.4% of the AAAA records advertise expiration times longer than 30 minutes. Moreover, our results indicate that 6.2% of the A records and 3.5% of the AAAA records announce an expiration time shorter than one minute.

In total, we find that resolver-less DNS is feasible with regard to real-world DNS record expirations times because they lead only to a small number of additional DNS lookups per minute per website.

2.4.3 ECS Support by Authoritative Nameserver

EDNS client subnet intends to speed up the data delivery from CDNs by resolving DNS queries relative to the client's location as indicated by an IP address prefix [5]. However, ECS is a controversial performance optimization, especially with respect to its privacy properties [14]. Furthermore, there exist reasonable alternatives such as anycast services [16] and the feature of connection migration within the QUIC transport protocol [26]. Nonetheless, we are investigating in this section the interaction between our proposal and real-word ECS. We begin by describing the applied methodology of this measurement. Subsequently, we summarize our results.

Data Collection

Google Public DNS is well-known to support ECS. Thus, we retrieved the DNS records for the Alexa Top Ten Thousand Sites from Google Public DNS using the DNS lookup tool *dig*. *Dig* allows us to explicitly signal support for ECS by specifying a client subnet. During our measurements, we specified a client subnet matching the IP address of our test server located within a data center of our hosting provider Hetzner Online GmbH. We conducted this measurement on the 5th of August 2019 and successfully retrieved 9999 DNS records that included an ECS response. The failed retrieval can be attributed to a DNS failure of the authoritative nameserver.

Results

The analysis of the retrieved DNS records yields, that 88.8% of these records announce a global scope. Thus, only the remaining 11.2% of the investigated hostnames provide DNS resolutions based on the presented client subnet. The size of the scopes provided in these 11.2% of the DNS records varied for our announced client subnet with the IPv4 address 159.69.184.0. 10.0% of the responses covered a scope of /23 and /24 containing 512 and 256 IP addresses, respectively. However, we assume that the results of this measurement vary depending on the announced client subnet. Furthermore, we expect these scopes to be larger if the presented client's source address is part of a larger IP address block of the same AS as this may hint a physical proximity of these addresses. Concerning our proposal, we prefer large scopes of IP addresses for which a DNS record is valid because this reduces the number of required DNS lookups by the web server.

In total, we find that almost 90% of the DNS records are valid on a global scope which supports the feasibility of our proposal. For the remaining hostnames ECS can make additional DNS lookups necessary. However, this depends on the ECS configuration of the authoritative nameserver and the clients' source addresses.

2.5 Discussion

In this section, we compare the privacy and security properties of our proposal to the traditional DNS. We start by presenting our security considerations. Subsequently, we describe the privacy impact of the proposed resolver-less DNS.

2.5.1 Security Considerations

Translating the security threat of the resolver-less DNS to the traditional DNS, we have a scenario where the web server colludes with the client's recursive resolver. Therefore, this scenario is similar to a client visiting google.com and using Google Public DNS as the recursive resolver. In the following, we investigate the security risks of such a design by differentiating four cases:

1. The server presents a non-registered hostname and a fake DNS record.
2. The server presents a non-registered hostname and a true DNS record.

3. The server presents a registered hostname and a fake DNS record.
4. The server presents a registered hostname and the true DNS record.

The process of domain name registration requires the registering entity to identify itself and therefore mitigates abuse. In case 1 and 2, the adversary links to a non-registered hostname. In case 1 the fake DNS record leads the client to connect to that non-registered hostname, while the true DNS record in case 2 prevents this connection establishment because there does not exist a DNS record for non-registered hostnames. Thus, adversaries can exploit case 1 to launch for example phishing attacks with non-registered, look-alike hostnames aiming to steal user data/ passwords. In case 3, the adversary can redirect traffic of a registered hostname to a malicious server using a fake DNS record. However, there exist some mitigations such as the Strict-Transport-Security HTTP response header that can be used to prevent user agents to connect to these hostnames using an insecure connection. Case 4 describes the designated use case where the client connects to the correct IP address of the registered hostname.

In total, only case 1 and 3 present security issues and they can always be mitigated when the established connections deploy server authentication as it is the default for TLS connections. If the client attempts to establish insecure connections to a hostname using a DNS record retrieved via resolver-less DNS, it must make a second DNS lookup on this hostname using a fallback DNS mechanism. Subsequently, the client must only send application data using this insecure connection if both DNS mechanisms yielded the same result. Otherwise, the name resolution of the fallback DNS mechanism must be used to establish the insecure connection. This approach prevents attacks derived from case 1 or 3, if the fallback DNS mechanism still provides trustworthy DNS records.

The resilience against censorship via tampered DNS resolver presents another important security aspect. Here, resolver-less DNS enables user agents to retrieve DNS records for domain names which cannot be correctly resolved by the client's tampered DNS resolver. Thus, our proposal makes censorship on the web more difficult when these connections are established using server authentication.

Based on the above security considerations, we find that the security guarantees of our proposed resolver-less DNS are adequate or even better than the guarantees of the traditional DNS.

2.5.2 Privacy Considerations

The deployment of resolver-less DNS contributes to a reduced number of DNS queries a client has to send towards the used fallback DNS resolver. This benefits the privacy posture of the user as the DNS resolver obtains only a diminished view on the user's browsing activities. However, our proposal does not replace the traditional DNS and occasionally requires fallback DNS lookups. For example, DNS queries to recursive resolvers are necessary to bootstrap the first connection which then can provide relevant DNS records using our proposal. Furthermore, the proposed resolver-less DNS requires user agents to verify DNS records used for insecure connections via the traditional DNS.

To address these remaining privacy leaks, we proposed a whitelisting mechanism for DNS records from trusted websites possibly those hosted by Google or Cloudflare. As a result, these whitelisted DNS records are trusted to same extend as DNS records originating from the fallback DNS resolver. Thus, the user agent is not required to conduct fallback DNS lookups for the purpose of validating the DNS records received via resolver-less DNS. In total, we find a widespread deployment of resolver-less DNS would significantly improve the user's privacy posture with respect to its used recursive DNS resolver.

2.6 Related Work

In this section, we compare our proposal to related work sharing the goal of a faster and/or a more privacy-friendly DNS.

Oblivous DNS [21] is designed to prevent the recursive resolver from learning the user's IP address. For this purpose, a second DNS resolver is used as a proxy that learns only the client's IP address but not the content of the DNS query. This design provides significant privacy protections compared to the status quo. As a drawback, Oblivous DNS increases the DNS lookup time due to the additional latency and computational overhead. Furthermore, it does not support the performance-optimization EDNS client subnet. In total, resolver-less DNS provides better performance properties than Oblivious DNS but is not always supported by web server. Thus, Oblivious DNS may be a privacy-friendly alternative to the traditional DNS whenever our proposal requires to fetch DNS records via a fallback mechanism.

The DNS Anonymity Service combines a broadcast mechanism for popular DNS records with an anonymity network to conduct additional DNS lookups [6]. Unlike our proposal, the DNS Anonymity Service causes additional network traffic for downloading the broadcasted DNS records and suffers additional network latency when the client resolves hostnames via the anonymity network. In total, the performance gains of this clean-slate approach are vague as they depend on the user's browsing behavior. Furthermore, this approach does not integrate well into the existing DNS and requires additional Internet infrastructure to be deployed.

DNS prefetching describes a popular performance optimization where browsers start resolving the hostname of hyperlinks before the user clicks on them. However, privacy research on this mechanism indicates severe privacy problems. For example, it was shown that the recursive resolver could even infer the search terms the user entered into the search engine based on DNS prefetching [15].

2.7 Conclusions

As our objective, we hope to raise awareness for the performance and privacy limitations of traditional DNS. To address these real-world problems, we presented resolver-less DNS. Our evaluation substantiates the feasibility of this proposal and its significant performance and privacy benefits. Furthermore, the proposed design is rather simple and operators of web server are incentivized to support resolver-less DNS as it improves the page loading time of their websites.

2.8 References

[1] Mike Bishop. 2019. *Hypertext Transfer Protocol Version 3 (HTTP/3)*. Internet-Draft draft-ietf-quic-http-22. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-22 Work in Progress.

[2] Samantha Bradshaw and Laura DeNardis. 2019. Privacy by Infrastructure: The Unresolved Case of the Domain Name System. *Policy & Internet* 11, 1 (2019), 16–36. https://doi.org/10.1002/poi3.195 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/poi3.195

[3] Thomas Callahan, Mark Allman, and Michael Rabinovich. 2013. On modern DNS behavior and properties. *ACM SIGCOMM Computer Communication Review* 43, 3 (2013), 7–15.

[4] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. 2017. A Longitudinal, End-to-End View of the DNSSEC Ecosystem. In *26th USENIX Security Symposium (USENIX Security 17)*. 1307–1322.

[5] Carlo Contavalli, Wilmer van der Gaast, David C Lawrence, and Warren "Ace" Kumari. 2016. Client Subnet in DNS Queries. RFC 7871. https://doi.org/10.17487/RFC7871

[6] Hannes Federrath, Karl-Peter Fuchs, Dominik Herrmann, and Christopher Piosecny. 2011. Privacy-preserving DNS: analysis of broadcast, range queries and mix-based protection methods. In *European Symposium on Research in Computer Security*. Springer, 665–683.

[7] Ilya Grigorik. 2019. *Resource Hints*. W3C Working Draft. W3C. https://www.w3.org/TR/2019/WD-resource-hints-20190307/.

[8] Dominik Herrmann, Christian Banse, and Hannes Federrath. 2013. Behavior-based tracking: Exploiting characteristic patterns in DNS traffic. *Computers & Security* 39 (2013), 17–33.

[9] Austin Hounsel, Kevin Borgolte, Paul Schmitt, Jordan Holland, and Nick Feamster. 2019. Analyzing the costs (and benefits) of DNS, DoT, and DoH for the modern web. *arXiv preprint arXiv:1907.08089* (2019).

[10] HTTP Archive. 2019. Report: State of the Web. Retrieved July 31, 2019 from https://www.httparchive.org/reports/state-of-the-web

[11] Alexa Internet Inc. 2019. Alexa Top 1,000,000 Sites. Retrieved August 6, 2019 from http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

[12] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. 2002. DNS performance and the effectiveness of caching. *IEEE/ACM Transactions on networking* 10, 5 (2002), 589–603.

[13] Ruogu Kang, Laura Dabbish, Nathaniel Fruchter, and Sara Kiesler. 2015. “My Data Just Goes Everywhere:” User Mental Models of the Internet and Implications for Privacy and Security. In *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*. 39–52.

[14] Panagiotis Kintis, Yacin Nadji, David Dagon, Michael Farrell, and Manos Antonakakis. 2016. Understanding the Privacy Implications of ECS. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Juan Caballero, Urko Zurutuza, and Ricardo J. Rodríguez (Eds.). Springer International Publishing, Cham, 343–353.

[15] Srinivas Krishnan and Fabian Monrose. 2010. DNS Prefetching and Its Privacy Implications: When Good Things Go Bad. In *Proceedings of the 3rd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More (LEET'10)*. USENIX Association, Berkeley, CA, USA, 10–10. http://dl.acm.org/citation.cfm?id=1855686.1855696

[16] Kurt Erik Lindqvist and Joe Abley. 2006. Operation of Anycast Services. RFC 4786. https://doi.org/10.17487/RFC4786

[17] Mark Nottingham. 2017. Web Linking. RFC 8288. https://doi.org/10.17487/RFC8288

[18] Jianping Pan, Y Thomas Hou, and Bo Li. 2003. An overview of DNS-based server selections in content distribution networks. *Computer Networks* 43, 6 (2003), 695–711.

[19] Paul Pearce, Ben Jones, Frank Li, Roya Ensafi, Nick Feamster, Nick Weaver, and Vern Paxson. 2017. Global Measurement of DNS Manipulation. In *26th USENIX Security Symposium (USENIX Security 17)*. 307–323.

[20] Reseaux IP Europeens Network Coordination Centre. 2019. RIPE Atlas – Internet measurement network. Retrieved August 2, 2019 from https://atlas.ripe.net/

[21] Paul Schmitt, Anne Edmundson, Allison Mankin, and Nick Feamster. 2019. Oblivious DNS: practical privacy for DNS queries. *Proceedings on Privacy Enhancing Technologies* 2019, 2 (2019), 228–244.

[22] Steve Souders. 2009. Velocity and the Bottom Line. Retrieved May 24, 2019 from http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html/

[23] Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2018. Tracking Users Across the Web via TLS Session Resumption. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC '18)*. ACM, New York, NY, USA, 289–299. https://doi.org/10.1145/3274694.3274708

[24] Erik Sy, Moritz Moennich, Tobias Mueller, Hannes Federrath, and Mathias Fischer. 2019. Enhanced Performance for the encrypted Web through TLS Resumption across Hostnames. *CoRR* abs/1902.02531 (2019). arXiv:1902.02531 http://arxiv.org/abs/1902.02531

[25] Erik Sy, Tobias Mueller, Moritz Moennich, and Hannes Federrath. 2019. Accelerating QUIC's Connection Establishment on High-Latency Access Networks. *CoRR* abs/1907.01291 (2019). arXiv:1907.01291 http://arxiv.org/abs/1907.01291

[26] Erik Sy, Tobias Mueller, Moritz Moennich, and Hannes Federrath. 2019. Accelerating QUIC's Connection Establishment on High-Latency Access Networks. *arXiv e-prints* (Jul 2019). arXiv:1907.01291

[27] Matteo Varvello, Jeremy Blackburn, David Naylor, and Konstantina Papagiannaki. 2016. EYEORG: A Platform For Crowdsourcing Web Quality Of Experience Measurements *(CoNEXT '16)*. ACM, New York, NY, USA, 399–412. https://doi.org/10.1145/2999572.2999590

3 Enhanced Performance and Privacy for TLS over TCP Fast Open

Summary of this publication

Citation	Erik Sy, Tobias Mueller, Christian Burkert, Hannes Federrath, and Mathias Fischer. Enhanced Performance and Privacy for TLS over TCP Fast Open. *Proceedings on Privacy Enhancing Technologies 2020*, 2 (2020), Accepted for publication.
Ranking	CORE: B GGS: B Microsoft Academics: [B|A++]
Status	Accepted for publication.
Type of paper	Research paper
Aim	This paper investigates privacy defects and performance limitations of TCP Fast Open and introduces the TCP Fast Open Privacy protocols to address these issues.
Methodology	The used methods include a performance and privacy review of TCP Fast Open and an empirical study based on browser and web server measurements. Furthermore, a prototype of the TCP FOP protocol has been implemented for this work and subsequently evaluated based on performance measurements and simulations.
Contribution	This article provides the first description of tracking via TCP Fast Open cookies. These cookies enable passive network attackers and online services to link website visits to the same user. Furthermore, the results indicate considerable performance limitations of TCP Fast Open caused by websites served concurrently from numerous different IP addresses. This real-world load balancing approach leads to an average failure rate of abbreviated TCP Fast Open handshakes of about 40% upon the first revisit of a website. Measurements with popular web browsers such as Chrome and Firefox suggest, that tracking periods via these cookies are unrestricted. Moreover, these cookies enable tracking across private browsing modes, browser restarts and even across different browsers. Online trackers can exploit tracking via TCP Fast Open cookies to track users as a third-party across different websites. However, user tracking is also feasible across several websites, if they are served from the same IP address. To counter these significant privacy issues, the paper introduces the TCP Fast Open Privacy protocol. This cross-layer protocol uses an encrypted TLS channel to send the single-use cookies from the server to the client. This protects against user tracking by network-based observers. Moreover, countermeasures against user tracking by online services are proposed, such as an expiration time for TCP Fast Open cookies. Furthermore, the performance

	of TCP Fast Open for website revisits is improved by associating the cookies with the hostname known from the TLS protocol instead of the servers IP address. Based on an implemented prototype and a simulation, the performance of TCP FOP is evaluated. For example, TCP FOP reduces the overhead delay of the first website revisits by about 83.6 ms for an average LTE mobile connection.
Co-authors' contribution	The article was co-authored by Tobias Mueller, Christian Burkert, Prof. Dr. Hannes Federrath and Prof. Dr. Mathias Fischer. Tobias Mueller implemented the TCP FOP protocol, conducted the performance measurements with this prototype and supported me to describe these parts of the paper. Tobias Mueller, Christian Burkert, and Prof. Dr. Mathias Fischer assisted to revise and refine this paper. Prof. Dr. Hannes Federrath provided general feedback.

Abstract

Small TCP flows make up the majority of web flows. For them, the TCP three-way handshake induces significant delay overhead. The TCP Fast Open (TFO) protocol can significantly decrease this delay via zero round-trip time (0-RTT) handshakes for all TCP handshakes that follow a full initial handshake to the same host. However, this comes at the cost of privacy limitations and also has some performance limitations. In this paper, we investigate the TFP deployment on popular websites and browsers. We found that a client revisiting a web site for the first time fails to use an abbreviated TFO handshake in 40% of all cases due to web server load-balancing using multiple IP addresses. Our analysis further reveals significant privacy problems of the protocol design and implementation. Network-based attackers and online trackers can exploit TFO to track the online activities of users. As a countermeasure, we introduce a novel protocol called TCP Fast Open Privacy (FOP). TCP FOP prevents tracking by network attackers and impedes third-party tracking, while still allowing 0-RTT handshakes as in TFO. As a proof-of-concept, we have implemented the proposed protocol for the Linux kernel and a TLS library. Our measurements indicate that TCP FOP outperforms TLS over TFO when websites are served from multiple IP addresses.

Keywords: TCP Fast Open, Online Tracking, Protocol Design

3.1 Introduction

TCP is the standard network protocol for transmitting information on the Internet and a TCP connection is usually used to establish a subsequent TLS connection. Nowadays about 80% of HTTP web requests are encrypted [10] Retrieving a popular web page requires HTTPS connections to on average 20 different hosts [31], which sums up to 20 TCP and 20 TLS handshakes. Especially for short web flows, these handshakes represents a significant overhead.

To decrease the delay of a TCP handshake, the TCP Fast Open Protocol (TFO) [5] has been deployed by the most popular operating systems and browsers, even though it is not yet actively used by all of them. It shortens TCP's three-way handshake by one round-trip time for all connections that follow an initial TFO connection to the same host. With the initial TFO handshake, the server verifies the client IP address and sends an identifier (Fast Open cookie) to the client as proof of this verification. The client can then use this cookie for all successive TFO connections to the server as long as its IP address has not changed. However, while providing significant speedup, this identifier can be used to link TFO sessions. Thus, an online tracker can exploit it to collect profiles of users' browsing behavior. Furthermore, as TFO messages are sent unencrypted, users can be tracked via passive network monitoring like dragnet surveillance. Tracking via the TFO protocol is limited as it requires a matching client and server IP address as well as a matching server port for reusing cached Fast Open cookies.

Despite this limitation, tracking via Fast Open cookies can be more effective than tracking based on IP addresses. For example, TFO tracking enables to differentiate between devices sharing the same public visible IP address, e.g., as a result of Network Adress Translation (NAT). Furthermore, TFO tracking allows the attacker to extend the tracking periods compared to IP addresses tracking in cases where the public IP address are dynamically assigned. Worse, a TFO tracking period is not terminated by a browser restart, nor is it restricted to the scope of a single application on that host, but only by a restart of the kernel or the host (as this clears the kernel's TCP cache). This is especially worrisome on mobile devices, which are *always on* and are seldomly restarted. Moreover, TFO tracking is independent of conventional tracking practices such as HTTP cookies or browser fingerprinting [6] that use other protocols than IP and TCP.

While the most effective countermeasure is to disable TFO entirely, this prevents the round-trip time savings during connection establishment. To balance the legitimate needs of online privacy and faster TLS over TCP connections, we additionally propose the TCP Fast Open Privacy (TCP FOP) protocol as a countermeasure to TFO tracking.

In summary, this paper makes the following contributions:

- To the best of our knowledge, we are the first to describe tracking via TFO cookies. Passive network attackers and online services can use these cookies to link website visits to the same user. We find that the TFO protocol provides no measures to restrict such a tracking mechanism.
- We found that under real-world conditions, the first revisit of a website supporting the TFO protocol fails in 40% of all cases to perform an abbreviated handshake. The main reason for this is server load balancing, i.e., the same website is concurrently served from multiple different IP addresses. This represents a considerable performance limitation of TFO.
- We investigate the TFO configuration of popular browsers and found that the tracking periods for Chrome, Firefox, and Opera seem to be not restricted at all. We successfully tracked successive connections from these browsers for a period of ten days. Furthermore, tracking is feasible for the tested setups across private browsing modes, browser restarts, and even across different browsers running on the same host. Online trackers can utilize TFO to track users as a third-party across multiple websites, and even across different websites, as long as they are served from the same IP address.
- We propose TCP FOP as a cross-layer solution to overcome the described privacy limitations of TLS over TFO. Furthermore, TCP FOP allows abbreviated handshakes for website revisits independently of the server's IP address. This significantly improves the performance of website revisits compared to TLS over the TFO protocol. For that,

our solution uses an encrypted TLS channel to send Fast Open cookies from the server to the client. We implemented TCP FOP into the Linux kernel and in a TLS library to demonstrate its real-world applicability. The evaluation of our prototype indicates no additional delay compared to TFO/TLS connection establishments.

Note that we responsibly disclosed our privacy concerns regarding the TFO protocol to the vendors of popular browsers. As a result of this disclosure, Mozilla deprecated the TFO protocol on all branches of Firefox for all platforms [15]. Furthermore, Microsoft removed support for TFO from the private browsing mode of the Microsoft Edge browser [3].

The remainder of this paper is structured as follows: Section 3.2 describes the connection establishment of the TCP Fast Open protocol, its deployment within the Alexa Top Sites, and evaluates its real-world performance limitations. Section 3.3 reviews tracking via TFO cookies, privacy threats arising from host-based as well as network-based attackers, and investigates the feasibility of the presented tracking mechanism for popular browsers. Section 3.4 summarizes TCP FOP as well as its implementation and presents evaluation results. Related work is reviewed in Section 3.5. Section 3.6 concludes the paper.

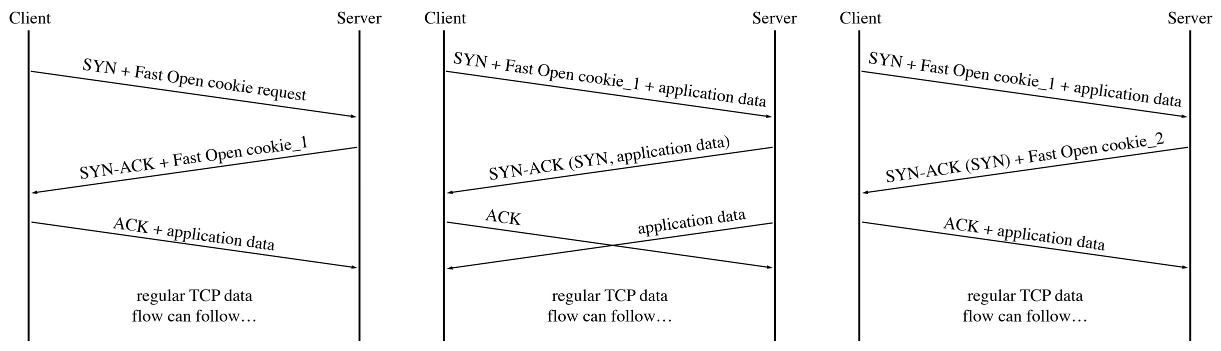
3.2 TCP Fast Open

In this section, we briefly describe the protocol handshake of TCP Fast Open (TFO). Subsequently, we investigate the deployment of the TFO protocol for the Alexa Top Million Sites. We also analyze the performance impact of real-world load-balancing on the rate of zero round-trip time (RTT) connection establishments.

3.2.1 Background on TFO's Connection Establishment

TFO is defined in RFC 7413 [5] as an experimental TCP mechanism. It allows saving up to one round-trip time compared to the standard TCP handshake [23]. For that, during a successful TFO handshake the client obtains a cookie from the server, which it can use in subsequent connections. The TFO cookie is encrypted and authenticated by the server and opaque to the client. RFC 7413 does not specify a general construction scheme for these cookies. However, it has to contain information about the client's publicly visible IP address. Thus, if the client presents a cookie which matches its publicly visible IP address, the server accepts this as proof that the client can receive messages at the claimed IP address. The server then does not need to validate the client's source address via additional message exchanges anymore. This allows the client to establish a connection without waiting for the server's response. Thus, application data can be sent immediately along with the first client message. Figure 3.1 shows a schematic of the TFO handshakes.

Initial handshake: At the beginning, the client has no information about the server. Similar to a TCP three-way handshake [23], the client initiates a connection by sending a SYN to the server as shown in Figure 3.1a. This SYN includes a TCP option that requests a TFO cookie from the server. The server confirms the connection request with a message containing a SYN-ACK and a TFO cookie. The client then caches the TFO cookie for the establishment of subsequent connections. To complete the three-way handshake, the client sends an ACK. The now established connection is a standard TCP connection.


Client
Server
SYN + Fast Open cookie request
SYN-ACK + Fast Open cookie_1
ACK + application data
regular TCP data flow can follow…
Client
Server
SYN + Fast Open cookie_1 + application data
SYN-ACK (SYN, application data)
ACK
application data
regular TCP data flow can follow…
Client
Server
SYN + Fast Open cookie_1 + application data
SYN-ACK (SYN) + Fast Open cookie_2
ACK + application data
regular TCP data flow can follow…

a) Initial Handshake (HS) b) 0-RTT Handshake c) Rejected 0-RTT HS

Figure 3.1: Handshakes in TCP Fast Open protocol.

0-RTT handshake: For subsequent connections to the same server, the client utilizes the previously retrieved TFO cookie. For that, it sends the cookie as part of the SYN message to the server as shown in Figure 3.1b and c. Additionally, the client can include application data as payload within the SYN message. Upon receiving the client's connection request, the server validates the included TFO cookie.

A cookie is valid for a connection request, if the claimed IP address of the client matches the one associated with the cookie. For valid cookies, the server accepts the connection request with the attached application data. As a response, the server sends a SYN-ACK, which acknowledges the client's SYN message and the length of the received application data (see Figure 3.1b). This SYN-ACK message can contain application data as a payload. In total, this abbreviated connection establishment saves one round-trip time of delay compared to TCP's three-way handshake.

In the case of an invalid cookie, the server drops the application data of the client as shown in Figure 3.1c. For that, the server sends a SYN-ACK which only acknowledges the client's SYN but not the application data. Moreover, the server generates a new TFO cookie for the client and attaches this as a payload to the SYN-ACK. Thereafter, the client replaces the cached *cookie_1* with the fresh *cookie_2*, which can be used in subsequent 0-RTT handshakes (see Figure 3.1c). Note, that a rejected 0-RTT handshake only causes the same delay as a standard TCP three-way handshake.

3.2.2 Evaluation

In this section, we first investigate the deployment of TFO within the Alexa Top Million Sites. This allows us to determine an upper limit of websites which possibly deploy TFO to track their visitors. Based upon a sample size of approx. 30 000 hostnames within the Alexa Top Million Sites, we then investigate to which extent changing server IP addresses affects the performance gains achievable by TFO.

Deployment of TFO

Major operating systems such as Windows, macOS, Linux, FreeBSD, Android, and iOS support the TFO protocol, which is a precondition for its widespread adoption. However, these implementations do not set TFO as a default for all TCP connections and thus it still requires modifications to the client- and server-side applications to be used. The RFC describing the TFO protocol was published in 2014 [5]. We thus assume that our measurement of the TFO deployment investigates an early-stage in the wide-spread adoption of this protocol that we expect in the near future.

To approximate the deployment of TCP Fast Open on the Internet, we investigate the support for TFO within the Alexa Top Million Sites [1]. For this purpose, we sent a SYN packet containing a TFO cookie request to the first IP address mentioned in the DNS record of the corresponding hostname. In this measurement, we did not treat hostnames hosted by Content Delivery Networks (CDNs) differently. Our measurements succeeded to receive a response from a server for 97.1% of the investigated hostnames. The remaining 2.9% of our measurements resulted in errors. In detail, we observed errors in the name resolution for 19 498 domain names. Furthermore, we did not receive a response to our TFO connection request from 9254 hosts. If the respective host responded with a SYN-ACK including a TFO cookie, we consider this site to support the TFO protocol and the contrary otherwise. We limited our scans to port 443 on the targeted web server as this is the standard port for HTTPS web services [24]. We conducted this measurement from an IPv4 address on the 10th of August 2018 using a dedicated Python script.

Alexa Top lists	Share of hostnames with TFO-support
Alexa Top 10	60.0%
Alexa Top 100	28.0%
Alexa Top 1K	12.4%
Alexa Top 10K	5.9%
Alexa Top 100K	3.4%
Alexa Top 1M	3.2%

Table 3.1: Websites with TFO-support in Alexa Top lists

Table 3.1 shows the number of hostnames supporting the TCP Fast Open protocol within different Alexa Top lists. We find that 60% of the ten most popular hostnames support the protocol. However, this fraction decreases with the size of the Alexa Top list. While 28% among the Top Hundred hostnames still enable TCP Fast Open handshakes, this share decreases to 3.2% within the Top Million sites. We assume that higher-ranked websites tend to adopt new protocols such as TCP Fast Open earlier than other websites.

Performance Limitations of TFO

Repeated connections to a hostname are not necessarily served from the same IP address due to server load balancing. However, the TFO protocol instructs to utilize a cached Fast Open cookie only if the source IP address, destination IP address, and the destination port match those of the TCP connection in which the cookie was issued. As a result, any time the hostname is resolved to a different IP address, the client experiences a cache miss even if a Fast Open cookie from that hostname is stored in the TCP cache. To assess the performance impact of this design, we observe the IP addresses of the responding servers, while connecting to a hostname several times. We conducted this measurement on August 24th, 2018 using a virtual machine in

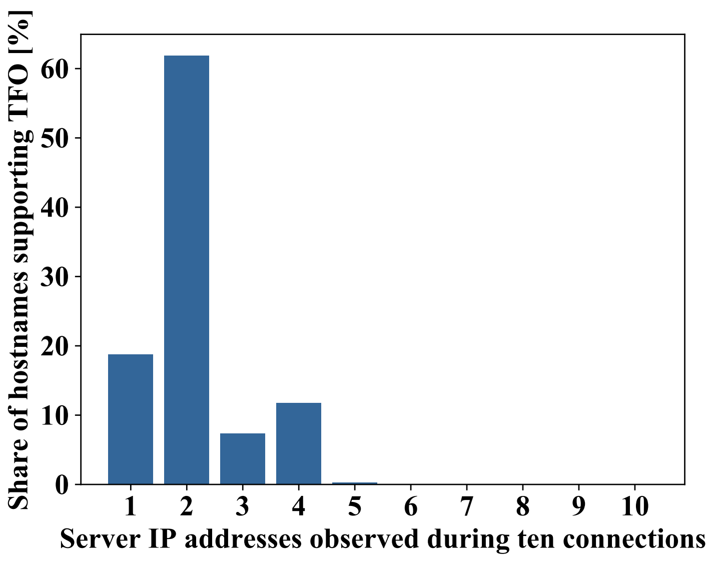


Figure 3.2: Share of hostnames with TFO-support plotted over the number of observed server IP addresses for ten connections.

the data center of our university. We used a dedicated Python script to send a TFO connection request using a static IPv4 address. Subsequently, we observed the server's response. When the responded SYN-ACK includes a TFO cookie, we consider this host to support TFO and the contrary otherwise. Note, that we did not attempt handshakes using a previously retrieved cookie. In total, we connected to 32 099 hostnames that we identified in our previous measurement (see Section 3.2.2) as sites supporting the TFO protocol. We connected to every hostname ten times with intervals of 45 minutes between each successive connection and observed the respective server IP addresses. This time interval was chosen for reasons of convenience because our test setup required about 40 minutes to initiate connections to the entire set of investigated hostnames. Note, that the selected time interval is significantly larger than the expiration time of DNS records of popular websites, which mostly expire within 5 minutes [28]. During this measurement, we experienced a failure rate of 1.2% that was mainly caused by unresponsive hosts.

About 94% of the investigated hostnames indicated, support for the TFO protocol at each of these connections. The remaining 4.8% can be attributed to hostnames that are served from multiple servers and not all of them support the TFO protocol. This includes servers not supporting TFO on a different IP address, but also on the same address in case anycast is used to reach multiple physical servers behind the same address. Figure 3.2 shows the number of observed IP addresses per site for the 30 218 hostnames always supporting TFO. Our results indicate, that at least 81% of the tested hostnames are served from several IP addresses. We also found that the second connection to a hostname introduces 11 876 new IP addresses compared to the first connection. As shown in Figure 3.3, the second TFO connection to a hostname fails in 39.3% of all attempts due to the fact that the hostname is served from a different IP address than the first. For the third connection to a hostname, we observe an average failure rate of 24.7%. On average, the tested hostnames are served each from 2.1 different IP addresses across our measurements. We conclude that the binding of TFO cookies to a specific server IP address presents a significant real-world performance limitation in the context of web browsing.

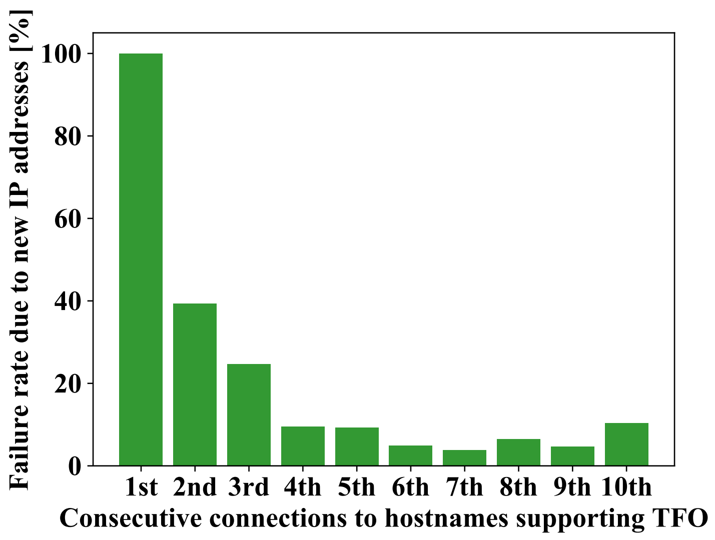


Figure 3.3: Failure rate to conduct abbreviated handshakes due to fresh server IP addresses plotted over successive connections to hostnames with TFO-support.

3.3 Tracking via TCP Fast Open

In this section, we begin by introducing a basic approach to track users via the TFO protocol. Based on that, we describe practical tracking scenarios using this approach. Afterward, we compare TFO-based tracking to IP-based tracking. Finally, we evaluate the default configuration of popular web browsers regarding TFO to assess the real-world impact of the presented tracking mechanism.

3.3.1 Basic Tracking Approach

Essential to TFO-based tracking are the cookies that the TFO protocol uses to authenticate a client upon consecutive connections to a server. These cookies are generated by the server and permit the attacker to identify clients with up to 16 bytes of entropy [5]. They enable the server to link all connections in which the same unique cookie is used and to attribute them to the same client. Moreover, a failed authentication as shown in Figure 3.1c allows the server to link the fresh *cookie_2* to the same client that was previously using *cookie_1*. Following TFO's specification, the client should then use the fresh *cookie_2* for subsequent connections to the same server. In total, the server can track its clients via Fast Open cookies in an essentially similar fashion as with the widespread HTTPS cookie.

Attacker Model Our attacker model assumes that the attacker can read network packets including their IP and TCP headers. The attacker is capable of extracting Fast Open cookies from the TCP headers and storing them for future reference in association with the respective tracking profile. As a limitation, the attacker is not capable to conduct user tracking based on any other protocol except for IP and TCP. The attacker cannot break cryptographic primitives. Hence, they cannot violate the confidentiality or authenticity of the TFO cookie without access to the respective secret key. The attacker has no direct access to a client and cannot violate the integrity of the software run by the client. Additionally, a host-based attacker located on the server has access to the secret key used to encrypt and authenticate TFO cookies and can thus issue custom TFO cookies.

3.3.2 Tracking Scenarios

While tracking by a single web server presents a privacy issue in itself, it is amplified if a tracker can identify a user across several visited websites. The remainder of this section, describes third-party tracking, tracking across virtual domains, and tracking by a network-based attacker, all of which allow tracking via the TFO protocol across multiple websites.

Scenario 1: Third-party Tracking Third-party tracking refers to a practice, where a party other than the targeted website can link website visits to the same user. This is a common practice on the Internet considering that Alexa Top 500 websites include on average 17.7 third-party trackers [7]. The presented tracking mechanism allows identifying users across all websites where a corresponding tracker is included as a third-party resource. However, to distinguish the various first-party sites, i.e., referrers, that a user visited, the tracker requires an additional identifier such as an HTTP referrer or a dedicated URL per first-party.

Scenario 2: Tracking Across Virtual Domains In virtual hosting, multiple virtual domains are hosted on a single server or pool of servers. This approach allows sharing resources like the IP address and server hardware across domains. When domain name $\mathscr{A}$ and $\mathscr{B}$ share the same IP address, then a TFO connection to both websites will contain the same cookie. Hence, an operator of a virtual hosting platform such as a Content Delivery Network (CDN) can link visits of the same user across the hosted virtual domains. In detail, a Fast Open cookie issued during a connection to domain name $\mathscr{A}$ will be reused by the client when connecting to domain name $\mathscr{B}$ allowing the operator of these hosts to link these connections based on the same Fast Open cookie to the same user.

Scenario 3: Tracking by a Network-based Attacker Since TCP itself provides no confidentiality, a passive, network-based attacker can observe the content of TCP headers as long as no protective measure is taken on lower protocol layers, e.g., IPsec. As a consequence, the attacker can use TFO cookies for tracking purposes as described above. This is particularly sensitive if the network-based attacker is located on the public Internet and targets a client that is situated in a local network which uses address translation (NAT). In that case, the attacker would be otherwise unable to distinguishing specific users. As a result, this undermines the efforts of protocols such as TLS 1.3 [26] that aim to protect against tracking by network-based attackers.

3.3.3 Comparison to Tracking via IP Addresses

In this section, we describe how the presented mechanism extends the capabilities of tracking via IP addresses despite TFO's requirement for matching client and server IP addresses to reuse cached Fast Open cookies. To illustrate this comparison, we introduce a scenario of devices sharing the same IP address and another scenario where a device is placed behind a Network Address Translator (NAT), whose publicly visible IP address changes dynamically.

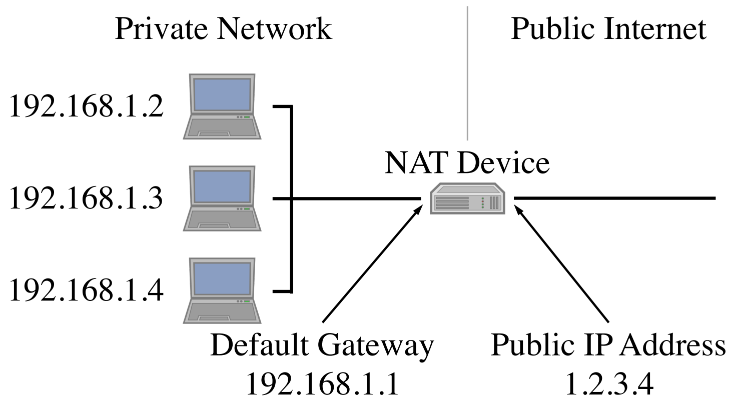


Figure 3.4: NAT device used for a private network.

Distinguishing Devices Sharing an IP Address

The Internet is dominated by the use of IP version 4 and the available address space is already exhausted [14]. This makes the sharing of IPv4 addresses necessary. Furthermore, the transition to IP version 6 is difficult and it is only supported by about a quarter of the most popular 10 000 websites and the global Internet users, respectively [28, 8]. Thus, the sharing of IP addresses is a common scenario from small home networks up to large carrier-grade NATs spanning several thousand devices. Tracking based on IP addresses becomes infeasible when a large number of devices share the same IP address as they form a large anonymity set. Tracking via Fast Open cookies does not have the same limitation. The tracker can provide each device with a unique Fast Open cookie. Thus, a client presenting this cookie during a future connection request can be clearly linked to the prior connection that was used to issue the cookie.

Prolonged Identification behind NAT

On the Internet, a significant share of users uses a dynamically assigned IP address [32]. Figure 3.4 shows a common network topology, where a user's device is located in a private network behind a NAT gateway that always uses the same local IP address. Furthermore, we assume that the public IP address of the NAT gateway changes dynamically. In this setup, user tracking via IP addresses stops when the NAT gateway receives a new public IP address because a tracker on the Internet cannot link the old to the new IP address. In the case of TFO, the client's static, local IP address always fulfills the requirement of a matching client IP address during the reuse of cached cookies. Thus, the client device reuses cached TFO cookies independently of its publicly visible IP address assigned to the NAT. In total, in the described setup, tracking via TFO cookies allows extending the feasible tracking periods compared to when using only the client's publicly visible IP address.

Note, that the tracking periods achievable via TFO are limited by the uptime of an OS, as a restart clears the corresponding TCP cache. However, especially mobile devices such as smartphones can achieve an uptime of several days or weeks under real-world conditions and thus allow for tracking periods of a similar duration. Another limiting factor is the TCP cache size, which under certain circumstances can lead to the eviction of older entries [2].

3.3.4 Evaluation

To explore the real-world feasibility of user tracking via the TFO protocol, we investigated popular web browsers on different operating systems. In total, we conducted eight browser experiments, whose methodology and results we present in the remainder of this section.

Status of TFO on the test systems

The privacy problems of TFO are relevant to all applications that use this protocol. However, within the scope of this work, we focus our investigation on popular web browsers because of their important role to protect users' web browsing behavior against online tracking without user consent. In our sample of popular web browsers, we included the Top 3 mobile and the Top 6 desktop browsers [27]. We tested those browsers on up-to-date versions of Android, iOS, Linux, macOS, and Windows 10 and investigated their support for the TFO protocol by analyzing the network traffic between browser and server. We find, that the deployment of the TFO protocol in popular browsers is still at an early stage. Thus, only Microsoft Edge on Windows 10 supports TFO by default. Firefox, Chrome, and Opera support the TFO protocol as an experimental feature on several operating systems as shown in Table 3.2. Note, that TFO is activated by default within Firefox Nightly and Firefox Beta under macOS and Windows 10, which indicates preparations to further deploy TFO across the Firefox platforms. Our tests for iOS 11 and Android Kernel 4.10 did not reveal any popular browser which supports TFO. Note, that the TFO implementation of Microsoft Edge did not work reliably within the IPv4 network stack. To overcome this issue, we tested this browser with an IPv6 network stack, while all other test systems deployed IPv4.

Feasible tracking periods

This measurement gives a lower boundary of feasible tracking periods via the presented approach. For that, we visited a website that supports TFO. In between different visits, we closed the browser tab that was in use and left the browser idle in the background of the operating system. After one hour, we attempted to revisit the same website served from the same IP address. By a manual analysis of the network traffic between the browser and the server we observed whether the browser attempted to use the cached Fast Open cookie from the first website visit to establish the fresh connection. If the browser makes use of the cookie, we can then assume that it is feasible to track users with the deployed test setup and for the duration of our test. Next, we repeated this measurement with a fresh TCP cache and increasing the delays between consecutive website visits up to ten days. We found that none of the IPv4-based test setups indicated a restriction of the feasible tracking period. Thus, we could track all Chrome, Firefox, and Opera setups for the entire test period of ten days as shown in Table 3.2. Furthermore, this indicates that the tested operating systems do not limit the feasible TFO tracking periods via the TFO protocol. For the Microsoft Edge browser, we were required to conduct this experiment on an IPv6 network stack, which diverts from our other browser test setups. We observed for the Windows 10 default configuration that the Edge browser utilizes temporary IPv6 addresses [18] within the available address block. As a privacy feature, the lifetime of these temporary addresses is limited to 24 hours by Windows 10. Thus, this test setup changes its global IPv6 address after 24 hours, even when the assigned IPv6 address block remains the same. However, the TFO

Browser/Test system	Status	Tracking periods	Tracking across					
			Third-parties	Virtual hosts	IP addr. changes	Private browsing modes	User applications	Browser restarts
Chrome v68/Ubuntu 18.04	support	unrestricted	viable	viable	blocked	viable	viable	viable
Firefox v61/Ubuntu 18.04	support	unrestricted	viable	viable	blocked	viable	viable	viable
Firefox v61/macOS 10.13	support*	unrestricted	viable	viable	blocked	viable	viable	viable
Firefox v61/Windows 10	support*	unrestricted	viable	viable	blocked	viable	viable	viable
Edge v42/Windows 10	default	24 hours	viable	viable	blocked	viable	viable	viable
Opera v54/Ubuntu 18.04	support	unrestricted	viable	viable	blocked	viable	viable	viable

*Activated by default within Firefox Nightly and Firefox Beta.

Table 3.2: TCP Fast Open default configuration of popular browsers. Note, that the experiments with the Edge browser only worked reliably with IPv6, while all other setups were tested with IPv4. Due to the feature of temporary IPv6 addresses, the default Windows 10 behavior issues a fresh temporary IPv6 address every 24 hours, which is then used by the Edge browser.

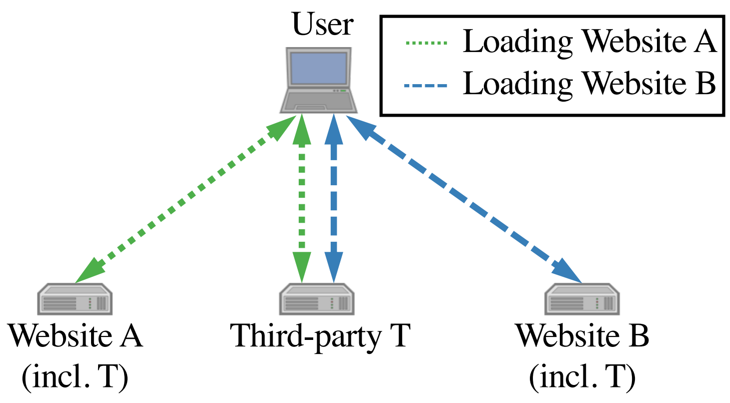


Figure 3.5: Testbed to measure browser behavior regarding third-party tracking.

protocol only uses cached Fast Open cookies if the source IP address of the test system is the same as in the TFO connection from which the cookie was retrieved. As a result, the observed tracking periods terminate with the change of the temporary IPv6 address.

Tracking across third-parties

This measurement investigates the feasibility of third-party tracking via the TFO protocol. Figure 3.5 shows a schematic of the deployed test setup for this experiment. To conduct this measurement, we require two websites $\mathscr{A}$ and $\mathscr{B}$ that include the same third-party $\mathscr{T}$. We visited website $\mathscr{A}$ and validated that the browser received a Fast Open cookie from the third-party $\mathscr{T}$ by manually analyzing the network traffic. After closing the browser tab in-use, we waited for 30 minutes for open TFO connections to time out [13]. We then visited website $\mathscr{B}$ and investigated the network traffic between the browser and the third-party $\mathscr{T}$. If the browser attempted to use the cached Fast Open cookie for the connection establishment with $\mathscr{T}$, we concluded that third-party tracking via TCP Fast Open is feasible with this browser. As shown in Table 3.2, none of the tested browsers applied mechanisms to prevent third-party tracking via Fast Open cookies. Thus, third-parties present on several websites can track the same user's visits across all those sites.

Tracking across virtual hosts

This experiment is used to investigate the feasibility of tracking across virtual hosts. It requires two websites whose DNS entries direct to the same IP address. We connected to one of these websites and afterward closed the browser tab and waited for 30 minutes to ensure that TCP connections to that website timed out. We then connected to the second website and monitored the respective network traffic of this connection. If the second connection uses the Fast Open cookie, which was retrieved during the connection to the first website, we conclude that tracking across virtual hosts is feasible with the tested browser. Our evaluation indicates that the investigated browsers do not prevent tracking across virtual hosts (see Table 3.2). Therefore, when multiple websites are served from the same IP address, the service operator can identify users across those hosted websites.

Tracking across IP address changes

This test investigates the browser behavior regarding TFO when the operating system gets a new IP address assigned. To assess this behavior, we visited a website and closed the browser tab afterward. While the browser was running idle in the background of the operating system, we assigned a new IP address to the device. Then, we revisited the website with the tested browser instance and monitored the network traffic of the connection. If the browser reuses the cached Fast Open cookie of the website, we can conclude that tracking across IP address changes is feasible. We observed, that user tracking is not feasible across IP address changes of the client as shown in Table 3.2. However, considering a common consumer setup, where devices reside in a private network that is connected to the Internet through a NAT gateway, such devices typically keep their local IP addresses unchanged indefinitely, since DHCP servers deterministically reassign the same local IP address based on unchanging features like the client's MAC address. In such a setup, the client's unchanging local IP address is independent of the public IP address, which is assigned to the NAT gateway. Thus, even after a change of the public IP address, the client will try to connect using the previously cached Fast Open cookies, which are bound to the unchanged local sender address. As a consequence, this allows a tracking server to learn a client's new publicly visible IP address and continue its tracking activities across the IP address change.

Tracking across private browsing modes

This experiment explores whether user tracking via TFO is feasible across browsers' default and private browsing mode. To assess this browser behavior, we visited a website in the default mode of a browser. Then, the respective browser tab was closed and the private browsing mode activated. While monitoring the network traffic, we then revisited the website. If the connection establishment in the private browsing mode used the previously retrieved Fast Open cookie, we could conclude that tracking across the browsing modes of the tested browser is feasible. As indicated in Table 3.2, all setups allow a remote online tracker to identify their user across changes of their browsing mode. This observed behavior presents a breach in the respective privacy modes, which aim to discard cookies at the end of each private session [16].

Tracking across browser restarts

This measurement tests whether tracking across browser restarts is feasible. To investigate this browser behavior, we first visit a website and retrieve a fresh Fast Open cookie. Then, we restart the browser and revisit the same website while we monitor the respective network traffic. If the browser reuses the cookie of the previous browser instance, then we conclude that tracking across browser restarts is feasible with the deployed setup. We find, that none of the tested browsers prevents tracking via Fast Open cookies across a browser restart.

Tracking across user applications

This experiment explores tracking across different user applications on the same device. To conduct the experiment, we retrieve a website and leave the respective browser idle in the background of the OS. Afterward, we use another application with support for the TCP Fast Open protocol such as another browser or curl to retrieve the same website. By monitoring the network traffic between the website and the operating system, we find out whether the second application reuses the TCP Fast Open cookie of the tested browser. If so, then tracking across user applications is feasible. Our results indicate, that user tracking across applications on the same client operating systems is viable for all tested setups (see Table 3.2).

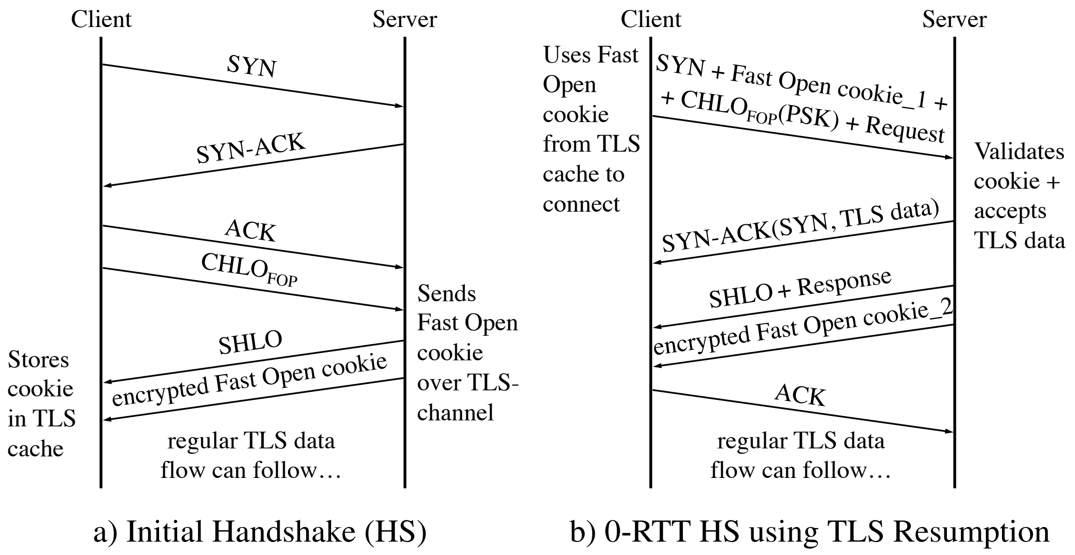

Client
Server
SYN
SYN-ACK
ACK
CHLO_FOP
SHLO
encrypted Fast Open cookie
Sends Fast Open cookie over TLS-channel
Stores cookie in TLS cache
regular TLS data flow can follow...
a) Initial Handshake (HS)
Client
Server
Uses Fast Open cookie from TLS cache to connect
SYN + Fast Open cookie_1 + + CHLO_FOP(PSK) + Request
Validates cookie + accepts TLS data
SYN-ACK(SYN, TLS data)
SHLO + Response
encrypted Fast Open cookie_2
ACK
regular TLS data flow can follow...
b) 0-RTT HS using TLS Resumption

Figure 3.6: Proposed handshakes in TCP Fast Open protocol using TLS 1.3 as a secure channel.

Summary As a summary of these browser measurements, we find that the use of the TFO protocol leads to huge privacy risks such as unrestricted tracking periods and cross-browser tracking. Furthermore, our results indicate that this tracking mechanism is very persistent and cannot be terminated by browser restarts or a change of the browsing mode. We recommend browser vendors to refrain from deploying the TFO protocol due to the presented privacy problems.

3.4 TCP Fast Open Privacy

In this section, we introduce the TCP Fast Open Privacy (TCP FOP) protocol, that addresses the performance and privacy limitations of TLS over TFO. Then, we describe the implementation of the novel TCP FOP protocol. Subsequently, we evaluate the privacy and performance provided by the TCP FOP protocol. Finally, to further substantiate the real-world applicability of TCP FOP, we analyze the effects of TCP protocol entrenchment on the proposed protocol.

3.4.1 Design of TCP FOP

TCP FOP builds upon the TFO protocol and continues the approach to use Fast Open cookies to reduce the latency of the connection establishment. These Fast Open cookies are generated by the server and sent over an encrypted channel to the client. Because there is no encrypted channel within TCP, we propose to use TLS for this purpose and to create a cross-layer solution. Our proposal requires an extension to TLS with an additional message type that allows the client to request Fast Open cookies and enables the server to provide such cookies over an encrypted channel. The client stores a received cookie along with the corresponding timestamp, the hostname, and a context identifier. The timestamp is used to limit the period for which a cached cookie can be used to attempt an abbreviated handshake. The hostname is authenticated within the TLS handshake and can therefore be associated with the cookie. The cached cookies are then used for abbreviated handshakes for matching hostnames independently of the server's IP address. This modification allows TCP FOP to anticipate the load balancing of websites across several IP addresses. However, it requires the involved servers to share the cryptographic secret, that is used to encrypt/decrypt the corresponding Fast Open cookies. The context identifier is provided by an application and marks the context in which the cookie was retrieved. Thus, a browser can mark for example each Fast Open cookie from a third-party with the context identifier of the corresponding first party. As a result, this third-party cannot track the client across several first party websites via the Fast Open cookie because the cached cookie can only be used for an identical context identifier. By changing the context identifier for events like browser restarts or changes between browsing modes, user tracking across these events/contexts is mitigated. Thus, to restrict user tracking via host-based attackers, applications need to provide context identifiers to limit the usage of retrieved Fast Open cookies. Additionally, Fast Open cookies should not be reused to set up several abbreviated connections.

In the following, we describe the details of the TCP FOP handshakes. Figure 3.6 shows a schematic of the proposed handshakes that use TLS version 1.3 [26] as an encrypted channel.

Initial handshake: First, client and server use a standard three-way handshake to establish a TCP connection. Second, the client starts the modified TLS handshake with a *client hello message* (CHLO_{FOP}) that indicates support for TCP FOP. The server responds with a *server hello message* (SHLO) that completes the cryptographic handshake and allows sending subsequent messages over the established TLS channel. Afterward, the server generates a fresh Fast Open cookie for the client and sends it over the TLS channel. Finally, the client stores this message along with a timestamp, the hostname, and a context identifier in its TLS cache.

API Name	Description
TCP_Fast_Open_COOKIE_GEN	This server-side API enables an application to retrieve a Fast Open cookie for a specific client from the kernel.
TCP_Fast_Open_COOKIE_SET	This client-side API enables an application to fill the TCP cache with a specific cookie.

Table 3.3: Kernel API changes

0-RTT handshake: To establish a new secure connection to the same website with an abbreviated 0-RTT handshake, the TLS implementation checks whether a valid Fast Open cookie is available in its TLS cache. A Fast Open cookie is valid if it has not yet expired and the respective hostname and context identifier match. Assuming a valid cookie, the client then calls a kernel function with the cookie as a parameter that attempts to open a TCP FOP connection. Then, the client sends a SYN message to the server which includes the respective Fast Open cookie as shown in Figure 3.6b. Subsequently, the client starts its TLS handshake without waiting for the server's response. Figure 3.6b shows a 0-RTT TLS handshake using session resumption for connection establishment and directly sending an encrypted data request. The TLS resumption handshake decreases the delay and saves expensive cryptographic operations compared to a full TLS handshake by leveraging key material exchanged in an earlier session. Upon receiving the client's messages, the server validates the contained Fast Open cookie before accepting the payload in the SYN packet. The server then accepts the connection and the received TLS data by sending a SYN-ACK message which acknowledges the message of the client. Otherwise, the server acknowledges the connection only and drops the TLS data, which leads to a rejected 0-RTT handshake. Following the flow of an accepted 0-RTT handshake attempt as shown in Figure 3.6b, the server's TLS application validates the client's session resumption data. Assuming this data to be valid, the server answers by sending a SHLO, a response to the client's request and a fresh Fast Open cookie. In total, this protocol flow (see Figure 3.6b) allows the client to directly send encrypted application data without waiting for a response from the server.

Note, that in TLS version 1.3 [26] the session resumption mechanism should not reuse identifiers for connection establishment to prevent tracking by a network-based attacker. To extend this protection against this attacker to the TCP layer, a Fast Open cookie should not be reused for setting up multiple connections.

3.4.2 Implementation of TCP FOP

To assess the feasibility of TCP FOP, we implemented it in the Linux 4.18 kernel and in the wolfSSL TLS library. Our implementation required only minor modifications of in total about 300 lines of code (LoC) for Linux and 400 LoC for wolfSSL, including comments and debug output.

Kernel support

The Linux Kernel 4.18 already supports TCP Fast Open. Thus, we largely reused the available TCP Fast Open implementation and as a noteworthy change, we added two new APIs to the kernel as shown in Table 3.3. The API *TCP_Fast_Open_COOKIE_GEN* enables the server's TLS application to retrieve a fresh Fast Open cookie for a specific client connection from its kernel. For the client, we added the inverted API call which allows including a Fast Open cookie into the kernel's cache before the subsequent connection establishment.

Our prototype aims to evaluate that the presented cross-layer approach adds no substantial complexity to TCP and TLS. The implemented *TCP_Fast_Open_COOKIE_GEN* API works independently of TCP's connection handling. The *TCP_Fast_Open_COOKIE_SET* API provides a cookie that can be subsequently used within a TCP handshake. However, this affects only the initialization of TCP's connection establishment. This creates no external constraints on the handshake protocol itself. Furthermore, a Linux kernel API to delete specific TCP Fast Open cookies [2] already exists, which also modifies the initialization of TCP's connection establishment. The proposed *TCP_FOP_COOKIE_SET* API only provides the logical counterpart to the existing deletion mechanism. Our implementation of TCP FOP introduces only lightweight modifications to TCP, which lead to no external constraints of TCP's connection handling.

TLS support

We implemented our prototype as part of the open-source TLS library wolfSSL that provides support for TLS 1.3. For practicality reasons, we decided to add Fast Open cookies to the session resumption mechanism of TLS 1.3. However, for the productive use of TCP FOP we recommend an implementation as a dedicated TLS mechanism, i.e., an extension, but leave that for future work. We use TLS only as a data channel for the cookie and to place the cookie in the TCP cache during the establishment of a new connection. Therefore, our implementation does not affect TLS's cryptographic components and its connection handling.

In the following, we briefly describe our workaround of implementing TCP FOP by adapting the session resumption mechanism of TLS 1.3. On the server-side, we include a new Fast Open cookie which we generated with the *TCP_Fast_Open_COOKIE_GEN* API into each TLS session resumption ticket. Thus, the TLS server would subsequently send NewTicket messages which contain the Fast Open cookie and the original session resumption ticket. Upon receiving such a message, the client stores it along with its own connection state such as encryption keys within its TLS cache. To establish a subsequent connection to the same hostname, the client first validates that a session resumption with that website complies with its privacy configuration. Assuming that this validation was successful, the client extracts the Fast Open cookie from the cached session resumption ticket and stores it in the kernel's TCP cache using the *TCP_Fast_Open_COOKIE_SET* API. This API identifies the receiver based on its IP address. Note, that this approach requires the application to learn the IP address of a given hostname as it is common for applications doing their independent DNS lookups. In case, the application delegates the name resolution to the kernel, an additional Kernel API would be required to associate a Fast Open cookie with the resolved IP address of a given hostname. Subsequently, the client uses the session resumption ticket to establish a resumed TLS session over TCP's Fast Open extension. After this connection is established, the client deletes the used Fast Open cookie within the cache. To avoid client identification based on session resumption tickets through a network-based attacker, the client shall not reuse the same ticket to set up connections with the server. Analogous, each fresh session resumption ticket is required to contain a fresh Fast Open cookie, so that a client cannot be identified by linking cookies.

3.4.3 Evaluation of TCP FOP

This section starts with an assessment of the privacy properties of TCP FOP and a subsequent comparison to the previously existing TFO protocol. Next, a performance evaluation of TCP FOP is presented based on experiments with the implemented prototype. To investigate the real-world applicability of TCP FOP, this section ends with a feasibility analysis studying possible deployment issues.

Privacy Evaluation

Tracking via Fast Open cookies is independent of alternative tracking mechanisms such as HTTP cookies, browser fingerprinting, or IP addresses. To protect the privacy of users, a network-based attacker can observe each utilized Fast Open cookie only once, namely during the 0-RTT handshake of TCP FOP. From the perspective of a network-based attacker, these Fast Open cookies are encrypted and single-use data blocks and thus cannot be linked to a specific

Privacy characteristic	TCP Fast Open Protocol	TCP Fast Open Privacy Protocol
Tracking via network-based attacker	viable	blocked through single-use cookies & encrypted channel
Tracking across third-parties	viable	blocking possible through context identifier
Tracking across virtual hosts	viable	blocking possible through context identifier
Tracking across private browsing modes	viable	blocking possible through context identifier
Tracking across browser restarts	viable	blocking possible through context identifier
Tracking across user applications	viable	blocking possible through context identifier
Tracking across IP address changes	blocked	blocking possible through context identifier
Tracking periods	unrestricted	restriction possible through expiration of cookies

Table 3.4: Comparison of privacy characteristics between the TCP Fast Open protocol and our TFO proposal utilizing TLS as a secure channel.

user. Therefore, the TCP FOP protocol prevents attackers to use Fast Open cookies to re-identify specific users and to establish user profiles. Hence, tracking by network-based attackers is not possible anymore, which is the most important privacy achievement of TCP FOP.

However, when facing host-based attackers, the TCP FOP protocol faces a performance versus privacy tradeoff. The best user privacy is achieved when doing initial handshakes only, while the best performance is achieved during sequences of 0-RTT handshakes. However, host-based attackers can link 0-RTT handshakes to the same user by linking their Fast Open cookies. In an initial handshake, the user does not reuse Fast Open cookies from a prior connection, therefore user tracking is prevented. However, an initial TCP FOP handshake requires an additional round-trip time compared to the 0-RTT connection establishment, which impacts performance. TCP FOP provides a mechanism to balance this tradeoff in the context of specific applications. For that, Fast Open cookies expire after a certain lifetime, which limits the maximum tracking period to this lifetime. The performance impact of such a lifetime approach has been studied in prior research work [29]. This study of users' browsing behavior indicates that 17.7% of all revisits of websites can use 0-RTT handshakes, if the lifetime of Fast Open cookies is set to five minutes. Increasing this lifetime to 60 minutes allows using 0-RTT handshakes for 48.3% of all website revisits. Thus, this approach allows to strictly enforce an upper limit for the feasible tracking period, while short Fast Open cookie lifetimes still enable a significant share of 0-RTT connection establishments.

The second countermeasure of the TCP FOP protocol against host-based attackers uses context identifiers associated with cached Fast Open cookies. These context identifiers are intended to strictly enforce privacy policies in applications and therefore prioritize privacy over performance. This approach restricts a client to use only cached Fast Open cookies for 0-RTT handshakes if their context identifier is identical to the active context of the application. Defining the context based on the visited party, virtual host, IP address, browsing mode, user application, and browser session logically excludes the tracking approaches observed in Section 3.3.4. For example, by switching to the private browsing mode, previously cached Fast Open cookies cannot be used for 0-RTT handshakes, as they have been retrieved from the context of a different browsing mode. However, each additional dependency on the context identifier causes a further restriction for the use of cached Fast Open cookies, which will eventually affect the ratio of initial and 0-RTT handshakes. Table 3.4 summarizes our findings from Section 3.3.4 for the TFO protocol and compares them to the privacy characteristics of TCP FOP. We find that the TCP FOP protocol can mitigate all privacy issues of TLS over TFO.

Performance Evaluation

We evaluate the performance of the TCP FOP protocol in two parts: First, we conduct experiments to investigate whether the usage of the proposed TCP FOP/TLS incurs a delay overhead compared to connections using TFO/TLS or standard TCP/TLS. Second, we study the performance of TCP/TLS, TFO/TLS, and TCP FOP/TLS in a scenario with real-world load-balancing.

Experiment using the TCP FOP Prototype We compare our implemented prototype of TCP FOP to implementations of standard TCP and TFO. For that, we compare the required time to download a small web page from a single host using one of these three transport protocols in combination with TLS 1.3. For this experiment, we use two virtual machines, one acting

Network	TCP/TLS		TFO/TLS		TCP FOP/TLS	
latency [ms]	Initial [ms]	Resumed [ms]	Initial [ms]	Resumed [ms]	Initial [ms]	Resumed [ms]
≈0.3	28.9 (3.6)	20.2 (2.7)	29.9 (3.5)	22.3 (2.9)	29.6 (3.6)	22.2 (2.9)
50 ms	189.8 (2.5)	132.6 (1.7)	190.0 (2.4)	83.7 (1.9)	190.0 (2.6)	83.8 (2.2)
100 ms	340.2 (2.1)	233.1 (1.4)	340.3 (2.1)	135.1 (1.6)	340.7 (2.1)	135.4 (1.6)
150 ms	490.3 (1.8)	332.9 (1.3)	490.7 (1.8)	185.3 (1.4)	491.1 (1.8)	185.7 (1.4)

Table 3.5: Mean duration to establish a connection between the client-server pair and to download a small website using TCP/TLS, TFO/TLS, and TCP FOP/TLS. The standard deviation of the mean duration is denoted in brackets. Initial connections use the initial handshake of the respective TCP variant and a full TLS 1.3 handshake. Resumed connections utilize a 0-RTT handshake if supported by the respective TCP variant and a resumed 0-RTT TLS 1.3 connection establishment.

as web server and the other as client. The virtualization is realized on the same host using qemu 2.8 and libvirt 3.0.0. This test setup leads to short network latencies with an average ping of 0.3 milliseconds (ms) between the virtual machines. The host of the virtual machines was equipped with an Intel Xeon E5-1660 v4 CPU with 32 GB of RAM and ran Debian stretch. The client and the server machines were set up with 4 GB of RAM and were running an Ubuntu 18.10 with our modified Linux kernel (see Section 3.4.2) that supports the TCP FOP protocol. The server ran the example server program shipped with our modified wolfSSL library. The program responds to a successful connection establishment with a short string. The client ran the corresponding example client program of wolfSSL that establishes a TLS session to the server, waits for the short string from the server and terminates the TLS session upon its reception. Note that we used our modified wolfSSL implementation as described in Section 3.4.2 for this measurement. In our experiment, we established and resumed a new TLS connection via standard TCP, TFO, or TCP FOP. All of these TLS connections used the forward-secure cipher suite *TLS_AES_128_GCM_SHA256*. The initial TLS connection uses an initial handshake of the corresponding TCP variant. The 0-RTT TLS resumption handshake uses an abbreviated TCP connection establishment if supported by the respective TCP variant. To account for skews in the measurements, we repeated the experiment 1000 times and measured the elapsed wall-clock time. We conducted our measurements with the client's network interface configured to simulate network latencies of 0.3 ms, 50 ms, 100 ms, and 150 ms with iproute2's `tc` program. We recorded and inspected the network traffic of the virtual network interface to validate a correct behavior of our evaluation setup.

The results of the measurement are shown in Table 3.5. We find that for minimal network latencies of 0.3 ms TLS over the standard TCP provides the best performance results, while the usage of TFO/TLS and TCP FOP/TLS leads to similar values. Especially, in the case of a resumed handshake the usage of the standard TCP provides a performance gain of almost ten percent. We assume that TFO and TCP FOP have a computational overhead by generating, validating, and handling the Fast Open cookies that incurs this delay.

In total, our results indicate only small differences between the performance of the initial handshake measurements that are consistently less than a millisecond between the investigated TCP variants for the same network latency. For resumed handshakes, the performance benefits of TFO/TLS and TCP FOP/TLS are significant for larger network latencies. These protocols complete the resumption handshake with time savings larger than 50% for network latencies of 50 ms and above compared to TLS over standard TCP. These benefits account for the saved round-trip time of the 0-RTT handshakes of TFO and TCP FOP. Between the performance of TFO/TLS and TCP FOP/TLS, we find only insignificant differences. As a result of this measurement, we find that TFO/TLS and TCP FOP/TLS have similar computational overhead.

Simulation considering Load-balancing Clients using the TCP FOP/TLS associate a retrieved Fast Open cookie with the hostname of the respective online service. This allows them to attempt 0-RTT handshakes with online services with matching hostnames independently of the server's IP address. Conversely, the TFO/TLS is restricted to conduct 0-RTT handshakes only when the server's IP address matches the one associated with the cached Fast Open cookie. This simulation investigates the performance benefits of this adapted design of the TCP FOP/TLS protocol stack.

Our test setup consists of a client, a network link, and a website. For the network link, we assume a mobile LTE connection, with a round-trip time of 60 ms as it is common in the U.S. [20].

Note, that the round-trip time for 3G and WiFi connections are on average longer than for LTE connections [19]. Statistically, an average website requires 20 TCP/TLS connections to several hosts for its retrieval [31]. To resemble a real-world website, our test website directly links to 19 resources, each of them on a separate host. From the perspective of a domain tree, these 19 hosts are on the same hierarchical level. Furthermore, we assume that all hosts in this test setup support the TCP FOP protocol and use on average the same load-balancing approaches as observed in Section 3.2.2 for hosts supporting TFO. The client measures the elapsed wall-clock time to establish connections to all 20 hosts that need to be involved to retrieve the website.

Table 3.6 summarizes the results for successive revisits of the test website. As the TCP FOP/TLS protocol stack can establish 0-RTT connections independently of the IP address associated with a hostname, it saves on each revisit of the website two round-trip times compared to the initial website visit. One RTT can be saved when connecting to the primary host, and another RTT can be saved by successfully establishing 0-RTT connections to the 19 secondary hosts. For each website revisit with the TCP FOP/TLS, this reduces the delay overhead for establishing connections to all 20 hosts by 120 ms compared to the initial website visit. As indicated in Figure 3.3, the failure rate of the TFO/TLS protocol stack depends on the number of prior visits to a website. We used a tree diagram to compute the probabilities of saving zero, one, or two RTT during the connection establishment with all 20 hosts. We find, that for the first revisit the probability of saving a RTT during the connection establishment to all hosts is 60.7% and on average, the delay overhead is reduced by 36.4 ms. Note, that the saving of the TCP FOP/TLS protocol stack for the same task is more than three times higher with 120 ms. For the second and third revisit to the website, the achieved reductions are 45.5 ms and 63.1 ms, respectively. Thus, we observe that the TCP FOP/TLS protocol stack significantly outperforms TFO/TLS, if real-world load-balancing of websites is considered.

Feasibility Analysis

Efforts to deploy alternative TCP versions on the Internet indicate that middleboxes such as Firewalls and NAT devices modify or block unfamiliar TCP packets [25]. Hence, even simple changes to TCP require a long time before they exhibit a significant deployment [9]. The TFO protocol faces similar problems that hinder its rapid deployment on the web [21, 5]. The presented TCP FOP embeds a standard TCP three-way handshake in the initial handshake. This standard handshake is common on the web and thus will not cause any issues with middleboxes [25]. The exchanged messages in TCP FOP's 0-RTT handshake are identical to the messages in the TFO protocol (see Figure 3.1b and c). Note, that for privacy reasons the *cookie_2* of a rejected 0-RTT handshake should be discarded by the client and not be used to establish new TCP FOP connections. These identical protocol flows for TFO's and TCP FOP's 0-RTT handshakes avoid further deployment issues. Thus, middleboxes supporting TFO's 0-RTT handshake will by default also support the TCP FOP protocol. As a result, the deployment of TCP FOP/TLS on the Internet is not causing additional compatibility issues beyond the ones from the TFO/TLS protocol stack.

Compared to the usage of TFO, the cross-layer optimization TCP FOP requires TLS libraries to control Fast Open cookies. As a drawback, this interconnects TCP and TLS impacting the portability and maintenance of TCP FOP. However, with *Kernel TLS* there exists an implemented example for a performance-optimization causing a similar drawback between TLS libraries and Kernel functions [11]. In detail, *Kernel TLS* aims to save resources by delegating some

	1^{st} Revisit		2^{nd} Revisit		3^{th} Revisit	
Simulation	TFO/TLS	TCP FOP/TLS	TFO/TLS	TCP FOP/TLS	TFO/TLS	TCP FOP/TLS
Probability to save zero RTT	39.3%	0.0%	24.6%	0.0%	8.1%	0.0%
Probability to save one RTT	60.7%	0.0%	75.1%	0.0%	78.5%	0.0%
Probability to save two RTT	0.0%	100.0%	0.3%	100.0%	13.4%	100.0%
Mean delay overhead over LTE	-36.4 ms	-120.0 ms	-45.5 ms	-120.0 ms	-63.1 ms	-120.0 ms

Table 3.6: Analysis of the delay overhead of TFO/TLS and TCP FOP/TLS compared to TCP/TLS. We simulate the retrieval of a sample website and consider load-balancing as observed in Section 3.2.2. We assume a RTT of 60ms for the LTE connection.

expensive computations from the user space to the kernel space. Overall, we find that TCP FOP provides sufficient performance and privacy benefits to justify a cross-layer solution.

Summary Based on our evaluation, we find that TCP FOP/TLS significantly improves the privacy and real-word performance of TLS over TFO. With respect to privacy, all identified privacy issues of TLS over TFO can be addressed by TCP FOP/TLS. From a performance perspective, the TCP FOP/TLS protocol stack outperformed TLS over TFO in our test scenario that resembles the retrieval of an average real-world website. Furthermore, we conclude that TCP FOP/TLS will experience fewer deployment issues than TLS over the TFO protocol.

3.5 Related Work

To the best of our knowledge, we are the first to report on the privacy aspects of the TFO protocol. While research work on privacy issues within TCP that allows distinguishing clients or servers based on their TCP timestamps [17, 22] exists, our reported storage-based tracking mechanism is unrelated to such tracking approaches via the TCP timestamps. A similar tracking mechanism has been reported for the QUIC transport protocol [30]. However, QUIC's address validation tokens are distributed via an encrypted channel and do not allow a passive network observer to link different connections to the same user. Furthermore, TLS session resumption mechanisms [29] enable user tracking. Compared to the TFO protocol, Session resumption in TLS 1.3 does not enable a passive network observer to track a user's online activities. Thus, the TFO protocol provides substantially lower privacy guarantees than TLS 1.3. Additionally, prior research includes a proposed TCP extension which directly allows encrypting TCP packets [4]. This approach significantly increases the complexity of TCP by including typical TLS functionality. However, our proposed TCP FOP aims to be a lightweight protocol modification where Fast Open cookies are issued via a TLS-encrypted channel. The limitation of the TFO protocol to anticipate load balancing with multiple server IP addresses has been pointed out in prior work [12]. We contributed by investigating this limitation under real-world conditions and find that approx. 40% of the first website revisits fail to establish an abbreviated connection setup. TCP FOP presents a novel protocol which anticipates web server load balancing to achieve a higher share of abbreviated handshakes and fully protects against tracking via network-based attackers. To the best of our knowledge, we are the first to present such a cross-layer approach for the TLS over TFO stack.

3.6 Conclusion

TFO provides considerable latency improvements compared to TCP's three-way handshake. However, its usage on the Internet raises alarming privacy concerns. Therefore, we urge vendors of operating systems and browsers to discourage the deployment of the TFO protocol. To address the privacy problems at hand, we designed and implemented the TCP FOP protocol. Our analysis indicates that TLS over TCP FOP protects against user tracking by network-based attackers. Furthermore, the protocol can also restrict third-party tracking, enables each application to control its privacy properties and to balance the trade-off between lower delays and privacy protection. To that end, applications can influence the lifetime of cached Fast Open cookies, which represents the maximum feasible tracking period. TCP FOP/TLS not only provides better privacy protection, but it also provides significant performance gains in terms of delay compared to TLS over TFO. TCP FOP/TLS can carry out abbreviated TCP handshakes even when the website is served from multiple IP addresses, e.g., when being part of server load balancing. Our measurements indicate that TFO/TLS fails to establish an abbreviated handshake with a chance of 39.3% during the first revisit of a website. We attribute this mainly to server load balancing under which TCP FOP/TLS will always be able to carry out an abbreviated handshake. We conclude that based on our evaluation the proposed TCP FOP protocol leads to substantial enhancements of the performance and privacy of TLS over TCP Fast Open.

Acknowledgment

We thank the anonymous reviewers for their insightful comments and suggestions. We also thank our shepherd, Alan Mislove, for providing valuable feedback and guidance in the revision process. This work is supported in part by the German Federal Ministry of Education and Research under the reference numbers 16KIS0381K and 16KIS0922K.

3.7 References

[1] Alexa Internet Inc. 2018. Alexa Top 1,000,000 Sites. Retrieved January 21, 2019 from http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

[2] Julian Anastasov. 2018. IP-TCP_METRICS. Retrieved January 21, 2019 from man7.org/linux/man-pages/man8/ip-tcp_metrics.8.html

[3] Praveen Balasubramanian. 2019. Privacy problems of TCP Fast Open. Retrieved May 21, 2019 from mailarchive.ietf.org/arch/msg/tcpm/7QtnB9FCF-pKeUpNt64woJ-kCy8

[4] Andrea Bittau, Michael Hamburg, Mark Handley, David Mazieres, and Dan Boneh. 2010. The case for ubiquitous transport-level encryption.

[5] Yuchung Cheng, Jerry Chu, Sivasankar Radhakrishnan, and Arvind Jain. 2014. TCP Fast Open. RFC 7413.

[6] Peter Eckersley. 2010. How unique is your web browser?. In *PET Symposium*. Springer.

[7] Steven Englehardt and Arvind Narayanan. 2016. Online tracking: A 1-million-site measurement and analysis. In *CCS*.

[8] Google LLC. 2019. Google IPv6 Statistics. Retrieved August 20, 2019 from https://www.google.com/intl/en/ipv6/statistics.html

[9] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. 2011. Is it still possible to extend TCP?. In *IMC*.

[10] HTTP Archive. 2018. Report: State of the Web. Retrieved January 21, 2019 from https://www.httparchive.org/reports/state-of-the-web

[11] Kernel development community. 2019. Kernel TLS. Retrieved August 20, 2019 from www.kernel.org/doc/html/latest/networking/tls.html

[12] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The QUIC transport protocol: Design and Internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*.

[13] Linux man-pages project. 2018. tcp - TCP protocol. Retrieved January 21, 2019 from man7.org/linux/man-pages/man7/tcp.7.htmll

[14] Kieren McCarthy. 2019. OK, this time it's for real: The last available IPv4 address block has gone. Retrieved August 20, 2019 from https://www.theregister.co.uk/2018/04/18/last_ipv4_address/

[15] Mozilla Corporation. 2018. User tracking via TCP Fast Open. Retrieved January 21, 2019 from bugzilla.mozilla.org/show_bug.cgi?id=1500224

[16] Mozilla Foundation. 2018. Private Browsing - Use Firefox without saving history. Retrieved January 21, 2019 from https://support.mozilla.org/en-US/kb/private-browsing-use-firefox-without-history

[17] Steven J Murdoch. 2006. Hot or not: Revealing hidden services by their clock skew. In *CCS*.

[18] Dr. Thomas Narten, Richard P. Draves, and Suresh Krishnan. 2007. Privacy Extensions for Stateless Address Autoconfiguration in IPv6. RFC 4941.

[19] OpenSignal. 2014. LTE Latency: How does it compare to other technologies? Retrieved January 21, 2019 from opensignal.com/blog/2014/03/10/lte-latency-how-does-it-compare-to-other-technologies/

[20] OpenSignal. 2018. State of Mobile Networks: USA (July 2018). Retrieved January 21, 2019 from opensignal.com/reports/2018/07/usa/state-of-the-mobile-network

[21] Christoph Paasch. 2016. Network support for TCP Fast Open. Retrieved January 21, 2019 from nanog.org/sites/default/files/Paasch_Network_Support.pdf

[22] Libor Polcák, Jakub Jirásek, and Petr Matousek. 2014. Comment on" Remote Physical Device Fingerprinting". *IEEE Trans. Dependable Sec. Comput.* 11 (2014).

[23] Jon Postel. 1981. Transmission Control Protocol. RFC 793.

[24] Jon Postel and Joyce K. Reynolds. 1994. Assigned Numbers. RFC 1700.

[25] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. 2012. How hard can it be? designing and implementing a deployable multipath TCP. In *NSDI*.

[26] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446.

[27] StatCounter. 2018. Desktop Browser Market Share Worldwide. Retrieved January 21, 2019 from http://gs.statcounter.com/browser-market-share

[28] Erik Sy. 2019. Enhanced Performance and Privacy via Resolver-Less DNS. *arXiv preprint arXiv:1908.04574* (2019).

[29] Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2018. Tracking Users Across the Web via TLS Session Resumption *(ACSAC '18)*.

[30] Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2019. A QUIC Look at Web Tracking. *PET Symposium* 3.

[31] Erik Sy, Moritz Moennich, Tobias Mueller, Hannes Federrath, and Mathias Fischer. 2019. Enhanced Performance for the encrypted Web through TLS Resumption across Hostnames. *arXiv preprint arXiv:1902.02531* (2019).

[32] Yinglian Xie, Fang Yu, Kannan Achan, Eliot Gillum, Moises Goldszmidt, and Ted Wobber. 2007. How Dynamic Are IP Addresses? *SIGCOMM Comput. Commun. Rev.* 37, 4 (Aug. 2007).

4 Tracking Users across the Web via TLS Session Resumption

Summary of this publication

Citation	Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2018. Tracking Users Across the Web via TLS Session Resumption (**Outstanding Paper Award**). In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC '18).* ACM, New York, NY, USA, 289–299. https://doi.org/10.1145/3274694.3274708
Ranking	CORE: A GGS: A+ Microsoft Academics: [A|A++]
Status	Published
Type of paper	Research paper
Aim	This paper studies the privacy impact of TLS session resumption mechanisms and analyzes the trade-off between privacy and performance for these mechanisms.
Methodology	Applied methods include a privacy review on the data flow of TLS and empirical evaluations based on browser and web server measurements as well as an analysis of DNS traffic.
Contribution	This article provides the first description of user tracking via TLS session resumption. To assess the real-world impact of this tracking mechanism, this work studies the TLS configuration of 48 web browser and the Alexa Top Million Sites. The results suggest, those session resumption mechanisms are a commonly deployed performance-optimization on the web and several popular browsers enable tracking periods of at least 24 hours via this approach. Subsequently, this paper introduces a novel *prolongation attack*, which allows extending the tracking period beyond the session resumption lifetime. Simulating characteristic browsing behavior based on real-world DNS traffic, this prolongation attack can yields for an average user tracking periods of up to eight days by assuming a session resumption lifetime of 24 hours. Furthermore, the paper highlights that a session resumption lifetime of seven days, which presents the recommended upper limit in the TLS 1.3 standard, allows tracking up to 65% of all users permanently by at least one hostname in the DNS dataset. To mitigate these privacy issues, the paper proposes several changes to the TLS standard and the browser default configuration. While the disabling of TLS session resumption is the most effective privacy protection, it results in a performance loss for website revisits. Therefore, the paper investigates the performance versus privacy (tracking period) trade-off at hand. It finds, that short lifetimes of session resumption mechanisms such as 60 minutes al-

	low already resumptions in 48.3% of the connection establishments. Furthermore, the presented results suggest that session resumption lifetimes longer than 24 hours provide only insignificant additional performance benefits.
Co-authors' contribution	The article was co-authored by Christian Burkert, Prof. Dr. Hannes Federrath and Prof. Dr. Mathias Fischer. The co-authors assisted to revise and refine this paper.

Abstract

User tracking on the Internet can come in various forms, e.g., via cookies or by fingerprinting web browsers. A technique that got less attention so far is user tracking based on TLS and specifically based on the TLS session resumption mechanism. To the best of our knowledge, we are the first that investigate the applicability of TLS session resumption for user tracking. For that, we evaluated the configuration of 48 popular browsers and one million of the most popular websites. Moreover, we present a so-called prolongation attack, which allows extending the tracking period beyond the lifetime of the session resumption mechanism. To show that under the observed browser configurations tracking via TLS session resumptions is feasible, we also looked into DNS data to understand the longest consecutive tracking period for a user by a particular website. Our results indicate that with the standard setting of the session resumption lifetime in many current browsers, the average user can be tracked for up to eight days. With a session resumption lifetime of seven days, as recommended upper limit in the draft for TLS version 1.3, 65% of all users in our dataset can be tracked permanently.

Keywords: Session IDs, Session Tickets, PSK Identity, Tracking Period, Browser Measurement

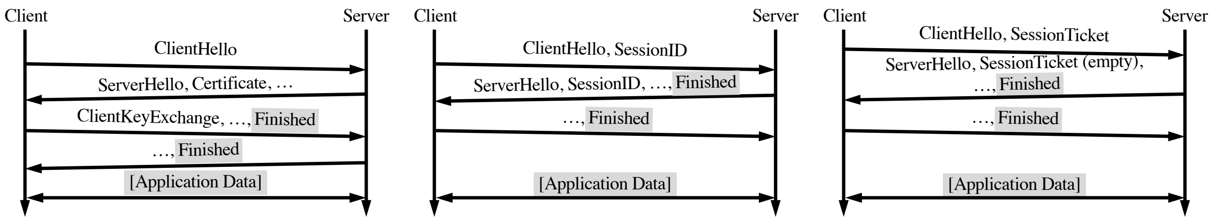
4.1 Introduction

User tracking via HTTP cookies and browser fingerprinting is a reality [2, 15, 4]. Tracking mechanisms are commonly used to observe conversions, namely whether an advertisement on website A leads to a desired user activity on website B. Herrmann et al. [5] revealed that temporary tracking mechanisms can be used to identify users based on their characteristic browsing patterns over longer time periods. They found that $85,4\%$ of users can be identified based on their browsing behaviour if the temporary tracking mechanism lasts up to 24 hours. Also, the creation of long-term browsing profiles is possible, if a tracker can observe a large share of a user's browsing activity. Big players like Google and Facebook leverage the wide-spread use of their advertising networks and social plugins to track users across websites and gain detailed user profiles.

However, as users are increasingly aware of the privacy threat from tracking, they use privacy-friendly browsers, private browsing modes, and browser extensions to restrict tracking practices such as HTTP cookies. Browser fingerprinting got more difficult, as trackers can hardly distinguish the fingerprints of mobile browsers. They are often not as unique as their counterparts on desktop systems [4, 12]. Tracking based on IP addresses is restricted because of NAT that causes users to share public IP addresses and it cannot track devices across different networks.

As a result, trackers have an increased interest in additional methods for regaining the visibility on the browsing habits of users. The result is a race of arms between trackers as well as privacy-aware users and browser vendors.

One novel tracking technique could be based on TLS session resumption, which allows abbreviating TLS handshakes by leveraging key material exchanged in an earlier TLS session. Thus, it introduces a possibility to link two TLS sessions. However, continuous user tracking via TLS session resumption is only possible as long as the browser is not restarted, because this clears the TLS cache. Especially mobile devices are *always on* and seldomly restarted. Finally, the feasibility of user tracking via TLS session resumption depends on the TLS configuration of both server and browser, as well as on the user's browsing behaviour. It is unknown so far whether this approach is feasible for user tracking in real-world scenarios.


Client
Server
ClientHello
ServerHello, Certificate, ...
ClientKeyExchange, ..., Finished
..., Finished
[Application Data]
Client
Server
ClientHello, SessionID
ServerHello, SessionID, ..., Finished
..., Finished
[Application Data]
Client
Server
ClientHello, SessionTicket
ServerHello, SessionTicket (empty), ..., Finished
..., Finished
[Application Data]

a) Full Connection Establishment b) Session Resumption with Session ID c) Session Resumption with Session Ticket

Figure 4.1: Handshakes in TLS 1.2 with and without session resumption, highlighting encrypted data.

To the best of our knowledge, we are the first to report on the applicability of TLS session resumption for user tracking. The main contributions of our paper are:

- We measure session resumption lifetimes of all Alexa Top Million websites and the same for 48 browsers. We assess the real-world configuration of these mechanisms and derive the possible duration of user tracking, respectively.
- We introduce the *prolongation attack* that allows to extend the tracking period beyond the session resumption lifetime.
- We analyse the impact of the prolongation attack on the resulting tracking periods and on the ratio of permanently trackable users, based on an additional DNS dataset to derive the users' browsing behaviour. Our results indicate that based on a session resumption lifetime of one day, as standard setting in many popular browsers, the average user can be tracked for up to 8 days. With a session resumption lifetime of seven days, which reflects the recommended upper limit by the draft on TLS version 1.3, even 65% of all users in our dataset could be tracked permanently by at least one website in the Alexa dataset.
- We propose countermeasures that require modifying the TLS standard and the configuration of popular browsers to impede tracking based on TLS session resumption. Most effective is to disable TLS session resumption completely.

The remainder of this paper is structured as follows: Section 4.2 describes the background on TLS session resumption. Section 4.3 reviews the privacy problems arising from the usage of TLS and Section 4.4 explains the collection of data we used in this paper. Section 4.5 summarizes our major results on tracking via TLS session resumption. Countermeasures are summarized in Section 4.6 and related work is reviewed in Section 4.7. Section 4.8 concludes the paper.

4.2 Background

In this section, we describe session identifiers (IDs) and session tickets as methods to resume a previous TLS connection for TLS up to version 1.2 [20]. Then, we regard session resumption via pre-shared keys (PSK), which is proposed in the draft of TLS 1.3 [19] and not compatible with previous resumption methods. Finally, we compare the presented mechanisms to each other.

4.2.1 Session ID Resumption

In this mechanism, the server assigns a random session ID during the initial handshake with the browser (client). Client and server store this session ID along with the session keys and connection states. To resume a session, the client sends the stored session ID with the first protocol message (ClientHello) to the server, as shown in Figure 4.1 b). If the server recognises the connection and is willing to resume the session, it replies with the same session ID to re-establish the respective session.

4.2.2 Session Ticket Resumption

This approach is defined in RFC 5077 [3] as an extension of the TLS protocol. In its initial ClientHello message, the client is required to express support for session ticket resumption, which will be acknowledged with an appropriate ServerHello response. After the key exchange between server and client, the server provides the client with an encrypted session ticket, which is transmitted outside of the TLS encrypted channel. This ticket contains the session keys and connection states, which are encrypted with the private *session ticket encryption key* (STEK) of the server. The client stores this session ticket along with the used session key and connection states.

Upon reconnection, the client includes the session ticket within the ClientHello message as shown in Figure 4.1 c). The server then decrypts the session ticket with the STEK and retrieves session key and connection state. If the server accepts the ticket, then the session can be resumed with an abbreviated handshake.

4.2.3 Session Resumption via Pre-Shared Keys

The draft of TLS 1.3 [19] replaces session IDs and session tickets with the concept of session resumption via pre-shared keys (PSK), which works as shown in Figure 4.2. After the initial handshake, the server sends a PSK identity to the client. The content of the PSK identity depends on the server and may contain a database lookup key or a self-encrypted and self-authenticated ticket. The client stores this PSK identity along with its own session keys.

In a subsequent handshake, the client provides this PSK identity within the ClientHello message to the server as seen in Figure 4.2 b) and 4.2 c). Depending on the content of the PSK identity, the server decrypts the ticket and uses the contained session keys and connection states to resume the session, or the server uses the contained lookup key to find the session keys and connection states in its own database.

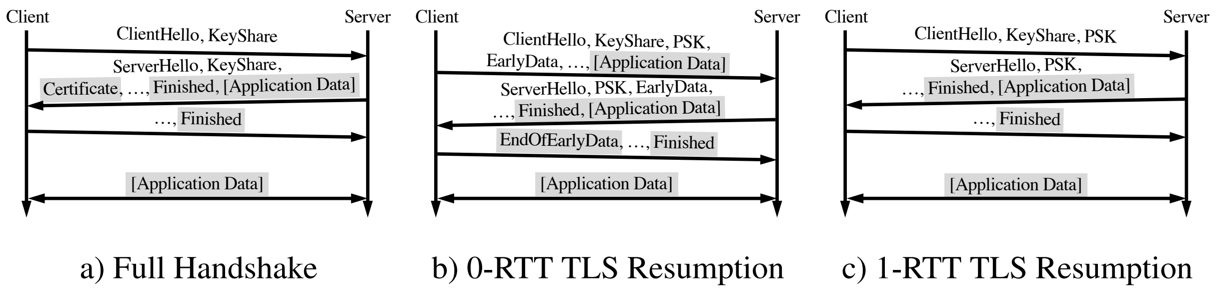

Client
Server
ClientHello, KeyShare
ServerHello, KeyShare,
Certificate, ..., Finished, [Application Data]
..., Finished
[Application Data]
a) Full Handshake
Client
Server
ClientHello, KeyShare, PSK,
EarlyData, ..., [Application Data]
ServerHello, PSK, EarlyData,
..., Finished, [Application Data]
EndOfEarlyData, ..., Finished
[Application Data]
b) 0-RTT TLS Resumption
Client
Server
ClientHello, KeyShare, PSK
ServerHello, PSK,
..., Finished, [Application Data]
..., Finished
[Application Data]
c) 1-RTT TLS Resumption

Figure 4.2: Handshakes in TLS 1.3 with and without session resumption, highlighting encrypted data.

4.2.4 Comparison of Session Resumption Mechanisms

Session resumption via PSK introduces several improvements regarding tracking. In contrast to session tickets and session IDs, the server sends the PSK identity after an initial handshake through the encrypted TLS channel. Furthermore, the server can issue multiple PSK identities at once, thus each resumption attempt uses a different PSK identity. These improvements by session resumption via PSK, protect against a correlation of single user sessions by a passive network-based observer.

RFC 5246 [20] recommends a lifetime of session IDs of less than 24 hours, the draft of TLS 1.3 [19] extends this duration to seven days. No maximum lifetime is specified for session tickets in the RFC 5077 [3]. However, servers may provide a hint to the client about their individually supported maximum lifetime. Thus, for session tickets the lifetime may exceed 24 hours.

The full handshake of TLS 1.3 (see Figure 4.2 a) requires one round trip less to complete in comparison to TLS 1.2 (see Figure 4.1 a). Thus, regarding performance, the methods of session IDs, session tickets, and 1-round-trip time (RTT) session resumption via PSK require the same number of round trips as the full handshake of TLS 1.3, while 0-RTT session resumption via PSK can save one additional round trip.

Another distinction of the discussed resumption mechanisms is their ability to provide forward secrecy. Forward secrecy describes the property of secure communication protocols, that a compromised long-term cryptographic key does not lead to a compromise of the confidentiality of past sessions. TLS 1.2 and TLS 1.3 generally support forward secrecy by using Diffie-Hellman key establishment to negotiate a temporary, symmetric session key.

From a security perspective, TLS 1.3 improves previous TLS protocol versions by supporting forward secrecy for 1-RTT session resumption (see Figure 4.2 c) through Diffie-Hellman key establishment, however it is not mandatory [19]. In the case of 0-RTT resumptions, forward secrecy cannot be realised for the first application data transmitted by the client. Thus, session resumption in TLS 1.2 and partially in TLS 1.3 reduces the communication security compared to a fresh TLS session.

Another drawback of 0-RTT resumption is that servers need to implement countermeasures against replay attacks, which TLS itself guards against for other resumption mechanisms. Thus, in TLS 1.3, session resumptions with a reduced number of round trips can only be realised at the cost of reduced security guarantees.

Table 4.1 provides a brief overview of the differences between the TLS session resumption mechanisms.

4.3 Privacy Problems with TLS Session Resumption

In this section, we describe the impact of session resumption lifetimes on users' privacy. Subsequently, we review the consequences of an unrestricted use of session resumption mechanisms with third-party online trackers.

4.3.1 Lifetime of Session Resumption Mechanisms

Always on and *always with* are characteristics of mobile devices such as smartphones and tablets that provide a ubiquitous access to the Internet and account for about half of all web browsing activities [23]. A web browser along with its TLS cache can remain active for multiple days in the background of mobile operating systems. Thus, very long session resumption lifetimes of several days or weeks are technically feasible.

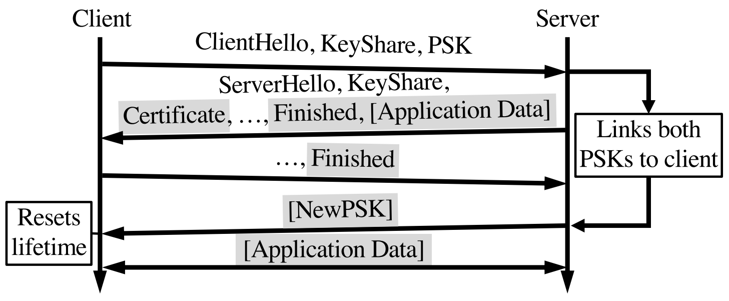

Client
Server
ClientHello, KeyShare, PSK
ServerHello, KeyShare,
Certificate, ..., Finished, [Application Data]
Links both PSKs to client
..., Finished
Resets lifetime
[NewPSK]
[Application Data]

Figure 4.3: Prolongation attack, where the client attempts a TLS 1.3 1-RTT resumption and the server falls back to a full handshake. The server can link both PSKs to the same user, while the user's resumption lifetime is prolonged with the new PSK.

Furthermore, the attempt of a client to resume a session by transmitting an identifier to the server leaks the identifier to the server regardless of whether the session is resumed or rejected (see Figure 4.3). Thus, the identifier leakage is sufficient to correlate the initial and the newly established session to the same entity.

To further extend the capability of online tracking, a website might issue a new identifier (session ID, session ticket or PSK identity) on each revisit, and thus track a user indefinitely as long as the time between two visits does not exceed the session resumption lifetime of the user's browser. We refer to this server behaviour as *prolongation attack*. The TLS standard does not define client behaviour in a way to prevent this attack. While Figure 4.3 refers to TLS 1.3 1-RTT, similar attacks apply to 0-RTT and previous TLS versions.

4.3.2 Third-Party Tracking via Session Resumption

Third-party tracking refers to a practice, where a party, other than the targeted website, can track a user's visit. It is a widespread phenomenon on the Internet with an average of 17.7 third-party

	Session ID	Session Tickets	0-RTT via PSK	1-RTT via PSK
Server stores its own secret TLS state	yes	no	optional	optional
Number of RTT compared to full handshake	-1 RTT	-1 RTT	-1 RTT	identical
Initial handshake contains unencrypted identifier	yes	yes	no	no
Identifier reuse for multiple connections	yes	yes	should not	should not
Forward secrecy	no	no	no	optional
Uses one key for sessions of multiple users	no	yes	optional	optional
Recommended limit of the resumption lifetime	24h [20]	>24h [3]	7 days [19]	7 days [19]

Table 4.1: Comparison of TLS session resumption mechanisms

trackers per website across the Alexa Top 500 categories [2]. Google with its various hostnames is present on nearly 80% of the Alexa Top Million Sites [9, 2] and thus can gain deep insights on users' browsing behaviour.

As for tracking via session resumption, third-party trackers can recognise users based on the identifier, which they present to resume a previous TLS session. Thus, the tracker can link multiple observed visits of the user across sites, where the tracker is included as a third-party. However, to distinguish the various first-party sites a user visited, the tracker requires an identifier such as a HTTP referrer or a custom URL per first-party.

4.4 Data Collection

In this section, we describe our various data sources which we use to determine the feasibility of online tracking based on TLS session resumption. For our empirical analysis, we accumulated the TLS configuration of popular online services and browsers. Furthermore, we investigate web browsing patterns of users based on a DNS traffic data set.

4.4.1 Alexa Top Million Data Set

To get an estimate on the usage of TLS session resumption in the web and to conduct a qualitative analysis of the used session resumption mechanisms we measured the HTTPS behaviour of the Alexa Top Million Sites [9]. Over a period of 31-days in March and April of 2018, we connected daily to each and every site in the Alexa Top Million on TCP port 443 using the OpenSSL Toolkit [18] and recorded the handshake behaviour.

To probe the configured session resumption lifetime for session IDs and session tickets used by the Alexa Top Hundred Thousand, we revisited each website periodically in intervals of five minutes or less, each time presenting the identical session ID or session ticket, respectively.

To measure which sites share their session ID cache or STEK with other online services, we saved the TLS connection state of a session with one site and tried to pairwise reconnect this state to the other websites in the sample. Subsequently, we created groups of sites that allow mutual session resumption. This pairwise evaluation causes quadratic cost, therefore we reduced our sample to the Alexa Top Thousand Sites.

We conducted all scans from our university campus and followed best practices of active scanning [1].

4.4.2 Browser Measurements

To assess the browser behaviour in regard to session resumption we used a sample of 48 browsers for mobile and desktop platforms as shown in Table 4.2. This sample includes the most popular browsers with respect to their global market share [22, 16, 14], which were publicly accessible for either iOS, Android, and/or desktop operating systems. Besides the most popular browsers, we included Tor Browser, Orbot, Brave, Cliqz, JonDoBrowser, and Ghostery Privacy Browser as explicitly privacy-friendly browsers to our sample.

To gather the configured maximum resumption lifetimes of each browser and for each different resumption mechanisms, we used a test website with a custom JavaScript probe. We attempted to resume a session after varying intervals of up to 24 hours since the initial handshake. On each connection attempt, the server checks and records if the initially established session resumption identifier is transmitted. We tried only one session resumption per initial handshake to avoid potential side effects like a prolongation of the browser's resumption lifetime.

To test if browsers enable third-parties to link user activities on different websites as described in Section 4.3.2, we used a testbed as illustrated in Figure 4.4. The browser consecutively retrieves the websites A and B, which include the same third-party T. We observe, whether a session resumption with T is possible from the context of different first parties A and B.

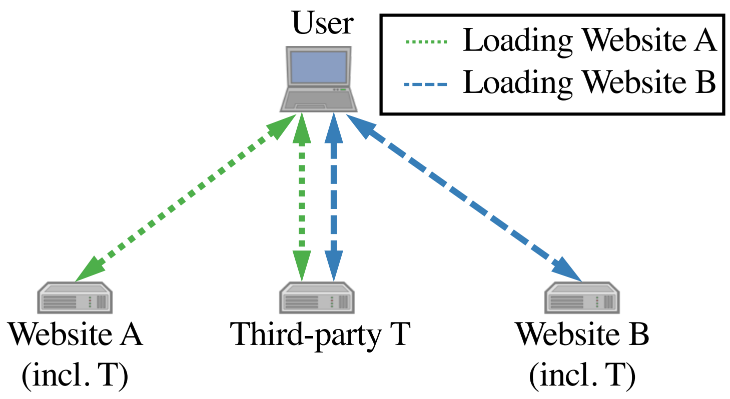


Figure 4.4: Testbed to measure browser behaviour in regard of third-party tracking.

4.4.3 DNS Data Set

To estimate the impact of the presented prolongation attack (see Section 4.3.1) on the tracking period of a user, we evaluate real-world browsing patterns. Furthermore, these browsing patterns allow us to approximate the ratio of resumed revisits for a given session resumption lifetime.

We obtained the data set used in [5, 11, 6], which contains the pseudonymized DNS traffic logs of 3862 students over a period of two months between April 30, 2010 and June 29, 2010. The data set originates from the student housing network at the University of Regensburg.

Our sample of DNS logs contains only the fraction of Internet traffic that originates from the students' unique and static IP address accessible within their room. Note that this sample might not cover the whole traffic of all users and thus our conclusions might only describe a lower boundary in regard to user tracking via TLS session resumption. A descriptive statistic of the used DNS data set can be found in [5].

We restricted our evaluations to a pseudonymized DNS data set to address ethical concerns. Besides a pseudonym for the source IP address, we also pseudonymized target hostnames on a per user basis. This approach prevents background knowledge attacks that use knowledge about the browsing history of another user within this data set.

Consequently, each record in our data set consists of a pseudonym for the source IP address, the query time, a pseudonym for the target hostname, and query type. For our analysis we considered only DNS queries of regular name resolution for IPv4 addresses as well as IPv6 addresses.

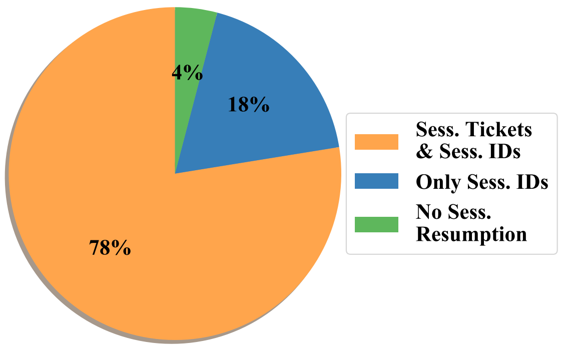


Figure 4.5: Supported session resumption mechanisms for TLS-enabled sites in Alexa Top Million on 15th April 2018.

4.5 Evaluation

To evaluate the feasibility of tracking based on TLS session resumption, we analyse in this section the configuration of servers and browsers and to which extent they restrict the session resumption mechanisms. Afterwards, we investigate real-world browsing patterns and check whether the technical restrictions of browsers are suitable to protect users' privacy against online tracking services.

4.5.1 Evaluation of Server Configurations

The feasibility of TLS session resumption as a tracking mechanism depends considerably on the configuration of the server. In this section, we investigate the adaptation of TLS session resumption mechanisms, real-world configurations for the session resumption lifetime and security-related configuration issues.

Adoption of TLS Session Resumption Mechanisms

Figure 4.5 shows the support of TLS-enabled Alexa Top Million Sites for session resumption based on IDs or tickets. This measurement from the 15th April 2018 does not include session resumption via PSK because TLS 1.3 was still a draft at that point. In total, we found 691 280 sites among the Alexa Top Million that supported TLS. 95.9% of these sites do either support session IDs or both IDs and tickets. The remaining 28 236 sites do not support TLS session resumption by, for example, providing an empty string as an ID which does not allow to resume a session.

With 536 088 websites, 77.6% of all TLS-enabled Alexa Top Million Sites do also support session tickets. Note that, if a client-server pair supports both session resumption mechanisms, then TLS session tickets will be the preferred resumption mechanism [3].

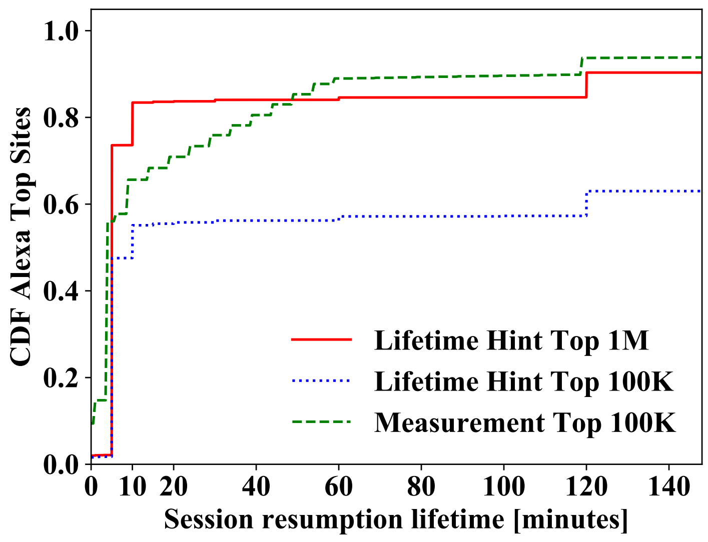


Figure 4.6: Cumulative distribution of Alexa Top Sites over short hinted and measured lifetimes of session tickets.

Lifetime of Session Resumption Mechanisms

The server can include a *lifetime hint* along with the session ticket or PSK identity. If the browser respects lifetime hints, it will only try to resume previous sessions within this hinted lifetime.

The lifetime hints of TLS session tickets for the Alexa Top Million set (solid, red line) and Alexa Top Hundred Thousand set (dotted, blue line) are shown as cumulative distribution in Figure 4.6. This plot is normalised to the total number of obtained tickets, which were 535 306 and 56 407 for the respective sets on the 24th March 2018. With 46% and 71%, a lifetime hint of five minutes is the most popular configuration within the Alexa Top Hundred Thousand and Alexa Top Million respectively.

In both sets, around 2% of all sites use a lifetime hint of zero seconds. We interpret this configuration as erroneous because this effectively prevents session resumption. Instead, server operators should deactivate the session ticket extension to save resources for both server and client if they choose not to support session tickets.

We observe, that more than 80% of TLS session ticket enabled sites within the Alexa Top Million chose lifetime hints of less than or equal to ten minutes. However, around 10% of the remaining sites use lifetime hints larger than 24 hours. Google and Facebook as market leaders in behavioural advertising show particularly large hinted session resumption lifetimes. Facebook's lifetime hint of 48 hours is above the 99.99% percentile of all session ticket hints that we collected in our scans. Google and various of its domains are configured to a ticket lifetime of 28 hours, which is above the 97, 13% percentile in the Alexa Top Million Sites (see Figure 4.7).

To validate these results about lifetime hints, we measured the maximum session resumption lifetimes for session IDs and session tickets. We limited our sample size to the Alexa Top Hundred Thousand Sites for practicality reasons. Figure 4.6 and 4.8 show these results as green, dashed line for tickets and IDs, respectively. We find, that around 10% of the sites do not allow a resumption with session tickets immediately after the initial visit, while, for session IDs these

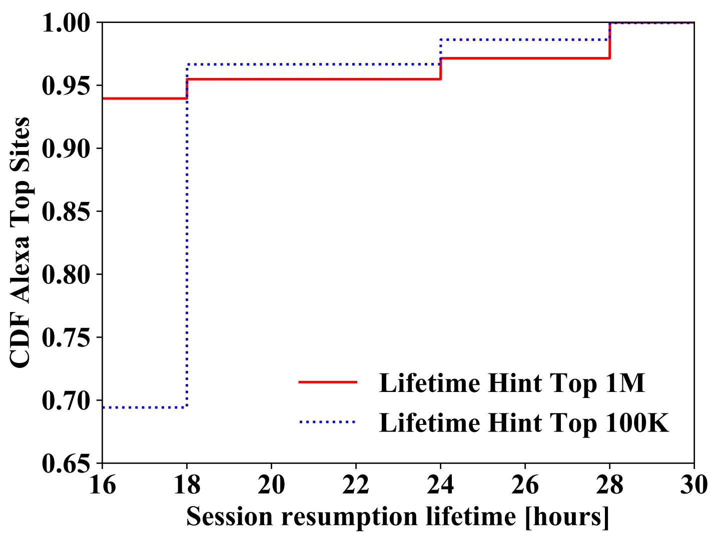


Figure 4.7: Cumulative distribution of Alexa Top Sites over long hinted lifetimes of session resumption tickets.

are 23% of the websites. We observe in both figures, that approximately 40% of the sites support maximum resumption lifetimes above five minutes. More than 20% of websites within the Alexa Top Hundred Thousand that support session IDs allow resumptions after more than two hours.

As can be seen in Figure 4.6, our measurement of the maximum resumption lifetime deviates within a range of 10 to 55 minutes from the blue, dotted plot of the corresponding lifetime hints. So far, we are not able to explain this zig-zag shape of the green plot.

Security Issues of TLS Server Configurations

In this section, we empirically measure TLS state sharing, where multiple sites share their cryptographic secrets for user sessions. Furthermore, we evaluate the lifetime of *Session Ticket Encryption Keys* (STEK). These two measurements allow us to assess the vulnerable period and the number of vulnerable websites in case of a compromise of a site's STEK.

A passive, network-based observer in possession of a compromised STEK can decrypt the initial and the resumed sessions in TLS 1.2 due to the missing forward secrecy for session resumption (see Section 4.2.4). This issue becomes more severe as STEKs are shared across multiple hostnames, which increases the number of potentially affected sessions as well as the attack surface.

We measured the shared TLS state by attempting to resume sessions of website A, where A is within Alexa Top Thousand, on every other site of the Alexa Top Thousand.

Figure 4.9 shows that 72% of the Alexa Top Thousand do either not support session resumption or do not share their TLS state with any other site of the Alexa Top Thousand. However, around 16% of websites share their TLS state with at least 25 other sites of the Alexa Top Thousand. The largest shared TLS state within the Alexa Top Thousand counts 84 sites and belongs to Google.

RFC 5077 [3] recommends a scheme for the construction of TLS session tickets, which includes an identifier for the respective STEK. With daily scans of the Alex Top Million Sites for 31-days,

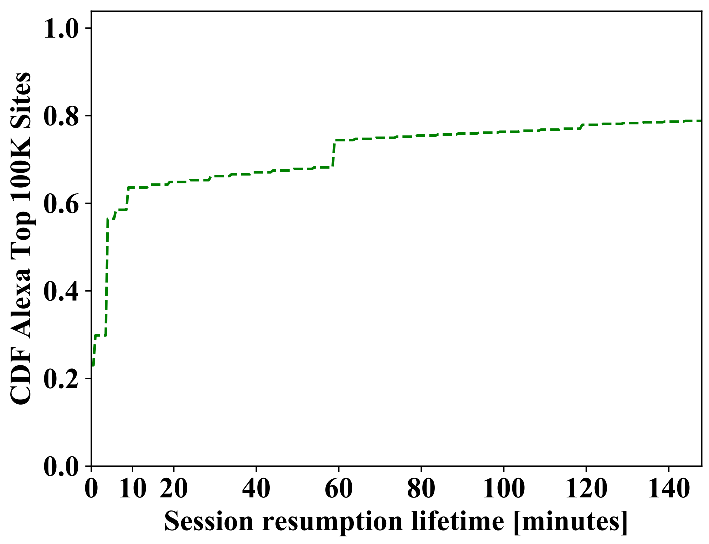


Figure 4.8: Cumulative distribution over the measured session resumption lifetime with session IDs.

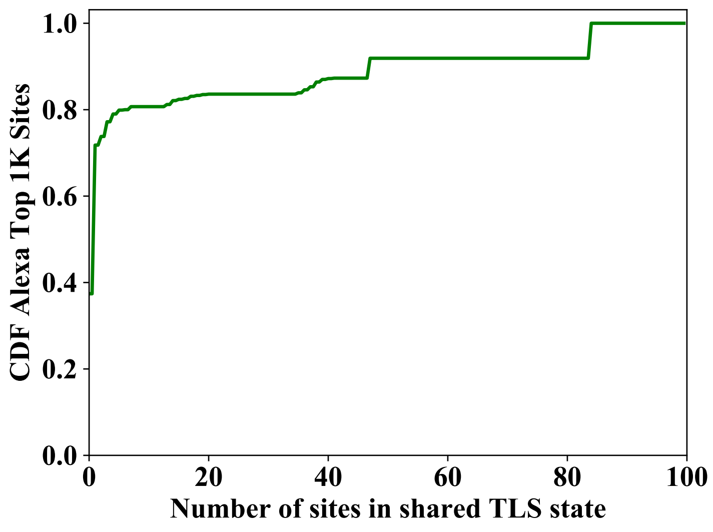


Figure 4.9: Cumulative distribution over the size of shared TLS states among sites within the Alexa Top Thousand.

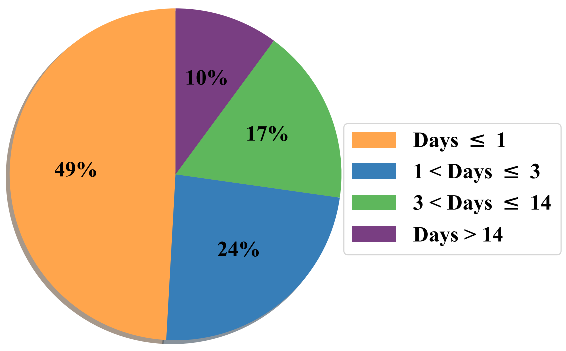


Figure 4.10: Distribution of Alexa Top Million Sites over the period for which they encrypt session tickets with the identical cryptographic key.

we obtained a sample of over 16 million session tickets. In the next step, we assigned each site their corresponding set of STEK identifiers.

Figure 4.10 shows, that 73% of the TLS session ticket enabled Alexa Top Million Sites to change their STEK within three days. However, 10% of the websites with support for session tickets did not change their STEK within two weeks. Consequently, a compromised STEK from such a website allows a network-based observer to decrypt all sessions that use session tickets with this site for at least two weeks.

Note that TLS implementations might deviate from the recommended construction scheme for session tickets [3]. Our setup expects session tickets to follow the recommended structure, deviating websites are therefore falsely classified as exchanging STEKs with every new ticket. Thus, Figure 4.10 shows only a lower boundary of websites that change their STEK less frequently than every 24 hours.

4.5.2 Evaluation of Browser Configurations

In this section, we investigate the default configuration of popular web browsers to determine the feasibility of user tracking via TLS session resumption. First, we report on the session resumption lifetime of browsers for session IDs and tickets. Then, we evaluate the capability of third-parties to track users across different first-party websites.

Lifetime of Session Resumptions Mechanisms

The results of measuring the session resumption lifetime are displayed in Table 4.2. We note that only three browsers from the privacy-friendly group do not support TLS session resumption, the remaining 45 browsers all support at least one resumption mechanisms.

We observe, that two-thirds of all tested browsers allow only for a session resumption lifetime of up to 60 minutes for both resumption mechanisms. Our measurement period is limited to 24 hours, therefore a stated one-day period might, in fact, be longer.

Some browsers such as Chrome, Opera, Firefox, and Safari were tested on multiple platforms. For these browsers, we find that the configuration of the session resumption lifetime is not consistent across the investigated platforms. For example, the desktop version of Safari is

Plt.	Browser	Lifetime for Sess. ID	Lifetime for Sess. Ticket	Third-party tracking
Desktop	360 Sec. Bro. v9.1	600 min	540 min	blocked
	Amigo v61.0	30 min	30 min	viable
	Brave	30 min	30 min	viable
	Chrome v66.0	60 min	30 min	viable
	Cliqz	1 day	10 min	viable
	Coc Coc v68.4	30 min	30 min	viable
	Comodo Dra. v63	30 min	30 min	viable
	Firefox v59.0	1 day	1 day	viable
	Internet Expl. v11	600 min	600 min	viable
	JonDoBrowser	-	-	blocked
	Konqueror v5.0	30 min	30 min	blocked
	K-Meleon v75.1	1 day	1 day	viable
	Lunascape v6.15	600 min	540 min	viable
	Maxthon v5.2	30 min	30 min	viable
	Microsoft Edge v41	600 min	600 min	blocked
	Opera v52	30 min	30 min	viable
	Pale Moon v27	1 day	10 min	viable
	QQ Browser v10	30 min	30 min	viable
	QupZilla v2.2.6	30 min	30 min	viable
	Safari v11.1	1 day	1 day	viable
	SeaMonkey v2.49	1 day	1 day	viable
	Sleipnir v6.2	30 min	30 min	blocked
	Sogou Expl. v8	30 min	30 min	viable
	SRWare Iron v65	30 min	30 min	viable
	Tor Browser	-	-	blocked
	UC Browser v7.0	30 min	30 min	viable
	Vivaldi v1.14	30 min	30 min	viable
Android	Amigo v1.10.187	60 min	30 min	viable
	Android Bro. v7.1.2	30 min	30 min	viable
	Brave v1.0.42	60 min	30 min	viable
	Cheetah Bro. v5.22	60 min	30 min	viable
	Chrome v61.0	30 min	60 min	viable
	Cliqz v1.6.2	60 min	30 min	viable
	Firefox v56.0	1 day	20 min	viable
	Ghostery Priv. v1.3	30 min	30 min	viable
	Maxthon v4.5.10	30 min	30 min	viable
	Opera Mini v30.0	15 sec	15 sec	viable
	Orbot v16.0.0	-	-	blocked
	QQ Bro. v1.2.0	1 day	1 day	viable
	Samsung Intern. v6	50 min	50 min	viable
	Sleipnir v 3.5.7	60 min	30 min	viable
	SRWare Iron v61.0	60 min	30 min	viable
	UC Browser v12.0	1 day	30 min	viable
	Yandex Bro. v18.1	50 min	30 min	viable
iOS 11	Chrome v62.0	120 min	-	viable
	Firefox v9.2	120 min	-	viable
	Opera v16.0	120 min	-	viable
	Safari v11.0	120 min	-	viable

Table 4.2: TLS session resumption configuration of browsers

configured to a session resumption lifetime of 24 hours, while the iOS version only supports resumptions for session IDs of up to 120 minutes.

Furthermore, Table 4.2 shows that the lifetime for both investigated resumption mechanisms differ for Cliqz (desktop), Pale Moon (desktop), Firefox (Android), and UC Browser (Android) by more than 23 hours. For browsers on iOS, we observe a homogeneous behaviour in our measurements, which can be explained by Apple's requirement that all browser apps in the App Store use Apple's WebKit framework as rendering engine [8].

Third-party Tracking

We analysed whether the default configuration of browsers allows resuming TLS sessions in the context of different first-party websites. Table 4.2 shows, that only the desktop browsers 360 Secure Browser, Konqueror, Microsoft Edge, and Sleipnir restrict the session resumption support for third-parties. Note, that the privacy-friendly browsers JonDoBrowser, Tor Browser, and Orbot do not support TLS session resumption mechanisms and thus also prevent third-party tracking in this regard.

Our results show, that third-party tracking via TLS session resumption is feasible for the large majority of investigated popular browsers. However, our results about the session resumption lifetime (see Section 4.5.2) indicate the session resumption lifetime is limited within the majority of investigated browsers. An explanation of our results regarding third-party tracking could be that browser vendors are mostly unaware of third-party tracking via TLS session resumption.

4.5.3 Evaluation of Real-World User Traffic

In this section, we use real-world data from a DNS traffic data set to assess the impact of different session resumption lifetimes on the achievable length of tracking periods as well as the share of permanently trackable users, respectively using the prolongation attack. Subsequently, we analyse the impact of a chosen session resumption lifetime on the expectable frequency of session resumptions for revisits.

Assumptions and Limitations

Lacking access to a comprehensive data set of actual web browsing activity, we resort to a DNS traffic data set as described in Section 4.4.3. For our evaluation, we assume that all DNS name resolution queries in the data set led to a TLS session. As discussed in Section 4.3, we reason that *always on* devices such as smartphones or tablets can achieve runtime durations for browsers and their TLS cache of several days for average users. Therefore, we assume that users would not have cleared the TLS cache of their browser within the tracking period. Note, that the following evaluations only approximate actual website visits since we cannot account for DNS caching effects.

Longest Consecutive Tracking Period

Tracking mechanisms become more capable the longer it is possible to link user behaviour to a known entity. Recall that by using the prolongation attack as described in Section 4.3.1, the period in which user activities can be linked via session resumption identifiers can be extended beyond a single session resumption lifetime. The extent to which this prolongation is possible depends on the time between consecutive visits of a website.

Definition 1

In the context of session resumption we use the term consecutive tracking period *to refer to a sequence of visits $v_1,\dots,v_n$ to an online service where no interval between two visits exceeds the given session resumption lifetime l. More formally, we define the predicate P as*

$$P(v_1,\dots,v_n,l) \Leftrightarrow |t_i - t_{i+1}| \le l, \forall i \in \{1,\dots,n-1\},$$

where t_i denotes the time of visit v_i.

Definition 2

We further define the longest consecutive tracking period $lctp$ *of a sequence of visits $v_1,\dots,v_n$ given a session resumption lifetime l as the length of the longest subsequence of visits that still fulfils the consecutive tracking period property, or more formally as*

$$lctp(v,l) \mapsto \max_{i,j\in\{1,\dots,n\},i<j} |t_i - t_j|[P(v_i,\dots,v_j,l)],$$

where t_i again denotes the time of visit v_i.

Figure 4.11 shows the longest consecutive tracking period as a function of the session resumption lifetime determined from said DNS data set with the same assumption as stated above. Each data point depicts the maximum tracking period overall websites of the respective user, or, more formally,

$$lctp_{\max}(u,l) \mapsto \max_{s\in S} lctp(v_{u,s},l),$$

where S denotes the set of all visited sites and $v_{u,s}$ the sequence of visits by user u to site s. We plotted $lctp_{\max}$ for the median user (blue, dotted line) and the average user (green, dashed line) of our data set within total 3862 users.

We observe, that the length of the longest consecutive tracking period differs widely among users. While some can be tracked with a session resumption lifetime of 12 hours throughout the whole 61-day period of data collection, the median user can only be tracked for a period of about three days.

We also note that the gradient of these plots exhibits stronger increments for session resumption lifetimes shorter than 20 minutes or longer than seven hours. It seems as if these seven hours were chosen to overcome a user's sleeping phase with reduced online activity because for such longer lifetimes, the tracking period becomes longer than one day.

Share of Permanently Trackable User

Next, we evaluate the share of those users in our DNS data set that can be monitored throughout the 61-days sample period. We uphold the assumptions as stated above.

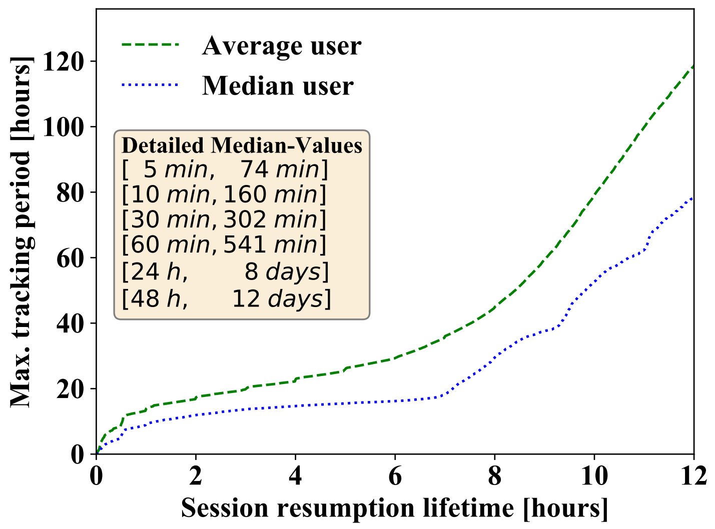


Figure 4.11: The longest consecutive tracking period per user plotted as a function of the session resumption lifetime for the prolongation attack scenario.

Definition 3

We define a user as permanently trackable by a given site, if the visits to this site fulfil the consecutive tracking period property and the time between the first (resp. last) visit and the start (resp. end) of the sample is less than or equal to the given session resumption lifetime. With the last restriction we ensure, that the user tracking can possibly continue beyond the boundaries of our sample period.

As can be seen in Figure 4.12, with a session resumption lifetime of seven days, 65% of all users within our data set can be permanently tracked by at least one website. By limiting the session resumption lifetime to 24 hours, the share of permanently trackable user declines to only 1.3%.

Ratio of Resumed Revisits

In the interest of providing empirical data to quantify performance gains from resumed session, we investigate the impact of different session resumption lifetimes on the ratio of resumed revisits. We denote revisits which happen within the session resumption lifetime of the previous visit as *resumed revisit*.

Definition 4

We define as resumption ratio *of a sequence of visits $v_1, \ldots, v_n$ by the same user to an online service given a session resumption lifetime l as*

$$rr(v,l) \mapsto \frac{|\{i \in \{1,\ldots,n-1\} \wedge |t_i - t_{i+1}| \leq l\}|}{n-1},$$

where t_i denotes the time of visit v_i.

Figure 4.13 shows the cumulative distribution of website revisits as a function of the interval between two visits by the same user.

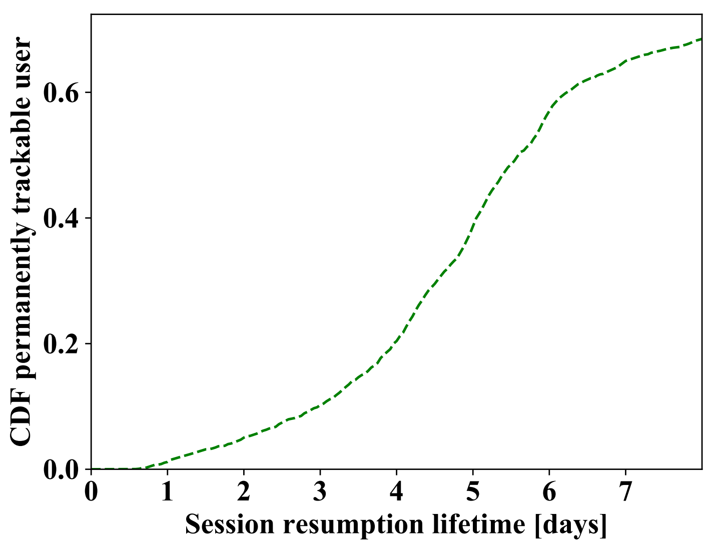


Figure 4.12: Cumulative distribution of permanently trackable user over session resumption lifetime for the prolongation attack scenario.

We observe a large share of 17.7% of all revisits can be resumed with a session resumption lifetime of five minutes. It can be further observed, that the probability of a revisit decreases continuously with the time since the last visit. Moreover, we find that about half of all revisits occur during the first hour and 95.2% within the first week.

4.6 Countermeasures

Ultimately, this leads us to a discussion of potential countermeasures. **A complete protection against tracking via TLS session resumption is achieved by deactivating this feature** as it is practised by the privacy-friendly JonDoBrowser and Tor Browser. A strict deactivation consequently excludes a browser from performance gains, such as the reduced number of round trips of TLS 1.3 0-RTT session resumption.

RFCs on session resumption mechanisms such as TLS 1.3 [19] should be extended in a way, that they exclude a possible prolongation of the lifetime of session resumption mechanisms (see Section 4.3.1). We recommend that a server-initiated renewal of a session identifier must not lead to a prolongation of client-side expiry dates. Instead, **a client must stick to the expiry date of the initial session identifier**. This protects against the prolongation attack (see Section 4.3.1).

The **recommended upper limit of the session resumption lifetime in TLS 1.3 [19] of seven days should be reduced** to hinder tracking based on this mechanism. We propose an upper lifetime limit of ten minutes based on our empirical observations. We note, that more than 80% of the Alexa Top Million Sites restrict the session resumption lifetime to less or equal to ten minutes by their own choice and 27,7% of all revisits of a site occur during this period. Furthermore, the average visit duration of popular websites is of the order of ten minutes [13], thus this lifetime limit hinders the correlation of multiple page visits by the same user.

Browser vendors should address the issue of third-party tracking via TLS session resumption, either by deactivating session resumption for third-parties or by allowing only session resumptions to third-parties if the first party site is identical.

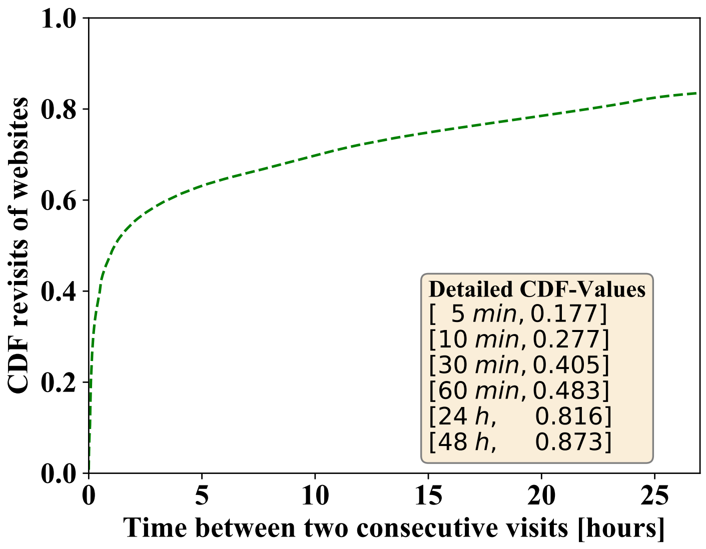


Figure 4.13: Cumulative distribution of revisits of websites plotted over the time between two visits by the same user.

Furthermore, considering that TLS 1.3 1-RTT session resumption does not lead to a reduced number of round trips compared to the full handshake, the temporal gains of this mechanism have its origin in the reduced computational complexity of the abbreviated handshake and are rather small. For a latency of 133 ms, which is in the range of 3G mobile network connections, we approximate the temporal gain of TLS 1.3 1-RTT to be in an order of 1% compared to the full handshake. Thus, **due to the low temporal gains, it seems acceptable to deactivate TLS 1.3 1-RTT session resumption** for privacy reasons.

TLS 1.3 0-RTT provides higher temporal gains by reducing the number of required round trips. However, it is not a replacement of TLS 1.3 1-RTT due to its reduced security guarantees (see Section 4.2.4). Thus, by **limiting the support for session resumption to TLS 1.3 0-RTT** the number of resumed sessions should decrease, while the temporal gains are rather high.

Finally, we reported in Section 4.5.1 that at least 10% of the Alexa Top Million Sites use their STEK for a period of at least two weeks. Since the STEK makes it possible to decrypt all sessions that use session tickets from such a site, we suggest that **a minimum STEK change rate should be standardised** to reduce this vulnerable period.

4.7 Related Work

The impact of TLS on the privacy of its users has already been a focus of previous research. Wachs et al. [24] show in their work, that the TLS *client certificate authentication* transmits unique client certificates in plaintext and thus allows a passive eavesdropper to re-identify and track users. However, websites do not widely use TLS *client certificate authentication* due to its complexity [24]. Thus, it is less feasible to be employed by online tracking service to observe users' browsing behaviour.

Empirical research by Husák et al. [7] investigated the feasibility of monitoring TLS handshakes to fingerprint and identify clients. They found, that especially the supported cipher suite lists vary among various client applications and their versions. This allows them, to infer the client application with a certain precision based on the observed cipher suit list. While this result may be beneficial to network security monitoring to detect anomalies, it is not suitable for commercial user tracking because of the few observed TLS client configurations during the handshake, do not allow to uniquely distinguish users.

Springall et al. [21] investigated security problems of TLS session resumption such as STEK lifetime, forward secrecy and TLS state sharing based on measurements of the Alexa Top Million Sites.

However, to the best of our knowledge, the feasibility of user tracking based on TLS session resumption has not been investigated so far. The privacy implications of session resumption have only been of concern to software projects. The Tor Browser disabled session resumption due to privacy considerations [17]. Moreover, the chromium project lists session resumption mechanisms as capable to allow client identification [10].

4.8 Conclusion

In this paper, we studied the feasibility of user tracking via TLS session resumption. For that, we evaluated the configuration of popular browsers and online services as well as behavioural user patterns.

Our results indicate, that most major browsers support TLS session resumption. Only three privacy-friendly browsers deactivated this feature. Almost all investigated browsers support tracking periods of at least 30 minutes based on TLS session resumption. We even observed several browsers with session resumption lifetimes of at least 24 hours, e.g., Firefox or Safari. Additionally, we investigated whether browsers protect against third-party tracking based on TLS session resumption, which allows to re-identify users across different websites in which the third-parties are embedded as well. Our results indicate that the majority of tested browsers in their standard configuration does not protect users against such third-party tracking.

In addition to the browsers, we also checked the TLS configurations of web servers delivering the Alexa Top Million Sites. We found that the majority of these websites use resumption lifetimes of up to ten minutes, which might be an indication that tracking based on TLS session resumption is not widely applied yet. However, we also observe that especially big players like Google and Facebook use exceptionally long session resumption lifetimes of 28 and 48 hours, respectively. Nevertheless, as longer session resumption lifetimes decrease the load on web servers significantly, this is not a clear indication that the tracking technique is used by them.

As the main contribution of this paper, we present a *prolongation attack* against the TLS standard, which allows to extend the tracking period beyond the session resumption lifetime. Based on a real-world data set on DNS traffic from 2010 with 3862 users we found that with a session resumption lifetime of 24 hours there is one website in the dataset that would be able to track the average user over a period of eight days. The draft of TLS 1.3 [19] proposes an upper session resumption lifetime of seven days. We analysed that such a configuration of session resumption mechanisms would make 65% of all users permanently trackable by at least one single website from our data set. However, compared to today the data set contains fewer web requests coming from mobile devices. They render tracking based on TLS session resumption easier, because of their always on-property and as they amplified the frequency users access certain (major) websites.

To mitigate the presented privacy problems, we propose countermeasures to the TLS standard and to browser vendors. The most effective technique is to disable TLS session resumption in browsers completely.

In summary, we hope that our work leads to greater awareness of the privacy risks coming from TLS session resumption and fosters further research on this topic.

4.9 References

[1] Z. Durumeric, E. Wustrow, and J A. Halderman. 2013. ZMap: Fast Internet-wide Scanning and Its Security Applications.. In *USENIX Security Symposium*, Vol. 8. 47–53.

[2] S. Englehardt and A. Narayanan. 2016. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1388–1401.

[3] P. Eronen, H. Tschofenig, H. Zhou, and J. A. Salowey. 2008. Transport Layer Security (TLS) Session Resumption without Server-Side State. RFC 5077. https://doi.org/10.17487/RFC5077

[4] A. Gómez-Boix, P. Laperdrix, and B. Baudry. 2018. Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. In *WWW 2018: The 2018 Web Conference.*

[5] D. Herrmann, C. Banse, and H. Federrath. 2013. Behavior-based tracking: Exploiting characteristic patterns in DNS traffic. *Computers & Security* 39 (2013), 17–33.

[6] D. Herrmann, M. Kirchler, J. Lindemann, and M. Kloft. 2016. Behavior-based tracking of Internet users with semi-supervised learning. In *Privacy, Security and Trust (PST), 2016 14th Annual Conference on*. IEEE, 596–599.

[7] M. Husák, M. Čermák, T. Jirsík, and P. Čeleda. 2016. HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting. *EURASIP Journal on Information Security* 2016, 1 (2016), 6.

[8] Apple Inc. 2018. App Store Review Guidelines. Retrieved May 15, 2018 from https://developer.apple.com/app-store/review/guidelines/

[9] Alexa Internet Inc. 2018. Alexa Top 1,000,000 Sites. Retrieved March 15, 2018 from http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

[10] A. Janc and M. Zalewski. 2014. Technical analysis of client identification mechanisms. Retrieved March 15, 2018 from https://www.chromium.org/Home/chromium-security/client-identification-mechanisms

[11] M. Kirchler, D. Herrmann, J. Lindemann, and M. Kloft. 2016. Tracked without a trace: linking sessions of users by unsupervised learning of patterns in their DNS traffic. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*. ACM, 23–34.

[12] P. Laperdrix, W. Rudametkin, and B. Baudry. 2016. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 878–894.

[13] SimilarWeb LTD. 2018. Top Websites Ranking. Retrieved May 5, 2018 from https://www.similarweb.com/top-websites

[14] NetMarketShare. 2018. Browser Market Share. Retrieved March 15, 2018 from https://netmarketshare.com/browser-market-share.aspx

[15] P. Papadopoulos, N. Kourtellis, and E. P. Markatos. 2018. Cookie Synchronization: Everything You Always Wanted to Know But Were Afraid to Ask. *arXiv preprint arXiv:1805.10505* (2018).

[16] J. Papenbrock. 2018. Aktuelle Marktanteile der Browser. Retrieved March 15, 2018 from https://www.browser-statistik.de/statistiken/

[17] M. Perry. 2012. Disable TLS Session resumption and Session IDs. Retrieved March 15, 2018 from https://trac.torproject.org/projects/tor/ticket/4099

[18] The OpenSSL Project. 2018. OpenSSL Cryptography and SSL/TLS Toolkit. Retrieved March 15, 2018 from https://www.openssl.org/

[19] Eric Rescorla. 2018. *The Transport Layer Security (TLS) Protocol Version 1.3*. Internet-Draft TLS13. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-tls-tls13-28 Work in Progress.

[20] E. Rescorla and T. Dierks. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246. https://doi.org/10.17487/RFC5246

[21] D. Springall, Z. Durumeric, and J A. Halderman. 2016. Measuring the security harm of TLS crypto shortcuts. In *Proceedings of the 2016 Internet Measurement Conference*. ACM, 33–47.

[22] StatCounter. 2018. Browser Market Share Worldwide. Retrieved March 15, 2018 from http://gs.statcounter.com/browser-market-share

[23] StatCounter. 2018. Desktop vs Mobile vs Tablet Market Share Worldwide. Retrieved March 15, 2018 from gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide

[24] M. Wachs, Q. Scheitle, and G. Carle. 2017. Push away your privacy: Precise user tracking based on TLS client certificate authentication. In *Network Traffic Measurement and Analysis Conference (TMA), 2017*. IEEE, 1–9.

5 Enhanced Performance for the encrypted Web through TLS Resumption across Hostnames

Summary of this publication

Citation	Erik Sy, Moritz Moennich, Tobias Mueller, Hannes Federrath, and Mathias Fischer. 2019. Enhanced Performance for the encrypted Web through TLS Resumption across Hostnames. *Preprint arXiv:1902.02531* (2019). http://arxiv.org/abs/1902.02531
Status	Unpublished
Type of paper	Research paper
Aim	This paper proposes and evaluates changes to the TLS session resumption mechanisms to achieve significant performance improvements for the TLS connection establishments during website visits.
Methodology	The used methodology comprises a performance review on the protocol flow of TLS and empirical research based on browser and web server measurements. Furthermore, performance evaluations are conducted based on simulations.
Contribution	This article describes performance limitations of the TLS 1.3 session resumptions mechanism for web browsing. To reduce the overhead of the TLS connection establishment during web browsing, the paper introduces a new TLS extension. This TLS extension enables the server to inform the client about the possibility of resuming the current connection state with hinted other hostnames. To prepare the evaluation of this proposal, this article conducts a performance measurement for full and resumed TLS handshakes. Furthermore, the paper investigates the page loading behavior of the Alexa Top Thousand websites. Note, that a successful session resumption between two hostnames requires these hostnames to share their secret TLS state. Thus, we investigated trust relations within the domain trees of each Alexa Top Thousand Site based on the presented TLS certificates and successful resumptions between nodes within the same domain tree. To evaluate the proposed TLS extension, we simulate TLS resumptions across hostnames for the Alexa Top Thousand websites. Assuming, that each trust group within a domain tree requires only one full TLS handshakes, we find that our proposal reduces the number of required full handshakes by 58.7% upon the first visit of an average website. For the computational costs of the connection establishment, this saves about 44% of the required CPU time. In total, the TLS 1.3 connection establishment for an average website is up to 30.6% faster than the current status quo.
Co-authors' contribution	The article was co-authored by Moritz Moennich, Tobias Mueller, Prof. Dr. Hannes Federrath and Prof. Dr. Mathias Fischer. Moritz Moennich assisted to measure the trust-relations and loading behavior of the

	Alexa Top Thousand sites. Tobias Mueller assisted in the measurement of the delay overhead of TLS connection establishment. Prof. Dr. Mathias Fischer assisted to revise and refine this paper. Prof. Dr. Hannes Federrath provided general feedback.

Abstract

TLS can resume previous connections via abbreviated resumption handshakes that significantly decrease the delay and save expensive cryptographic operations. For that, cryptographic TLS state from previous connections is reused. TLS version 1.3 recommends to avoid resumption handshakes, and thus the reuse of cryptographic state, when connecting to a different hostname. In this work, we reassess this recommendation, as we find that sharing cryptographic TLS state across hostnames is a common practice on the web. We propose a TLS extension that allows the server to inform the client about TLS state sharing with other hostnames. This information enables the client to efficiently resume TLS sessions across hostnames. Our evaluation indicates that our TLS extension provides huge performance gains for the web. For example, about 58.7% of the 20.24 full TLS handshakes that are required to retrieve an average website on the web can be converted to resumed connection establishments. This yields to a reduction of 44% of the CPU time consumed for TLS connection establishments. Furthermore, our TLS extension accelerates the connection establishment with an average website by up to 30.6% for TLS 1.3. Thus, our proposal significantly reduces the (energy) costs and the delay overhead in the encrypted web.

Keywords: Transport Layer Security, Secure Socket Layer, PSK Identity, Sharing of TLS State

5.1 Introduction

To communicate securely, the web increasingly adopts transport encryption via TLS [3]. Nowadays, more than 75% of all web requests are protected using TLS encryption [5]. This practice benefits communication security, but leads to additional performance penalties. Most web transactions are short transfers that are significantly delayed by the TLS connection establishment.

To accelerate the connection establishment, TLS 1.3 [12] and its predecessors provide session resumption mechanisms. They abbreviate the TLS handshake based on a shared secret exchanged during a prior TLS session between the client and server. In total, these resumption handshakes significantly reduce computational overhead for cryptographic operations and save up to one round-trip between client and server.

Load balancing can make it necessary to provide the content of a hostname via different servers. To sustain the benefits of TLS resumptions in such a setup, the respective servers can share their secret TLS state related to the client's connection. Furthermore, TLS 1.3 [12] allows resumption handshakes across hostnames when they share the same TLS certificate. Thus, the client can resume a previous connection to hostname $\mathscr{A}$ with hostname $\mathscr{B}$. However, TLS 1.3 [12] recommends not to use TLS resumption across hostnames. We find that this recommendation of TLS 1.3 leads to a significant performance limitation.

As an illustration for such a performance limitation, we assume a website served from the hostname *www.example.com*. Moreover, we assume that the hostname *example.com* is operated by the same entity and provides a redirect to *www.example.com*. A client that retrieves the website via the hostname *example.com* thus needs to establish two TLS connections. Following the specification of TLS 1.3, the client is required to establish both connections with two full handshakes. However, as both hostnames are operated by the same entity we assume that they can share their secret TLS state with each other. Thus, a performance optimized website retrieval requires only a full handshake to *example.com* and subsequently conducts a resumption handshake to connect to *www.example.com*.

In this work, we propose and evaluate a TLS extension that aims to enable the client to efficiently resume TLS connections across hostnames. In summary, this paper makes the following contributions:

- We introduce a TLS extension that enables the efficient use of resumption handshakes across different hostnames.
- We simulate the loading behavior of popular websites to assess the performance impact of our proposal. Our results indicate, that our approach can convert about 58.7% of the required full TLS handshakes to performance optimized resumption handshakes upon the first visit of an average website. This reduces the computational costs for the respective TLS handshakes by about 44% and accelerates the establishment of TLS 1.3 connections by up to 30.6%.

The remainder of this paper is structured as follows: Section 5.2 briefly introduces TLS resumption and describes the performance problems of TLS resumption per hostname that we aim to solve. Section 5.3 summarizes the proposed TLS extension. The evaluation results are presented in Section 5.4. Security and privacy considerations of the proposed TLS extension are discussed in Section 5.5, and related work is reviewed in Section 5.6. Section 5.7 concludes the paper.

5.2 Problem Statement

In this section, we review the session resumption mechanism as known from TLS 1.3 [12]. Then, we describe the performance problem of TLS resumption per hostname that we aim to solve.

5.2.1 TLS resumption

Transport Layer Security (TLS) is a cryptographic protocol that provides authentication, confidentiality, and data integrity for end-to-end communication. It finds widespread use in applications such as web browsing, email, and voice over IP (VoIP). The latest version of it is TLS 1.3 [12]. It provides the option to establish an encrypted channel with either a full or a resumed handshake. Compared to the full handshake, the resumed handshake of TLS 1.3 provides significant performance gains. The most time-consuming operations in a TLS full handshake are related to the authentication of the server's identity [15]. These operations involve the computation of a signature with the server's private key. Additionally, they require the client to verify the server's certificate, which includes to check the full certificate chain and to verify the server's signature with the public key contained in the presented certificate.

Resumed handshakes reuse the cryptographic state of a previous connection between two communication partners. In the resumption handshake, the communication partners authenticate each other by their ability to reuse the cryptographic state of a previous connection. As a result, resumption handshakes skip the computationally expensive public-key cryptography to authenticate the server's identity. Besides a lower computational overhead for resumed connections, TLS 1.3 provides also a resumed handshake that requires one round-trip time (RTT) less than the full handshake to establish the encrypted channel. As a drawback, these resumed TLS 1.3 handshakes impact the client's privacy as the shared secret can be used as a tracking mechanism [14]. Furthermore, the resumed 0-RTT handshake does not protect the server's applications against replay attacks, which TLS itself guards against for other handshake modes. For a comprehensive description of this reduced security guarantees of the 0-RTT resumed handshake and more details on TLS 1.3, we refer readers to RFC 8446 [12].

5.2.2 The limitation of TLS resumption per SNI

According to the specification of TLS 1.3 [12], a resumption handshake *should* only be used when the hostname of the server, also known as server name indication (SNI), matches the server's hostname of the original TLS session. Because of this recommendation of TLS 1.3, the use of resumption handshakes is practically restricted to revisits to servers with the same SNI. This approach provides only performance benefits during the first visit of a website if it is required to establish multiple TLS connections to the same SNI.

However, we find that the web has a complex structure where accessing a web page requires an average of 19 TCP connections to several hosts [5]. Assuming that all of these connections are TLS encrypted, this results in 19 TLS full handshakes for the first visit of an average website. TLS resumption across hostnames can potentially further reduce this large number of TLS full handshakes to realize performance improvements.

Only with respect to network latency, these TLS full handshakes can cause a significant latency overhead x during web page loading compared to the use of resumed 0-RTT handshakes. Note, that each resumed 0-RTT handshake saves one round-trip time compared to the full handshake. For the 19 connections of the average website, this reduced latency overhead can sum up to 19 RTTs when all connections are established sequentially. The lower boundary of the reduced latency overhead is one RTT if all 19 connections are established in parallel. Equation 5.1 shows these boundaries of the latency overhead x.

$$RTT \leq x \leq \sum_{i=0}^{19} RTT_i \tag{5.1}$$

The average round-trip time for mobile LTE connection in the U.S. is approx. 60 ms to reach popular online services [8]. Thus, for the considered average web page the induced latency overhead by these initial handshakes can reach up to 1.14 s for LTE connections. Note, that the round-trip time for 3G and WiFi connections are on average longer than for LTE connections, with a latency of 212 ms and 151 ms respectively [7]. Therefore, these connections types experience an even longer latency overhead for the same web page.

However, the use of resumed handshakes also significantly reduces the latency overhead with respect to the CPU time. Measurements of the CPU time for TLS 1.2 [6] indicate that a resumed handshake requires only 0.3 ms compared to a full handshake with 6.9 ms. Thus, based on this measurement the computational effort of a full handshake seem to be 23 times higher than for a resumed handshake. With respect to the latency overhead, each resumed connection saves about 6.6 ms in CPU time on the sampled test setup. For the average website, this leads to savings in the page loading time between 6.6 ms for parallel connections and 125.4 ms for 19 sequential connections.

In total, the above paragraphs highlight that the latency overhead of loading web pages can be significantly reduced if TLS full handshakes are replaced with resumed handshakes. However, the restriction of TLS 1.3 to use resumption handshakes only for revisits to the same SNI provides a practical barrier to increase the number of resumed handshakes during the first visit of a website. For example, a redirect from *https://example.com* to *https://www.example.com* requires two TLS full handshakes because these are two different SNIs. Similar, if *https://www.example.com* includes resources from *https://static.example.com* the loading of the website requires two independent TLS full handshakes. This policy of TLS 1.3 is appropriate if it is intended that a TLS resumption handshake between the different SNIs *example.com*, *www.example.com*, and *static.example.com* should not be possible. For example, if *www.example.com*, and *static.example.com* are operated by different entities that do not trust each other. However, if *www.example.com*, and *static.example.com* are operated by the same entity or by entities that trust each other, then resumption handshakes between these SNIs should be possible. Hence, this approach would use resumption handshakes with SNIs, that have not been visited previously. As a result, the latency overhead from loading a website that includes resources from many SNIs for the first time can be significantly decreased by this approach, when some of these SNIs have a trust relationship between each other.

Note, that the specification of TLS 1.3 [12] appreciates that performance optimizations are feasible if clients resume to servers with different hostnames. However, it does not provide an approximation of these performance gains nor does it describe means to implement the exploitation of these performance optimizations.

5.3 TLS Resumption across Hostnames

In this section, we propose a TLS extension that guides the use of TLS resumption across different hostnames, i.e., it allows to inform clients that a TLS connection with SNI $\mathscr{A}$ can also be resumed with SNI $\mathscr{B}$. This information can be forwarded during the TLS handshake via our proposed TLS extension with the type *resumption_across_sni*.

Clients that support the TLS extension can indicate this within their *ClientHello* message. For this purpose, they include the appropriate extension type with an empty data field to their message. Servers must only send the *resumption_across_sni* extension to clients who signaled their support for it within their *ClientHello*. In this case, the server sends a list of SNIs that support the resumption based on the TLS state of the original session. This list of SNIs is sent within the data field of the *resumption_across_sni* extension, which in turn is sent as extension data for the servers certificate but is not part of the certificate itself. Note, that this part of the server's response uses transport encryption to protect against network-based attackers.

Upon receiving the server's response, the client must verify that each SNI provided within the *resumption_across_sni* extension is also contained in the server certificate. The specification of TLS 1.3 [12] instructs that a client can only resume a session with a new SNI $\mathscr{B}$, if the server certificate presented in the original session with SNI $\mathscr{A}$ is valid for SNI $\mathscr{B}$. Thus, the client

Network	TLS 1.2		TLS 1.3		
latency	Full	Resumed	Full	1-RTT resumed	0-RTT resumed
≈0.3 ms	26.66 ms	2.69 ms	29.17 ms	6.34 ms	6.57 ms
50 ms	237.86 ms	154.20 ms	190.06 ms	160.12 ms	109.61 ms
100 ms	439.08 ms	304.50 ms	340.81 ms	310.27 ms	209.72 ms
150 ms	639.15 ms	454.621 ms	490.87 ms	460.26 ms	309.44 ms

Table 5.1: Mean duration to establish a TLS connection via different handshake modes between the client-server pair and to conduct a short data transfer.

should only use the session resumption mechanism with SNIs that can be authenticated by the private key of the server's TLS certificate.

To establish a connection to one of these SNIs, the client can directly use a resumption handshake for which it utilizes the state of the original session. To support the proposed TLS extension on the server-side, the involved SNIs are required to share cryptographic state among each other. For TLS 1.3 this depends on the construction approach of the *pre-shared keys* that are used to conduct the resumption of a prior session. The encrypted TLS connection state either directly contains a *pre-shared key* or the connection state contains a database lookup that refers to the *pre-shared key* in a database instead. In the direct case, the involved SNIs need to share a secret key that enables them to retrieve the secret server-side connection state from the *pre-shared key* provided by the client. In the database case, the different SNIs need to share their access to a common database containing their secret server-side connection states.

Note, that the proposed TLS extension can be used in combination with TLS version 1.2 and lower. Compared to TLS 1.3, these older TLS versions do not apply the concept of encrypted extensions. Hence, the server's response has to be transmitted in the unencrypted list of extensions within the respective *ServerHello* message.

5.4 Evaluation

In this section, we investigate the real-world impact of our proposed TLS extension. For that, we first study the delay overhead of different TLS handshakes to substantiate the performance benefits of TLS resumption. Subsequently, we investigate real-world websites and the number of sequential TLS connections required to load them as well as existing trust relationships in between hostnames within the respective domain trees. Finally, we simulate the performance impact of our proposed TLS extension on the page loading behavior of the investigated websites.

5.4.1 Delay overhead of TLS connection establishment

In this section, we study the delay overhead and the CPU time of different TLS handshake modes. The delay overhead is the additional time experienced by a user if a full TLS handshake instead of a resumed handshake is used. The CPU time reflects more an economic and environmental perspective, as a CPU is capable to resume more connections than conducting full TLS handshakes within the same time frame.

Evaluation setup

For the experiment, we compare the required time to download a small web page from a single host using standard TCP together with different TLS handshake versions to establish the encrypted channel. The tested TLS handshakes are full TLS 1.2, TLS 1.2 resumption via session identifier, full TLS 1.3, 1-RTT TLS 1.3 resumption, and 0-RTT TLS 1.3 resumption. We use two virtual machines, one acting as web server and the other one as client. The virtualization is realized on the same host using qemu 2.8 and libvirt 3.0.0. This test setup leads to short network latencies with an average ping of 0.3 milliseconds (ms) between the virtual machines. The host machine was equipped with an Intel Xeon E5-1660 v4 CPU with 32GB of RAM and runs Debian stretch. The client and server virtual machines were both set up with 4 GB of RAM and were running an Ubuntu 18.10, respectively. The server ran the example server program shipped with the wolfSSL library [11]. After successfully establishing a TLS connection, this program responds with a short string to the client's request. The client ran the corresponding example client program of wolfSSL that establishes a TLS session to the server, issues a request and waits for the server's response before it terminates the TLS session.

For TLS 1.3 we used the forward-secure cipher suite *TLS_AES_128_GCM_SHA256*. As this particular cipher suite is not available in TLS 1.2, we used the most similar forward-secure cipher suite *ECDHE_RSA_AES128_GCM_SHA256*. To account for skews in the measurements, we repeated the experiment 1000 times and measured the elapsed wall-clock time. We conducted our measurements with the client's network interface configured to simulate network latencies of 0.3 ms, 50 ms, 100 ms, and 150 ms with iproute2's `tc` program. We recorded and inspected the network traffic of the virtual network interface to validate a correct behavior of our evaluation setup.

Measuring the elapsed time

The delay overhead measurements are summarized in Table 5.1. We find that independent of the tested TLS version and network latency the resumed connection establishment saves at least 22.6 ms compared to the full handshakes. For TLS 1.2 and a latency of approximately 0.3 ms this reduces the overhead of the connection establishment by a factor of almost ten. The absolute reduction of the delay overhead between the full handshake and resumed connection establishment increases with an increasing network latency, except for the TLS 1.3 1-RTT resumed handshake. This behavior can be attributed to the saved round-trip times from resumed TLS 1.2 and resumed TLS 1.3 0-RTT handshakes compared to the corresponding full handshake. We observe, that a TLS 1.2 full handshake requires one RTT delay to establish the transport connection, another two RTTs to establish the encrypted channel, before the last RTT requests and retrieves the small web page. As expected, the TLS 1.3 full handshake abbreviates this TLS connection establishment by a single RTT.

Considering the number of required round-trips times for each measurement, we find that the duration of the connection establishment is shorter for a network latency of approximately 0.3 ms than for higher network latencies. For example, the full TLS 1.3 handshake has a delay overhead of one RTT for the TCP connection, one RTT to establish the encrypted channel, and another RTT to request and retrieve the small web page. Thus, the delay overhead of the data transfer is about 0.9 ms for a network latency of approximately 0.3 ms and 150 ms for a latency of 50 ms. Subtracting this delay overhead from the respective results of Table 5.1, leads to a gap of about 10 ms as shown in Equation 5.2.

$$190.06\,ms - 150\,ms - 29.17\,ms - 0.9\,ms = 9.99\,ms \tag{5.2}$$

We assume that this time gap is caused by parallel operations of both peers that occur when the latency in between them is smaller than the time required to compute the necessary cryptographic operations. Thus, with higher latencies peers will sequentially compute operations during connection establishment, while shorter network latencies induce overlapping times for cryptographic computations at client and server.

We define the reduced delay by using a 1-RTT resumed handshake or a 0-RTT resumed handshake instead of a full TLS 1.3 connection establishment as Δ_{1RTT} and Δ_{0RTT}, respectively. Based on our measurements, we find that Δ_{1RTT} is in an interval between 22.83 ms and 30.61 ms (see Equation 5.3).

$$22.83\,ms \leq \Delta_{1RTT} \leq 30.61\,ms \tag{5.3}$$

However, for Δ_{0RTT} such an interval depends on the round-trip time between the client and server, as shown in Equation 5.4. Thus, for larger network latencies the performance benefit of using a 0-RTT resumed handshake instead of a 1-RTT resumed connection establishment increases.

$$22.3\,ms + RTT \leq \Delta_{0RTT} \leq 31.43\,ms + RTT \tag{5.4}$$

For an average LTE connection in the U.S. with a network latency of 60 ms [8], we thus expect that a TLS 1.3 connection using a 0-RTT handshake instead of a full handshake reduces the delay overhead at least by 82.3 ms.

Measuring the CPU time

For this measurement, we deployed the same test setup as described in Section 5.4.1. Furthermore, we utilized the same example programs of the wolfSSL library [11] to run the server and client. However, we started these programs with the *time* program, which measures the CPU time that was consumed during the execution of another program. Thus, it provides us a metric to assess the conducted computations on the client- and server-side during the corresponding TLS connection establishment.

Table 5.2 provides the mean values of the CPU time for the different TLS versions based on 10 000 TLS connection establishments each. A full TLS handshake consumes between 7.84 ms and 9.22 ms of CPU time per peer. A resumed TLS 1.2 handshakes requires a CPU time between 0.76 ms and 1.33 ms. Resumed connection establishments with TLS 1.3 require between 2.33 ms and 2.62 ms of CPU time per peer. Note, that the measured resumed TLS 1.3 handshakes conduct a Diffie-Hellman key exchange, which establishes a forward-secure communication channel. This forward-secrecy is not provided by resumed TLS 1.2 handshakes, which leads to a smaller CPU time compared to resumed TLS 1.3 connections.

In total, our results indicate that with respect to the CPU load at least six resumed TLS 1.2 connections can be established instead of a full TLS 1.2 handshake. For TLS 1.3, around three resumed handshakes can be performed with the same CPU time as a full handshake. Thus, an increased usage of resumed handshakes on the web allows to significantly reduce the required CPU time of TLS connection establishments. Note, that savings with respect to the CPU time provide various benefits, like a reduced energy consumption of the peers and reduced hardware requirements.

Peer	TLS 1.2		TLS 1.3		
	Full	Resumed	Full	1-RTT resumed	0-RTT resumed
Server	8.02 ms	1.33 ms	7.84 ms	2.33 ms	2.62 ms
Client	8.26 ms	0.76 ms	9.22 ms	2.38 ms	2.46 ms

Table 5.2: Mean CPU time to establish a TLS connection via different handshake modes between the client-server pair and to conduct a short data transfer.

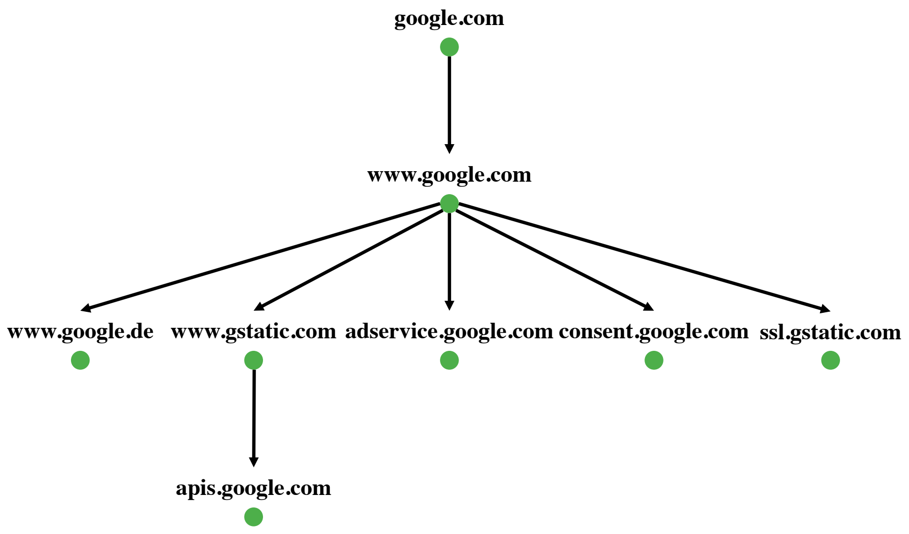


Figure 5.1: Domain tree of root-domain google.com, which holds the first rank within the Alexa Top1K Sites.

5.4.2 Investigating the loading behavior of the Alexa Top 1K Sites

Performance improvements of the proposed TLS extension over standard TLS resumption are possible if a visited hostname shares its TLS state with another hostname that the client will connect to later on. Hence, TLS resumption across hostnames can provide performance improvements within the visit of a single website and in between visits of different websites. In the following, we will investigate the benefits of our proposal with respect to a single visit of a website. For this purpose, we derived domain trees for the Alexa Top 1K Sites [1]. These domain trees indicate the causal relations between the hostnames requested during loading the respective page. Figure 5.1 shows such a domain tree for the root-domain *google.com*.

To generate domain trees, we analyzed results of the online service *urlscan.io*. It scans websites with the Google Chrome browser in headless mode and provides an overview of the browser's requests during the page loading. Furthermore, this service also reports the origin of a new request that allows us to derive the causal sequence of those requests. We conducted our scans of the Alexa Top 1K Sites on the 8th of November 2018. We successfully retrieved the domain trees for 839 websites and experienced errors for the remaining 161 domain trees, which manifests an error rate of 16.1%. Errors occurred for various reasons on protocols like IP, DNS, SPDY, and TLS. For example, due to timeouts, unreachable IP addresses, a failed domain name resolution, and protocol errors. In the following, we assume that the successfully collected website data is qualified to represent the average loading behavior of the Alexa Top 1K Sites.

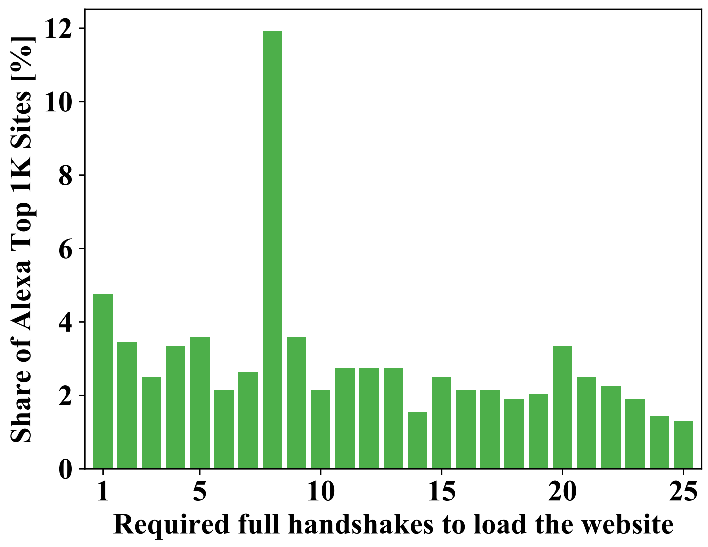


Figure 5.2: Share of Alexa Top 1K Sites over the number of required full TLS handshakes to retrieve the website. Note, that this plot is cut off at 25 full handshakes.

Based on the collected data, we find the 839 websites required connections to 17 525 different hostnames to be loaded. About 97% of these different hostnames supported TLS encryption. As our evaluation focuses on the performance improvements for TLS connections, we excluded the remaining 541 hostnames without support for TLS from our evaluation. On average, we observe that each website within the Alexa Top 1K Sites requires TLS connections to 20.24 different hostnames for its retrieval. Figure 5.2 shows the distribution of the required number of TLS connections to different hostnames for the websites in our data set. We find, that 95.2% of the investigated websites require more than a single TLS connection for its retrieval. Furthermore, we note that 73.3% of the analyzed websites require less than 26 full TLS handshake to load the site. We find that 11.9% of all observed websites require eight full TLS handshakes, which is the most common configuration. Note that also *google.com* utilizes this configuration as shown in Figure 5.1.

Using the generated domain trees, we determine the shortest path between the root-domain and all other hostnames within the domain tree. The longest determined path indicates the maximum number of sequentially established TLS connections required to load the page. For example, the domain tree of *google.com* requires four sequential TLS connection to load the website (see Figure 5.1). This longest path traverses the hostnames *google.com*, *www.google.com*, *www.gstatic.com*, and *apis.google.com*. We use the longest path of sequential TLS connections as metric to describe the impact of TLS connection establishment on the website loading performance. We assume that a website requires all its TLS connections to be established before the loading of the site can be completed. For the loading of *google.com*, this leads to four times the delay overhead of a TLS full handshakes until all required TLS connection can be established.

Figure 5.3 presents a distribution of the investigated websites over the number of sequential TLS connections required loading the site. As expected from Figure 5.2, we find that 4.8% of the investigated websites require only one sequential TLS connection to retrieve the site. 63.0% of the websites within the Alexa Top 1K Sites require less than five sequential TLS connections to load them, while no website requires more than eight sequential TLS connections.

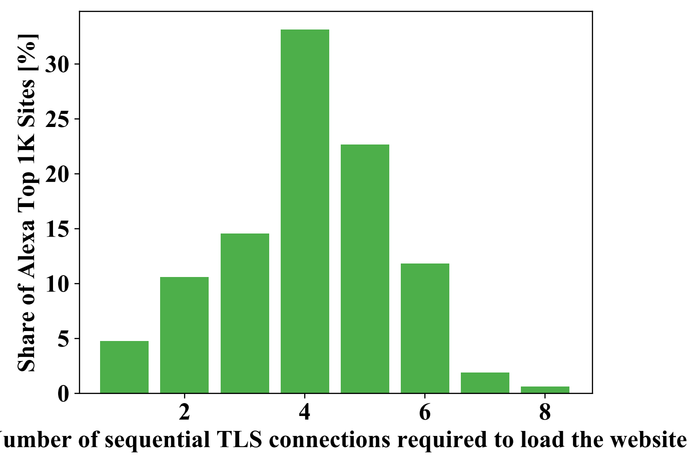


Figure 5.3: Share of Alexa Top 1K Sites is plotted over the number of required sequential TLS connections to load the respective website.

5.4.3 Measuring trust-relations within the domain trees of the Alexa Top 1K Sites

To successfully conduct TLS resumption across hostnames, we require the involved hosts to share their secret TLS state. Within this evaluation, we define the sharing of secret TLS state between different hostnames as a trust-relation. We use two approaches to assess trust relations between different hostnames. First, we assume a trust-relation exists, if a private key of a TLS certificate is valid to authenticate both hostnames. Second, if a TLS session can be successfully resumed across two hostnames, then we also assume a trust-relation between two hostnames.

To identify trust relations between hostnames based on TLS certificates, we analyze our data collection described in Section 5.4.2. The results of the Alexa Top 1K Sites include the *Subject Alternative Names (SAN)* stated in the respective TLS certificate. This SAN-list indicates the hostnames that can be authenticated via the presented certificate and thus can be used to infer trust relations.

We analyzed the SAN-lists of the 16 984 hostnames supporting TLS that we identified during the scan of the Alexa Top 1K Sites. For each domain tree, we grouped all hostnames that have a trust-relation with each other. In total, we find that on average an Alexa Top1K Site connects to 20.24 different hostnames, which form 9.49 trust groups on average. The mean size of a trust groups is 2.13 as shown in Table 5.3. This result indicates that trust relations between nodes of domain trees are a common phenomenon on the web. However, these determined trust relations do not assure that TLS resumption across different hostnames are feasible.

Results based upon TLS certificates	Results based upon TLS resumption	Union of both evaluations
2.13	1.50	2.42

Table 5.3: Mean size of TLS trust groups within the domain trees of each Alexa Top 1K Site

To further substantiate the feasibility of our proposal, we attempted to resume TLS sessions between all nodes of each domain tree. We conducted this measurement on the 25th of January 2019 using the OpenSSL library version 1.1.0f [9] to establish a TLS connection to each node

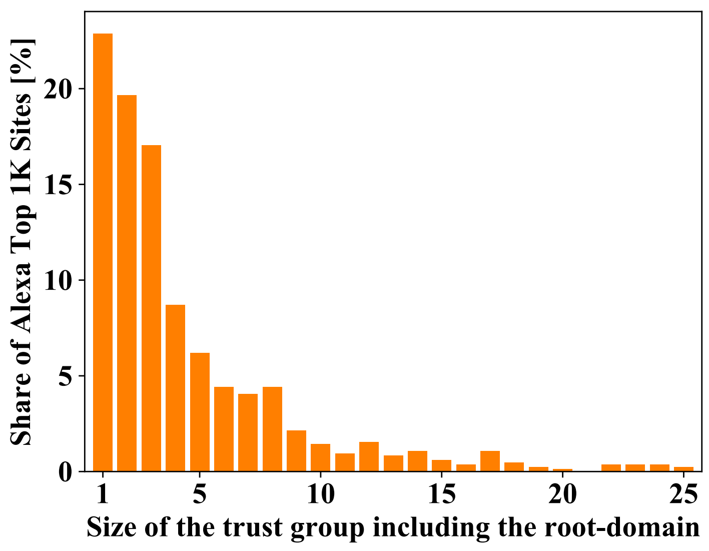


Figure 5.4: Share of Alexa Top 1K Sites is plotted over the size of the trust group that includes the root-domain. Note, that this plot is cut off at a trust group size of 25.

of a respective domain tree. For this measurement, we used the resumption mechanisms session ID's and session tickets to resume a connection via TLS versions 1.0, 1.1, and 1.2. We observed that with an exception of 54 hostnames all other hostnames preferred connections via TLS 1.2. Furthermore, our results indicate session tickets are the preferred resumption mechanism for 81.2% of the hostnames.

For this experiment, we assume that websites supporting the latest TLS version 1.3 will also support one of the earlier TLS versions. Subsequently, we attempted to resume each initial connection to the domain at each of the other hostnames within the domain trees. This approach provided us with a list of trust relations for each node of a domain tree. We evaluated these lists similar to the SAN-lists to determine trust relations within each domain tree of the Alexa Top 1K Sites.

Based upon this measurement, we find that each Alexa Top 1K Site has on average 13.51 trust groups. As shown in Table 5.3, the mean size of these trust groups is 1.50. We observe, that the size of trust groups differ between our conducted investigations. This indicates that some trust relations identified via TLS certificates do not enable the resumption of TLS session across the corresponding hostnames. Subsequently, we study these both sets of trust-relation by computing their union. We notice that also a significant share of trust relations determined via successful TLS resumptions is uncovered by the trust relations identified via TLS certificates. Considering the trust relations of both sets, we find a mean size of the trust groups per Alexa Top 1K Site of 2.42.

Figure 5.4 provides additional insights into the distribution of the group size. It plots the share of Alexa Top 1K Sites in dependence of the size of trust groups including the root-domain. We find, that 77.12% of the root-domains have a trust-relation to another hostname within their domain tree. Furthermore, 90.94% of the root-domains have a trust-relation to less than eleven hostnames within their domain tree. Popular configurations have one or two trust relations for the root-domain, which represent 19.66% and 17.04% of the Alexa Top 1K Sites respectively. Furthermore, we observe that 40 websites within the Alexa Top 1K Sites are served via their root-domain and do not require an additional TLS connection to a subdomain for their retrieval.

5.4.4 Simulating TLS resumption across hostnames

In this section, we study the real-world impact of TLS resumption across hostnames. For this purpose, we conduct a simulation based on the loading behavior of the Alexa Top 1K Sites (see Section 5.4.2) and the measured trust relations within their domain trees (see Section 5.4.3). We start by investigating the performance impact of our proposal when loading a single website. Subsequently, we extend this scenario by analyzing the visits to different websites.

Visiting a single website

Our proposal is capable to significantly accelerate the first visit of an average website. To substantiate this claim, we provide simulation results for the Alexa Top 1K Sites in this section.

Our simulation assumes the domain trees and trust groups as described in Section 5.4.2 and 5.4.3. Furthermore, we assume that a full TLS handshake to any member of a trust group enables us to subsequently establish resumed connections to all other hostnames within the same trust group. In the real world that might not be always the case, as multiple TLS connections might be established in parallel, resulting in more full handshakes.

Using the proposed TLS extension, we can reduce the number of required full handshakes to download a website by 58.75% on average. Absolute numbers are provided in Table 5.4 and indicate that our approach converts on average 11.89 full to resumed handshakes. Furthermore, Table 5.4 indicates that the traditional TLS resumption mechanism cannot reduce the number of full handshakes to different hostnames. However, the traditional TLS resumption is still beneficial when multiple TLS connections to the same hostname are required for loading the website.

Assuming that each resumed TLS connection saves approximately 6 ms of CPU time of each peer (see Table 5.2), then our proposal reduces the required CPU time to load a website on average by 71.34 ms. Assuming a full handshake to require 8 ms of CPU time per peer, then our proposal saves 44.06% of the CPU time to load an average website initially.

Without TLS resumption	With TLS resumption	With TLS resumption across hostnames
20.24	20.24	8.35

Table 5.4: Mean number of required full TLS handshakes to different hostnames to download a website of the Alexa Top 1K list for the first time. Note, that within this simulation trust relations between hostnames are assumed based on the union of presented TLS certificates and practical TLS resumptions.

Figure 5.5 shows the share of Alexa Top 1K Sites in dependence on the number of required full TLS handshakes to retrieve the website. The results of our simulation are shown as blue circles and the default website loading behavior is marked with green squares (see Figure 5.2). We find that the share of websites that require less full handshakes significantly increased when TLS

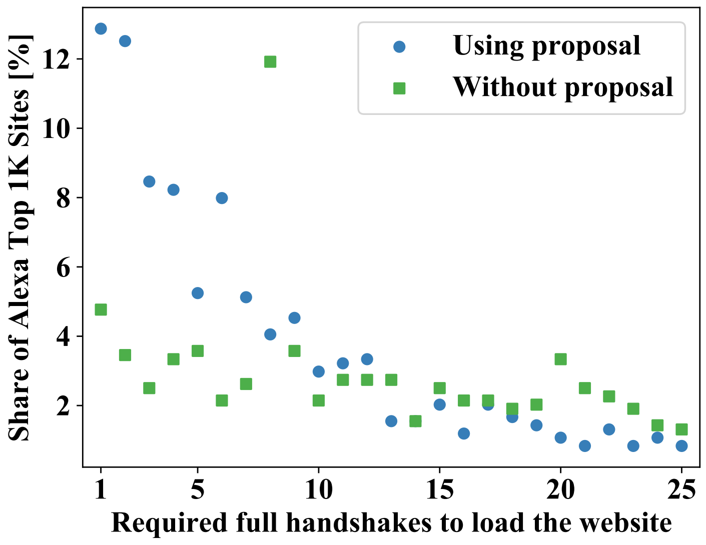


Figure 5.5: This plot shows the share of Alexa Top 1K Sites over the number of required full TLS handshakes to retrieve the website. The blue circles mark the values considering TLS resumption across hostnames, while the green squares plot the current default (see Figure 5.2). Note, that this plot is cut off at 25 full handshakes.

resumption across hostnames is used. For example, the share of Alexa Top 1K Sites requiring only one full TLS handshake is 12.87% when using TLS resumption across hostnames and 4.76% without considering our proposal. In total, the share of Alexa Top 1K Sites that require less than five full handshakes is now 42.07% compared to 14.09% without our TLS extension. Furthermore, 95.95% of the Alexa Top 1K Sites require less than 26 full TLS handshakes following our approach compared to 73.3% of these websites without enabling TLS resumption across hostnames.

In the following, we investigate the number of required sequential TLS full handshakes to retrieve an average Alexa Top 1K Site. As shown in Table 5.5, the longest path of full TLS handshakes within the domain tree of an average Alexa Top 1K Site is 4.04, independently of the support for traditional TLS resumption. Our proposal significantly reduced the number of TLS full handshakes required to retrieve an average website. The longest path of full TLS handshakes decreased to 2.46. Thus, to establish all TLS connections required to retrieve a website on average requires only 2.46 full TLS handshakes and 1.58 resumed handshake instead of 4.04 full handshakes. Equation 5.5 allows the computation of $\Delta_{connect}$, which we define as the reduced delay until all TLS connections of an average website are established using TLS resumption across hostnames instead of the current default website loading.

$$\Delta_{connect} = \begin{cases} 1.58 * \Delta_{1RTT} & \text{for 1-RTT resumptions} \\ 1.58 * \Delta_{0RTT} & \text{for 0-RTT resumptions} \end{cases} \tag{5.5}$$

With respect to relative numbers, we achieve the most significant improvement for short network latencies. To establish all connections for an average website via full TLS 1.3 handshakes requires $4.04 * 29.17\ ms = 117.85\ ms$ for a network latency of approximately 0.3 ms. TLS resumption across hostnames allows to save $1.58 * 22.83\ ms = 36.07\ ms$ in this scenario, leading to a performance gain of 30.6% until all connections are established.

Without TLS resumption	With TLS resumption	With TLS resumption across hostnames
4.04	4.04	2.46

Table 5.5: Mean length of the longest path of required full TLS handshakes to different hostnames to retrieve an average website of the Alexa Top 1K list for the first time.

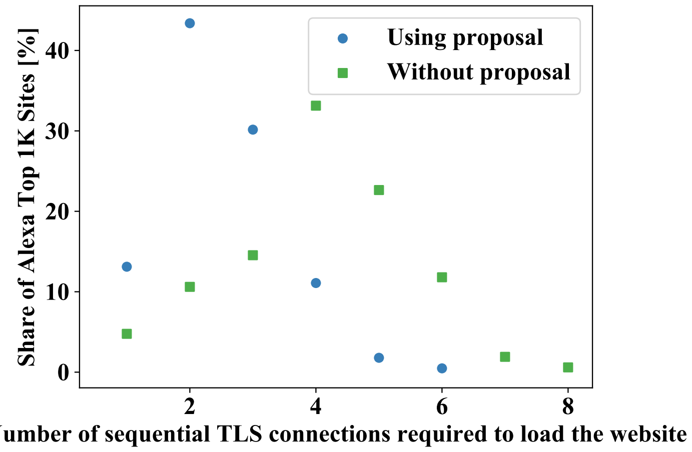


Figure 5.6: This plot shows the share of Alexa Top 1K Sites over the number of required sequential TLS connections to load the respective website. The blue circles represent the values considering TLS resumption across hostnames, while the green squares plot the current default (see Figure 5.3).

Figure 5.6 provides additional insights on the number of required sequential TLS full handshakes to retrieve an average Alexa Top 1K Site. Comparing our simulation results plotted as blue circles to the default website loading behavior represented as green squares (see Figure 5.3), we find that TLS resumption across hostnames significantly reduces the number of sequential TLS full handshakes. For example, the number of Alexa Top 1K Sites require less than three sequential TLS full handshakes for their retrieval has approximately quadrupled by using TLS resumption across hostnames. Furthermore, the share of Alexa Top 1K Sites that require more than five sequential full TLS handshakes decreased from 36.95% to 2.26%.

Visiting different websites

TLS resumptions are not just feasible within the context of a single website. For example, if different websites require connections to the same hostname, then a sequential visit to these websites makes TLS resumptions feasible. In this section, we deploy the same assumptions as in Section 5.4.4 to study visits to different websites. However, we consider only trust relations from the TLS certificates for the simulation as an experimental resumption test between all 16 984 hostnames present in the Alexa Top 1K sites would result in almost 290 million connection attempts to a rather small group of servers.

In our test scenario, we measure the average of the required number of full handshakes to different hostnames, when visiting different websites within the Alexa Top list. Table 5.6 summarizes these results for the Alexa Top 100 and the Alexa Top 1K Sites. We find, that the traditional TLS resumption converts about 2.5% of the full handshakes to resumed connection establishments in the Alexa Top 100 test scenario. In the same scenario, our proposed approach converts 64.2% of the full handshakes to resumed connection. In the larger Alexa Top 1K simulation, we find that traditional TLS resumption allows to reduce the number of full TLS handshakes by 4.1%, while our approach reduces this number by 55.5% less full TLS handshakes at the same time. These results substantiate, that TLS resumption across hostnames significantly improves performance for the encrypted web.

Alexa Top list	Without TLS resumption	With TLS resumption	With TLS resumption across hostnames
Alexa Top 100	25.98	25.34	9.30
Alexa Top 1K	41.78	40.04	18.58

Table 5.6: Mean number of required full TLS handshakes to different hostnames to download two websites of a given Alexa Top list. Note, that within this simulation trust relations between hostnames are assumed based on the presented TLS certificates.

Summary In summary, our results indicate that resumed handshakes have a significantly reduced delay and computational overhead compared to full TLS handshakes. Moreover, we find that the loading of an average website requires more than 20 TLS connections to separate hostnames, which tend to have a trust relation with each other. To reduce the overhead caused by the TLS connection establishment, TLS resumption across hostnames provides an efficient mechanisms. Thus, the average website can be retrieved with 44.06% less CPU time consumed. Furthermore, the establishment of the required connections to retrieve the website can be accelerate with up to 30.6% based on our TLS 1.3 test measurements.

5.5 Discussion

In this section, we discuss the security and privacy impact of the proposed TLS extension. Subsequently, we review developments on the Internet that may affect the impact and benefit of our proposal.

5.5.1 Security considerations

The authentication of the server's identity is a critical part of the TLS handshake. In the full TLS handshake, the client validates the server's identity based on public-key cryptography. Thus, the server is required to present a valid certificate containing a public key that confirms the claimed identity. Furthermore, the client validates that the server can generate a fresh signature with the private key corresponding to the presented certificate/public key.

Within a resumed TLS handshake, these computationally expensive public-key operations are omitted. The server is authenticated by its knowledge of a cryptographic secret related to the original TLS session, which allows the server to decrypt parts of the resumption handshake. This practice presents an indirect authentication of the server's identity because the client does not validate the server's certificate and the server's possession of the corresponding private key within the resumed TLS session. Thus, resumption handshakes require the client to trust the correctness of the server authentication during the original session.

In the scenario of TLS resumption across hostnames, the risk arises that an attacker exploits this weaker validation of the server's identity during the resumption handshake for impersonification attacks. For example, the client connects with a full TLS handshake to a legitimate but malicious SNI. The corresponding malicious server then advertises session resumption to an illegitimate SNI that would not withstand a validation of its identity by the client during a full TLS handshake. By using a resumption handshake to connect to this illegitimate SNI, the client may successfully establish a TLS connection without noticing the insufficient authentication of the server's identity.

To avoid such attacks, resumption handshakes need to be restricted to SNI values that are valid with respect to the server certificate presented in the original session. Note, that this restriction is already a requirement in the specification of TLS 1.3 [12].

5.5.2 Privacy considerations

Session resumption mechanisms in TLS 1.3 allow tracking a user across several visits to the same hostname [14]. TLS 1.3 uses unique pre-shared key (PSK) identities for session resumption that can be linked to a specific user. Thus, every time a user presents a cached PSK identity during a connection attempt, this new connection can be linked to the original connection where that unique PSK was issued by the server to the client. By enabling session resumption across several hostnames, the operators of these hostnames can link user visits to any of these hostnames. For that, the operators of the contacted hostnames need to share logs with each other. These logs contain an entry on the freshly issued PSKs to a user, details on the user activities, and if applicable information on the presented PSK by the client during the resumption handshake. Thus, by using the currently available session resumption mechanisms of TLS and when extending this across hostnames the problem of illegitimate user tracking across hostnames is facilitated.

User tracking across hostnames is a long established privacy problem on the Internet [2]. Assuming that two different websites want to track a specific user, e.g., via HTTP cookies or via TLS session resumption mechanisms, they can use a simple web link with a unique URL. If the user follows that link, then the operators of the hostnames associate the user in their logs with the observed web link. By sharing their logs and matching the included web links the operators can extract the user behavior across the respective hostnames. This example indicates that simple web links are sufficient to enable user tracking across hostnames and our proposed approach does not introduce completely new privacy problems for the user.

The average performance of TLS connection establishments depends on the ratio of resumed handshakes per full handshakes. The higher this ratio is, the lower is the overhead of the TLS connection establishment. The lifetime of the TLS session resumption mechanisms impacts this performance for a given browsing session, because a shorter lifetime leads to less resumed handshakes per full handshakes. Furthermore, this lifetime presents an upper bound for the feasible tracking period by the session resumption mechanism [14]. By using resumption handshakes across hostnames, we yield the same ratio of resumed handshakes per full handshakes within a shorter lifetime of the session resumption mechanism. For example, we consider a website that requires TLS connections to two hostnames that support session resumption between

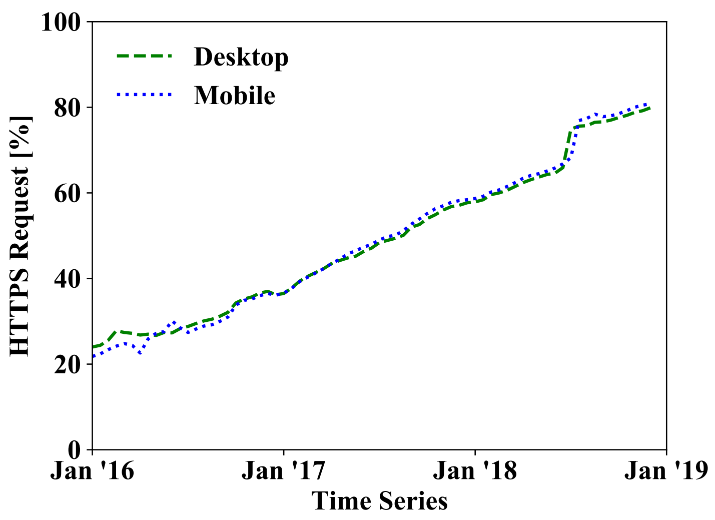


Figure 5.7: Percent of all requests indicating TLS support is plotted over a time series from January 2016 to December 2018.

each other. By using our approach, we can directly retrieve the website with a full and a resumed TLS connection leading to a ratio of one. Without the usage of resumption handshakes across hostnames, it requires two retrievals of the same website to yield the same ratio of a resumed handshake per full handshake. Thus, our approach allows setting the lifetime of the session resumption mechanism to a single website visit to achieve the same performance as the two website retrievals following the current practice of session resumption. A short lifetime of the resumption mechanism restricts the feasible tracking periods and therefore our approach can lead to improvements for the user's privacy without impacting the performance of the TLS connection establishment.

5.5.3 Chances and limitations

In this section, we review the growing TLS adoption on the web as this development positively contributes to the real-world impact of our proposed TLS extension. Subsequently, we investigate the adoption of TLS 1.3 0-RTT resumption handshake by popular websites. This handshake mode does not inherently protect against replay attacks, which restricts its use cases compared to 1-RTT resumption handshake. If the 0-RTT handshake mode exhibits a lower adoption compared to the 1-RTT resumption mode, then this practice limits the feasible performance optimizations of TLS 1.3 resumption handshakes across hostnames.

Growing TLS adoption on the web

The adoption of TLS has significantly increased during the last years [3]. To substantiate this statement, Figure 5.7 shows of the share of encrypted HTTPS requests over a three years' time series. This data is collected by regularly retrieving websites using desktop (dashed line) and mobile (dotted line) browsing environments. For further details on the methodology of these web scans, we refer the reader to [5].

Alexa Top lists	Share of websites supporting TLS 1.3		
	Full mode	1-RTT SR	0-RTT SR
Alexa Top 10	10.0%	10.0%	0.00%
Alexa Top 100	9.0%	8.0%	0.00%
Alexa Top 1K	12.7%	11.6%	0.90%
Alexa Top 10K	13.1%	12.6%	0.44%
Alexa Top 100K	9.2%	8.9%	0.19%
Alexa Top 1M	8.1%	7.1%	0.05%

Table 5.7: Websites with TLS 1.3-support in Alexa Top lists. Additionally the support of the 1-RTT and 0-RTT session resumption (SR) handshake is indicated.

As shown in Figure 5.7, the share of HTTPS request on the web increased by more than 50% during the plotted period. Research work [3] expects even further growth of the TLS deployment on the web. Prior research[14] on TLS session resumption indicates that about 96% of the websites supporting TLS do also support a session resumption mechanism. Thus, we expect that the increased deployment of TLS on the web will raise the absolute number of hostnames that support session resumption.

Measuring the adoption of TLS 1.3 handshake modes

Besides the full handshake mode, TLS 1.3 provides the option to support 1-RTT and 0-RTT resumption handshakes. We collected data on the Alexa Top Million Sites [1] to investigate the adopation of the different TLS 1.3 handshake modes. We scanned these websites on January 25, 2019 using the library boringssl [10]. For that, we first established a TLS 1.3 connection via a full TLS handshake. If this connection establishment succeeded, then we directly tried to establish connections via 1-RTT and 0-RTT resumption handshakes.

Table 5.7 summarizes our findings. 80 799 websites within the Alexa Top Million list support the TLS 1.3 full handshake. 87.6% of these websites also support session resumption via 1-RTT handshakes. However, within the Alexa Top 100K this share reaches 96.9%. Thus, our results indicate widespread support for session resumption by websites supporting TLS 1.3. Furthermore, our results indicate that 491 websites among the Alexa Top Million list support the 0-RTT handshake mode. This represents a share of 0.6% of the respective websites within the Alexa Top Million list that support TLS 1.3. However, within the Alexa Top 1K this share raises to 7.1% with nine websites out of 127 websites supporting TLS 1.3. Overall, the support of 0-RTT handshakes is within our sample lists, at least one magnitude lower compared to the support of 1-RTT resumption handshakes.

In total, we find our measurement investigates an early stage of the deployment of TLS 1.3 on the web. At the time of our scan, still many server vendors and CDNs did not support TLS 1.3 [4]. Furthermore, some server vendors and CDNs do support TLS 1.3 except for TLS 1.3 0-RTT resumption handshakes [4]. We expect that the number of websites supporting TLS 1.3 and their resumption handshakes will raise during the next years as more server vendors and CDNs will support this new TLS version by default. We assume that the lower deployment of TLS 1.3 0-RTT resumption handshakes is also caused by the lower security guarantees compared to the

1-RTT resumption case. Thus, we expect this handshake mode will remain less widely adopted compared to the 1-RTT handshake mode.

5.6 Related Work

To the best of our knowledge, we are the first to investigate the performance benefit of TLS resumption across hostnames. However, prior work [14, 13] did already report the sharing of TLS state across hostnames, which is a pre-requisite for resuming a TLS session with another hostname.

Furthermore, also RFC 8446 [12] briefly discusses the opportunity of performance enhancements based on TLS resumption across hostnames. This RFC concludes that TLS resumption across hostnames is possible within certain security limitations that lead to a high failure rate for resumption attempts. However, this RFC did not consider the option of a TLS extension to inform the client about other hostnames that are capable to resume the respective session. We argue that our proposed TLS extension minimizes this failure rate, as the client will only attempt a resumption handshake with hostnames directly recommended by the server during the original TLS session.

5.7 Conclusions

A TLS extension supporting clients to conduct TLS resumption across hostnames is overdue. Our evaluations indicate, that the proposed TLS extension yields great performance optimizations for clients and servers on the existing web without affecting the user's privacy and communication security.

5.8 References

[1] Alexa Internet Inc. 2018. Alexa Top 1,000,000 Sites. Retrieved January 25, 2019 from http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

[2] Tomasz Bujlow, Valentín Carela-Español, Josep Sole-Pareta, and Pere Barlet-Ros. 2017. A survey on web tracking: Mechanisms, implications, and defenses. *Proc. IEEE* 105, 8 (2017), 1476–1510.

[3] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. 2017. Measuring HTTPS adoption on the web. In *26th USENIX Security Symposium*. 1323–1338.

[4] Ilya Grigorik. 2019. TLS has exactly one performance problem: it is not used widely enough. Retrieved January 28, 2019 from https://istlsfastyet.com/

[5] HTTP Archive. 2018. Report: State of the Web. Retrieved January 25, 2019 from https://www.httparchive.org/reports/state-of-the-web

[6] Zi Lin. 2015. TLS Session Resumption: Full-speed and Secure. Retrieved January 17, 2019 from https://blog.cloudflare.com/tls-session-resumption-full-speed-and-secure/

[7] OpenSignal. 2014. LTE Latency: How does it compare to other technologies? Retrieved Oktober 31, 2018 from https://opensignal.com/blog/2014/03/10/lte-latency-how-does-it-compare-to-other-technologies/

[8] OpenSignal. 2018. State of Mobile Networks: USA (July 2018). Retrieved Oktober 31, 2018 from https://opensignal.com/reports/2018/07/usa/state-of-the-mobile-network

[9] OpenSSL Software Foundation. 2019. Cryptography and SSL/TLS Toolkit. Retrieved January 28, 2019 from https://www.openssl.org

[10] The Chromium Projects. 2019. BoringSSL – fork of OpenSSL that is designed to meet Google's needs. Retrieved January 28, 2019 from http://www.boringssl.com/

[11] wolfSSL Inc. 2019. Embedded TLS Library. Retrieved January 28, 2019 from https://www.wolfssl.com

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

[13] Drew Springall, Zakir Durumeric, and J. Alex Halderman. 2016. Measuring the Security Harm of TLS Crypto Shortcuts. In *Proceedings of the 2016 Internet Measurement Conference (IMC '16)*. ACM, New York, NY, USA, 33–47. https://doi.org/10.1145/2987443.2987480

[14] Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2018. Tracking Users Across the Web via TLS Session Resumption. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC '18)*. ACM, New York, NY, USA, 289–299. https://doi.org/10.1145/3274694.3274708

[15] wolfSSL Inc. 2019. Benchmarking wolfSSL and wolfCrypt. Retrieved January 16, 2019 from https://www.wolfssl.com/docs/benchmarks/

6 A QUIC Look at Web Tracking

Summary of this publication

Citation	Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2019. A QUIC Look at Web Tracking. *Proceedings on Privacy Enhancing Technologies 2019*, 3 (2019), 255 – 266. https://content.sciendo.com/view/journals/popets/2019/3/article-p5.xml
Ranking	CORE: B GGS: B Microsoft Academics: [B|A++]
Status	Published
Type of paper	Research paper
Aim	This paper studies the privacy impact of the QUIC protocol and presents countermeasures.
Methodology	Applied methods include a privacy review of the flow of the QUIC protocol and an empirical analysis based on browser and web server measurements.
Contribution	This article provides the first description of the feasibility of user tracking via the QUIC transport protocol. It finds, that user tracking can be conducted across several website visits via two independent mechanisms. The first tracking mechanisms uses QUIC's source-address token and can be exploited through online services. The second mechanism uses QUIC's server config. If an online service tracks a user via this latter mechanism, this enables also a passive observer on the network to track the user's visits to the corresponding online service. The article determines that tracking periods of at least 11 days are feasible via the reported mechanisms. As a limitation of these tracking mechanisms, they do not sustain a browser restart because this clears the cached connection data. To mitigate the impact of these significant tracking mechanisms which affect a global user base, the paper proposes several measures. For example, to counter tracking via QUIC's server config, the paper suggests logs to collect the issued server configs of online services. This practice allows identifying online services that use large numbers of server configs, which indicates potential user tracking via this mechanism. As a reaction to this suspicious behavior, the log operators warn the user about the higher risk of user tracking when visiting that specific service. To prevent tracking via source-address token, the paper proposes immediate steps based on the current design of QUIC's connection establishment such as an expiration time for these tokens. Subsequently, the article introduces changes for QUIC's handshakes, that solve the problem of tracking via the source-address token and still allow 0-RTT connection establishments.

	In total, the presented countermeasures substantiate that the feature of 0-RTT connection establishment within QUIC does not require user tracking by design.
Co-authors' contribution	The article was co-authored by Christian Burkert, Prof. Dr. Hannes Federrath and Prof. Dr. Mathias Fischer. Christian Burkert and Prof. Dr. Mathias Fischer assisted to revise and refine this paper. Prof. Dr. Hannes Federrath provided general feedback.

Abstract

QUIC has been developed by Google to improve the transport performance of HTTPS traffic. It currently accounts for approx. 7% of the global Internet traffic. In this work, we investigate the feasibility of user tracking via QUIC from the perspective of an online service. Our analysis reveals that the protocol design contains violations of privacy best practices through which a tracker can passively and uniquely identify clients across several connections. This tracking mechanisms can achieve reduced delays and bandwidth requirements compared to conventional browser fingerprinting or HTTP cookies. This allows them to be applied in resource- or time-constrained scenarios such as real-time biddings in online advertising. To validate this finding, we investigated browsers which enable QUIC by default, e.g., Google Chrome. Our results suggest that the analyzed browsers do not provide protective measures against tracking via QUIC. However, the introduced mechanisms reset during a browser restart, which clears the cached connection data and thus limits achievable tracking periods. To mitigate the identified privacy issues, we propose changes to QUIC's protocol design, the operation of QUIC-enabled web servers, and browser implementations.

Keywords: QUIC Transport Protocol, Network Security, Protocol Design, Privacy Protections, Browser Measurements

6.1 Introduction

The QUIC protocol is designed with the aim to provide privacy assurances comparable to TLS [35]. To achieve this goal, QUIC traffic is mostly encrypted with the exception of a few early handshake messages. Furthermore, QUIC switches potential client identifiers such as the *source-address token* frequently to limit the possibility for network-based attackers, e.g., internet service providers or intelligence services, to track users across several connections.

In the light of mass surveillance, network-based attackers represent a legitimate concern. However, online tracking for advertising or web analytics poses a similar threat to users' privacy. Therefore, browsers are required to protect their users' privacy against tracking through web servers that is not conducted in mutual agreement. Nonetheless, there is a trade-off between performance and privacy for browsers, in which reduced user privacy can yield a higher browser performance. For example, the usage of TLS session resumption mechanisms accelerates connection establishment, but can be used for user tracking at the same time [34]. Browsers such as Google Chrome or Opera balance this privacy versus performance trade-off by limiting the time frame of TLS session resumption [34]. QUIC allows to establish secure connections with a zero round-trip time (0-RTT) handshake. The widespread TCP/TLS 1.2 alternative requires at least three round-trips [17].

In this work we investigate the impact of QUIC's performance enhancements on user privacy. We find that QUIC's *source-address token* and QUIC's *server config* can be used by a server to identify a user across several connections. These tracking approaches exploit that a client stores data from the server for reuse in a subsequent connection. Both mechanisms allow a user identification based on the first message that a client sends to establish a connection with 0-RTT.

Compared to common online tracking practices such as HTTP cookies [10] or browser fingerprinting [1], the presented tracking mechanisms can provide performance advantages. They do not require an tracker to actively request information from a user's browser. This saves additional delays and bandwidth consumption, which otherwise restricts the practical applicability of such tracking methods [2]. For example, a third-party tracker hosting popular web fonts would directly affect the page load time of a website by performing browser fingerprinting and consequently impair the user experience [15]. Furthermore, the additional delay is a disadvantage for the highly time-constrained practice of real-time bidding in online advertising [22]. QUIC-based tracking does not come with these drawbacks.

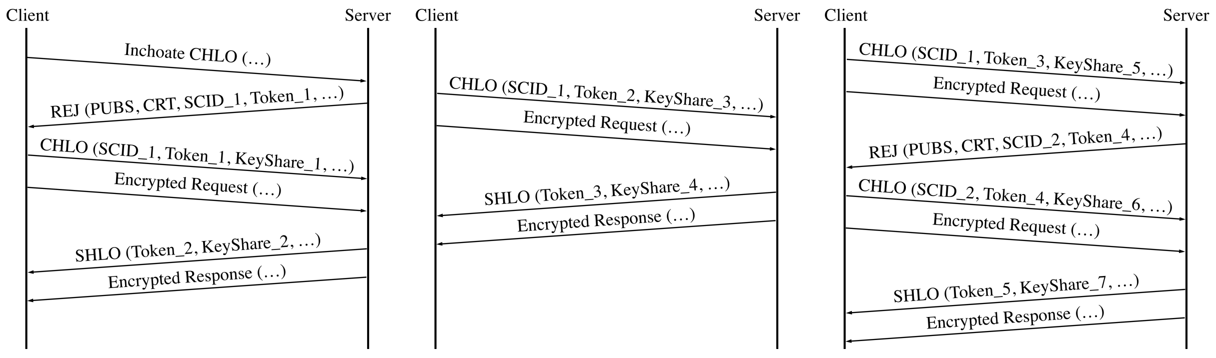
To the best of our knowledge, we are the first to report on user tracking via the QUIC protocol. The main contributions of our paper are:

- We describe tracking via QUIC's *source-address token*, which allows online services to link several website visits to the same user. Furthermore, we present tracking via QUIC's *server config*, which enables online services and also network-based attackers to track users across multiple sessions.
- We investigate the configuration of browsers that support QUIC connections by default. Our results indicate that the investigated browser configurations do not restrict the presented tracking mechanism. Furthermore, the tested browsers do not prevent third-parties from exploiting the presented mechanisms to track users across different visited websites.
- We propose countermeasures that mitigate the presented tracking mechanisms. We address tracking via the *server config* by ensuring that users attempt 0-RTT handshakes only if they observe the same *server config* during a website visit. To mitigate tracking via the *source-address token* we provide an action list for browser vendors to improve user privacy. Subsequently, we review an alternative design for QUIC's connection establishment aiming for better privacy protections.

The remainder of this paper is structured as follows: Section 6.2 describes the background on the QUIC protocol handshake. Section 6.3 reviews QUIC's tracking mechanisms and their resulting privacy problems. Section 6.4 summarizes our major results on tracking via the QUIC protocol, before we discuss the practical impact of our results and possible countermeasures in Section 6.5. Related work is reviewed in Section 6.6, and Section 6.7 concludes the paper.

6.2 Background on QUIC's Handshake

To set up a secure transport connection, the QUIC protocol incorporates some functionality of TCP, TLS, and HTTP/2. QUIC's design allows to concurrently conduct the cryptographic and the transport handshake, which provides performance improvements. In this section, we describe the cryptographic handshake of the QUIC protocol.


Client
Server
Inchoate CHLO (...)
REJ (PUBS, CRT, SCID_1, Token_1, ...)
CHLO (SCID_1, Token_1, KeyShare_1, ...)
Encrypted Request (...)
SHLO (Token_2, KeyShare_2, ...)
Encrypted Response (...)
Client
Server
CHLO (SCID_1, Token_2, KeyShare_3, ...)
Encrypted Request (...)
SHLO (Token_3, KeyShare_4, ...)
Encrypted Response (...)
Client
Server
CHLO (SCID_1, Token_3, KeyShare_5, ...)
Encrypted Request (...)
REJ (PUBS, CRT, SCID_2, Token_4, ...)
CHLO (SCID_2, Token_4, KeyShare_6, ...)
Encrypted Request (...)
SHLO (Token_5, KeyShare_7, ...)
Encrypted Response (...)

a) 1-RTT full handshake b) 0-RTT handshake (HS) c) Rejected 0-RTT HS

Figure 6.1: Handshakes in QUIC transport protocol.

QUIC provides two modes for the connection setup: The zero round-trip time (0-RTT) and the one round-trip time (1-RTT) full handshake, as shown in Figure 6.1. The full handshake is required for the initial connection between the client and the server. Subsequent handshakes can use the abbreviated 0-RTT mode utilizing cached information from previous connections between the respective client-server pair. This behavior is indicated by reusing the *server config identifier* `SCID_1` and *source-address token* `Token_2` from the connection shown in Figure 6.1 a) in the subsequent handshake (see Figure 6.1 b). The SCID describes a 16-byte identifier that allows the peers to reference a specific *server config*, whereas the token describes an encrypted and authenticated block which is opaque to the client. The token contains the client's publicly visible IP address and a timestamp as seen by the server.

The initial handshake starts with the client sending an *inchoate client hello* (CHLO) message, as shown in Figure 6.1 a). This message contains a randomly generated *connection identifier* (ConnID), which is used by both parties to refer to this connection. The ConnID is an optional part of the public header of a QUIC packet. Furthermore, the ConnID allows migrating a connection to an endpoint's new IP address and/or port. This becomes necessary after NAT rebinding [31] or when a new IP address gets assigned to an endpoint.

Next, the server responds with a *reject* (REJ) message. This message contains among others: (i) the *server config* including the server's long-term Diffie-Hellmann public value (PUBS), (ii) a certificate chain authenticating the server (CRT), (iii) a signature from the server's private key, (iv) a *server config identifier* (SCID) and (v) a *source-address token* (token). While (i)-(iii) are required to authenticate the server's identity and to establish the secure channel, the token is used to authenticate the client's publicly visible IP address. Subsequently, the client responds with a complete CHLO. Now, the client can compute the *initial keys* as a shared value of the server's PUBS and its own ephemeral Diffie-Hellman private key. The *initial keys* can be used to send encrypted requests to the server before the server responds to the complete CHLO. Note, that data encrypted with the initial keys do not provide forward-secrecy [18].

The server can compute the *final* and forward-secure key based on the client's *ephemeral Diffie-Hellman public value* (KeyShare) contained in the complete CHLO. Therefore, the *server hello* (SHLO) and all consecutive messages are forward-securely encrypted. The SHLO message contains amongst others a new token, the server's KeyShare, and the ConnID.

For a 0-RTT handshake as shown in Figure 6.1 b), the client follows the same protocol starting from the complete CHLO. Thus, the client needs to cache the *server config* received with the

REJ message and the latest received source-address token for subsequent 0-RTT connection setups.

Moreover, the server can reject the client's 0-RTT handshake attempt as shown in Figure 6.1 c). Failures like an invalid token or an expired server config presented by the client can cause the server to reject. If this occurs, the server will reply with a REJ message and both parties can proceed with the initial handshake protocol starting from its third message.

6.3 Tracking via QUIC

QUIC's novel approach to build secure 0-RTT connections uses long-term Diffie-Hellmann public values for the key exchange. This novelty as well as the mixing of cryptographic and transport-layer handshakes, make it advisable to conduct a privacy analysis of this protocol design.

In this section, we analyze which parameters of QUIC's handshake can be used by a malicious server to track a user across several connections. Subsequently, we describe the impact of QUIC's tracking duration on user privacy. Subsequently, we review the consequences of the unrestricted use of QUIC's cross-session identifiers by a third-party online tracker.

6.3.1 QUIC's identifiers suitable for user tracking

The QUIC protocol includes identifiers which are bound to a single user session such as the ConnID. However, to track users' browsing behavior over long periods, a tracker needs to link consecutive user sessions to each other. Thus, the tracker requires an identifier, which allows recognizing the user across multiple sessions.

A QUIC protocol client is required to cache the retrieved *server config* and the *source-address token* across sessions and to present them to the server when establishing a new 0-RTT connection. Therefore, it needs to be investigated, whether this cached data can be exploited by a server as a storage-based tracking mechanism that assigns unique identifiers to users.

We observe, that the *server config* contains several tag/value pairs that may be used as a hidden channel for user tracking. For example, the server can assign a distinct *server config identifier* (SCID) to each user, which provides 16-byte of entropy. A tracking server would then be able to link the connection in which a unique *server config* was initially issued to those connections where the client provides the same *server config* within its CHLO.

Note, that the tag/value pairs of the *server config* such as the SCID are transmitted in cleartext within the CHLO and the REJ message. Thus, also a passive observer of the exchanged messages can identify a user based on the used unique SCID.

The *source-address token* is a unique, authenticated-encrypted data block provided by the server, which cannot be decrypted by the client. For the purpose of IP address spoofing prevention, it contains the user's publicly visible IP address and a timestamp as seen by the server. The client caches the *source-address token* and presents it to the server during the setup of a new connection. This allows a server to link the connection where the *source-address token* is initially issued with each subsequent connections where the same token is presented during the CHLO message. Finally, this enables the server to identify a chain of connections associated with a user.

Moreover, a rejected 0-RTT handshake as shown in Figure 6.1 c) does not terminate a tracking period, because the server can link the observed *source-address token* and/or *server config* from the inchoate CHLO with the newly assigned ones in the server's REJ message.

6.3.2 Achievable tracking periods

Always on and *always with* are characteristics of mobile devices such as smartphones and tablets that provide ubiquitous access to the Internet. These mobile devices account already for about half of all web browsing activity [32]. On such devices, a web browser along with its cache can remain active for several days in the background of the operating system. Thus, also the retrieved *server configs* and tokens remain accessible until the web browser is restarted or they eventually expire.

To protect users against tracking via QUIC, it is required to restrict the maximum tracking duration independently of browser restarts. The QUIC protocol provides the *server config time to live* (STTL), which describes a hint by the server about the remaining lifetime of the provided *server config*. The STTL could constitute an upper limit to feasible tracking periods via QUIC's *server config*. This would be the case, if the client no longer used previously retrieved *server configs* or *source-address tokens* to establish connections after the STTL expired.

The protocol design must not allow a prolongation of the STTL if it shall serve as an effective upper limit for tracking periods. However, the documentation of the QUIC protocol [36] describes that server config update (SCUP) messages are used to refresh the *server config* and *source-address token*. Thus, SCUP messages can extend the period over which 0-RTT connections can be established by overwriting the cached data with the fresh *server config* and its corresponding STTL. This leads to a situation, in which every server that is revisited within the given STTL is able to extend the tracking period beyond the STTL by making use of SCUP messages. Note that the documentation of QUIC [12, 35] does not describe a maximum tracking duration regarding prolongations based on SCUP messages.

Moreover, the QUIC protocol does not include a mechanism that explicitly restricts tracking periods based on QUIC's *source-address token*. Thus, it depends on the QUIC implementation to restrict tracking via *source-address tokens*.

6.3.3 Third-party tracking

To collect comprehensive profiles about a user's browsing behavior, third-party tracking can be applied. Third-party tracking refers to a practice, where a party, other than the targeted website, can track a user's visit. It is a widespread phenomenon on the Internet with an average of 17.7 third-party trackers per website across the Alexa Top 500 categories [9]. Especially Google with its various hostnames is present on nearly 80% of the Alexa Top Million Sites [9, 3] and thus can gain deep insights into users' browsing behavior.

As for tracking via QUIC, third-party trackers can recognize users based on the token that the latter present during the 0-RTT connection attempt. Thus, a tracker can link multiple observed visits of a user across sites, where the tracker is present as a third-party. However, to distinguish the various first-party sites a user visited, the tracker requires an identifier such as an HTTP referrer or a custom URL per first-party.

Note, that the documentation of QUIC [12, 35] does not contain information on the protection of users against such third-party tracking.

6.4 Evaluation

To evaluate the feasibility of user tracking via the QUIC protocol, we investigate the default configuration of browsers supporting the QUIC protocol. Subsequently, we analyze the configuration of QUIC servers deployed by popular websites.

6.4.1 Evaluation of browser configurations

The default configuration of browsers can significantly contribute to the protection of users' privacy against online trackers. In this section, we present our measurement of browsers' QUIC configuration.

We sampled the most popular browsers as provided by [25] to investigate if they enable the QUIC protocol by default. Furthermore, we included the Chromium browser to our set of test candidates, because this relatively popular browser employs the same user-agent as the Chrome browser and would thus not be listed separately. We found that the QUIC protocol is supported by Chrome, Chromium, and Opera on desktop operating systems and by Chrome on the Android mobile operating system.

To assess the default configuration of these browsers, we intercepted and analyzed the network traffic between the browser and a provided website that supports the QUIC protocol. We captured the network traffic using Wireshark [7] which ran on the same computer as the tested browser. For the Android device running the mobile version of Chrome, we provided Internet access over a dedicated Wi-Fi, where we intercepted the network traffic on the Wi-Fi access point. To investigate the encrypted network traffic, we used the network logging systems of these browsers [28].

During our measurements, the browsers remained running in the background of their operating systems and we explicitly did not restart or update the browsers during our measurements.

Token lifetime

With our first test, we investigate the time until a browser no longer uses a previously assigned token when revisiting a website. We refer to this period as the token lifetime. To assess the token lifetime, we visit a website and validate that the connection was established with the QUIC protocol. Afterwards, we close the browser tab which we used to visit the website and leave the browser idle in the background of the operating system for a given period of time. Next, we revisit the website with the browser and analyze the network traffic to observe whether the browser transmitted an old token during the handshake. We repeated this measurement for increasingly longer periods of intermediate waiting.

Plt.	Browser	Lower boundary of token lifetime	Honors STTL	Third-party tracking
Desktop	Chrome	11 days	no	viable
	Opera	11 days	no	viable
	Chromium	11 days	no	viable
Android	Chrome	11 days	no	viable

Table 6.1: Browsers' default QUIC configuration

Our results suggest that the token lifetime is unrestricted in the evaluated browsers (see Table 6.1). However, on the basis of our measurements, we can only exclude the existence of such restrictions for the first 11 days for the investigated browsers. This duration substantiates that the investigated browsers do not restrict this tracking mechanism.

STTL adherence

Our second test examines whether browsers transmit the cached server config and the cached token after the *server config time to live* has expired. We measured this browser behavior by visiting a website that provided a *server config* with a STTL of two days. Following this, we closed all browser tabs with a connection to that site and the browser remained idle in the background of the operating system. After the STTL expired, we reconnected to the website and observed whether the browser still transmitted the expired *server config* and token during the first reconnection attempt.

As shown in Table 6.1, none of the investigated browsers stopped using the *server config* after the STTL expired. Furthermore, we found that all tested browsers continued to send their cached *source-address token* in the subsequent CHLO after the STTL expired. Therefore, the observed browser configurations indicate that not even STTLs as short as the duration of a single page visit would protect against the presented tracking mechanisms, if browsers do not enforce the STTL expiry themselves.

Third-party tracking

In the third browser measurement, we assess whether third-party tracking is feasible via the browser's QUIC configuration. For this purpose, we load a website $\mathscr{A}$ which includes a third-party $\mathscr{T}$, as it is shown in Figure 6.2. Afterwards, we close all browser tabs and wait for all QUIC connections to time out. The upper limit for an idle timeout is specified in the QUIC protocol as ten minutes for implicit connection shutdowns [36]. Next, we retrieve website $\mathscr{B}$ which includes the same third-party $\mathscr{T}$. Based on the intercepted network traffic, we analyze this second QUIC handshake with $\mathscr{T}$, which we conducted within the context of the website $\mathscr{B}$. If a token or *server config* from $\mathscr{A}$'s context has been reused to connect to $\mathscr{T}$ in the context of visiting $\mathscr{B}$, then we conclude that third-party tracking is feasible for the tested browser.

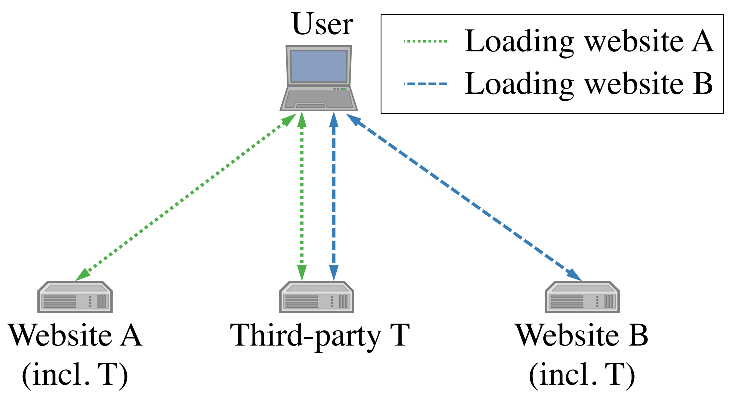


Figure 6.2: Testbed to measure browser behavior in regard to third-party tracking.

Our results indicate that all investigated browsers do not protect against third-party tracking via QUIC (see Table 6.1). This highlights the practical impact of the presented tracking approach, because third-party tracking is a wide-spread practice on the web [30].

6.4.2 Evaluation of server configurations

In the server-part of our evaluation, we analyse the degree of distribution of QUIC-enabled web sites, the configuration of QUIC's *server config time to live* among these sites, and their turnover of *server configs*.

Availability of QUIC-enabled servers

In our next experiment, we investigate the deployment of QUIC-enabled servers within the Alexa Top Million websites [3]. This provides an approximation of the total share of QUIC-enabled websites on the Internet.

We conducted this measurement on the 18th of June 2018 from the campus of the University of Hamburg. To scan the Alexa Top Million Sites, we applied a similar methodology as used in prior research work [29]. Thus, we used the tool Zmap [8] with the gQUIC-module [29] to send a *client hello* message to UDP port 443 of each Alexa-listed host. If the server responded with a valid *reject* message, we concluded that the server is supporting the QUIC protocol.

We found 186 websites within the Alexa Top Million that support the QUIC protocol. Of these sites, 141 were related to Google. The remaining 45 sites such as www.paris.fr, www.seagate.com or www.sony.jp employ a similar configuration, but we could not identify a common entity behind these websites. Table 6.2 presents the share of QUIC-enabled websites within different Alexa Top lists. With 20% and 21%, we find a significant adoption of QUIC within the Top Ten and the Top Hundred Sites, respectively. However, this share decreases for larger Top Alexa lists to only 0.0186% within the Alexa Top 1 Million. As several top ranked websites deploy the QUIC protocol, the share of the QUIC protocol on the global Internet traffic accounts for approximately seven percent [17].

The small number of websites supporting QUIC indicates that the number of online trackers potentially exploiting the presented mechanisms is still small. However, the impact of a tracking

Alexa Top list	Share with QUIC support
Alexa Top 10	20.00%
Alexa Top 100	21.00%
Alexa Top 1K	8.10%
Alexa Top 10K	1.69%
Alexa Top 100K	0.19%
Alexa Top 1M	0.02%

Table 6.2: Websites with QUIC support in Alexa Top lists

mechanism is also significantly influenced by the size of the affected user base and the length of feasible tracking periods. In 2019, around 60% of global Internet users are affected by the presented tracking mechanism, as this is the market share of the Chrome browser [33]. Thus, any entity aiming to track its users, can deploy QUIC on its servers and achieve tracking periods of longer than 10 days for a significant share of its users (see Section 6.4.1).

Our measurement investigates still an early stage of QUIC's deployment on the Internet. For example, the upcoming third version of HTTP will use QUIC instead of TCP as the transport protocol [4]. Thus, we expect that more browser vendors and websites will add support for QUIC to their products and services within the next years.

Server config time to live

To analyze the default configuration of the identified 186 QUIC-enabled websites, we used the tool quic-grabber [29] to establish QUIC connections to them. We captured the network traffic of the respective handshakes with Wireshark [7] and extracted the handshake parameters. We investigated the *server config time to live* (STTL) issued by the server, because it may be used by a QUIC client to temporally limit the usage of the presented tracking mechanisms.

We observed that servers issue the same *server config* with a decreasing STTL over time, until the server starts distributing a fresh *server config*. To account for this server behavior, we collected the *server configs* of these servers within intervals of ten minutes for a two-day period.

Our results indicate, that the domains belonging to Google deploy STTLs between 32 and 48 hours. Divergently, the other 45 domains exhibited STTLs between 4311 and 4320 hours, which are approximately 180 days. Note, that an STTL of 180 days as configured by these web servers does not contribute to the protection of user privacy, because it allows online trackers to aggregate long-term user profiles containing characteristic browsing behavior.

STTLs of 180 days indicate that a server allows for uninterrupted 0-RTT connection establishment within this period, provided that clients use the respective *server config*. Consequently, the server is required to securely store and hold the private keys corresponding to the Diffie-Hellmann public values (PUBS) that were issued as part of the respective *server config* for at least this period. Thus, from a security perspective, it seems reasonable to considerably reduce the observed STTLs of 180 days.

Server config turnover

Our measurements also indicate that the investigated web servers issue new *server configs* at least every twelve hours. This enables a server to refresh a client's *server config* at similar intervals upon a client's revisit by using SCUP messages (see Section 6.3.2). If we assume, that servers update *server config* whenever possible, then only a small fraction of website revisits would benefit from such long STTLs as 180 days compared to a two day STTL, because empirically, 87.3% of all website revisits occur within the first two days after the previous visit [34].

6.5 Discussion

In this section, we discuss the practical impact of the presented tracking mechanisms as well as protective measures against tracking via QUIC.

6.5.1 Practical impact of tracking via QUIC

To discuss its practicality, we compare tracking via QUIC to conventional browser fingerprinting, HTTP cookies and IP address-based tracking. We present use-cases in which the presented tracking approach is advantageous compared to these other popular tracking mechanisms, and discuss how the limitation of tracking interruptions through browser restarts can be overcome by using a secondary tracking mechanism as a fallback.

Comparison to other tracking mechanisms

As shown in Table 6.3, every considered tracking mechanism has its limitations, which reduces its practical applicability. Consequently, the highest long-term user identification rates can be achieved by combining different tracking mechanisms which compensate individual limitations. However, the simultaneous execution of multiple tracking mechanisms would increase client- and server-side resource consumption, both computationally and bandwidth-wise. To save resources, it seems interesting to an online tracker to prioritize tracking via QUIC over browser fingerprinting and HTTP cookies. Tracking via QUIC achieves unique user identification and is completed with the receipt of the CHLO message. It reduces the bandwidth overhead and tracking delay compared to browser fingerprinting. In contrast, browser fingerprinting negatively impacts the page load time of a website and thus might not be applicable in contexts where, for example, a third-party tracker hosts web fonts required for the rendering of the website. Tracking with QUIC can be applied even under such constrained circumstances.

Compared to HTTP cookies, QUIC-based tracking reduces the tracking delay in two scenarios: Either the encrypted request that includes the client's HTTP cookie is sent over a forward-secure connection or the server rejects the encrypted client request. In both scenarios, the server can track the user via QUIC one round trip before the server can decrypt the client's request to obtain the included HTTP cookie. This makes user identification via QUIC favorable in highly time-constrained scenarios such as real-time biddings for online advertising [37].

Just like QUIC-based tracking, IP address-based tracking is fully passive and does not introduce additional computational or communication overhead. However, IPv4 addresses are often shared

Properties	Browser Fingerprinting	HTTP Cookies	IP Address Tracking	Tracking via QUIC
Tracking delay	additional round-trip time after connection setup & processing time	requires completed connection setup	none	none
Bandwidth requirements	additional requests & response	low	low	low
Accuracy	limited [19]	unique	limited	unique
Coverage	HTTP connections (restrictions apply [24])	HTTP connections (restrictions apply [23])	IP connections	QUIC connections
Survives a change of IP Address	yes	yes	no	yes
Survives a browser restart	yes	yes (restrictions apply [14])	yes	no

Table 6.3: Comparison of tracking via QUIC to widely used tracking mechanisms.

nowadays [5], which renders this approach less accurate than tracking via QUIC with respect to unique user identification.

Limitation of QUIC-based tracking

Continuous tracking via QUIC is only possible as long as the browser is not restarted, because this clears the cached state of prior QUIC connections. We note that mobile devices are *always on* and rarely restarted. Furthermore, desktop operating systems provide a *sleep* mode which presumably reduces the occasions to restart the OS. Thus, the OS provides only limited restrictions to feasible tracking periods. Moreover, the avoidance of browser restarts allows the user to easily continue a prior browsing session as respective browser tabs are still available. Thus, we assume that incentives exist for the user to seldom restart a browser. From our perspective, tracking periods of several days or even weeks are feasible under real-world circumstances for a significant part of users.

The QUIC-based tracking mechanism is further limited by the ability of active network-based attackers to alter the *server config* and *source-address token*. These identifiers are transmitted in cleartext over the network within the CHLO and REJ messages. In the following, we describe the possible impact of such malicious behavior on user identification.

The server's REJ message and the client's CHLO message allow a network-based attacker to retrieve the *source-address token* and the *server config* which the server assigned to a user. Subsequently, the network-based attacker can establish own connections to the server, which make use of the observed *source-address token* and *server config*. Thereby, the network-based attacker would impersonate the user from the perspective of a tracking server.

Furthermore, the attacker can manipulate a server's REJ message and insert a different *source-address token* and/or *server config*. This causes the client to use this inserted data during the subsequent connection setup. However, a connection between the client-server pair can only be successfully established if the server is able to cryptographically validate the provided *source-address token* and *server config*. In exploiting this behavior, an attacker can trick a user to impersonate another user by inserting the latter's *source-address token* and/or *server config* into the former's server response.

As investigated by Lychev et al. [21], the manipulation of the user's *source-address token* or *server config* within the CHLO message will lead to failures within the connection establishment, because they are used as input into the encryption key derivation process. Therefore, the client and server will compute different encryption keys for the connection in such an event, which leads to a communication failure.

As a consequence, a network-based attacker can—to a limited extent—manipulate the *source-address token* and the *server config* that a client uses for the subsequent connection establishment. Hence, a user identification based on the presented tracking mechanisms can be misled by a network-based attacker.

6.5.2 Countermeasures

A simple and effective countermeasure against user tracking via the QUIC protocol is to establish every new connection with a full handshake. However, this prevents 0-RTT connection establishment and thus leads to performance losses during website revisits. In the following, Section 6.5.2 presents measures to protect users against tracking via the *server config*. To mitigate the privacy issues originating from *source-address tokens*, we propose directly applicable measures for browser vendors in Section 6.5.2. Subsequently, we review the design of a privacy-friendly connection establishment in Section 6.5.2.

Protection against tracking via the *server config*

This privacy problem is inherent to QUIC's handshake design which relies on previously shared PUBS for connection establishment. To solve this problem, we propose a mechanism to monitor issued *server configs*. Note, that this proposal is similar to the established certificate transparency [20] where issued TLS certificates are collected in logs. To gather data, every user can submit an observed *server config* to such a log. Alternatively, the server operator can submit the issued *server config* to a log and provide the user with a cryptographic signature from the log operator to proof this transaction. Note, that the latter approach is similar to the *Signed Certificate Timestamp* [20] extension of TLS. Such a log allows to discover server operators that issue a large number of unique *server configs* within a given period, which presents an indication for user tracking via the *server config*.

If log operators suspect a website to conduct tracking via the *server config*, they should recommend users to not reuse the *server config* for new connections and to conduct a full QUIC handshake instead. If the server submitted the respective *server config* to the log, then the log operator should sign the request with a dedicated private key to mark that *server config* as one that is suspicious to be used for user tracking.

This proposal mitigates tracking via *server configs* on a large scale, if the log operators allow only a single *server config* to be used within a specific period and mark all other *server configs* in the same period as suspicious.

Quick measures to protect against tracking via QUIC's token

In this section, we present steps that can be applied immediately by browser vendors and the QUIC working group to mitigate the privacy impact of the QUIC protocol.

As a first mitigation measure, **browser vendors should implement an expiry time for tokens**. The choice of an appropriate expiry time presents a trade-off between performance and privacy. A study of this trade-off based on real-world DNS traffic recommends limiting the expiry time to ten minutes [34]. Note, that on average, 27.7% of revisits to websites occur within this period. However, an expiry time of 60 minutes, allows in average 48.3% of website revisits to use 0-RTT handshakes, but a web tracker can observe a greater share of users' online activities in return. Furthermore, the results of the study suggest [34] that an expiry time longer than 24 hours does not significantly contribute to a higher share of 0-RTT handshakes. Thus, setting the expiry time to 48 hours instead of 24 hours increases the share of abbreviated handshakes only by 5.7 percentage points.

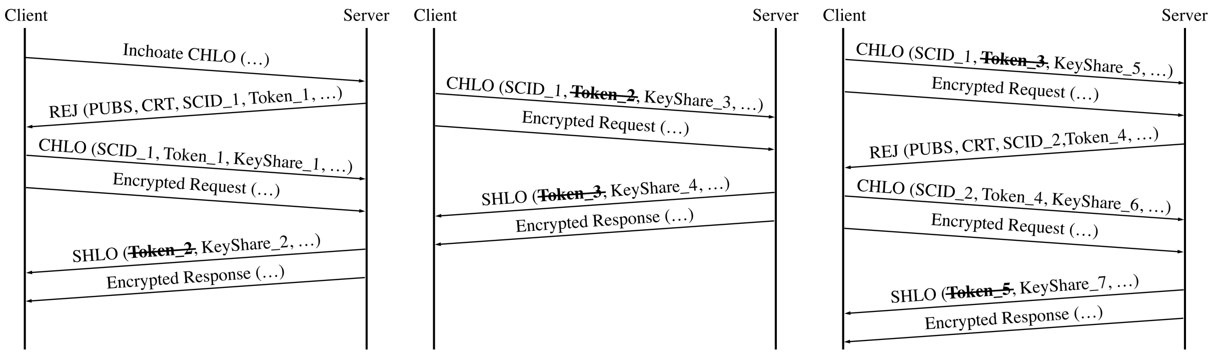
Furthermore, **a *server config update* message should not prolongate the feasible tracking period via tokens**. Instead, the expiry of the first token received from a web server should lead to a full handshake within the subsequent connection establishment, regardless of later expiry dates of tokens that were part of SHLO messages in the meantime.

The STTL hints on how long the server intends to support connection establishments for a given *server config*. Depending on the server implementation, it becomes more likely that a server rejects 0-RTT connection attempts after a *server config* expired. Anticipating this server behavior, it is beneficial from a privacy perspective to **conduct a full handshake after the respective cached *server config* has expired**.

A QUIC server uses the token to validate a client's IP address. Therefore, if a client changed the IP address after receiving a token, the server is more likely to respond with a REJ message during the next 0-RTT connection attempt. From a privacy perspective, the change of an IP address provides the opportunity to disassociate past activities from future ones, therefore breaking linkability at network-level. Thus, it seems reasonable to **establish new connections with a full handshake whenever a client's IP address changed**, to also disassociate activities in QUIC. Note that QUIC allows to explicitly migrate existing connections across different IP addresses. In this case, the server is already in a position to link both IP addresses to the same user based on the migration process. Thus, our proposal excludes this use case of connection migration.

To address the issue of third-party tracking via QUIC, browser vendors should restrict the access to cached *source-address tokens*. **Cached tokens of third-parties should only be used within the context of the same visited website**.

Privacy-friendly connection establishment to protect against tracking via QUIC's token


Client
Server
Inchoate CHLO (...)
REJ (PUBS, CRT, SCID_1, Token_1, ...)
CHLO (SCID_1, Token_1, KeyShare_1, ...)
Encrypted Request (...)
SHLO (~~Token_2~~, KeyShare_2, ...)
Encrypted Response (...)
Client
Server
CHLO (SCID_1, ~~Token_2~~, KeyShare_3, ...)
Encrypted Request (...)
SHLO (~~Token_3~~, KeyShare_4, ...)
Encrypted Response (...)
Client
Server
CHLO (SCID_1, ~~Token_3~~, KeyShare_5, ...)
Encrypted Request (...)
REJ (PUBS, CRT, SCID_2,Token_4, ...)
CHLO (SCID_2, Token_4, KeyShare_6, ...)
Encrypted Request (...)
SHLO (~~Token_5~~, KeyShare_7, ...)
Encrypted Response (...)

a) Full handshake (HS) b) 0-RTT handshake c) Rejected 0-RTT HS

Figure 6.3: Proposal for handshakes in the relaxed source-address validation scenario with no longer required tokens marked bold and struck through.

We propose an alternative handshake design, that requires clients to only use tokens within their CHLO message if this message is a direct response to a server's REJ message. Figure 6.3 provides a sketch of these handshakes, where no longer required tokens are marked in bold and struck through.

This proposal applies strict source-address validation during the full and the rejected 0-RTT handshake, as shown in Figure 6.3 a) and c), respectively. However, the 0-RTT connection establishment does not conduct a validation of the source-address (see Figure 6.3 b). To mitigate excessive IP address spoofing, this design requires QUIC servers to monitor the number of unrequited connections. If the number of unrequited connections exceeds a specified threshold, the server falls back to an operation mode that rejects all 0-RTT connection handshakes (see Figure 6.3 c). Otherwise, the server allows 0-RTT handshakes without requiring a token.

By logical conclusion, this design is privacy-friendly because source-address tokens are not reused across different connections. Furthermore, it enables 0-RTT connection establishments whenever the number of unrequited connections are below the specified threshold. The definition of the threshold itself presents a trade-off between the protection against IP address spoofing and the performance of the connection establishment.

The original QUIC connection establishment requires the first visit of a website to be a 1-RTT full handshake because the client is required to retrieve a corresponding token, as shown in Figure 6.1. Assuming that a client is in possession of a corresponding *server config*, then our proposal allows establishing a 0-RTT handshake upon the first website visit. Therefore, our design saves a round-trip compared to the connection establishment of the original QUIC, if the server experiences a low number of unrequited connections and the client received its *server config* out of band. An out of band provision may take place via DNS. Thus, the client receives a valid *server config* before connecting to the website similar to the TLS extension *Encrypted Server Name Indication* [26], which distributes cryptographic keys via DNS.

6.6 Related Work

The security properties of QUIC have been discussed in prior work. A first cryptographic analysis of QUIC was done by Fischlin et al. [11]. Moreover, Jager et al. [13] analyzed the key exchange of QUIC and presented an attack on the confidentiality of the encrypted messages. Lychev et al. [21] demonstrated attacks against QUIC which lead to connection failures, server load, or server DOS.

In this work, we investigated QUIC version 39 which was negotiated by default between Google's servers and Chrome version 67 in June 2018. The IETF aims to redesign the key exchange of QUIC by adapting TLS 1.3 [12], which does currently not deploy a concept similar to QUIC's *server config*. However, the adaption of TLS 1.3 requires QUIC implementations to solve the privacy problem of TLS session resumption [34] which is not present within the current QUIC variant. Furthermore, the adoption of TLS 1.3 will not solve the privacy problems related to the *source-address token*.

We observe, that proposals such as OPTLS 1.3 [16] and the IETF draft on a semi-static Diffie-Hellman key establishment [27] envision to add a handshake design to TLS 1.3 which is similar to the discussed QUIC handshake and requires the long-term storage of a server's public key by the client. If adopted, this would lead to similar privacy considerations as discussed in the context of QUIC's *server config*.

Note, that the current IETF draft on QUIC [12] discusses the privacy consequences arising from the migration of QUIC connections across different IP addresses of a client. However, this tracking mechanism does not allow tracking across several user connections as our presented approaches do, because the ConnID which is used for the migration process is randomly chosen by the client at the beginning of each new connection.

To the best of our knowledge, we are the first to report on user tracking via QUIC across several connections and validated our findings based on current browser implementations. While there exist several web tracking approaches [6], the presented tracking mechanisms are exceptional for allowing unique user identification based on the first client message in an Internet-scale deployed transport protocol. Other tracking mechanisms via the HTTPS stack such as TLS session resumption or IP addresses require either a previous establishment of a TCP connection or do not provide unique user identification.

6.7 Conclusion

The presented tracking mechanisms via QUIC's *source-address token* and *server config* enable web servers to track their user. In the case of tracking via the *server config* even network-based attackers are able to identify clients across several consecutive connections. To the practical impact of tracking via QUIC contributes that widely used browsers such as Google Chrome support QUIC by default. Furthermore, our measurements indicate that these browsers do not employ protective measures against tracking via the presented approaches. Thus, also online third-party trackers can exploit these mechanisms and benefit from the potentially reduced delays compared to tracking via HTTP cookies.

The introduced countermeasures show that user tracking is not an inevitable by-product of realizing 0-RTT connections. We find, that the proposed privacy-friendly connection establishment for QUIC mitigates tracking via source-address token and furthermore directs QUIC to new performance gains. Our proposal allows a 0-RTT connection establishments upon the first visit of a website, if the *server config* is provided out of band, e.g. via DNS. Compared to the original QUIC protocol, this saves an extra round-trip during the first connection establishment.

In summary, we hope that our work leads to a greater awareness about the privacy risks in the QUIC transport protocol and spurs further research on user tracking via the HTTPS stack.

6.8 References

[1] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. 2014. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 674–689.

[2] Abdurhman Albasir, Kshirasagar Naik, Bernard Plourde, and Nishith Goel. 2014. Experimental study of energy and bandwidth costs of web advertisements on smartphones. In *Mobile Computing, Applications and Services (MobiCASE), 2014 6th International Conference on*. IEEE, 90–97.

[3] Alexa Internet Inc. 2018. Alexa Top 1,000,000 Sites. Retrieved March 25, 2018 from http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

[4] Mike Bishop. 2019. *Hypertext Transfer Protocol Version 3 (HTTP/3)*. Internet-Draft draft-ietf-quic-http-18. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-18 Work in Progress.

[5] Mohamed Boucadair, Mat Ford, Phil Roberts, Alain Durand, and Pierre Levis. 2011. Issues with IP Address Sharing. RFC 6269. https://doi.org/10.17487/RFC6269

[6] Tomasz Bujlow, Valentín Carela-Español, Josep Sole-Pareta, and Pere Barlet-Ros. 2017. A survey on web tracking: Mechanisms, implications, and defenses. *Proc. IEEE* 105, 8 (2017), 1476–1510.

[7] Gerald Combs. 2017. Tshark- the Wireshark Network Analyser. *URL http://www.wireshark.org* (2017).

[8] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. 2013. ZMap: Fast Internet-wide Scanning and Its Security Applications.. In *USENIX Security Symposium*, Vol. 8. 47–53.

[9] S. Englehardt and A. Narayanan. 2016. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1388–1401.

[10] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan Mayer, Arvind Narayanan, and Edward W Felten. 2015. Cookies that give you away: The surveillance implications of web tracking. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 289–299.

[11] Marc Fischlin and Felix Günther. 2014. Multi-stage key exchange and the case of Google's QUIC protocol. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1193–1204.

[12] Jana Iyengar and Martin Thomson. 2018. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Internet-Draft draft-ietf-quic-transport-12. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-12 Work in Progress.

[13] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. 2015. On the security of TLS 1.3 and QUIC against weaknesses in PKCS# 1 v1. 5 encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1185–1196.

[14] Chris K Karlof, Umesh Shankar, et al. 2007. A Usability Study of Doppelganger, A Tool for Better Browser Privacy. (2007).

[15] Georgios Kontaxis and Monica Chew. 2015. Tracking Protection in Firefox For Privacy and Performance. *CoRR* abs/1506.04104 (2015). arXiv:1506.04104 http://arxiv.org/abs/1506.04104

[16] Hugo Krawczyk and Hoeteck Wee. 2016. The OPTLS protocol and TLS 1.3. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 81–96.

[17] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, et al. 2017. The QUIC transport protocol: Design and Internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 183–196.

[18] Adam Langley and Chang Wan-Teh. 2018. QUIC Crypto. Retrieved June 25, 2018 from https://www.chromium.org/quic

[19] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 878–894.

[20] Ben Laurie, Adam Langley, and Emilia Kasper. 2013. Certificate Transparency. RFC 6962. https://doi.org/10.17487/RFC6962

[21] Robert Lychev, Samuel Jero, Alexandra Boldyreva, and Cristina Nita-Rotaru. 2015. How secure and quick is QUIC? Provable security and performance analyses. In *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 214–231.

[22] Yishay Mansour, S. Muthukrishnan, and Noam Nisan. 2012. Doubleclick Ad Exchange Auction. *CoRR* abs/1204.0535 (2012). arXiv:1204.0535 http://arxiv.org/abs/1204.0535

[23] Jonathan R Mayer and John C Mitchell. 2012. Third-party web tracking: Policy and technology. In *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 413–427.

[24] Keaton Mowery and Hovav Shacham. 2012. Pixel perfect: Fingerprinting canvas in HTML5. *Proceedings of W2SP* (2012), 1–12.

[25] Refsnes Data. 2018. The Most Popular Browsers. Retrieved June 25, 2018 from www.w3schools.com/browsers/

[26] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. 2018. *Encrypted Server Name Indication for TLS 1.3*. Internet-Draft draft-ietf-tls-esni-02. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-02 Work in Progress.

[27] Eric Rescorla and Nick Sullivan. 2018. *Semi-Static Diffie-Hellman Key Establishment for TLS 1.3*. Internet-Draft draft-rescorla-tls13-semistatic-dh-00. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-rescorla-tls13-semistatic-dh-00 Work in Progress.

[28] Erik Roman and Matt Menke. 2018. NetLog: Chrome's network logging system. Retrieved June 25, 2018 from https://www.chromium.org/developers/design-documents/network-stack/netlog

[29] Jan Rüth, Ingmar Poese, Christoph Dietzel, and Oliver Hohlfeld. 2018. A First Look at QUIC in the Wild. In *International Conference on Passive and Active Network Measurement*. Springer, 255–268.

[30] Sebastian Schelter and Jérôme Kunegis. 2016. On the Ubiquity of Web Tracking: Insights from a Billion-Page Web Crawl. *arXiv preprint arXiv:1607.07403* (2016).

[31] Pyda Srisuresh and Kjeld Egevang. 2000. *Traditional IP network address translator (Traditional NAT)*. Technical Report.

[32] StatCounter. 2018. Desktop vs Mobile vs Tablet Market Share Worldwide. Retrieved June 25, 2018 from gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide

[33] StatCounter. 2019. The Most Popular Browsers. Retrieved January 25, 2019 from http://gs.statcounter.com/browser-market-share

[34] Erik Sy. 2018. Cross-domain Tracking with TLS Session Resumption. In *11th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2018)*.

[35] The Chromium Project. 2018. QUIC, a multiplexed stream transport over UDP. Retrieved June 25, 2018 from https://www.chromium.org/quic

[36] The Chromium Project. 2018. QUIC Wire Layout Specification. Retrieved June 25, 2018 from https://www.chromium.org/quic

[37] Shuai Yuan, Jun Wang, and Xiaoxue Zhao. 2013. Real-time bidding for online advertising: measurement and analysis. In *Proceedings of the Seventh International Workshop on Data Mining for Online Advertising*. ACM, 3.

7 Surfing the Web Quicker Than QUIC via a Shared Address Validation

Summary of this publication

Citation	Erik Sy. 2019. Surfing the Web quicker than QUIC via a shared Address Validation **(Best Student Paper Award)**. In *The 27th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2019)*. Split, Croatia.
Ranking	CORE: B
Status	Accepted for publication
Type of paper	Research paper
Aim	This paper aims to save a round-trip time during the connection establishments of the QUIC protocol. For this purpose, a mechanism is introduced to allow the sharing of the validation of a client's source address across different hostnames.
Methodology	Applied methods include a performance review on the protocol flow of QUIC's transport handshake and an empirical measurement of the configuration of web server serving popular hostnames. Furthermore, a performance simulation is conducted in the scope of this paper.
Contribution	This article provides the first description of a performance enhancement for QUIC's connection establishment based on a shared address validation across different hostnames. In detail, this paper proposes to reuse QUIC's address validation token between different hostnames that have a trust-relation to each other. This practice allows the peers saving a round-trip compared to QUIC's initial connection establishment. The performance evaluation of this proposal is conducted for the Alexa Top Thousand Sites assuming that hostnames with an existing TLS trust-relationships between each other are willing to conduct a shared address validation. The results indicate, that 60% of the required connection establishments during the retrieval of an average website can save a round-trip time using our proposal. Websites often require sequential connection establishments to different hostnames and our proposal may allow saving a round-trip time within several of these sequential connections. In total, we find that the establishment of all required connections for retrieving an average website can be accelerated by 142.2 ms assuming a round-trip time of 90 ms.
Co-authors' contribution	This article presents an independent work of the author without contributions by co-authors.

Abstract

QUIC is a performance-optimized secure transport protocol and a building block of the upcoming HTTP/3 standard. To protect against denial-of-service attacks, QUIC servers need to validate the IP addresses claimed by their clients. So far, the QUIC protocol conducts address validation for each hostname separately using validation tokens. In this work, we review this practice and introduce a new QUIC transport parameter to allow a shared address validation across hostnames. This parameter indicates to the client, that an issued validation token can be used to abbreviate the address validation when connecting to specific other hostnames. Based on trust-relations between real-world hostnames we evaluate the performance benefits of our proposal. Our results suggest that a shared address validation saves a round-trip time on almost 60% of the required handshakes to different hosts during the first loading of an average website. Assuming a typical transatlantic connection with a round-trip time of 90 ms, we find that deploying our proposal reduces the delay overhead to establish all required connections for an average website by 142.2 ms.

Keywords: QUIC transport protocol, address validation token, performance enhancement

7.1 Introduction

The world wide web is closely tied to the HTTP network protocol. The upcoming version HTTP/3 will replace the traditional TLS over TCP stack with the new QUIC protocol [3]. Thus, it is likely that QUIC will be widely adopted on the web within the forthcoming years. Further use cases include, but are not limited to the Domain Name System (DNS) [6].

A goal of the QUIC protocol is to reduce the delay required to establish secure connections. Faster connection establishments contribute to shorter page loading times, that correlate with an improved experience for web users [14]. To achieve this goal QUIC provides the feature of a zero round-trip time handshake that conducts the transport and the cryptographic connection establishment at the same time. However, this feature is only applicable when reconnecting to a QUIC server as it requires cached state of a prior connection. The first connection to a hostname requires up to two additional round-trips. One additional round-trip is caused by the cryptographic connection establishment and the other by the transport handshake.

In this work, we are addressing the performance limitations caused by this additional round-trip of the transport handshake that is used to validate the client's source address. To illustrate the shortcomings of QUIC's current design, we assume that a client connects sequentially to the hostnames *example.com* and *www.example.com*. Furthermore, we assume both of these hostnames are operated by the same entity and a strict address validation is deployed by the responding QUIC servers. We find that the default QUIC behavior causes per connection one additional round-trip for the validation of the client's address. As a result of this process, the client's address has been validated at the expense of two additional round-trips by the same entity via different hostnames.

We propose a performance-optimized approach that allows reusing address validation tokens across hostnames that have a trust-relation to each other. The client connects to the first hostname with an additional round-trip to validate the client's address and receives a validation token. The second handshake to the other hostname uses the validation token received during the first connection to pass the address validation without an additional round-trip. This approach saves in total a round-trip time compared to the default QUIC behavior.

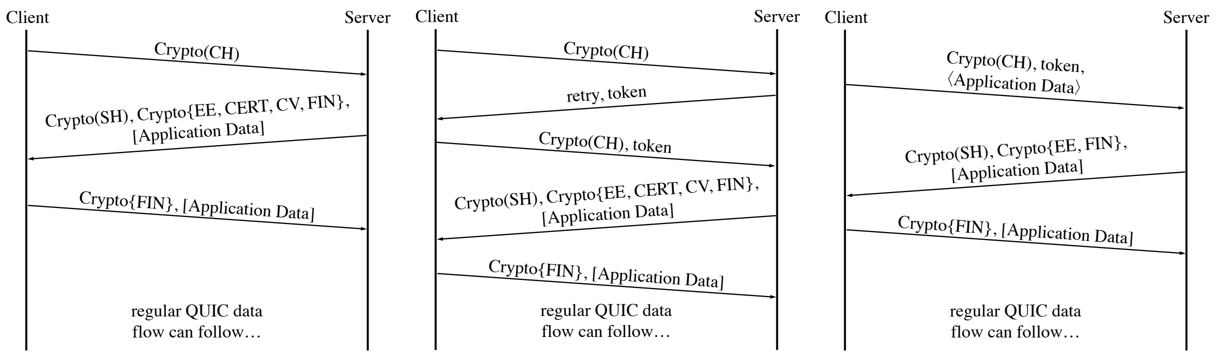
In summary, this paper makes the following contributions:

- We propose a new transport parameter for the QUIC protocol that enables a shared address validation across different hostnames.
- We demonstrate the performance gains yielded by our proposal for loading popular websites. Our results indicate that deploying the introduced transport parameter saves a round-trip time on almost 60% of the required connection establishments to different hosts during the first loading of an average website.

This proposal was presented to the IETF's QUIC working group, which aims to integrate the proposed shared address validation in the QUIC protocol [13].

The remainder of this paper is structured as follows: Section 7.2 describes the performance problem of the QUIC transport protocol that we aim to solve. Section 7.3 summarizes the proposed shared address validation and evaluation results are presented in Section 7.4. Related work is reviewed in Section 7.5, and Section 7.6 concludes the paper.

7.2 Problem Statement


Client
Server
Crypto(CH)
Crypto(SH), Crypto{EE, CERT, CV, FIN}, [Application Data]
Crypto{FIN}, [Application Data]
regular QUIC data flow can follow...
Client
Server
Crypto(CH)
retry, token
Crypto(CH), token
Crypto(SH), Crypto{EE, CERT, CV, FIN}, [Application Data]
Crypto{FIN}, [Application Data]
regular QUIC data flow can follow...
Client
Server
Crypto(CH), token, ⟨Application Data⟩
Crypto(SH), Crypto{EE, FIN}, [Application Data]
Crypto{FIN}, [Application Data]
regular QUIC data flow can follow...

a) Initial handshake b) Initial handshake with retry c) 0-RTT handshake

Figure 7.1: Handshakes in IETF QUIC protocol, where the brackets indicate different levels of encryption. Round brackets indicate no encryption and curved brackets denote encryption based on the handshake traffic secret. Square brackets signal encryption using the application traffic secret and angle brackets indicate the use of the early traffic secret for encryption.

In this section, we review the connection establishment of the IETF QUIC protocol [7]. Subsequently, we describe the problem that we aim to solve and our threat model.

7.2.1 Connection Establishment with IETF QUIC

Google developed and deployed the initial design of the QUIC protocol [8], which we will refer to as Google QUIC (gQUIC). In 2016, the Internet Engineering Task Force (IETF) started to standardize QUIC [7], which is still a work in progress. The IETF's draft of QUIC differs significantly from gQUIC, e. g., it uses TLS 1.3 [9] to conduct the cryptographic handshake. However, both QUIC variants aim to improve the performance of HTTPS traffic by conducting the cryptographic and the transport handshake concurrently, while the widespread approach using TLS over TCP conducts these handshakes sequentially. Figure 7.1 shows a schematic of QUIC's connection establishment.

Initial Handshake with and without Retry As shown in Figure 7.1 a), the client starts with the ClientHello (CH) message containing lists of supported cipher suites, protocol versions, TLS extensions, and public keys suitable for key exchange. All messages that are part of the cryptographic connection establishment follow the TLS 1.3 [9] protocol and are emphasized with *Crypto* in Figure 7.1. The server responds with its unencrypted ServerHello message, which is indicated by round brackets in Figure 7.1. Subsequently, the server computes the handshake traffic secret and sends the Encrypted Extensions (EE), the server's certificate (CERT), the Certificate Verify (CV), and the handshake finished (FIN) messages encrypted with this secret, shown as curved brackets in Figure 7.1. In the CV message, the server provides a fresh proof for its ownership of the certificate's private key. Whereas FIN messages signal a successful handshake and contain hashes of the exchanged handshake messages to verify that both peers observed the same messages. The server can now calculate the application traffic secret and may send encrypted application data. This encryption type is indicated with square brackets in Figure 7.1.

Upon receiving the ServerHello message, the client computes the handshake traffic secret. This enables the client to decrypt the received EE, CERT, CV, and FIN messages. The client can now authenticate the server's identity based on the provided CERT and CV messages. Then, the client validates the hashes contained in the server's FIN message and calculates the application traffic secret. Finally, the client responds with its own FIN message and may send encrypted application data to the server.

After a connection is successfully established, the server can provide the client with TLS resumption tickets and address validation tokens that can be used on subsequent connections. Resumption tickets allow resuming previous connections via abbreviated handshakes that decrease the delay overhead and save expensive cryptographic computations during connection establishment. A validation token is an encrypted and authenticated data block which is opaque to the client. It usually contains the client's visible IP address as seen by the server. If the client provides such a token during a subsequent connection establishment, this allows the server to compare the client's claimed IP address with the previously observed clients' IP address in the token.

Depending on its configuration, the server may per default or dynamically decide to strictly validate the client's IP address before proceeding with the cryptographic handshake. Figure 7.1 b) shows an initial handshake that validates the client's source address with an additional round-trip. Here, the server sends a retry message and an address validation token to the IP address claimed with the client's first message. To proof the ownership of the claimed IP address, the client is required to provide the received token along with its ClientHello message. Upon receiving the token from the client, the server validates it, which involves a comparison of the IP address stored in the token with the one claimed by the client. If the token is valid, the client-server pair will proceed with a standard initial connection establishment starting from the ServerHello message (see Figure 7.1 a).

0-RTT Connection Establishment Furthermore, the QUIC protocol provides zero round-trip time (0-RTT) connection establishments as shown in Figure 7.1 c). Here, the client can send data encrypted with the early traffic secret without waiting for a response from the server using TLS resumption. Thus, the client requires a previously retrieved resumption ticket and a pre-shared key (PSK) to encrypt these early data and to signal the used PSK to the server. Optionally, the client can provide an address validation token as shown in Figure 7.1 c) to anticipate a server's retry request.

Upon receiving these messages, the server starts to validate the client's token and IP address. In case of a positive validation result, the server begins with its own cryptographic operations. To derive the early traffic secret, the server uses also information provided in the *pre_shared_key* extension of the ClientHello message. Assuming, that the server can successfully decrypt the provided early data, it will signal this with the *pre_shared_key* extension in the ServerHello message. After sending the ServerHello message, the server will derive the handshake traffic secret and use it for encrypted transmission of the EE and FIN messages. Note, that this handshake does not require the server to provide a CERT and CV message to authenticate its claimed identity. Instead, the peers authenticate each other by successfully resuming the previous TLS session using the pre-shared key. This abbreviated authentication during resumed connections significantly decreases the delay and saves expensive cryptographic operations. Subsequently, the server can derive its application data secret and respond with encrypted application data to the client's early data.

Upon receiving the server's messages, the client will derive the handshake traffic secret and decrypt the server's EE and FIN message. Then, the client reviews the validation data contained in the server's FIN message to validate, that the exchanged messages have not been tampered with during transit. If the client did not determine a modification of the received data, it will provide the server its own FIN message. Finally, the client can derive its application traffic secret and the regular data flow follows.

Note, that application data encrypted with the early traffic secret does not provide forward-secrecy and might be replayed in other connections. For a comprehensive discussion on the weaker security guarantees of early data, we refer readers to RFC 8446 [9].

7.2.2 Performance Limitation of Address Validation

A QUIC server can validate the IP address claimed by a client by sending it a retry message with a token. Only, if the client returns this issued token, the server proceeds with the cryptographic handshake. This practice protects the server from spending expensive cryptographic operations on malicious handshake attempts claiming an illegitimate IP address. As a drawback, this IP spoofing defense increases the delay overhead of the connection establishment by a round-trip time.

To avoid this additional delay during the establishment of a new connection, a client can present a token from a previous connection to the same hostname along with the ClientHello message. If the server accepts this token as valid for the IP address claimed by the client, then it will directly proceed with the cryptographic connection establishment. This practice requires the client to retrieve a token in a previous QUIC connection to the same hostname.

QUIC does not consider using a retrieved address validation token to connect to new hostnames not matching the connection that has been used to retrieve the token. As a result, connections to every fresh hostname cannot present an address validation token leading to the described additional delay overhead if the server enforces address validation. This is a performance limitation if trust-relations between different hostnames are available. To illustrate this limitation, we assume a trust-relation between the hostnames *example.com* and *www.example.com*, which mutually accept their issued address validation tokens. Assuming the same network latency to both hostnames, a client establishing fresh connections to both hostnames experiences an additional delay overhead of twice the round-trip time due to address validation. However, if the client sequentially connects to these hostnames and reuses a token obtained in the first connection to establish the latter connection, this bisects the additional delay overhead to a single round-trip time.

To put this into perspective, a typical latency in North America is < 45 ms and < 90 ms for transatlantic connections [15]. However, several regions in the world exist which suffer from high network delays, often exceeding 300 ms [5]. In total, this example illustrates that using address validation tokens across different hostnames can provide significant performance gains, especially for connections experiencing high latencies.

7.2.3 Threat Model

To clarify the security aspects of the described problem, we define our threat model in the following.

The considered adversary is able to spoof the source address of its packets. However, we assume the address validation tokens to be cryptographically secure, thus attackers cannot generate valid tokens. In total, our adversary affects the security objective of availability. By spoofing the IP address of a victim's endpoint, the attacker causes the QUIC server to send its response to the victim, which is also known as a reflection attack. Moreover, the attacker can also directly attempt to exhaust the resources of the server by requesting connections from various spoofed IP addresses.

7.3 Shared Address Validation across Hostnames

This section describes our approach of a shared address validation across hostnames for the QUIC protocol. For that, we introduce the new transport parameter *validation_group* that aims to enable a shared address validation across hostnames.

Deviating from the standard initial handshake, the server unilaterally includes the *validation_group* value in its list of transport parameters. In detail, the *validation_group* presents a one-bit value, which is set to one if this feature is supported and zero otherwise.

The client finds the declared *validation_group* value in the server's EE message. If this value is set to zero, then the client reasons that the server does not support this feature and proceeds with its default behavior.

In case the *validation_group* value is set to one, the client concludes that the server supports a shared address validation across the hostnames, for which the presented TLS certificate is valid. We define a validation group to be a list of hostnames for which a single TLS certificate is valid and that mutually support address validation using tokens issued by any member of the same group. To support a shared address validation, the client associates received tokens during that connection with the validation group at hand.

To open a new connection to any member of a validation group, the client can use cached tokens associated with the same group. Tokens received during such a new connection must be associated with the same validation group.

The QUIC server does not provide a TLS certificate during a resumed connection establishment. In this case, the tokens received during the connection should be associated with the TLS certificate of the original connection, for which the server's identity was authenticated using a TLS certificate.

To mitigate tracking via address validation tokens [11], a client-side expiration mechanism is required for them. The lifetime of these tokens presents a performance versus privacy trade-off, where shorter lifetimes lead to better privacy protection. Based on an analysis of characteristic Internet traffic, the authors of [10] recommend a lifetime of ten minutes to address this trade-off in web browsers.

To support a shared address validation across hostnames, the cryptographic secret used to generate and decrypt the tokens needs to be shared across the servers serving these hostnames. Note, that address validation tokens are designed for single-use only. If a member of a validation group is not properly configured and fails to validate a legitimate token, then this process uses up a cached token of the respective validation group. In the worst case, this behavior can lead to reduced performance compared to the status quo. For example, if the client needs to respond to a retry message of a server because it used up its cached tokens on not properly configured servers.

7.4 Evaluation

In this section, we evaluate the performance benefit of our proposal for web browsing. For this purpose, we analyze the shared address validation across hostnames for the Alexa Top 1K Sites [1] using an analytical model. We start by describing our assumptions for this evaluation. Subsequently, we introduce the analyzed data set and summarize our results.

7.4.1 Assumptions

To analyze the loading behavior of popular websites, we assume that each established connection enforces a strict address validation. This assumption describes the current practice of TCP Fast Open [4] and gQUIC [8]. However, IETF QUIC provides an operation mode with a relaxed address validation known as stateless retry. Nonetheless, we believe the strict address validation to become the most popular operation mode of QUIC servers. As the draft on IETF QUIC is still in an early stage, we are not aware of online services deploying IETF QUIC at a production level. Thus, we cannot determine the configuration of real-world deployments of IETF QUIC to assess the accuracy of this assumption at the moment.

Moreover, this evaluation assumes that the investigated websites deploy the QUIC protocol to support their HTTPS connection establishment. To justify this assumption, we point out that numerous browser vendors and content delivery networks contribute to the draft of the QUIC

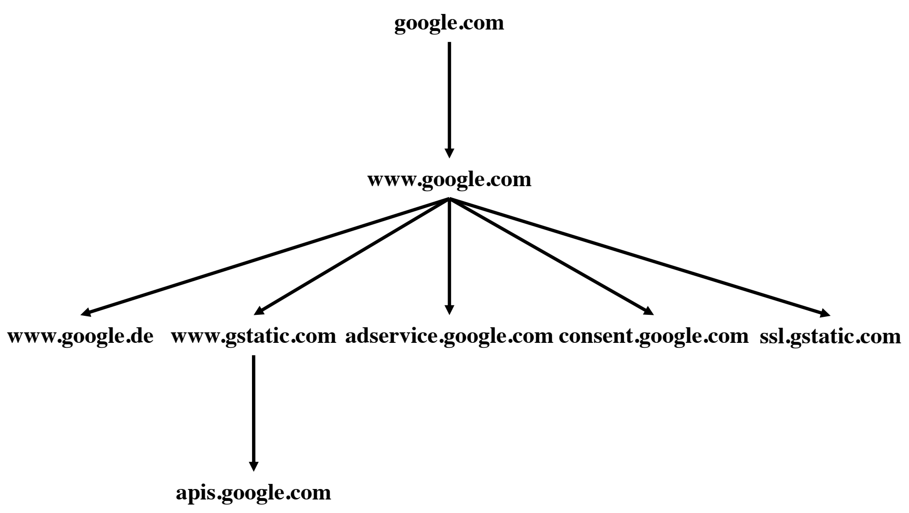


Figure 7.2: Domain tree of the website google.com.

protocol [7] and intend to deploy this protocol in their products. Furthermore, the upcoming HTTP/3 uses the QUIC protocol [3] by default, thus it will be deployed to serve websites. In total, it seems likely that the QUIC protocol will be widely deployed on the Internet within the forthcoming years.

In addition, to evaluate our proposal for popular websites we require information on real-world hostnames that trust each other with regard to the availability of their servers. To approximate this information, we assume that hostnames trusting each other with respect to the confidentiality of their communications are also willing to have a trust-relation concerning the availability of their servers. We define that a trust-relation with respect to confidentiality exists, if hostnames share a secret cryptographic state with each other such as the private key of their TLS certificate or they enable the resumption of TLS sessions across their hostnames. From our perspective, it seems reasonable that such closely cooperating hostnames, which are often operated by the same entity, are willing to trust each other in terms of the validation of their clients' IP addresses.

7.4.2 Data Set

This paper uses a data set on trust-relations of the Alexa Top 1K Sites, that is described and evaluated in [12]. The data set aggregates a scan of the Alexa Top 1K Sites [1] performed on the 8th of November 2018. In total, the scan successfully retrieved the domain trees of 839 websites. A domain tree describes an overview of the sequence of established connections to different hostnames during the retrieval of a website (see Figure 7.2). Furthermore, the data set collected real-world TLS trust-relations between the different hostnames within each domain tree. For that, the data set defines a TLS trust-relation between two hostnames if they either share the same TLS certificate or enable the resumption of TLS sessions between their hostnames.

7.4.3 Results

In this section, we analyze the performance benefits of our proposal for the first loading of popular websites. First, we investigate the number of connections necessary for the retrieval of

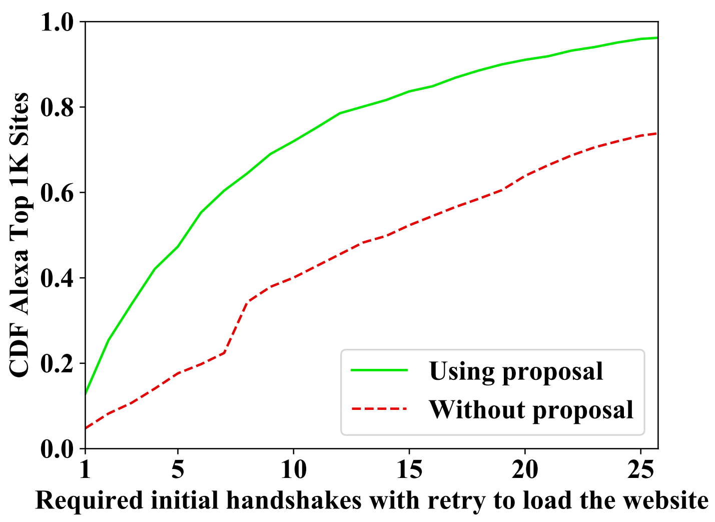


Figure 7.3: A cumulative distribution function (CDF) of Alexa Top 1K Sites is shown in dependence on the number of required initial handshakes with retry to retrieve the website. The green line marks the values considering the introduced shared address validation, while the red, dashed plot notes the current default. Note, that this plot is cut off at 25 handshakes.

an average website, that can each save a round-trip time during the connection establishment by deploying shared address validation. Subsequently, we observe that most websites require a client to sequentially establish connections to different hosts. Based on this practice, we analyze our proposal by measuring the aggregated delay overhead for establishing all QUIC connections necessary for the retrieval of the respective website successively. This evaluation of successive retrievals allows approximating the benefit of our proposal for the loading time of a website.

Share of Abbreviated Handshakes The analyzed data set [12] indicates, that TLS trust-relations are a common practice on the web. In total, the average Alexa Top 1K Site requires the client to establish 20.24 encrypted connections to different hostnames. Figure 7.3 plots the cumulative distribution function of the Alexa Top 1K Sites over the number of required initial handshakes with retry for their retrieval. The red, dashed plot represents the status quo and indicates that 95.2% of these websites require more than a single secure connection. Note, that 73.3% of the investigated sites can be loaded with at most 25 initial handshakes with retry. Thus, we cut off this plot after 25 of these handshakes for reasons of clarity.

The green plot represents the results of our analytical model using a shared address validation in case of a TLS trust-relation between the corresponding hostnames. If such a trust-relation exists, the client will reuse a received address validation token across this group of hostnames that trust each other. Thus, our model converts an initial handshake with retry (status quo) to an initial handshake without retry, if the client previously connected to any member of the same validation group to retrieve corresponding address validation tokens. Note, that our model investigates each website visit separately. Each converted handshake reduces the delay overhead of the specific connection establishment by a round-trip. Figure 7.3 shows that the use of a shared address validation shifts the distribution of the Alexa Top 1K Sites towards less required initial handshakes with retry. For example, the share of sites requiring a single initial handshake with retry increased from 4.8% to 12.9% by using our proposal. Furthermore, we find that 42.1% of the investigated websites can be retrieved with less than five initial handshakes with retry by using the introduced proposal compared to 14.1% without deploying the proposal. In total, we find that 96.0% of the Alexa Top 1K Sites can be retrieved with at most 25 initial handshakes with retry by deploying a shared address validation.

Our results for the average Alexa Top 1K Site suggest, that a shared address validation reduces the number of initial handshakes with retry from 20.24 to 8.35 for the first visit of an average website. Thus, 11.89 initial QUIC connections can be converted to use an address validation token received from a member of the same validation group. In total, the proposed practice saves for 58.75% of the established QUIC connections a round-trip time. This yields a reduction of 11.89 round-trips during the establishment of the required connections.

Delay Overhead The studied data set [12] provides insights into the sequence of established connections and which retrieved resources triggered the establishment of additional connections. Figure 7.2 provides the domain tree of *google.com* that marks initiated connections to different hostnames with arrows. We observe that the longest path within the domain tree requires four sequential connection establishments via the hostnames *google.com*, *www.google.com*, *www.gstatic.com*, and *apis.google.com*. Thus, the website retrieval of *google.com* is impacted by about four times the delay overhead of a single connection establishment.

Figure 7.4 plots the cumulative distribution of Alexa Top 1K Sites over the length of their longest paths of initial handshakes with retry. The red, dashed plot indicates the current status quo. We observe, that the Alexa Top 1K Sites require no more than eight sequential connections for their retrieval. The single most popular configuration requires four sequential connections and is used by 33.1% of the Alexa Top 1K Sites. In total, we find that 63.0% of the Alexa Top 1K Sites can be retrieved with four or less sequential connections.

The plot marked with a green line provides the results for deploying the proposed shared address validation. In our simulation, we convert if applicable an initial handshake with retry to an initial handshake, where the client presents an address validation token obtained from another member of the same validation group. Each of these converted handshakes saves a round-trip time during the corresponding connection establishment. Note, that our evaluation repeats the computation of the longest paths of initial handshakes with retry after simulating the shared address validation. Thus, the longest paths with and without using the introduced shared address validation can deviate from each other.

We find, that the deployment of our proposal leads to a significant reduction of necessary initial handshakes with retry for the Alexa Top 1K Sites. For example, using a shared address validation reduces the share of websites requiring more than five sequential connections from 37.0% to 2.3%. Furthermore, the number of websites requiring less than three connections increased from 15.4% to 56.5% when using our proposal.

Our results indicate, that the average Alexa Top 1K Site requires 4.04 sequential initial handshakes with retry. The proposed shared address validation reduces this value to 2.46. In total, our proposal saves 1.58 times the round-trip time until the last connection required for loading an average website is established. This presents a reduction of the longest path of initial handshakes with retry by 39.1% for the average website. Assuming a transatlantic connection with a round-trip time of 90 ms [15], our proposal reduces the delay overhead for establishing all required QUIC connections by 142.2 ms.

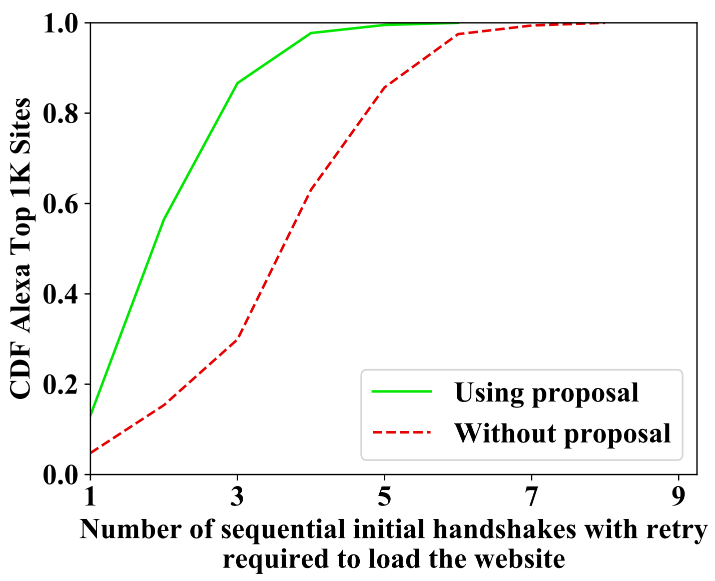


Figure 7.4: This plot shows a CDF of the Alexa Top 1K Sites over the number of required sequential initial QUIC connections with retry to load the respective website. The green line represents the values considering the introduced shared address validation, while the red, dashed plot marks the current default.

7.5 Related Work

To the best of our knowledge, we are the first to investigate the benefit of a shared address validation across hostnames for transport handshakes. Prior work includes the draft of the QUIC transport protocol, as it is designed to support a shared address validation across servers serving the same hostname.

Related work includes [12], that article investigates trust-relations within the domain trees of the Alexa Top 1K Sites and proposes TLS session resumption across hostnames to increase the number of resumed TLS handshakes during web browsing. However, this work focuses on QUIC's transport handshake and the task of validating the client's source address which are unrelated to TLS session resumption mechanisms.

Moreover, HTTP version 2 (HTTP/2) [2] can be considered as related work as it allows reusing TLS connections across different hostnames if these hostnames can be served from the same server address. In detail, HTTP/2 aims to reduce the number of established connections to yield performance gains. However, our proposal reduces the delay of the connection establishment itself. Thus, our proposal can be applied if HTTP/2 connection reuse is not feasible. For example, if two trusting hostnames are served via different IP addresses or a connection has already timed out.

7.6 Conclusions

This work proposes a new transport parameter for the QUIC protocol to support clients to use address validation tokens across hostnames. Our evaluation demonstrates, that such a shared address validation significantly reduces the delay overhead of QUIC's connection establishment on the real-world web.

7.7 References

[1] Alexa Internet Inc. 2019. Alexa Top 1,000,000 Sites. Retrieved March 13, 2018 from http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

[2] Mike Belshe, Roberto Peon, and Martin Thomson. 2015. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540. https://doi.org/10.17487/RFC7540

[3] Mike Bishop. 2019. *Hypertext Transfer Protocol Version 3 (HTTP/3)*. Internet-Draft draft-ietf-quic-http-19. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-19 Work in Progress.

[4] Yuchung Cheng, Jerry Chu, Sivasankar Radhakrishnan, and Arvind Jain. 2014. TCP Fast Open. RFC 7413. https://doi.org/10.17487/RFC7413

[5] Agustin Formoso, Josiah Chavula, Amreesh Phokeer, Arjuna Sathiaseelan, and Gareth Tyson. 2018. Deep Diving into Africa's Inter-Country Latencies. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2231–2239.

[6] Christian Huitema, Melinda Shore, Allison Mankin, Sara Dickinson, and Jana Iyengar. 2019. *Specification of DNS over Dedicated QUIC Connections*. Internet-Draft draft-huitema-quic-dnsoquic-06. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-huitema-quic-dnsoquic-06 Work in Progress.

[7] Jana Iyengar and Martin Thomson. 2019. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Internet-Draft draft-ietf-quic-transport-19. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-19 Work in Progress.

[8] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The QUIC transport protocol: Design and Internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 183–196.

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

[10] Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2018. Tracking Users Across the Web via TLS Session Resumption. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC '18)*. ACM, New York, NY, USA, 289–299. https://doi.org/10.1145/3274694.3274708

[11] Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2019. A QUIC Look at Web Tracking. *Proceedings on Privacy Enhancing Technologies* 3 (2019).

[12] Erik Sy, Moritz Moennich, Tobias Mueller, Hannes Federrath, and Mathias Fischer. 2019. Enhanced Performance for the encrypted Web through TLS Resumption across Hostnames. *arXiv preprint arXiv:1902.02531* (2019).

[13] Sy, Erik. 2019. Saving a round-trip time in the initial handshakes with retry. Retrieved May 27, 2019 from https://github.com/quicwg/base-drafts/issues/2552

[14] Matteo Varvello, Jeremy Blackburn, David Naylor, and Konstantina Papagiannaki. 2016. EYEORG: A Platform For Crowdsourcing Web Quality Of Experience Measurements. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies (CoNEXT '16)*. ACM, New York, NY, USA, 399–412. https://doi.org/10.1145/2999572.2999590

[15] Verizon. 2019. IP Latency Statistics. Retrieved March 13, 2019 from https://enterprise.verizon.com/terms/latency/

8 QUICker Connection Establishment with Out-Of-Band Validation Tokens

Summary of this publication

Citation	Erik Sy, Christian Burkert, Tobias Mueller, Hannes Federrath, and Mathias Fischer. 2019. QUICker Connection Establishment with Out-Of-Band Validation Tokens. In *2019 IEEE 44th Conference on Local Computer Networks (LCN) (LCN 2019)*. Osnabrück, Germany.
Ranking	CORE: A GGS: A- Microsoft Academics: A-
Status	Accepted for publication
Type of paper	Research paper
Aim	This paper introduces a cross-layer optimization to save a round-trip time during QUIC's connection establishment. For this purpose, out-of-band validation tokens are introduced to share the validation of the client's source address across various entities such as other QUIC servers or DNS resolvers.
Methodology	The used methodology comprises a performance review of QUIC's connection establishment, the design of a QUIC protocol using out-of-band validation tokens, and a simulation evaluating the performance properties of the proposed protocol flow.
Contribution	This paper describes a performance-optimization of QUIC's connection establishment using the presented out-of-band validation tokens. These tokens enable a shared address validation between a QUIC server and a trusted entity issuing the token. The article introduces two distribution mechanisms for these out-of-band tokens. One distribution mechanism uses DNS resolver to issue validation tokens to the clients. The other distribution mechanisms use other QUIC servers to provide these tokens. In total, this approach can save up to 100% of the round-trips required to conduct the source address validation during QUIC's connection establishment.
Co-authors' contribution	The article was co-authored by Christian Burkert, Tobias Mueller, Prof. Dr. Hannes Federrath, and Prof. Dr. Mathias Fischer. Christian Burkert and Prof. Dr. Mathias Fischer assisted to revise and refine this paper. Tobias Mueller and Prof. Dr. Hannes Federrath provided general feedback.

Abstract

QUIC is a secure transport protocol that improves the performance of HTTPS. An initial QUIC handshake that enforces a strict validation of the client's source address requires two round-trips. In this work, we extend QUIC's address validation mechanism by an out-of-band validation token to save one round-trip time during the initial handshake. The proposed token allows sharing an address validation between the QUIC server and trusted entities issuing these tokens. This saves a round-trip time for the address validation. Furthermore, we propose distribution mechanisms for these tokens using DNS resolvers and QUIC connections to other hostnames. Our proposal can save up to 50% of the delay overhead of an initial QUIC handshake. Furthermore, our analytical results indicate that 363.6 ms in total can be saved for all connections required to retrieve an average website, if a round-trip time of 90 ms is assumed.

8.1 Introduction

This paper investigates the design of the QUIC protocol [1], which is currently standardized. It is a secure transport protocol designed to replace TLS over TCP within the upcoming HTTP/3 version. As the world wide web is closely tied to the Hypertext Transfer Protocol (HTTP) and the standardization work on QUIC receives widespread support, we expect the QUIC protocol to be widely deployed on the Internet within the forthcoming years.

The majority of web traffic consists of short-lived connections, for which the connection establishment represents a significant delay overhead [2]. QUIC's initial handshake requires two round trips to establish the connection. One round-trip accounts for the cryptographic connection establishment and the other for a challenge-response mechanism known as stateless retry, which validates the source address claimed by the client to prevent IP spoofing attacks. Moreover, QUIC provides zero round-trip time handshakes for resumed sessions. This allows clients to send encrypted requests directly without waiting for the server's first handshake messages.

To further improve the performance of QUIC's initial handshake, we propose a mechanism to save one additional round trip by outsourcing the address validation mechanism.

To illustrate the basic idea, we assume a website (google.com) that trusts a DNS resolver (Google DNS) to issue address validation tokens. Thus, if a client resolves google.com at Google DNS, it also retrieves an opaque token encoding its publicly visible source address. Subsequently, the client includes this out-of-band validation token in its connection request sent to google.com. Later, the web server validates that the presented token matches the claimed source address of the client. If so, the address validation is completed and in total, a round-trip time has been saved.

In summary, this paper makes the following contributions:

- We propose out-of-band validation tokens that enable a shared address validation between a QUIC server and trusted entities issuing such tokens.
- We propose mechanisms to distribute out-of-band validation tokens via DNS resolvers and other QUIC connections.

- We demonstrate the performance improvements gained by out-of-band validation tokens. Our results indicate that our proposal saves up to 50% of the delay overhead of initial QUIC connection establishments.

The remainder of this paper is structured as follows: Section 8.2 introduces the proposed out-of-band validation token. Evaluation results are presented in Section 8.3. Related Work is reviewed in Section 8.4 and Section 8.5 concludes the paper.

8.2 Out-Of-Band Validation Token

This section introduces the out-of-band validation token for the QUIC protocol. Subsequently, distribution mechanisms for such out-of-band tokens are proposed using DNS resolvers and QUIC connections to other hostnames.

8.2.1 Token Design

Address validation tokens present a defense mechanism against source address spoofing by malicious clients. For this purpose, the QUIC server compares the claimed client address with the previously observed source address encoded in the presented token. So far, only QUIC servers themselves can issue address validation tokens for their connections. Out-of-band validation tokens extend this mechanism by allowing external entities to issue these tokens.

To generate these out-of-band tokens for arbitrary client's, the QUIC server is required to share instructions and a secret key with the corresponding external entity. Upon receiving an out-of-band token, the client imports it in its cache, marks it as received in an out-of-band situation, and associates the QUIC server's hostname to it. To establish a fresh connection to the respective hostname, the client includes a cached token in the send initial packet. If the client's cache contains several tokens, the client must prefer the usage of validation tokens received by the QUIC server itself over cached out-of-band tokens.

The server may share different secret keys with different external entities. This approach allows a selected invalidation of tokens that have been issued using a specific secret key. Thus, the QUIC server can revoke the secret key provided to an external entity if, e.g., a large number of unrequited connection requests is observed that use tokens issued by the same key. If a setup with dedicated secret keys per external entity is deployed, it is recommended to attach an identifier to the token, that indicates which key was used to generate the specific token.

Note, that according to the draft of IETF QUIC [1] the server treats an invalid token as if the client did not present a token. Thus, the number of required round-trips during the connection establishment is identical if the client presents an invalid out-of-band token or the client's connection request does not contain a token at all.

8.2.2 Token Distribution Mechanisms

To substantiate the real-world benefit of out-of-band tokens, we present in this section two distribution mechanisms for such tokens. First, we introduce the distribution via the Domain

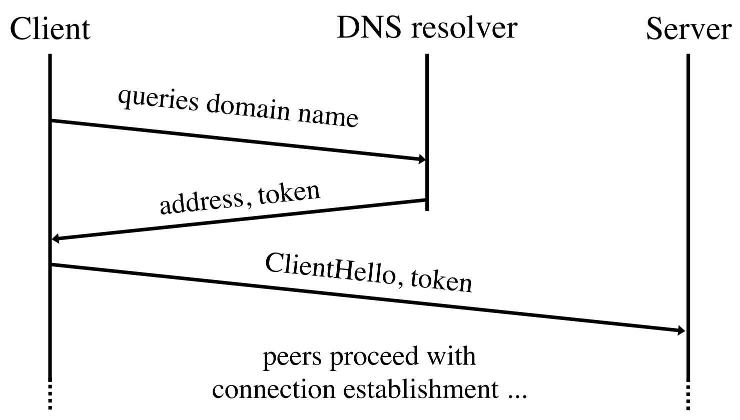


Figure 8.1: Proposed connection establishment avoiding stateless retry by using a DNS resolver to issue validation token.

Name System (DNS). Then, we describe a distribution mechanism using QUIC connections to other hostnames for this purpose.

Distribution via DNS Resolver To save a round-trip time via the proposed out-of-band tokens, the client needs to receive the token before sending the connection request to the corresponding QUIC server. Furthermore, clients query a domain name to look up the source address before they send their connection request. Thus, DNS seems to be a suitable place to distribute out-of-band tokens as the connection request often directly follows the corresponding DNS lookup. Figure 8.1 provides a schematic of this proposed distribution mechanism. This proposal is not limited to a specific DNS standard and can be applied to the traditional DNS and newer versions deploying transport encryption such as DNS over Transport Layer Security (DOT) and DNS over HTTPS (DOH). In general, the DNS protocol needs to be extended by a new record type, which we define as QUICTOKEN.

If the client wishes to establish a fresh QUIC connection to a domain name for which it has not a cached token for future connections available, it queries the domain name (including the QUICTOKEN record type) as shown in Figure 8.1. The DNS resolver proceeds with the default resolution of the source address associated with the domain name. Additionally, if the DNS resolver supports the record type QUICTOKEN and is capable to generate valid out-of-band tokens for this queried domain name, it can include such a token in the response sent to the client.

Note, that to construct valid out-of-band tokens, the resolver needs to be trusted by the server hosting the specific domain name. Thus, the respective server operator shared in advance the instructions and the secret keys required to generate valid tokens for this domain. Upon receiving the source address and the token, the client constructs its QUIC connection request and attach the obtained out-of-band token to it before sending it to the received server source address. Subsequently, the server validates the presented token and proceeds with its normal connection establishment.

To ensure that clients do not reuse tokens across different connections, it is required that the record type QUICTOKEN must not be cached except by the client. This can be realized by setting the Time to Live (TTL) of the QUICTOKEN record type to zero seconds. Note, that this

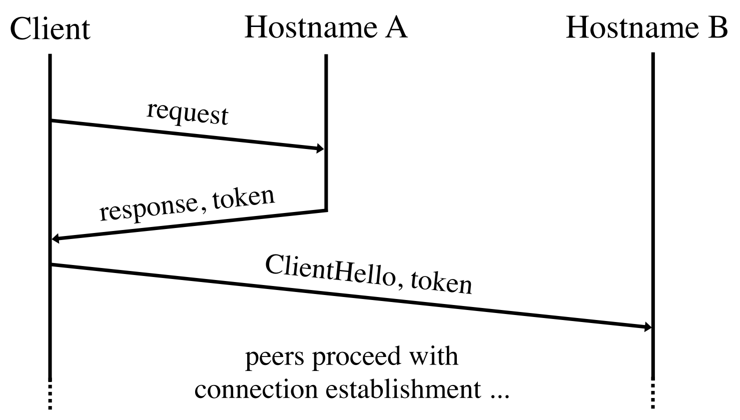


Figure 8.2: Proposed distribution mechanism using the QUIC connection to hostname A to receive an out-of-band token valid for hostname B.

configuration of QUICTOKEN does not affect the caching mechanisms of for example A or AAAA record types.

A limitation of this distribution mechanism arises if the DNS resolver is located within the same private network as the client. In this case, the client's source address as seen by the DNS resolver might mismatch the publicly visible source address as seen by the QUIC server. Thus, the address validation is likely to fail because the source address encrypted in the token does not match the claimed source address as observed by the QUIC server. This issue can be solved by for example moving the DNS resolver to a public IP address or using STUN [3] to learn the client's public source address.

Distribution via other QUIC Connections This distribution mechanism assumes that the client first establishes a QUIC connection to hostname A before it sends a connection request to hostname B. Furthermore, we assume that hostname B allows hostname A to issue valid out-of-band tokens for its service and therefore shares instructions and its secret key required to construct these tokens for arbitrary source addresses. We propose a new EXTERNAL_TOKEN frame for the QUIC protocol, which allows QUIC servers to provide clients with out-of-band tokens for arbitrary hostnames. However, tokens for future connections to the same hostname A should use the existing NEW_TOKEN frame. Note, that tokens for future connections are regarded as trustworthy as they are issued by the same server which is also consuming them. However, out-of-band tokens are not treated as trustworthy because the client does not validate that the entity issuing these tokens is authorized to do so. Compared to the NEW_TOKEN frame of the QUIC protocol, tokens received via the EXTERNAL_TOKEN frame are only used to establish a fresh connection if the client would otherwise send the connection request without an attached address validation token.

Figure 8.2 shows a schematic of this distribution mechanism. The client has an established QUIC connection to hostname A. Hostname A reasons based upon its provided response, that the client is likely to establish a connection to hostname B. This is for example the case, if the provided response contains a hyperlink to a resource hosted by hostname B. To speed up this connection establishment between the client and hostname B, hostname A decides to provide an out-of-band token for the client's source address valid for hostname B.

Upon receiving this EXTERNAL_TOKEN frame from hostname A, the client checks first if it has a token for future connections for hostname B. If not, the client establishes a fresh connection to hostnames B by attaching the received out-of-band token to its connection request. Otherwise, the client will prefer to include its cached token for future connections (received in a previous connection to the hostname B) in the connection request to hostname B. Upon receiving the client's connection request, hostname B validates the included address validation token and proceeds with the usual connection establishment.

It seems reasonable to expect that a QUIC server will only issue out-of-band tokens for other hostnames for which it is likely that the client will soon connect to them. Out-of-band tokens should have an expiration mechanism, thus received tokens may expire if no connection is established to a corresponding hostname within a short period.

8.3 Evaluation and Discussion

To demonstrate the feasibility of our proposal, we evaluate and discuss aspects of its performance, security.

8.3.1 Performance

In this section, we present a performance evaluation for out-of-band tokens. First, we describe our results with respect to the establishment of a single QUIC connection. Subsequently, we evaluate the performance impact of our proposal on an average website visit.

Benefits for single Connections Using an out-of-band token to validate the client's source address saves a round-trip compared to using QUIC's stateless retry mechanism. As the QUIC protocol is still work in progress, only experimental implementations of its design exist. Thus, we will use an analytical model to approximate the performance benefit of our proposal on the delay overhead of the connection establishment. For this evaluation, we approximate the delay overhead for the initial connection establishment as shown in Equations 8.1 and 8.2. Here, $t_{Default}$ and $t_{Proposal}$ indicate the delay overhead for the current status quo and our proposal, respectively. Furthermore, RTT denotes the round-trip time between both peers and t_{proc} marks the total time required by the peers to process the connection establishment.

$$t_{Default}(RTT) = t_{proc} + 2 * RTT \tag{8.1}$$

$$t_{Proposal}(RTT) = t_{proc} + RTT \tag{8.2}$$

Within our analytical model, we assume that the processing of the connection setup t_{proc} takes 40 ms independent of the round-trip time. We chose 40 ms because this approximates the time by the TLS 1.3 over TCP protocol stack for a similar task [6].

Figure 8.3 plots the delay overhead of the initial handshake over the round-trip time for our analytical model. We find, that the green, dashed line indicating our proposal provides significantly better results than QUIC's status quo marked by a red, dotted plot. The performance improvement achieved by our proposal depends on the RTT. Assuming a transatlantic connection with a round-trip time of 90 ms [7], we find that a connection establishment using our proposal requires

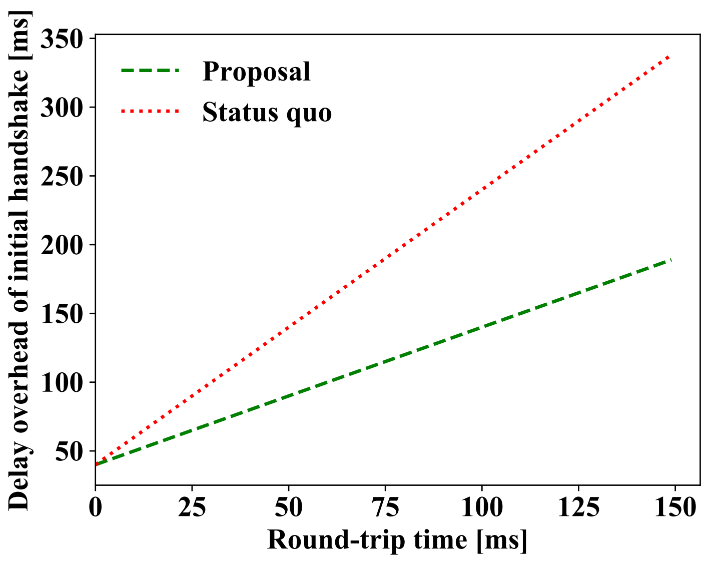


Figure 8.3: This plot shows the simulated delay overhead of QUIC's initial connection establishment over the round-trip time. The red, dotted line plots QUIC's default behavior and the green, dashed line deploys our proposed out-of-band validation tokens.

only 60% of the default delay overhead. Furthermore, we derive from Equation 8.1 and 8.2 that our proposal reduces the investigated delay overhead by 50% when RTT converges to infinity.

Gains for Web Browsing As the QUIC protocol will be a building block of the upcoming HTTP/3 network protocol, it seems interesting to evaluate the performance impact of our proposal on website retrieval. A recent study reported, that the retrieval of popular websites requires on average 20.24 encrypted connections to different hostnames [6]. For this evaluation, we assume that all of these hostnames support the QUIC protocol and that they all enable the client's DNS resolver to issue out-of-band tokens. Thus, we find that we can save a round-trip time during each connection establishment if the corresponding web server enforces a strict source address validation before proceeding with the cryptographic handshake.

Furthermore, it is found by [6], that the average popular website requires up to 4.04 sequentially established connections. This finding can be attributed to the fact, that a retrieved web resource often triggers the establishment of additional connections to retrieve further resources. Thus, saving a round-trip time via the proposed out-of-band tokens allows in total to save on average 4.04 round-trips until all required connections are established. Assuming a round-trip time of 90 ms, as it is typical for transatlantic connections [7], we find that 363.6 ms can be saved until the last connection for retrieving the website is established.

8.3.2 Security

In this section, we review possible security concerns with respect to out-of-band validation tokens. First, we address the impact of our proposal on the mitigation of Denial-of-Service attacks. Then, we look at risks arising from using address validation tokens from possibly unauthorized origins.

Denial-of-Service Attacks By sharing the instructions and the secret keys to generate address validation tokens with other entities, the risk that this confidential information gets compromised increases. In case of a compromise, an adversary can issue tokens for arbitrary source addresses. Thus, the adversary can send connection requests with a spoofed source address to the QUIC server, that contain a valid token for the claimed address. As a result, the server experiences a large number of spoofed connection requests that consume its available resources up to a Denial-of-Service attack. To mitigate such an event, the server should monitor connection requests associated with trusted secret keys. If the number of spoofed connection requests exceeds a threshold, the server revokes that specific secret key to mitigate Denial-of-Service attacks. After such a revocation all tokens issued by this key are treated as invalid. However, the revocation of a secret key might also cause a stateless retry for legitimate connection requests and thus causes a performance degradation for these connection attempts. To address this security versus performance tradeoff, it is advised to provide different secret keys to different entities. Thus, a revoked key affects only validation tokens expected to be issued by a specific entity. In total, key revocation provides an effective mechanism to protect against the considered Denial-of-Service attacks.

Unauthorized Origins In our proposal, the client does not validate whether an external entity is authorized by the affected QUIC server to issue out-of-band tokens. Thus, it is necessary to review the case in which an unauthorized external entity issues invalid out-of-band tokens. The draft of IETF QUIC [1] instructs that servers treat invalid tokens (for future connections) as if the client did not present a token at all. Therefore, the client experiences no drawbacks by presenting an invalid out-of-band validation token and in total, the client does achieve the same performance as including no token in the connection request. However, if the client has a valid token for future connections and an invalid out-of-band token from an unauthorized origin in its cache, then only a connection request including the token for future connections can save a round-trip during the address validation. Concluding, tokens for future connections are more trustworthy, as they have been retrieved via an authenticated connection to the respective QUIC server. For this reason, the usage of tokens for future connections should be preferred over out-of-band validation tokens because clients do not validate that a trust-relation between the entity issuing the out-of-band tokens and the QUIC server consuming them exists. However, out-of-band tokens are preferential compared to using no token at all, as they can reduce the delay overhead of the connection establishment by up to a round-trip time but always achieve at least the performance of a handshake without a token.

8.4 Related Work

Performance improvements of the QUIC protocol with respect to the performance penalty caused by a stateless retry are actively discussed within the Internet Engineering Task Force (IETF) QUIC working group. So far, these discussions focus on extending the number of entities that are allowed to issue address validation tokens for other hostnames either based on existing

TLS trust-relations [5] or based on the source address from which a respective hostname is served [4].

This work extends the applicability of the related work because clients can use out-of-band tokens upon the first connection request to any QUIC server, assuming that their DNS resolver is capable to provide a corresponding token. Thus, this proposal outperforms the related work by saving up to 100% of the stateless retries usually required if a strict source address validation is enforced.

8.5 Conclusions

This paper proposes out-of-band validation tokens for a shared address validation between a QUIC server and trusted entities issuing these tokens. Our evaluation indicates, that the proposed tokens enable significant performance gains for clients and servers without affecting the user's privacy and communication security.

8.6 References

[1] Jana Iyengar and Martin Thomson. 2019. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Internet-Draft draft-ietf-quic-transport-19. Internet Engineering Task Force. Work in Progress.

[2] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The QUIC transport protocol: Design and Internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 183–196.

[3] Philip Matthews, Jonathan Rosenberg, Dan Wing, and Rohan Mahy. 2008. Session Traversal Utilities for NAT (STUN). RFC 5389. https://doi.org/10.17487/RFC5389

[4] Kazuho Oku. 2019. *Address-bound Token for QUIC*. Internet-Draft draft-kazuho-quic-address-bound-token-00. Internet Engineering Task Force. Work in Progress.

[5] Erik Sy. 2019. Surfing the Web quicker than QUIC via a shared Address Validation. *arXiv preprint arXiv:1903.09466* (2019).

[6] Erik Sy, Moritz Moennich, Tobias Mueller, Hannes Federrath, and Mathias Fischer. 2019. Enhanced Performance for the encrypted Web through TLS Resumption across Hostnames. *arXiv preprint arXiv:1902.02531* (2019).

[7] Verizon. 2019. IP Latency Statistics. Retrieved March 13, 2019 from https://enterprise.verizon.com/terms/latency/

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

Summary of this publication

Citation	Erik Sy, Tobias Mueller, Moritz Moennich, and Hannes Federrath. 2019. Accelerating QUIC's Connection Establishment on High-Latency Access Networks. In *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC) (IPCCC 2019)*. London, United Kingdom (Great Britain).
Ranking	CORE: B GGS: B Microsoft Academics: B
Status	Accepted for publication
Type of paper	Research paper
Aim	This paper aims to optimize the performance of QUIC connection establishments directly following a corresponding DNS lookup. For this purpose, a QuicSocks proxy is introduced that conducts the DNS lookup on behalf of the user and supports the process of connection establishment between client and server.
Methodology	The used methodology comprises a performance review of the protocol flow of DNS lookups as well as the protocol flow of the QUIC protocol. Furthermore, the protocol flow of the QuicSocks proxy has been designed and analytically evaluated. The conducted experiments include an implementation of a QuicSocks prototype as well as performance measurements using this prototype. Furthermore, empirical measurements using 800 test nodes deployed in several access networks and their pre-configured DNS resolver have been conducted and analyzed.
Contribution	This article introduces a novel QuicSocks design aiming to accelerate the DNS resolution and the connection establishment for QUIC connections. For this purpose, the client sends initial handshake messages without a prior resolution of the respective domain name to the new QuicSocks proxy. This proxy is located in a favorable network position such as a co-location with the ISP's DNS resolver. This QuicSocks proxy is conducting the DNS lookup on behalf of the client and forwards the client's initial messages to the corresponding IP address. Furthermore, the QuicSocks proxy responds to stateless retry requests of the QUIC server. This paper includes analytical and empirical evaluations of the proposed design. Results indicate, that 10% of the investigated Atlas Ripe nodes save at least 30 ms to complete their QUIC connection establishment using our proposal compared to the status quo. Moreover, in the scope of this work, a QuicSocks prototype has been implemented. The performance measurements based

	on the prototype indicate that the overhead of the QuicSocks proxy is lightweight and contributes less than 1.2 ms to a QUIC connection establishment.
Co-authors' contribution	The article was co-authored by Tobias Mueller, Moritz Moennich and Prof. Dr. Hannes Federrath. Tobias Mueller assisted in implementing the QuicSocks prototype and in conducting the prototype-based performance measurements. Moritz Moennich assisted in the measurement study using the RIPE Atlas network. Prof. Dr. Hannes Federrath provided general feedback.

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.

Keywords: QUIC Transport Protocol, SOCKS Proxy, DNS, QuicSocks Proxy

9.1 Introduction

For U.S. households latency is the main web performance bottleneck for broadband access networks exceeding a throughput of 16 Mbit/sec [21]. 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 [25] and negatively impact the per-user revenue of online service provider [19]. 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 [21]. 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 [1]. 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 the 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 [14]. 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 [2].

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 9.2 introduces the QUIC and the SOCKS protocol and describes the performance problem that we aim to solve. Section 9.3 summarizes the proposed QuicSocks design and evaluation results are presented in Section 9.4. Related work is reviewed in Section 9.5, and Section 9.6 concludes the paper.

9.2 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.

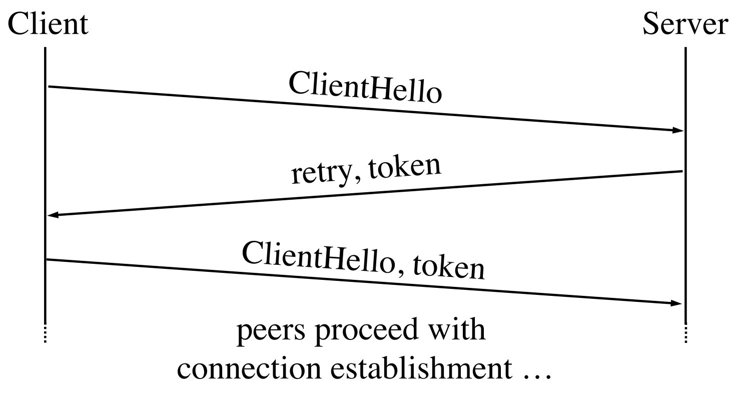


Figure 9.1: Schematic of QUIC's stateless retry mechanism.

9.2.1 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 [8]. The QUIC transport protocol will replace TLS over TCP in the upcoming HTTP/3 network protocol [1], 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 [22], mitigates head-of-line blocking [11], and can be extended because of a lower interference through middleboxes [7].

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.

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 9.1, 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.

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 [5], or clients changing

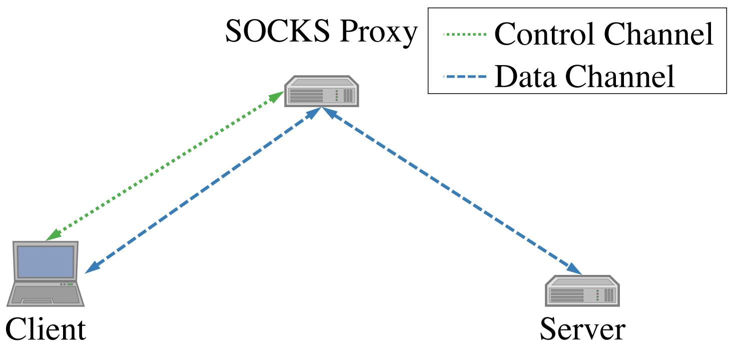


Figure 9.2: Schematic of a network packet exchange between client and server employing an intermediate SOCKS proxy.

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.

9.2.2 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 [10], and privacy-enhancing technologies such as TOR onion routing [3]. Figure 9.2 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 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.

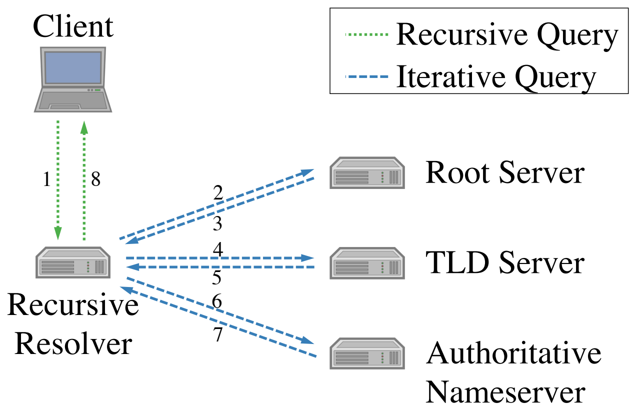


Figure 9.3: Schematic of a complete DNS query using a recursive resolver.

9.2.3 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 [9]. A popular website requires connections to about 20 different hostnames [23]. 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 [23]. 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 9.3. 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 9.3 indicate such a complete iterative query involving the DNS root server, Top Level Domain (TLD) 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% [9]. 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 [21]. However, a fraction of about 5% of the users experience a RTT longer than 20 ms [28]. 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 [28]. 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 [21]. 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.

9.3 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.

9.3.1 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 [17].
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.

9.3.2 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 9.4. 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 QuicSocks 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 datagrams' 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 9.4. 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

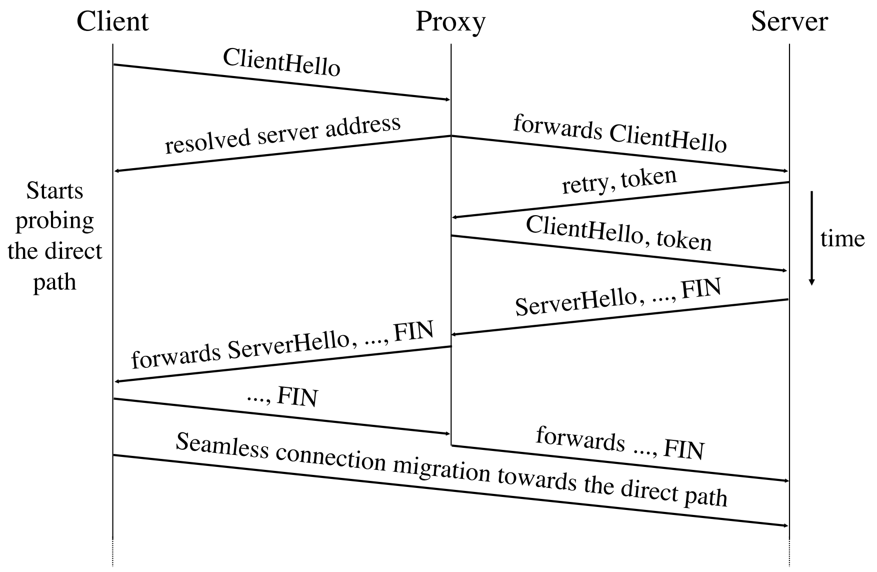


Figure 9.4: Protocol flow of a QUIC handshake via the proposed QuicSocks proxy, who resolves the server's IP address. The handshake includes an optional stateless retry initiated by the server. After the server and client established forward secure keys and exchanged FIN messages, the client initiates the connection migration towards the direct path.

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.

9.3.3 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 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.

9.4 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.

9.4.1 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 9.5). Furthermore, our model is reduced to the network latency between the involved peers. As shown in Figure 9.5, we denote the round-trip time between client and DNS resolver/ QuicSocks proxy as RTT_{DNS}. $\text{RTT}_{\text{direct}}$ and $\text{RTT}_{\text{Server}}$ describe the round-trip time between server and client, and between server and QuicSocks proxy, respectively.

Table 9.1 presents the evaluation results for our analytical model. A connection without stateless retry requires RTT_{DNS} to resolve the domain name and subsequently $\text{RTT}_{\text{direct}}$ 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 $\text{RTT}_{\text{Server}}$ 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

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 9.1: 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.

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.

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} [20]. A typical RTT_{direct} in the LTE mobile network of the U.S. is 60 ms to reach popular online services [15]. For this example, we assume RTT_{DNS} and RTT_{Server} to be each 30 ms, while RTT_{direct} is 60 ms.

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

Table 9.2: 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.

Table 9.2 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 [15].

9.4.2 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

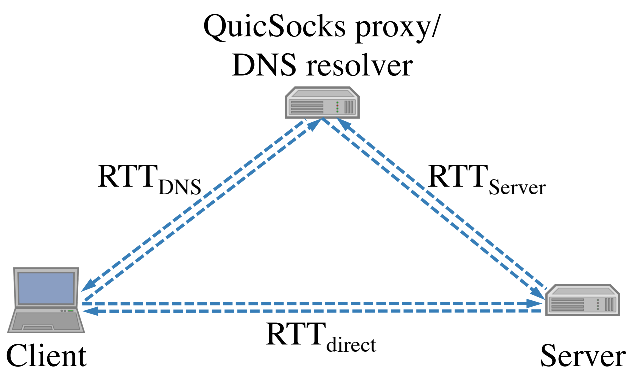


Figure 9.5: Network topology with colocation of DNS resolver and QuicSocks proxy. Furthermore, this overview marks the round-trip time (RTT) between different peers.

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.

Data Collection Our data collection aims to measure $\mathrm{RTT}_{\mathrm{DNS}}$, $\mathrm{RTT}_{\mathrm{Server}}$, and $\mathrm{RTT}_{\mathrm{direct}}$ for different real-world clients. We use nodes of the RIPE Atlas network [16] 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 [13] that may return different physical endpoints when pinged from the client and the server, respectively. We used message 6 in Figure 9.3, 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 $\mathrm{RTT}_{\mathrm{DNS}}$ and $\mathrm{RTT}_{\mathrm{Server}}$ 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 $\mathrm{RTT}_{\mathrm{DNS}}$, $\mathrm{RTT}_{\mathrm{Server}}$, and $\mathrm{RTT}_{\mathrm{direct}}$ 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.

Results To accelerate a connection establishment via our proposal, we require $\mathrm{RTT}_{\mathrm{Server}}$ to be smaller than $\mathrm{RTT}_{\mathrm{direct}}$. Figure 9.6 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 $\mathrm{RTT}_{\mathrm{direct}}$, $\mathrm{RTT}_{\mathrm{Server}}$, and $\mathrm{RTT}_{\mathrm{DNS}}$ as solid, dashed, and dotted lines, respectively. Our results indicate for almost all clients $\mathrm{RTT}_{\mathrm{Server}}$ is significantly smaller than $\mathrm{RTT}_{\mathrm{direct}}$. For 51% of the considered RIPE Atlas nodes, $\mathrm{RTT}_{\mathrm{Server}}$ is at least 5 ms smaller than $\mathrm{RTT}_{\mathrm{direct}}$. Furthermore, 36.7% of the nodes experience $\mathrm{RTT}_{\mathrm{Server}}$ to be at least 10 ms smaller than $\mathrm{RTT}_{\mathrm{direct}}$. As can be observed in Figure 9.6, almost no nodes experiences $\mathrm{RTT}_{\mathrm{Server}}$ to be longer than 40 ms, while a tail of 10% of the respective RIPE Atlas nodes observe a longer $\mathrm{RTT}_{\mathrm{direct}}$. In this long tail, we find 7.2% and 3.8% of the nodes to have a $\mathrm{RTT}_{\mathrm{Server}}$ that outperforms $\mathrm{RTT}_{\mathrm{direct}}$ by at least 40 ms and 50 ms, respectively. Furthermore, Figure 9.6 provides a plot of $\mathrm{RTT}_{\mathrm{DNS}}$. We find, that 60% of the nodes have a RTT with their ISP-provided DNS resolver of less than 10 ms. Moreover, $\mathrm{RTT}_{\mathrm{DNS}}$ is almost always smaller than $\mathrm{RTT}_{\mathrm{direct}}$ 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 9.1 with the measured RTT. Figure 9.7 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 9.7 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 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.

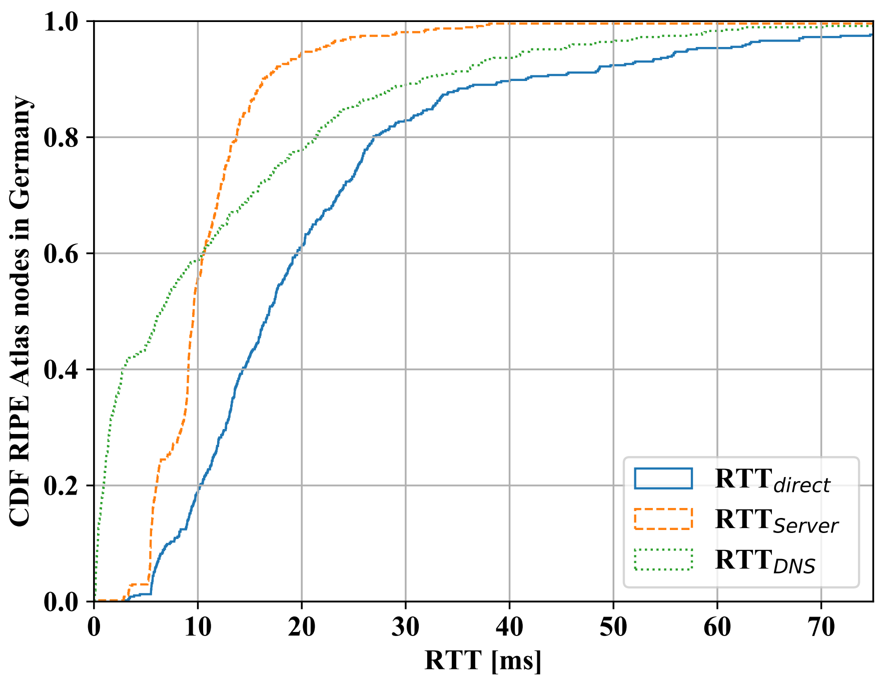


Figure 9.6: Cumulative distribution of the RIPE Atlas nodes in Germany using an ISP-provided DNS resolver over RTT_{DNS}, RTT_{Server}, and RTT_{direct}.

9.4.3 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.

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 prototype 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.

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

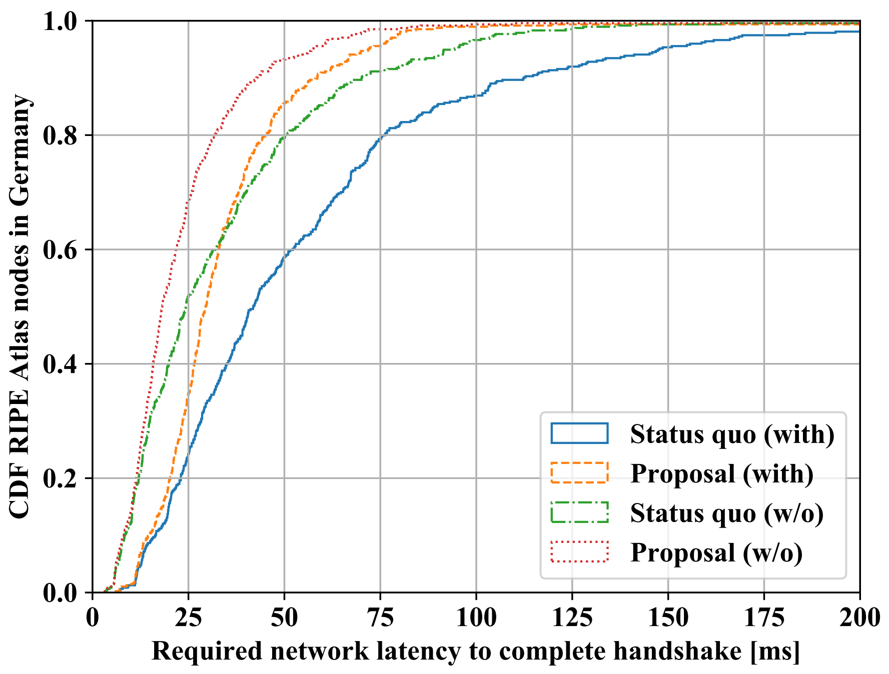


Figure 9.7: Cumulative distribution of the RIPE Atlas nodes in Germany using an ISP-provided DNS resolver over the required network latency of the QUIC handshake. The plot compares the status quo versus our proposal for handshakes with and without stateless retry.

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.

9.5 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 [4]. 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 [18] 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 [24]. Therefore, this approach seems less feasible when QUIC is used.

The ASAP [27] 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 [17]. 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 [26]. Another DNS-based mechanism aiming to reduce latency uses Server Push [6] 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.

9.6 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.

9.7 References

[1] Mike Bishop. 2019. *Hypertext Transfer Protocol Version 3 (HTTP/3).* Internet-Draft draft-ietf-quic-http-20. Internet Engineering Task Force. Work in Progress.

[2] Karl Bode. 2017. OneWeb's Low-Latency Satellite Broadband Plan Gets FCC Approval. Retrieved May 24, 2019 from https://www.dslreports.com/shownews/OneWebs-LowLatency-Satellite-Broadband-Plan-Gets-FCC-Approval-139833

[3] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. *Tor: The second-generation onion router*. Technical Report. Naval Research Lab Washington DC.

[4] Greenstein, Ben and Gao, Nancy. 2019. Chrome Lite Pages - For a faster, leaner loading experience. Retrieved May 16, 2019 from http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

[5] Seppo Hätönen, Aki Nyrhinen, Lars Eggert, Stephen Strowes, Pasi Sarolahti, and Markku Kojo. 2010. An experimental study of home gateway characteristics *(SIGCOMM '10)*. ACM, 260–266.

[6] Paul E. Hoffman and Patrick McManus. 2018. DNS Queries over HTTPS (DoH). RFC 8484. https://doi.org/10.17487/RFC8484

[7] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. 2011. Is it still possible to extend TCP? *(SIGCOMM '11)*. ACM, 181–194.

[8] Jana Iyengar and Martin Thomson. 2019. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Internet-Draft draft-ietf-quic-transport-20. Internet Engineering Task Force. Work in Progress.

[9] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. 2002. DNS performance and the effectiveness of caching. *IEEE/ACM Transactions on networking* 10, 5 (2002), 589–603.

[10] Hiroshi Kitamura. 2001. A SOCKS-based IPv6/IPv4 Gateway Mechanism. RFC 3089. https://doi.org/10.17487/RFC3089

[11] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 183–196.

[12] Marcus D. Leech. 1996. SOCKS Protocol Version 5. RFC 1928. https://doi.org/10.17487/RFC1928

[13] Walter Milliken, Trevor Mendez, and Dr. Craig Partridge. 1993. Host Anycasting Service. RFC 1546. https://doi.org/10.17487/RFC1546

[14] Jonathan O'Callaghan. 2019. Blue Origin To Launch Satellites For Company Battling SpaceX And Others For Space Internet Supremacy. Retrieved May 24, 2019 from https://tinyurl.com/y2damy3m

[15] OpenSignal. 2018. USA Mobile Network Experience Report January 2019 . Retrieved May 24, 2019 from https://www.opensignal.com/reports/2019/01/usa/mobile-network-experience

[16] Reseaux IP Europeens Network Coordination Centre. 2019. RIPE Atlas – Internet measurement network. Retrieved May 24, 2019 from https://atlas.ripe.net/

[17] Daniel Senie and Paul Ferguson. 2000. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827. https://doi.org/10.17487/RFC2827

[18] Giuseppe Siracusano, Roberto Bifulco, Simon Kuenzer, Stefano Salsano, Nicola Blefari Melazzi, and Felipe Huici. 2016. On the Fly TCP Acceleration with Miniproxy. In *Proceedings of the 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization (HotMIddlebox '16)*. ACM, New York, NY, USA, 44–49. https://doi.org/2940147.2940149

[19] Steve Souders. 2009. Velocity and the Bottom Line. Retrieved May 24, 2019 from http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html/

[20] Srikanth Sundaresan, Walter de Donato, Nick Feamster, Renata Teixeira, Sam Crawford, and Antonio Pescapè. 2011. Broadband Internet Performance: A View from the Gateway *(SIGCOMM '11)*. ACM, New York, NY, USA, 134–145. https://doi.org/10.1145/2018436.2018452

[21] Srikanth Sundaresan, Nick Feamster, Renata Teixeira, and Nazanin Magharei. 2013. Measuring and mitigating web performance bottlenecks in broadband access networks. In *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 213–226.

[22] Erik Sy. 2019. Surfing the Web quicker than QUIC via a shared Address Validation. (2019). arXiv:1903.09466

[23] Erik Sy, Moritz Moennich, Tobias Mueller, Hannes Federrath, and Mathias Fischer. 2019. Enhanced Performance for the encrypted Web through TLS Resumption across Hostnames. (2019). arXiv:1902.02531

[24] Erik Sy, Tobias Mueller, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2019. Enhanced Performance and Privacy for TLS over TCP Fast Open. arXiv:cs.CR/1905.03518

[25] Matteo Varvello, Jeremy Blackburn, David Naylor, and Konstantina Papagiannaki. 2016. EYEORG: A Platform For Crowdsourcing Web Quality Of Experience Measurements *(CoNEXT '16)*. ACM, New York, NY, USA, 399–412. https://doi.org/10.1145/2999572.2999590

[26] Ashish Vulimiri, Philip Brighten Godfrey, Radhika Mittal, Justine Sherry, Sylvia Ratnasamy, and Scott Shenker. 2013. Low Latency via Redundancy. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '13)*. ACM, New York, NY, USA, 283–294. https://doi.org/10.1145/2535372.2535392

[27] Wenxuan Zhou, Qingxi Li, Matthew Caesar, and P. Brighten Godfrey. 2011. ASAP: A Low-latency Transport Layer *(CoNEXT '11)*. ACM, New York, NY, USA, Article 20, 12 pages. https://doi.org/10.1145/2079296.2079316

[28] Liang Zhu, Zi Hu, John Heidemann, Duane Wessels, Allison Mankin, and Nikita Somaiya. 2015. Connection-oriented DNS to improve privacy and security. In *2015 IEEE symposium on security and privacy*. IEEE, 171–186.

10 Conclusion and Outlook

The enhancement of core Internet protocols is an important and ongoing task. The current decade highlighted the needs for additional privacy protections and performance improvements for these protocols. Additional privacy protections are necessary to guard users against the documented mass surveillance on the Internet [11]. With respect to the performance of web browsing, it has been realized that the users' available bandwidth does not present a bottleneck anymore. Therefore, the Internet community increasingly investigates opportunities to reduce the overhead of connection establishment to accelerate web browsing.

This thesis revises and refines the privacy protections and performance improvements of core Internet protocols. It reports tracking mechanisms within the investigated protocols and conducts several experiments to further explore the feasibility of user tracking based on the identified mechanisms. To protect the user's privacy, this thesis proposes, implements and evaluates several countermeasures, which are adapted to the protocols specific requirements. The development of these countermeasures highlighted for some tracking mechanisms a trade-off between performance and privacy. To better understand this trade-off, this thesis also studied behavioral patterns in users' online activities based on a DNS traffic data set. In the course of this research work, further performance limitations have been identified. To address these performance issues, this thesis proposes, implements and evaluates several mitigations. In total, this thesis contributes to the privacy protections and performance improvements of DNS, TCP, TLS, and QUIC.

In the following, Section 10.1 concludes this thesis and future work is reviewed in Section 10.2.

10.1 Conclusion

The results of this thesis indicate that the privacy protections of core Internet protocols can be improved. To foster these improvements, several identified tracking mechanisms have been reported to browser vendors. It contributes to the practical impact of this thesis that the TCP Fast Open protocol has been deactivated in the Firefox browser and that Microsoft Edge deactivated TFO within its private browsing mode. Furthermore, the Firefox browser is now blocking third-party tracking via TLS session resumption as a reaction to the presented research. However, not every stakeholder in the Internet community welcomes additional privacy protections. For example, the popular Google Chrome browser has not been improved by mitigating the reported tracking mechanisms. Another approach to foster the enhancement of core Internet protocols are contributions to the Internet Engineering Task Force (IETF), which develops the corresponding Internet standards. In this context, an Internet-Draft on the presented proposal of TLS resumptions across hostnames has been submitted to the IETF [25]. In total, it is a learning from the presented thesis that the impact of research work can be significantly improved by cooperating with the IETF and browser vendors. Furthermore, these cooperations allow improving the privacy protections of Internet protocols at an early stage of their development, which makes changes much easier compared to applying modofications to widely deployed protocol.

10.2 Outlook

This thesis revealed privacy flaws and performance limitations in state-of-the-art Internet protocols. However, the work on these findings is not yet completed as countermeasures need to be adopted by software vendors and applied to Internet standards to improve the real-world Internet. Furthermore, there exist many other network protocols that should be investigated with respect to their privacy protections and performance limitations. Some promising directions for future work are provided in the following:

- The feasibility of the investigated tracking mechanisms depends to a great extend on browser configurations. Thus, it seems reasonable to monitor the default configuration of popular browsers to observe changes in the feasibility of these tracking mechanisms. This practice raises the awareness for poor default settings in browsers affecting the user's privacy.

- DNS-over-TLS (DoT) [10] and DNS-over-HTTPS (DoH) [9] apply transport encryption to the traditional DNS protocol. A possible study could investigate whether these protocols make use of TCP Fast Open or TLS Session Resumption mechanisms which would facilitate user tracking through the DNS resolver.

- The proposed resolver-less DNS design can be extended by a validation mechanism using DNSSEC. This would require the web server to serve the full DNSSEC *Chain of Trust* to its client to keep the delay caused by this validation mechanism small. The feasibility of this approach still needs to be investigated as the possibly large size of the *Chain of Trust* can present a significant overhead for the communicating peers.

- A formal privacy analysis of the investigated protocols could identify additional privacy leaks in these protocols or at least ascertain the absence of such information leaks.

- Some performance optimization require the client to cache some state across several connections such as a public key to accelerate future connection establishments. However, the client has difficulties to validate that a large group of users received the same public key or whether the client received a personalized key for the purpose of user tracking. Here, the design of a mechanism would be helpful that allows a large group of users to validate that they did not receive personalized keys.

- The Network Time Security (NTS) protocol [7] is the proposed replacement for the Network Time Protocol (NTP) [14]. Time protocols are part of the Internet infrastructure. Some security mechanisms have an expiry time such as TLS certificates. Thus, tampered time information can lead to significant security risks. Future work could investigate the feasibility of user tracking via NTS server. At least this could allow to monitor the IP addresses used by a client. Subsequently, this information could be used to assign location information to the client based on publicly available databases mapping locations to IP addresses.

- TCP Fast Open and QUIC make use of address validation tokens to save a round-trip time during the validation of the clients source address. An alternative design of these tokens could omit the client's IP address encoded in these address validation tokens. Therefore a privacy-friendly validation token would only allow to approve that a previous connection between the peers existed. If the server is not experiencing a denial-of-service attack, this weaker validation mechanism may be sufficient for the server to accept a client request for an abbreviated connection establishment.

Bibliography

[1] Richard Baskerville. 1993. Information systems security design methods: implications for information systems development. *ACM Computing Surveys (CSUR)* 25, 4 (1993), 375–414.

[2] Tomasz Bujlow, Valentín Carela-Español, Josep Solé-Pareta, and Pere Barlet-Ros. 2015. Web tracking: Mechanisms, implications, and defenses. *arXiv preprint arXiv:1507.07872* (2015).

[3] Thomas Callahan, Mark Allman, and Michael Rabinovich. 2013. On modern DNS behavior and properties. *ACM SIGCOMM Computer Communication Review* 43, 3 (2013), 7–15.

[4] Juan Miguel Carrascosa, Jakub Mikians, Ruben Cuevas, Vijay Erramilli, and Nikolaos Laoutaris. 2015. I always feel like somebody's watching me: measuring online behavioural advertising. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM, 13.

[5] Steven Englehardt and Arvind Narayanan. 2016. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1388–1401.

[6] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. 2017. Measuring HTTPS adoption on the web. In *26th USENIX Security Symposium*. 1323–1338.

[7] Daniel Fox Franke, Dieter Sibold, Kristof Teichel, Marcus Dansarie, and Ragnar Sundblad. 2019. *Network Time Security for the Network Time Protocol*. Internet-Draft draft-ietf-ntp-using-nts-for-ntp-20. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-ntp-using-nts-for-ntp-20 Work in Progress.

[8] Gertjan Franken, Tom Van Goethem, and Wouter Joosen. 2018. Who left open the cookie jar? a comprehensive evaluation of third-party cookie policies. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 151–168.

[9] Paul E. Hoffman and Patrick McManus. 2018. DNS Queries over HTTPS (DoH). RFC 8484. https://doi.org/10.17487/RFC8484

[10] Zi Hu, Liang Zhu, John Heidemann, Allison Mankin, Duane Wessels, and Paul E. Hoffman. 2016. Specification for DNS over Transport Layer Security (TLS). RFC 7858. https://doi.org/10.17487/RFC7858

[11] Susan Landau. 2013. Making Sense from Snowden: What's Significant in the NSA Surveillance Revelations. *IEEE Security & Privacy* 4 (2013), 54–63.

[12] Ben Laurie, Adam Langley, and Emilia Kasper. 2013. Certificate Transparency. RFC 6962. https://doi.org/10.17487/RFC6962

[13] Timothy Libert. 2015. Exposing the Invisible Web: An Analysis of Third-Party HTTP Requests on 1 Million Websites. *International Journal of Communication* 9 (2015), 18.

[14] Jim Martin, Jack Burbank, William Kasch, and Professor David L. Mills. 2010. Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905. https://doi.org/10.17487/RFC5905

[15] Matthias Marx, Erik Sy, Christian Burkert, and Hannes Federrath. 2018. *Anonymity Online – Current Solutions and Challenges*. Springer International Publishing, Cham, 38–55. https://doi.org/10.1007/978-3-319-92925-5_4

[16] Łukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Diaz. 2015. The leaking battery. In *Data Privacy Management, and Security Assurance*. Springer, 254–263.

[17] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website Fingerprinting at Internet Scale.. In *NDSS*.

[18] Andreas Pfitzmann and Marit Hansen. 2010. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. (2010).

[19] Venkataraman Ramesh, Robert L Glass, and Iris Vessey. 2004. Research in computer science: an empirical study. *Journal of systems and software* 70, 1-2 (2004), 165–176.

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

[21] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. 2018. *Encrypted Server Name Indication for TLS 1.3*. Internet-Draft draft-ietf-tls-esni-02. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-02 Work in Progress.

[22] Steve Souders. 2009. Velocity and the Bottom Line. Retrieved May 24, 2019 from http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html/

[23] Erik Sy. 2019. Enhanced Performance and Privacy via Resolver-Less DNS. *Preprint arXiv:1908.04574* (2019). http://arxiv.org/abs/1908.04574

[24] Erik Sy. 2019. Surfing the Web quicker than QUIC via a shared Address Validation **(Best Student Paper Award)**. In *The 27th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2019)*. Split, Croatia.

[25] Erik Sy. 2019. *TLS Resumption across Server Name Indications for TLS 1.3*. Internet-Draft draft-sy-tls-resumption-group-00. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-sy-tls-resumption-group-00 Work in Progress.

[26] Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2018. Tracking Users Across the Web via TLS Session Resumption **(Outstanding Paper Award)**. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC '18)*. ACM, New York, NY, USA, 289–299. https://doi.org/10.1145/3274694.3274708

[27] Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2019. A QUIC Look at Web Tracking. *Proceedings on Privacy Enhancing Technologies* 2019.3 (2019), 255 – 266. https://content.sciendo.com/view/journals/popets/2019/3/article-p5.xml

[28] Erik Sy, Christian Burkert, Tobias Mueller, Hannes Federrath, and Mathias Fischer. 2019. QUICker Connection Establishment with Out-Of-Band Validation Tokens. In *2019 IEEE 44th Conference on Local Computer Networks (LCN) (LCN 2019)*. Osnabrück, Germany.

[29] Erik Sy, Moritz Moennich, Tobias Mueller, Hannes Federrath, and Mathias Fischer. 2019. Enhanced Performance for the encrypted Web through TLS Resumption across Hostnames. *Preprint arXiv:1902.02531* (2019). http://arxiv.org/abs/1902.02531

[30] Erik Sy, Tobias Mueller, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2020. Enhanced Performance and Privacy for TLS over TCP Fast Open. *Proceedings on Privacy Enhancing Technologies* 2020.2 (2020), Accepted for publication.

[31] Erik Sy, Tobias Mueller, Matthias Marx, and Dominik Herrmann. 2018. AppPETs: A Framework for Privacy-preserving Apps. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC '18)*. ACM, New York, NY, USA, 1179–1182. https://doi.org/10.1145/3167132.3167415

[32] Erik Sy, Tobias Mueller, Moritz Moennich, and Hannes Federrath. 2019. Accelerating QUIC's Connection Establishment on High-Latency Access Networks. In *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC) (IPCCC 2019)*. London, United Kingdom (Great Britain).

[33] Gareth Tyson, Shan Huang, Felix Cuadrado, Ignacio Castro, Vasile C Perta, Arjuna Sathiaseelan, and Steve Uhlig. 2017. Exploring http header manipulation in-the-wild. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 451–458.

[34] Narseo Vallina-Rodriguez, Srikanth Sundaresan, Christian Kreibich, and Vern Paxson. 2015. Header enrichment or ISP enrichment?: Emerging privacy threats in mobile networks. In *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*. ACM, 25–30.

[35] Matteo Varvello, Jeremy Blackburn, David Naylor, and Konstantina Papagiannaki. 2016. Eyeorg: A platform for crowdsourcing web quality of experience measurements. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. ACM, 399–412.

[36] Liang Zhu, Zi Hu, John Heidemann, Duane Wessels, Allison Mankin, and Nikita Somaiya. 2015. Connection-oriented DNS to improve privacy and security. In *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 171–186.

Final declaration

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

I hereby declare, on oath, that I have written this dissertation by my own and have not used other than the acknowledged resources and aids.

Hamburg, den ____________ ______________________________

Erik Sy