Enhanced Performance and Privacy for TLS over TCP Fast Open

Erik Sy
University of Hamburg

Tobias Mueller
University of Hamburg

Christian Burkert
University of Hamburg

Hannes Federrath
University of Hamburg

Mathias Fischer
University of Hamburg

ABSTRACT

Small TCP flows make up the majority of web flows. For them, the TCP three-way handshake represents a significant delay overhead. The TCP Fast Open (TFO) protocol provides zero round-trip time (0-RTT) handshakes for subsequent TCP connections to the same host. In this paper, we present real-world privacy and performance limitations of TFO. We investigated its 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 about 40% of the time due to web server load-balancing. Our analysis further reveals significant privacy problems in the protocol design and implementation. Network-based attackers and online trackers can exploit these shortcomings to track the online activities of users. As a countermeasure, we introduce a novel protocol called TCP Fast Open Privacy (FOP). It overcomes the performance and privacy limitations of TLS over TFO by utilizing a custom TLS extension. TCP FOP prevents tracking by network attackers and impedes third-party tracking, while still allowing for 0-RTT handshakes as in TFO. As a proof-of-concept, we have implemented the proposed protocol. Our measurements indicate that TCP FOP outperforms TLS over TFO when websites are served from multiple IP addresses.

CCS CONCEPTS

• **Security and privacy**; • **Networks** → *Network protocols*; *Network protocol design*;

KEYWORDS

TCP Fast Open, Online Tracking, Protocol Design

1 INTRODUCTION

TCP is the standard protocol for transmitting information on the Internet. The web is moving towards an encrypted ecosystem with about 80% of the client requests deploying encryption [11]. A popular web page requires an average of about 20 TLS over TCP connections to several hosts [32]. Hence, 20 TCP and TLS handshakes are required for initiating these connections. Especially for short web flows, this handshake delay represents significant overhead.

The TCP Fast Open Protocol (TFO) [7] is adopted by most operating systems and browsers, even though it is not yet actively used by all of them. It works by abbreviating TCP's three-way handshake for all consecutive connections between a client-server pair. As such, TFO requires an initial full TCP handshake to let the server verify the client IP and to exchange an identifier as proof of this verification for all successive TFO connections. However, while providing significant speedups, this identifier can be used to link TFO sessions and thus be exploited by an online tracker to collect profiles of the browsing behavior of users. Furthermore, as TFO messages are unencrypted, it also enables user tracking through passive network monitoring like dragnet surveillance. Tracking via the TFO protocol is also independent of traditional tracking practices utilizing HTTP cookies or browser fingerprinting [8]. Thus, existing countermeasures against conventional tracking do not protect against tracking via TFO.

To balance the legitimate needs of online privacy and faster TLS over TCP connections, we propose the TCP Fast Open Privacy (TCP FOP) protocol as main contribution of this paper.

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 this tracking mechanism.
- We found that under real-world conditions, the first revisit of a website supporting the TFO protocol fails, on average, approximately 40% of the time to perform an abbreviated handshake. The reason for this is mainly server load balancing, i.e., the same website is concurrently served from numerous different IP addresses which is a considerable performance limitation of TFO.
- We investigate the configuration of the TFO protocol as utilized by popular browsers. We observe that tracking periods for the Chrome, Firefox, and Opera browsers 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 on the same OS. Online trackers can utilize the presented mechanism to track users as a third-party across multiple websites, and to track users across visits to 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 conducting abbreviated handshakes for website revisits independently of the IP address of the server, which significantly improves the performance of website revisits compared to TLS over the TFO protocol. 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 the real-world feasibility of our proposal. The evaluation of our prototype indicates no additional delay compared to TFO/TLS connection establishments.

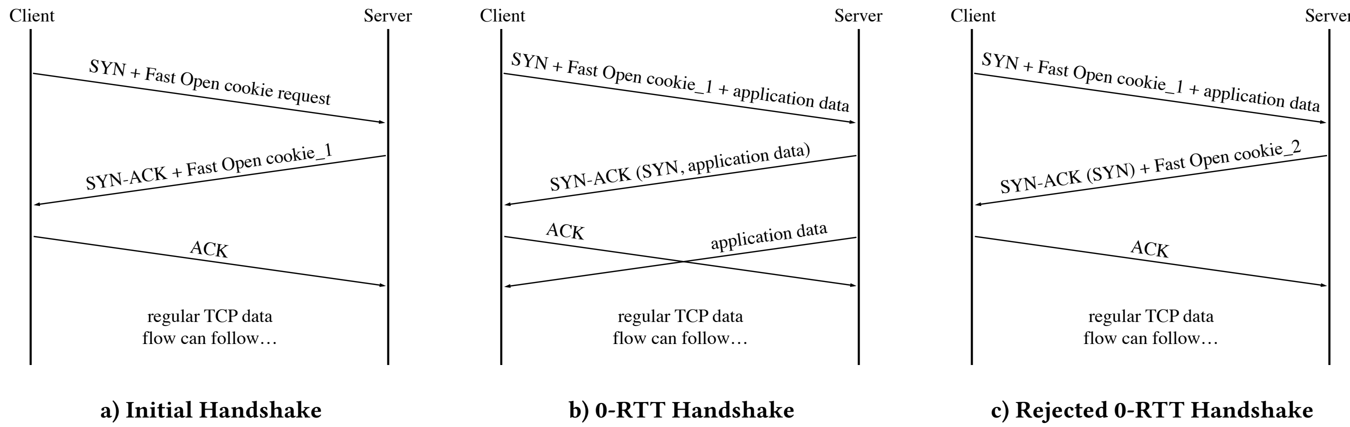


Figure 1: Handshakes in TCP Fast Open protocol.

Note that we responsibly disclosed the presented 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].

The remainder of this paper is structured as follows: Section 2 describes the connection establishment of the TCP Fast Open protocol, its deployment within the Alexa Top Sites lists, and evaluates its real-world performance limitations. Section 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 4 summarizes TCP FOP, its implementation, and presents evaluation results. Related work is reviewed in Section 5, and Section 6 concludes the paper.

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 within the Alexa Top Million Sites. In this process, we also analyze the performance impact of real-world load-balancing on the rate of zero round-trip time (RTT) connection establishment.

2.1 Background on TFO's Connection Establishment

TFO is defined in RFC 7413 [7] as an experimental TCP mechanism. It allows saving up to one round-trip time compared to the standard TCP handshake [23]. It differs from standard TCP in so far, that during a successful TFO handshake the client caches a cookie, which is presented by the client in subsequent connections. The TFO cookie is encrypted and authenticated by the server and opaque to the client. It contains among others information about the client's publicly visible IP address. Thus, if the client is able to present a cookie which matches its publicly visible IP address, the server can assume that IP source address spoofing is unlikely by this client. For these matching IP addresses, the server does not validate the client's source address via an additional message exchange. This abbreviation allows the client to establish a connection without waiting for the server response. Thus, application data can be sent immediately along with the first client message. Figure 1 shows a schematic of the TFO handshakes.

Initial handshake: At the beginning, the client has no information about the server. Like in a TCP three-way handshake [23] the client initiates a connection by sending a SYN to the server as shown in Figure 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 as an opaque data block for the establishment of subsequent connections. The server is now waiting for the client to confirm the SYN-ACK. For that, the client sends an ACK that completes the three-way handshake. The now established connection is now a standard TCP connection.

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 1b and c. Additionally, the client can include application data as a payload within the SYN message. Upon receiving the client's connection request, the server validates the included TFO cookie.

If the cookie is valid, then 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 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 case of an invalid cookie, the server drops the application data of the client as shown in Figure 1c. In doing so, 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 1c). Note, that a rejected 0-RTT handshake only incurs the same delay as a standard TCP three-way handshake.

2.2 Evaluation

In this section, we first investigate the deployment of TFO within the Alexa Top Million Sites. With respect to user tracking via TFO, 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 websites within the Alexa Top Million, we then investigate to which extent changing server IP addresses affect the performance gains that are achievable by TFO.

2.2.1 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. RFC 7413 that describes the TFO protocol was published in December 2014 [7]. 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 package containing a TFO cookie request as shown in Figure 1 a) to each of these sites. If the respective website 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 the standard port for HTTPS web services [24]. We conducted this measurement on the 10th of August 2018 with a dedicated Python script.

Table 1: Websites with TFO-support in Alexa Top lists

Alexa Top lists	Share of websites 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 1 shows the number of websites supporting the TCP Fast Open protocol within different Alexa Top lists. We find that 60% of the ten most popular websites support this protocol. However, this fraction decreases with the size of the Alexa Top list. While 28% among the Top Hundred websites 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, which leads to a gap in the deployment share between the different Alexa Top lists.

2.2.2 Performance Limitations of TFO. Revisits of a website are not necessarily served from the same IP address. 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 website is served from a different IP address the client experiences a cache miss even if a Fast Open cookie from that website is stored in the TCP cache. To assess the performance impact of this TCP Fast Open design, we observe the IP addresses of the responding servers while visiting a website several times. We conducted this measurement from an IPv4 address on August 24th, 2018. In total, we connected to 32 099 websites that we identified in our previous measurement as sites supporting the TFO protocol. We visited these sites ten times within intervals of 45 minutes between each successive visit and observed the respective server's IP addresses. About 94% of these websites indicated, at each of these visits, support for the TFO protocol. The remaining 6% can be attributed to websites that are served from multiple servers and not all of them support the TFO protocol. Figure 2 shows for these 30 218 websites the number of observed IP addresses per site. Our results indicate, that at least 81% of the tested websites are served from several IP addresses. We also find, that the second visit of a website introduces 11 876 new IP addresses compared to the first visit. As shown in Figure 3, the first TFO revisit of a website fails with a chance of 39.3% due to the fact that the website is served from a different IP address. For the third visit of a site, we observe an average failure rate of 24.7%. On average, the tested websites 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.

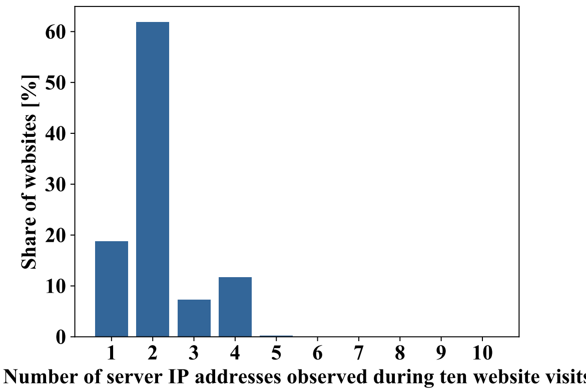


Figure 2: Share of websites plotted over the number of observed server IP addresses for ten website visits.

3 TRACKING VIA TCP FAST OPEN

In this section, we begin by introducing a method which allows user tracking via the TFO protocol. Then, we describe details on tracking through host-based and network-based attackers via this protocol. Afterwards, we point out limitations of the user identification via Fast Open cookies. Finally, we evaluate the default configuration of popular web browsers regarding TFO to assess the real-world impact of the presented tracking mechanism.

3.1 User Identification via Fast Open Cookies

The TFO protocol utilizes cookies to authenticate a client upon consecutive visits of a web service. These cookies are generated by the server and allow to identify clients with up to 16 bytes of entropy [7]. Furthermore, they allow the server to link all connections to the same client, where a unique cookie was used. Moreover,

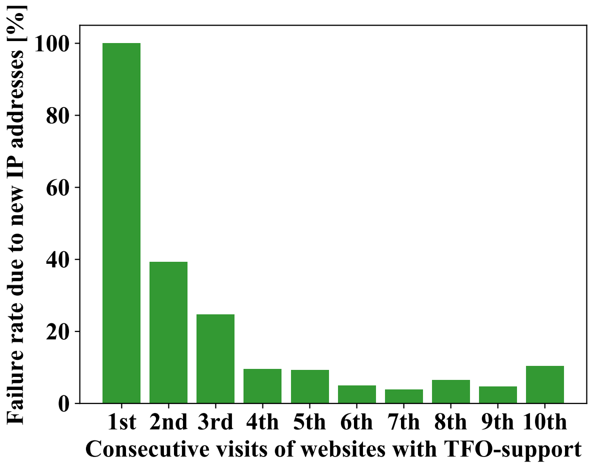


Figure 3: Failure rate to conduct abbreviated handshakes due to fresh server IP addresses plotted over successive visits of the websites with TFO-support within Alexa Top Million.

a failed authentication as shown in Figure 1c allows the server to link the fresh *cookie_2* to the same client that was previously using *cookie_1*. Following the specification, the client should then use the fresh *cookie_2* for subsequent connections to the same server.

While the specification of TCP Fast Open does not discuss its possible impact on the client's privacy, it remains unclear so far whether and how the implementations address the mitigation of the following privacy risks.

3.1.1 Length of Tracking Periods. The aggregation of long-term user profiles is facilitated if a tracking mechanism enables long tracking periods. Furthermore, tracking can become impracticable when the tracking period of a mechanism is too short and potentially expires before a user revisits the tracking server. Hence, the feasible tracking duration is a significant characteristic of tracking mechanisms. Tracking periods via TFO are limited by the uptime of an OS because 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. Furthermore, the TCP cache has a limited size, which under certain circumstances leads to the replacement of old entries through fresh ones [2].

3.1.2 Linking several Applications to the same User. Popular operating systems such as Windows 10 [14], Linux [25] and macOS [5] implement TCP Fast Open within their kernels. They also provide a cache to store information about TCP connections such as the described Fast Open cookies. This TCP cache is shared across all user space applications. As a result, applications $\mathcal{A}$ and $\mathcal{B}$ on the same OS instance will possibly use the same Fast Open cookie when communicating with the same online service. Thus, by using TFO tracking, a tracker can link several applications on the same OS instance to one client.

3.2 Tracking by a Host-based Attacker

Tracking through a single visited website presents a privacy issue. However, these privacy risks are amplified if a tracker can identify a user across several visited websites. In the remainder of this section, we describe third-party tracking and tracking across virtual domains which both allow tracking users across multiple websites via the TFO protocol.

3.2.1 Third-party Tracking. Third-party tracking refers to a practice, where a party, other than the targeted website, can track a user's visit. This is a common practice on the Internet where Google's software, alone, is present on approximately 80% of the Alexa Top Million Sites [9]. Furthermore, previous measurements indicate that a website within the Alexa Top 500 categories includes on average 17.7 third-party trackers [9]. The presented tracking mechanism allows identifying users across all websites where a corresponding tracker is included as a third-party. However, to distinguish the various first-party sites that a user visited, the tracker requires an additional identifier such as an HTTP referrer or a custom URL per first-party.

3.2.2 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 website $\mathcal{A}$ and $\mathcal{B}$ 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 can link visits of the same user across the hosted virtual domains.

3.3 Tracking by a Network-based Attacker

TCP provides an unencrypted connection. Thus, a network-based attacker can observe the content of TCP headers and can link observed Fast Open cookies to specific clients as a tracking web server can do. This undermines the efforts of protocols such as TLS 1.3 [27] that aim to protect against tracking by a network-based attacker. As a result, a tracker would simply use Fast Open cookies to identify users for TLS sessions that all run over TFO.

3.4 Limitations of the Cookie-based User Identification

Fast Open cookies are part of the TCP header and therefore transmitted over the network in clear text and without any protection of integrity. Consequently, a network-based attacker is able to manipulate them. Neither TCP nor the Fast Open extension provide a mechanism that allows the client to verify the integrity of received cookies. Furthermore, a network-based attacker can replay an observed Fast Open cookie to establish a connection with a 0-RTT handshake. Consequently, Fast Open cookies do not assure the identity of a client in the presence of a network-based attacker.

3.5 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 presented in this section.

3.5.1 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 behaviour against illegitimate online tracking.

In our sample of popular web browsers, we included the Top 3 mobile [29] and the Top 6 desktop browsers [28]. 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 under several operating systems as shown in Table 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.

3.5.2 Feasible tracking periods. This measurement studies the 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 pauses between consecutive website visits, for pauses of 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 as shown in Table 2 for the entire test period of ten days. Furthermore, this indicates that the tested operating systems do currently not apply restrictions for feasible 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, that within the Windows 10 default configuration, 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 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.

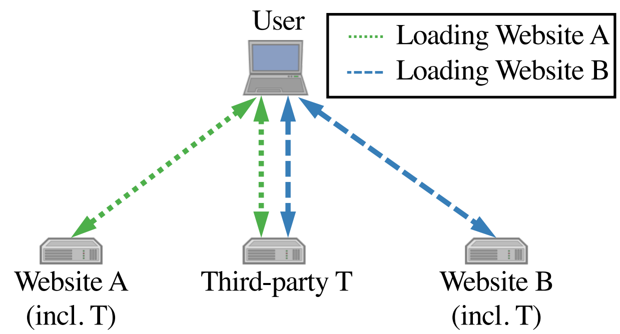


Figure 4: Testbed to measure browser behavior regarding third-party tracking.

3.5.3 Tracking across third-parties. This measurement investigates the feasibility of third-party tracking via the TFO protocol. Figure 4 shows a schematic of the deployed test setup for this experiment. To conduct this measurement, we require two websites $\mathcal{A}$ and $\mathcal{B}$ that include the same third-party $\mathcal{T}$. We visited website $\mathcal{A}$ and validated that the browser received a Fast Open cookie from the third-party $\mathcal{T}$ 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 $\mathcal{B}$ and investigated the network traffic between the browser and the third-party $\mathcal{T}$. If the browser attempted to use the cached Fast Open cookie for the connection establishment with $\mathcal{T}$, we could conclude that third-party tracking via TCP Fast Open is feasible with this browser.

As shown in Table 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.

3.5.4 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 afterwards 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 2). Therefore, when multiple websites are served from the same IP address, the service operator can identify users across those hosted websites.

3.5.5 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 afterwards. 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

Table 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 where 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.

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.

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 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.

3.5.6 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 switched to the private browsing mode. 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 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].

3.5.7 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 (see Table 2).

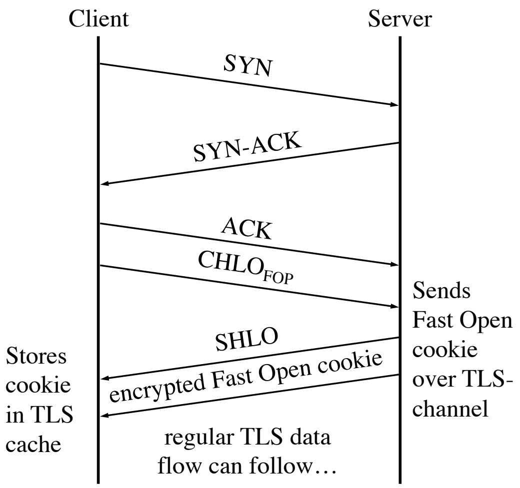
3.5.8 Tracking across user applications. This experiment explores tracking across different user applications on the same device. To conduct the experiment, we retrieve a website and leave the respective browser idle in the background of the OS. Afterwards, we use another application with support for 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 2).

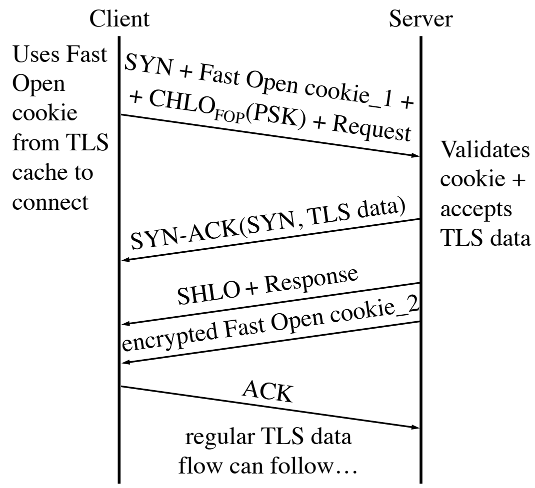
Summary. As a summary of these browser measurements, we find that the use of the TFO protocol leads to great privacy risks such as unrestricted tracking periods and cross-browser tracking. Furthermore, our results indicate that this tracking mechanism is very persistent and cannot be terminated by browser 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.

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.



a) Initial Handshake



b) 0-RTT Handshake using TLS Session Resumption

Figure 5: Proposed handshakes in TCP Fast Open protocol using TLS 1.3 as a secure channel.

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. 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 to attempt abbreviated handshakes for matching hostnames independently of the server's IP address. This modification allows TCP FOP to anticipate 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 usage of the cached cookie is only allowed for 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 are describing the details of the TCP FOP handshakes. Figure 5 shows a schematic of the proposed handshakes that use TLS version 1.3 [27] 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. Afterwards, 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.

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 5b. Subsequently, the client starts its TLS handshake without waiting for the server's response. Figure 5b shows a 0-RTT TLS handshake using session resumption for connection establishment and directly sending an encrypted data request. 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 entire client's message. Otherwise, the server acknowledges the connection only and drops the TLS data that leads to a rejected 0-RTT handshake. Following the flow of an accepted 0-RTT handshake attempt as shown in Figure 5b, the server's TLS application validates the client's session resumption data. Assuming these data to be valid, the server

Table 3: Kernel API changes

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.

answers by sending a $SHLO_{TLS}$, a response to the client's request and a fresh Fast Open cookie. By following this protocol flow (see Figure 5b), the client-server pair can establish connections that cover the transport and cryptographic handshake with 0-RTT.

Note, that in TLS version 1.3 [27] 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 to set up several connections.

4.2 Implementation of TCP FOP

In order to assess the feasibility of TCP FOP, we implemented it on top of the Linux 4.18 kernel as well as the wolfSSL TLS library. Our implementation required only minor modifications totaling about 300 lines of code (LoC) for Linux and 400 LoC for wolfSSL, including comments and debug output.

4.2.1 Kernel support. The Linux Kernel 4.18 already supports TCP Fast Open. Thus, we largely reused the available TCP Fast Open implementation for our demonstration, and as a noteworthy change, we added two new APIs to the kernel as shown in Table 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 the workings of TCP and TLS. We observe, that 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 which 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 delete mechanism. Thus, we conclude based upon our implementation [3], that TCP FOP introduces only lightweight modifications to TCP which lead to no external constraints of TCP's connection handling.

4.2.2 TLS support. We implemented our prototype [4] 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. We use TLS only as a data channel for the cookie and to set the cookie into 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 an extended NewTicket, 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. 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 the 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 a reuse of the same cookie.

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 previous 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.

4.3.1 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 data blocks and thus cannot be linked to a specific user. Therefore, the TCP FOP protocol prevents attackers to use Fast Open Cookies to re-identify specific users and to

Table 4: Comparison of privacy characteristics between the TCP Fast Open protocol and our TFO proposal utilising TLS as a secure channel.

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

establish user profiles. Hence, tracking via 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 [30]. 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 to use 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 to 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 in dependence of the visited party, virtual host, IP address, browsing mode, user application, and browser session logically excludes the tracking approaches observed in Section 3.5. 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, that will eventually affect the ratio of initial and 0-RTT handshakes.

Table 4 summarizes our findings from Section 3.5 for the TFO protocol and compares them to the privacy characteristics of TCP FOP. We find that the TCP FOP protocol can heal all privacy issues of TLS over TFO.

4.3.2 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 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 of the virtual machines was equipped with an Intel Xeon E5-1660 v4 CPU with 32GB of RAM and ran Debian stretch. The client and server machine were set up with 4 GB of RAM and were running an Ubuntu 18.10 with our modified Linux kernel (see Section 4.2.1) 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 4.2.2 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 used thereby an initial handshake of the corresponding TCP variant. The 0-RTT TLS resumption handshake used an abbreviated TCP connection establishment, if supported by the respective TCP

Table 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.

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)

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 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 which 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 to 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 a 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. Divertingly, the TFO/TLS is restricted to conduct 0-RTT handshakes only when the server's IP address matches the one associated to 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 [32]. 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 2.2.2 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 6 summarizes the results for successive revisits of test website. As the TCP FOP/TLS protocol stack can establish 0-RTT connections independently of the IP address associated to 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 visit at the website. As indicated in Figure 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.

4.3.3 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 [26]. Hence, even simple changes to TCP require a long time before they exhibit a significant deployment [10]. The TFO protocol faces similar problems that hinder its rapid deployment on the web [7, 21]. The presented TCP FOP embeds a standard TCP three-way handshake

Table 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 2.2.2. We assume a RTT of 60ms for the LTE connection.

	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

in the initial handshake. This standard handshake is common on the web and thus will not cause any issues with middleboxes [26]. The exchanged messages in TCP FOP's 0-RTT handshake are identical to the messages in the TFO protocol (see Figure 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.

A problem might arise with middleboxes that block TFO due to its initial handshake and not because of its 0-RTT connection establishment. These middleboxes would hinder the deployment of TFO, but not the one of TCP FOP as it uses the standard TCP three-way handshake during the initial handshake. As a result, the TCP FOP protocol achieves from an analytical perspective a better deployment on the Internet than TFO. However, we approximate the practical benefit of this modification to be small because empirical investigations indicate a success-rate of TFO connection establishments on the Internet of about 80% and these investigations mainly highlight other deployment issues of TFO [21].

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 less deployment issues than TLS over the TFO protocol.

5 RELATED WORK

To the best of our knowledge, we are the first to report on 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 mechanisms has been reported for the QUIC transport protocol [31]. However, QUIC's address validation tokens are distributed via an encrypted channel and do not allow a passive network observer to correlate different connections to the same user.

Furthermore, TLS session resumption mechanisms [30] 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 then TLS 1.3.

Additionaly, prior research includes a proposed TCP extension which directly allows to encrypt TCP packets [6]. This approach significantly increase 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 webserver 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.

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 fully protects against user tracking by network-based attackers. Furthermore, the protocol can also restrict third-party tracking and 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 evaluations the proposed TCP FOP protocol leads to substantial enhancements of the performance and privacy of TLS over TCP Fast Open.

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] Anonymous Author. 2018. Author's patch to the Linux Kernel to implement TCP FOP. Retrieved January 21, 2019 from http://paste.ubuntu.com/p/fyfTrgXmBj/

[4] Anonymous Author. 2018. Author's patch to Wolfssl to implement TCP FOP. Retrieved January 21, 2019 from http://paste.ubuntu.com/p/xJfgBh4SZW/

[5] Apple Inc. 2018. Your App and Next Generation Networks. Retrieved January 21, 2019 from https://developer.apple.com/videos/play/wwdc2015/719/

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

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

[8] Peter Eckersley. 2010. How unique is your web browser?. In *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 1–18.

[9] 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.

[10] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. 2011. Is it still possible to extend TCP?. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, 181–194.

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

[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*. ACM, 183–196.

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

[14] Microsoft. 2016. Windows 10 build 14352. Retrieved January 21, 2019 from https://developer.microsoft.com/en-us/microsoft-edge/platform/changelog/desktop/14352/

[15] Mozilla Corporation. 2018. User tracking via TCP Fast Open. Retrieved January 21, 2019 from https://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 *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 27–36.

[18] Dr. Thomas Narten, Richard P. Draves, and Suresh Krishnan. 2007. Privacy Extensions for Stateless Address Autoconfiguration in IPv6. RFC 4941. https://doi.org/10.17487/RFC4941

[19] OpenSignal. 2014. LTE Latency: How does it compare to other technologies? Retrieved January 21, 2019 from https://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 https://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, 5 (2014), 494–496.

[23] Jon Postel. 1981. Transmission Control Protocol. RFC 793. https://doi.org/10.17487/RFC0793

[24] Jon Postel and Joyce K. Reynolds. 1994. Assigned Numbers. RFC 1700. https://doi.org/10.17487/RFC1700

[25] Sivasankar Radhakrishnan, Yuchung Cheng, Jerry Chu, Arvind Jain, and Barath Raghavan. 2011. TCP Fast Open. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies (CoNEXT '11)*. ACM, New York, NY, USA, Article 21, 12 pages. https://doi.org/10.1145/2079296.2079317

[26] 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 *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 29–29.

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

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

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

[30] 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

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

[32] 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).