Hindawi
Security and Communication Networks
Volume 2019, Article ID 3107543, 16 pages
https://doi.org/10.1155/2019/3107543

Research Article

QoS3: Secure Caching in HTTPS Based on Fine-Grained Trust Delegation

Abdulrahman Al-Dailami,[1,2] **Chang Ruan**,[1] **Zhihong Bao**,[1] **and Tao Zhang**[3]

[1]*School of Computer Science and Engineering, Central South University, Changsha 410083, China*
[2]*Faculty of Computer and Information Technology, Sana'a University, Sana'a, Yemen*
[3]*School of Computer Science and Engineering, Changsha University, Changsha 410083, China*

Correspondence should be addressed to Chang Ruan; ruanchang@csu.edu.cn

Received 6 June 2019; Revised 24 September 2019; Accepted 1 November 2019; Published 28 December 2019

Academic Editor: Clemente Galdi

With the ever-increasing concern in network security and privacy, a major portion of Internet traffic is encrypted now. Recent research shows that more than 70% of Internet content is transmitted using HyperText Transfer Protocol Secure (HTTPS). However, HTTPS encryption eliminates the advantages of many intermediate services like the caching proxy, which can significantly degrade the performance of web content delivery. We argue that these restrictions lead to the need for other mechanisms to access sites quickly and safely. In this paper, we introduce QoS3, which is a protocol that can overcome such limitations by allowing clients to explicitly and securely re-introduce in-network caching proxies using fine-grained trust delegation without compromising the integrity of the HTTPS content and modifying the format of Transport Layer Security (TLS). In QoS3, we classify web page contents into two types: (1) public contents that are common for all users, which can be stored in the caching proxies, and (2) private contents that are specific for each user. Correspondingly, QoS3 establishes two separate TLS connections between the client and the web server for them. Specifically, for private contents, QoS3 just leverages the original HTTPS protocol to deliver them, without involving any middlebox. For public contents, QoS3 allows clients to delegate trust to specific caching proxy along the path, thereby allowing the clients to use the cached contents in the caching proxy via a delegated HTTPS connection. Meanwhile, to prevent Man-in-the-Middle (MitM) attacks on public contents, QoS3 validates the public contents by employing Document object Model (DoM) object-level checksums, which are delivered through the original HTTPS connection. We implement a prototype of QoS3 and evaluate its performance in our testbed. Experimental results show that QoS3 provides acceleration on page load time ranging between 30% and 64% over traditional HTTPS with negligible overhead. Moreover, QoS3 is deployable since it requires just minor software modifications to the server, client, and the middlebox.

1. Introduction

To protect users' privacy and information security, many services on the Internet such as Twitter, Facebook, and Google offer access only through HTTPS that provides data encryption. Currently, a major portion of the overall Internet traffic which already passes 72% [1] is delivered using HTTPS. In the HTTPS protocol, the TLS becomes the mainstream end-to-end encryption standard for web content transmission since it provides (1) data integrity and authenticity, (2) data privacy and confidentiality, and (3) server/client authentication. Furthermore, HTTP 2.0 by default uses TLS as encryption layer for all the web content transmissions.

However, since TLS requires all functionalities must be end-to-end between the endpoints, it leads to potential losses in both time efficiency and network utilization. In fact, the sessions on the Internet are enhanced by useful modules along the path between the endpoints, which provide in-network optimization services such as caching, parental filtering, intrusion detection, and optimizations to the web content such as data compression and transcoding. These modules, which are usually referred to as middleboxes, provide many advantages to the end-users by reducing the

page load time by 50% [2]. They also provide advantages to the Internet Service Providers (ISPs) by improving the network utilization, which can be increased by 33% [3, 4]. However, these benefits will be lost when employing TLS. As a result, the end-users should make a trade-off between performance and privacy according to the requirements of the content providers.

To enable in-network service functionalities, the middleboxes are deployed as MitM between the endpoints in the network using the Split-TLS technique [5], which is not secure. In Split-TLS, the middlebox is installed as a gateway for the client. In order to make the clients trust the middlebox, the middlebox certificate is installed on the client Trusted Root Certification Authority Store. In this case, the middlebox is trusted by the client as root Certification Authority (CA) and any certificate signed by the middlebox will be trusted and valid. The middlebox forwards the traffic back and forth between the client and the server using two separate TLS connections: one between the client and the middlebox and the other between the middlebox and the target server. Although the Split-TLS method is widely used, it has several drawbacks that come from its insecure design. The main drawback is that the clients have to fully trust the middlebox to communicate with the target server on behalf of them in a secure manner. Furthermore, the middlebox has full access to the traffic sent between the client and the target server, which means that the middlebox can read and modify the content. Consequently, the Split-TLS method breaks the end-to-end TLS encryption. For this reason, many issues have been mentioned about the growing concerns in web content privacy and security in [6].

Using in-network services with end-to-end encryption is a recent hot topic which causes several debates in the IEEE ETI [7] and ETSI MSP [8] networking communities. To enable the middleboxes to provide in-network functionality with TLS encryption, several RFC documents and academic proposals [9–14] have been published as an attempt to design a protocol that solves the problem of middleboxes with TLS encryption. The recently proposed schemes such as Blindbox [10], mcTLS [9], Embark [15], and mbTLS [11] discuss and attempt to address issues of the content privileges and permissions related to the middleboxes in the TLS session. The proposed solutions are based on extensions to the TLS as in mcTLS, providing a new encryption technique that enables the middlebox to search for specific keywords on the encrypted content as in Blindbox, or outsourcing the in-network functionalities to the cloud as in mbTLS and Embark. The schemes provide stricter privacy than Split-TLS. Industrial efforts by Google [12], Ericsson AT&T [14], and Akamai [16, 17] also try to work out solutions to combine encryption with using the in-network middlebox functionalities.

However, these proposed schemes still suffer from technical limitations and challenges. Some of those solutions incur a significant overhead to the protocol and complicate the protocol design as in mcTLS and mbTLS, while the others require modifications to the TLS format, which requires upgrading all middleboxes and endpoints to support those solutions. Nonetheless, the topic is still active within IEEE ETI and ETSI MSP working groups.

In the current approaches, to enable caching proxies with HTTPS, the client and the server should fully trust the caching proxy. In this case, the client has to accept the content served from the caching proxy without any means to validate the end-to-end integrity of the content and whether the content is the same as in the origin server or not, which results in breaking the end-to-end integrity that TLS provides.

We observe that the template of a website is public to all users and there are no privacy concerns about it, e.g., the logo of the website, CSS files, base HTML, and the javascript on the webpages. Therefore, the web content provider can implement fine-grained classification for web content by determining the private content, which can be served over direct end-to-end HTTPS connection, and public content, which can be served over a delegated HTTPS connection where the client trusts the middlebox as a proxy for this connection. In this case, the public objects can be served over a delegated HTTPS connection without concerning any privacy problems. Unlike Split-TLS, this mechanism provides a fine-grained and safe trust delegation for the caching proxy middlebox.

The challenge in sending the public content over the delegated HTTPS connection is that the web page is vulnerable to MitM attacks, and to address this, we can use secure hash values (checksums) to protect the integrity of the public content. The Subresource Integrity (SRI) [18] tag can be used to protect the embedded public content in a web page. However, SRI supports limited types of elements, such as the *<script> and <link>* elements. Web page elements such as *<img>, <embed>, <audio>, <video>*, and all the other static objects elements are not supported by SRI.

In this paper, we propose QoS3, a protocol that is built over the HTTPS to provide fine-grained trust delegation to the proxy in a secure way, which allows endpoints to reintroduce in-network caching proxies explicitly and securely without compromising the integrity of the HTTPS content and modifying the format of the TLS. Using QoS3, the client will open two separate HTTPS connections to the server. One will allow the client to delegate the trust to trusted middleboxes (caching proxies) along the path to the origin server. And the other will be directed to the origin server, without delegating the trust to any middlebox on the path. In QoS3, we propose extending an emerging technique called the SRI tag to support QoS3 content classification (public and private tags) and all the static objects types (elements) such as images and media files. QoS3 has five key features: (1) providing fine-grained trust delegation to the caching proxy by using the direct HTTPS connection and delegated HTTPS connection; (2) enabling content providers to determine which parts of the webpages are public and therefore exposed to the caching proxies; (3) providing clients with explicit control and knowledge over which middlebox (trusted proxy) is part of the session; (4) preserving the public content authenticity and integrity by means of the secure hash values (checksums); and (5) being deployable.

We have implemented the QoS3 prototype which is a simple QoS3 extension to the HTTPS clients, servers, and proxies. The experiment evaluation for the QoS3 shows that QoS3 provides a significant improvement on the page load time with negligible overhead when loading sample webpages from Alexa's top 500 websites downloaded in our local testbed. In most cases, the QoS3 integration into applications is fairly simple. Our experiments indicate that when serving public content objects especially over the high latency paths, QoS3 allows the performance-oriented middleboxes to provide improvements on the page load time ranging between 30% and 64%.

This paper has the following main contributions:

(1) An efficient fine-grained trust delegation mechanism while preserving the data integrity of the delegated session

(2) A practical extension to HTTP/TLS which allows reintroducing the trusted in-network caching proxies between the endpoints in the secure web content communications by delegating the trust to the trusted proxy for the public content only

(3) Extensions to the SRI tag to support QoS3 content classification

(4) A prototype implementation of QoS3 tested in controlled environments

The rest of this paper is organized as follows. Section 2 describes the background of HTTPS, the implications of HTTPS, and the motivation for QoS3. Section 3 provides an overview of QoS3 design space, the approach, and operating model of QoS3. Section 4 provides prototype implementation for QoS3; it also evaluates and validates QoS3 deployability and performance. Section 5 gives some discussions on QoS3. Section 6 surveys related work. Finally, in Section 7, we conclude the paper and discuss future work.

2. Background and Motivation

In this section, we describe the factors that affect the web system performance with HTTPS, introduce the vulnerabilities of the websites that use a mix of HTTP and HTTPS content, and discuss the implications of the HTTPS. Finally, we discuss the motivation of designing QoS3.

2.1. Background

2.1.1. HTTPS Overview. HTTPS naturally uses TLS protocol for communications encryption. TLS is a standard protocol that provides confidentiality and authenticity over TCP connections. Currently, TLS is used to provide security to traditional protocols such as SMTP, XMPP, and IMAP; particularly, using HTTP over TLS is often referred to as HTTPS. The client and the server start every TLS connection with a handshake. In this handshake, the client authenticates the server by using the Public Key Infrastructure (PKI) and sometimes the server also authenticates the client using the same mechanism. The PKI is also used for key exchange in order to create a secure connection between the endpoints.

Generally, requesting a web page resource over HTTPS involves long steps of processing in the HTTPS protocol stack. Figure 1 shows the steps that the client follows in order to fetch a resource from the web server. First, a TCP connection is established between the client and the server. This will cost one RTT; then the client assumes that the server is HTTP, not HTTPS, by sending an HTTP GET request to the server. The server will respond with 302 response to inform the client that this server is HTTPS and the client must establish a TLS session for secure HTTP transmission. This will cost one RTT. The client then establishes a new TCP connection to the server using the 443 TLS port, and this will cost additional one RTT. Subsequently, the server and the client establish TLS handshake that costs two RTTs for key exchange and three RTTs for certificate verification from the CA. The last RTT is for HTTP object request. To sum up, fetching web object from web server using HTTPS in the worst case costs 8 RTTs: 2 RTTs for the initial HTTP connection, 1 RTT for second TCP connection over TLS port 443, 2 RTTs for TLS key exchange, and 3 RTTs for certificate verification by fetching Online Certificate Status Protocol (OCSP) records from the CA, as shown in Figure 1.

2.1.2. HTTPS Implications. Although HTTPS and TLS bring both advantages and disadvantages simultaneously, the advantages are noticeable but little attention is paid to their drawbacks. In this section, we accentuate the constraints of TLS protocol and concentrate on the privacy and performance implications of HTTPS.

First, all web page objects are blindly fetched over HTTPS, without any distinction of the privacy or size of the objects. In most times, these objects cause worthless performance overheads. For instance, as shown in recent research, the handshake of the TLS protocol takes over 42% of the exchanged data in the HTTPS websites [6].

Second, HTTPS stops middleboxes, such as transponders and caches, from checking packets and enhancing end-user performance. As a result, the traffic on the ISP's links will be increased, and therefore the web page load time increases.

Third, many websites carelessly apply the HTTP Strict Transport Security (HSTS) [19] as a way for deploying HTTPS automatically. Yet, more than 59% of websites deploy HSTS incorrectly which in turn cause flaws in security, such as improperly setting the *max-age* parameter or redirecting the request to the HTTP domain. Unexpectedly, the websites that correctly deploy HSTS are also exposed to several types of attacks because of stale policies or timestamp modifications [20].

2.1.3. Mixed Content Websites and MitM Attacks. To overcome HTTPS limitations and improve page load time, content providers provide website content using HTTP and HTTPS. They provide the cacheable static content using HTTP and the other using HTTPS; this is called a mixed content website. Currently, websites with mixed content are not supported by many web browsers [20]. Websites that contain mixed content are not encouraged because they

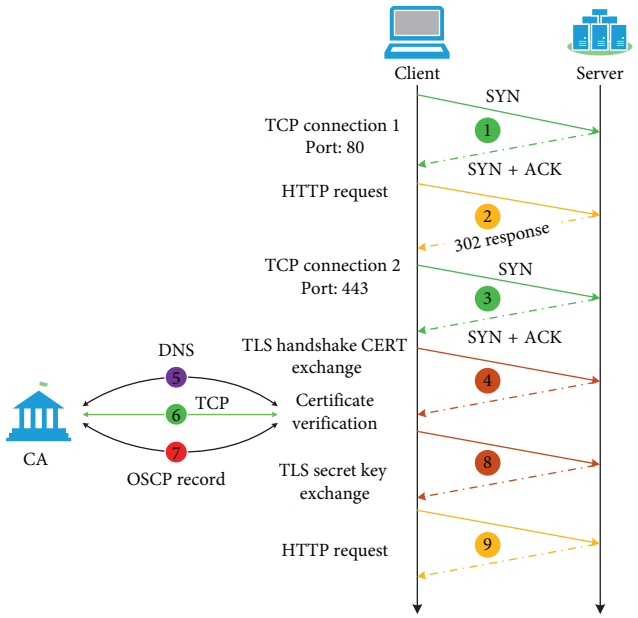

Figure 1: The cost of HTTPS connection.

make the websites vulnerable to MitM attacks. Mixing both HTTPS and HTTP content indicates that the content that served through the HTTP connection will be plain/unencrypted and therefore not protected. The HTTP connections could be easily intercepted by the attacker. The attacker then is able to inject malicious code to the intercepted content and send the compromised content to the user. The attacker can use the hacked content to steal the user passwords which are stored in the cookies and other information. Furthermore, the attacker can replace the web page DoM objects by others and therefore change the appearance of the web page. For example, an attacker can perform a MitM attack on the HTTP connection to steal the user information stored in the cookies by intercepting the content served over HTTP and inject the malicious code into the original content. Then, the content is sent to the user which allows the attacker to compromise the user's computer [21].

2.2. Motivation

2.2.1. Ignoring Caching Proxies with Previous Approaches. In order to overcome the shortcomings of HTTPS, several optimization research works are proposed, and new standards have emerged. HTTP 2.0 protocol solves many problems at the level of application layer such as Head of Line (HoL) Blocking [22]. HTTP 2.0 now supports one connection with multiple HTTP streams multiplexed over a single TCP connection. Besides, HTTP 2.0 is a binary protocol that makes it faster when processing HTTP requests and responses.

At the TLS level, several optimization works are also proposed such as TLS false start [23] and TLS session caching using Session ID [24] to eliminate one RTT from the TLS handshake. The latest is the TLS 1.3 [25], in which the TLS connection establishment costs only one RTT. OCSP stapling addresses the latency of OCSP record verification process [26]. At the TCP layer, the TCP fast open allows sending the ClientHello message within the TCP SYN packet that will reduce one RTT [27]. Recently, many new transport protocols such as [28–31] are proposed, of which QUIC [28] supports connections by multiplexing over User Datagram Protocol (UDP). We find that most of the research efforts focus on improving web page delivery through optimizing the cost of HTTPS connection setup-related problems such as minimizing the number of RTTs. On the other hand, we claim that there is a valuable entity in the networks whose functions are prevented by HTTPS, e.g., caching proxies.

There are several recent studies [32–35] indicating the importance of middleboxes (specifically caching proxies) in wired networks and cellular networks. These middleboxes have proved that it can add great advantages to both the end-users and the networks operators. These advantages greatly motivate us to propose the QoS3 system in order to maintain these middleboxes' functionalities.

2.2.2. Insecure Web Content Delivery with Previous Approaches. At present, the web services are provided using two different ways: the first has no privacy, at which the web page content is delivered over the insecure plain HTTP [36]; and the second guarantees the confidentiality and privacy, at which the web page content is delivered over HTTPS.

One approach is to use Split-TLS [5] with Intercepting Proxy "which enables the proxy to inspect and modify traffic between the browser and the target application" [37], operating as a MitM between the end-user and the origin web server. The proxy tool forwards web requests with the ability to have full control over any requests such as modifying or dropping them.

Another approach is that one could encrypt web page content transferred between the client and the server with different keys and share selected subgroup of the different encryption keys with the in-network middleboxes to allow middleboxes to decrypt traffic and make their optimizations. Even though this method addresses the preceding solution downsides, it imperils the forward secrecy, possibly adds latency overheads, complicates the protocol design, and requires in-network hardware upgrades [9].

The third approach is splitting content up, with content served via both HTTP and the secure HTTPS [38]. But, this approach has inconsistency with HSTS security policy. Moreover, HTTPS becomes the mainstream for web delivery, and this approach depends on plain HTTP to transfer the public content. Since plain HTTP is no longer supported by the future web delivery protocols such as HTTP 2.0 and QUIC, another way is to explicitly trust a proxy. In this approach, the proxy will terminate the connection in order to apply its functionality on the web content [14]. Unfortunately, in explicitly trusted proxy scheme, there is no integrity guarantee for the web objects sent/received via the trusted proxy server. Furthermore, in this scheme, there is no control over which content can be exposed to the trusted proxy and which content is not. In short, all the contents sent/received by the client are exposed to the middlebox, and

the middlebox has read/write access to the data. Furthermore, the client cannot detect if the content is modified and cannot verify that the content is from the origin server.

3. QoS3 System Design

In this section, we describe a new framework for re-designing the HTTPS web content transport with middleboxes such as caching proxy and introduce the novelty of QoS3 approach for enabling the secure coexistence of TLS with specific performance-oriented middleboxes (caching proxies).

3.1. Overview. QoS3 aims to allow clients to explicitly and securely re-introduce in-network caching proxies without compromising the integrity of the HTTPS content and modifying the format of TLS. To achieve better web content transport, QoS3 splits the HTTPS connections into the delegated HTTPS connection and the direct HTTPS connection and delivers the web page content over both. However, this scheme has two significant challenges:

(i) For delegated HTTPS connection, overcoming the security problems that might occur, specifically the MitM attacks discussed in Section 2

(ii) Determining which content to send via delegated HTTPS connection and which to send via the direct HTTPS connection

QoS3 tackles the two challenges by employing object-level checksums to protect the public content and tagging the web page objects as public or private using the extended SRI tags.

Concisely, in QoS3 design, we aim to make the QoS3 compatible with current infrastructure with minimal changes to the existing software/protocol. The QoS3 features should follow and satisfy the level of the security offered by the original TLS and at the same time allow the middlebox to have access to the public content in a specific HTTPS connection without enabling it to add any modifications to the public content. The client should be able to authenticate the middlebox and know that the middlebox is in the link between endpoints.

In QoS3, we focus in the application layer proxies/middleboxes such as caching proxy servers which have the capability to cache the web objects. The application layer proxies/middleboxes are typically deployed as a shared gateway for all clients in the enterprise's network or the ISP gateways. The proxies/middleboxes deployed in the gateways typically implement application layer functionality such as caching. In the case of large organization networks, a network-wide proxy server can be deployed as a gateway for the network. The proxy server can be deployed as a gateway which provides a caching service for the delegated HTTPS connections between the client and the server.

3.2. Web Page Content Classification. To determine the type of the web page object, we classify the web page content based on privacy into public and private content. The public contents are the parts of the web page which are general for all users and are not specialized or personalized for a specific user such as the images of the web page banner, JavaScripts, and the CSS files. The knowledge of the public content does not reveal any further information about the private content of the website other than the content that can be accessed through the website DNS or IP address. The public objects will be requested by the client via the delegated HTTPS connection which allows the middlebox to intercept the objects sent/received by the client and the server and store them in the middlebox cache storage. As a result, the future requests to the same object (which already stored in the proxy cache) can be served from the proxy instead of the origin server, which in turn speeds up the web page load time for the client and saves the available bandwidth for the service provider or the enterprise. The private contents are, to some extent, personalized, and they are special to each user; the cache-hit-rate for these objects is very low. An example of this type of website content is personalized images or advertisements. The private objects will be requested via the original (direct) HTTPS connection.

We perform a simple statistical analysis in order to provide a summary of the expected public and private content classification shown in Figure 2. The data are collected from the top 10 Alexa's websites. We find that all websites we have investigated contain a significant portion of objects that are public content and any one can access it, with a real number ranging between 35% and 94%. Therefore, we note that the number indicates a potential gain.

Interestingly, from the performance point of view, we note that the public portions of the website are significantly cacheable, while the private portions of the website are extremely personalized and in most cases are not cacheable. For example, the recent Facebook study demonstrated that the portion of the website content that is extremely personalized in most cases is not cacheable in the in-network cache [39]. In addition, general content, for example, images, HTML-XML, and scripts, accounts for more than 50% of the bytes in the web communications [40]. Therefore, focusing only on these features can provide significant benefits.

3.3. MitM Attack Prevention. To eliminate MitM attacks on public content, QoS3 protects and verifies the public content using one of the following approaches.

3.3.1. Digital Signature Verification. One approach is to send the public content over the delegated HTTPS connection attached with a hash value signed by the server's public key for each public resource as in self-certifying names in Information Centric Networks [41]. In this arrangement, the signed hash value provides integrity for the resource sent over the delegated HTTPS connection (i.e., the trusted proxies can read the web object's plain text, but it cannot add any modifications without the server's private key). Then, a caching proxy can act as a man in the middle for the delegated HTTPS connection, but not for the direct HTTPS connection. Since the proxy server is acting as a man in the middle for the delegated HTTPS connection, it can

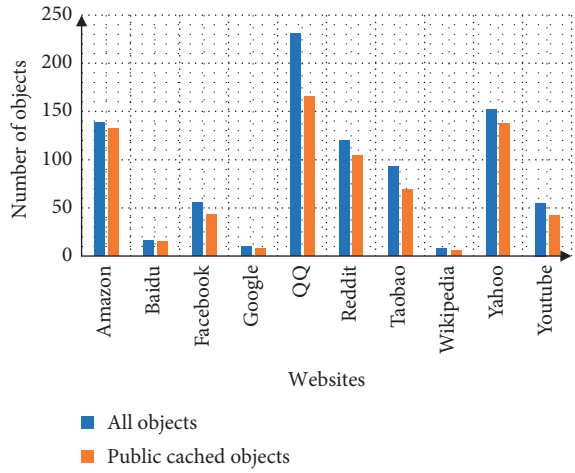

Figure 2: Top 10 websites' objects classification based on privacy.

read the web object's plain text of the delegated HTTPS connection. However, because the public web page objects are sent over this connection protected by the signed hash value and only the server has the key to regenerate the signed hash value, the proxy cannot compromise or do any modifications to the web objects.

This scheme has the following three significant challenges. First, it incurs overhead in the size of the messages more than simple checksums. More significantly, in TLS 1.3, the Integrity Only Cypher Suits have been removed [25], and the support for the integrity only cypher suits is limited nowadays. Second, in the future, this approach would require modifications to the TLS specification. Moreover, in this approach, there is a challenge about the reuse and consistency of the server's public and private keys. The public and private keys may be changed (e.g., certificate renew), and the client will not be able to verify the signed hash value attached with each public resource. Finally, there are also challenges on how to maintain the signatures on the proxy cache, which may require modifications to the caching module.

3.3.2. Checksum Verification. The second approach is to send the public content over the delegated HTTPS connection and send a hash value (checksum) for each public resource in the web page over the direct HTTPS connection, as described in Figure 3. In this procedure, the hash value (checksum) which provides integrity for the resource is sent over the direct HTTPS connection. It means that the trusted proxies can read the web object's plain text of the delegated HTTPS connection, but it cannot add any modifications because the client can detect any modifications by using the checksum received from the origin server over the direct HTTPS connection. At that point, a caching proxy can act as a man in the middle for the delegated HTTPS connection, but not for the direct HTTPS connection.

We found that the use of simple checksums is sufficient for QoS3 and it is better to follow in our design because of its simplicity and because it does not require modifications on TLS protocol and proxy cache module.

Generally, the MitM attacks are prevented by QoS3 through

(1) Generating checksums for the public objects that will be delivered via the delegated HTTPS connection.

(2) Sending these checksums via the direct HTTPS connection.

(3) Validating the public objects by comparing the checksum calculated by the client with the checksum received from the server over the direct HTTPS connection. Alternatively, we can achieve the public object validation by using a checksum signed by the server private key.

(4) Overcoming HSTS security policy by sending all data over two HTTPS connections. One is direct HTTPS between client and the server and the other is delegated connection, by which the trusted proxy is interposed between the client and the target server and applies in-network functionality (caching service) on the content.

3.4. QoS3 Architecture and Environment. In QoS3, we propose improvements to the web browsers, proxies, and the web servers in order to address the above challenges. Figure 3 shows the architecture for QoS3. In QoS3, the QoS3-enabled clients initiate two separate HTTPS connections. One is a direct connection. In this connection, the client will create a HTTPS connection to the server without trusting or delegating the trust to any middlebox along the path between the client and the server. And the other is a delegated HTTPS connection. In this connection, the client will delegate the trust to the middlebox by allowing the middlebox to terminate the TLS connection as in Split-TLS scheme. QoS3 serves end-to-end encryption by using direct HTTPS connection between the client and the server while still allowing the proxy server to cache public contents.

First, the client connects to the server using a direct HTTPS connection. Then, it sends an HTTP request for the web page. After the client receives the base HTML file, it parses the DoM tree for querying object tags. If the server supports QoS3 protocol, the server will tag the public content as public and the private content as private. If the client finds the tags, it then initiates another HTTPS connection, i.e., the delegated one. With the user agreement and by completing the TLS handshake successfully with the proxy server, the client delegates trust to middlebox for the delegated HTTPS connection.

After completion of the two TLS connections establishment, the client communicates with the server as in the original HTTPS using the direct HTTPS connection, while in the delegated HTTPS connection, the QoS3 proxy server will be able to decrypt the encrypted data and have access to the web objects' plain text. As such, the proxy server can apply its functionality (i.e., caching) on the data received from the origin server, while still preserving the logic of end-to-end encryption between the client and the server over the direct HTTPS connection. In comparison with the Split-TLS where as soon as a root certificate is installed in the client, the

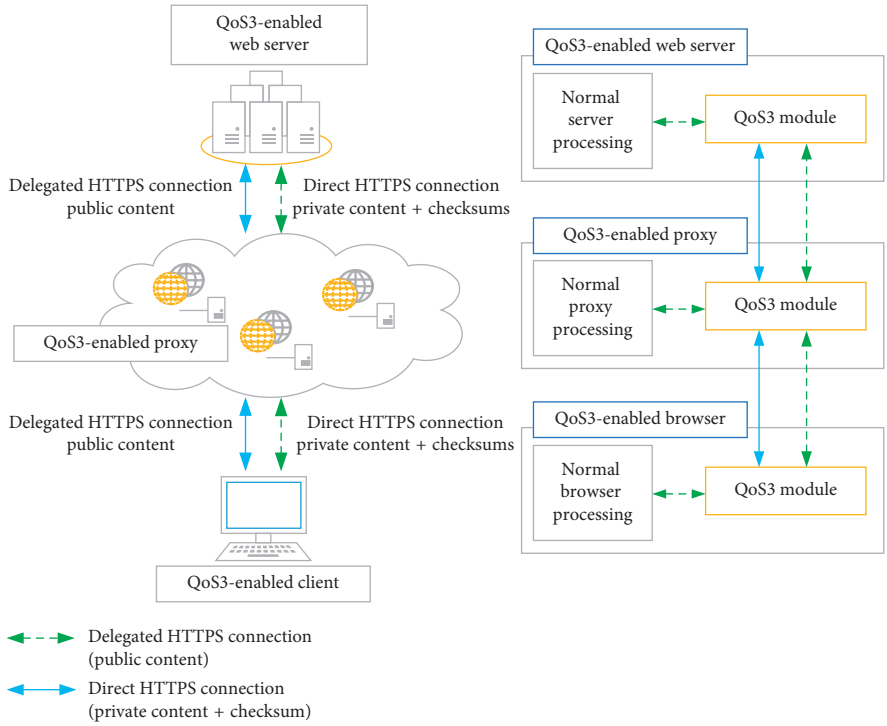

Figure 3: The framework of the QoS3 system.

middlebox has full access to the traffic flows between the client and the server for all the connections, QoS3 presents fine-grained security for the clients. Furthermore, the trust level between the client and the QoS3 proxy server is stricter than the trust level between the client and the Split-TLS proxy server. Particularly in QoS3, the client can control which content the proxy server can have access to by requesting the public content only via the delegated HTTPS connection. Furthermore, the proxy server can never modify the public web objects traffic, as the client applies checksum verification on public content for detecting any modifications made by the middlebox.

Next, we describe our scheme by discussing the client, the proxy, and the server plugin modules of QoS3 in detail.

3.5. QoS3-Enabled Web Servers. A QoS3 web server is like the conventional web server excluding the following functions.

3.5.1. Content Tagging. The QoS3 web server will accept object tags for the web page contents it serves. The web page objects can be labeled as either public or private, and conservatively, the private tag label will be set by default. The tags allow QoS3-enabled web servers and QoS3 web clients to control which web page content should be sent/received over the delegated HTTPS connection and which should be sent over the direct HTTPS connection. The public and private tags can be manually or automatically assigned to the web page content. Manually assigning the web page content may be overwhelming and cumbersome. Fortunately, web page content auto assignment can be done using some methods that can be used for automatically creating tags for the web page content. An example of these techniques is the template-extraction techniques [42]. These techniques can determine the templates of the web page automatically, and based on that, the web page content in that template can be tagged as public.

3.5.2. Web Page Objects' Hash Value (Checksum). A QoS3 server computes a checksum for every object labeled as the public one in the web page/website. These checksums are sent to the client together with the objects in the web page content template (Figure 4). The checksums allow the client to verify that the web page objects sent through the delegated HTTPS connection have not been compromised. Unlike QoS2 [38], we argue that the checksum can be calculated at runtime and cached for one day as the default value. Alternatively, as in Subresource Integrity (SRI), the checksums are calculated previously and embedded in the web page HTML file as SRI tag. Therefore, for each request, QoS3 first checks for the checksum in the cache. If it does not exist, then the checksum will be calculated. This will save the possible latency that may occur due to the checksum calculation. In case the client receives a checksum that does not match with the calculated checksum by the client because the object is updated in the origin server and not in the caching proxy, the client will request the object again; if there is still a mismatch, then the client will use the direct HTTPS connection to get that object again from the origin server.

3.5.3. SRI Tag. SRI is a new World Wide Web Consortium (W3C) specification for security feature which enables the web browsers to ensure that an object marked with the SRI tag is transferred without any modification on the path [18].

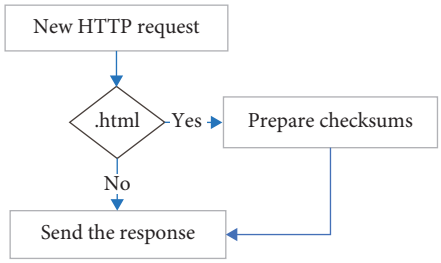

Figure 4: The QoS3-enabled server module.

We can leverage the SRI feature on QoS3 design. Unfortunately, SRI supports limited types of web page objects. Therefore, we propose extensions to the SRI to support QoS3 tags (e.g., public and private) and all the cacheable static web objects such as media files (images, flash, audio, etc.) as shown in Figure 5.

As shown in Figure 4, when QoS3-enabled server receives a new HTTP request, it checks if the request is for the base HTML file of the web page. If it is, the server verifies that the checksums exist. Otherwise, it calculates the checksums and sends it to the client alongside with the *.html* file. The checksums' management can be manually done by the content provider through the proposed extended SRI tags or automatically done using the QoS3 extension.

3.6. QoS3-Enabled Clients. The QoS3 client is the same as the traditional web browser except that it understands the QoS3 protocol. Specifically, the QoS3-enabled browser understands the use of the direct HTTPS connection as a direct connection between the client and the server where the private HTTPS requests are sent over this connection, as well as the checksums of the public web page objects and the delegated HTTPS connection in which the browser will delegate trust to specific trusted proxy servers and use this connection for the public content transport as detailed in Figure 6.

In addition to the traditional web browser standard functionality, the QoS3-enabled web client computes cryptographic checksums for the public objects as well, which will be received by the client via the delegated HTTPS connection. The QoS3 client verifies the generated checksum against the previously received checksums from the server via the direct HTTPS connection using the validation module shown in Figure 7.

Validated public web page objects are passed to the standard browser processing engine to parse and interpret them in order to be painted on the client screen. On the other hand, the checksum verifications process may fail for some reasons such as MitM attacks on the delegated HTTPS connection. In this case, the public web page object will be requested again via the direct HTTPS connection as shown in Figure 7. As part of future works, we plan to expand QoS3 by adding further extensive protocols to clarify the two options when checksum verification fails and determine whether it is safe to resend the public objects over delegated HTTPS or the direct HTTPS connection.

3.7. QoS3-Enabled Proxy. A QoS3-enabled proxy is a web proxy that understands the QoS3 protocol. A QoS3-enabled forward proxy is deployed as the transparent proxy server that can act as an intermediary for the HTTPS communications between the clients and the origin servers. The QoS3 proxy should not make any modifications to the objects cached on its cache storage. The Content-Encoding, Content-Type headers, and Content-Range must remain unchanged. This requirement can be achieved by setting the *Cache-Control: no-transform* directive by the server when serving the public objects to tell that the proxy does not make any modifications to the object. The following section describes the mechanism by which a QoS3-enabled proxy can be delegated and establish the trust with the client.

3.8. QoS3-Enabled Proxy Certificate. In order to help the QoS3-enabled HTTPS client to recognize and differentiate the QoS3 trusted proxies from other servers such as web servers, the CA issues special public key digital certificates to the QoS3-enabled proxy servers. More precisely, the CA can use the *Extended Key Usage* extension as stated in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile [43] to specify a key purpose *QoS3Proxy* (the IANA should assign a new object identifier for this key purpose). This *Extended Key Usage* extension should be marked as critical by the CA. This mechanism is inspired by [14].

3.9. TLS Connection Establishment with QoS3. QoS3-enabled client establishes two HTTPS connections with two different contexts. As shown in Figure 8, the first one is a direct connection which does not allow in-network proxies to terminate the SSL connection. The second is a delegated connection. In this context, the client allows in-network trusted proxies to terminate the SSL/TLS connection and apply in-network optimization functionality (i.e., caching) on the content.

In the following, we describe how the QoS3-enabled client establishes the two HTTPS connections using the two different contexts:

3.9.1. Context 1: Delegated HTTPS Connection. Figure 9 show the TLS handshake steps for the delegated HTTPS connection using QoS3. In this context, the client establishes an SSL connection with a customized TLS context in which the client will add the trusted proxy certificates as valid certificates. The client will negotiate delegated HTTPS connection using "Delegated" value in the Application Layer Protocol Negotiation (ALPN) extension field. The QoS3 proxy will then observe that this HTTPS connection is to be used for a public web page object. The proxy then will terminate SSL connection and reply to the client with hello message and proxy certificate. When a QoS3-enabled client/browser receives the proxy server certificate, it will check whether the proxy certificate includes an *Extended Key Usage* extension and will check whether the key purpose id denoted by *QoS3Proxy* is included. If the proxy certificate

```
<link rel="stylesheet" href="https://csu.example.com/QoS3/style.css"
integrity="sha384-+/M6kredJcxdsqkczBUjMLvqyHb1K/JThDXWsBVxMEeZHEaMKEOEct339VItX1zB"
crossorigin="anonymous" tag="Public">

<script src="https://csu.example.com/QoS3/script.js"
integrity="sha384-MBO5IDfYaE6c6Aao94oZrIOiC6CGiSN2n4QUbHNPhzk5Xhm0djZLQqTpL0HzTUxk"
crossorigin="anonymous" tag="Public" ></script>

<img src="https://csu.example.com/QoS3/img.png" alt="QoS3 img" height="8" width="6"
integrity="sha384-oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8K/uxy9rx7HNQlGYl1kPzQho1wx4JwY8wC"
crossorigin="anonymous" tag="Public">

<embed src="https://csu.example.com/QoS3/flash.swf"
integrity="sha384-7CPQa3/oac01bNe8mprhnSE68Q+xi2FHEMxrEq0nT32b9LDjZ1B3fQ0/NB24ZU9O"
crossorigin="anonymous" tag="Public" >

<audio src="https://csu.example.com/QoS3/audio.mp3"
integrity="sha384-qXiML0naZScJ2lH3oe28sgHJQTToTPAZB59AAYKnxUh2BDEETVfA6E8qxi+xeTSO"
crossorigin="anonymous" tag="Public"></audio>

<video src="https://csu.example.com/QoS3/video.mp4"
integrity="sha384-H/03sXUyCY+zZ06+nIuywCDFW4JUHYjzzk5qHuFKt2dY/ROc3QZrCIsRLHyzea3A"
crossorigin="anonymous" tag="Public"></video>
```

Figure 5: The proposed extended SRI tags.

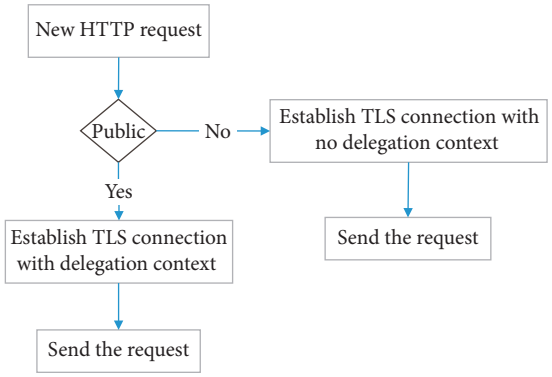

Figure 6: The QoS3 client-side module—request handler.

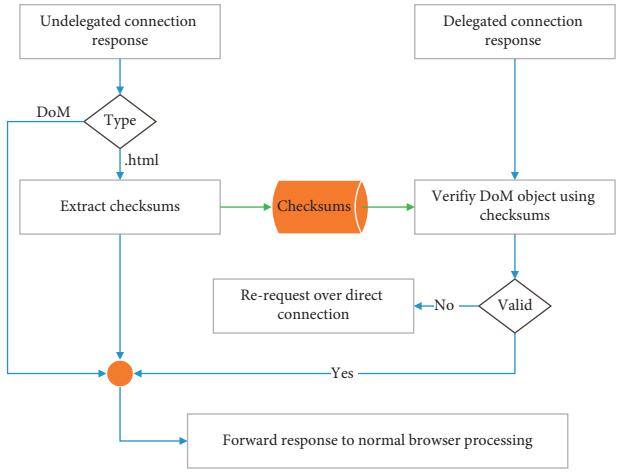

Figure 7: QoS3 client-side module—response handler.

contains the key purposes id for the QoS3 digital certificate, the QoS3 client concludes that the certificate belongs to a QoS3 proxy. The client then verifies that the QoS3 proxy

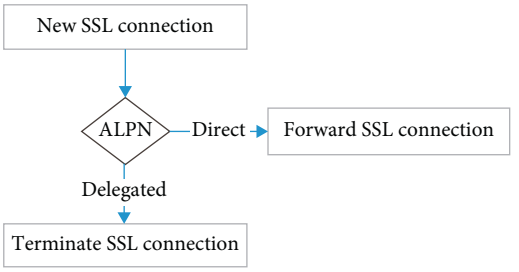

Figure 8: QoS3 proxy module.

certificate is a valid EV certificate and checks whether the certificate is in the trusted list. After the user has verified the proxy certificate, the QoS3 client then maintains this connection as a public resource connection. The QoS3-enabled client will send all the public resources to the server via this connection.

3.9.2. Context 2: Direct HTTPS Connection. Figure 10 shows the TLS handshake steps for direct HTTPS connection using QoS3. In this context, the client establishes a TLS connection by the default TLS context. The client will negotiate the direct HTTPS connection using "Direct" value in the ALPN extension field. Using ALPN, the QoS3-enabled proxy server will then observe that this is a direct HTTPS connection and should be used for the private web page objects transport. Then, the ClientHello message will be forwarded to the web server by the QoS3 enabled proxy, and therefore this connection will be end-to-end between the QoS3-enabled client and the QoS3-enabled web server.

4. Evaluation

In this section, we strive to affirm the QoS3 deployment and measure the advantages of QoS3. Moreover, we discuss the

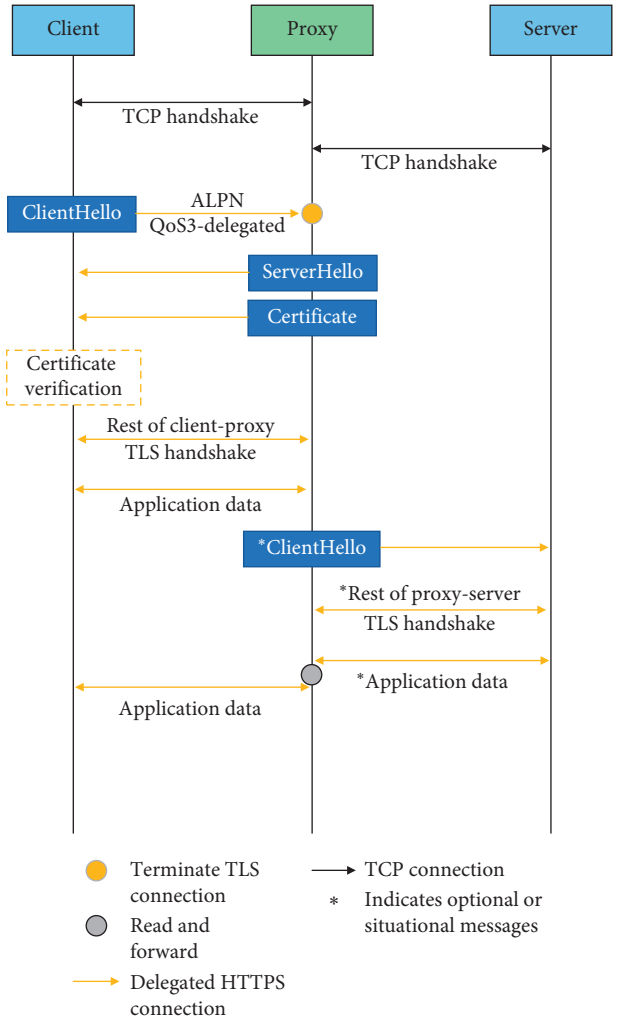

Figure 9: Delegated HTTPS connection context.

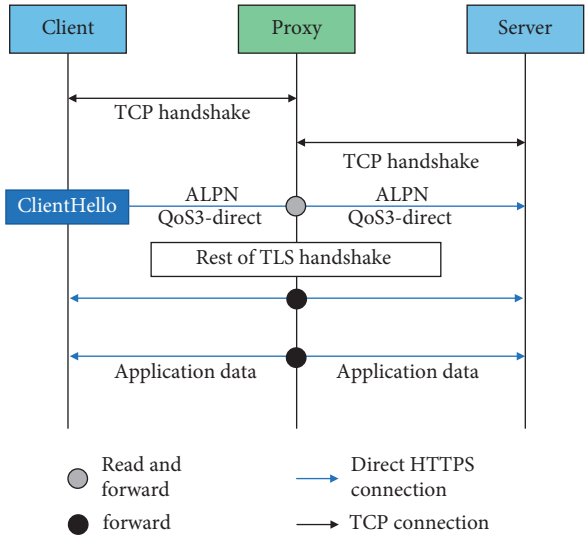

Figure 10: Direct HTTPS connection.

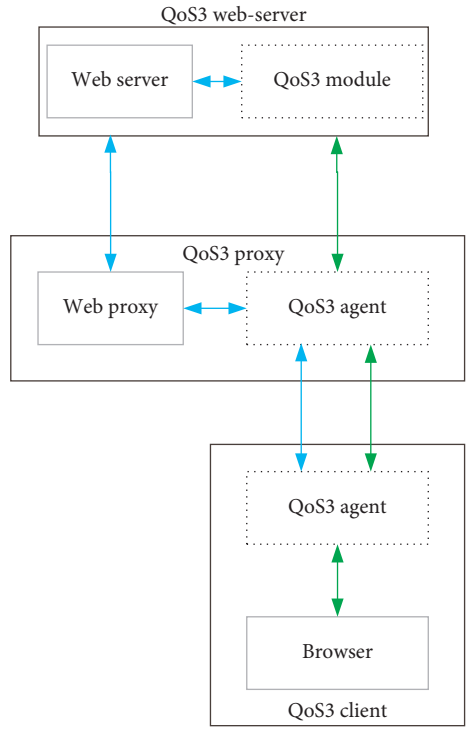

Figure 11: QoS3 local testbed.

testbed setup for the QoS3 implementation, the modifications required for client and server software, and the relevant QoS3 implementation details. Also, we discuss the required software changes at each endpoint based on the QoS3 design principles outlined in the preceding section.

4.1. QoS3 Experimental Setup. A local testbed has been setup in order to assess QoS3 functionality and the overall performance of QoS3. The local testbed consists of three computers. Two of them run Ubuntu 16.04 and one runs windows 7. Figure 11 represents the QoS3 testbed and implementation setup. In this testbed, we deploy a caching proxy (SQUID [44]) and a web server (Apache [45]) which are hosted in two different server machines with 2.8 GHz CPU and 8 GB RAM. The client computer, caching proxy server, and the web server machines are linked together by 1G Local Area Network (LAN). In order to simulate the Wide Area Network (WAN) environments, we use WANem [46] between the caching proxy server and the web server to simulate the WAN latency and the Linux tc command [47] on the caching proxy to set different latency values between the browser and caching proxy.

In our local testbed network, we set up a local web server using the Apache web server with OpenSSL software. We use Google Chromium browser in order to issue HTTPS requests to the local testbed server.

In order to show the potential gain that we can get from QoS3, we download six website templates that simulate popular websites listed in the Alexa Top 500 websites. These templates emulate the different website properties (i.e., number of objects and the size of objects), as shown in Table 1. In order to show a lower bound on the viable advantages of QoS3, we do not tag all static objects as public content.

Table 1: Test websites' properties.

Test page	Number of objects	Avg. objects' size
*TestWebsite*1	36	10.85 KB
*TestWebsite*2	49	1.87 KB
*TestWebsite*3	50	2.50 KB
*TestWebsite*4	33	4.84 KB
*TestWebsite*5	29	291.05 KB

We implement a QoS3 as an extension for both the client, the proxy server, and the web server. In our implementation, the client explicitly trusts the middlebox over the delegated HTTPS connection. By implementing QoS3, we verify that QoS3 solution is simple and compatible with the existing infrastructure. Furthermore, it can be deployed quickly without requiring a significant change in the client, proxy, and the web server.

4.2. QoS3 Functionality and Performance Validation

4.2.1. QoS3 Functionality Assessment. In order to make sure that QoS3 functions as expected, the following questions are addressed:

(1) Can the client download the webpages correctly without any errors?

(2) Can the proxy server cache the public content successfully without modifying it when storing in the cache?

(1) QoS3 Functionality Evaluation Methodology. At the server side, the QoS3 plugin tags the web page content as public or private and appends the checksums for the public objects. The experiments in our local testbed consist of ten executions. We use a Chromium extension (Average Load Time Tester [48]) to allow the client to make ten requests to each of the six chosen website URLs. During the test experiments, we record the average or the cumulative test results from the testing processes. On the client side, every HTTPS request will be forwarded to the QoS3 plugin which in turn will check the request type. If the request is for the base *.html* document, the QoS3 plugin will forward the request via the direct HTTPS connection to the web server. The server then responds with the *.html* file. The file will contain the checksums of the public objects. The QoS3 plugin in the client side receives the *.html* file and then extracts the checksums in order to store them temporarily for the following objects' validation process. After that, the *.html* response is forwarded to the normal browser engine for further processing. The QoS3 plugin parses the HTML DoM tree. Based on that, QoS3 plugin forwards the request of the objects embedded on the DoM tree either via direct HTTPS connection or delegated HTTPS connection based on the tag attached to the object. If the request is for a public object, in this case, the request is forwarded via the delegated HTTPS connection to the proxy server which in turn checks whether the object is in its cache or not. If it is not in the cache store, the proxy will request the object from the origin server, then store a copy of it in its cache, and forward the object to the client computer.

At the client side, the QoS3 plugin will receive the response from the proxy via the delegated HTTPS connection. Then, the QoS3 plugin creates a checksum for the received object and compares it with the one received from the origin server to check whether the received object is modified or not. After that, the object is forwarded to the normal browser processing engine for further processing.

Our goal from this is to test that the objects are retrieved from the server via the proxy through the delegated connection if they are not modified by the proxy server.

(2) Test Result. We find that QoS3 plugin in the client side creates the two HTTPS connections with the two different contexts (direct and the delegated HTTPS connections) successfully without any errors. The *.html* files are received via the direct HTTPS connection and the objects tagged as public are served over the delegated HTTPS connection. The proxy server is successfully trusted by the client, and it can cache the public objects. The server can serve the web page successfully, and the checksums are sent to the client via the direct connection successfully.

While doing the experiments, we find that the proxy may modify the objects stored in the cache. The change might be in the Content-Encoding, Content-Range, and Content-Type. In order to overcome this issue, the *Cache-Control: no-transform* tag should be set for the public objects when using QoS3. In this way, the system works correctly without any problems.

4.2.2. QoS3 Performance Evaluation. In order to determine the actual improvements of QoS3, we perform a page load time test in our QoS3 implementation environment on our own local testbed. Figure 11 shows our experimental testbed and how the QoS3 agent modules are installed. We implement a QoS3 agent for both the client, the proxy server, and the web server. Using those agents/modules, QoS3 is enabled in our testbed.

The latencies are varied between the web client, the caching proxy, and the web server. By changing all latencies, we can recognize how the location and placement of the middleboxes impact the performance benefits of QoS3. In Figures 12–15, we make a comparison of the page load time for the different latencies between the client and both the web server and the prospective proxy server. In order to define the latency from the proxy server to the web server in the local testbed, we make a latency test using ping command and traceroute to the web servers from the Alexa's top 500 websites. From the results that we get from the latency test, we select 25 ms for the fast links scenario, 75 ms and 250 ms for wireless and mobile links, and 400 ms for the slow links such as satellite links. For the latency between the client and the caching proxy, we also use ping and traceroute commands to do a latency test for the local ISP latency and then select 5 ms latency. Using this latency, we can recognize the benefits under the low latency links and the high latency links. We find that there is a performance improvement (30%) in the low latency links scenario and performance improvement up to 64% in the high latency links scenario. We also observe that improvements in the performance are a

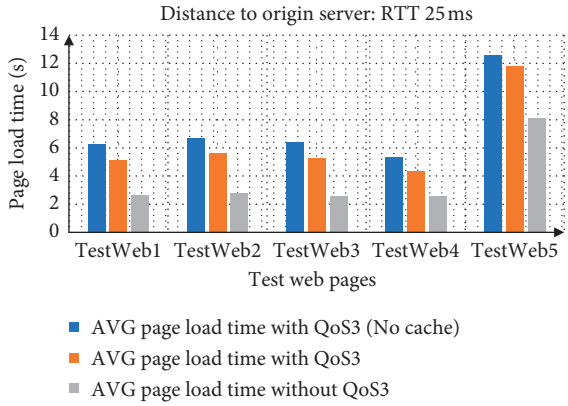

Figure 12: Page load time analysis—25 ms latency.

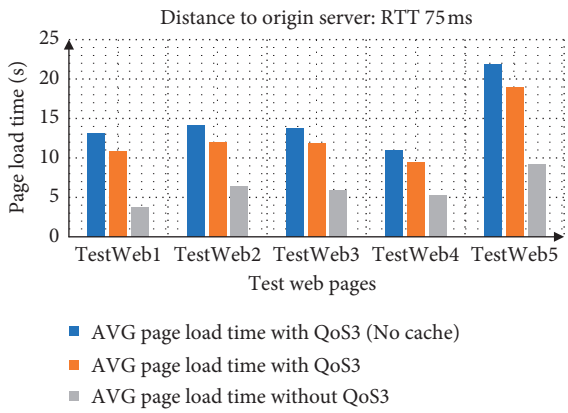

Figure 13: Page load time analysis—75 ms latency.

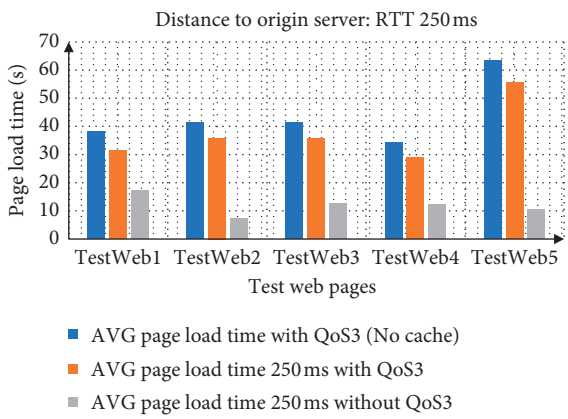

Figure 14: Page load time analysis—250 ms latency.

function of both the number of the cacheable public DoM objects, the sizes of the public DoM objects, and the dependencies between DoM objects.

5. Discussion

5.1. Deployment-Related Challenges. QoS3 requires minor changes to both web browsers, web proxy servers, and web servers. Fortunately, the required changes can simply be accomplished without affecting the normal operations of traditional HTTPS. The integration of QoS3 with the traditional HTTPS servers, clients, and proxies will not affect the none-QoS3-enabled servers, clients, and proxies. Therefore, QoS3-enabled clients can work with none-QoS3 servers and vice versa. It is also compatible with the middleboxes (caching proxies).

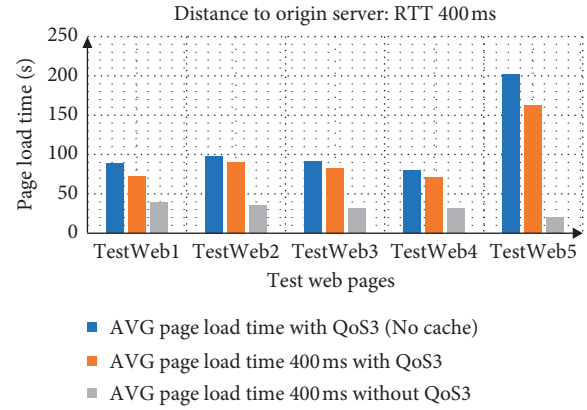

Figure 15: Page load time analysis—400 ms latency.

5.2. Video Streaming. Video content is increasing on the web, and according to Cisco, they are expecting that 82% of Internet traffic will be for video streaming traffic in 2020 [49]. Currently, QoS3 supports video file download, but it does not support video streaming. We plan to improve QoS3 to support video streaming by adding integrity hash value to each video stream chuck, which enables the client to verify the integrity of the video streams served from the proxy cache store.

5.3. Tagging or Classifying the Web Page Content. In order to overcome the utmost deployment hurdle for classifying or tagging the web page objects into either public content or private content, we can utilize and apply one of the conventional template-extraction methods [42]. The public content can be tagged using a particular template by matching the objects in the web page with the template.

5.4. Motivation for Adoption. It is fairly obvious that there is motivation for ISPs and enterprises as well as the end-users for adopting tools and infrastructure to support QoS3. Both the ISPs, enterprises, and the end-users are able to reap enormous improvements from enabling QoS3. Specifically, ISPs and enterprises can reduce traffic by 30%, and users can reduce page load times by at least 50%. Even though the web content provider might be resistive, several web content providers are intended to reduce page load times [50]. Therefore, they have the motivation to adopt optimizations methods such as QoS3.

5.5. Other In-Network Functionality Support. In QoS3, we focus specifically on performance-oriented middleboxes.

Additional work to QoS3 is needed to support security-oriented middleboxes like deep packet inspection systems and intrusion detection systems.

5.6. Third-Party Content. In case the web page contains object hosted in a third-party or external content provider, this object will be fetched through the traditional HTTPS (direct HTTPS connection) from the third-party external content provider.

5.7. Browser Cache. The TLS encryption does not affect the functionality of the client-side browser caches. Fortunately, in-network caches and browser caches are orthogonal to each other. Therefore, different benefits can be gained from both levels of caching. QoS3 was specifically designed to recover the benefits of in-network caches.

5.8. Future Transport Protocols. According to recent web performance studies, several novel transport protocols are in the development process in order to improve the performance of secure web content browsing on the Internet, such as QUIC. However, these protocols do not explicitly support middleboxes. QoS3 supports these protocols.

5.9. Security Measurements and Threat Models. The influence of the different types of MitM attacks can be classified as follows:

(i) *JavaScript Execution*. Any attempt to inject malicious code inside javascript files sent over QoS3 delegated HTTPS connection will be detected using the checksums that are sent from the origin server to the QoS3 client over the direct HTTPS connection.

(ii) *Cookie Stealing*. QoS3 uses two HTTPS connections for web page delivery. One is for delegating trust to middleboxes, and the other is direct to the origin server. Even though the cookie is protected by the "Secure flag," this will not protect the cookie from the trusted proxy. However, QoS3 ensures that Cookie is sent only over a direct HTTPS connection.

(iii) *Request Forgery*. If an attacker attempts to forge or initiate a fake request on behalf of the client, the QoS3 client can detect such forged requests using the checksums received from the origin server via the direct HTTPS connection.

6. Related Work

A considerable amount of literatures have been published on enabling trusted in-network middleboxes to inspect the web content encrypted communications between the endpoints. In this section, we provide a brief summary of the literatures relating to our work QoS3.

6.1. MitM TLS Proxies. The most popular method of providing the in-network middleboxes access to the encrypted communications between the endpoints is by installing the middlebox's certificate in the root certificate directory on the clients [51, 52], in order to enable the in-network middleboxes to act as a man in the middle between the endpoints. However, this method will bring major risk and vulnerabilities. To examine this issue, a recent study [53] reports that 75% of the in-network trusted TLS middleboxes make the clients vulnerable to MitM attacks by other adversaries on the Internet. These trusted in-network middleboxes break the end-to-end security of the TLS connection, and the client will be vulnerable by any vulnerability on the trusted middlebox [54, 55].

6.2. Caching Encrypted Content. Caching encrypted content [56] is a recent proposal that allows the transparent caching of encrypted content on the middleboxes, and it can be deployed in the edge of the network of the ISP while allowing Content Providers (CPs) to have control over their content exclusively. Unlike QoS3, this approach requires new special requirements for content providers and ISPs to allocate and manage their cache partitions storage. QoS3 is easily deployable, more practical, and less costly.

6.3. Multicontext TLS. This scheme enables the client and the server to give the in-network middleboxes either read-only or read-write access permissions to specific TLS records in the TLS session [9]. The idea of the encryption context is introduced in this scheme. The encryption context consists of a group of symmetric encryption keys for data encryption, and a group of Message Authentication Code (MAC) keys for message integrity in order to control the access level permissions to the traffic sent between the endpoints through the middlebox. It means that the endpoints can define a set of privileges using the context in order to enable the middleboxes to read or modify the content sent between the endpoints. In mcTLS, the access control mechanism is applied at the TLS record level. Specifically, the access control and the integrity verification are achieved using 3 MAC codes for every single TLS record. The MAC codes are encrypted using the context keys. The first is the reader MAC code, which can be generated by the middleboxes that have read access permission; the second is the writer MAC, which can be generated by middleboxes that have write access permissions; and the third is the endpoint MAC, which should be generated by the target server to provide end-to-end (E2E) integrity and used by the client for E2E verification. But, in case of caching proxy, when the content is served from the proxy cache, it is not clear how the E2E MAC is constructed. In this case, we assume that the proxy constructs the E2E MAC code and the client assumes that the content is served from the target server. Consequently, there is no E2E integrity verification and the client cannot detect if the proxy has tampered with the content served from the proxy cache. Therefore, the client should fully trust the proxy. mcTLS may be weak in several cases, for example, Poisoning Caches with an Unknown Key Share Attack [57]. Furthermore, mcTLS complicates the TLS protocol by making significant changes to the handshake protocol, and

the sizes of the handshake messages are significantly large. Avoiding changes to the TLS protocol is required both for security and for easy deployment because TLS is an elusive protocol [58–60], and significant modifications introduce risks for new vulnerabilities.

6.4. Middlebox TLS (mbTLS). This scheme enables outsourcing middlebox functionalities and allows outsourced functionalities to take place in TLS connections [11]. mbTLS leverages the Software Guard Extensions (SGX) technology to enable outsourced middlebox functionalities in a secure way. In mbTLS, during the TLS handshake, the endpoints will open a secondary connection to all the middleboxes that will participate in the TLS session for authentication and authorization. The endpoints send the TLS session keys to the middleboxes participating in the TLS session after they complete the primary TLS handshake. Alternatively, mbTLS assure the middlebox integrity using Trusted Execution Environment (TEE) technology.

6.5. Authentication in Delegated Service for CDN with HTTPS. This scheme proposed a solution to address the authentication delegation problem for CDN in HTTPS [61]. The proposed solution is based on an emerging protocol, DNS based Authentication of Named Entities (DANE), which is currently standardized to improve the current PKI encryption mode. The basic idea is that they extend DANE TLSA record to include the certificate of the origin server and the certificate of the delegated CDN server, which enables the client to identify the delegation relationship. This solution is suited for the content provider who uses a CDN network. This is different from QoS3. In QoS3, the trust delegation is granted to the proxy server from the client and the server does not need to delegate trust to any proxy. QoS3 is suited for application layer caching proxies which are typically deployed as a shared gateway for all the clients in a network such as ISP gateways and enterprise network gateway.

6.6. Explicit Proxy Support. Alongside the academic work for allowing in-network middleboxes with TLS, several proposals have been introduced, which propose modifying and adding extensions to the TLS protocol. One aspect of this type of works is to allow the TLS client to specify an explicit proxy with privileges to decrypt and modify the TLS traffic [12–14,62]. Unlike these proposals, QoS3 provides fine-grained trust delegation and allows the content provider to apply fine-grained control over web content with end-to-end content integrity guarantee.

Another approach is the QoS2 scheme [38], which allows caching web content by caching proxies over HTTP and guarantees integrity as in the TLS. This approach uses HTTP connection for cacheable public content and HTTPS for other contents and protects the content served over HTTP using checksums. The main drawback of this approach is the use of HTTP and HTTPS connections. For that reason, this approach suffers from the compatibility with HTTP Strict Transport Security (HSTS) security policy that enforces all content should be served over secure HTTPS connections. And many popular websites have implemented this policy to secure their servers and clients from MitM Stripping attacks. Moreover, the trend in web delivery is that HTTPS becomes mainstream for web delivery. For example, the web delivery protocols SPDY and HTTP 2.0 [22] use HTTPS by default and recently QUIC [28] only supports HTTPS. Hence the HTTP support will not have place in the future of the web. As a result, it is less likely to be deployable. In QoS3, we only use HTTPS to download both the public and private content, with fine-grained trust delegation by using two HTTPS connections, one is direct, and the other is delegated. We also proposed extensions to the SRI to support web page content tagging and including the secure hash values for the public content. Unlike the original SRI, QoS3 generates the SRI tags and checksums automatically.

7. Conclusion and Future Work

We present a framework that enables the client to provide fine-grained trust delegation to caching proxies and content provider to classify web content using QoS3 protocol to explicitly and securely re-introduce in-network caching proxies in the HTTPS communications, without compromising the integrity of the HTTPS content and modifying the format of the TLS. Using QoS3, the TLS connections between the client and the server remain encrypted and keep their end-to-end integrity as in the original TLS. QoS3 (1) provides clients with explicit control and knowledge over which middleboxes (trusted proxies) are involved in a particular connection and for which content, (2) allows the content providers to define the public parts of the web page content which can be served over the delegated HTTPS connection and thus can be cached on the in-network caching proxies, (3) maintains the integrity and authenticity of public content using checksums, and (4) is easily deployable. We also provide a proof of concept implementation for QoS3 as an extension to the HTTPS on the client, proxy, and the server. By using QoS3, the caching proxies can provide an improvement on the page load time ranging between 30% and 64%. We believe enabling QoS3 with video streaming applications and using template-extraction methods for enabling auto object classification are good research directions for QoS3 enhancement.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (grant nos. 61872403 and 61972421).

References

[1] Fortinet Inc, *Quarterly Threat Landscape Report Q3 2018*, Fortinet Inc, Sunnyvale,CA, USA, 2018, https://www.fortinet.com/content/dam/fortinet/assets/threat-reports/threat-report-q3-2018.pdf.

[2] C. Coarfa, P. Druschel, and D. S. Wallach, "Performance analysis of TLS web servers," *ACM Transactions on Computer Systems*, vol. 24, no. 1, pp. 39–69, 2006.

[3] Y. Elkhatib, G. Tyson, and M. Welzl, "Can SPDY really make the web faster?," in *Proceedings of the IFIP Networking Conference*, pp. 1–9, IEEE, Trondheim, Norway, June 2014.

[4] J. Erman, V. Gopalakrishnan, R. Jana, and K. Ramakrishnan, "Towards a SPDY'ier mobile web?," *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 2010–2023, 2015.

[5] J. Jarmoc and Dell Secure Works Counter Threat Unit, "SSL/TLS interception proxies and transitive trust," in *Proceedings of the Black Hat Europe*, Abu Dhabi, UAE, 2012.

[6] D. Naylor, A. Finamore, I. Leontiadis et al., "The cost of the "s" in https," in *Proceedings of the 10th ACM International Conference on Emerging Networking Experiments and Technologies*, pp. 133–140, ACM, Sydney, Australia, December 2014.

[7] ETSI, *Cyber: Middlebox Security Protocol; Part 3: Profile for Enterprise Network and Data Centre Access Control*, ETSI, Sophia Antipolis Cedex, France, 2019, https://www.etsi.org/deliver/etsi_ts/103500_103599/10352303/01.01.01_60/ts_10352303v010101p.pdf.

[8] IEEE SA—IEEE Encrypted Traffic Inspection (ETI), Enabling the Development of an Accepted Way of Traffic Inspection, 2019, https://standards.ieee.org/industry-connections/icsg/eti.html.

[9] D. Naylor, K. Schomp, M. Varvello et al., "Multi-context TLS (mcTLS): enabling secure in-network functionality in TLS," in *Proceedings of the ACM SIGCOMM Computer Communication Review*, vol. 45, pp. 199–212, ACM, New York, NY, USA, October 2015.

[10] J. Sherry, L. Chang, R. A. Popa, and S. Ratnasamy, "Blindbox: deep packet inspection over encrypted traffic," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 213–226, 2015.

[11] D. Naylor, R. Li, C. Gkantsidis, T. Karagiannis, and S. Peter, "And then there were more: secure communication for more than two parties," in *Proceedings of the 13th International Conference on Emerging Networking Experiments and Technologies*, pp. 88–100, ACM, Incheon, South Korea, December 2017.

[12] R. Peon, *Explicit Proxies for http/2.0-draft-rpeon-httpbis-exproxy-00*, IETF Informational Internet-Draft, Fremont, CA, USA, 2012.

[13] D. McGrew, D. Wing, Y. Nir, and P. Gladstone, "TLS Proxy Server Extension - draft-mcgrew-tls-proxy-server-01," IETF Informational Internet Draft, Fremont, CA, USA, 2012.

[14] S. Loreto, J. Mattsson, R. Skog et al., "Explicit trusted proxy in http/2.0 - draft-loreto-httpbis-trustedproxy20-01," IETF Informational Internet Draft, Fremont, CA, USA, 2014.

[15] L. Chang, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, "Embark: securely outsourcing middleboxes to the cloud," in *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*, pp. 255–273, Santa Clara, CA, USA, March 2016.

[16] M. Kasbekar, "Https request enrichment," US Patent App. 14/868,771, 2016.

[17] C. E. Gero, J. N. Shapiro, and J. B. Dana, "Terminating SSL connections without locally-accessible private keys," US Patent 9,647,835, 2017.

[18] D. Akhawe, F. Braun, F. Marier, and J. Weinberger, "Sub-resource Integrity," 2016, https://www.w3.org/TR/SRI/.

[19] J. Hodges, C. Jackson, and B. Adam, "Http strict transport security (HSTS)," RFC No. 6797, 2012.

[20] M. Kranch and B. Joseph, "Upgrading https in mid-air: an empirical study of strict transport security and key pinning," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, February 2015.

[21] P. Chen, N. Nikiforakis, C. Huygens, and L. Desmet, "A dangerous mix: large-scale analysis of mixed-content websites," in *Information Security*, pp. 354–363, Springer, Cham, Switzerland, 2015.

[22] M. Belshe, R. Peon, and M. Thomson, "Hypertext transfer protocol version 2 (http/2)," RFC No. 7540, 2015.

[23] L. Adam, N. Modadugu, and B. Moeller, "Transport layer security (TLS) false start," RFC No. 7918, 2016.

[24] T. Dierks and E. Rescorla, "The transport layer security (TLS) protocol version 1.2," RFC No. 5246, 2008.

[25] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," Technical Report, IETF, Fremont, CA, USA, 2018.

[26] X. Phillip Hallam-Baker, "509v3 transport layer security (TLS) feature extension," RFC No. 7633, 2015.

[27] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan, "TCP fast open," in *Proceedings of the Seventh Conference on Emerging Networking EXperiments and Technologies*, p. 21, ACM, Tokyo, Japan, December 2011.

[28] R. Hamilton, J. Iyengar, I. Swett et al., *Quic: A UDP-Based Secure and Reliable Transport for http/2. IETF, Draft-TSVWG-Quic-Protocol-02*, IETF, Fremont, CA, USA, 2016.

[29] R. Chang, J. Wang, J. Huang, Y. Pan, and N. Xiong, "S3: size-aware sequential scheduling to meet deadlines in data center networks," *Journal of Internet Technology*, vol. 19, no. 7, pp. 1961–1972, 2018.

[30] J. Huang, S. Li, R. Han, and J. Wang, "Receiver-driven fair congestion control for tcp outcast in data center networks," *Journal of Network and Computer Applications*, vol. 131, pp. 75–88, 2019.

[31] S. Liu, J. Huang, Y. Zhou, J. Wang, and T. He, "Task-aware TCP in data center networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 1, pp. 389–404, 2019.

[32] X. Jin, E. Li, L. Vanbever, and J. Rexford, "Softcell: scalable and flexible cellular core network architecture," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, pp. 163–174, ACM, Santa Barbara, CA, USA, December 2013.

[33] X. Xu, Y. Jiang, F. Tobias, E. Katz-Bassett, D. Choffnes, and R. Govindan, "Investigating transparent web proxies in cellular networks," in *Passive and Active Network Measurement*, pp. 262–276, Springer, Cham, Switzerland, 2015.

[34] J. Erman, A. Gerber, M. T. Hajiaghayi, D. Pei, and S. Oliver, "Network-aware forward caching," in *Proceedings of the 18th International Conference on World Wide Web*, pp. 291–300, ACM, Madrid, Spain, April 2009.

[35] J. Erman, A. Gerber, M. Hajiaghayi, D. Pei, S. Sen, and O. Spatscheck, "To cache or not to cache: the 3G case," *IEEE Internet Computing*, vol. 15, no. 2, pp. 27–34, 2011.

[36] T. Zhang, J. Wang, J. Huang, J. Chen, Y. Pan, and G. Min, "Tuning the aggressive TCP behavior for highly concurrent http connections in intra-datacenter," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3808–3822, 2017.

[37] A. Jeffries, "Concepts of interception caching," 2018, https://wiki.squid-cache.org/SquidFaq/InterceptionProxy.

[38] Z. Zhou and T. Benson, "Towards a safe playground for https and middle boxes with QoS2," in *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, pp. 7–12, ACM, London, UK, August 2015.

[39] Q. Huang, K. Birman, R. van Renesse, W. Lloyd, S. Kumar, and H. C. Li, "An analysis of facebook photo caching," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pp. 167–181, ACM, Farmington, PA, USA, November 2013.

[40] M. Butkiewicz, H. V. Madhyastha, and V. Sekar, "Understanding website complexity: measurements, metrics, and implications," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, pp. 313–328, ACM, Berlin, Germany, November 2011.

[41] X. George, C. N. Ververidis, V. A. Siris et al., "A survey of information-centric networking research," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 1024–1049, 2013.

[42] Z. Bar-Yossef and S. Rajagopalan, "Template detection via data mining and its applications," in *Proceedings of the 11th International Conference on World Wide Web*, pp. 580–591, ACM, New York, NY, USA, 2002.

[43] D. Cooper, S. Santesson, S. Farrell et al., "509 public key infrastructure certificate and certificate revocation list (CRL) profile," Technical Report, IETF, Fremont, CA, USA, 2008.

[44] SQUID Org. Squid: Optimising Web Delivery, 2018.

[45] Apache Org. Apache Http Server Project, 2018.

[46] WANem Org. Wanem: the Wide Area Network Emulator, 2018.

[47] B. Hubert, *TC-Linux Man Page*, 2010, http://man7.org/linux/man-pages/man8/tc.8.html.

[48] Fanplayr Inc, "Average load time tester," Fanplayr Inc, Menlo Park, CA, USA, 2018.

[49] CEV, "Virtual networking index (VNI)," 2018, http://www.cisco.com/web/solutions/sp/vni/vni_forecast_highlights/index.html.

[50] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Scott, "Low latency via redundancy," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, pp. 283–294, ACM, Santa Barbara, CA, USA, December 2013.

[51] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson, "Analyzing forged SSL certificates in the wild," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, pp. 83–97, IEEE, San Jose, CA, USA, May 2014.

[52] S. Stephen, *How the Nokia Browser Decrypts SSL Traffic: A "Man in the Client"*, 2013, https://freedom-to-tinker.com/2013/01/11/how-the-nokia-browser-decrypts-ssl-traffic-a-man-in-the-client/.

[53] X. de Carné de Carnavalet and M. Mannan, "Killed by proxy: analyzing client-end TLS interception software," in *Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, USA, February 2016.

[54] CEV, "CVE-2012-3372: vulnerability in cyberoam DPI devices, common vulnerabilities and exposures list," 2012, https://nvd.nist.gov/vuln/detail/CVE-2012-3372.

[55] CEV, "CVE-2016-1280: self-signed certificate with spoofed trusted issuer CN accepted as valid. Common vulnerabilites and exposures list," 2016, https://nvd.nist.gov/vuln/detail/CVE-2016-1280.

[56] A. Araldo, G. Dan, and D. Rossi, "Caching encrypted content via stochastic cache partitioning," *IEEE/ACM Transactions on Networking*, vol. 26, no. 1, pp. 548–561, 2018.

[57] K. Bhargavan, I. Boureanu, A. Delignat-Lavaud, P.-A. Fouque, and C. Onete, "A formal treatment of accountable proxying over TLS," in *Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP)*, pp. 799–816, IEEE, San Francisco, CA, USA, May 2018.

[58] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud et al., "A messy state of the union: taming the composite state machines of TLS," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, pp. 535–552, IEEE, San Jose, CA, USA, May 2015.

[59] K. Bhargavan, A. D. Lavaud, C. Fournet, A. Pironti, and P. Y. Strub, "Triple handshakes and cookie cutters: breaking and fixing authentication over TLS," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, pp. 98–113, IEEE, San Jose, CA, USA, May 2014.

[60] K. Bhargavan and G. Leurent, "Transcript collision attacks: breaking authentication in TLS, IKE, and SSH," in *Proceedings of the Network and Distributed System Security Symposium–NDSS*, San Diego, CA, USA, February 2016.

[61] J. Liang, J. Jiang, H. Duan, L. Kang, T. Wan, and J. Wu, "When https meets CDN: a case of authentication in delegated service," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, pp. 67–82, IEEE, San Jose, CA, USA, May 2014.

[62] Y. Nir, *A method for sharing record protocol keys with a middlebox in*, TLS working group, Internet-draftdraft-NIR-TLS-keyshare-02, Fremont, CA, USA, 2012.