Towards a Safe Playground for HTTPS and MiddleBoxes with QoS2

Zhenyu Zhou
Duke University
zzy@cs.duke.edu

Theophilus Benson
Duke University
tbenson@cs.duke.edu

ABSTRACT

The increase in concern over network privacy has prompted adoption of HTTPS to sky-rocket with over 50% of traffic delivered using HTTPS. Unfortunately encryption HTTPS eliminates the benefits provided by middleboxes such as proxies and caches. We claim that these limitations, highlight the need for alternative mechanisms for quickly and safely viewing websites.

QoS2 argues for fine-grained identification of common content and user-specific content, which are then delivered over either HTTP or HTTPS respectively. The main challenge in enabling QoS2, lies in ensuring that security is not compromised, namely preventing to Man in the Middle attacks. QoS2 overcomes these attacks by judiciously employing object level checksums which are sent exclusively over an HTTPS connection. To quantify the benefits of QoS2, we have manually tagged content for a number of sites and emulated an QoS2 server: initial results are promising with QoS2 providing a 20%-70% speed up over traditional HTTPS.

CCS Concepts

•**Networks** → **Network security; Network management;** *Network experimentation;*

Keywords

Network performance; network security; transport layer security; network management

HotMiddlebox'15, August 17-21, 2015, London, United Kingdom
 ISBN 978-1-4503-3540-9/15/08…$15.00
DOI: http://dx.doi.org/10.1145/2785989.2785998

1. INTRODUCTION

In the wake of the recent breaches in end-user privacy, by the NSA and other parties [1], the Internet has grown even more "secure" with content providers, such as Google, wantonly and indiscriminately employing HTTPS [2] across all services. According to a recent study [3], over 50% of web content is served over HTTPS.

This enhanced security comes at an unexpected cost: the TLS layer and the encryption it provides, renders middleboxes ineffective. Unfortunately, middleboxes are crucial for a number of reasons ranging from improved performance to security. Middleboxes such as caches, transcoders, and proxies provide tremendous benefits to end-users (reducing page load times by as much as 50% [4]), and to Internet Service Providers (reducing network utilization by 33% [5, 6]). Instead, by employing TLS, these benefits are lost. Essentially, a trade-off is made between privacy and performance with content providers forcing end-users to choose privacy over performance.

Existing approaches [7] for enabling these middleboxes over HTTPS argue for delegating trust to "Trusted middleboxes"; these "Trusted middleboxes" are given special keys that allow them to terminate and split TLS connections. There are several drawbacks with these approaches: first, it requires a fork-lift upgrade of the middleboxes within the existing infrastructure; and second, once employed these middleboxes provide users with little control over privacy. Essentially, these "Trusted middleboxes" can be used to compromise the user's privacy. We argue that existing approaches can be too coarse-grained and provide the end-user with little to no control over what is sent in the clear or provide control at the expense of significant complexity.

In this paper, we try to answer the following question:
Can we develop a framework that reintroduces middleboxes without requiring delegation of trust?

Interestingly, we observe that middleboxes can be introduced without delegating trust if the content provider applies fine-grained control over which content is kept private and thus encrypted (served over HTTPS) and which is kept public and thus transferred over HTTP.

Our observation is motivated by the fact that the template of a website is identical for all users and thus known. For example, the Facebook logo, base-HTML, CSS, and several java-script pages are identical for all users. Given this, we claim that little privacy is lost if these common objects are sent over clear text (in HTTP).

The challenge in sending mixed-context, a page with mixed HTTP and HTTPS objects, is that it leaves the page vulnerable to Man-in-the-Middle attacks (MitM). To enable fine-grained control over privacy in a secure manner, we have developed a new framework called QoS2. QoS2 allows content providers to determine which content is privacy (sent over HTTPS), as well as which is public (sent over HTTP). To prevent MitM, QoS2 generates checksums for all public content and sends these checksums over a secure HTTPS connection (thus preventing tampering).

In this paper, we take the first step towards realizing QoS2 by analyzing the design space and empirically evaluating the benefits of employing QoS2. Our initial experiments show that by moving objects in the critical path from HTTPS to HTTP, QoS2 enables performance oriented middleboxes to improve page load time by 20% - 70%.

Roadmap. The rest of this paper is organized as follows. We present background on middleboxes and TLS in Section 2; in Section 3, we discuss related works; in Section 4, we discuss the design space and present the architecture of QoS2; in Section 5, we evaluate the potential benefits of QoS2; in Section 6, we discuss deployment challenges; and finally, we conclude with a discussion of future works and open issues in Section 7.

2. BACKGROUND

In this section, we discuss HTTPS and its implications on performance, present a brief taxonomy of web-content, and finally present an overview of the vulnerabilities of sites with a mixture of HTTP and HTTPS content (more widely known as mixed-content sites).

2.1 Implications of HTTPS

The increase in adoption of TLS and HTTPS has a multitudes of benefits and drawbacks. While the benefits are prominent, we find that until recently the drawbacks have received little attention. In this section, we highlight the drawbacks of TLS and focus on its implications for performance and privacy. Websites implement HTTPS:

1. **Blindly:** All Objects are retrieved via HTTPS, regardless of size or privacy of information and in certain cases these objects unnecessarily incur performance overheads. For example, a recent study showed that for websites with HTTPS, the TLS handshake accounts for over 42% of data exchanged [3].
2. **With harmful consequences:** HTTPS prevents middleboxes, e.g. caches and transcoders, from inspecting packets and optimizing end-user performance. These can result in a significant increase in the traffic on the ISP's links and further increase page load times.
3. **Carelessly:** Many sites use HSTS [8] as a mechanism for automatically employing HTTPS. However, over 59% of websites do so incorrectly resulting in security flaws, e.g. redirect to HTTP domain or improperly set the parameter `max-age`. Surprisingly, even the sites that configure HSTS correctly are vulnerable to a number of attacks due to stale policies or modifications of time-stamps [9].

2.2 Classification of Web Content

We argue that web-content can be classified based on its sensitivity or privacy:

1. **Public Content**: These are the parts of a webpage which are identical for all users, e.g. javascript, banner images and CSS. We claim these aspects are public knowledge: they are identical for all users and thus knowing the DNS or IP of the website is sufficient for an adversary to determine the contents of these objects. Orthogonally, knowing the content of these objects does not leak any additional information beyond what is leaked by the IP or DNS.
2. **Private Content**: These are the relatively personalized content which are unique to each user, e.g. pictures or advertisements. Moreover, these objects are often non-cacheable, e.g., passwords, or may incur relatively low cache hit-rates, e.g, personal pictures and advertisements.

Interestingly, we observe that from a performance perspective, the public content is the highly cacheable content where as the private content is highly specific and often un-cacheable. For example a recent study from Facebook showed that highly personalized content is less cacheable within the network [10]. Furthermore, the public content, e.g. scripts, HTML-XML and images, accounts for over 50% of the bytes [11]: focusing solely on them will provide significant benefits.

2.3 MitM: Stripping Attacks

Today, websites exclusively serve content over either HTTP or HTTPS with very few websites supporting a mixture of both. In fact, many browsers do not load websites with mixed-content [9]. Mixed-content websites are dis-encouraged as they leave the websites vulnerable to MitM attacks.

The mixture of HTTP and HTTPS, implies that content served over the HTTP content will be unencrypted and thus unprotected. An attacker can easily hijack the HTTP connections and send comprised content to the user. This compromised content can in turn be used to steal information stored in cookies or worst modify

the web page's DOM and thus its appearance. For example, an attack can conduct a MitM on the HTTP connection and use this connection to inject java-script code to steal information stored in cookies [12].

Takeaways. Web content can be roughly classified into public and private content. Private content incurs low cache hit-rates and should not be cached for sensitivity issues. To enable caching and other performance oriented middlebox optimizations, public content should be sent over HTTP. Unfortunately, using a mixture of HTTP and HTTPS leads to a number of security vulnerabilities.

3. RELATED WORKS

Implications of Middleboxes: There are many recent works [13, 4, 5, 6] demonstrating the importance of middleboxes within cellular networks and wired networks. These middleboxes have been shown to provide tremendous benefits to both the networks and the end users. Thus there is a great incentive to ensure that these middleboxes remain operational. QoS2 presents a system to achieve just that.

Middleboxes in the Age of HTTPS: The closest related work [7, 14, 15, 16] enables middleboxes by allowing the end-points to share keys with the middleboxes or involve the middleboxes in the key exchange. Using these resulting keys, the middleboxes are able to terminate the connection and apply network functions on the traffic. These approaches fall into two classes: in the first [7, 14, 15], all traffic is encrypted with one key and by sharing this key with the middleboxes, the end-points are forced to delegate all their trust to the third party middlebox. Where-as in the second [16], the end-points employs different keys to encrypt different portions of the traffic. By selectively sharing different keys, the end-points are able to decide which content the middleboxes has access to. Unlike QoS2, these approaches require a complex key-exchange protocol and require a fork-lift upgrade of the in-network middleboxes. QoS2 attacks a different point in the design space, arguing instead that content provider should judiciously apply encryption at a fine granularity rather than sharing keys. This allows QoS2 to be incrementally deployed but limits the applicability of QoS2 to performance oriented middleboxes.

Other approaches [17, 18, 19, 20] to enable middleboxes on encrypted traffic attack the problem at a lower layer: transport [17, 18] and network [19, 20]. These approaches require separate key management schemes.

Orthogonal approaches include BlindBox [21], which aims to perform a set of middlebox functionality on encrypted traffic without decrypting it. While BlindBox focuses on security oriented middleboxes, QoS2 focuses on performance oriented middleboxes.

4. RETHINKING WEB-SERVERS

In this section, we explore the design space for re-architecting secure content delivery and reason about a strawman approach, QoS2, for enabling the safe co-existence of TLS with certain middleboxes.

Currently, the web supports two forms of privacy, w.r.t web pages: the first, no privacy, in which the pages are served over HTTP; and the second, total privacy, in which the pages are served over HTTPS.

In previous sections, we argued that these options are not expressive enough; they either provide privacy (HTTPS with no middleboxes) or good-performance (HTTP with middleboxes). Thus, we claim that what is required is an option that straddles both, providing privacy with high performance.

One natural way to achieve this option is to split connections up, with content sent over both HTTP and HTTPS but this has two challenges:

- Overcoming the security issues discussed in Section 2.
- Determining which content to send over HTTP and which to send over HTTPS.

Alternatively, one could encrypt content with different keys and selectively share a subset of these keys with the middleboxes within the network. Although this addresses the drawbacks of the previous solution, this approach compromises forward secrecy, complicates the protocol, potentially introduces latency overheads, and requires upgrades of in-network hardware.

In this work, we explore the former: our framework, QoS2, presented in Figure 1, explicitly addresses both challenges by proposing enhancements to both the web-servers and the web-browsers. At a high-level, QoS2 prevents and overcomes MitM attacks by (1) creating checksums for each object sent over HTTP (2) sending these checksums over an HTTPS connection, and (3) verifying the objects sent over HTTP against the checksums transferred over HTTPS.

4.1 QoS2 Web-Servers

A QoS2 web-server is similar to a traditional web server except in the following ways:

Tagged Content: A QoS2 server accepts tags for the content served. Content can be tagged as either private or public, with the default tag being conservatively set to private. The tags enable the web-server to determine which content should be sent over HTTP and which over HTTPS. These tags can be statically or dynamically assigned. Statically assigning content may be cumbersome and overwhelming, fortunately, there are a number of techniques that are immediately amendable to dynamic create tag. For example, template-extraction techniques [22] can be used to automatically determine templates and subsequently tag objets in the template as public.

Content Checksum: A QoS2 server calculates and maintains a checksum for each content that is tagged as public. This checksum is sent to the client alongside

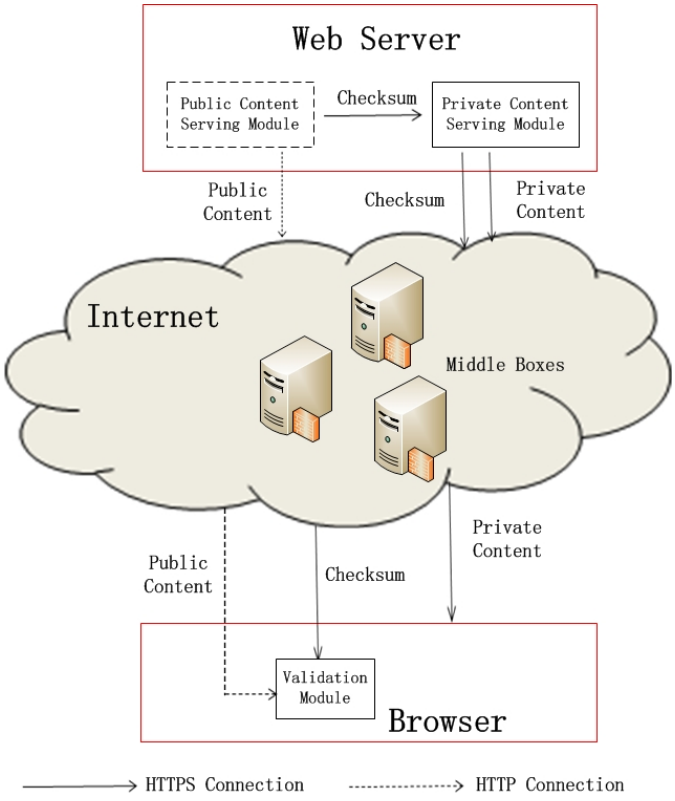

Figure 1: *QoS2 architecture.*

the object. The checksum enables the client to verify that the objects sent over HTTP have not been compromised. This checksum can be calculated a priori or calculated at run-time. There are a few trade-offs between the two: a priori calculation ensures that there is no additional latency for calculating the checksums but at the risk of stale checksums. Where-as, runtime calculation ensures that the checksums are never stale but incurs additional latency. As part of future work, we intend to explore both alternatives and make recommendations as to when one should be used over the other.

Control Channel: Unlike SPDY and HTTP/2.0 servers which maintain one connection, the QoS2 server maintains at least two connections to every client: a secure connection (over HTTPS) and an unsecured one (over HTTP). The server uses the secure connection to transfer both checksums and private content. This ensures that the checksums are not tampered with.

4.2 QoS2 Enabled clients

A QoS2 enabled client is a web-browser that understands the QoS2 protocol. Namely, the browser understands the use of the HTTPS connection as a control channel: HTTP requests are sent over the control channel, as are checksums and private content. In addition to the normal behavior, the QoS2 client also calculates a checksum for the content received over HTTP (labeled 2 in Figure 2), and the validation module at client side verifies the computed checksums against the checksums received over the control channel(labeled 3 in Figure 2). Validated content is passed to the normal browser processing engine to be parsed, interpreted, and painted to the screen. Alternatively, checksums may fail for a number of reasons including: an attack on the HTTP channel or stale information. When a checksum check fails, the client re-requests the content (labeled 4 in Figure 2), and the web-server responds with the content over the HTTPS connection (labeled 5 in Figure 2). As part of future works we intend to develop more extensive protocols to disambiguate the two situations and determine when it is safe to resend through HTTP.

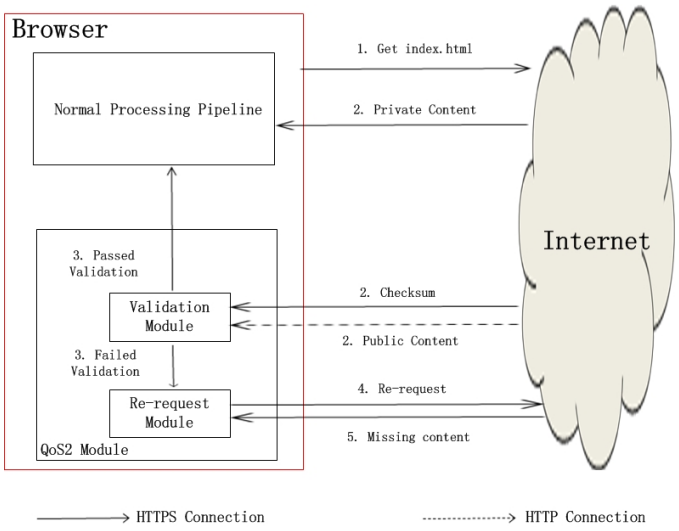

Figure 2: *Client browsing in QoS2.*

5. EMPIRICAL VALIDATION OF QoS2

In this section, we attempt to quantify the benefits of QoS2. To do this, we download four websites from the Alexa-Top 100 list and manually tag content as either private or public. First we tag all content as private,

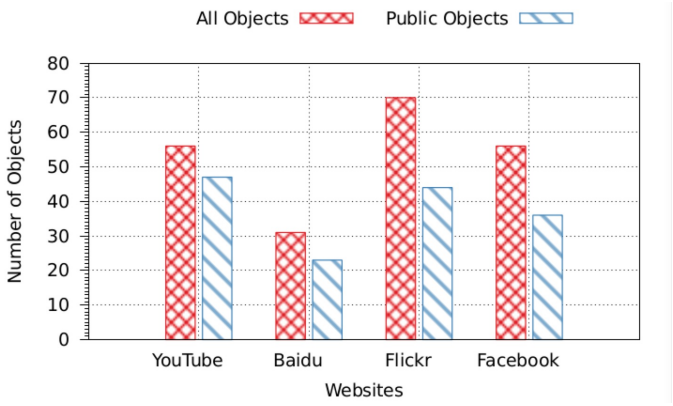

Figure 3: *Classification of web objects based on privacy.*

then manually prune out public content. Note, we do not claim to tag all public content, thus our numbers provide a lower bound on the potential benefits of QoS2.

In Figure 3, we present a summary of our classification. We observe that all websites contain a non-negligible fraction of public content with the actual number varying between 35% to 89%. Regardless, we note that these numbers are indicative of potential gains.

To determine the actual gains, we emulated QoS2 on our private testbed and examined page load times. In our testbed, we setup a web server and cache-proxy both hosted on different servers with 2.00 GHz CPU and 16 GB of RAM. The web server, caches, and the web browsers are all interconnected through a 1G Local Area Network.

To emulate WAN conditions, we use the Linux *tc* command [23] on the origin web server and the caches to inject varying amounts of latency. We vary the latencies between the origin servers, the cache-proxies, and the web-client. Varying all latencies allows us to understand how QoS2's benefits are impacted by the placement and location of the middleboxes.

In Figure 4, we compare the load time for varying latencies to the origin server and the potential proxies. To determine the latency to the origin web server, we perform latency tests to the Alexa top 100 websites and select the 10th percentile (25ms), the 50th percentile (75ms) and the 95th percentile (400ms). For latencies to the cache, we use trace-routes to determine latencies with the Metro-Area network. We believe that this can simulate the real network environment well. Then we allocate the private or public content at two servers, via HTTP and HTTPS respectively, and collect the latency to load the whole page under different conditions.

This latency allows us to understand benefits under wired networks (with low latencies) and cellular networks (with high latencies). We observe a 20% performance improvement in low latency networks and a 70% performance improvement in high latency networks. We observe that the improvements are function of both the dependencies between objects and the size of the public objects.

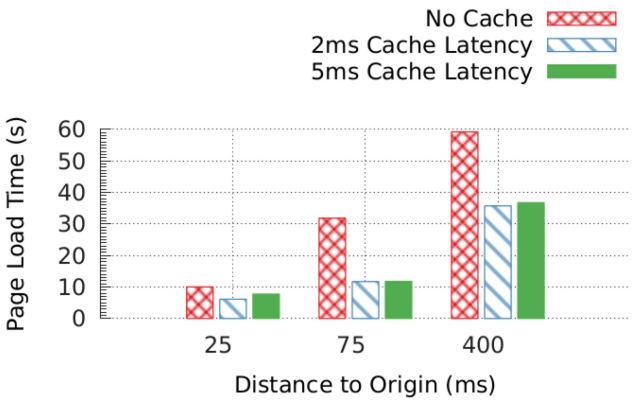

Figure 4: *Analysis of page load times under QoS2.*

6. DISCUSSION

Deployment Challenges: QoS2 requires changes to both the web-servers and the web-browsers. Fortunately, these changes can be easily accomplished with minor to changes to both. More over, given that SPDY current supports priority-based tags for content and performs interesting optimizations to improve load time based on these tags. We believe that our proposed tagging mechanism can piggy back on existing mechanisms within the browser.

Tagging Content: The greatest deployment hurdle lies in tagging content as either private or public. As discussed earlier, these burdens can be significantly reduced by applying variations on traditional template extraction techniques [22]. All content identified as part of a template can be tagged as public content.

Incentives for Adoption: The incentives for ISPs and end-users to leverage infrastructure and tools that supports QoS2 is quite clear. As both ISPs and end-users are poised to reap tremendous benefits from the introduction of QoS2. More concretely, ISPs may get a 30% reduction in traffic and users may gain at least a 50% improvement in page load times. Although content-provider may be resistant, many content-providers are determined to reduce page load times [24] and thus are aptly motived to adopt techniques like QoS2.

Security Oriented Middleboxes: QoS2 explicitly focuses on performance oriented middleboxes. Yet more worked is required to enable QoS2 to support security oriented middleboxes, such as, intrusion detection systems and deep packet inspection systems

Browser cache: Browser caches operate at the application level and are thus unaffected by TLS encryption. Fortunately, browser caches are orthogonal to in-network caches with both levels of caching provide different benefits. QoS2 is explicitly designed to re-introduce the benefits provided by in-network caches.

Future transport protocols: In light of recent studies on web performance, a number of novel transport protocols are being developed to improve the performance of secure web browsing, e.g. Quic. Yet, these protocols fail to explicitly incorporate middleboxes. Future work could evaluate QoS2 in the context of such novel protocols.

7. CONCLUSION AND FUTURE WORKS

In this paper, we argued that instead of coarsely serving content exclusively over HTTP or HTTPS, content-provider should be able to determine at a finer-granularity which content is served over HTTP or HTTPS. We presented a strawman that enables content providers and users to safely browse mixed-context websites. We showed, through empirical evaluations, that this approach can help improve page load times.

As part of future works, we are planning to extend

QoS2 in a number of ways. First, by exploring techniques to disambiguate between stale checksums and attacks on the HTTP channel. Our current approach conflates both and reacts in a heavy handed manner. Second, by implementing checksum verification in the browser. Third, by enhancing the nginx and Apache platforms to support QoS2. Finally, we hope to evaluate QoS2 under a variety of workloads, to understand how the benefits of QoS2 are impacted by the choices made by content-providers.

8. ACKNOWLEDGEMENTS

We thank our anonymous reviewers for their thoughtful comments and feedback. This work was supported by NSF Award CSR-1409426.

9. REFERENCES

[1] Ball, James and Borger, Julian and Greenwald, Glenn. Revealed: How US and UK Spy Agencies Defeat Internet Privacy and Security. *The Guardian*, 6, 2013.

[2] Tim Dierks. The transport layer security (TLS) protocol version 1.2. 2008.

[3] David Naylor, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio Munafò, Konstantina Papagiannaki, and Peter Steenkiste. The Cost of the "S" in HTTPS. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, pages 133–140, New York, NY, USA, 2014. ACM.

[4] Xing Xu, Yurong Jiang, Tobias Flach, Ethan Katz-Bassett, David Choffnes, and Ramesh Govindan. Investigating Transparent Web Proxies in Cellular Networks. *Passive and Active Measurement Conference*, 2015.

[5] Jeffrey Erman, Alexandre Gerber, Mohammad T. Hajiaghayi, Dan Pei, and Oliver Spatscheck. Network-aware Forward Caching. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 291–300, New York, NY, USA, 2009. ACM.

[6] Jeffrey Erman, Alexandre Gerber, Mohammad Hajiaghayi, Dan Pei, Subhabrata Sen, and Oliver Spatscheck. To Cache or not to Cache: The 3G case. *Internet Computing, IEEE*, 15(2):27–34, 2011.

[7] S. Loreto, J. Mattsson, R. Skog, H. Spaak, G. Gus, D. Druta and M. Hafeez. Explicit Trusted Proxy in HTTP/2.0. *IETF Internet-Draft*, 2014.

[8] Hodges, Jeff and Jackson, Collin and Barth, Adam. HTTP Strict Transport Security (HSTS). *IETF Internet-Draft*, 2012.

[9] Michael Kranch and Joseph Bonneau. Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning. *Network and Distributed System Security (NDSS) Symposium*, 2015.

[10] Qi Huang, Ken Birman, Robbert van Renesse, Wyatt Lloyd, Sanjeev Kumar, and Harry C. Li. An Analysis of Facebook Photo Caching. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 167–181, New York, NY, USA, 2013. ACM.

[11] Michael Butkiewicz, Harsha V. Madhyastha, and Vyas Sekar. Understanding Website Complexity: Measurements, Metrics, and Implications. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 313–328, New York, NY, USA, 2011. ACM.

[12] Ping Chen, Nick Nikiforakis, Lieven Desmet, and Christophe Huygens. A Dangerous Mix: Large-scale analysis of mixed-content websites. In *Proceedings of the 16th Information Security Conference (ISC)*, 2013.

[13] Xin Jin, Li Erran Li, Laurent Vanbever, and Jennifer Rexford. SoftCell: Scalable and Flexible Cellular Core Network Architecture. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 163–174, New York, NY, USA, 2013. ACM.

[14] S. Loreto, J. Mattsson, R. Skog, H. Spaak, G. Bourg, D. Druta and M. Hafeez. Explicitly authenticated proxy in HTTP/2.0. *IETF Internet-Draft*, July 2014.

[15] D. McGrew, D. Wing, Y. Nir and P. Gladstone. TLS Proxy Server Extension. *IETF Internet-Draft*, July 2012.

[16] Thomas Fossati, Vijay K. Gurbani and Vladimir Kolesnikov. Love all, trust few: On trusting intermediaries in HTTP. In *ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*. ACM, 2015.

[17] Andrea Bittau, Michael Hamburg, Mark Handley, David Mazières, and Dan Boneh. The case for ubiquitous transport-level encryption. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 26–26, Berkeley, CA, USA, 2010. USENIX Association.

[18] A. Bittau, D. Boneh, M. Hamburg, M. Handley, D. Mazieres and Q. Slack. Cryptographic protection of TCP streams. *IETF Internet-Draft*, July 2014.

[19] Sneha Kasera, Semyon Mizikovsky, Ganapathy S. Sundaram, and Thomas Y. C. Woo. On Securely Enabling Intermediary-based Services and Performance Enhancements for Wireless Mobile Users. In *Proceedings of the 2Nd ACM Workshop on Wireless Security*, WiSe '03, pages 61–68, New York, NY, USA, 2003. ACM.

[20] Yongguang Zhang and Bikramjit Singh. A Multi-Layer IPSEC Protocol. In *USENIX Security Symposium*, volume 9, 2000.

[21] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. BlindBox: Deep Packet Inspection over Encrypted Traffic. In *Proceedings of the 2015 ACM SIGCOMM Conference*, 2015.

[22] Ziv Bar-Yossef and Sridhar Rajagopalan. Template Detection via Data Mining and Its Applications. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 580–591, New York, NY, USA, 2002. ACM.

[23] Bert Hubert. TC–Linux man page. http://lartc.org/manpages/tc.txt, 2010.

[24] Ashish Vulimiri, Philip Brighten Godfrey, Radhika Mittal, Justine Sherry, Sylvia Ratnasamy, and Scott Shenker. Low Latency via Redundancy. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 283–294, New York, NY, USA, 2013. ACM.