The Cost of the "S" in HTTPS

The Cost of the "S" in HTTPS

David Naylor⋆, Alessandro Finamore†, Ilias Leontiadis‡, Yan Grunenberger‡,
Marco Mellia†, Maurizio Munafò†, Konstantina Papagiannaki‡, and Peter Steenkiste⋆

⋆Carnegie Mellon University †Politecnico di Torino ‡Telefónica Research

{dnaylor, prs}@cs.cmu.edu {finamore, mellia, munafo}@tlc.polito.it
{ilias, yan, dina}@tid.es

ABSTRACT

Increased user concern over security and privacy on the Internet has led to widespread adoption of HTTPS, the secure version of HTTP. HTTPS authenticates the communicating end points and provides confidentiality for the ensuing communication. However, as with any security solution, it does not come for free. HTTPS may introduce overhead in terms of infrastructure costs, communication latency, data usage, and energy consumption. Moreover, given the opaqueness of the encrypted communication, any in-network value added services requiring visibility into application layer content, such as caches and virus scanners, become ineffective.

This paper attempts to shed some light on these costs. First, taking advantage of datasets collected from large ISPs, we examine the accelerating adoption of HTTPS over the last three years. Second, we quantify the direct and indirect costs of this evolution. Our results show that, indeed, security does not come for free. This work thus aims to stimulate discussion on technologies that can mitigate the costs of HTTPS while still protecting the user's privacy.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols

Keywords

HTTP 2.0; HTTPS; TLS; privacy; security; Web proxies

1. INTRODUCTION

The HyperText Transfer Protocol (HTTP) was first introduced in the early '90s. Since then, the Internet has changed significantly, becoming a vital infrastructure for communication, commerce, education, and information access. HTTPS, the secure version of HTTP, runs HTTP on top of SSL/TLS[1]. While originally geared toward services that require data confidentiality or authentication between

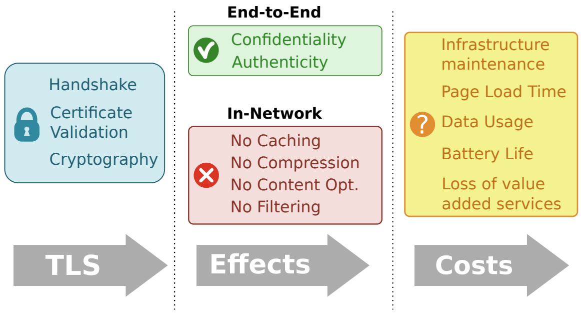

Figure 1: The HTTPS adoption impact chain.

client and server, like online banking or e-commerce, the increasing personalization of the web has led to a number of other services adopting HTTPS, such as GMail, Facebook, and even YouTube. Furthermore, HTTP is on the verge of a new milestone: the standardization of HTTP 2.0 [10] is slated for the end of 2014. Some discussions assume TLS will be used for *all* connections, mirroring a fundamental design decision of SPDY [14], which was used as the starting point for HTTP 2.0.

Given users' growing concerns about security and privacy on the Internet, adopting encryption by default in all HTTP communication sounds like a good idea. However, security always comes at a cost, and HTTPS is no different (graphically depicted in Fig. 1). In this paper, we aim to categorize and quantify the cost of the "S" in HTTPS.

First, we look at the way HTTPS adoption has evolved over the past three years. Such an analysis is important because, besides quantifying trends, it sheds light on the cost of deploying HTTPS for web services, a cost that seems to be diminishing: 50% of web traffic flows today are secure, including, for the first time, large content (e.g., 50% of YouTube streaming flows are over HTTPS).

Second, we study how TLS impacts latency, data consumption, and battery life for clients. HTTPS requires an additional handshake between the client and the server in addition to the added computational cost of cryptographic operations. We study how significant these costs are for fiber, Wi-Fi, and 3G connections.

Lastly, while encryption provides a clear value to the end user in terms of confidentiality and authentication, it could have implications that are harder to assess. Over the past 15 years, an increasing number of network functionalities have been performed by transparent and explicit middleboxes, aiming to reduce the amount of backbone traffic, compress content before transmission on expensive wireless links, fil-

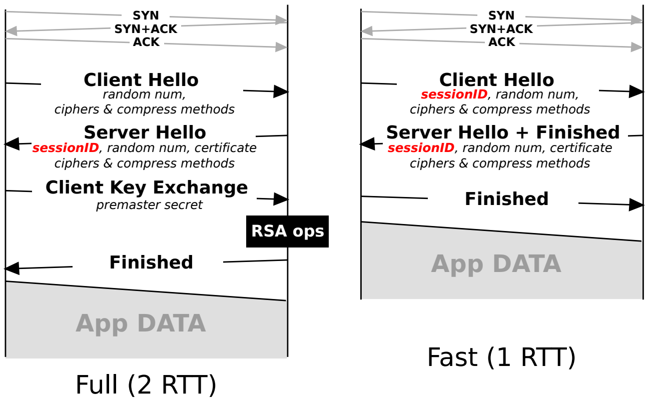

Figure 2: SSL/TLS negotiation.

ter inappropriate/undesired content, and protect users and organizations from security threats. These boxes suddenly become blind in the presence of encryption. We show that there are clear cases where losing such functionality may not only harm network efficiency, but also increase latency by more than 50% and consume up to 30% more energy for 3G mobile devices.

Using three different datasets, captured in residential and cellular networks, as well as controlled experiments, our work shows that: (i) HTTPS usage is increasing despite potential deployment costs, (ii) HTTPS has a perceptible impact on clients in terms of latency, (iii) its data overhead seems to be limited, (iv) it could lead to significantly increased battery consumption for large objects.

This creates a complex tradeoff that depends on many factors, including context and personal preference. While specific workloads will always be served through HTTPS (e.g., financial transactions), there are a number of applications where encryption could be optional or the service could be assisted by *trusted proxies*, which bring back the benefits of value-added services, as described in a recent IETF proposal [9]. How to best manage this tradeoff is an open question that challenges the research community. We hope this paper stimulates discussion in that direction.

2. HTTPS OVERVIEW

SSL/TLS is the standard protocol for providing authenticity and confidentiality on top of TCP connections. Today, it is used to provide a secure version of traditional protocols (e.g., IMAP, SMTP, XMPP, etc.); in particular, the usage of HTTP over TLS is commonly known as HTTPS.

Each TLS connection begins with a *handshake* between the server and the client. In this handshake, the Public Key Infrastructure (PKI) suite is used to authenticate the server (and sometimes the client) and to generate cryptographic keys to create a secure channel for data transmission.

Fig. 2 (left) sketches the steps in a *full* TLS negotiation. In this scenario, the client and the server incur different costs. On the server side, the primary cost is computing the session key. This involves complex public key cryptography operations (typically RSA), limiting the number of connections per second the server can support [2, 3, 4]. For clients, the major cost is latency. This depends on (i) server performance, (ii) the distance between client and server since a *full* negotiation requires 2 RTTs (3 including the TCP handshake), and (iii) the latency to verify the server's certificate with the PKI (e.g., an OSCP/CRL check).

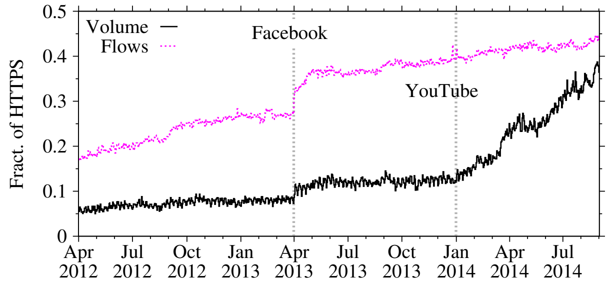

Figure 3: Evolution of HTTPS volume and flow shares over 2.5 years. Results from Res-ISP dataset. Vertical lines show the transition to HTTPS for Facebook and YouTube.

Unsurprisingly, a few optimizations have been proposed to reduce handshake costs. Hardware accelerators, GPU architectures [12], or "rebalancing" the RSA computations [3] can easily boost server performance by a factor of 10. Also, the TLS standard provides a *fast* negotiation mechanism, shown in Fig. 2 (right). In this case a `SessionID` is used to retrieve a previously negotiated session key, (i) avoiding the cost of creating a new session key and (ii) reducing the handshake to 1 RTT (2 including the TCP handshake). Note that the adoption of fast negotiation is controlled by the server; as we see in Sec. 4, only some services deploy it.

3. HTTPS USAGE TRENDS

A common belief is that deploying HTTPS increases infrastructure costs (to accommodate the resulting computational, memory, and network overhead) in addition to the cost of certificates (up to $1,999/year each[1]). Thus, one would expect services to carefully deploy HTTPS only when needed. To test this, we examine recent HTTPS usage trends. We collected per-flow logs from a vantage point monitoring the traffic of about 25,000 residential ADSL customers of a major European residential ISP ("Res-ISP"). The vantage point runs Tstat [7], which implements a classifier supporting both HTTP and TLS identification. For TLS traffic, Tstat parses the `ClientHello` and `ServerHello` TLS handshake messages to extract (i) Server Name Indication (SNI), i.e., the hostname to which the client is attempting to connect, and (ii) Subject Common Name (SCN) carried in the server certificate, i.e., the name the server itself presents. Tstat is also able to identify the presence of SPDY in the TLS connections. In the following, we use this rich ISP dataset to characterize the evolution of HTTPS usage. At the time of writing, HTTPS and HTTP combined represent 75% of all TCP traffic (by volume) in Res-ISP.

Fig. 3 reports the evolution of the HTTPS traffic share from April 2012 to September 2014. Both volume and flow shares are shown. The growth of HTTPS adoption is striking, with the HTTPS flow share more than doubling in two years. In September 2014, 44.3% of web connections already use HTTPS.[2] The sharp bump in April 2013 is due to Facebook enabling HTTPS by default for all users [13].

[1] https://www.symantec.com/page.jsp?id=compare-ssl-certificates

[2] Curiously, only 5.5% of flows successfully negotiated SPDY, despite 55% of clients offering the option. This highlights

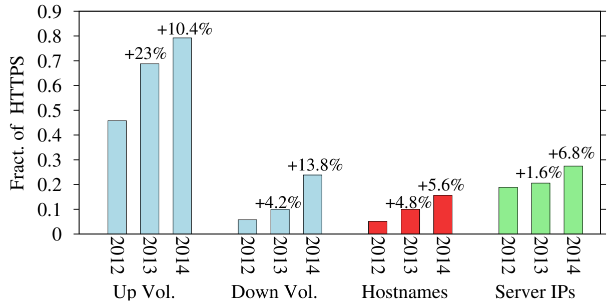

Figure 4: Comparing HTTPS shares over three one-week periods in the Res-ISP dataset. Percentages highlight year-to-year growth.

Looking at volume, we see a much slower growth. Intuitively, one would expect that most HTTPS flows carry small, privacy-sensitive objects. We see this until January 2014 when YouTube began delivering video content over HTTPS, clearly increasing the HTTPS volume share. As of September 2014, as much as 50% of YouTube's aggregate traffic volume is carried over HTTPS.

Fig. 4 further details HTTPS trends with respect to upload volume, download volume, number of hostnames, and number of server IP addresses. We compare the first week of April in 2012, 2013 and 2014 (results are consistent for other weeks). Percentages show the year-to-year increase. The growth of HTTPS is again evident. For instance, HTTPS accounts for 80% of the upload volume in 2014; it was only 45.7% in 2012. This reflects the fact privacy-sensitive information tends to be uploaded using HTTPS more and more. Interestingly, the fraction of data downloaded using HTTPS is smaller than the fraction uploaded. However, YouTube's shift to HTTPS in 2014 dramatically changed the landscape: HTTPS download volume more than doubled compared to 2013. Fig. 4 also highlights a constant year-by-year increase for both the fraction of hostnames and server IP addresses accessed using TLS. Interestingly, 72% of the TLS hostnames are accessed exclusively over TLS in 2014.

These results clearly show that, despite the perceived costs, services are rapidly deploying HTTPS. This shift is undoubtedly related to the recent attention toward guaranteeing end-users privacy. However, this may also be an indication of increasingly manageable infrastructural costs. This is consistent with the report from the GMail team after their switch to HTTPS: "On our production frontend machines, SSL/TLS accounts for less than 1% of the CPU load, less than 10KB of memory per connection and less than 2% of network overhead."[3] Reports from Facebook are similar.[4]

Takeaway: *HTTPS accounts for 50% of all HTTP connections and is no longer used solely for small objects, suggesting that the cost of deployment is justifiable and manageable for many services.*

that SPDY is only supported by a handful of (popular) services, namely Google and Facebook.

[3] https://www.imperialviolet.org/2010/06/25/overclocking-ssl.html

[4] http://lists.w3.org/Archives/Public/ietf-http-wg/2012JulSep/0251.html

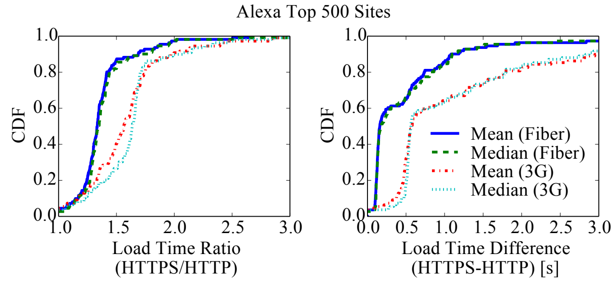

Figure 5: Webpage load time inflation for the Alexa top 500.

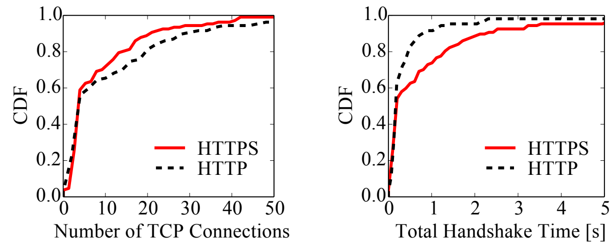

Figure 6: Number of TCP handshakes (left) and total handshake time (right) for the Alexa top 500 (fiber).

4. WEBPAGE LOAD TIME

We now investigate how much HTTPS affects load time, an important quality-of-experience metric for web browsing.

Overall Page Load Time: First we quantify the HTTPS page load time overhead through active experiments. We load each of the top 500 Alexa sites 20 times, first using HTTP and then using HTTPS. For the download, we use PhantomJS[5], a fully-fledged headless browser running on a Linux PC. From each run, we extract the average and median load times. The test PC is connected first using a 3G USB modem and then via fiber, two typical real-world environments. The local cache is cleared between page loads.

Results are reported in Fig. 5. Plots show the Cumulative Distribution Function (CDF) of the ratio of HTTPS to HTTP page load time (left) and the absolute difference (right). The benchmark shows that using HTTPS significantly increases load time. This is especially evident on 3G, where the extra latency is larger than 500 ms for about 90% of websites, and 25% of the pages suffer an increase of at least 1.2 s (i.e., an inflation factor larger than 1.5x). On fiber, the extra latency is smaller; still, for 40% of the pages, HTTPS adds more than 500 ms (1.3x).

We also captured an HTTP Archive (HAR) for each page, which contains statistics about the individual connections used to load the page. Fig. 6 (left) shows 40% of the sites open fewer TCP connections when loaded over HTTPS compared to HTTP. Even so, Fig. 6 (right) shows nearly half of the sites spend *more time* establishing those connections; the increased time per handshake is caused primarily by TLS negotiation overhead (see below). By examining the HARs, we see that many of the sites using fewer connections serve fewer objects (1–3 fewer on average, but in some cases upwards of 100 fewer) and do so from fewer hosts (typically 1–2 fewer, but in some cases up to 40 fewer). We suspect these are intentional changes to the HTTPS version of the site designed to avoid costly TLS handshakes.

[5] http://phantomjs.org/

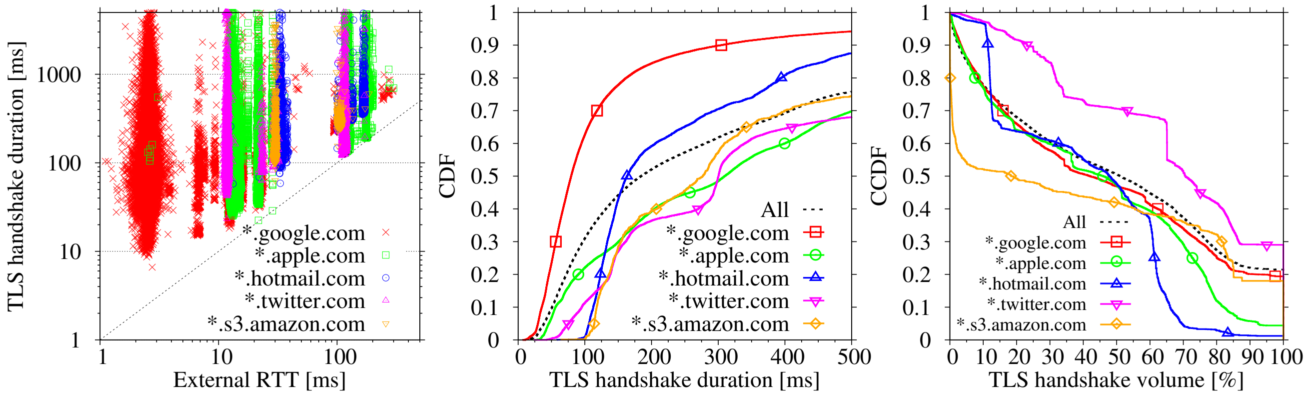

Figure 7: Quantifying TLS handshake costs. Scatter plot of the TLS handshake duration with respect to server distance (left). TLS handshake duration CDF (center). Ratio of TLS handshake bytes with respect to total TCP connection bytes CCDF (right).

TLS Handshake delay cost: Webpage load time has been an active area of research [8, 5, 6, 15, 11]; understanding the exact cause(s) of the inflation noted above is quite complex and out of scope for this work. However, it is still interesting to understand whether the overall page latency is primarily affected by network latency or by protocol overhead. To better understand this, we modified Tstat to extract the (i) duration and (ii) number of bytes of each TLS handshake from a one-hour pcap trace collected on April 3rd 2014 from Res-ISP. About 1 million TLS flows are present.

Fig. 7 (left) shows a scatter plot of the TLS handshake duration with respect to the minimum external RTT (i.e., the RTT between the vantage point and the remote server[6]). The external RTT is reasonably representative of the distance to the remote server. For this analysis we selected popular services as representative cases (we classify based on the Server Name Indication in the `ClientHello`).

First, all services exhibit vertical clusters of points, which likely reflect the different data centers offering that service. For instance, when the external RTT is larger than 100 ms, the server is outside Europe. More interestingly, no matter how close the servers are, all clusters contain samples with very high TLS handshake duration, i.e., up to several seconds. To better capture this effect, Fig. 7 (center) reports the CDF of the TLS handshake duration for individual services and for the traffic aggregate (black dotted line). Google services (which are also the closest) exhibit the smallest TLS negotiation delay, though 10% of measurements are more than 300 ms. Since a full TLS handshake requires at least 2 RTT (1 RTT in case of `SessionID` reuse), services handled by U.S. servers (e.g., Hotmail, Twitter, Amazon S3) experience huge extra costs. For instance, for Twitter, negotiation takes over 300 ms for more than 50% of the HTTPS connections. In general, 5% of requests experience a handshake at least 10 times longer than the RTT. This might be due to client or server overhead, network congestion, or a slow OCSP check.

Looking closer, we find that 4% of clients experience at least one connection with a TLS handshake duration higher than 300 ms. For such connections, 50% (75%) have an internal RTT (i.e., the RTT between the vantage point and the end-user device) of 51 ms (97 ms). The same holds true with less conservative thresholds (e.g., 1 second). This demonstrates that even clients with good network connectivity can still significantly suffer from TLS handshake overhead. TLS fast negotiation can help to reduce the handshake latency, but we find this being used in only 30% of the connections. We speculate this represents a lower bound, but, unfortunately, based on available data, we cannot assess the achievable upper bound obtained from a wider adoption of TLS fast negotiation.

Takeaway: *The extra latency introduced by HTTPS is not negligible, especially in a world where one second could cost 1.6 billion in sales.*[7]

5. DATA USAGE

HTTPS also impacts the volume of data consumed due to (i) the size TLS handshake and (ii) the inability to utilize in-network caches and compression proxies.

TLS Handshake Data Cost: The impact of the TLS handshake overhead depends on how much use the connection sees; the more data transferred, the lower the relative cost of the negotiation packets. Fig. 7 (right) reports the Complementary Cumulative Distribution Function (CCDF) of the ratio between TLS handshake size and total bytes carried in the TCP connection. Results refer to a peak hour in April 2014 for the Res-ISP dataset and are consistent with other time periods. We see that many TLS connections are not heavily used. In fact, for 50% of them, the handshake represents more than 42% of the total data exchanged. However, some services, like those running on Amazon S3, do actually use connections efficiently, reducing the impact of the negotiation cost. Some services also try to mask negotiation latency by "pre-opening" connections before they actually need to send data. In this case, the negotiation overhead is 100% if the connection is never used. This is captured in the rightmost part of Fig. 7 (right), which also highlights how this optimization is heavily used by Google, Amazon S3, and Twitter, but is not by Hotmail and Apple

[6] Tstat extracts RTT per-flow statistics monitoring the time elapsed between TCP data segments and the corresponding TCP ACK.

[7] http://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales

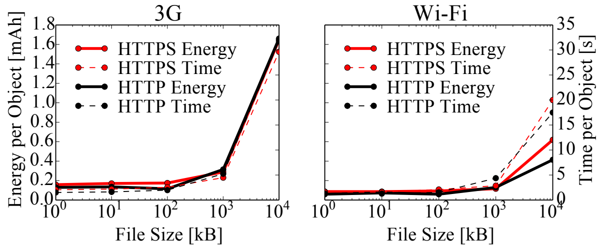

Figure 8: Energy consumption on 3G (left) & Wi-Fi (right).

services. Despite all this variability, the average TLS negotiation overhead amounts to 5% of the total volume in this dataset.

In-Network Proxies: HTTPS prevents in-network content optimizations, like proxies that perform compression and caching. To evaluate the impact of this loss, we analyze logs from two production HTTP proxies for mobile networks: Transp-Proxy and OptIn-Proxy. Transp-Proxy refers to a transparent proxy in a major European mobile carrier serving more than 20 million subscribers. OptIn-Proxy, on the other hand, is an explicit proxy serving 2000 mobile subscribers daily in a major European country. For Transp-Proxy, we analyze the past two years of traffic and for OptIn-Proxy we consider a week-long trace from May 2014.

Caching (ISP Savings): An ISP can save upstream bandwidth by serving static content from its own transparent cache. In the Transp-Proxy dataset, the average cache hit ratio over the past two years was 14.9% (15.6% of the total data volume), amounting to savings of 2 TB per day for a single proxy instance serving 3 million subscribers. For OptIn-Proxy, we see daily savings of 1.3 GB, which, if scaled up to the Transp-Proxy population, matches the observed Transp-Proxy savings.

We also witnessed a decrease in cache efficiency: the cache hit ratio of Transp-Proxy dropped from 16.8% two years ago to 13.2% in June 2014. Based on our analysis, it is not easy to conclude how much of the decreasing effectiveness of caching is related to the adoption of HTTPS and how much is caused by the increased personalization of web traffic. Either way, savings of this size can be still be significant to network operators; such savings will be totally eliminated if content delivery moves entirely to HTTPS.

Compression (Users Savings): Before returning content to users, web proxies typically apply lossless (e.g., gzip) compression to objects and, in more aggressive settings, even scale or re-encode images. This functionality is particularly helpful in cellular networks where the capacity is limited and where users often have restrictive data allowances. The Transp-Proxy trace shows a compression ratio of 28.5% (i.e., the last-mile of the network and the users save one-third of the original data size). In terms of average volume, this amounts to only 2.1 MB per user per day (on average a mobile user downloads less than 10 MB per day). For heavy users, though, this may translate to significant savings (e.g., more than 300 MB per month).

Takeaway: *Most users are unlikely to notice significant jumps in data usage due to loss of compression, but ISPs stand to see a large increase in upstream traffic due to loss of caching.*

	Total Energy (mAh)		Avg. Current (mA)	
	3G	*Wi-Fi*	*3G*	*Wi-Fi*
HTTP	210.8	175.7	633	520
HTTPS	217.7	178.0	653	531

Table 1: Energy consumed loading CNN homepage.

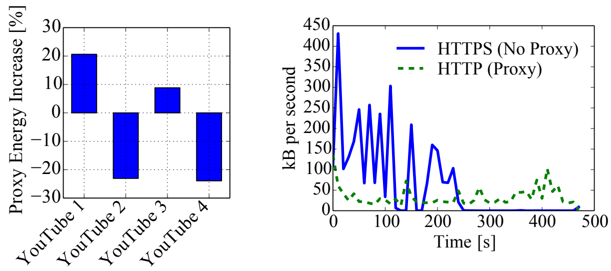

Figure 9: Comparing YouTube video playback over HTTP (with proxy) and HTTPS (without proxy): energy consumption increase when using HTTP+proxy (left) and download rate over time for one video (right). Results for 3G.

6. BATTERY LIFE

HTTPS has the potential to negatively impact battery life (particularly on mobile devices) due to (i) the extra CPU time required for the cryptographic operations and (ii) increased radio uptime due to longer downloads.

Synthetic Content: To measure the raw energy overhead of HTTPS, we instrumented a Galaxy S II with a power meter that samples the current drawn by the phone every 200 μs. We used the test phone to download synthetic objects over 3G and Wi-Fi (i.e., an access point connected to a fiber link). Objects range in size from 1 kB to 1 MB and are hosted on a web server under our control. Objects are downloaded 100 times each over HTTP and HTTPS using `curl` (compiled for Android). We configured the server to deliver traffic avoiding any proxy cache along the path.[8] During our tests, the screen was on at its minimum brightness.

Fig. 8 shows both average time (right y-axis) and energy (left y-axis) to complete each download. It is immediately clear that energy consumption is strongly correlated to download time; this is not surprising, as leaving the radio powered up is costly. (We also see a slight overhead for large objects over HTTPS on Wi-Fi but not on 3G, but we were unable to precisely determine the cause. The difference is less than one standard deviation.) The key takeaway here is, download time aside, we do not see a noticeable overhead due to cryptographic operations.

Real Content: We complement the previous analysis by loading real content. We mirror the CNN homepage on our controlled webserver and download it 50 times using Chrome for Android over HTTP and HTTPS (enforcing 20 seconds of wait between consecutive downloads). Results are listed in Table 1 (numbers presented are cumulative for all 50 loads). As in the previous benchmark, HTTPS tests do not show an appreciable increase in energy costs.

In a second experiment, we play four 5- to 12-minute YouTube videos. Since the YouTube app does not deliver

[8]The 3G carrier used in the experiment runs transparent proxies acting on traffic to port 80. By configuring the webserver to listen on a different port, we bypass the cache.

video content over HTTPS for mobile devices (nor does the YouTube mobile site), we first force the phone to load the desktop version of the YouTube portal. Over Wi-Fi, there was no difference; on 3G, on the other hand, our network's Web proxy significantly impacted the HTTP results. For two videos, playback over HTTP (with proxy) consumes nearly 25% less energy than over HTTPS (without proxy); for the other two, 10%–20% more (Fig. 9 left).

The differences are caused by two distinct proxy behaviors. First, the proxy throttles the download rate (Fig. 9 right) to reduce congestion and avoid wasting bandwidth if the user abandons the video. Without the proxy (HTTPS), the whole video loads immediately and the radio sleeps while it plays. With the proxy (HTTP), the download is slow and steady, lasting the duration of the video. Without the opportunity to sleep, playback over HTTP consumes more energy. Second, the proxy injects javascript into the pages it returns, which, among other things, rewrites the URLs sent to YouTube to request encodings and qualities more appropriate for mobile devices. For YouTube 2 and YouTube 4, the player requests the content in the `webm` format, for which our phone does not have hardware decoding support; the proxy changes `webm` to `mp4`, which our phone can decode in hardware. The benefit of hardware decoding outweighed the cost of radio uptime.

Of course, these numbers should be taken with a grain of salt, since using the desktop version of YouTube on a phone is unrealistic (but they are still relevant to PC users connecting via USB modem or tethering). We played the same four videos with YouTube's mobile portal in addition to two new videos from Vimeo's mobile site and verified that the mobile players request `mp4` from the start, so the proxy does not help decrease decoding costs. The mobile video was still throttled.

Stepping back, we see these results as concrete examples of a proxy helping and a proxy hurting end users, suggesting that (1) operators should think carefully about how they configure middleboxes and (2) the community should think carefully about shutting them out by switching to HTTPS by default.

Takeaway: *HTTPS' cryptographic operations have almost no impact on energy costs, but the loss of proxies can significantly impact battery life (positively and negatively).*

7. LOSS OF VALUE-ADDED SERVICES

From the previous experiments it becomes evident that the most significant HTTPS overheads on client performance are increased latency and the inability to utilize "useful" middleboxes that may bring content closer to the client or reduce its size through network-aware compression. There are a number of other in-network services that would also be affected by ubiquitous adoption of HTTPS, but the effects of these losses are more difficult to quantify.

For example, ubiquitous encryption will render all deep packet inspection (DPI) boxes ineffective. The advantage that in-network DPIs have is the ability to observe the traffic of multiple clients at the same time to draw inferences, while access to application layer content allows them to block threats by searching for pre-defined signatures (e.g., a known malware binary). Unsophisticated DDoS attacks may still be detectable through statistical analysis of the HTTPS traffic, but application layer fingerprinting will have to be pushed to the client.

Simpler URL filtering will also become ineffective. For instance, a number of telecommunications providers today provide parental filtering through the use of explicit blacklists, such as the Internet Watch Foundation[9] list. Through direct communication with IWF, we found out that only 5% of their current blacklist is pure domains or sub-domains that could still be blocked in the presence of HTTPS. To maintain full functionality, the IWF list would have to again be moved to the client, where one can still observe the complete URL being accessed.

Other opt-in services offered by some providers are similarly affected, like content prioritization (e.g., postponing ad delivery) or blocking tracking cookies. (Interestingly, losing the ability to block tracking cookies hurts privacy, which is one of the goals of using TLS to begin with.)

Takeaway: *Though difficult to quantify, the loss of in-network services is potentially substantial; some of that functionality could be equally well performed on the client, while some may require a total rethink, like DPI-based Intrusion Prevention Systems (IPSes).*

8. CONCLUSION

Motivated by increased awareness of online privacy, the use of HTTPS has increased in recent years. Our measurements reveal a striking ongoing technology shift, indirectly suggesting that the infrastructural cost of HTTPS is decreasing. However, HTTPS can add direct and noticeable protocol-related performance costs, e.g., significantly increasing latency, critical in mobile networks.

More interesting, though more difficult to fully understand, are the *indirect* consequences of the HTTPS: most in-network services simply cannot function on encrypted data. For example, we see that the loss of caching could cost providers an extra 2 TB of upstream data per day and could mean increases in energy consumption upwards of 30% for end users in certain cases. Moreover, many other value-added services, like parental controls or virus scanning, are similarly affected, though the extent of the impact of these "lost opportunities" is not clear.

What *is* clear is this: the "S" is here to stay, and the network community needs to work to mitigate the negative repercussions of ubiquitous encryption. To this end, we see two parallel avenues of future work: first, low-level protocol enhancements to shrink the performance gap, like Google's ongoing efforts to achieve "0-RTT" handshakes.[10] Second, to restore in-network middlebox functionality to HTTPS sessions, we expect to see trusted proxies [9] become an important part of the Internet ecosystem.

Acknowledgements

The research leading to these results has received funding from the European Union under the FP7 Grant Agreement n. 318627 (Integrated Project "mPlane"), from NSF under award number CNS-1040801, and from DoD, Air Force Office of Scientific Research, National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a.

[9]https://www.iwf.org.uk

[10]http://blog.chromium.org/2013/06/experimenting-with-quic.html

9. REFERENCES

[1] The Transport Layer Security (TLS) Protocol — RFC 5246.

[2] G. Apostolopoulos, V. Peris, and D. Saha. Transport layer security: How Much Does It Really Cost? In *INFOCOM 1999*.

[3] C. Castelluccia. Improving Secure Server Performance by Rebalancing SSL/TLS Handshakes. In *USENIX Security Symposium*, 2005.

[4] C. Coarfa, P. Druschel, and D. S. Wallach. Performance Analysis of TLS Web Servers. *ACM Trans. Comput. Syst.*, 24(1):39–69, Feb. 2006.

[5] Y. El-khatib, G. Tyson, and M. Welzl. Can SPDY Really Make the Web Faster? In *IFIP Networking 2014*.

[6] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan. Towards a SPDY'Ier Mobile Web? In *CoNEXT '13*.

[7] A. Finamore, M. Mellia, M. Meo, M. M. Munafò, and D. Rossi. Experiences of Internet Traffic Monitoring with Tstat. *IEEE Network*, 25(3), 2011.

[8] P. Guy. Not as SPDY as You Thought. `http://goo.gl/RQkTwx`, June 2012.

[9] HTTPBis Working Group. Explicit Trusted Proxy in HTTP/2.0. `http://goo.gl/BUxQ22`, February 2014.

[10] IETF HTTPbis Working Group. Http/2. `http://http2.github.io/`.

[11] S. Ihm and V. S. Pai. Towards Understanding Modern Web Traffic. In *IMC 2011*.

[12] K. Jang, S. Han, S. Han, S. Moon, and K. Park. SSLShader: Cheap SSL Acceleration with Commodity Processors. In *NSDI 2011*.

[13] S. Renfro. Secure browsing by default. `http://goo.gl/B7U3jv`, July 2013.

[14] The Chromium Projects. Spdy. `http://www.chromium.org/spdy`.

[15] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. Demystifying Page Load Performance with WProf. In *NSDI 2013*.