# Client-side Energy Efficiency of HTTP/2 for Web and Mobile App Developers

Shaiful Alam Chowdhury, Varun Sapra, Abram Hindle
Department of Computing Science
University of Alberta, Edmonton, Canada
Email: {shaiful, vsapra, abram.hindle}@ualberta.ca

***Abstract*—Recent technological advancements have enabled mobile devices to provide mobile users with substantial capability and accessibility. Energy is evidently one of the most critical resources for such devices; in spite of the substantial gain in popularity of mobile devices, such as smartphones, their utility is severely constrained by the bounded battery capacity. Mobile users are very interested in accessing the Internet although it is one of the most expensive operations in terms of energy and cost.**

**HTTP/2 has been proposed and accepted as the new standard for supporting the World Wide Web. HTTP/2 is expected to offer better performance, such as reduced page load time. Consequently, from the mobile users point of view, the question arises: does HTTP/2 offer improved energy consumption performance achieving longer battery life?**

**In this paper, we compare the energy consumption of HTTP/2 with its predecessor (i.e., HTTP/1.1) using a variety of real world and synthetic test scenarios. We also investigate how Transport Layer Security (TLS) impacts the energy consumption of the mobile devices. Our study suggests that Round Trip Time (RTT) is one of the biggest factors in deciding how advantageous HTTP/2 is compared to HTTP/1.1. We conclude that for networks with higher RTTs, HTTP/2 has better energy consumption performance than HTTP/1.1.**

## I. Introduction

In recent years, the popularity of mobile devices (e.g., smartphones, and tablets) has dramatically increased. As of 2014, more than 1.4 billion smartphones were used globally [7], which induced a 70% increase in worldwide mobile data traffic [2]. With the recent technological advancements, there has been an exponential improvement in memory capacity and processing capability of mobile devices. Moreover, these devices come with a wide range of sensors and different I/O components, including digital camera, Wi-Fi, GPS, etc.—thus inspiring the development of more sophisticated mobile applications. These new opportunities, however, come with new challenges: the availability of these devices is severely constrained by their bounded battery capacity. A survey [50] has indicated that a longer battery life is one of the most desired features among smartphone users. Unfortunately, the advancement in battery technology is minimal compared to the improvement in computing abilities, thus amplifying the increasing importance of energy efficient application development [7].

The energy consumption of servers has also become a subject of concern for large data centers—consuming at least one percent of the world's energy [11]. Data centers must cater to the continually increasing demand for storage, networking and computation capabilities. In 2010, 4.3 terawatt-years of energy was consumed within the US by LAN switches and routers [38]. Energy efficiency was reported as one of the pivotal issues even by Google, facing the scale of operations, as cooling becomes a very important operational factor [8]. Another very important aspect of energy consumption is the environment: energy consumption has a detrimental effect on climate change, as most of the electricity is produced by burning fossil fuels [20]. Reportedly, 1000 tonnes of $CO_2$ is produced every year by the computer energy consumption of mid-sized organizations [27].

With the increased penetration of the mobile devices, the Internet usage on these smartphones is also mounting. According to eMarketer [15], it is expected that Internet access from mobile devices will dominate substantially by 2017. Accessing the Internet, however, is undoubtedly one of the most energy expensive use cases for mobile users [31].

Loading Web pages has become more resource intensive than ever, and this poses challenges to the inefficient HTTP/1.1 protocol which has served the Web for more than 15 years. HTTP/1.1, with only one outstanding request per TCP connection, has become unacceptable for today's Web, as a single page might require around 100 objects to be transferred [46]. HTTP/2—mainly based on SPDY, a protocol proposed and developed by Google [49]—is the second major version of HTTP/1.1 and is expected to overcome the limitations of its predecessor in the contexts of end-user perceived latency, and resource usage [25]. The Internet Engineering Steering Group (IESG) has already approved the final specification of HTTP/2 as of February, 2015 [46]. It is no exaggeration to state that "the future of the Web is HTTP/2" [4].

While HTTP/2 is expected to reduce page load time, we ask if using HTTP/2 improves energy consumption over using HTTP/1.1? In other words, is HTTP/2 going to be more mobile-user-friendly by offering longer battery life? Subsequently, should mobile application developers switch to this new HTTP/2 protocol for developing applications with HTTP requests? A recent study claimed the positive impact on energy consumption through efficient HTTP requests [31]. HTTP/2 is based on the promise of making efficient HTTP requests but the more complicated operations might require more CPU usage, such as dealing with encryption—a requirement in

HTTP/2. Will this extra computation harm its energy consumption?

In this paper, we study and compare the energy efficiency of HTTP/1.1 and HTTP/2 on mobile devices using a real hardware based energy measurement system: the Green Miner [28]. Our observations/contributions can be summarized as:

1) Using Transport Layer Security (TLS) incurs more energy consumption than HTTP/1.1 alone.
2) HTTP/2 performs similarly to HTTP/1.1 for very low *round trip time* (RTT).
3) For a significantly higher RTT, HTTP/2 is more energy efficient than HTTP/1.1.

In addition, we show the perils related to software energy measurements. We observed that energy measurement of software can be very tricky and making an incorrect conclusion is very likely in the absence of enough domain knowledge or controls. In such a case, an energy-aware software developer, in spite of having all the required energy measurement equipment, might not be measuring what they intend to measure.

## II. Background

In this section, we review the evolution of the HTTP protocol and the motivation for HTTP/2. We also define some of the terms that are frequently used in software energy consumption research.

### A. *Hyper Text Transfer Protocol (HTTP) and Its Limitations*

Hypertext Transfer Protocol (HTTP) was proposed in 1989 and documented as HTTP v0.9 in 1991 by Tim Berner Lee, laying out the foundation for modern World Wide Web [51]. In 1997, IETF published HTTP/1.1 [26] as the new improved official standard and more features and fixes were added afterwards: persistent connections, pipelining requests, improved caching mechanisms, chunked transfer encoding, byte serving etc. Users were not only able to request a hypertext resource from the servers but could also request images, Javascript, CSS and other types of resources.

According to HTTP Archive [29], as of April 2015, most Web applications are composed of HTML, images, scripts, CSS, flash and other elements, making the size of an average page more than 1.9 MB. It can take more than 90 requests over 35 TCP connections to 16 different hosts to fetch all of the resources of a Web application [46]. Although new features were proposed in HTTP/1.1 to handle such Web applications, some of these features suffered from their own limitations. For example, pipelining was never accepted widely among browsers because of the FIFO request-response mechanism, which can potentially lead to the head of line blocking problem resulting in performance degradation [46]. To keep up the performance of Web applications, Web developers have come up with their own techniques like domain sharding—splitting resources across different domains; spriting—e.g., combining a number of images into a single image; in-lining—avoiding sending each image separately; and concatenation of resources—aggregating lots of smaller files (Javascript for example) into a bigger one. These techniques, however, come with their own inherent problems [46].

### B. *SPDY and HTTP/2*

Google recognized the degrading performance of Web applications [22], and in mid-2009 they announced a new experimental protocol called SPDY [9]. While still retaining the semantics of HTTP/1.1, SPDY introduced a framing layer on top of TLS persistent TCP connections to achieve multiplexing and request prioritization. It allowed SPDY to achieve one of its major design goals to reduce page load time by up to 50% [24]. SPDY reduced the amount of data exchanged through header compression, and features such as server push also helped to reduce latency.

SPDY showed the need and possibility of a new protocol in place of HTTP/1.1 to improve Web performance. SPDY was the basis for the first draft of the HTTP/2 protocol [10]. HTTP/2 is a binary protocol that incorporates the benefits provided by SPDY and adds its own optimization techniques. It uses a new header compression format HPACK to limit its vulnerability to known attacks. HTTP/2 uses Application Layer Protocol Negotiation (ALPN) over a TLS connection as compared to Next Protocol Negotiation (NPN) used by SPDY. However, unlike SPDY, it does not make the use of TLS mandatory [46]. In early 2015, IESG allowed HTTP/2 to be published as the new proposed standard [36].

### C. *Power and Energy*

In this paper we focus on power use and energy consumption induced by a change in workload: switching from HTTP/1.1 to HTTP/2. Power is the rate of doing work or the rate of using energy; energy is defined as the capacity of doing work [3]. In our case, the amount of total energy used by a device within a period is the energy consumption, and energy consumption per second is the power usage. Power is measured in *watts* while energy is measured in *joules*. A task that uses 4 watts of power for 60 seconds, consumes 240 joules of energy. For tasks with the same length of time, mean-watt is often used to reduce noise in the measurement. This difference between power (rate) and energy (aggregate) is important to understand—improving one does not necessarily imply improving the other.

### D. *Tail Energy*

Some components including NIC (Network Interface Card), sdcard, and GPS on many smartphones suffer from tail energy—a component stays in a high power state for sometimes even after finishing its task [3], [41], [42]. This is inefficient as the application consumes energy without doing any useful work in this period. In 3G for example, approximately 60% of the total energy can be wasted only because of this tail energy phenomenon [6], which is a concern for mobile application developers.

## III. Methodology

### A. Green Miner

In order to run and capture the energy consumption profiles for HTTP/2 and HTTP/1.1, the Green Miner test bed [28] was used. Green Miner—a continuous testing framework similar to a continuous integration framework but with a focus on energy consumption testing—consists of five basic components: a power supply for the phones (YiHua YH-305D); 4 Raspberry Pi model B computers for test monitoring; 4 Arduino Unos and 4 Adafruit INA219 breakout boards for capturing energy consumption; and 4 Galaxy Nexus phones as the systems under test.

A constant voltage of 4.1V, generated by the YiHua YH-305D power supply, is passed to the Adafruit INA219 breakout board and subsequently goes to the Android phones. The INA219 reports voltage and amperage measurements to the Arduino that aggregates and communicates it to the Raspberry Pi. The Raspberry Pi sets up and monitors tests by initiating the test cases on a phone through ADB shell, and it controls the USB communication power (by using the Arduino Uno). Finally, the collected data (i.e., total energy consumption for a test case) is uploaded to a centralized server.

In order to disable cellular radios and bluetooth, the airplane mode was enabled in each phone and then Wi-Fi was re-enabled so that the phones can access the Internet. The phones were connected to a WPA secured wireless N network located in the same room, and thus ensuring very low variability of Internet access in order to have reliable measurements for our test scripts. The GreenMiner is fully described in the prior literature [28], [44].

### B. Writing a Test Script

In order to emulate a use case for the Android clients, a test script is required. For example, to emulate the use case where a user wants to load the Google home page to search for an item, we need a test script that can load a browser, write www.google.com in the address bar, and can press enter to load the webpage. This test can be automated by injecting various touch inputs into the input systems – these events can also be captured during actual use. A sequence of such actions (a test script) represents a specific use case for a user. The Green Miner executes the test script on the actual devices to execute the user actions (e.g., tap, swipe, enter etc.).

### C. Collecting Mozilla Firefox Nightly Versions

We have selected 10 versions of Mozilla Firefox Nightly (mobile US versions) to conduct our experiments [34]—using more than one Mozilla Firefox Nightly version improves generality and ensures that our results/observations are not contaminated by energy bugs that can be present in a specific version. Nightly versions—also known as Central in contrast to Aurora, Beta, and Release—are committed each day and are used to test the effectiveness of new features before including them in the actual releases [47]. We opted for the Nightly versions so that we could test a constantly changing codebase and avoid single version bugs while improving generality. The versions used in this paper, however, exhibit a stable energy consumption profile without any significant differences in terms of energy consumption.

Of the 10 Firefox versions, 9 versions were from January, 2015 to March, 2015 (three versions from each month with equal time intervals) and one version was from April, 2015. These versions had HTTP/2 support enabled by default while HTTP/1.1 can be enabled by disabling HTTP/2. The test scripts can enable or disable HTTP/2 within the Firefox browser: to test HTTP/1.1, HTTP/2 was disabled. We could not use Chromium in our tests as one cannot force newer Chromium versions to use HTTP/1.1 with TLS when HTTP/2 is enabled, regardless of disabling HTTP/2 in Chromium. Green Miner removes and installs Mozilla Firefox Nightly for each separate test, thus ensuring no caching advantages for any of the runs.

### D. Deploying a HTTP/2 Server

Among several implementations of HTTP/2 servers [1], we decided to deploy and experiment with the H2O [37] webserver, located at University of Alberta, Canada. H2O supports both HTTP/1.1 and HTTP/2 thus enabling a fair comparison between the two technologies. Besides, the performance of H2O was found significantly better than other implementations like Nginx [37]. The final version of HTTP/2 specification is also supported including NPN, ALPN, Upgrade and direct negotiation methods; dependency and weight-based prioritization; and server push. For the Gopher Tiles tests (described later) and the Twitter and Google tests, we relied on 3rd party webservers and webservices. This helped us to measure real world performance; when the page load time varies depending on different network scenarios.

### E. Workload

Our objective is to observe the performance of HTTP/2 compared to HTTP/1.1 with benchmarks that can represent real world scenarios. Recent observations for popular websites suggest that on average 2 MB of data needs to be downloaded in order to load a full page, and on average 100 objects must be downloaded [46]. Previous studies have found that the number of objects can play a key role in SPDY performance—the closest relative of HTTP/2 [49]. Although the evaluation criteria was different (page load time), this would be practical to do the similar for our analysis. Consequently, we experimented with the following benchmarks with varying number of objects and sizes. Table I shows the summary of our benchmarks.

*1) World Flags with fgallery:* We installed fgallery [14], a static photo gallery generator, on our own H2O server [37] that shows thumbnails of a set of images installed on the server. For our experiments, images of the world flags were used; a similar benchmark was used by Wang et al. [49]. The fgallery loads all the given images as thumbnails along with the full view of the first flag. The users have the option to view the subsequent flags one after another. Instead of using 50 world flags, we used all the country flags to make the workload

TABLE I
DESCRIPTION OF THE WORKLOADS

| | Number of Resources | | | | | Size(KB) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | HTML | Image | CSS | JS | Other | HTML | Image | CSS | JS | Other | Total |
| **World Flags** | 1 | 238 | 1 | 5 | 1 | 0.92 | 1261.87 | 4.61 | 117.73 | 27.47 | 1412.60 |
| **Gopher Tiles** | 1 | 180 | 0 | 0 | 1 | 17.14 | 165.80 | 0.00 | 0.00 | 0.76 | 183.70 |
| **Google** | 4 | 6 | 1 | 5 | 1 | 162.53 | 434.03 | 34.95 | 840.90 | 1.18 | 1473.62 |
| **Twitter** | 1 | 4 | 2 | 3 | 2 | 53.37 | 197.37 | 125.74 | 588.43 | 81.80 | 1046.73 |

heavier. The H2O server does not support HTTP/2 without TLS, leading us to experiment with three different settings: 1) HTTP/1.1 without TLS, [1] 2) HTTP/1.1 with TLS [2] and 3) HTTP/2 with TLS. [3]

*2) Gopher Tiles:* We also used another HTTP/2 server, developed by using the open-source *Go* programming language, which hosts a grid of 180 tiled images.[4] This demo server enables experiments with added artificial latencies. This is very important for our evaluation, as previous study observed significant performance variations with differing RTTs [40]. We captured the energy consumption of our Android devices for downloading the tiled images with different RTTs for both HTTP/1.1 and HTTP/2. The server, however, does not have TLS option for HTTP/1.1. On the contrary, its HTTP/2 implementation works only with TLS. As a result, we were able to evaluate the performance for only two settings: 1) HTTP/1.1 without TLS and 2) HTTP/2 with TLS.

*3) Google and Twitter:* In order to work with real websites, we have selected Google and Twitter for our evaluation because of their adoption of HTTP/2. This type of workload helps to investigate how HTTP/2 reacts for systems that are distributed; it is expected that for such highly accessed servers, Google and Twitter distribute different resources at different nodes, even if not totally at different domains. In contrast to the previous workloads, these two sites do not have access without TLS. This led us to experiment with two settings: 1) HTTP/1.1 with TLS and 2) HTTP/2 with TLS. For both the websites, the data collection period was from 2015-04-18 to 2015-04-19.

For Google, all the requests from our android devices were automatically redirected to *google.ca* and the resource statistics as reported in Table I are for HTTPS, as of writing Google does not support HTTP/1.1 without TLS.

Twitter requests, on the other hand, were redirected to *mobile.twitter.com*. Interestingly for Twitter, we observed that different resources (e.g., images) were downloaded for our mobile Mozilla Firefox Nightly versions than FireFox or Chrome in our Desktop computers.

[1]http://pizza.cs.ualberta.ca:1800/

[2]https://pizza.cs.ualberta.ca:1801/

[3]Same as HTTPS but with different browser setting

[4]Gophertiles https://http2.golang.org/gophertiles (last accessed: 2015-APR-22)

## *F. Validation*

*1) Problems with energy measurement:* Aggarwal et al. [3], using the Green Miner, observed that a single measurement for a particular setting could be misleading, as there is variation in the measurements because of several factors unrelated to the application of interest. Consequently, taking the average from at least 10 runs produces more accurate results. In this paper, we repeated each test 20 times for world flags and 15 times for others (after several tests we found that distributions of 15 were indistinguishable from 20 repeats).

Green Miner enables us to collect energy consumption measurements for different tasks (partitions) in our tests so that we can attribute energy consumption more accurately to a particular task. For example, in our world flags experiment, our script for capturing energy consumption for HTTP/1.1 with TLS has different tasks including App loading, disabling HTTP/2 (to enable HTTP/1.1), and page loading. We are, however, only interested in page load section so that we can compare it with the same section for HTTP/2. The challenge is that tasks, such as configuration, before the page load section for HTTP/1.1 with TLS is very different than HTTP/1.1 without TLS and HTTP/2 with TLS. Mozilla Firefox Nightly versions used in our experiments default to HTTP/2 support, hence forcing to HTTP/1.1 requires more configuration. As a result, for HTTP/2 with TLS experiments our tests do not have to change the browser's configuration: any encrypted request will automatically be a HTTP/2 request. Configuring the browser to use HTTP/1.1 with TLS requires many taps and clicks. These extra inputs can place the CPU into a different power state than if no configuration was done. [5] This is not required for HTTP/1.1 without TLS, as none of the servers used in our study support HTTP/2 without TLS. Consequently, any request without HTTPS will automatically be HTTP/1.1 (without TLS).

This different sequence of operations before the same page load section is a problem; modifying the about:config page might result in different power states for different components including CPU, screen, and NIC [7], [41], [42]. This could impact the subsequent operations' energy either positively (when components in high power states reduce the execution time significantly and nullify the effect of operating in high

[5]In Mozilla Firefox Nightly about:config, we need to disable *network.http.sdpy.enabled* and *network-.http.sdpy.enabled.http2draft*

power states) or negatively (the reduction in execution time is not significant enough). In either case, our measurements for HTTP/1.1 with TLS would be affected by the previous task's energy consumption leading to an unfair evaluation. In order to verify this hypothesis—to measure how inaccurate the measurement is—we captured the page load energy consumption for the same protocol (HTTP/1.1 without TLS) twice:[6] once without changing Mozilla Firefox Nightly config file (as we do not need to disable HTTP/2 for unencrypted HTTP/1.1) and another time by changing the config file (disable HTTP/2). This two settings should give us the same average measurement if the later one is not affected by either the different power states of the components or the tail energy.

Test runs were averaged and compared against each other using 2-sided paired t-tests paired by Mozilla Firefox Nightly version. Besides, we observed the effect size by calculating *Cohen's d*.[7] Unfortunately, the small P-value ($<< 0.05$) and the large *Cohen's d* (7.02) suggest that these two settings produce significantly different measurements, thus implying the CPU state was not consistent.

*2) A potential solution:* The challenges we faced in measuring energy consumption led us to configure our test scripts differently: what if we apply a sleep period before accessing the main task, the page load? And how long is required to have accurate measurements? Our hypothesis is that this inactive/idle time would help our Android devices to come to the stable state, i.e., same CPU state.

We experimented with three different periods: 40 seconds, one minute and 2 minutes. The p-value for the 40 seconds period was still lower than $0.05$, and slightly higher than $0.05$ with 60 seconds of idle time. This P-value, however, becomes very high (0.86) for two minutes of waiting time with very low *Cohen's d* (0.08)—suggesting the very little difference between these two settings comes from randomness and led us to conduct our experiments for world flags with two minutes waiting time before loading the page.

The time-line graphs in Figure 1 show the average power usage over 20 runs for each Nightly version for both 1 minute and 2 minutes of waiting time. On the contrary, a box-plot in the Figure represents the distribution of power usages for a specific setting by all the versions across all the 20 runs. Encouragingly, the median value (from the box-plot) is very similar to the average values (from the time-line), implying the accuracy of Green Miner in measuring energy consumption.

Results suggest that with one minutes of waiting time the difference is obvious (one setting always consumed less energy than the other), whereas there is no such trend for two minutes of waiting time. Moreover, the variations over different runs for two minutes stable time are significantly lower than for one minute of stable time (box plot in Figure 1)—implying better accuracy can be achieved with longer waiting time before executing the actual operations. Interestingly for both settings, the power usage with 1 minute stable time is also higher than 2 minutes stable time. This is not surprising as the CPU is expected to operate in a low power state after having a significant amount of idle time.

We, however, imposed only 1 minute of waiting time for experiments other than world flags, as the previous operations before page load are exactly the same for all the settings. This energy measurement validation approach is crucial, revealing the need for controlling the states of components like the CPU.

## IV. Experiment and Result Analysis

### A. World Flags

Figure 2 shows the performance of three different settings for world flags with fgallery: HTTP/1.1 (without TLS), HTTP/1.1 with TLS and HTTP/2 with TLS. Interestingly, HTTP/1.1 and HTTP/2 exhibit very similar performance for our world flags workload when encryption is applied. The high P-value ($>> 0.05$) in Table II and low *Cohen's d* ($< 0.3$) between these two settings confirm that the observed small difference come from randomness in data collection (i.e., the difference is not significant).

This observation is not surprising as previous studies [40] also found that HTTP and SPDY perform very similarly when the RTT is low. And for this experiment with world flags the RTT between our clients and the server was very low (close to 0 ms).

The performance of HTTP/1.1 without encryption is, however, very interesting as it clearly outperforms HTTP/1.1 with TLS although a previous study [21] found improved response time with encrypted messages compared to plain HTTP. Our observations, however, complement the assumptions made by Naylor *et al.* [35]: 1) The required handshaking mechanism for HTTPS consumes energy, which is not present in unencrypted communication; 2) As the browser also takes the responsibility for encryption/decryption, this may lead to more CPU usage and subsequently more energy consumption; 3) The browser verifies if the server, with HTTPS support, is authenticated by examining the server's certificate, which needs more work to be completed.

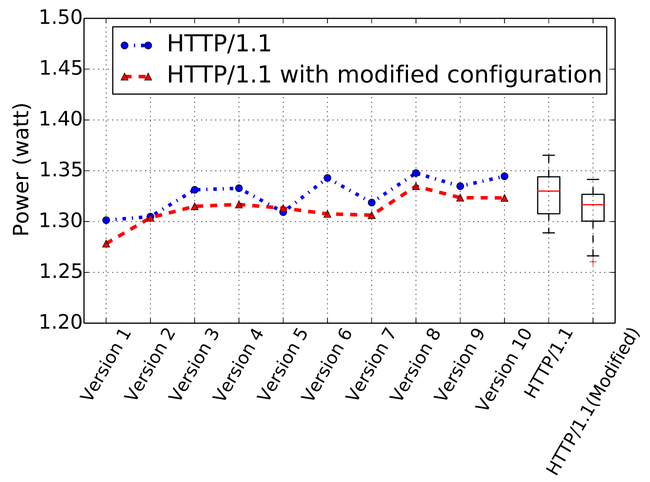
### B. Gopher Tiles

In order to compare the performance of HTTP/2 with HTTP/1.1 for different network scenarios, experiments with Gopher tiles were conducted with different latencies: 0 ms, 30 ms, 200 ms and 1000 ms. The result is shown in Figure 3.

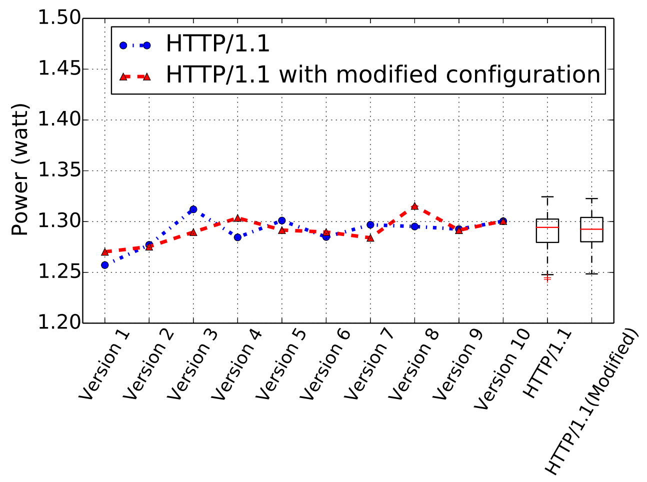
The better performance of HTTP/1.1 than HTTP/2 for low RTT corroborates our findings for world flags: with no latency, HTTP/2 does not offer any improvement over HTTP/1.1, and secured encrypted transmission becomes an overhead which leads to more energy consumption. HTTP/1.1 loses its advantage over HTTP/2 once latency is 30 ms latency or larger. We suspect that HTTP/2 would have outperformed HTTP/1.1 if implemented without TLS. This overhead from

[6]one single value is actually the average value of 20 measurements

[7]In order to represent energy consumption by a Mozilla Firefox Nightly version to calculate *Cohen's d*, we took the average of 20 runs.



(a) Waiting time one minute



(b) Waiting time two minutes

Fig. 1. Comparing Power usages for the same protocol with different settings

TABLE II
P-VALUE FOR PAIRED T-TEST AMONG DIFFERENT SETTINGS FOR WORLDFLAGS WITH FGALLERY

| | HTTP/1.1 | HTTP/1.1 with TLS | HTTP/2 with TLS |
|---|---|---|---|
| **HTTP/1.1** | 1 | 5e-11 | 3e-09 |
| **HTTP/1.1 with TLS** | 5e-11 | 1 | 0.244 |
| **HTTP/2 with TLS** | 3e-09 | 0.244 | 1 |

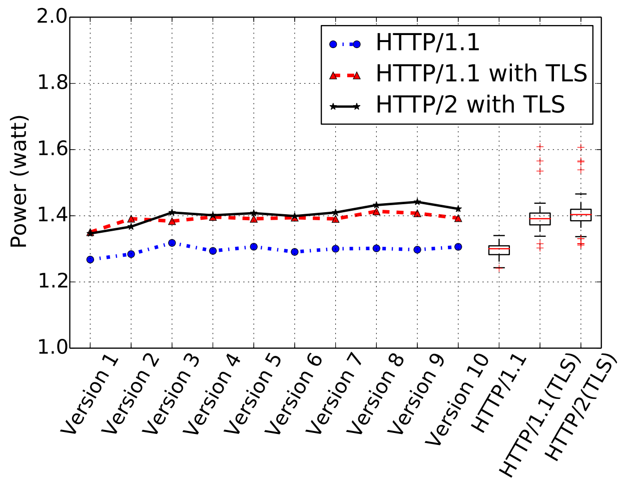


Fig. 2. Power usage of different settings for world flags with fgallery

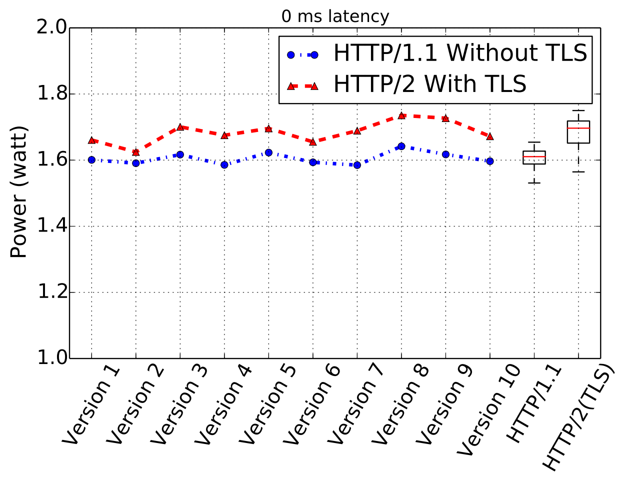
TLS, however, becomes negligible for RTT 200 ms and 1000 ms; HTTP/2 significantly outperforms HTTP/1.1 with high RTT. For all the cases, the P-values were low ($<< 0.05$) and *Cohen's d* values—the effect size—were very high.

One interesting and useful observation is that although the total energy consumption for both protocols with 1000 ms latency is much higher than the energy consumption with 200 ms (the download time is much longer), the power usage for 200 ms latency is higher than for 1000 ms latency. [8] This could be because of the higher power states of components like CPU and NIC for 200 ms than for 1000 ms, as faster downloading/processing might push the hardware components to more aggressive energy consuming states. This complements previous findings that completion time is not necessarily proportional to a device's energy consumption—different hardware components can have different states of operation; a CPU, for example, can operate at different frequencies, and thus can have different energy profiles in different scenarios [43].

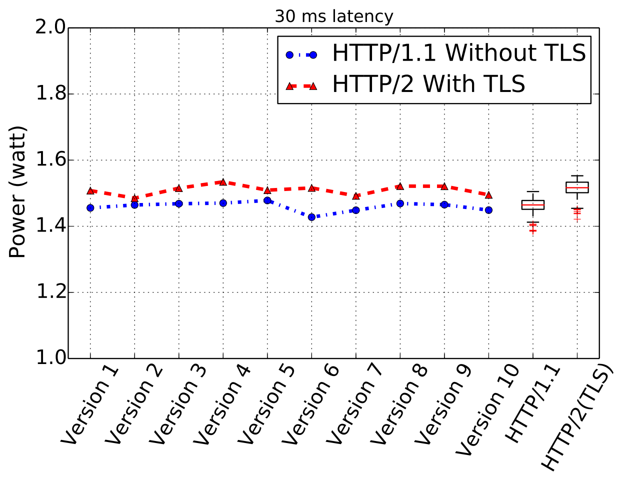
## *C. Google and Twitter*

We also experimented with Google Search and Twitter home pages—two of the most accessed sites by the Internet users [5]. Instead of experimenting with different types of user interactions in these sites, we only considered the page load times similar to a previous study [49]. Figure 4 shows the performance of both protocols with TLS; as mentioned earlier, these two sites automatically redirect requests to HTTPS. Although the P-value (0.028) and *Cohen's d* (0.88, large) suggest that the difference between the two settings for

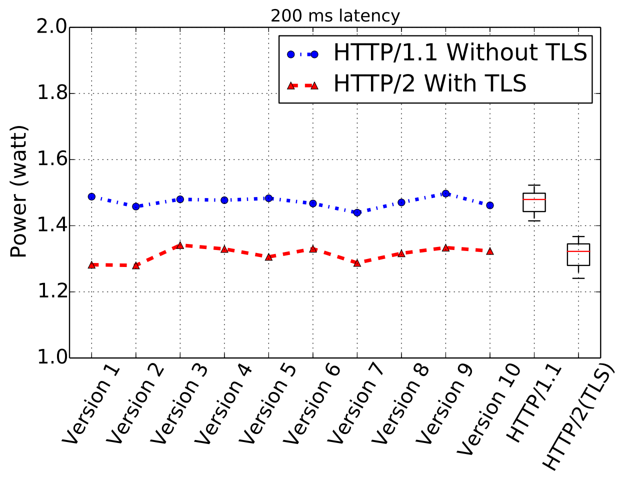
[8] We calculate power by dividing the total energy consumption by the duration.



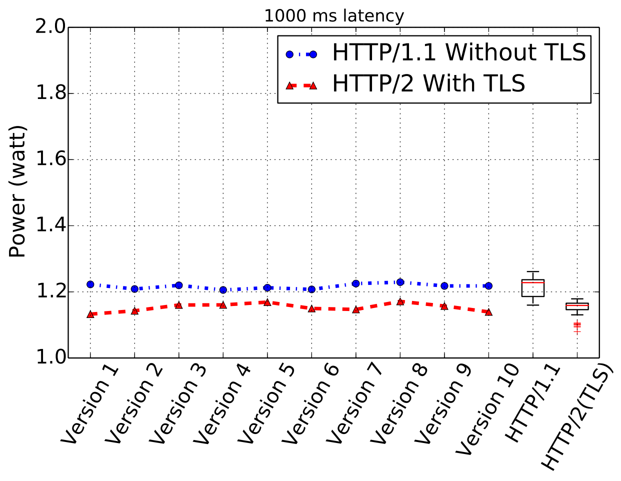
(a) RTT 0 ms



(b) RTT 30 ms



(c) RTT 200 ms



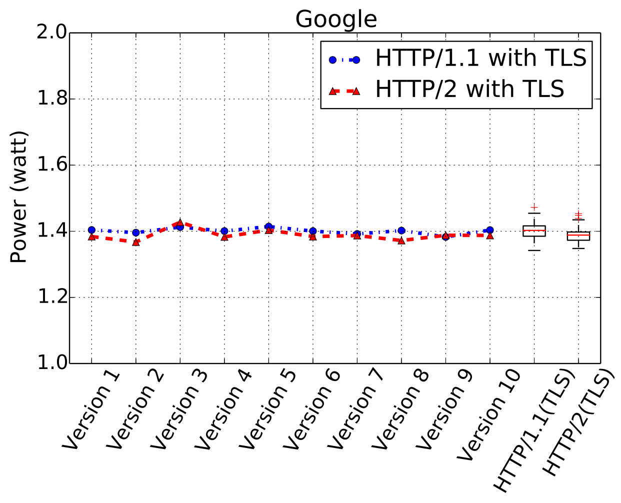
(d) RTT 1000 ms

Fig. 3. Power usage for Gopher tiles with different RTTs

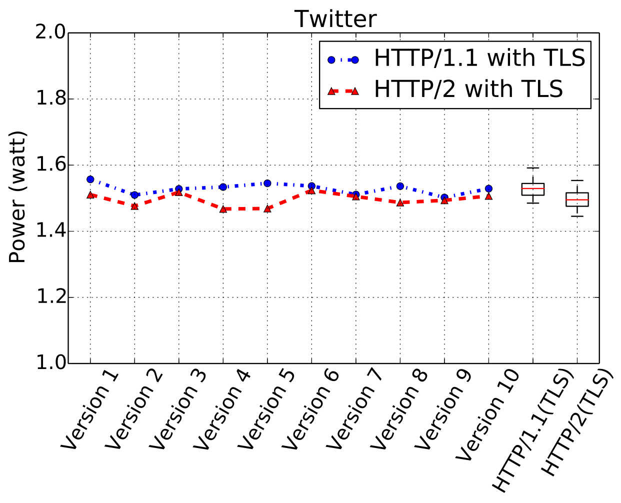
Google is statistically significant, the improvement in power usage reduction for HTTP/2 is very little for Google. In case of Twitter, however, the difference is not only statistically significant (P-value $<<$ 0.05 and large *Cohen's d* of 2.1), but the improvement in energy efficiency for HTTP/2 is large. We attribute this behaviour to the differing RTTs for these two sites from our client; the RTT for Google was around 20 ms and was around 80 ms for Twitter. This observation between Google and Twitter is very significant as it reveals how HTTP/2 will affect the mobile users in their everyday lives—if not better, HTTP/2 does not perform worse, at least for these two very top sites.

## V. Discussion

When the RTT is 30ms or more, the difference in energy consumption between HTTP/1.1 and HTTP/2 is obvious. One might argue that the difference, although significant statistically, is not convincing enough to become a deciding factor. This statement is true when only considered for a single mobile device and for a browsing period of one second (as we compared in *watt)*. But when the effect of this difference is considered globally for billions of mobile devices and for average users' browsing time, the energy saving would be colossal. Moreover, in some of our experiments (e.g., Gopher Tiles), in contrast to HTTP/2, HTTP/1.1 was experimented



(a) Power usage for Google



(b) Power usage for Twitter

Fig. 4. Power usage for Google and Twitter

without TLS, otherwise the difference could have been much more conspicuous.

It is encouraging that although HTTP/2 design goals are mainly oriented towards faster page load time, in most cases it also offers better energy performance than its predecessor.

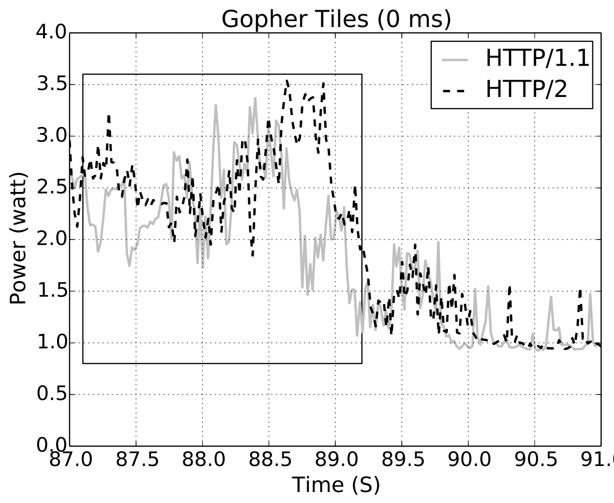
*Why is HTTP/2 more energy efficient for larger RTTs?*

In HTTP/1.1, GET requests are processed in the exact order they are received. Moreover, multiple TCP connections may be established to achieve concurrency. In a high latency network, these factors result in longer waiting periods and also use additional computational resources for establishing TCP connections. The longer waiting times between subsequent network operations produce more tail energy leaks. HTTP/2 solved these issues through the multiplexing of GET requests over a single TCP connection per domain. Also, HTTP/2 eliminated the overhead of transferring redundant header fields and compresses the header metadata through the HPACK algorithm [13]. These factors reduce the network operation times between the client and the server supporting HTTP/2 which is vital in reducing energy consumption. The *prioritization* mechanism of HTTP/2 also helps in faster page loading, which can be a significant factor in reducing energy consumption.

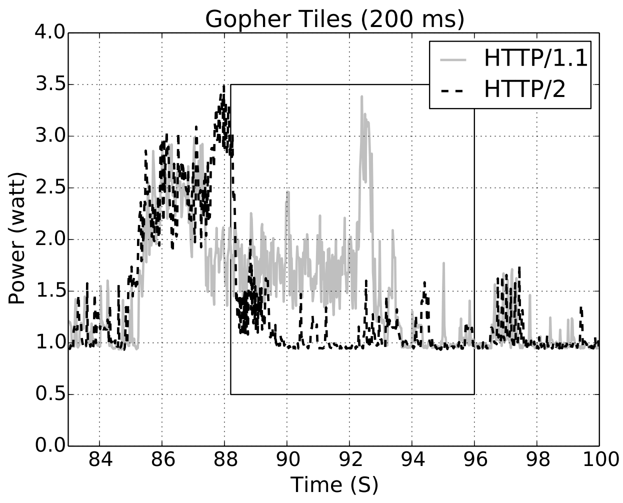
In case of a very low RTT (or no RTT), these factors (e.g., head of line blocking, and header overheads) do not play a significant role both in page loading and power usage, as we observed previously (Figure 3). In order to have more insight, we also captured power usage over-time for different settings. Figure 5 illustrates the power usage over time for selected configurations during the page load partitions. Figure 5 (a), using a single version of Firefox, shows the more power usage by HTTP/2 than HTTP/1.1 at the beginning (highlighted by the rectangle). We believe this is because of the encrypted transmission (HTTPS) employed by HTTP/2 but not by HTTP/1.1. This overhead, however, becomes negligible with the increase in RTT. Although the power usage of HTTP/2 for high latency networks are higher for the very first few seconds, Figure 5 (b) and (c), it becomes almost flat afterwards (with a few small spikes). It is easy to notice that (rectangles in b and c) page load time for HTTP/2 is much shorter in high latency networks. On the contrary, HTTP/1.1's network operations stay active for a much longer time and consume much more energy.

It is important to mention that GreenMiner does not know when a page load is completed so that it can stop the test immediately. This led us to experiment with fixed test durations for our different settings. For example, for the 1000 ms latency test with Gopher, our predefined test duration for the page load partition was 40 seconds. This is the tightest limit considering the slow page load time of HTTP/1.1. If the actual page load time could have been configured through GreenMiner, the difference would be even more significant. This conclusion can also be made from the Twitter experiment. Figure 5 (d) shows a very significant spike for HTTP/2 at the end (second rectangle), although the page load was completed much earlier (rectangle 1). During the time highlighted by the second rectangle, HTTP/2 was switching to a different home page image after loading the first one long before, in contrast to HTTP/1.1 which, at that time, was not completely done with loading the first image. This is a clear illustration of the energy efficiency of HTTP/2 over HTTP/1.1.

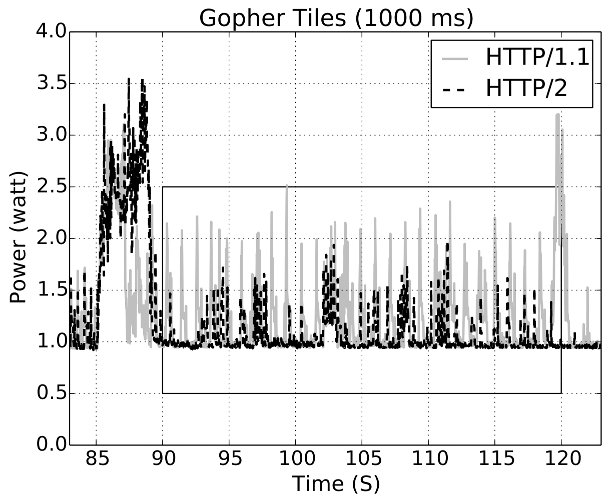
Web app developers should really consider adopting HTTP/2 as their transport protocol. If RTT is 30ms or more there should be noticeable gains in load time, usability, and energy consumption for their clients. Also, HTTP/2 mech-



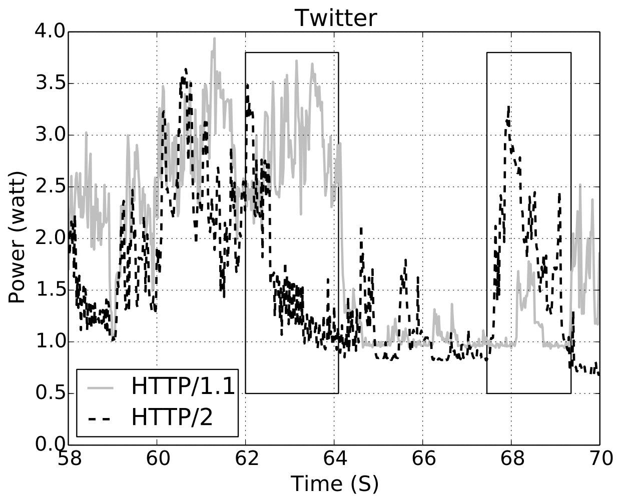
(a) Gopher with 0 ms latency



(b) Gopher with 200 ms latency



(c) Gopher with 1000 ms latency



(d) Twitter

Fig. 5. Power usage Over time

anisms are great examples of making a trade-off between computation and network operations toward producing energy smart systems.

## VI. Threat to Validity

We did not experiment with HTTP/2 server push and left it as future work; it would be interesting to see how this feature affects the overall performance. The workload we experimented with also affects our observation; although we backed our results, in most cases, with previous studies. The Mozilla Firefox Nightly versions—although unlikely—can have their own inherent energy bugs and can contaminate the results. The realism of our test-cases can be argued for and against as a balance was to be made between synthetic tests (gopher tiles), realistic tests (world flags) and real-world subjects (Twitter and Google). More websites and more browsers and servers could be tested. Generalization was harmed by using only Mozilla's HTTP/2 clients. Future work should investigate the gain for app developers making HTTP requests from smartphone apps using libraries like Apache HTTPLib.

## VII. Related Work

We divide this section into two subsections: studies related to optimization in mobile device energy consumption and performance evaluation of Web protocols.

## A. *Mitigation of Energy Bugs/hotspots in Applications*

Banerjee *et al.* [7] observed that the main sources of energy consumption in smartphones are the I/O components. As I/O components are accessed by applications through system calls, capturing those system calls can help to find energy bugs or hotspots in a particular application [42], [3]. Pathak *et al.* [41] observed that I/O operations consume more energy partly because of the tail energy phenomenon. According to the authors, this tail energy leak can be mitigated by bundling I/O operations together. Li *et al.* [32] proposed a Color Transformation Scheme for Web applications to find the most energy efficient color scheme while maintaining the enticement and readability at the same time.

Othman *et al.* [39] claimed that up to 20% energy savings is achievable by uploading tasks from mobile devices to fixed servers. A similar study by Miettinen *et al.* [33] suggests that most of the mobile applications were found to be suitable for local processing. This could be the result of limited available resources with such devices, resulting in deficient number of computationally expensive mobile applications. Trestian *et al.* [48] examined the impact of different network related aspects on mobile device's energy consumption in case of video streaming. The authors addressed the impact of several factors on mobile energy efficiency: video quality, selection of TCP or UDP as the transport layer protocol, link quality, and network. A similar study by Gautam *et al.* [18] suggests that applying algorithmic prefetching can help in saving substantial energy of mobile devices. Rasmussen *et al.* [44] found that a system with Ad-blockers, in most cases, is more energy efficient than systems without Ad-blockers.

## B. *Performance of Web Protocols*

**SPDY Studies**: A study [23] on the page load times of the top 100 websites suggests that SPDY can improve page load time by 27-60% over HTTP without SSL and 39-55% with SSL. In a different study by Google [24], SPDY over mobile networks on an Android device was found to improve the page load time by 23%. Contradicting those observations, Erman *et al.* [16] found that unlike wired connections, SPDY doesn't outperform HTTP over cellular networks. Latency in cellular networks can continuously vary due to radio resource connection state machines, and TCP doesn't account for such variability which results in unnecessary re-transmissions. This affects SPDY more due to its use of a single connection.

Wang *et al.* [49] found that multiplexing and longer RTTs help SPDY to achieve an improvement over HTTP. However, the improvement is significantly reduced due to Web page dependencies, browser computations or under high packet loss. Padhye *et al.* [40] compared SPDY and HTTP on a dummy Webpage simulating different network conditions.

**HTTP/2 Studies**: In HttpWatch [30], the performance of HTTP/2, SPDY and raw HTTPS (HTTP with TLS) protocols were compared using different parameters. Compared to SPDY, request and response header size were found to be smaller for HTTP/2—indicating compression achieved in HTTP/2 using HPACK is more efficient than the DEFLATE algorithm used by SPDY. However, SPDY's response message sizes were smaller as they compressed textual resources.

The compression used by HTTP/2 also allowed headers [45] and images to be smaller. In terms of number of connections, due to multiplexing over a single connection, HTTP/2 and SPDY performed better than raw HTTPS. For page load time, HTTP/2 was consistently found to be better than SPDY (HTTPS performed worst). Centminmod [17] community's administrator benchmarked HTTP/1.1, SPDY 3.1 and HTTP/2 performance on different servers—Nginx, H2O and OpenLiteSpeed depending on the protocols supported by them. For all three servers HTTP/2 and SPDY 3.1 performed better than HTTP/1.1. Between HTTP/2 and SPDY, the performance of HTTP/2 on H2O server was best followed by SPDY/3.1 on Nginx and HTTP/2 on OpenLiteSpeed. Similar results were observed under 3G mobile network. Other studies [4], [19] compared HTTP/2 performance with HTTP/1.1 under different latency conditions and showed the performance benefits of HTTP/2 over HTTP/1.1.

Cherif *et al.*[12] used HTTP/2's server push feature in a *Dynamic Adaptive Streaming* (DASH) session to reduce the initial load time of a video. Loading time under HTTP and HTTPS increased with the increase in RTT, and at an RTT of 300 ms loading time with HTTP/2 outperformed HTTP and HTTPS by 50%. This gain was attributed to the fast increase in TCP receiver window size due to server push in HTTP/2.

## VIII. Conclusions and Future Work

*Does HTTP/2 save energy?* Yes, when round trip times are above 30ms and when TLS is being used, our tests indicate that HTTP/2 outperforms HTTP/1.1 with TLS in most scenarios. The Mozilla Firefox Nightly implementation of HTTP/2 consumes less energy than the HTTP/1.1 implementation to do the same work regardless of the webserver used in the tests.

The advantage of HTTP/2 highly depends on the round trip time between the client and the server. HTTP/1.1 becomes expensive, for large number of TCP connections, with large number of objects. This becomes even worse when the RTT is higher, which is common in cellular data networks. HTTP/2, on the other hand, does not have this problem as it deals with one single connection by incorporating a multiplexing technique. We also observed that accessing the Internet has become more energy expensive for mobile clients after adopting HTTPS as the standard to ensure user's privacy and security.

We conclude that the web served over HTTP/2 is going to be more mobile user friendly especially on high-latency wireless and Wi-Fi networks, and that energy-aware application developers should adopt HTTP/2.

In this paper, we did not consider the energy consumption of HTTP/2 servers; we only evaluated the energy usage of HTTP/2 from the context of mobile clients. This would be an interesting avenue of research to investigate how energy efficient HTTP/2 is from the server side perspective.

## Acknowledgment

Shaiful Chowdhury is grateful to the Alberta Innovates - Technology Futures (AITF) to support his PhD research. Abram Hindle is supported by a NSERC Discovery Grant.

## References

[1] HTTP/2 Implementations. https://github.com/http2/http2-spec/wiki/Implementations. (last accessed: 2015-APR-22).

[2] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014-2019. Technical report, February 2015.

[3] K. Aggarwal, C. Zhang, J. C. Campbell, A. Hindle, and E. Stroulia. The Power of System Call Traces: Predicting the Software Energy Consumption Impact of Changes. In *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*, CASCON '14, pages 219–233, November 2014.

[4] Akamai. HTTP/2 is the future of the Web, and it is already here! http://http2.akamai.com/demo/. (last accessed: 2015-APR-22).

[5] Alexa. The top 500 sites on the web. http://www.alexa.com/topsites. (last accessed: 2015-APR-22).

[6] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, pages 280–293, Chicago, Illinois, USA, November 2009.

[7] Banerjee, Abhijeet and Chong, Lee Kee and Chattopadhyay, Sudipta and Roychoudhury, Abhik. Detecting Energy Bugs and Hotspots in Mobile Apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 588–598, Hong Kong, China, Novemeber 2014.

[8] L. A. Barroso, J. Dean, and U. Hölzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, 23(2):22–28, March 2003.

[9] M. Belshe. A 2x Faster Web. http://googleresearch.blogspot.ca/2009/11/2x-faster-web.html. (last accessed: 2015-APR-22).

[10] M. Belshe and R. Peon. SPDY Protocol. http://tools.ietf.org/html/draft-ietf-httpbis-http2-00. (last accessed: 2015-APR-22).

[11] H. Brotherton. *Data center energy efficiency*. PhD thesis, Purdue University, May 2014.

[12] W. Cherif, Y. Fablet, E. Nassor, J. Taquet, and Y. Fujimori. DASH Fast Start Using HTTP/2. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '15, pages 25–30, Portland, Oregon, USA, March 2015.

[13] H. de Saxce, I. Oprescu, and Y. Chen. Is http/2 really faster than http/1.1? In *Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on*, pages 293–299, April 2015.

[14] Y. DElia. fgallery: a modern, minimalist javascript photo gallery. http://www.thregr.org/~wavexx/software/fgallery/. (last accessed: 2015-APR-22).

[15] eMarketer. Smartphone users worldwide will total 1.75 billion in 2014. http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536. (last accessed: 2015-APR-22).

[16] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan. Towards a SPDY'Ier Mobile Web? In *Proceedings of the 9th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 303–314, Santa Barbara, California, USA, December 2013.

[17] eva2000. HTTP/2 - h2o vs OpenLiteSpeed vs Nginx SPDY/3.1. http://community.centminmod.com/threads/http-2-h2o-vs-openlitespeed-vs-nginx-spdy-3-1.2564/. (last accessed: 2015-APR-22).

[18] N. Gautam, H. Petander, and J. Noel. A Comparison of the Cost and Energy Efficiency of Prefetching and Streaming of Mobile Video. In *Proceedings of the 5th Workshop on Mobile Video*, MoVid '13, pages 7–12, Oslo, Norway, February 2013.

[19] Go Language. Go + HTTP/2. http://http2.golang.org/gophertiles/. (last accessed: 2015-APR-22).

[20] Í. Goiri, M. E. Haque, K. Le, R. Beauchea, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. Matching Renewable Energy Supply and Demand in Green Datacenters. *Ad Hoc Networks*, 25(0):520 – 534, February 2015.

[21] A. Goldberg, R. Buff, and A. Schmitt. A comparison of http and https performance. http://www.cs.nyu.edu/artg/research/comparison/comparison.html. (last accessed: 2015-APR-22).

[22] Google. Make the Web Faster. https://developers.google.com/speed/?csw=1. (last accessed: 2015-APR-22).

[23] Google. SPDY: An experimental protocol for a faster web. http://www.chromium.org/spdy/spdy-whitepaper. (last accessed: 2015-APR-22).

[24] Google. SPDY Performance on Mobile Networks. http://developers.google.com/speed/articles/spdy-for-mobile. (last accessed: 2015-APR-22).

[25] I. H. W. Group. Http/2. https://http2.github.io/. (last accessed: 2015-APR-22).

[26] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. RFC 2608, RFC Editor, June 1999. http://www.rfc-editor.org/rfc/rfc2608.txt.

[27] A. Hindle. Green Mining: Investigating Power Consumption Across Versions. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 1301–1304, Zurich, Switzerland, June 2012.

[28] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky. GreenMiner: A Hardware Based Mining Software Repositories Software Energy Consumption Framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 12–21, Hyderabad, India, May 2014.

[29] HTTP Archive. HTTP Trends. http://httparchive.org/trends.php?s=All&minlabel=Oct+22+2010&maxlabel=Apr+15+2015. (last accessed: 2015-APR-22).

[30] HttpWatch. A Simple Performance Comparison of HTTPS, SPDY and HTTP/2. http://blog.httpwatch.com/2015/01/16/a-simple-performance-comparison-of-https-spdy-and-http2/. (last accessed: 2015-APR-22).

[31] D. Li and W. G. J. Halfond. Optimizing energy of http requests in android applications. In *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, DeMobile 2015, pages 25–28, New York, NY, USA, 2015. ACM.

[32] D. Li, A. H. Tran, and W. G. J. Halfond. Making Web Applications More Energy Efficient for OLED Smartphones. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 527–538, Hyderabad, India, June 2014.

[33] A. P. Miettinen and J. K. Nurminen. Energy Efficiency of Mobile Clients in Cloud Computing. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 4–4, Boston, MA, USA, June 2010.

[34] Mozilla. Index of /pub/mozilla.org/mobile/nightly. http://ftp.mozilla.org/pub/mozilla.org/mobile/nightly/. (last accessed: 2015-APR-22).

[35] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. 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.

[36] M. Nottingham. HTTP/2 Approved. http://www.ietf.org/blog/2015/02/http2-approved/. (last accessed: 2015-APR-22).

[37] K. Oku, T. Kubo, D. Duarte, N. Desaulniers, M. Hrsken, M. Nagano, J. Marrison, and D. Maki. H2o - an optimized http server with support for http/1.x and http/2. https://github.com/h2o/h2o. (last accessed 2015-APR-22).

[38] A.-C. Orgerie, M. D. d. Assuncao, and L. Lefevre. A Survey on Techniques for Improving the Energy Efficiency of Large-scale Distributed Systems. *ACM Comput. Surv.*, 46(4):47:1–47:31, March 2014.

[39] M. Othman and S. Hailes. Power Conservation Strategy for Mobile Computers Using Load Sharing. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2(1):44–51, January 1998.

[40] J. Padhye and H. F. Nielsen. A comparison of SPDY and HTTP performance. Technical Report MSR-TR-2012-102, July 2012.

[41] A. Pathak, Y. C. Hu, and M. Zhang. Where is the Energy Spent Inside My App?: Fine Grained Energy Accounting on Smartphones with Eprof. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 29–42, Bern, Switzerland, April 2012.

[42] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-grained Power Modeling for Smartphones Using System Call Tracing. In *Proceedings of the 6th Conference on Computer Systems*, EuroSys '11, pages 153–168, Salzburg, Austria, April 2011.

[43] G. Pinto, F. Castor, and Y. D. Liu. Mining Questions About Software Energy Consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 22–31, Hyderabad, India, June 2014.

[44] K. Rasmussen, A. Wilson, and A. Hindle. Green Mining: Energy Consumption of Advertisement Blocking Methods. In *Proceedings of*

*the 3rd International Workshop on Green and Sustainable Software*, GREENS 2014, pages 38–45, Hyderabad, India, June 2014.

[45] H. Si and A. Hada. Introduction of HTTP/2 and Performance Comparison of HTTP/1 and HTTP/2. http://www.ixiacom.com/about-us/news-events/corporate-blog/introduction-http2-and-performance-comparison-http1-and-http. (last accessed: 2015-APR-22).

[46] D. Stenberg. HTTP/2 Explained. *SIGCOMM Comput. Commun. Rev.*, 44(3):120–128, July 2014.

[47] TheOldFox. What is FireFox Nightly ? https://support.mozilla.org/en-US/questions/970739. (last accessed: 2015-APR-22).

[48] R. Trestian, A.-N. Moldovan, O. Ormond, and G. Muntean. Energy consumption analysis of video streaming to Android mobile devices. In *Proceedings of the 2012 IEEE Network Operations and Management Symposium*, NOMS'12, pages 444–452, Maui, HI, USA, April 2012.

[49] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. How Speedy is SPDY? In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 387–399, Seattle, WA, USA, April 2014.

[50] V. Woollaston. customers really want better battery life. http://www.dailymail.co.uk/sciencetech/article-2715860/Mobile-phone-customers-really-want-better-battery-life-waterproof-screens-poll-reveals.html (last accessed: 2015-APR-22).

[51] World Wide Web Consortium (W3C). The Original HTTP as defined in 1991. http://www.w3.org/Protocols/HTTP/AsImplemented.html. (last accessed: 2015-APR-22).