Is HTTP/2 More Energy Efficient Than HTTP/1.1 for Mobile Users?

Shaiful Alam Chowdhury[1], **Varun Sapra**[1], **and Abram Hindle**[1]

[1]**Department of Computing Science, University of Alberta, Canada**

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 battery life. Mobile users are very interested in accessing the Internet while 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, 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 is HTTP/2 compared to HTTP/1.1. We conclude that for networks with higher RTTs, HTTP/2 has better energy consumption performance than HTTP/1.1.

Keywords: HTTP/2, HTTP/1.1, Energy Performance, Software Engineering

1 INTRODUCTION

In recent years, the popularity of mobile devices (e.g., smartphones, and tablets) has dramatically increased, as millions of users are using these devices in their daily lives. These devices offer users the opportunity to have more computational power and even more capability in the palm of their hand than most users had on their desktop just a few years ago. It is reported that, as of 2014, more than 1.4 billion smartphones are being used globally [8], and that worldwide mobile data traffic grew approximately 70% [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 Camera, Wifi, 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 [56] indicates that a longer battery life is one of the most desired features among smartphone users. Unfortunately, the advancement in battery technology is inconspicuous compared to the improvement in computing abilities—implicitly registering the importance of energy efficient application development.

Energy consumption, in addition to the mobile devices, has also become a subject of concern for large data centers—consuming at least one percent of the world's energy [12]. The is because of the continually increasing demand for storage, networking and computation capabilities. Reportedly, 4.3 TeraWatt-year energy was consumed in US in 2010 only from the deployment of LAN switches and routers [42]. Energy efficiency was reported as one of the pivotal issues, even by Google, for the scale of operation—putting aside the associated energy cost for such a big data centers—as cooling becomes very important operational factor [9]. Another very important aspect of energy consumption is the environment: energy consumption has detrimental effect on climate change, as most of the electricity is produced by burning fossil fuels [22]. Reportedly, 1000 tonnes of CO_2 is produced every year by computer energy consumption of mid-sized organizations [31].

With the increased penetration of the mobile devices, the Internet usage on these smartphones is also on the rise. According to eMarketer [16], 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 [45].

Loading Web pages has become more resource intensive than any time before and poses challenges to the inefficient HTTP/1.1 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 each page requires around 100 objects to be transferred [51]. HTTP/2—mainly based on SPDY, a protocol proposed and developed by Google—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 [27]. The Internet Engineering Steering Group (IESG) has already approved the final specification of HTTP/2 as of February, 2015 [51]; the future of the Web is HTTP/2 [4].

While HTTP/2 is expected to reduce page load time, we ask what would be the energy consumption by this replacement of HTTP/1.1? In other words, is HTTP/2 going to be more mobile user friendly by offering longer battery life and thus providing more browsing capabilities?

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 [32]. 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 similar to HTTP/1.1 for very low RTT.
3. For a significantly higher RTT, HTTP/2 is more energy efficient than HTTP/1.1.

To the best of our knowledge, we are the first to study the energy efficiency of mobile devices for different Web protocol implementations.

2 BACKGROUND

In this section, we review the evolution of HTTP protocol and the motivation for HTTP/2. We also define some of the terminologies that are frequently used in this field, and subsequently will be used in this paper.

2.1 Hyper Text Transfer Protocol (HTTP)

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 [57]. HTTP v0.9 was a simple protocol having only one method, GET, meant to fetch only hypertext from the server. However, early 1990's was a period of rapid development for Internet and as the Web evolved the need for more capabilities to be added to HTTP became obvious. In 1997, IETF published HTTP/1.1 in RFC 2608 [29] as the new improved official standard and more features and fixes were added to it in RFC 2616 [19] in 1999. HTTP/1.1 added many new features—persistent connections, pipelining requests, improved caching mechanisms, chunked transfer encoding, byte serving—to support modern Internet at that time. Users can now not only request a hypertext resource from the servers but can also request images, Javascript, CSS and other type of resources. Since then, HTTP/1.1 has been the communication protocol for almost all the browsers and the devices for the Web [51, 29, 19].

2.2 Limitations of HTTP/1.1

It has now been more than 15 years since the standardization of HTTP/1.1, and a lot has changed in the world of Internet in this time frame. According to HTTP Archive [33], as of April 2015, most Web applications are composed of HTML, images, scripts, CSS, flash and other elements making a page more than 1.9 MB in size on average. 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 [51]. 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 first in first out request response mechanism can potentially lead to head of line blocking problem resulting in performance degradation [51].

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1280v1 | CC-BY 4.0 Open Access | rec: 4 Aug 2015, publ: 4 Aug 2015

To keep up the performance of Web applications, Web developers have come up with their own techniques like domain sharding, spriting, and inlining and concatenation of resources. These techniques, however, come with their own inherent problems [51].

2.3 SPDY and HTTP/2

Google recognized the degrading performance of Web applications [24], and in mid-2009 they announced a new experimental protocol called SPDY [10]. 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 goal to reduce page load time by upto 50% [26]. SPDY reduced the amount of data exchanged through header compression, and features such as server push also helped to reduce latency. Despite its features and it being standardized, Google SPDY faced criticism [51]. It required TLS in its implementation. Organizations were hesitant to adopt it as it benefited Google Chrome Browser and was not an IETF standard. Moreover, its compression algorithm was found to be vulnerable to CRIME (Compression Ratio Info-leak Made Easy) and BREACH (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext) attacks [51]. Consequently, as of April 2015, only 3.9% of the Web servers used SPDY [54].

SPDY, however, showed the need and possibility of a new protocol in place of HTTP/1.1 to improve Web performance. The IETF took notice and the work on a new protocol HTTP/2 started in 2012. SPDY was the basis for the first draft of the HTTP/2 protocol [11]. 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 doesn't make the use of TLS mandatory [51]. In early 2015, IESG allowed HTTP/2 to be published as the new proposed standard [40]. HTTP/2 still faces some criticism [35] for relying heavily on SPDY for its development and being prepared on an "unrealistically short schedule" thereby "missing a lot of ripe opportunities" [35].

2.4 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, means-watts are often used to reduce noise in the measurement. This difference between power and energy is important to understand—improving one does not necessarily imply improving another, as discussed later in this paper.

2.5 Energy Efficiency

Energy efficiency, as defined by the World Energy Council, is the reduction of energy consumed by a given service or activity [37]. In case of mobile devices, however, energy efficiency is more complicated, and can be achieved from different level of optimizations, such as function approximation [7], locating and resolving energy bugs and hotpots [45], and hardware optimization. The last case (e.g., significantly improved battery technology) does not seem to be achieved in near future. For example, Banerjee et al. [8] found that mobile computing power grew exponentially while battery capacity did not.

2.6 Tail Energy

Some components including NICs, sdcard, and GPS on many smartphones suffer from tail energy—a component stays in high power state for sometimes even after finishing its task [3, 45, 46]. This is unfortunate 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 alarming for the mobile application developers.

2.7 Energy Bugs/Hotspots

An energy bug—in contrast to the traditional programming bugs producing incorrect/unwanted result—leads to reduced battery life [58]. The sources of energy bugs can be of different types: *no sleep bug*—result of waking up the CPU, but not returning back to sleep mode; *loop bug*—waiting for an

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1280v1 | CC-BY 4.0 Open Access | rec: 4 Aug 2015, publ: 4 Aug 2015

event to happen, periodically inspecting changes in a variable, and thus unnecessarily using CPU cycles. An energy hotspot, however, points to a segment of an application where the energy consumption is significantly higher than other segments, which is not necessarily the result of an energy bug [8]. For example, an energy hotspot can be the result of an unwanted tail energy phenomenon.

3 RELATED WORK

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

3.1 Mobile Device's Energy Consumption

3.1.1 Modeling energy consumption

Measuring a system's energy consumption was reported as one of the most difficult tasks in the context of energy efficient software development [47]. Consequently, modeling and predicting energy consumption in stead of the actual measurement is expected to alleviate this problem.

Utilization-based power models take into consideration the utilization statistics of individual components of a system like CPU, screen brightness, Wi-Fi etc. Regression analysis is conducted, to model application's energy consumption, by using the utilization statistics and the corresponding energy consumption [13, 28, 49]. Utilization based models, however, suffer from several issues [46]: tail power phenomenon—some components (e.g., NICs, sdcard, GPS) stay in high power state for a specific period after switching to inactive state, which can not be captured in utilization based models; components without quantitative utilization—some components do not have any kind of quantitative utilization, yet a system call can switch the component's state.

The drawbacks of the widely explored utilization based approaches motivated new energy modeling approaches, such as energy model based on system calls invocations. Pathak *et al.* [46] modeled energy consumption of different Android and Windows mobile phones using system call traces, and the logs of context switch events were captured to estimate CPU power usage. Unlike utilization based approaches, this model can mitigate the tail power energy problem. The proposed model is based on Finite State Machine (FSM), where each state in FSM represents a power state of a specific component or a set of components. Aggarwal *et al.* [3] followed up with a much simpler approach to understand energy consumption of mobile applications—counts of different groups of system calls invoked by applications were used to develop energy regression model.

Instruction based modeling was also studied. In order to estimate software energy consumption using program instruction cost, Shuai *et al.* [30] proposed *eLens* —a tool that can estimate energy profiles at instruction level, method level, and thus can estimate the energy consumption of the whole software system.

3.1.2 Energy bugs/hotspots in applications

In order to detect hotspots and bugs in Android applications, a test generation framework was offered by Banerjee *et al.* [8]. The authors experimented on 30 different applications available on Google Play/F-Droid. The primary assumption of the authors is that the main sources of energy consumption in smartphones are the I/O components. As I/O components are accessed by applications by invoking system calls, capturing those system calls can help to find energy bugs or hotspots in a particular application.

Pathak *et al.* [45], using *eprof*, have observed that I/O operations are mostly responsible for smpartphone energy drains, mostly because of the tail energy phenomenon. This is unfortunate, as tail energy is the energy-wastage by a component in its inactive state (i.e., no application is currently using that component).

3.1.3 Improving energy efficiency

Different studies have been conducted to understand different avenues to improve energy efficiency. For example, Li *et al.* [36] proposed a Color Transformation Scheme (CTS) for Web applications. The objective of the proposed technique is to find the most energy efficient color scheme while maintaining the enticement and readability at the same time. This requires retrieving color information and modeling association between colors with structural dependencies.

Study by Othman *et al.* [43] claimed that up to 20% energy savings is achievable by uploading task from local mobile device to a fixed server. All jobs, however, are not suitable for migration, as offloading

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1280v1 | CC-BY 4.0 Open Access | rec: 4 Aug 2015, publ: 4 Aug 2015

might lead to even worse energy efficiency. Moreover, asymmetric communication—more receives than sends—is needed to achieve such energy efficiency; transmitting data is much more expensive than receiving especially when the primary focus is energy efficiency. The main challenge is to know the energy profiles of a task for both local and remote executions. A similar study by Miettinen *et al.* [38] 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 deficient number of computationally expensive mobile applications.

Trestian *et al.* [53] examined the impact of different network related aspects on mobile device's energy consumption in case of video streaming. In general, the authors addressed the impact of several factors on mobile energy efficiency in video streaming: video quality, selection of TCP or UDP as the transport layer protocol, link quality, and network. A similar study by Gautam *et al.* [20] suggests that applying algorithmic prefetching can helps toward saving substantial energy of mobile devices. This is, however, based on the assumption that the algorithm is accurate in selecting appropriate videos based on user predilection—an inaccurate algorithm might backfire by downloading lots of videos that the user might never watch.

In a separate study, Rasmussen *et al.* [48] found that a system with Ad-blocker, in most cases, is more energy efficient than systems without Ad-blockers.

3.2 Performance of Web Protocols

Previous studies have compared the performance of SPDY protocol with HTTP/1.1. As performance of a page load time can depend on different factors including bandwidth, latency, and number of objects, the observation might be different depending upon the test conditions. Overall, the majority of these works point towards the better performance of SPDY over HTTP/1.1.

SPDY Studies: SPDY whitepaper [25] provides a comparison of SPDY against HTTP in terms of page load time of about top 100 websites. According to this study, SPDY showed 27-60% improvement in page load time over HTTP over TCP without SSL and 39-55% with SSL. In a different study by Google [26], SPDY over mobile networks on an Android device was found to improve the page load time by 23% for Webpages across a set of 31 popular domains.

However, contradicting these results, Erman *et al.* [17] compared the performance of SPDY and HTTP over real world cellular networks on 20 Web pages for four months. Their study suggests 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 thereby resulting in unnecessary re-transmissions. This affects SPDY more due to its use of a single connection. However, in case of WiFi connection SPDY was found to outperform HTTP by 4-56%.

Wang *et al.* [55] presented an in-depth comparison of SPDY and HTTP by measuring page load time and identifying conditions under which SPDY performs better. They conducted their evaluation on artificial Web pages as well as on top 200 Alexa sites. It was 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. Under mobile environment, benefits provided by SPDY in slow networks were found to be undermined by large computation delays. Padhye *et al.* [44] compared the performance of SPDY and HTTP on a dummy Webpage simulating different network conditions. Although SPDY outperformed HTTP, improvement in HTTP performance was observed after implementing Web optimization techniques.

HTTP/2 Studies: In HttpWatch [34], 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 DEFLATE algorithm used by SPDY. However, for response message size, SPDY had smaller size for textual resources.

It was attributed to padding bytes added to HTTP/2 data frame. For image resources, HTTP/2 response was 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). In a different study by Ixia [50], HTTP/2 header compression was found to be 75% more efficient than no header compression in HTTP/1.1. Centminmod [18] community's administrator benchmarked HTTP/1.1, SPDY 3.1 and HTTP/2 performance on different servers—Nginx, H2O and OpenLiteSpeed depending upon the protocols supported by them. For all three servers HTTP/2

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1280v1 | CC-BY 4.0 Open Access | rec: 4 Aug 2015, publ: 4 Aug 2015

and SPDY 3.1 performed better than HTTP/1.1. Among HTTP/2 and SPDY, 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, 21] 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.*[14] uses HTTP/2's server push feature in a *Dynamic Adaptive Streaming* (DASH) session to reduce initial load time of a video. They compared the performance benefits achieved using HTTP/2 against HTTP/1.1. 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 fast increase in TCP receiver window size due to server push in HTTP/2. None of these HTTP studies, however, considered differences in energy consumption.

4 METHODOLOGY

4.1 Green Miner

In order to run and capture the energy consumption profiles of implementations of these protocols, the Green Miner test bed [32] 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 major components: YiHua YH-305D power supply (phone power supply), Raspberry Pi model B computer (test monitoring), Arduino Uno (energy measurement), Adafruit INA219 breakout board (energy measurement), and four Galaxy Nexus phones (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 Arduino Uno collects the measurements of voltage and amperage used by a phone, which is then reported to a Raspberry Pi through a serial USB connection. The Raspberry Pi setups 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 radio, 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. Interested readers are encouraged to study review relevant Green Miner literature [32, 48].

Green Miner measurements are more accurate and use less instrumentation than techniques like PowerTutor [59]; PowerTutor estimates energy consumption with a liner model rather than physically measuring actual energy consumption.

4.2 Writing a Test Script

In order to emulate an 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. Touch inputs such as touch start, touch end, tap, touch swipes can be captured using the Android emulator and then encoded into a test-script meant to emulate the same actions. A sequence of such actions (a test script) represents a specific use case for an user. The Green Miner executes the test script on the actual devices to execute the user actions (e.g., tap, swipe, enter etc.).

4.3 Collecting Mozilla Firefox Nightly Versions

We have selected 10 versions of the open source Mozilla Firefox Nightly (mobile US versions) to conduct our experiments [39]—using more than one Mozilla Firefox Nightly version can 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 [52]. We opted for the Nightly versions so that we could test a constantly changing codebase and avoid single version bugs. The versions used in this paper, however, have stable energy consumption profiles and do not show 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

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1280v1 | CC-BY 4.0 Open Access | rec: 4 Aug 2015, publ: 4 Aug 2015

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.

4.4 Deploying HTTP/2 Server

Among several implementations of HTTP/2 servers [1], we decided to deploy and experiment with the H2O [41] webserver. 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 [41]. 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 and the Twitter and Google tests we rely on 3rd party webservers and webservices. This helps us better measure real world performance when the page load time varies depending on different network scenarios.

4.5 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 [51]. Previous studies has found that the number of objects can play a key role in SPDY performance—the closest relative of HTTP/2 [55]. Although the evaluation criteria was different (page load time), this would be practical to do the same for our analysis. Consequently, we experimented with the following benchmarks with varying number of objects and sizes. Table 1 shows the summary of our benchmarks.

Table 1. 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	1412.6
Gopher Tiles	1	180	0	0	1	17.14	165.80	0	0	0.76	183.7
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

4.5.1 World Flags with fgallery

We installed fgallery [15], a static photo gallery generator, on our own H2O server that shows thumbnails of a set of images installed on the server. For our experiments, images of the world flags were used, as similar benchmark was used by other similar studies [55]. 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 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]

4.5.2 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 [44]. 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

[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)

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1280v1 | CC-BY 4.0 Open Access | rec: 4 Aug 2015, publ: 4 Aug 2015

TLS. As a result, we were able to evalute the performance only for two settings: 1) HTTP/1.1 without TLS and 2) HTTP/2 with TLS.

4.5.3 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. In contrast to the previous workloads, these two sites do not have access without TLS. This also 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 1 are for HTTPS, as there is no HTTP/1.1 without TLS for Google.

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.

4.6 Validation

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 15 repeats offer very similar accuracy to 20 repeats).

Misleading observations, however, might come from other factors. Before comparing the energy consumptions between different protocols, it is necessary to know how accurate is our measurement were for those factors. Differences between different measurements for the same settings (even after taking average from more than 10 repeats) ares not desirable, and for such a case we might wrongly conclude that a particular protocol is better than the other. Green Miner enables us to collect energy consumption measurements for different tasks (partitions) in our scripts so that we can measure a task's energy consumption more accurately. 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 for 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. This is because of the default HTTP/2 support by the Mozilla Firefox Nightly versions used in our experiments, which must be disabled for other experiments. As a result, for HTTP/2 with TLS experiments our scripts does 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 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 results different power states for different components including CPU, screen, and NIC, and might exhibits the tail power phenomenon [8, 45, 46]. This suggest that 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 is the measurement—we captured the page load energy consumption for the same protocol (HTTP/1.1 without TLS) twice:[6] one without changing Mozilla Firefox Nightly config file (as we do not need to disable HTTP/2 for unencrypted HTTP/1.1) and another by changing the config file (disable HTTP/2). This two settings should give us the same average measurement if the later one is neither affected by different power states of the components, nor by the tail power phenomenon.

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]

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

[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.

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1280v1 | CC-BY 4.0 Open Access | rec: 4 Aug 2015, publ: 4 Aug 2015

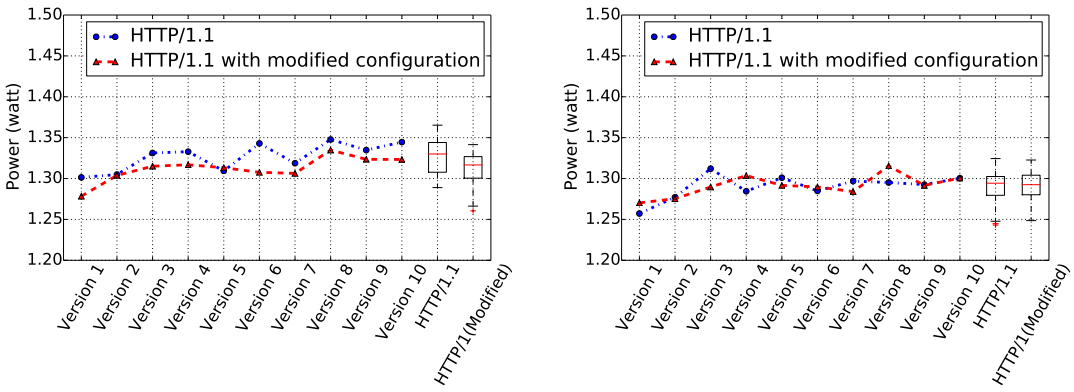
Unfortunately, the P-value ($<< 0.05$) and the *Cohen's d* (7.02) suggest that these two settings produce significantly different measurements, thus implying the CPU state was not consistent. This led us to configure our test scripts differently: what if we apply a sleep period before accessing the main task—page load? And what length of that time 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 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 come from randomness and leading 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 one minute and two minutes of waiting time. On the contrary, a box-plot in the Figure represents the distribution of power usage for a specific setting by all the versions across all the 20 runs. Encouragingly, the median value (from 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 other), whereas there is no such trend for two minutes waiting time. Moreover, the variations over different runs for two minutes stable time is significantly lower than one minute 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 one minute stable time is also higher than two minutes stable time. This is not surprising as the CPU is expected to operate in low power state after having a significant amount of idle time.

We, however due to time constraints and acceptable accuracy, imposed only one minute waiting time for other experiments than world flags, as the previous operations before page load are exactly the same for all the settings (described later).



(a) Waiting time one minute

(b) Waiting time two minutes

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

5 EXPERIMENT AND RESULT ANALYSIS

5.1 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 exhibits very similar performance for our world flags workload when encryption is applied. The high P-value ($>> 0.05$) in Table 2 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 [44] also found that HTTP and SPDY performs very similarly when the RTT is low. And for this experiment with world flags the RTT between our clients and server was very low (close to 0 ms).

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1280v1 | CC-BY 4.0 Open Access | rec: 4 Aug 2015, publ: 4 Aug 2015

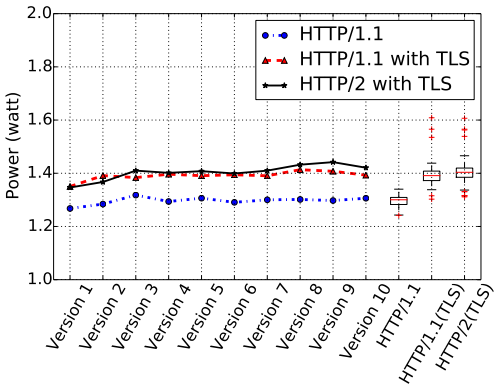


Figure 2. Power usage of different settings for world flags with fgallery

Table 2. 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

The performance of HTTP/1.1 without encryption is, however, very interesting as it clearly outperforms HTTP/1.1 with TLS although previous study [23] found improved response time with encrypted messages compared to plain HTTP. We hypothesize several reasons that can lead to energy inefficiencies when encryption is applied: 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.

5.2 Gopher Tiles

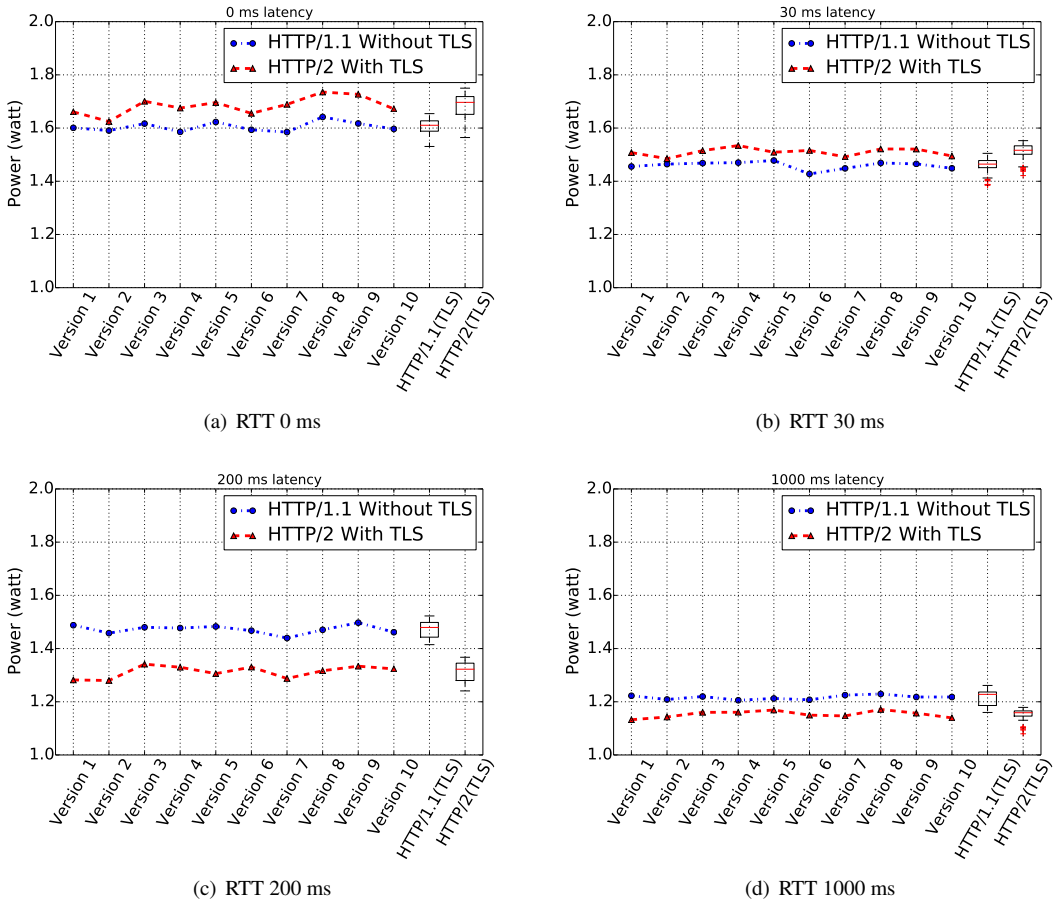
In order to compare the performance of HTTP/2 with HTTP/1.1 for different network scenarios, experiments with Gopher tiles were conducted with different latencies: 0 ms, 30 ms, 200 ms and 1000 ms. The result is shown in Figure 3.

The better performance of HTTP/1.1 versus HTTP/2 complements our findings for world flags: with no latency, HTTP/2 does not offer any improvement over HTTP/1.1, and secured encrypted transmission becomes overhead which leads to more energy consumption. The HTTP/1.1 advantage between these two protocols become less with 30 ms latency. We believe HTTP/2 would have outperformed HTTP/1.1 if implemented without TLS. This overhead from 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 very low ($<< 0.05$) and *Cohen's d* values 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 1000 ms latency. [8] This could be because of the higher power states of component 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 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 [47].

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

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1280v1 | CC-BY 4.0 Open Access | rec: 4 Aug 2015, publ: 4 Aug 2015



(a) RTT 0 ms

(b) RTT 30 ms

(c) RTT 200 ms

(d) RTT 1000 ms

Figure 3. Power usage for Gopher tiles with different RTTs

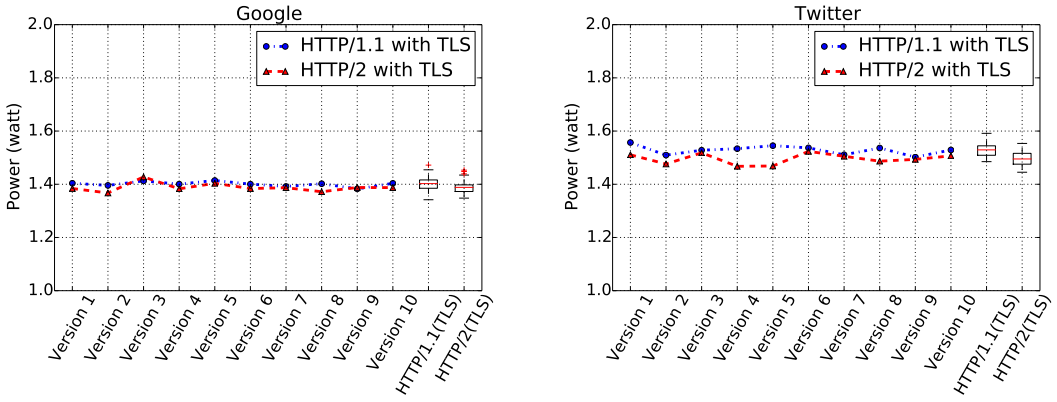
5.3 Google and Twitter

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 previous study [55]. 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 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 in not only statistically significant (P-value $<<$ 0.05 and *Cohen's d* 2.1, large), significant improvement in energy efficiency is observed for HTTP/2. Interestingly, we found that this is because of the RTT for these two sites from our clients; 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 effect the mobile users in their everyday lives—if not better, HTTP/2 never performs worse, at least for these two very top sites.

6 THREAT TO VALIDITY

We could not experiment with HTTP/2 server push and left it as one of our future work; it would be interesting to see how this feature effects the overall performance. The workload we experimented with also effect our observation; although we backed our results, in most cases, with previous studies. The Mozilla Firefox Nightly versions can have their own inherent problems—i.e., presence of energy bugs and instability—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.

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1280v1 | CC-BY 4.0 Open Access | rec: 4 Aug 2015, publ: 4 Aug 2015



(a) Power usage for Google

(b) Power usage for Twitter

Figure 4. Power usage for Google and Twitter

7 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. 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 through encryption.

In our tests HTTP/2 never performs worse than its predecessor especially when TLS is mandatory, and as the RTT goes up, it always exhibits better performance. We conclude that the web served over HTTP/2 is going to be more mobile user friendly and induce less energy consumption especially on high-latency wireless and wi-fi networks.

In this paper, however, 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 a interesting avenue of research to investigate how energy efficient HTTP/2 is from the server side perspective—server energy consumption is considered as one of the most important factors for the scaling of operations within large data centers.

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*

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1280v1 | CC-BY 4.0 Open Access | rec: 4 Aug 2015, publ: 4 Aug 2015

9th ACM SIGCOMM Conference on Internet Measurement Conference, IMC '09, pages 280–293, Chicago, Illinois, USA, November 2009.

[7] A. Banerjee and A. Roychoudhury. Energy-aware Design Patterns for Mobile Application Development (Invited Talk). In *Proceedings of the 2nd International Workshop on Software Development Lifecycle for Mobile*, DeMobile 2014, pages 15–16, Hong Kong, China, November 2014.

[8] 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.

[9] 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.

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

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

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

[13] A. Carroll and G. Heiser. An Analysis of Power Consumption in a Smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 21–21, Boston, MA, USA, June 2010.

[14] 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.

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

[16] 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).

[17] 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.

[18] 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).

[19] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616, RFC Editor, June 1999. `http://www.rfc-editor.org/rfc/rfc2616.txt`.

[20] 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.

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

[22] 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.

[23] 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).

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

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

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

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

[28] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. K.

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1280v1 | CC-BY 4.0 Open Access | rec: 4 Aug 2015, publ: 4 Aug 2015

John. Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, HPCA '02, pages 141–150, Cambridge, Mass., USA, February 2002.

[29] 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.

[30] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan. Estimating Mobile Application Energy Consumption Using Program Analysis. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 92–101, San Francisco, CA, USA, May 2013.

[31] 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.

[32] 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.

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

[34] 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).

[35] P.-H. Kamp. HTTP/2.0: The IETF is Phoning It in. *Commun. ACM*, 58(3):40–42, March 2015.

[36] 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.

[37] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos. Cloud Computing: Survey on Energy Efficiency. *ACM Comput. Surv.*, 47(2):33:1–33:36, December 2014.

[38] 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.

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

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

[41] K. Oku, T. Kubo, D. Duarte, N. Desaulniers, M. Hörsken, 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).

[42] 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.

[43] 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.

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

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] A. Shye, B. Scholbrock, and G. Memik. Into the Wild: Studying Real User Activity Patterns to

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1280v1 | CC-BY 4.0 Open Access | rec: 4 Aug 2015, publ: 4 Aug 2015

Guide Power Optimizations for Mobile Architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 168–178, New York, NY, USA, December 2009.

[50] 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).

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

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

[53] 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.

[54] W3Techs.com. Usage of SPDY for websites. http://w3techs.com/technologies/details/ce-spdy/all/all. (last accessed: 2015-APR-22).

[55] 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.

[56] 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).

[57] 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).

[58] J. Zhang, A. Musa, and W. Le. A Comparison of Energy Bugs for Smartphone Platforms. In *1st International Workshop on the Engineering of Mobile-Enabled Systems*, MOBS'13, pages 25–30, San Francisco, CA, USA, May 2013.

[59] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES/ISSS '10, pages 105–114, Scottsdale, Arizona, USA, October 2010.

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1280v1 | CC-BY 4.0 Open Access | rec: 4 Aug 2015, publ: 4 Aug 2015