Energy efficiency of HTTP2 over HTTP1.1 on the mobile web

CSE-E5440: Energy-efficient Mobile Computing
Group C

Raatikka, Marko
`marko.raatikka@aalto.fi`

Lucas, Triefenbach
`lucas.triefenbach@aalto.fi`

April 30, 2016

1 Part I

1.1 Introduction

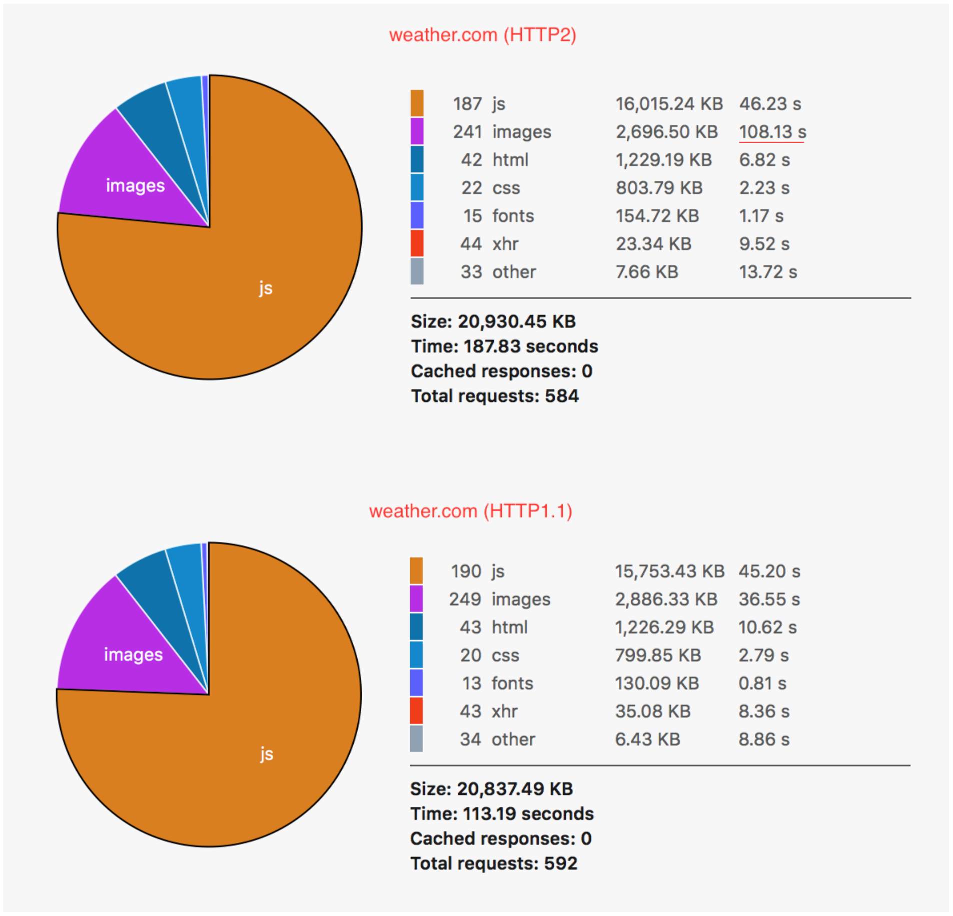
With the increasing computation power and connection speed of mobile phones in recent years, the amount of data flowing through the Internet to mobile end devices continues to grow. While web applications have become ever more complex and the amount of HTML, JavaScript and CSS required to render a web page increases, HTTP1.1 – the prevailing transfer protocol of today's web – has failed to keep up with the pace. One of the culprits of HTTP1.1 is that it can only handle one request per a TCP connection and that it can only have a certain maximum number of connections per host (5-8 connections depending on the browser). To work around these shortcomings developers have used techniques such as domain sharding (hosting assets across multiple domains), image spriting and inlining CSS & JavaScript in the HTML. However, a transport layer problem can not be efficiently solved at the application level.

The work for a new version of HTTP protocol was set forth by Google in 2012 in the form an open networking protocol called SPDY. Later, in 2015, building on the work done for SPDY, HTTP2 was proposed as the future standard protocol. The biggest improvements HTTP2 introduces over HTTP1.1 are request/response multiplexing (to avoid a so called *head-of-line blocking* inherent in HTTP1), header compression, prioritization of requests and server push mechanism. Web services are slowly migrating into adopting HTTP2, but as of today, only about 7% of the websites have fully migrated to the new protocol. [1][3][5]

This work sets to study the impact of HTTP2 on mobile power consumption. Previous work on the topic has been done by [4]. However, this study focuses more on generating high traffic and mimicking common, real-life navigation patterns. Due to time constraints testing is limited to a single browser as opposed to testing across various versions. The content is organized as follows: chapter 1.2 describes the method used, the testbed environment and the test suite. Chapter 1.3 continues by presenting the experiment results, which are later reviewed in Chapter 1.4. Last, an outline for the part II of the study is given.

1.2 Design and Implementation

The energy efficiency of the HTTP1.1 and HTTP2 protocols were measured by generating large amounts of network traffic. Six different web services were chosen as the targets, including Flickr, Yahoo, Facebook, Twitter, Weather.com and Instagram. These web services were selected based on their popularity and support for HTTP2 clients (browsers). Facebook, Twitter and Weather.com were eventually omitted from the final experiment corpus: the network traffic generated by Facebook was highly optimized, and therefore, it was assumed that it would not introduce any interesting statistical differences. Twitter on the other hand was deemed too lightweight (consisting of mostly text-based content), while Weather.com suffered from peculiar latency issues when transmitting images over HTTP2 as shown by Figure 1.



weather.com (HTTP2)

🟧	187	js	16,015.24 KB	46.23 s
🟪	241	images	2,696.50 KB	108.13 s
🟦	42	html	1,229.19 KB	6.82 s
🟦	22	css	803.79 KB	2.23 s
🟦	15	fonts	154.72 KB	1.17 s
🟥	44	xhr	23.34 KB	9.52 s
⬜	33	other	7.66 KB	13.72 s

Size: 20,930.45 KB
Time: 187.83 seconds
Cached responses: 0
Total requests: 584

weather.com (HTTP1.1)

🟧	190	js	15,753.43 KB	45.20 s
🟪	249	images	2,886.33 KB	36.55 s
🟦	43	html	1,226.29 KB	10.62 s
🟦	20	css	799.85 KB	2.79 s
🟦	13	fonts	130.09 KB	0.81 s
🟥	43	xhr	35.08 KB	8.36 s
⬜	34	other	6.43 KB	8.86 s

Size: 20,837.49 KB
Time: 113.19 seconds
Cached responses: 0
Total requests: 592

Figure 1: Weather.com was excluded from the final experiment corpus because it exhibited abnormally long response times when image assets were fetched over HTTP2. This was done the prevent introducing bias to the energy consumption measurements.

The experiment was conducted using Firefox (version 46.0) as, at the time of this writing, no other publicly available browser supports manually disabling HTTP2. The implementation of a robust and reproducible experiment environment required automation of the navigation logic. Browser automation frameworks Appium[1] and Mozilla's Robocop[2] were originally considered for this task. However, Appium lacked support for the Firefox browser, and Robocop

[1]Appium - `appium.io`
[2]Robocop - `https://wiki.mozilla.org/Auto-tools/Projects/Robocop`

– Mozilla's internal tool for testing Firefox on mobile – missed proper documentation. Thus, an alternative approach – including the use of Android's low level input API – was taken. The utilization of the input API was inspired by an open-source energy consumption measurement framework called GreenMiner [2], which uses this technique extensively.

The Android input API allows for the emulation of device gestures, and consequently, automation of browser navigation. The trade-off of this approach is that it requires root access to the device. Therefore the Samsung S4 used for the experiment was rooted using ClockworkMod recovery (version 6.0.4.7) and CyanogenMod (version 12), which ships with Android 5.1.1. The upside of installing a custom ROM – a customized OS image installed on the device's ROM memory – like CyanogenMod is that any pre-installed Google services or bundled Samsung software are not included. This helped in minimizing any unwanted power consumption generated by these type of services running in the background unknowingly.

Monsoon Solutions' Power Monitor[3] was used for collecting the energy consumption data. The negative and positive terminals of the smartphone battery were connected to the power monitor using copper wire. The two reading terminals of the Samsung S4 battery were isolated with tape to allow the smartphone to be started using external power (the Power Monitor). The experiment scripts used for navigating to the different websites were implemented as Bash shell scripts. The scripts were transferred onto the device via USB using ADB (Android Debug Bridge). ADB also allowed controlling the execution of the test scripts from the laptop, so no human interaction was required while running the tests.

Prior to running the test suite different smartphone features that might have caused noise (e.g. the energy consumption of background processes) were disabled. This included disabling automatic brightness, notifications, sounds, vibrations, widgets, lights and the phone's power saving features. Features enabled for experiment purposes were WiFi, Airplane mode and Developer options. The measurements were conducted in the Aalto open WiFi network during the late evening (22:00) to ensure low latency and minimal network congestion. All caching functionality of the Firefox browser was disabled to maximize network traffic (assets would always be fetched over the network). Crash reporting and monitoring capabilities of the browser were disabled so that they would not cause spikes in power consumption during the experiment. The browser HTTP2 support could be toggled on and off via the Firefox configuration interface. The cache and HTTP configuration flags used are listed in Table 1. For further information the interested reader is referred to the MozillaZine Knowledge Base[4].

[3]`https://www.msoon.com/LabEquipment/PowerMonitor/`

[4]MozillaZine Knowledge Base - `http://kb.mozillazine.org/`

Flag	**Value**
browser.cache.check_doc_frequency	1
browser.cache.disk.enable	FALSE
browser.cache.memory.enable	FALSE
network.http.spdy.enabled.http2	TRUE/FALSE

Table 1: The Firefox configuration flags set in the experiment.

After the smartphone and the laptop have been connected to the Power Monitor and the power measurement UI (Power Tool UI) has been opened the experiment is conducted as follows:

1. Start the Power Monitor with an output voltage of 3.7V and sampling rate of 5000Hz
2. Boot the phone and ensure it has been configured as described above
3. Connect the smartphone to a laptop via its micro USB connection (for ADB communication)
4. Install Firefox 46.0 via ADB and configure it as described above (partially automated)
5. Execute the test suite n number of times for either HTTP2 or HTTP1.1 (fully automated).

Upon executing the test suite the power monitoring is manually started via the Power Tool UI for recording the energy consumption of the test run. The rundown of the *test suite* is as follows:

1. Close stale instances of Firefox (if any)
2. Ensure brightness is set to maximum and no screen dim is enabled (a fail-safe mechanism)
3. Upload experiment scripts to the phone via ADB
4. Wait t_{bwt} seconds and start Firefox (open at *about:blank*)
5. Wait t_{nwt} seconds and navigate to a target website
6. Swipe down the page 3-4 times to force dynamic/lazily loaded content to be loaded waiting t_{swt} seconds after each swipe
7. Navigate to a subpage and wait t_{nswt} seconds
8. Repeat steps **6-7** for five different subpages
9. Navigate to *about:blank* and repeat steps **5-8** for each target website

10. Wait t_{nwt} seconds, close Firefox and wait t_{bwt} seconds before concluding the experiment.

The wait times referred to above are given in Table 2. The tap wait time t_{twt} was used when tapping was required for loading more page content (e.g. tapping on a *Load more* button). In addition to running the test suite to gather energy consumption metrics, the test suite was also run separately to capture network traffic (payload sizes, latency, number of requests) via the Firefox WebIDE[5].

Variable	**Duration (s)**	**Description**
t_{nwt}	15	Navigation wait time
t_{nswt}	10	Subpage navigation wait time
t_{bwt}	5	Base wait time
t_{swt}	3	Swipe wait time
t_{twt}	2	Tap wait time

Table 2: Wait time variables used in the experiment.

1.3 Results

The average network latency – or Round Trip Time (RTT) – and packet loss measured when connecting to the three target web services (Yahoo, Instagram, Flickr) via the Aalto open WiFi is presented in Table 3.

Web service	**RTT (ms)**	**RTT std. dev (ms)**	**Packet loss**
Yahoo	158	32	4.7%
Instagram	116	10	0.3%
Flickr	132	12	0.3%

Table 3: RTT and packet loss experienced when being connected to the target websites via the Aalto open WiFi during the experiment.

For both of the protocols (HTTP1.1 and HTTP2) the test suite was run three times. Figure 2 visualizes the average power level for each of the test runs. The average is computed as a moving average using R[6] and a generalized additive model (GAM) provided by the *geom_smooth()* function of the *ggplot2*[7] plotting library (version 2.1.0). As seen in the figure, it took a total of 400 seconds to complete a single run of the test suite.

Figure 3 presents the power levels of a test run done over HTTP1.1. The samples (x-axis) have been averaged so that the average power level is calculated at intervals of 5000Hz (sampling rate of the Power Monitor). That is, each data point represents the average of a segment of 5000 data points (power levels). The smoothed (moving) average is plotted for easier comparison. Observations

[5]Firefox WebIDE - `https://developer.mozilla.org/en-US/docs/Tools/WebIDE`
[6]R: statistical computing framework - `https://www.r-project.org/`
[7]ggplot2 - `http://docs.ggplot2.org/`

made on the data are labeled with letters *A*-*E*. These observations are discussed later in Chapter 1.4.

A summary of the mean power and total energy consumption of the three experiment runs for both HTTP2 and HTTP1.1 are given in Table 4. The metrics are calculated for samples within the time frame of 24-395 seconds, which corresponds to the time when the first navigation is executed up until the time when navigating off to *about:blank* after the last target website.

For more accurate analysis of the effect of network traffic, Figure 4 provides a network request pie chart for the payload sizes, latency and number of requests for both HTTP2 and HTTP1.1. This data is based on a single run executed for both HTTP2 and HTTP1.1 in addition to the actual experiment.

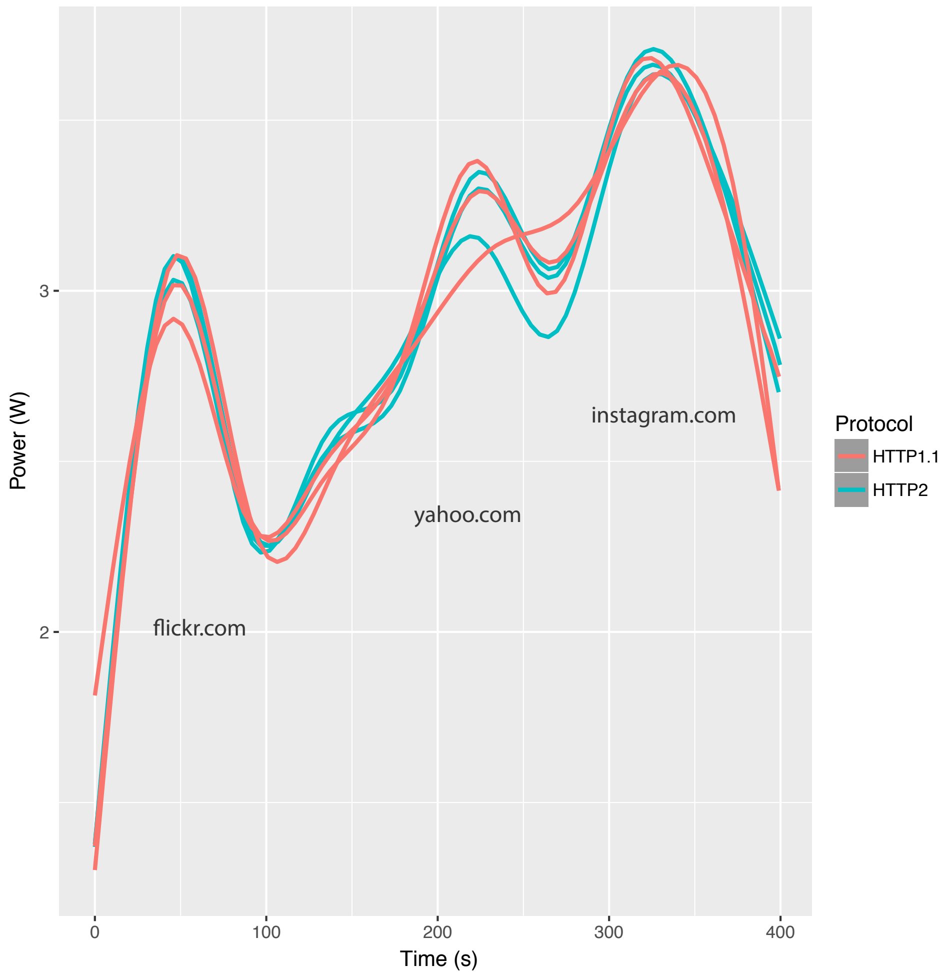


Figure 2: Average power level of each six test runs with HTTP2 and HTTP1.

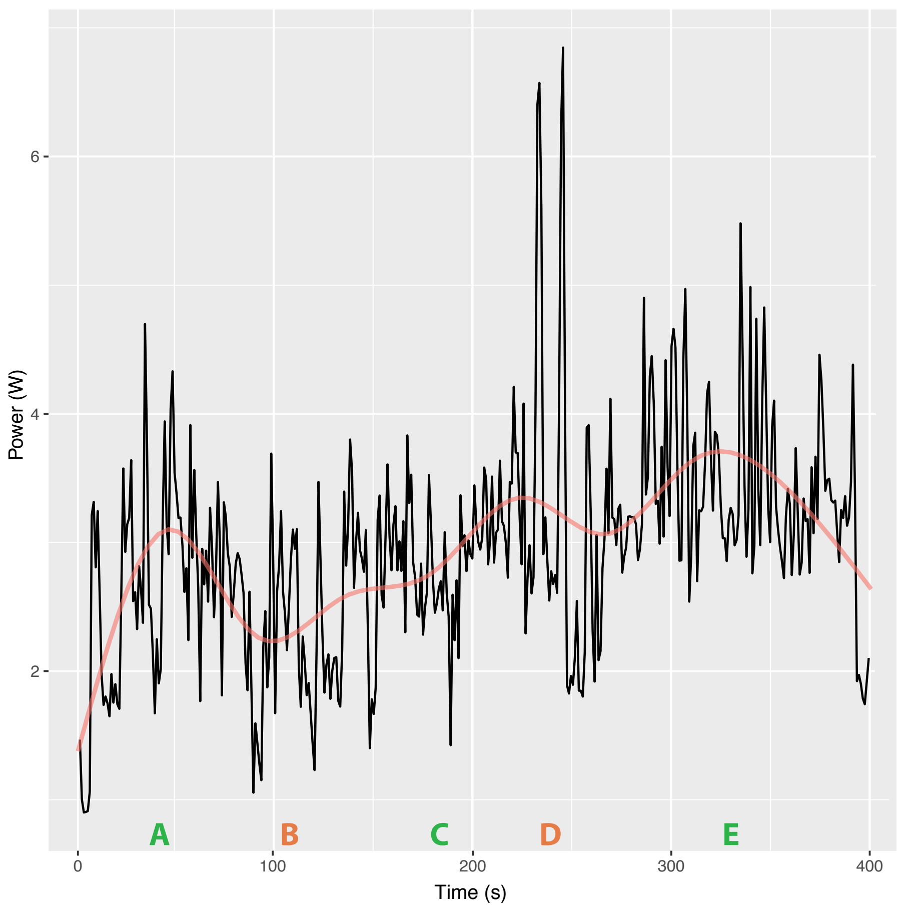
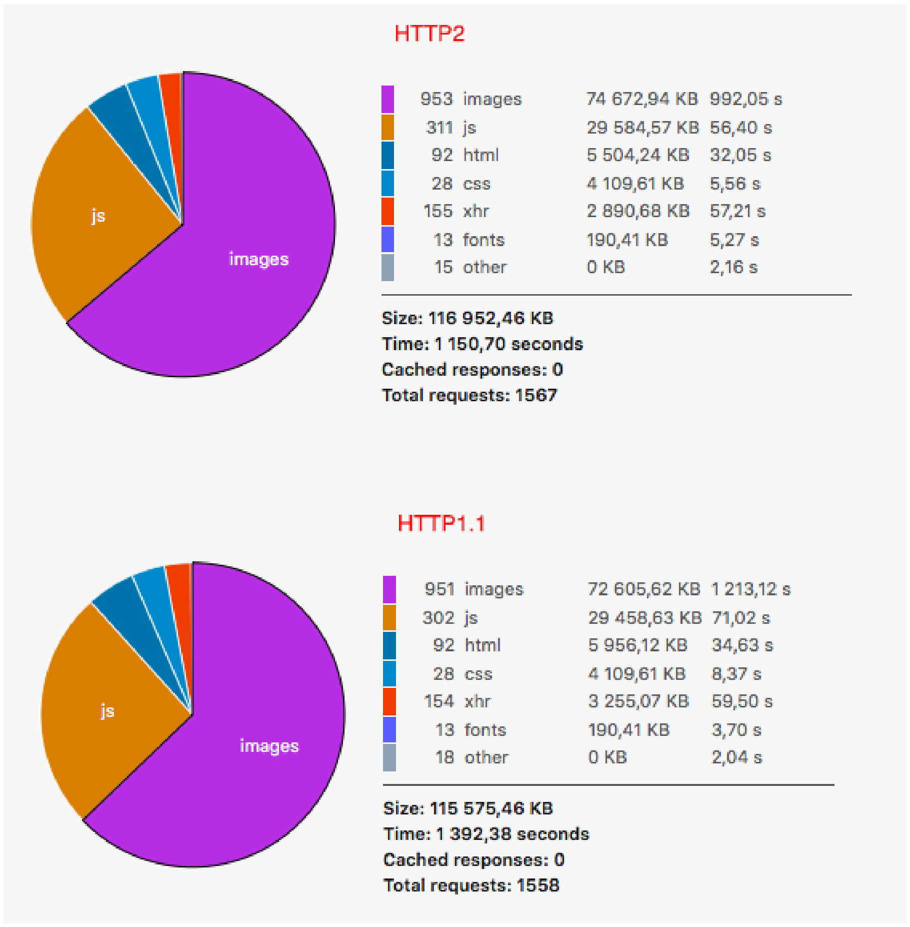


Figure 3: Average power levels (of every 1s segment) and the smoothed average of a test run done over HTTP1.1.

	HTTP1.1	**HTTP2**
Mean (W)	2.987	2.995
Std.dev (W)	0.984	1.007
Total (J)	1114.2	1117.2

Table 4: Energy consumption of HTTP1.1 and HTTP2 over the time frame of 24-395 seconds (from first navigation up until the exit navigation).



HTTP2

🟪	953	images	74 672,94 KB	992,05 s
🟧	311	js	29 584,57 KB	56,40 s
🟦	92	html	5 504,24 KB	32,05 s
🟦	28	css	4 109,61 KB	5,56 s
🟥	155	xhr	2 890,68 KB	57,21 s
🟦	13	fonts	190,41 KB	5,27 s
⬜	15	other	0 KB	2,16 s

Size: 116 952,46 KB
Time: 1 150,70 seconds
Cached responses: 0
Total requests: 1567

HTTP1.1

🟪	951	images	72 605,62 KB	1 213,12 s
🟧	302	js	29 458,63 KB	71,02 s
🟦	92	html	5 956,12 KB	34,63 s
🟦	28	css	4 109,61 KB	8,37 s
🟥	154	xhr	3 255,07 KB	59,50 s
🟦	13	fonts	190,41 KB	3,70 s
⬜	18	other	0 KB	2,04 s

Size: 115 575,46 KB
Time: 1 392,38 seconds
Cached responses: 0
Total requests: 1558

Figure 4: The network traffic averages for HTTP2 and HTTP1.1 when browsing the target websites.

1.4 Discussion

One of the early goals was to compare the energy consumption of the HTTP protocols in different network (latency) conditions as in [4] it was found that significant difference in the energy consumption was observed in the vincinity of 250ms, but not on low or high latencies. The objective was to test if this indeed was the case. Unfortunately however, implementing a setup for reliable latency adjustment (e.g proxying the traffic via VPN) was not done due to time constraints. Therefore, experiments were only performed under the Aalto open WiFi. Another option would have been to force the phone to use a 2G or 3G network. However, 2G proved too unreliable (high packet loss), while 3G actually provided relatively low latency (150ms) similar to that of the Aalto open WiFi connection – most likely due to the nearby location of the network antenna.

As seen in Table 3 Yahoo exhibited higher latency and significantly higher packet loss. This was believed to be due to the fact that the pages include a relatively high number of ads from various mobile advertisement providers. Furthermore, it can be argued that the CDNs (Content Delivery Network) for Yahoo are less ideally positioned then that of Instagram and Flickr. However, the average latencies are still within a reasonable range (116-158ms) to assume a low latency testbed.

Figure 2 shows that the energy consumption difference between the two protocols is minimal. Three runs for both of the protocol is not however enough to draw conclusions. The bump in the average power level for `yahoo.com` (also seen in the two power spikes in Figure 3) can be attributed to *layout trashing* observed when swiping down `https://yahoo.com/style` on mobile. Unfortunately, this was beyond the control of the authors, so it was later decided that */style* would be omitted from the subpages to navigate to in any future experiments. One of the HTTP1.1 runs did not seem to spike up in this manner, though, when browsing `yahoo.com`. This only goes to tell how important it would be to have had more test runs executed (to minimize the effect of anomalies).

Some interesting observations can be made from the graph in Figure 3. First, it can be seen that the base power of the phone is around 1W. Launching and running Firefox increases the base power level to the vincinity of 1.8W (at 12-24 seconds). Labels *A* and *B* represent the power levels when browsing `instagram.com` (at 24-120 seconds). The reason why the power level suddenly drops well below 2W twice (*B*) was due to the fact that the page visited (`https://flickr.com/photos/tags`) was light on content, and was actually navigated to twice, consecutively. As the URL would not change when navigated to the second time, Firefox was smart enough to not load the page preserving energy in the process. The fact that this subpage was visited twice in a row was an implementation bug missed by the authors when conducting the tests.

Label *C* and *D* represent the power levels when browsing `flickr.com` (at 135-250 seconds). The spikes (*D*) were due to layout trashing as referred to earlier. Last, label *E* represents the browsing of `instagram.com` (at 260-395 seconds). Judging by the graph, it seems as if Instagram would be the most energy inefficient. However, as the order of target websites in the corpus was not randomized, such conclusions can not be drawn. It might very well be that the progressive increase in the power level is due to increasing device temperature, or Firefox's excess allocation of memory and/or processes over time.

The energy consumption results summarized in Table 4 show that the consumption was virtually identical between HTTP2 and HTTP1.1 across the test runs. However, Figure 4 indicates that HTTP2 was more performant in fetching assets over network. For example, HTTP2 spent 20% less time on average for fetching assets. It could be argued that the better latency of HTTP2 comes with the price of slightly higher power consumption.

An effort was given to follow the best practices of software energy experimentation as per **ENERGISE**[8] – a mnemonic coined by the authors of [2]. Due to time constraints the tests were run only on one Firefox version (46.0). Moreover, only three test runs were done for both protocols, which clearly is not enough to draw any statistically significant conclusions. However, in other aspects of the ENERGISE practices the experiment was conducted to the authors' best ability. Especially, the automation of the test suite was a significant achievement, which will help in improving and adjusting the testbed for part II of this study.

In addition to evaluation of potential power saving mechanisms, a few items are left as TODOs for the part II of this study.

2 Part II

Evaluation of potential power saving mechanisms...

TODOs:

- proxy traffic via a VPN to simulate a high latency condition
- randomize the order of the target websites in the corpus
- increase test run count to at least 10 for better confidence
- increase generated network traffic (increase corpus or implement deeper navigation patterns)
- evaluation of the effect of HTTP2 best practices on energy consumption

[8]Software energy experimentation best practices: `https://github.com/ds4se/chapters/blob/c6f960b5955ef835752f265eed7337ff51cf35e0/abramhindle/energymining.md#lets-energise-your-software-energy-experiments`

2.1 Results

TBD

2.2 Proposal for Improvements

TBD

2.3 Conclusions

TBD

References

[1] Google Inc. SPDY Performance on Mobile Networks, 2016. `http://developers.google.com/speed/articles/spdy-for-mobile`. Accessed 29.04.2016.

[2] Abram Hindle, Alex Wilson, Kent Rasmussen, E. Jed Barlow, Joshua Charles Campbell, and Stephen 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, New York, NY, USA, 2014. ACM.

[3] Q-Success. Usage of HTTP/2 for websites, 2016. `http://w3techs.com/technologies/details/ce-http2/all/all`. Accessed 30.04.2016.

[4] Varun Sapra Shaiful Alam Chowdhury and Abram Hindle. Is HTTP/2 More Energy Efficient Than HTTP/1.1 for Mobile Users? *PeerJ Preprints*, 2015.

[5] Daniel Stenberg. HTTP2 Explained. *ACM SIGCOMM Computer Communication Review*, 44(3):120–128, 2014.