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`

May 16, 2016

1 Part I

1.1 Introduction

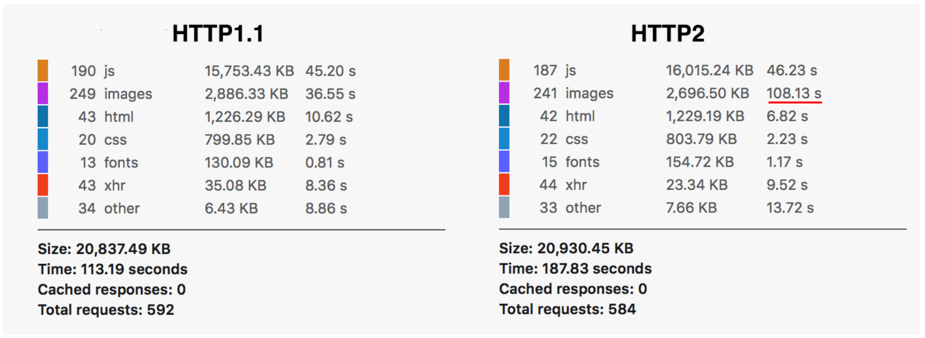
With the increasing computation power and connection speed of mobile phones in recent years, the amount of data flowing through the Internet to mobile end devices continues to grow. While web applications have become ever more complex and the amount of HTML, JavaScript and CSS required to render a web 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. Three different web services were chosen as the target websites: `flickr.com`, `instagram.com` and `yahoo.com`. These web services were selected based on their popularity and support for HTTP2 clients (browsers). The original corpus included Facebook, Twitter and Weather.com as well. They were however omitted from the corpus due to various reasons: the network traffic generated by Facebook was highly optimized, and therefore, it was assumed that it would not introduce any interesting statistical differences between HTTP1 and HTTP2. 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.



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

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

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) running on Samsung S4 (with Android 5.1.1). 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 – Mozilla's internal tool for testing Firefox on mobile – missed up-to-date 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 in some cases it requires root privilegs. Therefore the Samsung S4 used for the experiment was rooted using ClockworkMod recovery (version 6.0.4.7) and CyanogenMod (version 12). 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

[1] Appium - `appium.io`
[2] Robocop - `https://wiki.mozilla.org/Auto-tools/Projects/Robocop`
[3] `https://www.msoon.com/LabEquipment/PowerMonitor/`

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 orchestrating 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 introduced 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 also disabled. The browser HTTP2 support was toggled on and off via the Firefox configuration interface (`about:config`). 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].

Table 1: The Firefox configuration flags set in the experiment to disable caching and toggle HTTP2 support.

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

After connecting the smartphone and the laptop to the Power Monitor and opening the power measurement UI (Power Tool UI) the experiment was conducted as follows:

1. Start the Power Monitor setting 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 the laptop via USB (for ADB communication)
4. Install Firefox 46.0 via ADB and configure its settings as described above (partially automated)

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

5. Execute the test suite n number of times for either HTTP2 or HTTP1.1 (fully automated).

Upon executing the test suite power monitoring is manually triggered via the Power Tool UI to record the energy consumption of the test runs. 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
3. Upload experiment scripts to the phone via ADB
4. Wait t_{bwt} seconds and start Firefox (opens at *about:blank*)
5. Wait t_{nwt} seconds and navigate to a target website
6. Swipe down the page 5-6 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 4-6 different subpages
9. Navigate to *about:blank* and repeat steps **1-8** for each target website
10. Wait t_{nwt} seconds, close Firefox and wait another 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].

Table 2: Wait time variables used in the experiment.

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

[5]Firefox WebIDE - `https://developer.mozilla.org/en-US/docs/Tools/WebIDE`

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. 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 (six) test runs. The results are plotted 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 over each 5000Hz interval (the sampling rate set for the Power Monitor). That is, each data point represents the average of a segment of 5000 data points (power levels). The moving (smoothed) average is included in the plot for comparison. Observations made based on the graph are labeled with letters *A-E*, and 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 the respective network traffic statistics, Figure 4 provides a visualization of the payload sizes, latency and number of requests for both HTTP2 and HTTP1.1. This data is based on two individual runs done over both HTTP1.1 and HTTP2.

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

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

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

[6]R: statistical computing framework - `https://www.r-project.org/`

[7]ggplot2 - `http://docs.ggplot2.org/`

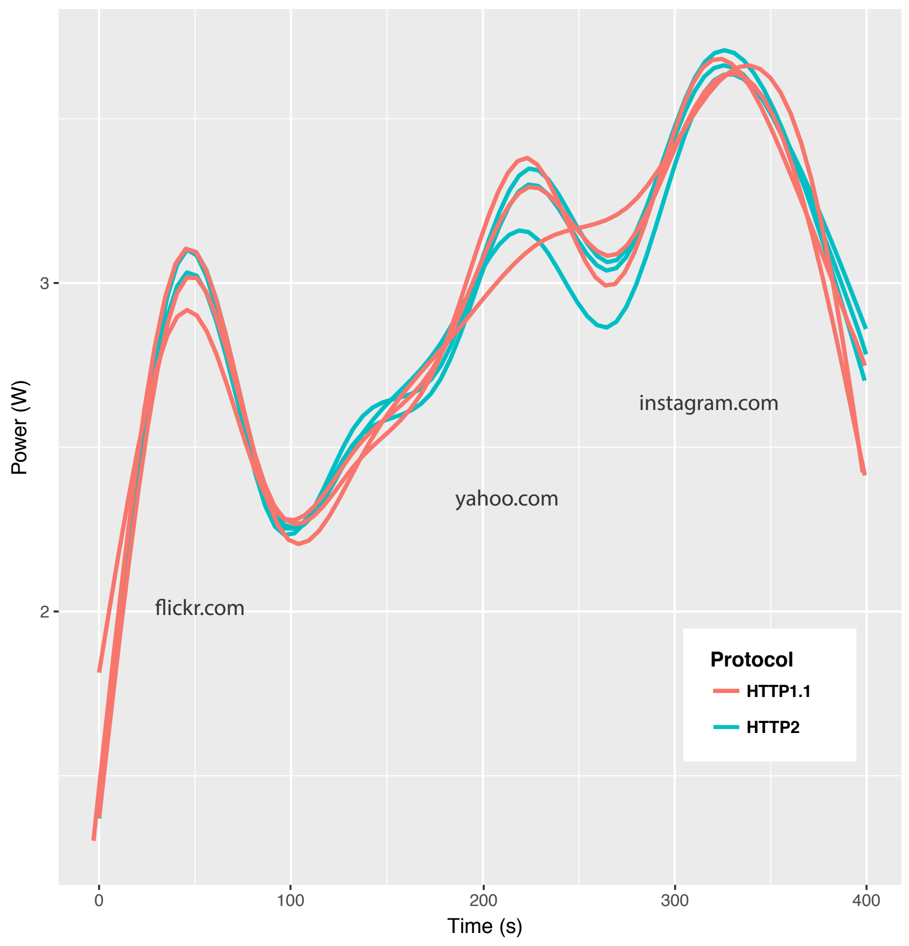


Figure 2: Smoothed average power draw of each six test runs over HTTP1.1 and HTTP2.

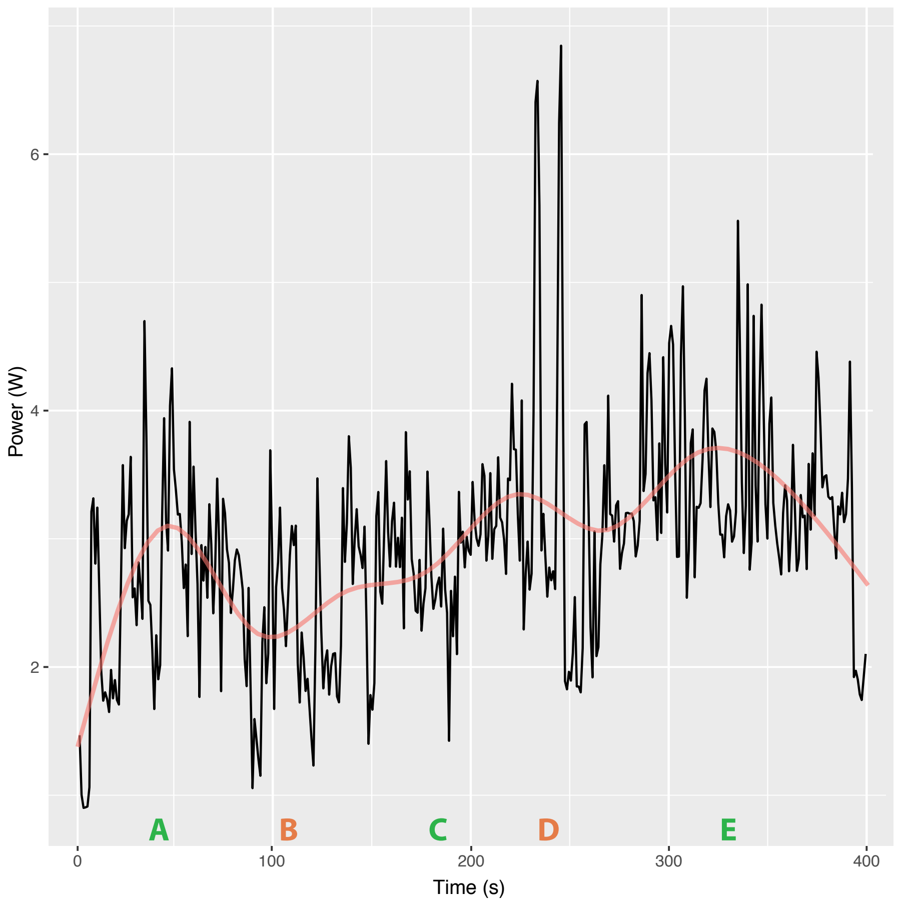
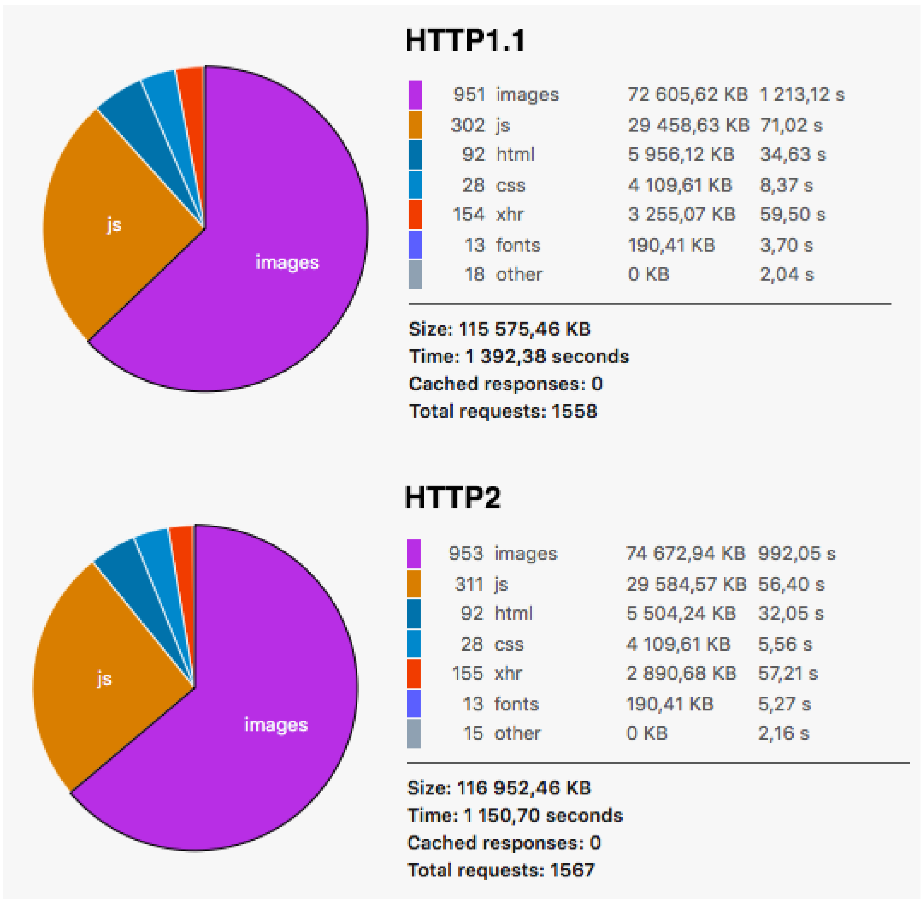


Figure 3: Average power draw and smoothed average of a test run done over HTTP1.1.



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

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

Figure 4: The network traffic statistics for single test suite runs over HTTP1.1 and HTTP2.

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 lower or higher 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 in part I of this study. An alternative approach 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.com` exhibited higher latency and 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 test conditions.

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* (by the GPU) observed when swiping down `https://yahoo.com/style` on mobile. Thus, it was decided that */style* be omitted from the corpus for part II experiments. One of the HTTP1.1 runs did not seem to exhibit this "spiky" behaviour, though. This only goes to show how important it would have been to perform more test runs to minimize the effects of anomalies.

Some interesting observations can be made from the graph in Figure 3. First, it can be seen (from the raw data) that the base power of the phone is around 1.2W. Launching and running Firefox increases the base power draw to the vincinity of 1.65W (12-24 seconds). Labels A and B represent the power draw when browsing `instagram.com` (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 (fixed in part II).

Label C and D represent the power draw 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 draw 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 HTTP1.1 and HTTP2 across the test runs. However, Figure 4 suggests that HTTP2 had better network performance. For example, HTTP2 spent 20% less time on average for fetching assets. Thus, it could be argued that HTTP2 had higher energy utility – 22% higher kB/s/J based on Table 4 and Figure 4.

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 helped in improving and adjusting the testbed for part II of this study.

2 Part II

Based on part I observations and feedback the following improvements/adjustments were made to the testbed:

- increase of test run count to 10 for better confidence
- traffic throttling via a VPN for simulating high latency conditions
- URL manifest generation and more in-depth network traffic metrics
- test suite adjustments:
  - changed test suite to be run separately for each target website instead of profiling them all in one go
  - omitted `yahoo.com/style` from the corpus
  - removed duplicate navigation to `flickr.com/photos/tags`

High latency conditions were simulated by tunneling the smartphone network traffic via a VPN located in US west coast. The average ping measured varied around 300 ± 60ms. Accurate high latency RTTs are provided in Table 7.

The goal of part II of the assignment was to propose solutions for improving the energy-efficiency of the mobile application/service studied in part I. Since

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

this study investigated the performance HTTP protocols rather than an individual application/service, the solution is proposed in form of HTTP2 best practices for application developers. The rest of this paper is organized as follows: Chapter 2.1 presents the results of part II after applying the aforementioned testbed improvements. Chapter 2.2 continues with a dicussion of the results and presents an outline for HTTP2 best practices. Last, Chapter 2.3 concludes with a summary of the study and suggestions for future work.

2.1 Results

In part II energy consumption and traffic metrcis were captured separately for each target website so that any increase in device temperature or tail energy from previous activity would not be reflected on the power draw of the consecutive run. The *Visit Duration* given in Table 5 denotes the time between the initial navigation to and the exit navigation from a target website. All results have been computed based on this time frame.

Table 5: URL manifest; the target websites and subpages visited. The payload and RTT values are the average of all the runs (HTTP1.1 and HTTP2) executed in low latency conditions.

Site	**Visit Duration** (s)	**Avg. Payload** (kB)	**Avg. RTT** (ms)
https://flickr.com	170	64 261	131.6 ± 6.6
/explore			
/photos/tags/primavera			
/photos/tags/amazing			
/photos/tags/rural			
/photos/tags/brazil			
https://instagram.com	200	32 937	112.7 ± 4.7
/nhl			
/binghuiliu			
/jordanspieth			
/warriors			
/nba			
/ufc			
https://yahoo.com	150	20 162	133.3 ± 8
/tech			
/news			
/business			
/tv			

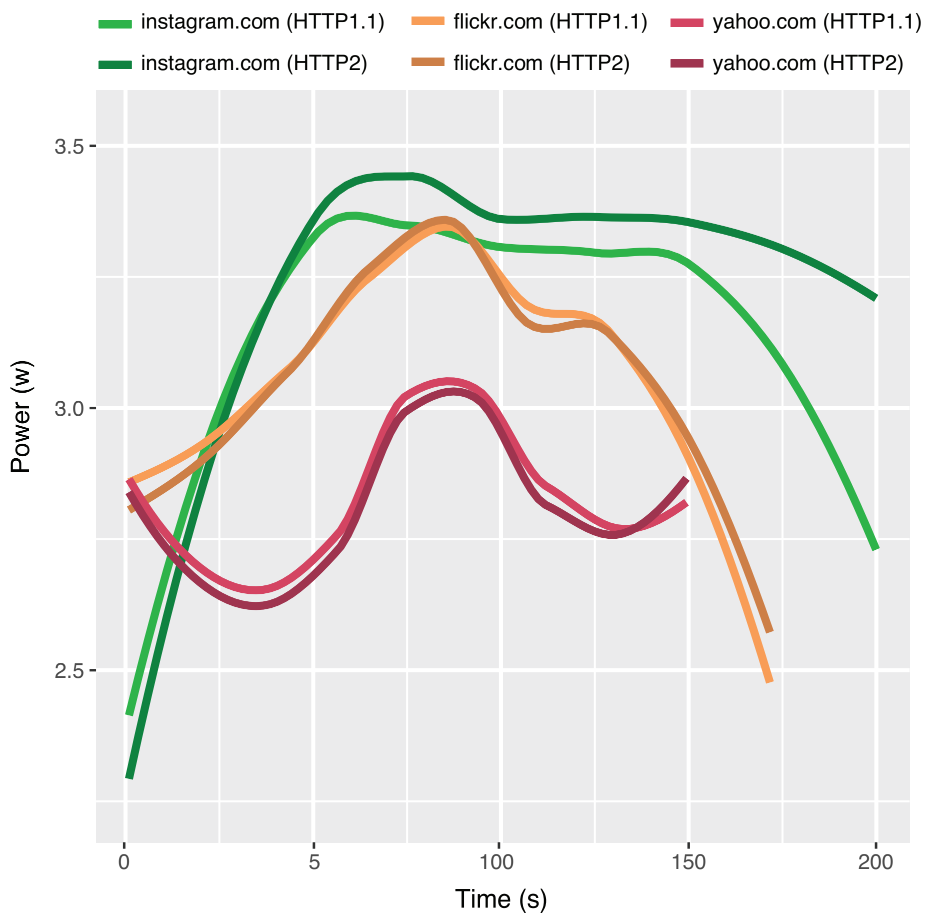


Figure 5: Average power draw of HTTP1.1 and HTTP2 per target website (low latency, 100-130ms).

Table 6: Energy consumption of the target websites over HTTP1.1 & HTTP2. HL denotes high latency conditions of 300ms RTT (110-130ms otherwise).

HTTP1.1 / HTTP2	**Total energy** (J)	**Power average** (W)	**Power std.dev** (W)
Flickr	519.7 / 520.1	3.075 / 3.077	0.832 / 0.748
Flickr (HL)	659.8 / 532.3	3.315 / 3.149	0.841 / 0.797
Instagram	631.4 / 646.2	3.172 / 3.247	0.705 / 0.754
Instagram (HL)	659.8 / 670.0	3.315 / 3.366	0.841 / 0.857
Yahoo	417.4 / 413.9	2.801 / 2.778	0.624 / 0.597
Yahoo (HL)	447.6 / 446.6	3.004 / 2.997	0.586 / 0.611

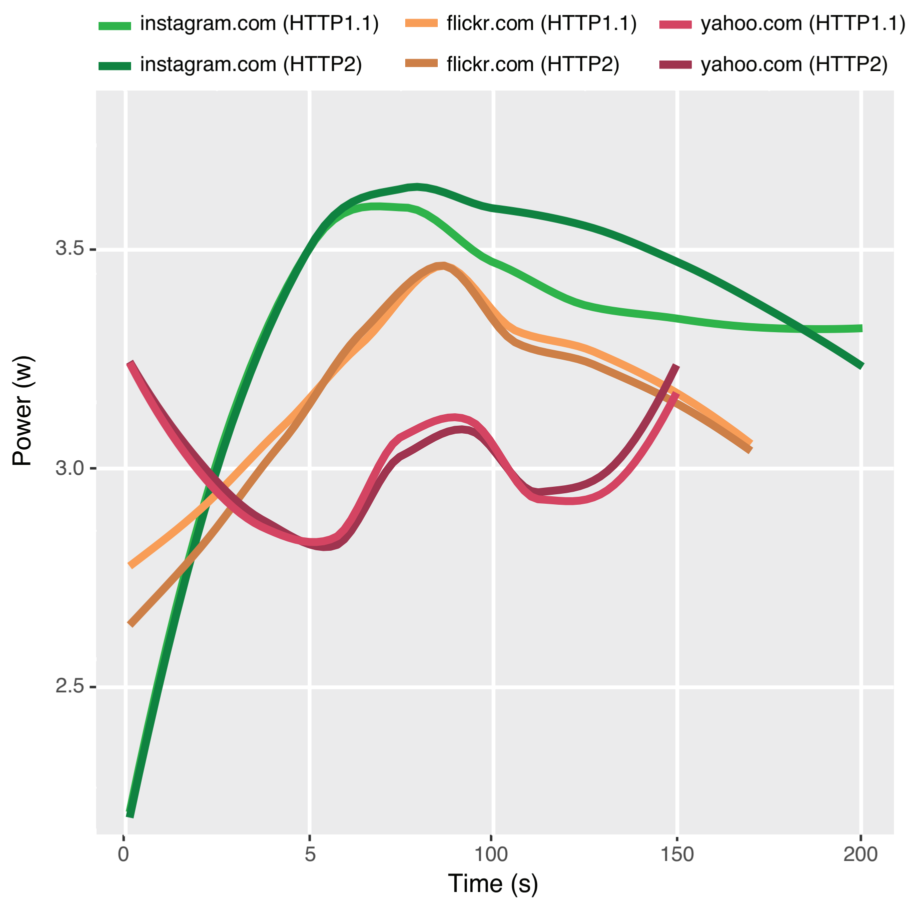
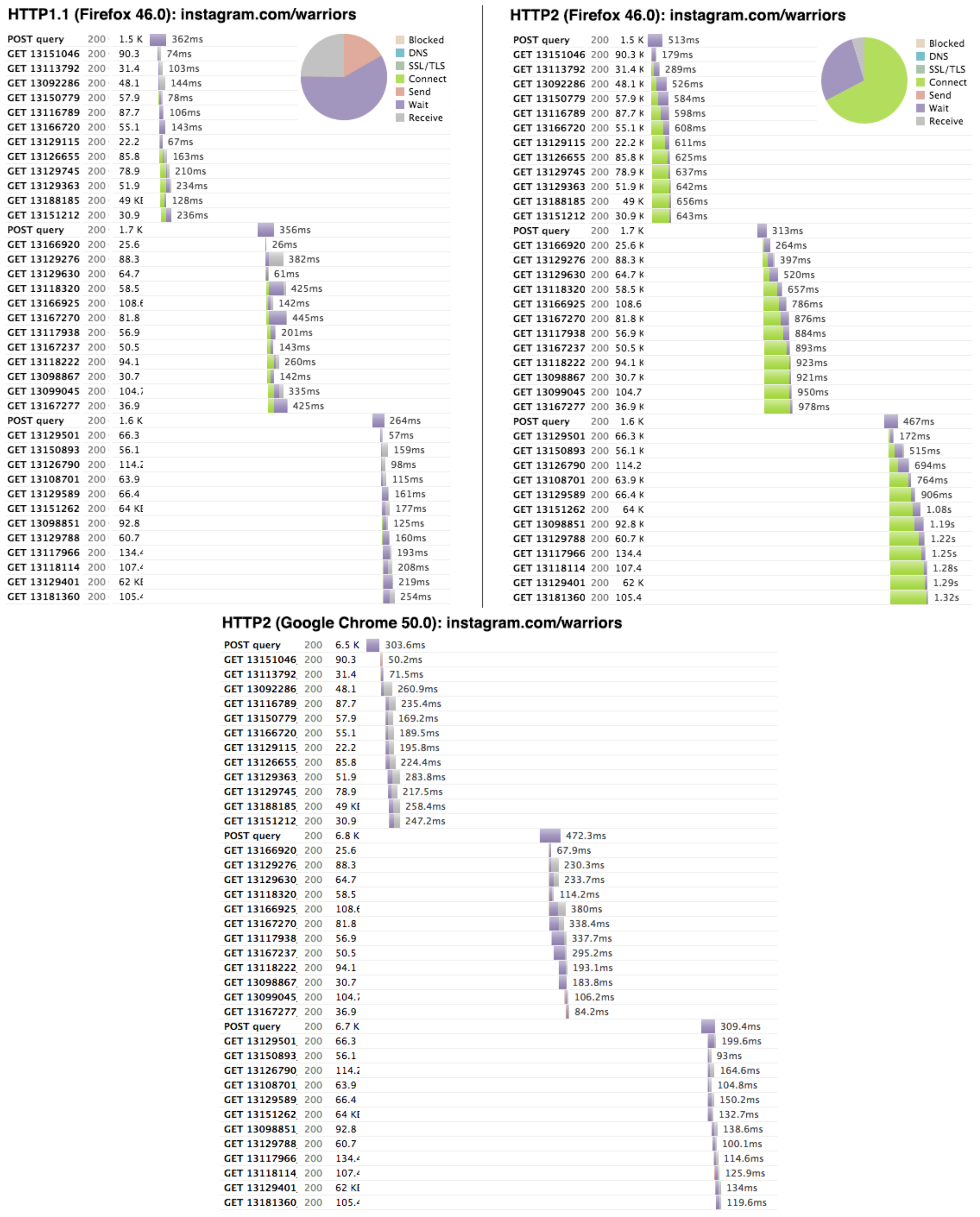


Figure 6: Average power draw of HTTP1.1 and HTTP2 per target website (high latency, 300ms).

Table 7: Network traffic statistics and energy utility (kB/s/J) for each target website during the (low latency) test runs. The average latency and energy utility under high latency conditions are also included for comparison.

HTTP1.1 / HTTP2	**Flickr**	**Instagram**	**Yahoo**
Total Payload (kB)	63669 / 64853	33210 / 32664	19833 / 20491
Total Response Time (s)	1135 / 936	117 / 170	127 / 119
Number of Requests	587 / 606	469 / 464	511 / 509
Total Bit Rate (kB/s)	56.1 / 69.3	283.8 / 192.1	156.2 / 172.2
Image Bit Rate (kB/s)	8.1 / 10.2	51.9 / 33.2	18.8 / 24.4
JS Bit Rate (kB/s)	59.7 / 64.3	354.5 / 301.8	87.2 / 87.8
Energy Utility (kB/s/J)	0.108 / 0.133	0.449 / 0.297	0.374 / 0.416
Latency / RTT (ms)	131.6 ± 6.6 (HL: 306 ± 50.5)	112.7 ± 4.7 (HL: 295.6 ± 56.3)	133.3 ± 8 (HL: 304 ± 77.8)



HTTP1.1 (Firefox 46.0): instagram.com/warriors

POST query	200	1.5 K	362ms
GET 13151046	200	90.3	74ms
GET 13113792	200	31.4	103ms
GET 13092286	200	48.1	144ms
GET 13150779	200	57.9	78ms
GET 13116789	200	87.7	106ms
GET 13166720	200	55.1	143ms
GET 13129115	200	22.2	67ms
GET 13126655	200	85.8	163ms
GET 13129745	200	78.9	210ms
GET 13129363	200	51.9	234ms
GET 13188185	200	49 K	128ms
GET 13151212	200	30.9	236ms
POST query	200	1.7 K	356ms
GET 13166920	200	25.6	26ms
GET 13129276	200	88.3	382ms
GET 13129630	200	64.7	61ms
GET 13118320	200	58.5	425ms
GET 13166925	200	108.6	142ms
GET 13167270	200	81.8	445ms
GET 13117938	200	56.9	201ms
GET 13167237	200	50.5	143ms
GET 13118222	200	94.1	260ms
GET 13098867	200	30.7	142ms
GET 13099045	200	104.7	335ms
GET 13167277	200	36.9	425ms
POST query	200	1.6 K	264ms
GET 13129501	200	66.3	57ms
GET 13150893	200	56.1	159ms
GET 13126790	200	114.2	98ms
GET 13108701	200	63.9	115ms
GET 13129589	200	66.4	161ms
GET 13151262	200	64 K	177ms
GET 13098851	200	92.8	125ms
GET 13129788	200	60.7	160ms
GET 13117966	200	134.4	193ms
GET 13118114	200	107.4	208ms
GET 13129401	200	62 K	219ms
GET 13181360	200	105.4	254ms

HTTP2 (Firefox 46.0): instagram.com/warriors

POST query	200	1.5 K	513ms
GET 13151046	200	90.3 K	179ms
GET 13113792	200	31.4 K	289ms
GET 13092286	200	48.1 K	526ms
GET 13150779	200	57.9 K	584ms
GET 13116789	200	87.7 K	598ms
GET 13166720	200	55.1 K	608ms
GET 13129115	200	22.2 K	611ms
GET 13126655	200	85.8 K	625ms
GET 13129745	200	78.9 K	637ms
GET 13129363	200	51.9 K	642ms
GET 13188185	200	49 K	656ms
GET 13151212	200	30.9 K	643ms
POST query	200	1.7 K	313ms
GET 13166920	200	25.6 K	264ms
GET 13129276	200	88.3 K	397ms
GET 13129630	200	64.7 K	520ms
GET 13118320	200	58.5 K	657ms
GET 13166925	200	108.6	786ms
GET 13167270	200	81.8 K	876ms
GET 13117938	200	56.9 K	884ms
GET 13167237	200	50.5 K	893ms
GET 13118222	200	94.1 K	923ms
GET 13098867	200	30.7 K	921ms
GET 13099045	200	104.7	950ms
GET 13167277	200	36.9 K	978ms
POST query	200	1.6 K	467ms
GET 13129501	200	66.3 K	172ms
GET 13150893	200	56.1 K	515ms
GET 13126790	200	114.2	694ms
GET 13108701	200	63.9 K	764ms
GET 13129589	200	66.4 K	906ms
GET 13151262	200	64 K	1.08s
GET 13098851	200	92.8 K	1.19s
GET 13129788	200	60.7 K	1.22s
GET 13117966	200	134.4	1.25s
GET 13118114	200	107.4	1.28s
GET 13129401	200	62 K	1.29s
GET 13181360	200	105.4	1.32s

HTTP2 (Google Chrome 50.0): instagram.com/warriors

POST query	200	6.5 K	303.6ms
GET 13151046	200	90.3	50.2ms
GET 13113792	200	31.4	71.5ms
GET 13092286	200	48.1	260.9ms
GET 13116789	200	87.7	235.4ms
GET 13150779	200	57.9	169.2ms
GET 13166720	200	55.1	189.5ms
GET 13129115	200	22.2	195.8ms
GET 13126655	200	85.8	224.4ms
GET 13129363	200	51.9	283.8ms
GET 13129745	200	78.9	217.5ms
GET 13188185	200	49 K	258.4ms
GET 13151212	200	30.9	247.2ms
POST query	200	6.8 K	472.3ms
GET 13166920	200	25.6	67.9ms
GET 13129276	200	88.3	230.3ms
GET 13129630	200	64.7	233.7ms
GET 13118320	200	58.5	114.2ms
GET 13166925	200	108.6	380ms
GET 13167270	200	81.8	338.4ms
GET 13117938	200	56.9	337.7ms
GET 13167237	200	50.5	295.2ms
GET 13118222	200	94.1	193.1ms
GET 13098867	200	30.7	183.8ms
GET 13099045	200	104.7	106.2ms
GET 13167277	200	36.9	84.2ms
POST query	200	6.7 K	309.4ms
GET 13129501	200	66.3	199.6ms
GET 13150893	200	56.1	93ms
GET 13126790	200	114.2	164.6ms
GET 13108701	200	63.9	104.8ms
GET 13129589	200	66.4	150.2ms
GET 13151262	200	64 K	132.7ms
GET 13098851	200	92.8	138.6ms
GET 13129788	200	60.7	100.1ms
GET 13117966	200	134.4	114.6ms
GET 13118114	200	107.4	125.9ms
GET 13129401	200	62 K	134ms
GET 13181360	200	105.4	119.6ms

Figure 7: A snippet of the network waterfall for `instagram.com/warriors` when browsed over HTTP1.1 and HTTP2 with Firefox 46.0 and Google Chrome 50.0. Browsing `instagram.com` over HTTP2 on Firefox 46.0 seemed to suffer from long delays in establishing a TCP connection when fetching image assets.

2.2 Discussion

Initially for part II measurements an attempt was made to manually turn on Android's *Performance Governor* during the experiments to ensure a more predictable, base CPU power draw. However, this did not turn out to be possible as is lead to instable, laggy behaviour.

The following question were laid out based on part I results:

> *Did you get exactly the same content every time?*
>
> Slight variance in the total payload size and number of requests for each of the test runs was observed. This was likely due to varying network conditions both in the client's and the server's end. However, based on the raw data and author's observations these variations were minimal enough to be cancelled out by having the test suite executed multiple times (10). No attempt was made for automatically detecting significant changes in the network conditions during the experiment. However, nothing in the final data pointed to any discrepancies. Only a couple of test runs were re-run as the power draw was noticeably different (contained anomalies).
>
> *What are the potential causes of the difference in energy consumption? Do they transfer data in a different way?*
>
> HTTP2 is able to handle multiple request more efficiently by multiplexing them (using streams). Requests made on the same connection can be responded to out of order, which means the client can use a single TCP connection to make all their requests. When properly supported by both the client and the server, this results in reduced memory, network and CPU load for both parties as a new TCP connection need not be opened for each request. The respective payload size for HTTP2 is slightly bigger due to the higher number of required HTTP header fields.

Tables 7 and 6 show that HTTP2 provides better energy utility, especially in the case of high latency conditions. This was not however the case with Instagram, which – much like Weather.com in part I of the study – exhibited longer response times when transferring image assets. This in turn lead to higher energy consumption in both low and high latency scenarios (even though in high latency conditions the difference narrowed down a bit) as the TCP connections were kept alive for a longer time. This discrepancy was believed to have been caused by an incompability between the client's (Firefox 46.0) and the server's HTTP2 implementation.The disabling of caching might also have affected the functionality of the server in unexpected ways. Analysing the network traffic waterfall of e.g. `instagram.com/warriors` as depicted in Figure 7 shows that Firefox 46.0 seemed to suffer from long delays in establishing a TCP connection over HTTP2. Google Chrome (50.0) did not exhibit this behaviour, which further supports the aforementioned assumption.

Further Figures 6 and 5 show, that in general not only the average power consumption is increased because of peak energy in higher latency settings, but that this power increase is present in the whole measurement. This can be attributed to the higher latency in the way, that the increased latency leads to longer communication times and an increased amount of communication. Therefore we see an increase in whole measurement, since we have an increased tail energy in the communications and this increase propagates from the peaks to the valleys of the graphs. Due to the fast navigation and most likely some background communication with the server (although much lower than in the peaks), the power measurements never drop to level of the ones in the low latency measurements. It is also not clear if Firefox operates in a higher power mode,if http2 is deactivated. At the moment there are no indications, that this might occur. While there is an increase in the power consumption in the slightly higher latency, it is apparent, that the access times of the websites for the user didn't change significantly. This can be seen in the figures 6 and 5 quite clearly. The peak energies still occur at the same time and also the drops in power levels occur at the same time. So the main communication to the server starts and ends nearly the in same amount of time, which indicates, that probably a less optimal burst size is chosen by the communication device, resulting in more frequent and therefore more expensive communication in the high latency setting. This observation can be backed by the experience made, while conducting the experiment. For the user there is no apparent difference in the loading times between using the WIFI network with or without VPN.

As for proposing improves in the energy consumption of the service/application no concrete improvements can be proposed. As mentioned in the beginning of part 2 it is not possible in the nature of the topic. Proposing improvements to HTTP1.1 or HTTP2 would result in basically a new standard with its own implications. Also the development of new Network protocols is not the aim of the course and any non speculative proposal would go beyond the scope of the amount of work of the course. What can be said is, that when migrating to, HTTP2 one has to take care not to use many of the HTTP1.1 typical features. With gaining use of HTTP2 the best practices in Web Development will be changing [**?**]:
Many of the latency avoiding workarounds of HTTP1.1 don't work that well with HTTP2 or become obsolete. The main tool of reducing latency in HTTP2 is multiplexing, while HTTP1.1 offers workarounds like pipelining, keep alive header or domain sharding. When using HTTP2 especially domain sharding becomes a problem, since it increases the amount of TCP connections, while multiplexing makes efficient use of a single connection. At the same time it removes the need of bundling multiple related assets such as CSS and Javascript files. This allows much more aggressive caching and requires less downloaded data if changes in one of assets occur. For example in HTTP1.1 one changing

asset would result in a whole bundle being reloaded. Although this offers a much better caching one has to consider older browsers (29% of the current Browsers) which prefer bundled assets and also the compression of a bundle might be give a better compression rate that separately compressing every asset [?]. Therefore our proposal would be to keep the HTTP1.1 optimized parts of a website for older browsers only. At the same time implement the rest of the Website purely in HTTP2 using the current best practices for HTTP2. Implementing parts in HTTP1.1 for HTTP2 supporting browsers should be completely avoided, unless there is a valid reason to resort to HTTP1.1 (for example compressed bundles or missing HTTP2 support of a service). In other words we propose to keep the standards as separate as possible and using HTTP2 where-ever possible, allowing HTTP2 to be as efficient as possible. This proposal obviously a bit utopic at the moment. But with gaining acceptance of HTTP2 and improve in HTTP2 optimization and quality of best practises, the practicality of the proposal will increase.

2.3 Conclusions and Future Work

In this experiment the automation of the test suite was crucial for ensuring accurate and reproducable data capture. Part II measurements alone took almost 7 hours to run (average test run duration of 200s × 120 test runs). It would have been impractical to do the measurements manually. For future work a automated stopping, starting and saving of the power measurement should be implemented using Monsoons power tool API. Also the amount of testruns, tested devices, latencies and tested websites should be increased. At the time of the experiment the amount of mainly HTTP2 based Websites and web services was still limited. Also the time constraints given by the length of the course and the time needed to implement the automation, made this unfortunately impossible.

Regarding the results we could see in our experiment, that in low latency conditions HTTP2 has little impact, whereas in high latency conditions it can be result in significant energy savings for high content websites. The main benefit in low latency settings seems to be the increased energy utility, while the total energy consumption is not greatly affected, when implemented right (compare instagram 7). In the high latency setting the impact was much greater. This was an expected result as indicated by the result of the experiments using GreenMiner [2]. It was surprising, that such popular websites as Instagram and Weather.com had major issues with HTTP2. In general HTTP2 is still in its infancy and requires careful configuration. As such, it can even increase energy consumption if compatibility with existing systems/browsers is not tested for. But it offers great potential for the future, considering it manages to sometimes outperform or if not at least match the performance and energy efficiency of HTTP1.1, which is already near its limit. Our proposal reflects this and HTTP2 seems to be the long needed successor for the deprecated HTTP1.1 Standard.

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.