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

Marko Raatikka[1]***, Lucas Triefenbach***[1]

[1]*Aalto University School of Science, Espoo, Finland*

Aalto University
School of Science

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 todays web – has failed to keep up with the pace.

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.

This work sets to study the impact of HTTP2 on mobile power consumption. Previous research on the topic has been done by Varun *et al.* in [1]. However, this study focuses on measuring the energy efficiency of HTTP1.1 vs. HTTP2 on mobile using common, real-life navigation patterns.

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

Design and Implementation

The energy efficiency of the HTTP1.1 and HTTP2 protocols were measured by generating large amounts of network traffic. Three different target websites were chosen: `instagram.com`, `yahoo.com` and `flickr.com`. The experiments were conducted using Firefox (version 46.0) running on Samsung S4 with Android 5.1.1. Androids low level input API was utilized for interacting with the browser in automated fashion. Inspiration for this approach was taken from *GreenMiner* – an open-source energy consumption measurement framework. Access to the Android input API required root privileges, and therefore the device was rooted using ClockworkMod recovery (version 6.0.4.7) and CyanogenMod (version 12).

For each target website a test suite was run 10 times in high (+300ms) & low (110-130ms) latency conditions both with and without HTTP2 support. This amounted to 120 individual test runs in total. Furthermore, a separate set of runs were conducted for collecting related network traffic statistics. 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 (Android Debug Bridge)
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 5-6 different subpages
9. Navigate to *about:blank* and repeat steps **1-9** 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 as follows:

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

The duration of a single test run varied by website: 150s (yahoo.com), 170s (flickr.com) and 200s (instagram.com). In part I of the experiment only six test runs were conducted (3 runs for each protocol in the same network conditions), the results of which are presented in **Figure 1** for comparison.

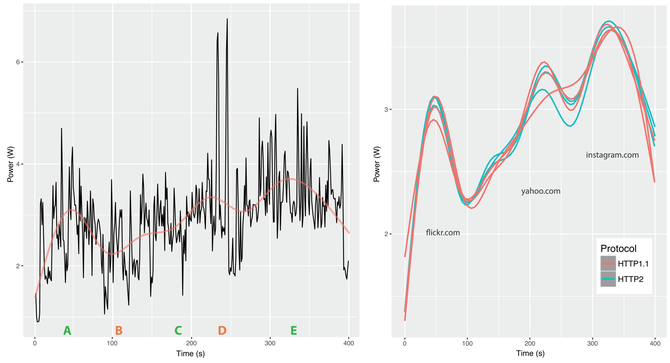

Figure 1: Average power draw of a test run done over HTTP1.1, and smoothed average of all six test runs done over HTTP1.1/HTTP2 during part I.

Results

Figure 1 and **Tables 1** and **2** present the energy consumption and network traffic statistics when browsing each of the three target websites over HTTP1.1 and HTTP2.2. Each line in **Figure 1** represents the average of 10 test runs done with a given combination of a target website, protocol and network condition. In **Tables 1** and **2** *HL* refers to measurements done in high latency conditions ($\sim$ 300ms).

Table 1: Energy consumption of the target websites over HTTP1.1 and HTTP2.

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

Table 2: Network traffic statistics of each target website during the test runs.

HTTP1.1 / HTTP2	Flickr	Instagram	Yahoo
Total Payload (kB)	63669 / 64853	33210 / 32664	19833 / 20491
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
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)

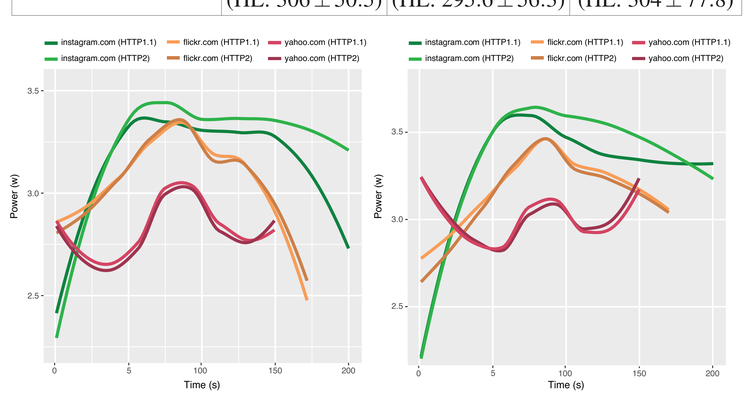

Figure 2: Power draw of navigating the target websites over HTTP1.1 and HTTP2 in low (left) and high (right) latency conditions.

Conclusions

Only slight differences were observed between HTTP1.1 and HTTP2 energy consumption. Moreover, the better energy efficiency of HTTP2 fits within the error margin. As seen in **Table 2**, HTTP2 does however provide better energy utility (better bit rates with similar energy consumption). Though, this is not the case with Instagram, which exhibits longer response times and higher energy consumption over HTTP2. Observing the network waterfall of Instagram (omitted for brevity) it appears that image assets are stuck in *Receiving* state longer and more often when requested over HTTP2. A more detailed explanation for this phenomenon will be provided in the final report.