Web Servers Energy Efficiency Under HTTP/2

Varun Sapra[1] **and Abram Hindle**[1]

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

ABSTRACT

Server energy consumption has been a subject of research for more than a decade now. With Internet scaling rapidly all over the world, more servers are being added continuously. With global warming and financial cost associated with running servers, it has now become a more pressing concern to optimize the power consumption of these servers while still not affecting the performance. The optimization that can be carried out at the hardware level has its limits and therefore the onus comes on to the software developers as well to optimize their web interacting services and use protocols that are more efficient.
Recently, Internet Engineering Task Force (IETF) formalized the specification for the successor of HTTP/1.1 protocol. Named HTTP/2, it has been projected to overcome all the limitations of HTTP/1.1 protocol for which web services developers have to optimize their applications. Understandably, HTTP/2 has been drawing a lot of interest from users, web administrators to big organizations. With HTTP/2 as the future of the Internet communication and servers acting as the backbone of the Internet, we are interested in knowing if HTTP/2 will provide energy efficiency benefits to servers or it will just improve users web experience.
In this paper, we evaluate the energy efficiency of two web servers while they communicate over HTTP/1.1 and HTTP/2 protocol. We also investigate how Transport layer security (TLS) affects the power consumption of the servers. In our tests, we have introduced HTTP/2 features one by one so that readers can see for themselves what benefits the HTTP/2 over HTTP/1.1. Our study suggests that multiplexing and Round Trip time (RTT) are the biggest factors helping HTTP/2 achieve its design goals. We conclude that even with huge TLS associated cost with HTTP/2, on high latency networks it can help servers to be more energy efficient while improving their performance as well.

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

1 INTRODUCTION

The world wide web paved the way for an exponential growth of Internet and Communication Technologies (ICT). In the United States, as of 2013, 74.4 percent of all households have access to The Internet and of these, 73.4 percent reported using a high-speed Internet connection [21]. A Similar trend has been observed in Great Britain where households with Internet access has increased from 57% in 2006 to 86% in 2015 [45]. These increasing number of users with faster internet access have shifted the focus of software vendors to provide their services through the web instead of the desktop based applications. It has led to the growth of data centers and Content Distribution Networks (CDNs) hosting clusters of tens to several thousand interconnected servers [9] providing web- based services to the users.

Data centers have become the backbone of today's ICT infrastructure. However, energy consumption at these data centers has become a subject of concern. According to Brown et al. [12], about 61 billion kilowatt-hours (kWh) in 2006 was used by servers and data centers in the USA for a total cost of about $4.5 billion. It is almost double the amount of energy consumed for similar purposes in 2000. Between 2005 to 2010, the electricity consumption at data centers throughout the world increased by about 56% accounting for 1.5% of the electricity used globally [40]. Most of the world's energy produced by burning fossil fuels [23], a non-renewable source of energy, and also the amount of Green House Gases produced by them contributes significantly towards global warming and climate change phenomenon. Such high costs of operations and environmental concerns has raised the need for reduction in energy utilization at these data centers.

It has been estimated that 40% of the power consumption at the data centers is consumed by the

servers [12] Several measures such as careful air flow handling, elevated temperatures, free cooling, better power system architectures have been designed to improve Power Usage Efficiency (PUE) of these servers and bring down the costs of operating a data center [9]. However, with increasing internet traffic and requirement to add more servers, there is a limit to levels of efficiency that can be achieved by such hardware based measures. It now necessitates designing measures that can improve the energy efficiency of interaction between servers and their users over the Internet.

The web initially meant to serve a collection of hyperlinked textual information, has since transformed to serve multimedia content and provide access to services like emails, social networks, business applications, search engines over the Internet. Loading content rich and complex web pages have become more resource intensive [14]. It highlighted the inefficiencies of HTTP/1.1 protocol which has been serving the web for more than 15 years. Render a web page may now require fetching more than 100 resources from the server [51]. However, the HTTP/1.1 protocol making one request at a time over a TCP connection and suffering from Head of Line Blocking problems is becoming a limiting factor for today's web. To overcome these limitations, HTTP/2 - a protocol based on the implementation of SPDY by Google [54] and a major revision over HTTP/1.1 has been published by the Internet Engineering Steering Group (IESG) in February 2015 [51]. HTTP/2 with features like one TCP connection per domain, Stream multiplexing and Server Push is expected to replace HTTP/1.1 as the preferred protocol for the web.

One of the primary design goals of HTTP/2 is to reduce page load time (PLT) for the end user. To achieve this, HTTP/2 optimizes the parallel HTTP requests by bundling them into a single TCP connection. Moreover, in a shift from traditional request-response mechanisms of HTTP/1.1, HTTP/2 allows servers to send resources to the clients without requiring the clients to explicitly request those resources. HTTP/2 optimizations have been shown to save energy for the mobile client side devices [56]. In our previous study [14], we also found client-side HTTP/2 to be more energy efficient than HTTP/1.1 with or without Transport Layer Security (TLS) applied. However, does these optimizations and reduction in PLT helps HTTP/2 speaking servers to be more energy efficient than HTTP/1.1? Or HTTP/2 servers consume more energy to achieve the designed goals of reduced PLT?

In this paper, we empirically study and compare the energy efficiency of HTTP/1.1 and HTTP/2 web servers. We employ a hardware-based energy measurement system: The Green Miner [34]. In summary, our findings/contributions can be summarized as follows:

1. Using HTTP/1.1 with TLS is most energy expensive for the servers compared to plain HTTP/1.1 and HTTP/2.
2. HTTP/2 multiplexing helps servers save energy and allow faster processing of requests.
3. For high RTTs, servers communicating over HTTP/2 connections use less energy.

The remainder of this paper have been organized as follows: in section II we provide a background for the technologies and terminology used throughout this article. Section III describes the methodology developed for all our tests. Section IV discusses the experiments and provides an analysis of their results. In Section V, we present conclusions we arrived at from our test results. Section VI discusses validatory aspects of our approach. In Section VII, we discuss past research works that relate to our work. Finally, we conclude this paper by briefing the reader about the useful results that he can take from our work.

2 BACKGROUND

In this section, we review the evolution of HTTP protocol and what factors acted as motivation to come up with HTTP/2. We also review some of the terms frequently used in software energy consumption research.

2.1 Hyper Text Transfer Protocol (HTTP) and Its Limitations

Hyper Text Transfer Protocol (HTTP) proposed by Tim Berner Lee in 1989 and officially documented as HTTP v0.9 in 1991 has revolutionized the modern world communication [57]. A simple sequential text-based request-response protocol, it has changed over the years to adjust to the growing demands and in 1997, IETF published the official specification for HTTP/1.1 [29]. HTTP/1.1 added a lot many new features to the original HTTP - persistent connections, chunked transfer encoding, pipelining, new caching mechanisms, session cookies, etc. Users can now request not only text-based HTML files but also multimedia resources like images, video files.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

However, the web has been growing at an exceedingly fast pace. According to HTTP Archive [35], as of April 2016, most of the websites today on Internet are composed of HTML, CSS, Java Scripts, images, videos and other contents, making the size of an average web page to be about 2.24 MB. For the top 300K web pages in the world, it can now take 40 TCP connections to make 100 requests over 18 different domains to completely fetch and render a web page in the browser [51], [35]. Although HTTP/1.1 added new features to handle such websites, those features faced their limitations when adopted on the web. Pipelining being one such widely known never used in practice feature as it can lead to Head of Line Blocking problems thereby completely stalling the connection. However, users with access to faster Internet connection started to look for real-time responsiveness of the websites. As intermediate measures web developers were coming up with their own these optimizations like, domain sharding (splitting your website are awkward resources across domains to allow more TCP connections), inlining (embedding resources into the parent document), spriting (combining multiple images into a single image). These techniques, however, means more work for the developers and doesn't help much as the underlying problem is HTTP/1.1 was not able to utilize a TCP connection to its full potential [51].

2.2 SPDY and HTTP/2

Google felt the degrading performance of web services due to the protocol limitations and in 2009, they announced a new experimental protocol called SPDY [10]. SPDY introduced a new binary framing layer on top of TCP connection to achieve full bi-directional request and response multiplexing, requests prioritization and header compression. These features helped SPDY to meet its design goal of reducing page load time by 50% [26]. The success of SPDY pushed IETF, to bring an official new standard for the web. Building upon the most useful features of SPDY and adding new features on top of it, IESG announced the official HTTP/2 standard in early 2015 [44]. Unlike SPDY, HTTP/2 did not make TLS mandatory. Also, HTTP/2 uses a more efficient header compression algorithm called HPACK. The support for Next Protocol Negotiation (NPN) from SPDY has been discontinued, and Application Layer Protocol Negotiation (ALPN) has replaced it in HTTP/2.

2.3 Server Push

Server Push feature was introduced in SPDY and has been adopted by HTTP/2. It allows a server to send preemptively resources to the client based on a request received previously from the client. It relies on the assumption that once a client has requested a particular resource; it will then need a next few set of resources to render that resource completely. When a server receives a request for that particular resource, it sends PUSH_PROMISE frames along with the response indicating to the client the response streams that will be initiated by the server. It helps save the request time for those pushed resources and thereby reducing the page load time. For example, on receiving the request for HTML home page of a website, a server can initiate the push streams for necessary CSS, JS resources required to render that web page.

We choose to describe this feature in more detail here as we have Server Push enabled for all our experiments with HTTP/2.

2.4 Power and Energy

In this paper, we focus on change in energy consumption behavior of a server while serving all the requested resources over an HTTP/1.1 and HTTP/2 connection respectively. Power is the rate of doing work or the rate of using energy; energy is defined as the capacity of doing work [5]. For our use case, the amount of total energy consumed by the server in serving all the requesting clients is its energy consumption and energy consumed per second for that period is its power usage. The measurement unit for power is watts while joules are used to measure energy. A server consuming 4 watts of power for 10 seconds in serving the clients, consumes 40 joules of energy for that period. In all our tests, the results have been presented in joules of energy consumed by the server.

3 METHODOLOGY

In this section, we describe the energy measurement testbed, web servers, and clients deployed for our study. We then describe the optimizations one may need to perform to carry out load tests and also discuss the validity of the data collected through the tests.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

3.1 Green Miner

To capture energy consumption profiles of the web server for HTTP/2 and HTTP/1.1, we have used Green Miner [34] as our test bed. Green Miner is a continuous testing framework that allows you to capture the power performance data for your software applications. The Green Miner setup for this work involved the following components: a Raspberry Pi model B computer for monitoring tests, executing tests, collecting data and uploading test data; a USB hub; an Arduino Uno and an Adafruit INA219 current sensor for capturing energy consumption; and another Raspberry Pi Model B computer on which the servers under test were deployed. The two Raspberry Pis' are directly connected to each other via a 100 Mbit category 5 Ethernet cable. Some tests were also carried out by replacing the test monitoring Raspberry Pi with an Intel Core i5-4200M 2.5 GHz, Lenovo Thinkpad L540 laptop having 8 GB of RAM and running Ubuntu 14.04 LTS. To run the tests, we then deployed the Green Miner's software framework on the laptop as well.

The power supply provided by YIHUA 305D Lab Grade generates a constant voltage of 4.1V. It is used to power both the Raspberry Pis'. The monitoring Pi/laptop initiates the tests by transferring test scripts and necessary applications to the Pi under test. It then issues test commands through Secure Shell(SSH). The Arduino Uno collects the measurements of voltage and amperage used by the Pi under test and reports it to the monitoring Raspberry Pi/laptop through a USB serial connection. It then processes, annotate and package this data into a test report to be uploaded onto the web server. Interested readers can refer to the works published by Green Miner authors [34], [49].

3.2 Writing a Test Script

A test script is required on Pi under test to start the server, make it wait enough to receive resources requests from the clients running on monitoring Pi/laptop and once the requests are done, to shut down the server. The HTTP clients are started on the monitoring PI once the server has started. It required identifying the time taken by a server to start and wait for CPU usage spike to go back to idle state. This is done to make sure that our energy measurements for serving resources by the server are not affected by the CPU, which may have gone into High power state due to the start of the server. It has been referred to as the tail power phenomenon by the researchers in the past [8], [48], [47].

3.3 Deploying and Configuring HTTP/2 Servers

HTTP/2 being formalized recently in February 2015, the several implementations available for HTTP/2 servers [2] doesn't support all the features yet. We specifically looked for ones that have support for HTTP/1.1 as well as server push feature of HTTP/2 as it was one of the primary requirements for our test cases. We finally decided to deploy and experiment with H2O version 1.8.0 and Jetty version 9.3.8v20160115 servers [46], [1].

H2O server is among one of the most mature implementations of HTTP/2 servers available. In some of the benchmark studies, its performance has been found to be much faster than Nginx [46]. It implements latest draft specification of HTTP/2 including features like dependency based prioritization, server push, client cache aware push mechanism and reprioritization. H2O allows the user to specify the resources one would like to push through mruby [20] scripting in its configuration file. It is also possible to verify that the resources got pushed or not through server access logs.

Jetty is an open source HTTP server and servlet container. It was among the first ones to implement HTTP/2 support and has a small memory footprint leading to better performance. Jetty's implementation of HTTP/2 require JDK 8 and provides an Application Layer Protocol Negotiation (ALPN) extension for protocol negotiation at TLS layer. The ALPN extension should match the version of JDK 8 and is placed in the Jetty's class path. Jetty defines its own strategy to support server push through its PushCacheFilter class by analyzing the incoming requests from the client and dynamically trying to predict the other resources that may be required by the client in subsequent requests. It maintains a fingerprint of the client requests to implement this strategy. Web developers are only required to tell jetty to enable server push feature for their web applications by specifying the PushFilter in their web.xml files.

However, to be consistent in our experiments across H2O and Jetty we wanted to push the same files for the same request as we were doing for H2O. Jetty being an open source project, we were able to achieve this requirement by extending Jetty's Push strategy to define our own static server push strategy which allowed us to specify the same resources as H2O that should be pushed by the Jetty server when an incoming request comes for a particular resource.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

We also made sure to make the configuration of H2O and Jetty as similar as possible. For this, we increased the H2O idle timeout to 30 seconds as Jetty's. SSL cipher suite was configured to use ECDHE-RSA-AES128-GCM-SHA256 encryption. Both servers are using the same key and the certificate file. The request and the error logs were disabled for both the servers. Jetty server monitors its web application directory at regular intervals to check if any new content is added and can dynamically start serving it. As it can induce extra CPU usage and was not required for our use case, this was disabled as well.

3.4 Deploying HTTP/1.1 and HTTP/2 clients

To make the requests to the servers, we once again looked through the available HTTP/2 implementations for an HTTP client application. Nghttp2 project [53] provides an implementation of HTTP/2 as a reusable C library and has three subprojects - nghttp: an HTTP/2 client, nghttpx: HTTP/2 server and proxy and h2load: an HTTP/2 benchmarking tool. We are interested in h2load as it allows you to emulate both HTTP/1.1 and HTTP/2 clients. However, the current implementation of h2load doesn't accept resources pushed by a server. We modified the h2load source code and recompiled it to enable push support in h2load. Also, h2load only requests and receive the resources from the server but does not render the content received in response.

In our tests, each HTTP/1.1 client makes 6 TCP connections to the server. An HTTP/2 client by IETF specification fetches all the resources from a single domain over one TCP connection. For all HTTP/1.1 tests, Pipelining was kept disabled as this feature is still not used in practice and in most browsers it is disabled by default [51]. We also verified the common headers sent in a request to the H2O and Jetty servers by Mozilla Firefox version 45.0 and added those headers to our requests as well to make our client requests as close to as the ones made by a browser. h2load tells you the amount of time it took to fetch all the resources from the server and the total load size that has been fetched. As we disabled the access request logs on the server side, the load size fetched helped us to verify that apart from the requested resources all the resources that have been pushed by the server has also been successfully received. We confirmed the fetched load size for every test to determine whether a test has been successful or not. Those tests in which the load size came out to be smaller than expected may be due to request failures, or TCP connection failures were considered failed, and their results were discarded.

In the case of HTTP/2, for ALPN negotiation at TLS layer, it is required that both the client and the server machines have OpenSSL version 1.0.2 or greater installed. We are using OpenSSL version 1.0.2f.

3.5 TCP and Kernel parameters tuning

In the tests, we want to put our servers under heavy load and see how it affects their energy consumption behavior. There is useful advice available to tune the operating system parameters if one is planning to do a load test [3], [42], [28] to avoid running into any limitations imposed by the platform. Following the advice, we made sure that TCP window scaling is enabled and increased the TCP receive and send window buffer sizes. Connection listening queue size and incoming packet queue sizes were also increased. We also increased the port range such that at least 64k ports are available for client connections. To make sure, that the client and server machines do not run out of the available file descriptors, their values were set to be much higher than the default. The TCP congestion control algorithm was set to CUBIC for its high performance [30] compared to other algorithms. These changes were made at the client as well as at the server end.

3.6 Workload

Our objective is to emulate the communication between a client and a server as close to real-world scenarios as possible. Recent observations suggest that multimedia rich websites of today on average may require about 2.2 MB of data to be downloaded to render completely in a web browser [8]. It may involve fetching hundreds of objects from the web server. Research done in the past suggest that the number of objects fetched from a web server can play a vital role in the performance of SPDY [54] - a predecessor and driver for HTTP/2 specifications. Therefore, we made sure that our test load size and the number of resources fetched can represent a real-world website load.

Following the work done previously [14], we once again used FGallery [17], a static photo gallery generator and deployed it onto the H2O and the Jetty server. The Fgallery resources deployed consists of the home HTML page, CSS, JS, JSON and Image files. We configured the number of resources that should be requested by a client. There has been no formal specification on which files should be pushed

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

Table I. DESCRIPTION OF THE WORKLOAD

Resource Type	Resource Count	Resource Size (in KB)	Push Candidate (HTTP/2)
HTML	1	0.728	No
CSS	21	27	Yes
JS	27	201.10	Yes
PNG	60	11.9	Yes
JPEG	132	2385.36	No
Total:	**240**	**2626.08**	

by the server [51] and different implementations either have their own strategies to decide push candidate resources or they leave it to the user to decide [46]. For our experiments, we chose the resource files necessary to render the Fgallery web page as candidate resources that should be pushed by the server when it receives a request for the home HTML file.

We configured the total load to be fetched by an HTTP/1.1 and an HTTP/2 client to 240 resources constituting about 2.56MB in size. Of these 240 resources, we chose 108 to be pushed when an HTTP/2 client request the server for index/Home HTML page of the website. These pushed resources constitute 240KB in size which is approximately 10% of the total load to be fetched. In contrast to 240 requests made by an HTTP/1.1 client, an HTTP/2 client will have to request only 132 resources and rest of the resources will be pushed by the server as soon as the HTML resource is requested.

Table I presents the summarized details of workload deployed.

3.7 Data Collection and its Validity

In previous research with Green Miner [14], [5] it has been observed that a single measurement for a particular setting could give some false positive results as there could be variation in the measurements due to the factors which are not under control and are unrelated to the application under test. Therefore, it was suggested that taking an average of at least ten runs for each test will produce more accurate test results. For our tests, we initially started with 15 tests for each of the test scenario and later on reduced it to 10 each as we found that our average measurements are not getting affected by decreasing the tests repetitions to 10.

Green Miner enables us to collect energy consumption for different phases of our tests. For example, in our tests different phases of a test included starting the Green Miner measurement service, launch the server, wait for the server to receive requests from the clients, stop the server. Chowdhury et al. [14] found that a previous phase of the test may affect the energy measurements of the task we are interested in. This may happen because of the different sequence of operations in the previous phase or the different amount of time taken by the previous phase to perform its task which may leave the system components in a High power state. Although their findings were on Android devices, we followed their advice to apply a sleep period before our main task gets started. It helped as the time taken by the Jetty server to start and then for the system under test to go back to idle state was found to be considerably high in comparison to H2O which is a very lightweight server. We, therefore, imposed an idle period of about one minute to be followed, once the server has started. It allowed the test Raspberry Pi (system under test) to go back to the idle state before the server starts receiving the requests.

Green Miner reports 50 power readings for every second in its test report. In our test designs, we logged the time at which the clients start sending the requests to the server and the time at which all the requested resources have been received. We use these logged times to take the power readings from the Green Miner's test reports to calculate the amount of energy consumed by Pi under test. We report our results in terms of energy consumed as we noticed that power consumption on the test machine at any instant of time when the server is receiving requests from the clients remain almost constant varying at maximum in between 2.2 Watts to 2.4 Watts. The difference in energy consumption comes from the amount of time for which the test system remains under this high power state which is the time taken by the server to respond to all the requests. This time varies depending on the number of clients interacting with the server and whether the protocol used to communicate is HTTP/1.1 or HTTP/2.

Here, we would also like to reiterate that server access and error logs were disabled on both the servers. For each test, we verify its successful completion from the size of the load fetched by h2load and

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

successful completion of all the requests. As one client fetches 2.56 MB of data from the server, and if ten clients are requesting FGallery resources from the server the data fetched should be about 25.6 MB. Otherwise, the test is considered unsuccessful and its data is discarded. However, while writing the tests we ran each test individually as well and also kept the server logs enabled. The motive was to verify the correct behavior of request-response mechanism at both the server and the client ends.

4 EXPERIMENTS CATEGORIZATION AND RESULT ANALYSIS

For an easy understanding of the reader, we divide our experiments to measure server energy efficiency into the following three sets based upon test conditions and protocols :

1. Energy performance evaluation under HTTP/1.1 vs. HTTP/1.1 with TLS vs. HTTP/2.
2. Energy consumption behavior under HTTP/2 with streams multiplexing enabled.
3. Server energy consumption under HTTP/1.1 and HTTP/2 with varying latencies over the network.

4.1 Servers' energy performance evaluation under HTTP/1.1 vs HTTP/1.1 with TLS vs. HTTP/2 communication

Figure 1 shows the energy performance of H2O server under HTTP/1.1 vs. HTTPS vs. HTTP/2 protocols. The clients, in this case, were running on monitoring Raspberry Pi. From the figure, it can be clearly seen that server energy consumption for all three protocols is increasing as the number of clients are increasing. We checked the power consumption, and it turned out at any given moment while serving the requests the server power consumption is almost constant irrespective of the number of the clients. Instead, it is the total time taken which is increasing as the number of clients increases. That's why the relationship of energy consumption for the number of clients approximately comes out to be linear in all our tests. For one client test, the difference in energy consumption under different protocols although present but is hard to identify due to the scale of y-axis in the graph.

We conducted this evaluation for H2O server from the monitoring laptop as well. Figure 2 shows the H2O energy consumption under different protocols in case of h2load clients running from the laptop. The behavior of H2O energy consumption under different protocols was found to be similar as above. However, note that the energy consumed by H2O server when the client end is our laptop is much lower than energy consumed while running the clients on Raspberry Pi. This difference has been shown in Figure 3.

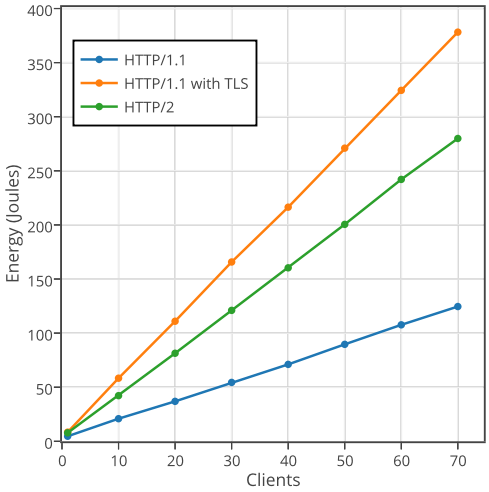

Figure 1. Energy performance of H2O server. Client - RPI

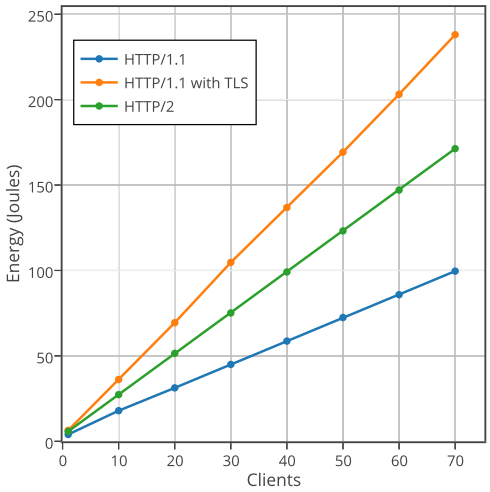

Figure 2. Energy performance of H2O server. Client - Laptop

The difference is due to the difference in hardware configurations of our laptop, a 4-core machine and Raspberry Pi, a single core machine. The hardware configuration differences enable h2load to generate all the clients on the laptop and establish TCP connections with the server at a much faster pace than it can

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

do on Raspberry Pi. Due to these differences the time it takes for all the clients to receive the requested responses and end their processing is reduced by a significant margin when they are running on the laptop. To further confirm the statistical significance of differences in energy consumption results on Raspberry Pi and the laptop we calculated P-values for all the test cases in case of 10 clients. All of the P-values came out to be very low ($\ll 0.05$) and has been summarized in Table II.

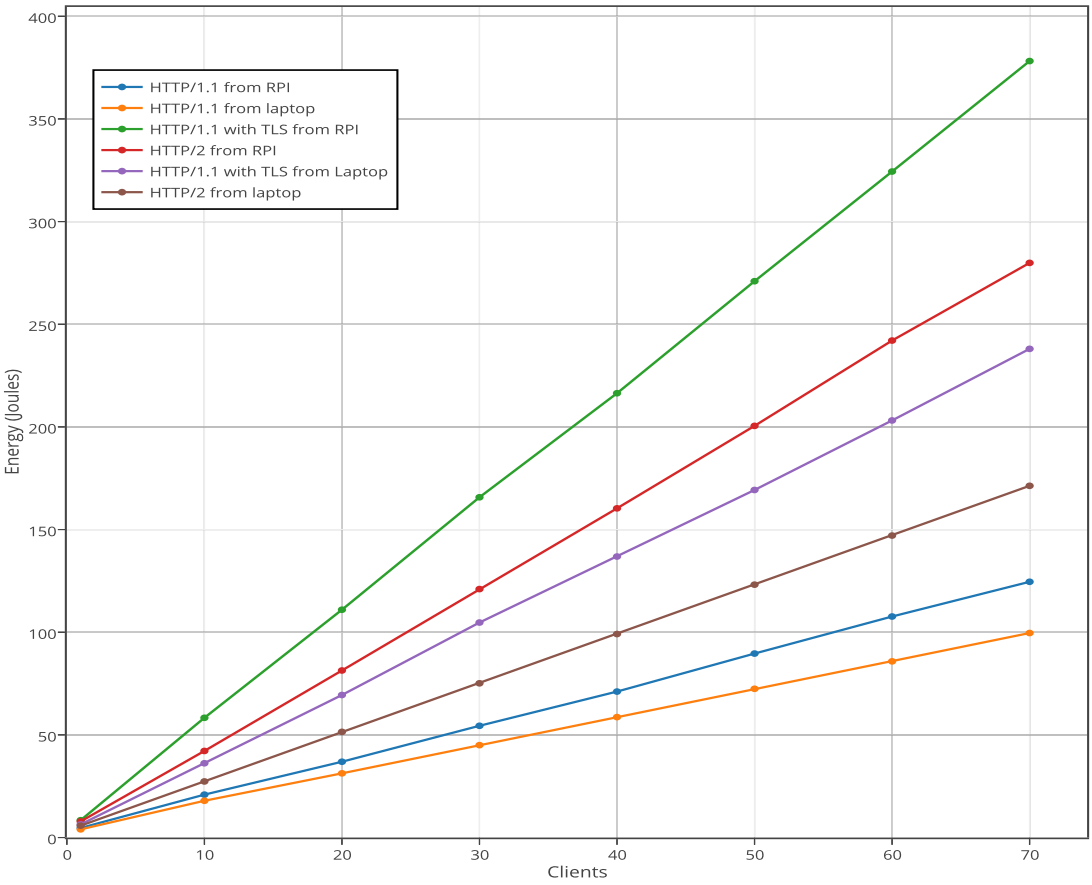

Figure 3. Overall Comparison of Energy performance of H2O server. Clients – RPI vs Laptop

Table II. P-VALUE FOR PAIRED T-TEST AMONG DIFFERENT SETTINGS BETWEEN RPI AND LAPTOP (FOR 10 CLIENTS)

	HTTP/1.1 From RPI	HTTP/1.1 with TLS from RPI	HTTP/2 from RPI	HTTP/1.1 from Laptop	HTTP/1.1 with TLS from Laptop	HTTP/2 from Laptop
HTTP/1.1 From RPI	1	<<.00001	<<.00001	<<.00001	<<.00001	<<.00001
HTTP/1.1 with TLS from RPI	<<.00001	1	<<.00001	<<.00001	<<.00001	<<.00001
HTTP/2 from RPI	<<.00001	<<.00001	1	<<.00001	<<.00001	<<.00001
HTTP/1.1 from Laptop	<<.00001	<<.00001	<<.00001	1	<<.00001	<<.00001
HTTP/1.1 with TLS from Laptop	<<.00001	<<.00001	<<.00001	<<.00001	1	<<.00001
HTTP/2 from Laptop	<<.00001	<<.00001	<<.00001	<<.00001	<<.00001	1

Also, note that the energy performance of the H2O server is worst under HTTP/1.1 with TLS for all the client values. This can be due to an extra handshake introduced at the TLS layer that the connection has to go through for it to be established. It adds an extra RTT on top of HTTP/1.1 thereby increasing latency and specifically the finish time in our case. Moreover, both the client and the server has to perform

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

extra work for encryption/decryption of HTTP messages. This keeps the CPU in continuous usage for the duration of the communication. Interestingly, our observation is very well supported by the research done in the past by Naylor et. Al [43]. However, it also contradicts the results published by Goldberg et al. [24] and Grigorik [27]. It should be noted that those contradictory studies performed optimization for TLS communications before doing any comparison. In our tests, we did not do any such optimization at TLS layer and rely on default implementation available.

One may ask that TLS layer is involved in communication over HTTP/2 protocol as well but still HTTP/2 has better performance than HTTP/1.1 with TLS. It is because HTTP/2 makes only one TCP connection between the server and the client. Therefore, it has to do TLS handshake only once. It means one extra round trip for HTTP/2. However, an HTTP/1.1 client communicating over TLS make approximately six parallel TCP connections between the client and the server. That means six extra round trips happen to perform 6 TLS handshakes over those TCP connections.

We also performed the above set of experiments for the Jetty Server while running the clients on Raspberry Pi and the laptop. Figure 4, 5 and 6 shows the Jetty's energy performance while communicating with clients on Raspberry Pi, laptop and overall respectively. All the observations made for H2O server also apply to Jetty server as well. However, note that Jetty requires more system resources for requests processing as compared to H2O server. It initiates a new thread on the test Pi for every connection. Our test Raspberry Pi had limited capabilities (Single core, limited Java Heap Space) and therefore we were able to scale the Jetty server for only up to 40 clients. While trying to do the tests for more than 40 clients, for some of the clients Jetty took so long to serve the requests that connections started to get dropped due to connection idle timeout behavior coming into play.

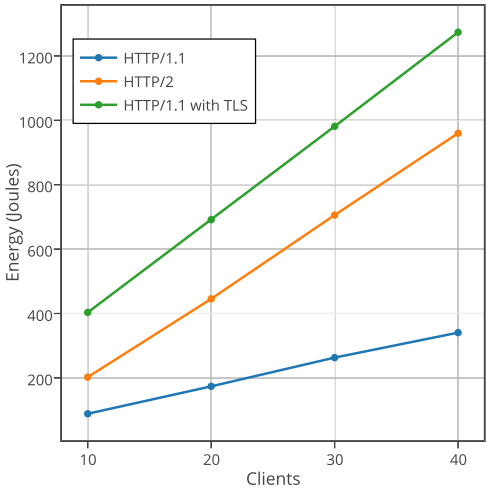

Figure 4. Energy performance of Jetty server. Client -RPI

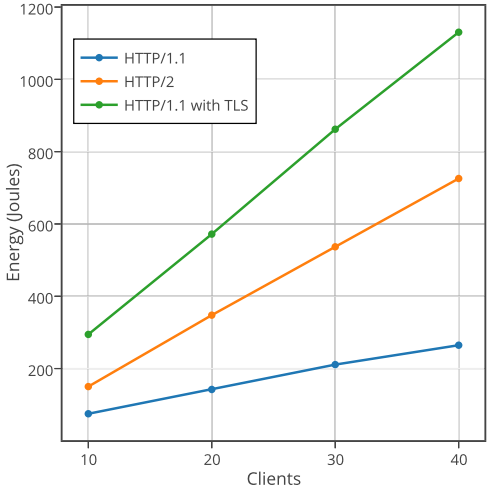

Figure 5. Energy performance of Jetty server. Client - Laptop

4.2 Energy consumption behavior under HTTP/2 with streams multiplexing enabled

In the previous set of experiments, we did not use HTTP/2's multiplexing capabilities as we wanted to investigate if only one TCP connection can help an HTTP/2 communicating server achieve better energy performance. However, this being not the observed case, we decided to enable multiplexing of HTTP/2 streams. HTTP/2 implemented a binary framing layer to enable bidirectional flow of interleaved request response frames between the client and the server. It allows HTTP/2 to achieve a truly multiplexed communication over a single TCP connection without running into the problems of Head of Line blocking faced with HTTP/1.1.

For this part, we conducted our experiments using H2O server on Raspberry Pi under test and h2load's HTTP/1.1 and HTTP/2 clients on both the laptop as well as the Raspberry Pi. We did not test with Jetty Server because assembling dissembling of frames in multiplexing would have required more CPU work at both the client and the server ends. It will result in for us that instead of scaling the number of clients we would have to reduce them as more connection failure would have occurred due to idle timeout.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

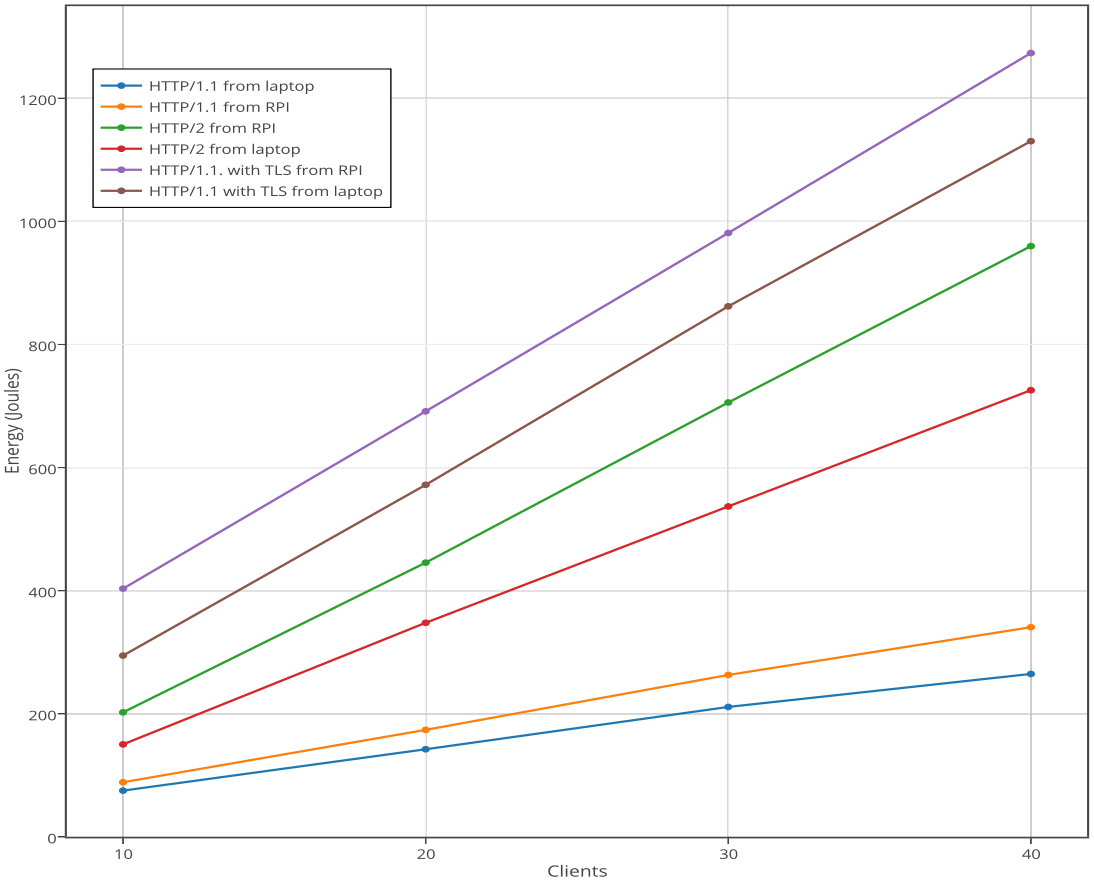

Figure 6. Overall Comparison of Energy performance of Jetty server. Clients – RPI vs Laptop

The currently available implementations of HTTP/2 clients start a new stream for every request. However, the number of concurrent streams that can be communicating at any point in time is initially decided between the client and the server when the connection is established for the first time. H2O server has hard coded this limit to 100 concurrent streams at maximum. By HTTP/2 specifications [44], a client has to agree to the server communicated settings to establish a successful connection. Therefore, we cannot go beyond 100 concurrent streams at once. For the experiments, we choose to test HTTP/2 energy behavior for 1, 10, 50 and 100 concurrent streams.

Figure 7 and Figure 8 shows the energy consumption behavior of H2O server while interacting with a different number of HTTP/2 streams from clients on Raspberry Pi and the laptop respectively. It can be seen that as the number of HTTP/2 streams are being increased from 1 to 100, the energy consumption of the server has reduced. It is because we are now doing request in parallel as well as responses are being received in parallel. It led to all the requests by the clients to be finished earlier as compared to any HTTP/2 channel with a lesser number of concurrent streams. However, even with 100 streams enabled the energy consumption by the server over an HTTP/2 channel is much higher compared to HTTP/1.1. It means that cost of encrypted communication on an HTTP/2 channel is much greater compared to the concurrency benefits achieved and also in comparison to a plain HTTP/1.1 connection. Also, the energy consumption behavior is still found to be linear with the increase in the number of clients irrespective of the number of streams. A slight deviation from linear behavior between 30 – 50 clients in figure 7 and between 50 – 70 clients in figure 8 for HTTP/2 connection with 50 streams could be due to the random behavior of the system either at the client end, or the server end or both ends thereby affecting the data collected.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

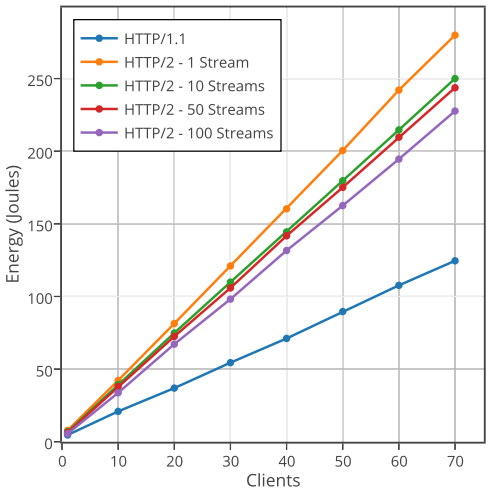

Figure 7. H2O energy Performance with HTTP/2 multiplexing. Clients on RPI.

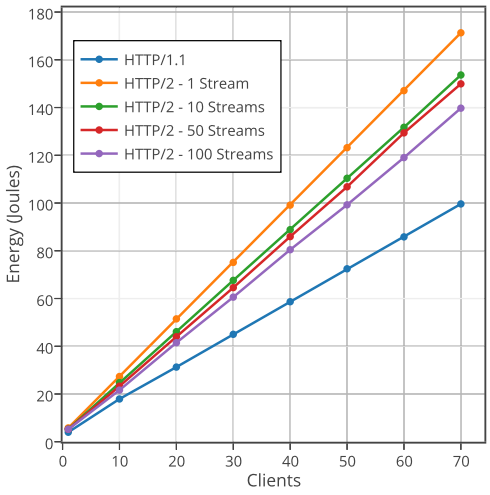

Figure 8. H2O energy Performance with HTTP/2 multiplexing. Clients on laptop.

4.3 Server energy consumption under HTTP/1.1 and HTTP/2 with varying latencies over the network

The earlier two sets of experiments (Part A and Part B) discussed above were performed with network latency between the client and the server approximately close to 0ms. In 2010, Mike Belshe published a study [11], revealing how in real world communications it is the latency that affects the page load time. This study acted as a driver for SPDY whose design goal was to reduce latency and its features were adopted by HTTP/2 in future. Also, in our previous study [14], HTTP/2 showed its real benefits on high RTT links. We, therefore, decided to introduce artificial latency on the network link between our client and the server. One can add artificial latency by manipulating traffic control settings [37] on a Linux based system. Following [14], [22] we introduced artificial latencies at the values of 30ms, 200ms, and 1000ms. This set of experiments now completely utilize HTTP/2 potential as we have maximum possible concurrent streams enabled, and latency added to the network as well. Specifically, the scenarios needed by HTTP/2 to show its benefits [28]. These set of experiments were also conducted only for H2O server and h2load's HTTP/1.1 and HTTP/2 clients on the laptop.

Figure 9 shows the energy performance of H2O server for HTTP/1.1 and HTTP/2 at 30ms latency between the network. When there is only one client, H2O server shows a little better energy efficiency under HTTP/2 as compared to HTTP/1.1. However, as we started to increase the number of the clients the server energy performance with HTTP/1.1 got better. From our past results [14], we did not expect any improvement for HTTP/2 at 30ms latency so we can say it is still better.

We then inspected the energy performance behavior at 200ms latency. The results are shown in figure 10. Surprisingly, up till about ten clients, energy performance of H2O is much better for HTTP/2 as compared to HTTP/1.1. However, just beyond the ten clients the trend reverses and HTTP/1.1 start showing much better performance. From our logs, we found that as the number of HTTP/1.1 clients are increasing the overall throughput achieved is also increasing for them thereby reducing the time required to finish all the requests. For example, with 10 clients, the response time per client on average is 1.13s (total response time being 11.36s). With 30 clients, the response time per client on average is reduced to about 0.82 seconds and similarly with 50 clients, the time required per client has decreased to 0.64s. We then dug deeper into the h2load source code to investigate out how HTTP/1.1 clients are able to have better throughout. It turns out, h2load calculates the number of HTTP/1.1 clients conceptually from the number of connections that are being made in multiples of 6. Every 6 connections are considered as 1 client. So, as we go beyond 10 clients, for example, say 30 clients, 180 TCP connections are being made in the background to fetch all of the requested resources. The level of concurrency achieved by these parallel 180 TCP connection is much higher than what parallel 30 HTTP/2 TCP connections can achieve, and therefore more the number of parallel HTTP/1.1 connections lesser is the effect of latency on them. In our experiments, time to finish all the requests has played a bigger role than the number

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

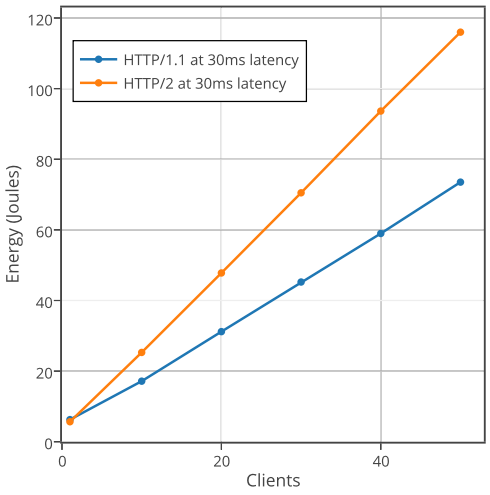

Figure 9. H2O energy efficiency at 30ms latency for HTTP/1.1 and HTTP/2 communication. Client: laptop

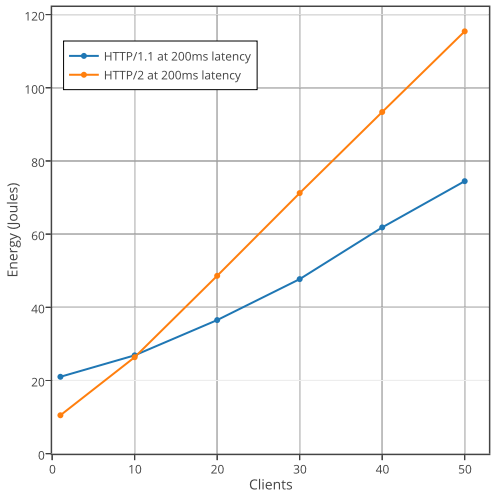

Figure 10. H2O energy efficiency at 200ms latency for HTTP/1.1 and HTTP/2 communication. Client: laptop

of sockets being used by the connections at the client end or the server end. The performance effects of parallel TCP connections have been described in details in the research works done in past by many researchers [31], [32], [6].

Figure 11, shows the energy performance at 1000ms latency. At such high latency, HTTP/2 is apparently performing better than HTTP/1.1 up till 50 clients. However, considering that energy efficiency under HTTP/2 and HTTP/1.1 is growing linearly, and slope of HTTP/2 is much greater than for HTTP/1.1, the two lines are bound to intersect at some point in time for some number of clients. It is similar to the behavior observed for 200ms latency and resulted from how HTTP/1.1 connections are made and number of clients calculated by h2load; instead of following from the differences in performance for HTTP/2 and HTTP/1.1.

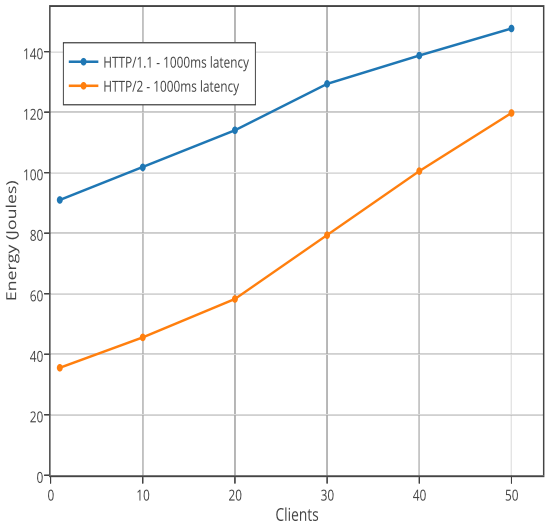

Figure 11. H2O energy efficiency at 1000ms latency for HTTP/1.1 and HTTP/2 communication. Client: laptop

5 DISCUSSION

We showed that HTTP/2 used at its full strength and under high latency conditions can help improve the server energy efficiency. The performance observed under HTTP/1.1 for a large number of clients can be

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

a point of contention. However, that is more of an h2load issue and how it chooses to depict multiple HTTP/1.1 clients' behavior which does not truly represent how an HTTP/1.1 client may work in practice. h2load's HTTP/1.1 implementation allows a very high level of parallel connections being achieved under HTTP/1.1, which in practice are limited to a maximum of 6 TCP connections per client.

The rate at which Internet is growing and Exabytes of information are being added to it annually; it calls for more servers to serve the web and the data centers managing them. Server energy has environmental as well as financial costs associated with it. Therefore, HTTP/2 apart from making the web faster, helping in making servers more energy efficient for the same workload is an added advantage.

What makes HTTP/2 more energy efficient and impervious to higher latencies ?

In HTTP/1.1, to achieve request parallelism a client opens approximately 6 TCP connections to a domain. However, the request - response mechanism over these 6 TCP connections is still sequential as pipelining is not used in practice because it brings a set of issues like Head of Line Blocking along with it. These sequential requests do not achieve true parallelism and on high latency networks keeps TCP connections active for longer periods of time. It also adds to the time for which computational resources are being kept in active power states at both ends of communication thereby making HTTP/1.1 energy inefficient at high latency networks.

HTTP/2 design goals were able to address these issues by implementing a framing layer which allowed achieving the full request-response multiplexing in a bidirectional communication. It achieved this concurrency over a single TCP connection per domain and therefore makes less TCP connection to load a website. Moreover, the HPACK algorithm enabled HTTP/2 to compress headers in communication [53]. Also, the server push helped save a lot many round trips that would have been required otherwise between the client and the server. All these factors together help to reduce the network time required for communication. It reduces the period for which computation resources are being used actively thereby improving servers to be more energy efficient than before.

We also observed that HTTP/2 implementation is quite robust to changes in the latency and may not affect the energy efficiency of a server much as latency increases. Figure 12 shows this effect at 30ms and 200ms latency values. Note that, for 30ms and 200ms, as the HTTP/2 clients/connections interacting with the server increased the energy performance of the server at these two latency values almost comes to be similar. It confirms that HTTP/2's design makes it impervious to higher latencies.

Organization managing data centers, or having lot many servers should consider switching to HTTP/2 as their application layer protocol. They will be able to see the financial and time-saving benefits if they have a lot many users from remote locations in the world. Their user's web experience will also improve with faster page load times, starting from latency values of as low as 30ms.

6 THREATS TO VALIDITY

With our test load size, we tried to emulate it as close as possible to a real-world web site. Although, 108 files being pushed may appear like a lot many resources has been chosen as push candidates but they make only 10% of the total website load. Also, h2load clients fetch the content and discard it. It doesn't render the fetched content. Real browsers will fetch the content and spend time in rendering it as well. However, we did test that the resources in our Fgallery load get rendered correctly in the browsers of our laptops.

In a real world, a client and a server communicating with each other will have a lot many networking middle-boxes in their path. Also, packet loss is a very common phenomenon on such communication paths. However, we did not observe any packet loss on our local Ethernet connection. Client distributed throughout the world communicating with our deployed server can be a closer representative of the real world behavior.

For our load tests, we tuned the TCP and kernel parameters following published research. However, we suspect that the behavior can vary across machines. Also, our server test Pi being a single core machine cannot achieve true parallelism while serving the requests.

We would also like to point out that the HTTP/2 server and the client implementations are still under development. During our tests, we faced a lot many situations where something did not work correctly because it doesn't follow the official specification. e.g. Jetty Server implemented server push as a recursive push feature instead of pushing the files sequentially as stated in HTTP/2 standard. Similarly, H2O implementation for pushed stream IDs was not correct. For such cases, we resolved the errors either

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

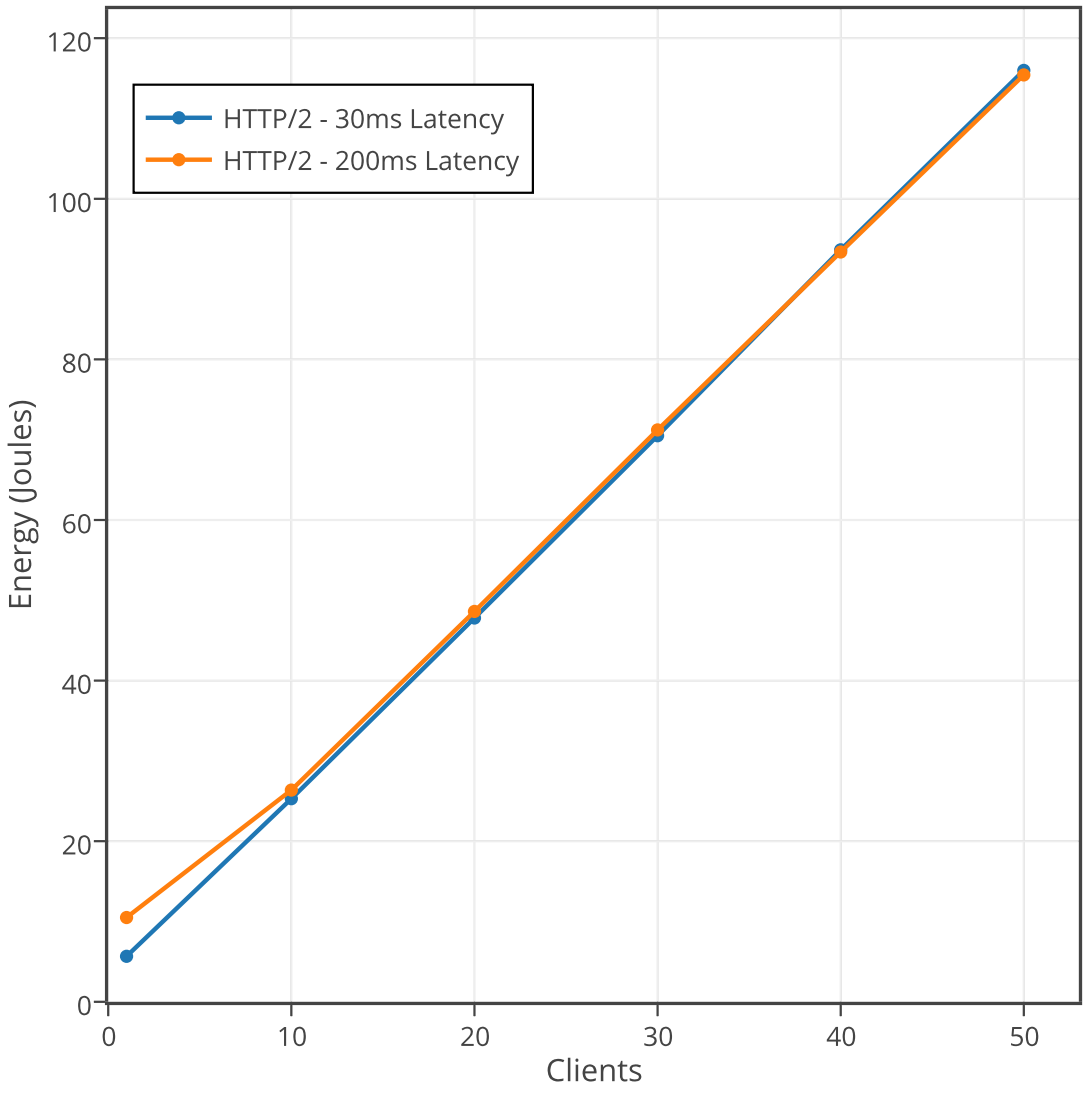

Figure 12. H2O energy efficiency unaffected from increase in latency from 30ms to 200ms for HTTP/2 communication.

by ourselves and in case we were not able to resolve it we discussed the issues with the developers of the respective projects to help get it rectified.

7 RELATED WORKS

A lot of research has been done in the field of Energy optimizations for a data center. Similarly, researchers interested in web protocols have published performance studies of different protocols and have adapted them for their work. We review these works in two separate subsections: 1) Energy optimization for servers and data centers; 2) Performance evaluation of web protocols.

7.1 Energy optimization for servers and datacenters

To characterize the performance of the servers several benchmarks have been developed. SPECpower [15] and JouleSort [50] are two such benchmarks characterizing the efficiency of the servers. SPECpower computes the performance to power ratio for a system running a business class application on a Java enterprise platform. It reports the performance metrics for the server across its whole utilization period, unlike JouleSort which indicates the performance per watt only for the duration of the peak usage of the server.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

Benchmarks have also been developed to measure the efficiency of storage class server systems as well. Emerald Program [7] and SPC-2/E [16] by the Storage Performance Council are two of many such efforts. Both of these benchmarks monitor storage servers for request access and report transactions throughput per watt. Tsirogiannis et al. [52] studies database servers for their energy efficiency. Their work was among the first ones to characterize the power profile of standard database operations like scans, joins, sorts, etc. They concluded that, for database servers, the highest performing profiles happen to be the most energy efficient as well.

For data centers, a benchmark called Power Usage Efficiency (PUE), measuring total power consumption divided by server power consumption, is used to characterize their power utilization efficiency. Unfortunately, most of the datacenters have PUE value of 2 or greater [9]. As a remedy, Meisner et.al. [41] suggested the use of active low power modes, saving energy at a performance cost, to achieve high energy efficiency proportionality. Their work was motivated by a study which concluded that existing low power workloads neither achieve energy efficiency nor low latency in operations.

7.2 Performance evaluation of web protocols

SPDY Studies: In one of the first studies on SPDY [25], researchers at Google evaluated the top 100 websites for their page load time. Their results showed that SPDY can improve page load time of these websites by 27-60% over HTTP and 39-55% where traffic is carried over SSL. As a follow-up, a white paper published by Google [26], suggests SPDY can improve the page load time up to 23% over mobile networks on an Android device.

Contradicting these findings by Google, Erman et al. [19] found that SPDY doesn't outperform HTTP over cellular networks due to poor interaction between SPDY and TCP. On a cellular network, radio resource connection states can continuously vary, and as TCP doesn't account for such variability, this may result in unnecessary re-transmissions. SPDY is more affected by it due to the use of a single connection. Cardaci et al. [13] evaluated SPDY over high latency satellite channels and found it to be performing marginally better than HTTP.

Wang et al. [54] studied different factors which can affect the performance of a connection and found that multiplexing and longer RTTs help SPDY to achieve an improvement over HTTP. However, these benefits are significantly reduced under a high packet loss connection or due to web page dependencies and computation delays involved at both ends of the communication. Elkhatib et al. [18] found that SPDY can decrease as well as increase the performance depending on the network condition and the web page load size. However, they found SPDY connections to be more susceptible to packet loss.

HTTP/2 Studies: Researchers working in the field of video streaming have been quick to adopt HTTP/2 and test how its various features benefits streaming services such that they can play without any lags or latency delays.

Wei et al. [55] developed a K-push strategy where the server pushes the next K-segments to the client without the client requesting each of these K-segments. They used their approach in their next study to reduce power consumption in video streaming onto mobile devices [56]. They actively push the next K-segments of a video to the mobile device to change the HTTP request/response of schedule on the device by matching it with the radio scheduled for reduced power consumption. They claimed to achieve 17% battery savings on the device by using this strategy.

Han et al. [33] modified HTTP/2 server push mechanism to push only the metadata of the resources to the client instead of the actual resource. It is then up to the client to determine from the metadata received whether it wants the resource to be pushed or not. Their strategy improved page load time on mobiles by 45%, and energy consumption saving of up to 37% was reported as well.

Kim et al. [39] studied the performance impacts of HTTP/2 on the popular websites in Korea. They built a test bed to deploy an HTTP/2 server and cloned the popular websites to it. Their simulation found that HTTP/2 increased the time to load some of the websites on 3G/LTE networks. HttpWatch [36] compared the performance of HTTPS, SPDY and HTTP/2. Their analysis found that HPACK is a much more efficient compression algorithm compared to DEFLATE used in SPDY. HTTP/2 and SPDY performed better than HTTPS. However, for page load timings HTTP/2 performance is found to be the best among all three protocols.

There has been some criticism as well for HTTP/2 from the research community. Adi et al. [4] found HTTP/2 servers vulnerable to the low-rate DOS attacks by altering the packets sent to the server. It has also been suggested that [38] push for HTTP/2 to be TLS only is because it benefits some of the

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

corporates and doesn't benefit everyone as SSL has significant costs involved.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we evaluated the energy performance of two web servers communicating with clients over HTTP/1.1, HTTPS and HTTP/2 protocols. HTTP/2 had server push benefits available in all of the tests. Initially, we found that HTTP/1.1 with TLS is the most energy inefficient for the servers to communicate on. We then investigated the benefits of HTTP/2 stream multiplexing. HTTP/1.1 though out did HTTP/2 in the absence of latency it does not represent a real world scenario as 0ms latencies are unrealistic. Lastly, we introduced artificial latencies in our tests to see how an HTTP/2 connection benefiting from all of its features helps servers save energy in comparison to a plain HTTP/1.1 connection.

We concluded that web communicated over a fully optimized HTTP/2 connection on high RTT links would be more energy efficient for web servers compared to HTTP/1.1 and HTTP/1.1 with TLS. It will be useful for web administrators and servers' maintainers at data centers as not needing to optimize HTTP/1.1 connections anymore will mean less work for them. It will lower the electricity costs of the servers and will also help in reducing the consumption of non-renewable sources of energy thereby helping the environment as well.

For this study, our testbed was installed in a lab and client- server communication happened over an Ethernet link. In a real world, packets travel through many networking middle- boxes before they make their way to the client or the server. Therefore, it will be interesting to see if with the help of available cloud services similar experiments can be carried out in a distributed setup and the results presented gets replicated or not. Also, as there is no formal strategy for server push, it can be played around with to see what benefits clients the most. It can be an interesting research problem to come up with a server push strategy that can be generalized for most of the websites on the Internet.

REFERENCES

[1] Eclipse jetty canonical repository. `https://github.com/eclipse/jetty.project`. (last accessed 2016-APR-25).

[2] HTTP/2 Implementations. `https://github.com/http2/http2-spec/wiki/Implementations`. (last accessed: 2016-APR-25).

[3] Jetty project. `http://www.eclipse.org/jetty/documentation/current/high-load.htm`. (last accessed 2016-APR-25).

[4] E. Adi, Z. Baig, C. P. Lam, and P. Hingston. Low-rate denial-of-service attacks against http/2 services. In *IT Convergence and Security (ICITCS), 2015 5th International Conference on*, pages 1–5. IEEE, 2015.

[5] 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.

[6] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic. Parallel tcp sockets: Simple model, throughput and validation. In *INFOCOM*, volume 2006, pages 1–12, 2006.

[7] S. N. I. Association. Emerald power efficiency measurement specification v1.0, august 2011. 73. `http://www.snia.org/sites/default/files/EmeraldMeasurementV1_0.pdf`. (last accessed: 2016-APR-25).

[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. Clidaras, and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3):1–154, 2013.

[10] M. Belshe. A 2x Faster Web. `http://googleresearch.blogspot.ca/2009/11/2x-faster-web.html`. (last accessed: 2016-APR-25).

[11] M. Belshe. More bandwidth doesn't matter (much). `https://www.belshe.com/2010/05/24/more-bandwidth-doesnt-matter-much/`. (last accessed: 2016-APR-25).

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

[12] R. Brown et al. Report to congress on server and data center energy efficiency: Public law 109-431. *Lawrence Berkeley National Laboratory*, 2008.

[13] A. Cardaci, L. Caviglione, A. Gotta, and N. Tonellotto. Performance evaluation of spdy over high latency satellite channels. In *Personal Satellite Services*, pages 123–134. Springer, 2013.

[14] S. A. Chowdhury, V. Sapra, and A. Hindle. Is http/2 more energy efficient than http/1.1 for mobile users? *PeerJ PrePrints*, 3:e1571, 2015.

[15] S. P. E. Corporation. Specpower_ssj 2008. `http://www.spec.org/power_ssj2008/`. (last accessed: 2016-APR-25).

[16] S. P. Council". Spc benchmark 2/energy extension v1.4, november 2012. 73. `http://www.storageperformance.org/specs/SPC-2_SPC-2E_v1.4.pdf`. (last accessed: 2016-APR-25).

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

[18] Y. Elkhatib, G. Tyson, and M. Welzl. Can spdy really make the web faster? In *Networking Conference, 2014 IFIP*, pages 1–9. IEEE, 2014.

[19] 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.

[20] Y. M. M. et al. mruby. `https://github.com/mruby/mruby`. (last accessed 2016-APR-25).

[21] T. File and C. Ryan. Computer and internet use in the united states: 2013. *American Community Survey Reports*, 2014.

[22] "Go Language". Gophertiles. `https://http2.golang.org/gophertiles`. (last accessed: 2016-APR-25).

[23] 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*, 25:520–534, 2015.

[24] 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: 2016-APR-25).

[25] Google. Make the Web Faster. `https://developers.google.com/speed/?csw=1`. (last accessed: 2016-APR-25).

[26] Google. SPDY: An experimental protocol for a faster web. `http://www.chromium.org/spdy/spdy-whitepaper`. (last accessed: 2016-APR-25).

[27] I. Grigorik. Is tls fast yet? `https://istlsfastyet.com/`. (last accessed: 2016-APR-25).

[28] I. Grigorik. *High Performance Browser Networking: What every web developer should know about networking and web performance*. " O'Reilly Media, Inc.", 2013.

[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. Ha, I. Rhee, and L. Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[31] T. J. Hacker, B. D. Athey, and B. Noble. The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pages 10–pp. IEEE, 2001.

[32] T. J. Hacker, B. D. Noble, and B. D. Athey. Improving throughput and maintaining fairness using parallel tcp. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2480–2489. IEEE, 2004.

[33] B. Han, S. Hao, and F. Qian. Metapush: Cellular-friendly server push for http/2. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, pages 57–62. ACM, 2015.

[34] 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.

[35] HTTP Archive. HTTP Trends. `http://httparchive.org/trends.php`. (last accessed: 2016-APR-25).

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016

[36] 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: 2016-APR-25).

[37] B. Hubert. Linux advanced routing & traffic control. http://lartc.org/manpages/tc.txt. (last accessed: 2016-APR-25).

[38] P.-H. Kamp. Http/2.0: the ietf is phoning it in. *Commun. ACM*, 58(3):40–42, 2015.

[39] H. Kim, J. Lee, I. Park, H. Kim, D.-H. Yi, and T. Hur. The upcoming new standard http/2 and its impact on multi-domain websites. In *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*, pages 530–533. IEEE, 2015.

[40] J. Koomey. Growth in data center electricity use 2005 to 2010. *A report by Analytical Press, completed at the request of The New York Times*, page 9, 2011.

[41] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 319–330. IEEE, 2011.

[42] D. Mosberger and T. Jin. httperf—a tool for measuring web server performance. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):31–37, 1998.

[43] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste. The cost of the s in https. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 133–140. ACM, 2014.

[44] M. Nottingham. HTTP/2 Approved. http://www.ietf.org/blog/2015/02/http2-approved/. (last accessed: 2016-APR-25).

[45] Office for National Statistics. Internet access – households and individuals 2015. http://webarchive.nationalarchives.gov.uk/20160105160709/http://www.ons.gov.uk/ons/dcp171778_412758.pdf. (last accessed: 2016-APR-25).

[46] 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).

[47] 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.

[48] 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.

[49] 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.

[50] S. Rivoire, M. A. Shah, P. Ranganathan, C. Kozyrakis, and J. Meza. Models and metrics to enable energy-efficiency optimizations. 2007.

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

[52] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the energy efficiency of a database server. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 231–242. ACM, 2010.

[53] T. Tsujikawa. Nghttp2: Http/2 c library. https://github.com/nghttp2/nghttp2. (last accessed 2016-APR-25).

[54] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. How speedy is spdy? In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 387–399, 2014.

[55] S. Wei and V. Swaminathan. Low latency live video streaming over http 2.0. In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, page 37. ACM, 2014.

[56] S. Wei, V. Swaminathan, and M. Xiao. Power efficient mobile video streaming using http/2 server push. In *Multimedia Signal Processing (MMSP), 2015 IEEE 17th International Workshop on*, pages 1–6. IEEE, 2015.

[57] World Wide Web Consortium (W3C). The Original HTTP as defined in 1991. http://www.w3.org/Protocols/HTTP/AsImplemented.html. (last accessed: 2016-APR-25).

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2027v1 | CC-BY 4.0 Open Access | rec: 4 May 2016, publ: 4 May 2016