Evaluating the Features of HTTP/2 for the Internet of Things

Conference Paper · May 2017

CITATION
1

READS
1,284

1 author:

Asma Elmangoush
College of Industrial Technology, Misurata
43 PUBLICATIONS **389** CITATIONS

SEE PROFILE

Evaluating the Features of HTTP/2 for the Internet of Things

Asma Elmangoush
College of Industrial Technology, Electronic Engineering Department
Misrata, Libya
asma.elmangoush@ieee.org

***Abstract* – The standard specification of the second major upgrade of the HTTP protocol was finalized and approved recently. Since most of its technical features are inherited from SPDY, an application-layer protocol developed by Google, HTTP/2 is expected to solve many of the shortcomings and inflexibilities of HTTP/1.1 in Web performance. Moreover, it could lead to better implementation in the Internet of Things (IoT) systems. This paper aims to evaluate the features of HTTP/2 protocol and its benefit for IoT systems. For that reason, a testbed is implemented using open source software and Raspberry Pi boards. The testbed enables the deployment of practical IoT scenarios to investigate the performance of the HTTP/2 protocol.**

***Index Terms*—HTTP, Internet of Things, Application Protocols, Evaluation.**

I. INTRODUCTION

Nowadays, we are witnessing the era of connecting all kind of physical and virtual objects to the Internet. This enables the transformation from connecting people to connecting things. This new model, named the Internet of Things (IoT), extends the connected world exponentially including more physical objects besides computers and smartphones in a global network. According to Cisco's estimation, the number of Internet connections should grow nearly 2.5-fold, from 4.9 billion in 2015 to 12.2 billion in 2020 [1]. Moreover, Intel estimates that around 26 smart objects for each human being will be connected to the Internet by 2020 [2].

The integration of a massive number of devices with heterogeneous features and capabilities into the Internet, demands a fully interconnected and scalable framework. The main challenges for IoT are driven from integrating low-power devices and low-bandwidth networks. The heterogeneity of integrated communications technologies, targeted service domain and data representation, highlights the need to study the communication requirements of various services and the traffic patterns in the system. Focusing on the connectivity aspects, it is noticed the highly fragmented situation of the current protocol stack, which is needed to be supported due to the variation in connected devices and application's requirements [3]. As Fig. 1 illustrates, the protocol stack for IoT includes various standardized protocols at transport, network and data link layers, where each layer is contributing to addressing at least one challenge in the new communication paradigm, such as scalability, reliability and security.

Typically, the IoT systems utilize IP-compatible open protocols that are standardized, in order to be widely deployable. Despite the fact that IP-based protocols require large memory footprint, the IP-based approach is gaining more momentum in large-scale deployments in the context of Smart services. In addition to Hypertext Transfer Protocol (HTTP), which is the de facto Internet transport protocol, various protocols have been proposed to support such developments, such as the Constrained Application Protocol (CoAP) [4] and Message Queue Telemetry Transport (MQTT) [5], aiming to address the requirements resulting from integrating resource-constrained devices.

Recently, the HTTP Working Group has published the specification of HTTP/2 in RFC 7540 [6], which aims to

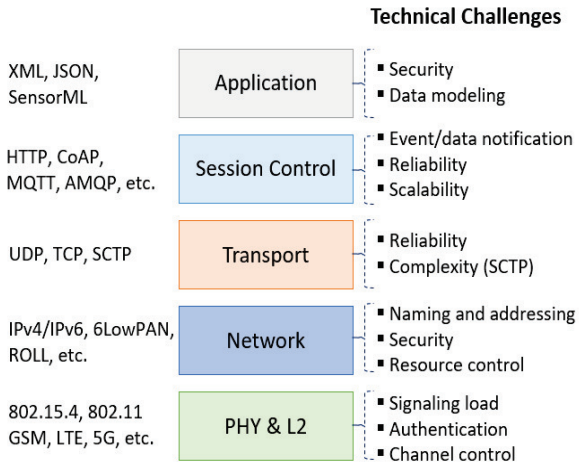

Figure 1. Protocol Stack and Technical Challenges.

 ISBN 978-9959-9641-0-6

enable a more efficient use of network resources and a reduced perception of latency by introducing header field compression and allowing multiple concurrent exchanges on the same connection. However, there are still no sufficient study that validate the efficiency of using HTTP/2 in the IoT systems.

This paper contributes to a better understanding of the HTTP/2 protocol from a practical standpoint. In this paper, we review the second major version of the HTTP protocol and evaluate its performance as an application layer protocol for IoT systems. The evaluation process considered also various channel conditions and resources at the embedded devices commonly used for IoT. The rest of the paper is organized as follows, Section II presents a classification of applications protocols commonly used in IoT and an overview of the second version of the HTTP protocol. Section III provides a description of the designed testbed and evaluation tests. Section IV presents the results obtained through the evaluation process and discuss the technical futures of the tested protocols in IoT scenarios. Finally, Section V provides conclusions and future work.

II. Background and Related Work

A. Classification of IoT Protocols

The lack of a protocol that handles different requirements of IoT applications has resulted in a fragmented market between many protocols. The comparison between different protocols has been the subject of discussions in recent research work [7], [8]. Moreover, various protocols have been proposed to the IoT development, aiming to address the requirements of integrating resource-constrained devices and supporting ubiquitous access. Table 1 presents a comparison of the IoT transport protocols. These protocols could be categorized under two main communication paradigms:

- The Request/Response (RR) model, commonly used in distributed system to exchange information through message passing between a sender and a receiver. The RR model adapts the polling mechanism to enable users to retrieve the state of other entities.
- The Publish/Subscribe (PS) model, which is based on an event broker to forward updates (notifications) to interested users (subscribers), regarding changes of senders' (publishers') statuses.

The RR model is adapted in common transport protocols like HTTP/1.1 and CoAP, applying the RESTful architecture. Other protocols adapting the PS model include MQTT and Advanced Message Queuing Protocol (AMQP). On the one hand, the polling mechanism used in Req/Res model is considered insufficient to be implemented within systems that have infrequent status changes. On the other hand, the Pub/Sub model lacks the end-to-end delivery reliability due to the existence of intermediary entities (i.e. broker) between publishers and subscribers.

The OneM2M consortium [9], which aims to become the world-wide accepted standard for Machine-to-Machine (M2M) and IoT communication, has specified a high-level architecture at both the field and infrastructure domains, Furthermore, the oneM2M protocol working group has specified mapping of its standard Application Programming Interfaces (APIs) to several underlying transport protocols to meet the requirements of various IoT use cases. However, there are no sophisticated guidelines of protocol stack deployment for different IoT services.

B. The HTTP/2 Protocol

HTTP/2 was recently introduced in RFC 7540 [6]. It has been developed by the IETF HTTP Working Group, however, it is mainly based on Google's experimental

TABLE 1. IoT Transport Protocols Comparison

Protocol	HTTPv1	HTTPv2	CoAP	MQTT	AMQP
Standards	IETF RFC2616	IETF RFC 7540	IETF RFC7252	Proposed OASIS standard MQTT	OASIS AMQP
Architecture Style	Client/server model RESTful	Client/servers model	Client/server model RESTful	Brokered style	Brokered style
Transport	TCP	TCP	UDP	TCP	TCP
Messaging	Request/Response	Supports multiplexing of request/response	Request/Response	Publish/Subscribe (P2P or Brokered)	Publish/Subscribe
Header	Text-based	Binary (header compression)	4Byte Binary-based	Fixed length of 2Byte	8Byte
Message size	Larger, partly because status detail is text-based	Configurable by server	Small to fit in single IP datagram with 4byte header	Up to 256MB with 2byte header	Unlimited with 8byte header
Dynamic Discovery	No	No	Yes	No	No
Service levels (QoS)	All messages get the same level of service	Priority mechanism of streams	Confirmable or non-confirmable messages	Three quality of service settings	Different 3 QoS levels
Data distribution	One-to-one	One-to-one and one-to-many	One-to-one	One-to-one and one-to-many	One-to-one and one-to-many
Security	Typically based on Secure Sockets Layer (SSL) or TLS	Requires TLS version 1.2 or higher	Typically based on SSL or TLS	Simple Username/ Password Authentication, SSL for data encryption	SASL authentication, TLS for data encryption

 ISBN 978-9959-9641-0-6

SPDY protocol. The second version of the HTTP protocol aims at enabling a more efficient use of network and server resources, and a reduced perception of latency by introducing header field compression facility and allowing multiple concurrent exchanges on the single connection from browsers to a Web site. HTTP/2 also introduces unsolicited push of representations from servers to clients. Currently, HTTP/2 is used by almost 12% of all websites [10].

Some work has been to study the main aspects of the HTTP/2 protocol, and how its features impact the web performance. The results presented in [11], shows that the HTTP/2 specified mechanisms can lead to clear performance enhancement in web page loading time. Moreover, the tests presented in [12] show that HTTP/2 has better energy consumption performance than HTTP/1.1.

The main improvements of HTTP/2 that could be beneficial for the IoT solutions include:

1) Header Compression and binary encoding

In HTTP/1 the header is text-based and its size keeps increasing with request and response messages. HTTP/2 attempts to decrease the header size by using a special algorithm, called HPACK. It is a binary encoding algorithm defined in RFC7541 [13], that efficiently compressing HTTP/2 header fields. This feature shall reduce the size of request and response messages, and make HTTP/2 more suitable for embedded devices and low bandwidth internet connections.

2) Streams and Multiplexing

Multiplexing is a process of sending a list of HTTP requests in a single TCP connection. This feature aims to solve the head of line blocking issue with the HTTP/1 protocol. HTTP/2 introduces the notion of stream: each client request is assigned to a particular stream, and all streams are multiplexed over a single TCP connection. As a consequence, requests do not influence each other and can be simultaneously answered by the server. Hence, HTTP/2 solves the head of line blocking issue while at the same time reducing the number of TCP connections to be handled on the server side. In addition, this has a better latency improvement comparing with the pipelining feature that was introduced in HTTP 1.1.

3) Server Push

Server push enables server to push resources to the client before asking them. When a client requests a file or web site to server, the server can send other resources anticipating that the client will make other requests to get all the page dependencies (CSS, JavaScript, images, etc.). The client can then choose to refuse or accept the data, in which case, it will be stored in the browser's cache for later use. In IoT systems, a server pushing functionality could enable pushing resources to clients in a Pub/Sub model. This feature is useful when sending notifications to multiple end-users.

III. Evaluation Methodology

HTTP/2 is a new protocol and there are many implementations of it, which support client and/or server functionality. However, not all of them provides a full implantation of specified features as described in the RFC 7540 document. This poses an essential challenge in selecting the suitable tools for our evaluation process. Table 2 shows a brief comparison of supported features in some available implementations.

For the testing purpose in this research, the NGINX [14] was selected to use as a server. It is available as a pre-compiled HTTP/2 stack with configurable options. In addition, NGINX is one of the well-known and trusted servers with a huge community support and good documentation available online. In the evaluation testbed, the HTTP/2 server was implemented on a 64-bit Ubuntu 14.04 PC (8GB RAM, 2.2GHx Duel-Core). In the client side, Hyper [15] was found to be an easy fit due to its

TABLE 2. Open-source Implementations for HTTP/2

Name	Role	Programming language	Installation	HTTP/2 Support	SSL support
H2O [17]	Server, Proxy	C	Binary or source	+ It fully implements prioritization of HTTP/2 responses. - Limited documentation	Yes
Apache [18]	Server	C	Binary or source	+ Supported via the module mod_h2. + Appropriate patches must be applied to the source code of the server.	Yes
NGINX [14]	Server	C	Binary or source	+ Supported via the Module ngx_http_v2_module + Enables weighted and dependency-based prioritization.	Yes
Hyper [15]	Client, server	Python	Binary or source	+ Supports the final version of the HPACK draft specification + Good documentation.	Yes
Jetty [19]	Client, intermediary, server	Java	Binary only	+ Developed as part of the Eclipse foundation	Yes

 ISBN 978-9959-9641-0-6

ease of use. It could be used directly from command-line and therefore HTTP messages could be debugged easily. Two Raspberry Pi boards (model B) were used as clients, which were connected to the HTTP/2 server via a WLAN switch, as depicted in Fig 2.

In addition, a network emulator tool was used to configure a testbed setup as close as possible to real circumstance in IoT systems. This emulator is used to introduce delay and different packet drop probabilities in the communication channel. The netem tool, which is inbuilt in many Linux distributions, was used in the presented tests. The values of different error probabilities in Gilbert-Elliot Model were set to the values observed in real time services on the internet [16]. The Gilbert-Elliot Model is a common way to sample a regular connection. It basically defines that there are two states, a good and a bad state, that each has a certain probability of packet drop. Switching states also have a defined probability.

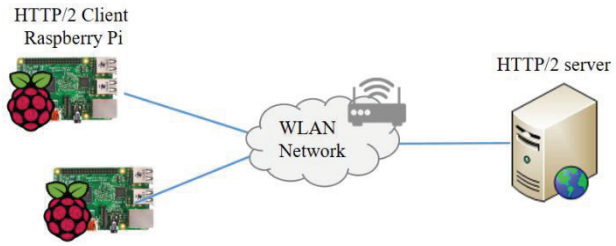

Figure 2. Evaluation Testbed.

To evaluate the HTTP/2 protocol usage as an application protocol for IoT services, a number of tests were executed. The conducted tests included four scenarios, commonly composed in IoT services:

1. A client requests one small data packet (less than 1Kb) from the server.
2. Clients pulling a stream of data in the same size (less than 1Kb) from the server, where the availably of the data was announced in advance.
3. Clients pulling a stream of data in different sizes from the server.
4. Clients pushing a stream of data in small size (50 bytes) to the server.

Each test scenario was performed under three different channel conditions to observe the performance in various communication circumstances:

1. No limitations: this represent the perfect networking situation.
2. 1 second delay: this emulates a high delay network, were each packet get delayed for 1 sec.
3. Gilbert-Elliot Model: widely used model for describing error patterns in transmission channels.

IV. Results and Technical Discussion

In this section, the results of previously described tests will be presented and discussed.

The first test scenario is very simple, where the clients request a data packet of small size (612 Bytes) from the server. Fig. 3 and Fig. 4 plot the response time and measured throughput of the traffic between the server and one client, respectively. The results illustrate that HTTP/1 is the fastest with minimum code footprint. While both HTTP/1(with SSL) and HTTP/2 have similar performance, that is due to the overhead caused by encryption.

Figure 3. Mean Response Time of Single Request Scenario.

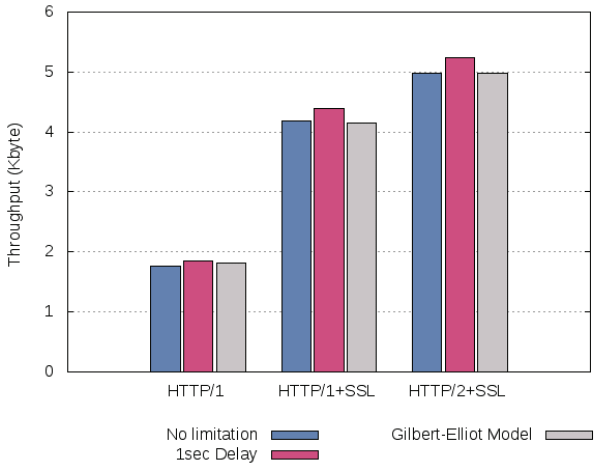

Figure 4. Mean Throughput of Single Request Scenario.

The next test scenarios purpose to check the performance of the streaming and multiplexing feature of HTTP/2 protocol. With HTTP/2 protocol, clients can make multiple GET requests to the server in a single TCP connection. This shall lead to a more efficient bandwidth consumption.

In the second test scenario, all clients make a series of requests with a rate of 5 req/s to pull a stream of data from the server, each packet in the size of 612 Bytes. In this case, the HTTP/2 protocol outperformed the other

 ISBN 978-9959-9641-0-6

protocols deployment significantly, as the response time plotted in Fig. 5 presents. Furthermore, the throughput measured during the tests, in Fig. 6, shows how the benefit of the multiplexing feature of HTTP/2 in reducing the consumed bandwidth.

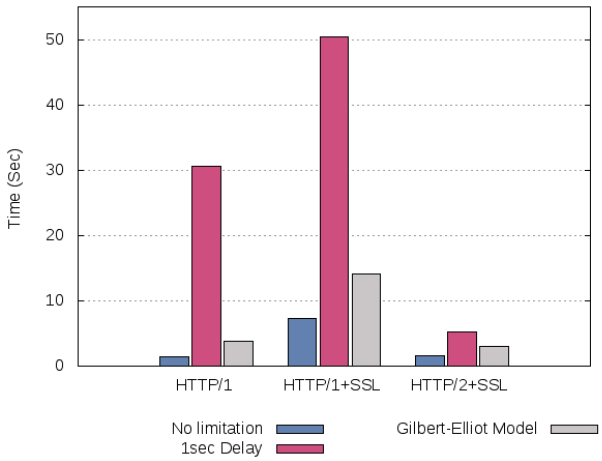

Figure 5. Mean Response Time of Pulling Small Packets Stream Scenario.

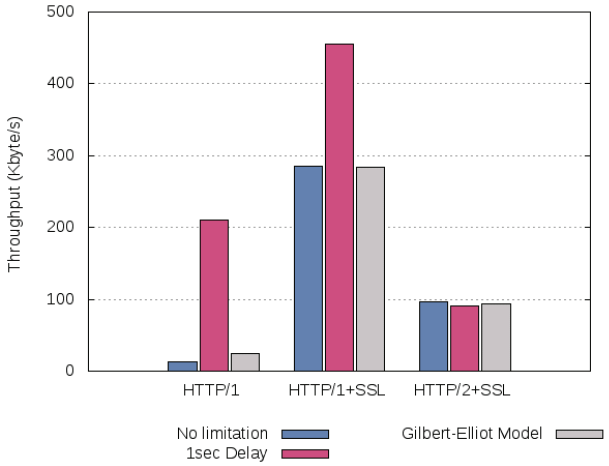

Figure 6. Mean Throughput of Pulling Small Packets Stream Scenario.

As the IoT involves heterogeneous data traffic between devices and server/gateway, a third test has been performed. In this test, the clients conduct multiple requests to get data files in a two different size, a big file 11MB and a small file 612 Bytes alternatively. This scenario aimed to test the ability of HTTP/2 to avoid the head of line blocking problem, which results from the First-in First-out request-response mechanism. As Fig. 7 shows, HTTP/2 has a huge delay compared to the other two cases. The measure throughput was also nearly equal in all cases. It's obvious that HTTP/2 brings no advantage in the performance in similar scenarios.

Fig. 8 shows the result for the fourth test, where POST method from all clients are pushing small size of data to the server in 10Hz request rate. The performance of HTTP/2 is superior as the plot shows. The response time was significantly lower when using HTTP/2.

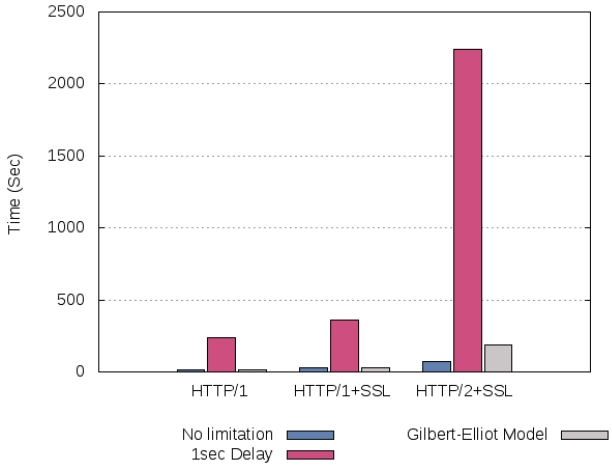

Figure 7. Mean Response Time of Multiple Size Packets Stream Scenario.

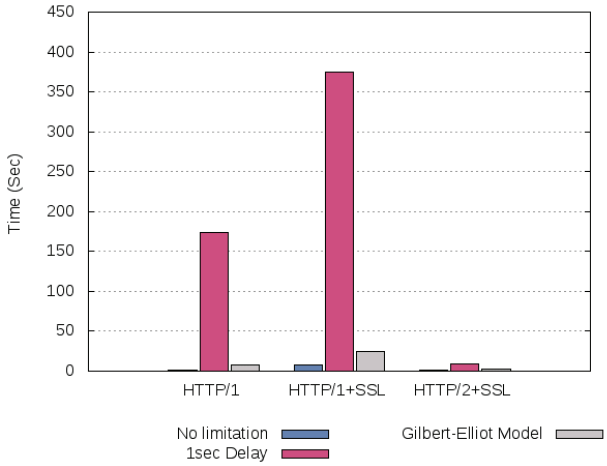

Figure 8. Mean Response Time of Pushing Small Size Packets Stream.

V. Conclusion and Future Work

Recently, many web browsers have supported HTTP/2 protocol and more websites are using it to improve the user's web experience. It has been improved that HTTP/2 protocol can improve the web performance significantly. However, its performance in IoT service has not been investigated. This paper contributes to test the performance of the HTTP/2 protocol in IoT scenarios.

In this paper, we reviewed the HTTP/2 protocol features and evaluate its performance as an application layer protocol for IoT systems. The evaluation process considered various channel conditions and resources commonly used for IoT. In the one hand, the results show that HTTP/2 has similar performance as HTTP/1.1 for small data transfers. Although the code footprint is similar to HTTP/1.1 which is very high for constrained nodes in IoT. On the other hand, HTTP/2 is significantly

 ISBN 978-9959-9641-0-6

faster when many concurrent requests have to be done. This is due to the support of multiplexing in HTTP/2, which makes multiple requests over single TCP connection. On the web, HTTP/2 streams help to solve the head of line blocking but in our tests, it did not seem very useful. We can also conclude that large packet size with SSL encryption makes HTTP/2 really heavy to be used in IoT.

As future work, the testbed shall be extended to add more clients and evaluate the effect of increasing number of nodes. In addition, more protocols, such as MQTT and AMQP, will be evaluated.

REFERENCES

[1] Cisco, "Cisco Global Cloud Index: Forecast and Methodology 2015-2020," 2016.

[2] Intel, "Infographic: Guide to The Internet of Things." [Online]. Available: http://www.intel.com/content/www/us/en/intelligent-systems/iot/internet-of-things-infographic.html.

[3] A. Elmangoush, A. A. Corici, R. Steinke, M. Corici, and T. Magedanz, "A Framework for Handling Heterogeneous M2M Traffic," *Procedia Computer Science.*, vol. 63, pp. 112–119, 2015.

[4] Z. Shelby, K. Hartke, and C. Bormann, "RFC 7252: The Constrained Application Protocol (CoAP)." 2014.

[5] International Business Machines Corporation (IBM), "MQTT V3.1 Protocol Specification," 2010.

[6] M. Belshe, R. Peon, and M. Thomson, "RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2)." 2015.

[7] T. Yokotani and Y. Sasaki, "Comparison with HTTP and MQTT on required network resources for IoT," in *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, 2016, pp. 1–6.

[8] J. E. Luzuriaga, M. Perez, P. Boronat, J. C. Cano, C. Calafate, and P. Manzoni, "A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks," in *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, 2015, pp. 931–936.

[9] oneM2M, "OneM2M." [Online]. Available: http://onem2m.org/.

[10] W3Techs, "Usage Statistics of HTTP/2 for Websites, February 2017." [Online]. Available: https://w3techs.com/technologies/details/ce-http2/all/all. [Accessed: 13-Feb-2017].

[11] H. de Saxce, I. Oprescu, and Y. Chen, "Is HTTP/2 really faster than HTTP/1.1?," in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2015, pp. 293–299.

[12] S. A. Chowdhury, V. Sapra, and A. Hindle, "Is HTTP/2 more energy efficient than HTTP/1.1 for mobile users?," *PeerJ Prepr.*, 2015.

[13] "RFC 7541 - HPACK: Header Compression for HTTP/2," 2015.

[14] "HTTP/2 Supported in Open Source NGINX 1.9.5 | NGINX." [Online]. Available: https://www.nginx.com/blog/nginx-1-9-5/. [Accessed: 20-Feb-2017].

[15] "Hyper: HTTP/2 for Python — hyper 1.0.0 documentation." [Online]. Available: https://python-hyper.org/en/latest/. [Accessed: 20-Feb-2017].

[16] O. Hohlfeld, R. Geib, and G. Hasslinger, "Packet Loss in Real-Time Services: Markovian Models Generating QoE Impairments," in *2008 16th Interntional Workshop on Quality of Service*, 2008, pp. 239–248.

[17] "H2O - an optimized HTTP server with support for HTTP/1.x and HTTP/2." [Online]. Available: https://github.com/h2o/h2o.

[18] "The Apache HTTP Server Project." [Online]. Available: http://httpd.apache.org/. [Accessed: 20-Feb-2017].

[19] "Jetty - Servlet Engine and Http Server." [Online]. Available: https://www.eclipse.org/jetty/. [Accessed: 20-Feb-2017].

 ISBN 978-9959-9641-0-6