Performance Comparison and Evaluation of WebSocket Frameworks: Netty, Undertow, Vert.x, Grizzly and Jetty

Yukun Wang[1], Lei Huang[1], Xiaoyou Liu[2], Tao Sun[2], Kai Lei[1, *]
[1]Shenzhen Key Lab for Information Centric Networking & Blockchain Technology (ICNLAB),
School of Electronics and Computer Engineering (SECE), Peking University, Shenzhen 518055, P.R. China
[2]The Network Information Center, University Town of Shenzhen, Shenzhen 518055, P.R. China
Email: {ykwang, lhuang}@pku.edu.cn, {liuxy, suntao}@utsz.edu.cn
Corresponding Author*: leik@pkusz.edu.cn

Abstract—**The WebSocket protocol emerges to supersede existing bidirectional communication technologies that use HTTP as a transport layer. Currently, there are many network application frameworks that support the WebSocket protocol, but have different behaviors in performance of various aspects. To study and compare the performance of common WebSocket frameworks, say Netty, Undertow, Vert.x, Grizzly and Jetty, in this paper, we use concurrency test, flow test, connection test and resource occupancy test. The experiment results show that Netty and Undertow perform better in highly concurrent environments, while Grizzly is suitable for large flow conditions. The results also show that with persistent connection, Netty far outperforms other frameworks, and that Vert.x and Undertow can handle most requests within relatively shorter time. Besides, Netty and Vert.x occupy less CPU and memory resources in comparison with other frameworks.**

Keywords- WebSocket; Netty; performance evaluation; concurrency test; flow test

I. INTRODUCTION

Before the emergence of WebSocket, creating web applications that need bidirectional communication between a client and a server (e.g., instant messaging and gaming applications) has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls [1]. The WebSocket Protocol is designed to supersede existing bidirectional communication technologies that use HTTP as a transport layer [1]. Such technologies were implemented as trade-offs between efficiency and reliability because HTTP was not initially meant to be used for bidirectional communication [1]. Some researchers adopt a completely different basic network architecture called NDN [2] and can achieve congestion control based on RCP [3]. Currently, with the development of web technologies, there are a number of web application frameworks that support the WebSocket protocol, such as Netty, Undertow, Vert.x, Grizzly and Jetty. Netty is a NIO client server framework which enables quick and easy development of network applications with high performance [4]. Undertow is a flexible performant web server written in java, providing both blocking and non-blocking API's based on NIO [5]. Eclipse Vert.x is a tool-kit for building reactive applications on the JVM [6]. The Grizzly NIO framework has been designed to help developers to take advantage of the Java NIO API to build scalable and robust servers [7]. Eclipse Jetty provides a Web server and javax.servlet container, plus support for HTTP/2, WebSocket, OSGi, JMX, JNDI, JAAS and many other integrations [8].

This paper aims to compare and evaluate the performance of the above mentioned frameworks from various aspects (all implemented in Java). To design the experiments, we mainly focus on four aspects: concurrency, flow, connection type and resource occupancy. To achieve solid conclusions, we conduct the experiments and analyze the results in detail. It should be mentioned that there are also other WebSocket frameworks that we do not compare in this paper, such as Spray, Node.js and Go because they are not implemented in Java, therefore, it may be somewhat not fair to compare them together.

The main contributions of this paper can be outlined as follows:

- We compare the prevailing Netty framework with other widely used WebSocket frameworks, say Undertow, Vert.x, Grizzly and Jetty, making a conclusion of which situation they ought to be used.
- By means of concurrency test, flow test, connection test and resource occupancy test, we can evaluate performance from various aspects, including the number of concurrent requests, the flow amount, the connection type and CPU and memory resource occupancy.

The rest of this paper is organized as follows. Section II discusses related work. Section III introduces the experiment environment, including hardware environment, server environment and testing tools. Section IV describes our test methodology in detail. Section V presents the corresponding results and analysis. Finally, Section VI concludes.

II. RELATED WORK

To the best of our knowledge, there have been few studies evaluating and analyzing common WebSocket frameworks, say Netty, Undertow, Vert.x, Grizzly and Jetty. However, there have been some studies considering WebSocket protocol

978-1-5386-4870-4/18/$31.00 ©2018 IEEE

performance evaluation. D. Skvorc evaluated the performance of WebSocket protocol with respect to the underlying TCP protocol, and compared the two against the latency and amount of generated network traffic [9]. D. Skvorc found that except a small overhead imposed due to initial handshaking, WebSocket-based communication does not consume any more network traffic than plain TCP based communication [9]. Gábor Imre presented a WebSocket benchmark infrastructure created for measuring server-side performance of the WebSocket protocol [10]. This infrastructure enables black-box measurements independently of the server-side implementation.

Besides, there have been some studies related to WebSocket algorithms and applications. Ajinkya Mulay proposed a WebSocket connection management algorithm for IoT, focusing on improving and adapting the WebSocket protocol for connecting IoT devices [11]. The algorithm can be applied to early warning earthquake alert applications. Hirotaka Nakajima proposed HTTP over WebSocket proxy system to defeat delay [12]. HTTP was used to achieve end-to-end communication. Each HTTP request and response was encapsulated into WebSocket packet at the client and server. Junjie Feng presented a solution for runtime browser session migration and management based on WebSocket. The protocol provides a persistent connection between the server and clients, and either of them can initiate data transfer at any time [13].

There have also been some studies with respect to the applications of Netty, Jetty and Vert.x. Shouheng Zhang described a server structure based on Netty for an internet-based laboratory [14]. It has excellent scalability. Zhang Yu put forward a kind of optimization design method of communication service system for vehicle remote monitoring based on the Netty pattern [15]. The system can achieve high efficiency and accuracy of data communication and information exchange. Lin Biying focused on using Jetty as a server network management system, with a custom asynchronous Servlet to improve the performance of an intelligent network management system [16]. Pratibha P. Dhekale used Jetty server to achieve efficient data search using Map Reduce framework [17]. Venkatesh-Prasad Ranganath proposed a set of communication patterns to enable the construction of medical systems by composing devices and apps in Integrated Clinical Environments (ICE) [18]. The proposed patterns have been successfully implemented on Vert.x.

III. Experiment Environment

In this section, we introduce the hardware environment and server configuration of the five WebSocket frameworks, as well as the testing tools used to evaluate the performance of these frameworks.

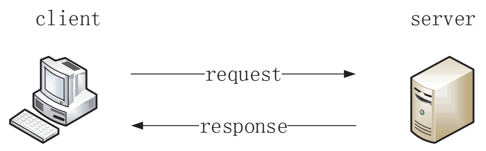

Fig. 1. Testbed setup.

A. Hardware Environment

The testbed of our experiment consists of two parts, the client and the server, as shown in Fig. 1. The five selected WebSocket frameworks are implemented and deployed separately on the server. The Apache Bench is deployed on the client. Due to the feature of Apache Bench that it occupies little resources which can be omitted in the experiment, we deploy the server and the client into one single machine.

The machine in our testbed runs Microsoft Windows 7 SP1 x64, with an Intel Core i5-3470 processor with four cores, 8GB of RAM and a 500GB disk. All non-essential processes were killed to prevent their impact on the experiment results.

B. Server Configuration

It's mainly to compare WebSocket frameworks of Netty, Undertow, Vert.x, Grizzly and Jetty in our study, so we implement and deploy these five frameworks separately.

- Apache: To achieve stable and accurate results, we use Apache 2.4.29, which is the latest stable version available. Compared with earlier versions of Apache, this version contains core enhancements, module enhancements and program enhancements [19].
- Netty: We use Netty 4.1.19.Final, which is also the latest stable version. Netty is an asynchronous event-driven network application framework for rapid development of maintainable high performance protocol servers and clients [4].
- Undertow: For Undertow, we use 1.4.12.Final. It is the current stable branch, which is recommended for production use. Undertow is sponsored by JBoss. It has a composition based architecture that allows you to build a web server by combining small single purpose handlers [5].
- Vert.x: Vert.x 3.5.0. Eclipse Vert.x is a tool-kit for building reactive applications on the JVM. It is also event driven and non-blocking [6].
- Grizzly: We use Grizzly 2.4.0, the latest stable release. Grizzly is to help developers to build scalable and robust servers using NIO as well as offering extended framework components, such as WebSocket [7].
- Jetty: We use the latest release of Eclipse Jetty, 9.4.8.v20171121. Eclipse Jetty provides a Web server and javax.servlet container, plus support for HTTP/2, WebSocket, OSGi, JMX, JNDI, JAAS and many other integrations. These components are open source and available for commercial use and distribution [8].

C. Testing Tools

We divide the tests into four parts: concurrency test, flow test, connection test and resource occupancy test. We use two testing tools in order to make our tests more comprehensive. For the first three parts, we use Apache Bench to accomplish the test, while for the last part, we use VisualVM.

- Apache Bench: Apache Bench is a stress testing tool provided in Apache to benchmark Apache Hypertext Transfer Protocol (HTTP) server as well as other servers. It especially shows how many requests per second the server is capable of serving. In our test, we use Apache Bench provided in Apache 2.4.29, which is the latest stable version available [19].
- VisualVM: VisualVM is a visual tool integrating command line JDK tools and lightweight profiling capabilities [20]. It is designed for both development and production time use. We use the latest version 1.4 to inspect the CPU and memory resources occupied by the different WebSocket frameworks. We can get both visual and numerical results through it.

IV. TEST METHODOLOGY

The experiments evaluated the results from four aspects, i.e. concurrency test, flow test, connection test and resource occupancy test. To achieve accurate and effective results, we must follow the one-factor-at-a-time experiment principle [21] to conduct out tests.

A. Concurrency Test

Concurrency test aims to observe and evaluate the performance of different WebSocket frameworks at different levels of concurrency number. In our test, we implement the five WebSocket frameworks separately, we use non-persistent HTTP connections, and the servers reply with HTML documents of the same length (less than 1KB). We set the total number of requests to 10,000 and vary the concurrent request number from 10 to 1,000 (more specifically, we use 10, 100, 200, 500 and 1,000 as the Apache Bench concurrency parameter) to test the performance of the five different frameworks. We record a few parameters provided by the Apache Bench tool, including Requests per Second (RPS), Time per Request (TPR) and 90% processing time.

For all the tests, we follow the one-factor-at-a-time principle, and to avoid the inaccuracy brought by accidental deviation, each test is repeated three times and the average number is adopted. Besides, we reboot the server after each single test to make sure that they are equally treated.

B. Flow Test

Flow test is to test the performance of different frameworks when faced with different amount of data flow. In our test, we implement the five WebSocket frameworks separately, we use non-persistent HTTP connections, the concurrent request number is fixed to 100 and the total number of requests is fixed to 10,000, while the servers reply with HTML documents of different lengths (less than 1KB, 5KB, 10KB, 20KB, 50KB and 100KB). We record the Time per Request (TPR) parameter provided by the Apache Bench tool.

Like in concurrency test, each test is repeated three times and the average number is adopted. Besides, the server is rebooted after each single test.

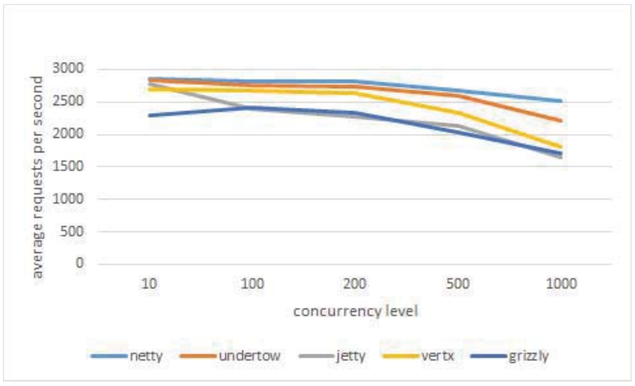

Fig. 2. Concurrency test result.

C. Connection Test

We use connection test to see the different performance of these frameworks using persistent connections (i.e. HTTP keep-alive) and non-persistent connections (i.e. HTTP close). In our test, we implement the five WebSocket frameworks separately, the concurrent request number is fixed to 100 and the total number of requests is fixed to 10,000, and the servers reply with HTML documents of the same length (less than 1KB), while we use persistent connections and non-persistent connections. We record the Requests per Second (RPS) parameter provided by the Apache Bench tool.

Same as in concurrency test, each test is repeated three times and the server is rebooted after each single test.

D. Resource Occupancy

Resource occupancy test is conducted to observe and compare the CPU and memory resources occupied by different frameworks. In our test, we implement the five WebSocket frameworks separately, we use non-persistent HTTP connections, the concurrent request number is fixed to 100, the total number of requests is fixed to 10,000, and the servers reply with HTML documents of the same length (less than 1KB). We use VisualVM to observe the CPU and memory resources taken up by the frameworks.

Same as the tests above, each test is repeated three times and the server is rebooted after each single test.

V. EXPERIMENT RESULTS

A. Results and Analysis of Concurrency Test

We did concurrency test to the five WebSocket frameworks at different concurrency levels, with the document length less than 1KB and completed 10,000 requests. The results are shown in Fig. 2. As we can see, with the increase of concurrency level, the number of requests handled per second decreases, this is because the frameworks cannot handle so many concurrent requests at the same time. Besides, with the change of the concurrency level, Netty has the highest RPS most of the time. When the concurrency level is less than 500, Undertow has similar behaviors to Netty, however, when the concurrency level comes to 1,000, Netty's performance far

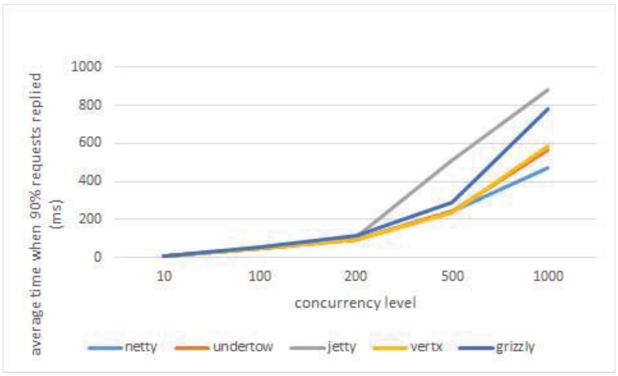

Fig. 3. Time when 90% requests handled.

outperforms other frameworks. Behind Undertow is Vert.x, while Jetty and Grizzly have relatively smaller RPS.

Fig. 3 shows the average time when 90% of the requests are replied. It can be seen that when the concurrency number is large, Netty has an absolute advantage. But when the concurrency level is not that high, Vert.x and Undertow can handle 90% of the requests as fast as possible.

Through concurrency test, we can see that Netty and Undertow are suitable for circumstances with large number of concurrent requests, while Grizzly and Jetty do not perform well under the same circumstance. Vert.x and Undertow can handle most requests in a short time when the concurrency level is not that high.

B. *Results and Analysis of Flow Test*

We did flow test to study the performance of different frameworks when faced with different amount of data flow. The results are shown in Fig. 4. It is shown that when the data returned per request is less than 20KB, Netty, Undertow and Vert.x far outperform Grizzly and Jetty. However, with the

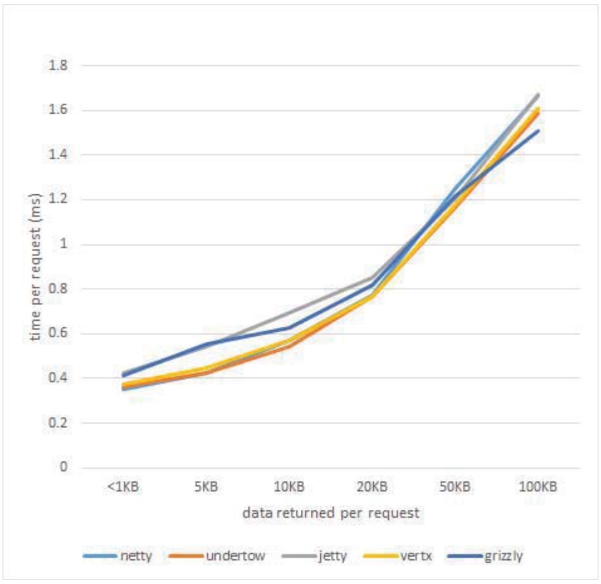

Fig. 4. Flow test result.

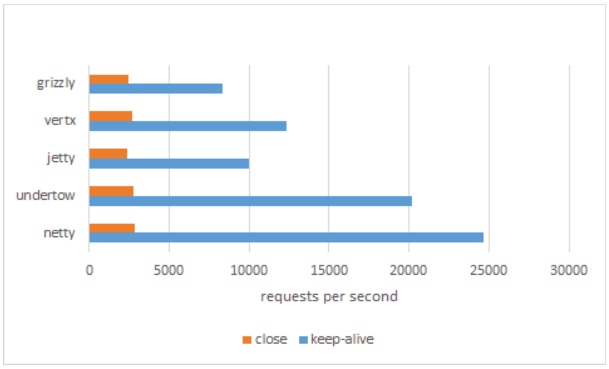

Fig. 5. Connection test result.

increase of data, Grizzly performs better and better, and when data returned per request comes to 100KB, Grizzly performs the best, i.e. has the least time per request.

Through flow test, we can draw the conclusion that Grizzly is fit for large data flow conditions, under which the data returned per request is relatively large.

C. *Results and Analysis of Connection Test*

We did connection test to observe the performance of the five frameworks using persistent connections (i.e. HTTP keep-alive) and non-persistent connections (i.e. HTTP close). With non-persistent connections as default, the Apache Bench sets up an HTTP connection, completes data transfer and closes the connection after that. But with persistent connections, the connection may not be closed immediately, and several requests may be replied with the same connection, so the throughput may be improved.

It is shown is Fig. 5 that with persistent connections, all frameworks perform far better than with non-persistent connections. However, the performance of Netty and Undertow, especially Netty, is even better than any other framework, with nearly 25,000 requests per second.

Through connection test, we can see that Netty and Undertow, especially Netty, can have excellent performance when using persistent connections.

D. *Results and Analysis of Resource Occupancy*

We did resource occupancy test to analyze the CPU and

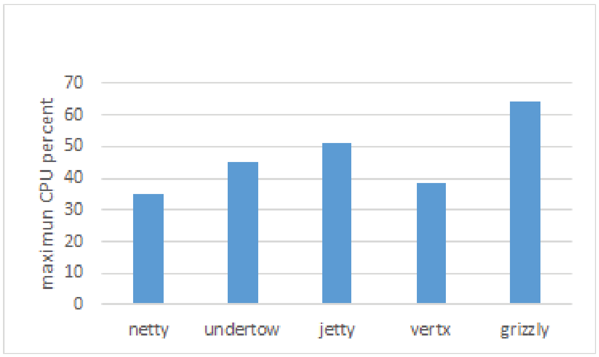

Fig. 6. Maximum CPU occupancy.

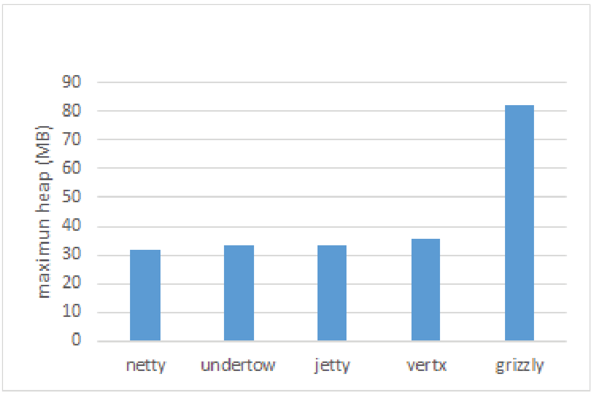

Fig. 7. Maximum memory heap occupancy.

memory resources taken up by these frameworks. The results are shown in Fig. 6 and Fig. 7. We can see that Netty and Vert.x take the least CPU resource, while Grizzly occupies the most CPU resource and memory heap.

VI. CONCLUSIONS

This paper presents a measurement study of the five WebSocket frameworks, i.e. Netty, Undertow, Vert.x, Grizzly and Jetty. To the best of our knowledge, this is the first paper to compare and evaluate the performance of these WebSocket frameworks from aspects of concurrency, flow, connection type and resource occupancy.

In short, compared with other WebSocket frameworks, Netty and Undertow perform better under highly concurrent situations, while Grizzly is more suitable for large flow conditions. When using persistent connections under concurrency level 100, Netty and Undertow, especially Netty, performs far better than any other framework. At a relatively high concurrency level, Vert.x and Undertow can handle most of the requests in time. Finally, Netty and Vert.x occupy less CPU resource, while Grizzly takes up more CPU and memory resources in comparison.

In our experiments, we follow the one-factor-at-a-time principle, and only consider the difference of the different WebSocket frameworks, not including the architecture design or the programming language. So it is the focus of our future work to study the performance effect in combination with architecture design and programming language. Besides, this paper only compares performance of the frameworks, and their security and extensibility also need further study. Finally, the above mentioned frameworks may be applied in decomposed wireless networks [22], and we are also interested in their performance in that scenario.

ACKNOWLEDGMENT

This work has been financially supported by Shenzhen Key Fundamental Research Projects (Grant No. JCYJ20170412150946024 and JCYJ20170412151008290) and Guangdong Pre-national project 2014GKXM054.

REFERENCES

[1] I. Fette and A. Melnikov, "The WebSocket Protocol", IETF RFC 6455, Dec. 2011.

[2] K. Lei, S. Zhong, F. Zhu, K. Xu and H Zhang, “A NDN IoT Content Distribution Model with Network Coding Enhanced Forwarding Strategy for 5G”, IEEE Transactions on Industrial Informatics, vol. PP, Dec. 2017, pp. 1-1.

[3] K. Lei, C. Hou, L. Li and K. Xu, “A RCP-Based Congestion Control Protocol in Named Data Networking”, Proc. IEEE International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, IEEE, Sep. 2015, pp. 538-541.

[4] http://netty.io/.

[5] http://undertow.io/.

[6] http://vertx.io/.

[7] https://javaee.github.io/grizzly/.

[8] http://www.eclipse.org/jetty/.

[9] D. Skvorc, M. Horvat and S. Srbljic, “Performance evaluation of Websocket protocol for implementation of full-duplex web streams”, Proc. International Convention on Information and Communication Technology, Electronics and Microelectronics, IEEE, May. 2014, pp. 1003-1008.

[10] G. Imre and G. Mezei, “Introduction to a WebSocket benchmarking infrastructure”, Proc. Zooming Innovation in Consumer Electronics International Conference, IEEE, Jun. 2016, pp. 84-87.

[11] A. Mulay, H. Ochiai and H. Esaki, “IoT WebSocket Connection Management Algorithm for Early Warning Earthquake Alert Applications”, Proc. 10th International Conference on Utility and Cloud Computing, ACM, Dec. 2017, pp. 189-194.

[12] H. Nakajima, M. Isshiki and Y. Takefuji, “Websocket proxy system for mobile devices”, 2nd Global Conference on Consumer Electronics, IEEE, 2013, pp. 315-317.

[13] J. Feng and A. Harwood, “BrowserCloud: A Personal Cloud for Browser Session Migration and Management”, Proc. 24th International Conference on World Wide Web, ACM, May. 2015, pp. 1491-1496.

[14] S. Zhang and S. Zhu, “Server structure based on netty framework for internet-based laboratory”, Proc. IEEE International Conference on Control and Automation, IEEE, Jun. 2013, pp. 538-541.

[15] Y. Zhang, L. Yu, Y. Li and X. Che, “Optimization Design Method of Communication Service System for Vehicle Remote Monitoring Based on Netty Pattern”, Proc. Chinese Automation Congress (CAC), IEEE, Oct. 2017, pp. 682-686.

[16] B. Lin and Y. Pu, “Jetty improves the performance of network management system based on TR069 protocol”, Proc. IEEE International Conference on Intelligent Computing and Intelligent Systems, IEEE, Oct. 2010, pp. 799-801.

[17] PP. Dhekale and N. Jichkar, “Efficient Data Search Using Map Reduce Framework”, Proc. 2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare, Coimbatore, 2016, pp. 1-4.

[18] VP. Ranganath, JK. Yu, J. Hatcliff and Robby, “Communication patterns for interconnecting and composing medical systems”, Proc. Engineering in Medicine and Biology Society, IEEE, Aug. 2015, pp. 1711-1716.

[19] http://www.apache.org/.

[20] http://visualvm.github.io/.

[21] R. Jain, “The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling”, John Wiley & Sons, Inc., New York, NY, 1991.

[22] L. Dai and B. Bai, “Optimal Decomposition for Large-Scale Infrastructure-Based Wireless Networks”, IEEE Transactions on Wireless Communications, vol. 16, Aug. 2017, pp. 4956-4969.