Asynchronous Publish/Subscribe Architecture over WebSocket for Building Real-time Web...

Article *in* Internetworking Indonesia Journal · December 2015

CITATIONS
0

READS
72

2 authors, including:

Bens Pardamean
Binus University
60 PUBLICATIONS **108** CITATIONS

SEE PROFILE

Asynchronous Publish/Subscribe Architecture Over WebSocket for Building Real-time Web Applications

Jastian Ganaputra and Bens Pardamean
Bina Nusantara University, Jakarta

***Abstract* — This study applied the Publish/Subscribe architectural pattern utilizing Amazon Simple Queue Service as a message broker to facilitate the Publish/Subscribe architecture on HTML5 featuring the WebSocket protocol for the transport layer. The application of the Publish/Subscribe pattern on WebSocket connections resolved the issue of lost messages on WebSocket connections. The framework performances were compared to WebSocket connections itself with regards to messages loss, response latency times, and bandwidth network out. The results revealed that the proposed framework in this study led to message loss reduction.**

***Index Terms* — WebSocket, HTML5, Publish/Subscribe, Message Broker**

I. INTRODUCTION

DURINGweb technology growth, the use of real-time data by web applications will increase in the coming years [1]. The need for real-time websites will subsequently increase; this includes the features of simple use-cases for real-time web application approaches in data display, statistics, dashboard monitoring, notifications, instant messaging, and multi-player gaming. Real-time website based on pushing information/messages to clients rather than a simultaneous request to server for updates can lead to a much different technical architecture. Building a real time web application requires a connection that provides two-way communications with minimum response latency time, which leads to a more responsive user interaction. The application must also have a server capable of handling message interchanges with clients.

A. WebSocket

This study's approach to building a real-time web application utilizes WebSocket, a protocol that provides full duplex communications channels over a single TCP connection. The WebSocket protocol by provides a standardized way for the server to send content to the browser without client solicitation as the trigger. Messages can also be passed bi-directionally between a browser and the server while maintaining an open connection. Additionally, some early adopters of WebSocket technology have raised concerns regarding WebSocket's compatibility with Enterprise Web Middleware Infrastructures (EWMI), such as proxy servers, load balancers, firewalls, message brokers, etc [1].

Relative to the HTTP protocol, WebSocket protocol provides better latency time [2]. The process of data transmission through WebSocket connection is not impervious towards data loss, especially with the increasing number of connections. When a WebSocket session fails, the protocol does not have the ability to resolve and define messages as those that have been received and those that have been lost [3]. Therefore, distributed applications and more complex logic processes among applications require a model that can manage the distribution of notifications/messages to clients based on constraints specified upon subscription.

B. Publish/Subscribe

The Publish/Subscribe pattern is a paradigm provides asynchronous communications for the distributed system class. Message Broker is an architecture that mediates communication among applications based on the Publish/Subscribe pattern for message validation, message transformation, and message routing. Publish/Subscribe systems facilitate users with the ability to express their interest in a subscription and subsequently receive notifications based on their interests for any publisher-generated events [4].

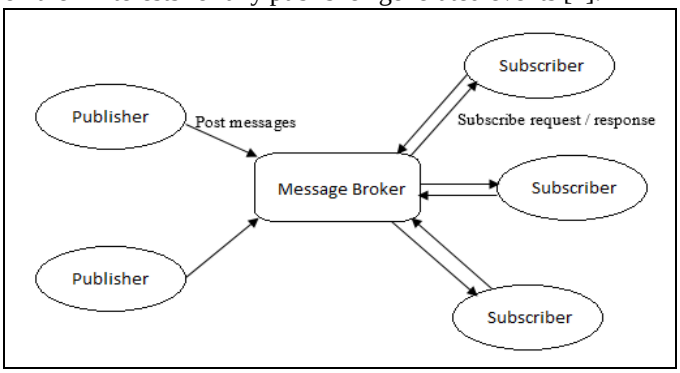

Fig.1 Publish/Subscribe Schema [5]

As shown in Figure 1, the publisher issues or post messages to the message broker. Subscribers must register with a specific message broker of that performs the task of relaying relevant messages from publishers to subscribers.

C. Problem Formulation

With an increase in the number of WebSocket connections, data loss becomes a confounding problem. However, there is no protocol that addresses and defines data loss during data transmission processes between server and clients.

D. Hypotheses

The hypotheses are stated as follows:

H1: Applying Publish/Subscribe pattern method reduces message loss during message transmission on WebSocket connections;

H2: Applying Publish/Subscribe pattern method increases the response latency time of WebSocket connections;

H3: Applying Publish/Subscribe pattern increases the bandwidth network out of WebSocket connections.

II. Related Works

Previous studies have proposed model solutions related to WebSocket connection and compare them to various methods used for real-time data communication. Through comparative analyses, WebSocket is found to be a protocol that provides efficient communications between Web Servers and Clients with the lowest response latency time relative to HTTP Polling, AJAX long polling, COMET, HTTP Streaming, and WebSocket [2,6]. Pimentel and Nickerson [7] compare one-way latency between a client and server using various methods such as polling and long polling with WebSocket using WindComm application.

Alamsi and Kuma [1] evaluate the performance of WebSocket, showing that WebSocket not only contributes less data transmission over the network but also significantly simplifies development by abstracting the sending and receiving of data. Additionally, the Publish/Subscribe architecture provides better scalability [8] and reliability to avoid message congestion, conflict, and loss [9].

Khanaferov identifies the best approach for developing asynchronous web applications [10]. The study proposes an asynchronous Publish/Subscribe framework solution by using Java Message Service as a Message Broker. A similar approach is also used by Roussele [11]. Khanaferov implements the Java Applet plug-in towards the asynchronous server push transport layer [10]. The resulting framework consists of a Publish/Subscribe pattern to facilitate an asynchronous message passing interface. However, data loss due to concurrent increase of user request remains unresolved, leading to this study's hypotheses and problem formulations.

III. Methodology

This study was quantitative in nature with the intent to determine web application architecture's performance and effectiveness in providing proposed solutions for building real-time, web-based applications. Measurements included the observed difference in response latency time between the proposed solution of WebSocket with applied Publish/Subscribe model and the WebSocket connection on its own (without the Publish/Subscribe model). The resulting model can serve as a guide for web developers in designing a real-time, web-based application.

A. Study Design

To conduct the experiment, two web application servers were built for transmitting messages. The first experimental application server was built over WebSocket without implementing Publish/Subscribe pattern method while the second experimental application server was implemented with the Publish/Subscribe pattern method. The technology used for both of applications was identical since WebSocket technology is a feature in HTML5. This study used HTML5 for content structuring and presentation for webs that support WebSocket connection. Javascript code was used for the client-side programming and connection establishment. The server-side programming utilized Microsoft ASP.NET, a Microsoft-supported technology designed for web development to produce dynamic web pages with easy development and deployment. Microsoft ASP.NET also possessed many features, tools, and classes that reduce coding efforts. AWS SQS (Amazon Web Service Simple Queue Service) was used as a message broker to implement the Publish/Subscribe pattern.

Both experimental scenarios were conducted with 30 testing iterations. Every set of iteration was divided into the following:

- 50 messages from 50 concurrent users;
- 100 messages from 100 concurrent users;
- 500 messages from 500 concurrent users;
- 1,000 messages from 1,000 concurrent users;
- 5,000 messages from 5,000 concurrent users;
- 10,000 messages from 10,000 concurrent users.

To simulate up to 10,000 users request, a third-party tool, Loader.io was used. Loader.io is an open-source, web-based application that allows clients to stress-test applications.

The output results from both experimental scenarios were response latency time, messages sent with responses, and bandwidth network out. The output data were then processed and statistically analyzed.

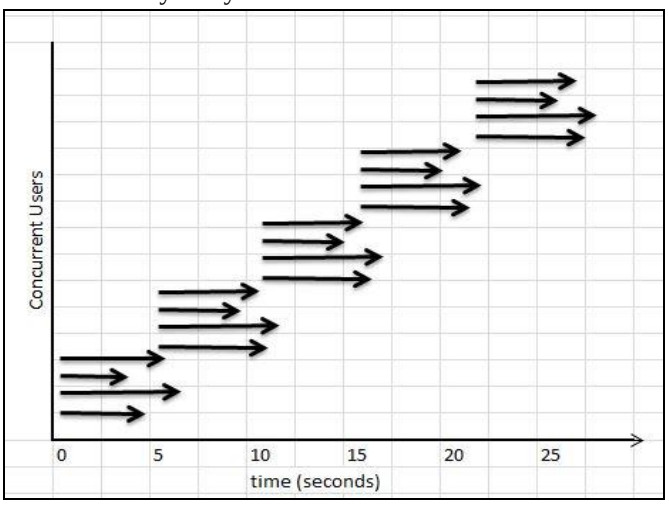

Fig.2 Concurrent Users Test Diagram

Figure 2 shows the method by which the test worked. Each arrow represents a client making requests to the web server. The length of the arrow represents the response time for a client to access the server in a given test.

B. Testing Environment

The testing environment was run in the cloud computing platform, Amazon Elastic Compute Cloud (AWS EC2). The server host operating system assigned dedicated CPU cores and was configured with the following specifications:

- Operating System: Windows Server 2012 R2 Standard 64-bit
- Processor: Intel® Xeon® CPU E5-2670 v2 @ 2.50GHz
- Memory: 1024 MB RAM

Application Server Environment:

- Internet Information Services 8 Application Server Manager
- .Net Framework 4.5

Testing tools:

- Internet Explorer 11
- Loader.io
- SolarWinds Real-Time Bandwidth Monitor

C. Data Collection

To collect the data for analysis, simulations were conducted by establishing environments in web application servers, one with Publish/Subscribe pattern application on WebSocket and one without the Publish/Subscribe pattern application. For both environments, the tests were performed by sending messages from clients. The output results of the tests were the data used for analysis.

D. Analytical Method

The study used independent t-test statistical analysis. The T-test analysis compared the response latency time between the two experimental groups. The same test was used to view the bandwidth network out as influenced by the application of Publish/Subscribe pattern on WebSocket connection

IV. Result

A. Data Loss

Data Loss was analyzed by comparing total response counts of messages sent to server and success response counts.

Table 1 shows the result of response success per response sent with 30 iterations for different concurrent users. The success rate was calculated from the total of success response divided by the total request sent. The data loss rate was calculated from 100% minus success rate in percent.

Based on Table 1, it can be seen that data loss is non-existent in the first two cases of Group 1 (without Publish/Subcribe). For the third case with 500 concurrent connections, a 0.74% rate of data loss occurred. Furthermore, under higher load connections (with 1,000, 5,000, and 10,000 concurrent connections), the data loss ratio increased concurrently. For Group 2 (with Publish/Subcribe), data losses occurred once the concurrent users connection reached 5,000, though the success rate is higher than that of Group 1. For the 10,000 connections case, Group 2 data loss was reduced by over 7% relative to Group 1. Despite this improvement, Group 2 still experienced data loss. However, the lost messages can be re-queued to AWS SQS message broker, delegating the task of handling data loss problem to the message broker for resolution.

Since Group 2 with the application of Publish/Subscribe pattern has no messages loss for 50, 100, 500, and 1,000 concurrent users, independent t-test method was used to compare the two groups for 5,000 concurrent users connections and 10,000 concurrent users connections, as shown in Table 2.

TABLE I
DATA LOSS COMPARISON BETWEEN GROUPS

Concurrent Users	Group 1 (without Pub/Sub)		Group 2 (with Pub/Sub)	
	Success	Loss	Success	Loss
50	100%	0%	100%	0%
100	100%	0%	100%	0%
500	99.24%	0.74%	100%	0%
1,000	98.86%	1.14%	100%	0%
5,000	94.30%	5.70%	99.02%	0.98%
10,000	91.71%	8.29%	99.15%	0.85%

With p-value of less than alpha-level of 0.05 for 5,000 concurrent users and 10,000 concurrent users, Hypothesis H1 was accepted. In other words, applying Publish/Subscribe pattern to WebSocket significantly reduced data loss during message transmission processes on WebSocket connections.

B. Response Latency Times

Response latency analysis was calculated by measuring the difference in time stamps between the messages was sent and when the client received the message.

Figure 3 shows the sequence of 'get timestamp' to calculate the response latency time. Timestamp was retrieved before sending messages to server then retrieved again when a subscribing client received the messages.

Table 3 shows the result of the response latency t-test for

TABLE II
DATA LOSS T-TEST RESULT BETWEEN GROUPS

Concurrent Users	Group 1 (without Publish/Subscribe)			Group 2 (with Publish/Subscribe)			p-value ($\alpha = 0.05$)
	N	Mean (Total Data Lost)	Std. Deviation	N	Mean (Total Data Lost)	Std. Deviation	
5,000	30	285	2.45	30	49	1.73	< 0.05*
10,000	30	829.20	18.62	30	85.20	2.23	< 0.05*

*significant at $p < 0.05$

Group 1 and Group 2 for each test case of 50, 100, 500, 1,000, 5,000, and 10,000 concurrent users with each test case conducted in 30 iterations.

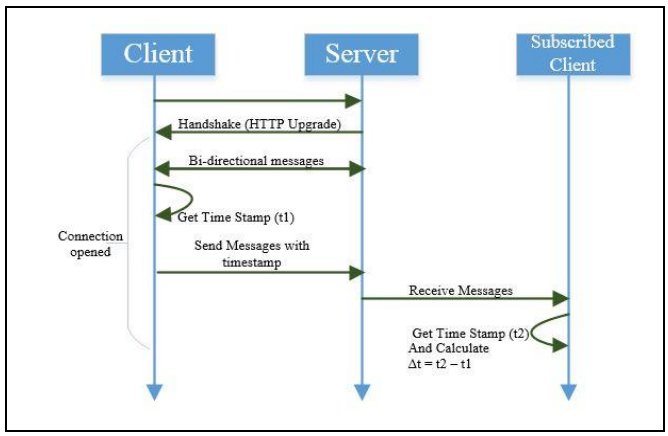

Fig.3 Get Time Stamp Sequence Diagram

For the 50 concurrent users case, Group 2 with Publish/Subscribe pattern has a higher response latency mean value with an approximate difference of 1.17 milliseconds. However, the standard deviation of Group 2 is lower than standard deviation of Group 1, indicating that the former a more stable response latency time across the 30 iterations. The p-value of 0.06168 is higher than the 0.05 alpha level. Thus, there is no significant difference in response latency times between the two groups for 50 concurrent users connections.

For 100 concurrent users case, Group 2 also has a higher response latency mean value but a lower standard deviation than Group 1. The p-value of 0.05681 is greater than 0.05 alpha level, rendering an interpretation of no significant difference of response latency times between Group 1 and Group 2 for 100 concurrent users connections.

For each of 500, 1,000, 5,000, and 10,000 concurrent users cases, Group 2 has the same pattern of having higher response latency mean values but lower standard deviations than Group 1. The p-value for each case is lower than the 0.05 alpha level. Therefore, there is a significance difference in response latency times for 500, 1,000, 5,000, and 10,000 concurrent users connections.

The t-test results concluded that:

- Applying Publish/Subscribe pattern does not significantly increase response latency times of WebSocket connections for 50 and 100 concurrent users connections, leading to the rejection of Hypothesis H2 for 50 and 100 concurrent users connections.
- Applying Publish/Subscribe pattern results in significant increases in response latency times of WebSocket connection for 500, 1,000, 5,000, and 10,000 concurrent users connections, leading to the acceptance of Hypothesis H2 for 500 or higher (up to 10,000) concurrent users connections.

C. Bandwidth Network Out

Bandwidth analysis was performed through the measurements of the bandwidth network out average when requests from client to server application until the subscribed clients get response from server.

Independent t-test analysis was also used to view how the bandwidth network out was influenced by the application of the publish/subscribe pattern on WebSocket connection.

TABLE III
DATA LOSS COMPARISON BETWEEN GROUPS

Concurrent Users	Group 1 (without Publish/Subscribe)			Group 2 (with Publish/Subscribe)			p-value (α = 0.05)
	N	Mean (ms)	Std. Deviation	N	Mean (ms)	Std. Deviation	
50	30	36.83	2.89	30	38	1.70	0.06168
100	30	87.67	3.97	30	89.37	2.68	0.05681
500	30	292.77	6.04	30	300.20	3.4	2.21×10^{-7}*
1,000	30	489.43	7.76	30	497.50	4.53	7.56×10^{-6}*
5,000	30	678.87	11.27	30	688.97	4.87	3.25×10^{-5}*
10,000	30	793.63	10.50	30	805.30	5.48	1.32×10^{-6}*

*significant at $p < 0.05$

TABLE IV
BANDWIDTH NETWORK OUT COMPARISON BETWEEN GROUPS

Concurrent Users	Group 1 (without Publish/Subscribe)			Group 2 (with Publish/Subscribe)			p-value (α = 0.05)
	N	Mean (Kbps)	Std. Deviation	N	Mean (Kbps)	Std. Deviation	
50	30	10.52	0.23	30	10.51	0.21	0.9392
100	30	18.92	0.61	30	19.03	0.51	0.4809
500	30	44.47	1.16	30	44.80	1.23	0.2944
1,000	30	70.24	0.77	30	70.80	0.77	0.0074*
5,000	30	286.58	2.21	30	298.44	3.11	3.73×10^{-24}*
10,000	30	307.25	4.61	30	332.44	4.94	3.54×10^{-28}*

*significant at $p < 0.05$

Table 4 shows the result of bandwidth consumption for Group 1 and Group 2 with each group tested using 50, 100, 500, 1,000, 5,000, and 10,000 concurrent users request with 30 iterations for each case. Bandwidth network out usage increased with higher concurrent users connection. Bandwidth gap between Group 1 and Group 2 also increased with concurrent users connection. For the 50 concurrent users connection, the mean bandwidth gap between Group 1 and Group 2 is 0.01 Kbps. For the last case with 10,000 concurrent users connection, the gap was 25.19 Kbps.

For 50, 100, and 500 concurrent users connections, between the two groups, differences in mean value and standard deviation were miniscule. Furthermore, each of p-value was higher than the 0.05 alpha level. Thus, there was no significant difference in bandwidth network out consumption between Group 1 and Group 2 for these cases. In contrast, for 1,000, 5,000, and 10,000 concurrent users connections, each p-value was lower than the 0.05 alpha level. This indicated a significant difference in bandwidth network out consumption between Group 1 and Group 2 for these cases.

From t-test result can be concluded that

• Applying Publish/Subscribe pattern did not significantly increase bandwidth network out of WebSocket connections for 50, 100, and 500 concurrent users connections. Hypothesis H3 was rejected for these cases.

• Applying Publish/Subscribe pattern led to significant increases in bandwidth network out of WebSocket connection for 1,000, 5,000, and 10,000 concurrent users connections. Hypothesis H3 was accepted for these cases.

V. CONCLUSION

The Publish/Subscribe pattern model was implemented on WebSocket connections to compare message loss, response latency time, and bandwidth network out between implemented Publish/Subscribe on WebSocket and WebSocket itself. The tests had two groups; the first with applied Publish/Subscribe onto WebSocket and the second with is WebSocket connection on its own. Each group performed 30 iterations for different numbers of client connections. The result showed that the group with Publish/Subscribe pattern has a decrease in messages lost by over 7% but at the expense of higher bandwidth network out consumption and higher response latency time, particularly at the higher end of client connection numbers (between 1,000-10,000 concurrent users).

REFERENCES

[1] Alamsi, A., & Kuma, Y. *Evaluation of WebSocket Communication in Enterprise Architecture.* Available: http://presentation.amiralmasi.com/ Evaluation of WebSocket Communication in Enterprise Architecture_ AmirAlmasi_YohanesKuma.pdf, 2013.

[2] Liu, Q., & Sun, X. Research of Web Real-Time Communication Based on Web Socket. *Int. J. Communication, Network and System Sciences*, Vol. 5, No 12, 2012, pp. 797-801.

[3] Hapner, E. M., & Suconic, C. The MessageBrokerWebSocket Subprotocol. Available: http://tools.ietf.org/search/draft-hapner-hybi-messagebroker-subprotocol-01, 2011.

[4] Li, W. A Content-based Publish/Subscribe Framework over Structured Peer-to-Peer Networks, Master thesis, computer science, University of British Columbia, Vancouver, Canada. 2008. Available: http://www.cs.ubc.ca/grads/resources/thesis/Nov08/Li_Wei.pdf.

[5] Hauer, J. H., Handziski, V., Köpke, A., Willig, A., & Wolisz, A. A Component Framework for Content-based Publish/Subscribe in Sensor Networks, in *Proc. of 5th European Conference on Wireless Sensor Networks (EWSN 2008)*, Bologna, Italy. 2008.

[6] Rakhunde, S. M. Real Time Data Communication over Full Duplex Network Using Websocket.*IOSR Journal of Computer Science*, Vol. 5, 2014, pp. 15-19.

[7] Pimentel, V., & Nickerson, B. G. Communicating and Displaying Real-time Data with WebSocket. *IEEE Internet Computing*, Vol. 16, No. 4, 2012, pp. 45-53.

[8] Mühl, G. Large-Scale Content-Based Publish/Subscribe System. 2002. Available under Simple publication rights for ULB.

[9] Yan, Z. L., & Dai, M. A. Realtime Group Communication Architecture Based on WebSocket. *International Journal of Computer and Communication Engineering*, Vol. 1, No. 4, 2012, pp. 408-411.

[10] Khanaferov, D. Publish-Subscribe Architecture for Building Non-Polling Asynchronous Web Applications, Master thesis, computer science, California State University, Northridge, US. 2013. Available: http://scholarworks.csun.edu/handle/10211.2/3233.

[11] Rousselle, P. Implementing the JMS Publish/Subscribe API. *Dr. Dobb's Journal*, Vol. 27, No. 4, 2002, pp. 28.

Jastian Ganaputra is a graduate student in Information Technology at BINUS Graduate Program, Bina Nusantara University, Jakarta, Indonesia.

Bens Pardamean is an Associate Professor of Computer Science and Director of Bioinformatics & Data Science Research Center, Bina Nusantara University, Jakarta, Indonesia.