Research on Server Push Methods in Web Browser based Instant Messaging Applications

Kai Shuang
State Key Laboratory of Network & Switching Technology, Beijing, China
Email: shuangk@ bupt.edu.cn

Kai Feng
State Key Laboratory of Network & Switching Technology, Beijing, China
Email: fk187andhk@gmail.com

Abstract—**Nowadays, web browser based instant messaging applications are becoming more and more popular. In these applications, the function of server push has caused lots of problems because of the shortages of HTTP protocol. HTTP Polling, HTTP Long Polling and HTTP Streaming are the most popular solutions for server push methods, and with the rapid development of HTML5 standard and other Web technologies, WebSocket based real-time applications seem to be very promising. This paper introduces different methods of server push, and has an experiment to compare their delays and unnecessary connection costs. At the end of this paper, a conclusion is drawn to show the usage scenarios of different methods and have a summary for the using of server push methods in Web browser based real-time application.**

Index Terms—**Server Push, Web Browser, Instant Messaging Application, HTTP, WebSocket**

I. Introduction

With the development of web browsers and web based technologies, many applications are developed by them on desktops and mobile devices. The web based instant messaging applications are becoming more and more popular, like Web QQ [1], SparkWeb [2], Gtalk on Gmail [3] and so on. Because web browsers have the advantages of cross-platform easily and no explicit installation, a well designed and developed web application can be run on both PC desktop and smart phone perfectly only if there are web browsers on them.

The communicating application is always requiring the support of real-time and low delay as possible, which is, however, one weakness for web browser based technologies. Because the base communicating protocol of web browser is HTTP [4]. HTTP is a request/response protocol which means that server cannot push messages to browser if there are no requests. HTTP has become a focused study in the area of web technologies research, especially its performance evaluation for data transferring as described in [5] and its security issues in [6]. So, how to make web browser based communicating applications get new messages or notifications at first time is one of the major problems for web developers. And there are many Flash plugin and Ajax [7] based solutions have been invented to achieve the goal of implementing the server push function. Besides, the new standard of HTML is also changing the situation.

Being the representative of new web based technologies, HTML5 [8] is changing the way of web application development. HTML5 is the latest revision of the HTML standard, a language for structuring and presenting content for the World Wide Web. New features have been added to make it easier and faster for developing web browser based communicating applications.

As mentioned, many kinds of server push methods have been invented. In [9], a framework based on the object-oriented design and the approach of push web server for physical object interactions has used the technology of HTTP based message exchanging. Flash plugin based server push implementation is one of the methods, but not all browsers support it, like the Safari on iPhone [10]. In this paper, study has been conducted on web browsers in which only native support features like HTTP and HTML5 are used. In order to have a summary and comparision for different methods of server push in web based instant messaging applications, an experiment has been designed and implemented.

In section II, the technical background of different server push methods and the requirements for web based instant messaging applications are introduced. The experiment for comparisons of different solutions including architecture and results is showed in Section III. We also draw conclusions and plan future works in Section IV.

II. Background and Requirements

Instant messaging (IM) is a form of communication over the Internet, which offers quick transmission of text-based messages from sender to receiver. In push mode between two or more people using personal computers or other devices, along with shared clients, instant messaging basically offers real-time direct written language-based online chat [11]. In the definition of instant messaging, there are three key features: text-based, bi-directionally exchanged and real-time.

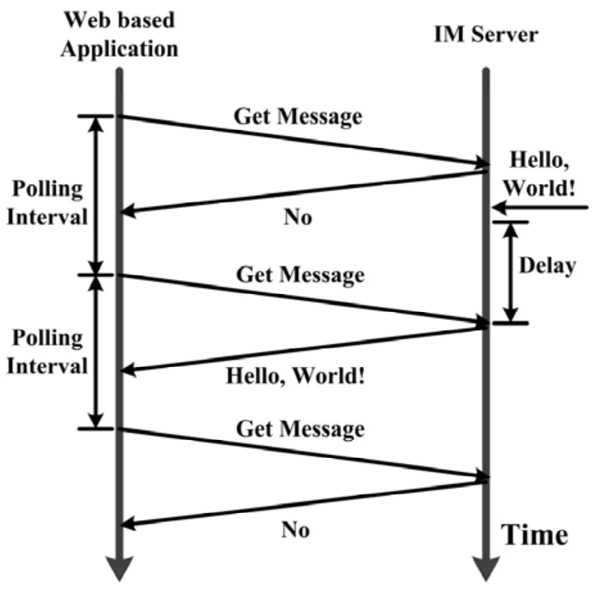

Figure 1. Workflow of HTTP Polling

For the scenario of web browser, text-based is easy to implement. But both bi-directionally and real-time text-based information transportations need the server push function. If the server does not support the server push mechanism, the new messages cannot be sent to client directly. Obviously, it will add unnecessary latencies.

To deal with the problems, some web developers come up with lots of HTTP based communicating mechanisms to achieve server push like Polling, Bosh [12], Comet [13] and HTML iframe tag [14]. In essence, all of them are based on HTTP Polling, Long Polling and Streaming. On the other hand, HTML5 brings a new kind of communicating mechanism which is called WebSocket [15] in IETF. Their working details are as below.

A. HTTP Polling

HTTP Polling is the easiest way to simulate the implementation of server push. HTTP is a request/response based protocol, so the server cannot send data to browser or HTTP client directly if the client side did not initialize any requests. HTTP Polling is using the request/response model to implement the server push.

In the HTTP Polling, the web browser based instant messaging application will send a HTTP request to server for querying whether there are any new messages coming every other a few seconds. The waiting time between two requests can be called the Polling Interval; the time between new messages arrived to server and been taken away by application can be called the Delay.

As Fig. 1 shows, the web browser based instant messaging application should send a HTTP request every polling interval to get new messages. If there are no updation on server side, the request will be responed immediately without any messages; if there are new messages coming during the polling interval, the response will take these messages to application directly.

HTTP Polling has a big problem that it cannot ensure the real-time characteristic, if the messages come to server during the client polling interval, the server cannot send them to application directly because there are no requests. Then these messages have to be stored on server side for waiting requests, exactly thus causes the delay. And of course, the delay time is less than or equal to the polling interval, so if the polling interval of application is very small, the delay can be smaller too. But it will bring about another problem that many useless requests will be sent to server which will cause large bandwidth consumption and increase the server stress.

Obviously, this is a good solution if the exact interval of message delivery is known, because the application can synchronize the client request to occur only when information is available on the server. However, real-time data is often not that predictable, making unnecessary requests inevitable and as a result, many connections are opened and closed needlessly in low-message-rate situations.

HTTP Polling also has advantages. The biggest one is that it is very easy to realize both on application and server side. Server does not need to do any extensions to support HTTP Polling, and application just needs to start a timer to send one request every interval.

B. HTTP Long Polling

In order to get new messages in real-time, web developers come up with a strategy called HTTP Long Polling. Compared with HTTP Polling, HTTP Long Polling has many advantages like fewer HTTP requests and nearly real-time messages.

HTTP Long Polling needs the support of server side, when one request coming, and there are no new messages the server would not response immediately but keep the request open for a set of periods. The server will waits a few seconds until messages coming or connection timeout. After receives the response, application in web browser will initialize a new request.

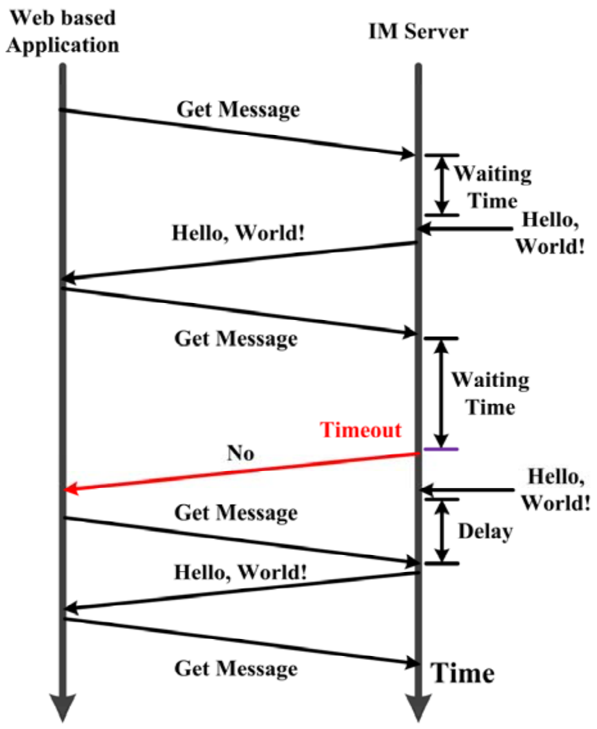

Figure 2. Workflow of HTTP Long Polling

As Fig. 2 shows, the server side needs to support the HTTP Long Polling mechanism. After receiving the HTTP request, the server will check whether there are new messages. If there are no updates, the server will block this request a few seconds which means that no response should be sent immediately; if there are new messages coming during the server waiting time, the server will send response carrying the new messages to application directly. The application then initializes a new request immediately. But if there are still no updates over the waiting time until timeout, the server will response application with an empty respond, and the application still needs to start a new request immediately.

The HTTP Long Polling can get messages in real-time, but there is still delay existing. If there are new messages coming when the server has responded the request and still has no request coming, the server has to store the messages and wait for the next request. It will cause the delay, but the good thing is that the delay is very short because it depends on the HTTP message tranferring time. The delay time is less than or equal to twice transferring time, and of course it is very small. So the HTTP Long Polling can guarantee nearly real-time messages pushing.

There are still some disadvantages about the HTTP Long Polling. The server side web server must support the Long Polling scheme, and the HTTP Long Polling will always occupy on TCP connection which is called HTTP Persistant Connection, the number of HTTP connections at one time between browser and server with same hostname are limited [16], which means that if the Long Polling takes one connection, it may have influences on the web surfing to the same hostname, but of course it is a slight influence. Besides, it is important to be aware of the server side technical limitations of Long Polling. Since connections might be kept alive for long durations of time, the web server must be capable of handling such large numbers of simultaneous connections. Meanwhile, when there is a high message volume, Long Polling does not provide any substantial performance improvements over traditional polling. In fact, it could be worse, because the Long Polling might spin out of control into an unthrottled, continuous loop of immediate polls.

HTTP Long Polling is the most popular implementation of server push mechanism. Most of mature instant messaging applications in the world are using it or its extensions like Web QQ and Gtalk on Gmail.

C. HTTP Streaming

Although HTTP Long Polling can get new messages in real-time, it still needs lots of HTTP requests or connections to get the responses which means that Long Polling will become Polling if there is a high message volume. HTTP Streaming is an implementation that can reduce the number of HTTP connections and get real-time data at the same time.

In HTTP Streaming, browser sends a complete HTTP request, but the server sends and maintains an open response which is continuously updated and kept open indefinitely. The response is then updated whenever a message is ready to be sent, but the server never signals to complete the response, thus keeping the connection open to deliver future messages.

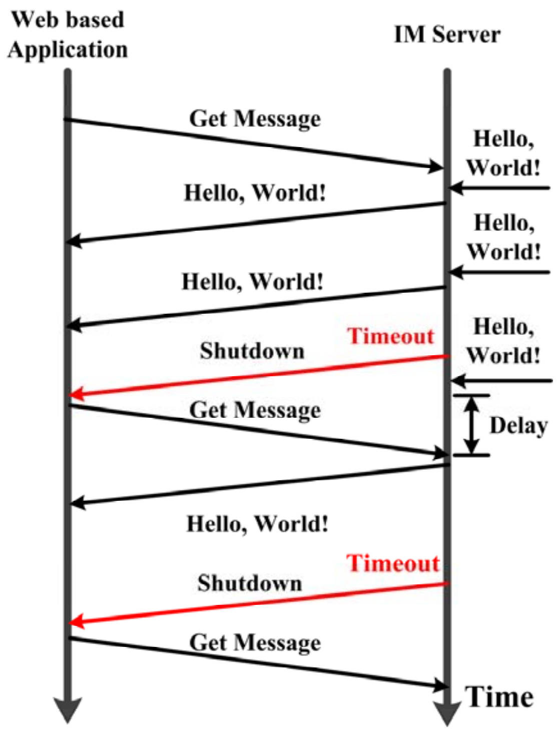

Figure 3. Workflow of HTTP Streaming

As Fig. 3 shows, the server side will keep the response open until HTTP connection timeout. During the open time, if there are new messages coming, server can send them to application directly as one part of response stream. The application should be aware of the message streaming and combine them as one entire message. HTTP Streaming needs the support of both client and server sides, and it still has the scenario of delay. If there are new messages arriving server after the connection timeout and before the next request coming, it occurs the delay. Obviously, the delay time is very small because it is less than or equal twice of transferring time. HTTP Streaming not only can get real-time data nearly at the first time, but also reduce the number of HTTP requests or connections to a great extent.

HTTP Streaming still has many disadvantages, since HTTP Streaming is still encapsulated in HTTP, intervening firewalls and proxy servers may choose to buffer the response, increasing the latency of the message delivery. Under this circumstance, many applications may choose to fall back to HTTP Long Polling mode in case a buffering proxy server is detected. Besides, the implementation of HTTP Streaming in client side still has a problem of browser compatibility especially in Microsoft Internet Explorer, because the HTTP Streaming can be implemented by both HTML IFrame tag and Ajax. If the application uses the IFrame tag, it will cause phantom click and throbber of doom [17] and have a big influence on user experience seriously.

Above all, all the HTTP based server push methods have one common shortcoming: HTTP request contains

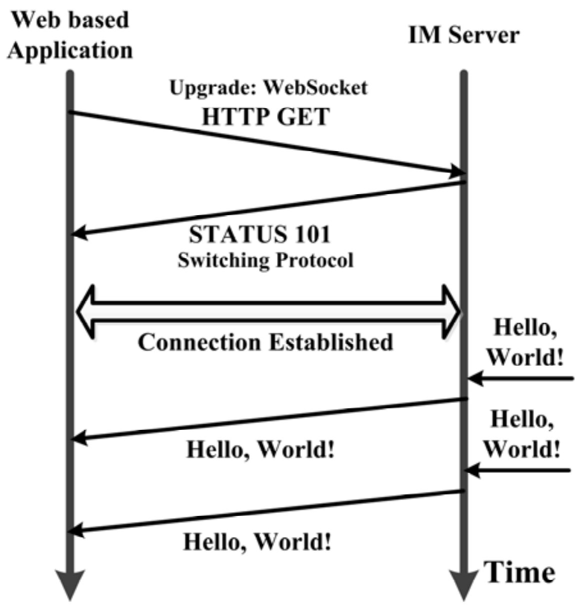

Figure 4. Workflow of WebSocket

lots of additional, unnecessary header data and introduce latency, but full-duplex connectivity requires more than just the downstream connection from server to client. Obviously, HTTP wasn't designed for real-time, full-duplex communication. On the other hand, it is no doubt that different implementations have their usage scenario and environment. At the present time, most of web browser based instant messaging applications are still using HTTP Polling, Long Polling and Streaming.

D. WebSocket

WebSocket is a technology providing for bi-directional, full-duplex communication channels, over a single TCP socket in web browsers and web servers. In other words, WebSocket defines a full-duplex single socket connection over which messages can be sent between client and server. And of course, WebSocket needs the support of both server and browser sides with WebSocket stack and APIs.

As Fig. 4 shows, to establish a WebSocket connection, the browser and server should upgrade from the HTTP protocol to the WebSocket protocol during their initial handshake. It means that the WebSocket can reuse the port of HTTP connection.

The handshake HTTP message is a normal GET request of a field with name Upgrade whose value is WebSocket. If the server side supports the WebSocket protocol, it will return a response immediately with a 101

```
GET ws://www.websocket.org/test HTTP/1.1
Origin: http://websocket.org
Connection: Upgrade
Host: www.websocket.org
Sec-WebSocket-Key: uRovscZjNol/umbTt5uk
Upgrade: websocket
Sec-WebSocket-Version: 13
```

```
HTTP/1.1 101 WebSocket Protocol Handshake
Date: Wed, 10 Feb 2013 18:30:00 UTC
Connection: Upgrade
Server: Jetty Server
Upgrade: WebSocket
Access-Control-Allow-Origin:
http://www.websocket.org
Access-Control-Allow-Credentials: true
Sec-WebSocket-Accept:
rLHCkw/SKsO9GAH/ZSFhBATDKrU=
Access-Control-Allow-Headers: content-type
```

status code which means the protocol switching. After the response, the HTTP session is over and its carrier TCP connection will be used by the WebSocket protocol. From this point, WebSocket is a supplement of HTTP for bi-directional, full-duplex communication. The initial handshake HTTP messages are given above.

Once established, WebSocket data frames can be sent back and forth between the client and the server in full-duplex mode, both the server and browser sides can send and receive messages at any time. The delay is only occurs in transporting latency and the header of WebSocket is much more simple than HTTP. In the case of WebSocket text frames, each frame starts with a 0x00 byte, ends with a 0xFF byte, and contains UTF-8 data in between. Of course it can improve the bandwidth utilization of server side. The HTML5 WebSocket API interface is listed in Appendix A.

WebSocket also has disadvantages because not all browsers and web servers support it [18], but there are more and more network utilities are updating to meet the requirements. And in other words, one WebSocket connection means that one TCP connection is at work all the time between browser and server. It is no doubt that it may have an influence on the concurrent performance of web server. In the scenario of browser based instant messaging applications on different mobile devices, WebSocket is not a good choose because it is difficult to maintain a stable TCP connection if the device keeps moving.

From above, there are mainly four kinds of server push methods and basically all of them have relationship with HTTP protocol. The four methods have their advantages , disadvantages and different usage scenarios. An experiment on these four implementations will be designed to test their availability, usability and best using scenario. Meanwhile, a comprehensive comparison has been made between them in order to provide a reference for the web browser based instant message applications.

III. EXPERIMENT

In order to study the features and characteristics of different server push methods, we designed an experiment for testing the communication mechanisms used by web browser based instant messaging applications.

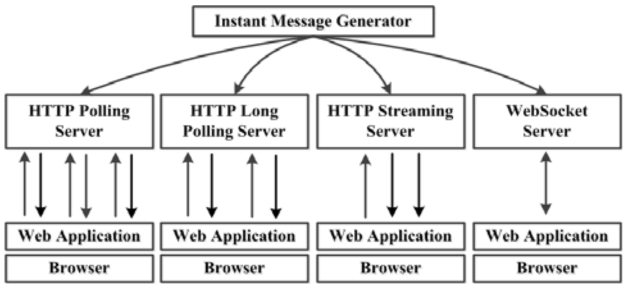

Figure 5. Experiment architecture

A. Architecture

As Fig. 5 shows, the architecture of experiment includes three parts: instant message generator, servers supporting different push implementation and browser based web applications.

Instant Message Generator: The instant message generator is a server generating messages with Poisson distribution and sends these messages to different pushing server randomly. Because the instant messages come to instant message server according to the Poison distribution, the generator should simulate the realistic scenario and try to make messages that mach the conditions.

Web Servers: The web servers are supporting different kinds of server push implementations. All of them have their message queue for storing and managing the messages to be pushed to the client side. When one message is received by server, it should be marked by one timestamp in order to calculate the delay. When the client requests coming, the servers will get all of the stored messages out and put them in the response, then clear the message queue.

Web Applications in Browser: The web applications will get the instant messages and calculate the delay includes the transporting latency. Because every message contains the timestamp which records the time when it is received by server, the application can substract the timestamp with the received time to get the sum of the transporting latency and delay. Both the browser and server sides can configure some parameters to adjust the server push implementation up to the communication mechanism.

B. Implementation

First, some terminologies should be defined in order to describe the implementation and result more convenient:

Delay: The time difference between one message arrived at web application and it arrived web server, and its value includes the transporting latency between client and server. This parameter is the most important value we focused on.

Transport Time: Transporting data on wire still needs to cost time, but because the experiment is under the scenario of LAN and message contents are too short to be transferred, so its value can be ignored.

Interval: Used by HTTP Polling mode to describe the time between two HTTP requests. In theory, with the increase of interval value, the delay value is decreasing.

Timeout: Every web server has its max time for holding one HTTP connection, this parameter can be configured. But for better user experience without the server initiatives shutdown the HTTP connection, a timeout value is set which means that if timeout event occurs, the server should close this HTTP connection and wait for the opening request initialized by browser.

Sleep Time: The web server cannot check the message queue all the time because it will cause the block of server, so this parameter is configured to make the thread sleep a few milliseconds which means that the server will query the message queue every Sleep Time. This parameter is very important to HTTP Long Polling and HTTP Streaming but has no influence on HTTP Polling. Because under the HTTP Polling, web server just needs to query when one request has come. From the perspective of a computer, the cost of *Interval* and *Sleep Time* are much more different. *Interval* means the time between two HTTP requests, *Sleep Time* represents the internal time between two database queries. If the value of *Interval* and *Sleep Time* are equal, it is no doubt that the cost of *Interval* is bigger because the server has to parse the HTTP request and construct the response everytime. In general, the value of *Sleep Time* is smaller than *Interval*.

Besides, the message generator can generate messages which are according the Poisson distribution. The generator will send these messages to different servers at the first time and servers will record the time of messages arriving. When the web applications get these messages, they will record current time and use it to minus the time of messages arriving to server which is contained in the message content. And in this experiment, the applications and servers are running on the same machine so that the transporting delays are ignored.

$$Delay = T_{application} - T_{server} - T_{transport} \qquad (1)$$

$$T_{transport} \approx 0 \qquad (2)$$

As in (1) and (2), *Delay* value equals to the difference of the time one message arrived at application and it arrived at the server side, the value of transporting time is approximately equal to 0.

$$Delay_{Polling} \le Interval + T_{connection} . \qquad (3)$$

With the definition of *Delay*, we can conclude that the *Delay* will always less than or equal to the *Interval* in the HTTP Polling mode, as in (3). The connection establishment time can be ignored.

$$Delay_{LongPolling} \le SleepTime + T_{connection} . \qquad (4)$$

$$Delay_{Strea\min g} \le SleepTime + T_{connection} . \qquad (5)$$

For the HTTP Long Polling and Streaming mode in (4) and (5), the value of *Delay* is always less than the *Sleep*

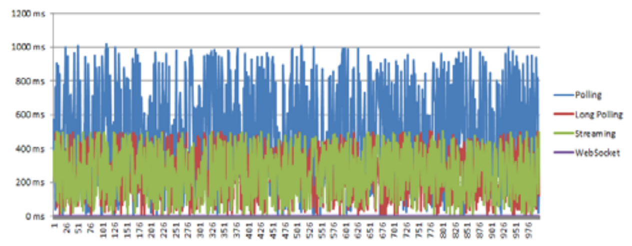

Figure 6. Experiment architecture

Time. The connection establishment time still can be ignored as in (6).

$$Delay_{WebSocket} = T_{transport} \approx 0. \qquad (6)$$

And for the WebSocket, because messages can be sent to web application as soon as the server receives new message. Then the parameter *Delay* of WebSocket is up to the transporting time which can be ignored as in (6).

From above, it is no doubt that the *Delay* of WebSocket is the smallest, and the *Delay* of HTTP Polling and HTTP Long Polling is much more similar if the parameter *Interval* and *Sleep Time* are same. For the HTTP Streaming, the most important variable which can have influences on *Delay* is just the *Sleep Time*, the HTTP connection establish time also can affect its *Delay* but is limited by the number of HTTP requests.

Except for the *Delay*, we still pay much attention on the number of HTTP requests or connections. The number of HTTP connections have a large influence on the performance of web servers because it will cost lots of time for dealing with HTTP request especially in the scenario of web server is stressing. In general, HTTP Polling mode needs much more requests than other server push methods, WebSocket just needs one HTTP request to establish the connection.

For the implementation, we use the Apache HTTP Componetns HttpCore 4.2 [19] as the basic protocol stack. The message generator and different servers are developed by Java language and running on a Ubuntu system. All the web applications are running on Mozilla Firefox 19.0 [20] with the operating system Windows 7.

In this experiment, the *Interval* parameter which is used by HTTP Polling equals 1000ms, the *Timeout* parameter equals 1 minute and the *Sleep Time* is 500ms , they are both used in the scenarioes of HTTP Long Polling and HTTP Streaming. The value of connection establishement and transport time are approximately equal to 0ms.

C. Result

As Fig. 6 shows, the web application receives 1000 messages in total, the *Interval* of HTTP Polling is 1000 milliseconds, and both the *Sleep Time* of HTTP Long Polling and HTTP Streaming are 500 milliseconds. The *Delaies* for different server push methods are different and changing over time. In the Polling mode, *Delay* is always less than or equal to the *Interval*, for the Long Polling and Streaming modes, the value of *Delay* is always less than the *Sleep Time*.

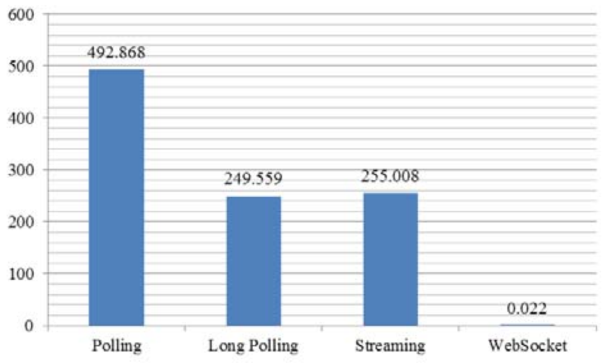

Figure 7. Average delay(ms) of different methods

Fig. 7 shows that the HTTP Polling and Long Polling have nearly same delays, and HTTP Polling has nearly twice delay than HTTP Long Polling. It is no doubt that the WebSocket has the least latency of real-time communication because the server can send new messages to web application as soon as possible via the established WebSocket connection.

From Fig. 8, it can be observed that HTTP Polling mode costs much more requests than others in order to get messages real-time. The HTTP Long Polling still needs to send more requests than HTTP Streaming because Long Polling will end one HTTP session if there are new messages. The WebSocket just needs one HTTP request to establishment the connection, and after that all the messages are transferred in WebSocket frame.

Our experiment and the measurement have showed that the WebSocket is the best way of implementing server push function in web browser based real time communication applications, other server side push methods for providing real-time data involve HTTP request and response headers, which contains lots of additional and unnecessary header data. But there are problems because WebSocket has some compatibility problems that not all browsers and servers are supporting it. HTTP Streaming has a better performance than HTTP Long Polling which needs much fewer HTTP requests, but HTTP Streaming has a problem of displaying in browser which influences the user experience seriously. HTTP Long Polling does not has the display problems but it will degenerate to HTTP Polling if there are

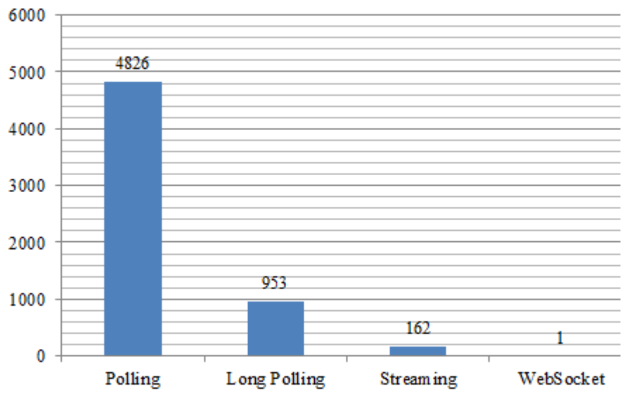

Figure 8. Number of HTTP requests sent by different methods to get 1000 messages

frequently new messages coming. Nowadays, HTTP Long Polling and HTTP Streaming are the most popular solutions for the problems of server push and have been used in many stable web based applications. For HTTP Polling, although it is the first attempt for web application in real-time data transporting and has many problems like high delay and too many requests, it still has its using scenario. When the network is not very well like in the scenario of mobile environment, for the web browser based real-time application, maintaining a persistent connection is not a good choice and of course HTTP Polling is better because the interval of different requests can leave enough time for network accessing becomes stable.

IV. Conclusions

Weighing the advantages and disadvantages, the best way to implement server push for the web browser based real-time application is WebSocket, if the WebSocket connection cannot be established or interrupted by bad network environment, the application can use HTTP Long Polling or Streaming instead. The HTTP Polling is the last choice if all of above methods are failed and cannot work well.

We will further study the influences of different server push methods on web servers especially the performance study. We also wants to design a common framework containing all of these methods for web based real-time applications.

Appendix A HTML5 WebSocket API

The WebSocket API has been defined in the HTML5 specification. Developers can use these APIs in JavaScript to create real-time communicating applications. The WebSocket interface in web browser is given below.

```
interface WebSocket {
    readonly attribute DOMString URL;
    const unsigned short CONNECTING = 0;
    const unsigned short OPEN = 1;
    const unsigned short CLOSED = 2;
    readonly attribute unsigned short readyState;
    readonly attribute unsigned long
bufferedAmount;
    attribute Function onopen;
    attribute Function onmessage;
    attribute Function onclose;
    boolean send(in DOMString data);
    void close();
};
WebSocket implements EventTarget;
```

To connect to the server with WebSocket connection, just create a WebSocket instance with an URL as parameter. If the server supports the WebSocket protocol stack, the connection is established by upgrading from the HTTP protocol during the initial handshake between web browser and the server.

The interface of WebSocket defines three states: *CONNECTING*, *OPEN* and *CLOSED*, developer can get the status of WebSocket connection by the attribute *readyState*. Besides, the WebSocket interface also implements the *EventTarget*, it means that the interface can get events from browser and invoke related functions to deal with them. When the state of connection from *CONNECTING* changed to *OPEN*, the callback function *onopen* will be called; If the state changed to *CLOSE*, the function *onclose* will be called. And when there are messages coming from serve side, the *onmessage* function will be called and user can get the data at real time.

Except the states and callback functions, developer can invoke the *send* method to send data to the server side and the *close* function to close the WebSocket connection.

Acknowledgment

National Key Basic Research Program of China (973 Program)(2009CB320504); Innovative Research Groups of the National Natural Science Foundation of China (61121061); Important national science & technology specific projects: Next-generation broadband wireless mobile communications network (2012ZX03002008-002-03); Independent Research Project for the Base (K2012005).

References

[1] Tencent Web QQ Application [OL]. http://web.qq.com
[2] About chat in Gmail, Google Chat [OL]. http://support.google.com/chat/bin/answer.py?hl=en&answer=161934
[3] Ignite Realtime: SparkWeb Instant Message Client [OL]. http://www.igniterealtime.org/projects/sparkweb/index.jsp [2012-12]
[4] IETF, RFC 2616. Hypertext Transfer Protocol -- HTTP/1.1 [S]. June 1999.
[5] Paul Davern, Noor Nashid, Cormac J Sreenan, Ahmed Zahran, "Client-Side Framework for Automated Evaluation of Mechanisms to Improve HTTP Performance", Journal of Networks, Vol 7, No 11, pp. 1749-1759, November 2012.
[6] Yi Xie, Shengsheng Tang, Xiangnong Huang, Chenghua Tang, "Modeling Web Session for Detecting Pseudo HTTP Traffic", Journal of Computer, Vol 8, No 2, pp. 341-348, February 2013.
[7] Jesse James Garrett. "Ajax: A New Approach to Web Applications" [OL]. Adaptive Path, LLC. [2005-2-18] http://adaptivepath.com/publications/essays/archives/000385.php
[8] W3C, WD-html5-20121025. Specification of HTML5 [S]. 2012.
[9] Runhe Huang, Kei Nakanishi, Jianhua Ma, Bernady O. Apduhan, "An Object-oriented Design and Push Web Server based Framework for Physical Object Interactions and Services", Journal of Software, Vol 3, No 8, pp. 34-41, November 2008.
[10] Jobs, "Why Apple banned Flash from the iPhone" [OL]. http://news.cnet.com/8301-30685_3-20003742-264/steve-jobs-letter-explaining-apples-flash-distaste
[11] Wikipedia, The Free Encyclopedia. Instant Messaging [OL]. http://en.wikipedia.org/wiki/Instant_messaging

[12] XMPP Technologies. BOSH: Bidirectional-streams Over Synchronous HTTP [OL]. http://xmpp.org/about-xmpp/technology-overview/bosh
[13] Alex Russell. “Comet: Low Latency Data for the Browser” [OL]. http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser
[14] W3Schools Online Web Tutorials. HTML Iframe Tag [OL]. http://www.w3schools.com/tags/tag_iframe.asp
[15] IETF, RFC 6455. The WebSocket Protocol [S]. 2011.
[16] Steve Souders. “Roundup on Parallel Connections” [OL]. http://www.stevesouders.com/blog/2008/03/20/roundup-on-parallel-connections
[17] Infrequently Noted. “What else is burried down in the depth’s of Google’s amazing JavaScriot?” [OL]. http://infrequently.org/2006/02/what-else-is-burried-down-in-the-depths-of-googles-amazing-javascript
[18] Support for HTML5. “When can I use WebSocket” [OL]. http://caniuse.com/#feat=websockets
[19] Apache Software Foundation HTTP Components. HttpCore [OL]. http://hc.apache.org/index.html
[20] Mozilla Firefox Web Browser [OL]. http://www.mozilla.org/en-US/firefox/new/

Kai Shuang was born in Liaoning Province, China, in 1977. He received Ph.D. degree from State Key Laboratory of Networking & Switching Technology at BUPT in 2006. Now, he is a associate professor at BUPT. His current research interests include next generation network technology, softswitch technology, distributed computation, cloud computing and web services technology.

Kai Feng was born in Shanxi Province, China, in 1988. He received master degree from State Key Laboratory of Networking & Switching Technology at BUPT in 2013.