Communicating and Displaying Real-Time Data with WebSocket

Amninder S Narota* Yousef Alghamdi† Rajan Subramaniam‡
Department of Computer Science,
Central Michigan University
*narot1a@cmich.edu †algha1ya@cmich.edu ‡subra3r@cmich.edu

Abstract—WebSockets isn't intended to replace AJAX and is not strictly even a replacement for Comet/long-poll (although there are many cases where this makes sense). The paper is not only test the connection between server and browser but it also provide how web socket is supported in different Operating Systems.

The purpose of this research is to provide a low-latency, bi-directional, full-duplex and long-running connection between a browser and server. WebSockets opens up new application domains to browser applications that were not really possible using HTTP and AJAX (interactive games, dynamic media streams, bridging to existing network protocols, etc).

The fact that a Websocket is a fully bi-directional communication channel between the browser and server immediately opens up some very interesting opportunities for web applications. Because the connection is persistent, the server can now initiate communication with the browser. The server can send alerts, updates, notifications. This adds a whole new dimension to the types of applications that can be constructed.

The test results showcases the escalation in the performance of the websocket over the ajax in terms of data distribution per millisecond.

Index Terms—Real-Time Data, Websocket, real time display data, analysis, communction, AJAX, data communications aspects, Web servers

1 Introduction

AJAX is rapidly becoming an integral part of several websites, several well established brands online now use AJAX to handle their web applications because it provides better interactivity to their users, this is due to the fact that implementing AJAX on a website, does not require a page to be reloaded for dynamic content on web pages. While there are numerous reasons to switch to AJAX there are quite a few matters that would make you reconsider using this combination of technologies as well. Below are some of the advantages and disadvantages of using AJAX.

1.1 Advantages of AJAX [22]

- **Better interactivity [23]:** This is most striking benefit behind why several developers and webmasters are switching to AJAX for their websites. AJAX allows easier and quicker interaction between user and website as pages are not reloaded for content to be displayed.
- **Easier navigation [23]:** AJAX applications on websites can be built to allow easier navigation to users in comparison to using the traditional back and forward button on a browser.
- **Compact:** With AJAX, several multi purpose applications and features can be handled using a single web page, avoiding the need for clutter with several web pages.

- **Backed by reputed brands:** Another assuring reason to use AJAX on your websites is the fact that several complex web applications are handled using AJAX, Google Maps is the most impressive and obvious example, other powerful, popular scripts such as the vBulletin forum software has also incorporated AJAX into their latest version.

1.2 Disadvantages of AJAX [10]

- **The back and refresh button are rendered useless:** With AJAX, as all functions are loaded on a dynamic page without the page being reloaded or more importantly a URL being changed (except for a hash symbol maybe), clicking the back or refresh button would take you to an entirely different web page or to the beginning of what your dynamic web page was processing. This is the main drawback behind AJAX but fortunately with good programming skills this issue can be fixed.
- **It is built on javascript:** While javascript is secure and has been heavily used by websites for a long period of time, a percentage of website surfers prefer to turn javascript functionality off on their browser rendering the AJAX application useless, a work around to this con is present as well, where the developer will need to code a parallel non-javascript version of the dynamic web page to cater to these users.

WebSocket is a standardized interface for continuous, bi-directional and low-overhead communication between Web browser clients and back-end servers. A WebSocket connection has lower latency levels and lower bandwidth requirements than AJAX/Comet and related design patterns. As a result, WebSocket allow developers to readily create dynamic browser-based applications that would be difficult or impractical to implement using the HTTP request-response model.

The advantages of WebSocket are achieved without browser plug-ins in many modern browsers, including the Safari browser for iOS devices, and some WebSocket implementations include an emulation capability for legacy browsers.

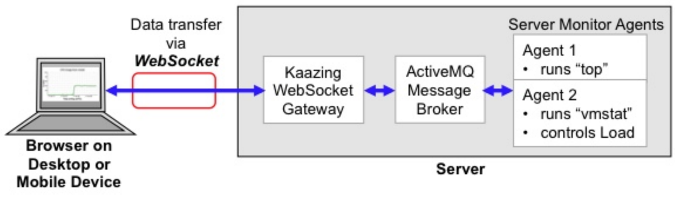

Fig. 1. Demonstration System Schematic

The properties of WebSocket make the interface ideally suited for dynamic web applications in which large amounts of time-critical data must be transmitted to and displayed in the browser. A potential application which could require this type of capability is remote monitoring of servers used in large-scale software systems.

To illustrate the feasibility of the use of WebSocket for this type of application, Kaazing Corp. (kaazing.com) commissioned Bergmans Mechatronics LLC [5] to develop a simple WebSocket-based Amazon Elastic Compute Cloud (EC2) performance monitor.

The resulting system [5], shown schematically in Figure 1, operates as follows. Two Server Monitor Agent programs execute on the EC2 server at a rate of once per second. One agent executes the top shell command and the other executes the vmstat shell command. The results of these commands are transmitted as a Streaming Text Orientated Messaging Protocol (STOMP) [20] message over conventional TCP socket to an ActiveMQ message broker on the server.

2 Background & Related Work

In an extant era Internet became a potential information repository for an individual in terms of personal and business perspective. Recent trends in WWW- World Wide Web revolves around the concept known as dynamic data where efficiency plays a vital role. Certain applications like teleconferencing, social networking works on the real time data feeds where efficiency of the system is graded in term of latency. The conventional methods are suitable for the static data types and it is proved that their performance is degraded in real time data communication scenario where, it experience high latency in network communication. The conventional methods employed over the years include HTTP polling, HTTP Long Polling, Comet [18] etc. In this chapter we are going to discuss the earlier HTTP sockets and the current method websocket.

2.1 HTTP Polling

HTTP polling [18] works on the principle of sending http requests between the server and the client. The sequence involves the client sending http request to the request and after some time known latency time the server responds to the client via http request.

2.2 HTTP Long Polling

HTTP polling [18] works on the principle of sending http requests between the server and the client. The sequence involves the client sending http request to the request and after some time known latency time the server responds to the client via http request.

2.3 WebSocket Web Browser

WebSocket is a TCP based protocol, which enable full-duplex (two-way) bidirectional communication between the client and the server side. The goal of this technology is to formulate the flow of requests between the server and client in a more dynamic way thereby promoting this technique to handle more real time data. It is commonly referred as HTML5 WebSocket and it is used in the client side that is especially for the web browser. WebSocket is developed based on the Java Script API [8]. The working principle of websocket does not depend on the conventional http request it involves asynchronous communication pattern between the server and the client. In the asynchronous process the server will not wait for the client authentication instead it will simultaneously transmits the message between the server and the client. This convention is highly suitable for the real time data communication, which is frequently used in the real world applications like online chat, social network groups etc.

2.4 WebSocket (Mobile Communication)

WebRTC Real Time Communication is a standard protocol that works on the principle of real time data communication between the server and the client. In this case the client will be a mobile browser. WebRTc [1] is based on the websocket but its communication channel is based on the cellular network. The ultimate goal of the WebRTC is to provide a communication with a minimum handshake and low latency between the mobile user and the server [15]. This technique will be an idle choice for the real time data communication and it follows the similar working principle of websocket, where it handles the asynchronous message flow between the server and the mobile client. This technique also equipped with the concept known as http page compression where all the data (Webpages) will be compressed and will be rendered via mobile client. The webpage compression technique is used in the process of minimizing the power consumption of the mobile device during the synchronization period.

2.5 Websocket Server

The architecture of WebSocket Server is based on two technologies,

1) ***PyPy*:** PyPy is an implementation of Python Language that works with a tracing JIT(Just in Time Compilation). JIT will run the program, identify code paths that are executed very often and compile Python into native machine code. The end

result is higher performance compared to the standard CPython (sometimes by a factor of 10 or more [17]).

In the past, most programs written in any language have had to be recompiled and sometimes, rewritten for each computer platform. One of the biggest advantage of JIT that you only have to write and compile a program once. JIT-compilation takes place on same system that the code runs, it can be fine tuned for that particular system [16]. Adapting to run-time metrics, a JIT-compiler can not only look at the code and the target system, but also at how the code is used. It can instrument the running code, and make decisions about how to optimize according to, for example, what values the method parameters usually happen to have.

2) ***Autobahn*:** Autobahn is an open source real time framework for Web, Mobile and the Internet of Things that is based on WebSocket. The Autobahn project provides open-source implementation of WebSocket protocols in different languages:

- AutobahnPython
- AutobahnAndroid

To create a WebSocket server, we will write a protocol class to specify the behavior of the server. The second thing to note is that we override a callback method defined in the Autobahn whenever the callback related event happens.

Our system will contact among peers in JSON formatted data structure. JSON is smaller than corresponding XML, JSON is faster, i.e simpler syntax which makes it to easier to parse. JSON just happens to be a very good fit for our research and a natural way to evolve. It is minimal, textual and a subset of javascript. Specifically, it is subset of ECMA-262 [6]. A number of people independently discovered that JavaScript's object literals were an ideal format for transmitting object-oriented data across the net.

2.6 Web Client

Web client of Radio on map will be browser based client. It will have two ways of real time streaming data. Down data streaming (a radio streaming) and up data streaming (a client feed data). Data exchange between client and server will be in JSON format that provide flexibility for the application in term of dealing with different type of browsers and platforms,because client applicationneeds to support different browsers, realtime data exchange and communication, which websocket is the best tool for dealing with long pulling, handle connection errors and formate of exchange data. That is why websocket client implement as main platform to be use to handle connection between clients and server.

2.6.1 Long Polling

Client application and server has two ways of long pulling. In client, income streaming data (radio streaming) will require continue pulling request that websocket has build-in library handle it automatically. Also that apply on the server side pulling request from the client (client data stream), which handles automatically by websocket [14]. Long pulling request will be handle automatically by websocket.

In Http Long Polling the server holds on to the response unless the server has an update, as soon as the server has an update, it sends it and then the client can send another request. Disadvantage is the additional header data that needs to be sent back and forth causing additional overhead.

Realtime example of Http Long polling is Facebook Messaging where it sends a XHR (Ajax) [12]. Hamid Elgendi [7] explains in the implementation of Http Long Polling that request and leaves it open for almost 40 seconds. Instead of sending a request each second to check for new messages (Polling), Long-Polling Ajax Requests sends requests continuously on the long-term. So instead of having 120 Requests sent to the server in two minutes we can only make 3-4 requests.

2.6.2 Connection Operations

There will be three process of connection between server and client that are open connection, close connection and connection errors. Three operations will be handledby websocket library which provides automatic control of connection operations.Open connection process of handshakingand configuring protocol are done by the websocket library that simply done by creating websocket object that has all configurations and protocols data attached with. That apply too on the close connection operation. Connection errors such that client disconnected for some reason websocket will atomically try to reconnected withclient application.

2.6.3 Data Exchange

Data will be exchange between client and server with JSON format that due to the support of different type of browsers and platforms. Web socket transmits data to JSON object with build in library that interprets, parse and format JSON object atomically.

Web socket is perfect tool be used with web client of Radio on map that not only provide simplify API ,but also save time by auto handling connection requests, real time stream and data formatting.

2.7 Mobile Communication: Android

2.7.1 JSR 356

JSR 356 is a java based application interface used to integrate websocket in both client and sever side.Any android client can communicate with the server in the websocket approach using JSR356.It enables the android client to initiate a connection and equips the server so that it awaits connections from all the peers and both the sides are very well equipped with callback listeners such as onOpen, onMessage and onClose.

2.7.2 jWebSocket

JWebsocket is the real time java based websocket, which is used for the real time data communication between the server and the mobile user. JWebSocket is a open source framework which provides a java server and java client and it will be a idle choice for the android platform devices [11]. At present websocket is compatible for all kinds of browsers like chrome, safari and Firefox. The added feature in the JWebsocket is that, it provides a flash plug in which enables cross platform compatibility between different browsers.

***F*unctionalities:** The main functionalities of the JWebsocket involve a typical handshake, request and response between the server and the mobile client.

***H*andshake:**

- ***C*lient [11]:** Initially client (Mobile user) starts the handshake by sending the request message to the server.
  1) GET {path} HTTP/1.1
  2) GET {path} HTTP/1.1
  3) Upgrade: WebSocket
  4) Connection: Upgrade
  5) Host: {hostname}:{port}
  6) Origin: http://{host}[:{port}]
  7) Sec-WebSocket-Key1: {sec-key1}
  8) Sec-WebSocket-Key2: {sec-key2}
  9) 8 Bytes generated {sec-key3}
- ***S*erver [11]:** The server replies to the client handshake
  1) HTTP/1.1 101 WebSocket Protocol Handshake
  2) Upgrade: WebSocket
  3) Connection: Upgrade
  4) Sec-WebSocket-Origin: http://{hostname}[:{port}
  5) Sec-WebSocket-Location: ws://{hostname}:{port}/
  6) 16 Bytes MD5 Checksum
- ***C*lient (Chat Sequence):** The client Mobile (Android) user initiate the chat with the server [11].
  1) GET /services/chat/;room=Foyer HTTP/1.1

2) Upgrade: WebSocket
3) Connection: Upgrade
4) Host: jwebsocket.org
5) Origin:http://jwebsocket.org
6) Sec-WebSocket-Key1: 4 @1 46546xW%0l 1 5
7) Sec-WebSocket-Key2: 12998 5 Y3 1 .P00
8) n̂:ds[4U

- **Server Reply:** The server will reply to the mobile user in a asynchronous manner [11].

1) HTTP/1.1 101 WebSocket Protocol Handshake
2) Upgrade: WebSocket
3) Connection: Upgrade
4) Sec-WebSocket-Origin:http://jwebsocket.org
5) Sec-WebSocket-Location: ws://jwebsocket.org/services/chat
6) 8jKS'y:G*Co,Wxa-

3 Method

There are two types of research which are intended in this research methods: Qualitative and Quantitative. In this research we learn when to use each type of research, how to conduct research with members of our intended audience, and how we can use the data we collect to inform in our research. Qualitative, quasi-quantitative and quantitative research methods are discussed separately.

Qualitative VS Quantitative method

- Qualitative Method
  1) Provides depth of understanding
  2) Asks "Why?"
  3) Studies motivations
  4) Enables discovery
- Quantitative Method
  1) Measures level of occurrence
  2) Asks "How many?" and "How often?"
  3) Studies actions
  4) Provides proof

Qualitative Research

The goal of qualitative research [19] is to gain insights into an intended audience's lifestyle, culture, motivations, behaviors, and preferences. The research is going to be conducted by selecting small group of people chosen for particular characteristics for convening discussion or observing individual's behaviors during the communication by keeping the discussion fairly unstructured, so that participants are free to make any response and are not required to choose from a list of possible responses.

Qualitative research can not be quantified or subjected to statistical analysis or projected to the population from which the respondents were drawn because participants are not selected randomly (to be representative of the population as a whole) and because not all participants are asked precisely the same questions.

Quantitative Research

The goal of this research is measurement of particular variables by conducting quantitative research [9] by selecting large group of people using a structured questionnaire containing predominantly forced-choice or closed-ended questions. The results of quantitative research [9] can be analyzed using statistical techniques.

Most surveys are custom studies that are designed to answer a specific set of research questions. Some surveys, however, are omnibus studies, in which we will add questions about topic to an already existing survey. A number of national and local public opinion polls offer this option. Following are the steps for the surveys:

- Plan the study
- Determine how the sample will be obtained and contacted
- Develop and pretest the questionnaire
- Collect the data
- Analyze the result

3.1 Html Http Long polling

We developed two different systems to achieve the results. *Aghaei, Nematbakhsh, Farsani* [2] apprised the advent of Web4.0 with the introduction of AJAX for the WebOS. Apart from the Advantages mentioned in **Section 1.1** Aron Weiss cited that "*AJAX development is proliferating in the field, powering many of the Web-based applications that are behind the WebOS buzz*" [21] which we believe is true to some extent but the performance can be further enhanced with the introduction of Websocket to the web data communication vogue technology cluster.

Server side scripting for AJAX was developed using Django and client side scripting was developed in HTML and JQuery for Ajax calling. The URL was provided to listen for the GET request from the client and the response was listened back. This is discussed in detail in **Section *3.1.1***.

The request/response cycle that we are so heavily reliant on in the web development world is largely a myth. In reality, the browser and the server are two asynchronous nodes in a network. A request does not guarantee a response. In point of fact, the entire AJAX protocol could be built using Websockets technology. This makes Websockets a literal superset of AJAX. So it makes sense that we might abandon a limiting technology for a broader technology.

We believe that while REST/AJAX has served the web community well, it is probably time to start looking ahead to the future. And while it might be tempting at first to try to fit Websockets into the REST architecture [3] that we are all so familiar with, We believe that ultimately this will prove to be limiting. The best approach is to embrace realtime communication as a new development paradigm, and see what interesting ideas and patterns we can come up with based on this new concept.

To achieve results for WebSocket request and response we developed server locally. The server was provided with port number and local IP address to which the request was listened. Both client and server scripting was written in Python language and both the system ran locally. This methodology is further discussed in **Section 3.1.2**.

3.1.1 Request-Response using Ajax

Our method was implemented using Django and Jquery AJAX to see how they would perform under stress. Code is mentioned in **Appendix B.1**. It sends a simple data structure to server which echo it back. As soon as the response comes back, it starts over till it is done X number of iterations.

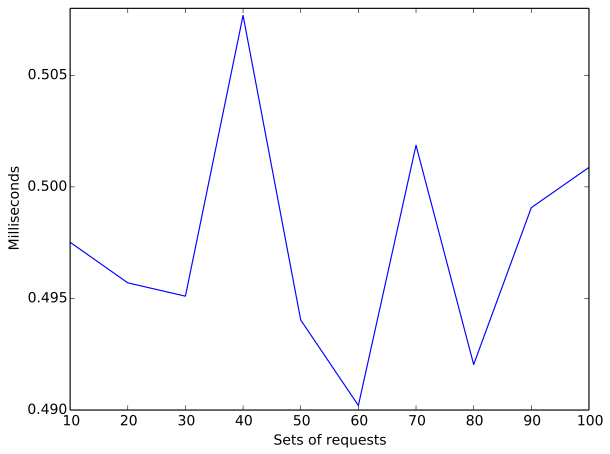

Fig. 2. Ajax time for request and response

We ran test of 10 calls of 10 to 100 sets of Ajax calling with an average call of 596 request. On average it took 0.49 milliseconds to listen each request.

3.1.2 Request-Response using Websocket

The advantage with WebSockets (over AJAX) is basically that there is less HTTP overhead. Once the connection has been established, all future message passing is over a socket rather than new HTTP request/response calls. So, we assume that WebSockets can send and receive much more messages per unit time.

The system was implemented using library Autobahn [4] to create client and server for listening requests. The client first listens to the web protocol on the port on which server is running and the data is sent to the server on that port. The time was recorded at the time of sending and at the time of receiving. The result is obtained when server sends back the

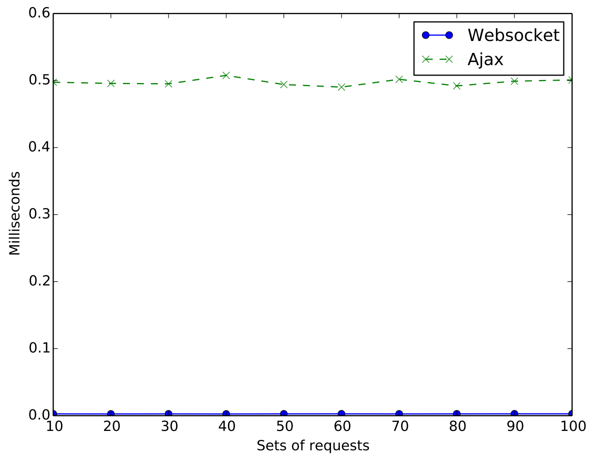

Fig. 3. Websocket vs Ajax

response. The code of client is mentioned in **Appendix A.1**.

The Data Structure object sent over network was in JSON format and followed REST api [3] pattern for communication as it helps to organize very complex application into simpler resources. [13]

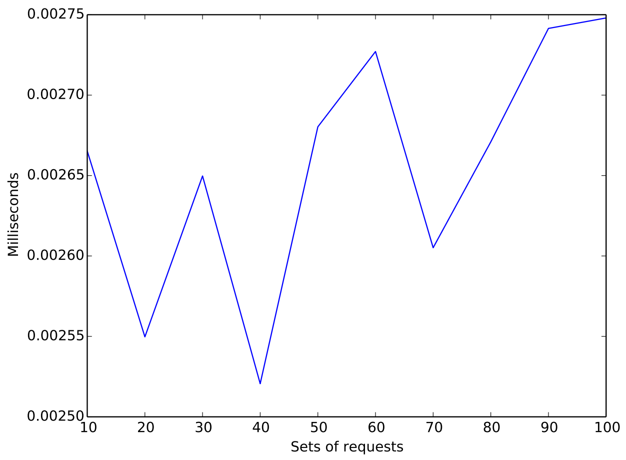

Fig. 4. Websocket time for request and response

In this method, sample data structure was sent to the server at different interval of times and time difference was recorded. On average 294 requests were made to server and time taken for average response was 0.0026 milliseconds.

The server listens to the change on the port defined and records the change. In our method, response was sent to the same requesting the peer. The code snippet is defined in **Appendix A.2**. Every

4 Results

To test with new method for request-response time for long polling, we found that Websocket takes comparatively less time. We performed 10 different sets of test and each time performance of Websocket showed significant results which is shown in Figure 3.

We waned to compute the probability that our random outcome is within a specified interval, i.e

$$P(a \leq X \leq b) \tag{1}$$

where a could be -infinity and/or b could be +infinity. for continuous random variables, this probability corresponds to the area bound by a and b and under the curve. The probability X is a specific value, i.e

$$\begin{aligned} & P(a \leq X \leq b) \\ = & P(a < X \leq b) \\ = & P(a \leq X < b) \\ = & P(a < X < b) \end{aligned}$$

We generated the Normal Distribution as mentioned in *Equation 2* since it has a number of interesting properties that make it useful for our statistical analysis.

$$Y = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}(\frac{x-\mu}{\sigma})^2} \tag{2}$$

Normal Distribution of Ajax call is shown in *Figure 5*. We found that the 5962 data points with $\mu = 0.496828916408$ and $\sigma = 0.288937525397$ were approximately normally distributed over 0.49.

Also, we found that 2499 data points with $\mu = 0.00265611189863$ and $\sigma = 0.00162777352991$ in web socket are normally distributed approximately over 0.0026 as shown in *Figure 6* where μ and σ are mean and standard deviation.

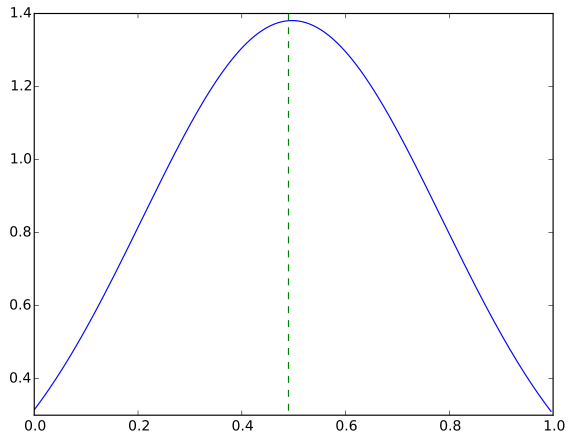

Fig. 5. Normal Distribution of Ajax Calls

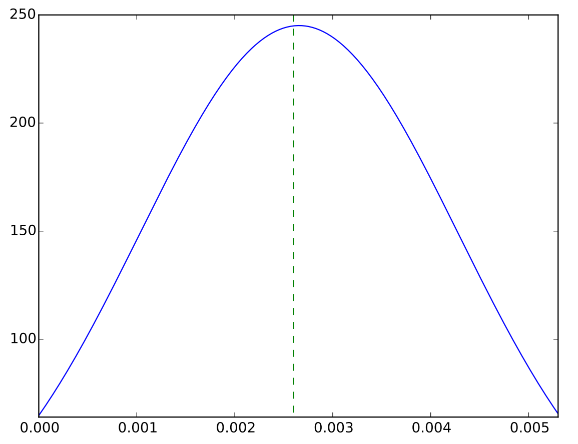

Fig. 6. Normal Distribution of Websocket Calls

5 Conclusion

We introduced a new approach of Communicating and Displaying real-time data using Websocket over the traditional method Ajax. The main goal of the research was to analyze and test whether the new approach using Websocket is more efficient when compared to Ajax and the result of this experiment showcased the efficiency of the web socket approach over the ajax technique in terms of message response time under the typical Request-Response scenario between client and the server.

We spotted a considerable increased efficiency in Websocket, when compared to the ajax in the communication process. We tested the acquired result by performing Normal distribution over the data points distributed per milli second for both the cases and it is proved that web socket approach aces ajax by covering data points at the minimum time intervals which in fact is an ideal requirement for the effective real-time data communication.

6 Future Work

We have the analyzed and tested the performance of web socket over ajax in terms of request and response time in a typical client-server environment but we have to test the performance of web socket in different areas and one among them is the power consumption. At present mobile driven world power consumption is regarded as a vital factor for evaluation process. We intend to analyze the power consumption pattern of both the cases and test whether web socket is more efficient than ajax in terms of power consumption.

References

[1] *HTML5 Connectivity Methods and Mobile Power Consumption*. W3.org, October 2010.

[2] S. Aghaei, M. A. Nematbakhsh, and H. K. Farsani. Evolution of the world wide web: From web 1.0 to web 4.0.

[3] A. Anisenkov, S. Belov, A. Di Girolamo, S. Gayazov, A. Klimentov, D. Oleynik, and A. Senchenko. Agis: The atlas grid information system. In *Journal of Physics: Conference Series*, volume 396, page 032006. IOP Publishing, 2012.

[4] Autobahn.ws. Open-source real-time framework for web, mobile & internet of things.

[5] K. Corps. Real-time web-based platform provides competitive differentiator. 2012.

[6] ECMA. *Standard ECMA-262*, 3 edition, December 1999.

[7] H. Elgendy. Understanding ajax long-polling requests. http://webcooker.net/ajax-polling-requests-php-jquery/.

[8] J.-P. Erkkilä. *WebSocket Security Analysis*. PhD thesis, Aalto University School of Science.

[9] M. Fleischmann, J. M. Bloemhof-Ruwaard, R. Dekker, E. van der Laan, J. A. van Nunen, and L. N. V. Wassenhove. Quantitative models for reverse logistics: A review. *European Journal of Operational Research*, 103(1):1 – 17, 1997.

[10] Y. Gan and H. Yang. Ajax and web services integrated framework based on duplicate proxy pattern. In *Information Technologies and Applications in Education, 2007. ISITAE '07. First IEEE International Symposium on*, pages 297–302, Nov 2007.

[11] jwebsocket.org. Introduction websockets for android. October 2010.

[12] J. Lengstorf and P. Leggetter. What is realtime? In *Realtime Web Apps*, pages 3–13. Springer, 2013.

[13] L. Li and W. Chou. Design and describe rest api without violating rest: A petri net based approach. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 508–515. IEEE, 2011.

[14] Q. Liu and X. Sun. Research of web real-time communication based on web socket. 2012.

[15] K. ma and R. Sun. Multiple wide tables with vertical scalability in multitenant sensor cloud systems. *International Journal of Distributed Sensor Networks*, 2013(867693):10, 2013.

[16] F. Maruyama, S. Matsuoka, and K. Shimura. Openjit 2: The design and implementation of application framework for jit compilers.

[17] PyPy.org. Pypy performance optimization, October 2014.

[18] S. M. Rakhunde. Real time data communication over full duplex network using websocket. In *IOSR Journal of Computer Science (IOSR-JCE)*, page 19, 2014.

[19] C. Stenbacka. Qualitative research requires quality concepts of its own. *Management Decision*, 39(7):551–556, 2001.

[20] V. Wang, F. Salim, and P. Moskovits. *The Definitive Guide to HTML5 WebSocket*. Apress, February 2013.

[21] A. Weiss. Webos: say goodbye to desktop applications. *Networker*, 9(4):18–26, 2005.

[22] C. Ye. The advantages research about the sshdwr + ajax design pattern in software developmen. In *Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011 International Conference on*, volume 9, pages 4570–4573, Aug 2011.

[23] G. Zanker. Ajax. *The Classical Review (New Series)*, 55:19–20, 3 2005.

Appendix A

A.1 Client Code Snippet

```python
import sys
from twisted.internet import reactor
from autobahn.twisted.websocket import WebSocketClientFactory, \
                                       WebSocketClientProtocol, \
                                       connectWS

from time import mktime
import time
import datetime
import json


class BroadcastClientProtocol(WebSocketClientProtocol):
    """
    Simple client that connects to a WebSocket server, send a
    message and print everything it receives.
    """

    def sendMsg(self):
        client_time = float("%.20f"%time.time())
        self.sendMessage("{}".format(client_time).encode('utf8'))
        reactor.callLater(0.1, self.sendMsg)

    def onOpen(self):
        self.sendMsg()

    def onMessage(self, payload, isBinary):
        if not isBinary:
            dict = json.loads("{}".format(payload.decode('utf8')))
            client_time =  float(dict[u'client_time'])
            now = float("%.20f"%time.time())
            if client_time > now:
                diff = client_time - now
            else:
                diff = now - client_time
            with open("data_100.txt", "a+") as f:
                f.write("{}\n".format(diff))
            f.close()
            print diff


if __name__ == '__main__':

    if len(sys.argv) < 2:
        print("Need WebSocket server address, i.e. ws://localhost:9000")
        sys.exit(1)
```

```
factory = WebSocketClientFactory(sys.argv[1])


factory.protocol = BroadcastClientProtocol
connectWS(factory)

reactor.run()
```

A.2 WebSocket python Server Code Snippet

```python
import sys
from time import mktime
import time
import datetime

from twisted.internet import reactor
from twisted.python import log
from twisted.web.server import Site
from twisted.web.static import File

from autobahn.twisted.websocket import WebSocketServerFactory, \
WebSocketServerProtocol, listenWS

class BroadcastServerProtocol(WebSocketServerProtocol):

    def onOpen(self):
        self.factory.register(self)

    def onMessage(self, payload, isBinary):
        if not isBinary:
            msg = "{}".format(payload.decode('utf8'))
            client_time = float(msg)
            server_time = float("%.20f"%time.time())
            self.factory.broadcast(\
                '{"client_time": %.20f, "server_time": %.20f}' \
                %(client_time, server_time))


    def connectionLost(self, reason):
        WebSocketServerProtocol.connectionLost(self, reason)
        self.factory.unregister(self)


class BroadcastServerFactory(WebSocketServerFactory):
    """
        Broadcast message to all clients
    """
    def __init__(self, url, debug = False, debugCodePaths = False):
        WebSocketServerFactory.__init__(\
            self, url, debug=debug, debugCodePaths = debugCodePaths)
        self.clients = []
        self.tickcount = 0
        self.tick()

    def tick(self):
        self.tickcount += 1
        self.broadcast("tick %d from server" % self.tickcount)
        # reactor.callLater(1, self.tick)

    def register(self, client):
        if not client in self.clients:
            print("registered client: {}".format(client.peer))
            self.clients.append(client)

    def unregister(self, client):
        if client in self.clients:
            print("unregisterd client: {}".format(client.peer))
            self.clients.remove(client)

    def broadcast(self, msg):
        print("{}".format(msg))
        for c in self.clients:
            c.sendMessage(msg.encode('utf8'))
            print("message sent to {}".format(c.peer))
```

APPENDIX B

B.1 Ajax Code Snippet

```
function waitForMsg(){
    /* This requests the url "msgsrv.php"
    When it complete (or errors)*/
    var dt = new Date();
    var time = dt.getTime()/1000 //unix time in milliseconds
    var delay = 100;
    var error_delay = 150
    $.ajax({
        type: "GET",
        url: "msgsrv.php",
        data: { time: time, set: delay },
        async: true,
        cache: false,
        timeout:50000,

        success: function(data){
          console.log(data);
            addmsg("new", data);
            setTimeout(
                waitForMsg,
                delay
            );
        },
        error: function(XMLHttpRequest, textStatus, errorThrown){
            addmsg("error", textStatus + " (" + errorThrown + ")");
            setTimeout(
                waitForMsg,
                error_delay);
        }
    });
};
```

B.2 Python code snippet to listen to Ajax Call

```
from random import randint
from time import sleep
from django.http import HttpResponse, HttpResponseNotFound
from time import mktime
import datetime

def retmsg(request):
    client_time = float(\
        request.GET.get('time'))
    server_time = float(\
        mktime(datetime.datetime.now().timetuple()))
```

```
diff = client_time - server_time
s = request.GET.get('set')
file_name = "data_{}.txt".format(s)
with open(file_name, "a+") as f:
    f.write("{}\n".format(diff))
f.close()


return HttpResponse("{}:{}".format(s, diff))
```