UNIVERSITY OF GOTHENBURG

Evaluation of WebSocket Communication in Enterprise Architecture

Bachelor of Science Thesis Software Engineering and Management

AMIR ALMASI
YOHANES KUMA

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, June 2013

Evaluation of WebSocket Communication in Enterprise Architecture

AMIR ALMASI
YOHANES KUMA JOTE

Examiner: MATTHIAS TICHY

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2013

Evaluation of WebSocket Communication in Enterprise Architecture

Amir Almasi
Software Engineering and Management
University of Gothenburg
Mobile: +46 70 071 02 65
Email: almaasi.amir@gmail.com

Yohanes Kuma
Software Engineering and Management
University of Gothenburg
Mobile: +46 76 246 44 98
Email: yohaneskuma@gmail.com

***Abstract*—The adoption of new technologies in enterprise environments are always challenging. These challenges are regarding the compatibility of a new technology with the existing architecture. WebSocket is one of the new technologies in terms of distributed enterprise applications. WebSocket is a recently standardized protocol for exchanging real-time data in distributed applications including web applications. WebSocket significantly contributes in faster data transmission by introducing a bi-directional communication. However, enterprises including Volvo IT, which was the industrial collaborator of this study, are interested on more research regarding the concerns involved with the adoption of WebSocket technology in enterprise environments. This study aimed at two objectives. The first objective was to discuss WebSocket technology with regard to Volvo IT enterprise architectural principles; the second objective targeted on investigating enterprise web middleware infrustructure challenges while adopting WebSocket technology. Targeting these two important objectives, qualitative and design research approaches were employed. By means of qualitative strategy, WebSocket technology was discussed based on the most relevant Volvo IT enterprise architectural principles. The design research focus was to develop a WebSocket application prototype targeting design recommendations to overcome the challenges of EWMI. The WebSocket application prototype was confronted against a simulated laboratory which is similar to Volvo IT EWMI architecture. The findings from the two employed research approaches revealed the gained benefits and incompatibility concerns when adapting WebSocket technology.**

***Index Terms*—WebSocket, Proxy Server, Firewall, Load Balancer, Middleware Infrastructure, Enterprise Architecture, Enterprise Architectural Principles, Web Applications**

I. Introduction

The need for real-time data has significantly increased in recent years by different technologies such as communication, security, gaming, embedded, etc. For example, medical situations [1] and military services that fulfill critical functions increasingly require more real-time data. Also technologies that fulfill less critical functions such as wireless consumer technologies and automotive embedded systems require increasing amounts of real-time data [2].

More importantly though, real-time data requirements are experiencing substantial growth in terms of web applications since they provide more interactive communication experience for users. Real-time data communication in terms of web applications is considered as soft real-time data exchange. Examples of web applications which require real-time data include business-intelligence systems [3], collaborative web applications [4] [5], multi-player online games [6] [7], and cloud computing [8]. These web applications fulfill their real-time data thirst by adding additional complexity on Hyper Text Transfer Protocol (HTTP). However, although the above web applications employ HTTP, one of the ideal communication protocols for real-time communication is the WebSocket protocol [10].

WebSocket is a bi-directional communication protocol based on a single TCP connection, which enables simultaneous data transmission in both directions between client and server. By utilizing WebSocket technology, a persistent connection between client and server is created that significantly reduce the exchanged messages by eliminating the unnecessary HTTP requests and overhead headers [9]. While WebSocket is believed to be the next generation backbone for real-time web applications, it is in early stage to be used in enterprise environment. Adoption of WebSocket technology in real-time enterprise applications requires further research.

In addition, some early adopters of WebSocket technology raise concerns regarding the compatibility of WebSocket technology with Enterprise Web Middleware Infrastructures (EWMI), such as proxy servers, load balancers, firewalls, message brokers, etc [10][34]. Worsening the issue, there is a limited amount of research suggesting design recommendations with respect to EWMI.

This study evaluates WebSocket technology in terms of both EWMI challanges and enterprise architectural principles. The architectural principles of Volvo IT were the main focus of this study. For investigating EWMI challanges, the study analyzed the impact of EWMIs on WebSocket communication in a simulated laboratory environment which was similar to Volvo IT EWMI architecture. A prototype was designed to recommend possible ways of overcoming EWMI challenges. Considering the study duration, the design recommendations will be limited to client and server endpoints without modifying EWMIs. The following specific research questions are addressed:

- RQ1- What are the concerns of WebSocket communication in terms of enterprise architectural principles?
- RQ2- How can enterprises benefit from WebSocket technology while managing the challenges of EWMI?

- RQ3- What tradeoffs should enterprise architects consider in their decisions to choose WebSocket communication?

The next section introduces the technologies mentioned in this study. Section 3 outlines the methodological approach of the study. The findings of the study are presented at section 4. In section 5, the findings are discussed in-depth from the authors point of view. Finally, there is conclusion of the study on section 6.

II. Theoretical Background

A. Real-time Web Communication

Applications with real-time requirement employ various approaches in order to satisfy their needs. More specifically, web applications based on client and server architecture utilize different protocols which are defined in Internet Protocol Suite [29]. The User Datagram Protocol (UDP) has been used by applications, such as Voice Over IP (VOIP), as real-time communication protocol. However, UDP is not preferred by most web applications since it does not guarantee the delivery as well as the right sequence of the data [30]. Most web applications rather prefer to use application layer protocols over Transmission Control Protocol (TCP) in Open Systems Interconnection (OSI) model since TCP guarantees the delivery and sequence of data [31]. OSI model conceptually makes the communication process easier by dividing different communication functionalites into abstraction layers [29]. The use of application layer in web applications facilitates process-to-process communication [55]. Among those application protocols over TCP, HTTP is the most widely used, especially in the World Wide Web (WWW).

HTTP is compatible with all client applications, specifically web browsers, as well as EWMIs. However, in terms of real-time communication, HTTP has some drawbacks. Since HTTP is a synchronous request-response protocol, browsers must send HTTP request each time in order to retrieve the data from servers. This unidirectional nature of HTTP leads to exchange unnecessary messages between clients and servers [9]. Furthermore, another drawback is HTTP header overhead where inevitable information is added as header. Simply put, the HTTP protocol was not designed for real-time communication. Employing HTTP protocol for real-time communication adds complexity, and additional unnecessary network traffic and latency [9].

B. Enterprise Architecture

The word enterprise application refers to complex software with specific responsibilities usually serving to large amounts of clients. Currently, enterprises have complex, scalable, distributed, component-based, and mission-critical applications [32]. More specifically, distributed enterprise applications provide secure, fault-tolerant, and scalable services. Enterprise architecture usually possess higher level reference model. Specific applications in the enterprises will use the reference model as a base for further detailed application architecture [36]. Besides, enterprise architects give more emphasis to non-functional quality attributes. Likewise, Volvo IT has two main documents called Volvo Group Target Architecture (VGTA) and Architecture for Volvo Systems (AVS) which are used as a reference model. To make the non-functional quality attributes more stressed, Volvo Group developed 10 principles called Volvo Group Architecture Principle (VGAP) which are used as guidelines for assuring the non-functional properties of Volvo Group enterprise applications. The following are the 10 VGAPs:

- Conformity to standards
- Autonomous and loose coupling between components and applications
- Simplicity in solutions and work methods
- Strive for usage of existing Volvo services
- Robust solutions
- Performance focus from the start
- Secure solutions
- Good Integration solutions
- Usage of Agile work methods and design principles
- Maintainable solutions

C. Enterprise Web Middleware Infrastructures

Enterprises employ software and/or hardware components beside server endpoints in order to compartmentalize the various components and increase the quality of the services. These software and hardware components can be generalized as Enterprise Web Middleware Infrastructure (EWMI). Some of EWMIs include proxies, firewalls, and load Balancers.

1) Proxy Server: A proxy server is a computer system located between client requesting data and the server endpoint serving the data [38]. Proxy servers are used for caching, authentication and authorization, encryption and decryption, collect statistics about web traffic, and for filtering sensitive information from web requests [38][39][40]. In terms of OSI layers [29], there are two types of proxy servers: Application layer proxy and Transport layer proxy. Application layer proxy server, for instance HTTP proxy, receives HTTP request from the client, takes an action based on defined responsibility, and responds to the request as HTTP response [41][42]. The second type of proxy server, commonly based on TCP protocol, receives the transport layer packets, and redirects the packets to the client or server in order to improve the TCP throughput [41].

Another important aspect considering proxy servers, regardless of any specific protocol, is the use of proxy server. Considering this aspect, proxy servers are categorized into three main categories, namely Explicit proxy, Transparent proxy, and Reverse proxy [43]. In the case of Explicit proxy servers, clients explicitly connect to a proxy server by configuring client application, such as web browsers. In this case, web browsers will be configured to communicate to the server endpoint through the specific port and IP address of the proxy server. Whereas, in Transparent proxy servers case, both client and server are unaware of the existence of these kinds of proxy servers in the communication since Transparent proxy servers do not modify the request or response. The Reverse proxy servers are logically located near the server endpoint

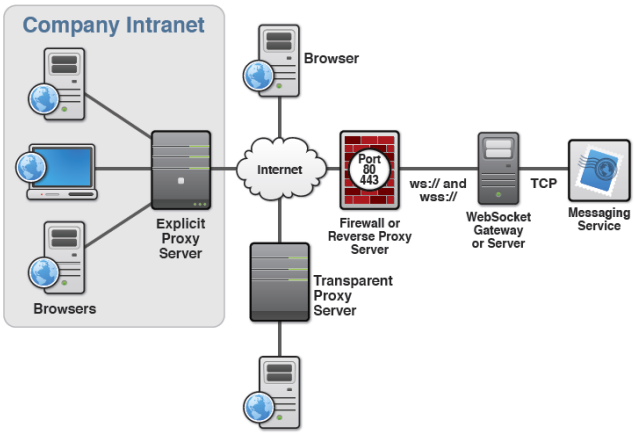

Fig. 1. WebSocket Architecture with EWMI [34]

and have various responsibilities, such as load balancing, content caching [30], encryption and decryption of the data, etc. Reverse proxy servers handle requests and send responses back as if it comes from the main server. Figure 1 illustrates the three aforementioned types of proxy servers.

2) Firewalls: Firewalls are software and/or hardware components which control the incoming and outgoing network traffic by analyzing the data packets [44]. It uses predetermined set of rules to allow traffic. There are three types of firewalls depending on the OSI model layer they focus on:

- Stateless IP-Packet Filtering Firewalls - This type of firewalls operates on Network Layer based on OSI model. It uses the information of IP-Packet such as source and destination address, port number, and Packet type for filtering [44]. Each IP Packet is treated independently regardless of whether the packet itself is part of existing traffic stream.
- Stateful Multilayer-Inspection Firewalls - Such firewalls operate mainly on Transport Layer in the OSI model. They buffer IP Packets until enough information is available to make a judgment about connection state [44]. Knowledge about the connection state enables the firewall to employ additional rules in the filtering process.
- Application Layer Filtering Firewalls- The third kind of firewalls operate on Application Layer in the OSI model enabling them to create a virtual air-gap between the two sides of the network [44]. The virtual air-gap powers firewalls to detect the attempt of unwanted protocol to bypass the firewall on allowed port. They usually accomplish the filtering on a per-process basis instead of filtering connections on a per-port basis [44]. In addition, such firewalls perform elaborated logging and auditing of traffic passing through them.

3) Load Balancers: Load balancers are software or hardware components used to facilitate the participation of several servers for the same services and do the same work [45]. There are two load balancer categories in terms of traffic routing:

- Content-blind Load Balancers - these types of load balancer, also known as Layer-3, 4 Switch, are unaware of the application information contained in incoming requests [47]. In these types, load balancer will route the traffic as soon as it receives TCP request [46].
- Content-aware Load Balancers -these types of load balancers, also known as Layer-7 Switch, work based on Application Layer [47] in OSI model and route HTTP requests to the appropriate server. HTTP traffic will be routed when the load balancer receives HTTP request. One of the advantages considering these types of load balancers is the support for implementing more sophisticated policies [46].

D. WebSocket Implementation Challenges

Adopting WebSocket protocol is not relatively challenging in EWMI architecture, especially in the case of firewalls, since WebSocket protocol uses the standard HTTP ports, 80 (HTTP) and 443 (HTTPS), for WebSocket (WS) and WebSocket secure (WSS) connections respectively. In addition, with regard to Explicit proxy server, clients will first issue HTTP CONNECT to Explicit proxy server before establishing WebSocket communication. This makes Explicit proxies not to be problematic for WebSocket [10]. In spite of that, there are still some other concerns when it comes to traversal of EWMIs such as proxy servers, firewalls, and load balancers. Table I summarizes the concerns. The details are explained as follows:

1) Concerns Regarding Proxy Servers: The concerns with regard to WebSocket communication over proxy servers can be categorized into three groups. The first comes from persistent connection behavior of WebSocket. HTTP proxy servers may choose to aggressively close WebSocket connections (persistent TCP connections) when there are too many persistent connections [33]. In addition, transparent and/or reverse proxy server can close WebSocket connection since it may appear as if they are trying to connect with an unresponsive HTTP server [34]. The second concern is also due to HTTP proxy server. Such proxies usually try to buffer the response from the server endpoint before sending it to the client endpoint [34]. The buffering in most cases continues until their connection with server endpoint is closed, which then send the data to the client endpoint. This behavior of the proxy server raises incompatibility issues for WebSocket communication. The third concern comes when there appears a transparent and/or reverse proxy server. Such proxy servers usually strip off certain HTTP header elements, including WebSocket upgrade element, causing the connection not to be upgraded to WebSocket protocol [34]. As can be noticed on Fig 2, the modification of the HTTP header elements will certainly happen on Function Processing Part I and Part II. [35] Also mentions that the proxy server will modify HTTP header before forwarding data to server or client endpoints.

2) Concerns Regarding Firewalls: Considering the three types of firewalls discussed on section 2.B.2, Stateless IP Packet filtering firewalls and stateful Multi-Layer-Inspection firewalls do not appear to be a concern for WebSocket commu-

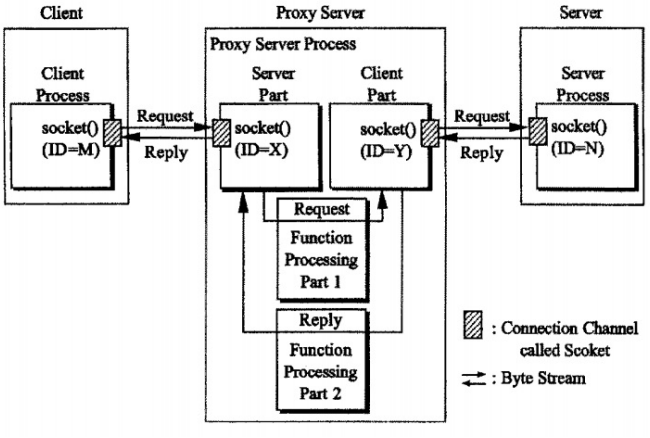

Fig. 2. proxy server architecture [42]

nication since they operate in the lower layers in OSI model. Besides, the fact that WebSocket use the standard ports, make the filtering process not to block WebSocket communications. However, Application Layer filtering firewalls could certainly become a challenge in WebSocket communications since they perform the filtering based on Application Layer protocols. Hence, WebSocket traffic will not be routed if an Application Layer filtering firewall is unaware of WebSocket protocol.

3) Concerns Regarding Load Balancers: Content-blind Load Balancers should not introduce a challenge in WebSocket communication as they operate in the lower layers of OSI model. Content-aware load balancers, however, will certainly be a challenge when they are not compatible with WebSocket protocol.

III. Research Methodology

A. Research Strategy

This study employed two parallel approaches to address the research questions. The first approach was a qualitative study approach in order to address the RQ1. By using this approach, WebSocket technology was evaluated in depth with regard to VGAP. Design research, also known as developmental research, was performed as the second approach in order to address RQ2 and RQ3. Design research was appropriate since the main objective of the research questions were to get in-depth understanding and provide support for the technology rather than explaining the technology [37]. In addition, design research was fundamentally a suitable choice in this case because the relevant elaborated research studies concerning the two questions were lacking at the time.

TABLE I
Summary of EWMI Challenges on WebSocket Communication

WebSocket Communication Challenges	EWMIs Causing the Challenges
Closure of persistent TCP connection	1) HTTP Proxy Servers 2) Transparent Proxy Servers 3) Reverse proxy Servers
Stripping off the WebSocket upgrade header from HTTP	1) Transparent Proxy Servers 2) Reverse proxy Servers
Unawareness of WebSocket communication protocol	1) Application Layer Firewalls 2) Content-aware Load Balancers 3) HTTP Proxy servers

B. Research Setting

This study was conducted in collaboration with Volvo IT. Specifically, the design research approach was conducted in a simulated laboratory environment. The simulated laboratory environment was designed based on Volvo IT EWMI architecture. Accordingly, a reverse proxy server called Nginx [50] version 1.1.19 was configured which was exposed to external networks. The Nginx reverse proxy server was configured to route the network traffic to a server endpoint connected in Local Area Network (LAN). The server endpoint had WebSocket implementation support in order to be responsive for WebSocket client endpoints. Java Enterprise Edition was used as the platform for server endpoint. The client endpoint programming language was HTML5 and Javascript. The client endpoint application ran on web browsers accessing the Nginx reverse proxy server through the Internet. WireShark [51] and NetBeans HTTP Server Monitor [52] were used to observe and analyze the network traffic.

C. Qualitative Approach

Qualitative approach was performed in order to get in-depth understanding about WebSocket technology and VGAP. First, VGAP and VGTA documents constituting the main architecture principles were studied. This step had contributed substantially to the intellectual understanding of VGAPs. In order to reduce the risk of misinterpretation of the principles, interviews with the industry supervisor were conducted. In the second step, a systematic literature review was conducted by using keywords focusing on specific points of interests regarding VGAPs and WebSocket. The literatures are searched from IEEEXplore, ACM DL, Springlink, Google Scholar, Chalmers Library, and Gothenburg University Library. hhen in doubt, findings from the second step were used to clarify our understanding of VGAPs from the first step. Considering the fact that VGAPs target on wide range of application development, the authors prioritized the relevant VGAPs with regard to WebSocket technology. After analyzing the selected VGAPs, the authors gained specific insights about each of them. After a series of iterations, the insights were categorized into five themes. The themes are discussed in section 4.A.

D. Design Research Approach

As [48] pointed out, a design research by itself involves design. Most literatures, however, give little guidance dealing with how to do design research [49]. But for this study, the work of [48] was used as the guideline to design the design research. The design research approach was started with a

pre-study activity. The main goal in the pre-study activity was to understand WebSocket communication implementation, technique wise, without involving EWMI in to the concern. The second activity dealt with clarifying the goals of the design research. Literatures were reviewed in order to describe the desired situation and scope down the focus of the research to specific goals. In the third activity, an empirical analysis of Volvo IT architecture documents with regard to EWMI as well as conducting interviews were performed in order to understand the existing situation and determine the factors which should be addressed to attain the specified goals.

As a fourth activity a prototype was developed in order to address one or more factors identified in the third activity concerning EWMI. On the fifth activity, the impact of the prototype on addressing the desired situation was analyzed. Parallel execution of the activities was performed for more efficient activity execution [48].

IV. Results

A. Evaluation of WebSocket Technology

Seven VGAPs were selected based on the relevancy of the principles with regard to WebSocket technology. The following themes were categorized based upon the similarity of the insights gained from the selected VGAPs and their importance to Volvo group. For instance, the insights from comformity to standards, security, and documentation are discussed as one theme in standardization of WebSocket technology. The gained insight of security principle is in this category since the principle meant to address the current known security standards rather than the security of the written code. Similar analogy was followed for the rest of the themes.

1) Standardization of WebSocket Technology: WebSocket is first specified as part of HTML5 specification for achieving real-time connectivity in modern HTML5 applications. Currently, WebSocket involves two standard specifications: WebSocket Protocol and WebSocket API [10]. The protocol is developed by Internet Engineering Task Force (IETF). IETF has standardized WebSocket Protocol on December 2011 with publishing number RFC 6455 [11]. The protocol is under the state Proposed Standard up on writing this report, which means that the protocol is relatively stable [11]. However, IETF recommends not implementing a protocol on a Proposed Standard state on a disruptive-sensitive environment [12]. WebSocket API, which enables use of WebSocket protocol, is standardized by World Wide Web Consortium (W3C). The technical report drafted by W3C for WebSocket API is under Candidate Recommendation stage [13], which means that W3C believes the API is stable and appropriate for implementations, having in mind that it may still change based on implementation experiences [14].

RFC 6455 WebSocket protocol has specified how WebSocket handshake will be performed securely by using Globally Unique Identifier (GUID). Upon receiving a handshake upgrade request, a server needs to concatenate Sec-WebSocket-Key header value with GUID, transform the result to Base64-encoded SHA-1 hash, and send the result as the value of Sec-WebSocket-Accept header [11]. The client endpoint, then, will perform the same algorithm to compare the returned key before upgrading the connection. This mechanism enables WebSocket protocol to be preventive for cross-protocol attacks, which will protect those server endpoints, which are unaware of WebSocket, from such attacks since such servers do not have knowledge about GUID of the WebSocket protocol [10].

In addition, RFC 6455 WebSocket protocol specifies an origin header where web browsers can set their origin information during WebSocket handshake. It states that if the origin indicated is unacceptable to the server, then it should respond to WebSocket handshake with a reply containing HTTP 403 Forbidden status code [11]. This will protect browser-server WebSocket connection from Denial of Service (DoS) attack [10]. Even though the origin header prevents DoS attack on HTTP layer, the DoS threat can still persist in TCP layer. WebSocket API provides a solution for such attacks by letting WebSocket constructor to open a new Network Socket each time a connection is established, so that the browser itself will take the responsibility to limit the number of TCP sockets opened to a particular host [10]. However in terms of non-browser client applications, the origin header will not protect from DoS, because the non-browser client application can open sockets with different origins [10].

Man-in-the-middle can induce malicious HTTP requests in WebSocket frame. Such an attack will cause HTTP cache poisoning, especially on transparent proxy servers [10]. Cache poisoning used to be the major security issue during the standardization of WebSocket protocol [15]. To protect the server endpoints and EWMI from such attacks, RFC 6455 WebSocket protocol implements Masking of the payload data in WebSocket frames. Each frame through the wire is masked with a new Masking Key, using an algorithm which is difficult to be predicted by man-in-the-middle [11].

It can be easily noted that the masked data can be understood by a WebSocket-aware man-in-the-middle. Therefore, it worths to authenticate the transmitted data in order to ensure secure data transmission. In terms of authenticating WebSocket client endpoint, WebSocket protocol has given the freedom for applications to implement any of those standard authentication mechanisms used by HTTP servers such as cookies, HTTP authentication, or TLS authentication. Connection confidentiality and integrity is guaranteed by RFC 6455 WebSocket protocol, since it supports implementation of connections over Transport Layer Security (TLS). Such connections use WSS URIs, which is similar to HTTP over TLS. But, it is important to notice that the security provided by TLS depends greatly on the strength of the algorithms negotiated during TLS handshake [11]. W3C provided the criteria on the strength of TLS algorithms on REC-wsc-ui-20100812 [16]. WebSocket protocol emphasizes that both the client and server endpoints must validate any incoming data for violation of protocol-wise and/or application-specific criteria which determines safety of input data.

2) Effect of WebSocket in Enterprise Application Performance: Performance is becoming one of the main requirements in current applications. Similarly, distributed applications are showing more and more interests in to high-performance applications. In order to have high-performance application, different performance domains in an application should be considered, specifically in terms of enterprise distributed applications. Considerations regarding Network Bandwidth are relatively important when it comes to distributed applications performance. Network Bandwidth represents the overall capacity of the connection; the greater the capacity, the more likely that better performance will result [17]. Another important aspect in data transmission domain is Latency [18]. To a large extent, low Latency has become considerably important from points of view of users in order for a faster web prefetching [18] [19]. Low Latency can be achieved by employing proper protocols and standards while designing enterprise distributed applications.

WebSocket protocol provides a faster communication in distributed applications. It introduces a dramatic reduction of Network Bandwidth by eliminating the unnecessary HTTP header data. The size of this HTTP header, depending on the application, can reach up to 2000 bytes per each HTTP request. WebSocket protocol, instead of HTTP headers, adds only two bytes overhead [20] [21]. There have been many studies performed revealing the bandwidth reduction. For instance, [9] demonstrates a 500-to-one or 1000-to-one bandwidth reduction in terms of employing WebSocket protocol. This reduction of unnecessary HTTP headers becomes significantly important when a distributed application is servicing to a large number of users. To build on top of that, [22] reveals a comparison of stock web application employing HTTP polling and WebSocket. In this demonstration, the stock information sent from the server to the client is 20 characters and the request and response headers together contain 871 bytes of information. Table II shows the result of an experiment on bandwidth comparison with regard to different numbers of client.

Another dramatic improvement, while employing WebSocket rather than HTTP, is the bi-directional communication over a single persistent connection [10] [23]. Bi-directional communication enables the data pushes from both client and server endpoints at any time, in contrast with HTTP solutions which are based on request-response. In this manner, the transmitting time for an HTTP request will be cut out and the server endpoint will be able to push the updated data to client endpoints. This bi-directional nature of WebSocket protocol not only reduces Latency from 150 milliseconds, which is in current HTTP solutions, to 50 milliseconds, but also reduces the server memory and CPU consumption [21]. This three-to-one reduction of Latency has a substantial contribution in improving the performance of distributed applications where the update frequencies are required to be 10 to 500 milliseconds [9] [24].

3) Implementation Simplicity of WebSocket in Enterprise Architecture: One of the basic focuses of distributed enterprise application architecture, nowadays, is the simplicity of inter-component interaction. In addition to binary formats, WebSocket protocol supports messages in the text format (UTF-8) [11]. This capability of the protocol adds a significant advantage in simplifying the interaction between distributed enterprise applications. On the other hand, the standardization of WebSocket API plays a significant role towards the simplicity concept. WebSocket API is designed to make use of WebSocket protocol so much simpler. It encapsulates most of connection oriented computation away from the application developer. For instance, instantiating WebSocket constructor with WS or WSS Uniform Resource Locator (URL) is enough for an application developer to accomplish WebSocket handshake and upgrade the connection to WebSocket [13]. Besides, the standard is designed in event-driven manner, which makes the interaction process asynchronous and simpler. Table III discusses the event handlers with their responsibility.

Enterprise distributed applications use known frameworks and efficient programming languages in order to satisfy the needs. Among different frameworks, Java Enterprise Edition (JEE) and ASP.NET stand out. These two frameworks support WebSocket technology by providing the underlying API implementations for developers in order to simply use WebSocket.

WebSocket API implementations are part of new proposed JEE Specification, Version 7 [25]. An enterprise developer can simply employ WebSocket technology by either programmatically implementing or using annotations. In order to implement WebSocket server endpoint, both ways are effectively simple.

TABLE II
COMPARISON OF HTTP POLLING AND WEBSOCKET [22]

Number of clients	Polling over HTTP	WebSocket
1000	6,968,000 bits per second (6.6 Mbps)	16,000 bits per second (0.015 Mbps)
10000	69,680,000 bits per second (66 Mbps)	160,000 bits per second (0.153 Mbps)
100000	696,800,000 bits per second (665 Mbps)	1,600,000 bits per second (1.526 Mbps)

TABLE III
WEBSOCKET API EVENT HANDLERS

Event Handler	Event
onopen	Invoked when the WebSocket connection is cleanly established for the specific WebSocket object instance.
onmessage	Invoked when a WebSocket message is received. Parsing of the WebSocket Frame is encapsulated from the application developer.
onclose	Invoked when a WebSocket closing handshake is received or when WebSocket object instance fail the connection due to other reasons.
onerror	Invoked when the buffer memory is full while sending message.

In programmatic approach, WebSocket application is defined by extending Endpoint class and overwriting certain APIs, for instance onMessage(), depending on application behavior. In annotation approach, the proper annotations are employed on Plain Old Java Object (POJO) which adds the desired behavior at run-time [26]. The fact that Java WebSocket API are corresponding to the standard WebSocket APIs makes it effectively easy for developers and also allows developers to perform certain configurations. One of the most important configurations, considering enterprise distributed applications, is WebSocket handshake modication. This configuration can be set (by calling modifyHandshake() method) to access HTTP cookie, or any other header, in handshake request. In addition to standard APIs, Java WebSocket API adopts session management per each client endpoint. This session management feature enables clients properties observation (by calling getUserProperties() method) in order to associate specific application behavior with a particular session [26].

As of JEE, ASP.NET framework provides implementation of WebSocket API in a form of an abstract class called WebSocketHandler [27]. To make session management easy and simple, the framework provides one WebSocketHandler instance per each WebSocket connection. WebSocketHandler implementation has a session management static component, WebSocketCollection, where the instances can be added and removed. The collection provides an API called Broadcast() which makes sending message to WebSocket instances so simple and easy. As illustrated in the following snippet code, the framework encapsulates WebSocket Handshake to be seamlessly integrated with HTTP listener implementation (IsWebSocketRequest and AcceptWebSocketRequest) [27].

```
public void Process_Req(HttpContext con)
{
    if(con.IsWebSocketRequest)
    {
     con.AcceptWebSocketRequest
        (new W_HandlerImplementation());
    }
}
```

It is important to notice here that once WebSocket Handshake is completed by HTTP listener implementation, ASP.NET framework completely decouples WebSocket communication in Websockethandler instance.

The simplicity of WebSocket implementation in distributed applications has significant contributions in maintainability as well. The isolation of WebSocket API with different responsibilities also facilitates developers to easily detect a problem or extend the application. In addition, the fact that WebSocket API is designed in event-driven manner, significantly contributes in loose coupling of WebSocket implementation from other components.

4) Effect of WebSocket on Enterprise Application Robustness: Robustness is essentially an important aspect in enterprise applications where the applications should continue performing well despite changes in operating environment [28]. The importance of the robustness becomes even more considerable in enterprise distributed applications where the distributed applications including software and hardware components are connected to each other via a network such as Internet. Regarding the distributed applications, protocols have significant effect on applications robustness. However, the applications itself could be designed in such a way that they are able to cope with unpredictable and unusual situations such as network disconnectivity, invalid input-data, limited resources, etc. Network disconnectivity is relatively an important issue when it comes to WebSocket since WebSocket uses persistent TCP connection. In other words, the presence of persistent TCP connection is a must in order to send and receive messages using WebSocket protocol.

WebSocket devoted several attributes and events, in addition to onerror API function, in order to increase the robustness of the application concerning network connectivity. For instance, readyState and bufferedAmount attributes could contribute in application robustness. ReadyState attribute indicates the status of the connection [10]. This attribute indicates different values depending on WebSocket status. The value of this attribute could be checked each time before calling the send API. If the connection has not been established yet, the data can be, for instance, buffered in the application. Table IV contains the different possible values of readyState attribute.

Another important attribute is bufferedAmount. This attribute represents the buffered data size to be sent over Internet [10] [13]. On behalf of the application, browsers will buffer the outgoing data, after calling WebSocket send API [10]. This attribute could be used to control the rate of sending data in order to prevent network saturation [10].

As indicated in table IV, close event is triggered in onclose API when WebSocket connection is either expectedly or unexpectedly closed. This event has three main properties: wasClean, code, and reason. Each of these event properties fundamentally facilitate error-handling. In brief, wasClean indicates the connection closure status which is either expected or unexpected due to underlying TCP connection, for instance. Code property represents the closure code, which is a numerical code indicating the reason for closure. And finally, reason property represents the additional explanation for the connection closure.

Another important WebSocket property is Ping-and-Pong frames as defined in the RFC 6455 WebSocket protocol [13]

TABLE IV
DIFFERENT VALUES OF READYSTATE ATTRIBUTE

Constant Value	Numeric Value	Description
CONNECTING	0	Connection is in progress but not established yet
OPEN	1	Connection has been established. Messages can be sent and received
CLOSING	2	Closing connection is in progress.
CLOSED	3	Connection has been closed or could not be open.

[11]. Ping-and-Pong frames initiated by WebSocket endpoints are significantly important in order to avoid the disconnection because of persistent idle TCP connection. Ping-and-Pong frames going back and forth reveal health of application endpoints [10]. Accordingly, this property contributes in application robustness since the non-responsive Ping frame indicates an issue and triggers close event with corresponding connection close code.

Simply put, the properties, attributes and events in WebSocket API together empower the robustness of the application when unexpected situations appear, specifically network disconnection.

5) Contribution of WebSocket on Enterprise Applications Maintenance: Flexibility and extensibility of architectural design in distributed enterprise applications are important for the applications maintenance. The philosophy behind the design of WebSocket protocol hugely emphasizes on these behaviors. WebSocket protocol is designed to be low level, asynchronous communication protocol just over TCP while still maintaining the application layer promises, which includes identifying communication partners, determining resource availability, and synchronizing communication [11][29]. This nature enables enterprise architects to design simple higher level subprotocol over WebSocket. RFC 6455 WebSocket protocol already specifies a header called Sec-WebSocket-Protocol where client endpoints list their wish-list of subprotocols. The server endpoint can choose one of the subprotocols, or can even decline not to use subprotocol at all, by using the same header during WebSocket handshake. Extending and maintaining WebSocket-based web applications will be fairly easy since the authority to change, scale out, or even decline the subprotocol is in the hands of the enterprise architect.

Moreover, the text based (UTF-8) data format of WebSocket frame payload significantly facilitates scaling out of text-based data serialization formats. If, for some reasons, a WebSocket-based web application is required to extend or completely change the text-based data serialization format, it can be easily accomplished by modifying the onmessage API in WebSocket endpoints.

B. Findings from the WebSocket Prototype

A simple WebSocket-based application prototype was first developed for the purpose of evaluating the communication through the simulated laboratory environment. As first iteration, both the prototype and Nginx reverse proxy server was configured to accept normal HTTP requests. As discussed in section 2.A, the Nginx reverse proxy server did not allow the client and server endpoints to establish WebSocket connection. Further analysis of the network traffic revealed that when the HTTP request arrived at the server endpoint, the HTTP request did not have upgrade:websocket header. In addition, it was noticed that Nginx reverse proxy server changed the connection:upgrade header to connection:close. Figure 3 and Figure 4 illustrates the HTTP request received by the server endpoint before and after configuring Nginx reverse proxy respectively.

Notably, the main reason for the WebSocket connection failure of the first iteration was the modification of HTTP header. As many studies recommend [34] [10] transmitting encrypted data over Transport Layer Security (TLS) is fundamentally a good design recommendation in order to prevent the HTTP headers modification by EWMIs. To resolve the problem, the WebSocket-based application prototype and the Nginx reverse proxy server was configured to transmit encrypted data over TLS by using WSS URL. However, in contrast with the theoretical recommendations, WebSocket connection was not established between the client and server endpoints. As the first iteration, the same modification of HTTP request headers was observed during further analysis of the network traffic.

V. Discussion

Based on the finding of this study, the authors believe that WebSocket technology looks very promising for real-time distributed applications. Both the protocol and API of WebSocket are relatively stable to be used by enterprise architects. WebSocket not only contributes into the less data transmission over the network, but also makes the development significantly simpler for developers by abstracting sending and receiving data. A developer can easily call API functions in server and client endpoints in order to send the data. The data can be sent at anytime in a bi-directional manner. Also, WebSocket makes the real-time applications more robust and maintainable by devoting specific properties and events.

Even though WebSocket technology has many advantages, the finding from design research revealed that the concerns of WebSocket communication considering EWMIs are literally the major bottlenecks. According to the finding, the Nginx reverse proxy server is incompatible when employing WebSocket technology. The Nginx reverse proxy server adds conection:close on HTTP request during WebSocket handshake. As the standard of HTTP/1.1 states [53], a server endpoint must close the connection when receiving connection:close header. As discussed in Table I in theoretical background, this behavior does not allow persistent connection to

upgrade	websocket
connection	Upgrade
host	localhost:8181
origin	https://localhost:8181
pragma	no-cache
cache-control	no-cache
sec-websocket-key	unxPs8xY5WhzwtLW5DErfq==
sec-websocket-version	13
sec-websocket-extensions	x-webkit-deflate-frame
user-agent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (K...
cookie	JSESSIONID=051e569d9197b6d70bbd57bbccee; USER_...

Fig. 3. A Successfull HTTP Request Header for WebSocket Handshake

host	129.16.238.163
x-real-ip	217.210.248.194
x-forwarded-for	217.210.248.194
x-forwarded-proto	https
connection	close
user-agent	Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:19.0) Gecko/20...
accept	text/html,application/xhtml+xml,application/xml;q=0.9,...
accept-language	en-US,en;q=0.5
sec-websocket-version	13
origin	https://129.16.238.163
sec-websocket-key	M45MTSi0ZDFScM34kTNBUw==
cookie	JSESSIONID=055f7cefbb1f718c7a19407f3d3d
pragma	no-cache
cache-control	no-cache

Fig. 4. Unsucessful HTTP Request Header for WebSocket Handshake

be established. Moreover, Nginx reverse proxy server removes the connection:upgrade and upgrade:websocket header. In this case, the server endpoint will never receive the WebSocket connection handshake request. The authors think that these specific behaviors of Nginx reverse proxy server, more or less, will be reflected on other reverse proxy servers.

Most IT artifacts of Volvo IT, one way or another involves real-time data to be transmitted between components. From the discussions and documents such as VGTA, the authors understand that the enterprise applications at Volvo IT are more geared towards component-based and distributed architecture which use OSI-based computer network as a basic means of communication. The authors believe that using WebSocket technology for the real-time communication between components will save lots of time and money. One scenario could be Volvo Yard Management System which facilitates the delivery of stocks in a more effective way. In this scenario, WebSocket will have substantial impact on improving the real-time communication among different components in the system.

There are some home-works for Volvo IT in order to benefit from WebSocket technology. The most significant challenge will be regarding their reverse proxy server, and in general the EWMIs. The compatibility of WebSocket technology with regard to EWMIs must be researched more. The authors suggest that further researches should be focused on making the EWMIs compatible for WebSocket technology. For instance, Nginx reverse proxy server (starting from version 1.3.13) can be configured to be compatible with WebSocket by adding few commands to the default configuration file (see Appendix) [54]. Furthermore, as pointed out on the third theme, WebSocket API standard is already implemented on the latest versions of major platforms such as JEE-7 and ASP.NET. However, based on the understanding of documentations and discussions, Java for Volvo Systems (JVS) makes use of JEE-6 API implementations. To produce more consistent and maintainable WebSocket-based applications, Volvo IT will need to upgrade Java application servers, such as Glassfish or Apache Tomcat, to the version which implement JEE API for WebSocket.

VI. Conclusion

Inevitably, the use of real-time data by web applications will be increasing in the coming years. One of the important technologies which come with real-time web applications is WebSocket technology. This study has investigated two important concerns regarding adoption of WebSocket technology in enterprises. The first concern was how mature WebSocket technology is when evaluated based on enterprise architectural principles. The second concern was how to resolve the challenges of WebSocket technology coming from EWMIs. Based on the nature of the two concerns, the study employed two research approaches as research strategy, namely qualitative approach and design research approach. In terms of the first concern, the study found that WebSocket technology is mature enough to be used by enterprises. However, the study did not find a design recommendation in client and server endpoints which could resolve the challenges coming from traversal of EWMIs. The outcomes from the study indicated that enterprises, such as Volvo IT, need to carefully consider on conducting more research on their EWMI architecture in order to make them compatible with WebSocket technology. Considering the advantages Volvo IT would gain from WebSocket technology, the authors strongly recommend Volvo IT to take a management decision for the required changes and further researches for supporting WebSocket technology.

Acknowledgment

This research study was made possible through the help and support from everyone, including: supervisors, family and friends. Especially, we would like to dedicate our acknowledgment of gratitude toward the following significant advisors and contributors.

First and foremost, we would like to thank our supervisor Morgan Ericsson for his valuable support and encouragement. Second, we would like to thank our industry contact Micael Andersson for devoting time and resources.

Finally, we sincerely thank to our family, and friends who provide the advice and financial support. Sepcifically Amir Hossein Tayarani Yoosefabadi, your advice to choose this research area is valueable. The product of this research study would not be possible without all of them.

References

[1] GABRIEL, V. and FEDOR, L. *Modern approach in multiple patients ECG monitoring*. Biomedical and Health Informatics (BHI), 2012 IEEE-EMBS International Conference on, 5-7 Jan. 2012 2012. 131-134.

[2] CHA, S.-H. 2013. *Developing a HTML5-Based Real-Time Monitoring System for Wireless Sensor Networks*. In: JIN, D. and LIN, S. (eds.) Advances in Mechanical and Electronic Engineering. Springer Berlin Heidelberg.

[3] AGHAEE, S. and PAUTASSO, C. 2010. *Mashup development with HTML5*. Proceedings of the 3rd and 4th International Workshop on Web APIs and Services Mashups. Ayia Napa, Cyprus: ACM.

[4] WESSELS, A., PURVIS, M., JACKSON, J. and RAHMAN, S. *Remote Data Visualization through WebSockets*. Information Technology: New Generations (ITNG), 2011 Eighth International Conference on, 11-13 April 2011 2011. 1050-1051.

[5] MARION, C. and JOMIER, J. 2012. *Real-time collaborative scientific WebGL visualization with WebSocket*. Proceedings of the 17th International Conference on 3D Web Technology. Los Angeles, California: ACM.

[6] BIJIN, C. and ZHIQI, X. *A framework for browser-based Multiplayer Online Games using WebGL and WebSocket*. Multimedia Technology (ICMT), 2011 International Conference on, 26-28 July 2011 2011. 471-474.

[7] KURYANOVICH, E., SHALOM, S., GOLDENBERG, R., PAUMGARTEN, M., STRAU, D., LEE-DELISLE, S., RENAUDEAU, G., WAGNER, J., BERGKNOFF, J., DANCHILLA, B. and HAWKES, R. 2012. *A Real-Time Multiplayer Game Using WebSockets*. HTML5 Games Most Wanted. Apress.

[8] KIM, Y.-H., LIM, I.-K., KANG, S.-G. and LEE, J.-K. 2011. *Mobile Cloud e-Gov Design and Implementation Using WebSockets API*. In: PARK, J., YANG, L. and LEE, C. (eds.) Future Information Technology. Springer Berlin Heidelberg.

[9] LUBBERS, P., ALBERS, B. and SALIM, F. 2011. *Using the WebSocket API*. Pro HTML5 Programming. Apress.

[10] WANG, V., SALIM, F. and MOSKOVITS, P. 2013. *The Definitive Guide to HTML5 WebSocket*. Apress.

[11] IETF. 2011. *The WebSocket Protocol RFC 6455* [online] [Accessed 2013-04-04] http://datatracker.ietf.org/doc/rfc6455/

[12] BRADNER, S. 1996. *The Internet Standards Process – Revision 3* [online] [Accessed 2013-04-04] http://www.rfc-editor.org/rfc/rfc2026.txt
[13] W3C. 2005. *World Wide Web Consortium Process Document* [online] [Accessed 2013-04-04] http://www.w3.org/2005/10/Process-20051014/tr
[14] W3C. 2005. *World Wide Web Consortium Process Document* [online] [Accessed 2013-04-04] http://www.w3.org/2005/10/Process-20051014/tr
[15] HUANG, L.-S., CHEN, E. Y., BARTH, A., RESCORLA, E. and JACKSON, C. 2011.*Talking to yourself for fun and profit*. Proceedings of W2SP.
[16] W3C. *Web Security Context: User Interface Guidelines* [online] [Accessed 2013-04-04] http://www.w3.org/TR/wsc-ui/
[17] MITCHELL, B. *Introduction to Computer Network Speed* [online] [Accessed 2013-04-08] http://compnetworking.about.com/od/speedtests/a/network-speed.htm
[18] OSSA, B., SAHUQUILLO, J., PONT, A. and GIL, J. 2012. *Key factors in web latency savings in an experimental prefetching system*. Journal of Intelligent Information Systems, 39, 187-207.
[19] GRIGORIK, L. 2012. *Latency: The New Web Performance Bottleneck* [online] [Accessed 2013-04-08] http://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/
[20] 2009. *Web socket protocol in "last call"?* [online] [Accessed 2013-04-05] http://www.ietf.org/mail-archive/web/hybi/current/msg00784.html
[21] LUBBERS, P., ALBERS, B. and SALIM, F. 2010. *Using the HTML5 WebSocket API*. Pro HTML5 Programming. Apress.
[22] LUBBERS, P. and GRECO, F. 2010. *HTML5 WebSockets: A Quantum Leap in Scalability for the Web, SOA World Magazine* [online] [Accessed 2013-04-08] http://soa.sys-con.com/node/1315473
[23] CASARIO, M., ELST, P., BROWN, C., WORMSER, N. and HANQUEZ, C. 2011. *HTML5 WebSocket*. HTML5 Solutions: Essential Techniques for HTML5 Developers. Apress.
[24] PIMENTEL, V. and NICKERSON, B. G. 2012. *Communicating and Displaying Real-Time Data with WebSocket*. Internet Computing, IEEE, 16, 45-53.
[25] DEMICHIEL, L. and SHANNON, B. 2013. *Java Platform, Enterprise Edition (Java EE) Specification, v7*. [online] [Accessed 2013-04-12] http://download.oracle.com/otn-pub/jcp/java_ee-7-pfd-spec/JavaEE_Platform_Spec.pdf?AuthParam=1365752162_0418bd315257ed35a26d0faecfd70f9d
[26] DOBRIC, D.2012. *WebSockets in ASP.NET and JavaScript* [online] [Accessed 2013-04-12] http://developers.de/blogs/damir_dobric/archive/2012/01/29/websockets-in-asp-net-and-javascript.aspx
[27] COWARD, D. 2013. *Java API for WebSocket*. [online] [Accessed 2013-04-12] http://download.oracle.com/otn-pub/jcp/websocket-1_0-pfd-spec/JSR356_ProposedFinalDraft1.pdf?AuthParam=1365847983_0cc3cb5e43fdba0dd82015b8b9a46cd9
[28] VOGEL, J. and WIDMER, J. 2008. *Robustness in Network Protocols and Distributed Applications of the Internet*. In: SCHUSTER, A. (ed.) Robust Intelligent Systems. Springer London.
[29] Wikipedia, 2013. *OSI model* [Accessed 2013-03-19] http://en.wikipedia.org/wiki/OSI_model
[30] WAKAMIYA, N., MURATA, M. and MIYAHARA, H. 2002. *On Proxy-Caching Mechanisms for Cooperative Video Streaming in Heterogeneous Environments*. In: ALMEROTH, K. and HASAN, M. (eds.) Management of Multimedia on the Internet. Springer Berlin Heidelberg.
[31] Wikipedia, 2013. *Firewall* (computing) [Accessed 2013-03-27] http://en.wikipedia.org/wiki/Firewall_(computing)
[32] 2013. *What is Enterprise Application* [online][Accessed 2013-03-27] http://msdn.microsoft.com/en-us/library/aa267045(v=vs.60).aspx
[33] OKAMOTO, T., TERAI, T., HASEGAWA, G. and MURATA, M. 2006. *A Resource/Connection Management Scheme for HTTP Proxy Servers*. In: GREGORI, E., CONTI, M., CAMPBELL, A., OMIDYAR, G. and ZUKERMAN, M. (eds.) NETWORKING 2002: Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications. Springer Berlin Heidelberg.
[34] LUBBERS, P .2010.*How HTML5 Web Sockets Interact With Proxy Servers*[online][Accessed 2013-02-25] http://www.infoq.com/articles/Web-Sockets-Proxy-Servers
[35] FU-XIANG, G., YU, Y., QIONG, L., LIANG, L., LAN, Y. and ZHIBIN, Z. 2004. *Design and implementation of A bi-directional proxy server*. Wuhan University Journal of Natural Sciences, 9, 760-764.
[36] BASS, L., CLEMENTS, P. and KAZMAN, R. 2003. *Software Architecture in Practice* (2nd ed.). Boston, MA, USA: Addison-Wesley.
[37] DRIJVERS, P.H.M., 2003. *Learning algebra in a computer algebra environment*. Utrecht: CD- Press, Center for Science and Mathematics Education.
[38] SAINI, K. 2011.*Squid Proxy Server 3.1: Beginner's Guide*. Birmingham Packt Publishing.308p
[39] MAIER, R., HDRICH, T. and PEINL, R. 2005. *Infrastructure*. Enterprise Knowledge Infrastructures. Berlin.
[40] WESSELS, D., 2004. *Squid: The Denitive Guide*. OReilly,
[41] NECKER, M., SCHARF, M. and WEBER, A. 2005. *Performance of Different Proxy Concepts in UMTS Networks*. In: KOTSIS, G. and SPANIOL, O. (eds.) Wireless Systems and Mobility in Next Generation Internet. Springer Berlin Heidelberg.
[42] YU, G., YUKIO, H., and TAKAO, A. 1998. *Autonomic buffer control of web proxy server*. Worldwide Computing and Its ApplicationsWWCA'98, 428-438.
[43] Wikipedia, 2013. *Proxy Server* [Accessed 2013-03-19] http://en.wikipedia.org/wiki/Proxy_server
[44] VACCA, R. 2007. *Protecting Servers and Clients with Firewalls*. Practical Internet Security. Springer US.
[45] WILLY, T. 2006. *Making applications scalable with Load Balancing* [online][Accessed 2013-03-27] http://1wt.eu/articles/2006_lb/index.html
[46] KARAKOS, A., PATSAS, D., BORNEA, A. and KONTOGIANNIS, S. 2005. *Balancing HTTP Traffic Using Dynamically Updated Weights, an Implementation Approach*. In: BOZANIS, P. and HOUSTIS, E. (eds.) Advances in Informatics. Springer Berlin Heidelberg.
[47] GILLY, K., JUIZ, C. and PUIGJANER, R. 2011. *An up-to-date survey in web load balancing*. World Wide Web, 14, 105-131.
[48] BLESSING, L. and CHAKRABARTI, A.2009. *DRM: A Design Research Methodology*. Springer London.
[49] BLESSING, L. *A Design Research Methodology* [online]. Available at: http://www.tu-berlin.de/fileadmin/fg89/PDFs/Forschung/Flyer_Blessing_en.pdf [Accessed February 20, 2013]
[50] 2013. *Nginx* [Accessed 2013-05-22] http://nginx.org/
[51] 2013. *Wireshark* [Accessed 2013-05-22]http://www.wireshark.org/
[52] STROBL, R.2006. *Monitoring HTTP Communication* [Accessed 2013-05-22]https://blogs.oracle.com/roumen/entry/netbeans_quick_tip_29_monitoring
[53] W3C. *Header Field Definitions: Hypertext Transfer Protocol – HTTP/1.1* [online] [Accessed 2013-05-22] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html
[54] Nginx. *WebSocket proxying* [online] [Accessed 2013-05-22] http://nginx.org/en/docs/http/websocket.html
[55] Wikipedia. *Application layer* [online] [Accessed 2013-06-01] http://en.wikipedia.org/wiki/Application_layer

APPENDIX

The following snippet configuration command is used by Nginx Proxy server to make it compatible to WebSocket

```
location /chat/ {
    proxy\_pass http://backend;
    proxy\_http_version 1.1;
    proxy\_set\_header Upgrade $http_upgrade;
    proxy\_set\_header Connection "upgrade";
}
```