Dipen Hamal

Near Real-Time Communication in the WWW

Extensible Messaging and Presence Protocol

Helsinki Metropolia University of Applied Sciences
Bachelor of Engineering

Media Engineering

Bachelor's Thesis

21 February 2012

Author(s) Title Number of Pages Date	Dipen Hamal Near real time communication in the World Wide Web (WWW) 49 pages + 4 appendices 28 March 2012
Degree	Bachelor of Engineering
Degree Programme	Media Engineering
Specialisation option	Java and Dot Net
Instructor(s)	Kari Aaltonen, Thesis Supervisor & Principal Lecturer Hannu Markanen, Supervisor & Lecturer

The goal of this project was to research real time communication protocol and the possibility of replacing flash media server with an existing open source server. Extension of the protocol, its extensibility and performance of the open source real time server were analyzed. This project was carried out for Metropolia UAS in order to find out alternative solutions for the eLearning environment project called Knowledge Practice Laboratory (KP-Lab), running on flash media server.

The client side of the project was a browser based cross-platform application fully implemented in actionscript 3. In order to run the application, browsers should have the latest version of flash player installed. For the server side implementation and overriding the server's functionality, Java programming language was used. The project used the schema provided by the server for storing information into database and it was a near real time application.

The application was tested using all the major browsers: Mozilla, Firefox and Chrome. Performance test of the server was done by registering over two thousand users and running the automated test. The tests verified the server to be fully developed and mature enough to be taken into consideration as a better alternative to flash media server.

Keywords	XMPP Protocol, Server, plugin, extensibility, real time, KP-lab

Contents

List of Abbreviations

Adobe FB	Adobe Flash Builder
AIM	AOL (America on Line) Instant Messenger
API	Application Programming Interface
CPU	Central Processing Unit
DNS	Domain Name System
FTP	File Transfer Protocol
GSSAPI	Generic Security Service Application Program Interface
GUI	Graphical User Interface
HSQLDB	HyperSQL Database
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol over SSL
IBM	International Business Machine
IDE	Integrated Development Environment
IM	Instant Messaging
IMAP	Internet Mail Access Protocol
IP	Internet Protocol
JAR	Java Archive
JOSSO	Java Open Single Sign On
KPE	Knowledge Practice Environment
KP-Lab	Knowledge Practice Laboratory
MD5	Message Digest Algorithm
ms	millisecond
MSN	MicroSoft Network
MXML	Macromedia eXtensible Markup Language
OSI	Open Systems Interconnection
POP	Post Office Protocol
REST	Representational State Transfer
RFC	Request for Comments
RPC	Remote Procedure Calls
RTMP	Real Time Messaging Protocol
RTMPT	Real Time Messaging Protocol Tunneled
RTMPS	Real Time Messaging Protocol over SSL (Secure Socket Layer)
RTMPE	Real Time Messaging Protocol Encrypted

RTMPTE	Real Time Messaging Protocol Encrypted and Tunneled
SASL	Simple Authentication and Security Layer
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSO	Single Sign-On
TCP	Transmission Control Protocol
TLS	Transport Layer Security
WWW	World Wide Web
XAMPP	X (any of these operating systems: Linux, Mac OS X, Solaris, and Windows), Apache, MySQL, PHP and Perl
XEPs	XMPP Extension Protocol
XHTML	Extensible Hyper Text Transfer Protocol
XML	EXtensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
XSF	XMPP Standards Foundation

1 Introduction

The main aim of the thesis is to explore the already existing real-time communication protocol, Extensible Messaging and Presence Protocol (XMPP) that is considered to be the alternative to Real Time Messaging Protocol (RTMP) used by flash media server. Another objective of the thesis is to determine whether Openfire, real time collaboration server that has built in support for XMPP is suitable for KP-lab and capable of replacing flash media server.

This thesis includes the research study associated with the KP-lab, near real-time online collaborating tool. KP-lab uses flash media server to broadcast messages and as sync server for overall synchronization. KP-lab intends to replace flash media server with an existing open source real time server.

People have always communicated. However the communication channel and the way people communicate has always changed. The latest trend, Internet has always been able to attract more users due to its simplicity to use and continuous development. Merely used to share static information via web pages in the beginning, Internet now is used for communicating all sorts of information and in a dynamic manner. Excessive uses of social networking, online collaboration tools and global concentration on it for sharing information have outlined a need for faster and reliable communication. Near real time communication is a need that should be achieved without compromising security and reliability.

The thesis is divided into two main parts, one theoretical and one practical. Theoretical part consists of research on XMPP and how it varies from existing protocols. Online articles, books and research studies are used for this part. Practical part consists of application design that supports the protocol and uses Openfire as a communication server. This part is used to evaluate the extensibility of the protocol and overall performance of the server. Tests will be carried out to determine the performance capability of the server and obtained results will be used as evaluation criteria.

The thesis will not describe the application development and implementation of the protocol in depth. Evaluation will be done upon the assumption that flash media server and Openfire server have same configuration and are used under similar conditions.

2 What is Near-Real Time Communication?

Real time communication system is anything that allows or guarantees instant delivery of whole information. Since the communication system depends on number of components and their performance, it is impossible to achieve any system that has no latency, the time required for receiving input and responding to the received input. Due to this fact, the scope of this project is limited towards achieving near real time communication.

Near real time communication over the Internet is sharing information to and collaborating with others in almost real time depending on the presence of the associated users. Any application or tools that are near-real time in nature offers synchronization, faster response time and reliability. Latency should be comparatively low in order to be near-real time. Also, merely being faster does not make any application real time until and unless it is reliable. Data lost during the transmission or lesser throughput should be extremely low or completely avoided under any circumstances.

2.1 Scope of Near-Real Time Communication

It is an undeniable fact that the one who lacks information lags behind. Unlike centuries ago, when information was limited to certain group of scholars and nobles, these days there is vast amount of information freely and easily accessible on the Internet. Everyone who is connected to the Internet has access to information, however, the one who gets information faster might get all the benefits. Vast amount of information is being exchanged on the Internet and since the life is getting faster, these days getting information on time matters the most.

Information delivered on time can be a life saver and might have more use than delivered late. Some of the information is always useful, however, some information that is delivered late might just be the reference from the past. In order to preserve the importance of the information and maximize its usefulness, timely delivery of information is very important.

Near-real time communication is critical in many sectors where timely information is essential. Aeronautical, space administration, finance and health sector are some of the sectors where delay in information is unacceptable. A slight change in the shared information might deviate result into creating havoc.

Instant messaging, presence awareness technology, online gaming, application sharing, desktop sharing, voice over IP, video and audio conferencing tools are some of the real time collaboration tools [1]. Using these tools, users sense or get result immediately in timely manner and data transmission without noticeable loss or no loss at all.

2.2 Prerequisites of Real-time Communication System

Near Real time communication is said to be achieved only when the exchange of messages can be executed with in fraction of a second. This can happen only when the processing time is very short. In order to do so, the system should be able to process messages without requirement to store them or to perform any operations. Additional latency caused due to polling can add to processing time completely slowing down the system. The completely active system that incorporates event/data-driven processing capabilities can minimize the need for polling there by reducing the latency period. [2, 2-4].

Processing of data before storing makes the system faster. Compared to conventional database which queries only the existing data, real time system has to deal with data in the fly. This might push the system to wait for longer time or into loophole in case the data received is delayed, truncated or out of sequence. To avoid this, the system should be able to time out individual processing in order to unblock the delayed operation and handle the imperfect streams. [2, 2-4].

Another important feature that any system requires in order to achieve near real time communication is clustering and scalability. Clustering allows application to split over multiple machines supporting distributed operations and scalability. This helps to resource management, load balancing and reducing single machine from getting overloaded with streams. At the same time support for multi-threaded operations reduces latency by avoiding external events. [2, 2-4].

The communication can with reason be referred to as near real time only when the stream processing system can deliver response to high volume of request with very low latency. Some of the most important rules the any system should follow to be real time are listed below:

- Keep the data moving
- Query using SQL on streams
- Handle stream imperfections
- Generate predictable outcomes
- Integrate stored and streamed data
- Generate predictable outcomes
- Guarantee data safety and availability
- Partition and scale application automatically
- Process and response instantaneously

The system should be able to process requests with minimal overhead which can be achieved only when the system provides supports for all the critical functionalities with optimized performance. [2, 2-4].

2.3 Collaborative e-learning

Termed as Knowledge Practices Laboratory, KP-lab is an online collaboration tool that simplifies collaboration with the team members in an innovative manner. That is why KP-lab considers development of collaborative tools to be co-evolution process of researchers, developers and user. Specially focused towards creating sharing and working with knowledge, it is suited best for education purposes or in workplaces where team work is an essential part of project's success. [3]

Document uploading and sharing, instant messaging, creating and editing of shared spaces, adding members to those shared spaces, time management are some important features that KP-lab provides. Shared space is an accessible virtual working space that contains knowledge objects and has information about their contents and

existing relations with each other. Figure 1 is the screen shot of the KP-Lab environment.

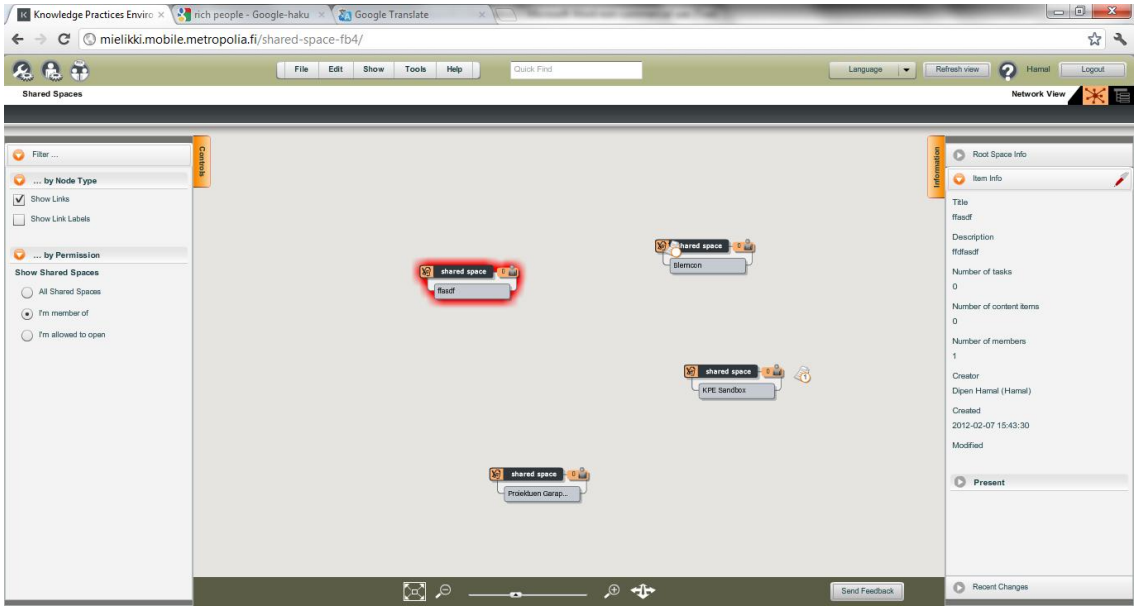


Figure 1. Root Space of the KP-Lab or entry point of authenticated users.

As shown in figure 1, object highlighted in red and other three similar looking objects are shared objects. Members can be added to those objects and those members can collaborate online in real time with the members associated with particular shared object.

Technically speaking, KP-lab is a web based collaborating tool that runs on web server and accessible via web browsers with flash plugin. End user's web interface is developed using action script, scripting language and most of the server side implementation is done using Java programming language. KP-lab relies on flash media server for broadcasting of messages and overall synchronization of the shared spaces and the objects it contains. SyncServices API uses flash media server for overall data synchronization.

2.4 Flash Media Server and the Communication Protocol used

Adobe Flash Media Server is a media streaming platform and a scripting engine. It is used for rich Internet applications that deliver services like video on demand, live web event broadcast, IM. [4, 1] It is a server with open socket that allows persistent connection with the client. Over that connection, client can send/receive presence infor-

mation, audio, video and data streams. Upon connection, it allows clients to make RPC (Remote procedure calls) on server side that calls methods on specific clients. Also, it allows using shared objects and subscribing to them which can be used for synchronizing complex data structure and calling remote methods on multiple clients, all at one time. Flash media server uses rtmp as communication protocol and uses HTTP in absence of rtmp. [5]

Real time messaging protocol is a TCP based protocol used for transmission of audio, video and data between adobe flash player and flash media server. RTMP can be configured into five (5) different ways depending upon the need. First type of configuration is simply RTMP and it does not use any encryption and uses port number 1935 for connection as default port. Referred to as RTMPT, tunneling over HTTP is second type of configuration and uses port number 80 as default. Third one, RTMPS considers about the security and uses port number 443 for connection. RTMPE is more secure configuration and more enhanced, encrypted and faster than RTMPS. It scans the port in the order of 1935, 443, 80 if port number is not specified. Using port number 80 as default, RTMPTE is the most enhanced configuration that encrypts communication channel and tunnels over HTTP. [6, 8-9]. Figure 2 shows the architecture of flash media server connection.

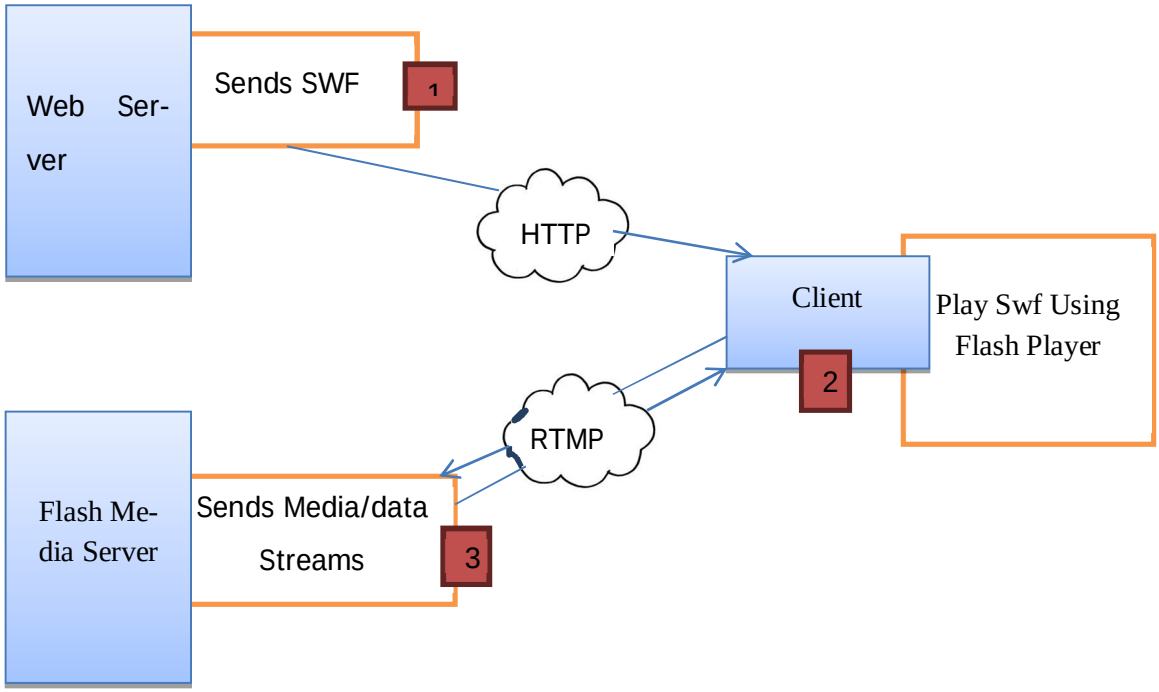


Figure 2. System Architecture of flash media server connection. [6, 9].

As shown in figure 2, client gets swf from the web server and using the RTMP protocol, client is able to send or receive media streams. Such swf files can only be played if the client has flash player installed.

In order to deliver the media (audio and video) streams smoothly, the protocol splits larger chunks into smaller fragments. During the RTMP session, the protocol defines several channels to be used independently for exchanging packets. The TCP based RTMP allows real time communication maintaining single persistent connection with the client. [7]

3 Web Communication Protocols

Web communication protocols are the technology used to exchange information over the Internet. With the advent of Internet, there has been a rise in number of information exchange protocols. Depending upon the nature and capabilities provided by the protocol, clients can have access to the information and interaction with the resources stored in servers. Since protocols are dedicated to perform some specific tasks, use of them has been widely varied depending upon correlation between the requirements and the capabilities they have to offer. Sometimes they can be used in combination with others to fulfill the nature of the services to be provided. Due to the incapability of existing protocols to perform all the required task or one particular task, new protocols are being developed every now and then.

3.1 Transmission Control Protocol/ Internet Protocol

Transmission Control Protocol (TCP) and Internet Protocol (IP) are two different protocols that are complementary to each other and so are referred to as one. TCP/IP defines the instructions and rule that computers use to exchange information meaning that computers at sending end and at the receiving end, both should have copy of TCP/IP program. Figure 3 illustrates the TCP/IP connection between client and server over the Internet.

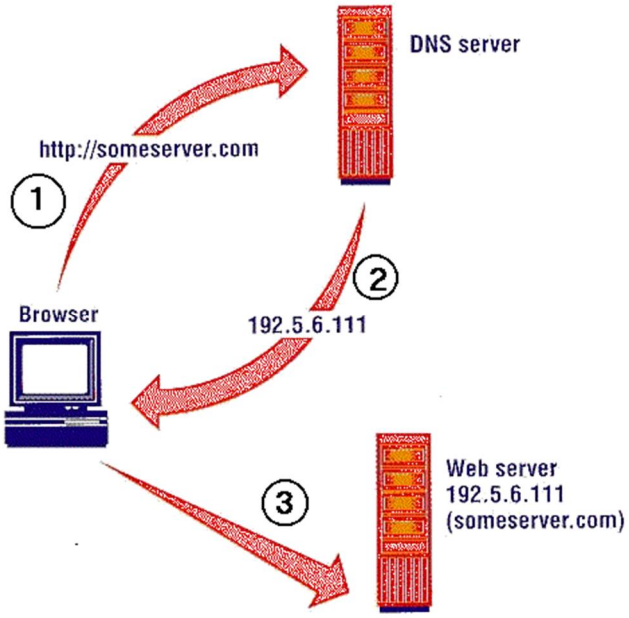


Figure 3. TCP/IP connection using IP address

As shown in figure 3, client request the Domain Name Service (DNS) server asking the ip address of someserver.com. DNS server will translate the domain name into its designated ip address and sends it back to the client. Then the client sends request to the web server with that particular ip address.

IP, the lower layer or also network layer in Open Systems Interconnection (OSI) model is the protocol used to route information to the proper address. It does not guarantee security and reliable data transmission. However it is the foundation of internet as every packet exchanged are routed using it. [8, 28] Each computer connected to the Internet/network is assigned a unique ip address. It makes sure that the information is sent to the intended address. Every packet sent contains an ip address and control information enabling it to be controlled and guided to the required destination [9, 15-16]. It is a connectionless protocol and therefore does not keep track of the routes taken. It depends on TCP for reliable delivery of information. It supports routing, data fragmentation, identification of the protocol, multicasting and broadcasting and prevents loop in the network. [8, 28-31]

TCP, the higher layer, guarantees reliable data transmission between two end points. It breaks down the information into several smaller packets and assigns different route to each packet for faster transmission. Upon arrival at the destination, all the packets

are reassembled. If any data are out of order or corrupted, it reorders the packet thereby reducing the loss of information during transmission. It requires IP address and port number of both client and server in order to make a connection. [9, 42]

3.2 Mail Protocol

Electronic mails or Emails are sent over the TCP connection. Sending and receiving of emails requires combination of two protocols: Simple Mail Transfer Protocol (SMTP) and Post Office Protocol (POP) or Interactive Mail Access Protocol (IMAP). SMTP is responsible for sending emails to the destination server and used port number 25 as default port. Due to its inability to queue messages, it requires either of most commonly used POP3 or IMAP at the receivers end. All of these protocols use TCP connection for transmission and delivery of emails. POP3 or IMAP allows for saving, downloading, creating and deleting of individual messages from mail boxes. [9, 51-52]. Figure 4 shows how mail server sends and receives email.

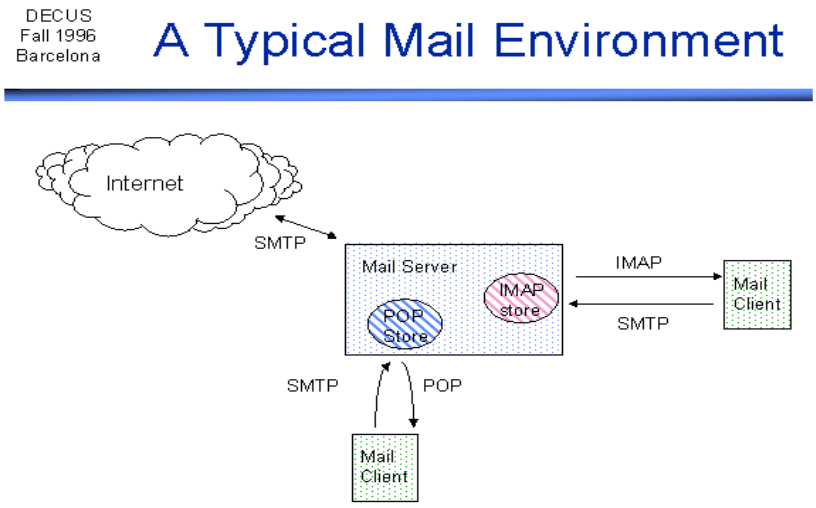


Figure 4. Email sending and receiving.

As illustrated in figure 4, mail server sends or receives email using SMTP protocol. Upon receiving emails, it stores them using either POP3 or IMAP. Authorized client can then download, save, or delete email from mail server.

3.3 File Transfer Protocol

File transfer protocol (FTP) is a protocol used to copy file/s from one host to another host across TCP/IP connection. Unlike client-server application, it creates two connections between the hosts. It uses connection over port number 20 to transfer data and uses connection over port number 21 to send control information. Only authorized users are allowed to request transfer of files. Its limited use for only transferring files has kept it safe from being exploited by hackers. [9, 50] More practical use of it can be seen in a websites that allows uploading and downloading of documents. Figure 5 shows two modes of FTP connection.

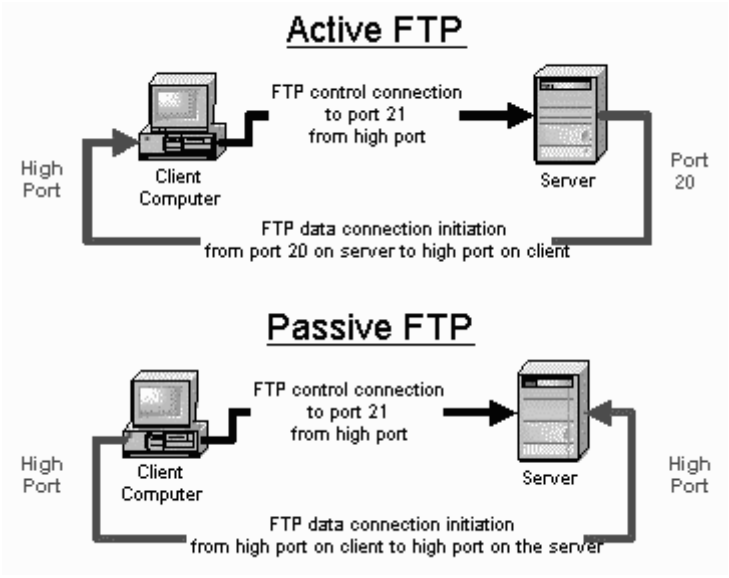


Figure 5. Establishing FTP connection in different modes.

As shown in figure 5, FTP connection can be done in two ways. In the case of Active mode, both client and server open the port however in case of passive mode, only server opens the port for connection to listen for incoming traffic.

3.4 Hyper Text Transfer Protocol

Web pages designed using Hypertext Markup Language (HTML) and accessible via web browsers are transferred using Hypertext Transfer Protocol (HTTP). HTTP is used to communicate between web browser and web servers over the TCP/IP connection in order to transfer data. It is request response protocol i.e. client sends request to the server establishing TCP/IP connection and server sends response to the request and

the connection is closed when the response is complete. It uses port number 80 as default port for establishing connection. [9, 48] Figure 6 shows the request response model of HTTP connection.

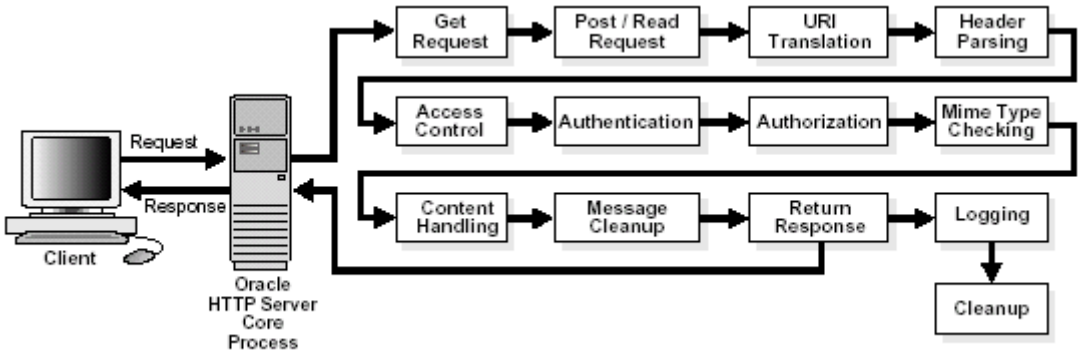


Figure 6. Request-Response model of HTTP

As shown in figure 6, client sends request to the server. Before responding to the request from the client, server checks all the necessary information and finally responds with the data requested if requested data exists.

A more secure and stateful protocol is Hypertext Transfer Protocol over secure socket layer (HTTPS). It is HTTP with SSL support to secure the connection between two communicating computer and encrypt information during the transmission. It uses port number 443 as default port. Any web page transferred using this protocol uses https: at the front of its Uniform Resource Locator (URL).

3.5 What do they have in common?

Except TCP/IP, all other protocols (FTP, email protocols, HTTP) connectionless, stateless and are based over TCP/IP connection. Stateless means all the transactions or interactions that happen between client/Server are independent of others. It also means that the computer keeps no record or track of state of interactions and each transaction has to be handled based entirely on the information it carries. Connectionless mean the connection is terminated every time the transaction is complete.

TCP/IP might be considered not to be stateless and connection less. However, each packets travel independently without any reference to other packets. Even though TCP connection is maintained as long as all the packets are received, it is immediately disconnected after all the packets are received. Also TCP does not reassemble the packets based on the state that is maintained however it does so based on the information carried by packets. [9, 16, 42]

Stateless design requires extra additional information to be added every time there is a transaction and server needing to interpret that information every time resulting in increase of overhead. Computer expects new and complete data every time new request is made. Connection less design requires connection to be established every time new request is made there by adding to the overhead.

4 Extensible Messaging and Presence Protocol (XMPP)

4.1 Description of the protocol

Previously referred to as Jabber, Extensible Messaging Presence Protocol (XMPP) is an open-source IM protocol designed to support real-time messaging, presence management and request-response services. Like any other existing communication protocols, it defines a format for exchanging data. Even though, it is based on client-server architecture, peer-to- peer communication is also allowed between two clients or two servers as shown in figure 7. [10, 3-5]

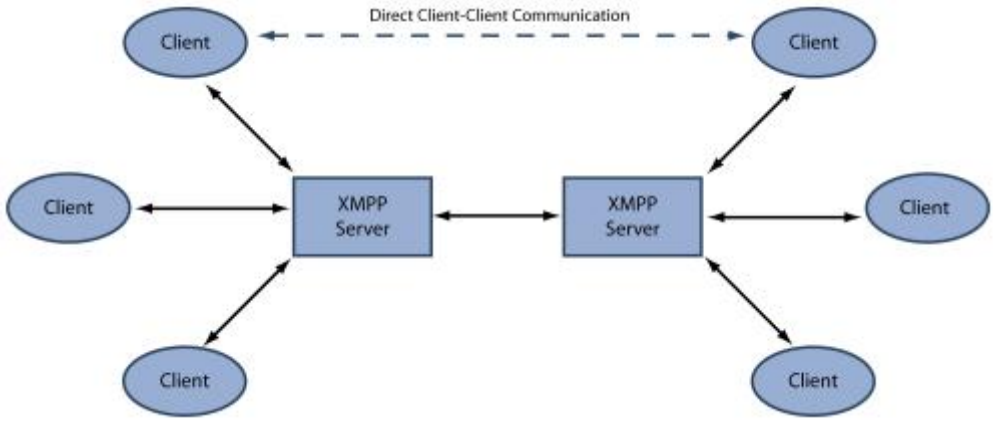


Figure 7. Communication between servers and clients. [11]

Figure 7 illustrates the communication between servers and clients. It shows that clients on different servers can communicate with each other via two communicating servers. Even direct client to client communication can be maintained after the clients authenticate with their servers.

XMPP allows exchanging XML stanzas in near real time. Use of XML as data exchange format gives communication an added benefit of rich and extensible structure. XML makes it more human readable and easily debugged. With an easily extensible ability of XML, it gains an ability to add new features that are both backward and forward compatible. Support for encrypted communication between end to end through use of Transport Layer Security (TLS) and strong authentication mechanism via Simple Authentication and Security Layers (SASL) make it more secure and helps eliminating spams and unauthorized messages. XMPP supports mainly small bits of information exchange which gives XMPP extremely low latency making it extremely useful for real time communication. However, it can function as signaling layer for moving large blocks from point to point. [12, 4]

It can be used in two ways: one for creating services and other for creating application. Some of the services that can be created using it are channel encryption, authentication, presence sharing and detection, buddy list, notifications, one-to-one messaging, multi-party messaging, service discovery, peer-to- peer media sessions. It can be used to make any kind of real time applications. Some of its important implementations are seen in gaming industry, collaborative shared spaces, synchronization, geolocation, groupchatting, designing system controls, data syndication, middleware and cloud computing. [13, 4-6] Though it was primarily designed for instant messaging (IM), its ability is not limited just to IM. Any tasks that benefit from exchange of structured messages can rely heavily on it.

4.2 Why Real Time Collaboration Protocol?

Due to the increase in the use of Internet for online collaboration and communication, real-time communication is increasingly becoming necessity for web applications. Since XMPP provides polling based solutions which in turn reduces the response time and latency, it perfectly favors the real time communication. Just being faster is not the

only feature that real time communication requires. Communication over the Internet has to be reliable and secure. It has built in support for Transport Layer Security (TLS) and Simple Authentication and Security Layer (SASL) making it more secure. In addition, it can be customized according to the needs.

XMPP is an open source, open standard protocol without platform dependency which makes it easier to implement and standardize. Server based on it, especially eJabbered are rapidly developing into more than just instant messaging servers. Such servers are easy to setup, scale and create a cluster out of many servers. It is based on decentralization meaning anyone can run their own XMPP server without the need to have central master server. [11] This favors any companies, organizations, offices, schools or even households to have their own private network which can be separated from public networks behind the firewall. Many successful startups have used this protocol. Chesspark, online multiplayer chess gaming site, is developed based on this.

XMPP provides best solution for the problems that existing web framework cannot solve or do not attempt to solve [14]. Many Internet giants including Google, Facebook and Twitter have used this to provide the best solutions proving it to be the best solution lying around. Google Wave used this protocol as the base for their federation protocol i.e.; server to server communication. Facebook uses XMPP for real time messaging. This has also motivated larger number of developers to indulge into it.

4.3 Comparison of communication Protocols

Even though XMPP is not as mature as HTTP, it can over shadow many of HTTP capabilities with its ability to communicate in real time, extensibility, security and many other features that it has to offer. However there are certain points where it still looks like a child in front of a giant when compared with HTTP. Table 1 shows the comparison between two protocols: decade old XMPP and much matured HTTP.

Table 1. Comparison of two different communication protocols

Comparisons	XMPP	HTTP
Pushing Data	Application receives notifications whenever new information is available without needing to request server for information thereby reducing latency almost to zero. It is bidirectional.	HTTP Client gets new information only when server responds to the client's request. It is unidirectional totally based on request-response.
Pleasing Firewalls	It is firewall friendly. As client initiates the connection, the server can push all the data on the same connection like in response to the HTTP request upon connection establishment.	HTTP callbacks and arbitrary connection are very hard to implement in practice in the presence of firewall.
Improving Security	Since, it is built on top of TLS and SASL technologies; it provides solid foundation of security for XMPP connections.	HTTPS (HTTP in combination with SSL/TLS) also called secure HTTP provides security though it depends upon the correctness of implementation of the web browser, server software and cryptographic algorithms supported.
Bigger and Better Options	It includes three different low level tools: <presence>, <message>, and <iq> and hundreds of extensions	It supports handful of operations: GET, POST, PUT, DELETE and so on.
Statefulness	It is stateful which makes scalability a challenge as tools used to scale HTTP can be used. However it's increasing popularity and with maturity, it might see some tools especially dedicated for its scalability.	It is stateless Protocol making it easier to scale.
Maturity	As it is new compared to HTTP, there are very limited number of dedicated developers, specialized severs and libraries.	There is larger number of developers who understand it easily, more libraries and much matured HTTP server exists.
More Overhead	It does not focus on short lived session and simple request causing more overhead to set up, maintain and destroy XMPP sessions.	HTTP is session less protocol so it drops connection immediately after handling the request thereby resulting in lesser overhead.

[12, 28-30]

Based on the comparisons provided in table 1, it could be said XMPP has the potential to solve many problems that HTTP has not been able to do. Its extensibility and security makes it better than HTTP and also it is real time in nature compared to HTTP's request-response model.

4.4 Real Time Network

Also referred to as real time network, XMPP network needs certain actors to be a complete network. Servers, Clients, Components and plug-ins are those actors that complete the network.

Server

XMPP server is the backbone of the network. Server's role is to route xml stanzas from one user to another whether they are on the same network or are on the remote network. Beside that server provides basic messaging and presence features. [12, 6] XMPP Network is formed when two or more servers communicate with each other. Like Google wave, when set of public XMPP servers are able to communicate with each other, and then it is termed as federated XMPP networks. Such servers are easy to set up and are available for almost all platforms. It allows users to connect to it and communicate messages.

Clients

Client is another important actor of the network. Clients make the majority of entities/actors that are connected to the servers. Though some servers allow anonymous login, clients need to authenticate to the servers if anonymous login is restricted. Clients can send xml stanzas, messages or presence which is tunneled by server to the required destination or targeted users. Usually, clients in the network are human i.e. IM users; however, automated services also exist as bots. [12, 7]

Components

Components are other entities that connect to the server except clients. Components enhance the capabilities to the server by providing additional services. They have their address and identity within the server though they run externally and communicate over component protocol. Each component becomes a separately addressable entity within the server and act as a sub-server. Components can manage roster and server

allows them to internally route or manage stanzas for themselves. Though they need to authenticate to the server, authentication is not as complicated compared to the SASL client's authentication. [12, 7]

Plug-ins

Server behavior or capabilities can be extended simply by using plug-ins. Plug-ins for servers are generally a set of software components that add specific abilities to servers. These are the components written in same programming language as servers are written in. Plug-ins may overlap with components to greater extent; however, they are not portable between different servers and have less overhead compared to external components as they do not require communicating over network socket and serializing XML. [12, 8]

4.5 Connection Life Cycle

Connection

XMPP based communication is all about exchange of XML stanzas. Stanzas can be exchanged only via XMPP stream. XMPP stream cannot exist without the connection to the XMPP server. In order to connect to the server, client should provide the valid credentials. Similar to the E-mail services, each client has unique identity that has "@" symbol as suffix followed by server's domain name. Only after the connection is established, users are able to exchange stanzas. [12, 18]

Stream Set Up

XMPP stream gets started immediately after the connection with server is established. It is opened after sending <stream:stream> element to the server and then the server opens the stream and responds back by sending the similar stream opening tag. When streams are opened from the both end, stanzas are ready to be exchanged. After the stream is set up, server immediately responds by sending a <stream:features> element that outlines all the available features on the stream. [12, 19]

Authentication

XMPP allows secure communication. Authentication is based on SASL protocol and supports various authentication mechanisms depending on the server used. Plain-text and MD5 digest based authentication are generally used authentication mechanism;

however, Kerberos or token based authentication can also be done though they are a bit more complicated to achieve. After authentication, client binds a resource for the connection and starts a session. Server to server authentication is slightly different as they need to exchange TLS certificates and verify each other. Also sometimes, receiver verifies the sender's identity via DNS by using dialback protocol. [12, 20]

Disconnection

Final stage of the connection life cycle is disconnecting. Disconnecting involves two steps: terminating the session and disconnecting. Sending unavailable presence to the server before sending the closing </stream:stream> tag is the optimal way to terminate session. By doing so, any arriving stanza is received safely. After receiving the closing stream tag, server terminates its stream related to that particular client. [12, 20]

4.6 Extensions (XEPs) of the Protocol

Since XMPP is extensible and an open standard, developers can define new extensions that focused towards solving existing problems or adding new feature to the protocol. With consent to transfer ownership to XSF, developers can submit specifications to XMPP Extensions Editor which upon approval from XMPP council is published as experimental extensions. To have the status of Final, thus created new extension has to go through several refining and reviewing. [15]

XEP States

- Experimental – Upon acceptance of new extensions proposal by XMPP council, such protocols are deemed experimental and published as experimental XEP. It is the initial stage of protocol specification before getting the status of Draft with possibilities of many modifications.
- Proposed – It is the state at which the experimental extensions are considered suitable to advance towards state of Draft or Active.
- Draft – After all discussions, technical reviews and upon approval from the council, the new extension is granted the state of Draft with issuing advices not to be used in very critical application. It is hoped to be implemented in production environment.

- Final – Any extension is considered in Final state after it has been in Draft state for at least six (6) months and has very less chance of modifications. With implementation in at least two codebases, it is granted the status of Final. After the extension is in Final state it is considered safe to be deployed.
- Active – Any extension that is approved to advance from being experimental automatically get the status of Active.
- Deferred – Deferred status is applicable if the extension is not updated in twelve (12) months' time.
- Retracted – Upon receiving the request from the author to remove extension from further considerations, XMPP Extensions Editor grants status of Retracted to the extension.
- Rejected – If XMPP councils deems the extension to be unacceptable and rejects from further processing, then it gets the status of Rejected.
- Deprecated – Any extension is deprecated if it is outdated by modern protocol and no more implementations are encouraged.
- Obsolete – All those deprecated extensions are made obsolete when the council determines not to use them in any kind of application development. [15]

Appendix 1 shows the lists of active and final XEPs.

5 Real Time Collaboration Server

Any server that has a built in support for the XMPP protocol is XMPP server. Previously referred to as Wildfire, Openfire is XMPP based real time collaboration server written in Java programming language. It is an open-source, independent of the operating system that provides a very secure platform. It provides full support to Spark, XMPP based real-time collaboration client and compatible to most of the XMPP clients.

5.1 Selection of Real Time Server

Out of many existing XMPP servers, there was a need to select the one that matches our requirement and fits into our criteria. Being open-source was the first and a very important requirement. Second was to be able to support real-time collaboration. Easy to set-up and configuring was another additional requirement. Since Openfire is open-source real time collaboration and a cross-platform server, these two reasons were

good enough to select it for our research purpose. Besides it happened to be written in Java programming language with clear and good documentation which made it easier to understand and implement. Among all the existing XMPP servers, it is easier to set-up and logs all the information that makes administering easy. Also it allows configuring to set up any secured connections. With full support to the protocol, it is easily extendable. It comes with several features that support streaming communication. [16, 11-12]

Having met those important requirements, Openfire was deemed the best alternative. Benefits provided by it were not limited to only meeting those requirements. It has its own embedded database that can be used for small scale deployment. For large scale deployment and if required, it could be connected to external MySQL, Oracle, Microsoft SQL Server, PostgreSQL or IBM DB2 database by simply providing link to those databases. Another very appealing feature is it's user interface. Graphical User Interface (GUI) is very easy to use and straightforward not needing to have high level of technical understanding. It also allows remote access via web browsers.

Openfire is secure as SSL encryption can be enabled in communication includes sensitive information. It also provides with additional feature that allows preventing anonymous users from accessing into the system and using the service. In case the default authentication is not suitable when security is not a major concern, it can be overwritten suiting to the needs and required methods of authentication. It not only supports for monitoring users, servers and server connections but also logs all the anomalies which could be referred later on to monitor.

Openfire allows installing plug-in to add new services that are not its native feature. Many new plug-ins are being developed every day and there are exists many useful plug-ins such as content filtering, broadcast, monitoring service and packet filtering. It also supports clustering which can help solve scalability issues.

5.2 Support Provided by Real Time Collaboration Server

Openfire is developed to support the XMPP protocol defined by Request for Comments (RFC) 3920 and RFC 3921. In addition, it also provides support for numerous extensions to the protocol that are defined through the XEP process at xmpp.org. Table 2 and 3 lists the support provided by Openfire to the extension.

Table 2. Basic IM Protocol Suite Support. [17]

Specification	Supported
RFC 3920: XMPP Core	Yes
RFC 3921: XMPP IM	Yes
XEP-0030: Service Discovery	Yes
XEP-0077: In-Band Registration	Yes
XEP-0078: Non-SASL Authentication	Yes
XEP-0086: Error Condition Mappings	Yes

Table 2 lists all extensions that Openfire provides to the basic IM functionalities. As shown in the table, service discovery, in-band registration and non-sasl authentication are some to the basic IM functionalities that it provides support for.

Table 3. Intermediate IM Protocol Suite Support. [17]

Specification	Supported
XEP-0073: Basic IM Protocol Suite	Yes
XEP-0004: Data Forms	Yes
XEP-0020: Feature Negotiation	No
XEP-0045: Multi-User Chat	Yes
XEP-0047: In-Band Bytestreams	Yes
XEP-0065: SOCKS5 Bytestreams	Yes
XEP-0071: XHTML-IM	Yes [1]
XEP-0096: File Transfer	Yes
XEP-0115: Entity Capabilities	Yes

Table 3 lists all extensions that Openfire provides support to the intermediate IM functionalities. As shown in the table, file transfer, entity capabilities, data forms, feature negotiation are some of the intermediate IM functionalities and it provides full support to these extensions. Appendix 2 lists all the extensions that it provides support for.

5.3 Plugin Support

Openfire has web-based admin console plugin installed as default. This makes running and operating it very easy. Admin console could be accessed via browser by simply providing the linking url: http://localhost:9090 or http://127.0.0.1:9090 in case the it is installed locally in the same computer or using the ip address or domain name of the server in case it is required to access admin console on the remote computer/server. Domain name of the remoter server could also be replaced by the IP address of the computer.

Several other plugins are supported for achieving specific tasks. One of the most important plugin support is broadcast plugin. This plugin would enable broadcasting messages to the large number of user. By default, this plugin would send messages to all the users of the system. However, it could be also used to send the message to the specific group of people. Another important and the most used plugin is user service plugin. Manually creating users was very time consuming and hectic. This plugin would allow adding, deleting and modifying users by simply sending the HTTP request to the server. For our test, it was very useful and proved very fruitful as it was possible to create and register multiple users at one time by executing Java Application to send multiple requests.

Another plugin is search plugin which helps searching of the users registered in the same server. Search could be made by user's name, username and the email. Monitoring service plugin can also be installed and it allows monitoring the server statistics, chat archiving and viewing chat logs. Many more plugins like Clustering, Email Listener, Content Filter, Presence Service, Registration and Subscription are also supported.

Installation of plugins to Openfire is very easy. As all the available open source plugins could be viewed under the plugins page of the admin console and by simply pressing plus (+) at the side of the desired plugin, it could be installed. Plugins not listed as available and that are commercial can also be installed by simply uploading the JAR file to the plugins page of the admin console. They can also be deployed by copying the JAR file to the plugin directory inside of Openfire's installation directory.

6 Application Design and Implementations

Since the client side user interface for Knowledge Practices Environment (KPE) lab is flash application and was designed and implemented using actionscript, this project used actionscript as a scripting language for design and implementation. Adobe Flash Builder (FB) has been the best choice of KPE developers for designing, improving and maintaining the client side that is the reason why this project considered FB as the most suitable among the available Integrated Development Environments (IDEs). The project used the latest version of FB at that time. In order to design the client that supports XMPP in full and to meet the requirements, the project also used two different Application Programming Interfaces (APIs): XIFF and Smack. For testing the performance, the project used The Grinder as a load testing framework. The project used Openfire as main collaboration server and XAMPP as web server for the application and database to store the data.

6.1 Application Description

The application has been created as a chat client for the Openfire server. Even though the main idea behind this project is to find out whether the server is capable enough to serve KP-lab, the whole project has been implemented outside of it. However, those implementations are applicable to the lab as well. Also being a web based application, this implementation genuinely suits to the operating environment of the KP-lab. Basically, this application can be used for authentication, IM, presence information sharing, profile creating, conference meeting also called multi user chat and broadcasting of messages. The application also has drawing capability as an additional feature.

To really determine the performance capability of the real time server in relation to the eLearning lab, the project also included a simple drawing board where the users could draw something over the direction of mouse dragging. When the users are drawing lines or anything visible, it would also broadcast the coordinates of the mouse pointer continuously, if the position of mouse pointer is changed. As a result of this, the other users registered with the server and who are currently logged in would also get the same drawing on their drawing board. In other words, the system would draw the same drawing onto their canvas based on the broadcasted canvas.

6.2 Development Tools & APIs used

XAMPP is an open source cross platform development server. Developers can use it to test web application without the need to connect to the Internet and deploying to the production server. It is easy to install and requires no configuration. Since default configuration is not good enough from security point of view, it is better to limit its scope to development purpose only but not use it for production purpose. It is the compilation of free softwares: Apache HTTP (web) server, MySQL database server to store data, PHP for server side scripting and Perl.

Adobe FB is an Eclipse based development tool for building mobile, desktop and rich Internet applications. Applications are built using actionscript. However, it is also possible to carry out development work based on other programming languages such as PHP and Java. It has built in support for android based devices, IOS and Blackberry tablets.

The Grinder is a framework that allows running distributed test suitable for measuring the performance of anything that has Java API. Load testing being the main criteria, this is useful for testing HTTP web servers, Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) web services, application servers and some custom protocols. All the tests are written in Jython, a powerful scripting language. Jython is an implementation of the high-level, dynamic, object-oriented language Python seamlessly integrated with the Java platform. This project used three point five (3.5) version of The Grinder. [18]

XIFF is open source flash library that supports XMPP protocol for creating instant messaging and presence client. Extension architecture makes it possible to extending the protocol according to the needs. The library also includes the extension that supports XML-RPC over XMPP, Multi-user conferencing (XEP - 0045), Service browsing (XEP-0030) and Extensible HTML (XHTML) message support (XEP-0071). Since the client side of this project was a flash application, this API was the most important one and specifically used.

Similarly, Smack is an open source client library used for the same purpose as XIFF. However, it could be used if the client is based on Java but not flash. Even though the client side of this project was totally based on flash, this library was also used to override the services provided by the Openfire. For example, it was used to create customized authentication that could replace the authentication method provided by the server in order to make it suitable for this project's purpose. Default authentication was easily overridden by creating a plugin for the server. Plugin used was a Java Archive (JAR) application that was created using Java programming language. It was placed in the plugin directory of the server in order to override authentication method.

SchemaSpy was also used as a support for this project. It is basically a java based tool for creating visual representation out of metadata of a schema in a database. The output created by it can be viewed in a browser.

6.3 Overall Design

Application client is based on open source flex application framework. It is entirely tag-based, event driven and fully implemented in Macromedia eXtensible Markup Language (MXML). All the visual components are designed using MXML. It is XML based markup language for designing user interface and it is combined with actionscript in order to achieve the project's objectives. It is event driven in a sense that any interaction with the application can provoke events which can be utilized to execute some actions that might result in a different view, some processing or results depending upon the need of the application. The application utilizes the user inputs from mainly two input devices: keyboard and mouse to generate events. For example, when a user of the application clicks a visual component of the user interface, the component, if programmed to

generate event, will generate event which could execute some actions. Application framework allows to fully separate view code from the other codes used for implementing application logic.

Application client is web based and can be accessible only via web browsers that have flash player plugin installed and in any platform. The client is exported as swf file or simply flash file and to be able to access from web browser over the Internet, it was deployed to the web server.

Table 4. MXML code used for designing login user interface.

```
<s:states>
<s:State name="loginState"/>
<s:State name="ConnectedState"/>
</s:states>
<s:Form includeIn="loginState" x="158" y="134" width="265"
height="153">
     <s:FormItem label="UserName">
             <s:TextInput id="userName" text="user0"/>
     </s:FormItem>
     <s:FormItem label="Password">
             <s:TextInput id="password" text="abcd" />
     </s:FormItem>
     <s:FormItem label="Server">
           <s:TextInput id="server" change="onServerInputChange()"
                   text="{ChatManager.serverName}"/>
       </s:FormItem>
       <s:FormItem>
              <s:Button id="btnConnect" label="Connect"
                    click="connectEvent(event)"/>
      </s:FormItem>
</s:Form>
```

The piece of code, entirely based on MXML as shown in table 4 is used to create the user interface for login form. When deployed, it will generate two input field for the user to provide login credentials along with a button which will direct to another view, ConnectedState when clicked and if the information provided are correct.

```
public function connectEvent(event:Event):void
{
if (!userName.text.match(pattern))
     Alert.show("Missing UserName");

else if (!password.text.match(pattern))
      Alert.show("Missing password");
else if (!server.text.match(pattern))
      Alert.show("Missing Server");
else
       connect();
}
```

Table 5. EventHandler for connect button used for authenticating user.

Figure 8 displays the output generated by this piece of code as shown in table 5 when run in web browsers.

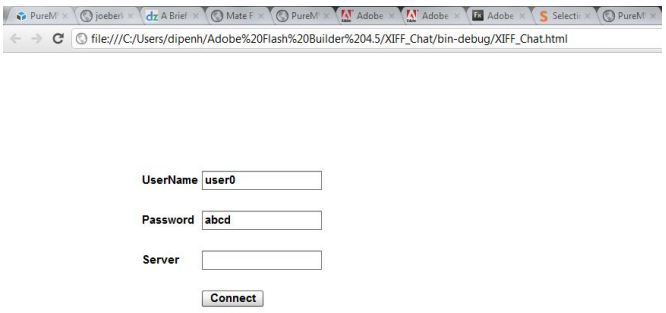


Figure 8. User interface for user login.

When the connect button as shown in figure 8 is clicked it will generate and dispatch event which is handled or listened by existing event listener/s. As shown in figure 9, connect() method from table 5 will also be executed which is responsible for the whole login process. Depending upon the response from the connect() method, the view of the application will be updated.

The whole cycle from user interaction with the application to event generation, event handling and result processing (view updating in this case) has been clearly shown in the figure 9. The figure again utilizes the login process.

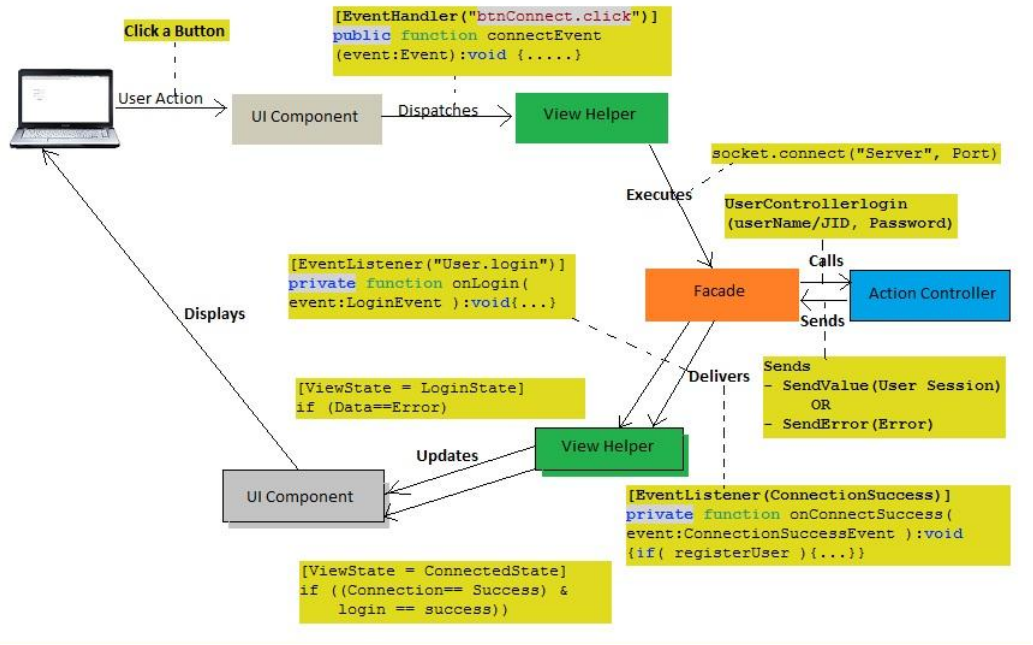


Figure 9. Eventhandling for authenticating users.

When successfully connected to the server, it would authenticate the user with the provided credentials. If the credentials are valid, the view would be updated to the connected state. As shown in the figure 9, the action controller or Openfire, real time server in this case would respond according to the login credentials provided. Depending upon the response from the server, the view of the application will be updated. The application utilizes the same phenomena for other processes as well. However being the real time, IM application and based on XMPP protocol, it is also capable of getting response from server without any user interaction. Such response would also generate result or update views. For example, if the client associated with the server would send some messages to the client that has logged in, the server would direct those messages to that particular client and in response to that event, the application would create a pop up window that includes information about the user who is sending the message and the message itself.

6.4 Application Architecture

Whether the Openfire was running with the embedded database or the external MySQL database, the application always used only one database. In case of the embedded database System, the server used the built in database with its own schema for storing information related to users, services and property of the server itself. As shown in

figure 10, the server is shipped with built in HSQLDB which it uses as back end data storage. HSQLDB is SQL relational database engine written in Java and very useful for persistent data storage due to its flexibility, smaller size and faster processing capability.

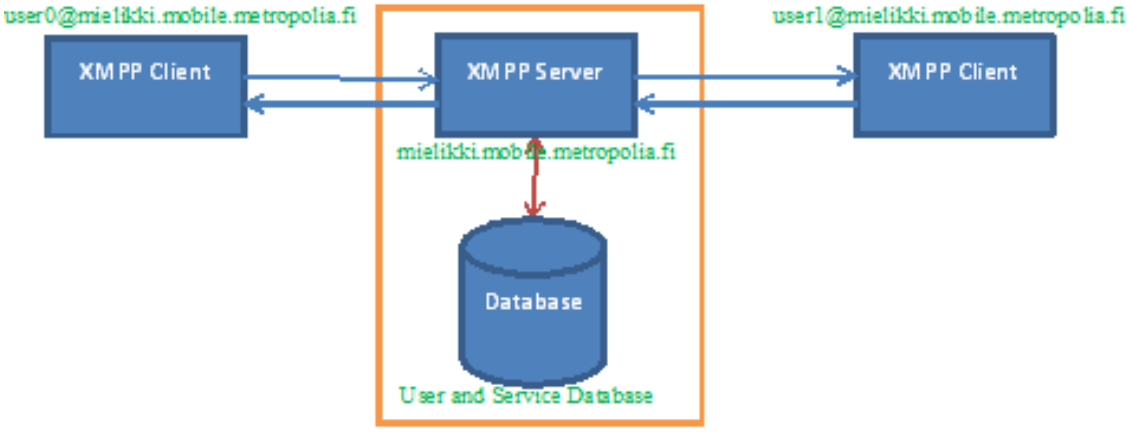


Figure 10. System Architecture with embedded database.

In case of the external MySQL database, database used the schema provided by the Openfire. As shown below in figure 11, all the information is stored in external MySQL database. This application used Openfire, MySQL database as external database running on XAMPP accessible over localhost or IP address, 127.0.0.1.

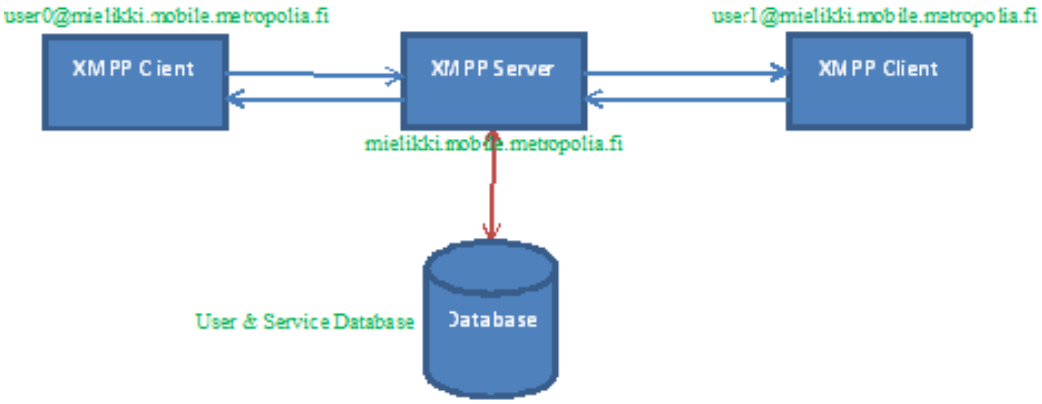


Figure 11. System Architecture when external database is connected.

The server would fetch all the necessary information related to user, authentication, presence, properties and others from the external database, Openfire.

The schema provided by the server had all the necessary tables and with all the necessary relationships. Therefore, it was not necessary to create extra tables. However, it

does allow creating our own tables with custom fields that matches our needs and allows configuring that would allow communicating with the server. For example, the project used the external database with default schema and also created a table named user_provider that would only store necessary information, username and password only, related to users. The server was configured so that it would authenticate users based on the information fetched from the newly created database table, user_provider instead of the default table, ofuser. For that purpose custom user provider, a userProvider plugin was created for the server. Also the following piece of code as shown in the table 6 could be used to configure the server so that it would use the table named user_provider instead of table, ofuser for authentication and storing of user information.

Table 6. Configuration of Openfire

```
<jive>
  ...
  <provider>
    <auth>
      <className>org.jivesoftware.openfire.auth.JDBCAuthProvider</className>
    </auth>
    <user>
      <className>org.jivesoftware.openfire.user.JDBCUserProvider</className>
    </user>
  </provider>
  <jdbcAuthProvider>
     <passwordSQL>SELECT password FROM user_provider WHERE
username=?</passwordSQL>
     <passwordType>plain</passwordType>
  </jdbcAuthProvider>
  <jdbcUserProvider>
     <loadUserSQL>SELECT name FROM user_provider WHERE
username=?</loadUserSQL>
     <userCountSQL>SELECT COUNT(*) FROM user_provider</userCountSQL>
     <allUsersSQL>SELECT username FROM user_provider</allUsersSQL>
     <searchSQL>SELECT username FROM user_provider WHERE</searchSQL>
     <usernameField>username</usernameField>
     ...
  </jdbcUserProvider>
   ...
 </jive>
```

Table 6 displays the some contents from the configuration file of Openfire used for system configuration. In order to configure the the server, changes have to be made in configuration file, Openfire.xml that exists in the installation directory of the server and the service has to be restarted for making changes. However, the use of custom user provider and authentication was only for the purpose of testing the server's con-

figuration and flexibility. The whole project was designed to depend on the default schema provided by the server.

The schema consists of thirty four (34) tables and it uses ofuser table for storing all the necessary information related to user creation. The table includes information such as username as primary key, name, email, creation date, modification date, plain and encrypted password. User authentication is also done based on the user information available in ofuser table. Upon registration, user could change information about their presence, create profile, add/ remove users to/from their friend lists, create groups, and rooms. The information related to those actions is stored respectively into table ofPresence, ofvcard, ofroster, ofrostergroup. Figure 12 illustrates the schema of the database design and the relation between the existing tables.

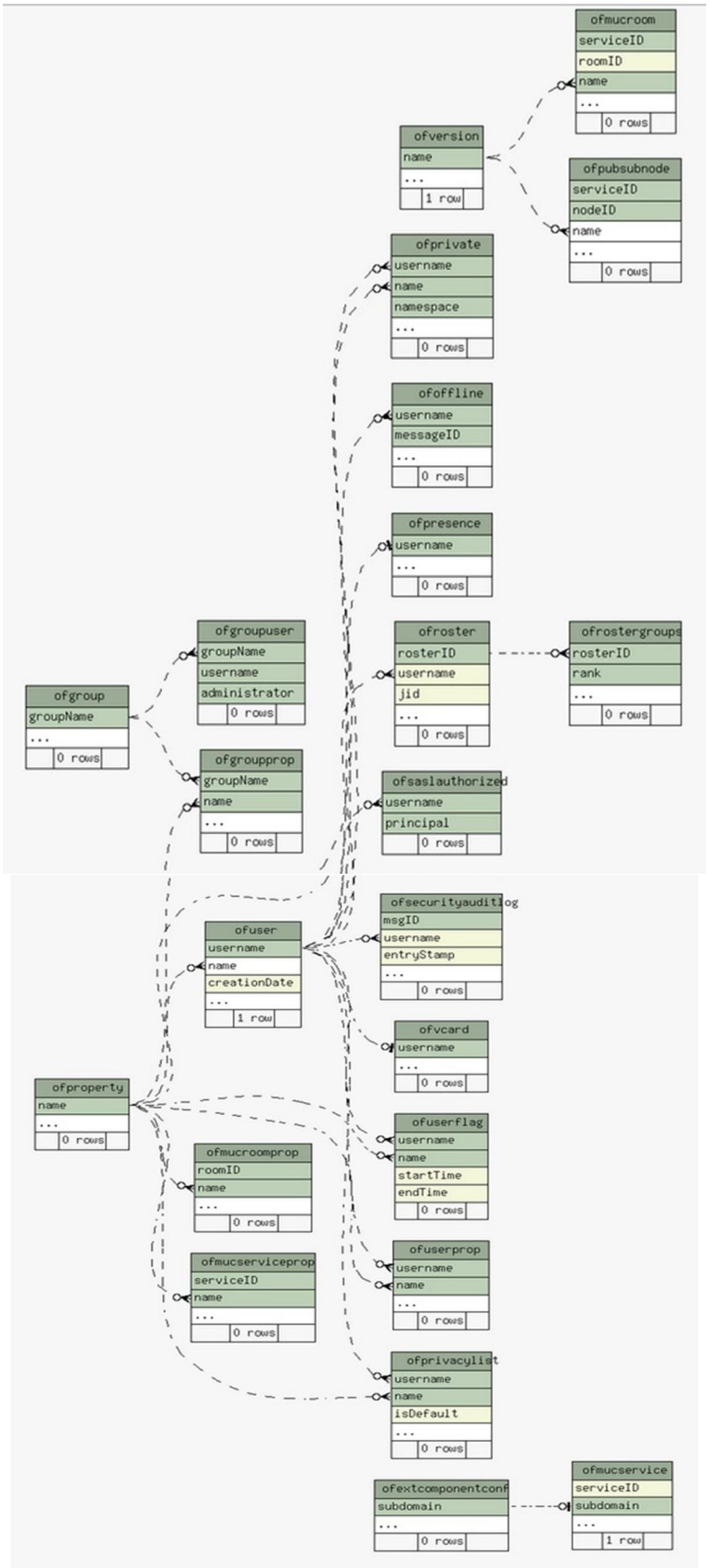


Figure 12. Schema of the database design used by Openfire.

The schema also provides with a table named ofpubsub for storing information about publishing and subscription. However, this application has not utilized this table as it does not require notification from any services nor does it publish anything that could be subscribed.

6.5 Application User Interface

As explained earlier, the application's user interface can only be accessed via web browsers. Upon opening the application, it will load the login page. The login page cannot be bypassed and the users can be able to use the application only when they provide the valid credentials. When the authentication is complete, the users can access into the application. The application has a simple outlook and it is easy to use the interface in a sense that anybody with basic computing knowledge can use the application in the way it was meant to be used.

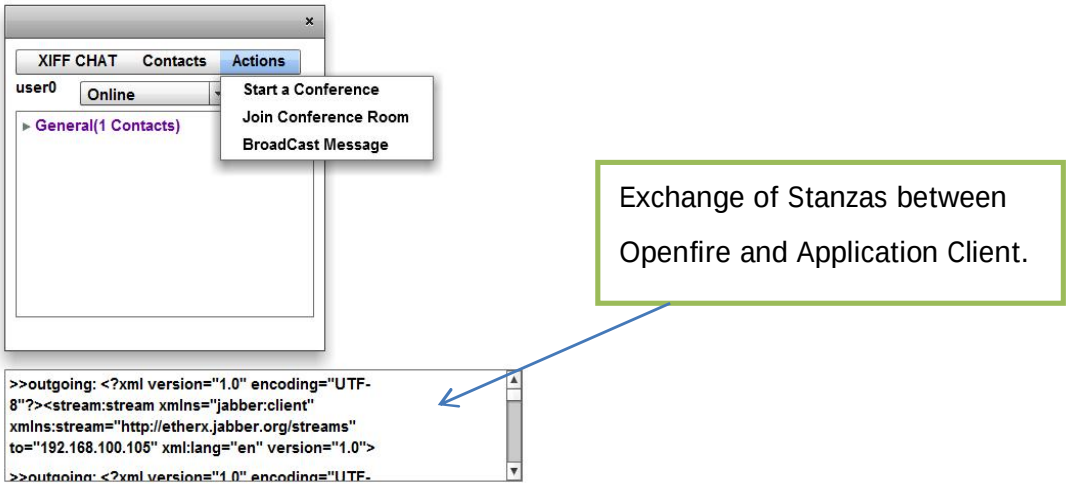


Figure 13. View presented after successful user authentication.

Figure 13 shows the view of the application after the user has authenticated. The user interface lists the people who are under different group headings as categorized by the user. The user interface consists of drop down menu bar which allows the user to perform certain actions. Each menu items when clicked will perform an action which is responsible for generating different views as popups on top of the main application. Under the XIFF CHAT menu items, the application has a menu button which when clicked will generate a popup as shown in the figure 14 which allows users to update their profile information. It also allows uploading the profile picture of the user.

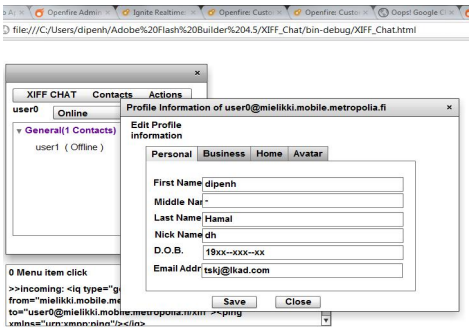


Figure 14. Profile Updates

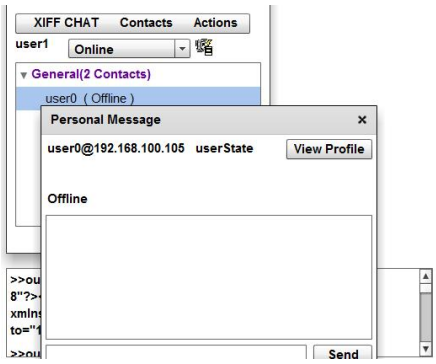


Figure 15. Pop up window for chatting

Similarly, figure 15 is the resultant view of the application when the user double clicks on one of the users from his friends' list. On that new popup window, the user can start a conversation with the particular user whose user id is displayed on top of that window. The application will result in the same view when one of the users from the friend's list performs the same action.

The application has two (2) menu buttons: Add Contact and Remove Contact under the contact menu item. Add contact allows adding a person to the friend's list and remove contact allows removing people from the friend's list. For both purposes, authenticated user has to provide the user id of the particular person. The user id consists of the username suffixed with the domain name (mielikki.mobile.metropolia.fi in our case) of the server following '@' symbol. If some other users will also add this user, s/he will get an invitation in the form of popup and when accepted, the application will start exchanging the presence information between the users. Also, if this user is removed by another user, the application will stop sharing the presence information and the users will not be listed under each other's list of friends. Application does have a drop down menu for changing the presence which on changing will send the current presence to other users in the friend's list via the server.

The application also allows starting a conference which is a multi-user chat room. Authenticated user can start a conference room with unique id and can send invitation or get invitation to join the room. Members for the room can be either manually added or selected from the friend list. The conference looks as shown in figure 16. Each user who has joined the room can view what others have written or sent.

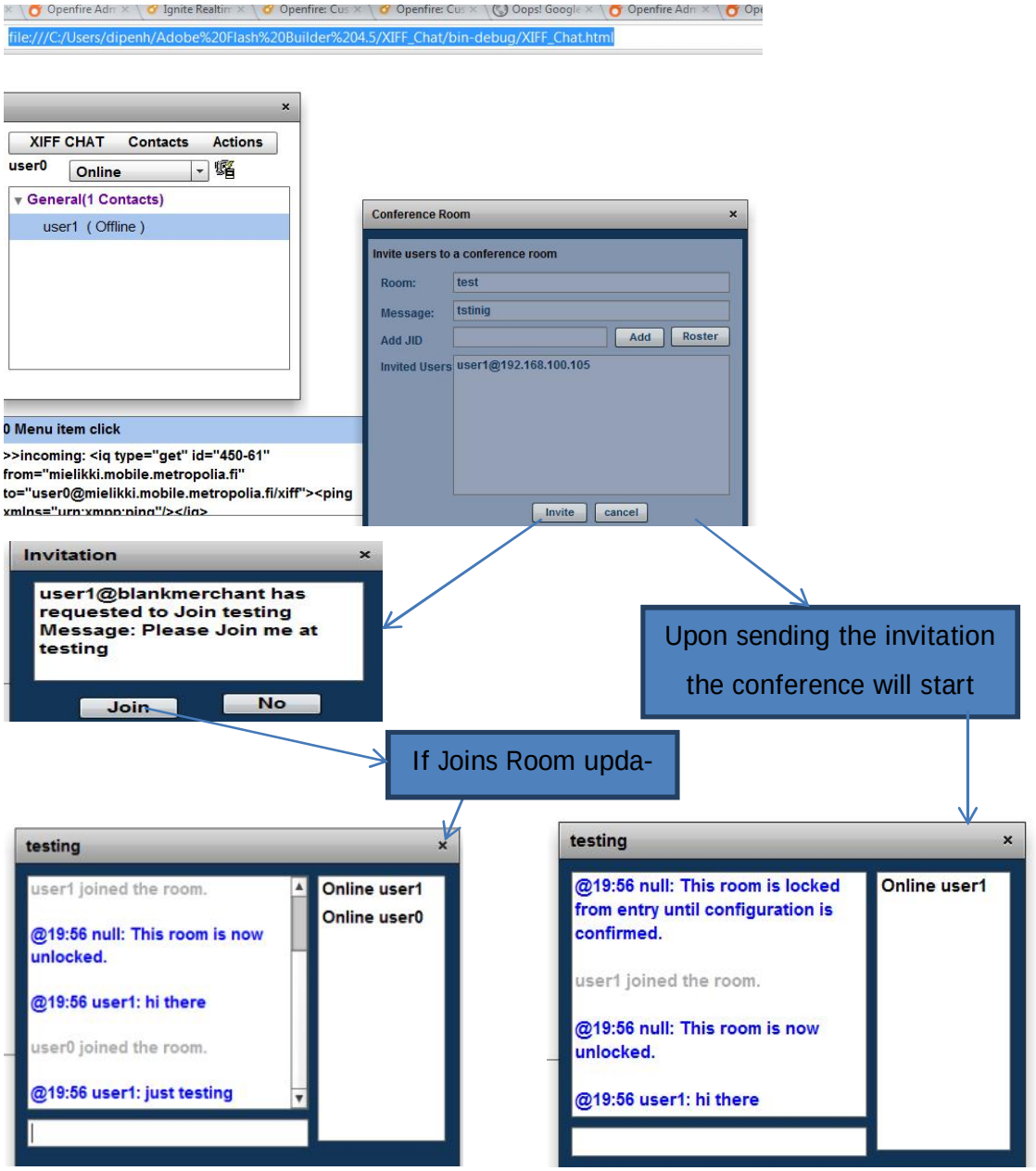


Figure 16. Process of starting a Conference.

Similarly broadcasting is one of the important features of the application. Broadcasting of messages can be of two types. User can broadcast the message as normal or alert (notification) type. Normal type of broadcast is received both by online and offline users and when the normal broadcast is received, a new popup window similar to chat window appears with messages and information about the sender. However alert type is received by only online users and when it is received, the application gets the notifications and displays the number of new broadcast received. Alert type message in-

cludes the message and sender's username and can be read by simply hovering the mouse icon over the small computer icon aligned with the menu bar.

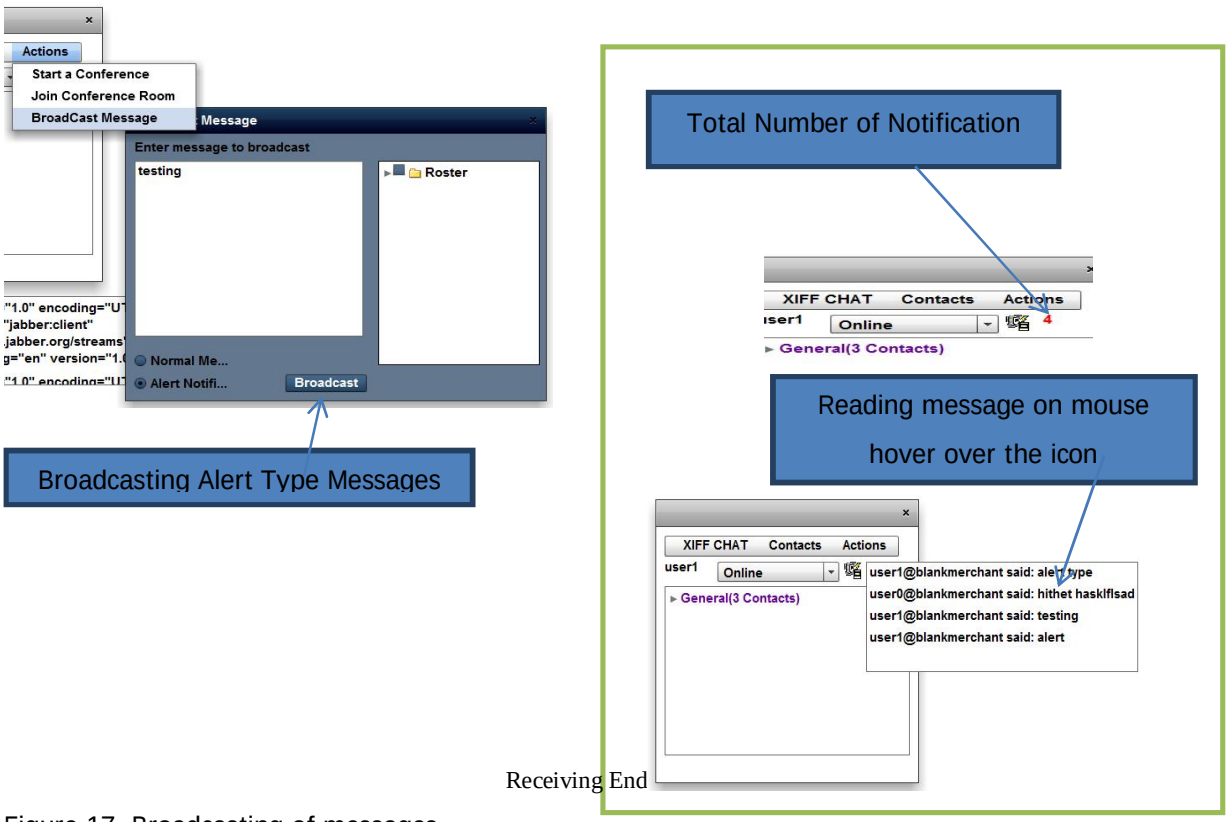


Figure 17. Broadcasting of messages.

Figure 17 shows the broadcasting of alert type messages and the view that is displayed at the receivers end. Since broadcasting of coordinates is very important use case for KP-lab, this application also includes similar feature. In this application, broadcasting of coordinates is utilized to draw some figures on the drawing board based on the coordinates broadcasted. Figure 18 displays the figure drawn based on the received coordinates of mouse pointer which is exactly similar to the one drawn by the broadcaster

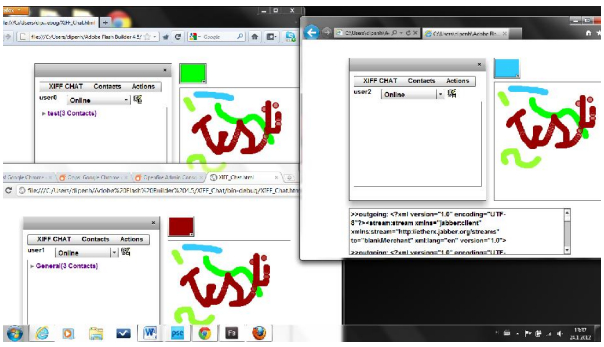


Figure 18. Image drawn based on co broadcasted coordinates.

Figure 18 also displays the application's compatibility with three major browsers: Mozilla Firefox, Google Chrome and Internet Explorer. Also it shows how efficiently Openfire broadcasts the coordinates.

7 Test and Result Analysis

The application worked very smoothly when only two users are communicating with each other or a user broadcasts message to all the registered users. The total number of registered users was only twenty five (25) in the beginning. Since KPE is an online collaboration tool that has thousands of registered users, in worst case scenario there might come a time when all of those users log in and start using the service simultaneously. In order to cope with that situation, Openfire should be capable of handling all the transactions smoothly without loss of data and on timely manner. Failure to do so would cause system failure and the idea of real-time collaboration would make no sense.

7.1 Testing Real Time Collaboration Server

Registering thousands of users for this small scale research project was not possible and even asking them to use the application simultaneously was out of question. In order to test the XMPP Server, Openfire's capability and performance in the worst case scenario, a simple system was again created with one server (Openfire), MySQL database and two clients in LAN. Figure 19 illustrates the system architecture and the system configuration of the server and clients used.

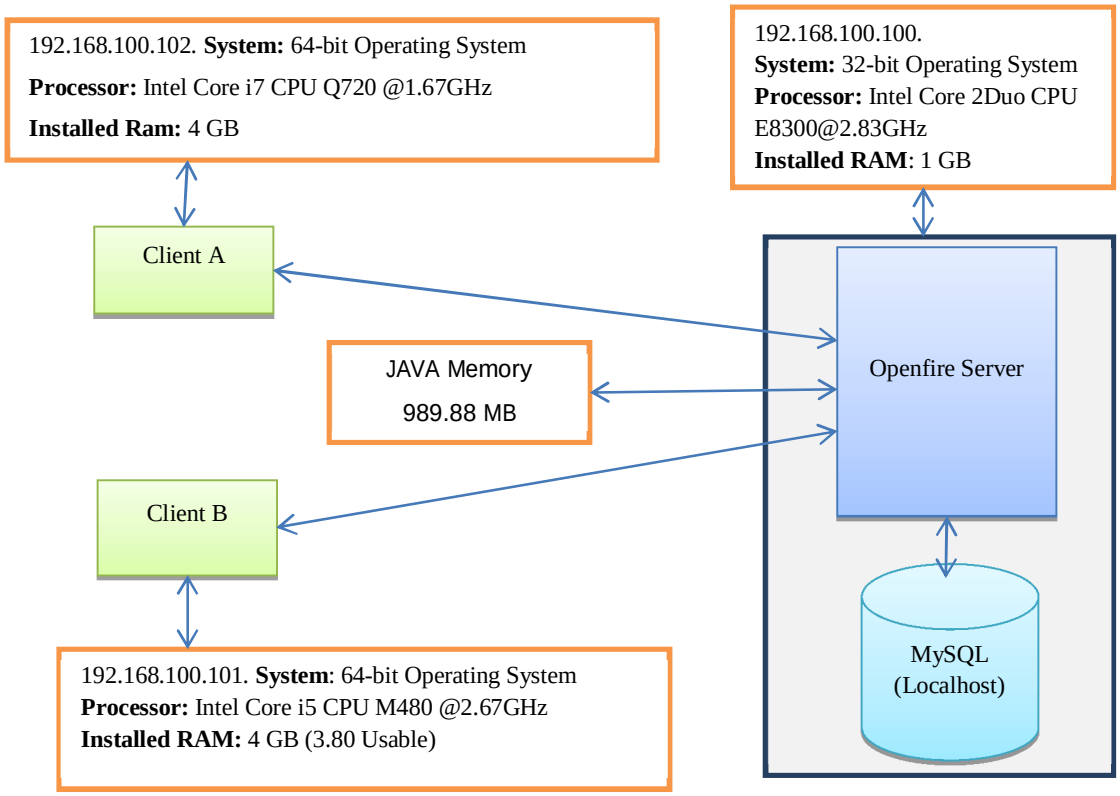


Figure 19. System Architecture for the application.

Openfire, real time collaborating server and MySQL database were installed in the same computer. The server was optimized to fully utilize the memory and was assigned almost all available memory capacity to JVM. The server was connected to the external MySQL database installed on the same computer accessible over localhost. Using the user service plugin, five thousands users were created and registered to the server. In order to test the server, the Grinder was used for load testing. It used the Jython to send multiple HTTP requests to the server from two clients. Table 7 shows the piece of script used to send HTTP request.

Table 7. Jython Script used for sending HTTP request.

```
agentID = int(grinder.properties["grinder.agentID"])
processID = int(grinder.processName.split("-").pop())
host = '192.168.100.100'
domain = 'dipenh-pc'
boshUrl = 'HTTP://' + host + ':7070/HTTP-bind/'
boshWait = 1
userPrefix = 'user'
numThreads = 1
# Create an HTTPRequest for each request
request101 = Test(101, 'Initiate a BOSH session').wrap(HTTPRequest(url=boshUrl))
request201 = Test(201, 'Authenticate').wrap(HTTPRequest(url=boshUrl))
request301 = Test(301, 'Bind resource').wrap(HTTPRequest(url=boshUrl))
request401 = Test(401, 'Request a session from the server').wrap(HTTPRequest(url=boshUrl))
request501 = Test(501, 'Get roster').wrap(HTTPRequest(url=boshUrl))
request601 = Test(601, 'Change presence').wrap(HTTPRequest(url=boshUrl))
request701 = Test(701, 'Send one to one message').wrap(HTTPRequest(url=boshUrl))
request801 = Test(801, 'Make an empty request to the server').wrap(HTTPRequest(url=boshUrl))
request901 = Test(901, 'Terminate the session').wrap(HTTPRequest(url=boshUrl))
```

Most of the time only one client, Client A as shown in figure 19 was used to send the HTTP request. Final test was done using both clients. Several tests were carried out before actually recording the tests. Three tests, two by Client A and the final test using both Client A and Client B were recorded for further analysis. For test one (1), Client A used five (5) processes and one hundred (100) threads in two (2) runs. For test two (2), five (5) processes, one thousand (1000) threads in three (3) runs. For the final test, Client A used five (5) processes and one thousand (1000) threads in two(2) runs and Client B used six (6) processes and one thousand (1000) threads in one (1) run.

The Grinder processed the entire HTTP request to be sent to Openfire. The test involved sending nine (9) different types of requests using a random usernames from among the registered users. Following are the lists of HTTP request sent to the server:

- request 101: Initiate BOSH session for HTTP bind.
- request 201: User Authentication
- request 301: Binding resource
- request 401: Requesting a session from Openfire
- request 501: Getting Roster
- request 601: Changing Presence of the authenticated user
- request 701: Sending one to one message to random user.

- request 801: Pinging to the server. Sending empty request
- request 901: Finally terminating the session.

7.2 Result and Analysis

While running the test, hundreds and thousands of users were authenticated to Openfire via HTTP request. All of those users were authenticating, requesting session, fetching roster, changing their presence information, sending messages, sending empty request and terminating the session simultaneously. During this test session, the server was recorded consuming JAVA memory up to three hundred and seventy eight point thirty three (378.33) MB which is just over thirty eight (38) percentage of the total memory assigned. Since the Monitoring Service plugin was installed to the server, it was also easy to monitor the number of connected and active users, packets count or number of packets sent and received by server, active conversations in the admin console provided by the server. Figure 20 generated by monitoring service plugin displays the information about the connected users, active conversations and packets exchanged per minute during the first recorded test session.

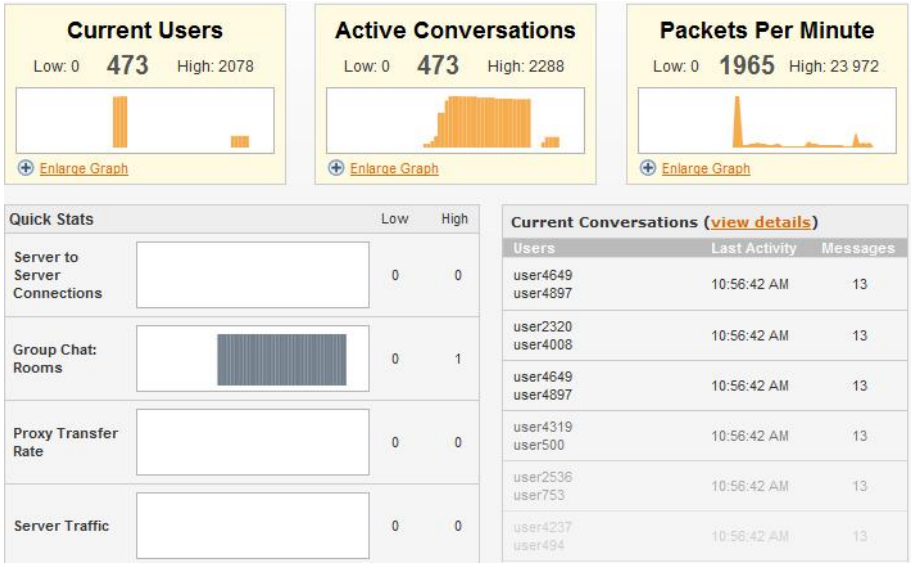


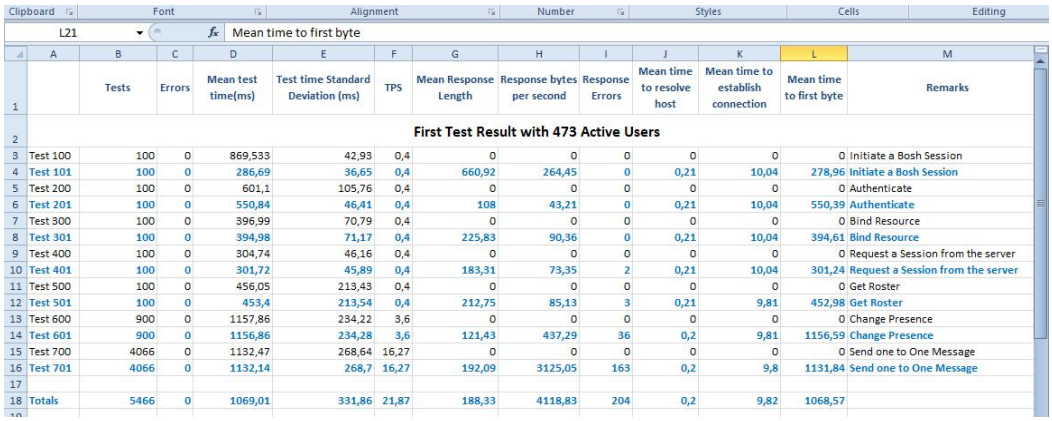
Figure 20. Image generated by Monitoring Service Plugin.

During the test, admin console of the Openfire was a bit slower and after the number of connected clients reached above two thousand, admin console was very slow to respond and the monitoring service was not able to perform at all. It was not able to generate any reports regarding the number of connected users, active conversations

and also information about the packets exchanged. These reports generated by using the monitoring service were mere reflection of information regarding the connected users and packets exchanged. In order to measure the real time nature, response time from the server and it's reliability were the most important thing to be measured. Appendix 3 contains the pictorial representation of first test, showing number of active users and the number of packets transferred in detail.

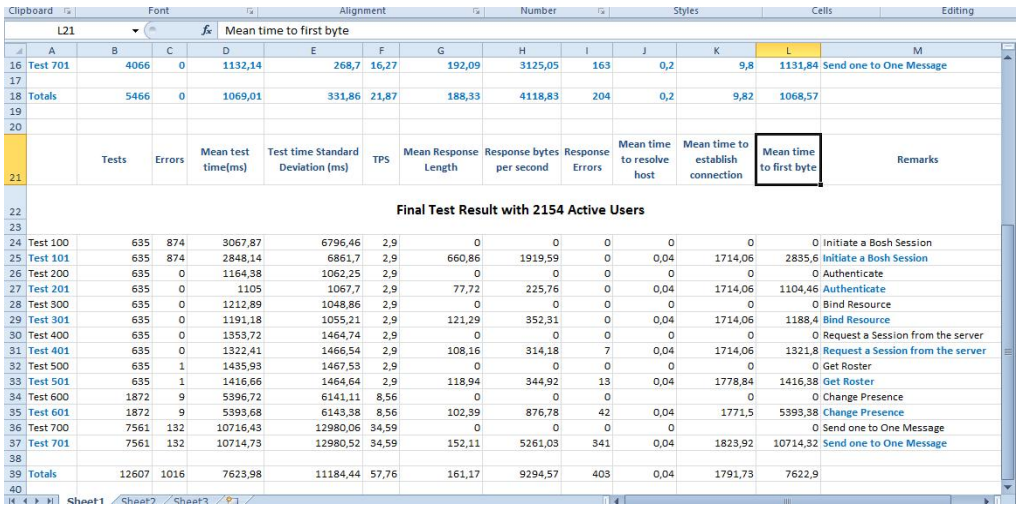
The Grinder framework has additional feature of recording all the test results in a separate log directory. It collects and records all the data related to the HTTP request and time measurement of all the response to the request. It records all the details of the necessary information and separately for each process created. Table 8 and 9 illustrate the summary of information collected during the first and final tests respectively.

Table 8. Time Measurement when 473 users are active.



	Tests	Errors	Mean test time(ms)	Test time Standard Deviation (ms)	TPS	Mean Response Length	Response bytes per second	Response Errors	Mean time to resolve host	Mean time to establish connection	Mean time to first byte	Remarks
First Test Result with 473 Active Users												
Test 100	100	0	869,533	42,93	0,4	0	0	0	0	0	0	Initiate a Bosh Session
Test 101	100	0	286,69	36,65	0,4	660,92	264,45	0	0,21	10,04	278,96	Initiate a Bosh Session
Test 200	100	0	601,1	105,76	0,4	0	0	0	0	0	0	Authenticate
Test 201	100	0	550,84	46,41	0,4	108	43,21	0	0,21	10,04	550,39	Authenticate
Test 300	100	0	396,99	70,79	0,4	0	0	0	0	0	0	Bind Resource
Test 301	100	0	394,98	71,17	0,4	225,83	90,36	0	0,21	10,04	394,61	Bind Resource
Test 400	100	0	304,74	46,16	0,4	0	0	0	0	0	0	Request a Session from the server
Test 401	100	0	301,72	45,89	0,4	183,31	73,35	2	0,21	10,04	301,24	Request a Session from the server
Test 500	100	0	456,05	213,43	0,4	0	0	0	0	0	0	Get Roster
Test 501	100	0	453,4	213,54	0,4	212,75	85,13	3	0,21	9,81	452,98	Get Roster
Test 600	900	0	1157,86	234,22	3,6	0	0	0	0	0	0	Change Presence
Test 601	900	0	1156,86	234,28	3,6	121,43	437,29	36	0,2	9,81	1156,59	Change Presence
Test 700	4066	0	1132,47	268,64	16,27	0	0	0	0	0	0	Send one to One Message
Test 701	4066	0	1132,14	268,7	16,27	192,09	3125,05	163	0,2	9,8	1131,84	Send one to One Message
Totals	5466	0	1069,01	331,86	21,87	188,33	4118,83	204	0,2	9,82	1068,57	

Table 9. Time Measurement when 2154 users were active.



	Tests	Errors	Mean test time(ms)	Test time Standard Deviation (ms)	TPS	Mean Response Length	Response bytes per second	Response Errors	Mean time to resolve host	Mean time to establish connection	Mean time to first byte	Remarks
Test 701	4066	0	1132,14	268,7	16,27	192,09	3125,05	163	0,2	9,8	1131,84	Send one to One Message
Totals	5466	0	1069,01	331,86	21,87	188,33	4118,83	204	0,2	9,82	1068,57	
Final Test Result with 2154 Active Users												
Test 100	635	874	3067,87	6796,46	2,9	0	0	0	0	0	0	Initiate a Bosh Session
Test 101	635	874	2848,14	6861,7	2,9	660,86	1919,59	0	0,04	1714,06	2835,6	Initiate a Bosh Session
Test 200	635	0	1164,38	1062,25	2,9	0	0	0	0	0	0	Authenticate
Test 201	635	0	1105	1067,7	2,9	77,72	225,76	0	0,04	1714,06	1104,46	Authenticate
Test 300	635	0	1212,89	1048,86	2,9	0	0	0	0	0	0	Bind Resource
Test 301	635	0	1191,18	1055,21	2,9	121,29	352,31	0	0,04	1714,06	1188,4	Bind Resource
Test 400	635	0	1353,72	1464,74	2,9	0	0	0	0	0	0	Request a Session from the server
Test 401	635	0	1322,41	1466,54	2,9	108,16	314,18	7	0,04	1714,06	1321,8	Request a Session from the server
Test 500	635	1	1435,93	1467,53	2,9	0	0	0	0	0	0	Get Roster
Test 501	635	1	1416,66	1464,64	2,9	118,94	344,92	13	0,04	1778,84	1416,38	Get Roster
Test 600	1872	9	5396,72	6141,11	8,56	0	0	0		0	0	Change Presence
Test 601	1872	9	5393,68	6143,38	8,56	102,39	876,78	42	0,04	1771,5	5393,38	Change Presence
Test 700	7561	132	10716,43	12980,06	34,59	0	0	0	0		0	Send one to One Message
Test 701	7561	132	10714,73	12980,52	34,59	152,11	5261,03	341	0,04	1823,92	10714,32	Send one to One Message
Totals	12607	1016	7623,98	11184,44	57,76	161,17	9294,57	403	0,04	1791,73	7622,9	

Out of the information collected using the Grinder, time to resolve host, time to establish connection, and time to first byte were of most importance. Also, the number of HTTP errors would provide important information regarding the reliability of the real time server. Table 8, representing the first test with 473 active users, shows that the Grinder carried out five thousand four hundred and sixty six (5466) successful tests without any failures and only two hundred and four (204) response errors. Mean time to resolve host, identifying i.p. address of the server from DNS was only zero point two (0.2) milliseconds (ms) and it took fraction of second to establish connection with the server which certainly proved that Openfire is very efficient. Average time to receive first byte from the server was just fractions of seconds more than one (1) second. This test displays the real time nature of the server.

Compared to the first test, final test using two thousand one hundred and fifty four (2154) users executed using two different clients showed a bit different result. As shown in the table 8, the Grinder executed twelve thousand six hundred and seven (12607) successful test but with one thousand and sixteen (1016) failures. However compared to the mass number of tests, HTTP response error was only four hundred and three (403). Time to resolve host was zero point zero four (0.04) ms and time to establish connection with the Openfire was one thousand seven hundred and ninety one point seventy three (1791.73) ms which is more compared to the first test. Also average time taken to receive first byte was seven thousand six hundred and twenty two point nine (7622.9) ms which is comparatively very long.

During the final test, CPU usage of both the clients reached 100 percent and the memory usage was recorded even up to 3.8 GB. Due to the maximum usage of memory, JRE ran out of memory and The Grinder killed few processes during the tests. One of the clients crashed and had to be stopped. Both the clients were very slow due to excessive usage of CPU and memory. Also this could be the reason why the time to receive first byte was longer compared to the first test because this could have created longer time lag or delay between starting the test for individual threads and sending HTTP requests. At the same time, Openfire had only 16.22 MB of segmented cached memory of which segments for offline messages, user presences might have been totally consumed during the first test. Openfire depends on its cached memory for better performance. Due to the lesser remaining cached memory; it must have served the

compromised performance. However, it did not crash during this test and it was recorded to consume 421 MB of memory at some point during the test.

8 Conclusion

Instant response time from Openfire and reliability on delivering message in a secure way proved it to be a real time collaboration server. Such features enable it to be an alternative to flash media server. It certainly performed better when there were only hundreds of users logged in simultaneously. Also the final test with thousands of users displayed the server's capability and real time nature even though the server's capacity was very limited. Bearing in mind that Openfire was installed in a machine with similar configuration and memory as a flash media server, its performance could have been outstanding. The question whether the real time collaboration server, Openfire could replace Flash Media Server could have been easily answered if the whole project implementation was done for KP-lab itself. However, there were certain technical problems that prevented for doing so. In order to test it, different implementations were done allowing KP-lab to run on flash media server to prevent any cause of failure that might stop KP-Lab or affect its performance.

If the test was done for KP-lab, there was maximum probability of having two issues: data redundancy and security & synchronization. Other problems could also occur but those were minor issues and could be ignored. Problem with data redundancy could occur because Openfire has its own database schema that defines relationship between different tables. This schema could be used in the existing database or the Openfire could be configured to authenticate users from the existing database. However, the access to the existing database was restricted and not accessible. For this reason, a separate database was needed and the table associated with users of newly created database had to be filled with all users' information in order to authenticate against the Openfire.

The second problem with security issues and data synchronization could occur if we tried to overcome the first problem. Users could authenticate against Openfire as anonymous users. However, if anonymous login was allowed, there would be a maximum possibility that anyone could send HTTP request to Openfire as an anonymous

user which could bring unwanted messages and results. Also, there could be a serious problem of data synchronization due to authorization issues if the users were using the service as anonymous users. Furthermore, single sign on mechanism could be employed for authenticating users. However, JOSSO authentication was implemented for KP-lab and Openfire could provide SSO service only via GSSAPI or Kerberos.

Based on the performance of the Openfire and the test results, it certainly proved XMPP to be the best protocol for real time communication and turned out to be one of the best platforms for real time communication and collaboration. However, it lags behind flash media server in some of the sectors. Unlike flash media server, it does not support exchanging of audio/ video streams and remote shared objects. However there is already a plugin called Redfire, red5 plugin for Openfire, which could be used to deliver audio/video stream along with XMPP messaging and signaling. In place of remote shared object, Openfire could use pubsub, powerful protocol extension to XMPP to subscribe to items and getting updates from them upon information updates.

Openfire's performance capability is unquestionable. Internet giant, Google used Openfire as XMPP server for Google Wave. Openfire was used as federated server in order to communicate between the servers using the federation protocol, extension of XMPP protocol. Even though Google Wave was a major failure not because it was technically inefficient but because of other reasons, it was very sophisticated and real time in nature. Because Openfire is being used increasingly for commercial purposes in many institutions and organizations, it is developing and improving rapidly. One of its advantages is that it can support for plugins and XMPP extensively.

Openfire is not just platform independent and based on open protocol; it is open source which gives it a huge advantage over flash media server. Since XMPP is also capable of cross protocol communication with Yahoo, AIM, MSN and other protocol, Openfire gains more advantage over flash media server. Although, Openfire may not provide support for audio/video streams, KP-lab does not provide any service that involves audio/video. Therefore, flash media server's advantage over Openfire might not be so meaningful unless KP-lab starts services related to media streaming. With so many features and extensibility, Openfire could be used in place of flash media server. It seems quite useful for KP-lab to start using Openfire and actually testing the perfor-

mance. Even though replacement of flash media server immediately could mean a disaster for KP-lab, it would be a good idea to start using some services provided by Openfire in the beginning and gradually move towards replacing service provided by flash media server one after another.

References

1. Real time Collaboration [online].
   URL: HTTP://searchdomino.techtarget.com/definition/real-time-collaboration.
   Accessed 3 February 2012.

2. Stonebraker Michael. The 8 Requirements of Real-Time Stream Processing [online].
   URL: HTTP://www.cs.brown.edu/~ugur/8rulesSigRec.pdf
   Accessed 27 February 2012.

3. KP-lab. Short Introduction [online].
   URL: HTTP://www.KP-lab.org/project-overview/objectives-of-the-project.
   Accessed 5 February 11 2012.

4. Adobe. Adobe Flash Media Server Installation Guide [online].
   URL:HTTP://livedocs.adobe.com/flashmediaserver/3.0/docs/
   flashmediaserver_install.pdf.
   Accessed 5 February 2012.

5. Tzurel Avi. Flash Media Server Family [online].
   URL: HTTP://www.slideshare.net/DSPIP/flash-media-server-2990582
   Accessed 4 February 2012.

6. Adobe. Flash Media Server 3 [online].
   URL: HTTP://www.scribd.com/doc/40390941/16/
   Flash-Media-Server-communication-protocol-RTMP
   Accessed 3 February 2012.

7. Introduction to Real Time Messaging Protocol [online].
   URL: HTTP://www.playerdiy.com/blog/
   introduction-of-rtmp-real-time-messaging-protocol
   Accessed 3 February 2012.

8. Chandra Praphul, Lide David. Wi-Fi Telephoney: Challenges and Solutions for Voice over WLANs. Burlington, USA: Elsevier Inc.; 2007.

9. Rhee Man young. Internet Security: Cryptographic principles, algorithms and protocols. West Sussex, England: John Wiley & Sons Inc.; 2003.

10. Lee Stephen, Smelser Terence. Jabber Programming. NY, USA: M&T Books; 2002.

11. Isode. Isode's Presence, Real Time Messaging and XMPP Strategy [online].
URL: HTTP://www.isode.com/whitepapers/xmpp.html.
Accessed 26 December 2011.

12. Moffitt Jack. Professional XMPP programming with javascript and jquery. USA: Wrox Press; January 2010.

13. Saint-Andre Peter, Smith Kevin, Troncon Remko. XMPP: The Definitive Guide. CA, USA: O'Reilly Media Inc.; 2009.

14. Elankumaran Pradeep. Why XMPP will be huge very soon [online]. Intridea; 16 February 2009.
URL: HTTP://intridea.com/2009/2/16/
why-xmpp-will-be-huge-very-soon?blog=company
Accessed 2 January 2012.

15. Saint-Andre Peter. XEP-0001: XMPP Extension Protocols [online]. XMPP standards foundation; 10 March 2012.
URL: HTTP://xmpp.org/extensions/xep-0001.html#intro.
Accessed 5 January 2012.

16. Sharma Mayank. Openfire Administratin: A practical step-by-step guide to rolling out a secure instant Messaging service over your network. Birmingham, UK: Packt. Publishing Ltd.; 2008

17. Ignite realtime. Protocol Support [online].
URL:HTTP://www.igniterealtime.org/builds/openfire/docs/latest/documentation/protocol-support.html.
Accessed 6 January 2012.

18. Aston Philip. The Grinder, a java load testing framework [online].
URL: HTTP://grinder.sourceforge.net/.
Accessed 8 January 2012.

Appendices

Appendix 1: List of Active and Final XEPs.

Number	Name	Type	Status	Date
XEP-0001 (PDF)	XMPP Extension Protocols	Procedural	Active	2010-03-
XEP-0002 (PDF)	Special Interest Groups (SIGs)	Procedural	Active	2002-01-
XEP-0004 (PDF)	Data Forms	Standards	Final	2007-08-
XEP-0009 (PDF)	Jabber-RPC	Standards	Final	2011-11-
XEP-0012 (PDF)	Last Activity	Standards	Final	2008-11-
XEP-0019 (PDF)	Streamlining the SIGs	Procedural	Active	2002-03-
XEP-0027 (PDF)	Current Jabber OpenPGP Usage	Historical	Active	2006-11-
XEP-0030 (PDF)	Service Discovery	Standards	Final	2008-06-
XEP-0049 (PDF)	Private XML Storage	Historical	Active	2004-03-
XEP-0053 (PDF)	XMPP Registrar Function	Procedural	Active	2008-10-
XEP-0054 (PDF)	vcard-temp	Historical	Active	2008-07-
XEP-0055 (PDF)	Jabber Search	Historical	Active	2009-09-
XEP-0068 (PDF)	Field Standardization for Data Forms	Informational	Active	2011-10-
XEP-0076 (PDF)	Malicious Stanzas	Humorous	Active	2003-04-
XEP-0077 (PDF)	In-Band Registration	Standards	Final	2012-01-
XEP-0082 (PDF)	XMPP Date and Time Profiles	Informational	Active	2003-05-
XEP-0083 (PDF)	Nested Roster Groups	Informational	Active	2004-10-
XEP-0085 (PDF)	Chat State Notifications	Standards	Final	2009-09-
XEP-0100 (PDF)	Gateway Interaction	Informational	Active	2005-10-
XEP-0114 (PDF)	Jabber Component Protocol	Historical	Active	2012-01-
XEP-0126 (PDF)	Invisibility	Informational	Active	2005-08-
XEP-0127 (PDF)	Common Alerting Protocol (CAP) Over	Informational	Active	2004-12-
XEP-0128 (PDF)	Service Discovery Extensions	Informational	Active	2004-10-
XEP-0130 (PDF)	Waiting Lists	Historical	Active	2006-09-
XEP-0132 (PDF)	Presence Obtained via Kinesthetic Excita-	Humorous	Active	2004-04-
XEP-0133 (PDF)	Service Administration	Informational	Active	2005-08-
XEP-0134 (PDF)	XMPP Design Guidelines	Informational	Active	2004-12-
XEP-0138 (PDF)	Stream Compression	Standards	Final	2009-05-
XEP-0143 (PDF)	Guidelines for Authors of XMPP Extension	Procedural	Active	2011-07-
XEP-0145 (PDF)	Annotations	Historical	Active	2006-03-
XEP-0146 (PDF)	Remote Controlling Clients	Informational	Active	2006-03-

Number	Name	Type	Status	Date
XEP-0147 (PDF)	XMPP URI Scheme Query Components	Informational	Active	2006-09-
XEP-0148 (PDF)	Instant Messaging Intelligence Quotient	Humorous	Active	2005-04-
XEP-0149 (PDF)	Time Periods	Informational	Active	2006-01-
XEP-0153 (PDF)	vCard-Based Avatars	Historical	Active	2006-08-
XEP-0157 (PDF)	Contact Addresses for XMPP Services	Informational	Active	2007-01-
XEP-0160 (PDF)	Best Practices for Handling Offline Mes-	Informational	Active	2006-01-
XEP-0169 (PDF)	Twas The Night Before Christmas (Jabber	Humorous	Active	2009-12-
XEP-0170 (PDF)	Recommended Order of Stream Feature	Informational	Active	2007-01-
XEP-0174 (PDF)	Serverless Messaging	Standards	Final	2008-11-
XEP-0175 (PDF)	Best Practices for Use of SASL ANONY-	Informational	Active	2009-09-
XEP-0178 (PDF)	Best Practices for Use of SASL EXTERNAL	Informational	Active	2011-05-
XEP-0182 (PDF)	Application-Specific Error Conditions	Procedural	Active	2008-03-
XEP-0183 (PDF)	Jingle Telepathy Transport	Humorous	Active	2006-04-
XEP-0185 (PDF)	Dialback Key Generation and Validation	Informational	Active	2007-02-
XEP-0199 (PDF)	XMPP Ping	Standards	Final	2009-06-
XEP-0201 (PDF)	Best Practices for Message Threads	Informational	Active	2010-11-
XEP-0202 (PDF)	Entity Time	Standards	Final	2009-09-
XEP-0203 (PDF)	Delayed Delivery	Standards	Final	2009-09-
XEP-0205 (PDF)	Best Practices to Discourage Denial of	Informational	Active	2009-01-
XEP-0207 (PDF)	XMPP Eventing via Pubsub	Humorous	Active	2007-04-
XEP-0222 (PDF)	Persistent Storage of Public Data via	Informational	Active	2008-09-
XEP-0223 (PDF)	Persistent Storage of Private Data via	Informational	Active	2008-09-
XEP-0239 (PDF)	Binary XMPP	Humorous	Active	2008-04-
XEP-0245 (PDF)	The /me Command	Informational	Active	2009-01-
XEP-0263 (PDF)	ECO-XMPP	Humorous	Active	2009-04-
XEP-0295 (PDF)	JSON Encodings for XMPP	Humorous	Active	2011-04-

Appendix 2: List of XEPs supported by Openfire.

Specification	Suite
XEP-0004: Data Forms	Intermediate
XEP-0012: Last Activity	-
XEP-0013: Flexible Offline Message Retrieval	-
XEP-0030: Service Discovery	Basic
XEP-0033: Extended Stanza Addressing	-
XEP-0045: Multi-User Chat	Intermediate
XEP-0049: Private XML Storage	-
XEP-0050: Ad-Hoc Commands	-
XEP-0054: vcard-temp	-
XEP-0055: Jabber Search [2]	-
XEP-0059: Result Set Management	-
XEP-0060: Publish-Subscribe	-
XEP-0065: SOCKS5 Bytestreams	Intermediate
XEP-0077: In-Band Registration	Basic
XEP-0078: Non-SASL Authentication	Basic
XEP-0082: Jabber Date and Time Profiles	-
XEP-0086: Error Condition Mappings	Basic
XEP-0090: Entity Time	-
XEP-0091: Legacy Delayed Delivery	-
XEP-0092: Software Version	-
XEP-0096: File Transfer	Intermediate
XEP-0106: JID Escaping	-
XEP-0114: Jabber Component Protocol	-
XEP-0115: Entity Capabilities	Intermediate
XEP-0124: HTTP Binding	-
XEP-0126: Invisibility	-
XEP-0128: Service Discovery Extensions	-
XEP-0138: Stream Compression	-
XEP-0163: Personal Eventing via Pubsub	-
XEP-0175: Best Practices for Use of SASL ANONYMOUS	-
XEP-0203: Delayed Delivery	-

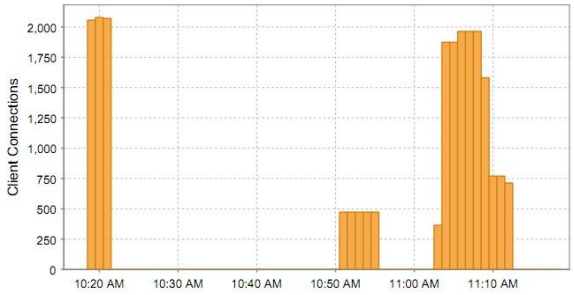
Appendix 3: Report generated by monitoring service plugin.

dipenh-pc
Jan 31, 2012 - Jan 31, 2012

1. Conversations
Conversations between users.

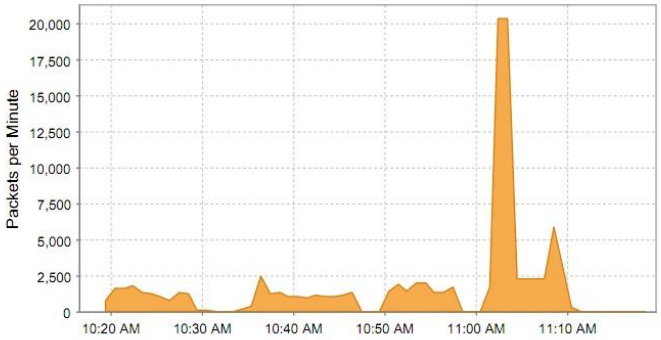
dipenh-pc
Jan 31, 2012 - Jan 31, 2012

1. Client Connections
Number of Clients Connected Directly to the Server.



dipenh-pc
Jan 31, 2012 - Jan 31, 2012

1. Packet Count
Number of Packets Sent and Received by Server.



Appendix 4: Glossary of Terms

APIs
: interface implemented by an application which allows other applications to communicate with it.

authentication
: the process of determining whether someone or something is, in fact, who or what it is claims to be.

clustering
: connecting two or more computers together in such ways that they behave like a single computer.

connectionless protocol
: refers to network protocols in which a host can send a message without establishing a connection with the recipient.

cross-platform
: that can run on any platform or operating systems.

database
: organized collection of data.

data redundancy
: repetition of field in two or more tables in a database system..

event
: an action that is usually initiated outside the scope of a program and that is handled by a piece of code inside the program.

eventhandler
: an asynchronous callback subroutine that handles inputs received in a program.

firewall
: a device or set of devices designed to permit or deny network transmissions based upon a set of rules and is frequently used to protect networks from unauthorized access.

IDEs
: software applications that provide comprehensive facilities to computer programmers for software development.

latency
: the time required for receiving input and responding to the received input.

load balancing
: distributing workload across multiple computers or a computer cluster.

logging

keeping log of all communications.

Openfire

real time collaboration server based on XMPP protocol.

packet

the unit of data that is routed between an origin and a destination on the Internet or any other packet-switched network.

peer-to-peer

also termed as P2P, it refers to a computer network in which each computer in the network can act as a client or server for the other.

plugins

tools or application to extend the functionality.

polling

refers to actively sampling the status of an external device by a client program as a synchronous activity.

presence

information that conveys ability and willingness of a potential communication partner. Such as "Free to chat", "Away", "Offline".

protocol

special set of rules that communicating components should follow in order to communicate.

roster

the name of the contact list for XMPP.

scalability

the ability to retain performance levels when adding additional processors.

schema

cognitive framework or concept that helps organize and interpret information.

server

a program or a computer that fulfills request of the client programs/computers.

service discovery

automatic detection of devices and services offered by these devices on a com puter network.

session

a series of interactions between two communication end points that occur during the span of a single connection.

SSO Authentication

Single sign-on (SSO) is a property of access control of multiple related, but independent software systems. It is the ability for a user to enter the same id

and password to logon to multiple applications within an enterprise.

stateless protocol

a communications protocol that treats each request as an independent transaction that is unrelated to any previous request.

stream

to transfer data to a computer so that it can be used as it is downloaded.

synchronization

an adjustment that causes something to occur or recur in unison.

system architecture

the conceptual model that defines the structure, behavior, and more views of a system.

throughput

output relative to input.