NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A PRELIMINARY ANALYSIS FOR PORTING XML-BASED CHAT TO MYSEA

by

Claire E. R. LaVelle

June 2008

Thesis Advisor: Cynthia E. Irvine
Co-Advisor: Thuy D. Nguyen

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2008	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE : A Preliminary Analysis for Porting XML-based Chat to MYSEA		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Claire E. R. LaVelle			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

The Monterey Security Architecture (MYSEA) is a distributed multilevel secure (MLS) computing environment. MYSEA does not presently support chat, an Internet application that provides near-real-time collaboration capability. Chat capability that implements the Extensible Messaging and Presence Protocol (XMPP) standards has been recognized by the Department of Defense (DoD) as a mandatory standard. The primary goal of this thesis is to determine if a chat server that implements the XMPP and the XMPP Instant Messaging (XMPP-IM) standards could be ported to MYSEA.

To accomplish this goal, a set of selection criteria was developed and the open-source jabberd14 server was selected for this study. Its functionality was tested on different operating system environments (Fedora 7, RedHat 8, STOP OS 7 beta). This study also includes a functional analysis of the XMPP and XMPP-IM specifications, the related XMPP extensions supported by the jabberd14 server, a preliminary security analysis and a survey of the jabberd14 server code.

The results of this project show that implementation of the XMPP jabberd14-1.6.0 server on the MYSEA platform under STOP 7 OS is feasible. The results also provide stepping stones toward a full-scale development effort to provide MLS-aware chat services in the MYSEA network.

14. SUBJECT TERMS Information assurance, MLS, Chat, XMPP, MYSEA, Jabber			**15. NUMBER OF PAGES** 145
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

A PRELIMINARY ANALYSIS FOR PORTING XML-BASED CHAT TO MYSEA

Claire E. R. LaVelle
Civilian, Naval Postgraduate School
M.A., Mills College, 2006

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2008

Author: Claire E. R. LaVelle

Approved by: Cynthia E. Irvine, Ph.D.
Thesis Advisor

Thuy D. Nguyen
Co-Advisor

Peter J. Denning, Ph.D.
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Monterey Security Architecture (MYSEA) is a distributed multilevel secure (MLS) computing environment. MYSEA does not presently support chat, an Internet application that provides near-real-time collaboration capability. Chat capability that implements the Extensible Messaging and Presence Protocol (XMPP) standards has been recognized by the Department of Defense (DoD) as a mandatory standard. The primary goal of this thesis is to determine if a chat server that implements the XMPP and the XMPP Instant Messaging (XMPP-IM) standards could be ported to MYSEA.

To accomplish this goal, a set of selection criteria was developed and the open-source jabberd14 server was selected for this study. Its functionality was tested on different operating system environments (Fedora 7, RedHat 8, STOP OS 7 beta). This study also includes a functional analysis of the XMPP and XMPP-IM specifications, the related XMPP extensions supported by the jabberd14 server, a preliminary security analysis and a survey of the jabberd14 server code.

The results of this project show that implementation of the XMPP jabberd14-1.6.0 server on the MYSEA platform under STOP 7 OS is feasible. The results also provide stepping stones toward a full-scale development effort to provide MLS-aware chat services in the MYSEA network.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS AND ACRONYMS

ACL	Access Control List
AES	Advanced Encryption Standard
BBS	Bulletin Board System
C	Confidential
COTS	Commercial Off The Shelf
CVS	Version Control System
DAC	Discretionary Access Control
DNS	Domain Name Server
DoD	Department of Defense
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transport Protocol
IETF	Internet Engineering Task Force
IM	Instant Messaging
IMAP	Internet Message Access Protocol
IP	Internet Protocol
IRC	Internet Relay Chat
JID	Jabber Identifier
JUD	Jabber User Directory
MAC	Mandatory Access Control
MLS	Multilevel Secure/Multilevel Security

MUC	Multi-User Chat
MYSEA	Monterey Security Architecture
OLM	On-line Messages
POSIX	Portable Operating System Interface
RBAC	Role-Based Access Control
RFC	Request For Comments
S	Secret
SASL	Simple Authentication and Security Layer
SHA	Secure Hash Algorithm
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SSS	Secure Session Services
STOP	Secure Trusted Operating System
SVN	Subversion
TCB	Trusted Computing Base
TCP	Transport Control Protocol
TLS	Transport Layer Security
TOE	Target of Evaluation
TS	Top Secret
XEP	XMPP Extension Protocols
XSF	XMPP Standards Foundation
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

ACKNOWLEDGMENTS

I would like to thank Dr. Cynthia Irvine and Thuy Nguyen for their guidance and patience all through the process of this thesis.

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0414102. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

Near-real-time communication capability is a valuable collaboration tool. Chat provides that service via instant messaging and group chat room. Today, there are many chat protocols[1], but XMPP has been selected by the Department of Defense (DoD)'s Information Technology Standards Council as a "mandatory standard." [1]. As a result, incorporating chat capability into the Monterey Security Architecture (MYSEA) environment is highly desirable. MYSEA provides a secure distributed networking computing environment that supports commercial-off-the-shelf (COTS) products augmented with high assurance components that enforce the multilevel security (MLS) policy. However, the multilevel nature of the MYSEA environment requires special testing and a complex porting effort. This study is the first step towards porting chat to an MLS environment such as MYSEA.

B. PURPOSE OF STUDY

The goal of this study is to elicit requirements for a chat server that follows the XMPP standards [2], [3] and determine the best open source XMPP server available today that could be ported to run on a high assurance MLS server. To validate the choice made based on a set of selection criteria, the selected XMPP server is installed on different platforms and its basic behavior is tested. The lessons learned from the experimentation will help to define the important technical and security issues that will need to be addressed to develop an MLS-aware chat service for the MYSEA environment.

[1] Examples of other protocols are AIM [4], Sametime [5], ICQ [6], and IRC [7], [8].

C. ORGANIZATION OF THIS STUDY

This study presents its results in the following structure:

- Chapter II gives background information relevant to the porting of an XMPP chat server to the MYSEA environment. The topics addressed are chat, Extended Messaging and Presence Protocol (XMPP), Jabber and the Monterey Security Architecture (MYSEA).
- Chapter III describes the process of selecting an XMPP open source chat server to port to the MYSEA environment.
- Chapter IV reports on the installation and testing of the selected XMPP chat server.
- Chapter V discusses the technical and security issues that need to be addressed before the porting of the XMPP server can start.
- Chapter VI outlines future work that needs to be performed in order to accomplish a successful port of the XMPP server, and provides a conclusion to this study.

II. BACKGROUND

This chapter provides background information on various topics relevant to the porting of a jabber server in the MYSEA environment. Section A relates to chat, Section B to XMPP, Section C to Jabber, Section D to Monterey Security Architecture (MYSEA) and the last section relates to the XTS-400.

A. CHAT

Instant Messaging (IM) is pre-Internet (mid 1960s) [9] and can be equated to instantaneous e-mail. It allows two users to communicate based on typed text in almost real-time. It was at first implemented on a peer-to-peer architecture (see Figure 1). It is still very popular and is found on most platforms ranging from general purpose computers to mobile devices with limited functionality. It is the starting point for chat.

Chat is more recent and has incorporated IM. Chat is also a text-based near real-time communication protocol, but has the advantage of supporting multiple users simultaneously in the same conversation via the concept of group chat rooms. A room is a virtual place that more than one user can sign into and exchange ideas about the topic of the room. It was designed to take advantage of the client-server architecture. However, major chat providers run their services on peer-to-peer networks, such as Google Talk or Skype [10] [11].

The main technological difference between e-mail and IM or chat is the mechanism by which the messages are distributed. In the e-mail scheme, messages are stored on servers and users need to "ask" for messages in order to get them (by login or clicking on the inbox), while in IM or chat, messages are pushed to clients[2]. Today, IM is the core of chat technology because chat, which offers multi-user chat functionality, can also support instant messages between two users. For the purpose of this thesis, IM

[2] This is a general fact when two users are active. Chat has storage and delivery policies that use a push algorithm when a user is off-line.

means the ability to communicate between two users only and chat means the ability to communicate with more than one user in a “room” context.

Figure 1 and Figure 2 show the basic topology of a peer-to-peer network and a client-server architecture, respectively. In the peer-to-peer architecture, there is no single point of failure. However, the scalability is not as great as for a client-server network because as the network grows there is overhead associated with managing the network and some nodes must fill that duty. Those nodes are elevated to the role of coordinator and could become a point of failure if not maintained properly. Becoming a “super-node” is voluntary and maintaining the service relies on individual expertise. Those two factors might constrain the size of the network. In the client-server architecture, the single point of failure associated with the server can be overcome by introducing redundancy. A client-server architecture can grow to be very large by adding servers without destabilizing the infrastructure, because of its distributed quality [12]. XMPP, which will be discussed in the next section, “is not wedded to any specific network architecture, to date it usually has been implemented via a client-server architecture [2]”.

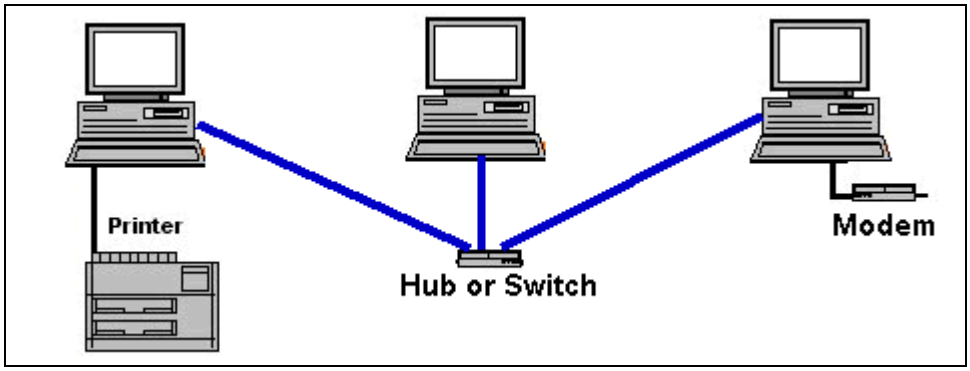


Figure 1. Simple peer-to-peer network. From [13]

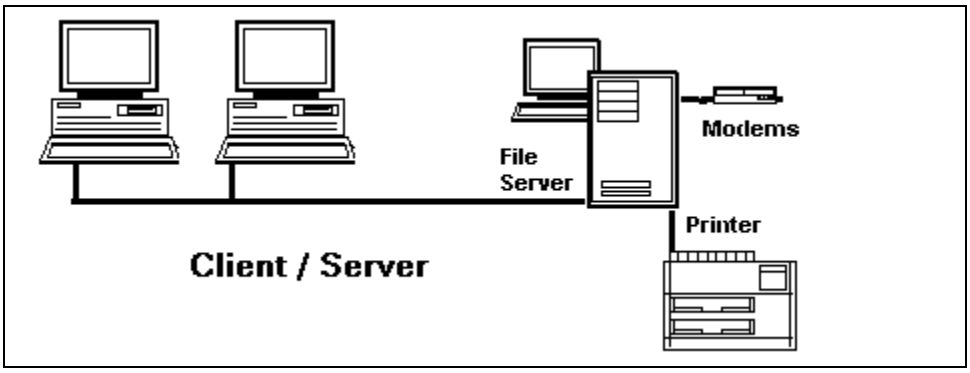
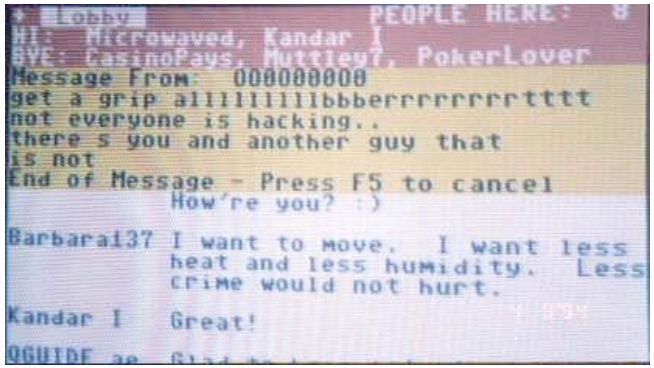


Figure 2. Simple Client/Server network. From [13]

The goals of chat along with the requirements for both servers and clients are described in the Internet Relay Chat (IRC) RFC 2810 through 2813. The IRC protocol emerged in order to allow for text-based conferencing, which came out of the Bulletin Board concept [14] [15]. IRC was invented to allow multiple users to communicate together via forums. The basic concepts are documented in a previous RFC numbered 1459 of the same name (IRC). RFC 2811 – *Internet Relay Chat: Channel Management* specifies how the characteristics and properties of channels, called forums, are managed by IRC servers [16]. RFC 2812 – *Internet Relay Chat: Client Protocol*, builds on the IRC architecture of the two previous RFCs and defines the client protocol [17]. RFC 2813 – *Internet Relay Chat: Server Protocol* describes the server protocol [18].

IM legacy systems such as AOL and others run proprietary software on dedicated servers hosted on hardware owned by service providers. Today, serious open source chat-related projects such as Jabber are gaining in popularity using XMPP, a subset of XML, to build their protocols. Table1 summarizes a time line of events that relate to IM, chat and XML.

Date	Events
Mid 1960s	IM is supported on multi-user OSes like Multics and CTSS [19].
1980s	Chat features are introduced to IM via the Bulletin Board System (BBS) concept [15]. Bulletin Board System software allowed a computer to use the phone system to dial (or Telnet) into a server in order to download programs and data, read news and communicate with other users connected to that same network. BBSs are the precursors of the way the Internet works today in that they brought multiple applications into one network (like gaming, exchanging files) and allowed for interaction between users of that network. However, it was localized phenomenon because it required users to dial into a computer in order to participate in the network. The cost of participating in remote networks was proportional to the buying power of users and their desire to spend lots of money on long distance phone calls [20].
Late 1980s through early 1990s	AOL (America On-Line) capitalizes on Quantum Link online service for Commodore 64 computers, which offered user-to-user messages with logged on users and called it On-Line Messages (OLM) [9].

	 Screen shot of a Quantum Link OLM
1988	Development of IRC (Internet Relay Chat)
Mid 1990s	Graphical User Interface (GUI)s similar to today's interfaces start appearing. In 1996 a company called Mirabilis, later bought by AOL, came out with ICQ, an instant messaging client, which became AOL Instant Messenger in 1997. AOL was given two patents for instant messaging technology. At that time, instantaneous communication became wide spread and parties such as Excite, Yahoo, MSN started offering their own version of instant messaging. Those protocols could not interact, which forced users to have multiple subscriptions if they wanted to participate in more than one network.
1993	Publication of RFC 1459 IRC (Internet Relay Chat Protocol). This document is a response to the multiple protocols problem described above. Implementations conformant to the core IRC protocol are interroperable.
February 1998	XML 1.0 became a W3C recommendation. Like HTML, XML has tags, but the main difference is that these tags can be user defined. XMPP defines a subset of XML tags. XML is still on version 1.1.

February 2000	Publication of RFC 2778 – *A model for Presence and Instant Messaging* and RFC 2779 – *Instant Messaging / Presence Protocol* [21] [22]. RFC 2778 defines the entities that participate in a presence and instant messaging model, their roles, the services provided and the terminology associated with the service. RFC 2779 –*Instant Messaging / Presence Protocol* (IMPP), has the goal of defining the minimum requirements for all IMPP, so that different implementations can "understand" each other and the protocol can become vendor-independent.
April 2000	Publication of RFC 2810 –*IRC: Architecture*, RFC 2811 –*IRC: Channel Management*, RFC 2812 – *IRC: Client Protocol* and RFC 2813 – *IRC: Server Protocol* (see description above and [14] [16] [17] [18]).
2000	Jabber is developed as an intermediary between other IM protocols. Because of the large number of vendor-specific protocols in the chat space, there was a need for software that was able to translate from one vendor to another. This was Jabber's first motivation as an open source project. Jabber uses XML.
October 2004	Publication of RFC 3920 –*XMPP: Core*, RFC 3921 – *XMPP: Instant Messaging and Presence*, and RFC 3923 –*End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol*. These documents will be discussed in detail in Section B. Extensible Messaging and Presence Protocol (XMPP) [2] [3] [23].

Table 1. Evolution of IM and Chat — Event Timeline

This project focuses on the XMPP and XMPP-IM protocols, which are based on RFC 2778 *–A Model for Presence and Instant Messaging* and RFC 2779 *–Instant Messaging / Presence Protocol* [21], [22]. RFC 2778 describes an abstract model, which includes the various presence and IM-aware entities involved, defines terminology and outlines the services that the system provides. The goal of that RFC is to lay a foundation so that implementers in the field are able to communicate with a set of common terms. The model presented in RFC 2778 does not discuss architectural issues, but instead outlines the fundamental components to support two services: a Presence Service and an Instant Messaging Service. A Presence Service allows users to advertise their status (offline, available, away, busy and so on) and an Instant Messaging Service supports the delivery of text messages between two users in near real-time.

In order to support Presence, a chat system must implement basic functionalities, such as namespace administration, common presence format, presence lookup and notification, presence caching and replication, and performance. In order to support Instant Messaging (IM), a chat system must implement basic functionalities such as common message format, reliability, performance and presence format. A detailed description of those requirements can be found in the RFCs mentioned above.

In addition to these functionality requirements, there are also security requirements such as access control, message authentication integrity and confidentiality. Access control refers to blocking unauthorized entities from viewing restricted information, as well as letting the legitimate subscribers view authorized information. Encryption is advocated to accomplish confidentiality and to detect tampering or error and replay type attacks in the context of network communication. Other security considerations, such as the amount of privilege that user as well as administrator roles should have, the importance of having the protocol detect spoofing which could lead to denial of service either for one particular user or for the entire system, are discussed in [22].

B. EXTENSIBLE MESSAGING AND PRESENCE PROTOCOL (XMPP)

XMPP is a protocol for streaming XML elements. “Streaming is a technique for transferring data such that it can be processed as a steady and continuous stream [24].” Streaming is well known with technologies such as audio or video because the technique allows for the browser to start displaying the information even though not all the bytes have arrived yet. Even though chat does not necessarily involve a browser, the streaming principle comes from the fact that there is one stream for an entire conversation. “An XML stream is a container for the exchange of XML elements between any two entities over a network [2].” An XML stream is represented by the tag stream symbolized by <stream> </stream> (the / represent the ending of the stream). A stream is the root level of communication. Streaming XML can be used in many contexts, including chat. In this context, XMPP streams XML stanzas to allow for the exchange of structured information between two entities in a network. Stanzas are defines in RFC 3920 as “a discrete semantic unit of structured information that is sent from one entity to another over an XML stream [2].” Stanzas are subsets of streams. XMPP meets the IM and Presence requirements of RFC 2779 [22] and provides an extended framework for exchanging XML data for chat. There are three main documents that describe the protocol; RFC 3920: XMPP –*Core* [2], RFC 3921: XMPP –*Instant Messaging and Presence* [3] and RFC 3923 –*End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol* [23]. The protocols defined in these RFCs can be applied to different network architectures. However, to date, XMPP has been commonly used in the client-server architecture and thus the RFCs are framed with in the context of that architecture. XMPP defines three functional entities: servers, clients and gateways. Gateways translate messages between chat services that implement XMPP and those that do not use XMPP. The standard for gateways is included in RFC 3922 –*Mapping XMPP to the IETF's CPIM specifications* [25], which defines the mapping between XMPP and the Common Presence and Instant Messaging (CPIM) protocol.

In order for entities to communicate, there is a need for unique identifiers. For historical reasons[3], the address of an XMPP entity is called a Jabber Identifier (JID). Those addresses are formed based on a domain identifier, a node identifier and a resource identifier, respectively (node@domain/resource).

XMPP only binds to TCP connections. “In the context of client-to-server communications, a server must allow a client to share a single TCP connection for XML stanzas sent from client-to-server and from server-to-client. In the context of server-to-server communications, a server must use one TCP connection for XML stanzas sent from the server to the peer and another TCP connection for stanzas from the peer to the server, for a total of two TCP connections [2]”. A stream will stay open until either an error occurs or the client tears the session down.

XMPP only uses a subset of the XML vocabulary. The stanzas for chat are XML fragments that are associated with one of the following tags: <message>, <presence> and <iq>. There could be other tags/elements inside each stanzas. All of those tags have attributes and some of those attributes are shared among them.

The responsibilities of XMPP servers and clients are given below.

1. XMPP Server Responsibilities.

The server has the following responsibilities:

- Managing connections (client-to-server –port 5222 and legacy port 5223– and server-to-server –port 5269),
- Routing XML stanzas among entities over XML streams,
- Tracking resource binding—resource binding is the action of associating a resource with a stream and a user ID. This is optional, but highly encouraged and can be required by either the server or the client.

[3] XMPP was formulated by the same people who are managing the Jabber Software Foundation. So “J” stands for Jabber.

- Support for Transport Layer Security[4] (TLS) for encrypting XML streams [2]. This protocol protects against eavesdropping, tampering and message forgery.
- Support for Simple Authentication and Security Layer[5] (SASL) for authentication of XML streams [2].
- Support for Unicode normalization and string normalization of addresses, and XML stream via the application of Nameprep[6] and Stringprep[7].
- Support for basic stanza semantics of the basic elements <message>, <presence> and <iq>,
- Generating errors related to streams, SASL, TLS, and XML stanzas.

2. XMPP Client Responsibilities

The client connects directly to a server via port 5222 over a TCP connection and uses XMPP to take advantage of the services that are available on the server. In addition, the client must support:

- XML streams including the use of TLS and SASL [2],
- Basic stanza semantics of <message/>, <presence/> and <iq/>,
- Generation of errors related to streams, SASL, TLS, and XML stanzas.

3. Security Issues

XMPP addresses confidentiality and integrity with TLS and addresses mutual authentication with SASL (between client-server and server-server). Even though RFC

[4] TLS is the successor to Secure Socket Layer (SSL). It provides secure communications in the Internet through cryptography.

[5] The SASL protocol is represented by a series of challenges and responses that the entities send each other. SASL is usually used with TLS as they complement each other to address basic communication security issues in the Internet.

[6] Nameprep is the process of Unicode normalization. It allows for case-folding to lowercase and removal of some generally invisible code points before it is suitable to represent a domain. Nameprep is defined in RFC 3491.

[7] Stringprep is uses to normalize a string before it is put on the wire. Stringprep is defined in RFC 3454.

3920 requires the implementation of both the SASL and TLS mechanisms to satisfy the security demands of RFC 2779 –*Instant Messaging / Presence Protocol Requirements*, RFC 3923 –*End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP)* discusses additional mechanisms that can be used to meet the security requirements prescribed in RFC 2779. The first mechanism is end-to-end signing and the second is object encryption [23]. Both the signing and the encryption are done using Secure/Multipurpose Internet Mail Extensions (S/MIME)[8]. Using S/MIME improves the confidentiality of messages as it encrypts the entire message including the header information. However, since S/MIME requires public keys, deleted or out–of-date certificates can cause problems with decryption of messages [23]. RFC 3923 requires that the following cryptographic algorithms be supported for an implementation to be compliant:

- SHA-1
- RSA with the SHA-1 signature algorithm
- RSA key transport
- AES-128 encryption algorithm in CDB mode [23]

RFC 3920 contains advices regarding the order in which the protocol should be initiated. For example, "if the TLS negotiation occurs between two servers, communications MUST NOT proceed until the DNS hostnames … have been resolved" or "if the initiating entity chooses to use TLS, TLS negotiation MUST be completed before proceeding to SASL negotiation." [2] Strong implementation of this advice can minimize the risks of malicious message injections.

C. JABBER

Jabber is an open-source XML streaming protocol, and is a precursor to the XMPP standard. Jabber and XMPP are often confused because the same non-profit organization, namely the Jabber Foundation, is behind both the XMPP standards (RFCs

[8] Secure/Multipurpose Internet Mail Extensions (S/MIME) uses certificates, so a certificate authority is assumed here.

3920 and 3921) and the jabberd14 and jabberd2 projects. The distinction between Jabber and XMPP is that XMPP is a standard and Jabber an implementation of that standard[9]. There are other projects (commercial and open-source) that adhere to the XMPP standard (Chapter IV shows some open-source projects). In addition to the core functionalities based on the XMPP standard, the Jabber foundation is continually working on extensions [27]. Extensions are not included in any RFCs, but follow the basic logic of those documents and are written in the same format/language, giving no implementation details. Those extension documents are the basis for different implementations in the open-source community. Extensions allow both clients and servers to gain functionality, and improve usability and security.

D. MONTEREY SECURITY ARCHITECTURE (MYSEA)

"MYSEA is an innovative architecture to provide trusted security services and integrated operating system mechanisms that can protect distributed multi-domain computing environments from malicious code and other attacks [28]." This architecture is composed of COTS and of high assurance software. The COTS components allow users to run productivity applications while core elements of the architecture are high assurance and enforce policies (Bell-LaPadula and Biba [29] [30]). "The purpose of MYSEA is to provide a trusted distributed operating environment for enforcing multi-domain security policies, which supports unmodified COTS productivity applications [28]."

Figure 3 shows the MYSEA architecture. Users on the Multilevel Security (MLS) network can log in at their classification level via a Trusted Path Extension device (TPE). The MYSEA server supports SMTP, IMAP and HTTP. Based on this thesis and some additional porting steps, chat services could be implemented on MYSEA. MYSEA also uses IPsec to establish protected channels at different session levels on the MLS network.

Part of the high assurance core that resides on MYSEA and makes it a MLS system is called the Secure Session Services (SSS). The purpose of this high assurance

[9] Note that the use/invention of XMPP is anterior to the creation/acceptance of XMPP as a standard. Depending on the version of the software, Jabber can be considered more or less compliant to the XMPP protocol. The differences between core "Jabber protocols and XMPP" can be found in RFC 3920 Appendix D [2].

code is to listen on the different ports (services which MYSEA supports) and then spawn processes executing different applications at the right classification level. (These spawned applications are not trusted.) SSS is a trusted process that is functionally equivalent to the inetd daemon that manages network services on Unix/Linux systems. The ability to run an application as an inetd-spawned daemon is a critical MYSEA porting requirement.

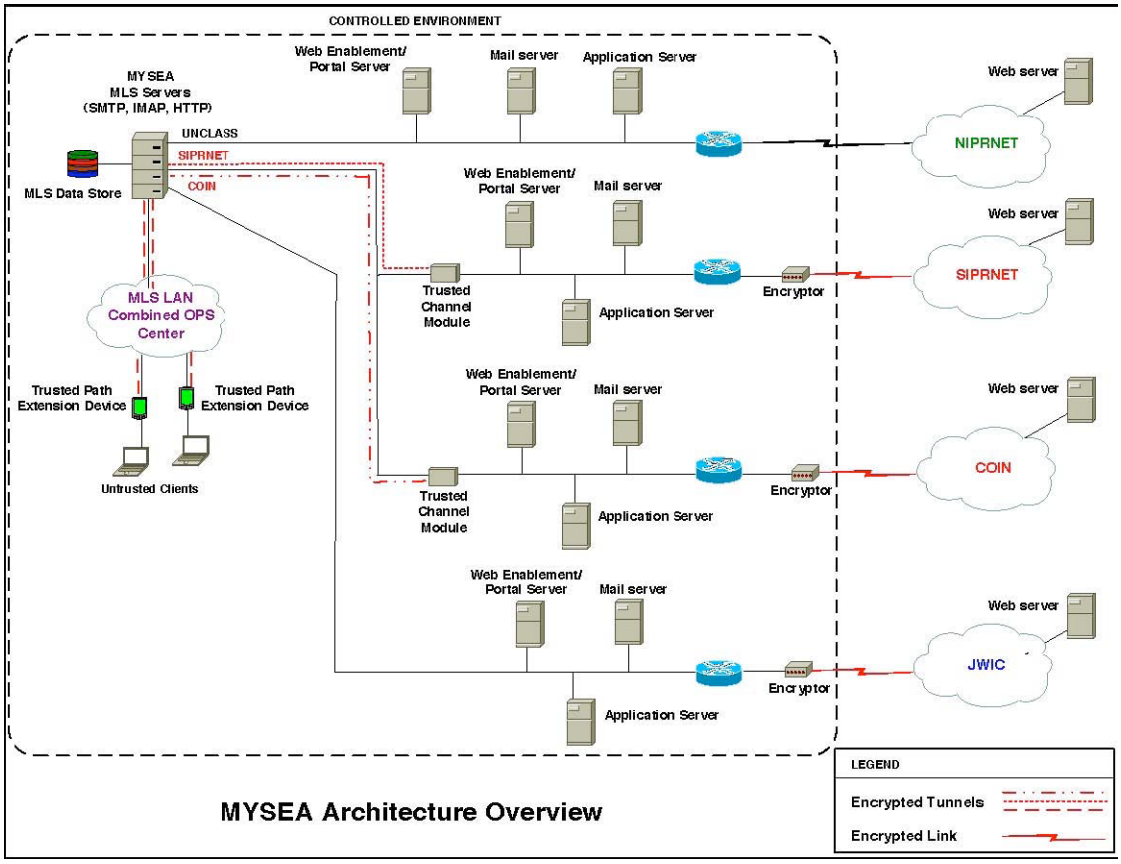


Figure 3. MYSEA Architecture Overview. From [28]

E. XTS-400 ARCHITECTURE

MYSEA runs on a platform called an XTS-400 and made by BAE [31]. The XTS-400 is a multilevel secure computer system [32]. The XTS-400 is the result of combining an Intel x86 for the hardware and Secure Trusted Operating Program (STOP) for the operating system. "The XTS-400 architecture can host, and be trusted to separate multiple, concurrent data sets, users, and networks at different sensitivity levels [33]." This architecture provides

both a trusted and non-trusted environment. The trusted environment is for performing administrative tasks and executing privileged applications. The non-trusted environment supports both custom and binary-compatible Unix/Linux applications. The interface to the system is a command-line Unix/Linux like environment. Even though the XTS-400 can be used in many different ways—for example, as a workstation—MYSEA uses it as a MLS server.

Currently MYSEA services run on STOP 6.3, but these will migrate to STOP 7. In addition to the traditional discretionary access control (DAC) and mandatory access control (MAC), STOP 7 also provides Role-Based Access Control (RBAC) [32]. RBAC does not assign permission to users directly. Users get permission based on the role that they are assigned. This makes policy management easier and allows access controls to be tailored to an organization's needs. The use of RBAC to control users is different than with a traditional Access Control List (ACL) mechanism because it uses groups. Grouping users allows for easier administrative management as users do not have to be granted permissions individually, however, it might be less granular. The RBAC does not invalidate the Bell-LaPadula [29] multilevel security policy and the Biba [30] multilevel integrity policy [32]. Because of its rating, and accreditation, the STOP OS is often used for guard technology.

F. SUMMARY

This chapter provided an overview of Chat, XMPP and Jabber. It also clarified some definitions between terms and provided a historical time line for the technologies. In addition, this chapter summarized the purpose and architecture of the MYSEA testbed and highlighted the important security mechanisms that pertain to this project of the XTS-400. The next chapter presents the selection process of the XML-based chat server needed to continue this study.

III. CHAT ENGINES SELECTION

For this project an XMPP server that can be ported to run on the MYSEA server must be chosen. The selected server must meet a set of MYSEA-specific requirements. This chapter first gives an overview of the available open source chat servers and the XMPP extensions that might be suitable to this project. Then, it prioritizes the selection criteria and summarizes all of the servers that were considered. Finally it presents the selected XMPP server and the rational for its selection.

A. CHAT SERVERS

The open source XMPP community is centered around jabber.org [34] and xmpp.org [35], both of which promote XMPP as the language for chat and Instant Messaging (IM). Each organization has a website. The jabber.org site lists most of the servers with open source-like licenses available today. Most of those implementations accommodate a broad range of host platforms including Windows, Linux, Macintosh, Solaris, HP-UX, AIX and FreeBSD. The servers are written in a variety of programming languages ranging from C to Erlang, in addition to Java, C++ and Perl. The servers adhere to the XMPP standards expressed in the different RFCs mentioned in the background chapter, but implement different optional functionalities, called extensions.

The RFC compliant core functionalities are insufficient to make a server competitive in today's market. This is the reason for the many extensions developed by the XSF (XMPP Standards Foundation). The XSF "develops extensions to XMPP through a standards process centered around XMPP Extension Protocols (XEPs) [27]." The extensions go through a review and approval process by the XMPP Council "and modification based on implementation experience and interoperability testing" is also part of the process of developing new extensions [27]. The XEPs listed in Appendix A are in some stage within the reviewing/approval process. Most of the extensions have been assimilated as part of the core code, but not all. An example of an extension that is external to the core server and that must be installed separately is the Multi-User Chat (XEP-0045). This extension gives the ability to the server to support group chat rooms.

1. Selection Criteria for the Server

The criteria that were considered in order to choose a jabber server to port to MYSEA are listed below. Since all requirements are not imperative, there is a list of "Must have" requirements and a list of "Nice to have" requirements. The requirements are ordered by importance in their respective lists, with the most important first. A brief explanation follows each requirement.

"Must have" requirements:

- **Open source license.** A license that lets anyone copy, use and modify the code of the server and library dependencies freely.
- **inetd capability**. MYSEA does not support standalone daemon services and requires all applications to run as inetd processes. This is because ported applications generally have low integrity and should only run as single-level processes (i.e., the session level of the user). In MYSEA, the inetd capability is handled by a number of SSS processes, each of which listens to a specific port. For XMPP, the SSS process designated to handle XMPP traffic would listen on port 5222 (client-to-server socket) and would spawn chat processes at the classification level of the users' session.
- **POSIX threads**. STOP OS 6.x does not support pthreads library and thus MYSEA can only support non-threaded applications at this time. This condition will change when MYSEA is ported to STOP OS 7, which supports pthreads.
- **Written in C or C++**. C is native to MYSEA. Having the code of the server written in C or C++ will facilitate the porting not only because necessary compilers are already available, but because there would be no need to study the security issues associated with using another language that is not currently implemented.

- **Flat file storage**. The server should not require a database in order to operate. At the present, MYSEA does not support any databases. A server that requires a database for information storage would entail an extra step in the porting process.
- **Static IP**. The server needs to be able to operate with a static IP address in the MYSEA environment.
- **Code size.** The main exercise for the porting is to modify the code in order to comply with what STOP OS 6.x/MYSEA supports. If the code is small, the code review and the modification to the code will be more manageable and easier to test.

"Nice to have" requirements:

- **RFC compliant.** The XMPP protocol is documented in RFC 3920 and 3921, but it is up to each chat server team to implement the requirements. The result might be more or less compliant with the specification standard.
- **Maturity**. A large user population and following of administrators is likely to result in richer resources (mailing list, forums) and better documentation. It also implies a better tested code base.
- **Version control**. The use of versioning software such as Version Control System (CVS) or Subversion (SVN) will allow for systematic code maintenance.
- **IPv6**. Although the current MYSEA does not support IPv6, a future MYSEA version will include that capability.
- **Feature full**. The ability to support a fair number of XEPs will guarantee that the server is keeping up with the main stream chat servers and can support features that modern clients implement.

- **Code Modularity.** The porting effort would be easier if the code is modular, i.e., composed of logical parts/modules. Modular decomposition also facilitates better analysis of architectural properties such as layering, cohesion, and coupling.

Table 2 and Table 3 identify the servers examined for this project. In Table 2, the columns list the name of the projects (only open source projects were considered for this porting effort) and the rows list the requirements (mixing the “Must have” with the “Nice to have” in the order of importance). An “X” indicates that the server fulfills that requirement and a blank indicate that either the requirement is not met or that further code analysis is necessary to determine the ability of the server to meet the particular requirement. The last row summarizes the number of requirements fulfilled by each server. Table 3 lists all of the features supported by the different servers. Some of those features are part of the core and some require separate software to be installed. The XEPs are in numerical order, and are not ordered by importance. An “X” indicates that the server either already has incorporated the extension in its code or can support that additional functionality, as in the case of MUC or JUD. More details regarding each XEP listed in Table 3 can be found in Appendix A and a full implementation/technical description can be found on the web [27].

It is important to note that some of the extensions depend on other extensions that may not be listed in the Table 3 (see Chapter V). Also, some of the requirements for this project might not be described in the documentation and can only be discovered as the code is installed, analyzed and executed.

Requirements	**Ejabberd** [36]	**jabberd14** [37]	**jabberd2** [38]	**Openfire** [39]	**Tigase** [40]
MUST HAVE REQUIREMENTS					
Open source	X	X	X	X	X
Inetd					
POSIX thread	X	X	X	X	X
Written in C/C++		X	X		
Flat file storage	X	X			X
Static IP	X	X	X	X	X
Code size					
NICE TO HAVE REQUIREMENTS					
Authentication	X	X	X	X	X
Encryption	X	X	X	X	X
RFC compliant	X	X	X	X	X
Maturity	X	X	X	X	X
Version control	X	X	X		
IPv6	X	X	X		
Code Modularity	X	X	X		
TOTAL	**11**	**12**	**11**	**7**	**8**

Table 2. Requirements for the server

XEP number	Ejabberd	jabberd14	jabberd2	Openfire	Tigase
XEP-0004	X			X	X
XEP-0012	X	X	X	X	X
XEP-0013		X		X	
XEP-0016	X	X	X	X	X
XEP-0020				X	
XEP-0030	X	X	X	X	X
XEP-0033		X		X	
XEP-0045	X	X		X	X
XEP-0047					X
XEP-0048		X		X	X
XEP-0049	X	X	X	X	X
XEP-0050	X	X		X	X
XEP-0054	X	X	X	X	X
XEP-0059				X	
XEP-0060	X			X	
XEP-0065	X			X	
XEP-0066					X
XEP-0077	X		X	X	X
XEP-0079			X		
XEP-0085					X
XEP-0092	X	X	X	X	X
XEP-0095					X
XEP-0096				X	X
XEP-0100				X	
XEP-0106				X	
XEP-0114	X	X	X	X	X
XEP-0115	X			X	
XEP-0124				X	X
XEP-0138	X		X	X	
XEP-0145	X	X			
XEP-0153	X	X	X	X	X
XEP-0163	X			X	
XEP-0191			X		
XEP-0199		X	X		X
XEP-0202			X	X	
XEP-0203			X	X	
XEP-0206	X			X	X
TOTAL	**19**	**15**	**15**	**29**	**21**

Table 3. XEP by server

2. Outcome

Based on the requirements score in Table 2, jabberd14 was chosen. However, it is important to note that the *inetd* requirement was not met. The inetd requirement represents a challenge for MYSEA as newer applications are built with standalone daemons for performance reasons [41].

B. CHAT CLIENTS

The open source XMPP clients were chosen based on their compatibility both with the platforms on which they were to be installed and with the different servers that were implemented and tested (see Chapter IV).

1. Selection Criteria for the Clients

XMPP clients do not need to be very sophisticated. As long as they can perform the basic chat functionalities reliably – registering with the chat server, participating in instant messaging and creating and configuring group chat—, they can be considered candidates. However, below are some criteria that should be considered before choosing an XMPP-aware client for deployment in the MYSEA environment. The requirements are arranged by order of importance.

- **Open source license.** A license that lets anyone copy, use and modify the code of the client.
- **Multi-platform**. The client should be available on more than one platform so that users are able to use a variety of operating systems.
- **Authentication capability**. Mutual application-level authentication is highly desirable between the clients and servers and would complement the user authentication provided in MYSEA for user login and session level negotiation.
- **Encryption capability**. Encryption is highly recommended for communication over a network and would complement communications protection already supported in the MYSEA environment.

- **Multi-language**. In a coalition use case, the server might support users from other nations with other languages besides English.
- **Maturity**. A mature project implies a popular product with well-maintained code, documentation and available customer support.
- **Feature full**. Using a server and clients that can support approved XMPP extensions will improve the user experience and may augment application-level security.

For more information on open source client projects, a list is available on the web at [34].

2. Outcome

Although many chat clients meet the requirements described above, PSI [42], Gajim [43] and Pidgin [44] were used for the different testing described in Chapter IV.

C. SUMMARY

This chapter summarized the porting requirements used to select the jabberd14 server for use in MYSEA. The XMPP extensions supported by different servers are also addressed. Most modern clients support all of the features that the jabberd14 server supports. The three clients chosen fulfill all of the requirements mentioned in the selection criteria of this chapter (sub-section B-1). The next chapter describes the experimentation with the jabberd14 server on Linux and XTS platforms. It also includes the test plan and a summary of the problems encountered.

IV. EXPERIMENTATION

The process to port an application to MYSEA involves two preliminary steps. First, install and test the application on a RedHat 8 system first, and then on a standalone XTS-400 system. If these two steps are successful, any problems encountered subsequently are MYSEA specific and can be addressed more easily. This thesis follows this methodology. The first platform on which the Jabber server was installed and tested is the RedHat 8 operating system (OS). This platform was used because MYSEA currently runs on the STOP 6.3 OS which supports the RedHat 8 APIs. Since, MYSEA is scheduled to migrate to STOP OS 7, the jabberd14 server was also tested on Fedora core 7 OS and STOP 7 OS, which supports Linux 2.6 APIs.

The first section of this chapter lists the functionalities of the jabberd14-1.4.x server and of the jabberd14-1.6.x server. The second section discusses the testing plan and procedures. The remaining sections discuss each installation in detail and, for selected sections, the test results are also included.

A. FUNCTIONALITIES OF THE JABBERD14 SERVER

The jabberd14 server adheres to [2], [3] standards and to a number of XEP standards [27]. For this thesis, experiments were conducted on two different versions of jabberd14, version 1.4.x and version 1.6.x, whose functionalities are described below.

The following functionalities are supported by jabberd 1.4.x as described in the *Jabberd 1.4.x Administration Guide* (an installation guide) [45]:

- Accept TCP socket connections from compatible clients and server-side components
- Manage XML streams to and from clients and components
- Deliver the core Jabber data types to authorized clients and components

- Maintain session information for connected clients (usually IM users)
- If necessary, open connections to and validates connections from other Jabber server, then routes data to them.
- Store information on behalf of components and especially IM users, including each user's contact list and some client preferences [45].

Some of the functionalities listed above implement the core requirements dictated by the XMPP standards [2] [3] and some implement extensions (XEP). Some of the extensions are transparent, i.e., they are fused into the jabberd14-1.4.x code, and some extensions require external modules. Jabberd14-1.4.x can support JUD (Jabber User Directory) and MUC (Multi-User Chat), as external modules.

As the server evolved from version 1.4.x to 1.6.x, functionalities were added to the core by incorporating new XEP extensions and replacing depreciated extensions. Below is a list of the added functionalities available in jabberd14 version 1.6.x as described in [46] that are relevant to the port to MYSEA[10]:

- Improved compliance to RFC 3920 and RFC 3921 with the implementation of jadc2s client connection manager,
- Support for privacy lists (XEP-0016),
- Support for Dutch, English, French, German, Hungarian and Italian and allows for support of additional languages files,
- SASL authentication is possible on client links as well as on inter-server links thanks to the jadc2s connection manager. Client authentication can support CRAM-MD5, PLAIN, GSSAPI, DIGEST-MD5, NTLM, SRP, OTP, KERBEROS_V4. EXTERNAL is used on inter-server links,

[10] Innovations that concern database implementation or other aspects of the server that are not used in this project are not mentioned here. A full list of the functionality of the jabberd14-1.6.x can be found at http://jabberd.org/news/.

- Support for Flexible Offline Message Retrieval (XEP-0013),
- Support for XMPP Ping (XEP-0199),
- Support for a full namespace,
- Support for the xml:lang,
- Support for base_dir as a handler to the directory of *.stanza files. The files can be read and parsed as stanzas. The processed data can be used as inject messages to the server (scripts)[11],
- Passwords are no longer cached in memory by the server,
- Possible to block account names from being registered and enforce minimum and maximum lengths of the username for new accounts,
- Can block re-registration of a user ID (JID) for a configurable amount of time (defaults to half a year),
- Possible to migrate from old filespools to newer storage handlers by reconfiguring the server and then importing the old data (using the I command line option),
- All components of the server, except the client connection manager, can be restarted without users' sessions being dropped. This allows reconfiguration and software upgrades while the service is running,
- Session manager understands the internal session protocol of the jabberd2 project. This allows development and use of components acting as client managers, for both server implementations at the same time,
- Uses libpopt for command line parsing,
- Detect and ignore stale pid files,
- Service discovery replaces browsing and is generated automatically,

[11] There might be a potential security issue here.

- jabber:iq:admin, jabber:iq:filter namespaces and mod_groups module removed,
- Support for the jabber:iq:agent and jabber:iq:agents is disabled [46].

B. TESTING OVERVIEW

Testing constitutes a fundamental part of the experimentation exercise. The testing was done on all three platforms using the same clients and the same network topology as illustrated in Figure 4.

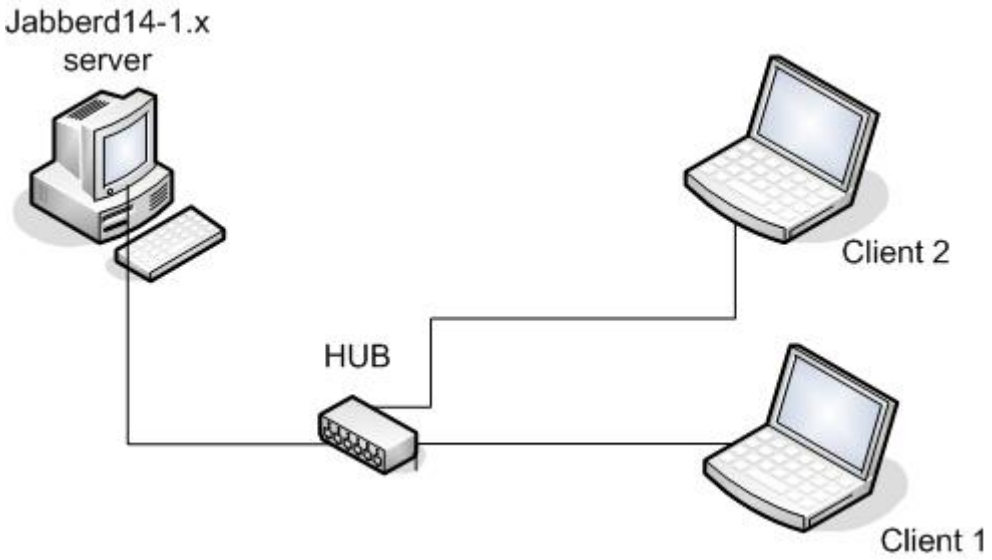


Figure 4. Network Topology of Test Setup

Client 1 and Client 2, represented by the laptops, use the open-source chat client, Gajim, to connect to the server. Installation instructions for the Gajim client can be found in Appendix E. The reason why the Gajim client was chosen is because it is recommended in Chapter III, works with all of the servers that had to be tested and its web site precisely documents the XEPs that the client supports [43][12]. The latter reason is important because to test the functionality of the server, the client also has to implement

[12] The information is on the Wiki at http://trac.gajim.org/wiki/GajimFaq#which-xeps-does-gajim-support

those functionalities. The jabberd14 server, represented by a desktop in Figure 5, runs either in a virtual machine (VM) or on the XTS-400 machine.

1. Test Plan

A test plan is required to verify that the servers behave as expected when given basic tasks to perform. The tests must verify that both the core modules and the MUC and the JUD modules function according to their respective specifications. Two kinds of tests were performed; functional tests and exception tests. Functional tests are those that test if the server correctly performs a task that it claims it can do, like letting a currently registered user connect to the server with a valid JID. Exception tests are those that test if the server will perform a task that it should not do, like registering a user with an invalid JID. This test plan includes five categories from which useful functionality tests were selected. The five categories are as follows:

A. Account management

B. Presence management

C. Instant Messaging (IM) management

D. Multi-User Chat (MUC) management

E. Service discovery and Jabber User Directory (JUD) management.

The tests are described in Table 4 through Table 8 (one for each category) and are organized as follows. The first column represents a reference test number, which is used in Appendix F. The “X” in the second column indicates that the test is considered an exception test. The “Test Description” column describes the functionality to be tested and the “Test Objective” column states the goal of the test in relation to the server. Most of the tests require at least two separate machines with the client installed. On each of these machines, one valid account needs to be created. Appendix F describes the assumptions and the environment required for each test. The MUC section could have

included more tests because there are many settings that can be made in the configuration window of the group chat room, but the focus of these functional tests is security and privacy.

It is important to note that the jabberd14 servers were tested with their default configuration settings—the configuration file was not changed except for items required to function (e.g., name of the server, IP address, patches, flat file storage configuration). Some of the tests represent compliance tests. These tests involve the fundamental requirements of the XMPP standard and do not depend on a configuration setting. An example of compliance testing is the registration with a valid or invalid JID. In the configuration file, there is no option for the administrator to allow for registration with an invalid JID. However, some tests depend on configurable options, for example allowing registration on the legacy port 5223. In order for the server to allow users on the 5223 port, the configuration file can be changed. The criterion for deciding if a test case is an exception or a functional test depends on the default configuration. In other words, if the configuration file needs to be changed beyond the standard settings, then that test case is considered an exception test. Exception tests are expected to fail.

Table 4. Account management

Test #	Exception Test	Test Description	Test Objective
A1		Register account with a valid JID	Ensures that server can register accounts
A2	X	Register account with an invalid JID	Ensures that server does not register users with an invalid JID
A3		Register account with a password	Ensures that server enforces registering with a password
A4	X	Register account without a password	Ensures that server does not allow registration without a password
A5	X	Connect to server using SSL (legacy on port 5223)	Ensures that the server does not connect to a client on legacy port 5223
A6		Connect to server without SSL (port 5222)	Ensures that server connects to a client on the reserved port 5222
A7	X	Connect to server with SSL (port 5222)	Ensures that server does not connect to a client on the reserved

			port with SSL
A8		Connect with a password	Ensures that server enforces the use of passwords
A9	X	Connect without a password	Ensures that server enforces the use of passwords
A10		Edit personal information	Ensures that server performs resource binding
A11		Add a contact that has a valid JID	Ensures that server lets users add contacts and displays them
A12	X	Add a contact that has an invalid JID	Ensures that server checks for the JID validity before adding contacts
A13		Delete contacts (buddies) from the contact list	Ensures that server updates the buddy list
A14		Forbid contacts to see status	Ensures that server applies the right restriction to the right contacts
A15		Delete an account on client only	Ensures that server is not deleting information when deletion is not requested
A16		Delete an account on both the client and the server	Ensures that server is not retaining information about deleted account
A17		Connect to the server with a deleted account after test A15 is performed	Ensures that server does not purge its records
A18	X	Connect to the server with a deleted account after test A16 is performed	Ensures that server does not allow deleted users to connect
A19	X	Send server message to all users connected to the server	Ensures that server does not allow broadcasting to all registered users
A20		Retrieve other users' contact information	Ensures that the server keeps information on its users
A21		Transfer files between clients	Ensures that file transfer functionality is available
A22		Retrieve communication history information	Ensures that server is storing records
A23		Deny a contact from viewing status for the current session	Ensures that privacy rules regarding user status can be enforced by the server for the current session
A24		Deny a contact from sending a message for the current session	Ensures that privacy rules regarding blocking messages from user can be enforced by the server for the current session
A25	X	Register with an account that was deleted from both the	Ensures that the server does not allow a user to re-register with the

		server and the client	same JID after the account was deleted on the client as well as on the server
A26	X	Register with an account that was deleted from the client only	Ensures that the server does not allow a user to re-register with the same JID after the account was deleted on the client only

Table 5. Presence management

Test #	Exception Test	Test Description	Test Objective
B1		Change presence to Available	Validates that users can become active after having been in a different state
B2		Change presence to Free for Chat, Away, Not Available, Busy, Invisible and Offline	Validates that presence information can be changed and is reflected to users who subscribe to the presence
B3		Display offline contacts	Ensures that server can display correct presence for contacts

Table 6. Instant Messaging (IM) management

Test #	Exception Test	Test Description	Test Objective
C1		Start a chat with someone registered with the server and is on the contact list	Ensures that server can retrieve information from users who are on the user's contact list
C2		Start a chat with someone registered with the server but is not in the contact list	Ensures that server can retrieve information from users who do not subscribe to the user's roster
C3	X	Start a chat with someone by mistyping their JID (user ID)	Ensures that server uses Stringprep
C4	X	Start a chat with someone by mistyping their JID (server name)	Ensures that server checks for JID validity
C5		Have multiple chat sessions with different valid contacts	Ensures that the server does not mix chat sessions in a one to many paradigm
C6		Have multiple chat sessions with the same valid contact	Ensures that server does not mix chat sessions in a one to one paradigm
C7		Add Emoticons to chat messages	Ensures that server can transmit special characters

Table 7. Multi-User Chat (MUC) management

Test #	Exception Test	Test Description	Test Objective
D1		Create a default group chat with a valid JID	Ensures that server implements MUC
D2	X	Create a default group chat with an invalid JID	Ensures that server enforces valid JID
D3		Join a private group chat with proper credentials (password required)	Ensures that server can enforce that the moderator[13] can set rules for joining password-protected group chat rooms
D4	X	Join a private group chat without membership (member list enforced)	Ensures that server can enforce that the moderator can set rules for joining membership-protected group chat rooms
D5	X	Join a private group chat without the necessary credentials (password required)	Ensures that server can enforce group chat rules set by moderator
D6		Join a publicly open group chat	Ensures that server allows any user to join public a group chat room
D7		Configure a group chat room after it is created (as its creator)	Ensures that the configuration options of a group chat room are available to the moderator/ of the room
D8	X	Configure a room after it is created as a normal[14] user	Ensures that the configuration options of group chat rooms are not available to normal users
D9		Inviting contacts to join private group chat rooms	Ensures that server can enforce MUC settings
D10		Communicating in authorized group chat rooms	Ensures that server keeps the flow of messages in the proper order and label their authorship properly
D11		Ban a user from a group chat room when having the role of moderator	Ensures that the server manages configuration options of group chat rooms properly
D12	X	Ban a user from a group chat room when not having the role of moderator	Ensures that the server makes the distinction between the moderator and normal users

[13] The Moderator is the creator of the group chat room

[14] A normal user is not the creator of a group chat room

D13	X	Joining the group chat room after been banned	Ensures that banning a user from a group chat is a permanent/persistent setting in the configuration file.
D14		Kick (temporarily remove) a user from a group chat room when having the role of moderator	Ensures that the moderator of a group chat room can kick users
D15		Joining a group chat room after been kicked	Ensures that kicking is only on a session basis

Table 8. Service discovery and Jabber User Directory (JUD) management

Test #	Exception test	Test Description	Test Objective
E1		List the rooms available and the number of the participants in each room	Ensures that server implements service discovery (XEP-0030)
E2		Access the user directory	Ensures that the server implements JUD

The following sections explain in detail the testing performed on each of the test platforms.

C. REDHAT 8 PLATFORM

This installation and testing represents the first step towards porting the jabberd14 server to MYSEA. The XTS-400 with STOP 6.3 is closest to the RedHat 8 OS, so this step is intended to demonstrate that the jabberd14 server can indeed run on RedHat 8.

This experiment installed the software package published by Van Emery in 2003 [47] that integrates the jabberd14-1.4.2, the jud-0.5 and the muc-conference-0.5.2 source code together as an installable package. The installation was made in a virtual machine (VM) with a bridge Ethernet connection. This server is configured for a small user population – about a hundred users— that resides on a private intranet and does not support dynamic registration.

1. Configuration Issues with the Server

Jabber14-1.4.2 implements all the core functionalities mentioned in Section A of this chapter in addition to supporting extension 0045 (XEP-0045) and the obsolete extension 0094 (XEP-0094). These extensions are implemented by two external modules MUC (Multi-User Chat) version 0.5.2 and JUD (Jabber User Directory) version 0.5. For this test, this server was configured to disallow client dynamic registration and client dynamic group chat room creation. These settings were done by the author of the software package because of the small user base that the server would have to support and the control that the administrator wanted to keep [47][15]. Not having dynamic registration means that users would contact the server's administrator to get an account and have their name as the group chat room administrator in the configuration file in order to create group chat rooms (see Figure 6).

2. Installation of Jabber14-1.4.2 Server

Because dependency issues were taken care of by the distribution package, it was a straight forward installation. The jabberd14-1.4.2 server was first installed based on the instructions in [47]. Appendix B contains the installation instructions and the missing steps from the original instructions. The distribution also includes scripts to automate the following administrative tasks:

- Creating user accounts,
- Checking information about user accounts created on the server,
- Generating statistics about the server:
    1. the number of connections that are present and encrypted, or non-encrypted,
    2. the number of successful logins for the day,
    3. the number of failure logins for the day,
    4. the number of connections to other servers,

[15] Assumptions section

- Backing up the configuration files, user accounts and other spool files.

The deletion of user accounts is as simple as deleting the user.xml[16] file from the spool directory of the server.

There is no script to add group chat because the changes need to be made in the jabberd.xml configuration file itself as follows:

```
<service id='conference.im.mysea.edu'>
    <load>
      <conference>./mu-conference-0.5.2/src/mu-
conference.so</conference>
    </load>
    <conference xmlns="jabber:config:conference">
      <roomlock/>
      <persistent/>
      <public/>
      <vCard>
        <FN>testroom</FN>
        <FN>Public Chatrooms</FN>
        <DESC>Public Chatrooms at mysea.edu</DESC>
        <URL>http://conference.im.mysea.edu/</URL>
      </vCard>
      <history>40</history>
      <logdir>/var/log/jabberd/</logdir>
      <sadmin>
        <user>phil@im.mysea.edu</user>
      </sadmin>
      <notice>
        <join>has joined</join>
        <leave>has left</leave>
        <rename>is now known as</rename>
      </notice>
    </conference>
  </service>
```

<roomlock/> indicates that group chat rooms cannot be created dynamically.

In this case, the user called *phil* is the only user that has the privilege to create group chat room

Figure 5. Configuration settings for group chat rooms

Figure 6 shows that the configuration settings for group chat rooms must be specified within the <service/> tag. The <roomlock/> tag means that non-administrator users cannot create group chat rooms and the <persistent/> tag means that the rooms are permanent (cannot be deleted by users). In Figure 6, only *phil* has the ability to create group chat rooms because his JID is configured to be the administrator of the rooms (between the <user/> tag).

[16] The word user in user.xml is replaced by the registered name of the actual user.

3. Testing of Jabber14-1.4.2 Server

The server was only informally tested using PSI clients on RedHat 9 VMs and Windows XP laptops. There are no testing tables for this installation because they were generated after the decision to move over to Fedora 7 was made.

4. Summary for the RedHat 8 Platform

The installation and testing of the jabberd14-1.4.2 server on the RedHat 8 was a success. The experience gained was useful for the subsequent experimentations.

D. FEDORA 7 PLATFORM

The installation and testing of jabberd14-1.6.1.1 server (the newest stable version) on Fedora 7 represents the first step towards porting the newer version of jabberd14 server to MYSEA on the STOP 7 environment. The Fedora 7 binary was installed as a VM on VMWare and the jabberd14 version 1.6.1.1 source code was downloaded from the official web site for the jabberd14 Project located at [34].

1. Installation of Jabberd14-1.6.1.1 Server

In addition to downloading and installing the jabberd14-1.6.1.1 server on the Fedora 7 VM, the MUC module was also installed. The JUD was not installed because of time constraints. Jabberd14-1.6.1.1 depends on GnuTLS, pthreads (and its development package) and Libtasn1 (and its development package and tools) [48], [49], [50]. Those additional components can be installed via standard package manager. Installation instructions were compiled for this exercise and can be found in Appendix C. One of the challenges was finding the patch for the configure.ac file. The patch was released via a JAdmin posting and corrected two problems [51]. The first problem involved a typographical error in a file name (the original configure.ac file had jabberd14-1.6.1.1 while the correct name of the file is jabberd14-1.6.1.1-p1). The second part of the patch corrected the configuration file to allow the program to check if libtasn1 version 0.3.0 or

greater was installed[17]. The default configurations of the server were not modified beyond the minimum settings for this experiment.

2. Testing of Jabberd14-1.6.1.1 Server

The topology of the test network matches Figure 5 and the results of the functional and exception tests are in Table 9 through Table 12 below. In these tables, the first four columns are identical to the first four columns in Table 4 through Table 8 in Section B of this chapter. The test procedures for these tests are in Appendix F. An "X" indicates that the particular test is an exception test and the expected result is "Fail." The last four columns represent the expected and observed test results. The word "Pass" indicates that the action that is described in the "Test Description" was successful. The word "Fail" indicates that the action described in the "Test Description" failed to happen. "NC" stands for non configurable. The "NC" test was not conducted. The analysis of the test results follows the tables. Because this server did not have the JUD module installed, the Service discovery and Jabber User Directory (JUD) management tests were not performed.

[17] Code dependency is discussed in Chapter VI.

Table 9. Account management on the Fedora 7 platform

Test #	Exception Test	Test Description	Test Objective	Expected results for user1	Expected results for user2	Actual result for user1	Actual result for user2
A1		Register account with a valid JID	Ensures that server can register accounts	Pass	Pass	Pass	Pass
A2	X	Register account with an invalid JID	Ensures that server does not register users with invalid JID	Fail	Fail	Fail	Fail
A3		Register account with a password	Ensures that server enforces registering with a password	Pass	Pass	Pass	Pass
A4	X	Register account without a password	Ensures that server enforces registering with a password	Fail	Fail	Fail	Fail
A5	X	Connect to server using SSL (legacy on port 5223)	Ensures that the server does not connect to a client on the legacy port 5223	Fail	Fail	Fail	Fail
A6		Connect to server without SSL (port 5222)	Ensures that server connects to a client on the reserved port 5222	Pass	Pass	Pass	Pass
A7	X	Connect to server with SSL (port 5222)	Ensures that server does not connect to a client on the reserved port with SSL	NC	NC	NC	NC
A8		Connect with a password	Ensures that server enforces the use of passwords	Pass	Pass	Pass	Pass
A9	X	Connect without a password	Ensures that server enforces the use of passwords	Fail	Fail	Fail	Fail
A10		Edit personal information	Ensures that server performs resource binding	Pass	Pass	Pass	Pass
A11		Add a contact that has a valid JID	Ensures that server lets users add contacts and displays them	Pass	Pass	Pass	Pass
A12	X	Add a contact that has an invalid	Ensures that server checks for	Fail	Fail	Fail	Fail

		JID	the JID validity before adding contacts				
A13		Delete contacts (buddies) from the contact list	Ensures that server updates the buddy list	Pass	Pass	Pass	Pass
A14		Forbid contacts to see status	Ensures that server applies the right restriction to the right contacts	Pass	Pass	Pass	Pass
A15		Delete an account just on client only	Ensures that server is not deleting information when deletion is not requested	Pass	Pass	Pass	Pass
A16		Delete an account on both the client and the server	Ensures that server is not retaining information about deleted account	Pass	Pass	Pass	Pass
A17		Connect to the server with a deleted account after test A15 is performed	Ensures that server does not purge its records	Pass	Pass	Pass	Pass
A18	X	Connect to the server with a deleted account after test A16 is performed	Ensure that server does not allow deleted users to connect	Fail	Fail	Fail	Fail
A19	X	Send server message to all users connected to the server)	Ensures that server does not allow broadcasting to all registered users	Fail	Fail	Fail	Fail
A20		Retrieve other users' contact information	Ensures that the server keeps information on its users	Pass	Pass	Pass	Pass
A21		Transfer files between clients	Ensures that file transfer functionality is available	Pass	Pass	Pass	Pass
A22		Retrieve communication history information	Ensures that server is storing records	Pass	Pass	Pass	Pass
A23		Deny a contact from viewing	Ensures that privacy rules	Pass	Pass	Pass	Pass

		status for the current session	regarding user status can be enforced by the server for the current session				
A24		Deny a contact from sending a message for the current session	Ensures that privacy rules regarding blocking messages from user can be enforced by the server for the current session	Pass	Pass	Pass	Pass
A25	X	Register with an account that was deleted from both the server and the client	Ensures that the server does not allow a user to re-register with the same JID after the account was deleted on the client as well as on the server	Pass	Pass	Fail	Fail
A26	X	Register with an account that was deleted from the client only	Ensures that the server does not allow a user to re-register with the same JID after the account was deleted on the client only	Fail	Fail	Fail	Fail

Table 10. Presence management on the Fedora 7 platform

Test #	Exception Test	Test Description	Test Objective	Expected results for Test user 1	Expected results for Test user 2	Actual result for user1	Actual result for user2
B1		Change presence to Available	Validates that users can become active after having been a different state	Pass	Pass	Pass	Pass
B2		Change presence to Free for Chat, Away, Not Available, Busy, Invisible and Offline	Validates that presence information can be changed and is reflected to users who subscribe to the presence	Pass	Pass	Pass	Pass
B3		Display offline contacts	Ensures that server can display correct presence for contacts	Pass	Pass	Pass	Pass

Table 11. Instant Messaging (IM) management on the Fedora 7 platform

Test #	Exception Test	Test Description	Test Objective	Expected results for Test user 1	Expected results for Test user 2	Actual result for user1	Actual result for user2
C1		Start a chat with someone registered with the server and is on the contact list	Ensures that server can retrieve information from users who are on the user's contact list	Pass	Pass	Pass	Pass
C2		Start a chat with someone registered with the server but is not in the contact list	Ensures that server can retrieve information from users who do not subscribe to the other user's roster	Pass	Pass	Pass	Pass
C3	X	Start a chat with someone by mistyping their JID (user ID)	Ensures that server uses Stringprep	Fail	Fail	Fail	Fail
C4	X	Start a chat with someone by mistyping their JID (server name)	Ensures that server checks for JID validity for contacts	Fail	Fail	Fail	Fail
C5		Have multiple chat sessions with different valid contacts	Ensures that the server does not mix chat sessions in a one to many paradigm	Pass	Pass	Pass	Pass
C6		Have multiple chat sessions with the same valid contact	Ensures that server does not mix chat sessions in a one to one paradigm	Pass	Pass	Fail	Fail
C7		Add Emoticons to chat messages	Ensures that server can transmit special character	Pass	Pass	Pass	Pass

Table 12. Multi-User Chat (MUC) management on the Fedora 7 platform

Test #	Exception Test	Test Description	Test Objective	Expected results for Test user 1	Expected results for Test user 2	Actual result for user1	Actual result for user2
D1		Create a default group chat with a valid JID	Ensures that server implements MUC	Pass	Pass	Pass	Pass
D2	X	Create a default group chat with an invalid JID	Ensures that server enforces valid JID	Fail	Fail	Fail	Fail
D3		Join a private group chat with proper credentials (password required)	Ensures that server can enforce that the moderator can set rules for joining password-protected group chat rooms	Pass	Pass	Pass	Pass
D4	X	Join a private group chat without membership (member list enforced)	Ensures that server can enforce that the moderator can set rules for joining membership-protected group chat rooms	Fail	Fail	Fail	Fail
D5	X	Join a private group chat without necessary credentials (password required)	Ensures that server can enforce group chat rules set by moderator	Fail	Fail	Fail	Fail
D6		Join a publicly open group chat	Ensures that server allows any user to join public a group chat room	Pass	Pass	Pass	Pass
D7		Configure a group chat room after it is created (as its creator)	Ensures that the configuration options of a group chat room are available to the moderator of the room	Pass	Pass	Pass	Pass
D8	X	Configure a room after it is created as a normal user	Ensures that the configuration options of group chat room are not available to normal users	Fail	Fail	Fail	Fail
D9		Inviting contact to join private group chat rooms	Ensures that server can enforce MUC settings	Pass	Pass	Pass	Pass

D10		Communicating in authorized group chat rooms	Ensures that server keeps the flow of messages in the proper order and label their authorship properly	Pass	Pass	Pass	Pass
D11		Ban a user from a group chat room when having the role of moderator	Ensures that the server manages configuration options of group chat rooms properly	Pass	Pass	Pass	Pass
D12	X	Ban a user from a group chat room when not having the role of moderator	Ensures that the server makes the distinction between moderator and normal users	Fail	Fail	Fail	Fail
D13	X	Joining the group chat room after been banned	Ensures that banning a user from a group chat is a permanent/persistent setting in the configuration file	Fail	Fail	Fail	Fail
D14		Kick (temporarily remove) a user from a group chat room when having the role of moderator	Ensures that the moderator of a group chat room can kick users	Pass	Pass	Pass	Pass
D15		Joining a group chat room after been kicked	Ensures that kicking is only on a session basis	Pass	Pass	Pass	Pass

3. Test Analysis

A7 is a test that could not be configured on the client because the server is the one to enforce the use of SSL. The observed result of test A18 was not as expected. The server did not alert the user that the connection failed because the account had been deleted; instead, the icon in the client appeared to be connecting, but hung. This behavior does not mean that it is a server issue; it could also be a bug in the client software (the software might not be able to interpret the error messages that the server is sending). Sending raw XML stanzas via a Telnet connection directly to the server would most likely the cause of the problem. In test A25, it was expected that the server would not allow a deleted user to re-register immediately with the same JID because the configuration file has a regtimeout[18] value of 157680000 seconds. As a consequence, a "conflict in account creation" error message was received. For test C6, the observed result is different than the expected result. Here, it was predicted that one user could have multiple IM sessions with the same contact, but in reality the client can detect that an IM session is already in progress and does not open another window. The MUC tests were all successful. Behaving as predicted, they provide an extra layer of privacy and protection for its users.

4. Summary for the Fedora 7 Experiment

The installation and testing of the jabberd14-1.6.1.1 server on Fedora 7 were successful. The next step was to install the same jabberd14 software on STOP 7 and repeat the tests.

[18] A regtimeout value of 0 means that users can re-register with the same JID immediately after the account has been deleted and a regtimeout value of -1 means that deleted accounts are blocked from re-registering with that same JID forever.

E. XTS400 STOP 7 PLATFORM

This installation and testing exercise represents the final jabberd14 porting experiment. Because of the success of the experiment of the server on the Fedora 7 platform, expectations were high that it would work on the STOP 7.

1. Installation of Jabberd14-1.6.0 Server

The original plan was to use the same jabberd14-1.6.1.1 software that was used on Fedora 7 on STOP 7. However, this plan was changed after it was discovered that the use of the GnuTLS package cannot be disabled in version 1.6.1.1. GnuTLS is free software that implements the TLS protocol, which is described in the RFC 4346. It provides the API that can be used by applications to use TLS to secure the transport layer in a network [48]. GnuTLS has dependencies that are not available in STOP 7, and due to time constraint, jabberd14 version 1.6.0 was used instead of version 1.6.1.1. Version 1.6.0 does not require GnuTLS, as it still uses OpenSSL[19] as the mean to secure the transport layer and the functionality of the jabberd14-1.6.0 server is identical to the 1.6.1.1[20] [46].

A number of challenges were encountered during the server installation on STOP 7. The main ones for this experiment were:

- MAC policy. The default security level for the administrator is System Low (syslo) and the default levels of the CD drive and Ethernet device are System High (syshi). To streamline the testing process, the security level used for the test sessions was also syslo. Hence, the level of the Ethernet device must be set to syslo. The security level of the CD drive was temporarily set to syslo to read in the downloaded software and reset back to syshi as required by the MYSEA convention. Additional steps were also required to properly set the security level of the administrator so that the administrator could change the devices' security levels. These steps are not complicated, but need to be done in the proper order and not overlooked.

[19] With the prior versions, there was a toggle to configure the server with or without openssl

[20] According to the 1.6.1.1 release notes, there were no new features introduced in 1.6.1.1 from the 1.6.0 server.

- Dependencies. STOP 7 did not provide pkg-config, expat, libidn or glib packages[21] or a package manager such as yum to install packages. The source code for these packages was installed separately before the jabberd14-1.6.0 server could be installed. A dependency chart is documented in Chapter V.
- No GUIs. The only way to have multiple command prompts on different screens is to use the ALT-F keys combination and login multiple times. Having a minimum of two screens is necessary because the MUC module needs to be launched separately than the server, but needs to run concurrently to the server. In addition to using multiple screens to start individual programs, having multiple prompts is very practical when looking at logs and monitoring the service concurrently.

In addition to these challenges, the wget command did not work for downloading the source code on the jabberd14-1.6.0 web site[22]. In order to install the server source code along with its dependencies and the source code for the MUC and JUC modules and their dependencies, two methods were used. The first method used CDs that were made on a Fedora 8 machine and the second method used the wget command to retrieve files from an FTP server on a networked Fedora 8 machine.

There is an additional challenge concerning the JUD module. This module's make file assumes the existence of a file called *platform-settings*. To work around that dependency, platform-settings file used in the RedHat 8 environment was brought over to the STOP system. This allowed the JUD module to compile successfully.

Appendix D contains the installation guide to set up jabberd14-1.6.0 server on an XTS-400 with a STOP 7 beta operating system.

[21] An explanation of the purpose of those packages is in Chapter V

[22] Even though the name of the site could be resolved into an IP address, the STOP 7 was not able to follow the path on the server in order to download the software.

2. Testing Jabberd14-1.6.0 Server

The same test network topology depicted in Figure 5 was used. The tests and test results are summarized in Table 13 through Table 17. The tests conducted in Table 9 through Table 12 are the same tests that were performed on the Fedora 7 platform, with the addition of the Service discovery and Jabber User Directory (JUD) management tests (Table 17). The same Gajim client was used and therefore, the same procedures described in Appendix F were used. Only the PSI client from [47] was used in the case of the E2 test (explanation provided in the test analysis after the tables).

Table 13. Account management on the STOP 7 platform

Test #	Exception Test	Test Description	Test Objective	Expected results for user1	Expected results for user2	Actual result for user1	Actual result for user2
A1		Register account with a valid JID	Ensures that server can register accounts	Pass	Pass	Pass	Pass
A2	X	Register account with an invalid JID	Ensures that server does not register users with invalid JID	Fail	Fail	Fail	Fail
A3		Register account with a password	Ensures that server enforces registering with a password	Pass	Pass	Pass	Pass
A4	X	Register account without a password	Ensures that server enforces registering with a password	Fail	Fail	Fail	Fail
A5	X	Connect to server using SSL (legacy on port 5223)	Ensures that the server does not connect to a client on legacy port 5223	Fail	Fail	Fail	Fail
A6		Connect to server without SSL (port 5222)	Ensures that server connects to a client on the reserved port 5222	Pass	Pass	Pass	Pass
A7	X	Connect to server with SSL (port 5222)	Ensures that server does not connect to a client on the reserved port with SSL	NC	NC	NC	NC
A8		Connect with a password	Ensures that server enforces the use of passwords	Pass	Pass	Pass	Pass
A9	X	Connect without a password	Ensures that server enforces the use of passwords	Fail	Fail	Fail	Fail
A10		Edit personal information	Ensures that server performs resource binding	Pass	Pass	Pass	Pass
A11		Add a contact that has a valid JID	Ensures that server lets users add contacts and displays them	Pass	Pass	Pass	Pass
A12	X	Add a contact that has an invalid	Ensures that server checks for	Fail	Fail	Fail	Fail

		JID	the JID validity before adding contacts				
A13		Delete contacts (buddies) from the contact list	Ensures that server updates the buddy list	Pass	Pass	Pass	Pass
A14		Forbid contacts to see status	Ensures that server applies the right restriction to the right contacts	Pass	Pass	Pass	Pass
A15		Delete an account just from on the client	Ensures that server is not deleting information when deletion is not requested	Pass	Pass	Pass	Pass
A16		Delete an account on both the client and the server	Ensures that server is not retaining information about deleted account	Pass	Pass	Pass	Pass
A17		Connect to the server with a deleted account after test A15 is performed	Ensures that server does not purge its records	Pass	Pass	Pass	Pass
A18	X	Connect to the server with a deleted account after test A16 is performed	Ensures that server does not allow deleted users to connect	Fail	Fail	Fail	Fail
A19	X	Send server message to all users connected to the server	Ensures that server does not allow broadcasting to all registered users	Fail	Fail	Fail	Fail
A20		Retrieve other users' contact information	Ensures that the server keeps information on its users	Pass	Pass	Pass	Pass
A21		Transfer files between clients	Ensures that file transfer functionality is available	Pass	Pass	Pass	Pass
A22		Retrieve communication history information	Ensures that server is storing records	Pass	Pass	Pass	Pass
A23		Deny a contact from viewing	Ensures that privacy rules	Pass	Pass	Fail	Fail

		status	regarding user status can be enforced by the server				
A24		Deny a contact from sending a message	Ensures that privacy rules regarding blocking messages from user can be enforced by the server for the current session	Pass	Pass	Fail	Fail
A25	X	Register with an account that was deleted from both the server and the client	Ensures that the server does not allow a user to re-register with the same JID after the account was deleted on the client as well as on the server	Pass	Pass	Fail	Fail
A26	X	Register with an account that was deleted from the client only	Ensures that the server does not allow a user to re-register with the same JID after the account was deleted on the client only	Fail	Fail	Fail	Fail

Table 14. Presence management on the STOP 7 platform

Test #	Exception Test	Test Description	Test Objective	Expected results for Test user 1	Expected results for Test user 2	Actual result for user1	Actual result for user2
B1		Change presence to Available	Validates that users can become active after having been in a different state	Pass	Pass	Pass	Pass
B2		Change presence to Free for Chat, Away, Not Available, Busy, Invisible, and Offline	Validates that presence information can be changed and is reflected to users who subscribe to the presence	Pass	Pass	Pass	Pass
B3		Display offline contacts	Ensures that server can display correct presence for contacts	Pass	Pass	Pass	Pass

Table 15. Instant Messaging (IM) management on the STOP 7 platform

Test #	Exception Test	Test Description	Test Objective	Expected results for Test user 1	Expected results for Test user 2	Actual result for user1	Actual result for user2
C1		Start a chat with someone registered with the server and is on the contact list	Ensures that server can retrieve information from users who are on the user's contact list	Pass	Pass	Pass	Pass
C2		Start a chat with someone registered with the server but is not in the contact list	Ensures that server can retrieve information from users who do not subscribe to the user's roster	Pass	Pass	Pass	Pass
C3	X	Start a chat with someone by mistyping their JID (user ID)	Ensures that server uses Stringprep	Fail	Fail	Fail	Fail
C4	X	Start a chat with someone by mistyping their JID (server name)	Ensures that server checks for JID validity for contacts	Fail	Fail	Fail	Fail
C5		Have multiple chat sessions with different valid contacts	Ensures that the server does not mix chat sessions in a one to many paradigm	Pass	Pass	Pass	Pass
C6		Have multiple chat sessions with the same valid contact	Ensures that server does not mix chat sessions in a one to one paradigm	Pass	Pass	Fail	Fail
C7		Add Emoticons to chat messages	Ensures that server can transmit special character	Pass	Pass	Pass	Pass

Table 16. Multi-User Chat (MUC) management on the STOP 7 platform

Test #	Exception Test	Test Description	Test Objective	Expected results for Test user 1	Expected results for Test user 2	Actual result for user1	Actual result for user2
D1		Create a default group chat with a valid JID	Ensures that server implements MUC	Pass	Pass	Pass	Pass
D2	X	Create a default group chat with an invalid JID	Ensures that server enforces valid JID	Fail	Fail	Fail	Fail
D3		Join a private group chat with proper credentials (password required)	Ensures that server can enforce that the moderator can set password protected group chat rooms	Pass	Pass	Pass	Pass
D4	X	Join a private group chat without membership (member list enforced)	Ensures that server can enforce that the moderator can set membership protected group chat rooms	Fail	Fail	Fail	Fail
D5	X	Join a private group chat without the necessary credentials (password required)	Ensures that server can enforce group chat rules set by moderator	Fail	Fail	Fail	Fail
D6		Join a publicly open group chat	Ensures that server allows any user to join public a group chat room	Pass	Pass	Pass	Pass
D7		Configure a room after it is created (as its creator)	Ensures that the configuration options of a group chat room are available to the moderator of the room	Pass	Pass	Pass	Pass
D8	X	Configure a group chat room after it is created as a normal user	Ensures that the configuration options of group chat room are not available to normal users	Fail	Fail	Fail	Fail
D9		Inviting contacts to join	Ensures that server can enforce MUC	Pass	Pass	Pass	Pass

		private group chat rooms	settings				
D10		Communicating in authorized group chat rooms	Ensures that server keeps the flow of messages in the proper order and label their authorship properly	Pass	Pass	Pass	Pass
D11		Ban a user from a group chat room when having the role of moderator	Ensures that the server manages configuration options of group chat rooms properly	Pass	Pass	Pass	Pass
D12	X	Ban a user from a group chat room when not having the role of moderator	Ensures that the server makes the distinction between moderator and normal users	Fail	Fail	Fail	Fail
D13	X	Joining the group chat room after been banned	Ensures that banning a user from a group chat is a permanent/persistent setting in the configuration file	Fail	Fail	Fail	Fail
D14		Kick (temporarily remove) a user from a group chat room when having the role of moderator	Ensures that the moderator of a group chat room can kick users	Pass	Pass	Pass	Pass
D15		Joining a group chat room after been kicked	Ensures that kicking is only on a session basis	Pass	Pass	Pass	Pass

Table 17. Service discovery and Jabber User Directory (JUD) management on the STOP 7 platform

Test #	Exception test	Test Description	Test Objective	Expected results for Test user 1	Expected results for Test user 2	Actual result for user1	Actual result for user2
E1		List the rooms available and the number of the participants in each room	Ensures that server implements service discovery (XEP-0030)	Pass	Pass	Pass	Pass
E2		Access the user directory	Ensures that the server implements JUD	Pass	Pass	Pass	Pass

3. Test Analysis

The results that were obtained from the tests in Table 13 through 17 are the same as from Table 9 through 12 (Fedora 7). Although, the Service discovery and Jabber User Directory (JUD) management tests were not performed on the Fedora platform, they were performed on the STOP 7 platform. As shown in Table 17, the service discovery mechanism adequately retrieved the names of the public[23] group chat rooms, the number of participants presently in each room, the description of the rooms and their respective group chat room ID[24] (i.e., public@conference.thesis.nps.edu). However, the feature to search for other users (JUD) did not work with the Gajim client. As the mouse was moved over the JUD features, a text box would display a message indicating that either the service was not implemented on the server or it was a legacy service (implying that is was no longer supported). In order to confirm that the server was indeed supporting the JUD module, another client (PSI from [47]) was used. This test successfully confirmed that the jabberd14-1.6.0 server supports JUD.

4. Summary of the XTS-400 Experiment

In this sub-section, the challenges of installing the jabberd14-1.6.0 on STOP 7 beta were discussed. A summary of the test cases was also included. The success of installing and testing jabberd14-1.6.0 on STOP 7 beta implies that the goal of porting jabberd14-1.6.0 server to a MLS environment can be undertaken.

F. SUMMARY

It is encouraging to have successfully installed and tested the jabberd14-1.6.0 server, its dependencies and its external modules (MUC and JUD) on the STOP 7 because it indicates that this study can continue. In the next chapter, Chapter V, code review and technical and security issues that might affect the porting of chat to the MLS environment of MYSEA are discussed.

---

[23] The server did not display the name of group chat room protected by membership, a password or both.

[24] ID for a group chat room has no correlation with a JID for a user.

THIS PAGE INTENTIONALLY LEFT BLANK

V. TECHNICAL AND SECURITY ISSUES

This chapter describes the code structure, technical and security issues based on the installation and testing of the jabberd14-1.6.0 server on the STOP OS 7 described in the previous chapter. There are three major sections. The first section gives an overview of the structure of the code of the jabberd14-1.6.0 server and the code dependencies that will be needed to support a subsequent module-by-module code review. The second section discusses the known technical issues for porting the server code to the STOP OS 7 environment and the last section discusses security issues.

A. CODE SURVEY

This section consists of three sub-sections. The first sub-section describes the external library dependencies that are needed in order to install the server and the external extensions that were tested (MUC and JUD). The second sub-section shows the XEP's dependencies between each other and in relation to the core modules. Finally, the last sub-section describes the structure of the code of the server.

1. External Library Dependencies

Unmodified STOP 7 supports a number of libraries. However, not all of the libraries that were needed to successfully install and run jabberd14-1.6.0 server were present on the STOP 7. For this reason, four additional packages had to be installed. Below is the list of external libraries that jabberd14-1.6.0 along with the MUC and the JUD modules require:

- **expat**. "Expat is a library, written in C, for parsing XML documents [52]." This is an open-source project that underlies the XML parser for the Mozilla project, perl's XML::Parser and other open-source XML parsers. This is a "very fast parser with high standard for reliability, robustness and correctness [52]." This open source parser is available on sourceforge [53]. This project is

mature, active and well supported [54]. The version used for this thesis is 2.0.1. It requires autoconf-2.52[25], and an 'm4' macro package. STOP 7 implements autoconf and m4.

- **libidn**. This GNU library is an implementation of the stringprep[26] used for internationalized domain names. It is written in C [50]. It is especially important because it offers profiles for Kerberos 5, Nameprep, SASL and XMPP. Libidn is developed for the GNU/Linux system, but runs on over 20 other platforms, including Windows [50]. libidn is a well documented and well maintained open source GNU library. The version used for this thesis is 0.6.8 and the README-alpha file in the package claims that "the program is unsecured and should not be used in a production environment [55]." Libidn depends on libtool and pkg-config. STOP 7 implements libtool[27] but not pkg-config.

- **pkg-config**. This helper tool supports inserting the correct compiler options on the command line when compiling an application or library [56]. The version that was installed for this thesis is 0.23. This library has no dependencies that are not already available on STOP OS 7.

- **glib**. This library forms the basis for projects such as GTK+[28] and GNOME[29]. It provides data structure for C, portability wrappers, and interfaces for such runtime functionality as event loops, threads dynamic loading and object systems [57]. The version of glib used for this thesis is 2.0.7. Glib requires pkg-config and GNUmake. GNUmake is part of STOP OS 7.

---

[25] autoconf is a package for creating Bourne shell scripts to configure source code packages using templates [58].

[26] stringprep is a framework for preparing Unicode text strings in order to increase the likelihood that string input and string comparison work in ways that make sense for typical users throughout the world. The stringprep protocol is useful for protocol identifier values, company and personal names, internationalized domain names, and other text strings [26]. The stringprep standard is documented in RFC 3454.

[27] Libtool is a generic library support script [59].

[28] GTK+ is a toolkit for creating graphical user interfaces. It puts together glib, pango, cairo and ATK [60].

[29] GNOME is a project similar to GTK+. More details are at [61].

- **platform-settings**. This is not a library, but a file that contains a bash shell script that the server generates to set platform-specific build environment variables when the source code is compiled [62]. However, the jabberd14-1.6.0 source code that was installed on STOP OS 7 did not generate that file. As a shortcut to installing the module, the platform-settings file generated by the jabberd1.4.2 on RedHat 8 was copied to the STOP OS 7 environment. This worked because the same JUD code was used on both platforms.

Figure 6 shows the dependency tree for the jabberd14-1.6.0 as configured in the previous chapter including the external MUC and JUD modules.

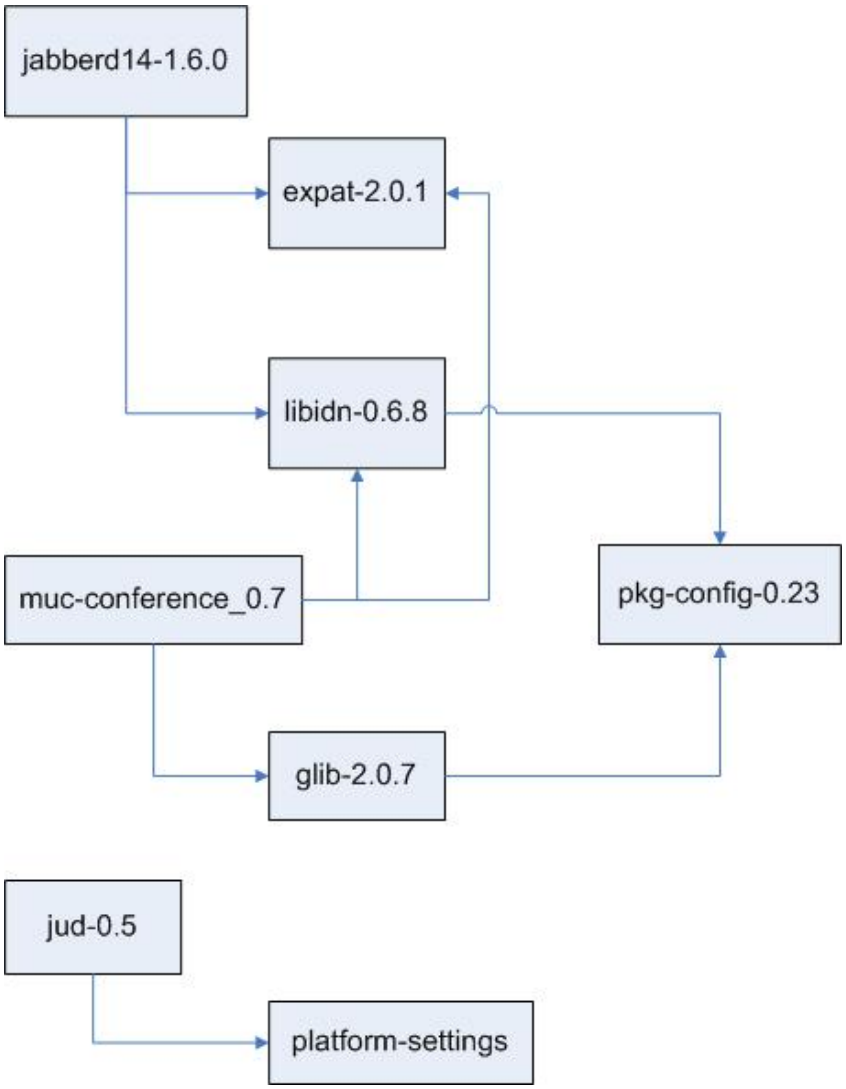


Figure 6. Library dependencies

A number of packages must be installed in a specific order.

a. For jabberd14, the following build ordering is required:

1. pkg-config-0.23. This package does not have dependencies that are not already available on the STOP 7 operating system
2. libidn-0.6.8. This package depends on pkg-config.
3. expat-2.0.1. This package does not have dependencies that are not already available on the STOP 7 operating system

b. For the MUC module, the following build ordering is required:

1. pkg-config-0.23.
2. libidn-0.6.8.
3. expat-2.0.1.
4. glib-2.0.7. This package depends on pkg-config.
5. jabberd14-1.6.0. This package depends on expat and libidn

c. For the JUD module, the following build ordering is required:

1. pkg-config-0.23.
2. libidn-0.6.8.
3. expat-2.0.1.
4. platform-settings bash file in the jabberd14-1.6.0 directory
5. jabberd14-1.6.0

2. XEP's Dependencies

In addition to the basic functionalities required by RFC 3920 and RFC 3921, Jabberd14-1.6.0 server supports fifteen XEPs (see Chapter III for details). The main purpose of the XEPs is to extend the XMPP-core and XMPP-IM capabilities. For this analysis, the XEPs are divides into two groups: directly-supported and indirectly-supported. Table 18 and Table 19 summarize these two groups, respectively.

XEP number	XEP Name	Status
XEP-0012	Last activity	Draft
XEP-0013	Flexible offline message retrieval	Draft
XEP-0016	Privacy Lists	Draft
XEP-0030	Service Discovery	Final
XEP-0033	Extended stanza addressing	Draft
XEP-0045	Multi-User Chat	Draft
XEP-0048	Bookmarks	Draft
XEP-0049	Private XML storage	Active
XEP-0050	Ad-Hoc commands	Draft
XEP-0054	vcard-temp	Active
XEP-0092	Software version	Draft
XEP-0114	Jabber component protocol	Active
XEP-0145	Annotations	Active
XEP-0153	vCard-based avatars	Active
XEP-0199	XMPP Ping	Draft

Table 18. Directly-supported XEP

XEP number	XEP Name	Status
XEP-0004	Data forms	Final
XEP-0060	Publish-Subscribe	Draft
XEP-0068	Field standardization for data forms	Active
XEP-0082	XMPP data and time profiles	Active
XEP-0115	Entity capabilities	Draft
XEP-0128	Service discovery extensions	Active
XEP-0131	Stanza headers and Internet metadata (SHIM)	Draft
XEP-0223	Best practices for persistent storage of public data via Publish-subscribe	Proposed

Table 19. Indirectly-supported XEPs

All of the XEPs that jabberd14-1.6.0 supports require the server to meet the XMPP-core standard [2]. Furthermore, some XEPs also require the server to support XMPP-IM [3], other XEPs or external standards. For example, XEP-0223, an indirectly-supported XEP, depends on XEP-0115, XEP-0060 and XEP-0030.

Figure 7 shows a dependency tree for the directly and indirectly supported XEPs. At the root is the XMPP-core and XMPP-IM. The arrows mean “supports” or “is required for.” The requirements for core functionality are met either directly or indirectly via other dependencies.

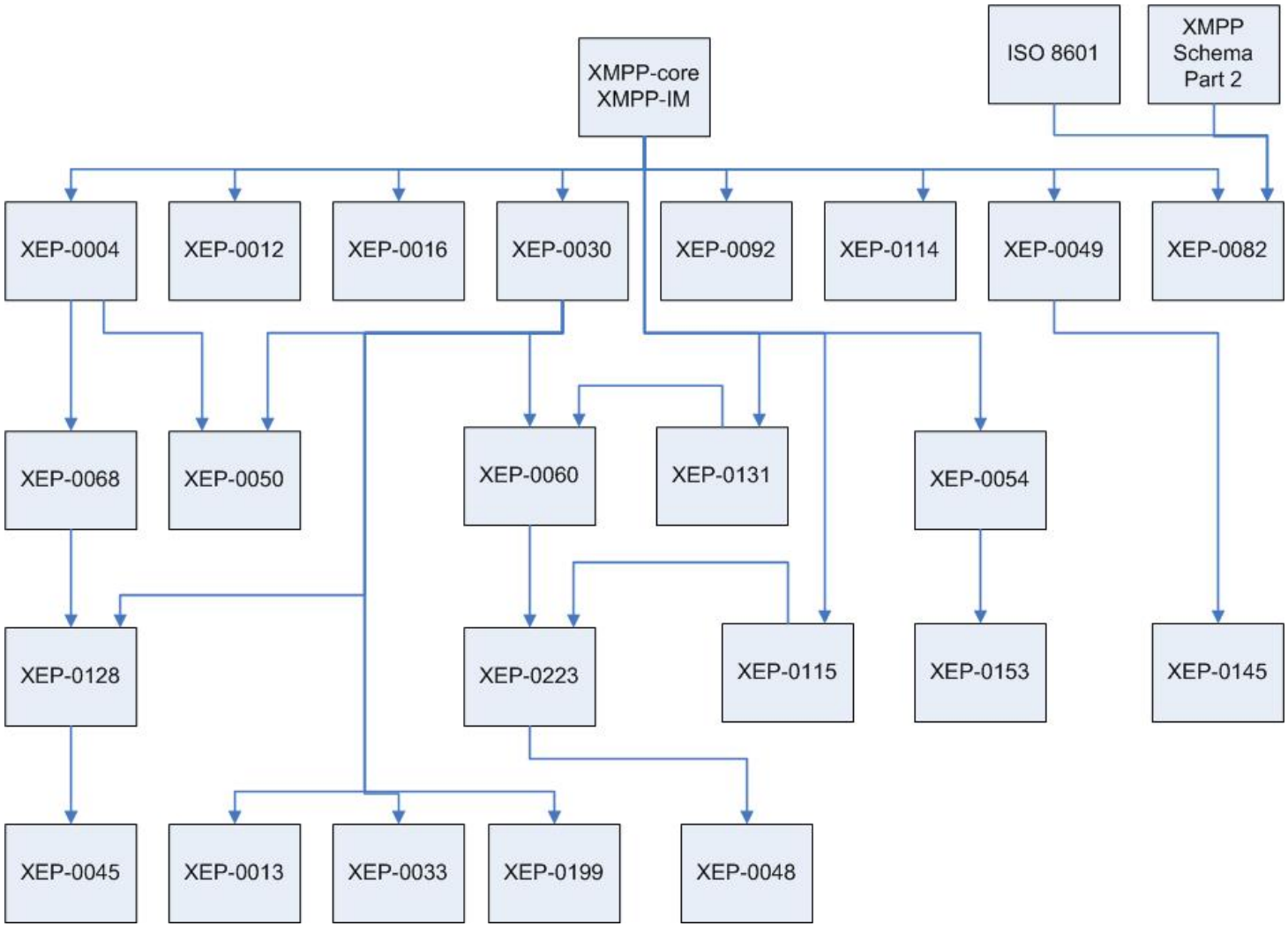


Figure 7. XEPs dependency tree

Clearly some extensions have more dependencies than others. However, half of the XEPs that the jabberd14-1.6.0 server supports do not have any dependencies besides the core. It is interesting to note that XEP-0082 is the only XEP that requires ISO 8601 [63] and XMPP Schema Part 2, both of which are not XEPs.

3. Structure of the Code of the Jabberd14-1.6.0 Server

The jabberd14-1.6.0 server is composed of seven major modules. Below is a description of the functionality of each module, including the sql module, even though this project uses flat file storage. The modules are arranged alphabetically. It is important to note that modules are hierarchically organized and some modules encapsulate other modules. More information can be found at [64].

- **dialback**. This module manages the server to server connections. It includes four different files. The dialback.c is responsible for dispatching the work to either dialback_in.c or dialback_out.c. These modules handle the incoming or outgoing connections between servers. The directory also contains a header file for module interface declaration. [65].
- **dnsrv**. This module implements the Domain Name Server (DNS) resolver. This mechanism does not read the /etc/hosts file, but instead makes DNS queries. This module interacts with the server-to-server connection managers as it gets all stanzas that are not intended to be delivered locally. This module consists of only three source code files, including a header file [66].
- **jabberd**. This module contains the executable base (binary) of the server which implements the XML router. A router normally deals with packets; however, an XML router deals with stanzas. The file jabberd.c contains the entry point and control functions. In addition to XML routing, this module contains the C code that manages threads, a heartbeat mechanism, a logging service, a configuration file handler, a network sockets handler, an interface to the database and an access control list verifier [67]. The jabberd module also contains two other modules (base and lib). The base module is made out of C

files to take care of base handlers, which connects targets to sources to the XML router [68]. The lib module contains the library functions used by the entire server [69].

- **jsm**. “The jabber session manager is the component of jabberd14 that handles the visible part of transporting and storing messages, managing presence and presence subscriptions [70].” In the jsm directory, there is a directory called modules on which the jsm functionality rests. This module consists of two parts: the base component and a collection of external modules, each implements a specific JSM functionality, e.g., presence and in-band registration.
- **pthsock**. This module manages the client-to-server connections. “The task of this component is to accept incoming TCP/IP requests from jabber clients, and to forward the received stanzas to the session manager of the user [71].”
- **xdb_file**. This module implements the storage of persistent data in XML files [72].
- **xdb_sql**. This module handles the storage of persistent data using a SQL database [73].

Figure 8 illustrates the different modules, including their sub-modules as an attempt to graphically summarize the description above.

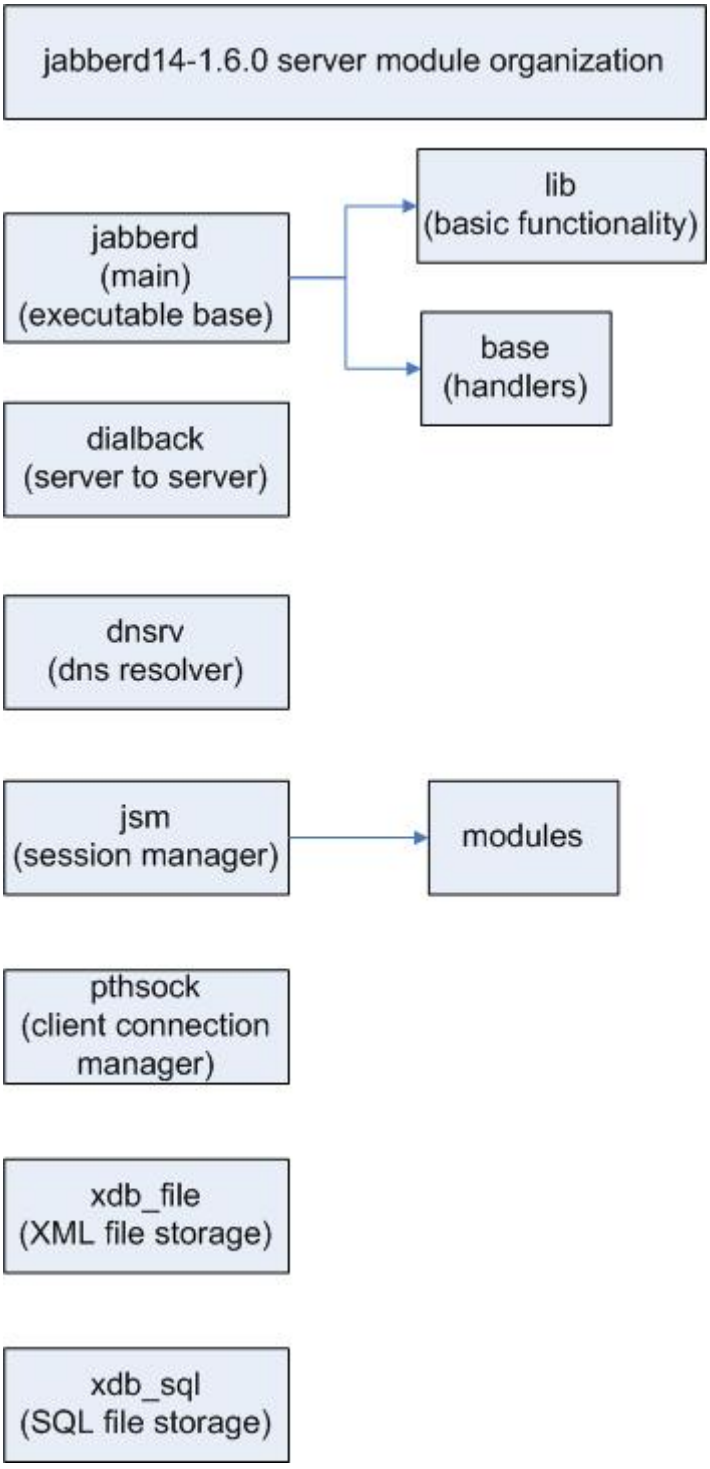


Figure 8. jabberd14-1.6.0 module composition

a. Entry Point of Jabberd14-1.6.0 Server

The main entry point of the Jabber server is in the jabberd.c file. Since the service can be configured with special options (like IPv6) and can be invoked with different flags on the command line, *main()* checks for those configurations and settings and initializes the server accordingly. After checking parameters and initializing the signal handlers, *main()* calls the following functions[30] in the order listed. This list was made with the help of the jabberd.org documentation [64]:

pth_init(). This function is from the gnupthread library; external to the jabberd code. It initializes the Pth library.

heartbeat_birth(). This function calls the new function _new_beat(), which allocates memory for a new "heartbeat ring" structure, initializes the "heartbeat ring" and starts a new thread for the heartbeat mechanism by calling pth_spawn().

register_beat(). This function registers a function to receive heartbeats i.e., to be called regularly. This includes setting up a record that defines the heartbeat for the jabberd signal handler and inserting the new record in the global "heartbeat ring."

mio_init(). The MIO module manages the input and output to the service, hiding the detail of sockets from the components of jabberd. This function must be called before the other functions in the MIO module can be used. This function creates the variables that will be needed for communication. It checks to determine if TLS/SSL is to be used and initializes those services accordingly. It then allocates and initializes module databases, sets up the "karma" heartbeat, and creates and starts the MIO thread. Karma is "an input/output rate limiting system that the Jabber team came up with to prevent bandwidth hogging [74]."

base_init(). This function fills in part the run-time memory pool data structure to contain the base configuration handlers. It does this by calling: base_accept(),

[30] Some of those functions might be called with parameters, but for clarity purposes, the parameters are not included here.

base_connec(), base_dir(), base_file(), base_format(), base_load(), base_null(), base_stderr(), base_syslog(), base_to(), base_unsubscribe(), base_importspool().

deliver_init(). This function initializes the XML stanza delivery system. It registers additional configuration handlers that are specific to the delivery mechanisms.

After initialization, *main()* enters a loop, from where it calls: **pth_ctrl()**,and **pth_sleep(60)** by reading the pth schedulers's average load every 60 seconds. Both the pth_ctrl and pth_sleep are from the gnupth library. This loop keeps the server alive.

The above list illustrates the initialization and run-time control logic of jabberd. An exhaustive review of the code is outside the scope of this thesis.

b. Exploring XEPs

In Chapter III, we listed the different XEPs that jabberd14 server advertises to support [34]. After assessing the code in more detail, it appears that not all of the extensions that were claimed to be supported actually are supported. Furthermore, there are code modules for XEPs that are not listed at [34]. Individual XEPs are implemented as sub-modules of the jsm (Jabber session manager) module. Figure 9 shows the structure of the *modules* directory, inside the jsm directory, and indicates which XEP(s) jabberd14 implements.

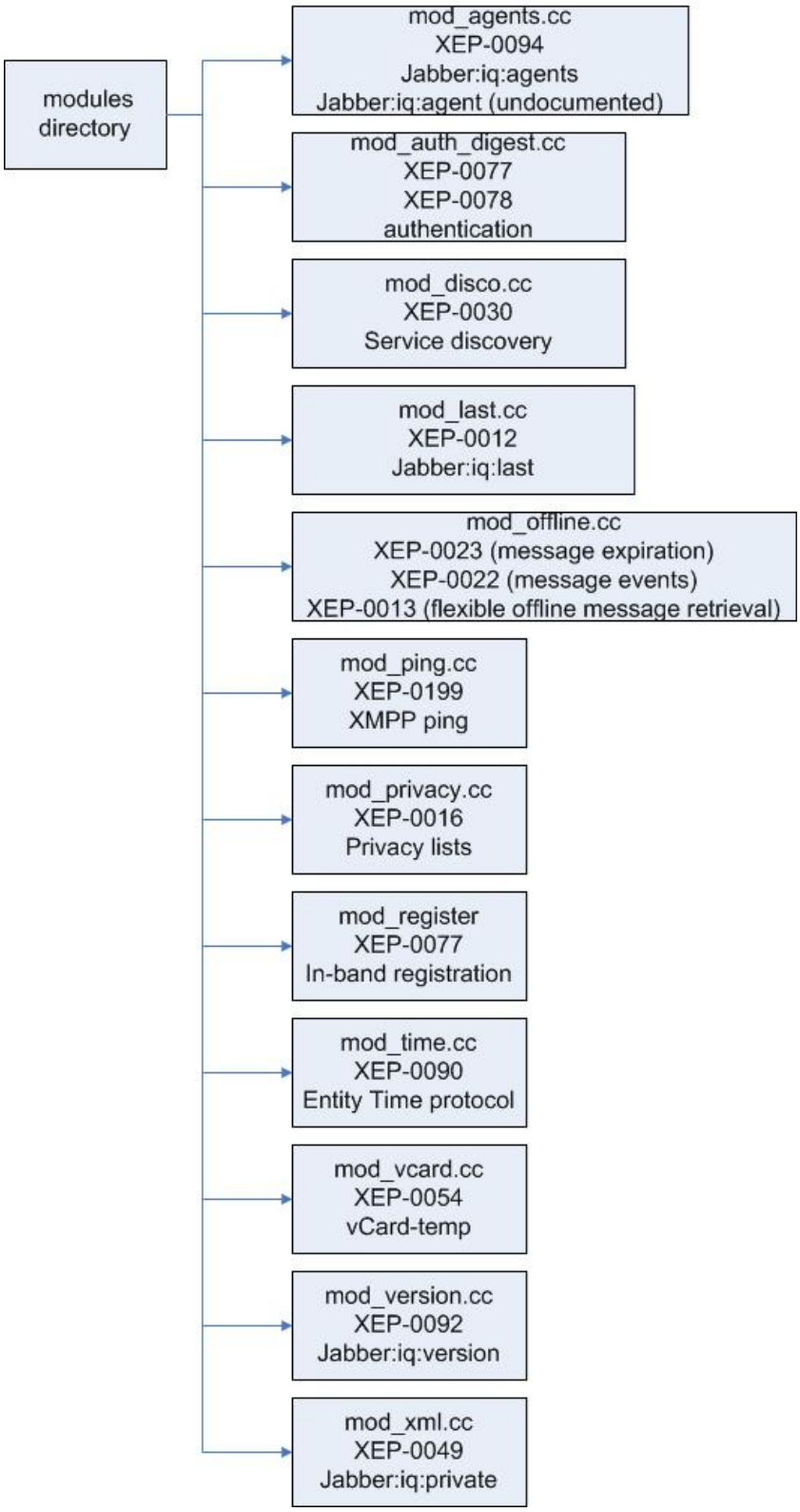


Figure 9. The *modules* Directory

Table 20 shows the XEPs that are implemented in the *modules* directory. Note that the XEP-0045 (MUC) is not mentioned in the table since it is an external module and must be installed separately. The JUD functionality is not present in this table either because it is an undocumented[31] module.

XEP number implemented	Description	Status	Claimed to be supported
XEP-0011	Jabber browsing	Deprecated	No
XEP-0012	Last Activity	Draft	Yes
XEP-0013	Flexible Offline Message Retrieval	Draft	Yes
XEP-0016	Privacy Lists	Draft	Yes
XEP-0022	Message Events	Deprecated	No
XEP-0023	Message Expiration	Deprecated	No
XEP-0030	Service Discovery	Final	Yes
XEP-0049	Private XML Storage	Active	Yes
XEP-0054	vcard-temp	Active	Yes
XEP-0077	In-Band Registration	Final	No
XEP-0078	Non-SASL Authentication	Deprecated	No
XEP-0090	Entity Time	Deprecated	No
XEP-0092	Software Version	Draft	Yes
XEP-0094	Agent Information	Obsolete	No
XEP-0199	XMPP Ping	Draft	Yes

Table 20. XEPs implemented in the *modules* directory

[31] There is no XEP standard that was reviewed and accepted by the XSF for this functionality.

B. TECHNICAL ISSUES

The most important technical issue to be resolved by the subsequent porting work is transforming the server from running as a standalone daemon to running as an inetd or xinetd daemon. Instead of the jabberd14-1.6.0 server listening with its default daemon on ports 5222 and 5269, the SSS process would listen on those ports and spawn instances of the server at the classification level of the communicating entities.

During the installation of jabberd14-1.6.0 on STOP 7, the pwd.h and grp.h were both commented out in the jabberd.c file because the STOP 7 does not implement those services in the same way as a traditional Linux installation. Line 171 in poptconfig.c in the pkg-config-0.23 was also commented out because it uses the getuid() function, which also deals with the way user credentials are gathered and checked. These shortcuts facilitated the installation and testing of the jabberd14-1.6.0 server on the XTS-400. Because of time constraints, the code was not modified in order to obtain the user information from the appropriate database maintained by the operating system.

C. SECURITY CONCERNS

The security concerns come from two different main sources. The first one is the code and relates to bugs, memory leaks, buffer overflow and other flaws and vulnerabilities that can exists in code. The second source of security concern resides in the configuration of the server. Since the first issue cannot be addressed without a thorough code review, which is outside the scope of this thesis, the issue dealing with the configuration file is explored below.

Securing the jabberd14-1.6.0 server in the MYSEA environment can be done by hardening the configuration settings of the jabberd.xml file. Below is a list of recommendation that can be enforced by modifying the configuration file. They will make the server more secure even though they will reduce the usability of the server.

- **In-band registration**. In the Internet environment, having in-band registration setting makes sense, but it might not make sense in an intranet such as MYSEA. This feature allows anyone to register on the server. If users need to

go through an administrator to get an account on the MYSEA server, the chance of supporting unwanted users should be inexistent.

- **No remote administration**. The administrator might be tempted to allow him/herself to maintain the server from off site, but it would be quite a breach if the administrator account were to be broken.
- **Password protect private chat room by default**. Some clients do not have the option to configure the group chat rooms that they create and so would not have the ability to choose extra protection for their private chat rooms. Some users also do not care or do not know how to configure their rooms and only use the default settings. Default password protection will ensure that passwords are used.

This is not an exhaustive list and this is not a list that would work in all environments, but it is one that applies in the MYSEA case where the goal is to guarantee a high level of assurance while still using COTS.

D. SUMMARY

This chapter discussed the initial code review for the jabberd14-1.6.0 server, brought up some technical issues that affect the future porting exercise and discussed configuration issues that could improve the security of the server. The next chapter discusses the future work needed to complete the porting of the jabberd14-1.6.0 server to MYSEA.

VI. FUTURE WORK AND CONCLUSION

This chapter discusses the tasks that still remain before the porting of jabberd14-1.6.0 server to STOP OS 7 can be considered complete. These tasks focus on testing and code review. The testing has two main parts and is addressed in the first section. The following section discusses the code review. This chapter also states the conclusion of this study in the third section.

A. TESTING

The testing that was described in Chapter IV used a client that is claimed to be compliant with the XMPP protocol and verified the basic functionality of the different Jabber server versions on different platforms. Those tests were sufficient for this study. However, additional compliance tests should be performed before any code modification is undertaken. Compliance testing requires that all of the MUST, MUST NOT, SHALL, SHALL NOT, SHOULD, and SHOULD NOT from RFC 3920, RFC 3921 and all of the XEP standards that the server claim to support be turned into test cases [2], [3], [27]. Those test cases can then be written in XML and bundled into a script that would report the returned values as the scripts runs. The expected returned values would be recorded before testing and compared against the actual values after testing. The script would run on a Telnet session. This kind of testing procedure could be inspired by the Tigase Test Suite [75]. Even though the suite only provides for testing servers that use the database option for storage, the suite is open source and could be modified. By using the code base of the Tigase Test Suite, not only compliance and functionality would be tested, but also performance and stability [40].

An older project that is still an active and part of the sourceforge repository is the Jabber Test Suite (also called JabberTest) [76]. For this suite, the focus is concentrated on performance analysis. The suite is composed of several tools that can automate various tests, such as “timing of account creating, message delivery, and concurrent session limits [77].” Use of this test suite would be most appropriate after the server has been modified because running the server with inetd might have an impact on performance.

After or along with testing the server for compliance, it is imperative to undertake a careful architectural analysis. The goal of the architectural analysis is to evaluate the use of the server in a multilevel environment and the kind of problems that will have to be dealt with when clients from different classifications join group chat rooms.

B. CODE REVIEW

The source code of the jabberd14-1.6.0 server as well as the external MUC module version 0.5.2, JUD module version 0.5, and all external libraries (expat-2.0.1, libidn-0.6.8, glib-2.0.7 and pkg-config-0.23) need to be reviewed in detail.

There are two main goals to the code review. The first is to understand the flow of the code and be able to target the areas where the code needs modification in order to comply with the STOP 7 environment. This tracing will also expose part of the code that might be superfluous or that the environment should not support for security or other reasons. Database storage might be an example. The second goal is to discover bugs, buffer overflows, memory leaks and the like in the software.

C. CONCLUSION

This study's goal was to perform preliminary work to facilitate porting XML-based chat to an MLS environment, such as MYSEA. The tasks performed included developing selection criteria, applying the criteria to select an open source Jabber server, experimenting and testing it on both Linux and XTS-400 platforms, and defining the technical and security issues to be addressed by subsequent porting work. The goal was accomplished. It was assessed that more thorough testing and code review is needed before porting the jabberd14-1.6.0 server to the MYSEA environment can be completed.

APPENDIX A: XMPP EXTENSIONS

This appendix provides a reference to the full description of each XEP (XMPP extension) mentioned in sub-section A 1 of chapter III [27]. The XEP standards process [78] defines different types of extensions. There are four types: Standard Track, historical, informational and procedural. Depending on their types, extensions have different development states. The Status column lists the current state of each XEP. The Active status is applicable to all extension types except for Standard Track. A proposed extension is assigned Active status after it has been reviewed and accepted by the XMPP Council. Draft status only applies to Standard Track extensions. A Draft extension means that the extension has "undergone extensive discussion and technical review" and is ready for implementation and deployment in operational environments [78]. Final status is also only for Standard Track extensions and is reached after the Draft state. It indicates that the extension has been coded in at least two separate ways and has been voted for acceptance by the XMPP Council.

XEP number	XEP name	Status	XEP Reference
0004	Data forms	Final	http://www.xmpp.org/extensions/xep-0004.html
0012	Last Activity	Draft	http://www.xmpp.org/extensions/xep-0012.html
0013	Flexible offline message retrieval	Draft	http://www.xmpp.org/extensions/xep-0013.html
0016	Privacy lists	Draft	http://www.xmpp.org/extensions/xep-0016.html
0020	Feature negotiation	Draft	http://www.xmpp.org/extensions/xep-0020.html
0030	Service discovery	Final	http://www.xmpp.org/extensions/xep-0030.html
0033	Extended stanza addressing	Draft	http://www.xmpp.org/extensions/xep-0033.html
0045	Multi-user chat	Draft	http://www.xmpp.org/extensions/xep-0045.html
0047	In-Band Bytestreams	Draft	http://www.xmpp.org/extensions/xep-0047.html
0048	Bookmarks	Draft	http://www.xmpp.org/extensions/xep-0048.html
0049	Private XML storage	Active	http://www.xmpp.org/extensions/xep-0049.html
0050	Ad-Hoc commands	Draft	http://www.xmpp.org/extensions/xep-0050.html
0054	vcard-temp	Active	http://www.xmpp.org/extensions/xep-0054.html
0059	Result set management	Draft	http://www.xmpp.org/extensions/xep-0059.html

0060	Publish-subscribe	Draft	http://www.xmpp.org/extensions/xep-0060.html
0065	SOCKS5 bytestreams	Draft	http://www.xmpp.org/extensions/xep-0065.html
0066	Out of band data	Draft	http://www.xmpp.org/extensions/xep-0066.html
0072	SOAP Over XMPP	Draft	http://www.xmpp.org/extensions/xep-0072.html
0077	In-Band Registration	Final	http://www.xmpp.org/extensions/xep-0077.html
0079	Advanced Message Processing	Draft	http://www.xmpp.org/extensions/xep-0079.html
0085	Chat state notifications	Draft	http://www.xmpp.org/extensions/xep-0085.html
0092	Software version	Draft	http://www.xmpp.org/extensions/xep-0092.html
0095	Stream initiation	Draft	http://www.xmpp.org/extensions/xep-0095.html
0096	File transfer	Draft	http://www.xmpp.org/extensions/xep-0096.html
0100	Gateway interaction	Active	http://www.xmpp.org/extensions/xep-0100.html
0106	JID Escaping	Draft	http://www.xmpp.org/extensions/xep-0106.html
0114	Jabber component protocol	Active	http://www.xmpp.org/extensions/xep-0114.html
0115	Entity capacities	Draft	http://www.xmpp.org/extensions/xep-0115.html
0124	Bi-directional-streams over synchronous HTTP (BOSH)	Draft	http://www.xmpp.org/extensions/xep-0124.html
0138	Stream compression	Draft	http://www.xmpp.org/extensions/xep-0138.html
0145	Annotations	Active	http://www.xmpp.org/extensions/xep-0145.html
0153	VCard-based avatars	Active	http://www.xmpp.org/extensions/xep-0153.html
0163	Personal eventing via pubsub	Draft	http://www.xmpp.org/extensions/xep-0163.html
0191	Simple communications blocking	Draft	http://www.xmpp.org/extensions/xep-0191.html
0199	XMPP ping	Draft	http://www.xmpp.org/extensions/xep-0199.html
0202	Entity time	Draft	http://www.xmpp.org/extensions/xep-0202.html
0203	Delayed delivery	Draft	http://www.xmpp.org/extensions/xep-0203.html
0206	XMPP over BOSH	Draft	http://www.xmpp.org/extensions/xep-0206.html

APPENDIX B: REDHAT 8 INSTALLATION INSTRUCTIONS

This appendix represents Van Emery's installation instructions reformatted. It is a step by step instruction guide for installing jabberd14-1.4.2 server that was used for the experiment on Red Hat 8 VM. This installation assumes either connection to the Internet in order to download source code or the use of the CD that contains all the necessary files (make sure the CD is automatically detected in the VM). Use the direction under "Using the Internet" or skip to the section called "Using the CD."

- Start VMWare, start a RedHat 8 machine, make sure the Ethernet option is bridge (Not NAT).
- Open a terminal window (under system tools)
- Create a directory called Downloads in root. (mkdir Downloads) and navigate to it (cd Downloads)

Using the Internet

- Open a browser and navigate to www.**vanemery**.com/Linux/Jabber/**jabberd**.html. On that page, scroll to the section called "Getting the Code."
- Click on jabber-1.4.2.tar.gz (674K),
- Select "save this file to disk", navigate to the Downloads directory, click "OK." Click "Save." Click "Close" when the downloading is done.
- Repeat the step above for the following files: mu-conference-0.5.2.tar.gz (46K), jud-0.5.tar.gz (5.3K), and vanjabfiles.tar.gz (7.5K)
- Right click on daemontools-0.70-5.i686.rpm and choose "Save Link Target As…" Click "Save". Click "Close."
- Close the browser.

Using the CD

- Open the root folder, open the Downloads folder
- Insert the CD in the machine
- Open the “Jabber Installation” folder on the CD
- Open the RedHat8 folder on the CD
- Copy all of the documents from the “RedHat8 modules” folder into the Downloads directory
- Eject the disk
- Close all windows except the terminal and navigate to /root/Downloads.

Common installation steps

- [root@im Downloads]# cp jabber-1.4.2.tar.gz /usr/local
- [root@im Downloads]# cd /usr/local
- [root@im local]# gunzip jabber-1.4.2.tar.gz
- [root@im local]# tar xvf jabber-1.4.2.tar
- [root@im local]# mv jabber-1.4.2 jabber
- [root@im local]# rm jabber-1.4.2.tar. At the question, type “y.”
- [root@im local]# cd jabber
- [root@im jabber]# ./configure --enable-ssl
- [root@im jabber]# make

Next, create a user named "jabber" and some directories with the proper permissions. You will note that we first make sure that UID 400 is not in use. Since the user "jabber" only exists for running jabberd, we are giving it a UID < 500.

- [root@im jabber]# grep :400: /etc/passwd
- [root@im jabber]# useradd -u 400 -M -d /usr/local/jabber jabber

- [root@im jabber]# mkdir --mode 0770 /etc/jabberd
- [root@im jabber]# mkdir --mode 0770 /var/run/jabberd
- [root@im jabber]# mkdir --mode 0770 /var/log/jabberd
- [root@im jabber]# chown jabber.jabber /etc/jabberd
- [root@im jabber]# chown jabber.jabber /var/run/jabberd
- [root@im jabber]# chown jabber.jabber /var/log/jabberd
- [root@im jabber]# cd /root/Downloads

Next, unpack, compile, and install the multiuser conference component.

- [root@im Downloads]# cp mu-conference-0.5.2.tar.gz /usr/local/jabber
- [root@im Downloads]# cd /usr/local/jabber
- [root@im jabber]# gunzip mu-conference-0.5.2.tar.gz
- [root@im jabber]# tar xvf mu-conference-0.5.2.tar
- [root@im jabber]# rm mu-conference-0.5.2.tar. At the question, type “y.”
- [root@im jabber]# cd mu-conf*
- [root@im mu-conference-0.5.2]# make
- [root@im mu-conference-0.5.2]# cd /root/Downloads

Next, unpack, compile, and install the JUD component.

- [root@im Downloads]# cp jud-0.5.tar.gz /usr/local/jabber
- [root@im Downloads]# cd /usr/local/jabber
- [root@im jabber]# gunzip jud-0.5.tar.gz
- [root@im jabber]# tar xvf jud-0.5.tar
- [root@im jabber]# mv jud-ansi-c jud-0.5

- [root@im jabber]# rm jud-0.5.tar At the question, type “y.”
- [root@im jabber]# cd jud-0.5
- [root@im jud-0.5]# make
- [root@im jud-0.5]# cd ..

Next, make a directory for the conference and set permissions.

- [root@im jabber]# mkdir /usr/local/jabber/spool/im.mysea.edu
- [root@im jabber]# cd /usr/local/
- [root@im local]# chown -R jabber.jabber /usr/local/jabber
- [root@im local]# chmod 0770 /usr/local/jabber
- [root@im local]# cd /root/Downloads

Next, install the daemontools package.

- [root@im Downloads]# rpm -ivh daemontools-0.70-5.i686.rpm
- [root@im Downloads]# rpm -q daemontools

Next, extract the scripts.

- [root@im Downloads]# cp vanjabfiles.tar.gz /usr/local/jabber
- [root@im Downloads]# cd /usr/local/jabber
- [root@im jabber]# gunzip vanjabfiles.tar.gz
- [root@im jabber]# tar xvf vanjabfiles.tar

Add the jabber TCP ports to /etc/services.

- [root@im jabber]# cp /etc
- [root@im etc]# gedit services &
- Add the following lines:

Jabber Ports

jabc 5222/tcp # Unencrypted jabber client-to-server

jabc-ssl 5223/tcp # SSL encrypted jabber client-to-server

jabs2s 5269/tcp # Jabber server-to-server

- Close and save the file
- [root@im etc]# cd /usr/local/jabber

Next, configure the server with the appropriate ownership and permissions.

- [root@im jabber]# cp –v jabber.xml.* /etc/jabberd
- [root@im jabber]# mv jabber.xml jabber.xml.org
- [root@im jabber]# cp -v jabber.xml.org /etc/jabberd
- [root@im jabber]# cd /etc/jabberd
- [root@im jabberd]# cp -v jabber.xml.c2s jabber.xml
- [root@im jabberd]# chown jabber.jabber *
- [root@im jabberd]# chmod 0660 *
- [root@im jabberd]# ls –l

The result of the ls –l command should be (other information might be different):

```
total 60
-rw-rw----    1 jabber   jabber       8380 Jul 27 01:48 jabber.xml
-rw-rw----    1 jabber   jabber       8380 Jul 22 07:27
jabber.xml.c2s
-rw-rw----    1 jabber   jabber      20667 Jul 19 18:49
jabber.xml.org
-rw-rw----    1 jabber   jabber       9620 Jul 27 01:48
jabber.xml.s2s
```

Next, setup the Linux run scripts so that jabberd can be started and stopped automatically.

The script is called jabberd.run (make sure you are root)

- [root@im jabberd]# cd /usr/local/jabber
- [root@im jabberd]# tar xvf vanjabfiles.tar
- [root@im jabberd]# cp jabberd.run /etc/init.d/jabberd
- [root@im Downloads]# cd /etc/init.d
- [root@im init.d]# chown root.root /etc/init.d/jabberd

- [root@im init.d]# chmod 0750 /etc/init.d/jabberd
- [root@im init.d]# ls -l jabberd

```
-rwxr-x---    1 root     root         2983 Jul 17 23:22 jabberd
```

- [root@im init.d]# chkconfig --add jabberd
- [root@im init.d]# chkconfig --list jabberd

```
jabberd         0:off   1:off   2:off   3:on    4:on    5:on    6:off
```

Next, setup automatic log rotation using jabberd.logrotate file. Global defaults are configured in /etc/logrotate.conf . Need to be root in order to do this.

- [root@im init.d]# cd /usr/local/jabber
- [root@im jabber]# cp jabberd.logrotate /etc/logrotate.d/jabberd
- [root@im Downloads]# cd /etc/logrotate.d
- [root@im logrotate.d]# chown root.root jabberd
- [root@im logrotate.d]# chmod 0660 jabberd
- [root@im logrotate.d]# ls -l jabberd

The result of the ls –l command should be:

```
-rw-rw----    1 root     root          217 Jul 17 23:35 jabberd
```

Next, create a user that will be the administrator of the group chat.

- [root@im logrotate.d]# cd /usr/local/jabber/spool
- [root@im spool]# mkdir conference.im.mysea.im[32]
- [root@im spool]# cd conference.im.mysea.im
- [root@im spool]# gedit rooms.xml &
- Click “Yes”, close the file.
- [root@im conference.im.mysea.edu]# cd /usr/local/jabber
- [root@im jabber]# cp jabadd /usr/local/sbin[33]
- [root@im jabber]# cd /usr/local/sbin/

[32] This folder will hold the MUC rooms. If a room is deleted, the server will have to be restarted in order for the change to take effect.

[33] All of the scripts can be copied into the /usr/local/sbin directory and all their permissions can be changed. Other script might need to have the FQDN variable edited also (jabdel)

- [root@im sbin]# chown root.root usr/local/sbin
- [root@im sbin]# chmod 0770 jabadd
- [root@im sbin]# gedit jabadd &
- Change the FQDN to “im.mysea.edu”, save and close
- [root@im sbin]# jabadd testuser

Give it a password and enter. Then, type n and enter. To verify that testuser.xml was created, cd to /usr/local/jabber/spool/im.mysea.edu and ls

Next, edit the configuration file

- cd /etc/jabberd
- ifconfig (record the IP address of the machine)
- gedit jabber.xml
- Change all of the instances of jabs.org with the name of the server (im.mysea.edu) and change all of the instances of 172.17.77.2 with the IP address of the server (found 2 steps back)
- Search for “thor” and replace it by testuser.
- Close and save jabber.xml file

Next, make a TLS/SSL certificate (self-signed) and place it in the /usr/local/jabber directory.

- [root@im jabberd]# cd /usr/local/jabber
- [root@im jabber]# /usr/bin/openssl req -new -x509 -newkey rsa:1024 -days 3650 -keyout privkey.pem -out key.pem
- Enter a pass phrase (you will be asked for it later) and all other information when prompted
- [root@im jabber]# /usr/bin/openssl rsa -in privkey.pem -out privkey.pem

- [root@im jabber]# cat privkey.pem >> key.pem
- [root@im jabber]# rm privkey.pem. type “y.”
- [root@im jabber]# chown jabber.jabber key.pem
- [root@im jabber]# chmod 0400 key.pem
- [root@im jabber]# ls -l key.pem

The result of the ls –l command should be:

```
-r--------    1 jabber   jabber       2274 Jul 17 16:53 key.pem
```

Next, start the service as a root user.

- [root@im jabber]# cd /etc
- [root@im etc]# gedit hosts
- Add the following line:

IP address put in jabber.xml file im.mysea.edu im

- Restart the computer. Log on, start a terminal session
- [root@im root]# /etc/init.d/jabberd start [34]

The result of the jabberd start command should be similar to[35]:

```
Starting jabberd: jabberd.
[root@im root]# 20030717T15:45:40: [notice] (-internal): initializing
server
```

Before connecting clients, change the hosts file on the client machine[36] to include the IP address of the server and its name (im.mysea.edu). Once you have changed the hosts file, restart the computer. The server requires connection on port 5223.

[34] To check the status of the service type /etc/init.d/jabberd status

To stop the service type: /etc/init.d/jabberd stop (or use Ctrl c key combination)

To check if all the ports are listening: netstat -ta

[35] Time stamp can not be exactly the same

[36] You need to be administrator or root to do that.

Component	File + Size	Notes
Psi Jabber Client	psi-0.9-1.i386.rpm (1.2M)	RPM, for RH 9 Linux x86
Psi Icon Sets	psi-iconsets-0.1-0.i386.rpm (120K)	RPM, for RH 9 Linux x86
QSSL Package	qssl-2.0-0.i386.rpm (277K)	RPM, RH 9, has SSL/TLS libs
Win32 Client	psi-0.9-setup.exe (2.8M)	Win32 installer, with SSL

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C: FEDORA 7 INSTALLATION INSTRUCTIONS

This appendix is the step by step instructions for installing Jabberd14-1.6.1.1 that was used for the experiment on Fedora 7 VM. This installation assumes that you are using the CD that contains all the necessary files (make sure the CD is automatically detected in the VM). The files on the CD are inside the folder called Fedora 7 modules and are as follow: adminguide_files, adminguide.html, jabberd14-1.6.1.1.tar.gz, mu-conference_0.7.tar.gz and patch.html

- Start VMWare, start a Fedora 7 machine, make sure the Ethernet option is bridge (Not NAT).
- Turned the firewall and SELinux off (System → Firewall and SELinux →select Disabled)

Change the hosts file.

- Open a terminal window (Applications → System Tools → Terminal),
- ifconfig (record the IP address of the machine)
- gedit /etc/hosts
- Add the following line under the 127.0.0.1 line:

The IP address retrieved two steps ago im.mysea.edu im

- Close and save the file
- Verify that the gnutls package is installed (Applications → Add/Remove Software→ List). If the gnutls-1.4.1 or better is not "checked", then check the box.
- Install pthreat: check both pth – 2.0.7-1 and pth-dev – 2.0.7-1.
- Install libtasn1: check the libtasn1 – 0.3.9-1, libtasn1-devel – 0.3.9-1 and libtasn1-tools – 0.3.9-1. Click on the "Apply" button,

"Continue," click "OK for dependencies, "OK." Close the Package Manager window when done.

- Double click the directory called "root's Home" on the Desktop
- Double click the directory called Download
- Insert the CD in the machine
- Double click "Computer" on the Desktop, double click on CD-RW/DVD+RW
- Double click the folder called jabber installation and the folder called called "Fedora 7 modules"
- Drag the contain of the folder called "Fedora 7 modules" into the Download directory
- Close all the windows, except the terminal window
- Eject the CD (right click on it)

Installing Jabberd

- In the terminal window, type: cd /root/Download
- gunzip jabber14-1.6.1.1.tar.gz
- tar xfv jabber14-1.6.1.1.tar
- cd jabberd14-1.6.1.1
- gedit configure.ac &
- Apply the patch below. Instead of having to type all those lines, you can copy and paste by opening the file called patch.html in the Downloads directory.

```
Modified:
   branches/RELEASE-1_6_1/configure.ac

Modified: branches/RELEASE-1_6_1/configure.ac
===================================================================
=======
```

```
--- branches/RELEASE-1_6_1/configure.ac       (original)
+++ branches/RELEASE-1_6_1/configure.ac       Wed Aug  1 11:48:38 2007
@@ -6,7 +6,7 @@


 AC_INIT(jabberd/jabberd.h)
-AM_INIT_AUTOMAKE(jabberd14,1.6.1.1)
+AM_INIT_AUTOMAKE(jabberd14,1.6.1.1-p1)
 AM_CONFIG_HEADER(config.h)
 AC_LANG(C++)

@@ -272,6 +272,15 @@
     AC_DEFINE(HAVE_GNUTLS_EXTRA,,[if GnuTLS-extra should be compiled
in])
 fi

+dnl check for libtasn
+PKG_CHECK_MODULES(LIBTASN, libtasn1 >= 0.3.0, haslibtasn=yes,
haslibtasn=no)
+if test $haslibtasn = "yes" ; then
+    CPPFLAGS="$CPPFLAGS $LIBTASN_CFLAGS"
+    LDFLAGS="$LDFLAGS $LIBTASN_LIBS"
+else
+    AC_MSG_ERROR([Couldn't find required libtasn1 installation])
+fi
+
 dnl check for libidn
 AC_ARG_WITH(libidn, AC_HELP_STRING([--with-libidn=DIR],
            [Where to find libidn (required)]),
=====================================================================
```

- Save and close the file
- Close the patch.html file if you used it
- ./configure (need to be located in the jabberd directory)
- make install
- cd ..
- rm jabber14-1.6.1.1.tar. At the prompt type “y”
- gunzip mu-conference_0.7.tar.gz
- tar xfv mu-conference_0.7.tar.gz
- cd mu-conference_0.7
- make

Configure the jabber server

- cd /usr/local/etc
- gedit jabber.xml
- Change all of the instances of localhost with im.mysea.edu
- Search for <dhparams> and comment that line out.
- Search for “<service id=”inject” and comment out all the lines until “</service>”
- The flat file option needs to be activated[37]. Locate the <xdb> tag. Remove all of the instructions between the <xdb> and </xdb> tags, including those tags. Add the following text:

```
<xdb id="xdb">
    <host/>
    <ns/>
    <load>
    <xdb_file>/usr/lib/libjabberdxdbfile.so</xdb_file>
    </load>
    <xdb_file xmlns="jabber:config:xdb_file">
    <spool><jabberd:cmdline
flag='s'>/usr/local/var/spool/jabberd</jabberd:cmdline></spool>
     <timeout>3600</timeout>
      <sizelimit>500000</sizelimit>
    </xdb_file>
  </xdb>
```

- To implement MUC, search for the word browse and include the following text:

```
<item category="conference" type="public" jid="conference.im.mysea.edu"
name="Public Conferencing">
    <ns>http://jabber.org/protocol/muc</ns>
  </item>
```

- Search the word service and add the following code before “<service id=“c2s”>” tag:

```
<service id="muclinker">
```

[37] The instructions are in the README.filespool in the directory where jabberd14 resides.

```
  <host>conference.im.mysea.edu </host>
  <accept>
    <ip>IP of the server[38]</ip>
    <port>31518</port>
    <secret>secret</secret>
  </accept>
</service>
```

- Close and save the file
- cd /root/Download
- cd mu-conference_0.7
- cp mu-default.xml muc.xml
- gedit muc.xml
- Replace all instances of localhost to im.mysea.edu except the one between the <ip> tag where you need to type the IP address of the server. Change the port to 31518.
- Close and save the file
- mkdir syslogs
- mkdir spool
- cd spool
- mkdir chat.im.mysea.edu
- cd ..
- cd ..
- cd jabberd14-1.6.1.1/xdb_file/.libs
- cp libjabberdxdbfile.so libjabberdxdbfile.la libjabberdxdbfile.lai libjabberdxdbfile.so.2 libjabberdxdbfile.so.2.0.0 libjabberdxdbfile.so.2.0.0T xdb_file.o xdbfiletool /usr/lib
- Restart the computer

[38] Note that this must be the IP (of the form 0.0.0.0)

Start the service,

- Open a terminal window
- cd /root/Download/jabberd14-1.6.1.1
- jabberd/jabberd[39]
- open another terminal
- src/mu-conference –c muc.xml &[40]

Now, refer to the Appendix B in order to use clients to test the server.

[39] In order to start the service in debugging mode type jabberd/jabberd –D. It is also helpful to pipe it to a file in order to read the debugging as it goes much too fast on the screen `>/tmp/debug 2>&1`.

[40] In order to run in debugging mode, exclude the & at the end of the command.

APPENDIX D: XTS-400 INSTALLATION INSTRUCTIONS

This appendix is the step by step instructions for installing Jabberd14-1.6.0 that was used for the experiment on XTS-400 STOP 7.0 beta 1. This installation assumes that you are using the CD that contains all the necessary files. Before using the XTS-400, perform the steps in Appendix B of Networking Configuration of [79].

- Log in[41] (change your privilege level to syslo:admin.admin by typing sec_label –p –l :syslo)
- vi /xts/etc/hosts
- Add the following line:
- 172.20.108.51 thesis.nps.edu jabber
- Close and save the file

Getting the source code from the CD

- Load the CD
- mkdir download
- sec_label –p –l :sys
- sec_label –l :syslo /dev/sdb
- sec_label –p –l :syslo
- mount –r /dev/sdb /mnt/cdrom[42]
- cp –r /mnt/cdrom/Jabber Installation/XTS400 modules/* /download
- umount /mnt/cdrom
- sec_label –l :sys /dev/sdb

[41] Make sure you are syslo:admin.admin (check by typing sec_label –p). If you are not, then execute this command: sec_label –p –l :syslo

[42] The system that those instructions were tested on had the /mnt/cdrom directory path already made.

- Eject the CD

Installing the pkg-config dependency

- cd /download/
- gunzip pkg-config-0.23.tar.gz
- tar xvf pkg-config-0.23.tar
- cd pkg-config-0.23
- ./configure
- vi poptconfig.c and comment out line 171[43]
- make install
- cp /usr/local/bin/pkg-config /xts/bin
- cd ..
- rm pkg-config-0.23.tar. Type “y” when prompted.

Installing the expat dependency

- gunzip expat-2.0.1.tar.gz
- tar xvf expat-2.0.1.tar
- cd expat-2.0.1
- ./configure
- make install
- cd ..
- rm expat-2.0.1.tar. Type “y” when prompted.

Installing the libidn dependency

- gunzip libidn-0.6.8.tar.gz
- tar xvf libidn-0.6.8.tar
- cd libidn-0.6.8
- ./configure
- make install
- cd ..
- rm libidn-0.6.8.tar. Type “y” when prompted.

[43] You can also search for the word getuid in the file. The first instance of it is line 171. This editing has to be done because the method in that line use the unistd.h and it is not implemented on STOP 7.

Installing the glib dependency

- gunzip glib-2.0.7.tar.gz
- tar xvf glib-2.0.7.tar
- cd glib-2.0.7.
- ./configure
- cp /usr/local/lib libglib-2.0.la libglib-2.0.so libglib-2.0.so.0 libglib-2.0.so.0.0.7 /xts/lib
- make install
- rm glib-2.0.7.tar. Type “y” when prompted.

Preparing to install Jabberd14

- cd /usr/local
- mkdir var
- cd var
- mkdir spool
- mkdir log
- cd spool
- mkdir jabberd
- cd jabberd
- mkdir thesis.nps.edu
- cd ..
- cd ..
- cd log
- mkdir jabberd

Installing Jabberd14

- cd /download
- gunzip jabberd14-1.6.0.tar.gz
- tar xvf jabberd14-1.6.0.tar
- cd jabberd14-1.6.0/jabberd
- vi jabberd.cc
- Commented out:

#include <pwd.h> , #include <grp.h> and from line 266 to line 283.

- Close and save the file
- cd ..
- ./configure
- make install
- Move the libraries starting with libidn.a until libjabberdxdbsql.so.1.0.0 from /usr/local/lib to /xts/lib using the cp command
- cd /download
- rm jabberd14-1.6.0.tar. Type “y” when prompted.
- cd jabberd14-1.6.0
- mkdir spool
- cd spool
- mkdir thesis.np.edu
- cd /download

Configuring the server

- cd /usr/local/etc
- vi jabber.xml
- Change all of the instances of localhost with the name of the server (thesis.nps.edu) ,
- The flat file option needs to be activated[44]. Locate the <xdb> tag. Remove all of the instructions between the <xdb> and </xdb> tags.

[44] The instructions are in the README.filespool in the directory where jabberd14 resides.

Add the following text:

```
<xdb id="xdb">
    <host/>
    <ns/>
    <load>
    <xdb_file>/usr/local/lib/libjabberdxdbfile.so</xdb_file>
    </load>
    <xdb_file xmlns="jabber:config:xdb_file">
    <spool><jabberd:cmdline
flag='s'>/usr/local/var/spool/jabberd/thesis.nps.edu</jabberd:cmdline><
/spool>
     <timeout>3600</timeout>
      <sizelimit>500000</sizelimit>
    </xdb_file>
  </xdb>
```

- To implement MUC, search for the word “browse” and include the following text:

```
<item category="conference" type="public"
jid="conference.thesis.nps.edu" name="Public Conferencing">
    <ns>http://jabber.org/protocol/muc</ns>
  </item>
```

- Search the word “service” and add the following code before “<service id=“c2s”>” tag:

```
<service id="muclinker">
    <host>conference.thesis.nps.edu</host>
    <accept>
      <ip>172.20.108.51</ip>
      <port>31518</port>
      <secret>secret</secret>
    </accept>
  </service>
```

- Search for the sequence log id and comment out the 6 lines that include <syslog>locall0</syslog> and uncomment the alternative that includes <file>/usr/local/var/log/jabberd/error.log</file> below.

Installing MUC

- cd /download
- gunzip mu-conference_0.7.tar.gz
- tar xvf mu-conference_0.7.tar
- cd mu-conference_0.7
- make
- cd /download
- rm mu-conference_0.7.tar. Type “y” when prompted.
- cd mu-conference_0.7
- mkdir spool
- mkdir syslogs
- cd spool
- mkdir thesis.nps.edu

Configuring muc

- cd /download/mu-conference_0.7
- cp muc-default.xml muc.xml
- vi muc.xml
- Change all of the localhost for thesis.nps.edu and IP addresses to 172.20.108.51. Change the port to 31518. Save and close the file

Installing jud

- cd /download
- gunzip jud-0.5.tar.gz
- cp jud-0.5.tar /download/jabberd14.1.6.0
- cp platform-settings /download/jabberd14.1.6.0
- cd jabberd14-1.6.0
- tar xvf jud-0.5.tar
- make
- rm jud-0.5.tar. Type “y” when prompted.

Configuring jud

- vi /usr/local/etc/jabber.xml

- search for word browse and add the following lines

```
<service type="jud" jid="jud.thesis.nps.edu" name="thesis.nps.edu User
Directory">
      <ns>jabber:iq:search</ns>
      <ns>jabber:iq:register</ns>
    </service>
```

- search for word service and add the following lines

```
<service id="jud">
    <host>jud.thesis.nps.edu</host>
    <load><jud>./jud/jud.so</jud></load>
    <jud xmlns="jabber:config:jud">
    <vCard>
      <FN>User Directory on thesis.nps.edu</FN>
      <DESC>This provides a simple user directory service.</DESC>
      <URL>http://jud.thesis.nps.edu/</URL>
    </vCard>
    </jud>
  </service>
```

Copying things around

- cp /download/jabberd14-1.6.0/jsm/.libs/libjabberdsm.so /usr/local/lib

- cp /download/jabberd14-1.6.0/xdb_file/.libs/libjabberdxdbfile.so /usr/local/lib

- cp /download/jabberd14-1.6.0/pthsock/.libs/libjabberdpthsock.so /usr/local/lib

- cp /download/jabberd14-1.6.0/dnsrv/.libs/libjabberddnsrv.so /usr/local/lib

- cp /download/jabberd14-1.6.0/dialback/.libs/libjabberddialback.so /usr/local/lib

- cp /usr/local/lib/libgthread-2.0.la libgthread-2.0.so libgthread-2.0.so.0 libgthread-2.0.so.0.0.7 libexpat.so.1 libexpat.so.1.5.2 /xts/lib

To start the services

- cd /download/jabberd14-1.6.0
- jabberd/jabberd[45]
- Press the alt F2 combination to get another screen, log in
- src/mu-conference –c muc.xml &[46]

Now, refer to the Appendix B in order to use clients to test the server.

[45] In order to start the service in debugging mode type jabberd/jabberd –D. `>/tmp/debug 2>&1`.

[46] In order to run in debugging mode, exclude the & at the end of the command.

APPENDIX E: INSTALLATION INSTRUCTIONS FOR GAJIM ON WINDOWS XP PROFESSIONAL

This appendix gives the step by step process to install and run the Gajim version 0.11.4 client on Windows XP Professional version 5.1.2600 that was used for the testing sub-section of Chapter IV. The machine used below is a laptop Inspiron 5150.

Power on the Windows XP Professional machine and log in as a valid user with enough privilege to install software.

Installing Gajim

- Insert the CD in the drive and browse the CD
- Double click the Jabber Installation folder
- Click and drag the icon named gajim-0.11.4+gtd-4.exe onto the Desktop. Select "Copy here" if necessary.
- Close the browsing window and eject the CD
- Double click the icon named gajim-0.11.4+gtd-4.exe
- On the Welcome to the Gajim Setup Wizard, click Next
- On Select destination Location, click Next
- Make sure that Full installation is selected in the drop down box and click Next
- On Select start menu folder, click Next
- Make sure the "Create a desktop icon" is selected and click Next
- Click Install
- Deselect the box "Launch application" and click Finish
- Right click on gajim-0.11.4+gtd-4.exe and select "Delete", click Yes when prompted

Setting up the Environment

On the server machine

- Open a terminal window and type "ifconfig"
- Record the IP address for later use
- Type "exit"

On the client machine (Windows)

- On the Desktop, double click My Computer
- Double click "Local Disk (C:)" (if the files are hidden, click the "show the content of this folder)
- Double click the "WINDOWS" folder
- Double click the "system32" folder
- Double click the "drivers" folder
- Double click the "etc" folder

- Right click the “hosts” file and choose open
- Choose Notepad from the list of applications offered and click OK
- Place your cursor at the end of the “127.0.0.1 localhost” and press the Enter key
- Type the ip address of the server recorded in “**On the server machine**” part above, enter a tab and type the name of the server (for example: 172.20.108.51 thesis.nps.edu jabber)
- Close and save the hosts file
- Close the browsing window
- Click Start → shut down
- Click Restart
- Restart the computer

Starting Gajim

- Click Start → Program → Gajim → Gajim (the Gajim Account Creation Wizard will open if an account was not already created.) Click Cancel if the Wizard is open, and proceed to Appendix F.

APPENDIX F: TEST PROCEDURES

The appendix contains the step by step instructions that were taken to test the jabberd14 server on the Fedora 7 and STOP 7 beta operating systems. All of the tests below use the Gajim client and assume that jabberd14-1.6.1 server and jabberd14-1.6.0 server is used on Fedora 7 or XTS-400 STOP 7 beta, respectively. The installation instructions for setting up these platforms are in Appendix C and D, respectively. Because the JUD module was not installed in the Fedora experiment, hence it was not tested on that platform. This document is formatted with im.mysea.edu for the name of the server, but can be used for the STOP 7 by replacing the server name by thesis.nps.edu.

A. Account management (A1 through A26)

A1 Register account with a valid JID

1. Start Gajim
2. Edit → Accounts → New
3. Select the radio button "I want to register to a new account"
4. Click Forward
5. Username: testuser1
6. Server: im.mysea.edu
7. Password: 1234
8. Retype Password: 1234
9. Click Forward (The client will display a search bar and an announcement box)
10. Click Finish
11. Enter some personal information and click OK[47]
12. Close the account dialogue box
13. Actions → Quit[48]

A2 Register account with an invalid JID

1. Start Gajim
2. Edit → Accounts → New
3. Select the radio button "I want to register to a new account"
4. Click Forward
5. Username: test&user

[47] A welcome message is delivered by the server and make the client blink.

[48] There might be a message regarding a history log file that it is making, click OK

6. Server: im.mysea.edu
7. Password: 1234
8. Retype Password: 1234
9. Click Forward (The client will display an alert box labeled: “Invalid Jabber ID”)
10. Click OK
11. Change the user name to be testuser2
12. Change the server to be im.myse.edu (if testing the thesis.nps.edu, change the name to thesis.ns.edu)
13. Click Forward (After searching for a few seconds, the client will display an alert box indicating that “An error occurred during account creating: Could not connect to im.myse.edu”
14. Click Cancel
15. Close the account dialogue box
16. Actions → Quit

A3 Register account with a password

1. Start Gajim
2. Edit → Accounts → New
3. Select the radio button “I want to register to a new account”
4. Click Forward
5. Username: testuser12
6. Server: im.mysea.edu
7. Password: 1234
8. Retype Password: 1234
9. Click Forward (The client will display a search bar and an announcement box informing of the successful account creation)
10. Click Finish
11. Enter some personal information and click OK[49]
12. Close the account dialogue box
13. Actions → Quit

A4 Register account without a password

1. Start Gajim
2. Edit → Accounts → New
3. Select the radio button “I want to register to a new account”
4. Click Forward
5. Username: testuser123
6. Server: im.mysea.edu
7. Password:
8. Retype Password:
9. Click Forward (The client will display an alert box “Invalid password : You must enter a password for the new account”)
10. Click OK

[49] A welcome message is delivered by the server and make the client blink.

11. Click Cancel
12. Close the account dialogue box
13. Actions → Quit

A5 Connect to server using SSL (legacy on port 5223)

1. Start Gajim
2. Right click on the first account that was registered (testuser1) → status → Offline
3. Edit → Accounts
4. Select the testuser1 account
5. Click Modify
6. Click the Connection tab
7. Check the checkbox "use SSL (legacy)[50]"
8. Click Save
9. Close
10. Right click on the testuser1 in the Gajim's main window → status → available (this will take a few seconds until a message comes back: "Cannot connect to im.mysea.edu")
11. Close the account dialogue box
12. Actions → Quit

A6 Connect to server without SSL (port 5222)

1. Start Gajim
2. Right click on the first account that was registered (testuser1) → status → Offline
3. Edit → Accounts
4. Select the testuser1 account
5. Click Modify
6. Click the Connection tab
7. Deselect the checkbox "use SSL (legacy)"
8. Click Save
9. Close
10. Right click on the testuser1 in the Gajim's main window → status → available (the icon will turn green again)
11. Actions → Quit

A7 Connect to server with SSL (port 5222)

The Gajim client does not allow for this configuration explicitly. The server has to enforce that setting.

A8 Connect with a password (in order to check this case, extra steps in the account screen need to be done)

1. Start Gajim
2. Edit → Accounts

[50] Note that the port in the gray box at the bottom of the dialogue box changed to 5223.

3. Select the testuser1 account
4. Click Modify
5. Uncheck the “Save password” checkbox
6. Click Save
7. Click Close (make sure that the user is offline)
8. Right click on the user → status → available (a dialog box will appear asking for a password)
9. Enter 1234
10. Click OK (the icon will turn green again)
11. Actions → Quit

A9 Connect without a password (in order to check this case, extra steps in the account screen need to be done)

1. Start Gajim
2. Edit → Accounts
3. Select the testuser1 account
4. Click Modify
5. Uncheck the “Save password” checkbox
6. Click Save
7. Click Close (make sure that the user is offline)
8. Right click on the user → status → available (a dialog box will appear asking for a password)
9. Click OK
10. An alert window will come up with the following message: “Authentication failed with im.mysea.edu”
11. Click OK
12. Actions → Quit

A10 Edit personal information

1. Start Gajim
2. Right click on the user
3. Select Modify account
4. Click on the Personal Information tab
5. Click on “Edit Personal Information…”
6. Enter your name and all other information desired
7. Click OK
8. Actions → Quit

A11 Add a contact that has a valid JID (this assumes that another client has successfully registered with the server on a different machine. Refer to A1 to register a user)

1. Start Gajim
2. Right click on the user → Add contact…
3. User ID: testuser1@im.mysea.edu
4. Click Add (the testuser1 screen will bring a window with the message “I would like to add you to my contact list”)

5. Click Authorize (Authorization accepted window will appear once the authorized button is pressed)[51]
6. Click OK on all of the dialogue boxes so that both users are added in each other's contact list.
7. Actions → Quit

A12 Add a contact that has an invalid JID (same assumptions as for A11)

1. Start Gajim
2. Right click on the user → Add contact…
3. User ID: test&user1@im.mysea.edu[52]
4. Add (a window Invalid User ID will appear on the screen)
5. Click OK
6. Click Cancel
7. Actions → Quit

A13 Delete contacts (buddies) from the contact list (same assumptions as for A11)

1. Start Gajim
2. Right click on the contact → Remove from Roster (a window displaying the following message appears on the screen: Contact "usertest1" will be removed from your roster)
3. Click OK

A14 Forbid contacts to see status (same assumptions as for A11)

1. Start Gajim
2. Right click on the contact → Subscription → Forbid him/her to see my status (a window displaying: Authorization has been removed will appear on the screen
3. Click OK (the user that was removed get a dialogue box signaling that it was removed from the contact's list. Click OK)

A15 Delete an account on client only (this test assumes that there is an account properly registered with the server)

1. Start Gajim
2. Edit → Accounts
3. Select testuser1
4. Click on remove[53] (an alert window opens and requires the client to choose between: "remove account only from Gajim or Remove account from Gajim and from server")

[51] Gajim has this feature that as soon as a user request to add a contact, than it automatically asks the other contact if it wants to subscribe to that user's roster.

[52] Giving an invalid server name returns the user with a question mark and a "server-unavailable" under the user icon.

[53] If the user is online and there are chat conversations open, there might be different dialogue box open.

5. Click Remove (because the client was connected with the server, an additional dialogue box appear. Click OK)

A16 Delete an account on both the client and the server (this test assumes that there is an account properly registered with the server)

1. Start Gajim
2. Edit → Accounts
3. Select testuser1
4. Click on remove[54] (an alert window opens and requires the client to choose between: "remove account only from Gajim or Remove account from Gajim and from server")
5. Select the second radio button (if the client was online, then there will be an additional dialogue box. Click OK)
6. Click Remove
7. Click Close
8. Actions → Quit

A17 Connect to the server with a deleted account after test A15 is performed

1. Start Gajim
2. Edit → Accounts
3. Click New
4. Click Forward
5. Username: testuser1
6. Server: im.mysea.edu
7. Password: 1234
8. Click Forward
9. Click Finish
10. Click Close
11. Actions → Quit

A18 Connect to the server with a deleted account after test A16 is performed (if there are no more users in this client, Gajim will try to connect or register. Click Cancel)

1. Start Gajim
2. Edit → Accounts
3. Click New
4. Click Forward
5. Username: testuser1
6. Server: im.mysea.edu
7. Password: 1234
8. Click Forward
9. Click Finish[55]

[54] If the user is online and there are chat conversations open, there might be a different dialogue box open. Also if the account was just open and the server welcome message was not read, Gajim will not allow for the account to be removed.

[55] Even though the user appears in the client main window, it is unable to connect.

10. Click Close
11. Actions → Quit

A19 Send server message to all users connected to the server)

1. Start Gajim
2. Actions → Advanced → Administrator → Send Server Message
3. Click Send and Close (the client should indicate that it has receive a message and that message should let it know that this action is "not-allowed" – need to double click on the message first)
4. Close the window
5. Actions → Quit

A20 Retrieve other users' contact information (this test assumes that at least one contact is in the contact list)

1. Start Gajim
2. Right click on a user in the contact list → information
3. Click the different tab to see what information was entered here
4. Click Close
5. Actions → Quit

A21 Transfer files between clients

1. Start Gajim
2. Right click on the contact → Send file
3. Browse and choose a file
4. Click Send
5. On the recipient of those files, an alert window will ask if the file should be accepted. If the answer is positive, then the client can save the document. If cancel is selected, the message is discarded[56].
6. Actions → Quit

A22 Retrieve communication history information

1. Start Gajim
2. Right click on the contact → History (a dialogue box appears and searching through the conversation is possible)
3. Click Close
4. Actions → Quit

A23 Deny a contact from viewing your status for the current session

1. Start Gajim
2. Actions → Advanced → Privacy Lists
3. In the empty textfield, type the name of the list (private list)
4. Click New
5. Click Add

[56] The sender is notified when the recipient does not allow the file transfer.

6. Select Deny
7. In the JabberID textfield type testuser2@im.mysea.edu
8. Check the “to view my status” textbox
9. Click Save
10. Click Close
11. Select “Active for this session”
12. Click Close
13. Change the status of the testuser1
14. The status is still updated
15. Actions → Advanced → Privacy Lists
16. Select private list
17. Click Delete
18. Click Close
19. Actions → Quit

A24 Deny a contact from sending you a message for the current session

1. Start Gajim
2. Actions → Advanced → Privacy Lists
3. In the empty textfield, type the name of the list (private list)
4. Click New
5. Click Add
6. Select Deny
7. In the JabberID textfield type testuser2@im.mysea.edu
8. Check the “to send me messages” textbox
9. Click Save
10. Click Close
11. Select “Active on each startup”
12. Click Close
13. Right click on the testuser1 from testuser2 machine and select Start Chat
14. Type some test in the window and press the enter key

A25 Register with an account that was deleted from both the server and the client (this test assumes that there is a properly registered account already in the client. Refer to A1 to set up a valid account)

1. Start Gajim
2. Edit → Accounts
3. Click on testuser1 (highlights it)
4. Click Remove
5. Choose “Remove account from Gajim and from server”
6. Click Remove (if the account was connected, then an additional dialogue box will appear warning that removing the user will disconnect it)
7. Click OK
8. Click New
9. Choose “I want to register for a new account”
10. Click Forward

11. Username: testuser1
12. Server: im.mysea.edu
13. Password: 1234
14. Retype Password: 1234
15. Click Forward (A dialogue box appears with the words “Conflict”)
16. Click Cancel
17. Click Close on the Accounts dialogue box
18. Actions → Quit

A26 Register with an account that was deleted from the client only (this test assumes that there is a properly registered account already in the client. Refer to A1 to set up a valid account)

1. Start Gajim
2. Edit → Accounts
3. Click on testuser1 (highlights it)
4. Click Remove
5. Choose “Remove account only from Gajim”
6. Click Remove (if the account was connected, then an additional dialogue box will appear warning that removing the user will disconnect it)
7. Click OK
8. Click New
9. Choose “I want to register for a new account”
10. Click Forward
11. Username: testuser1
12. Server: im.mysea.edu
13. Password: 1234
14. Retype Password: 1234
15. Click Forward (A dialogue box appears with the words “Conflict”)
16. Click Cancel
17. Click Close on the Accounts dialogue box
18. Actions → Quit

B. Presence management (B1 through B3) – All of those tests assumes that the user is connected properly to the server.

B1 Change presence to Available (this test assumes that the current presence is a presence other than available)

1. Right click on testuser1 → Status → Available (the icon reflect the status by showing a green icon)

B2 Change presence to Free for Chat and Away and Not Available, and Busy, and Invisible and Offline (this test assumes that the current presence is a presence other than Free for Chat or Away, or Not Available, or Busy, or Invisible or Offline)

1. Right click on testuser1 → Status → Free For Chat or Away or Not Available, or Busy, or Invisible or Offline
2. Type a message[57] (the icon reflect the status by showing a different icon that the one that was displayed before)
3. Click OK

B3 Display offline contacts

1. View → Show Offline Contacts

C. Instant Messaging (IM) management (C1 through C7)

C1 Start a chat with someone registered with the server on the contact list (this test assumes that a valid user has been added to the contact list)

1. Right click on the contact → Start Chat
2. Type some text in the window and press enter (the recipient receives the typed message)
3. Close the window

C2 Start a chat with someone registered with the server not in the contact list (this test assumes that there is another user on another machine that has not been already added in the contact list, but that is properly registered with the server)

1. Actions → Start Chat
2. Enter testuser1@im.mysea.edu
3. Click OK
4. Type some text in the window and press enter (the recipient receives the typed message)
5. Close the windows on both clients

C3 Start a chat with someone by mistyping their JID (user ID)

1. Actions → Start Chat
2. Enter test&user1@im.mysea.edu
3. Click OK (An alert window with Invalid JID pop up on the screen)
4. Click OK

C4 Start a chat with a contact by mistyping their JID (server name)

1. Actions → Start Chat
2. Enter testuser1@im.myea.edu
3. Click OK
4. Type some text in the window and press enter (The chat window specifies that the service is unavailable)

[57] The message typed is displayed under the user presence to all of those that subscribe to its presence.

C5 Have multiple chat sessions with different valid contacts (this test assumes that two contacts have been added to the list[58])

1. Right click on the contact → Start Chat
2. Type some text in the window and press enter (the recipient receives the typed message)
3. Repeat step 1 and 2 with another contact in the list (the window at the initiator of the message will have tabs for each user name).
4. Close the window when done

C6 Have multiple chat session with the same valid contact

1. Right click on the contact → Start Chat
2. Type some text in the window and press enter (the recipient receives the typed message)
3. Repeat step 1 and 2 with the same contact in the list[59]
4. Close the window when done

C7 Add Emoticons to chat messages

1. Right click on the contact → Start Chat
2. Click on the face icon button (bottom left of the window) and choose an icon
3. Type some text in the window and press enter (the recipient receives the typed message including the icon[60])
4. Close the window when done

D. Multi-User Chat (MUC) management (D1 through D15)

D1 Create a default[61] group chat with a valid JID

1. Start Gajim
2. Right click on a user → Group Chat → Join New Group Chat
3. Nickname: testuser1
4. Room: private@conference.im.mysea.edu
5. Click Join
6. Click OK
7. Connect to the private group chat room with another client using the steps 1-5
8. Type some text in both windows and press enter
9. Close the group chat room windows on both clients

D2 Create a default group chat with an invalid JID

1. Right click on a user → Group Chat → Join New Group Chat[62]

---

[58] Creating two separate accounts in one of the client will substitute for a separate machine.

[59] The client engine or the server recognizes that the two participants are already in contact with instant messaging and does not allow the creation of another window or tab.

[60] Icon shows if the client program supports icons otherwise the text representation of the icon is displayed

[61] Default indicates that no special configuration were done

[62] If the tests are performed in order, the textfields will be already filled with the information from the previous test.

2. Nickname: testuser1
3. Room: priv&ate@conference.im.mysea.edu
4. Click Join (An alert window with the words “invalid group chat Jabber ID” popped on the screen)
5. Click OK
6. Click Cancel

D3. Join a private group chat with proper credentials (in this case the group chat requires a password, but the user knows it) (this test assumes that a group chat that requires a password to join has been created.) In order to create a room with a password, refer to test D1 steps 1-6 and check the “A password is required to enter” and enter the password of 1234

1. Right click on a user → Group Chat → Join New Group Chat
2. Nickname: testuser1
3. Room: private@conference.im.mysea.edu
4. Password: 1234
5. Click Join
6. Type some text and enter
7. Close the group chat room windows in both clients

D4. Join a private group chat without membership (in this case the group chat has a member list that enforces access) (this test assumes that a group chat that was configured to require an invitation in order to join has been created.) In order to create a room that require an invitation, refer to test D1 steps 1-6 and check the “An invitation is required to Enter” and click OK.

1. Right click on a user → Group Chat → Join New Group Chat
2. Nickname: testuser1
3. Room: private@conference.im.mysea.edu
4. Click Join (An alert window with the words “Unable to join group chat: You are not in the members list” popped on the screen)
5. Click OK (a chat window will appear on the screen, but it is blank and the part where the user can type is disabled.)

D5. Join a private group chat without the necessary credentials (in this case the group chat requires a password) (this test assumes that a group chat that requires a password to join has been created.) In order to create a room with a password, refer to test D1 steps 1-6 and check the “A password is required to enter” and enter a password

1. Right click on a user → Group Chat → Join New Group Chat
1. Nickname: testuser1
2. Room: private@conference.im.mysea.edu
3. Click Join (An alert window with the words “Unable to join group chat: A password is required” popped on the screen)
4. Click OK (a chat window will appear on the screen, but it is blank and the part where the user can type is disabled.)

D6. Join a publicly open group chat (this test assumes that an open group chat was created.) In order to create an open group chat room refers to test D1 steps 1-6.

1. Right click on a user → Group Chat → Join New Group Chat
2. Nickname: testuser1
3. Room: public@conference.im.mysea.edu
4. Click Join
5. Type some text and press enter
6. In another client, repeat steps 1-5
7. Close the group chat room windows in both clients

D7. Configure a group chat room after it is created (as its creator)

1. Right click on a user→ Group Chat → Join New Group Chat
2. Nickname: testuser1
3. Room: private@conference.im.mysea.edu
4. Click Join
5. Click the Actions button (left of the send button) → Configure Room
6. Make all of the adjustment necessary
7. Click OK
8. Close the window

D8. Configure the room after it is created as a normal user (this test assumes that a public chat room was created. Refer to D1 to create a default chat room)

1. Right Click on testuser1→ Group Chat → Join New Group Chat
2. Nickname: testuser1
3. Room: private@conference.im.mysea.edu
4. Click Join
5. Click the Actions button (left of the send button) → Configure Room (the configuration option is grayed out)
6. Close the window

D9. Invite contact to join private group chat rooms (this test assumes that both a private room requiring an invitation called private was created – refer to D1 steps 1-6 – and a contact was added to the contact list – refer to A11)

1. Right click on contact → Invite to… → Private[63]
2. In the receiving party, open the message announcing that testuser1 has invited you to the group chat private@conference.im.mysea.edu
3. Click accept
4. Click Join
5. Type some text and click enter

D10. Communicating in authorized group chat rooms

1. Right click on a user → Join New Group Chat → either create a new group chat, or use the default from previous tests

[63] Note that the name *Private* is the name of the room that was created prior.

2. Click Join
3. Type test and enter once done
4. Repeat bullet 1 through 3 for another user (the window will replicate the text typed in the different windows and indicate when users log in)

D11. Ban a user from a group chat room when having the role of moderator/creator (this test assumes that a chat room – public or private – was created and that at least two users can join it. Refer to D1 to create a default chat room)

1. Right Click on a user→ Group Chat → Join New Group Chat→ either create a new group chat, or use the default from previous tests
2. Click Join
3. Type some text and enter
4. Repeat step 1 through 5 for another user
5. In the right portion of the room, right click on testuser2 → Occupant Actions → Ban
6. Enter a reason (optional)
7. Click OK (the room will reflect the decision by indicating that "testuser2 has been banned: reason here)[64] The entry text area will immediately become grayed out in the banned user.

D12. Ban a user from a group chat room when not having the role of moderator/creator

1. Right click on a user → Group Chat → Join New Group Chat→ use the default from previous tests
2. Click Join
3. Type some text and enter
4. Repeat step 1 through 5 for another user
5. In the window of a user other than the one who created the room, Right click on testuser2 → Occupants Actions → Ban (This is grayed out)[65]

D13. Joining a group chat room after been banned (this test assumes that D11 was just performed)

1. Right Click on the user that got banned in D11→ Group Chat → Join New Group Chat→ use the default from previous tests
2. Click Join (An alert window with Unable to join group chat appears with the explanation: "You are banned from this group chat")
3. Click OK
4. Close the window

D14. Kick (temporarily remove) a user from a group chat room when having the role of moderator/creator

1. Right Click on a user→ Group Chat → Join New Group Chat

[64] Banning creates a rule in the room configuration and the only way to get rid of the ban is for the owner of the room to remove that rule from the configuration file

[65] Only the moderator (the creator of the room) is allows to make those decisions

2. Nickname: testuser1
3. Room: private@conference.im.mysea.edu
4. Click Join
5. Type some text and enter
6. Repeat step 1 through 5 for another user
7. In the right portion of the room, right click on testuser2 → Occupants Actions → Kick
8. Enter a reason (optional)
9. Click OK (the room will reflect the decision by indicating that "testuser2 has been kicked: reason here) The entry text area will immediately become grayed out in the banned user.
10. Close the window of the chat room in the client that was just kicked

D15. Joining a group chat room after been kicked (this assumes that test D14 was just run)

1. In the user that was kicked, Right Click on that user→ Group Chat → Join New Group Chat
2. Accept the information already filled in as it is the last room that was joined
3. Type some text and enter[66]
4. Close the group chat room windows in both clients

E. Service discovery and Jabber User Directory (JUD) management (E1 through E2)

E1. List the rooms available and the number of participants in each room

1. Start Gajim
2. Right click on a user→ Discover Services…
3. Double click on Public Chatrooms
4. Click Close

E2. Access the user directory (this test is performed with the PSI version 0.9 client [eme03] and assumes that at least one user is properly registered with the server)

1. Start PSI
2. Click on the PSI icon (bottom left button) → Browse Services
3. Double click on jud.thesis.nps.edu
4. Search for users
5. Click Close
6. Click on the PSI icon (bottom left button) → Quit

[66] Kicking is not a permanent state, but based on sessions. It is only necessary to join the room again to be allowed to converse again in that room.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Business Wire, "XMPP Emerging as Chat Standard for the Federal Government; Jabber, Inc.'s Expanding Public Sector Footprint Enhanced by XMPP Expertise," 2006 http://findarticles.com/p/articles/mi_m0EIN/is_2006_August_14/ai_n16620565. August 14, Last visited 5/20/08.

[2] P. Saint-Andre (Editor), "Extensible Messaging and Presence Protocol (XMPP): Core," October 2004, http://www.faqs.org/rfcs/rfc3920.html. Last visited 6/2/08.

[3] P. Saint-Andre (Editor), "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence," October 2004, http://www.xmpp.org/rfcs/rfc3921.html. Last visited 6/2/08.

[4] "AIM.com," http://dashboard.aim.com/aim. Last visited 6/9/08.

[5] IBM Lotus Sametime, "Your onramp to Unified Communications and Collaboration," http://www-306.ibm.com/software/lotus/sametime/ Last visited 6/9/08.

[6] "ICQ Everybody, Everywhere," http://www.icq.com/ Last visited 6/9/08.

[7] IRC.org, "Internet Relay Chat (IRC) Your true Internet Relay Chat source," http://www.irc.org/ Last visited 6/9/08.

[8] J. Oikarinen and D. Reed, "Internet Relay Chat Protocol," May 1993, http://www.faqs.org/rfcs/rfc1459.html,. Last visited 5/20/08.

[9] Wikipedia, "Instant Messaging," http://en.wikipedia.org/wiki/Instant_messaging, February 2008. Last visited 6/9/08.

[10] GoogleTalk portal, http://www.google.com/talk/, Last visited 4/29/08.

[11] Skype portal, http://www.skype.com/, Last visited 4/3/08.

[12] Client-server, http://en.wikipedia.org/wiki/Client-server, Last visited 5/29/08.

[13] The NOPSIN Group Inc., "freePCTech," http://freepctech.com/pc/002/networks002.shtml, Last visited 5/22/08.

[14] C. Kalt, “Internet Relay Chat: Architecture,” April 2000, http://www.faqs.org/rfcs/rfc2810.html, Last visited 5/20/08.

[15] C.J.P. Moschovitis, H. Poole, T. Schuyler, and T.M. Senft, “History of the Internet,” 1999, www.abc-clio.com, Last visited 4/22/08.

[16] C. Kalt, “Internet Relay Chat: Channel Management,” April 2000, http://www.faqs.org/rfcs/rfc2811.html, Last visited 5/20/08.

[17] C. Kalt, “Internet Relay Chat: Client Protocol,” April 2000, http://www.faqs.org/rfcs/rfc2812.html, Last visited 5/20/08.

[18] C. Kalt, “Internet Relay Chat: Server Protocol,” April 2000, http://www.faqs.org/rfcs/rfc2813.html, Last visited 5/20/08.

[19] T. Van Vleck, “The History of Electronic Mail,” http://www.multicians.org/thvv/mail-history.html, Last visited 5/29/08.

[20] J. Scott, “BBS Documentary,” www.bbsdocumentary.com. Last visited 4/3/08.

[21] M. Day, Lotus, S. Rosenberg, dynamicsoft, H. Sugano, Fujistsu, “A Model for Presence and Instant Messaging,” February 2000, http://www.ietf.org/rfc/rfc2778.txt, Last visited 5/20/08.

[22] M. Day, Lotus, S. Aggarwal, Microsoft, G. Mohr, Activerse, J. Vincent, Into Networks, “Instant Messaging / Presence Protocol Requirements,” February 2000, http://www.ietf.org/rfc/rfc2779.txt, Last visited 5/29/08.

[23] P. Saint-Andre (Editor), “End-to-End Signing and Object Encryption for the Extensible Messaging Presence Protocol,” October 2004, http://www.ietf.org/rfc/rfc3923.txt, Last visited 5/29/08.

[24] Webopedia.com, Streaming, http://www.webopedia.com/TERM/s/streaming.html, Last visited 4/25/08.

[25] P. Saint-Andre (Editor), “Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM),” October 2004, http://www.rfc-archive.org/getrfc.php?rfc=3922, Last visited 5/20/08.

[26] P. Hoffman, IMC & VPNC, M. Blanchet, Viagenie, “Preparation of Internationalized Strings (“stringprep”),” December 2002, http://www.ietf.org/rfc/rfc3454.txt, Last visited 5/8/08.

[27] XMPP Standards Foundation, “XMPP Extensions,” http://www.xmpp.org/extensions/ Last visited 6/10/08.

[28] C.E. Irvine, T.E. Levin, T.D. Nguyen, D. Shifflett, J. Khosalim, P.C. Clark, A. Wong, F. Afinidad, D. Bibighaus, J. Sears, “Overview of a High Assurance Architecture for Distributed Multilevel Security,” in *Proceedings of the 5th IEEE Systems, Man and Cybernetics Information Assurance Workshop, United States Military Academy*, 2004, pp. 38-45.

[29] D.E. Bell, L. LaPadula, “Secure Computer System: Unified Exposition and Multics Interpretation” Tech. Rep. ED-TR-75-306, MITRE Corp., Hanscom AFB, MA 1975.

[30] K.J. Biba, “Integrity Considerations for Secure Computer Systems” Tech. Rep. ED-TR-76-372, MITRE Corp., 1977.

[31] BAE SYSTEMS, “Integrated Information Assurance: The XTS-400,” http://www.digitalnet.com/solutions/information_assurance/xts400_trusted_sys.htm, Last visited 6/9/08.

[32] BAE SYSTEMS, “Systems Information Technology. STOP Version 7: Platform for Secure Cross-Domain Application Development,” http://www.baesystems.com/ProductsServices/bae_prod_csit_xtsstop7.html, Last visited 6/1/08.

[33] Wikipedia.org, http://en.wikipedia.org/wiki/XTS-400, Last visited 6/2/08.

[34] Jabber Organization, http://jabber.org, Last visited 6/2/08.

[35] XMPP Standards Foundation, http://www.xmpp.org, Last visited 6/5/08.

[36] ejabberd, www.ejabberd.im, Last visited 4/20/08.

[37] jabberd14 Project, http://jabberd.org/, Last visited 6/10/08.

[38] jabberd2 Project, http://jabberd2.xiaoka.com/, Last visited 5/29/08.

[39] Openfire, http://www.igniterealtime.org/projects/openfire/index.jsp, Last visited 4/30/08.

[40] TIGASE.ORG, http://www.tigase.org/, Last accessed 5/29/08.

[41] The Linux Documentation Project, http://tldp.org/HOWTO/archived/WWW-HOWTO/WWW-HOWTO-7.html, Last visited 4/30/08.

[42] PSI, http://psi-im.org/, Last visited 5/22/08.

[43] Gajim, www.gajim.org, Last visited 6/2/08.

[44] Pidgin, http://www.pidgin.im/, Last visited 5/2/08.

[45] The Jabber Manual Team, "Jabberd1.4.x Administration Guide," 2004, http://jabberd.org/1.4/doc/adminguide, Last visited 6/18/08.

[46] jabberd14, http://jabberd.org/news/ Last visited 6/10/08.

[47] Van Emery, "Using jabberd as a Private Instant Messaging Service," 2003, www.**vanemery**.com/Linux/Jabber/**jabberd**.html, Last visited 6/2/08.

[48] jas, "The GNU Transport Layer Security Library," 02/18/08, http://www.gnu.org/software/gnutls, Last visited 3/18/08.

[49] B. Barney, "POSIX Threads Programming," 5/19/08 https://computing.llnl.gov/tutorials/pthreads/, Last visited 6/18/08.

[50] "GNU IDN Library – Libidn," http://josefsson.org/libidn/, Last visited 6/2/08.

[51] "[jadmin] jabberd14-1.6.1.1 make error (asn1) hints please," http://mail.jabber.org/pipermail/jadmin/2007-October/099695.html, Last visited 5/12/08.

[52] C. Cooper, "Using Expat," http://www.xml.com/pub/a/1999/09/expat/index.html. 1999, Last visited 6/2/08.

[53] "The Expat XML Parser," http://expat.sourceforge.net/, Last visited 6/2/08.

[54] "Expat XML Parser," http://sourceforge.net/projects/expat/, Last visited 6/2/08.

[55] S. Josefsson, (Libidn) "README-alpha," 2006.

[56] pkg-config, http://pkg-config.freedesktop.org/wiki/, Last visited 6/2/08.

[57] The GTK+ Project, http://www.gtk.org/, Last visited 5/6/08.

[58] Description page. http://invisible-island.net/autoconf/autoconf.html, Last visited 5/29/08.

[59] Free Software Foundation, Inc., "GNU Libtool – The GNU Portable Library Tool," 2008, http://www.gnu.org/software/libtool/libtool.html, Last visited 6/2/08.

[60] "The GTK+ Project," http://www.gtk.org/overview.html, Last visited 5/8/08.

[61] "GNOME: The Free Software Desktop Project," http://www.gtk.org/overview.html, Last visited 5/9/08.

[62] P. Curtis, "[jadmin] No platform-settings file for Yahoo Transport on OS X Tiger server," 2005, http://mail.jabber.org/pipermail/jadmin/2005-May/095555.html, Last visited 6/2/08.

[63] International Organization for Standardization, "Numeric representations of Dates and Time," http://www.iso.org/iso/date_and_time_format, Last visited 5/10/08.

[64] "jabberd14-implementation of an instant messaging server using the Jabber/XMPP protocols in C," http://jabberd.org/codedoc/index.html, Last visited 4/30/08.

[65] "dialback Directory Reference," http://jabberd.org/codedoc/dir_91b33e64cab139823edc5b7ff9a483a2.html, Last visited 5/8/08.

[66] "dnsrv Directory Reference," http://jabberd.org/codedoc/dir_b9a8410acaca88619d34b480129ce1cb.html, Last visited 5/8/08.

[67] "jabberd Directory Reference," http://jabberd.org/codedoc/dir_116879c8fe121c441b553a37f7e1633b.html#_details, Last visited 5/8/08.

[68] "Base Directory Reference," http://jabberd.org/codedoc/dir_688d28322d6e15e35aa9ec64cb1dee18.html, Last visited 5/8/08.

[69] "lib Directory Reference," http://jabberd.org/codedoc/dir_10d906a57e0bab9a2643f82b1f899021.html, Last visited 5/8/08.

[70] "jsm Directory Reference," http://jabberd.org/codedoc/dir_b1173ba875483b984821b47913be8f59.html, Last visited 5/8/08.

[71] "pthsock Directory Reference," http://jabberd.org/codedoc/dir_96d8aed2629794315b23680b88cc5ba5.html, Last visited 5/8/08.

[72] "xdb_file Directory Reference," http://jabberd.org/codedoc/dir_f16246d04214d4769785e5aa1f02f9dd.html, Last visited 5/8/08.

[73] "xdb_sql Directory Reference," http://jabberd.org/codedoc/dir_f91b2057b7b5bf1da07aff031960dcd8.html, Last visited 5/8/08.

[74] jabber 1.4.x configuration file. http://jabberd.org/1.4/doc/conf Last visited 5/22/08.

[75] "Tigase Test Suite," http://projects.tigase.org/testsuite/trac, Last accessed 5/29/08.

[76] D. Puryear, "The Jabber Test Suite," 2003, http://sourceforge.net/projects/jabbertest/, Last accessed 5/23/08.

[77] "Jabber Test Suite (JabberTest)," http://sourceforge.net/projects/jabbertest, Last visited 5/23/08.

[78] XMPP Standards Foundation, "XEP-0001," http://www.xmpp.org/extensions/xep-0001.html#states-Draft, Last visited 5/28/08.

[79] Naval Postgraduate School Center for Information System Security Studies and Research, "XTS-400: System Administration Final Installation and Setup *STOP 7.0 beta 1*," April 2008.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. John Campbell
   National Security Agency
   Fort Meade, MD

4. Steve Davis
   NRO
   Chantilly, VA

5. Dr. Tim Fossum
   National Science Foundation

6. Dr. Greg Larson
   IDA
   Alexandria, VA

7. Dr. John Monastra
   Aerospace Corporation
   Chantilly, VA

8. Boyd Fletcher
   SPAWAR
   San Diego, CA

9 Dr. Cynthia E. Irvine
   Naval Postgraduate School
   Monterey, CA

10 Thuy D. Nguyen
   Naval Postgraduate School
   Monterey, CA

11. Claire LaVelle
   Naval Postgraduate School
   Monterey, CA