xmpproxy, a proxy server for XMPP

Project description and plan

Ralph Krimmel

January 10, 2012

Contents

1 Abstract **3**

2 Introduction **3**

3 Technology **5**
3.1 Software choice 5
3.2 Djabberd 5
3.3 xmpproxy 5
3.3.1 Configuration 5
3.3.2 Logging 5
3.3.3 Modules 6

4 Tasks **7**

5 Schedule **8**

6 Literature **9**

1 Abstract

"The extensible messaging and presence protocol xmpp is an open technology for real time communication which powers a wide range of applications including instant messaging, presence, multi-party chat, voice and video calls, collaboration, lightweight middleware, content syndication and generalized routing of xml data." $_{http://xmpp.org/about}$

Being widely used in several large communication platforms such as `Google talk`, `XMPP` has become an important protocol for communication via network. This document will describe an issue using `XMPP`'s instant messaging capabilities in a multi client environment and propose a possible solution. It also contains a time schedule for implementing this solution as a project for the course "Applied IT project" at the University of Gothenburg.

2 Introduction

The `XMPP` or "jabber"-protocol supports multiple clients to be connected to the same account at the same time. From the server sides view a connected client is called a *ressource*. The consequence of multiple ressources for the same account is that the server has to decide to which ressource an incoming message, or *stanza* in xmpp terminology, is routed to. This is done via the so called *presence priority* a client can connect with. The client with the higher priority will receive the message. If two or more ressources have the same priority, the server may use some other rule to decide between those or deliver the message to all of them. For example, the server could use the most recent connect time or the most recent activity time. However, the server is not allowed to deliver the stanza to an available resource with a negative priority.

This behaviour leads to the problem, that conversations and logfiles of conversations may not be complete on every client, if one client connects with a higher priority. Figure 1 shows an example where this would be the case. Client A does not know the parts of the conversation that are held from Client B. Also, not just incoming messages will be missing, Client A will not even know about outgoing stanzas from Client B.

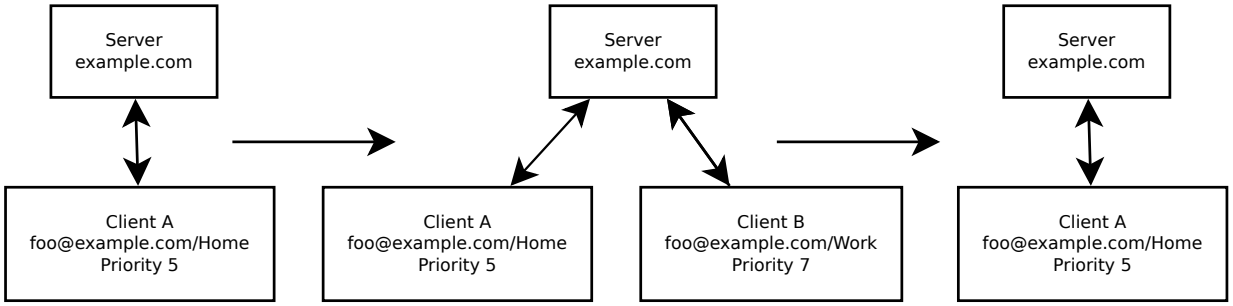

Figure 1: Clients with 2 different priorities are connected to the same account.

A possible solution for this issue may be the use of a proxy server as it can be seen in figure 2. This

server will make sure, that every incoming stanza will reach both clients and every outgoing stanza will be sent to every ressource as well as to the targeted server.

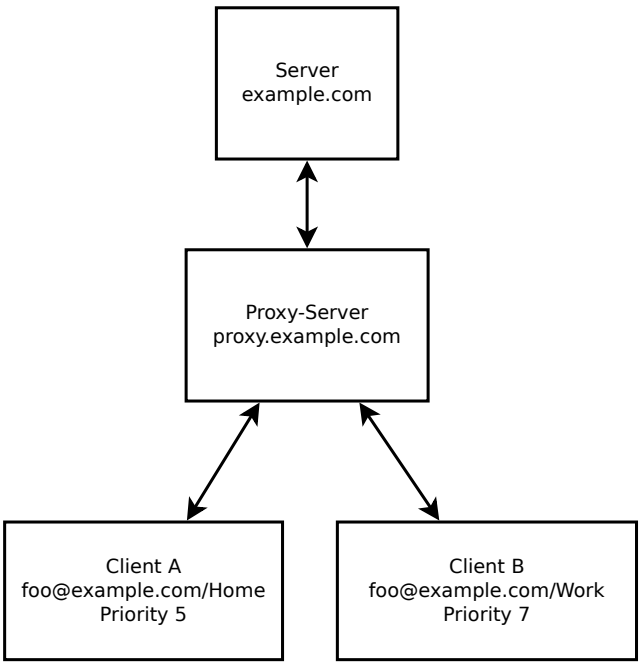

Figure 2: Two clients connected to a proxy.

3 Technology

3.1 Software choice

For implementing the xmpp proxy server several design descisions had to me made. First of all a programming language was chosen. Possible languages were Perl, C, C++ and Java because of the preferences of the author. The criteria for the programming language was first of all the availability of proper libraries that deal with the xmpp protocol, both server and client side. Unfortunately, this excluded all languages except perl because there are just few xmpp server libraries at all. Therefore the xmpp proxy server will make use of the `djabberd` xmpp/jabber server although it lacks a proper documentation.

3.2 Djabberd

Djabberd is a modular, scalable and extensible jabber server written in perl where almost everything defers to hooks to be implemented by plugins. It is written and maintained by *Brad Fitzpatrick*, *Artur Bergman* and *Jonathan Steinert*. When this document was created, the most recent version of djabberd was 0.85, released on the thirteenth of June in 2011. Having a modular structure, djabberd is a perfect framework for building the required proxy server because it offers hooks for modifying the behaviour of the authentication and authorization process, roster storage and message delivery. Also, `djabberd` can be used in environments which require a high performance for it is asynchrone/event-based, using epoll on linux 2.6.

3.3 xmpproxy

3.3.1 Configuration

The configuration of `xmpproxy` is done via a single xml configuration file. The file, `proxy.xml`, can be found in the same directory as the `xmpproxy` executable. The file stores all the non-ephemeral information of the user database (names, passwords, options) and `xmpproxy` specific options like the node name of the server. Here is an example configuration file:

```
1 <config>
2   <node name="fluttershy"></node>
3
4   <user name="test" passwd="bar" resource="access">
5     <account jid="test@milk-and-cookies.net" passwd="test" resource="xmpproxy"/>
6     <account jid="bla@milk-and-cookies.net" passwd="bla" resource="xmpproxy"/>
7   </user>
8 </config>
```

Cave: The syntax of this file may change in future versions of `xmpproxy`.

3.3.2 Logging

`xmpproxy` uses `log4perl` to create nice looking, configurable log files. It offers five different log levels (ordered from the most to the least verbose):

- DEBUG
- INFO
- WARN
- ERROR
- CRITICAL

The desired loglevel can be adjusted by setting the environmental variable "LOGLEVEL" to the wanted value, e.g.
`$ export LOGLEVEL=DEBUG`

3.3.3 Modules

This is just a short description of the available modules. For a full API documentation see *refman.pdf*

xmpproxy::Userdb and xmpproxy::User Those are classes for the management of users in xmpproxy. A `xmpproxy::Userdb` object stores instances of User objects. It also offers methods to add, delete and modify users. A `xmpproxy::User` object holds information of a user. That information is username, password and a list of `DJabberd::Queue::ClientOut` objects, which represent a client connection.

DJabberd::Authen::Proxy Module that queries the xmpp user database and checks the provided credentials.

Djabberd::Connection::ClientOut and DJabberd::Queue::ClientOut Those two classes are needed to make DJabberd connecting to another Jabberserver as a client. `DJabberd::Stanza::SASL` is used if SASL authentication is supported.

DJabberd::RosterStorage::Proxy Module for roster ("buddylist") managment. Has methods to fetch and modify the rosters for each proxy account. Currently just basic functions for getting rosters.

DJabberd::Delivery::Local The task of this module is to deliver incoming message/presence stanzas properly. To do this, the "to"-attribute has to be rewritten to deliver the message to `xmpproxy` users.

DJabberd::Delivery::Proxy Module for delivering all outgoing message/presence stanzas.(Partly) implements XEP-0280(Message carbons). This means that it mirrors all outgoing message stanzas to each connected client. Also, for forwarding purposes, the "from" attribute is rewritten.

DJabberd::IQ Module that handles all IQ stanzas. Client side functions have been added.

DJabberd::Message Module that handles all message stanzas. Client side functions have been added.

DJabberd::Presence Module that handles all presence stanzas. Client side functions have been added.

4 Tasks

The following tasks are to be completed:

- Studies of literature and source code
- Userdatase `xmpproxy::Userdb` and `xmpproxy::User`
- Module for proxy authentication `DJabberd::Authen::Proxy`
- Basic XMPP client functionalities `DJabberd::Connection::ClientOut`, `DJabberd::Queue::ClientOut` and `DJabberd::Stanza::SASL`
- Module for proxy roster handling `DJabberd::RosterStorage::Proxy`
- Module for proxy presence handling `DJabberd::Presence` and `DJabberd::Delivery::Proxy`
- Module for message delivery `DJabberd::Message` and `DJabberd::Delivery::Proxy`
- Module for basic IQ handling `DJabberd::IQ`
- (Optional) Module for proxy groupchat
- (Optional) Module for proxy jingle (Voice/Video via jabber)
- (Optional) root admin bot
- Write Documentation
- Write Report
- Create presentation

5 Schedule

The `xmpproxy` software will be written in the context of the course "Applied IT Project" course at the University of Gothenburg. This requires to "formulate and execute a finite Applied IT project within a limited time-frame". Therefore it is neccessary to schedule the tasks described in the previous sections. Table 1 shows the the estimated time every task will take. The minimum required amount of time this project will need is 200 hours. Due to the size of the project, the time plan will cover 280 hours of work. Those 300 hours are distributed over 18 weeks. Four of those 18 weeks won't be available for activity because of exam preparation and as backup time in case of unforseen consequences (illness, design mistakes) or implementation difficulties. The project is beeing executed mid semester while one other 7.5 ECTS course is absolved. That means that 50% of the effort can be put into `xmpproxy`. Therefore, the average weekly time used for working on `xmpproxy` is approximately 20h.

Date (11/12)	**Task**	**Time (in h)**
1.10. - 7.10.	Studies of literature	20
8.10. - 14.10.	Project plan	20
15.10. - 21.10.	First exam period	-
22.10. - 25.10.	User database	10
26.10. - 29.10.	Module for proxy authentication	10
30.10. - 13.11.	Basic XMPP client functionalities	40
14.11. - 20.11.	Module for proxy roster handling	20
21.11. - 27.11.	Module for proxy presence handling	20
28.11. - 3.12.	Module for message delivery	20
4.11. - 9.11	Module for basic IQ handling	20
10.12. - 16.12.	Second exam period	-
17.12 - 22.12	Module for basic IQ handling	20
23.12 - 3.12	Documentation,Report,Praesentation	80
4.1 - 9.1	Backup time	-
-	(Optional) Module for proxy groupchat	?
-	(Optional) Module for proxy jingle (Voice/Video via jabber)	?
-	(Optional) root admin bot	?

Table 1: Time plan

6 Literature

- `Programming jabber - extending XML messaging` by DJ Adams
- `http://xmpp.org/rfcs/rfc6120.html`
- `http://xmpp.org/rfcs/rfc6121.html`
- `http://xmpp.org/rfcs/rfc6122.html`
- `http://search.cpan.org/~mart/DJabberd-0.85/`
- All the DJabberd sources