2015

Distributed Systems (cs4532)

LAB 3 - DESIGN A CHAT ROOM SERVER

KIERAN FRASER - 11492108

Contents

Abstract

This report explores the architectures and associated protocols of a chat server. The server itself can support group and private messaging. An implementation of an "optimal" solution is then suggested in the form of a customized protocol influenced, in part, by XMPP. The pros and cons of choosing the particular architecture and protocol is then discussed in greater detail. The high level features of the chat room discussed in this report include:

- Group Messaging
- Private Messaging
- Identity/Authentication

Background Research

Two protocols in particular are discussed in this report, **IRC** (Internet Relay Chat), which supports communication in the form of text via a client-server architecture, and **XMPP**, which supports communication in the form of XML stanzas via a decentralized client-server architecture. These two protocols were brought to the forefront of our group discussions mainly due to their reliability and widespread use. IRC is an "old school" solution, however it was argued that it is the most simple to implement and it does work efficiently even if it lacks some of the features that XMPP can provide such as persistence. XMPP, on the other hand, is in widespread use throughout the world today and is used on a daily basis by *WhatsApp* users. *WhatsApp* have implemented a customized version of XMPP called **FunXMPP**. Again, the solution discussed later in this report, took inspiration from the ideology of customizing an existing protocol to suit a particular need.

IRC

As discussed above, IRC implements a client-server architecture which is primarily text based.
It works over TCP and messages are sent as single lines of text. Commands to the server are prefixed by a "/". For example: "/join <channel name>"

Each IRC message may consist of up to three main parts:

- The prefix (optional)
- The command
- The command-parameters (max of 15)

The command and all parameters are separated by one ASCII space character (0x20) each. IRC messages are also always lines of characters terminated with a CR-LF (Carriage Return – Line Feed) pair which **cannot** exceed 512 characters in length.

The format of the messages sent from the client to the server are in the form:

prefix command command-parameters\r\n

In this example the prefix would be the client sending the message "nickname@host". Based on the "command" sent from the client, the server will do something with the payload. For example the whole message could be "nickname@host PRVMSG destnickname@host message-contents", in

which the server would know to send the message only to the specified client. Other commands would then be available for group chat.
IRC handles the identity of each user by allowing a unique username be chosen. The server then keeps track of these usernames and associates it with a socket connection. The server will listen on each of these sockets for messages. In order to detect a connection from each client to confirm its active state the server must poll each connection. Therefore the **PING** command is used if the server hasn't received communication from a client for a specified amount of time. If the client doesn't make a timely response the connection is closed. With regards group chats, if a latecomer is to join a group chat there is no way for the client to be updated with previous messages. This is because IRC was designed to be live real time communication. Messages are only received if the client is online.

Notable Pros for IRC:

- Simple message format
- No serialisation
- Chatroom design and protocol are outlined in the specification
- PM design and protocol are outlined in the specification
- Method to prove client state existence outlined in the specification

Notable Cons for IRC:

- It's not modern technology
- Restricted to text-based chat
- Complex specification

XMPP

XMPP is an open technology used for real-time communication. Unlike IRC, XMPP communicates using **XML** as its base. As mentioned above, it utilizes a decentralized client-server architecture similar to the World Wide Web which adds to the strength of its reliability. For example if a server was to go down, the whole system wouldn't cease to operate as the request would simply be routed to a different server.

XMPP handles identity of its clients using a JabberID (**JID**) as an address for sending/receiving messages. The format of this idea resembles email addresses: **userName@jabber.org**. A standard message format in XMPP would be:

```
<message from="johnDoe@jabber.com" to="janeDoe@jabber.com" type="chat">
        <body> A chat message </body>
</message>
```

The XML "stanza" contains the JID of the client sending the message, the JID of the client receiving the message, the type of message and the content itself. The above example would be a PM. XMPP provides facilities for group chats also, MUC or "Multi-User Chat". The chatroom itself is assigned a JID (e.g. distributedsystems@group.jabber.org) and users have a nickname which is used to identify themselves in the chatroom (e.g. distributedsystems@group.jabber.org/Kieran). To send a message to the group chat and XML stanza is sent from the client with a destination address to the chatroom JID. The server will then take the contents and sender of the message and form individual XML stanzas for each client in the group chat and send them to each individual client. The message type for each message is "groupchat".

XMPP maintains client state through a list of connected JID's in a group. Each time a client joins, leaves or changes their "status", presence XML stanzas are sent to the group chats address. Latecomers to the group chat can also be accommodated (or not at all depending on the need). The server persists messages and can send the new comer the previous messages if necessary.

Whatsapp uses a customized version of XMPP named **FunXMPP**. Essentially it's a slimmed down version of XMPP. *Whatsapp* being a mobile application, the less overhead the more efficient the communication. Therefore the message format for FunXMPP differs slightly from regular XMPP:

```
<message to="34123456789@s.whatsapp.net" type="text" id="message-1417651059-2"
t="1417651059">
  <body>Test</body>
</message>
```

Whatsapp message in regular XMPP

```
<\x59 \xa5="01234567890@\x91" \xa7="\xa2" \x44="message-1417651059-2"
\xa1="1417651059">
   <\x12>Test</\x12>
</\x58>
```

Whatsapp message in FunXMPP

The idea behind this was to map common keywords (such as "message" and "text") to a single byte using a hash map in order to decrease the size of the message being sent. The syntax of a byte is "\xnn" where "nn" is a hexadecimal value.

Whatsapp does use a regular XMPP JID to identify one user from another. On sign up, each user is assigned a username and password. Previously this was generated based on a mobile phones IMEI number which is unique for each device. However, this limited the use of the whatsapp application to that particular device resulting in restriction of use. To rectify this, at sign up the app now sends a unique 5 digit pin to the Whatsapp server. A message is then sent back to the user's phone number from Whatsapp signifying to the app that a request for a JID be sent to the Whatsapp server. The JID is then generated on the Whatsapp server by concatenating the user's phone number and country code.

Client requests are sent to randomly chosen domain servers of which Whatsapp has many in the form of "d1.whatsapp.net", "d2.whatsapp.net" etc. In a reverse engineering investigation carried out by *whitehatgroup*, the client connects to the randomly chosen DNS via the following TCP/UDP port: socket://d3.whatsapp.net:5222. They also performed a trace-route on a message so as to identify visually how the packets are passed from server to server (Figure 1).

Figure 1 – Trace-route performed on a Whatsapp message

Notable Pros for XMPP

- No central master
- Open standards
- Extendable/Modular
- Scalable
- Supports encryption

Notable Cons for XMPP

- Doesn't support quality of service
- Focus on mobile platform

Proposed Implementation

High level Functionality

The following is the proposed protocol and architecture for the chat server. It was decided to focus on simplicity as much as possible and to strive for maximum efficiency and reliability over the addition supporting of extra features such as file sharing. The protocol takes inspiration from XMPP and IRC.

The architecture is comprised of a simple decentralized client-server model whereby clients and the chat server interact via a TCP connection. Each client is uniquely identifiable by a username which is generated by the user itself when they join but is confirmed to be unique by the server. The server has a database of users of which to confirm the uniqueness of the chosen name. The server will also map the unique username to "Nicknames" to be used within the chat itself.

If users are to log in to the chat session from different network locations, the server will be able to update their associated IP address once the user has authenticated themselves with their unique username and a self-chosen password. In this manner the server can persist the identity of the user and send the messages to the correct client IP address. Again this requires that the server save on behalf of a user a unique id (already mentioned), a password, a status and an associated IP address. The status would be the last known status of the user, such as "online" or "offline". Again, the IP address would be the last known IP address of the user. The IP address would be updated however if the user was to log in (send a request to the server with correct username and password pair) from a different location.

The chat server will support both private and group messaging using the same methodology as XMPP. This would mean that when a client wishes to send a message to just one other client, they will include that client in the message they send to the server as well as a "type" which the server knows is defined as a private message. In the same respect for a group message, there will be a unique group ID. When a client wishes to send a message to the group it will simply send the message to the server with the group ID as the destination recipient and a "type" defined as a "group-message". The server will then relay the message to all members (all unique ID's) associated with that particular group ID. Essentially the server will have a set of predefined forms of input message which define how it will handle the particular message. The server will also have various input commands which will dictate an action to perform.

Functionality in Detail

The client and server will be communicating via a TCP connection and the format of the messages sent back and forth will be JSON. This was reasoned the best option as plain text (such as IRC) would be time consuming to handle and parse and XML (such as standard XMPP) is less well supported than JSON. An example of the format of the JSON passed between client and server is as follows:

{
Command: “msg”
To: UID
Type: PM/GroupID
Message: “Some Text”
From: UID
}

The format of the UID would be similar to that of XMPP. An example of which would be:

KieransUniqueName@server.hostname

The server will first look at the command value of the JSON object. If the command is “msg” it will look for specific tags such as “To”, “Type” and “Message”. If for example the command was “invite” the server would look simply for a UID of the user to invite to the group chat and a UID of the group chat to add the user to. In this manner, various functions can be carried by the user. A number of commands deemed necessary are as follows:

Commands

- Invite – As other users are unaware of the UID of the group chat, they can only be added by users who are already in the chat and know the UID (this is the method *Whatsapp* and *Facebook* implement)
- Leave group – To remove your association from the group UID
- Msg – To send a message to the group chat or a PM
- Update Details – To change personal details such as your display name which is mapped to your UID, or when logging out this command could be sent to change the status of the user.
- Ping – Sent by the client to the server to relay that the connection is still healthy
- History – When added to a group chat, request the previous n messages

It was also argued that in order for latecomers to attain a record of the history of messages the server could pull on the resources of the client. Essentially the server would check which server had a good number of previous messages and would request them. The server would then forward these messages to the client that arrived late to the conversation. The argument against this method of persisting the history of messages was that we would rely on the client to “store” the messages and it could complicate the process if there was a very high fluctuation in clients arriving and leaving the conversation as there may be no client that has a full picture of the conversation (but only snapshots). This is why it was chosen to have a database on the server and for the server to save the previous *n* messages. The client would then keep a timestamp of last log on time. When the client logs on to the server, this time is compared with the timestamp of the last message sent in the group chat. If the time is found to be different a “History” command is sent from the client to the server along with the timestamp to request for the previous messages. This is the same process for private messages sent to a user when they are offline. If their current status is “offline” the message is stored in the database. Once the user comes online the server searches the database with any messages for that client UID. Once delivered, all private messages are deleted from the server to avoid confusion and multiple sending of messages.

In order for the server to persist client state, the “Ping” command is used. This is the same methodology as that outlined for XMPP above. If a set amount of time has passed without hearing from the client, a “Ping” is sent to the client. If the client doesn’t reply in a timely manner with a “Ping” command of its own then the server attempts to reconnect afresh with the last known IP address of the client or, failing that, the client’s status is changed to “offline” and all pending messages for the client are stored in the server’s database.

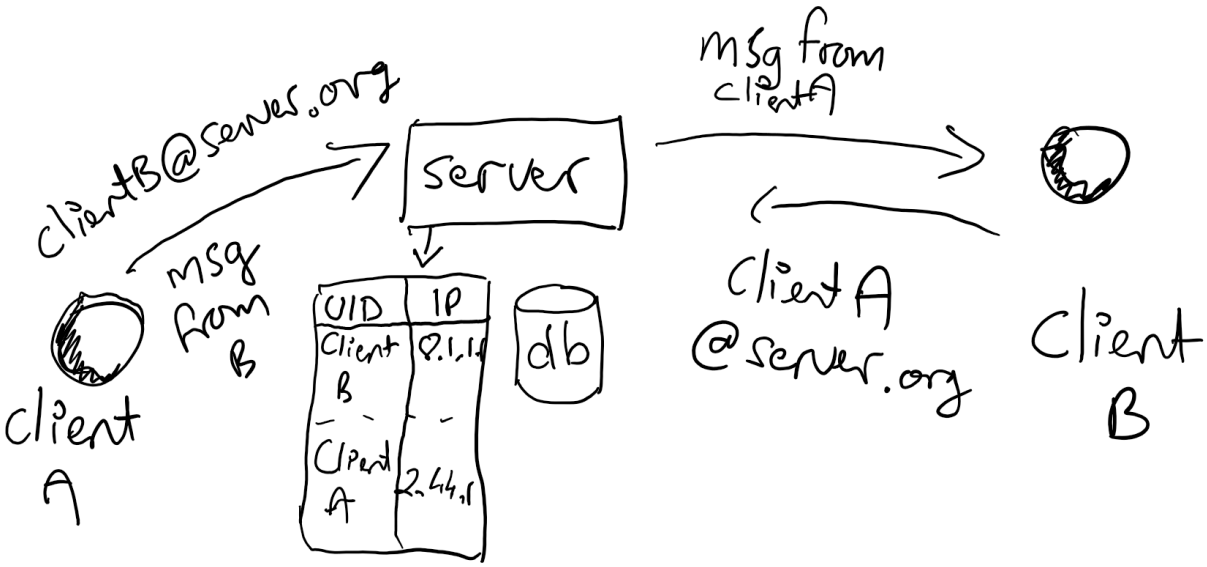

Figure 2 – Private message sent from Client A to the server. The server matches the unique id of Client B to an associated IP address and forwards the message. Client B then returns the favour.

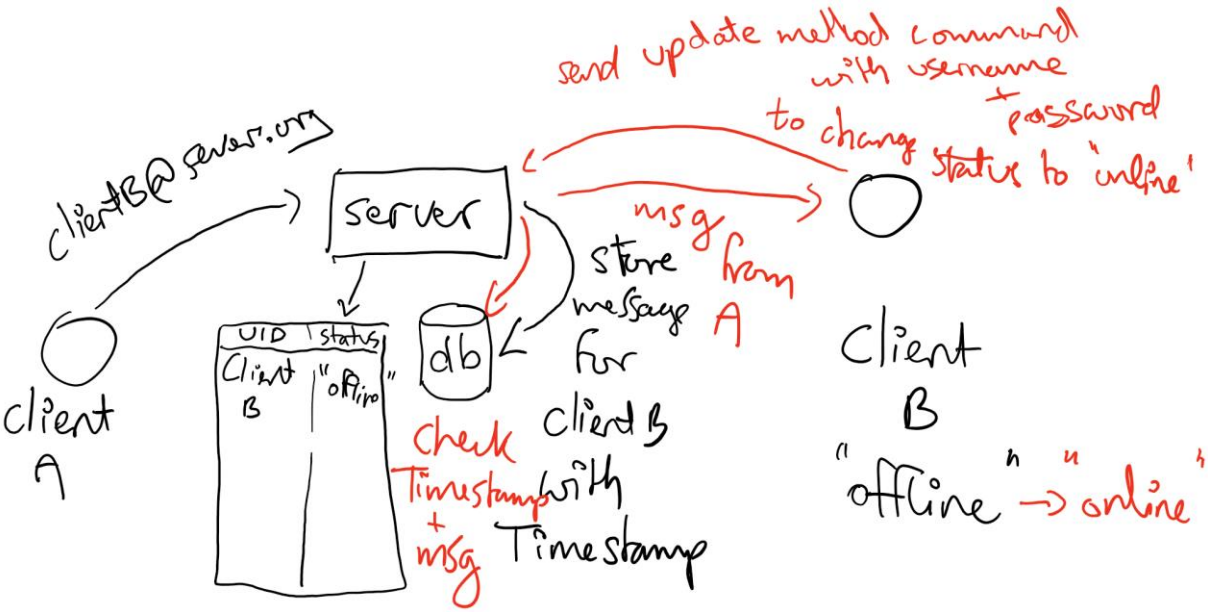

Figure 3 – Client A is sends a message to the server for Client B. The server checks the status of Client B. As Client B is currently offline the server stores the message in the database. In red: Client B comes back online and “logs in” by sending an update command to the server containing the unique username and password. The server now changes the associated status of Client B to “online” and proceeds to check if any messages in the database are for Client B.

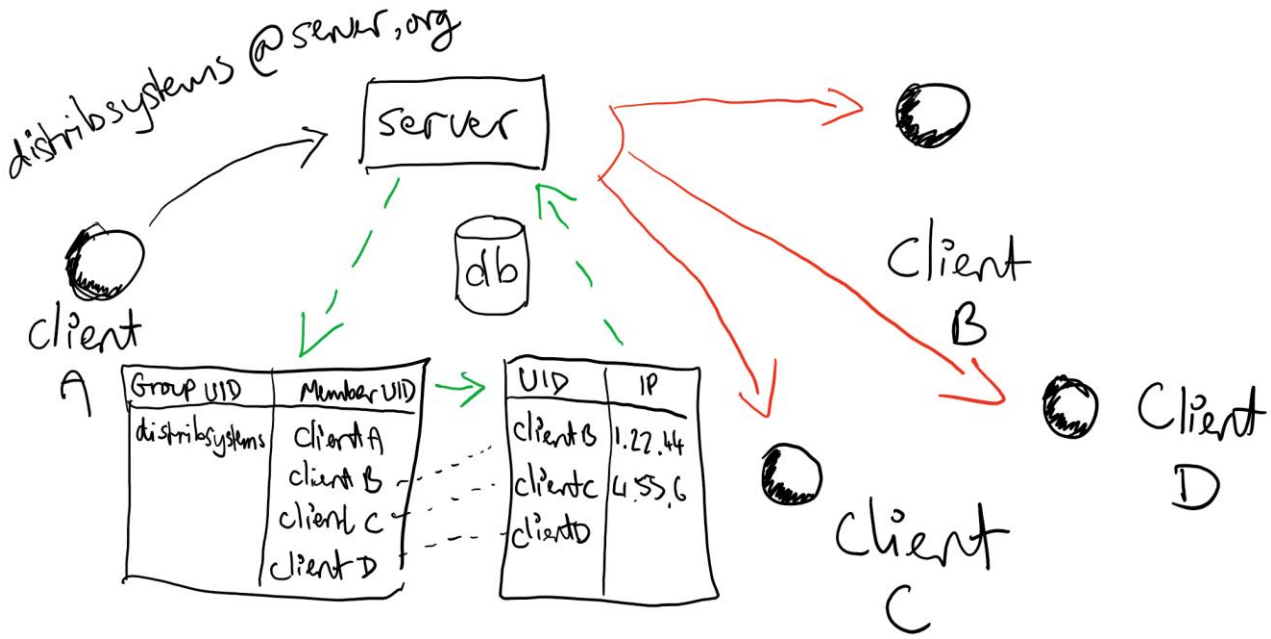

Figure 4 – Client A sends a message to the group chat. The UID of the group chat is in the "to" field of the message and "group-message" is the type. The server deals with this by looking up the associated UID's of the members of the group and matches them to their IP address. The message is then sent to each individual client in the list (apart from the original sender).

References

Understanding IRC protocol - http://books.msspace.net/mirrorbooks/irchacks/059600687X/irchks-CHP-13-SECT-2.html

IRC Server Protocol - https://tools.ietf.org/html/rfc2813

XMPP Protocol - http://xmpp.org/extensions/xep-0198.xml

FunXMPP Protocol - https://github.com/WHAnonymous/Chat-API/wiki/FunXMPP-Protocol

Session Initiation Protocol (SIP) - http://www.siptutorial.net/SIP/index.html

Whatsapp Architecture - http://blog-bhaskaruni.blogspot.ie/2014/02/whatsapp-architecture.html

Whatsapp - http://whitehatgroup.in/wp-content/uploads/2014/03/how_whatsapp_works.pdf