XMPP-Proxy

Trabajo Práctico Especial 2016/2

Grupo 5

Índice

1. Protocolos y aplicaciones desarrolladas

a. Objetivo

El objetivo de este trabajo práctico es implementar un servidor proxy para el protocolo XMPP (Extensible Messaging and Presence Protocol) que pueda ser usado por clientes XMPP para el envío y recepción de mensajes de chat.

El proxy que se implementó le provee al cliente XMPP algunas funcionalidades adicionales que no son características de dicho protocolo. Alguna de ellas son la manipulación de los mensajes cambiando los caracteres y la silenciación de un usuario.

Para poder dar con el objetivo se decidió recrear el siguiente escenario:

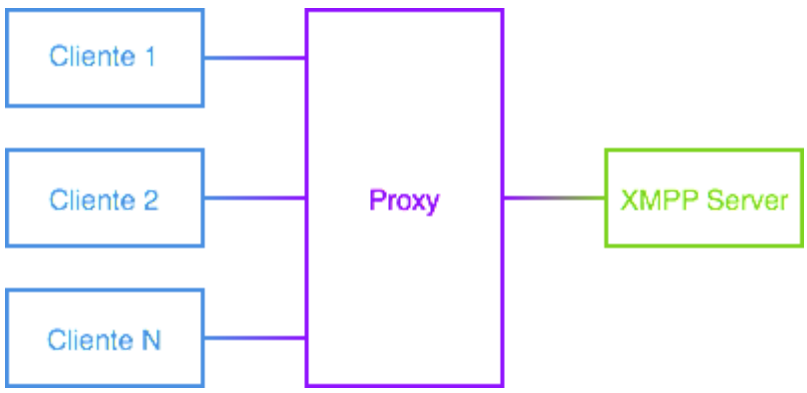


Figura 1: Estructura general del proxy y sus conexiones a los clientes y Servidor XMPP

b. Handshake

Cuando el cliente quiere conectarse al servidor XMPP, en realidad se está conectando al proxy el cual le responde como si fuera el mismo servidor XMPP. Lo que hace el proxy es simplemente reenviar lo que el cliente le intenta mandar al servidor y viceversa. El proxy le solicita al cliente que la autenticación sea plana y no codificada, es así que el proxy reconoce con quién está hablando.

Una vez que el proxy sabe quién es el cliente se procede a simular que el cliente está conectado directamente con un servidor XMPP.

Para poder mantener la conexión entre cliente-proxy y proxy-servidor, el proxy cuenta con dos mapas internos. El primero se utiliza para identificar una conexión dado un socket channel de un cliente, es decir una conexión cliente-proxy. El segundo se utiliza para identificar una conexión dado un socket channel de un server, es decir, una conexión proxy-server.

Figura 2: Se representa la conexión de un cliente al proxy mediante un socket channel. También se representa la conexión entre el proxy y el Server XMPP mediante otro socket channel.

c. Flujo de conexión cliente, proxy y servidor

Para cada cliente que se conecta el proxy cuenta con una clase llamada Proxy Connection donde se guardan los datos de dicha conexiones sean:

- Jid
- Username
- Client channel
- Server channel

Client channel es el socket channel mediante el cual se establece una conexión cliente proxy mientras que el Server channel es el socket que comunica el proxy con el servidor XMPP.

Para el análisis y tratamiento de los mensajes se utilizaron expresiones regulares para parsear las stanzas. Dichas expresiones regulares sirven por ejemplo para analizar qué tipo de mensaje es el que recibió el proxy y transformar el contenido en caso de ser necesario.

d. Configuración Server XMPP

Con el fin de usar un server XMPP el cual proxear se decidió instalar Prosody en una máquina Ubuntu en Amazon Web Services. La misma se encuentra disponible en ec2-54-69-136-236.us-west-2.compute.amazonaws.com

La configuración del server es la siguiente:

```
admins = { "admin@nowixmppserver" }
authentication = "internal_plain"
allow_registration = true;
modules_enabled = {
    "roster";
    "register";
    "admin_adhoc";
    "saslauth";
}
log = {
    debug = "/var/log/prosody/prosody.log";
    error = "/var/log/prosody/prosody.err";
    -- "*syslog"; -- Uncomment this for logging to syslog
    -- "*console"; -- Log to the console, useful for debugging with daemonize=false
}
daemonize=false
VirtualHost "nowixmppserver"
```

e. Métricas

Se cuenta con una recolección de métricas las cuales pueden ser obtenidas a través de monitoreo remoto. Las métricas son almacenadas en memoria con lo que proveen estadísticas de la sesión actual del proxy.

Las métricas recolectadas son la cantidad de accesos realizados, bytes enviados y bytes recibidos tanto para cada usuario en particular como para el proxy en general. Éstas son volátiles y se resetan cada vez que arranca el proxy.

Como accesos realizados se entiende la cantidad de veces que el proxy procesa un mensaje recibido y/o enviado.

Como bytes recibidos se entiende la cantidad de bytes que el servidor lee de un cliente xmpp o un servidor xmpp.

Como bytes enviados se entiende la cantidad de bytes que el servidor le envía a un cliente xmpp o servidor xmpp.

Las métricas son devueltas en formato JSON para que puedan ser analizadas por otras aplicaciones.

```
➜  ~ nc localhost 5224
Bienvenido a la administración del proxy!
login admin=123
201 - Login succesful
get metrics
200 -
metrics: {
        current_time_millis: 1479228312608,
        bytes_received: 0,
        bytes_sent: 0,
        accesses: 0
```

```
}
```

f. Multiplexor de Cuentas

El proxy cuenta con la opción de multiplexación de usuarios. A partir del archivo de configuraciones, el proxy cuando inicia toma los usuarios que están multiplexados y también chequea el flag para saber si está habilitado o no.

Por defecto todas las conexiones son hacia el Server XMPP que se haya elegido por defecto. En el caso de que no lo sea y el usuario este multiplexado lo manda al servidor de multiplexación de ese usuario.

Para esto fue fundamental habilitar la autenticación en plano para saber el jid del usuario antes de conectarlo con el server default. Lo que se hace para obtener el nombre de usuario del cliente que se quiere conectar es simular un intento de conexión donde el proxy se hace pasar por el server respondiendole al cliente como si lo fuese, mientras tanto se guardan los mensajes enviados por el cliente en una cola. Una vez obtenido el jid del cliente pasan a enviarse los mensajes guardados en la cola al server XMPP para establecer conexión.

g. Monitoreo Remoto

El proxy puede ser monitoreado y configurado desde el puerto 5224. Desde aquí se pueden obtener las métricas recolectadas y configurar las propiedades del server.

La definición y modo de utilización del protocolo de administración se detalla en Protocolo de administración

Para acceder al mismo es necesario autenticarse.

h. Librerías Externas

Se utilizaron las siguientes librerías externas para la implementación del proxy:

- Logback V 1.1.2
- Google Guava V 19.0
- Apache Commons V 3.4

- Commons codec 1.10

i. Silenciar usuarios

Se interpretó como usuario silenciado a aquel usuario que no puede enviar mensajes salientes a ningún usuario.

Para implementar esta funcionalidad se decidió crear un filtro llamado Silent User que contiene la información de los usuarios silenciados al momento. Al momento de arrancar el proy se lee una property del archivo de configuraciones para setear estos valores iniciales.

Si un usuario está silenciado y quiere enviar un mensaje saliente se le informa mediante un mensaje de error su condición de silencio.

j. Conversor L33t

Para implementar esta funcionalidad se creó un filtro llamado Transformations que se encarga de parsear el mensaje que envía el cliente XMPP y quedarse con la parte del cuerpo del mensaje, pudiendo así realizar los siguientes reemplazos:

- a por 4 (cuatro)
- e por 3 (tres)
- i por 1 (uno)
- o por 0 (cero)
- c por < (menor)

Las transformaciones que realiza el proxy se activan de manera global para todos los usuarios. Las mismas pueden activarse y/o desactivarse cambiando el archivo de configuraciones del proxy y modificando la propiedad *transformations_enabled*.

k. Registros de acceso

i. Sobre los logs

Para implementar los logs del proxy se decidió utilizar la librería Logback con el siguiente patrón para los logs:

```
%date{yy-MMM-dd HH:mm:ss}\t%-5level\t%msg%n
```

Los logs son guardados en un archivo en el root del proyecto llamado xmpp-server.log.

Cada log contiene: fecha de creación, tipo de log (debug, info, error, warn) y un mensaje.

A continuación se detallan todos los logs del proxy:

ii. Admin Logs

Inicio del administrador del proxy

```
YY-MM-DD HH:MM:SS INFO  Admin proxy started
```

Nueva conexión entre un admin y el proxy

```
YY-MM-DD HH:MM:SS DEBUG Accepted new admin connection from XXX.XXX.XXX.XXX:PORT
```

Modificar una propiedad del archivo de configuraciones

```
YY-MM-DD HH:MM:SS INFO  SET 200 - Updated [configurationProperty] configuration
```

Listar todas las configuraciones posibles

```
YY-MM-DD HH:MM:SS INFO GET 200 - all configurations
```

Listar una propiedad de configuración

```
YY-MM-DD HH:MM:SS DEBUG GET 200 -  [configurationProperty] configuration
```

Listar una propiedad de configuración que no existe

YY-MM-DD HH:MM:SS DEBUG GET 403 - [configurationProperty] configuration not found

Login exitoso

YY-MM-DD HH:MM:SS DEBUG LOGIN 201 - Login successful

Login fallido

YY-MM-DD HH:MM:SS INFO LOGIN 401 - Login failed

Comando no encontrado

YY-MM-DD HH:MM:SS ERROR 400 - Bad request

Conexión cerrada por el cliente

YY-MM-DD HH:MM:SS INFO LOGOUT 200 - Connection closed by admin

Error al intentar cerrar la conexión

YY-MM-DD HH:MM:SS ERROR Error trying to close connection [Socket]

iii. Client logs

Inicio proxy

YY-MM-DD HH:MM:SS INFO Client proxy started

Nueva conexión entre proxy y cliente

YY-MM-DD HH:MM:SS DEBUG Accepted new client connection from XXX.XXX.XXX.XXX:PORT to XXX.XXX.XXX.XXX:PORT

Nuevo usuario intentando conectarse

YY-MM-DD HH:MM:SS INFO Client [jidClient] is attempting to connected

Conexión de un cliente

YY-MM-DD HH:MM:SS INFO Client [jidClient] has connected

Desconexión de un cliente

YY-MM-DD HH:MM:SS DEBUG Client [jidClient] has connected

YY-MM-DD HH:MM:SS DEBUG XMPP Proxy [XXX.XXX.XXX.XXX:PORT] to XMPP Server [XXX.XXX.XXX.XXX:PORT] socket closed

Error al cerrar conexión

YY-MM-DD HH:MM:SS ERROR Error closing channels

Mensaje filtrado

YY-MM-DD HH:MM:SS INFO Message from [jidClient] filtered

I. Protocolo de administración

La aplicación del protocolo de administración permite manejar las distintas funcionalidades del proxy.

Para esto se implementó un protocolo del tipo request-response. El protocolo que se diseñó requiere de autenticación para poder modificar y/o consultar las configuraciones. Luego de que el administrador se autentica de manera correcta, ya está habilitado para poder operar sobre las configuraciones del proxy.

Los comandos que recibe el proxy son interpretados por línea y las respuestas que otorga el proxy están formadas por un código de error/éxito seguido de ¨-¨ y luego un mensaje descriptivo.

Se decidió que el protocolo no fuera case sensitive, pudiendo escribir los comandos ya sea en mayúscula o en minúscula.

Las operaciones que puede realizar un administrador son:

i. Help

Sintaxis

GET

Descripción

Cuando se ingresa get se muestran todas las propiedades configurables del proxy

Respuesta

200 - Configuration updated

ii. Habilitar / Deshabilitar silenciado

Sintaxis

SET silenceuser_enabled=true

Descripción

Cuando esta configuración se setea en true, la funcionalidad del silenciado está habilitada. Con cualquier otro valor la funcionalidad estará desactivada.

Respuesta

200 - Configuration updated

iii. Silenciar un usuario

Sintaxis

SET filter_silenceuser=usuario@server.com

Descripción

Silencia al usuario usuario@server.com

Respuesta

200 - Configuration updated

iv. Desilenciar un usuario

Sintaxis

UNSET filter_silenceuser=usuario@server.com

Descripción

Desilencia al usuario usuario@server.com

Respuesta

200 - Configuration updated

v. Habilitar / Deshabilitar silenciado

Sintaxis

SET silenceuser_enabled=true

Descripción

Cuando esta configuración se setea en true, la funcionalidad del silenciado está habilitada. Con cualquier otro valor la funcionalidad estará desactivada.

Respuesta

200 - Configuration updated

Habilitar / Deshabilitar multiplexor

Sintaxis

SET multiplexing_enabled=true

Descripción

Cuando esta configuración se setea en true, la funcionalidad del silenciado está habilitada. Con cualquier otro valor la funcionalidad estará desactivada.

Respuesta

200 - Configuration updated

Multiplexar un usuario

Sintaxis

SET filter_multiplexing=usuario@server.com

Descripción

Multipliexa al usuario usuario@server.com

Respuesta

200 - Configuration updated

Desmultiplexar un usuario

Sintaxis

UNSET filter_silenceuser=usuario@server.com

Descripción

Desmulriplexa al usuario usuario@server.com

Respuesta

200 - Configuration updated

Definición en formato ABNF

```
s                   =       command_line / response
command_line        =       command [SP params] CRLF;
params              =       key["=" value];
command             =       ("login"/"logout"/"get"/"set"/"unset");
key                 =       alphanumeric;
value               =       alphanumeric;
response            =       code SP "-" SP (metrics/list/text) CRLF;
code                =       3DIGIT;
metrics             =       CRLF "metrics: {" CRLF metrics_list "}"  CRLF;
metrics_list        =   *( %x09 text ":" SP alphanumeric"," CRLF);   text is key number
value
list                =       0*(text "," SP) text
alphanumeric        =       1*(ALPHA/DIGIT);
number              =       1*DIGIT;
text                =       1*CHAR;
```

Problemas encontrados

- El primer problema desarrollando el conversor l33t. Al utilizar Adium cuando el cliente desea enviar el carácter **<**, lo convierte automáticamente a < El problema radica cuando se quería enviar un **&** y Adium lo enviaba como **&amp**, al querer reemplazar todas las A por un 4 se perdía el carácter & que se había enviado inicialmente.
- Otro problema se presentó en el filtro para silenciar usuarios. No se guardaba el jid de los clientes que estaban conectados. Por lo que para implementar dicha funcionalidad era sumamente necesario saber con qué cliente estaba hablando el proxy. Lo que se hizo es guardar el contenido del tag jid que estaba dentro del flujo de autenticación

- Otro problema surgió cuando se implementó multiplexación ya que tampoco se tenía el jid antes de iniciar sesión. Pero se descubrió que al utilizar autenticación plana dentro del tag auth se encuentra el usuario y contraseña del cliente.

Limitaciones de la aplicación

- Al utilizar una autenticación plana solo se puede conectar a servidores que trabajen con el tag auth. Si el server le responde con un challenge al cliente nuestro proxy no podría responder a esto ya que conlleva otro tipo de seguridad que no es soportado.

Posibles extensiones

- Mejorar las métricas añadiendo más valores y estadísticas
- Hacer que las métricas sean NO volátiles
- Soportar más de un usuario de admin
- Modificar más de una propiedad del archivo de configuración dentro del mismo comando. Hoy por hoy para modificar una propiedad hace falta ejecutar un comando por cada modificación

Conclusiones

Es muy interesante el poder entender el protocolo y así de esta manera poder manipular y transformar los mensajes provenientes de los clientes. Al principio no era una tarea del todo clara, si bien se entendía la consigna, era muy difícil pensar en cómo llevarla a cabo. Los logs del servidor XMPP ayudaron mucho a entender de qué se trataba el protocolo junto con los mensajes que pasaban por el proxy.
La mayor dificultad estuvo en poder desarollar un flujo completo entre el cliente xmpp y el servidor, pero antes de escribir cualquier línea de código fue clave diseñar la arquitectura.

Ejemplos de prueba

Para probar las distintas funcionalidades el proxy se sugieren los siguientes casos de prueba:

- **Silenciar un usuario**

- Abrir una conexión netcat **nc localhost 5224**
  Reemplazar localhost por el host donde este corriendo el proxy
- Loguearse ingresando el comando **login admin=123**
- Ingresar el comando **SET filter_silenceuser=usuario@server.com**
  Reemplazar usuario@server.com por el usuario que se quiere silenciar
- El protocolo debe retornar una respuesta con código 200 informando que la configuración fue actualizada

- **Conectarse con un usuario silenciado**

- Configurar Adium con un nuevo usuario con las siguientes configuraciones
  A. Jabber ID: El usuario que se quiere crear
  B. Password: La contraseña para dicha cuenta
  C. Connect server: localhost (Donde está corriendo nuestro proxy)
  D. Port 5222
  E. Register Account: nowixmppserver puerto: 5222
- A continuación iniciar sesión con el usuario creado

- **Escribir un mensaje desde un usuario silenciado**

- Iniciar sesión con un usuario que esté silenciado
- Escribir un mensaje a otro usuario
- El proxy debería informarle al usuario que se encuentra silenciado

- **Cambio de configuraciones del proxy**

- Abrir una conexión netcat **nc localhost 5224**
  Reemplazar localhost por el host donde este corriendo el proxy
- Loguearse ingresando el comando **login admin=123**
- Multiplexar un usuario ingresando el comando **SET filter_multiplexing=usuario@host**

Reemplazar usuario por el jid del usuario que se quiere multiplexar y host por el host correpondiente

- **Verificar los logs**

- Dentro del root del proyecto se encuentra un archivo llamado xmpp-server.log donde estan los logs del proyecto
- Hacer un tail -f del archivo para verlo actualizado

- **Métricas**

- Abrir una conexión netcat **nc localhost 5224**
  Reemplazar localhost por el host donde este corriendo el proxy
- Loguearse ingresando el comando **login admin=123**
- Solicitar las métricas ingresando **GET metrics**

Guía de instalación detallada y precisa.

En primer lugar se necesita un cliente XMPP, en el caso de utilizar OSX recomiendo Adium y en el caso de Linux Pidgin. Por otro lado también se necesita un servidor XMPP el cual se eligió Prosody por la fácil configuración.

Una vez descargado Prosody se debe reemplazar el archivo de configuración por el mencionado en Configuración Server XMPP

Luego se procede a la creación de un cliente XMPP completando los siguientes campos:

F. Jabber ID: El usuario que se quiere crear
G. Password: La contraseña para dicha cuenta
H. Connect server: localhost (Donde está corriendo nuestro proxy)
I. Port 5222
J. Register Account: nowixmppserver puerto: 5222

Es importante deshabilitar la opción Require SSL/TSL

Ahora si, nos encontramos en condiciones de correr el proxy.

Si se desea modificar variables de del proxy se puede hacer como lo indica Protocolo de administración

Instrucciones para la configuración.

Para la configuración del Proxy en primer lugar se debe iniciar una conexión netcat: **nc localhost 5224** que es donde está corriendo el proxy para ser configurado.

Dicho protocolo requiere autenticación así que es importante loguearse con las credenciales provistas de la siguiente manera:

```
login admin=123
```

Siendo admin el nombre del usuario admin y 123 su clave correspondiente.

Luego se pueden configurar las variables del proxy tal como lo indica Protocolo de administración

Para finalizar la conexión del administrador sólo hace falta escribir el siguiente comando

```
logout
```

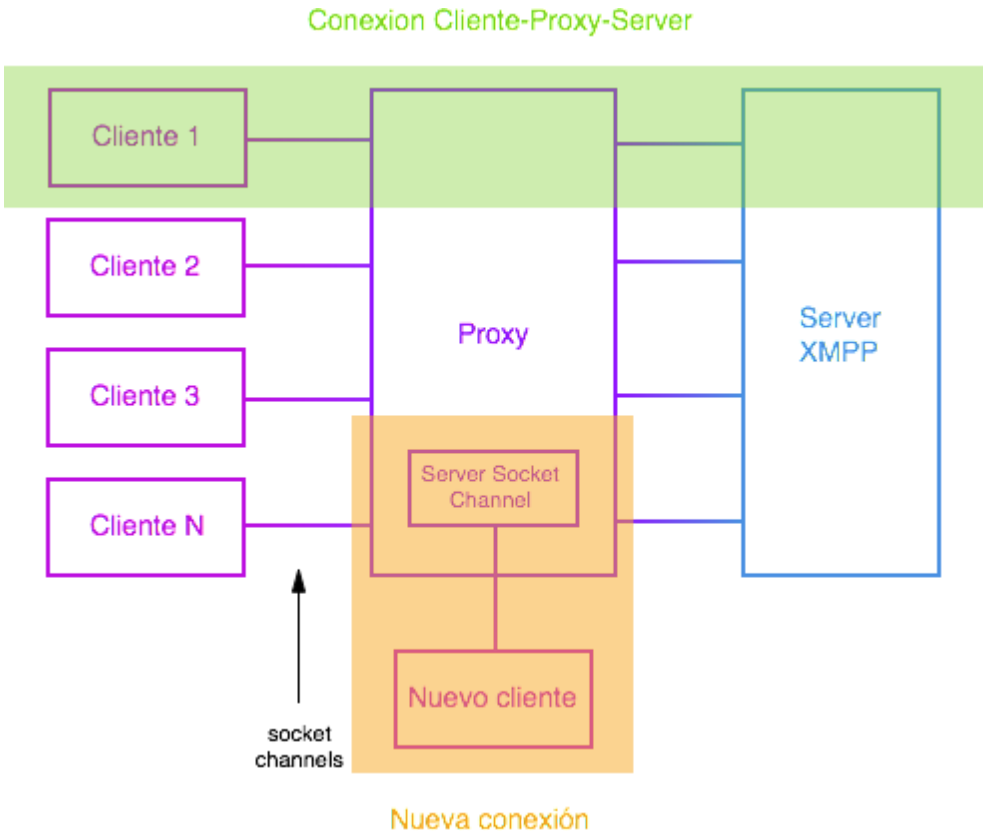
Ejemplos de configuración y monitoreo.

A continuación se muestran ejemplos del uso del protocolo de administración y algunos comandos disponibles

```
➜  ~ nc localhost 5224
Bienvenido a la administración del proxy!
login admin=123
201 - Login succesfull
get
200 -
admin_password,filter_silenceuser,admin_user,xmpp_server_host,xmpp_server_hostna
me,transformations_enabled,xmpp_proxy_host,xmpp_server_port,filter_multiplexing,xm
pp_proxy_port,multiplexing_enabled,xmpp_proxy_admin_port,silenceuser_enabled
get filter_silenceuser
200 - pepe@localhost2,noelia.lopez@nowixmppserver
get filter_multiplexing
200 - otromas@localhost:5223,user@localhost:5223,otromas@localhost\:5223
esteesunxomandoquenoexiste
400 - Command Unknown
set filter_silenceuser=silence@nowixmppserver
200 - Configuration updated
get filter_silenceuser
200 - pepe@localhost2,noelia.lopez@nowixmppserver,silence@nowixmppserver
set silenceuser_enabled=true
200 - Configuration updated
get silence_enabled
403 - Not found
get silenceuser_enabled
200 - true
```

Documento de diseño del proyecto

Este proxy cuenta con la siguiente arquitectura:



En primer lugar en la columna izquierda vemos todos los clientes que están conectados a nuestro proxy mediante un socket channel. Cada socket channel es distinto según el cliente.

En segundo lugar en la columna de la derecha vemos otros sockets channels que simbolizan la conexión de cada cliente con el servidor xmpp. Cada socket es distinto según el cliente.

Es por esto que en color verde se simboliza lo que es el flujo entero de una conexión. Dos sockets channels, uno que va de cliente a proxy y otro que va de proxy al servidor XMPP.

En color naranja vemos el intento de una nueva conexión de un cliente. El nuevo cliente se conecta al Server Socket Channel del proxy. Este último se encarga de aceptar la conexión pedida por el cliente y al hacerlo se abre un nuevo socket entre cliente y proxy como se detalla anteriormente.

Correcciones 2-12-2016

Write no bloqueante

Se tomaron en cuenta las notas sobre la escritura bloqueante y la no verificación de si se había podido escribir todo y se corrigió mediante la registración de las keys en modo write y/o read. La comunicación del buffer para escribir se utilizó el attachment de la key tal como fue sugerido en las correcciones.

A modo de demostrar que es verdaderamente no bloqueante se achicó el tamaño del socket de salida para escribir a un número mucho menor al default (146988 bytes) y se lo dejó en 1024 bytes con el objetivo de forzar múltiples escrituras para un mismo mensaje.

Decidí no implementar lecturas parciales más allá de que entiendo que implementarlo conlleva un trabajo sobre el entendimiento de cuándo un mensaje XMPP termina pero debido a las condiciones del trabajo y por cuestiones de tiempo no fue implementado. Lo que hubiese hecho para poder implementarlo es integrar la lectura con un parser de XML sobre un buffer de lectura. Si el parser puede verificar que el mensaje es un mensaje bien formado se pasaría a procesar el mismo.

Para mejorar la optimización y performance se fuerza el encoding a UTF-8 y se usa un solo buffer para la lectura en contraprestación a la entrega anterior.

Psi

Se pudo verificar que los problemas ocurridos en la demostración se debieron a que psi no aceptaba el doble handshake para establecer una conexión.

Para multiplexar a un usuario fue fundamental habilitar la autenticación en plano para saber el jid del usuario antes de conectarlo con el server default. Lo que se hace ahora para obtener el nombre de usuario del cliente que se quiere conectar es simular un intento de conexión donde el proxy se hace pasar por el server respondiendole al cliente como si lo fuese, mientras tanto se guardan los mensajes enviados por el cliente en una cola. Una vez obtenido el jid del cliente pasan a enviarse los mensajes guardados en la cola al server XMPP para establecer conexión.

Prosody Server

Comencé a probar el servidor XMPP de manera local por lo que ya no se encuentra disponible el servidor XMPP en Amazon Web Services

Generación del JAR

Se deben ejecutar los siguientes comandos:

- mvn clean
- mvn install
- cp config.properties target/config.properties
- java -cp target/xmpp-proxy-server-1.0-SNAPSHOT-jar-with-dependencies.jar ar.edu.itba.proxy.MainProxy

Ejecución del JAR

Se creó un script de bash que permite generar el archivo ejecutable y correrlo. Lo que se debe hacer es ejecutar el archivo run.sh que se encuentra dentro del repositorio.

Configuración

Para debuggear fácilmente el proxy se incluyó una variable del tipo boolean que permite ser modificada a través del protocolo de admin. Esta variable cuando esta en true permite ver todos los prints del proxy.

Cambios de configuración en el servidor XMPP

También me pude dar cuenta que con adium, mi cliente XMPP, funcionaba el hecho de autenticarse en plano sin ninguna configuración extra en el servidor XMPP. Al querer usar psi, este no recibía por parte del servidor XMPP el mechanism PLAIN. Lo que hice fue cambiar un par de configuraciones de prosody para que permita la conexión de los usuarios autenticarse en plano.

Nueva configuración

```
admins = { "admin@nowixmppserver" }
authentication = "internal_plain"
allow_registration = true;
modules_enabled = {
  "roster"; -- Allow users to have a roster. Recommended ;)
  "register"; -- Allow users to register on this server using a client and change passwords
  "admin_adhoc"; -- Allows administration via an XMPP client that supports ad-hoc
commands
  -- "legacyauth";
  "saslauth"; -- Authentication for clients and servers. Recommended if you want to log
in.
  --"admin_telnet"; -- Opens telnet console interface on localhost port 5582
  -- "watchregistrations"; -- Alert admins of registrations
}
log = {
  -- "*syslog"; -- Uncomment this for logging to syslog
  debug = "/var/log/prosody/prosody.log";
  error = "/var/log/prosody/prosody.err";
}
c2s_ports = { 5222 }
daemonize=false
allow_unencrypted_plain_auth=true
-- disable_sasl_mechanisms = { }
VirtualHost "nowixmppserver"
```