Protocolos de Comunicación

Trabajo Especial

María de la Puerta E. - 50009

Alexis Medvedeff - 50066

Juan Pablo Rey - 50265

Federico Bond - 52247

10/06/2015

Índice

Objetivo

El objetivo del trabajo es implementar un servidor proxy para el protocolo XMPP (Extensible Messaging and Presence Protocol) [RFC6120] que pueda ser usado por clientes XMPP como Pidgin y Psi para el envío y recepción de mensajes de chat. El proxy proveerá al usuario algunos servicios extras que el servidor de origen XMPP no provee (como ser la manipulación del contenido del mensaje).

Características del servidor proxy a implementar

- El servidor proxy DEBE soportar múltiples clientes de forma concurrente y simultánea.
- DEBE tener en cuenta en la implementación aquellos factores que afecten la performance.
- El servidor proxy DEBE reportar los fallos a los UA usando toda la potencia del protocolo XMPP.
- El servidor proxy DEBE dejar registros de los accesos en la consola y/o en un archivo que permitan entender que requests están pasando por el proxy y su resultado.
- El sistema DEBE implementar mecanismos que recolecten métricas para entender el funcionamiento del sistema. Las métricas incluyen cantidad de accesos al servidor y de bytes transferidos y cualquier otra que se considere relevante para el entendimiento dinámico del sistema.
- Se DEBE implementar mecanismos que permitan configurar el sistema para que un JID sea *mapeado* a un servidor origen distinto del *default*.
- El sistema DEBE implementar mecanismos que permitan filtrar los mensajes entrantes y salientes a un cierto usuario.
- Se DEBE implementar transformaciones del texto de los mensajes utilizando el formato *l33t*.
- Adicionalmente se implementará la extensión SI File Transfer.
- La configuración referida a transformaciones, multiplexado, etc., DEBE poder ser modificada en tiempo de ejecución de forma remota.
- El servidor DEBE exponer un servicio (para el cual se DEBE proveer un protocolo) para que sea posible monitorear el funcionamiento del mismo. El mismo DEBE proveer acceso a las estadísticas recolectadas.

Descripción de los protocolos y aplicaciones desarrolladas

Sockets

El proxy se desarrolló utilizando sockets no bloqueantes, particularmente *Java NIO*. Se eligió esta opción ya que los sockets no bloqueantes son más performantes en servidores concurrentes. Por otro lado, debido a que la implementación de conexiones persistentes era de carácter obligatorio, nos inclinamos aún más por esta opción.

Comportamiento común entre protocolos

Tanto el protocolo XMPP como el de administración tiene partes en común. Ambos se basan en XML a lo largo de toda la comunicación. Para manejar esto, implementamos una estructura de capas dada por las siguientes:

- TCP
- XML
- XMPP / "Admin"

En la capa TCP se encuentran las clases responsables de establecer la conexión entre el cliente y el proxy y entre el proxy y el servidor de origen. Estas se basan en abstracciones sobre el modelo de streams y permiten pensar toda la comunicación como "conversaciones" full-duplex TCP que contienen dichos streams (uno en cada dirección). De esta forma pudimos también modelar las conversaciones de capas superiores como streams (de xml, de XMPP y de admin).

En la capa XML utilizamos la librería Aalto (versión 0.9.11). Hay básicamente dos tipos de librerías para manejo de XML. Las primeras son estilo DOM que se arman en memoria el árbol completo representado por el XML pero no resultan útiles para documentos muy grandes. Por otro lado, existen las orientadas a eventos como SAX donde es posible ir procesando el documento e ir leyendo eventos que representan los elementos que van apareciendo. AAlto nos permite operar orientado a eventos de forma no bloqueante. De esta forma, a medida que van llegando los comandos de los clientes o del servidor, los vamos interpretando y actuando en consecuencia.

En la última capa se encuentran el protocolo XMPP y el protocolo de administración definido en este trabajo. Ambos usan un formato XML y es por esto que corren encima de la capa de manejo de XML, compartiendo toda la funcionalidad de *parseo* de XML.

En el gráfico 2, presente en la sección correspondiente, se puede ver una representación de las clases más importantes donde se ven plasmadas estas tres capas.

Protocolo de administración

Para la configuración del proxy definimos un protocolo con sintaxis XML, cuyos comandos consisten en *tags* que pueden contar con atributos y son *case sensitive*. El contenido de los *tags* define la configuración de ciertos parámetros, ya sea la activación de la transformación de mensajes a formato *l33t*, el multiplexado de cuentas, entre otras posibilidades.

Para utilizar el administrador, luego de conectarse al puerto correspondiente, se debe declarar en primer lugar al XML de la siguiente forma: `<?xml version=1.0?>`. A continuación, se debe abrir un *tag* `<admin>` bajo el cual se establecerá toda la secuencia de mensajes. Luego, es necesario validarse como administrador para poder establecer las configuraciones deseadas. Para cada comando ingresado, se obtendrá una respuesta en un *empty-element tag* que puede ser positiva (`<success/>`) o negativa (`<error/>`) o se obtendrá una respuesta más detallada en caso de ser un mensaje de información.

A continuación se muestran los comandos que implementa el protocolo de administración:

- `<usr>`*username*`</usr>`
- `<pass>`*password*`</pass>`
- *<leet>on|off</leet>*
- `<stats>`*1|2|3|4|5*`</stats>`
- `<silence value=`*”on|off”*`>`*username*`</silence>`
- `<origin usr="`*username*`">`*address*`</origin>`
- `<quit/>`

Las respuestas posibles son:

- `<success/>`
- `<error/>`
- `<error>`

```
        <code>error code</code>
        <message>error description</message>
</error>
```

El protocolo cuenta con distintos códigos de error, que se detallan a continuación:

- 100 - Malformed XML input
- 101 - Unrecognized tag
- 102 - Wrong silence value
- 103 - Wrong leet value
- 104 - Unknown error
- 105 - Wrong admin credentials

Registro de estadísticas

El comando `stats` del administrador permite acceder a estadísticas de uso del proxy. A medida que se realizan conexiones y se envían mensajes, se guardan ciertos datos que pueden ser útiles para el administrador. Se registran las siguientes estadísticas:

- Cantidad de accesos al sistema (`<stats>1</stats>`)
- Cantidad de bytes leídos (`stats>2</stats>`)
- Cantidad de bytes escritos (`<stats>3</stats>`)
- Cantidad de mensajes enviados por clientes (`stats>4</stats>`)
- Cantidad de mensajes recibidos por clientes (`stats>5</stats>`)

Pruebas de performance

Se cuenta con un script desarrollado en *Go* en la carpeta scripts el cual permite poner a prueba la performance del proxy. Esta prueba crea 20000 conexiones concurrentes en 10 workers con un solo thread. Se corrió en una computadora con procesador Intel® Core™ i5-3210M CPU @ 2.50GHz × 4 y 4GB de RAM sobre el sistema operativo Ubuntu versión 14.04. Utilizamos el comando time para calcular el tiempo que tardó la prueba. Los resultados son los siguientes:

Prueba	Tiempo
1	2.976s
2	2.976s

3	2.856s
4	3.093s

Tabla 1: resultados de pruebas de performance

Las pruebas realizadas permiten comprobar que el rendimiento del proxy es aceptable ante una gran cantidad de conexiones concurrentes.

Problemas encontrados durante el diseño y la implementación

Por la complejidad del protocolo XMPP y de la funcionalidad requerida decidimos desde un principio tomar un enfoque de diseño incremental y lo más declarativo que se pudiera. Esto nos permitió ser muy flexibles a la hora de agregar nueva funcionalidad (incluso ortogonal) y de encontrar bugs, pero complejizó considerablemente la estructura (generando incluso ciertas maestrías de dominio en zonas específicas del código).

Si bien la arquitectura hizo posible el testeo unitario de partes clave del sistema, esta no fue para nada sencilla: terminamos teniendo que desarrollar muchísimo código utilitario. En cuanto al testing de carga, la naturaleza misma del protocolo XMPP hizo que tuviéramos que desarrollar un sistema que no dependiera del *rendez-vous.*

Limitaciones de la aplicación

La implementación tiene algunas limitaciones en su funcionamiento. En primer lugar, el reactor no puede utilizar múltiples hilos de ejecución por lo que la performance cuenta con cierta limitación.

En segundo lugar, no se soportan mecanismos seguros de autenticación de usuarios. Debido a que estos procedimientos tienen más complejidad y exceden el objetivo de este trabajo.

Librerías utilizadas

Utilizamos las siguientes librerías para tareas comunes:

- Aalto XML: para parsing no bloqueante de streams XML
- Apache Commons: para codificación y decodificación Base64 para las credenciales de autenticación XMPP.
- JUnit / Mockito: para testing.
- SLF4J: para logeo.

Posibles extensiones

Una posible mejora para este sistema es implementar al reactor con múltiples *threads*. Esto permitiría mejorar la performance y escalabilidad del proxy.

Otra extensión consiste en implementar los mecanismos de autenticación seguros que admite el protocolo XMPP. Estos son comúnmente usados por los clientes pero no fue implementado ya que excede el alcance del trabajo realizado.

Por otro lado, en lo que al protocolo de administración respecta, se podría implementar utilizando estándares XMPP, particularmente "XEP-0050: Ad-Hoc Commands".

Conclusiones

Gracias a este trabajo práctico ganamos experiencia en el desarrollo de aplicaciones de red concurrentes. También aprendimos un protocolo de amplia difusión como es XMPP y apreciamos los desafíos involucrados en el desarrollo de aplicaciones de alto rendimiento para sistemas en red.

Guías de instalación y configuración

---

1. El archivo de configuración del proxy se encuentra en `src/main/resources` y se llama *proxy.properties*. Aquí se deben setear los puertos, credenciales de acceso al administrador y las demás propiedades allí presentes.
2. Compilar el proyecto con maven: `mvn clean package`.
3. Correr con `./bin/gossip`.

Ejemplos de configuración y monitoreo

C:Cliente - P:Proxy

Ejemplo 1 - Log in

```
C:<?xml version=”1.0”?>
C:<admin>
P:<?xml version=”1.0”?>
C:<usr>admin</usr>
P:<success/>
C:<pass>1234</pass>
P:<success/>
```

Ejemplo 2 - Error por tags mal formados

```
C:<?xml version="1.0"?>
C:<admin>
P:<?xml version="1.0"?>
C:<usr>admin</user>
P:<error>
       <code>100</code>
       <message>Malformed XML input</message>
 </error>
```

Ejemplo 3 - Comando stats

```
C:<?xml version="1.0"?>
C:<admin>
P:<?xml version="1.0"?>
C:<usr>admin</usr>
P:<success/>
C:<pass>1234</pass>
P:<success/>
C:<stats>1</stats>
P:<stats>
        <type>1</type>
```

```
        <desc>Number of read bytes</desc>
        <value>100</value>
  </stats>
C:<stats>3</stats>
P:<stats>
        <type>3</type>
        <desc>Number of connections to proxy</desc>
        <value>1</value>
  </stats>
```

Ejemplo 4 - Comando leet

```
C:<?xml version="1.0"?>
C:<admin>
P:<?xml version="1.0"?>
C:<usr>admin</usr>
P:<success/>
C:<pass>1234</pass>
P:<success/>
C:<leet>on</leet>
P:<success/>
C:<leet>off</leet>
P:<success/>
```

Gráficos

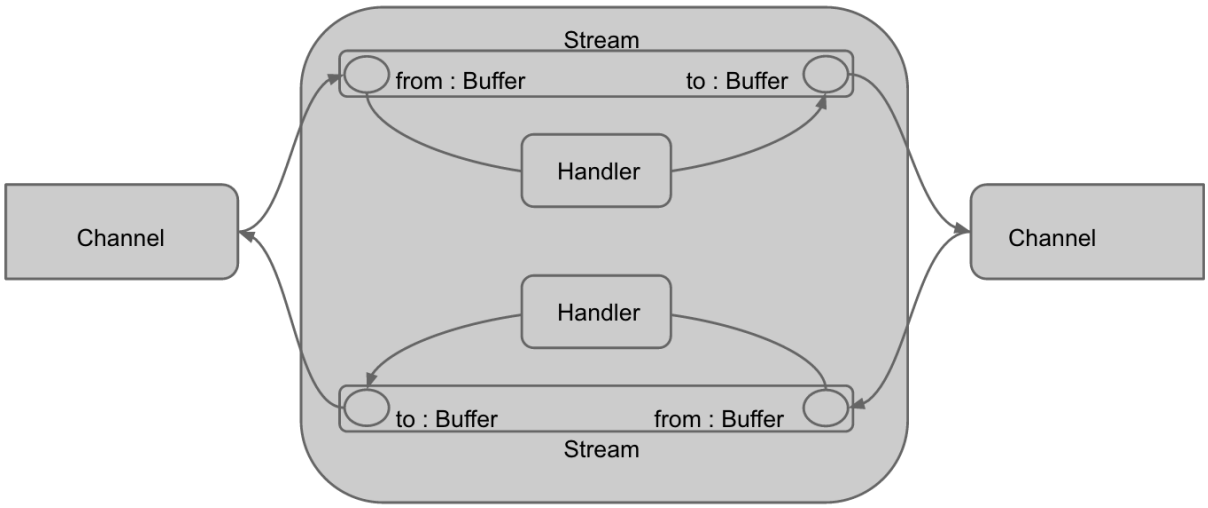

***Gráfico 1**: Estructura general de conexión cliente-servidor*

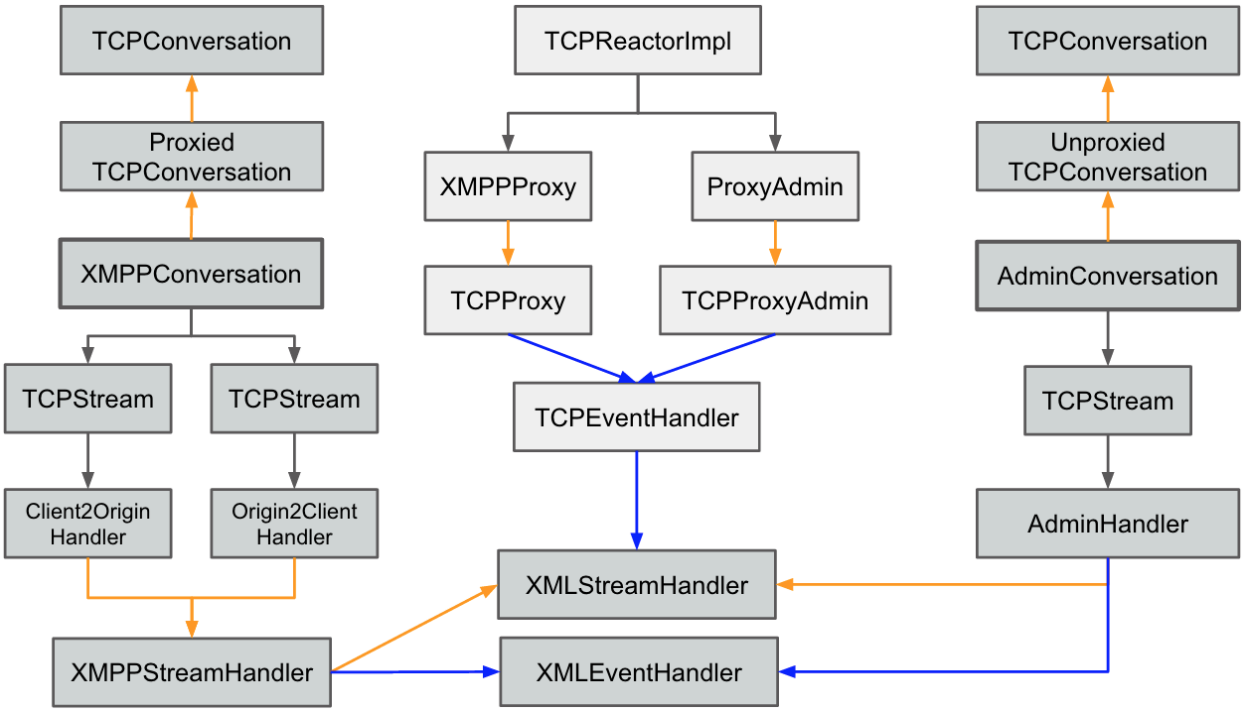

***Gráfico 2**: Vista general de clases. Azul: "implementa", Naranja: "extiende", Negro: "tiene".*