Diseño y Desarrollo de una Aplicación P2P de Mensajería para la ESPOL, Usando Tecnología JXTA

Xavier Fernando Calle Peña
Vanessa Inés Cedeño Mieles
Cristina Lucía Abad Robalino, Ing.
Facultad de Ingeniería en Electricidad y Computación
ESCUELA SUPERIOR POLITECNICA DEL LITORAL
Campus Gustavo Galindo, Guayaquil, Ecuador
xcalle@fiec.espol.edu.ec
vcedeno@fiec.espol.edu.ec
cabad@fiec.espol.edu.ec

Abstracto

Este resumen presenta una aplicación de mensajería instantánea par-a-par (P2P) para la ESPOL. Como las políticas de la institución impiden la utilización de aplicaciones de comunicación en las salas de cómputo, decidimos presentar nuestra aplicación desarrollada en JXTA como una alternativa. La tecnología JXTA es un grupo de protocolos que permite a cualquier dispositivo conectado a la red comunicarse y colaborar de una manera P2P. Los nodos de JXTA, también llamados peers, crean una red virtual superpuesta donde cada peer puede interactuar con otros peers y recursos directamente incluso cuando algunos de ellos se encuentran detrás de firewalls y NATs o se encuentran en diferentes redes. Queremos otorgar a los estudiantes una herramienta de colaboración académica la cual les permita mantener una conversación en línea con múltiples usuarios de todos los puntos de la institución como son oficinas administrativas, facultades, laboratorios. Con nuestra aplicación de mensajería basada en JXTA no dependemos de un servidor central, diferenciándonos de otras aplicaciones de mensajería disponibles. La red P2P se basa en una serie de nodos que se comportan a la vez como clientes y como servidores de los demás nodos de la red. Cualquier nodo puede iniciar o completar una transacción compatible. Los nodos pueden diferir en configuración local, velocidad de proceso, ancho de banda de su conexión a la red y capacidad de almacenamiento. Estas características conllevan a ventajas de tolerancia a fallos, escalabilidad, bajo costo de implementación, mejor rendimiento y alta disponibilidad de recursos.

Abstract

This article presents a Peer-to-Peer (P2P) application for instant messaging for ESPOL. Since the policies prevent the use of these kind of applications in the computer labs, we decided to present our application developed in JXTA as an alternative. JXTA is a set of protocols that allows any device connected to the network, to communicate and collaborate in a peer-to-peer (P2P) manner. JXTA nodes, also called peers, create a virtual overlay network where each peer can interact with other peers and resources directly even if some of them are behind firewalls and NATs or belong to a different network. We want to provide students with a collaborative academic tool which allows them to chat with multiple users from any point of the Institution (e.g., administrative offices, departments, and laboratories). Our JXTA-based instant messaging application does not depend on a central server. The P2P network is based on a series of nodes that behave at the same time as clients and servers. The nodes can differ in local configuration, processing speed, bandwidth and storage capacity. This yields improved fault tolerance, scalability, low implementation costs, good performance and good resource availability.

1. Introducción

La mensajería instantánea es un servicio que permite la colaboración entre usuarios y puede ser utilizado en instituciones académicas para incentivar la colaboración en las áreas académicas y de investigación. Lastimosamente, estos servicios frecuentemente son bloqueados en empresas y universidades con la finalidad de evitar que los usuarios de la institución pierdan tiempo y disminuyan su productividad.

Surge entonces la necesidad de un servicio de mensajería cuyo acceso pueda ser restringido a los usuarios de La ESPOL para que puedan comunicarse entre ellos únicamente, mas no con usuarios externos. Como veremos más adelante, existen alternativas de mensajería instantánea, pero ninguna del todo idónea para ser utilizada en instituciones universitarias con miras a la colaboración académica y sin permitir comunicaciones con usuarios externos. Por esta razón, hemos desarrollado una aplicación de mensajería instantánea universitaria, utilizando una arquitectura par-a-par (P2P) [1], la cual es

de fácil administración, escalable y flexible. Esta aplicación de mensajería basada en JXTA [2] no depende de un servidor central. Los nodos de la red P2P se comportan a la vez como clientes y como servidores de los demás nodos.
El resto de este artículo está estructurado de la siguiente manera: en la Sección 2, discutimos los antecedentes y motivación del problema. En la Sección 3 describimos la arquitectura seleccionada, mientras que en la Sección 4 detallamos el diseño planteado. Los detalles de rendimiento se presentan en la Sección 5. Finalmente, en la Sección 6 concluimos y detallamos el trabajo futuro.

2. Motivación

Las políticas en la ESPOL especifican que en el interior de las salas de los centros de cómputo se prohíba el uso de cualquier programa o sitio de comunicación en línea, como por ejemplo MSN Messenger, Yahoo Messenger, etc. Este tipo de políticas buscan evitar que los estudiantes se distraigan u ocupen el tiempo de máquina en actividades no relacionadas con la institución.
Lastimosamente, este tipo de políticas también disminuyen las posibilidades de colaboración productiva entre los usuarios de la institución. Por esta razón, surge la necesidad de una herramienta de mensajería que pueda ser restringida a los usuarios de la institución.
Un ejemplo de un tipo de aplicación de mensajería que puede ser utilizada únicamente entre los usuarios de una institución es Jabber [3], el cual es un protocolo libre gestionado por Jabber Software Foundation basado en el estándar XML para mensajería instantánea y el cual podría adaptarse para desarrollar una aplicación estrictamente universitaria. El problema de Jabber es que utiliza una arquitectura cliente/servidor, lo cual otorga un punto único de administración y de fallo.
Queremos otorgar una aplicación alternativa de mensajería, que pueda ser restringida a los usuarios de la institución, proporcione mecanismos de colaboración que incentive la colaboración entre usuarios de la institución, y que a la vez sea escalable y tolerante a fallos. Con esta finalidad hemos diseñado y desarrollado una aplicación de mensajería instantánea universitaria basada en JXTA, la cual al utilizar la tecnología P2P no necesita de un servidor central. Nuestra aplicación permitirá a los estudiantes poder comunicarse con sus compañeros universitarios con la finalidad de incentivar la colaboración durante los estudios, investigaciones y demás actividades universitarias.
Actualmente existen otros sistemas de mensajería instantánea basados en tecnología JXTA disponibles para descargar de Internet como MyJXTA [4] y JIM (JXTA Instant Messenger) [5]. Estos otorgan la funcionalidad de un chat P2P pero no cumplen los requerimientos para ser implementados en instituciones académicas ya que cualquiera puede acceder a ellos y no buscan maximizar la colaboración académica. Por estas razones, hemos diseñado una aplicación de mensajería instantánea cuyo uso será restringido para que solamente permita conectarse a los estudiantes y demás miembros de la Universidad, validando el ingreso con la información de las cuentas de usuarios de la base de datos del CSI.

3. Arquitectura P2P

Una red P2P se basa en una serie de nodos que se comportan a la vez como clientes y como servidores de los demás nodos de la red. De esta manera, aún cuando uno o más nodos fallen o se desconecten, los demás nodos pueden seguir comunicándose entre sí. Un esquema P2P proporciona varias ventajas con respecto a un enfoque cliente/servidor, como una muy buena tolerancia a fallos, gran escalabilidad, bajos costos de implementación, buen rendimiento y alta disponibilidad de recursos.
Como plataforma P2P seleccionamos el proyecto JXTA de Sun Microsystems. El nombre JXTA se deriva de la palabra juxtapose (yuxtaponer) que significa colocar dos entidades lado a lado o en una proximidad. Al escoger este nombre el equipo desarrollador de Sun reconoció que las soluciones P2P siempre existirán al lado de las soluciones cliente/servidor, en vez de reemplazarlas completamente.
El Framework .NET de Microsoft [6], así como JXTA, provee una rica plataforma para la creación de aplicaciones P2P. Fundamentalmente JXTA y .NET tienen propósitos completamente diferentes, .NET se enfoca más en la arquitectura tradicional de cliente/servidor de entrega de servicios. Aunque la tecnología .NET puede formar las bases para una aplicación P2P esta solución demandaría recrear todos los mecanismos que ya están definidos por los protocolos de JXTA. Por esto escogemos la tecnología JXTA, porque nos permite enfocarnos en los aspectos de diseño propios de la aplicación sin perder tiempo en detalles de mecanismos de comunicación P2P.

3.1. Arquitectura JXTA [7]

La arquitectura de software JXTA está dividida en tres capas y provee un grupo de protocolos y una referencia de implementación de código abierto (ver Figura 1).

Figura 1. Arquitectura de JXTA [8].

La Capa de Plataforma (Núcleo de JXTA) encapsula propiedades comunes a la red P2P: incluye bloques para permitir mecanismos clave para aplicaciones P2P, como descubrimiento (discovery), transporte (incluye atravesar firewalls y cajas NAT), la creación de peers y grupos de peers.
La Capa de Servicios incluye servicios de red que pueden no ser absolutamente necesarios para que una red P2P pueda operar, pero son comunes o deseados en el ambiente P2P. Ejemplos de servicios de red incluyen buscar e indexar, directorio, sistemas de almacenamiento, mecanismos para compartir archivos, sistemas de archivos distribuidos, agregación de recursos y renta, traslado de protocolos, autenticación y servicios de PKI (infraestructura de clave pública).
La Capa de Aplicaciones incluye implementación de aplicaciones integradas, como mensajes instantáneos P2P, compartir documentos y recursos, manejar y entregar contenido de entretenimiento, sistemas de mail P2P y muchos otros. La frontera entre servicios y aplicaciones no es rígida. Una aplicación para un cliente puede ser vista como un servicio para otro cliente. Todo el sistema es diseñado para ser modular, permitiendo a los desarrolladores escoger entre una colección de servicios y aplicaciones que satisfagan sus necesidades.

3.2. Componentes de JXTA

La red JXTA consiste en una serie de nodos interconectados, también llamados peers. Los peers se pueden auto-organizar en grupos de peers, que proveen un grupo de servicios comunes. Ejemplos de servicios que pueden ser proveídos por un grupo de peers incluyen compartir documentos o aplicaciones de Chat.

Los peers de JXTA publican (advertise) sus servicios en documentos XML llamados publicaciones (advertisements). Las publicaciones permiten que otros peers en la red aprendan a cómo conectarse e interactuar con los servicios de los peers.
Los peers de JXTA usan tuberías (pipes) para enviar mensajes entre ellos. Los pipes son mecanismos de transferencia de mensajes asincrónicos y unidireccionales usados para el servicio de comunicación. Los mensajes son simplemente documentos XML cuyo sobre contiene información de enrutamiento y de credenciales. Los pipes están limitados a los puntos finales específicos como al puerto TCP y direcciones IP asociadas.

3.3. Aspectos claves de la arquitectura de JXTA

La arquitectura JXTA se distingue de otros modelos de redes distribuidas por:

- El uso de documentos XML (publicaciones) para describir recursos de red.
- Abstracción de pipes para peers, sin depender de un sistema de nombres central como un DNS (Domain Name Service).
- Un esquema uniforme de direccionamiento de peers (ID's de Peers).

4. Diseño

A continuación describimos los principales aspectos del diseño de nuestra aplicación de mensajería instantánea:

- chat de 1 a 1,
- chat entre múltiples usuarios simultáneamente,
- creación de grupos,
- agrupación de usuarios,
- cartelera comunal,
- envío de archivos,
- notificación del Estado de cada usuario, y
- búsqueda de usuarios conectados.

Para la autenticación de los usuarios en nuestra aplicación utilizamos el Servicio de Autenticación de la ESPOL (ver Figura 2). De esta manera se logra uno de los objetivos de la aplicación, el cual es proveer una herramienta de colaboración restringida a las personas que forman la comunidad universitaria. Este servicio Web verifica si el usuario y contraseña pertenecen a la base de datos de alumnos, profesores y trabajadores de la Institución.

Figura 2. Autenticación de usuarios.

Para la implementación del envío de un mismo mensaje a todo un grupo de conversación en la opción de chat de uno-a-muchos utilizamos una combinación de multicast ingenuo con diseminación por “chismes” [9], en la cual el origen envía una copia del mensaje a cada miembro del grupo. Esta solución es adecuada y eficiente para nuestra aplicación porque en una conversación uno a muchos no se suele involucrar una gran cantidad de usuarios en el grupo. Esta solución “ingenua” no es escalable a grupos grandes, pero esto no representa un problema ya que el escenario planteado no lo requiere. Si un usuario desea agregar contactos a una conversación éstos serán parte del árbol teniendo como nodo anfitrión al nodo que los invocó. En la Figura 3 se muestra cuando el usuario 1 agrega tres contactos a una conversación. Luego el usuario 3 agrega 2 usuarios más a la conversación.

Cuando el usuario 1 envía un mensaje se entrega una copia a cada uno de sus contactos invocados, y el usuario 3 hace lo mismo con sus contactos invocados.

Buscar a colaboradores es fácil, ya que se proporciona una opción que permite ver a todos los usuarios del sistema, e inclusive realizar búsquedas en base a ciertas propiedades de los usuarios (por ejemplo, por el usuario del colaborador). Si un usuario encuentra a un usuario con quien desea colaborar, lo puede añadir a su lista de contactos (ver Figura 4).

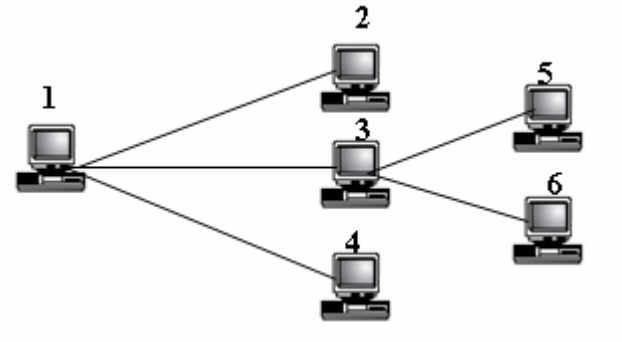

Figura 3. Contactos agregados a una conversación.

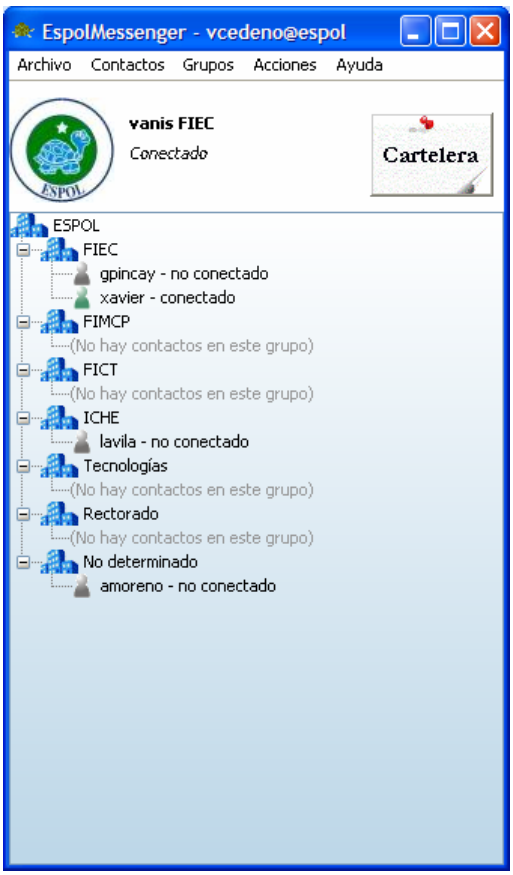

Figura 4. Aplicación de mensajería P2P.

Para incentivar la colaboración entre los usuarios de la red par-a-par de nuestro chat desarrollamos la cartelera comunal. Ésta es un espacio en el cual los usuarios pueden publicar o consultar mensajes escritos por otros usuarios. Para esto creamos el servicio de cartelera. Los datos del anuncio, como lo son, usuario, asunto, fecha/hora y mensaje, son escritos en un Advertisement y

publicados usando el servicio de Discovery, permitiendo que la información sea guardada en la caché por otros peers y replicada al resto. Con este propósito utilizamos advertisements de JXTA de vida larga los cuales permanecen en la caché de los peers por dos semanas. Para facilitar la consulta de los anuncios en la cartelera, se decidió que cada uno esté etiquetado con una estampa de tiempo indicando su hora de publicación. Pero en un sistema P2P, no hay un reloj que sea aplicable a todo el sistema, ya que cada nodo tiene su propio reloj, y puede darse el caso que uno o más nodos tengan configurada la hora del sistema de manera incorrecta. Para evitar inconvenientes de anuncios con estampas de tiempo incorrectas, la hora de publicación del anuncio de cartelera la obtenemos contactando a un servidor SMTP. En caso de que no haya un servidor SMTP en la red de la institución, se puede utilizar a cualquier otro servidor que retorne la hora del sistema en su saludo inicial.

En un diseño preliminar, nos encontramos con la situación de que cuando un usuario se conectaba a nuestra aplicación desde otra computadora que no sea la suya no obtenía la lista de contactos ya que la misma se almacenada en su computador. En una aplicación cliente/servidor la solución es sencilla ya que la lista de contactos se puede almacenar en el servidor y no en el computador del usuario. Pero en una aplicación P2P no existe un servidor central que pueda administrar las listas de contactos de los usuarios. Una solución es que cada vez que se haga una modificación en la lista se publique un advertisement con los cambios. Pero se corre el riesgo de que la próxima vez que el usuario se conecte no obtenga la última lista publicada sino una versión anterior y se representen resultados viejos, ya que el peer que la tiene en su caché no se encuentra conectado o no replicó la lista lo suficiente a sus demás peers para tener redundancia. En consecuencia tomamos la decisión de no incorporar la opción de almacenar la lista de contactos en un peer externo. Como solución alterna, se optó por permitir exportar la lista y posteriormente importarla. La lista puede ser exportada en un documento XML y almacenada en el disco duro o algún otro dispositivo externo. Pero exportar e importar la lista cada vez que el usuario se conecta resulta molestoso, y además, puede que cuando el usuario se conecte no tenga a la mano el dispositivo en donde almacenó la lista de contactos y por lo tanto no los pueda importar. Este problema lo solucionamos respaldando la lista en un correo electrónico enviado a la cuenta de e-mail de la ESPOL del usuario. De esta manera al abrir la aplicación la lista puede ser recuperada y cuando se cierra la aplicación, puede ser actualizada. El proceso es el siguiente: al cerrar una sesión nuestra aplicación crea un mensaje de correo que contiene la lista de contactos, el cual se envía al servidor SMTP (Simple Mail Transfer Protocol) abriendo un socket al puerto 25 del host smtp.espol.edu.ec. En el título del correo se especifica que es un mensaje que no debe ser borrado si se desea preservar la lista de contactos. Luego al iniciar una sesión, nuestra aplicación utiliza el mismo usuario y contraseña de autenticación para conectarse al servidor POP con el puerto 110 (pop.espol.edu.ec) para hacer una búsqueda en los mails del usuario y rescatar la lista de contactos. Cabe recalcar que esta información es enviada por un canal seguro, utilizando SSL. Si la lista de contactos no se puede recuperar del correo del usuario, se intenta recuperar la última lista que el usuario dejó en la computadora que está utilizando. Si no se recupera ninguna, se presenta una lista vacía, la cual puede ser llenada si el usuario la almacenó anteriormente, gracias a la opción de exportar que le brindamos.

5. Rendimiento

JXTA es escalable al alcance del sistema de mensajería Universitaria planteado. Para poder medir la capacidad de escalabilidad de JXTA, un ambiente de benchmarking fue construido en el laboratorio inalámbrico de Ingeniería de Sun Microsystems en Menlo Park, California [10]. El ambiente de prueba del benchmarking en el laboratorio inalámbrico de Sun consiste en una variedad de hardware corriendo Solaris 8, Solaris 9 y Linux. Una máquina controladora llamada jxta03 que corre Solaris 9 en una E450 con 4 procesadores, es usada para simular la distribución de peers y para controlar las pruebas de benchmarks.

Las Pruebas de Carga son para medir los límites de la carga JXTA que los súper peers pueden tolerar mientras escalan razonablemente bien. El término razonablemente bien es definido como manejo de carga sin perder ningún mensaje.

Se realizaron también pruebas de latencia en los cuales se mide el tiempo que le toma a un mensaje ir de un punto designado a otro. Se utiliza 5 TCP peers y 10 minutos de prueba.

Como se puede observar en la Figura 5 a medida que aumentan los mensajes enviados el tiempo de respuesta no crece demasiado. Se obtiene un tiempo mínimo de 27 milisegundos y uno máximo de 69970 milisegundos. Con la cantidad de 108576 mensajes se obtuvo un tiempo promedio de 9332.62 milisegundos. Esto muestra la escalabilidad de la red al aumentar la carga de los mensajes en la red JXTA.

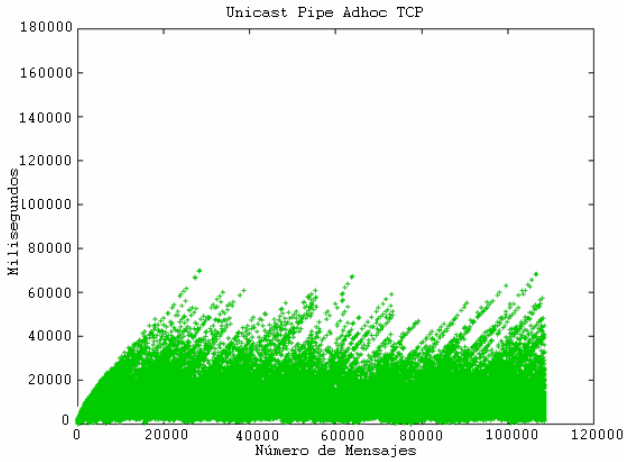

Figura 5. Pruebas de latencia de pipes entre peers. [11]

6. Conclusiones

En este resumen presentamos una aplicación de mensajería P2P desarrollada en JXTA para la ESPOL. JXTA es una tecnología en constante evolución que provee una capa de middleware la cual permite desarrollar de forma más eficiente aplicaciones que funcionarán en un ambiente peer-to-peer, ocupándose de aspectos tales como conexión, seguridad, localización, entre otros.
La aplicación de mensajería Universitaria propuesta demuestra las ventajas de utilizar JXTA para la implementación de servicios distribuidos que sean escalables, seguros y fáciles de administrar. Su implementación en los laboratorios y oficinas de la institución facilitará las actividades colaborativas entre los miembros de la comunidad, la cual se espera incida en un incremento de la calidad educativa e investigativa de la misma.

Tras haber hecho una investigación sobre demás trabajos en el área, podemos concluir que somos los primeros en sugerir las siguientes soluciones novedosas para el diseño de aplicaciones P2P empresariales y universitarias: multicast ingenuo para chat uno-a-muchos, contactar a un servidor SMTP para que proporcione un reloj global al sistema, y exportar/importar archivos de configuración vía SMTP/POP. Esas soluciones pueden ser adoptadas en otros sistemas P2P empresariales y universitarios, permitiéndoles mantener su naturaleza P2P y a la vez sacar ventaja de los servidores ya instalados en la institución.

7. Referencias

[1] Wikipedia La Enciclopedia Libre, “Peer-to-peer”, Artículo, http://es.wikipedia.org/wiki/Redes_P2P.
[2] JXTA Get connected, “Project JXTA”, http://www.jxta.org.
[3] Wikipedia La Enciclopedia Libre, “Jabber” Artículo, http://es.wikipedia.org/wiki/Jabber.
[4] Myjxta: Home, “Project Home”, http://myjxta.jxta.org/.
[5] Instant J.I.M Messenger: Home, “J.I.M - Latest Stuff”, http://jxtaim.sourceforge.net/index.html.
[6] Wilson, Brendon, “JXTA”, http://www.brendonwilson.com/projects/jxta-book/, Junio 2002, pp. 47.
[7] Sun Microsystems, “JXTA v2.3.x: Java™ Programmer’s Guide”, http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf, 7 Abril 2005, pp. 9-20.
[8] Sun Microsystems, “Peers on the Expanded Web”. Imagen. “JXTA v2.3.x: Java™ Programmer’s Guide”, http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf, 7 Abril, 2005. pp. 9.
[9] Y.-H. Chu, S. G. Rao, S. Seshan, H. Zhang. “A case for end system multicast”, IEEE J-SAC, 20(8), Octubre 2002, pp. 1456-1471.
[10] JXTA Get connected, “JXTA P2P Platform Benchmark Plan (Draft)”, http://bench.jxta.org/benchplan.html.
[11] JXTA Get connected, “JXTA Unicast Pipe Latency Test Results”. Imagen. “Unicast Pipe Adhoc TCP”, http://bench.jxta.org/loadtest_results/stable/ucplat.html.