ESTUDIO DE PROTOCOLOS DE BUSQUEDA P2P EN REDES INALAMBRICAS: CHORDWIRELESS, APLICACIÓN PARA COMPARTIR ARCHIVOS

POR

ZULAY QUIROZ CANDELARIO

Tesis de Maestría en Ingeniería de Sistemas

UNIVERSIDAD DE LOS ANDES

Enero de 2006

CONTENIDO

GUIA PARA EL LECTOR 4
RESUMEN 5
INTRODUCCION 6
DESCRIPCIÓN DEL PROBLEMA 7
OBJETIVOS 8
Objetivos Específicos 8
1 MARCO TEORICO 9
1.1 CONCEpTOS DE REDES P2P 9
1.1.1 Clasificación de ArquitecturasP2P 9
1.2 PROTOCOLOS DE BÚSQUEDA DISTRIBUIDA 13
1.2.1 Chord 13
1.2.2 CAN (Content Addressable Network) [3],[4] 20
1.2.3 JXTA [5,6] 27
1.3 SEGURIDAD PEER TO PEER 38
1.3.1 Diseño de Sistemas Seguros 40
1.4 APLICACIONES SEGURAS PEER TO PEER EXISTENTES 50
1.4.1 Proyecto SECURE 50
1.4.2 MOTION: Sistema De Control De Acceso [19] 52
1.4.3 SCISHARE [20] 54
2 DISEÑO 57
2.1 COMPARACION DE PROTOCOLOS DE BUSQUEDA 57
2.1.1 Complejidad 57
2.1.2 Escalabilidad 59
2.1.3 Capacidad de Procesamiento 60
2.1.4 Propuesta de Implementación 61
2.2 SEGURIDAD 63
3 IMPLEMENTACION 65
3.1 Prototipo 1 65
3.2 prototipo 2 66
3.3 pruebas 67
3.3.1 Consumo de Memoria 68

3.3.2 Tiempos de Procesamiento 71
3.3.3 Seguridad 73
3.4 CONSIDERACIONES ADICIONALES 74
4 CONCLUSIONES Y TRABAJOS FUTUROS 76
4.1 CONCLUSIONEs 76
4.2 TRABAJOS FUTUROS 77
ANEXO A. VIstA De LA APLICACION 78
ANEXO B. diagrama de clases prototipo 1 82
Bibliografía 84

GUIA PARA EL LECTOR

La siguiente guía permite al lector conocer de manera previa los capítulos de este reporte y la relación entre ellos

Resumen Ejecutivo: Encontrará un resumen de los objetivos, el proceso del trabajo de grado y el resultado de manera sucinta.

Capitulo 1 Introducción: Este capitulo es una introducción a la tesis, contiene la descripción del problema y los objetivos

Capitulo 2 Marco Teórico: Contiene información de los temas investigados previamente al diseño e implementación de dos prototipos. De igual forma ponen en contexto al lector acerca del diseño de sistema P2P

Capitulo 3 Diseño: Contiene el diseño propuesto para un sistema P2P móvil, sustentado en las definiciones del capitulo 2.

Capitulo 4 Implementación: Contiene los resultados de la implementación de los dos prototipos presentados en el capitulo 3, presentando una vista de la aplicación y las tecnología utilizadas en el proceso.

RESUMEN

Razonabilidad de iniciar este trabajo de grado: La proliferación de los dispositivos y su gran aceptación en el mercado hacen del área de telefonía móvil, un área aun poco explorada para el desarrollo de aplicaciones y/o soluciones informáticas

Objetivo del trabajo de grado: Realizar una investigación sobre los mejores protocolos de búsqueda y de seguridad existentes para constituir una red P2P para tecnología móvil y proponer el esquema de desarrollo a seguir, probado a través de un prototipo como parte integral del trabajo de grado

Proceso para llegar a los resultados: Para los resultados obtenidos, se hizo una investigación inicial de las redes P2P, para luego seleccionar las problemáticas de búsqueda y de seguridad como foco inicial, comparando las diferentes soluciones ya propuestas y seleccionando posibles soluciones adaptándolos al contexto móvil.

Resultados: Durante el análisis se encontró y seleccionó (por su eficiencia en cuanto a la complejidad búsqueda y escalabilidad) CHORD como protocolo de búsqueda, planteando dos escenarios de arquitectura aplicables para dicho protocolo en el contexto móvil, para luego seleccionar la mejor alternativa de solución teniendo en cuenta las restricciones del mundo móvil. Después de monitorear el comportamiento de la memoria y los tiempos de ejecución de los dos prototipos planteados se propone, como la mejor solución a corto plazo y según las limitaciones actuales de los dispositivos móviles, CHORD utilizando jerarquía de peers.

Siguientes pasos: Mejorar tiempos de respuesta buscando eficiencias en procesos del protocolo tales como el proceso de estabilización para poder adaptar la solución P2P pura. Análisis del tema legal en el contexto colombiano para este tipo de aplicaciones. Construir un estándar para ser usado por otros aplicativos móviles.

INTRODUCCION

El modelo de computación más conocido y usado es el de cliente-servidor, donde el servidor proporciona uno más servicios a clientes según las peticiones realizadas. Sin embargo en los últimos años un nuevo esquema ha emergido para superar algunos de los inconvenientes de tener una arquitectura centralizada como lo es la de cliente/servidor. A este nuevo esquema se le ha denominado *Peer-to-Peer*, cuya abreviatura es P2P, y que consiste en una red de iguales, donde no se hace distinción entre servidor y cliente, cualquiera que pertenezca a una red P2P puede desempeñar los dos roles. Esta característica hace de las redes P2P más confiables pues no hay un punto central de falla y los servicios ofrecidos por esta no se ven afectados si alguno de los integrantes de la red llegara a fallar, mientras que para la arquitectura cliente-servidor si el servidor falla los servicios que este ofrece no pueden ser proporcionados. La escalabilidad es otra fortaleza de la redes P2P, debido a su naturaleza pueden crecer sin que esto afecte el nivel de servicio, por el contrario mientras mas crecimiento exista, más amplia es la cobertura de los servicios ofrecidos[1]. Caso contrario ocurre con las redes centralizadas. Entre las más famosas aplicaciones P2P se tienen Kazaa, MSN Mesenger, Gnutella, entre otras.

De otro lado la proliferación de dispositivos móviles cada vez más poderosos, si se compara con su tamaño y con la capacidad de procesamiento de los primeros PC´s, además de las facilidades de portabilidad que ofrece y de la gran acogida, hacen de esta línea tecnológica una de las más interesantes en el campo de la investigación y de la industria. Pero aun más interesante se convierte si se une la portabilidad de los móviles con escalabilidad las redes P2P.

Esta tesis explora el mundo P2P móvil desarrollando una aplicación P2P móvil para compartir archivos, utilizando CHORD, un protocolo de búsqueda distribuida seleccionado, luego de comparar algunos de los más conocidos y usados, pero aplicando dos métodos distintos : el primero es una red P2P pura, totalmente móvil, es decir sus participantes son dispositivos móviles y no existe jerarquía de peers, mientras que el segundo método consiste de super-peers o r*endezvous* y *edge peers*[2],

[1] Cuando se habla de servicios se incluyen no solo aplicaciones sino también procesamiento y almacenamiento.
[2] Usando la terminología JXTA

que para este caso los edge peers o peers livianos, cuya poca capacidad de procesamiento los limita para realizar algunas tareas, son los móviles. Mientras que los súper peers son dispositivos de más capacidad y que realizan las tareas más arduas dentro de la comunidad P2P a la que pertenecen. Dichas aplicaciones mostrarán resultados para decidir cuál de las dos implementaciones es la más apropiada para un ambiente móvil P2P.

Los prototipos fueron implementados usando JXME y JXTA para Java. JXME es la herramienta de desarrollo móvil de *Sun Mycrosistems*, mientras que JXTA es un conjunto de protocolos que facilitan la comunicación P2P.

DESCRIPCIÓN DEL PROBLEMA

Uno de los principales retos que una aplicación peer-to-peer debe afrontar es el manejo eficiente para la localización de información distribuída a través de los participantes de una comunidad o grupo P2P. Para ello múltiples soluciones han sido presentadas CAN, CHORD, FREENET, TAPESTRY, Protocolos de Inundación, entre otras. Todos ellos son sistemas que ofrecen una solución a dicho problema, enfocados desde diferentes puntos de vista, por ejemplo la solución de CAN a este problema es usar un espacio virtual cartesiano de n dimensiones, CHORD por su parte resuelve el problema, basándose en un espacio en forma de anillo.

Durante el desarrollo de esta tesis se estudiaron tres soluciones al problema de nombres en una red P2P, ellos son: CAN , CHORD y JXTA. Dichas soluciones se compararon según parámetros de complejidad y escalabilidad para determinar cual sería la escogida para implementar una solución P2P móvil.

Pero además del problema de localización de información distribuida entre una comunidad P2P se debe agregar otra restricción, pero por parte de los dispositivos móviles, capacidad de procesamiento, almacenamiento, memoria, pantallas pequeñas, carga de batería.

Entonces, ¿ Cómo es el comportamiento de CHORD a nivel de procesamiento en un sistema P2P móvil ? Para ello se propone analizar dos prototipos el primero conformado por peer móviles y el segundo usando la filosofía de jerarquía de peers JXTA.

OBJETIVOS

Objetivo Principal Realizar una investigación sobre los mejores protocolos de búsqueda y de seguridad existentes para constituir una red P2P para tecnología móvil y proponer el esquema de desarrollo a seguir, probado a través de un prototipo como parte integral del trabajo de grado.

Objetivos Específicos

- Estudiar los protocolos de búsqueda distribuida con el fin de seleccionar el más adecuado para una aplicación P2P.
- Con base en los dos puntos anteriores, construir un prototipo a través de simulación de móviles sobre el protocolo de búsqueda escogido, utilizando dos arquitecturas P2P diferentes
- Aplicar conceptos de criptografía a las dos soluciones planteadas y analizar el comportamiento en cada una.
- Proponer la mejora alternativa encontrada para tecnología inalámbrica luego de evaluar cuál de las arquitecturas tiene un mejor desempeño.

1 MARCO TEORICO

1.1 CONCEPTOS DE REDES P2P

En general, una red informática entre iguales (en inglés peer-to-peer y más conocida como P2P) se refiere a una red que no tiene clientes y servidores fijos, sino una serie de nodos que se comportan como clientes y servidores de los demás nodos de la red. Este modelo de red contrasta con el modelo cliente-servidor. Cualquier nodo puede iniciar o completar una transacción. Los nodos pueden diferir en configuración local, velocidad de proceso, ancho de banda, conexión a la red y capacidad de almacenamiento [21].

Debido a que la mayoría de los computadores domésticos no tienen una IP fija, sino que le es asignada por el proveedor (ISP) en el momento de conectarse a Internet, no pueden conectarse entre sí porque no saben las direcciones que han de usar de antemano. La solución habitual es realizar una conexión a un servidor (o servidores) con dirección conocida (normalmente IP fija), que se encarga de mantener la relación de direcciones IP de los clientes de la red, de los demás servidores y habitualmente información adicional, como un índice de la información de que disponen los clientes. Tras esto, los clientes ya tienen información sobre el resto de la red, y pueden intercambiar información entre sí, sin intervención de los servidores [21].

Las tecnologías del P2P se han asociado a Napster, el servicio de música compartida que se ha descargado casi cuarenta millones de veces desde su lanzamiento a finales de 1999. Napster permite que los usuarios busquen y descarguen música en formato MP3 directamente de los discos duros de otros usuarios sobre Internet, sin la restricción en compartir material con derechos de autor [22].

1.1.1 Clasificación de Arquitecturas P2P

Básicamente las arquitecturas p2p para compartir archivos centran su clasificación en dos aspectos: centralización y estructura de la red.

Grado De Centralización Por este aspecto las arquitecturas P2P se clasifican en:

- Totalmente Descentralizada o P2P Puro

Todos los nodos en la red realizan exactamente las mismas tareas, actuando como servidores y clientes a la vez, no hay coordinación central de sus actividades [23].

Los nodos están directamente interconectados y envían mensajes a sus vecinos. En algunas redes P2P, cada peer que recibe una petición devuelve el resultado y en paralelo remite la petición a uno, algunos o a todos (dependiendo del protocolo) sus vecinos. Puesto que no hay unidad de organización central, los peers tienen que mantener la topología y la conectividad por sí mismos [24].

La ventaja de los sistemas P2P puros, es el hecho de que no hay punto de falla (es decir servidor central). Por lo tanto, la supervivencia de la red P2P aumenta [24].

Una desventaja de una red completamente distribuida P2P sin una autoridad central es que es difícil poner al día el sistema (e.g., siempre que una nueva versión de un componente esté disponible). El problema más importante de los sistemas puros P2P es el proceso de iniciación. Cuando el software P2P se inicia en una máquina, por lo menos un peer de la topología total debe ser conocido para permitir la conexión a la red. En la mayoría de los casos, esto es hecho generalmente proporcionando un servidor fijo, bien conocido, que proporcione una lista de algunos peers P2P. Otra posibilidad es emplear conexiones a direcciones IP al azar en ciertos rangos de dirección donde hay una buena posibilidad de conseguir una conexión. Sin embargo, se mira este acercamiento como poco amigable y por lo tanto, se utiliza solamente como ultimo recurso [24].

Ejemplos típicos de aplicaciones con este tipo de topología P2P son Gnutella, Freenet ,PAST y Chord [24].

- Parcialmente Centralizada

La base es igual a la de los sistemas totalmente descentralizados. Sin embargo, algunos de los nodos asumen un papel más "importante" que el resto de los nodos, actuando como índices centrales locales para los archivos compartidos por los peers locales. Estos nodos se llaman "Supernodos", y la manera por la cual se seleccionan para estas tareas especiales varía de sistema a sistema. Es importante observar que estos Supernodos no constituyen por si solos, puntos de falla para una red P2P,

puesto que se asignan dinámicamente y en caso de falla o ataque malévolo la red tomará la acción para sustituirlos por otros [23].

Hay varias ventajas al usar un servidor central. El protocolo se puede cambiar y mejorar fácilmente sin tener el problema de convencer a todos los clientes que soporten este protocolo (aunque esto no se mira siempre como ventaja porque un protocolo se debe diseñar siempre de tal manera que sea bastante flexible). Además, el uso de un servidor central facilita el diseño de la funcionalidad de seguridad. Los mecanismos de la autenticación y de autorización pueden ser empleados porque todo el tráfico significativo (e.g. peticiones de la búsqueda, información de la conexión, etc.) puede ser enviado a un solo servidor central [24].

- Híbrida

Hay un servidor central que facilita la interacción entre los peers manteniendo los directorios de archivos compartidos almacenados en los respectivos PC's de usuarios registrados en la red, en forma de meta- datos. La interacción end-to-end es entre dos clientes peer, no obstante estos servidores centrales facilitan esta interacción realizando las operaciones de búsqueda e identificando los nodos de la red (es decir las computadoras) donde se localizan los archivos [23].

Napster es un ejemplo clásico de este tipo de sistemas P2P [23].

Estructuras De Red Los sistemas P2P constituyen redes altamente dinámicas de peers con topología compleja. Esta topología crea una red de recubrimiento, que puede no tener relación con la red física que conecta los diferentes nodos. Los sistemas P2P se pueden distinguir por el grado en el cual estas redes de recubrimiento contienen alguna estructura. Por estructura se refiere a la manera en la cual el contenido de la red está situado con respecto a la topología de la red [23]. Básicamente se pueden distinguir 3 tipos de redes P2P basadas en el anterior concepto:

- Redes no estructuradas

En estas redes, el lugar de los datos (archivos), está completamente desligado de la topología de red. Debido a que no hay información sobre cuáles nodos probablemente tienen archivos relevantes, se buscan esencialmente a través de búsqueda aleatoria,

en la cual varios nodos son probados e interrogados sobre la posibilidad que tengan archivos que concuerden con la consulta.

Los sistemas P2P no estructurados, difieren en la forma por la cual construyen la topología de recubrimiento, y la forma en la cual distribuyen consultas de nodo a nodo. La ventaja de tales sistemas es que pueden acomodar una población de nodos altamente transitorios, La desventaja es que es duro encontrar los archivos deseados sin búsquedas distribuidas complejas. Por esta razón, los sistemas P2P no estructurados son considerados no escalables.

A este grupo de sistemas pertenece Gnutella [23].

- Redes estructuradas Han emergido principalmente para direccionar soluciones de escalabilidad que los sistemas no estructurados están afrontando. Los métodos de búsquedas aleatorias adoptadas por sistemas no estructurados parecen ser inherentemente no escalables, y los sistemas estructurados fueron propuestos, para que la topología de red de recubrimiento sea herméticamente controlada y los archivos (o punteros hacia ellos) sean colocados en locaciones especificadas con precisión. Estos sistemas proveen relación entre el identificador del archivo y la localización, en forma de tabla distribuida de enrutamiento, de tal modo que las búsquedas pueden ser eficientemente enrutadas al nodo con el archivo deseado [23].

La desventaja de sistemas estructurados es que es duro mantener la estructura de requerida de enrutamiento para cada población de nodos transitorios, en el cual los nodos entran y salen a grandes tasas [23].

Dentro de la redes P2P existentes y que presentan esta característica están Chord, CAN, PAST, Tapestry, entre otras. [23]

- Redes poco estructuradas Entran entre las dos presentadas anteriormente. Las localizaciones de los archivos son afectadas por acierto en el enrutamiento, pero no están completamente especificadas, así que no todas las búsquedas son exitosas. Ejemplo de este tipo de redes es Freenet

La tabla 1, resume las características en cuanto a la clasificación presentada, para algunos de los sistemas P2P existentes:

	Redes No Estrucuturadas	**Redes Poco Estructuradas**	**Redes Estructuradas**
Híbrido	Napster		
Totalmente Descentralizado	Gnutella	Freenet	Can, Chord, Tapestry
Parcialmente Centralizado	Kazaa , Gnutella		

Tabla 1. Sistemas P2P.

1.2 PROTOCOLOS DE BÚSQUEDA DISTRIBUIDA

En esta sección se describen tres protocolos de búsqueda en redes P2P: CHORD, CAN y JXTA.

1.2.1 Chord

CHORD se caracteriza por: "dada una llave, este la proyecta a un nodo. La localización de los datos puede ser implementada asociando cada llave a un ítem de datos." [3]

El protocolo Chord [3] utiliza SHA-1, una función de hash consistente para asignar un identificador de m bits a cada nodo y a cada llave. Las funciones de hash consistentes son funciones hash con algunas propiedades adicionales ventajosas, por ejemplo, permiten a los nodos unirse y dejar el sistema con la mínima interrupción. En la siguiente sección se explica en qué consiste el hashing consistente.

Hashing Consistente [3] [9] Cada nodo Chord tiene un único identificador (ID) de m bits, obtenido haciendo hashing sobre la dirección IP del nodo y un índice virtual del nodo. Chord visualiza los identificadores en un espacio circular denominado Anillo Chord, cuyo tamaño es 2^m. Las llaves también son proyectadas en este espacio de identificadores, haciendo hashing sobre ellas a identificadores de llaves de m bits. m

es un entero que puede ser elegido con un valor lo suficientemente grande para que la probabilidad de que dos nodos o dos llaves reciban el mismo identificador sea baja. Los identificadores del anillo Chord están numerados desde 0 a 2m – 1.

La función hash calcula la llave identificadora haciendo un hashing a la llave, y el identificador del nodo a través de un hashing de la dirección IP del nodo. Los identificadores del nodo y de la llave son dispuestos en un círculo de tamaño llamado el Anillo Chord (Chord Ring).

Una llave k es asignada a un nodo cuyo identificador (ID) es igual o mayor que el identificador de la llave. Dicho nodo es llamado nodo sucesor de k, denotado por sucesor(k), y es el primer nodo en el sentido de las manecillas del reloj que dista de k en el círculo.

El Hashing consistente permite a los nodos entrar y salir de la red con movimientos mínimos de llaves. Para mantener mapeos correctos a sucesores cuando un nodo n se une a la red, ciertas llaves previamente asignadas al sucesor de n serán asignadas a n. Cuando un nodo sale de la red, todas las llaves asignadas a n son reasignadas a su sucesor. Ningún otro cambio en la asignación de llaves a nodos ocurre.

La figura 1 muestra un espacio identificador con 8 ID's disponibles (2^3). Existen 3 nodos, los cuales fueron proyectados (usando sus respectivas direcciones IP) a los identificadores 0,1 y 3. Los nodos responsables por los identificadores de llaves (1, 2 y 6) se obtienen encontrando el sucesor para los respectivos ID's de las llaves. Así, el nodo responsable por la llave cuyo ID es 6, es el sucesor(6), el cual es el nodo cuyo ID es cero , ya que es el primer nodo que se encuentra después del identificador en 6 en el sentido de las manecillas del reloj. Los responsables de las otras dos llaves se calculan haciendo el mismo procedimiento.

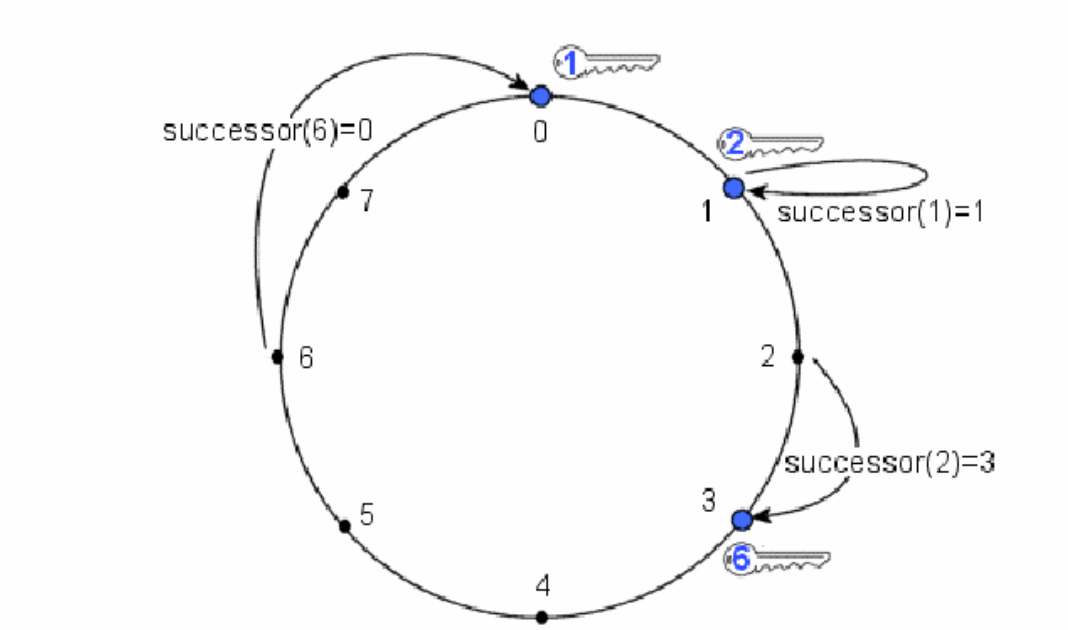


Figure 1. Circulo identificador de Chord con tres nodos (m=3). [3]

Búsqueda Un nodo Chord utiliza dos estructuras de datos para realizar operaciones de búsqueda: una lista de sucesores y una tabla "finger". Únicamente la lista de sucesores se requiere para la corrección, así que Chord tiene cuidado de mantener su exactitud. La tabla finger acelera operaciones de búsqueda, pero no necesita ser exacta, así que Chord es menos agresivo para mantenerla. La siguiente discusión primero describe cómo realizar (sino retardarse) operaciones de búsqueda correctas con la lista de sucesores, y en seguida describe cómo acelerarlas con la tabla "finger". [9]

Cada nodo Chord mantiene una lista de las identidades y las direcciones del IP de sus r sucesores inmediatos en el anillo Chord. El hecho de que cada nodo conoce su propio sucesor significa que un nodo puede procesar siempre operaciones de búsqueda correctamente: si la llave deseada está entre el nodo y su sucesor, el último nodo es el sucesor de la llave; si no las operaciones de búsqueda se pueden remitir al sucesor, que mueve las operaciones de búsqueda estrictamente más cerca a su destino. Un nuevo nodo n conoce a sus sucesores en su primera unión al anillo Chord, solicitando a un nodo existente realizar operaciones de búsqueda para el

sucesor de n; n entonces pide a ese sucesor su lista de sucesores. Las r entradas en la lista proporcionan tolerancia a fallas: si el sucesor de un nodo inmediato no responde, el nodo puede sustituir la segunda entrada en su lista de sucesores. Todos los *r* sucesores tendrían fallas simultáneamente interrumpiendo el anillo Chord, un acontecimiento que se puede hacer muy improbable con valores modestos de *r*. [9]

La principal complejidad implicada en las listas de sucesores consiste en la notificación de un nodo existente cuando un nuevo nodo debe ser su sucesor. El procedimiento de estabilización descrito en [2] hace esto de una manera que garantice la conectividad preservada de los apuntadores del sucesor del anillo de Chord. Las operaciones de búsqueda realizadas únicamente con listas de sucesores requerirían un promedio de intercambios de N/2 mensaje, donde N es el número de peers. Para reducir el número de mensajes a O(log N) [9], cada nodo debe mantener información adicional para agilizar el proceso, información almacenada en una tabla denominada finger table, la cual se define en la tabla 2.

Notación	**Definición**
Finger[k].start	$(n+2^{k-1}) \bmod 2^m \; 1 \le k \le m$
.interval	*[Finger[k].start, Finger[k+1].start]*
.node	Primer nodo >= n.*finger[k].start*
Sucesor	Próximo nodo en el circulo identificador, generalmente Finger[1].node
Predecesor	Nodo previo en el círculo identificador.

Tabla 2. Definición Finger Table en CHORD [1]

Para especificarla [1] la describe así: sea m el número de bits en los identificadores de llaves/nodos. Cada nodo n mantiene una tabla de enrutamiento, con al menos m entradas, llamadas finger table. La i^{th} entrada en las tablas del nodo n contiene la identidad del primer nodo s que sigue a n en al menos 2^{i-1} en el círculo identificador. Esto es, s= sucesor$(n+2^{i-1})$ donde $1 \le i \le m$, llamando al nodo s el i^{th} finger del nodo n, y denotado por n.finger[i].node.

Una entrada de la *finger table* incluye el identificador Chord y la dirección IP del nodo (y número de puerto) del nodo relevante. Por la definición anteriormente citada, se puede notar que el primer *finger* de n es su inmediato sucesor.

La figura 2, muestra un ejemplo de cómo serían las *finger tables* de nodos.

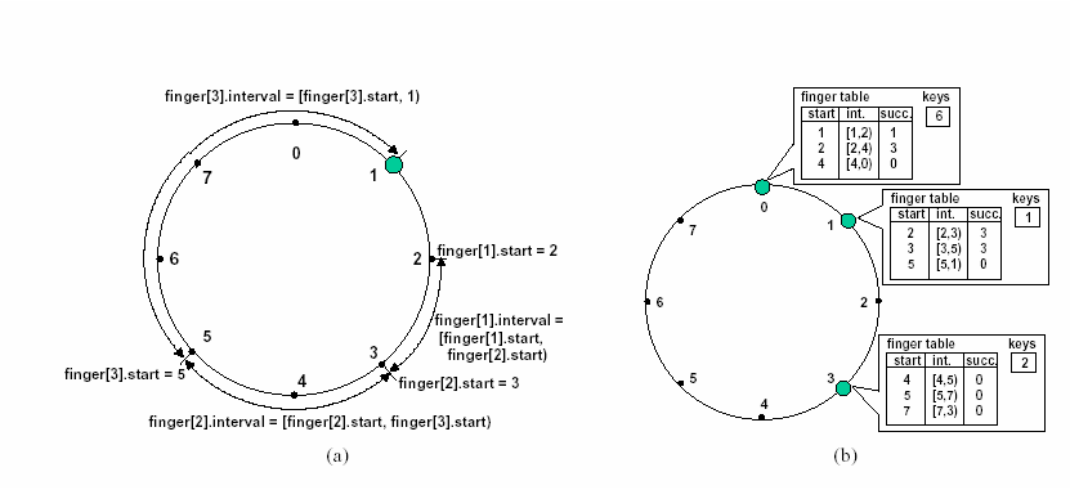


Figura 2. Ejemplos de Finger Tables en CHORD [1]

Según las características de diseño de Chord, ya presentadas y como lo afirma [1], cada nodo *n* almacena sólo información de algunos nodos del anillo, debido a esto no es posible que un nodo *n* tenga suficiente información en su *finger table* para encontrar el sucesor de la llave *K*. Para ello CHORD ofrece encontrar el sucesor de una llave así:

"Una búsqueda para el sucesor de *n* iniciada en el nodo *r* comienza determinando si n está entre *r* y el inmediato sucesor de *r*. Si es así, la búsqueda termina y el sucesor de r es retornado. De lo contrario, se reenvía la búsqueda al nodo más lejano en su tabla que precede a *n*, si llamamos a este nodo *s*. El mismo procedimiento es repetido por *n* hasta que la búsqueda finalice. "[2]

La figura 3 muestra el pseudo código para la búsqueda del sucesor del nodo con identificación *id*. El ciclo principal está en *find_predecessor*, que envía *preceding_node_list* RPCs a una sucesión de otros nodos; cada RPC busca las tablas del otro nodo para los nodos más cercano a *id*. Cada iteración fijará el n' a un nodo entre el actual n' e *id*. Puesto que *preceding_node_list*

nunca retorna una identificación (ID) mayor que la *id*, este proceso nunca llegará más allá del sucesor correcto. Puede ser más corto, especialmente si un nuevo nodo se ha recientemente unido con una ID justo antes de *id*; en ese caso la comprobación para *id* $\notin$ (n', n'.successor] asegura de que persista *find_predecessor* hasta que encuentre un par de nodos que contengan *id.*

```
// Ask node n to find id's successor; first
// finds id's predecessor, then asks that
// predecessor for its own successor.
n.find_successor(id)
  n' = find_predecessor(id);
  return n'.successor();

// Ask node n to find id's predecessor.
n.find_predecessor(id)
  n' = n;
  while (id ∉ (n', n'.successor()])
    l = n'.preceding_node_list(id);
    n' = max n'' ∈ l s.t. n'' is alive
  return n';

// Ask node n for a list of nodes in its finger table or
// successor list that precede id.
n.preceding_node_list(id)
  return {n' ∈ {fingers ∪ successors}
    s.t. n' ∈ (n, id]}
```

Figura 3. Algoritmo de Búsqueda CHORD [2]

Dos aspectos del algoritmo de búsqueda lo hacen robusto. Primero, un RPC a *preceding_node_list* en el nodo *n* devuelve una lista de los nodos que *n* cree están entre él y la *id* deseada. Cualquiera de ellos se puede utilizar para progresar hacia el sucesor de *id*; deben todos ser insensibles para que las operaciones de búsqueda fallen. En segundo lugar, el ciclo *while* asegura de que el *find_predecessor* se intente mientras puede encontrar cualquier nodo más cercano a *id*. Mientras los nodos tienen cuidado de mantener punteros correctos del sucesor, *find_predecessor* tendrá éxito eventualmente.

En el caso generalmente en el cual la mayoría de los nodos tienen información correcta de la tabla "finger", cada iteración del ciclo *while* elimina mitad de la distancia restante al objetivo. Esto significa que los saltos iniciales recorren largas distancias en el espacio de identificadores y los últimos saltos recorren distancias más pequeñas. Es de importante observar que este algoritmo no proporciona en sí mismo un límite de O(logN); la estructura de la tabla "finger", que el algoritmo examina, garantiza que cada salto cubrirá la mitad de la distancia restante. Este comportamiento es la fuente de las características logarítmicas del algoritmo.

Inserción y Salida de Nodos [1] En una red dinámica, los nodos pueden unirse y salirse de ella en cualquier tiempo. Cuando un nuevo nodo r se une a la red, este debe:

1. Inicializar su finger table.
2. Los nodos existentes deben actualizar sus tablas para reflejar la existencia de r.
3. Notificar a las capas más altas de software para que esta pueda transferir a los valores asociados con las llaves por las que el nodo nuevo es ahora responsable.

Para el evento en que un nodo sale de la red las operaciones a realizar son similares a las presentadas cuando un nodo ingresa a la red. La tabla 3 resume las operaciones básicas que CHORD ofrece:

Función	**Descripción**
Insert (Key , Value)	Inserta la llave Key con su respectivo valor buscando **(binding)** en N distintos nodos.
Lookup (key)	Retorna el valor asociado a Key
Update (key,newval)	Inserta la pareja Key/Newval
Join (n)	Causa que un nodo se adicione como un servidor al sistema Chord cuyo nodo n hace parte de este.
Leave ()	Salida del sistema Chord

Tabla 3. Operaciones de Chord. [1]

COMPLEJIDAD BASADA EN TEOREMAS [1] El protocolo CHORD se caracteriza por su simplicidad y eficiencia demostrada [1], basado en los siguientes teoremas:

- TEOREMA1 Para cualquier conjunto de N nodos y K llaves, con alta probabilidad:
  1. Cada nodo es responsable de al menos (1+e)K/N llaves
  2. Cuando un $(N+1)^{st}$ nodo se une o sale de la red, la responsabilidad por O(K/N) llaves "cambia de manos".

- TEOREMA2 Con alta probabilidad, el número de nodos que deben ser contactados para encontrar un sucesor en una red de N nodos es O(log N)

- TEOREMA3 Con alta probabilidad, cualquier nodo uniendo o saliéndose de una red de N nodos usara $O(\log^2 N)$ mensajes para restablecer las invariantes de enrutamiento y las finger tables

1.2.2 CAN (Content Addressable Network) [3],[4]

CAN es esencialmente una tabla hash distribuida, a la escala de Internet-que proyecta nombres de archivos a su localización en la red.

Las operaciones básicas realizadas por CAN son la inserción, operaciones de búsqueda y la cancelación de la pareja (llave, valor) en la tabla hash distribuida. Cada nodo de CAN almacena una parte (llamada una "zona") de la tabla de hash, así como la información sobre un número pequeño de zonas "adyacentes" a la tabla. Las peticiones de inserción, las operaciones de búsqueda o la cancelación para una llave particular se enrutan vía nodos intermedios al nodo que mantiene la zona que contiene la llave. Las características principales de la CAN son (las cuales serán analizadas en las próximas secciones):

- Totalmente descentralizado: No requiere ninguna forma de control, organización o configuración centralizada.

- Escalable: los nodos mantienen solamente una pequeña parte del estado del sistema, independiente del número de nodos en el sistema.
- Tolerante a fallas: los nodos pueden enrutar aún en presencia de fallas.

Diseño Concebido como un mecanismo de indexación, no sólo para sistemas peer-to-peer sino para muchos otros sistemas con funcionalidad distinta tal como se expresa en [4], basa sus fundamentos de diseño en un plano Cartesiano virtual o lógico de d-dimensiones, totalmente independiente de la topología física. Dicho espacio coordinado entero se reparte dinámicamente entre todos los nodos en el sistema tal que cada nodo "posee" su zona individual, distinta dentro del espacio total. La figura 4 muestra un espacio cartesiano de dos dimensiones con 5 nodos.

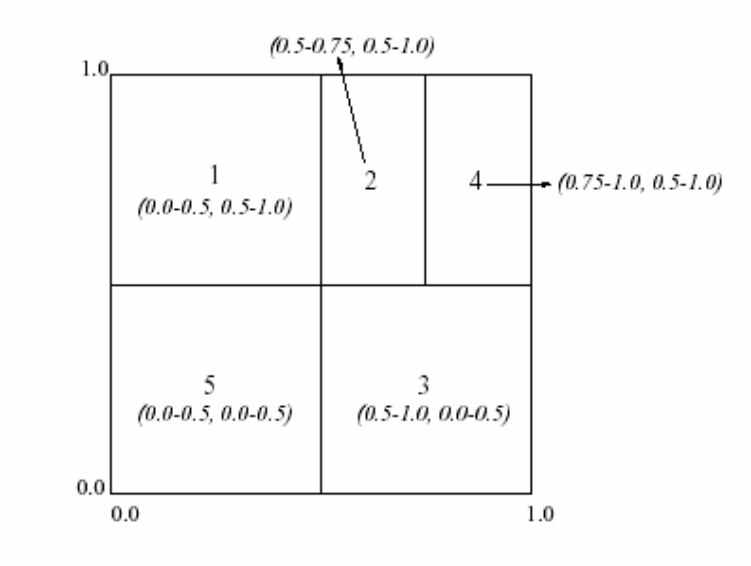


Figura 4. Espacio Cartesiano de 2-dimensiones con 5 nodos [4]

Este espacio coordinado virtual se utiliza para almacenar pares (llave, valor) como sigue: para almacenar un par (K1,V1), K1 es determinísticamente proyectado sobre un punto P en el espacio coordinado usando una función uniforme de hash(en realidad se aplican dos funciones de hash sobre K1 para obtener dos coordenadas (x y y)) . El par correspondiente (llave, valor) entonces se almacena en el nodo que es dueño de la zona dentro de la cual el punto P se encuentra (este proceso se resume en la figura

5). Para recuperar la entrada correspondiente a la llave K1, cualquier nodo puede aplicar la misma función de hash para proyectar K1 sobre el punto P y luego recuperar el valor correspondiente del punto P. Si el punto P no es poseído por el nodo que realiza la petición o sus vecinos inmediatos, la petición se debe enrutar a través de la infraestructura de CAN hasta alcanzar el nodo en cuya zona P se encuentra.

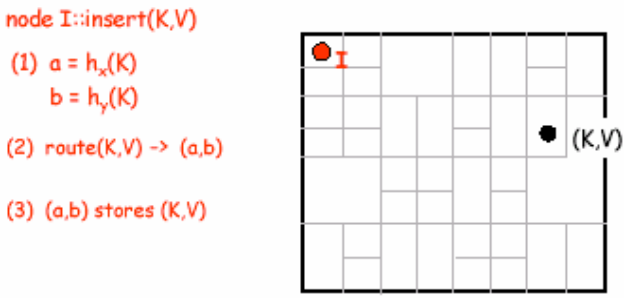


Figura 5. Ejemplo de Inserción de un par (llave, valor) en CAN [11]

Enrutamiento [4] Debido al diseño de plano cartesiano de CAN, se observa que el enrutamiento trabaja siguiendo la trayectoria de una línea recta a través del espacio cartesiano desde el origen a las coordenadas destino. Un nodo CAN mantiene una tabla de enrutamiento coordinada que guarda la dirección IP y la zona coordinada virtual de cada uno de sus vecinos en el espacio coordinado. Tal información es suficiente para enrutar entre dos puntos en el espacio: El mensaje CAN incluye los coordenadas destino. Usando su conjunto de coordenadas vecinas, un nodo enruta un mensaje hacia su destino por un simple reenvío a su vecino con las coordenadas más cerradas o cercanas a las coordenadas destino. La figura 6 muestra un ejemplo de enrutamiento

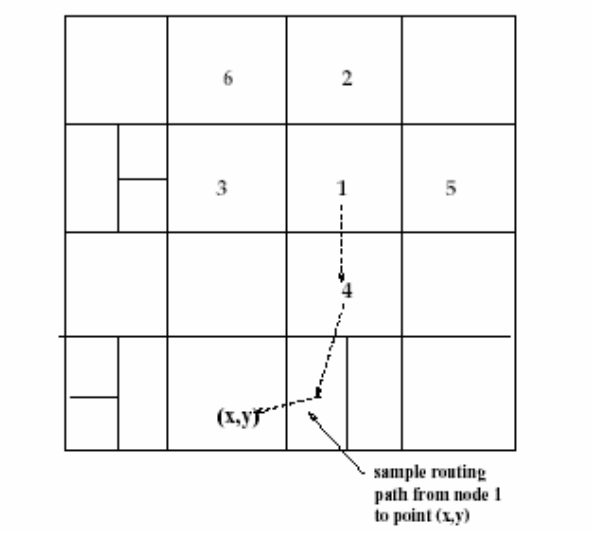


Figura 6. Enrutamiento de CAN [4].

Para un espacio *d*-dimensional particionado en *n* zonas iguales mantiene *2d* dimensiones (dos vecinos por dimensión) y el promedio de longitud de trayectoria de enrutamiento es $(d/4)(n^{1/d})$ (cada dimensión tiene $(n^{1/d})$ nodos, un destino, estará en promedio $(1/4)(n^{1/d})$ de cada una de las *d-dimensiones)*.

Nota: En un espacio coordinado d-dimensional, dos nodos son vecinos si sus planos coordinados se traslapan a lo largo de del d-1 dimensiones y lindan a lo largo de una dimensión.

Inserción de Nodos [4] Como se dijo anteriormente, el espacio completo de CAN, es dividido entre los nodos que se encuentran actualmente en el sistema. Para obtener tal particionamiento, cada vez que un nuevo nodo se une a CAN, una zona existente es dividida en dos mitades, una de las cuales es asignada al nuevo nodo. La división es hecha siguiendo un ordenamiento bien definido de las dimensiones, decidiendo a lo largo de cuál dimensión una zona será dividida, así esas zonas pueden ser re-combinadas cuando salen nodos. Por ejemplo para un espacio 2-d, una zona debería primero dividirse a lo largo de la dimensión X, luego la Y, Luego X nuevamente seguida de Y, y así sucesivamente. La figura 7 representa la evolución de un espacio CAN de 2-dimensiones con 5 nodos uniéndose. El primer nodo que se une, se hace

dueño del espacio completo. Cuando el segundo nodo se une, el espacio se divide en dos y cada nodo obtiene una mitad. El tercer nodo llega, selecciona una zona y la divide por la mitad, este proceso se repite cada vez que un nuevo nodo llega.

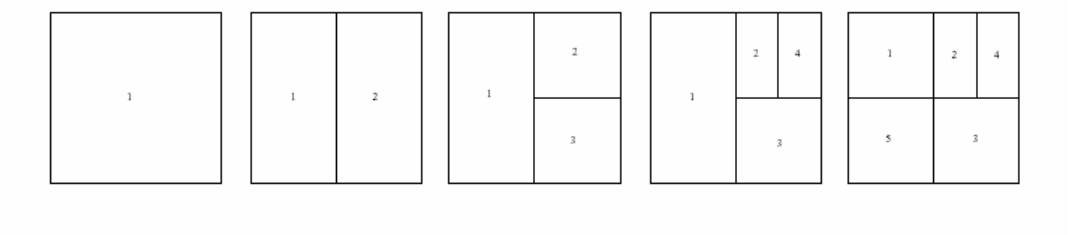


Figura 7. Particionamiento de CAN cuando 5 nodos se unen sucesivamente. [4]

Los nuevos nodos que se unen al sistema CAN son asignados a su propia porción del espacio coordinado partiendo la zona asignada de un nodo existente por la mitad, como sigue:

- El nuevo nodo identifica un nodo que existe ya dentro CAN, usando cierto mecanismo iniciación.
- Usando el mecanismo de enrutamiento de CAN, elige aleatoriamente un punto P en el espacio y envía una petición de JOIN al nodo cuya zona contiene a P. Este mensaje se envía vía cualquier nodo CAN existente. Cada nodo CAN entonces, utiliza el mecanismo de enrutamiento CAN para remitir el mensaje, hasta que alcanza el nodo cuya zona contiene P. La zona será partida, y la mitad será asignada al nuevo nodo.
- El nuevo nodo aprende las direcciones del IP de sus vecinos, y notifica a los vecinos de la zona partida de modo que la enrutamiento pueda incluir el nuevo nodo.

INICIACION

Un nuevo nodo CAN primero descubre la dirección IP de cualquier nodo en el sistema. El funcionamiento de CAN no depende de los detalles de cómo se hace esto, pero CAN utiliza el mismo mecanismo de iniciación que Yallcast y YOID. Los autores de CAN asumen que CAN tiene un dominio de nombres DNS asociado, y que este resuelve la dirección IP de uno o más nodos bootstrap CAN. Un nodo bootstrap

mantiene una lista parcial de nodos CAN que cree se encuentran actualmente en el sistema.

Para unirse a CAN, un nuevo nodo busca el nombre del dominio en DNS para obtener la dirección IP del nodo de inicio. El nodo bootstrap provee entonces las direcciones IP de algunos nodos que se encuentran actualmente en el sistema y que son escogidos aleatoriamente.

¿CÓMO ENCONTRAR ZONA?

El nuevo nodo elige aleatoriamente un punto P en el espacio y envía una petición del JOIN destinada al punto P. Este mensaje se envía a CAN vía cualquier nodo existente de CAN. Entonces, cada nodo de CAN utiliza el mecanismo de enrutamiento de CAN para remitir el mensaje, hasta alcanzar el nodo cuya zona contiene a P. Este nodo actual de la zona, parte su zona por la mitad y asigna una mitad al nuevo nodo. Teniendo en cuenta el mecanismo de particionamiento descrito anteriormente. Los pares (llave, valor) de la mitad de la zona que se entregará también se transfieren al nuevo nodo.

ENSAMBLANDO LA RUTA

Una vez el nodo ha obtenido su zona, el nuevo nodo aprende las direcciones IP de sus vecinos fijados desde el ocupante anterior. Este sistema es un subconjunto de los vecinos del nodo anterior, más el nodo en sí mismo. De igual forma, el nodo anterior actualiza su sistema de vecinos, eliminando aquellos nodos que ya no serán más sus vecinos. Finalmente, los vecinos del nuevos y viejo nodos deben ser informados de esta reasignación de espacio. Cada nodo en el sistema envía un mensaje de actualización, seguido por reajustes periódicos, de su zona actualmente asignada, a todos sus vecinos. Éste estilo de actualizaciones aseguran de que todos sus vecinos aprendan rápidamente sobre el cambio y pongan al día sus propios sistemas de vecinos. La figura 8 muestra un ejemplo de un nuevo nodo (nodo 7) que se une a un sistema CAN de 2-dimensiones. Como se puede ver, la adición de un nuevo nodo afecta solamente un número pequeño de nodos existentes en un lugar muy pequeño del espacio coordinado. El número de vecinos que un nodo mantiene depende solamente de la dimensionalidad del espacio coordinado y es independiente del

número total de nodos en el sistema. Así, la inserción del nodo afecta únicamente O(d) nodos existentes, lo cual es importante para CANs con un alto número de nodos.

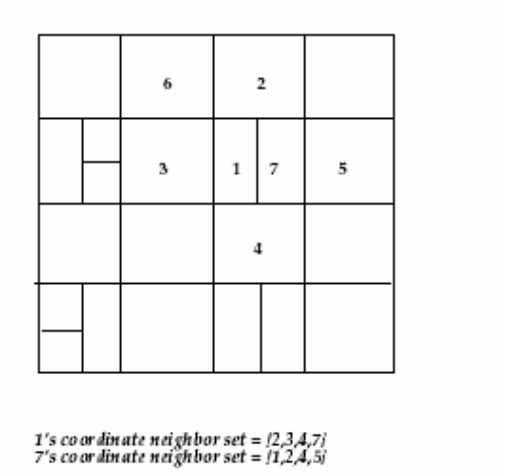


Figura 8. Nodos uniéndose a CAN [4]

Salida de Nodos [4] Cuando un nodo sale de CAN, necesita asegurarse que la zona que ocupaba sea asumida por los nodos restantes. El procedimiento normal para hacer esto es para que el nodo entregue explícitamente su zona y la base de datos asociada (llave, valor) a uno de sus vecinos. Si la zona de uno de los vecinos se puede combinar con la zona del nodo que sale para producir una sola zona válida, se hace esto. Si no, entonces la zona se da al vecino cuya zona actual es la más pequeña, y ese nodo entonces manejará temporalmente ambas zonas.

CAN también necesita ser robusto a fallos del nodo o de la red, donde uno o más nodos llegan a ser simplemente inalcanzables. Esto es manejado con un algoritmo TAKEOVER que asegura que uno de los vecinos del nodo fallido asuma el control de la zona. Sin embargo en este caso los pares (llave, valor) almacenados por el nodo que sale se darán por perdidos hasta que el estado sea restaurado por los sostenedores de datos. Bajo condiciones normales un nodo envía mensajes periódicos de actualización a cada uno de sus vecinos enviando las coordenadas de su zona y una lista de sus vecinos y de sus coordenadas. La ausencia prolongada de un mensaje de actualización de un vecino señala su falta.

Una vez que un nodo haya decidido que su vecino ha muerto este inicia el mecanismo TAKEOVER y comienza a correr el contador. Cada vecino del nodo fallido hará esto independientemente, con el contador de tiempo iniciado en proporción al volumen de nodos de la zona del nodo. Cuando expira el contador, un nodo envía un mensaje de la TAKEOVER que transporta el volumen de su zona para todos los vecinos del nodo fallido. Al recibir un mensaje TAKEOVER, un nodo cancela su propio contador si el volumen de la zona en el mensaje es más pequeño que su propio volumen, o contesta con su propio mensaje TAKEOVER. De esta forma, un nodo vecino es eficiente al escoger cuál está todavía vivo, y cuál tiene un volumen pequeño de zona.

Complejidad [4] Según lo descrito en las secciones anteriores, la complejidad de CAN esta definidos por: "Para un espacio d-dimensional particionado en n zonas iguales, en promedio la longitud de la trayectoria de enrutamiento es así (d/4)(n 1/d) y nodos individuales mantienen 2d vecinos. Estos resultados de escalamiento significan que para un espacio d-dimensional, se puede crecer el número de nodos (y por lo tanto de zonas) sin el aumento por nodo del estado (tabla de enrutamiento) mientras que la longitud de trayectoria crece como O(dn1/d)".

1.2.3 JXTA [5,6]

JXTA define un sistema de protocolos abiertos para las redes peer-to-peer. Estos protocolos basados en XML, describen operaciones complejas tales como descubrimiento de peers, enrutamiento, intercambio básico de mensajes de consulta/respuesta, y propagación de la red a través de peers rendezvous.

JXTA, se puede analizar generalmente en tres capas; la capa base, la capa de servicios y la capa de aplicación según lo mostrado en la figura 9. La capa base se ocupa del establecimiento del peer, el manejo de la comunicación, tal como enrutamiento y otras tareas de nivel bajo. La capa de servicios trata con conceptos de nivel más alto tales como indexación, búsqueda y archivos compartidos. En el tope está la capa de aplicaciones la cual incluye, entre otras, aplicaciones para correo y almacenamiento.

JXTA fue diseñado para proporcionar una plataforma general encima de la cual servicios y aplicaciones podrían ser construidos. Esta plataforma fue pensada para ser fina y pequeña, abasteciéndose de primitivas interesantes y poderosas para ser usadas por servicios y aplicaciones

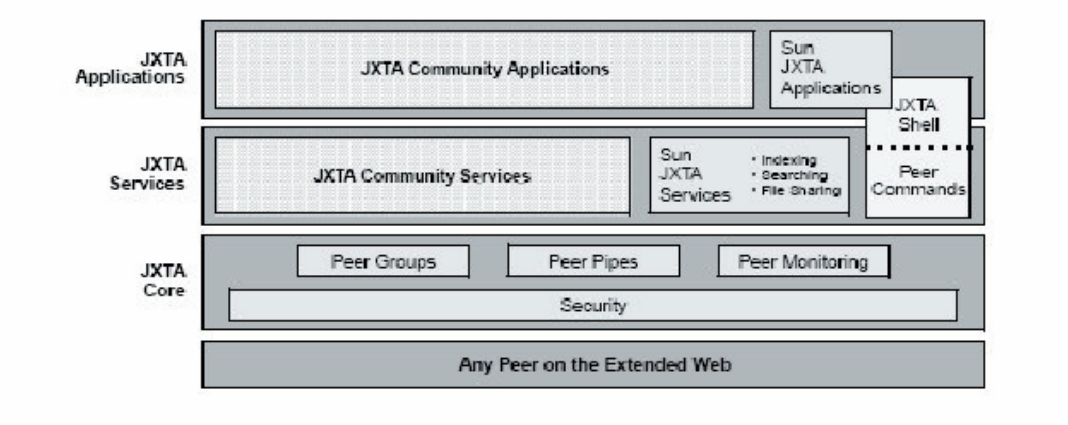


Figura 9. Capas JXTA [5]

Una red de JXTA se compone de varios componentes:

- Peer: es el elemento básico de cualquier red de JXTA. Un peer implementa los protocolos básicos JXTA, y funciona independiente y asincrónicamente del resto de peers. Un peer proporciona aplicaciones y/o servicios de red y es capaz de publicarse el mismo.
- Grupo Peer: es una colección de peers. JXTA define protocolos que los peers utilizan para crear, ensamblar, y monitorear grupos. Un grupo proporciona servicios tales como descubrimiento de peer, membresía, control de acceso, pipes y supervisión.
- Pipes: representa conexiones virtuales entre peers. Un pipe punto a punto conecta a dos peers; una pipe de propagación conecta un peer de difusión con múltiples oyentes. Peers conectados a Pipes no necesariamente están conectados físicamente en forma directa; pueden ser conectados a través de múltiples pipes intermedios.
- Mensajes: son los datos intercambiados a través de los pipes entre peers y puntos finales. JXTA define un formato para todos los mensajes. Cada peer puede definir el formato del contenido del mensaje mientras ese formato esté

acorde con la especificación XML. Sin embargo, para que dos peers intercambien información significativa, deben entender el formato del mensaje del otro.

- Advertisements o Anuncios: son estructuras de metadatos para describir recursos de la red JXTA. Todos los peers y servicios entienden los anuncios.

Organización de Red [6] La red JXTA es una red ad-hoc, multi-hop, integrada por peers conectados. Las conexiones en la red pueden ser transitorias, y el enrutamiento de mensajes entre los peers no es determinístico. Los peers pueden unirse o dejar la red en cualquier momento, y las rutas pueden cambiar con frecuencia.

Los peers pueden tomar cualquier forma así como también pueden comunicarse usando protocolos de JXTA. La organización de la red no es un mandato del framework JXTA, pero en la práctica cuatro clases de peers son utilizados:

1. Minimal Edge Peer: estos peers pueden enviar y recibir mensajes, pero no almacenan anuncios ni enrutan mensajes de otros peers. Peers en dispositivos con recursos limitados (un teléfono celular, o PDA's) podrían pertenecer a este tipo de peer.

2. Full-Featured Edge Peer: estos peers pueden enviar y recibir mensajes, y almacenar anuncios. Un simple peer responde a las peticiones de descubrimiento con la información encontrada en sus anuncios almacenados, pero no remite ninguna petición de descubrimiento.

3. Rendezvous Peer: Un peer rendezvous es como cualquier otro peer, y mantiene un depósito de anuncios. Sin embargo, los peers rendezvous también transmiten las peticiones de descubrimiento para ayudar a otros peers a descubrir recursos. Cuando un peer se une a un grupo de peers, este busca automáticamente un peer rendezvous. Si ningún peer rendezvous es encontrado, este dinámicamente se hace peer rendezvous para ese grupo de peers. Cada peer rendezvous mantiene una lista de otros peers rendezvous conocidos y también de los peers que lo estén utilizando como rendezvous.

Cada grupo peer mantiene su propio conjunto de peers rendezvous, y puede tener tantos peers como sean necesarios. Únicamente los peers rendezvous que son miembros de un grupo peers verán peticiones específicas de búsqueda del grupo peer.

Los edges peer envían peticiones de búsqueda y de descubrimiento a los peers rendezvous, quienes alternadamente remiten peticiones que no pueden contestar a otros peers rendezvous conocidos. El proceso del descubrimiento continúa hasta que un peer tiene la respuesta o la petición muere. Los mensajes tienen por defecto *tiempo-para-vivir* (TTL) de siete saltos. Los *loopbacks* son prevenidos manteniendo la lista de peers a lo largo de la trayectoria del mensaje.

4. Relay Peers: mantienen información sobre las rutas a otros peers y enrutan mensajes a peers. Un peer primero mira en su cache local para enrutar información. Si no se encuentra, el peer envía consultas a los peers relay solicitando información de la ruta. Los peers relay también retransmiten mensajes en favor de peers que no pueden dirigirse directamente a otro peer (Ambientes NAT), tendiendo un puente sobre diversas redes físicas y/o lógicas.

Cualquier peer puede implementar los servicios requeridos para ser un peer relay o un peer rendezvous. Los servicios relay y rendezvous se pueden poner en ejecución a la vez en el mismo peer.

- SRDI La plataforma JXTA 2.0 J2SE soporta un servicio Índice Distribuido de Recurso Compartido (SRDI) que proporciona un mecanismo más eficiente para propagar peticiones en la red JXTA. Los peers rendezvous mantienen un índice de los anuncios publicados por los peers edge. Cuando peer edges publican nuevos anuncios, ellos utilizan el servicio SRDI para crear índices a los anuncios en su rendezvous. Con esta jerarquía, las consultas se propagan únicamente entre rendezvous, lo que reduce significativamente el número de peers implicados en la búsqueda de un anuncio.

Cada rendezvous mantiene su propia lista de rendezvous conocidos dentro del grupo peer. Un rendezvous puede recuperar información de rendezvous desde un sistema

predefinido de rendezvous bootstrap, o semilla. Los rendezvous periódicamente seleccionan al azar un número dado de peers rendezvous y les envía una lista al azar de sus rendezvous conocidos. Los rendezvous también purgan periódicamente rendezvous que no responden. Así, mantienen una red consistente de peers rendezvous conocidos. Cuando un peer publica un nuevo anuncio, el anuncio es puesto en un índice por el servicio SRDI usando llaves tales como el nombre del anuncio o la identificación. Solamente los índices del anuncio son empujados al rendezvous por SRDI, reduciendo al mínimo la cantidad de datos que necesitan ser almacenados en el rendezvous. El rendezvous también empuja el índice a los peers rendezvous adicionales (seleccionados por el cálculo de una función hash del índice del anuncio).

- Consultas Una configuración de ejemplo se muestra en la figura 10. El Peer A es un peer edge, y se configura para utilizar al Peer R1 como su rendezvous. Cuando el Peer A inicia una petición de descubrimiento o de búsqueda, se la envía inicialmente a su peer rendezvous (R1, en este ejemplo) y también vía multicast a otros peers en la misma subred.

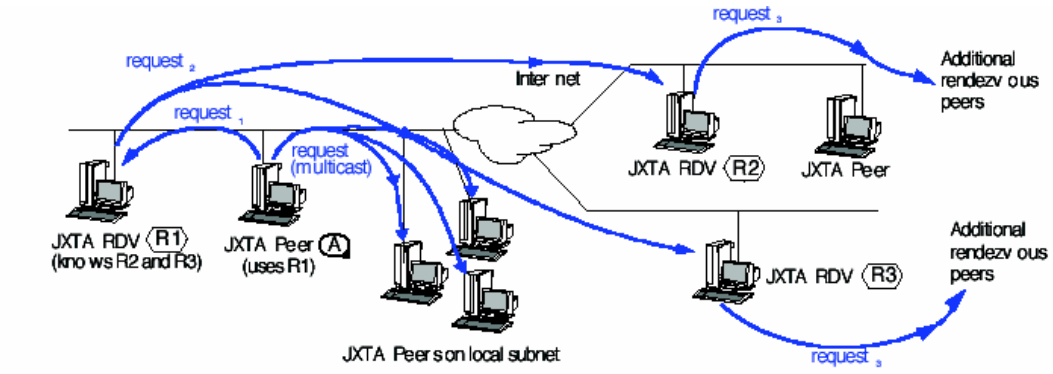


Figura *10. Propagación de Petición Vía Peers Rendezvous [6].*

Las consultas locales dentro de una subred se propagan a peers vecinos usando métodos de transportes como broadcast o multicast. Los peers que reciben la consulta responden directamente al peer que realizó la petición, si contienen la información en cache local.

Las consultas más allá de la subred se envían al peer rendezvous conectado. El peer rendezvous procura satisfacer la consulta buscando en su cache local. Si contiene la

información solicitada, contesta directamente al peer de la petición y no propaga más la petición. Si contiene el índice para el recurso en su SRDI, notificará al peer que publicó el recurso y este peer le responderá directamente al peer que realizó la petición. (Los rendezvous almacenan solamente el índice del anuncio, y no el anuncio en sí mismo).

Si el peer rendezvous no contiene la información solicitada, se utiliza un algoritmo para encaminar el sistema de rendezvous a buscar un rendezvous que contenga el índice. Una cuenta de saltos se utiliza para especificar el número máximo de veces que la petición puede ser remitida. Una vez la consulta alcanza el peer, este contesta directamente al originador de la consulta.

La figura 11 representa una vista lógica de cómo el servicio SRDI trabaja. El peer 2 publica un nuevo anuncio, y un mensaje SRDI se envía a su rendezvous, R3. Los índices serán almacenados en R3, y se pueden enviar a otro rendezvous en el grupo del peer. Ahora, el peer 1 envía una consulta solicitando este recurso a su rendezvous, R1. El rendezvous R1 comprobará su cache local de entradas SRDI, y propagará la consulta si no se encuentra. Cuando el recurso es localizado en el peer 2, el peer 2 responderá directamente a P1 con el anuncio solicitado.

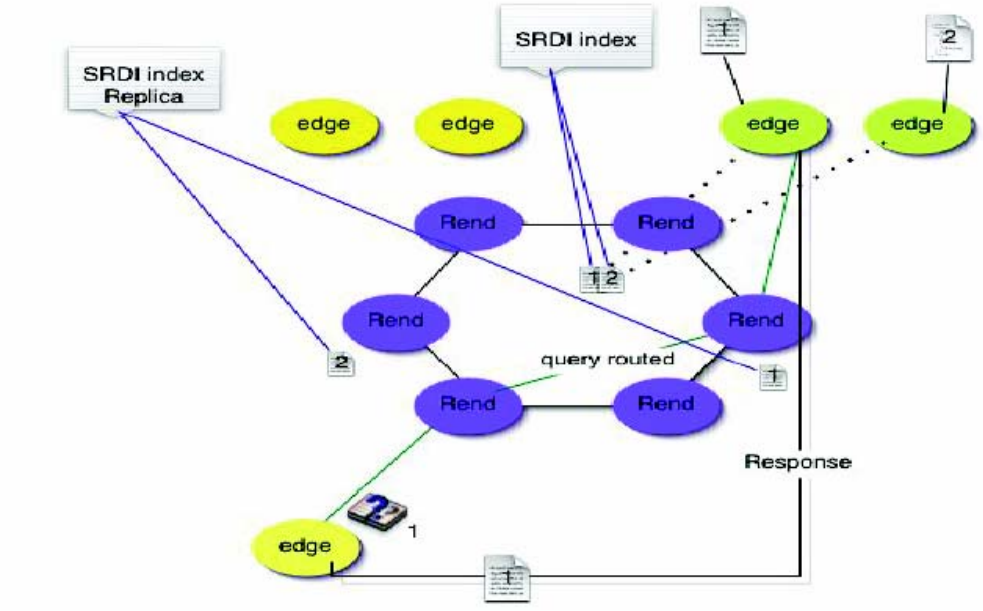


Figura 11. Servicio SRDI [6]

Protocolos JXTA [6] JXTA define una serie de formatos de mensajes XML, o protocolos, para la comunicación entre los peers. Los peers utilizan estos protocolos para descubrirse, anunciar y descubrir recursos en la red, y mensajes de comunicación y de ruta.

Hay seis protocolos JXTA:

- Peer Discovery Protocol (PDP)
- Peer Information Protocol (PIP)
- Peer Resolver Protocol (PRP)
- Pipe Binding Protocol (PBP)
- Endpoint Routing Protocol (ERP)
- Rendezvous Protocol (RVP)

Todos los protocolos JXTA son asincrónicos, y se basan en un modelo de pergunta/respuesta. Un peer JXTA utiliza uno de los protocolos para enviar una consulta a uno o más peers en su grupo. Puede recibir cero, una, o más respuesta a su consulta. Por ejemplo, un peer puede utilizar PDP para enviar una consulta de descubrimiento preguntando a todos los peers conocidos en el grupo del peer. En este caso, múltiples peers contestarán probablemente con respuestas al descubrimiento. En otro ejemplo, un peer puede enviar una petición de descubrimiento que pide un pipe específico llamado "aardvark". Si este pipe no se encuentra, cero respuestas serán enviadas en la contestación.

- Peer Discovery Protocol (PDP) Se utiliza para descubrir cualquier recurso publicado de un peer. Los recursos se representan como anuncios. Un recurso puede ser un peer, grupo de peers, pipe, servicio, o cualquier otro recurso que tenga un anuncio. PDP permite a un peer encontrar anuncios de otros peers. El PDP es el protocolo de descubrimiento por defecto para todos los grupos de peers definidos por el usuario y el grupo peer por defecto. Los servicios de descubrimiento personalizado pueden elegir no usar el PDP. Si un grupo peer no tiene su propio servicio de descubrimiento, el PDP se utiliza para probar peers para anuncios. Hay maneras

múltiples de descubrir información distribuida. El proyecto JXTA J2SE utiliza una combinación de IP multicast en la subred y el uso de peers rendezvous, una técnica basada en crawling. Los peers Rendezvous proporcionan el mecanismo de enviar peticiones a partir de un peer conocido al siguiente ("Crawling" alrededor de la red) para dinámicamente descubrir la información. Un peer puede ser preconfigurado con un predefinido conjunto de peers rendezvous.

- Peer Information Protocol Una vez se localiza un peer, sus capacidades y el estado pueden ser consultados. El protocolo de información del peer (PIP) proporciona un sistema de mensajes para obtener la información de estado del peer. Un mensaje PING de PIP se envía a un peer para chequear si el peer está vivo y conseguir información sobre el peer. El mensaje ping de PIP especifica si una respuesta completa (anuncio del peer) o un ACK deben ser retornados. Un mensaje PeerInfo se utiliza para enviar un mensaje en respuesta a un mensaje PING. Este contiene la credencial del remitente, la identificación del anuncio, la identificación del peer fuente y la identificación del peer objetivo, uptime, y el anuncio del peer.

- Peer Resolver Protocol (PRP) PRP permite a los peers enviar peticiones genéricas de consulta a otros peers e identificar respuestas que emparejan. Las peticiones de consultas se pueden enviar a un peer específico, o se pueden propagar vía los servicios rendezvous dentro del alcance de un grupo peer. El PRP utiliza el servicio de Rendezvous para diseminar una pregunta entre los múltiples peers, y utiliza mensajes unicast para enviar consultas a peers específicos. El PRP es un protocolo que apoya peticiones de consultas genéricas. PIP y PDP se construyen usando PRP, y proporcionan consulta/respuesta específicas. PIP se utiliza para consultar la información de estado específica y PDP se utiliza para descubrir recursos de peer. El PRP se puede utilizar para cualquier pregunta genérica que pueda ser necesaria en una aplicación. Por ejemplo, el PRP permite a peers definir e intercambiar preguntas para encontrar o buscar información de servicios tal como el estado del servicio, el estado de un pipe de punto final, etc. El mensaje de consulta del solucionador es usado para enviar solicitudes de consultas para un servicio a otro miembro del grupo peer. El mensaje de consulta del solucionador contiene la credencial del remitente, la identificación única de la consulta, un controlador de servicio específico, y la consulta. Cada servicio puede registrar un controlador en el

servicio solucionador del grupo para procesar peticiones de consultas y generar respuestas. El mensaje de respuesta del solucionador se utiliza para enviar un mensaje en respuesta a un mensaje de consulta. El mensaje de respuesta del solucionador contiene la credencial del remitente, una única identificación de la pregunta/consulta, un controlador específico del servicio y la respuesta. Múltiples mensajes de consulta pueden ser enviados. Un peer puede recibir cero, una, o más respuestas a una petición de consulta.

Los peers también pueden participar en el SRDI. SRDI proporciona un mecanismo genérico, donde los servicios de JXTA pueden utilizar un índice distribuido de recursos compartidos con otros peers que se agrupan como sistema de peers con más capacidades tales como peers rendezvous. Estos índices se pueden utilizar para dirigir consultas en la dirección donde la consulta tenga mas probabilidades de ser resuelta, y re-propagar mensajes a los peers interesados en éstos mensajes propagados. El PRP envía un mensaje de manejador SRDI al controlador en uno o más peers en el grupo. El mensaje de manejador SRDI se envía a un controlador específico, y contiene una secuencia que será interpretada por el controlador apuntado.

- Pipe Binding Protocol (PBP) PBP es utilizado por los miembros de un grupo peer para atar un anuncio de pipe a un punto final de pipe. La conexión virtual del Pipe (camino) se puede hacer sobre cualquier número de conexiones físicos de transporte de la red tales como TCP/IP. Cada extremo del pipe trabaja para mantener la conexión virtual y para reestablecerla, en caso de necesidad, atando o encontrando los puntos finales del pipe. Un pipe se puede ver una cola de mensajes abstracta, soportando crear, abrir/resolver (bind), cerrar (unbind), cancelar, enviar, y recibir operaciones. Implementaciones reales de pipes, pueden diferir, pero todas las implementaciones obedientes utilizan PBP para atar el pipe a un punto final. Durante la operación abstracta de crear, peer locales conectan el punto final del pipe al transporte del pipe. El mensaje de consulta PBP es enviado por un punto final de pipe del peer para encontrar un punto final de pipe limitado al mismo anuncio de pipe. El mensaje de consulta puede pedir por información no obtenida de la cache. Esto se utiliza para obtener información actualizada de un peer. El mensaje de consulta

también puede contener de manera opcional una identificación de peer, que si está presente indica que solamente el peer especificado debe responder a la consulta.

El mensaje de respuesta PBP es enviado de nuevo al peer de la petición por cada par limitado al pipe. El mensaje contiene la identificación del pipe, el par donde se ha creado el InputPipe correspondiente, y un valor booleano que indica si el InputPipe existe en el peer especificado.

- Endpoint Routing Protocol (ERP) ERP define un conjunto de mensajes request/query que se utilizan para encontrar información de enrutamiento. Esta información de ruta es necesaria para enviar un mensaje desde un peer (la fuente) a otro (el destino). Cuando un peer es solicitado para enviar un mensaje a una dirección de punto final de peer dada, él primero mira en su cache local para determinar si tiene una ruta a este peer. Si no encuentra una ruta, envía una petición consulta de la ruta a sus relays disponibles preguntando por información de la ruta.

Cuando un peer relay recibe una consulta de ruta, él comprueba si conoce la ruta. Si es así, devuelve la información de la ruta como una enumeración de saltos. Cualquier peer puede consultar un peer relay por información de una ruta, y cualquier peer en un grupo del puede ser un relay. Los peers relay típicamente guardan en su cache información de rutas. La información de una ruta incluye la identificación del par fuente, la identificación del peer de destino, tiempo-para-vivir (TTL) para la ruta, y una secuencia ordenada de las identificaciones del peers gateway. Dicha secuencia puede no ser completa, pero debe contener por lo menos el primer relay. Las peticiones de consultas de ruta son enviadas por un peer a un relay del peer para solicitar la información de la ruta. La consulta puede indicar una preferencia para puentear el contenido del router y buscar dinámicamente una ruta nueva.

Los mensajes de respuesta de la ruta son enviados por un par relay en respuesta a peticiones de información de rutas. Este mensaje contiene la identificación del peer destino, la identificación del peer y anuncio del peer del router que conoce la ruta hacia el destino, y una secuencia ordenada de uno o más relays.

- Rendezvous Protocol (RVP) El protocolo Rendezvous (RVP) es responsable de propagar mensajes dentro de un grupo peer. Mientras que diversos grupos peer pueden tener diversos medios de propagar mensajes, el protocolo Rendezvous define un protocolo simple que permite:

    - Peer conectar al servicio (ser capaz de propagar mensajes y recibir mensajes propagados
    - Controlar la propagación del mensaje (TTL, detección del loopback, etc.).

El RVP es utilizado por el protocolo PRP y por el protocolo PBP para propagar mensajes.

Complejidad Según lo descrito anteriormente, el modelo de búsqueda y enrutamiento JXTA cae dentro de la categoría de algoritmos de inundación, de igual forma lo confirma [7] en una breve comparación de los dos protocolos de búsqueda.

El modelo de inundación es un modelo P2P puro, en donde, cada petición de un peer se inunda (difusión/broadcast) a los peers directamente conectados, los cuales a su vez inundan a sus peers etc., hasta que se contesta la petición o un número máximo de los pasos del inundamiento (típicamente 5 a 9) ocurren. Este modelo es utilizado típicamente por Gnutella y consume mucho ancho de banda de la red. Consecuentemente, las características de escalabilidad del modelo se confunden a menudo con sus características de alcanzabilidad. Si la meta es alcanzar a todos los peers en la red, entonces este modelo no demuestra ser muy escalable, sino que es eficiente en comunidades limitadas tales como una red corporativa [10]. Manipulando el número de conexiones de cada nodo y apropiadamente fijando el parámetro de la TTL de los mensajes de la petición, el modelo inundado de peticiones puede escalar a un horizonte accesible de centenares de millares de nodos. [10]

Para Gnutella como lo afirma [10] , el número de mensajes enviados en una búsqueda en este sistema esta dado en función K, es decir el número promedio de vecinos y la variable TTL (Time-To-Live, tiempo para vivir), como K^{TTL} , debido a que la variable TTL es un parámetro que se utiliza para disminuir el número de mensajes en la red , si no existe este parámetro Gnutella entiende que es TTL se puede igualar al número de

nodos de la red por lo que se puede cambiar la formula a K^N donde N es el número de nodos en la red.

Como se explicó en secciones anteriores, JXTA incluye una jerarquía de peers, dentro de las cuales se encuentran los peers rendezvous, quienes son los encargados de la transmisión de mensajes de una petición, además de almacenar información del contenido que manejan los peers de los cuales es responsable. Debido a esta característica son estos los peers que facilitan el proceso de búsqueda y es entre ellos donde se realiza la interacción de mensajes de búsqueda. Debido a esto y basándonos en la complejidad expuesta para la red Gnutella, se puede obtener una complejidad aproximada para JXTA con Ns número de peer rendezvous y ks número promedio de vecinos por peers rendezvous como Ks^{Ns}

1.3 SEGURIDAD PEER TO PEER

La seguridad en los sistemas informáticos se ha convertido en uno de las características más importantes de su diseño. Las aplicaciones peer to peer no son la excepción, y mas aún teniendo en cuenta que la comunicación entre peers es directa y no hay un intermediario o entidad que garantice de forma segura dicha comunicación.

Los objetivos básicos que un sistema seguro debe cumplir se resumen en [12]:

1. Disponibilidad: es un requisito para asegurar que los sistemas trabajan puntualmente y que el servicio no está negado a los usuarios autorizados. Este objetivo protege contra:

    a. Tentativas intencionales o accidentales para desarrollar borrados no autorizados de datos o que se pueda causar una negación de servicio o de datos
    b. Procurar utilizar el sistema o los datos para propósitos no autorizados

2. Integridad: tiene dos facetas:
    a. Integridad de datos: propiedad que los datos no se han alterado de una manera desautorizada mientras se estaba almacenado, durante el procesamiento o cuando estaba en tránsito.
    b. Integridad del Sistema: la calidad que un sistema tiene al realizar una función prevista en una manera intacta, libre de manipulación no autorizada.

3. Confidencialidad: es el requisito que la información privada o confidencial no sea divulgada a individuos no autorizados. La protección de la confidencialidad se aplica a los datos en el almacenaje, durante el procesamiento, y mientras que está en tránsito.

4. Responsabilidad: es el requerimiento necesario para que las acciones de una entidad se puedan remontar únicamente a esa entidad. Esto abarca la no-repudiación, y se relaciona con el control de acceso y a la autenticación.

5. Aseguramiento: es la base para la confianza que las medidas de seguridad, técnicas y operacionales, trabajan según lo previsto para proteger el sistema y la información que este procesa. El aseguramiento es esencial; sin él los otros objetivos no se resuelven.

6. Privacidad: es el requerimiento de una entidad para determinar el grado con el cual esta interactúa con su ambiente, incluyendo buena voluntad de compartir la información sobre sí mismo con otras.

Los requisitos anteriormente citados son válidos de igual forma para aplicaciones peer-to-peer y en el desarrollo de este documento se enunciarán algunas de las aproximaciones para resolverlos.

Pero antes es importante anotar que debido a la naturaleza distribuida de los sistemas P2P, el uso de una entidad central como parte esencial de los procesos de seguridad, desaparece y por el contrario cada peer es responsable de velar por la seguridad del sistema. Lo cual hace mas confiable al sistema pues no hay un punto de falla único,

pero aumenta complejidad en el manejo de todas las políticas de aseguramiento. A pesar de esto, y debido a los diferentes tipos de redes P2P que se pueden formar (parcialmente centralizada, hibrida, totalmente descentralizada) la mayoría de las soluciones planteadas para el manejo seguro de las aplicaciones aun son delegadas a una o varias entidades dentro del sistema.

En las siguientes secciones se iniciará con algunos conceptos o más bien metodologías criptográficas usadas para dar solución a problemas concernientes a seguridad. Y luego se expondrán algunos sistemas de seguridad P2P.

1.3.1 Diseño de Sistemas Seguros

Encripción El cifrado o encripción, es el proceso de convertir los datos (texto plano) en una forma sin sentido (texto encriptado) de modo que llegue a ser inteligible únicamente a las partes autorizados. El cifrado se basa en llaves. Al igual que las llaves en el mundo real abriendo bloqueos, las llaves de cifrado pueden abrir el texto cifrado (bloqueado).

Basadas en el uso de llaves, la encripción se puede clasificar en dos amplias categorías -- cifrado simétrico y cifrado asimétrico.

- Encripción Simétrica La encripción simétrica es el tipo de encripción más usada hoy usado y era la única disponible hasta que la encripción asimétrica fue introducida.

En un algoritmo criptográfico simétrico, los procesos de cifrado y de descifrado utilizan la misma llave. Por lo tanto, el remitente y el receptor del mensaje deben tener una llave secreta compartida por la cual cifran y descifran mensajes enviados entre ellos. En la figura 14, se puede ver un mensaje que es pasado entre los usuarios Alicia y Bob. Están utilizando una sola llave para cifrar y para descifrar un mensaje. Se aclara que la llave fue intercambiada entre ellos vía otro canal. Observe también que Alicia dio la llave directamente a Bob vía un cierto método (idealmente de una manera que

evita que otros vean la llave, porque cualquier persona puede descifrar el mensaje si la tienen).

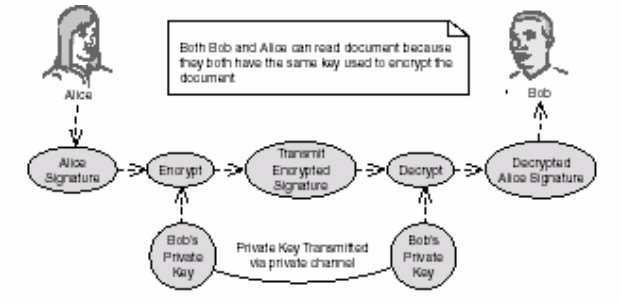


Figura 14. Encripción Simétrica [13]

Mientras que este método es una manera simple pero poderosa para la comunicación segura, tiene sus desventajas debido a esa sola llave. Requiere que una llave secreta se utilice para cualquier par de entidades que se comunican. Aunque esto es práctico para las comunicaciones en escala reducida, no es escalable para un sistema en grande donde se comunican entre sí una gran cantidad de usuarios si todos los pares de usuarios tienen llaves únicas de modo que ningún otro, inclusive en el mismo grupo, pudiera pasar mensajes secretos. Sin embargo, es un buen método si un grupo de usuarios necesitan enviar mensajes que se pueden leer por cualquier persona con una llave.

La otra preocupación está sobre compartir la llave. ¿Cómo comparten un remitente y un receptor una llave de cifrado con seguridad inicialmente? Se perdería el propósito si un canal de comunicaciones inseguro fue utilizado. Por lo tanto, tales sistemas dependen de un *out-of-band* asegurando el canal para compartir la llave. ¿Y si un canal tan seguro estaba presente, por qué no usar éste para la transferencia de datos también?

Sin embargo, el cifrado simétrico es aún utilizado ampliamente. Comparado a otras formas de cifrado, el cifrado simétrico es muy rápido, así que llega a ser cada vez más

importante pues el tamaño de los datos incrementa el cifrado. Por lo tanto, se elige el cifrado simétrico cuando los datos que se cifrarán son grandes.

Dos de los algoritmos simétricos más populares del cifrado son RC4 y DES. DES (*Data Encripción Strandard*) fue publicado por FIPS 46 (*Federal Information Processing Standards Publication*). RC4 cuyo dueño es RSA.

- Encripción Asimétrica El cifrado asimétrico o encripción de llave pública está llegando a ser cada vez más popular como medio seguro de comunicación múltiple. Confía en dos llaves (por lo tanto, asimétrico) para las interacciones seguras. Cada usuario que participa tiene un par de llaves generadas por un algoritmo. Las llaves tienen una característica matemática única que cualquier mensaje cifrado por una se pueda descifrar solamente por la otra.

El usuario es libre de señalar una de éstas como llave pública y la otra como llave privada. La llave pública se utiliza para representar al usuario y puede ser distribuida extensamente en cualquier canal que el usuario desea. El usuario guarda la llave privada de manera confidencial. Cualquier remitente que desee comunicarse con el usuario puede utilizar la llave pública del usuario para cifrar el mensaje. El resto es asegurado, únicamente el usuario podrá descifrar el mensaje. De igual forma, si el usuario desea responder a otro usuario, él o ella pueden cifrar la llave pública del otro para cifrar el mensaje. Esto puede ir un paso más lejos, y el usuario puede cifrar el mensaje con su llave privada y la llave pública del receptor para asegurar que únicamente la persona apropiada reciba el mensaje pero también para que el receptor reconozca la fuente del mensaje.

Un ejemplo de cómo este tipo de cifrado trabaja se muestra en la figura 15. Primero, el usuario Alicia cifra el documento para Bob con la llave pública de Bob. El mensaje se transmite y Bob desencripta el mensaje con su llave privada. El proceso es idéntico para otros usuarios, diferentes de Alicia, que tienen acceso a la llave pública de Bob. El desciframiento se puede hacer solamente por Bob -- mientras Bob no deje copiar a cualquier persona su llave privada.

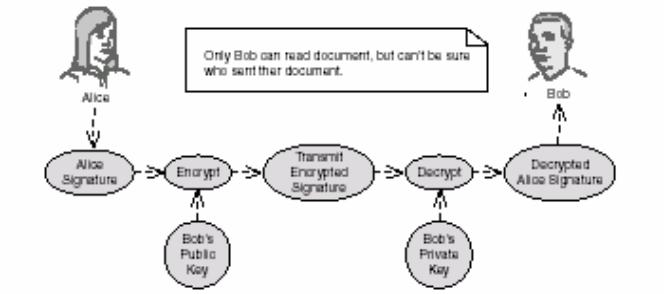


Figura 15. Encripción asimétrica. [13]

Debido a que la encripción es reversible (es decir, la llave privada se puede utilizar para cifrar y la llave pública para descifrar, los mensajes pueden ser creados esto esencialmente prueba que vinieron del dueño de la llave). Es decir según lo demostrado en la figura 16, si el mensaje es cifrado por Alicia usando su llave privada, Bob puede descifrarla usando la llave pública de Alicia. Si, según lo demostrado en esta figura, el documento se cifra primero con la llave pública de Bob y luego la llave privada de Alicia, el documento es solamente descifrado por Bob y, porque solamente la llave pública de Alicia puede cifrar este contenido, sólo Alicia habría podido enviar el mensaje.

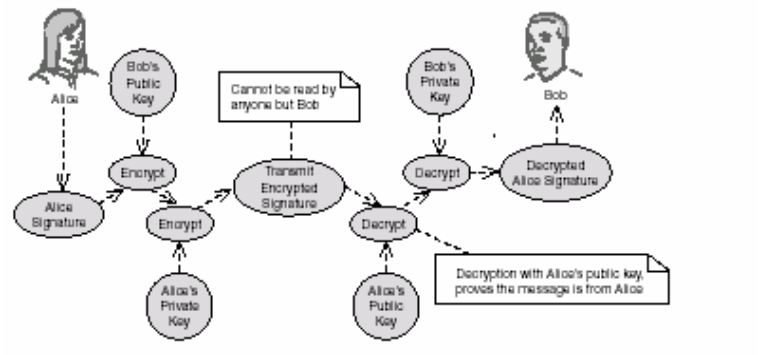


Figura 16. Autenticación Asimétrica. [14]

La aproximación resuelve el problema principal con el cifrado simétrico causado por el intercambio de lo que aquí es una llave privada ya que las llaves privadas nunca se intercambian. La cuestión de cómo transmitir con seguridad llaves privadas no se presenta. Los procesos del cifrado/descifrado tienen una desventaja ya que consumen más tiempo de transformación y memoria para los cálculos que el cifrado

simétrico, como se afirma en [14] "los algoritmos de encripción pública usualmente requieren de 100 a 1000 veces mas potencia de procesamiento que los algoritmos de llaves simétricas". A menos que se esté transfiriendo cantidades grandes de datos, el tiempo de ejecución de los algoritmos es manejable para la mayoría de las aplicaciones (segundos a unos pocos minutos dependiendo del hardware y del tamaño del mensaje).

Un buen acercamiento es utilizar una combinación de llaves simétricas y asimétricas. El cifrado de llave pública se puede usar para intercambiar con seguridad un sistema de llaves simétricas. Después de que se haga esto, el cifrado simétrico se puede utilizar para cifrar todos los intercambios subsecuentes, así que usted puede beneficiar del los dos. SSL y TLS, por ejemplo, utilizan tal acercamiento.

CERTIFICADOS Un certificado digital es un documento que contiene una sentencia (generalmente corta) firmada por un principal [14]. En [15] se define así:

"Un certificado es una forma de credencial. Algunos ejemplos pueden ser la licencia de conducción, la tarjeta de Seguridad Social, o la partida de nacimiento. Cada uno de éstos tiene cierta información sobre él que identifica y una cierta autorización que indica que algún otro ha confirmado su identidad. Algunos certificados, tales como el pasaporte, son bastante importantes en la confirmación de la identidad que nadie desearía perderlos, a fin de que alguien los utilice para autenticar.

Un certificado digital son datos que funcionan como un certificado físico. Un certificado digital es información incluida con la llave pública de una persona que las ayudas a otras a verificar que una llave sea genuina o válida. Los certificados digitales se utilizan para frustrar tentativas de sustituir la llave de una persona por otra. Un certificado digital consiste de tres cosas:

- Una llave pública.
- Información del certificado (información de la "identidad" del usuario, tal como nombre, identificación del usuario, etcétera).
- Una o más firmas digitales.

El propósito de la firma digital en un certificado es indicar que la información del certificado ha sido atestiguada por alguna otra persona o entidad. La firma digital no atestigua la autenticidad del certificado en su totalidad; atestigua solamente que la información firmada de la identidad va junto con, o está limitada a, la llave pública.

Así, un certificado es básicamente una llave pública con una o dos formas de identificación unidas, más una estampilla de la aprobación de algún otro individuo confiable. "

- DISTRIBUCIÓN DE CERTIFICADOS. [15] Los certificados se utilizan cuando es necesario intercambiar llaves públicas con alguna otra persona. Para grupos pequeños de gente que desean comunicarse con seguridad, es fácil intercambiar manualmente los diskettes o emails que contienen la llave pública de cada dueño. Esta es distribución de llave pública manual, y es práctica únicamente hasta cierto punto. Más allá de ese punto, es necesario poner sistemas en un lugar que pueda proporcionar la seguridad necesaria, almacenamiento y mecanismos de intercambio para que así compañeros de trabajo, socios de negocio, o cualquiera puedan comunicarse si es necesario. Estos pueden venir en forma de depósitos de almacenamiento únicamente, llamados Servidores de Certificados, o sistemas más estructurados que proporcionen características adicionales en el manejo de llaves llamadas Infraestructuras de Llave Publica (PKIs).

o Servidores de Certificado: Un servidor de certificados también llamado servidor de llave, es una base de datos que permite a los usuarios a enviar y recuperar certificados digitales. Un servidor de certificado proporciona generalmente algunas características administrativas que permiten a una compañía a mantener sus políticas de seguridad -- por ejemplo, permitiendo únicamente a esas llaves que resuelvan ciertos requisitos para ser almacenado

o Infraestructuras de Llave Pública (PKIs): Una PKI contiene las facilidades de almacenamiento de certificados para un servidor de certificados, pero también proporcionan las facilidades para el manejo del certificado (la capacidad de publicar, revocar, almacenar, recuperar, y confiar certificados). La característica principal de un

PKI es la introducción lo que es conocido como una Entidad o Autoridad Certificadora, o CA, que es una entidad humana (persona, grupo, departamento, compañía, u otra asociación) que una organización ha autorizado para publicar certificados a sus usuarios a. (el rol de una CA es análogo a la oficina de pasaportes del gobierno de un país.) Una CA crea certificados y los firma digitalmente usando la llave privada de la CA. Debido a su rol en la creación de certificados, la CA es el componente central de una PKI. Usando la llave pública de CA, cualquier persona que desee verificar la autenticidad de un certificado verifica la firma digital de la CA que publica, y por lo tanto, la integridad del contenido del certificado (lo más importante, la llave pública y la identidad del dueño del certificado).

Firmas Digitales La criptografía se emplea para implementar un mecanismo conocido como firma digital. Esta emula el papel de las firmas convencionales, verificando a una tercera parte que un mensaje o un documento es una copia inalterada producida por el firmante [14]. Existen dos aproximaciones para la generación de firmas digitales MAC y Firmas Digitales con Llaves Públicas.

- Códigos de Autenticación de Mensajes (MAC). [13] Un Código de Autenticación de Mensaje, comúnmente denominados MAC, es una manera simple de asegurar la autentificación. MAC confía en la encripción de llaves simétricas, es decir asume que el remitente y el receptor comparten una llave secreta común. Una manera simple de generar un MAC es:

    1. Concatenar el mensaje con la llave secreta (o combinarlos de cierta otra manera)
    2. Hacer hash de los datos resultantes para conseguir el MAC.
    3. Anexar el MAC al mensaje que se enviará.

El receptor puede, alternadamente, recibir el mensaje y el MAC, recalcular el MAC desde del mensaje y la llave secreta, compararlo con el MAC recibido, y verificar su autenticidad.

- Firmas Digitales con Llaves Públicas. [13] Las firmas Digitales de Llaves Públicas son otra forma de autentificación. Muchos sistemas populares disponibles hoy, tales como Pretty Good Privacy (PGP), se basa en esta metodología. Este tipo de firma digital confía en el concepto de encripción asimétrico. La figura 17 demuestra cómo Alicia puede crear una firma encriptada que Bob pueda descifrar con la llave pública de Alicia para probar que la firma era de Alicia.

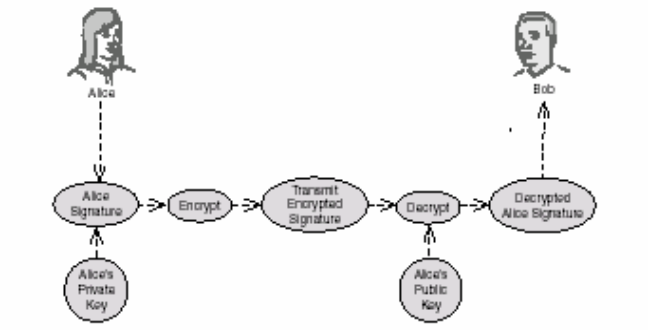


Figura 17. Proceso de Firma Digital de Llaves Públicas. [13]

Las firmas digitales de llaves públicas se pueden utilizar para firmar documentos incluyendo el documento en la firma o cifrando un documento ya encriptado (como en el cuadro 8,5). De manera opcional, que es lo más usado, una suma de comprobación del documento se calcula y es almacenada en la firma, luego que el receptor descifre la firma, la suma de comprobación encontrada en el mensaje se compara a la suma de comprobación del documento transmitido. Esto es simple y rápido porque únicamente una suma de comprobación pequeña se procesa con los algoritmos criptográficos. Además de actuar como firma, esta suma también prueba que el documento no se ha modificado desde que el documento fue firmado.

Los datos se pueden firmar digitalmente usando los siguientes pasos:

1. Calcular el hash del mensaje que se firmará.
2. Encriptar este hash con la llave privada.
3. Los datos encriptados obtenidos son la firma digital y se pueden unir con el documento.

El receptor desencripta la firma empleando la llave pública del emisor y calcula el hash del mensaje. Si concuerda el hash con la firma desencriptada, la firma es válida.

- Funciones Hash o de Resumen [14] Según [14] una función de resumen segura h = H(M), donde M es el mensaje, H(M), la función aplicada, y h el valor obtenido debería poseer las siguientes propiedades:

1. Dado *M*, debe ser fácil calcular *h*
2. Dado *h*, debe ser extremadamente difícil calcular *M.*
3. Dado *M*, debe ser extremadamente difícil encontrar otro mensaje *M'* , tal que *H(M) = H(M')*

Dos funciones de resumen o hash son ampliamente utilizadas son el algoritmo MD5 (llamado así por que es el quinto de una serie de algoritmos hash desarrollados por Ron Rivest) y el SHA (*Secure Hash Algoritm*) adoptado para su estandarización por el NIST. Ambos han sido probados y analizados cuidadosamente, y puede considerarse que son apropiadamente seguros durante el futuro previsible, al mismo tiempo que son razonablemente eficientes.

- MD5: El algoritmo MD5 emplea cuatro vueltas, cada una aplicando una de cuatro funciones no lineales a cada uno de los dieciséis segmentos de 32 bits de un bloque de texto fuente de 512 bits. El resultado es resumen de 128 bits. MD5 es uno de los algoritmos más eficientes disponibles hoy día.
- SHA: SHA es un algoritmo que produce un resumen de 160 bits. Se basa en el algoritmo de Rivest MD4 con algunas operaciones adicionales. Es sustancialmente mas lento que MD5, pero el resume de 160 bits ofrece una mayor seguridad contra los ataques de fuerza bruta y del cumpleaños.

Confianza "La Noción de Confianza es un fenómeno que los humanos usan cada día para promover interacción y aceptar riesgos en situaciones donde sólo información parcial está disponible, permitiendo a una persona que asuma que otra se comportará como se esperaba. A pesar de los estudios extensivos en las ciencias sociales (sociología, psicología y filosofía), el concepto de confianza aun sigue siendo efímero,

causa de su rigurosa definición. Esto es debido en parte a que la confianza es en gran parte invisible e implícita en sociedad" [16].

Generalmente confianza se asocia o relaciona con términos como "confidencia, creer, fe, esperanza expectativa, dependencia y confianza sobre la integridad, habilidad o carácter de una persona o cosa. " [16] Pero básicamente, su definición suele clasificarse como subjetiva, pues depende generalmente del punto de vista del autor.

Basados en el concepto o mas bien noción de confianza, algunos trabajos de investigación en lo que tiene que ver con seguridad, han utilizado aproximaciones del concepto, para definir modelos donde "la confianza y el riesgo se ligan inexorablemente y la necesidad que ambos sean considerados al tomar una decisión sobre una ruta ambigua, el resultado depende de las acciones de otra"[16].
En el mundo cliente/servidor, la confianza se considera muy simple. La gente generalmente confía en una autoridad central, tal como Verisign. Esto también hará que confíen en el resto de entidades certificadas por Verisign. Así, una jerarquía completa de confianza se puede construir con cada entidad confiable certificando que otra puede ser confiable. En este ejemplo, Verisign formaría la base de esta jerarquía.

La confianza es un concepto que se maneja diferente en sistemas P2P debido a su carencia de centralización. Debido a que no hay autoridad confiable central para comenzar, el concepto entero de la confianza usado en redes cliente/servidor llega a ser sin sentido en P2P.

Muchos acercamientos interesantes se han formulado para acercar a esta solución. Un acercamiento es que cada peer clasifique a otro peer basado en sus interacciones con él. La puntuación o *ratings* pueden ser firmados para poderlos autenticar si son requeridos. Así, cada peer construye una lista de la confianza, que se puede analizar en cualquier momento, al igual que la confianza evaluada totalmente.

1.4 APLICACIONES SEGURAS PEER TO PEER EXISTENTES

1.4.1 Proyecto SECURE

En la figura 5 se muestra la arquitectura básica de SECURE basado en la noción de confianza.

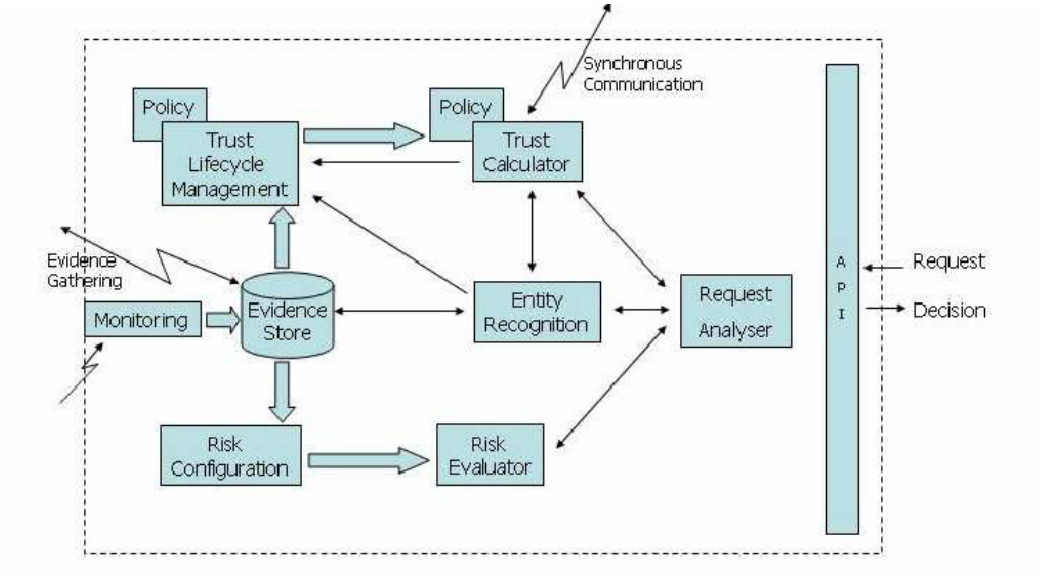


Figura 18. Framework SECURE [16]

Su funcionamiento comienza "Cuando una petición de interacción es realizada por un principal, p, esta pasa a través del API llega al analizador de la petición (Request Analyser). Este ultimo solicita la información sobre p a partir de tres fuentes, del Componente de Reconocimiento de la Entidad (Entity Recognition), de la Calculadora de Confianza (Trust Calculator) y del Evaluador del Riesgo (Risk Evaluator)" [16].

La interacción de los componentes del modelo para llevar a cabo el proceso de acceso seguro se define a continuación:

1. "La Verificación de que p sea reconocido se solicita al componente de Reconocimiento de la Entidad (Entity Recognition), que es responsable de reconocer entidades nuevas o previamente encontradas. El componente de Reconocimiento de la Entidad (Entity Recognition) se puede consultar por

cualquiera de los otros componentes para proporcionar cada vez que sea necesario capacidades de reconocimiento" [16].

2. "El analizador de peticiones solicita un cálculo a la Calculadora de Confianza. La calculadora de confianza computa el menor punto fijo, usando la información recopilada del componente de Manejo del Ciclo de Vida de Confianza y de su política local de confianza. El proceso de calcular un nivel de confianza para p se puede delegar a otra entidad, iniciando comunicación síncrona con una entidad remota" [16].

3. "La política local de la calculadora de confianza es actualizada basada en la información alimentada desde el componente de Manejo del Ciclo de Vida de Confianza. Este componente maneja la formación, la evolución, y explotación de la confianza basada en los datos copiados del Almacén de Evidencias. " [16].

4. "El almacén de evidencia es donde se almacenan todos los datos de confianza y riesgo relacionado. El almacén de la evidencia se pone al día con los datos recogidos del Recolector de evidencia, por ejemplo recomendaciones y actualizaciones de seguridad recogidas en un proceso asincrónico, y del componente de supervisión. El almacén de evidencia también está respondiendo a peticiones por recomendación de otras entidades" [16].

5. "El componente de supervisión observa la interacción real con p, así como el transporte de los resultados de tal interacción al almacén de evidencia" [16].

6. "El analizador de la petición también solicita una estimación del riesgo desde el evaluador del riesgo. El evaluador de riesgo calcula el riesgo potencial de la petición, basado en la información local almacenada en el componente de configuración del riesgo, el cual es actualizado con la información del almacén de evidencia. Toda la información obtenida sobre p es estimada y agregada.

El cálculo de la confianza y el riesgo se devuelve al analizador de la solicitud. El analizador de la petición es capaz de proporcionar una decisión con respecto a p en relación con a la interacción posible." [16].

Este modelo es aplicable para manejar decisiones de enrutamiento en redes Ad Hoc, aplicaciones P2P de colaboración y transacciones de M-Commerce [16].

Entre otros sistemas P2P que usan la noción de confianza para establecer relaciones seguras, están TrustMe [17] y Poblano [18].

1.4.2 MOTION: Sistema De Control De Acceso [19]

MOTION (MObile Teamwork Infrastructure for Organizations Networking) es una arquitectura orientada al servicio que soporta grupos de trabajo móviles y confía en un middleware P2P. En la figura 2 se muestra la arquitectura básica de MOTION.

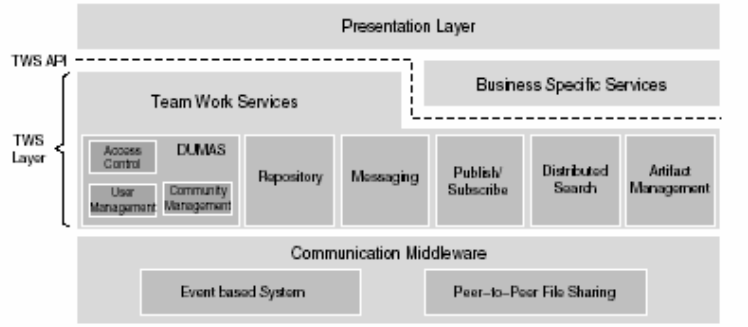


Figura 19. Arquitectura de MOTION [19]

Para el caso que no concierne se analizará la parte correspondiente al control de acceso la cual es facilitada por DUMAS (Dynamic User Management and Access Control System).

DUMAS tiene tres funcionalidades principales:

Control de Acceso: Esta compuesta de primitivas para asignar permisos de acceso a usuarios y roles, remover permisos de acceso, definir nuevos permisos de acceso y asignar semánticas (tales como métodos) a estos.
Manejo de usuarios: consiste en registrar nuevos usuarios al ambiente de MOTION, asignarles y removerles roles, y borrarlos. Los usuarios, obviamente, heredan permisos de acceso de los diferentes roles que ellos son miembros.
Manejo de comunidad: Una comunidad representa un conjunto de usuarios compartiendo recursos. Operaciones sobre comunidades incluyen crearlas, borrarlas, hacerlas sub-comunidad de otras comunidades y removerlas de otras comunidades.

La arquitectura de DUMAS esta inspirada en el middleware subyacente de P2P para compartir archivos y el requerimiento para soporte de movilidad. Dos categorías de peers son distinguidas:

Peers L1: Los pares L1 son los pares que tienen la capacidad de mantener una infraestructura regular de seguridad. Esto incluye la inteligencia completa para asignar permisos, quitar permisos, validar y almacenar listas del control de acceso (ACLs). Una característica importante de un par L1 es que es responsable de proteger algunos recursos. La responsabilidad de proteger un servicio incluye almacenar ACLs relacionadas con este servicio y entregar certificados de autorización. Para utilizar un servicio, un usuario debe presentar sus capacidades al proveedor de servicio (Ver figura 2). Estas capacidades son certificados de autorización (ACs), entregados por los peers L1 basados en ACLs recibidas. Cada peer dispone un par de claves publica/privada.

Peers L2: Los peers L2 son dispositivos que carecen de recursos para instanciar de manera total la maquina DUMAS.
Una primitiva, como la adición de un usuario a una comunidad, consiste de tres pasos: validar el sistema de certificados de autorización presentado por el usuario (V), ejecutar la función verdadera (E), y publicar el evento (E). Los pares realizan todas o un subconjunto estas tareas dependiendo de sus niveles. Las cuales se resumen en la tabla 4 para los dos tipos de peers (L1 y L2).

Generalmente, para utilizar un servicio, un usuario debe presentar sus certificados de autorización a los proveedores de servicio (Ver Figura 3). Mientras que él está en línea, el usuario realiza una búsqueda distribuida que especifica la clase de certificados de autorización que él está buscando. Algunos de los peers L1 pueden contestar, enviando algunos certificados de autorización por un período corto de validez. Estos certificados de autorización son bastante pequeños (hasta 300 octetos) para poderlos almacenar en los peers L2 y presentar a los proveedores de servicio en el intercambio de servicios. Un certificado de la autorización tiene la forma (permiso de acceso, identificación del usuario, identificación del objeto, fecha de vencimiento, firma del peer). La firma es la del par L1 que entrega el AC. Este peer debe ser colocado como uno de los peers responsables de manejar ACLs para el servicio para el cual entregó los certificados de autorización. Cualquier acción realizada en el ambiente de MOTION se acompaña con un sistema de certificados de autorización.

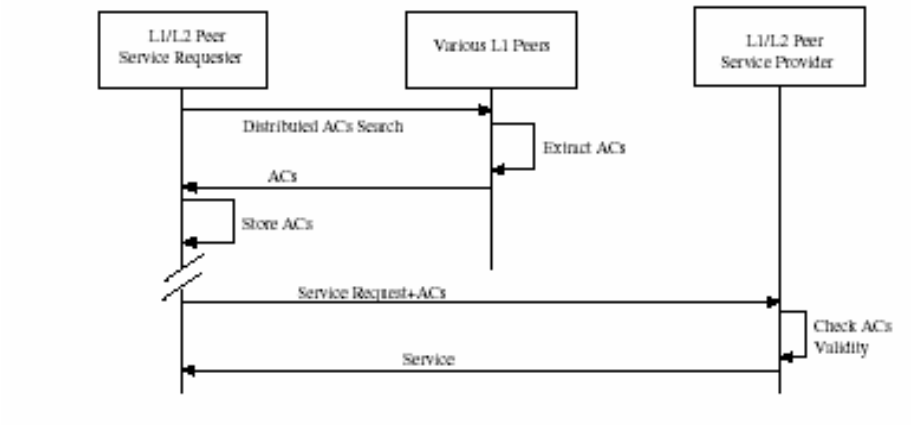


Figura 20. Proceso de Autorización en MOTION [19]

1.4.3 SCISHARE [20]

SCISHARE es un sistema P2P para compartir archivo, cuyo objetivo principal es proveer confidencialidad e integridad en toda comunicación, y hacer cumplir el control de acceso de recursos (canales de comunicaciones e información compartida).

El modelo de seguridad de SCISHARE adapta un número de soluciones de seguridad construidas alrededor de la infraestructura de la llave pública (PKI) X.509 para proporcionar confidencialidad, integridad, autentificación, y autorización.

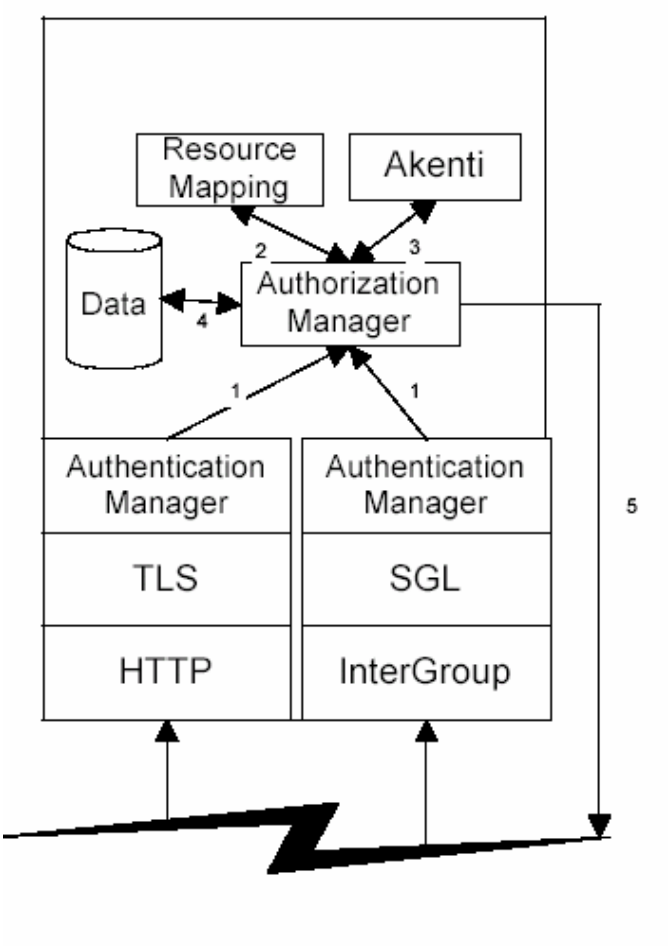


Figura 21. Arquitectura de un Peer SCISHARE

La figura 21 muestra la arquitectura en un peer. SGL se utiliza para enviar y recibir consultas/*querys* dentro de un grupo y TLS se utiliza para el resto de la comunicación. Los encargados de la autenticación (*Autentication Manager*) son utilizados para proporcionar control de acceso de granularidad gruesa a los canales de comunicaciones. El encargado de la autorización (*Authorization Manager*) y el motor de *Akenti* para proporcionar control de acceso de granularidad fina a los canales de comunicaciones y a la información compartida. Cuando el peer recibe un mensaje de la consulta o mensaje de solicitud de transferencia primero es procesado por el

Autentication Manager apropiado. Las consultas son procesadas por el *Autentication Manager* asociado a SGL y las peticiones de la transferencia son procesadas por el que está asociado a TLS. Si el *Autentication Manager* aprueba el mensaje, se pasa al *Authorization Manager* (paso 1). Si el mensaje es una petición de transferencia, el *Authorization Manager* determina la política asociada al contenido solicitado solicitado desde *Resource Mapping* (paso 2). El componente *Resource Mapping* mantiene la relación entre un recurso y su política asociada. La política y la identidad del solicitante son luego manejadas por el motor Akenti, el cual retorna las acciones que se permiten al solicitante se realicen en el recurso (paso 3). Si el solicitante tiene acceso de lectura, los datos se recuperan del almacén de datos (paso 4) y enviados al solicitante (paso 5).

SGL. SGL proporciona los servicios de seguridad requeridos por las aplicaciones que utilizan comunicación confiable de grupo en ambientes de área amplia (wide-area). SGL establece canales multicast seguros entre componentes de aplicación. Un canal de comunicaciones multicast seguro de SGL es establecido primero intercambiando una llave de sesión entre los componentes de aplicación legítimos. Esta llave entonces se utiliza para alcanzar confidencialidad de mensaje multicast e integridad de los datos del multicast dentro del grupo.

Akenti. Akenti proporciona servicios escalables de autorización en ambientes de red altamente distribuida. Los dueños del recurso pueden, de una manera flexible y segura, definir y desplegar remota e independientemente las políticas distribuidas del recurso usadas para el control de acceso de granularidad fina. El motor de Akenti, utiliza conocimiento centralizado mínimo para recoger con seguridad la política distribuida en forma de certificados durante la fase de evaluación. Después de evaluar la política para un recurso, Akenti puede publicar un token de autorización firmado al dueño de una llave pública X.509, autorizándolo para realizar acciones con respecto al recurso.

2 DISEÑO

En este capitulo se comparan los diferentes protocolos de búsqueda teniendo en cuenta diferentes parámetros tales como complejidad y capacidad de procesamiento para llegar a proponer soluciones de implementación adecuadas en redes P2P móviles. De igual forma se plantea una solución sencilla a problemas básicos de seguridad tales como autenticación e integridad, teniendo en cuenta las restricciones de procesamiento en dispositivos móviles.

2.1 COMPARACION DE PROTOCOLOS DE BUSQUEDA

Los protocolos de búsqueda, revisados en este documento, presentan orientaciones diferentes para solucionar el problema de localización en redes distribuidas como son las peer-to-peer. En esta sección se realizará una breve comparación de cada uno de ellos en cuanto a escalabilidad, complejidad y capacidad de procesamiento, basados en resultados obtenidos en otros trabajos de investigación.

Antes de revisar cada uno de los puntos de comparación, es necesario hacer una diferenciación entre CAN y CHORD con JXTA. CAN y CHORD basan su mecanismo de búsqueda a través de topologías geométricas (anillo, espacio cartesiano) realizando proyección de contenido (a través de una función hash) a llaves almacenadas en una zona específica. Mientras que JXTA, cae dentro de la categorización de redes de inundación, en donde los superpeer inundan la red para conseguir un recurso.

2.1.1 Complejidad

La siguiente tabla hace una comparación de las complejidades de cada uno de los sistemas, además de algunas características principales en las que difieren:

	CAN	CHORD	JXTA
Variables	d: número de dimensiones. N: número de nodos	N: número de Nodos	N_s: número de Nodos actuando como superpeers. k_s: número promedio de vecinos por nodo.
Topología	Espacio Cartesiano	Anillo	Aleatoria
Longitud de Ruta	$O(dN^{1/d})$	O(log N)	Variable, Aproximadamente Ks^{Ns}
Tamaño Tabla de enrutamiento	2^d	Log N	Variable

Resultados de trabajos de investigación reflejan la superioridad de CHORD, al compararlo con CAN, y redes de inundación tal como lo muestra la figura 22. Dicha gráfica representa una comparación entre protocolos de búsqueda visualizando como se comporta cada uno de ellos en términos del número de saltos o longitud de ruta (abscisas) aumentando el número de nodos en la red (ordenadas).

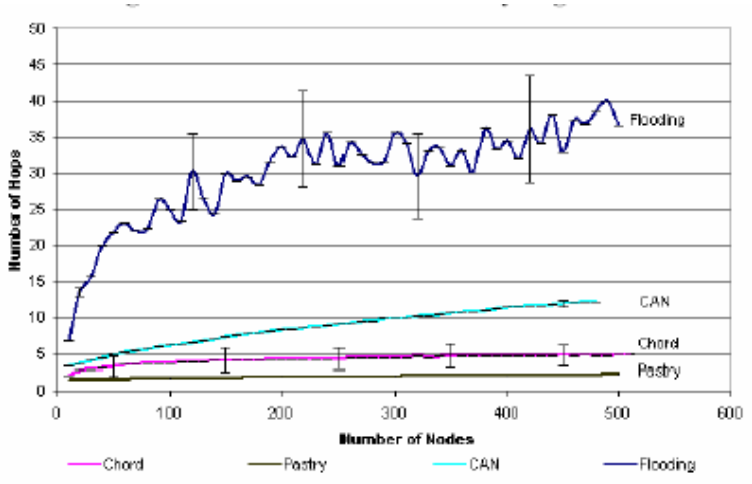


Figura 22. Gráfico Comparativo de Mecanismos de Búsqueda P2P. [8]

De igual forma si se utiliza un ejemplo real de una red p2p donde el número de nodos es$N=2^{16}$ y para el caso de CAN se utilizarían dos dimensiones d=2 y para JXTA número promedio de nodos vecinos de un rendezvous o superpeer es igual a 2 (Ks=2) y el número de nodos rendezvous es 16 (Ns=16), se observa:

	CAN	CHORD	JXTA
Longitud de Ruta	512	16	2^{16} =65536
Tamaño Tabla de enrutamiento	4	16	Variable

2.1.2 Escalabilidad

Como se revisó en las secciones anteriores ambos sistemas tienen como objetivos de diseño, que el sistema sea escalable. Sin embargo, según las complejidades resumidas, se observa que mientras el costo de una búsqueda en CHORD crece logarítmicamente según el número de nodos en la red, para CAN crece exponencialmente según el número de nodos pero aumentado la base, lo que hace de CHORD más escalable que CAN. Sin embargo es de anotar que las tablas de enrutamiento aumentan para CHORD más no para CAN si el número de nodos aumenta. Si se observa JXTA en términos de su escalabilidad, esta es aun más compleja debido a la variabilidad en algunos de sus parámetros (p.e el número de nodos sobre los cuales tiene información un superpeer, no es constante), sin embargo si dejamos este último parámetro como una constante se observa que la escalabilidad del sistema depende del número de nodos y crece exponencialmente según este parámetro. La figura 23 resume estas afirmaciones. El eje de las X representa el parámetro variable para cada uno de los protocolos el cual corresponde para todos como el número de nodos en la red, mientras el eje Y representa el número de saltos.

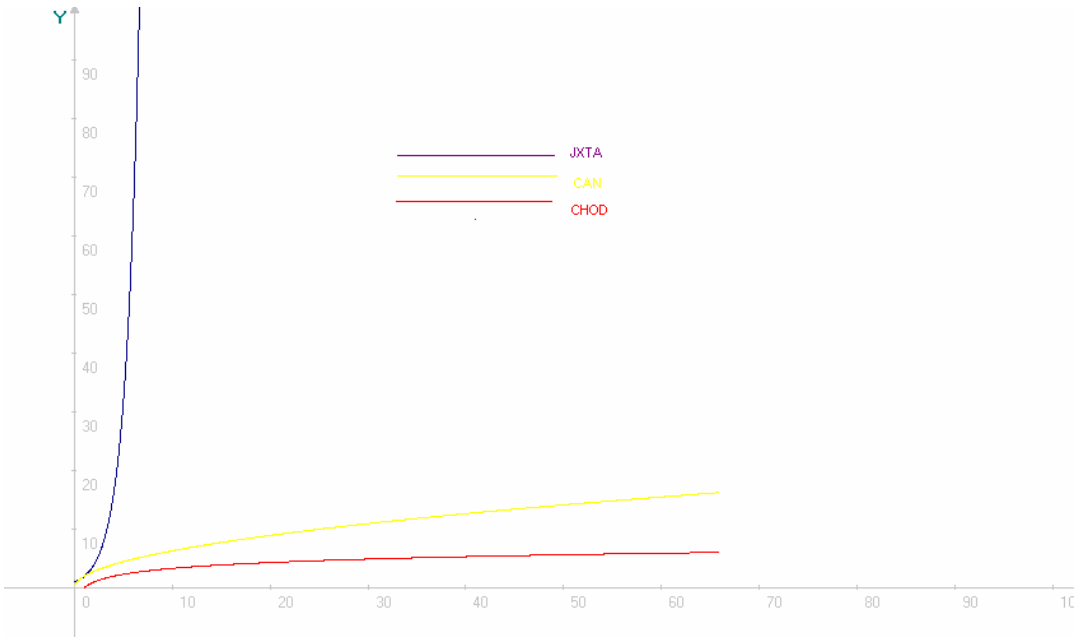


Figura 23. Comparación de Escalabilidad entre CAN, CHORD y JXTA

Como se observa CAN y CHORD presentan mayor escalabilidad que JXTA, y aun así CHORD supera a CAN.

2.1.3 Capacidad de Procesamiento

Según las definiciones de cada uno de los protocolos de búsqueda descritos a lo largo del presente trabajo, se observa que el único protocolo que admite jerarquía de peers es JXTA, mientras que las otras dos aproximaciones basan sus soluciones en una distribución de tareas de forma plana (todos por igual). En JXTA los peers rendezvous y relay apuntan a ser los más robustos en cuanto a capacidad de procesamiento y almacenamiento debido a sus funciones, mientras que para los otros protocolos la asignación de responsabilidades entre la red está dada por "igual" y la basan según una distribución hash, lo que requiere capacidad de procesamiento suficiente para este tipo de tareas. Como se sabe los dispositivos móviles son restringidos en cuando a capacidad de procesamiento, almacenamiento y batería, lo que se convierte en un impedimento si se tiene una red P2P totalmente distribuida formada por dispositivos móviles. Debido a la capacidad limitada de procesamiento para estas máquinas, JXTA se convierte en una aproximación bastante buena pues lo peer rendezvous, podrían

ser laptops con mejor desempeño que los dispositivos móviles (PDA's y teléfonos móviles) los cuales caerían dentro de la categoría de *Minimal Edges Peers*, permitiendo así un balanceo de carga en el sistema.

2.1.4 Propuesta de Implementación

Varios son los ítems a tener en cuenta para tomar una decisión en cuanto a la solución que ayude a resolver este problema:

- La movilidad es uno de los factores más importantes dentro de la aplicación a desarrollar, pero a pesar del gran avance tecnológico en cuanto a dispositivos de este tipo, la capacidad de procesamiento y almacenamiento de estos tiene sus limitaciones

- La movilidad también es sinónimo de dinamismo. Lo que hace que en este tipo de redes la entrada y salida de nodos sea constante.

Con los tres aspectos analizados anteriormente para cada uno de los protocolos estudiados, se tiene que aunque JXTA ofrece una solución apropiada para dispositivos móviles restringidos por su capacidad de procesamiento al presentar una jerarquía de peers, la complejidad en cuanto a la búsqueda es superada por CHORD, así como también la escalabilidad, lo que hace de CHORD el protocolo seleccionado como una solución al problema de búsqueda distribuída.

De la selección de CHORD como protocolo eficiente de búsqueda distribuida y de la características de los dispositivos móviles de proponen dos prototipos para analizar el comportamiento de una red P2P a nivel:

1. La primera solución consiste de una red P2P pura donde conformadas por dispositivos móviles que implementan el protocolo CHORD en su totalidad.

2. El segundo prototipo por el contrario es una red parcialmente centralizada, donde los "supernodos", dispositivos cuya capacidad de procesamiento supera la de un teléfono móvil (computadores alámbricos e inalámbricos, mainframes , etc) , implementan el protocolo CHORD. Mientras que los teléfonos móviles se conectan vía http a los formadores del anillo.

La primera propuesta se caracteriza por ser totalmente distribuída, heredando todas la ventaja de un red P2P pura como lo son la escalabilidad y la alta disponibilidad, sin embargo y debido a la intermitencia de los dispositivos móviles debido a las condiciones en las cuales esta trabaja (altas intermitencias en la red, duración de la batería, entre otros) hacen constantes la salida y entrada de nodos de la red, implicando un trabajo mas duro en el proceso de mantener la red (proceso de estabilización, inserción y salida de nodos), tarea para la que los dispositivos móviles no están en capacidad de realizar debido a sus restricciones.

De otro lado, la segunda propuesta representa una solución acorde a las condiciones de un dispositivo móvil descritas anteriormente, pero sacrificando las fortalezas de funcionar en una arquitectura P2P pura. Para ello se utiliza el concepto de jerarquía de peers de dos niveles

- Superpeers: nodos o peers robustos en cuanto a procesamiento y almacenamiento, estables en la red en cuanto a entrada y salida en la comunidad peer. Por esta razón realizan las tareas fuertes de una comunidad P2P como lo es son el enrutamiento de mensajes y la búsqueda.
- Peer Livianos, debido a sus limitaciones en hardware cuya función principal es solicitar servicios dentro de la comunidad P2P.

Con la anterior estructura los superpeers implementan el protocolo CHORD y los peer livianos, que para nuestro caso son dispositivos móviles, no realizan tareas duras, por el contrario a través de una petición se puede acceder a los servicios de compartir y buscar archivos en la red a través de los superpeers.

2.2 SEGURIDAD

De otro lado dentro del contexto de aplicaciones móviles P2P en el cual el objeto de esta tesis se encuentra, la seguridad, tema que requiere de análisis, de ahí la razón de las secciones anteriores.

En sistemas cliente/servidor los procesos de autenticación y autorización son manejados a través de una entidad central, sin embargo, y debido a la naturaleza descentralizada de sistemas P2P, esta noción cambia completamente y es cada peer actuando como servidor en ciertos casos quien debe realizar dichas funciones.

Muchas de las aplicaciones P2P adhieren a sus sistemas de seguridad algunas de las aproximaciones descritas en la sección 2.3 para el diseño de sistemas seguros, sin embargo, por simplicidad y debido a la naturaleza de los dispositivos móviles la solución planteada en esta sección en cuanto a seguridad es simple.

La aplicación maneja la seguridad desde las siguientes perspectivas:

- Autenticación.

  Según lo observado en la sección 2.3 muchas de las aproximaciones descritas, son usadas para la autenticación (encripción, firmas digitales y certificados). Sin embargo para nuestro sistema, el problema de autenticación se va a manejar a través de certificados firmados por una entidad conocida. Cada peer debe presentar su certificado para poder tener interacción dentro del grupo peer. Generalmente en los sistemas P2P hay una fase de inicialización o bootstrap, fase en la cual el usuario necesita conocer alguno de los nodos en el grupo para poder unirse, en esta fase es cuando el nodo que desea ingresar a la comunidad P2P debe mostrar el Certificado acreditando ser confiable, el nodo que esta estableciendo la entrada a la red para el nuevo nodo debe verificar la autenticidad

del usuario, una vez se haya realizado la verificación de manera exitosa, el nodo se une a la red, de lo contrario el nodo no es admitido.

- Integridad.

La encripción es la metodología más usada para el manejo de integridad de datos. Sin embargo como se observó dos tipos de encripción existen, la asimétrica y la simétrica, cada una tiene ventajas sobre la otra. La encripción simétrica es mas rápida en términos de procesamiento que la simétrica, mientras que la encripción asimétrica es mas robusta en cuanto a la seguridad en si, que los algoritmos de encripción simétrica.

Debido a que los dispositivos móviles son más limitados en cuanto a recursos computacionales, la encripción asimétrica podría comprometer el desempeño de la aplicación. Aunque la encripción simétrica compromete algo de seguridad debido a una única llave compartida, el procesamiento rápido de los algoritmos de este tipo, convierten este tipo de encripción como la mejor opción para manejar la integridad entre peers móviles. En nuestras soluciones el intercambio básico de mensajes, así como el intercambio de archivos se realizará utilizando algoritmos de encripción simétrica. La llave compartida que se usa dentro del grupo se asigna en el momento del ingreso de un nodo una vez se ha verificado el certificado.

3 IMPLEMENTACION

3.1 PROTOTIPO 1

El prototipo 1 como se describió en la sección anterior basa su razón de ser en una arquitectura P2P totalmente distribuida, la figura 24 muestra gráficamente como se encuentran distribuidos lógicamente los peers armando un anillo CHORD, según las especificaciones de dicho protocolo. En esta primera aproximación los peers que intervienen en el anillo son dispositivos móviles.

Figura 24. Distribución Lógica de Peers Prototipo 1

Para esta primera solución se usó J2ME Wireless ToolKit de Sun como herramienta de sedarrollo. De igual forma se usaron las siguientes librerías:

- JCHORD [25]: Implementación en J2SE del protocolo de CHORD, la cual se adaptó a J2ME teniendo en cuenta que este último es un subconjunto de J2SE. Para poder adaptar esta librería al ambiente de desarrollo en J2ME se hicieron cambios en el manejo de *arrays*, serialización de objetos y el uso de la clase *BigInteger* .

- Para la serialización de objetos de igual forma se basó en la librería Burlap y se hizo el desarrollo para la serialización de los objetos de la aplicación.

- Para la implementación de la clase *BigInteger se utilizó la librería BouncyCastle* que ya lo tiene implementado.

Para este primer prototipo, cada peer, que para este caso son dispositivos móviles, implementa cada una de las funciones del API CHORD. Es decir, para este caso, cada dispositivo es capaz de participar en la inserción y búsqueda de una llave en la red, que para nuestro caso corresponde a un archivo, así como también, está constantemente, a través del proceso de estabilización, verificando los estados de los nodos con los cuales tiene relación (sucesor y predecesor), para mantener la integridad del anillo.

3.2 PROTOTIPO 2

La figura 25 muestra la distribución lógica de los peers para el prototipo 2. Para este prototipo los peer se ubican lógicamente formando una arquitectura P2P hibrida, donde los súper peers forman un anillo CHORD, mientras que los dispositivos móviles se conectan a la comunidad a través de uno de los súper nodos para obtener acceso a los servicios ofrecidos por el API CHORD,

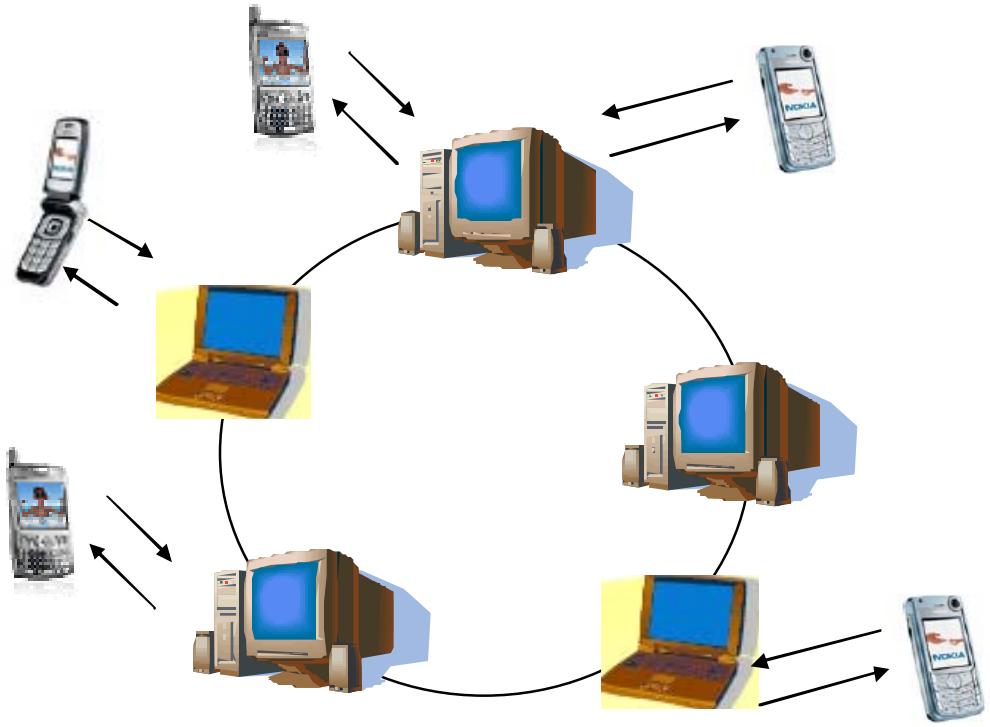


Figura 25. Distribución Lógica de Peers Prototipo 2

Para la implementación de esta solución se utilizó JXTA como herramienta facilitadora en la construcción de aplicaciones P2P, y aunque algunos de los protocolos ofrecidos por esta plataforma fueron utilizados, la inserción y búsqueda de archivos en la red son ofrecidas a través de la implementación por cada uno de los súper peers del protocolo CHORD. JXME por su parte, se utilizó para el desarrollo de la comunicación del dispositivo móvil con la red JXTA, utilizando mensajes http que los súper peers entienden debido a su configuración como proxys JXME.

3.3 PRUEBAS

Por restricciones de tiempo y de recursos las pruebas realizadas sobre los dos prototipos fueron hechas a través de los simuladores para móviles y/o PDAs ofrecidos por la herramienta J2ME Wireless Toolkit y con redes conformadas por un máximo de

tres nodos. Los simuladores usan los recursos del PC's tanto de CPU y memoria así como también de red para simular un móvil en un ambiente MIDP

Los tres peers usados para las pruebas presentan la siguiente configuración:

- PEER1

  Procesador Intel Pentium III 501 MHz, 128 MB de RAM

- PEER2

  Procesador Intel Celeron 1.10 GHz, 240 MB de RAM

- PEER3

  Procesador Pentium M 1.4 GHz, 512 MB de RAM

3.3.1 Consumo de Memoria

Para monitorear el consumo de memoria de la aplicación se utilizó el monitor de memoria que viene con la suite de J2ME Wirelles Toolkit. Durante las pruebas básicas de los casos de uso de la aplicación se monitorearon las dos soluciones en cuanto al uso de memoria, obteniendo lo siguiente:

Carga de la Aplicación Las figuras 26 y 27 presentan gráficamente el consumo de memoria que cada una de las aplicaciones requiere al cargar los aplicativos. La diferencia no es altamente significativa entre las dos soluciones, observando que el prototipo 2 consume un poco menos.

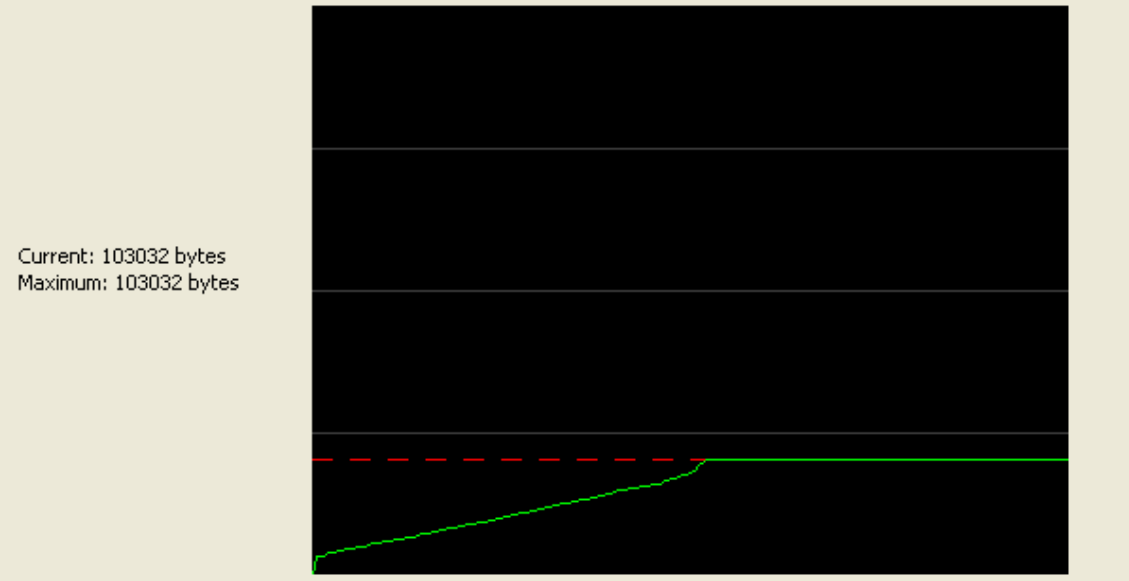


Figura 26. Carga de la Aplicación Prototipo 1.

Figura 27. Carga de la Aplicación Prototipo 2

Flujo Normal de la Aplicación Además la ejecución de cada uno de los procedimientos del API CHORD, cada peer implementando dicho protocolo ejecuta

constantemente el proceso de estabilización, el cual garantiza la configuración correcta de cada peer en el anillo y permite el éxito de las otras operaciones CHORD. Al observar el comportamiento de la ejecución de este proceso en un móvil del prototipo 1 se observa que el consumo de memoria aumenta hasta llegar aproximadamente a los 165 Kb según la capacidad del dispositivo, y aun más se aumenta el consumo de memoria si a la par del proceso de estabilización se ejecuta otra solicitud tal como compartir o buscar. Las figuras 28 y 29 reflejan el comportamiento de estos procesos en el peer etiquetado como peer 2.

Para el prototipo 2, el proceso de estabilización, que como se observa en el prototipo 1 es costoso, no es ejecutado por los móviles, por el contrario es ejecutado por los superpeers, lo que hace que el comportamiento de la memoria sea casi constante en los móviles que utilizan el prototipo 2. Y se traduce en un ahorro del 57.5% promedio en el consumo de de memoria. La figura 30, representa el comportamiento de consumo de memoria para el peer etiquetado como peer 2.

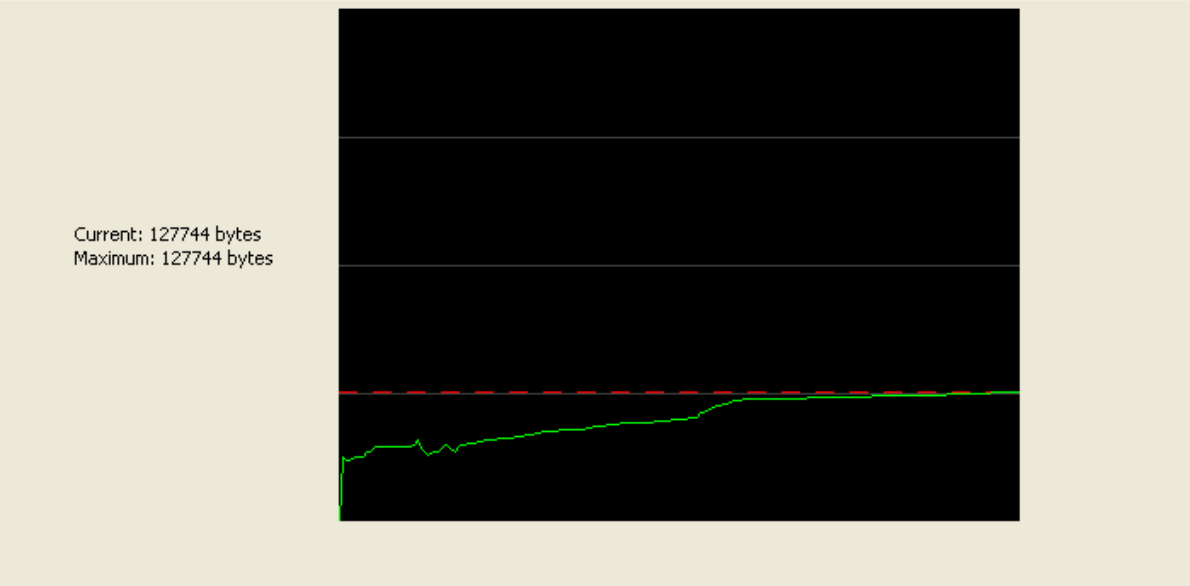


Figura 28. Proceso de Estabilización CHORD. Prototipo 1

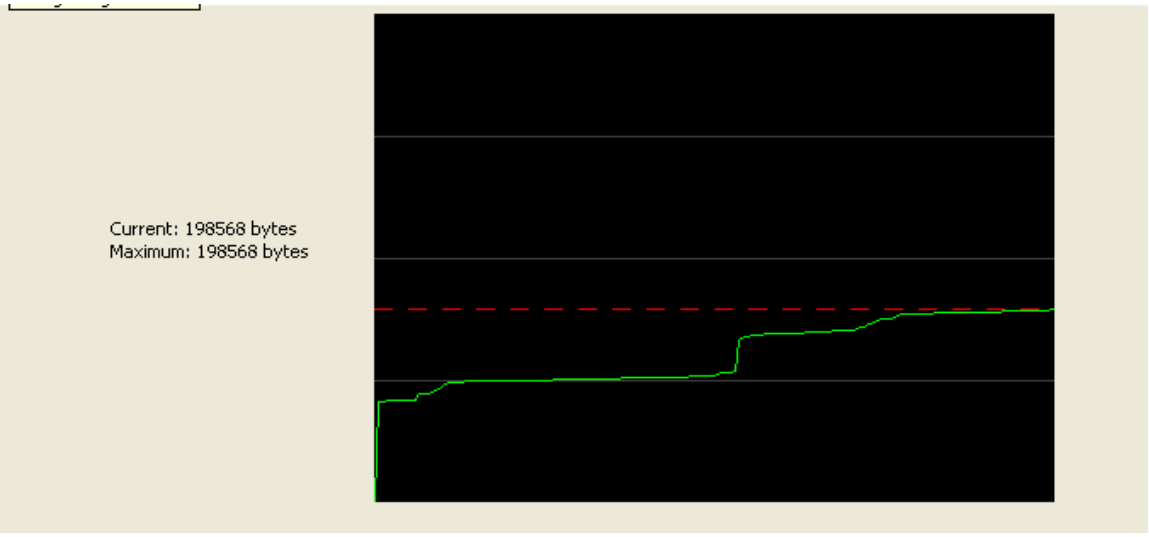


Figura 29. Proceso de Estabilización CHORD + Función del API de CHORD. Prototipo 1

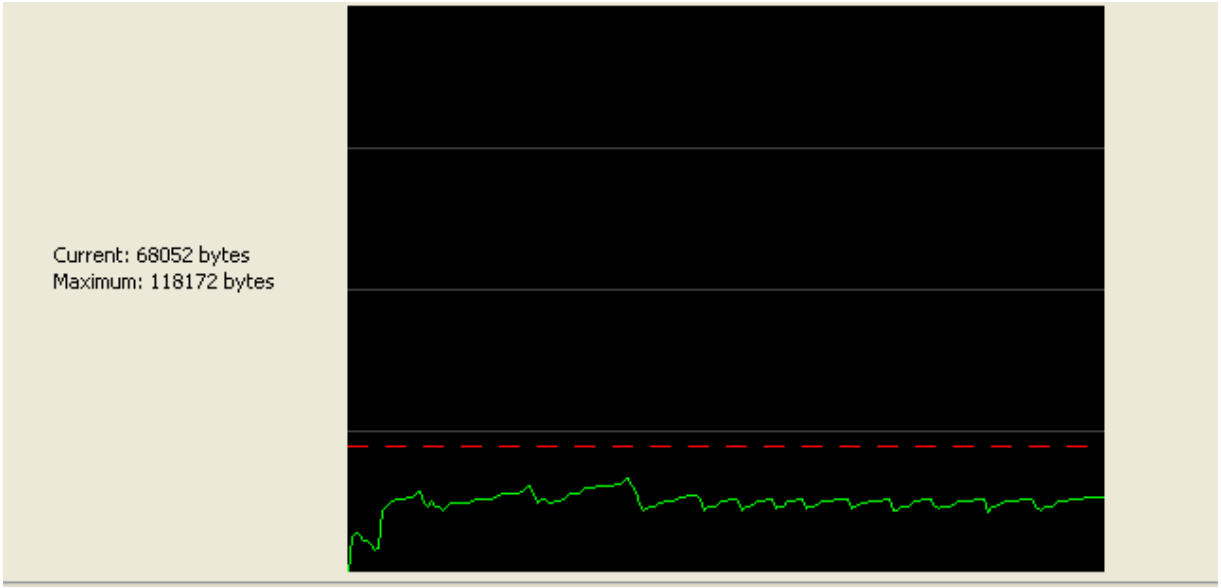


Figura 30. Procesamiento Prototipo 2

3.3.2 Tiempos de Procesamiento

La tabla 4, resume los tiempo medidos[3] para cada una de la funciones del API de CHORD ejecutándose desde el prototipo1, variando el número de nodos en la red.

	1 Nodo	2 Nodos	3 Nodos
Connect	0.2	6	7
Stabilize	33.8	209	266.9
Insert	525.4	998.5	1087.5
Lookup	285.8	280.7	291.7
Leave	261.7	272.1	251.7

Tabla 4. Tiempos de Procesamiento Prototipo1

La tabla 5 por su parte resume los tiempo medidos para cada una de la funciones del API de CHORD implementada del el móvil en el prototipo 2, variando el número de nodos en la red.

	1 Nodo	2 Nodos	3 Nodos
Connect	11	10	7
Insert	2.1	2.2	1.9
Lookup	1.8	1.6	2.1
Leave	3.6	2.5	2.6

Tabla 5. Tiempos de Procesamiento Prototipo2

Al comparar las tablas 4 y 5 se nota considerablemente la diferencia en los tiempos de procesamiento entre los dos esquemas. La diferencia marcada entre los tiempos de procesamiento de los dos prototipos para cada una de las funciones del API CHORD, se debe básicamente a la ventaja en cuanto a especificaciones de hardware (memoria y CPU) que los súper peers tienen, al compararlos con las capacidades que poseen los dispositivos móviles.

[3] Los tiempos medidos en este trabajo están dados en segundos

3.3.3 Seguridad

Como se muestra en la tabla 6, fue casi imposible probar la seguridad en el prototipo 1 pues los tiempos aumentan considerablemente si se comparan los resultados con los plasmados en la tabla 4. Este comportamiento dificulta las pruebas con más de un nodo en la red.

	1 Nodo	**2 Nodos**	**3 Nodos**
Connect	0.1		
Stabilize	705.7		
Insert	43218		
Lookup	41467		
Leave	21757		

Tabla 6. Tiempos de Procesamiento en Seguridad Prototipo2

La tabla 7 por su parte resume los tiempo medidos para cada una de la funciones del API de CHORD implementadas desde el móvil en el prototipo 2, usando seguridad no varían si se compara su comportamiento con los presentados en la tabla 5 cuando el prototipo se esta ejecutando sin seguridad.

	1 Nodo	2 Nodos	3 Nodos
Connect	10	10	9
Insert	2.3	2.5	2.1
Lookup	1.9	1.7	2.3
Leave	3.5	3.2	2.9

Tabla 7. Tiempos de Procesamiento en Seguridad Prototipo2

Los resultados en cuanto a los tiempos de procesamiento añadiendo seguridad a las soluciones muestra cómo la solución etiquetada como prototipo 2 supera a la plasmada en el prototipo 1. Lo anterior se debe nuevamente a la intervención de los súper peers en el proceso de encripción y desencripción en el prototipo 2, mientras que para el prototipo 1, estos dos procesos son realizados por dispositivos con mucha menos capacidad de memoria y procesamiento. Actualmente computadores de escritorios y portátiles superan a lo dispositivos moviles en mas de 500 veces su memoria RAM, además de tener procesadores cada vez mas poderosos.

La diferencia entre los dos procesos se debe de igual forma a las restricciones que presenta la plataforma J2ME consecuente con las restricciones en el ambiente móvil, pues la serialización de objetos no está implementada con J2ME y la transmisión de archivos a través de sockets se debe hacer byte a byte, hacer este proceso adicionándole seguridad implica encriptar y desencriptar cada byte transmitido por los dispositivos moviles que participan en la solución del prototipo 1. Para la segunda propuesta no sucede de la misma forma pues el intercambio de mensajes que tiene implementado JXTA y JXME, hace que el envío de ellos no requiera implementación por parte del desarrollador, por el contrario a pesar que el archivo se envía encriptado desde el móvil, la recepción del mensaje por parte del súper peer es desencriptado de manera rápida y eficiente, una vez este recibe la petición, envía los mensajes necesarios según el protocolo CHORD a los demás peers en la red.

3.4 CONSIDERACIONES ADICIONALES

En esta sección se resumirán muchas de las experiencias adquiridas durante el trabajo de implementación y que pueden ayudar a futuras trabajos.

Para el desarrollo del prototipo 1 como ya se había descrito se utilizó J2ME Wireless Toolkit como herramienta de desarrollo y pruebas, utilizando cada una de las utilidades que esta ofrece tales como la creación de proyectos, la generación de comprimidos jar, los simuladores y el monitor de memoria. A pesar de las facilidades enunciadas ofrecidas por esta suite, muchas son las limitaciones que en el ambiente de desarrollo se presentan, pues este subconjunto de J2SE no posee muchas de las clases

acostumbradas a usar desde J2SE. Por citar algunos ejemplos, no tiene la clase BigInteger, el uso de diversas implementaciones de listas y arrays se resume en el uso de vectores, la serialización de objetos no existe en la implementación J2ME para la especificación CLDC. A todo esto se suma las restricciones en cuanto al tamaño del vector o array que se puede manejar.

Para el prototipo 2 se utilizó JXTA y JXME. JXTA compuesto de una serie de protocolos que facilitan la interacción de peers en ambientes P2P, presenta alta complejidad en el funcionamiento de algunos de sus protocolos e implementación de ellos, convirtiéndose esto en una de sus mayores desventajas. De otro lado JXME por encontrarse aun en etapa de desarrollo presenta limitaciones en cuanto a documentación, tutoriales, y aplicaciones de ejemplos lo que hace difícil su adopción, aun más si se requiere de implementaciones complicada. Problemas de comunicación entre peers JXME y JXTA se encontraron durante la etapa de implementación debido a la incompatibilidad de versiones JXME y JXTA, lo que me parece una debilidad de las dos tecnologías.

,

4 CONCLUSIONES Y TRABAJOS FUTUROS

4.1 CONCLUSIONES

Son muchos los aspectos a tener en cuenta en el desarrollo de aplicaciones P2P, durante el desarrollo de esta tesis algunos de ellos fueron aprehendidos, entre ellos el de la búsqueda distribuida, la cual no resulta tan sencilla como ir a buscar a un sitio central. El estudio de muchas soluciones a este aspecto, seleccionaron la mas eficiente por su comportamiento eficiente para traslaparlo a un ambiente móvil y observar su funcionamiento. Las dos soluciones planteadas muestran como la solución planteada no es del todo viable para un ambiente P2P móvil puro. Los resultados en cuanto a tiempo y memoria presentados en el capitulo anterior así lo reflejan. Sin embargo una variable a analizar minuciosamente dentro del API de CHORD es el proceso de estabilización, pues en el seguimiento que se hizo del prototipo 1, el consumo de memoria y procesamiento es alto y más aun si se combina con otras de las operaciones del API.

La solución propuesta con el prototipo 2 es viable según las condiciones de hardware planteadas actualmente para los dispositivos móviles, sin embargo con el acelerado avance tecnológico que día a día tienen este tipo de dispositivos, la solución propuesta por el prototipo 1 podría ser viable, pues una ventaja significativa de esta solución es el tema de costos, pues no se haría inversión en máquinas súper poderosas que soporten el procesamiento de la red, sino por el contrario son los móviles los que harían dicha tarea, además de garantizar disponibilidad y escalabilidad total que ofrecen las redes P2P puras. Con las condiciones actuales de hardware para los dispositivos móviles y la inestabilidad de las redes móviles, y los tiempos arrojados en las pruebas para el prototipo 1, esta solución no sería la más adecuada, su puesta en producción requiere de algunos ajustes tanto a nivel del protocolo como tal así como de implementcación.

De otro lado al combinar cada uno de los prototipos con la solución de seguridad planteada se observa nuevamente como el prototipo 2 supera al prototipo 1, agregando también que el prototipo 1 con seguridad, se convierte en una solución

poco viable pues los tiempos son muy altos, y teniendo en cuenta la inestabilidad de una red móvil, con esos tiempos el éxito de una transacción sería poco probable. La solución de seguridad planteada en este trabajo, aunque muy simple, se presentó como una solución liviana teniendo en cuenta la naturaleza de los dispositivos móviles y la naturaleza de la aplicación, pues su razón de ser no es una aplicación corporativa, militar o de e-bussiness, que necesitan un nivel de seguridad mucho más exigente. Aún así, el rendimiento no fue el esperado para el primer prototipo, por lo que se propone dejar este tema de estudio a trabajos futuros en el área ya que con el método tradicional de encripción no se lograron los resultados esperados.

4.2 TRABAJOS FUTUROS

Las pruebas realizadas de los prototipos fueron realizadas sobre simuladores de dispositivos móviles, no obstante pruebas sobre teléfonos reales permitiría afinar algunas funciones además de obtener mediciones más reales del comportamiento de cada uno de los prototipos.

Como se observó en el capitulo de implementación y pruebas, el consumo de memoria en el proceso de estabilización para el prototipo 1 es alto, por lo que sería interesante analizar dicho proceso y procurar mejorarlo para hacer viable dicha solución. Algunos temas de implementación podrían ser mejorados, más aun si la capacidad de procesamiento de los dispositivos móviles aumenta.

Una aplicación P2P como producto final del trabajo de tesis fue uno de los logros más relevantes, sin embargo procesos de replicación e indexamiento automático no fueron incorporadas en el diseño de la aplicación, aspectos que pueden ser relevantes para próximos estudios.

La aplicación construida puede ser vista como un conjunto de servicios que sirven como base a otro tipo de aplicaciones tales como instant message, juegos, aplicaciones de colaboración, entre otras, las cuales podrían ser construidas y exploradas

ANEXO A. VISTA DE LA APLICACION

La aplicación se ejecuta con los simuladores de la suite J2ME Wireless ToolKit de Sun. En el cargue inicial de la aplicación se presenta un menú con las opciones mostradas en la figura 31.

Figura 31. Menú Principal

Cuando se selecciona la opción Conectarse, la aplicación realiza las operaciones necesarias para conectarse al anillo, haciendo llamado a los procedimientos create () o

join(idNodo) del API Chord. El metodo create se llama cuando es el primer nodo en el anillo y el join cuando ya existe por lo menos un nodo formando el anillo. La figura 32 presenta la pantalla indicando que ya se encuentra conectado al anillo.

Figura 32. Conexión al anillo

Una vez se encuentra el nodo conectado a la red, se pueden comenzar a hacer las tareas de Compartir y Buscar. Para la acción Compartir, el aplicativo lee del sistema de archivos del teléfono móvil, para que el usuario pueda seleccionar de manera fácil y rápida el archivo que desea compartir en la red. Para realizar esta tarea se usó el API FileConnection de J2ME que permite acceder al sistema de archivos del teléfono. Ver figura 33. Luego que se ha seleccionado el archivo, la aplicación solicita ingresar las

palabras claves con las que luego se buscaría el archivo en la red, y una vez el usuario confirma , se hace el llamado al procedimiento InsertarValorKey del API Chord, informándole al usuario en que nodo fue guardado el archivo (Figura 34).

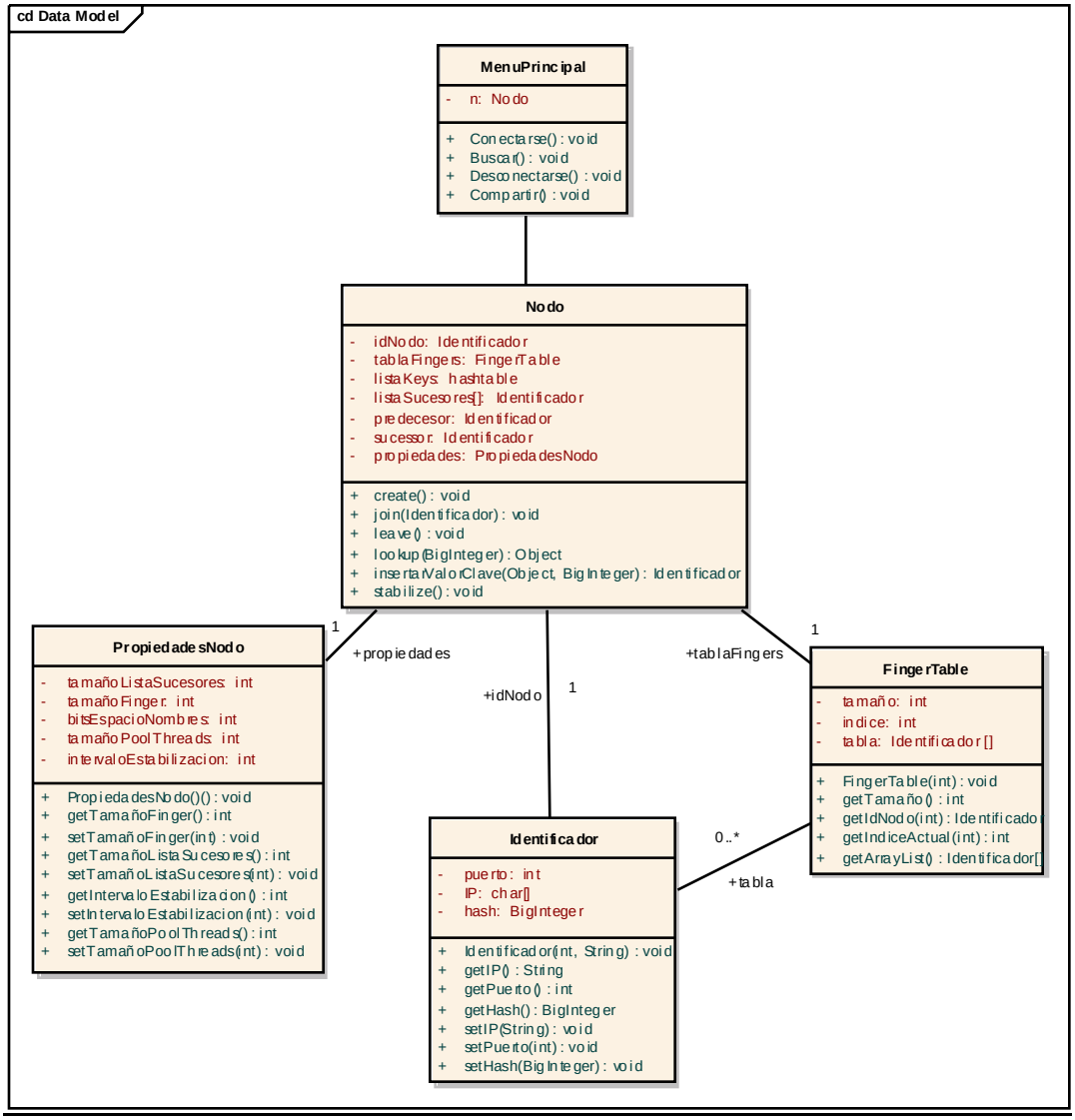
Figura 33. Sistema de Archivos del Teléfono

Figura 34. Nodo que Almacena el Archivo

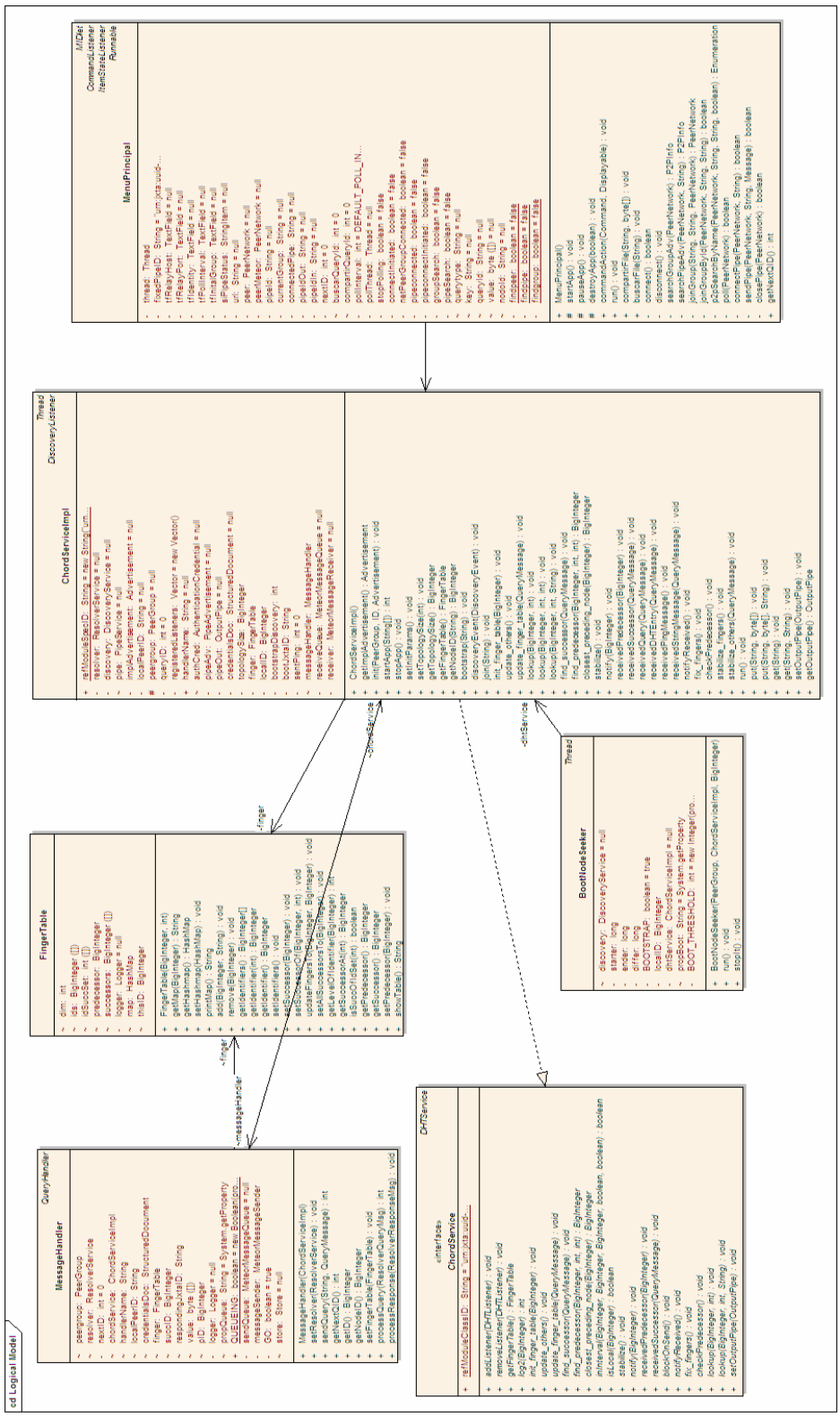
Para la acción Buscar, el aplicativo solicita ingresar las palabras claves con las que luego se buscaría el archivo en la red, y una vez el usuario confirma, se hace el llamado al procedimiento lookup del API Chord, devolviéndole al usuario el archivo y permitiendo guardarlo en el sistema de archivos del teléfono.

Para la Desconexión del anillo el aplicativo llama al procedimiento leave() del API Chord.

ANEXO B. DIAGRAMA DE CLASES PROTOTIPO 1



ANEXO C. DIAGRAMA DE CLASES PROTOTIPO



BIBLIOGRAFÍA

[1]. Dabek, Frank et al. Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service. MIT Laboratory for Computer Science. [online]. http:// pdos.csail.mit.edu/papers/chord:hotos01/hotos8.pdf

[2]. Stoica, Ion et al. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications". MIT Laboratory for Computer Science. JAnuary 10, 2002. [online] http:// pdos.csail.mit.edu/papers/ chord:sigcomm01/chord_sigcomm.pdf

[3]. Patricia Gonzales; Marcelo Zuñiga. Redes P2P y JXTA. Universidad Técnica Federico Santa María. Chile 2005. [online] http://www.inf.utfsm.cl/~rmonge/sd/trabajos/trejo-zu%F1iga-p2p.pdf

[4]. Ratnasamy, Sylvia et al. A scalable content-addressable network. In Proceedings of SIGCOMM 2001, August 2001. [online] http://www.eecs.harvard.edu/~mema/ courses/cs264/papers/p13-ratnasamy.pdf

[5]. Gong, Li . JXTA: A Network Programming Environment. Sun Microsystems, Inc. 2001. [online] http://www.jxta.org/project/www/docs/JXTAnetworkProgEnv.pdf

[6]. JXTA v2.3.x: Java Programmer's Guide. 2005 Sun Microsystems. [online] http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf

[7]. Gnutella vs. JXTA. CM316: Multimedia Systems Coursework, ECS. [online] http://mms.ecs.soton.ac.uk/mms2002/papers/17.pdf

[8]. M. Kelaskar, V. Matossian, P. Mehra, D. Paul, and M. Parashar. A Study of Discovery Mechanisms for Peer-to-Peer Applications. Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ. [online] http://www.caip.rutgers.edu/TASSL/Papers/p2p-p2pws02-discovery.pdf

[9]. Burkard, Timo. Herodotus: A Peer-to-Peer Web Archival System. MASSACHUSETTS INSTITUTE OF TECHNOLOGY. May 2002. [online] http://pdos.csail.mit.edu/papers/chord:tburkard-meng.pdf

[10] Milojicic, Dejan S et al. Peer-to-Peer Computing. HP Laboratories Palo Alto .July 3rd , 2003. [online] http://www.hpl.hp.com/techreports/2002/HPL-2002-57R1.pdf

[11] Setia, Sanjeev . Distributed Hash Tables (DHTs) Chord & CAN. [online] http://cs.gmu.edu/~setia/cs699/lecture3.pdf

[12] Kowalski, Stewart;Nick Edwards. A Security and Trust Framework for Wireless World: A Cross Issue Approach. Wireless World Research Forum (WWRF). [online] http://www.ambient-networks.org/docs/A_Security_and_Trust_Framework_for_a_Wireless_World_%20A_Cross_Issue_Approach.pdf

[13] Krishnan, Navaneeth. JXTA and Security. [online] http://java.sun.com/developer/Books/networking/jxta/jxtap2pch08.pdf

[14]. Coulouris, George et al. Sistemas Distribuidos: Conceptos y Diseños. Addison Wesley, 2001

[15]. Network Associates, Inc. and its Affiliated Companies. An Introduction to Cryptography. 1990-1999. [online] http://www.cryptorights.org/keys/info/pgp-docs/PGP_Intro_to_Crypto.PDF

[16] Cahill, Vinny .et all. Using Trust for Secure Collaboration in Uncertain Environments. April 3, 2003. [online] http://www.cis.strath.ac.uk/research/publications/papers/strath_cis_publication_262.pdf

[17]. Singh, Aameek; Liu, Ling. TrustMe: Anonymous Management of Trust Relationships in Decentralized P2P Systems. [online] http://www.cc.gatech.edu/~aameek/publications/trustme-p2p03.pdf

[18]. Rita Chen, William Yeager. Poblano, A Distributed Trust Model for Peer-to-Peer Networks, Sun Microsystems

[19]. Fenkam, Pascal et al. Towards an Access Control System for Mobile Peer-to-Peer Collaborative Environments. Technical University of Vienna, Distributed Systems Group Argentinierstrasse 8/184-1, A-1040 Vienna, Austria. [online] http://www.infosys.tuwien.ac.at/.../sd/papers/ TowardsAnAccessControlSystemForMobileP2PCollaborativeEnvironments.pdf

[20]. Berket, Karlo; Essiari, Abdelilah; Muratas, Artur. PKI-Based Security for Peer-to-Peer Information Sharing. [online] http://citeseer.ist.psu.edu/cache/papers/cs2/516/http:zSzzSzfemto.orgzSzp2p2004zSz paperszSzberket.pdf/berket04pkibased.pdf

[21]. *Wikipedia, La enciclopedia libre*, [web site] http://es.wikipedia.org/wiki/P2P

[22]. Werbach, Kevin. Wireless Peer to Peer. November, 2000. [online] http://www.thefeature.com/article?articleid=7971

[23]. The eBusiness Centre. A Survey of Peer-to-Peer File Sharing Technologies. Athens University of Economics and Business. 2002. [online] http://www.mimuw.edu.pl/~alx/ask/androutsellis-theoto02survey.pdf

[24]. Roman Kurmanowytsch, Engin Kirda, Clemens Kerer and Schahram Dustdar. OMNIX: A topology-independent P2P middleware. Technical University of Vienna, Distributed Systems Group. [online] http://www.infosys.tuwien.ac.at/Staff/ck/files/umics2003.pdf

[25]. The Jcord Library. [online] https://jchord.dev.java.net/

[26] .Burlap Serialization. [online] http://www.caucho.com/resin-3.0/protocols/burlap.xtp

[27]. Bouncy Castle The lightweight API. [online]http://www.bouncycastle.org/