UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

ÁREA TÉCNICA

TITULACIÓN DE INGENIERO EN SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Implementación de un protocolo de enrutamiento Ad-Hoc

en dispositivos móviles

TRABAJO DE FIN DE TITULACIÓN

AUTORES: Medina Jiménez, Astrid Carolina

Macas Romero, Rommel Landivar

DIRECTOR: Torres Tandazo, Rommel Vicente, PHD

LOJA-ECUADOR

2013

CERTIFICACIÓN

PHD.

Rommel Torres Tandazo

DIRECTOR DEL TRABAJO DE FIN DE TITULACIÓN

C E R T I F I C A:

Que el presente trabajo, denominado: “Implementación de un protocolo de enrutamiento ad-hoc en dispositivos móviles" realizado por los profesionales en formación: Medina Jiménez Astrid Carolina y Macas Romero Rommel Landivar; cumple con los requisitos establecidos en las normas generales para la Graduación en la Universidad Técnica Particular de Loja, tanto en el aspecto de forma como de contenido, por lo cual me permito autorizar su presentación para los fines pertinentes.

Loja, Noviembre de 2013

f)……..

DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS

“Nosotros Medina Jiménez Astrid Carolina y Macas Romero Rommel Landivar declaramos ser autores del presente trabajo y eximimos expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones legales.

Adicionalmente declaramos conocer y aceptar la disposición del Art. 67 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice: “Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado que se realicen a través, o con el apoyo financiero, académico o institucional (operativo) de la Universidad”

f).....

Autor: Medina Jiménez Astrid Carolina

Cédula 1104603723

f).....

Autor: Macas Romero Rommel Landivar

Cédula 1103984579

DEDICATORIA

Dedico este trabajo principalmente a Dios, por permitirme haber llegado a este momento tan importante de mi formación profesional. A mis padres Luis y Sandra, a mi hermana Karla que son el pilar de mi vida, a mi familia por demostrarme su cariño y apoyo incondicional. A mis amigos que han estado en todo momento a mi lado y a todas las personas que de una u otra forma me han acompañado en el transcurso de mi carrera.

Astrid

El presente trabajo va dedicado a mis padres como muestra de gratitud por el gran esfuerzo de vida para conmigo, por la guía y el amor que me han brindado, a mis hermanos, porque cada uno a su estilo, supieron motivarme a conseguir este logro, a mi esposa por ese inmenso cariño y comprensión que me ha compartido, a mis dos hijos Peyton y Alejandro que son mi razón de ser y seguir adelante. Sin olvidar a mis amigos, a ese grupo de personas con las que comparto ideales y los cuales puedo apoyarme cuando los obstáculos de la vida se hacen presentes.

Rommel

AGRADECIMIENTO

Queremos expresar nuestro más sincero agradecimiento a Dios y a la Virgen del Cisne por permitirnos terminar este proyecto con éxito.

A nuestras familias, que han sido pilares fundamentales de nuestras vidas, y que con su apoyo hemos podido dar este paso tan importante en nuestras vidas profesionales.

A nuestro Director de Tesis, Dr. Rommel Torres por su guía en este trabajo de fin de carrera, así como a los docentes de nuestra Universidad, que nos impartieron su conocimiento para formarnos en el camino a la profesionalización.

A nuestros amigos, quienes con su amistad incondicional durante todo este tiempo aportaron para poder llegar a obtener una meta más planteada en nuestras vidas.

Y a cada una de las personas que pusieron un granito de arena y colaboraron para la consecución de nuestro objetivo.

Astrid y Rommel

ÍNDICE DE CONTENIDOS

CAPÍTULO III

CAPÍTULO IV

CAPÍTULO V

RESUMEN EJECUTIVO

Las redes móviles Ad-Hoc son aquellas que no necesitan un punto de acceso para la comunicación, ya que son temporales y auto configurables (Torres, 2011). Son un gran ejemplo de la forma en cómo la comunicación ha cambiado y brinda nuevas alternativas de mejora en la transmisión de cualquier tipo de información. Los protocolos, algoritmos o modelos deben ser diseñados de forma eficiente para que puedan aprovechar las capacidades de los dispositivos móviles, es por ello que el presente trabajo analiza las características del protocolo de enrutamiento BCHP (Backup Cluster Head Protocol) en redes Ad-Hoc, e investiga su implementación como una solución de comunicación en dispositivos móviles.

Los sistemas operativos seleccionados para realizar la implementación de BCHP son: Ubuntu y Android, que por sus características son los más adecuados para realizar este proyecto. La transmisión de datos se realiza entre tres dispositivos, sin la necesitad de un punto de acceso para que los nodos intercambien información entre sí, tomando en cuenta las características del protocolo a implementar.

Palabras clave: Ad-hoc, BCHP, Ubuntu, Sistema Operativo, Protocolo de Enrutamiento.

ABSTRACT

Ad- Hoc mobile networks are those that don't need an access point for communication, as they are temporary and self configurable (Torres, 2011). They are a great example of the way how communication has changed and provide new alternatives for improving the transmission of any information. The protocols, algorithms or models should be designed efficiently so they can leverage the capabilities of mobile devices, which means this work analyzes the characteristics of BCHP routing protocol (Backup Cluster Head Protocol) in Ad-Hoc networks, and investigates their implementation as a communication solution for mobile devices.

The selected operating systems for implementing BCHP are: Ubuntu and Android, which by their characteristics are best suited for this project. The data transmission is between three devices, without an access point for nodes to exchange information among themselves, taking into account the characteristics of the protocol to implement.

Keywords: Ad-Hoc, BCHP, Ubuntu, OS, Routing Protocol.

INTRODUCCIÓN

Una red Ad-Hoc, consiste en un grupo de ordenadores que se comunican entre sí a través de señales de radio sin usar un punto de acceso centralizado (Wu & Stojmenovic, 2004). Los dispositivos que se encuentran dentro de un rango de transmisión definido pueden comunicarse directamente. Esta tecnología es utilizada en varios campos como en el ejército, juegos de video y comunicación entre dispositivos móviles. (Kulkarni, Spackmann & Giri, 2006).

Se puede definir que la forma de comunicación inalámbrica más rápida de utilizar son las redes Ad-Hoc, que están siendo utilizadas en una gran variedad de dispositivos como: sensores, computadoras portátiles, impresoras etc., que presentan varias capacidades que facilitan actividades cotidianas, pudiéndose aprovechar su continua evolución para contar con nuevas prestaciones y características que permiten utilizar redes ad-hoc para establecer enlaces de transmisión de información entre dispositivos, convirtiéndose en redes sencillas de utilizar, sin la necesidad de usar o depender de equipos intermediarios de comunicación.

Para la comunicación entre los dispositivos móviles es necesario tener reglas que permitan realizar el intercambio de información, para esto existen los protocolos de enrutamiento, nos centraremos en uno especifico BCHP (Backup Cluster Head Protocol) (Torres, 2011) aunque es necesario mencionar su predecesor o base de fundamento que es CBRP (Cluster Based Routing Protocol) (Kumar, Rishi, & Madan, 2010). BCHP es una mejora del primero. Estos protocolos proporcionan mecanismos para la elaboración y el mantenimiento de las tablas de enrutamiento.

Es así que con la utilización de redes Ad-Hoc y BCHP se pretende desarrollar una solución que implementa las características de este protocolo y permita la comunicación en dispositivos móviles sin la necesidad de equipos intermedios.

La tesis está estructurada de la siguiente manera: como primera parte se da una introducción general del trabajo realizado, definición del problema, hipótesis, objetivos y trabajos relacionados.

El Capítulo II muestra el marco referencial de la investigación, el estado del arte acerca de la comunicación inalámbrica y su influencia en los dispositivos móviles, el Capítulo III detalla el diseño de la solución de la problemática. El Capítulo IV está compuesto por la implementación de BCHP. Las pruebas se detallan en el Capítulo V y finalmente se propone las conclusiones, recomendaciones y líneas futuras.

CAPÍTULO I

ANTECEDENTES DE LA IMPLEMENTACIÓN DE UN PROTOCOLO DE ENRUTAMIENTO AD-HOC EN DISPOSITIVOS MÓVILES

1. Antecedentes

1.1. Motivación

Los avances tecnológicos y la evolución de las redes inalámbricas han llevado a la integración en nuestra sociedad de dispositivos informáticos pequeños y portátiles, estos dispositivos se caracterizan por ser móviles teniendo capacidades de procesamiento y comunicación en red, por lo que dan posibilidades muy elevadas de crear nuevos mecanismos que permitan una comunicación eficaz y sencilla (Baz, Ferreira, Álvarez, & García, 2011).

Este avance junto con el acceso generalizado a las redes de computadores e Internet ha permitido llegar a la era del intercambio en línea de la información (Torres, 2011). Por lo que es importante crear o mejorar las formas de comunicación, algoritmos, protocolos y aplicaciones que en un futuro serán necesarios.

La presente investigación abordará la utilización de un protocolo de enrutamiento ad-hoc para dispositivos móviles que permite crear una red móvil ad-hoc que no utiliza un punto de acceso para el intercambio de información, el objetivo de la implementación del protocolo es la de ser utilizado en entornos donde la infraestructura de comunicación normal no está disponible. El protocolo a implementar se denomina BCHP (Backup Cluster Head Protocol), que se encuentra en simulación en la Tesis Doctoral “Contribución a la Gestión en Redes Móviles Ad-Hoc”. (Torres, 2011).

1.2. Justificación

La forma más común de utilizar las redes inalámbricas es aquella en la cual los clientes móviles se conectan a una estación base que está encargada de controlar las comunicaciones, la estación base tiene una cierta área de cobertura en la cual todos los clientes que controla pueden comunicarse entre sí, el alcance a clientes que se encuentran en otras redes se hace a través de un segmento generalmente fijo. Los clientes pueden desplazarse y cambiar de estación base sin que su cobertura sea afectada, mediante un proceso llamado handover.

A pesar de que la topología suscriptor-estación base, es la más utilizada, las redes ad-hoc plantean nuevos retos, en este tipo de redes no existe una infraestructura de red sino que se compone de los propios nodos móviles autónomos comunicándose entre sí por enlaces inalámbricos. Es así que en este entorno desaparece el control centralizado de la red, como comúnmente se utiliza, para esta comunicación los nodos deben asumir responsabilidades de encaminamiento y de mantenimiento de la red. El control de la red está distribuido entre los mismos nodos.

De esta manera, debido a este nuevo reto de control de red, se debe utilizar protocolos de encaminamiento que permitan la comunicación entre los nodos móviles teniendo en cuenta la infraestructura Ad-Hoc, uno de esos protocolos es BCHP, establecido en la Tesis Doctoral “Contribución a la Gestión en Redes Móviles Ad-Hoc”. (Torres, 2011). Las características de este protocolo son analizadas para la implementación en dispositivos móviles.

1.3. Hipótesis

Luego de algunos estudios realizados referente a la comunicación entre dispositivos móviles y los diferentes protocolos de enrutamiento desarrollados se tiene la siguiente hipótesis:

Es posible plantear una solución que implemente en un entorno real las características de enrutamiento del protocolo BCHP, simulado en NS2 en la Tesis Doctoral “Contribución a la Gestión en Redes Móviles Ad-Hoc” (Torres, 2011), teniendo como escenario la creación de una red de comunicación en tres dispositivos móviles sin la inclusión de equipos intermedios.

1.4. Objetivos

El propósito de la presente tesis es cumplir con los objetivos definidos a continuación:

1. Implementar las características del protocolo BCHP en dispositivos móviles.
2. Realizar la comunicación de dispositivos móviles sin equipos intermediarios.
3. Verificar el funcionamiento de la solución planteada en dispositivos móviles.

CAPÍTULO II

COMUNICACIÓN INALÁMBRICA Y SU INFLUENCIA EN LOS DISPOSITIVOS MÓVILES

2. Marco Referencial

2.1. Definiciones generales

A continuación se muestra las definiciones de las tecnologías que se utilizan en el presente proyecto de tesis:

2.1.1. Redes Inalámbricas

Las redes inalámbricas permiten una forma de comunicación entre nodos sin necesidad de una conexión física, es decir sin cables, por medio de ondas electromagnéticas. Estas redes permiten que dispositivos remotos puedan comunicarse, además de no limitar a los dispositivos a estar fijos en un lugar determinado. (Shen, Jaikaeo, Srisathapornphat & Huang, 2002.) Se pueden clasificar las redes inalámbricas por su cobertura:

WWAN: Redes Inalámbricas de área extensa que son utilizadas para la transmisión de datos en espacios muy grandes, siendo las que más alcance tienen que pueden llegar a estar dentro de una ciudad como también dentro de cualquier país del mundo.

WMAN: Redes Inalámbricas de red metropolitana, es la suma de muchas redes de área local interconectadas, poseen gran versatilidad ya que a distancias cortas (4 – 10 Km) tienen una velocidad efectiva entre 1 a 10 Mbps y suelen ser usadas por redes corporativas.

WLAN: Redes de Área Local Inalámbrica, es una red que cubre alrededor de 100 metros, lo mismo que una red de área local de una empresa, existen varias tecnologías pero el estándar IEEE 802.11 es el más utilizado.

WPAN: Redes Inalámbricas de área personal, siendo redes de poca cobertura, su uso más general es para la conexión de periféricos dentro de un rango de unos metros.

La Fig. 1 nos brinda una clasificación por cobertura de las redes inalámbricas.

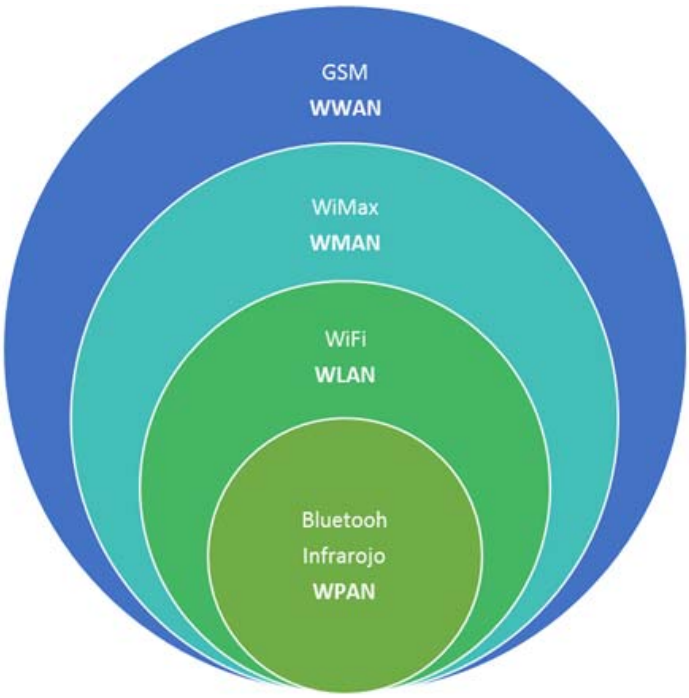

Fig. 1: Clasificación por cobertura de las Redes Inalámbricas (Autores, 2013)

Tabla 1: Cuadro comparativo de las tecnologías de las redes inalámbricas

TECNOLOGÍA	FRECUENCIA DE OPERACIÓN	VELOCIDAD DE TRASMISIÖN	ALCANCE MÁXIMO
Bluetooh	2.4 GHz	1,3,53 Mbps	100m
IRDA	2.4 GHz	9600bps y 4 Mbps	1m
WiFi	2.4 y 5.4 GHz	11-54 Mbps	300m
WiMax	2.5 a 3.5 GHZ	75 Mbps	50 Km
GSM	1900 MHz	9.6kbps	---

Tabla 2: Cuadro de tasación de tecnologías inalámbricas

TECNOLOGÍA	VELOCIDAD DE TRASMISIÓN	ALCANCE MÁXIMO	Facilidad de encontrar dispositivos con esta tecnología en nuestro medio	Total
Bluetooh	3	2	3	8
IRDA	1	1	1	3
WiFi	4	4	5	13
WiMax	5	5	1	11
GSM	2	5	3	10

De la clasificación anterior se ha tomado tecnologías representativas de cada clasificación (Perkins, 2003) para poder determinar la más idónea para la implementación que se hará en este proyecto de investigación, en la **¡Error! No se encuentra el origen de la referencia.** se puede encontrar las diferentes características de cada tecnología y en la **¡Error! No se encuentra el origen de la referencia.** la tasación correspondiente a cada una.

Con los resultados analizados se puede decir que tecnología más conveniente para esta tesis es Wi-Fi, además esta decisión se respalda por ser una de las tecnologías más utilizada como se muestra en sección 2.2 del presente trabajo.

Cabe destacar que una de las grandes ventajas de las redes inalámbricas es la facilidad de acceso a recursos en lugares donde se imposibilita la utilización de cables (Weiser, 2009), se puede compartir información almacenada en las computadoras conectadas a la red como: ejecución remota de programas de aplicación, bases de datos, documentos en general, directorios, la instalación de una red de este tipo es rápida, fácil y elimina la necesidad de tender cables entre paredes y techos, además permite a la red llegar a puntos de difícil acceso, las redes inalámbricas son versátiles, duraderas, eficaces y elimina las conexiones físicas entre nodos (Ramos, 2010).

Se puede identificar dos topologías de redes inalámbricas, las ad-hoc y las de infraestructura (Fig. 2), teniendo como principal diferencia la necesidad de usar equipos intermedios de comunicación para el enrutamiento como AP's, siendo caso de estudio dentro de esta tesis las redes inalámbricas Ad-Hoc. (Johnson, 2007).

Fig. 2: Redes Inalámbricas (Johnson, 2007)

2.1.2. Redes Inalámbricas Ad-Hoc

Las redes inalámbricas Ad-Hoc, también conocidas como MANET (Mobile Ad-hoc Networks), tienen como propósito, en dispositivos móviles, proporcionar flexibilidad y autonomía aprovechando (Alles, 2011) los principios de auto-organización, las redes móviles son descentralizadas, es decir no dependen de una administración central o nodo central, sino

que consta de nodos móviles con características que permiten el envío de información. (Wu & Stojmenovic, 2009).

Debido a la movilidad de los nodos, este tipo de redes presenta cambios frecuentes de topología, por lo que para su manejo se debe establecer reglas de creación y mantenimiento (H., B., 2005).

2.1.3. Protocolos de Enrutamiento

Los protocolos de enrutamiento proporcionan mecanismos distintos para elaborar y mantener las tablas de enrutamiento de los diferentes routers de la red, además determinan la mejor ruta para llegar a un host remoto (Jiménez, Zerna, Chávez, & Basurto, 2011).

Para la clasificación de los protocolos de encaminamiento se toma en cuenta varios criterios (Hernández, 2011) como:

- El alcance, unicast, broadcast o multicast, geocast, etc.
- El modo de descubrimiento de rutas, proactivo, reactivo, híbrido.
- Tipo de algoritmo que implementa, vector de distancias, estado de enlace.

En la clasificación basada en el alcance se distinguen dos familias: los protocolos unicast y multicast, los primeros son los que transmiten información de un único destino a un único receptor, mientras que los multicast envían la información a un grupo de nodos.

Basado en el modo de descubrimiento de rutas existen dos esquemas: el esquema reactivo y el esquema proactivo.

Los protocolos proactivos intentan tener, en cada momento e independiente de las necesidades de enrutamiento, una visión precisa del estado de la red. Busca mantener actualizadas las tablas de enrutamiento por medio del envío de mensajes de forma periódica, esto permite una respuesta rápida ante solicitudes de ruta y puede ofrecer un buen comportamiento en los escenarios de movilidad alta, sin embargo para el mantenimiento permanente de las actualizaciones de rutas se tienen una sobrecarga importante de red con los mensajes de control (Doumenc, 2008).

Los protocolos reactivos obtienen información de encaminamiento solo cuando es necesario, en otras palabras, son protocolos que trabajan bajo demanda que buscan la ruta hacia un destino en el momento en que se quiere enviar información, ocasiona una sobrecarga menor a la de los protocolos proactivos pero los retardos son mayores.

Además existen protocolos híbridos que combinan los dos esquemas antes mencionados, utilizando, por ejemplo, proactividad para las cercanías del nodo considerado pero buscando las rutas bajo demanda para nodos más alejados. La utilización de protocolos proactivos, reactivos e híbridos dependerá del escenario y del patrón de movimiento considerado. (Maldonado, 2012).

Otras formas de comunicación de los protocolos es la agrupación de nodos cercanos en un esquema denominado cluster. (Magnus, 2009) Estos protocolos son conocidos como protocolos de enrutamiento jerárquico, teniendo un nodo jefe de cluster para concentrar y distribuir la información generada dentro de cada agrupación, Un ejemplo de este tipo de protocolos es el Backup Cluster Head Protocol (BCHP). Este tipo de comunicación es el utilizado en el presente trabajo investigativo. En la sección 2.4 se especifica más sobre el mismo.

Debemos considerar como principales ventajas de los protocolos de enrutamiento jerárquico:

- La baja demanda para el descubrimiento de Rutas (Clustering)
- Reparación de enlaces rotos a nivel local
- Optimación de enrutamiento con el acortado de rutas.

2.2. Utilización de WiFi

A continuación se brinda algunas estadísticas de la preferencia hacia el tipo de dispositivo utilizado para redes WiFi a nivel mundial como localmente. (INEC, 2011).

Un informe publicado recientemente pone de manifiesto la subida del uso del WiFi público entre los usuarios de redes inalámbricas. (Browlee & Liang, 2011).

Debido al uso de dispositivos móviles durante el segundo cuatrimestre de 2011 la utilización del WiFi se vio incrementada en un 240%. La mayor cuota de uso de WiFi se la llevan los portátiles con un 48%, mientras que los smartphones tienen un 35% y las tabletas un porcentaje de utilización de WiFi del 17% que ayudan a mostrar el espectacular aumento de uso del servicio.

Debido a la movilidad de estos dispositivos se muestra las zonas en las que más usuarios se conectan a redes inalámbricas. En primer término los restaurantes, seguidas de zonas con centros comerciales y cafés. En estos casos ocho de cada diez usuarios se conectan mediante sus teléfonos de nueva generación y tabletas. Los portátiles son la elección preferida en caso de estar en universidades, en bibliotecas y en hoteles, ver Fig. 3 y Fig. 4 (JiWire, 2012).

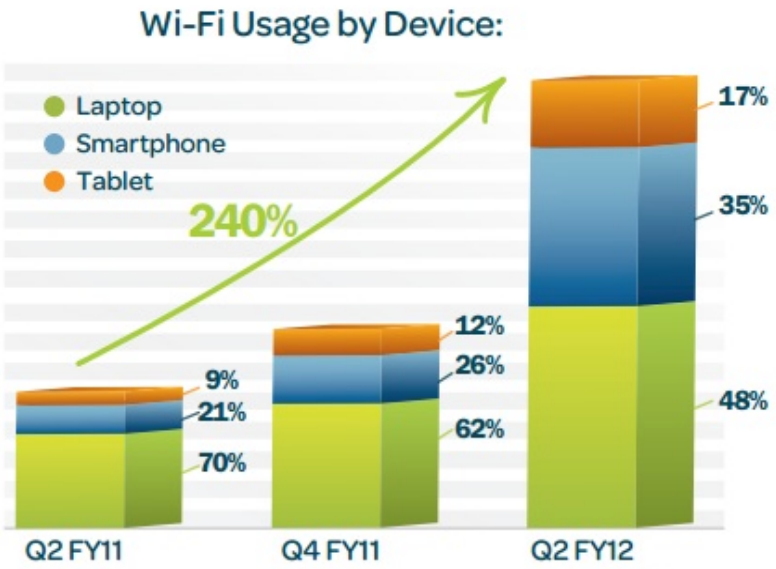

Fig. 3: Uso de WiFi por dispositivo (JiWire, 2)

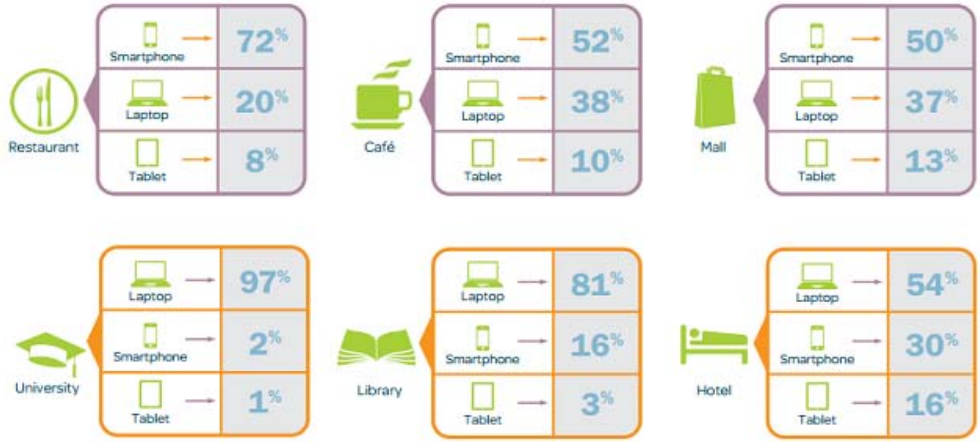

Fig. 4: Lugares donde más se utiliza WiFi (JiWire, 2012)

2.3. CBRP

Cluster Based Routing Protocol es un protocolo de enrutamiento diseñado para redes móviles, CBRP divide los nodos de la red ad-hoc en conjuntos disjuntos de dos saltos de diámetro de nominados “clusters,” que pueden superponerse entre sí, cada grupo elige un líder de grupo (clusterhead) que se encarga de mantener la información de pertenencia de los nodos al grupo y del proceso de enrutamiento (Kumar, Rishi, & Madan, 2010).

Funcionamiento

La operación de CBRP es completamente distribuida y se basa principalmente en los siguientes componentes (Maldonado, 2012):

Formación de clusters: cada nodo mantiene una tabla donde almacena información de sus nodos vecinos, como es la identidad y el estado del enlace de ese nodo (uni o bi-direccional). La tabla de vecinos se mantiene actualizada mediante la difusión de mensajes Hello, que contienen información sobre el estado del nodo, la tabla de vecinos y la tabla de adyacencias. Un nodo puede ser "clusterhead" en la red ad-hoc, si posee el id más bajo y algún enlace bi-direccional en su tabla de vecinos. (Jiang, 1999).

Descubrimiento de adyacencias al cluster: de este proceso está encargado la cabeza del grupo o clusterhead con el objetivo de descubrir todos los enlaces bi-direccionales vinculados en sus agrupaciones adyacentes, para ello utiliza la información de sus clusterheads vecinos que cada nodo mantiene en su tabla de adyacencias al cluster (CAT).

Enrutamiento: tiene dos pasos: primero descubre la ruta desde el nodo fuente al destino y luego enruta los paquetes. La diferencia con otros protocolos radica en que CBRP es capaz de reparar rutas defectuosas, si los siguientes nodos de la ruta se encuentran en la base de datos de la topología mantenida al radio de dos saltos, y también haciendo uso de esta misma información, es capaz de optimizar el enrutamiento mediante el acortado de rutas, si en el camino encuentra un sucesor directo hacia el destino. (Yasseing, 2010).

2.4. Análisis del protocolo BCHP

2.4.1. BCHP

Existen algunos inconvenientes para diseñar un protocolo de enrutamiento para redes MANET, debido principalmente que las redes móviles tienen un cambio dinámico de topología por lo tanto los protocolos de enrutamiento deben descubrir dinámicamente las rutas para llegar a su destino.

Backup Cluster Head Protocol es un protocolo de enrutamiento jerárquico, que divide los nodos en subconjuntos liderados y mantenidos por un Jefe de Cluster y mejora la disponibilidad de la red por la inclusión de nodos (JCR) Jefes de Cluster de Respaldo. Este protocolo está formado por tres etapas:

Etapa 1. La creación:

Los nodos no pertenecen a ningún clúster cuando inician, por lo que su estado empieza, como se muestra en la Fig. 5 con UNDECIDED, en el cual se calcula la métrica, que es enviada mediante mensajes de difusión para que con estos valores cada nodo cree su tabla de enrutamiento y se pueda determinar el nodo JC (Jefe de Clúster) y JCR (Jefe de Clúster de Respaldo), el resto de nodos que se unan al cluster se los denomina nodos comunes.

***Fig. 5*: Creación *del clúster* (Autores, 2013)**

Etapa 2. El mantenimiento:

Para el mantenimiento del clúster son necesarios tres conceptos básicos, como se muestra en la Fig. *6* primero debe existir la jerarquía dentro del clúster, es decir conocer el movimiento y estado de los nodos, si un nodo JC está dentro de la cobertura de otro nodo JC entonces se inicia un periodo de contención, y luego de que éste expira se elige nuevamente el nodo JC y JCR del nuevo clúster.

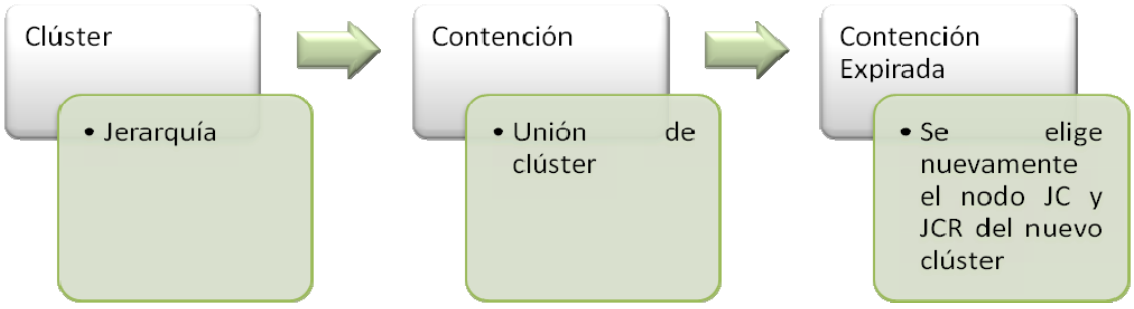

Fig. *6*: Mantenimiento del clúster (Autores, 2013)

Etapa 3. El enrutamiento entre cluster:

Existen tres niveles de enrutamientos, descritos en la Fig. 7.

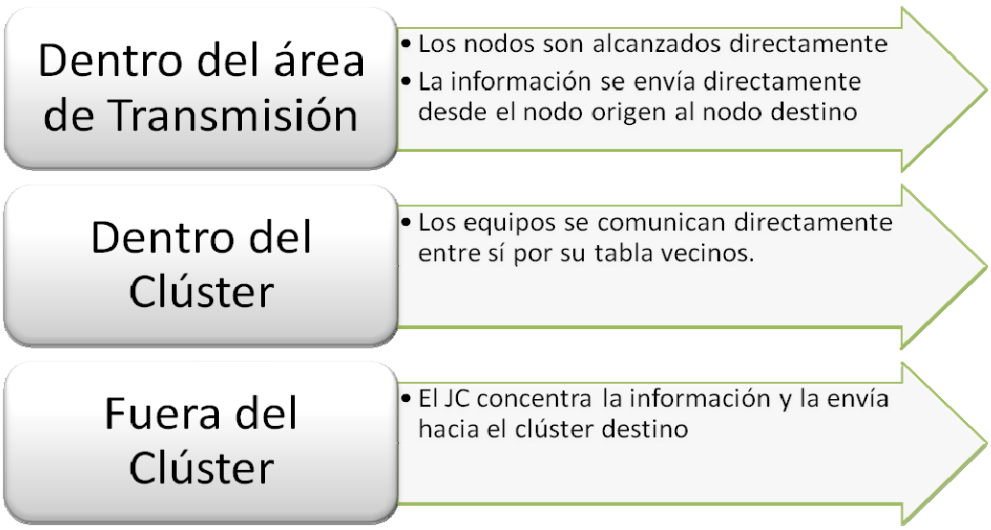

Fig. 7: Enrutamiento entre cluster (Autores, 2013)

Las funciones necesarias para la operación de BCHP son:

- Registro de los clúster de los cuáles puede interactuar.
- Tablas de vecinos
- Tabla de adyacencias
- Base de datos de topología de 2 saltos
- La información de su estado dentro del protocolo
- El valor de su métrica

Para lograr cumplir con los objetivos de este protocolo se han adicionado los estados mostrados en **¡Error! No se encuentra el origen de la referencia.**.

Tabla 3: Estados adicionales de BCHP

ESTADO	FUNCIÓN
CLUSTER_HEAD_BACKUP	Identifica el nodo JCR (Jefe de Clúster de Respaldo)
NMS	Identifica el nodo SA (Servidor de Administración)
NMS_RESPALDO	Identifica el nodo SAR (Servidor de Administración de Respaldo)

Todos los nodos tienen campos adicionales para almacenar información del nodo SA, SAR, JC, y JCR, sin embargo sólo estos últimos tienen conocimiento sobre las direcciones del otro, los demás nodos tienen este campo con un valor nulo.

2.4.2. Definición de Clases

Definición de la estructura lógica

En la Fig. 8 se muestran los componentes de un nodo con BCHP.

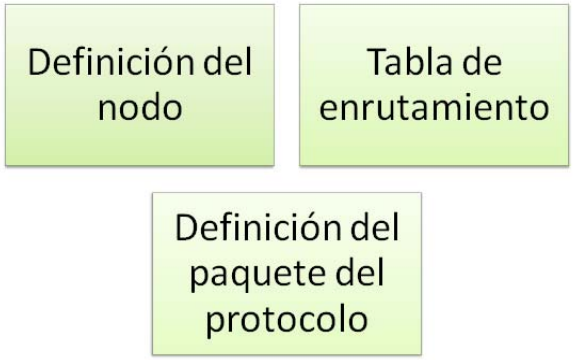

Fig. 8: Componentes BCHP (Autores, 2013)

2.4.3. Paquetes Utilizados

En la **¡Error! No se encuentra el origen de la referencia.** y **¡Error! No se encuentra el origen de la referencia.** se describe los paquetes y mensajes utilizados por BCHP.

Tabla 4: Paquetes de BCHP

PDU	Descripción
RREQ	Paquete de solicitud de enrutamiento basado en origen, utilizado en el descubrimiento de rutas.
RREP	Paquete de Respuesta de Ruta, utilizado en el descubrimiento de rutas.
Routing	Paquete para el enrutamiento, participa en el proceso de enrutamiento y acortamiento de rutas.
ERROR	Paquete para informar errores de enrutamiento, participa en la reparación de rutas local.

Tabla 5: Mensajes de BCHP

PDU	Descripción
HELLO Vecino	Envía la tabla de rutas a todos los nodos a través de un broadcast, participa en la elección del JC, JCR. Mantenimiento del clúster y actualización de la tabla de rutas.
HELLO Clúster Extensión	Parte del mensaje HELLO, se utiliza en el descubrimiento de clusters adyacentes.

Mensaje HELLO

Consta de dos partes y es enviado por todos los nodos en forma de broadcast, la Fig. 9, muestra las partes del mensaje HELLO utilizadas para la actualización de la tabla de rutas así como también para la elección de los nodos JC y JCR.

El Estado se lo informa según los siguientes valores: UNDECIDED (0), JC (1), NM (2), JCR (3), la Longitud muestra la cantidad de vecinos listados, la Métrica almacena el valor obtenido del cálculo relacionado con las capacidades del nodo como procesamiento, movilidad o estado de la batería. Bandera L informa del estado del enlace con el vecino, Bandera R identifica si el vecino tiene un estado igual a JC, El campo Dirección del vecino es el identificador del vecino, el Rol del Vecino tiene la información del estado del vecino remitente y la métrica del Vecino es el peso calculado.

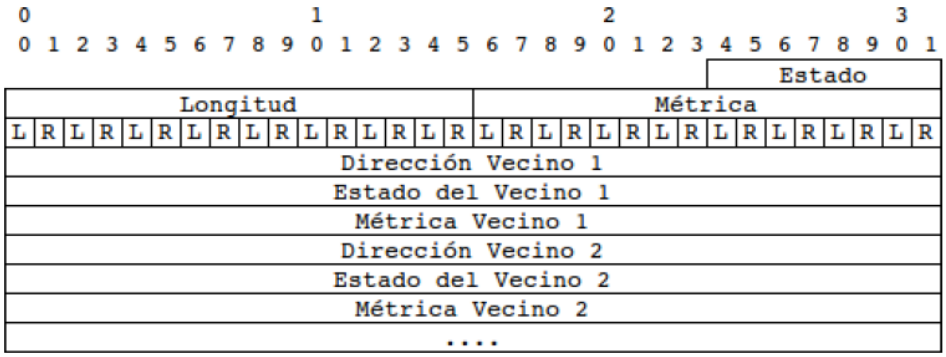

Fig. 9: Estructura del mensaje HELLO para nodos vecinos (Torres, 2011)

Para el descubrimiento de clúster adyacentes, la creación y mantenimiento del backbone, la elección del nodo SA y SAR se utilizan los campos mostrados en la Fig. 10 Así la cantidad de JC adyacentes listados se almacena en el campo Longitud, en Bandera L se encuentra el tipo de enlace con el JC, la Dirección del JC adyacente es la dirección del nodo vecino, Bandera R identifica si el vecino tiene un estado igual a JC, la Dirección del vecino es el identificador del vecino y la Métrica del JC adyacente es el peso calculado.

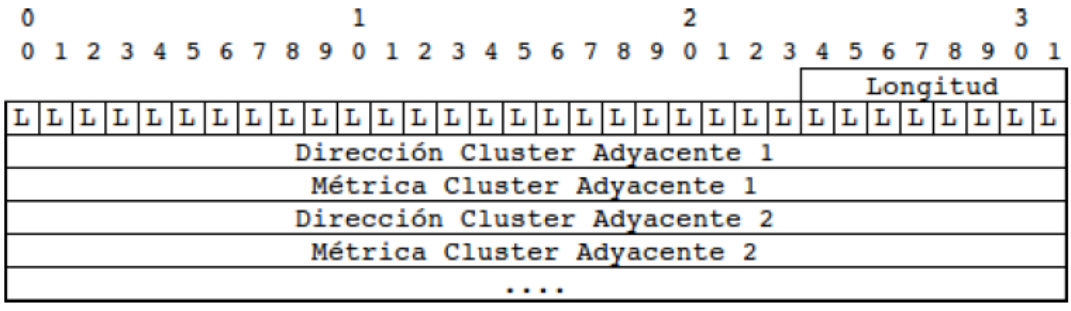

Fig. 10: Estructura del mensaje HELLO para nodos adyacentes (Torres, 2011)

Solicitud de Ruta

Los campos utilizados en esta parte del mensaje se encuentran en la Fig. 11, estos campos están compuestos por Tipo que cuando es solicitud de ruta, el protocolo lo identifica colocando en este campo valor binario 10, Num1 que es la cantidad de pares JC y nodos Gateway a los cuales se envía la solicitud de ruta, Num2 es la cantidad de Clúster que la solicitud de ruta está atravesando al momento, Identificación donde está de forma única cada solicitud de ruta, la Dirección de destino, la del nodo Gateway, del JC y la dirección de Clúster.

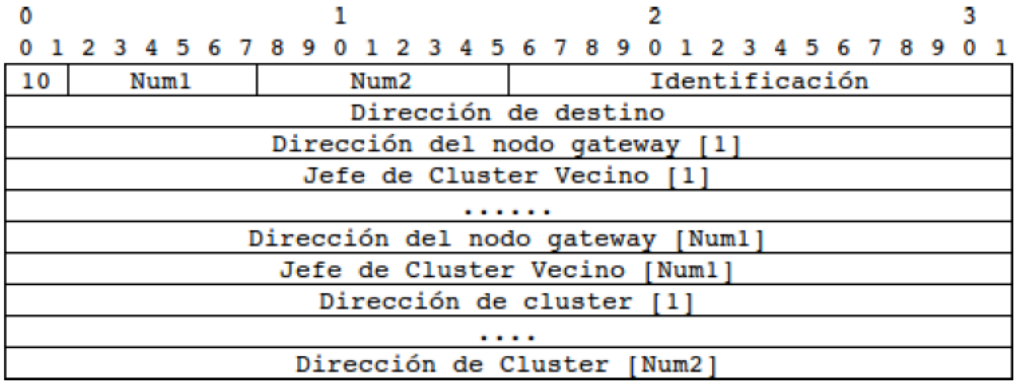

Fig. 11: Estructura del mensaje RREQ (Torres, 2011)

Respuesta de solicitud de Ruta

Los campos de este paquete para la respuesta de solicitud de Ruta se encuentran en la Fig. 12 los cuales están compuestos de: Tipo se coloca el valor binario 01 que identifica cuando es respuesta de ruta, Num1 es la cantidad de Clúster que la solicitud de ruta está atravesando al momento, Bandera G indica que esta respuesta es prescindible debido que es una sugerencia producto un proceso de acortamiento de ruta en un nodo intermedio, Num2 es el número de direcciones en la ruta calculada, Identificación identifica en forma única a cada respuesta, Dirección del Clúster y la Ruta Calculada.

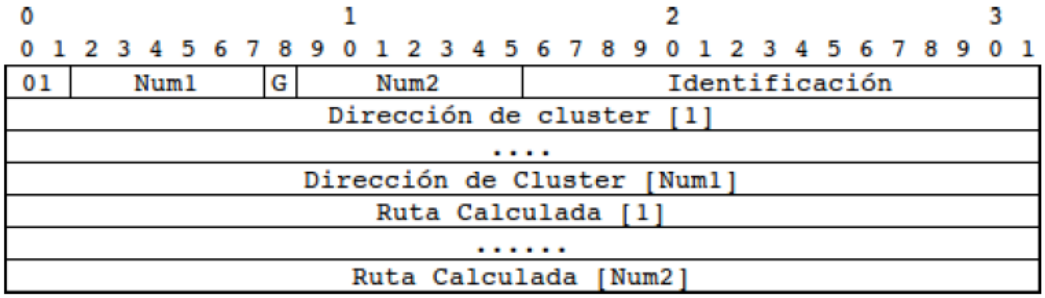

Fig. 12: Estructura del mensaje RREQ respuesta de solicitud de ruta (Torres, 2011)

Enrutamiento

Los campos se detallan en la Fig. 13, estos campos llevan un Tipo con el valor binario 00 que indica un paquete normal de CBRP, Num con la cantidad de direcciones que forman la ruta, Bandera S que indica si la ruta ha sido acortada, Bandera R indicando si la ruta ha sido localmente reparada, Número Actual con la dirección del último nodo visitado y la Dirección compuesta por la secuencia de direcciones que forma la ruta.

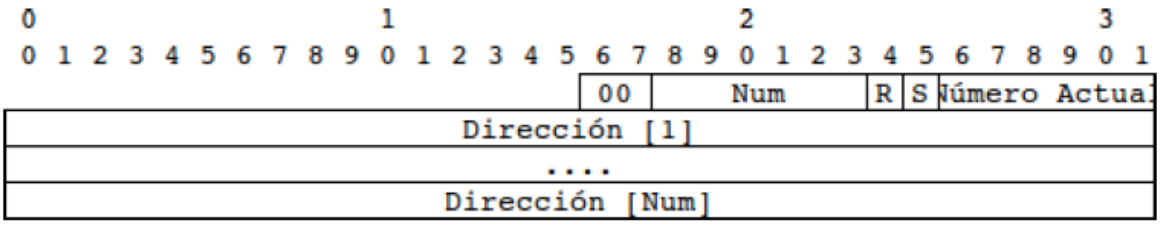

Fig. 13: Estructura del mensaje de enrutamiento (Torres, 2011)

Error

El paquete de error está compuesto por los campos Tipo con el valor binario 11 que es un paquete de error de CBRP, Num que muestra la cantidad de direcciones que forman la ruta, Número Actual compuesto por la dirección del último nodo visitado, Dirección con la secuencia de direcciones que forma la ruta, Enlace dañado desde dirección y el Próximo salto inalcanzable como se muestra en la Fig. 14.

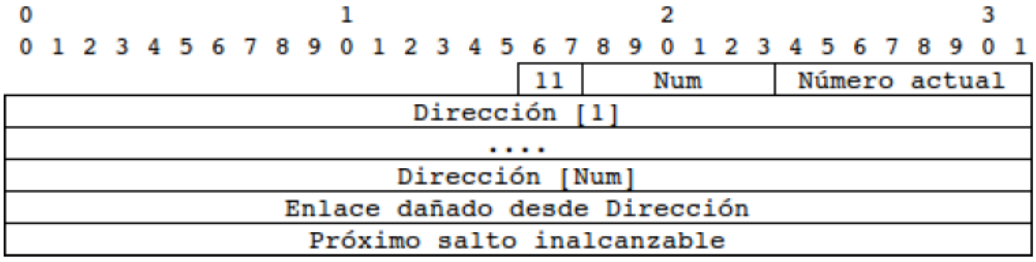

Fig. 14: Estructura del paquete de Error (Torres, 2011)

2.4.4. Tabla de Enrutamiento

Según el tipo de nodo se cuenta o no con dos tablas de enrutamiento, como se muestra en la Fig. 15, cada nodo cuenta con una tabla de vecinos dentro del dominio de clusters, esta tabla de enrutamiento le permite tener un conocimiento de la topología del cluster al cual pertenece y cuando uno nodo es JC o JCR además debe mantener una tabla de rutas adicional, la tabla de adyacencias la cual mantiene registros de los cluster vecinos.

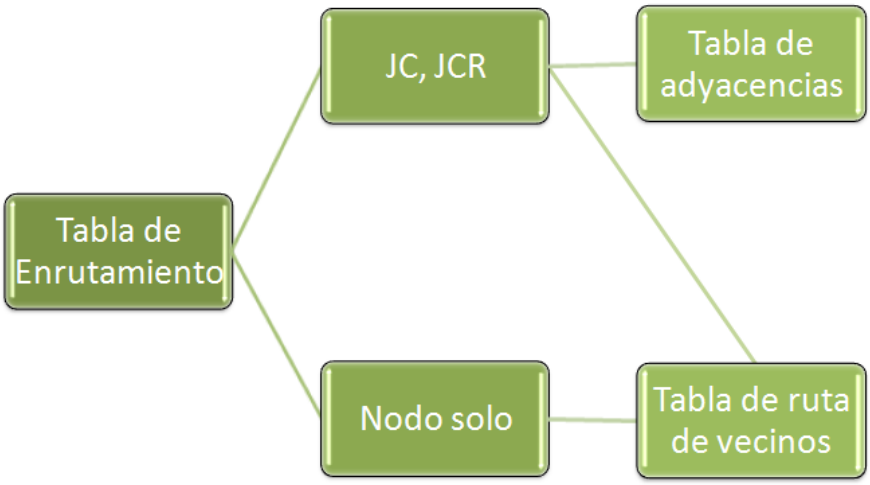

Fig. 15: Tipos de tablas de enrutamiento (Autores, 2013)

Tabla 6: Entradas para la tabla de ruta de vecinos

Id Vecino	Estado Vecino	Estampa de Tiempo	Estado enlace	Contención	Métrica

Cada nodo tiene una tabla de vecinos, los campos se muestra en la **¡Error! No se encuentra el origen de la referencia.**, dentro del dominio del cluster, esta tabla de enrutamiento le permite al nodo tener conocimiento de la topología del clúster al cual pertenece. Además nos permite almacenar la métrica para su posterior uso en la elección del jefe de cluster y para el proceso de contención del cluster. A continuación se detallan las entradas en la tabla de rutas:

Identificador del vecino: se almacena la identificación de cada uno de los vecinos del nodo, el tamaño de este campo es de 128 bits ya que puede almacenar dirección IPv4 o IPv6.

Estado vecino: se almacena de qué tipo de nodo es el vecino, el tamaño de este campo es de 3 bits.

Estampa de tiempo: tiene un campo de tamaño de 64 bits para almacenar el tiempo en la cual esta ruta tuvo la última actualización.

Estado del enlace: donde se determina si el nodo está dentro del rango de transmisión, tiene un tamaño de 1 bit.

Contención: nos permite almacenar dos bits con información para determinar cuál es el nodo que está en contención con el JC, es decir cuando dos nodos en estado JC están dentro del dominio de un cluster.

Métrica: almacena la métrica calculada y enviada por los vecinos del nodo. Sirve para los procesos de mantenimiento del cluster y elección del JC.

Como se muestra en la **¡Error! No se encuentra el origen de la referencia.**. cuando un nodo es JC, JCR, SA o SAR además de la tabla ruta de vecinos, se tiene una tabla adicional, denominada tabla de adyacencias que está encargada de mantener los registros de los cluster vecinos.

La red entre clusters es denominada backbone debido a que a través de esta se envía la información de enrutamiento. La tabla de adyacencias está compuesta por una dupla formada por entradas para los jefes de cluster y los respectivos nodos puente, que son aquellos que pueden tener más de un nodo JC, es decir pueden pertenecer a más de un cluster.

A continuación se muestra el formato de la tabla de adyacencias:

Tabla 7: Entradas para la tabla de ruta de adyacencias

id Vecino	Estado Vecino	Estampa de Tiempo	Métrica	NP1	Estampa de Tiempo NP1
				NP2	Estampa de Tiempo NP2
				NPn	Estampa de Tiempo NPn

Identificador del vecino: almacena la identificación de cada uno de los vecinos del nodo. Dependiendo del protocolo de la capa IP, puede ser IPv4 o IPv6.

Estado del vecino: almacena información de que tipo de nodo es el jefe cluster. Sirve para identificar cual nodo es el SA y SAR, tiene un tamaño de 3 bits.

Estampa de tiempo: tiene la información acerca del tiempo en la cual esta ruta tuvo la última actualización. Tiene un tamaño de 64 bits.

Métrica: almacena la métrica calculada y enviada por los vecinos del nodo. Se utiliza para los procesos de elección del SA y del SAR**.**

NP1 … NPn: tiene la especificación de los nodos puente que sirven para comunicarse con los jefes de cluster vecinos.

Estampa de tiempo NP1 … NPn: almacena la última actualización hacia el nodo puente respectivamente.

2.5. Trabajos relacionados con las diferentes formas de comunicación en redes Ad-Hoc.

Simulation of Mobile Ad hoc Routing Strategies (Boukhalkhal, Yagoubi, Djoudi, Ounten, & Benmohammed, 2009): Muestra las estrategias utilizadas por los diferentes protocolos, proactivos, reactivos e híbridos, para evaluar en detalle su rendimiento, se comparan tres protocolos, DSDV, AODV y CBRP esta comparación se la realiza mediante el uso del simulador NS2.

Este proyecto permite tener una visión de los diferentes protocolos de enrutamiento y su rendimiento en el simulador NS2, especialmente del protocolo CBRP, base para realizar el protocolo BCHP, que es el que va a ser implementado en la presente tesis y que se encuentra también desarrollado en NS2.

Contribución a los Modelos de Gestión de Redes Móviles Ad-Hoc (Torres, 2011): se trata de una tesis doctoral que muestra una solución eficiente para la gestión de redes móviles Ad-Hoc. La solución contempla dos componentes principales: la definición de un modelo de gestión para redes móviles de alta disponibilidad y la creación de un protocolo de enrutamiento jerárquico asociado al modelo. Como parte del modelo se diseña el protocolo de enrutamiento Ad-Hoc denominado Backup Cluster Head Protocol (BCHP), que utiliza como estrategia de encaminamiento el empleo de cluster y jerarquías. Cada cluster tiene un Jefe de Cluster que concentra la información de enrutamiento de gestión y la envía al destino cuando esta fuera su área de cobertura. Para mejorar la disponibilidad de la red el protocolo utiliza un Jefe de Cluster de Respaldo el que asume las funciones del nodo principal del cluster cuando este tiene un problema.

Esta tesis doctoral realiza el diseño del protocolo BCHP, por consiguiente es la base del análisis de este proyecto. BCHP se encuentra desarrollado en el simulador NS2 y el objetivo de nuestra tesis es la implementación de sus características en entornos reales de comunicación.

The Serval Project (Gardner-Stephen, 2012): Este proyecto se basa en dos sistemas, el primero es temporal y se basa en redes auto-organizadas y configuradas para zonas de desastre formadas por pequeñas torres de telefonía. El segundo sistema es permanente para áreas remotas que no requieren infraestructura y crea una red telefónica entre dispositivos móviles que tengan WiFi, este sistema está diseñado especialmente para teléfonos móviles que pueden operar en otras frecuencias sin licencia, ambos sistemas también se pueden combinar.

Este proyecto es el que más se asemeja a nuestro trabajo, aunque actualmente se encuentra en fase de desarrollo. Por lo que se ha tenido énfasis en el estudio de este proyecto para tener una base comparativa con el protocolo que se utiliza en el mismo, el cual se denomina B.A.T.M.A.N.:

B.A.T.M.A.N. (Better Approach To Mobile AdHoc Networking) (Gardner-Stephen, 2012) es un protocolo de enrutamiento que en la actualidad se encuentra en fase de desarrollo por la comunidad “Freifunk”. BATMAN es un protocolo proactivo para redes inalámbricas Mesh Ad-Hoc, incluyendo las redes móviles Ad-Hoc.

Este protocolo mantiene proactivamente la información sobre la existencia de todos los nodos de la red Mesh que son accesibles con la comunicación de un solo salto o de múltiples saltos, la estrategia que utiliza B.A.T.M.A.N. es determinar para cada destino de la red Mesh un único salto vecino que pueda ser usado como la mejor puerta de enlace para comunicarse con el nodo destino.

El algoritmo que utiliza B.A.T.M.A.N. (Neumann, 2008) se centra en aprender sobre el mejor salto próximo para cada destino, hace un análisis estadístico de la pérdida de paquetes del protocolo y la velocidad de propagación y no depende del estado o topología de la información de otros nodos.

Los paquetes de este protocolo contienen solo una cantidad limitada de información, los paquetes perdidos debido a enlaces poco fiables, no son contrarrestados con redundancia sin embargo son detectados y utilizados para mejores decisiones de enrutamiento. B.A.T.M.A.N. elige la ruta más confiable para las decisiones del próximo salto de los nodos individuales. B.A.T.M.A.N. no tiene una entidad central que conozca todas las posibles vías a través de la red, todo nodo solo determina el dato para elegir el próximo salto.

De esta manera, el algoritmo del protocolo se lo puede expresar de la siguiente manera: cada nodo transmite mediante broadcast mensajes, llamados Mensajes de Originator OGM, para informar a sus nodos vecinos sobre su existencia, los vecinos vuelven a difundir los OGM para avisar a sus vecinos sobre la existencia del iniciador original del mensaje y así sucesivamente.

La red se inunda de mensajes de originador, estos mensajes están compuestos por la dirección del originador, la dirección del nodo al que transmite el paquete un TTL y un número secuencial. B.A.T.M.A.N detecta la presencia de Originadores B.A.T.M.A.N, no importa si el camino de comunicación desde/hacia un Originador es de un salto o de multisalto. El protocolo no trata de encontrar el camino completo de la ruta, en cambio solo aprende cual enlace local es la mejor puerta de enlace de cada originador. También mantiene el rastro de los nuevos originadores e informa a sus vecinos sobre su existencia. El protocolo asegura que una ruta consiste solo de enlaces bidireccionales.

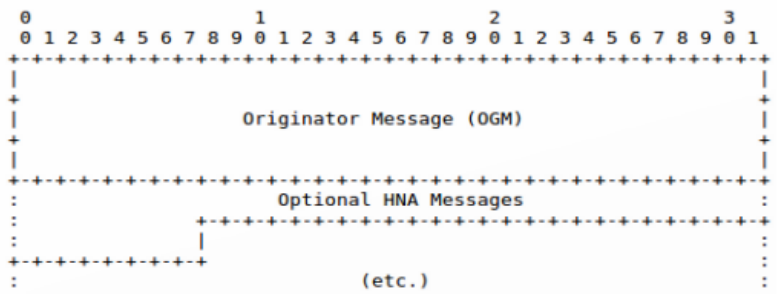

Fig. 16: Formato del paquete general de B.A.T.M.A.N. (Trujillo, 2010)

La Fig. 16, nos muestra como se encuentra el formato general del paquete del protocolo B.A.T.M.A.N.:

Cada paquete B.A.T.M.A.N está encapsulado en un paquete de datos sencillo.

Un paquete B.A.T.M.A.N consta de un mensaje originador y cero o más mensajes adjuntos de extensiones HNA.

Mensaje Originado OGM: tiene un tamaño fijo de 12 octetos.

Mensajes opcionales de extensiones HNA: Un HNA (Host Network Announcement) es un tipo de mensaje usado para anunciar una puerta de enlace a una red o a un host. Cada mensaje de extensión que sigue un OGN está asociado con el OGM precedente y debe ser procesado respectivamente. Los mensajes HNA tienen un tamaño fijo de 5 octetos.

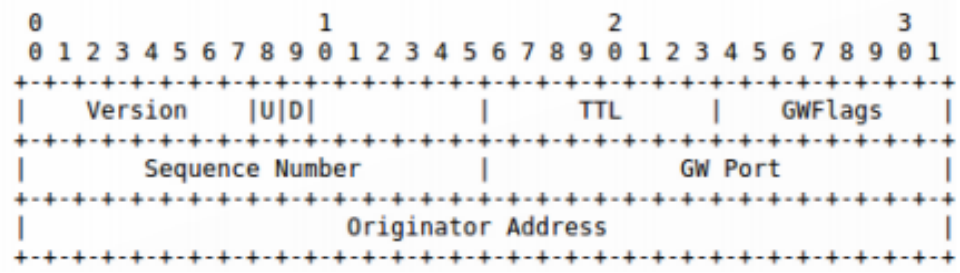

Fig. 17: Formato OGM (Trujillo, 2010)

El formato OGM está compuesto por los siguientes campos: (Ver **Fig.** 17)

- Versión: cada paquete recibido con un espacio de versión diferente debe ser ignorado.
- Bandera Is-direct-link: esta bandera indica si un nodo es un vecino directo o no.
- Bandera unidireccional: esta bandera indica si el nodo es bidireccional o no.
- TTL (Time To Live): El TTL puede ser usado para definir un límite superior de un número de saltos que un OGM puede ser transmitido.
- Banderas de puerta de enlace (GWFlags): Anuncia si el nodo tiene acceso a internet.
- Número secuencial: el originador de un OGM enumera consecutivamente cada nuevo OGM con un incremento (de uno) el número secuencial.
- Dirección del originador: La dirección IPv4 de una interfaz B.A.T.M.A.N en la cual ha sido creado el OGM.

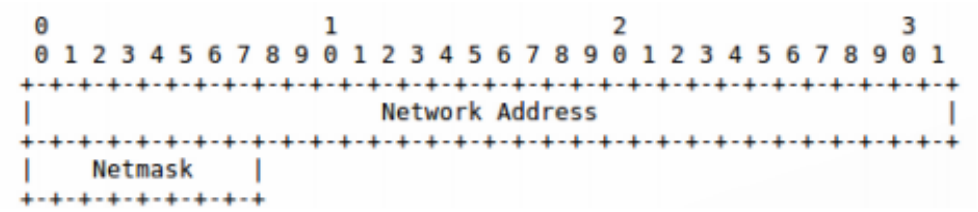

Fig. 18: Formato de la extensión HNA (Trujillo, 2010)

En la Fig. 18 se puede observar el Formato de la extensión HNA que tiene los siguientes campos:

- Máscara de red: el número de bits presentando el tamaño de la red anunciada.
- Dirección de red: La dirección de red Ipv4 de la red anunciada.

El protocolo B.A.T.M.A.N. trabaja en dispositivos celulares móviles, por lo que su estudio es de gran ayuda para conocer como es la implementación en los teléfonos Android.

Estos trabajos se basan en redes que pueden estar funcionando solamente con el tipo de comunicación WiFi de los dispositivos móviles, la solución que planteamos es diferente, además de utilizar una red WiFi se incorpora en los dispositivos características de un protocolo de enrutamiento que mejora la disponibilidad de la red por inclusión de nodos JCR (Jefe de Cluster), lo que permite ser un protocolo diseñado especialmente para escenarios en los cuales la comunicación debe ser rápida y sin equipos intermedios.

CAPÍTULO III

DISEÑO DE LA SOLUCIÓN

3. Propuesta de solución

Este proyecto permitirá a varios dispositivos (3) móviles conectarse en una red ad hoc mediante la implementación de las características del protocolo BCHP (Backup cluster Head Protocol), un protocolo de enrutamiento jerárquico, las plataformas a utilizar son Ubuntu y Android, el lenguaje de programación es C y java. La tecnología de comunicación es Wi-Fi.

Gracias al análisis que se ha desarrollado en el presente trabajo de fin de carrera, se conoce la problemática del tema planteado “Implementación de las características de un protocolo de enrutamiento ad-hoc en dispositivos móviles” se tiene la información necesaria para plantear una propuesta de solución, la misma que se basa en la utilización de dispositivos móviles como medio para trabajar sobre el sistema operativo Ubuntu.

Existen algunos trabajos relacionados que se basan en redes que pueden estar funcionando con el tipo de comunicación WiFi de los dispositivos móviles, la solución que planteamos es diferente ya que, además de utilizar una red WiFi, incorpora en los dispositivos las características del protocolo de enrutamiento que mejora la disponibilidad de la red por inclusión de nodos JCR (Jefe de Cluster), lo que permite ser un protocolo diseñado especialmente para escenarios críticos en los cuales la comunicación debe ser rápida, sencilla y confiable. En el presente trabajo de investigación se referencia indistintamente a los dispositivos móviles como nodos.

En la Fig. 19 se muestra el modelo de comunicación propuesto, para el escenario de pruebas del mismo se ha planteado hacerlo con tres dispositivos móviles conectados al mismo segmento de red, dentro del alcance de las antenas de transmisión inalámbrica de cada uno.

Para realizar la asignación de los roles de cada dispositivo, se plantea hacerlo de acuerdo al orden de conexión de cada uno, es decir el primer nodo en ingresar a la red será el nodo JC (Jefe de Cluster), el JCR (Jefe de Clúster de Respaldo) será el siguiente nodo en establecer conexión en la red con el JC, el tercer nodo en establecer la conexión será un nodo común, sin embargo se plantea como característica propia del protocolo la sucesión de roles en el segmento de red, de este modo el JCR puede llegar a ser JC ante la ausencia del primer nodo, al igual que el nodo común (tercer nodo) podrá ser JCR y JC respectivamente de darse la pérdida de los nodos anteriores.

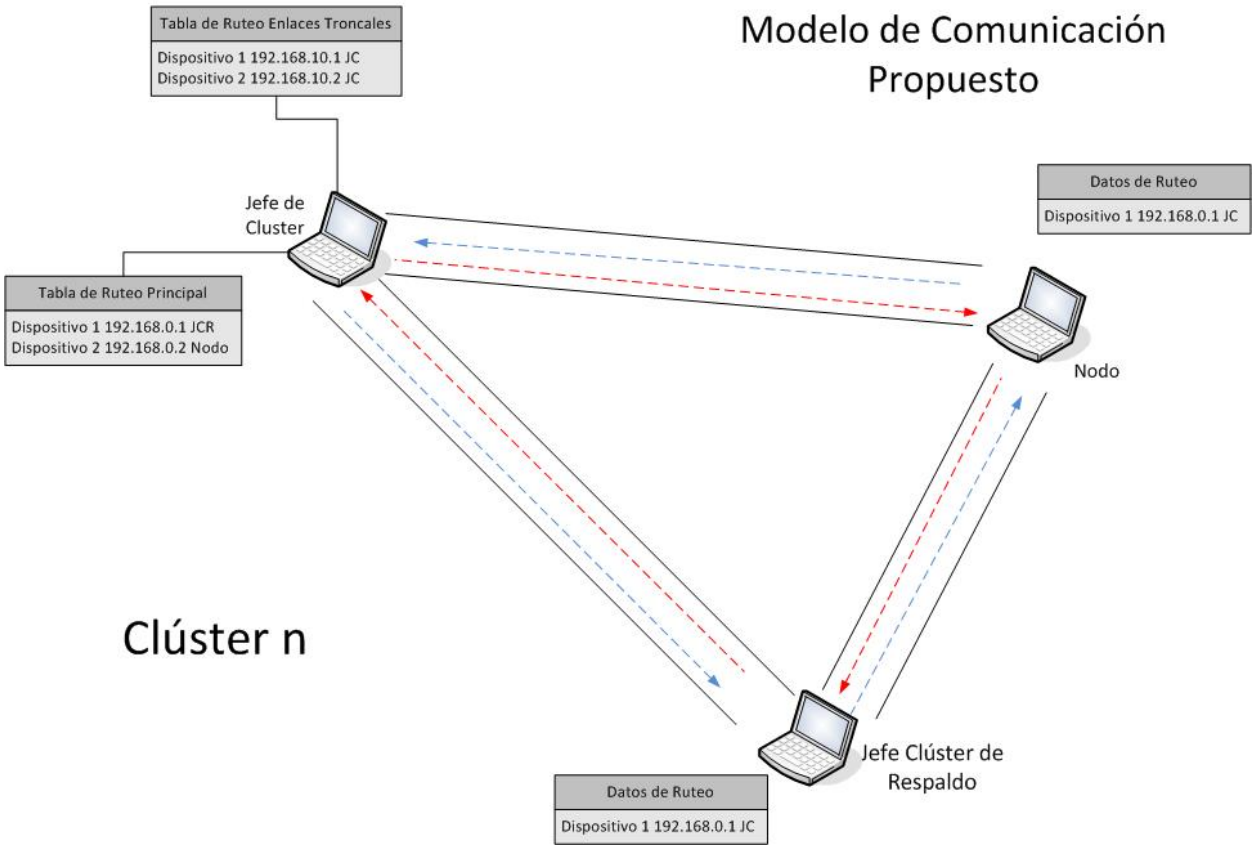

Fig. 19: Modelo de comunicación propuesto (Autores, 2013)

3.1. Escenario de BCHP

El funcionamiento de la aplicación se basa en la creación de una comunicación Ad-Hoc entre los dispositivos, para lo cual se toma como base el modelo propuesto en esta tesis, que incluye:

- Tres dispositivos móviles comunicados entre sí.
- Protocolo de enrutamiento BCHP.
- Software de comunicación. (Java Chat BCHP, Anexo 2)
- Sistema operativo Linux Ubuntu LTS 10.04 o Android 4 ICS. (Android Libre, 2012)
- Tecnología Wi-Fi Direct en el dispositivo móvil Android. (Noticiero Universal, 2013).

Existen algunos factores que son necesarios para el funcionamiento del protocolo. Estos factores están compuestos por las características de hardware y software que afectarán directamente en el desarrollo de la implementación de BCHP como se muestra a continuación:

- Los dispositivos móviles deben tener la capacidad de tener tarjetas inalámbricas.
- Compilador GCC (GNU Compiler Collection) en los sistemas Ubuntu,
- Correr y compilar Java (ambos SO).
- Estos requisitos se encuentran detallados en la sección 4.2

Teniendo en cuenta estas características se procede a desarrollar la solución al proyecto como se muestra en la Fig. 20, que muestra de manera general el esquema de comunicación propuesto.

.

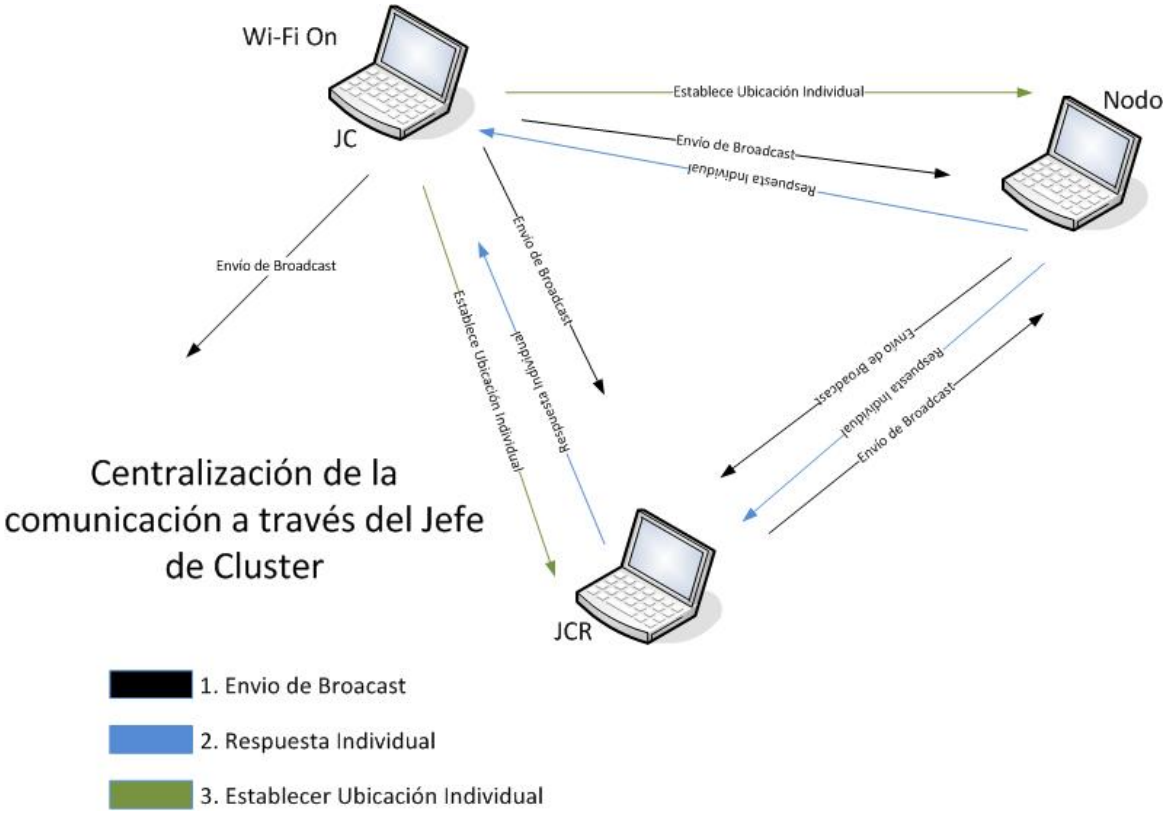

Fig. 20: Centralización de la comunicación (Autores, 2013)

La implementación del protocolo debe verificar la existencia de nodos vecinos y la existencia o no de un cluster, si no pasa a formarlo adquiriendo así la distinción de Jefe de Cluster que centraliza toda la información del cluster, tiene la capacidad de unir a más nodos al cluster estableciendo conexiones directas con cada uno de ellos.

La Fig. 20 muestra como se realiza la operación del descubrimiento de nodos dentro del alcance y cómo se mantiene la conexión entre todos. El JC establece la ubicación de los nodos al enviar una petición en broadcast para llegar a todos los dispositivos que se encuentren en el rango, mientras escucha que dispositivo devuelve esta petición.

De esta manera se puede crear una lista de dispositivos para crear el cluster, estableciendo una conexión directa con cada uno de los equipos que el usuario permita pertenecer al mismo. Una vez que se ha determinado el equipo con el que se desea conectar se establece la conexión.

Este proceso se detalla en la Fig. 21 en donde se muestra la comunicación general entre dos dispositivos a través del modelo TCP/IP. (Murray, 2010). Es así que la solución planteada es referente a este modelo que nos brinda una comunicación integral de extremo a extremo, definiendo en cada fase el método de empaquetado de datos para asegurar la comunicación.

Al ser una solución integral, se ha tenido algunos inconvenientes referentes a la infraestructura donde debe funcionar, propio de la diversidad de dispositivos que existe alrededor del mundo, por lo que se ha planteado una solución genérica que debe funcionar en la mayor parte de dispositivos.

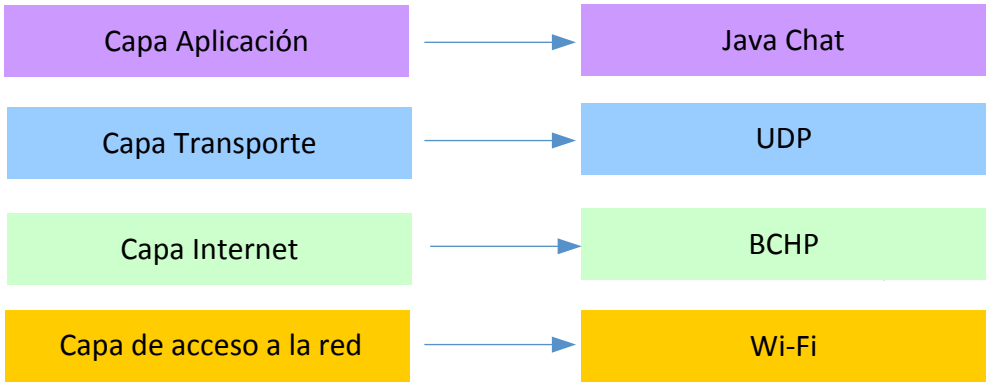

Fig. 21: Arquitectura TCP/IP (Autores, 2013)

Para realizar el trabajo en las diferentes capas de abstracción del modelo TCP/IP se desarrolla la implementación de librerías con las bases del protocolo BCHP que funcionan en la capa de transporte, pero que interactúan con capas inferiores (Internet y Acceso a la red) para establecer parámetros de configuración de hardware como las tarjetas de red e interfaces. Las librerías se ejecutan a nivel de sistema operativo y reemplazan a las librerías originales de enrutamiento.

La arquitectura TCP/IP, se muestra en la Fig. 21, en donde se plasma las tecnologías y protocolos que se utilizarán en cada capa de esta arquitectura.

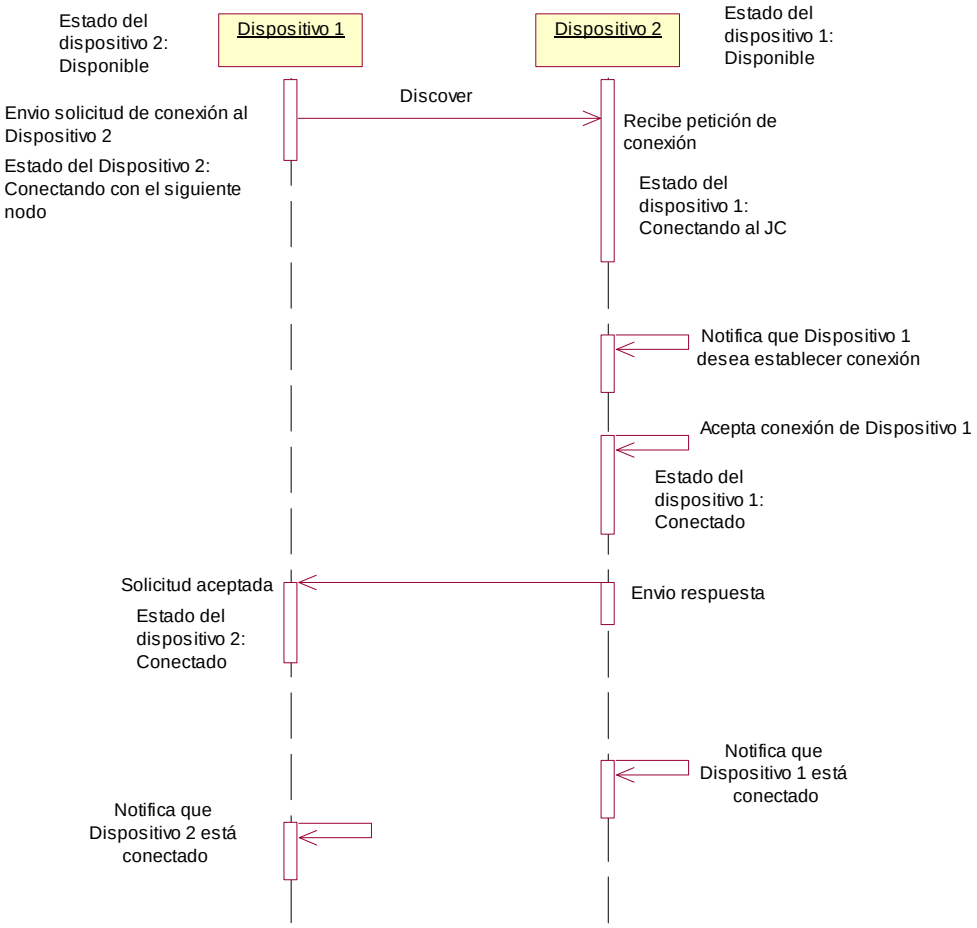

Fig. 22: Comunicación entre dispositivos (Autores, 2013)

El algoritmo utilizado para la asignación del estatus de cada nodo se puede observar en la Fig. 23, donde se crea un flujo de datos en el que se ve claramente la lógica del funcionamiento de la asignación del estado de cada nodo. Para aplicar este algoritmo se ha establecido que al menos deben existir dos nodos.

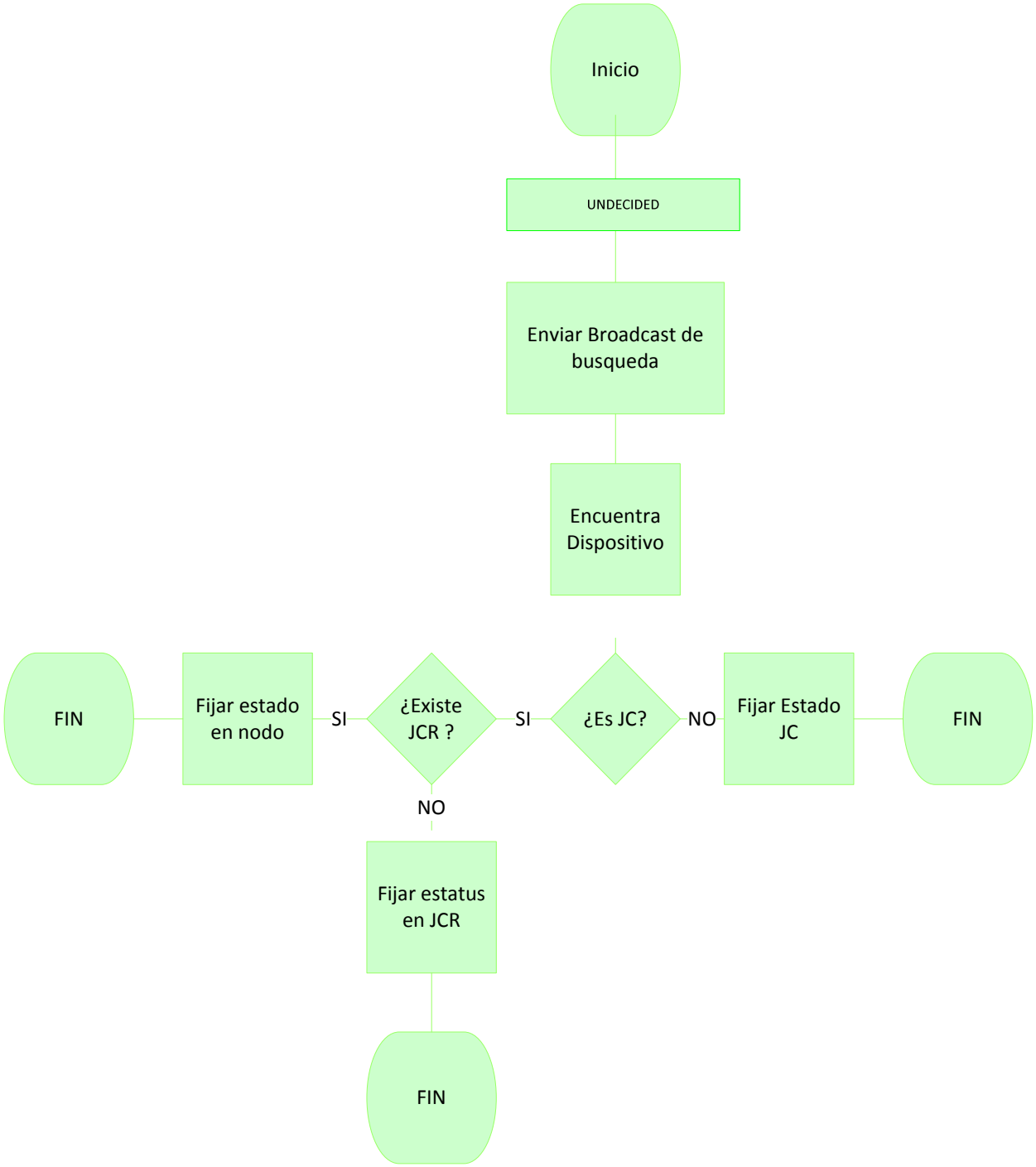

Fig. 23: Flujo de asignación de estados (Autores, 2013)

3.2. Fases BCHP

3.2.1. Inicialización del nodo

La fase inicial de todo el proceso (ver Fig. 24) se realiza cuando un nodo tiene el estado de UNDECIDED y luego a través del protocolo realiza una búsqueda de dispositivos cercanos (dentro del alcance) mediante el envío de paquetes Hello.

Cuando el nodo, luego de realizar la solicitud de búsqueda, decide establecer una conexión con cualquier dispositivo de la lista, empieza la formación del cluster, el JC y el JCR llevan un registro el uno del otro, se inicializa la tabla de enrutamiento. Esta tabla cuenta con campos como: la IP del dispositivo, identificación del dispositivo, dirección MAC y un estado, estos campos permiten mejorar la disponibilidad de la red y el tiempo de convergencia debido a que son referencias almacenadas en memoria para la identificación y ubicación de los nodos.

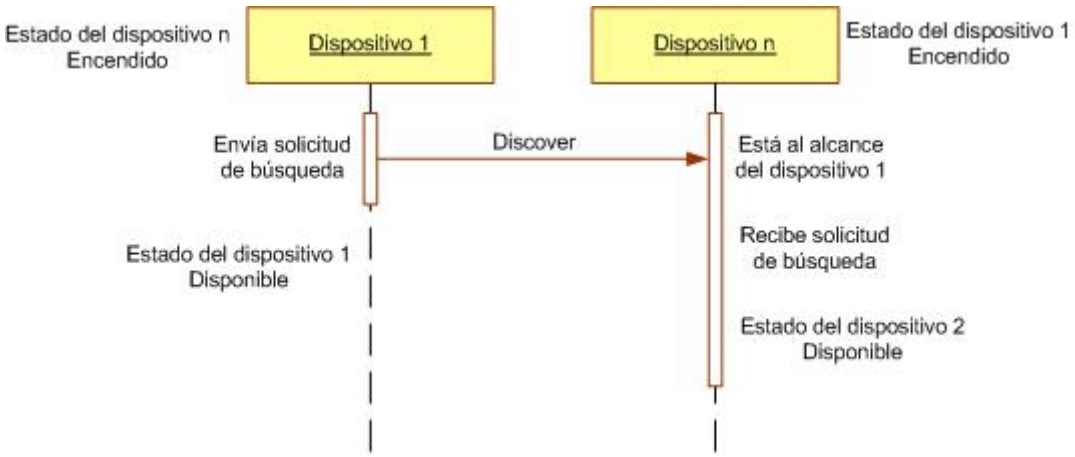

Fig. 24: Inicialización del nodo (Autores, 2013)

3.2.2. Formación del cluster y elección del JC y JCR

Para la formación del cluster, en la solución de la presente tesis, se toma en cuenta el orden de conexión, (ver Fig. 25), como métrica para la asignación del Jefe de Cluster y Jefe de Cluster de Respaldo. Este proceso se realiza mediante un Id para cada nodo, el cual se genera de acuerdo al orden de conexión, todo en torno a la definición del escenario principal que se ha desarrollado en la presente investigación. Es así que el JC será el nodo que haya pedido establecer una conexión con otro nodo, y el JCR el nodo que acepte la solicitud de conexión.

Si un nodo cambia su estado, se comunica, de forma inmediata, a todos sus vecinos a través de un mensaje de broadcast.

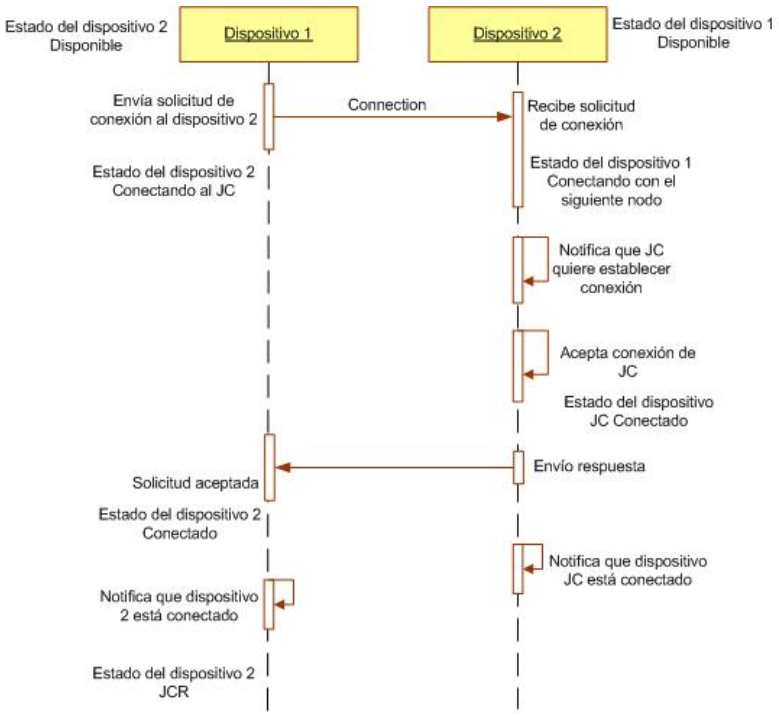

Fig. 25: Formación del cluster (Autores, 2013)

3.2.3. Inclusión de un nodo en el cluster

Si un nodo es incluido en el cluster, a partir de la conexión se almacena la información del nodo quien pasará a tener el estado de N1, significa que en caso de que el JC o el JCR falle el nodo N1 será el siguiente para tomar esos lugares. (Fig. 26).

De acuerdo al escenario planteado, se utilizan tres nodos, teniendo como restricción que para que exista comunicación se necesita al menos dos nodos, es decir el JC y JCR.

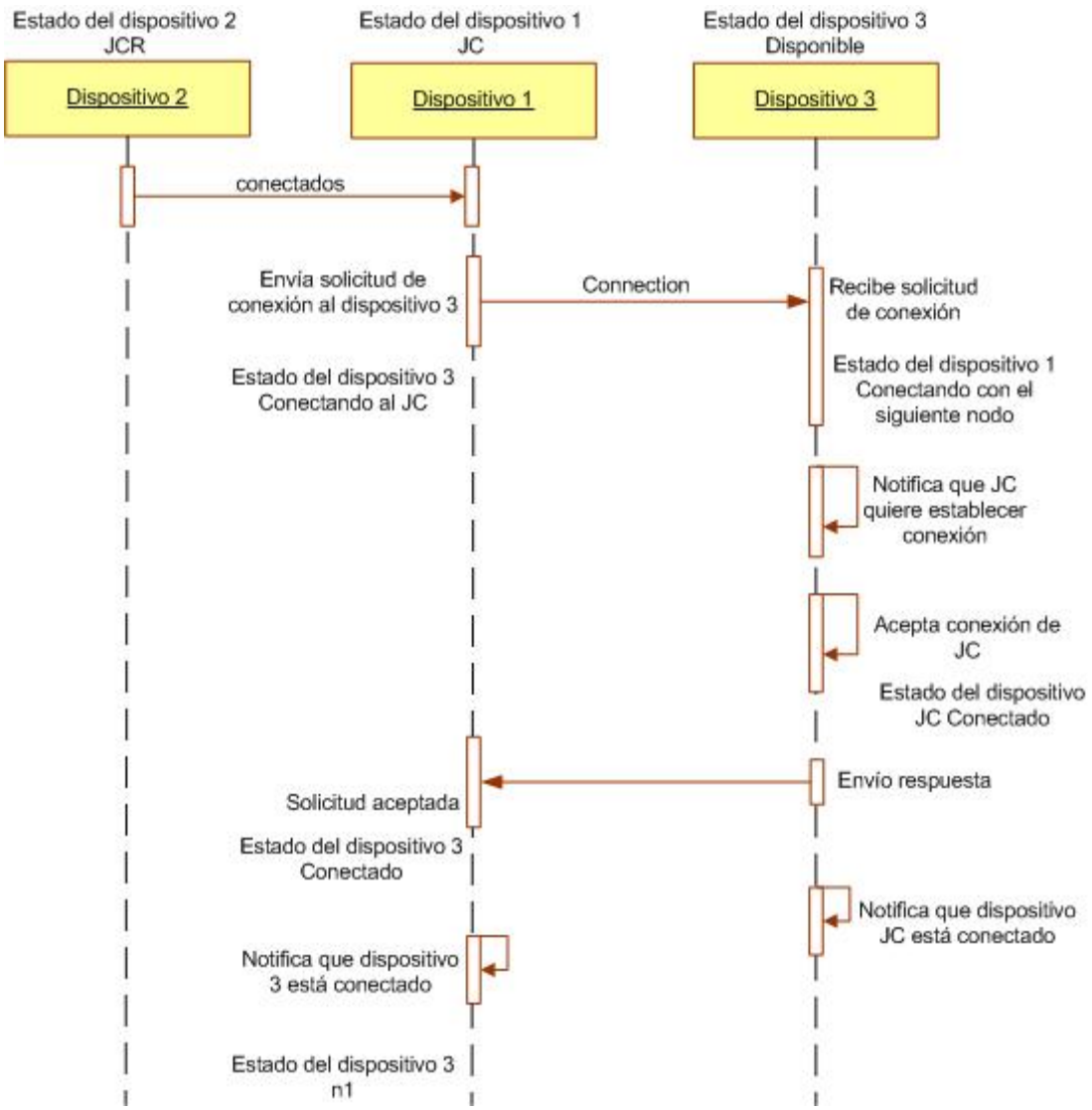

Fig. 26: Inclusión de un nodo en el cluster (Autores, 2013)

3.2.4. Pérdida de Jefe de Cluster

El JC recibe notificaciones del JCR y de los nodos comunes y este envía una respuesta al JCR cada cierto tiempo, cuando el JCR no recibe respuesta de la notificación enviada a su JC cambia automáticamente su estado a Jefe de Cluster e informa a sus nodos inmediatamente a través de un mensaje, de su nuevo estado, este proceso además debe elegir el nuevo Jefe de Cluster de respaldo que sería N1, es decir se lo designa dependiendo del orden de conexión que se tiene. El tiempo de espera para ser JC es de tres segundos (se realiza las pruebas con tres segundos para que no exista demasiada acumulación de datos en los logs y pérdida de eficiencia de la red debido a la cantidad de mensajes de actualización de la información). El flujo correspondiente a esta fase se encuentra en la Fig. 27.

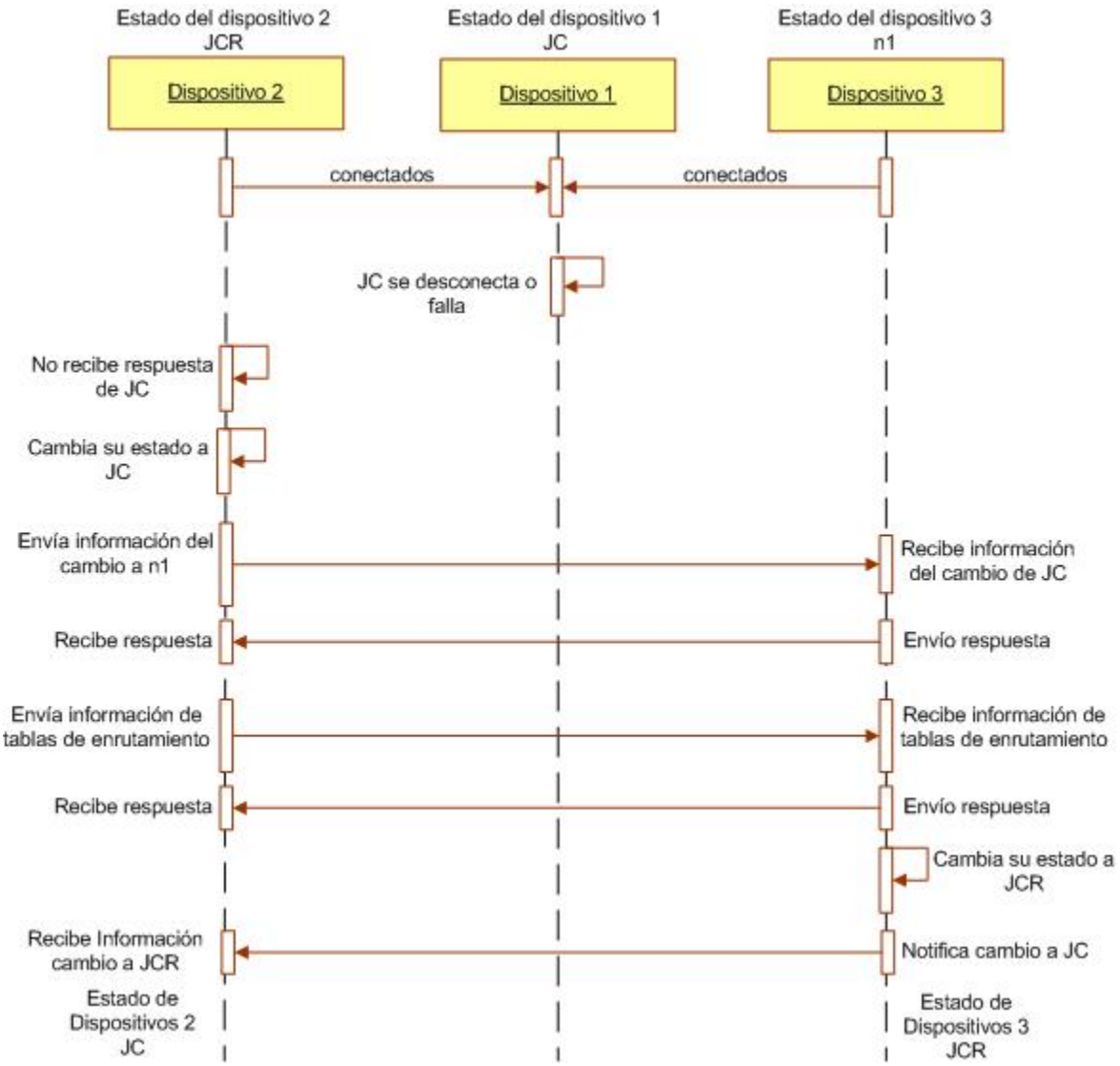

Fig. 27: Pérdida del JC (Autores, 2013)

3.2.5. Pérdida de Jefe de Cluster de Respaldo

Cuando el JC no detecta la presencia del JCR, ya sea porque no ha recibido una actualización de la información de gestión o porque los periodos de actualización han expirado, su tabla de enrutamiento elige el JCR, que será el siguiente nodo con el que estableció una conexión, e informa al nuevo Jefe de Cluster de Respaldo de sus responsabilidades. (Ver Fig. 28)

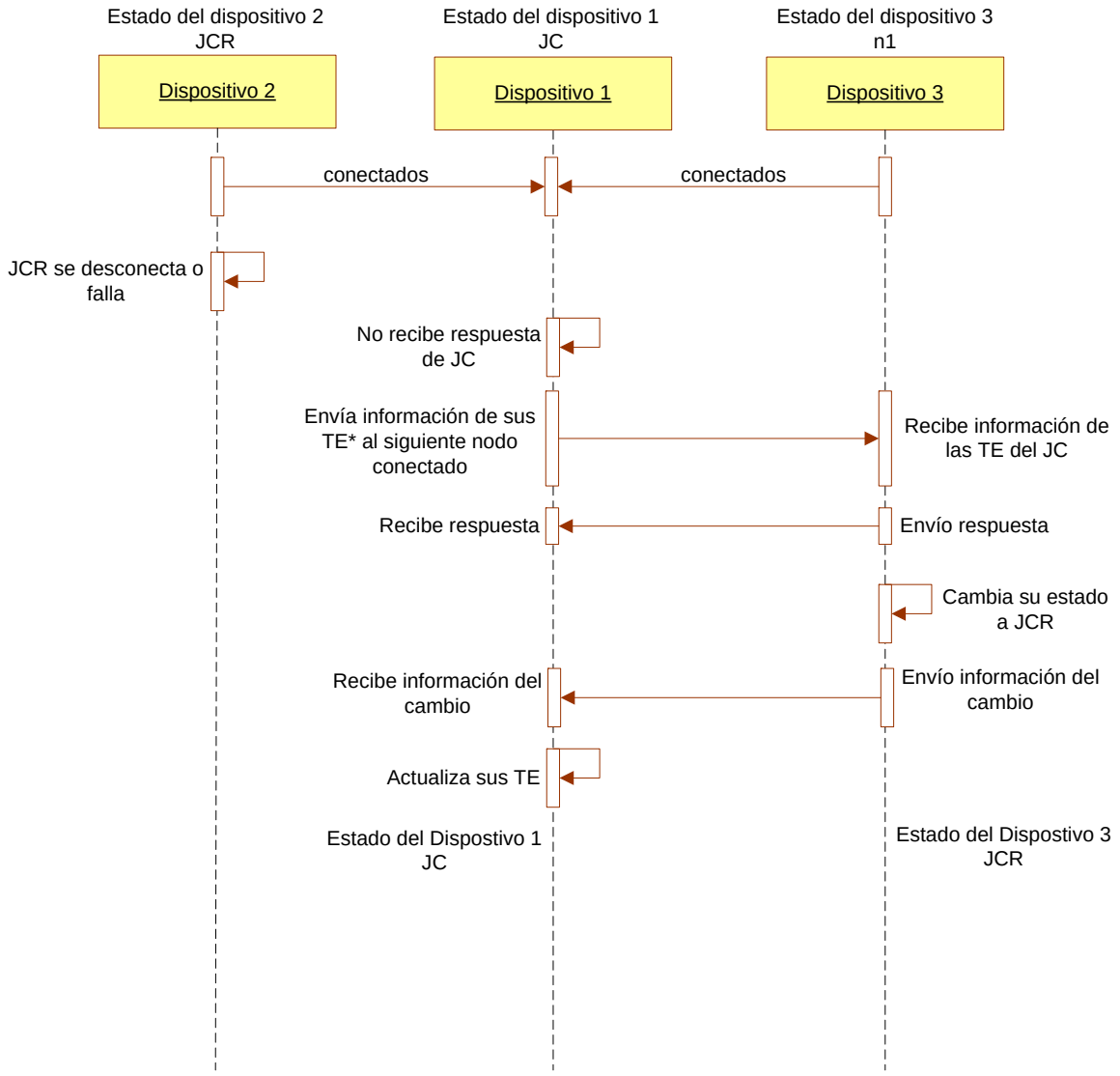

Fig. 28: Pérdida del JCR (Autores, 2013)

*TE= Tablas de enrutamiento

3.2.6. Expiración de temporizador

Este evento es invocado cuando el temporizador ha expirado, en caso de que la entrada expirada sea un JC, el nodo JCR cambia su estado a JC y se elige un nuevo JCR y a través de un mensaje de broadcast se informa al resto de nodos de su cambio de estado. Si la entrada expirada es un JCR el siguiente nodo que se haya conectado pasará a ser JCR, de igual manera se informa al resto de miembros el nuevo estado.

3.2.7. Mantenimiento de las tablas de enrutamiento

Cada vez que se establece la conexión con un dispositivo se extrae la información de ese nodo para agregar una entrada en la base de datos topológica, ya sea para actualizar la información de un nodo conocido o para agregar los datos de un nuevo nodo. Además se le asigna una prioridad según el orden que haya establecido la conexión en el caso de un nuevo dispositivo, para poder establecer su llegada al cluster y establecer cuando puede llegar a ser JCR o JC de ser necesario.

CAPÍTULO IV

IMPLEMENTACIÓN DE BCHP EN DISPOSITIVOS MÓVILES

4. implementación de BCHP

4.1. Implementación del protocolo BCHP en dispositivos Android

Dentro del marco de investigación de la implementación de las características del protocolo BCHP en dispositivos móviles, se proyecta el desarrollo tanto para dispositivos móviles como smartphones y computadores portátiles. Para la parte de los teléfonos celulares se tomará como base los dispositivos móviles inteligentes con sistema operativo Android 4.0 Ice CreamSandwich. (Pomeyrol, 2012).

Se escoge este tipo de dispositivos debido a que en dicha versión del Sistema Operativo se integra capacidad de uso de la tecnología Wi-Fi Direct©, dicha tecnología es una propuesta de comunicación basado en el tipo de redes Ad-Hoc impulsada por la Wi-Fi Alliance, permitiendo en la definición de la misma, la comunicación Ad-Hoc multipunto con dispositivos que tengan la capacidad de Utilizar Wi-Fi Direct©.

Basados en esta premisa se propone modificar el tipo de comunicación que utiliza Wi-Fi Direct© para implementar las características del protocolo BCHP, desarrollando un programa en tecnología java que modifique las librerías de comunicación del Sistema Operativo.

Se analizaron algunas otras opciones para poder implementar las características del protocolo BCHP en smartphones, siendo una de las principales la modificación de la comunicación a nivel de kernel (Morris, 2001), pero se tiene limitantes para su aplicabilidad, siendo la primera el cambio en la licencia que hizo Google© (propietario de Android) al uso del SDK de android, que es el kit de herramientas que permiten crear aplicaciones y realizar modificaciones al kernel o Sistema Operativo de los dispositivos, en dicha modificación a la licencia de uso, se incluye una cláusula que prohíbe el desarrollo de aplicaciones o modificaciones del sistema que deriven en una fragmentación del mismo, lo cual no permite realizar el cambio necesario para plantear esta solución.

Se podría trabajar en Sistemas Android anteriores a la versión 4.0. (Movistar) para evitar la restricción cambio de la licencia en el SDK, pero esto nos trae algunos otros problemas como:

- Teléfonos con Hardware que no soporta el tipo de conexión Ad-Hoc.

Se necesita permisos especiales (root) en los dispositivos que permitan realizar los cambios necesarios, y este es un procedimiento que no se utiliza normalmente en los teléfonos debido al peligro que se corre de dañar el Sistema Operativo del dispositivo, o de perder la garantía de fábrica del equipo. (Google Project, 2011)

Por lo tanto la opción que se utiliza es la de desarrollar una aplicación que interactúe con Wi-Fi Direct© para la implementación de las características del protocolo BCHP.

4.1.1. Desarrollo en Android

En el desarrollo de la solución se procede a realizar la codificación del software que interactúa con la tecnología Wi-Fi Direct© que permite implementar las características del protocolo BCHP.

El escenario planteado se da con tres dispositivos, mismos que fueron prestados por la UTPL a través del departamento de Ciencias de la Computación y Electrónica los cuales son:

Google Nexus (**¡Error! No se encuentra el origen de la referencia.**)

Tabla 8: Características Google Nexus (Android, 2012)

Fabricante	Samsung
Memoria	RAM 1024MB
Almacenamiento Interno	16000MB
Conectividad	GSM/GPRS/EDGE 850/900/1800/1900 UTMS HSDPA 850/1900/2100 Wi-Fi 802.11 a/b/g/n
Sistema Operativo	Android 4.0 (Ice Cream Sandwich)
Hardware	1.2 Ghz Dual-core

Se desarrolla un prototipo (Fig. 29), en el cual se presenta una interfaz, que muestra que el dispositivo se ha inicializado, y está en la búsqueda de algún dispositivo o clúster dentro del alcance (10 metros aprox.), de encontrar otros dispositivos, se presenta la información básica del mismo (Nombre del Dispositivo), con la opción de comunicarse con el mismo y formar un cluster, si en cambio lo que encuentra es un cluster formado, nos muestra la opción de unirse al mismo.

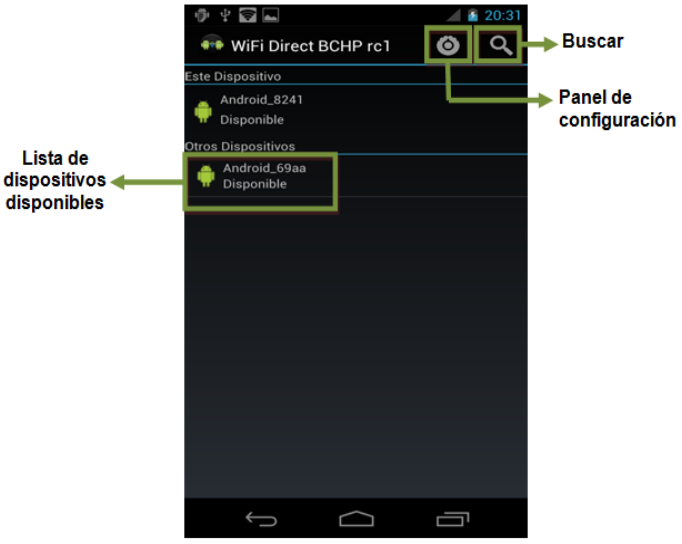

Fig. 29: Interfaz Android – BCHP (Autores, 2013)

4.2. Implementación del protocolo BCHP en dispositivos portátiles

Para la implementación de BCHP en dispositivos portátiles se ha establecido una serie de pasos necesarios para desarrollar con éxito la implementación como se muestra a continuación:

Instalación de Ubuntu: El sistema operativo utilizado para la implementación de BCHP es Ubunto 10.04 que posee las características necesarias para la ejecución de este protocolo.

Instalación de controladores de la tarjeta inalámbrica: Se debe verificar que la tarjeta inalámbrica se encuentre correctamente instalada, si no es así se debe revisar las configuraciones según el tipo de dispositivo y realizar la instalación.

Instalación de GCC: GCC (GNU Compiler Collection) es un compilador integrado del proyecto GNU para C, C++, Objective C y Fortran; es capaz de recibir un programa fuente en cualquiera de estos lenguajes y generar un programa ejecutable binario en el lenguaje de la máquina donde ha de correr. Este compilador nos permite crear el ejecutable del protocolo para que funcione en los dispositivos móviles.

El código de la aplicación se encuentra detallado en el Anexo 1.

Instalación del protocolo BCHP: La instalación del protocolo se hace a través de un ejecutable que permite al dispositivo móvil poder manejar la conexión de los dispositivos a través de BCHP. A continuación se muestran las funciones que debe cumplir este protocolo:

4.2.1. Inicialización de los nodos

La primera fase del protocolo BCHP es la inicialización de los nodos, para que esto ocurra, una vez codificado el protocolo (Ver Fig. 32), se procede a realizar algunas configuraciones para que el dispositivo portátil pueda trabajar con BCHP, en la Fig. 30 se muestran las líneas de comandos que se deben realizar para comenzar a trabajar, y en la **¡Error! No se encuentra el origen de la referencia.** se describe cada uno de ellos. Primeramente se desactiva la interfaz wlan0. Se activa el modo Ad-Hoc, se trabaja en el canal 11, (debido a que en el entorno de trabajo en el que se desarrolla el proyecto, las redes adyacentes trabajaban en una diversidad de canales desde el 1 hasta el 9, se escoge el canal 11 porque no se encuentra utilizado en nuestro entorno, sin embargo se puede utilizar cualquier canal del 1 al 14 dependiendo su disponibilidad. Seguidamente con estas indicaciones se puede levantar la interfaz wlan.

Se crea una red Ad-Hoc denominada “AdHocBCHP” y se le asigna una, el archivo en donde se almacena la información de enrutamiento es ip_forward para lo cual se verifica si se tiene permisos de escritura.

Se ejecuta el demonio BCHP para poder iniciar la comunicación bajo este protocolo que indica que se va a almacenar un log de la ejecución, verifica el modo del demonio, el temporizador se lo establece en tres segundos, es decir cada tres segundos las tablas de enrutamiento se deben actualizar y se trabaja con la interfaz wlan0.

Tabla 9: Configuraciones BCHP

COMANDO	DESCRIPCIÓN	
ifconfig wlan0 down	Desactiva la interfaz wlan0	
iwconfig wlan0 mode ad-hoc	Activa el modo Ad-Hoc	
iwconfig wlan0 cannel 11	Trabaja en el canal 11	
ifconfig wlan0 up	Levanta la interfaz wlan0	
iwconfig wlan0 essid “AdHocBCHP”	Crea una red Ad-Hoc	
iwconfig wlan0 10.10.10.x	Asignación de una IP al nodo	
echo >1 /proc/sys/net/ipv4/ip_forward	Verifica permisos de escritura	
modprobe kbchp	Ejecuta el demonio BCHP	
bchpd –l –D –r 3 –i wlan0	-l	Almacena en un log la ejecución de BCHP
	-D	Verifica el modo del demonio
	-r3	Temporizador (3 segundos)
	-i wlan0	Interfaz utilizada

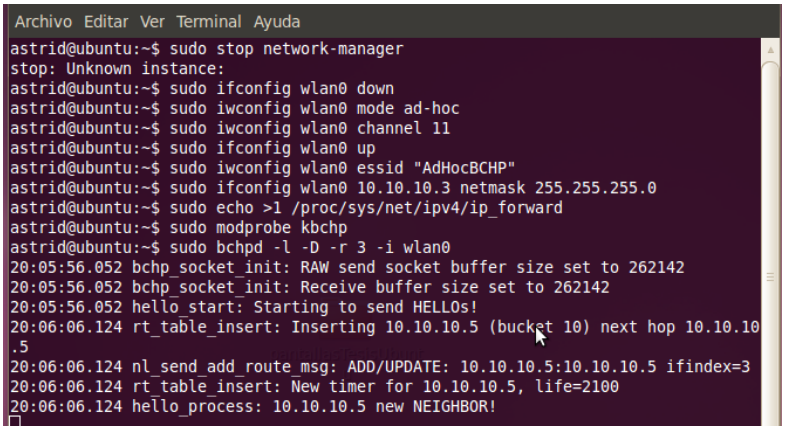

Fig. 30: Inicialización de los nodos en Ubuntu (Autores, 2013)

4.2.2. Formación del cluster y elección del JC y JCR

Cuando un nodo inicia, posee un estado UNDECIDED, debido a que no pertenece a ningún cluster, se identifica en la red a través del envió de mensajes de difusión con su IP y su estado, esto permite que se actualicen las tablas de enrutamiento de los nodos presentes.

Dentro del modelo especificado en este trabajo de investigación se ha planteado que el nodo que será elegido JC, sea el primer nodo en conectarse, cambiando su estado a CLUSTER HEAD, de esta forma al enviar en broadcast su información, todos los nodos del cluster lo reconocen como JC, y el próximo nodo en formar parte del cluster se reconoce como JCR.

En la Fig. 31 se observa la clase Main del protocolo, donde se encuentran las clases a las que se hará referencia directa para el funcionamiento del protocolo (bchp_hello,bchp_neigboor, routing_table), los mismos que se abordaran más adelante.

Se identifican también las clases que permiten cambiar el estado de los nodos para poder identificarlos como JC y JCR (set_cluster_head(), set_backup_cluster_head()).

.

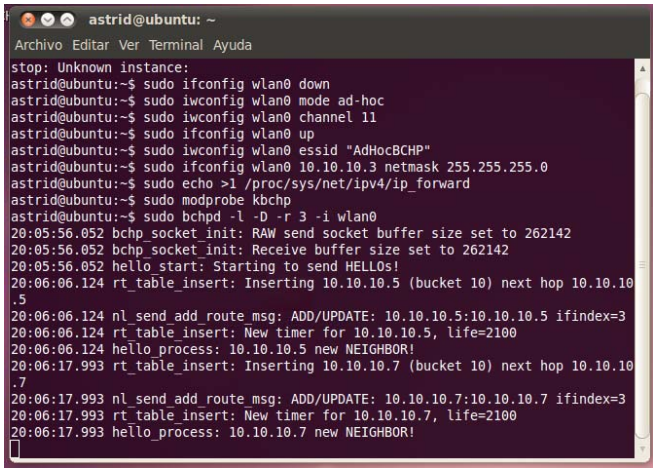

Fig. 31: Nodos vecinos (Autores, 2013)

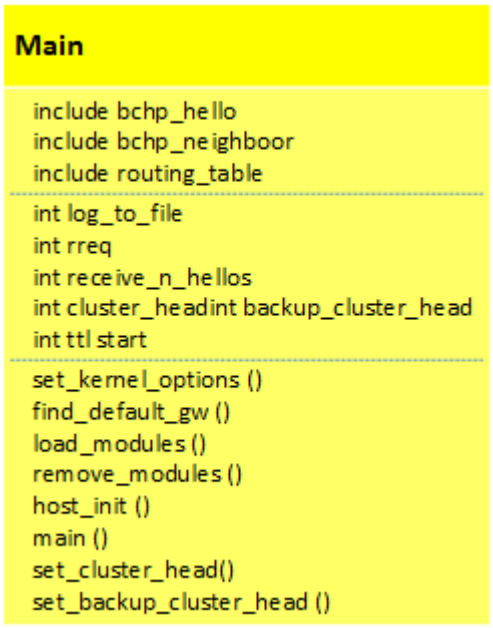

Fig. 32: Main BCHP (Autores, 2013)

4.2.3. Inclusión de un nodo en el cluster

Una vez formado el cluster y determinado el JC y JCR, cuando un nodo entra en cluster debe anunciarse, a través de los mensajes HELLO, esto permite que se lo identifique en las tablas de enrutamiento y conozca la existencia de los nodos designados previamente como JC y JCR y el estado que le corresponde es MEMBER.

Una vez establecido el estado, a través de la clase bchp_neighbor (Fig. 33), se lo puede incluir en la comunicación del cluster, haciendo uso de las funciones de la clase routing_table (Fig. 34) se mantiene actualizada la información de estado y forma de llegar a él.

Cabe destacar que estas clases interactúan con todos los nodos del cluster, permitiendo la actualización de su información de Ip y estado en funcionamiento, para las pruebas del mismo se ha trabajado con un tiempo de 3 segundos (sección 4.2), esto reconoce cuales dispositivos están aún dentro del clúster y cuales ya no lo son.

Al darse el escenario en que uno de los nodos no envíe su información actualizada se procede con los procesos de cambio de estado para poder mantener la comunicación dentro del clúster, mismos que se detallan adelante.

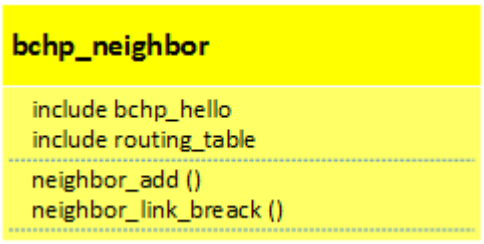

Fig. 33: Vecinos BCHP (Autores, 2013)

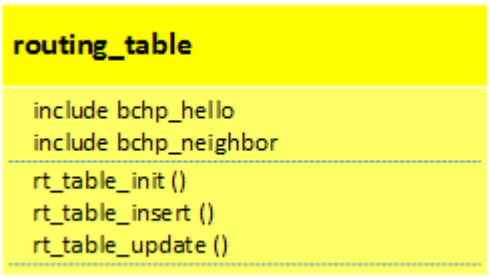

Fig. 34: BCHP tabla de enrutamiento (Autores, 2013)

4.2.4. Pérdida del Jefe de Cluster

Si el Jefe de Cluster por alguna razón deja de funcionar, el Jefe de Cluster de Respaldo pasa a tomar su lugar, de tal manera que los demás dispositivos conectados no sufren ningún inconveniente y puedan seguir enviado mensajes con el resto de dispositivos conectados. El método que realiza esta función se observa en la Fig. 35.

4.2.5. Pérdida del Jefe de Cluster de Respaldo

Dentro del funcionamiento normal del enrutamiento del protocolo, puede darse el escenario en donde el nodo designado como JCR no se encuentre activo o deje de responder repentinamente, esto es detectado cuando el JC no recibe actualizaciones de parte del nodo JCR dentro de los periodos de actualización de su tabla de enrutamiento, cuando esto sucede, el siguiente nodo que se haya conectado al clúster es designado como JCR.

En la Fig. 35 podemos ver la clase bchp_Agent que ejecuta los temporizadores de actualización de las tablas de enrutamiento, los mismos que permiten identificar la respuesta accesible o inalcanzable de un nodo y poder determinar la mejor opción ante esta situación.

4.2.6. Expiración del Temporizador

La actualización de las tablas de enrutamiento se hace mediante el uso de temporizadores, estos permiten manejar el tiempo de vida de las entradas de las tablas de enrutamiento. El funcionamiento implica el uso de disparadores, que sirven de indicadores al Jefe de Cluster para cambiar de ruta, actualizar el estado de un nodo y elegir un nuevo JCR, cabe destacar que el mismo proceso es el que permite elegir al JC gracias a la alerta del temporizador.

La clase bchp_Agent (Fig. 35) tiene incluido los métodos necesarios para activar los temporizadores y manejar el protocolo. Esta clase se conjuga con otros métodos y clases para poder ofrecer las características propias del protocolo BCHP, debido a esto se puede reaccionar ante circunstancias como la pérdida del Jefe de Clúster de Respaldo.

Además la Fig. 35 está compuesta por los diferentes diagramas que representan a los métodos de manejo de las tablas de vecinos, los cuales nos permiten identificar nuestro entorno en el clúster, poder tomar decisiones como la elección de JC o JCR, además de alimentar las tablas de ruteo del cluster, estos métodos toman en consideración todos los escenarios posibles que se pueden presentar en la vida real.

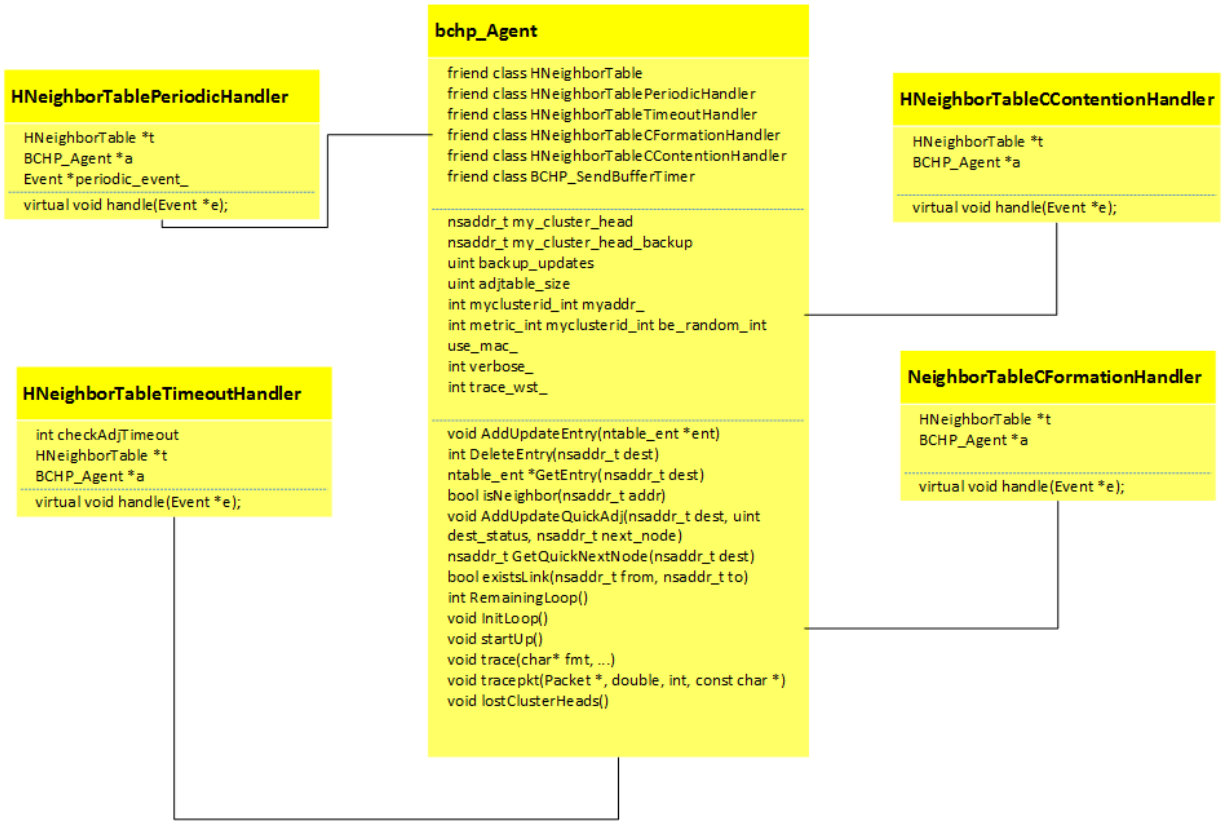

Fig. 35: Agente BCHP (Autores, 2013)

4.2.7. Mantenimiento de Tablas de Enrutamiento

Las tablas de enrutamiento permiten identificar las rutas de comunicación entre los nodos, y estas se alimentan cada vez que un mensaje o paquete llega a su destino, extrayendo información como el estado del nodo y los saltos necesarios para poder llegar a su destino.

Dependiendo del origen del nodo, esta entrada es introducida o no en la tabla de rutas, además de introducir su estado. (Fig. 36).

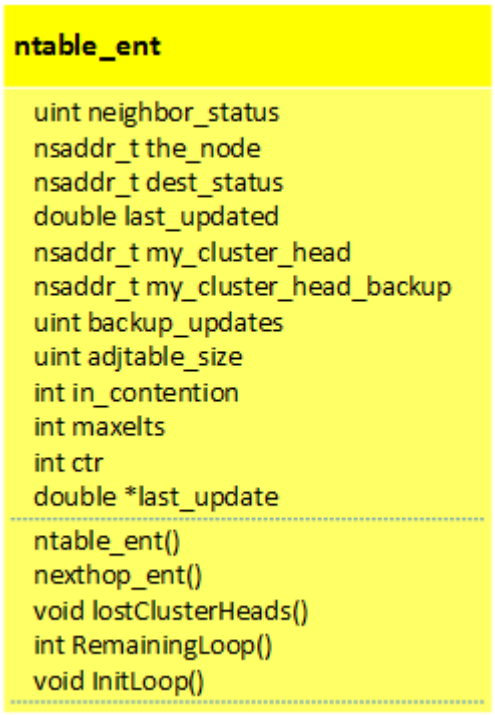

Fig. 36: Mantenimiento de rutas (Autores, 2013)

CAPÍTULO VI

PRUEBAS DE FUNCIONALIDAD DE BCHP

5. Pruebas

En el siguiente capítulo se muestra las pruebas realizadas para comprobar la implementación del protocolo en los dispositivos móviles.

5.1. Pruebas de la solución en Android

Se plantea como objetivos de las pruebas con la aplicación algunas características del protocolo BCHP, los cuales permiten verificar la eficacia de la solución planteada, los puntos evaluados y sus resultados se resumen en el cuadro (**¡Error! No se encuentra el origen de la referencia.**) a continuación.

Tabla 10: Pruebas Android BCHP

Objetivo de Prueba	Resultado	Observación
Inicialización del Nodo	✓	
Búsqueda de Dispositivos/clusters adyacentes	✓	
Elección de Jefe de Cluster	✓	
Elección de Jefe de Cluster de Respaldo	✗	Esta característica es primordial del Protocolo BCHP, sin la cual no podemos probar el protocolo
Comunicación entre dos dispositivos	✓	
Comunicación entre tres dispositivos	✗	No se pudo mantener conexión simultánea con los tres dispositivos
Sucesión de Estados de los nodos	✗	Al no tener comunicación entre al menos tres nodos y no poder fijar el estado de JCR, no se puede superar este objetivo.

De los objetivos planteados en las pruebas anteriormente, se detalla a continuación algunos resultados obtenidos con las gráficas respectivas, en la Fig. 37 se muestra la interfaz en donde se muestra el listado de todos los dispositivos disponibles dentro del alcance de Wi-Fi Direct BCHP y las opciones al escoger uno de ellos (Conectar y Desconectar) para establecer una conexión como en la Fig. 38. Una vez establecida la conexión el estado del dispositivo cambia a Invitado (ver Fig. 39) y se puede dar comienzo a la transferencia de información entre estos dispositivos, en la Fig. 40 se puede observar la interfaz, sin ningún dispositivo disponible.

Fig. 37: Listado de dispositivos disponibles (Autores, 2013)

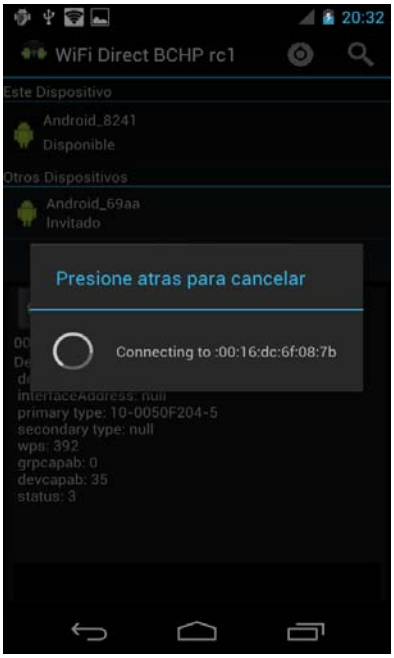

Fig. 38: Estableciendo conexión Wi-Fi Direct BCHP(Autores, 2013)

Fig. 39: Conexión Establecida Wi-Fi Direct BCHP (Autores, 2013)

Fig. 40: Dispositivos no Disponibles (Autores, 2013)

5.1.1. Resultados Obtenidos en Android

Luego de analizar las pruebas realizadas se pudo concluir que la solución planteada no cubrió los escenarios establecidos, y que se debe considerar que existen una gran cantidad de modelos de teléfonos celulares con diferentes características de hardware en el mercado mundial, lo que dificultaría aún más el poder obtener un conjunto de características estándar para poder implementar una solución que funcione correctamente con la mayoría de modelos.

La tecnología de Wi-Fi Direct© se ha certificado para permitir la comunicación Ad-Hoc multipunto en los dispositivos certificados con la misma, pero dicha tecnología solo se ha implementada parcialmente en los teléfonos de la generación actual, lo cual no permite poder implementar el protocolo de comunicación BCHP actualmente.

5.2. Pruebas de la solución en Ubuntu

Para las pruebas con BCHP en Ubuntu se ha creado una aplicación en lenguaje Java que muestra de forma gráfica la conexión entre los tres computadores portátiles. Además se realizan pruebas de envío de paquetes con Wireshark. La comprobación se realiza con diferentes escenarios, que permiten verificar el comportamiento de los dispositivos con el protocolo.

Como primer escenario se comprueba el levantamiento de la red ad-hoc y la corrida del demonio del protocolo.

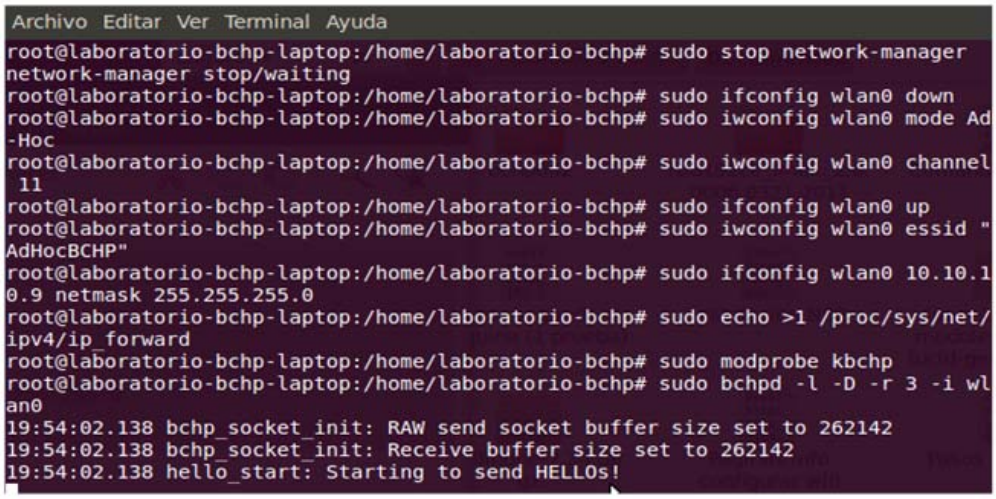

Fig. 41: Levantamiento de BCHP (dispositivo – laboratorio-bchp) (Autores, 2013)

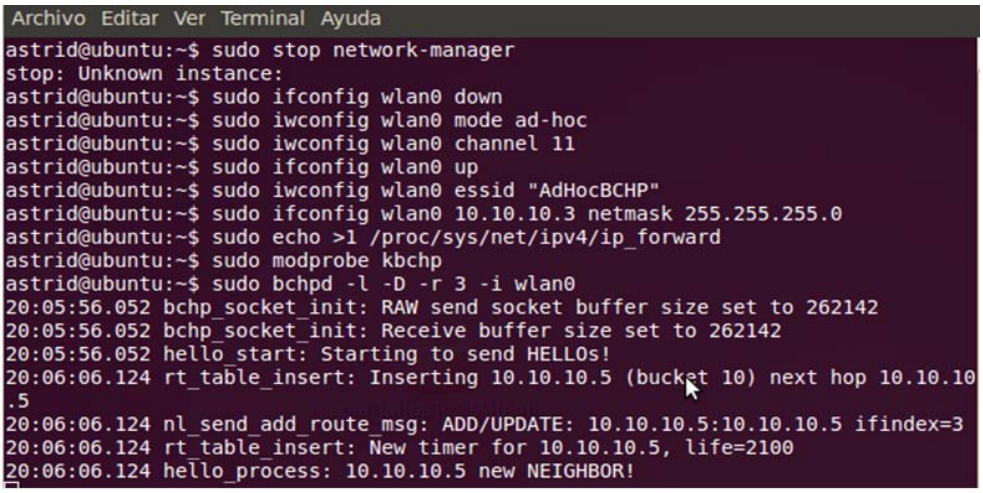

Fig. 42: Levantamiento de BCHP (dispositivo - astrid) (Autores, 2013)

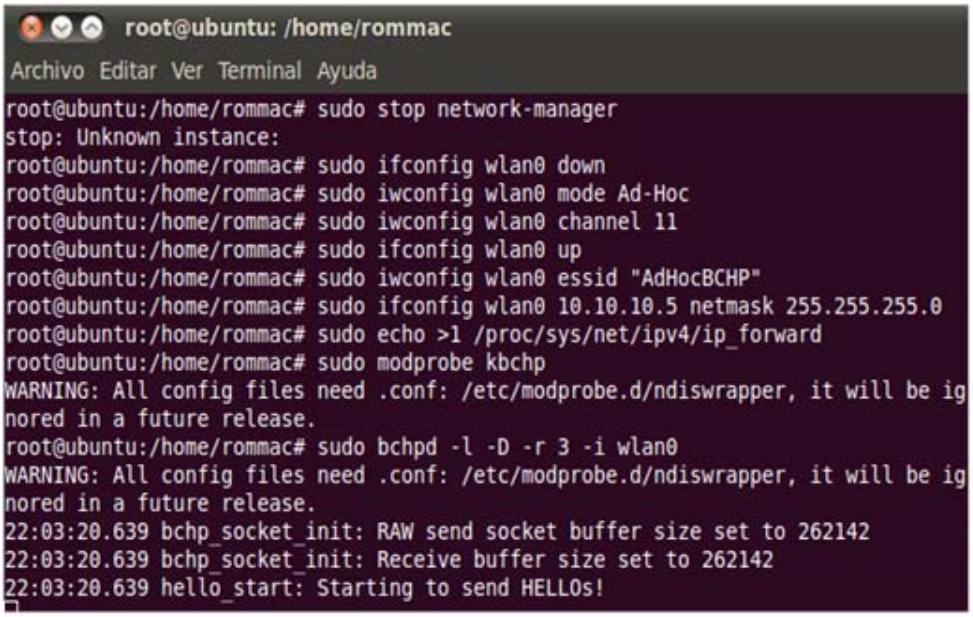

Fig. 43: Levantamiento de BCHP (dispositivo - rommac) (Autores, 2013)

Una vez levantado el protocolo en las tres máquinas (Fig. 41, Fig. 42 y Fig. 43) y asignada la IP a cada una (Tabla 11), se procede a hacer pruebas de conectividad mediante el envió de solicitudes ICMP Ping, las cuales resultan exitosas, como se muestra a continuación:

Tabla 11: Asignación IPs

Dispositivo	Dirección IP
astrid (Jefe de Cluster)	10.10.10.3
rommac (Jefe de Cluster de Respaldo)	10.10.10.5

laboratorio-bchp (Nodo Comun)	10.10.10.9

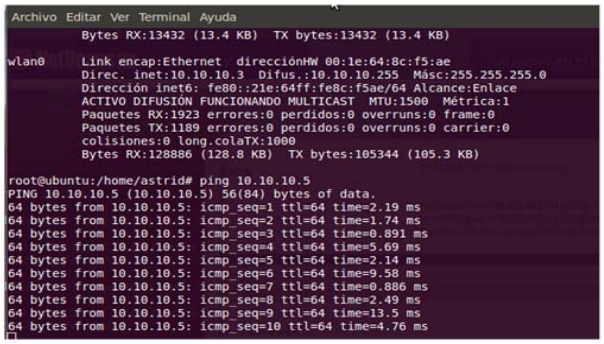

Fig. 44: Ping de 10.10.10.3 a 10.10.10.5 (Autores, 2013)

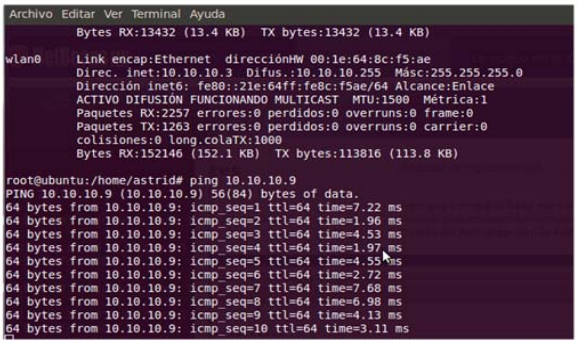

Fig. 45: Ping de 10.10.10.3 a 10.10.10.9 (Autores, 2013)

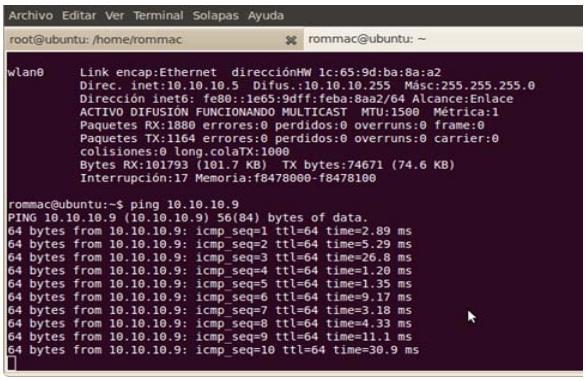

Fig. 46: Ping de 10.10.10.5 a 10.10.10.9 (Autores, 2013)

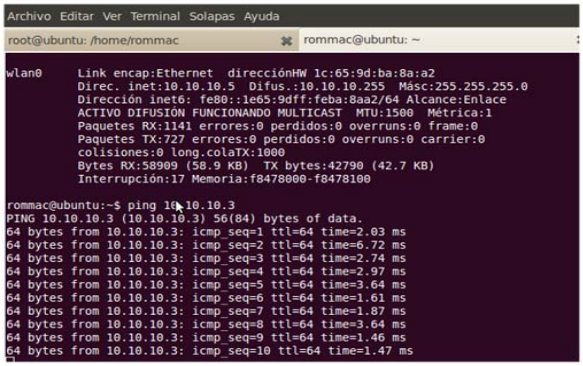

Fig. 47: Ping de 10.10.10.5 a 10.10.10.3 (Autores, 2013)

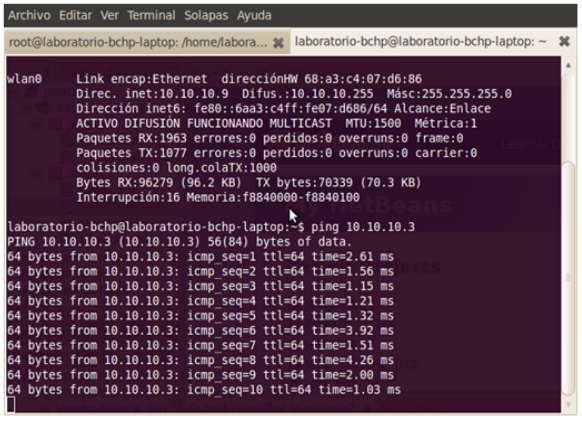

Fig. 48: Ping de 10.10.10.9 a 10.10.10.3 (Autores, 2013)

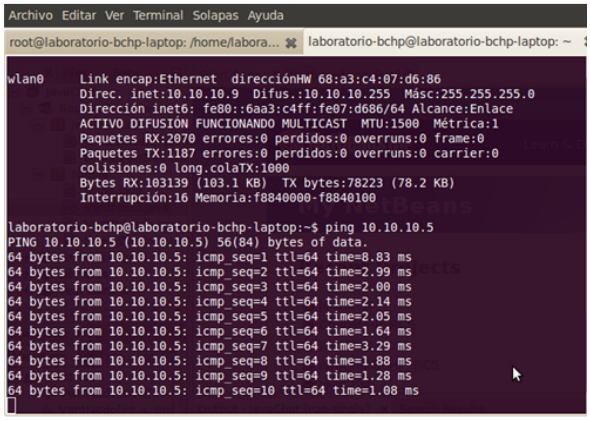

Fig. 49: Ping de 10.10.10.9 a 10.10.10.5 (Autores, 2013)

De la **¡Error! No se encuentra el origen de la referencia.** a la Fig. 49 se observa que los tres dispositivos han establecido conexión entre todos.

Una vez terminadas las pruebas a nivel de consola, se procede a probar la aplicación "Java Chat BCHP", el código de esta aplicación se encuentra en el Anexo 2, la misma que permite enviar mensajes entre los nodos del cluster formado, además de integrar un sistema de cuadros de alerta cuando existe un cambio en los nodos y las funciones que están desempeñando en el esquema del cluster.

5.2.1. Resultados Obtenidos en Ubuntu con Java Chat BCHP

5.2.1.1. Inicialización del nodo

Para inicializar la conexión en la aplicación (el protocolo BCHP ya debe estar configurado y ejecutado en cada uno de los dispositivos portátiles, tal como se muestra en la sección anterior), el primer nodo en conectarse es el Jefe de Cluster, en el cual aparece las siguientes pantallas (Fig. 50), la de gestión de mensajes, y una consola de registro de actividad, que indica al JC la interacción que tenga con otros dispositivos.

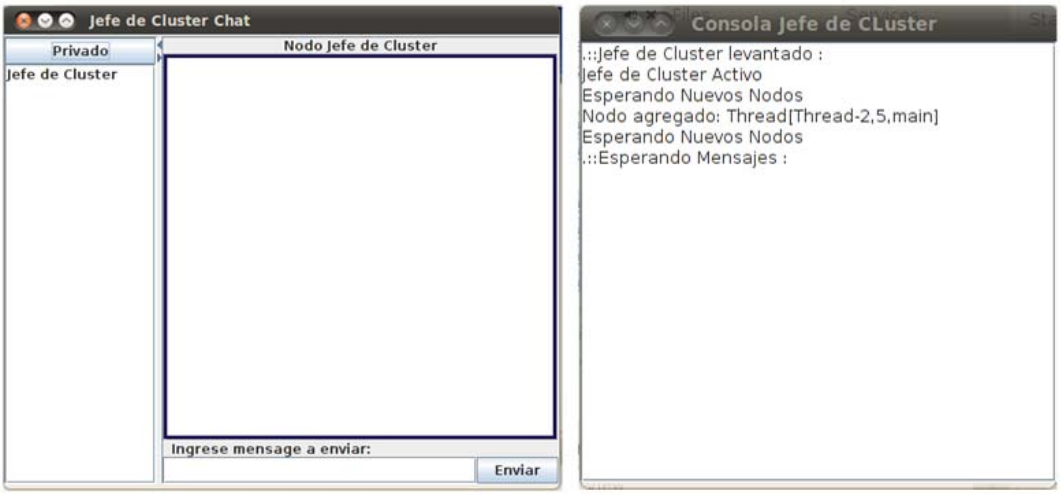

Fig. 50: Interfaz JC - Java Chat BCHP (Autores, 2013)

Los demás nodos al iniciar la aplicación deben introducir los datos de configuración acorde al esquema de red configurado hasta ahora, teniendo que ingresar la IP del Jefe de Cluster (Fig. 51) previamente levantado y además identificarse con un Nick (Fig. 52).

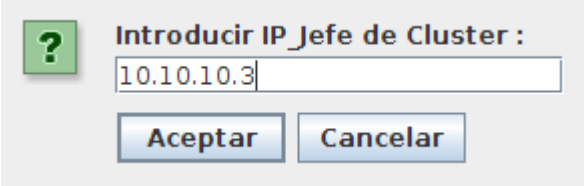

Fig. 51: Introducir IP del JC (Autores, 2013)

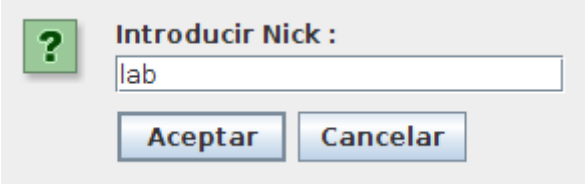

Fig. 52: Introducir Nick (Autores, 2013)

5.2.1.2. Formación del cluster y elección del JC y JCR

La formación del cluster y elección del JC y JCR se realiza de manera interna como se explica en la sección 4.2.2.

5.2.1.3. Inclusión de un nodo en el cluster

Como se muestra en la Fig. 53 al momento de empezar con la conexión se crea una ventana que permite ver la lista de usuarios o dispositivos conectados, con los que se puede intercambiar información entre todos o escoger un dispositivo concreto y enviar un mensaje. El Jefe de Cluster tiene una consola que permite ver los mensajes y los nuevos nodos conectados con información más específica. (Fig. 54).

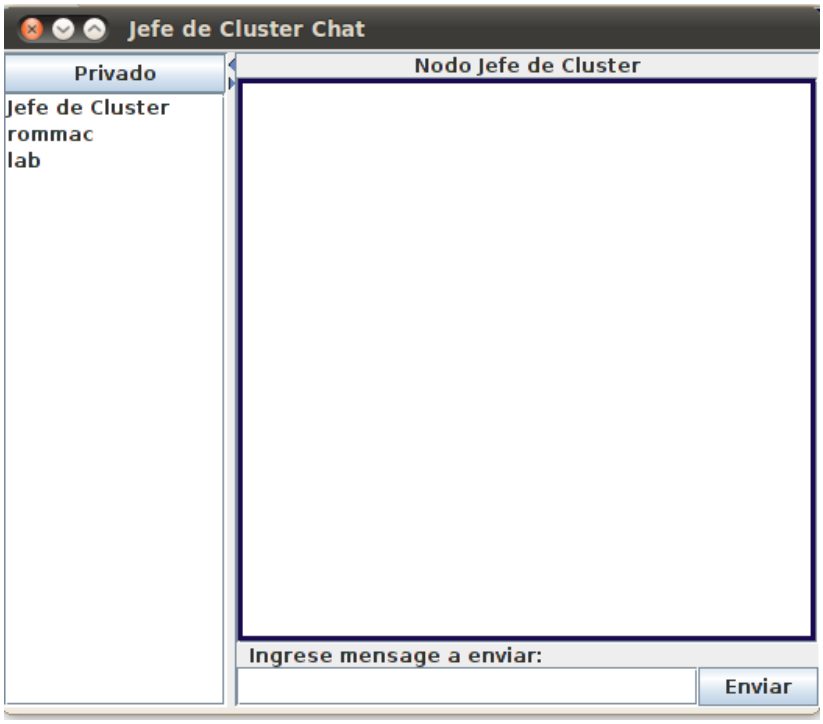

Fig. 53: Lista de dispositivos conectados (Autores, 2013)

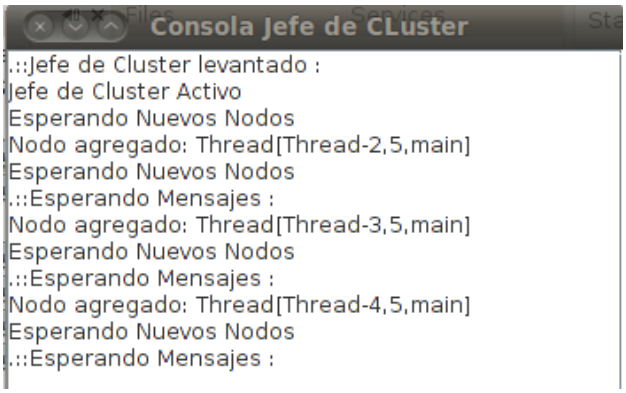

Fig. 54: Consola del JC (Autores, 2013)

5.2.1.4. Pérdida del Jefe de Cluster

Si el Jefe de Cluster pierde conexión, la aplicación Java Chat BCHP, muestra un mensaje (Ver Fig. 55) con el cambio del JC por el JCR a los nodos que aún se encuentren conectados dentro del cluster. Esta prueba de pérdida del JC se dio en un ambiente de movilidad superior a los 25 metros aproximadamente.

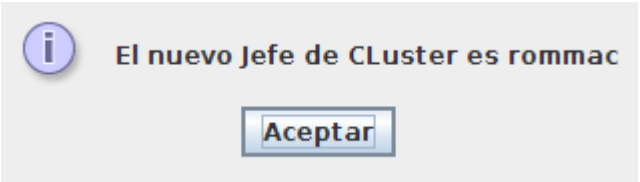

Fig. 55: Nuevo JC (Autores, 2013)

5.2.1.5. Pérdida del Jefe de Cluster de Respaldo

Si el Jefe de Cluster de Respaldo pierde conexión, el nodo siguiente, pasa a tomar su lugar, y se levanta se envía una notificación a los nodos que aún se encuentren conectados en el cluster (Fig. 56).

Fig. 56: Nuevo JCR (Autores, 2013)

5.2.1.6. Expiración del Temporizador

El tiempo de espera de los temporizadores es ajustable al momento de inicializar el protocolo, con el fin de mejorar las prestaciones en escenarios diferentes. En las pruebas realizadas con la aplicación Java Chat BCHP se toma como base un tiempo de tres segundos.

5.2.1.7. Mantenimiento de las tablas de enrutamiento

El mantenimiento de las tablas de enrutamiento se realiza en base a los tiempos de la expiración de los temporizadores, es decir, cada tres segundos se realiza una verificación y actualización de la comunicación entre los dispositivos y sus direcciones IP.

5.3. Captura de paquetes de BCHP en Wireshark

La *Fig. 57* se observa los primeros paquetes que se envían desde el nodo que actuará como Jefe de Clúster, en el que se informa que se ha conectado a la red un nodo, y que estará en modo escucha de nuevos nodos. Este paquete posee una dirección origen y como destino esta broadcast, lo cual nos indica que se envió a toda la red de manera general.

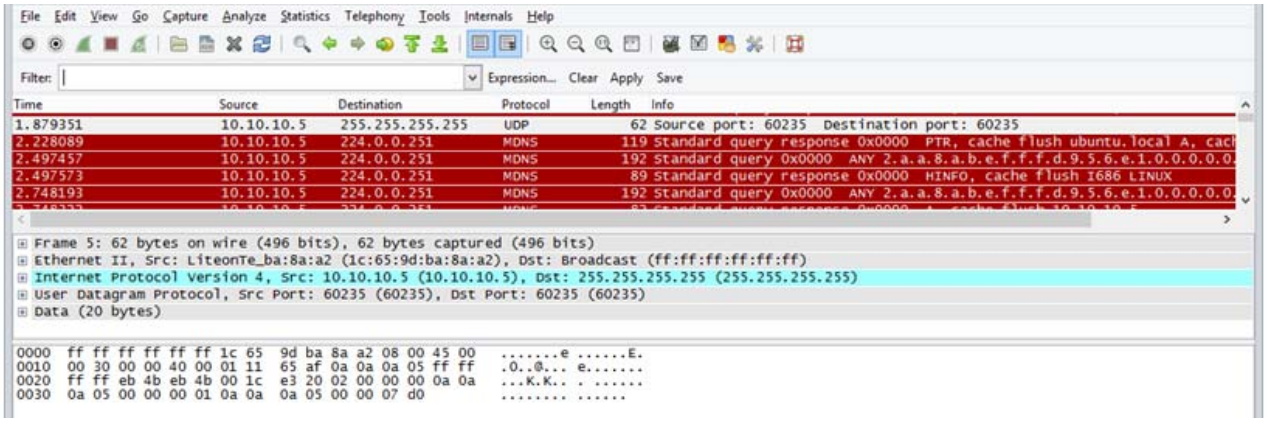

Fig. 57: Primeros paquetes enviados (Autores, 2013)

El paquete de la Fig. 58 es del tipo MDNS. Permite identificar localmente una máquina, y como muestra de esto tenemos, como origen, la dirección 10.10.10.5, que se asigna previamente a la interfaz inalámbrica del nodo en la fase de configuración del protocolo (Revisar sección 4.2) y teniendo como destino la dirección 224.0.0.251 (Multicast DNS) para poder resolver el nombre de dominio local, que en estos casos terminara en .local para su identificación.

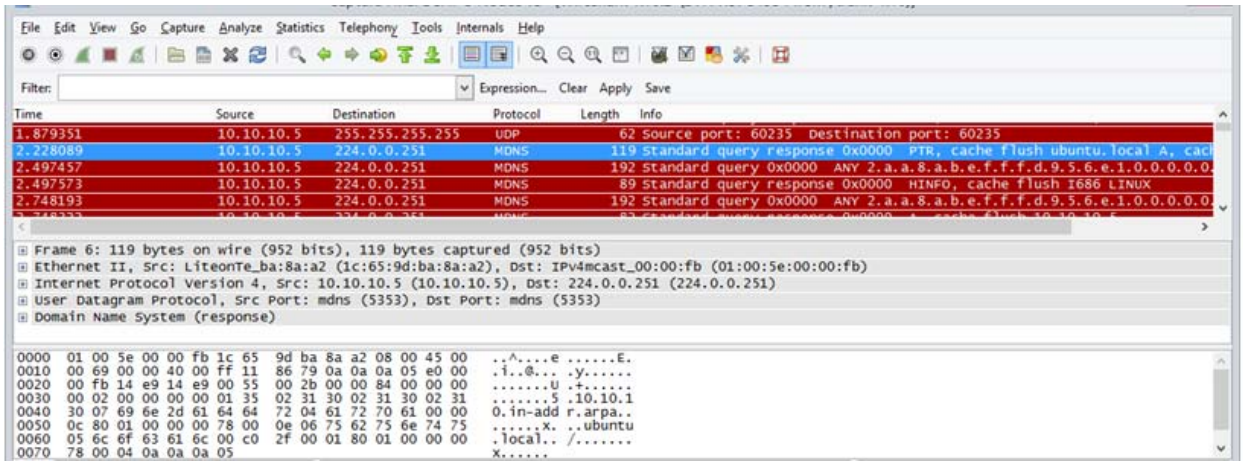

Fig. 58: Paquetes MDNS (Autores, 2013)

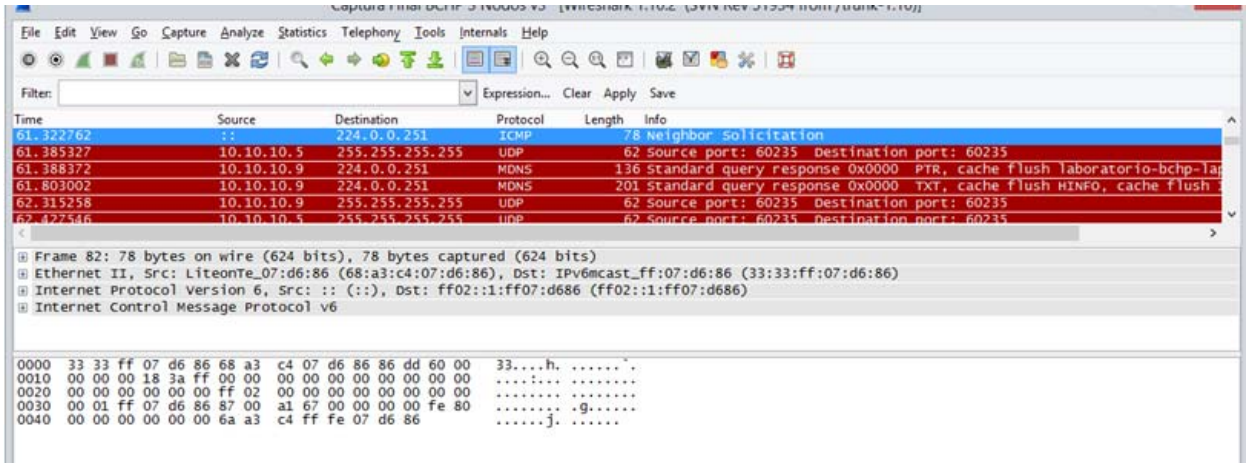

Fig. 59: Búsqueda de Nodos (Autores, 2013)

Siguiendo con el proceso de comunicación, el nodo empieza a buscar más nodos conectados, verifica si existe algún dispositivo en enrutamiento dentro del segmento de red, lo hace repetidamente para poder actualizar las tablas de enrutamiento como se muestra en la Fig. 59, para las pruebas del proyecto de tesis se ha establecido un valor de 3 segundos, una vez comprobado que no existe ningún nodo conectado procede a declararse Jefe de Cluster, a la espera de próximos nodos que se conecten dentro de la red.

En la Fig. 60 se observa el mensaje de pertenencia a un grupo que emite el nodo JC (10.10.10.9) a través del protocolo IGMPv3, en el cual se especifica que en el segmento de red, ya existe un grupo de asociación, esto lo pueden identificar todos los nuevos nodos que se conecten a la misma. El puerto que se utiliza para implementar BCHP es el 60235, debido a que este puerto no está siendo utilizado por ninguna aplicación, permite su libre utilización.

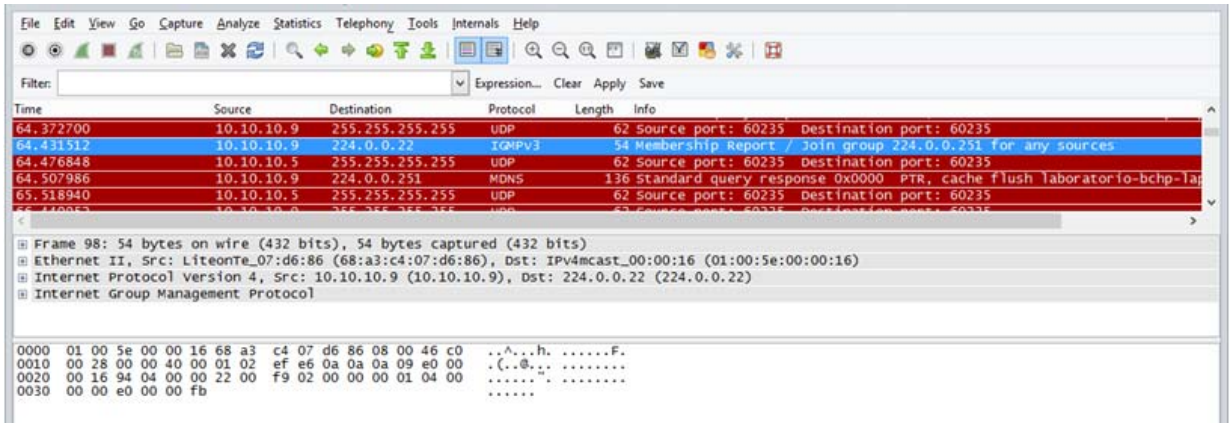

Fig. 60: Grupo de asociación (Autores, 2013)

En la Fig. 61 se hizo una prueba básica de conectividad de los nodos, se utiliza un echo a través de un ping directo a cada nodo, demostrando que se pueden comunicar entre ellos, reconociendo sus direcciones Ip.

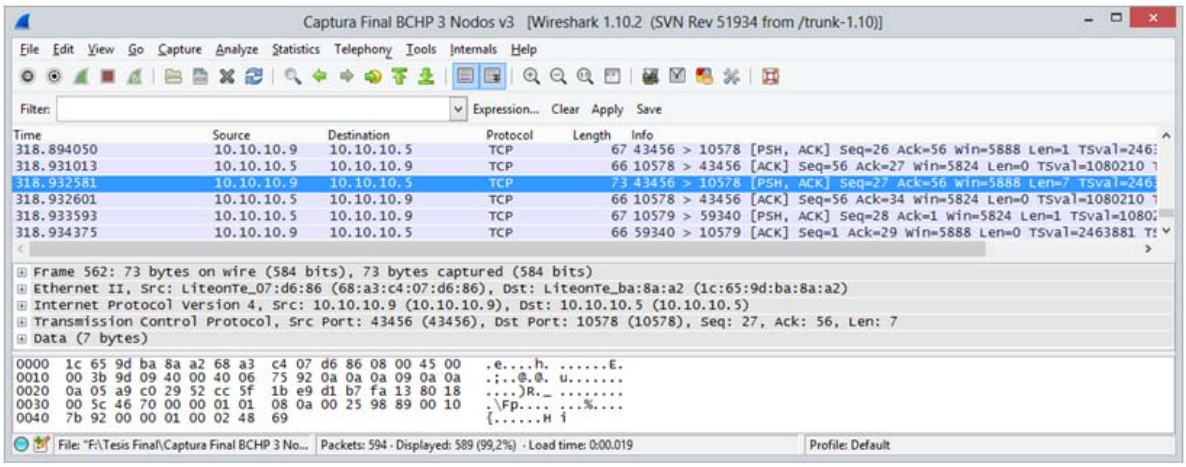

Fig. 61: Pruebas de Conectividad (Autores, 2013)

CONCLUSIONES

Durante la ejecución del trabajo de fin de titulación se puede concluir lo siguiente:

Finalizado el trabajo de investigación se puede concluir que los objetivos planteados previamente se han cumplido, se analizó dos formas de implementar las características del protocolo de enrutamiento BCHP en dispositivos móviles: la primera en smartphones Android, y la segunda en computadores portátiles con sistema operativo Ubuntu, culminando con éxito en la última implementación.

La implementación del protocolo en Android tuvo algunos inconvenientes debido a la escasa información sobre el tema y a las limitantes sobre el manejo del sistema operativo, sin embargo se logró simular algunas de las características, como inicialización del cluster, formación del cluster, inclusión de un nodo y elección del JC, que pueden servir de base para otras investigaciones acerca de los protocolos de enrutamiento ad-hoc en teléfonos celulares.

Se requiere de un hardware específico para la exitosa implementación del protocolo BCHP ya que se tiene una dependencia crítica del hardware de los dispositivos. En el capítulo 3 se detalla el escenario utilizado para las pruebas del BCHP, que incluye la capacidad de operar en modo Ad-Hoc del Wi-Fi y el manejo de multihilos de procesador.

El desarrollo de Java Chat BCHP es parte de la solución planteada en la presente tesis para demostrar el funcionamiento del protocolo, permitiendo una interfaz de comunicación entre los nodos conectados del cluster. Además de demostrar los estados que un nodo puede adquirir en los diferentes escenarios que pueda tener en la ejecución del protocolo.

En resumen se puede decir que se ha cumplido con la hipótesis y los objetivos propuestos en la presente tesis esto es:

Se cumple con la hipótesis y los objetivos planteados, es decir que si es posible plantear una solución que implemente en un entorno real las características de enrutamiento del protocolo BCHP, simulado en NS2 en la Tesis Doctoral “Contribución a la Gestión en Redes Móviles Ad-Hoc” (Torres, 2011), teniendo como escenario la creación de una red de comunicación en tres dispositivos móviles sin la inclusión de equipos intermedios.

Se realiza la comunicación entre los dispositivos móviles sin equipos intermedios. Para observar mejor esta característica se desarrolla la aplicación Java Chat BCHP que permite intercambiar datos entre los tres dispositivos conectados en una misma red ad-hoc.

Se verifica el funcionamiento de la solución planteada en dispositivos móviles a través del planteamiento de escenarios de comunicación, por medio del envío de paquetes ICMP (Protocolo de Mensajes de Control de Internet), de solicitud (echo ICMP) y respuesta (echo). Y con la ayuda de la aplicación Java Chat BCHP, se puede constatar que se está enviando información entre nodos exitosamente.

RECOMENDACIONES Y TRABAJO FUTURO

A partir de esta investigación se derivan varias recomendaciones y líneas de trabajo futuro:

Se recomienda la implementación del protocolo BCHP en hardware básico abierto como Arduino, ya que este permite modificar o desarrollar un sistema operativo en su totalidad completamente adaptado a las necesidades del protocolo, sin tener restricciones de uso en la licencia del software.

Debido a los pocos dispositivos disponibles que se ha tenido para la implementación del protocolo, sería una recomendación probar BCHP en un laboratorio que cuente con más computadores portátiles para visualizar con precisión el funcionamiento del protocolo.

Una línea de investigación futura puede ser la creación o adaptación de una herramienta de monitoreo que reconozca el protocolo de enrutamiento de BCHP, debido a que las herramientas existentes en el mercado actualmente no reconocen el protocolo y lo confunden con protocolos ya conocidos.

Actualmente el protocolo BCHP está probado en un solo cluster (3 nodos), quedando como futura investigación la ejecución de este protocolo en más clusters para poder obtener más indicadores de funcionamiento, tal como se encuentra en la simulación en NS2.

REFERENCIAS

- Alles, A. (2011). Internetworking. Recuperado el Marzo de 2013, de http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.116.9820&rep=rep1&type=pdf
- Android. (2012). Android Developers. Recuperado el Marzo de 2012, de http://developer.android.com/index.html
- Antón, G. H. (2012). ANDROID. Recuperado el Diciembre de 2012, de Historia de la Informática: http://histinf.blogs.upv.es/2012/12/14/android/android-gabriel-herraiz-anton/
- Baz, A., Ferreira, I., Álvarez, M., & García, R. (2011). Dispositivos Móviles. Recuperado el Junio de 2012, de http://156.35.151.9/~smi/5tm/09trabajos-sistemas/1/Memoria.pdf
- Boukhalkhal, A., Yagoubi, M. B., Djoudi, M., Ounten, Y., & Benmohammed, M. (2009). Simulation of Mobile Ad hoc Routing Strategies. Laghouat, Algeria.
- Browlee, B., & Liang, Y. (2011). Mobile Ad hoc Networks. An Evaluation of Smartphone Tecnologies . Canadá.
- Corriero, N., Mottola, A., & Zhupa, E. (2011). How to work with Android within a (FB-) Aodv network. Bari, Italy.
- Doumenc, H. (2008). Estudio Comparativo de Protocolos de Encaminamiento en Redes Vanet. Madrid: España.
- El Android Libre. (Noviembre de 2012). Recuperado el 05 de Febrero de 2013, de http://www.elandroidelibre.com/2012/09/android-en-el-futuro-las-caracteristicas-que-podriamos-ver-en-los-proximos-smartphones.html
- El Noticiero Universal. (Enero de 2013). Recuperado el Febrero de 2013, de http://elnoticierouniversal.com/post/41282116549/el-futuro-sera-android
- Gardner-Stephen, P. (2012). http://www.servalproject.org/. Recuperado el Febrero de 2012, de http://www.servalproject.org/
- Google Project Hosting. (2011). Google Projects. Recuperado el Marzo de 2012, de http://code.google.com/p/adhoc-on-android/

- H., B. (2005). The history of mobile ad-hoc networks. Recuperado el 18 de 12 de 2012, de http://www.ZATZ.com: http://zatz.com/computingunplugged/article/the-history-of-mobile-ad-hoc-networks/
- Hernández, E. (2011). Estudio e implementación de protocolos de enrutamiento de redes malladas inalámbricas. Recuperado el Enero de 2013, de http://oa.upm.es/14847/
- Instituto Nacional de Estadísticas y Cencos (INEC). (2011). Reporte de estadísticas sobre tecnologías de la información y comunicaciones . Ecuador.
- Jiménez, E., Zerna, J., Chávez, P., & Basurto, J. (2011). Análisis de la eficiencia de los Protocolos de Enrutamiento. Guayaquil, Ecuador.
- JiWire. (2012). Engaging Mobile Audiences. Recuperado el Noviembre de 2012, de http://www.jiwire.com/
- Johnson D., H. Y. (February de 2007). The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks for IPv4 Internet Engineering Task Force (IETF). RFC 4728 . Microsoft Research.
- Kulkarni, A., Spackmann, R., & Giri, K. (2006). Self-organized management of mobile adhoc networks, Military Communications Conference. Military Communications Conference, 2006. MILCOM 2006. IEEE, (págs. 1-7).
- Kumar, M., Rishi, R., & Madan, D. K. (2010). Comparative Analysis of CBRP, DSR, AODV Routing Protocol in MANET. Haryana, India.
- M. Jiang, J. L. (August de 1999). Cluster based routing protocol (cbrp) functional specification. internet draft, manet working group .
- M. Yassein, N. H. (2010). Improvement on cluster based routing protocol by using vice cluster head, in: Next Generation Mobile Applications. Next Generation Mobile Applications, Services and Technologies (NGMAST), 2010 Fourth International Conference on, (págs. 137-141). Amman.
- Magnus Frodigh, J. L. (2009). Ad Hoc Networks. http://people.cs.vt.edu/~hamid/Mobile_Computing/papers/frodigh_ericsson00.pdf.

- Maldonado, V. (2012). Comparación de Protocolos de Enrutamiento. Loja, Ecuador.
- Morris, J. L. (2001). Capacity of Ad-Hoc Wireless Network. Proceedings of the 7th annual international conference on Mobile computing and networking, (págs. 61-69). New York.
- Movistar. (2012). Comunidad Movistar. Recuperado el Noviembre de 2012, de Blog Android: http://comunidad.movistar.es/t5/Blog-Android/Android-el-sistema-operativo-del-futuro/ba-p/517849
- Murray D., D. M. (2010). An Experimental Comparison of Routing Protocols in Multi Hop Ad Hoc Networks. Australasian Telecommunication Networks and Applications Conference, ATNAC . Nueva Zelanda.
- Neumann A., A. C. (2008). Better Approach To Mobile Ad-hoc Networking (B.A.T.M.A.N.) Internet Engineering Task Force (IETF).
- Perkins, C., & Royer, E. (2003). Ad-hoc on-demand distance vector routing. Mobile Computing Systems and Applications, 2003. Proceedings. WMCSA 2003. Second IEEE Workshop on, (págs. 90-100). New Orleans, LA.
- Pomeyrol, J. (2012). Qué Futuro le espera a Android. My Computer .
- Ramos. (2010). Historia de la Informática. Recuperado el Abril de 2013, de http://histinf.blogs.upv.es
- Shen, C.-C., Jaikaeo, C., Srisathapornphat, C., & Huang, Z. (2002). The guerrilla management architecture for ad hoc networks. MILCOM 2002. Proceedings, (págs. 467-472).
- StatCounter. (2012). StatCounter Global Stats. Recuperado el Mayo de 2012, de http://gs.statcounter.com/
- TELECO. (2012). Estadísticas de Telecomunicaciones en el Mundo. Recuperado el Agosto de 2012, de http://www.itu.int/es/Pages/default.aspx
- Torres, R. (2011). CONTRIBUCIÓN A LA GESTIÓNN EN REDES MÓVILES AD HOC. Tesis Doctoral . Madrid, España.

- Torres, R. (2011). Management Ad Hoc Networks Model for rescue and emergency scenaries.
- Trujillo, M. B. (2010). Evaluación experimental de redes malladas basadas en el protocolo B.A.T.M.A.N. Madrid, España: Universidad Carlos III de Madrid.
- Weiser, M. (2009). The computer for the 21st century. Scientific American , 94-104.

ANEXOS

Anexo 1. Código del protocolo BCHP

Main

```
int set_kernel_options()
{
    int i, fd = -1;
    char on = '1';
    char off = '0';
    char command[64];
    if ((fd = open("/proc/sys/net/ipv4/ip_forward", O_WRONLY)) < 0)
            return -1;
    if (write(fd, &on, sizeof(char)) < 0)
            return -1;
    close(fd);
  for (i = 0; i < MAX_NR_INTERFACES; i++) {
            if (!DEV_NR(i).enabled)
                continue;
            memset(command, '\0', 64);
            sprintf(command, "/proc/sys/net/ipv4/conf/%s/send_redirects",
                    DEV_NR(i).ifname);
            if ((fd = open(command, O_WRONLY)) < 0)
                return -1;
            if (write(fd, &off, sizeof(char)) < 0)
                return -1;
            close(fd);
```

```
        memset(command, '\0', 64);
        sprintf(command, "/proc/sys/net/ipv4/conf/%s/accept_redirects",
                DEV_NR(i).ifname);
        if ((fd = open(command, O_WRONLY)) < 0)
            return -1;
        if (write(fd, &off, sizeof(char)) < 0)
            return -1;
        close(fd);
    }
    memset(command, '\0', 64);
    sprintf(command, "/proc/sys/net/ipv4/conf/all/send_redirects");
    if ((fd = open(command, O_WRONLY)) < 0)
        return -1;
    if (write(fd, &off, sizeof(char)) < 0)
        return -1;
    close(fd);
    memset(command, '\0', 64);
    sprintf(command, "/proc/sys/net/ipv4/conf/all/accept_redirects");
    if ((fd = open(command, O_WRONLY)) < 0)
        return -1;
    if (write(fd, &off, sizeof(char)) < 0)
        return -1;
    close(fd);
    return 0;
}
```

```
int find_default_gw(void)
{
    FILE *route;
    char buf[100], *l;
    route = fopen("/proc/net/route", "r");
    if (route == NULL) {
            perror("open /proc/net/route");
            exit(-1);
    }
    while (fgets(buf, sizeof(buf), route)) {
            l = strtok(buf, " \t");
            l = strtok(NULL, " \t");
            if (l != NULL) {
                if (strcmp("00000000", l) == 0) {
                        l = strtok(NULL, " \t");
                        l = strto
k(NULL, " \t");
                        if (strcmp("0003", l) == 0) {
                            fclose(route);
                            return 1;
                        }
                }
            }
    }
    fclose(route);
```

```
    return 0;
}
*Cargar los modulos de kernel */


void load_modules(char *ifname)
{
    struct stat st;
    char buf[1024], *l = NULL;
    int found = 0;
    FILE *m;
    memset(buf, '\0', 64);
    if (stat("./kbchp.ko", &st) == 0)
        sprintf(buf, "/sbin/insmod kbchp.ko ifname=%s &>/dev/null", ifname);
    else if (stat("./kbchp.o", &st) == 0)
        sprintf(buf, "/sbin/insmod kbchp.o ifname=%s &>/dev/null", ifname);
    else
        sprintf(buf, "/sbin/modprobe kbchp ifname=%s &>/dev/null", ifname);
    if (system(buf) == -1) {
       fprintf(stderr, "No se ha podido cargar el modulo kbchp\n");
       exit(-1);
    }
    usleep(100000);
    /* Check result */
    m = fopen("/proc/modules", "r");
    while (fgets(buf, sizeof(buf), m)) {
```

```
        l = strtok(buf, " \t");
        if (!strcmp(l, "kbchp"))
            found++;
        if (!strcmp(l, "ipchains")) {
            fprintf(stderr,
                    "El modulo de kernel ipchains a sido cargado adecudamente.\n");
            exit(-1);
        }
    }
    fclose(m);

    if (found < 1) {
        fprintf(stderr,
                "No se puede cargar el modulo del kernel, revisar instalación... %d\n",
                found);
        exit(-1);
    }
}
void remove_modules(void)
{
        int ret;
ret = system("/sbin/rmmod kbchp &>/dev/null");
        if (ret != 0) {
                fprintf(stderr, "No se puede remover el modulo de kernel kbchp\n");
        }
```

```
}
void host_init(char *ifname)
{
    struct sockaddr_in *ina;
    char buf[1024], tmp_ifname[IFNAMSIZ],
        ifnames[(IFNAMSIZ + 1) * MAX_NR_INTERFACES], *iface;
    struct ifconf ifc;
    struct ifreq ifreq, *ifr;
    int i, iw_sock, if_sock = 0;
    memset(&this_host, 0, sizeof(struct host_info));
    memset(dev_indices, 0, sizeof(unsigned int) * MAX_NR_INTERFACES);
    if (!ifname) {
        /* No se ha enviado interfaz, revisa primero la wireless. */
        iw_sock = socket(PF_INET, SOCK_DGRAM, 0);
        ifc.ifc_len = sizeof(buf);
        ifc.ifc_buf = buf;
        if (ioctl(iw_sock, SIOCGIFCONF, &ifc) < 0) {
            fprintf(stderr, "No se puede cargar la informacion del wireless\n");
            exit(-1);
        }
        ifr = ifc.ifc_req;
        for (i = ifc.ifc_len / sizeof(struct ifreq); i >= 0; i--, ifr++) {
            struct iwreq req;
            strcpy(req.ifr_name, ifr->ifr_name);
            if (ioctl(iw_sock, SIOCGIWNAME, &req) >= 0) {
```

```
                strcpy(tmp_ifname, ifr->ifr_name);
                break;
            }
        }
        /* Did we find a wireless interface? */
        if (!strlen(tmp_ifname)) {
            fprintf(stderr, "\nNo se puede encontrar una interfaz wireless!\n");
            fprintf(stderr, "Use -i <interfase> para sobreescribir...\n\n");
            exit(-1);
        }
        strcpy(ifreq.ifr_name, tmp_ifname);
        if (ioctl(iw_sock, SIOCGIFINDEX, &ifreq) < 0) {
            alog(LOG_ERR, errno, __FUNCTION__,
                    "No se puede obtener el indice de %s", tmp_ifname);
            close(if_sock);
            exit(-1);
        }
        close(iw_sock);
        ifname = tmp_ifname;
        alog(LOG_NOTICE, 0, __FUNCTION__,
             "Asociar a %s, sobreescribir con -i <if1,if2,...>.", tmp_ifname);
}
strcpy(ifnames, ifname);
/* Inicializar el numero de secuencia en cero del request rreq_id */
this_host.seqno = 1;
```

```
this_host.rreq_id = 0;
/* Ninguna interfaz activa hasta ahora... */
this_host.nif = 0;
gettimeofday(&this_host.bcast_time, NULL);
/* Buscar los indices de todas las interfaces para el broadcast... */
if_sock = socket(AF_INET, SOCK_DGRAM, 0);
iface = strtok(ifname, ",");
do {
        strcpy(ifreq.ifr_name, iface);
        if (ioctl(if_sock, SIOCGIFINDEX, &ifreq) < 0) {
            alog(LOG_ERR, errno, __FUNCTION__, "No se puede obtener el indice de %s",
                    iface);
            close(if_sock);
            exit(-1);
        }
        this_host.devs[this_host.nif].ifindex = ifreq.ifr_ifindex;
        dev_indices[this_host.nif++] = ifreq.ifr_ifindex;
        strcpy(DEV_IFINDEX(ifreq.ifr_ifindex).ifname, iface);
        /* Obtener la IP de la interface... */
        ina = get_if_info(iface, SIOCGIFADDR);
        if (ina == NULL)
            exit(-1);
        DEV_IFINDEX(ifreq.ifr_ifindex).ipaddr = ina->sin_addr;
        /* Obtener la netmask de la interface... */
        ina = get_if_info(iface, SIOCGIFNETMASK);
```

```
            if (ina == NULL)
                exit(-1);
            DEV_IFINDEX(ifreq.ifr_ifindex).netmask = ina->sin_addr;
            ina = get_if_info(iface, SIOCGIFBRDADDR);
            if (ina == NULL)
                exit(-1);
            DEV_IFINDEX(ifreq.ifr_ifindex).broadcast = ina->sin_addr;
            DEV_IFINDEX(ifreq.ifr_ifindex).enabled = 1;
            if (this_host.nif >= MAX_NR_INTERFACES)
                break;
    } while ((iface = strtok(NULL, ",")));
    close(if_sock);
    /* Cagar los modulos del kernel */
    load_modules(ifnames);
    /* Activar IP forwarding y fijar las otras opciones del kernel... */
    if (set_kernel_options() < 0) {
            fprintf(stderr, "No se pueden fijar las opciones del kernel!\n");
            exit(-1);
    }
}
void signal_handler(int type)
{
    switch (type) {
    case SIGSEGV:
            alog(LOG_ERR, 0, __FUNCTION__, "SEGMENTACION FALLIDA!!!! Saliendo!!! "
```

```
                "Para obtener core dump, compilar con la opcion DEBUG.");
    case SIGINT:
    case SIGHUP:
    case SIGTERM:
    default:
        exit(0);
    }
}
int main(int argc, char **argv)
{
    static char *ifname = NULL;
    fd_set rfds, readers;
    int n, nfds = 0, i;
    int daemonize = 0;
    struct timeval *timeout;
    struct timespec timeout_spec;
    struct sigaction sigact;
    sigset_t mask, origmask;
    progname = strrchr(argv[0], '/');
    if (progname)
        progname++;
    else
        progname = argv[0];
    /* Usar la opcion DEBUG por defecto */
    debug = 1;
```

```
    memset (&sigact, 0, sizeof(struct sigaction));
    sigact.sa_handler = signal_handler;
    sigaction(SIGTERM, &sigact, 0);
    sigaction(SIGHUP, &sigact, 0);
    sigaction(SIGINT, &sigact, 0);
    sigaddset(&mask, SIGTERM);
    sigaddset(&mask, SIGHUP);
    sigaddset(&mask, SIGINT);


void CLASS set_clusterhead(in_addr source)
{
    struct timeval now;
    rt_table_t *rt = NULL;
    u_int32_t seqno = 0;
    gettimeofday(&now, NULL);
    rt = rt_table_find(source);
    if (!rt) {
        DEBUG(LOG_DEBUG, 0, "%s NEIGHBOR!", ip_to_str(source));
        rt = rt_table_insert(source, source, 1, 0,
                             ACTIVE_ROUTE_TIMEOUT, VALID, 0, ifindex);
    } else {
        if (rt->flags & RT_UNIDIR)
            return;
        if (rt->dest_seqno != 0)
            seqno = rt->dest_seqno;
```

```
        rt_table_update(rt, source, 1, seqno, ACTIVE_ROUTE_TIMEOUT,
                        VALID, rt->flags);
}
if (!llfeedback && rt->hello_timer.used)
        hello_update_timeout(rt, &now, ALLOWED_HELLO_LOSS * HELLO_INTERVAL);
int min_addr = a->in_addr_;
int min_metric = a->metric_;
ntable_ent *tmp = t->clusterhead;
int next_min_metric = min_metric;
int next_min_addr = min_addr;
while (tmp) {
if ((tmp->neighbor_status == CLUSTER_UNDECIDED) &&
(tmp->link_status == LINK_BIDIRECTIONAL)) {
if (tmp->neighbor_metric < min_metric)
{
next_min_addr = min_addr;
next_min_metric = min_metric;
min_addr = tmp->neighbor;
min_metric = tmp->neighbor_metric;
t->my_ch = tmp;
}
else if (tmp->neighbor_metric < next_min_metric) {
next_min_addr = tmp->neighbor;
next_min_metric = tmp->neighbor_metric;
t->my_chb = tmp;
```

```
    }
    }
    tmp = tmp->next;
    }
    if ((min_addr == a->myaddr_) ||
    (next_min_addr == a->myaddr_)) {
    if (min_addr == a->myaddr_) {
    t->my_status = CLUSTER_HEAD;
    t->my_cluster_head_backup = next_min_addr;
    t->my_cluster_head = 0;
    t->my_ch = NULL;
    HAMAN_Agent::no_of_clusters_++;
    } else if (next_min_addr == a->myaddr_) {
    t->my_status = CLUSTER_HEAD_BACKUP;
    t->my_cluster_head_backup = 0;
    t->my_cluster_head = min_addr;
    t->my_chb = NULL;
    }
    p = t->getBroadcastPacket();
    s.schedule(a->ll, p, 0);
    return;
}
```

AGENTE

```
/* bchp.h
*/
```

```
struct BCHP_RtRepHoldoff { // Taken from DSR's implementation
ID requestor;
ID requested_dest;
int best_length;
int our_length;
};
struct BCHP_SendBufEntry {
Time t; // insertion time
BCHP_Packet p;
};
class BCHP_SendBufferTimer : public TimerHandler {
//Taken from DSR's implementation
public:
BCHP_SendBufferTimer(BCHP_Agent *a) : TimerHandler() { a_ = a;}
void expire(Event *e);
protected:
BCHP_Agent *a_;
};
class BCHP_Agent : public Tap, public Agent {
friend class HNeighborTable;
friend class HNeighborTablePeriodicHandler;
friend class HNeighborTableTimeoutHandler;
friend class HNeighborTableCFormationHandler;
friend class HNeighborTableCContentionHandler;
friend class BCHP_SendBufferTimer;
```

```
public:
BCHP_Agent();
virtual int command(int argc, const char * const * argv);
virtual void recv(Packet *, Handler* callback = 0);
void Terminate(void);
void lost_link(Packet *p);
static int no_of_clusters_ ;
// Jinyang It prints out how many clusters that have been formed in total
void tap(const Packet *p);
protected:
void process_cluster_update(Packet * p);
void forwardSRPacket(Packet *);
void startUp();
void trace(char* fmt, ...);
void tracepkt(Packet *, double, int, const char *);
Trace *logtarget; // Trace Target
PriQueue *ll_queue; // link level output queue
MobileNode* node_;
int myaddr_; // My address...
int metric_;
// My WCA metric ..
int myclusterid_ ;
// Identificador del cluster ID Added RT ene/2011
int be_random_;
HNeighborTable *ntable;
```

```
// Randomness/MAC/logging parameters used in DSR's implementation
int use_mac_;
int verbose_;
int trace_wst_;
private:
int off_mac_;
int off_ll_;
int off_ip_;
int off_bchp_;
ID net_id, MAC_id; // our IP addr and MAC addr
NsObject *ll; // our link layer output
PriQueue *ifq; // output interface queue
/******** internal state ********/
BCHP_SendBufferTimer send_buf_timer;
BCHP_SendBufEntry send_buf[SEND_BUF_SIZE];
RequestTable request_table;
RouteCache *route_cache;
int route_request_num; // number for our next route_request
//below codes r inherited from DSR for dealing with route errors
bool route_error_held; // are we holding a rt err to propagate?
ID err_from, err_to; // data from the last route err sent to us
Time route_error_data_time; // time err data was filled in
void handlePktWithoutSR(BCHP_Packet& p, bool retry);
void handlePacketReceipt(BCHP_Packet& p);
void handleForwarding(BCHP_Packet& p);
```

```
void handleRREQ(BCHP_Packet &p);
int handleRREP(BCHP_Packet &p);
bool ignoreRouteRequestp(BCHP_Packet& p);
void sendOutPacketWithRoute(BCHP_Packet& p,
bool fresh, Time delay = 0.0);
void sendOutRtReq(BCHP_Packet &p, int max_prop = MAX_SR_LEN);
void getRouteForPacket(BCHP_Packet &p, bool retry);
void acceptRREP(BCHP_Packet &p);
void returnSrcRouteToRequestor(BCHP_Packet &p);
bool replyFromRouteCache(BCHP_Packet &p);
void processBrokenRouteError(BCHP_Packet& p);
void xmitFailed(Packet *pkt);
void undeliverablePkt(Packet *p, int mine);
void dropSendBuff(BCHP_Packet &p);
void stickPacketInSendBuffer(BCHP_Packet& p);
void sendBufferCheck();
void sendRouteShortening(BCHP_Packet &p,
int heard_at, int xmit_at);
void BroadcastRREQ(BCHP_Packet &p);
int UnicastRREQ(BCHP_Packet &p, nsaddr_t nextcluster);
void fillBCHPPath(Path &path, hdr_bchp *&bchph);
void testinit();
friend void BCHP_XmitFailureCallback(Packet *pkt, void *data);
friend int BCHP_FilterFailure(Packet *p, void *data);
};
```

Tablas de enrutamiento

```
class ntable_ent {
public:
ntable_ent() { bzero(this, sizeof(struct ntable_ent));}
nsaddr_t neighbor; //target node
uint neighbor_status; //whether neighbor is a cluster member,
//head or undecided
uint link_status; // whether the link with this neighbor is
// bidirectional or unidirectional link
double last_update; //last update time for this entry
Packet* n_pkt;
Event* timeout_event; //time out event sent out
ntable_ent *next; //pointer to the next entry
double nseid; //added by zhaoqi 2007.6.28
};
class nexthop_ent {
public:
nexthop_ent() { bzero(this,sizeof(struct nexthop_ent)); }
nsaddr_t next_node;
Event* timeout_event;
nexthop_ent *next;
};
class adjtable_ent {
public:
```

```
adjtable_ent() { bzero(this, sizeof(struct adjtable_ent));}
nsaddr_t neighbor_cluster;
//the neighboring cluster head
nexthop_ent *next_hop;
//the next hop node in order to reach neighbor_cluster
adjtable_ent *next;
//pointer to the next adjtable_ent
};
class nextnode {
public:
nextnode() {bzero(this,sizeof(struct nextnode));}
nsaddr_t the_node;
nsaddr_t dest_status;
double last_updated;
};
class HNeighborTable {
friend class HNeighborTablePeriodicHandler;
//The handler function inside is invoked
//periodically to send out BROADCAST messages
friend class HNeighborTableTimeoutHandler;
// The handler function inside is invoked
// to remove outdated neighbortable entry
friend class HNeighborTableCFormationHandler;
// The handler function is invoked to
// perform cluster formation algorithm.
```

```
friend class HNeighborTableCContentionHandler;
// The handler function is invoked to
// resolve cluster head contention
friend class BCHP_Agent;
public:
HNeighborTable(BCHP_Agent *a_);
void AddUpdateEntry(ntable_ent *ent);
int DeleteEntry(nsaddr_t dest);
ntable_ent *GetEntry(nsaddr_t dest);
bool isNeighbor(nsaddr_t addr);
-Jinyang */
void AddUpdateQuickAdj(nsaddr_t dest, uint dest_status, nsaddr_t next_node);
nsaddr_t GetQuickNextNode(nsaddr_t dest);
bool existsLink(nsaddr_t from, nsaddr_t to);
void AddUpdateAdj(nsaddr_t dest, nsaddr_t next_hop, int primary);
void DeleteAdjEntry(nsaddr_t dest,int primary);
adjtable_ent *GetAdjEntry(nsaddr_t dest_cluster, int primary);
int RemainingLoop();
void InitLoop();
ntable_ent *NextLoop();
/*
*
* Added RT
*
* For Cluster_head_bakcup election
```

```
*
*/
ntable_ent *electCHB();
Packet *getUpdatePacket();
Packet *getBroadcastPacket();
void processUpdate(Packet *);
void startUp();
#ifdef BCHP_DEBUG
char *printNeighbors();
char *printHELLOpkt(Packet *p);
char *PrintTwoHopNeighbors();
#endif
uint no_of_clusterheads;
uint my_status;
//whether I am undecided, JC, JCR ..etc
uint my_size;
// Size of my neighbor table
nsaddr_t my_cluster_head;
// used only by the cluster head backup Added RT 9-02-2011
nsaddr_t my_cluster_head_backup;
// used only by the cluster head. Added RT 9-2-2011
uint backup_updates;
// for backups updates maintenance
uint adjtable_size;
Event *periodic_event;
```

```
Event *c_form_event;
Event *c_contention_event;
HNeighborTablePeriodicHandler *periodic_handler;
HNeighborTableTimeoutHandler *timeout_handler;
HNeighborTableCFormationHandler *c_formation_handler;
HNeighborTableCContentionHandler *c_contention_handler;
protected:
void lostClusterHeads();
nextnode *adjtable;
adjtable_ent *adjtable_1;
adjtable_ent *adjtable_2;
private:
BCHP_Agent *a;
ntable_ent *head;
ntable_ent *prev;
ntable_ent *now;
ntable_ent *my_ch;
// Used only by the Cluster Head Backup. Added RT 9-2-2011
ntable_ent *my_chb;
// Used only by the cluster head. Added RT 9-2-2011
int in_contention;
int maxelts;
int ctr;
double *last_update;
// a supplementary table used to updating the main one
```

```
class HNeighborTablePeriodicHandler : public Handler {
public:
HNeighborTablePeriodicHandler(BCHP_Agent *a_, HNeighborTable *t_);
virtual void handle(Event *e);
private:
HNeighborTable *t;
BCHP_Agent *a;
Event *periodic_event_;
};
class HNeighborTableTimeoutHandler : public Handler {
public:
HNeighborTableTimeoutHandler(BCHP_Agent *a_, HNeighborTable *t_);
virtual void handle(Event *e);
int checkAdjTimeout(Event *e, int primary);
private:
HNeighborTable *t;
BCHP_Agent *a;
};
class HNeighborTableCFormationHandler : public Handler {
public:
HNeighborTableCFormationHandler(BCHP_Agent *a_, HNeighborTable *t_);
virtual void handle(Event *e);
private:
HNeighborTable *t;
BCHP_Agent *a;
```

```
double preenergy; //added by zhaoqi 2006.6.30
};
class HNeighborTableCContentionHandler: public Handler {
public:
HNeighborTableCContentionHandler(BCHP_Agent *a_, HNeighborTable *t_);
virtual void handle(Event *e);
private:
HNeighborTable *t;
BCHP_Agent *a;
double preenergy;
//added by zhaoqi 2007.6.30
};
```

Anexo 2. Java Chat BCHP

Java Chat BCHP es una aplicación que nos permite mostrar de forma gráfica el funcionamiento del protocolo BCHP, esta aplicación está realizada en lenguaje Java. A continuación se muestra el código principal de la aplicación.

```
/*
 * Server.java
 */
package JefeDeCluster;

/**
 *
 * @author
 */
import Nodos.VentNodo;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;

public class JefeDeCluster extends JFrame {

    JTextArea txaMostrar;
```

```
public JefeDeCluster() {
    super("Consola Jefe de Cluster");
    txaMostrar = new JTextArea();
    this.setContentPane(new JScrollPane(txaMostrar));
    setSize(350, 350);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);

}

public JefeDeCluster(String nameNewServer) {
    super("Consola Jefe de Cluster " + nameNewServer);
    txaMostrar = new JTextArea();
    this.setContentPane(new JScrollPane(txaMostrar));
    setSize(350, 350);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // setVisible(true);
}

public void mostrar(String msg) {
    txaMostrar.append(msg + "\n");
}

public void runJC() {
    ServerSocket serv = null;//para comunicacion/conexion
```

```
ServerSocket serv2 = null;//para enviar mensajes
boolean listening = true;
try {
    serv = new ServerSocket(60235);
    serv2 = new ServerSocket(60236);
    mostrar(".::Jefe de Cluster activo :");
    mostrar("Jefe de Cluster levantado ");
    String ip_Nodo = ip_host();
    //VentCliente vcliente = new VentNodo("127.0.0.1", "JefeDeCluster");
    VentNodo vnodo = new VentNodo(ip_Nodo, "Servidor");
    while (listening) {
        Socket sock = null, sock2 = null;
        try {
            mostrar("Esperando Usuarios");
            sock = serv.accept();
            sock2 = serv2.accept();
        } catch (IOException e) {
            mostrar("Accept failed: " + serv + ", " + e.getMessage());
            continue;
        }
        threadJefeDeCluster user = new threadJefeDeCluster(sock, sock2, this);
        user.start();//llama al hilo
    }

} catch (IOException e) {
```

```
        mostrar("error :" + e);
    }
}

public void runJC_act(String nameNewServer, String ipNewServer) {
    ServerSocket serv = null;//para comunicacion/conexion
    ServerSocket serv2 = null;//para enviar mensajes
    boolean listening = true;
    try {
        serv = new ServerSocket(60235);
        serv2 = new ServerSocket(60236);
        mostrar(".::Jefe de Cluster activo :" + nameNewServer);
        mostrar("Jefe de Cluster levantado " + nameNewServer);
        setVisible(true);

        VentNodo vnodo = new VentNodo(ipNewServer, nameNewServer);



        while (listening) {
            Socket sock = null, sock2 = null;
            try {
                mostrar("Esperando Usuarios");
                sock = serv.accept();
                sock2 = serv2.accept();
            } catch (IOException e) {
```

```
                    mostrar("Accept failed: " + serv + ", " + e.getMessage());
                    continue;
                }
                threadJefeDeCluster user = new threadJefeDeCluster(sock, sock2, this);
                user.start();//llama al hilo
            }

        } catch (IOException e) {
            System.out.println("Jefe de Cluster nuevo levantado");
            JOptionPane.showMessageDialog(null, "Jefe de Cluster ya levantado");
//          try {
//              new VentNodo(2).new_Users();
//          } catch (Exception ex) {
//              Logger.getLogger(JefeDeCluster.class.getName()).log(Level.SEVERE, null,
ex);
//          }

        }
    }

    public static String ip_host() throws UnknownHostException, SocketException {
//      InetAddress Address = InetAddress.getLocalHost();
//      //String add=Address.getCanonicalHostName().toString();
//      String add = Address.getHostAddress().toString();
//      System.out.println("-> " + add);
//      return add;
```

```
            NetworkInterface ni = NetworkInterface.getByName("wlan0");
            Enumeration<InetAddress> inetAddresses = ni.getInetAddresses();



            while (inetAddresses.hasMoreElements()) {
               InetAddress ia = inetAddresses.nextElement();
               if (!ia.isLinkLocalAddress()) {
                  System.out.println("IP:  "  +  ia.getHostAddress()  +  "\nNOMBRE:  "  +
ia.getHostName());
                  return ia.getHostAddress().toString();
               }
            }
            return null;
         }

         public static void main(String abc[]) throws IOException {
            JefeDeCluster ser = new JefeDeCluster();
            ser.runJC();
         }
      }

      /*
       * Nodo.java
       */
      package Nodos;
```

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.io.*;
import java.net.Socket;
import java.util.Vector;
import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.JOptionPane.*;
import JefeDeCluster.JefeDeCluster;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Administrador
 */
public final class VentNodo extends JFrame implements ActionListener {

    String mensajeCliente;
```

```
JTextArea panMostrar;//textarea donde llegan los sms
JTextField txtMensage;
JButton butEnviar;
JLabel lblNomUser;
JList lstActivos;
JButton butPrivado;
Nodo nodo;
JMenuBar barraMenu;
JMenuItem acercaD;
Vector<String> nomUsers;
Vector<String> ipUsers;
VentPrivada ventPrivada;

/**
 * Creates a new instance of Nodo
 */
public VentNodo() throws IOException {
    super("Nodo Chat");
    txtMensage = new JTextField(30);
    butEnviar = new JButton("Enviar");
    lblNomUser = new JLabel("Usuario <<  >>");
    lblNomUser.setHorizontalAlignment(JLabel.CENTER);
    panMostrar = new JTextArea();
    panMostrar.setColumns(25);
    txtMensage.addActionListener(this);
```

```
        butEnviar.addActionListener(this);
        lstActivos = new JList();
        butPrivado = new JButton("Privado");
        butPrivado.addActionListener(this);

        barraMenu = new JMenuBar();

        panMostrar.setEditable(false);
        //panMostrar.setForeground(Color.black);//color de texto
        panMostrar.setBorder(javax.swing.BorderFactory.createMatteBorder(3, 3, 3, 3, new
Color(25, 10, 80)));

        JPanel panAbajo = new JPanel();
        panAbajo.setLayout(new BorderLayout());
        panAbajo.add(new JLabel("  Ingrese mensage a enviar:"), BorderLayout.NORTH);
        panAbajo.add(txtMensage, BorderLayout.CENTER);
        panAbajo.add(butEnviar, BorderLayout.EAST);
        JPanel panRight = new JPanel();
        panRight.setLayout(new BorderLayout());
        panRight.add(lblNomUser, BorderLayout.NORTH);
        panRight.add(new JScrollPane(panMostrar), BorderLayout.CENTER);
        panRight.add(panAbajo, BorderLayout.SOUTH);
        JPanel panLeft = new JPanel();
        panLeft.setLayout(new BorderLayout());
        panLeft.add(new JScrollPane(this.lstActivos), BorderLayout.CENTER);
        panLeft.add(this.butPrivado, BorderLayout.NORTH);
```

```
        JSplitPane sldCentral = new JSplitPane();
        sldCentral.setDividerLocation(100);
        sldCentral.setDividerSize(7);
        sldCentral.setOneTouchExpandable(true);
        sldCentral.setLeftComponent(panLeft);
        sldCentral.setRightComponent(panRight);
        setLayout(new BorderLayout());
        add(sldCentral, BorderLayout.CENTER);
        add(barraMenu, BorderLayout.NORTH);
        txtMensage.requestFocus();//pedir el focus
        nodo = new Nodo(this);
        nodo.conexion();
        //nomUsers=new Vector();
        ponerActivos(nodo.pedirUsuarios());//nombre del user
        ponerActivos_Ip(nodo.pedirUsuarios_IP());

//            presentar_lista_usuarios();
        ventPrivada = new VentPrivada(nodo);
        setSize(450, 430);
        setLocation(120, 90);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public VentNodo(String ip, String server_name) throws IOException { //Jefe de cluster
        super("BCHP Chat");
```

```
        txtMensage = new JTextField(30);
        butEnviar = new JButton("Enviar");
        lblNomUser = new JLabel("Usuario <<  >>");
        lblNomUser.setHorizontalAlignment(JLabel.CENTER);
        panMostrar = new JTextArea();
        panMostrar.setColumns(25);
        txtMensage.addActionListener(this);
        butEnviar.addActionListener(this);
        lstActivos = new JList();
        butPrivado = new JButton("Privado");
        butPrivado.addActionListener(this);

        barraMenu = new JMenuBar();

        panMostrar.setEditable(false);
        //panMostrar.setForeground(Color.black);//color de texto
        panMostrar.setBorder(javax.swing.BorderFactory.createMatteBorder(3, 3, 3, 3, new
Color(25, 10, 80)));

        JPanel panAbajo = new JPanel();
        panAbajo.setLayout(new BorderLayout());
        panAbajo.add(new JLabel("  Ingrese mensage a enviar:"), BorderLayout.NORTH);
        panAbajo.add(txtMensage, BorderLayout.CENTER);
        panAbajo.add(butEnviar, BorderLayout.EAST);
        JPanel panRight = new JPanel();
        panRight.setLayout(new BorderLayout());
```

```
panRight.add(lblNomUser, BorderLayout.NORTH);
panRight.add(new JScrollPane(panMostrar), BorderLayout.CENTER);
panRight.add(panAbajo, BorderLayout.SOUTH);
JPanel panLeft = new JPanel();
panLeft.setLayout(new BorderLayout());
panLeft.add(new JScrollPane(this.lstActivos), BorderLayout.CENTER);
panLeft.add(this.butPrivado, BorderLayout.NORTH);
JSplitPane sldCentral = new JSplitPane();
sldCentral.setDividerLocation(100);
sldCentral.setDividerSize(7);
sldCentral.setOneTouchExpandable(true);
sldCentral.setLeftComponent(panLeft);
sldCentral.setRightComponent(panRight);
setLayout(new BorderLayout());
add(sldCentral, BorderLayout.CENTER);
add(barraMenu, BorderLayout.NORTH);
txtMensage.requestFocus();//pedir el focus
nodo = new Nodo(this);
// setNombreUser("JefeDeCluster");
nodo.conexion(ip, server_name);
nomUsers = new Vector();
ipUsers = new Vector();
// agregarUser("JefeDeCluster");
//nomUsers=new Vector();
ponerActivos(nodo.pedirUsuarios(server_name));//nombre del user
```

```
            ponerActivos_Ip(nodo.pedirUsuarios_IP(ip));
            ventPrivada = new VentPrivada(nodo);

            setSize(450, 430);
            setLocation(120, 90);
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            setVisible(true);
        }

        public VentNodo(int a, String ip_server,String ip, String client_name,String server_name) throws IOException { //servidor
            //super("Nodo Chat");
            txtMensage = new JTextField(30);
            butEnviar = new JButton("Enviar");
            lblNomUser = new JLabel("Usuario <<  >>");
            lblNomUser.setHorizontalAlignment(JLabel.CENTER);
            panMostrar = new JTextArea();
            panMostrar.setColumns(25);
            txtMensage.addActionListener(this);
            butEnviar.addActionListener(this);
            lstActivos = new JList();
            butPrivado = new JButton("Privado");
            butPrivado.addActionListener(this);

            barraMenu = new JMenuBar();
```

```
        panMostrar.setEditable(false);
        //panMostrar.setForeground(Color.black);//color de texto
        panMostrar.setBorder(javax.swing.BorderFactory.createMatteBorder(3, 3, 3, 3, new
Color(25, 10, 80)));

        JPanel panAbajo = new JPanel();
        panAbajo.setLayout(new BorderLayout());
        panAbajo.add(new JLabel("  Ingrese mensage a enviar:"), BorderLayout.NORTH);
        panAbajo.add(txtMensage, BorderLayout.CENTER);
        panAbajo.add(butEnviar, BorderLayout.EAST);
        JPanel panRight = new JPanel();
        panRight.setLayout(new BorderLayout());
        panRight.add(lblNomUser, BorderLayout.NORTH);
        panRight.add(new JScrollPane(panMostrar), BorderLayout.CENTER);
        panRight.add(panAbajo, BorderLayout.SOUTH);
        JPanel panLeft = new JPanel();
        panLeft.setLayout(new BorderLayout());
        panLeft.add(new JScrollPane(this.lstActivos), BorderLayout.CENTER);
        panLeft.add(this.butPrivado, BorderLayout.NORTH);
        JSplitPane sldCentral = new JSplitPane();
        sldCentral.setDividerLocation(100);
        sldCentral.setDividerSize(7);
        sldCentral.setOneTouchExpandable(true);
        sldCentral.setLeftComponent(panLeft);
        sldCentral.setRightComponent(panRight);
        setLayout(new BorderLayout());
```

```
    add(sldCentral, BorderLayout.CENTER);
    add(barraMenu, BorderLayout.NORTH);

    txtMensage.requestFocus();//pedir el focus

    nodo = new Nodo(this);

    nodo.conexionNewUsers(ip,client_name);


   ponerActivos(nodo.pedirUsuarios(server_name,client_name));//nombre del user
    ponerActivos_Ip(nodo.pedirUsuarios_IP(ip_server,ip));

    ventPrivada = new VentPrivada(nodo);
    setSize(450, 430);
    setLocation(120, 90);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
}

public VentNodo(int a) {
}

public void setNombreUser(String user) {
    lblNomUser.setText("Usuario " + user);
```

```
    }

    public void mostrarMsg(String msg) {
        // System.out.println("VentNodo=> "+msg);
        this.panMostrar.append(msg + "\n");
    }

    public final void ponerActivos(Vector datos) {
        nomUsers = datos;
        ponerDatosList(this.lstActivos, nomUsers);
        if(nomUsers.size()>1){
            JOptionPane.showMessageDialog(null, "El Jefe de Cluster de Respaldo es " + nomUsers.get(1));
        }

    }

    public final void ponerActivos() {
        //nomUsers = datos;
        ponerDatosList(this.lstActivos, nomUsers);
        if(nomUsers.size()>1){
            JOptionPane.showMessageDialog(null, "El Jefe de Cluster de Respaldo es " + nomUsers.get(1));
        }

    }
```

```
    public final void ponerActivos_Ip(Vector datos) {
        ipUsers = datos;
        //ponerDatosList(this.lstActivos,nomUsers);
    }

    public void agregarUser(String user) {
        nomUsers.add(user);
        ponerDatosList(this.lstActivos, nomUsers);
    }

    public void retirraUser(String user)//unicamente para cuando se desconecta el Jefe de
Cluster.
    {
        if (user.equals("Jefe de Cluster")) {
            ipUsers.remove(0);
            nomUsers.remove(user);
            ponerDatosList(this.lstActivos, nomUsers);
            //poner nuevo Jefe de Cluster
            new_JefeCluster();
            // System.exit(0);
        }
    }

//-----------------------------------------------------------------------------------------
    public void retirraUser()//unicamente para cuando se desconecta el Jefe de Cluster.
```

```
        {
            ipUsers.remove(0);
            nomUsers.remove(0);
            ponerDatosList(this.lstActivos, nomUsers);
            new_JefeCluster();
        }

        public void new_JefeCluster() {
            try {
                this.setVisible(false);
                JOptionPane.showMessageDialog(null, "El nuevo Jefe de Cluster es " +
nomUsers.get(0));
                if((nodo.ip_host().toString()).equals(ipUsers.get(0))){
                    JefeDeCluster ser = new JefeDeCluster(nomUsers.get(0).toString());
                    ser.runJC_act(nomUsers.get(0).toString(), ipUsers.get(0).toString());
                }
                Nodo.IP_JC = ipUsers.get(0);//ip del nuevo Jefe de Cluster
                new_Users();
                //System.out.println("Entra aqceui ");
            } catch (UnknownHostException ex) {
                Logger.getLogger(VentNodo.class.getName()).log(Level.SEVERE, null, ex);
            } catch (SocketException ex) {
                Logger.getLogger(VentNodo.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
```

```
    public void new_Users() {
        for (int i = 1; i < this.nomUsers.size(); i++) {
            try {
                new VentNodo(i,ipUsers.get(0).toString(), ipUsers.get(i).toString(),
nomUsers.get(i).toString(),nomUsers.get(0).toString());

            } catch (Exception ex) {
                Logger.getLogger(VentNodo.class.getName()).log(Level.SEVERE, null, ex);
            }
        }

    }

    public void retirraUser(String user, String ipAddress) {

        nomUsers.remove(user);
        ipUsers.remove(ipAddress);
        ponerDatosList(this.lstActivos, nomUsers);
    }

    private void ponerDatosList(JList list, final Vector datos) {
        list.setModel(new AbstractListModel() {
            @Override
            public int getSize() {
                return datos.size();
            }
```

```
        @Override
        public Object getElementAt(int i) {
            return datos.get(i);
        }
    });
}

@Override
public void actionPerformed(ActionEvent evt) {
    String comand = (String) evt.getActionCommand();
    if (evt.getSource() == this.butEnviar || evt.getSource() == this.txtMensage) {
        String mensaje = txtMensage.getText();
        nodo.flujo(mensaje);
        txtMensage.setText("");
    } else if (evt.getSource() == this.butPrivado) {
        int pos = this.lstActivos.getSelectedIndex();
        if (pos >= 0) {

            ventPrivada.setAmigo(nomUsers.get(pos));
            ventPrivada.setVisible(true);
        }
    }
}

public void mensageAmigo(String amigo, String msg) {
```

```
            ventPrivada.setAmigo(amigo);

            ventPrivada.mostrarMsg(msg);

            ventPrivada.setVisible(true);

        }

        public static void main(String args[]) throws IOException {

            Nodo.IP_JC= JOptionPane.showInputDialog("Introducir IP Jefe de Cluster :",
"localhost");

            // Nodo.IP_SERVER="127.0.0.1";

            VentNodo p = new VentNodo();

        }

    }
```