UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

IMPLEMENTACIÓN DEL PROTOCOLO HLMP EN ANDROID

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

FELIPE IGNACIO VALVERDE CAMPOS

PROFESOR GUÍA:
SERGIO OCHOA DELORENZI

MIEMBROS DE LA COMISIÓN:
JOHAN FABRY
TOMAS BARROS ARANCIBIA

Este trabajo de memoria ha sido parcialmente financiado por el proyecto Fondecyt Nro: 1120207

SANTIAGO DE CHILE
AGOSTO 2012

Resumen

El uso de dispositivos móviles, tales como smartphones y notebooks, ha ido en aumento estos últimos años a lo largo del mundo. En particular, se ha registrado un alza importante en el acceso a estos y a su uso en Chile. Hoy en día, es difícil encontrar a personas que no posean dispositivos móviles, ya sea laptops, smartphones o tablets. Entre los dispositivos móviles tipo smartphones, el sistema operativo Android se ha posicionado como uno de los grandes a la hora de mover capital e iniciativas, destacando también que grandes empresas desarrolladoras de hardware para telefonía móvil han apoyado su progreso. Otro punto a favor de Android, es la facilidad con que se puede desarrollar aplicaciones debido a su amplia comunidad y completa documentación en línea, lo cual ha permitido que los usuarios perciban a este sistema operativo como confiable.

Los anteriores dispositivos móviles, por lo general, se conectarán a redes WiFi para obtener acceso a Internet. Sin embargo, existe otro tipo de conexión WiFi cuya característica radica en comunicar una red de dispositivos sin necesidad de un punto centralizado de acceso. Es así como nace High Level MANET Protocol (HLMP), un protocolo que tiene como objetivo proveer comunicación entre estos dispositivos móviles, generando una red interconectada que utiliza los mismos equipos disponibles en la red como antenas receptoras y emisoras. Teniendo en cuenta que HMLP se implementó para tres sistemas operativos, dejando de lado una alternativa de código libre, es que el presente trabajo de memoria tuvo como finalidad expandir el escenario de implementación de dicho protocolo HLMP a un cuarto sistema operativo que es particularmente Android.

El desarrollo se focalizó para smartphones, utilizando el equipo HTC Nexus One. El trabajo consistió en la adaptación de una biblioteca y una aplicación que, entre otras cosas, lograron ser compatible con la implementación original. Entre los elementos destacados del desarrollo, se puede mencionar la capacidad de transmitir archivos y mensajes de texto entre los equipos de la red. Por medio de datos experimentales, se logró concluir que HLMP para Android logró superar el tiempo promedio de conexión a la red, en relación a su implementación original. Sin embargo, las mediciones de la tasa de transferencia de archivos se mantuvieron relativamente cercanas a la implementación original, siendo la de este desarrollo un poco más baja.

Agradecimientos

Debo partir por Vivian Campos, mi madre, quien con su amor me enseñó desde que era pequeño que los sueños se pueden concretar con esfuerzo y constancia, es gracias a ella que he completado uno de mis anhelos, ser un profesional. También agradezco a Débora Campos, quien con su paciencia y experiencia me ayudó a no dejar de creer en mí. Además, quiero destacar a Gabriela Gacitúa, con su dedicación y compañía logré terminar esta importante etapa.

Finalmente quiero agradecer a Sergio Ochoa, Francisco Rodríguez y Nicolás Malbrán por colaborar con su conocimiento y disposición a concretar este trabajo.

Contenidos

Resumen ii

Agradecimientos iii

Contenidos iv

1 Introducción 1

1.1 Justificación y Motivación del Trabajo 2

1.2 Objetivos de la Memoria 2

1.3 Implicancias de la Memoria 3

2 Trabajos Relacionados 5

2.1 OLSR (Optimized Link State Routing Protocol) 5

2.1.1 OLSR-Mesh-Tether 5

2.2 B.A.T.M.A.N. (Better Approach To Mobile Adhoc Networking) 5

2.2.1 ServalProyect 6

3 Descripción de la Solución 6

3.1 Ambiente Operacional 6

3.2 Arquitectura de Software 7

3.2.1 Módulo de Interoperabilidad 8

3.2.2 Módulo de Red 8

3.2.3 Módulo de Comunicación 9

3.3 Alternativas Analizadas 9

3.3.1 Módulo de Interoperabilidad 10

3.4 Requisitos de la Solución 11

3.5 Factibilidad técnica 11

3.6 Prueba Piloto 12

4 Diseño de la Solución 13

4.1 Estructura Básica de Android 13

4.2 Diseño de HLMP para Android 14

4.2.1 HLMP-Java 14

4.2.2 Android-AdHoc 15

4.2.3 HLMPConnect 16
5 Implementación de la Solución 16
5.1 HLMP-Java 17
5.2 Android-AdHoc 20
5.3 HLMPConnect 22
6 Resultados Obtenidos y Lecciones Aprendidas 24
6.1 Resultados Obtenidos 24
6.2 Lecciones Aprendidas 26
7 Conclusiones y Trabajo a Futuro 27
Bibliografía y Referencias 30
Anexo A 32
Anexo B 33
Anexo C 34
Anexo D 35
Anexo E 36
Anexo F 37
Anexo G 38
Anexo H 39
Anexo I 40
Anexo J 41

1 Introducción

El uso de los dispositivos móviles ha ido en aumento estos últimos años en el mundo. Dicho uso se explica tanto por la automatización y el mejoramiento en efectividad de aplicaciones laborales (por ejemplo compartir contenidos, comunicación colaborativa, entre otros), como también por la necesidad de comunicación entre personas (streaming[1] y redes sociales).

Los dispositivos móviles son muy variados, estos van desde teléfonos inteligentes (smartphones) y tabletas digitales (slates) hasta laptops y tablets. Las capacidades de procesamiento de estos dispositivos no solo se asemejan a los PC's convencionales, sino que éstas mejoran continuamente. Además, estos dispositivos son capaces de utilizar las redes inalámbricas – provistas por routers o access points – para tener acceso a redes locales o a internet. Otra característica de conexión importante, que poseen estos dispositivos, radica en su capacidad de comunicarse entre ellos sin la necesidad de utilizar infraestructura (por ejemplo, access point).

Estas redes ad hoc sin infraestructura fueron llamadas MANET[2] (Corson, et al., 1999) y se forman dinámicamente utilizando las antenas de los dispositivos que se encuentran físicamente cercanos, por esa razón, no requieren una infraestructura física intermedia. Este tipo de redes se utilizan para apoyar las interacciones entre personas en diversos escenarios y también como base para construir diversas aplicaciones de software, como por ejemplo: sistemas de apoyo para las comunicaciones durante catástrofes (Monares, et al., 2011), comunicación dentro de los túneles de las minas de extracción o sistemas de inspección técnica de obras civiles (Rodríguez, 2011). Esta última, es la tesis de magíster de Juan Francisco Rodríguez (alumno del DCC[3] en la FCFM[4] de la Universidad de Chile), quien desarrolló el protocolo HLMP[5] (Rodríguez-Covili, et al., 2011) para tres sistemas operativos: Windows Mobile, Windows Vista y Windows 7.

Teniendo en cuenta que HMLP está implementado para tres sistemas operativos, dejando de lado una alternativa de código libre, el presente trabajo de memoria para optar al grado de Ingeniero Civil en Computación tiene como finalidad expandir el escenario de implementación de dicho protocolo HLMP, incorporando un cuarto sistema operativo: Android. La implementación de HLMP para Android en principio será desarrollada para que funcione en smartphones, luego se analizarán posibles extensiones a otros tipos de hardware.

[1] Distribución de contenido multimedia a través de una red
[2] **M**obile **A**d **H**oc **N**etworks
[3] **D**epartamento de **C**iencias de la **C**omputación
[4] **F**acultad de **C**iencias **F**ísicas y **M**atemáticas
[5] **H**igh **L**evel **M**ANET **P**rotocol

1.1 Justificación y Motivación del Trabajo

Según los estudios de las empresas norteamericanas Gartner, Inc. (Pettey, et al., 2011) y Canalys (Canalys, 2011), dedicadas al análisis de las industrias de negocios, el uso en el mundo de dispositivos móviles, en este caso smartphones, con el sistema operativo Android ha ido en aumento entre el 2009 y el 2011. Más aún, se pronostica un crecimiento considerable de Android para el 2015, a pesar de la competencia que éste tiene con iOS, Windows y Symbian. Este crecimiento se ve fundamentado por dos grandes razones: (1) las grandes empresas de producción de dispositivos móviles - agrupadas en la Open Handset Alliance[6] - han aportado con el desarrollo de Android; (2) el desarrollo en Android no se ve afectado por la adquisición de licencias privativas, que impiden el fluir de los desarrollos como en la competencia; de esta manera el tiempo de desarrollo y lanzamiento es menor que en los otros escenarios.

Teniendo en cuenta este conocimiento, es indispensable desarrollar el protocolo HLMP para que sea soportado en Android, pues tanto el desarrollo de aplicaciones como la investigación se han centrado en esta tecnología móvil. Con el desarrollo de la biblioteca HLMP se permitirán eventuales procesos de colaboración entre los dispositivos que usen Android, o bien que se comuniquen mediante el protocolo HLMP (por ejemplo: aplicaciones en Windows Mobile, Windows Vista o Windows 7). Además, se facilitará el desarrollo de aplicaciones colaborativas móviles sobre la plataforma Android.

El problema a resolver en el ámbito de esta memoria radica en que la actual implementación de HMLP para Windows hace uso de bibliotecas propias de este sistema operativo, las cuales facilitan gran parte del proceso de comunicaciones entre dispositivos. Por ende, esta memoria no sólo implica implementar HLMP para Android, sino también, definir una solución reutilizable para realizar las funciones de red anteriormente mencionadas.

Finalmente, este desarrollo tampoco consta en sólo la implementación de un protocolo de comunicación en un sistema operativo sobre los dispositivos móviles, sino también, en abordar la interoperabilidad entre los diferentes sistemas operativos con los dispositivos soportados, teniendo en cuenta temas de ruteo de mensajes y de aplicaciones que utilicen la infraestructura desarrollada.

1.2 Objetivos de la Memoria

El objetivo general de este trabajo de memoria es implementar del protocolo HLMP para ser usado en teléfonos inteligentes que utilizan el sistema operativo Android. Los objetivos específicos que se desprenden del objetivo general son los siguientes:

[6] Alianza comercial que desarrolla estándares abiertos para dispositivos móviles

- Implementar el protocolo HMLP para Android, teniendo como objetivo la versión 2.3.6 de dicho sistema operativo, considerando como plataforma computacional el teléfono HTC Nexus One de Google.

- Crear una biblioteca que encapsule los servicios implementados en HLMP para Android. De esa manera, dichos servicios podrán ser utilizados como base por distintas aplicaciones de software.

- Permitir la comunicación entre aplicaciones que utilicen el protocolo HLMP, aunque estas corran en diferentes sistemas operativos.

- Generar una aplicación básica para Android 2.3.6 con el fin probar las capacidades de la implementación del protocolo HLMP realizadas.

1.3 Implicancias de la Memoria

La biblioteca HLMP permite abordar una importante gama de situaciones, tanto en la elaboración como en el enriquecimiento sus procesos. Esto se ve reflejado mediante los proyectos “Sistema de apoyo para las comunicaciones durante catástrofes” (Monares, et al., 2011) y “Sistema de inspección técnica de obras civiles” (Rodríguez, 2011), ambos utilizando tecnología móvil.

Por otra parte, en Chile se ha registrado (Aduanas, 2012) un aumento importante en las importaciones de teléfonos celulares (incluidos los smartphones) en los últimos años; destacando el alza entre enero y abril del 2010 estimado en $296.000.000 dólares, entre enero y abril del 2011 en $335.000.000 dólares y entre enero y abril del 2012 $465.000.000 dólares. De esta manera, los dispositivos con Android se encuentran en la vida cotidiana de las personas.

En cuanto a Android, el desarrollo de aplicaciones no requiere el uso de licencias privativas, posee documentación en línea completa (Android-Developer) y una experimentada comunidad de desarrollo a lo largo del mundo. Asimismo la cantidad de proyectos abiertos facilitan la reutilización de herramientas. Todo esto se traduce en la eficiencia de desarrollos con un menor tiempo de lanzamiento y la posibilidad de reorganizar los recursos.

En consecuencia, mezclando los tres conceptos anteriores, se logra visualizar que existe un nicho para desarrollar tecnología que utilice el protocolo de comunicación HLMP para

comunicar diferentes dispositivos, en particular este desarrollo para Android, dentro de las siguientes áreas:

- Turismo

  Muchos lugares turísticos poseen reseñas e indicaciones, sin embargo, a veces estas entorpecen el pasar de los visitantes o simplemente carecen de información. Esta biblioteca permite compartir contenido relacionado sin intervenir en el espacio y ocupar la tecnología que ya poseen los visitantes.

- Aplicaciones co-localizados

  Este escenario se presenta fuertemente en lugares como hospitales o centros de rehabilitación, en donde las personas (frecuentemente niños) se encuentran inhabilitados para movilizarse autónomamente. De esta manera, la utilización de HLMP no solo serviría como entretenimiento, sino también, podría potenciar otros factores en la rehabilitación de los pacientes.

- Redes sociales físicas

  El uso de las redes sociales ha penetrado diferentes rubros como entretenimiento, difusión, trabajo, etcétera. Sin embargo, un tópico menos explotado es el de las redes sociales físicas; estas corresponden a una comunidad que comparte la característica de estar ubicados físicamente cercanos, por ende, puede ser considerada de propósito general y crear una gama de sub rubros. Para ser más concreto basta pensar en una universidad, en donde los sub rubros pueden ser publicidad de casinos de comida y otros servicios, novedades y notificaciones personalizadas al alumnado, puntos de encuentro y grupos de trabajo, entre otros. De esta manera, el uso cotidiano de dispositivos móviles y la necesidad de comunicación entre personas cercanamente ubicadas permiten a HLMP generar múltiples aplicaciones comerciales, colaborativas y de difusión.

  Es así como en el contexto de generar instancias de trabajo (Vergara, 2012) se desarrolló una aplicación que buscar acercar las redes sociales hacia un plano físico utilizando HLMP para la plataforma Windows 7, con el propósito principal de potenciar los espacios colaborativos de trabajo. Por lo tanto, se refuerza la posibilidad de generar la misma instancia para la plataforma Android.

- Situaciones de emergencia

  En algunos casos de emergencia los canales de comunicación se ven afectados por la necesidad de una fuente eléctrica constante que no se encuentra en funcionamiento, por lo tanto, instrumentos como telefonía celular, parlantes u otros no logran

utilizarse. Es así como HLMP se puede utilizar en aplicaciones que permiten tener comunicación localizada tanto por mensajería como por voz y video.

2 Trabajos Relacionados

El contexto de esta biblioteca, es conectar dispositivos móviles por medio de redes WiFi del tipo Ad-Hoc, en particular MANET's en Android. Desde esta perspectiva, la búsqueda de trabajos relacionados se centro en las MANETs existentes y aplicaciones en Android que las utilicen.

2.1 OLSR (Optimized Link State Routing Protocol)

Es una de las implementaciones más eficientes en cuanto a recursos y resultados de MANETs. Al igual que HLMP, es un tipo de MANET activa o dinámica, la cual por medio de mensajería de bajo costo de computo, pero elevado en transferencia, permite conocer el estado de la red en todo momento. A diferencia de HLMP, esta MANET tiene optimizado el proceso de conocer la red. Además posee una estandarización internacional por parte de la organización IETF[7], quienes desarrollan y promocionan estándares de comunicación en internet.

2.1.1 OLSR-Mesh-Tether

Este es una aplicación para Android que actualmente se encuentra en desarrollo. Es un fork de la aplicación Bernacle, la cual ofrece el servicio de iniciar redes WiFi del tipo Ad-Hoc. Esta aplicación sólo ejecuta el inicio del *deamon*[8] de OLSR para Android, pero no posee una interfaz.

2.2 B.A.T.M.A.N. (Better Approach To Mobile Adhoc Networking)

Es una implementación de MANET en el lenguaje C y forma parte de los módulos del Kernel de Linux desde la versión 2.6.8. Al igual que OLSR, también es un tipo de MANET activo o dinámico.

[7] **I**nternet **E**ngineering **T**ask **F**orce
[8] Proceso computacional que corre en el background

2.2.1 ServalProyect

Es una aplicación para Android que actualmente se encuentra en desarrollo, no obstante, es una aplicación estable con funciones básicas de comunicación como mensajería, streaming de audio y video. Ha sido utilizada en varios ámbitos, desde la emergencia ocurrida tras el terremoto de Haití en el 2010, como la comunicación en los parques nacionales en Australia. Este desarrollo posee una gran comunidad de desarrollo, liderada por precursores de teléfonos celulares de los años 90'.

Pese a existir estos protocolos, se encontraron pocas aplicaciones que los utilicen como soporte para intercomunicar otros dispositivos móviles. Esto puede ser tomado como una ventaja a la hora de implementar el protocolo HMLP, pues parece ser un área no explotada en el ambiente Android para aplicaciones de propósito general.

3 Descripción de la Solución

La solución requiere especificar dos aspectos: hardware y software. A continuación se describirá cada una de éstas y además se comentará acerca de los requisitos, factibilidad y otros temas relacionados sobre la evaluación previa al desarrollo.

3.1 Ambiente Operacional

Este ambiente busca corroborar la interoperabilidad de las implementaciones para los diferentes sistemas operativos. Teniendo en cuenta que el protocolo HLMP hace uso de las redes Wi-Fi[9] para conectar los dispositivos inalámbricamente, el problema a resolver abarca tres casos. El primer caso es comunicar mediante HLMP dos Smartphone con sistema operativo Android; es decir, estos Smartphone hacen uso de la biblioteca desarrollada en este trabajo de título para una comunicación directa entre los dispositivos. Esto se puede apreciar en la Figura 1.

[9] **Wi**reless **Fi**delity, es un conjunto de estándares para redes inalámbricas basado en las especificaciones IEEE.802.11

El segundo caso es comunicar un Smartphone con sistema operativo Android y otro con sistema operativo Windows, es decir, se hace uso la implementación previa y la presentada en este documento para una comunicación directa entre los dispositivos. Esto se puede apreciar en la Figura 1.

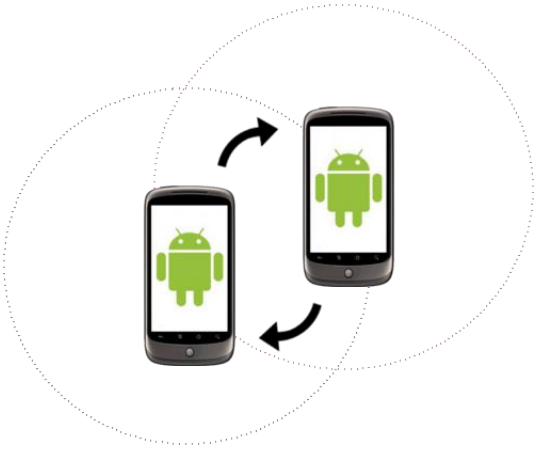


Figura 2: MANET entre dos Android

Figura 1: MANET entre un Android y Windows Mobile

Por último, la idea es comunicar dos dispositivos con sistema operativo Android mediante el paso de otro dispositivo con sistema operativo Windows o Android. Se hará uso de ambas implementaciones. Esto se puede apreciar en la **¡Error! No se encuentra el origen de la referencia.**.

3.2 Arquitectura de Software

Este desarrollo se constituye de dos componentes de software, una es la biblioteca HLMP para ser usada en Android y la otra es una aplicación que hace uso de esta biblioteca que permite reconocer a los dispositivos de la red, transferir mensajes de texto y archivos. Para el caso de la biblioteca HLMP, el esquema arquitectónico se hereda del desarrollo original. Este se compone de tres módulos: Comunicación, Red e Interoperabilidad. En la **¡Error! No se encuentra el origen de la referencia.** se puede apreciar gráficamente la arquitectura. A continuación se dará una breve reseña de cada componente.

Figura 3: MANET usando dispositivo intermedio

3.2.1 Módulo de Interoperabilidad

Esta capa está a cargo de procedimientos que son delegados al sistema operativo que tienen relación con el proceso de conexión a la MANET. Algunos de ellos son: configuración del perfil de WLan Ad-Hoc, control del dispositivo de red inalámbrico, configuración de dirección IP y sub-máscara de red, DAD en fase fuerte, lectura de notificaciones del sistema operativo, entre otros.

3.2.2 Módulo de Red

Esta componente implementa los servicios TCP [10]y UDP[11], necesarios para el intercambio de datos en la MANET. Al momento de ser recibidos, los mensajes son validados y encolados, para ser luego atendidos por la capa de Comunicación. Esta capa tiene también la responsabilidad de administrar las conexiones TCP, con dispositivos remotos que se encuentran dentro de la vecindad del usuario.

[10] **T**ransmission **C**ontrol **P**rotocol
[11] **U**ser **D**atagram **P**rotocol

3.2.3 Módulo de Comunicación

Este módulo implementa la API[12] o interfaz que los desarrolladores pueden usar para dar soporte a la comunicación en aplicaciones colaborativas móviles. Administra también los servicios de organización de mensajes y empaquetamiento de datos, entre otros.

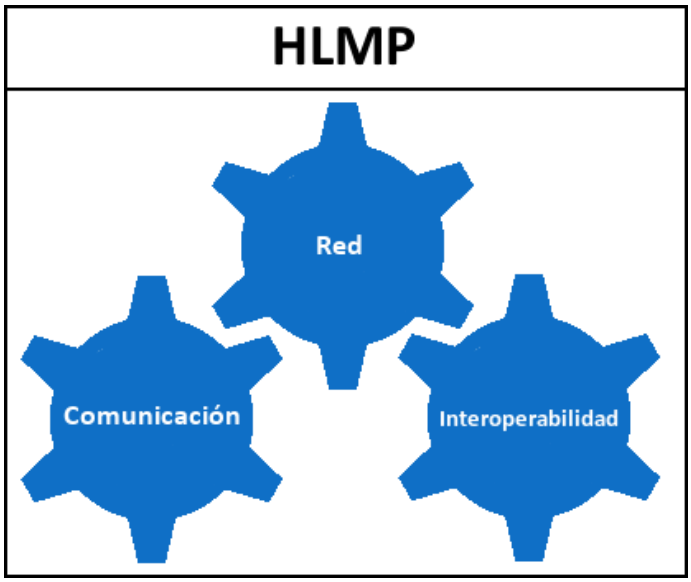


Figura 4: Arquitectura de HLMP

Con respecto a la aplicación para Android, que hará uso de este protocolo MANET, tendrá las mismas características que la aplicación original para Windows Mobile que fue desarrollada con HLMP, es decir, podrá listar los usuarios disponibles, comunicarse con los usuarios en la red, compartir archivos y ver el estado de transferencia. Sin embargo, la representación visual de la red en forma de grafo se dejará para futuros desarrollos por motivos de tiempo.

3.3 Alternativas Analizadas

A continuación se mencionan las características analizadas que permitieron dilucidar cuál fue la mejor alternativa para abordar el desarrollo de esta solución.

[12] **A**pplication **P**rogramming **I**nterface

3.3.1 Módulo de Interoperabilidad

Para este desarrollo de HLMP se utiliza el teléfono Nexus One, que posee el sistema operativo Android 2.3.6. A su vez, la forma en que HLMP se comunica es mediante el uso de WiFi con una red Ad-Hoc. El inconveniente de esto, es que la API de la versión 2.3.6 de Android no ofrece una forma de configurar ni conectarse a redes de tipo Ad-Hoc para comunicar dispositivos inalámbricamente. Sin embargo, Android posee versiones posteriores y al verificar el soporte de conexión entre dispositivos se logró conocer que solo desde versión Android 4.0 (denominada Ice Cream Sandwich, lanzado en Octubre del 2011) se soporta nativamente un tipo de conexión entre dispositivos, no obstante, este tipo de conexión es utilizando Wifi-Direct (WiFi-Alliance), el cual no es compatible a una red Ad-Hoc, por motivos tales como la comunicación es cifrada y además permite paralelamente utilizar WiFi. Por lo tanto, de ninguna manera desde las API's de Android se puede configurar una red Ad-Hoc.

El tipo de red Ad-Hoc es indispensable para HLMP, por lo tanto, la búsqueda para soportar este tipo de conexión fue más allá de la API de Android. De esta manera, teniendo acceso de super usuario en la versión Android 2.3.6, se trabajó en al menos listar las redes Ad-Hoc y generar una conexión entre dos dispositivos. Esto se logró por medio de 3 pasos:

1. Obtener acceso de super usuario para generar las modificaciones. Para esto se adaptaron a Linux scripts conocidos que se encuentran disponibles para Windows. Esto debido a que el ambiente de desarrollo (SDK[13], IDE[14] para desarrollar, etc) se está realizando en Ubuntu 11.10. El código fuente y sus binarios se encuentra en un sistema de versiones ubicado en https://github.com/fvalverd/Android-Easy-Rooting-Toolkit. Fue probado para el dispositivo Nexus One.

2. Cambiar el binario wpa_supplicant, una herramienta que controla las redes inalámbricas en Linux y por ende Android. La razón es que esta herramienta, y otras del sistema base de Linux, en Android son minimalistas, debido al espacio y procesamiento de los dispositivos. Para esto, es necesario utilizar un parche (ver Anexo J) para permitir listar las redes WiFi tipo Ad-Hoc, en consecuencia, la API de Android reconoce y permite conectarse a las redes Ad-Hoc listadas, no obstante, no permite unirse a una no listada.

3. Ajustar parámetros de configuración para cada red Ad-Hoc en los archivos de configuración de wpa_supplicant.

Adicionalmente, se buscó una aplicación para listar las redes Ad-Hoc automáticamente, esta fue encontrada satisfactoriamente. La aplicación se llama WiFi Ad Hoc enabler y puede ser

[13] **S**oftware **D**evelopment **K**it
[14] **I**ntegrated **D**evelopment **E**nvironment

encontrada en el Google Play Market (WiFi-AdHoc-Enabler), es gratuita y también requiere tener acceso de super usuario en el dispositivo. La cualidad de esta aplicación es que realiza los mismos pasos antes mencionados. Sin embargo, lo anterior no es suficiente para conectarse a una red Ad-Hoc cualquiera, pues se requiere que la red exista previamente, pero, la existencia de las aplicaciones Bernacle (Bernacle) y WiFi Tether (WiFi-Tether), las cuales permiten (por medio de permiso de super usuario) crear una conexión WiFi del tipo Ad-Hoc, permitieron creer que existen métodos alternativos para lograr el uso automatizado de dicho tipo de redes.

Con respecto a HLMP, importante es que solo la componente Interoperabilidad se ve afectada, dependiendo del dispositivo móvil y la versión del sistema operativo Android. Por lo tanto, la separación de los módulos de HLMP (desde su implementación original) permite desarrollar las demás componentes independiente a que dispositivo móvil se utilice en el sistema operativo Android.

3.4 Requisitos de la Solución

La implementación de HLMP para Android satisface las mismas características que la versión original para otras plataformas. De esta manera, se mencionarán algunas de ellas que están presentes en esta implementación:

- Conexión Ad-Hoc automatizada, el usuario solo deberá configurar su nombre de usuario.
- Comunicación con las plataformas Windows Xp, Windows 7 y Windows Mobile.
- Conexión y desconexión de la red manteniendo las configuraciones.

3.5 Factibilidad técnica

Antes de comenzar a desarrollar esta herramienta, se verificó lo siguiente para estar seguros de un prospero producto.

- **Hardware**: El equipo HTC Nexus One se encontró disponibles desde un comienz, las pruebas sobre obtener permiso de super usuario para conectar/crear una conexión Ad-Hoc fueron realizadas exitosamente antes de comenzar el desarrollo. Por otra parte, se solicitó un nuevo equipo para examinar e investigar posibles futuros caminos a seguir con respecto a WiFi-Direct, el equipo en cuestión fue el Samsung Galaxy Nexus.
- **Comunicación**: No existía desarrollo de esta biblioteca en este tipo de dispositivos. Por lo tanto existió una incertidumbre sobre la dificultad y el tiempo estimado. Sin embargo, se

conoció sobre algunos casos que implementaron bibliotecas y aplicaciones similares (ver Sección Trabajos Relacionados), por lo tanto, se respaldo un resultado favorable sobre la comunicación con diferentes tipos de MANET en Android.

- **Cambios proyectados**: Este desarrollo consiste en la adaptación de una herramienta existente, por ende nuevos desafíos o nuevos objetivos no se esperaron ni ocurrieron durante el desarrollo, más bien aparecieron dificultades técnicas y de aprendizaje.

- **Soporte de Software**: La implementación de HLMP en Android debe ser sobre el lenguaje Java porque toda aplicación en Android se construye sobre este lenguaje, por lo tanto, para utilizar la biblioteca de manera expedita se ocupó el mismo lenguaje. Además, en menor medida se ocuparon los lenguajes Bash, C y C++ para entender, manipular y modificar las herramientas para configurar la red Ad-Hoc.

- **Usabilidad sobre la aplicación**: El grueso de este desarrollo se centró en implementar la biblioteca HLMP, por ende, sabiendo los anteriores desafíos e incertidumbres se desarrolló una aplicación lo más usable posible en el tiempo disponible.

3.6 Prueba Piloto

Para realizar la primera prueba de comunicación entre dispositivos por medio de una MANET, se dispuso de un computador con Windows XP con la aplicación desarrollada por el creador de HLMP y un smartphone (HTC Nexus One) con la aplicación desarrollada en este trabajo.

Para que el dispositivo con Android pudiese interactuar en una red Ad-Hoc, se creó manualmente la red utilizando la aplicación Bernacle (mencionada en la anterior sub-sección: alternativas analizadas para el módulo de interoperabilidad) con la misma configuración de la implementación original de HLMP y una IP fija. Paralelamente, se ejecutó la aplicación implementada para Android que permite enviar y recibir mensajes de textos desde otros dispositivos por medio de HLMP. Los primeros intentos fueron sin éxito, pues se tuvo que realizar modificaciones para que el dispositivo con Android permitiera crear puntos de comunicación entre los dispositivos (sockets), además de permitir recibir información por el protocolo UDP correctamente. Finalmente, se logró generar la comunicación entre estas máquinas y se tuvo una comunicación fluida mediante el subprotocolo de mensajería de texto de HLMP.

Esta prueba sirvió para detectar a tiempo los problemas de conexión en Android además de otros problemas de interacción en la implementación de HLMP para Android.

4 Diseño de la Solución

El diseño de la biblioteca HLMP fue pensado en el contexto de un desarrollo en Android, por lo tanto, en esta sección primero se explicará brevemente algunos elementos esenciales para un desarrollo en este sistema operativo, con el fin de entender las componentes reutilizadas y creadas en el diseño de HLMP para Android, al final se explicará el diseño de la solución.

4.1 Estructura Básica de Android

Las aplicaciones en Android se deben programar en el lenguaje de programación Java – junto a otros recursos de configuración – y se compilan con el Android SDK con el fin de generar un único archivo (de extensión .apk) para ser instalado en los dispositivos. Una vez instalado, la aplicación vivirá en su propia sandbox[15] por motivos de seguridad y ejecución de procesos en Android. Adicionalmente, y de manera opcional, se pueden utilizar componentes nativas de Linux que no son accesibles desde la API[16] de Android. Estas rutinas deben ser escritas en los lenguajes C o C++ y son compiladas con el Android NDK[17] para ser utilizadas desde la aplicación escrita en Java mediante JNI[18].

La aplicación se ejecuta con permisos de usuario básico, desde esta perspectiva, si se requiere hacer cambios en la configuración del sistema operativo, solo podrá ser utilizando lo que ofrece la API de Android. Sin embargo, si se modifica el sistema operativo Android para obtener acceso de super usuario, se podrán hacer cambios desde cualquier rutina sobre todo el sistema operativo.

Toda aplicación a desarrollar debe contener un único archivo de configuración – de nombre *AndroidManifest.xml* – ubicado en el directorio raíz del desarrollo. En este archivo se declaran los elementos a utilizar por la aplicación, la forma en que se relacionan y los permisos necesarios, además de otras configuraciones. Existen dos elementos importantes en esta configuración, primero la clase a instanciar para mantener un estado global de la aplicación, segundo las clases a instanciar de las diferentes componentes de aplicación que constituyen la aplicación.

Existen cuatro tipos de componentes de aplicación para desarrollar una aplicación en Android, cada una con diferentes propósitos y ciclos de vida que definen cómo se crean y destruyen. Nos

[15] Mecanismo para ejecutar programas con seguridad y de manera separada de los otros.
[16] **A**pplication **P**rogramming **I**nterface
[17] **N**ative **D**evelopment **K**it
[18] **J**ava **N**ative **I**nterface

centraremos en dos, las cuales son de mayor de interés desde el punto de vista de un usuario final; los *Activities* y *Services*. La primera representa un proceso con interfaz de usuario desplegada en el dispositivo y la segunda un proceso sin interfaz de usuario que se ejecuta paralelamente a otras del tipo anterior. De esta manera, una aplicación puede tener más de una componente de aplicación y estas componentes pueden iniciar otras mediante diferentes mecanismos, sin embargo al momento de ejecutar la aplicación comenzará con la ejecución de solo una componente configurada en el archivo de configuración de la aplicación.

A veces las componentes de aplicación requieren compartir contenido entre ellas, para esto existen al menos dos formas, una es por medio de solicitudes de acciones mediante *Intents* (además estas solicitudes poseen la capacidad de iniciar componentes de aplicación), la otra forma es manteniendo actualizado el estado global de la aplicación por medio de la instancia de la clase configurada en el archivo de configuración de la aplicación.

4.2 Diseño de HLMP para Android

A diferencia de las otras implementaciones de HLMP para Windows, esta implementación no solo es una biblioteca para utilizar en un desarrollo, sino que es la estructura inicial para comenzar una aplicación que utilizará HLMP en Android.

HLMP para Android se compone de tres elementos, el primero denominado *Android-AdHoc* encargado de la administración del WiFi en el dispositivo, el segundo denominado *HLMP-Java* relacionado al comportamiento del protocolo HLMP, y el tercero denominado *HLMPConnect* es el esqueleto de la aplicación que administra los recursos anteriores y permite la comunicación con el usuario.

4.2.1 HLMP-Java

Es la implementación del protocolo HLMP y está basada en el desarrollo de HLMP para Windows XP, Vista y Mobile. Esta biblioteca se componen de tres módulos y tres sub-protocolos, los módulos son Red, Comunicación e Interoperabilidad (ver Figura 4), están encargados respectivamente de la apertura y control de sockets, administración de eventos y mensajes y por último el control de la conexión WiFi del dispositivo. De igual manera, los tres sub-protocolos son Ping, Chat y FileTransfer, encargados respectivamente de comprobar la conexión de otros dispositivos, proveer un chat entre los usuarios y permitir la transferencia de archivos.

La mayor parte del código es la traducción de la implementación original desde el lenguaje C# a Java para ser utilizado en Android y otros sistemas operativos. Esta traducción fue inicialmente desarrollada por Nicolás Malbrán (estudiante de *Ing. Civil en Computación*, *FCFM*, *Universidad de Chile*) y fue corregida, completada y adaptada para funcionar en Android por el autor de esta memoria. Los cambios específicos serán mencionados en la siguiente sección llamada Implementación de la Solución, no obstante, algunos cambios realizados fueron la adaptación de algunas bibliotecas para codificar Objetos, la estandarización del envío y recepción de bytes en los sockets y enmendar errores como *busy waitings*[19], *time out*[20] en los sockets y otros menos significativos.

La arquitectura de HLMP para Android no difiere de la implementación original, salvo que el módulo de interoperabilidad no se implementó, sino que se delegó mediante una interfaz para implementar en Java, esto con el fin de que sea implementado dependiendo del sistema operativo. De esta manera, la aplicación HLMPConnect se encargará de implementar este comportamiento, pues es quien interactúa directamente con el sistema operativo Android.

4.2.2 Android-AdHoc

Es una aplicación para Android que permite iniciar una conexión WiFi del tipo Ad-Hoc, destacando entre ellos el Nexus One de HTC. Esta aplicación es una modificación minimalista, realizada por el autor de este trabajo de título, de la aplicación Bernacle.

La principal ventaja de esta aplicación es que posee una estructura simple y fácil de portar a otros proyectos, como es el caso del protocolo HLMP para Android. Consta principalmente de 4 elementos, entre ellos dos componentes de aplicación – un Activity para que el usuario comience o termine la conexión Ad-Hoc y un Service que administra dicha conexión – la extensión, mediante una subclase, del estado global Application para acceder y configurar los componentes de aplicación, y por último los procedimientos nativos en el lenguaje C y C++ que permiten la conexión Ad-Hoc.

La mayor parte de los cambios de Bernacle a Android-AdHoc constaron en remover elementos adicionales a la conexión Ad-Hoc, por ejemplo el control de usuarios, las tasas de transmisión, entre otras cosas. Otros cambios importantes fueron agregar condiciones que permitieron crear sockets en Android y la reestructuración del código para que este servicio pudiese ser portado y configurado en otro proyecto.

[19] Rutina que repetidamente verifica si una condición es verdadera sin realizar alguna acción
[20] Periodo de tiempo para que un sistema computacional gatille un evento si no realiza acciones

4.2.3 HLMPConnect

Es una aplicación para Android que, utilizando los dos proyectos anteriores, logra completar el protocolo HLMP de manera autónoma en este sistema operativo. Este desarrollo es el esqueleto para crear otras aplicaciones que utilicen HLMP. Además, utilizando los sub-protocolos de HLMP-Java, esta aplicación ofrece un chat y la transferencia de archivos entre los usuarios que se encuentren en el alcance de la MANET y soporten los sub-protocolos necesarios.

Consta de al menos cinco componentes de aplicación de Android del tipo Activity; uno para conectarse y desconectarse de la MANET, del mismo modo uno para ajustar los parámetros de la conexión Ad-Hoc de la MANET, otro para listar a los usuarios conectados, el siguiente para utilizar el chat y un quinto para administrar la transferencia de archivos. Para un mayor detalle de sobre las funcionalidades y la interfaz revisar Anexo E, Anexo F y Anexo G.

5 Implementación de la Solución

En esta sección se explicará cómo se desarrolló lo mencionado en la sección Diseño de la Solución para HLMP en Android, por lo tanto, se describirá el desarrollo de sus componentes denominadas, HLMP-Java, Android-AdHoc y HLMPConnect.

Este desarrollo fue realizado con el IDE de programación Eclipse 3.7.2, utilizando el control de versiones Git con la plataforma GitHub. Para automatizar el empaquetamiento de la aplicación para Android se utilizó el plugin ADT para el IDE Eclipse, el cual manipuló Android SDK r17 para compilar el código fuente. En el caso de las componentes nativas para Linux desde Android se utilizó manualmente el Android NDK r6 para compilarlas. El código fuente de la biblioteca (incluidas las versiones para Windows) HLMP puede ser adquirida desde el repositorio https://github.com/fvalverd/High-Level-MANET-Protocol.

Todas las pruebas fueron realizadas con los equipos HTC Nexus One (Android 2.3.6), Samsung Nexus S (Android 4.0.4) y Sony Ericsson Xperia X10 mini (Android 2.1), además de algunos laptops con Windows XP y Windows 7 utilizando la versión de HLMP ya desarrollada y estable.

A continuación se describirán las 3 componentes de Android HLMP.

5.1 HLMP-Java

Es la implementación del protocolo HLMP en el lenguaje Java, su creación fue una traducción literal (sin cambios de algoritmos ni estructura de clases) de la implementación de HLMP en lenguaje C# para Windows. Al igual que el anterior desarrollo, se puede acceder al código fuente desde la plataforma GitHub en https://github.com/fvalverd/HLMP-Java.

La razón de que esta implementación posea un repositorio a parte del repositorio principal de la biblioteca HLMP se debe a que si otro proyecto necesita utilizar esta biblioteca de HLMP en Java, por ejemplo HLMPConnect, utilizará este repositorio como sub-módulo de Git y tendrá acceso a sus actualizaciones y podrá efectuar cambios para agregarlos, dado que Git no permite obtener versiones controladas dentro del repositorio de los archivos ni directorios.

Este desarrollo se basó en el trabajo realizado por Nicolás Malbrán, en el contexto del curso Desarrollo de Aplicaciones Web perteneciente al DCC de la FCFM, Universidad de Chile. Consistió en traducir dos de los tres módulos de HLMP (Comunicación y Red) y dos de los tres sub-protocolos de comunicación (Ping y Chat) también de HLMP. Fue desarrollado y medianamente probado en el ambiente Linux (Ubuntu 11.04 - 32 bits) entre redes WiFi del tipo Infraestructura, en consecuencia, muchas consideraciones para las redes Ad-Hoc no fueron previstas (apertura de sockets, funcionamiento de routing[21], interoperabilidad con Windows, entre otras cosas).

De esta manera, lo realizado durante este trabajo de título sobre la componente heredada de HLMP, consistió en verificar y corregir su funcionamiento en redes Ad-Hoc en Linux y Android, adaptar algunas bibliotecas para ser usadas desde Android, corregir los errores encontrados, solucionar la implementación de la componente de Interoperabilidad de HLMP, corregir la interoperablidad con la implementación de HLMP para los sistemas Windows y por último agregar el sub-protocolo de transferencia de archivos faltante.

El primer cambio en esta biblioteca fue al momento de intentar utilizar el desarrollo con una aplicación para el sistema operativo Ubuntu 11.10 (una distribución de Linux) que utilizaba HLMP. Esta aplicación fue inicialmente desarrollada por Nicolás Malbrán y levemente modificada por el autor de este trabajo, con el fin de establecer un chat mediante HLMP. El problema encontrado fue que la biblioteca solo permitía iniciar la comunicación pero no desconectarse. La razón se debió a un *busy waiting* encontrado al momento de detener algunos *threads* que forman parte del módulo Comunicación de HLMP, dedicados a manejar los eventos.

[21] Encontrar la mejor ruta posible dada una topología de red para enviar paquetes

El error consistió en una implementación específica de la estructura de datos *queue*[22] (ver Anexo B) que no permitía funcionar de manera segura por medio de threads.

Posteriormente, se conectaron dos laptops, un Linux con la biblioteca en Java y el otro un Windows con la implementación original. Los problemas aparecieron instantáneamente, pues verificando el manejo con los sockets se encontró que estaban mal configurados el *time out* y el *time linger*[23] de los socket TCP, por lo tanto ocurrían dos problemas: la conexión se perdía al cabo de un par de segundos y/o no se completaba el envío de información. Para ser más exactos, solo estaba la intención de configurar el *time out*, sin embargo estaba configurado como un *time linger*. El arreglo fue ajustar los parámetros adecuadamente, el *time out* inicialmente 0 (no se cierra) y solo cuando fuera necesario se cambiaba a un valor por defecto, y con respecto al *time linger*, la configuración de HLMP contempla este valor, por lo tanto simplemente se utilizó.

Luego, al cambiar la topología de la red por una de tres equipos en donde cada equipo estaba conectado con los otros, ocurrían desconexiones solo en los equipos con Linux una vez que se enviaban mensajes, tanto mensajes sobre el sub-protocolo chat como en el ping. Por lo tanto, se comenzó a revisar el *ruteo* en la implementación de Java y se encontró que la implementación del algoritmo Dijkstra para caminos mínimos en grafos poseía un error leve pero grave, el cual impedía hacer *routing*. El problema fue que una propiedad de la estructura de datos Stack, de implementación en Java, utilizada para calcular un paso intermedio requería un parámetro adicional para funcionar y al no tenerlo entregaba valores nulos que a su vez generaban excepciones. Este arreglo fue sencillo pero difícil de detectar.

Tras esto, solo resto un cambio antes de comenzar a utilizar esta biblioteca en Android. Este cambio constó en mejorar el sistema de notificación de eventos en la biblioteca. La forma en que este desarrollo notifica los eventos es por la suscripción a la biblioteca HLMP que, haciendo uso del patrón de diseño de POO[24] Observer, notifica cambios a los objetos que implementaban la interfaz específica para el tipo de notificación. Sin embargo, las últimas notificaciones gatilladas desde que se solicitaba el término de conexión hasta que se efectuaba no eran realizadas, esto se debió a que la rutina que controlaba la notificación de eventos funcionaba como un thread y al momento de solicitar su interrupción no esperaba a que los eventos se gatillaran antes de terminar. La solución fue sincronizar el término de su labor.

De esta manera, teniendo una biblioteca medianamente probada, la siguiente tarea fue utilizar dicha biblioteca en el desarrollo de una aplicación para Android. Por lo tanto, el desarrollo de

[22] Estructura de datos caracterizada por ser una secuencia de elementos, en ella insertar elementos se hace por un lado de la secuencia y retirar elementos por el otro.
[23] Periodo de tiempo de un socket que una vez cerrado espera para enviar información almacenada.
[24] **P**rogramación **O**rientada a **O**bjetos

esta biblioteca fue paralelo a HLMPConnect, pues los ajustes se realizan en la medida que se necesitaron. Empero, a continuación se comentará sobre dichos cambios.

Al momento de importar este desarrollo como sub-módulo de Git en HLMPConnect, apareció solo una dependencia sin resolver. Esta dependencia tuvo relación con la forma en que se guardaban los parámetros de HLMP en memoria persistente. El método utilizado por la implementación original de HLMP fue codificar los objetos que almacenaban estos datos en XML. La traducción de Nicolás Malbrán fue adaptada para funcionar con la biblioteca *javax.xml.bind*, la cual no se encuentra presente en el soporte de Java para Android 2.3.6 (*Apache Harmony's SE 6*). Por lo tanto se tuvo que investigar sobre si era conveniente importar esta biblioteca o buscar otra. La resolución fue utilizar otra llamada *SimpleXML*, pues tanto el tamaño como las dependencias fueron mejores.

Desde este punto, ya fue posible comenzar a pensar en automatizar la creación de la red Ad-Hoc, es así entonces como surge el desarrollo de Android-AdHoc.

Con todo lo anterior modificado, se comenzó a probar la interoperabilidad con la implementación para Windows, en particular un laptop con Windows 7 y su respectiva aplicación. El sub-protocolo ping no tuvo problemas, funcionó correctamente, pero el sub-protocolo chat no funcionó, generando mensajes no legibles. La razón tardó un poco en ser clarificada, esta tuvo relación con el *endianness*[25] que hace uso C# en Windows, debido a que este lenguaje utiliza por omisión la nomenclatura *little-endian*[26] para almacenar su información, mientras que Java utiliza por omisión *big-endian*[27]. De esta manera, la solución consistió en adaptar el funcionamiento de HLMP-Java para codificar y decodificar los *chunks*[28] de bytes enviados y recibidos en *little-endian* (ver Anexo D).

El siguiente cambio tuvo relación con determinar qué hacer con el módulo de interoperabilidad de HLMP, quien se encarga de manipular el controlador WiFi para conectarse a la red MANET (iniciar conexión Ad-Hoc). Antes de verificar esto, tanto la aplicación de Linux como Android se comunicaban mediante redes WiFi tipo infraestructura, la limitante esencial fue que directamente desde el lenguaje Java y/o desde la API de Android no se puede manipular el controlador WiFi en Linux ni en Android para crear conexiones WiFi del tipo Ad-Hoc. Por lo tanto, este desarrollo depende directamente del sistema operativo en que se encuentre. Teniendo esto presente, no quedo otra opción más que delegar esta tarea a cada sistema operativo.

Debido a que este desarrollo tiene como objetivo soportar HLMP en Android, solo se implementó una solución concreta para Android, no obstante, se dejó una base para

[25] Formato en el que se almacenan los datos de más de un byte en la memoria de un computador
[26] Formato endianess en que la codificación de byte parte de la menos significativa a la más significativa
[27] Formato endianess en que la codificación de byte parte de la más significativa a la menos significativa
[28] Fragmentos de información que posee meta-información sobre su origen

posiblemente desarrollar el proceso de automatizar la conexión Ad-Hoc tanto en Linux como en MacOS (también este sistema operativo pues el desarrollo solo depende de Java). La solución fue proveer una API mediante una interfaz (ver Anexo C) que permite hacer cambios y obtener el estado de la conexión, en consecuencia, cada sistema operativo lo tiene que implementar. Además, se modificó el componente principal del módulo de Red de HLMP (ver Anexo C), para que hiciera uso de esta API y así automatizar el proceso de conectarse a la red Ad-Hoc.

Finalmente y con el fin de completar la traducción de la biblioteca original de HLMP, se implementó el sub-protocolo llamado FileTransfer, cuyo objetivo es el de permitir a los equipos compartir una lista de archivos compartidos y utilizando los identificadores de estos archivos descargarlos desde los otros dispositivos mediante solicitudes. La complejidad de implementar este protocolo radicó en sincronizar la recepción y envío de archivos, manejar los archivos en la memoria física de Android (lo cual se delegó mediante una interfaz, debido a que depende del sistema operativo), transformar en chunks de bytes los archivos y por último disponer de un sistema de notificación de eventos mediante el patrón de diseño POO Observer sobre los estados de transferencia de los archivos (open, transfer percent, failed y complete) tanto en la descarga como en la transferencia.

5.2 Android-AdHoc

Android-AdHoc un desarrollo minimalista basado en la aplicación Bernacle, dejando solo la capacidad de configurar algunos parámetros de la red Ad-Hoc y la interfaz para conexión y desconexión. Se puede acceder al desarrollo de la aplicación desde la plataforma GitHub desde el repositorio ubicado en https://github.com/fvalverd/Android-AdHoc.

Esta aplicación requiere tener permiso de super usuario en tiempo de ejecución, debido a que Android no soporta desde su API la conexión WiFi a redes Ad-Hoc ni la creación de estas, por lo tanto, debe ejecutar rutinas nativas del Sistema Operativo Linux (Android es un tipo de distribución de Linux) para configurar adecuadamente la conexión WiFi tipo Ad-Hoc en Android.

Las rutinas antes mencionadas están escritas en los lenguajes C , C++ y Bash. Las dos primeras son utilizadas desde Java por medio de JNI, mientras que la última es utilizada por las dos primeras. La compilación de estas rutinas se automatizó con el Android NDK mediante reglas de compilación (ver Anexo A), las cuales tienen como objetivo generar los binarios y bibliotecas de C o C++.

Estas rutinas, en parte, emulan el comportamiento de las herramientas de configuración de red en Linux como *ipconfig*, *iwconfig* y *wpa_supplicant*, además del manejo de algunos controladores de tarjetas inalámbricas en los dispositivos móviles (activación y desactivación de

los módulos del Kernel de Linux). Es por esta última característica que no todos los dispositivos pueden hacer uso de la aplicación, de hecho durante el desarrollo, una persona ajena al desarrollo intentó utilizar este proyecto con el dispositivo Galaxy S GT-I9003L y los resultados no fueron positivos. Sin embargo, gracias a la comprensión del comportamiento heredado de Bernacle se logró encontrar el error específico y proponer un par de soluciones para un posterior desarrollo de este trabajo.

Por otra parte, y de manera específica, este desarrollo comenzó desde un fork de Bernacle llamado olsr-mesh-tether, cuyo función es la de proveer de una conexión Ad-Hoc al protocolo MANET OLSR. La razón de partir desde este fork fue porque los desarrolladores de olsr-mesh-tether removieron de Bernacle algunos elementos dispensables para la creación de la red como el enrutamiento NAT, el protocolo de red DHCP y la encriptación de paquetes de red WEP. De esta manera, se aprovechó el trabajo realizado por ellos y solo se tuvo como objetivo minimizar la aplicación y ajustarla para ser utilizados en otros proyectos como HLMP para Android.

Resumiendo los cambios realizados, se listarán los más importantes a continuación:

- **Se eliminó todo lo relacionado con OLSR.**
  Rutinas de inicio, código fuente de OLSR, referencias al proyecto.
- **Se eliminaron las interfaces innecesarias.**
  Visualización de usuarios conectados y tráfico de la red.
- **Se agregó lo necesario para crear sockets y transmitir por UDP en una red WiFi Ad-Hoc.**
  Se automatizó la modificación dinámica la tabla IP del Linux para crear sockets UDP, utilizando la herramienta de red para Linux ip route. Para esto se agregó el comando: "*ip route add 224.0.0.0/4 dev ${brncl_if_lan}*"
  Esta herramienta establece las reglas de tráfico en los controladores de red.
- **Se mejoró la activación/desactivación del WiFi desde Android.**
  Al utilizar la aplicación, conectando o desconectado a una red Ad-Hoc, el estado del WiFi quedaba como UNKNOWN. Se mejoró en ambos casos.
- **Se mejoró los estados de la red WiFi Ad-Hoc.**
  Al intentar conectar la red y no tener éxito, no existía un estado que permitiera identificar la situación. Se agregó el estado "failed" y se ajustó el código para soportarlo, tanto en el reconocimiento como en la notificación.
- **Se ajustó la asignación IP como aleatoria en la máscara de red de HLMP**
  Utilizando la máscara 255.255.0.0 y los dos últimos elementos al azar.
- **Se modificó la estructura de clases y recursos.**
  Este es el cambio más importante para HLMP, debido a que fue pensado para formar la base de las aplicaciones que utilizando las otras componentes de Android HLMP permitiera funcionar en conjunto.

Finalmente, especificando el último cambio mencionado anteriormente, este desarrollo consta de 4 elementos de Android que se comunican entre sí, estos se listan a continuación:

- **Un Activity como clase abstracta** que ofrece la interfaz de usuario para comenzar la conexión. Además, deja a disposición de la subclase las operaciones necesarias tras la notificación de cambio de estado de la red Ad-Hoc.
- **Un Application** que por su estado global a lo largo de la aplicación Android, recibe la petición de comenzar la red Ad-Hoc e interactuar con el encargo de levantar la red, además notifica los cambios de red al Activity anterior, entre otras funcionalidades.
- **Un Service** que por su carácter de funcionar sin interfaz de usuario, gestiona la red WiFi Ad-Hoc y genera las notificaciones pertinentes. Hace uso de elementos nativos de Linux.
- **Un Activity** que permite configurar algunos parámetros de la red desde el Activity antes mencionado.

5.3 HLMPConnect

Esta es una aplicación para Android que utiliza la biblioteca HLMP-Java y reutiliza la aplicación Android-AdHoc para permitir la interacción entre smartphones con sistema operativo Android mediante la MANET HLMP que provee de los servicios ping, chat y transferencia de archivos. Ha sido probada con las versiones 2.1, 2.3.6 y 4.0.4 de Android, sobre los smartphones HTC Nexus One, Samsung Nexus S y Sony Ericsson Xperia X10 mini. El código fuente, al igual que los demás desarrollos de esta memoria, puede ser encontrado desde GitHub en la siguiente URL https://github.com/fvalverd/HLMPConnect.

Los principales desafíos en este desarrollo fueron generar la biblioteca HLMP para Android, mantener los threads de ejecución de cada componente de aplicación de Android por medio de Objetos Android Handler, independientemente de haber iniciado los Activities soportar la recepción de mensajes tanto para la interfaz de los sub-protocolos chat y transferencia de archivos, proveer de una aplicación que permitiera seleccionar archivos desde el sistema de archivos y finalmente crear un sistema para la medición de tiempos de conexión de transferencia de archivos.

De esta manera, el primer punto a tratar fue el acoplamiento del desarrollo Android-AdHoc. La forma de implementarlo fue que HLMPConnect extendiera tanto el estado global de aplicación (mencionado en la componente anteriormente descrita) como el Activity encargado de interacción de los usuarios para iniciar la red Ad-Hoc. Esto permitió parametrizar e inicializar el funcionamiento de la biblioteca HLMP-Java.

Luego de esto último, todo lo demás del desarrollo fue la interacción entre la biblioteca HLMP-Java y los Activities de HLMPConnect por medio del sistema de notificación de eventos y mensajes de HLMP-Java. Lo esencial de estos Activities, es que representan la interfaz de usuario para utilizar los sub-protocolos chat y transferencia de archivos.

Recordemos que cada componente de aplicación tiene su propio thread de ejecución dentro de la aplicación, del mismo modo, la API de Android no permite hacer cambios directos por medio de rutinas públicas, en consecuencia, la solución para realizar cambios desde fuera de los Activities fue utilizar Handlers para cada Activity. Por lo tanto, estos Handlers solo podían ser visibles una vez seleccionado los Activities, es decir una vez creados (hacer click en la pantalla sobre la pestaña del chat o la transferencia de archivos). Estos Handlers debieron ser agregados al estado global de aplicación para ser accedidos desde quienes fueran los encargados de administrar los eventos gatillados y mensajes.

Es así como surgen los objetos Manager, creados por el autor de esta memoria, para administrar los mensajes de los sub-protocolos y los eventos de los sub-protocolos y la biblioteca HLMP. Lo ideal hubiese sido que fueran los mismo Activities quienes administraran esta labor, pero como se explicó anteriormente, estos objetos solo existen una vez seleccionados en la interfaz, por lo tanto se perdería información relevante.

Ahora, de manera específica, la capacidad de compartir una lista de archivos requirió implementar la capacidad de seleccionar y borrar archivos desde el sistema de archivos de Android. El problema se presentó porque Android no posee en su API una aplicación para seleccionar archivos, pues Android, por su sistema de seguridad al momento de ejecutar aplicaciones, no está orientado a navegar por el directorio de archivos. De esta manera, el desarrollo se centró en buscar una aplicación para navegar en los archivos del dispositivo.

En beneficio de la aplicación, se encontró el desarrollo llamado FileDialog, el cual mediante un llamado semi-sincronizado utilizando *Intents* permitió navegar a lo largo de los directorios visibles del dispositivo y obtener la ruta absoluta del archivo seleccionado.

Por último, por medio de del sistema de eventos de HLMP-Java y del sub-protocolo de transferencia de archivos, se logró medir el tiempo en milisegundos en las etapas inicial y final tanto para el inicio de la conexión Ad-Hoc para ingresar a la MANET, como para la descarga de archivos. Así, se pudo generar cada registro con fecha y hora además de la información pertinente. Luego, al término de los procesos de evento antes mencionado, se adjuntó la información a un archivo, el cual almacena cada acción mencionada. El formato de datos puede ser revisado en Anexo H y Anexo I.

6 Resultados Obtenidos y Lecciones Aprendidas

En esta sección se resumirán los resultados obtenidos, mencionando inclusive aquellos que no se plantearon como objetivos. Luego se realizará una especie de autocrítica en donde se abordará las prácticas utilizadas y fundamentará su realización durante el desarrollo.

6.1 Resultados Obtenidos

El primer logro fue representar una biblioteca estable para ser utilizada en Android utilizando el lenguaje de programación Java, permitiendo tener disponibilidad de esta biblioteca tanto en Linux como en Android.

El segundo logro fue crear una aplicación para Android que por medio de la biblioteca HLMP-Java pudiese comunicarse entre dispositivos con el mismo sistema operativo Android y las implementaciones de HLMP para Windows, más específicamente, corroborado tanto en Windows XP como en Windows 7, pero no así para Windows Mobile pues la disponibilidad de equipos no lo permitió.

Además, experimentalmente se midió el rendimiento de la biblioteca HLMP-Java por medio de la aplicación HLMPConnect en dos ámbitos: tiempo promedio de conexión a la MANET y velocidad de transferencia en dos topologías de red.

El primer hito se midió con tres smartphones con Android, estos fueron HTC Nexus One (Android 2.3.6), Samgung Nexus S (Android 4.0.4) y Sony Ericsson Xperia X10 mini (Android 2.1). Estos tiempos fueron medidos con la implementación mencionada en la sección anterior por parte de la aplicación HLMPConnect. En el caso del equipo HTC Nexus One, se tuvo un tiempo promedio de 4,07 segundos (ver Anexo H). Esto experimentalmente es un gran logro debido a que las implementaciones para Windows a veces tardaron hasta 2 minutos (dato corroborado por Juan Francisco Rodríguez, creador de HLMP).

Del mismo modo, el equipo Samsung Nexus S tuvo un promedio de 3 segundos y el Sony Ericsson Xperia X10 mini un promedio de 7 segundos. Esto se puede explicar por las mejoras de software y hardware en estos equipos, debido a que a la última versión de Android a esta fecha (Android 4.0.4) posee mejoras considerables en todo aspecto del sistema operativo. De igual manera, el otro equipo con sistema operativo Android 2.1 tardó el doble de tiempo que el equipo HTC Nexus One. No obstante, la razón de que estos tiempos sean menores que en la implementación para Windows, se debe a dos razones de fondo. Primero, la implementación de la red Ad-Hoc en Android no solicita al sistema operativo el recurso de red, sino que, lo apaga en

caso de estar encendido y luego lo manipula mediante rutinas de bajo nivel autónomamente. Segundo, esta implementación no sigue el estándar de las redes Ad-Hoc, el cual dice que solo comienza a transmitir una vez que otro equipo este en el mismo canal[29], la misma ESSID [30] buscando pares. Por lo tanto el tiempo se reduce considerablemente.

El segundo hito tratado mostró el mismo resultado para todos los equipos en la primera topología de red que constó solo de dos equipos. Esta topología presentó una rapidez de 124,02 KB/seg (ver Anexo I). Es un valor que en promedio es menor que el de la implementación original para los Windows, esto se explica debido a varios motivos sobre la implementación de HLMP-Java, desde el ruido generado por la compatibilidad *endianness*, como por no tener optimizaciones ligadas al computo paralelo y sincronizado sobre las descargas y transferencias en el sub-protocolo de transferencia de archivos, además de las condiciones generadas por el procesamiento en Android.

Por otra parte, con la segunda topología hubo resultados desfavorables en cuanto a la transferencia de archivos pero favorables el sub-protocolo chat. Esta topología de red se compuso de tres dispositivos en línea, es decir, un equipo intermedio que se conecta a los otros dos directamente, pero los otros dos directamente solo al intermedio.

El primer caso fue tener como dispositivo intermedio a uno con la implementación para Android. Dado este escenario, se realizó la petición de un archivo de aproximadamente 10MB desde uno de los equipos de un extremo al otro extremo, sin embargo, la petición no llegó a destino. No obstante, por medio del sub-protocolo chat los mensajes si llegaron a destino, estableciéndose una comunicación fluida. Se intentó luego la misma petición de archivo, esta vez con un archivo más liviano de aproximadamente 10KB, pero la solicitud tampoco llegó a destino.

El segundo caso fue tener como dispositivo intermedio a uno con la implementación para Windows 7. Dado este escenario, se realizó la misma petición y el tiempo promedio de transferencia fue de 64 KB/seg. Esta menor tasa de transferencia se puede explicar por el proceso de *routing* que se debe realizar la biblioteca HLMP.

[29] Frecuencia de ondas de radio por la cual está transmitiendo el controlador WiFi
[30] **E**xtended **S**ervice **S**et **I**Dentifier, también conocido como el nombre de la red WiFi

6.2 Lecciones Aprendidas

Durante este desarrollo ocurrieron acontecimientos que pudieron ser manejados de otra manera o investigados un poco más antes de ser realizados. Es por esto que en esta sub-sección se pretende a continuación mencionar algunos de ellos.

El primero de ellos dice relación con las redes WiFi tipo Ad-Hoc, antes de este desarrollo no había existido manipulación de este tipo de redes por parte del memorista, entonces existía el sesgo de creer que se parecían a las redes WiFi tipo Infraestructura. En la etapa inicial del desarrollo se perdió mucho tiempo en entender el comportamiento que debía tener cada equipo en la topología de red. Lo importante fue comprender que bastaba con transmitir en un canal (frecuencia) con un mismo ESSID (nombre de la red) y según el estándar esperar a que la topología de red tuviese más de un nodo para funcionar. Del mismo modo, se supo que no siempre se sigue el estándar, por ejemplo esta implementación para Android no espera a usuarios para transmitir.

El segundo hito, dice relación con el estado del hardware de los equipos para realizar pruebas. Por confiar en que un laptop con Windows XP poseía en buen estado su controlador WiFi, se perdió aproximadamente una semana intentando buscar un error inexistente.

El tercer tema corresponde a las alternativas que se posean para solucionar un problema con software de terceros. Este fue el caso al momento de decidir sobre qué desarrollo reutilizar para manipular y tener acceso a las redes WiFi del tipo Ad-Hoc. En primera instancia se había escogido la alternativa *Wifi Tether*, debido a que fue el primer desarrollo conocido sobre el tema, sin embargo, al momento de utilizarla, tras un tiempo considerable de inspección en el código, el memorista se percató que esta era una versión anterior a la probada, la cual era incompatible con el Smartphone HTC Nexus One y que la versión útil no poseía el código abierto hasta ese entonces para ser utilizado. Por fortuna, en la etapa de reconocimiento de alternativas, se conoció una segunda herramienta llamada Bernacle, que sin argumentos en ese momento, trajo una serie de beneficios como ahorrar tiempo de compilación de las rutinas nativas utilizando el Android NDK y no tener que compilar la versión de Android completamente (un par de horas como mínimo) para tener los binarios necesarios, como fue el caso de la primera alternativa.

Otro punto, no menos importante, fue el poco registro de los cambios realizados (salvo el control de versiones, que hubo que leerlo detenidamente) para escribir este documento y para no repetir las posibles soluciones fallidas a problemas similares. Si bien es cierto, no es un punto crítico, no obstante hubiese optimizado el tiempo en algunos momentos. Esto se pudo haber realizado con el uso de una wiki, la cual permite búsqueda a lo largo de los documentos y posee los servicios esenciales para este tipo de labores.

Finalmente, un tema no menor, fue la carencia de TDD[31] durante el desarrollo, esto toma mucho sentido a la hora de medir la calidad del software desarrollado y/o utilizado, pues al momento de hacer ajustes o generar nuevas características se tuvo que corroborar manualmente cada servicio, asegurándose de que no se estropeara una componente en pos del desarrollo de otra. Pese a que casi siempre se indica que TDD es la mejor opción para desarrollar, en este caso se decidió desde un comienzo no utilizarlo, por razones como el hecho de utilizar software de terceros sin tests (con el fin de mejorarlo y expandirlo), sumado a la escasa experiencia sobre desarrollos en Android por parte del memorista y las constantes pruebas de nuevas herramientas. Un argumento secundario, que apoyó esta decisión, fue la estimación en las horas de trabajo, las cuales pronosticaron un tiempo mayor por utilizarlo, básicamente por la inexperiencia en el área de las comunicaciones (necesidad de Double-Patterns[32] tipo Mocks o Stubs) y el tiempo limitado que posee este trabajo de título, el cual no solo emplea tiempo para desarrollar, sino también, para investigar sobre temas específicos, reuniones de trabajo además de la escritura de este documento.

7 Conclusiones y Trabajo a Futuro

Positivamente se puede decir que fue un desarrollo exitoso, debido a que se logró el objetivo general por medio de los cuatro objetivos específicos planteados. Se logró disponer de una biblioteca de HLMP para ser usada en Android, prueba de ello es la creación de la aplicación HLMPConnect que permite comunicarse con otros dispositivos en la MANET por medio de un ping, chat y la transferencia de archivos. Del mismo modo, se logró tener interoperabilidad con las implementaciones originales de HLMP para Windows XP y Windows 7.

Adicionalmente, la aplicación no solo resultó funcionar en la versión 2.3.6 de Android, sino también, en la versión 2.1 y 4.0.4, soportando además del equipo HTC Nexus One a los equipos Samsung Nexus S y Sony Ericsson Xperia X10 mini. Del mismo modo, la biblioteca quedó disponible para ser utilizada en las distribuciones de Linux como Ubuntu 11.10.

Uno de los resultados, obtenidos mediante experimentación, indicó que esta implementación mejoró sustancialmente el tiempo necesario para que un dispositivo se pueda conectar a la MANET. De esta manera, se abre el abanico de posibilidades para nuevas aplicaciones que requieran un tiempo de respuesta relativamente corto, del orden de los 4 segundos promedio para la implementación de Android 2.3.6 y de 3 segundos promedio para Android 4.0.4.

[31] **T**est-**D**riven **D**evelopment
[32] Patrón de diseño para Tests sobre el estandar xUnit

Por otra parte, los mismos resultados experimentales, mostraron que aún queda trabajo por realizar, pues el problema mencionado sobre el *routing* en la transferencia de archivos, para la implementación de HLMP para Android, quedó pendiente de solucionar por motivos de tiempo. Sin embargo, es importante destacar que el logro es sustancial pese a este detalle.

Finalmente, se menciona el principal logro alcanzado, al cual dice relación sobre soportar las redes WiFi de tipo Ad-Hoc en la implementación de HLMP para Android, pese a no estar disponible desde la API de Android. Si bien es cierto, esto genera la limitante de necesitar tener acceso de super usuario en los dispositivos, sin embargo, esta limitante debe ser analizada en el desarrollo específico a realizar, pues dependiendo de eso puede ser despreciable o no.

Los resultados obtenidos en este trabajo de título fueron positivos en cuanto a sus objetivos, sin embargo, los desafíos a tratar sobre estos temas de conectividad pueden ser más grandes y detallados con el fin de lograr mejores y más grandes resultados. Desde esta perspectiva, se propone trabajar algunas áreas como fortalecer la biblioteca y la aplicación para Android desarrolladas, evaluar las próximas y actuales tecnologías o extensiones a utilizar, generar el soporte a otros sistemas operativos como Linux e iOS (laptops, tables, smartphones y el PDA iPod touch) y sin duda alguna, generar una mayor y mejor difusión sobre esta tecnología.

Para ser más específico con lo anterior, a continuación se lista una serie de posibles tareas.

- Implementar la biblioteca HLMP en Android como componentes de aplicación de Android tipo Services, con el fin de tener un servicio continuo y no depender de que el Activity esté visible en el dispositivo.
- Mejorar el rendimiento del sub-protocolo de transmisión de archivos en HLMP-Java (cómputo paralelo y seguro en la descarga y transmisión de archivos).
- Rediseñar algunas rutinas originales de HLMP en HLMP-Java, existen varios por códigos repetidos (falta aplicar patrones de diseño POO).
- Implementar una representación visual de la topología de la red mediante grafos, al igual que la implementación para Windows XP y 7 para la aplicación en Android.
- Implementar una configuración de parámetros específicos del funcionamiento de HLMP-Java en la aplicación en Android.
- Expandir la capacidad de almacenamiento de archivos recibidos en la memoria SD para la aplicación en Android.
- Determinar la licencia de la biblioteca en Android, debido a que se utilizan elementos licenciados con GPL, Apache y MIT.
- Implementar un sub-protocolo de transmisión de audio en HLMP-Java.
- Implementar un sub-protocolo de transmisión de video en HLMP-Java.

- Evaluar la posibilidad de expandir HLMP a conexiones WiFi del tipo WiFi-Direct, para así no depender del permiso de super usuario, además de tener los beneficios de este tipo de conexión como la encriptación y la capacidad de utilizar el WiFi independientemente.
- Implementar el módulo de interoperabilidad de HLMP para Linux (al menos Ubuntu).
- Implementar el módulo de interoperabilidad de HLMP para Mac OS X (se aprovecha del hecho de que está todo implementado en Java).
- Implementar la capacidad de cifrar los mensajes enviados por la MANET.

Bibliografía y Referencias

Ableson F. Collins C., Sen R. Unlocking Android: A Developer's Guide. Manning Publications, 2009.

Android-Developer Official documentation for Android developer. [en línea] - http://developer.android.com/index.html [Consulta 27/08/2012]

Bernacle Barnacle Wifi Tether lets you turn your Android phone into a wireless ad-hoc access point in three easy steps. [en línea] - https://play.google.com/store/apps/details?id=net.szym.barnacle [Consulta 27/08/2012]

Canalys. Google's Android becomes the world's leading smart phone platform - January 31, 2011. [en línea] - http://www.canalys.com/newsroom/google%E2%80%99s-android-becomes-world%E2%80%99s-leading-smart-phone-platform [Consulta 27/08/2012]

Corson S., Macker J. Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations. IETF RFC 2501. 1999.

Dyck, J. A Survey of Application-Layer Networking Techniques for Real-time Distributed Groupware. University of Saskatchewan, Technical Report HCI-TR-06-06, 2006.

Huang, C., Lan, K., Tsai, C. A survey of opportunistic networks. Proc. of the 22nd Intl. Conf. on Advanced Information Networking and Applications, pp. 1672–1677, IEEE Press, Washington, DC, USA, 2008.

Monares A., Ochoa S.F., Pino J.A., Herskovic V. Mobile Computing in Urban Emergency Situacions: Improving the Support to Firefighters in the Field. Expert Systems with Applications 38 (2), pp. 1255-1267, 2011.

Pettey C., Stevens H. Gartner Says Android to Command Nearly Half of Worldwide Smartphone Operating System Market by Year-End 2012 - Gartner, Inc. - April 7, 2011. [en línea] - http://www.gartner.com/it/page.jsp?id=1622614 [Consulta 27/08/2012]

Rodríguez-Covili J.F. Infraestructura de Trabajo Colaborativo Móvil para Inspección Técnica de Obras. Tesis de Magíster en Ciencias, Departamento de Ciencias de la Computación. Universidad de Chile. - 2011.

Rodríguez-Covili J.F., Ochoa S.F., Pino J.A. High level MANET protocol: Enhancing the communication support for mobile collaborative work. Journal of Network and Computer Applications 35(1), pp. 145-155, 2012.

Rodríguez-Covili, J.F. Ochoa, S.F., Pino, J.A., Messeguer, R., Medina, E., Royo, D. A Communication Infrastructure to Ease the Development of Mobile Collaborative Applications. Journal of Network and Computer Applications 34(6), pp. 1883-1893, 2011.

Aduanas, Importaciones Julio 2012, Servicio Nacional de Aduanas Gobierno de Chile. [en línea] - http://www.aduana.cl/prontus_aduana/site/artic/20070416/pags/20070416165920.html [Consulta 27/08/2012]

Tanenbaum A. Computer Networks. Prentice-Hall - 1996.

Vergara C. Extendiendo las redes sociales hacia un plano físico // Tesis de Ingeniería Civil mención Computación, Departamento de Ciencias de la Computación. Universidad de Chile.. - 2012.

WiFi-AdHoc-Enabler Arend, Tool for root/unroot Android devices in order to get privileges. [en línea] - https://play.google.com/store/apps/details?id=nl.arendmedia.wifiadhocenabler [Consulta 27/08/2012]

WiFi-Alliance Wifi-Direct. [en línea] - http://www.wi-fi.org/discover-and-learn/wi-fi-direct [Consulta 27/08/2012]

WiFi-Tether This program enables wifi-tethering for "rooted" handsets. [en línea] - https://play.google.com/store/apps/details?id=com.googlecode.android.wifi.tether [Consulta 27/08/2012]

Anexo A

```
1  LOCAL_PATH := $(call my-dir)
2
3  # libhardware_legacy stub
4  include $(CLEAR_VARS)
5
6  LOCAL_MODULE    := libhardware_legacy
7  LOCAL_SRC_FILES := hardware_legacy_stub.c
8
9  include $(BUILD_SHARED_LIBRARY)
10
11 # the main binary
12
13 include $(CLEAR_VARS)
14
15 LOCAL_MODULE := wifi
16
17 LOCAL_SRC_FILES := main.cc
18 LOCAL_CPP_EXTENSION := .cc
19 LOCAL_CFLAGS := -Wall -Wextra -Werror -O3
20 LOCAL_C_INCLUDES := $(LOCAL_PATH)/../include
21
22 LOCAL_SHARED_LIBRARIES := libhardware_legacy
23 LOCAL_LDLIBS := -llog
24
25 include $(BUILD_EXECUTABLE)
```

Anexo B

```
public synchronized Event draw()
{
        while (itemCount == 0)
        {
                try {
                        wait();
                } catch (InterruptedException e) {
                }
        }

        ...
}
```

Con problemas

```
public synchronized Event draw()  throws InterruptedException
{
        while (this.itemCount == 0)
        {
                wait();
        }
        ...
}
```

Arreglo

Anexo C

```
Boolean connect = false;
int timeOutIpChange = 0;
while (!connect) {
    try {
        log("NETHANDLER: connect adhoc...");
        wifiHandler.connect();
        connect = true;
        log("NETHANDLER: connect adhoc... OK");
    }
    catch (Exception e) {
        e.printStackTrace();
        log("NETHANDLER: connect adhoc... FAILED! " + e.getMessage());
        timeOutIpChange++;
        if (timeOutIpChange > netData.getWaitForStart()) {
            throw new Exception("timeout, adhoc can't start");
        }
        Thread.sleep(netData.getWaitTimeStart());
    }
}

while (wifiHandler.getConnectionState() == WifiConnectionState.STOP) {
    log("NETHANDLER: waiting request to AdHoc...");
    Thread.sleep(1000);
    log("NETHANDLER: waiting request to AdHoc...");
}

// Wait for connect
while (wifiHandler.getConnectionState() != WifiConnectionState.CONNECTED) {
    log("wifiHandler.state=" + wifiHandler.getConnectionState());
    if (wifiHandler.getConnectionState() == WifiConnectionState.FAILED ||
            wifiHandler.getConnectionState() == WifiConnectionState.STOP) {
        netHandlerState = NetHandlerState.STOPFORCED;
        throw new Exception("adhoc Failed!");
    }
    Thread.sleep(1000);
    log("wifiHandler.state=" + wifiHandler.getConnectionState());
}
log("wifiHandler.state=" + wifiHandler.getConnectionState());
```

Cambio en el módulo de red de HLMP

```
public interface WifiHandler {

    public void connect();
    public void disconnect();
    public int getConnectionState();
    public int getIpState();
    public InetAddress getInetAddress();
}
```

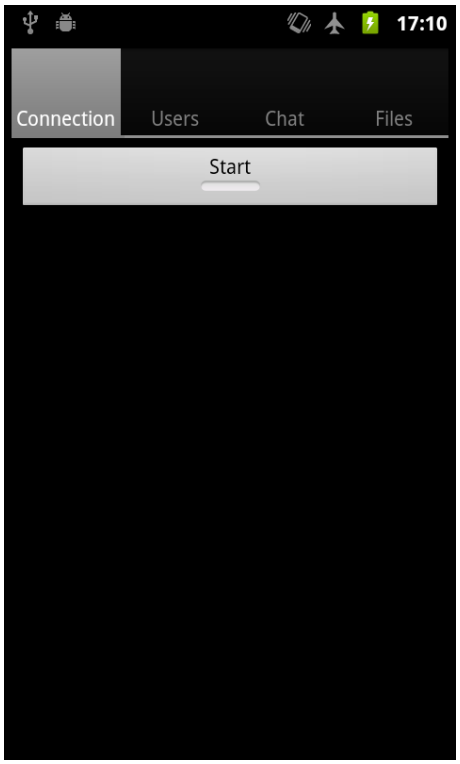
API a modo de Interfaz para el Módulo de Interoperabilidad

Anexo D

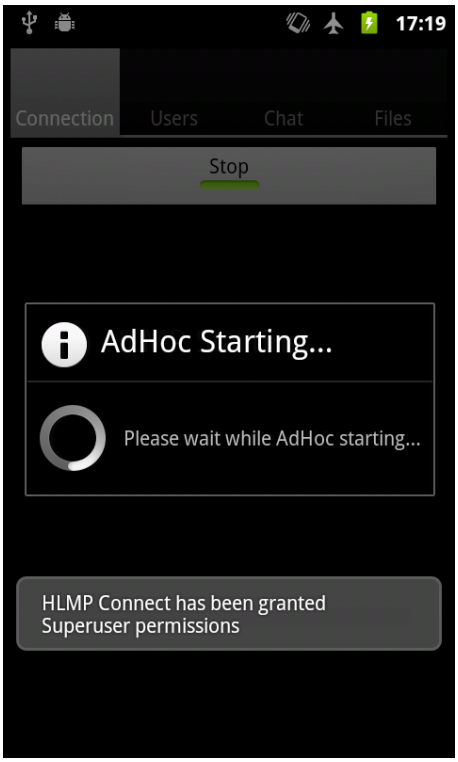
```
public static final byte[] stringToByte(String s) {
    try {
        return s.getBytes("UTF-16LE");
    } catch (UnsupportedEncodingException e) {
        return null;
    }
}

public static final String byteToString(byte[] b) {
    try {
        return new String(b, "UTF-16LE");
    } catch (UnsupportedEncodingException e) {
        return null;
    }
}
```

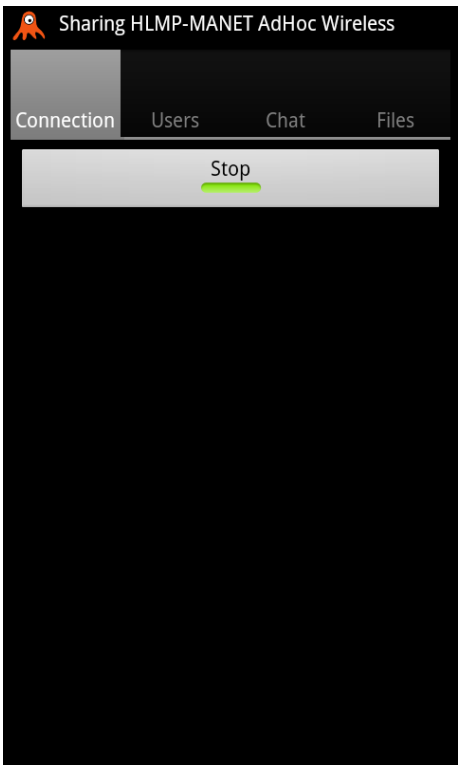
```
public static byte[] UUIDtoBytes(UUID id) {
    ByteBuffer byteBuffer = ByteBuffer.wrap(new byte[16]);
    byteBuffer.order(ByteOrder.LITTLE_ENDIAN);
    byteBuffer.putLong(id.getMostSignificantBits());
    byteBuffer.putLong(id.getLeastSignificantBits());
    return byteBuffer.array();
}

public static UUID bytesToUUID(byte[] bits) {
    ByteBuffer bb = ByteBuffer.wrap(bits);
    bb.order(ByteOrder.LITTLE_ENDIAN);
    UUID uuid = new UUID(bb.getLong(), bb.getLong());
    return uuid;
}

public static void writeLong(long datum, byte[] dst, int offset) {
    ByteBuffer buffer = ByteBuffer.allocate(8).order(ByteOrder.LITTLE_ENDIAN);
    buffer.putLong(datum);
    byte[] datumToByteArray = buffer.array();

    for (int i=0; i<8; ++i) {
        dst[offset+i] = datumToByteArray[i];
    }
}

public static final long readLong(byte[] src, int offset) {
    ByteBuffer bb = ByteBuffer.wrap(src, offset, 8).order(ByteOrder.LITTLE_ENDIAN);
    return bb.getLong();
}
```

Anexo E

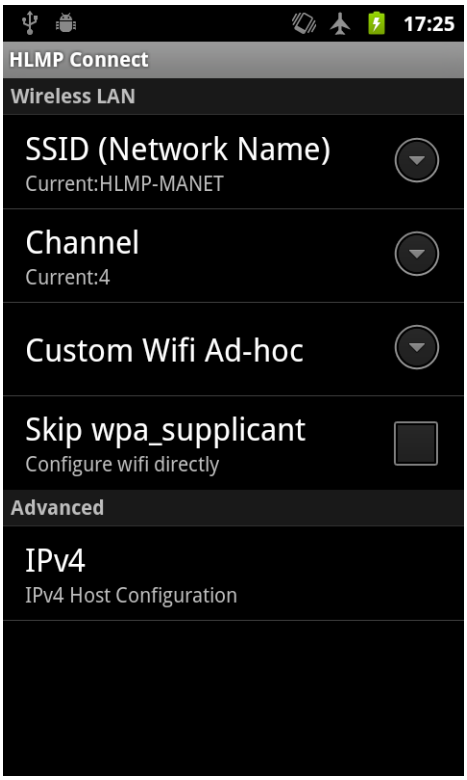

17:10
Connection
Users
Chat
Files
Start

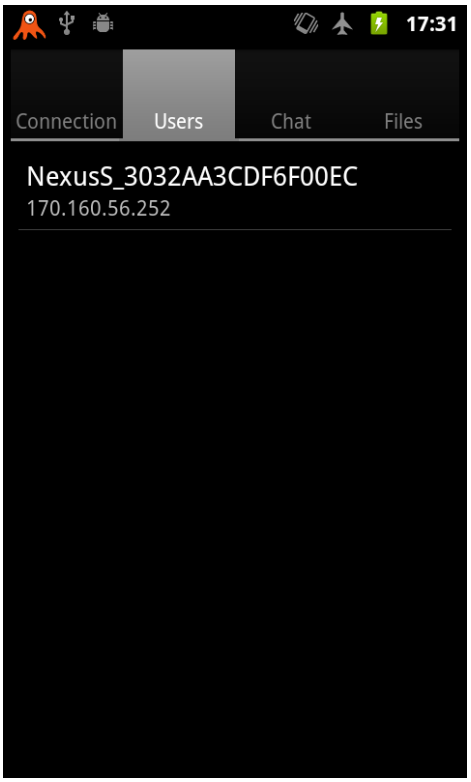

17:11
Connection
Users
Chat
Files
Stop
MANET Username
NexusOne_HT0BEP800261
Connect
Cancel

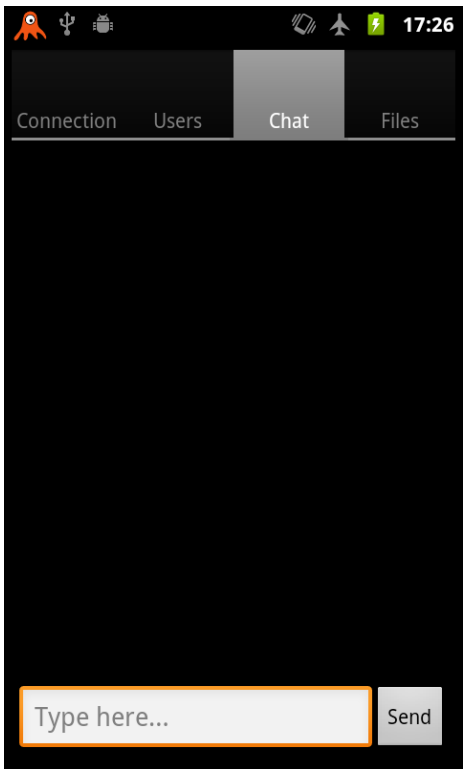

17:19
Connection
Users
Chat
Files
Stop
AdHoc Starting...
Please wait while AdHoc starting...
HLMP Connect has been granted
Superuser permissions

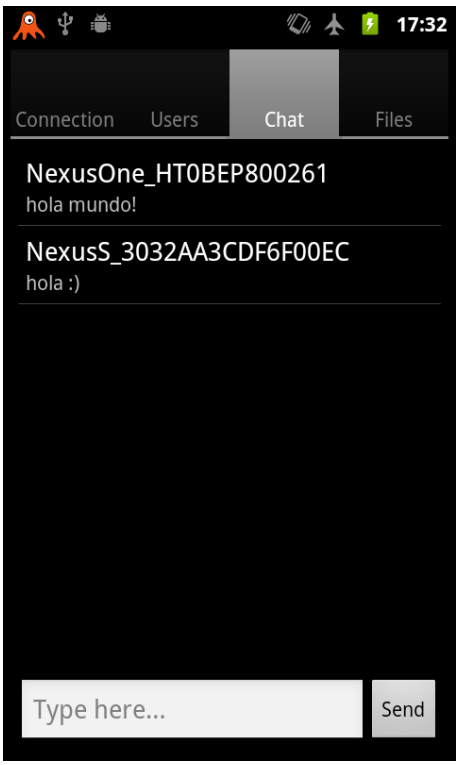

Sharing HLMP-MANET AdHoc Wireless
Connection
Users
Chat
Files
Stop

Anexo F

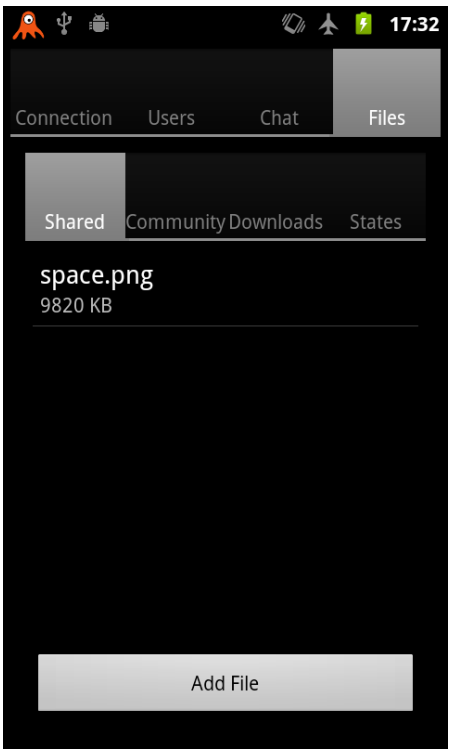

17:25
HLMP Connect
Wireless LAN
SSID (Network Name)
Current:HLMP-MANET
Channel
Current:4
Custom Wifi Ad-hoc
Skip wpa_supplicant
Configure wifi directly
Advanced
IPv4
IPv4 Host Configuration

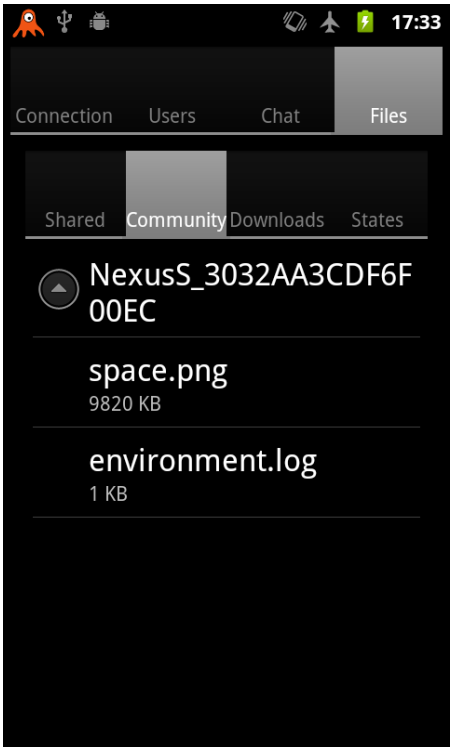

17:31
Connection
Users
Chat
Files
NexusS_3032AA3CDF6F00EC
170.160.56.252

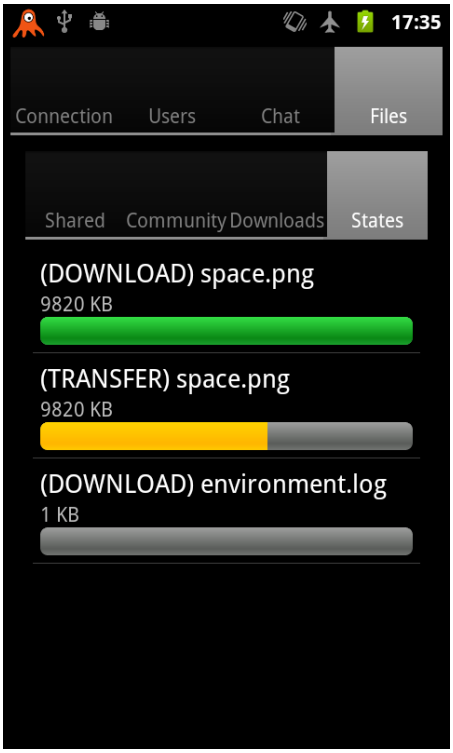

17:26
Connection
Users
Chat
Files
Type here...
Send

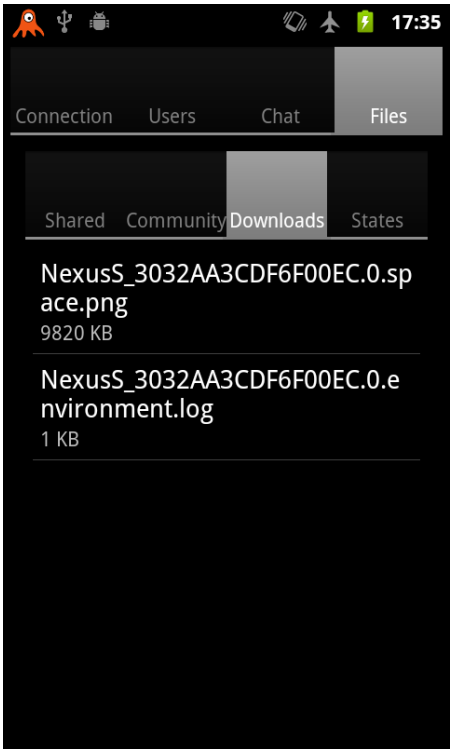

17:32
Connection
Users
Chat
Files
NexusOne_HT0BEP800261
hola mundo!
NexusS_3032AA3CDF6F00EC
hola :)
Type here...
Send

Anexo G


17:32
Connection
Users
Chat
Files
Shared
Community
Downloads
States
space.png
9820 KB
Add File


17:33
Connection
Users
Chat
Files
Shared
Community
Downloads
States
NexusS_3032AA3CDF6F00EC
space.png
9820 KB
environment.log
1 KB


17:35
Connection
Users
Chat
Files
Shared
Community
Downloads
States
(DOWNLOAD) space.png
9820 KB
(TRANSFER) space.png
9820 KB
(DOWNLOAD) environment.log
1 KB


17:35
Connection
Users
Chat
Files
Shared
Community
Downloads
States
NexusS_3032AA3CDF6F00EC.0.space.png
9820 KB
NexusS_3032AA3CDF6F00EC.0.environment.log
1 KB

Anexo H

Fecha	Tiempo (seg)
Wed Jul 11 21:17:33 GMT-04:00 2012	4,051
Wed Jul 11 21:17:44 GMT-04:00 2012	4,077
Wed Jul 11 21:17:53 GMT-04:00 2012	4,057
Wed Jul 11 21:18:05 GMT-04:00 2012	4,062
Wed Jul 11 21:48:25 GMT-04:00 2012	4,063
Wed Jul 11 21:53:24 GMT-04:00 2012	4,069
Wed Jul 11 21:59:56 GMT-04:00 2012	4,081
Wed Jul 11 22:03:02 GMT-04:00 2012	4,102
Wed Jul 11 22:06:43 GMT-04:00 2012	4,092
Wed Jul 11 22:08:20 GMT-04:00 2012	4,08
Wed Jul 11 22:12:04 GMT-04:00 2012	4,087
Wed Jul 11 22:21:32 GMT-04:00 2012	4,04
Wed Jul 11 22:22:32 GMT-04:00 2012	0,054
Wed Jul 11 22:33:40 GMT-04:00 2012	4,069
Wed Jul 11 22:34:39 GMT-04:00 2012	4,064
Wed Jul 11 23:16:13 GMT-04:00 2012	4,077
Wed Jul 11 23:22:05 GMT-04:00 2012	4,076
Wed Jul 11 23:34:02 GMT-04:00 2012	4,076
Thu Jul 12 01:07:17 GMT-04:00 2012	4,088
Thu Jul 12 09:55:00 GMT-04:00 2012	7,073
Thu Jul 12 10:23:32 GMT-04:00 2012	4,084
Thu Jul 12 10:29:34 GMT-04:00 2012	4,099
Thu Jul 12 10:43:39 GMT-04:00 2012	4,062
Thu Jul 12 10:47:35 GMT-04:00 2012	4,094
Thu Jul 12 12:12:01 GMT-04:00 2012	4,09
Thu Jul 12 12:25:26 GMT-04:00 2012	4,077
Thu Jul 12 12:33:00 GMT-04:00 2012	4,073
Thu Jul 12 12:49:35 GMT-04:00 2012	4,079
Thu Jul 12 12:58:04 GMT-04:00 2012	4,073
Thu Jul 12 12:58:41 GMT-04:00 2012	4,069
Thu Jul 12 12:59:13 GMT-04:00 2012	4,089
Tue Jul 17 17:11:51 GMT-04:00 2012	5,065
Tue Jul 17 17:12:11 GMT-04:00 2012	4,097
Tue Jul 17 17:12:21 GMT-04:00 2012	4,074
Tue Jul 17 17:13:05 GMT-04:00 2012	4,072
Tue Jul 17 17:13:52 GMT-04:00 2012	4,052
Tue Jul 17 17:14:05 GMT-04:00 2012	4,067
Tue Jul 17 17:14:20 GMT-04:00 2012	4,049
Tue Jul 17 17:14:37 GMT-04:00 2012	4,07
Tue Jul 17 17:14:49 GMT-04:00 2012	4,069
PROMEDIO (40 medidas)	**4,07355**

Anexo I

Fecha	Tamaño (Bytes)	Tiempo (seg)	Rapidez (KB/seg)
Thu Jul 12 10:41:36 GMT-04:00 2012	9820,7431640625	79,414	123,6651366769
Thu Jul 12 12:37:58 GMT-04:00 2012	9820,7431640625	79,011	124,2958975847
Thu Jul 12 12:51:12 GMT-04:00 2012	9820,7431640625	79,019	124,2833136848
Thu Jul 12 13:01:05 GMT-04:00 2012	9820,7431640625	79,14	124,0932924446
Thu Jul 12 13:02:28 GMT-04:00 2012	9820,7431640625	79,041	124,2487210949
Thu Jul 12 13:07:03 GMT-04:00 2012	9820,7431640625	79,063	124,2141477564
Thu Jul 12 13:13:46 GMT-04:00 2012	9820,7431640625	79,032	124,2628702812
Thu Jul 12 13:15:10 GMT-04:00 2012	9820,7431640625	80,038	122,7010065727
Thu Jul 12 13:16:34 GMT-04:00 2012	9820,7431640625	79,04	124,2502930676
Tue Jul 17 17:35:19 GMT-04:00 2012	9820,7431640625	79,018	124,284886533
PROMEDIO (10 medidas)			**124,0299565697**

Anexo J

```
diff --git a/ctrl_iface.c b/ctrl_iface.c
index ef93533..5accae6 100644
--- a/ctrl_iface.c
+++ b/ctrl_iface.c
@@ -28,6 +28,13 @@
 #include "wpa_ctrl.h"
 #include "eap.h"

+#define ANDROID_IBSS_HACK
+
+#ifdef ANDROID_IBSS_HACK
+/// NOTE: don't confuse WifiService.parseScanResult
+#define ANDROID_IBSS_PREFIX "(*)"
+#define ANDROID_IBSS_PREFIX_LEN 3
+#endif

 static int wpa_supplicant_global_iface_interfaces(struct wpa_global *global,
                                                   char *buf, int len);
@@ -230,6 +237,13 @@ static int wpa_supplicant_ctrl_iface_status(struct wpa_supplicant *wpa_s,
                                                 ssid_len = _res;
                                         _ssid = ssid_buf;
                                 }
+#ifdef ANDROID_IBSS_HACK
+                                if (ssid->mode == IEEE80211_MODE_IBSS)
+                                        ret = os_snprintf(pos, end - pos, "ssid=%s%s\nid=%d\n",
+                                                          ANDROID_IBSS_PREFIX, wpa_ssid_txt(_ssid, ssid_len),
+                                                          ssid->id);
+                                else
+#endif
                                 ret = os_snprintf(pos, end - pos, "ssid=%s\nid=%d\n",
                                                   wpa_ssid_txt(_ssid, ssid_len),
                                                   ssid->id);
@@ -574,12 +588,14 @@ static int wpa_supplicant_ctrl_iface_scan_results(
                                         return retpos - buf;
                                 pos += ret;
                         }
+#ifndef ANDROID_IBSS_HACK
                         if (res->caps & IEEE80211_CAP_IBSS) {
                                 ret = os_snprintf(pos, end - pos, "[IBSS]");
                                 if (ret < 0 || ret >= end - pos)
                                         return retpos - buf;
                                 pos += ret;
                         }
+#endif
                         if (!res->wpa_ie_len && !res->rsn_ie_len) {
                                 ret = os_snprintf(pos, end - pos, "\t");
                                 if (ret < 0 || ret >= end - pos)
@@ -587,6 +603,12 @@ static int wpa_supplicant_ctrl_iface_scan_results(
                                 pos += ret;
                         }

+#ifdef ANDROID_IBSS_HACK
+                        if (res->caps & IEEE80211_CAP_IBSS)
+                                ret = os_snprintf(pos, end - pos, "\t%s%s",
+                                                  ANDROID_IBSS_PREFIX, wpa_ssid_txt(res->ssid, res->ssid_len));
+                        else
+#endif
                         ret = os_snprintf(pos, end - pos, "\t%s",
                                           wpa_ssid_txt(res->ssid, res->ssid_len));
                         if (ret < 0 || ret >= end - pos)
@@ -792,6 +814,21 @@ static int wpa_supplicant_ctrl_iface_set_network(
                         return -1;
                 }
```

```
+#ifdef ANDROID_IBSS_HACK
+	if (os_strcmp(name, "ssid") == 0) {
+		// check prefix
+		if ((value[0] == '"') && (os_strncmp(value+1, ANDROID_IBSS_PREFIX,
+		  ANDROID_IBSS_PREFIX_LEN) == 0)) {
+			if (wpa_config_set(ssid, "mode", "1", 0) < 0) {
+				wpa_printf(MSG_DEBUG, "CTRL_IFACE: failed to set IBSS on '%s'",
+					value);
+				return -1;
+			}
+			value += ANDROID_IBSS_PREFIX_LEN;
+			value[0] = '"';
+		}
+	}
+#endif
	if (wpa_config_set(ssid, name, value, 0) < 0) {
		wpa_printf(MSG_DEBUG, "CTRL_IFACE: Failed to set network "
		  "variable '%s'", name);
@@ -846,6 +883,11 @@ static int wpa_supplicant_ctrl_iface_get_network(
		return -1;
	}

+#ifdef ANDROID_IBSS_HACK
+	if ((os_strcmp(name, "ssid") == 0) && (ssid->mode == IEEE80211_MODE_IBSS))
+		os_snprintf(buf, buflen, "\"%s%s", ANDROID_IBSS_PREFIX, value+1);
+	else
+#endif
	os_snprintf(buf, buflen, "%s", value);
	buf[buflen - 1] = '\0';

diff --git a/events.c b/events.c
index bb5be64..c591f30 100644
--- a/events.c
+++ b/events.c
@@ -479,9 +479,12 @@ wpa_supplicant_select_bss(struct wpa_supplicant *wpa_s, struct wpa_ssid *group,
		}

		if (bss->caps & IEEE80211_CAP_IBSS) {
+//#ifdef ANDROID_IBSS_HACK // FIXME
+			if (ssid->mode != IEEE80211_MODE_IBSS) {
			wpa_printf(MSG_DEBUG, "   skip - "
				"IBSS (adhoc) network");
			continue;
+			}
		}

		selected = bss;
```