Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

DISEÑO DE UNA RED VOIP SOBRE PROTOCOLO BLUETOOTH

Miguel Ventura Pérez
Asesorado por el Phd. Ing. Edmundo Enrique Ruiz Carballo

Guatemala, febrero de 2010

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO DE UNA RED VOIP SOBRE PROTOCOLO BLUETOOTH

TRABAJO DE GRADUACIÓN
PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR:

MIGUEL VENTURA PÉREZ
ASESORADO POR EL PHD. ING. EDMUNDO ENRIQUE RUIZ CARBALLO

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO ELECTRÓNICO

GUATEMALA, FEBRERO DE 2010

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	Inga. Glenda Patricia García Soria
VOCAL II	Inga. Alba Maritza Guerrero de López
VOCAL III	Ing. Miguel Ángel Dávila Calderón
VOCAL IV	Br. Luis Pedro Ortíz de León
VOCAL V	Br. José Alfredo Ortíz Herincx
SECRETARIA	Inga. Marcia Ivónne Véliz Vargas

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

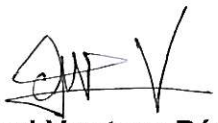
DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. José Antonio de León Escobar
EXAMINADOR	Ing. Julio Rolando Barrios Archila
EXAMINADOR	Inga. Magdalena Puente Romero
SECRETARIA	Inga. Marcia Ivónne Véliz Vargas

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la Ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO DE UNA RED VOIP SOBRE PROTOCOLO BLUETOOTH,

tema que se me fuera asignado por la Dirección de la Escuela Mecánica Eléctrica, el día 8 de julio de 2009.



Miguel Ventura Pérez

Guatemala, 16 de noviembre de 2009

Ingeniero
Julio César Solares Peñate
Coordinador del Area de Electrónica
Escuela de Mecánica Eléctrica
Facultad de Ingenieria
Universidad de San Carlos de Guatemala

Ingeniero Solares:

Por este medio me perfmito inforormale que he revisado el trabajo de graduación titulado "**Diseño de una red VoIP sobre protocolo Bluetooth**", desarrollado por el estudiante **Miguel Ventura Perez**, quien contò con mi asesoría.

Considero que el trabajo elaborado por el estudiante **Ventura Perez**, satisface los requisitos exigidos en la Facultad, por lo que recomiendo su aprobación.

Agradezco a usted la atención a la presente, atentamente

Ing. MsEE. PhD. Enrique Edmundo Ruiz Carballo
Colegiado No. 2225

FACULTAD DE INGENIERIA
Escuelas de Ingeniería Civil, Ingeniería Mecánica Industrial, Ingeniería Química, Ingeniería Mecánica Eléctrica, Técnica y Regional de Post-grado de Ingeniería Sanitaria.
Ciudad Universitaria, zona 12
Guatemala, Centroamérica

Guatemala, 24 de noviembre de 2009

Señor Director
Ing. Mario Renato Escobedo Martínez
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

Señor Director:

Por este medio me permito dar aprobación al Trabajo de Graduación titulado: **"DISEÑO DE UNA RED VoIP SOBRE PROTOCOLO BLUETOOTH"**, desarrollado por el estudiante **Miguel Ventura Pérez**, ya que considero que cumple con los requisitos establecidos.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,

ID Y ENSEÑAD A TODOS

Ing. Julio César Solares Peñate
Coordinador de Electrónica

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 01. 2010.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Área, al trabajo de Graduación del estudiante; Miguel Ventura Pérez titulado: DISEÑO DE UNA RED VoIP SOBRE PROTOCOLO BLUETOOTH", procede a la autorización del mismo.

Ing. Mario Renato Escobedo Martínez

GUATEMALA, 14 DE ENERO 2,010.

UNIVERSIDAD DE SAN CARLOS
FACULTAD DE INGENIERIA
DIRECCION ESCUELA DE INGENIERIA
MECANICA ELECTRICA
DE GUATEMALA

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

Ref. DTG.045-2010

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al trabajo de graduación titulado: **DISEÑO DE UNA RED VOIP SOBRE PROTOCOLO BLUETOOTH,** presentado por el estudiante universitario **Miguel Ventura Pérez,** autoriza la impresión del mismo.

IMPRÍMASE.

Ing. Murphy Olympo Paiz Recinos
Decano

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala, febrero de 2010

/cc
cc. archivo

ACTO QUE DEDICO A:

Dios	Por que con sus actos en mi vida me llenó de fuerza por medio de la fe.
Mi papá	Por darme amor, amistad y con su ejemplo trabajador, el mejor de los valores.
Mi mamá	Por ser el Angel que Dios escogió para cuidarme y guiarme en la vida.
Mi hermano	Por tantas alegrías que vivimos juntos y por mostrarme muchas veces que la vida no es tan complicada.
Laura María	Por ser el motivo de mi vida desde el momento que la conocí.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
AP	*Access Point*
GHz	*Giga Hertz*
GSM	*Global System Mobile communications*
Hz	*Hertz*
IP	*Internet Protocol*
Kbit	Kilobit
LAN	*Local Área Network*
PG	*Phone Gateway*
USB	*Universal Serial Bus*
WLAN	*Wireless Local Área Network*

GLOSARIO

Access Point (Punto de acceso). Dispositivo inalámbrico central de una *WLAN* que mediante sistema de radio frecuencia se encarga de recibir información de diferentes estaciones móviles para su centralización, o para su enrutamiento.

Ancho de banda Medida de capacidad de comunicación o velocidad de transmisión de datos de un circuito o canal analógico. Se mide en ciclos por segundo o hertzios (Hz). En transmisiones digitales, se mide en bits por segundo (bps) y cuanto más grande sea este número, más rápida será la transmisión.

Asíncrono Tipo de comunicación que envía datos usando control del flujo sin necesidad de sincronizar entre un terminal origen y un terminal destino.

Bit Unidad mínima de información que puede ser trabada por un ordenador.

Bluetooth Estándar de comunicación inalámbrica que utiliza FHSS, capaz de transmitir a velocidades de 1 Mbps a una distancia de 10 metros entre aparatos que implementen esta tecnología.

Dirección *IP* Dirección de 32 bits del protocolo Internet asignada a un ordenador conectado a Internet. La dirección *IP* tiene un componente del propio ordenador y un componente de la red. Este número tiene el formato de cuatro grupos de hasta tres dígitos, separados por un punto, por ejemplo 172.16.253.90.

Encriptación Proceso de cifrar la información para proteger su uso o visualización no autorizado durante el proceso de transmisión o cuando se guarda en algún medio transportable.

Gateway (Compuerta, pasarela). Programa o equipo que se encarga de traducir la información contenida en dos protocolos diferentes.

GSM (*Global System Mobile communications*). Sistema Global de Comunicaciones Móviles. Sistema digital de telecomunicaciones usado principalmente para telefonía móvil cuya principal virtud es la compatibilidad entre redes.

LINUX Sistema operativo multitarea y multiusuario de 32 bits para *PC* desarrollado inicialmente por Linus Toorvald, modificado y mejorado por programadores de todo el mundo. Su distribución es gratuita.

MODEM	(Modulador/Demodulador). Dispositivo que adapta las señales digitales para su transmisión a través de una línea analógica, normalmente telefónica
PDA	(*Personal Digital Assistant*). Asistente Personal Digital. Dispositivo de reducidas dimensiones y portátil que se utiliza para controlar tareas habituales del usuario, como la gestión de una agenda, un listado de direcciones o lista de cosas por hacer. También se pueden emplear para enviar faxes y mensajes de correo electrónico.
Protocolo	(*Protocol*). Conjunto de reglas y normas que determinan cómo se realiza un intercambio de datos, asegurando que los datos recibidos son idénticos a los datos enviados.
Router	Es el elemento que decide el camino más adecuado para la transmisión de mensajes en una red compleja, que está soportando un tráfico intenso de datos.
WLAN	(*Wireless Local Area Network*). Red de área local inalámbrica. También conocida como red *wireless*. Permite a los usuarios comunicarse con una red local o a Internet sin estar físicamente conectado. Opera a través de ondas y sin necesidad de una toma de red (cable) o de teléfono.

RESUMEN

En el presente trabajo de graduación se aborda el tema de diseño de una solución de telecomunicaciones para empresas pequeñas o medianas, es indispensable en la actualidad contar con un sistema de comunicación eficiente entre empelados. Al referirnos a eficiente debemos de tomar en cuenta parámetros tales como, costo, dificultad de implementación, durabilidad del sistema.

Se ha recabado información general para realizar este diseño, pero también se ha tomado en cuenta un modelo específico de dispositivo de usuario final, para fines prácticos de ilustración; es decir para no generalizar y por ende dar un ejemplo puntual de cómo realizar la configuración de este dispositivo, de tal manera que sea más fácil guiarse con otros modelos.

Las tecnologías utilizadas en esta investigación son actualmente de alta demanda en telecomunicaciones de bajo rango y a su vez poseen un bajo costo y una configuración sencilla; siendo los dos pilares de esta investigación las tecnologías *Bluetooth y VoIP*.

OBJETIVOS

General:

Implementar una comunicación entre dispositivos portátiles, utilizando *VoIP* sobre una red *Bluetooth* para realizar llamadas entre estos dispositivos.

Específicos:

1. Investigar sobre puntos de acceso (*Access Point*) *Bluetooth* y su conexión a una red de teléfono.

2. Implementar los parámetros en la red *Bluetooth* que se requieren para tener una comunicación de voz sobre protocolo de *internet*.

3. Lograr simplificar la programación de los dispositivos portátiles para que estos se integren a nuestra red.

4. Demostrar que este diseño es eficiente y así lograr reducir costos de llamadas en empresas para empleados móviles en un área limitada.

5. Realizar documentación e información sólida de nuestro diseño para que sea fácil su mantenimiento y su futura actualización.

INTRODUCCIÓN

El tema principal de este trabajo de graduación consiste en el diseño de una red *VoIP* sobre protocolo *Bluetooth*. Esto implica incluir un primer capítulo de inducción para plantear las tecnologías, dispositivos, y diseños existentes así como los que se utilizaran y mencionaran a través de este trabajo de graduación.

Se incluirán definiciones literales, explicaciones detalladas y algunos ejemplos de las dos tecnologías pilares en este trabajo de graduación, también se incluirán definiciones de versiones anteriores o paralelas que puedan ayudar a robustecer este trabajo.

También se incluirán los conceptos básicos y características principales de los dispositivos disponibles en el mercado actual. Ya que es sobre estos que se basa nuestro diseño. Al mismo tiempo se incluirán versiones de *software* y/o sistemas operativos; esto para los dispositivos que aplique.

Se propondrá un diseño de red, el cual a la vez de incluir definiciones y conceptos también incluirá una explicación del porque del uso de estas tecnologías sobre otras existentes.

Por último se incluye un capítulo que se basa casi en su totalidad en una investigación existente que se ha adaptado para nuestro diseño, del como programar el dispositivo especificado para montar la aplicación deseada.

1. INTRODUCCIÓN

1.1. Motivación

Junto con el lenguaje corporal, el lenguaje hablado es la forma más natural para la comunicación humana. Esto es, probablemente, la razón por la que la forma principal de comunicación rápida y efectiva dentro de una organización es la llamada de teléfono. Por lo general, cada empleado tiene un teléfono en su escritorio y las llamadas internas a nivel empresa no tienen costo. Sin embargo, si un empleado es altamente movilizado, lo anterior ya no es funcional. En estos casos, el empleado se equipa con un teléfono celular; lo que, usualmente, representa una solución costosa para la empresa u organización.

La mayoría de teléfonos celulares utilizan tecnología *GSM* para hacer llamadas. En entornos donde el empleado se mueve dentro de un área limitada, como un edificio, es una idea evidente reducir costos proporcionando una infraestructura local; pero no se puede simplemente comenzar a instalar estaciones *GSM* para tener acceso local. Sin embargo, se tiene que tener en cuenta que al instalar otra forma de radio comunicación se deben retener las propiedades de *GSM* en los teléfonos, esto, por cuestiones de flexibilidad de opciones.

La opción más obvia para radio comunicaciones es una Red inalámbrica de área local, o *WLAN*, por sus siglas en inglés, la cual provee más que el suficiente ancho de banda para la comunicación de voz. Hoy en día, es muy sencillo obtener una conexión inalámbrica dentro de una ciudad.

Desafortunadamente, pocos teléfonos móviles poseen una interfaz inalámbrica, pero sí la mayoría tienen interface *Bluetooth*, aunque ésta solamente proporciona una fracción del ancho de banda *WLAN*, pero sigue siendo suficiente para la comunicación de voz.

Más aún, muchos dispositivos móviles, tal como *PDAs*, tienen interfaz *Bluetooth* de salida y entrada de audio. Esto significa que la misma infraestructura que se usa para las llamadas telefónicas comunes se puede utilizar para nuestra aplicación.

1.2. Idea general

La idea principal es equipar un edificio determinado, con Puntos de Acceso *Bluetooth*. Estos proporcionan una red para los dispositivos móviles, tales como teléfonos celulares o *PDAs*. Los *Access Point* están conectados con el *Gateway* del teléfono utilizando una cableado de una red existente, por ejemplo, *Ethernet*. Un dispositivo móvil utiliza un *Access Point* para conectarse al *Gateway* del teléfono. A través de esta conexión, se crean y reciben llamadas telefónicas. Los datos de audio tienen que ser divididos en paquetes de datos antes de ser enviados al *Gateway* del teléfono. La opción obvia es utilizar *Voice over IP* o *VoIP*.

Incluso, esta infraestructura podría ser utilizada para enrutamiento inteligente de llamadas, en función de la fuerza de la señal o los servicios basados en localización.

1.3. Requerimientos

1.3.1. La aplicación

Desde el punto de vista de un usuario final, realizar una llamada utilizando *VoIP* debería ser lo más simple posible. Esto no debería ser más complicado que hacer una llamada telefónica ordinaria (*GSM*), ni tanto como hacer una llamada telefónica *VoIP* utilizando una computadora. Deberá tener muy pocas opciones de configuración como sea posible. La capacidad de hacer una llamada *VoIP* debe estar integrada en el teléfono como fichero en los ajustes, tan semejante con las demás opciones como sea posible. Si esto resulta ser inoperante o muy difícil, una aplicación debería ser provista tal que permita al usuario ingresar un número telefónico o seleccionarlo de una libro de direcciones con el fin de colocar una llamada *VoIP* o *GSM*.

Debería mostrar la intensidad de la señal disponible de las redes *GSM* y *Bluetooth*, y permitir al usuario hacer cualquier configuración necesaria. El usuario debe ser capaz de seleccionar si hacer una llamada *GSM* o una llamada *VoIP*, o si permitir al dispositivo móvil que tome la decisión.

1.3.2. La red

El dispositivo móvil debe detectar y conectarse automáticamente a los *Access Point* (AP). El aparato debe conectarse a múltiples *APs*, si es posible, con el fin de tratar apropiadamente la señal si la conexión con alguno se rompe. En condiciones óptimas, esto debe hacerse de tal manera que la llamada telefónica sea afectada mínimamente.

El *Gateway* de teléfono se encarga del enrutamiento de las llamadas telefónicas a su destino.

La red debe ser construida de tal manera que minimice la latencia, lo cual es crítico para las llamadas telefónicas. Todos los usuarios deben ser autenticados previo a realizar una llamada telefónica. La red también debe ser segura, esto significa que escuchas ilegales o manipulación de comunicación de usuarios no sea factible. Dependiendo de la elección del administrador, el acceso a internet de los dispositivos móviles es anónimo, autenticado o imposible.

1.3.3. El dispositivo

El dispositivo móvil debe tener interfaz *Bluetooth*, altavoz y micrófono. También necesita un método de entrada y salida amable al usuario. El dispositivo tiene que proporcionar suficiente capacidad informática para manejar *VoIP*.

El usuario debe ejecutar el código arbitrario en el dispositivo. El código de usuario debe tener el privilegio de crear y utilizar las conexiones *Bluetooth*, para acceder al altavoz y el micrófono y ejecutar una aplicación gráfica de usuario.

Además, es favorable que tenga soporte para Internet Protocol (*IP*), modificación de tablas de enrutamiento y soporte para la creación de conexiones encriptadas.

Desde que la modificación de *software* existente es más fácil, rápido y menos propenso a errores que escribir desde cero, un *software* de código abierto existente es deseable para el dispositivo móvil.

1.4. Diferencias entre *Bluetooth y Wi-Fi*

Bluetooth se utiliza principalmente en un gran número de productos tales como teléfonos, impresoras, módems y auriculares. Su uso es adecuado cuando puede haber dos o más dispositivos en un área reducida sin grandes necesidades de ancho de banda. Su uso más común está integrado en teléfonos y *PDA*, bien por medio de unos auriculares *Bluetooth* o en transferencia de ficheros. *Bluetooth* tiene la ventaja de simplificar el descubrimiento y configuración de los dispositivos, ya que éstos pueden indicar a otros los servicios que ofrecen, lo que redunda en la accesibilidad de los mismos sin un control explícito de direcciones de red, permisos y otros aspectos típicos de redes tradicionales.

Wi-Fi es similar a la red *Ethernet* tradicional y como tal el establecimiento de comunicación necesita una configuración previa. Utiliza el mismo espectro de frecuencia que *Bluetooth* con una potencia de salida mayor que lleva a conexiones más sólidas. A veces se denomina a *Wi-Fi* la "*Ethernet* sin cables". Aunque esta descripción no es muy precisa, da una idea de sus ventajas e inconvenientes en comparación a otras alternativas. Se adecua mejor para redes de propósito general: permite conexiones más rápidas, un rango de distancias mayor y mejores mecanismos de seguridad.

Las tecnologías inalámbricas *Bluetooth* y *Wi-Fi* son tecnologías complementarias.

Se espera que ambas tecnologías coexistan: que la tecnología *Bluetooth* sea utilizada como un reemplazo del cable para dispositivos tales como *PDAs*, teléfonos móviles, cámaras fotográficas, altavoces, auriculares etc. Y que la tecnología *Wi-Fi* sea utilizada para el acceso *Ethernet* inalámbrico de alta velocidad.

1.4.1. Aplicaciones

A continuación describiremos las aplicaciones de las tecnologías *Bluetooth y WLAN*:

- *Bluetooth* es una tecnología diseñada para comunicación entre dispositivos electrónicos. Es la evolución de la comunicación infrarroja.
- *Wi-Fi* es una tecnología de comunicación para accesar a internet de manera inalámbrica, o bien, a redes locales (*LAN*). En otras palabras, *Wi-Fi* es una *WLAN* (*Wireless Local Area Network*).
- *Bluetooth* puede ofrecer mayor versatilidad, en cambio *Wi-Fi* está mas adaptada a transferencias de datos de mayor volumen.
- *Bluetooth* tiene una velocidad de comunicación de 1Mb/seg, aproximadamente 20 veces la velocidad de una conexión telefónica a internet.
- *Wi-Fi* tiene una velocidad de conexión de 11 megabytes por segundo, 200 veces más rápido que una conexión telefónica.
- *Bluetooth* no se utiliza para conectarse a internet, salvo como “módem” para ordenadores portátiles utilizando redes *GPRS*.
- *Wi-Fi* no se utiliza para comunicación entre dispositivos

1.4.2. Comparación técnica

Tabla I. Diferencias técnicas entre *Bluetooth y WLAN*

Especificación Técnica	***Bluetooth***	***Wi-Fi***
Banda de radio	ISM 2.4 GHz	2.4 GHz
Frecuencia de salto	1,600 saltos/seg.	Sin saltos
Velocidad de transmisión de vos y datos	721 kb/seg.	11 Mbps (aunque la velocidad real de transmisión depende en última instancia del número de usuarios conectados a un *Access Point*)
Cobertura	Rango entre 50 - 250metros. Dependiendo del dispositivo *Bluetooth*.	Buena cobertura, de 300 a 400 metros con buena conectividad con determinados obstáculos.
Dispositivos con que trabaja	Alrededor de 600 dispositivos.	Alrededor de 450 aparatos.
Popularidad	Nueva tecnología, con un rápido crecimiento.	Adoptado masivamente.
Acceso público	El número de *Hotspots* crece exponencialmente.	El número de *Hotspots* crece exponencialmente
Consumo de batería	Aproximadamente, un quinto del consumo que utiliza *WLAN*.	-
Seguridad	128 bit encriptación	64 bit encriptación

Una *WLAN* cubre distancias mas grandes, mientras que el *Bluetooth* logra solo distancias de 10 metros aproximadamente, bastante reducida si queremos cubrir inalámbricamente una casa entera para llegar con internet hasta el ultimo rincón. *Bluetooth* esta diseñado especialmente para dispositivos y teléfonos celulares, para mouse y teclados inalámbricos. Si queremos cubrir distancias mayores con *Bluetooth*, habría que tener *Bluetooth* de Clase 1, los que llegarían a cubrir incluso 150 metros, pero son mas difíciles de conseguir y por ende también mas caros y los clase 2 y 3.

Lo mismo se aplica para tecnologías *Wi-Fi*, hay distintos tipos que cubren distintas distancias y también hay algunos más rápidos que otros.

1.5. Comparación de costos

Se compararon los costos de la implementación de una red *VoIP* sobre una *WLAN* y los costos de *VoIP* sobre *Bluetooth*.

1.5.1. *VoIP* sobre WLAN

Los requerimientos tecnológicos para el diseño e instalación de una red *VoIP* sobre *WLAN* son los siguientes:

- Chasis y procesador de llamadas 3Com *Super Stack* NBX 100:

  Es un componente esencial para una solución de telefonía en red, puede soportar hasta 200 dispositivos conectados entre aparatos telefónicos, buzones de voz, líneas PSTN y hasta 12 puertos para consolas de atención o Conmutadores, capacidad de almacenamiento de mensajes hasta de 80 horas y siendo de simple instalación y configuración.

- Teléfonos 3Com NBX1102:

  Cuenta con pantalla de despliegue de 16 caracteres por 2 líneas, 18 botones para programación y acceso fácil de funciones como: correo de voz, transferencia de llamadas, remarcado de llamada, parlante de dos vías y conectividad *Ethernet* de 10Mbps.

- Módulo de energía para teléfono 3Com NBX:

  Permite a un teléfono NBX recibir alimentación eléctrica sobre Ethernet transmitida por N3Com Network Jack o cualquier fuente de alimentación. Soporta cableado *Ethernet* de Categoría 5/5e sin afectar la velocidad o calidad del tráfico de voz.

- *Software* para conexiones de Gateway H.323 3Com NBX:

  Este *software* provee una permanente y disponible conexión al puerto H.323, brinda soporte al Estándar H.323 y dispositivos que puedan interactuar con los dispositivos de las soluciones 3Com NBX.

- *Software* para Administración NBX NetSet:

  NetSet permite a los usuarios individuales y a los administradores personalizar por completo la solución NBX, para satisfacer los requerimientos personales. Esta herramienta, basada en navegador y protegida por contraseña ofrece control de las capacidades utilizadas con mayor frecuencia, menús desplegables para cada utilidad además de añadir capacidades

Los costos de los requerimientos de *hardware y software*, para una red capaz de manejar 10 dispositivos móviles, se muestran en la siguiente tabla:

Tabla II. Costos de una red *VoIP* sobre *WLAN*

Cantidad	Marca/Modelo	Precio Unitario US($)	Sub-Total US($)
1	Chasis y Procesadores de llamadas 3Com NBX 100	15,000.00	15,000.00
10	Teléfonos 3Com NBX1102	150.00	1,500.00
10	Módulo de Energía para teléfono 3Com NBX	40.00	400.00
1	*Software* para administración NBX NetSet	5,000.00	5,000.00
1	*Software* para conexión de Gateway H.323 3Com NBX	5,000.00	5,000.00
TOTAL			**US($) 26,900.00**

1.5.2. *VoIP* sobre *Bluetooth*

Los requerimientos, tanto de *hardware*, como *software*, para el diseño e instalación de una red *VoIP* sobre protocolo *Bluetooth* son los siguientes:

- *Bluetooth Access Point*: El *Bluetooth Access Poi*nt marca Belkin, con Puerto USB, modelo F8T030, tiene un radio de *Bluetooth* Clase 1, y soporta tazas de transferencia arriba de los 723Kbps en un rango de los 100 metros. Soporta *Bluetooth* habilitado de PC's y Macs, Laptops, Pocket Pc's y PDA's

- Dispositivos electrónicos con *Bluetooth*: Pueden ser teléfonos celulares, *PDA's*, computadoras portátiles, etc. En la actualidad la mayoría de aparatos móviles de telefonía tienen incorporada la función de *Bluetooth*, por lo que en la integración de precios, se omite la compra de más dispositivos.

Figura 1. *Bluetooth* Access Point

- *Phone Gateway*: Se puede utilizar el *phone gateway* que la empresa pueda tener actualmente; si este fuera el caso, el valor inicial de inversión no se toma en cuenta. Si la empresa no cuenta con uno el valor aproximado se muestra en la tabla.

Los costos de los requerimientos para una red capaz de manejar 10 teléfonos celulares y con un alcance de al menos 500m, se muestran en la siguiente tabla:

Tabla II. Costos de una red *VoIP* sobre *Bluetooth*

Cantidad	**Marca/Modelo**	**Precio Unitario US($)**	**Sub-Total US($)**
5	Belkin *Bluetooth* Access Point F8T030	170.00	850.00
10	Dispositivos celulares con acceso *Bluetooth*	0	0
1	Phone Gateway	1,500.00	1,500.00
TOTAL			**US($) 2,350.00**

2. TECNOLOGÍA EXISTENTE

2.1. *Bluetooth*

Figura 2. Logo internacional para *Bluetooth*.

2.1.1. Definición

Bluetooth, es una especificación industrial para Redes Inalámbricas de Área Personal (WPANs) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2.4 GHz. Está estandarizado por el grupo *Bluetooth Special Interest Group,* o Grupo de Interes Especial; integrado por mas de 9000 compañías de telecomunicaciones y otras ramas relacionadas con las tecnologías de red, que dirigen el desarrollo de la tecnología inalámbrica *Bluetooth*.

Los principales objetivos que se pretenden conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles y fijos.
- Eliminar cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

Los dispositivos que con mayor frecuencia utilizan esta tecnología pertenecen a sectores de las telecomunicaciones y la informática personal, como *PDAs*, teléfonos móviles, computadoras portátiles, ordenadores personales, impresoras o cámaras digitales.

2.1.2. Clasificación

Bluetooth es un protocolo diseñado especialmente para dispositivos de bajo consumo, con baja cobertura y basados en transceptores de bajo costo. Este protocolo permite que los dispositivos que lo implementan puedan comunicarse entre ellos cuando se encuentran dentro de su alcance. Las comunicaciones se realizan por radiofrecuencia de forma que los dispositivos no tienen que estar alineados y pueden incluso estar en habitaciones separadas si la potencia de transmisión lo permite. Estos dispositivos se clasifican como "Clase 1", "Clase 2" o "Clase 3" en referencia a su potencia de trasmisión, siendo totalmente compatibles los dispositivos de una clase con los de las otras.

Tabla IV. Alcance de las distintas clases de *Bluetooth*.

CLASE	Potencia máxima permitida (mW)	Rango aproximado (m)
Clase 1	100	~100
Clase 2	2.5	~10
Clase 3	1	~1

Los dispositivos con *Bluetooth* también pueden clasificarse según su ancho de banda:

Tabla V. Clasificación de *Bluetooth* según su ancho de banda.

Versión	Ancho de Banda (Mbit / s)
Versión 1.2	1
Versión 2.0 + EDR	3
UWB *Bluetooth*	53 - 480

2.1.3. Funcionamiento

Cada dispositivo *Bluetooth* tiene un reloj físico, el cual cambia su frecuencia a una velocidad de 1600Hz en un modo pseudo-al azar. Con el fin de comunicarse uno con otro, dos dispositivos necesitan sincronizar sus relojes y su patrón de salto de frecuencia. Después de la sincronización, se forma un *piconet*. Un *piconet* siempre consta de un maestro, que establece el tiempo y patrón de salto de frecuencia del reloj, y arriba de 7 esclavos cuya función es aceptar estos ajustes. Un dispositivo puede ser un maestro solamente en un *piconet*, pero puede ser un esclavo en cualquier número de *piconets*. Esto permite la formación de *scatternets*, que pueden encaminar los datos a través de múltiples *piconets*. Sin embargo, no se provee el estándar para permitir tal enrutamiento.

Bluetooth se distingue entre distintos tipos de conexión. Las conexiones ACL son fiables, lo que significa que ellas reconocerán si fueron recibidas, retransmitirán si reconocen la recepción y se reportará un error si la retransmisión finalmente fracasa. Los enlaces ACL son comparables con el protocolo *TCP* de la pila *IP*.

Las transmisiones ASB y las transmisiones de servicio público para la radiotelevisión (PSB) son poco fiables. No hay manera de que el remitente sepa si su paquete llegó a su destino del todo. Estos tipos de paquetes permiten solamente al “maestro” enviar difusiones en un *piconet*.

Las conexiones ACL, ASB y PSB están enmarcadas, esto significa que es posible enviar datos, lo cual es más grande que un paquete real, ya que los datos serán distribuidos en varios paquetes. Estas conexiones también son asincrónicas, que básicamente significa que el ancho de banda no se les ha reservado.

Las conexiones SCO y eSCO son sincrónicas, no fiables y no sin enmarcar. Sobre el establecimiento de la conexión, las franjas horarias son reservadas para estos paquetes, y ningún otro dispositivo puede enviar datos durante la franja de tiempo reservado. Estos se utilizan principalmente para enviar datos de audio PCM a losa auriculares o sistemas de habla de manos libres.

2.1.4. Versiones

2.1.4.1. *Bluetooth* v.1.1

En 1994, Ericsson inició un estudio para investigar la viabilidad de una nueva interfaz de bajo costo y consumo para la interconexión vía radio (eliminando así cables) entre dispositivos como teléfonos móviles y otros accesorios. El estudio partía de un largo proyecto que investigaba unos multicomunicadores conectados a una red celular, hasta que se llegó a un enlace de radio de corto alcance, llamado *MC link*. Conforme este proyecto avanzaba se fue haciendo claro que éste tipo de enlace podía ser utilizado

ampliamente en un gran número de aplicaciones, ya que tenía como principal virtud que se basaba en un chip de radio

2.1.4.2. *Bluetooth* v.1.2

A diferencia de la 1.1, provee una solución inalámbrica complementaria para co-existir *Bluetooth* y Wi-Fi en el espectro de los 2.4 GHz, sin interferencia entre ellos. La versión 1.2 usa la técnica "*Adaptive Frequency Hopping (AFH)*", que ejecuta una transmisión más eficiente y un cifrado más seguro. Para mejorar las experiencias de los usuarios, la V1.2 ofrece una calidad de voz (*Voice Quality - Enhanced Voice Processing*) con menor ruido ambiental, y provee una más rápida configuración de la comunicación con los otros dispositivos *Bluetooth* dentro del rango del alcance, como pueden ser PDAs, *HIDs* (*Human Interface Devices*), computadoras portátiles, computadoras de escritorio, *headsets*, impresoras y celulares.

2.1.4.3. *Bluetooth* v.2.0:

Creada para ser una especificación separada, principalmente incorpora la técnica "*Enhanced Data Rate*" (*EDR*) que le permite mejorar las velocidades de transmisión en hasta 3Mbps a la vez que intenta solucionar algunos errores de la especificación 1.2.

2.1.4.4. *Bluetooth* v.2.1:

Simplifica los pasos para crear la conexión entre dispositivos, además el consumo de potencia es 5 veces menor.

2.1.4.5. *Bluetooth* v3.0 (mediados 2009):

Aumenta considerablemente la velocidad de transferencia. La idea es que el nuevo *Bluetooth* trabaje con *WiFi*, de tal manera que sea posible lograr mayor velocidad en los *smartphones*.

2.1.4.6. Futuro de *Bluetooth*

- *Ultra Wide Band Bluetooth*: El 28 de marzo de 2006, el *Bluetooth* SIG anunció su intención de utilizar *Ultra-Wideband*/MB-OFDM como capa física para futuras versiones de *Bluetooth*. La integración de UWB creará una versión de la tecnología *Bluetooth* con opción a grandes anchos de banda. Esta nueva versión permitirá alcanzar los requisitos de sincronización y transferencia de grandes cantidades de datos así como de contenidos de alta definición para dispositivos portátiles, proyectores multimedia, televisores y teléfonos *VOIP*.

- *Ultra Low Power Bluetooth*: El 12 de junio de 2007, Nokia y el *Bluetooth* SIG anunciaron que Wibree formará parte de la especificación de *Bluetooth* como versión de muy bajo consumo. Sus aplicaciones son principalmente dispositivos sensores o mandos a distancia. Puede resultar interesante para equipamiento médico. La propuesta de Nokia es utilizar esta tecnología como enlace de bajo coste hasta un teléfono móvil que actúe de puerta de enlace hacia otras tecnologías como UMTS, *Wi-Fi* o incluso el mismo *Bluetooth*.

2.1.5. Información técnica

La especificación de *Bluetooth* define un canal de comunicación de máximo 720 kb/s (1 Mbps de capacidad bruta) con rango óptimo de 10 m (opcionalmente 100 m con repetidores).

La frecuencia de radio con la que trabaja está en el rango de 2,4 a 2,48 GHz con amplio espectro y saltos de frecuencia con posibilidad de transmitir en Full Duplex con un máximo de 1600 saltos/s. Los saltos de frecuencia se dan entre un total de 79 frecuencias con intervalos de 1Mhz; esto permite dar seguridad y robustez.

La potencia de salida para transmitir a una distancia máxima de 10 metros es de 0 dBm (1 mW), mientras que la versión de largo alcance transmite entre 20 y 30 dBm (entre 100 mW y 1 W).

Para lograr alcanzar el objetivo de bajo consumo y bajo costo, se ideó una solución que se puede implementar en un solo chip utilizando circuitos CMOS. De esta manera, se logró crear una solución de 9×9 mm y que consume aproximadamente 97% menos energía que un teléfono celular común.

El protocolo de banda base (canales simples por línea) combina conmutación de circuitos y paquetes. Para asegurar que los paquetes no lleguen fuera de orden, los slots pueden ser reservados por paquetes síncronos, un salto diferente de señal es usado para cada paquete. Por otro lado, la conmutación de circuitos puede ser asíncrona o síncrona. Tres canales de datos síncronos (voz), o un canal de datos síncrono y uno asíncrono, pueden ser soportados en un solo canal. Cada canal de voz puede soportar una tasa de transferencia de 64 kb/s en cada sentido, la cual es suficientemente adecuada

para la transmisión de voz. Un canal asíncrono puede transmitir como mucho 721 kb/s en una dirección y 56 kb/s en la dirección opuesta, sin embargo, para una conexión asíncrona es posible soportar 432,6 kb/s en ambas direcciones si el enlace es simétrico.

2.1.6. Arquitectura hardware

El hardware que compone el dispositivo *Bluetooth* está compuesto por dos partes:

- Un dispositivo de radio: encargado de modular y transmitir la señal.

- Un controlador digital: compuesto por una *CPU*, por un procesador de señales digitales (*DSP - Digital Signal Processor*) llamado *Link Controller* (o controlador de Enlace) y de los interfaces con el dispositivo anfitrión. El LC o *Link Controller* está encargado de hacer el procesamiento de la banda base y del manejo de los protocolos ARQ y FEC de capa física. Además, se encarga de las funciones de transferencia (tanto asíncrona como síncrona), codificación de Audio y cifrado de datos.

  El CPU del dispositivo se encarga de atender las instrucciones relacionadas con *Bluetooth* del dispositivo anfitrión, para así simplificar su operación. Para ello, sobre el *CPU* corre un *software* denominado *Link Manager* que tiene la función de comunicarse con otros dispositivos por medio del protocolo LMP.

2.2. *VoIP*

2.2.1. Definición

Voz sobre Protocolo de Internet, también llamado Voz sobre *IP*, VozIP, *VoIP* (por sus siglas en inglés), es un grupo de recursos que hacen posible que la señal de voz viaje a través de Internet empleando un protocolo *IP (Internet Protocol*). Esto significa que se envía la señal de voz en forma digital en paquetes en lugar de enviarla (en forma digital o analógica) a través de circuitos utilizables sólo para telefonía como una compañía telefónica convencional o PSTN (*Public Switched Telephone Network*, Red Telefónica Pública Conmutada).

Los Protocolos que son usados para llevar las señales de voz sobre la red IP son comúnmente referidos como protocolos de Voz sobre *IP* o protocolos IP. Pueden ser vistos como implementaciones comerciales de la "Red experimental de Protocolo de Voz" (1973), inventada por ARPANET. El tráfico de Voz sobre *IP* puede circular por cualquier red *IP*, incluyendo aquellas conectadas a Internet, como por ejemplo redes de área local (LAN).

Es muy importante diferenciar entre Voz sobre *IP* (*VoIP*) y Telefonía sobre *IP*.

- *VoIP* es el conjunto de normas, dispositivos, protocolos, en definitiva *la tecnología* que permite la transmisión de la voz sobre el protocolo *IP*.
- Telefonía sobre *IP* es el conjunto de *nuevas funcionalidades* de la telefonía, es decir, en lo que se convierte la telefonía tradicional debido a los servicios que finalmente se pueden llegar a ofrecer gracias a poder portar la voz sobre el protocolo *IP* en redes de datos.

2.2.2. Ventajas

La principal ventaja de este tipo de servicios es que evita los cargos altos de telefonía (principalmente de larga distancia) que son usuales de las compañías de la Red Pública Telefónica Conmutada (PSTN). Algunos ahorros en el costo son debidos a utilizar una misma red para llevar voz y datos, especialmente cuando los usuarios tienen sin utilizar toda la capacidad de una red ya existente en la cual pueden usar para *VoIP* sin un costo adicional. Las llamadas de *VoIP* a *VoIP* entre cualquier proveedor son generalmente gratis, en contraste con las llamadas de *VoIP* a PSTN que generalmente cuestan al usuario de *VoIP*.

El desarrollo de codecs para *VoIP* ha permitido que la voz se codifique en paquetes de datos de cada vez menor tamaño. Esto deriva en que las comunicaciones de voz sobre *IP* requieran anchos de banda muy reducidos. Junto con el avance permanente de las conexiones ADSL en el mercado residencial, éste tipo de comunicaciones, están siendo muy populares para llamadas internacionales.

Hay dos tipos de servicio de PSTN a *VoIP*: "Discado Entrante Directo" (*Direct Inward Dialling: DID*) y "Números de acceso". DID conecta a quien hace la llamada directamente al usuario *VoIP* mientras que los Números de Acceso requieren que este introduzca el número de extensión del usuario de *VoIP*. Los Números de acceso son usualmente cobrados como una llamada local para quien hizo la llamada desde la PSTN y gratis para el usuario de *VoIP*.

2.2.3. Funcionalidad

VoIP puede facilitar tareas que serían más difíciles de realizar usando las redes telefónicas comunes:

- Las llamadas telefónicas locales pueden ser automáticamente enrutadas a un teléfono *VoIP*, sin importar dónde se esté conectado a la red. Uno podría llevar consigo un teléfono *VoIP* en un viaje, y en cualquier sitio conectado a Internet, se podría recibir llamadas.
- Números telefónicos gratuitos para usar con *VoIP* están disponibles en Estados Unidos de América, Reino Unido y otros países de organizaciones como Usuario *VoIP*.
- Los agentes de c*all center* usando teléfonos *VoIP* pueden trabajar en cualquier lugar con conexión a Internet lo suficientemente rápida.
- Algunos paquetes de *VoIP* incluyen los servicios extra por los que PSTN (Red Publica Telefónica Conmutada) normalmente cobra un cargo extra, o que no se encuentran disponibles en algunos países, como son las llamadas tripartitas, retorno de llamada, remarcación automática, o identificación de llamadas.

2.2.4. Características principales

En muchos países del mundo, *IP* ha generado múltiples discordias, entre lo territorial y lo legal sobre esta tecnología, está claro y debe quedar claro que la tecnología de *VoIP* no es un servicio como tal, sino una tecnología que usa el Protocolo de Internet (*IP*) a través de la cual se comprimen y descomprimen de manera altamente eficiente paquetes de datos o datagramas, para permitir la comunicación de dos o más clientes a través de una red como la red de *Internet*. Con esta tecnología pueden prestarse servicios de Telefonía o Videoconferencia, entre otros.

2.2.4.1. Estándar

El Estándar de *VoIP* fue definido en 1996 por la UIT (Unión Internacional de Telecomunicaciones) y proporciona a los diversos fabricantes una serie de normas, con el fin de que puedan evolucionar en conjunto. Por su estructura el estándar proporciona las siguientes ventajas:

- Permite controlar el tráfico de la red, por lo que se disminuyen las posibilidades de que se produzcan caídas importantes en el rendimiento. Las redes soportadas en *IP* presentan las siguientes ventajas adicionales:
    - Es independiente del tipo de red física que lo soporta. Permite la integración con las grandes redes de *IP* actuales.
    - Es independiente del *hardware* utilizado.
    - Permite ser implementado tanto en *software* como en *hardware*, con la particularidad de que el *hardware* supondría eliminar el impacto inicial para el usuario común.
    - Permite la integración de Vídeo y TPV

2.2.4.2. Arquitectura de Red

El Estándar define tres elementos fundamentales en su estructura:

- Terminales: Son los sustitutos de los actuales teléfonos. Se pueden implementar tanto en *software* como en hardware.
- *Gatekeepers*: Son el centro de toda la organización *VoIP*, y serían el sustituto para las actuales centrales. Normalmente implementadas en *software*, en caso de existir, todas las comunicaciones pasarían por él.

- *Gateways*: Se trata del enlace con la red telefónica tradicional, actuando de forma transparente para el usuario.

Con estos tres elementos, la estructura de la red *VoIP* podría ser la conexión de dos delegaciones de una misma empresa. La ventaja es inmediata: todas las comunicaciones entre las delegaciones son completamente gratuitas. Este mismo esquema se podría aplicar para proveedores, con el consiguiente ahorro que esto conlleva.

- Protocolos de *VoIP*: Es el lenguaje que utilizarán los distintos dispositivos *VoIP* para su conexión. Esta parte es importante ya que de ella dependerá la eficacia y la complejidad de la comunicación.
  Por orden de antigüedad (de más antiguo a más nuevo):
  - H.323 - Protocolo definido por la ITU-T
  - SIP - Protocolo definido por la IETF
  - Megaco (También conocido como H.248) y MGCP - Protocolos de control
  - *Skinny Client Control Protocol* - Protocolo propiedad de Cisco
  - *MiNet* - Protocolo propiedad de Mitel
  - *CorNet-IP* - Protocolo propiedad de Siemens
  - IAX - Protocolo original para la comunicación entre PBXs Asterisk (Es un estándar para los demás sistemas de comunicaciones de datos, actualmente esta en su versión 2 - IAX2)
  - *Skype* - Protocolo propietario *peer-to-peer* utilizado en la aplicación *Skype*
  - IAX2 - Protocolo para la comunicación entre PBXs Asterisk en reemplazo de IAX
  - MGCP- Protocolo propietario de Cisco
  - weSIP - Protocolo licencia gratuita de VozTelecom

Como hemos visto *VoIP* presenta una gran cantidad de ventajas, tanto para las empresas como para los usuarios comunes. La pregunta sería ¿por qué no se ha implantado aún esta tecnología? A continuación analizaremos los aparentes motivos, por los que *VoIP* aún no se ha impuesto a las telefonías convencionales.

2.2.5. Parámetros de la *VoIP*

Este es el principal problema que presenta hoy en día la penetración tanto de *VoIP* como de todas las aplicaciones de *IP*. Garantizar la calidad de servicio sobre *Internet*, que solo soporta "mejor esfuerzo" (*best effort*) y puede tener limitaciones de ancho de banda en la ruta, actualmente no es posible; por eso, se presentan diversos problemas en cuanto a garantizar la calidad del servicio.

2.2.6. Códecs

La voz ha de codificarse para poder ser transmitida por la red *IP*. Para ello se hace uso de Códecs que garanticen la codificación y compresión del audio o del video para su posterior decodificación y descompresión antes de poder generar un sonido o imagen utilizable. Según el Códec utilizado en la transmisión, se utilizará más o menos ancho de banda. La cantidad de ancho de banda suele ser directamente proporcional a la calidad de los datos transmitidos.

Entre los codecs utilizados en *VoIP* encontramos los G.711, G.723.1 y el G.729 (especificados por la ITU-T) Estos Codecs tienen este tamaño en su señalización:

- G.711: bit-rate de 56 ó 64 Kbps.
- G.722: bit-rate de 48, 56 ó 64 Kbps.
- G.723: bit-rate de 5,3 ó 6,4 Kbps.
- G.728: bit-rate de 16 Kbps.
- G.729: bit-rate de 8 ó 13 Kbps.

Esto no quiere decir que es el ancho de banda utilizado, por ejemplo el Codec G729 utiliza 31.5 Kbps de ancho de banda en su transmisión.

2.2.7. Retardo o latencia

Una vez establecidos los retardos de tránsito y el retardo de procesado la conversación se considera aceptable por debajo de los 150 ms.

2.2.8. Calidad del servicio

La calidad de este servicio se está logrando bajo los siguientes criterios:

- La supresión de silencios, otorga más eficiencia a la hora de realizar una transmisión de voz, ya que se aprovecha mejor el ancho de banda al transmitir menos información.
- Compresión de cabeceras aplicando los estándares RTP/RTCP.
- Priorización de los paquetes que requieran menor latencia. Las tendencias actuales son:
    - CQ (*Custom Queuing*) (Sánchez J.M:, *VoIP*'99): Asigna un porcentaje del ancho de banda disponible.
    - PQ (*Priority Queuing*) (Sánchez J.M:, *VoIP*'99): Establece prioridad en las colas.

- WFQ (*Weight Fair Queuing*) (Sánchez J.M:, *VoIP*'99): Se asigna la prioridad al tráfico de menos carga.
- DiffServ: Evita tablas de encaminados intermedios y establece decisiones de rutas por paquete.

- La implantación de IPv6 que proporciona mayor espacio de direccionamiento y la posibilidad de *tunneling*.

2.2.9. Futuro de *VoIP*

El ancho de banda creciente a nivel mundial, y la optimización de los equipos de capa 2 y 3 para garantizar el QoS (*Quality of Service*) de los servicios de voz en tiempo real hace que el futuro de la Voz sobre *IP* sea muy prometedor. En Estados Unidos los proveedores de voz sobre *IP* como Vonage consiguieron una importante cuota de mercado. En España, gracias a las tarifas planas de voz, los operadores convencionales consiguieron evitar el desembarco masivo de estos operadores. Sin embargo la expansión de esta tecnología está viniendo de mano de los desarrolladores de sistemas como Cisco y Avaya que integran en sus plataformas redes de datos y voz. Otros fabricantes como Alcatel-Lucent, Nortel Networks, Matra, Samsung y LG también desarrollan soluciones corporativas de voz sobre *IP* en sus equipos de telecomunicaciones.

2.3. SIP y RTP

El SIP, *Session Initiation Protocol,* o Sesión de iniciación de protocolo, es un control de protocolos para comunicación por audio y video (RFCSIP). Fue creado para proveer un "*superset*" de funciones de teléfono convencional. Puede utilizarse para hacer, re direccionar, contestar y terminar llamadas telefónicas, aunque el protocolo pude también ser usado para controlar otras

formas de comunicación como los *streams* de video o mensajes instantáneos. El SIP puede cargar la Sesión de descripción del protocolo, (*Session Description Protocol)* o SDP, para describir las propiedades de la sesión tal como: cuál numero de puerto y códec utilizar.

SIP introduce SIP URI del forma sip: nombre @ dominio, los cuales se utilizan para identificar a un compañero.

Por ejemplo, la dirección IP de Luis es 1.2.3.4, y su teléfono SIP se escucha en el puerto por defecto, 5060. Las llamadas al SIP: Luis@1.2.3.4 le enviarán un mensaje SIP INVITE. Si Luis quiere aceptar la llamada, envía de vuelta ACEPTAR, después una cadena de RTP se crea para llevar los datos de audio.

Sin embargo, uno no suele saber la dirección *IP* del interlocutor al que desea llamar. Esta es la razón por la que la parte del dominio de la URI por lo general apunta a un proxy de SIP. Cada usuario puede registrarse a si mismo a este proxy para publicar información tal como su dirección IP.

Por ejemplo, Luis trabaja en la Universidad de San Carlos, que tiene un proxy SIP. Su URI es Luis@sip.sancarlos.gt; cuando una persona quiere contactar a Luis, el proxy reenvía el mensaje *INVITE* a la dirección bajo la cual se había registrado previamente. A partir de aquí, todas las comunicaciones entre la persona que llama y Luis es directa, el proxy no se ingresa más. El *Real-Time Protocol* (Protocolo de tiempo real), o RTP, se utiliza para transferir los datos reales de audio. RTP es un protocolo basado en UDP, por lo que los paquetes pueden perderse o llegar fuera de orden. Probablemente la propiedad más importante de un paquete RTP es la marca de tiempo. Esto identifica cuando una determinada pieza de audio debe ser ejecutada. La marca

de tiempo depende de la frecuencia de muestreo, por ejemplo, 20 ms a 8000 Hz utilizan 160 pasos de tiempo.

Hay que tener en cuenta que SIP tiene problemas con la traducción de direcciones de red (NAT). Por ejemplo, uno podría registrarse a sí mismo o enviar un INVITE utilizando una dirección *IP* privada, que a su vez no se pueden enrutar correctamente por sus compañeros. Además, tanto RTP y SIP establecen conexiones punto a punto. Esta es sólo posible a través de NAT, si los puertos transmitidos están configurados correctamente, o si se utiliza un método como STUN.

Otra opción es dejar que el proxy de SIP trabaje como un agente de usuario *Back to Back* y dejar a ambos puntos conectarse al proxy en lugar de uno al otro.

2.4. Trabajo relacionado

Los problemas de este trabajo de graduacion ya han sido objeto de esfuerzos de investigación. Anteriormente, el problema de comunicación de bajo costo y corto alcance, ya ha sido objeto de varias investigaciones. La IETF tiene algunos grupos de trabajo en MANETs, que es otra palabra utilizada para redes ad-hoc. Sin embargo, la mayoría del trabajo se dirige hacia WLAN, y es difícil encontrar documentos que mencionen explícitamente la tecnología *Bluetooth*.

2.4.1. Connectblue

Igor Gurovski y Velimir Karadzic investigaron cómo crear redes *Bluetooth* ad-hoc con sistemas embebidos muy pequeños (*connectblue*). Sin

embargo, no hicieron una investigación públicamente disponible. Aún si la hicieran, sería poco probable que dicha aplicación pudiera ser utilizada directamente en un *Smartphone*

2.4.2. Lunar

Lunar es un protocolo de red ad-hoc proactiva, desarrollado por la Universidad Uppsala y la Universidad de Basel. Olaf Rensfield desarrolló dos variantes: una que se ejecuta en el núcleo de Linux y una que se ejecuta completamente en un espacio de usuario y utiliza un dispositivo *tun*. Lamentablemente, estas modificaciones *Bluetooth* no son parte de la distribución de Luna, ni están publicadas.

2.4.3. OLSR (*Optimized Link State Routing)*

OLSR es un protocolo de red ad-hoc proactivo, que ha sido creado en el RFC3626 experimental. Existen varias implementaciones, la más popular fue escrita originalmente por Andreas Tonnesen . Ha sido adoptado por freifunk.net para crear *WLAN* gratis meshes(redes de malla).

OLSR trabaja completamente en una capa *IP* y es por tanto fácil de transportar en las diferentes redes físicas y sistemas operativos. Sin embargo, tiene el inconveniente de que, a fin de descubrir un nodo, el protocolo subyacente debe tener ya establecida una conexión. Para conexiones *Bluetooth*, esto significa que OLSR no es adecuado para el descubrimiento de nodos, pero si es adecuado para la construcción de topologías de enrutamiento.

2.4.4. Voz sobre *Bluetooth* basado en MANETs

Stefan Ribhegge, trabajó casi el mismo tema que este trabajo de graduación. Él estableció la comunicación de Voz sobre *Bluetooth* utilizando dispositivos móviles, sin embargo, él se enfocó en la transferencia de voz, a través de redes *Bluetooth* puras. Este trabajo de graduación usa *Access Points* para alcanzar una compuerta central de teléfono. El transmite datos de voz sin comprimir sobre una conexión *Bluetooth* RFCOMM y diseña un protocolo reactivo de red ad-hoc que se adapta especialmente al *Bluetooth*. Debido a la falta de dispositivos físicos, el simuló extensivamente la escalabilidad de su red ad-hoc. Dado que no hay implementación de la presente red ad-hoc, la red se vuelve disponible y el método de enviar datos de audio, no es tan sofisticado como una RTP.

3. DISPOSITIVOS DISPONIBLES

Esta sección ofrece una breve descripción acerca de los sistemas operativos de los teléfonos *smartphones* de hoy día. Mientras que otros dispositivos pueden usar la red *Bluetooth* descrita en esta tesis también, estos no serán listados aquí. Esto se debe a que la integración de las llamadas telefónicas tradicionales se requiere en la aplicación, lo cual no puede hacerse en un PDA, por ejemplo.

Posteriormente, se da un breve repaso del dispositivo seleccionado para este trabajo de graduación.

3.1. Sistemas operativos

3.1.1. Dispositivos *Windows Mobile*

Existe un creciente número de teléfonos celulares que funcionan con *Windows Mobile*. Estos dispositivos normalmente ofrecen suficiente perfil tecnológico para *VoIP*, una interfaz gráfica de usuario muy fácil de usar, una biblioteca de sockets BSDstyle para conexiones de internet y entrada y salida de audio. Estos también permiten la colocación de llamadas GSM y acceso a la libreta de direcciones. Sin embargo, tal parece que el número de aplicaciones de código abierto es bastante limitado, y que la mayoría de *software* de terceros es *freeware*, *shareware* o *software* comercial de código cerrado.

3.1.2. Dispositivos Symbian

Symbian es probablemente el sistema operativo de *smartphone* más popular. Tiene distintas variantes que en su mayoría introducen nuevas clases de interfaz gráfica de usuario: Nokia maneja sus celulares con Series 60 y Series 80, mientras que Sony Ericsson, Siemens y Motorola usan UIQ. Recientemente, Nokia abrió un portal para recursos de *software* abierto. Ellos están constantemente realizando esfuerzos para crear una armazón (*framework*) *VoIP* de recurso abierto. Utilizando GnuBox es posible utilizar IP sobre conexiones *Bluetooth*. Mientras que Symbian API está bien documentada, no hay documentación disponible gratuitamente acerca de cómo escribir drivers. La API *Bluetooth* parece estar limitada a investigaciones y a la creación de conexiones RFCOMM.

3.1.3. Dispositivos Linux

Si bien los *smartphones* de Linux ya han sido liberados en Asia, ninguno de ellos parece haber sido disponible en Europa o en los E.E.U.U. Esto cambió cuando Motorola empezó a enviar sus Linux Smartphones en todo el mundo. Con el fin de portar una aplicación de comando de línea a un Linux Smartphone, uno solamente necesita compilarlo usando un compilador cruzado, si la aplicación ha sido programada con portabilidad en mente. Sin embargo, el ambiente Linux en los teléfonos puede variar fuertemente desde un ambiente usual de escritorio de Linux, por lo que requieren algún tipo de ajuste. Por ejemplo, la mayoría de aplicaciones gráficas para Linux, están escritas para el sistema X *Windows System*, mientras que la mayoría de teléfonos escriben gráficas directamente a un dispositivo *framebuffer*. Esto es, sin embargo, usualmente no un problema muy grande. Por ejemplo, todas las aplicaciones necesarias están disponibles como aplicaciones de comando de línea.

Existen, esfuerzos en curso para ejecutar *Smartphones* Linux o *Windows*. Como funciones avanzadas, tal como los dispositivos de túnel, o conexiones IP *Bluetooth* para Linux como implementaciones de código abierto, debería ser posible utilizarlas en el teléfono y modificarlas para adaptarlas a nuestras necesidades.

Por esta razón, elegí la serie Motorola A, ya que es la única serie de teléfonos Linux disponible al momento; y nos enfocaremos en el modelo A780.

3.2. El Motorola A780

Motorola introdujo el A780 a finales del 2004. Es ampliamente reconocido como el primer teléfono Linux disponible en Europa y América para finales del 2005, estaba disponible por alrededor de Q4,200.00, sin contratos con operador. Existen tres comunidades que tratan de sacar el máximo provecho del Linux instalado: Motorola fans, openezx y ccmove, formados para unir esfuerzos en el entendimiento de los teléfonos y crear aplicaciones para los mismos.

3.2.1. Hardware general

El Motorola A780 se compone principalmente de 2 procesadores: uno llamado Procesador de Aplicaciones que se ejecuta con Linux, y otro llamado *Baseband Procesor* o Procesador de Banda-Base, el cual se ejecuta con un Sistema Operativo propietario en tiempo real.

Si bien, el Procesador de Aplicación se basa en una pantalla táctil (*touchscreen),* recibe la entrada del teclado y el *touchscreen* , ejecuta sonidos, llamadas telefónicas y corre los programas visibles por el usuario, el Procesador

de Banda-Base comunica GSM y otras tecnologías de teléfonos móviles, ambos procesadores se comunican entre sí usando *USB*.

Figura 3. Teléfono Motorola A780

La pantalla táctil tiene una resolución de 320x240 pixeles y soporta millones de colores. El dispositivo /dev/ezx_ts dice la posición y presión del toque.

Una cámara y un chip GPS también están integrados en el teléfono. El Procesador de Aplicación tiene 48 megabytes de RAM y 48 megabytes de una memoria flash DiskOnChip. Esta almacena varios archivos de sistema.

Además, hay un mini lector de memoria TransFlash para más memoria persistente, que viene equipado con una tarjeta de memoria flash de 256 megabytes.

3.2.2. *Bluetooth*

El Motorola A780 viene con una pila de *Bluetooth* propietaria. Ho Ming Shun, alias CYPH, proporciona un paquete para sustituirla por la pila BlueZ Linux. Esta contiene el módulo de núcleo BlueZ habitual, biblioteca y herramientas de entorno de usuario en forma binaria, y un pequeño programa que inicializa el chip *Bluetooth* y establece la dirección de *hardware*.

3.2.3. Sonido

El núcleo utiliza un controlador de sonido de sistema sonido abierto, *OpenSound System* (OSS), para tener acceso al altavoz y micrófono. Existen tres dispositivos de sonido: */ dev / dsp, / dev/dsp16* y */ dev / audio*.

Los experimentos demostraron que el */ dev / dsp* solamente permite la salida estéreo, y el */ dev/dsp16* sólo permite salida mono. Ambos utilizan firmados de 16-bit (2'complementos) enteros. */ dev / dsp* se puede establecer en 8000 ó 44100 Hz, y */ dev/dsp16* sólo a 8000Hz. Es posible, para ambos, escribir de *raw* audio u ondas PCM a los dispositivos y leer de *raw* audio de ellos.

Por defecto, la salida de sonido está dirigida al mismo parlante que el de tono de llamada. Para las conversaciones telefónicas esto es molesto. Afortunadamente, es posible cambiar el dispositivo de salida de sonido hacia el auricular para llamadas telefónicas.

Con el fin de hacer esto, se abre */dev/mixer* y se ajusta *SOUND_MIXER_OUTSRC*. Los posibles valores se pueden encontrar en *output_enum* en el archivo de cabecera del Linux kernel *ezx -common.h*. Para

reproducir un tono de timbre, se debe establecer *LOUDERSPEAKER_OUT*, para conversaciones telefónicas a *EARPIECE_OUT*.

3.2.4. Hardware codec GSM

Como el objetivo es hacer llamadas telefónicas *VoIP*, utilizando un codec hardware de voz puede ser una gran manera de ahorrar uso de la CPU y de la vida de la batería. Según el OpenEZX wiki, el altavoz y el micrófono están adjuntos al llamado chip PCAP2. Este chip se puede comunicar con el procesador de aplicación y el procesador de banda base.

El procesador de banda base se encarga de la codificación GSM. La codificación de hardware GSM no es realmente necesaria, ya que el procesador de aplicación es lo suficientemente potente como para hacer en él la obtención del flujo de audio por medio de *software*.

3.2.5. *Software* Pre instalado

El *software* instalado en el teléfono consiste principalmente de una modificación de "*MontaVista Linux Consumer Electronics Edition 3.0* ".Este contiene el Linux Kernel versión 2.4.20, y una interfaz gráfica de usuario, Qt/Embedded 2.3, que se ejecuta en el dispositivo de framebuffer. Ambos, Linux y Qt/Embedded han sido modificados por MontaVista y Motorola para ajustarse a sus necesidades. Las modificaciones de Linux se han hecho públicas, mientras que Qt/Embedded existe bajo una licencia de propiedad, que no requiere publicar el código fuente. Motorola parece haber nombrado a su distribución "ezx", ya que esta cadena aparece en todo el teléfono. El teléfono se entrega con una serie de aplicaciones de usuario, que incluye un

cliente de correo electrónico, un navegador Web, un explorador de archivos de sistema, etc. Estos programas pueden iniciarse desde el menú de programas.

3.2.6. Flasheo

Antes de arrancar el Linux Kernel, el gestor de arranque establece ciertos bloques en la memoria flash DiskOnChip siendo estos protegidos contra escritura. Estos bloques no pueden ser modificados después. Existen dos herramientas para eludir esta limitación: los servicios de teléfono de Motorola, que no están disponibles al público, y los *flashkits de* CYPH. Utilizando un FlashKit de CYPH, es posible cambiar el contenido de los archivos de sistema de solo lectura, incluso sustituirlos por otros archivos de sistemas, y utilizar un Linux Kernel de auto-compilado.

3.2.7. Flex Bits

El A780, como la mayoría de los teléfonos móviles, puede configurarse por los llamados bits flex. Estos bits habilitan o deshabilitan funciones instaladas en el teléfono. Por ejemplo, un cliente de VPN se encuentra en /idioma/preloadapps/qtapps/motoVPN.

Este cliente comprende varios protocolos VPN propietarios. Iniciando la aplicación motoVPN se muestra la máscara de cliente, pero los valores son olvidados inmediatamente. La conexión a un servidor VPN no posible. Esto es probablemente porque el flexbit correcto no está habilitado. Habilitar ciertos flexbits también activa un cliente *VoIP* integrado, de acuerdo a Motorola fans. Sin embargo, para hacer esto, se necesita de las herramientas del Servicio telefónico de Motorola, las cuales no están legalmente disponibles al público.

3.2.8. Obtener una Shell

En la configuración del teléfono, uno puede seleccionar si el teléfono va al módem o entra en modo de almacenamiento masivo, cuando se conecta vía USB. Sin embargo, cuando la cadena "MotNet" se envía a /proc/usbdswitch, esta entra en modo de red y se asigna a sí misma la IP 192.168.1.2. La computadora conectada puede acceder a la cadena usando IP sobre USB.

Dos notables *daemons* se ejecutan en el teléfono: samba y telnet. Samba tiene 2 acciones (shares) pre-configuradas: "Home", la cual corresponde a /diska, y system que corresponde al directorio raíz. Telnet permite acceso a la raíz sin contraseña. Ambos *daemons* son controlados por inetd y sólo permite conexiones procedentes de las redes 192.168.1/24 y 169.254/16.

Una *Bash Shell* se encuentra instalada en el teléfono, sin embargo carece de algunas funciones tal como complementación de comandos e historia. El espacio de usuario instalado no tiene algunos programas importantes, como awk, gunzip, killall, netstat, o route. Esta es la razón por la que OpenEZX aconseja instalar *busybox*, y optimizar la biblioteca de C y el espacio de usuario para sistemas integrados.

Probablemente, la manera más fácil para configurar el teléfono en modo de red es usando Sam's Revitch "USBMode para A780", la cual tiene una interfaz gráfica de usuario.

3.2.9. Diseño de archivos de sistema

Después de ingresar con el perfil de root, se inicia un intérprete de comandos (bash Shell). El sistema se parece a un sistema Linux típico, tiene directorios como /etc, /dev/, /var, /usr, y así sucesivamente. *Mount* revela una serie de puntos de montaje:

rootfs on / type rootfs (rw)
/dev/root on / type cramfs (rw)
none on /ram type ramfs (rw)
proc on /ram/proc type proc (rw)
/dev/tffsa on /usr/language type cramfs (rw)
/dev/roflash2 on /usr/setup type cramfs (rw)
none on /dev/pts type devpts (rw)
/dev/mtdblock2 on /ezxlocal type vfm (rw,noatime)
/dev/mmca1 on /mmc/mmca1 type vfat (rw,noatime)
/dev/tffsb1 on /diska type vfat (rw,noatime)

La primera cosa que destaca es que el *mount* no es totalmente exacto, ya que la mayoría de estos archivos de sistema son de solo lectura. De hecho, /ram es de escritura, pero los archivos se borran después de un cierre de sesión. /ezxlocal es un archivo de sistema vfm, persistente y de lectura-escritura, de unos 5.2 megabytes, de los cuales cerca de la mitad es libre en un teléfono nuevo. /diska es un archivo de sistema FAT, de 40 megabytes, que parece que existe principalmente para datos de usuario. /mmc/mmca1 es la tarjeta TransFlash, que es, por defecto, también un archivo de sistema FAT. El hecho de que / ezxlocal sea un archivo de sistema vfm es muy importante, ya que VFM soporta archivos especiales de Unix, tal como los dispositivos de los nodos y enlaces simbólicos. Estos son necesarios para diversos fines.

3.2.10. Instalación de aplicaciones de terceros

Al conectar el teléfono a través de USB, cambia a "modo de almacenamiento masivo" y una parte de su disco (/diska) puede ser montado por el *USB*. El teléfono reconoce los archivos con terminación .mpkg como archivos instalables. Un archivo mpkg es un archivo tar comprimido con gzip que contiene una aplicación binaria, un símbolo del programa para el menú y un archivo .desktop que contiene referencias a los archivos y el nombre del programa. Al hacer clic sobre el archivo en el Explorador, se copia la aplicación a un destino oculto para el usuario (por ejemplo /diska/.system/QTDownload) y se agrega una entrada en el menú del programa.

Mientras que Motorola parece no querer que terceros escriban aplicaciones de sus teléfonos Linux, no pudieron impedir la creación de aplicaciones no autorizadas de terceros, tales como la EditorE, un editor de texto, y eKonsole, una consola de texto.

3.2.11. Compilación

El procesador de aplicaciones es una *CPU* ARM con algunas personalizaciones realizadas por Intel. El más notable es el "*Wireless* MMX", que consiste de una unidad de punto flotante paralelizado. Hay dos compiladores para este chip, ambos basados en "*Crosstools* para gcc": uno llamado "*Leprechauns*" (duende), (basado en gcc 3.3.2) y "*Crosstool* para EZX" (basado en gcc 3.3.6). Ambos compiladores parecen exhibir comportamientos extraños cuando se invoca la optimización de banderas.

Compilar una aplicación básica es muy fácil: En lugar de invocar el gcc, ld, etc uno simplemente prefija estos con el arm-linux-, para invocar la

compilación cruzada. Por ejemplo, 'arm-linux-gcc-helloworld, c-o hellorworld-arm 'compila un programa de línea de comando *hello world* para la plataforma arm.

Las cosas se ponen un poco más complicadas cuando la aplicación utiliza scripts de configuración, pero por lo general pasando "-host=rm-linux" a las aplicaciones de trabajo. Además, si la aplicación a-ser-compilada depende de las bibliotecas, la forma más fácil de instalarlas es todas en una carpeta, imitando un típico directorio layout /usr/. Por ejemplo, la configuración correcta para construir un *openvpn* (que depende de openssl) tiene este aspecto:

./configure --enable-pthread --disable-lzo \
--with-ssl-headers=/ezx/phone/armbuild/ezxlocal/usr/include \
--with-ssl-lib=/ezx/phone/armbuild/ezxlocal/usr/lib \
--prefix=/ezxlocal/usr/ --host=arm-linux

3.2.12. Escritura de aplicaciones GUI

Sam Revitch publicó un documento llamado "*Unofficial EZX Software Development Kit* ", que consiste de un Qt y archivos de cabecera EZX, parcialmente en ingeniería inversa de archivos de objeto C + +.

No contiene, sin embargo, ningún archivo de biblioteca, por razones de derechos de autor. Esos deben ser copiados desde el teléfono al equipo de desarrollo. También incluye una aplicación gráfica *Hello World*, instrucciones de compilación, y el Makefile puede crear archivos mpkg. Sin embargo, los programas creados con este kit de herramientas caen bajo la GPL, como los archivos utilizados con encabezado Qt son distribuidos bajo la licencia GPL. Alternativamente, uno podría comprar una licencia comercial de Trolltech (la compañía que desarrolla Qt/Embedded).

3.2.13. Lo que no es posible

Sin flashear el teléfono, lo cual es arriesgado, es imposible modificar los archivos de sistema de raíz (*root*). Esto significa que es imposible añadir nuevos usuarios, cambiar las contraseñas o iniciar los *daemons* en el arranque. Tampoco es posible acceder al núcleo de syslog o dmesg, cargando la *IPv6* o puenteando el módulo kernel o conseguir el *raw* socket del módulo kernel para funcionar correctamente.

4. DISEÑO DE RED

4.1. Requerimientos

Como se presentó en la sección de Requerimientos, la capa de red que proporciona los servicios *IP* debería ser de baja latencia, segura, autenticada, robusta, y auto-configurable. La baja latencia es importante, ya que el objetivo es hacer llamadas telefónicas, lo cual requiere tiempos de respuesta rápidos.

La red debe ser segura con el fin de prevenir las escuchas y la manipulación de datos. Solamente los usuarios autorizados deben tener acceso al servicio. Cuando se esté dentro el rango de un Access Point inalámbrico, los dispositivos deberían conectarse y auto-configurarse automáticamente. Más aún, la red debe ser fuerte contra interrupciones y automáticamente restablecer las conexiones o utilizar rutas redundantes para asegurar la entrega de paquetes. Óptimamente, todos estos objetivos deberían ser alcanzados utilizando componentes gratuitos, abiertos y estandarizados.

El dispositivo móvil tiene una interfaz *Bluetooth* para acceder a la red inalámbrica y tiene potencia de cómputo limitada. Esto significa que algunos requerimientos descritos arriba pueden no ser factibles, o pueden solamente ser factibles a expensas de otra característica o función. Un ejemplo sería la encriptación, la cual incrementa la latencia de conexión. Si la latencia es muy alta para la telefonía, la fuerza de encriptación necesitaría reducirse o incluso, deshabilitarse.

El enfoque utilizado en este trabajo de graduacion para crear una red es, primero, establecer conectividad básica y luego construir más funciones encima de eso.

Esta sección presenta paso por paso cómo crear conexiones *Bluetooth IP*, asignando direcciones *IP*, estableciendo la ruta, asegurando la conexión y la recuperación de las fallas de enlace. Esto luego resume lo que los *daemons* deben estar ejecutando en el *Phone Gateway*, el *Access Point* y el dispositivo móvil, y cómo deben estar configurados.

4.2. Conexiones Bluetooth IP

4.2.1. ¿Por qué *IP*?

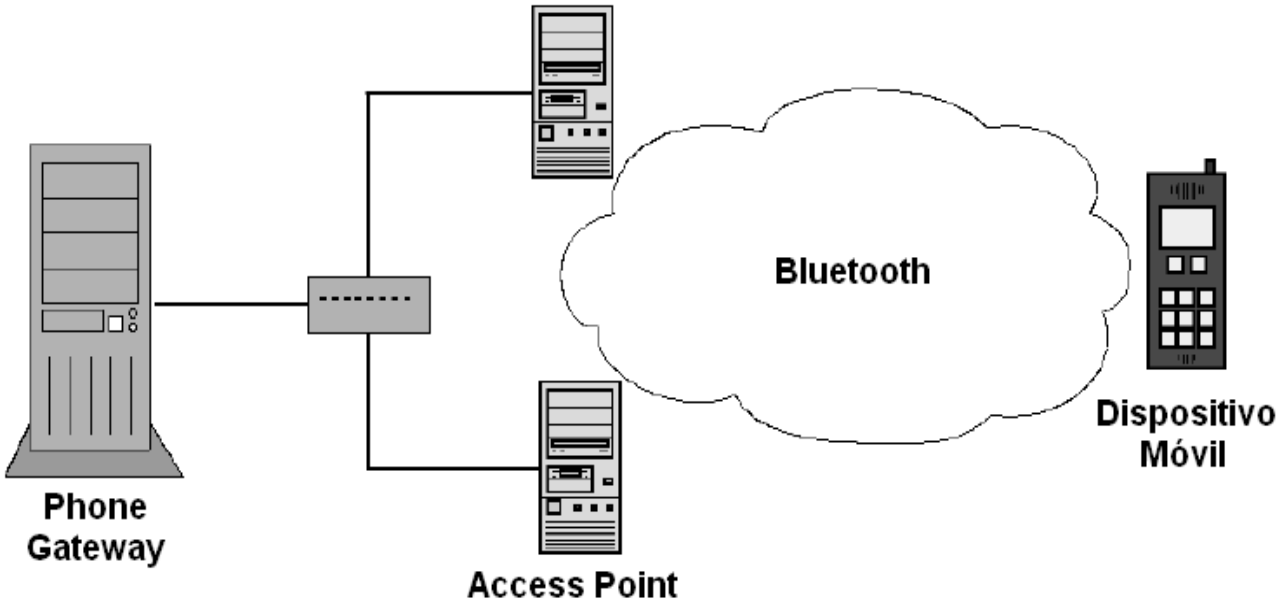
Teóricamente, debería ser posible enviar y recibir paquetes de datos de audio directamente sobre el *Bluetooth*.

Estos paquetes pueden ser luego traducidos en paquetes *IP* por los *Access Points*, quienes luego los envían al *Phone Gateway*. Esto podría ahorrar al menos 28 bytes por paquete: 20 por el encabezado IP y 8 por el encabezado UDB. Un *GSM-enconded RTP strem* genera alrededor de 52 paquetes por segundo, por lo que el total de ancho de banda ahorrado es de alrededor de 1.42 kilobyte por segundo. Afortunadamente, el ancho de banda no es tan escaso para que este tipo de economía sea necesaria. Mas aún, hay dos razones por las que esto no tiene sentido: la razón por la que el *IP* fue inventado en primer lugar, fue para interconectar diferentes redes físicas sin la necesidad de tal inflexibilidad artificial.

Además, modificar las aplicaciones existentes para no utilizar *IP* es muy difícil, sino es que imposible. Por ejemplo, los clientes *VoIP* normalmente

asumen que ellos tienen una *IP* y la dicen a sus compañeros; nosotros tal vez usemos *software* encriptado que utilice *IP-in-IP tunneling*, la cual se encuentra en los dispositivos kernel, para *tunnel IP into IP*, etc. El punto final es que el ahorro de no utilizar la *IP* en los dispositivos móviles no vale la pena el costo extra.

Figura 4. Diseño de la red, simplificado



4.2.2. Conectividad

El primer paso es establecer una conexión *Bluetooth*, que tiene capacidades IP. El dispositivo móvil debe escanear regularmente los dispositivos *Bluetooth* que ofrecen sus servicios como una compuerta *IP*.

Actualmente, hay dos maneras de hacer manejar IP sobre *Bluetooth*: una es creando una línea serial virtual, llamada RFCOMM y luego crear una conexión PPP sobre ella. La manera mas novedosa es la llamada PAN, y utiliza directamente una capa de *Bluetooth*, la cual resulta en gastos generales más bajos por paquete.

Las pilas del *Bluetooth* de Linux *BlueZ* se entregan con un *PAN daemon,* (*pand*), trabajando de lleno. *Windows* también soporta PAN. Con el fin de crear un PAN, uno necesita escuchar conexiones entrantes, y el otro extremo necesita iniciar la conexión.

La opción obvia para escuchar el *daemon* sería ejecutarlo en el Access Point, mientras el dispositivo móvil inicia la conexión. Para poder hacer esto, el dispositivo móvil necesita conocer la dirección *Bluetooth* del *Access Point*, la cual no puede asumirse que es conocida en adelante, porque mantener una lista de direcciones *Bluetooth* puede crear conflicto con nuestro mínimo requerimiento de configuración. Esto no es un problema, sin embargo, porque el dispositivo móvil puede simplemente escanear dispositivos *Bluetooth* que se encuentren en el rango de conexión. Sin embargo, esto puede tomar un tiempo relativamente largo (60 segundos), dependiendo del número de dispositivos *Bluetooth* presentes.

Cuando hay pocos dispositivos presentes, un escaneo típico toma aproximadamente 10 segundos. El *pand* puede desempeñar este escaneo. Puede también detectar fallas en la conexión *Bluetooth*, buscar otro *Access Point* y crear una nueva conexión.

El comportamiento por defecto es usar *Bluetooth* SDP, según sus siglas en inglés *Bluetooth Service Discovery Protocol*. Esto significa que, con el fin de establecer una conexión PAN, se siguen los siguientes pasos:

1. Rastreo de dispositivos *Bluetooth* disponibles.
2. Elegir un dispositivo arbitrario que no haya sido elegido aún, si no hay ningún dispositivo disponible, regresar al punto1.
3. Utilizar SDP para investigar si el dispositivo se escucha en un canal PAN, si no, regresar al punto 2.
4. Conectar el equipo: Si la conexión falla, regresar al punto 2.
5. Si la conexión se pierde, regresar al punto 1.

En la práctica, se pudo observar que la conexión a un *Access Point* a veces falla por errores SDP. Tal parece que había una entrada PAN faltante o corrompida en el *sdpd* (*BlueZ SDP daemon*), ejecutándose en el *Access Point*, Afortunadamente, el *pand* puede utilizarse sin SDP, eliminando el punto 3 descrito arriba, del proceso, lo cual también incrementa la velocidad por unos segundos al momento de establecer la conexión. Cuando la conexión se establece exitosamente, una interface BNEP es creada y puede configurarse como cualquier interface. Todo el proceso de rastrear los dispositivos y establecer la conexión toma alrededor de 20 segundos. Este tiempo puede incrementarse a medida de que más dispositivos *Bluetooth* estén presentes.

4.2.3. Ajuste de la conexión *Bluetooth*

Cada enlace *Bluetooth* tiene un tiempo de supervisión de enlace. Si ningún dato es exitosamente enviado o recibido durante ese tiempo de supervisión de enlace específico, el enlace se borrará. *BlueZ* establece el tiempo por defecto de 20 segundos. Tan pronto la detección de fallas de enlace

es requerida para llamadas telefónicas, tiene sentido disminuir este valor de conformidad a otros tiempos.

Uno puede sospechar que el *pand* usaría una conexión *Bluetooth* confiable, tal como ASB, para reflejar la poca fiabilidad de la *IP*. Sin embargo, el transporte ACL confiable es utilizado. Por defecto, los paquetes son reenviados por un tiempo indefinido, pero es posible ajustar los tiempos de retransmisión, en incrementos de 0.625ms. Como los paquetes son regulados comúnmente cerca de 60ms para compensar el retraso, este valor esta configurado a 50 ms (80 pasos de tiempo).

El paquete por defecto es DH5, el cual permite para datos de usuario de 300 a 320 kbit/s (el limite teórico es 433 kbit/s). Hcitool del paquete Bluez-utils, permite cambiar de tipo de paquete, pero esto parece no tener efecto sobre la conexión con el Motorola A780.

4.2.4. Conexiones múltiples

Dado que un piconet puede consistir de más de 8 dispositivos (1 máster y 7 esclavos), el Dispositivo Móvil, puede conectarse a más de un Access Point a la vez. Sin embargo, cada Pand solamente puede crear una conexión. Esto significaría que si queremos conectarnos a dos *Access Points* utilizando las herramientas existentes, necesitaríamos ejecutar dos *pands*. Esto representa una serie de problemas, primero que nada, el iniciar dos *pands* simultáneamente, lleva a dos indagaciones del desempeño al mismo tiempo. Si ambos tratan de conectarse al mismo *host* a la vez, pueden ocurrir condiciones *race.* Ya que el *Bluez* busca sus conexiones PAN por medio de las direcciones *Bluetooth*. Luego, mientras una *pand* crea su conexión y utiliza una

capa *IP* dada, la otra tratará constantemente de descubrir y conectarse a un *Access Point*.

Esto interfiere con el envío de datos *Bluetooth*, por ejemplo, la capa *IP*: *Bluetooth* divide su banda base en 79 canales físicos y cambia el canal que utiliza 1600 veces por segundo. Sin embargo, cuando se investiga, esta frecuencia, disminuye para hacer que los dispositivos sean más fácil de encontrar.

Esta frecuencia de salto de cambio de canal, interfiere con un correcto envío y recepción de datos de usuario. Este comportamiento fácilmente puede observarse cuando se envía una solicitud ICPM ping a un dispositivo *Bluetooth*. Cuando solamente hay un *pand* conectado, el tiempo de viaje de vuelta promedio *Artt* (*Average round trip time*) es 34.001 ms con una derivación estándar (*sder*) de 11.880 ms. Cuando se ejecuta un *pand* y las otras están indagando**,** estos valores incrementan drásticamente a un *Artt* de 184.168 ms con un *sder* de 292.543 ms. Cuando ambos *pand* finalmente se conectan, esto todavía tiene un impacto en la conexión, mientras el *Artt* y el *sder* incrementan a 82.302 ms y 27.824 ms, respectivamente.

Para resolver el problema de indagación, se modificó el *pand* para que soporte múltiples conexiones, usando tres nuevos argumentos de comando de línea: *maxconn, ingint* y *lockfile*. En vez de abrir solo una conexión y esperar a que esta se caiga, se trata de crear hasta un número <maxconn> de conexiones luego de la primera indagación. Esto, luego espera <inqint> segundos para que las conexiones se caigan, y, si <maxconn> conexiones no han sido alcanzadas, esto ejecuta una nueva indagación y trata de conectarse a más *Access Points*. Esto reduce la cantidad de tiempo gastado en la indagación, y por lo tanto, mejora la calidad de conexión. Más adelante, si se

usa con un argumento *lockfile*, esto solamente indaga si no hay ninguna conexión establecida al momento o si ésta puede adquirir una consulta de escritura de advertencia en el *lockfile*.

4.2.5. Problemas encontrados

Como se describió en la sección 4.2.2, el utilizar SDP causa algunos problemas, los cuales podrían solucionarse si no se utiliza SDP.

Si la pérdida de conexión es detectada solamente de un extremo, el otro extremo deja la conexión abierta y se rechaza conexiones futuras del mismo dispositivo, con el mensaje de error: "*bne0p: file exists*"¸hasta que el tiempo de conexión expira. Como el tiempo de expiración de la conexión es rápido, esto no debería ser un problema.

En el A780, no es posible auto-configurar la interfaz creada recientemente, ya que el archivo de raíz del sistema está protegido contra escritura. Sin embargo, el *pand* sabe cuando crea una interfaz y parte de esta tesis fue crear un *patch* que ejecute un script aleatorio luego de la creación. Un *patch* muy similar fue aceptado por Marcel Holtmann, del mantenimiento de *BlueZ*, y estará en el siguiente lanzamiento de *Bluez-utils*. Ahora, es trivial configurar la interfaz una vez que salga.

4.2.6. Implicaciones de seguridad

El cifrado y la autenticación no son problema de discusión; éstos serán manejados más adelante en este capítulo. De cualquier manera, es posible conducir un ataqUe de un *denial of service* (DoS), instalando *Access Points* falsos, lo cual provee un servicio PAN, pero no un acceso *IP*. El dispositivo puede detectar esto al medir la calidad del enlace a la compuerta del teléfono

en un *per-Access Point basis*. Sin embargo, en la implementación actual, el Dispositivo Móvil solamente cambia de *Access Point* cuando la conexión *Bluetooth* falla y no cuando falla en el enrutamiento correcto de *IP*.

4.3. Elección de la dirección IP

Como una o más interfaces con capacidades *IP* son suministradas por un *pand*, nosotros necesitamos asignar direcciones *IP*. Básicamente, hay tres maneras de hacer esto:

1. Asignar una dirección estática.
2. Obtener una dirección dinámica utilizando *DHCP*
3. Utilizar configuración cero, también llamado *zeroconf*.

Asignando una dirección *IP* estática contradice nuestro requerimiento de configuración mínima, ya que cada Dispositivo Móvil necesitaría que un usuario configure su dirección *IP*. Desafortunadamente, utilizar *DHCP* no es factible en un A780. Con el fin de utilizar DHCP es necesario el soporte de *raw socket*, el cual requiere un módulo kernel. En este trabajo fue factible compilarlo y cárgalo en el A780. Enviar paquetes DHCP también es posible. Sin embago, sobre la recepción de un paquete DHCP, el teléfono falló , lo cual hizo que la pantalla se tornara negra y que la batería fuera removida para reiniciar el teléfono. Este error no fue posible corregirlo.

La última opción es utilizar *zeroconf*. De acuerdo a la especificación, un dirección de la sub-red 169.254/16 se elegirá pseudo-al azar. Luego, se envían paquetes ARP, con el fin de detectar conflictos de dirección. Obviamente un espacio de implementación de usuario también necesita acceso a los *raw sockets*, para poder enviar paquetes ARP. Sin embargo, podemos (sin

protección) asumir que la probabilidad de un conflicto de direcciones es relativamente pequeña. Por supuesto, mientras más dispositivos estén conectados, el riego de que ocurran conflictos *IP* es mayor.

Se escribió un *zeroconfigurator* de *IP* para Linux, el cual genera una dirección IP basada en la dirección de interfaz de hardware y el reloj del sistema, y no utiliza ningún mecanismo para chequear confilictos de direcciones. En caso de utilizar *olsr (ver abajo)*, hay un *plugin olsr* que asigna direcciones automáticamente de manera similar a *zeroconf* y utiliza *olsr* para detectar y reparar conflictos de direcciones. Sin embargo, esto nunca ha sido lanzado debido a motivos de patentes de *software*. Es necesario tomar nota que existe una discusión en curso de cómo detectar y reaccionar ante conflictos de dirección en *olsr*, ya que esto se considera un problema severo.0

4.3.1. Problemas con el Motorola A780

En el Motorola A780, *telnetd* permite acceso a la raíz sin una clave, si el cliente *IP* viene de las sub-redes 192.168.1/24 ó 169.254/16. El archivo de sistema de la raíz está protegido contra escritura, y esto no puede cambiar sin hacer un flasheo del teléfono, de lo cual ya se habló. Por lo tanto, 169.254/16 no es una sub-red óptima para configurar la IP. Se utilizará la sub-red 10.1/16.

4.4. Enrutamiento de la *Phone Gateway*

El Dispositivo Móvil necesita ser capaz de establecer una conexión *IP* con el *Phone Gateway*, utilizando un *Access Point*. Como el AP ya tiene una ruta al PG, este simplemente enruta el paquete mientras hace una traducción de dirección de red, o NAT, por sus siglas en inglés (*Network Address Translation)*, para ocultar la dirección *IP* privada del Dispositivo Móvil.

4.4.1. Rutas estáticas

Mientras cada Dispositivo Móvil sea conectado solamente a un *Access Point,* el enrutamiento puede ser muy fácil. El Dispositivo Móvil sólo necesita conocer la dirección *IP* del Access Point, la cual utiliza como compuerta. El AP no necesita conocer la dirección *IP* del Dispositivo Móvil en adelante, simplemente enruta las respuestas utilizando NAT. Esto crea un sufijo, paradar a cada *Access Point* la misma dirección *IP*, por ejemplo, 10.1.0.1. Ningún Dispositivo móvil puede elegir esta dirección en adelante. Esto obviamente no funciona más adelante si el Dispositivo Móvil se conecta a más de un AP.

4.4.2. Usos y no usos para olsr

Otro enfoque es utilizar un algoritmo de red *ad-hoc*. El *olsr.org* OLSR daemon es una implementación popular para el protocolo OLSR (*Optimized Link State Routing)*. Este puede desplegarse fácilmente bajo los sistemas operativos Unix *oid* y *Windows*. Cuando el *daemon* esta ejecutándose, este emite los llamados mensajes *HELLO*, cada intervalo dado (el tiempo por defecto es 20 veces por segundo), en cada interfaz (especificada). Cuando este recibe un mensaje *HELLO* desde otro *host*, lo adhiere a su lista de vecinos directos. Esos vecinos luego intercambian topología de información, lo cual significa que cada *host* le dice a los otros, cuales rutas tiene para otros *hosts*. Las rutas son, usualmente, calculadas utilizando el sistema métrico de camino más corto, pero también hay una opción para usar el sistema métrico de calidad de enlace, el cual se basa en estadísticas de perdidas de paquetes. Mientras esto puede parecer óptimo para nuestros requerimientos, hay un gran ***drawback:*** Un número aleatorio de *routers* existe entre el *Access Point* y el *Phone Gateway*, y nosotros no podemos instalar y ejecutar un *olsr* en cada uno de ellos. Uno o

más *routers* (no ejecutando Olsr) entre dos *hosts* resultan en *hosts* no “visibles” a los otros, y por lo tanto, no encuentran una ruta entre el *Phone Gateway* y el *Access Point*, porque los *routers* no reenvían los paquetes emitidos. Incluso si lo hicieran, esto resultaría en que el *Phone Gateway* pensaría que tiene que enrutar a la dirección 10.1/16, esto sería descartado por el primer *router* de cualquier manera. Por lo tanto, *olsr* no puede reemplazar NAT en los *Access Points*.

Sin embargo, *olsr* puede ser útil para tratar de expandir el rango de la red *Bluetooth*. Si un *pand* también se escucha en un Dispositivo Móvil “A”, otro Dispositivo Móvil “B”, que no está dentro del rango del Access Point, puede conectarse con “A”, el cual tiene conexión al *Access Point*. *Olsr* calcularía la ruta de B al *Phone Gateway* a través de A.

Tambien, si el Dispositivo Móvil está conectado a múltiples *Access Points*, el esquema de ruta para las “rutas estáticas” ya no es funcional. *Olsr* es una buena solución para este problema. Utilizando paquetes *HELLO*, se recoge información sobre el entorno de la red. Los nodos pueden anunciar rutas fuera de la red *olsr* utilizando los mensajes de *Host and Netword Association (HNA).* Cada *Access Point*, por lo tanto, anuncia la ruta al *Phone Gateway*. Hay que notar que cada *Access Point* también necesita añadir al *Phone* Gateway su propia tabla de enrutamiento por la siguiente razón: si un dispositivo móvil se conecta a dos *Access Points*, ambos anunciarán una conexión al *Phone Gateway* con la máscara de red 255.255.255.255. El Dispositivo Móvil reenviará estos mensajes HNA a cada *Access Point*. Si el AP aún no tiene una entrada para el PG con la máscara de red 255.255.255.255, *olsr* la agrega, utilizando el otro *Access Point* como compuerta. Ahora, la primera ruta hacia el PG para cada AP, es a través del otro AP. Ambos *Access*

Points reenviarán mensajes designados al *Phone Gateway* a través del otro, sin haberse nunca encontrado.

4.5. Asegurar la conexión

Hasta ahora, todos los canales de comunicación son completamente inseguros. Todas las comunicaciones *Bluetooth* pueden ser escuchadas utilizando el equipo adecuado, ya que están totalmente sin encriptar. La capa IP (*IP* layer) está, también, completamente sin encriptar, lo que significa que un *Access Point* malicioso puede interceptar y manipular la comunicación a su voluntad. Además, como el *Access Point* acepta la conexión *Bluetooth* y dispone de un enrutamiento *IP* para todos, cualquiera puede utilizar esta red para acceder a Internet. En algunos casos, esto podría ser algo no deseado.

4.5.1. Encriptacion o cifrado *Bluetooth*

Una opción obvia sería utilizar el cifrado (encriptación**)** de *Bluetooth*. Prácticamente todos los dispositivos *Bluetooth* son compatibles con esta característica y, en general, se considera segura. Sin embargo, esto no se aplica a nuestro escenario. Con el fin de establecer un canal seguro con otro dispositivo *Bluetooth*, se necesita un pre-shared secret, llamado *PIN*. Una clave simétrica se genera de este *PIN*. En los *dispositivos cliente*, este PIN usualmente tiene 4 o 5 dígitos, pero puede ser una cadena UTF-8 de hasta 16 bytes.

Si una red entera utiliza este PIN para cifrar sus comunicaciones, nadie que sepa este PIN podría, desde un punto de vista teórico, descifrar todas las comunicaciones. Un escenario similar sería para cifrar una red usando un algoritmo AES y publicar la clave AES, que podría comprometer la seguridad.

Más importante aún, el cifrado termina en el *Access Point*, lo cual significa que el *Access Point*, o cualquier *host* que puede manipular la comunicación entre el dispositivo móvil y el *Phone Gateway*, puede poner en peligro los datos. Se puede concluir que el cifrado de *Bluetooth* no es adecuado para nuestro escenario. Sin embargo, si otros medios de asegurar la conexión no son viables, se podría utilizar el cifrado *Bluetooth* para hacer más difícil, aunque no imposible, manipular o “husmear” la comunicación "en el aire".

4.5.2. OpenVPN

Otra opción obvia para la seguridad de la red es crear una red privada virtual o *VPN* (*Virtual Private Network*). Existe una gran cantidad de proyectos de código abierto. Los más notables son FreeS/WAN y openswan; ambos permiten IPsec en el kernel de Linux. Sin embargo, estos requieren grandes modificaciones en el kernel de Linux. En contraste, OpenVPN corre en espacio de usuario y sólo necesita el módulo tun del kernel.

OpenVPN es un proyecto de código abierto que puede ser utilizado para crear conexiones TLS encriptadas punto a punto. Con el fin de establecer esa conexión, lo primero que deben hacer ambos, el cliente y el servidor, es intercambiar sus claves públicas. OpenVPN requiere countermeasures para desactivar cualquier intervención creada en medio de la conexión. La mejor manera de hacerlo es crear un Certificado de Autoridad (CA) la cual hace (registra) la clave pública de los servidores. La clave pública del CA se puede instalar en el dispositivo móvil, junto con la aplicación del teléfono, de modo que el dispositivo móvil pueda verificar la integridad de la clave pública del servidor. Utilizando este canal, lento pero seguro, ambos lados pueden intercambiar una clave simétrica y usarla para encriptar sus comunicaciones. Cada *host* en la

VPN, por lo tanto, se asigna a una dirección IP, l XUl puede usarse para una comunicación segura.

OpenVPN soporta *PAN,(Pluggable Authentication Modules*), esto significa que para conectarse a la VPN, los clientes necesitan suministrar a OpenVPN un nombre de usuario y contraseña, la cual podría verificarse usando un servidor LDAP existente. El servidor OpenVPN puede ejecutarse en el *Phone Gateway*, asegurando todas las comunicaciones entre el dispositivo móvil y el *Phone Gateway*.

Por otra parte, OpenVPN puede configurarse para ignorar de cual *IP* real proviene un paquete encriptado, lo que permite a los clientes cambiar sus *IPs* manteniendo su canal de *VPN*. Sin embargo, un cambio de dirección *IP* sólo se permite después de que expira el tiempo de conexión a la VPN actual. Las implicaciones de esto se discuten en la sección 4.6.

OpenVPN crea una interfaz de red, usualmente llamada tun0. Esta interfaz se auto-configura para una IP de un grupo de direcciones de la VPN del servidor OpenVPN.

4.5.3. Acerca del acceso a *Internet*

Si bien el objetivo principal de la red es permitir la comunicación *VoIP*, la comunicación *IP* normal se establece también. Si el dispositivo móvil advierte toda la *Internet* (0.0.0.0 / 0) en lugar de sólo la *Phone Gateway* a través de HNA, cualquier dispositivo *Bluetooth* conectado puede acceder a *Internet*. Si esto no es lo deseado, el *Access Point* debe bloquear todo el tráfico procedente de cualquier interfaz *Bluetooth* y que no va a la *Phone Gateway*.

Si sólo los usuarios autenticados pueden acceder a Internet, la solución lo más fácil es dejar que la *Phone Gateway* haga la traducción de las direcciones de la red. De esta forma, la conexión a Internet también se encripta entre la PG y los dispositivos móviles.

Por último, si no se desea ninguna conexión a internet, la *Phone Gateway* simplemente no traduce las direcciones de la red, ni tampoco enruta paquetes desde la *VPN* a los *hosts* fuera de la *VPN*.

Hay que tomar en cuenta que es imposible el "sólo permitir tráfico *DNS*", por ejemplo, con el fin de resolver el nombre de la *Phone Gateway.* El enfoque trivial es permitir únicamente el tráfico del puerto UDP 53, que es el puerto por defecto de la *DNS*. Sin embargo, cualquier tráfico UDP, tal como el *IP-over-UDP-tunnel,* de la OpenVPN*,* podría utilizar el *Access Point* para acceder a *Internet*. Incluso si el tráfico *DNS* se verifica a nivel de aplicación, es posible construir un túnel a través de *IP*.

4.6. Detección y recuperación de fallas del enlace

Mientras se establezca solamente una conexión *Bluetooth*, la detección de fallos en el enlace no es crítica.

Cuando el único enlace se cae, se tarda unos 30 segundos (en el mejor de los casos) para establecer una nueva conexión. Esto, por supuesto, interrumpe cualquier conversación telefónica en curso. Sin embargo, si se establecen múltiples conexiones, es fundamental detectar una caída de conexión, a fin de utilizar rutas alternativas. OLSR fue creado de manera explícita para este propósito. Este envía, regularmente, paquetes *HELLO* y si 3

paquetes no son reconocidos, decide que el vínculo está roto. Luego, calcula nuevas rutas, si es posible.

Ambos, Olsrd y OpenVPN modifican la tabla de enrutamiento. De forma predeterminada, esto lleva a una interrupción en la conexión si la ruta cambia. A continuación se da un ejemplo de una tabla de enrutamiento, modificada por un Olsrd, para un dispositivo móvil conectado a dos *Access Points*:

Tabla VI. Ejemplo de una tabla de enrutamiento modificada.

DESTINO	COMPUERTA	GENMASK	BANDERAS	LFACE
5.6.7.8	10.1.0.1	255.255.255.255	UGH	bnep1
5.6.7.8	10.1.0.2	255.255.255.255	UGH	bnep0
10.1.0.1	0.0.0.0	255.255.255.255	UH	bnep1
10.1.0.2	0.0.0.0	255.255.255.255	UH	bnep0
10.1.0.0	0.0.0.0	255.255.0.0	U	bnep0
10.1.0.0	0.0.0.0	255.255.0.0	U	bnep1
127.0.0.0	0.0.0.0	255.0.0.0	U	lo

La interfaz *Bluetooth* bnep0 y la bnep1, creadas por pand, están configuradas con direcciones de la subred 10.1.0.0/16. Olsrd detectó que estas interfaces están conectadas a los hosts 10.1.0.1 y 10.1.0.2, respectivamente. Además, ambos hosts advierten que pueden actuar como un *router* de la *Phone Gateway*, 5.6.7.8.

Ahora bien, cuando se conecta la Openvpn, se añadirán las rutas a la *Phone Gateway* y una ruta por defecto usando la conexión encriptada. Hay que tener en cuenta que OpenVPN asigna direcciones *IP* en pares (en este caso 10.8.0.5 y .6), donde la parte inferior de la dirección es la dirección de la puerta

de enlace. Como el servidor OpenVPN sólo responde a su dirección real de *VPN* (en este caso: 10.8.0.1), la primera entrada, simplemente asegura que se envía al destino correcto.

Tabla VII. Ejemplo de una tabla de enrutamiento con ruta añadida.

DESTINO	COMPUERTA	GENMASK	BANDERAS	LFACE
10.8.0.5	0.0.0.0	255.255.255.255	UH	tun0
5.6.7.8	10.1.0.1	255.255.255.255	UGH	bnep1
5.6.7.8	10.1.0.2	255.255.255.255	UGH	bnep0
10.1.0.1	0.0.0.0	255.255.255.255	UH	bnep1
10.1.0.2	0.0.0.0	255.255.255.255	UH	bnep0
10.8.0.1	10.8.0.5	255.255.255.255	UH	tun0
10.1.0.0	0.0.0.0	255.255.0.0	U	bnep0
10.1.0.0	0.0.0.0	255.255.0.0	U	bnep1
127.0.0.0	0.0.0.0	255.0.0.0	U	lo
0.0.0.0	10.8.0.5	128.0.0.0	UG	tun0

Hay que tener en cuenta que Olsrd, por lo general, sólo agrega una de las múltiples entradas HNA al mismo destino. De cualquier forma, si se destruye una conexión *Bluetooth*, todas sus rutas se destruyen automáticamente. Olsrd repara nuevas rutas HNA después de sólo unos segundos, por lo que por un corto tiempo, no habría ninguna ruta a la *Phone Gateway*. Utilizando un simple *parche*, Olsrd añade múltiples rutas HNA hacia el mismo destino, si todas comparten las mismas (más bajas) *hopcount*. Así que, si una conexión falla, todavía hay una entrada válida en la tabla de enrutamiento para el *Phone Gateway*.

Además, nótese que OpenVPN piensa que añadió la actual ruta activa de la *Phone Gateway* a la tabla de enrutamiento, a pesar del hecho de que ya existía. De este modo, al salir o reiniciar, se elimina esta entrada. Mientras tanto, Olsrd no vuelve a añadirla, ya que mantiene una lista interna de sus modificaciones a la tabla de enrutamiento, y por lo tanto piensa que la ruta todavía existe. Este comportamiento se puede corregir, sin embargo, al decirle a Openvpn que no modifique automáticamente, la tabla de enrutamiento (route-noexec), pero que ejecute los comandos especificados en la *route-up-options* .

Si una interfaz está destruida, pero existe otra conexión, OpenVPN simplemente utiliza la nueva ruta. Sin embargo, esto significa que desde el punto de vista de la *Phone Gateway*, el cliente OpenVPN cambia su dirección IP. Gracias a la opción de "flotar", el servidor OpenVPN aceptará paquetes desde cualquier *host* si la clave de enlace es válida. Esto permite restablecer rápidamente la conexión, cuando el cliente cambia de dirección IP. Sin embargo, en primer lugar se requiere que la conexión agote el tiempo de espera. Como OpenVPN utiliza paquetes *UDP*, se puede definir libremente el sentido del tiempo de espera. Dado que una recuperación rápida de fallas en el enlace es necesaria, OpenVPN se configuró para enviar un mensaje *keep-alive* una vez por segundo, y considerar el tiempo de espera, de 3 segundos sin mensajes.

Lamentablemente, cuando una nueva conexión *Bluetooth* aparece, Olsrd puede cambiar la tabla de enrutamiento para utilizar un nuevo Access Point como *router* a la *Phone Gateway*. Esto rompe la conexión VPN en la misma forma que si el enlace bajara. Después del período de tiempo de espera de los servidores (*ping timeout period*), se detectará una falla en el enlace. Esto parece absurdo, ya que la creación de una nueva conexión debe hacer que una red sea más estable, no interrumpirla. Sin embargo, es poco probable que una

nueva conexión *Bluetooth* se cree durante una llamada a telefónica. Usando la opción de bloqueo de archivos, no se harán intromisiones durante una llamada. Pero el teléfono podría recién haber terminado de indagar y tratar de conectarse a los nuevos dispositivos en el comienzo de una llamada telefónica. Esto puede conducir a una interrupción de la conexión justo al principio de una conversación telefónica.

Esta es también la razón por la que la calidad de enlace Oslrd métrica no debe ser utilizada. Esto causaría que el dispositivo móvil a cambiara de *Access Point rather ofter*. Cada cambio de este tipo necesitaría poner en marcha el tiempo de espera en el servidor OpenVPN antes de que el nuevo *Access Point* pueda ser reconocido.

4.7. Latencia

Dependiendo de la calidad de la conexión a Internet, una latencia de hasta 60 ms entre el *Access Point* y el *Phone Gateway* puede considerarse normal. La conexión *Bluetooth* PAN entre la AP y el dispositivo móvil añade, además, alrededor de 50-70 ms, y el túnel VPN añade 40-60 ms. En el peor de los casos, se trata de un retraso de la red prima de alrededor de 200 ms. Además, los datos de audio deben ser leídos desde la tarjeta de sonido, codificados, enviados, recibidos, amplificados para compensar la fluctuación, decodificados y escritos en el búfer de tarjeta de sonido. En total, esto se acumula en un poco menos de un segundo de retraso de audio, medido por un *ping* en el teléfono.

4.8. Resumen

El *Phone Gateway* ejecuta *ASTERISK*, o un servidor de telefonía similar, como *Back to Back User Agent*. También ejecuta OpenVPN en modo de servidor. La openVPN utiliza PAM para verificar las credenciales de los usuarios. Dependiendo de la elección del administrador, el *Phone Gateway* hace o no, la traducción de las direcciones de la red. Cada uno de los *Access Points* ejecuta un *pand* que "escucha", para permitir que los dispositivos móviles se conecten. Las direcciones IP de la subred 10.1.0.0/16 se asignan a las interfaces de bnep, creado por el *pand*. Los *Access Points* hacen la traducción de direcciones de red para que los dispositivos móviles puedan conectarse a la *Phone Gateway*.

Dependiendo de la elección del administrador, todas las conexiones a hosts que no son la *Phone Gateway*, están bloqueadas. Por otra parte, *olsrd* se ejecuta en todas las interfaces de bnep, y anuncia las rutas a la *Phone Gateway*.

En cada dispositivo móvil, pand se ejecuta y trata de conectarse a múltiples *Access Points*. Las direcciones *IP* de la subred 10.1.0.0/16 se asignan a las interfaces bnep creadas por el *pand*. *Olsrd* funciona en todas las interfaces de *bnep* y agrega rutas a la puerta *Phone Gateway*. Openvpn intenta crear una conexión segura a la *Phone Gateway*. En ambos lados, Olsrd envía paquetes cada 2 segundos. *Bluetooth* intenta reenviar los paquetes por 50 ms, y considera si un enlace se pierde si ningún paquete se envió o recibió con éxito durante 2.5 segundos. OpenVPN, en ambos lados, anvía mensajes *keepalive* una vez por segundo, si ningún otro tipo de tráfico pasa por el canal seguro. Ambos extremos de la OpenVPN considera que la conexión ha agotado el tiempo cuando ningún tráfico desde el otro lado se ha recibido durante 3

segundos. En este caso, el servidor OpenVPN acepta paquetes encriptados válidos desde cualquier dirección *IP* para restablecer rápidamente la conexión.

Si se rompe la conexión *Bluetooth* activa, y otra conexión *Bluetooth* se puede utilizar como ayuda de emergencia, el canal seguro se interrumpe durante unos 3 a 4 segundos.

5. LA APLICACIÓN DEL TELÉFONO

El usuario requiere una aplicación del teléfono unificada, para que pueda realizar ambos tipos de llamadas telefónicas: *GSM* y *VoIP*; tener acceso a la libreta de direcciones y hacer todas las configuraciones necesarias. Esta sección ofrece una visión general de cómo se ha implementado esta aplicación en el Motorola A780, o porque no podría implementarse.

Figura 5. Captura de pantalla de la aplicación del teléfono.

5.1. Inicialización del *Software* existente

Por lo general, las inicializaciones necesarias, tales como iniciar los *daemons* correctos y cargar de los módulos del *kernel* se hacen al momento del arranque. Sin embargo, esto no es posible en esta plataforma, por lo que tiene

que ser hecho por la aplicación. Para ello, es necesario tener privilegios de la *root*. Una forma de lograr esos privilegios, sería establecer el bit SUID, pero es más fácil llamar a los *start-stop-daemon* instalados en */sbin*, ya que esto no requiere un *chmod*.

En primer lugar, un script carga el módulo *BlueZ*, modificado por Ho Ming Shun, y se inician los *daemons* del *Bluetooth hcid* y *pand*. Esto, luego inserta el módulo Kernel *tun* para ser utilizado por la *openvpn*. También puede iniciar *olsrd* y establecer un nombre de servidor (*nameserver*).

La aplicación del teléfono directamente carga *openvpn*, y se comunica con él a través de una *Unix pipe*. Esto es necesario porque la *openvpn* pide un nombre de usuario y una clave, las cuales pasan a módulos PAM para su autenticación. Normalmente, la o*penvpn* no admite este comportamiento, y quiere que los datos sean escritos en un archivo temporal para ser leídos desde un *tty*. La o*penvpn* se modificó para leer desde *stdin*. Después de que una comunicación *openvpn* se establece, *linphone* se inicia y se registra a si misma en el *Phone Gateway*. Esto es necesario con el fin de recibir llamadas entrantes.

5.2. Configuración

Todas las opciones de configuración del usuario son accesibles desde el menú "*Configuration*". Estas opciones son luego serializadas y almacenadas en un archivo. Además, las opciones relevantes para la *VoIP* están escritas en el archivo de configuración del *linphone*. Después de escribir los nuevos archivos de configuración, *openvpn* se reinicia y el *linphone* finaliza hasta que se establece una nueva conexión *VPN*.

5.3. Calidad del enlace

Dos veces por segundo, el cliente envía un *ping* (solicitud ICM) a la *Phone Gateway*. Una vez por segundo el número de respuestas ICMP recibidas es leído y escrito en un búfer de anillo de longitud 8. Luego, la calidad del enlace se obtiene mediante el cálculo de la cuota de los paquetes recibidos y ampliándolo a 8. Luego se despliega en la barra de estado del teléfono, justo al lado de la fuerza de recepción GSM habitual.

Como alternativa para enviar dos paquetes *ping* por segundo al *Phone Gateway*, existen un número de métodos para medir la calidad del enlace *Bluetooth*. La opción obvia es *'hcitool lg'* que siempre regresa 255 en el Motorola A780. *'hcitoo rssi'* es otra opción que regresa mas valores significativos. *'ifconfig bnepX'* siempre muestra 0 errores. Ante una falla del enlace, éste se destruye sin mostrar un solo error. Finalmente, uno puede usar la extensión de calidad de enlace de *olsrd* y analizar el valor del enlace actual. Todas estas técnicas no demuestran que tan buena es la conexión con el *Phone Gateway*. No pude encontrar una forma de leer la fuerza de recepción GSM, sin embargo, esto se muestra en la esquina superior izquierda de la pantalla, por lo que el usuario la verá todo el tiempo.

5.4. Hacer y contestar llamadas telefónicas

Con el fin de realizar cualquier llamada de teléfono, el usuario introduce un número de teléfono. Luego, él o ella pueden hacer clic en *GSM* o *VoIP*, dependiendo de qué tipo de llamada de teléfono se desea. Alternativamente, el usuario puede configurar el modo a "Auto", para que el teléfono decida qué tipo de llamada debe hacerse. En la actualidad, las llamadas *VoIP* sólo se hacen cuando la calidad del enlace es mejor que 5/8. Idealmente, una tabla de precios

puede ingresarse en el teléfono, de manera que se utilice automáticamente el método más barato, pero esto no se aplicó. Al llamar a la aplicación *writelocks/ezxlocal/blue* se bloquea para que *pand* no indague. Las pruebas han demostrado que las investigaciones durante una llamada telefónica pueden corromper la llamada.

5.5. Llamadas *VoIP*

Existe un número de teléfonos *VoIP* de código abierto, los más comunes son *linphone*, *KPhone* y *minisip*. *Linphone* tiene una calidad de voz decente, un cliente de línea de comando e instrucciones de construcción detalladas para compilación cruzada, por lo que se eligió para la aplicación.

Si el usuario recibe una llamada telefónica *VoIP*, *linphone* lo señala usando un sonido de timbre y escribiendo una notificación, la cual es re-direccionada a la aplicación unificada del teléfono. Se modificó *linphone* para cambiar la salida de sonido al altavoz, antes de enviar el sonido de timbre. Al usuario se le pregunta si quiere aceptar o ignorar la llamada, y se le notifica a *linphone* en consecuencia. Si el usuario acepta la llamada, la salida de sonido se enciente en el auricular.

Luego de modificar *Linphone* para usar el dispositivo de sonido correcto, */dev/dsp16*, no parece funcionar muy bien. No había salida de sonido en todos los Dispositivos Móviles, y la salida de sonido en el otro extremo estaba tan perturbada que las palabras eran totalmente incomprensibles. Sonaba como si hubiera “agujeros” en los datos de voz. Estos problemas no se debieron a una potencia insuficiente de *CPU*. Si el *thread* considera que es muy lento para procesar los datos *VoIP*, se despliega el mensaje: “*Must cátchup n milisecons*”

Figura 6. Vista de la llamada entrante

5.5.1. Arreglando la salida de audio en un extremo

Cuando se inicia *linphone* con la opción *'-d 6'*, se muestra toda la salida de depuración. Había otro problema obvio; ya que este mensaje se repitió una y otra vez: *rtp.getg(): preguntar por paquetes demasiado viejos ! oldest=52765* sobre la recepción de paquetes de RTP, *linphone* almacena una lista, la cual se ordena por fecha y hora. También crea una lista de llamados *MSFilters* para cada corriente; un filtro para entrada de audio y otro para salida de audio.

Una configuración típica es aquella en que una cadena de filtros se lee desde la tarjeta de sonido, codifica la de datos de audio y los envía en un paquete RTP. La otra cadena de filtro consiste en un filtro de procesamiento de paquetes RTP entrantes, un filtro para decodificarlos y, finalmente, un filtro para

la salida de los datos de audio usando de la tarjeta de sonido. Un filtro que no tiene ningún tipo de filtros de entrada se denomina filtro de origen. Aquí hay un ejemplo de salida de una cadena de filtros:

Message:ms_filter_add_link: OssRead,0 -> GSMEncoder,0
Message:ms_filter_add_link: GSMEncoder,0 -> RTPSend,0
Message:ms_filter_add_link: RTPRecv,0 -> GSMDecoder,0
Message:ms_filter_add_link: GSMDecoder,0 -> OssWrite,0

Sobre cada iteración del *loop* principal, el *thread* primero procesa todos los filtros de origen y luego procesa la cadena de filtros. La función de procesos *RTORecy* recuerda la hora y fecha del último paquete procesado en la variable *MSRtpRecv.prev.ts.* Al principio de una cadena RTP, ésta es inicializada a 0. Luego utiliza *rtp_session_recvm_with_ts()* de la librería *oRTP* para obtener un paquete con su marca de hora, y cualquier paquete mas reciente de la lista de paquetes recibidos.

Como las marcas de tiempo de RTP no tienen que iniciar en 0 necesariamente, la librería *oRTP* recuerda la diferencia entre la marca de tiempo actual de la RPT y la representación para el *linphone*. También asegura que suficientes paquetes sean almacenados para compensar la fluctuación. Luego, llama *rtp_getq()* para finalizar la obtención correcta del paquete. O, en este caso, para ver que las marcas de tiempo requeridas ya fueron descartadas.

El primer intento para solucionar este problema fue simplemente ignorar el error y regresar el paquete más viejo. Esto funcionó, pero presentó una latencia bastante alta de entre 6 y 10 segundos para la conexión de voz.

Como esto no era muy deseable, decidí que el cálculo de la fecha y hora era de alguna manera, erróneo y que tal vez la manera más simple de solucionar el problema era ignorándolo. En *rtp_session_recvm_with_ts(),* utilicé la fecha y hora del primer paquete de la lista, y agregué un retraso por fluctuación, convertido a la escala de fecha y hora (*RTPSession.rtp.jittctl.jitt_comp_ts*). Esto, sin embargo, me trajo de vuelta al problema inicial, donde no se reproducía ningún sonido, porque los paquetes fueron descartados muy pronto. Agregando un valor arbitrario en lugar de *jitt.comp.ts*, de alguna manera se resolvió el problema. Esto lleva a la conclusión de que la compensación del retraso por fluctuación simplemente era muy grande. Esta se calcula en *jitter_control_set_payload()*, basado en la variable *JitterControl.jitt_comp* (en milisegundos) y las tasas de tiempo del códec de voz.

La variable *jitt.comp* es leída desde un archivo de configuración, el valor por defecto es 60. El máximo de este valor y del tamaño del bloque de la tarjeta de sonido, dividida por 8, es finalmente utilizado como el retraso por fluctuación (*jitter delay*). El tamaño de bloques de la tarjeta de sonido en un Motorola A780 es extraordinariamente alto, 8192, lo cual resulta en un tiempo de retraso de 1024 milisegundos. Al eliminar la comparación con el tamaño de bloque de la tarjeta de sonido, el retraso por fluctuación se ajusta a un valor por defecto de 60. Esto lleva a un claro bajo retarde de salida de audio.

5.5.2. Arreglando la salida de audio en el otro extremo

Como el problema anterior era un problema de marca de tiempo, era evidente investigar primero en esta dirección. La función de proceso *RTPSend* llama a *get_new_timestamp()* para generar una marca de tiempo. *Linphone* utiliza dos métodos para crear *timetamps.* Si los paquetes RTP son generados

periódicamente, el instante de muestreo nominal como se determina en el reloj de muestreo será utilizado para generar marcas de tiempo, no como una lectura del reloj del sistema. *Linphone* hace exactamente esto, incrementando la última marca de tiempo (*MSRtpSend.ts)* para cada paquete por el valor dependiente del códec: *MSRtpSpend.ts_inc.*

Sin embargo, esto también calcula una actualización de la marca de tiempo *clockts* desde el reloj del sistema. Se utiliza *clockts* si éste es mas nuevo que *ts+2*ts_inc*, de otra manera se usa *ts+ts_inc*.

En el Motorola A780, *clockts* cas nunca es mas nuevo que *ts+2*ts_inc.* Al eliminar el comportamiento de compilación RFC y simplemente, siempre calculando la marca de tiempo del reloj del sistema, la calidad del sonido se vuelve aceptable. Se comparó las incrementaciones reales con las que se hubieran hecho por *ts_inc.* La salida es básicamente una repetición de lo siguiente:

clock inc: 0, clockts/ts+inc diff: -160
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 0, clockts/ts+inc diff: -160
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 0, clockts/ts+inc diff: -160
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 0, clockts/ts+inc diff: -160
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 0, clockts/ts+inc diff: -160
clock inc: 80, clockts/ts+inc diff: -80

clock inc: 80, clockts/ts+inc diff: -80
clock inc: 0, clockts/ts+inc diff: -160
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 0, clockts/ts+inc diff: -160
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 0, clockts/ts+inc diff: -160
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 0, clockts/ts+inc diff: -160
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 0, clockts/ts+inc diff: -160
clock inc: 2880, clockts/ts+inc diff: 2720

El *ts_inc* del códec GSM es 160, asi que el comportamiento de compilación RFC sería incrementar la marca de tiempo por 160 en cada iteración. Sin embargo, esta lista presenta paquetes que son enviados de manera irregular. Mas seguido de lo que deberían, y luego no enviados para nada en un corto período de tiempo. Nótese que el total de incremento de la marca de tiempo de la lista, solamente difiere por 80 con el incremente de compilación RFC.

Una explicación sería que el reloj del sistema no es confiable y falla, y que los “agujeros” en la cadena de audio son causados por el repentino “gran· incremento que causa el *clockts*. Como el *clockts* regularmente es más grande que ts+2*tc_inc, este será el que se utilice. Esto conlleva a que las marcas de tiempo se incrementen un 50% más rápido de lo que deberían.

Cambiando la comparación: *clockts* > ts+2*tc_inc, a la de *clocks* > ts+4000, lleva a un incremento constante de marca de tiempo de 160. De cualquier modo, esto no mejora la calidad del sonido. Así que la explicación más probable es que el reloj del sistema es preciso y que algún tipo de problema de programación interrumpe la codificación regular del audio. Esto también explica porque el audio sonaba como si tuviera “agujeros”. La marca de tiempo debería haber incrementado por el valor por un valor mas pequeño que ese. Como resultado, el audio se ejecutó muy tarde, no teniendo nada en el medio.

Aún así, esta no es una solución perfecta. Cerca de cada tercio de iteración, la marca de tiempo no se incrementa nada, lo cual significa que dos paquetes llegaron a la misma marca de tiempo. En otras palabras: cerca de ¼ de paquetes son descartados.

Un enfoque simple es suavizar esos incrementos. Ya que sabemos que el incremento típico es 80, 0, 80, podemos cambiarlo a 80, 40, 40, chequeando si la marca de tiempo real incrementó. Si no lo hizo, esta será incremetnada por *ts_inc/4*. Esto entrega una buena calidad de sonido. Sin embargo, esta no es una solución muy limpia, ya que *ts_inc/4* simplemente se interpola desde la necesidad de obtener el valor 40, de alguna forma. Aún así, esta solución trabaja también con otros códecs, tal como ALAW yULAW. A continuación se presenta la salida de este método:

clock inc: 40, clockts/ts+inc diff: -120
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 40, clockts/ts+inc diff: -120

clock inc: 40, clockts/ts+inc diff: -120
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 80, clockts/ts+inc diff: -80
clock inc: 40, clockts/ts+inc diff: -120
clock inc: 2760, clockts/ts+inc diff: 2600

Todavía sería mucho mejor no tener dichos “grandes” retrasos, pero esto parece ser un problema de programación fuera del alcance de *linphone*.

Incluso con estos cambios, la calidad de sonido es mala en el otro extremo, cerca de 1 a 5 llamadas, mientras que es muy buena en el primer extremo. No se pudo establecer el porque. El terminar una llamada y volver a hacerla, generalmente, “arregla” el problema. El autor de *linphone* fue notificado acerca de ambos temas sobre el sonido y las soluciones.

5.6. Llamadas GSM

5.6.1. Análisis de la aplicación del teléfono

L a única manera “oficial” de hacer una llamada telefónica *GSM* es utilizando Java. Esto, por supuesto, no es muy práctico, ya que llamar a una aplicación Java desde C++ es lento y consume memoria. Sin embargo, la aplicación de Motorola parece tener una forma de hacer llamadas telefónicas, directamente, así que es interesante imitar ese comportamiento.

El candidato para una inspección mas detallada sería, obviamente, la aplicación llamada “*phone*”.

Al bloquear a través de los archivos de sistema, un puede fácilmente identificar */usr/lib/ezx/lib/libez.xphone-xcale-r.so*, como la probable librería responsable para manejar las llamadas telefónicas. Utilizando una “cadena” en la librería confirma esto, ya que contiene cadenas tales como *“MakeCall”* o *“VoiceCall_Java_MakeCall”*. Si uno pudiera ejecutar “*ltrace”* en la aplicación del teléfono para ver que llamadas a librerías son hechas, debería ser posible imitar este comportamiento en un programa C, utilizando las mismas llamadas a librerías y vinculándolas contra la librería. Sin embargo, esto no es posible fácilmente, ya que parece que Motorola implementó algunas contra-medidas para prohibir específicamente esto:

- Después de aproximadamente medio minuto de ejecutarse, cuando se esté rastreando “phone”, la función *ltrace* recibe un *SIGKILL*, el cual causa que esto termine.
- Mientras *ltrace* rastree “phone”, no es posible hacer llamadas telefónicas.

Afortunadamente, también existe la función “*strace”*, la cual registra todas las llamadas del sistema hechas por un programa. *Strace* recibe un no *SIGKILL*, y no es imposible hacer llamadas telefónicas mientras se esta ejecutando. Mientras está parado, la salida de “*strace-p ‘pidof phone’”* se ve de la siguiente manera:

```
select(19, [3 5 11 13 16 18], [], NULL, {0, 370000}) = 0 (Timeout)
times({tms_utime=2440, tms_stime=421, tms_cutime=566, tms_cstime=284}) = 684762
times({tms_utime=2440, tms_stime=421, tms_cutime=566, tms_cstime=284}) = 684762
times({tms_utime=2440, tms_stime=421, tms_cutime=566, tms_cstime=284}) = 684762
ioctl(3, FIONREAD, [0]) = 0
times({tms_utime=2440, tms_stime=421, tms_cutime=566, tms_cstime=284}) = 684762
select(19, [3 5 11 13 16 18], [], NULL, {0, 80000}) = 0 (Timeout)
times({tms_utime=2440, tms_stime=421, tms_cutime=566, tms_cstime=284}) = 684770
times({tms_utime=2440, tms_stime=421, tms_cutime=566, tms_cstime=284}) = 684770
ioctl(3, FIONREAD, [0]) = 0
```

Parece como si uno o más *QT Timers* estén ejecutándose dentro de la aplicación, mientras este espera una acción del usuario o una llamada entrante. Cuando se hace una llamada telefónica, este se activa. A continuación se presenta un extracto del *strace* cuando se esta llamando a 12345678901:

```
access("/tmp/emergency_call", F_OK) = 0
open("/tmp/emergency_call", O_RDONLY) = 21
fstat64(21, {st_mode=S_IFREG|0644, st_size=17592186044416, ...}) = 0
fstat64(21, {st_mode=S_IFREG|0644, st_size=17592186044416, ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x400d8000
read(21, "911\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 4096) = 132
```

fstat64(21, {st_mode=S_IFREG|0644, st_size=17592186044416, ...}) = 0
close(21) = 0
[...]
writev(12, [{"\1\2024\0\0\0\0\0", 8}, {"12345678901\0"
"\374\tA\n\0\0\0\20\376\32\0\370\2\25\0\310"..., 52}], 2) = 60
read(12, "\1\202\5\0\0\0\0\0", 8) = 8
read(12, "\0\0\0\0\26", 5) = 5
[...]
access("/ezxlocal/download/appwrite/phone", F_OK) = 0
lstat64("/ezxlocal/download/appwrite/phone",
{st_mode=S_IFDIR|0755, st_size=17592186044416, ...}) = 0
open("/ezxlocal/download/appwrite//phone/rc_d", O_RDWR|O_CREAT, 0666) = 22
fstat64(22, {st_mode=S_IFREG|0644, st_size=17592186044416, ...}) = 0
lseek(22, 296, SEEK_SET) = 296
write(22, "12345678901\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 72) = 72
close(22) = 0

Al parecer, sobre otras cosas, la aplicación del teléfono primero abre */tmp/emergency_call* para ver si la llamada saliente va hacia un número de emergencia. Esto es importante, ya que debería ser posible llamar a números de emergencia incluso sin una tarjeta SIM o sin ingresar el número de PIN. Luego de eso, la aplicación escribe el número de teléfono, entre otras cosas, al descriptor de archivo 12. Mas tarde, escribe el número de teléfono al */ezxlocal/download/appwrite/phone/rc_d*. Estos son los únicos dos lugares donde el número de teléfono aparece en la salida de *strace (strace output)*.

Al ver a */ezxlocal/download/appwrite/phone/rc_d* se pueden ver más números de teléfono, separados por caracteres especiales. Estos parecen ser los últimos números a los que se ha llamado. Esto, probablemente, no tiene nada que ver directamente con colocar una llamada telefónica. La opción obvia

es entonces, investigar luego en el descriptor de archivo 12. Cuando se envía *SIGTERM* al teléfono, se finaliza la aplicación, pero se reinicia inmediatamente. *Strace'ing* la recien iniciada aplicación de teléfono, despliega lo siguiente:

socket(PF_FILE, SOCK_STREAM, 0) = 12
connect(12, {sa_family=AF_FILE, path="/tmp/tapisock"}, 15) = 0
fcntl64(12, F_GETFL) = 0x2 (flags O_RDWR) fcntl64(12, F_SETFL, O_RDWR) = 0
getpid() = 676
writev(12, [{"\2\201 \0\0\0\0", 8}, {"\244\2\0\0\n\0\0\0\0\0\0", 12},
{"\0\2\0\4\0\6\0\7\0\10\0\t\0\n\0\ r\0\16\0\21", 20}], 3) = 40
read(12, "\2\201\4\0\0\0\0\0", 8) = 8
read(12, "\0\0\0\0", 4) = 4

Esto, básicamente, significa que el descriptor de archivo 12 es un archivo *socket* a */tmp/tapisock*. En la inicialización, algunos bytes que probablemente tienen significado especial son escritos a este. Cuando se hace una llamada telefónica, se escriben algunos otros bytes, seguidos por el número de teléfono como una cadena C (terminada por el byte 0), seguidos de otros bytes más. Cuando *strace'ing* la aplicación de teléfono múltiples veces, descubrí que los bytes de inicio y los bytes antes de el número de teléfono son idénticos, mientras que los bytes luego del número de teléfono, varían.

5.6.2. Hacer llamadas GSM

Como no fue posible encontrar el significado de los bytes que están luego del número de teléfono, decidí que tal vez no eran importantes y se podían reemplazar por bytes nulos para rellenar el mensaje a 60 bytes (sin contar el mensaje de inicio). Esto funcionó. El siguiente código C puede ser usado para hacer llamadas telefónicas a un número específico en *argv[1]* el cual no excede 52 decimales:

```
char *tapiname = "/tmp/tapisock";
struct sockaddr *addr; char buf[100] = { 0 };
ssize_t socksize = sizeof(addr->sa_family) + strlen(tapiname) + 1;
int fd = socket(PF_FILE, SOCK_STREAM, 0);
addr = (struct sockaddr*) malloc(socksize);
addr->sa_family = AF_FILE;
strcpy(addr->sa_data, tapiname);
memset(buf, 0, 100);
memcpy(buf, "\2\201\0\0\0\0\244\2\0\0\n\0\0\0\0\0\0\0\2\0\4\0\6"
"\0\7\0\10\0\t\0\n\0\r\0\16\0\21\1\2024\0\0\0\0", 48);
memcpy(buf+48, argv[1], strlen(argv[1]));
connect(fd, addr, socksize-1);
write(fd, buf, 100);
```

Estos 100 bytes hace una llamada telefónica al número especificado en *argv[1]*. Pero no solamente eso, también imitan completamente el comportamiento de hacer una llamada telefónica desde la libreta de direcciones o del teclado de marcación. La misma ventana de marcado aparece y también se encarga del audio, finalizar la llamada, etc. Esto es muy conveniente, ya que hubiera sido más difícil encargarse de la decodificación del audio entrante y la codificación del audio saliente.

5.6.3. Problemas encontrados

Hubo una pequeña sorpresa cuando se integró este código en la aplicación de teléfono unificada. Luego de hacer una llamada, la aplicación se finalizó. Esto puede ser fácilmente evitado utilizando *fork()* y *exec()* para ejecutar aplicaciones distintas, las cuales luego colocan la llamada telefónica.

5.7. Acceso a la libreta de direcciones

Con el fin de seleccionar cómodamente un número de teléfono para llamar, es necesario integrar la libreta de direcciones del teléfono. Afortunadamente, un miembro de *Motorolafans*, llamado *matiu,* ideó una estructura de la base de datos de la librera de direcciones, apoyándose en */ezxlocal/sysDatabase/native.db*. Es una base de datos *Berkeley* con varias sub-bases de datos. Una de ellas, llamada "*780_contact_table*" o "*contact_table*", dependiendo el modelo del teléfono. Cada fila en la sub-base de datos representa una única entrada de contacto, la cual básicamente, es un arreglo de cadenas C con longitud ajustada. *Matiu* también creó *usync*, que es una aplicación para exportar la base de datos de la libreta de direcciones a varios formatos, tal como un texto plano, con valores separados por comas, o un archivo de texto *vCard*. *Usync* está destinado para correr en una estación de regular de trabajo de Linux, no en el teléfono. Se requiere una librería de base de datos *Berkeley* versión 4.2. Como *usync* tiene un diseño muy modular, fue fácil aislar la parte que lee el dato de un contacto de la base de datos. Se planeó integrar esto a la aplicación del teléfono.

Sin embargo, no fue posible leer ningún dato de la base de datos mientras se ejecutaba en el Motorola A780. La librería de base de datos *Berkeley* reportó que abrió la base de datos pero cuando se recupera un registro, reportó "*DB_NOTFOUND: no matching key/data pair found"*. Traté de rastrear las llamadas del sistema hechas por la librería, pero no se encontró ninguna evidencia. Luego de abrir la base de datos, en lo cual si tuve éxito, no había ningún dato escrito o leído desde el archivo de la base de datos, incluso ni cuando fallaba el acceso al primer registro.

La base de datos *Berkeley* viene con aplicaciones de ejemplo. Una de ellas, *ex_access* lee el número de cadenas de el usuario y lo escribe en la base de datos. Luego despliega todas las entradas de la base de datos. Este programa si corrió exitosamente en mi estación de trabajo Linux, pero no en el teléfono. No pudo, incluso, leer de una base de datos existente, como la creada desde *ex_access*. La librería de la base de datos *Berkeley* compilada por el teléfono es obviamente inefectiva. No pude encontrar la razón para este comportamiento.

Otra opción para seleccionar un número desde la libreta de direcciones para la aplicación del teléfono sería usar un "*copy & paste*". Mientras se ejecuta la aplicación del teléfono, uno puede abrir la libreta de direcciones, seleccionar una entrada y copiar el número deseado al *clipboard*. Sin embargo, no es posible pegarlo en la aplicación del teléfono. Tal parece que los reproductores *QT/Embedded* no soportar esto, solamente los reproductores Motorola menos conocidos permiten el acceso al portapapeles. Hay dos aplicaciones disponibles que soportan el portapapeles: *EditorE* y *eKonsole*. Pero, no fue posible encontrar el código fuente para ningúno de los dos. Solamente puedo asumir que son códigos cerrados porque utilizan *Motorola's SDK* para teléfonos Linux, los cuales no son disponibles al público.

5.8. Consumo de potencia

Enviar y recibir paquetes *Bluetooth* consume energía. Si no hay *Access Points* en el rango, el dispositivo constantemente hace indagaciones *Bluetooth*. Si se conecta a un *Access Point*, este constantemente envía *pings*. Una manera mas eficiente, en términos de consumo de energía, sería enviar mensajes *keep-alive* con menos frecuencia, cuando la aplicación no esté en

primer plano ó cuando la tapa del teléfono esté cerrada. Desafortunadamente, no existe actualmente, una manera de detectar esto.

5.9. Instalación

Motorola intentó que las aplicaciones de usuario se instalen utilizando un archivo *.mpkg*. Este sería extraído a un directorio, desde el cual podría ejecutarse. Esto, sin embargo, no tiene mucho sentido en nuestro escenario, ya que un gran número de *daemons* y módulos *kernel* necesitan instalarse (*busybox, BlueZ, tun, openvpn, olsr* y la aplicación del teléfono. Mientras *openezx*, o un proyecto similar, no establezca una manera estandarizada de instalar paquetes en un ambiente similar a los distribuidos por Linux, lo mejor es dar instrucciones detalladas en un archivo tipo *readme*.

CONCLUSIONES

1. Se logró montar la red *Bluetooth* únicamente con *Access Points* conectados entre si y direccionados a un *Phone Gateway*, a su vez se configuraron los dispositivos para que se conecten a esta red y realicen llamadas vía *IP*.

2. Para lograr utilizar *VoIP* en los dispositivos del usuario final, es necesario que éste cuente con un sistema operativo libre y si este no fuera el caso habría que instalar uno de estas características.

3. La tecnología *Bluetooth* brinda la ventaja por su precio, que sea incluida en dispositivos celulares o *PDA* de bajo costo.

4. Uno de las configuraciones que más trabajo y tiempo requiere es la instalación de la aplicación en el dispositivo telefónico.

5. Instalaciones de parches fueron realizadas, debido a que el sistema inicial poseía vulnerabilidades.

6. Las funcionalidad de las llamadas vía *GSM* no fueron afectadas al momento de implementar nuestro diseño.

RECOMENDACIONES

Se deben de tomar en cuenta recomendaciones de diseño e implementación, para obtener un resultado óptimo al momento de poner en producción nuestro sistema.

1. Validar la compatibilidad de los dispositivos principales, siendo estos los *Access points* y el *phone Gateway;* esto para obtener una fuente solidad de señal para nuestra red.

2. Realizar una configuración correcta de los *Access points*, así como elegir una ubicación física en la cual se aproveche al máximo la irradiación de señal de estos dispositivos.

3. Al momento de realizar cambios en el sistema central de los dispositivos telefónicos, validar tanto las funciones principales como las secundarias para que éstas no sufran cambios que afecten su funcionamiento normal.

4. Antes de implementar el sistema, validar actualizaciones existentes, tanto de dispositivos como de sistemas operativos; esto para ver las ventajas que se pueden tener con estas actualizaciones, o tomar la decisión de seguir con el equipo y sistema actual, luego de realizar las pruebas exhaustivas.

BIBLIOGRAFÍA

1. ***Bluetooth* Core Specification, Version 1.2.** http://*Bluetooth*.org/foundry/adopters/document/*Bluetooth*_Core_Specification_v1.2, Julio , 2009.

2. **BlueZ - Official Linux *Bluetooth* protocol stack**. http://bluez.org. Septiembre, 2009.

3. Igor Gurovski & Velimir Karadzic. **Self-configuring *Bluetooth* Networks.** http://www.connectblue.se/fileadmin/Connectblue/PDF/White_papers/Selfconfiguring_*Bluetooth*_Networks.pdf. Julio, 2009 .

4. Keagy, Scott. **Integración de Redes de Voz y Datos.** Cisco Systems Inc. España, Madrid, 2001. Pág. 400-430.

5. **Implementing and Extending the Optimized Link State Routing Protocol** http://www.olsr.org/index.cgi?action=doc, 2009.

6. **Motorolafans**. http://www.motorolafans.com, Agosto, 2009.