
## Universidad Tecnológica de la Mixteca

# IMPLEMENTACIÓN DE UN SISTEMA DE CAPTURA DE PAQUETES EN REDES INALÁMBRICAS 802.11 Y BLUETOOTH

TESIS PARA OBTENER EL TÍTULO DE:
**INGENIERO EN COMPUTACIÓN**

PRESENTA:

**EMMANUEL MENDOZA ACEVEDO**

DIRECTOR DE TESIS:
**M.C. GABRIEL GERÓNIMO CASTILLO**

Huajuapan de León, Oax. Abril del 2005

# Contenido

# Prólogo

Las redes inalámbricas están siendo altamente implementadas en los ámbitos laborales, principalmente en el empresarial y en el educativo, para brindar movilidad a los usuarios, entre otras cualidades. El uso de las ondas electromagnéticas como medio de transmisión ha provocado que las redes inalámbricas presenten diferentes vulnerabilidades ante los ataques de los agresores, y que por lo tanto, la seguridad englobada en éstas sea un factor determinante para su implementación. El *sniffing* es el ataque mayormente implementado por los agresores debido a la gran cantidad de información que se puede recolectar y con la cual se pueden llevar a cabo otros ataques más ofensivos.

La implementación de los ataques *sniffing* se desarrolla con el apoyo de las aplicaciones denominadas *sniffers*. Estas aplicaciones pueden obtenerse en forma comercial o en forma gratuita. Los *sniffers* gratuitos están diseñados principalmente para trabajar sólo en redes 802.11 sobre el sistema operativo Linux y son frecuentemente utilizados por los agresores, en cambio, los *sniffers* comerciales son ampliamente usados por los administradores de red y por las personas que requieren de la elaboración de pruebas a sus productos inalámbricos. Estas aplicaciones llegan a alcanzar precios muy elevados y son diseñados exclusivamente para ejecutarse sobre Windows. Los *sniffers* comerciales involucran redes 802.11 y Bluetooth.

En el presente trabajo se desarrolla una aplicación *sniffer* que brinda la capacidad para capturar los datos transmitidos en las comunicaciones de las redes inalámbricas, enfocándose a las basadas en las tecnologías 802.11 y Bluetooth. Esta aplicación tiene la cualidad para ejecutarse en una PC que tenga instalado algún sistema operativo como Linux, Solaris, MAC OS o Windows, debido a que el lenguaje de programación Java fue utilizado para su desarrollo. La aplicación también cuenta con la capacidad de almacenar los datos capturados en archivos, para que de esta manera, la información capturada pueda ser utilizada en otro momento.

El objetivo principal de este trabajo fue establecer la metodología fundamental para el desarrollo y la implementación de una aplicación *sniffer* orientada a las redes 802.11 y Bluetooth.

La documentación que se presenta en esta tesis, está clasificada en los siguientes capítulos: en el capítulo1 se presenta una breve introducción del proyecto de tesis, tratando los antecedentes de las aplicaciones *sniffer* y los objetivos del proyecto, en la última parte de este capítulo se describen las arquitecturas de seguridad de las tecnologías de comunicación 802.11 y Bluetooth; en el capítulo 2 se habla de los ataques comúnmente implementados en las redes inalámbricas, haciendo mención principal sobre el ataque *sniffing* y los métodos existentes para detectarlo y prevenirlo; en el capítulo 3 se presentan los principales protocolos pertenecientes a las redes 802.11 y Bluetooth, abarcando sus definiciones, sus estructuras y el análisis de algunos paquetes que los incluyen; en el capítulo 4 se lleva a cabo el análisis y el diseño de la aplicación *sniffer* a desarrollar; en el capítulo 5 se describe como fue desarrollada está aplicación y, finalmente, en el capítulo 6 se exponen los resultados obtenidos y el trabajo a futuro.

# Seguridad en Redes Inalámbricas

## 1.1 Introducción

En estos últimos años, las redes de computadoras se han estado expandiendo en todo el mundo, lo que ha ocasionado que muchas personas las hayan aceptado como parte fundamental en su infraestructura de comunicaciones. Debido a los bajos costos de instalación, la seguridad, la confiabilidad y la robustez que proporcionan las redes alámbricas como Ethernet (IEEE 802.3), estas han llegado a ser muy utilizadas en el ámbito empresarial y educativo. Sin embargo, con el avance de la tecnología se han desarrollado e implementado redes inalámbricas que, aparte de igualar las velocidades de transmisión con que las redes alámbricas cuentan, son capaces de proporcionar movilidad a sus usuarios y portabilidad a su estructura física.

En las redes inalámbricas existen tecnologías que son utilizadas de acuerdo a las necesidades del usuario, entre las más usadas se encuentran los protocolos 802.11 y Bluetooth. El protocolo 802.11 es utilizado normalmente en WLANs (*Wireless Local Area Network,* Redes Inalámbricas de Área Local), las cuales son implementadas comúnmente en lugares donde la instalación de cables puede ser difícil. En cambio el protocolo Bluetooth es utilizado para interconectar dispositivos periféricos o móviles con otros dispositivos que se encuentran a distancias cortas. Este protocolo es frecuentemente implementado para WPAN (*Wireless Personal Area Network,* Redes Inalámbricas de Área Personal), IEEE 802.15, aunque también son implementadas en WLANs pero con una tasa de transferencia y un alcance de señal menor[1].

Debido al notable uso de las redes inalámbricas, estas han requerido de medidas de seguridad confiables para proteger la información que manejan de posibles atentados. Para lograr este objetivo, se han estado replanteando los mismos protocolos de seguridad que se implementan con las redes alámbricas. Estos protocolos se basan en los principios de seguridad que las redes de computadoras deben cumplir (confidencialidad, integridad y no repudiación) y deben de resguardar la seguridad de los diferentes ataques.

Existen dos grupos de ataques contra la seguridad de una red: los ataques pasivos, en los que el agresor escucha la información que circula por el medio de transmisión sin llegar a alterar ésta en modo alguno; y los ataques activos, en los que tiene lugar una sustitución, eliminación, falsificación o introducción de información en el flujo de datos existentes en la red.

Este trabajo propone conocer detalladamente el funcionamiento del ataque pasivo *sniffing* para redes 802.11 y Bluetooth, ya que éste se encarga de capturar información del medio de transmisión de la red para después manipularla de forma lícita. Los programas que llevan a cabo este ataque se les conoce con el nombre de *sniffers*.

## 1.2 Antecedentes

Los *sniffers* son aplicaciones que se establecen en una máquina, dentro de una red, para capturar información que no es dirigida a ellas; son aplicaciones que se utilizan para escuchar en la red o monitorear las transmisiones, su objetivo es obtener la información que se esté transmitiendo. Existen dos tipos de *sniffers*, los que son utilizados para escuchar el contenido de los mensajes (conversación telefónica, un mensaje de correo electrónico o una transferencia de archivo) y los que permiten analizar

[1] 1Mbps y 10mts. contra 11Mbps y 100mts de IEEE 802.11, donde Mbps es Mega Bits Por Segundo y mts son metros.

el tráfico de una red. De esta forma, de los usuarios depende que los *sniffers* se usen para fines ilícitos o administrativos.

Se encontraron algunos *sniffers* para redes 802.11 y Bluetooth que trabajan en diferentes plataformas, estos se encuentran disponibles como software libre y como software comercial.

Algunos ejemplos de *sniffers* que trabajan bajo la plataforma Linux son:

- *Kismet*, es un detector de redes 802.11, un *sniffer* y un sistema detector de intrusos. Esta aplicación analiza tráfico 802.11a, 802.11b y 802.11g, y se puede ejecutar en otros sistemas operativos como MacOS, BSD y Linux-ARM [URL 1].
- *AirTraf*, es un analizador de redes 802.11b desarrollado en Java, C y PHP [URL 2].
- *Ethereal*, es un analizador basado en Unix con la capacidad de trabajar principalmente bajo MacOS, OpenBSD, BeOS, Solaris, Iris. Orientado a redes alámbricas y 802.11. Las versiones actuales tienen soporte para mostrar archivos hcidump [URL 3].
- *Mognet*, es un *sniffer* muy sencillo desarrollado en Java orientado a redes 802.11b [URL 4].
- *Hcidump*, es analizador de paquetes Bluetooth que trabaja en la capa HCI y se ejecuta en la consola de Linux [URL 5].

Algunos ejemplos de *sniffers* que trabajan bajo la plataforma Windows son:

- *LinkFerret*, es una herramienta que provee diferentes utilidades como un monitor de red y un analizador de paquetes para redes 802.11b [URL 6].
- *Sniffer Wireless*, es un poderoso analizador comercial con múltiples funciones para la administración de una red. Trabaja sobre redes 802.11 [URL 7].
- *AiroPeek*, es un analizador de paquetes para IEEE 802.11, es comercial [URL 8].
- *FTE4BT,* fue el primer analizador de protocolos y *sniffer* de paquetes Bluetooth v1.2, desarrollado por Frontline Test Equipment, Inc. para uso comercial [URL 9].
- *Merlin I y Merlin II*, son analizadores de paquetes Bluetooth para las versiones 1.1 y 1.2 respectivamente, desarrollado por LeCroy Corporation para uso comercial [URL 10].

La mayoría de los *sniffers* que trabajan sobre el protocolo 802.11 cuentan con características como el análisis de tráfico de paquetes, detección de redes ad-hoc o IBSS (*Independent Basic Service Set,* Conjunto Básico de Servicios Independiente), BSS *(Basic Service Set,* Conjunto Básico de Servicio) y ESS (*Extended Service Set,* Conjunto Extendido de Servicios) así como las estaciones clientes y puntos de acceso, el mapeo gráfico de redes, la identificación de problemas de red, la decodificación de la clave de encriptación WEP (W*ired Equivalent Privacy,* Privacidad Equivalente a la Alámbrica) y la compatibilidad con archivos tcpdump. Estos *sniffers* trabajan con tarjetas de red que cuenten con los chips Prism2, Orinoco, Atheros o Cisco y que además tengan soporte para manejar el modo de monitoreo. En cuanto a los *sniffers* que trabajan sobre el protocolo Bluetooth, estos requieren de hardware especializado para la captura de paquetes en el aire sin haber establecido algún tipo de comunicación anteriormente (FTE4BT, Merlin I y Merlin II).

En la UTM se han desarrollado aplicaciones similares de monitoreo de paquetes para redes alámbricas, como son las tesis de: “Analizador y visualizador de paquetes ICMP, ARP y ARP”, e “IPv6: El nuevo protocolo de Internet, instalación, pruebas y análisis de paquetes en Linux”. Este

trabajo es una continuación de estos, pero ahora, en el área de redes inalámbricas [1][2].

## 1.2 Objetivos

El objetivo general de este proyecto de tesis es desarrollar e implementar una aplicación *sniffer* que trabaje sobre redes inalámbricas IEEE 802.11 y Bluetooth. Para cubrir dicho objetivo, se deben llevar a cabo los siguientes puntos específicos: establecer y documentar el funcionamiento general de una aplicación *sniffer* para redes inalámbricas, y desarrollar la aplicación *sniffer* bajo la plataforma Linux utilizando el lenguaje de programación Java junto con la librería jpcap para 802.11 y la generación de una nueva librería para Bluetooth.

## 1.3 Justificación

La seguridad es una parte fundamental en las redes de computadoras, principalmente en las redes inalámbricas, ya que por la naturaleza del medio de transmisión que utilizan (frecuencia de radio, RF) es muy vulnerable a los diferentes tipos de ataques de seguridad que existen. El *sniffing* es uno de los principales ataques contra la seguridad de una red y por lo tanto es importante analizar su funcionamiento para conocer la forma de combatirlos.

Los protocolos 802.11 y Bluetooth son manejados en este trabajo debido a la comercialización que tienen dentro del mercado de las comunicaciones inalámbricas, es por esto que la aplicación *sniffer* está desarrollada en el lenguaje Java, el cuál además de ser un lenguaje multiplataforma (se puede ejecutar en diferentes sistemas operativos como Linux, Windows, MacOS, etc.) es el lenguaje más apropiado para el desarrollo de aplicaciones Bluetooth [URL 11].

## 1.4 Seguridad inalámbrica

La seguridad es la combinación de procesos, procedimientos y sistemas usados para asegurar la confidencialidad, integridad o disponibilidad de la información. La confidencialidad es la protección de la información de los usuarios no autorizados. La integridad protege el contenido de los datos de las modificaciones no autorizadas. La disponibilidad es el proceso de asegurar que un sistema o dato será accesible cuando se necesite. Con este modelo, algunos de los retos de seguridad más actuales para los dispositivos inalámbricos que requieren de medidas de seguridad incluyen a los dispositivos perdidos y robados, los ataques de los propios miembros de la red, ataques *man-in-the-middle*[2] y la clonación del dispositivo. Otros asuntos de seguridad incluyen a los virus, los ataques de denegación de servicio y los dispositivos de intercepción de señal [3]. El funcionamiento de estos ataques se explicarán en el siguiente capítulo.

Por lo tanto, un protocolo de red debe cumplir con las siguientes funcionalidades:

- *Autenticación.* Se debe validar la identidad del usuario.
- *Encriptamiento.* No admitir *eavesdroppers* (husmeadores) en las transmisiones de datos.
- *Control de acceso.* Asegurar que los usuarios sólo vean la información autorizada para ellos.
- *Robo y terminación del usuario.* Deshabilitar dispositivos cuando estos caen en las manos de un usuario no autorizado.

A continuación se presenta como los protocolos 802.11 y Bluetooth implementan estas funcionalidades.

[2]Estos ataques incluyen la manipulación y el rastreo de la información.

### 1.4.1 Especificaciones de seguridad del IEEE 802.11

El estándar IEEE 802.11, también conocido como WiFi (*Wireless Fidelity,* Fidelidad Inalámbrica*),* define las capas físicas (PHY) y la subcapa MAC (*Medium Access Control,* Control de Acceso al Medio) que establecen los casos de compatibilidad que los fabricantes del equipo inalámbrico deben considerar para la implementación de una WLAN. Dentro de las PHYs se definen los medios de transmisión por ondas de radio frecuencia, que incluyen FHSS (*Frecuency Hopping Spread Spectrum,* Espectro Extenso de Saltos de Frecuencia*)* y DSSS *(Direct Sequence Spread Spectrum,* Espectro Extenso de Secuencia Directa*),* y por ondas infrarrojas (IR). En la subcapa MAC se especifican los estándares de comunicación entre las estaciones móviles clientes y las estaciones base o APs (*Acces Points*, Puntos de Acceso), así como también entre las estaciones clientes. El protocolo del estándar IEEE 802.11 incluye autenticación para establecer la identidad de las estaciones, servicio de asociación y reasociación a un AP, el procedimiento opcional de encriptación WEP, el manejo de energía que reduce el empleo de la energía en las estaciones móviles, el reconocimiento de paquetes que hacen que las transmisiones inalámbricas sean fiables, la sincronización de tiempos para coordinar a las estaciones, la secuenciación con detección de duplicación y recuperación de datos, y la fragmentación y el reensamblaje de paquetes [4].

#### 1.4.1.1 La PHY

Las PHYs de las WLANs ofrecen ciertos beneficios y limitaciones de seguridad que dependen de la naturaleza del medio físico que utilizan. Las WLANs que implementan el medio IR no son muy empleadas en el ámbito empresarial debido a que cuentan con más limitaciones que otras PHYs como las ondas de radio frecuencia. Aunque sus velocidades son similares a las ondas de radio frecuencia, IR restringe la conectividad de la red a un espacio en donde no existen obstáculos como las paredes, el piso o el techo de alguna construcción. Por otro lado, si se manejan WLANs con transmisiones vía IR en exteriores, las transmisiones podrían ser afectadas por los rayos ultravioleta que el sol emite, ocasionando posibles pérdidas de información.

En otro caso, en las ondas de radio frecuencia, las señales viajan sobre un extenso rango de frecuencias para hacer que las comunicaciones sean confiables, íntegras y seguras. Esto permite que los dispositivos que trabajan sobre esta tecnología eviten la interferencia con otros dispositivos y sean menos vulnerables ante las señales de ruido.

En FHSS, los efectos de la interferencia son controlados debido al uso de un código de saltos que se encuentra programado en los transceptores, en donde este código define el orden y las frecuencias en las que la señal portadora de datos debe basar el comportamiento de su trayectoria. Por lo tanto, para que la señal sea interceptada, los transceptores deben sincronizar sus transmisiones de acuerdo al código de saltos, y en dado caso que existiera interferencia, la señal de datos se retransmitiría sobre otra frecuencia. En cambio con la tecnología DSSS, en lugar de usar una serie de saltos de frecuencias se usa un conjunto de bits redundantes que son insertados en cada bit transmitido, y por medio de estos bits, los receptores son capaces de ejecutar rutinas de recuperación de datos en señales basadas sobre análisis estadísticos. De esta forma, se evita la retransmisión de la información en casos de interferencia. Como consecuencia de esto, DSSS llega a alcanzar velocidades de transmisión arriba de 11Mbps a distancias de kilómetros y FHSS de 2 a 3 Mbps a una distancia de 1000 pies.

#### 1.4.1.2 La subcapa MAC

El IEEE 802.11 provee servicios de seguridad que se encuentran implementados en la subcapa MAC. Estos servicios de seguridad son el servicio de autenticación y el mecanismo de encriptación WEP.

### 1.4.1.2.1 Autenticación

La IEEE 802.11 define dos esquemas de autenticación, la autenticación por sistemas abiertos (o*pen system*) y la autenticación por clave compartida (s*hared key*).

La autenticación o*pen system* es una autenticación nula, en donde todas las terminales móviles que solicitan acceso son aceptadas en la red. Este tipo de autenticación es la autenticación predeterminada que se maneja en las redes inalámbricas. Esta autenticación es usada principalmente por los establecimientos que brindan servicios que no requieren de ningún nivel de seguridad.

En la autenticación *shared key* se usa una clave compartida que sirve para autenticar a las terminales móviles. El mecanismo de encriptación WEP debe estar implementado en la red para habilitar este tipo de autenticación. El proceso de autenticación *shared key* consiste en que, cuando una terminal móvil le solicita autenticación a la terminal base o en otro caso a un AP, este le envía un número aleatorio de 128 bytes encriptado, haciendo uso para la encriptación de la clave compartida con el mecanismo de encriptación WEP. La terminal móvil vuelve a encriptar el número aleatorio encriptado recibido, usando la misma clave compartida, y lo envía de regreso a la terminal base. Si el resultado de desencriptar la información que recibe la terminal base coincide con la enviada, la terminal móvil es aceptada en la red. Todas las terminales móviles que se les permitió conectarse a la red usan la misma clave compartida, es por eso que este método de autenticación sólo se habilita para verificar si la terminal móvil pertenece al grupo de las terminales con permisos para conectarse a la red, aunque no haya forma de distinguir a las terminales una de otra y tampoco haya medios para que la terminal móvil autentique a la red. El IEEE 802.11 no define funciones de manejo de claves compartidas, ni tampoco especifica como se deben distribuir estas a cada estación. La responsabilidad de la distribución de las claves recae en el administrador de la red.

La autenticación de las terminales móviles también se puede llevar acabo con el manejo del SSID (*Service Set Identifier*, Identificador del Conjunto de Servicio) de una red. El SSID consta de una serie de caracteres que identifica a alguna red inalámbrica en particular y es comúnmente usado para controlar el acceso de las terminales, es decir, si alguna terminal móvil desea integrarse a una WLAN, esta debe de proporcionarle el SSID correspondiente para que pueda ser aceptada. El SSID es también usado para la distribución del tráfico hacia una red en particular, lo cual involucra que éste pueda ser fácilmente extraído desde cualquier paquete que circula en la red.

Otra forma rudimentaria de autenticación se lleva a cabo con el uso de una lista de control de acceso basada en las direcciones MAC de los usuarios autorizados. Esta lista está incluida en los APs y su disponibilidad depende de las empresas que los fabrican. Sin embargo, la seguridad brindada es pobre debido a que las direcciones MAC pueden ser falsificadas u obtenidas del tráfico de la red fácilmente.

### 1.4.1.2.2 WEP

El IEEE 802.11 define al WEP como un mecanismo opcional para implementar la confidencialidad e integridad del tráfico de una red. El WEP es usado para proveer el proceso de encriptación y desencriptación para la autenticación de una terminal móvil. Sin embargo, es importante mencionar que WEP sólo provee niveles de seguridad de terminal a terminal y no de extremo a extremo, es decir, únicamente brinda seguridad de transmisión entre dos terminales móviles, entre una terminal móvil y una terminal base, o entre dos terminales base.

El mecanismo de encriptación WEP se basa en el uso de una clave confidencial, la cual se comparte entre las terminales móviles y las terminales base para llevar acabo el proceso de autenticación *shared key* y para la encriptación de los datos que son enviados y recibidos entre las terminales. El uso de la misma clave para el proceso de autenticación y encriptación es un riesgo en la seguridad de una red, es por esto, que el estándar IEEE 802.11 sugiere que la mejor solución para que una red 802.11 esté protegida, es usando la clave compartida en el mecanismo de encriptación WEP e implementando un

método de autenticación diferente a la autenticación *shared key*.

En la encriptación WEP (Ver figura 1.1) se usa una *clave secreta* de 40 ó de 128 bits, según se requiera el nivel de seguridad, y un PRGN (*Pseudo Random Number Generator*, Generador de Números Seudoaleatorios). Dos procesos son aplicados a los datos o *texto llano,* uno encripta el texto llano y el otro los protege de la modificación no autorizada mientras se están transmitiendo. La clave secreta se concatena con un IV (*Initialization Vector*, Vector de Inicialización) aleatorio de 24 bits, resultando una nueva clave (*semilla*) que es la entrada del PRGN. El PRGN genera una *clave seudoaleatoria* que es usada para encriptar los datos a través de la operación XOR. La clave seudoaleatoria tiene la misma longitud, en bytes, que el texto llano más 4 bytes que conforman al ICV (*Integrity Checked Value*, Valor de Chequeo de Integridad). El ICV se obtiene de aplicarle al texto llano un algoritmo de integridad[3] para protegerlo de posibles modificaciones no autorizadas que pudiera haber sufrido durante su transmisión. EL ICV se concatena con el texto llano para después ser encriptado con la operación XOR a nivel de bit, que se vale de la clave seudoaleatoria, para que de esta forma se genere el texto encriptado o *texto cifrado*. El texto cifrado se transmite junto con el IV para que se pueda llevar acabo el proceso de desencriptación del *mensaje*. La figura 1.2 muestra detalladamente este proceso.

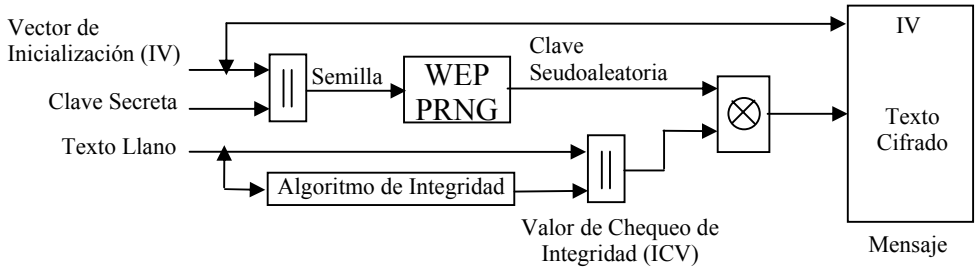


Figura 1.1 - Proceso de encriptación WEP

Para el proceso de desencriptación el receptor debe contar con la clave secreta, para que junto con el IV y el PRNG, se genere la clave seudoaleatoria de la misma forma como se mencionó en el proceso de encriptación. Recordando que el IV, al igual que el texto cifrado, se extrae del mensaje recibido, es así como al aplicar la operación XOR a nivel de bit entre la clave seudoaleatoria y el texto cifrado, se desencripta el mensaje y se obtiene el texto llano con su respectivo ICV que el transmisor ha enviado. De aquí que, para checar la integridad de los datos, si se verifica que el ICV recibido es igual al ICV generado por el receptor (ICV') se determina que el mensaje no fue modificado en el transcurso de su transmisión y por lo tanto es auténtico.

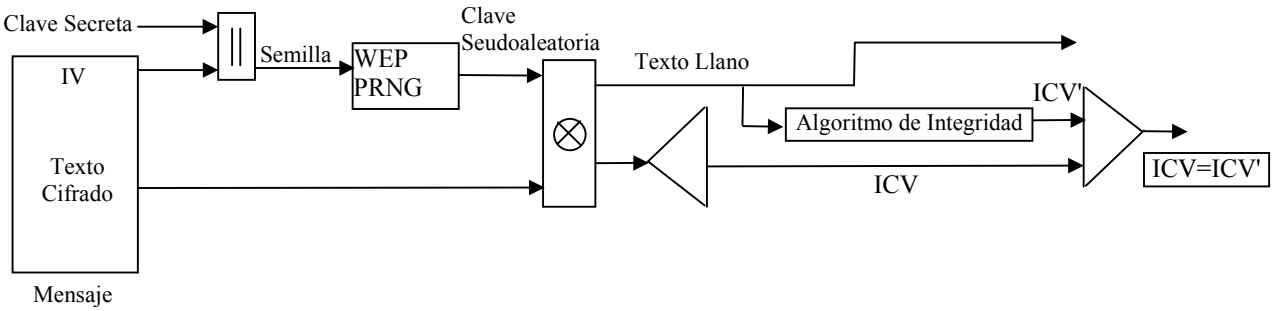


Figura 1.2 – Proceso de desencriptación

[3]El algoritmo de integridad usado es el CRC-32 (*Cyclic Redundant Code*, Código de Redundancia Cíclica) que genera un ICV de 32 bits.

A pesar de la fortaleza de WEP para proteger la integridad y la confidencialidad de los datos, este presenta algunas limitaciones. La inclusión del IV en los mensajes para que los receptores puedan realizar la desencriptación, resulta conflictiva si el IV es frecuentemente reutilizado en transmisiones posteriores, dando origen a que algún *eavesdropper* pueda ser capaz de criptoanalizar la clave seudoaleatoria generada por el IV y la clave secreta, y por lo tanto pueda desencriptar los mensajes que usan ese mismo IV. De esta forma, algunos agresores podrían hacer uso de la clave seudoaleatoria para crear mensajes válidos e insertarlos en la red. Por esto, que cambiar el IV después de cada mensaje transmitido es un método simple para preservar la efectividad del mecanismo de encriptación WEP.

Otro problema es que la confidencialidad de los datos dependa de algún manejo externo para la distribución de las claves secretas que se comparten entre las terminales base y las terminales clientes, ya que es improbable que la clave compartida entre los usuarios permanezca definitivamente confidencial. Es por esto, que los administradores se encargan de la distribución de las claves para que los usuarios no se involucren en esta función. Aunque esta no es la mejor solución debido a que las claves permanecen guardadas en las terminales de los usuarios en donde estas son vulnerables.

### 1.4.2 Especificaciones de seguridad de Bluetooth

El protocolo Bluetooth va dirigido a las comunicaciones inalámbricas que requieren de un bajo costo y un bajo consumo de energía a distancias cortas entre dispositivos móviles y redes de área local (ya sean alámbricas o inalámbricas), soportando la comunicación de voz y de datos en transmisiones punto a punto o de punto a multipunto. Inherente al estándar IEEE 802.11, Bluetooth cuenta con especificaciones en la PHY y en la capa de enlace, siendo esta última, en donde se ejecuten todas las funciones de seguridad [5][6].

Al igual que 802.11, Bluetooth utiliza como medio de transmisión las ondas de radio frecuencia en la banda de 2.4 GHz, implementando la técnica FHSS para evitar interferencias con otros dispositivos. Para prevenir las interferencias, FHSS cambia la frecuencia de transmisión 1600 veces por segundo y en caso de que existiera una interferencia en alguna frecuencia, la señal se retransmitiría en otra. En la transmisión de los datos, FHSS puede operar sobre 23 ó 79 canales para los saltos de frecuencia. El patron de estos saltos es establecido por el dispositivo maestro en cada piconet[4].

#### 1.4.2.1 Arquitectura de seguridad

Las funciones de seguridad de Bluetooth están definidas en la arquitectura de seguridad de esta tecnología y principalmente se aplican en la capa de enlace (Ver figura 1.3). Algunas de estas proveen funcionalidades para los mecanismos de autenticación y de encriptación.

En las especificaciones de seguridad de Bluetooth se manejan tres modos de seguridad. En el *modo 1* no se ejecuta ninguna función de seguridad, siendo este el modo más inseguro. En el *modo 2* las funciones de seguridad son aplicadas después del establecimiento del canal, es decir, de que la solicitud de conexión entre los dispositivos, hecha por el *manejador de enlace,* haya sido aceptada. Este modo de seguridad se aplica sobre el nivel de L2CAP (*Logical Link Control and Adaptation Protocol*, Protocolo de Control y Adaptación del Enlace Lógico), permitiendo el establecimiento de diferentes políticas de seguridad flexibles para las *aplicaciones*, incluyendo aquellas que se estén ejecutando en paralelo con diferentes requisitos de seguridad. En este modo, el *manejador de seguridad* es utilizado para decidir si se implementarán los mecanismos de autenticación y encriptación, basándose en las *bases de datos de servicios y dispositivos* (Ver figura 1.3). A este modo de seguridad también se le conoce como seguridad aplicada a nivel de servicios.

[4]Red de dispositivos Bluetooth que se forma entre un dispositivo maestro y sus dispositivos esclavos, con una capacidad máxima de hasta 8 dispositivos. Dos o más piconets interconectadas forman una scatternet.

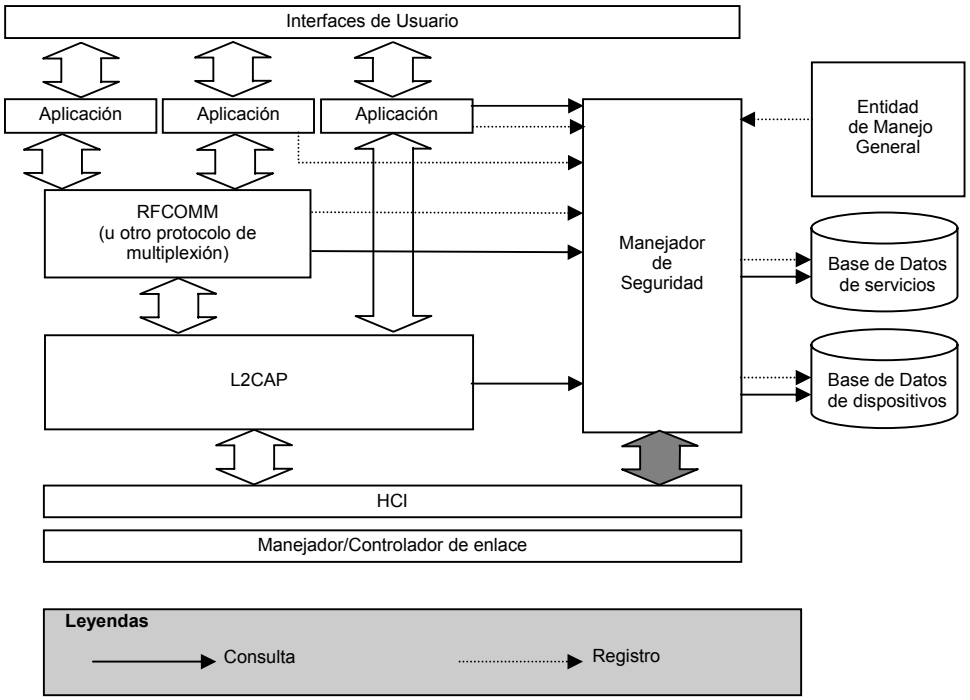


Figura 1.3 – Arquitectura de Seguridad

En el *modo 3* las funciones de seguridad son aplicadas antes del establecimiento del canal sobre el nivel LMP (*Link Manager Protocol*, Protocolo del Manejador de Enlace). LPM puede aplicar autenticación y/o encriptación a la conexión dependiendo de las respuestas que le proporciona el manejador de seguridad a través del HCI *(Host Controller Interface*, Interfaz del Controlador del Computador). A este modo de seguridad también se le conoce como seguridad aplicada al nivel de enlace.

Bluetooth permite niveles de seguridad que son definidos por los dispositivos y los servicios, y son controlados por el manejador de seguridad. Para los dispositivos existen dos niveles de seguridad: los *dispositivos de confianza*, que son aquellos que tienen acceso a todos los servicios debido a una relación definida por una autenticación previa, y los *dispositivos sin confianza* los cuales tienen el acceso restringido a los servicios ya sea por una relación definida por una autenticación previa o por el hecho de que el dispositivo sea desconocido. Toda la información referente a los dispositivos que surge de las autenticaciones es almacenada en la base de datos de dispositivos.

Los servicios cuentan con tres niveles de seguridad que pueden requerir de los mecanismos de autorización, autenticación y encriptación. La información acerca de estos mecanismos aplicados a los servicios es almacenada en la base de datos de servicios. En el primer nivel los dispositivos que ya hayan sido autenticados pueden requerir de algún mecanismo de autorización para tener acceso a los servicios. En caso de que los dispositivos sean de confianza el acceso a los servicios es automático, pero si fueran sin confianza es necesaria la autorización manual. La autorización puede hacerse manualmente con el uso del PIN (*Personal Identification Number*, Número Personal de Identificación) de cada dispositivo. En el segundo nivel el dispositivo debe ser autenticado para obtener el acceso a los servicios. En cambio, en el tercer nivel de seguridad para el acceso a los servicios no es necesario ningún mecanismo de autorización o autenticación. En los tres niveles de seguridad se puede requerir o no del mecanismo de encriptación para resguardar la confidencialidad de los datos.

### 1.4.2.2 Autenticación

El mecanismo de autenticación es usado por un dispositivo (*verificador*) para identificar a otro quien le ha realizado una petición de conexión (*demandante*). Para esto, después del envío de la petición de conexión, el dispositivo demandante recibe un número aleatorio de 128 bits ($AU\_RAND_A$) por parte del dispositivo verificador, con el cuál podrá obtener un valor calculado con la ayuda del algoritmo de autenticación $E_1$ (SRES). Este valor es enviado al dispositivo verificador para ser comparado con el valor calculado por este dispositivo (SRES'), y en caso de que ambos coincidieran, se podrá determinar que el dispositivo demandante ha sido autenticado (SRES=SRES'). En la figura 1.4 se presenta este proceso.

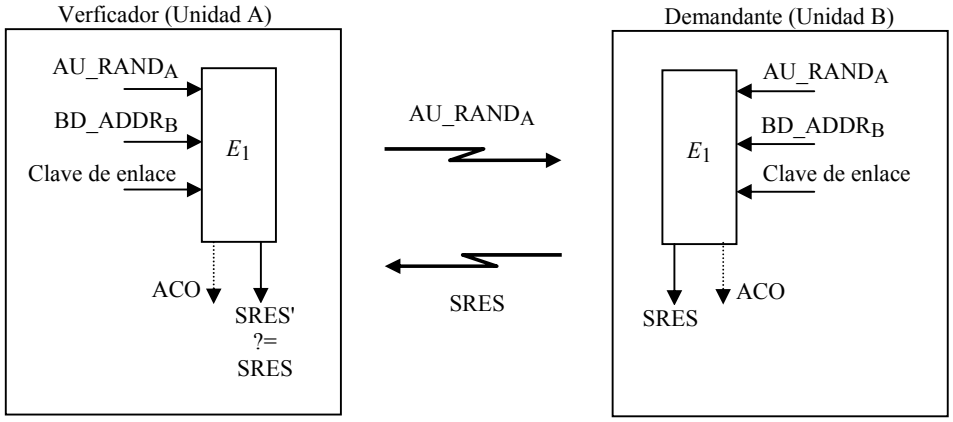


Figura 1.4 – Proceso de Autenticación

Para generar los SRESs, $E_1$ hace uso de la dirección del dispositivo demandante ($BAD\_ADDR_B$), una clave de enlace determinada y el $AU\_RAND_A$. Además, como se observa en la figura, $E_1$ también genera el parámetro ACO (*Authenticated Ciphering Offset*, Compensación Cifrada Autenticada) pero este es utilizado en el mecanismo de encriptación. El BD_ADDR utilizado es una dirección de 48 bits definida por el IEEE y es única para cada dispositivo. Las claves de enlace, por cuestiones de seguridad, pueden ser temporales o semipermanentes y al momento de transmitirse deben estar previamente encriptadas para evitar que sean legibles si llegaran a ser capturadas por algún *eavesdropper*. De acuerdo a las siguientes circunstancias estas claves pueden llegar a ser generadas:

- Por una unidad (clave de unidad, $K_A$), cuando un dispositivo Bluetooth es instalado desde una unidad A. Esta clave es usada en todas las conexiones hechas por A.
- Por cada par de unidades A y B (clave de combinación, $K_{AB}$), para obtener una mayor seguridad. Esta clave es usada únicamente en las conexiones realizadas entre A y B.
- Por un dispositivo maestro (clave maestra, $K_{master}$), para la realización de transmisiones multipunto.
- Durante la inicialización de un dispositivo (clave de inicialización, $K_{init}$), cuando no se logra establecer una clave de enlace $K_{AB}$.

Las claves $K_A$ y $K_{AB}$ se crean a través del algoritmo $E_{21}$ utilizando como parámetros un número aleatorio (RAND) y la dirección del dispositivo (BD_ADDR). Por otro lado, las claves $K_{init}$ y $K_{master}$ se crean a través del algoritmo $E_{22}$ utilizando un RAND y un PIN común para cada dispositivo. El RAND es un número aleatorio de 128 bits comúnmente generado por software, también generado para el proceso de encriptación. PIN es un número seleccionado por el usuario que puede ser de 1 hasta 16 bytes, dependiendo del nivel de seguridad que se desea agregarle al sistema. Este puede también ser fijado dentro de un dispositivo aunque esta situación no es muy segura. $E_{21}$ y $E_{22}$, al igual que $E_1$, son algoritmos basados en el algoritmo de cálculo por bloque SAFE+ (*Secure And Fast Encription Routine*,

Rutina de Encriptación Rápida y Segura).

### 1.4.2.3 Encriptación

El mecanismo de encriptación se puede implementar después de que los dispositivos hayan sido autenticados. Estos dispositivos pueden solicitar un modo de encriptación para hacer que las transmisiones sean seguras y los datos sean confidenciales, ya sea en transmisiones punto a punto o en transmisiones multipunto. Durante el proceso de autenticación se crea una clave de enlace la cual se utiliza para generar la clave de encriptación a través del algoritmo $E_3$ basado en SAFE+. La longitud de la clave de encriptación se puede establecer de entre 8 hasta 128 bits, esto se debe a las diferentes especificaciones de encriptación que las leyes de los países pueden marcar y a que de esta forma se facilitaría el acoplamiento de seguridad de próximas actualizaciones, evitando así el rediseño de los dispositivos Bluetooth.

Para que el mecanismo de encriptación se lleve a cabo, el dispositivo esclavo debió haber aceptado la petición de encriptación que el dispositivo maestro le envió. Si el dispositivo esclavo aceptó y si la clave de enlace que establecieron es una clave maestra, entonces se podría implementar cualquiera de estos tres modos de encriptación: no se aplicaría la encriptación en sus transmisiones, el tráfico de las transmisiones multipunto no serían encriptadas pero el tráfico de las transmisiones punto a punto si o tanto las transmisiones multipunto como las punto a punto serían encriptadas. Después de haber elegido el modo de encriptación, los dispositivos deben negociar la longitud de la clave de encriptación que usarán basándose en los argumentos mencionados, y si en dado caso no llegaran a un acuerdo, la negociación se abortaría y el proceso de encriptación no podría iniciarse.

El proceso de encriptación (Ver figura 1.5) comienza cuando el dispositivo maestro (Unidad A) le envía al dispositivo esclavo (Unidad B) un número aleatorio de encriptación ($EN\_RAND_A$) para que este pueda generar su clave de encriptación ($K_C$). Para generar $K_C$ se aplica $E_3$, utilizando como parámetros de entrada el $EN\_RAND_A$, la clave de enlace y el COF (*Ciphering Offset Number,* Número de Compensación Cifrado). El COF es un número de 96 bits que puede ser la concatenación de la dirección de la unidad A ($BD\_ADDR_A$ U $BD\_ADDR_A$), si la clave de enlace es una clave maestra, o el ACO en caso de que la clave no lo fuera. El proceso de encriptación esta basado en el algoritmo de cálculo por flujo $E_0$, el cual se vale de los valores de $K_C$, $BD\_ADDR_A$ y la señal del reloj de la unidad A ($Reloj_A$) para generar una clave cifrada ($K_{cifrado}$). El dispositivo A se encarga de sincronizar su reloj con el reloj de la unidad B, esto con la finalidad de que $K_{cifrado}$ sea diferente para cada paquete y se obtenga un mayor grado de seguridad. Con la creación de $K_{cifrado}$ se pueden encriptar los datos aplicando la operación XOR entre estos dos parámetros ( $Datos_{A\text{-}B} \otimes K_{cifrado}$). Para el proceso de desencriptación los datos se obtienen de forma similar, únicamente se realiza la operación XOR entre los datos encriptados y $K_{cifrado}$.

En la actualidad, el proceso de encriptación ha presentado algunas anomalías que comprometen la seguridad de Bluetooth. El algoritmo utilizado $E_0$ ha sido el objetivo central de algunos ataques para obtener la clave de encriptación ($K_C$). Sin embargo, estos no han podido ser implementados debido a la alta complejidad computacional que requieren. Además, existen estudios prácticos que se han estado enfocando sobre métodos para adivinar o robar las claves de enlace utilizadas para generar las claves de encriptación y que son también utilizadas en el mecanismo de autenticación. Estos estudios, principalmente, están orientados en las claves establecidas durante los procedimientos de comunicación que se llevan a cabo entre dos dispositivos previamente desconocidos (*Pairing*).

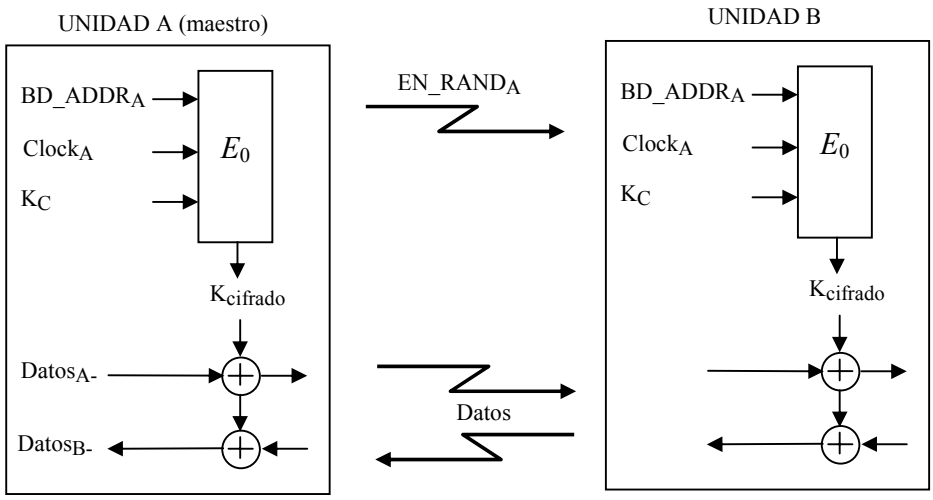


Figura 1.5 – Proceso de Encriptación

CAPÍTULO 2

# Ataques Inalámbricos

## 2.1 Introducción

En algunas organizaciones la información que se maneja puede llegar a ser invaluable, es por eso que deben contar con un diseño de seguridad rígido tanto en sus políticas de seguridad como en su infraestructura de comunicaciones. De esta forma se logra que la información sea íntegra, auténtica, confidencial y esté disponible en el momento que se le requiera, permitiendo que se alcancen los objetivos organizacionales. Cuando se implementa la tecnología inalámbrica en la infraestructura de comunicaciones, es importante considerar las vulnerabilidades que esta presenta, debido a los costos que implicarían los daños ocasionados por los diferentes ataques de seguridad a las que está expuesta.

Los ataques son operaciones desautorizadas y perjudiciales, con los cuales los atacantes o agresores satisfacen sus beneficios personales o simplemente lo llevan acabo para hacer daño. Estos agresores son comúnmente personas externas a las empresas que pueden obtener acceso a la red con el simple hecho de "husmear" sobre las comunicaciones inalámbricas. Otras formas de ataque se llevan acabo a través de aplicaciones maliciosas que involucran a los virus, gusanos, caballos de Troya u otro software indeseable que daña o puede devastar un sistema.

Al igual que las redes alámbricas, las redes inalámbricas presentan las mismas vulnerabilidades de seguridad, además de otras que son inherentes a la naturaleza de su protocolo y medio físico que manejan. Estas vulnerabilidades consisten básicamente de las facilidades de explotación que los medios físicos y los mecanismos de autenticación y encriptación presentan, siendo estos últimos en los cuales la mayoría de los ataques están basados. De esta forma la información puede ser capturada o afectada para fines lícitos o ilícitos, haciendo referencia a la información como los datos, los procesos o los servicios que una organización dispone.

En este capítulo se presentará un bosquejo general de los ataques más específicos de las redes inalámbricas, enfocándonos principalmente en el ataque *sniffing,* siendo éste del cual parte este trabajo de tesis.

## 2.2 Clasificación de los ataques inalámbricos

En general, en las redes de computadoras existen diversas maneras de agrupar a los diferentes tipos de ataques. Una manera sencilla y comprensible de agruparlos es de acuerdo a la forma en que estos se implementan, ya sea de forma pasiva o activa. En las redes inalámbricas, los ataques pasivos más sobresalientes se encuentran el *sniffing* y el *scanning,* y por los ataques activos el *spoofing, hijacking, man-in-the-middle y jamming* [7][8].

### 2.2.1 Ataques pasivos

Estos ataques son los más frecuentes y fáciles de implementar sobre las comunicaciones inalámbricas debido a los bajos costos que involucra su desarrollo. Estos ataques se llevan a cabo cuando alguien escucha u obtiene acceso al tráfico de un red sin llegar a alterar su contenido, siendo esta la principal causa para que estos ataques sean difíciles de detectar (Ver figura 2.1). Un ataque pasivo sobre una red inalámbrica puede o no llegar a ser malicioso, ya que pueden ser usados para fines educativos o actividades administrativas. Existen pocas formas en que un ataque pasivo se puede llevar a cabo, los comúnmente usados serán explicados a continuación.

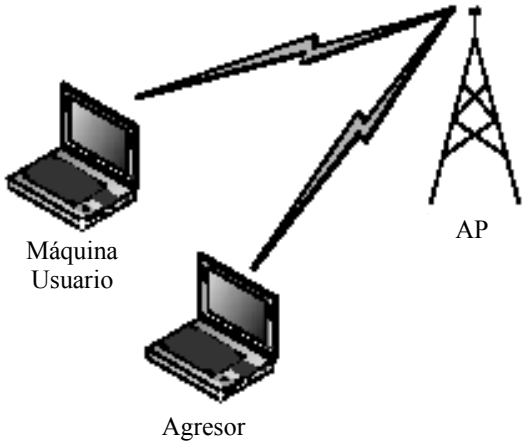


Figura 2.1 – Ataque *sniffing* sobre una red inalámbrica

### 2.2.1.1 *Sniffing*

En *sniffing* existen diferentes técnicas para la reunión de información, entre las más usadas se encuentran el *eavesdropping* (husmeo) y el *análisis de tráfico*.

El *eavesdropping,* al igual que por análisis de tráfico*,* se hace uso de una tarjeta o adaptador para redes inalámbricas que trabaje sobre el mismo rango de frecuencias y use el mismo método de transmisión que emplea la red inalámbrica objetivo, permitiendo así que el agresor pueda capturar el tráfico transmitido sobre esta red. Para esto, los adaptadores deben contar con características propias de cada tipo de red. Por ejemplo, en las redes 802.11 se utilizan adaptadores que soportan el modo de operación por monitoreo[5], con el cual se podrán recibir todos los paquetes, conteniendo información 802.11, que se encuentran al alcance del adaptador en una frecuencia específica. Sin este modo, el adaptador no podrá capturar el tráfico sin antes haber establecido algún tipo de comunicación y los paquetes sólo contendrán información Ethernet o de capas de nivel superior. En redes Bluetooth ocurre algo similar, es necesario contar con adaptadores que tengan la capacidad de sincronizarse[6] con la piconet objetivo para después colocarse en modo de monitoreo y así poder capturar el tráfico de banda base que se encuentra a su alcance. Es importante mencionar que los adaptadores Bluetooth con este tipo de características son difíciles de conseguir y por lo tanto su uso es limitado al área comercial. Por esto, las empresas dedicadas a la comercialización de *sniffers* Bluetooth se apoyan de hardware especializado que muchas veces es manufacturado por ellos mismos.

Con la ayuda de alguna aplicación *sniffer* que tenga soporte para estos tipos de adaptadores de red, en *eavesdropping*, el agresor puede explorar transmisiones y descubrir redes inalámbricas sin ningún esfuerzo. Algunos *sniffers* cuentan con herramientas para romper la seguridad que los mecanismos de encriptación ofrecen en las transmisiones inalámbricas. Estas herramientas consisten principalmente en descifrar las claves que utilizan los mecanismos de encriptación a través de la aplicación de algoritmos estadísticos al tráfico de la red objetivo. Con la obtención de estas claves, los datos transmitidos pueden ser descifrados fácilmente permitiendo que la información llegue a ser legible para el agresor. Herramientas como *AirSnort* o *WEPCrack* cuentan con estas capacidades [URL 12][URL 13].

En el análisis de tráfico o análisis del flujo del tráfico, el agresor monitorea las transmisiones inalámbricas para obtener conocimiento detallado sobre el diseño de la red. Este conocimiento consiste principalmente en conocer que tipo de usuarios hacen uso de la red, que información puede ser accesible, cuales son las capacidades de los equipos clientes sobre la red, en que momentos y cuando la

[5]Conocido también como modo rfmon (*Radio Frecuency Monitoring*, Monitoreo de Radio Frecuencia).
[6]Emparejamiento del reloj interno del adaptador.

red es menor y mayormente usada, y cuál es el área cubierta por la red. Este conocimiento es frecuentemente usado para tomar decisiones con relación a la administración de la red, por ejemplo, con el análisis del tráfico se pueden descubrir posibles cuellos de botella, fugas de tráfico, controlar el acceso a los servicios y demás. Para la implementación de estos tipos de ataques, una considerable cantidad de información es reunida del flujo de los mensajes transmitidos entre los equipos.

#### 2.2.1.2 *Scanning*

El *scanning* es el acto de explorar a los dispositivos inalámbricos con el fin de encontrar y explotar servicios o procesos. En redes 802.11, este ataque es implementado principalmente para descubrir dispositivos o redes disponibles que se encuentren al alcance del agresor. Para llevarlo a cabo se utilizan herramientas como *Netstumbler* [URL 14]. En redes Bluetooth, este ataque es frecuentemente implementado sobre dispositivos como celulares, PDAs (*Personal Digital Assistant*, Asistente Digital Personal) y laptops. Los agresores utilizan herramientas como *bluesniff*, para encontrar dispositivos Bluetooth no descubribles y conocer los servicios con que cuentan. Con esta información, los agresores son capaces de explotar dichos servicios y de esta forma comprometer la seguridad del dispositivo [URL 15].

### 2.2.2 Ataques Activos

Cuando algún agresor realiza modificaciones a los mensajes, flujos de datos o archivos, se dice que se está cometiendo un ataque activo. Debido a estas modificaciones, es posible detectar a estos tipos de ataques aunque estos no puedan evitarse. Una vez que el agresor haya obtenido suficiente información de alguna red inalámbrica con el uso de algún ataque pasivo, puede iniciar un ataque activo en contra de esa red. Existen muchas variaciones de ataques activos, la mayor parte de estos son idénticos a los ataques que se encuentran en las redes alámbricas. Entre los ataques más específicos en redes inalámbricas se encuentran los que incluyen acceso desautorizado como el *spoofing*, modificación de contenido como el *hijacking* y *man-in-the-middle*, y los que incluyen denegación de servicios como el *flooding* o el *jamming*.

#### 2.2.2.1 *Spoofing*

El *spoofing* es el ataque activo frecuentemente implementado en las redes inalámbricas debido a su sencillez. En este ataque el agresor es capaz de usar un dispositivo no autorizado para hacerlo pasar por algún dispositivo válido o autorizado que pertenezca a la red objetivo. En algunas redes inalámbricas, se utilizan como medios de seguridad la aplicación de métodos de filtración de direcciones MAC para permitir el acceso a ciertos dispositivos. En este contexto, para que el agresor pueda adquirir privilegios y acceso a los servicios de la red, únicamente debe asignarle direcciones MAC válidas a su dispositivo. Dependiendo de los privilegios adquiridos, el agresor podría tener acceso a la información importante desde la manejada en servicios básicos como las cuentas de email hasta la que se relaciona con la administración de una red.

Otra manera de implementar este ataque, es con la asignación de direcciones IP válidas al dispositivo del agresor, en caso de que la red objetivo use filtrado de este tipo de direcciones. Para encontrar direcciones IP o MAC válidas únicamente se necesita capturar y analizar el tráfico de una red con la ayuda de un *sniffer*.

Actualmente sobre las redes 802.11, se puede ejecutar un particular tipo de *spoofing* que se basa en las vulnerabilidades que presenta el mecanismo de autenticación utilizado por estas redes. Este ataque consiste principalmente en descifrar la clave compartida (*shared key*), usada en el mecanismo de autenticación, con la ayuda de alguna herramienta como *AirSnort* o *WEPCrack*. Con la obtención de la clave compartida, el agresor se puede autenticar como un usuario legítimo y podrá tener acceso a los

servicios de la red. En redes Bluetooth ocurre algo similar, el agresor es capaz de obtener las claves de enlace o el PIN de algún dispositivo para hacerse pasar por un usuario legítimo.

#### 2.2.2.2 *Hijacking*

El *hijacking* consiste en "el secuestro" de una red inalámbrica o una sesión. Este tipo de ataque puede llegar a ser indetectable para los administradores, provocándoles la incapacidad de diferenciar a el agresor de algún usuario legítimo. Para llevar a cabo este ataque existen diversas técnicas de implementación, en su mayoría se involucran a los dispositivos de red que se encargan de distribuir el tráfico correspondiente a cada máquina terminal, entre los cuales se pueden mencionar a los enrutadores y APs. Muchos de estos dispositivos cuentan con tablas dinámicas en donde se almacenan las direcciones IP y/o MAC pertenecientes a las máquinas terminales. Estas tablas son construidas de acuerdo al tráfico que pasa a través de los dispositivos distribuidores o también por las peticiones ARP que realizan las máquinas terminales. Es por eso que los agresores, con el envío de mensajes ARP, pueden asignarle una dirección IP conocida a su dirección MAC para poder ser registrados en las tablas dinámicas de los dispositivos enrutadores, debido a que estos no cuentan con algún tipo de autenticación para máquinas terminales válidas. De esta forma, todo el tráfico que vaya dirigido a la dirección IP es entregada a la máquina del agresor.

En otro caso, dependiendo de las habilidades del agresor, el *hijacking* se puede implementar con la ayuda de algún otro ataque. Por mencionar alguno, si el agresor se hiciera pasar por la puerta de enlace (*gateway*) o alguna otra máquina específica dentro de una red a través de *spoofing*, entonces todas las máquinas o la máquina atacada se conectarán a la máquina del agresor, permitiéndole la capacidad de obtener información confidencial como nombres de usuarios y passwords.

#### 2.2.2.3 Ataques *man-in-the-middle*

Los ataques *man-in-the-middle* son diseñados para corromper la confidencialidad e integridad de las sesiones. Estos ataques requieren de información significativa acerca de la red y normalmente el agresor la utiliza para hacerse pasar por algún recurso perteneciente a esta.

En redes 802.11, los ataques *man-in-the-middle* suelen ser implementados con el uso de un AP (*Access Point*, Punto de Acceso) malicioso. La forma más conocida, se lleva a cabo cuando el agresor coloca un AP malicioso de amplia cobertura, en el rango entre alguna computadora usuario y un AP legítimo. Si se le asigna al AP malicioso el SSID usado por la red, el cuál puede ser fácilmente obtenido con la ayuda de algún *sniffer*, la computadora usuario no sabrá si se conectará con algún AP no autorizado. Si el AP malicioso cuenta con una excelente señal, el agresor conseguirá que la computadora usuario se conecte al AP malicioso y así podrá obtener información invaluable acerca de la red, tal como de las peticiones de autenticación se logrará obtener la clave compartida que puede estar en uso. En la implementación de este ataque, los agresores suelen usar una laptop con dos tarjetas inalámbricas, una que se desempeñará como el AP malicioso y la otra para retransmitir las peticiones hacia el AP legítimo. Es por esto que, si el agresor cuenta con los suficientes conocimientos, cuando una terminal usuario intente establecer una conexión con algún otro dispositivo de red, el agresor la interceptará y la podrá terminar de establecer, adquiriendo la capacidad de poder introducir datos o modificar las comunicaciones.

En Bluetooth, el agresor aprovecha la mala configuración de seguridad que pueden tener dos dispositivos víctimas para poder llevar a cabo este ataque. Una implementación se puede realizar con la obtención de las claves de enlace y los BD_ADDR de los dispositivos, ya sea por medio de un *sniffer* o por métodos de adivinación. Con esta información el agresor puede interceptar y crear nuevos mensajes para ambos dispositivos, o puede tener la capacidad para establecer dos comunicaciones punto a punto y por lo tanto hacer que los dispositivos puedan ser esclavos o maestros.

#### 2.2.2.4 DoS

Los ataques DoS (*Denegation of Services*, Denegación de Servicios) consisten básicamente en mantener ocupados a los servicios ofrecidos por un dispositivo o una red, hasta conseguir que estos servicios no sean disponibles a los usuarios legítimos. Existen muchas formas de implementar un ataque DoS. Entre las más comunes en redes inalámbricas se encuentran los que se llevan a cabo cuando el agresor le realiza constantes solicitudes de respuesta a un dispositivo objetivo hasta inhabilitar sus servicios (*flooding*).

En redes 802.11, los APs son el principal objetivo de los agresores cuando después de haber sido inundado con envíos de solicitudes de respuesta, el AP podría bloquearse de tal forma que los usuarios legítimos no puedan ser capaces de establecer una conexión con él. En otras situaciones el agresor primero implementaría un ataque *man-in-the-middle,* haciéndose pasar por un AP, con lo que tendría la capacidad para no enviar los datos recibidos del dispositivo víctima a su destino.

En redes Bluetooth, este ataque es implementado por *flooding,* provocando que además de inhabilitar los servicios disponibles por los dispositivos víctimas como teléfonos y PDAs, sus baterías pueden llegar a degradarse.

#### 2.2.2.5 *Jamming*

La mayoría de las comunicaciones inalámbricas trabajan con frecuencias públicas, es decir, no existe alguna licencia para poder trabajar sobre estas y por lo tanto manejan los mismos rangos de frecuencias. Debido a esto, existen múltiples dispositivos electrónicos que pueden interferir en las comunicaciones de las redes inalámbricas, entre los más destacados se encuentran los teléfonos inalámbricos, los monitores para bebés y los hornos de microondas. Además, entre las mismas tecnologías de comunicación inalámbrica existen problemas de interferencias, como lo es la coexistencia de redes 802.11 y Bluetooth, que llegan a entorpecer sus transmisiones aunque las dos tecnologías manejen diferentes métodos de transmisión (DSSS y FHSS respectivamente).

Esta vulnerabilidad que presentan las redes inalámbricas, permite que los agresores implementen el ataque *jamming*, siendo este un particular tipo de ataque DoS, el cual irradia las transmisiones de una red o miembros de esta con el uso de hardware diseñado para emitir señales potentes, y así inutilizar los servicios con que cuentan. El agresor puede ejecutar diferentes técnicas de implementación de *jamming*. En muchos casos es utilizado como una herramienta para llevar a cabo otros ataques.

### 2.3 Detección de *sniffers*

La ventaja más importante de los *sniffers* inalámbricos es sin duda la indetectabilidad con que cuentan. Esta característica surge de la naturaleza propia del funcionamiento de las tarjetas de red que se encuentran configuradas con el modo de monitoreo. A diferencia del modo promiscuo, utilizado por los *sniffers* alámbricos, con el modo de monitoreo sólo se pueden recibir paquetes y no se tiene la capacidad para realizar transmisiones al mismo tiempo. En cambio con el modo promiscuo, los adaptadores de red alámbricos tienen la capacidad de responder a peticiones ARP y de esta forma podrían ser detectados por el administrador. En otras palabras, la detección de *sniffers* inalámbricos que usen adaptadores configurados con el modo de monitoreo puede ser análoga a la forma en que se detecta a una persona sintonizando una estación de radio, es decir, no existe.

En consecuencia, la confidencialidad de la información transmitida en las redes inalámbricas es inevitablemente vulnerable ante este particular tipo de ataque. Es por esto que en la siguiente sección se explicará por qué la encriptación de la información es la forma más conveniente de combatirlo.

## 2.4 Prevención de *sniffers*

La única forma de proteger la información manejada en una red inalámbrica de los ataques *sniffing* es el uso de mecanismos de encriptación. En el estándar 802.11, WEP es un mecanismo de encriptación el cuál, debido a la mala distribución de su clave compartida y a las debilidades del algoritmo RC4 que utiliza, puede ser fácilmente quebrantado por un agresor inexperto (Ver secciones 1.4.1.2.2). Por otra parte, el mecanismo de encriptación usado por Bluetooth también presenta conflictos en el manejo de las claves de enlace y su algoritmo $E_0$ que lo pueden convertir en un mecanismo inservible (Ver sección 1.4.2.3).

Como resultado de las vulnerabilidades que presentan 802.11 y Bluetooth v1.1, la IEEE y la SIG han tomado medidas correctivas para encarar estos inconvenientes. Por parte de la IEEE, el grupo de trabajo i se ha encargado de reforzar la seguridad de 802.11 con la creación del nuevo estándar 802.11i, pero este aún no ha podido ser completado y su implementación requiere del uso de dispositivos con nuevas capacidades. Debido a esto y por exigencias de seguridad por parte de los consumidores, la alianza Wi-Fi y la IEEE desarrollaron la especificación WPA (*Wi-Fi Protected Access*, Acceso a Wi-Fi Protegido) para incrementar el nivel de protección de los datos y el control de acceso para las WLANs existentes y futuras. Esta especificación provee de una encriptación de datos y una autenticación de usuarios mejoradas con el simple hecho de actualizar el firmware de los dispositivos Wi-Fi existentes. Por otra parte, el SIG ha reforzado el manejo de las claves de enlace con la versión 1.2 de Bluetooth. En esta versión no se permitirá el uso de claves de unidad como claves de enlace, aunque se introducirá el concepto de claves de grupo para incrementar la seguridad. Adicionalmente, se utilizará el protocolo asimétrico de encriptación Diffie-Hellmann para el establecimiento de las claves de combinación.

A pesar de estas modificaciones que se han desarrollado en 802.11 y Bluetooth, puede ser necesario incrementar la seguridad de estas tecnologías con el uso de aplicaciones o protocolos de encriptación que se desempeñen en capas de niveles superiores a la capa de enlace, siendo esta en la cual los mecanismos de encriptación de 802.11 y Bluetooth son aplicados. Algunos de los comúnmente implementados en ambas tecnologías se describen brevemente a continuación.

*SSL (Secure Sockets Layer)*

SSL es un protocolo desarrollado por *Netscape Communications* para proveer seguridad y privacidad sobre las comunicaciones de Internet, principalmente en los navegadores *web*. SSL es el protocolo más ampliamente usado en el comercio electrónico para realizar las transacciones. Este protocolo es implementado entre el protocolo TCP/IP y protocolos de la capa de aplicación como lo son HTTP (*Hypertext Transfer Protocol*, Protocolo de Transferencia de Hipertexto), FTP (*File Transfer Protocol*, Protocolo de Transferencia de Archivos) y Telnet. SSL proporciona confidencialidad, integridad, autenticación (tanto del servidor como del cliente) y no repudiación por usar algoritmos criptográficos. SSL soporta una variedad de algoritmos, entre los más utilizados están RSA, RC2, RC4, IDEA, DES, triple-DES y MD5.

*TLS (Transport Layer Security)*

Este protocolo fue introducido por la IETF (*Internet Engeneering Task Force*) para proveer privacidad e integridad de los datos entre dos aplicaciones comunicándose sobre la Internet. Está basado en la versión 3.0 del protocolo SSL aunque se le han hecho algunas adopciones. SSL/TLS ha sido la base de seguridad para otros protocolos como PCT (*Private Communications Technology*, Tecnología de Comunicaciones Privadas) de Microsoft, STLP (*Secure Transport Layer Protocol*, Protocolo Seguro de la Capa de Transporte) y WTLS (*Wireless Transport Layer Security*, Seguridad Inalámbrica de la Capa de Transporte) usado por el protocolo WAP para las comunicaciones de PDAs y teléfonos celulares (*Wireless Access Protocol*, Protocolo de Acceso Inalámbrico).

*SSH (Secure Shell)*

SSH es un protocolo que permite crear conexiones seguras entre dos computadoras que pertenezcan a una red. Este protocolo provee autenticación cliente y servidor, confidencialidad e integridad de los datos en servicios de FTP y Telnet principalmente. Al igual que TLS, SSH trabaja sobre conexiones TCP/IP en los cuales se negocian métodos de intercambio de claves, algoritmos de claves públicas, algoritmos simétricos de claves, algoritmos de autenticación y algoritmos hash. Entre los más usados se encuentran RSA, DES, y triple-DES. SSH es utilizado, además de ser implementado como un programa, sobre sistemas basados en Unix, sin embargo, ya existen algunas aplicaciones SSH basadas en Windows.

*IPSec (IP Security)*

Este protocolo fue desarrollado por la IETF para proveer de encriptación y autenticación en la capa de transporte y de esta forma proteger a los paquetes IP transmitidos entre dispositivos con soporte IPSec. Actualmente, IPSec es soportado por IPv4 y es obligatoriamente implementado en IPv6. IPSec es comúnmente encontrado en dispositivos de red como enrutadores, switches, firewalls y servidores de acceso remoto. Para brindar los servicios de autenticación, integridad y confidencialidad, IPSec hace uso de los protocolos de transmisión AH (*Authentication Header*, Encabezado de Autenticación) y ESP (*Encapsulating Security Payload*, Carga Encapsulada de Seguridad), además de algún protocolo para el manejo de claves.

# Protocolos de comunicación IEEE 802.11 y Bluetooth

## 3.1 Introducción

Antes de tratar con el desarrollo de la aplicación *sniffer,* es necesario conocer la estructura de los datos que son transmitidos en forma de paquetes sobre las comunicaciones inalámbricas, y que por lo tanto, son capturados durante la ejecución de esta aplicación. Para esto, se han consultado las especificaciones MAC (*Medium Access Control*, Control de Acceso al Medio) y PHY (Physical Layer, Capa Física) para LAN inalámbricas por parte del estándar 802.11, y de los protocolos HCI (*Host Control Interface*, Interfaz del Control del Anfitrión), L2CAP (*Logical Link Control Adaptation Protocol*, Protocolo de Adaptación y Control de Enlace Lógico), SDP (*Service Discovery Protocol*, Protocolo de Servicio de Descubrimiento) y RFCOMM (*Radio Frequency Comm*, Puerto Serial Inalámbrico) por parte de la tecnología Bluetooth [4] [9].

En estos momentos, las tecnologías 802.11 y Bluetooth han evolucionado tanto en el ámbito comercial como en el campo de la investigación. En cuanto a 802.11, se está migrando de la 802.11b[7] a la 802.11g[8] debido a las altas velocidades de transmisión que esta nueva tecnología conlleva. Por otra parte, sobre Bluetooth, la versión 1.1 está teniendo un gran impacto, principalmente sobre los dispositivos móviles como PDAs y celulares, aunque actualmente ya se están estableciendo nuevos protocolos, perfiles y servicios que se están incorporando a la versión 1.2.

La completa compatibilidad que existe entre los dispositivos con las nuevas especificaciones y los dispositivos actualmente en uso, no afectará la estructura de los datos transmitidos entre ellos.

## 3.2 Estructura de los datos sobre 802.11

Las especificaciones del estándar IEEE 802.11 con respecto al modelo OSI (*Open System Interconnection*, Interconexión de Sistemas Abiertos), están enfocadas a la PHY y a la subcapa MAC para las WLANs. La subcapa MAC establece un conjunto de reglas para determinar como accesar al medio y enviar los datos, pero los detalles de transmisión y recepción de estos pertenecen a la PHY (Ver figura 3.1).

La subcapa LLC (*Logical Link Control*, Control Lógico de Enlace) se encarga de encapsular la información del protocolo 802.11 para que pueda ser manejada por protocolos de nivel superior como el protocolo IP (*Internet Protocol*, Protocolo de Internet), sus especificaciones se encuentran establecidas en el estándar IEEE 802.2.

En 802.11 se cuentan con diferentes tipos de PHYs, las cuales han surgido de las mejoras que se le han estado implementando a esta tecnología. La información capturada por el *sniffer* está codificada de acuerdo a la subcapa MAC, es por esto que no importa que PHY se esté utilizando. A continuación se explicará la estructura de los paquetes que se manejan en la subcapa MAC.

[7] La tecnología 802.11b es también conocida como Wifi (*Wireless Fidelity*, Fidelidad Inalámbrica) y su velocidad de transmisión máxima es de 11Mbps

[8] La velocidad de transmisión máxima inicial de la tecnología 802.11g fue de 54Mbps y ha llegado a alcanzar hasta los 75Mbps actualmente

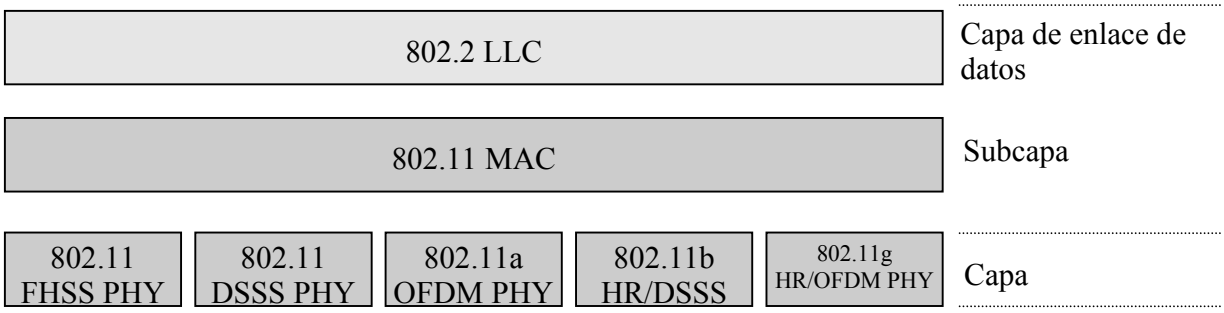


Figura 3.1 – El protocolo 802.11 y su relación con el modelo OSI

### 3.2.1 Formato de los paquetes MAC

La estructura de los paquetes MAC consiste principalmente de un encabezado MAC, los datos y un FCS (*Frame Check Sequence*, Secuencia de Chequeo del Paquete). En la figura 3.2 se muestra la estructura general de estos paquetes desde el byte menos significativo al más significativo.

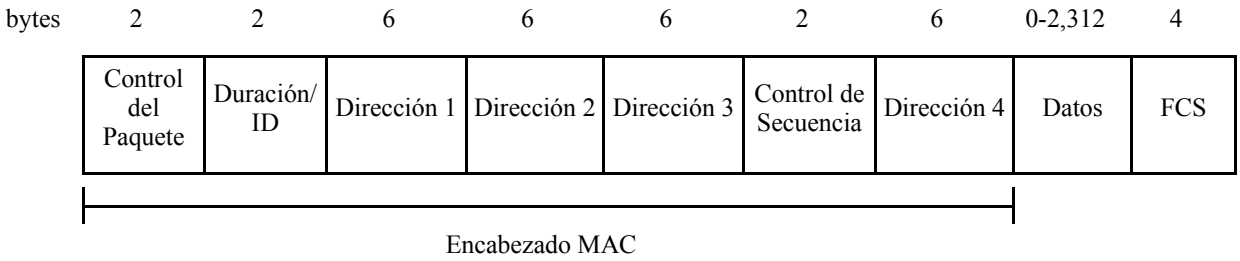


Figura 3.2 – Formato general de los paquetes MAC

En el encabezado MAC se encuentra suficiente información para el manejo de la fragmentación, el proceso de transmisión, el mecanismo de encriptamiento y los datos que transporta el paquete. La longitud de este encabezado, así como los datos, es variable y depende del tipo de paquete que se esté transmitiendo, es por esto que los campos Dirección 2, 3 y 4, y el Control de Secuencia, pueden estar o no presentes. En el FCS se almacena un CRC (*Cyclic Redundancy Code*, Código de Redundancia Cíclica) de 32 bits que permite verificar la integridad del paquete.

#### 3.2.1.1 Control del paquete

Este campo se compone de dos bytes en los cuales se integran los subcampos *Versión del Protocolo*, *Tipo* y *Subtipo*, *Para DS*, *Desde DS*, *Más Fragmentos*, *Retransmisión*, *Administración de Energía*, *Más Datos*, *WEP* y *Orden*. En la figura 3.3 se muestra la estructura de este campo.

El campo *Versión del Protocolo* especifica que versión del 802.11 MAC es contenida en este paquete. Actualmente, sólo se ha desarrollado una versión del 802.11 MAC, es por eso que el valor de este campo es de 0. Cuando los estándares IEEE cambien el MAC y promulguen que esta versión es incompatible a la inicial, entonces este campo tendrá otros valores.

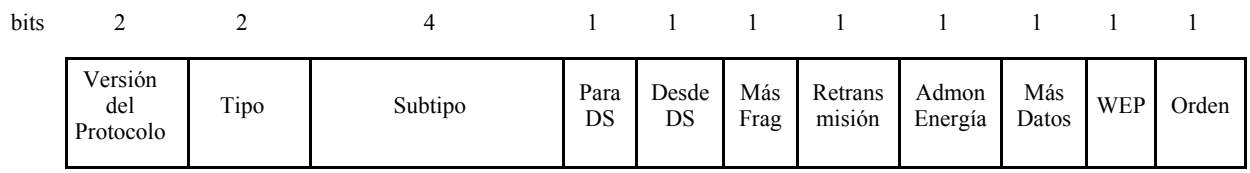


Figura 3.3 – Control del paquete

Los campos *Tipo* y *Subtipo* identifican el tipo de paquete usado. Como se mencionó anteriormente, los valores de estos campos dependen del tipo de los datos que se estén transmitiendo. En la tabla 3.1 se muestran los valores de estos identificadores.

Tabla 3.1 – Identificadores de los campos *Tipo* y *Subtipo*[9]

| **Valor del Subtipo** | **Nombre del Subtipo** |
|---|---|
| **Paquetes de Administración (Tipo=00)** | |
| 0000 | Petición de asociación |
| 0001 | Respuesta de asociación |
| 0010 | Petición de reasociación |
| 0011 | Respuesta de reasociación |
| 0100 | Petición de sondeo |
| 0101 | Respuesta de sondeo |
| 1000 | Beacon |
| 1001 | ATIM (*Announcement Traffic Indication Message*, Mensaje de indicación de anuncio de tráfico) |
| 1010 | Deasociación |
| 1011 | Autenticación |
| 1100 | Deautenticación |
| **Paquetes de Control (Tipo=01)** | |
| 1010 | Power Save (PS)-Poll |
| 1011 | RTS |
| 1100 | CTS |
| 1101 | Reconocimiento (ACK) |
| 1110 | Contention-Free (CF)- End |
| 1111 | CF-End+CF-Ack |

[9]Los valores de los identificadores para ambos campos están escritos desde el bit más significativo al menos significativo.

| Valor del Subtipo | Nombre del Subtipo |
|---|---|
| **Paquetes de Datos (Tipo=10)** | |
| 0000 | Datos |
| 0001 | Datos+CF+Ack |
| 0010 | Datos+CF-Poll |
| 0011 | Datos+CF-Ack+CF-Poll |
| 0100 | Datos nulos (no existen datos transmitidos) |
| 0101 | CF-Ack (no existen datos transmitidos) |
| 0110 | CF-Poll (no existen datos transmitidos) |
| 0111 | Datos+CF-Ack+CF-Poll |
| **Tipo del paquete igual a 11 es reservado** | |

Con respecto a los campos *Para DS* y *Desde DS*, sus valores indican si los paquetes van dirigidos o no a un DS (*Distribution System*, Sistema de Distribución)[10]. Además, ayudan a determinar que campos de direcciones deben aparecer en los paquetes de datos (Ver sección 3.2.1.3). En la tabla 3.2 se muestra la interpretación de los valores de estos campos.

Tabla 3.2 – Interpretaciones de los campos Para DS y Desde DS

| | **Para DS = 0** | **Para DS = 1** |
|---|---|---|
| **Desde DS = 0** | Todos los paquetes de Administración y Control, y los paquetes de Datos dentro de una red Ad-hoc | Paquetes de Datos entrando al DS |
| **Desde DS = 1** | Paquetes de Datos saliendo desde el DS | Paquetes de datos distribuidos desde un AP a otro en diferentes DSs (WDS[11]) (*Bridge*) |

Para el manejo de la fragmentación de un paquete se hace uso del campo *Más Fragmentos*, que cuando es igual a 1 indica si existen fragmentos subsecuentes de algún paquete de datos o de administración. *Más fragmentos* es igual a 0 en cualquier otro caso.

Con el campo *Retransmisión*, cuando su valor es igual a 1 indica que el paquete ya fue transmitido. Esto le ayuda al dispositivo destino a no procesar paquetes duplicados.

Ya que 802.11 está orientado a dispositivos móviles como PDA's y Laptops, el campo *Administración de Energía* brinda soporte para el consumo de energía sobre estos dispositivos. Cuando este campo es igual a 1, indica que el dispositivo transmisor se colocará en modo de ahorro de energía después del envío de paquetes. En los APs este campo siempre es 0 debido a que estos dispositivos no cuentan con soporte para ahorro de energía. Sin embargo, tienen la capacidad de almacenar los

[10]Un sistema de distribución es un componente lógico del 802.11 que es usado para enviar los paquetes a sus destinos correspondientes. En la implementación actual de las WLANs, las redes Ethernet son utilizadas con este fin.
[11]*Wireless Distribution System*, Sistema de Distribución Inalámbrico

paquetes que vayan dirigidos a los dispositivos móviles que se encuentren "durmiendo". Por otro lado, con el uso del campo *Más Datos*, los APs les indican a los dispositivos dormidos que disponen de paquetes dirigidos a ellos *(Más Datos* igual a 1) y para la recuperación de estos, los dispositivos dormidos deben enviar paquetes de tipo PS-Poll a sus APs correspondientes.

Algunas veces es importante el orden en que los paquetes deben ser transmitidos debido al costo de procesamiento adicional que se pueda llevar en la recepción y transmisión de paquetes MAC. De aquí parte la función del campo *Orden* ya que cuando se requiere de un servicio que requiera de un orden estricto de entrega de paquetes, el valor del campo es 1.

Otro campo de suma importancia que controla la confidencialidad y la autenticación de los datos es *WEP*. Cuando este campo se coloca en 1, indica que el mecanismo de encriptación WEP ha sido aplicado a los datos del paquete y por lo tanto la estructura del paquete cambia ligeramente. En la figura 3.4 se muestra el paquete modificado después de pasar a través del mecanismo de encriptación WEP.

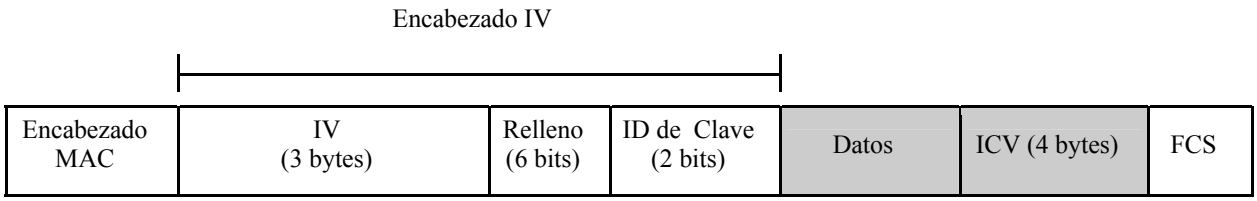


Figura 3.4 – Extensiones WEP del paquete

Los campos *Datos* e *ICV* (*Integrity Check Value*, Valor para el Chequeo de Integridad) están protegidos por WEP, en donde, *Datos* son los datos del paquete e *ICV* es el valor de verificación de la integridad de estos datos, ambos encriptados con el uso de una clave compartida y un *IV* (*Initialization Vector*, Vector de Inicialización). El campo *ID de Clave* identifica la clave default que fue utilizada para encriptar el paquete; si es usado un mapeo de claves entonces su valor es 0.

### 3.2.1.2 Duración/ID

Este campo de 16 bits puede ser usado en una de las tres formas que se muestran en la figura 3.5. Para el control de acceso al medio, los dispositivos inalámbricos deben monitorear los encabezados de todos los paquetes que reciben y por consiguiente deben actualizar el NAV (*Network Allocation Vector*, Vector de Asignación de Red)[12]. De aquí que cuando el bit 15 del campo *Duración/ID* es 0, el valor de este campo indica la cantidad de tiempo (microsegundos, μseg) con que se debe actualizar el NAV (Ver figura 3.5a). De la misma forma, pero para paquetes transmitidos durante CFP (*Contention Free Periods*, Periodos de Contención Libre), el valor de este campo es interpretado como un NAV constante de 32,768 μseg (Ver figura 3.5b).

Por otra parte, el campo *Duración/ID* tiene una interpretación diferente en los paquetes PS-Poll, debido a que este tipo de paquete es utilizado por los dispositivos móviles para recuperar sus paquetes almacenados por los APs mientras se encontraban en modo de ahorro de energía, del campo *Duración/ID* se obtiene el AID (*Association ID*, Identificador de Asociación) que indica el BSS (*Basic Service Set*, Conjunto Básico de Servicios) al cual los dispositivos móviles pertenecen (Ver figura 3.5c).

[12]El NAV es un cronómetro que indica la cantidad de tiempo que el medio de transmisión estará reservado para algún dispositivo inalámbrico. Cuando el NAV es 0 significa que el medio se encuentra disponible.

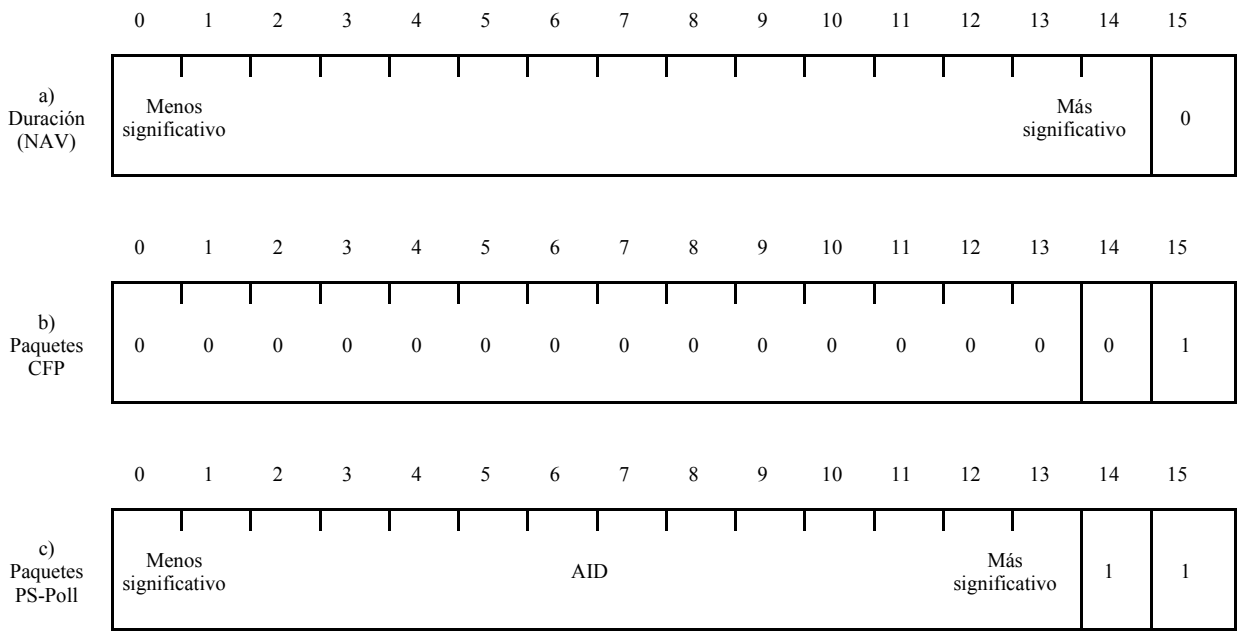


Figura 3.5 – El campo Duración/ID

### 3.2.1.3 Direcciones

En el encabezado MAC pueden aparecer hasta cuatro campos de direcciones, las cuales son usadas para diferentes propósitos dependiendo sobre el tipo del paquete. Las direcciones manejadas están compuestas de 48 bits de acuerdo al estándar IEEE 802 y fueron diseñadas para identificar a un dispositivo, un grupo de dispositivos o a todos los dispositivos que integran una red (*unicast*, *multicast* o *broadcast* respectivamente). Las direcciones son utilizadas para diferentes finalidades, como las que se presentan a continuación.

- DA (*Destination Address*, Dirección destino). Es un identificador IEEE MAC de 48 bits que corresponde al dispositivo receptor final, el cual manejará el paquete para el procesamiento de protocolos de las capas más altas.
- SA (*Destination Address*, Dirección fuente). Es un identificador IEEE MAC de 48 bits que identifica al dispositivo que ha creado el paquete.
- RA (*Receiver Address*, Dirección del receptor). Es una dirección IEEE MAC de 48 bits que indica que dispositivo inalámbrico procesará el paquete. Si se trata de una estación inalámbrica, entonces RA es el DA. En caso de que el paquete vaya dirigido a un dispositivo conectado en una red Ethernet a través de un AP, RA contendrá la dirección MAC del AP.
- TA (*Transmitter Address*, Dirección del transmisor). Es una dirección IEEE MAC de 48 bits que identifica el dispositivo inalámbrico que transmite el paquete dentro del medio inalámbrico.
- BSSID (*Basic Service Set ID*, Identificador del Conjunto Básico de Servicios). Es una dirección MAC que identifica la red inalámbrica a la cual un dispositivo ha sido asignado. En redes *Ad-hoc* se genera un BSSID aleatorio de acuerdo a las cláusulas establecidas en IEEE 802, para evitar conflictos con direcciones MAC legales. En cambio, para las redes con infraestructura, el BSSID es la dirección MAC perteneciente al AP.

Ya que el uso de los campos *Dirección* dependen sobre el tipo y subtipo del paquete, es común que en la mayoría de los paquetes se utilicen tres campos para el DA, SA y BSSID. Sin embargo, para los paquetes de tipo datos, el uso de los campos *Dirección* dependen de la forma en que la red se encuentre

diseñada. Es por esto que los valores de los campos *Para DS* y *Desde DS* son de suma importancia para estos casos (Ver tabla 3.3).

Tabla 3.3 – Uso de los campos *Dirección* en los paquetes de datos

| Función | Para DS | Desde DS | Dirección 1 (receptor) | Dirección 2 (transmisor) | Dirección 3 | Dirección 4 |
|---|---|---|---|---|---|---|
| IBSS | 0 | 0 | DA | SA | BSSID | No usado |
| Para AP (infraestructura) | 1 | 0 | BSSID | SA | DA | No usado |
| Desde AP (infraestructura) | 0 | 1 | DA | BSSID | SA | No usado |
| WDS (puenteo) | 1 | 1 | RA | TA | DA | SA |

### 3.2.1.4 Control de Secuencia

Este campo de 16 bits es utilizado en el proceso de desfragmentación y ayuda a eliminar la duplicación de paquetes. Para esto, se hacen uso de sus subcampos *Número de Fragmento* y *Número de Secuencia* (Ver Figura 3.6).

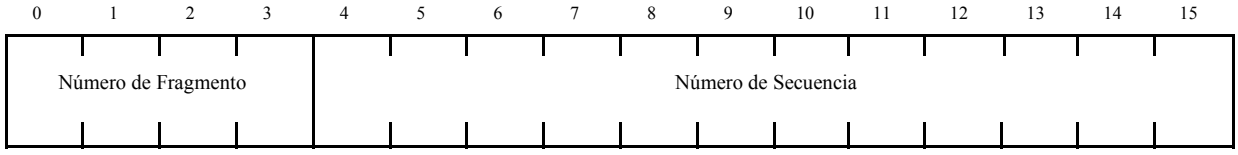


Figura 3.6 – El campo de *Control de Secuencia*

Con el subcampo *Número de Fragmento* se tiene control en el reensamblaje de los paquetes, colocando un identificador numérico de 4 bits, el cual, adquiere un valor de cero para el primer fragmento y un valor incrementado en uno para los fragmentos sucesivos. Este identificador permanece constante en todas las retransmisiones de los fragmentos.

En el manejo de paquetes de niveles más altos por la subcapa MAC, a cada uno de estos paquetes les es asignado un número secuencial. Este valor es almacenado en el subcampo *Número de Secuencia* (12 bits). La asignación de estos valores se basan de acuerdo al funcionamiento de un contador de módulo 4096, en donde al primer paquete se le asigna un *Número de Secuencia* de 0 y para los paquetes subsecuentes el valor de este subcampo se incrementa en uno. En las retransmisiones de estos paquetes este valor no cambia, de la misma forma que ocurre con sus fragmentos en el caso de que los paquetes hayan sido fragmentados.

### 3.2.1.5 Datos

En este campo se encuentra la información específica de los diferentes tipos de paquetes que pertenecen a protocolos de capas superiores como IP. La longitud de este campo es variable y depende del tipo de paquete. Su capacidad máxima es de 2,304 bytes, incluyendo la información agregada por WEP (Encabezado IV y el campo ICV), y su capacidad mínima es de 0 bytes.

### 3.2.1.6 FCS

Este campo contiene un CRC (*Cyclic Redundance Code*, Código de Redundancia Cíclica) de 32 bits con el cual se puede verificar la integridad del paquete. Para el cálculo del CRC, se hace uso del encabezado MAC y del campo *Datos*, sin embargo, es importante mencionar que bajo redes en donde se implemente 802.3 además del 802.11, el FCS debe ser recalculado debido a que los encabezados MAC son diferentes en ambas tecnologías, aunque estas utilicen el mismo método de obtención del CRC.

Con el valor del FCS del paquete transmitido, los dispositivos receptores pueden verificar si el paquete fue alterado durante su transmisión comparándolo con el CRC que estos han calculado. En caso de que su CRC coincida con el del paquete, se debe de enviar un paquete de reconocimiento positivo al dispositivo transmisor. En caso contrario, el tiempo de espera terminará y por lo tanto el dispositivo transmisor deberá retransmitir el paquete. En 802.11 no existen los reconocimientos negativos.

## 3.2.2 Análisis de los paquetes 802.11

Para lograr una mayor comprensión sobre la estructura del protocolo 802.11, a continuación se llevará a cabo el análisis detallado del contenido del paquete que se muestra en la figura 3.7. El paquete fue capturado con la aplicación *sniffer* que se ha desarrollado en este trabajo. La información relacionada con la aplicación se expondrá en el siguiente capítulo.

| No byte (Hexadecimal) | Contenido del paquete en bytes (Hexadecimal) | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | 08 | 02 | 00 | 00 | ff | ff | ff | ff | ff | ff | 00 | 11 | 09 | 23 | 04 | f3 |
| 0010 | 00 | 07 | E9 | d3 | 89 | 77 | 40 | f3 | aa | aa | 03 | 00 | 00 | 00 | 08 | 00 |
| 0020 | 45 | 00 | 00 | 4e | 46 | 1b | 00 | 00 | 80 | 11 | 72 | 1a | C0 | a8 | 00 | 1a |
| 0030 | c0 | a8 | 00 | ff | 00 | 89 | 00 | 89 | 00 | 3a | 98 | 2e | 80 | 57 | 01 | 10 |
| 0040 | 00 | 01 | 00 | 00 | 00 | 00 | 00 | 00 | 20 | 46 | 44 | 45 | 46 | 46 | 43 | 46 |
| 0050 | 47 | 45 | 4a | 45 | 45 | 45 | 50 | 46 | 43 | 43 | 41 | 43 | 41 | 43 | 41 | 43 |
| 0060 | 41 | 43 | 41 | 43 | 41 | 43 | 41 | 41 | 41 | 00 | 00 | 20 | 00 | 01 | | |

Figura 3.7 – Contenido de un paquete capturado en una red 802.11

Para llevar a cabo el análisis, el paquete será seccionado en: encabezado 802.11 MAC, encabezado 802.11 LLC, encabezados siguientes y datos.

**Encabezado 802.11 MAC**

El encabezado 802.11 puede estar conformado por un número variable de bytes debido a la cantidad de direcciones MAC que maneja (hasta 4 direcciones MAC), en este caso el encabezado 802.11 está conformado por 24 bytes (sección cerrada con línea gruesa en la figura 3.7) ya que cuenta con 3 direcciones MAC. Para el análisis de este encabezado se explicará el significado de cada byte que lo conforman.

*Los bytes 0 y 1*

Los primeros 2 bytes del encabezado forman al campo *Control del Paquete* tomándolos de derecha a izquierda (0x0208). En este campo se incluye información necesaria para el adecuado procesamiento del paquete. En este caso, en el byte 0 (0x08), los primeros 2 bits indican que se está utilizando la versión 0 del protocolo 802.11 y los siguientes 6 bits indican que el paquete transporta datos de protocolos de niveles superiores (Ver tabla 3.1). En el byte 1 (0x02), los primeros 2 bits indican que el paquete está saliendo de un DS (*Distribution System*, Sistema de Distribución) y los siguientes bits indican, respectivamente, que el paquete: puede no ser un fragmento o puede ser el último fragmento, no ha sido retransmitido, la estación que lo transmitió está activa, la estación que lo transmitió no tiene paquetes almacenados, no se le ha aplicado el mecanismo de encriptación WEP y no debe ser procesado en un orden estricto.

*Los bytes 2 y 3*

Estos bytes forman al campo *Duration/ID* tomándolos de derecha a izquierda (0x0000). Este campo determina que el medio permanecerá ocupado 0 μseg en la transmisión de este paquete.

*Los bytes del 4 al 9*

Estos bytes representan la dirección MAC destino del paquete y se toman de izquierda a derecha (ff:ff:ff:ff:ff:ff). Por su valor el paquete va dirigido a todas las estaciones de la red (Broadcast).

*Los bytes del 10 al 15*

Estos bytes representan el BSSID de la red al que va dirigido el paquete y se toman de izquierda a derecha (00:11:09:23:04:f3). Como se puede apreciar el BSSID es una dirección MAC.

*Los bytes del 16 al 21*

Estos bytes representan la dirección MAC origen del paquete y se toman de izquierda a derecha (00:07:e9:d3:89:77).

*Los bytes 22 y 23*

Estos bytes forman al campo de *Control de Secuencia* tomándolos de derecha a izquierda (0xf340). Este campo es usado en el proceso de desfragmentación y para descartar la duplicación de paquetes. Los primeros 4 bits del campo (subcampo Número de Fragmento) indican que el paquete es el fragmento número 0, por lo que se puede concluir con la ayuda del campo *Control del Paquete* (Ver *los bytes 0 y 1*), que la información del paquete no ha sido fragmentada. Los 12 bits restantes del campo *Control de Secuencia* (subcampo *Número de Secuencia*) establecen que el paquete tiene un número de secuencia igual a 3892, lo que significa que los paquetes subsecuentes tendrán un número de secuencia mayor (3893, 3894, etc).

**Encabezado 802.2 LLC**

De acuerdo al análisis del encabezado anterior se puede argumentar que el paquete contiene datos de protocolos de capas superiores a 802.11 (Ver *los bytes 0 y 1*). En este caso, por la estructura de los datos se deduce que el siguiente encabezado pertenece al protocolo 802.2 LLC. Este encabezado es usado por 802.11 para transportar protocolos de la capa de red. Para conseguirlo, 802.2 hace uso del método SNAP (*Sub-network Access Protocol*, Protocolo de Acceso a la Subred), insertando un encabezado SNAP, en el cual se almacenan los parámetros para el encapsulado de los datos pertenecientes a la subcapa LLC. El encabezado SNAP se compone de los campos *DSAP* (*Destination Service Access Point*, Punto de Acceso del Servicio Destino), *SSAP* (*Source Service Access Point*, Punto de Acceso del Servicio Origen), *Control*, *Código Organizacional* y *Tipo*. En la figura 3.7, los bytes del 24 al 31 representan al encabezado 802.2 LLC (sección cerrada con línea punteada).

*Los bytes 24 y 25*

Estos bytes forman a los campos *DSAP* y *SSAP* respectivamente. El valor 0xaa es siempre usado para ambos campos.

*El byte 26*

Este byte forma al campo Control. Este campo, con el valor de 0x03, denota a los datos siguientes al encabezado SNAP como información no enumerada. La información no enumerada indica que las transmisiones no son orientadas a conexión y que los datos no necesitan ser secuenciados o reconocidos.

*Los bytes del 27 al 29*

El campo *Código Organizacional* está formado por estos bytes y es usado para determinar como serán interpretados los bytes siguientes al encabezado SNAP. La información IP es encapsulada por 802.2 LLC utilizando el método marcado en RFC 1042, el cuál especifica el uso del OUI (*Organizationally Unique Identifier*, Identificador Organizacionalmente Único) 0x00-00-00. En este caso el campo *Código Organizacional* contiene este OUI[13].

*Los bytes 30 y 31*

Estos bytes forman al campo *Tipo*, el cual es usado para indicar el protocolo de red que se está transportando en el paquete. Este campo es el mismo que los paquetes Ethernet incluyen en sus encabezados. En este caso, el campo *Tipo* indica que el protocolo IP (0x0800) está incluido en el paquete. En redes IP, este campo puede adquirir también el valor de 0x0806 para especificar el uso del protocolo ARP.

**Encabezados siguientes**

Estos encabezados constan del encabezado IP (sección sombreada) seguido por el encabezado UDP (sección cerrada con línea delgada) como se muestra en la figura 3.7. El análisis de estos encabezados puede hacerse de forma análoga a los encabezados anteriores, aunque en este trabajo no hayan sido considerados. Sólo por brindar información diremos que los últimos 8 bytes del encabezado IP (c0 a8 00 1a c0 a8 00 ff) representan las direcciones IP origen (192.168.0.26) y destino (192.168.0.255) respectivamente, y que los primeros 4 bytes del encabezado UDP (00 89 00 89) representan los puertos origen (137) y destino (137) que en este caso se trata de los puertos netbios_ns.

**Datos**

Después del encabezado UDP vienen los datos del paquete (sección libre de la figura 3.7). Estos constan de 50 de los 1,500 bytes (tomando en cuenta los encabezados 802.2 LLC, IP y UDP) que pueden ser transmitidos por el protocolo 802.11 sobre redes IP (RFC 1191) y en los cuales normalmente se incluye información de protocolos o aplicaciones de capas superiores (ej. FTP, HTTP, TELNET, etc.).

El análisis anterior del paquete que se muestra en la figura 3.27 se resume en la tabla 3.4.

[13]Algunos distribuidores de dispositivos inalámbricos pueden usar un OUI asignado para sus propias comunicaciones. La lista de los OUIs asignados pueden ser encontrados en http://standards.ieee.org/regauth/oui/oui.txt

Tabla 3.4 – Análisis del paquete de la figura 3.7

| No. de Byte | Campo (s) | Valor (es) | Significado |
|---|---|---|---|
| **Encabezado 802.11** | | | |
| 0 y 1 | Control del Paquete | 0x0208 | Paquete de datos, está saliendo de un DS, no ha sido fragmentado ni retransmitido y no está encriptado entre otras cosas. |
| 2 y 3 | Duration/ID | 0x0000 | NAV igual a 0 μseg |
| 4-9 | Dirección MAC destino | 0xffffffffffff | ff:ff:ff:ff:ff:ff |
| 10-15 | BSSID | 0x0011092304f3 | 00:11:09:23:04:f3 |
| 16-21 | Dirección MAC origen | 0x0007e9d38977 | 00:07:e9:d3:89:77 |
| 22 y 23 | Control de Secuencia | 0xf340 | Número de secuencia igual a 3892 |
| **Encabezado 802.2 LLC** | | | |
| 24 | DSAP | 0xaa | |
| 25 | SSAP | 0xaa | |
| 26 | Control | 0x03 | Información IP no enumerada |
| 27-29 | Código Organizacional | 0x000000 | Encapsulamiento por RFC 1042 |
| 30 y 31 | Tipo | 0x0800 | Paquete IP |

## 3.3 Estructura de los datos sobre Bluetooth

Las especificaciones del protocolo de comunicación Bluetooth lo conforman diferentes protocolos, los cuales desempeñan actividades correspondientes a las capas del modelo OSI. A diferencia del protocolo 802.11, las especificaciones de Bluetooth comprenden desde la capa física hasta la capa de aplicación (Ver figura 3.8).

Entre los protocolos comprendidos por la aplicación *sniffer* implementada se encuentran HCI, L2CAP, RFCOMM y SDP. Los protocolos inferiores no se tomaron en cuenta debido a la fuerte dependencia que tienen con el hardware utilizado. Estos puntos serán vistos más a fondo en el siguiente capítulo.

### 3.3.1 Formato de los paquetes HCI

El HCI provee de un método uniforme, a través de una interfaz de comandos, para el *Controlador de bandabase* y el *Administrador de enlace*, además proporciona acceso al estado del hardware y los registros de control. Estos comandos son utilizados por el *Host* (anfitrión) y son enviados hacia el *Controlador del Host* por medio de un canal físico, como lo son los dispositivos USB y las PC Cards. En respuesta a las peticiones del *Host*, el *Controlador del Host* le envía información relacionada con los comandos en forma de eventos. No obstante, los eventos son también usados para realizar notificaciones al *Host* cuando algo ocurre (Ver figura 3.9).

Para HCI existen diferentes formatos de paquetes, esto depende del tipo de información que transporten. De aquí que los paquetes pueden ser de comandos, eventos o datos. Para los paquetes de datos, y de acuerdo al tipo de enlace que existe entre los dispositivos maestros y esclavos en una *piconet*, se cuentan con los paquetes SCO (*Synchronous Connection-Oriented*, Sincronización Orientada a Conexión) y ACL (*Asynchronous Connection-Less*, Asincronización Sin Conexión). En las siguientes secciones se explicarán las estructuras de cada uno de estos paquetes.

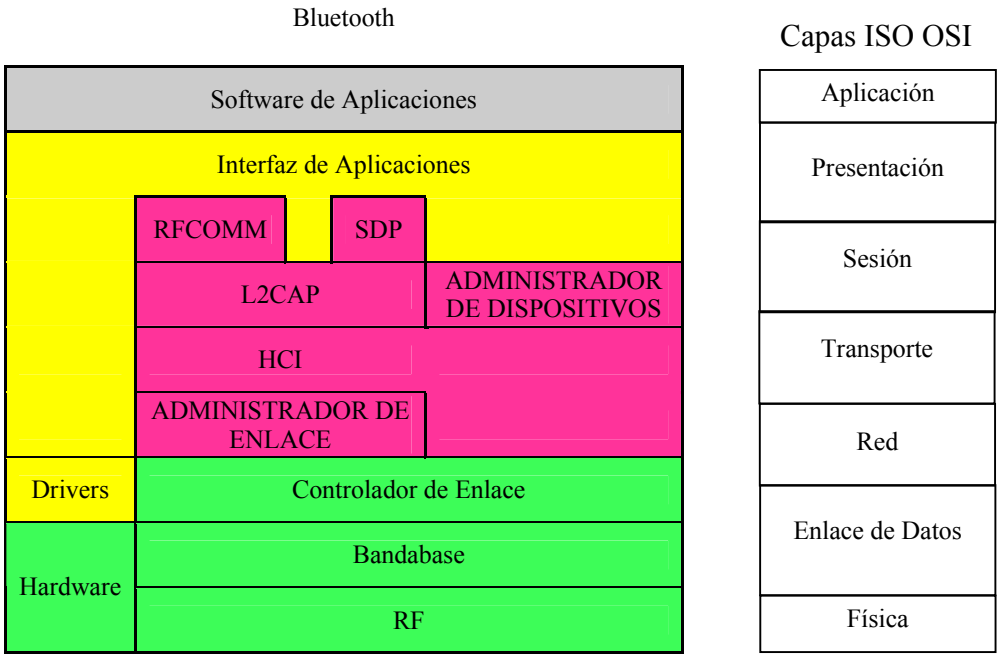


Figura 3.8 – Correspondencias entre Bluetooth y el modelo OSI

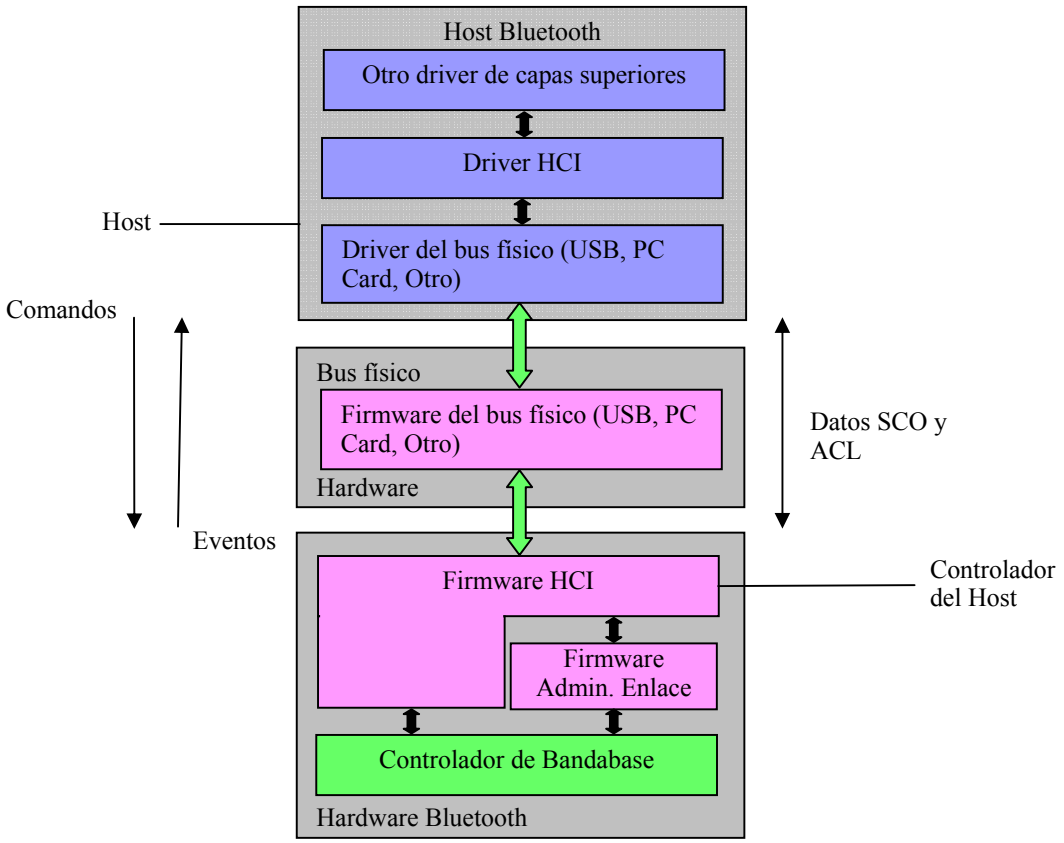


Figura 3.9 – Capas inferiores del software del protocolo Bluetooth y el intercambio de información entre el Host y el Host Controller

### 3.3.1.1 Paquete de comandos

En la estructura de los comandos HCI, cada comando cuenta con un determinado número de parámetros dependiendo de su función. Es por esto que la longitud del paquete es variable. En la figura 3.10 se muestra la estructura de los comandos HCI en unidades de bits.

En los paquetes, a cada comando se le asigna un campo de 2 bytes llamado *Opcode*, con el cual se identifican los diferentes tipos de comandos. Este campo se divide en dos subcampos llamados *OGF* (*OpCode Group Field*, Campo OpCode del Grupo) y *OCF* (*OpCode Command Field*, Campo OpCode del Comando), en donde *OGF* ocupa los 6 bits más significativos de *OpCode* y su función consiste en identificar el grupo al que pertenece el comando, mientras que el *OCF* ocupa los 10 bits restantes del *OpCode* e identifica el tipo de comando.

La *Longitud Total de los Parámetros* es un campo con un byte de longitud, que indica la cantidad de bytes que utilizan todos los parámetros contenidos en el paquete. En cambio, los campos *Parámetro 0* hasta el *N*, representan el conjunto de parámetros específicos del comando, en donde cada parámetro tiene una longitud entera de bytes. Los diferentes tipos de comandos con sus parámetros correspondientes se presentan en la tabla 3.5.

En la tabla también se incluye la información solicitada por algunos comandos en forma de parámetros. Estos parámetros son devueltos al *Host* a través del evento *Command Complete* (Comando Completo) cuando el comando ha sido procesado correctamente por el *Controlador del Host*. En caso de que ocurriera un error en la ejecución del comando, el evento *Command Status* (Estado del Comando) indicará el código del error correspondiente[14].

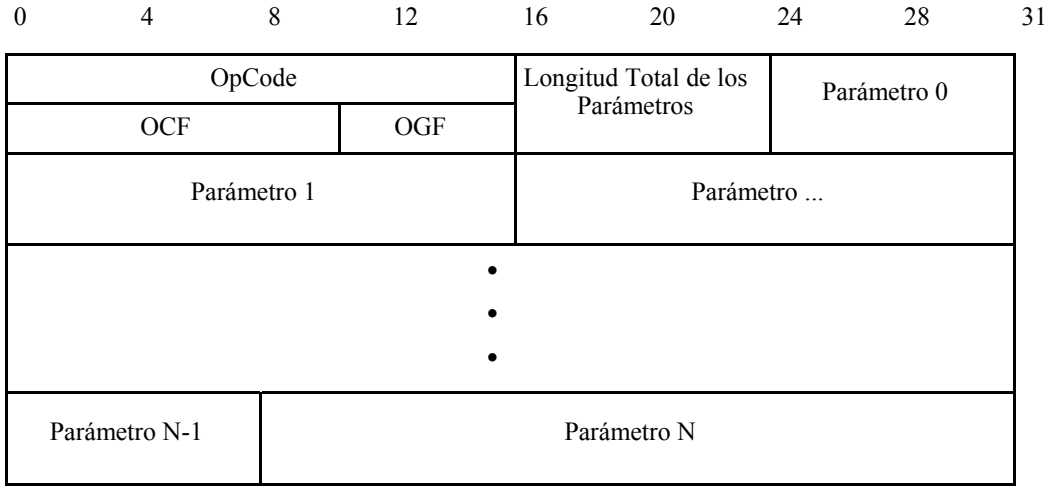


Figura 3.10 – Paquete de los comandos HCI

[14]La lista de los códigos de error se encuentran en el capítulo 6 de la Parte H:1 en [2].

Tabla 3.5 – Comandos HCI

| Comando | OCF | Parámetros | Parámetros devueltos |
|---|---|---|---|
| **Comandos de Control del Enlace (OGF=0x01)** | | | |
| Inquiry | 0x0001 | LAP, Inquiry_Length, Num_Responses | |
| Inquiry_Cancel | 0x0002 | | Status |
| Periodic_Inquiry_Mode | 0x0003 | Max_Period_Length, Min_Period_Length, LAP, Inquiry_Length, Num_Responses | Status |
| Exit_Inquiry_Periodic_Mode | 0x0004 | | Status |
| Create_Connection | 0x0005 | BD_ADDR, Packet_Type, Page_Scan_Repetition_ Mode, Page_Scan_Mode, Clock_Offset, Allow_Role_Switch | |
| Disconnect | 0x0006 | Connection_Handle, Reason | |
| Add_SCO_Connection | 0x0007 | Connection_Handle, Packet_Type | |
| Accept_Connection_Request | 0x0009 | BD_ADDR, Role | |
| Reject_Connection_Request | 0x000A | BD_ADDR, Reason | |
| Link_Key_Request_Reply | 0x000B | BD_ADDR, Link_Key | Status, BD_ADDR |
| Link_Key_Request_Negative_Reply | 0x000C | BD_ADDR | Status, BD_ADDR |
| PIN_Code_Request_Reply | 0x000D | BD_ADDR, PIN_Code_Length, PIN_Code | Status, BD_ADDR |
| PIN_Code_Request_Negative_Reply | 0x000E | BD_ADDR | Status, BD_ADDR |
| Change_Connection_Packet_Type | 0x000F | Connection_Handle, Packet_Type | |
| Authentication_Requested | 0x0011 | Connection_Handle | |
| Set_Connection_Encryption | 0x0013 | Connection_Handle, Encryption_Enable | |
| Change_Connection_Link_Key | 0x0015 | Connection_Handle | |
| Master_Link_Key | 0x0017 | Key_Flag | |

| Comando | OCF | Parámetros | Parámetros devueltos |
|---|---|---|---|
| Remote_Name_Request | 0x0019 | BD_ADDR, Page_Scan_Repetition_ Mode, Page_Scan_Mode, Clock_Offset | |
| Read_Remote_Supported_Features | 0x001B | Connection_Handle | |
| Read_Remote_Version_Information | 0x001D | Connection_Handle | |
| Read_Clock_Offset | 0x001F | Connection_Handle | |
| **Comandos de las Políticas del Enlace (OGF=0x02)** | | | |
| Hold_Mode | 0x0001 | Connection_Handle, Hold_Mode_Max_Interv al, Hold_Mode_Min_Interv al | |
| Sniff_Mode | 0x0003 | Connection_Handle, Sniff_Max_Interval, Sniff_Min_Interval, Sniff_Attempt, Sniff_Timeout | |
| Exit_Sniff_Mode | 0x0004 | Connection_Handle | |
| Park_Mode | 0x0005 | Connection_Handle, Beacon_Max_Interval, Beacon_Min_Interval | |
| Exit_Park_Mode | 0x0006 | Connection_Handle | |
| QoS_Setup | 0x0007 | Connection_Handle, Flags, Service_Type, Token_Rate, Peak_Bandwidth, Latency, Delay_Variation | |
| Role_Discovery | 0x0009 | Connection_Handle | Status, Connection_Handle, Current_Role |
| Switch_Role | 0x000B | BD_ADDR, Role | |
| Read_Link_Policy_ Settings | 0x000C | Connection_Handle | Status, Connection_Handle, Link_Policy_Settings |
| Write_Link_Policy_ Settings | 0x000D | Connection_Handle, Link_Policy_Settings | Status, Connection_Handle |
| **Comandos del Controlador del Host y Bandabase (OGF=0x03)** | | | |
| Set_Event_Mask | 0x0001 | Event_Mask | Status |
| Reset | 0x0003 | | Status |

| Comando | OCF | Parámetros | Parámetros devueltos |
|---|---|---|---|
| Set_Event_Filter | 0x0005 | Filter_Type, Filter_Condition_Type, Condition | Status |
| Flush | 0x0008 | Connection_Handle | Status, Connection_Handle |
| Read_PIN_Type | 0x0009 | | Status, PIN_Type |
| Write_PIN_Type | 0x000A | PIN_Type | Status |
| Create_New_Unit_Key | 0x000B | | Status |
| Read_Stored_ Link_Key | 0x000D | BD_ADDR, Read_All_Flag | Status, Max_Num_Keys, Num_Keys_Read |
| Write_Stored_Link_Key | 0x0011 | Num_Keys_To_Write, BD_ADDR[i], Link_Key[i] | Status, Num_Keys_Written |
| Delete_Stored_ Link_Key | 0x0012 | BD_ADDR, Delete_All_Flag | Status, Num_Keys_Deleted |
| Change_Local_Name | 0x0013 | Name | Status |
| Read_Local_Name | 0x0014 | | Status, Name |
| Read_Connection_ Accept_Timeout | 0x0015 | | Status, Conn_Accept_Timeout |
| Write_Connection_ Accept_Timeout | 0x0016 | Conn_Accept_Timeout | Status |
| Read_Page_Timeout | 0x0017 | | Status, Page_Timeout |
| Write_Page_Timeout | 0x0018 | Page_Timeout | Status |
| Read_Scan_Enable | 0x0019 | | Status, Scan_Enable |
| Write_Scan_Enable | 0x001A | Scan_Enable | Status |
| Read_Page_Scan_ Activity | 0x001B | | Status, Page_Scan_Interval, Page_Scan_Window |
| Write_Page_Scan_ Activity | 0x001C | Page_Scan_Interval, Page_Scan_Window | Status |
| Read_ Inquiry_Scan_Activity | 0x001D | | Status, Inquiry_Scan_Interval, Inquiry_Scan_Window |
| Write_Inquiry_ Scan_Activity | 0x001E | Inquiry_Scan_Interval, Inquiry_Scan_Window | Status |
| Read_ Authentication_Enable | 0x001F | | Status, Authentication_Enable |
| Write_ Authentication_Enable | 0x0020 | Authentication_Enable | Status |
| Read_Encryption_Mode | 0x0021 | | Status, Encryption_Mode |
| Write_Encryption_Mode | 0x0022 | Encryption_Mode | Status |

| Comando | OCF | Parámetros | Parámetros devueltos |
|---|---|---|---|
| Read_Class_of_Device | 0x0023 | | Status, Class_of_Device |
| Write_Class_of_Device | 0x0024 | Class_of_Device | Status |
| Read_Voice_Setting | 0x0025 | | Status, Voice_Setting |
| Write_Voice_Setting | 0x0026 | Voice_Setting | Status |
| Read_Automatic_Flush_ Timeout | 0x0027 | Connection_Handle | Status, Connection_Handle, Flush_Timeout |
| Write_Automatic_Flush_ Timeout | 0x0028 | Connection_Handle, Flush_Timeout | Status, Connection_Handle |
| Read_Num_Broadcast_ Retransmissions | 0x0029 | | Status, Num_Broadcast_Retran |
| Write_Num_Broadcast_ Retransmissions | 0x002A | Num_Broadcast_Retran | Status |
| Read_Hold_Mode_Activity | 0x002B | | Status, Hold_Mode_Activity |
| Write_Hold_Mode_Activity | 0x002C | Hold_Mode_Activity | Status |
| Read_Transmit_ Power_Level | 0x002D | Connection_Handle, Type | Status, Connection_Handle, Transmit_Power_Level |
| Read_SCO_ Flow_Control_Enable | 0x002E | | Status, SCO_Flow_Control_Enable |
| Write_SCO_Flow _Control_Enable | 0x002F | SCO_Flow_Control_Ena ble | Status |
| Set_Host_Controller_ To_Host_Flow_Control | 0x0031 | Flow_Control_Enable | Status |
| Host_Buffer_Size | 0x0033 | Host_ACL_Data_Packet _Length, Host_SCO_Data_Packet _Length, Host_Total_Num_ACL_ Data_Packets, Host_Total_Num_SCO_ Data_Packets | Status |
| Host_Number_Of_ Completed_Packets | 0x0035 | Number_Of_Handles, Connection_ Handle[i], Host_Num_Of_Complet ed_Packets [i] | |
| Read_Link_Supervision_ Timeout | 0x0036 | Connection_Handle | Status, Connection_Handle, Link_Supervision_ Timeout |
| Write_Link_Supervision_ Timeout | 0x0037 | Connection_Handle, Link_Supervision_ Timeout | Status, Connection_Handle |
| Read_Number_Of_Supported_IAC | 0x0038 | | Status, Num_Support_IAC |

| Comando | OCF | Parámetros | Parámetros devueltos |
|---|---|---|---|
| Read_Current_IAC_LAP | 0x0039 | | Status, Num_Current_IAC, IAC_LAP[i] |
| Write_Current_IAC_LAP | 0x003A | Num_Current_IAC, IAC_LAP[i] | Status |
| Read_Page_Scan_Period_ Mode | 0x003B | | Status, Page_Scan_Period_Mode |
| Write_Page_Scan_P eriod_Mode | 0x003C | Page_Scan_Period_Mod e | Status |
| Read_Page_Scan_Mode | 0x003D | | Status, Page_Scan_Mode |
| Write_Page_Scan_Mode | 0x003E | Page_Scan_Mode | Status |
| **Comandos de los Parámetros Informacionales (OGF=0x04)** | | | |
| Read_Local_Version_ Information | 0x0001 | | Status, HCI Version, HCI Revision, LMP Version, Manufacturer_Name, LMP Subversion |
| Read_Local_Supported_Features | 0x0003 | | Status, LMP_Features |
| Read_Buffer_Size | 0x0005 | | Status, HC_ACL_Data_Packet_Length, HC_SCO_Data_Packet_Length, HC_ Total_Num_ACL_Data_Packets, HC_Total_Num_SCO_Data_Pac kets |
| Read_Country_Code | 0x0007 | | Status, Country_Code |
| Read_BD_ADDR | 0x0009 | | Status, BD_ADDR |
| **Comandos de los Parámetros de Estado (OGF=0x05)** | | | |
| Read_Failed_ Contact_Counter | 0x0001 | Connection_Handle | Status, Connection_Handle, Failed_Contact_Counter |
| Reset_Failed_ Contact_Counter | 0x0002 | Connection_Handle | Status, Connection_Handle |
| Get_Link_Quality | 0x0003 | Connection_Handle | Status, Connection_Handle, Link_Quality |
| Read_RSSI | 0x0005 | Connection_Handle | Status, Connection_Handle, RSSI |
| **Comandos de Prueba (OGF=0x06)** | | | |
| Read_Loopback_Mode | 0x0001 | | Status, Loopback_Mode |
| Write_Loopback_Mode | 0x0002 | Loopback_Mode | Status |
| Enable_Device_Under_ Test_Mode | 0x0003 | | Status |
| **Los restantes OGF y OCF están reservados** | | | |

### 3.3.1.2 Paquete de eventos

Al igual que en los comandos, cada evento HCI cuenta con un número determinado de parámetros dependiendo de su función. Es por esto que la longitud del paquete que lo contiene es variable. En la figura 3.11 se muestra la estructura de los eventos HCI en unidades de bits.

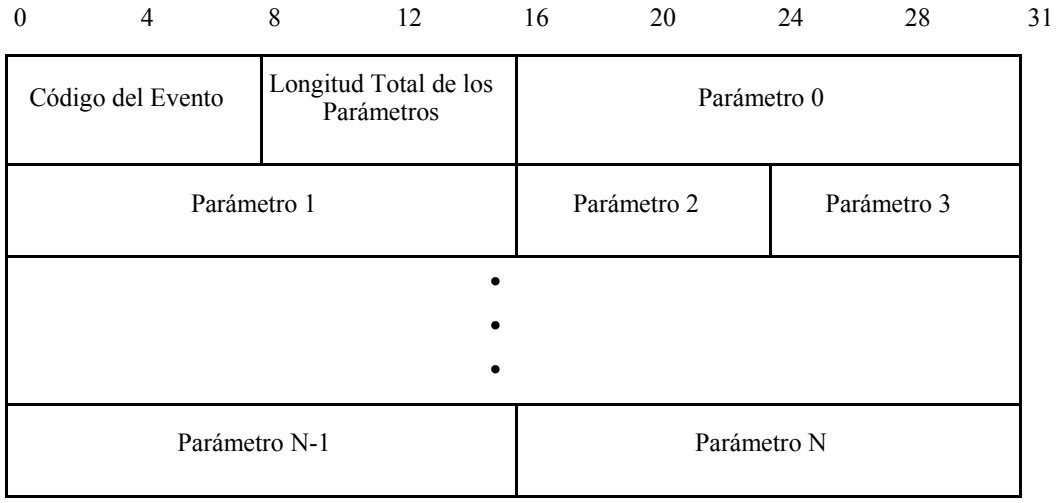


Figura 3.11 – Paquete de los eventos HCI

En el primer campo de los paquetes de eventos, se establece un *código del evento* único de un byte de longitud con el cual se identifican los eventos de otros. Seguido de este campo, la cantidad total en bytes de los parámetros correspondientes al evento es determinada por el valor del campo *Longitud Total de los Parámetros*. Los campos *Parámetro 0* hasta el *N*, representan cada uno de los parámetros del evento. En la tabla 3.6 se presentan los códigos y parámetros correspondientes a los eventos del HCI.

Tabla 3.6 – Eventos HCI

| Evento | Código | Parámetros |
|---|---|---|
| Inquiry Complete | 0x01 | Status |
| Inquiry Result | 0x02 | Num_Responses, BD_ADDR[i], Page_Scan_Repetition_Mode[i], Page_Scan_Period_Mode[i], Page_Scan_Mode[i], Class_of_Device[i] Clock_Offset[i] |
| Connection Complete | 0x03 | Status, Connection_Handle, BD_ADDR, Link_Type, Encryption_Mode |
| Connection Request | 0x04 | BD_ADDR, Class_of_Device, Link_Type |
| Disconnection Complete | 0x05 | Status, Connection_Handle, Reason |
| Authentication Complete | 0x06 | Status, Connection_Handle |
| Remote Name Request Complete | 0x07 | Status, BD_ADDR, Remote_Name |
| Encryption Change | 0x08 | Status, Connection_Handle, Encryption_Enable |
| Change Connection Link Key Complete | 0x09 | Status, Connection_Handle |
| Master Link Key Complete | 0x0A | Status, Connection_Handle, Key_Flag |
| Read Remote Supported Features Complete | 0x0B | Status, Connection_Handle, LMP_Features |

| Evento | Código | Parámetros |
|---|---|---|
| Read Remote Version Information Complete | 0x0C | Status, Connection_Handle, LMP_Version, Manufacturer_Name, LMP_Subversion |
| QoS Setup Complete | 0x0D | Status, Connection_Handle, Flags, Service_Type, Token_Rate, Peak_Bandwidth, Latency, Delay_Variation |
| Command Complete | 0x0E | Num_HCI_Command_Packets, Command_Opcode, Return_Parameters |
| Command Status | 0x0F | Status, Num_HCI_Command_Packets, Command_Opcode |
| Hardware Error | 0x10 | Hardware_Code |
| Flush Occurred | 0x11 | Connection_Handle |
| Role Change | 0x12 | Status, BD_ADDR, New_Role |
| Number Of Completed Packets | 0x13 | Number_of_Handles, Connection_Handle[i], HC_Num_Of_Completed_Packets[i] |
| Mode Change | 0x14 | Status, Connection_Handle, Current_Mode, Interval |
| Return Link Keys | 0x15 | Num_Keys, BD_ADDR [i], Link_Key[i] |
| PIN Code Request | 0x16 | BD_ADDR |
| Link Key Request | 0x17 | BD_ADDR |
| Link Key Notification | 0x18 | BD_ADDR, Link_Key,Key_Type |
| Loopback Command | 0x19 | HCI_Command_Packet |
| Data Buffer Overflow | 0x1A | Link_Type |
| Max Slots Change | 0x1B | Connection_Handle, LMP_Max_Slots |
| Read Clock Offset Complete | 0x1C | Status, Connection_Handle, Clock_Offset |
| Connection Packet Type Changed | 0x1D | Status, Connection_Handle, Packet_Type |
| QoS Violation | 0x1E | Connection_Handle |
| Page Scan Mode Change | 0x1F | BD_ADDR, Page_Scan_Mode |
| Page Scan Repetition Mode Change | 0x20 | BD_ADDR, Page_Scan_Repetition_Mode |
| | ... | |
| Bluetooth Logo Testing | 0xFE | |
| Vendor-specific debug | 0xFF | |
| **Los códigos restantes están reservados** | | |

### 3.3.1.3 Paquete de datos

Las estructuras de los paquetes de datos ACL y SCO se encuentran en las figuras 3.12 y 3.13 respectivamente.

En la creación de una conexión de datos/voz entre dos dispositivos Bluetooth, se establece un identificador único de 12 bits llamado *Connection Handle* (Manejador de Conexión). Este identificador es solamente conocido entre el *Host* y el *Controlador del Host*. En los paquetes de datos, tanto ACL como SCO, el *Connection Handle* es el primer campo transmitido.

Para los paquetes ACL, 4 bits son ocupados por las banderas *PB* (*Packet Boundary*, Límite del Paquete) y *BC* (*Broadcast*, Difusión) después del campo *Connection Handle*. Los significados de estas banderas se encuentran en la tabla 3.7. Para los paquetes SCO, estos bits son reservados para ser usados

en un futuro.

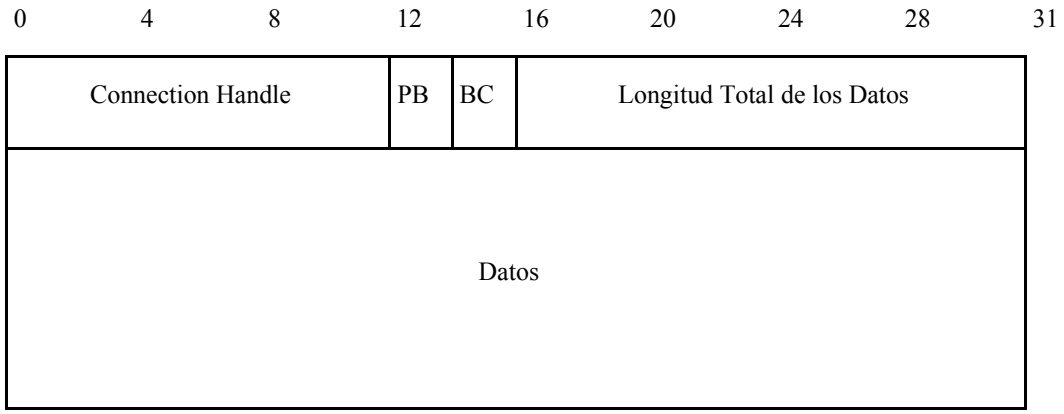


Figura 3.12 – Paquete de datos ACL

El campo *Longitud Total de los Datos*, en ambos paquetes de datos, indica la longitud total de bytes que *Datos* comprende. El campo *Datos* contiene información de otros protocolos de capas superiores como L2CAP. En paquetes ACL esta información consiste principalmente en datos, en cambio, para paquetes SCO consiste en voz.

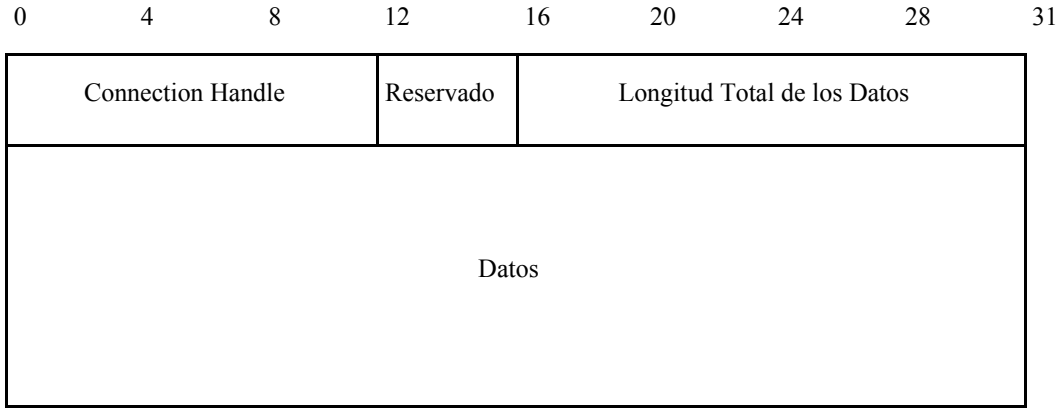


Figura 3.13 – Paquetes de datos SCO

Tabla 3.7 – Banderas PB y BC en paquetes de datos ACL

| Valor | Descripción |
|---|---|
| **Bandera PB** | |
| 00 | Reservado para un uso futuro |
| 01 | Continuar la fragmentación del paquete del Mensaje de Capas Superiores |
| 10 | Primer paquete del Mensaje de Capas Superiores (ejemplo, el comienzo de un paquete L2CAP) |
| 11 | Reservado para un uso futuro |
| **Bandera BC en paquetes transmitidos desde el *Host* al *Controlador del Host*** | |

| Valor | Descripción |
|---|---|
| 00 | No difundir. Sólo punto a punto. |
| 01 | Difusión Activa: el paquete es enviado a todos los esclavos activos (ej. El paquete no es usualmente enviado durante periodos guía en el modo *park*) y este puede ser recibido por esclavos en modo *sniff* y *park*. |
| 10 | Difusión en *Piconet*: el paquete es enviado a todos los esclavos y todos los esclavos en modo *park* (ej. El paquete es enviado durante periodos guía si hay esclavos en modo *park*), y este puede ser recibido por esclavos en modo *sniff*. |
| 11 | Reservado para un uso futuro |
| **Bandera BC en paquetes trasnmitidos desde el *Controlador del Host* al *Host*** | |
| 00 | Punto a punto. |
| 01 | Paquete recibido en un esclavo que no está en modo *park* (ya séa Difusión Activa o Difusión en *Piconet*) |
| 10 | Paquete recibido en un que está en modo *park* (Difusión en *Piconet*) |
| 11 | Reservado para un uso futuro |

## 3.3.2 Formato de los paquetes L2CAP

El protocolo L2CAP brinda soporte para la multiplexión de protocolos de niveles superiores, segmentación y reensamblaje de paquetes, y transmisión de información con QoS (*Quality Of Service*, Calidad de Servicio). Este protocolo sigue un modelo de comunicación basado en canales, en donde cada canal representa un flujo de datos entre entidades L2CAP sobre dispositivos remotos. De aquí parte, que los paquetes L2CAP tengan una estructura contemplada para el uso de canales como se muestra en la figura 3.14.

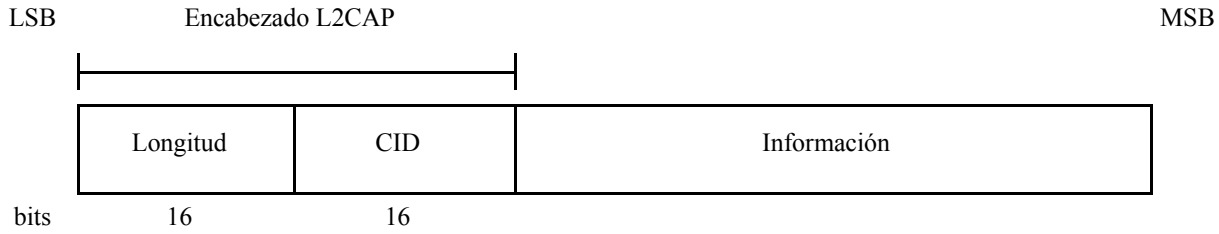


Figura 3.14 – Estructura general de los paquetes L2CAP

En la figura, el campo *Longitud* indica el tamaño en bytes que tiene la información del paquete (campo Información), mientras que el campo *CID* (*Channel Identifier*, Identificador del Canal) contiene el valor local que representa al extremo final de un canal lógico en un dispositivo remoto. La información del paquete puede contener datos propios de L2CAP o de protocolos de niveles superiores, esto depende del *CID* al que vaya dirigido. En la tabla 3.8 se presentan las interpretaciones de los posibles valores de *CID*.

En el manejo de los canales, el identificador nulo se define como un identificador ilegal y por lo tanto no debe ser usado por los dispositivos. Los valores desde el 0x0001 hasta el 0x0030 son reservados para representar funciones específicas de L2CAP. En cambio, para los CIDs restantes, las implementaciones de estos son libres de manejar de acuerdo a la forma que más encaje en esa

implementación en particular.

En L2CAP, los canales pueden ser orientados a conexión o sin conexión. Para transmitir datos sobre canales orientados a conexión, estos deben ser asignados dinámicamente y configurados a través del envío de paquetes de señalización entre los dispositivos transmisores. Después de haberse establecido los canales del dispositivo local y del remoto, estos pueden comenzar a transmitir su información. Los paquetes transmitidos sobre estos canales hacen uso del formato general de los paquetes L2CAP (Ver figura 3.14). Por otro lado, para las transmisiones sobre canales sin conexión, no es necesaria la configuración de algún canal, ya que el canal 0x0002 está reservado para llevar a cabo esta función. En las siguientes secciones, se explicarán los formatos de los paquetes para los canales de señalización y recepción sin conexión.

Tabla 3.8 – Definiciones de los CID

| CID | Descripción |
|---|---|
| 0x0000 | Identificador nulo |
| 0x0001 | Canal de señalización |
| 0x0002 | Canal de recepción sin conexión |
| 0x0003-0x0030 | Reservados para un uso futuro |
| 0x0040-0xFFFF | Asignación dinámica |

### 3.3.2.1 Paquetes de Señalización

El canal de señalización (0x0001) es un canal reservado para crear y establecer canales de datos orientados a conexión, y para negociar cambios en las características de estos canales. Para llevar a cabo estas funciones se hacen uso de los paquetes de señalización. Estos son usados para transmitir comandos de señalización entre entidades L2CAP sobre dispositivos remotos. Múltiples comandos pueden ser enviados en un sólo paquete, mientras el tamaño máximo del paquete no se exceda, y deben ser transmitidos por el canal 0x0001. La estructura del paquete se muestra en la figura 3.15.

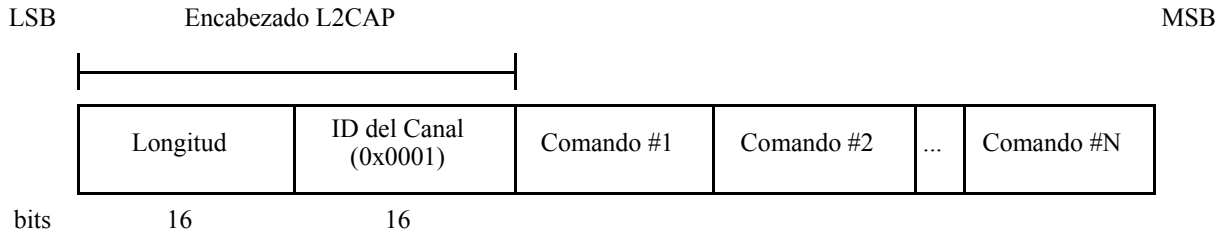


Figura 3.15 – Paquete L2CAP en el canal de señalización

Los comandos de señalización cuentan con una forma de requisito y otra de respuesta. El formato de estos comandos se muestra en la figura 3.16.

El formato de estos comandos se compone básicamente de los campos *Código*, *Identificador*, *Longitud* y *Datos*. El *Código* contiene un valor de 1 byte con el que se identifica el tipo de comando. El *Identificador*, es un campo de 1 byte de longitud que ayuda a emparejar los requisitos con las respuestas de los comandos. Esto con el fin de que al dispositivo solicitante se le entreguen las

respuestas correspondientes a sus peticiones, en caso contrario las respuestas serán descartadas. La *Longitud*, es un campo de 2 bytes que indica el número de bytes que tiene la información comprendida en el campo *Datos*. Esta información es específica de los comandos y es necesaria para su ejecución por los dispositivos replicadores, aunque para algunos comandos puede no ser requerida. En la tabla 3.9 se presentan los códigos de los comandos así como la información que transportan.

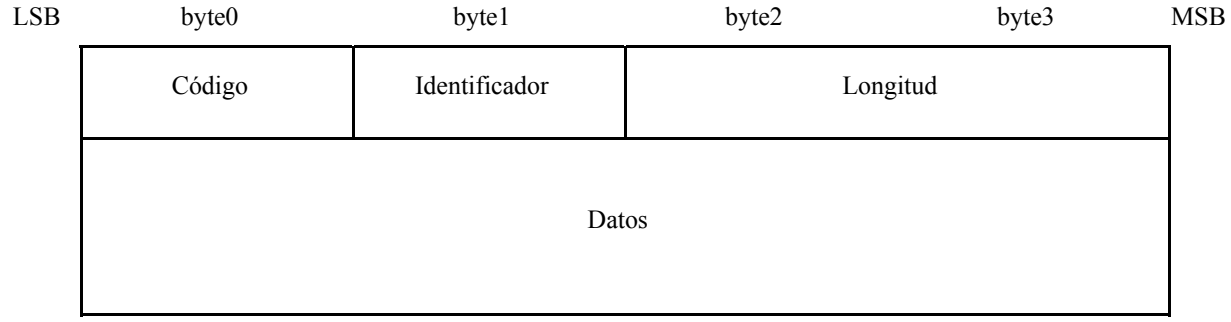


Figura 3.16 – Comando de señalización

Tabla 3.9 – Comandos de señalización

| Código | Descripción | Datos |
|---|---|---|
| 0x00 | RESERVADO | |
| 0x01 | Rechazo de comando | Reason, Data(Opcional) |
| 0x02 | Requisito de conexión | PSM, Source CID |
| 0x03 | Respuesta de conexión | Destination CID, Source CID, Result, Status |
| 0x04 | Requisito de configuración | Destination CID, Flags, Options |
| 0x05 | Respuesta de configuración | Source CID, Flags, Result, Config |
| 0x06 | Requisito de desconexión | Destination CID, Source CID |
| 0x07 | Respuesta de desconexión | Destination CID, Source CID |
| 0x08 | Requisito de eco | Data (Opcional) |
| 0x09 | Respuesta de eco | Data (Opcional) |
| 0x0A | Requisito de información | InfoType |
| 0x0B | Respuesta de información | Data (Opcional), InfoType, Result |

### 3.3.2.2 Paquetes del canal de recepción sin conexión

El canal de recepción sin conexión (0x0002) es usado como un canal donde a uno o a un grupo de dispositivos les llega la información transmitida por otro sin que se haya establecido una conexión previa. A esto viene aunado el concepto de canales orientados a grupos, en donde un canal representa a un grupo de dispositivos, con lo cual se logra que los datos enviados por este lleguen a todos los miembros del grupo. En implementaciones con este tipo de canales existen algunos inconvenientes para los miembros del grupo, como la carencia de calidad de servicios (QoS) y el no garantizar que los datos transmitidos lleguen a ellos. Esto provoca que los canales orientados a grupos no sean confiables, sin embargo, esta confianza puede ser implementada por protocolos de capas superiores.

En la figura anterior se muestra la estructura de los paquetes transmitidos sobre el canal de recepción sin conexión, que a diferencia de los paquetes sobre canales orientados a conexión, estos cuentan con el campo *PSM* (*Protocol Service Multiplexer*, Multiplexor de Protocolos de Servicios). Este campo tiene una longitud mínima de 2 bytes, aunque debido a su estructura basada en el mecanismo de extensión ISO 3309, esta longitud puede incrementarse. Los valores de *PSM* se dividen en dos rangos. Los valores del primer rango son definidos por el Bluetooth SIG e indican los protocolos. En cambio, los valores del segundo rango son asignados dinámicamente y son usados en conjunción con el protocolo SDP. Este último rango de valores puede ser usado para soportar múltiples implementaciones de un protocolo en particular. En la tabla 3.10 se muestran los posibles valores del campo PSM.

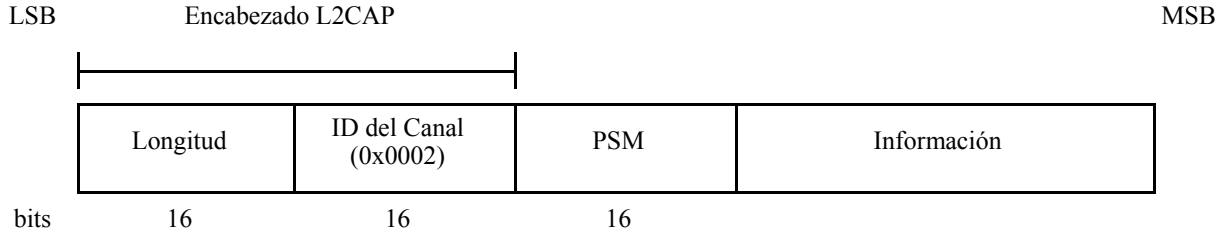


Figura 3.17 – Paquete L2CAP del canal de recepción sin conexión

Tabla 3.10 – Definiciones de los valores PSM

| PSM | Descripciones |
|---|---|
| 0x0001 | Service Discovery Protocol (SDP) |
| 0x0003 | RFCOMM |
| 0x0005 | Telephony Control Protocol |
| < 0x1000 | RESERVADOS |
| 0x1001-0xFFFF | Asignados dinámicamente |

## 3.3.3 Formato de los paquetes SDP

El protocolo SDP brinda los medios para la localización de los servicios proporcionados o disponibles a través de un dispositivo Bluetooth, así como para la determinación de los atributos de estos servicios. Los servicios son entidades que pueden proveer información, ejecutar una acción o controlar un recurso en nombre de otra entidad. Los servicios pueden ser implementados como software, hardware o una combinación de estos. Los atributos definen las características de un servicio y comúnmente incluyen el tipo o clase del servicio y la información necesaria del mecanismo o protocolo para utilizar el servicio.

En las transacciones SDP se involucran un servidor y un cliente situados en dispositivos remotos. El servidor cuenta con una lista de registros en donde se encuentran almacenados los servicios y sus atributos, mientras que el cliente tiene la capacidad de recuperar esta información. El flujo de la información entre el servidor y el cliente SDP se basa en el modelo de envío de requisitos, hechas por el cliente, y de la contestación de estos en forma de respuestas, realizadas por el servidor, que se

transmiten mediante el uso de PDUs (*Protocol Data Unit*, Unidad de Datos del Protocolo). Diferente de los protocolos anteriores, en SDP primero se transmiten los bytes más significativos antes de enviar los menos significativos (estándar *Big Endian*). Cada PDU cuenta con una estructura como la mostrada en la figura 3.18.

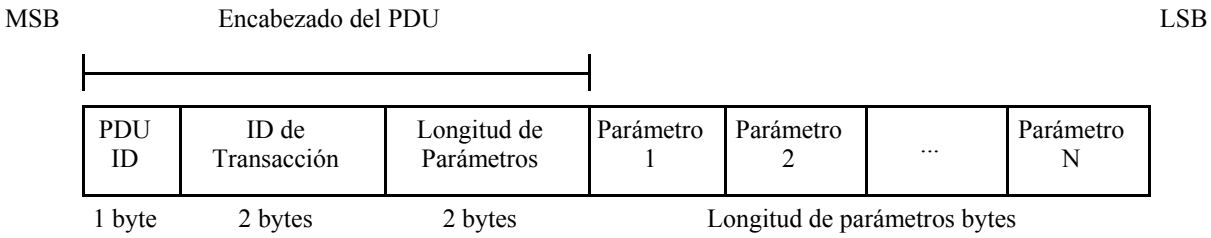


Figura 3.18 – Formato del PDU SDP

En la figura, los PDUs primeramente se componen del campo *PDU ID*, que sirve para identificar el tipo de PDU y con el que se puede determinar el significado y los parámetros que lo conforman. El campo *ID de Transacción* cuenta con 2 bytes de longitud para almacenar el valor que identifica a un requisito PDU. Este valor es usado para emparejar las respuestas PDU con sus requisitos correspondientes. El campo *Longitud de Parámetros* contiene el número de bytes de los parámetros del PDU que son representados por los campos *Parámetro 1... N*. En la tabla 3.11 se muestran los tipos de PDU con sus parámetros correspondientes.

Tabla 3.11 – Tipos de PDU SDP

| PDU ID | Descripción | Parámetros |
|---|---|---|
| 0x00 | Reservado | |
| 0x01 | Respuesta de Error | ErrorCode, ErrorInfo |
| 0x02 | Requisito de Búsqueda de Servicios | ServiceSearchPattern, MaximumServiceRecordCount, ContinuationState |
| 0x03 | Respuesta de Búsqueda de Servicios | TotalServiceRecordCount, CurrentServiceRecordCount, ServiceRecordHandleList, ContinuationState |
| 0x04 | Requisito de Atributos del Servicio | ServiceRecordHandle, MaximumAttributeByteCount, AttributeIDList, ContinuationState |
| 0x05 | Respuesta de Atributos del Servicio | AttributeListByteCount, AttributeList, ContinuationState |
| 0x06 | Requisito de Búsqueda de Servicios y Atributos | ServiceSearchPattern, MaximumAttributeByteCount, AttributeIDList, ContinuationState |
| 0x07 | Respuesta de Búsqueda de Servicios y Atributos | AttributeListsByteCount, AttributeLists, ContinuationState |
| 0x08-0xFF | Reservados | |

La mayoría de estos parámetros se encuentran representados por los elementos de datos que SDP define. Los elementos de datos cuentan con una estructura general que facilita el manejo de varios tipos de datos con complejidades diferentes. Esta estructura se explicará en la siguiente sección.

### 3.3.3.1 Formato de los elementos de datos

Un elemento de datos se conforma de un encabezado con un byte de longitud y un campo *Datos* con una longitud establecida por el encabezado (Ver figura 3.19). El encabezado se compone de un *descriptor de tipos* y otro de *tamaños*. Los 5 bits más significativos son ocupados por el *Descriptor de tipo* para indicar el tipo de información que el elemento de datos almacena en su campo datos, en cambio, los bits restantes son utilizados por el *Descriptor de tamaño* para establecer la longitud en bytes de la información almacenada. En la tabla 3.12 se muestran los diferentes tipos de datos que los elementos de datos pueden representar.

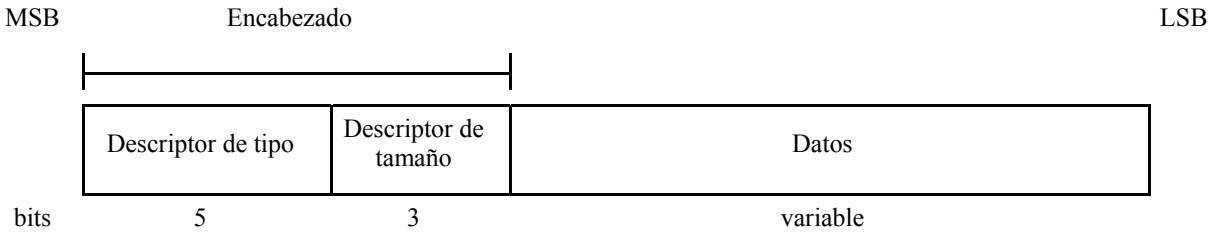


Figura 3.19 – Estructura general de los elementos de datos SDP

Tabla 3.12 – Tipos de datos en los elementos de datos

| Valor del descriptor de tipos | Valores válidos del descriptor de tamaños | Descripción |
|---|---|---|
| 0 | 0 | Nil, the null type |
| 1 | 0, 1, 2, 3, 4 | Unsigned integer |
| 2 | 0, 1, 2, 3, 4 | Signed twos-complement integer |
| 3 | 1, 2, 4 | Un UUID (*Universally Unique Identifier*, Identificador Universalmente Único) |
| 4 | 5, 6, 7 | Text string |
| 5 | 0 | Bolean |
| 6 | 5, 6, 7 | Data element sequence, elemento de datos cuyo campo datos es una secuencia de elementos de datos. |
| 7 | 5, 6, 7 | Data element alternative, elemento de datos cuyo campo datos es una secuencia de elementos de datos del cual un elemento está para ser seleccionado. |
| 8 | 5, 6, 7 | Un URL (*Uniform Resource Locator*, Localizador Uniforme de Recursos) |
| 9-31 | | Reservados |

El *Descriptor de tamaño* es representado por un índice de tamaños de 3 bits seguidos por 0, 8, 16 o 32 bits adicionales. Los 3 bits del índice de tamaños son los bits menos significativos de los encabezados de los elementos de datos. En la tabla 3.13 se presentan las interpretaciones del índice de tamaño para la obtención de la longitud de los datos.

Tabla 3.13 – Interpretaciones del índice de tamaños

| Índice de tamaños | Bits adicionales | Tamaño de los datos |
|---|---|---|
| 0 | 0 | 1 byte. Excepción: si el elemento de datos es *nil* el tamaño de los datos es 0 bytes. |
| 1 | 0 | 2 bytes |
| 2 | 0 | 4 bytes |
| 3 | 0 | 8 bytes |
| 4 | 0 | 16 bytes |
| 5 | 8 | El tamaño de los datos está contenido en los 8 bits adicionales, los cuales son interpretados como un entero sin signo. |
| 6 | 16 | El tamaño de los datos está contenido en los 16 bits adicionales, los cuales son interpretados como un entero sin signo. |
| 7 | 32 | El tamaño de los datos está contenido en los 32 bits adicionales, los cuales son interpretados como un entero sin signo. |

### 3.3.4 Formato de los paquetes RFCOMM

El protocolo RFCOMM provee una emulación de puertos seriales sobre el protocolo L2CAP, contando con un multiplexor para el manejo apropiado de estos. Este protocolo está basado en el estándar ETSI TS 07.10 [10], aunque algunas especificaciones del estándar se han omitido y otras se han adaptado. De forma similar a L2CAP, a cada conexión que se establece entre una aplicación servidor y una cliente le es asignada un DLCI (*Data Link Channel Identifier*, Identificador del Canal de Enlace de Datos) de 6 bits con el fin de que cualquier conexión pueda ser identificada. Los valores de DLCI que pueden ser asignados se encuentran entre 2 y 61, el DLCI igual a 0 es reservado para el canal dedicado al control de multiplexores y los valores 1, 62 y 63 no son usados. RFCOMM maneja diferentes tipos de paquetes en sus transmisiones, los cuales consisten en el comando SABM (*Set Asynchronous Balanced Mode*, Modo de Colocación Asíncrono Balanceado), la respuesta UA (*Unnumbered Acknowledgement*, Reconocimiento No enumerado), la respuesta DM (*Disconnected Mode*, Modo Desconectado), el comando DISC (*Disconnect*, Desconectar) y el comando/repuesta UIH (*Unnumbered Information with Header check*, Información No enumerada con Chequeo de Encabezado). En la figura 3.20 se presenta la estructura general de estos paquetes.

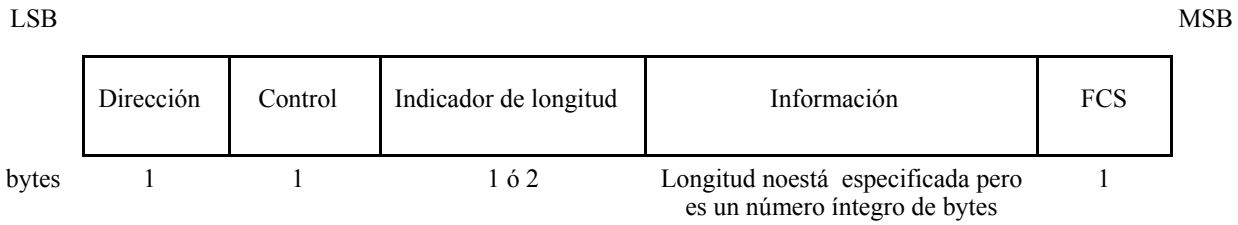


Figura 3.20 – Estructura básica de los paquetes RFCOMM

El campo *Dirección* se compone de los bits *EA*, *C/R* (*Command/Response*, Comando/Respuesta) y el subcampo *DLCI* (Ver figura 3.21). El valor de *DLCI* identifica la conexión hacia donde el paquete va dirigido (*D1...D6*). El bit *EA* es utilizado para extender la longitud en bytes del campo *Dirección*. Cuando *EA* es 1 significa que el byte que lo contiene es el último que conforma al campo, en cambio si

*EA* es 0 indica que el byte siguiente también conforma al campo *Dirección*. En RFCOMM sobre Bluetooth v1.1, el bit *EA* es 1 pero su valor puede ser cambiado en futuras versiones. Por otro lado, el bit *C/R* identifica al paquete como un comando o como una respuesta y sus valores dependen sobre el tipo del paquete, así como de la estación Iniciador y Contestador (Ver tabla 3.14). El Iniciador es la estación que toma la iniciativa para inicializar el multiplexor (ej. envío del comando SABM en DLCI 0) y el Contestador es la estación que acepta la inicialización del multiplexor (ej. envío de la respuesta UA en DLCI 0).

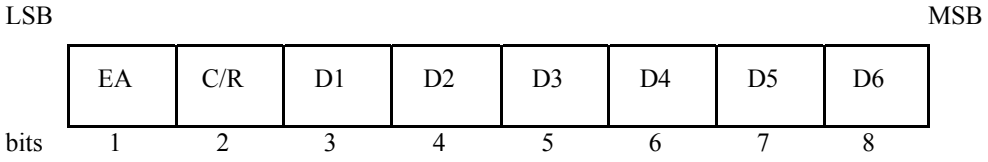


Figura 3.21 – Campo *Dirección*

Tabla 3.14 – Uso del bit *C/R*

| Comando/Respuesta | Dirección | Valor de C/R |
|---|---|---|
| **Para paquetes SABM, UA, DM y DISC** | | |
| Comando | Iniciador → Contestador | 1 |
| | Contestador → Iniciador | 0 |
| Respuesta | Iniciador → Contestador | 0 |
| | Contestador → Iniciador | 1 |
| **Para paquetes UIH** | | |
| Comando/Respuesta | Iniciador → Contestador | 1 |
| | Contestador → Iniciador | 0 |

El campo *Control* contiene el tipo de paquete y al bit *P/F*. En la tabla 3.15 se muestra los valores de este campo correspondientes al tipo del paquete, así como el número de bit que ocupa el bit *P/F* (bit 5). El bit *P/F* es usado como el bit *P* (*Poll*, Solicitar) en paquetes de comandos y como el bit *F* (*Final,* Final) en paquetes de respuestas. De forma general, el bit *P* colocado en 1 es usado por una estación para solicitarle a otra estación una respuesta o una secuencia de respuestas, en cambio el bit *F* es usado por una estación para indicar que el paquete de respuesta se transmitió como resultado de un comando solicitante. En paquetes UIH el significado del bit *P/F* se redefine de acuerdo al uso del control de flujo basado en créditos. En caso de que no se esté utilizando este tipo de control, el bit *P* siempre es colocado a 0 y el bit *F* se comporta en su forma general. Por otro lado, en caso de que se esté usando el control de flujo basado en créditos, en la estructura del paquete RFCOMM es agregado el campo *Créditos* (1 byte) entre los campos *Indicador de longitud* e *Información* siempre y cuando *P/F* sea 1, pero si *P/F* es 0 la estructura se mantiene como en la figura 3.20. El valor del campo *Créditos* significa el número de paquetes que el transmisor tiene disponible en su buffer para el canal identificado por el *DLCI*.

Tabla 3.15 – Definiciones de los bits del campo *Control*[15]

| Tipo de paquete | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Comando SABM | 1 | 1 | 1 | 1 | P/F | 1 | 0 | 0 |
| Respuesta UA | 1 | 1 | 0 | 0 | P/F | 1 | 1 | 0 |
| Respuesta DM | 1 | 1 | 1 | 1 | P/F | 0 | 0 | 0 |
| Comando DISC | 1 | 1 | 0 | 0 | P/F | 0 | 1 | 0 |
| Comando y respuesta UIH | 1 | 1 | 1 | 1 | P/F | 1 | 1 | 1 |

El campo *Indicador de longitud* establece, en 7 ó 15 bits, la longitud de los datos contenidos en el campo *Información* (Ver figura 3.22). En el primer byte se encuentra un bit *EA* el cual indica si el campo *Indicador de longitud* está constituido por 1 (*EA* igual a 1) ó 2 bytes (*EA* igual a 0).

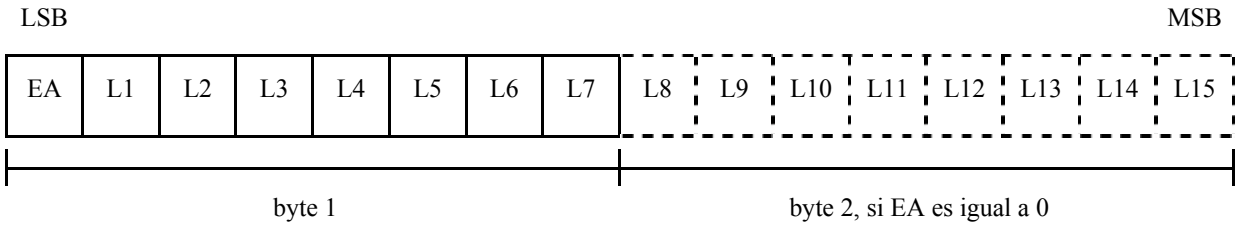


Figura 3.22 – Campo *Indicador de Longitud*

Enseguida del campo *Indicador* se tiene al campo *Información*, en el cual están almacenados los datos de protocolos de capas superiores o los comandos en caso de que se esté utilizando el canal de control (DLCI 0). El campo *Información* únicamente se encuentra en los paquetes UIH.

El último campo que compone a un paquete RFCOMM es el *FCS* (*Frame Checking Sequence*, Secuencia de Chequeo del Paquete), en el cual se almacena un valor utilizado para verificar la integridad del paquete. En el cálculo de este valor, grupos de campos diferentes son requeridos dependiendo sobre el tipo del paquete. Para los paquetes SABM, DISC, UA y DM se requieren de los campos *Dirección*, *Control* y el *Indicador de longitud*, en cambio para los paquetes UIH sólo se requieren de los campos *Dirección* y *Control*.

### 3.3.4.1 Formato de los comandos del canal de control

En el inicio de comunicación entre una aplicación servidor y una cliente, un canal de control es establecido con un DLCI igual 0. Este canal es usado para transmitir información entre dos multiplexores. La comunicación entre estos multiplexores se lleva a cabo con el envío de comandos, a los cuales se les asocia una respuesta. Los comandos son transportados en paquetes UIH y múltiples comandos pueden ser incluidos mientras la longitud máxima del paquete no se exceda. La estructura de los comandos se muestra en la figura 3.23.

[15]Bits ordenados del menos al más significativo

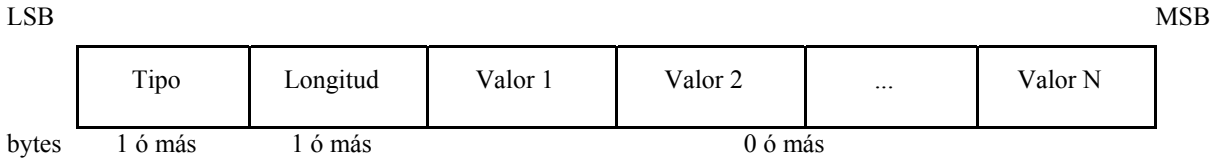


Figura 3.23 – Estructura de los comandos o mensajes de RFCOMM

La estructura de los comandos se componen de los campos *Tipo* para indicar el tipo de comando, *Longitud* para indicar el número de campos *Valor* (cada campo *Valor* tiene 1 byte de longitud) y los campos *Valor1…ValorN* que contienen los parámetros del comando. Como se muestra en la figura 3.23, lo campos *Tipo* y *Longitud* pueden estar constituidos por una secuencia de 1 ó más bytes, esto depende del valor del bit de extensión *EA*. Este bit es colocado a 1 en el último byte de la secuencia, mientras que en otros bytes es colocado a 0. Si únicamente un byte es transmitido entonces el bit *EA* es 1. En la estructura del campo *Tipo*, a diferencia del campo *Longitud*, el primer byte contiene un bit *C/R* para indicar si el mensaje es un comando (*C/R* igual a 1) o una respuesta (*C/R* igual a 0). En la figura 3.24a se muestra el primer byte del campo *Tipo* y en la figura 3.24b sus bytes subsecuentes. Todos los bytes que conforman al campo *Longitud* tienen el formato que se muestra en la figura 3.24b.

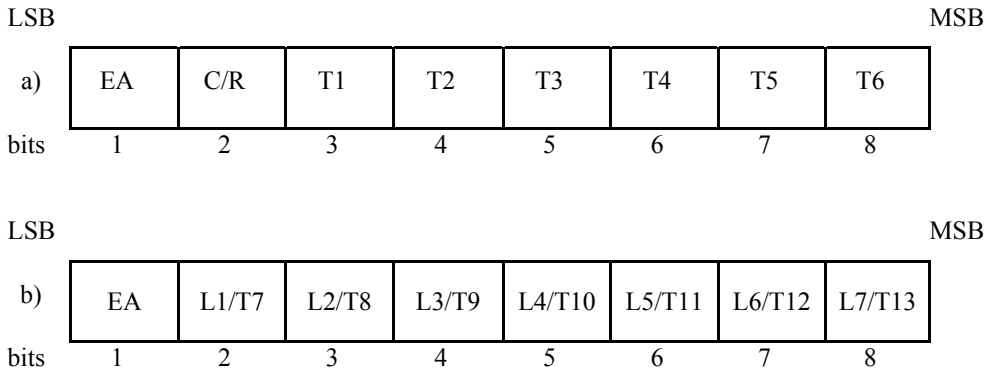


Figura 3.24 – a) Formato del primer byte del campo Tipo. b) Formato de los bytes del campo Longitud (bits L) y de los bytes subsecuentes del campo Tipo (bits T).

En la tabla 3.16 se presentan los valores de los campos específicos de cada tipo de comando.

Tabla 3.16 – Tipos de comandos RFCOMM

| Comando | Tipo | Longitud | Parámetros en Valor1...ValorN |
|---|---|---|---|
| Test Command (Test) | 0x08 | No especificada | Verification pattern |
| Flow Control On Command (Fcon) | 0x28 | 0 | |
| Flow Control Off Command (Fcoff) | 0x18 | 0 | |
| Modem Status Command (MSC) | 0x38 | 2 ó 3 | DLCI, V.24 signals, break signals (opcional) |
| Remote Port Negotiation Command (RPN) | 0x24 | 1 ó 8 | DLCI y Opcionales: Baud rate, number of data bits, number of stop bits, parity, parity type, flow control, XON character, XOF character, parameter mask |
| Remote Line Status (RLS) | 0x14 | 2 | DLCI, Line status |

| Comando | Tipo | Longitud | Parámetros en Valor1...ValorN |
|---|---|---|---|
| DLC parameter negotiation (PN) | 0x20 | 8 | DLCI, type of frames, type of convergence layer, priority, acknowledgement timer, maximum frame size, maximum number of retransmissions, window size |
| Non Supported Command Response (NSC) | 0x04 | 1 | Command Type |

## 3.3.5 Análisis de los paquetes Bluetooth

En esta parte se realiza el análisis del contenido de un paquete capturado en las transmisiones realizadas entre un dispositivo PDA y una Laptop con soporte Bluetooth (Ver figura 3.25). Para que el paquete pudiera completar su análisis, se requirió del análisis previo de otro paquete capturado momentos antes (Ver figura 3.26). Más adelante se explicará esta dependencia. Ambos paquetes fueron capturados con la aplicación *sniffer* desarrollada en este trabajo de tesis, aunque también pudieron haber sido capturados con el analizador de paquetes Hcidump de BlueZ.

Para llevar a cabo el análisis de los paquetes, se denominará al paquete de la figura 3.25 como el paquete 1 y se seccionará en: encabezado HCI, encabezado L2CAP y datos. De la misma forma, al paquete de la figura 3.26 se le denominará como el paquete 2 y se seccionará en: encabezado HCI, encabezado L2CAP, encabezado RFCOMM y datos.

### 3.3.5.1 Análisis del primer paquete Bluetooth capturado (Paquete 1)

| No byte (Hexadecimal) | Contenido del paquete en bytes (Hexadecimal) | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | 02 | 29 | 20 | 0c | 00 | 08 | 00 | 01 | 00 | 02 | 04 | 04 | 00 | 03 | 00 | 41 |
| 0010 | 00 | | | | | | | | | | | | | | | |

Figura 3.25 – Contenido del primer paquete Bluetooth a analizar

El paquete capturado que se muestra en la figura 3.25 se encuentra dividido en tres secciones: Encabezado HCI, Encabezado L2CAP y Datos. Describiremos cada una de ellas a continuación.

**Encabezado HCI**

Este encabezado está representado por la sección sombreada en la figura 3.25. En realidad, este encabezado está formado por los últimos 4 bytes debido a que el byte 0 (0x02) es proporcionado por las librerías que la aplicación *sniffer* utiliza para conocer el tipo del paquete HCI (Se trata de las librerías BlueZ que se explicarán en el capítulo 4). De acuerdo a la tabla 3.17, el paquete 1 es un paquete de datos ACL.

*Los bytes 1 y 2*

Los primeros 12 bits de estos bytes, tomándolos de derecha a izquierda, forman al campo *Connection Handle* (0x029). Este campo indica la conexión HCI a la cual el paquete va dirigido. Los 4 bits restantes del byte 2 (0x2) pertenecen a las banderas PB y BF respectivamente. La bandera PB ($10_2$) indica que este paquete es el primero que conforma a un mensaje de protocolos superiores (ej. L2CAP), aunque el mensaje puede estar constituido por un sólo paquete (bandera utilizada en el proceso de fragmentación). La bandera *BF* ($00_2$) indica que se está utilizando una transmisión punto a punto.

Tabla 3.17 – Interpretaciones del primer byte de algún paquete HCI capturado

| Byte 0 | Definiciones |
|---|---|
| 0x01 | Comando HCI |
| 0x02 | Paquete de datos ACL |
| 0x03 | Paquete de datos SCO |
| 0x04 | Evento HCI |

*Los bytes 3 y 4*

Estos bytes, tomándolos de derecha a izquierda, forman al campo *Longitud Total de los Datos* (0x000c), el cual indica el número de bytes que comprenden los datos del paquete ACL, que en este caso se trata de la longitud del encabezado L2CAP (12 bytes).

**Encabezado L2CAP**

Se considera que los bytes siguientes al encabezado HCI (los bytes del 5 hasta el 16) pertenecen al protocolo L2CAP, debido a que hasta el momento L2CAP ha sido el único protocolo superior a HCI que se ha establecido en las especificaciones Bluetooth v1.1. La sección cerrada de la figura 3.25 representa al encabezado L2CAP.

*Los bytes 5 y 6*

Estos bytes forman al campo *Longitud* y se interpretan de derecha a izquierda (0x0008). Este campo indica que los datos transportados por el protocolo L2CAP consisten de 8 bytes (sección Datos).

*Los bytes 7 y 8*

Estos bytes pertenecen al campo *CID* y son usados para especificar el extremo final del canal hacia donde se dirige el paquete. De acuerdo al valor de este campo, se puede argumentar que los bytes siguientes incluyen a uno o más comandos de señalización debido a que el paquete va dirigido hacia el CID 0x0001 (los bytes se toman de derecha a izquierda).

**Datos**

Como se mencionó anteriormente, los bytes del 9 al 16 (sección libre de la figura 3.25) son parte de los comandos de señalización usados en el control de los multiplexores que el protocolo L2CAP emplea. En estos bytes pueden estar comprendidos múltiples comandos.

*El byte 9*

Este byte forma al campo *Código* (0x02), el cuál indica el tipo de comando transmitido. En este caso se trata del comando Requisito de Conexión (Ver tabla 3.9). Este es usado por un dispositivo Bluetooth (*transmisor*) para entablar una conexión con algún dispositivo disponible *(receptor)*.

*El byte 10*

Este byte forma al campo *Identificador* (0x04) y es usado para emparejar los comandos requisitos con sus respuestas correspondientes. Por lo tanto, el comando Respuesta de Conexión, enviado por el *receptor*, deberá contener en su campo *Identificador* el valor de 0x04 para que el *transmisor* lo pueda aceptar.

*Los bytes 11 y 12*

Estos bytes, tomándolos de derecha a izquierda, pertenecen al campo *Longitud* (0x0004) y son

usados para indicar el número de bytes que comprenden los parámetros del comando. En este caso, los parámetros del comando Requisito de Conexión consisten de 4 bytes.

*Los bytes del 13 al 16*

Estos bytes, interpretándolos de derecha a izquierda, representan los parámetros del comando Requisito de Conexión, en este caso se trata de los parámetros *PSM* y *Source CID* respectivamente (Ver tabla 3.9). Estos parámetros son usados para configurar la conexión solicitada por el *transmisor*. *PSM* establece que el protocolo a usarse en la conexión es RFCOMM (0x0003, ver tabla 3.10) y *Source CID* indica que el *transmisor* utiliza el identificador 0x0041 como el extremo final de la conexión hacia donde llegarán los datos enviados por el *receptor*.

Por otro parte, si el *receptor* acepta la conexión, entonces este le enviará al *transmisor* el comando Respuesta de Conexión, en el cual le indicará el CID de la conexión que él podrá usar para el envío de sus datos.

El análisis anterior del paquete de la figura 3.25 se resume en la tabla 3.18.

Tabla 3.18 – Análisis del paquete de la figura 3.25

| No. de Byte | Campo (s) | Valor (es) | Significado |
|---|---|---|---|
| **Encabezado HCI** | | | |
| 0 | Tipo | 0x02 | Paquete ACL |
| 1 y 2 | Connection Handle, PB/BF | 0x029, 0x2 | Identificador de la conexión, No es un fragmento y fue transmitido de punto a punto |
| 3 y 4 | Longitud Total de los Datos | 0x000c | 12 bytes |
| **Encabezado L2CAP** | | | |
| 5 y 6 | Longitud | 0x0008 | 8 bytes |
| 7 y 8 | CID | 0x0001 | Canal de señalización |
| **Datos (1 comando)** | | | |
| 9 | Código | 0x02 | Comando Requisito de Conexión |
| 10 | Identificador | 0x04 | Identificador para la respuesta de conexión |
| 11 y 12 | Longitud | 0x0004 | 4 bytes de los parámetros |
| 13 y 14 | PSM | 0x0003 | La conexión manejará RFCOMM |
| 15 y 16 | Source CID | 0x0041 | Identificador del canal |

### 3.3.5.2 Análisis del segundo paquete Bluetooth capturado (Paquete 2)

El paquete capturado que se muestra en la figura 3.26 se encuentra dividido en cuatro secciones: Encabezado HCI, Encabezado L2CAP, Encabezado RFCOMM y Datos. Se describirá cada una de ellas a continuación.

**Encabezado HCI**

Este encabezado es interpretado de forma similar al encabezado HCI del paquete 1 (Ver Encabezado HCI en la sección 3.3.5.1). El paquete va dirigido a la conexión 0x029 (campo *Connection Handle*), es el primer paquete de un mensaje perteneciente a protocolos de capas superiores (bandera *PB*), es

enviado en una transmisión punto a punto (bandera *BF*) y la única diferencia es que los datos HCI que transporta están comprendidos en 23 bytes (campo *Longitud Total de los Datos* igual a 0x0017). El contenido de este encabezado es representado por la sección rodeada con línea gruesa en la figura 3.26.

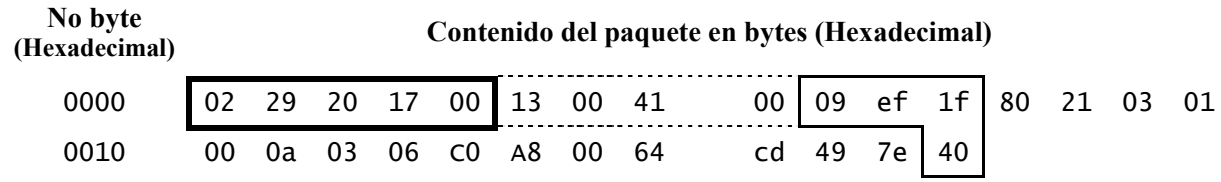


Figura 3.26 – Contenido del segundo paquete Bluetooth a analizar

**Encabezado L2CAP**

El análisis de este encabezado es análogo al realizado en el encabezado L2CAP del paquete 1 (Ver Encabezado L2CAP en la sección 3.3.5.1). Sólo que en este caso los datos L2CAP cuentan con 19 bytes de longitud (campo *Longitud* igual a 0x0013) y son destinados al extremo final 0x0041 del canal (campo *CID*). La sección cerrada en línea punteada de la figura 3.26 representa a este encabezado.

**Encabezado RFCOMM**

De acuerdo al resultado del análisis del paquete 1 (Ver *los bytes del 13 al 16* en la sección 3.3.5.1), el CID 0x0041 fue establecido como un extremo final del canal hacia donde se podrán enviar datos RFCOMM, por lo que se concluye que el encabezado sucesivo al encabezado L2CAP pertenece al protocolo RFCOMM. El contenido de este encabezado se presenta en la figura 3.26 como la sección rodeada con línea delgada.

*El byte 9*

Este byte forma al campo *Dirección* (0x09). Este campo es usado para indicar el DLC (*Data Link Channel*, canal de enlace de datos) hacia donde se dirige el mensaje RFCOMM (subcampo *DLCI*, últimos 5 bits) y para establecer si este se trata de un comando o una respuesta (subcampo *C/R*, segundo bit). En este caso, el mensaje es una respuesta[16] (Ver tabla 3.14) que va dirigido hacia el canal DLCI 0x02.

*El byte 10*

Este byte forma al campo *Control* (0xef). En este campo se indica el tipo de mensaje RFCOMM que se está transmitiendo así como el significado del bit *P/F* (bit 5). Con los valores de los primeros 4 y los 3 últimos bits de este byte ($111x1111_2$) se determina que el mensaje es de tipo UIH. En estos tipos de mensajes el valor del bit *P/F* es usado particularmente para determinar si el campo *Créditos* está o no incluido en la cabecera RFCOMM, ya que el control de flujo basado en créditos está siendo usado (este es negociado por el comando de control del multiplexor PN antes de que se haya establecido el primer DLC[17]). Por lo tanto, como *P/F* está colocado a 0 ($xxx0xxxx_2$) entonces se puede afirmar que el campo *Créditos* no es insertado después del campo *Indicador de Longitud*.

*El byte 11*

Con este byte se forma el campo *Indicador de Longitud* (0x1f) debido a que el bit *EA* (el primer bit) está colocado en 1, aunque puede estar formado por 2 bytes si *EA* fuera 0. Los últimos 7 bits de este

[16]Para inferir esta respuesta, tuvieron que ser analizados los paquetes de inicialización del multiplexor RFCOMM (paquetes UIH y UA).
[17]Para determinar el flujo basado en créditos, se analizaron los paquetes RFCOMM que transportarán comandos PN.

campo ($0001111_2$) indican que los datos del mensaje RFCOMM están contenidos en 15 bytes (longitud del campo *Información*).

*El byte 27*

Este byte forma al campo *FCS*, con el cuál se verifica la integridad del paquete. El valor de este campo (0x40) fue calculado para los campos *Dirección* y *Control*, por lo tanto, no se puede garantizar la integridad de los datos restantes a los comprendidos por estos campos.

**Datos**

Esta sección está conformada por los bytes del 12 al 26 como se muestran en la figura 3.26. Estos bytes conforman al campo *Información* del mensaje RFCOMM, pero el motivo por el cual se tratan en una sección a parte se debe a que el contenido de este campo pertenece a protocolos o aplicaciones de capas de nivel superior (ej. OBEX, PPP, HotSync, etc.).

El análisis anterior del paquete de la figura 3.26 se resume en la tabla 3.19.

Tabla 3.19 – Análisis del paquete de la figura 3.26

| No. de Byte | Campo (s) | Valor (es) | Significado |
|---|---|---|---|
| **Encabezado HCI** | | | |
| 0 | Tipo | 0x02 | Paquete ACL |
| 1 y 2 | Connection Handle, PB/BF | 0x029, 0x2 | Identificador de la conexión, No es un fragmento y fue transmitido de punto a punto |
| 3 y 4 | Longitud Total de los Datos | 0x0017 | 23 bytes |
| **Encabezado L2CAP** | | | |
| 5 y 6 | Longitud | 0x0013 | 19 bytes |
| 7 y 8 | CID | 0x0041 | Identificador de canal |
| **Encabezado RFCOMM** | | | |
| 9 | Dirección | 0x09 (0x0 , 0x02) | C/R = 0x0 (Respuesta), DLCI = 0x02 |
| 10 | Control | 0xef | Type = $111x1111_2$ (Mensaje UIH), P/F = $000x0000_2$ |
| 11 | Indicador de Longitud | 0x1f | 15 bytes |
| 12-26 | Datos | | Posiblemente información OBEX, PPP, HotSync, etc. |
| 27 | FCS | 0x40 | Valor CRC calculado sobre los campos Dirección y Control |

# Análisis y diseño del *sniffer*

## 4.1 Introducción

En general, el desarrollo de los proyectos de software se puede llevar a cabo utilizando los métodos de cascada o iterativo[18]. Actualmente, el método iterativo es el más usado en el modelado orientado a objetos debido, básicamente, a la obtención de un mejor control en el manejo de los riesgos y en el proceso de desarrollo del proyecto de software. Existen diferentes tipos de procesos para el desarrollo iterativo, entre los más sobresalientes se encuentran el RUP (*Rational Unified Process*, Proceso Racional Unificado) y el XP (*Extreme Programming*, Programación Extrema), cada uno orientado a proyectos con características particulares. El proceso iterativo RUP fue el más adecuado para el desarrollo de la aplicación *sniffer* por que, además de la gran disponibilidad de documentación y herramientas que existen para su implementación, los requerimientos de la aplicación no estarán cambiando constantemente como lo exige el proceso iterativo ágil XP [URL 16].

Una herramienta utilizada en conjunción con RUP para el desarrollo de la aplicación *sniffer* es UML (*Unified Modeling Language*, Lenguaje de Modelado Unificado). UML es una especificación de la OMG (*Object Management Group*, Grupo de Administración de Objetos) que define un lenguaje gráfico para visualizar, especificar, construir y documentar los artefactos[19] de los sistemas con objetos distribuidos [URL 17]. Este lenguaje no es únicamente usado para modelar la estructura, el comportamiento y la arquitectura de una aplicación, sino también los procesos de negocios y estructuras de datos.

En este apartado, UML se utiliza en la descripción y el diseño de la aplicación *sniffer* con artefactos como los diagramas de casos de uso, diagramas de secuencia y diagramas de clases [11].

## 4.2 Descripción general de la aplicación

En la actualidad, los *sniffers* inalámbricos han sido creados por comunidades de programadores independientes para su uso libre y por empresas dedicadas a la elaboración de software y equipo de prueba para su uso comercial. Los *sniffers* elaborados por estas empresas van dirigidos a los administradores de red, desarrolladores de software y empresas que utilicen tecnologías inalámbricas en sus productos con la finalidad de someterlos a pruebas antes de lanzarlos al mercado. La mayoría de estos *sniffers* son muy caros y normalmente trabajan sobre el sistema operativo Windows. Es por esto que algunos usuarios, principalmente administradores de red y desarrolladores de software, han decidido utilizar software libre.

En la comunidad del software libre los *sniffers* fueron desarrollados para fines educativos, aunque la mayoría de los usuarios suelen utilizarlos para implementar ataques inalámbricos con los que puedan explotar los servicios proporcionados por alguna red. A diferencia de los *sniffers* comerciales, los *sniffers* como software libre son comúnmente ejecutados bajo el sistema operativo Linux.

La aplicación *sniffer* desarrollada en este trabajo comprende funcionalidades y características básicas de los *sniffers* comerciales y libres que se consideran esenciales para el ámbito educativo. Estas consisten principalmente en permitir la creación de sesiones de captura de paquetes de datos en tiempo

[18]El método iterativo también se le ha conocido como método incremental, espiral y evolucionario entre otros.
[19]En RUP, es el término general usado para representar algún producto de trabajo como: código, gráficos Web, esquemas de base de datos, documentos de texto, diagramas, modelos, etc.

real tanto en redes 802.11 como en redes Bluetooth, así como proporcionar una serie de vistas en las que aparte de desplegar el contenido de los paquetes en formato ASCII y hexadecimal, también despliegue la estructura de los paquetes en forma detallada de acuerdo a los principales protocolos manejados en 802.11 y Bluetooth. Además, la aplicación *sniffer* cuenta con la opción para almacenar las sesiones de captura en algún archivo, lo que permitirá que los paquetes pertenecientes a alguna sesión puedan ser cargados y visualizados por la aplicación en cualquier instante. Algunas comparativas entre la aplicación *sniffer* desarrollada y otros *sniffers* se presentan en la Tabla 4.1.

Tabla 4.1 – Comparativas de *sniffers*

| ***Sniffers*** | **Captura de paquetes 802.11** | **Captura de paquetes Bluetooth** | **Multiplataforma** | **Interfaz Gráfica** |
|---|---|---|---|---|
| *Sniffer* desarrollado | × | × | × | × |
| Kismet | × | | × | × |
| Ethereal | × | | × | × |
| Airtraf | × | | × | × |
| Mognet | × | | × | × |
| Hcidump | | × | | |
| FTE4BT | | × | | × |

Es importante mencionar que como la aplicación *sniffer* se desarrolló con el lenguaje de programación Java, ésta puede ejecutarse en cualquier sistema operativo.

## 4.3 Descripción funcional

La aplicación *sniffer* cuenta con las funciones de: captura de paquetes en tiempo real, cargado de paquetes por medio de archivos y visualización del contenido de los paquetes. Estas funciones se explican brevemente a continuación.

- *Captura de paquetes en tiempo real*. Las sesiones de captura de paquetes pueden ser creadas para las comunicaciones 802.11 o Bluetooth, y son ejecutas en tiempo real. En cada sesión se desplegará una lista de todos los paquetes capturados, los cuales podrán ser almacenados en un archivo de forma opcional.
- *Cargado de paquetes por medio de archivos*. Los paquetes pertenecientes a alguna sesión de captura y que se encuentran almacenados en un archivo podrán ser cargados y visualizados en una lista de paquetes. Cuando la sesión de captura haya sido cargada, los paquetes podrán ser almacenados en otro archivo con un nombre o una localización diferente.
- *Visualización del contenido de los paquetes*. Para visualizar el contenido de algún paquete, este se seleccionará de una lista que debió de haber sido creada por una sesión de captura en tiempo real o por la carga de un archivo. Después de haber sido seleccionado el paquete, su contenido se visualizará en tres formatos: ASCII, Hexadecimal y por protocolos. En este último formato, se mostrarán los protocolos que conforman al paquete de acuerdo a su tipo. Para esto, sólo se tomaron en cuenta los protocolos más comunes en las redes 802.11 y Bluetooth.

En la figura 4.1 se muestra el diagrama representativo del funcionamiento de la aplicación *sniffer*.


Figura 4.1 – Diagrama de la aplicación *sniffer*

## 4.4 Restricciones

Las restricciones de la aplicación *sniffer* caen en el soporte que tiene para la captura de paquetes en las comunicaciones Bluetooth. Estas se derivan principalmente de los adaptadores de red y del software que brinda el soporte para manejar Bluetooth.

Con los adaptadores de red, la falta de soporte para trabajar en el modo de monitoreo ha originado que la aplicación *sniffer* sólo pueda capturar paquetes de las conexiones que hayan sido creadas entre el sistema local y otros dispositivos Bluetooth. Esto también origina que los paquetes capturados no transporten datos pertenecientes a los protocolos de Bandabase y LMP.

Por otra parte, el software que brinda el soporte Bluetooth en las computadoras es muy específico del sistema operativo. Es por esto que para llevar a cabo la captura de paquetes Bluetooth en tiempo real, sólo se puede realizar con las librerías BlueZ que fueron diseñadas para trabajar sobre el sistema operativo Linux[20].

## 4.5 Requerimientos de la aplicación

La computadora de escritorio o portátil deberá contener los requerimientos que se muestran en la tabla 4.2.

[20] BlueZ es soportado por Linux a partir del kernel 2.4.x

Tabla 4.2 – Requerimientos de la aplicación *sniffer*

| Hardware | Software |
|---|---|
| Procesador equivalente a Pentium II (Probado con 475MHz) | Java JRE 1.4.x o superior |
| Memoria RAM de 64 MB como mínimo, 128MB recomendado | Librerías libpcap para sistemas basados en Unix o winpcap para Windows |
| Capacidad en disco duro de 50MB | Librerías BlueZ |
| Adaptador de red Bluetooth versión >= 1.1 | |
| Adaptador de red 802.11 con soporte para trabajar en modo de monitoreo | |

## 4.6 Especificación de los casos de uso

En la figura 4.2 se muestra el diagrama de casos de uso de la aplicación *sniffer*. En las secciones siguientes se explicarán los casos de uso involucrados utilizando el formato de casos de uso para RUP y los diagramas de secuencia.

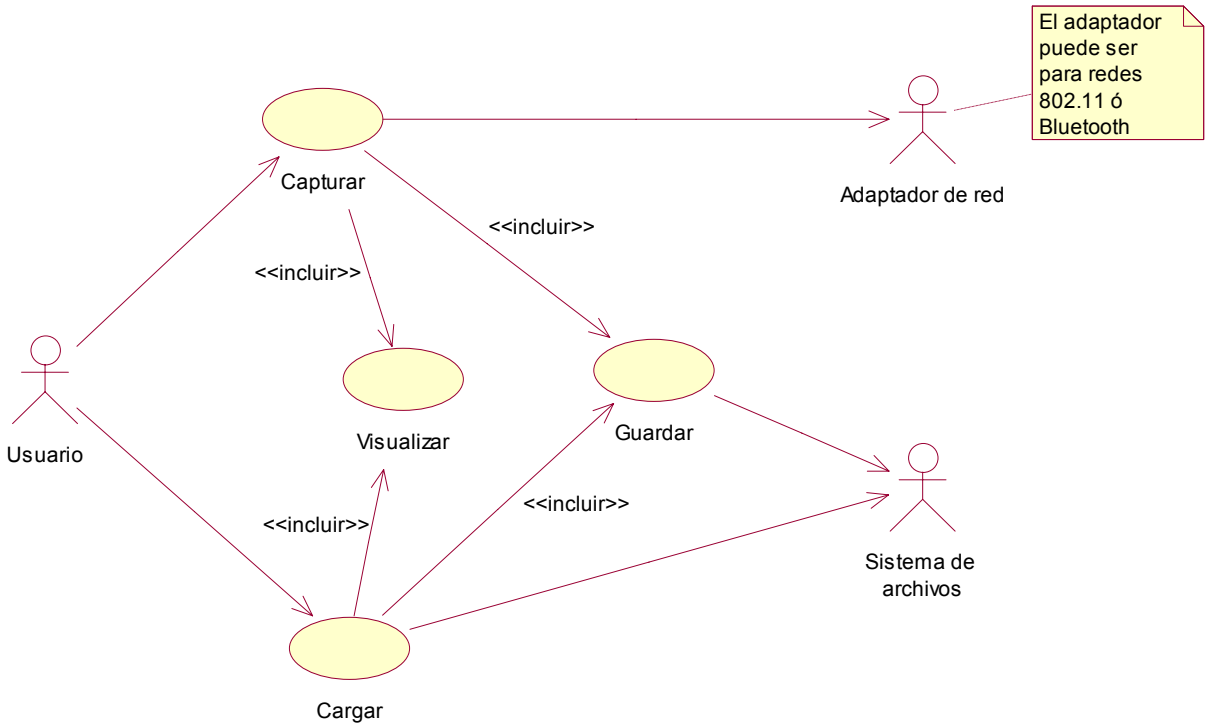


Figura 4.2 – Diagrama de casos de uso de la aplicación *sniffer*

### 4.6.1 Caso de uso: Capturar

Este caso de uso permite a un usuario crear una sesión de captura de paquetes en tiempo real con el apoyo de un adaptador para redes 802.11 ó Bluetooth. Los actores que interactúan en este caso de uso son el usuario y el adaptador de red. El usuario es el actor principal y el adaptador de red es un actor de soporte usado por el caso de uso. En la figura 4.3 se muestra el diagrama de secuencia correspondiente.

- **Flujo básico de eventos**

  1. El usuario inicia la sesión de captura de paquetes.
  2. El usuario indica el tipo de captura (802.11 ó Bluetooth) y el adaptador de red con el que se capturarán los paquetes.
  3. Mientras haya tráfico en la red, la aplicación *sniffer* comienza a recibir los paquetes del adaptador de red y los despliega en una lista.
  4. En cualquier momento, el usuario le indica a la aplicación terminar la sesión de captura de paquetes.

- **Flujos Alternativos**

  2.a Tráfico. La aplicación no podrá capturar paquetes si no existe tráfico en la red.

- **Requerimientos especiales**

  - Plataforma. La captura de paquetes en redes Bluetooth sólo se podrá realizar en Linux.
  - Adaptador de red. Los adaptadores deben estar funcionando. En el caso del adaptador para redes 802.11, este debe estar trabajando en modo de monitoreo.

- **Precondiciones**

  No se encontraron para este caso de uso.

- **Poscondiciones**

  - Visualiza. El usuario puede ejecutar el caso de uso visualiza.
  - Guarda. El usuario puede ejecutar el caso de uso guarda.

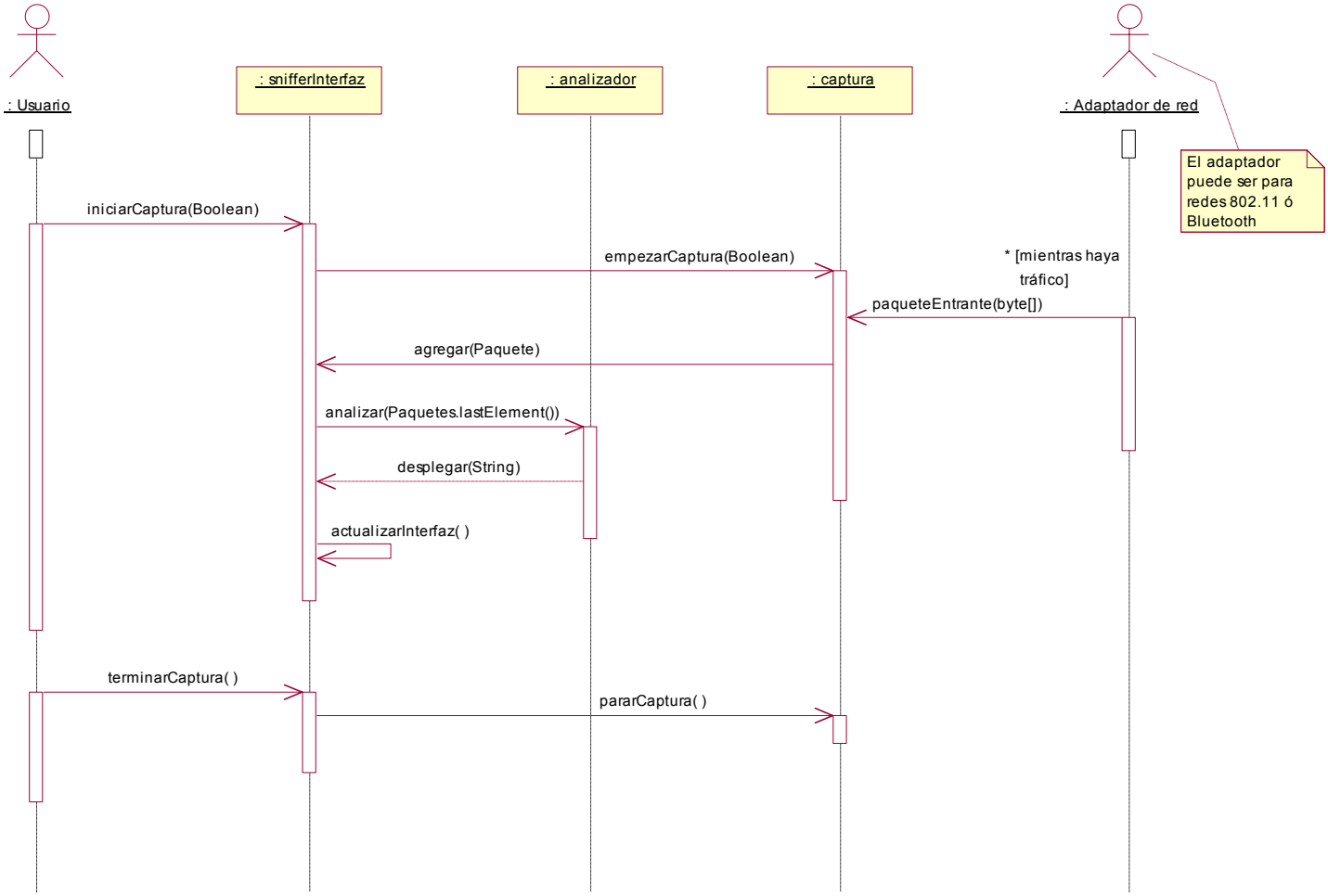


Figura 4.3 – Diagrama de secuencia del caso de uso Capturar

## 4.6.2 Caso de uso: Guardar

Este caso de uso se encarga de almacenar los paquetes mostrados por la aplicación *sniffer* en un archivo. Los actores que interactúan en este caso de uso son el usuario y el sistema de ficheros del sistema operativo. El usuario es el actor principal y el sistema de ficheros es el actor de soporte usado por el caso de uso. En la figura 4.4 se muestra el diagrama de secuencia correspondiente.

- **Flujo básico de eventos**

  1. El usuario desea guardar los paquetes mostrados por la aplicación *sniffer* en un archivo.
  2. El usuario introduce el nombre y la dirección del archivo.
  3. La aplicación *sniffer* creará el archivo con el apoyo del sistema de ficheros.

- **Precondiciones**

  - Sesión de captura. Para que este caso de uso se inicie debe existir una sesión de captura abierta, ya sea que haya sido creada en tiempo real o cargada desde un archivo.

- **Poscondiciones**

  No se encontraron para este caso de uso.

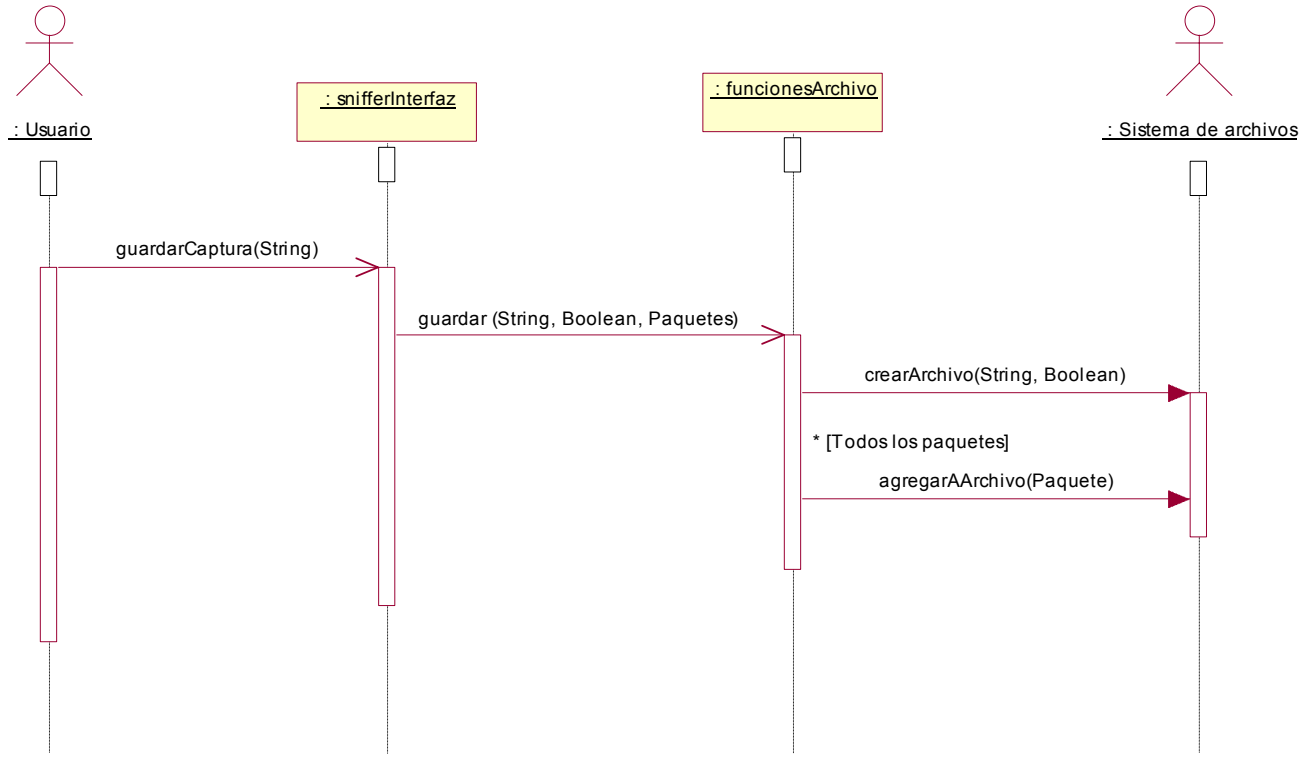


Figura 4.4 – Diagrama de secuencia del caso de uso Guardar

### 4.6.3 Caso de uso: Cargar

Este caso de uso permite a un usuario cargar una sesión de captura de paquetes desde un archivo. Los actores que interactúan en este caso de uso son el usuario y el sistema de ficheros del sistema operativo. El usuario es el actor principal y el sistema de ficheros es un actor de soporte usado por el caso de uso. En la figura 4.5 se muestra el diagrama de secuencia correspondiente.

- **Flujo básico de eventos**

  1. El usuario desea abrir una sesión de captura de paquetes.
  2. El usuario selecciona el archivo en donde se encuentra la sesión.
  3. La aplicación *sniffer* comienza a cargar todos los paquetes del archivo con la ayuda del sistema de ficheros.

- **Precondiciones**

  - Archivo. Para que se inicie este caso de uso debe existir un archivo en el cual haya sido guardado una sesión de captura.

- **Poscondiciones**
  - Visualizar. El usuario puede ejecutar el caso de uso visualizar.
  - Guardar. El usuario puede ejecutar el caso de uso guardar.

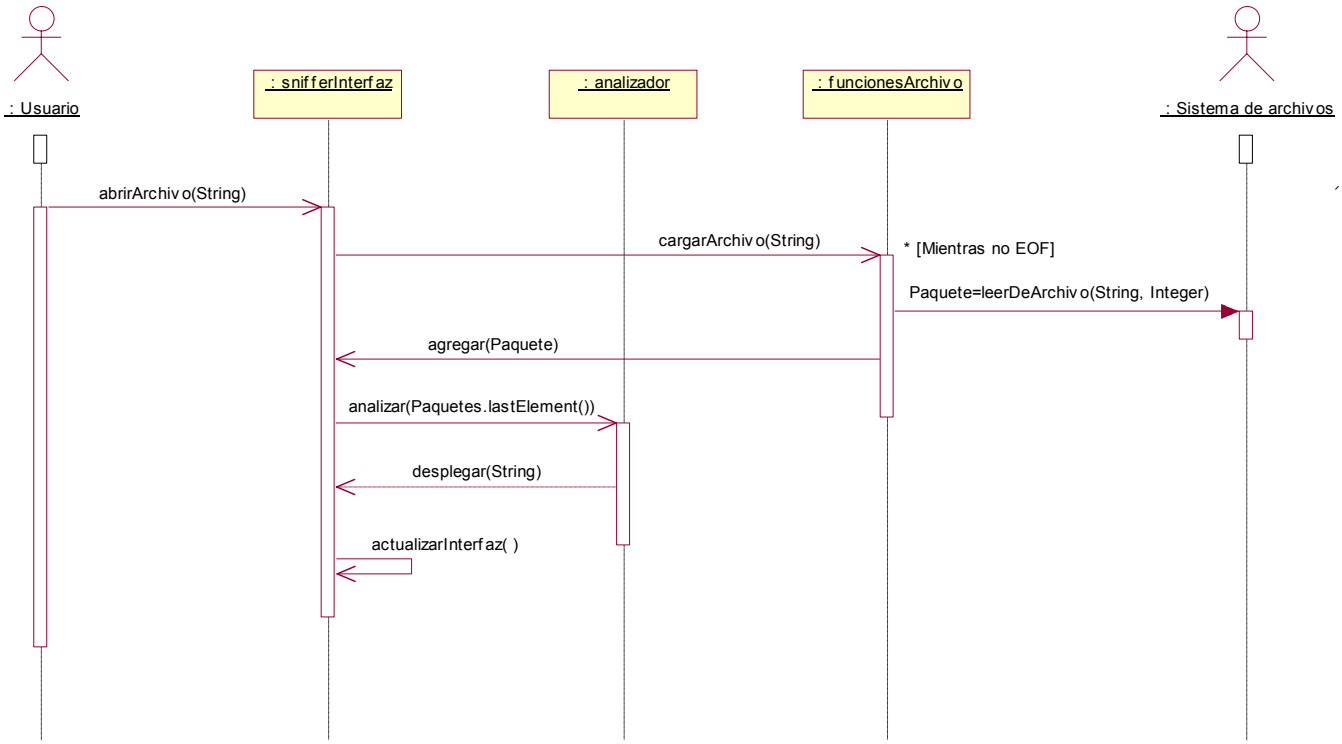


Figura 4.5 – Diagrama de secuencia del caso de uso Cargar

### 4.6.4 Caso de uso: Visualizar

Este caso de uso permite al usuario desplegar el contenido de algún paquete en los formatos ASCII, Hexadecimal y de acuerdo a los protocolos que comprende. El usuario es el único actor que interactúa con el caso de uso. En la figura 4.6 se muestra el diagrama de secuencia correspondiente.

- **Flujo básico de eventos**
  1. El usuario selecciona el paquete que desea visualizar.
  2. La aplicación *sniffer* despliega el contenido del paquete seleccionado byte por byte de acuerdo a los códigos ASCII y Hexadecimal.
  3. La aplicación analiza y despliega el contenido del paquete seleccionado de acuerdo a la estructura de los protocolos manejados en 802.11 y Bluetooth.

- **Precondiciones**
  - Sesión de captura. Para que este caso de uso se inicie debe existir una sesión de captura abierta, ya sea que haya sido creada en tiempo real o cargada desde un archivo.

- **Poscondiciones**
  - Detalles. El usuario podrá interactuar con las diferentes visualizaciones para obtener mayor información del paquete.

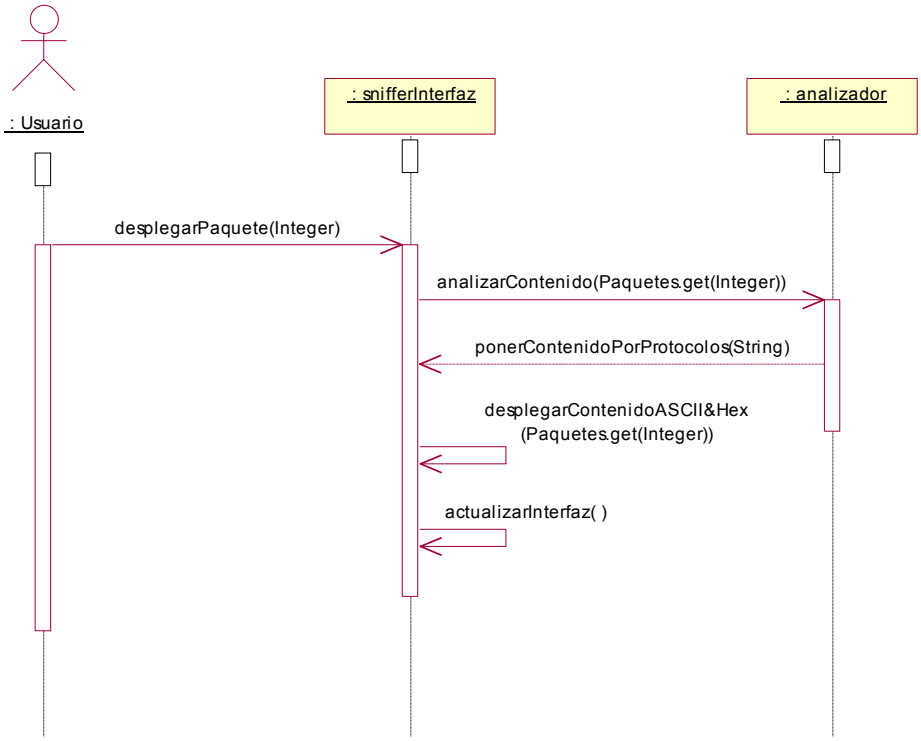


Figura 4.6 – Diagrama de secuencia del caso de uso Visualizar

En la figura 4.7 se presenta el diagrama de clases para la aplicación *sniffer*.

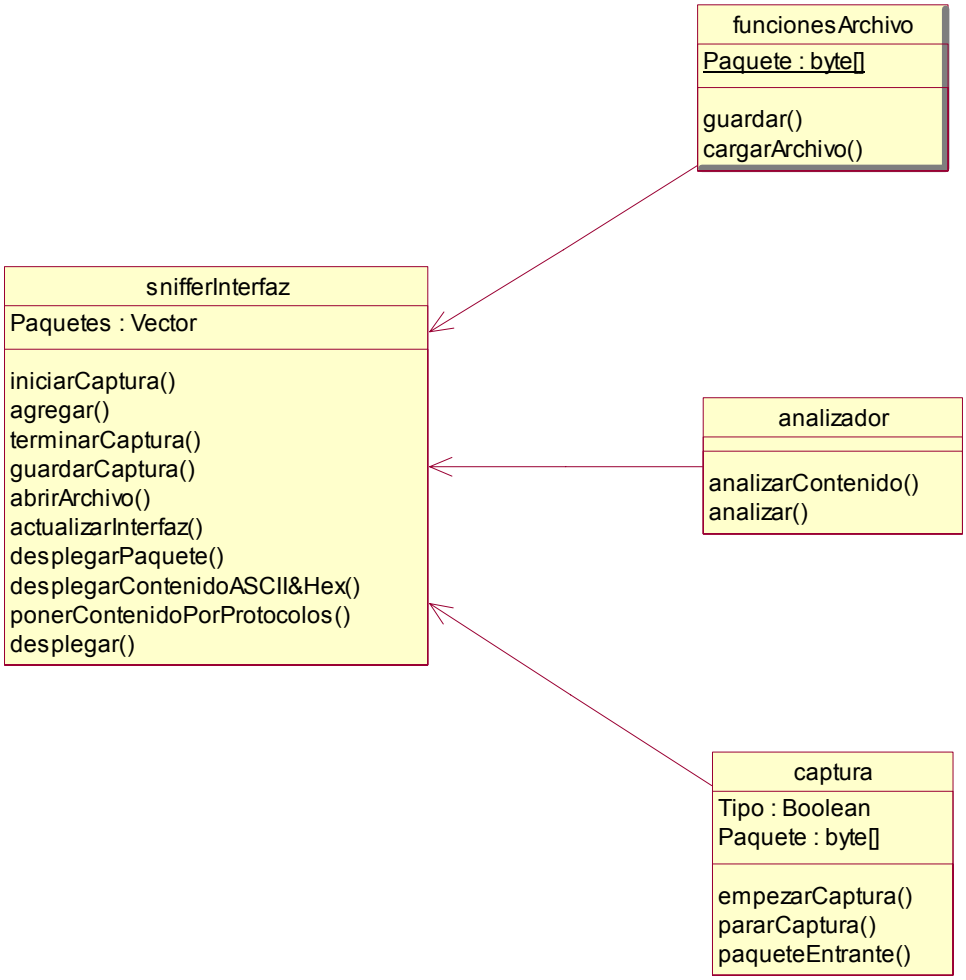


**Figura 4.7 – Diagrama de clases general para la aplicación *sniffer***

# Implementación del *sniffer*

## 5.1 Introducción

El desarrollo de la aplicación *sniffer* inalámbrica se realizó sobre el sistema operativo Linux para obtener un mejor control sobre los dispositivos de red que fueron utilizados. Se eligió la distribución Linux Fedora Core 1 debido al gran soporte que tiene para manejar redes inalámbricas. Principalmente, la versión del kernel que trae incluido en su instalación (kernel 2.4.22-1.2115.nptl) viene configurada para brindar un soporte y una compatibilidad completa con la tecnología Bluetooth. Sin embargo, en otras distribuciones Linux este kernel u otro más actual puede ser instalado para obtener estas características.

El lenguaje de programación Java fue utilizado para la creación de la aplicación *sniffer* empleando la edición estándar de Java 2, J2SE [URL 18]. Con el uso de este lenguaje, la aplicación *sniffer* hereda la capacidad para ejecutarse en cualquier sistema operativo así como la robustez ante las fallas que se puedan presentar.

Los dispositivos de red inalámbricos que se utilizaron fueron el adaptador Lucent Orinoco 802.11b Classic Gold PC Card para la captura de paquetes en las redes 802.11 y el adaptador USB Bluetake BT007X para la captura en las redes Bluetooth.

Para el manejo del adaptador Lucent Orinoco, se utilizó el controlador orinoco-0.13e con la aplicación del parche orinoco-0.13-dragorn-patch.diff para que pueda soportar el modo rfmon. Actualmente existe una versión más reciente de este controlador que no necesita de algún parche para soportar este modo, pero su uso no es muy recomendable debido a su falta de madurez (orinoco-0.15rc2) [URL 19].

En el caso del adaptador USB Bluetake BT007X, las librerías BlueZ se utilizaron para controlar el dispositivo. BlueZ es el *stack* de protocolos oficial de Linux que brinda soporte para manejar Bluetooth en alguna computadora. Estas librerías ya vienen incluidas en la instalación completa de Fedora Core 1 [URL 20].

En las siguientes secciones se explicará la forma en que se realizó la aplicación *sniffer*.

## 5.2 Desarrollo de la aplicación

La función principal de la aplicación *sniffer* se centra en la captura de paquetes en tiempo real, tanto en las redes 802.11 como en las redes Bluetooth.

El soporte para la captura de paquetes 802.11 se logró con el uso de la librería jpcap proporcionada por la OSTG (*Open Source Technology Group*, Grupo de Tecnología de Código Abierto) [URL 21]. Esta librería contiene clases escritas con JNI (*Java Native Interface*, Interfaz Nativa de Java) para encapsular las funcionalidades de captura brindadas por la famosa librería libpcap. Es por esto, que libpcap debe estar previamente instalado en el sistema operativo. En la mayoría de las distribuciones de Linux como Fedora, la librería libpcap ya se encuentra instalada. Por otro lado, gracias a la versión de libpcap que existe para Windows (winpcap), la librería jpcap puede ser utilizada y por lo tanto permite que la aplicación *sniffer* también pueda capturar paquetes sobre este sistema operativo.

Es importante mencionar que para la captura de paquetes 802.11, los adaptadores de red deben de estar funcionando en modo rfmon, aunque no todos los adaptadores tienen soporte para trabajar con

este modo[21]. Actualmente, para el adaptador Lucent Orinoco todavía no se ha encontrado algún controlador que trabaje con este modo bajo Windows, sin embargo este puede ser programado. En Linux, el adaptador Lucent Orinoco soporta el modo rfmon con el controlador orinoco-13e como se mencionó anteriormente y la forma de habilitarlo se lleva a cabo con el siguiente comando.

*$ iwpriv eth0 monitor <m> <c>*
*m – uno de los siguientes*
*0 – deshabilitar el modo rfmon*
*1 – habilitar el modo rfmon con encabezados Prism2 al paquete (ARPHRD_IEEE80211_PRISM)*
*2 – habilitar el modo rfmon sin encabezados Prism2 (ARPHRD_IEEE80211)*
*c – canal a monitorear*

Por otra parte, la captura de paquetes Bluetooth se logró con la creación de una librería que llamamos jhcicap. El desarrollo de esta librería se asemeja a la forma en que fue construida la librería jpcap. Es por esto, que en la construcción de jhcicap se utilizó JNI y algunas clases abstractas de jpcap para encapsular las funciones de captura de paquetes que se programan con el uso de las librerías BlueZ.

Cabe mencionar que para la captura de paquetes Bluetooth, es necesario que con los adaptadores de red se esté participando en una piconet o scatternet ya que los paquetes capturados pertenecen a las comunicaciones de las mismas.

Con esta información, a continuación se describirá la implementación de los módulos que conforman a la aplicación *sniffer*.

## 5.3 Descripción funcional

La aplicación *sniffer* provee las funciones de captura de paquetes en tiempo real, cargado y guardado de paquetes desde archivos y visualización de paquetes. En la figura 5.1 se presenta un diagrama de componentes en donde se muestran estas funciones y la forma en que se encuentran relacionadas.

## 5.4 Módulo de captura

Este módulo se encarga de realizar las sesiones de captura de paquetes en tiempo real y está compuesta por los submódulos de captura de paquetes 802.11 y captura de paquetes Bluetooth. En la clase *Capture* del apéndice A se encuentra el código fuente correspondiente al módulo de captura. En la figura 5.2 se presenta la pantalla en donde se elegirá el tipo de captura.

### 5.4.1 Captura de paquetes 802.11

En este módulo se involucra el paquete *Capture* de jpcap para poder realizar la captura de paquetes 802.11. Para esto, *Capture* proporciona una conjunto de clases que permiten personalizar la captura y con las que se puede establecer el adaptador por donde se recibirán los paquetes. Cabe mencionar que jpcap permite crear hasta 10 sesiones de captura simultáneas con diferentes adaptadores de red. Esta característica es heredada por la aplicación *sniffer*. El proceso de captura se describe brevemente con el siguiente algoritmo.

1. Incluir las clases que se emplean
   *import net.sourceforge.jpcap.capture.*;*

[21]En www.kismetwireless.net/cards.shtml se pueden encontrar algunos adaptadores que soportan el modo rfmon.

*import net.sourceforge.jpcap.net.RawPacket;*

2. Crear un una sesión de captura

   *class captureBox implements RawPacketListener{}*
   *Vector packets = new Vector();*
   *PacketCapture packetCapture = new PacketCapture();*
   *packetCapture.addRawPacketListener(captureBox);*

3. Seleccionar algún adaptador inalámbrico disponible

   *String devices = packetCapture.lookupDevices();*

4. Abrir el dispositivo seleccionado para la captura de paquetes

   *packetCapture.open(devices[x], true);*

5. Comenzar la captura de paquetes

   *packetCapture.capture(-1);*

6. Obtención de los paquetes uno por uno

   *captureBox.rawPacketArrived( RawPacket packet ) { packets.add(packet); }*

7. Detener y liberar los recursos de la captura de paquetes

   *packetCapture.endCapture();*
   *packetCapture.stop();*

En la figura 5.3 se muestra la implementación del algoritmo anterior.

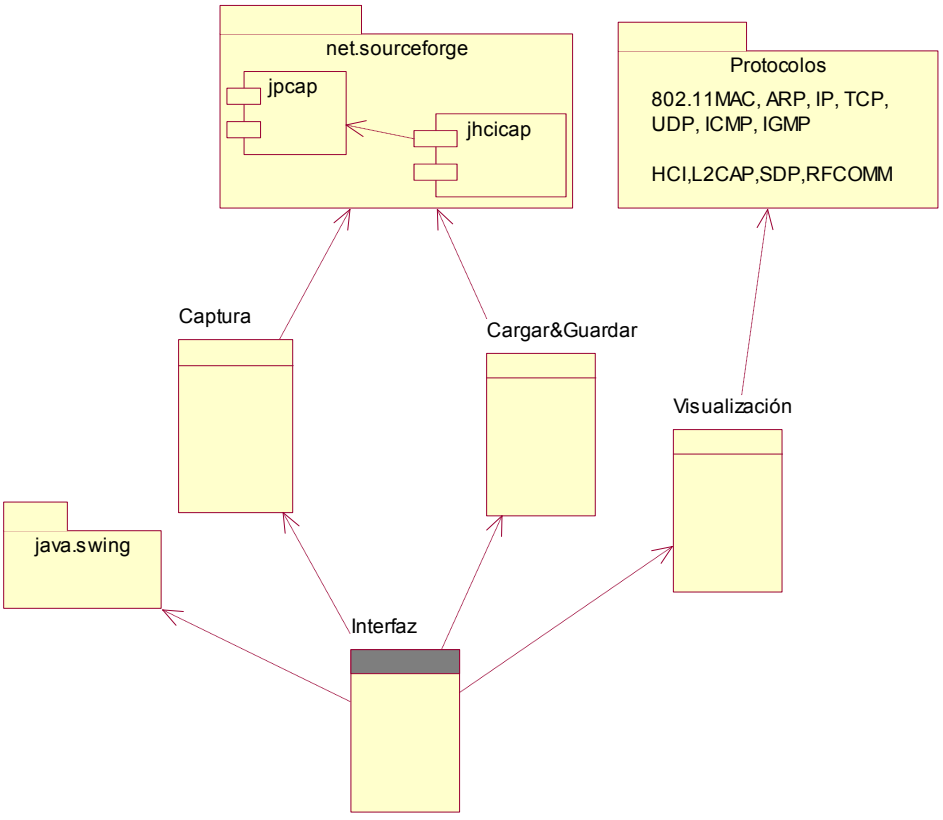


Figura 5.1 – Módulos de la aplicación *sniffer*


Figura 5.2 – Módulo de captura

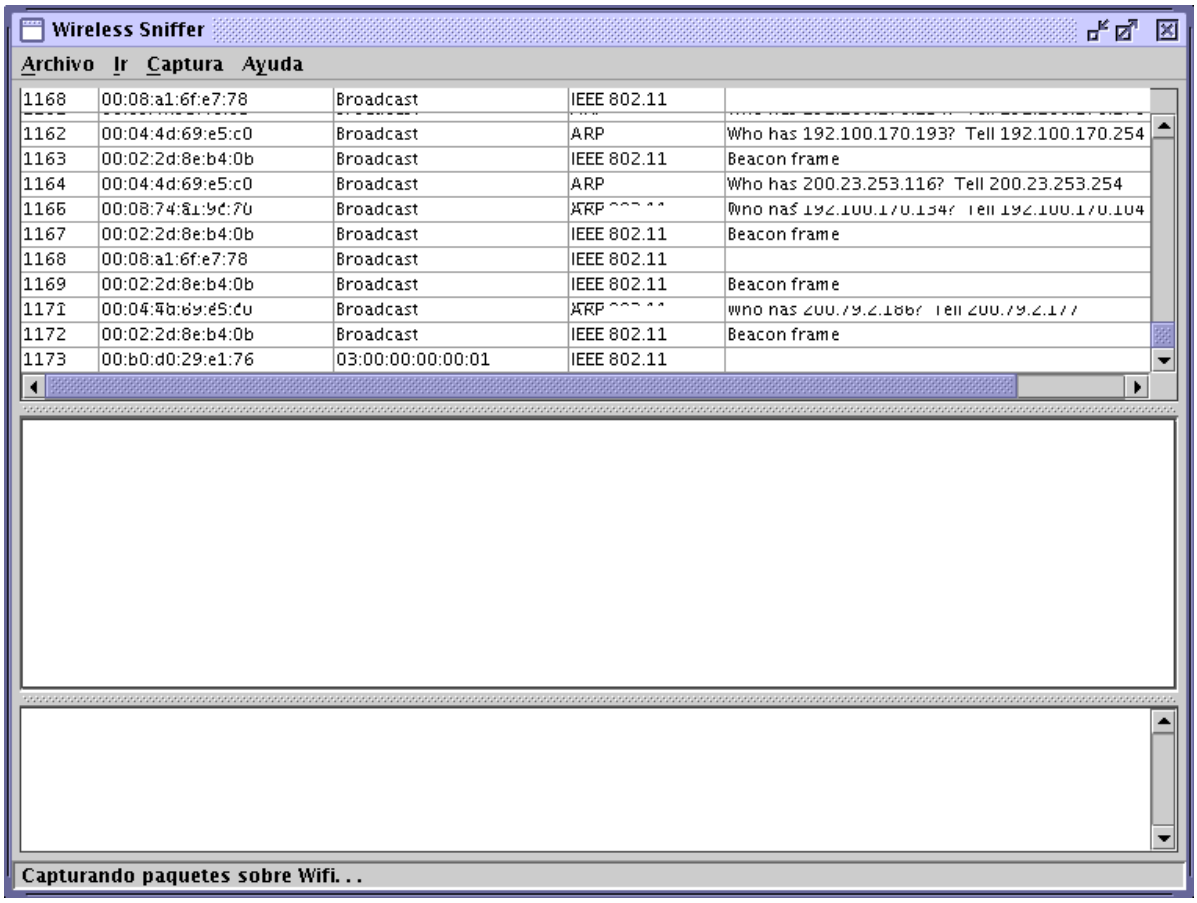


Figura 5.3 – Módulo de captura de paquetes 802.11

### 5.4.2 Captura de paquetes Bluetooth

En este módulo se involucra el paquete *Capture* de jpcap y la clase *PacketCapture* de jhcicap para poder realizar la captura de paquetes Bluetooth. Para esto, *PacketCapture* permite personalizar la captura y establecer el adaptador por donde se recibirán los paquetes. Algunas clases del paquete *Capture* de jpcap fueron reutilizadas. Cabe mencionar que jhcicap, al igual que la captura de paquetes 802.11, permite crear 10 sesiones de captura simultáneas con diferentes adaptadores de red. El proceso de captura se describe brevemente con el siguiente algoritmo.

1. Incluir las clases que se emplean
   *import net.sourceforge.jpcap.capture.*;*
   *import net.sourceforge.jhcicap.net.RawPacket;*
2. Crear un una sesión de captura
   *class captureBox implements RawPacketListener{}*
   *Vector packets = new Vector();*
   *packetCapture = new net.sourceforge.jhcicap.capture.PacketCapture();*
   *packetCapture.addRawPacketListener(captureBox);*
3. Seleccionar algún adaptador inalámbrico disponible
   *String devices = packetCapture.lookupDevices();*
4. Abrir el dispositivo seleccionado para la captura de paquetes
   *packetCapture.open(devices[x], true);*
5. Comenzar la captura de paquetes
   *packetCapture.capture(-1);*
6. Obtención de los paquetes uno por uno
   *captureBox.rawPacketArrived( RawPacket packet ) { packets.add(packet); }*
7. Detener y liberar los recursos de la captura de paquetes
   *packetCapture.endCapture();*
   *packetCapture.stop();*

En la figura 5.4 se muestra la implementación del algoritmo anterior.

## 5.5 Módulos de guardado y cargado

Estos módulos son los encargados de manipular las sesiones de captura almacenadas en archivos con extensión *.wcs* (*wireless capture session*, sesión de captura inalámbrica). Esta extensión se eligió para identificar a los archivos manejados por la aplicación *sniffer*.

El módulo de guardado crea los archivos *.wcs* para almacenar los paquetes 802.11 ó Bluetooth que fueron obtenidos de una sesión de captura realizada en tiempo real o cargada desde otro archivo *.wcs*. Por otro lado, el módulo de cargado se encarga de extraer estos paquetes desde un archivo previamente creado.

Los archivos *.wcs* son creados de acuerdo a los formatos tcpdump (formato libpcap) y hcidump. Las sesiones de captura 802.11 se almacenan en archivos con formato tcpdump y las sesiones de captura Bluetooth en archivos con formato hcidump. Por lo tanto, otros *sniffers* podrán leer las sesiones de captura que se encuentran almacenadas en los archivos creados por la aplicación *sniffer* y viceversa.

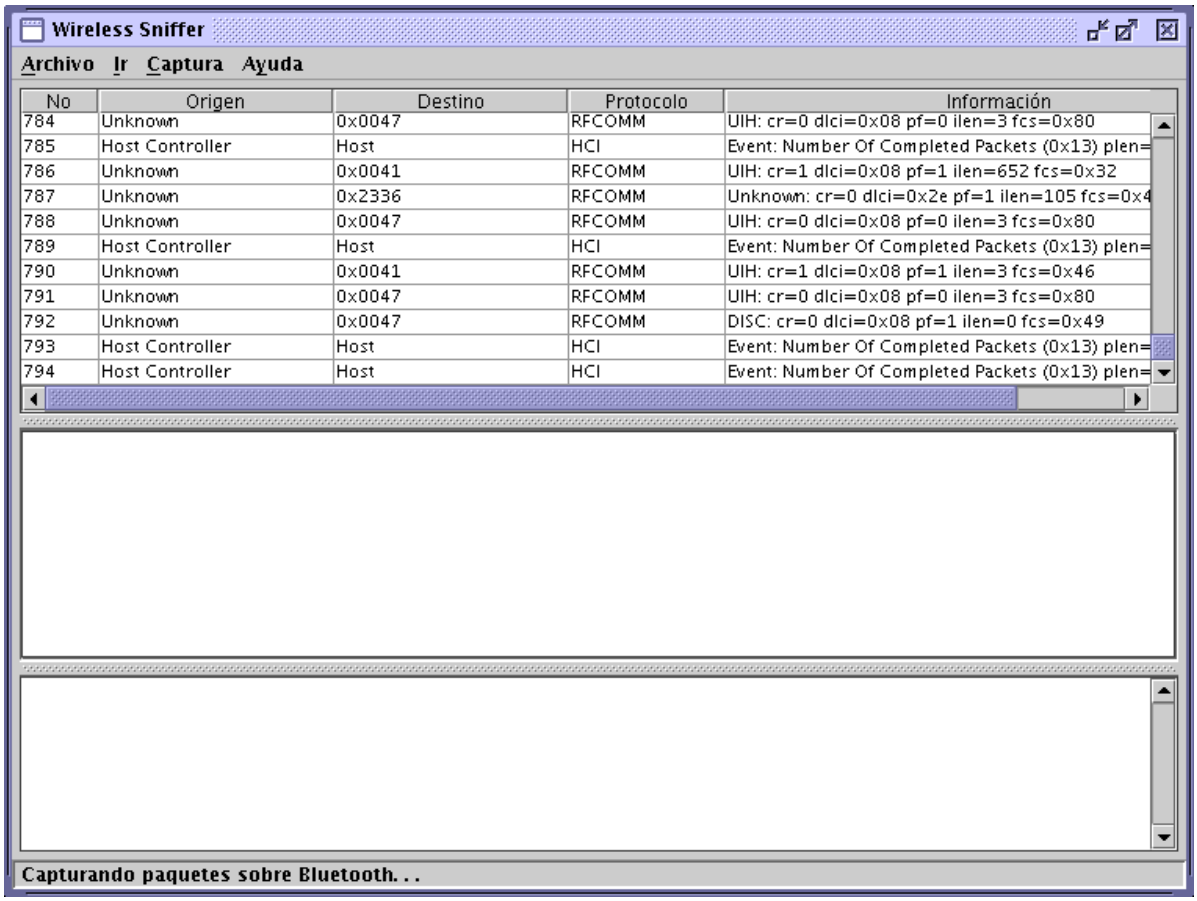


Figura 5.4 – Módulo de captura de paquetes Bluetooth

Como se puede observar en los algoritmos de captura (secciones 5.4.1 y 5.4.2), los paquetes capturados se almacenan en un objeto llamado *packets* de tipo *Vector*. Este objeto es también utilizado por el módulo de cargado para almacenar los paquetes que hayan sido extraídos de algún archivo y por el módulo de guardado para almacenarlos en un archivo. El proceso de cargado se describe brevemente con el siguiente algoritmo.

1. Incluir las clases que se emplean

   *import net.sourceforge.jpcap.capture.*;*
   *// import net.sourceforge.jhcicap.net.RawPacket para archivos hcidump*
   *import net.sourceforge.jpcap.net.RawPacket;*

2. Crear un una sesión de captura

   *class captureBox implements RawPacketListener{}*
   *Vector packets = new Vector();*
   *// new net.sourceforge.jhcicap.capture.PacketCapture() para archivos hcidump*
   *packetCapture = new PacketCapture();*
   *packetCapture.addRawPacketListener(captureBox);*

3. Abrir el archivo que contiene los paquetes

   *packetCapture.openoffline(nameFile);*

4. Comenzar a extraer los paquetes

   *packetCapture.capture(-1);*

5. Obtención de los paquetes uno por uno
   *captureBox.rawPacketArrived( RawPacket packet ) { packets.add(packet); }*
6. Liberar los recursos de la sesión de captura
   *packetCapture.stop();*

En la figura 5.5 se muestra la implementación del algoritmo anterior.

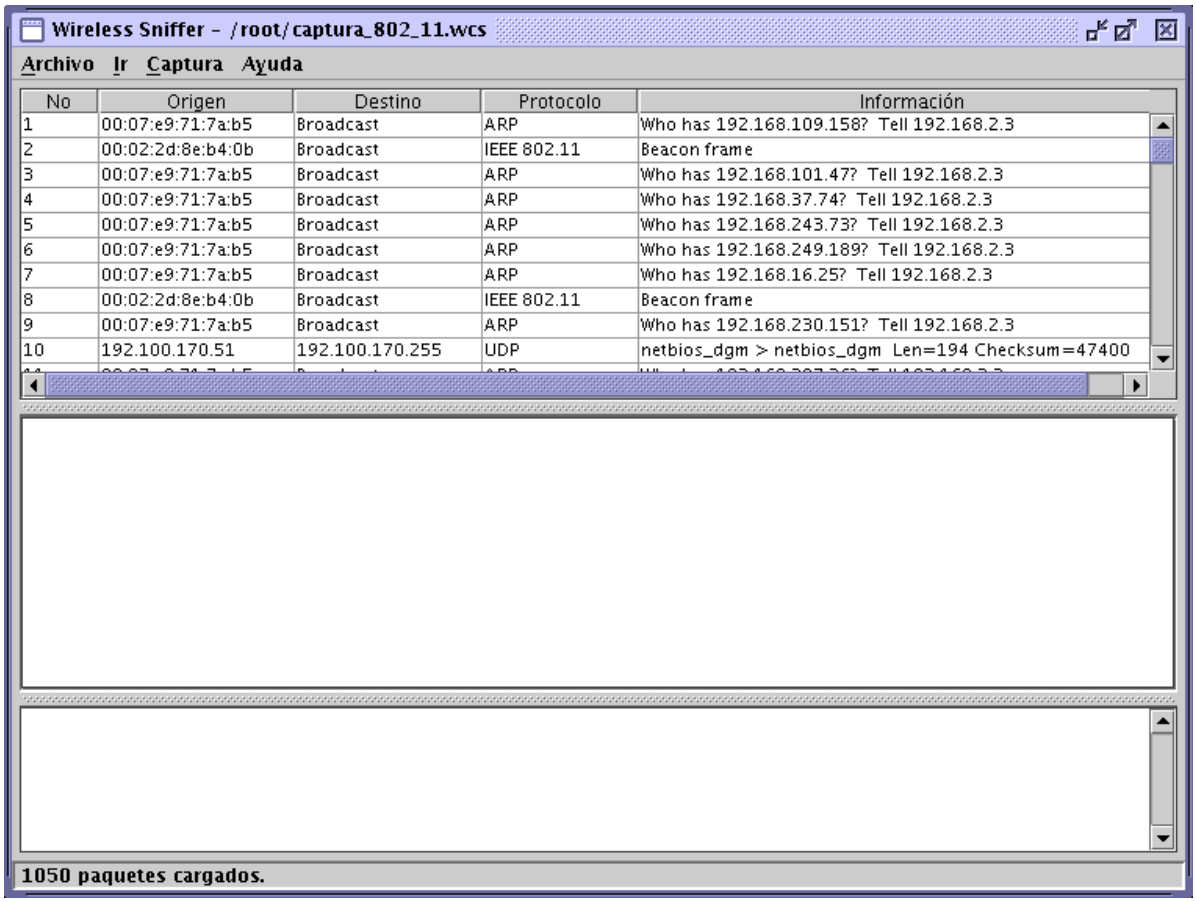


Figura 5.5 – Módulo de cargado

El proceso de guardado se basa en los formatos tcpdump y hcidump, y para su implementación se utilizaron las clases *TcpdumpWriter* y *HcidumpWriter* que están comprendidas en las librerías jpcap y jhcicap respectivamente. En la clase *FileFunctions* del apéndice A se encuentra el código fuente correspondiente a los módulos de guardado y cargado. En la figura 5.6 se presenta la implementación de estos módulos.

## 5.6 Módulo de visualización

Este módulo se encarga de desplegar el contenido de algún paquete seleccionado por el usuario en tres formatos: ASCII, Hexadecimal y por análisis de protocolos. La interfaz manejada para la visualización de los paquetes se diseñó de forma similar a la utilizada por el analizador de paquetes ethereal debido a su sencillez y a la familiaridad que existe con los usuarios [URL 3]. En la figura 5.7 se presenta el módulo de visualización.

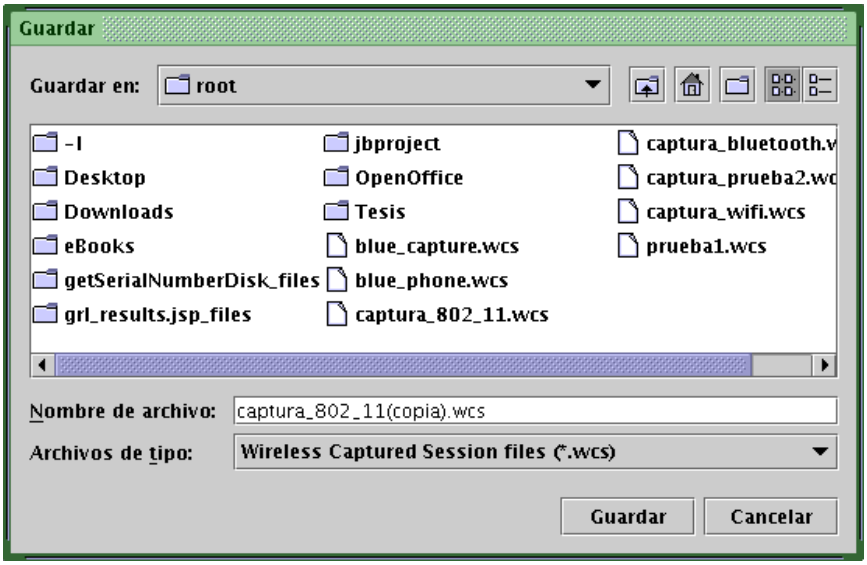


Figura 5.6 – Módulo de guardado

Como se muestra en la figura, los formatos ASCII y Hexadecimal se presentan juntos en el marco de abajo y se encuentran divididos por octetos de bytes para facilitar la comprensión visual por parte del usuario.

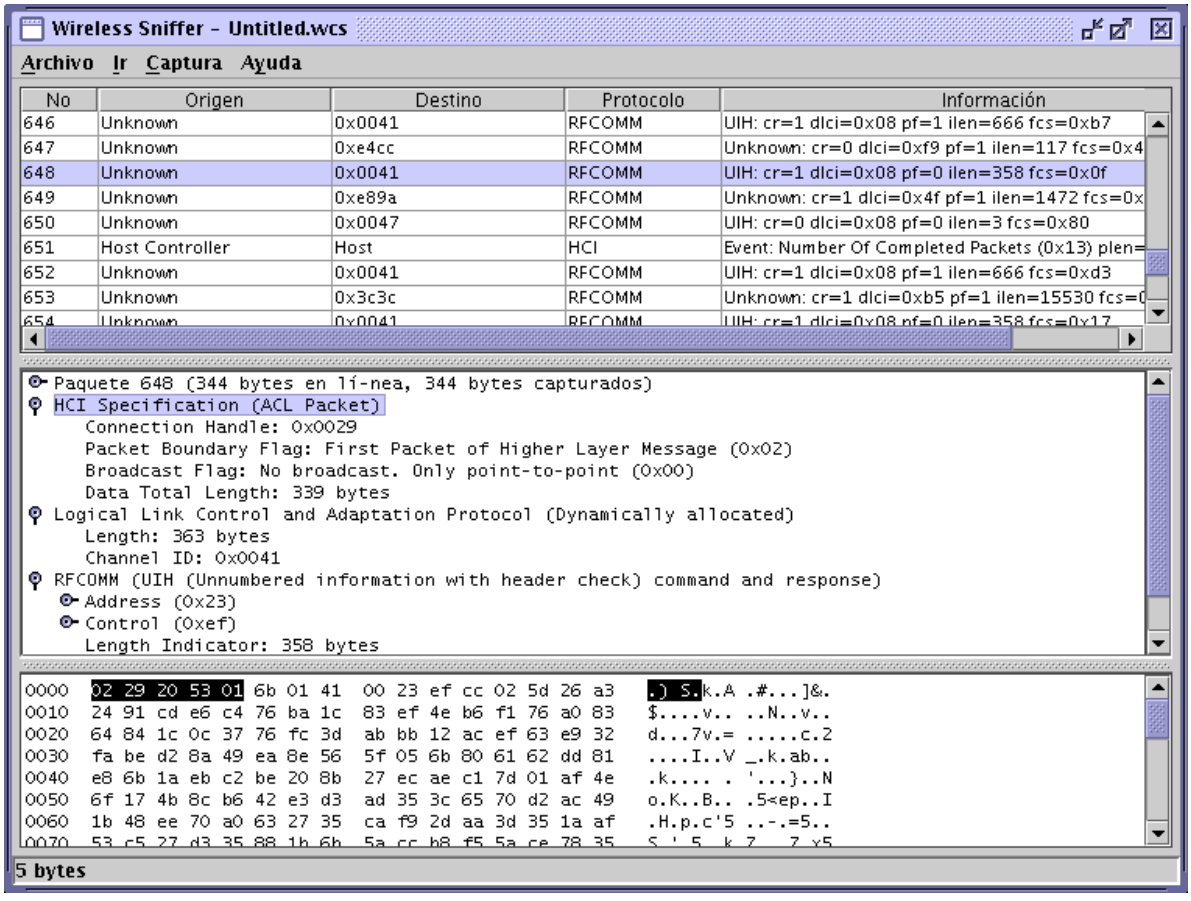


Figura 5.7 – Módulo de visualización

Por otro lado, el formato por análisis de protocolos se encuentra en el marco de en medio y está representado por los encabezados de los protocolos más comunes en las redes Bluetooth. Los encabezados se pueden expandir para observar con detalle los campos que incluyen. En esta versión de la aplicación *sniffer* sólo se tomaron en cuenta los protocolos HCI, L2CAP, SDP y RFCOMM para los paquetes Bluetooth, y 802.11 MAC, ARP, IP, TCP, UDP, ICMP y IGMP para los paquetes 802.11. El módulo de visualización fue diseñado para facilitar la agregación de más protocolos en próximas versiones.

En la clase *Parser* del apéndice A se encuentra el código fuente correspondiente al módulo de visualización.

# Conclusiones y Trabajo futuro

## 6.1 Conclusiones

La aplicación *sniffer* cumplió satisfactoriamente con el objetivo propuesto, ya que permite la captura del tráfico de una red 802.11 así como la de una red Bluetooth. Cada paquete capturado es analizado para desplegar la estructura de su contenido de acuerdo a los protocolos que lo conforman, proporcionando además de vistas auxiliares que representan este contenido en código ASCII y Hexadecimal. También, la aplicación tiene la capacidad para almacenar los paquetes capturados en archivos con formatos Tcpdump y Hcidump, los cuales pueden ser reutilizados en cualquier momento o por otra aplicación[22].

En el desarrollo de la aplicación *sniffer* se crearon unas clases Java (Jhcicap) que permiten la captura de paquetes Bluetooth y las cuales se aportan a la comunidad programadora para su uso o para la modificación de sus funcionalidades.

La aplicación *sniffer* puede ser utilizada por alguna persona quien deseé conocer y comprender la estructura y el funcionamiento de las tecnologías inalámbricas. Asimismo, puede ser usada por algún desarrollador de software o hardware para detectar y corregir los problemas que puedan presentar sus aplicaciones o proyectos inalámbricos.

En caso de que algún usuario utilice la aplicación *sniffer* para atentar la confidencialidad de la información perteneciente a una red "husmeando" sus transmisiones inalámbricas, éste tiene la capacidad de hacerlo en redes 802.11 y para lograrlo en redes Bluetooth debe ser un usuario válido que pertenezca a la misma piconet o scatternet. Esto debido a la necesidad del uso de un controlador que soporte el modo rfmon, algo que actualmente no se ha encontrado al alcance del público sobre ningún sistema operativo. Al no contar con el controlador para el modo rfmon, repercute en el tráfico capturado sobre las redes Bluetooth, porque el contenido de estos paquetes no incluye información referente a los protocolos de las capas más bajas como el protocolo de Bandabase y LMP, y por lo tanto, la aplicación *sniffer* únicamente se limita a analizar y desplegar los protocolos de capas superiores a LMP (HCI, L2CAP, SDP, RFCOMM, etc.). Por otra parte, en el caso de que se haya construido un controlador con soporte rfmon para algún adaptador Bluetooth, el único inconveniente sería que los paquetes capturados no contendrían información referente al protocolo HCI.

Existen algunas restricciones inherentes a la aplicación que impiden que ésta no se ejecute en su totalidad sobre otra plataforma distinta a Linux. Respecto a la captura de paquetes 802.11, la mayoría de los controladores para los adaptadores de red con soporte rfmon fueron diseñados para trabajar sobre Linux y sólo algunos trabajan sobre OS X, FreeBSD y otros sistemas basados en Unix. Sobre Windows, no se han encontrado controladores públicos ni reportes de que estos se pretendan construir. Por otro lado, respecto a Bluetooth, el uso de las librerías BlueZ inducen a que la aplicación *sniffer* sólo permita capturar paquetes sobre Linux, debido a que su creación fue proyectada para trabajar sobre este sistema operativo exclusivamente.

Otras de las conclusiones a la cual se llegó, es que con el uso de un adaptador de red con soporte para manejar el modo rfmon no se pueden detectar los ataques *sniffing*, y la mejor forma de protegerse de este ataque es con el uso de alguna herramienta de encriptación que se desempeñe en capas superiores a la capa de enlace.

[22] La aplicación sniffer junto con las librerías utilizadas en su desarrollo se pueden descargar de la página web http://mixtli.utm.mx/~resdi/html/docente_gabriel.html

## 6.2 Trabajo futuro

Dependiendo de las necesidades del usuario, la aplicación *sniffer* puede requerir de herramientas secundarias para obtener más información respecto al tráfico capturado o a la red inalámbrica. Estas herramientas pueden proporcionar información desde estadísticas y gráficas del tráfico hasta métodos para descifrar las claves de encriptación de la red. Muchas de estas herramientas ya han sido desarrolladas y podrían ser añadidas a la aplicación, pero si se desea obtener una mayor compatibilidad y rendimiento es necesario crearlas.

Las herramientas anteriores pueden ser fácilmente implementadas en la aplicación *sniffer* debido a la estructura modular con la que fue programada. Esto también permite otras características como la agregación de análisis de nuevos protocolos de red a la aplicación.

Definitivamente algo que incrementaría el potencial de la aplicación *sniffer* en lo concerniente a Bluetooth sería la construcción de controladores con soporte rfmon para los adaptadores de red, así como el uso o la construcción de librerías que permitan la captura de paquetes sobre otros sistemas operativos para que de esta forma se logre eliminar la exclusividad que BlueZ le proporciona a Linux. De la misma manera, también se podrían crear controladores con soporte rfmon para los adaptadores de red 802.11 sobre Windows.

Algunos trabajos que se pueden desarrollar a partir de la aplicación *sniffer* implementada consisten principalmente en conocer las estadísticas del tráfico de una red 802.11 en los distintos canales basándose en diagramas de utilización y la determinación del espectro de frecuencias que maneja la tecnología 802.11. Ambos trabajos pueden explotar características particulares de la aplicación *sniffer* como la capacidad que tiene para ejecutarse con 10 adaptadores de red diferentes al mismo tiempo en una computadora[23].

[23] El número de adaptadores de red soportados puede ser modificado desde el código fuente.

# Bibliografía


[1] Ramos González, Mara Ivett. *Analizador y visualizador de paquetes ICMP, ARP y RARP*. Universidad Tecnológica de la Mixteca. Octubre del 2001.

[2] Hernándex Sánchez, Carlos Hiram. *Ipv6: El nuevo protocolo de Internet, instalación, configuración, pruebas y análisis de paquetes en Linux*. Universidad Tecnológica de la Mixteca. Abril del 2002.

[3] Nichols, K. Randall y Lekkas, C. Panos.*Wireless Security: Models, Threats and Solutions*. McGraw Hill, 2002.

[4] LAN MAN Standars committee of the IEEE Computer Society. *Part 11:Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. ANSI/IEEE Std. 802.11, 1999 Edition.

[5] Träskbäck, Marjaana. *Security of Bluetooth: An overview of Bluetooth Security*. Department of Electrical Communications Engineering.

[6] Ford Long, Wong. *Overwiew of Bluetooth Security*. SIG$^2$ (Special Interest Group). 29 de marzo del 2003.

[7] Maxim, Merritt y Pollino, David. *Wireless Security*. MacGraw Hill, 2002.

[8] Barner, Bautts, Lloyd, Ouellet, Posluns, Zendzian. *Hack proffing your wireless network*. Syngress Publishing, 2002.

[9] Bluetooth SIG. *Specification of the Bluetooth System.* Volume 1. Version 1.1 February 22 2001.

[10] ETSI. *"TS 101 369 V6.3.0 (1999-03) (GSM 07.10 version 6.3.0 Release 1997)"*. ETSI. 1999.

[11] Fowler, Martin. *UML Distilled*. Addison Wesley. Third Edition. 15 de septiembre del 2003.


# URL'S


[URL 1] Kismet . *http://www.kismetwireless.net* . Último acceso: 10 de diciembre del 2003.

[URL 2] SourceForge Team. *http://sourceforge.net/projects/airtraf* . Último acceso: 10 de diciembre del 2003.

[URL 3] Ethereal Analyzer. *http://www.ethereal.com* . Último acceso: 15 de diciembre del 2003.

[URL 4] Node 99 Organization. *http://node99.org/projects/mognet/* . Último acceso: 20 de diciembre del 2003.

[URL 5] BlueZ Project, *http://www.bluez.org/*. Último acceso: 30 de agosto del 2004.

[URL 6] Baseband Technologies. *http://www.linkferret.ws* . Último acceso: 15 de diciembre del 2003.

[URL 7] Network Associates Technology, Inc. *http://www.networkassociates.com/us/products/sniffer/wireless/sniffer_wireless.htm*. Último acceso: 15 de diciembre del 2003.

[URL 8] WildPackets, Inc. *http://www.wildpackets.com/products/airopeek* . Último acceso: 14 de diciembre del 2003.

[URL 9] Frontline Test Equipment Inc. *http://www.fte.com*. Último acceso: 24 de Enero del 2005.

[URL 10] LeCroy Corporation. *http://www.catc.com* . Último acceso: 24 de Enero del 2005.

[URL 11] Sun Microsystems. *http://jcp.org/en/jsr/detail?id=082* . Último acceso: 5 de enero del 2004.

[URL 12] The Shmoo Group. *http://airsnort.shmoo.com/*. Último acceso: 7 de septiembre del 2004.

[URL 13] OSTG Open Source Technology Group. *http://wepcrack.sourceforge.net/*. Último acceso: 30 de septiembre del 2004.

[URL 14] NetStumbler. *http://www.netstumbler.com/*. Último acceso: 14 de octubre del 2004.

[URL 15] The Shmoo Group. *http://bluesniff.shmoo.com/*. Último acceso: 15 de octubre del 2004.

[URL 16] XP, Extreme Programming. *http://www.extremeprogramming.org/*. Último acceso: 11 de febrero del 2005.

[URL 17] OMG, Object Management Group. *http://www.uml.org/*. Último acceso: 11 de febrero del 2005.

[URL 18] Sun Microsystems. *http://java.sun.com/*. Último acceso: 10 de enero del 2005.

[URL 19] The Shmoo Group. *http://airsnort.shmoo.com/orinocoinfo.html*. Último acceso: 15 de enero del 2005.

[URL 20] Red Hat, Inc. *http://fedora.redhat.com/*. Último acceso: 30 de enero del 2005.

[URL 21] OSTG Open Source Technology Group. *http://jpcap.sourceforge.net/*. Último acceso: 5 de enero del 2005.

# Apéndice A. Código Fuente de las clases básicas

## Clase Captura

```
import net.sourceforge.jpcap.capture.*;
import net.sourceforge.jpcap.net.*;

/**
 * <p>Title: Wireless Sniffer</p>
 * <p>Description: sniffing wifi and bluetooth networks</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: UTM</p>
 * @author Emmanuel Mendoza Acevedo
 * @version 1.0
 */

public class Capture extends Thread implements RawPacketListener
{
  private WifiBlueSniffer Owner;
  private String device;
  private int snaplen;
  private boolean WifiNotBlue;
  private boolean stopped;
  private net.sourceforge.jpcap.capture.PacketCapture wifiSession = null;
  private net.sourceforge.jhcicap.capture.PacketCapture blueSession = null;

  public Capture( WifiBlueSniffer owner, String dev, int snapl,
                  boolean type ) throws CaptureDeviceOpenException {
    device = dev;
    snaplen = snapl;
    WifiNotBlue = type;
    Owner = owner;

    if ( WifiNotBlue )
    {
      wifiSession = new net.sourceforge.jpcap.capture.PacketCapture();
      wifiSession.addRawPacketListener(this);

      wifiSession.open(device, snaplen, true, wifiSession.DEFAULT_TIMEOUT);
    }
    else
    {
      blueSession = new net.sourceforge.jhcicap.capture.PacketCapture();
      blueSession.addRawPacketListener(this);

      blueSession.open(device, snaplen, true, blueSession.DEFAULT_TIMEOUT);
    }
  }

  public void rawPacketArrived( RawPacket packet )
  {
    if ( stopped )
      return;

    Owner.container.setSharedPacket(packet);
  }

  public void start()
```

```
  {
    stopped = false;

    super.start();
  }

  public void finished()
  {
    stopped = true;

    if ( WifiNotBlue )
      wifiSession.endCapture();
    else
      blueSession.endCapture();
  }

  public void run()
  {
    try
    {
      // capturar de forma infinita
      if ( WifiNotBlue ) {
        wifiSession.capture(-1);
        wifiSession.close();
      }
      else {
        blueSession.capture(-1);
        blueSession.close();
      }
    }
    catch(CapturePacketException cpe) {
      Owner.errorMessage( "El dispositivo no estÃ¡ disponible!\nIntenta " +
                          (WifiNotBlue ?"'ifconfig ":"'hciconfig") +
                          device + " up'" );
    }
  }
}
```

## Clase Analizador (Visualizar)

```
import net.sourceforge.jpcap.net.*;
import java.util.*;
import java.text.*;
import javax.swing.tree.*;
import Protocolos.*;

/**
 * <p>Title: Wireless Sniffer</p>
 * <p>Description: sniffing wifi and bluetooth networks</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: UTM</p>
 * @author Emmanuel Mendoza Acevedo
 * @version 1.0
 */

public class Parser {
  private static Hashtable extraInfoPacket = new Hashtable();
```

```
  public static Object[] getGeneralInfoPacket( int numPacket, Vector packetsVector
) {
    Object[] info = new Object[7];
    RawPacket packet = (RawPacket)packetsVector.get(numPacket);
    RawPacket previousPacket = null;
    String time1, time2;
    DecimalFormat decimalFormat = new DecimalFormat("0.000000");
    SimpleDateFormat dateTimeFormat = new SimpleDateFormat("MMM d, yyyy HH:mm:ss");

    // Mostrar la informaciÃ³n general del frame capturado
    info[0] = "Paquete " + (numPacket+1) + " (" +
(packet.getDroplen()+packet.getData().length) + " bytes en lÃÂnea, "
            + packet.getData().length + " bytes capturados)" +
              Utils.setExtraInfo(0,packet.getData().length); //Utils.INFO_FRAME;

    time1 = packet.getTimeval().toString();
    time1 = time1.substring(0,time1.indexOf((int)'s'));
    info[1] = "Tiempo de llegada: " +
dateTimeFormat.format(packet.getTimeval().getDate()) +
time1.substring(time1.indexOf((int)'.')) +
              Utils.setExtraInfo(0,0); //Utils.INFO_ARRIVAL_TIME;

    if ( numPacket > 0 ) {
      previousPacket = (RawPacket)packetsVector.get(numPacket-1);
      time2 = previousPacket.getTimeval().toString();
      time2 = time2.substring(0,time2.indexOf((int)'s'));

      info[2] = "Tiempo transcurrido desde el paquete anterior: " +
          decimalFormat.format(Double.parseDouble(time1) -
Double.parseDouble(time2))
          + " segundos" + Utils.setExtraInfo(0,0); //Utils.INFO_DELTA_TIME;
    } else {
      info[2] = "Tiempo transcurrido desde el paquete anterior: 0.000000 segundos"
+
              Utils.setExtraInfo(0,0); //Utils.INFO_DELTA_TIME;
    }

    if ( numPacket > 0 ) {
      previousPacket = (RawPacket)packetsVector.get(0);
      time2 = previousPacket.getTimeval().toString();
      time2 = time2.substring(0,time2.indexOf((int)'s'));

      info[3] = "Tiempo relativo al primer paquete: " +
          decimalFormat.format(Double.parseDouble(time1) -
Double.parseDouble(time2))
          + " segundos" + Utils.setExtraInfo(0,0); //Utils.INFO_RELATIV_TIME;
    } else {
      info[3] = "Tiempo relativo al primer paquete: 0.000000 segundos" +
              Utils.setExtraInfo(0,0); // + Utils.INFO_RELATIV_TIME;
    }

    info[4] = "NÃºmero de frame: " + (numPacket+1) + Utils.setExtraInfo(0,0);
//Utils.INFO_FRAME_NUMBER;

    info[5] = "Longitud del paquete: " +
(packet.getDroplen()+packet.getData().length) +
        Utils.setExtraInfo(0,0); //Utils.INFO_PACK_LENGTH;
```

```
    info[6] = "Longitud capturada: " + packet.getData().length +
              Utils.setExtraInfo(0,0); //Utils.INFO_CAPT_LENGTH;

    return info;
  }

/////////////// 802.11

  public static String[] getWifiPacketSummary( RawPacket rawPacket ) {
    // ObtenciÃƒÂ³n del frame genÃƒÂ©rico del paquete
    byte[] frame = rawPacket.getData();
    String[] data = new String[4];

    P802_11 wifiPacket = new P802_11(0,frame);

    if ( wifiPacket.getUpperLayerProtocol() != ProtocolIDs.UNKNOWN )
    {
      // Saber si se trata de un paquete IP o ARP
      Packet packet = wifiPacket.dataToPacket();

      if (packet instanceof IPPacket) {
        IPPacket ipPacket = (IPPacket) packet;
        data[0] = ipPacket.getSourceAddress();
        data[1] = ipPacket.getDestinationAddress();

        if (ipPacket instanceof ICMPPacket) {
          data[2] = (String) PIP.getIPProtocol(IPProtocol.ICMP);
          data[3] = PICMP.Information( (ICMPPacket) ipPacket);
        }
        else if (ipPacket instanceof IGMPPacket) {
          data[2] = (String) PIP.getIPProtocol(IPProtocol.IGMP);
          data[3] = PIGMP.Information( (IGMPPacket) ipPacket);
        }
        else if (ipPacket instanceof TCPPacket) {
          data[2] = (String) PIP.getIPProtocol(IPProtocol.TCP);
          data[3] = PTCP.Information( (TCPPacket) ipPacket);
        }
        else if (ipPacket instanceof UDPPacket) {
          data[2] = (String) PIP.getIPProtocol(IPProtocol.UDP);
          data[3] = PUDP.Information( (UDPPacket) ipPacket);
        }
        else {
          data[2] = (String) PIP.getIPProtocol(ipPacket.getIPProtocol());
          data[3] = ipPacket.toString();
        }
      }
      else if (packet instanceof ARPPacket) {
        wifiPacket.getInfoSummaryPacket(data);
        data[2] = (String) PEthernet.getEthernetProtocol(
            EthernetProtocol.ARP);
        data[3] = PARP.Information( (ARPPacket) packet);
      }
      else {
        data[2] = "Unrecognized";
        data[3] = "Unrecognized (Data frame)";
      }
    }
    else
      wifiPacket.getInfoSummaryPacket(data);
```

```
    return data;
  }

  public static boolean displayWifiPacket(int row, Vector packetsVector,
DefaultTreeModel detailInfo) {
    if ( row >= 0)
    {
      RawPacket rawPacket = (RawPacket) packetsVector.get(row);

      if (rawPacket != null) {
        byte[] frame = rawPacket.getData();
        DefaultMutableTreeNode currentInfo = null;
        Vector infoTree = new Vector();
        Object[] temp = null;
        int index = 0;

        infoTree.add("root");

        // Obtener la informaciÃƒÂ³n general del paquete
        infoTree.add(getGeneralInfoPacket(row, packetsVector));

        // Obtener la informaciÃƒÂ³n IEEE 802.11 y
        // Mostrar la informaciÃƒÂ³n del Protocolo IEEE 802.11 del frame capturado
        P802_11 wifiPacket = new P802_11(index,frame);

        temp = wifiPacket.getProtocolInfo();

        for (int i = 0; i < temp.length; i++)
          infoTree.add(temp[i]);

        // Obtener la informaciÃƒÂ³n IP ÃƒÂ³ ARP del paquete
        if ( wifiPacket.getUpperLayerProtocol() != ProtocolIDs.UNKNOWN )
        {
          Packet packet = wifiPacket.dataToPacket();
          index += wifiPacket.getHeaderLength() + P802_11.HEADER_802_2_LLC_LEN;

          if (packet instanceof IPPacket) {
            IPPacket ipPacket = (IPPacket) packet;

            infoTree.add(PIP.getInfoHeader(ipPacket, index));

            if (ipPacket instanceof ICMPPacket) {
              infoTree.add(PICMP.getInfoHeader((ICMPPacket)ipPacket,
wifiPacket.getHeaderLength() + 8 + ipPacket.getIPHeaderLength()));
            }
            else if (ipPacket instanceof IGMPPacket) {
              infoTree.add(PIGMP.getInfoHeader((IGMPPacket)ipPacket, index +
ipPacket.getIPHeaderLength()));
            }
            else if (ipPacket instanceof TCPPacket) {
              TCPPacket tcpPacket = (TCPPacket)ipPacket;

              infoTree.add(PTCP.getInfoHeader(tcpPacket, index +
ipPacket.getIPHeaderLength()
                                              ));

              //if ( tcpPacket.isPsh() || tcpPacket.isUrg() ) {
                if ( tcpPacket.getPayloadDataLength() > 0 )
```

```
                  infoTree.add("Data (" + tcpPacket.getPayloadDataLength() + "
bytes)" +

Utils.setExtraInfo(ipPacket.getIPHeaderLength()+index+tcpPacket.getTCPHeaderLength(
),
                                tcpPacket.getPayloadDataLength() ));
//Utils.DATA_FRAME);
              //}

            }
            else if (ipPacket instanceof UDPPacket) {
              UDPPacket udpPacket = (UDPPacket)ipPacket;

              infoTree.add(PUDP.getInfoHeader(udpPacket, index +
ipPacket.getIPHeaderLength()));
              infoTree.add("Data (" + (udpPacket.getLength()-
UDPPacket.UDP_HEADER_LEN) + " bytes)" +

Utils.setExtraInfo(ipPacket.getIPHeaderLength()+index+UDPPacket.UDP_HEADER_LEN,
                           udpPacket.getLength()-UDPPacket.UDP_HEADER_LEN));
//Utils.DATA_FRAME);
            }
            else {
              infoTree.add("Data (" + ipPacket.getIPData().length + " bytes)" +
                           Utils.setExtraInfo(ipPacket.getIPHeaderLength()+index,
                           ipPacket.getIPData().length)); //Utils.DATA_FRAME);
            }
          }
          else if (packet instanceof ARPPacket) {
            infoTree.add(PARP.getInfoHeader( (ARPPacket) packet, index));
          }
        }

        detailInfo.setRoot( (TreeNode)
Utilities.FieldOfFrame.processHierarchy(infoTree.toArray()));

        return true;
      }
    }

    return false;
  }

////////////// Bluetooth
  public static Object[] getBluePacketSummary(
net.sourceforge.jhcicap.net.RawPacket rawPacket,
      Vector packetsVector ) {
    // ObtenciÃ³n del frame genÃ©rico del paquete
    byte[] frame = rawPacket.getData();
    String[] data = new String[4];

    try {
    PHCI hciPacket = new PHCI(0,frame,rawPacket.getIncoming());

    switch ( hciPacket.getUpperLayerProtocol() ) {
      case ProtocolIDs.L2CAP: {
        PL2CAP l2capPacket = new
PL2CAP(hciPacket.getHCIHeaderSize(),frame,rawPacket.getIncoming());
```

```
        switch( l2capPacket.getUpperLayerProtocol(true) ) {
              case ProtocolIDs.SDP:
                PSDP sdpPacket = new PSDP(hciPacket.getHCIHeaderSize()+
                    l2capPacket.getL2CAPHeaderSize(),frame);

                l2capPacket.getInfoSummaryPacket(data);
                sdpPacket.getInfoSummaryPacket(data);

                // Checar si son paquetes orientados a conexiÃ³n
                if ( l2capPacket.getChannelID() >= PL2CAP.CID_DYNAMICALLY_ALLOCA )
{
                  extraInfoPacket.put(new Integer(packetsVector.size()),new
Integer(ProtocolIDs.SDP));
                }
                break;

              case ProtocolIDs.RFCOMM:
                PRFCOMM rfcommPacket = new PRFCOMM(hciPacket.getHCIHeaderSize()+
                    l2capPacket.getL2CAPHeaderSize(),frame);

                l2capPacket.getInfoSummaryPacket(data);
                rfcommPacket.getInfoSummaryPacket(data);

                // Checar su son paquetes orientados a conexiÃ³n
                if ( l2capPacket.getChannelID() >= PL2CAP.CID_DYNAMICALLY_ALLOCA )
{
                  extraInfoPacket.put(new Integer(packetsVector.size()),new
Integer(ProtocolIDs.RFCOMM));
                }
                break;

              case ProtocolIDs.TCS_BIN:
                data[3] = "TCS-BIN";

                // Checar su son paquetes orientados a conexiÃ³n
                if ( l2capPacket.getChannelID() >= PL2CAP.CID_DYNAMICALLY_ALLOCA )
{
                  extraInfoPacket.put(new Integer(packetsVector.size()),new
Integer(ProtocolIDs.TCS_BIN));
                }
                break;

              default:
                l2capPacket.getInfoSummaryPacket(data);
                break;
            }
        }
        break;

      case ProtocolIDs.VOICE:

        break;

      default:
        hciPacket.getInfoSummaryPacket(data);
        break;
    }

    data[3] = (rawPacket.getIncoming()!=0?"> ":"< ") + data[3];
```

```
    } catch (Exception ex){}


    return data;
  }

  public static boolean displayBluePacket(int row, Vector packetsVector,
DefaultTreeModel detailInfo) {
    if ( row >= 0)
    {
      net.sourceforge.jhcicap.net.RawPacket rawPacket =
(net.sourceforge.jhcicap.net.RawPacket) packetsVector.get(row);

      if (rawPacket != null) {
        byte[] frame = rawPacket.getData();
        //byte IncomingPacket = frame[0];
        //frame
        DefaultMutableTreeNode currentInfo = null;
        Vector infoTree = new Vector();
        Integer protocol;
        int offset = 0;

        infoTree.add("root");

        // Obtener la informaciÃ³n general del paquete
        infoTree.add(getGeneralInfoPacket(row, packetsVector));

        // Obtener y Mostrar la informaciÃ³n del Protocolo Bluetooth del frame
capturado
        PHCI hciPacket = new PHCI(offset, frame, rawPacket.getIncoming());

        infoTree.add(hciPacket.getProtocolInfo());

        offset += hciPacket.getHCIHeaderSize();

        // Obtener la informaciÃ³n de si existen los protocolos L2CAP
        // o de Voz
        switch ( hciPacket.getUpperLayerProtocol() ) {
          case ProtocolIDs.L2CAP:
            PL2CAP l2capPacket = new PL2CAP(offset,frame,rawPacket.getIncoming());

            infoTree.add(l2capPacket.getProtocolInfo());

            offset += l2capPacket.getL2CAPHeaderSize();

            protocol = new Integer(l2capPacket.getUpperLayerProtocol(false));

            // Check if exists an upper protocol for this packet
            if ( protocol.intValue() == ProtocolIDs.UNKNOWN ) {
              protocol = (Integer) extraInfoPacket.get(new Integer(row));

              if (protocol == null)
                protocol = new Integer(ProtocolIDs.UNKNOWN);
            }

            switch( protocol.intValue() ) {
              case ProtocolIDs.SDP:
                PSDP sdpPacket = new PSDP(offset,frame);
```

```
                infoTree.add(sdpPacket.getProtocolInfo());
                break;

              case ProtocolIDs.RFCOMM:
                PRFCOMM rfcommPacket = new PRFCOMM(offset,frame);

                infoTree.add( rfcommPacket.getProtocolInfo() );
                break;

              case ProtocolIDs.TCS_BIN:
                break;
            }
            break;

          case ProtocolIDs.VOICE:

            break;
        }

        detailInfo.setRoot( (TreeNode)
Utilities.FieldOfFrame.processHierarchy(infoTree.toArray()));

        return true;
      }
    }

    return false;
  }
}
```

## Clase FileFunctions

```
import net.sourceforge.jpcap.capture.*;
import net.sourceforge.jpcap.util.*;
import net.sourceforge.jhcicap.util.*;
import java.io.*;

/**
 * <p>Title: Wireless Sniffer</p>
 * <p>Description: sniffing wifi and bluetooth networks</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: UTM</p>
 * @author Emmanuel Mendoza Acevedo
 * @version 1.0
 */

public class FileFunctions {
  // Clase de definiciÃ³n de funciones para la manipulaciÃ³n de Archivos

  ///////////////////////
  ///////Guardar en archivos
  public static void fileSaver(String nameFile, WifiBlueSniffer owner) throws
IOException
  {
    int numPackets = owner.capturedPackets.size();

    if ( owner.WifiNotBluetooth ) {
```

```
      net.sourceforge.jpcap.net.RawPacket packet;
      writeTcpdumpHeader(nameFile, owner.snaplen, owner.LINK_LAYER_802_11);

      for(int i=0; i<numPackets; i++)
      {
        packet = (net.sourceforge.jpcap.net.RawPacket)owner.capturedPackets.get(i);
        appendPacketWifi(nameFile, packet);
      }
    }
    else {
      net.sourceforge.jhcicap.net.RawPacket packet;
      HcidumpWriter.writeHeader(nameFile);

      for(int i=0; i<numPackets; i++)
      {
        packet =
(net.sourceforge.jhcicap.net.RawPacket)owner.capturedPackets.get(i);
        appendPacketBluetooth(nameFile, packet);
      }
    }

  }

  /**
   * A Tcpdump file header is 24 bytes:
   *   4 bytes: Tcpdump signature "magic number" 0xA1B2C3D4
   *   2 bytes: Major version number, currently 2
   *   2 bytes: Minor version number, currently 4
   *   4 bytes: Time Zone offset, currently unused (set to zero)
   *   4 bytes: Time stamp accuracy, currently unused (set to zero)
   *   4 bytes: Snapshot length
   *   4 bytes: Link-layer type
   */
  public static void writeTcpdumpHeader(String filename, int snaplen, int
linklayer)
      throws IOException {

    byte[] headerArray = new byte[24];

    ArrayHelper.fillBytesLittleEndian(headerArray, 0xA1B2C3D4, 4, 0);
    ArrayHelper.fillBytesLittleEndian(headerArray, TcpdumpWriter.MAJOR_VERSION, 2,
4);
    ArrayHelper.fillBytesLittleEndian(headerArray, TcpdumpWriter.MINOR_VERSION, 2,
6);
    ArrayHelper.fillBytesLittleEndian(headerArray, 0, 8, 8);
    ArrayHelper.fillBytesLittleEndian(headerArray, snaplen, 4, 16);
    ArrayHelper.fillBytesLittleEndian(headerArray, linklayer, 4, 20);

    FileUtility.writeFile(headerArray, filename, false);
  }

  /**
   * A Tcpdump packet has the following format:
   *   seconds [4 bytes]
   *   microseconds [4 bytes]
   *   captured length [4 bytes]
   *   original packet length [4 bytes]
   *   packet data [variable length]
   */
```

```
  public static void appendPacketWifi(String filename,
      net.sourceforge.jpcap.net.RawPacket rawPacket) throws IOException
  {
    TcpdumpWriter.appendPacket(filename, rawPacket, TcpdumpWriter.LITTLE_ENDIAN);
  }

  /**
   * A Hcidump packet has the following format:
   *   length [2 bytes]
   *   in [1 byte]
   *   pad [1 byte]
   *   seconds [4 bytes]
   *   microseconds [4 bytes]
   *   packet data [variable length]
   */
  public static void appendPacketBluetooth(String filename,
      net.sourceforge.jhcicap.net.RawPacket rawPacket) throws IOException
  {
    HcidumpWriter.appendPacket(filename, rawPacket, HcidumpWriter.LITTLE_ENDIAN);
  }

  ////////////////////
  ///////Cargado de archivos
  public static void fileLoader(String nameFile, WifiBlueSniffer owner)
      throws CaptureFileOpenException, CaptureConfigurationException,
CapturePacketException
  {
    LoaderHandle handle = new LoaderHandle(owner);
    int snaplen = 0;
    boolean WifiNotBlue = false;
    net.sourceforge.jpcap.capture.PacketCapture wifiSession = null;
    net.sourceforge.jhcicap.capture.PacketCapture blueSession = null;

    blueSession = new net.sourceforge.jhcicap.capture.PacketCapture();
    blueSession.addRawPacketListener(handle);

    try
    {
      blueSession.openOffline(nameFile);
      snaplen = blueSession.getSnapshotLength();
    } catch ( CaptureFileOpenException e ) {
      // No es un archivo con paquetes Bluetooth
      WifiNotBlue = true;
    }

    if ( WifiNotBlue ) {
      wifiSession = new net.sourceforge.jpcap.capture.PacketCapture();
      wifiSession.addRawPacketListener(handle);
      wifiSession.openOffline(nameFile);

      switch( wifiSession.getLinkLayerType() ) {
        case WifiBlueSniffer.LINK_LAYER_802_11: WifiNotBlue = true; break;
        //case WifiBlueSniffer.LINK_LAYER_BLUETOOTH: WifiNotBlue = false; break;
        default: throw new CaptureConfigurationException("LinkLayer not
supported");
      }

      snaplen = wifiSession.getSnapshotLength();
    }
```

```
    owner.snaplen = snaplen;
    owner.WifiNotBluetooth = WifiNotBlue;

    // Comenzar la carga de paquetes
    if ( WifiNotBlue ) {
      wifiSession.capture(-1);
      wifiSession.close();
    }
    else {
      blueSession.capture(-1);
      blueSession.close();
    }
  }

  private static class LoaderHandle implements RawPacketListener
  {
    static WifiBlueSniffer Owner;

    public LoaderHandle( WifiBlueSniffer owner ) {
      Owner = owner;
    }

    public void rawPacketArrived( net.sourceforge.jpcap.net.RawPacket packet ) {
      Owner.addPacketToList(packet);
    }
  }

  ////////////////////

  // ExtensiÃƒÂ³n de los Archivos acceptados
  public static final String ext = "wcs";

  public static String getExtension(File f) {
    String ext = "";
    String s = f.getName();
    int i = s.lastIndexOf('.');

    if (i > 0 &&  i < s.length() - 1) {
      ext = s.substring(i+1).toLowerCase();
    }

    return ext;
  }
}
```

## Clases para la optimización de la captura de paquetes en tiempo real

```
import net.sourceforge.jpcap.net.RawPacket;
import java.util.*;

/**
 * <p>Title: Sniffer Wireless</p>
 * <p>Description: sniffing wifi and bluetooth networks</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: Personal</p>
 * @author Emmanuel Mendoza Acevedo
 * @version 1.0
```

```
 */

public class Utilities {

///// Clases para desplegar los paquetes en la lista y de esta forma
///// optimizar la captura en tiempo real
   public static class PacketsDelivery extends Thread {
     WifiBlueSniffer destiny;
     private boolean ok;

     public PacketsDelivery(WifiBlueSniffer dest) {
       destiny = dest;
     }

     public void start() {
       ok = true;
       super.start();
     }

     public void finished() {
       ok = false;
     }

     public void run() {
       while( ok ) {
         destiny.addPacketToList(destiny.container.getSharedPacket());

         try {
           sleep(1);
         }
         catch ( InterruptedException e ) {
           System.err.println( "Exception: " + e.toString() );
         }
       }
     }
   }

   public static class PacketsContainer {
     private Object []sharedPackets;
     private boolean writeable = true;
     private boolean readable = false;
     private int readLoc = 0, writeLoc = 0;
     private int numPackets;

     public PacketsContainer(int num) {
       numPackets = num;
       sharedPackets = new Object[num];
     }

     public synchronized void setSharedPacket( RawPacket packet ) {
       while( !writeable ) {
         try {
           wait();
         }
         catch ( InterruptedException e ) {
           System.err.println( "Exception: " + e.toString() );
         }
       }
```

```
        sharedPackets[writeLoc] = packet;
        readable = true;
        writeLoc = ++writeLoc % numPackets;

        if (writeLoc == readLoc ) {
          writeable = false;
        }

        notify();
      }

      public synchronized RawPacket getSharedPacket() {
        Object packet;

        while( !readable ) {
          try {
            wait();
          }
          catch ( InterruptedException e ) {
            System.err.println( "Exception: " + e.toString() );
          }
        }

        writeable = true;
        packet = sharedPackets[readLoc];

        readLoc = ++readLoc % numPackets;

        if ( readLoc == writeLoc ) {
          readable = false;
        }

        notify();

        return (RawPacket)packet;
      }
    }
}
```

# Lista de Figuras

# Lista de Tablas