Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Laboratorio de Comunicación y Redes

Desarrollo de un Agente SNMP para Dispositivos Móviles sobre la Plataforma Android

Trabajo Especial de Grado
Presentado ante la
Ilustre Universidad Central de Venezuela
por el Bachiller:

Fernando C. Hidalgo C.
C.I.: 19.672.772
fechidal89@gmail.com

para optar por el titulo de Licenciado en Computación

Tutor: Prof. Eric Gamess

Caracas, Abril 2014

Agradecimiento

A Dios por el don de la vida y por darme la oportunidad de realizar mis estudios universitarios en esta casa, la casa que vence la sombra.

A mis padres, en especial a mi madre Elina Castro, por ser esa persona que siempre me impulsó a ser mejor persona, a darlo todo por aprender cada día y a esforzarme por ser un mejor estudiante.

A mi tutor, Prof. Eric Gamess, por tenerme tanta paciencia, por dedicar su tiempo en este proyecto y por brindarme su conocimiento. Gracias profundamente.

A la Profa. Marlliny Molsalve y al Prof. Rhadames Carmona por su ejemplo de excelencia e inculcarme a temprana edad de mis estudios el mayor conocimiento de la carrera: Lógica y Programación.

Agradezco a mis compañeros, amigos y hermanos por vivir junto a mi esta experiencia tan magnifica de aprender, de disfrutar la vida universitaria al máximo y de no dejar ningún detalle atrás.

Fernando C. Hidalgo C.

Resumen

TÍTULO
Desarrollo de un Agente SNMP para Dispositivos Móviles sobre la Plataforma Android.

AUTOR
Fernando C. Hidalgo C.

TUTOR
Prof. Eric Gamess.

En este Trabajo Especial de Grado, se propone el desarrollo de un agente SNMP para dispositivos móviles que ejecuten el sistema operativo Android. Cada día, las TICs evolucionan con la aparición de nuevas tecnologías emergentes como es la llegada masiva de los nuevos equipos electrónicos portables (PDAs, *smartphone*, *tablets*, cámaras, etc.). Esto conlleva a que los seres humanos se actualicen constantemente y dispongan de estos dispositivos para facilitar su día a día y mantenerse conectados a la red de redes, el Internet.

Con la llegada de los *smartphones* y *tablets* a las redes empresariales, los administradores de red se ven en la necesidad de agregar redes inalámbricas para proveer el servicio de conexión al Internet a estos usuarios. Los empleados necesitan estar continuamente conectados al Internet para revisar sus cuentas de correo electrónicos, agendas, calendarios, y mantenerse siempre comunicados. Por lo tanto, la cantidad de dispositivos que presenta una red empresarial aumenta considerablemente y con ello el tráfico de red. Los administradores necesitan una forma para poder administrar y monitorear estos dispositivos que presentan un comportamiento dinámico en la red empresarial.

Por lo antes planteado, se desarrolló un agente SNMP que se ejecuta en *background* en la plataforma Android, como una extensión de la clase *Services*. Esta aplicación (*Activity*) presenta varios módulos donde se pueden enviar tramas *Traps* y configurar información del *System Group* (MIB-II) como el nombre del contacto del administrador y la ubicación del dispositivo. Actualmente, el agente tiene soporte para SNMPv1 y SNMPv2c a través de la biblioteca *SNMP Package*. Para esta versión del agente se implementó la MIB-II y un componente de la MIB HOST-RESOURCES para demostrar que la aplicación es totalmente extensible.

Para corroborar el correcto funcionamiento del agente en dispositivos móviles, se desarrolló una aplicación que evalúa su desempeño reportando métricas como: RTT (Round Trip Time) y PDR (Packet Delivery Ratio). Las pruebas que se realizaron en dispositivos reales mostraron tiempos de respuestas aceptables, demostrando así que los dispositivos portátiles pueden ser integrados en los sistemas de administración de redes sin comprometer su poder de cómputo o su sistema de comunicación.

Palabras claves: Administración de Red, SNMP, Agente, Android.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Para mediados de 1990, se introduce el protocolo SNMP que permite administrar remotamente dispositivos red. Como se verá a lo largo del documento, SNMP es un protocolo simple de administración de red que se basa en solicitudes y respuestas. Las solicitudes consisten en un conjunto de variables que pueden ser consultadas o modificadas en el ente administrado. Estas variables pueden ser estadísticas de protocolos de red (TCP, UDP, IP, ICMP, entre otros), estado del dispositivo, elementos de hardware o de software que tiene instalado y cualquier información que pueda ser consultada sobre el dispositivo administrado. Además del clásico uso de solicitud/respuesta, SNMP puede enviar notificaciones sobre sucesos que ocurren en el dispositivo administrado.

Con el auge que ha tenido la tecnología, los dispositivos han cambiado y las necesidades de comunicación también, los usuarios cada día son más exigentes. En específico, la telefonía móvil ha cambiado drásticamente aportando nuevos dispositivos y conexiones a redes de datos más potentes. El documento presenta los sistemas operativos que poseen los dispositivos móviles, realizando una breve comparación entre estos y determinando que la plataforma Android es la más usada debido a sus características. Por lo tanto se dedica el Capítulo 7 exclusivamente para el estudio de la plataforma Android presentando su arquitectura y sus aplicaciones.

Hoy en día muchas empresas han optado por implementar la plataforma Android en sus dispositivos móviles, obteniendo como resultado una venta masiva de estos. Vale destacar que los dispositivos Android poseen diferentes interfaces de comunicación que son administrables a través de este. Aunado a esto, el uso que se le da a los dispositivos móviles se ha diversificado debido a la facilidad de conexión al Internet. La mayoría de las aplicaciones que se hacen para estos dispositivos se comunican a través del Internet para proveer servicios de correo, redes sociales, mensajería instantánea y acceso a la web. Estas aplicaciones son desarrolladas con el lenguaje de programación Java sobre una versión optimizada de J2ME (Java 2 Micro Edition).

Actualmente, los usuarios son muy exigentes en la forma de comunicarse y de administrar la información relacionada con su entorno laboral. Uno de los motivos por el cual los usuarios utilizan estos dispositivos móviles es para actualizarse y mantener sincronizados cuentas de correos, agendas, recordatorios y calendarios para facilitar su día a día. Esto lleva a que siempre necesitan estar conectados a las redes de datos, tanto en la calle como en las oficinas. Al proveer el servicio de conexión a través de redes inalámbricas las empresas no tienen forma de administrar estos dispositivos que cada día son más. Por lo antes dicho, el presente Trabajo Especial de Grado realiza una propuesta y solución para que estos dispositivos puedan ser administrados remotamente a través del protocolo

SNMP. Así los administradores de red pueden tener un control total de todos los dispositivos que presenta la infraestructura de red.

A continuación se presenta la estructura del documento:

- **Capítulo 2: El Problema**. Plantea la problemática actual y objetivos requeridos para dar solución al problema.
- **Capítulo 3: Simple Network Management Protocol (SNMP)**. Introduce la especificación del protocolo de administración de red, describiendo su estructura, mensajes intercambiados entre los entes, y sus distintas versiones.
- **Capítulo 4: Structure of Management Information (SMI)**. Define la estructura de cómo los objetos de la MIB son representados. Se presentan la SMIv1 y la SMIv2.
- **Capítulo 5: Management Information Base (MIB)**. Especifica los conjuntos de objetos que son consultados o modificados por SNMP en los dispositivos administrados.
- **Capítulo 6: Dispositivos Móviles y sus Sistemas Operativos**. Hace una introducción a los SOs que existen actualmente en el mercado o que tuvieron un auge importante en la tecnología móvil. También se presenta una pequeña comparación entre ellos.
- **Capítulo 7: Plataforma Android**. Realiza un estudio de la plataforma Android abarcando los conceptos básicos para la creación de aplicaciones, ciclo de vida y procesos.
- **Capítulo 8: Herramientas Relacionadas con SNMP para Android**. Describe algunas aplicaciones que existen actualmente en el mercado relacionadas con el protocolo de estudio y la plataforma Android.
- **Capítulo 9: Marco Metodológico**. Realiza la descripción de la metodología utilizada para crear la solución propuesta.
- **Capítulo 10: Marco Aplicativo**. Describe el proceso de desarrollo e implementación de la solución a través de la metodología utilizada.
- **Capítulo 11: Evaluación de Desempeño**. Realiza el análisis de los tiempos de respuesta y capacidad de procesamiento de la solución implementada.
- **Capítulo 12: Conclusiones y Trabajos Futuros**. Como punto final se concluye el estudio realizado y se proponen trabajos futuros.

Capítulo 2

El Problema

En este capítulo, se expone la justificación del desarrollo de un agente SNMP, para así dar una solución a la problemática que presenta la administración de la infraestructura de red con respecto a los dispositivos móviles bajo la plataforma Android.

2.1. Planteamiento del Problema

La administración de las redes ha sido un tema de gran estudio resultando en la creación de distintos protocolos. La simplicidad es una característica que se debe tomar en cuenta a la hora de crear y elegir un protocolo de administración de red. En las empresas, la necesidad de administración de los elementos de red ha ido aumentando drásticamente debido a la cantidad de dispositivos que conforman la infraestructura. Por lo antes dicho, SNMP cumple una función muy importante en las labores de los administradores de red, ya que posee la capacidad de administración de una gran cantidad de dispositivos y de una forma simple.

Como se dijo antes, la cantidad de dispositivos en la infraestructura de red ha aumentado, ya sea porque se agregan nuevas estaciones de trabajo, se colocan adaptadores de red a los dispositivos que no poseen una interfaz de comunicación, se presentan nuevas necesidades en la lógica del negocio, o porque algunos dispositivos móviles visitan la red. Este último motivo es un caso muy particular, ya que no todos los dispositivos móviles en el mercado pueden ser administrados remotamente.

La tecnología celular ha mostrado un auge en los últimos años, presentado nuevas formas de comunicarse y prestando nuevos servicios. A los inicios de este auge solo se disponía de comunicaciones a través de mensajes en texto plano y llamadas telefónicas. Actualmente, los teléfonos celulares prestan servicios de conexión al Internet a través de distintas aplicaciones y una interfaz de usuario que ha evolucionado hacia las pantallas táctiles, además de ofrecer distintas interfaces de comunicación.

Los dispositivos móviles tienen la necesidad de estar constantemente conectados al Internet para proporcionar interacción con otros usuarios y con los servicios. Las empresas tienen como obligación mantener una red inalámbrica dentro de su infraestructura, en gran parte debido a la necesidad de conexión de los dispositivos móviles. Esta necesidad trae como consecuencia la falta de administración de los dispositivos móviles en las redes empresariales, ya que la mayoría de estos no presentan una aplicación que permita administrarlos remotamente.

Por la problemática antes planteada, se propone como solución la creación de un agente SNMP que permita la administración remota de dispositivos móviles bajo la plataforma Android, la cual que ha tenido mucho éxito como sistema operativo de teléfonos móviles.

2.2. Justificación del Problema

En la mayoría de las pequeñas y grandes empresas, los trabajadores poseen un dispositivo móvil que permite conexión al Internet a través de distintas interfaces. Es por esto que surge la necesidad de que estos dispositivos sean administrados remotamente y así poder tener control total de las conexiones y dispositivos que están presentes en la infraestructura de red.

2.3. Objetivos

A continuación se presentan los objetivos de la presente propuesta de Trabajo Especial de Grado.

2.3.1. Objetivo General

Desarrollar un agente SNMP para dispositivos móviles sobre la plataforma Android.

2.3.2. Objetivo Específicos

- Estudiar el protocolo SNMP y sus distintas versiones.
- Instalar un ambiente de desarrollo para la creación de aplicaciones en la plataforma Android.
- Investigar y estudiar el API que provee el SDK de Android.
- Diseñar e implementar la MIB-II basada en el API proveída por la plataforma.
- Crear de la documentación necesaria para la configuración y puesta en marcha del agente.
- Realizar pruebas sobre el agente con distintas aplicaciones que funcionen como estaciones de administración de red (manager).

2.4. Alcance

El agente SNMP a desarrollar deberá soportar el envío y recepción de mensajes SNMP (v1 y v2c) con la limitación de los objetos definidos en las MIBs que se implementen y que soporte la plataforma.

Capítulo 3

Simple Network Management Protocol

SNMP es un protocolo simple de administración de red que opera en capa de aplicación usando UDP como transporte. Los puertos usados en capa de transporte son el 161 en los dispositivos administrados para consultas y escritura y el 162 en las estaciones de administración para la recepción de los *Traps*. SNMP funciona con el paradigma cliente-servidor, donde existe una entidad administradora o *manager* (cliente) y una entidad administrada o *agent* (servidor), los cuales intercambian mensajes por los puertos antes mencionados a través de las primitivas *request-response*.

Existen varias versiones de SNMP que incluyen: SNMPv1 (versión 1), SNMPv2 (versión 2) mejor conocido como SNMPv2c y SNMPv3 (versión 3). Todas estas versiones serán estudiadas individualmente en las siguientes secciones.

3.1. Simple Network Management Protocol Version 1

El modelo de arquitectura en SNMP es una colección de estaciones de administración de red y elementos administrados de red. Las estaciones de administración de red también conocidas como NMSs o Network Management Systems ejecutan una aplicación de gestión donde monitorean y controlan los elementos de red. Los elementos de red pueden ser dispositivos como hosts, switches, routers, servidores, entre otros, los cuales poseen un agente responsable de responder a las solicitudes de administración realizadas por los NMSs. SNMP es usado para realizar la comunicación de la información de gestión entre los NMSs y los agentes en los elementos de red.

Algunos de los objetivos de la arquitectura de SNMP son descritos a continuación:

- Reducir el número y la complejidad en las funciones realizadas por el agente administrado. Esto es atractivo por los siguientes aspectos:
  - El costo de desarrollo del software del agente es reducido.
  - Se incrementa el grado de las funciones de administración remotamente, maximizando el uso de los recursos de Internet en las tareas de administración.
  - Se imponen menos restricciones en las herramientas de administración.
  - El conjunto de funciones de administración simplificadas son fáciles de entender y de usar por los desarrolladores de herramientas de administración.
- Conseguir que el paradigma funcional para monitorear y controlar los dispositivos sea lo suficientemente extensible.

- Obtener una independencia de la arquitectura y mecanismos de los elementos de red en particular.

SNMP admite una variedad de relaciones administrativas entre las entidades que participan en la comunicación usando el protocolo. La entidad residente en las estaciones de administración o en los elementos de red que comunica a ambos usando SNMP es denominado aplicación SNMP. El proceso que implementa SNMP y que da soporte a la aplicación SNMP es denominado entidad de protocolo [1].

Un grupo de agentes SNMP que poseen arbitrariamente un conjunto de entidades de aplicaciones SNMP es llamado comunidad SNMP. Cada comunidad SNMP es identificada por una cadena de caracteres, que determina su nombre. Un mensaje originado por una aplicación SNMP es autenticado por el nombre de la comunidad que lleva este, el cual debe ser el mismo nombre de comunidad a la que pertenece dicha aplicación. Un servicio de autenticación es un conjunto de esquemas de autenticación que poseen a su vez reglas para identificar un mensaje SNMP auténtico [1].

3.1.1. Operaciones Soportadas por SNMPv1

Básicamente SNMP presenta dos operaciones sobre la información administrada: consulta o modificación de las variables. Así, una entidad de protocolo que está lógicamente en el host remoto interactúa con el agente que reside en el elemento de red con el fin de recuperar (get) o modificar (set) variables. Por lo tanto, tiene dos consecuencias positivas:

- Limitar el número de funciones esenciales de administración realizadas por el agente administrado a dos: (1) la de obtener el valor de una configuración especificada u otro parámetro y (2) la de realizar una operación de asignación de un valor.
- Evitar dentro de la definición del protocolo el soporte de comandos de administración imperativos.

El alcance de la información administrada, comunicada por las operaciones de SNMP, está exactamente representada por instancias de todos los tipos de objetos no agregados que están definidos en el estándar del Internet MIB (Capítulo 5) o definidas en otra parte acordando con el conjunto de convenciones establecidas en el estándar del Internet SMI (Capítulo 4).

3.1.2. Funcionamiento de SNMPv1

Las acciones de nivel superior de una entidad de protocolo que genera un mensaje son las siguientes:

- Primero se construye una PDU según sea el caso, como un objeto ASN.1 (*Abstract Syntax Notation 1*).
- Al objeto ASN.1 se le agrega el nombre de la comunidad y las direcciones de transporte origen y destino, el cual es enviado al servicio que implementa el esquema de autenticación. Este servicio devuelve otro objeto ASN.1.
- La entidad de protocolo construye un mensaje con el objeto ASN.1, usando el nombre de la comunidad dando como resultado otro objeto ASN.1.
- Este nuevo objeto es serializado, con las reglas de codificación de ASN.1, y es enviado con el servicio de transporte a la entidad de protocolo par.

Análogamente, las acciones al nivel de la entidad que recibe el mensaje son las siguientes:

- Primero se realiza el parseo del datagrama entrante construyendo un objeto ASN.1 correspondiente al mensaje que contiene un objeto ASN.1. Si este falla, se descarta el datagrama y no se realizan más acciones.
- Seguidamente se verifica el número de versión del mensaje SNMP. Si no concuerda, se descarta el datagrama y no se llevan a cabo más acciones.
- Entonces la entidad de protocolo pasa el nombre de la comunidad y los datos de usuario que se encuentran en el objeto de mensaje de ASN.1, junto con las direcciones de origen y destino de la capa de transporte al servicio que implementa el esquema de autenticación deseado. Esto retorna un objeto ASN.1 o una señal de autenticación fallida (posiblemente una trama *Trap*).
- Por último la entidad de protocolo procesa la carga útil, en caso de algún fallo se descarta el datagrama, sino se retorna un mensaje como respuesta a la dirección de transporte origen que realizó la solicitud.

3.1.3. Especificación de las PDUs en SNMPv1

Las cinco PDUs definidas en SNMPv1 son:

- **GetRequest**: Es generada por la entidad de protocolo cuando desea hacer una solicitud del valor de una variable a un agente.
- **GetNextRequest**: Es generado por la entidad de protocolo cuando requiere saber el valor de la próxima variable de un agente en la MIB.
- **SetRequest**: Es generado por la entidad de protocolo para asignar valor a una variable presente en la MIB del agente.
- **GetResponse**: Es generado por el agente en respuesta a alguna solicitud hecha por una entidad de protocolo, tal como GetRequest, GetNextRequest o un SetRequest.
- **Trap**: Es generado desde el agente para notificar a la entidad de protocolo algún evento que haya ocurrido. A continuación se describen algunos *Traps* genéricos.

    - **coldStart**: Significa que la entidad de protocolo que envió el *Trap* está reinicializándose por sí misma y que de esta manera la configuración de los agentes o de la implementación de la entidad de protocolo puede ser alterada.
    - **warmStart**: Significa que la entidad de protocolo que envió el *Trap* está reinicializándose por sí misma y que de esta manera ni la configuración de los agentes ni de la implementación de la entidad de protocolo puede ser alterada.
    - **linkDowm**: Significa que la entidad de protocolo reconoce una falla en uno de los enlaces de comunicación del agente.
    - **linkUp**: Significa que la entidad de protocolo reconoce que uno de los enlaces de comunicación presente en el agente ha iniciado su servicio.
    - **authenticationFailure**: Significa que el mensaje que ha sido enviado no puede ser autenticado.
    - **egpNeigborLoss**: Significa que un vecino EGP (Exterior Gateway Protocol) de la entidad de protocolo no puede ser encontrado debido a que se encuentra inoperativo.
    - **enterpriseSpecific**: Significa que la entidad de protocolo reconoce que algún evento específico ha ocurrido.

3.1.4. Formato de las PDUs y Mensajes en SNMPv1

A continuación se presenta el formato de las PDUs antes descritas, y se explica cada campo del paquete que viaja en la carga útil. Un paquete SNMP tendrá de forma general los campos que se presentan en la Figura 3.1.

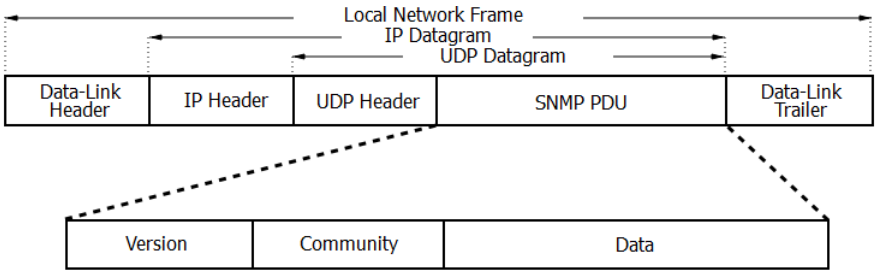

Figura 3.1: Mensaje SNMPv1

- **Version**: Es un entero que representa la versión del protocolo SNMP. El valor 0 en este campo significa que se está utilizando la versión uno (1) del protocolo.
- **Community**: Es una cadena de octetos que posee el nombre de comunidad donde fue generado el mensaje SNMP.
- **Data**: Varía según la PDU que se envíe y es donde se encuentra la carga útil del protocolo.

Como se mencionó antes, la información que viaja en el campo Data varía según el PDU que se esté construyendo o enviando, pero su estructura cambia solo cuando se construye un *Trap*, del resto los otros mensajes poseen el mismo formato, el cual se observa en la Figura 3.2.

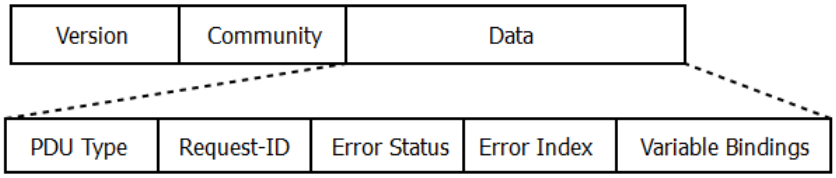

Figura 3.2: PDUs GetRequest, GetNextRequest, GetResponse y SetRequest de SNMPv1

- **PDUType**: Es un entero que representa el tipo de mensaje que se está enviando.
- **Request-ID**: Es un entero que se usa como identificador correlativo a las solicitudes y respuestas, evitando así mezclarlas o tener respuestas duplicadas.
- **Error-Status**: Es un entero que indica cuando un error ha ocurrido mientras se procesaba una solicitud.
- **Error-Index**: Es un entero que especifica más información indicando que variable en la lista ha causado el error.
- **Variable-bindings**: Es una lista de variables que son las instancias de los objetos administrados. Cada elemento de la lista tiene un par (Nombre-Valor), que identifica un objeto específico.

El campo Data para las tramas *Trap* posee el formato mostrado en la Figura 3.3.

- **PDUType**: Es un entero que representa el tipo de mensaje que se está enviando, en este caso su valor es 4 y corresponde a un *Trap*.

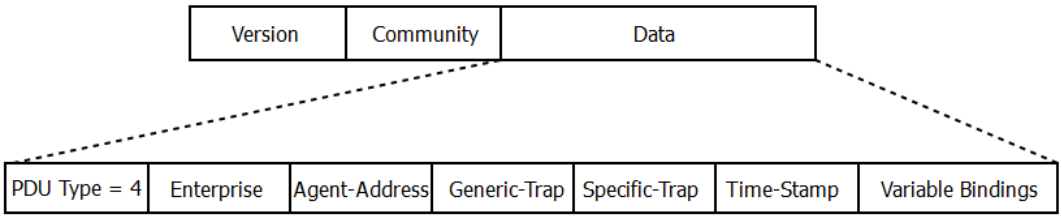

Figura 3.3: PDU Trap de SNMPv1

- **Enterprise**: Es un OBJECT IDENTIFIER del generador del *Trap*.
- **Agent-Address**: Es la dirección de red que origina el *Trap*.
- **Generic-Trap**: Es un entero que especifica el tipo de *Trap* genérica.
- **Specific-Trap**: Es un código específico, presente incluso si el *Generic-Trap* no es un `enterpriseSpecific`.
- **Time-stamp**: Representa el tiempo transcurrido entre la última (re)inicialización del elemento de red y la generación del *Trap*.
- **Variable-bindings**: Información relevante.

Como se pudo observar antes, los mensajes GetRequest, GetNextRequest, GetResponse y SetRequest poseen un formato en común, a diferencia de las tramas *Traps* que es el único que varía, aun así todos estos tienen dos campos en común (PDUType y Variable-bindings). En la Tabla 3.1 se muestran los valores correspondientes a cada mensaje para el campo PDUType.

PDU	Valor del Campo PDUType
GetRequest	0
GetNextRequest	1
GetResponse	2
SetRequest	3
Trap	4

Tabla 3.1: Valores del Campo PDUType

El siguiente campo en común es una lista de *variable-binding*, donde cada elemento es un par compuesto por el nombre de la variable y su respectivo valor (ver Figura 3.4).

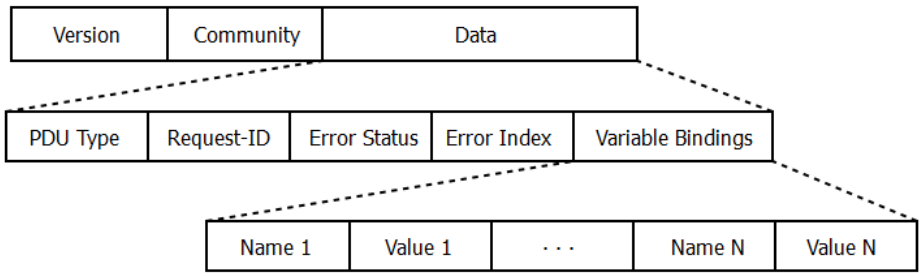

Figura 3.4: Campo Variable-Bindings de SNMPv1

3.2. Simple Network Management Protocol Version 2

Esta es la segunda versión de SNMP la cual trae algunas mejoras sobre su versión antecesora, como mayor soporte de errores y modificación del PDU *Trap* para manejarla con el mismo formato de los otros PDUs. Posee las mismas operaciones que SNMPv1 para mantener su compatibilidad hacia atrás y sumando otras haciendo posible la comunicación entre estaciones de administración. Este es especificado en el RFC 1901 [2] y en el RFC 1445 [3].

3.2.1. Elementos del Modelo

SNMPv2 Party

Un *SNMPv2 Party* es un modelo conceptual, un ambiente virtual de ejecución donde las operaciones están restringidas a un subconjunto administrativamente definido de todas las posibles operaciones de una entidad SNMPv2 en particular. Cuando una entidad SNMPv2 procesa un mensaje, el conjunto posible de operaciones especificadas para un *SNMPv2 Party* puede ser superpuesto o disjunto con respecto a los otros *SNMPv2 parties*, también puede ser un subconjunto apropiado o inapropiado de todas las posibles operaciones de la entidad SNMPv2.

Cada *SNMPv2 Party* comprende la siguiente arquitectura:

- Una identificación simple.
- Una ubicación lógica en la red, caracterizada por un dominio en el protocolo de transporte e información de direccionamiento.
- Un protocolo simple de autenticación asociado con unos parámetros para que todos los mensajes originados sean autenticados como originales e íntegros.
- Un protocolo simple de privacidad asociados a unos parámetros para que todos los mensajes recibidos sean protegidos de divulgación.

Un *SNMPv2 Party* conceptualmente puede ser representado por ASN.1, el cual posee los componentes a continuación:

- **partyIdentity**: Es la identificación del *party*.
- **partyDomain**: Es el dominio de transporte e indica el servicio de transporte por donde el *party* recibe el tráfico de red administrado.
- **partyAddress**: Es la información de direccionamiento a nivel de transporte y representa una dirección en el servicio de transporte por donde el *party* recibe el tráfico de red administrado.
- **partyMaxMessageSize**: Es el máximo tamaño del mensaje y esta representa la longitud en octetos del mensaje más largo que puede aceptar un *party*.
- **partyAuthProtocol**: Es el protocolo de autenticación e identifica un protocolo y los mecanismos por el cual todos los mensajes generados por el *party* son autenticados como íntegros y originales. En este contexto, el valor `noAuth` significa que los mensajes generados por el *party* no son autenticados como íntegros y originales.
- **partyAuthClock**: Es el reloj de autenticación y representa una noción del tiempo actual que especifica el *party*. La importancia de este componente es específico para el protocolo de autenticación.

- **partyAuthPrivate**: Es la clave de autenticación privada y representa cualquier valor secreto necesitado para soportar el protocolo de autenticación.
- **partyAuthPublic**: Es la clave de autenticación pública y representa cualquier valor público que se puede necesitar para soportar el protocolo de autenticación.
- **partyAuthLifeTime**: Es el tiempo de vida y representa una cota superior administrativa para el retardo de entrega aceptable para los mensajes generados por el *party*.
- **partyPrivProtocolo**: Es el protocolo de privacidad e identifica un protocolo y mecanismo por el cual todos los mensajes recibidos por el *party* son protegidos de divulgación. En este contexto, el valor `noPriv` significa que todos los mensajes recibidos por el *party* no son protegidos de divulgación.
- **partyPrivPrivate**: Es la clave privada y representa cualquier valor secreto necesitado para soportar el protocolo de privacidad.
- **partyPrivPublic**: Es la clave de privacidad pública y representa cualquier valor público que puede ser necesitado para soportar el protocolo de privacidad.

Si para todas las *SNMPv2 Parties* que pertenecen a una entidad SNMPv2, el protocolo de autenticación es `noAuth` y el protocolo de privacidad es `noPriv`, entonces la entidad es llamada no segura.

Entidad y Roles SNMPv2

Una entidad SNMPv2 es un proceso que realiza operaciones de administración, generando y/o respondiendo mensajes del protocolo SNMPv2. Cuando una entidad SNMPv2 está actuando como un *SNMPv2 Party* en particular, las operaciones de esta entidad están restringidas al subconjunto de operaciones que están definidas administrativamente para el *party*.

Una entidad SNMPv2 no requiere procesar múltiples mensajes concurrentemente, independientemente de si tales mensajes requieren asumir la identidad de una misma o diferente entidad administrativa. Por lo tanto, una implementación de una entidad SNMPv2 que soporte más de una entidad administrativa no necesita soportar procesamiento multihilo. Sin embargo, puede haber situaciones donde los desarrolladores puedan usar procesamiento multihilo.

Una entidad SNMPv2 puede tener un rol en específico, según el cual la entidad puede realizar distintas operaciones de administración SNMPv2 que están restringidas al subconjunto de operaciones definidas para el *SNMPv2 Party*. Estos roles son: estación de administración SNMPv2 y agente SNMPv2.

Una estación de administración SNMPv2 es un rol operacional asumido por un *SNMPv2 Party* cuando este inicia operaciones de administración SNMPv2 para la generación apropiada de mensajes o cuando este recibe y procesa notificaciones *Trap*. Un agente SNMPv2 es un rol operacional que es asumido por un *SNMPv2 Party* cuando este realiza operaciones de administración SNMPv2 en respuesta a mensajes recibidos, los cuales son generados por una estación de administración SNMPv2.

Contexto SNMPv2

Un contexto SNMPv2 es una colección de información administrada accesible por una entidad SNMPv2. La colección de información administrada identificada por un contexto puede ser local o remota (proxy). Cuando se hace referencia a un contexto SNMPv2 local, la entidad usa mecanismos de acceso local para obtener la información administrada, identificada por el contexto SNMPv2. Para un contexto SNMPv2 remoto, los recursos están identificados por una relación proxy. En este caso la entidad actúa como un agente proxy para acceder a la información administrada identificada por un contexto SNMPv2.

Gestión de Comunicación SNMPv2

Una gestión de comunicación SNMPv2 es una comunicación entre dos *SNMPv2 Party* sobre la información de administración que está contenida en un contexto SNMPv2 accesible por la entidad SNMPv2 apropiada. En particular, esta puede ser:

- Un query originado por un *party* sobre la información accesible del *party* destinatario, por ejemplo: getRequest, getNextRequest o getBulkRequest.
- Una aserción indicativa al *party* destinatario sobre la información accesible al *party*que lo originó, por ejemplo: Response, InformRequest, o SNMPv2-Trap.
- Una aserción imperativa por el *party* originario sobre la información accesible del *party* destinatario, por ejemplo: setRequest.
- Una confirmación a un *party* destinatario sobre la información recibida por el *party* originario, por ejemplo: un Response confirmando un InformRequest.

Cada gestión de comunicación es representada por un ASN.1 y posee los siguientes componentes:

- **dstParty**: Es el destino e identifica el *SNMPv2 Party* a quien está dirigida la comunicación.
- **srcParty**: Es el origen e identifica el *SNMPv2 Party* que origina la comunicación.
- **context**: Identifica el contexto SNMPv2 que contiene la referencia de la información administrada por la comunicación.
- **pdu**: Es la forma y significado del mensaje.

3.2.2. Especificación de las nuevas PDUs en SNMPv2

En SNMPv2 se mantienen algunas de las PDUs que se generaban en su versión antecesora, considerando algunos cambios en la interpretación de los campos. En la segunda versión del protocolo se agregan dos nuevas PDUs y la trama *Trap* se modifica tomando los mismos campos de un mensaje general interpretando los campos de forma distinta.

- **GetBulkRequest**: El propósito de esta PDU es transmitir gran cantidad de data, incluyendo, pero no limitando, la recuperación eficiente y rápida de largas tablas.
- **InformRequest**: Es usado para notificar la recepción de una notificación cuando ha ocurrido un evento o una condición está presente. Este es un mecanismo de notificación de entrega confirmada, aunque no es, por supuesto, garantía de entrega. Esta también permite la comunicación entre estaciones de administración.

- **SNMPv2-Trap**: Esta cumple la misma función que su versión antecesora, generando notificaciones cuando ocurra un evento o exista una condición.

En el RFC 3416 [4] se define otro PDU llamado *Report* el cual no es especificado por completo. Su uso y semántica queda de parte de los implementadores.

3.2.3. Formatos de las PDUs y Mensajes de SNMPv2

Como se mencionó anteriormente las PDUs en esta versión del protocolo poseen la misma estructura general para su formato, la cual es conservada de su versión antecesora, como se puede observar en la Figura 3.5.

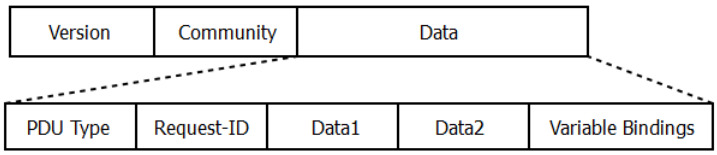

Figura 3.5: Mensaje SNMPv2

La interpretación que se le da a los campos `Data1` y `Data2` depende del PDU que se esté generando/recibiendo. Para las PDUs getRequest, getNextRequest, setRequest, response, SNMPv2-Trap e informRequest se tiene como interpretación de estos campos la información de error que ocurre en el procesamiento de un mensaje SNMPv2 o generación de un SNMPv2-Trap. En la Figura 3.6 se puede ver el formato, el cual se mantiene según su versión anterior.

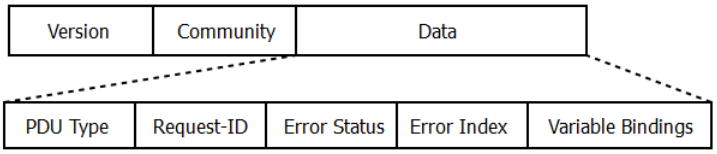

Figura 3.6: PDUs GetRequest, GetNextRequest, Response, SetRequest y SNMPv2-Trap de SNMPv2

Los errores (campo `Error Status`) son ampliados o más específicos en esta versión del protocolo y se encuentran en la Tabla 3.2.

Tipo de Dato	Descripción
0	noError
1	tooBig
2	noSuchName
3	badValue
4	readOnly
5	genErr
6	noAccess
7	wrongType
8	wrongLength
9	wrongEncoding
10	wrongValue

11	noCreation
12	inconsistentValue
13	resourceUnavailable
14	commitFailed
15	undoFailed
16	authorizationError
17	notWritable
18	inconsistentName

Tabla 3.2: Valores del Campo Error-Status en SNMPv2

Para el PDU getBulkRequest, esta interpretación cambia y se agregan dos nuevos conceptos en el procesamiento de los mensajes SNMPv2. Estos son: **non-repeaters** y **max-repetitions** (ver Figura 3.7).

- **non-repeaters**: Es un entero que indica que sobre las primeras N variables bindings no se realizarán solicitudes sucesivas.
- **max-repetitions**: Es un entero que indica el número de solicitudes sucesivas que se harán sobre las R variables bindings restantes. Siendo R=(L-N), donde L es el número total de variables bindings de la lista.

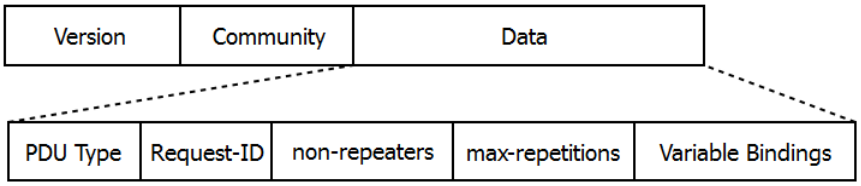

Figura 3.7: PDU getBulkRequest de SNMPv2

3.3. Simple Network Management Protocol Version 3

En esta nueva versión del protocolo se definen nuevas características trayendo mejoras y sobre todo cumpliendo metas y objetivos que en las versiones anteriores no se habían cumplido del todo. Para esta nueva versión se soporta un nuevo formato de mensaje SNMP, se logra la seguridad en los mensajes, se posee un acceso de control y los parámetros SNMP pueden ser configurados remotamente.

3.3.1. Elementos de la Arquitectura

Una entidad SNMP es una implementación de esta arquitectura. Cada una de ellas consiste en un SNMP engine y una o más aplicaciones asociadas a esta. Un SNMP engine provee servicios de envío y recepción de menajes, cifrado y autenticación de mensajes, y control de acceso a los objetos administrados. Entre una entidad SNMP y un SNMP engine existe una asociación uno-a-uno, la cual contiene los siguientes componentes listados a continuación y descritos en el RFC 3411 [5] (ver Figura 3.8):

- Despachador.
- Subsistema de procesamiento de mensaje.
- Subsistema de seguridad.
- Subsistema de control de acceso.

En un dominio administrativo, un snmpEngineID es un identificador único e inequívoco de un SNMP Engine. Como existe una asociación uno-a-uno con la entidad SNMP, entonces esta se puede reconocer de igual manera con este identificador.

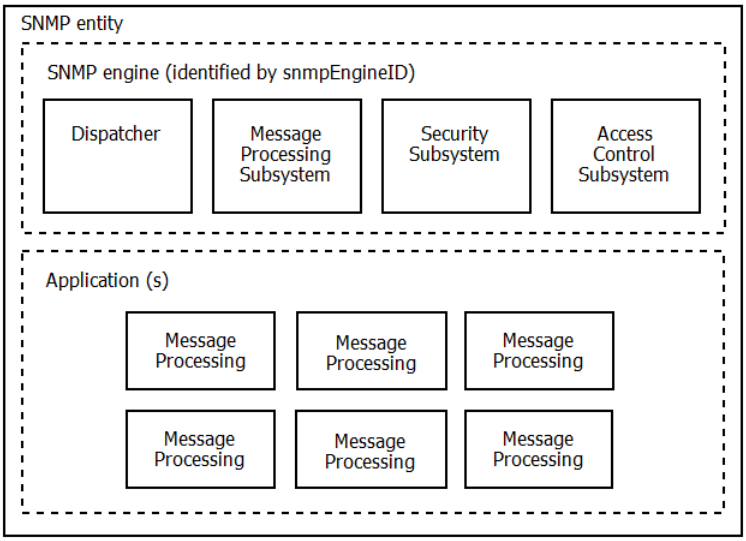

Figura 3.8: Entidad SNMP en SNMPv3

Para un SNMP Engine existe un solo despachador, permitiendo soportar múltiples versiones de SNMP, haciéndolo a través de:

- El envío y recepción de mensajes SNMP.
- Determinando la versión del mensaje SNMP e interactuando con el modelo de procesamiento correspondiente.
- Proveyendo una interfaz abstracta para las aplicaciones SNMP para la entrega de un PDU a una aplicación.
- Proveyendo una interfaz abstracta para las aplicaciones SNMP que permita a estas enviar un PDU a una entidad SNMP remota.

El subsistema de procesamiento de mensajes es responsable de la preparación de mensajes a enviar y de la extracción de la data en los mensajes recibidos. Este está conformado por múltiples modelos de procesamiento de mensajes como se observa en la Figura 3.9. Estos modelos definen un formato de una versión en particular de un mensaje SNMP y coordina la preparación y extracción de cada mensaje según un formato de una versión específica.

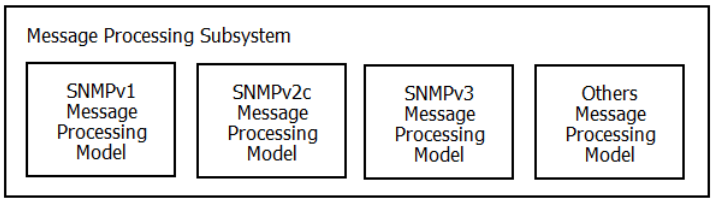

Figura 3.9: Subsistema de Procesamiento de Mensajes en SNMPv3

El subsistema de seguridad provee los servicios de seguridad tales como la autenticación y la privacidad de mensajes. Este está compuesto por múltiples modelos de seguridad que especifican las amenazas contra las que protege, los objetivos de sus servicios, así como los protocolos de seguridad para proporcionar la seguridad. Los protocolos de seguridad especifican mecanismos, procedimientos y objetos MIB usados para proveer los servicios de autenticación y privacidad. En la Figura 3.10 se ve como se compone el subsistema de seguridad.

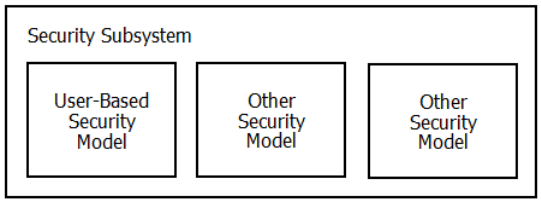

Figura 3.10: Subsistema de Seguridad en SNMPv3

El subsistema de control de acceso provee el servicio de autenticación por medio de uno o más modelos de control de acceso. Un modelo de control de acceso define un acceso particular en función de dar soporte a los derechos de acceso. En la Figura 3.11 se puede ver la composición del subsistema por los modelos de control de acceso.

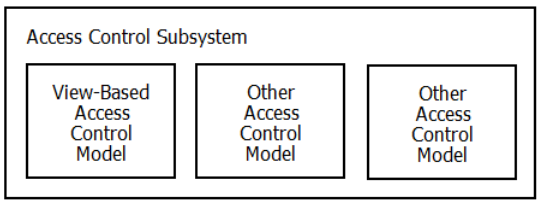

Figura 3.11: Subsistema de Control de Acceso

Para esta versión del protocolo se pueden ver distintos tipos de aplicaciones, incluyendo:

- Generadores de Comandos: Manipula y monitorea la data administrada.
- Respondedores de Comandos: Proveen el acceso a la data administrada.
- Originadores de Notificaciones: Inician un mensaje asíncrono.
- Receptores de Notificaciones: Procesan los mensajes asíncronos.
- Reenviadores Proxy: Reenvían mensajes entre entidades.

Todas estas aplicaciones hacen uso de los servicios que provee el *SNMP engine*.

3.3.2. Formato del Mensaje en SNMPv3

Para SNMPv3, se agregan varios campos para prestar los servicios de autenticación y privacidad, incluyendo los diferentes modos de acceso. El formato del mensaje SNMPv3 es descrito por el RFC 3412 [6]. En la Figura 3.12 se puede ver la comparación de los mensajes que son generados/recibidos en las distintas versiones de SNMP.

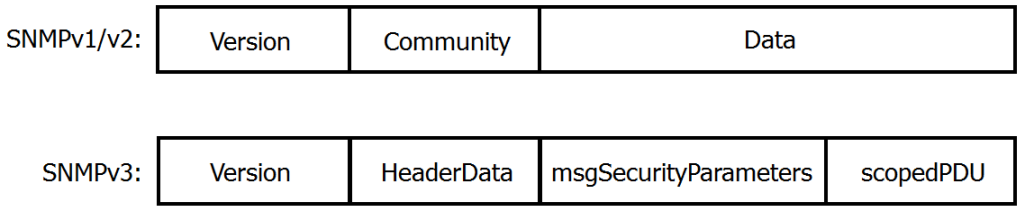

Figura 3.12: Mensajes Genéricos de SNMPv1, SNMPv2c y SNMPv3

Como se pudo observar en el mensaje SNMPv3, el primer campo del mensaje es la versión que indica que es la tercera versión del protocolo, luego se divide en tres partes las cuales son explicadas detalladamente a continuación:

- **HeaderData**:

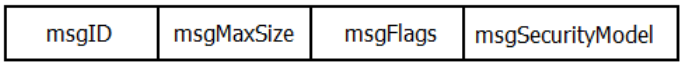

Figura 3.13: Campo HeaderData del Mensaje SNMPv3

  - **msgID**: Es usado entre dos entidades SNMP para coordinar las solicitudes y respuestas, y por el modelo procesamiento de mensaje para el procesamiento.
  - **msgMaxSize**: Es el máximo tamaño de mensaje soportado por el emisor.
  - **msgFlags**: Contiene un conjunto de bits que controlan el procesamiento del mensaje.
  - **msgSecurityModel**: Identifica qué modelo de seguridad está siendo usado por el emisor al generar el mensaje y por lo tanto qué modelo de seguridad debe usar el receptor para el procesamiento del mensaje.

- **msgSecurityParameters**: Es usado para la comunicación entre los módulos de modelos de seguridad en el envío y recepción entre SNMP engine. La data en este campo es usada exclusivamente por el modelo de seguridad, y el contenido y formato de la data es definido por él. Este campo es un String el cual no es interpretado por el modelo de procesamiento de mensaje, pero es pasado a la implementación local del modelo de seguridad indicado por el campo `msgSecurityModel` en el mensaje.
- **scopedPdu**:

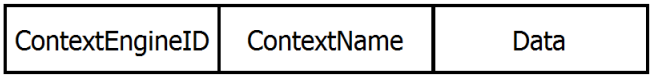

Figura 3.14: Campo scopedPDU del Mensaje SNMPv3

  - **contextEngineID**: Identifica de forma única, dentro de un dominio administrativo, donde una entidad SNMP puede darse cuenta de una instancia de un contexto particular con un contextName.

- **contextName**: Es la conjunción del contextEngineID que identifica un contexto en particular asociado con la información administrada contenida en el PDU del mensaje.
- **data**: Este campo contiene el PDU que debe ser uno de los que se especifican en el RFC 3416 [4].

3.3.3. Modelo de Seguridad Basado en Usuario (USM)

Este define los elementos de procedimiento para provisionar nivel de seguridad en los mensajes SNMP. Se hace una introducción en el RFC 3410 [7] y es descrito en el RFC 3414 [8], cuyos dos primeros capítulos definen contra qué amenazas defiende el Modelo de Seguridad basado en Usuario (USM) a los mensajes SMMPv3. Ellas son: modificación de la información, suplantación de identidad, modificación de flujo de mensajes y de la divulgación.

El USM utiliza MD5 y el SHA (Secure Hash Algorithm) como algoritmo de hash para realizar el cálculo del *digest* y así proporcionar integridad de los datos con estas características:

- Protegiendo directamente la data de los ataques de modificación.
- Proveyendo indirectamente autenticación del origen.
- Defendiendo directamente de los ataques de suplantación de identidad.

El USM utiliza indicadores sincronizados de tiempo para defenderse de ciertos ataques de modificación del flujo. Mecanismos automáticos de sincronización de reloj son especificados en el protocolo, sin dependencia de la hora de terceros y consideraciones de seguridad concomitantes.

Para el cifrado, el USM utiliza Data Encryption Standard (DES) con el modo de encadenamiento de bloques (CBC) si la protección de divulgación es deseada. El soporte de DES en el USM es opcional, principalmente debido a las restricciones de ciertos países en el uso de productos que incluyen tecnología criptográfica.

3.3.4. Control de Acceso Basado en Vista (VACM)

El propósito del RFC 3415 [9] es describir el Modelo de Control de Acceso basado en Vistas (VACM) para el uso en la arquitectura SNMP. El VACM puede estar asociado simultáneamente a una simple implementación con múltiples modelos de procesamiento de mensaje y múltiples modelos de seguridad, lo cual no es muy común.

El VACM define un conjunto de servicios que una aplicación puede usar para chequear el derecho de acceso. El modelo está compuesto por los siguientes elementos:

- **Groups**: Es un conjunto de cero o más tuplas (securityModel, securityName) en cuyo nombre los objetos de administración de SNMP se puede acceder (autentificación-privacidad, autentificación-no privacidad, y no autentificación-no privacidad).
- **SecurityLevel**: Identifica el nivel de seguridad que será asumido cuando se chequee los derechos de acceso.
- **Context**: Es una colección de información administrada accesible por una entidad SNMP.
- **MIB Views and View Families**: Las vistas MIB son un conjunto de variables de una MIB. Las familias MIB es una estructura que agrupa muchas vistas.

- **Access Policy**: La política de acceso define los derechos de acceso a los objetos, estos pueden ser: `read-view`, `write-view` y `notify-view`.

En la Figura 3.15 se puede observar cómo es tomada la decisión de acceso por el VACM.

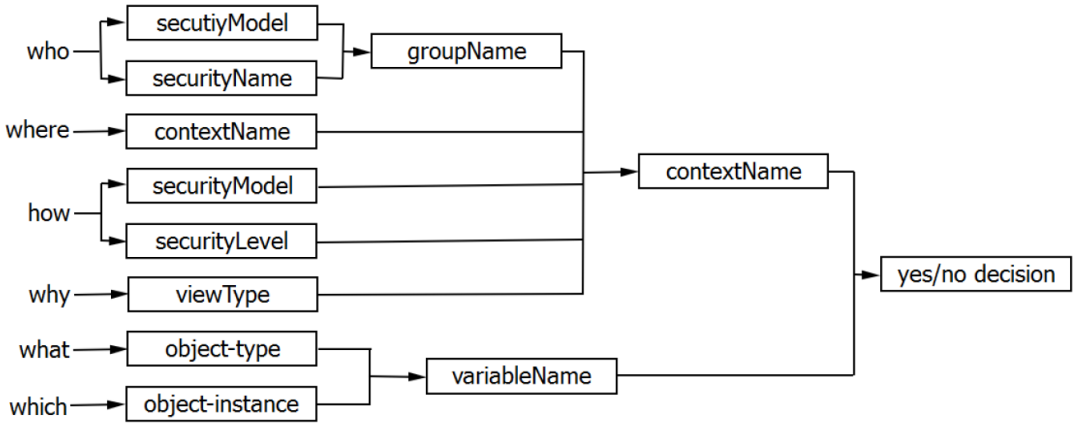

Figura 3.15: Proceso de Acceso Realizado por el VACM

Capítulo 4

Structure of Management Information

Para que los agentes y estaciones de administración intercambien data, ambos deben entenderla, independientemente de la forma de representar los datos internamente en las máquinas. Para que esto ocurra dos ítems deben ser estandarizados: el `abstract syntax` y el `transfer syntax`. El `abstract syntax` define la especificación para la notación de la data. El `transfer syntax` define la codificación para los elementos en el `abstract syntax`.

Para el `abstract syntax` de los mensajes se utiliza ASN.1, que define un lenguaje básico para los elementos y provee reglas para combinar los elementos en el mensaje. Las BERs (*Basic Encoding Rules*) proveen el `transfer syntax`. Las BERs están asociadas con el `abstract syntax` y provee una comunicación a nivel de bit entre las máquinas.

4.1. Structure of Management Information Version 1

Los objetos administrados son accedidos a través de un almacén virtual de información, denominado MIB (*Management Information Base*). Cada tipo de objeto (llamado `object type`) tiene un nombre, una sintaxis y una codificación. El nombre está representado únicamente como un `object identifier`. Un `object identifier` es un nombre asignado administrativamente. La sintaxis para un `object type` define la estructura de datos abstracta correspondiente. Por ejemplo, la estructura de un `object type` dado podría ser un *Integer* u *Octet String*. La codificación de un `object type` es simplemente como se representa usando la sintaxis del `object type`. Implícitamente se tiene la noción de que la codificación y la sintaxis están atadas, ya que esta representación se usa cuando se transmiten en la red.

Los nombres son usados para identificar objetos administrados, estos por naturaleza son jerárquicos. El concepto de `object identifier` es usado para modelar esta noción, el cual puede ser usado para propósitos distintos en el nombramiento de los tipos de objetos administrados. Un `object identifier` es una secuencia de enteros que recorre un árbol global. El árbol posee una raíz conectada a través de aristas a un número de nodos etiquetados, donde estos a su vez pueden tener hijos igualmente etiquetados, de manera que se pueden llamar a los hijos subárbol. Esta estructura puede continuar arbitrariamente hasta un nivel de profundidad. Una etiqueta es un par formado por un texto que corresponde a la descripción y un entero.

La raíz del árbol no posee una etiqueta, pero tiene por lo menos tres hijos directos: ccitt(0), iso(1) y joint-iso-ccitt(2). Bajo el nodo iso(1) existe un subárbol para el uso de las organizaciones

nacionales e internacionales, org(3). Seguido de este hay un hijo (subárbol) que es administrado por el Departamento de Defensa de los EEUU, dod(6). Para la comunidad del Internet el RFC 1155 [10] asume que el DoD asigna un nodo para esta y que es administrada por la *Internet Activities Board* (IAB). A continuación en el Ejemplo 1 se ve cómo se define el `object identifier` de la comunidad y se verán sus hijos en la Figura 4.1.

Ejemplo 1

```
internet        OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
```

Todos los object identifier del subárbol Internet comienzan con el prefijo: "1.3.6.1", inicialmente este presenta cuatro nodos definidos en el Ejemplo 2.

Ejemplo 2

```
directory       OBJECT IDENTIFIER ::= { internet 1 }
mgmt            OBJECT IDENTIFIER ::= { internet 2 }
experimental    OBJECT IDENTIFIER ::= { internet 3 }
private         OBJECT IDENTIFIER ::= { internet 4 }
```

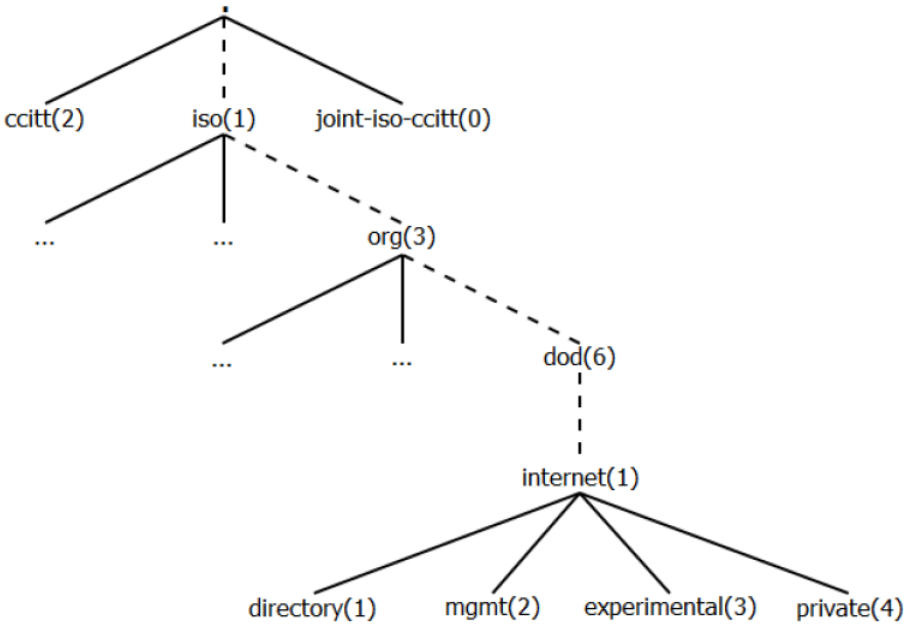

Figura 4.1: Recorrido al Subárbol internet(1)

La sintaxis es usada para definir la estructura correspondiente del `object type`. Un subconjunto de las construcciones de ASN.1 son las que definen esta estructura. El tipo `Object Syntax` define las diferentes sintaxis que pueden ser usadas en la definición de un `object type`. Los tipos primitivos que se permiten son los listados a continuación:

- *Interger*.
- *OctetString*.
- *ObjectIdentifier*.
- *Null*.

El tipo de constructor sequence es permitido, para así poder generar listas o tablas. Para las listas la sintaxis se muestra en el Ejemplo 3.

Ejemplo 3

```
SEQUENCE { <type1>,..., <typeN>}
```

donde cada `<type>` representa uno de los tipos primitivos de ASN.1. Para las tablas la sintaxis se muestra en el Ejemplo 4.

Ejemplo 4

```
SEQUENCE OF <entry>
```

donde `<entry>` es un constructor de lista. Algunas veces, las listas y tablas son referidas como tipos agregados.

En la Tabla 4.1 se muestran los tipos de datos definidos en SMIv1.

Tipo de Dato	Descripción
Integer	Representa números enteros positivos como negativos.
OctetString	Representa datos binarios para ser transmitidos en múltiplos de 8 bits.
Counter	Representa un entero no negativo que incrementa su valor hasta alcanzar el máximo $2^{32} - 1$.
Gauge	Representa un entero no negativo que incrementa o decrece. Su valor máximo es $2^{32} - 1$.
TimeTicks	Representa un entero no negativo que cuenta el tiempo en centésimas de segundos.
Opaque	Representa un valor que puede ser codificado como un `Octet String`.
IpAddress	Representa una dirección IPv4 de 32 bits. Es representado como un `Octet String` de longitud 4.
NetworkAddress	Representa una dirección de una posible familia de protocolos.

Tabla 4.1: Tipos de Datos en SMIv1

4.2. Structure of Management Information Version 2

Para esta nueva versión de SMI, descrito en el RFC 2578 [11] , se crean nuevos tipos de datos y nuevas descripciones opcionales en la estructura para tener una definición más específica del objeto administrado. Con respecto a los tipos de datos agregados para la segunda versión de SMI se tienen los siguientes presentados en la Tabla 4.2.

Tipo de Dato	Descripción
Integer32	Igual al tipo de datos Integer.
Counter32	Igual al tipo de dato Counter.
Counter64	Representa un entero no negativo que incrementa su valor hasta alcanzar el máximo $(2^{64} - 1)$.
Gauge32	Igual al tipo de dato Gauge.
Unsigned32	Representa números decimales de 0 a $2^{32} - 1$.
Bits	Representa una enumeración de enteros no negativos llamados bits

Tabla 4.2: Tipos de Datos en SMIv2

Los otros tipos de datos que existen en SMIv1 se mantienen, agregando los antes mencionados, para así dar soporte a ambas versiones en esta. Para la definición de los objetos en SMIv2 se agregan nuevas descripciones que se definen a continuación:

- **Units**: Campo opcional que contiene una definición textual de las unidades asociadas con el objeto (Ejemplo: segundos, milisegundos).
- **Max-Access**: Campo que debe estar presente, el cual sirve para identificar el máximo acceso sobre el objeto, pudiendo este ser leído, escrito o creado, o incluir su valor en una notificación. Sus valores de menor a mayor en el sentido de acceso son: `not-accessible`,`accessible-for-notify`, `read-only`, `read-write`, `read-create`.
- **Status**: Campo que debe estar presente. Este define si el objeto es actual o histórico. Es extendido agregando nuevos valores para él. (Ejemplo: `current`, `obsolete` o `deprecated`)
- **Augments**: Campo opcional que permite agregar una o más columnas a una tabla, representada por algún otro objeto.

Otras de las características agregadas en SMIv2 es que el árbol se extendió agregando nuevos campos en el nodo internet(1) como se ve en la Figura 4.2.

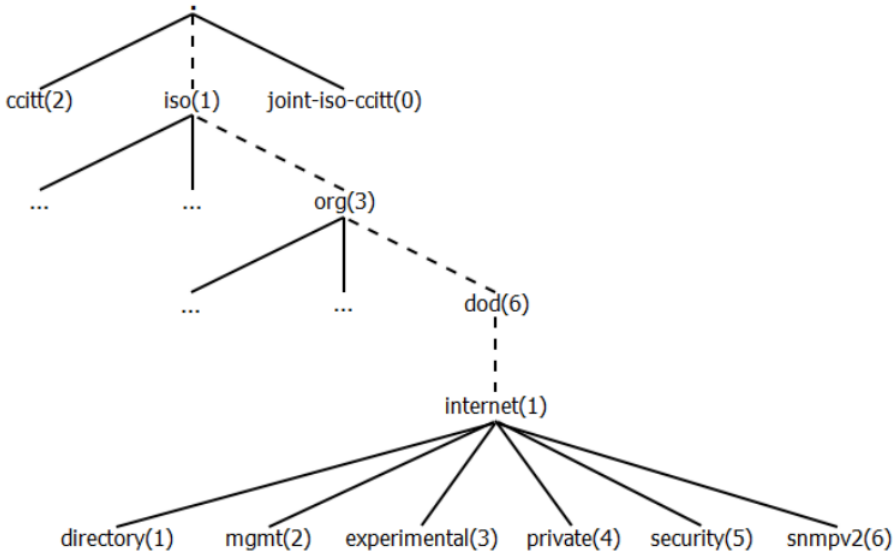

Figura 4.2: Recorrido al Subárbol internet(1) en SMIv2

Capítulo 5

Management Information Base

La MIB es un almacén virtual de información que contiene los objetos que son administrados. Su estructura es jerárquica (forma de árbol). Actualmente existen dos versiones de MIBs, el RFC 1156 [12] define la MIB-I como un conjunto de objetos para la suite de protocolos de Internet y el RFC 1213 [13] define una evolución de la MIB-I basada en experiencias de implementaciones y nuevos requerimientos operacionales (MIB-II).

5.1. Management Information Base version 1

La MIB-I, como se menciona antes, está definida en el RFC 1156 [12], el cual fue publicado en mayo de 1990. Esta fue dividida en ocho (8) grupos, simplificando así la asignación y aplicación de los OIDs. A continuación se describen los objetos, los grupos de objetos y la forma en que se definen los mismos, que son especificados en el documento antes mencionado.

Los objetos son definidos usando ASN.1 y los mecanismos usados para describirlos son especificados en la SMI [10]. Cada objeto posee un nombre, una sintaxis y una codificación, que especifican un `object type`. Si desea profundizar más sobre estos campos diríjase al Capítulo 4.

Los grupos de objetos son un conjunto de objetos que contienen solo los elementos esenciales. Para la MIB-I los grupos definidos son los siguiente: System, Interfaces, Address Translation, IP, ICMP, TCP, UDP y EGP. Existen dos razones para definir estos grupos:

- Proveer un método de asignación de OID.
- Proporcionar un método para las implementaciones de los agentes administrados a saber qué objetos deben implementar.

Se describe a continuación el formato de definición y debe poseer los siguientes campos:

- **Object**: Nombre textual para el `object type`, según su correspondiente `object identifier`.
- **Syntax**: Es el `abstract syntax` para el `object type`, usando ASN.1.
- **Definition**: Es la descripción semántica del `object type`.
- **Access**: Sus posibles valores son `read-only`, `read-write`, `write-only` o `not-accessible`.
- **Status**: Sus posibles valores son `mandatory`, `optional` u `obsolete`.

A continuación en el Ejemplo 5 se muestra la definición de la MIB-I según el RFC 1156 [12].

Ejemplo 5

```
RFC1156-MIB
DEFINITIONS ::= BEGIN
IMPORTS
mgmt, OBJECT-TYPE, NetworkAddress, IpAddress,
                   Counter, Gauge, TimeTicks
                       FROM RFC1155-SMI;
        mib        OBJECT IDENTIFIER ::= { mgmt 1 }
        system     OBJECT IDENTIFIER ::= { mib 1 }
        interfaces OBJECT IDENTIFIER ::= { mib 2 }
        at         OBJECT IDENTIFIER ::= { mib 3 }
        ip         OBJECT IDENTIFIER ::= { mib 4 }
        icmp       OBJECT IDENTIFIER ::= { mib 5 }
        tcp        OBJECT IDENTIFIER ::= { mib 6 }
        udp        OBJECT IDENTIFIER ::= { mib 7 }
        egp        OBJECT IDENTIFIER ::= { mib 8 }
END
```

5.2. Management Information Base version 2

La MIB-II definida en el RFC 1213 [13] es una evolución de la versión antecesora e incluye las siguientes características:

- Adiciones incrementales para reflejar los nuevos requerimientos operativos.
- Compatibilidad hacia atrás entre SMI/MIB y SNMP.
- Soporte mejorado para múltiples entidades de protocolo.
- Limpieza textual de la MIB para dar más claridad y legibilidad.

Para la segunda versión se hereda el prefijo obteniendo la definición mostrada en el Ejemplo 6.

Ejemplo 6

```
mib-2    OBJECT IDENTIFIER ::= { mgmt 1 }
```

La Figura 5.1 muestra cómo es el recorrido o prefijo de los `object identifier`, también se puede observar que han sido agregados nuevos grupos: `transmission` y `snmp`.

En el Ejemplo 7 se muestran algunas definiciones de objetos de los grupos de la MIB-II

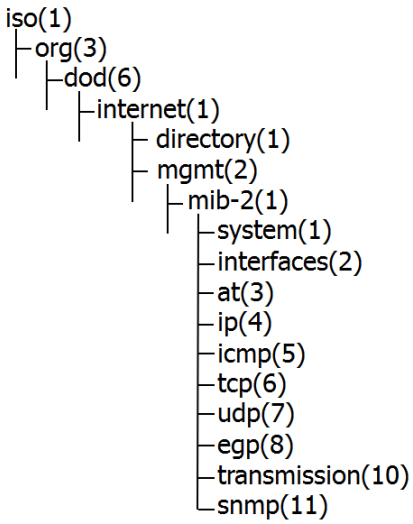

Figura 5.1: Grupos Principales de la MIB-2

Ejemplo 7

```
ifMtu OBJECT-TYPE
       SYNTAX INTEGER
       ACCESS read-only
       STATUS mandatory
       DESCRIPTION
            "The size of the largest datagram which can be
            sent/received on the interface, specified in
            octets. For interfaces that are used for
            transmitting network datagrams, this is the size
            of the largest network datagram that can be sent
            on the interface."
::= { ifEntry 4 }

ifPhysAddress OBJECT-TYPE
       SYNTAX PhysAddress
       ACCESS read-only
       STATUS mandatory
       DESCRIPTION
            "The interface's address at the protocol layer
            immediately 'below' the network layer in the
            protocol stack. For interfaces which do not have
            such an address (e.g., a serial line), this object
            should contain an octet string of zero length."
::= { ifEntry 6 }
```

En la Tabla 5.1 se estudiará cada uno de los grupos de la MIB-II, donde se muestra su prefijo y función para una mayor comprensión.

Grupo	Prefijo	Función
system	1.3.6.1.2.1.1	Sirve para obtener información del sistema.
interfaces	1.3.6.1.2.1.2	Permitir realizar un monitoreo de cada interfaz (up, down, etc).
at	1.3.6.1.2.1.3	Este grupo es considerado como obsoleto. Se mantiene aún por compatibilidad.
ip	1.3.6.1.2.1.4	Permitir obtener información sobre el enrutamiento y otros aspectos a nivel de IP.
icmp	1.3.6.1.2.1.5	Proporcionar información sobre los errores ICMP.
tcp	1.3.6.1.2.1.6	Sirve para monitorear el estado de las conexiones TCP, entre otros aspectos.
udp	1.3.6.1.2.1.7	Sirve para obtener información estadística de UDP.
egp	1.3.6.1.2.1.8	Proporcionar datos estadísticos del protocolo EGP.
transmission	1.3.6.1.2.1.10	Actualmente no existen objetos definidos para este grupo.
snmp	1.3.6.1.2.1.11	Permitir monitorear la ejecución de la aplicación SNMP en la entidad gestionada.

Tabla 5.1: Grupos Definidos para la MIB-II

Capítulo 6

Dispositivos Móviles y sus Sistemas Operativos

Los dispositivos móviles o *handheld* son computadores portátiles pequeños que pueden ser guardados en un bolsillo y caben perfectamente en la mano mientras se utilizan. Estos dispositivos poseen capacidad de procesamiento, memoria, mantenimiento de conexiones a diferentes tipos de redes, las cuales son limitadas. En principio el uso de estos dispositivos era específico debido a sus capacidades, pero esto ha cambiado con el surgimiento de nuevas tecnologías.

Los dispositivos móviles actuales poseen mayor capacidad, tanto de procesamiento como de almacenamiento. Estos dieron un gran cambio en la interacción con el usuario, en principio pantallas y botones pequeños, ahora existen dispositivos con pantallas táctiles, sensores de movilidad, acercamiento, etc. En el mercado hay una gran variedad de modelos y tamaños, los cuales tienen un sistema operativo que es el encargado de gestionar el hardware y aplicaciones que ejecutan estos.

Para el 2005, T38 y DuPont Global Mobility Innovation Team [14] propusieron tres estándares para la definición de los dispositivos móviles:

- **Dispositivos Móviles de Datos Limitados**: Tienen tamaño pequeño con pantallas basadas en texto, poseen servicios de SMS y WAP generalmente limitados.
- **Dispositivos Móviles de Datos Básicos**: Poseen pantallas medianas con navegación basada en íconos por medio de un cursor, ofrecen servicios SMS, acceso a e-mail, lista de direcciones y navegador web básico.
- **Dispositivos Móviles de Datos Mejorados**: Se caracteriza por tener pantallas de medianas a grandes con navegación de tipo *stylus*, ofreciendo los mismos servicios de los dispositivos móviles de datos básicos, agregando aplicaciones nativas ofimáticas y corporativas.

A continuación se describirán algunas características de los sistemas operativos que se ejecutan en dispositivos móviles con el fin de realizar una comparación para definir la plataforma de desarrollo, la información fue extraída de las fuentes [15] y [16].

6.1. Symbian OS

Symbian es un sistema operativo que fue producto de la alianza de varias empresas de telefonía móvil. Sus orígenes provienen de su antepasado EPOC32, utilizado en PDAs y handhelds de PSION. El 24 de junio de 2008, Nokia compra Symbian en su totalidad, tras un acuerdo con el resto de los

socios. El objetivo era establecer la Fundación Symbian y convertir este OS en una plataforma abierta.

Este objetivo en Symbian no fue completado, ya que su “código abierto” está disponible solo para usuarios registrados y solo puede ser modificado por sus asociados en la central de desarrollo en Japón.

En las opciones de desarrollo, Symbian posee un SDK (Software Development Kit) que permite desarrollar aplicaciones en C++ y Java utilizando la biblioteca QT.

Hay que resaltar que Nokia dejó su desarrollo a mediados de enero de 2013, eliminando el acceso a toda la información relacionada con este.

6.2. iOS de Apple

iOS es un sistema operativo móvil desarrollado por Apple originalmente para el iPhone, luego ha sido usado en los iPods e iPads. iOS es derivado de MAC OS X, que a su vez está basado en Darwin BSD. Oficialmente no se puede instalar ninguna aplicación que no esté firmada por Apple.

iOS tiene 4 capas de abstracción: (1) la capa del núcleo del sistema operativo, (2) la capa de “Servicios Principales”, (3) la capa de “Medios de comunicación” y (4) la capa de “CocoaTouch”. Todo el sistema se encuentra en la partición “/root” del dispositivo, ocupa poco menos de 500 megabytes.

Este sistema operativo ofrece como medio de desarrollo el kit iPhone SDK para realizar aplicaciones nativas. El lenguaje de programación principal es Objetive-C, un lenguaje Orientado a Objetos basado en C. El SDK puede descargarse gratis, pero para publicar la aplicación es necesario registrarse en el Programa de Desarrollo del iPhone.

6.3. BlackBerry OS

BlackBerry OS es un sistema operativo móvil desarrollado por RIM (*Research in Motion*) para sus dispositivos BlackBerry. BlackBerry OS es un sistema orientado al uso profesional y eficaz de gestión de correo electrónico, medios de comunicación instantáneos y virtuales, IM, y gestión de agendas.

Las opciones de desarrollo están limitadas por una firma digital y una cuenta de desarrollador en RIM, para así poder tener acceso a ciertas funcionalidades. La programación está basada en el lenguaje de programación Java usando MIDP 2.0 y CLDC 1.1.

6.4. Android

Android es un sistema operativo para dispositivos móviles basado en Linux. Inicialmente fue desarrollado por Android Inc., una firma comprada por Google en el 2005. Su desarrollo se extendió para soportar dispositivos como tablets, reproductores MP3, netbooks, PCs, televisores, etc.

Android es el principal producto de la Open Handset Alliance. Posee una tienda de aplicaciones en línea administrada por Google llamada Play Store, pero también tiene la posibilidad de obtener

software externamente. Google liberó la mayoría del código de Android bajo licencia Apache.

Las aplicaciones se ejecutan sobre una máquina virtual Dalvik con compilación en tiempo de ejecución. El lenguaje de programación por defecto es Java pudiendo también programar en C/C++ a través de la JNI (Java Native Interface).

6.5. Windows Phone

Windows Phone (anteriormente llamado Windows Mobile) es un sistema operativo móvil desarrollado por Microsoft diseñado para ser usado en teléfonos inteligentes y otros dispositivos móviles. Se basa en el núcleo del sistema operativo Windows CE y cuenta con un conjunto de aplicaciones básicas utilizando las APIs de Microsoft Windows.

El desarrollo de aplicaciones se realiza con Windows Mobile 6 SDK (Standard o Professional), el mismo puede ser utilizado en la plataforma Visual Studio .NET. Ambos kits son gratuitos pero no la plataforma de desarrollo Visual Studio .NET.

6.6. Comparación de los Sistemas Operativos para Dispositivos Móviles

En esta sección se puede ver en la Tabla 6.1 la comparación de algunos OS para dispositivos móviles que se mencionaron en secciones anteriores, donde se muestran unas variables que podrían o no soportar los OSs en los dispositivos.

SO	Symbian		iOS		BlackBerry		Android		W. Phone	
Variable	Val	Ptos	Val	Ptos	Val	Ptos	Val	Ptos	Val	Ptos
Soporte	Sí	10	Sí	10	Sí	10	Sí	10	Sí	10
Copy/Paste	Sí	10	Sí	10	Sí	10	Sí	10	No	0
Multitarea	Sí	10	Sí	10	No	0	Sí	10	No	0
Soporta Flash	Sí	10	No	0	Sí	10	Sí	10	No	0
Soporta HTML5	No	0	Sí	10	No	0	Sí	10	No	0
Administrador de e-mail	Sí	10	Sí	10	Sí	10	Sí	10	Sí	10
Navegador de Internet	Sí	10	Sí	10	Sí	10	Sí	10	Sí	10
Memoria expansible	Sí	10	Sí	10	Sí	10	Sí	10	Sí	10
Manejo de archivos	Sí	10	Sí	10	Sí	10	Sí	10	Sí	10
Organización de Apps	Sí	10	Sí	10	Sí	10	Sí	10	Sí	10
App Store	Sí	10	Sí	10	Sí	10	Sí	10	Sí	10
App dev	Sí	10	Sí	10	Sí	10	Sí	10	Sí	10
Widgets	No	0	No	0	Sí	10	Sí	10	Sí	10
Media sync	Sí	10	Sí	10	Sí	10	Sí	10	Sí	10

Total	-	120	-	120	-	120	-	140	-	100

Tabla 6.1: Comparación de OS para Dispositivos Móviles

Como se observó en la Tabla 6.1, el sistema operativo Android es el que posee mejores características y mayor soporte, esto lo deja como el mejor de la comparación y como opción definitiva en el desarrollo de esta propuesta de Trabajo Especial de Grado.

Aunado a esto, en la Figura 6.1 (tomado de [17]) se muestran unas estadísticas realizadas por StatCounter que demuestra nuevamente que la plataforma Android es la que posee más auge y uso en los últimos años.

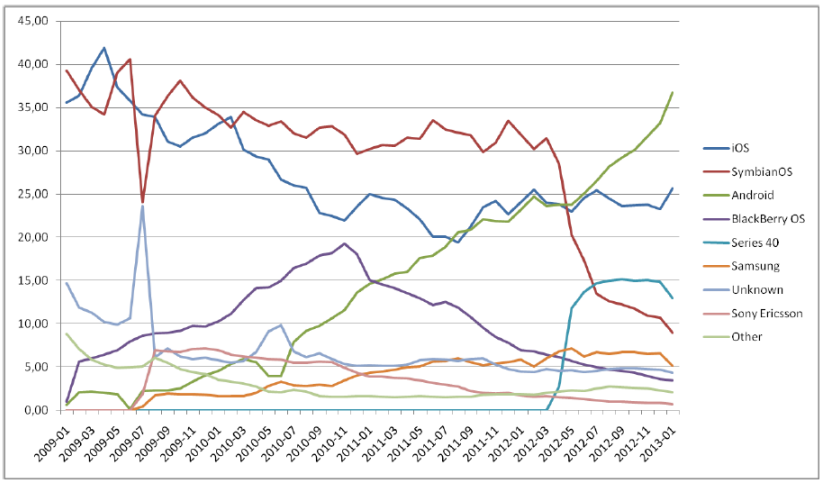

Figura 6.1: Comparación de OS para Dispositivos Móviles en el Mercado

Capítulo 7

Plataforma Android

Android es una plataforma de software *Open Source* elaborada para dispositivos móviles basada en el kernel de Linux 2.6, con interfaz de usuario, aplicaciones de usuario, bibliotecas, frameworks de aplicaciones, soporte multimedia y mucho más. Los componentes del sistema operativo están escritos en C/C++, en cambio las aplicaciones de usuario están hechas en el lenguaje de programación Java.

La plataforma es mantenida y desarrollada por Google y la Open Handset Alliance, un conjunto de empresas fabricantes de dispositivos y operadoras. La licencia de distribución es Apache 2.0, lo que la convierte en un software de libre distribución. Uno de los principales objetivos de Android es reunir todos los elementos necesarios en una plataforma que permitan al desarrollador controlar y aprovechar al máximo las capacidades y funciones de cualquier dispositivos móvil.

Para el desarrollo de aplicaciones sobre la plataforma se proporciona de forma gratuita un SDK[1] y la opción de un *plug-in* para el IDE Eclipse, donde se incluyen todas las APIs necesarias y un emulador integrado para su ejecución [18].

7.1. Arquitectura

La pila de software de Android está compuesta por los elementos que se muestran en la Figura 7.1 y que son descritos a continuación.

- **Linux Kernel**: Es el núcleo de servicios que son manejados por el kernel de Linux 2.6. Esta capa incluye los controladores de hardware, administración de procesos y memoria, seguridad, redes, y administración de energía. El kernel también provee una capa de abstracción entre el hardware y el resto de la pila.
- **Libraries**: Se ejecuta por encima del kernel e incluye varias bibliotecas escritas en C/C++ tales como:
    - Libc y SSL.
    - Una biblioteca para reproducción de música y video.
    - Un *surface manager* que provee la administración de la pantalla.
    - Bibliotecas gráficas que incluye SGL y OpenGL para gráficos 2D y 3D.

[1] https://developer.android.com/sdk/index.html#download

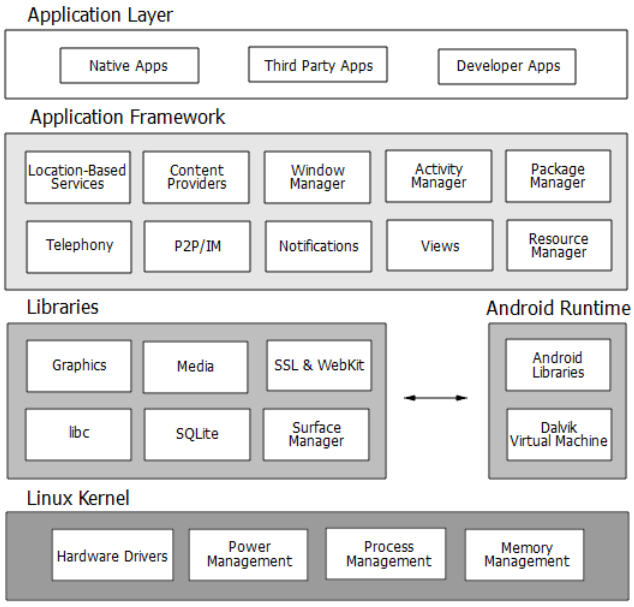

Figura 7.1: Arquitectura Android

- Soporte nativo para DB SQLite.
- Un motor WebKit para aplicaciones de tipo navegador.

- **Android Runtime**: Es el motor que impulsa las aplicaciones y, junto con las bibliotecas, es la base de la estructura de las aplicaciones. Esta capa está dividida en dos:
  - Android Libraries: Provee un conjunto de bibliotecas base que poseen las funcionalidades básicas de las bibliotecas del lenguaje de programación Java.
  - Dalvik Virtual Machine: Es una máquina virtual que ha sido optimizada para asegurar que el dispositivo corra múltiples instancias eficientemente.
- **Application Framework**: Esta provee las clases usadas para crear aplicaciones Android, proveyendo así una abstracción para el acceso al hardware, administración de la UI y los recursos de la aplicación.
- **Application Layer**: Contiene todas las aplicaciones, tanto nativas como creadas por terceros, construidas en la capa de aplicación usando un API. La application layer se ejecuta sobre el Android Runtime usando las clases y servicios puestos a disposición del application framework.

7.2. Dalvik Virtual Machine

Como se mencionó antes DVM (*Dalvik Virtual Machine*) es una máquina virtual donde se ejecutan las aplicaciones, la cual es un elemento clave de Android, ya que esta asegura múltiples instancias ejecutándose eficientemente en un simple dispositivo. Esta usa las funcionalidades de manejo de bajo nivel del kernel de Linux incluyendo la seguridad, hilos, administración de memoria y procesos. También es posible escribir aplicaciones en C/C++ que se ejecuten directamente sobre la capa del Linux OS.

Todos los accesos a los servicios del sistema y al hardware son gestionados usando Dalvik como un mediador. Mediante el uso de una máquina virtual para alojar ejecución de la aplicación,

los desarrolladores tienen una capa de abstracción que asegura que nunca tendrá que preocuparse acerca de una implementación de hardware en particular.

La DVM corre archivos ejecutables Dalvik, un formato optimizado que asegura el mínimo uso de memoria. Los ejecutables `.dex` son creados por la transformación de las clases Java (*.java) usando una herramienta disponible en el SDK.

7.3. Componentes de una Aplicación y Paquetes Android

Una aplicación para Android puede contener una combinación de componentes (*Activity*, *Service*, *Intent* o *IntentReceiver* y *Content Provider*) según lo indican [18] y [19].

Principalmente una aplicación se compone de una *activity*, por cada ventana que exista se creará una *activity*, pero para que estas funcionen se necesitan los *intents*. A continuación se ve qué papel cumplen estos 4 posibles componentes o elementos básicos de una aplicación para Android.

7.3.1. Activity

Una aplicación puede o no tener una interfaz de usuario. Si esta tiene una *user interface* (UI), entonces esta tendrá una o más *activity*. Esta es el componente principal como medio de comunicación entre la aplicación y el usuario. La *activity* emplea una o más *View* para presentar los elementos de la UI actual al usuario.

Las *Views* son elementos de UI que forman los bloques de construcción básicos de la UI. Las *Views* son jerárquicas y ellas saben cómo dibujarse a sí mismas. Cada vez que se cambie de *view* se cambiará de *activity*, esto quiere decir que si se tiene una aplicación de mensajería que posee una *view* que muestra los contactos y otra *view* para escribir el mensaje, cuando se cambie de una a otra la anterior quedará pausada y apilada en un historial para poder retornar en caso que sea necesario [20]. Para pasar de una *view* a otra, se necesita una clase llamada *Intent* que se estudia a continuación.

Para implementar una *activity*, los desarrolladores deben heredar de la clase *Activity* y ahí desarrollar la funcionalidad "principal"de la aplicación.

7.3.2. Intent e IntentReceiver

Como su nombre lo hace entender, un *intent* generalmente define la intención de hacer algún trabajo o acción. Los *intents* no siempre son invocados o creados por la aplicación, también pueden ser usados por el sistema para notificar a la aplicación de un evento específico. Este componente se implementa a través de una clase de nombre *BroadcastReceiver*.

Un *IntentReceiver* hace que se ejecute una aplicación al producirse un evento. A diferencia del *intent* anterior, la aplicación no tiene que estar corriendo para que ejecute sus *IntentReceiver*, es decir, el usuario no tiene por qué estar interactuando con el programa en ese momento.

7.3.3. Service

Un *Service* representa una aplicación sin UI, y que se ejecuta en segundo plano mientras otras aplicaciones están activas en la pantalla del dispositivo. Tiene un hilo propio y se ejecutan por tiempo

prolongado. Hay que tener en cuenta que estos no deben consumir demasiada CPU, sino acabará con la pila del dispositivo. Son implementados extendiendo de la clase *Service*.

7.3.4. Content Provider

El *Content Provider* es un mecanismo estándar para compartir información entre aplicaciones sin exponer la estructura de almacenamiento subyacente y la implementación. Con el *Content Provider* se puede almacenar datos en un archivo, en una base de datos SQLite o en cualquier otro formato que se considere.

7.3.5. Paquetes de Android

Android ofrece un gran número de APIs para desarrollar las aplicaciones [21]. A continuación se lista el núcleo de APIs que este provee y que la mayoría de los dispositivos pueden soportar.

- **android.util**: Paquete que contiene clases especializadas en contenedores, formateadores de string y utilidades de *parseo* para XML.
- **android.os**: Paquete del sistema operativo que provee los accesos a los servicios básicos, como paso de mensajes, comunicación entre procesos, funciones de reloj y depuración.
- **android.graphics**: Suministra un API gráfico para las clases que dan soporte a *canvas*, colores, primitivas de dibujo y que permite dibujar sobre los *canvas*.
- **android.text**: Provee herramientas de procesamiento de texto para mostrarlo en pantalla y *parsearlo*.
- **android.database**: Proporciona clases de bajo nivel requeridas para el manejo de cursores cuando se trabaja con bases de datos.
- **android.content**: Contiene el API usado para el acceso a la data y publicación por la prestación de servicios para hacer frente a los recursos, proveedores de contenido y paquetes.
- **android.view**: Las *view* son el núcleo para las clases con UI. Todos los UIs son construidos usando una serie de vistas para proveer los componentes de interacción con el usuario.
- **android.widget**: Construido con el paquete *View*, las clases *widget* son los elementos del UI que se pueden utilizar en las aplicaciones. Este incluye listas, botones y *layouts*.
- **com.google.android.maps**: Provee un API de alto nivel para acceder a los controles nativos de map que se puede usar en la aplicación.
- **android.app**: Suministra un API de alto nivel para acceder al modelo de la aplicación. El paquete de *app* incluye un API para los *activity* y los *Services* que constituyen la base de todas las aplicaciones para Android.
- **android.provider**: paquete que facilita el acceso del desarrollador a ciertos *Content Provider* estándar (como la base de datos de contactos), el paquete provider ofrece clases para facilitar el acceso a bases de datos estándar incluido en todas las distribuciones de Android.
- **android.telephony**: Proporciona la capacidad de interactuar directamente con la pila de teléfono del dispositivo, lo que le permite realizar, recibir y controlar las llamadas telefónicas, el estado del teléfono y mensajes SMS.
- **android.webkit**: Provee un API para trabajar con contenido Web, incluyendo el control *WebView* para exploradores embebidos en las actividades y un administrador de *cookie*.

7.4. Ciclo de Vida de las Aplicaciones

En Android cada aplicación se ejecuta en su propio proceso, resultando en beneficios de seguridad, gestión de memoria y ocupación de la CPU del dispositivo. El ciclo de vida [22] de una aplicación lo determina el sistema a partir de una combinación de estados, qué prioridad tienen, y cuánta memoria queda disponible en el sistema.

Cada proceso que se ejecuta en la plataforma es colocado en una pila. Cuando se utiliza un *activity* en primer plano, el proceso que aloja dicha *activity* se coloca en la parte superior de la pila, y el proceso anterior es bajado un espacio en la pila. Android trata de mantener los procesos en ejecución, siempre y cuando se pueda, pero no se puede mantener a cada proceso siempre en ejecución, además que la memoria del sistema es finita. Cada uno de los componentes básicos de Android tiene un ciclo de vida bien definido, para así controlar en cada momento en qué estado se encuentra dicho componente. Estos estados son manejados por métodos que posee el componente. Un ejemplo claro es mostrado en la Figura 7.2 donde se ven los distintos estados de la clases *Activity*, componente principal o más importante de una aplicación.

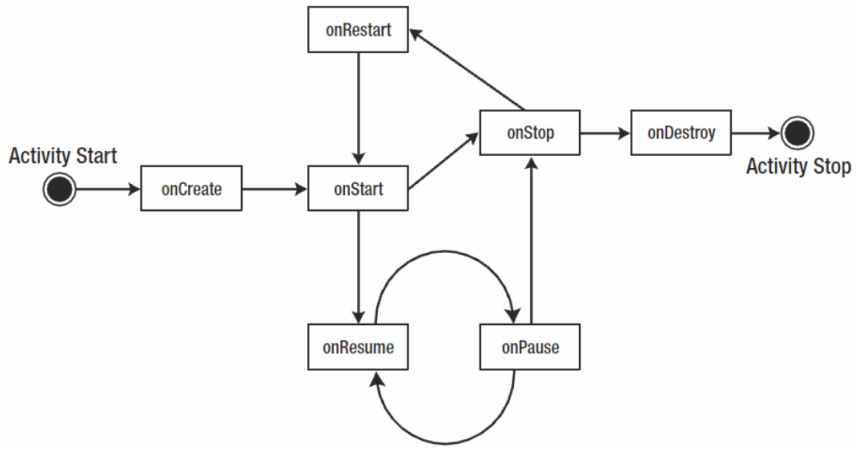

Figura 7.2: Ciclo de Vida de una Activity

A través de los métodos vistos en la Figura 7.2 se manejan los estados de una *activity* y que en la Tabla 7.1 se verá una breve descripción de los mismos.

Método	Propósito
onCreate()	Invocado cuando la activity es creada. La configuración se realiza aquí, también se proporciona acceso a cualquier estado almacenado previamente en la forma de Bundle.
onRestart()	Llamado si la *activity* se reinicia, si esta sigue en la pila, en vez de iniciarla de nuevo.
onStart()	Llamado cuando la *activity* llega a ser visible en la pantalla para el usuario.
onResume()	Invocado cuando la *activity* empieza la interacción con el usuario.
onPause()	Se le llama cuando la *activity* se detuvo, recuperando CPU y otros recursos.

onStop()	Llamado después que la *activity* ya no sea visible para el usuario.
onDestroy()	Invocado cuando una *activity* está siendo completamente removida del sistema de memoria.

Tabla 7.1: Métodos para cambiar el estado de una Activity

Como se observó en la Tabla 7.1, cada método del ciclo de vida de Android provee un propósito, estos están dentro de una "fase" particular del ciclo de vida. Se puede decir entonces que un ciclo de vida, aparte de ser administrado por métodos, tiene una fase en específico que se describe a continuación.

- ***Foreground phase***: Se refiere a cuando la *activity* se puede ver en la pantalla y encima de todo lo demás (cuando el usuario está interactuando con la *activity* para realizar una tarea).
- ***Visible phase***: Se refiere a cuando la *activity* está en la pantalla, pero esta no puede estar en la parte superior e interactuando con el usuario (por ejemplo cuando una ventana flotante o diálogo está encima de la *activity*).
- ***Entire-lifecycle phase***: Se refiere a los métodos que pueden ser invocados cuando la aplicación no está en pantalla del todo, antes de su creación o después que se ha ido antes de ser cerrada.

Estas fases son mostradas con sus respectivos estados en la Figura 7.3.

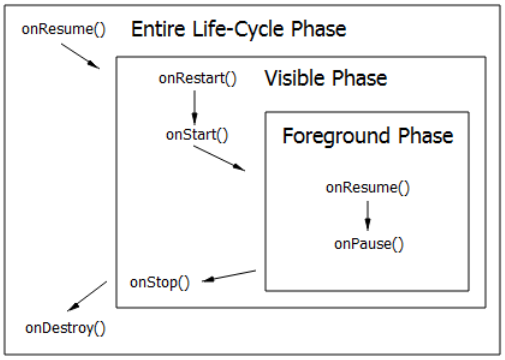

Figura 7.3: Fases del Ciclo de Vida

Por último, se estudia cómo Android clasifica los procesos de manera jerárquica según la importancia que tengan en un momento determinado para el sistema (obtenido de [23] y [24]):

- ***Foreground process***: Proceso en primer plano con el que el usuario interactúa. Éste hospeda una *activity* y el sistema lo eliminaría como última opción.
- ***Visible process***: Proceso que hospeda una *activity* pero que no está en primer plano, aunque sí visible. Esto ocurre por ejemplo cuando un cuadro de dialogo es ejecutado, se interactúa con el mismo pero se sigue viendo la pantalla detrás de éste. Este tipo de proceso será eliminado en estados de memoria muy críticos al igual que el ***Foreground process***.
- ***Background process***: Proceso que hospeda un *activity* pero no está visible en pantalla, es decir que si se elimina no tiene repercusión directa con el usuario.

- ***Empty process***: Proceso que no hospeda ningún componente visualmente activo. Estos serán eliminados con frecuencia y pueden mantenerse vivos si hay memoria suficiente para mejorar el tiempo de activación de otro componente de esta aplicación.

Estas prioridades son mostradas gráficamente en la Figura 7.4.

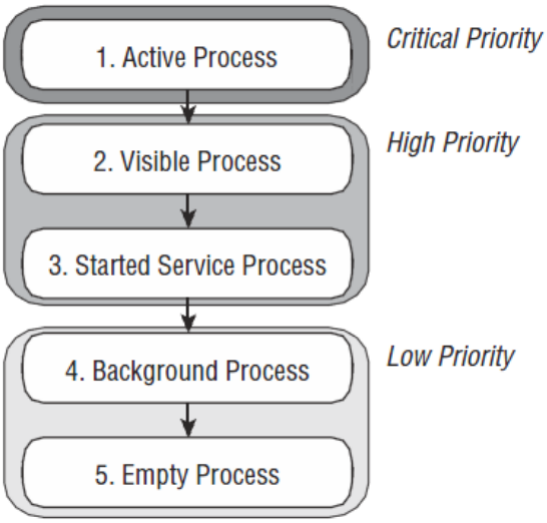

Figura 7.4: Prioridad de los Procesos en Android

Capítulo 8

Herramientas Relacionadas con SNMP para Android

A continuación se estudiarán varias aplicaciones que están relacionadas con la propuesta antes planteada. Solamente se encontraron aplicaciones que trabajan como estaciones de administración con las distintas versiones del protocolo SNMP.

8.1. SNMP MIB Browser

SNMP MIB Browser[1] es una aplicación creada por ZOHO Corporation que puede ser descargada libremente (sin costo alguno) del Play Store de Android. Soporta todas las versiones de SNMP y tiene una interfaz sencilla donde es posible cargar la MIB para posteriormente ser consultada a través del protocolo SNMP. Los formatos de archivos que soporta la aplicación para cargar una MIB son: `.mib`, `.my` y `.txt`.

La última actualización (versión 1.1) fue realizada el 24 de enero de 2013 y tiene un tamaño de 694kB. En SNMPv3, soporta autenticación MD5 y SHA. Para el cifrado se pueden utilizar varios algoritmos como: DES, 3DES, AES-128, AES-192 y AES-256.

SNMP MIB Browser es de fácil instalación, solo se debe descargar del *Play Store* y automáticamente se instalará. Con respecto a la interfaz de usuario es poco intuitiva y se debe tener un conocimiento intermedio para poder usarla. Como es un navegador de MIBs, este tiene la capacidad de agregar nuevas MIBs a la vista y consultarlas. No posee la capacidad de usar herramientas de administración como ping, tracert, escaneo de puertos y otras.

8.2. ezNetScan

ezNetScan[2] es una aplicación gratuita que puede ser descargada del *Play Store* de Android y que fue desarrollada por VRSSPL. Es una herramienta útil para los administradores de red ya que escanea la red inalámbrica mostrando todos los dispositivos conectados (*Discovery* de la red). Permite manejar una lista personalizada de dispositivos, donde el administrador tiene la opción de agregarle un ícono, comentario, nota o etiqueta al dispositivo administrado. Basado en las consultas de SNMP permite

[1] https://play.google.com/store/apps/details?id=com.zoho.snmpbrowser
[2] https://play.google.com/store/apps/details?id=com.vrsspl.eznetscan

obtener la lista del software instalado y la información de hardware de los dispositivos de red.

Actualmente se encuentra en la versión 2.1.3 con un tamaño de 2.1MB. Su última actualización fue el 10 de noviembre de 2012. Esta aplicación es fácil de instalar ya que con solo descargarla iniciará automáticamente su instalación. Posee una interfaz gráfica que no necesita mucho aprendizaje para su uso y es intuitiva. Como se mencionó, trabaja con SNMP para ciertas consultas y solo soporta hasta la versión 2. Posee varias herramientas que ayudan a la administración de los dispositivos y no posee la característica de recepción de *Traps*.

8.3. SNMP Manager

SNMP Manager[3] es un manager que fue elaborado por Y. Matsumoto y que posee la capacidad de enviar y recibir *Traps*. Cuenta con soporte limitado para SNMPv3 permitiendo cifrado únicamente con DES y para la autenticación los algoritmos MD5 y SHA.

SNMP Manager se puede descargar gratuitamente del *Play Store* de Android. Posee un tamaño de 500kB en su última actualización (15 de octubre de 2012). SNMP Manager presenta una instalación fácil. La interacción con el usuario es poco amena y no es intuitiva. Soporta las 3 versiones del protocolo SNMP y puede recibir y enviar *Traps*. SNMP Manager no posee herramientas de administración.

8.4. Comparación de las Características de las Herramientas Relacionadas

En la Tabla 8.1 se muestran las características de las aplicaciones antes descritas que trabajan con SNMP en la plataforma Android. La valoración que presentan cada una son apreciación del autor.

Categoría	SNMP MIB Browser	ezNetScan	SNMP Manager
Instalación	★★★★★	★★★★★	★★★★★
Usabilidad	★★★	★★★★	★★
Interfaz Gráfica	★★	★★★★★	★
Soporte para SNMPv1	★★★★★	★★★★★	★★★★★
Soporte para SNMPv2c	★★★★★	★★★★★	★★★★★
Soporte para SNMPv3	★★★★★		★★★★★
Configuración de la Consulta	★★★★	★★	★★★★
Herramienta ping		★★★★★	
Herramienta tracert		★★★★★	
Importar Nuevas MIBs	★★★★★		
Escáner de Puertos		★★★★★	
Net-Bios		★★★★★	
Envío y Recepción de Traps			★★★★★

Tabla 8.1: Comparación de las Características de las Herramientas Relacionadas con SNMP para Android

[3]https://play.google.com/store/apps/details?id=jp.ymatsumoto.snmpmanager

Capítulo 9

Marco Metodológico

Para lograr los objetivos planteados en el Capítulo 2, es necesario definir un esquema o metodología de trabajo que permita desarrollar la aplicación de manera estructurada y planificada. A continuación, se presenta la especificación de la metodología utilizada y otros detalles importantes que fueron tomados en cuenta para el desarrollo e implementación del agente SNMP.

9.1. Adaptación de la Metodología de Desarrollo

Hoy en día existen una gran variedad de metodologías de desarrollo que han ido cambiando de un enfoque secuencial a iterativo. Esto se debe a que en el desenvolvimiento de las fases o actividades de desarrollo, en un enfoque secuencial estas se realizan una sola vez, con poca adaptabilidad y flexibilidad, en cambio, en un enfoque iterativo estas fases se repiten por cada módulo que se vaya a desarrollar. Este cambio de enfoque se ha dado porque siempre en la ejecución de alguna de las fases surgen nuevos requerimientos por parte del cliente.

Por esta razón, se ha decidido trabajar con la metodología XP (eXtreme Programming)[1] la cual es un método iterativo basado en el modelo de desarrollo ágil. XP se basa en los cambios que ocurren en los requerimientos por parte del cliente que son inciertos e inevitables en un proyecto de desarrollo, haciendo énfasis en la adaptabilidad y menos en la previsibilidad.

La metodología define cuatro actividades básicas o fases que se realizan en el desarrollo de software: Planificación, Diseño, Codificación y Pruebas[2].

9.1.1. Planificación

En esta primera fase son generados los requerimientos, se planifica un tiempo estimado para cada uno de ellos, luego serán desarrollados mediante iteraciones que cubren pequeñas funcionalidades o características requeridas. La combinación de estas iteraciones provee un producto final funcional.

Parte de esta planificación consiste en organizar los requerimientos en función de su prioridad, de esta manera se tiene una visión general de las actividades que se pueden hacer de forma lineal o en paralelo.

[1] http://www.extremeprogramming.org
[2] http://www.extremeprogramming.org/rules.html

9.1.2. Diseño

El diseño crea una estructura que organiza la lógica del sistema, un buen diseño permite que el sistema crezca con cambios en un solo lugar. Los diseños deben de ser sencillos, si alguna parte del sistema es de desarrollo complejo, se divide en varias partes. Si hay fallos en el diseño o malos diseños, estos deben de ser corregidos cuanto antes.

Es importante codificar porque sin código no hay programas pero también hacer buen diseño evitará una gran cantidad de dependencias dentro de un sistema, lo que significa que el cambio en una parte del sistema no afectará a otras partes del mismo.

9.1.3. Codificación

El único producto verdaderamente importante del proceso de desarrollo del sistema es el código. Sin código no existe un producto de trabajo. La codificación también se puede utilizar para determinar cuál es la solución más adecuada y puede ayudar a comunicar pensamientos acerca de problemas de programación. Es la única actividad de la que no se puede prescindir.

9.1.4. Pruebas

A través de pruebas unitarias se busca corregir cada nuevo desarrollo, de esta manera eliminar errores puntuales antes de integrar con otros módulos. Una vez solucionados los problemas de programación se hacen pruebas con el usuario basadas en sus requerimientos, con el fin de validar el código.

9.2. Tecnologías a Utilizar

Será utilizado (1) el lenguaje de programación Java para realizar la aplicación para Android, (2) el IDE Eclipse para crear el entorno de desarrollo, (3) el ADT (Android Developer Tools) que facilita la creación de aplicaciones para Android, (4) *SNMP Package* para desarrollar los módulos de envío y recepción de mensajes en el agente, (5) Wireshark para el análisis de los paquetes intercambiados entre las entidades SNMP, (6) SNMP JManager y (7) AdminUCV NGN para la generación de mensajes SNMP y recepción de tramas *Traps*.

A continuación se describen brevemente cada una de las tecnologías mencionadas:

- Java: Lenguaje de programación creado por Sun Microsystems, Inc, que actualmente pertenece a Oracle Corporation. Es un lenguaje orientado a objeto que permite el desarrollo de aplicaciones sobre cualquier plataforma o sistema operativo gracias a que se ejecuta en una máquina virtual.
- IDE Eclipse: Es un reconocido entorno de desarrollo integrado donde se pueden crear proyectos grandes, importar bibliotecas, y añadir *plugins* para así facilitar el desarrollo de aplicaciones. Aún cuando Eclipse puede ser usado con diferentes lenguajes de programación (C, C++, Java, etc), la mayoría de los desarrolladores lo utilizan con Java.
- ADT: Es un *plugin* que extiende las capacidades de Eclipse para crear nuevos proyectos para Android, facilitando así la creación de UI, la adición de paquetes del API de desarrollo de Android, la depuración de la aplicación, la realización de pruebas y la publicación de la aplicación.

- *SNMP Package*: Es una biblioteca *Open Source* desarrollada en Java que provee clases abstractas e interfaces para el envío de paquetes SNMP. Esta biblioteca tambíen cuenta con el conjunto de tipos de datos y manejo de errores de SNMPv1 y SNMPv2c.
- Wireshark: Es un analizador de paquetes que provee un interfaz de usuario. Con este programa se pueden capturar una gran variedad de protocolos y colocar las interfaces de red en modo promiscuo. Wireshark provee un filtrado de paquetes y muestra los paquetes de varias formas (representación hexadecimal o representación humana).
- AdminUCV NGN: Es una aplicación *Open Source* de administración de red [25] con licencia GNU GPL, desarrollada en Java/C++, que soporta IPv4 e IPV6. Esta aplicación soporta las dos primeras versiones de SNMP (v1 y v2c). AdminUCV NGN posee herramientas gráficas para monitorear la red, un receptor de *Traps*, un compilador de MIBs y varias herramientas administrativas de red.
- SNMP JManager: Es una entidad administradora SNMP *open-source* [26] capaz de soportar las distintas versiones de SNMP (v1/v2c/v3), importar nuevas MIBs, soportar IPv4 e IPv6 y otras funcionalidades basadas en el protocolo SNMP.

9.3. Prototipo General de la Interfaz

Con el objetivo de proporcionar un modo sencillo para la configuración del agente SNMP, se define un prototipo de interfaz dividida en pestañas con el fin de realizar distintas actividades con mayor flexibilidad. El diseño general de la interfaz se puede ver en la Figura 9.1, donde se muestra: (1) el nombre de la aplicación, (2) el conjunto de pestañas y el *layout* donde estarán los distintos componentes (*TextView*, *EditText*, *RadioButtons*, *Buttons*, entre otros) según la funcionalidad de la pestaña o módulo de la aplicación.

Figura 9.1: Prototipo General de la Interfaz

Capítulo 10

Marco Aplicativo

En este capítulo se explica el diseño de la solución aplicando la metodología XP, que fue presentada en el Capítulo 9. Cada iteración describe la creación de un módulo en la aplicación y las mismas están divididas en fases (diseño, codificación y pruebas).

10.1. Análisis General

Antes de iniciar el desarrollo de la aplicación, se llevó a cabo una fase de análisis general donde se determinó de manera global los principales requerimientos de la aplicación, los cuales deben cubrir los objetivos definidos en el Capítulo 2 para dar solución al problema.

A continuación, se muestra cómo se estructuró la lista de requerimientos de la aplicación:

- Diseñar una interfaz de usuario basada en el prototipo planteado en el Marco Metodológico.
- Implementar un proceso en *background* utilizando el API que provee Android para la ejecución del agente.
- Dar soporte de SNMP para las versiones 1 y 2c.
- Crear un módulo para el envío de *Traps* versión 1 y 2c.
- Brindar un módulo de configuración del agente para el *System Group*.

10.2. Desarrollo de la Aplicación

A continuación se especifican cada una de las iteraciones que fueron necesarias y las fases implementadas en ellas según el modelo de programación XP.

10.2.1. Iteración 1: Desarrollo de la Interfaz Gráfica e Implementación del Proceso en Background

La interfaz gráfica es la aplicación que permite realizar la configuración, así como ejecutar y detener el agente que se encuentra en *background*. La misma está divida en cuatro módulos: *Config Agent*, *Traps v1*, *Traps v2c* y *System Group*.

- **Fase de Diseño**: Como se menciona en la Sección 9.3, los módulos se pueden acceder a través de las distintas pestañas diseñadas en el prototipo. Cada una de estas pestañas cumple

una función distinta dentro de la aplicación. A continuación se detallarán los módulos que se desarrollaron:

- Módulo *Config Agent*: Este módulo se encarga de obtener los parámetros para inicializar el agente. En la Figura 10.1 se observa los distintos campos que sirven para realizar una configuración previa antes de colocar el agente en ejecución. Para configurar el nombre de la comunidad SNMP se tienen dos campos: `Read-Only` que permite solo consultas SNMP y `Read-Write` que permite realizar tanto consultas como asignaciones de valores a los OIDs. El siguiente campo que se observa es *Port*, el cual define por qué número de puerto el agente estará escuchando las consultas SNMP. Vale destacar que el número de puerto debe ser un entero mayor estricto que 1.024 y menor o igual que 65.535. Seguidamente hay dos *RadioButton* que permiten seleccionar la versión en que estará funcionando el agente, ya sea v1 o v2c. Por último se tiene un *Switch* para iniciar o detener el servicio SNMP. Una vez que se inicie el proceso en *background* los campos de los parámetros son bloqueados para que el usuario no pueda realizar ningún cambio durante la ejecución.

Figura 10.1: Módulo Config Agent del Agente SNMP AgentDroid

- Módulo *Traps* v1: A través de este módulo se puede realizar el envío de tramas *Traps* v1. Cada campo presentado en la Figura 10.2 representa un parámetro necesario para el envío de la trama. Como primer campo se tiene la dirección IPv4 destino, ésta dirección debe ser la IPv4 que tiene la estación que cumpla la función de *manager* dentro de la infraestructura de red. Seguidamente se solicita el puerto, cuyo número debe ser un entero mayor estricto que 0 y menor o igual que 65.535. Cabe destacar que ese número debe coincidir con el puerto de escucha de la estación administradora, comúnmente se utiliza el puerto 162. Posterior al puerto se encuentra un campo para introducir el nombre de la comunidad SNMP, el cual es un *String*. Luego se solicita el tipo de *Trap* a enviar, donde el usuario tendrá las siguientes opciones: `coldStart`, `warmStart`, `linkDown`, `linkUp`, `authenticationFailure`, `enterpriseSpecific`. En caso de que se seleccione la opción `enterpriseSpecific` se añadirá otro campo llamado *Specific Trap*. Por último está el botón *Send Trap* para enviar el *Trap* con los datos suministrados.

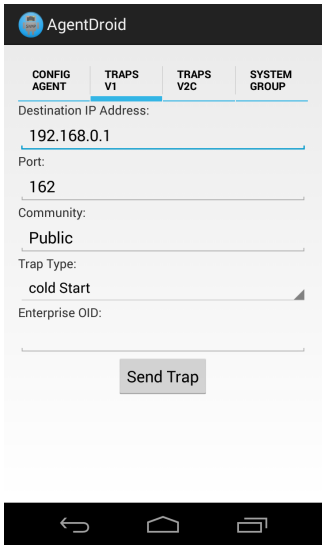

Figura 10.2: Módulo Traps v1 del Agente SNMP AgentDroid

- Módulo *Traps* v2c: Este módulo se desarrolló para el envío de *Traps* v2c. Como su homólogo *Traps* v1, tiene los primeros tres parámetros. El usuario debe elegir el tipo de *Trap* (`coldStart`, `warmStart`, `linkDown`, `linkUp`, `authenticationFailure` y `other`) que desea enviar y en caso de seleccionar `other` debe llenar los siguientes campos: *EnterpriseOID*, *Trap OID*, *Description* y *Data Type*. Los tipos de datos que puede seleccionar el usuario para el campo *Data Type* son los siguientes: `SNMPInteger`, `SNMPOctetString`, `SNMPObjectIdentifier`, `SNMPIPAddress` y `SNMPTimeTicks`. Por último se encuentra el botón *Send Trap* para enviar la trama.

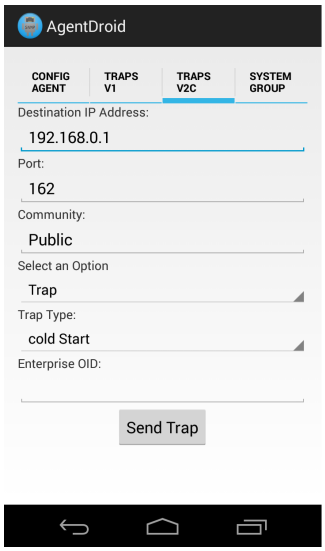

Figura 10.3: Módulo Traps v2c del Agente SNMP AgentDroid

- Módulo *System Group*: Este módulo se diseñó para mostrar información relacionada con el *System Group*, grupo que pertenece a la MIB-II. El *System Group* tiene OIDs que son del tipo `Read-Write`, por lo tanto estos campos son *EditText* para que sean modificados

(ver Figura 10.4). En caso contrario, para los OIDs de tipo `Read-Only` estos son *TextView*, ya que no se puede realizar ningún cambio sobre ellos. Por último, se encuentra un botón llamado *Save* para guardar los cambios realizados en los OIDs de tipo `Read-Write` y así puedan ser consultados posteriormente por el administrador.

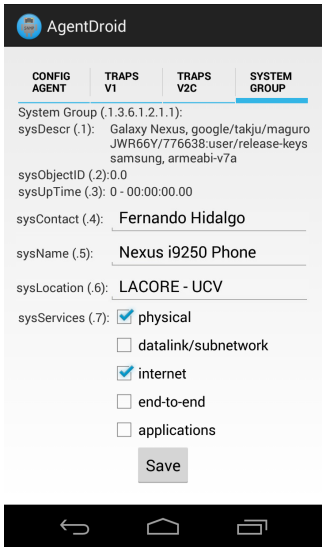

Figura 10.4: Módulo System Group del Agente SNMP AgentDroid

El proceso en *background* se puede iniciar desde el *Switch* que posee la interfaz gráfica en el módulo *Config Agent* (ver Figura 10.1). Este al ser iniciado, según el puerto que es dado por parámetro, se mantiene a la espera de una solicitud SNMP. Al recibir una solicitud SNMP, el agente realiza su procesamiento y crea una respuesta, ya sea con la información solicitada o envía un código de error con la lista de OIDs que recibió. Para que dicho proceso sea iniciado y detenido desde la interfaz gráfica se realiza una vinculación para asociar la interfaz gráfica y el servicio que se ejecuta en *background*.

- **Fase de Codificación**: Para realizar la interfaz gráfica se creó una clase llamada *AgentDroid* que heredó de la clase *Activity* que provee la plataforma Android. A través de un archivo XML se especifica cada componente (View) que se encuentra en la interfaz. Al iniciar la aplicación se ejecuta el método *onCreate(Bundle savedInstanceState)*, donde se asigna el archivo XML al *contentView* para mostrar por pantalla la interfaz de la aplicación, en este método también se realiza la inicialización de las variables y creación de métodos que escuchan los eventos que podría realizar el usuario. También al iniciar la aplicación se consulta por cual medio de transmisión está conectado el dispositivo móvil (red móvil o WiFi) y en el caso de que este conectado a una WLAN se muestra el SSID . En la Figura 10.5 se observa parte del XML utilizado para añadir los componentes a la interfaz gráfica.

```
<LinearLayout
    android:id="@+id/layoutForScroll1"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >

    <TextView
        android:id="@+id/TextComRO"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/communityRO" />

    <EditText
        android:id="@+id/EditComRO"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/communityDefault"
        android:ems="10"
        android:inputType="text" />

    <TextView
        android:id="@+id/TextComRW"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/communityRW" />
```

Figura 10.5: XML para Crear la Interfaz Gráfica de AgentDroid

Para crear el proceso en *background* se implementa una clase llamada *AgentSNMP* que hereda de la clase *Service*. En su método *onCreate()* se inicia un thread el cual hace la apertura del socket según el número de puerto que es obtenido como parámetro de la interfaz gráfica. Dicho thread tiene dos métodos: *startReceiving()* y *stopReceiving()*, que sirven para iniciar y detener su ejecución respectivamente y a su vez la apertura y cierre del socket. Para realizar el procesamiento de un mensaje SNMP, el thread en su método *run()* realiza el análisis del paquete para verificar que tipo de mensaje SNMP fue recibido y así realizar su procesamiento de manera adecuada. En las siguientes iteraciones se detallará el procesamiento del paquete según los tipos de mensaje SNMP. En la Figura 10.6 se observa el método *onCreate()* del servicio que se ejecuta en *background*, donde se obtienen los valores de la interfaz gráfica y se inicia el thread con los parámetros dados.

```
@Override
public void onCreate(){
    try {
        Context context  = getBaseContext();

        DBOIDHelper database = new DBOIDHelper(getBaseContext());
            ObjIdent oid = database.getOID("portSNMP");
            int port = Integer.parseInt(oid.getValue());
            oid = database.getOID("snmpVersionOne");
            int version = (oid.getValue().equalsIgnoreCase("true"))?0:1;
            oid = database.getOID("CommunityReadOnly");
            String communityRO = oid.getValue();
            oid = database.getOID("CommunityReadWrite");
            String communityRW = oid.getValue();
        database.cleanup();

        this.MIB = new Mibs(context, communityRO, communityRW);
        this.agent = new SNMPv1AgentInterface(version, port);
        this.agent.addRequestListener(MIB);
        this.agent.startReceiving();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figura 10.6: Método onCreate() del Servicio AgentSNMP

- **Fase de Pruebas**: Para verificar el correcto funcionamiento de esta primera iteración se realizaron pruebas funcionales y de usabilidad sobre la interfaz gráfica. Para cada módulo de

la interfaz, se corroboró el correcto funcionamiento y así se determinó que se ejecuta como se espera. Para comprobar la ejecución del servicio en *background* se utilizó una herramienta del SO que permite verificar las aplicaciones que se están ejecutando (ver Figura 10.7), de igual forma se observó que la vinculación con la interfaz funciona correctamente.

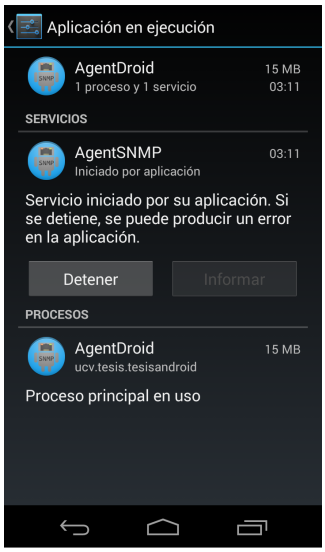

Figura 10.7: Demostración de la Ejecución del Servicio AgentSNMP en Background

10.2.2. Iteración 2: Desarrollo de la Recepción del Mensaje GetRequest y Envío de su Respuesta

En esta iteración se desarrolló la recepción de un mensaje GetRequest y el envío de su respectiva respuesta, ya sea con la información consultada o una respuesta con un código de error.

- **Fase de Diseño**: Una vez que se encuentra el proceso en *background*, se espera la llegada de mensajes SNMP solicitando información al agente. Cuando se recibe un mensaje GetRequest el campo *PDUType* debe contener como valor `0` según la especificación del protocolo. Una vez confirmado que la solicitud es un GetRequest, se comienza el procesamiento de la lista de OIDs consultados. A medida que se van extrayendo los OIDs de la lista, se van consultando uno por uno para ir creando la lista de respuesta con los valores obtenidos. Si ocurre algún error durante el procesamiento de algún OID se envía una respuesta con un código de error y el índice de la variable donde se generó el error.
- **Fase de Codificación**: Para el procesamiento de un mensaje GetRequest se utiliza la biblioteca *SNMP Package*. Con la biblioteca se realiza la extracción de los campos agrupados en estructuras TLV. Primero se comprueba si la versión SNMP del mensaje es la misma que está ejecutando el agente, seguido a esto se extrae la PDU y se invoca el método *processRequest(SNMPPDU pdu, String communityName)* para que este retorne una secuencia de TLVs donde estarán los OIDs con sus valores. Una vez consultados todos los OIDs se comprueba que la lista devuelta por el método sea la misma que se consultó y que no hubo ningún error. Por último, la lista se anexa al paquete que será enviado como respuesta a la consulta solicitada. En caso de que haya existido algún error será enviado en la respuesta el código de error que ocurrió y en que variable se originó, y la lista de OIDs sin valores. El

método *processRequest(SNMPPDU pdu, String communityName)* pertenece a la clase *Mibs*, la cual implementa la interfaz *SNMPRequestListener*. En la Figura 10.8 se observa parte del código que se emplea en la clase *Mibs* para la validación del *PDUType* e implementación de los OIDs.

```
if (pduType == SNMPBERCodec.SNMPGETREQUEST)
{
    if(!(communityName.equalsIgnoreCase(this.communityRW) || communityName.equalsIgnoreCase(this.communityRO))){
        errorIndex = 0;
        errorStatus = SNMPRequestException.FAILED;
        throw new SNMPSetException("Name community bad!", errorIndex, errorStatus);
    }

    if(snmpOID.toString().startsWith("1.3.6.1.2.1.1"))
    {
        /* sysDescr  DisplayString | Read-Only | Mandatory */
        if (snmpOID.toString().equals("1.3.6.1.2.1.1.1.0") || snmpOID.toString().equals("1.3.6.1.2.1.1.1"))
        {
            try
            {
                DBOIDHelper database = new DBOIDHelper(this.contextActivity);
                    ObjIdent oid = database.getOID("1.3.6.1.2.1.1.1.0");
                database.cleanup();
                SNMPVariablePair newPair = new SNMPVariablePair(new SNMPObjectIdentifier(snmpOID.toString()),
                                                                new SNMPOctetString(oid.getValue()));
                responseList.addSNMPObject(newPair);
            }
            catch (SNMPBadValueException e)
            {
                errorIndex = i+1;
                errorStatus = SNMPRequestException.FAILED;
            }
            continue;
        } // sysDescr
```

Figura 10.8: Validación del PDUType para el Mensaje GetRequest

- **Fase de Pruebas**: Para realizar las pruebas, se utiliza la herramienta Wireshark para verificar el correcto funcionamiento de este módulo. En la Figura 10.9 se observa la respuesta emitida por el agente que se ejecuta en el dispositivo móvil, donde se ve claramente cada uno de los campos del PDU de un mensaje GetResponse. Un estudio de desempeño del agente al enviar respuesta a un GetRequest se encuentra en el Capítulo 11.

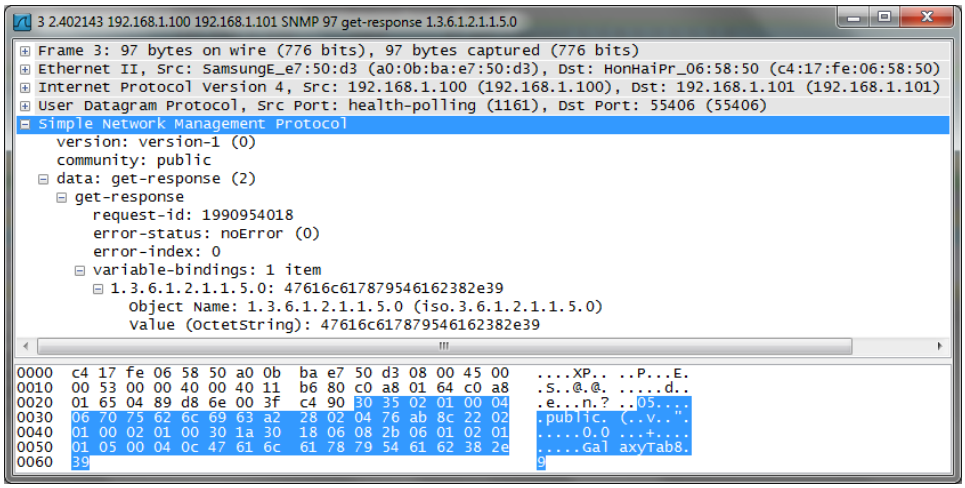

Figura 10.9: GetResponse Capturado en la Prueba del Mensaje GetRequest

10.2.3. Iteración 3: Desarrollo de la Recepción del Mensaje GetNextRequest y Envío de su Respuesta

A continuación se describen las fases que se llevaron a cabo para la recepción del mensaje GetNextRequest y el envío de su correspondiente respuesta.

- **Fase de Diseño**: Al recibir un mensaje GetNextRequest, el proceso en *background* debe verificar que posee en el campo *PDUType* el valor de `1`. Al realizar dicha verificación se extrae la lista de OIDs y se ejecuta la búsqueda en el árbol de las MIBs implementadas para encontrar el OID siguiente al consultado. Al encontrar el siguiente OID, se obtiene su valor y se crea una respuesta con la lista nueva de OIDs y sus respectivos valores. En caso de no encontrar un siguiente OID o de existir un problema en el procesamiento del mensaje, se emitirá una respuesta con la lista de OIDs originales y un código de error en el campo `Error-Status` como lo especifica el protocolo.
- **Fase de Codificación**: Tal como se realiza con el mensaje GetRequest, el procesamiento del mensaje GetNextRequest se lleva a cabo con la biblioteca *SNMP Package*. Para la verificación del tipo de mensaje que se está procesando se utiliza la clase *SNMPBERCodec*, que contiene un conjunto de atributos para comparar sus valores con el campo `PDUType`. Una vez que el mensaje es identificado como un GetNextRequest y su PDU es extraído, el thread invoca el método *processGetNextRequest(SNMPPDU pdu, String communityName)* de la clase Mibs. El método *processGetNextRequest* se encarga de realizar la búsqueda del siguiente OID al consultado en el árbol de MIBs y de crear la respuesta con una nueva lista de OIDs y sus valores. Para generar el árbol de MIBs, se implementa la clase *SNMPTreeGetNextRequest* y a través del método *SNMPTreeLoad(Context contAct)* se crea el árbol con los OIDs implementados en la clase *Mibs*. La búsqueda de un OID se realiza con el algoritmo de búsqueda binaria y para encontrar el siguiente OID se invoca el método *GetNext()*, de esta forma se obtiene el siguiente OID al consultado. Una vez que se encuentran todos los OIDs siguientes, se almacenan en una nueva lista para ser consultados invocando el método *processRequest(SNMPPDU pdu, String communityName)* de la clase *Mibs*. Luego de consultar los nuevos OIDs se agrega la lista con los valores al PDU de la respuesta y se envía.

```
public final static SNMPTreeGetNextRequest<String> SNMPTreeLoad(Context contAct )
{
    SNMPTreeGetNextRequest<String> root = new SNMPTreeGetNextRequest<String>("1.3.6.1.2.1");
    {
        /**
         * SYSTEM
         */
        SNMPTreeGetNextRequest<String> system = root.addNode("1.3.6.1.2.1.1", true);
        system.addLeaf("1.3.6.1.2.1.1.1"); //sysDescr
        system.addLeaf("1.3.6.1.2.1.1.2"); //sysObjectID
        system.addLeaf("1.3.6.1.2.1.1.3"); //sysUpTime
        system.addLeaf("1.3.6.1.2.1.1.4"); //sysContact
        system.addLeaf("1.3.6.1.2.1.1.5"); //sysName
        system.addLeaf("1.3.6.1.2.1.1.6"); //sysLocation
        system.addLeaf("1.3.6.1.2.1.1.7"); //sysServices
```

Figura 10.10: Extracto del Método SNMPTreeLoad para Crear el Árbol de MIBs Implementadas

- **Fase de Pruebas**: En esta fase se realizaron pruebas, donde se utiliza la herramienta Wireshark para verificar cada uno de los campos del PDU de la respuesta creada. En la Figura 10.11 se observa el mensaje GetResponse generado por el agente al recibir un mensaje del tipo GetNextRequest. Un estudio de desempeño del agente al enviar respuesta a un GetRequestNext se encuentra en el Capítulo 11.

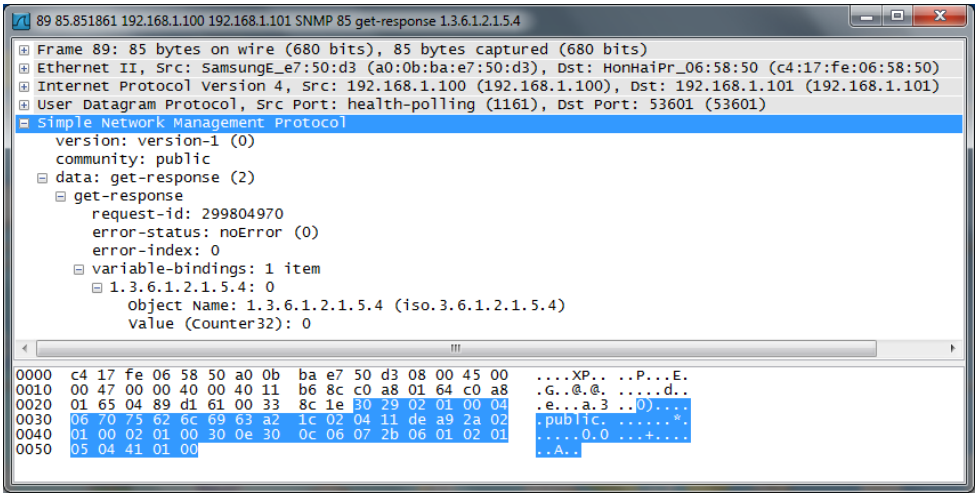

Figura 10.11: GetResponse Capturado en la Prueba del Mensaje GetNextRequest

10.2.4. Iteración 4: Desarrollo de la Recepción del Mensaje SetRequest y Envío de su Respuesta

En esta iteración se lleva a cabo la implementación de la recepción de un SetRequest por un agente y su procesamiento. Este mensaje posee una funcionalidad distinta a los antes mencionados debido a que sirve para asignar valores a los OIDs en vez de consultarlos. También en esta iteración se implementa el envío de un mensaje GetResponse como respuesta a al mensaje SetRequest.

- **Fase de Diseño**: Cuando el agente recibe un mensaje SNMP lo primero que realiza es la verificación del tipo de mensaje. Para saber que tipo de mensaje es, se utiliza el campo *PDUType*, el cual en este caso (SetRequest) debe tener como valor entero el número 3. Como se mencionó antes, la finalidad del mensaje SetRequest es distinta a los otros mensajes, ya que este asigna valores a las variables de las MIBs que son del tipo `Read-Write`. Luego de verificar el tipo de mensaje, se debe obtener la lista de OIDs con los valores que se desean asignar a cada uno. El valor que se asigna se debe mantener hasta que el administrador decida cambiarlo nuevamente y lo debe hacer con el nombre de comunidad `Read-Write`. Como respuesta a este mensaje se debe retornar la lista de OIDs con los valores que se asignaron. En caso de haber algún fallo en la asignación de estos valores se retornará un GetResponse con el respectivo código de error en el campo `Error-Status`.
- **Fase de Codificación**: La verificación del tipo de mensaje la realiza el thread que está a la escucha de los mensajes SNMP. Cuando se extrae la lista de OIDs con los valores a asignar, se invoca el método *processRequest(SNMPPDU pdu, String communityName)* para procesar el mensaje. Para validar que se tenga permiso de escritura se valida el nombre de comunidad, este debe ser igual al nombre de comunidad `Read-Write` con que se inició el servicio. Para efectuar el almacenamiento de los valores asignados se implementó una BD (Base de Datos) a través de la clase *DBOpenHelper* que hereda de *SQLiteOpenHelper*, esta última proveída por la plataforma Android para la creación y manejo de BD en el sistema. Para el control de los OIDs se tiene otra clase llamada *ObjIdent* que posee los atributos para obtener tanto el índice como el valor de un OID en específico. Por último existe la clase *DBOIDHelper* que es la encargada de realizar la apertura, operaciones y cierre de la BD. En la Figura 10.12 se observa cómo se realiza la apertura de la BD, la actualización del OID SysContact y por último el cierre de la BD. Luego de

que se asigna el valor correctamente al OID se almacena en la lista de respuesta para certificar que no hubo ningún problema, este proceso se hace con los demás OIDs de la lista de variables que existe en el mensaje de llegada. Una vez culminado el proceso de asignación se envía la lista de respuesta con los valores asignados en caso de que no haya existido algún error, de lo contrario de enviará el código de error que ocurrió mientras se realizaba el proceso.

```
/* sysContact DisplayString | Read-Write | Mandatory */
if (snmpOID.toString().equals("1.3.6.1.2.1.1.4.0"))
{
    if (snmpValue instanceof SNMPOctetString)
    {
        storedSNMPValue = (SNMPOctetString)snmpValue;

        try
        {
            DBOIDHelper database = new DBOIDHelper(this.contextActivity);
                database.update(snmpOID.toString(), storedSNMPValue.toString() );
            database.cleanup();
            SNMPVariablePair newPair = new SNMPVariablePair(snmpOID, storedSNMPValue);
            responseList.addSNMPObject(newPair);
        }
        catch (SNMPBadValueException e)
        {
            errorIndex = i+1;
            errorStatus = SNMPRequestException.FAILED;
        }
    }
    else
    {
        errorIndex = i+1;
        errorStatus = SNMPRequestException.BAD_VALUE;
        throw new SNMPSetException("Supplied value must be SNMPOctetString", errorIndex, errorStatus);
    }

    continue;
} // sysContact
```

Figura 10.12: Extracto de Código para Actualizar el OID SysContact del System Group

- **Fase de Pruebas**: Para verificar el funcionamiento de este módulo se realizaron pruebas validando cada uno de los tópicos que se debe cumplir. También se utilizó la herramienta Wireshark para realizar la captura de paquetes y ver el valor de cada uno de los campos (ver Figura 10.13). En el Capítulo 11 se realiza una comparación del desempeño de la respuesta de este mensaje con respecto a los mensajes GetRequest y GetNextRequest.

10.2.5. Iteración 5: Desarrollo de la Recepción del Mensaje GetBulkRequest y Envío de su Respuesta

En la siguiente iteración se comienza a desarrollar en el agente el soporte de SNMPv2c. Con el mensaje GetBulkRequest se pude obtener una cantidad considerable de información a través de una sola solicitud.

- **Fase de Diseño**: Como ya antes se ha mencionado, se debe validar el tipo de mensaje que ha recibido el agente. Hay que resaltar que el formato del mensaje GetBulkRequest es distinto a los demás (ver Figura 3.7), interpretando los campos `Error-Status` y `Error-Index` como `non-repeaters` y `max-repetitions` respetivamente. Una vez que se obtienen los campos del PDU se debe hacer la consulta según como indique los valores de los campos `non-repeaters` y `max-repetitions`. Cuando se tiene la nueva lista de OIDs consultados se debe agregar a la PDU del mensaje GetResponse. En caso de haber algún error será indicado en el campo `Error-Status` y el campo `Error-Index` indicará en que variable ocurrió dicho error.

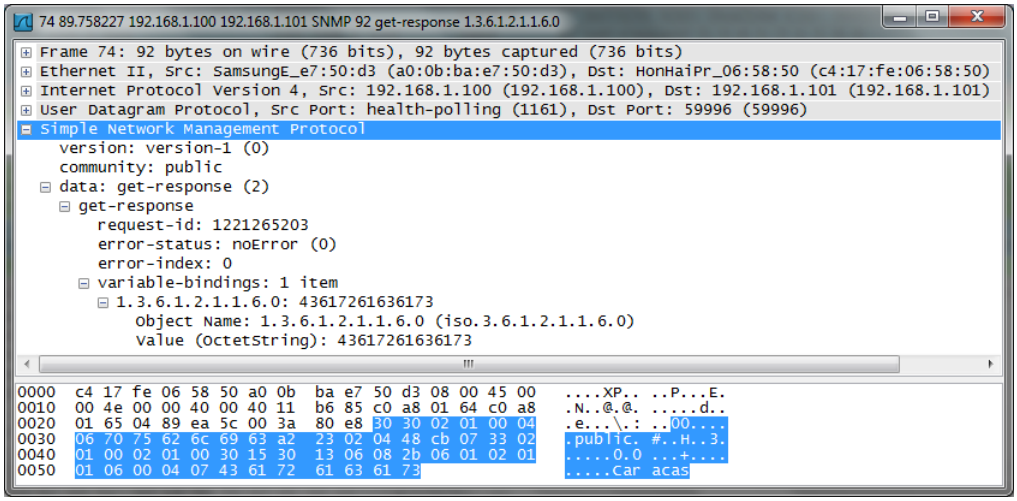

Figura 10.13: GetResponse Capturado en la Prueba del Mensaje SetRequest

- **Fase de Codificación**: Una vez validado que el mensaje es del tipo GetBulkRequest a través del proceso en *background*, se invoca el método *processGetBulkRequest(SNMPv2BulkRequestPDU pdu, String communityName)* de la clase *Mibs*, en el cual lo primero que se hace es obtener los valores de los campos `non-repeaters` y `max-repetitions`, y la lista de OIDs. Con estos valores se procede a obtener los nuevos OIDs que serán agregados al final de la lista para luego ser consultados. El procedimiento que se realiza es obtener los siguientes R OIDs a partir de los N+1 OIDs de la lista inicial, donde R=L-N siendo L el tamaño de la lista y N el número de OIDs que no se le realizarán consultas sucesivas (`non-repeaters`). Al obtener la nueva lista de OIDs, se invoca el método *processRequest* para realizar las consultas y así almacenar los valores obtenidos en la lista que se enviará como respuesta en el GetResponse. En la Figura 10.14 se observa un extracto de la clase *Mibs* realizando la validación del tipo de mensaje e invocando el método *processGetBulkRequest(SNMPv2BulkRequestPDU pdu, String communityName)*.

```
}else if (requestPDUType == SNMPBERCodec.SNMPv2BULKREQUEST)
{
    for (int i = 0; i < listenerVector.size(); i++){
        Mibs listener = (Mibs)listenerVector.elementAt(i);
        SNMPSequence handledVarList = listener.processGetBulkRequest(receivedBulkPDU, communityName);

        for (int j = 0; j < handledVarList.size(); j++){
            SNMPSequence handledPair = (SNMPSequence)handledVarList.getSNMPObjectAt(j);
            SNMPObjectIdentifier snmpOID = (SNMPObjectIdentifier)handledPair.getSNMPObjectAt(0);
            SNMPObject snmpObject = (SNMPObject)handledPair.getSNMPObjectAt(1);
            responseVarList.addSNMPObject(handledPair);
        }
    }

}else{
```

Figura 10.14: Invocación del Método processGetBulkRequest por el Agente SNMP

- **Fase de Pruebas**: Para esta iteración se realizaron las pruebas validando el funcionamiento correcto tanto del procesamiento como el envío de la respuesta. En la Figura 10.15 se observan los campos del mensaje GetResponse que fue capturado dando respuesta a un mensaje GetBulkRequest.

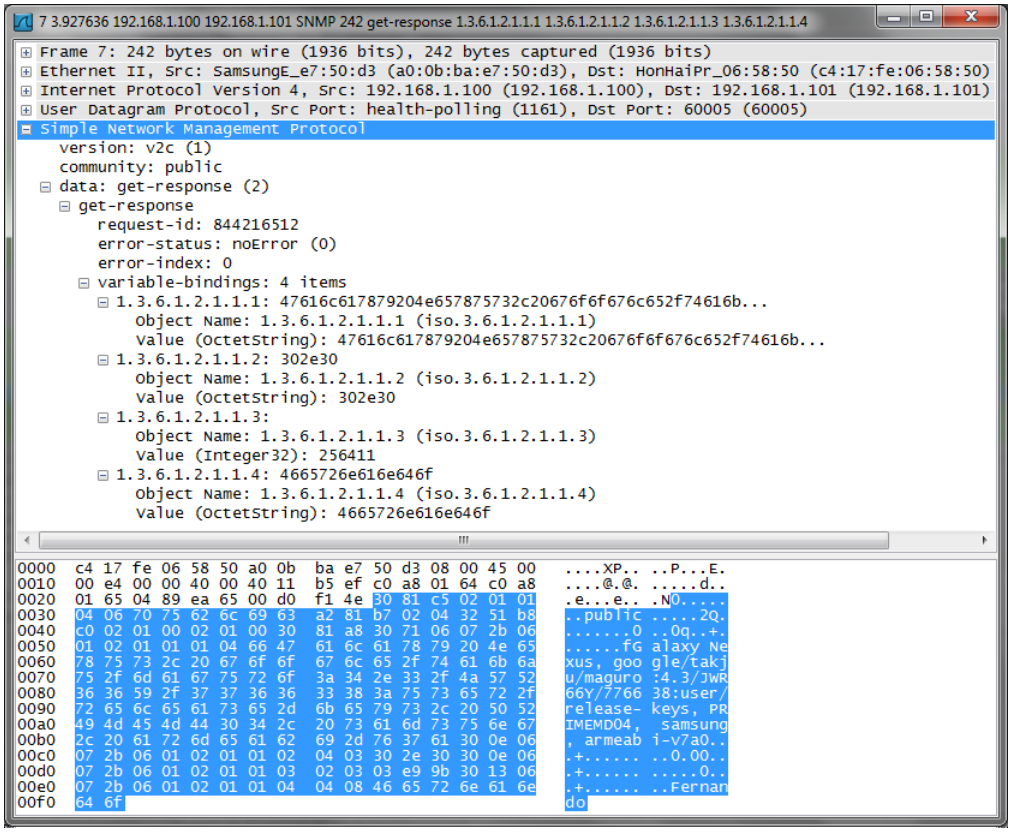

Figura 10.15: GetResponse Capturado en la Prueba del Mensaje GetBulkRequest

10.2.6. Iteración 6: Desarrollo del Envío de *Traps* (v1 y v2c)

Como se estudió en la Sección 3.1.3, el protocolo SNMP provee la opción al agente de enviar notificaciones cuando ocurra algún evento. Con la implementación de esta iteración se concluye el soporte del agente para SNMPv2c.

- **Fase de Diseño**: El envío de notificaciones por parte del agente a las estaciones de administración es un punto importante dentro del protocolo. Para el envío de *Traps* (v1 o v2c) es necesario tener la opción de hacerlo a través de la interfaz gráfica. En el caso de *Traps* v1, se debe dar soporte a las distintas notificaciones (`coldStart`, `warmStart`, `linkDown`, `linkUp`, `authenticationFailure`, `enterpriseSpecific`) que se pueden enviar según lo establecido por el protocolo. Para las *Traps* v2c, se deben considerar los distintos tipos de *Trap* de su versión antecesora agregando la opción de enviar otro tipo de *Trap* y que las primeras dos *variables bindings* deben ser `sysUpTime.0` y `snmpTrapOID.0`.
- **Fase de Codificación**: Como se observó en la subsección 10.2.1, existe un módulo para cada versión de *Traps*. En ambos módulos se solicitan los parámetros necesarios para enviar el *Trap*. Ya que estos mensajes se generan de manera asíncrona, son implementados por las clases *SenderTrapv1Background* y *SenderTrapv2Background* que heredan de la clase *AsyncTask<String[], Void, String>*. Cuando el usuario presiona el botón de enviar, se obtienen los parámetros de la interfaz y se crea la tarea asíncrona para que cree el *Trap* y lo envíe. A través del método *execute(String[] arg)*, se ejecuta la tarea y se envían los parámetros antes obtenidos. Al invocar *execute(String[] arg)* se ejecuta el método *doInBackground(String[]... arg0)*

que es donde realmente está la funcionalidad de la tarea. Para crear el *Trap* v1 y el *Trap* v2c se utilizan las clases *SNMPv1TrapPDU* y *SNMPv2TrapPDU* respectivamente. Para enviar cualquiera de las dos versiones de *Traps* se utiliza la clase *SNMPTrapsenderInterface* proveída por la biblioteca *SNMP Package*. Una vez que la tarea asíncrona finaliza, se ejecuta el método *onPostExecute(Boolean send)* para saber si la tarea pudo ejecutarse correctamente o si ocurrió algún error durante su ejecución. En caso de que la tarea haya o no finalizado correctamente al usuario se le mostrará en pantalla un mensaje con el resultado, ya sea que haya terminado correctamente o con algún error (ver Figura 10.16).

```
@Override
protected void onPostExecute(Boolean send){

    Builder builder = new Builder(AgentDroid.this);

    if(send){
        builder.setTitle("Sender Traps v1");
        builder.setMessage("SNMPv1Trap sent successfully");
        builder.setNeutralButton("Ok", null);
    }else{
        builder.setTitle("Sender Traps v1");
        builder.setMessage("SNMPv1Trap not sent ");
        builder.setNeutralButton("Ok", null);
    }
    AlertDialog alertDia = builder.create();
    alertDia.show();
    alertDia.setCancelable(false);

}
```

Figura 10.16: Método onPostExecute de la Clase SenderTrapv1Background

- **Fase de Pruebas**: En esta iteración se realizaron las pruebas unitarias, enviando todos los tipos de *Traps* que especifica el protocolo. En la Figura 10.17 se observa la recepción de un *Trap* v1 por parte de una entidad administradora que ejecuta la aplicación SNMP JManager v1.0. La notificación que realiza el *Trap* recibido en la imagen es `linkDown`.

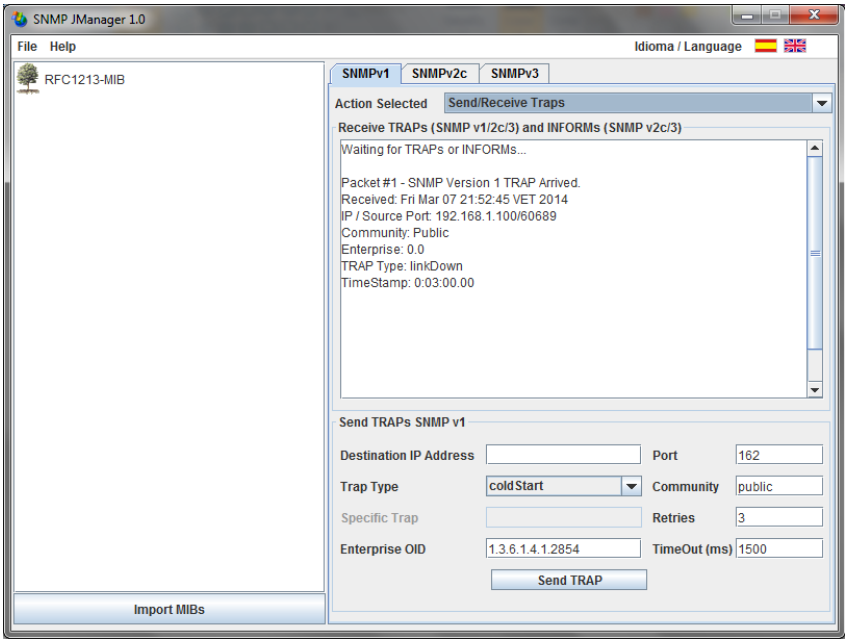

Figura 10.17: Recepción de un Trap v1 por la Aplicación SNMP JManager v1.0

Capítulo 11

Evaluación de Desempeño

En el presente capítulo se muestra un conjunto de pruebas y resultados con el fin de evaluar el rendimiento de la aplicación en distintos dispositivos. Para las pruebas realizadas se tomaron en cuenta las siguientes métricas:

- Round Trip Time (RTT): Es el tiempo que transcurre entre el momento que se envía una petición SNMP y la recepción de la respuesta correspondiente.
- Packet Delivery Ratio (PDR): Es la relación entre el número de respuestas recibidas y el número de peticiones SNMP enviadas.

En la Tabla 11.1 se describen los equipos que se utilizaron para las pruebas, mostrando algunas características que son determinantes en los resultados obtenidos.

Característica	Galaxy Nexus i9250	Galaxy Tab 8.9	PCs
Marca	Samsung	Samsung	HP
Modelo	i9250	GT-P7300	HP xw4600
Procesador	Dual-Core 1.2 GHz	Dual-Core 1 GHz	Core 2 Duo 2.6 GHz
Memoria RAM	1 GB	1 GB	4 GB
OS	Android v4.3	Android v3.0	Windows 7 Pro

Tabla 11.1: Dispositivos Utilizados en las Pruebas de Rendimiento

Dentro del marco investigativo del presente Trabajo Especial de Grado, se llevó a cabo la búsqueda de una aplicación o programa que pudiera realizar pruebas de rendimiento sobre dispositivos que soporten SNMP (específicamente agentes). Los resultados no fueron concluyentes ya que no se encontró ninguna aplicación que realice dichas pruebas, por lo que se tuvo que desarrollar una herramienta que pudiera medir el RTT y el PDR de un agente SNMP. En la Sección 11.1 se describirá el funcionamiento y forma de uso de la herramienta desarrollada.

11.1. Herramienta BenchmarkSNMP

BenchmarkSNMP es una aplicación de consola desarrollada en Java, que posee funcionalidades para elaborar pruebas de rendimiento sobre dispositivos que ejecuten un agente SNMP. Las métricas que evalúa la herramienta son el PDR y el RTT. Para cada métrica evaluada, se debe suministrar un conjunto de parámetros a la aplicación.

BenchmarkSNMP soporta SNMPv1 y SNMPv2c. En el caso de la métrica del PDR, se puede variar la cantidad de mensajes SNMP enviados por segundo y el tiempo que dura la ejecución de la prueba. Para el mensaje SetRequest, soporta una gran variedad de tipos de datos.

11.2. Fase de Pruebas y Análisis de Resultados

En esta sección se muestran los resultados obtenidos con la ejecución de la herramienta BenchmarkSNMP. Las pruebas de PDR se realizaron variando (1) la cantidad de OIDs enviados en un mensaje, (2) la cantidad de mensajes por segundo y (3) el número de computadores enviando mensajes al mismo tiempo. Con respecto al RTT, se varió el número de OIDs por paquete.

11.2.1. Pruebas de PDR del Mensaje GetRequest

A continuación se realiza la evaluación del mensaje GetRequest para los dispositivos descritos en la Tabla 11.1 en distintos escenarios.

11.2.1.1. Escenario con un OID

En la Figura 11.1 se muestran los resultados obtenidos para la prueba con 1 OID, variando el número de PC entre 1 y 3. En las abscisas, se varía el número de mensajes SNMP enviados por segundo por cada PC (1, 2, 4, 6, 8, 10, 12, 14 y 16). Para cada valor del número de mensajes enviados por segundos, hay 6 barras. Las dos primeras barras representan experimentos con 1 PC emisor, para el teléfono Galaxy Nexus i9250 y la tableta Galaxy Tab 8.9, en este orden. Las siguientes dos barras (tercera y cuarta) representan los resultados del PDR, con 2 PCs emisores, para el teléfono Galaxy Nexus i9250 y la tableta Galaxy Tab 8.9, respectivamente. Las últimas dos barras del grupo de seis representa los resultados del PDR, con 3 PCs emisores, para los dispositivos probados. Los resultados que se muestran son esperados ya que en todo par de columnas, el dispositivo con mayor capacidad de cómputo (Galaxy Nexus i9250) mantiene una ventaja aceptable sobre el otro (Galaxy Tab 8.9). Para valores de mensajes por segundo entre 1 y 10, se observa un buen comportamiento teniendo un PDR del 100 % con 1 PC. Para más de 1 PC, los valores del PDR empiezan a decaer un poco antes de 8 mps.

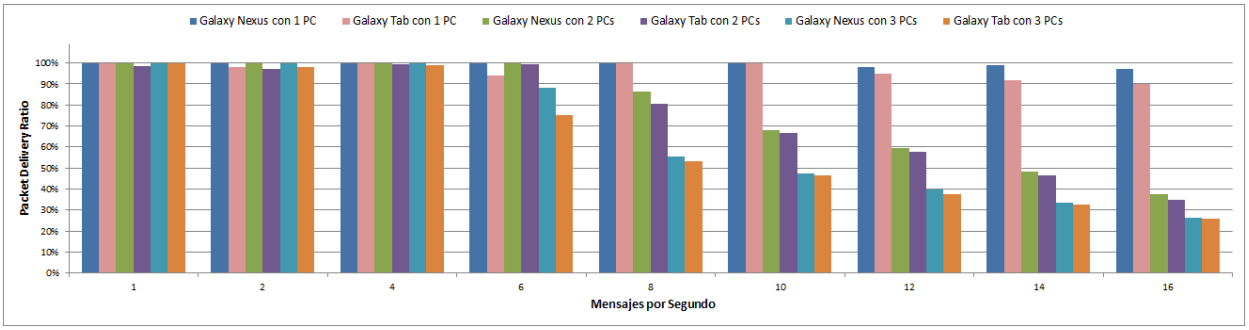

Figura 11.1: PDR del Mensaje GetRequest con 1 OID

11.2.1.2. Escenario con tres OIDs

En este nuevo escenario de pruebas, se realizan experimentos con 3 OIDs y un número de PC que varía entre 1 y 3. Se observan buenos resultados aun cuando se aumenta el número de OIDs por mensajes. Es notable que entre 1 y 8 mps, se obtiene un PDR del 100 % (ver Figura 11.2).

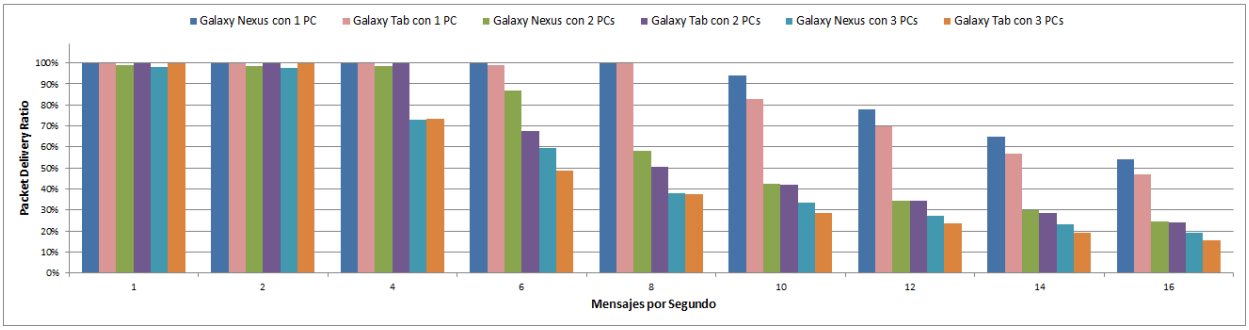

Figura 11.2: PDR del Mensaje GetRequest con 3 OIDs

11.2.1.3. Escenario con seis OIDs

En este escenario, los valores de los resultados disminuyen un poco, debido al número de OIDs que se envía en cada mensaje, lo cual acarrea mayor procesamiento. Aún cuando hay mayor tiempo de procesamiento, se obtiene en algunos casos un PDR del 100 % en ambos dispositivos como se puede apreciar en la Figura 11.3.

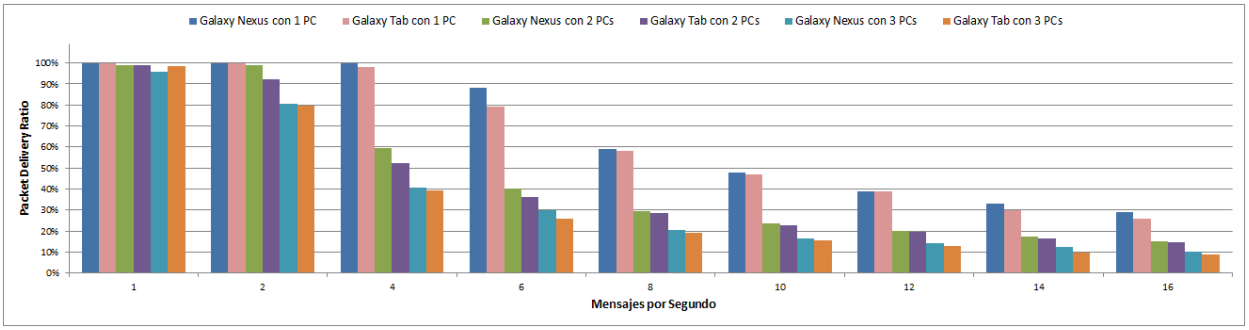

Figura 11.3: PDR del Mensaje GetRequest con 6 OIDs

11.2.2. Pruebas de PDR del Mensaje GetNextRequest

A continuación se realiza la evaluación del mensaje GetNextRequest para los dispositivos descritos en la Tabla 11.1 en distintos escenarios.

11.2.2.1. Escenario con un OID

En la Figura 11.4 se observa los resultados obtenidos al enviar mensajes GetNextRequest con 1 OID, variando el número de PCs desde 1 hasta 3. Se obtiene un comportamiento parecido al que se obtuvo en las pruebas con el mensaje GetRequest (ver Figura 11.1). El dispositivo que posee mayor poder de cómputo (Galaxy Nexus i9250) sigue teniendo ventaja sobre el otro (Galaxy Tab 8.9), y con valores del PDR de 100 % en algunos casos, en ambos dispositivos.

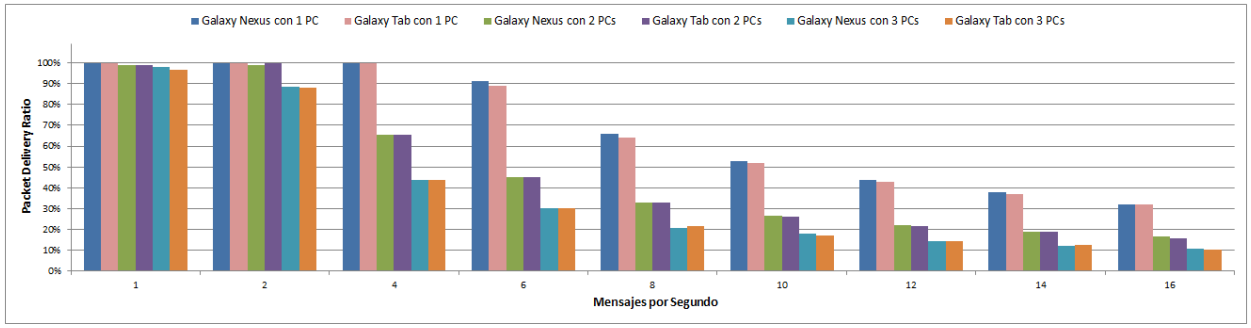

Figura 11.4: PDR del Mensaje GetNextRequest con 1 OID

11.2.2.2. Escenario con tres OIDs

En la Figura 11.5 se observa que existe un poco más de pérdida que en el caso anterior (ver Figura 11.4), ya que la búsqueda realizada para buscar el siguiente OID en el árbol de la MIB consume tiempo. Por lo tanto, el PDR disminuye a medida que se aumenta el número de mensajes por segundo.

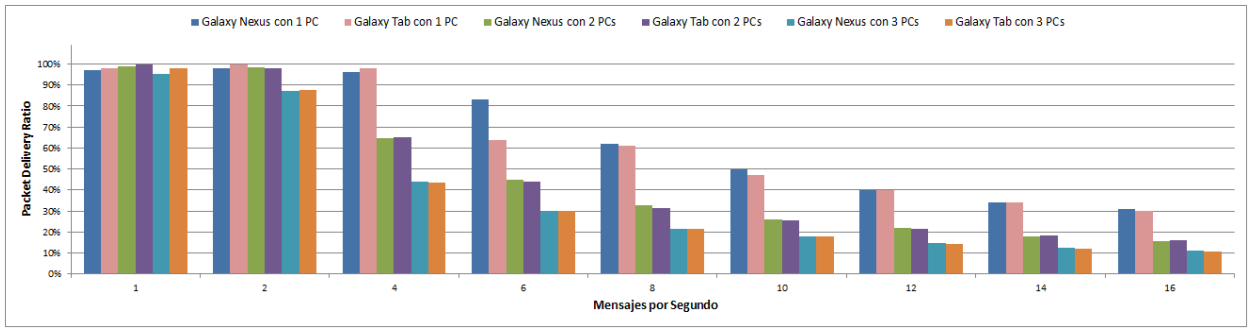

Figura 11.5: PDR del Mensaje GetNextRequest con 3 OIDs

11.2.2.3. Escenario con seis OIDs

Para este escenario, aún con el aumento de OIDs por cada mensaje, se obtienen resultados de PDR por encima del 95 % que va disminuyendo considerablemente. En la Figura 11.6 se observa que debido al tiempo de procesamiento de cada mensaje, existe una pérdida considerable al enviar 8 mps o más.

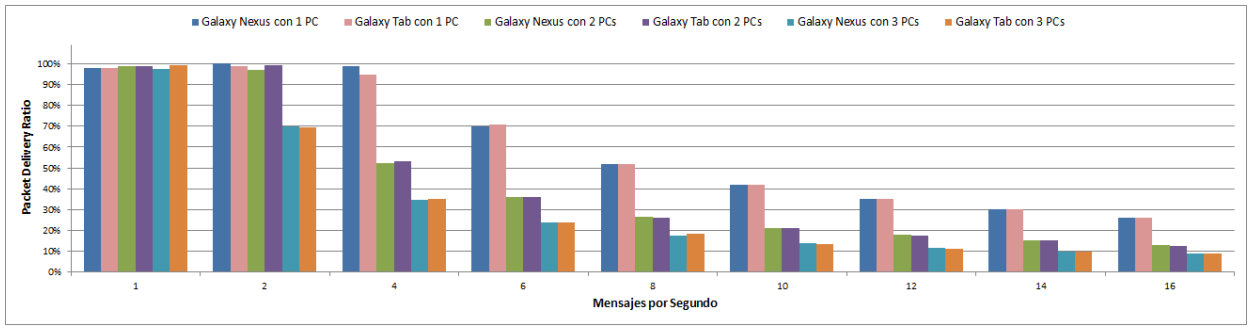

Figura 11.6: PDR del Mensaje GetNextRequest con 6 OIDs

11.2.3. Comparación de PDR para Diferentes Mensajes SNMP para el Galaxy Tab 8.9

En la Figura 11.7 se observa el PDR de los mensajes GetRequest, GetNextRequest y el SetRequest en un escenario de prueba donde 3 PCs envían mensajes con 6 OIDs. Los resultados son muy similares con una ligera ventaja para el GetRequest ya que es la petición que requiere menos cómputo.

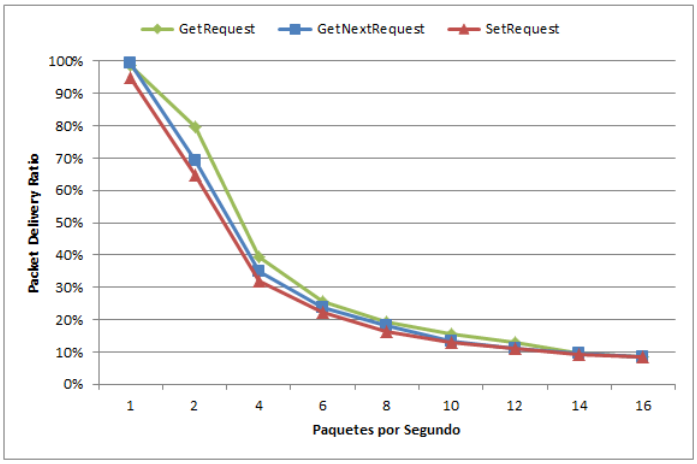

Figura 11.7: PDR de los Mensajes SNMP para el Galaxy Tab 8.9

11.2.4. Comparación de PDR para Diferentes Mensajes SNMP para el Galaxy Nexus I9250

Para el mensaje SetRequest se realizó solo la prueba de mayor exigencia (3 PCs con 6 OIDs). En la Figura 11.8 se observa que los valores de PDR para el mensaje SetRequest tienen un comportamiento muy parecido al mensaje GetNextRequest. Al igual que en la Figura 11.7, el mensaje GetRequest sigue teniendo una ventaja sobre los otros mensajes.

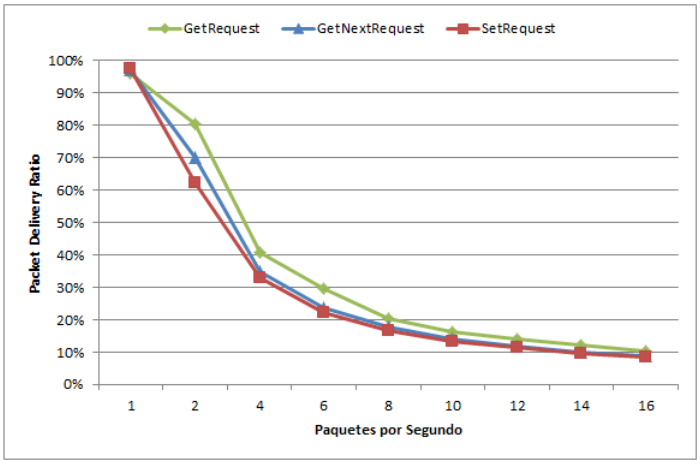

Figura 11.8: PDR de los Mensajes SNMP para el Galaxy Nexus I9250

11.2.5. Comparación del RTT para Diferentes Mensajes SNMP para el Galaxy Tab 8.9

En esta sección se muestran los resultados obtenidos para la prueba del RTT sobre el dispositivo Galaxy Tab 8.9. Como se mencionó al principio de este capítulo, las pruebas del RTT se realizaron variando la cantidad de OIDs que se enviaban en un mensaje. Al igual que las pruebas del PDR, las pruebas del RTT se ejecutaron con la herramienta BenchmarkSNMP.

En la Figura 11.9 se puede observar el RTT de cada mensaje. El comportamiento creciente que se obtiene es el esperado ya que al tener una mayor cantidad de OIDs por mensaje hay un mayor procesamiento, y por ende se tiene un tiempo de respuesta mayor. Se observa claramente una gran diferencia en los tiempos de respuesta del mensaje GetRequest y GetNextRequest debido a la búsqueda en el árbol de la MIB. Por otro lado el mensaje SetRequest tiene un comportamiento lineal creciente al aumentar el número de OIDs.

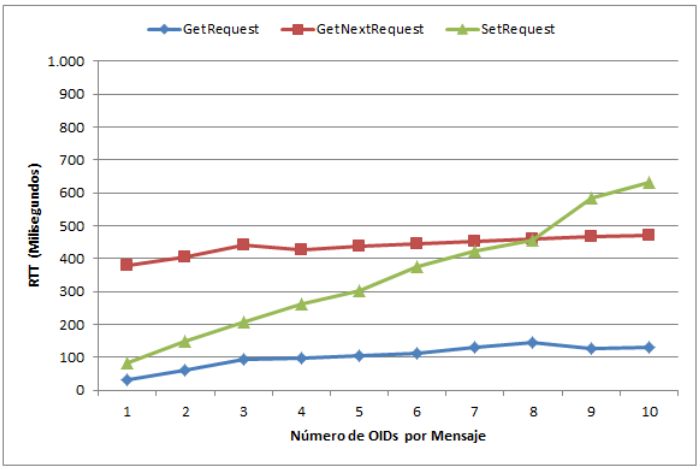

Figura 11.9: RTT de los Mensajes SNMP para el Galaxy Tab 8.9

11.2.6. Comparación del RTT para Diferentes Mensajes SNMP para el Galaxy Nexus I9250

En esta sección se muestran los resultados obtenidos para la prueba del RTT sobre el dispositivo Galaxy Nexus I9250. Como se mencionó al principio de este capítulo, las pruebas del RTT se realizaron variando la cantidad de OIDs que se enviaban en un mensaje. Al igual que las pruebas del PDR, las pruebas del RTT se ejecutaron con la herramienta BenchmarkSNMP.

Como se observa en la Figura 11.10, los resultados obtenidos son muy buenos para este dispositivo. Para el mensaje GetRequest se obtienen tiempos por debajo de los 100 ms y un comportamiento lineal con un creciemiento suave a medida en que se aumenta la cantidad de OIDs. Para el mensaje GetNextRequest se tiene un comportamiento muy parecido al GetRequest solo que posee una diferencia de 200 ms aproximadamente, debido al recorrido del árbol buscando el próximo OID. Con respecto al mensaje SetRequest, se tiene un comportamiento lineal que va creciendo a medida que se aumenta el número de OIDs, llegando a aproximadamente 560 ms.

Se puede observar que este dispositivo, por tener mejor procesador que el Galaxy Tab 8.9, tiene mejores tiempos de respuesta, pero un comportamiento muy parecido al antes mencionado.

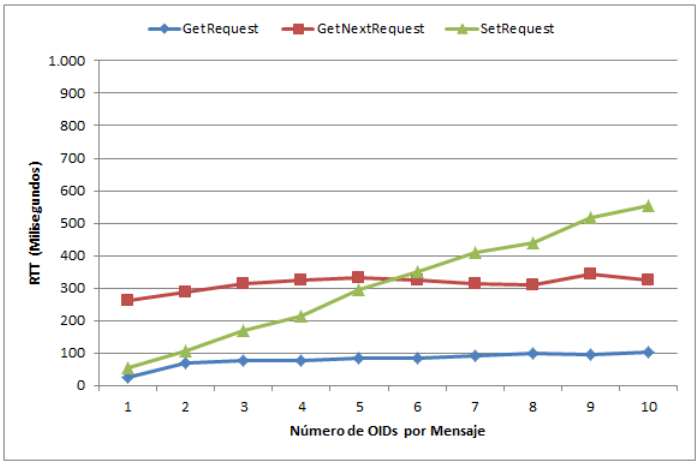

Figura 11.10: RTT de los Mensajes SNMP para el Galaxy Nexus I9250

Capítulo 12

Conclusiones y Trabajos Futuros

La administración de redes con el pasar del tiempo se hizo una tarea más compleja debido al aumento de dispositivos que se necesitaban monitorear. Con la llegada de las redes WLAN a las empresas, existe un dinamismo en el número de dispositivos conectados a la red, lo cual dificulta aun más la administración de la misma.

La mayoría de los equipos conectados a las redes WLAN son dispositivos móviles (*SmartPhones* o *Tablets*) que pueden ser personales o proveídos por la empresa para facilitar las tareas diarias del personal de la empresa. Hoy en día, la administración de estos equipos móviles es muy básica ya que no existe una comunicación directa con ellos, esta se debe hacer a través de los *Access Points* (APs) o de los routers inalámbricos lo que haría muy extensa la configuración de los mismos.

Por lo antes expuesto, en este Trabajo Especial de Grado, se creó una aplicación bajo la plataforma Android, la cual permite la administración a través de un agente SNMP para los dispositivos móviles que posean esta plataforma. *AgentDroid*, como se titula la aplicación, posee un módulo de configuración a través de GUI, que permite ejecutar el agente SNMP en *background*. Las MIBs implementadas se pueden consultar o modificar a través de SNMPv1 o SNMPv2c.

AgentDroid soporta en su totalidad los OIDs que pertenecen al árbol de la `MIB-II` y como prueba de que se puede extender su soporte a través del API, implementa el grupo de software instalados que pertenece a la MIB `HOST-RESOURCES`. Para validar el agente desarrollado, se realizaron unas pruebas de desempeño para obtener el RTT y el PDR.

Finalmente, es necesario mencionar que a pesar de que todos los objetivos propuestos al inicio de la investigación fueron alcanzados con éxito, se considera que existen ciertos trabajos complementarios que enriquecerían significativamente la aplicación. Dichos trabajos son los recomendados a continuación:

- Dar soporte a SNMPv3.
- Implementar una mayor cantidad de MIBs.
- Restringir las direcciones IPs que realizan consultas o modificaciones.
- Agregar un módulo para mostrar y configurar la información de red.
- Crear MIBs privadas que se basen en la información que se puede obtener a través del API (registro de llamadas telefónicas, libro de contactos, mensajes, sistema de archivos).

Referencias Bibliográficas

[1] J.Case, M. Fedor, M. Schoffstall, and J. Davin. *A Simple Network Management Protocol (SNMP).* RFC 1157. May 1990.

[2] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. *Introduction to Community-Based SNMPv2.* RFC 1901. January 1996.

[3] J. Galvin and K. McCloghrie. *Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2).* RFC 1445. April 1993.

[4] R. Presuhn. *Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP).* RFC 3416. December 2002.

[5] D. Harrington, R. Presuhn, and B. Wijnen. *An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks.* RFC 3411. December 2002.

[6] J. Case, D. Harrington, R. Presuhn, and B. Wijnen. *Message Processing and Dispatching for the Simple Network Management Protocol (SNMP).* RFC 3412. December 2002.

[7] J. Case, R. Mundy, D. Partain, and B. Stewart. *Introduction and Applicability Statements for Internet Standard Management Framework.* RFC 3410. December 2002.

[8] U. Blumenthal and B. Wijnen. *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3).* RFC 3414. December 2002.

[9] B. Wijnen, R. Presuhn, and K. McCloghrie. *View-Based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP).* RFC 3415. December 2002.

[10] M. Rose and K. McCloghrie. *Structure and Identification of Management Information for TCP/IP-Based Internets.* RFC 1155. May 1990.

[11] K. McCloghrie, D. Perkins, and J. Schoenwaelder. *Structure of Management Information Version 2 (SMIv2).* RFC 2578. April 1999.

[12] K. McCloghrie and M. Rose. *Management Information Base for Network Management of TCP/IP-based internets.* RFC 1156. May 1990.

[13] K. McCloghrie and M. Rose. *Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II.* RFC 1213. March 1991.

[14] F. Martínez. *Aplicaciones para Dispositivos Móviles.* Universidad Politécnica de Valencia. Años 2010-2011.

[15] J. Cano y D. Sánchez. *Investigación de Sistemas Operativos para Dispositivos Móviles entre 2008-2010, con objeto de Seleccionar el más Adecuado a Emplear como Plataforma Tecnológica para el Proyecto de Investigación "MEDIR".* Universidad Tecnológica de Pereira. Noviembre 2010.

[16] L. Figueroa. *SmartPhones: una Revolución en las Comunicaciones.* Universidad Francisco Gavidia. Revista Cuatrimestral, N° 33. Septiembre-Diciembre 2011.

[17] StatCounterGlobalStats. *Top 8 Mobile OS from Jan 2009 to Jan 2013 [En Línea].* <http://gs.statcounter.com/#mobile_os-ww-monthly-200901-201301>.

[18] M. Báez, A. Borrego, J. Cordero, L. Cruz, M. González, F. Hernández, D. Palomero, J. Rodríguez, D. Sanz, M. Saucedo, P. Torralbo y Á. Zapata. *Introducción a Android.* Universidad Complutense de Madrid. Julio 2012.

[19] M. Murphy. *Beginning Android.* Apress. June 2009.

[20] J. Aranaz. *Desarrollo de Aplicaciones para Dispositivos Móviles sobre la Plataforma Android de Google.* Universidad Carlos III de Madrid. Enero 2009.

[21] S. Hashimi and S. Komatineni. *Pro Android.* Apress. July 2009.

[22] W. Frank, C. Collins, and R. Sen. *Unlocking Android, A Developer's Guide.* Manning. June 2008.

[23] R. Meier. *Professional Android Application Development.* Wrox. November 2008.

[24] J. Balaguero. *Estudio de la Plataforma Android.* Universidad Politécnica de Cataluña. Julio 2008.

[25] K. Jiménez, W. López, and E. Gamess. *AdminUCV NGN: An Open Source Didactic Application for Network Management with Support for IPv4 and IPv6*. In proceedings of the 2009 International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS'09), Las Vegas, Nevada, USA, July 2009.

[26] G. Ayala, P. Poskal, and E. Gamess. *SNMP JManager: An Open Source Didactic Application for Teaching and Learning SNMP v1/2c/3 with Support for IPv4 and IPv6*. In proceedings of the seventh Latin American and Caribbean Conference for Engineering and Technology (LACCEI'2009), San Cristóbal, Venezuela, June 2009.