ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE INGENIERÍA

DESARROLLO DE UN SISTEMA DE MONITOREO PARA LA OBTENCIÓN DE LA INFORMACIÓN DE RED Y EL GRÁFICO DE SU TOPOLOGÍA, BASADO EN LA UTILIZACIÓN DE LOS PROTOCOLOS SNMP E ICMP

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

CARLOS ALONSO CONTRERAS GALLO

DIRECTOR: ING. CARLOS HERRERA

Quito, enero de 2006

DECLARACIÓN

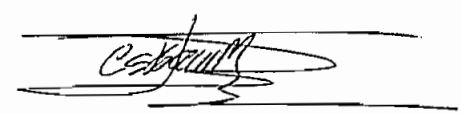
Yo Carlos Alonso Contreras Gallo, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y , que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Carlos Alonso Contreras Gallo

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Carlos Alonso Contreras Gallo, bajo mi supervisión.



Ing. CARLOS HERRERA
DIRECTOR DE PROYECTO

AGRADECIMIENTOS

Agradezco a Dios por sobre todas las cosas, ya que sin Él, nada sería posible de alcanzar.

Agradezco a mis padres, porque me impulsan cada día a seguir luchando por mis ideales.

Agradezco al Ingeniero Carlos Herrera, por ser un importante guía en el desarrollo de éste proyecto.

Un especial agradecimiento al Ingeniero y Amigo Héctor Moya; por sus consejos, enseñanzas y horas de dedicación para hacer posible éste trabajo.

A mi querida Gaby y a mis amigos: Hernán, Juan, Xavier y Esteban; por su apoyo, afecto y por estar presentes en los buenos y malos momentos de la vida.

Aquel que posee un buen amigo, tiene un tesoro en sus manos, y comprende que nada en la vida se puede realizar, si no hay un compañero que te impulse y te extienda su mano, para vencer los obstáculos.

CONTENIDO

RESUMEN

La presente Tesis: "Desarrollo de un Sistema de Monitoreo para la obtención de información de red y el gráfico de su topología, basado en la utilización de los protocolos SNMP e ICMP", considero que es una herramienta de ayuda para monitorear una red, y que concretamente contiene lo siguiente:

Capítulo Primero. Trata de conceptos claves en la Administración de la Red, presenta el protocolo ICMP como herramienta de exploración, el uso del protocolo SNMP para gestión de redes y avances hacia la versión 2 del mismo; la estructura de Información de Administración SMI, la Base de la Información de Gestión MIB y mejoras en la versión MIB II; y esboza brevemente el uso de los extremos virtuales para la comunicación con otros dispositivos.

Capítulo Segundo. Indica en primera instancia el ambiente de programación que ofrece Visual C++, el uso de sockets para la comunicación con otros componentes de red, estableciendo de ésta manera los elementos necesarios para el desarrollo de la aplicación; terminando con el diseño e implementación propiamente dicho de las clases que conforman tal aplicación.

Capítulo Tercero. Muestra las pruebas y resultados obtenidos utilizando el software diseñado, en diferentes redes ya definidas, exponiendo además las herramientas adicionales desarrolladas para un mejor monitoreo de red.

Capítulo Cuarto. Contiene las principales conclusiones y recomendaciones que se pueden ofrecer, como resultado de la realización de éste trabajo.

PRESENTACIÓN

Con el avance tecnológico, la utilización de redes resultan cada día más indispensables tanto para las empresas como para las instituciones; de tal manera que, a medida que se establece un mayor número de redes de área local, también se crea la necesidad de una gestión normalizada de cada una de ellas, siendo SNMP (Simple Network Management Protocol) o Protocolo Simple de Administración de Redes, el protocolo más utilizado para ésta tarea.

Las primeras herramientas utilizadas para el monitoreo de redes fueron muy simples como: Ping, a través del protocolo ICMP, que permite detectar la conectividad entre direcciones; ARP, para detectar interfaces de red; Traceroute, para detectar posibles rutas para que un paquete alcance su destino; etc., tratando de encontrar un protocolo estructurado para el monitoreo de redes de área local; con lo cual se crea SNMP, que al trabajar en conjunto con todas las anteriores, crean un grupo de elementos de monitoreo sólido y versátil, ampliando el panorama con ideas nuevas para una mejor administración de red.

Hay nuevas versiones del protocolo SNMP, como por ejemplo SNMPv2; considerándose ésta como nueva propuesta para el monitoreo de redes, aperturando así un nuevo camino para el mayor control en la gestión de una red.

Aprovechando que los agentes se encargan de coleccionar información que pueda ser relevante a cerca del funcionamiento de la red, una estación de monitoreo toma la información coleccionada por el agente, y se encarga de realizar estadísticas de alto nivel y de interpretar la información obtenida, de tal manera que, se pueda cumplir con los objetivos del monitoreo de redes.

El presente trabajo procura el desarrollo del software, que permita tal recolección de datos para la administración de una red, conjugando el protocolo ICMP y el protocolo SNMP en una aplicación, demostrando además mediante un esquema aproximado, la topología de una red y reportes relacionados con la misma.

CAPÍTULO 1: FUNDAMENTOS TEÓRICOS

1.1 PANORAMA INICIAL

El presente proyecto hace referencia a conceptos de administración y monitoreo de una red, por lo que se trata en este capítulo algunas definiciones a cerca de herramientas y protocolos que se utilizan para este fin; tales como el Protocolo SNMP (Simple Network Management Protocol), UDP (User Data Protocol) e ICMP (Internet Control Message Protocol). Así como definiciones acerca de la Base de Datos de la Información de Gestión (MIB) presente en equipos administrables, dentro de una red.
Esto permite tener una idea clara del tipo y manejo de datos recogidos, para la graficación de la topología de una red.

1.2 ADMINISTRACIÓN – MONITOREO

El monitoreo de redes tiene como principal objetivo coleccionar información útil acerca del funcionamiento de la red, y luego utilizarla para detectar tendencias y planear un mejor desempeño de la misma. También permite reducir cuellos de botella optimizando el servicio, detectar fallas y realizar diagnósticos que permitan lograr arreglos antes de que los usuarios finales vean mensajes de error e incrementar la disponibilidad y utilización del sistema teniendo en cuenta cuestiones de seguridad[1].

El objetivo fundamental es recopilar información útil de los equipos de ruteo administrables, tal que permita esquematizar de manera aproximada, la topología de una red. En caso de no haber equipos administrables, se esquematiza la topología de red de acuerdo a los dispositivos encontrados en la misma, tomando en cuenta que se tratará en ese caso, de una concentración de elementos alrededor de un solo centro.

---

[1] *Inchauspe Inés E., Monitoreo de redes, Universidad Nacional de Luján, 2001.*

1.2.1 RECURSOS DE RED A SER GESTIONADOS[2]

Los recursos de red a ser gestionados pueden ser: hardware y software.

Entre los elementos de hardware se cuentan:

- Conexiones y medio físico.
- Componentes de computadoras
- Elementos de conectividad e interconexión.
- Elementos de red para comunicaciones.

Los recursos de software que requieren administración son los siguientes:

- Sistemas operativos.
- Aplicaciones(cliente/servidor, por ejemplo).
- Software de interconexión.
- Software para comunicaciones de datos.

1.2.2 ADMINISTRACIÓN DE RED

La Administración de la Configuración como parte importante para la Administración de Red; trata sobre el monitoreo de la configuración de la red, siendo entonces ésta la base conceptual, sobre la cual se desarrolla una herramienta que permita extraer información de red, y así esquematizar la topología aproximada de la misma y la distribución de los equipos que la conforman.

Las primeras herramientas utilizadas para el monitoreo de redes fueron herramientas muy simples como *ping*, la cual permite detectar la conectividad entre hosts; *arp* conoce la dirección IP y busca direcciones MAC; *traceroute,* para detectar posibles rutas para que un paquete alcance su destino; *telnet y finger* para chequear el funcionamiento de las operaciones con el protocolo TCP; posteriormente el protocolo SNMP para gestionar elementos de conectividad

---

[2] *EGAS, Carlos; Gestión de Redes; CLEI; Ecuador 1998.*

como: switches, ruteadores, etc. Y así obtener información estadística sobre el funcionamiento de los mismos, abriendo nuevas posibilidades en la administración de redes.

Este tipo de herramientas que permiten localizar elementos presentes dentro de una red con sus respectivas interfaces, son de gran utilidad en el desarrollo de la aplicación, por lo que serán analizados con mayor detenimiento respecto de su funcionalidad.

Los protocolos de administración de red, especifican la comunicación entre un programa cliente(que un administrador invoca), y un programa servidor(da la información de tipo administrativa) que se encuentra en un dispositivo administrado. La definición de un protocolo establece un conjunto de reglas necesarias para la comunicación, por tanto, se puntualizan los formatos y el significado de los mensajes intercambiados[3].

1.3 ADMINISTRACIÓN EN TCP/IP

El Comité Asesor de Internet (Internet Advisory Board, IAB) ha elaborado o adoptado varias normas para la administración de la red. En su mayoría, éstas se han diseñado específicamente para ajustarse a los requisitos de TCP/IP, aunque cuando es posible cumplen con el modelo OSI.
Para cubrir dichas necesidades se han creado dos protocolos, ambos con funcionalidades parecidas:

1. SNMP: comprende el uso del Protocolo Simple para Administración de la Red (Simple Network Management Protocol, SNMP).

2. TCP/IP(CMOT): comprende las normas OSI para administración de la red, llamados Servicios Comunes de Información sobre la Administración (Common Management Information Services, CMIS), y al Protocolo Común de Información sobre la Administración (*Common Management Information Protocol, CMIP*). IAB ha publicado "Common Management Information

[3] *TANENBAUM Andrew; Redes de Computadoras; Tercera Edición; Prentice Hall; México 1997.*

Services and Protocol Over TCP/IP(CMOT)" como una norma para TCP/IP y para la administración OSI[4].

Para el desarrollo del presente proyecto, el protocolo de administración SNMP, se lo estudia a profundidad ya que es el objetivo fundamental para el monitoreo de red.

1.3.1 SNMP (Simple Network Management Protocol)

SNMP es un protocolo de capa aplicación, el cual permite consultar a los diferentes elementos que forman una red, (*ruteadores, switches, hubs, hosts, modems, impresoras, etc*), existiendo dos versiones SNMPv1 y SNMPv2.
Cada equipo conectado a la red ejecuta unos procesos (agentes), para que se pueda realizar una administración tanto remota como local de la red. Dichos procesos van actualizando *variables* (especie de históricos) en una base de datos, que hosts remotos pueden consultar[5].

Por ejemplo, en el caso de:

- *un **ruteador**: interfaces activas, la velocidad de sus interfaces seriales, número de errores, bytes emitidos, bytes recibidos, ...*
- *en una **impresora**: que se terminó el papel, ...*
- *en un **modem**: la pérdida de conexión, etc*
- *en un **switch**: puertos conectados, desconectar un puerto en el caso de IPs duplicadas, si el host está infectado de virus, etc*

1.3.1.1 SNMP: arquitectura

La Arquitectura de Administración de Red se compone de cuatro componentes principales:

- estación de administración
- agente de administración del dispositivo administrado

---

[4] *DIEZ Eduardo, Monitoreo, Universidad de los Andes, 2004.*
[5] *DIEZ Eduardo, Monitoreo, Universidad de los Andes, 2004..*

- base de información de administración,
- protocolo de administración.

SNMP facilita la comunicación entre la estación administradora y el agente de un dispositivo de red (o nodo administrado), permitiendo que los agentes transmitan datos estadísticos (*variables*) a través de la red a la estación de administración[6].

De mejor manera se puede visualizar lo antes explicado en la figura 1.1

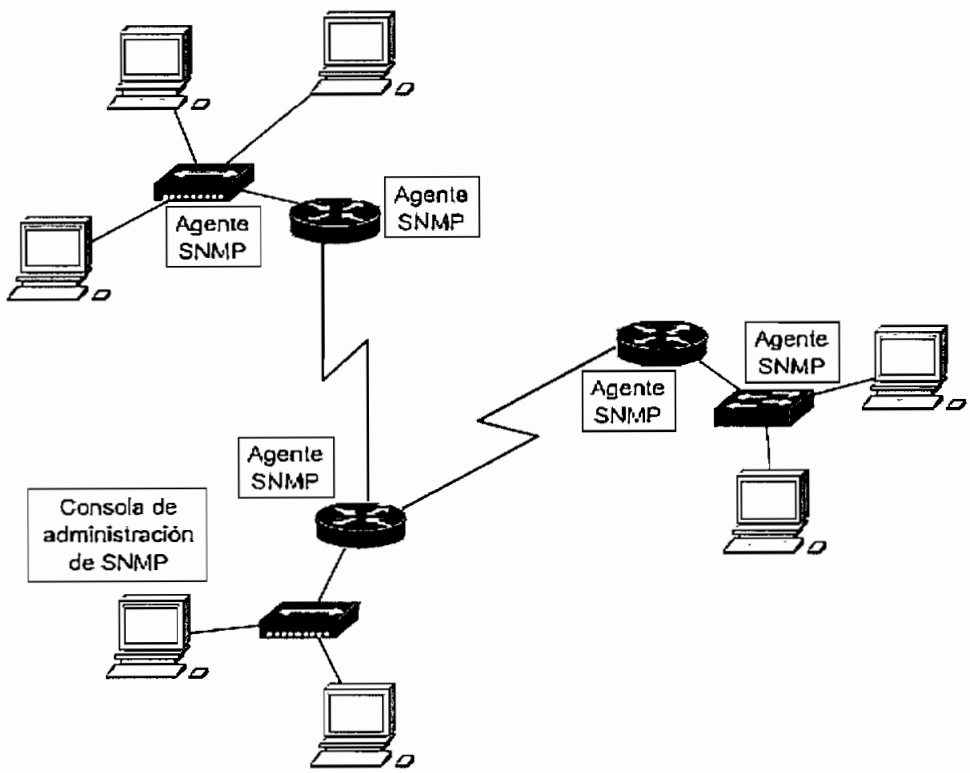


Figura 1.1 Componentes de la arquitectura SNMP

1.3.1.2 Componentes de SNMP

- **Estación de administración** es la interfaz del administrador de red en el sistema. Es la encargada de configurar, cambiar y consultar parámetros en los elementos administrados. Debe contener la información de todos y cada uno de los dispositivos gestionados[7].

---

[6] *DIEZ Eduardo, Monitoreo, Universidad de los Andes, 2004.*
[7] *CASE Jeffrey, SNMP Network Management, McGraw-Hill, 1996.*

- **Agente de administración** lo constituyen los dispositivos que están siendo monitoreados(Puentes, ruteadores, hubs, y switches). El agente se pone en contacto con la estación de administración y le ofrece la información solicitada.

- **Protocolo de administración**: protocolo de capa de aplicación diseñado para comunicar entre el Administrador y el Agente. Se establece de un conjunto de reglas necesarias para la comunicación, puntualizándose los formatos y el significado de los mensajes intercambiados[8]. Éste tipo de protocolos trabajan a nivel de aplicación, por lo que deben proporcionar un mecanismo de autenticación antes del acceso a la información de un agente gestionado.

- **Base de datos de Administración:** Una Base de Datos de Información de Gestión MIB(Management Information Base), contiene una agrupación de objetos; es decir, cada dispositivo administrado consta de información de estado contenido en sus MIBs.

  La función principal de una MIB, es la de proporcionar puntos de acceso(caminos) para la información de gestión, de modo que un agente pueda acceder a la misma de manera rápida. La MIB está organizada en niveles, que a su vez lo hace en módulos que contienen grupos de variables de interrelación[9].

1.3.1.3 SNMP: funcionamiento

La forma normal de uso del SNMP es:

1. **Pregunta**: que la estación administradora envíe una solicitud a un agente (proceso que atiende petición SNMP) pidiéndole información o enviando a actualizar su estado de cierta manera. Este método se conoce como *sondeo*.

---

[8] *TANENBAUM Andrew , Redes de Computadoras, tercera edición. Pearson, 1997*
[9] *McCLOGHRIE K, Rose, Management Information Base for Network Management of TCP/IP, RFC 1156, 1990.*

2. **Respuesta:** la información recibida del agente es la respuesta o la confirmación a la acción solicitada.
3. **Interrupción:** Es mejor que un agente pueda mandar la información al nodo administrador puntualmente, ante una situación predeterminada, por ejemplo una anomalía detectada en la red. Este método es conocido como interrupción[10].

El problema del sondeo es que se incrementa con los nodos administrados y en ocasiones puede llegar a perjudicar el rendimiento de la red.

1.3.1.4 SNMP: Arquitectura del Protocolo

SNMP fue diseñado como un protocolo de capa de aplicación que forma parte de la suite de protocolos TCP/IP. Está propuesto para operar sobre el *User Datagram Protocol* (UDP).

La figura 1.2 muestra como se relaciona una aplicación, de gestión SNMP que se encuentra en una estación de gestión, con un agente SNMP que está en el dispositivo gestionado. Desde la estación de gestión pueden enviarse tres tipos de mensajes SNMP en nombre de la aplicación de gestión: *GetRequest*, *GetNextRequest*, y *SetRequest*. Los dos primeros se usan para leer el valor de los objetos, y el restante para setearlos. Los tres mensajes son confirmados por el agente en forma de un mensaje *GetResponse*. Además, el agente puede enviar un mensaje *Trap* en respuesta a un evento que afecte a los objetos gestionados por el mismo. El mensaje Trap no requiere confirmación por parte de la estación de gestión.

Debido a que SNMP utiliza UDP, protocolo no orientado a conexión, SNMP es en si mismo no orientado a conexión. Por lo tanto, cada intercambio entre la estación de gestión y el agente es una transacción separada[11].

---

[10] *VICENTE Carlos, Gestión de Redes, Universidad de Oregon, 2004.*
[11] *LESCHENNE, Sebastián, Redes de Datos, SNMP – RMON, Universidad Nacional de Rosario, 2002.*

Con UDP, el protocolo SNMP se implementa utilizando los puertos 161 y 162.

- *puerto 161 se utiliza para las transmisiones normales de comando SNMP*
- *puerto 162 se utiliza para los mensajes de tipo "trap" o interrupción.*

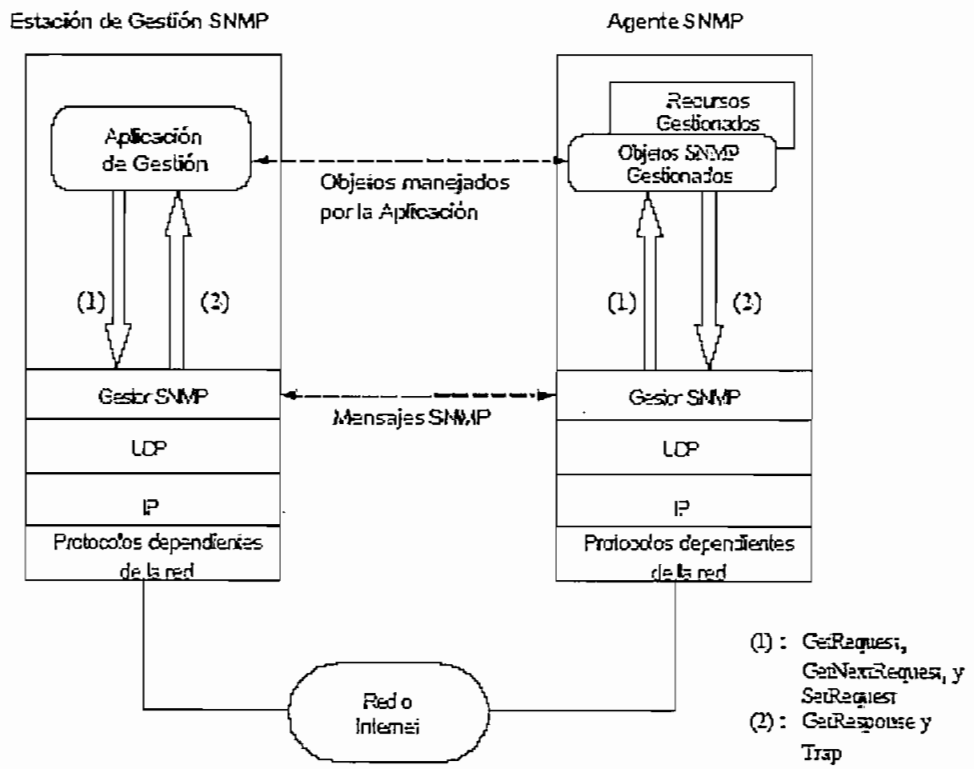


Figura 1.2. Relación entre una aplicación de gestión y los objetos gestionados.

1.3.2 SNMP V1[12]

Las únicas operaciones soportadas por SNMP son la inspección y la alteración de variables.

Específicamente, se pueden realizar tres operaciones:

- *Get*: una estación de gestión recibe un valor de un objeto desde una estación gestionada.
- *Set*: una estación de gestión altera un valor de un objeto en una estación gestionada.

[12] *LESCHENNE, Sebastián, Redes de Datos, SNMP – RMON, Universidad Nacional de Rosario, 2002.*

- *Trap*: una estación gestionada envía un valor de un objeto hacia una estación de gestión sin que esta lo solicite.

No es posible cambiar la estructura de una MIB agregando o borrando instancias de objetos. Tampoco es posible enviar comandos para que se realice una determinada acción. Además, sólo se proporciona acceso a *objetos hojas* dentro del árbol identificador de objetos. Es decir, no es posible acceder a una tabla completa o a una fila de una tabla mediante una acción. Estas restricciones simplifican notablemente la implementación de SNMP, aunque limitan la capacidad del sistema de gestión.

1.3.2.1 Comunidades y Nombres de Comunidad

SNMP involucra, además de las relaciones uno-a-muchos entre una estación de gestión y las estaciones gestionadas, relaciones uno-a-muchos entre una estación gestionada y las estaciones de gestión. Cada estación gestionada controla su propia MIB local y debe ser capaz de controlar la forma en que es usada esta MIB por otras estaciones de gestión. Hay tres aspectos a tener en cuenta:

- *Servicio de Autenticación*: la estación gestionada puede querer limitar el acceso a la MIB por parte de estaciones de gestión autorizadas.
- *Política de Acceso*: la estación gestionada puede querer otorgar diferentes privilegios de acceso a distintas estaciones de gestión.
- *Servicio Proxy*: una estación gestionada puede actuar como proxy de otras estaciones gestionadas. Esto puede involucrar la implementación de un servicio de autenticación y/o una política de acceso para los otros sistemas gestionados en el sistema proxy.

Todos estos aspectos están relacionados con la seguridad. SNMP, tal como está definido en la RFC 1157, proporciona un método de seguridad primitivo y de capacidad limitada conocido como "*comunidad*".
Una comunidad SNMP es una relación entre un agente SNMP y un conjunto de gestores SNMP que definen las características de autenticación, control de acceso y proxy. El concepto de comunidad es un concepto local, definido en la

estación gestionada. El sistema gestionado establece una comunidad para cada combinación deseada de las características de autenticación, control de acceso y proxy. Cada comunidad tiene un nombre de comunidad (dentro del agente), y las estaciones de gestión dentro de esa comunidad están provistas con el nombre de comunidad y deben utilizarlo en todas las operaciones Get y Set. El agente puede establecer varias comunidades.
Como las comunidades están definidas localmente en el agente, el mismo nombre de comunidad puede usarse por diferentes agentes. Esta igualdad de nombres no indica ninguna similitud entre las propiedades otorgadas a cada una de las comunidades.

Un servicio de autenticación es el encargado de asegurar que una comunicación es auténtica. En el caso de un mensaje SNMP, dicho servicio debería asegurarle al receptor que el mensaje proviene de donde dice ser. SNMP proporciona un esquema de autenticación trivial, cada mensaje desde una estación de gestión hacia un agente contiene el nombre de comunidad. Este nombre funciona como contraseña, y el mensaje se asume auténtico si el que lo envía conoce la contraseña.

1.3.2.2 Política de Acceso

Definiendo una comunidad, un agente limita el acceso a su MIB a un grupo seleccionado de estaciones de gestión. Usando más de una comunidad, el agente puede proporcionar diferentes categorías de acceso a la MIB para las distintas estaciones de gestión. Hay dos aspectos para este control de acceso:

- *SNMP MIB view*: un subconjunto de objetos dentro de la MIB. Para cada comunidad pueden definirse diferentes vistas de la MIB. El conjunto de objetos en una vista no necesita pertenecer a un único subárbol de la MIB.
- *SNMP access mode*: un elemento del conjunto (*read-only*, *read-write*). Para cada comunidad se define un modo de acceso.

En la tabla 1.1 se indica la relación entre la cláusula ACCESS de un objeto y el modo de acceso SNMP:

Tabla 1.1. Relación entre la cláusula ACCESS de un objeto y el modo de acceso SNMP.

Cláusula ACCESS del objeto	Modo de Acceso SNMP	
	READ-ONLY	READ-WRITE
read-only	Disponible para las operaciones get y trap	
read-write	Disponible para las operaciones get y trap	Disponible para las operaciones get, set y trap
write-only	Disponible para las operaciones get y trap, pero el valor es específico de la implementación	Disponible para las operaciones get, set y trap, pero el valor es específico de la implementación para las operaciones get y trap
not accesible	No disponible	

1.3.2.3 Formato SNMP

En SNMP la información se intercambia entre la estación de gestión y el agente en forma de mensajes SNMP. Cada mensaje incluye un campo Version con el número de versión de SNMP (version = 0, para SNMPv1), un campo Community con el nombre de comunidad, y uno de los cinco tipos de unidades de datos de protocolo (PDU). La figura 1.3 muestra informalmente esta estructura. El formato de las PDUs de *GetRequest*, *GetNextRequest* y *SetRequest* es el mismo que el de la PDU de *GetResponse*, con los campos error-status y error-index siempre en cero. Esto reduce el número de formatos de PDU con los que debe tratar la entidad SNMP.

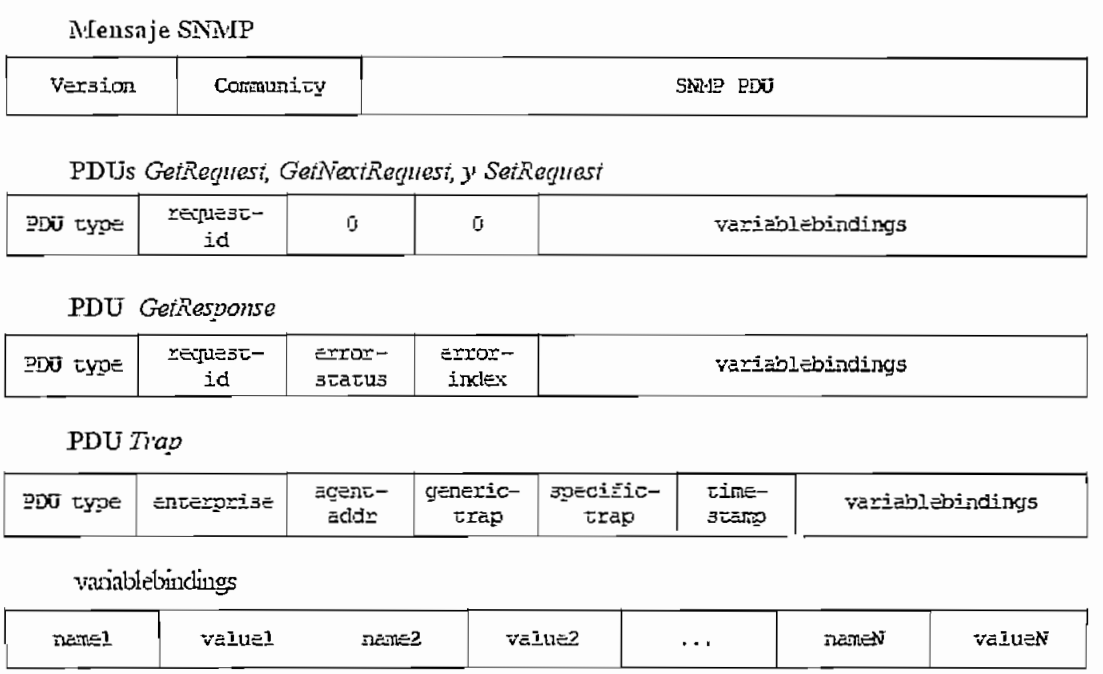


Figura 1.3. Formatos de SNMP.

1.3.2.3.1 GetRequest PDU

La PDU GetRequest es emitida por una entidad SNMP en nombre de una aplicación de la estación de gestión. La entidad emisora incluye los siguientes campos en la PDU:

- *PDU type:* Esto indica que es una PDU GetRequest.
- *request-id*: La entidad emisora asigna números de manera que cada pedido pendiente al mismo agente esté identificado únicamente. El request-id le permite a la aplicación SNMP correlacionar las respuestas recibidas con los pedidos pendientes. También le permite distinguir PDUs duplicadas debidas a un servicio de transporte no confiable.
- *variablebindings*: Esto lista las instancias de los objetos cuyos valores son requeridos.

1.3.2.3.2 GetNextRequest PDU

La PDU GetNextRequest tiene el mismo patrón de intercambio y el mismo formato que la PDU GetRequest. La diferencia está en que, el que responde el pedido, observa en el campo *variablebindings* el nombre del objeto y retorna la instancia del objeto que sigue en orden lexicográfico (el siguiente objeto con un registro almacenado disponible en el dispositivo administrable) al objeto nombrado.

Como ya se indicó, SNMP puede devolver sólo valores de instancias de objetos, por lo que "next" en GetNextRequest significa la próxima instancia de un objeto en orden lexicográfico, y no el próximo objeto.

La pequeña diferencia aparente ente GetRequest y GetNextRequest tiene importantes implicancias. Le permite a una estación de gestión descubrir dinámicamente la estructura de una MIB view, como así también proporciona un mecanismo para la búsqueda de tablas cuyas entradas son desconocidas.

1.3.2.3.3 GetResponse PDU

Una PDU GetResponse contiene el mismo request-id. La operación GetRequest es atómica: se reciben todos los valores, o no se recibe ninguno. Si la entidad receptora puede enviar todos los valores solicitados en la lista de variablebindings, la PDU GetResponse incluye el campo variablebindings con el valor de cada variable. Si al menos uno de los valores no se puede devolver, no se retorna ningún valor.

En una solicitud enviada por el gestor de la red, los subcampos estado del error e índice de error tienen valores nulos, lo mismo ocurre cuando una respuesta por parte del agente está exenta de errores[13].

En la tabla 1.2 se indica los posibles errores en las PDUs:

Tabla 1.2 Errores posibles en las PDUs SNMP.

ERROR	VALOR	SIGNIFICADO
noError	0	Operación correcta entre el administrador y el agente
tooBig	1	El tamaño de la PDU, excede el límite local.
noSuchName	2	El nombre del objeto requerido no coincide con los nombres disponibles en el MIB View
badValue	3	Existe inconsistencia en una PDU Set Request, en cuanto al formato de codificación (tipo, longitud o valor) para una variable.
readOnly	4	No está definido en el RFC 1157.
genErr	5	Se produce cuando otro tipo de error diferente a los ya mencionados ocurre, no se lo define explícitamente

Las diferentes condiciones de error son las siguientes:

[13] *MILLER Mark, Managing Internetworkswith SNMP, M&T Books, USA 1993.*

- Un objeto nombrado en el campo variablebindings no concuerda con ningún OID en la MIB view, o dicho objeto puede ser de un tipo que no tenga ningún valor de instancia asociado (por ejemplo, objetos tablas o filas). En cualquiera de los casos, se devuelve una PDU GetResponse con un error-status de *noSuchName* y un valor en el campo error-index que indica la posición del objeto que causó el problema en el campo variablebindings enviado. Por ejemplo, si la tercera variable del campo variablebindings enviado no está disponible, entonces el campo error-index contiene el valor 3.

- La entidad de respuesta podría devolver todos los valores solicitados, pero el tamaño de la PDU GetResponse excede una limitación local. En este caso, se retorna una PDU GetResponse con un error-status de *tooBig*.

- La entidad de respuesta no puede devolver al menos un valor de los solicitados por alguna otra razón. En este caso, se devuelve una PDU GetResponse con un errorstatus de *genErr* y un valor en el campo error-index indicando la posición del objeto que causó el problema.

Con SNMP se pueden recuperar sólo objetos independientes del árbol MIB. No es posible recuperar, por ejemplo, una fila completa de una tabla (por ejemplo la tabla IP routing) referenciando el objeto entry (por ejemplo, *ipRouteEntry*), o una tabla completa referenciando el objeto tabla (por ejemplo, *ipTable*). Sin embargo, una estación de gestión puede recibir una fila completa o la tabla completa de una sola vez incluyendo cada una de las instancias de los objetos en la lista de variablebindings.

1.3.2.3.4 SetRequest PDU

La PDU SetRequest es emitida por una entidad SNMP en nombre de una aplicación de la estación de gestión. Tiene el mismo patrón de intercambio y el mismo formato que la PDU GetRequest. SetRequest se utiliza para escribir el

valor de un objeto, por lo tanto la lista del campo variablebindings contiene el nombre del objeto y el valor al que se desea setear el mismo.

La entidad receptora de la PDU SetRequest responde con una PDU GetResponse que contiene el mismo request-id. En la operación SetRequest se actualizan todos los valores o no se actualiza ninguno.

Si la estación receptora puede actualizar todos los valores, responde con una PDU GetResponse que incluye en el campo variablebindings el valor de cada variable. Si al menos uno de los no puede setearse, entonces los valores de las variables no se devuelven en la respuesta y ningún valor es actualizado. Pueden devolverse las mismas condiciones que en el caso de GetRequest (*noSuchName*, *tooBig*, *genErr*). Además puede devolverse la condición de error *badValue* cuando hay al menos una inconsistencia entre el nombre y el valor de las variables. Las inconsistencias pueden darse por el tipo, la longitud, o el valor del valor suministrado.

1.3.2.3.5 Trap PDU

La PDU Trap es emitida por una entidad SNMP en nombre de una aplicación del agente. Es utilizado para notificar asincrónicamente a la estación de gestión sobre la ocurrencia de un evento significativo. Su formato es diferente al de otras PDUs. Sus campos son:

- *PDU type*: indica que esta es una PDU Trap.
- *enterprise*: identifica el subsistema de gestión de red que generó el trap (el valor es tomado desde *sysObjectID* en el grupo *System*).
- *agent-addr*: la dirección IP del objeto que generó el trap.
- *generic-trap*: uno de los tipos de traps predefinidos.

- *specific-trap*: un código que indica más específicamente la naturaleza del trap.

- *time-stamp*: el tiempo transcurrido entre la última (re)inicialización de la entidad que generó el trap y la generación del trap.

- *variablebindings*: información adicional relacionada con el trap (el significado de este campo es específico de la implementación).

El campo generic-trap puede tomar uno de los siguientes siete valores:

- *coldStart*(0): La entidad SNMP emisora se reinicializa de manera tal que la configuración del agente o la implementación de la entidad de protocolo puede alterarse. Típicamente, esto es una reiniciación inesperada debido a una falla importante.

- *warmStart*(1): La entidad SNMP emisora se reinicializa de manera tal que la configuración del agente o la implementación de la entidad de protocolo no va a alterarse. Típicamente, esto es un reiniciación de rutina.

- *linkDown*(2): Señala la falla de uno de los enlaces de comunicación del agente. El primer elemento en el campo variablebindings es el nombre y el valor de la instancia *ifIndex* para la interfaz referenciada.

- *linkUp*(3): Señala que uno de los enlaces de comunicación del agente se levantó. El primer elemento en el campo variablebindings es el nombre y el valor de la instancia *ifIndex* para la interfaz referenciada.

- *authenticationFailure*(4): Indica que la entidad emisora recibió un mensaje de protocolo que ha fallado la autenticación.

- *egpNeighborLoss*(5): Señala que un vecino EGP para quién la entidad emisora era un par EGP ha sido dado de baja y la relación ya no existe.

- *enterpriseSpecific* (6): Significa que la entidad emisora reconoce que ha ocurrido algún evento enterprise-specific. El campo specific-trap indica el tipo de trap.

A diferencia de las PDUs GetRequest, GetNextRequest, y SetRequest, la PDU Trap no genera una respuesta desde el otro lado.

La tabla 1.3 muestra la numeración posible para el subcampo tipo de PDU SNMP

Tabla 1.3 Numeración posible para el subcampo tipo de PDU SNMP

Tipo de PDU	Valor del subcampo tipo PDU
GetRequest-PDU	0
GetNextRequestPUD	1
GetResponse-PDU	2
SetRequest-PDU	3
Trap-PDU	4

1.3.3 SNMP V2[14]

La especificación de SNMPv2 se encuentra detallada en las RFC1901 a RFC1908. Las mejoras realizadas en SNMPv2 respecto a SNMPv1 son las que se enumeran a continuación:

- Nueva estructura de la información de gestión (SMI).
- Capacidad de gestionar entre dos estaciones de gestión.
- Nuevas operaciones definidas en el protocolo.
- Nueva Base de Información de Gestión (MIB SNMPv2)

1.3.3.1 Estructura de la Información de Gestión SNMPv2 (SMI SNMPv2)

La nueva estructura de información de gestión (SMI SNMPv2 o SMIv2) introduce cuatro conceptos nuevos:

[14] *LESCHENNE, Sebastián, Redes de Datos, SNMP – RMON, Universidad Nacional de Rosario, 2002.*

- Nuevas definiciones de objetos.
- Nuevas tablas conceptuales.
- Nuevas definiciones para la notificación.
- Nuevos módulos de información.

1.3.3.2 Operaciones de Protocolo

SNMPv2 proporciona tres tipos de acceso a la Información de Gestión:

1. *pedido-respuesta gestor-agente.*
2. *agente-gestor sin confirmación.*
3. *pedido-respuesta gestor-gestor.*

Los dos primeros tipos de interacción son las utilizadas en SNMPv1. El tercer tipo corresponde a un nuevo tipo de acceso. De esta manera una entidad SNMPv2 que actúa en un rol de gestor puede enviar un pedido a otra entidad semejante, y esta última responde al pedido. Esto le permite a una entidad gestora conocer información asociada con otra entidad gestora.

1.3.3.3 Formatos de PDU

Los formatos de las PDUs se muestran informalmente en la figura 1.4. Se puede observar que los formatos de las PDUs *GetRequest*, *GetNextRequest*, *SetRequest*, y *SNMPv2-Trap* son iguales a los de las PDUs de *Response* e *InformRequest*, con los campos *error-status* y *error-index* seteados en cero. Esto reduce los tipos de formatos con los que debe tratar una entidad SNMPv2.

Mensaje SNMPv2

Version	Community	SNMPv2 PDU

PDUs *GetRequest, GetNextRequest, SetRequest,* y *SNMPv2-Trap*

PDU type	request-id	0	0	variablebindings

PDUs *Response* e *InformRequest*

PDU type	request-id	error-status	error-index	variablebindings

PDU *GetBulkRequest*

PDU type	request-id	non-repeaters	max-repetitions	variablebindings

variable-bindings

name1	value1	name2	value2	...	nameN	valueN

Figura 1.4. Formatos de SNMPv2.

Al igual que en SNMPv1, el campo PDU type surge de la codificación. Antes de examinar cada una de las PDUs, veremos los campos comunes en las PDUs SNMPv2:

- *request-id*: igual que en SNMPv1.
- *error-status*: se utiliza para indicar si se produjo o no un error en respuesta a un pedido. La tabla 1.4 indica que códigos son apropiados para cada tipo de pedido. Notar que *noSuchName*(2), *badValue*(3), y *readOnly*(4) se usan para coexistir con SNMPv1.
- *error-index:* igual que en SNMPv1.
- *variable-bindings:* permite aplicar una misma operación a un grupo de instancias. El campo consta de una secuencia de pares, el primer elemento del par es un OID y el segundo es uno de los siguientes:

  - ✓ *value:* el valor asociado con cada instacia de objeto especificada en un pedido.
  - ✓ *unSpecified*: un valor NULL se usa en los pedidos de recuperación.
  - ✓ *noSuchObject*: indica que el agente no implementa el objeto referenciado por el OID.
  - ✓ *noSuchInstance*: indica que la instancia no existe para esta operación.
  - ✓ *endOfMibView*: indica un intento de referencia a un OID que está mas allá del final de la MIB en el agente.

La tabla 1.5 indica, en cuales PDUs se utilizan los valores mencionados.

Tabla 1.4. Uso de códigos de error en la Response-PDU.

	GetRequest, GetNextRequest, GetBulkRequest,	SetRequest	InformRequest
noError (0)	X	X	X
tooBig (1)	X	X	X
noSuchName (2)			
badValue (3)			
readOnly (4)			
genError (5)	X	X	X
noAccess (6)		X	
wrongType (7)		X	
wrongLength (8)		X	
wrongEncoding (9)		X	
wrongValue (10)		X	
noCreation (11)		X	
inconsistentValue (12)		X	
resourseUnavaible (13)		X	
commitFailed (14)		X	
undoFailed (15)		X	
authorizationError (16)	X	X	X
notWritable (17)		X	
inconsistentName (18)		X	

Tabla 1.5. Valores permitidos en la lista variable-bindings.

	value	unSpecified	noSuchObject	noSuchInstance	endOfMibView
GetRequest		X			
Response to GetRequest		X	X	X	
GetNextRequest		X			
Response to GetNextRequest		X			X
GetBulkRequest		X			
Response to GetBulkRequest		X			X
SetRequest	X				
Response to SetRequest		X			
SNMPv2-Trap	X				
InformRequest	X				
Response to InformRequest	X				

1.3.3.3.1 PDU GetRequest

El formato y la semántica de esta PDU son idénticas a la de SNMPv1. La diferencia está en la manera de manejar la respuesta. En SNMPv2 esta operación no es atómica, es decir, se devuelven valores en la lista de variable-bindings aunque no se puedan devolver todos los valores requeridos. Si ocurre alguna

condición de excepción (*noSuchName*, *noSuchInstance*), se devuelve el nombre de la variable junto con la indicación de la excepción en lugar del valor. En SNMPv2, la respuesta se construye procesando cada variable de la lista enviada de acuerdo con las siguientes reglas:

- Si una variable no tiene un prefijo de OID que concuerde exactamente con el prefijo de alguna variable accesible por este pedido, entonces el campo valor es seteado a *noSuchObject*.

- Caso contrario, si el nombre de la variable no concuerda exactamente con el nombre de alguna variable accesible por este pedido, entonces el campo valor es seteado a *noSuchInstance*. (Este caso solo se da para objetos columnas donde la fila no existe o está bajo creación)

- Caso contrario, el campo valor es seteado con el valor de la variable nombrada.

Si el procesamiento del nombre de la variable falla por cualquier otra razón, no se devuelve ningún valor. En cambio, la entidad retorna una PDU response con un error-status de *genErr* y un valor en el campo error-index que indica que variable de la lista variable-bindings causó el problema.

Si el tamaño del mensaje que debería llevar la respuesta excede una limitación local o el máximo tamaño de mensaje del emisor del pedido, se devuelve un mensaje con una PDU que tiene un error-status de *tooBig*, un error-index cero, y el campo variable-bindings vacío.

En SNMPv1, si uno de los valores no se podía devolver, el agente devuelve un mensaje con un error-status de *noSuchName*. En ese caso, el gestor SNMPv1 podía no retornar ningún valor a la aplicación de gestión, o implementar un algoritmo que remueva del pedido las variables que causan problemas para poder devolver un resultado parcial. Como muchas estaciones de gestión no implementan dicho algoritmo, no pueden interoperar eficientemente con agentes

que tienen variables no implementadas. Este problema dio orígen para que muchos agentes devuelvan valores arbitrarios para los objetos no soportados, en lugar de retornar el error *noSuchName*. Con la capacidad de devolver resultados parciales provista por SNMPv2 los agentes deberían eliminar el inconveniente de las asignaciones arbitrarias.

1.3.3.3.2 PDU GetNextRequest

El formato y la semántica de esta PDU son idénticas a la de SNMPv1. La diferencia es que puede devolver resultados parciales.
En SNMPv2, la respuesta se construye procesando cada variable de la lista enviada de acuerdo con las siguientes reglas:

- Se determina la variable (instancia de objeto) que está luego en orden lexicográfico a la varaible nombrada. El par resultante en el campo variable-bindings contiene el nombre y el valor de la variable correspondiente.

- Si no existe sucesor en orden lexicográfico, el par en el campo variable-bindings está formado por el nombre de la variable solicitada y el valor está seteado a *endOfMibView*.

Si el procesamiento de cualquier variable falla por alguna otra razón, se sigue el mismo procedimiento que GetRequest.

1.3.3.3.3 PDU GetBulkRequest

Una de las principales mejoras que proporciona SNMPv2 es esta PDU. Su propósito es minimizar el número de intercambios necesarios para recuperar una gran cantidad de información. La PDU GetBulkRequest le permitie a un gestor SNMPv2 recuperar datos de modo que la respuesta contenga tanta información como sea posible, teniendo en cuenta las restricciones del tamaño de un mensaje para SNMP que viaja en datagramas UDP.

Esta operación utiliza el mismo principio de selección que GetNextRequest, es decir, selecciona la próxima instancia en orden lexicográfico. La diferencia está en que GetBulklRequest permite especificar el número de sucesores lexicográficos a seleccionar. El campo variable-bindings incluye una lista de (N + R) nombres de variable. Para los primeros N nombres opera igual que GetNextRequest. Para los R nombres restantes se devuelven múltiples sucesores lexicográficos.

En esta PDU existen dos campos que no se encuentran en las demás PDUs: non-repeaters y maxrepetitions. El primero especifica el número de variables en la lista variable-bindings para las cuales debe ser retornado un sólo sucesor lexicográfico. El segundo determina la cantidad de sucesores lexicográficos que deben devolverse para las restantes variables en el campo variablebindings.

La operación GetBulkRequest elimina una de las mayores limitaciones de SNMPv1, que es su incapacidad de recuperar grandes bloques de información. Más aún, el uso de esta operación permite reducir el tamaño de las aplicaciones de gestión soportadas por el protocolo. La aplicación no tiene necesidad de tratar con algunos detalles del empaquetamiento de los pedidos.

No necesita hacer un procedimiento de prueba y error para determinar el número óptimo de variable-bindings dentro de una PDU de pedido. Además, si un pedido es demasiado grande, el agente puede retornar tanta información como pueda, en lugar de dar sólo el mensaje de error tooBig. De esta forma, el gestor tiene que simplemente repetir el pedido para los datos faltantes, evitando tener que rearmar el pedido original en una serie de pequeños pedidos.

1.3.3.3.4 PDU SetRequest

Al igual que en los casos anteriores, lo que diferencia a la PDU SetRequest de SNMPv2 con respecto a la de SMNPv1, es la forma en la que se manejan las respuestas.

Primero, el agente determina el tamaño del mensaje que encapsula una PDU de respuesta con la misma lista variable-bindings de nombres y valores. Si el mismo excede una limitación local o remota, se construye una PDU con un error-status de *tooBig*, un error-index de cero, y un campo variable-bindings vacío. Caso contrario, se construye una PDU de respuesta donde todos los campos tienen el mismo valor que los campos correspondientes del pedido recibido.

El campo variable-bindings es conceptualmente procesado en dos fases. En la primera, cada par variable-bindings, que constituye una operación Set individual, se valida. Si todos los pares se validan, luego cada variable se altera en la segunda fase. Entonces, al igual que en SNMPv1, la operación Set de SNMPv2 es atómica.

1.3.3.3.5 PDU SNMPv2-Trap

Cumple la misma función que la PDU Trap de SNMPv1, pero tiene un formato diferente. Dicho formato, es el mismo que el de las otras PDUs de SNMPv2 (excepto GetBulkRequest), facilitando la tarea de procesamiento en el receptor.

El campo variable-bindings en esta PDU contiene los siguientes valores:

- sysUpTime.0.
- snmpTrapOID.0: parte del grupo trap en la MIB SNMPv2.
- Si la cláusula OBJECT está presente en la invocación correspondiente de la macro NOTIFICATION-TYPE, entonces cada variable y su valor se copian en el campo variable-bindings.
- El agente puede optar por la inclusión de variables adicionales.

Al igual que en SNMPv1, no se recibe confirmación de la llegada del trap.

1.3.3.3.6 PDU InformRequest

Este mensaje se envía entre dos entidades SNMPv2 con rol de gestores para proporcionarle información de gestión a la aplicación de la entidad receptora del mensaje.
El campo variable-bindings de la PDU InformRequest contiene los mismos elementos que tiene el SNMPv2-Trap. Cuando se recibe una PDU de este tipo, la entidad receptora primero determina el tamaño del mensaje de respuesta. Si éste excede las restricciones de tamaño, se manda una PDU de respuesta con los valores de error-status en *tooBig*, error-index en cero y variable-bindings vacío.

Caso contrario, la entidad receptora le pasa la información a la aplicación correspondiente y genera una PDU de respuesta con los mismos valores en los campos request-id y variable-bindings que los que tiene el InformRequest recibido, un error-status de *noError* y un valor cero en el campo error-index.

1.3.3.4 SNMP: Primitivas[15]

En resumen SNMP define ocho mensajes que pueden enviarse:

1. GET REQUEST : Solicita uno o mas atributos de un objeto (o variable). Es transmitida por el nodo administrador y recibida por el agente que contesta.
2. GET NEXT REQUEST: Solicita el siguiente atributo de un objeto. Es transmitida por el nodo administrador y recibida por el agente que contesta.
3. GET BULK REQUEST (en SNMP v2): solicita un conjunto amplio de atributos en vez de solicitar uno a uno. Es transmitida por el nodo administrador y recibida por el agente que contesta.
4. SET REQUEST: actualiza uno o varios atributos de un objeto. Es transmitida por el nodo administrador y recibida por el agente que contesta.

[15] *VICENTE Carlos, Gestión de Redes, Universidad de Oregon, 2004.*

5. SET NEXT REQUEST: actualiza el siguiente atributo de un objeto. Es transmitida por el nodo administrador y recibida por el agente que contesta.
6. GET RESPONSE: devuelve los atributos solicitados. Es transmitida por el agente y recibida por el nodo administrador.
7. TRAP: informa fallos como la pérdida de comunicación con un vecino. Es transmitida por el agente y recibida por el nodo administrador.
8. INFORM REQUEST (en SNMP v2): describe la base local de información de gestión MIB para intercambiar información nodos de administración entre sí. Es transmitida por el nodo administrador y recibida por otro nodo administrador.

1.3.4 ESTRUCTURA DE LA INFORMACIÓN DE GESTIÓN GENERAL(SMI)

SMI presenta una estructura en forma de árbol global para la información de administración, convenciones, sintaxis y las reglas para la construcción de MIBs, define el marco general dentro del cual una MIB se puede definir y construir, identifica los tipos de datos que pueden usarse en la MIB y especifica como se representan y denominan los recursos dentro de la MIB[16].

La filosofía detrás de la SMI es fomentar la simplicidad y la extensibilidad dentro de la MIB. Por eso, la MIB puede solamente guardar tipos de datos simples: escalares y arreglos bidimensionales de escalares.

La SMI no soporta la creación o recuperación de estructuras de datos complejas, simplificando así la implementación y mejorando la interoperabilidad. Para proporcionar una forma estandarizada de representación de la información de gestión, la SMI debe proporcionar técnicas para:

- ✓ Definir la estructura de una MIB particular.

[16] *RFC 1155*

- ✓ Definir objetos individuales, incluyendo la sintaxis y el valor de cada objeto.
- ✓ Codificar los valores de los objetos.

- ✓ *Ejemplo de grupos de variables en MIB-2 en la SMI:* System (identifica el hardware y software), AT (traducción de dirección de Ethernet a IP), IP (contador de paquetes, fragmentación), ICMP (contador de cada tipo de mensaje ICMP), TCP y UDP (conexiones abiertas TCP), EGP (estadística de protocolo externo)

- ✓ *Ejemplo de codificación de objetos según SMI: iso.org.dod.internet.mgmt.mib_2.interfaces.iftable.ifEntry.variable.puerto* O su equivalente *.1.3.6.1.2.1.2.2.1.variable.puerto* [17]

Este formato para la representación de variables pueden ser expresadas tanto en ASCII como números separados por puntos, en una notación intermedia entre ASCII y ASN1 conocida como OID (*Object Identifier*) o descriptor.

1.3.5 Base de Información de Gestión (MIBS: Database of Management Information)[18]

La base de información de administración tiene una estructura de base de datos (según SMI) y reside en cada dispositivo administrado.

La base de datos contiene una serie de objetos (variables), que son datos sobre recursos reunidos en el dispositivo administrado

En la figura 1.5 se encuentra un ejemplo de un subárbol contenido en un agente SNMP.

---

[17] *LESCHENNE, Sebastián, Redes de Datos, SNMP – RMON, Universidad Nacional de Rosario, 2002*
[18] *McCLOGHRIE K, Rose, Structure and Identification of Management Information for TCP/IP, RFC 1155, 1991.*

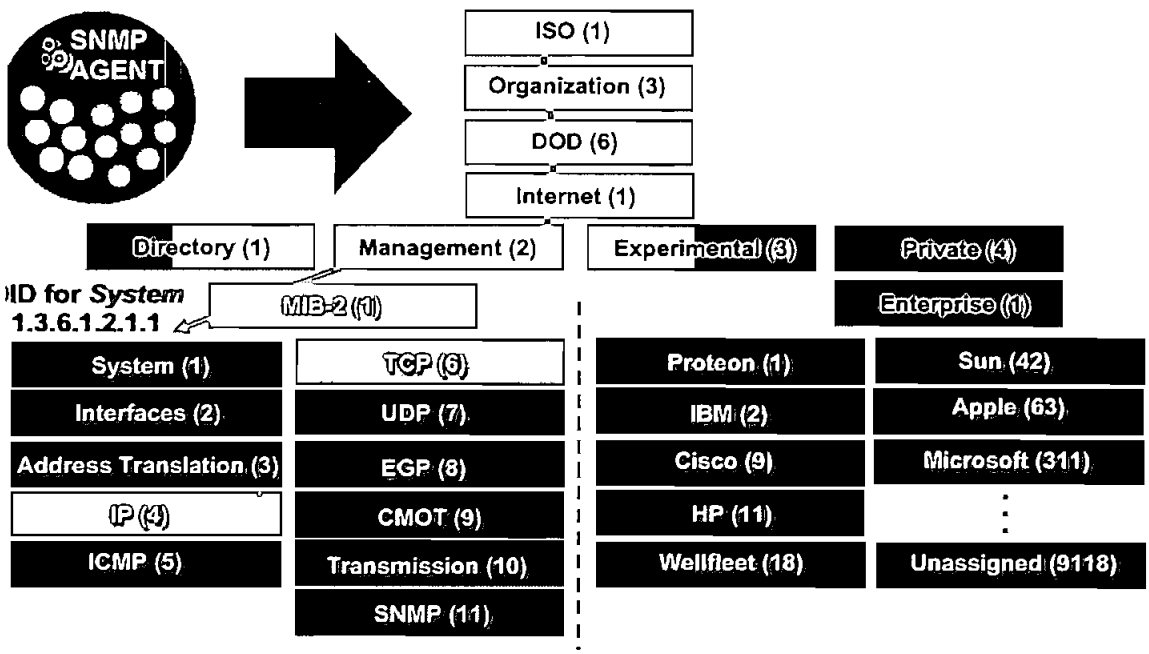


Figura 1.5 Subárboles de la MIB-2

1.3.5.1 Estructura de la MIB

Todos los objetos gestionados en el ambiente de SNMP están ordenados en una estructura de árbol. Las hojas del árbol son los verdaderos objetos gestionados, cada uno de los cuales representa algún recurso, actividad, o información relacionada que será gestionada. La estructura de árbol por sí misma define un agrupamiento de objetos dentro de grupos relacionados lógicamente como se muestra en la figura 1.6.

Asociado con cada tipo de objeto dentro de una MIB hay un identificador de objeto (OID) del tipo OBJECT IDENTIFIER. El OID es único para cada tipo de objeto, y sirve para nombrar al objeto. Su valor consiste en una secuencia de enteros denominados subindentificadores. Como es posible establecer un orden jerárquico a partir de los OID, además de servir para identificar los objetos, los OIDs sirven también para identificar la estructura del árbol. Comenzando por la raíz (sin nombre) del árbol, cada subidentificador del OID identifica un nodo en el árbol. Hay tres nodos en el primer nivel: ccitt (0), iso (1), joint-iso-ccitt (2).

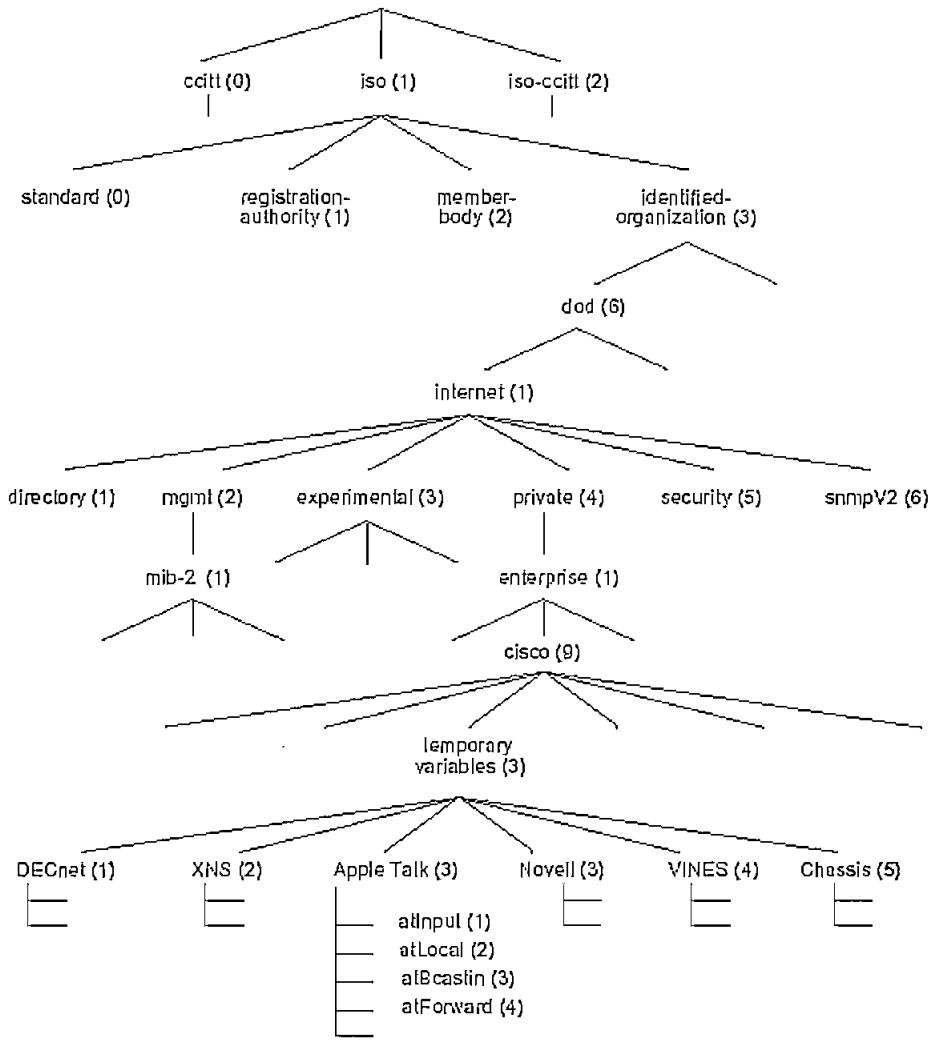


Figura 1.6 Grupos integrantes del subnodo Internet.

Bajo el nodo iso, un subárbol es para uso de otras organizaciones, una de las cuales es el *U.S. Departament of Defense* (dod). La RFC 1155 asume que un subárbol bajo el nodo dod será ubicado por el *Internet Activities Board* (IAB) como se muestra en el gráfico anterior.

Entonces, el nodo internet(1) tiene un valor de OID de 1.3.6.1. Este valor sirve como prefijo para los nodos del siguiente nivel inferior del árbol. El documento SMI define cuatro nodos bajo el nodo internet:

- ✓ directory(1): reservado para uso futuro con el OSI directory (X.500)

- ✓ mgmt(2): usado para objetos definidos en documentos aprobados por el IAB.
- ✓ experimental(3): usado para identificar objetos utilizados en experimentos de Internet.
- ✓ private(4): usados para identificar objetos definidos unilateralmente por los fabricantes.

El subárbol mgmt contiene las definiciones de las MIBs que han sido aprobadas por el IAB. Actualmente, dos versiones de la MIB han sido desarrolladas, mib-1 y mib-2. La segunda es una extensión de la primera. Ambas fueron provistas con el mismo OID en el subárbol de modo que sólo puede haber una de las MIBs presentes en cualquier configuración.

El subárbol *private* tiene actualmente definido sólo un nodo hijo, el nodo *enterprises(1)*. Esta porción del subárbol se usa para permitirle a los fabricantes mejorar la gestión de sus dispositivos y compartir esta información con otros usuarios y fabricantes quienes pueden necesitar interoperar con sus sistemas. Se asigna una rama dentro del subárbol *enterprises* a cada fabricante que se registre por un OID *enterprises.*

1.3.5.2 MIB-I y MIB-II[19]

La agrupación de objetos es totalmente conveniente por dos razones. La primera es que la agrupación lógica facilita el uso de identificadores de objetos en la estructura de un árbol. Segundo, gracias a la agrupación se establece la implementación directa de objetos, porque al se implementado un grupo por naturaleza se tratará con cada uno de lo objetos componentes del grupo. Así, de esa manera, tanto el software altamente sofisticado, como el usuario final se entienden cabalmente para facilidades de soporte, pues el estado establecido para un grupo conlleva al mismo estado para todos sus objetos miembros[20].

---

[19] *LESCHENNE, Sebastián, Redes de Datos, SNMP – RMON, Universidad Nacional de Rosario, 2002 & McCLOGHRIE K, Rose, Management Information Base for Network Management of TCP/IP, RFC1156*

[20] *MILLER Mark; Managing Internetworks with SNMP; USA 1993.*

MIB-II está definida en la RFC1213 y es una segunda versión de la base de información de gestión (MIB-I) definida en la RFC 1156. Al diseñar esta nueva MIB se tuvieron en cuenta los siguientes criterios para incluir un objeto:

- Un objeto debe ser esencial para la gestión de fallos o configuración.
- Sólo se permiten objetos débiles, con esto se quiere decir que pueden provocar un daño limitado. Este recaudo fue tomado debido a la escasa seguridad del protocolo SNMP.
- Evidencia de que esté en uso y que sea necesario.
- Se elimina el límite de 100 objetos propuesto para MIB-I.
- Para evitar variables redundantes, ningún objeto debe poder derivarse de otros objetos incluidos.

Se definieron 10 grupos, cada uno con funciones específicas, en la MIB-II:

- *system*
- *interfaces*
- *at*
- *ip*
- *icmp*
- *tcp*
- *udp*
- *egp*
- *dot3 (transmission)*
- *snmp*

1.3.5.2.1 Grupo system

Este grupo provee información general sobre el sistema gestionado. Está formado por 7 objetos escalares como se pesra en la figura 1.7. Con estos objetos se puede obtener la siguiente información:

- Descripción de la entidad (hardware y software).
- Identificación del subsistema de gestión de red contenido en la entidad.
- Tiempo desde que fue reinicializado el dispositivo
- Identificación e información para el contacto de la persona a cargo del nodo de gestión.
- Nombre administrativo del nodo de gestión.

- Locación física del nodo de gestión.
- Valor (número entero de 0 a 127) que indica el conjunto de servicios que ofrece la entidad.

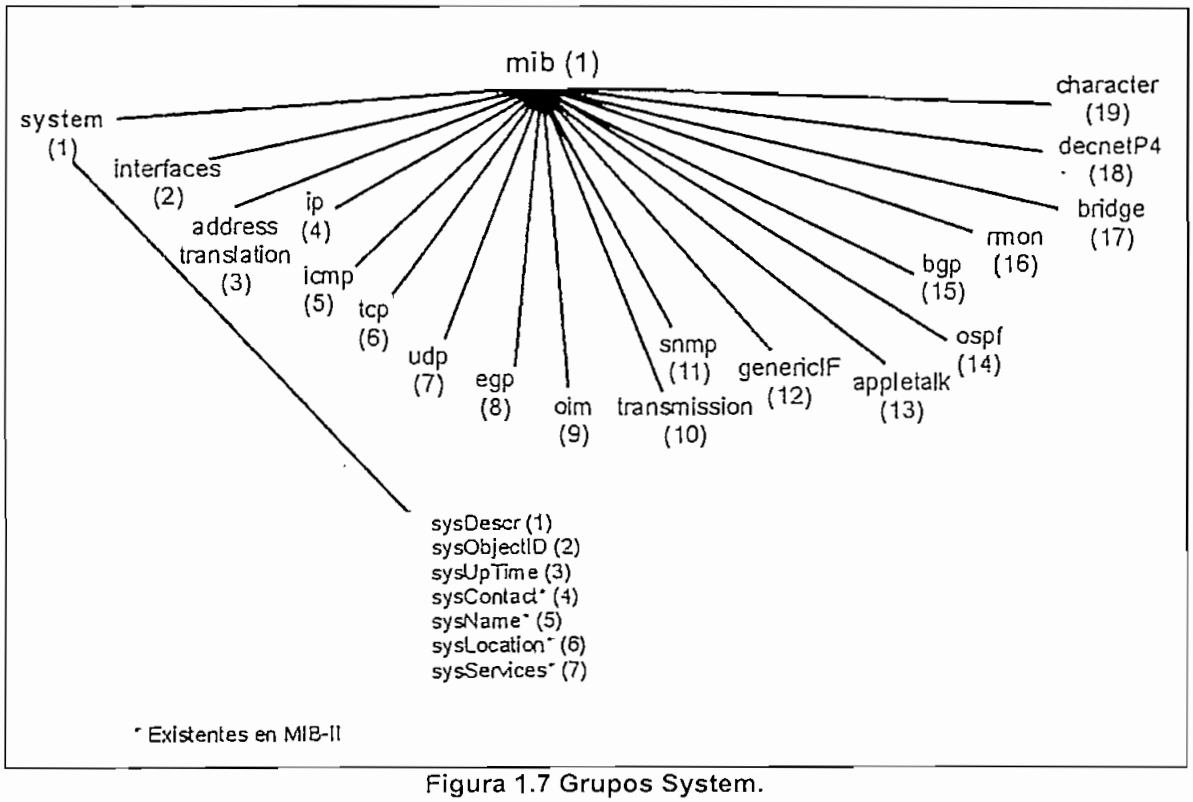


Figura 1.7 Grupos System.

1.3.5.2.2 Grupo interfaces

Este grupo contiene información sobre las interfases físicas de la entidad, incluyendo información de configuración y estadísticas sobre los eventos que ocurren en cada interfaz. Cada interfaz es pensada como si estuviera conectada a una subred, aunque se puede hallar con una conexión punto a punto. La implementación de este grupo es obligatoria. El grupo incluye un objeto escalar que representa el número de interfases asociadas a la entidad y una tabla (*ifTable*) que contiene una fila por cada interfaz. Algunos de los objetos almacenados en la tabla representan:

- Descripción de la interfaz.
- Tipo de interfaz (frame-relay, fddi, ppp, etc.).
- Máxima longitud en octetos de la PDU.
- Valor estimado de la tasa de transferencia.
- Estado de la interfase (*up*(1), *down*(2), *testing*(3)).

- Dirección física.
- Errores en paquetes.
- Cantidad de octetos entrantes y salientes.

Este grupo contiene información básica que es útil como punto de inicio para una función de gestión de red, como ser el monitoreo de *performance* o control de fallos. Por ejemplo, podría usarse para detectar una congestión en la red observando el número total de octetos entrantes y salientes del sistema. Una vez detectada la congestión pueden tomarse medidas para detectar quién o quienes están produciendo la congestión y si es posible solucionar el problema. La RFC1573 ha actualizado este grupo para obtener una funcionalidad mejor utilizando la estructura de datos SMIv2, como se indica en la figura 1.8.

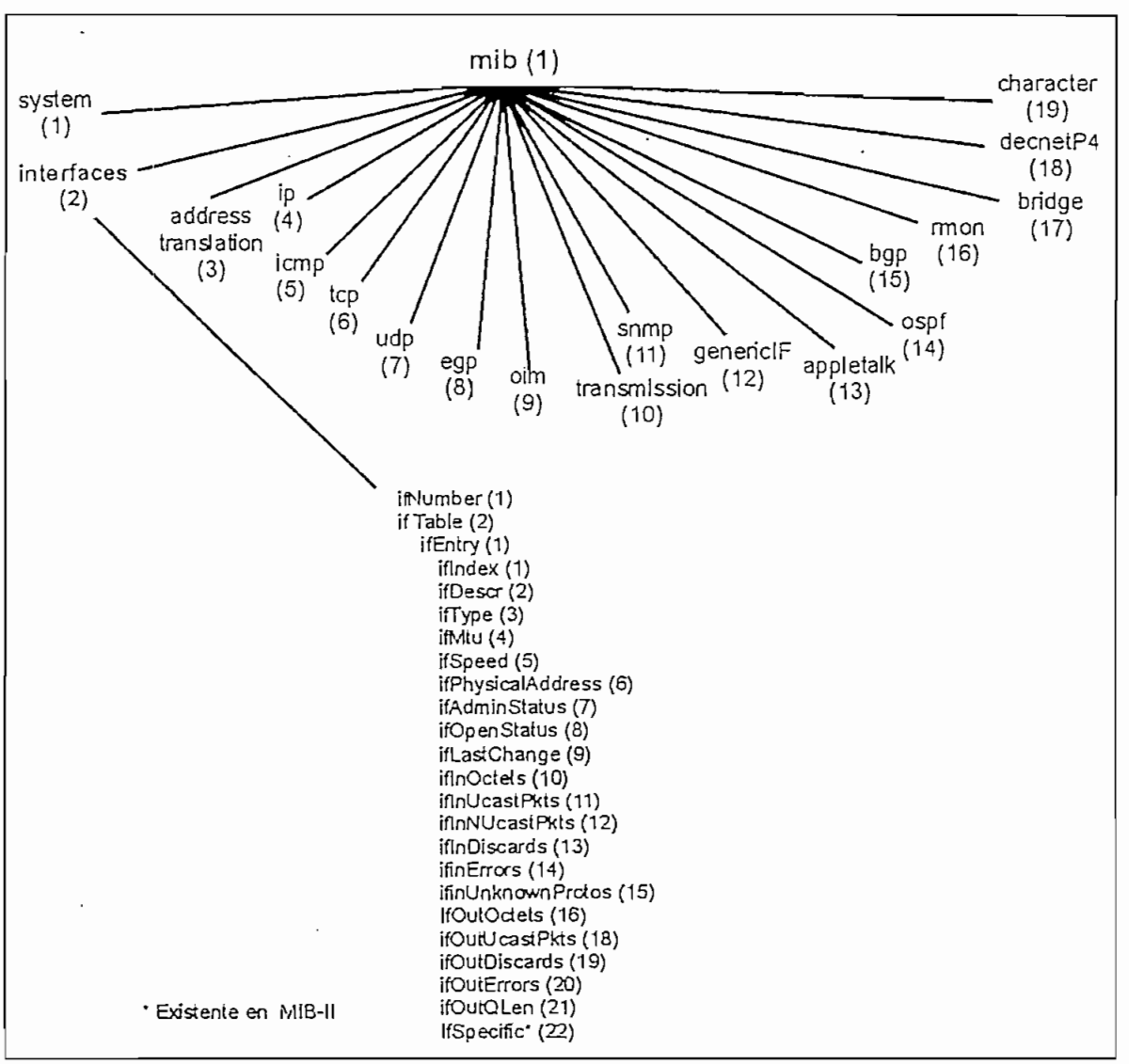


Figura 1.8 Grupos Interfaces.

1.3.5.2.3 Grupo at

El grupo *address translation (at)* provee el mapeo de las direcciones de red a direcciones físicas. La dirección de red típica es IP y la dirección física es la dirección MAC. En MIB-II este grupo está definido con el estado *deprecated*, y se incluye con el único propósito de compatibilidad con los nodos MIB-I. En MIB-II la información sobre mapeo de direcciones se halla en cada grupo de protocolo de red. Las razones para este cambio son:

- La necesidad de soportar nodos multiprotocolos: Cuando un nodo soporta más de un protocolo de red, se tiene más de una dirección de red asociada a cada dirección física.
- La necesidad de mapear la dirección en ambos sentidos: algunos protocolos, como el protocolo de ruteo (ES-IS), que requiere mapeo de la dirección física a la dirección de red.

Este grupo está formado por una única tabla *atTable* constituida por tres objetos columna: uno que indica la interfaz a la que hace referencia la fila, dirección física y dirección de red, como se muestra en la figura 1.9.

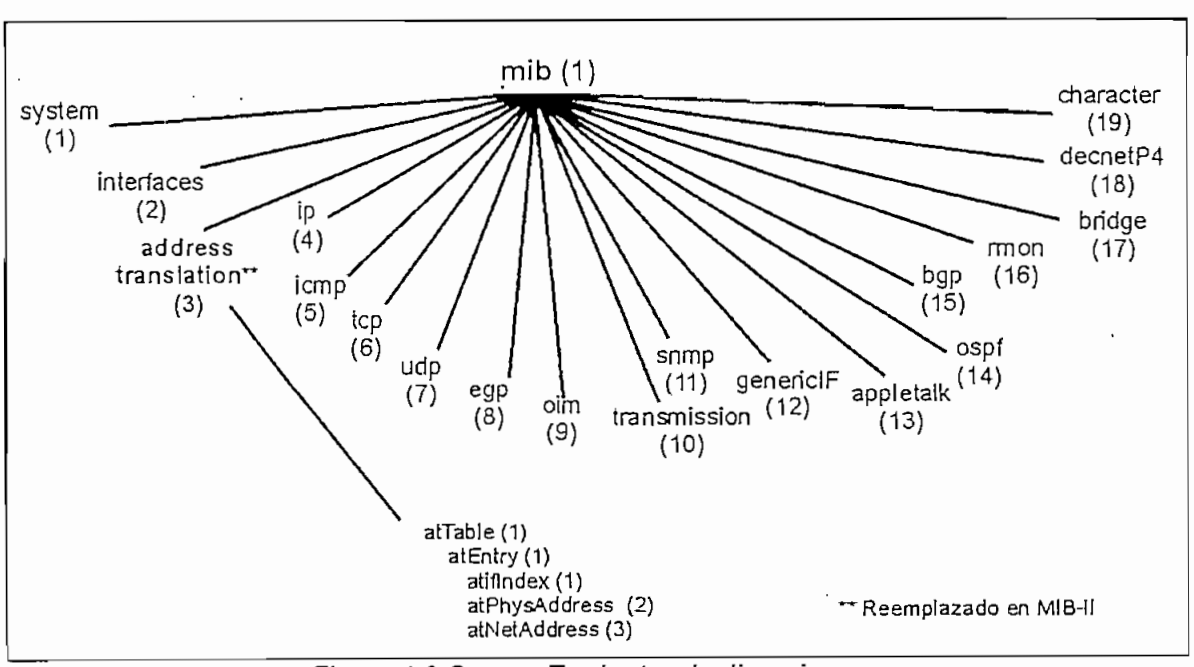


Figura 1.9 Grupos Traductor de direcciones.

1.3.5.2.4 Grupo ip

El grupo *ip* contiene información relevante sobre la operación e implementación del protocolo IP en un nodo. Una vez que el protocolo IP ha sido instalado en los dos extremos del sistema (hosts) y en los sistemas intermedios (ruteadores), no todos los objetos de este grupo son relevantes. Los objetos que no son relevantes tienen valor nulo.

El grupo está conformado por 21 objetos escalares útiles para el monitoreo de performance y control de fallos, y por tres tablas: *ipAddrTable*, *ipRouteTable* e *ipNetToMediaTable*.

La tabla *ipAddrTable* contiene información relevante sobre las direcciones IP asignadas a la entidad, con una fila por cada dirección IP. Se debe notar que con este grupo no pueden modificarse las direcciones IP asignadas debido a que los objetos son del tipo *read-only*.

La tabla *ipRouteTable* contiene información usada para el ruteo de Internet. Hay una fila por cada ruta conocida por la entidad. Esta tabla es útil para la configuración de monitoreo, y como los objetos son del tipo *read-write*, puede ser usada para controlar el proceso de ruteo.

La tabla *ipNetToMediaTable* permite la traducción de direcciones entre la dirección física y la dirección IP. Hay una fila por cada interfaz que no usa una técnica de mapeo con algoritmos. La información que contiene es la misma que la presente en el grupo *at*, con el agregado del objeto *ipNetToMediaType* que indica el tipo de mapeo utilizado: *other*(1), *invalid*(2), *dynamic*(3) o *static*(4).

Para solucionar los problemas que tenía la tabla *ipRouteTable* y para hacer más flexible la tabla de ruteo se eliminó la tabla y fue reemplazada con nuevas definiciones en la RFC1354.

La figura 1.10 muestra la estructura que tiene el Grupo IP.

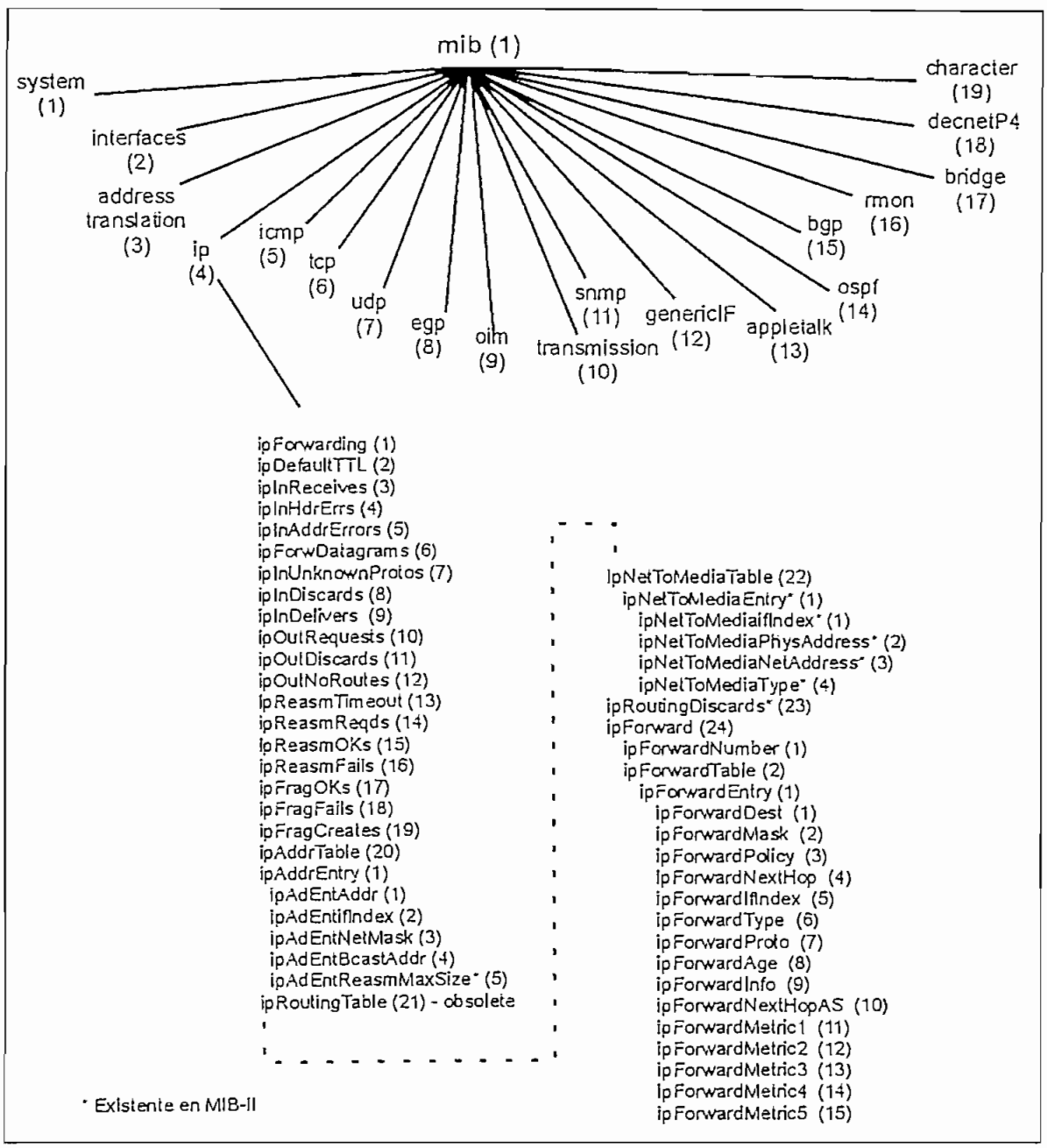


Figura 1.10 Grupos IP.

El problema específico de la tabla *ipRouteTable* era que sólo estaba indexada con el objeto *ipRouteDest*, que es la dirección destino IP de la ruta, y pueden aparecer varias rutas con el mismo destino, entonces el acceso a las filas de la tabla debía ser especificado por el protocolo en uso. Como SNMP no posee mecanismos especiales para el acceso a las tablas, sólo es posible definir una única ruta por cada dirección de destino, aunque el protocolo de ruteo permita el uso de rutas alternativas para el balanceo de cargas, confiabilidad u otras razones.

Para esto se definió un subgrupo dentro de *ip* llamado *ipForward* que consta de un objeto escalar y de la tabla *ipForwardTable*. El objeto escalar almacena el número de filas de la tabla.

La tabla *ipForwardTable* es la tabla que reemplaza a *ipRouteTable* , siendo sus objetos columna similares. La diferencia radica en que la nueva tabla está indexada con la dirección IP de destino (*ipForwardDest*), el mecanismo de ruteo con el que fue aprendida la ruta (*ipForwardProto*), la política para seleccionar entre rutas alternativas a un destino (*ipForwardPolicy*) y por el número de sistema autónomo del siguiente salto (*ipForwardNextHopAS*).

1.3.5.2.5 Grupo icmp

El protocolo ICMP definido en la RFC792, es una parte integral de la suite de protocolos TCP/IP. Es requerida su compañía junto a IP. Por lo tanto, todos los sistemas que implementen IP deben implementar ICMP. Esencialmente, ICMP provee información a cerca de los problemas en el entorno de comunicaciones. El grupo *icmp* contiene información relevante sobre la implementación y la operación de ICMP en un nodo. El grupo está formado por 26 contadores de mensajes ICMP enviados y recibidos, como se muestra en la figura 1.11.

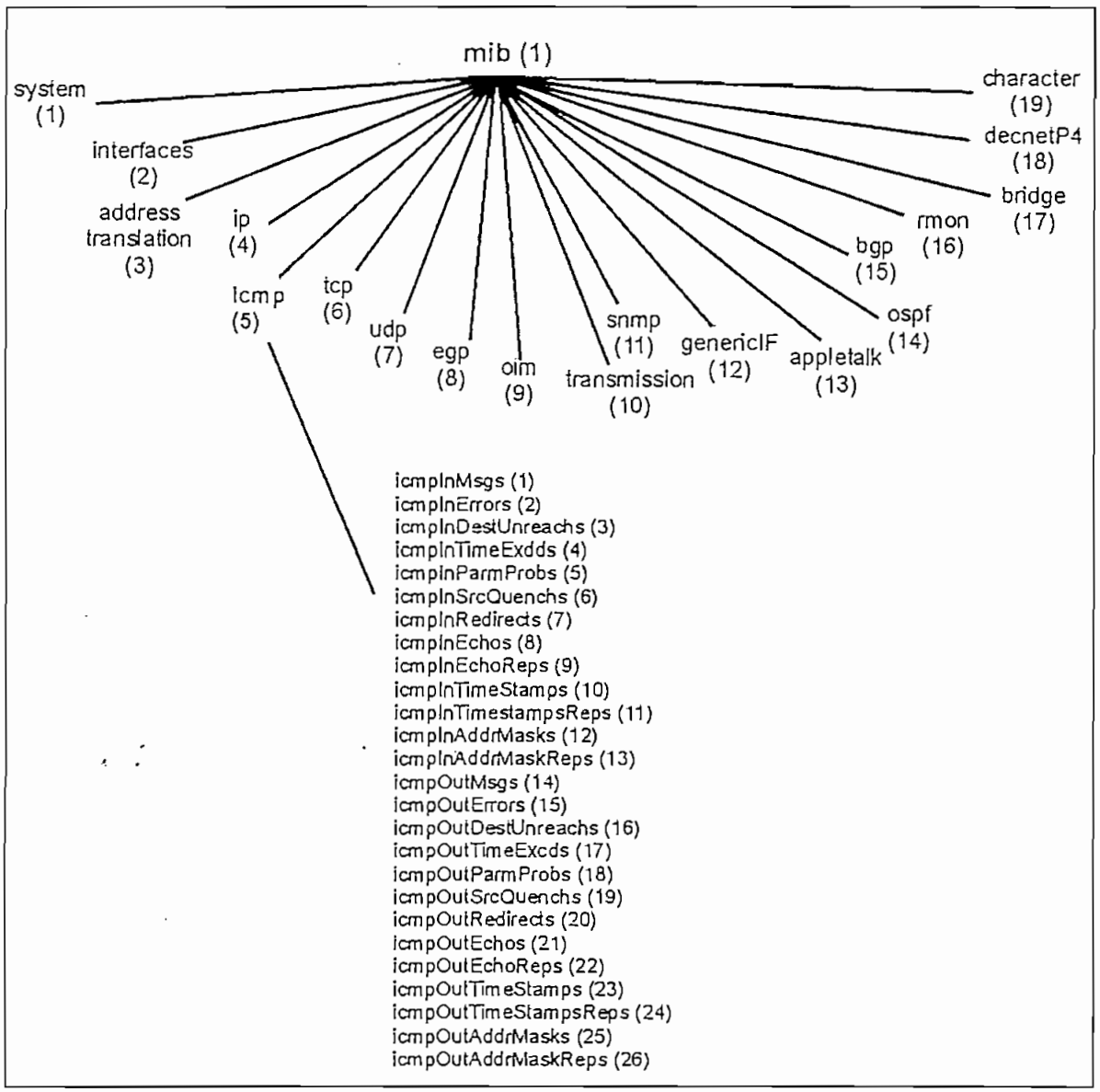


Figura 1.11 Grupos ICMP.

1.3.5.2.6 Grupo tcp

El grupo *tcp* contiene información relevante sobre la operación e implementación del protocolo TCP en un nodo. La única tabla que pertenece a este grupo es *tcpConnTable*. Además, el grupo esta conformado por 14 objetos escalares que contienen información sobre los segmentos recibidos, enviados, segmentos erróneos recibidos, segmentos retransmitidos, número máximo de conexiones TCP que soporta la entidad, etc., como se muestra en la figura 1.12.

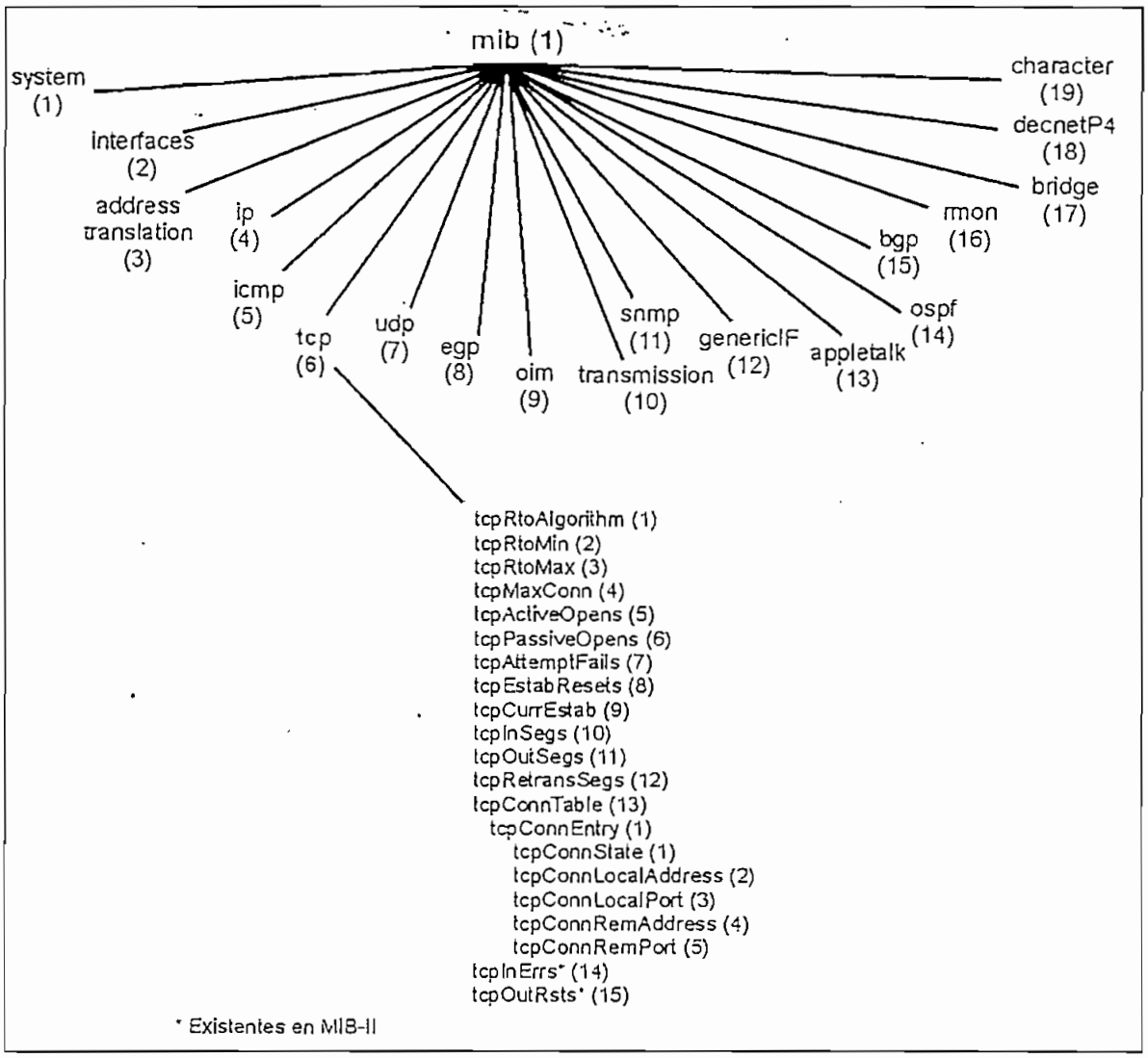


Figura 1.12 Grupos TCP

1.3.5.2.7 Grupo udp

El grupo *udp* contiene información relevante sobre la operación e implementación del protocolo UDP en un nodo. Este grupo posee cuatro objetos escalares que representan información sobre los datagramas enviados y recibidos y una tabla *udpTable*. Esta tabla almacena información sobre los extremos UDP de la entidad sobre la cual una aplicación local está recibiendo datagramas. Por cada usuario UDP, la tabla contiene la dirección IP y el puerto UDP para el usuario.

En la figura 1.13 se muestra la estructura de éste grupo.

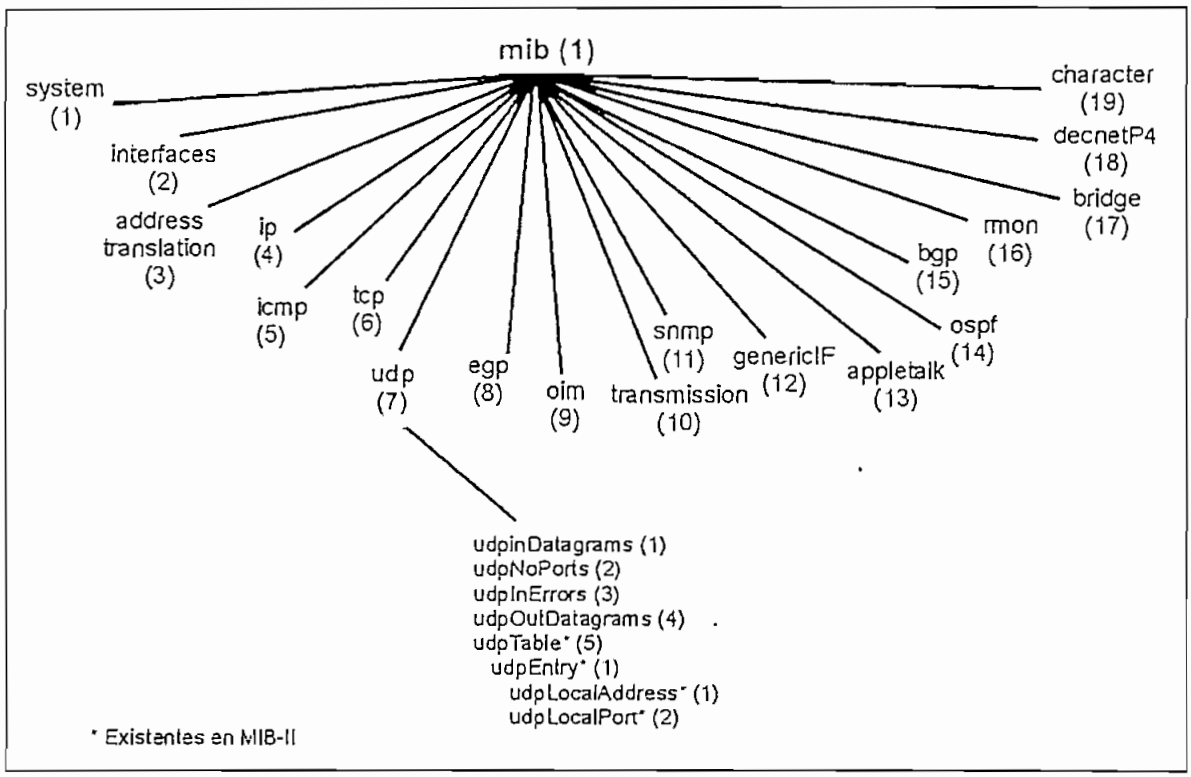


Figura 1.13 Grupos UDP

1.3.5.2.8 Grupo egp

El grupo *egp* contiene información relevante sobre la operación e implementación del protocolo External Gateway Protocol EGP en un nodo. Este grupo está formado por 5 objetos escalares, como se muestra en la figura 1.14, y que almacenan información sobre los mensajes EGP enviados y recibidos, y una tabla (*egpNeighTable*) que guarda información sobre cada uno de los gateways vecinos conocidos por la entidad.

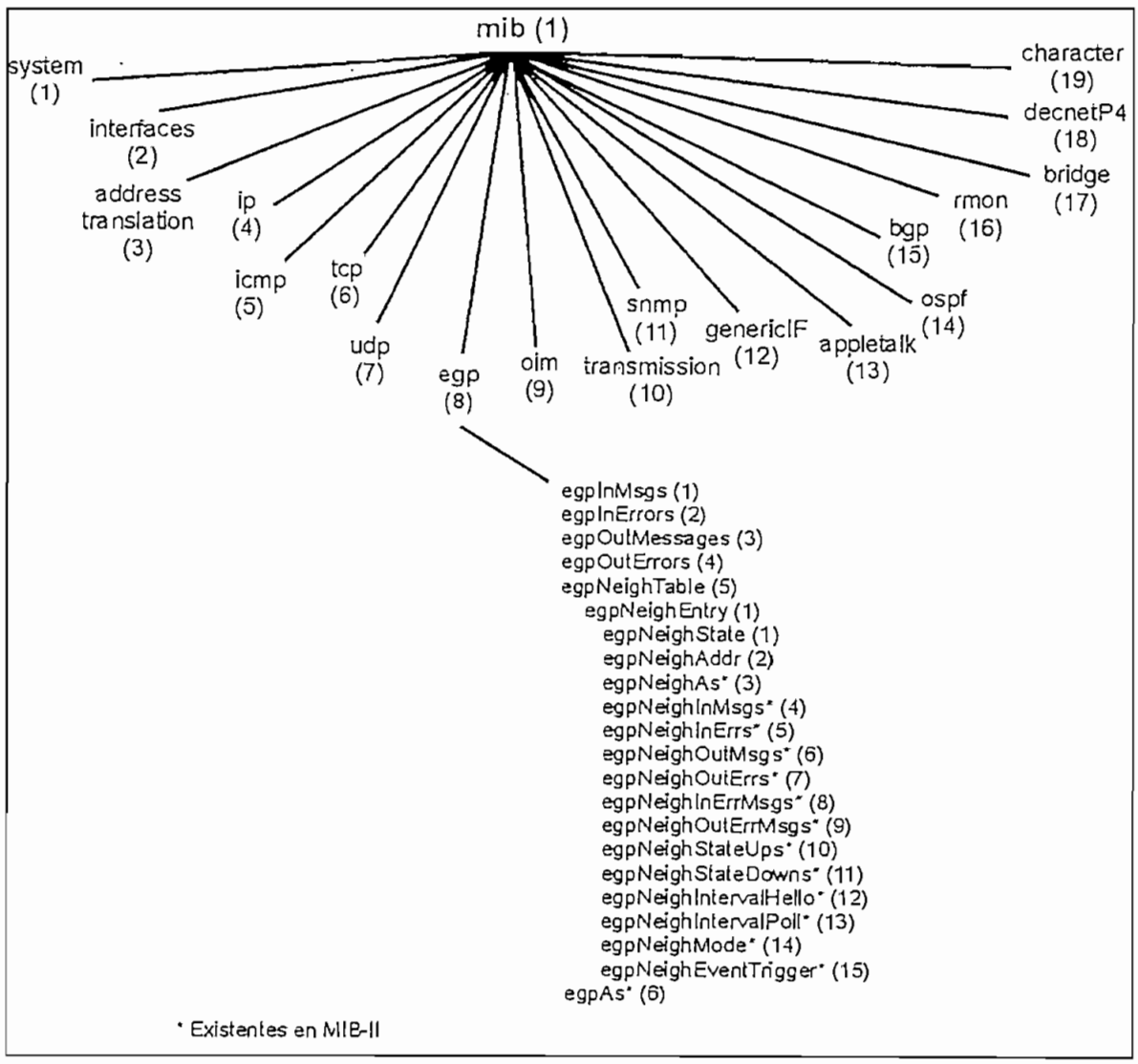


Figura 1.14 Grupos EGP

1.3.5.2.9 Grupo dot3 (transmission)

Este grupo está diseñado para contener objetos que provean detalles sobre el medio de transmisión subyacente para cada interfase del sistema. De hecho, este no es un grupo, sino simplemente un nodo en la jerarquía MIB-II debajo del cual se encuentran varios grupos de interfases específicas. A modo de ejemplo, se puede nombrar la MIB EtherLike que se encuentra definida en la RFC1643 y que es un subgrupo del nodo *transmission* de MIB-II.

La figura 1.15 contiene parte de la estructura mencionada.

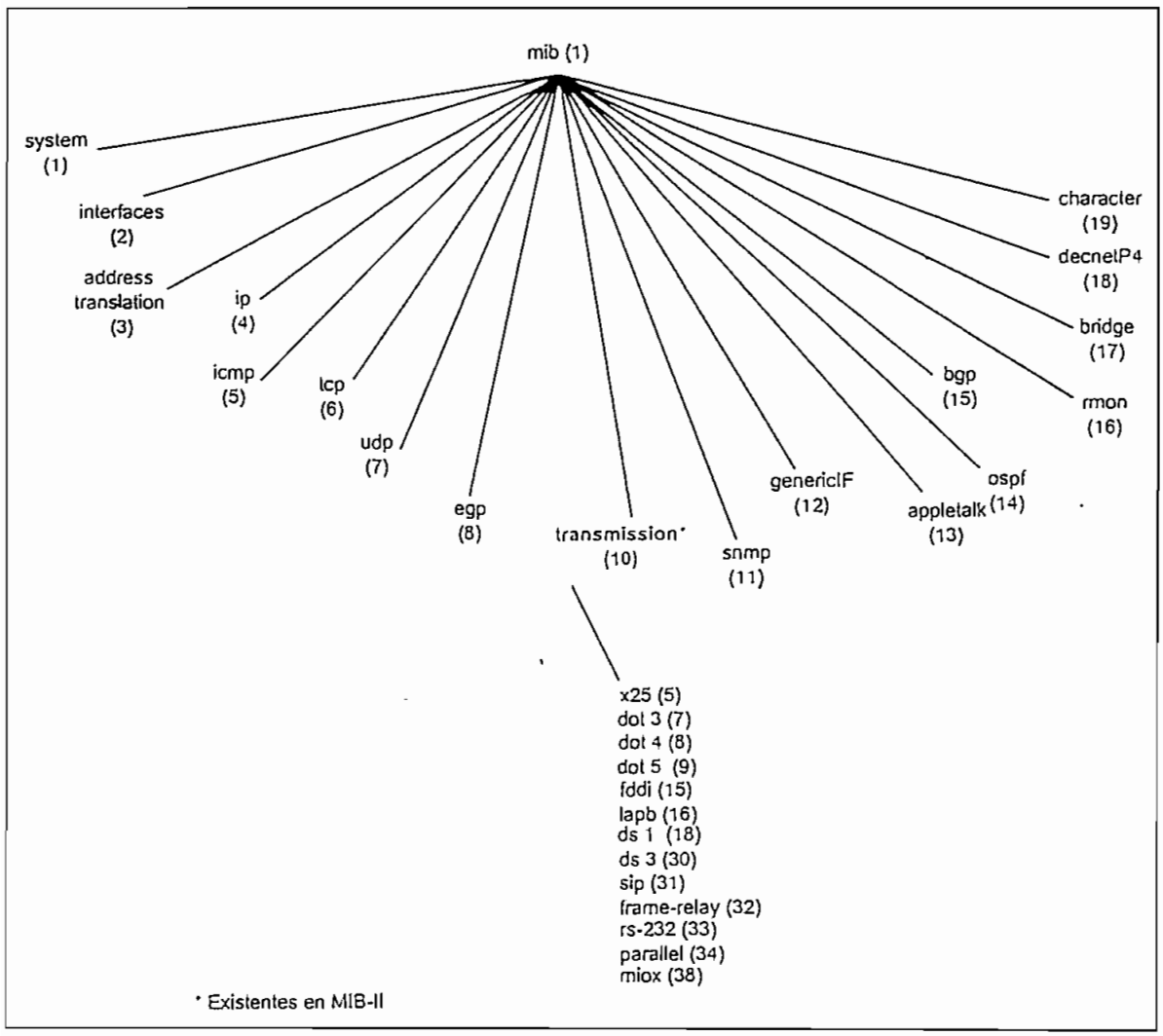


Figura 1.15 Grupos Transmission

1.3.5.2.10 Grupo snmp

El grupo *snmp* está formado por 30 objetos escalares que almacenan información relevante a cerca de la implementación y operación del protocolo SNMP. Algunos ejemplos sobre la información recolectada son los siguientes:

- Cantidad de comandos GetRequest, GetNextRequest y SetRequest enviados.
- Cantidad de comandos GetResponse recibidos.
- Cantidad de traps recibidas.
- Cantidad de paquetes recibidos con diversos errores (*tooBig, BadCommunityName,* etc.).

Los objetos que pertenecen al grupo SNMP se muestran en la figura 1.16 a continuación:

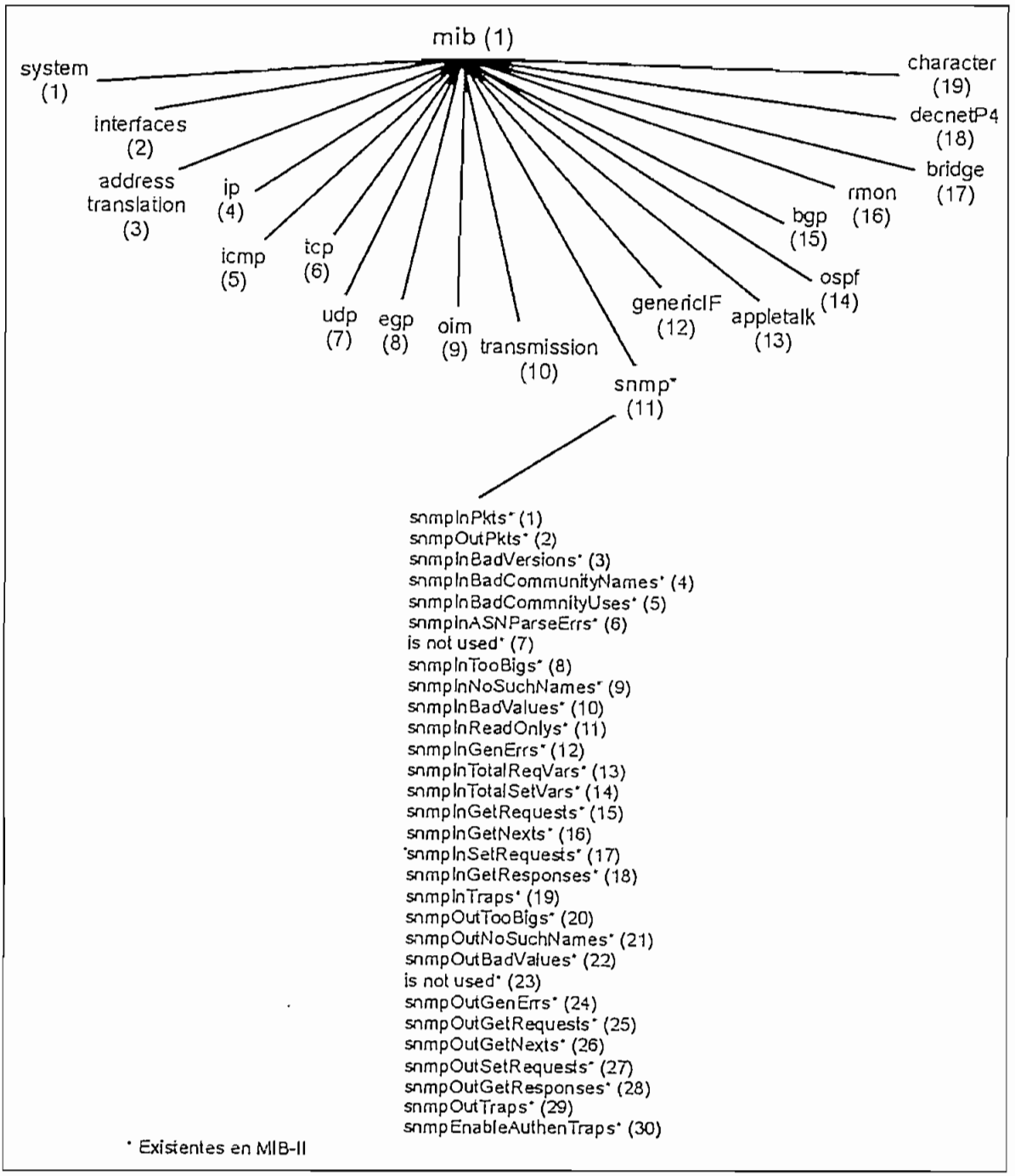


Figura 1.16 Grupos SNMP

1.3.5.3 MIB SNMPv2[21]

La MIB SNMPv2 define objetos que describen el comportamiento de una entidad SNMPv2. Esta MIB consiste de tres grupos:

- Grupo *System*: este es una expansión del grupo *system* original presente en MIB-II. Permite la descripción de objetos que pueden ser configurables dinámicamente.
- Grupo *SNMP*: Modifica el grupo *snmp* de MIB-II. Provee instrumentos básicos para gestionar la actividad del protocolo.
- Grupo *MIB objects*: Es una colección de objetos que gestionan la PDU SNMPv2-Trap y que permite cooperar entre entidades SNMPv2, todas actuando en el rol de gestor, para coordinar el uso de la operación Set.

1.3.5.3.1 Grupo system

Este grupo es el mismo grupo definido en MIB-II con el agregado de un objeto escalar (*sysORLastChange*) y una tabla (*sysORTable*). Cada fila de la tabla representa un objeto (*resource object*) que puede ser configurado dinámicamente. El objeto escalar representa el valor de *sysUpTime* del momento más reciente en que fue modificado algún valor de una instancia de *sysOID*. La tabla está formada por cuatro valores: un índice que identifica unívocamente cada fila, un identificador de objeto para la fila (análogo a *sysObjectID* en MIB-II), una descripción del objeto y el valor de *sysUpTime* en que fue instanciada la fila por última vez.

1.3.5.3.2 Grupo SNMP

Este grupo eliminó la mayoría de los objetos definidos en el grupo *snmp* de MIB-II y agregó dos nuevos objetos escalares. Los nuevos objetos incluidos son los siguientes:

---

[21] *LESCHENNE, Sebastián, Redes de Datos, SNMP – RMON, Universidad Nacional de Rosario, 2002.*

- *snmpSilentDrops*: Cantidad total de PDUs GetRequest, GetNextRequest, GetBulkRequest, SetRequest e InformRequest que fueron eliminadas porque el tamaño de la respuesta que contenía una PDU de respuesta alternativa con el campo variable-bindings vacío era mayor que una restricción local o que el tamaño máximo de mensaje permitido por la parte que envió el mensaje *request* .

- *snmpProxyDrops*: Cantidad total de PDUs GetRequest, GetNextRequest, GetBulkRequest, SetRequest e InformRequest que fueron eliminadas porque el contexto indicaba un proxy SNMPv2 y la transmisión del mensaje al proxy destino falló de forma que la PDU de respuesta no puede ser enviada a la parte que envió el mensaje *request* original.

La decisión de eliminar la mayoría de los objetos definidos en MIB-II (sólo quedan 6 de los 30 objetos originales), se debe a que las estadísticas del protocolo muy detalladas no son esenciales para depurar problemas reales y requieren demasiados recursos del agente.

1.3.5.3.3 Grupo MIB objects

Este grupo contiene información para el control de objetos MIB. Este grupo está formado por dos subgrupos *snmpTrap* y *snmpSet*. El primero está formado por dos objetos relacionados con las Traps:

- *snmpTrapOID*: es un identificador de las traps y notificaciones enviadas. El valor de este objeto siempre se envía como segundo objeto en la PDU SNMPv2-Trap y en la PDU InformRequest.
- *snmpTrapEnterprise*: es un identificador de la empresa asociada con el trap enviado.

El segundo subgrupo consiste de un único objeto (*snmpSerialNo*), que es usado para resolver los siguientes problemas:

- Se pueden ocasionar múltiples operaciones Set sobre un mismo objeto por parte de la estación de gestión, y es esencial que las operaciones se realicen en el orden en que fueron enviadas sin importar que a causa de la transmisión se hubiera cambiado el orden.
- El uso de operaciones Set sobre un mismo objeto por parte de diferentes gestores puede ocasionar una base de datos inconsistente o inexacta.

1.3.5.3.4 Extensión del grupo interfaces

El grupo *interfaces* de MIB-II es mejorado a través de una extensión especificada en la RFC1573 uilizando la definición SMIv2. La RFC1573 modifica mínimamente el grupo *interfaces* y agrega cuatro nuevas tablas:

- Extensión de la tabla *ifTable* (*ifXTable*)
- Tabla de pila (*ifStackTable*)
- Tabla de prueba(*ifTestTable*)
- Tabla de dirección de recepción (*ifRcvAddressTable*)

El grupo interfaces en MIB-II consta de un objeto escalar (ifNumber) y de una tabla (ifTable). La RFC1573 especifica una nueva descripción al objeto columna ifIndex:

"Un valor único, mayor que cero, para cada interfaz o subcapa de interfaz en el sistema gestionado. Se recomienda que los valores sean asignados de manera continua a partir de 1. El valor de cada subcapa "interfaz" se debe mantener constante por lo menos desde una re-inicialización de la entidad del sistema de gestión de red hasta la próxima re-inicialización."[22]

No es obligatorio que los valores de ifIndex sean números consecutivos a partir del 1 para permitir un agregado/eliminado de filas dinámico.

---

[22] *RFC 1573*

La RFC1573 permite la existencia de múltiples filas en la tabla para una única interfase física, siendo una fila por cada subcapa lógica. Sin embargo, este documento recomienda su uso con moderación. También, el documento recomienda que no sea creada una fila por cada circuito lógico. Por último, se han eliminado algunos de los objetos columna pertenecientes a ifTable debido a que no eran implementados frecuentemente, o se han definido nuevos objetos en las cuatro tablas especificadas en la RFC1573.

1.3.5.3.5 Extensión de la tabla ifTable (ifXTable)

La tabla *ifXTable* es una extensión de la tabla *ifTable* (está definida mediante la cláusula AUGMENTS). Esta tabla tiene los siguientes objetos columna:

- Contadores de paquetes broadcast y multicast entrantes y salientes de cada interfase.
- Contadores de paquetes y octetos de alta capacidad (64 bits).
- Un objeto que permite la habilitación/deshabilitación de envío de traps en caso de haberse caído o restituído una conexión.
- Un objeto que estima la tasa de transferencia en Mbps.
- Un objeto que indica si se evaluarán los paquetes en forma promiscua o si solamente se evaluarán los paquetes dirigidos a esa estación. Los paquetes broadcast y multicast no son considerados en esta definición.
- Un objeto que indica si hay un conector físico.

1.3.5.3.6 Tabla ifStackTable

Esta tabla identifica las relaciones entre múltiples filas de la tabla *ifTable,* que son soportadas por la misma interfaz física. Indica que subcapas corren sobre cada una de las otras subcapas.

Cada fila en la tabla identifica una relación entre dos filas de la tabla *ifTable*. Los objetos de la tabla son los siguientes:

- *ifStackHigherLayer*: valor de *ifIndex* asociado a la capa superior de ésta relación.
- *ifStackLowerLayer*: valor de *ifIndex* asociado a la capa inferior de ésta relación.
- *ifStackStatus*: objeto del tipo *RowStatus*.

1.3.5.3.7 Tabla ifTestTable

Esta tabla define objetos que permiten al gestor instruir a un agente para probar una interfaz sobre varios tipos de fallos. La tabla contiene una fila por cada interfaz. Cada fila en la tabla posee tres capacidades.

- ✓ Permite al gestor especificar un test en particular para que sea realizado sobre la interfase, configurando el valor de *ifTestType*.
- ✓ Permite al gestor obtener los resultados del test realizado por el agente, leyendo los valores de los objetos *ifTestResult* e *ifTestCode*.
- ✓ Provee un mecanismo para asegurar que sólo un gestor a la vez pueda invocar satisfactoriamente un test en particular, usando los objetos *ifTestId* e *ifTestStatus*.

1.3.5.3.8 Tabla ifRcvAddressTable

Esta tabla contiene una fila por cada dirección (broadcast, multicast o unicast) de la que recibe paquetes el sistema sobre una interfase en particular, excepto cuando está funcionando en modo promiscuo. La tabla está formada por tres objetos:

- ✓ *ifRcvAddressAddress*: una dirección broadcast, multicast o unicast que el sistema reconoce como dirección propia de destino para capturar paquetes.

- ✓ *ifRcvAddressStatus*: objeto del tipo RowStatus.
- ✓ *ifRcvAddressType*: indica si la dirección es del tipo: *other*(1), *volatile*(2) o *nonVolatile*(3). Una dirección *nonVolatile* continuará existiendo luego de un reinicio del sistema, mientras que una dirección *volatile* se perderá. Si una dirección es etiquetada como *other* indica que esta información no está disponible en la tabla.

1.3.6 ASN.1 (Abstract Syntax Notation One)[23]

Es un lenguaje formal diseñado para especificar la sintaxis de tipos de datos. Tipos de datos a ser intercambiados entre diferentes sistemas.
Algunos ejemplos:

- Telefonía (SS7, Otros)
- Banca, tarjetas de crédito
- Aviación
- Redes de Datos (SNMP, LDAP)

ASN.1 nos proporciona dos grandes grupos de funcionalidad :

- Un conjunto de reglas y una sintaxis para especificar tipos de datos.
- Un conjunto de reglas (BER) que especifican como codificar los tipos de datos anteriormente definidos para su transmisión a través de la red.

1.3.6.1 Sintaxis de Objetos

Cada objeto dentro de la SNMP MIB se define de un modo formal. La definición especifica el tipo de dato del objeto, sus estados permitidos y rangos de valores, y su relación con otros objetos dentro de la MIB.

---

[23] *INTERNATIONAL ORGANIZATION FOR STANDARIZATION; Information Technology: Open System Interconnection, Specification of Abstract Syntax Notation One; ISO 8824; USA , 1990.*

La notación ASN.1 se usa para definir cada objeto individual y también para definir la estructura completa de la MIB. Para mantener la simplicidad sólo se usa un subconjunto de elementos y características de ASN.1.

En la figura 1.17 se visualizan los tipos de datos permitidos definidos dentro del byte identificador.

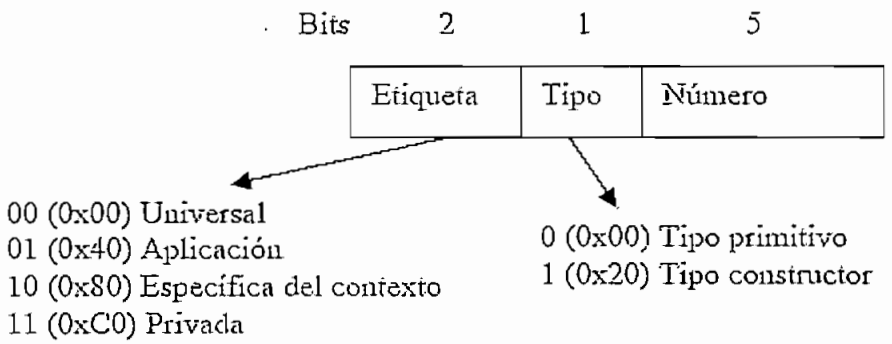


Figura 1.17 Primer byte de cada elemento enviado utilizando ASN.1

A continuación se describen los tipos de datos permitidos y se mencionan sus usos más comunes:

1.3.6.1.1 Tipos Universal.

La clase UNIVERSAL de ASN.1 está formada por tipos de datos independientes de la aplicación para uso general. Dentro de esta clase, sólo los siguientes tipos de datos están permitidos para definir objetos de MIB:

- ✓ integer (UNIVERSAL 2)
- ✓ octetstring (UNIVERSAL 4)
- ✓ null (UNIVERSAL 5)
- ✓ object identifier (UNIVERSAL 6)
- ✓ sequence, sequence-of (UNIVERSAL 16)

Como se muestra en la tabla 1.6, los cuatro primeros son tipos primitivos que sirven como bloques de construcción básicos de otros tipos de objetos. El último ítem en la lista menciona los tipos constructores sequence y sequence-of.

Tabla 1.6 Marcas de la Clase Universal.

Tipo primitiva ASN.1	*identificador en hex*
INTEGER	02
BIT STRING	03
OCTET STRING	04
NULL	05
OBJECT IDENTIFIER	06

Tipo Constructed ASN.1	*identificador en hex*
SEQUENCE	30

1.3.6.1.2 Tipos Aplicación-Extendida.

La clase APPLICATION de ASN.1 consta de tipos de datos que son relevantes para una aplicación particular. Cada aplicación, incluyendo SNMP, es autora de sus propios tipos de datos APPLICATION. La RFC 1155 lista un conjunto de tipos de datos *aplicación-extendida* para SNMP; otros tipos podrán definirse en futuras RFCs. Los siguientes tipos están definidos:

- ✓ ***networkaddress:*** permite la selección de un formato de dirección dentro de un número de familias de protocolos. Actualmente, la única dirección definida es IpAddress.

- ✓ ***ipaddress:*** es una dirección de 32 bits usando el formato especificado en IP.

- ✓ ***counter:*** es un entero no negativo que se puede incrementar pero no decrementar. Se especifica un valor máximo de $2^{32} - 1$; cuando el counter alcanza su máximo, comienza a incrementarse nuevamente desde cero.

- ✓ ***gauge:*** es un entero no negativo que puede incrementarse o decrementarse, con un valor máximo de $2^{32} - 1$. Si se alcanza el valor máximo, mantiene el mismo hasta ser reseteado.
- ✓ ***timeticks:*** es un entero no negativo que cuenta el tiempo en centésimas de segundos desde el momento referenciado en la definición de un objeto que utilice este tipo.
- ✓ ***opaque:*** este tipo soporta la capacidad de pasar datos arbitrarios. El dato es codificado como OCTET STRING para la transmisión. El dato en sí mismo puede estar en cualquier formato definido por ASN.1 u otra sintaxis.

El *counter* es uno de los tipos más comúnmente utilizados en la definición de objetos. Las aplicaciones típicas son contar el número de paquetes u octetos que fueron enviados o recibidos.
Como *counter* sigue contando desde cero una vez que alcanzó su máximo, el sistema de gestión debe ser capaz de distinguir si un valor x significa x o $(N * 2^{32}) + x$. La única manera que la estación de gestión tiene para poder distinguir el valor correcto es encuestar periódicamente el objeto para mantener el valor N actual. Como se usan contadores de 32 bits, esto no tiene que hacerse muy frecuentemente.

El *gauge* se usa para medir el valor actual de alguna entidad, como ser el valor actual de paquetes almacenados en una cola. Puede también usarse para guardar la diferencia en el valor de alguna entidad desde el comienzo hasta el final de un intervalo de tiempo. Esto habilita al *gauge* para ser usado como monitor de la tasa de cambio del valor de una entidad.

El tipo *timeticks* es un timer relativo. El tiempo se mide en relación a algún evento (como ser un arranque o una reinicialización) dentro del sistema gestionado. Los valores de los timers en un sistema no pueden ser comparados con valores de timers en otro sistema. Un tipo de timer absoluto es impracticable en SNMP debido a que al mayoría de los sistemas que corren la suite de protocolos TCP/IP no soportan un protocolo de sincronización de tiempo.

En la tabla 1.7 se muestra el identificador hexadecimal para cada Primitiva SNMP de tipo de aplicación.

Tabla 1.7 Marcas de la Clase Aplicación.

Tipos de aplicación Primitive SNMP	*identificador en hex*
IpAddress	40
Counter (Counter32 in SNMPv2)	41
Gauge (Gauge32 in SNMPv 2)	42
TimeTicks	43
Opaque	44
NsapAddress	45
Counter64 (available only in SNMPv2)	46
Uinteger32 (available only in SNMPv2)	47

1.3.6.1.3 Tipos Contexto – Específico

Las puntualizaciones de SNMP interpretan las marcas de tipo de Contexto – Específico y se resumen en la tabla 1.8 a continuación:

Tabla 1.8 Marcas de la Clase Contexto Específico SNMP.

Tipos de contexto específico dentro de un mensaje SNMP	*identificador en hex*
GetRequest-PDU	A0
GetNextRequestPUD	A1
GetResponse-PDU (Response-PDU in SNMPv 2)	A2
SetRequest-PDU	A3
Trap-PDU (obsolete in SNMPv 2)	A4
GetBulkRequest-PDU (added in SNMPv 2)	A5
InformRequest-PDU (added in SNMPv 2)	A6
SNMPv2-Trap-PDU (added in SNMPv 2)	A7

1.3.6.1.4 Tipos Privada

Los proveedores de equipos manejan éste tipo de marcas de clase, de acuerdo a su conveniencia.

1.3.7 CODIFICACIÓN[24]

1.3.7.1 Codificación mediante sintaxis de transferencia (BER)[25]

Los objetos en la MIB son codificados usando la especificación de codificación estandarizada *Basic Encoding Rules* (BER). Las reglas describen un método para codificar valores de cada tipo ASN.1 como una cadena de octetos. La codificación está basada en una estructura *type-length-value* (TLV), como se indica en la figura 1.18. Esto es, cualquier valor ASN.1 puede codificarse como una combinación de tres componentes de la siguiente manera:

- ***type***: indica el tipo ASN.1, como así también la clase del tipo, además indica si la codificación es *primitive* o *constructed*.
- ***length***: indica la longitud de la representación del valor.
- ***value***: representa el valor del tipo ASN.1 como una cadena de octetos.

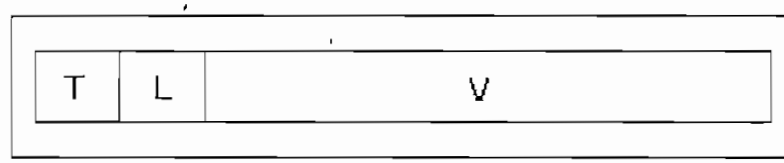


Figura 1.18 Estructura TLV simple.

Esta estructura es recursiva: para cualquier valor ASN.1 que consista de uno o más componentes, el componente *value* de la TLV se codifica a sí mismo como una o más estructuras TLV como lo muestra la figura 1.19.

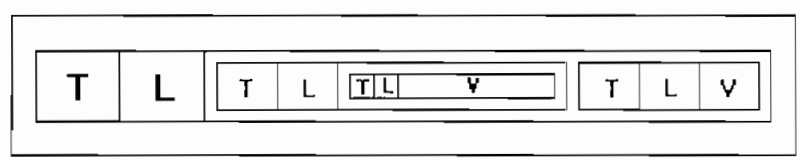


Figura 1.19 Estructura TLV estructurada.

[24] *ROSE Marshall; The Simple Book; Prentice Hall; USA 1994.*
[25] *MUKH Vijay, BER SNMP, Vijay Mukhi's Computer Institute, India, 2000.*

El método elegido depende del tipo ASN.1, del valor a codificar y del conocimiento o no de la longitud del valor basado en dicho tipo.

Usando los valores identificadores se puede fácilmente comunicarse con un agente SNMP y de igual manera interpretar de manera sencilla la contestación utilizando los identificadores antes expuestos.

1.3.7.2 Codificación Campo Tipo

El campo tipo proporciona la identificación, para la información codificada, de modo tal que "advierta" a los dispositivos para que reciban. Existen tres subcampos; como se indicó en la figura 1.7, en el campo tipo.

Los dos primeros bits (correspondientes al bit 8 y 7) especifican una clase de marca. Se usan los tipos de marcas definidos en las secciones: 1.3.6.1.1, 1.3.6.1.2 y 1.3.6.1.3 de éste trabajo. En la tabla 1.9 se indica los tipos de marcas y su numeración especificada par los bits 7 y 8.

Tabla 1.9 tipos de marcas y su numeración

Tipo	Bit 8	Bit 7
Universal	0	0
Aplicación	0	1
Contexto Específico	1	0
Privada	1	1

Las marcas del Tipo Universal se utilizan para los tipos de datos simples y estructurados.
Las del Tipo Aplicación se emplean con los tipos de datos definidos.
Las de Tipo contexto Específico se usan para las PDUs SNMP.
El bit sexto del campo tipo, puede ser de un formato *Primitive*(primitivo) si es igual a cero, o E*structured*(estructurado), si el bit contiene un uno. Cuando el valor del bit número 6 corresponde al de primitivo, significa que la información de administración se codifica directamente; si dicho bit es estructurado se entiende que se deben emplear uno, o más valores para la codificación de los datos, este es el caso de *sequense* y las PDUs SNMP.

Los otros 5 bits, representan el número de la marca, cuando el número en mención está entre 0 y 30 (como ocurre en el caso de SNMP).

Cuando el número de marca es mayor o igual que 31, los restantes 5 bits de éste octeto se llenan con uno y se usan octetos adicionales, en os cuales el octavo bit(el más significativo) es puesto en uno. El último de éstos octetos, tiene el bit 8 en cero. Para saber la numeración de la marca, se deben sumar los siete bits restantes de cada octeto adicional como se muestra en la figura 1.20.

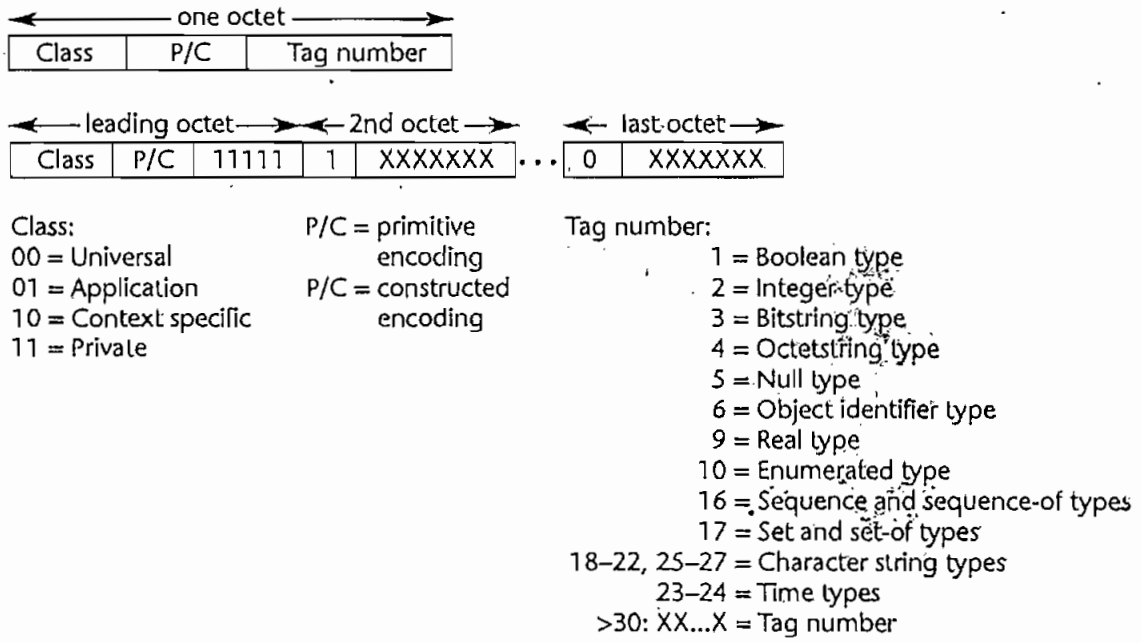


Figura 1.20 Subcampos del campo tipo.

1.3.7.3 Codificación Campo Longitud[26]

En el campo de longitud se determina el número de octetos que conformarán los datos del campo de valores. En este campo, se dan dos tipos de estructuras bien definidas, una "corta", y otra llamada "larga". En el primer caso, solo se tiene un octeto, cuyo octavo bit, es cero, por lo que los 7 bits siguientes podrían realizar un conteo de 0 a 127 en decimal.

[26] *MILLER Mark, Managing Internetworkswith SNMP, M&T Books, USA 1993.*

En la estructura larga, se distingue el bit más significativo(bit octavo) como un 1, los bits restantes establecen el número de octetos del campo longitud que irán a continuación de éste primero, se pueden tener de 1 a 126 octetos (el127 se reserva para uso futuro). De esta manera, se logra tener una cuenta de hasta $2^{1008}-1$ octetos posibles (el número 1008, es el resultado de la multiplicación, de los 126 octetos por 8 bits que lleva cada octeto). La figura 1.21 muestra de mejor manera ésta codificación.

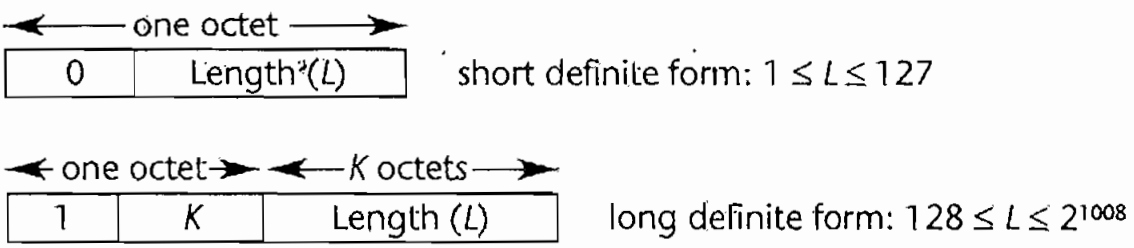


Figura 1.21 Campo longitud

1.3.7.4 Campo de Valores

El campo de valores, incluye la información que debe ser codificada, en otras palabras, en este campo, se encuentran los tipos definidos abstractamente e las PDUs SNMP. Todo este conjunto de definiciones totalmente relacionadas, para un correcto funcionamiento, conduce a una manera de representar la información en forma plenamente clara. A continuación, se proporcionan ejemplos de las reglas BER(*Basic Encoding Rules*), con los diversos tipos de datos de la SMI.

1.3.7.5 Codificación de Tipos de Datos Simples y Estructurados[27]

1.3.7.5.1 Codificación de Enteros

Los enteros se codifican con un campo tipo equivalente a 02H, el campo de longitud depende de la cantidad de datos. Finalmente, el campo de datos mismo, contiene la información en complemento de dos.

---

[27] *MILLER Mark, Managing Internet works with SNMP, M&T Books, USA 1993.*

En la figura 1.22 se indica la codificación del número 49:

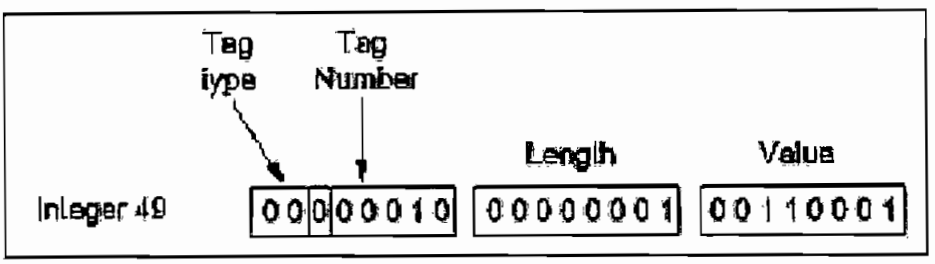


Figura1.22 codificación de enteros

1.3.7.5.2 Codificación de los enteros Bit String

Los BIT STRING, se codifican con un campo tipo igual a 03H, el campo de longitud provee el número de octetos del campo de datos, pero en este último, el primer octeto se utiliza como referencia del número de bits que no se usarán en los octetos siguientes (representación en forma de cruces).
En la figura 1.23 se indica un ejemplo de codificación del BIT STRING 110.

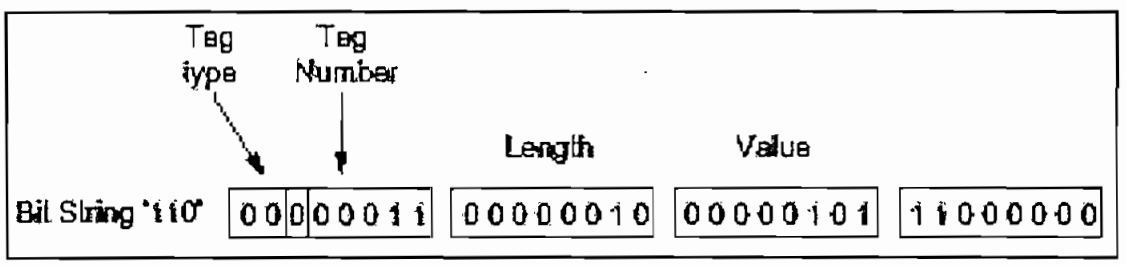


Figura1.23 codificación de BIT STRING

1.3.7.5.3 Codificación de Cadenas de Octetos

El tipo de datos OCTECT STRING se codifica de manera similar a la de INTEGER, pero con un valor de 04H, en el campo tipo.

El ejemplo que se indica en la figura 1.24, presenta la codificación de las letras *xy*.

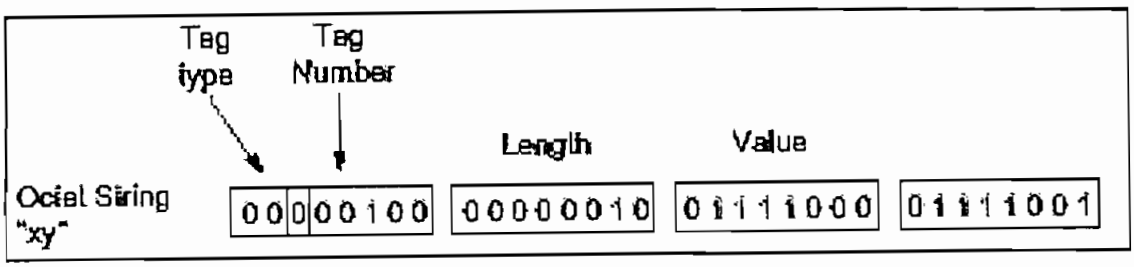


Figura 1.24 Codificación de cadenas de octetos

1.3.7.5.4 Codificación de NULL

El NULL, en su campo tipo tiene un valor igual a 05H, su campo de longitud es nulo y carece por tanto, de campo de datos propiamente dicho(figura 1.25)

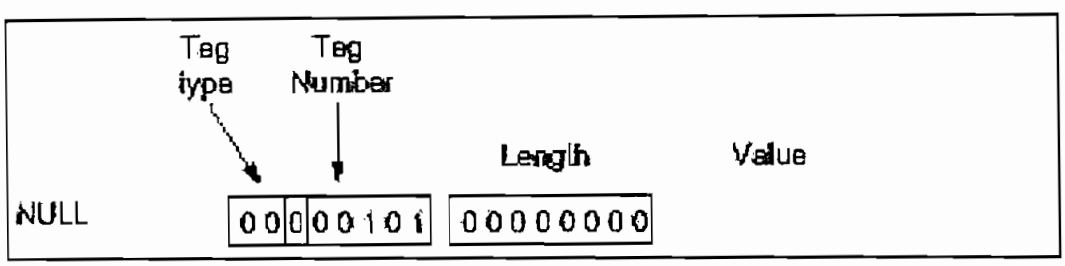


Figura 1.25 codificación de NULL

1.3.7.5.5 Codificación de un Identificador de Objetos

Para la codificación de un OI, en el campo tipo se necesita el valor de 06H, el campo longitud es variable, y en el campo de valores se establecen subidentificadores, obtenidos en base a identificadores que permiten ubicar al objeto en la estructura MIB, cuya cantidad es una menos que el número de identificadores. El primer subidentificador es el resultado de una fórmula matemática, la misma que involucra a los primeros dos identificadores del objeto. Si se hace referencia al esquema jerárquico MIB, un ejemplo de la codificación de un Identificador de Objeto sería como a continuación se indica.

Sea **X** un valor, que representa al primer identificador, y **Y** lo es para el segundo; entonces, si se toma a un objeto bajo la denominación: **1.3.6.1** , y se opera a través de la siguiente expresión matemática (establece SMI):

$$\text{Primer Subidentificador} = (X * 40) + Y$$

En este ejemplo se da el valor de 1 para X, y para Y el de 3(porque corresponden al primer y segundo identificador, respectivamente, en la secuencia nodal MIB). La expresión anterior se transformará en el valor de 43, para ser asignado al primer subidentificador:

$$\text{Primer Subidentificador} = (X * 40) + Y = (1 * 40) + 3 = 43$$

En la codificación, el campo de valores consta de la cantidad 43 en decimal para el primer subidentificador, ocupando un octeto; el segundo subidentificador requiere de i octeto (para representar al número 6, que es el tercer identificador, al cual, se lo toma como segundo subidentificador). Los demás identificadores se convierten en los subidentificadores sucesivos, y se representarán en los siguientes octetos(figura 1.26).

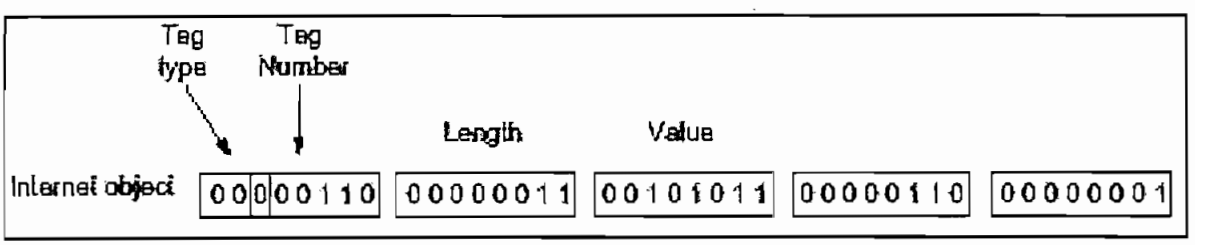


Figura 1.26 Codificación de un OI de 1.3.6.1

1.3.7.5.6 Codificación de Sequence

El tipo secuencia (sequence) se codifica siempre en forma estructurada. Los tipos de datos SÉQUENCE, se codifican, con un campo de tipo igual a 30H. El campo longitud es variable. En el campo de valores se da la información suficiente para identificar completamente al objeto, además de sus instancias.

En el ejemplo siguiente(figura 1.27), el campo longitud contiene 33H(51 en decimal), indicando que a continuación vendrán 51 octetos. En el campo de valores se tiene, primeramente una secuencia que destaca la ubicación nodal del objeto **sysDescr**(que se encuentra en el grupo **system): 1 3 6 1 2 1 1 1 0**

Donde 0, es un número que representa el valor del objeto.

Los otros 8 números reportan la ubicación del objeto en el dominio MIB.

Además, se cuenta con una marca de tipo Universal de una numeración igual a 6 (es decir, es un OI), pues para denotar los valores de un objeto tabular es necesario precisarlo inequívocamente.

La segunda secuencia de identificadores consta de un tipo de marca Universal, codificación de tipo primitivo, una marca igual a 4 (OCTECT STRING), una longitud de 27H (39 en decimal). Seguidamente se establecen las características de **sysDescr,** para describir un dispositivo de interconexión tipo puente.

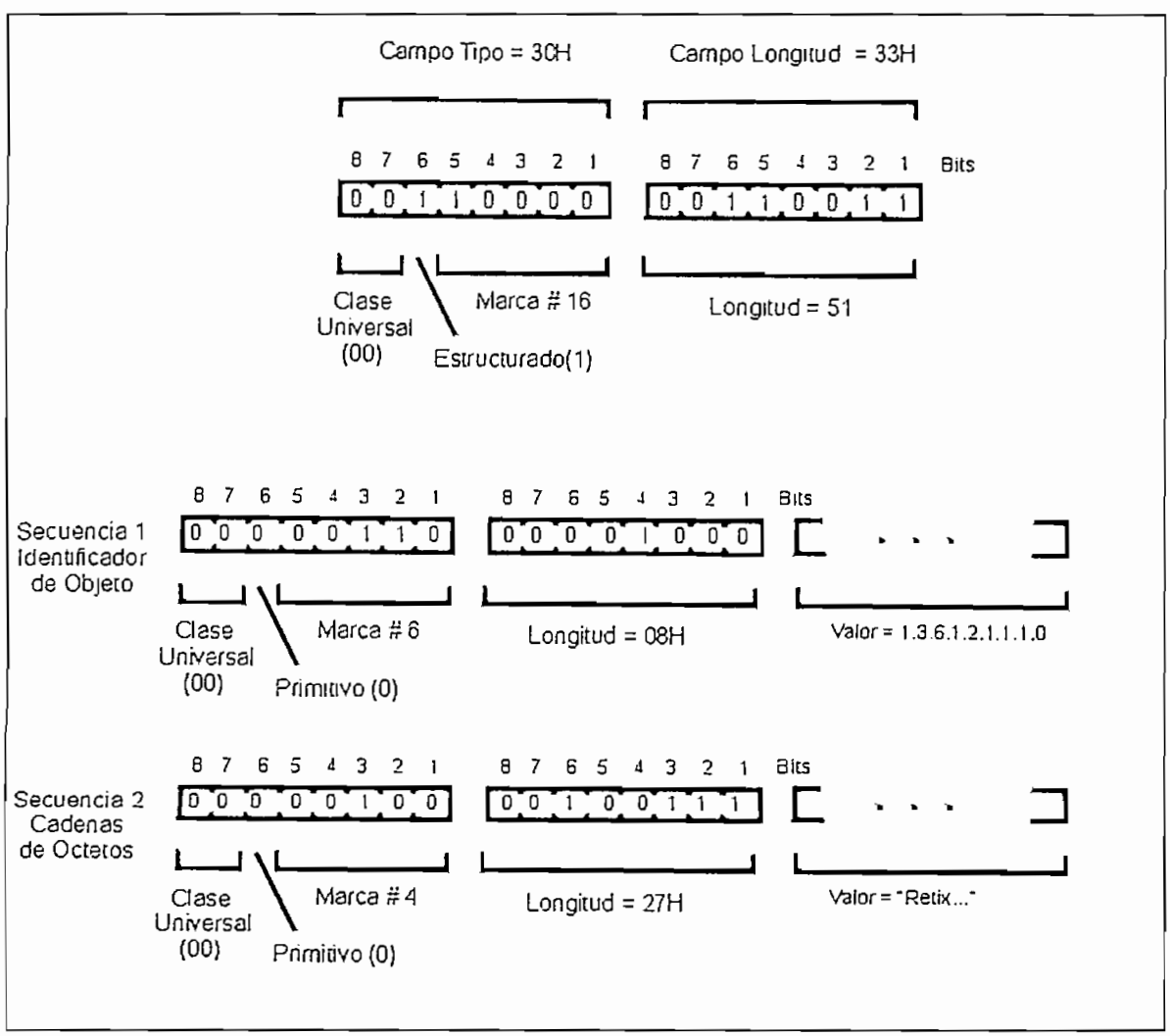


Figura 1.27 Codificación de SEQUENCE

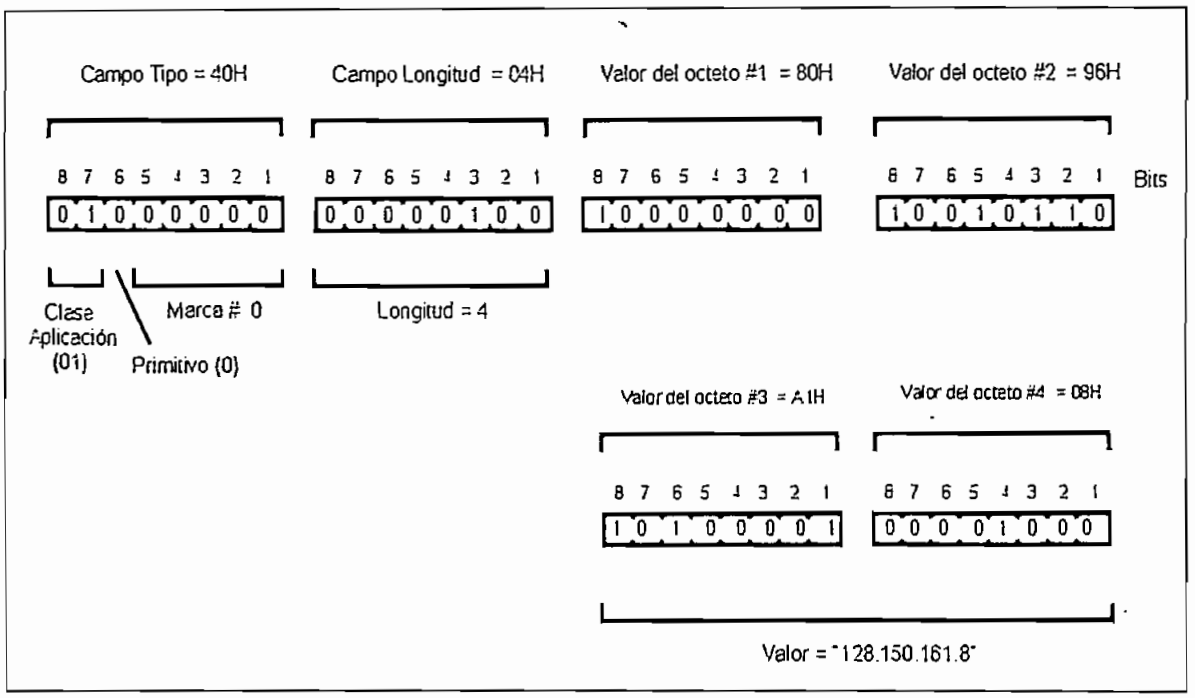


Figura 1.28 Codificación de ipAddress

1.3.7.6.2 Codificación de COUNTER

El tipo de datos Contador, se utiliza en el ejemplo(figura 1.29), para permitirle al objeto **icmpInMsgs**, perteneciente al grupo **icmp**, procesar el número de mensajes recibidos por un ruteador o un host. De este modo, el campo tipo posee el número 41H (representado en un calse aplicación, codificación primitiva, y una *marca* numérica igual a 1). El valor 190.105 da lugar a la presencia de 3 octetos, por lo que el campo de longitud debe tener el valor 03H. El campo de valores contiene el número 02 E6 99 H, que equivale a 190.105.

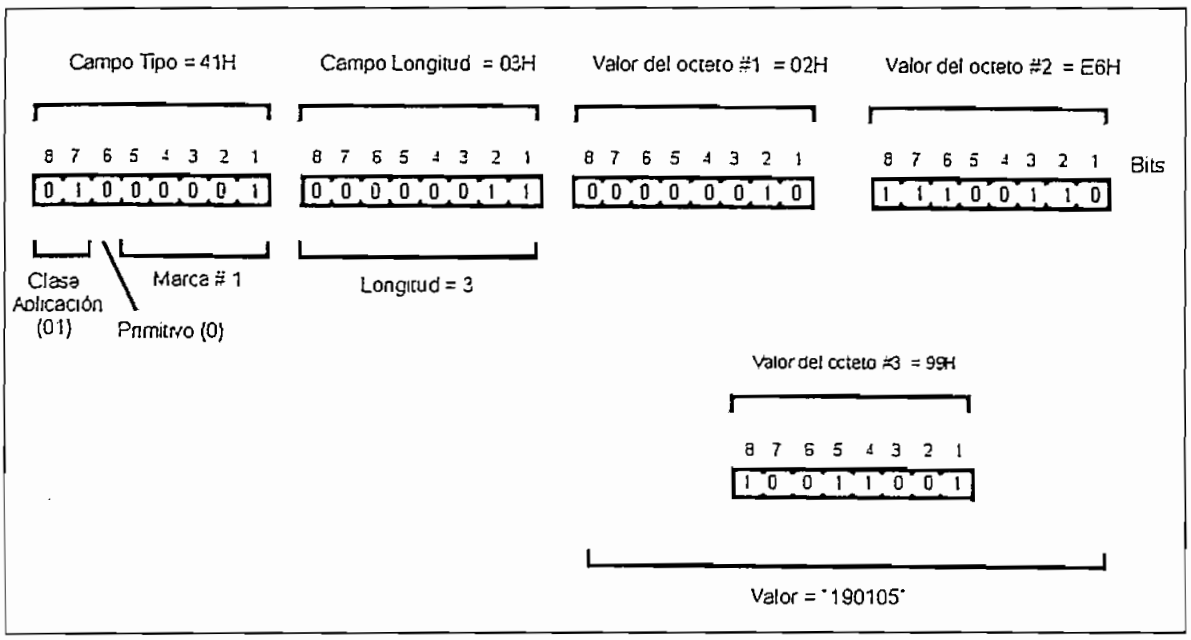


Figura 1.29 Codificación de COUNTER

1.3.7.6.3 Codificación de GAUGE

El GAUGE para el ejemplo (figura 1.30) cumple su función en una interfaz que puede soportar una cola máxima de 32 paquetes. Para codificar este valor de GAUGE, el campo tipo se pone en 42H (clase aplicación, codificación primitiva y número de marca igual a 2). Por el hecho de que el número 32 en decimal, se puede codificar a través de un solo octeto, en el campo de longitud, existe el valor de 01H. El campo de datos tiene un valor de 20 H (32 en decimal).

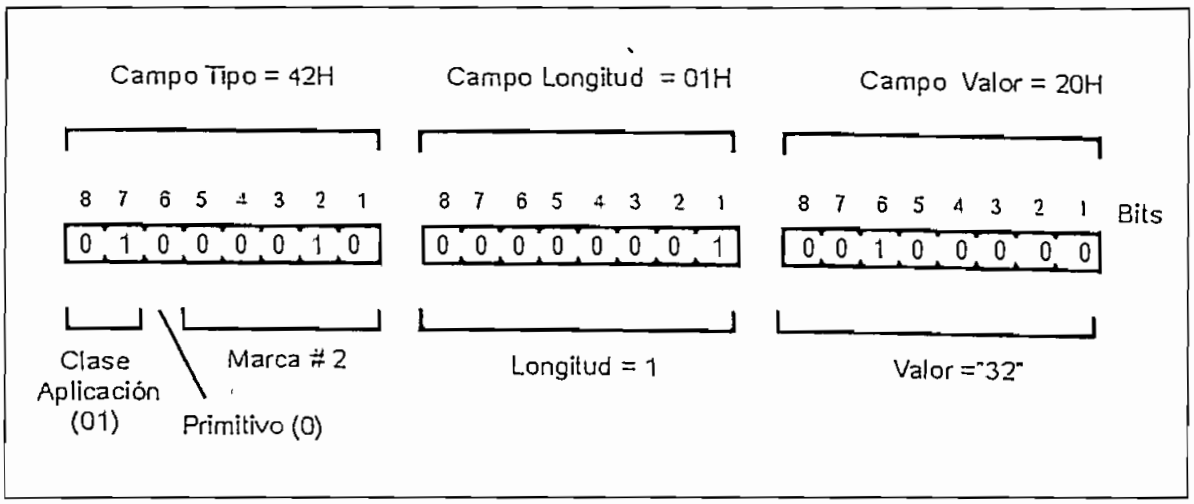


Figura 1.30 Codificación de GAUGE

1.3.7.6.4 Codificación de TIMETICKS

TIMETICKS se utiliza en este caso (figura 1.31) para el objeto **sysUpTime** (en el grupo **system**), para medir el tiempo desde que una entidad reinicializó al sistema. Para el ejemplo, el objetocontiene 263.61.156 centésiams de segundos. El campo tipo está en 43H (clase aplicación, codificación primitiva, y marca igual a 3), cuatro octetos codifican el valor de 263.691.156 (F B7 9B 94 H), por lo que el campo longitud tiene 4 H. el campo de datos, contiene el valor de TIMETICKS.

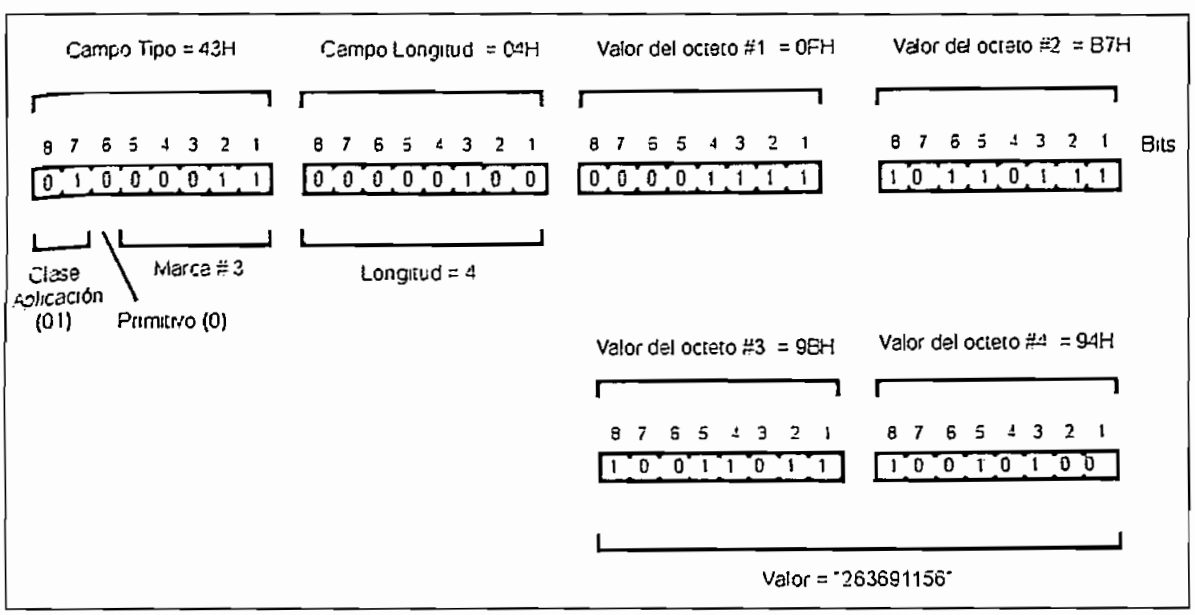


Figura 1.31 Codificación de TIMETICKS

1.3.7.6.5 Codificación de las PDUs SNMP[29]

Las PDUs de SNMP se codifican con una clase de tipo contexto específico, en forma estructurada y con numeraciones de marcas del 0 al 4.

El campo de tipo presenta rangos de A0H a A4H (porque se emplearán las PDUs de SNMPv1), los campos de longitud y datos contienen la información adecuada, como se presenta en la figura 1.32.

[29] *ROSE Marshall; The Simple Book; Prentice Hall; USA 1994.*

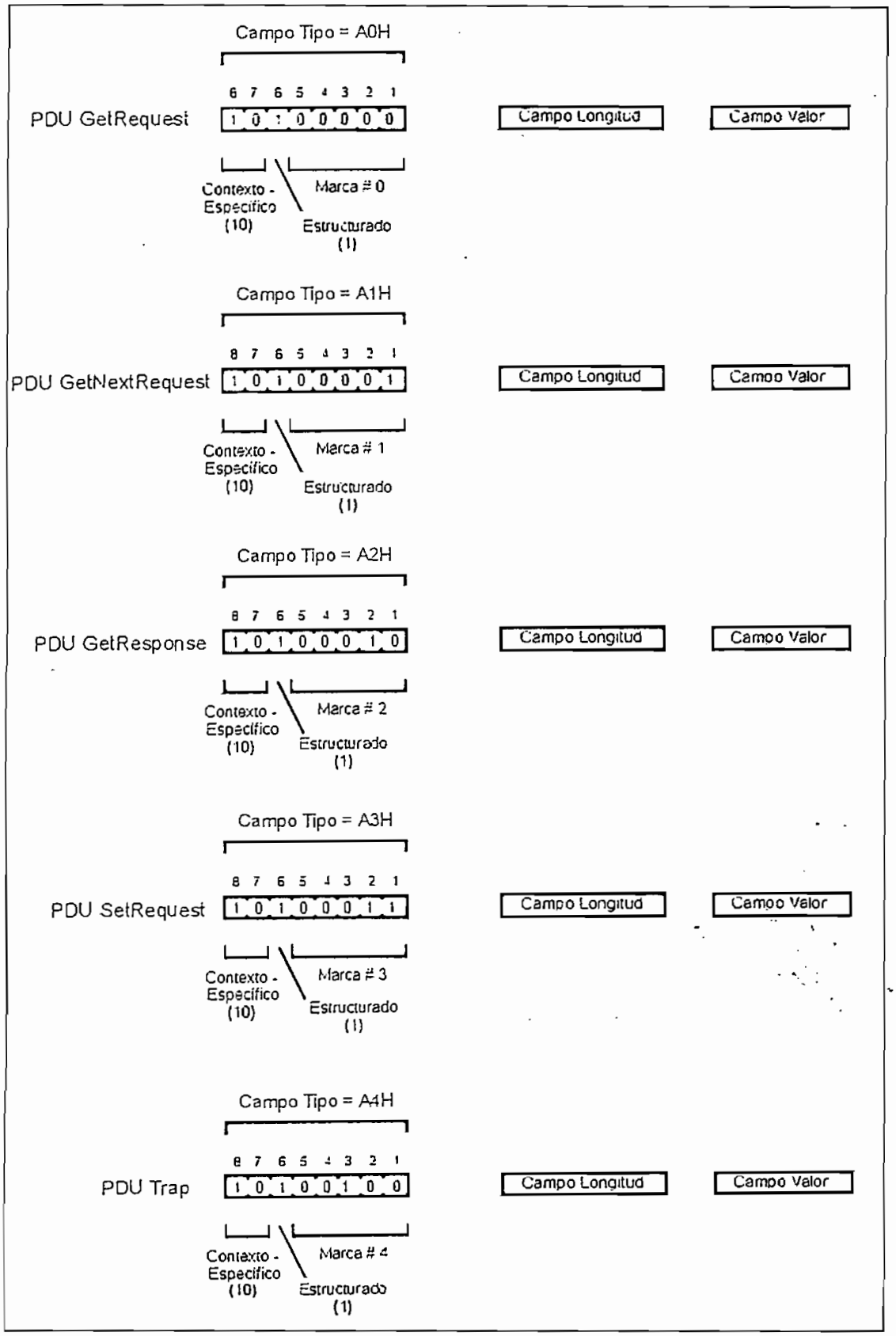


Figura 1.32 Codificación de las PDUs SNMP.

1.3.7.6.6 BER para una GetRequest

Aquí un ejemplo de codificación de un datagrama SNMP GetRequest , con petición sysDescr (1.3.6.1.2.1.1.1.0) (figura 1.33) y comunidad nula:

30	(00 | 1 |10000)	Tipo constructor n° 16	Mensaje de 32 bytes
20	(0010 | 0000)	Longitud (0x20 = 32)	
02	(0000 | 0010)	Tipo Integer	Versión 1
01	(0000 | 0001)	Longitud (0x01 = 1)	
00	(0000 | 0000)	Valor 0	
04	(0000 | 0100)	Tipo Octect String	Comunidad
00	(0000 | 0000)	Longitud	
a0	(10| 1| 00000)	Tipo específico de contexto y constructor con valor 0	GetRequest PDU: 26 bytes
19	(0001 | 1001)	Longitud (0x19=25)	
02	(0000 | 0010)	Tipo Integer	Request ID: 42
01	(0000 | 0001)	Longitud (0x01 = 1)	
2a	(0010 | 1010)	Valor 42	
02	(0000 | 0010)	Tipo Integer	Error Status: noErrors
01	(0000 | 0001)	Longitud (0x01 = 1)	
00	(0000 | 0000)	Valor = 0	
02	(0000 | 0010)	Tipo Integer	Err Index: 0
01	(0000 | 0001)	Longitud (0x01 = 1)	
00	(0000 | 0000)	Valor = 0	
30	(00 | 1 | 10000)	Tipo constructor n° 16	Lista de longitud variable: 14 bytes
0e	(0000 | 1110)	Longitud (0x0e = 14)	
30	(0011 | 0000)	Tipo constructor n° 16	Lista de longitud variable: 12 bytes
0c	(0000 | 1100)	Longitud (0x0e = 12)	
06	(00 | 0 |00110)	Tipo OBJECT-ID (0x06 = 6)	Identificador de objetos: 8 bytes
08	(0000 | 1000)	Longitud = 8 bytes	
2b	(0010 | 1011)	Objeto de Internet (0x2b = 43) Los primeros 2 n° a y b se codifican como 40a+b	Nombre: iso(1), org(3), dod(6), internet(1), mgmr(2), mib-2(1), system (1), sysDescr(1), 0
06	(0000 | 0110)	Dod	
01	(0000 | 0001)	internet	
02	(0000 | 0010)	mgmr	
01	(0000 | 0001)	Mib-2	
01	(0000 | 0001)	system	
01	(0000 | 0001)	sysDescr	
00	(0000 | 0000)	0	
05	(0000 | 0101)	Tipo NULL Longitud: 0	
00	(0000 | 0000)	Valor 0	

Figura 1.33 Datagrama SNMP GetRequest

1.3.8 ICMP: PROTOCOLO DE MENSAJES DE CONTROL DE INTERRED (INTERNET CONTROL MESSAGE PROTOCOL)[30]

Si un Ruteador no puede enrutar o entregar un Datagrama, o si detecta una situación anómala que afecta su capacidad de hacerlo (por ejemplo, la congestión), debe informar a la fuente original para que evite o solucione el problema.

ICMP es un mecanismo para realizar esta operación. Es considerado como una parte obligatoria de IP y debe ser incluido en todas sus implementaciones. ICMP es un protocolo de ***reporte de errores*** (no los corrige), además, ICMP solo puede informar del error a la fuente del Datagrama, es el host el que debe implementar mecanismos para enfrentar el problema[31].

La figura 1.34 indica la encapsulación de datagramas ICMP.

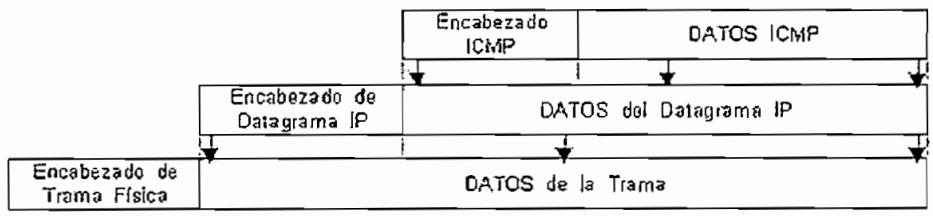


Figura 1.34 Encapsulamiento de Datagramas ICMP

Los mensajes de ICMP requieren doble encapsulación: Los mensajes ICMP viajan empaquetados en Datagramas IP. Aun así, no se considera a ICMP un protocolo de nivel superior a IP.

1.3.8.1 Formato del Mensaje ICMP

Aunque cada tipo de mensaje tiene su propio formato, todos ellos comparten los primeros tres campos: TIPO (8 bits), CODIGO (8 bits) y CHECKSUM (16 bits).

El campo TIPO identifica al tipo de mensaje ICMP y determina su formato. Puede tener alguno de estos valores:

---

[30] *FEIT Sidnie, "TCP/IP, Arquitectura, Protocolos e implementación con IPV6 y Seguridad IP", McGrawHill, España, 1998.*

[31] *TANENBAUM Andrew; Redes de Computadoras; Cuarta Edición; PrenticeHall; México 2003.*

En la figura 1.35 se indican los posibles valores del campo tipo para datagramas ICMP.

0	***Echo Reply*** (respuesta de eco)
3	***Destination Unreacheable*** (destino inaccesible)
4	***Source Quench*** (disminución del tráfico desde el origen)
5	***Redirect*** (redireccionar - cambio de ruta)
8	***Echo*** (solicitud de eco)
11	***Time Exceeded*** (tiempo excedido para un datagrama)
12	***Parameter Problem*** (problema de parámetros
13	***Timestamp*** (solicitud de marca de tiempo)
14	***Timestamp Reply*** (respuesta de marca de tiempo)
15	***Information Request*** (solicitud de información) - obsoleto-
16	***Information Reply*** (respuesta de información) - obsoleto-
17	***Addressmask*** (solicitud de máscara de dirección)
18	***Addressmask Reply*** (respuesta de máscara de dirección

Figura 1.35 Valores del campo tipo para ICMP

1.3.8.1.1 Mensajes Solicitud de Eco y Respuesta al Eco

Este es el tipo de mensaje que envía el host cuando se emplea el comando ping. Solicitud de Eco pide al host destino que responda con una Respuesta de Eco con un numero de secuencia apropiado.

En la figura 1.36 se indican los campos del mensaje solicitud de Eco y Respuesta al Eco para ICMP.

TIPO (8 o 0)	CODIGO (0)	CHECKSUM
Identificador		Numero de Secuencia
Datos Opcionales		

Figura 1.36 Mensaje Solicitud de Eco y Respuesta al Eco para ICMP

1.3.8.1.2 Mensaje Destino Inaccesible.

Es el mensaje empleado para reportar que no es posible entregar un Datagrama. En la figura 1.37 se describe el campo CODIGO del mensaje para los tipos de mensajes ICMP:

0	no se puede llegar a la red
1	no se puede llegar al host o aplicación de destino
2	el destino no dispone del protocolo solicitado
3	no se puede llegar al puerto destino o la aplicación destino no está libre
4	se necesita aplicar fragmentación, pero el flag correspondiente indica lo contrario
5	la ruta de origen no es correcta
6	no se conoce la red destino
7	no se conoce el host destino
8	el host origen está aislado
9	la comunicación con la red destino está prohibida por razones administrativas
10	la comunicación con el host destino está prohibida por razones administrativas
11	no se puede llegar a la red destino debido al Tipo de servicio
12	no se puede llegar al host destino debido al Tipo de servicio

Figura 1.37 Valores del campo Código para ICMP

En la figura 1.38 se indican los campos del mensaje Destino Inaccesible para ICMP.

TIPO (3)	CODIGO (0...12)	CHECKSUM
NO – USADO (debe ser cero)		
Encabezado IP + Primeros 8 bytes de Datos IP		

Figura 1.38 Mensaje Destino Inaccesible para ICMP

Los errores de red inaccesible por lo general implican fallas de enrutamiento. Debido a que el mensaje ICMP contiene la cabecera del datagrama que lo produjo (en el campo de datos), el origen sabrá cual destino es inaccesible.

1.3.8.1.3 Mensaje de Disminución del Tráfico desde el origen.

Debido a que IP funciona sin conexión un Ruteador no puede reservar memoria o recursos de comunicación antes de recibir los Datagramas. En consecuencia los Ruteadores pueden verse repentinamente saturados por el trafico. A esta situación se le llama congestión.

El congestionamiento se da por que un Host de alta velocidad genera Datagramas mas rápido de lo que el Ruteador puede manejar o porque muchos Host envían Datagrama a la misma dirección al mismo tiempo.

Cuando los Datagramas llegan mas rápido de lo que un Ruteador puede manejarlos, este los coloca en un buffer. Si los Datagramas son parte de una ráfaga pequeña, esto soluciona el problema, pero si continúan llegando Datagramas se saturan los buffers y el Ruteador debe descartar los nuevos Datagramas. Es entonces cuando el Ruteador genera un mensaje ICMP de Acallamiento de Origen solicitando a este reducir la tasa de envío de Datagramas. No existe un mensaje ICMP para revertir esta solicitud, en general poco después de bajar la tasa de envío, los Hosts la aumentan progresivamente hasta recibir otro mensaje de Acallamiento de Origen.

En la figura 1.39 se indican los campos del mensaje de Acallamiento de Origen para ICMP.

TIPO (4)	CODIGO (0)	CHECKSUM
NO - UTILIZADO (debe ser cero)		
Encabezado IP + 8 primeros bytes de Datos IP		

Figura 1.39 Mensaje de Acallamiento de Origen para ICMP

El objetivo de este mensaje era aliviar el problema de la congestión, pero no tuvo éxito. Se dejo al implementador decidir sobre cuando enviar estos mensajes, por lo que cada fabricante emplea su política favorita sin que ninguna solucione el problema del todo. Por otra parte, ICMP informa al Host de origen que su Datagrama ha sido descartado, pero puede que este Host no sea el causante de

la congestión. Además, Como responder al mensaje ICMP?. Documentos como ***Requisitos para los Ruteadores*** (RFC 1812) estipulan que NO se deben enviar mensajes de Acallamiento de Origen. Se esta trabajando en mecanismos mas eficientes.

1.3.8.1.4 Mensaje Redireccionar

Se asume que los Ruteadores conocen rutas correctas. Los hosts comienzan con información mínima de enrutamiento y aprenden nuevas rutas de los Ruteadores. En caso de que un host utilice una ruta no optima, el Ruteadores que lo detecta envía un mensaje ICMP Redireccionar solicitándole que actualice su tabla de enrutamiento IP.

En la figura 1.40 se indican los campos del mensaje de Redireccionar para ICMP.

TIPO (5)	CODIGO (0...3)	CHECKSUM
Dirección IP del Ruteador		
Encabezado de IP + 8 primeros bytes de Datos IP		

Figura 1.40 Mensaje Redireccionar para ICMP

1.3.8.1.5 Mensaje Tiempo Excedido

Debido a que los Ruteadores solo deciden sobre el próximo "Salto" usando tablas locales, errores en esas tablas pueden generar "ciclos de enrutamiento" para algún destino. Esto provoca que los Datagramas sean descartados por vencimiento de su TTL. Siempre que un Ruteador descarte un Datagrama ya sea por vencimiento de TTL o por vencimiento del Tiempo de Reensamblado, envía un mensaje de Tiempo Excedido a la fuente. En la figura 1.41 se indican los campos del mensaje Tiempo Excedido para ICMP.

TIPO (11)	CODIGO (0 o 1)	CHECKSUM
NO – UTILIZADO (debe ser cero)		
Encabezado de IP + 8 primeros bytes de Datos IP		

Figura 1.41 Mensaje Tiempo Excedido para ICMP

CODIGO = 0: Descartado por vencimiento de TTL

CODIGO = 1: Descartado por vencimiento de Tiempo de Reensamblado.

1.3.8.1.6 Mensaje Problema de Parámetros

Cuando un Ruteador o un Host encuentra un problema que no ha sido cubierto con los mensajes ICMP anteriores, envía este mensaje.

En la figura 1.42 se indican los campos del mensaje Problema de Parámetros para ICMP.

TIPO (12)	CODIGO (0 o 1)	CHECKSUM
Indicador	NO – Utilizado (debe ser cero)	
Encabezado de IP + 8 primeros bytes de Datos IP		

Figura 1.42 Mensaje Problema de Parámetros para ICMP

El campo indicador apunta al campo dentro del encabezado IP que generó el problema.

1.3.8.1.7 Mensaje Solicitud de Timestamp y Respuesta de Timestamp

Una técnica sencilla provista por TCP/IP para sincronizar relojes emplea ICMP para obtener la hora de otro host. Un host envía a otro una solicitud de Timestamp, solicitándole que informe su valor actual para la hora del día. El otro host envía una respuesta de Timestamp con esa información.

En la figura 1.43 se indican los campos del mensaje Solicitud de Timestamp y Respuesta de Timestamp para ICMP.

TIPO (13 o 14)	CODIGO (0)	CHECKSUM
Identificador		Numero de Secuencia
Timestamp Origen		
Timestamp al Recibir		
Timestamp al Transmitir		

Figura 1.43 Mensaje Solicitud de Timestamp y Respuesta de Timestamp para ICMP

1.3.8.1.8 Mensaje Solicitud de Mascara de Subred y Respuesta de Mascara de Subred

Para aprender la mascara de subred utilizada por la red local, un host puede enviar un mensaje ICMP Solicitud de Mascara de Subred a un Ruteador y esperar su Respuesta. Si el host no conoce la dirección del Ruteador, puede enviar este mensaje por difusión.

En la figura 1.44 se indican los campos del mensaje Solicitud de Mascara de Subred y Respuesta de Mascara de Subred para ICMP.

TIPO (17 o 18)	CODIGO (0)	CHECKSUM
Identificador		Numero de Secuencia
Mascara de Subred		

Figura 1.44 Mensaje Solicitud de Mascara de Subred y Respuesta de Mascara de Subred para ICMP

1.3.9 ARP (PROTOCOLO DE RESOLUCIÓN DE DIRECCIONES)

El protocolo de resolución de direcciones es responsable de convertir las dirección de protocolo de alto nivel(direcciones IP) a direcciones de red físicas.

En cada host se tiene una tabla que dice la correspondencia que hay entre la dirección física y la dirección IP. Cuando tenemos que enviar un paquete a una dirección IP de la que se desconoce la dirección física entra el funcionamiento el protocolo ARP para actualizar los valores de la tabla. Este protocolo es el encargado de obtener la dirección física de un host de la que conoce la dirección IP. Para conseguirlo debe acceder a recursos de bajo nivel.
Únicamente hay dos tipos de mensaje que tienen el mismo formato: petición ARP y respuesta ARP.

Los mensajes ARP van a ser encapsulados directamente en una trama Ethernet En el campo tipo de la cabecera de la trama Ethernet es necesario especificar que

contiene un mensaje ARP. El emisor se debe encargar de poner el valor correspondiente y el receptor de mirar el contenido de ese campo.

Como Ethernet asigna un único valor para los dos mensajes ARP, el receptor debe examinar el campo operación del mensaje ARP para determinar si el mensaje recibido es una petición o una respuesta.

1.3.10 SOCKETS [32]

Un socket es un punto final en la comunicación; en otras palabras, un objeto a través del cual una aplicación puede enviar o recibir paquetes de datos a través de la red. Un socket tiene un tipo y está asociado con un proceso en ejecución, y puede tener un nombre. En la actualidad, los sockets intercambian datos sólo con otros sockets en el mismo dominio de comunicación bajo el protocolo IP.

Un socket permite conectarse a un equipo remoto e intercambiar datos con el protocolo de control de transmisión (TCP) o con el protocolo de datagramas de usuario (UDP). Ambos protocolos se pueden usar para crear aplicaciones cliente servidor.

Por otra parte el tipo de comunicación establecida puede ser orientada a conexión o no orientada a conexión. Ambos tipos son bidireccionales. Una comunicación orientada a conexión funciona de forma parecida a una llamada telefónica; esto es, primero se establece la conexión, después se intercambian peticiones y respuesta, y finalmente se libera la conexión. En cambio, en una comunicación no orienta a conexión, no se realiza una conexión, simplemente hay un intercambio de peticiones y respuestas. En éste último caso, el servidor recibe junto con los datos la dirección del cliente, con el fin de poder devolverle la respuesta.

El tipo de comunicación orientado a conexión utiliza el protocolo TCP y es fiable; esto es, si se produce un error en la transmisión, la información será reenviada

---

[32] *CEBALLOS, Francisco Javier: "Visual C++", Alfaomega Grupo Editor, México, 1999.*

hasta que llegue correctamente a su destino. En cambio, el tipo de comunicación no orientado a conexión utiliza el protocolo UDP y no es fiable; esto es, utilizando este tipo de comunicación es posible que se pierdan paquetes de información, por lo que solo es recomendable utilizarlo cuando ello no provoca ningún perjuicio (por ejemplo, un mensaje enviado a todas los hosts con la hora del servidor para que todas ellas sincronicen su reloj con él). En ambos tipos de comunicación debe haber, al menos, un socket en la aplicación cliente (la que pide información) y otro socket en la aplicación servidora (la que proporciona información solicitada) por cada uno de os sockets de los clientes.

Los sockets permiten implementar una arquitectura cliente-servidor. La comunicación ha de ser iniciada por uno de los programas, por este motivo se denomina programa cliente. El segundo programa espera a que otro inicie la comunicación, por este motivo se denomina programa servidor.

En la figura 1.45 se indica un Esquema básico del funcionamiento de los sockets.

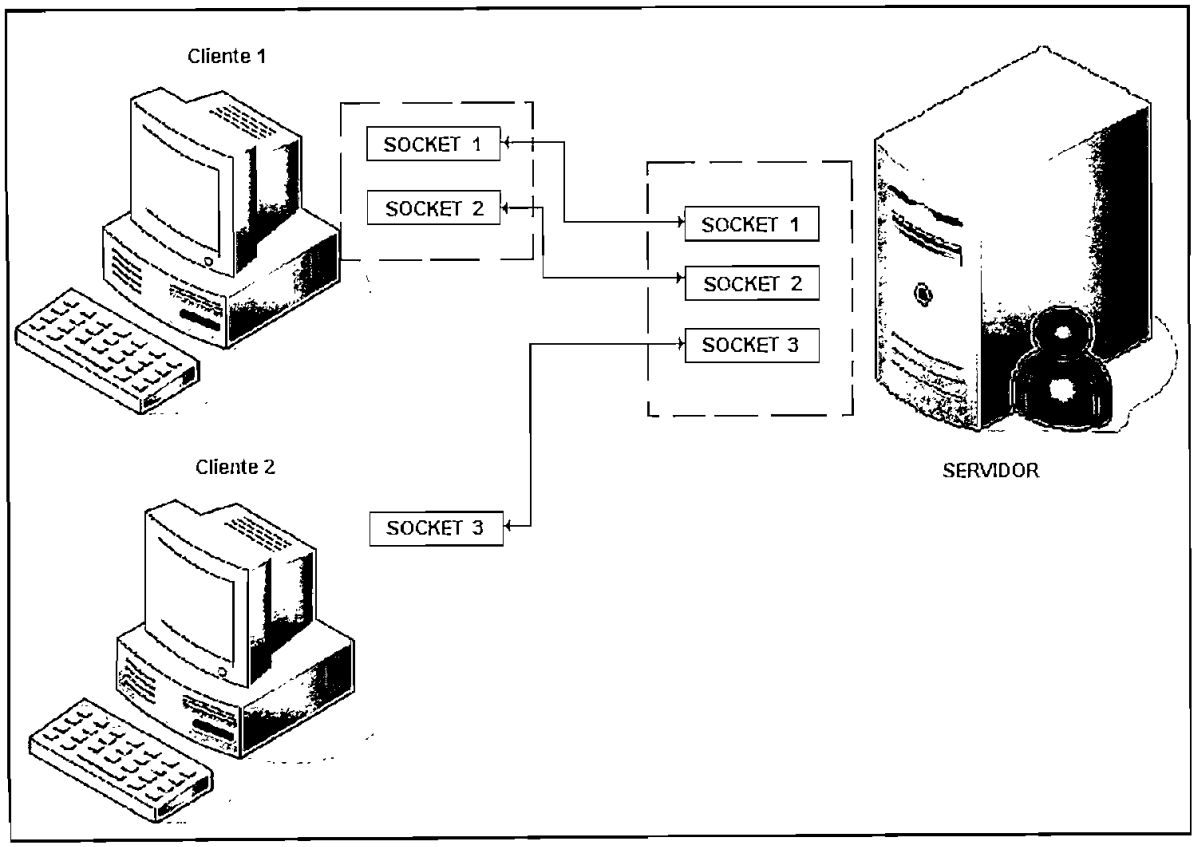


Figura 1.45 Esquema de funcionamiento de los sockets.

1.3.10.1 Arquitectura Cliente / Servidor

Una de las formas de comunicación entre dos programas que utilizan sockets es la arquitectura cliente / servidor. En este caso uno de los programas debe estar arrancado y a la espera de que otro cualquiera desee conectarse a él. Nunca da el primer paso en la conexión. Al programa que actúa de esta forma se lo conoce como servidor. Su nombre se debe a que normalmente es quien posee la información, sirviendo la misma a quien la solicite.

El otro programa es quien da el primer paso. En el momento de arrancarlo o cuando lo necesite, intentará conectarse con el servidor. Este programa se denomina cliente y su nombre se debe a que es quien solicita información al servidor.

Resumiendo, el servidor es el programa que permanece pasivo a la espera de que alguien solicite una conexión con él, mientras que el cliente es el programa que solicita la conexión.

1.3.10.2 La Conexión

Para poder realizar la conexión entre dos programas se necesitan los siguientes datos:

- *Nombre o dirección IP del servidor*: si se utiliza el nombre, entonces debe traducirse a su dirección IP.

- *Nombre del servicio o puerto del servidor*: si se utiliza el nombre del servicio, entonces debe traducirse a su número de puerto.

En el capítulo siguiente se describen los comandos que s utilizan para establecer la conexión, recepción de mensaje y el cierre de conexión de los sockets.

CAPÍTULO 2: DESARROLLO DE LA APLICACIÓN EN VISUAL C++

2.1 INTRODUCCIÓN

El manejo de sockets en el entorno de programación Visual C++, conlleva al uso de librerías y clases propias del mismo, y específicas para ello; por lo que, en éste capítulo se describen dichas herramientas y clases que se utilizan para establecer la comunicación con otros componentes de red, y así determinar la funcionalidad de las mismas para el desarrollo de la aplicación planteada.

2.2 DEFINICIÓN DE LA APLICACIÓN

La aplicación a desarrollarse e implementarse permite hacer una exploración mediante el uso del protocolo ICMP, de los elementos presentes dentro de una subred, así como de los elementos de otras subredes con las cuales se tenga comunicación; para después mediante el uso del protocolo SNMP, acceder a MIBs remotas de elementos administrables para distinguirlos de los otros componentes presentes, clasificarlos y determinar la información útil que permita esquematizar la topología de la red explorada.

Posteriormente y mediante el uso de la interfaz gráfica que ofrece el entorno de programación Visual C++, mostrar en pantalla y con ayudas visuales, la topología de red encontrada.

Para generar los datagramas necesarios para realizar la exploración, se hacen uso de clases propias diseñadas para alcanzar dicho objetivo basadas en clases CSocket.h y CAsyncSocket.h, las cuales permiten la creación de sockets sincrónicos y asincrónicos para la comunicación con otros dispositivos.

2.3 ENTORNO VISUAL C++[33]

Se indican a continuación elementos del lenguaje de programación y de la herramienta de desarrollo Visual C++, para la aplicación; empezando con una breve descripción de su entorno y sentencias de lenguaje.

2.3.1 LOS COMPONENTES DE VISUAL C++

Microsoft Visual C++ está formado por dos sistemas completos de desarrollo de aplicación para Windows en un producto. Si se desea, se pueden desarrollar programas para Windows en lenguaje C utilizando sólo la API Win32, se pueden utilizar muchas herramientas de Visual C++, incluyendo los editores de recursos, para simplificar la programación de bajo nivel de Win32.

Visual Studio 6.0 es un conjunto de herramientas de desarrollo que incluye a Visual C++ 6.0. el sistema de ayuda en línea (ahora integrado con el visualizador de la biblioteca MSDN) funciona igual que un explorador de red.

2.3.1.1 El editor de recursos.

Cuando se pulsa la pestaña ClassView se presenta un esquema de las clases del proyecto en desarrollo a manera de árbol, mostrando las funciones miembro y los datos miembro, y en el cual se nos permite manipular los componentes de la aplicación a nivel de clase C++.

Cuando se pulsa la pestaña ResourceView (Vista de Recursos) en la ventana del entrono de trabajo Visual C++, se puede seleccionar un recurso para su edición. La ventana principal alberga un *editor de recursos,* adaptado al tipo de recurso. la ventana también puede albergar un editor para menús y un editor gráfico potente para cuadros de diálogo, e incluye herramientas para editar íconos, mapas de bits y cadenas. El editor de diálogos permite la inserción de controles: estándares de Windows y los nuevos controles comunes de Windows.

---

[33] *KRUGLINSKI, David; "PROGRAMACIÓN AVANZADA CON VISUAL C++6.0"; MacGrawHill, Madrid, España,1999*

Entre éstos recursos se incluyen elementos específicos del proyecto, como archivos de mapa de Bits (BMP) y de icono (ICO), y recursos que son comunes a todos los programas de Visual C++, como las cadenas de mensajes de error.

Cuando se pulsa la pestaña FileView se presenta un esquema a manera de árbol que contiene los archivos que utiliza la aplicación en desarrollo.

En la figura 2.1 se indica las ventanas disponibles en el espacio de trabajo de Visual C++ 6.0.

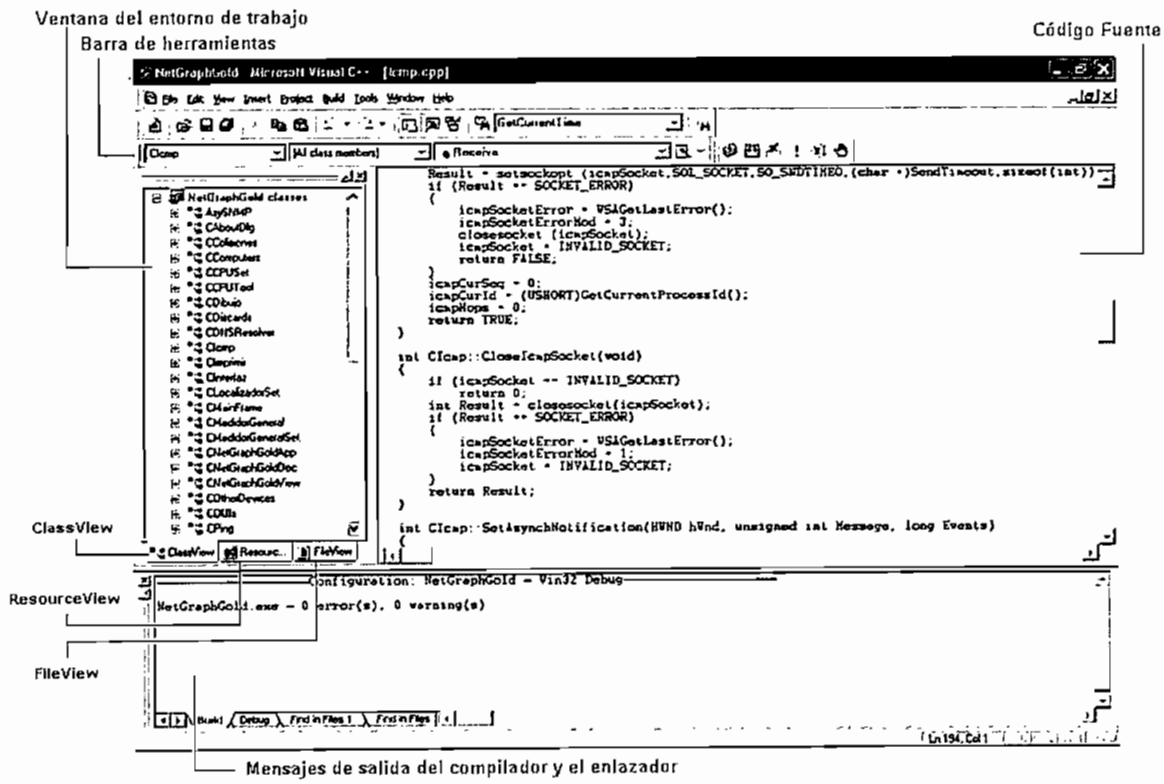


Figura 2.1 Ventanas de Visual C++ 6.0

2.3.1.2 El compilador de C/C++

El compilador de Visual C++ puede procesar código fuente tanto en C como en C++. Determina el lenguaje mirando la extensión del nombre del archivo de código fuente. Si la extensión es C, es un código fuente en C, mientras que si es CPP o CXX, el código es de C++.

2.3.1.3 El editor de código fuente

Visual C++ 6.0 incluye un sofisticado editor de código fuente que permite muchas características, como la asignación dinámica de colores en función de la sintaxis, la tabulación automática y la impresión mejorada. La característica AutoComplete de Visual C++, permite introducir el comienzo de una sentencia de programación para que el editor proporcione una lista de posibles terminaciones entre las cuales elegir, lo cual simplifica el uso de funciones API Win32.

2.3.1.4 El compilador de recursos

El compilador de recursos de Visual C++ lee un archivo RC (archivo de guión de recursos o plantilla de recursos) en ASCII que se encuentra en los editores de recursos y escribe un archivo binario RES que es el que utilizará el enlazador.

2.3.1.5 El enlazador

El enlazador lee los archivos OBJ y RES generados por el compilador de C/C++ y el compilador de recursos, y accede a los archivos LIB para el código de la MFC, el código de la biblioteca de tiempo de ejecución y el código de Windows. Después escribe el archivo EXE del proyecto.

2.3.1.6 El depurador

El depurador interactúa con Visual C++ para asegurar que los puntos de ruptura se guardan en disco. Los botones de la barra de herramientas insertan y eliminan puntos de ruptura y controlan la ejecución paso a paso. La Figura 2.2 muestra el depurador de visual C++. En las ventanas: Variables y Watch (Observación) se pueden ampliar los objetos para mostrar todos los datos del miembro de la clase derivada y de las clases base. Si se coloca el cursor sobre una variable simple, el depurador muestra su valor en una ventana pequeña.

Una característica adicional del depurador es la de: Editar y Continuar <<Edit and Continue>>, que permite depurar una aplicación, cambiar la

aplicación y después continuar la depuración con el nuevo código. Lo que ayuda a no abandonar manualmente el depurador , volver a compilar y después volver a depurar. Visual C++ 6.0 compila automáticamente los cambios y reinicia el depurador.

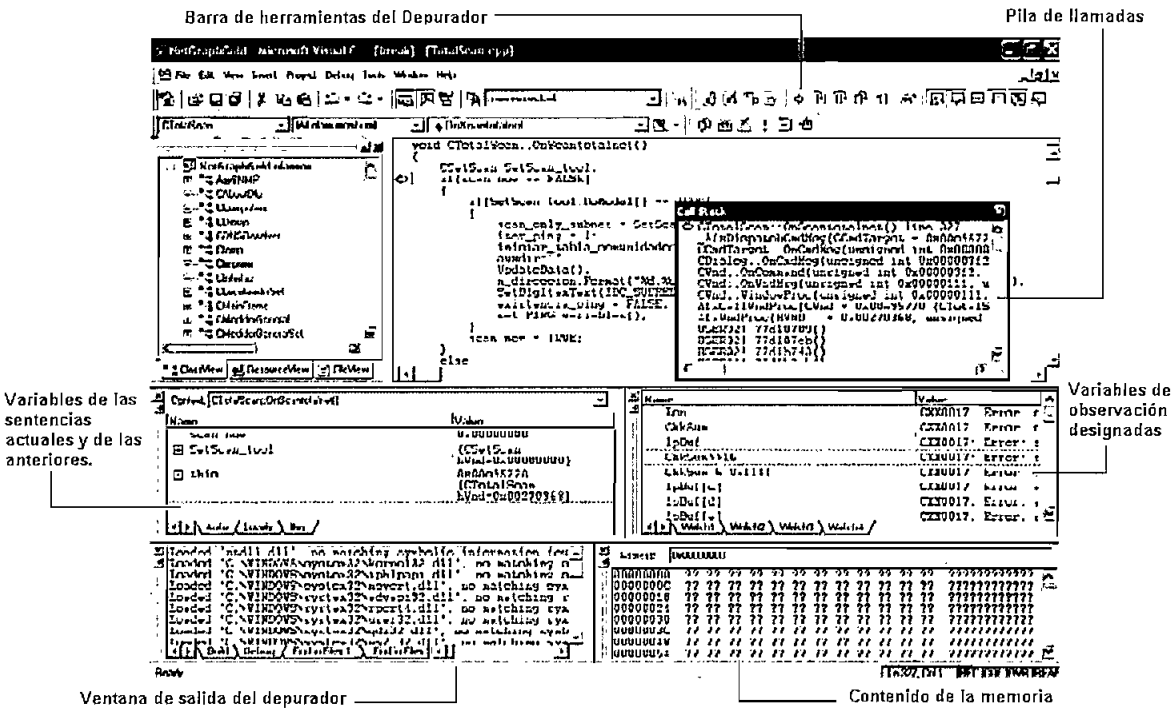


Figura 2.2 La Ventana del depurador de Visual C++ 6.0

2.3.1.7 AppWizard

El AppWizard (asistente de aplicaciones, figura 2.3) es un generador de código que crea un esqueleto de trabajo de una aplicación de Windows con características, nombres de clase y nombres de archivo de código fuente que se especifican a través de cuadros de diálogo. Con el AppWizard se empieza a crear una aplicación con gran rapidez.

2.3.1.8 ClassWizard

El ClassWizard (Asistente de clase, figura 2.4) es un programa (implementado como DLL) al que se puede acceder desde el menú View de Visual C++. El ClassWizard simplifícale mantenimiento del código de una clase de Visual C++. Ésta herramienta escribe los prototipos de una clase nueva, los cuerpos de la función y, si es necesario, el código para enlazar el mensaje de Windows a la función.

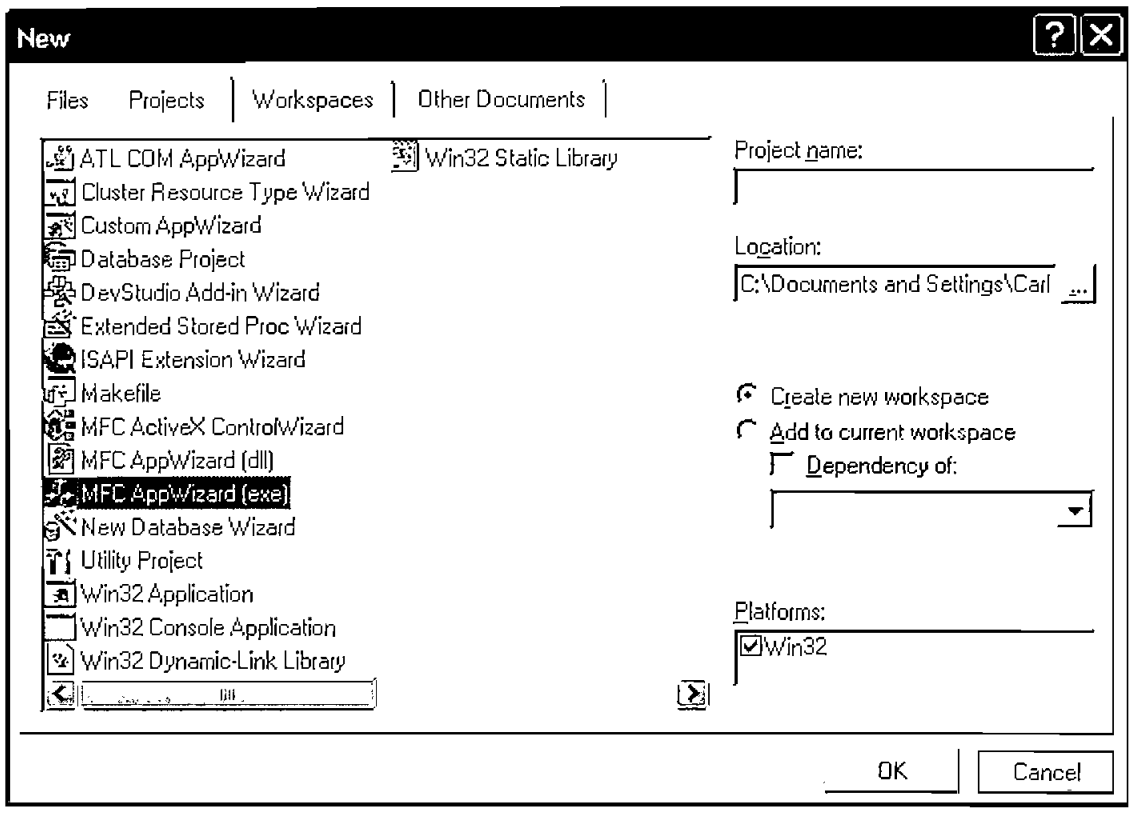


Figura 2.3 La Ventana de AppWizard de Visual C++ 6.0

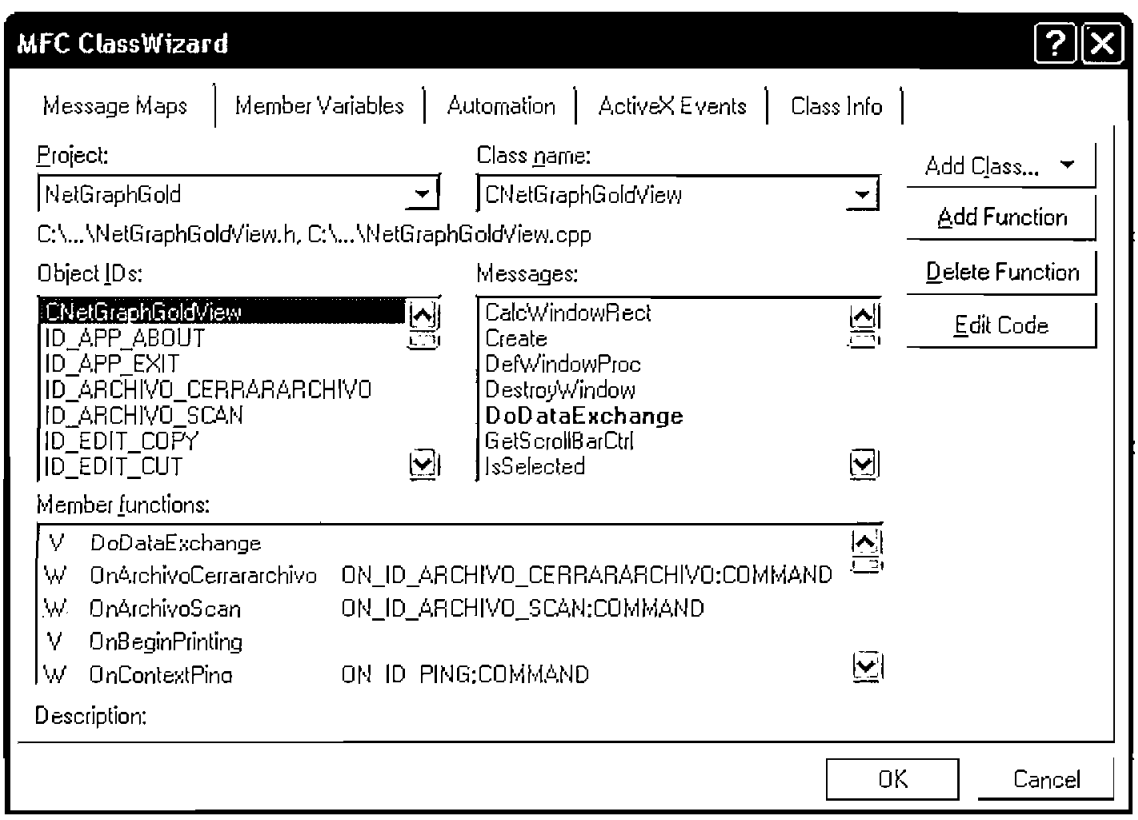


Figura 2.4 La Ventana de ClasWizard de Visual C++ 6.0

2.3.1.9 Clases

En general una clase es un conjunto de atributos, mientras que en programación una clase es un conjunto de objetos que comparte estructura y comportamiento común.

2.3.1.10 Ayuda en Línea

El sistema de ayuda se ha trasladado a una aplicación aparte llamada el Visualizador de la biblioteca MSDN. Éste sistema de ayuda se basa en HTML. Cada uno de los temas está cubierto en un documento HTML individual; después, todos están combinados en archivos indexados.

Se permite acceder a la ayuda de cuatro maneras:

- ✓ ***A través de un libro:*** Cuando se selecciona *Contents* (contenidos) en el menú de ayuda de Visual C++, la aplicación de la biblioteca MSDN pasa a una vista de contenido. En esta vista, la documentación del Visual Studio, Visual C++, el SDK de Win32 y más están organizados de manera jerárquica en libros y capítulos.

- ✓ ***A través de un tema:*** Cuando se selecciona Buscar (*Search*), en el menú de Ayuda Visual C++, se abre automáticamente el Visualizador de la Biblioteca MSDN. Se seleccionará la pestaña Índice (*index*), para introducir una palabra clave y ver los temas y artículos que incluyen esa palabra clave.

- ✓ ***Ayuda con F1:*** éste es el mejor. Simplemente se desplaza el cursor a un nombre de una función, macro o clase, se presiona la tecla F1 y el sistema de ayuda se pone a trabajar. Si se encuentra el nombre en varios sitios, en los archivos de ayuda de MFC y de Win32, se selecciona el tema de ayuda deseado en una ventana que tiene la lista de opciones.

2.4 IMPLEMENTACIÓN Y USO DE SOCKETS

2.4.1 SOCKET[34]

Para poder describir el uso de un socket, ser define primero al socket como: un identificador para un servicio particular en un nodo particular de una red. El socket se compone de una dirección de nodo y un número de puerto, que identifica el servicio. Por ejemplo, el puerto 80 de un nodo de Internet indica un servidor Web. Hay dos clases de socket: secuencias (bidireccionales) y datagramas.

Se conoce como ***socket UDP*** al socket que transmite datagramas en el Protocolo de datagramas de usuario (UDP).
El socket es una representación abstracta del extremo en el proceso de comunicación. Para que haya comunicación en una red (a través de la interfaz de sockets) el programa necesita un socket en cada lado que conversa.

La conversación entre los dos sockets puede ser orientada a conexión (punto a punto) o sin conexión. Las clases Windows Sockets, proporcionan servicios que permiten a las aplicaciones enlazarse a un puerto y una dirección IP en particular en un servidor, iniciar y aceptar una conexión, enviar y recibir datos, y cerrar una conexión.

Hay dos tipos de sockets:

1. Un socket de flujos (stream socket) proporciona un flujo de datos de dos vías, confiable, secuenciado y no duplicado utilizando TCP.

2. Un socket de datagrama proporciona un flujo bidireccional usando UDP.

[34] *MSDN Ayuda en Línea. www.msdn.com*

2.4.2 IMPLEMENTACIÓN DE SOCKETS

Algunos sistemas lo implementan como bibliotecas de programación, es el caso de Visual C++, pudiendo ser utilizadas las funciones que permiten comunicarse en una red llamándolas mediante código de programa.

2.4.3 IDENTIFICADOR DE SOCKETS

Los programas basados en sockets crean un socket y entonces, como paso aparte lo conectan a un extremo destino.

En un programa de red TCP/IP que transmite un datagrama utilizando protocolos sin conexión, éste especifica la dirección destino del datagrama pero no establece una conexión directa con el anfitrión sino que transmite el datagrama a la dirección destino. El software de red maneja el proceso de entrega.

Los diseñadores de Windows Sockets API crearon la función llamada *socket* de la clase CSocket, que permite que un programa obtenga un identificador de socket sin especificar la dirección destino, y la función *create* perteneciente a la clase asincrónica CAsyncSocket que permite la misma operación pero con sockets de éste último tipo.

La clase CSocket se deriva de la clase CAsyncSocket por lo que hereda las propiedades de encapsulación. Un objeto de la clase CSocket representa un nivel más alto de abstracción de Windows Sockets que el de un objeto CAsyncSocket.

2.4.4 CREACIÓN DE UN SOCKET

Al llamar a la función de creación del socket, éste devuelve un identificador por el cual se reconoce en un registro dentro de una tabla de descriptores para conocer información sobre un socket.

Las funciones siguientes son las que se usan con éste propósito:

SOCKET socket (int af, int type, int protocol);

Donde se debe especificar tres parámetros: la familia de protocolos (*af*), el tipo de socket (*type*) y el protocolo (*protocol*).

Y las funciones específicas que funcionan en ambas clases CAsyncSocket y CSocket , y de forma más específica:

BOOL Create(UINT nSocketPort = 0, int nSocketType = SOCK_STREAM, long lEvent = FD_READ | FD_WRITE | FD_OOB | FD_ACCEPT | FD_CONNECT | FD_CLOSE, LPCTSTR lpszSocketAddress = NULL);

BOOL Create(UINT nSocketPort = 0, int nSocketType = SOCK_STREAM, LPCTSTR lpszSocketAddress = NULL);

Donde se debe especificar: el puerto que va a utilizar el socket (*nSocketPort*) caso particular 161 en datagramas UDP para protocolo SNMP o 0 para lo cual la aplicación sola selecciona el puerto, el tipo de socket (*nSocketType*) que pueden ser **SOCK_STREAM** o **SOCK_DGRAM**(datagramas UDP), una máscara de eventos de red en la cual la aplicación está interesada recibir notificación(*lEvent*) (lectura, escritura conexión, cerrado, etc.), y la dirección de red a la cual conectar el socket diferenciada por notación de puntos (*lpszSocketAddress*) ejemplo: 192.168.22.8, o NULL que es el caso debido al uso que se le va a dar a la aplicación del socket.

El descriptor se almacena en una variable global previamente definida por la función llamada *m_hSocket,* y es ésta la que contiene el identificador para cada socket creado

2.4.4.1 Parámetros del Socket

1. **Familias de Protocolos**

   Indica la familia de protocolos que utiliza el programa. Por ejemplo, la constante simbólica AF_INET que identifica a la familia de protocolos TCP/IP al igual que PF_INET(las más utilizadas). Otras familias como PF_UNÍS y PF_NS que identifican a UNÍS y XNS respectivamente.

2. **Tipo de Comunicación**

   Es decir las orientadas a conexión y las no orientadas a conexión. Se emplean las constantes simbólicas **SOCK_STREAM** para flujo de bytes y **SOCK_DGRAM** para datagramas. La interfaz de sockets también define un socket básico **SOCK_RAW** (rawsocket). Donde el socket básico permite a un programa utilizar los mismos protocolos de nivel inferior que la red utiliza normalmente.

3. **Protocolo propiamente dicho.**

   En el conjunto de protocolos TCP/IP se tienen protocolos como: ICMP, UDP, IP y TCP, los cuales se pueden escoger para especificar la constante simbólica que empieza con el prefijo IPROTO_ . Así tenemos: IPROTO_TCP, IPROTO_ICMP, IPROTO_UDP, que especifican a cada protocolo.

De ésta manera se tiene que para especificar un socket ICMP para realizar la tarea de *ECHO* se utilizará:

SOCKET socket (AF_INET,SOCK_RAW,IPPROTO_ICMP)

Y para una aplicación de tipo SNMP:

BOOL Create(0,SOCK_DGRAM,NULL)

Donde posteriormente se especificará el puerto a ser utilizado mediante la función de conexión.

2.4.4.2 Selección de un Puerto.

En el caso de datagramas UDP que transportan información del protocolo SNMP, es necesario especificar el tipo de puerto que éste necesita para enviar o recibir mensajes desde o hacia un dispositivo administrable; y éste es el puerto 161, el cual se especificará en el momento de conexión del socket con su destino.

2.4.4.3 Descriptor de un Socket

Al crear un socket, el proceso devuelve un identificador o descriptor, que en realidad identifica un registro en la tabla de descripción.

Creado el socket se reserva un espacio de almacenamiento para la estructura del socket automáticamente, y el descriptor apunta a la estructura de datos interna del mismo y es a su vez un índice dentro de la tabla de descripción.

La estructura incluye elementos para almacenar valores de los parámetros de la función socket, no obstante, también contiene elementos como: IP local, IP remota, puerto local y puerto remoto.

2.4.4.4 Programación Winsock asíncrona[35]

Se utiliza la programación de sockets en modo asíncrono para que todo tipo de llamadas al receptor de mensajes, activen programas monohilo (subprogramas o rutinas invocados de forma concatenada) que hagan que winsock envíe o reciba llamadas en modo no bloqueante, manteniendo la interfaz de usuario a la escucha.

---

[35] *KRUGLINSKI, David; "PROGRAMACIÓN AVANZADA CON VISUAL C++6.0"; MacGrawHill, Madrid, España,1999.*

2.4.4.5 Selección de la Notificación Asincrónica

Una vez obtenido el descriptor, se establece un mensaje de notificación y se lo enlaza a la aplicación mediante un *handle*, para que cuando el socket reciba algún tipo de información, la aplicación reciba un mensaje de eventos de red del socket.

El mensaje es una constante simbólica y se define a partir del mensaje WM_USER(espacio reservado para mensajes propios de programa) y es recibido por la ventana en ejecución.

La función que permite ésta operación es:

int WSAAsyncSelect (SOCKET s, HWND hWnd, unsigned int wMsg long lEvent);

En la cual se deben especificar parámetros como: El socket para la cual la notificación es requerida es decir el descriptor(s), el *handle* identificando la ventana la cual recibirá el mensaje cuando el evento de red ocurra(hWnd), el mensaje a ser recibido(wMsg), una mascara especificando los eventos de red en los cuales la aplicación está interesada(lEvent) (eje: FD_READ|FD_WRITE).

2.4.4.6 Conexión de un socket

Un programa cliente orientado a conexión utiliza la función *connect* para configurar un socket de comunicación en red almacenando información en la estructura de datos del socket acerca de los extremos local y remoto. Requiere que se especifique un descriptor de socket, una estructura de dirección que contenga información sobre el anfitrión remoto y la estructura de dirección del socket.

Las funciones que se utilizan para conectar un socket con un destino son:

int connect (SOCKET s, const struct sockaddr FAR name, int namelen);*

BOOL Connect(LPCTSTR lpszHostAddress, UINT nHostPort);

En la primera se debe especificar: El socket para la cual la notificación es requerida es decir el descriptor(s), el nombre del socket a conectar(name), la longitud del anterior parámetro(namelen).
En la segunda función se debe especificar: la dirección de red del socket al cual éste objeto está conectado (lpszHostAddress)(eje: 128.56.22.8), el puerto de identificación de la aplicación del socket (nHostPort)(eje: 161). Siendo esta última función la que se utilizará para la conexión de tipo SNMP.

2.4.4.7 Envío y Recepción de Datos

Una vez creado el socket, y establecida la conexión para ciertos casos; se procede a enviar un paquete de información previamente creado.
Existen algunas funciones que permiten realizar dicha operación, pero las que se utilizarán en el programa son:

int sendto (SOCKET s, const char FAR * buf, int len, int flags, const struct sockaddr FAR * to, int tolen);

int Send(const void* lpBuf, int nBufLen, int nFlags = 0);

En la primera se debe especificar: el descriptor del socket(s), un buffer de memoria donde se contienen los datos a ser transmitidos(buf), la longitud de los datos en el buffer(len), un indicador especificando la manera en la cual la llamada está hecha(flags)(eje: 0), un puntero opcional a la dirección del socket objetivo(to), el tamaño de la dirección (tolen).
Ésta primera función se utilizará en la transmisión de paquetes ICMP.

En la segunda se debe especificar: el buffer que contienen los datos a ser transmitidos(lpBuf), la longitud de los datos contenidos en el buffer en bytes (nBufLen), especifica la forma en la cual la llamada está

hecha(nFlags)(general mente éste valor se lo pone en 0). Ésta función se utilizará para la transmisión de paquetes de información SNMP.

Para el caso de recepción de paquetes o datagramas, se utilizarán funciones como:

```
int recvfrom ( SOCKET s, char FAR* buf, int len, int flags, struct sockaddr FAR*
               from, int FAR* fromlen );

int ReceiveFrom( void* lpBuf, int nBufLen, CString& rSocketAddress, UINT&
                 rSocketPort, int nFlags = 0 );
```

En la primera se debe especificar valores como: el descriptor del socket del cual se quiere recibir información(s), un buffer para los datos de llegada(buf), la longitud del buffer(len), un indicador especificando la manera en la cual la llamada es hecha(flags), un puntero opcional que contiene la dirección de origen de retorno(from), un puntero opcional que con la longitud del buffer anterior(fromlen). Ésta función se utilizará para la recepción de mensajes del tipo ICMP.

En la segunda se especificará valores como: un buffer para los datos de entrada(lpBuf), la longitud del buffer en bytes(mBufLen), un referencia a un objeto de tipo CString que reciba la dirección IP en formato normal(rSocketAddress), referencia a una variable que almacene el puerto(rSocketPort), un indicador especificando la maneraen la cual la llamada es hecha(nFlags)(generalmente en 0). Ésta función se utilizará para recibir mensajes del tipo SNMP.

2.4.4.8 Cierre de un socket

Al terminar el envío y recepción de datos, se debe cerrar el socket creado a fin de terminar con la comunicación establecida y con la función de escucha que está a la espera de sucesos de red de los sockets. Esto se lo realiza con las siguientes funciones:

int closesocket (SOCKET s);
void Close( );

En las cuales se deben especificar valores como: descriptor del socket a cerrar(s), en la primera; o simplemente cerrando el socket en curso mediante la segunda función.

2.4.4.9 Funciones adicionales que contienen el uso de sockets.[36]

Éstas funciones de fácil uso, permiten conocer ya sea la dirección IP o el nombre de un *host* dentro de una red. Y éstas son:

GetHostByName : Ésta función estática llama a la función *gethostbyname* de Winsock. Consulta a un servidor de nombres y devuelve la dirección de conector correspondiente al nombre del host. Esta función expira por sí sola.

GetHostByAddress : Ésta función estática llama a la función *gethostbyaddr* de Winsock. Consulta a un servidor de nombres y devuelve el nombre de la máquina correspondiente a la dirección de conector. La función sufre el fin de temporización por sí sola.

Adicional a éstas funciones, también se tienen las del tipo asincrónico, como:

WSAAsyncGetHostByAddr : que tiene la misma funcionalidad que *gethostbyaddr* .

WSAAsyncGetHostByName : que tiene la misma funcionalidad que *gethostbyname.*

Salvo que éstas últimas envían mensajes de recepción, cuando se ha completado las peticiones asincrónicas.

Una función que permite obtener las direcciones MAC enviando mensajes de broadcast es:

---

[36] *MSDN Library Visual Studio 6.0a,, Copyright 1995 – 2000 Microsoft Corp.*

SendARP: perteneciente a la clase iphlpapi.h, y que obtiene la dirección MAC de un dispositivo local dado la dirección IP del mismo.

2.4.4.10 La interfaz NetBIOS

Se lo conoce como: Sistema básico de entrada y salida de red (NetBIOS). Interfaz de programación de aplicaciones (API) que pueden utilizar los programas en una red de área local (LAN). Proporciona a los programas un conjunto uniforme de comandos para solicitar los servicios de bajo nivel necesarios para administrar nombres, dirigir sesiones y enviar datagramas entre los nodos de una red.

2.5 ANÁLISIS DE LOS REQUERIMIENTOS DEL PROGRAMA

2.5.1 REQUERIMIENTOS GENERALES

Puesto que el objetivo del programa es graficar la topología de una red, se utiliza entonces un ambiente de ventanas; en el cual la ejecución del programa requiere que el equipo donde se realiza la aplicación, esté conectado a una red, y el ingreso de datos se hace en base a lo que requieren los protocolos de exploración como son ICMP y SNMP, en éste caso: direcciones IP y comunidades de lectura respectivamente, para posteriormente visualizar mediante iconos claramente diferenciados los diferentes elementos de una red.

El sistema operativo que permite tales herramientas es Windows, por lo que es la plataforma escogida, y el entorno Visual C++, la herramienta de programación ideal para el desarrollo de la aplicación.

2.5.2 REQUERIMIENTOS DEL INGRESO DE DATOS

El uso de protocolo SNMP conlleva al uso de comunidades tanto de lectura como de escritura en los equipos agentes administrables, por lo que es uno de los datos de ingreso para la exploración.

En el inicio del programa se requiere de la dirección IP actual del computador, para a partir de la subred de donde se encuentra, comenzar la exploración de los elementos presentes en dicha subred y adicionalmente de los elementos presentes en las subredes adyacentes; si así se desea.

Todo esto se lo realiza mediante el uso de cuadros de diálogo con campos específicos para los diferentes tipos de datos a ingresar; tanto para el programa principal; así como para las herramientas de elementos administrables, utilizando el mismo esquema de ingreso de datos ya mencionado.

Para el caso el consultas MIB, se dispone de un árbol de MIBs con OIDs organizadas en grupos jerárquicos ya definidos, de donde se podrá escoger objetos particulares SNMP que se desee consultar.

2.5.3 REQUERIMIENTOS DE SALIDA DE DATOS

Una vez terminada la exploración se necesita una presentación visual y en forma de lista de los componentes encontrados, organizados por tipo de dispositivo con sus respectivas características.

La visualización de la topología se lo hace en un espacio diseñado para exponer gráficos de red, permitiéndose una fácil exploración del mismo con ayuda de una lista organizada de elementos. A su vez se cuenta con herramientas que faciliten el monitoreo y estado de un elemento presente en la red, exponiéndose los resultados en campos destinados para ello.

2.6 DISEÑO FUNCIONAL DEL PROGRAMA

Para el desarrollo del programa se ha definido la siguiente organización estructural (figura 2.5) en cuanto a: entradas, proceso y salida de datos.

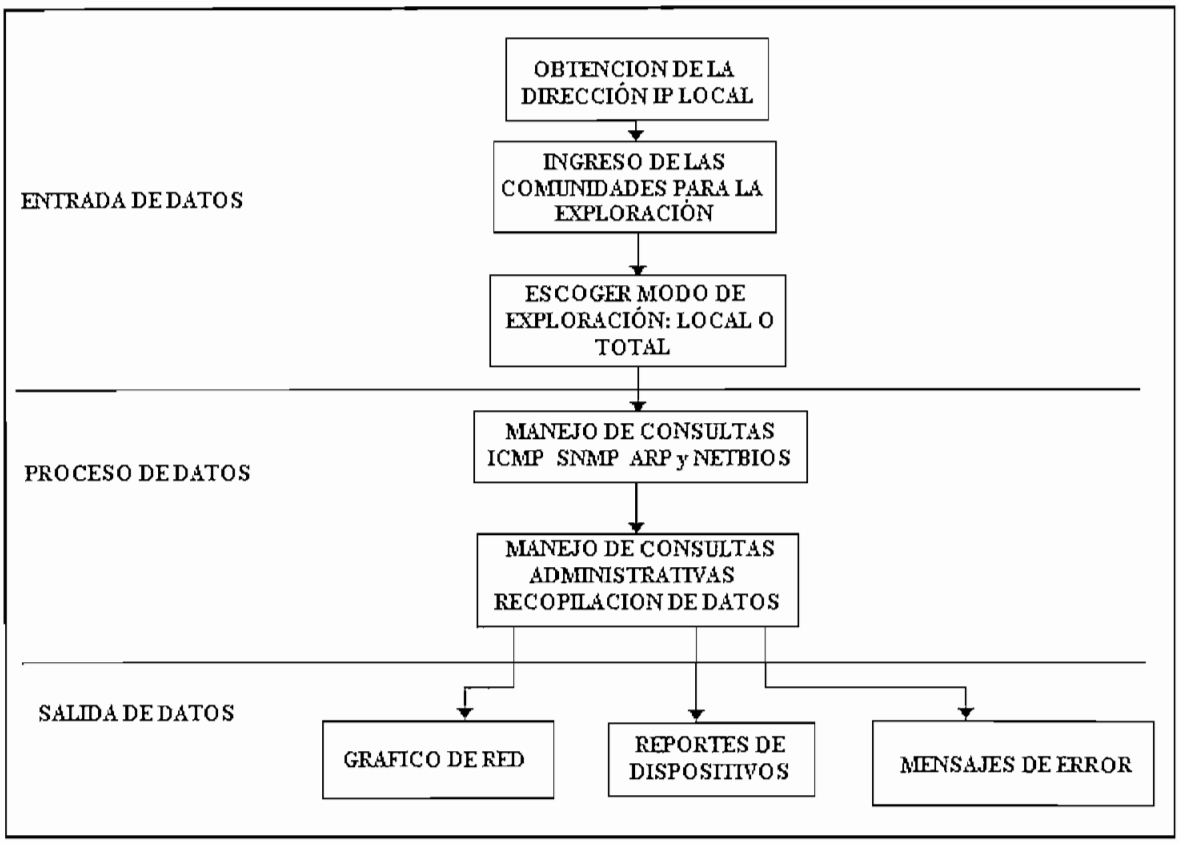


Figura 2.5 Esquema del Diseño Funcional del Programa

2.6.1 DETALLE DE ENTRADA DE DATOS

Los requerimientos para entrada de datos se presentan a continuación:

- ✓ ***Obtención de la Dirección IP local:*** es un dato que automáticamente se obtiene al encender el programa de exploración, puesto que se necesita que el computador esté conectado a una red para empezar dicho proceso.

- ✓ ***Ingreso de las comunidades para la exploración:*** por omisión ya se encuentran comunidades como: public o private, definidas en la mayoría de componentes administrables como predefinidas. Son datos que se ingresan en el campo respectivo de haber otras comunidades de lectura presentes en los dispositivos administrables.

- ✓ ***Escoger modo de exploración Local o Total:*** por omisión la opción es de exploración total, y significa que se explorará mediante datagramas ICMP(*echo*) dispositivos presentes en la subred presente para luego explorar las subredes adyacentes. La opción de exploración local establece solo una exploración de la subred donde el computador se encuentra presente.

2.6.2 DETALLE DEL PROCESO DE DATOS

Para el bloque de proceso de datos se ha definido de manera global el desarrollo de las operaciones que el programa debe realizar, y estas son:

- ✓ ***Manejo de consultas ICMP, SNMP, ARP y NetBIOS:*** Se realiza la exploración de la subred, usando primero mensajes de *echo(ICMP)*, luego de lo encontrado, mensajes de *ARP* para conocer las direcciones MAC de los componentes; posteriormente se envía mensajes *SNMP* con todas las comunidades ingresadas para establecer dispositivos administrables dentro de la subred en exploración, y por último mensajes de tipo *UDP*(NetBIOS), propios de Windows, y que permiten conocer el nombre de un computador o un dispositivo.

- ✓ ***Manejo de Consultas Administrativas Recopilación de Datos:*** Donde se clasifican primero los dispositivos encontrados, se establece un dispositivo principal(de haberlo), se diferencian entre switches, ruteadores, impresoras y otros; para posteriormente recopilar los datos que permitan esquematizar la topología de la subred encontrada.

Una vez terminado el proceso, y si la opción elegida fue de Exploración Total, se repite todo este bloque hasta culminar con todos los elementos de las subredes adyacentes y por ende de la red.

2.6.3 DETALLE DE SALIDA DE DATOS

Una vez terminada la exploración se presentan los datos, de la siguiente manera:

- ✓ ***Gráfico de la Red:*** se presenta la topología de la red utilizando iconos que diferencien unos dispositivos de otros, unidos por líneas punteadas cada uno hacia su centro respectivo, y divididos por mapas de subred a la que pertenecen. Adicionalmente se cuenta con un explorador de dispositivos a manera de árbol el cual permite un acceso más sencillo a un dispositivo en particular.

- ✓ ***Reportes de Dispositivos:*** se muestran en un menú adicional en la barra de menús, y cada uno despliega una lista a manera de reporte de los componentes encontrados y clasificados por tipo (computadores, impresoras, switches, ruteadores, etc), con sus respectivos detalles.

- ✓ ***Mensajes de Error:*** durante el proceso de exploración, se mantiene un visualizador de sucesos que permite observar el desarrollo del proceso de escaneo; y de existir errores, mensajes indicando los mismos. Adicionalmente y mediante mensajes de ventana propios de Windows, una vez terminada la exploración; de ser el caso, se presentan mensajes de error indicando los errores en un determinado procedimiento.

2.7 DISEÑO DE CLASES Y DESARROLLO DEL PROGRAMA

El ambiente de programación Visual C++ permite crear aplicaciones basadas en clases de programación y a manera de ventanas de Windows; con menús desplegables, menús contextuales, cuadros de diálogo, y cuadros de herramientas; facilitando de ésta manera el diseño de una

aplicación funcional con un sinnúmero de herramientas amigables para el usuario final.

Uno de estos instrumentos a manera de clase es la aplicación de sockets mediante clases basadas en ellos, que permiten el envío y recepción de mensajes a través de la red, creando además un conjunto de estructuras definidas, que ayudan a determinar cada uno de los campos pertenecientes a estos mensajes, como por ejemplo: el tipo de mensaje recibido o la forma de llenado del paquete a enviar.

Se puede crear clases personalizadas derivadas de las clases CSocket.h y CAsyncSocket.h, orientadas al Protocolo de Mensajes de Control de Internet(ICMP) y al Protocolo Simple de Administración de Red(SNMP) para utilizarlos en la aplicación.

Debido a que la aplicación usa una ventana principal y ventanas de diálogo para manejo de datos, hereda funcionalidad de las clases CFormView y CDialog las cuales permiten el ingreso y salida de datos, así como la visualización del esquema con la topología de red encontrada.

La aplicación debe soportar las funciones de las Clases Socket , de Diálogo y de Vista Principal para estar en capacidad de heredar las funciones definidas dentro de estas clases, y como la aplicación se orienta al uso de Protocolos ICMP y SNMP, encapsularemos clases que hereden dichas funciones y que permitan un manejo personalizado de los datos a enviar y recibir dentro de la red.

A continuación se presentan algunas de las clases principales desarrolladas para la implementación de la aplicación; mediante la tabla 2.1:

Tabla 2.1. Clases desarrolladas para su uso en la aplicación.

Clase	Clase Base	Detalle
Clcmp	CSocket	Esta clase implementa métodos para la emisión y recepción de datagramas ICMP
CAsySNMP	CAsyncSocket	Esta clase implementa métodos para la emisión y recepción de datagramas SNMP
CTotalScan	CDialog	Esta clase implementa métodos y funciones para el manejo de datos y se encarga de la exploración de la red, utiliza las anteriores clases.
CNetGraphGoldView	CFormView	Esta clase implementa métodos y funciones de ventana principal. Contiene las herramientas de exploración, construcción del gráfico y del visor de gráfico de topología.
CDibujo	CStatic	Esta clase implementa métodos y maneja la forma gráfica del esquema de la topología de red.

Existen otras clases derivadas de CDialog y que se utilizan como herramientas de medición como: Ping, Tracert, DNSResolver, Consultor SNMP, Medidores, etc; y que contienen los mismos principios básicos de programación de las clases principales ya mencionadas.

2.7.1 DESCRIPCIÓN DE LIBRERÍAS UTILIZADAS EN LA APLICACIÓN

"stdafx.h": Se incluye en los archivos o proyectos del sistema, para establecer información relacionada con la aplicación a través de componentes estándares, como clases, extensiones, sockets, etc.

"iphlpapi.h": Ésta cabecera incluye funciones de consulta de tipo ARP para extraer información como: direcciones MAC. Su uso se basa en funciones de la librería Winsock2.

"Winsock2.h": Los programas de winsock, o de sockets de Windows, requieren una biblioteca de enlace dinámico DLL llamada Winsock.dll (en el caso de las aplicaciones para 32 bits, la biblioteca se denomina Wsock32.dll), que se incluye en la API Windows Sockets. el compilador Visual C++ necesita una biblioteca Winsock de importación para los programas basados en Windows Sockets, por medio de winsock.h, la que a su vez incluye la cabecera windows.h.

"math.h": Ésta cabecera incluye funciones matemáticas útiles para cálculos complejos como seno, coseno, etc; y que permiten optimizar la funcionalidad del programa.

2.7.2 DISEÑO DE LA CLASE CICMP

El uso del Protocolo ICMP para transmisión de paquetes de información, tales como ping, sugiere el uso de un socket no conectado; por lo que se crea una clase que contenga los métodos y estructuras necesarias para transmitir y recibir dichos datagramas.

2.7.2.1 Encapsulamiento

Como se necesita generar paquetes de *echo* para ping, y con respuesta variable TTL(*echo también*), para *tracert* ; solo nos concentraremos en generar y colectar paquetes de éste tipo, y por lo tanto se encapsula en la clase las funciones que ayuden a generar y recuperar dichos paquetes.

2.7.2.2 Herencia

La clase CIcmp hereda funcionalidad de un objeto de la clase CSocket, por lo que un objeto perteneciente a la clase CIcmp tiene toda la funcionalidad de un objeto de CSocket, pero orientado al protocolo ICMP, y más específicamente al envío y recepción de paquetes ping.

2.7.2.3 Creación de la Clase CIcmp[37]

Se define como requerimiento principal la creación de un socket básico para la comunicación en la red. Se empieza entonces la clase orientada al uso del protocolo ICMP, con un método que cree un socket y lo enlace a una notificación de tipo asincrónica.

El paquete de *echo (ping)* a enviarse necesita de un método que arme los campos: tipo, código, suma de comprobación, llenado de buffer de envío de datos con la estructura correspondiente al mensaje y en el cual adicionalmente se especifique el *host* de destino.

Para la recepción del paquete respuesta (*echo reply*), se necesita de un método que obtenga información de respuesta como: tipo de paquete, dirección IP del emisor y un buffer de recepción para el mensaje.

La suma de comprobación, se la realizará como se recomienda dentro de la RFC 1071, implementándose como un método para llenar los datos de cabecera del paquete de *echo.*

Completado el proceso de envío y recepción, se tiene un método que libere el socket utilizado.

---

[37] *FUENTES SAMANIEGO, Fausto Raúl "Generador y colector de paquetes ICMP",EPN , Marzo 2001.*

2.7.2.4 Creación de un socket básico para utilizarse en ICMP

Éste método tiene por objetivo el abrir un socket básico dentro del nivel de capa de red para utilizarlo con el Protocolo de Mensajes de Control e Internet.

2.7.2.4.1 Algoritmo utilizado

Se resume de la siguiente forma:

- ✓ Creación del socket de tipo TCP/IP orientado al uso de protocolo ICMP.
- ✓ Notificación de falla de creación.
- ✓ Enlace a una notificación de tipo asincrónica.
- ✓ Notificación de falla de enlace.

2.7.2.4.2 Diagrama de Flujo

En la figura 2.6 se indica el diagrama de flujo del método creado.

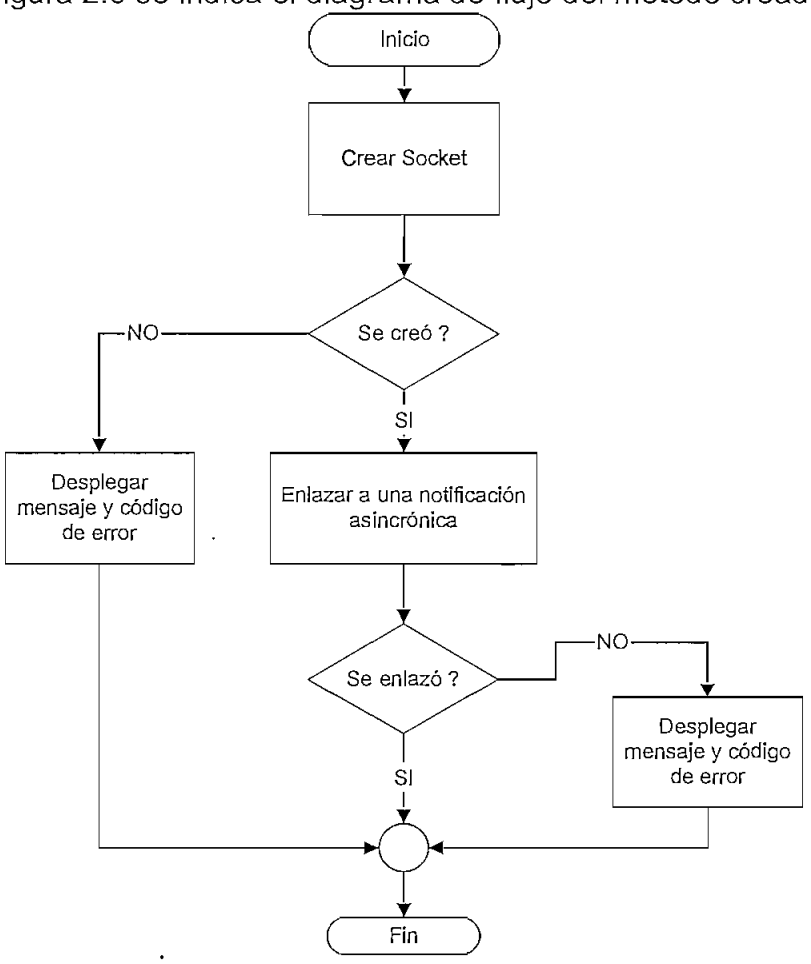


Figura 2.6 Diagrama de Flujo de la creación de un socket ICMP.

2.7.2.4.3 Descripción del método y funciones utilizadas[38]

Para la creación del método se utilizaron las siguientes funciones propias de la clase CSocket:

- ✓ Se abre la conexión de un socket especificando la función *socket* con los parámetros para TCP/IP y protocolo ICMP.
- ✓ Se utiliza la función *setsockopt* para cambiar opciones de socket.
- ✓ Se utiliza la función *WSAAsyncSelect* para enlazar el descriptor del socket creado a una notificación asincrónica de red.
- ✓ Se retorna mensaje de proceso completado o fallido, dependiendo del caso.

2.7.2.5 Envío de una petición de echo (ping)

Para realizar el envío de datos, se crea una cabecera del paquete ICMP, un buffer de datos y se especifica un destino.

2.7.2.5.1 Algoritmo utilizado

Se resume de la siguiente forma:

- ✓ Se crea un buffer tanto de envío.
- ✓ Se llena el buffer con el mensaje a ser enviado.
- ✓ Se especifica en la cabecera: el tipo, código, id, secuencia, tiempo de vida (TTL)(aquí el cambio para *trecert*), suma de comprobación (*Checksum*), del paquete a enviarse(*ping*).
- ✓ Se envía el datagrama por el socket no conectado a una dirección IP específica.
- ✓ Notificación de falla de envío.

[38] *MSDN Library Visual Studio 6.0a,, Copyright 1995 – 2000 Microsoft Corp.*

2.7.2.5.2 Diagrama de Flujo

En la figura 2.7 se indica el diagrama de flujo del método creado.

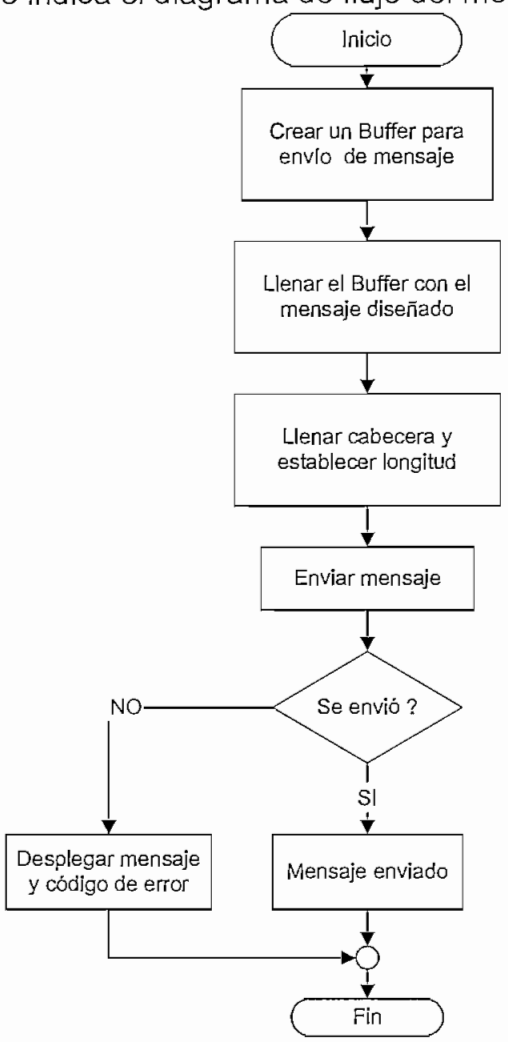


Figura 2.7 Diagrama de Flujo del envío de un mensaje ping.

2.7.2.5.3 Descripción del método y funciones utilizadas[39]

Para la creación del método se utilizaron las siguientes funciones:

- ✓ Se crea el buffer para los datos con capacidad de 64 bytes.
- ✓ Se lo llena con un mensaje, en éste caso el mensaje es: CarlosContrerasG repetido 4 veces.

[39] *MSDN Library Visual Studio 6.0a,, Copyright 1995 – 2000 Microsoft Corp.*

- ✓ Se complementa la cabecera con el tipo de paquete(*echo request* 0x08H), checksum, secuencia ,tiempo de vida TTL(depende del mensaje: ping o tracert), etc.
- ✓ Se envía el mensaje utilizando la función *sendto.*
- ✓ Se establece mensaje de éxito o fracaso, dependiendo del caso.

2.7.2.6 Suma de comprobación[40]

Se toma la forma de cálculo descrito por los estándares de protocolo IP, TCP, y la recomendación RFC 1071.

2.7.2.6.1 Algoritmo utilizado

En resumen el algoritmo de suma de comprobación es simple:

- ✓ Los octetos adyacentes a los que se aplicará la suma de comprobación se juntan en ares para formar enteros de 16 bits.

- ✓ Para generar la suma de comprobación, el campo que almacena la suma en si es borrado, se calcula el complemento a 1 de la suma sobre los octetos, y el complemento a 1 de esta suma se coloca en el campo suma de comprobación.

- ✓ Para revisar una suma de comprobación, se calcula la suma complementada a 1 sobre el mismo conjunto de octetos, incluido el campo suma de comprobación. Si el resultado es todo 1', la comprobación está realizada.

2.7.2.6.2 Diagrama de Flujo

En la figura 2.8 se indica el diagrama de flujo del método creado.

---

[40] *FUENTES Fausto; "Generador y Colector de paquetes ICMP",EPN , Marzo 2001.*

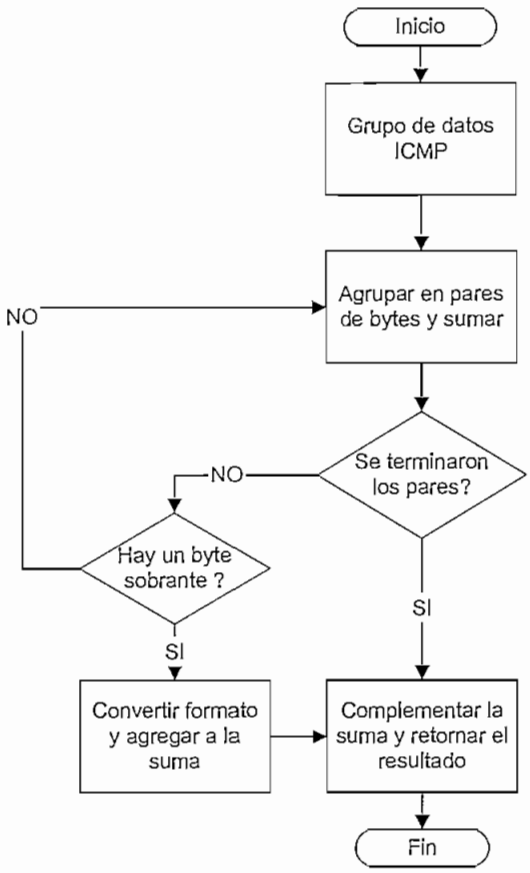


Figura 2.8 Diagrama de Flujo del proceso de la suma de comprobación.

2.7.2.6.3 Descripción del método y funciones utilizadas.

El método a implementarse devuelve la suma de comprobación del grupo de datos correspondientes al mensaje ICMP escritos en un buffer de datos. Se recupera este valor en un número entero largo de 32 bits por la magnitud a obtenerse de la suma.

Es necesario a éste método proporcionar un puntero al buffer que contiene los datos, a los que se requiere aplicar la operación, así como la longitud del buffer. Al final se tiene la suma de comprobación complementada a 1.

2.7.2.7 Recepción de un mensaje ICMP (echo replay)

Éste método tiene por objetivo el recibir datos desde un puerto perteneciente a un socket no conectado utilizado para captar mensajes del protocolo ICMP, y recuperarlos a un buffer destinado para ésta operación.

2.7.2.7.1 Algoritmo utilizado

Se resume de la siguiente manera:

- ✓ Se crea un buffer que de tamaño apropiado para la totalidad de datos a recibir.
- ✓ Se llena el buffer con los datos y la estructura del socket recibido con los datos del emisor del mensaje.
- ✓ Se comprueba la recepción.
- ✓ Se establece el tipo de mensaje recibido.
- ✓ Se retorna el resultado obtenido.

2.7.2.7.2 Diagrama de Flujo

En la figura 2.9 se indica el diagrama de flujo del método creado.

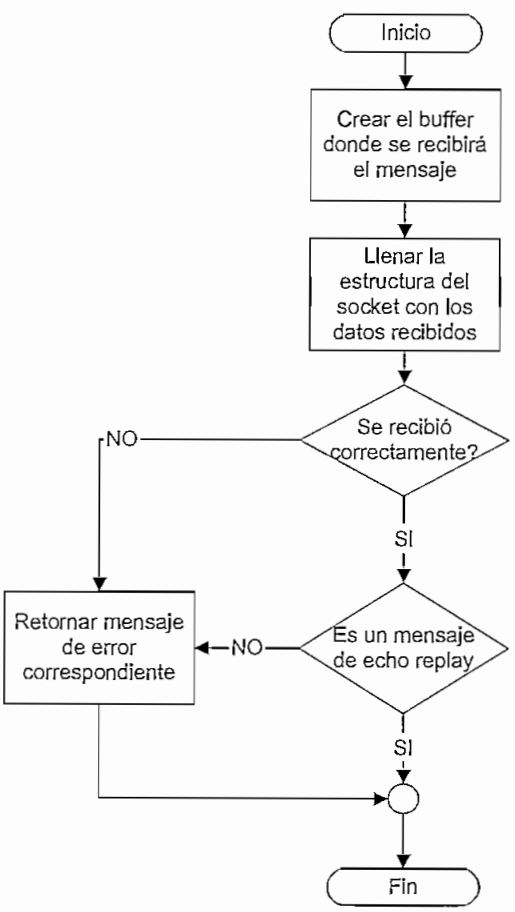


Figura 2.9 Diagrama de Flujo del proceso de recepción de un mensaje echo replay.

2.7.2.7.3 Descripción del método y funciones utilizadas.

Para la creación del método se utilizaron las siguientes funciones:

- ✓ Se crea un buffer de recepción con capacidad de 2064 bytes. Buffer recomendado para recepción de datos ICMP.
- ✓ Se receptan los datos desde el puerto recibido y se llena el la estructura del socket con los datos del emisor utilizando la función *recvfrom.*
- ✓ Se realiza la comprobación respectiva del mensaje para determinar si el estado de recepción fue bueno.
- ✓ Se determina si el tipo de mensaje que ha llegado corresponde con la constante literal ICMP_ECHOREPLY, predeterminada para mensajes de respuesta de echo.
- ✓ Se retorna el resultado obtenido.

2.7.2.8 Cierre del socket ICMP

Una vez terminada la operación de envío y recepción de mensajes ICMP, se procede a liberar recursos del sistema, cerrando el socket en uso.

2.7.2.8.1 Algoritmo utilizado

Se resume de la siguiente manera:

- ✓ Se especifica el socket a cerrar mediante el descriptor previamente almacenado
- ✓ Se cierra el socket apuntado.
- ✓ Se determina el estado final de la operación mediante un mensaje de estado.

2.7.2.8.2 Diagrama de Flujo

En la figura 2.10 se indica el diagrama de flujo del método creado.

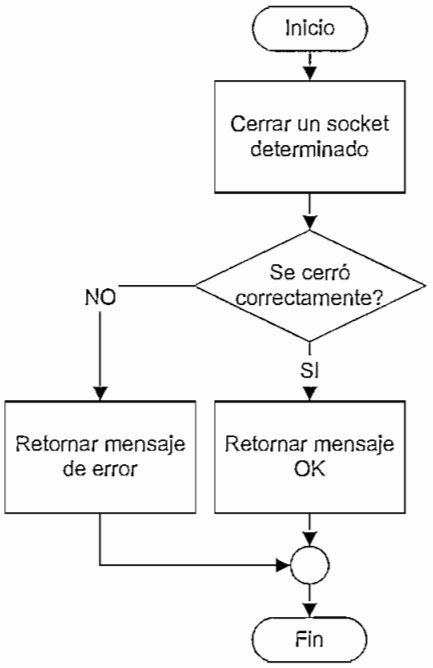


Figura 2.10 Diagrama de Flujo de cierre de un socket

2.7.2.8.3 Descripción del método y funciones utilizadas.

Para la creación del método se utilizaron las siguientes funciones:

- ✓ Se toma el descriptor del socket que se desea cerrar, previamente identificado.
- ✓ Se cierra el socket utilizando la función *closesocket* propia de la clase creada.
- ✓ En caso de error, se retorna un mensaje con el código de error ocurrido.

2.7.3 DISEÑO DE LA CLASE ASYSNMP

El uso del Protocolo SNMP para envío y recepción de mensajes de administración sugiere el uso de un socket que utilice éste servicio, y conectado al puerto 161 predeterminado para el uso de éste protocolo; por

lo que se crea una clase que contenga los métodos y estructuras necesarias para transmitir y recibir tales datagramas.

2.7.3.1 Encapsulamiento

Como se necesita generar datagramas UDP para transmisión de paquetes SNMP; solo nos concentraremos en generar y colectar paquetes de éste tipo, sin distinción del mensaje recibido; puesto que depende de la aplicación donde se utilice el objeto perteneciente a esta clase, para decodificar de manera personalizada el mensaje recibido. Por lo tanto se encapsula en la clase las funciones que ayuden a generar y recuperar dichos paquetes.

2.7.3.2 Herencia

La clase AsySNMP hereda funcionalidad de un objeto de la clase CAsyncSocket, por lo que un objeto perteneciente a la clase AsySNMP tiene toda la funcionalidad de un objeto de CAsyncSocket, pero orientado al envío y recepción de datagramas UDP que encapsulan mensajes de tipo SNMP.

2.7.3.3 Creación de la Clase AsySNMP

Se define como requerimiento principal la creación de un socket básico para la comunicación en la red orientado a la transmisión y recepción de datagramas UDP. Más específicamente al uso del protocolo SNMP, por lo que se crea la clase con un método que cree un socket, lo enlace a una notificación de tipo asincrónica, y lo conecte al puerto 161 propicio para el uso del protocolo mencionado.

El paquete de SNMP a enviarse necesita de un método que reciba la dirección especificando el *host* de destino con el que se desea establecer comunicación, un buffer para el envío de datos con la estructura correspondiente al mensaje y la longitud del mismo.

Para la recepción del mensaje de respuesta, se utilizan las funciones propias de la clase base, y se almacenan los datos de recepción en un buffer determinado para su posterior análisis; dependiendo de la aplicación que esté utilizando el objeto.

Completado el proceso de envío y recepción de datos, se utilizan nuevamente las funciones propias de la clase base para cerrar el socket, de manera que solamente se especifique el descriptor del mismo para así utilizar la función más adecuada.

2.7.3.4 Creación de un socket básico para utilizarse en SNMP

Éste método tiene por objetivo el abrir un socket básico para utilizarlo con el Protocolo Simple para Administración de la Red encapsulándolo en datagramas UDP.

2.7.3.4.1 Algoritmo utilizado

Se resume de la siguiente forma:

- ✓ Creación del socket de tipo TCP/IP orientado al uso de protocolo SNMP.
- ✓ Notificación de falla de creación.
- ✓ Enlace a una notificación de tipo asincrónica.
- ✓ Notificación de falla de enlace.

2.7.3.4.2 Diagrama de Flujo

En la figura 2.11 se indica el diagrama de flujo del método creado.

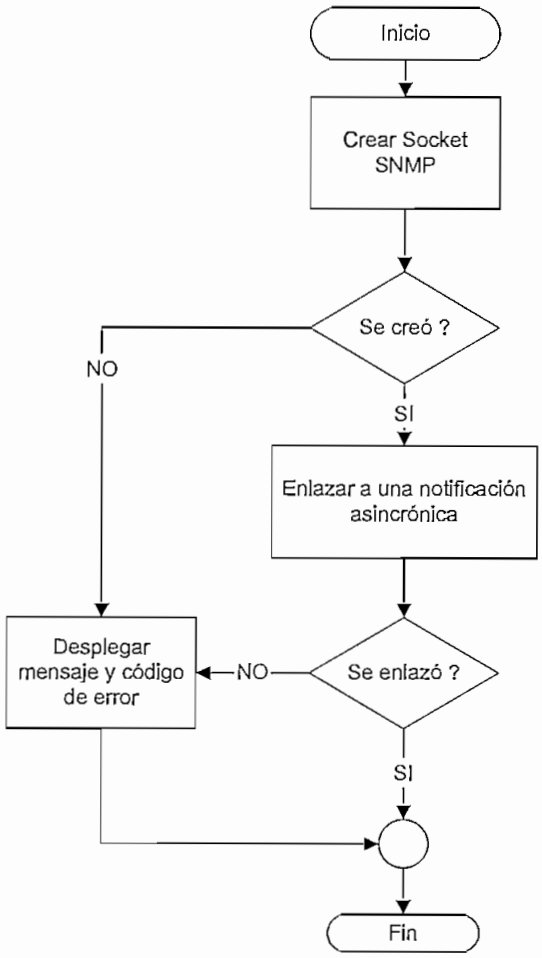


Figura 2.11 Diagrama de Flujo de la creación de un socket SNMP.

2.7.3.4.3 Descripción del método y funciones utilizadas.

Para la creación del método se utilizaron las siguientes funciones propias de la clase CAsyncSocket:

- ✓ Se abre la conexión de un socket especificando la función *Create (2.4.4 CREACIÓN DE UN SOCKET)* con los parámetros para TCP/IP y protocolo de datagramas, y obtener el descriptor en *m_hSocket*.

- ✓ Se utiliza la función *WSAAsyncSelect*[41] para enlazar el descriptor del socket creado a una notificación asincrónica de red.

- ✓ Se retorna mensaje de proceso completado o fallido, dependiendo del caso.

2.7.3.5 Envío de un mensaje de petición SNMP

Para realizar el envío de una petición SNMP, se establece una dirección de destino, se conecta con el puerto 161 para uso del protocolo SNMP y se ensambla un datagrama con los campos correspondientes al paquete de petición para transmitirlo a través de la red.

2.7.3.5.1 Algoritmo utilizado

Se resume de la siguiente forma:

- ✓ Se crea un buffer para el envío de datos.
- ✓ Se construye el datagrama de petición para el protocolo SNMP (versiones 1 o 2).
- ✓ Se llena el buffer con el mensaje a ser enviado.
- ✓ Se conecta el socket al puerto 161.
- ✓ Se envía el datagrama por el socket a una dirección IP específica.
- ✓ Se retorna mensaje de proceso completado o fallido.

2.7.3.5.2 Diagrama de Flujo

En la figura 2.12 se indica el diagrama de flujo del método creado.

---

[41] *MSDN Library Visual Studio 6.0a,, Copyright 1995 – 2000 Microsoft Corp.*

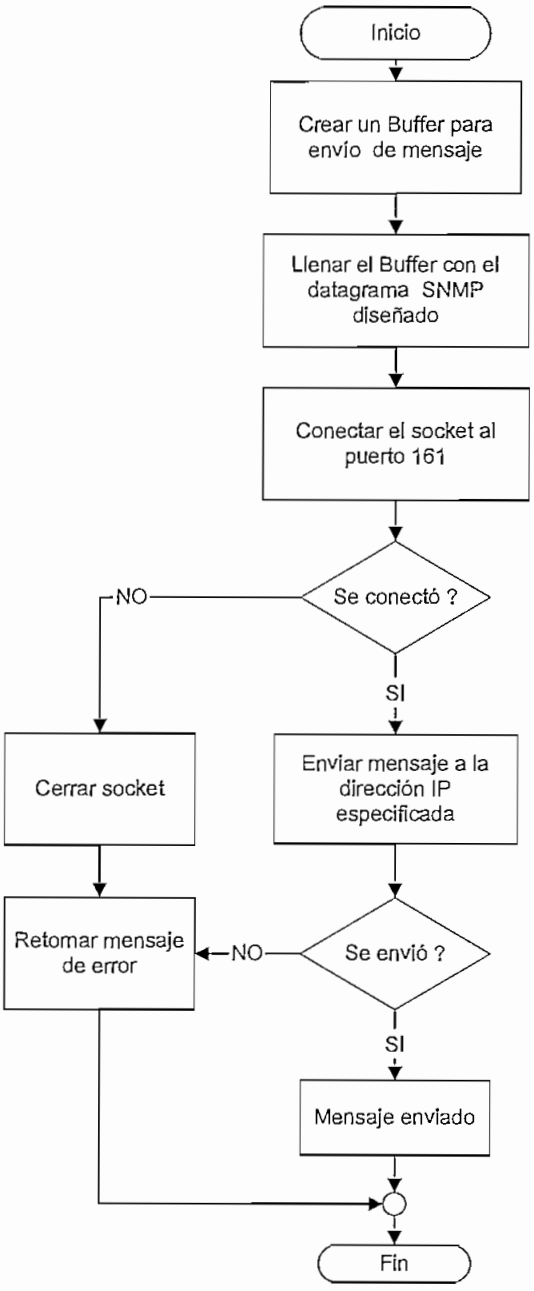


Figura 2.12 Diagrama de Flujo del envío de un datagrama SNMP.

2.7.3.5.3 Descripción del método y funciones utilizadas.

Para la creación del método se utilizaron las siguientes funciones:

- ✓ Se crea el buffer para los datos.
- ✓ Se lo llena con un mensaje SNMP, en éste caso el mensaje depende del tipo de petición que se desea hacer en la aplicación: GetRequest, GetNextRequest, etc.

- ✓ Se conecta el socket al puerto 161 y se establece la dirección IP de destino mediante el uso de la función *Connect*.
- ✓ Se envía el mensaje utilizando la función *Send.*
- ✓ Se establece mensaje de éxito o fracaso, dependiendo del caso.

2.7.3.6 Recepción de un mensaje SNMP

Éste se lo realiza directamente, ocupando la función ReceiveFrom propia de la clase base CAsyncSocket y heredada de la misma.

2.7.3.6.1 Algoritmo utilizado

Se resume de la siguiente manera:

- ✓ Se crea un buffer que de tamaño apropiado para la totalidad de datos a recibir.
- ✓ Se llena el buffer con los datos, la dirección IP y el puerto recibidos del emisor del mensaje.
- ✓ Se comprueba la recepción.
- ✓ Se retorna el resultado obtenido.

2.7.3.6.2 Diagrama de Flujo

En la figura 2.13 se indica el diagrama de flujo del método creado.

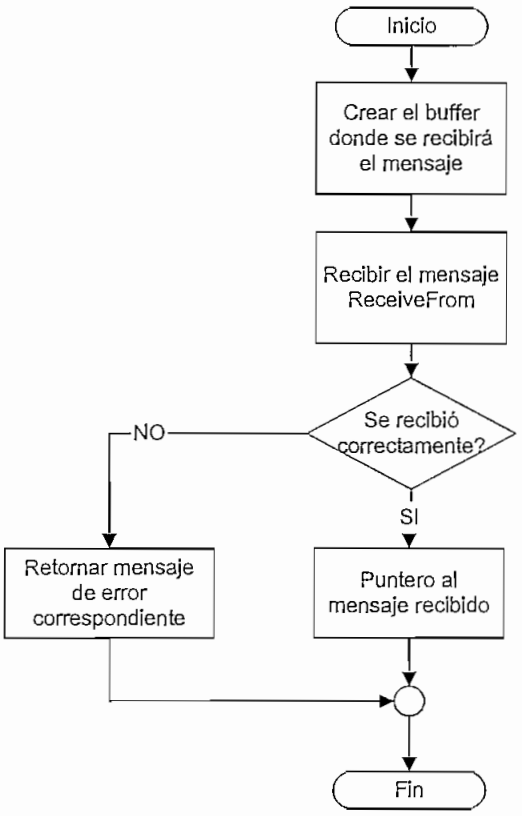


Figura 2.13 Diagrama de Flujo de la recepción de un datagrama SNMP.

2.7.3.6.3 Descripción del método y funciones utilizadas.

Para la creación del método se utilizaron las siguientes funciones:

- ✓ Se crea un buffer de recepción con capacidad de 500 bytes. Buffer apropiado para recepción de datos SNMP.

- ✓ Se receptan los datos desde el puerto recibido y se almacenan los datos del emisor utilizando la función *ReceiveFrom.*

- ✓ Se realiza la comprobación respectiva del mensaje para determinar si el estado de recepción fue bueno.

- ✓ Se retorna el resultado obtenido.

2.7.3.7 Cierre del socket SNMP

Una vez terminada la operación de envío y recepción de mensajes SNMP, se procede a liberar recursos del sistema, cerrando el socket en uso, mediante el uso directo de la función de cierre heredada de la calse base.

2.7.3.7.1 Algoritmo utilizado

Se resume de la siguiente manera:

- ✓ Se especifica el socket a cerrar mediante el descriptor, en ese caso automáticamente almacenado.
- ✓ Se cierra el socket apuntado.
- ✓ Se determina el estado final de la operación mediante un mensaje de estado.

2.7.3.7.2 Diagrama de Flujo

En la figura 2.14 se indica el diagrama de flujo del método creado.

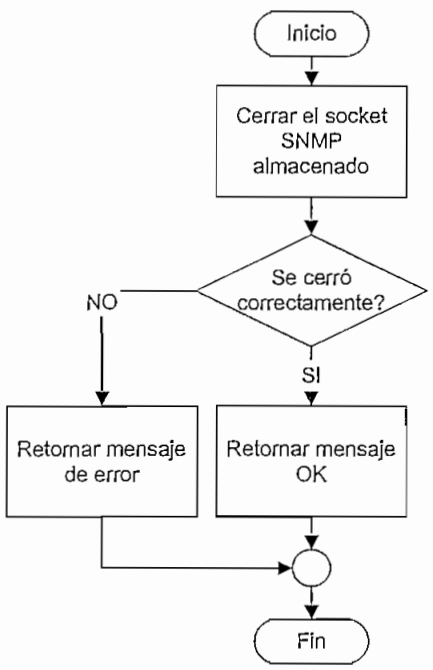


Figura 2.14 Diagrama de Flujo del cierre de un socket SNMP.

2.7.3.7.3 Descripción del método y funciones utilizadas.

Para la creación del método se utilizaron las siguientes funciones:

- ✓ Se toma el descriptor del socket que se desea cerrar, para el caso de SNMP, automáticamente almacenado en la variable *m_hSocket* parte de la estructura del socket previamente creado.
- ✓ Se cierra el socket utilizando la función *Close* propia de la clase creada.
- ✓ En caso de error, se retorna un mensaje con el código de error ocurrido.

2.7.4 DISEÑO DE LA CLASE CTOTALSCAN

La exploración de la subred o subredes a graficar, necesita del uso de las clases anteriormente construidas, y como los métodos ya especificados permitirán el envío y recepción de datagramas para los protocolos ICMP y SNMP, construimos una clase basados en ellos, y un procedimiento adecuado para almacenaje de datos y esquematización de la red, por lo que la clase a crearse pasa a ser uno de los pilares más importantes en e la construcción de la aplicación.

Adicionalmente se hace uso de herramientas para complementar la exploración tales como: protocolo ARP y mensajes de identificación NetBIOS, propicios para la exploración de direcciones MAC y nombres de los *hosts* sobre sistemas operativos que soporten este tipo de peticiones, para así facilitar la clasificación de los elementos encontrados.

2.7.4.1 Encapsulamiento

Como se necesita generar datagramas ICMP y SNMP para la exploración de una subred específica; nos concentraremos en generar y colectar paquetes de éste tipo, utilizando las clases Clcmp y AsySNMP ya creadas.

Se crean rutinas que encuentren todos los elementos presentes dentro de una subred, y dentro de un rango de 254 subredes diferentes, que vendrían a completar una red.

La clase a crearse encapsulan los objetos pertenecientes a las clases derivadas de sockets(Clcmp y AsySNMP), ocupando sus funciones de envío y recepción de datos; adicionalmente encapsulan funciones que permitan distinguir entre las diferentes respuestas a los mensajes de petición de SNMP, para conocer información importante de los elementos administrables encontrados.

2.7.4.2 Herencia

La clase CTotaScan hereda funcionalidad de un objeto de la clase CDialog, por lo que un objeto perteneciente a la clase CTotaScan tiene toda la funcionalidad de un objeto de CDialog, pero orientado a la exploración de elementos de red usando protocolos ICMP y SNMP, utilizando las clases Clcmp y AsySNMP creadas para ello.

2.7.4.3 Creación de la Clase CTotalScan

Debido a que ésta es una de las clases medulares en el desarrollo de la aplicación en general, se definen como requerimientos la creación de una interfaz visual que permita ingresar datos para la exploración de la subred o subredes, la exploración misma mediante la transmisión y recepción de datagramas ICMP y SNMP, la clasificación de elementos y recopilación de datos que permitan esquematizar la red explorada; para lo cual se crea la clase con los métodos apropiados que permitan lograr éste objetivo.

Debido a que la creación de ésta clase conlleva el uso de otras herramientas para la exploración y ésta a su ves se define de algunos procesos; se la ha dividido en tres partes principales que facilitan la conceptualización de lo que ésta clase debe contener. Éstas partes son:

- ✓ El ingreso de datos, donde se establecerá cuales son los datos necesarios para empezar la exploración de la subred.
- ✓ La exploración de la red, clasificación de los elementos y recopilación de datos de elementos administrables, los cuales nos dirán de forma más precisa como está conformada la red.
- ✓ La organización de todos los datos recogidos, para la construcción de tablas de información de la red explorada, y una tabla base general que contenga información necesaria para su graficación.

La clase por lo tanto necesita del uso de otras clases donde se puedan almacenar de forma ordenada los datos que se están recopilando, así como de una clase que se encargue de darles ese orden en el momento justo de la recopilación. En vista de ello se ayuda de clases como: CTableManage para el manejo de las tablas, y CComputers, CPrinters, CRouters, CSwitchLayer2, CSwitchLayer3, COtherDevices, para el almacenaje de la información; por lo que éstas poseerán funciones básicas y rutinas repetitivas no complejas que ayuden con ésta labor.

Se explica de forma global como éstas clases interactúan con la clase en desarrollo, por lo que se detalla en su momento que hace cada una de éstas clases.

Debido a la extensión de toda la operación de exploración definida, se resumieron todos los procesos que contiene la clase en el diagrama de flujo que viene a continuación.

2.7.4.3.1 Diagrama de Flujo General

La explicación más detenida sobre cada proceso se la detalla en los puntos correspondientes más adelante.

En la figura 2.15 se indica el diagrama de flujo del método creado.

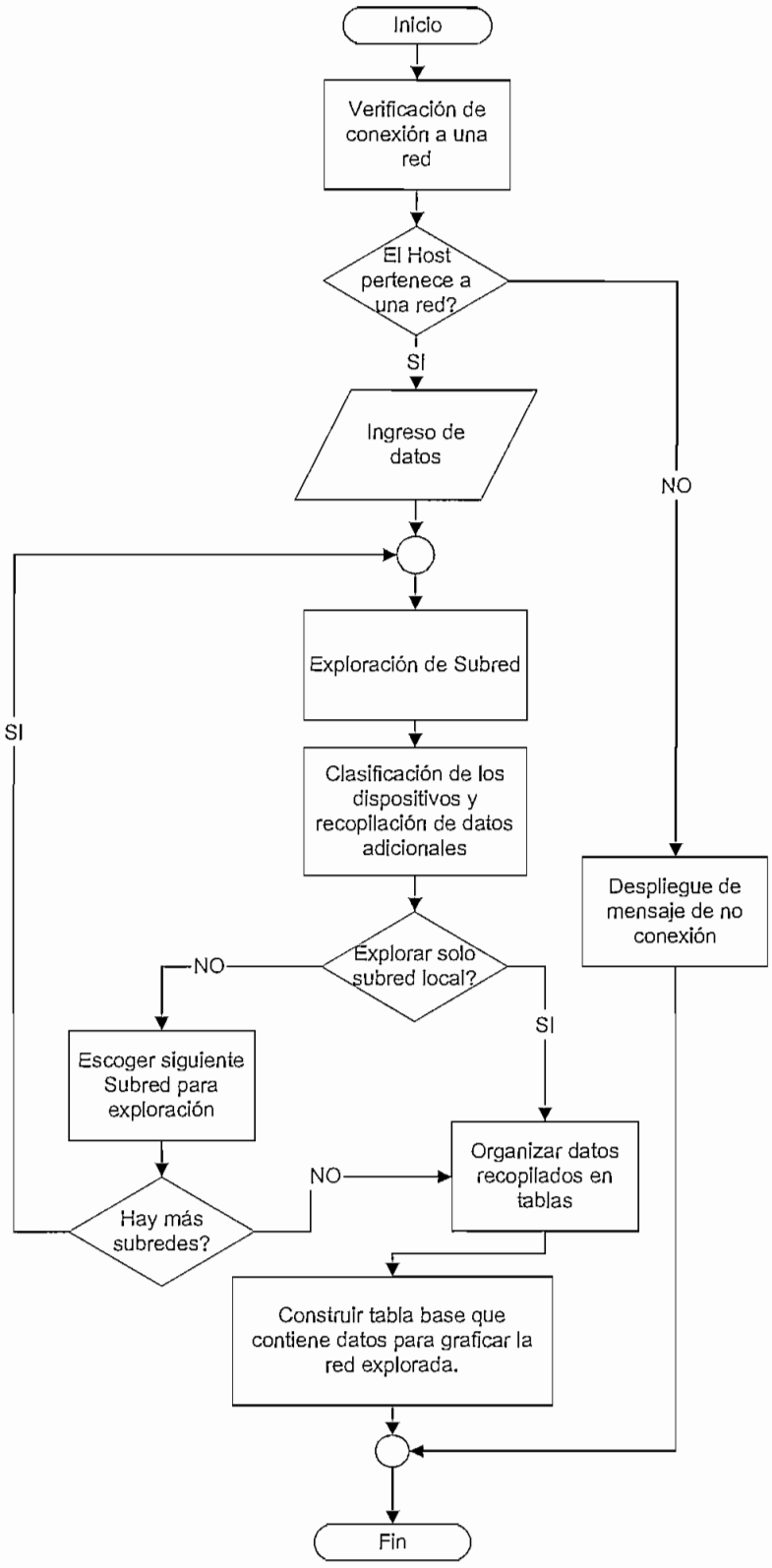


Figura 2.15 Diagrama de Flujo General de la clase CTotalScan.

2.7.4.3.2 Descripción del diagrama

Se resume en lo siguiente:

- ✓ Verificación del estado de conexión del host local a una red.
- ✓ Bloque de ingreso de datos, se lo detalla más adelante.
- ✓ Bloques de Exploración de la subred, Clasificación de los elementos y Recopilación de datos de elementos administrables, mutuamente relacionados; se los detalla más adelante.
- ✓ Verificación de tipo de exploración: subred local o red total. Por ejemplo si la dirección IP local es: 192.168.139.7, para el primer caso la exploración solo sería la subred 192.168.139.0(ó 192.168.139.x), en cambio si se trata del segundo caso la exploración sería 192.168.0.0(ó 192.168.x.x), buscando hasta 254 posibles elementos en cada subred adyacente.
- ✓ Bloque de Organización de datos recopilados por subred, y construcción de tablas de graficación.

2.7.4.4 Método del Ingreso de Datos

Para el bloque correspondiente al ingreso de datos, del diagrama de flujo de la figura 2.15, se necesita obtener una dirección IP referencial, definir un campo para ingreso de comunidades de lectura para utilizarlos con el protocolo SNMP, y la posibilidad de elegir entre el escaneo de red local o red total; todo dirigido a una interfaz visual a manera de dialogo que permita recopilar dichos datos.

2.7.4.4.1 Algoritmo utilizado

Se resume de la siguiente manera:

- ✓ Conocer la dirección IP local para tener una referencia de la subred a explorar.

- ✓ Iniciar la variables globales que se necesitan a lo largo del proceso de exploración.

- ✓ Ingreso de comunidades adicionales a las ya establecidas como: public y private.

- ✓ Elección del tipo de exploración de red: local (subred actual) o total(todas las subredes adyacentes).

- ✓ A partir de la dirección IP local se establece la dirección de subred a la que pertenece el *host* y se la establece como punto de partida.

2.7.4.4.2 Diagrama de Flujo

En la figura 2.15 se indica el diagrama de flujo del método creado.

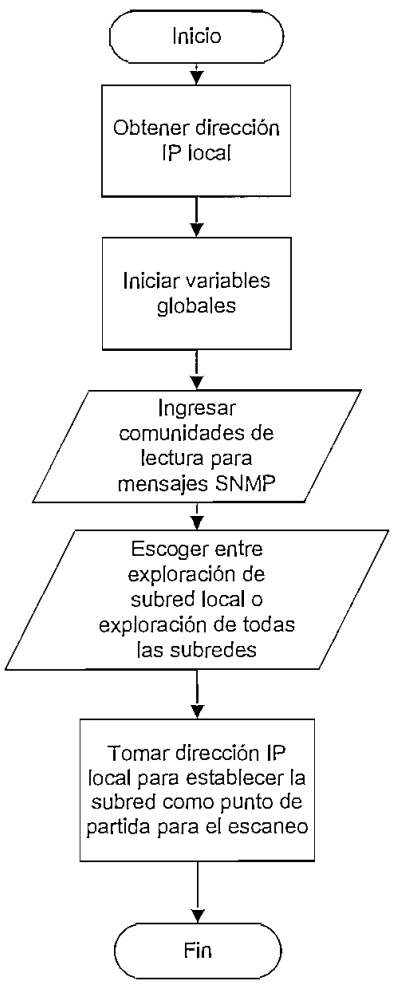


Figura 2.15 Diagrama de Flujo del bloque de ingreso de datos de la clase CTotalScan.

2.7.4.4.3 Descripción del método y funciones utilizadas.

Para la creación del método se utilizaron las siguientes funciones:

- ✓ Se obtiene la dirección IP local mediante el uso de las funciónes *gethostname* primero para obtener el nombre del computador local y luego *gethostbyname* para resolver la dirección (2.4.4.9 Funciones adicionales que contienen el uso de sockets).
- ✓ Se dan inicio a las variables locales como sockets, notificaciones y blanqueo de tablas para almacenaje de información.
- ✓ Se destina un campo y un procedimiento para el ingreso de comunidades a manera de cadenas de caracteres; adicionales a las ya establecidas por defecto(public y private).
- ✓ Se escoge por medio de controles de elección múltiple el tipo de exploración a realizarse, como: Exploración local(solo elementos presentes dentro de la subred actual) o exploración total(elementos pertenecientes a subredes adyacentes derivadas de la subred al principio explorada).
- ✓ Se tomo la dirección IP local para determinar la subred actual y empezar la exploración, por ejemplo: si la dirección del *host* es: 1923.168.139.7, entonces la subred de referencia será: 192.168.139.0 (ó 192.168.139.X).

2.7.4.5 Método de la exploración de la subred, clasificación de los elementos y recopilación de datos de elementos administrables.

Para el bloque correspondiente a la Exploración de la subred, Clasificación de los elementos y Recopilación de datos de elementos administrables, del diagrama de flujo de la figura 2.15, se necesita empezar con una dirección IP referencial de subred para la exploración de las 254 posibles direcciones de hosts, las comunidades de lectura para utilizarlos con el protocolo SNMP, en la búsqueda de posibles dispositivos administrables bajo el

mismo procedimiento, sus direcciónes MAC y su nombre; todo dependiendo de si el escaneo es de red local o de red total.

El diagrama de flujo general se lo divide en dos segmentos: la exploración propiamente dicha, y la clasificación de los elementos conjuntamente con la recopilación de datos; debido a que en la exploración se envían datagramas ping de reconocimiento, en busca de 254 posibles elementos, para en caso de éxito, continuar con direcciones MAC, búsqueda SNMP y por último el nombre del dispositivo; lo cual es una búsqueda al azar, en cambio que para la parte de clasificación se envía a los elementos encontrados paquetes definidos SNMP con preguntas puntuales para determinar su funcionalidad dentro de la red, y así clasificarlos y luego extraer datos sobre ellos.

La exploración se agiliza si existen elementos administrables dentro de la red que contengan información relevante, tales como: switches capa 3 o superiores, ruteadores, etc; los cuales proporcionan datos importantes como VLANs, enlaces WAN y tablas de red para una optimización en la búsqueda de elementos, su organización y clasificación.

Debido a que ambos segmentos están mutuamente relacionados, tanto el algoritmo utilizado, como el diagrama de flujo abarcan ambos aspectos del método y lo conjugan en uno solo.

2.7.4.5.1 Algoritmo utilizado

Se lo resume de la siguiente manera:

- ✓ Una vez obtenida la subred punto de partida de la exploración, se procede a realizar la búsqueda de hasta 254 posibles dispositivos presentes, usando protocolo ICMP.

- ✓ Si se tuvo éxito se realiza la búsqueda de direcciones MAC correspondientes a los elementos encontrados.

- ✓ Utilizando las comunidades ingresadas conjuntamente con la preestablecidas, se realiza una búsqueda de elementos administrables tomando la lista de elementos ya encontrados.

- ✓ Se resuelve los nombres de los *hosts* localizados.

- ✓ Se separan los computadores y dispositivos desconocidos, de los demás por clasificar.

- ✓ A los dispositivos administrables encontrados, mediante consultas como: número de interfaces, tipo de interfaces y objetos identificadores, se los clasifica en: computadores administrables, otros dispositivos administrables, impresoras de red, ruteadores, switches capa 2, switch capa 3 o superiores.

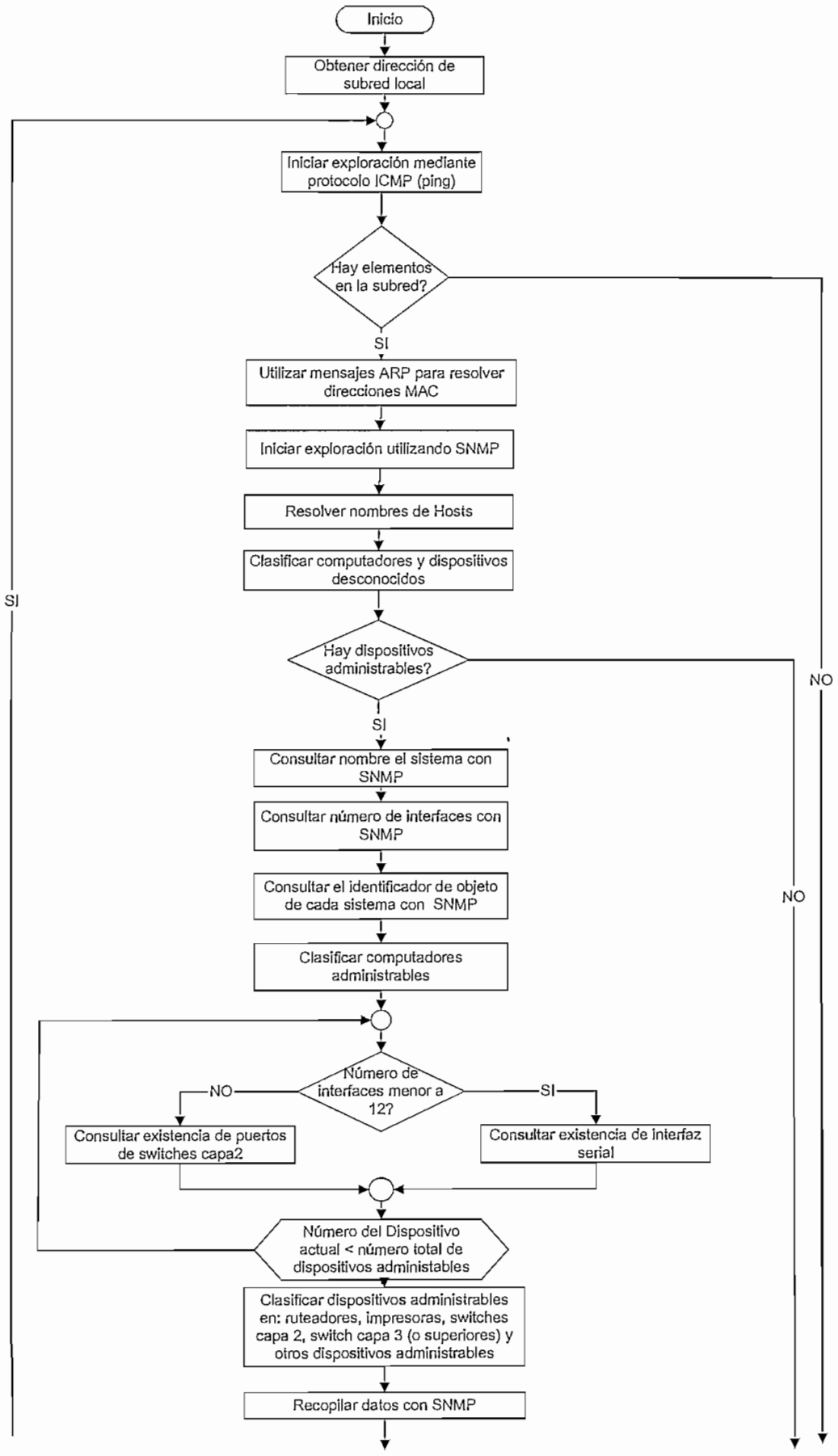
- ✓ De los elementos encontrados como: switch capa 3 o superiores, ruteadores, switches capa 2, se recopila información como tablas de ruteo, tablas de direcciones IP, tipos de enlaces WAN , puertos activos, dispositivos por puerto; que permitan la construcción de tablas de graficación por subred.

- ✓ Terminada la recopilación de datos se realiza un conteo de todos los elementos encontrados ya clasificados, y se organiza una tabla que contenga estos resultados.

- ✓ La operación se detiene si la exploración es de solo la subred local o se repite si se trata de la exploración de las subredes adyacentes también, hasta terminar con la red total.

2.7.4.5.2 Diagrama de Flujo

En la figura 2.16 se indica el Diagrama de Flujo del bloque de Exploración, clasificación y recopilación de datos de CTotalScan.


Inicio
Obtener dirección de subred local
Iniciar exploración mediante protocolo ICMP (ping)
Hay elementos en la subred?
SI
NO
Utilizar mensajes ARP para resolver direcciones MAC
Iniciar exploración utilizando SNMP
Resolver nombres de Hosts
Clasificar computadores y dispositivos desconocidos
Hay dispositivos administrables?
SI
NO
Consultar nombre el sistema con SNMP
Consultar número de interfaces con SNMP
Consultar el identificador de objeto de cada sistema con SNMP
Clasificar computadores administrables
Número de interfaces menor a 12?
NO
SI
Consultar existencia de puertos de switches capa2
Consultar existencia de interfaz serial
Número del Dispositivo actual < número total de dispositivos administables
Clasificar dispositivos administrables en: ruteadores, impresoras, switches capa 2, switch capa 3 (o superiores) y otros dispositivos administrables
Recopilar datos con SNMP

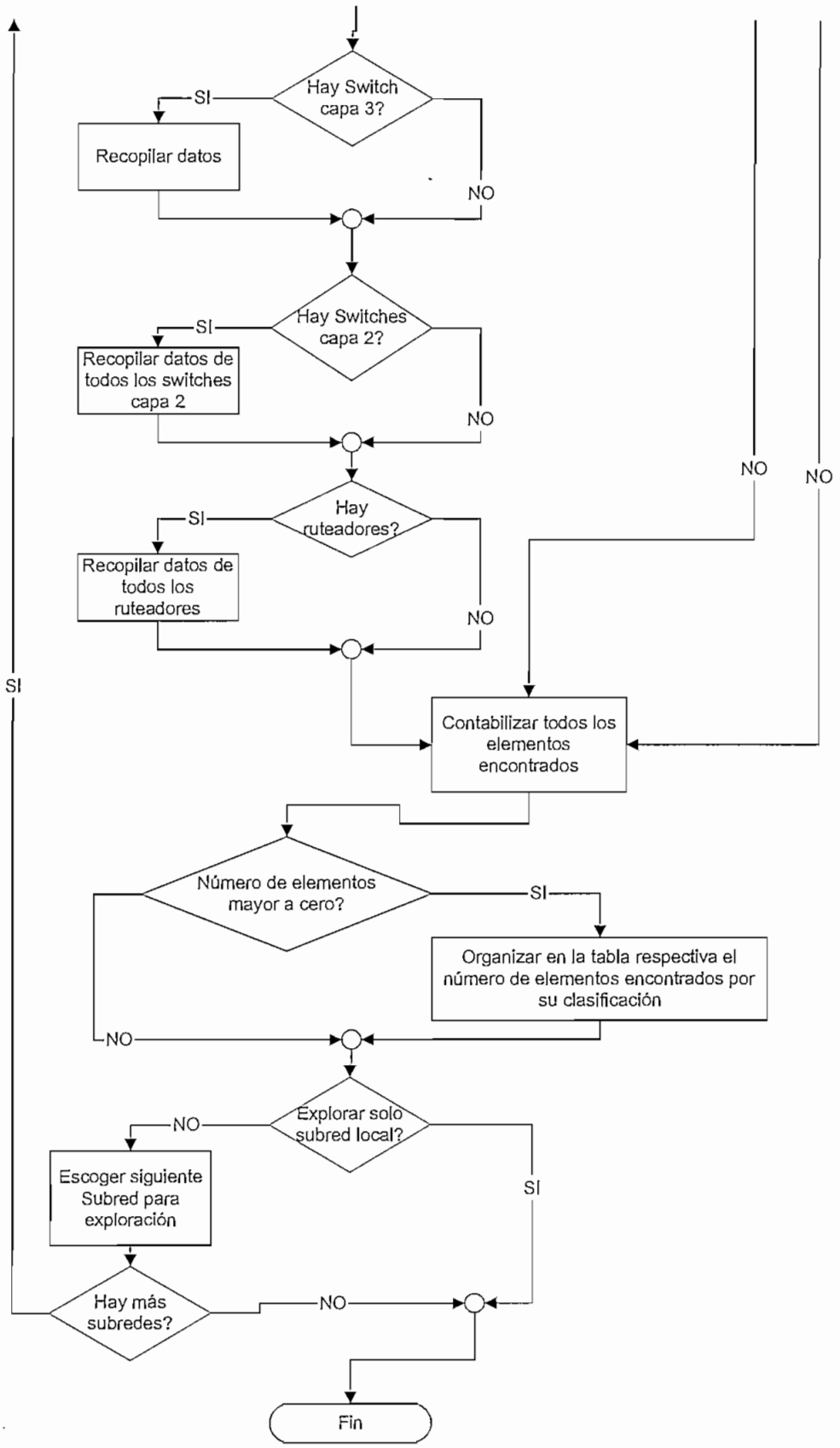


Figura 2.16 Diagrama de Flujo del bloque de Exploración, clasificación y recopilación de datos de CTotalScan.

2.7.4.5.3 Descripción del método y funciones utilizadas..

Para la creación del método se utilizaron las siguientes funciones:

- ✓ Sea el tipo de exploración: local o total, se procede a realizar primero la búsqueda de hasta 254 posibles dispositivos presentes en la subred donde se localiza el *host local*, usando protocolo ICMP, es decir ping mediante un objeto de la clase CIcmp; y en caso de éxito, se repite el procedimiento por dos ocasiones más, para confirmar las direcciones de los destinos alcanzados.

- ✓ Si se tuvo éxito se realiza la búsqueda de direcciones MAC correspondientes a los elementos encontrados, mediante el uso de la función *SendARP*, hacia las direcciones IP localizadas.

- ✓ Se realiza una búsqueda de elementos administrables tomando la lista de elementos ya encontrados, por tres ocasiones por cada comunidad preestablecida e ingresada, mediante el uso de SNMP; y enviando mensajes de tipo GETREQUEST preguntando por el nombre del sistema.

- ✓ Se resuelve los nombres de los *hosts* localizados, utilizando la función *WSAAsyncGetHostByAddr*[42] que resuelve los nombres de hosts remotos de manera asincrónica dadas las direcciones IP.

- ✓ Se separan los computadores y dispositivos desconocidos, de los demás tomando en cuenta que un computador puede tener un nombre, una dirección IP ni comunidad de administración, al igual que un dispositivo desconocido, que tampoco tenga una comunidad y que no soporte mensajes NetBIOS y por lo tanto no responda a la petición de nombre.

[42] *MSDN Library Visual Studio 6.0a,, Copyright 1995 – 2000 Microsoft Corp.*

- ✓ A los dispositivos administrables encontrados, se les realiza consultas SNMP como: número de interfaces (petición GETREQUEST, OID: 1.3.6.1.2.1.2.1.0), tipo de interfaces(petición GETNEXTREQUEST OID: 1.3.6.1.2.1.2.2.1.2) y objetos identificadores (petición GETREQUEST, OID: 1.3.6.1.2.1.1.2.0).

- ✓ Según la respuesta obtenida a la pregunta de objeto identificador, el del sistema Windows(1.3.6.1.4.1.311.1.1.3), a los elementos se los clasifica en: computadores administrables.

- ✓ Dependiendo del número de interfaces, se los divide en un grupo que contiene: Impresoras de red, ruteadores, menores a doce interfaces; y en otro que contiene switches capa 2, switch capa 3 o superiores, mayores a doce interfaces.

- ✓ De los elementos que pertenecen al primer grupo se verifica si alguno tiene una interfaz serial y se lo clasifica como ruteador, mientras que a los restantes como impresoras, dependiendo de su objeto identificador recopilado.

- ✓ De los elementos que pertenecen al segundo grupo, se averigua mediante SNMP, si tienen tablas de correspondencia de direcciones MAC con sus respectivos puertos y tablas para VLANs; para posteriormente clasificarlos como switches capa 2 o switch capa3, respectivamente.

- ✓ Si en la subred explorada existen dispositivos administrables como: switch capa3, se recopilan de él datos como tablas de ruteo, tablas de VLANs, tablas de red y tablas de direcciones IP; switches capa2, se recopilan de ellos tablas de interfaces activas y tablas de direcciones MAC con su correspondiente interfaz; ruteadores, se recopilan de ellos tablas de ruteo, tablas de interfaces, tablas de direcciones IP.

- ✓ Las tablas que se preguntan son por lo tanto: *ifTable*, *ipNetToMediaTable*, *ipAddrTable, ipRouteTable* tanto para switch capa3 como para ruteadores; y *ifTable, dot1dTpFdbTable* para switches capa2; mediante el uso de datagramas GETNEXTREQUEST(1.3.5.2.2 Grupo de interfaces y 1.3.5.2.4 Grupo IP).

- ✓ Terminada la recopilación de datos se realiza un conteo de todos los elementos encontrados en la subred, se clasifican y se organizan en una tabla que contengan los resultados, organizados por cada categoría(computadores, impresoras, ruteadores, etc) y por subred.

- ✓ Además por cada dato recopilado se maneja una clase CTableManage, la cual se encarga de actualizar los datos recopilados en las clases CComputers, CPrinters, CRouters, CSwitchLayer2, CSwitchLayer3, COtherDevices respectivamente, para así poder analizar un dispositivo y conocer: su dirección IP, su nombre, su dirección MAC, la subred a la que pertenece, la VLAN a la que pertenece (de ser el caso), el switch capa2 al que pertenece(de ser el caso), el puerto donde se localiza y si lo comparte con algún otro dispositivo, y en el caso de ruteadores el tipo de enlace serial que poseen sus interfaces(ejemplo: FrameRelay, etc).

- ✓ La operación se detiene si la exploración es solo para la subred local, o se repite si se trata de la exploración de las subredes adyacentes.

- ✓ En caso de no encontrarse ningún dispositivo y si la elección fue de exploración total, se procede a explorar las redes adyacentes, por ejemplo: si la subred explorada en un principio fue 192.168.139.0

entonces la subred por la que se empezaría sería 192.168.1.0 hasta alcanzar la subred 192.168.254.0, y terminar así con la red total.

2.7.4.6 Método de la organización de todos los datos recogidos, para la construcción de tablas de información de la red explorada.

Éste método es bastante preciso en su requerimiento, pues necesita de los datos recopilados y clasificados del anterior método, para crear la tabla general con datos específicos para graficación como: numero de elementos por subred, direcciones IP, switches capa2 con número de elementos por puerto(en caso de haber); y con campos destinados para el futuro almacenaje de coordenadas dentro de un plano, procedimiento del que se encarga otra clase más adelante.

2.7.4.6.1 Algoritmo utilizado

Se resume de la siguiente manera:

- ✓ Lectura de las tablas pertenecientes a ruteadores, switches capa2, switches capa3 en busca de información para la construcción de la tabla de graficación; pues éstos son nodos inteligentes de red que almacenan éste tipo de información.

- ✓ Lectura de tablas de computadores, impresoras y otros dispositivos, para ubicarlos según la subred a la que pertenecen.

- ✓ Construcción de la tabla de graficación con la información especificada.

2.7.4.6.2 Diagrama de Flujo

En la figura 2.17 se indica el Diagrama de Flujo del bloque de organización de todos los datos recogidos, para la construcción de tablas de información de la red explorada de CTotalScan.

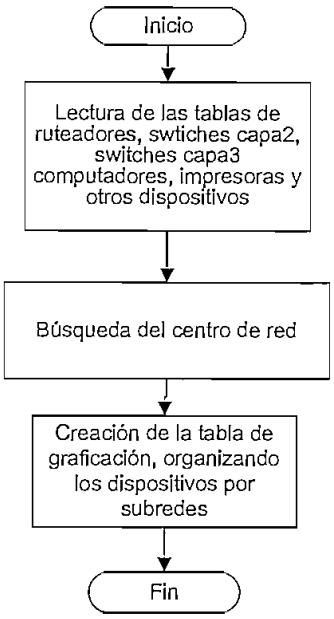


Figura 2.17 Diagrama de Flujo del bloque de organización de todos los datos recogidos, para la construcción de tablas de información de la red explorada de CTotalScan.

2.7.4.6.3 Descripción del método y funciones utilizadas

Se resume de la siguiente manera:

- ✓ Se leen de las tablas organizadas en las clases CComputers, CPrinters, CRouters, CSwitchLayer2, CSwitchLayer3, COtherDevices respectivamente, datos tales como: direcciónes IP, nombres, direcciónes MAC, subred a la que pertenecen, VLAN a la que pertenecen (de ser el caso), switch capa2 presentes (de ser el caso), puerto donde se localizan los dispositivos y lo comparten con algún otro dispositivo(de ser el caso), que permiten la graficación de la red.

- ✓ Se busca un centro general de red en caso de tenerse una exploración total, como por ejemplo: un switch capa3 multipuerto.

- ✓ En el caso de exploración de una subred local el centro es el dispositivo administrable que abarque más elementos.

- ✓ Mediante el uso de rutinas, se crea la tabla general de graficación; dividiendo a los dispositivos por subredes, creando además por cada dispositivo campos para almacenaje de coordenadas en un plano, nombre o dirección IP a presentarse en el gráfico, su tipo y elementos contiguos al mismo.

2.7.5 DISEÑO DE LA CLASE CNETGRAPHGOLDVIEW

Ésta clase maneja la interfaz visual con el usuario, contiene los botones, opciones y herramientas para la graficación; por lo tanto es la que reciba los datos de exploración, la tabla general de graficación y las complemente con datos, producto de los procedimientos y funciones para la esquematización de la red; es decir, es la que construye el gráfico de manera lógica para después ser dibujado con la ayuda de una clase adicional , y por último presentarlo en el visor destinado para ello.

La clase contiene los objetos: árbol de organización de la red, visor del contenido de la red, herramientas de reportes y herramientas adicionales. Por lo que contiene procesos que a medida que sean requeridos, se activan dependiendo de lo que seleccione el usuario; convirtiéndose en módulo principal, el método que realiza el gráfico llenando la tabla de graficación, con las coordenadas de la ubicación posterior a la distribución de cada elemento en el plano a dibujarse. Éste es entonces el método principal de la clase y por lo tanto, objeto de desarrollo para la presente.

Tomando en cuenta que la distribución de los elementos en un plano, se la realiza basándose en el número de elementos presentes y en el espacio que cada uno ocupa, es entonces necesario emplear funciones trigonométricas que proporcionen una repartición ordenada del espacio disponible, por lo que la clase en desarrollo debe soportar funciones matemáticas que organicen de manera sistemática dicho procedimiento.

La forma geométrica a utilizarse es circular; es decir, de distribución radial alrededor de un centro o centros, dependiendo de los nodos que se encuentren por subred; siendo entonces los elementos administrables tales como: switch capa3, switches capa2, ruteadores los que proporcionan el número de centros principales y secundarios alrededor de los cuales se localizan los elementos; tomando en cuenta también que existen subredes que no contengan dichos elementos, en cuyo caso se mantiene la distribución seleccionada, pero alrededor de un icono a manera de centro principal que simbolice la existencia de conexión con otras subredes.

2.7.5.1 Encapsulamiento

Como ésta clase se encarga de construir el gráfico de red de forma lógica a través de una tabla de graficación, se generan funciones que permitan tal operación, basándose en la distribución espacial seleccionada.

La clase a crearse encapsula los objetos pertenecientes a la clase CDibujo, la misma que esta asociada al visor gráfico; ocupando sus funciones al momento mismo de graficar. Adicionalmente encapsulará funciones derivadas de la librería math.h que permitan la distribución radial mencionada, y rutinas para llenar los datos en la tabla general.

2.7.5.2 Herencia

La clase CNetGraphGoldView hereda funcionalidad de un objeto de la clase CFormView, por lo que un objeto perteneciente a la clase CNetGraphGoldView tiene toda la funcionalidad de un objeto de CFormView, pero orientado a la presentación de gráficos de topología de red, utilizando objetos de las clases CTotalScan y CDibujo creadas para ello.

Adicionalmente hereda funcionalidad de objetos pertenecientes a las clases que tengan que ver con: el árbol de organización de la red, herramientas de reportes y herramientas generales.

2.7.5.3 Método para la graficación de una red

Lo que requiere la clase para empezar con su operación, es la tabla general que se la entrega la clase CTotalScan con los datos correspondientes a la subred o subredes exploradas, y acceso a funciones de la librería math.h para realizar el cálculo de las coordenadas de cada elemento.

Se debe tomar en cuenta que la clase CDibujo crea un mapa de bits en memoria con una capacidad de 4000x4000 píxeles, la cual representa el plano disponible para dibujo. La clase CDibujo y los valores mencionados se explican más adelante.

2.7.5.3.1 Algoritmo utilizado

Se resume de la siguiente manera:

- ✓ Lectura de la tabla de graficación; para determinar cuantas subredes se grafican.
- ✓ Por cada subred a graficar, se cuentan el número de elementos que la conforman, se identifican los centros principales y secundarios, se calcula la disposición de los elementos alrededor de cada uno de ellos y se distribuyen a fin de no causar sobreposiciones.
- ✓ Una vez terminada la distribución, se llena el árbol que contiene la organización de la red con los elementos encontrados.
- ✓ Por último se presenta en pantalla la primera subred presente en el árbol de organización.

2.7.5.3.2 Diagrama de Flujo

En la figura 2.17 se indica el diagrama de flujo del método creado.

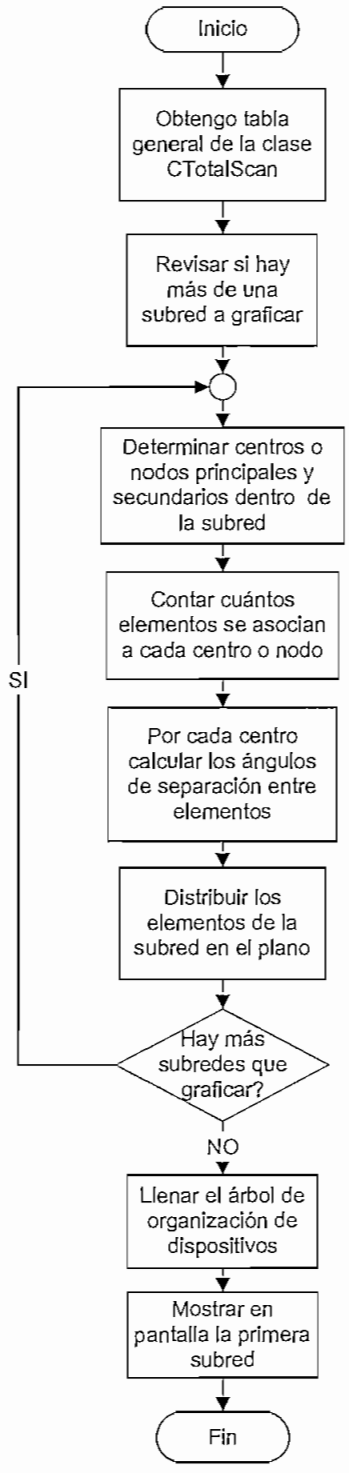


Figura 2.17 Diagrama de Flujo para la graficación de una red de la clase CNetGraphGoldView.

2.7.5.3.3 Descripción del método y funciones utilizadas

Se resume de la siguiente manera:

- ✓ Lectura de la tabla de graficación; que se logra de la exploración de la red y se la obtiene de la clase CTotalScan, para determinar cuantas subredes se grafican, y llenar los campos predeterminados de cada elemento con sus respectivas coordenadas en píxeles y el centro al cual pertenecen.

- ✓ Por cada subred a graficar, se cuentan el número de elementos que la conforman como: switches capa2, ruteadores, computadores, impresoras y otros.

- ✓ Se identifican los centros principales y secundarios, que pueden ser desde dispositivos administrables como switches capa2, hasta hubs, los cuales no tienen dirección IP pero se representan como un nodo de concentración de dispositivos.

- ✓ Se calcula la distribución de los elementos alrededor de cada centro, dividiendo el número total de elementos asociados a él, para 360° y así se obtiene el ángulo de separación entre cada uno.

- ✓ Se calcula la disposición de los elementos alrededor de cada uno de los centros, con funciones como seno y coseno con radios arbitrarios, obteniéndose sus respectivas coordenadas y distribuyéndolos a fin de no causar sobreposiciones.

- ✓ Si existen más subredes por graficar, se repite el mismo proceso hasta terminar toda la red.

- ✓ Una vez terminada la distribución, se llena el árbol que contiene la organización de la red con los elementos encontrados, a fin de facilitar la búsqueda de un elemento dentro de la red.

- ✓ Por último se presenta en pantalla la primera subred presente en el árbol de organización, utilizando funciones propias de la clase CDibujo que permiten plasmar en pantalla el resultado obtenido.

2.7.6 DISEÑO DE LA CLASE CDIBUJO

Esta clase se encarga de plasmar los iconos, líneas y figuras; que se presentan en pantalla en el visor predeterminado para la presentación del gráfico de red, por lo tanto se encarga de administrar el campo definido para ello.

El área destinada para la presentación del gráfico es de 773x610 píxeles y es menor al área total del gráfico, por lo que se presenta en pantalla el segmento que abarque el visor, de tal manera que sea posible mover el mismo sobre la superficie del gráfico real, y así explorar de manera visual la subred que se está presentando.

Se eligió un mapa de bits de dimensiones 4000x4000 píxeles para realizar los gráficos en memoria, teniendo en cuenta lo siguiente: las dimensiones de los iconos que representan cada dispositivo, un número alrededor de 255 dispositivos y con un margen de libertad apropiado, destinado para el dibujo de líneas de cada distribución radial, la ubicación de los nodos en el plano, y evitando en lo posible, superposiciones de figuras.

Además, el área de gráfico en pantalla es sensible a mensajes enviados desde un control periférico del computador como el ratón, por lo que presentan menús de opciones dependiendo de donde se realiza la acción sobre el respectivo control.

2.7.6.1 Encapsulamiento

Como ésta clase se encarga de dibujar en memoria, el gráfico de cada subred contenido de forma lógica en la tabla de graficación; se generan funciones que permitan ésta operación, dibujando cada elemento con su identificador respectivo y su conexión a otro elemento representado por una línea de unión.

La clase a crearse encapsula los objetos pertenecientes a una clase que permita la construcción de mapas de bits, uso de figuras geométricas y uso de fuentes para escritura en pantalla, tales como: CClientDC, la misma que esta asociada al visor gráfico propio del programa mediante un objeto derivado de la clase.

Los objetos que CClientDC encapsula, trabajan con un dispositivo de contexto que representa sólo el área del cliente de una ventana; en éste caso el área destinada al visor de gráfico de red.

2.7.6.2 Herencia

La clase CDibujo hereda funcionalidad de un objeto de la clase CStatic, por lo que un objeto perteneciente a la clase CDibujo tiene toda la funcionalidad de un objeto de CStatic, pero orientado a la presentación de gráficos de topología de red en un campo destinado para ello, utilizando objetos de la clase CClientDC, los cuales se encargan de manejar los diferentes elementos a manera de mapas de bits, con las funciones que heredan de ésta clase.

2.7.6.3 Descripción de las principales funciones utilizadas

Debido a que las operaciones que realiza la clase son de manejo de gráficos, las funciones creadas se las invoca desde la clase CNetGraphGoldView, para poder mostrarse en pantalla una subred seleccionada en el árbol de organización de la red; por lo que se explican a

continuación las funciones creadas y heredadas de la clase CClientDC, y de CDC clase base de CClientDC, que permiten tales operaciones:

- ✓ Primero se crea en memoria un espacio que albergue el dibujo de 4000x4000 píxeles, utilizando funciones, como: *CreateCompatibleDC* la cual aparta un espacio de memoria para un objeto CClientDC, *CreateCompatibleBitmap* el cual crea un mapa de bits de dimensiones específicas y *SelectObject* que une ambas instrucciones, para terminar con la operación.

- ✓ Para la graficación de dispositivos tales como: computadores, impresoras, ruteadores, switches en general; se crean mapas de bits en la vista de recursos de Visual C++ con sus formas respectivas, y luego se los añade en el lugar específico del plano con la función *BitBlt.*

- ✓ Para la graficación de líneas y rectángulos se utilizaron las funciones *LineTo* y *Rectangle* propias de CDC.

- ✓ Y para la función de ZOOM, se utilizó una función propia de CClientDC: *StretchBlt* la cual estrecha un gráfico a un área definida.

2.7.7 HERRAMIENTAS PRESENTES EN EL PROGRAMA

La aplicación desarrollada tiene además otras herramientas a las cuales se tiene acceso a través del menú creado para ello, y sus funciones se especifican en clases propias para cada herramienta, ocupando clases ya creadas como Clcmp, AsySNMP; y funciones anteriormente establecidas.

Con el fin de simplificar la explicación, se indica a continuación lo que cada una realiza:

- ✓ *Reportes de Red Total, Switch Capa3, Switch Capa2, Ruteadores, Computadores, Impresoras, Otros Dispositivos :* los cuales muestran los dispositivos encontrados en la exploración organizados en cada reporte, con sus respetivas características, tomadas de las tablas organizadas en las clases: CComputers, CPrinters, CRouters, CSwitchLayer2, CSwitchLayer3, COtherDevices.

- ✓ *DNSResolver:* el cual ocupa las funciónes *WSAAsyncGetHostByAddr y WSAAsyncGetHostByName (*2.4.4.9 Funciones adicionales que contienen el uso de sockets*)* para la resolución de nombres o direcciones IP específicas.

- ✓ *Ping :* la cual ocupa las funciones creadas en CIcmp, para tener una herramienta de respuesta de *echo* siempre disponible.

- ✓ *Tracert*: que utiliza las funciones creadas en CIcmp, con la única variante que se varia el tiempo de vida TTL para trazar la ruta.

- ✓ *SNMP:* el cual ocupa la clase AsySNMP para el envío y recepción de paquetes SNMP; con la variante de que se tiene a disposición un árbol con OIDs para escoger que tipo de peticiones enviar.

- ✓ *Buscador de OUIs:* el cual ocupa una simple rutina de búsqueda en un archivo que contiene las OUIs de cada fabricante de interfaz de red, una vez ingresado el dato a buscar para mostrarlo en pantalla.

- ✓ *Medidores:* el cual contiene algunos medidores de tráfico para monitorear las interfaces seriales de los ruteadores, a fin de observar el comportamiento de la red.

- ✓ *Localizador de dispositivos* : el cual una vez graficada la red, busca un dispositivo por su dirección IP y lo muestra en pantalla según la subred que lo contenga.

CAPÍTULO 3: PRUEBAS Y RESULTADOS OBTENIDOS.

Las pruebas que se realizaron corresponden a redes ya establecidas como son: la red perteneciente al Laboratorio de Informática en el sexto piso del edificio de Eléctrica – Química de la Escuela Politécnica Nacional(EPN) y la red perteneciente a la Secretaría Nacional de Telecomunicaciones(SNT).

Se realizaron pruebas en los modos de exploración local; es decir, solo la subred a la que pertenece el host local para la red de la EPN mencionada, y en modo de exploración total; es decir, la subred local y todas las subredes adyacentes, para la red de la SNT.

En algunos casos donde la red posea hosts con sistemas operativos diferentes de Windows, o con el servicio NetBIOS inactivo, será imposible determinar el nombre del host, puesto que uno de los paquetes que se envían en la exploración, necesita de éste servicio para mostrar el resultado correspondiente. Dado el caso, el dispositivo será clasificado de acuerdo a las respuestas de los otros paquetes enviados para los protocolos: ICMP y SNMP.

El gráfico es obtenido y presentado de manera automática, una vez terminada la exploración, en el espacio determinado para ello, y se facilita su exploración utilizando la herramienta del árbol de organización de los elementos presentes en la subred o subredes posicionado a la izquierda del mismo.

En cuanto a los reportes entregados y herramientas adicionales, se tienen submenús en la barra de menús que contienen las opciones de visualización de reportes de computadores, impresoras, ruteadores, switches, etc., con sus respectivas características; así como herramientas varias y de conectivaidad como: Ping, Tracert, DNSResolver, SNMP, y medidores de tráfico en general ubicados en los menús Reportes y Herramientas respectivamente.

3.1 EL PROGRAMA

El programa creado tiene por nombre NetGrpahGold, y su ventana principal se muestra en la figura 3.1.

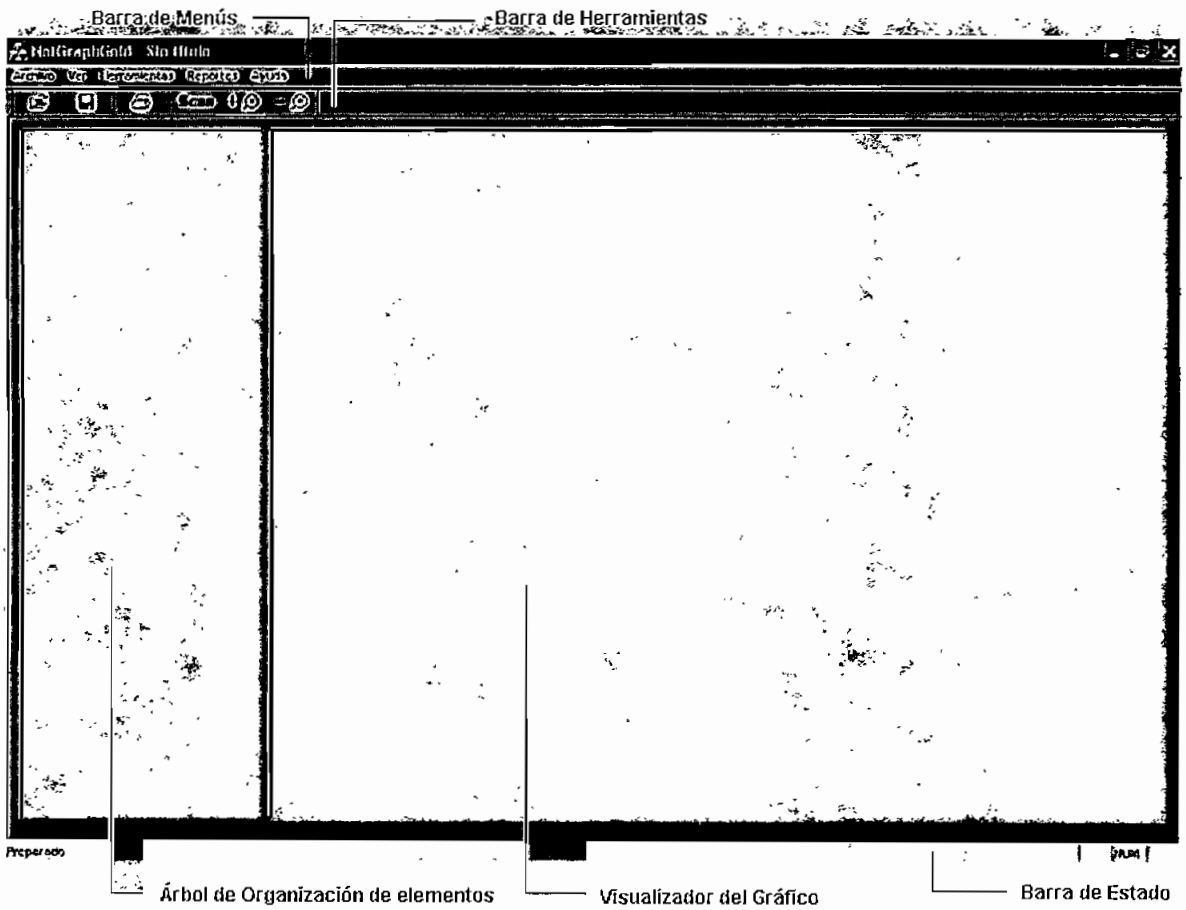


Figura 3.1 Ventana Principal del programa NetGraphGold

En la Barra de Herramientas encontramos los botones: Abrir, para abrir un archivo guardado *.ngg* ; Guardar, para guardar un proyecto de exploración terminado; Imprimir, para imprimir un gráfico específico; SCAN, para empezar la exploración de una red; y ZOOM + / - , para ver un gráfico en tamaño completo o cierta parte del gráfico.

En la Barra de Menús encontramos menús como: Archivo, para Abrir, Guardar un Proyecto, y Salir del programa; Ver, para visualizar las diferentes barras del programa; Herramientas, que contiene las herramientas de medición de tráfico y herramientas como: Ping, Tracert, DNSResolver, SNMP, Buscador de OUIs para identificar el fabricante de la interfaz de red,

y un Localizador de Dispositivos para encontrar un dispositivo en una red ya graficada; y Reportes, que contiene los Reportes de Computadores, Impresoras, Switches Capa2, Switches Capa3, Ruteadores y otros dispositivos presentes en una red, así como los elementos encontrados por cada subred.

En las figuras 3.2 y 3.3 se muestran los menús de Herramientas y Reportes respectivamente, con sus diferentes opciones.

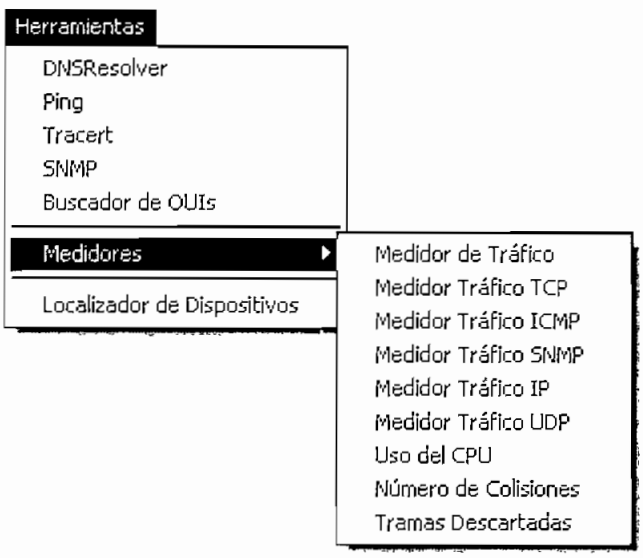


Figura 3.2 Menú de Herramientas del programa NetGraphGold

Figura 3. 3 Menú de Reportes del programa NetGraphGold

Al momento de pulsar el botón SCAN o escoger en el menú Archivo/Scan, la exploración se inicia y se muestra en pantalla un cuadro de diálogo que contiene los siguientes campos: dirección IP local del host, subred local a la que pertenece el host, visor de sucesos donde se muestra el estado de la exploración en curso, la lista de elementos encontrados donde se enlista cada elemento con su dirección IP y su tipo, un campo para ingreso de

comunidades para elementos administrables con su respectivo botón, una lista de comunidades de hasta un máximo de 30 comunidades, un campo que indica el estado del proceso, y un botón para inicio de la exploración donde se escogerá el modo de exploración a realizarse, como se muestra en la figura 3.4.

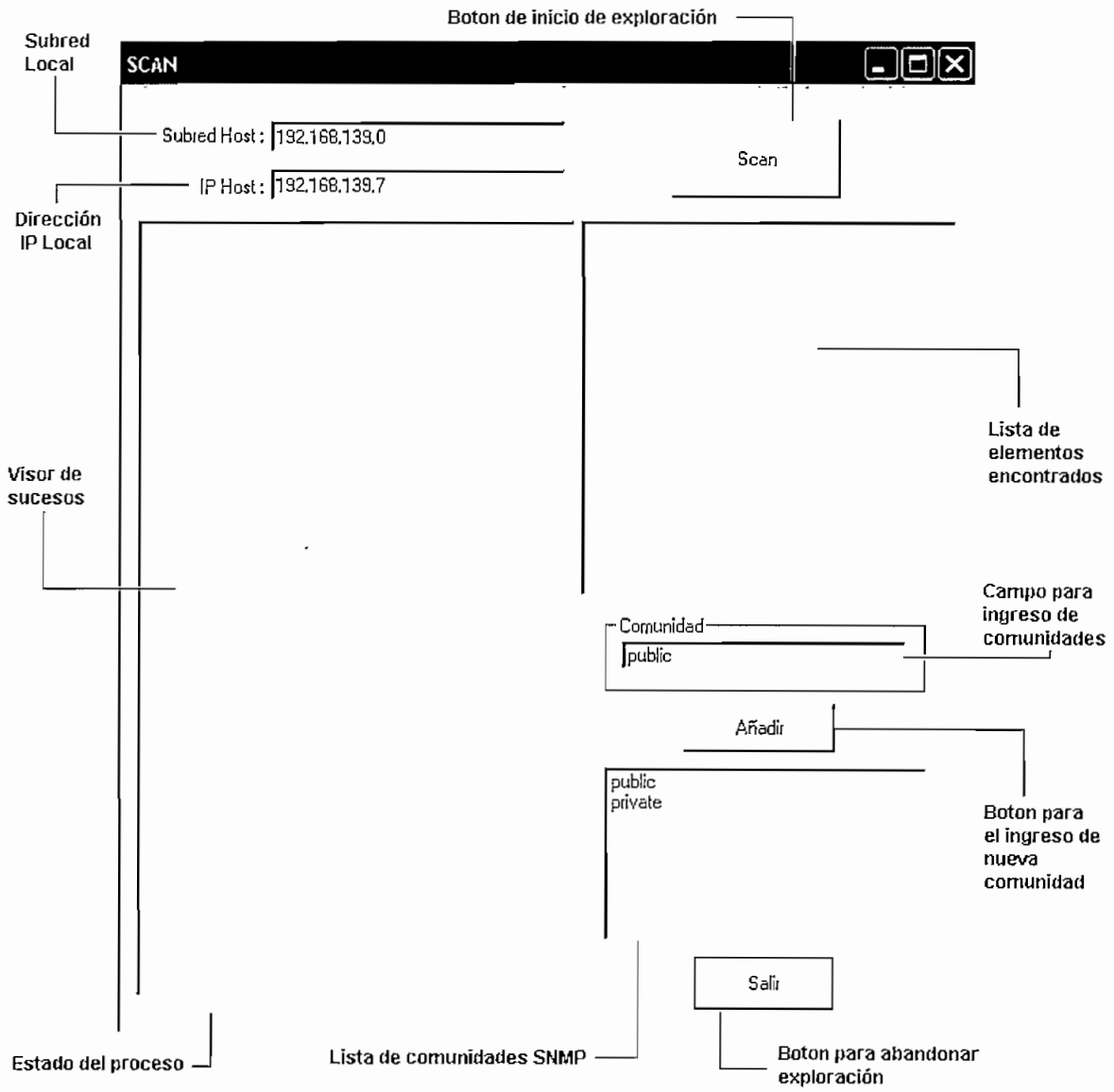


Figura 3. 4 Cuadro de Diálogo SCAN del programa NetGraphGold

Los íconos utilizados para representar los diferentes dispositivos dentro de una red establecida son:

 Que representa a un computador o un dispositivo desconocido

 Que representa a una impresora.

 que representa a un ruteador.

 Que representa a un Switch Capa2.

 Que representa a un Switch Capa3 o superior.

 Que representa una nube de comunicación entre dispositivos*.

Que representa la conexión de un dispositivo con otro.

* También puede representar un hub.

3.2 EXPLORACIÓN EN MODO LOCAL

La exploración en modo local permite encontrar mediante paquetes de *echo (ping)* los elementos presentes solo dentro de la subred local donde el host en uso se encuentra, para después buscar los elementos administrables dentro de la misma y por último proceder a graficar la subred.

Una vez pulsado el botón *Scan* en el cuadro de diálogo indicado en la figura 3.4, se procede a escoger el modo de exploración local en la ventana que se muestra en la figura 3.5; es decir, la exploración de los hosts presentes en la subred local.

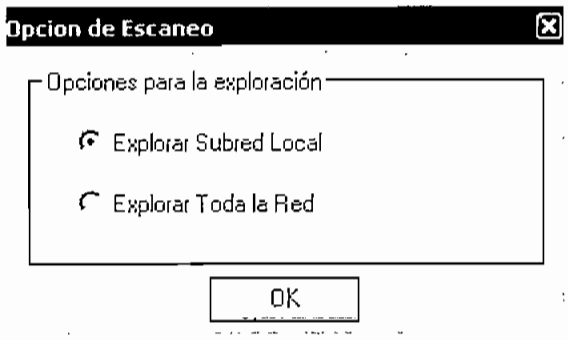


Figura 3. 5 Cuadro de Diálogo para opción de Modo de Exploración Local del programa NetGraphGold

La prueba realizada en la red perteneciente al Laboratorio de Informática en el sexto piso del edificio de Eléctrica – Química de la Escuela Politécnica Nacional ilustra mejor la exploración en modo local.

La red en cuestión consta de aproximadamente 90 dispositivos y se encuentran unidas mediante switches capa2 administrables y no administrables y hubs en una topología tipo estrella.

La figura 3.6 muestra el proceso de exploración que el programa NetGrpahGold realiza en busca de dispositivos presentes y activos dentro de una subred, mostrando en el visor de sucesos las operaciones realizadas, y direcciones IP encontradas; así como en la lista de dispositivos, los componentes de red clasificados e identificados con sus respectivas direcciones IP.

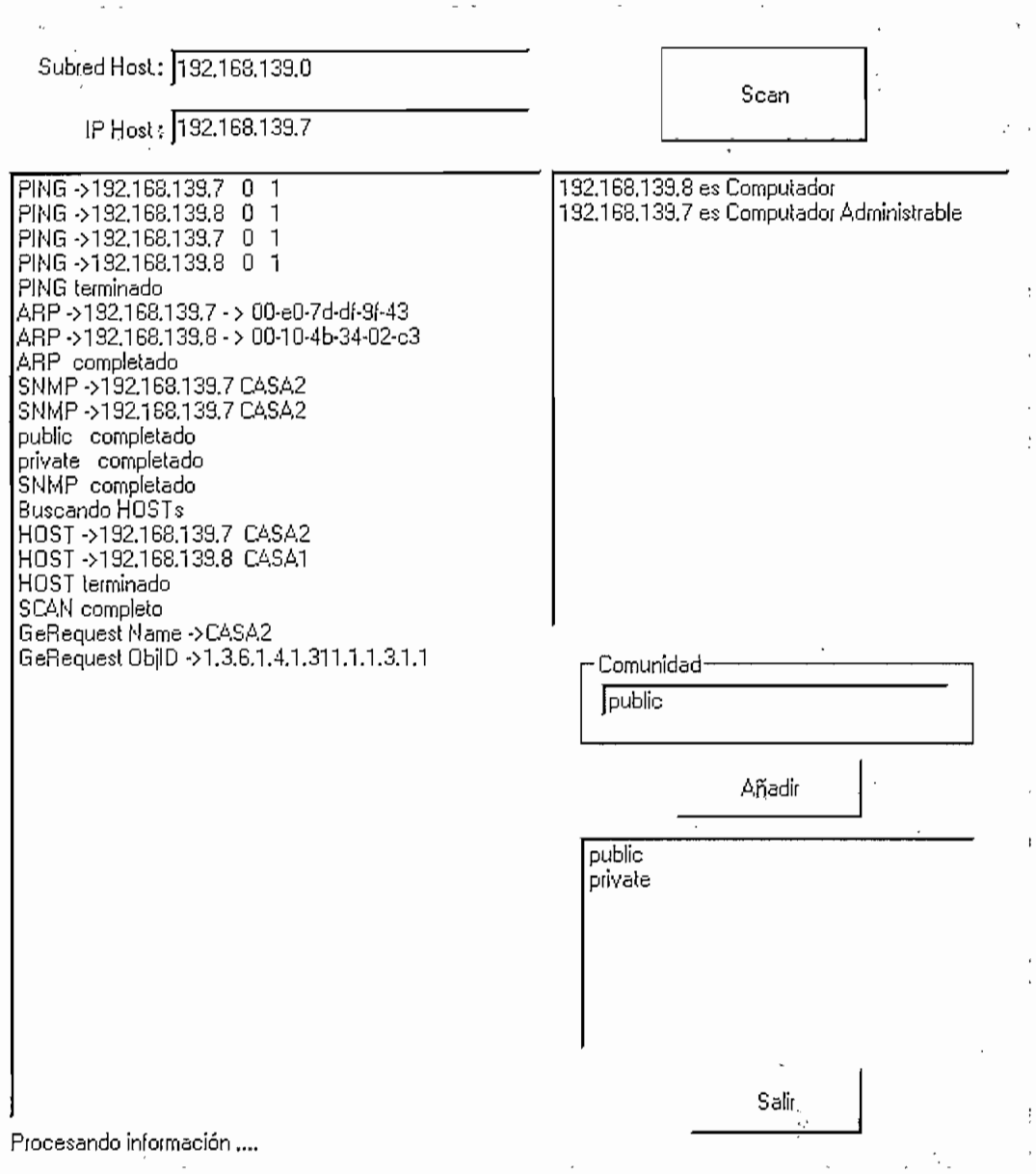


Figura 3. 6 Cuadro de Diálogo de las operaciones de Exploración en una subred con el Programa NetGraphGold

Las direcciones exploradas fueron desde la 192.168.77.1 hasta la 192.168.77.254 debido a que solo se hizo la exploración local y los reportes entregados fueron de un switch capa2, 77 computadores activos y 8 dispositivos desconocidos encontrados. La lista de dispositivos se detallan a continuación en: Reporte de Switches Capa2 (figura 3.7), Reporte de Computadores (figuras 3.8) y Reporte de Otros Dispositivos (figura 3.9) respectivamente.

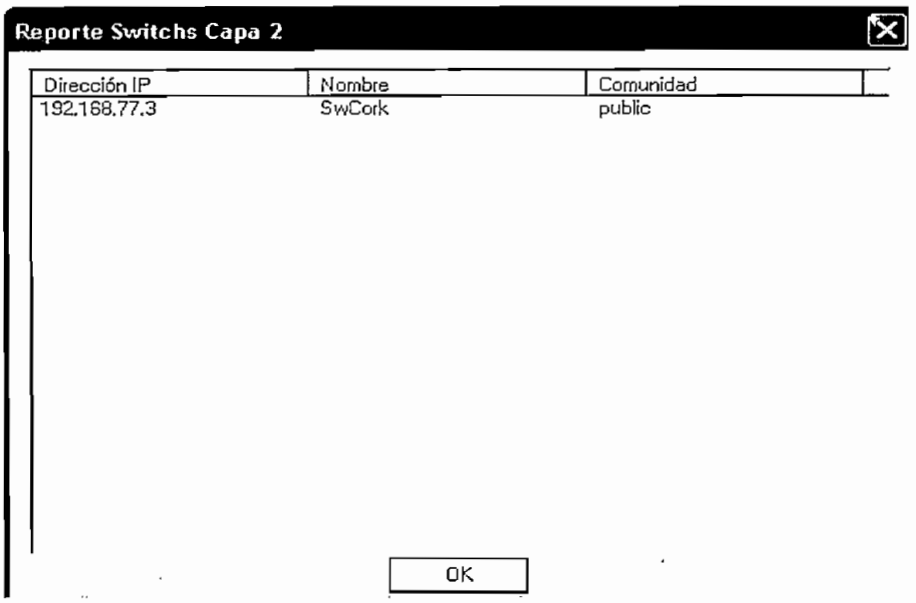


Figura 3. 7 Reporte de Switches Capa2 encontrados en la subred de la EPN

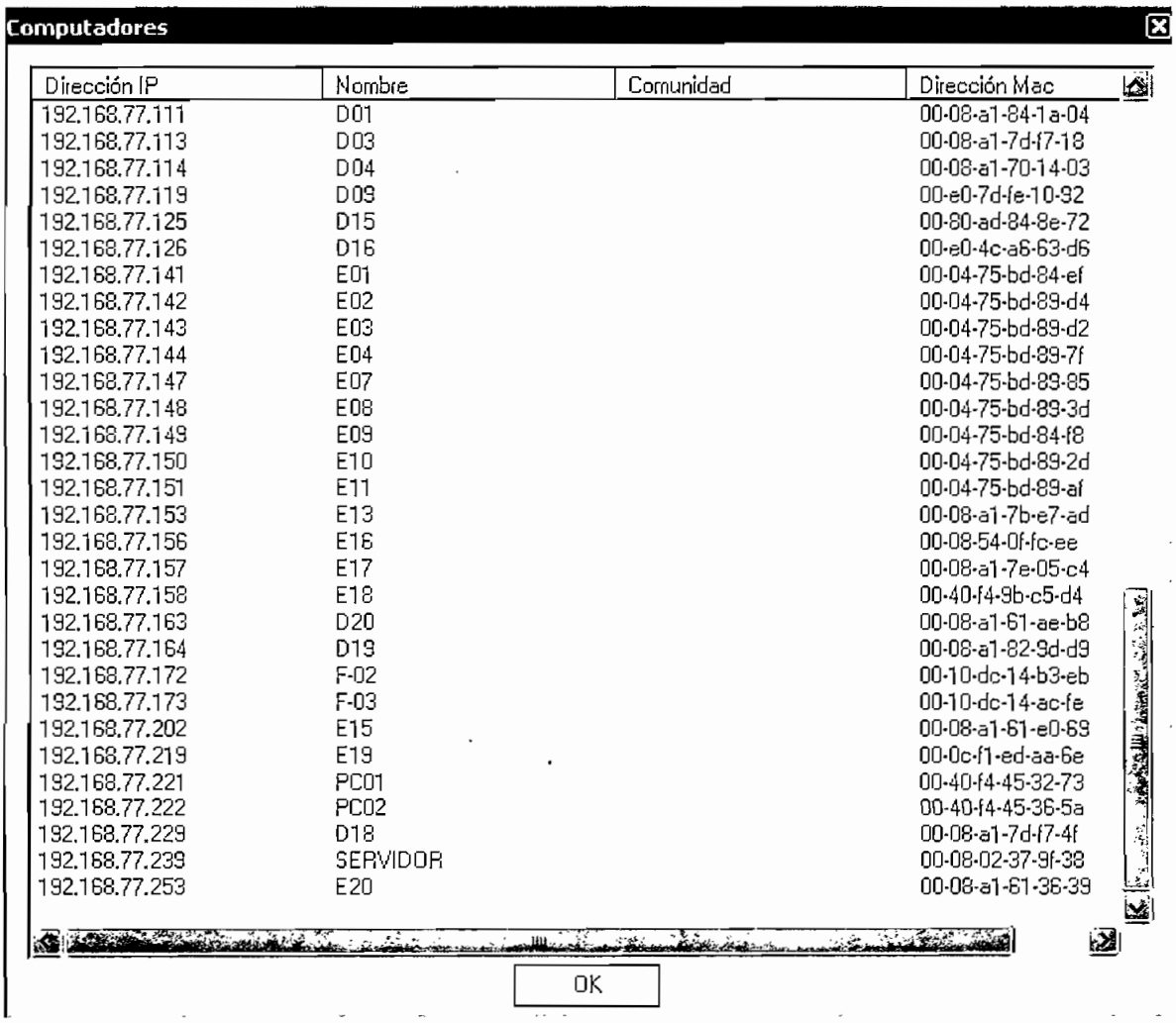


Figura 3. 8 Reporte de Computadores encontrados en la subred de la EPN

Figura 3. 9 Reporte de Otros dispositivos encontrados en la subred de la EPN

Y el gráfico obtenido de la red del Laboratorio de Informática en el sexto piso del edificio de Eléctrica – Química de la Escuela Politécnica Nacional (EPN) explorado el 18 de noviembre del año 2005 se lo indica de manera completa en la figura 3.10 a continuación, pudiendo distinguirse con claridad que se trata de una topología tipo estrella con una concentración mayor de dispositivos hacia un puerto del switch capa2 encontrado, y una concentración menor hacia los otros puertos del mismo switch.

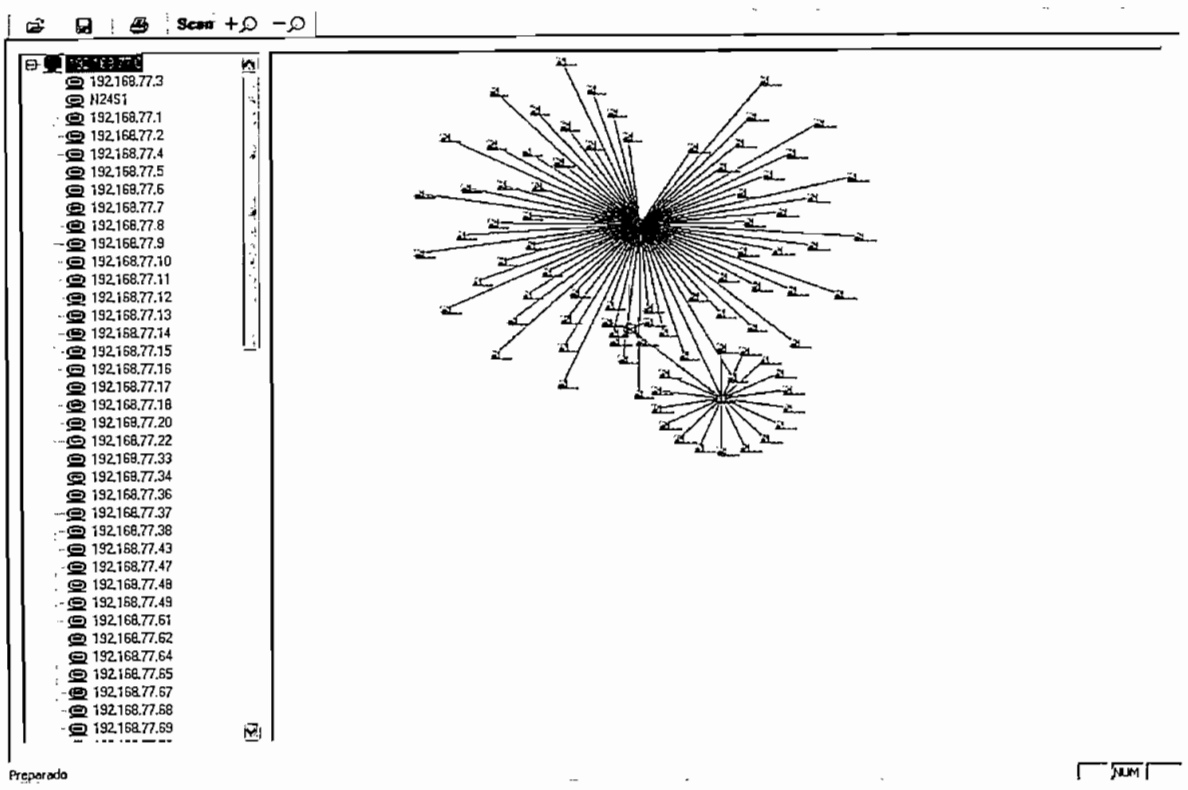


Figura 3. 10 Subred del Laboratorio de Informática del sexto piso del Edificio de Eléctrica – Química de la EPN.

También se puede observar que al costado izquierdo se encuentra una lista completa de los dispositivos que pertenecen a la subred explorada, facilitando la búsqueda de una dirección IP en particular para el usuario.

Como ayuda adicional se puede conocer las propiedades de cada elemento, basta con hacer doble clic sobre el elemento seleccionado o clic derecho opción propiedades, para conocer más acerca del dispositivo como: nombre, dirección IP, dirección MAC, tipo, comunidad y puerto al que pertenece; siempre y cuando se disponga de tal información, como se ilustra en la figura 3.11 tomando al elemento 192.168.77.157 como ejemplo en la subred explorada.

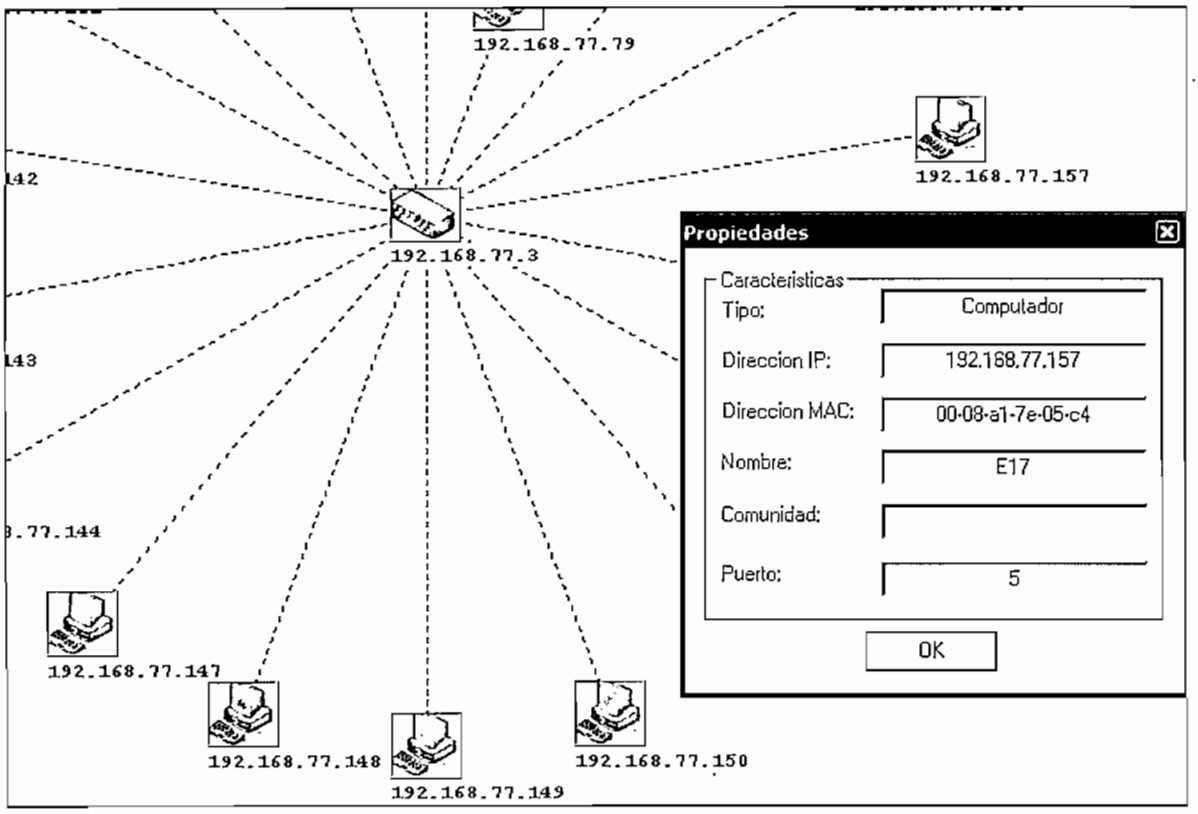


Figura 3. 11 Ventana Propiedades del elemento 192.168.77.157 de la red del Laboratorio de Informática del sexto piso de la del Edificio de Eléctrica – Química de la EPN.

3.3 EXPLORACIÓN EN MODO TOTAL(RED TOTAL)

La exploración en modo RED TOTAL permite encontrar mediante paquetes de *echo (ping)* los elementos presentes no solo dentro de la subred local donde se encuentra el host de prueba, sino también en las redes adyacentes, para después buscar los elementos administrables por cada subred y por último proceder a graficar cada subred con la información obtenida.

Una vez pulsado el botón *Scan* en el cuadro de diálogo indicado en la figura 3.4, se procede a escoger el modo de exploración de red total en la ventana que se muestra en la figura 3.12; es decir, la exploración de los hosts presentes en toda la red.

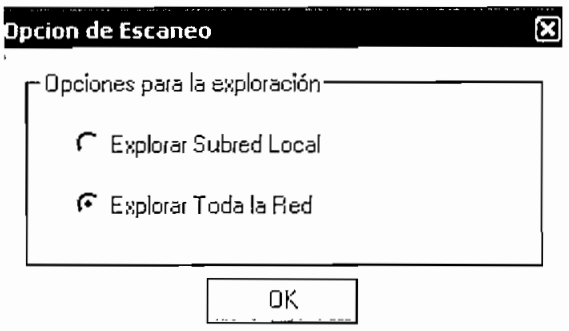


Figura 3. 12 Cuadro de diálogo para opción de Modo de Exploración Total del programa NetGraphGold

Si no existen más subredes que la local, la exploración continuará hasta terminar con todas las posibles subredes adyacentes, obteniéndose como resultado al final que la única subred encontrada es la local.

Para explicar de mejor manera, se toma el ejemplo de la red de la Secretaría Nacional de Telecomunicaciones (SNT), la cual consta de 19 subredes, un switch capa3 que funciona de núcleo, de aproximadamente 160 computadores, aproximadamente 170 dispositivos varios, 17 impresoras de red, 5 ruteadores y 10 switches capa2 activos para SNMP.

La figura 3.6 anteriormente mencionada, muestra el proceso de exploración que el programa NetGrpahGold realiza en busca de dispositivos presentes y activos dentro de una subred, mostrando en el visor de sucesos las operaciones realizadas, con la variante de que terminada la exploración de la subred local, comienza la exploración de todas las posibles subredes adyacentes; y en la lista de dispositivos, los componentes de red encontrados; es decir, si la subred local explorada en un inicio fue 192.168.139.0, las posibles subredes adyacentes serán desde la 192.168.1.0 hasta la 192.168.254.0, excluyendo la ya explorada, y comprendiéndose un rango de 254 posibles hosts por subred.

La red de la SNT posee enlaces WAN con sus regionales en Guayaquil y en Cuenca, por lo que se tienen distribuidos 4 ruteadores para tales operaciones, y un enlace para Internet en el cual se ocupa otro ruteador de mejores características.

La red de la SNT explorada el día 10 de noviembre del año 2005, entregó reportes de: Switch Capa3 (figura 3.13), Red Total (figura 3.14), Computadores (figura 3.15), Impresoras (figura 3.16), Ruteadores (figura 3.17), Switches Capa2 (figura 3.18) y Otros Dispositivos (figura 3.19) con los valores aproximados antes mencionados.

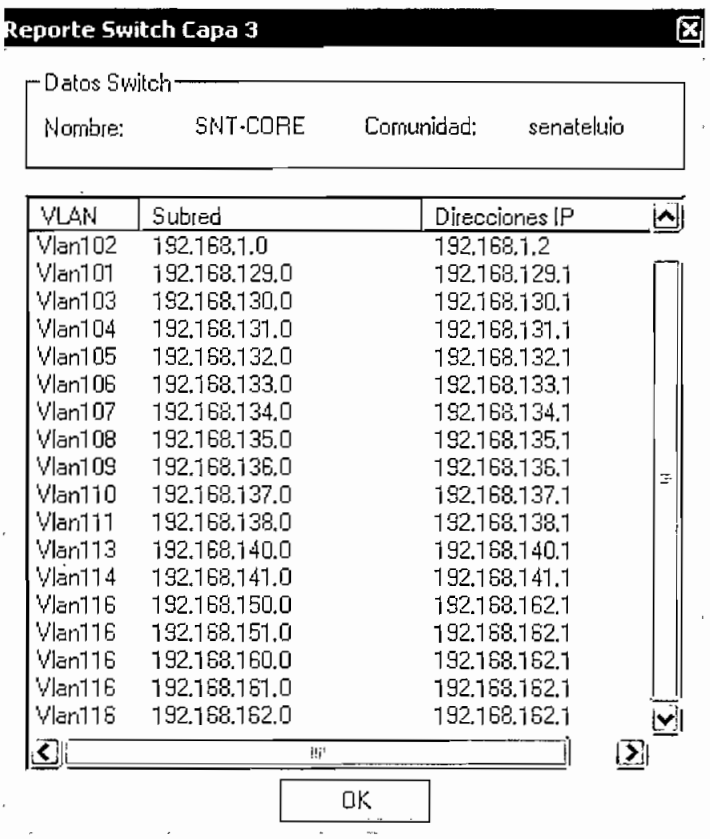
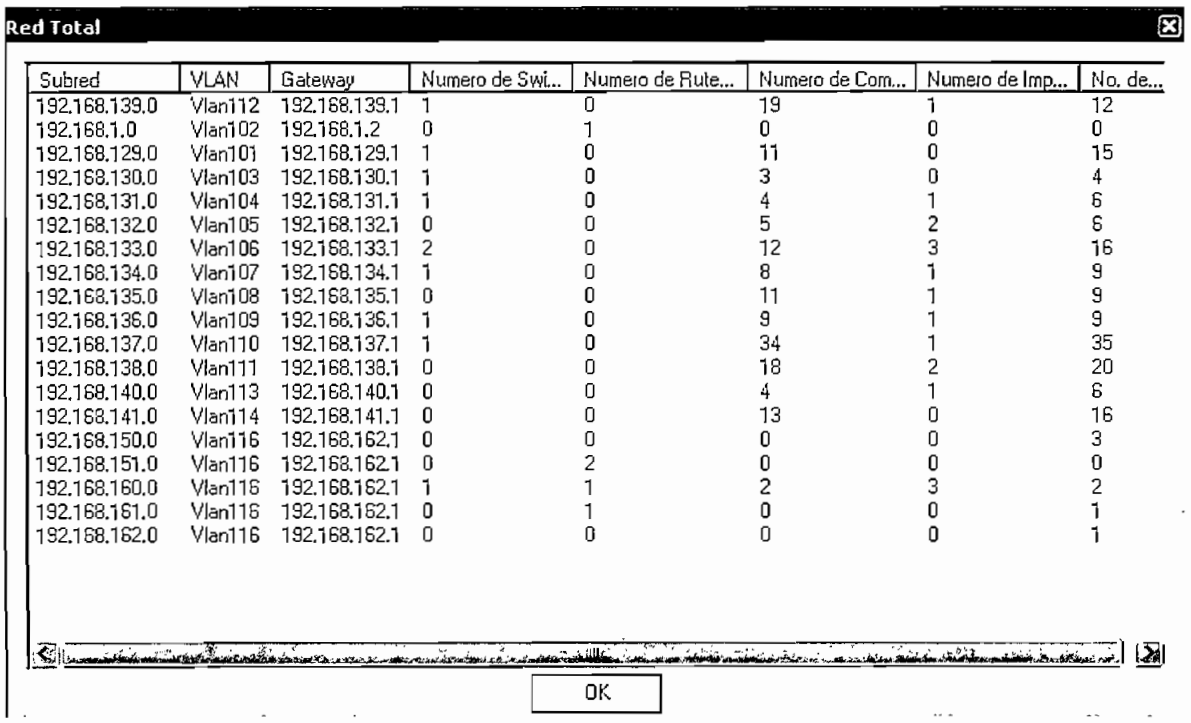
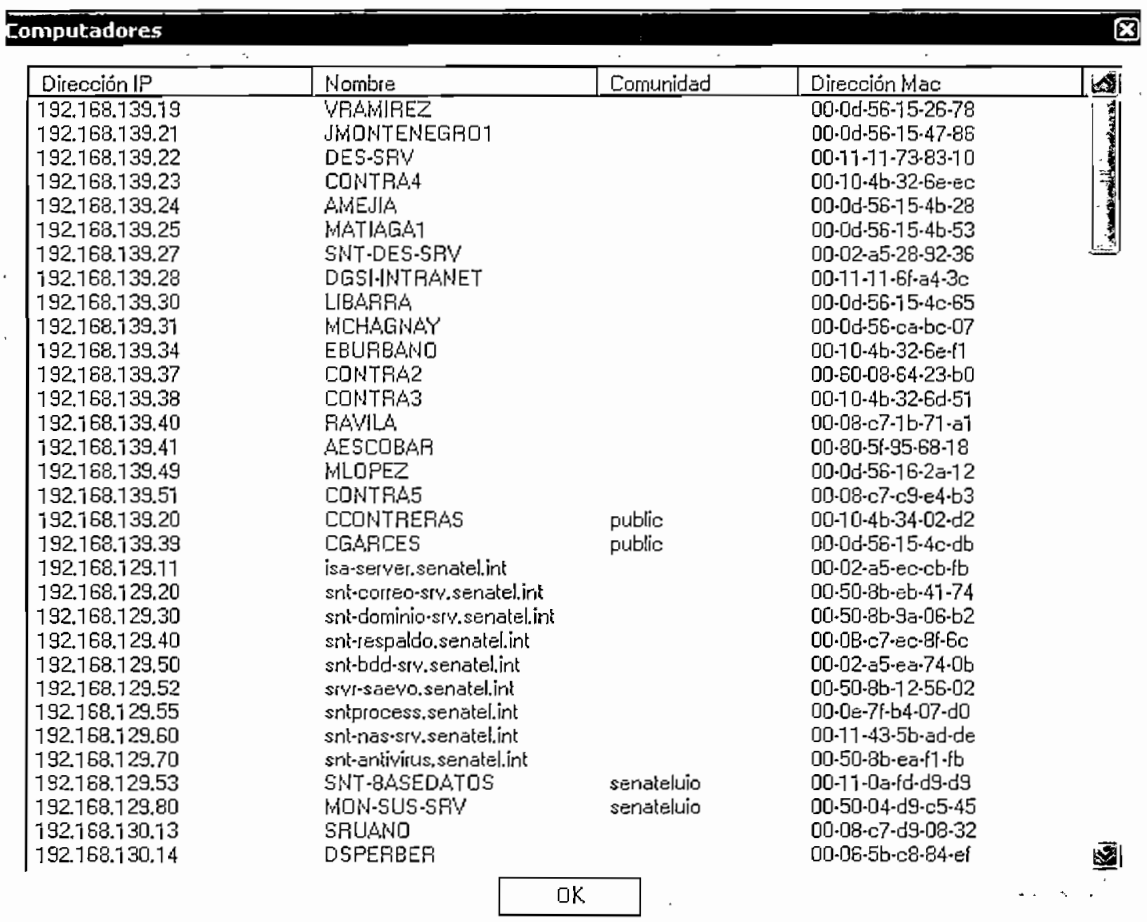


Figura 3. 13 Reporte del Switch Capa3 de la SNT del programa NetGraphGold



Subred	VLAN	Gateway	Numero de Swi...	Numero de Rute...	Numero de Com...	Numero de Imp...	No. de...
192.168.139.0	Vlan112	192.168.139.1	1	0	19	1	12
192.168.1.0	Vlan102	192.168.1.2	0	1	0	0	0
192.168.129.0	Vlan101	192.168.129.1	1	0	11	0	15
192.168.130.0	Vlan103	192.168.130.1	1	0	3	0	4
192.168.131.0	Vlan104	192.168.131.1	1	0	4	1	6
192.168.132.0	Vlan105	192.168.132.1	0	0	5	2	6
192.168.133.0	Vlan106	192.168.133.1	2	0	12	3	16
192.168.134.0	Vlan107	192.168.134.1	1	0	8	1	9
192.168.135.0	Vlan108	192.168.135.1	0	0	11	1	9
192.168.136.0	Vlan109	192.168.136.1	1	0	9	1	9
192.168.137.0	Vlan110	192.168.137.1	1	0	34	1	35
192.168.138.0	Vlan111	192.168.138.1	0	0	18	2	20
192.168.140.0	Vlan113	192.168.140.1	0	0	4	1	6
192.168.141.0	Vlan114	192.168.141.1	0	0	13	0	16
192.168.150.0	Vlan116	192.168.162.1	0	0	0	0	3
192.168.151.0	Vlan116	192.168.162.1	0	2	0	0	0
192.168.160.0	Vlan116	192.168.162.1	1	1	2	3	2
192.168.161.0	Vlan116	192.168.162.1	0	1	0	0	1
192.168.162.0	Vlan116	192.168.162.1	0	0	0	0	1

Figura 3. 14 Reporte de la Red Total de la SNT del programa NetGraphGold



Dirección IP	Nombre	Comunidad	Dirección Mac
192.168.139.19	VRAMIREZ		00-0d-56-15-26-78
192.168.139.21	JMONTENEGRO1		00-0d-56-15-47-86
192.168.139.22	DES-SRV		00-11-11-73-83-10
192.168.139.23	CONTRA4		00-10-4b-32-6e-ec
192.168.139.24	AMEJIA		00-0d-56-15-4b-28
192.168.139.25	MATIAGA1		00-0d-56-15-4b-53
192.168.139.27	SNT-DES-SRV		00-02-a5-28-92-36
192.168.139.28	DGSI-INTRANET		00-11-11-6f-a4-3c
192.168.139.30	LIBARRA		00-0d-56-15-4c-65
192.168.139.31	MCHAGNAY		00-0d-56-ca-bc-07
192.168.139.34	EBURBANO		00-10-4b-32-6e-f1
192.168.139.37	CONTRA2		00-60-08-64-23-b0
192.168.139.38	CONTRA3		00-10-4b-32-6d-51
192.168.139.40	RAVILA		00-08-c7-1b-71-a1
192.168.139.41	AESCOBAR		00-80-5f-95-68-18
192.168.139.49	MLOPEZ		00-0d-56-16-2a-12
192.168.139.51	CONTRA5		00-08-c7-c9-e4-b3
192.168.139.20	CCONTRERAS	public	00-10-4b-34-02-d2
192.168.139.39	CGARCES	public	00-0d-56-15-4c-db
192.168.129.11	isa-server.senatel.int		00-02-a5-ec-cb-fb
192.168.129.20	snt-correo-srv.senatel.int		00-50-8b-eb-41-74
192.168.129.30	snt-dominio-srv.senatel.int		00-50-8b-9a-06-b2
192.168.129.40	snt-respaldo.senatel.int		00-0B-c7-ec-8f-6c
192.168.129.50	snt-bdd-srv.senatel.int		00-02-a5-ea-74-0b
192.168.129.52	srvr-saevo.senatel.int		00-50-8b-12-56-02
192.168.129.55	sntprocess.senatel.int		00-0e-7f-b4-07-d0
192.168.129.60	snt-nas-srv.senatel.int		00-11-43-5b-ad-de
192.168.129.70	snt-antivirus.senatel.int		00-50-8b-ea-f1-fb
192.168.129.53	SNT-BASEDATOS	senateluio	00-11-0a-fd-d9-d9
192.168.129.80	MON-SUS-SRV	senateluio	00-50-04-d9-c5-45
192.168.130.13	SRUANO		00-08-c7-d9-08-32
192.168.130.14	DSPERBER		00-06-5b-c8-84-ef

Figura 3. 15 Reporte de Computadores de la SNT del programa NetGraphGold

Impresoras

Dirección IP	Comunidad	Nombre	Marca	Direccion Mac
192.168.139.8	public	NPI85DC15	HP	00-01-e6-85-dc-15
192.168.131.9	senatelcue		HP	00-60-b0-11-14-41
192.168.132.7	public	LXKF032EC	LEXMARK	00-04-00-0f-4c-37
192.168.132.8	public	NPI3D1D3A	HP	00-01-e6-3d-1d-3a
192.168.133.6	public	LXKF032D5	LEXMARK	00-04-00-0f-4c-ab
192.168.133.9	public	NPI5B7336	HP	00-01-e6-5b-73-36
192.168.133.41	public	NPICBCEA0	HP	00-30-c1-cb-ce-a0
192.168.134.9	public	NPI6088B0	HP	00-01-e6-60-b8-b0
192.168.135.8	public	LXKF032C4	LEXMARK	00-04-00-0f-4c-23
192.168.136.9	public	LXK1FB4A9	LEXMARK	00-04-00-f8-2d-95
192.168.137.3	public	LXKF0329C	LEXMARK	00-04-00-0f-4c-39
192.168.138.8	senatelcue		HP	00-60-b0-11-34-6d
192.168.138.9	public	NPI5B78DE	HP	00-01-e6-5b-78-de
192.168.140.9	public	NPI3526AE	HP	00-01-e6-35-26-ae
192.168.160.8	public	NPI505165	HP	00-01-e6-50-51-65
192.168.160.9	public	NPID6F22C	HP	00-11-85-d6-f2-2c
192.168.160.10	public	NPID66281	HP	00-11-85-d6-62-81

OK

Figura 3. 16 Reporte de Impresoras de la SNT del programa NetGraphGold

Ruteadores

Nombre	Comunidad	Direcciones IP	Interfaz	Tipo de Enlace
WWW	senateluio	192.168.1.1	FastEthernet0/0	ethernetCsmacd
		64.76.199.18	Serial0/0	FrameRelay
CUENCA	senateluio	192.168.162.3	Ethernet0	ethernetCsmacd
		192.168.151.1	Serial0	FrameRelay
Cisco_Cuenca	senatelcue	192.168.150.1	Ethernet0	ethernetCsmacd
		192.168.151.2	Serial0	FrameRelay
Cisco_Guayas	senatelgye	192.168.160.1	Ethernet0	ethernetCsmacd
		192.168.161.2	Serial0	PPP
GYE	senateluio	192.168.162.2	Ethernet0	ethernetCsmacd
		192.168.161.1	Serial0	PPP

OK

Figura 3. 17 Reporte de Ruteadores de la SNT del programa NetGraphGold

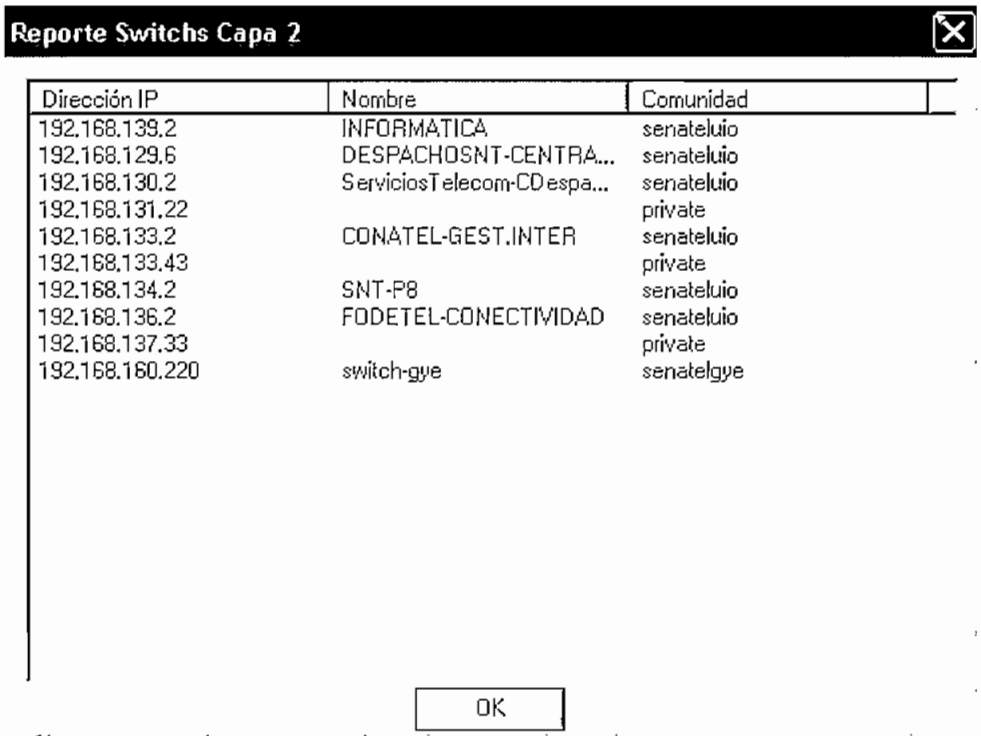


Figura 3. 18 Reporte de Switch Capa2 de la SNT del programa NetGraphGold

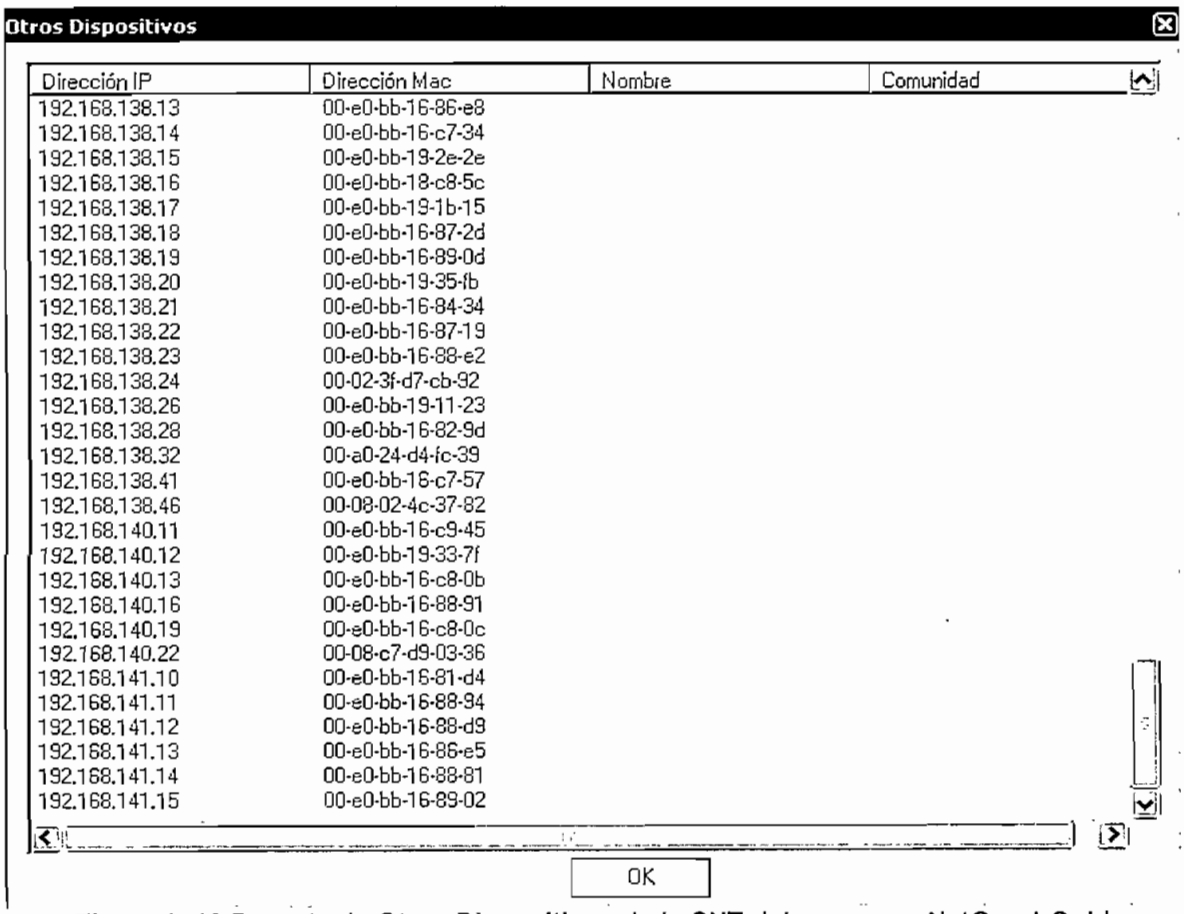


Figura 3. 19 Reporte de Otros Dispositivos de la SNT del programa NetGraphGold

Y el gráfico obtenido de la red de La Secretaría Nacional de Telecomunicaciones se lo indica de manera completa en la figura 3.20, pudiendo distinguirse con claridad que se trata de una topología tipo estrella con centro en un Switch Capa3 que hace la función de núcleo de la red y una distribución de subredes alrededor, con dispositivos por cada subred creada y VLAN correspondiente a cada subred.

Se puede también observar que cada subred, tiene elementos dentro de sí en el árbol de organización de dispositivos que se encuentra a la izquierda del gráfico, la cual permite una mejor exploración de los mismos con solo expandir la subred seleccionada. Cabe señalar que al escoger la subred en el árbol de organización, inmediatamente aparecerá su respectivo gráfico en el visor de gráficos de red.

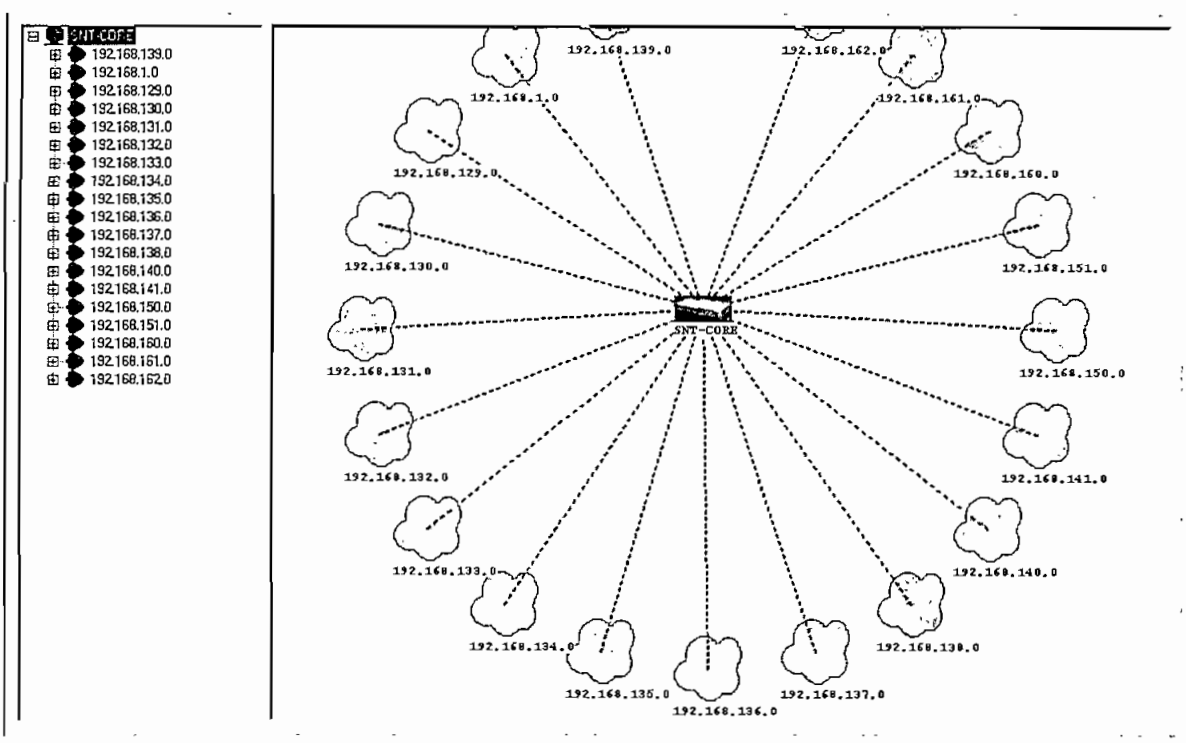


Figura 3. 20 Gráfico General de la Red de la SNT en el programa NetGraphGold

Escogiendo la primera subred que aparece en el árbol de organización de dispositivos; que para cualquier caso pasa a ser la subred local donde se encontraba el host de prueba, y en este caso en particular es la subred 192.168.139.0 que pertenece a la VLAN 112 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.21, la cual contiene un switch capa2, 19 computadores, 1 impresora de red y 12 dispositivos varios, todos distribuido alrededor del switch capa2, el cual se conecta directamente con el núcleo de la red.

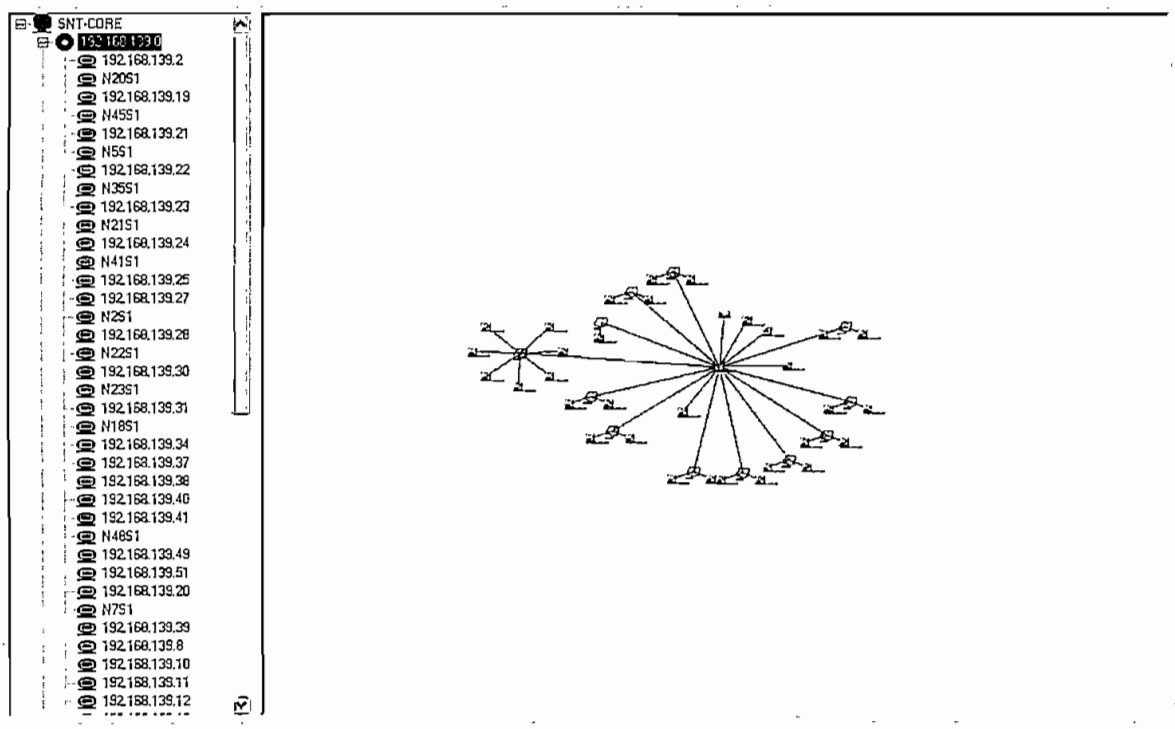


Figura 3. 21 Subred 192.168.139.0 de la SNT en el programa NetGraphGold

Escogiendo la siguiente subred que aparece en el árbol de organización de dispositivos; que en este caso es la subred 192.168.1.0 que pertenece a la VLAN 102 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.22, la cual contiene un ruteador que permite la comunicación con el Internet mediante un enlace serial de tipo FrameRelay (figura 3.17), y el cual se conecta directamente con el núcleo de la red.

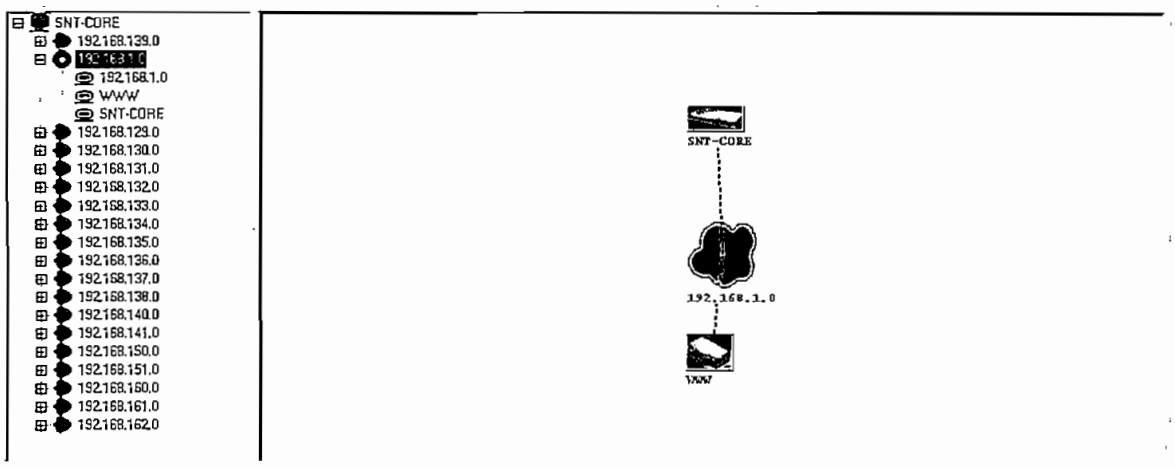


Figura 3. 22 Subred 192.168.1.0 de la SNT en el programa NetGraphGold

Escogiendo la siguiente subred que aparece en el árbol de organización de dispositivos; que en este caso es la subred 192.168.129.0 que pertenece a la VLAN 101 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.23, la cual contiene un switch capa2, 11 computadores y 15 dispositivos varios, todos distribuidos alrededor del switch capa2, el cual se conecta directamente con el núcleo de la red.

Eligiendo la siguiente subred que aparece en el árbol de organización de dispositivos; que en este caso es la subred 192.168.130.0 que pertenece a la VLAN 103 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.24, la cual contiene un switch capa2, 3 computadores y 4 dispositivos varios, todos distribuidos alrededor del switch capa2, el cual se conecta directamente con el núcleo de la red.

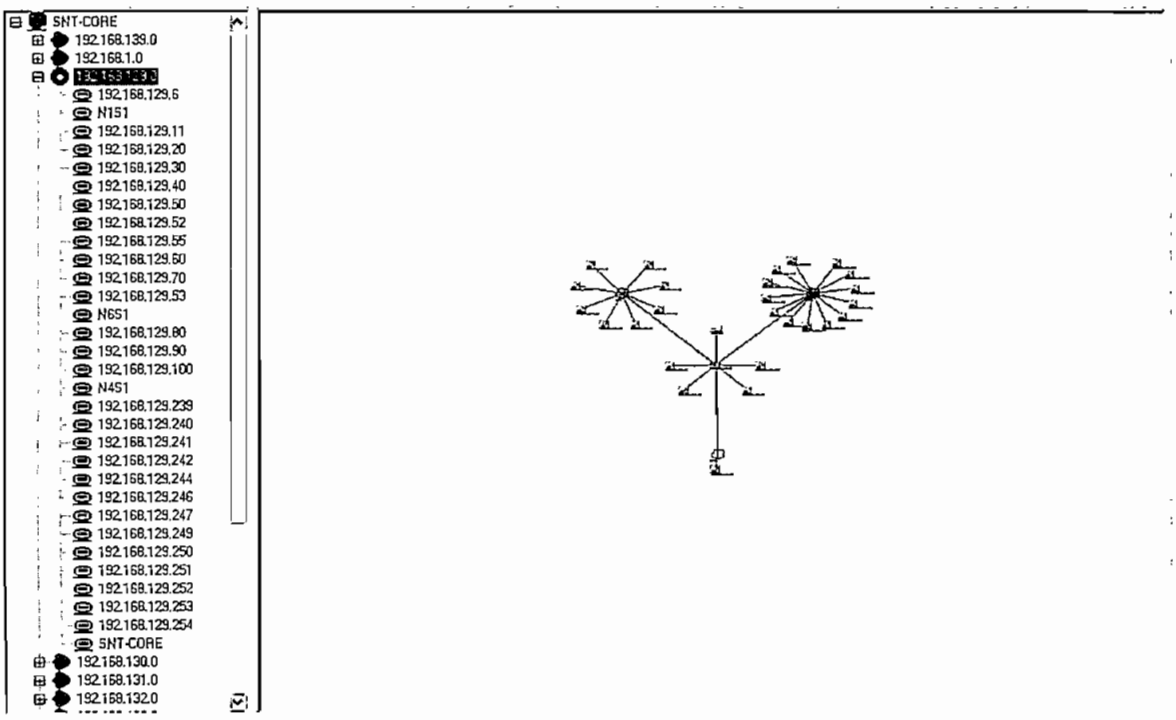


Figura 3. 23 Subred 192.168.129.0 de la SNT en el programa NetGraphGold

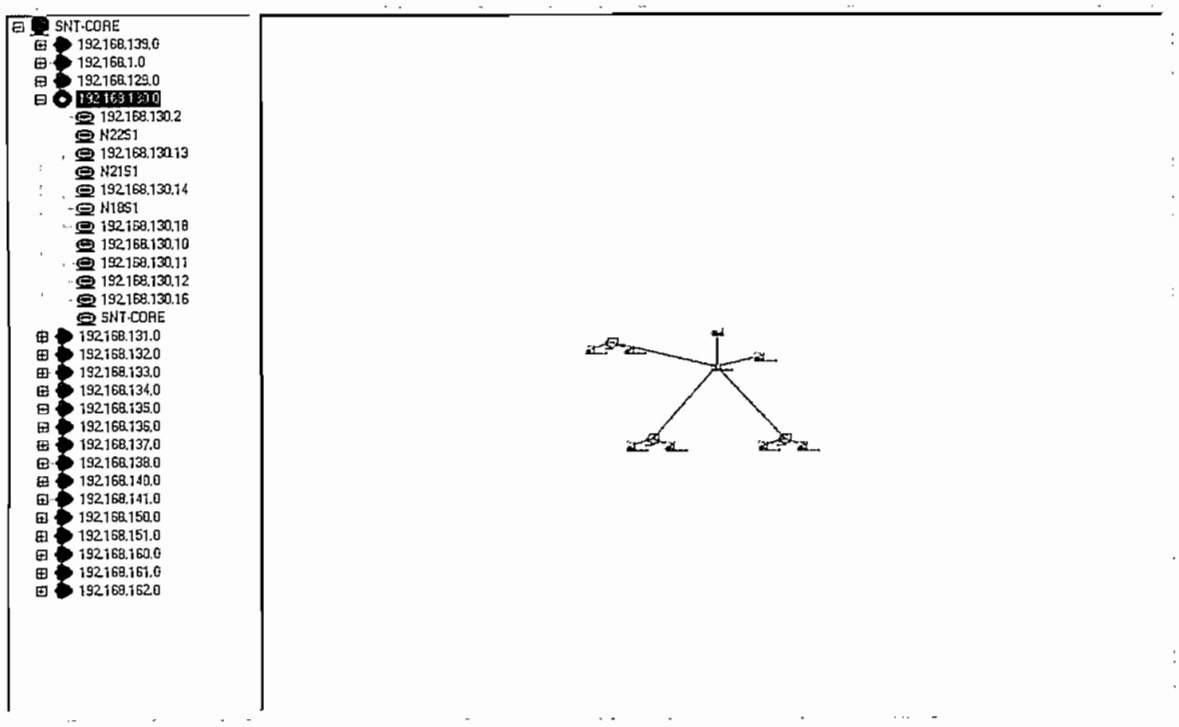


Figura 3. 24 Subred 192.168.130.0 de la SNT en el programa NetGraphGold

Optando por la siguiente subred que aparece en el árbol de organización de dispositivos; que en este caso es la subred 192.168.131.0 que pertenece a la VLAN 104 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.25, la cual contiene un switch capa2, 4 computadores, una impresora de red y 6 dispositivos varios, todos distribuidos alrededor del switch capa2, el cual se conecta directamente con el núcleo de la red.

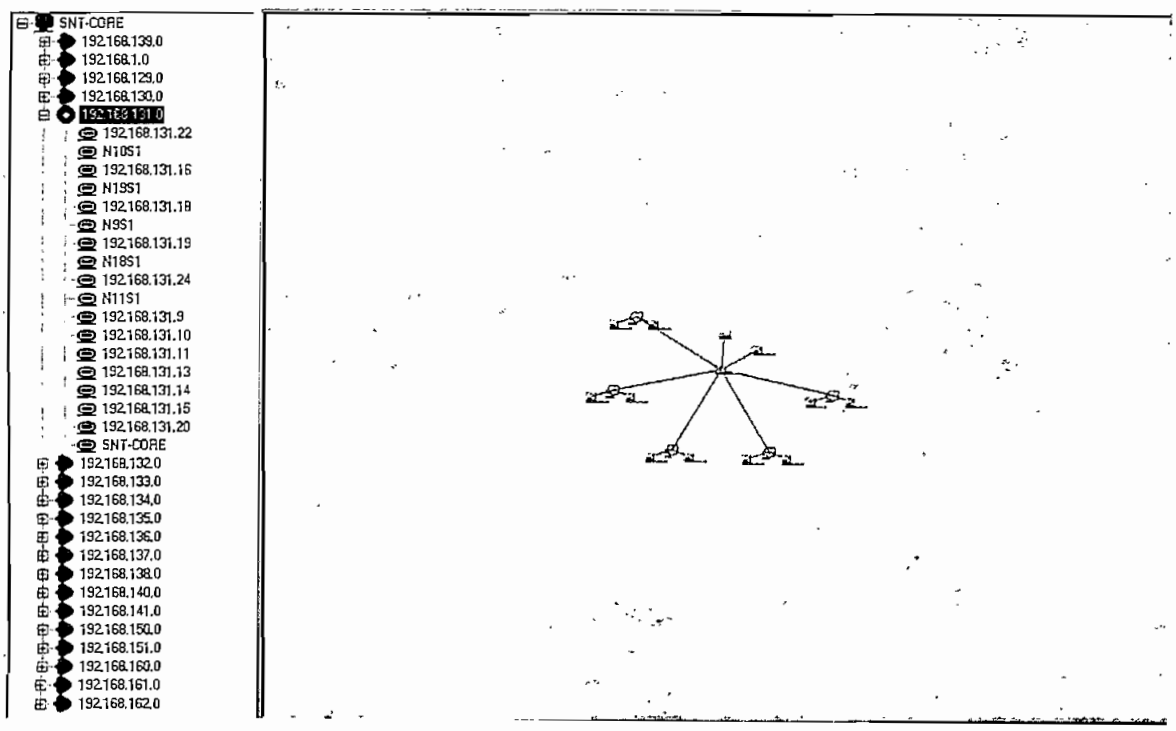


Figura 3. 25 Subred 192.168.131.0 de la SNT en el programa NetGraphGold

Escogiendo la siguiente subred que aparece en el árbol de organización de dispositivos; que en este caso es la subred 192.168.132.0 que pertenece a la VLAN 105 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.26, la cual contiene 5 computadores, 2 impresoras y 6 dispositivos varios, todos distribuidos alrededor de un switch capa2 no administrable o un hub representado por una nube, el cual se conecta a su vez con el núcleo de la red.

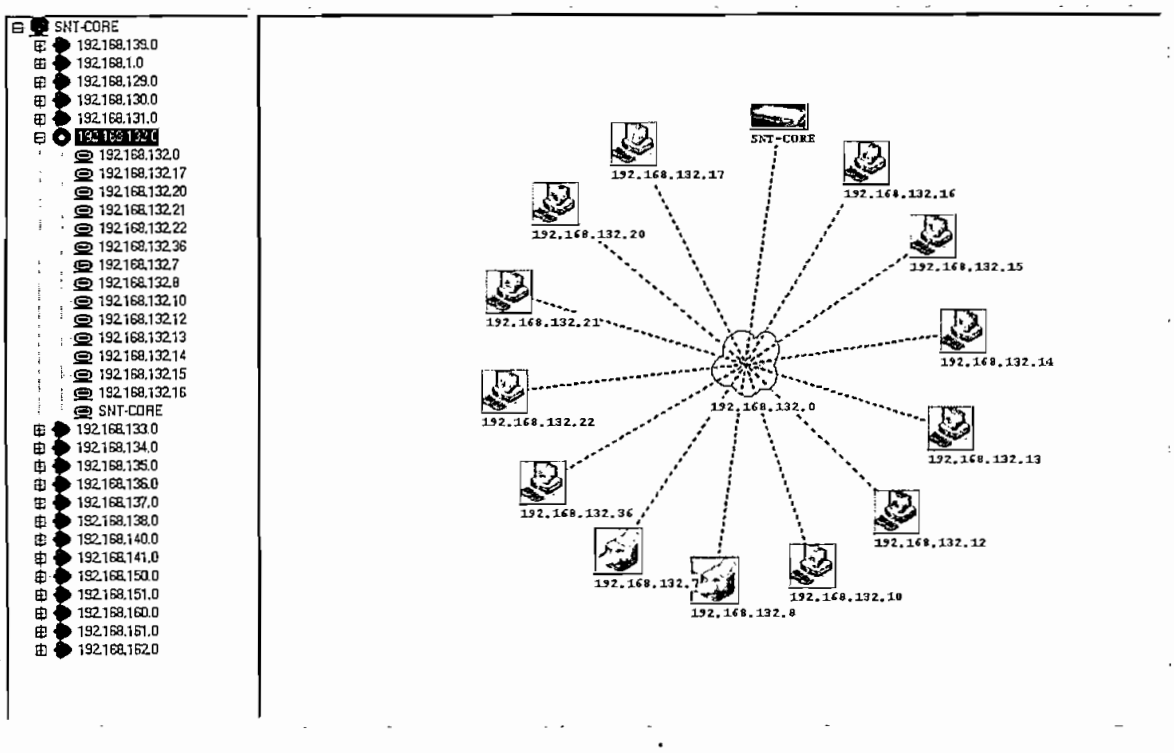


Figura 3. 26 Subred 192.168.132.0 de la SNT en el programa NetGraphGold

Eligiendo la siguiente subred que aparece en el árbol de organización de dispositivos; que en este caso es la subred 192.168.133.0 que pertenece a la VLAN 106 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.27, la cual contiene 2 switches capa2, 12 computadores, 3 impresoras de red y 16 dispositivos varios, todos distribuidos en ambos switches, los cuales se conectan entre sí y uno de ellos directamente con el núcleo de la red.

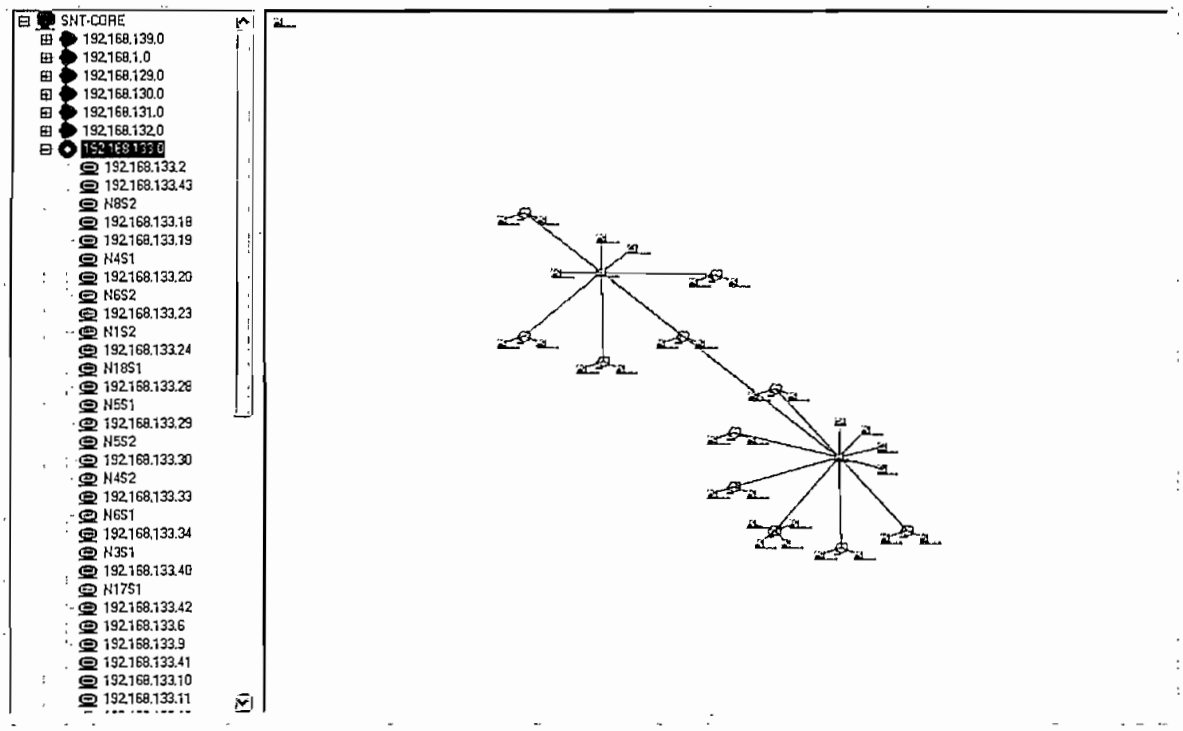


Figura 3. 27 Subred 192.168.133.0 de la SNT en el programa NetGraphGold

Optando por la siguiente subred que aparece en el árbol de organización de dispositivos; que en este caso es la subred 192.168.134.0 que pertenece a la VLAN 107 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.28, la cual contiene un switch capa2, 8 computadores, una impresora de red y 9 dispositivos varios, todos distribuidos alrededor del switch capa2, el cual se conecta directamente con el núcleo de la red.

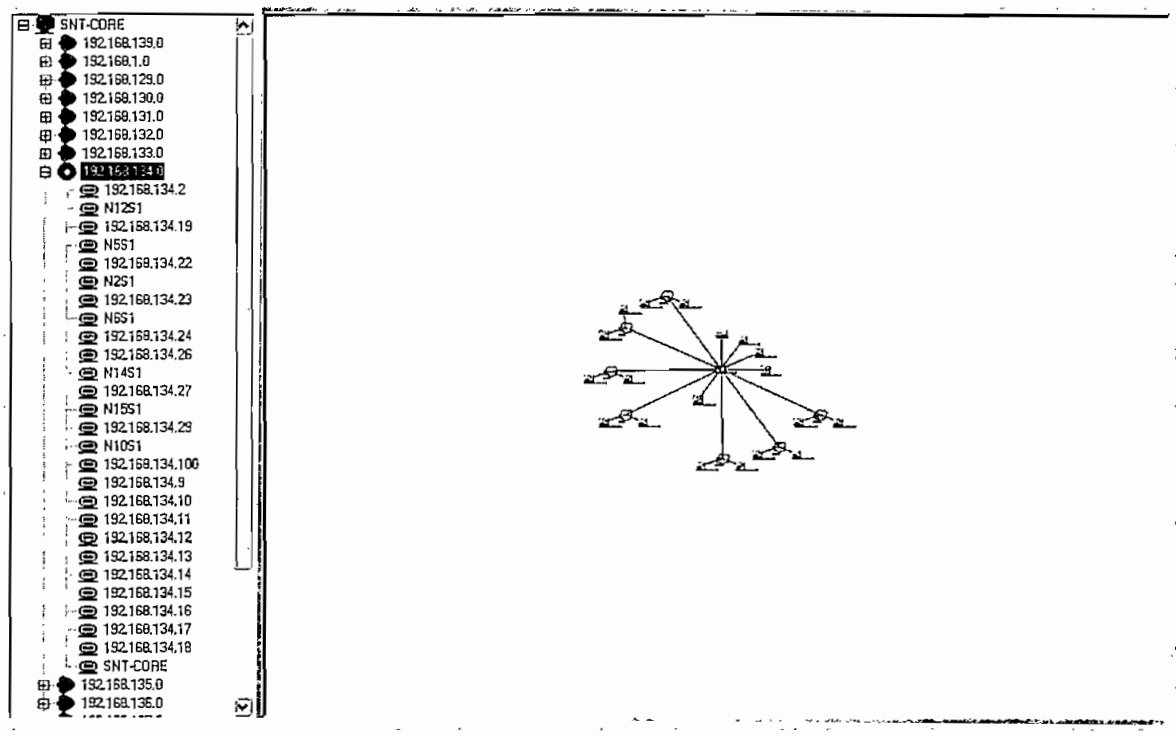


Figura 3. 28 Subred 192.168.134.0 de la SNT en el programa NetGraphGold

Escogiendo la siguiente subred que aparece en el árbol de organización de dispositivos; que en este caso es la subred 192.168.135.0 que pertenece a la VLAN 108 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.29, la cual contiene 11 computadores, una impresora de red y 9 dispositivos varios, todos distribuidos alrededor de un switch capa2 no administrable o un hub representado por una nube, el cual se conecta a su vez con el núcleo de la red.

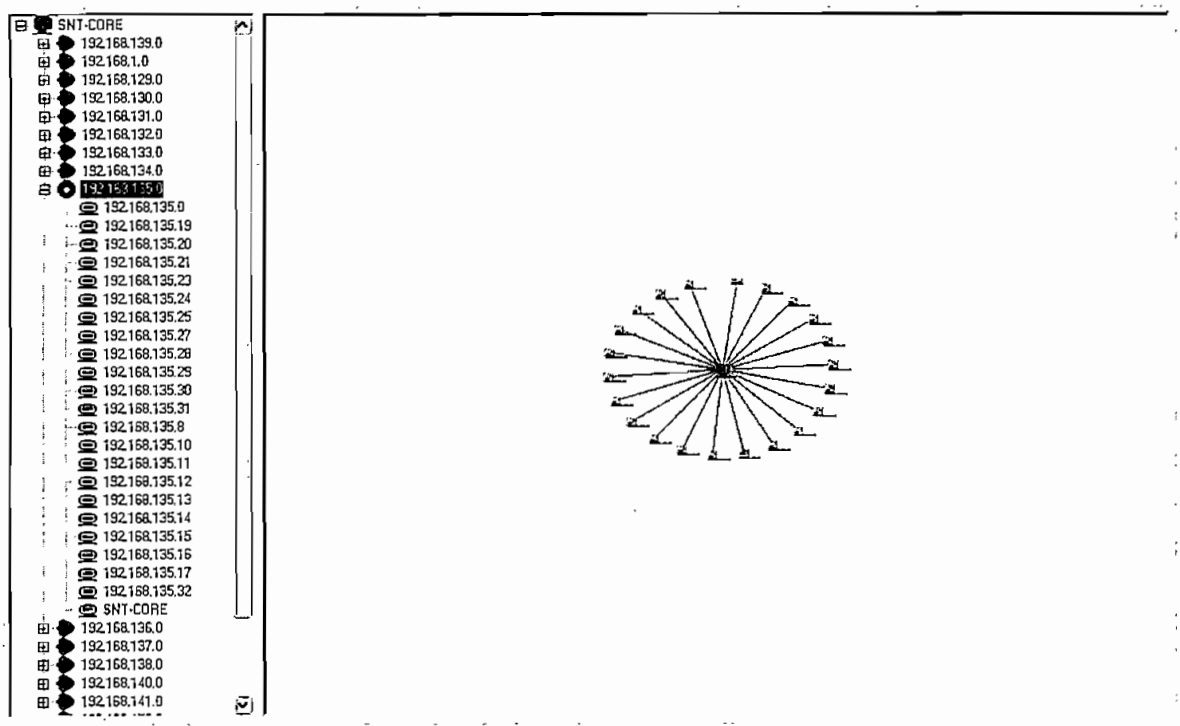


Figura 3. 29 Subred 192.168.135.0 de la SNT en el programa NetGraphGold

Eligiendo la siguiente subred que aparece en el árbol de organización de dispositivos; que en este caso es la subred 192.168.136.0 que pertenece a la VLAN 109 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.30, la cual contiene 1 switch capa2, 9 computadores, una impresora de red y 9 dispositivos varios, todos distribuidos alrededor del switch capa2, el cual se conecta directamente con el núcleo de la red.

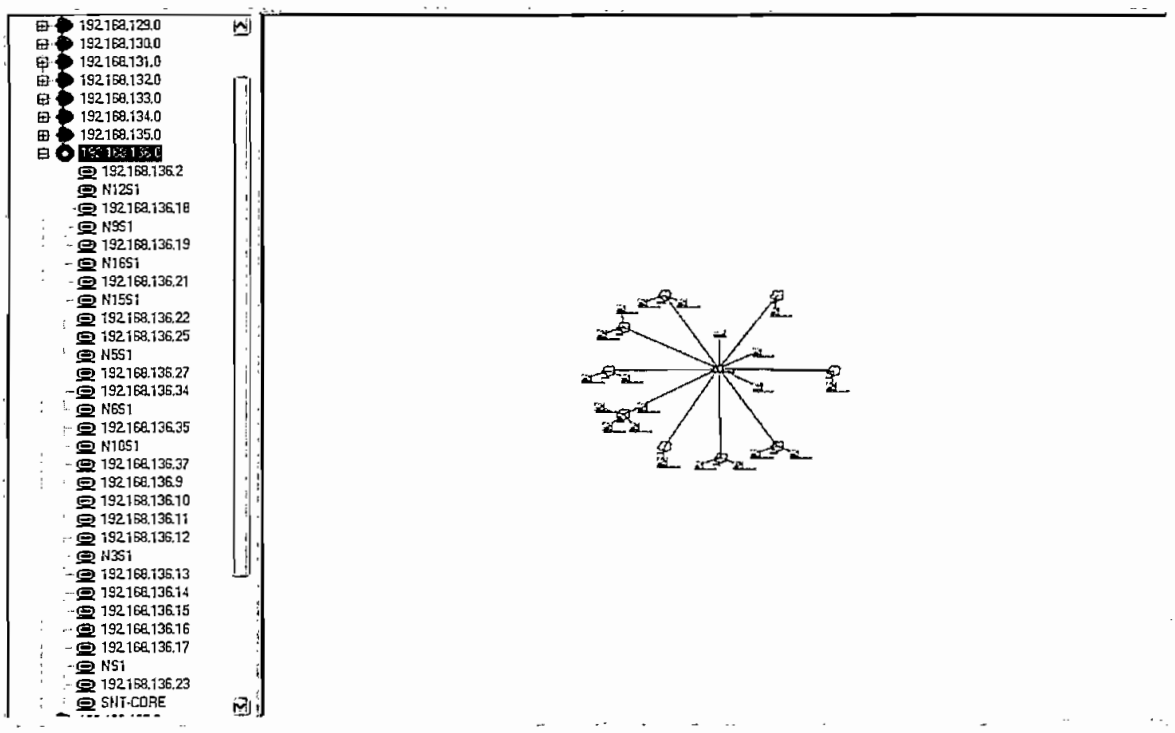


Figura 3. 30 Subred 192.168.136.0 de la SNT en el programa NetGraphGold

Optando por la siguiente subred que aparece en el árbol de organización de dispositivos; que en este caso es la subred 192.168.137.0 que pertenece a la VLAN 110 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.31, la cual contiene un switch capa2, 34 computadores, una impresora de red y 35 dispositivos varios, todos distribuidos alrededor del switch capa2, el cual se conecta directamente con el núcleo de la red. Cabe señalar que existe una concentración de varios elementos en un solo puerto del switch central, por lo que se entiende que en ese puerto existe un switch capa2 no administrable o un hub de al menos 48 puertos que contiene a los dispositivos allí conectados.

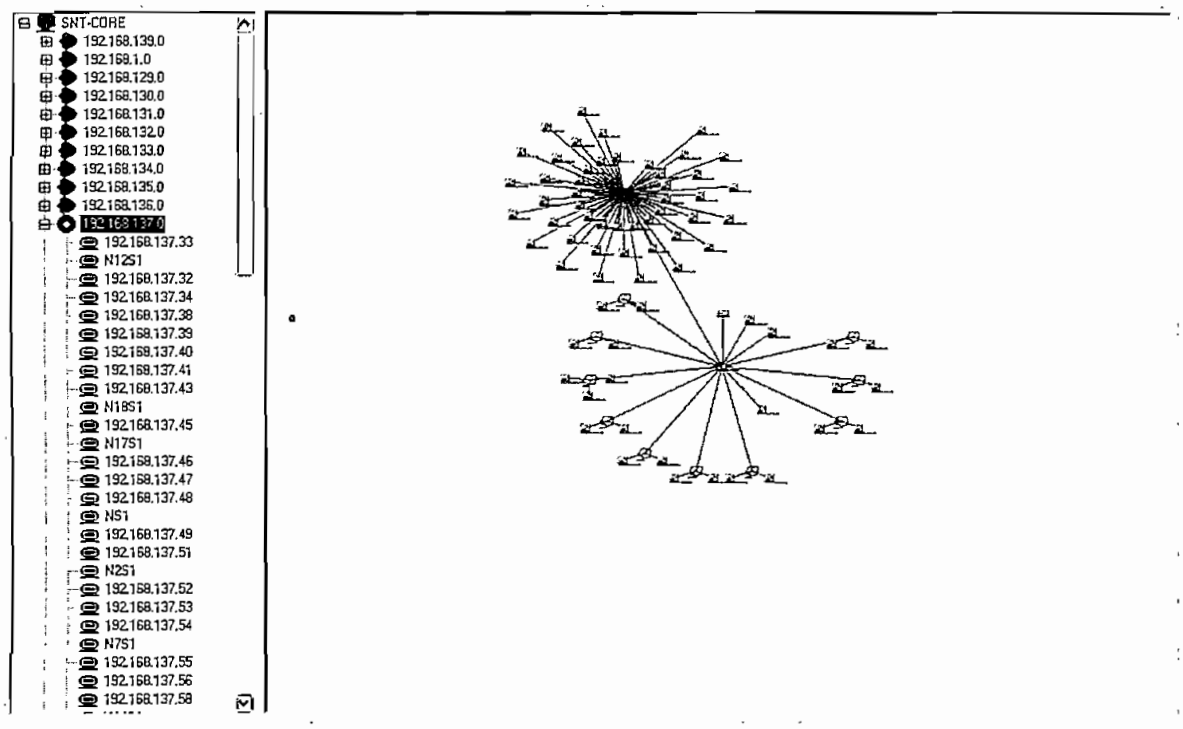


Figura 3. 31 Subred 192.168.137.0 de la SNT en el programa NetGraphGold

Escogiendo la siguiente subred que aparece en el árbol de organización de dispositivos; que en este caso es la subred 192.168.138.0 que pertenece a la VLAN 111 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.32, la cual contiene 18 computadores, 2 impresoras de red y 20 dispositivos varios, todos distribuidos alrededor de un switch capa2 no administrable o un hub de al menos 48 puertos, representado por una nube, el cual se conecta a su vez con el núcleo de la red.

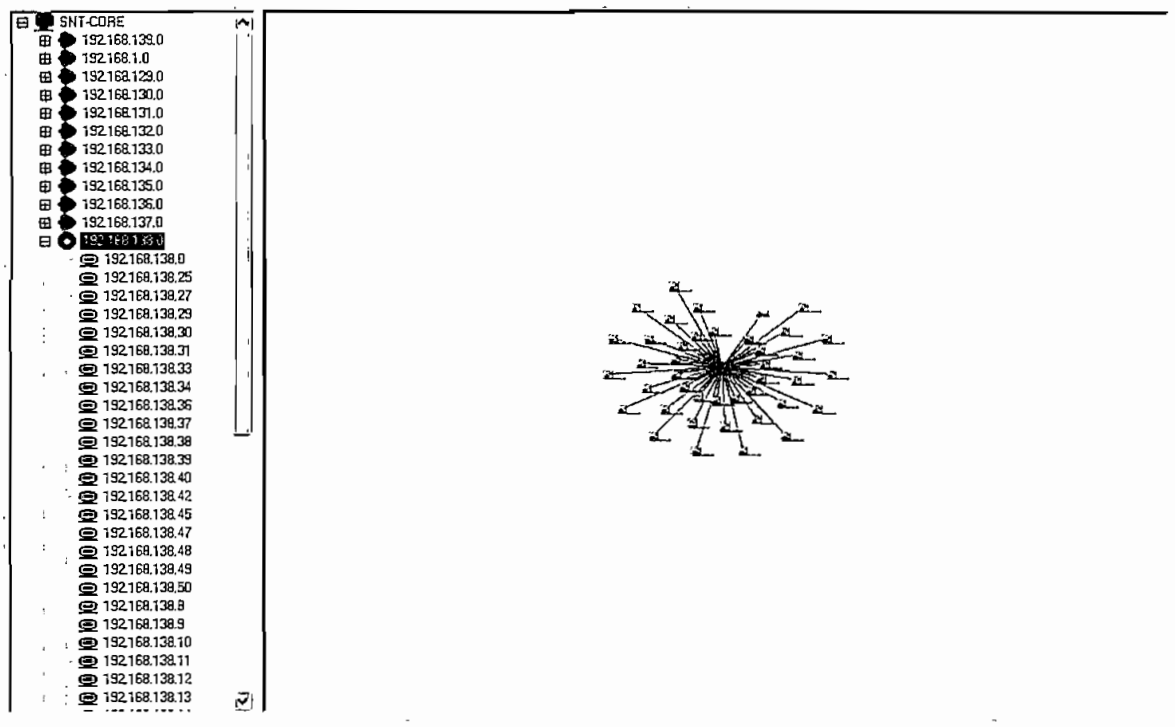


Figura 3. 32 Subred 192.168.138.0 de la SNT en el programa NetGraphGold

Eligiendo la siguiente subred que aparece en el árbol de organización de dispositivos; que en este caso es la subred 192.168.140.0 que pertenece a la VLAN 113 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.33, la cual contiene 4 computadores, una impresora de red y 6 dispositivos varios, todos distribuidos alrededor de un switch capa2 no administrable o un hub de al menos 24 puertos, representado por una nube, el cual se conecta a su vez con el núcleo de la red.

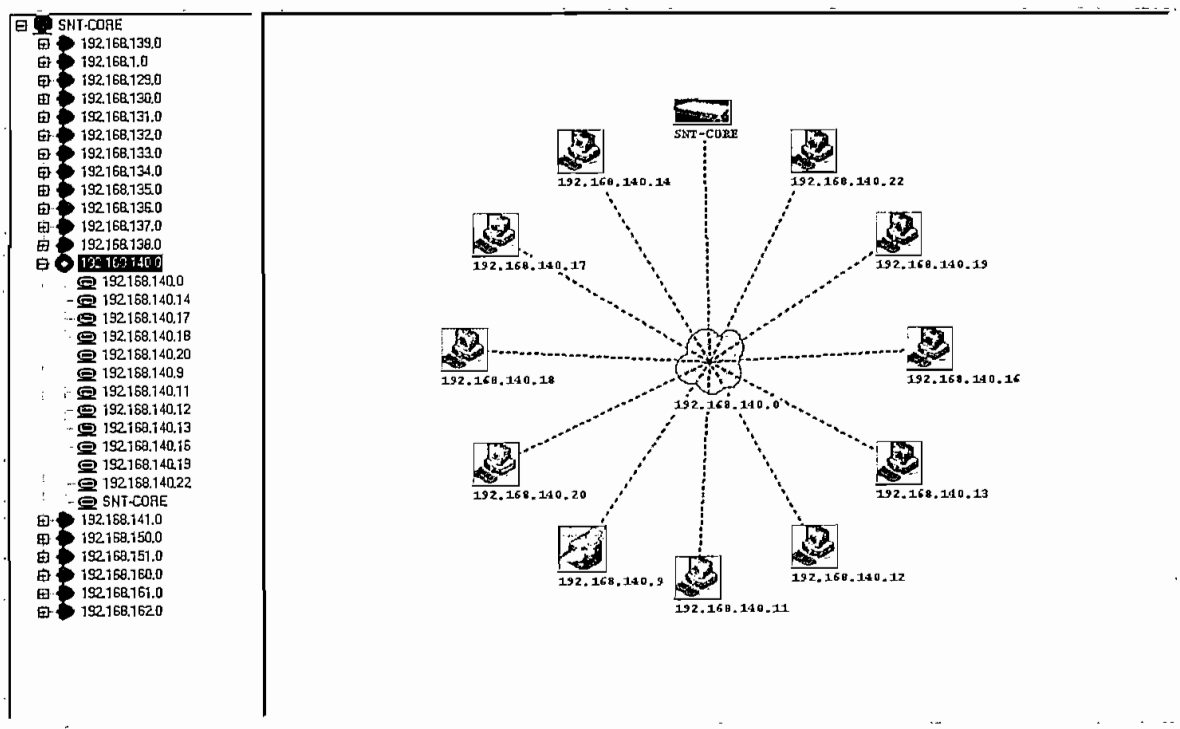


Figura 3. 33 Subred 192.168.140.0 de la SNT en el programa NetGraphGold

Optando por la siguiente subred que aparece en el árbol de organización de dispositivos; que en este caso es la subred 192.168.141.0 que pertenece a la VLAN 114 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.34, la cual contiene 13 computadores y 16 dispositivos varios, todos distribuidos alrededor de un switch capa2 no administrable o un hub de al menos 48 puertos, representado por una nube, el cual se conecta a su vez con el núcleo de la red.

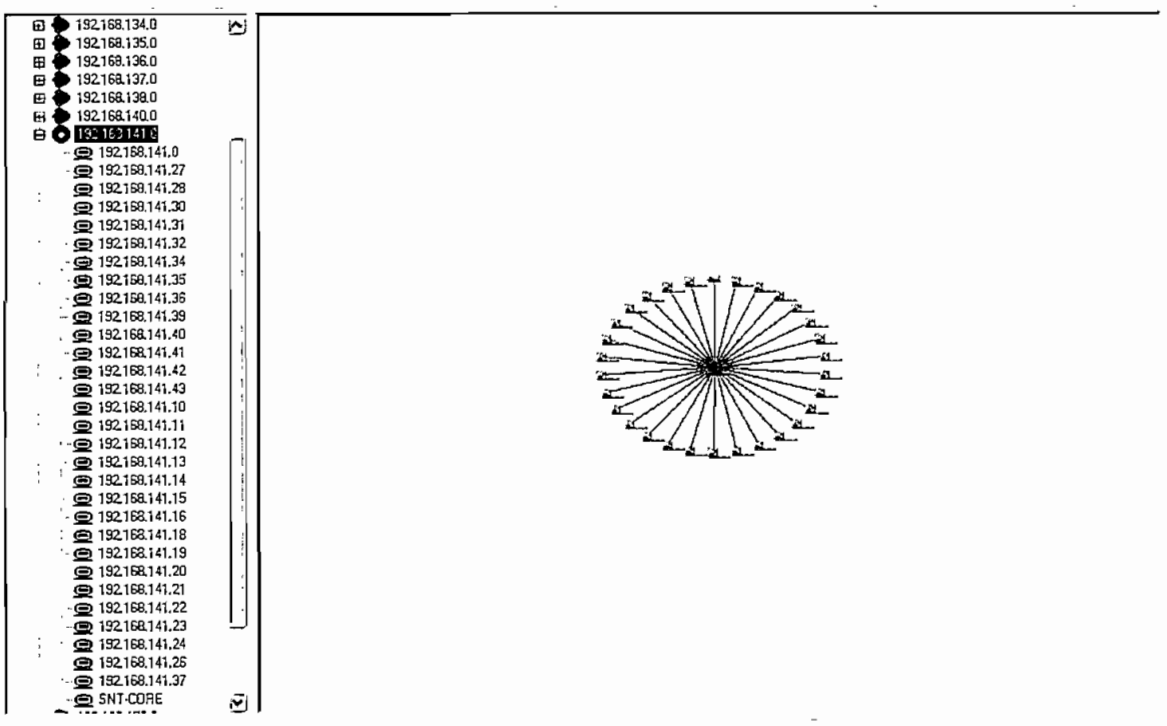


Figura 3. 34 Subred 192.168.141.0 de la SNT en el programa NetGraphGold

Escogiendo la siguiente subred que aparece en el árbol de organización de dispositivos; que en este caso es la subred 192.168.150.0 que pertenece a la VLAN 116 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.35, la cual contiene un ruteador y 3 dispositivos varios, todos distribuidos alrededor de un switch capa2 no administrable o un hub de al menos 12 puertos, representado por una nube, el cual se conecta a su vez con el ruteador. En ésta subred se tiene que los elementos se comunican con el núcleo a través del ruteador, puesto que en realidad es una de las regionales de la Secretaría Nacional de Telecomunicaciones en Cuenca, que se enlaza a su vez con la matriz en Quito a través de una red WAN tipo PPP (figura 3.17), como podremos ver en las siguientes subredes.

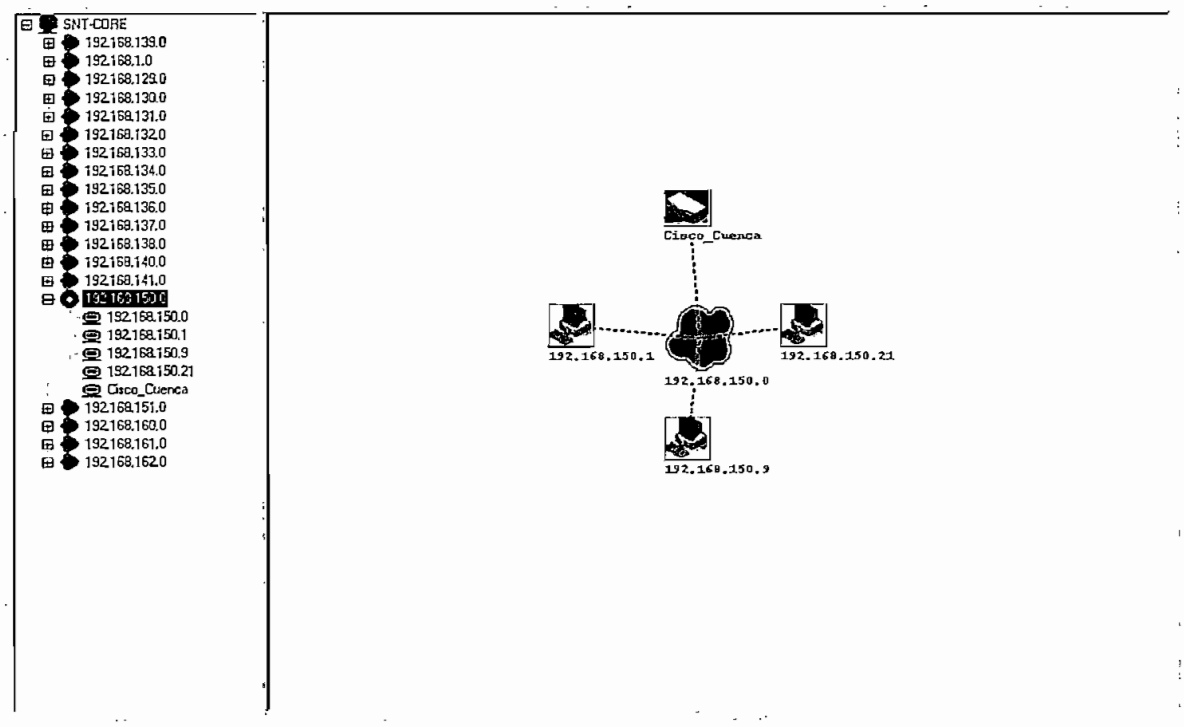


Figura 3. 35 Subred 192.168.150.0 de la SNT en el programa NetGraphGold

Las subredes 192.168.151.0 y 192.168.161.0 que pertenecen a la VLAN 116 (figura 3.14), presentan únicamente 2 ruteadores cada una, todos distribuidos alrededor de una nube que representa la conexión entre ellos; es decir, los enlaces alquilados a alguna empresa que les permite comunicarse entre las regionales de la Secretaría Nacional de Telecomunicaciones en Guayaquil y Cuenca con su matriz en Quito respectivamente.

Los enlaces establecidos entre las diferentes regionales son: SNT Quito – SNT Cuenca (figura 3.36) mediante FrameRelay (figura 3.17) y SNT Quito – SNT Guayaquil (figura 3.37) mediante PPP (figura 3.17).

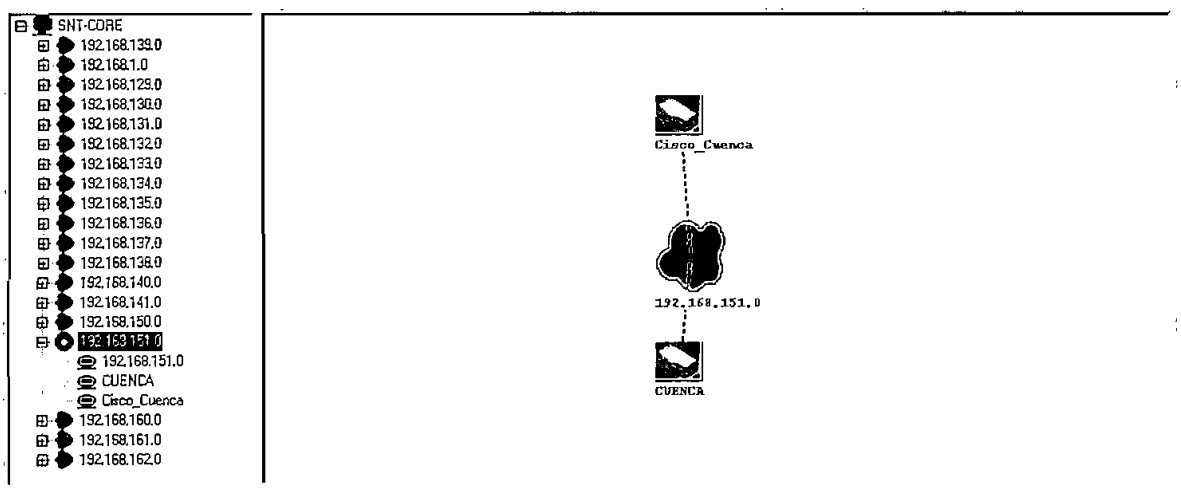


Figura 3. 36 Subred 192.168.151.0 de la SNT en el programa NetGraphGold

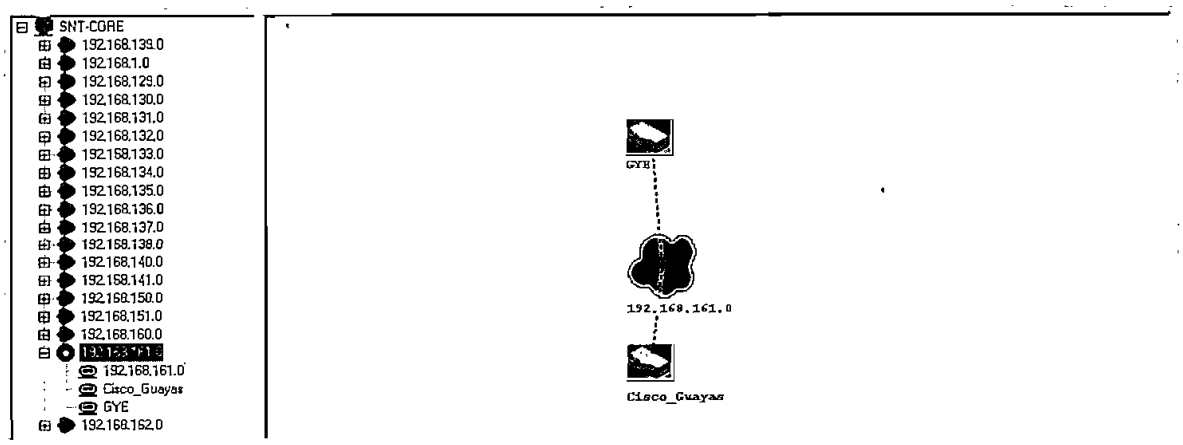


Figura 3. 37 Subred 192.168.161.0 de la SNT en el programa NetGraphGold

Eligiendo la subred 192.168.160.0 que pertenece a la VLAN 116 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.38, la cual contiene un switch capa2, un ruteador, 2 computadores, 3 impresoras de red y 2 dispositivos varios, todos distribuidos alrededor de un switch capa2 de al menos 24 puertos, el cual se conecta a su vez con el ruteador de la regional de la SNT en Guayaquil, que le permite comunicarse a través de un enlace PPP (figura 3.17) con la matriz de la SNT en Quito.

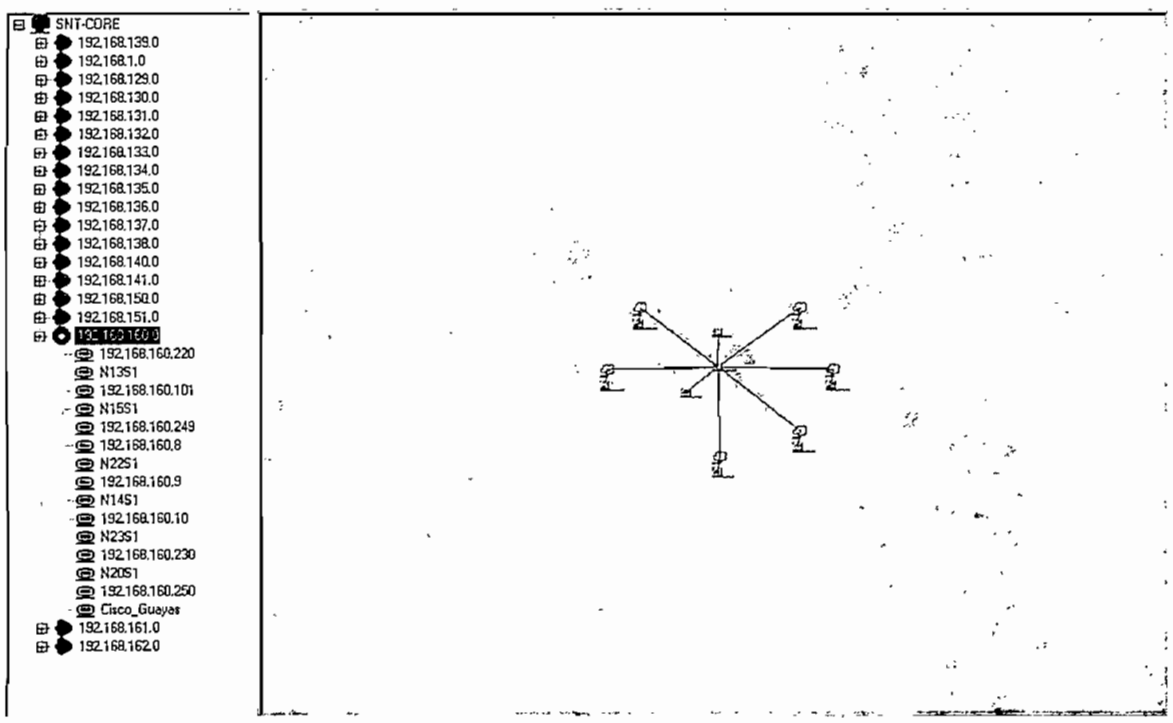


Figura 3. 38 Subred 192.168.160.0 de la SNT en el programa NetGraphGold

Eligiendo la subred 192.168.162.0 que pertenece a la VLAN 116 (figura 3.14), obtenemos el gráfico y los dispositivos contenidos en el mismo presentados en la figura 3.39, la cual contiene 2 ruteadores y un dispositivo vario, todos distribuidos alrededor de un switch capa2 no administrable de al menos 12 puertos, representado por una nube, el cual se conecta a su vez con el núcleo de la red. Ésta subred representa la conexión existente entre la matriz de la SNT en Quito con la regionales tanto en Cuenca como en Guayaquil.

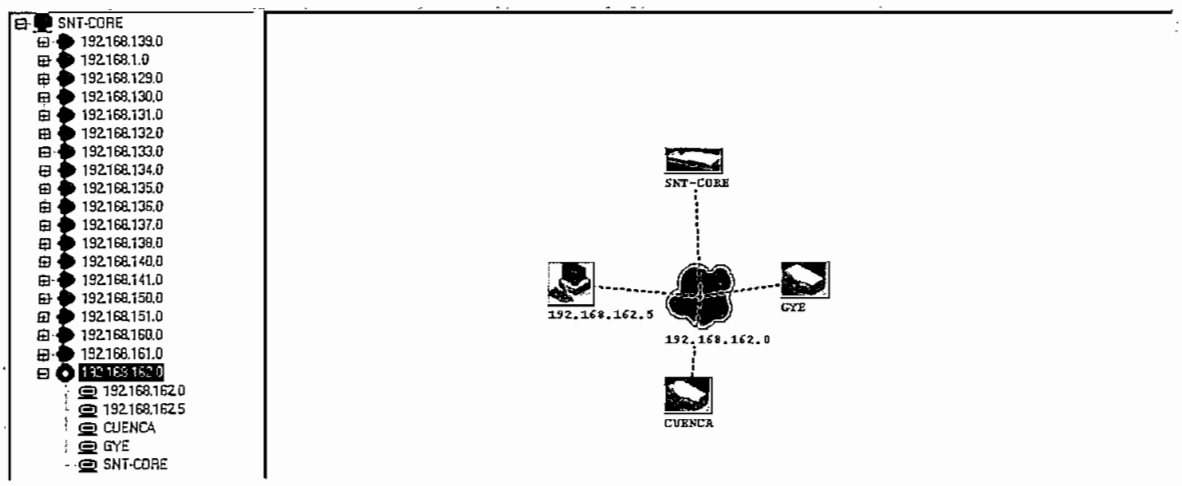


Figura 3. 39 Subred 192.168.162.0 de la SNT en el programa NetGraphGold.

3.4 LAS HERRAMIENTAS ADICIONALES

Las herramientas diseñadas para la ayuda en el monitoreo de una red, permiten medir tiempos de respuesta de un dispositivo, resuelven nombres de hosts o sus direcciones IP, miden el tráfico que cursa por un ruteador, o simplemente en base a preguntas específicas en SNMP, averiguan otros tipos de datos que el consultor diseñado para ello permita realizar y el dispositivo en cuestión, pueda responder.

Por ello las herramientas creadas para el programa NetGraphGold son:

DNSResolver

Esta herramienta permite resolver un nombre de un host dada su dirección IP o su dirección IP dado el nombre del host.

La figura 3.40 muestra las partes de la herramienta mencionada.

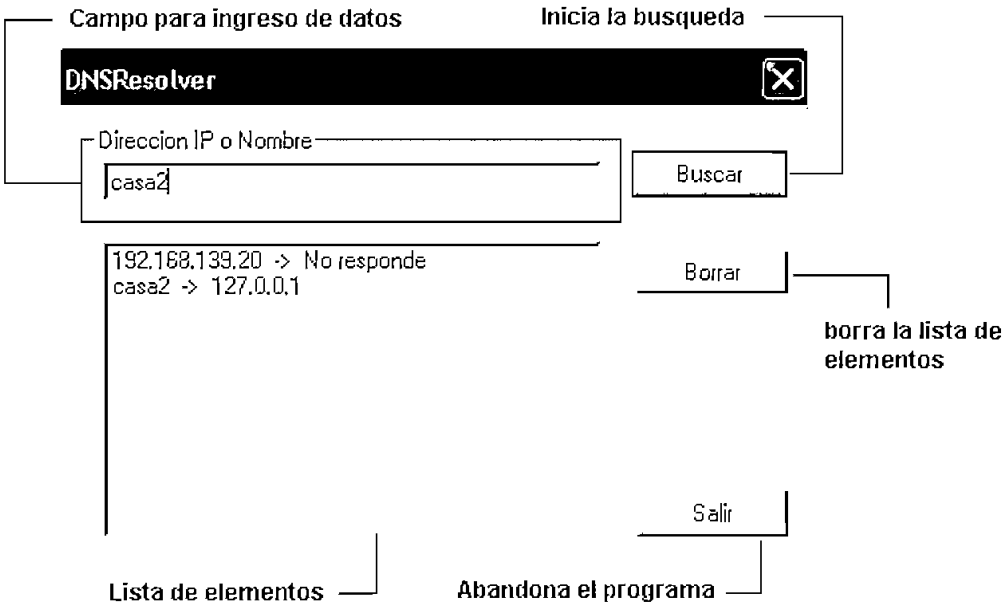


Figura 3. 40 Herramienta DNSResolver del programa NetGraphGold.

Ping

Esta herramienta permite medir el tiempo de respuesta, a la contestación de paquetes de echo enviados hacia una dirección IP específica.

La figura 3.41 muestra las partes de la herramienta mencionada.

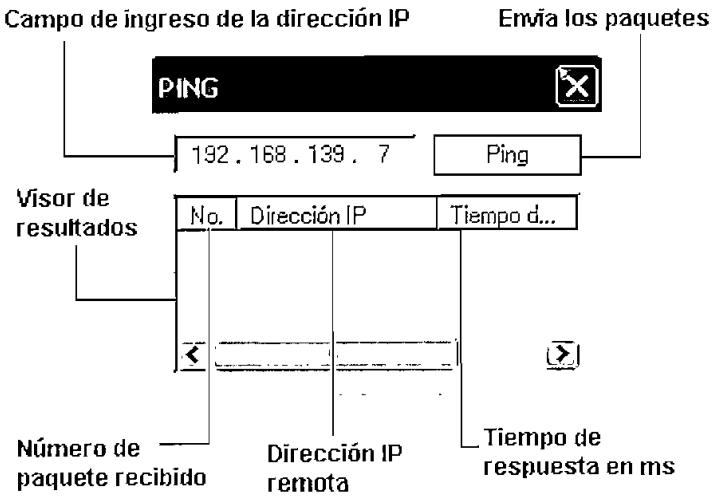


Figura 3. 41 Herramienta Ping del programa NetGraphGold.

Tracert

Esta herramienta permite determinar el número de saltos que un dispositivo realizar con un máximo de 30, y la ruta que traza para alcanzar una dirección IP específica.

La figura 3.42 muestra las partes de la herramienta mencionada.

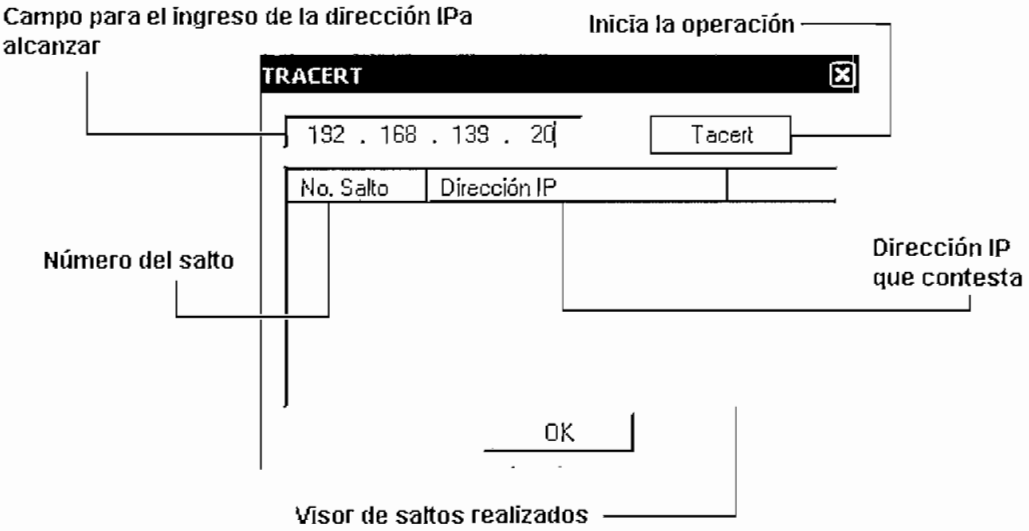


Figura 3. 42 Herramienta Tracert del programa NetGraphGold.

Buscador de OUIs

Esta herramienta permite determinar el fabricante de una interfaz de red específica, dada la dirección MAC de la misma.

La figura 3.43 muestra las partes de la herramienta mencionada.

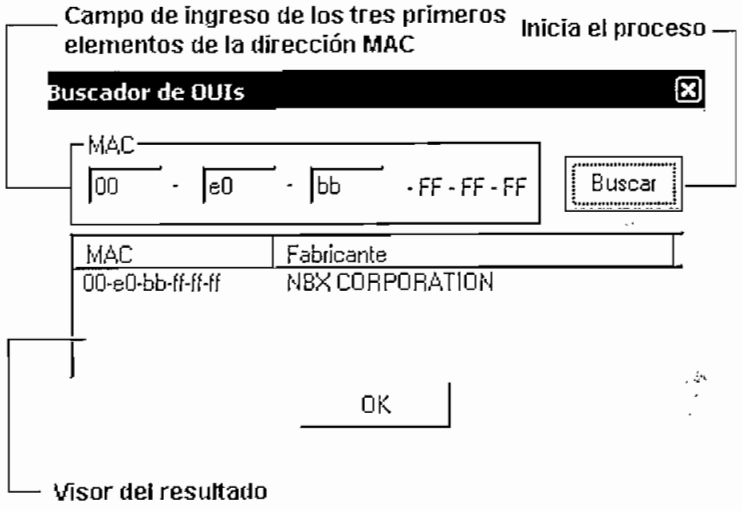


Figura 3. 43 Herramienta Buscador de OUIs del programa NetGraphGold.

SNMP

Esta herramienta permite realizar preguntas de tipo SNMP a un dispositivo que soporte dicho protocolo, ya sean éstas tomadas del árbol OID contenido en la herramienta o arbitrarias ingresadas en el campo destinado para ello.

La figura 3.44 muestra la herramienta mencionada.

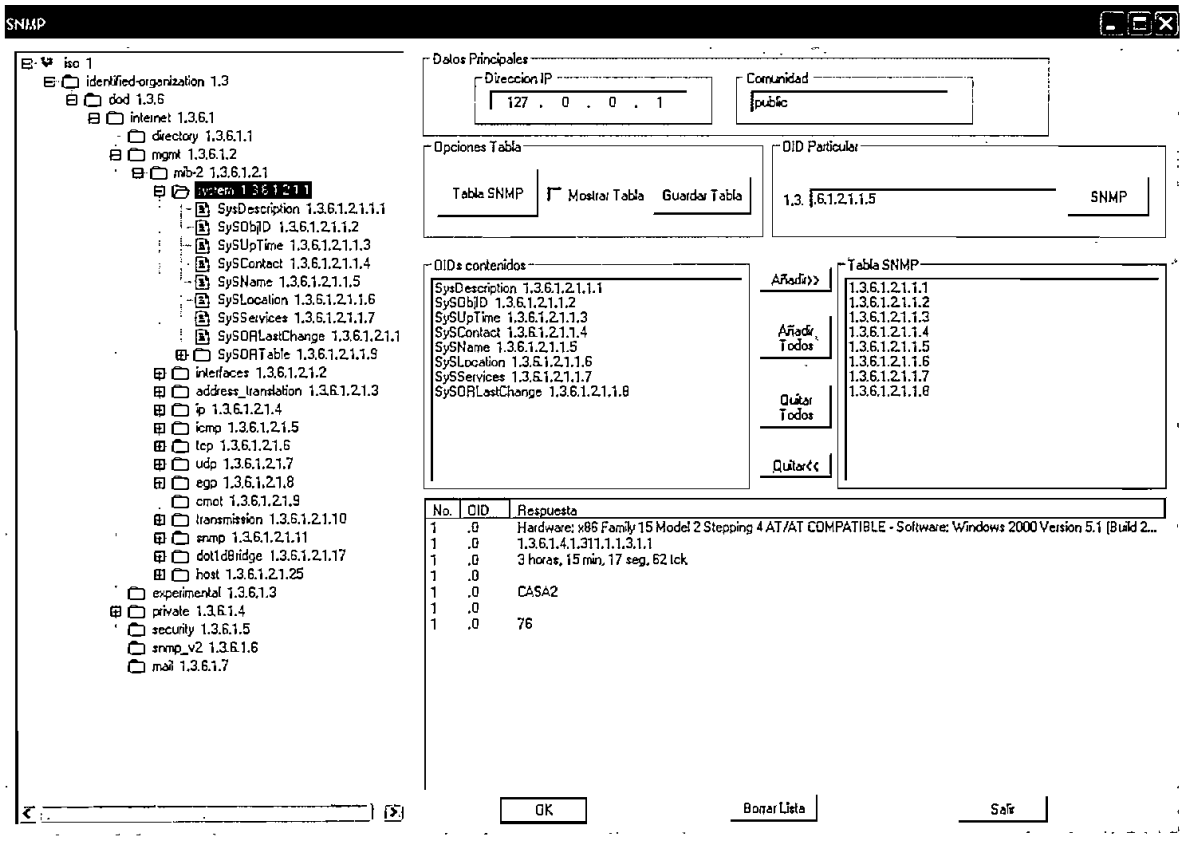


Figura 3. 44 Herramienta SNMP del programa NetGraphGold.

Las partes de esta herramienta se detallan a continuación:

El campo de ingreso de datos en la figura 3.45.

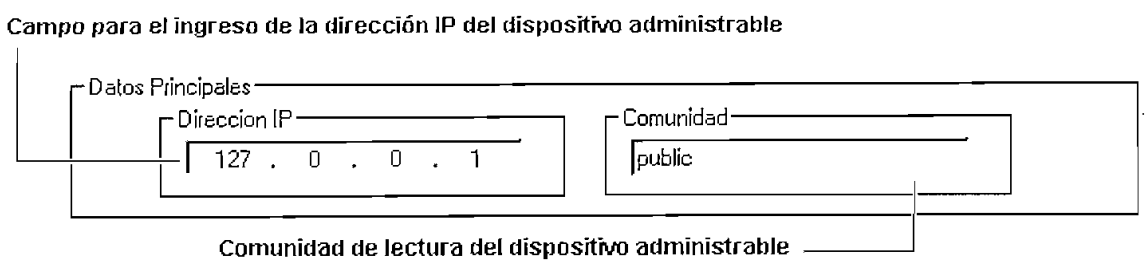


Figura 3. 45 Datos de ingreso para la Herramienta SNMP del programa NetGraphGold.

Árbol de OIDs con la carpeta de oids de sistema seleccionada como ejemplo, el cual contiene los elementos: descripción del sistema operativo en el host, Objeto OID identificador, lapso de tiempo desde su último reinicio, Nombre del contacto de la empresa fabricante, nombre del host, número de servicios levantados; mostrada en la figura 3.46.

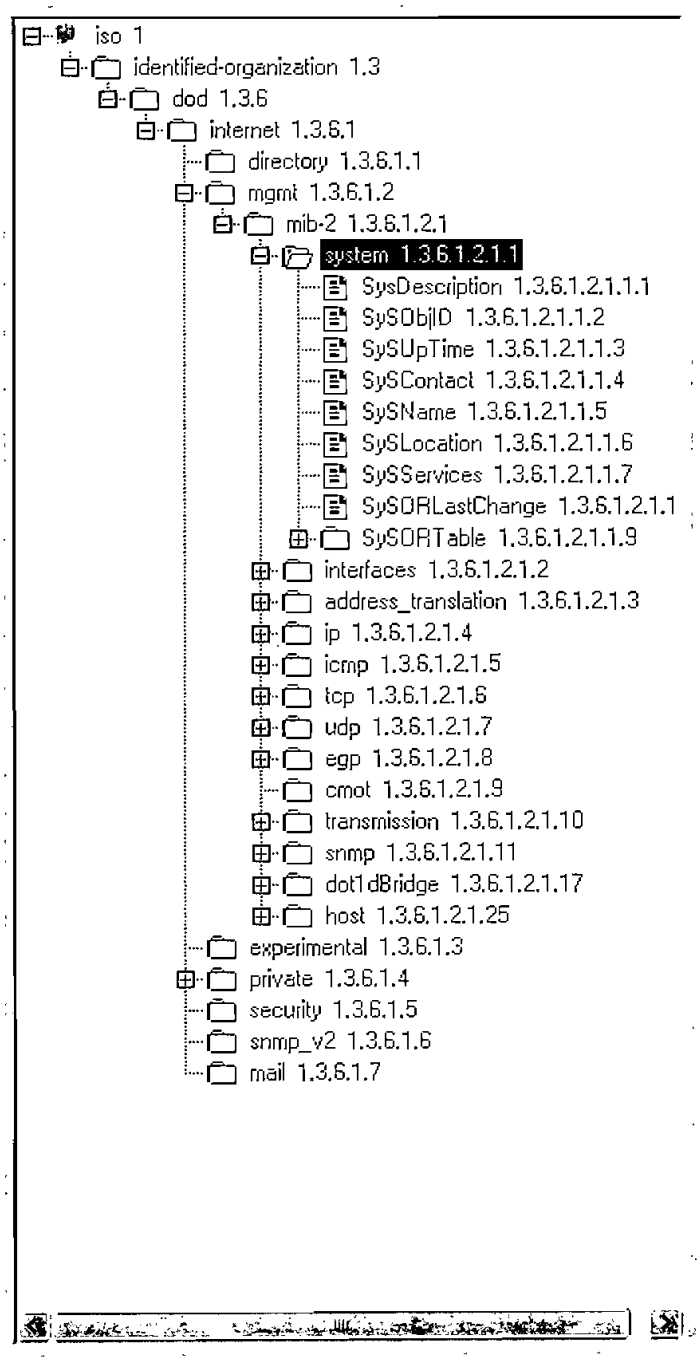


Figura 3. 46 Árbol OID para la Herramienta SNMP del programa NetGraphGold.

El campos de opciones de selección de oids a enviarse, figura 3.47.

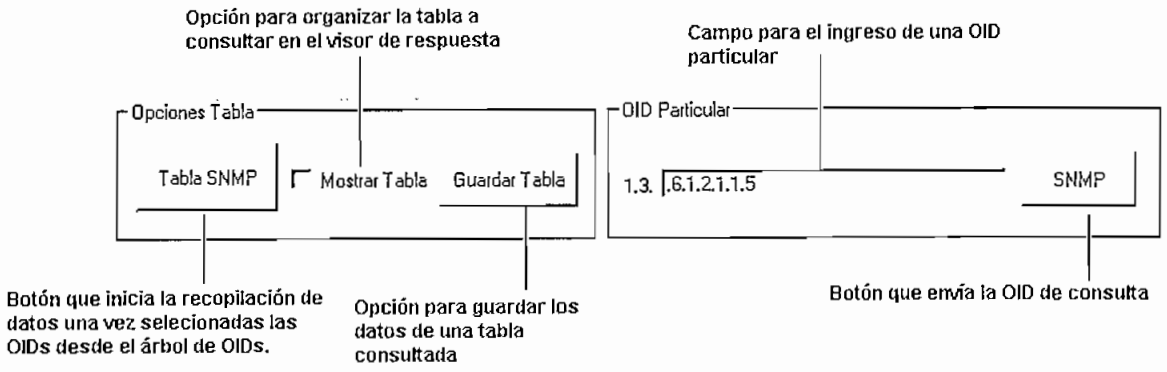


Figura 3. 47 Opciones para la Herramienta SNMP del programa NetGraphGold.

Si la selección fue la de utilizar las opciones de tabla, los campos a utilizarse serán: Campo de previsualización de OIDs seleccionadas desde el árbol de OIDs, campo de OIDs seleccionadas para consulta de tabla; para después presionar el botón de consulta "Tabla SNMP" de la figura 3.47 y observar los resultados obtenidos desde el dispositivo remoto. Lo anteriormente explicado se visualiza en la figura 3.48 con el ejemplo del grupo *system* seleccionado.

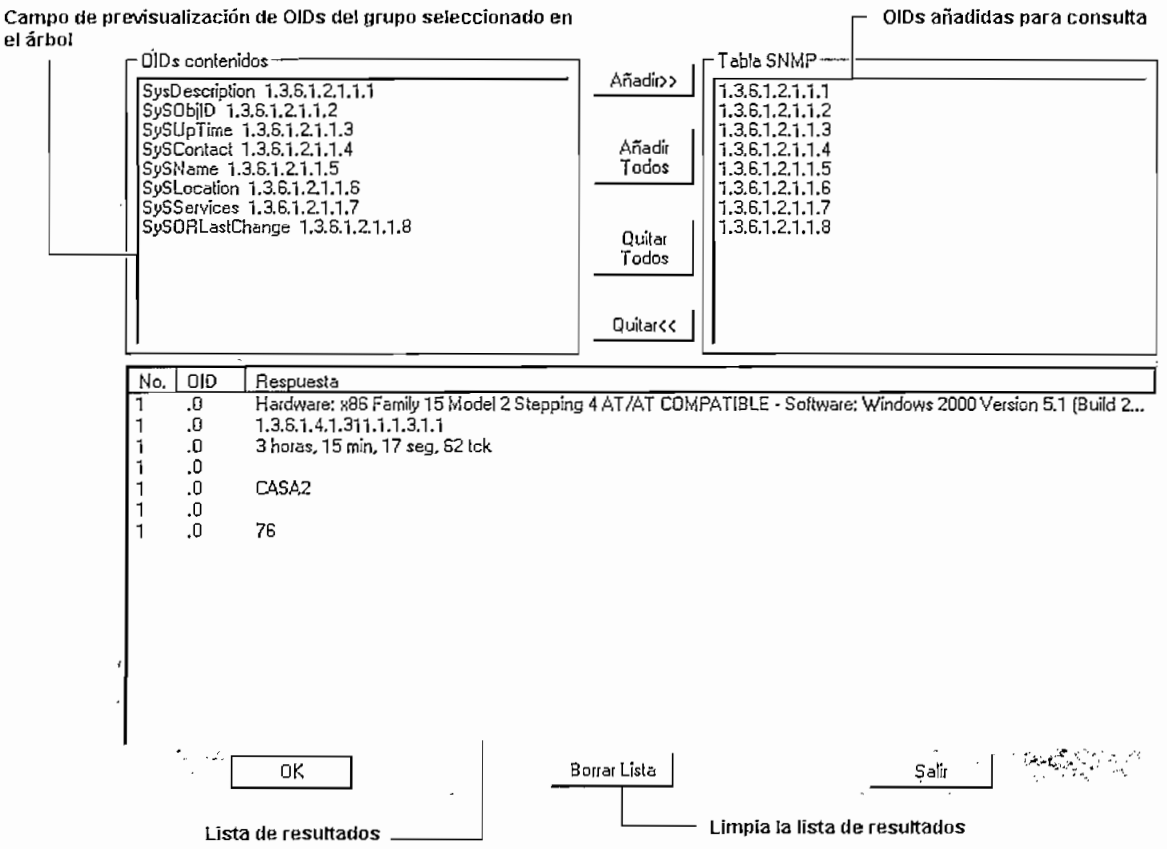


Figura 3. 48 Campos varios de la Herramienta SNMP del programa NetGraphGold.

Medidor de Tráfico

Esta herramienta permite medir el tráfico entrante y saliente hacia una interfaz serial de un ruteador dada su dirección IP, la comunidad de lectura, el ancho de banda y el intervalo de medición.

La figura 3.49 muestra la herramienta descrita.

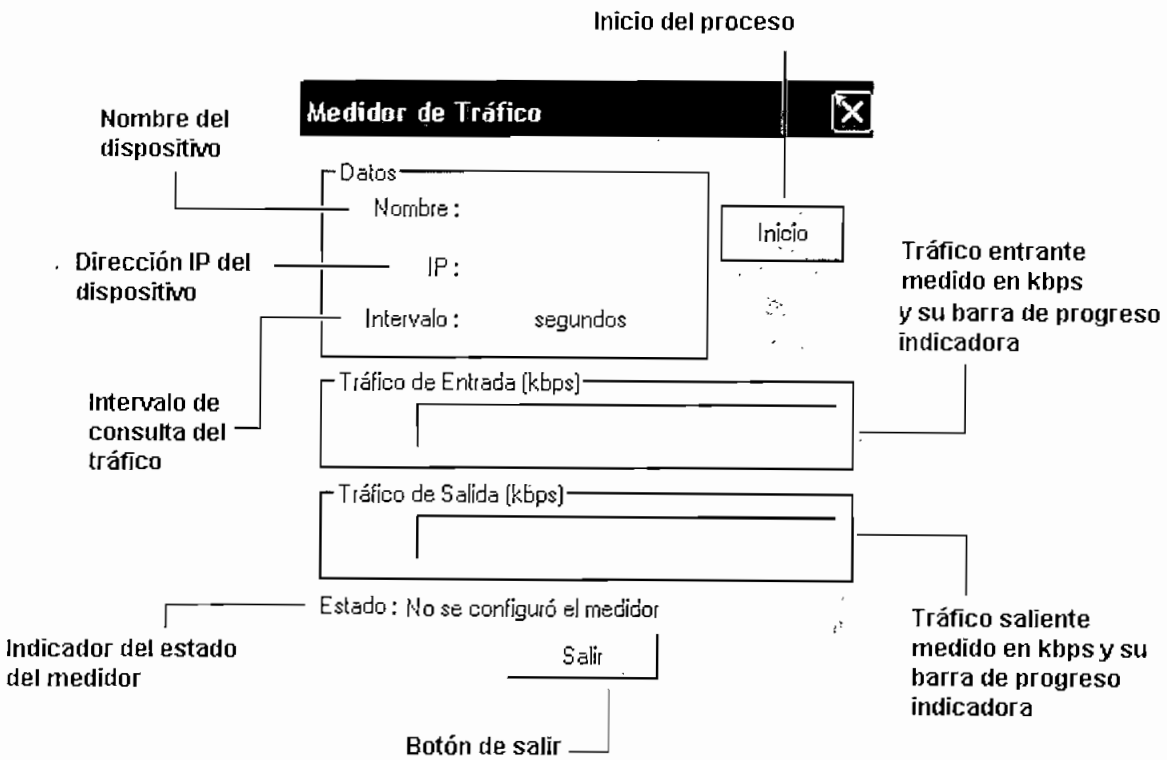


Figura 3. 49 Medidor de Tráfico del programa NetGraphGold.

Una vez pulsado el botón de Inicio de proceso aparece una ventana de diálogo para el ingreso de los datos del ruteador a monitorear, presentado en la figura 3.50.

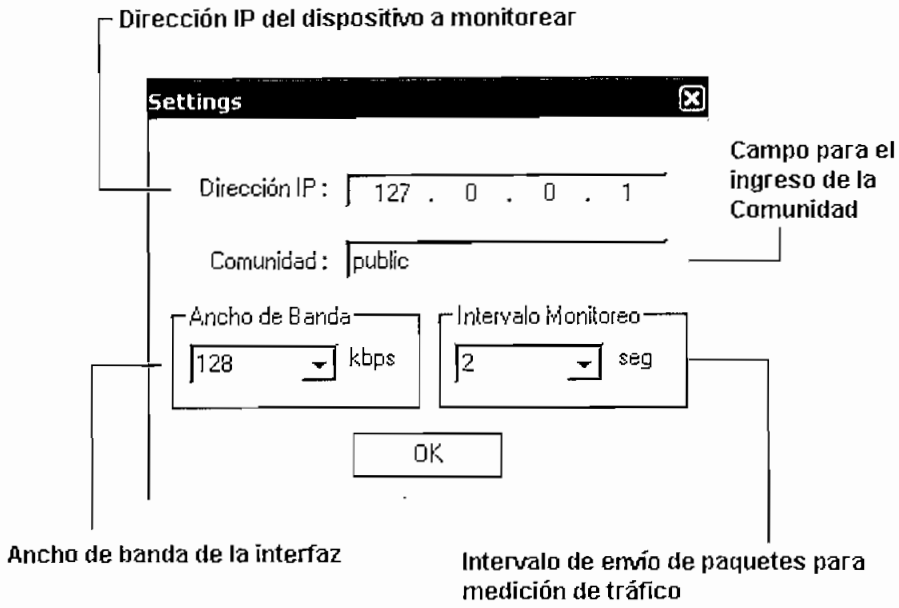


Figura 3. 50 Cuadro de diálogo de ingreso de datos para el Medidor de Tráfico del programa NetGraphGold.

Una vez ingresados los datos, aparece un cuadro de diálogo para escoger la interfaz serial del ruteador de la cual se desea medir el tráfico(figura 3.51).

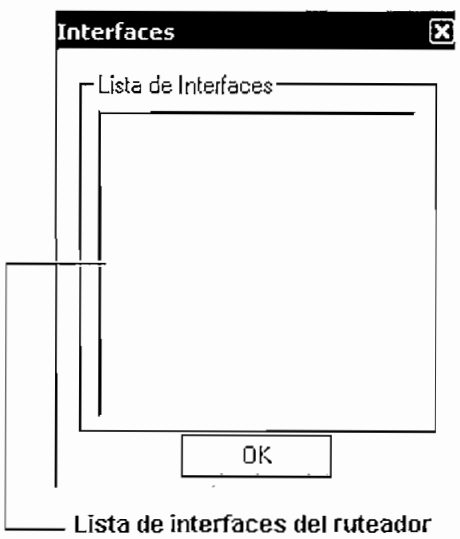


Figura 3. 51 Lista de interfaces obtenida desde el ruteador a medir el tráfico para el Medidor de Tráfico del programa NetGraphGold.

Medidor General

Esta herramienta permite medir el tráfico entrante y saliente de los protocolos TCP, UDP, ICMP, SNMP hacia un ruteador dada su dirección IP, la comunidad de lectura, y el intervalo de medición. Solo se necesita especificar en el menú de medidores de tráfico que tipo de tráfico se desea monitorear, para que esta herramienta mida lo seleccionado.

La figura 3.52 muestra la herramienta descrita.

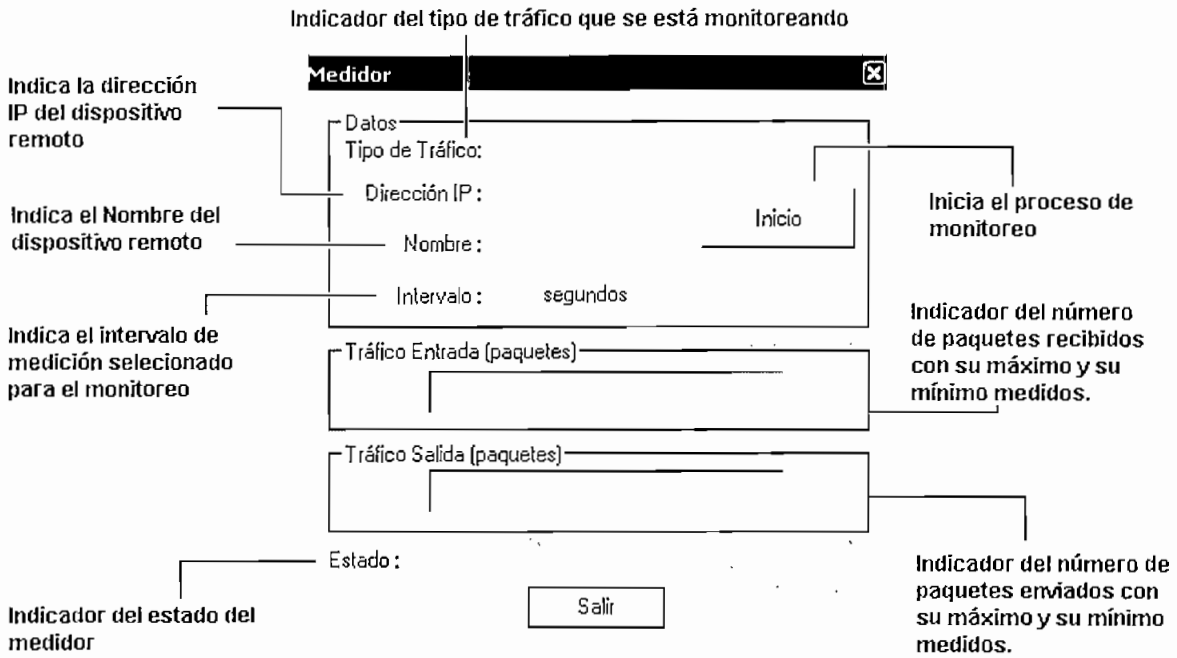


Figura 3. 52 Medidor de Tráfico General del programa NetGraphGold.

Una vez pulsado el botón de Inicio de proceso aparece una ventana de diálogo para el ingreso de los datos del ruteador a monitorear, presentado en la figura 3.53.

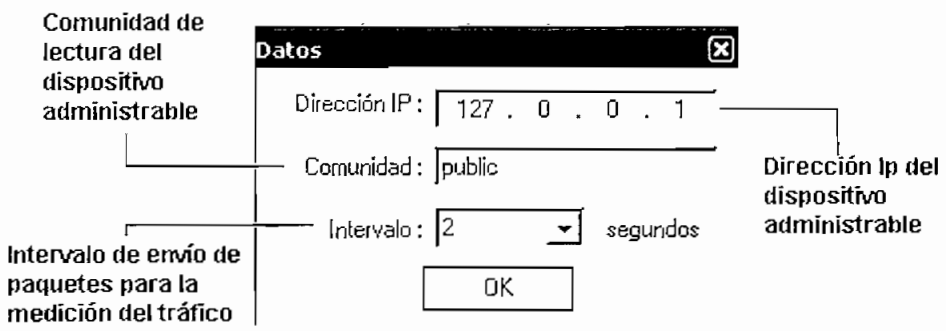


Figura 3. 53 Cuadro de diálogo de ingreso de datos para el Medidor de Tráfico General del programa NetGraphGold.

Localizador de Dispositivos

Esta herramienta permite encontrar un dispositivo dada su dirección IP dentro de una red ya explorada por el programa principal, y lo muestra en el visor de gráficos de red mediante un flecha de localización.

La figura 3.54 muestra la herramienta descrita.

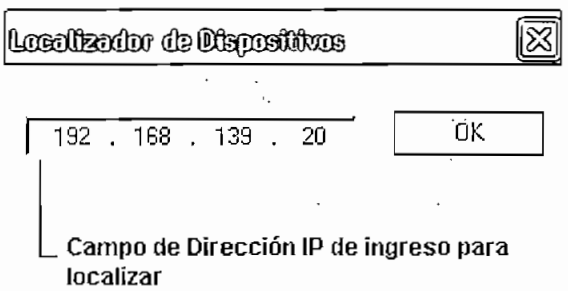


Figura 3. 54 Herramienta Localizador de Dispositivos del programa NetGraphGold.

Un ejemplo de localización se muestra en la figura 3.55, tomando la dirección 192.168.139.20 dentro de la red de la SNT, apareciendo en pantalla la subred donde se localiza el dispositivo y a continuación una flecha señalando al mismo.

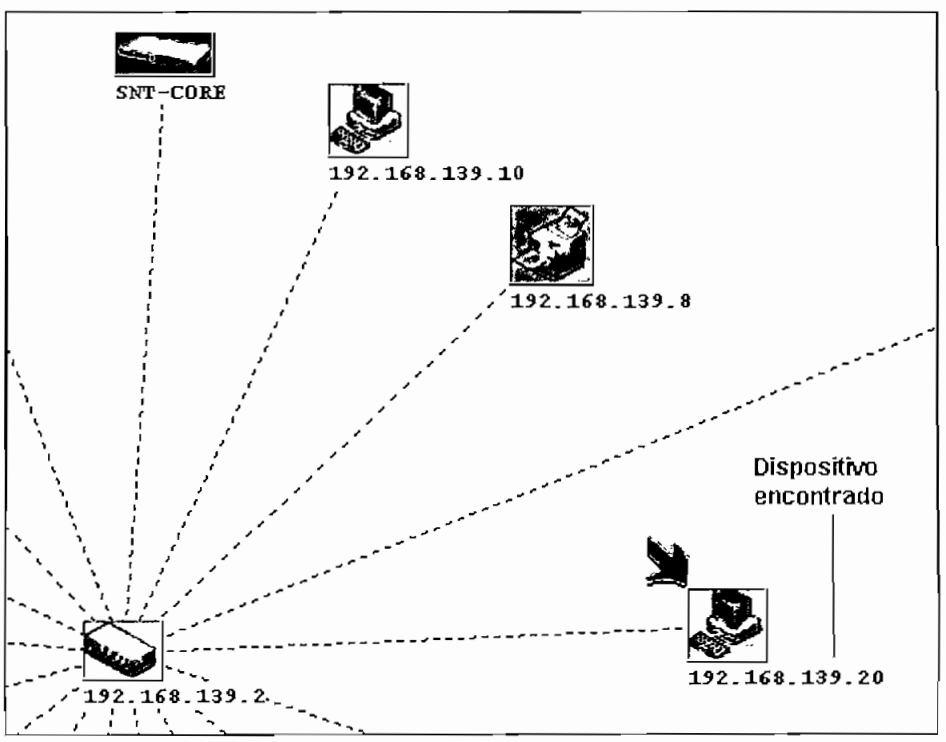


Figura 3. 55 Ejemplo de la Herramienta Localizador de Dispositivos del programa NetGraphGold en la red de la SNT.

Número de Colisiones

Esta herramienta permite medir el número de colisiones en una interfaz de un ruteador dada su dirección IP, la comunidad de lectura, en un intervalo de medición de 5 segundos, previamente establecido.

La figura 3.56 muestra la herramienta descrita.

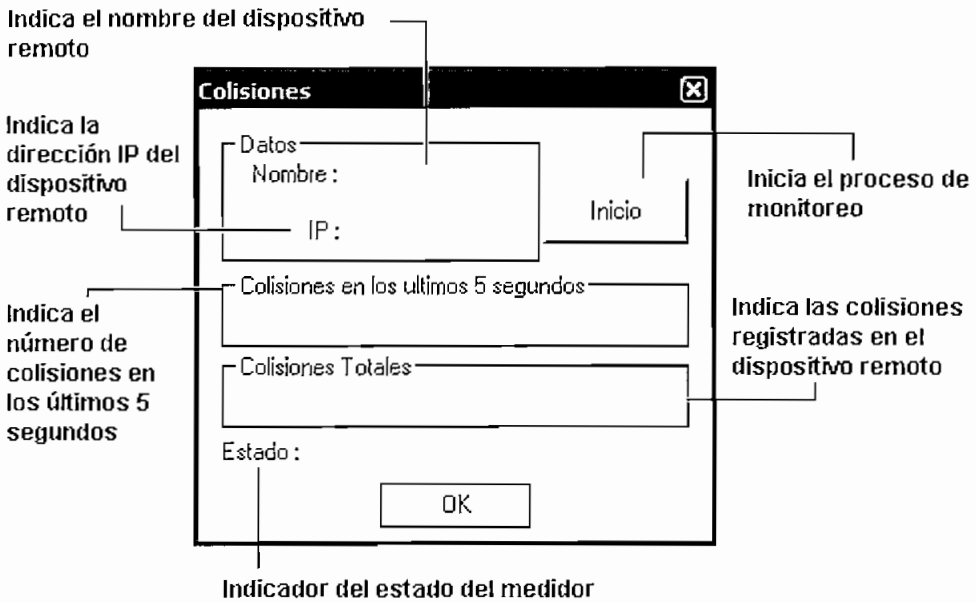


Figura 3. 56 Medidor de Colisiones del programa NetGraphGold.

Una vez pulsado el botón de inicio de proceso, aparece una ventana de diálogo para el ingreso de los datos del ruteador a monitorear, presentado en la figura 3.57. Donde al ingresar los datos, después aparecerá una ventana para seleccionar la interfaz del ruteador; la cual se va a monitorear, presentada en la figura 3.51.

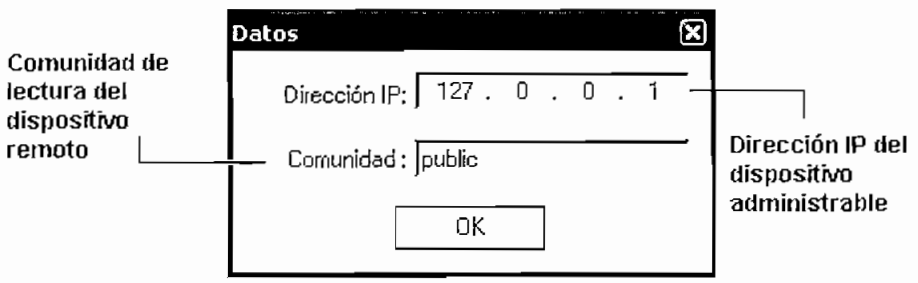


Figura 3. 57 Cuadro de diálogo de ingreso de datos para el Medidor de Colisiones del programa NetGraphGold.

Tramas Descartadas

Esta herramienta permite medir el número de Tramas Descartadas en una interfaz de un ruteador dada su dirección IP, la comunidad de lectura, en un intervalo de medición de 5 segundos, previamente establecido.

La figura 3.58 muestra la herramienta descrita.

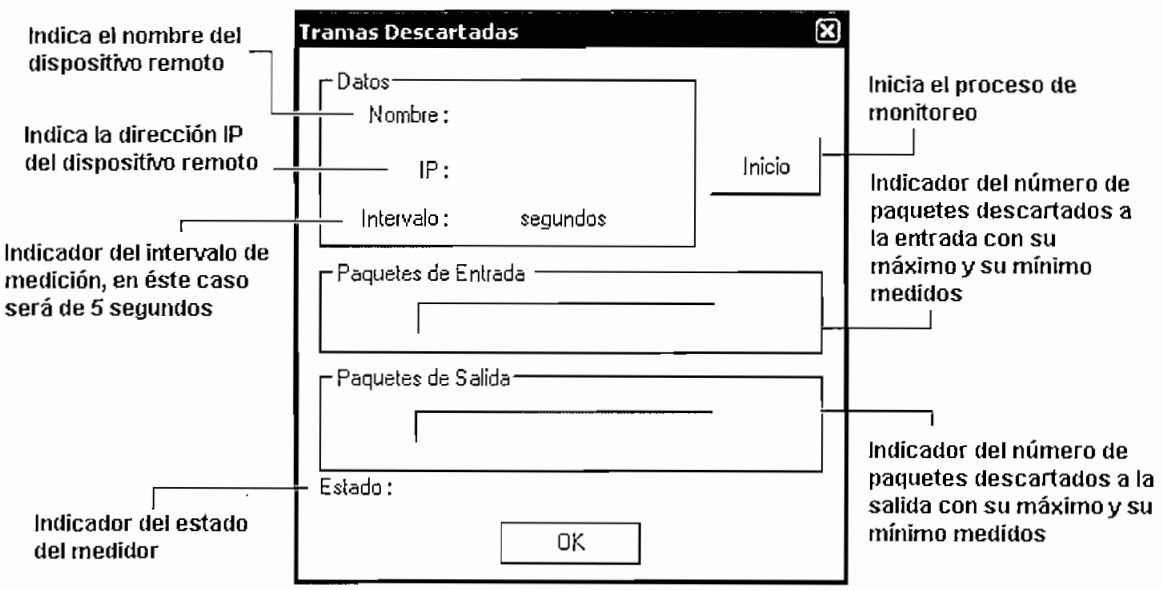


Figura 3. 58 Medidor de Tramas Descartadas del programa NetGraphGold.

Una vez pulsado el botón de inicio de proceso, aparece una ventana de diálogo para el ingreso de los datos del ruteador a monitorear, presentado en la figura 3.57. Donde al ingresar los datos, después aparecerá una ventana para seleccionar la interfaz del ruteador; la cual se va a monitorear, presentada en la figura 3.51, respectivamente.

Uso del CPU

Esta herramienta permite medir el uso del procesador de un ruteador cisco en tanto por ciento durante los últimos 5 segundos, previamente establecido como intervalo fijo de medición.

La figura 3.59 muestra la herramienta descrita.

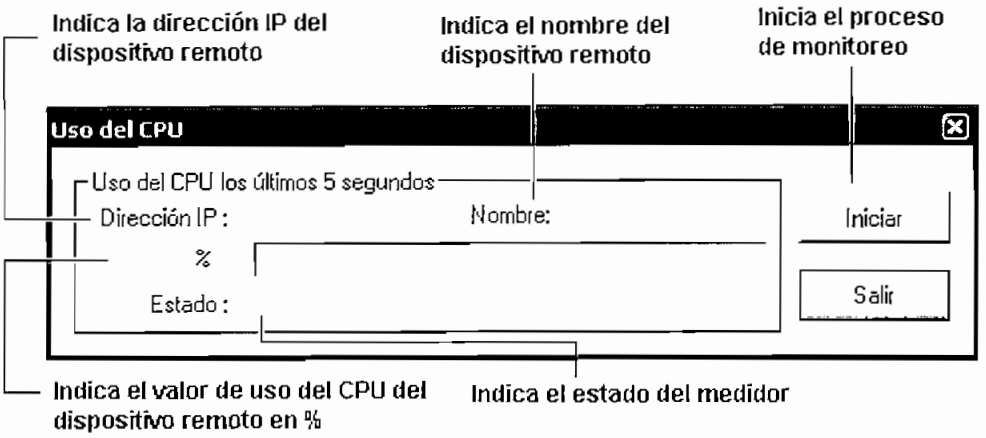


Figura 3. 59 Medidor del Uso del CPU de un ruteador del programa NetGraphGold.

Una vez pulsado el botón de inicio de proceso, aparece una ventana de diálogo para el ingreso de los datos del ruteador a monitorear, presentado en la figura 3.57.

3.5 COMPARACIÓN CON OTRO PROGRAMA.

En el mercado actual que nos ofrece el Internet, se pueden encontrar varios programas comerciales que permiten realizar exploraciones y monitoreo dentro de una red, y que además permiten obtener reportes de ella. Tales programas tienen diferentes costos dependiendo de su complejidad y versatilidad, llegando a bordear los miles y decenas de miles de dólares en algunos casos.

En el caso del programa *Network Inspector* de la casa comercial FLUKE, el cual monitorea una red y presenta los resultados en reportes de dispositivos como: dispositivos en general, switches, hubs, impresoras, dispositivos administrables y otros dispositivos; a manera de listas con sus respectivas características; representa uno de los más sencillos y prácticos que se los puede encontrar en la red Internet; con la particularidad de que se divide en dos módulos funcionales, el primero que es el "Agente", el cual realiza el monitoreo y se mantiene alerta una vez terminada la exploración, y el módulo de "Consola" en el que se presentan los resultados a manera de listas de elementos.

Al realizar una exploración en la red perteneciente al Laboratorio de Informática en el sexto piso del edificio de Eléctrica – Química de la Escuela Politécnica Nacional, en cuyos hosts se encuentra instalado el programa *Network Inspector* se obtuvieron los siguientes resultados organizados a continuación:

La red encontrada es de aproximadamente 100 dispositivos (figura 3.60), ya que el programa almacena en sus registros dispositivos que incluso ya no están presentes o activos en la red; 90 hosts (figura 3.61), 2 switches capa2 (figura 3.62) y 4 dispositivos administrables (figura 3.63), con sus respectivas características. Otra de las singularidades del programa Network Inspector, es que los símbolos al costado izquierdo de cada dispositivo indican la no conexión o error en el registro de cada uno, falta de información y alertas varias.

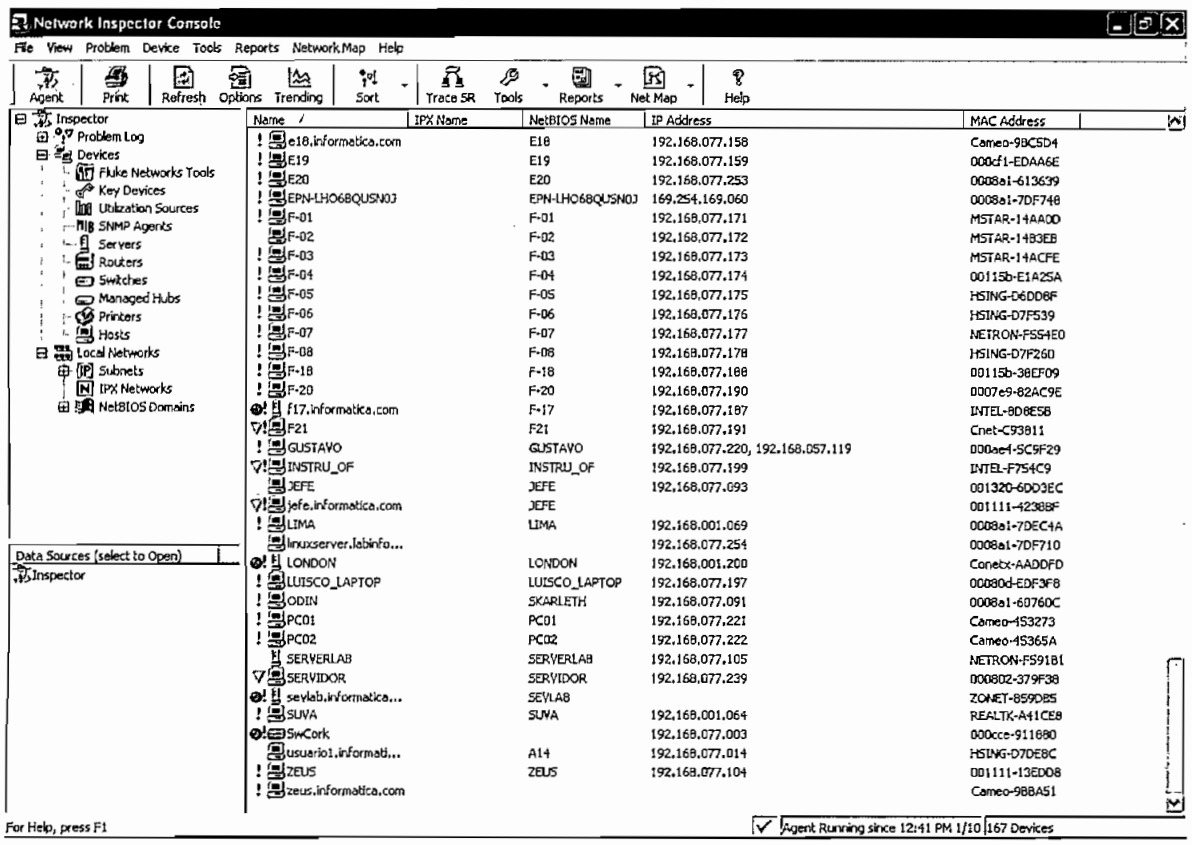


Figura 3. 60 Lista de dispositivos encontrados en la Subred del Laboratorio de Informática del sexto piso del Edificio de Eléctrica – Química de la EPN por el programa Network Inspector.

En la figura 3.61, se muestra la lista de hosts encontrados en la red del Laboratorio la EPN, indicando su nombre DNS, su nombre de NetBIOS, su dirección IP y su dirección MAC, ordenados por orden alfabético.

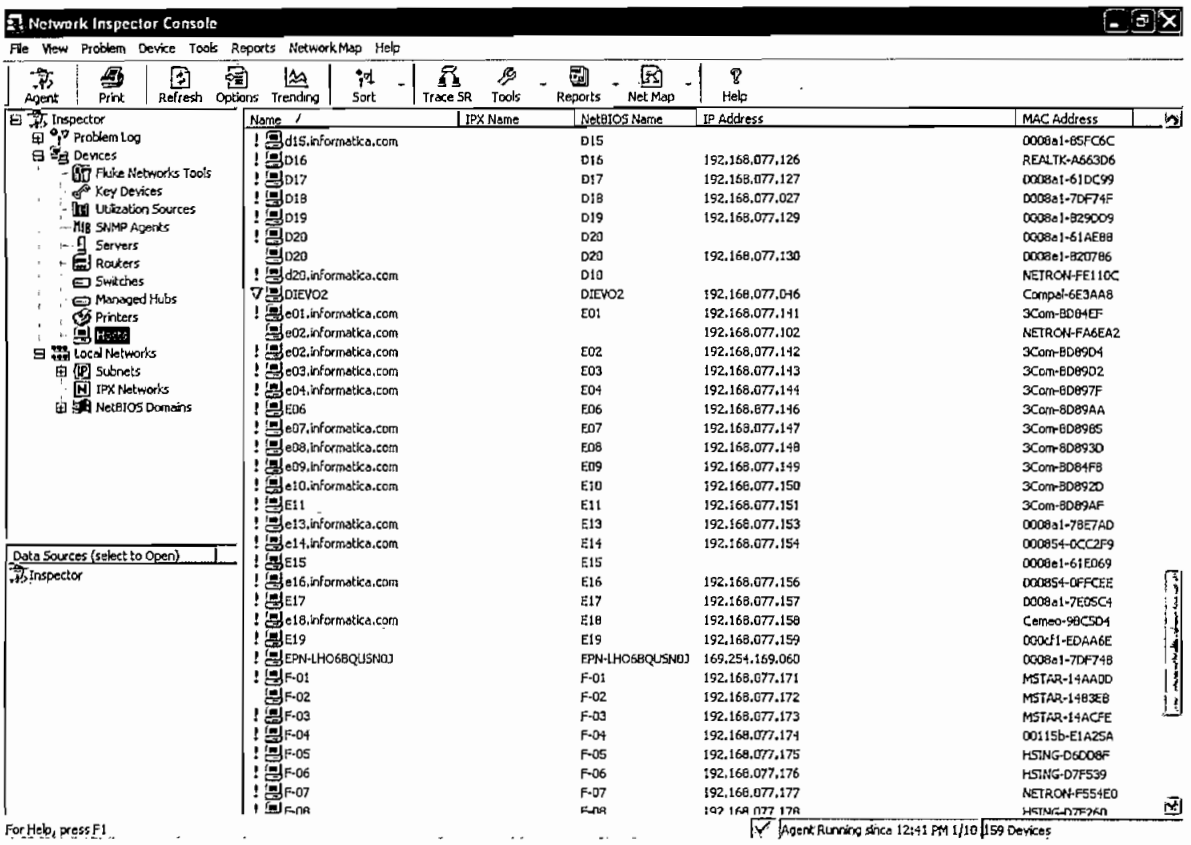


Figura 3. 61 Lista de hosts encontrados en la Subred del Laboratorio de Informática del sexto piso del Edificio de Eléctrica – Química de la EPN por el programa Network Inspector.

En la figura 3.62 se encuentra la lista de dispositivos administrables encontrados en la red del Laboratorio de la EPN, indicando su nombre, su dirección IP y su dirección MAC, ordenados por orden alfabético, con la diferencia de que los símbolos rojos a la izquierda del dispositivo indican la desconexión del mismo.

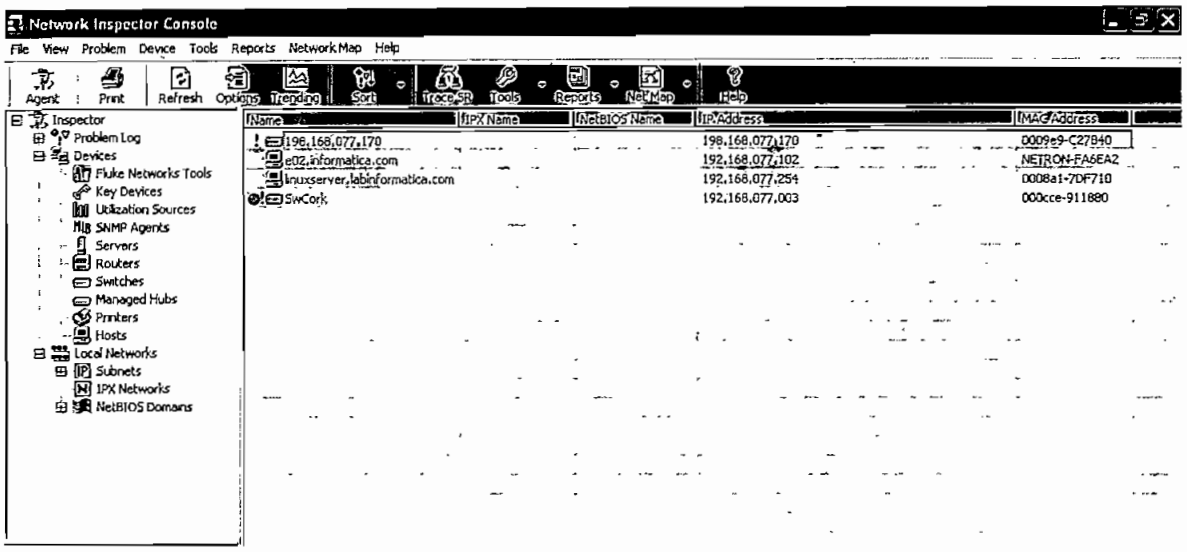


Figura 3. 62 Lista de dispositivos administrables encontrados en la Subred del Laboratorio de Informática del sexto piso del Edificio de Eléctrica – Química de la EPN por el programa Network Inspector.

En la figura 3.63 se encuentra la lista de switches capa2 encontrados en la red del Laboratorio de la EPN, indicando su nombre, su dirección IP y su dirección MAC; con la diferencia de que los símbolos de color azul indican falta de información y el rojo la desconexión o no presencia del equipo desde la última exploración.

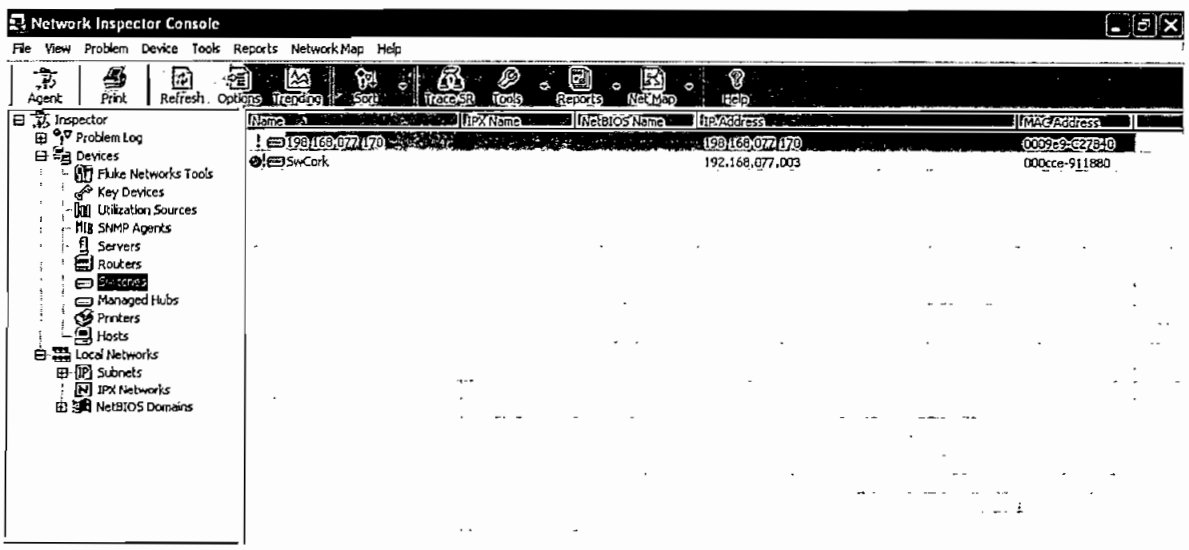


Figura 3. 63 Lista de switches capa2 encontrados en la Subred del Laboratorio de Informática del sexto piso del Edificio de Eléctrica – Química de la EPN por el programa Network Inspector.

Realizando una comparación de los resultados obtenidos en la red perteneciente al Laboratorio de Informática de la EPN, explorada tanto con en el programa desarrollado *NetGraphGold* como con en el programa *Network Inspector*, podemos concluir que aunque ambos registran la presencia de dispositivos y los clasifican según su tipo; utilizan diferentes formas de almacenamiento de registro; puesto que para el primero, el registro se crea en el instante mismo de la exploración y se lo visualiza con el gráfico de la topología; mientras que para el segundo, el registro nuevo se mezcla con el almacenado con anterioridad, indicando en las listas propias del programa, dispositivos que no están activos o encendidos en la red en conjunto con los activos.

CAPÍTULO 4: CONCLUSIONES Y RECOMENDACIONES.

4.1 CONCLUSIONES GENERALES.

1. En la actualidad hay muchas instituciones que mantienen una estructura de comunicaciones; muchas veces complejas dentro de sus empresas, debido a la necesidad de mantener la comunicación con sus diferentes filiales y de continuar a la vanguardia en la tecnología; por lo que precisan de la administración de la configuración de su red, y la implementación de políticas de gestión, que permitan un mejor manejo y control de los diferentes componentes que la conforman.

2. La administración de una red conlleva al dominio de sus componentes por parte del administrador o gestor, lo cual le permite ver en un principio el desempeño de la red, monitorear su funcionamiento y medir ciertos parámetros que le permitan conocer de mejor manera, los trabajos y tareas que realizan cada uno de sus diferentes dispositivos, y así determinar en segunda instancia, gracias a los métodos de control, si se necesitan corregir o prevenir posibles errores.

3. El conocimiento de la distribución y composición de una red mediante el uso de un gráfico, ayuda al administrador en el control y manejo de una red, permitiendo que herramientas especializadas en monitoreo de dispositivos, y el uso de protocolos especializados como SNMP para la gestión de una red e ICMP para conectividad, contribuyan al mejoramiento del dominio y regulación del uso de los recursos que se dispone en la red.

4. El uso de herramientas administradoras para la dirección de una red, es independiente de la complejidad de la distribución de los dispositivos dentro de la misma, por lo que es factible la implementación de políticas

de gestión que optimicen los recursos utilizables, incluso en redes de corta extensión.

4.2 CONCLUSIONES ESPECÍFICAS

1. El uso de sockets mediante procedimientos asincrónicos, es decir, ejecución de rutinas de programación, al momento de producirse un mensaje de ventana determinado previamente configurado en la aplicación; facilita el diseño e implementación de un programa de monitoreo y exploración, pues los sockets realizan sus operaciones de forma paralela a la ejecución del programa general, y de las demás funciones del host de prueba.

2. El uso de protocolos de administración como SNMP en sus distintas versiones, para recolección de datos significativos de los dispositivos administrables, comprende un correcto conocimiento y uso de sus operaciones, para el envío y recepción de paquetes, extrayendo información de gestión almacenado en las MIBs, independientemente de los objetos e instancias que soporte el dispositivo administrado, debido a que el protocolo es autónomo a la base de información.

3. El uso del protocolo ICMP para establecer conectividad de un dispositivo dentro de una red, permite determinar la presencia del mismo y la factibilidad de comunicación de otros dispositivos hacia él.

4. El conocimiento de la información de gestión que se dispone en los diferentes dispositivos administrables, tales como: switches capa2, ruteadores, switches capa3, etc., permite realizar preguntas puntuales mediante el uso del protocolo SNMP, que faciliten la diferenciación de los mismos dentro de una red constituida, y una vez clasificados, favorezca la recolección de información precisa para la diagramación de la topología de la red.

5. La presencia de dispositivos administrables de un nivel mayor de composición como un switch capa3 dentro de una red, centraliza la distribución de los dispositivos y organiza de mejor manera la disposición de los mismos debido al nivel de configuración que tal dispositivo necesita; es decir, mientras más inteligente sea un equipo de ruteo, almacenará mayor información útil para la esquematización de la topología de una red.

6. La implementación de un programa basado en clases para realizar la exploración de una red, permite separar en módulos mediante las clases diseñadas y propiamente dichas, cada uno de los diferentes procedimientos que tanto la exploración como la graficación así lo requieran.

7. La organización de la información obtenida en la exploración de una red, se la sistematiza y procesa por subred, de tal manera que se encuentran almacenados mediante tablas, los datos de cada dispositivo dentro de la aplicación, así que siempre se encuentran disponibles los datos pertenecientes a un dispositivo, de tal manera que se optimiza el funcionamiento del programa.

8. Gracias a la utilización del lenguaje de programación C++ y a las herramientas de interfaz gráfica de Visual C++, se pueden construir mapas de bits de memoria que manejen de manera adecuada un gráfico de topología de red, además permitiendo una interacción del usuario con el gráfico para obtener información u opciones del mismo.

9. En caso de no existir dispositivos administrables dentro de una red conformada, o de no estar configurados para ser administrados mediante el protocolo de gestión correspondiente, el resultado gráfico sería el más aproximado, básico y propicio para el caso, como es el de un esquema centralizado de dispositivos alrededor de una nube de comunicación.

10. Las herramientas adicionales construidas para apoyo de monitoreo de dispositivos, como por ejemplo, los medidores de tráfico para los ruteadores; permiten entender de mejor manera el comportamiento de dichos dispositivos a las condiciones de trabajo que experimentan dentro de la red.

11. Las operaciones y la nueva base de información de gestión (MIB II) que ofrece la versión mejorada del protocolo de administración SNMPv2, contribuye a un mejor funcionamiento de las herramientas de monitoreo y presentan un panorama más amplio para el desarrollo de nuevos instrumentos, en el uso del protocolo de administración de red.

12. El gráfico obtenido durante cualquier exploración, es un aproximado a la topología real de la red, pero que contiene una lógica y distribución acorde con los datos recogidos durante la exploración; ajustándose a los objetivos planteados para el desarrollo de la aplicación.

4.3 RECOMENDACIONES.

- El éxito de los requerimientos administrativos sobre un dispositivo a gestionar dentro de una red, depende del conocimiento que se tenga sobre las MIBs y por su puesto sobre el protocolo de administración propiamente dicho, porque en ausencia de éstos elementos las solicitudes administrativas serían erróneas, y por tanto, la información de gestión necesaria para realizar un gráfico aproximado de la topología de red en exploración, no se podría recuperar.

- Para optimizar el funcionamiento del programa, se aconseja utilizar un computador de al menos 1.6GHz en el procesador y 256 MB de memoria RAM; puesto que el programa realiza los gráficos en memoria, lo cual resta la capacidad de éste último recurso en particular.

BIBLIOGRAFÍA

ALCÁZAR ALARCOS Bernardo ; "*Gestión Internet*"; bernardo@aut.uah.es http://it.aut.uah.es/alarcos, 2000.

AMERICAN NATIONAL STANDARS INSTITUTE, "*American National Standard Code for Information Interchange*", USA 1997.

BARQUERA RAMÍREZ Jessica; "*SNMP*"; Centro Universitario ETAC; Ingeniería en Redes Computacionales; Saulo.com.

CASE Jeffrey, "*SNMP Network Management*", McGraw-Hill, 1996.

CEBALLOS, Francisco Javier: *"Visual C++",* Alfaomega Grupo Editor, México, 1999.

COMER Douglas, "*Interconectividad de redes con TCP/IP, Diseño e Implementación",* Pearson Educación, México, 2000.

DIEZ Eduardo, "*Monitoreo*", Universidad de los Andes, 2004.

DIEZ y RIEGA, Eduardo, *"Monitoreo - Network Management SNMP (Simple Network Management Protocol)",* Universidad de los Andes, Merida –Venezuela, 2000.

DIEZ Y RIEGA Eduardo ; *"MONITOREO SNMP"*; Universidad de los Andes; Merida Venezuela, deduar@ula.ve, 2001.

DIT-UPM, "*Gestión de Red",2000.*

EGAS Carlos; "*Gestión de Redes*"; CLEI; Ecuador 1998.

FEIT Sidnie, "*TCP/IP, Arquitectura, Protocolos e implementación con IPV6 y Seguridad IP*", McGrawHill, España, 1998.

FUENTES SAMANIEGO, Fausto Raúl *"Generador y colector de paquetes ICMP"*,EPN , Marzo 2001.

GESTIÓN DE REDES; *"ASN.1 Abstract Syntax Notation One"*; 2004.

HERRERA Carlos; *"La Técnica de Objetos Aplicada a la Gestión de redes de Telecomunicaciones. "*,Tesis de Grado, EPN, Ecuador , 1999.

INCHAUSPE Inés E., *"Monitoreo de Redes"*, Universidad Nacional de Luján, 2001.

INTERNATIONAL ORGANIZATION FOR STANDARIZATION; *"Information Technology: Open System Interconnection, Specification of Abstract Syntax Notation One"*; ISO 8824; USA , 1990.

KRUGLINSKI, David; *"PROGRAMACIÓN AVANZADA CON VISUAL C++6.0"*; MacGrawHill, Madrid, España,1999

LESCHENNE Sebastián;*"Modelo de Gestión Internet SNMP";* Redes de Datos 2002; Universidad Nacional de Rosario, 2002.

McCLOGHRIE K, Rose, *"Management Information Base for Network Management of TCP/IP"*, *RFC 1156*, 1990.

McCLOGHRIE K, Rose, *"Structure and Identification of Management Information for TCP/IP"*, RFC 1155, 1991.

MILLER Mark, *"Managing Internetworks with SNMP"*, M&T Books, USA 1993.

MUKH Vijay, *"BER SNMP"*, Vijay Mukhi's Computer Institute, India, 2000.

MSDN Ayuda en Línea. www.msdn.com

MSDN Library Visual Studio 6.0a, Copyright 1995 – 2000 Microsoft Corp.

RFC 1071, 2003.

RFC 1155, *"Structure and Identification of Management Information for TCP/IP – based Internets".*

RFC 1156, "*Management Information Base for Network Management of TCP/IP - based Internets*".

RFC 1157, "*A Simple Network Management Protocol(SNMP)*".

RFC 1212; "*MIB DEFINITIONS*".

RFC 1213-MIB; "*Management Information Base for Network Management of TCP/IP - based Internets: MIB II*"; 2003.

RFC 1441; "*Introduction to SNMPv2*".

RFC 1443; "*Textual Conventions for SNMPv2*".

RFC 1445; "*Administrative Model for SNMPv2*".

RFC 1446; "*Security Protocols for SNMPv2*".

RFC 1448; "*Protocol Operations for SNMPv2*".

RFC 1450; "*MIB for SNMPv2*".

RFC 1452; "Coexistence between SNMPv1 and SNMPv2".

RFC 1573, 2003.

ROSE Marshall; "*The Simple Book*"; Prentice Hall; USA 1994.

TANENBAUM, Andrew S.:" *Redes de computadoras"*, tercera edición. Pearson, 1997.

TANENBAUM Andrew; "*Redes de Computadoras*"; Cuarta Edición; PrenticeHall; México 2003.

VICENTE Carlos, "*Gestión de Redes*", Universidad de Oregon, 2004.

VIJAY MUKHI'S COMPUTER INSTITUTE; *"BER SNMP"*
vmukhi@giasbm01.vsnl.net.in, 2000.

ANEXOS

ANEXO 1

A continuación se incluyen algunos términos manejados a lo lago del desarrollo del presente trabajo:

MIB View: Se define como una colección, o familia, de subárboles, donde cada subárbol puede estar incluido o excluido de la MIB view. Se utiliza para restringir el acceso a determinados objetos a un grupo en particular.

Mapa de bits: Gráficos por ordenador o computadora almacenados y mantenidos como colecciones de bits que describen las características de los píxeles individuales en la pantalla, así como los datos generales del gráfico (como el tamaño, la paleta de colores utilizada o la resolución).

Píxel: Superficie homogénea más pequeña de las que componen una imagen, que se define por su brillo y color. Un píxel es el elemento más pequeño que el hardware y el software de presentación o impresión puede tratar para crear letras, números o gráficos.

ANEXO 2

A continuación se anexan algunas de las funciones extraídas de MSDN Library Visual Studio 6.0a, Copyright 1995 – 2000 Microsoft Corp.; con las cuales se se ha desarrollado la exploración y uso de sockets en el programa NetGraphGold; y funciones adicionales con las que se puede extraer del "*Editor de Registro de Windows*", cierta información como: el nombre del dominio al cual pertenece el Host, nombres de los servidores de LogOn para dominio, y cualquier otra información que se encuentre localizada en dicho registro.

WSAAsyncGetHostByAddr

The Windows Sockets **WSAAsyncGetHostByAddr** function asynchronously retrieves host information that corresponds to an address.

```
HANDLE WSAAsyncGetHostByAddr (
  HWND hWnd,
  unsigned int wMsg,
  const char FAR * addr,
  int len,
  int type,
  char FAR * buf,
  int buflen
);
```

Parameters

hWnd

[in] The handle of the window that will receive a message when the asynchronous request completes.

wMsg

[in] The message to be received when the asynchronous request completes.

addr

[in] A pointer to the network address for the host. Host addresses are stored in network byte order.

len

[in] The length of the address.

type

[in] The type of the address.

buf

[out] A pointer to the data area to receive the **HOSTENT** data. The data area *must* be larger than the size of a HOSTENT structure because the supplied data area is used by Windows Sockets to contain a HOSTENT structure and all of the data referenced by members of the HOSTENT structure. A buffer of MAXGETHOSTSTRUCT bytes is recommended.

buflen

[in] The size of data area for the *buf* parameter.

Return Values

The return value specifies whether or not the asynchronous operation was successfully initiated. It does not imply success or failure of the operation itself.

WSAAsyncGetHostByName

The Windows Sockets WSAAsyncGetHostByName function asynchronously retrieves host information corresponding to a host name.

```
HANDLE WSAAsyncGetHostByName (
 HWND hWnd,
 unsigned int wMsg,
 const char FAR * name,
 char FAR * buf,
 int buflen
);
```

Parameters

hWnd

[in] The handle of the window that will receive a message when the asynchronous request completes.

wMsg

[in] The message to be received when the asynchronous request completes.

name

[in] A pointer to the null-terminated name of the host.

buf

[out] A pointer to the data area to receive the **HOSTENT** data. The data area must be larger than the size of a HOSTENT structure because the supplied data area is used by Windows Sockets to contain a HOSTENT structure and all of the data referenced by members of the HOSTENT structure. A buffer of MAXGETHOSTSTRUCT bytes is recommended.

buflen

[in] The size of data area for the *buf* parameter.

Return Values

The return value specifies whether or not the asynchronous operation was successfully initiated. It does not imply success or failure of the operation itself.

WSAAsyncSelect

The Windows Sockets WSAAsyncSelect function requests Windows message-based notification of network events for a socket.

```
int WSAAsyncSelect (
 SOCKET s,
 HWND hWnd,
 unsigned int wMsg,
 long lEvent
);
```

Parameters

s

[in] A descriptor identifying the socket for which event notification is required.

hWnd

[in] A handle identifying the window that will receive a message when a network event occurs.

wMsg

[in] The message to be received when a network event occurs.

lEvent

[in] A bitmask that specifies a combination of network events in which the application is interested.

Here is a summary of events and conditions for each asynchronous notification message:

FD_READ:

when WSAAsyncSelect called, if there is data currently available to receive,
when data arrives, if FD_READ not already posted,
after recv or recvfrom called (with or without MSG_PEEK), if data is still available to receive.
Note when setsockopt SO_OOBINLINE is enabled, "data" includes both normal data and out-of-band (OOB) data in the instances noted above.

FD_WRITE:

when WSAAsyncSelect called, if a send or sendto is possible
after connect or accept called, when connection established
after send or sendto fail with WSAEWOULDBLOCK, when send or sendto are likely to succeed,
after bind on a connectionless socket. FD_WRITE may or may not occur at this time (implementation-dependent). In any case, a connectionless socket is always writeable immediately after a bind operation.

FD_ACCEPT:

when WSAAsyncSelect called, if there is currently a connection request available to accept,
when a connection request arrives, if FD_ACCEPT not already posted,
after accept called, if there is another connection request available to accept.

FD_CONNECT:

when WSAAsyncSelect called, if there is currently a connection established,
after connect called, when connection is established (even when connect succeeds immediately, as is typical with a datagram socket),
after calling **WSAJoinLeaf**, when join operation completes,
after **connect**, **WSAConnect**, or WSAJoinLeaf was called with a nonblocking, connection-oriented socket. The initial operation returned with a specific error of WSAEWOULDBLOCK, but the network operation went ahead. Whether the operation eventually succeeds or not, when the outcome has been determined, FD_CONNECT happens. The client should check the error code to determine whether the outcome was successful or failed.

FD_CLOSE: Only valid on connection-oriented sockets (for example, SOCK_STREAM)
when WSAAsyncSelect called, if socket connection has been closed, after remote system initiated graceful close, when no data currently available to receive (note: if data has been received and is waiting to be read when the remote system initiates a graceful close, the FD_CLOSE is not delivered until all pending data has been read), after local system initiates graceful close with shutdown and remote system has responded with "End of Data" notification (for example, TCP FIN), when no data currently available to receive, when remote system terminates connection (for example, sent TCP RST), and *lParam* will contain WSAECONNRESET error value.
Note FD_CLOSE is *not* posted after closesocket is called.

CAsyncSocket

A CAsyncSocket object represents a Windows Socket, which is an endpoint of network communication. The CAsyncSocket class encapsulates the Windows Sockets API, providing an object-oriented abstraction for programmers who want to use Windows Sockets in conjunction with MFC.

Windows CE does not support asynchronous event notification so, when writing an application for Windows CE, you should use the new CCeSocket class, instead of

CAsyncSocket. The CCeSocket class enables Windows CE applications to send and receive socket notifications for certain asynchronous events.

At a Glance

Header file:	Afxsock.h
Versions:	2.0 and later

socket

The Windows Sockets socket function creates a socket that is bound to a specific service provider.

```
SOCKET socket (
  int af,
  int type,
  int protocol
);
```

Parameters

af

[in] An address family specification.

type

[in] A type specification for the new socket.

The following are the only two *type* specifications supported for Windows Sockets 1.1:

Type	Explanation
SOCK_STREAM	Provides sequenced, reliable, two-way, connection-based byte streams with an out-of-band data transmission mechanism. Uses TCP for the Internet address family.
SOCK_DGRAM	Supports datagrams, which are connectionless, unreliable buffers of a fixed (typically small) maximum length. Uses UDP for the Internet address family.

In Windows Sockets 2, many new socket types will be introduced and don' need to be specified now because an application can dynamically discover the attributes of each available transport protocol through the **WSAEnumProtocols** function. Socket type definitions will appear in WINSOCK2.H, which will be periodically updated as new socket types, address families and protocols are defined.

protocol

[in] A particular protocol to be used with the socket that is specific to the indicated address family.

Return Values

If no error occurs, socket returns a descriptor referencing the new socket. Otherwise, a value of INVALID_SOCKET is returned, and a specific error code can be retrieved by calling **WSAGetLastError**.

CAsyncSocket::Create

BOOL Create(UINT *nSocketPort* = 0, int *nSocketType* = SOCK_STREAM, long *lEvent* = FD_READ | FD_WRITE | FD_OOB | FD_ACCEPT | FD_CONNECT | FD_CLOSE, LPCTSTR *lpszSocketAddress* = NULL);

Return Value

Nonzero if the function is successful; otherwise 0, and a specific error code can be retrieved by calling GetLastError. The following errors apply to this member function:

WSANOTINITIALISED A successful AfxSocketInit must occur before using this API.

WSAENETDOWN The Windows Sockets implementation detected that the network subsystem failed.

WSAEAFNOSUPPORT The specified address family is not supported.

WSAEINPROGRESS A blocking Windows Sockets operation is in progress.

WSAEMFILE No more file descriptors are available.

WSAENOBUFS No buffer space is available. The socket cannot be created.
WSAEPROTONOSUPPORT The specified port is not supported.
WSAEPROTOTYPE The specified port is the wrong type for this socket.
WSAESOCKTNOSUPPORT The specified socket type is not supported in this address family.

Parameters

nSocketPort
A well-known port to be used with the socket, or 0 if you want Windows Sockets to select a port.
nSocketType
SOCK_STREAM or SOCK_DGRAM.
lEvent
A bitmask which specifies a combination of network events in which the application is interested.
FD_READ Want to receive notification of readiness for reading.
FD_WRITE Want to receive notification of readiness for writing.
FD_OOB Want to receive notification of the arrival of out-of-band data.
FD_ACCEPT Want to receive notification of incoming connections.
FD_CONNECT Want to receive notification of completed connection.
FD_CLOSE Want to receive notification of socket closure.
lpszSockAddress
A pointer to a string containing the network address of the connected socket, a dotted number such as "128.56.22.8".

CAsyncSocket::Connect

BOOL Connect(LPCTSTR *lpszHostAddress*, UINT *nHostPort*);
BOOL Connect(const SOCKADDR* *lpSockAddr*, int *nSockAddrLen*);

Return Value

Nonzero if the function is successful; otherwise 0, and a specific error code can be retrieved by calling GetLastError. If this indicates an error code of

WSAEWOULDBLOCK, and your application is using the overridable callbacks, your application will receive an OnConnect message when the connect operation is complete.

Parameters

lpszHostAddress

The network address of the socket to which this object is connected: a machine name such as "ftp.microsoft.com", or a dotted number such as "128.56.22.8".

nHostPort

The port identifying the socket application.

lpSockAddr

A pointer to a SOCKADDR structure that contains the address of the connected socket.

nSockAddrLen

The length of the address in *lpSockAddr* in bytes.

sendto

The Windows Sockets sendto function sends data to a specific destination.

```
int sendto (
  SOCKET s,
  const char FAR * buf,
  int len,
  int flags,
  const struct sockaddr FAR * to,
  int tolen
);
```

Parameters

s

[in] A descriptor identifying a (possibly connected) socket.

buf

[in] A buffer containing the data to be transmitted.

len

[in] The length of the data in *buf*.

flags

[in] An indicator specifying the way in which the call is made.

to

[in] An optional pointer to the address of the target socket.

tolen

[in] The size of the address in *to*.

Return Values

If no error occurs, sendto returns the total number of bytes sent, which can be less than the number indicated by *len*. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code can be retrieved by calling **WSAGetLastError**.

CAsyncSocket::Send

virtual int Send(const void* *lpBuf*, int *nBufLen*, int *nFlags* = 0);

Return Value

If no error occurs, Send returns the total number of characters sent. (Note that this can be less than the number indicated by *nBufLen*.) Otherwise, a value of SOCKET_ERROR is returned, and a specific error code can be retrieved by calling GetLastError.

Parameters

lpBuf

A buffer containing the data to be transmitted.

nBufLen

The length of the data in *lpBuf* in bytes.

nFlags

Specifies the way in which the call is made. The semantics of this function are determined by the socket options and the *nFlags* parameter. The latter is constructed by combining any of the following values with the C++ OR operator:

MSG_DONTROUTE Specifies that the data should not be subject to routing. A Windows Sockets supplier can choose to ignore this flag; see also the discussion of the SO_DONTROUTE option in Windows Sockets Programming Considerations in the Win32 SDK documentation.

MSG_OOB Send out-of-band data (SOCK_STREAM only).

recvfrom

The Windows Sockets recvfrom function receives a datagram and stores the source address.

```
int recvfrom (
  SOCKET s,
  char FAR* buf,
  int len,
  int flags,
  struct sockaddr FAR* from,
  int FAR* fromlen
);
```

Parameters

s

[in] A descriptor identifying a bound socket.

buf

[out] A buffer for the incoming data.

len

[in] The length of *buf*.

flags

[in] An indicator specifying the way in which the call is made.

from

[out] An optional pointer to a buffer that will hold the source address upon return.

fromlen

[in/out] An optional pointer to the size of the *from* buffer.

Return Values

If no error occurs, recvfrom returns the number of bytes received. If the connection has been gracefully closed, the return value is zero. Otherwise, a value of

SOCKET_ERROR is returned, and a specific error code can be retrieved by calling **WSAGetLastError**.

CAsyncSocket::ReceiveFrom

int ReceiveFrom(void* *lpBuf*, int *nBufLen*, CString& *rSocketAddress*, UINT& *rSocketPort*, int *nFlags* = 0);
int ReceiveFrom(void* *lpBuf*, int *nBufLen*, SOCKADDR* *lpSockAddr*, int* *lpSockAddrLen*, int *nFlags* = 0);

Return Value

If no error occurs, ReceiveFrom returns the number of bytes received. If the connection has been closed, it returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code can be retrieved by calling GetLastError. The following errors apply to this member function:

Parameters

lpBuf

A buffer for the incoming data.

nBufLen

The length of *lpBuf* in bytes.

rSocketAddress

Reference to a CString object that receives a dotted number IP address.

rSocketPort

Reference to a UINT that stores a port.

lpSockAddr

A pointer to a SOCKADDR structure that holds the source address upon return.

lpSockAddrLen

A pointer to the length of the source address in *lpSockAddr* in bytes.

nFlags

Specifies the way in which the call is made. The semantics of this function are determined by the socket options and the *nFlags* parameter. The latter is constructed by combining any of the following values with the C++ OR operator:
MSG_PEEK Peek at the incoming data. The data is copied into the buffer but is not removed from the input queue.
MSG_OOB Process out-of-band data (see Windows Sockets Programming Considerations in the Win32 SDK documentation for a discussion of this topic).

CAsyncSocket::Close

virtual void Close( );

Remarks

This function closes the socket. More precisely, it releases the socket descriptor, so that further references to it will fail with the error WSAENOTSOCK. If this is the last reference to the underlying socket, the associated naming information and queued data are discarded. The socket object's destructor calls Close for you.

closesocket

The Windows Sockets closesocket function closes an existing socket.

```
int closesocket (
  SOCKET s
);
```

Parameters

s

[in] A descriptor identifying a socket to close.

Return Values

If no error occurs, closesocket returns zero. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code can be retrieved by calling **WSAGetLastError**.

gethostbyaddr

The Windows Sockets gethostbyaddr function retrieves the host information corresponding to a network address.

```
struct HOSTENT FAR * gethostbyaddr (
  const char FAR * addr,
  int len,
  int type
);
```

Parameters

addr

[in] A pointer to an address in network byte order.

len

[in] The length of the address.

type

[in] The type of the address.

Return Values

If no error occurs, gethostbyaddr returns a pointer to the HOSTENT structure. Otherwise, it returns a NULL pointer, and a specific error code can be retrieved by calling **WSAGetLastError**.

gethostbyname

The Windows Sockets gethostbyname function retrieves host information corresponding to a host name from a host database.

```
struct hostent FAR * gethostbyname (
  const char FAR * name
);
```

Parameters

name

[out] A pointer to the null-terminated name of the host to resolve.

Return Values

If no error occurs, gethostbyname returns a pointer to the HOSTENT structure described above. Otherwise, it returns a NULL pointer and a specific error number can be retrieved by calling **WSAGetLastError**.

gethostname

The Windows Sockets gethostname function returns the standard host name for the local machine.

```
int gethostname (
 char FAR * name,
 int namelen
);
```

Parameters

name

[out] A pointer to a buffer that receives the local host name.

namelen

[in] The length of the buffer.

Return Values

If no error occurs, gethostname returns zero. Otherwise, it returns SOCKET_ERROR and a specific error code can be retrieved by calling **WSAGetLastError**.

RegOpenKeyEx

The RegOpenKeyEx function opens the specified key.

```
LONG RegOpenKeyEx(
 HKEY hKey,        // handle to open key
 LPCTSTR lpSubKey, // address of name of subkey to open
 DWORD ulOptions,  // reserved
 REGSAM samDesired, // security access mask
 PHKEY phkResult   // address of handle to open key
);
```

Parameters

hKey

Handle to a currently open key or any of the following predefined reserved handle values:

HKEY_CLASSES_ROOT

HKEY_CURRENT_CONFIG

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

Windows NT: HKEY_PERFORMANCE_DATA

Windows 95 and Windows 98: HKEY_DYN_DATA

lpSubKey

Pointer to a null-terminated string containing the name of the subkey to open. If this parameter is NULL or a pointer to an empty string, the function will open a new handle to the key identified by the *hKey* parameter. In this case, the function will not close the handles previously opened.

ulOptions

Reserved; must be zero.

samDesired

Specifies an access mask that describes the desired security access for the new key. This parameter can be a combination of the following values:

Value	Meaning
KEY_ALL_ACCESS	Combination of KEY_QUERY_VALUE, KEY_ENUMERATE_SUB_KEYS, KEY_NOTIFY, KEY_CREATE_SUB_KEY, KEY_CREATE_LINK, and KEY_SET_VALUE access.
KEY_CREATE_LINK	Permission to create a symbolic link.
KEY_CREATE_SUB_KEY	Permission to create subkeys.
KEY_ENUMERATE_SUB_KEYS	Permission to enumerate subkeys.

KEY_EXECUTE	Permission for read access.
KEY_NOTIFY	Permission for change notification.
KEY_QUERY_VALUE	Permission to query subkey data.
KEY_READ	Combination of KEY_QUERY_VALUE, KEY_ENUMERATE_SUB_KEYS, and KEY_NOTIFY access.
KEY_SET_VALUE	Permission to set subkey data.
KEY_WRITE	Combination of KEY_SET_VALUE and KEY_CREATE_SUB_KEY access.

phkResult

Pointer to a variable that receives a handle to the opened key. When you no longer need the returned handle, call the **RegCloseKey** function to close it.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value is a nonzero error code defined in WINERROR.H. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.

RegQueryValueEx

The RegQueryValueEx function retrieves the type and data for a specified value name associated with an open registry key.

```
LONG RegQueryValueEx(
  HKEY hKey,          // handle to key to query
  LPTSTR lpValueName, // address of name of value to query
  LPDWORD lpReserved, // reserved
  LPDWORD lpType,     // address of buffer for value type
  LPBYTE lpData,      // address of data buffer
```

LPDWORD *lpcbData* // address of data buffer size
);

Parameters

hKey

Handle to a currently open key or any of the following predefined reserved handle values:

HKEY_CLASSES_ROOT

HKEY_CURRENT_CONFIG

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

Windows NT: HKEY_PERFORMANCE_DATA

Windows 95 and Windows 98: HKEY_DYN_DATA

lpValueName

Pointer to a null-terminated string containing the name of the value to query.

If *lpValueName* is NULL or an empty string, "", the function retrieves the type and data for the key's unnamed or default value, if any.

Windows 95 and Windows 98: Every key has a default value that initially does not contain data. On Windows 95, the default value type is always REG_SZ. On Windows 98, the type of a key's default value is initially REG_SZ, but RegSetValueEx can specify a default value with a different type.

Windows NT: Keys do not automatically have an unnamed or default value.

Unnamed values can be of any type.

lpReserved

Reserved; must be NULL.

lpType

Pointer to a variable that receives the type of data associated with the specified value. The value returned through this parameter will be one of the following:

Value	Meaning

REG_BINARY	Binary data in any form.
REG_DWORD	A 32-bit number.
REG_DWORD_LITTLE_ENDIAN	A 32-bit number in little-endian format. This is equivalent to REG_DWORD. In little-endian format, a multi-byte value is stored in memory from the lowest byte (the "little end") to the highest byte. For example, the value 0x12345678 is stored as (0x78 0x56 0x34 0x12) in little-endian format. Windows NT, Windows 95, and Windows 98 are designed to run on little-endian computer architectures. A user may connect to computers that have big-endian architectures, such as some UNIX systems.
REG_DWORD_BIG_ENDIAN	A 32-bit number in big-endian format. In big-endian format, a multi-byte value is stored in memory from the highest byte (the "big end") to the lowest byte. For example, the value 0x12345678 is stored as (0x12 0x34 0x56 0x78) in big-endian format.
REG_EXPAND_SZ	A null-terminated string that contains unexpanded references to environment variables (for example, "%PATH%"). It will be a Unicode or ANSI string depending on whether you use the Unicode or ANSI functions. To expand the environment variable references, use the **ExpandEnvironmentStrings**

	function.
REG_LINK	A Unicode symbolic link.
REG_MULTI_SZ	An array of null-terminated strings, terminated by two null characters.
REG_NONE	No defined value type.
REG_RESOURCE_LIST	A device-driver resource list.
REG_SZ	A null-terminated string. It will be a Unicode or ANSI string depending on whether you use the Unicode or ANSI functions.

The *lpType* parameter can be NULL if the type is not required.

lpData

Pointer to a buffer that receives the value's data. This parameter can be NULL if the data is not required.

lpcbData

Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the *lpData* parameter. When the function returns, this variable contains the size of the data copied to *lpData*.

If the data has the REG_SZ, REG_MULTI_SZ or REG_EXPAND_SZ type, then *lpcbData* will also include the size of the terminating null character.

The *lpcbData* parameter can be NULL only if *lpData* is NULL.

If the buffer specified by *lpData* parameter is not large enough to hold the data, the function returns the value ERROR_MORE_DATA, and stores the required buffer size, in bytes, into the variable pointed to by *lpcbData*.

If *lpData* is NULL, and *lpcbData* is non-NULL, the function returns ERROR_SUCCESS, and stores the size of the data, in bytes, in the variable pointed to by *lpcbData*. This lets an application determine the best way to allocate a buffer for the value's data.

Window NT: If *hKey* specifies HKEY_PERFORMANCE_DATA and the *lpData* buffer is too small, RegQueryValueEx returns ERROR_MORE_DATA but *lpcbData* does not return the required buffer size. This is because the size of the performance data can change from one call to the next. In this case, you must increase the buffer size and call RegQueryValueEx again passing the updated buffer size in the *lpcbData* parameter. Repeat this until the function succeeds. You need to maintain a separate variable to keep track of the buffer size, because the value returned by *lpcbData* is unpredictable.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value is a nonzero error code defined in WINERROR.H. You can use the **FormatMessage** function with the FORMAT_MESSAGE_FROM_SYSTEM flag to get a generic description of the error.