CONTENIDO

RESUMEN

El Proyecto de Titulación trata sobre el desarrollo de una aplicación gráfica, basado en el sistema operativo LINUX para el monitoreo y administración del tráfico de datos en redes LAN. Esta aplicación utiliza los comandos de conectividad y administración que el sistema operativo LINUX ofrece.

En el primer capítulo, se realiza un estudio de los conceptos fundamentales para el monitoreo y administración de las redes LAN. También se realiza la descripción del protocolo SNMP y de cómo éste funciona en el ambiente LINUX. Por último, se estudia el sistema operativo LINUX y las facilidades que ofrece para el monitoreo y administración de redes LAN.

En el segundo capítulo, se realiza un estudio de los comandos del sistema operativo LINUX que se utilizan para monitoreo y administración del tráfico en redes LAN. Además se describe el desarrollo de la aplicación gráfica que permite monitorear y administrar el tráfico de datos. Finalmente se describen los requerimientos mínimos de Software y Hardware, con los que debe cumplir el host donde se instala la aplicación.

En el tercer capítulo, se describe las pruebas realizadas por el programa de monitoreo y administración de tráfico de datos. Además, de los resultados obtenidos en éstas pruebas, se hace la interpretación de los resultados.

En el cuarto capítulo, se indican las conclusiones y recomendaciones que fueron obtenidas en el desarrollo de éste Proyecto de Titulación.

PRESENTACIÓN

Las necesidades de incrementar y mejorar el uso de las redes de información ha provocado que la administración y monitoreo de las mismas sea un factor preponderante en el campo de las telecomunicaciones, para que se pueda mantener un adecuado funcionamiento. Es aquí donde se hace necesaria una herramienta, que nos facilite el monitoreo y administración de tráfico de datos en redes LAN mediante el uso del sistema operativo LINUX.

A pesar de que los actuales enlaces y conexiones a Internet soy muy rápidas, es normal encontrar enlaces que tengan distintas necesidades, es decir, con conexiones que nos sirve de soporte para varios tipos de servicios. Para lo cual se debe tener bien definido que servicios son los fundamentales y el ancho de banda que requieren, para que los administradores de red distribuyan este recurso, logrando obtener un servicio mejorado en cuanto a la velocidad de conexión.

La necesidad de tener un recurso de open source es muy importante en lugares donde la administración y control de redes LAN resultan muy complejos y costosos, por los paquetes que actualmente existen en el mercado. Estos programas pueden también resultar complejos en su uso para los administradores de red

El presente trabajo procura el desarrollo de una aplicación gráfica basado en el sistema operativo LINUX para el monitoreo y administración del tráfico de datos en redes LAN. Empleando los comandos y utilidades que el sistema operativo LINUX posee para la administración y monitoreo de tráfico.

CAPÍTULO 1

MONITOREO Y ADMINISTRACIÓN DE REDES

1.1 INTRODUCCIÓN A LA ADMINISTRACIÓN Y MONITOREO DE REDES

1.1.1 DEFINICIÓN Y OBJETIVOS DE LA ADMINISTRACIÓN DE REDES

1.1.1.1 ADMINISTRACIÓN

La administración de redes consiste en la organización, control, toma de precauciones y supervisión de la red, para mantener su funcionamiento eficiente, mediante el empleo de herramientas de red, aplicaciones y dispositivos.

A continuación se destaca un conjunto de actividades que a corto plazo permiten realizar un seguimiento de las tareas administrativas y elaborar informes periódicos para su posterior estudio:

- Detección y aislamiento de fallas.
- Evaluación del tráfico de datos.
- Mantenimiento de registro histórico de problemas.
- Mantenimiento de configuraciones.
- Contabilidad de red.
- Control de acceso. [1]

1.1.1.2 OBJETIVOS DE LA ADMINISTRACIÓN DE RED

Los objetivos de la administración de red son los siguientes:

- Proporcionar herramientas automatizadas y manuales de administración de red para controlar posibles fallas o degradaciones en el desempeño de la misma.[2]
- Disponer de estrategias de administración para optimizar la infraestructura existente, optimizar el rendimiento de aplicaciones y servicios. Además, prever los crecimientos en la red esperados debido al cambio constante en la tecnología.

1.1.1.3 FUNCIONES DE ADMINISTRACIÓN DE RED [3]

Las funciones de administración de red se basan en dos procedimientos que ayudan a llevar a cabo numerosas tareas, estos procedimientos son los siguientes:

- **Monitoreo**

  El monitoreo es un proceso eminentemente pasivo, el cual se encarga de observar el estado y comportamiento de la configuración de red y sus componentes.

  También se encarga de agrupar todas las operaciones para la obtención de datos acerca del estado de los recursos de la red.

- **Control**

  El control es un proceso que se lo considera activo, debido a que permite tomar información de monitoreo y actuar sobre el comportamiento de los componentes de la red administrada.

  Abarca la configuración y seguridad de la red, como por ejemplo, alterar parámetros de los componentes de la red.

1.1.2 DEFINICIÓN Y OBJETIVOS DEL MONITOREO DE REDES.

1.1.2.1 MONITOREO

Monitoreo es la realización del estudio del estado de los recursos. Las funciones del monitoreo de red se llevan a cabo por agentes que realizan el seguimiento y registro de la actividad de red, la detección de eventos y la comunicación de alertas. [4]

El monitoreo de una red abarca 4 fases:

- Definición de la información de administración que se monitorea.
- Acceso a la información.
- Diseño de políticas de administración.
- Procesamiento de la información.

Los tipos de monitoreo son:

- Local.
- Remoto.
- Automático.
- Manual.

Los elementos monitoreados pueden ser:

- En su totalidad.
- En Segmentos.

El monitoreo puede ser realizado en forma:

- Continua.
- Eventual. [1]

1.1.2.2 OBJETIVOS DEL MONITOREO

Los objetivos del monitoreo son los siguientes:

- Identificar la información a monitorear.
- Diseñar mecanismos para obtener la información necesaria.
- Utilizar la información obtenida dentro de las distintas áreas funcionales de administración de red.
- Tomar nuevas medidas sobre aspectos de los protocolos, colisiones, fallas, paquetes, etc.
- Almacenar la información obtenida en Bases de Información de gestión para su posterior análisis.
- Del análisis, obtener conclusiones para resolver problemas concretos o bien para optimizar la utilización de la red.

Dentro del monitoreo de la actividad de la red, los eventos típicos que son monitoreados suelen ser:

- Ejecución de tareas como la realización de copias de seguridad o búsqueda de virus.
- Registro del estado de finalización de los procesos que se ejecutan en la red.
- Registro de las entradas y salidas de los usuarios en la red.
- Registro del arranque de determinadas aplicaciones.
- Errores en el arranque de las aplicaciones, etc.

En función de la prioridad que tengan asignados los eventos y de la necesidad de intervención, se pueden utilizar diferentes métodos de notificación o alerta tales como:

- Mensajes en la consola: método en el que se suele codificar en función de su importancia.
- Mensajes por correo electrónico: método mediante el cual se envía contenido el nivel de prioridad y el nombre del evento ocurrido.

- Mensajes a móviles: método utilizado cuando el evento necesita intervención inmediata del administrador de red. [4]

1.2 ASPECTOS FUNCIONALES DE LA ADMINISTRACIÓN DE RED

La Organización Internacional de Estándares ISO (International Organization for Standardizations) ha definido la arquitectura de Administración OSI (Open System Interconnection), cuya función es permitir supervisar, controlar y mantener una red de datos.

Ésta arquitectura de administración, se encuentra dividida en cinco categorías de servicios de administración denominadas Áreas Funcionales Específicas de Administración, las cuales se muestran a continuación:

- Administración de prestaciones.
- Administración de fallas.
- Administración de contabilidad.
- Administración de configuraciones.
- Administración de seguridad.

Los aspectos o categorías funcionales de la administración de red brindan servicio a las actividades de Monitoreo y Control de red. Y se las puede ubicar de la siguiente manera:

a) Monitoreo de la red: obtiene información de los elementos:

- Administración de prestaciones.
- Administración de fallas.
- Administración de contabilidad.
- Administración de configuraciones. [1]

b) Control de la red: actúa sobre los elementos:

- Administración de configuraciones.
- Administración de seguridad. [1]

1.2.1 ADMINISTRACIÓN DE PRESTACIONES O RENDIMIENTO

Es medir la calidad de funcionamiento, proveer información disponible del desempeño de la red (hardware y software), asegurar que la capacidad y prestaciones de la red correspondan con las necesidades de los usuarios, analizar y controlar parámetros como: utilización, rendimiento, tráfico, cuellos de botella, tiempo de respuesta, tasa de error, throughput, etc.; esto de los distintos componentes de red como switches, ruteadores, hosts, etc., para poder ajustar los parámetros de la red, mantener el funcionamiento de la red interna en un nivel aceptable, poder efectuar análisis precisos y mantener un historial con datos estadísticos y de configuración, predecir puntos conflictivos antes de que éstos causen problemas a los usuarios.

El conocimiento de esta información nos permite en el futuro tomar acciones correctivas como balanceo o redistribución de tráfico, establecer y reportar tendencias para ser utilizadas en la toma de decisiones y planificación del crecimiento.

Se debe definir claramente los parámetros de funcionamiento o desempeño alrededor de los cuáles, se van a organizar las tareas de Administración de prestaciones como las siguientes:

- Obtención de la información de funcionamiento de la red a través del monitoreo sobre los recursos disponibles.
- Análisis de la información recolectada para determinar los niveles normales de utilización de la red.
- Comparación entre los valores obtenidos y los normales, para generar acciones de inicio de alarmas que pueden generar la toma de medidas preventivas o correctivas. [5]

1.2.2 ADMINISTRACIÓN DE FALLAS

Los objetivos de esta área son: detección, aislamiento, corrección, registro y notificación de los problemas existentes en la red, sondeo periódico en busca de mensajes de error y establecimiento de alarmas.
Las consecuencias de estas fallas pueden causar tiempo fuera de servicio o la degradación inaceptable de la red, por lo que es deseable su pronta *detección y corrección*.
La diferencia entre falla y error esta en que un error es un evento aislado como la pérdida de un paquete o que éste no llegue correctamente, pero una falla es un funcionamiento anormal que requiere una intervención para ser corregido. La falla se manifiesta por un funcionamiento incorrecto o por exceso de errores.

Las acciones o procedimientos para esta corrección son:

- Determinar exactamente dónde está la falla.
- Aislar el resto de la red, para que pueda seguir operando sin interferencia.
- Reconfigurar la red para minimizar el impacto de operar sin el componente averiado.[5]
- Reparar o reemplazar el componente averiado para devolver la red al estado inicial.
- Si es posible, ejecutar un proceso de corrección automática y lograr el funcionamiento óptimo de la red.
- Registrar la detección y la resolución de fallas.
- Hacer seguimiento de la reparación de fallas.

Una buena política de Administración, es la prevención, es decir, debe adelantarse a los posibles problemas y resolverlos antes de que se produzcan.

1.2.3 ADMINISTRACIÓN DE LA CONFIGURACIÓN

Es el proceso de preparación de los dispositivos, puesto que la configuración de éstos, determina el comportamiento de los datos en la red.
Las funciones de ésta administración son: inicialización, desconexión o desactivación ordenada de la red o de parte de ella, mantenimiento y adición de componentes, reconfiguraciones, definición o cambio de parámetros de configuración, denominación de los elementos de la red, conocimiento de que dispositivos hay en la red, hardware y configuraciones de software de dichos dispositivos. [5]

Las tareas que se presentan en la administración de configuración son:

- Es deseable que el arranque y parada de componentes específicos, se puedan realizar de forma remota.
- Definir información de configuración de recursos.
- Mantener ésta información, por si se sufre un ataque, poder realizar una comprobación de la información de configuración para asegurar que permanece en un estado correcto.
- Modificación de propiedades de recursos e información al usuario de estos cambios.
- Control de versiones de software.
- Actualización de software.
- Establecer qué usuarios pueden utilizar qué recursos.
- Inicialización y finalización de servicios de red.

Las herramientas típicas para ésta administración son: monitorear la red para ver qué elementos hay activos y con qué características obtener la información, para saber de qué modo están conectados entre sí los diferentes elementos, ésta información se mantiene para ayudar a otras funciones de administración.

1.2.4 ADMINISTRACIÓN DE LA CONTABILIDAD

Su objetivo es controlar el grado de utilización de los recursos de red, controlar el acceso de usuarios y dispositivos a la red, obtener informes, establecer cuotas de uso, asignar privilegios de acceso a los recursos.
Finalmente se debe hacer un seguimiento del uso de recursos de la red por parte de un usuario o grupo de usuarios. Todo esto para regular apropiadamente las aplicaciones de un usuario o grupo y además permitir una buena planificación para el crecimiento de la red. [5]

Los objetivos de la función de administración de contabilidad son:

- Vigilancia de abuso de privilegios de acceso.
- Evitar sobrecargas en la red y perjuicios a otros usuarios.
- Uso ineficiente de la red.
- Modificar la forma de trabajo para mejorar prestaciones.
- Planificación del crecimiento de la red.
- Saber si cada usuario o grupo tiene lo que necesita.
- El aumento o disminución de derechos de acceso a recursos.

1.2.5 ADMINISTRACIÓN DE LA SEGURIDAD

Su objetivo es controlar el acceso a los recursos de la red, y protegerla de modo que no pueda ser dañada (intencional o involuntariamente), y que la información que es vulnerable pueda ser utilizada con una autorización apropiada. Comprende el conjunto de facilidades mediante las cuales, el administrador de la red modifica la funcionalidad que proporciona la red frente a intentos de acceso no autorizados.

En la Administración de Seguridad se pueden tener dos tipos de ataques:

- Ataques Activos.
- Ataques Pasivos. [1]

1.2.5.1 ATAQUES ACTIVOS

En este tipo de ataques existe evidencia del hecho por mal funcionamiento de componentes o servicios, o por sustitución de usuarios en ejecución de tareas orientados a tratar de conseguir información privilegiada o interrumpir un servicio crítico para la organización, puede ser desde el interior o del exterior.
Ejemplos de estos ataques son: modificación del contenido de los datos que circulan por la red, alteración del orden de llegada de los datos, supresión de mensajes con un destino particular, saturación de la red con datos inútiles para degradar la calidad de servicio, engaño de la identidad de un host o usuario para acceder a datos confidenciales, desconfiguraciones para sabotaje de servicios. [1]

1.2.5.2 ATAQUES PASIVOS

Ataques difíciles de detectar, ya que no se produce evidencia física del ataque pues no hay alteración de datos ni mal funcionamiento o comportamiento fuera de lo habitual de la red, escucha o “intercepción del tráfico de la red y los servicios involucrados”, estudio de parámetros de configuración de manera ilegal por parte del intruso, robo de información sensible para las organizaciones. [1]

Para cualquiera de los tipos de ataques se puede prevenir o solucionar a través de las siguientes actividades:

- Fortalecer políticas de administración y asignación de claves.
- Historiales de seguridad, para posterior análisis.
- Uso de cortafuegos para monitorear y controlar los puntos de acceso internos y externos a la red.
- Encriptar o cifrar de la información enviada por la red.
- Localizar la información importante.
- Registrar los usuarios que consultan dicha información y durante qué períodos de tiempo, así como los intentos fallidos de acceso.
- Señales de alarma.
- Establecimiento de mecanismos y políticas de prevención.

- Sistemas de detección de intrusos.
- Sensibilización de seguridad en el usuario.
- Mantenimiento del sistema operativo y sus aplicaciones relacionadas.
- Utilizar herramientas de monitoreo en los diferentes niveles.
- Configurar de manera segura los elementos y servicios de red.

1.3 MODELOS DE ADMINISTRACIÓN DE RED

Es la estandarización del empleo de una variedad de herramientas de red, aplicaciones y dispositivos para la administración de red. Para permitir que los componentes (de distintos fabricantes o proveedores) que conforman una red, y los sistemas operativos de los hosts puedan ínteroperar con el Sistema de Administración de Red.

Para la Administración de Red existen tres modelos fundamentales:

- Administración de Red OSI.
- Administración Internet.
- Arquitectura TMN (Telecommunications Management Network).

Administración de Red OSI. Definido por ISO, con el objetivo de lograr la administración de los recursos según el modelo de referencia OSI.

Administración Internet. Definido por la Fuerza de Tareas de Ingeniería de Internet IETF (Internet Enginnering Task Force) y la IAB (Internet Activities Board), para administrar según la arquitectura de red TCP/IP (Protocolo de Control de Transporte/ Protocolo de Internet, Transport Control Protocol / Internet Protocol).

Arquitectura TMN (Red de Administración de Telecomunicaciones). Definida por la ITU-T(Unión Internacional de Telecomunicaciones). Más que un modelo de red, define una estructura de red basada en los modelos anteriores. **[6]**

Los modelos OSI e Internet se refieren a redes de hosts, mientras que el modelo TMN es de utilidad para los grandes operadores de redes de telecomunicaciones.

1.3.1 MODELO DE ADMINISTRACIÓN INTERNET

El Modelo de Administración Internet depende de la existencia en cada dispositivo de "Agentes SNMP Protocolo Simple de Administración de Red (Simple Network Management Protocol)", que principalmente se encargan de la recolección de la información sobre dicho dispositivo, ésta información se puede dividir en tres tipos:

- Información de estado
- Advertencias, y
- Alarmas.

La información que los agentes recogen, la envían a una aplicación central que controla el sistema. Esta se compone de una base de datos jerárquica o MIB (Management Information Base), por un lado, y una consola de administración, por el otro. [7]

Para la gestión en Internet, el protocolo SNMP trabaja con otros componentes que cooperan con éste. Así, en el nivel superior, en la gestión se tiene:

- Estructura de Información de Gestión (SMI, Structure of Management Information), y
- Base de Información de Gestión (MIB, Management Information Base).

SNMP utiliza los servicios ofrecidos por estos dos componentes para realizar su trabajo. Los componentes del modelo de Administración de Internet se muestran en la figura 1.1. [8]

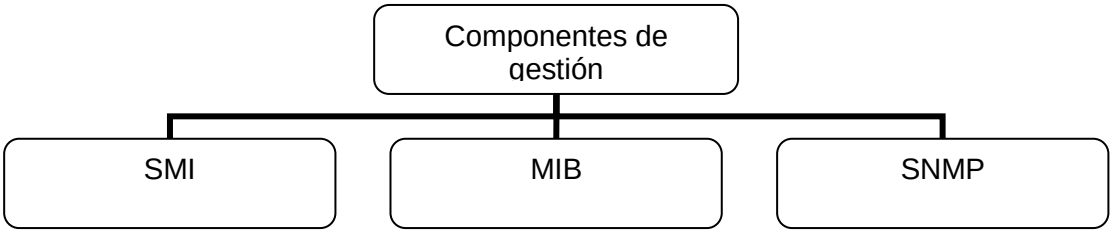


Figura 1.1: Componentes del Modelo de Administración Internet.

1.3.1.2 ESTRUCTURA E IDENTIFICACIÓN DE LA INFORMACIÓN DE GESTIÓN SMI (STRUCTURE AND IDENTIFICATION OF MANAGEMENT INFORMATION)

El SMI define las reglas para describir los objetos gestionados y cómo los protocolos sometidos a la gestión pueden acceder a ellos. La descripción de los objetos gestionados se hace utilizando ASN.1 (Abstract Syntax Notation 1, estándar ISO 8824), que es un lenguaje de descripción de datos.

1.3.1.3 BASE DE INFORMACIÓN DE GESTIÓN (MIB) [8]

Una MIB define un modelo conceptual de la información requerida para tomar decisiones de administración de red. La información que la MIB incluye tiene número de paquetes transmitidos, número de conexiones intentadas, datos de contabilidad, entre otros.

1.3.1.3.1 Especificaciones de la MIB [9]

La MIB define tanto los objetos de la red operados por el protocolo de administración de red, como las operaciones que pueden aplicarse a cada objeto. La MIB no incluye información de administración para aplicaciones como Telnet, FTP o SMTP, debido a los inconvenientes que se presentan al instrumentar aplicaciones de este tipo para la MIB por parte de las compañías fabricantes.

Para definir una variable u objeto MIB es necesario especificar lo siguiente:
Sintaxis: Especifica el tipo de datos de la variable, un valor entero, etc.
Acceso: Especifica el tipo de permiso como: Leer, leer y escribir, escribir, no accesible.
Estado: Define si la variable es obligatoria u opcional.
Descripción: Describe textualmente a la variable.

1.3.1.3.2 Grupos de la MIB [9]

La MIB-1 define 126 objetos de administración, divididos en los siguientes grupos:

- **Grupo de Sistemas**

  Usado para registrar información del sistema, por ejemplo:
  Compañía fabricante del sistema.
  Tipo de Software.
  Tiempo que el sistema ha estado operando.

- **Grupo de Interfaces**

  Registra la información genérica acerca de cada interfaz de red, como el número de mensajes erróneos en la entrada y salida, el número de paquetes transmitidos y recibidos, el número de paquetes de broadcast enviados, MTU del dispositivo, etc.

- **Grupo de traducción de dirección**

  Comprende las relaciones entre direcciones IP y direcciones específicas de la red que deben soportar, como la tabla ARP, que relaciona direcciones IP con direcciones físicas de la red LAN.

- **Grupo IP**

  Almacena información propia de la capa IP, como datagramas transmitidos y recibidos, conteo de datagramas erróneos, etc. También contiene información de variables de control que permite a las aplicaciones remotas ajustar el TTL (Time To Live) de omisión de IP y manipular las tablas de ruteo de IP.

- **Grupo TCP**

  Este grupo incluye información propia del protocolo TCP, como estadísticas del número de segmentos transmitidos y recibidos, información acerca de conexiones activas como dirección IP, puerto o estado actual.

- **Grupo de ICMP y UDP**

  Lo mismo que el grupo IP y TCP.

- **Grupo EGP**

  En este grupo se requieren sistemas (ruteadores) que soporten EGP(Protocolo de Gateway o Salida Exterior).

1.3.1.3.3 Base de Información de Gestión II (MIB-II) [9]

La MIB-II se crea para extender los datos de administración de red empleados en redes Ethernet y WAN(Wide Area Network) usando ruteadores para un enfoque a múltiples medios de administración en redes LAN y WAN. Se agregan dos grupos:

- **Grupo de Transmisión**

  Soporta múltiples tipos de medios de transmisión, como cable coaxial, cable UTP, cable de fibra óptica y sistemas T1/E1.

- **Grupo SNMP**

  Incluye estadísticas sobre tráfico de red SNMP.

1.3.1.3.4 Formato del árbol de Identificadores de Objeto (OID) [10]

El nombre mediante el cual puedan ser identificados de manera única los dispositivos se le denomina identificador de objeto (*object identifier)*. Éste es escrito como una secuencia de enteros separados por puntos; por ejemplo, la secuencia 1.3.6.1.2.1.1.1, especifica la descripción del sistema dentro del grupo system, del subárbol sysDescr.

Las figuras 1.2, 1.3 y 1.4 definen un ejemplo de la secuencia numérica; similar a un árbol con raíz y muchas ramas enlazadas directamente. Se puede usar la estructura de raíz, rama, subramas y hojas para diagramar todos los objetos de una MIB y sus relaciones.

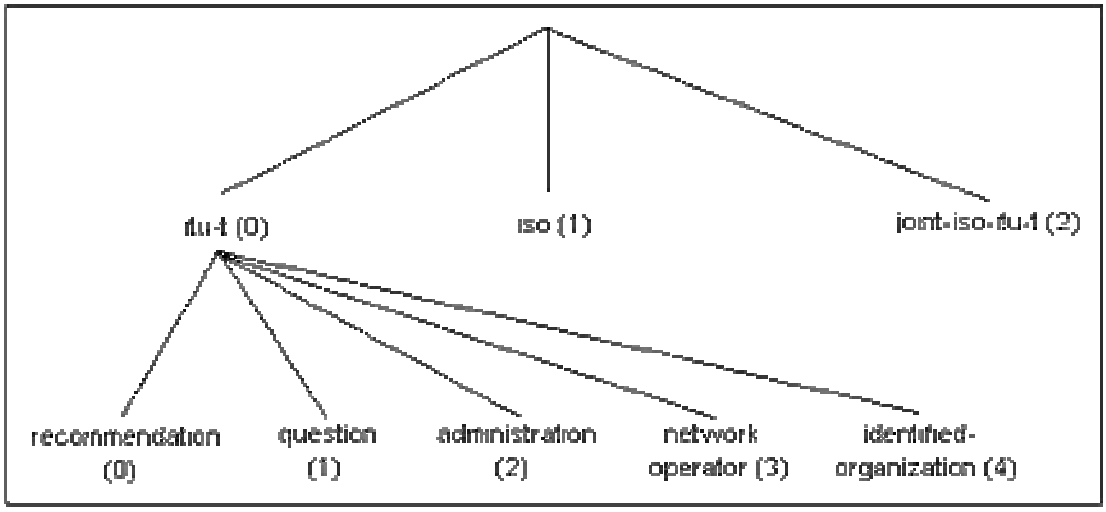


Figura 1.2.- Nodo raíz y valores de los componentes asignados para la ITU-T.

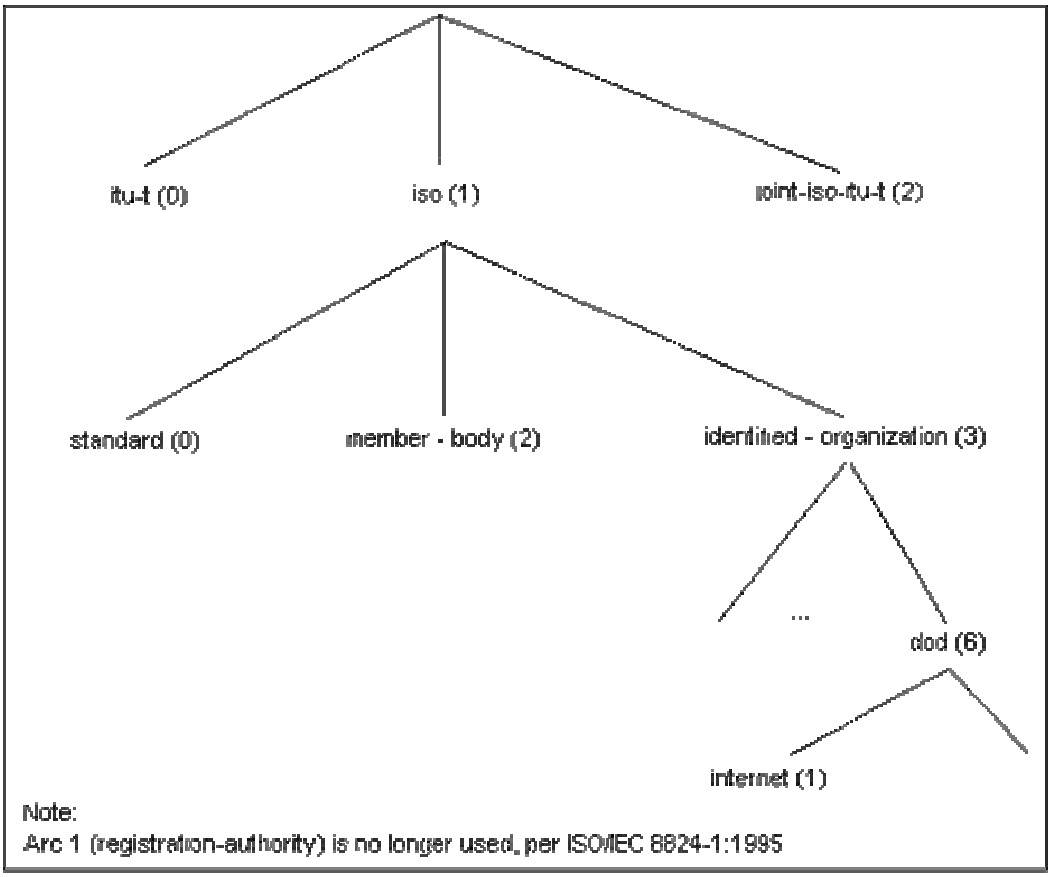


Figura 1.3.- Nodo raíz y valores asignados para la ISO.

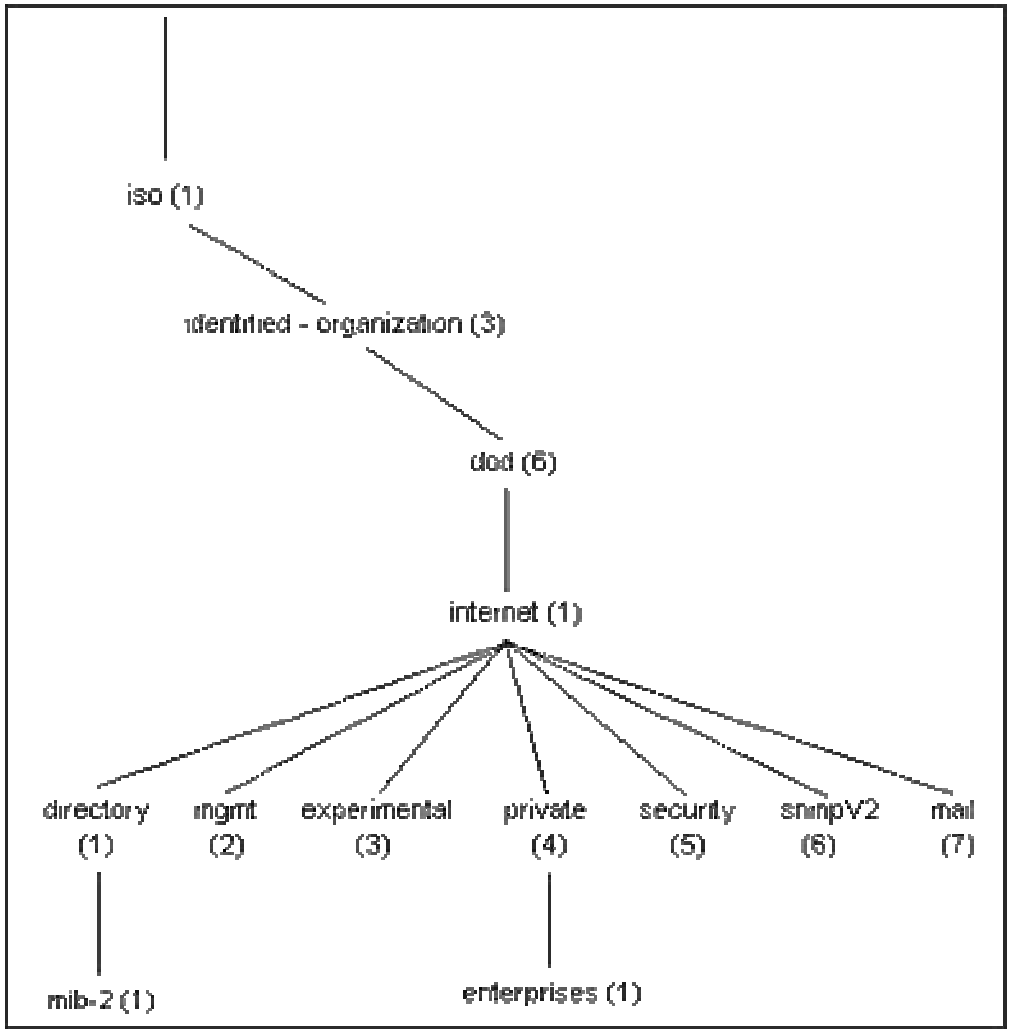


Figura 1.4: Valores de los componentes asignados para Internet.

De ésta forma, los grupos de trabajo de la IETF desarrollan MIBs, las cuales son puestas bajo una rama del árbol experimental, hasta que éstas MIBs sean publicadas en la rama Internet.

1.3.1.4 PROTOCOLO SIMPLE DE GESTIÓN DE RED SNMP (SIMPLE NETWORK MANAGEMENT PROTOCOL)

El protocolo SNMP define la forma más básica para el intercambio de información de gestión de redes donde existen: un gestor, agentes y bases de información de gestión.

La simplicidad que presenta produce deficiencias para transferir grandes cantidades de información, poca ó ninguna seguridad, entre otras cosas.

El protocolo SNMP funciona en la capa aplicación. SNMP ofrece un entorno de trabajo estandarizado y un lenguaje común empleado para la monitoreo y gestión de los dispositivos de una red, éste entorno consta de tres partes:

- Gestor SNMP
- Agente(s) SNMP
- MIB (Base de Información de Gestión)

El protocolo SNMP proporciona un formato de mensajes para el intercambio de información entre gestores y agentes de SNMP, ya que funciona bajo el ambiente cliente/servidor.
Los agentes SNMP emplean el puerto UDP(Protocolo de Datagramas de Usuario) 161, donde mantienen la escucha de peticiones por parte del gestor SNMP. El gestor por medio del puerto UDP 162 recibe las notificaciones que genera el o los agentes y en donde debe existir un proceso gestor de interrupciones (trap manager) que las procese. **[2]**

1.3.1.4.1 Gestores y Agentes [11]

La gestión se realiza a través de una sencilla interacción entre un gestor y un agente.
El agente almacena información de sistema, de interfaces, etc, en una base de información de gestión, mientras que el gestor tiene acceso a los valores de esta base.
Los agentes también pueden enviar al gestor mensajes de advertencia (denominados *traps*), si el programa que se ejecuta en el agente detecta algún error en el entorno.

1.3.1.4.2 Arquitectura de SNMP [12]

Se muestra a continuación los componentes del protocolo SNMP:

- Una estación de gestión.
- Un agente de gestión (incluidos agentes proxy).
- Una base de información de gestión (MIB).
- Protocolo de gestión de red.

Los elementos de la estación de gestión son los siguientes:

- Aplicaciones (para análisis de datos, etc.)
- Interfaz de usuario.
- Capacidad de convertir las solicitudes del usuario a peticiones de monitoreo, y
- Control a los elementos remotos y base de datos con información de las MIBs de los elementos de la red gestionados.

En la figura 1.5 se indica una arquitectura común de Administración de Red.

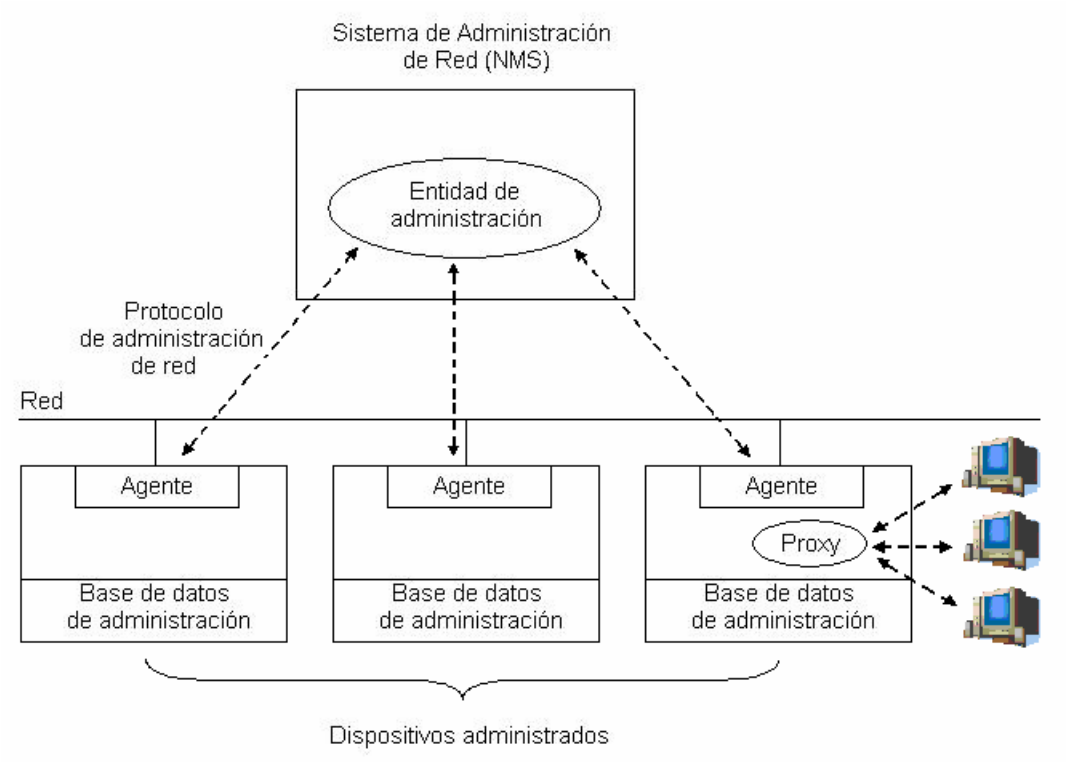


Figura 1.5.- Arquitectura común de Administración de Red. [13]

1.3.1.4.3 Tipos de mensajes SNMP [12]

En la figura 1.6 se muestran los diferentes tipos de mensajes que intercambian el gestor y el agente SNMP.

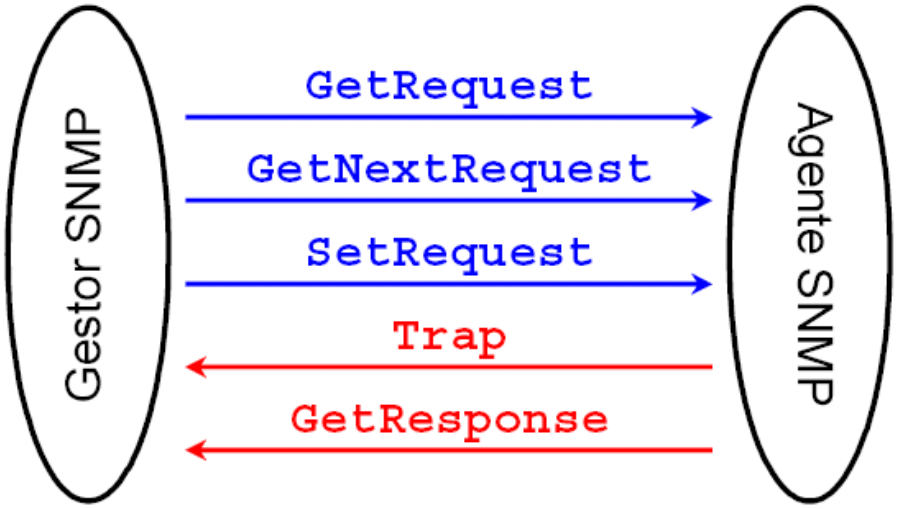


Figura 1.6.- Mensajes entre Gestor y Agente SNMP. [12]

Los diferentes tipos de mensajes se describen a continuación:

- **GetRequest:** el gestor pide al agente el valor de un dato.
- **GetNextRequest** es similar al *GetRequest*, permitiendo extraer datos de una tabla.
- **SetRequest:** el gestor pide al agente que modifique los valores de las variables que especifique. El agente modificará todos o ninguno de los valores.
- **GetResponse:** Respuesta del agente a las peticiones *GetRequest, GetNextRequest* y *SetRequest*.
- **Trap**: Mensaje generado por el agente en respuesta a un evento que afecte a la MIB o a los recursos gestionados. El gestor no confirma la recepción de un *trap* al agente.

1.3.1.4.4 Comunidades SNMP [12]

A continuación se describen las diferentes características que deben cumplir los agentes y comunidades SNMP:

- Cada agente es responsable de su MIB local, controlando sus estaciones gestoras.
- Control de acceso a la MIB de un agente: concepto de comunidad.
- Comunidad es la relación que se tienen entre un agente SNMP y un conjunto de estaciones de gestión SNMP, definen unas características de autentificación y control de acceso.
- El agente establece una comunidad para cada combinación deseada de autentificación y control de acceso, y a cada comunidad se le da un nombre de comunidad (community name) que es su nombre único dentro del agente. Este nombre las estaciones de gestión pertenecientes a una comunidad la emplean en todas las operaciones *GetRequest*, *GetNextRequest* y *SetRequest*.
- El agente puede establecer cualquier número de comunidades. Y a su vez una estación de gestión puede pertenecer a varias comunidades.
- Una estación de gestión debe almacenar los nombres de comunidad asociados a cada agente.
- Mediante el uso de comunidades, un agente puede limitar el acceso a su MIB en dos formas:
  - Vistas de la MIB: subconjunto de los objetos de la MIB.
  - Modos de Acceso: solo lectura o solo escritura.
- El agente sólo atiende la petición si el nombre de la comunidad es correcto para el tipo de acceso solicitado.

En la figura 1.7 se muestra un ejemplo de los modos de acceso a la MIB.

Figura 1.7.- Modos de Acceso a la MIB. [2]

1.3.1.5 PROTOCOLO SIMPLE DE GESTIÓN DE RED VERSIÓN 2 (SNMPV.2) [14]

Como resultado de una serie de propuestas para mejorar las características de SNMP, en 1996 se publica un nuevo estándar, el protocolo SNMPv.2. Los cambios se traducen fundamentalmente en una mejora de las prestaciones de intercambio de información de gestión, como la implementación de seguridad que fue una de las principales limitaciones de SNMPv.1.

1.3.1.5.1 Seguridad

SNMPv.2 define métodos para controlar las operaciones que están permitidas, a diferencia de SNMPv.1 que no incorpora ningún mecanismo de seguridad.
Pero luego, surgieron dos planteamientos diferentes en cuanto al modelo de seguridad, que han dado lugar a dos especificaciones conocidas como SNMPv.2* (La versión SNMPv.2* proporciona niveles de seguridad adecuados, pero no alcanzó el necesario nivel de estandarización y aceptación por el IETF) y SNMPv.2u (La identificación se lleva a cabo mediante usuarios. Es decir que un mismo usuario puede estar definido en varias entidades SNMP diferentes).

1.3.1.5.2 Principales características de SNMPv.2

Ésta versión ya permite el soporte de comunicación gestor-gestor (PDU-Protocol Data Unit- *informrequest*). La lectura de tablas completas en una sola operación (PDU *getbulkrequest*), es otra de las variantes. Pero la seguridad sigue siendo una falencia, ya que todavía se basa en “comunidades” (un nombre con un conjunto de operaciones permitidas) hasta la versión 3.

Los diferentes tipos de mensajes que se manejan en SNMPv.2 son los siguientes:

- get-request
- get-next-request
- get-bulk-request
- response
- set-request
- inform-request
- snmpV2-trap
- report (aparecía en los documentos de seguridad y después se eliminó junto con ellos).

1.3.1.5.3 Interoperabilidad con SNMPv.1

La versión 2 de SNMP permite ínteroperar con SNMPv.1, bajo las siguientes características:

- Los agentes pueden *emplear* tanto la versión 1 como la 2.
- Los gestores deben *emplear* tanto la versión 2 como la 1 para poder comunicarse con agentes de versión 1.
- El gestor intentará primero *emplear* la versión 2 con un agente, si:
  - El agente *emplea* versión 2 le contestará.
  - El agente *emplea* versión 1 responderá con un mensaje de error; el gestor entonces, lo intentará con versión 1, para ello deberá hacer conversiones (por ejemplo, convertir *getbulk* en *getnext).*

1.3.1.6 PROTOCOLO SIMPLE DE GESTIÓN DE RED VERSIÓN 3 (SNMPV.3) [15]

La falta de consenso en la versión final de SNMPv.2 (SNMPv.2u y SNMPv.2*) finalmente hizo que las mejoras a la seguridad sean descartadas.
En 1998, a partir de estas dos versiones anteriores, surge SNMPv.3 con las siguientes características:

- SNMPv.3 no modifica las PDUs de SNMPv.2
- Define una serie de capacidades de seguridad y un marco que hace posible su uso junto con las PDUs de SNMPv.2
- SNMPv.3 es SNMPv.2 con mayor seguridad y administración.

1.3.1.6.1 Mejoras en la seguridad en SNMPv.3

SNMPv.3 debió definir un conjunto de mecanismos de seguridad que incluyen la protección contra las amenazas de:

- Modificación de la información.
- Enmascaramiento.
- Modificación del flujo de mensajes.
- Revelación de contenidos.

Pero no se contempla la protección frente a:

- Denegación de servicio.
- Análisis de tráfico.

1.3.1.6.2 Arquitectura de Gestión de red en SNMPv.3

SNMPv.3 define una arquitectura de gestión de red:

- Colección de entidades SNMP que interaccionan entre sí.
    - Cada entidad implementa una parte de las capacidades de SNMP y puede actuar como agente, gestor o una combinación de ambos

- Cada entidad consiste en una colección de módulos que interaccionan entre sí para proporcionar servicios.
    - Una entidad incluye un motor SNMP, este motor implementa funciones para enviar y recibir mensajes, autentificar y encriptar/desencriptar mensajes, y controlar el acceso a los objetos gestionados.

1.3.2 OTROS PROTOCOLOS DE ADMINISTRACIÓN DE RED [2]

CMIP(Common Management Information Protocol) es una arquitectura de Administración de Red que provee un modo para que la información de control y de mantenimiento sea intercambiada entre un gestor y un elemento remoto de red. Las operaciones CMIS (Common Management Information Services) se pueden originar tanto en gestores como en agentes, permitiendo relaciones simétricas o asimétricas entre los procesos de administración. Sin embargo, la mayor parte de los dispositivos contienen las aplicaciones que sólo le permiten hacer de agente.

CMIP se implementa sobre el modelo OSI. Además existe una variante del mismo llamado CMOT (Common Management Information Protocol over TCP/IP) que se implementa sobre un modelo de red TCP/IP.

1.4 CARACTERÍSTICAS DEL SISTEMA OPERATIVO LINUX

1.4.1 INTRODUCCIÓN AL SISTEMA OPERATIVO GNU/LINUX [16] [17]

GNU/LINUX se ha desarrollado según las normas POSIX (Portable Operating System Interface) y en base al sistema UNIX; es capaz de ejecutar aplicaciones en modo gráfico, aplicaciones TCP/IP, edición de texto, transferencia de archivos entre sistemas Unix, software de correo, etc.

Cabe mencionar que el sistema gráfico de GNU/LINUX no es tan potente como el de texto pero puede ofrecer un ambiente más simple y cómodo para el usuario.

GNU/LINUX es un sistema operativo compatible *UNIX*. Se caracteriza porque es *libre* lo que implica que no hay que pagar ningún tipo de licencia para su uso, y por ser un sistema *abierto*, lo que significa que su código es público. El sistema está conformado por el *núcleo* o *kernel* y una serie de programas y bibliotecas que hacen posible su utilización. GNU/LINUX y gran parte de sus aplicaciones, bibliotecas y programas son distribuidos bajo la licencia GPL(Licencia Pública GNU).

1.4.1.1 SOFTWARE LIBRE Y OPEN SOURCE

Desde su origen LINUX se desarrolló bajo licencia pública GNU, lo que implica que se distribuye de manera gratuita y junto al código fuente de las aplicaciones incluidas, se puede copiar y distribuir libremente, o modificar el código fuente para mejorarlo o adaptarlo a nuestras necesidades, pero sin dejar de mencionar el autor original y de mantenerlo bajo la misma licencia.

Por otra parte, la Fundación de Software Libre FSF (Free Software Foundation), mediante su proyecto GNU, produce software (desde 1984) que puede ser utilizado libremente. En este modelo, el negocio no está en la ocultación del

código, sino en el software complementario añadido, en la adecuación del software a los clientes y en los servicios agregados, como el mantenimiento y la formación de usuarios.
Frente a un código de tipo propietario, en el cual un fabricante (empresa de software) encierra su código, ocultándolo y restringiéndose los derechos a sí misma, sin dar posibilidad de realizar ninguna adaptación ni cambios que no haya realizado previamente la empresa fabricante.

Los programas con código abierto ofrecen las siguientes consideraciones:

- Acceso al código fuente, ya sea para estudiarlo o modificarlo, para corregir errores, adaptarlo o añadir más prestaciones.
- Gratuidad: normalmente, el software, ya sea en forma binaria o en la forma de código fuente, puede obtenerse libremente o por una módica cantidad en concepto de gastos de empaquetamiento, distribución y valores añadidos.
- Evitar monopolios de software propietario: no depender de una única opción o único fabricante del software. Esto es más importante cuando se trata de una gran organización, ya sea una empresa o estado, los cuales no pueden (o no deberían) ponerse en manos de una determinada única solución y pasar a depender exclusivamente de ella.
- Un modelo de avance, no basado en la ocultación de información, sino en la compartición del conocimiento, para lograr progresos de forma más rápida, con mejor calidad, ya que las elecciones tomadas están basadas en el consenso de la comunidad.

1.4.1.2 QUE ES KERNEL O NÚCLEO DE GNU/LINUX

El *kernel* del sistema GNU/LINUX (al que habitualmente se le denomina LINUX) es el corazón del sistema: se encarga de arrancar el sistema, para que sea utilizable por las aplicaciones y los usuarios. Gestiona los recursos del host para la gestión de la memoria, sistema de archivos, entrada/salida, procesos e intercomunicación de procesos.

El núcleo o *kernel* es la parte básica de cualquier sistema operativo, y en él descansa el código de los servicios fundamentales para controlar el sistema entero. Básicamente, su estructura se puede separar en:

- *Gestión de procesos:* qué tareas se van a ejecutar y en qué orden y prioridad. Un aspecto importante es la planificación de CPU(Unidad Central de Procesamiento), cómo se optimiza el tiempo de la CPU para ejecutar las tareas con el mayor rendimiento o interactividad posible con los usuarios.
- *Intercomunicación de procesos y sincronización:* cómo se comunican tareas entre sí, con qué diferentes mecanismos y cómo pueden sincronizarse grupos de tareas.
- *Gestión entrada/salida (E/S):* control de periféricos y gestión de recursos asociados.
- *Gestión de memoria*: optimización del uso de la memoria, sistema de paginación y memoria virtual.
- *Gestión de archivos*: cómo el sistema controla y organiza los archivos presentes en el sistema y el acceso a los mismos.

1.4.2 CARACTERÍSTICAS DEL SISTEMA OPERATIVO LINUX [16] [17]

El sistema operativo LINUX fue desarrollado buscando la portabilidad de las fuentes: casi todo el software gratuito desarrollado para UNIX se compila en LINUX sin problemas. Y todo lo que se hace para LINUX (código del núcleo, controladores, librerías y programas de usuario) es de libre distribución.

En LINUX también se implementa el control de trabajos POSIX (que se usa en los shells csh y bash) y las pseudoterminales (dispositivos pty). Además, soporta consolas virtuales, lo que permite tener más de una sesión abierta en la consola de texto y conmutar entre ellas fácilmente.

LINUX soporta diversos sistemas de archivos para guardar los datos. Algunos de ellos, como el ext2fs, han sido desarrollados específicamente para LINUX.

LINUX implementa todo lo necesario para trabajar en red con TCP/IP. Desde controladores para las tarjetas de red más populares hasta SLIP/PPP, que permiten acceder a una red TCP/IP por puerto serial. También se implementan PLIP (para comunicarse por el puerto de la impresora) y NFS (para acceso remoto a archivos). Y también soporta los clientes de TCP/IP, como FTP, Telnet, y SMTP.

Con el fin de incrementar la memoria RAM (Memoria de Acceso Randómico) disponible, LINUX implementa la paginación con el disco duro, es decir, puede tener varios megabytes de espacio de intercambio o "swap" con el disco duro. Cuando el sistema necesita más memoria, expulsará páginas inactivas al disco, permitiendo la ejecución de programas más grandes o aumentando el número de usuarios que puede atender a la vez. Sin embargo, el espacio de intercambio no puede suplir totalmente a la memoria RAM, ya que el primero es mucho más lento que ésta.

1.4.2.1 MULTITAREA

La palabra multitarea describe la capacidad de ejecutar muchos programas al mismo tiempo sin detener la ejecución de cada aplicación.

Se le denomina *multitarea prioritaria o preventiva* ya que posibilita la ejecución simultánea de varios programas, siempre que las características del host lo permitan, es decir, cada programa tiene garantizada la oportunidad de ejecutarse, y se ejecuta hasta que el sistema operativo da prioridad a otro programa para su ejecución. Así, los programas se ejecutan hasta que permiten voluntariamente que otros programas también lo hagan.

El microprocesador sólo es capaz de hacer una tarea a la vez, pero las realiza en tiempos tan cortos que se escapan a nuestra comprensión, es por eso que en los momentos que no este trabajando para determinada tarea, se dedica a ejecutar otras que se le hayan pedido.

Es fácil ver las ventajas de disponer de multitarea prioritaria. Además de reducir el tiempo muerto (tiempo en el que no puede seguir trabajando en una aplicación

porque un proceso aún no ha finalizado), da la flexibilidad de no tener que cerrar las ventanas de las aplicaciones antes de abrir y trabajar con otras.
LINUX y otros sistemas operativos multitarea prioritaria consiguen el proceso de prioridad supervisando los procesos que esperan para ejecutarse, así como los que se están ejecutando. El sistema programa cada proceso para que dispongan de las mismas oportunidades de acceso al microprocesador. El resultado es que las aplicaciones abiertas parecen estar ejecutándose al mismo tiempo, pero en realidad, hay una demora de billonésimas de segundo entre el momento en que el procesador ejecuta una serie de instrucciones de una aplicación y el momento programado por LINUX para volver a dedicar tiempo a dicho proceso.

1.4.2.2 MULTIUSUARIO

La idea de que varios usuarios pudieran acceder a las aplicaciones o la capacidad de proceso de un único host era una utopía hace relativamente pocos años.
La capacidad de LINUX para asignar el tiempo de microprocesador simultáneamente a varias aplicaciones ha derivado en la posibilidad de ofrecer servicio a diversos usuarios a la vez, ejecutando cada uno de ellos una o más aplicaciones. La característica que más resalta de LINUX es que un grupo de personas puede trabajar con la misma aplicación al mismo tiempo, desde el mismo terminal o desde terminales distintos.

1.4.2.3 MULTIPLATAFORMA

Las plataformas de hardware en las que en un principio se puede utilizar LINUX son 386-, 486, Pentium Pro, Pentium II/III/IV. También existen versiones para su utilización en otras plataformas, como Alpha, ARM, MIPS, PowerPC y SPARC.

1.4.2.4 CONVIVENCIA CON OTROS SISTEMAS OPERATIVOS

Pueden estar juntos pero no funcionar al mismo tiempo. Cada vez que arrancamos un host, podemos elegir cual de ellos se debe cargar, y a partir de este momento sólo podremos utilizar aplicaciones destinadas al Sistema Operativo que estamos ejecutando.

1.4.2.5 SOPORTE EN REDES

LINUX soporta la mayoría de los protocolos comunes de Internet, incluyendo correo electrónico, noticias, gopher, telnet, web, ftp, pop, ntp, nfs, dns, snmp y muchos más. LINUX puede operar como cliente o servidor para todo lo nombrado y ha sido ampliamente usado y probado.

LINUX dispone de los dos principales protocolos de red para sistemas UNIX: TCP/IP y UUCP (Unix to Unix Copy Protocol).

Con LINUX, TCP/IP y una conexión a la red, puede comunicarse con usuarios y hosts por toda Internet mediante correo electrónico, noticias, transferencias de archivos con FTP y mucho más.

La mayoría de las redes TCP/IP usan Ethernet como tipo de red física de transporte. LINUX da soporte a muchas tarjetas de red Ethernet e interfaces para hosts personales, incluyendo adaptadores para hosts portátiles.

Dado que no todo el mundo tiene una conexión Ethernet, LINUX también proporciona el protocolo Internet sobre líneas Seriales SLIP (Serial Line Internet Protocol), el cual permite conectarse a Internet a través de un módem. Si un sistema LINUX dispone de conexión Ethernet y de módem, puede ser configurado como servidor de SLIP para otros usuarios.

El sistema proporciona la interfaz estándar de programación por "sockets", lo que virtualmente permite que cualquier programa que use TCP/IP pueda ser llevado a LINUX. El servidor de modo gráfico de LINUX también soporta TCP/IP, permitiendo ver aplicaciones que están corriendo en otros sistemas sobre su pantalla.

UUCP es un viejo mecanismo usado para transferir archivos, correo electrónico y noticias entre hosts UNIX. Clásicamente los hosts UUCP se conectan entre ellos mediante líneas telefónicas y módem, pero UUCP es capaz de funcionar también sobre una red TCP/IP.

1.4.3 CONCEPTOS BÁSICOS DE LINUX

Los conceptos fundamentales que se utilizan en el sistema operativo LINUX son los siguientes:

1.4.3.1 PROCESO

Es un algoritmo interpretado o compilado que está ejecutándose en un sistema y se encuentra residente en la memoria RAM. También es la representación de cualquier programa en ejecución el cual puede ser auditado, analizado, terminado, etc.

1.4.3.2 SERVICIO

Es un tipo de proceso que está a la escucha, es decir, no está haciendo nada a no ser que sea requerido, en cuyo caso atiende convenientemente la petición. Un servicio suele cargarse de forma permanente en memoria hasta que sea terminado o el Sistema Operativo lo cierre.

1.4.3.3 DEMONIO

Es un tipo especial de servicio que escucha a otros servicios o procesos. El demonio es un “programa que escucha a otro programa”. Un ejemplo claro de esto sería XINETD (extended Internet daemon) en sistemas Unix, que se encarga de escuchar a los procesos o servicios que soliciten conexión de red, y atiende esas peticiones.

1.4.3.4 SHELL O INTERFAZ DE USUARIO [18]

Es un programa encargado de comunicar el entorno de aplicaciones con el kernel o núcleo del sistema. Un shell puede ser tanto una interfaz gráfica como intérprete de comandos.

Un shell interpreta y ejecuta instrucciones que le hemos proporcionado por medio de una línea de comandos (prompt). Es decir, el intérprete de comandos recibe lo que se escribe en la terminal (sinónimo de shell o interfaz) y lo convierte en instrucciones para el sistema operativo.

El prompt es una indicación que muestra el intérprete para anunciar que espera una orden del usuario. Cuando el usuario escribe una orden, el intérprete la ejecuta. En dicha orden, puede haber programas internos o externos: Los programas internos son aquellos que vienen incorporados en el propio intérprete, mientras que los externos son programas separados.

En el mundo LINUX/Unix existen tres grandes familias de Shells como se muestra en la tabla 1.1. Estas se diferencian entre sí básicamente en la sintaxis de sus comandos y en la interacción con el usuario.

Tipo de Shell	Shell estándar	Versiones libres
AT&T Bourne shell	Sh	ash, bash, bash2
Berkeley "C" shell	Csh	tcsh
AT&T Korn shell	Ksh	pdksh, zsh
Otros interpretes	--	esh, gush, nwsh

Tabla 1.1.- Familias de SHELL

Hay muchas interfaces posibles como el sistema X Windows, el cual permite ejecutar comandos usando periféricos como el ratón y el teclado.

1.4.3.5 INODO

Se puede definir como un descriptor de una entrada de un sistema de archivos o archivos. Todo archivo tiene asociado un único inodo. Un inodo también puede ser definido como una clave numérica para el acceso al sistema plano de archivos, donde cada punto es capaz de recibir o entregar información.

1.4.3.6 FILESYSTEM (SISTEMA DE ARCHIVOS)

Una colección de todos los archivos y directorios de un sistema organizados en una jerarquía en forma de árbol.

1.4.3.7 SISTEMA DE DIRECTORIOS [19]

Los sistemas de archivos UNIX se caracterizan generalmente por tener una estructura jerárquica, dar tratamiento consistente a la información de los archivos y a la protección de ellos.

El sistema de LINUX sigue básicamente el mismo de UNIX. Pero se puede encontrar una nueva división del sistema de directorios respecto de los archivos que lo componen como es: Compartible y No Compartible dependiendo si a la información de un host local pueden o no acceder otros hosts, además Estática y No Estática cuando la información puede ser o no alterada por el usuario.

El primer sistema de directorios que se aplicó a LINUX en su momento cumplió con las necesidades básicas, pero se volvió restrictivo, ya que en los archivos, el nombre no podía superar los 14 caracteres y los 64 MB de espacio. Para resolver este problema surgió el primer sistema de archivos especialmente diseñado para LINUX, el ext2 (extended filesystem). En el desarrollo del *sistema extendido (ext2)* surgió un importante cambio en lo que respecta a sistemas de archivos, el *sistema virtual de archivos* (VFS, virtual filesystem) que existe entre el sistema real de archivos y el sistema operativo y sus servicios.

El logro del *sistema virtual de archivos (VFS),* es que a LINUX le permite ***montar*** una serie de sistemas de archivos diferentes y variados; es decir, todos los detalles de los sistemas de archivos son tratados por programas de modo que parezcan lo mismo tanto para el kernel como para las demás aplicaciones que se ejecutan en el sistema, en términos prácticos, es un modo de *unificar* los sistemas de archivos para poder ser manipulados por LINUX de un modo *uniforme*.

El sistema virtual de archivos ha sido implementado de tal forma que sea rápido y eficiente. Otra de sus funciones es analizar y verificar que la información sea almacenada adecuadamente. Una característica particular del sistema virtual de archivos, es que guarda cierta información (caché) en la medida que monta y utiliza los sistemas de archivos, de tal forma de acelerar el procesamiento de la información.

El sistema de archivos real de LINUX, el ext2, funciona a través de bloques de información que va siendo almacenada en el disco duro, pero sin fragmentar la información. En muchos sentidos el sistema de almacenaje por bloques de información es, muchas veces, poco óptimo perdiéndose mucho espacio por la asignación de éstos, pero a pesar de todo esto, es el mejor método hasta el momento.

Cada archivo en el sistema ext2 posee un inodo único y cada inodo posee un único número identificador. Los directorios tan solo son archivos especiales que contienen punteros a los inodos de sus archivos miembros.

1.4.3.7.1 Representación de dispositivos

Se debe conocer como se representa un dispositivo en LINUX. Aquí encontramos el directorio /dev, el cual contiene a los dispositivos del sistema.

En la tabla 1.2 se indica un ejemplo con diferentes dispositivos almacenados en el directorio /dev:

Primera disquetera	/dev/fd0
Segunda disquetera	/dev/fd1
Primer disco duro	/dev/hda
Primer disco duro, partición	/dev/hda1

primaria 1	
Disco duro SCSI (Small Computer System Interface)	/dev/sda

Tabla 1.2.- Directorios de /dev.

1.4.3.7.2 Organización de los directorios

La distribución de los directorios es en forma de árbol (tabla 1.3), el cual comienza en el directorio "/", también conocido como "directorio raíz". Directamente por debajo de “/” hay algunos subdirectorios importantes.

bin	Archivos binarios de comandos esenciales.
boot	Archivos estáticos de arranque.
dev	Archivos de dispositivos (virtuales).
etc	Archivos de configuraciones locales-globales.
home	Directorio donde se contienen las cuentas de usuarios.
lib	Bibliotecas compartidas.
media	Punto de montaje de dispositivos.
proc	Sistema de archivos virtual.
root	Directorio del administrador.
sbin	Archivos binarios esenciales del sistema.
tmp	Archivos temporales.
usr	Segunda jerarquía mayor.
var	Archivos de información (variables).

Tabla 1.3.- Directorios de LINUX.

A continuación se describen cada uno de los diferentes tipos de directorios:

- **BIN Y SBIN**

Los archivos localizados en estos directorios corresponden a archivos *ejecutables*; la diferencia es que dentro del directorio sbin estarán sólo los ejecutables por el administrador.

- **BOOT**

Contiene todos los elementos necesarios para arrancar el sistema. Aquí se encuentra todo lo que se ejecutará antes de que el kernel ejecute /sbin/init.

- **DEV**

Los archivos (especiales) ubicados en este directorio representan los dispositivos a los que LINUX puede acceder y utilizar. Cabe mencionar que estos archivos realmente no existen, sino que son la vía de acceso a los dispositivos.

- **ETC**

Los contenidos de éste directorio son muy importantes para el normal funcionamiento tanto de las aplicaciones como de los demonios. Aquí residen todos los archivos de configuración global ya sea de los demonios, así como las configuraciones globales de aplicaciones utilizables por todos los usuarios. Bajo este directorio se encuentra también el archivo de contraseñas del sistema.

- **HOME**

Bajo este directorio residen todas las cuentas de usuario. Cuentas que serán subdirectorios contenidos de la forma: /home/usuario.

- **LIB**

En este directorio residen todas las bibliotecas compartidas requeridas por los comandos en el sistema raíz.

- **MEDIA**

Este directorio es el punto de montaje para sistemas de archivos montados temporalmente. Este directorio no influye en el normal funcionamiento del sistema y puede contener puntos de montaje para dispositivos removibles y otros.

- **PROC**

Corresponde a un sistema de archivos virtual muy peculiar, pues si bien pueden listarse archivos contenidos en dicho directorio, éstos realmente no existen. Cuando el sistema requiere información de alguno de los contenidos de éste directorio y el sistema virtual de archivos solicita inodos, a medida que los archivos se abren /proc crea dichos archivos con información del kernel del sistema.

- **ROOT**

La ubicación de este directorio es arbitraria, bien puede residir bajo el directorio raíz o dentro del directorio de cuentas de usuario.

- **TMP**

Este directorio contiene los archivos temporales del sistema, es un modo de utilizar memoria física para el procesamiento de algunos datos. La información contenida en /tmp no se debe considerar permanente. Este directorio está asociado al uso de la memoria RAM.

- **USR**

Corresponde a la segunda jerarquía mayor de archivos del sistema. Suele contener información compartible, pero de sólo lectura. Suele contener directorios similares a los ubicados bajo el directorio raíz, tales como /bin, /sbin, /lib, /etc pero que contienen binarios o configuraciones no esenciales para el normal funcionamiento del sistema.

- **VAR**

Contiene archivos de información variable, cuyo ejemplo son los archivos de registro (logs) los cuales cambian constantemente. Otros archivos contenidos en este directorio son los correos entrantes, algunos archivos temporales, etc.

1.4.4 ADMINISTRACIÓN DE USUARIOS Y GRUPOS [18]

1.4.4.1 LA CUENTA ROOT

Los usuarios normales están restringidos comúnmente para que no puedan alterar los archivos de otro usuario en el sistema. Los permisos de los archivos en el sistema están preparados para que los usuarios normales no tengan permitido borrar o modificar archivos en directorios compartidos por todos los usuarios (como son /bin y /usr/bin).

Estas restricciones desaparecen para *root*. El usuario *root* puede leer, modificar o borrar cualquier archivo en el sistema, cambiar permisos y pertenencias en cualquier archivo, y ejecutar programas.

1.4.4.2 USUARIOS

Como se mencionó anteriormente, *root* es usuario especial que se distingue de los demás usuarios en los privilegios que tiene sobre el sistema. Este no tiene ninguna restricción sobre lo que puede hacer. Cuando se instala LINUX por primera vez la única cuenta que debe existir en el sistema es la del *root*. Debido al poder de este usuario, es peligroso utilizarlo habitualmente para tareas cotidianas que no necesiten privilegios especiales, ésta cuenta se debe dejar para las tareas de administración y mantenimiento del sistema.

Para el trabajo cotidiano hay que crear una cuenta personal con privilegios que protejan al sistema de los posibles errores cometidos. Cada persona que utilice el sistema debe tener su propia cuenta.

1.4.4.3 GRUPOS

La utilidad de un grupo de usuarios es la de permitir una administración ordenada de permisos sobre un conjunto de archivos. Cada usuario debe tener al menos un grupo que es el principal, pero podemos agrupar en varios grupos a un mismo usuario. Estos serían grupos secundarios.
Cada usuario pertenece a uno o más grupos. La única importancia real de las relaciones de grupo es la perteneciente a los permisos de archivos, cada archivo tiene un "grupo propietario" y un conjunto de permisos de grupo que define de qué forma pueden acceder al archivo los usuarios del grupo.
Hay varios grupos definidos en el sistema, como pueden ser bin, mail y sys. Los usuarios no deben pertenecer a ninguno de estos grupos; se utilizan para permisos de archivos del sistema. En su lugar, los usuarios deben pertenecer a un grupo individual.

1.4.4.4 PERMISOS

LINUX, como cualquier sistema Unix, es multiusuario, por lo que, los permisos de los archivos están orientados a dicho sistema. Los permisos de cualquier archivo tienen tres partes: permisos del propietario, permisos del grupo y permisos para el resto de usuarios. Así, un archivo pertenece a un determinado propietario y a un determinado grupo y, dependiendo de los permisos que tenga asociado dicho archivo, se podrá tener o no acceso a él.
Los permisos son de lectura (r), escritura (w) y ejecución (x).

1.4.5 REDES DE DATOS EN LINUX

1.4.5.1. Servicios sobre TCP/IP [17]

Los servicios más frecuentes de TCP/IP para el usuario en la actualidad son la conexión remota a otros hosts (Telnet, SSH *Secure Shell*), la utilización de

ficheros remotos (*Network File System* NFS) o su transferencia (*File Transfer Protocol* FTP, *HiperText Transfer Protocol,* HTTP).

Estos servicios se describen a continuación:

- Transferencia de archivos: el protocolo FTP permite obtener/enviar archivos de un host hacia otro. Para ello, el usuario debe tener una cuenta en el host remoto e identificarse a través de su nombre (*login*) y una palabra clave (*password*), o en hosts donde existen un repositorio de información en donde el usuario se conectará como anónimo (*anonymous*) para transferir estos archivos al host local.
- Conexión (*login*) remota: el protocolo de terminal de red (Telnet) permite a un usuario conectarse a un host remotamente. El host local se utiliza como terminal del host remoto y todo es ejecutado sobre éste permaneciendo el host local invisible desde el punto de vista de la sesión. Este servicio en la actualidad se ha reemplazado por el SSH por razones de seguridad. En una conexión remota mediante Telnet, los mensajes circulan tal cual (texto plano), o sea, si alguien “observa” los mensajes en la red, es similar a mirar la pantalla del usuario. SSH codifica la información, mediante la cual, hace que los paquetes en la red sean ilegibles a un nodo extraño.
- Sistemas de archivos en red NFS (*Network File Systems*): permite a un sistema acceder a los archivos sobre un sistema remoto en una forma más integrada que FTP. Los dispositivos de almacenamiento (o parte de ellos) son exportados hacia el sistema que desea acceder y éste los puede “ver” como si fueran dispositivos locales. Este protocolo permite a quien exporta poner las reglas y la formas de acceso, lo que (bien configurado) hace independiente del lugar donde se encuentre la información físicamente.
- Ejecución remota: permite que un usuario ejecute un programa sobre otro host. Existen diferentes maneras de realizar esta ejecución: a través de un comando (*rsh*, ssh, *rexec*) o a través de sistemas con RPC (*Remote Procedure Call*) que permiten a un programa en un host local ejecutar una función de un programa sobre otro host.
- Servidores de nombre (*name servers*): en grandes instalaciones existen un conjunto de datos que necesitan ser centralizados para mejorar su

utilización, por ejemplo, nombre de usuarios, palabras claves, direcciones de red, etc. Todo ello facilita que un usuario disponga de una cuenta para todos los hosts de una organización. El DNS (*Domain Name System*) es otro servicio de nombres pero guarda relación entre el nombre del host y la identificación lógica de este (dirección IP).

- Servidores de terminales gráficas (*network-oriented window systems*): permiten que un host pueda visualizar información gráfica sobre una pantalla que está conectado a otro host. El más común de estos sistemas es X Windows.

1.4.5.2. TCP/IP [17]

TCP/IP son protocolos que permiten la comunicación entre hosts.

IP es el principal protocolo de la capa de red. Es un protocolo que permite la entrega de paquetes (llamados datagramas IP), cuya característica principal es de ser un protocolo no orientado a conexión, debido a que cada uno de los paquetes puede seguir rutas distintas entre el origen y el destino, es no fiable porque los paquetes pueden perderse, dañarse o llegar retrasados.

TCP es un protocolo orientado a conexión, es decir, la comunicación se trata como un flujo de datos (*stream*). Para la operación de TCP es necesaria la sincronización del intercambio de señales de tres vías, y significa que antes de comunicarse entre dos dispositivos, deben llevar a cabo el proceso de sincronización para establecer una conexión virtual para cada sesión. Este proceso siempre lo inicia el cliente y para lo cual usa un puerto conocido del servicio que desea contactar.

El proceso ocurre así: primero el cliente inicia la sincronización enviando un paquete SYN para iniciar la conexión, en el siguiente paso el otro dispositivo recibe el paquete y responde con un acuse de recibo ACK, como último paso el dispositivo que inició la conversación responde con un ACK indicando que recibió el ACK enviado por el otro dispositivo y finaliza el proceso de conexión para esta sesión.

El protocolo UDP (*User Datagram Protocol*), es un protocolo no orientado a conexión (el host destino no debe necesariamente estar escuchando cuando un host establece comunicación con él). No proporciona mecanismos de control de flujo ni recuperación de errores y tiene la ventaja de que ejerce una menor sobrecarga a la red que las conexiones TCP, pero la comunicación no es fiable.
Existe otro protocolo alternativo llamado ICMP (*Internet Control Message Protocol*). ICMP se utiliza para mensajes de error o control en la red. Por ejemplo, si uno intenta conectarse a un *host*, el host local puede recibir un mensaje ICMP indicando *"host unreachable*". ICMP también puede ser utilizado para extraer información sobre una red. ICMP es similar a UDP, ya que maneja datagramas, pero es más simple que UDP, ya que no posee identificación de puertos (los puertos son buzones donde se depositan los paquetes de datos y desde donde las aplicaciones servidoras leen dichos paquetes) en el encabezamiento del mensaje.
Existen otros protocolos dentro de TCP/IP como ARP(*Address Resolution Protocol* que utiliza mensajes de brodcast o difusión para determinar la dirección MAC correspondiente a una dirección de red particular (dirección IP). Y RARP *(Reverse Address Resolution Protocol)* que utiliza mensajes de tipo *broadcast* para determinar la dirección de red asociada con una dirección de hardware en particular.

1.4.5.3 Configuración de la interfaz de Red (NIC, Network Interface Card) [17]

Los dispositivos de red se configuran automáticamente cuando se inicializa el hardware correspondiente. Por ejemplo, el controlador de Ethernet configura las interfaces *eth*[0..*n*] secuencialmente cuando se localiza el hardware correspondiente.
A partir de este momento, se puede configurar la interfaz de red, lo cual implica dos pasos: asignar la dirección de red al dispositivo e inicializar los parámetros de red al sistema. El comando utilizado para ello es el *ifconfig* (*interface configure*). Un ejemplo será:

ifconfig eth0 192.168.110.23 netmask 255.255.255.0 up

Lo cual indica configurar el dispositivo *eth0* con dirección IP 192.168.110.23 y máscara de red 255.255.255.0. El *up* indica que la interfaz pasará al estado activo (para desactivarla debería ejecutarse *ifconfig eth0 down*). El comando asume que si algunos valores no se indican, son configurados por defecto. En este caso, el *kernel* configurará este host como Tipo-C con la dirección IP 192.168.110.23 y la dirección de *broadcast* con 192.168.110.255.

Por ejemplo:

ifconfig eth0 192.168.110.23 netmask 255.255.255.0 up

1.4.5.4 Configuración del Sistema de Resolución de Nombres [17]

El siguiente paso es configurar el sistema de resolución de nombres que convierte nombres tales como *linuxero.com* en 192.168.110.23. El archivo /etc/resolv.conf es el utilizado para realizar esa tarea. Su formato es muy simple (una línea de texto por sentencia). Existen tres palabras clave para tal fin: *domain* (dominio local), *search* (lista de dominios alternativos) y *name server* (la dirección IP del DNS (*Domain Name Server)*.

Un archivo importante es el /etc/*host*.conf, que permite configurar el comportamiento del sistema de resolución de nombres. Su importancia reside en indicar dónde se resuelve primero la dirección o el nombre de un nodo. Esta consulta puede ser realizada al servidor DNS o a tablas locales dentro del host actual (/etc/hosts).

Ésta configuración indica que primero se verifique el /etc/hosts antes de solicitar una petición al DNS y también indica que retorne todas las direcciones válidas que se encuentren en /etc/hosts. Por lo cual, el archivo /etc/hosts es donde se colocan las direcciones locales o también sirve para acceder a nodos sin tener que consultar al DNS.

La consulta es mucho más rápida, pero tiene la desventaja de que si el nodo cambia, la dirección será incorrecta. En un sistema correctamente configurado, sólo deberán aparecer el nodo local y una entrada para la interfaz *loopback*. En referencia a la interfaz *loopback,* éste es un tipo especial de interfaz que permite realizar al nodo conexiones consigo misma (por ejemplo, para verificar que el

subsistema de red funciona sin acceder a la red). Por defecto, la dirección IP 127.0.0.1 ha sido asignada específicamente al *loopback* (un comando telnet 127.0.0.1 conectará con el mismo host).
Para el nombre de un host pueden utilizarse alias, que significa que ese host puede llamarse de diferentes maneras para la misma dirección IP.
El orden de consulta de las bases de datos para obtener el IP del nodo o su nombre será primero el servicio de DNS (que utilizará el archivo /etc/resolv.conf para determinar la IP del nodo DNS) y en caso de que no pueda obtenerlo, utilizará el de las bases de datos local (/etc/*hosts*).

1.4.5.5 SNMP EN LINUX [20]

El paquete Net-SNMP, conocido antes como UCD-SNMP, es un conjunto de herramientas referente para SNMP. Incluye un agente extensible, un demonio del SNMP, las herramientas para solicitar o para fijar la información de agentes SNMP, las herramientas para generar y para manejar desvíos de SNMP, una versión del comando ' netstat ' de UNIX usando SNMP, y un explorador de MIB.
Tiene soporte para SNMPv.1, SNMPv.2 y SNMPv.3.
La distribución contiene algunas herramientas de administración, que permiten desde la línea de comandos, enviar peticiones a dispositivos que ejecuten agentes SNMP. También contiene un programa agente SNMP, diseñado para ejecutarse sobre LINUX, que ofrece a gestores ejecutándose en la red (o en el propio sistema), información sobre el estado de los interfaces, tablas de encaminamiento, instante de inicio (uptime), información de contacto, etc.

1.4.5.5.1 Agentes SNMP

En la mayoría de los sistemas LINUX, se incluye un agente de SNMP. En versiones previas la librería *net-snmp* se denomina *ucd-snmp*. Esta librería ha sido muy actualizada y desarrollada e incluye las herramientas tradicionales de SNMP.

Los agentes de SNMP son modificables, tanto a través del propio código, es decir, con la API (interfaz para la programación de aplicaciones) proporcionada, como a través de comandos definidos en la configuración.
Como este es un software de agentes tan extendido, es conveniente estudiar su instalación y configuración, así como en las herramientas que proporciona.

1.4.5.5.2 Instalación de net-snmp

Una vez descargado y descomprimido en un directorio el código fuente del paquete, se puede proceder a compilarlo ejecutando los siguientes comandos desde un terminal:
$ configure
$ make all
Luego de esto quedará compilado y preparado para instalar. Esta librería no depende de otras, es auto contenida, lo que facilita su compilación, lo cual se podrá realizar con el comando *make install* desde el shell. A continuación se debe configurar el archivo */etc/snmp/snmpd.conf*
Las versiones actuales de LINUX incorporan ya el paquete de *net-snmp,* de forma que su instalación es mucho más sencilla y su configuración es rápida.
En la versión de LINUX se incluyen dos agentes. El primero *snmpd* es un agente que permanece escuchando en el puerto 161 (UDP), esperando recibir peticiones, cuando le llega una solicitud la procesa y devuelve la información. El segundo, *snmptrapd* se trata de un agente que procesa las alertas de otros agentes. Para ello permanece escuchando en el puerto 162 (UDP), cuando recibe una alerta por este puerto procede a guardarla en el registro (syslog). Sin embargo, también puede ser configurado para utilizar programas externos en el tratamiento de las alertas.
Los agentes de Net-snmp incluyen una serie de extensiones para poder obtener información específica del sistema como:

- Información general del sistema.
- Conexiones tcp/udp/ip/snmp abiertas y su estado.
- Discos Duros.

- Procesos y carga del procesador.

1.4.5.5.3 Configuración de los agentes SNMP

Finalizada la instalación de los agentes sólo resta adaptar el paquete a las necesidades del host. La librería incluye una buena documentación que describe el formato de los archivos de configuración.
En la página del manual *snmpd_config* se describe el funcionamiento general de los archivos de configuración. Si el agente no queda instalado con un archivo de ejemplo de configuración será necesario copiar o crear uno en */etc/snmp/snmpd.conf*.
Las primeras definiciones en el archivo de configuración definen las limitaciones para el acceso al agente desde cualquier servidor. Uno de los problemas más comunes es no ser capaz de acceder al agente, por que estas restricciones son muy fuertes o no se han definido correctamente. El funcionamiento es un tanto complejo, pero esto se debe a que el agente tiene soporte para la autenticación en SNMPv.1, en SNMPv.2c (con comunidades) y en SNMPv.3 (a través de usuarios y grupos).
Lo primero que se debe definir es una relación entre comunidades y modelos de seguridad en el agente SNMP, luego se define una relación entre modelos de seguridad y grupos, se definen zonas del árbol de la MIB y, finalmente, se indica el acceso permitido de los grupos a las zonas.
Se está incluyendo todos los accesos como comunidad "public" desde cualquier lugar al grupo MyROGroup, mientras que los accesos como comunidad "private" desde el servidor local se vinculan al grupo MyRWGroup.

1.4.5.5.4 Herramientas de Administración SNMP

Las herramientas de administración SNMP incluidas dentro de *net-snmp* son:

- *snmpstatus* que permite acceder al estado del agente.
- *snmpwalk* que permite *recorrer* la MIB del agente y sus variables.

- *snmpget* y snmpset que permiten, respectivamente, consultar y fijar atributos de SNMP.
- *snmptranslate* permite traducir de un identificador de objeto (OID) de la MIB a una cadena de caracteres representativa de éste.
- *snmpdelta*, establece un proceso de monitoreo sobre una o más variables del agente, de forma que recupera el valor de estas variables en períodos de tiempo definidos.
- *snmptest* es una herramienta de prueba del agente, al conectarse permite, a través de la interfaz de línea de comandos, recuperar cualquier variable que este contenga. Indica los métodos de comunicación usados contra el agente, y si fuera necesaria su depuración.
- *snmpnetstat*, es un comando atípico en las distribuciones de SNMP, ya que es particular de la versión net-snmp. Nos permite obtener un listado de los canales de comunicación abiertos en un host, al igual que netstat, pero utilizando un agente SNMP para recuperar la información.

BIBLIOGRAFÍA CAPÍTULO 1

[1] it.aut.uah.es.
Gestión De Redes (2004/2005)
http://it.aut.uah.es/mar/gestion/tema2.pdf

[2] info-ab.uclm.es.
Mantenimiento y monitorización de redes TCP/IP.
www.info-ab.uclm.es/asignaturas/42524/teoria/ar2Tema7x2.pdf

[3] det.uvigo.es.
Gestión Y Planificación De Redes Con Sistemas Inteligentes
www.det.uvigo.es/~mramos/gprsi/gprsi2.pdf

[4] linuxdata.com.ar.
Introducción a la Administración de una Red Local basada en Internet
http://www.linuxdata.com.ar/index.php?idmanual=tutorialadmionredss.html&manuale=1

[5] it.aut.uah.es
Gestión y Administración de Redes
http://it.aut.uah.es/alarcos/docente/garii/5monitorizacion&control.pdf

[6] dis.eafit.edu.co.
Especialización En Teleinformática
http://dis.eafit.edu.co/cursos/st724/2005-mauricio/Mod-ST724.pdf

[7] tvyvideo.com
Monitorear la red es monitorear la operación
http://www.tvyvideo.com/pragma/documenta/tv/secciones/TV/ES/MAIN/IN/INFORMES_ESPECIALES/doc_13446_HTML.html?idDocumento=13446

[8] ditec.um.es
Gestión de red
http://ditec.um.es/laso/docs/tut-tcpip/3376c414.html

[9] rincondelvago.com
ADMINISTRACIÓN DE REDES
http://html.rincondelvago.com/administracion-de-redes.html

[10] arcesio.net
Structure of Management Information (SMI) para SNMPv1
http://www.arcesio.net/snmp/asn1.html

[11] "Transmisión de Datos y Redes de Comunicaciones", Behrouz A. Forouzan Seg. Edición.

[12] det.uvigo.es
Gestión Y Planificación De Redes Con Sistemas Inteligentes
http://www.det.uvigo.es/~mramos/gprsi/gprsi3.pdf.

[13] Ford Meilee, Lew H. Kin, "Tecnología de Interconectividad de Redes", Prentice Hall 1998. Capitulo 6.

[14] det.uvigo.es
Gestión Y Planificación De Redes Con Sistemas Inteligentes
http://www.det.uvigo.es/~mramos/gprsi/ampl-gprsi3.pdf

[15] info-ab.uclm.es
Protocolos de Mantenimiento
Anexo: SNMPv3
http://www.info-ab.uclm.es/asignaturas/42621/transpas/dmrTema4-SNMPv3.pdf

[16] monografías.com
Redes bajo LINUX
http://www.monografias.com/trabajos/redeslinux/redeslinux.shtml

[17] uoc.edu
Administración Avanzada de GNU/Linux
www.uoc.edu/masters/softwarelibre/esp/materials/Admin_GNULinux.pdf

[18] Domine Linux paso a paso, Juan Cherre, 1 ed, Lima – Perú, editorial macro. Agosto 2001.

[19] grancisco.guidi.com
http://francisco.guidi.com/download.php?mod=d&tipo=html&id=2

[20] diariolinux.com
Gestión SNMP con Linux
http://www.diariolinux.com/articulos/printable.php?f=5

CAPÍTULO 2

DESARROLLO DE LA APLICACIÓN GRÁFICA DEL MONITOREO Y ADMINISTRACIÓN DE RED

2.1 DESCRIPCIÓN DE LOS COMANDOS DE LINUX PARA LA ADMINISTRACIÓN Y MONITOREO DE REDES.

2.1.1 ESTRUCTURA DE LOS COMANDOS

Un comando es una instrucción o conjunto de instrucciones que se da a un programa para que realice una acción determinada. Los comandos en esencia son funciones, a los que añadiendo ciertos argumentos y opciones, se puede desencadenar una acción en el sistema.
Un comando tiene la siguiente estructura:

<nombre del comando> [lista de opciones] [lista de argumentos] <retorno>

2.1.2 LISTADO DE LOS COMANDOS UTILIZADOS EN EL PROGRAMA [1]

El programa es un conjunto de comandos o instrucciones para realizar una determinada tarea o trabajo.
La clasificación de los comandos que se utilizan para desarrollar programas en LINUX son:

- Generales.
- Conectividad.
- Administración y Monitoreo.

2.1.2.1 Comandos Generales

A continuación se tiene un listado de los comandos generales que se utilizaron para el desarrollo de la aplicación:

- **UNAME**

  Permite obtener información sobre el tipo de sistema UNIX en el que se está trabajando, así también para determinar la versión de kernel, etc.

- **PS**

  Muestra la información sobre los procesos que se encuentran en ejecución.

- **KILL**

  Es utilizado para enviar señales a los procesos en LINUX, tales como terminar el proceso instantáneamente o dándole tiempo a un proceso para que pueda terminar.

- **HOSTNAME**

  Permite obtener el nombre que tiene configurado el host local o el que se lo puede asignar.

2.1.2.2 Comandos de Conectividad

A continuación se tiene un listado de los comandos utilizados para verificación de conectividad en el desarrollo de la aplicación:

- **PING**

  Es un comando que permite enviar y recibir peticiones de eco para determinar si un destino está listo para recibir información.

- **TRACEROUTE**

El comando traceroute es similar al comando ping, salvo que en lugar de probar la conectividad de extremo a extremo, traceroute verifica cada paso en el proceso, es decir, muestra el camino que se necesita para llegar a otro host.

- **NETSTAT**

Permite listar las conexiones de red actualmente activas, descargar el sistema de tablas de enrutamiento y estadísticas de la presente interfaz de red.

2.1.2.3 Comandos de Administración y Monitoreo

A continuación se indica un listado de los comandos para la Administración y Monitoreo de red utilizados en el desarrollo de la aplicación:

- **ADDUSER**

Este comando es utilizado para añadir un usuario al sistema.

- **USERDEL**

El comando USERDEL permite eliminar un usuario del sistema.

- **IFCONFIG**

Permite configurar y ver el estado de las interfaces de red en el host local. Es ayudado por los comandos **ifup,** que habilita la interfaz especificada e **ifdown**, que deshabilita la interfaz especificada. [2]

- **IP**

Este comando junto con otras opciones y argumentos muestra y manipula la tabla de ruteo. Ayuda en la gestión de paquetes, tráfico IP y muestra información relacionada con las interfaces del host local.

- **TCPDUMP**

Es un comando al que añadiéndole determinadas opciones, puede capturar paquetes que cursan a través de una determinada interfaz de un host en la red, para luego interpretarlos y generar resultados. Comprende o entiende todos los protocolos básicos de Internet, y puede ser usado para salvar o guardar información de los paquetes para su posterior inspección.

- **SNMPWALK**

El comando SNMPWALK sirve para recorrer un árbol MIB del agente usando la petición getnext-request del protocolo SNMP. Con este comando se puede obtener determinada información de la MIB a través del nombre de la rama o mediante número OID.

- **SAMBA** [3]

Samba es una aplicación, que brinda a los usuarios LINUX un gran número de posibilidades a la hora de interactuar con equipos Windows, ya que trabaja con el protocolo SMB(Server Message Block), el cual le permite a LINUX actuar como servidor y ofrecer servicios tales como:

  - Compartir uno o más sistemas de archivos.
  - Compartir impresoras, instalados tanto en el servidor como en los clientes.

Samba incluye varias herramientas de las cuales, para el desarrollo de la Aplicación se utilizó:

  - **nmblookup,** el cual es un comando que se usa para consultar nombres NetBIOS sobre TCP/IP a partir de direcciones IP o viceversa.

- **NMAP**

NMAP es un comando que se utiliza, para permitir el escaneo de redes y determinar los dispositivos que se encuentran activos, además de los servicios que estos ofrecen. Este comando proporciona también, la opción de detección remota del sistema operativo.

- **TC**

El comando TC permite mostrar y manipular las opciones de control de tráfico en determinada interfaz, además se puede implementar control de tráfico por clases.

- **HTB** (Hierarchical Token Bucket)

El comando HTB permite simular un enlace físico en varios enlaces de menor velocidad, además permite enviar diferentes tipos de tráfico hacia los distintos enlaces simulados, para lo cual, utiliza un algoritmo que divide el ancho de banda de manera que se puede especificar un ancho de banda mínimo y un máximo tanto para el enlace físico, como para los enlaces simulados. [4]

2.2 DESARROLLO DE LA APLICACIÓN GRÁFICA EN BOURNE SHELL (SH)

Bourne Shell (SH) es el shell o intérprete de comandos de LINUX, que hay en los sistemas operativos GNU. SH fue desarrollado de acuerdo al estándar “IEEE POSIX P1003.2/ISO 9945.2 Shell and Tools Standard”. Ofrece funcionalidades para programación, control de procesos y manejo de archivos.

Las características ofrecidas por SH son:

- Línea de comandos de edición.
- Ilimitado tamaño para historial de comandos.

- Ejecución síncrona de órdenes (una tras otra) o asíncrona (en paralelo).
- Distintos tipos de redirecciones de entradas y salidas para el control y filtrado de la información.
- Control del entorno de los procesos.
- Un lenguaje de programación de alto nivel, que incluye distinto tipos de variables, operadores, funciones, etc.

2.2.1 DESCRIPCIÓN DE LA APLICACIÓN GRÁFICA DESARROLLADA

La aplicación a desarrollarse contiene tres componentes, los cuales serán diseñados de la siguiente manera:

a) Monitoreo de dispositivos: Se realiza una búsqueda de los dispositivos que se encuentran activos en la red mediante el comando nmap. Luego, con el comando nmblookup, se obtiene el nombre de los hosts pertenecientes al entorno Windows, la cual corresponde a información del protocolo NetBIOS. Para obtener el nombre de los hosts dentro del entorno LINUX y para el nombre y sistema operativo de los dispositivos administrables, se utiliza el comando snmpwalk.
b) Monitoreo de tráfico: Se realiza la captura de paquetes por tipo de protocolo, a través de las opciones que ofrece el comando tcpdump.
c) Administración de tráfico: En base a los comandos tc y htb, se crean clases y filtros de acuerdo a direcciones IP y protocolos.

Luego, en base a la aplicación gráfica Qt/Designer, se muestra en pantalla y con ayudas visuales los resultados obtenidos, en cada uno de los componentes del programa.

2.2.2 LECTURA DE SCRIPTS PARA SH [5]

Un script para SH es un archivo tipo texto, cuyas líneas tienen comandos que son ejecutados (interpretados) por SH. Para que el intérprete ejecute las líneas de comandos de un script, se puede realizar las siguientes tareas:

- Ejecutar /bin/sh seguido del nombre del archivo (o redireccionando la entrada estándar para llamar al archivo).
- Emplear el caracter "./ "y el nombre del archivo a ejecutar.
- Agregar en la primera línea del archivo la cadena "#!/bin/sh", dar permiso de ejecución al archivo y teclear el nombre del archivo desde el intérprete de comandos.

2.2.3 EJECUCIÓN DE UN COMANDO EN SH [5]

Para determinar cual es la orden o acción, SH realiza las siguientes acciones:

1. Identifica la orden, los parámetros y eventualmente las variables de ambiente que se den junto con la orden.
2. Si la orden va precedida de una ruta (nombre completo de un archivo) y el archivo existe y es ejecutable, SH lo trata como un programa y lo carga a memoria para ejecutarlo. Si la ruta no conduce a un archivo ejecutable, SH presenta un mensaje de error.
3. Si la orden no esta precedida de una ruta determina, es decir, si se trata de un comando interno de SH (como ps o kill) y en caso de que sea, realiza la acción correspondiente.
4. Si la orden no está precedida de una ruta y no es un comando interno busca en varios directorios un archivo ejecutable con el nombre dado y si lo encuentra en alguno lo carga a memoria para ejecutarlo (el orden y los directorios donde busca se especifican en la variable de ambiente *PATH*).
5. En otro caso SH presenta un mensaje de error.

En caso que la orden corresponda a un archivo ejecutable (mediante la ruta completa o porque existe un archivo en un directorio de PATH), SH determinará como ejecutarlo, de dos maneras:

1. Si el script comienza con la cadena "#!" seguida del nombre del lenguaje de programación, SH emplea el programa que utiliza ese determinado

lenguaje, como intérprete del archivo.

2. Si el script no comienza con “ #! ”, lo interpreta con el programa SH. Es decir, lo trata como un script para el intérprete de comandos.

2.3 DESARROLLO DE LA APLICACIÓN GRÁFICA EN QT / DESIGNER

2.3.1 QT/DESIGNER [6]

Qt/Designer es una herramienta muy potente que permite diseñar de una forma muy sencilla y rápida ventanas de diálogo con las librerías Qt. Esta herramienta es una aplicación mediante la cual se puede realizar el diseño de aplicaciones de Interfaz Gráfica de Usuario GUI (Graphic User Interface) muy intuitiva.
Para la Aplicación a desarrollarse se utiliza las siguientes versiones de librerías:

- qt-designer.3.3.3
- qt 3.3.3
- qt-devel.3.3.3
- qtlibs.3.3.3
- qtlibs-devel 3.3.3

2.3.2 LIBRERÍA QT

Qt son un conjunto de librerías multi-plataforma para el desarrollo de la estructura de aplicaciones GUI, escritas en código C++.

2.3.2.1 Características de la Librería Qt

- Qt es una librería para la creación de interfaces gráficas. Se distribuye bajo una licencia libre, lo que nos permite incorporar éstas librerías en aplicaciones open-source.
- El lenguaje de programación que emplea la librería Qt es C++, aunque han aparecido adaptaciones a otros lenguajes como Python o Perl.
- Qt es una librería que trabaja en base a widgets (objetos), Signals(Señales) y Slots(Eventos). Las señales y eventos son el mecanismo para que los objetos puedan comunicarse entre ellos.

2.3.3 Componentes de Qt/Designer

A continuación se indican los pasos necesarios para crear un proyecto en Qt/Designer:

- Para crear una interfaz con QT/Designer basta con seleccionar el menú Archivo -> Nuevo... y allí se elige crear un archivo de Qt/Designer con extensión “ .ui ”.

La figura 2.1 muestra la pantalla inicial del Qt/Designer.

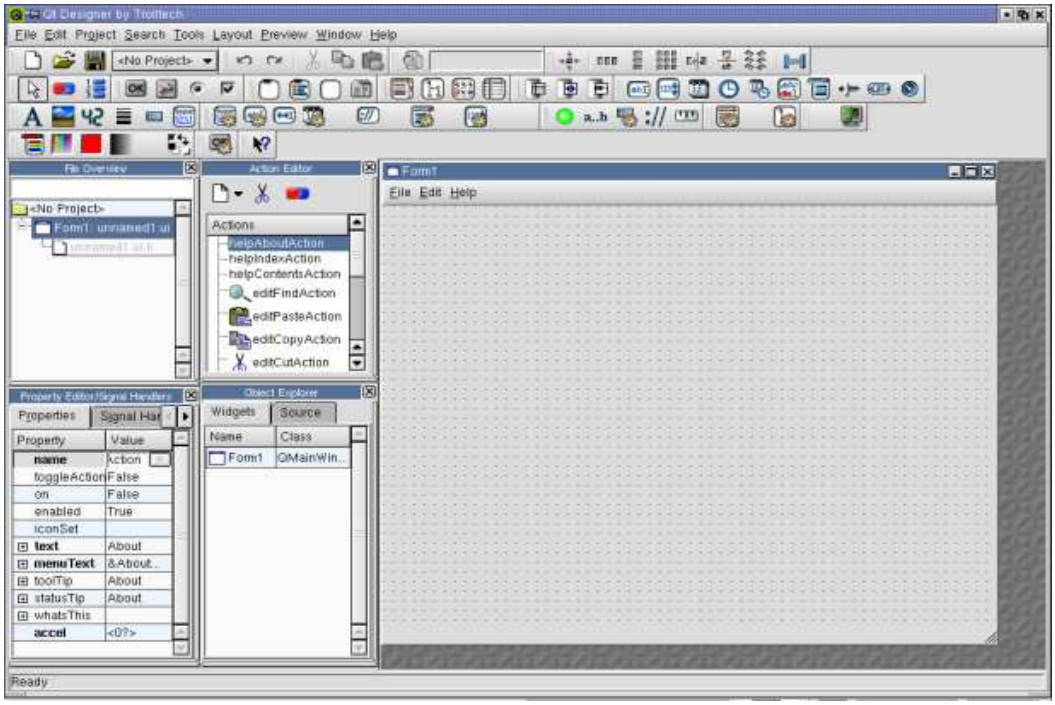


Figura 2.1.- Pantalla inicial de Qt Designer.

- Una vez que se eligió crear un nuevo archivo, aparece una ventana como la que se indica en la figura 2.2.

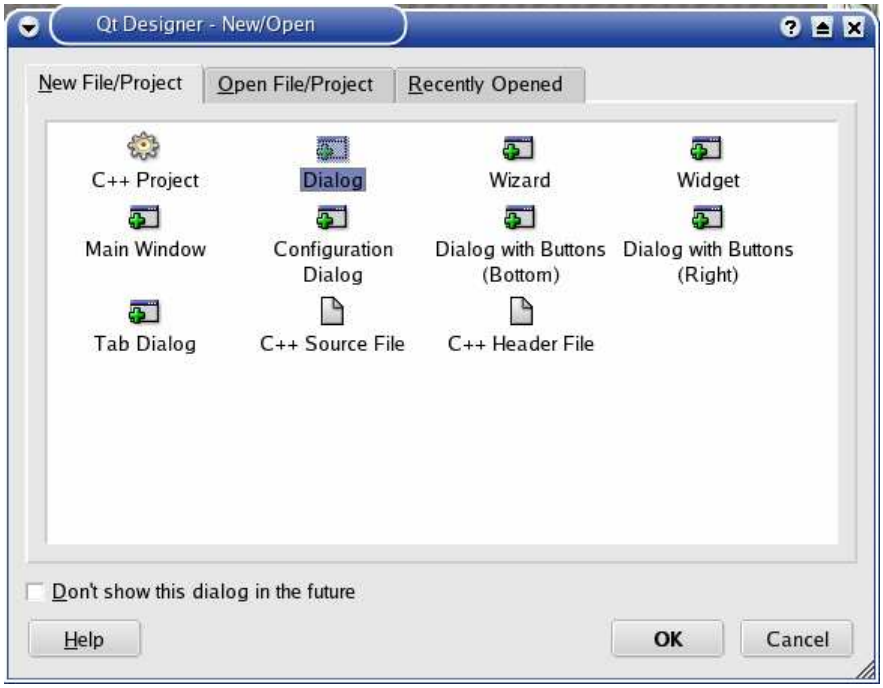


Figura 2.2.- Opciones para creación de un nuevo proyecto en Qt Designer.

De la figura 2.2 se pueden elegir las siguientes opciones:

- Dialog: Es la opción más simple, crea una ventana que incluye únicamente los botones de minimizar, maximizar y cerrar.
- Wizard: Permite realizar aplicaciones paginadas, además de esto, incluye los botones de cancelar, anterior (back), siguiente (next) o finalizar (finish).
- Dialog with buttons: Al igual que las anteriores, ésta también se deriva de la opción Dialog e incluye los botones de ayuda, aceptar y cancelar.

- Una vez seleccionado una de las opciones para la creación de un nuevo proyecto, se dispone de una paleta de widgets muy completa, tal como se muestra en la figura 2.3; que incluyen los widgets más comunes de las librerías Qt. El funcionamiento es de estilo “selecciona y dibuja”, es decir, basta con seleccionar un tipo de widget en la paleta y luego al ponernos sobre el formulario, se dibuja con la geometría que solicitamos.

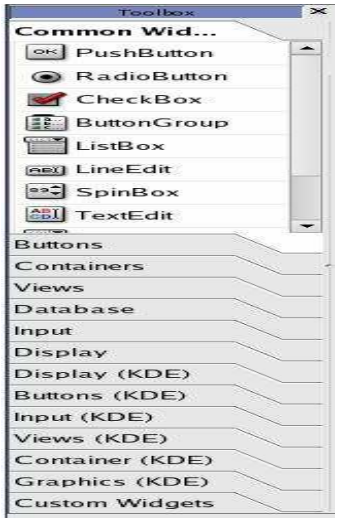


Figura 2.3.- Paleta de Widgets presentes en Qt Designer.

- En la figura 2.4, se observa el panel de las propiedades de un widget cualquiera. Los valores de cada propiedad se pueden editar fácilmente.

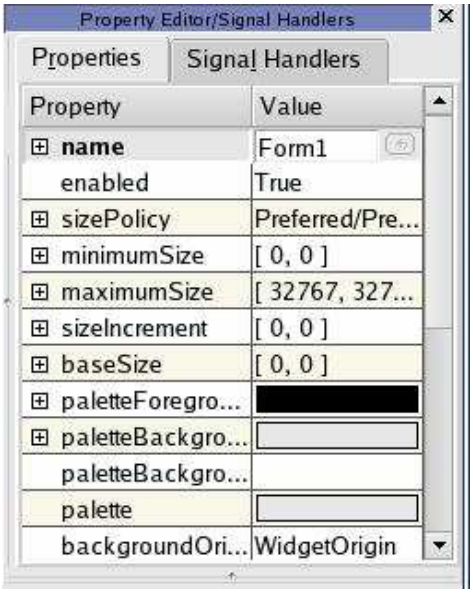


Figura 2.4.- Panel de Propiedades de Qt Designer

- A medida que se guardan los cambios que se van realizando a la interfaz, se puede observar que éstos modifican el contenido del fichero ".ui ". La descripción de la interfaz se guarda en formato XML.

2.3.3.1 Objetos de Qt/Designer

A continuación se describen las opciones de widgets (Figura 2.3), que se disponen para el diseño de interfaces en Qt/Designer.

- **Buttons**: Opción que permite añadir diversas clases de botones, (figura 2.5). Los botones que se pueden encontrar son pushbutton, toolbutton, radiobutton y checkbutton, éstos difieren en su forma, pero básicamente pueden realizar las mismas funciones.

Figura 2.5.- Ejemplos de Botones en Qt/Designer.

- **Containers:** Se define como contenedor a lo que se puede etiquetar con un titulo y además puede contener texto, imágenes e incluso otros objetos. Igualmente existen varios tipos de contenedores como: groupbox, frame, etc. Ejemplos de contenedores se muestran en la figura 2.6.

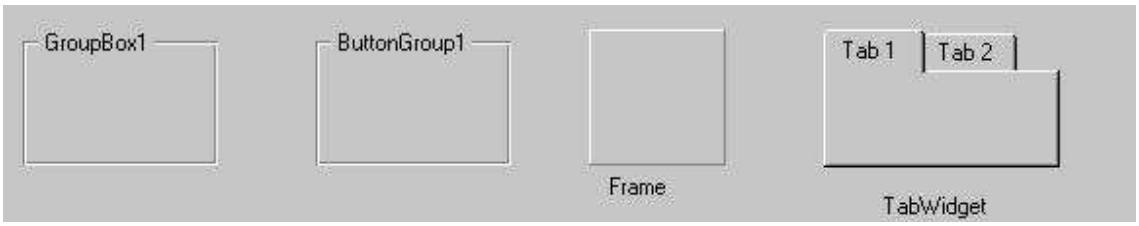


Figura 2.6.- Ejemplos de Contenedores de Qt/Designer

- **Inputs**: Esta herramienta incluye opciones como: LineEdit (líneas de edición), las cuales pueden servir para recibir datos que se ingresan por teclado, MultiLineEdit, combobox (listas desplegables) y otros como slider, spinbox o dial. La figura 2.7 muestra las clases de Inputs que se dispone en QT/Designer.

Figura 2.7.- Ejemplos de Inputs de Qt/Designer

- **Displays**: En ésta herramienta se incluyen opciones como: textlabel (etiquetas de texto), pixmapLabel (imágenes), progressbar (barras de progreso) además de LCDnumber, Line, etc. Algunas de éstas opciones se muestran en la figura 2.8

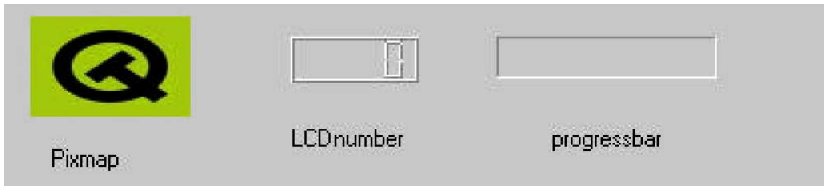


Figura 2.8.- Ejemplos de Displays en Qt/Designer

- **Views**: Son objetos que permiten desplegar información en forma de tablas, árbol de directorios, listas, etc. Dentro de ésta herramienta se incluyen opciones tales como se muestra en la figura 2.9.

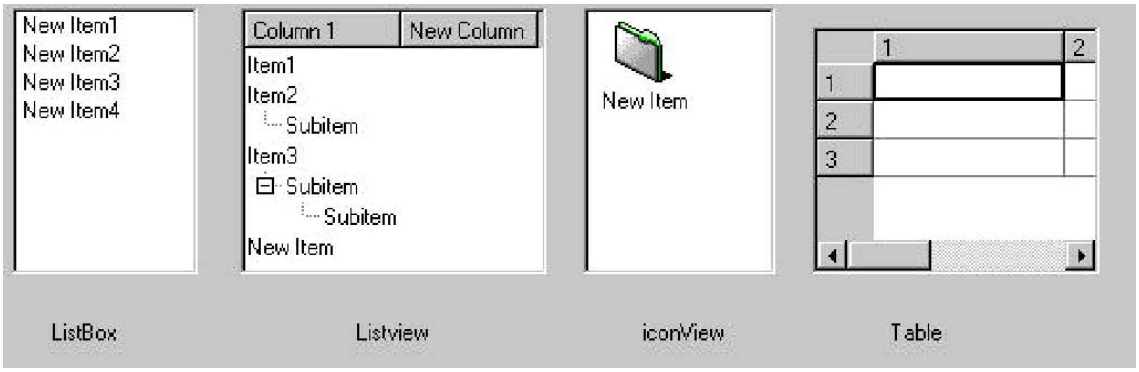


Figura 2.9.- Ejemplo de Views en Qt/Designer.

Todas estas herramientas se las puede encontrar dentro del menú *Tools*, aunque también se puede acceder a ellas, a través de sus respectivos iconos presentes en la barra de herramientas.

2.3.3.2 Slots y Signal en Qt/Designer

Los slots y signal son mecanismos de comunicación entre objetos, ésta es la principal característica de la librería Qt y es lo que hace distintas las librerías Qt del resto de herramientas para la elaboración de GUIs. Es un mecanismo de comunicación seguro, flexible y además implementado en C++. En la programación de GUIs se busca que los cambios producidos en un objeto sean comunicados a otros, por ejemplo, cuando hacemos click en un botón para que se

cierre una ventana, lo que en realidad se hace, es posibilitar la comunicación entre los dos objetos.

2.3.3.3 Compilación en Qt/Designer [6]

Para entender el proceso de compilación en Qt/Designer se explica con el siguiente ejemplo, el cual se muestra en la figura 2.10.

- Primero se debe crear el archivo “final.pro“ (project file), para que éste se encargue de la generación de archivos de compilación.
- Dentro del proyecto final.pro, se deben crear los archivos “ejemplo.ui.h”, “ejemplo.ui”, y “main.cpp”.
- La generación del archivo ejecutable del proyecto, se obtiene a través de la secuencia de comandos qmake y make, donde qmake, es una función que utiliza la información guardada en los archivos del proyecto para determinar qué debería ir en los makefiles (archivos generados y necesarios para el uso del comando make); además, make, determina automáticamente que partes de un programa necesitan ser (re)compiladas y envía los comandos que se describen a continuación:
    - El compilador de interfaz de usuario **uic** (User Interface Compiler) decodifica la información que está en formato XML y la pasa a C++, generando así los archivos “ejemplo.cpp” y “ejemplo.h”.
    - La herramienta **moc** (meta object compiler) sirve para obtener archivos de extensión “moc_ejemplo.cpp” que se encargan de implementar el mecanismo de slots/signal.
    - Finalmente, **g++** que es el compilador de C++ de LINUX genera el archivo ejecutable.

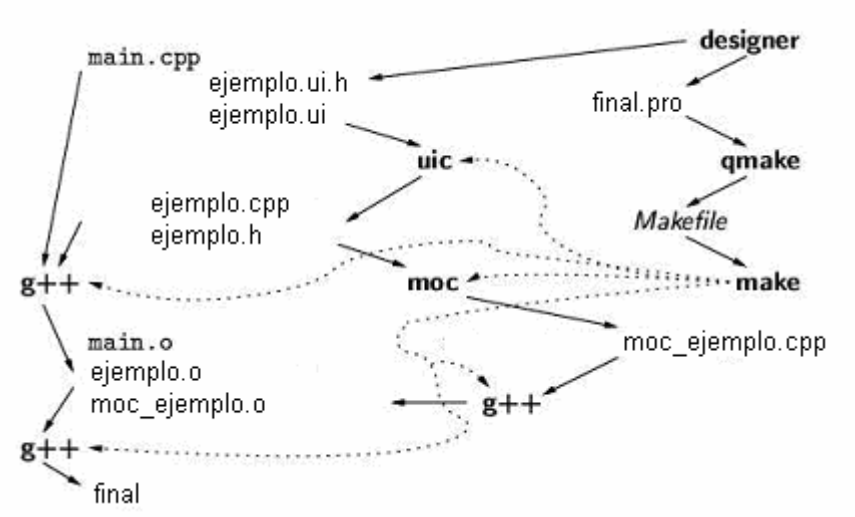


Figura 2.10.- Generación del archivo ejecutable en Qt/Designer.

2.3.4 AYUDAS PARA LA APLICACIÓN GRÁFICA

2.3.4.1 Ayuda en SH

La opción de ayuda para SH esta disponible en forma de texto plano, la misma que viene incluida en el sistema operativo LINUX.
Se permite acceder a esta ayuda de las siguientes maneras:

- A través del comando man, el cual es un paginador de manuales de comandos de LINUX y nos permite obtener ayuda detallada de los mismos.
- A través del navegador se puede acceder a los mismos manuales pero en formato html.

2.3.4.2 Ayuda en Qt/Designer

En el programa Qt/Designer se puede acceder a las ayudas de varias maneras, entre las que destacamos:

- ***A través de un libro:*** Cuando se selecciona contents (contenidos) en el menú de ayuda de Qt/Designer, la aplicación muestra la documentación organizada de manera jerárquica en libros y capítulos.
- ***Ayuda con Shift+F1:*** este simplemente desplaza el cursor a un nombre de una herramienta se muestra una corta ayuda sobre la función que esta desempeña.
- ***Ayuda con F1:*** Este nos enlaza al manual de contenidos con solo presionar esta tecla.

2.4 REQUERIMIENTOS DE LA APLICACIÓN GRÁFICA DE MONITOREO Y ADMINISTRACIÓN DE RED

2.4.1 REQUERIMIENTOS GENERALES

El objetivo de desarrollar el programa es utilizar los comandos y utilidades del sistema operativo LINUX para obtener una aplicación gráfica que permita monitorear y administrar el tráfico de datos en redes LAN, para lo cual se utiliza un ambiente de ventanas; en el cual la ejecución del programa requiere que el equipo donde se instale la aplicación esté conectado a una red y tenga la configuración de red básica para su funcionamiento, además deben haber sido ejecutados los comandos necesarios para el funcionamiento de la aplicación. En el caso de la administración del tráfico de datos, el programa debe ser ejecutado en el host que esté actuando como servidor de la red LAN, es decir, éste debe proveer las aplicaciones que el software de Administración de tráfico va a controlar.

2.4.2 REQUERIMIENTOS DE INGRESO DE DATOS

Inicialmente el programa debe recoger la información de red, además debe determinar las redes activas, luego comienza la exploración de dispositivos, monitoreo y administración de tráfico dentro de la red.

Todo esto se lo realiza mediante el uso de cuadros de diálogo con campos específicos para los diferentes tipos de datos a ingresar.

2.4.3 REQUERIMIENTOS DE SALIDA DE DATOS

Una vez terminado con el monitoreo y exploración, se necesita mostrar los resultados obtenidos, en forma de lista de los componentes o dispositivos encontrados y tabla de estadísticas de los resultados del monitoreo de tráfico.
Para un mejor entendimiento de los resultados, la lista de dispositivos encontrados se la clasifica en entorno Windows, entorno LINUX y dispositivos administrables; en cambio, por el lado de la tabla de paquetes capturados por el monitor de tráfico, se presentan los resultados en barras y porcentajes.

2.5 DISEÑO FUNCIONAL DEL PROGRAMA

Para el desarrollo del programa se han definido los siguientes componentes, los cuales se indican en la figura 2.11.

Estos componentes son los siguientes:

- Entrada de datos
- Procesamiento de datos y
- Salida de datos.

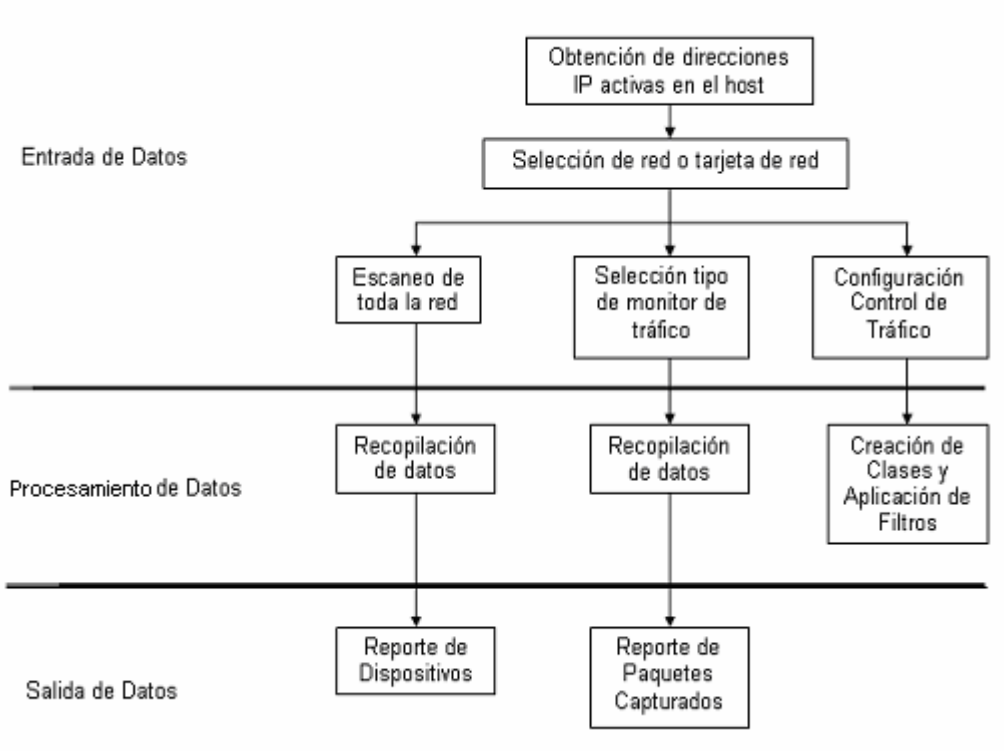


Figura 2.11.- Diseño Funcional del programa Alcatraz.

2.6 DESARROLLO DEL PROGRAMA

El ambiente de programación en SH nos permite crear aplicaciones en las cuales podemos llamar a ejecutar comandos internos de LINUX y los resultados enviarlos a archivos, en donde se los pueda modificar para su posterior presentación a través de la interfaz del Qt/Designer.

Dentro del ambiente de programación Qt/Designer se pueden elaborar ventanas con menús desplegables, menús contextuales, cuadros de diálogo y cuadros de herramientas; facilitando de esta manera el diseño de una aplicación funcional con herramientas amigables para el usuario final.

2.6.1 DISEÑO DEL PROGRAMA PARA EL MONITOREO DE DISPOSITIVOS

Para el funcionamiento adecuado del programa para el Monitoreo de dispositivos, es necesario que el host donde se ejecuta la aplicación cumpla con los siguientes requisitos:

- En el archivo /etc/snmp/snmpd.conf se debe incluir la red o redes a las que pertenece el host
- El servicio snmpd debe estar iniciado, y además
- Los comandos *nmap, nmblookup* deben estar instalados y funcionando correctamente.

En el diagrama de la figura 2.12, se muestran los pasos utilizados para la obtención de la respectiva información del monitoreo de dispositivos.

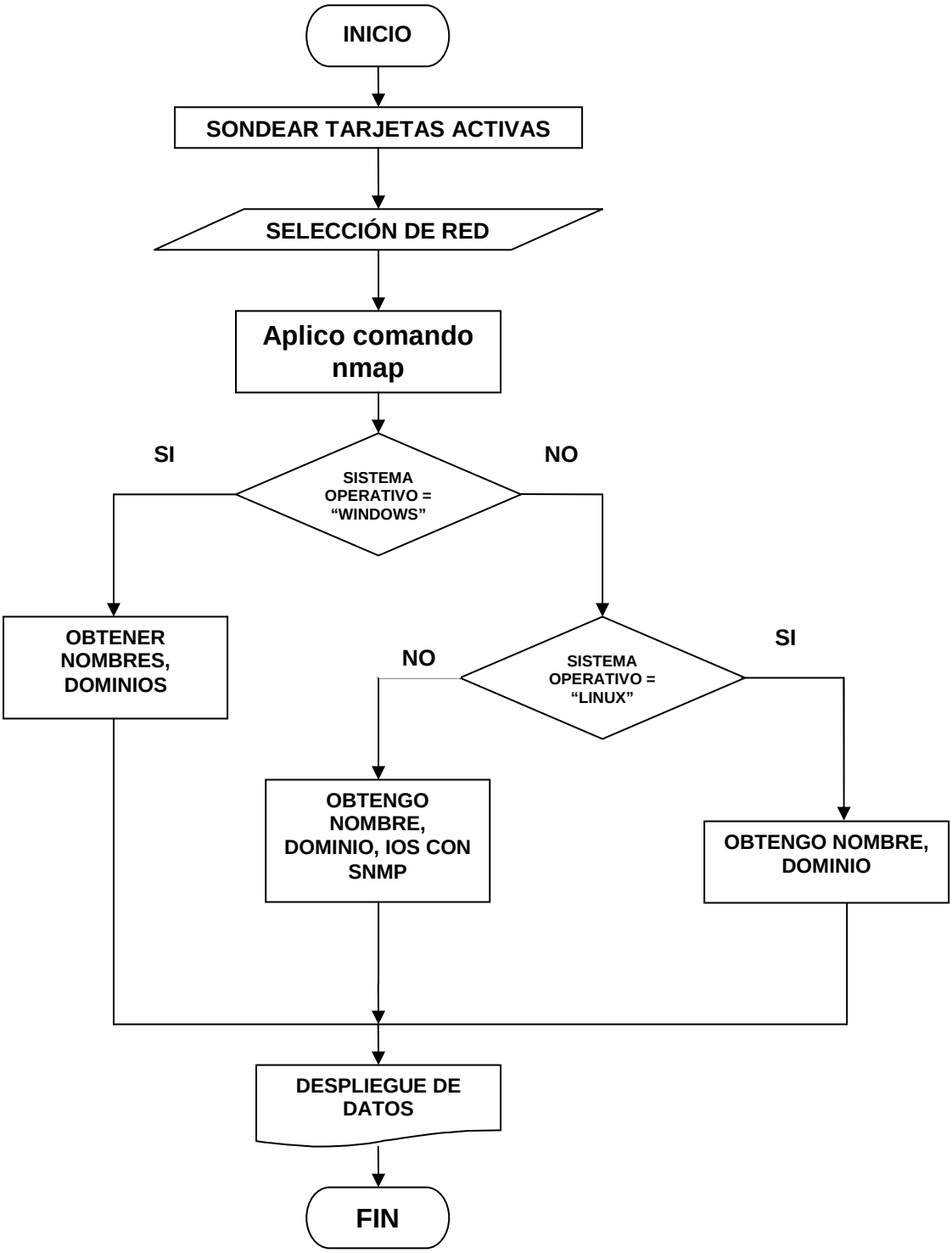


Figura 2.12.- Diagrama de Flujo del Monitor de dispositivos

2.6.1.1 Entrada de datos para el Monitoreo de dispositivos

El comando **ip route list** permite obtener la información de la red o redes activas configuradas en el equipo, es decir muestra las interfaces de red con su respectivo nombre, dirección ip, dirección de red y máscara de red.

Otra opción que presenta, es la del ingreso manual de la dirección de red, máscara e interfaz, además del ingreso de la comunidad SNMP a la cual se va a acceder en dispositivos administrables y equipos LINUX.

En la Figura 2.13 se indica el diagrama de flujo de la entrada de datos para el Monitoreo de dispositivos.

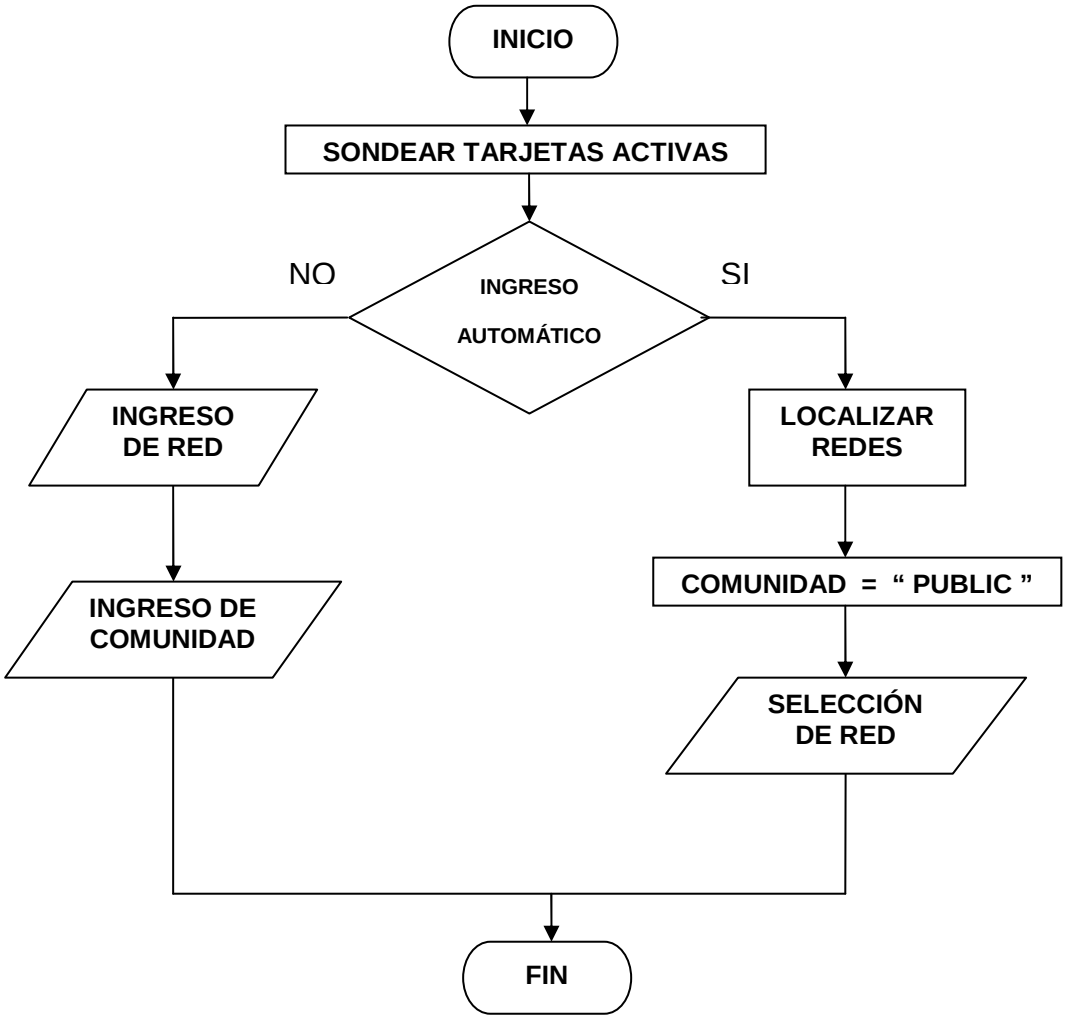


Figura 2.13.- Diagrama de Flujo de entrada de datos del Monitor de dispositivos

2.6.1.2 Procesamiento de datos para el Monitoreo de dispositivos

Una vez ingresados los datos, el programa de exploración de dispositivos se inicia con la ejecución del comando **nmap –O <dirección de red/máscara>,** donde -O es la opción que le indica a *nmap* realizar la exploración con obtención del sistema operativo, direcciones ip activas, direcciones MAC con su respectivo fabricante y tipo de dispositivo.

La distinción de los resultados obtenidos se la hace mediante la identificación del tipo de dispositivo, esto es, dividir entre hosts y dispositivos administrables; y si es un host, la subdivisión se realiza a través del tipo de sistema operativo.

Luego, a los dispositivos clasificados como host y sistema operativo Windows, se les extrae el nombre y dominio de red a través del comando **nmblookup –A <dirección ip>,** donde –A es la opción que permite la obtención del nombre del host.

Para los dispositivos clasificados como host y con sistema operativo LINUX se extrae el nombre y dominio de red a través del comando **snmpwalk –v1 –c <comunidad> <dirección ip> system.sysName.0** (donde –v es la opción de versión de SNMP, -c es la opción de comunidad SNMP). Si el dispositivo se encuentra clasificado como administrable, entonces, se procede a obtener su nombre (si lo tiene), tipo de sistema operativo IOS (Input Ouput System) para poder distinguir entre switch o ruteador, esto se lo hace mediante el comando **snmpwalk –v1 –c <comunidad> <dirección ip> 1.3.6.1.2.1.1.1**.

De los archivos generados con los resultados del comando *nmap* y con apoyo de los comandos para tratamiento de texto como *echo, cat, sed* y *awk*, se crean otros documentos en los que se resalta la información de direcciones IP, direcciones MAC, tipo de dispositivo y sistema operativo.

Para los archivos con resultados obtenidos del comando *nmblookup,* se realiza el tratamiento de texto necesario para generar un documento que contenga información del nombre del host y su respectivo dominio.

Los archivos que contienen información de nombre, dominio y tipo, para los dispositivos administrables, son generados de la misma manera que los anteriores, pero la fuente de ésta información son los resultados del comando *snmpwalk*.

Los archivos que contienen la información organizada de acuerdo a los requerimiento de salida de datos para el Monitoreo de dispositivos, son guardados en un directorio de Reportes con la fecha y hora en la que se inició el monitoreo. En la figura 2.14 se indica el diagrama de flujo del Procesamiento de datos del Monitor de dispositivos.

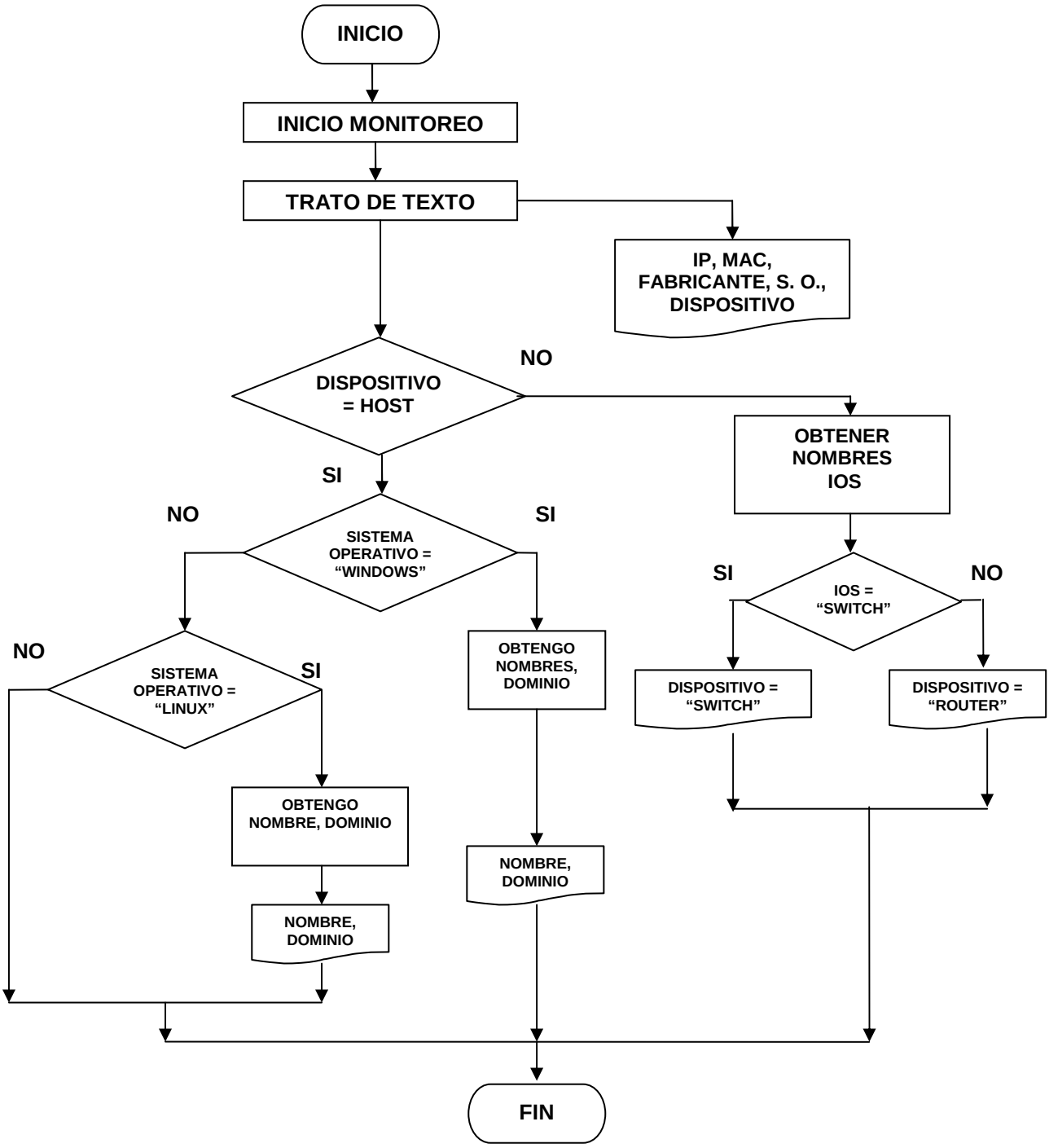


Figura 2.14.- Diagrama de Flujo del procesamiento de datos del Monitor de dispositivos

2.6.1.3 Salida de datos para el Monitoreo de dispositivos

El despliegue de resultados, se realiza a través de la lectura de los archivos generados en cada uno de los procesos anteriores, ubicándolos en listas con diferentes columnas, donde se presenta la información recopilada durante el monitoreo.

Los resultados se muestran a manera de reporte y clasificados por tipo, es decir, hosts Windows, hosts LINUX y dispositivos Administrables, cada uno con sus respectivos detalles.

Dentro del proceso de actualización de monitoreo de dispositivos, a través del comando *ping* se verifica si las direcciones IP que estuvieron en el monitoreo anterior siguen activas, de lo cual, se genera un documento que muestra un listado con las direcciones IP que no respondieron a este comando.

Existe además la posibilidad de acceder a reportes de monitoreos anteriores, a través de la opción archivos en la barra de menús, la cual desplegará el reporte en forma de texto plano en la pantalla.

2.6.2 DISEÑO DEL PROGRAMA PARA EL MONITOREO DE TRÁFICO

Para el correcto funcionamiento del Monitor de Tráfico es necesario tener instalado el comando *tcpdump*.

Dentro del monitoreo de tráfico, se pueden realizar las siguientes tareas:

- Monitoreo de Tráfico por protocolos.
- Monitoreo de Tráfico por puertos, y además
- Dentro de cada uno de los monitoreos, la opción de monitorear sólo el tráfico del host local.

En el diagrama de la figura 2.15 se muestra el procedimiento utilizado para realizar el monitoreo de tráfico.

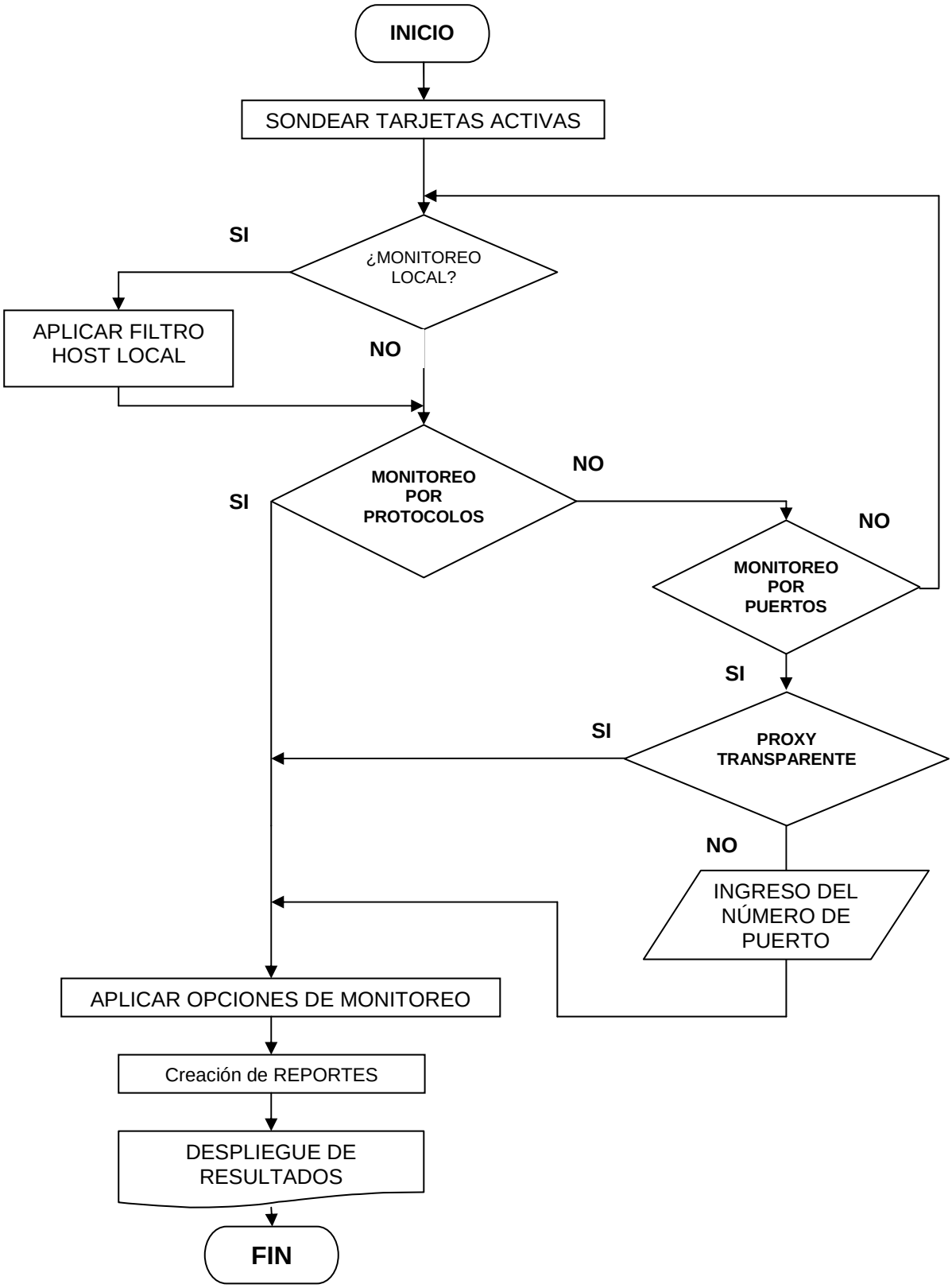


Figura 2.15.- Diagrama de Flujo del Monitor de Tráfico

2.6.2.1 Entrada de datos para el Monitoreo de Tráfico

Para obtener la información de las interfaces de red activas configuradas en el host, se lo hace a través del comando **ip –o -4 addr show**, el mismo que despliega información de las interfaces de red con su respectiva dirección ip, máscara y nombre.

Para el Monitor de Tráfico se debe seleccionar el tipo de monitoreo que se desea realizar, es decir, entre monitoreo de tráfico por protocolos o monitoreo de tráfico por puertos. Si la selección es por puertos, se debe además elegir si se utiliza o no proxy transparente, dentro de lo cual si se eligió proxy no transparente se deberá ingresar el número de puerto proxy.

2.6.2.2 Procesamiento de datos para el Monitoreo de Tráfico

Con los datos de información de interfaz de red, se procede con la ejecución del comando **tcpdump**, en el cual, tratándose del monitor de tráfico por protocolos la forma del comando es la siguiente: **tcpdump –i <interfaz> <protocolo>**, donde, dentro de la opción *interfaz* irá el nombre y en *protocolo* se pueden ingresar opciones como: tcp, udp, icmp y arp. Cuando la selección sea de monitor de tráfico por puertos, la forma del comando será así: **tcpdump –i <interfaz> port <puerto>,** donde, en la opción *puerto* se puede ingresar cualquiera de los números de puerto disponibles en el archivo /etc/services para su monitoreo.

La recopilación de los paquetes capturados por el comando *tcpdump* son enviados a un archivo de registro mediante la opción **–w**, lo cual, una vez terminado el proceso de captura, se crea un archivo con la información necesaria para obtener los resultados totales a partir de la opción **-r** del mismo comando *tcpdump.* Este procedimiento se lo realiza para cada uno de los monitoreo de puertos o protocolos, para luego realizar el cálculo de las estadísticas generales y parciales mediante el uso de comandos para tratamiento de texto.

Los archivos que se crean a partir del tratamiento de texto, son guardados en un directorio de Reportes con la fecha y hora en que inició el monitoreo.

En la figura 2.16 se indica el diagrama de flujo del procesamiento de datos del monitor de tráfico

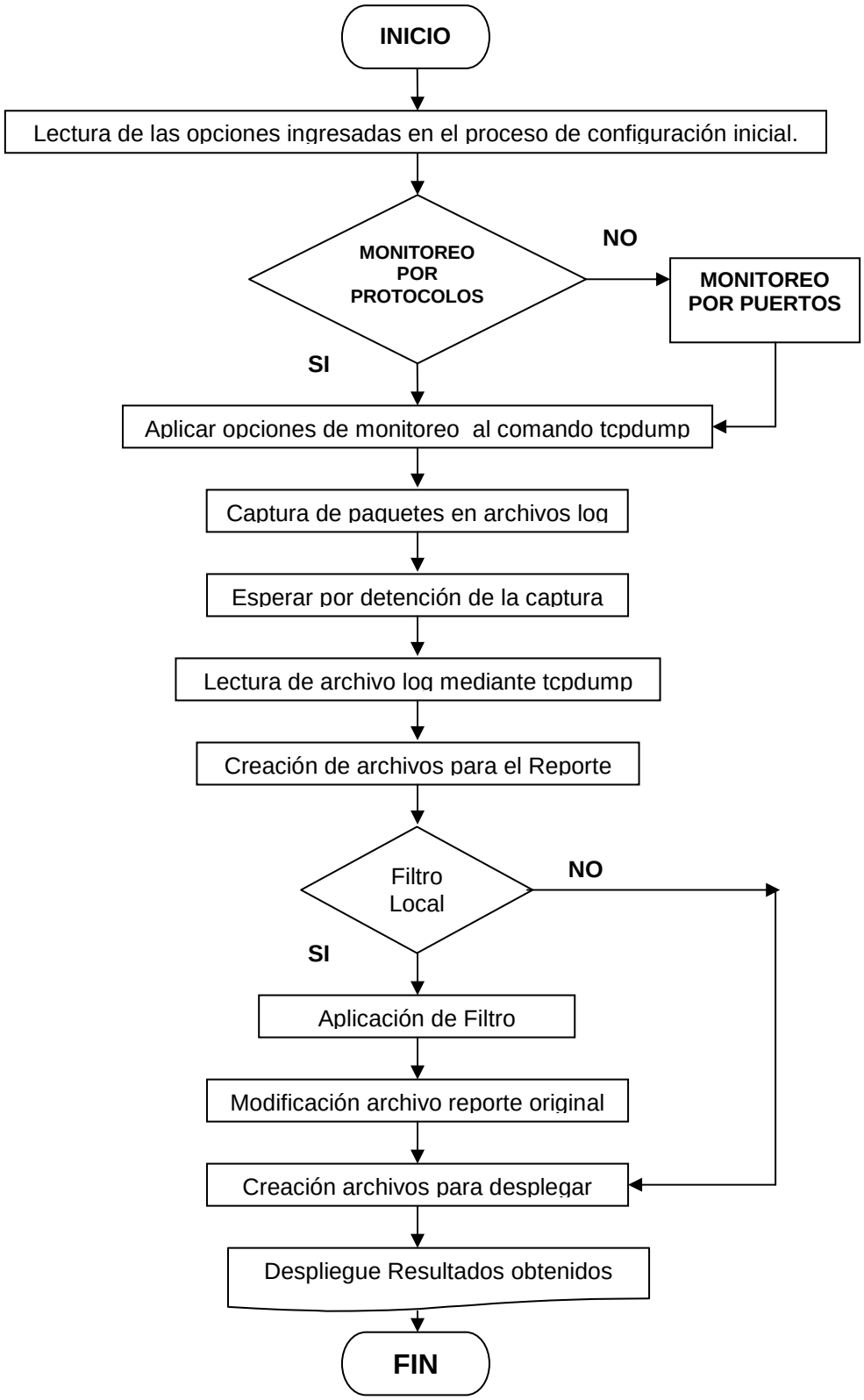


Figura 2.16.- Diagrama de Flujo del procesamiento de datos del Monitor de Tráfico.

2.6.2.3 Salida de datos del Monitoreo de Tráfico

La presentación de los resultados se realiza a través de la lectura de los archivos generados en cada uno de los procesos anteriores ubicándolos en listas donde se presenta la información detallada y estadística de lo recopilado durante el monitoreo.
Se muestra adicionalmente los porcentajes de paquetes capturados en barras para una mejor visualización de lo obtenido.
Existe además la posibilidad de observar reportes de monitoreos anteriores a través del menú y opción archivos de monitor de tráfico, la cual desplegará el reporte en forma de texto plano en pantalla.

2.6.3 DISEÑO DEL PROGRAMA PARA EL CONTROL DE TRÁFICO DE DATOS

Para el funcionamiento adecuado del Control de Tráfico es necesario tener instalado los comandos *htb* y *tc*.
Dentro de las tareas presentes para realizar el Control de Tráfico se tienen:

- Asignación de ancho de banda máximo del enlace.
- Elección de protocolos y asignación del respectivo ancho de banda.
- Edición de ancho de banda para tráfico no clasificado.
- Selección de direcciones IP, a las cuales se aplicará el control de tráfico.

En el diagrama de la figura 2.17 se muestra el proceso empleado para realizar el control de tráfico.

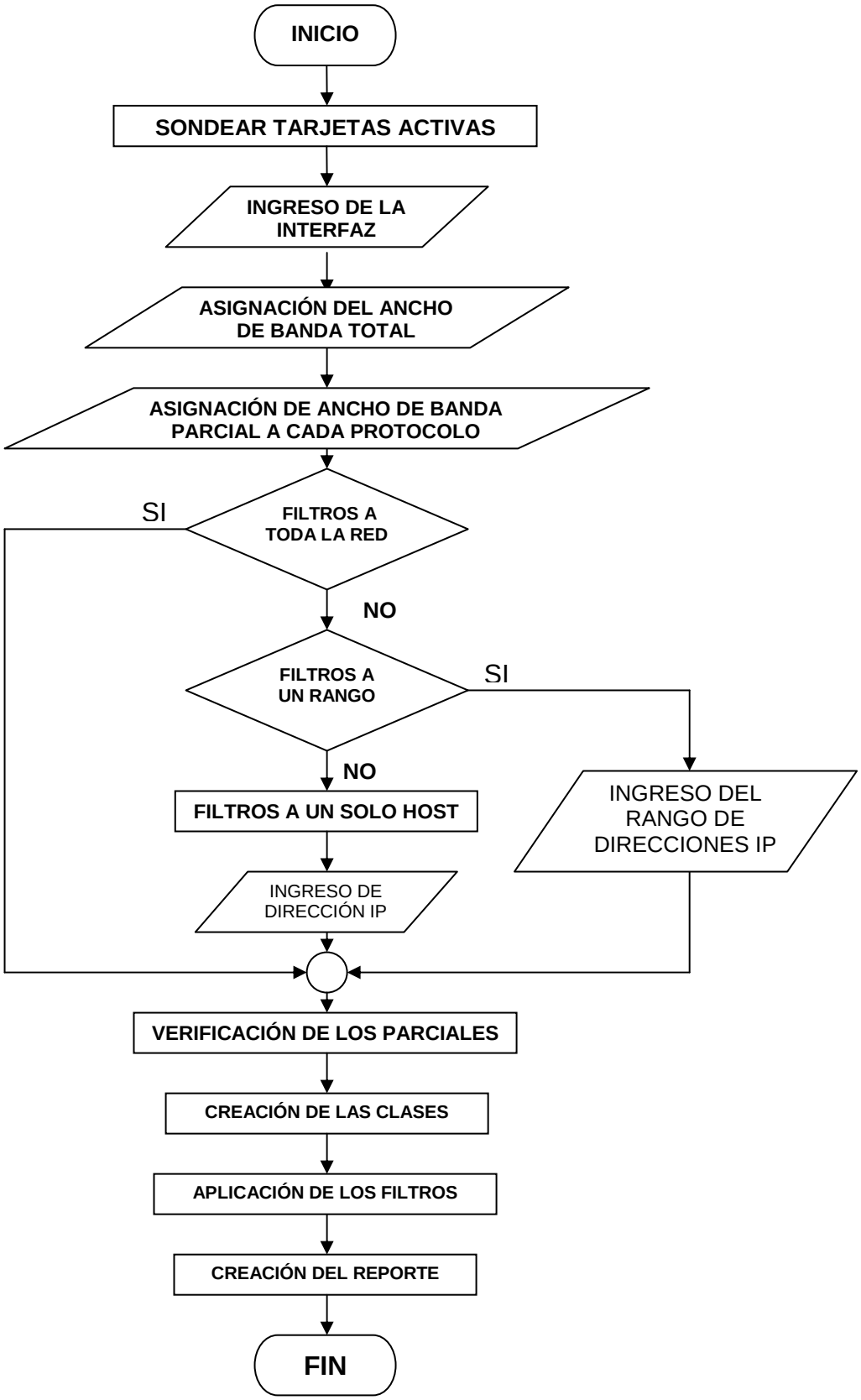


Figura 2.17.- Diagrama de Flujo del Control de Tráfico.

2.6.3.1 Entrada de datos para el Control de Tráfico

Para obtener la información de las redes activas configuradas en el equipo, se lo hace a través del comando **ip route list** mismo que despliega las interfaces de red con su respectivo nombre, dirección IP, dirección de red y máscara de red.
El usuario deberá conocer el ancho de banda máximo disponible en la interfaz del servidor que se conecta a la red LAN, para que éste sea asignado dentro del respectivo campo dentro de la aplicación, como el límite de ancho de banda de salida que brinda el servidor. Al ancho de banda máximo, se lo subdivide de acuerdo a la elección realizada por el usuario, en parciales de menor ancho de banda asignados a los protocolos ICMP, HTTP, FTP y SMTP.
Además, el programa permite elegir entre aplicar los filtros a todas las direcciones IP activas presentes en la red, a un rango de direcciones IP o a una determinada dirección IP. La lista de direcciones IP de los dispositivos que se verán afectados por los filtros, se la obtiene del reporte generado previamente en el monitoreo de dispositivos, de lo contrario la aplicación no funcionará correctamente.

2.6.3.2 Procesamiento de datos para el Control de Tráfico

Con los datos ingresados anteriormente el funcionamiento del programa se estructura de la siguiente manera:

- Se aplica el comando **tc qdisc add dev <interfaz> root handle 1: htb default 60**, el cuál indica que se llamará al comando *htb* a través del comando *tc.* Además se indica que se creará una clase con identificador 1: para una determinada interfaz. La sección *default 60* presente en éste línea de comando señala que, el tráfico no clasificado pasará a través de la clase 1:60.
- Luego con el comando **tc class add dev <interfaz> parent 1: classid 1:1 htb** se creará la clase raíz 1:1 dentro de la clase htb 1: ó 1:0. Además, dentro de la clase raíz se crearán:
    - La clase 1:10, la cual contiene a las clases que se crean por cada uno de los protocolos.

  - La clase 1:60, que llevará el tráfico no clasificado.
- Las clases pertenecientes a la clase 1:10, se crean de acuerdo a los protocolos elegidos por el usuario, con sus respectivos anchos de banda.

En la figura 2.18 se indica la creación de los diferentes tipos de clases.

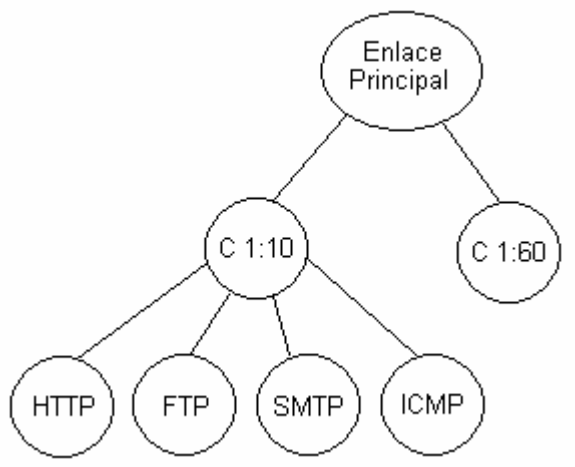
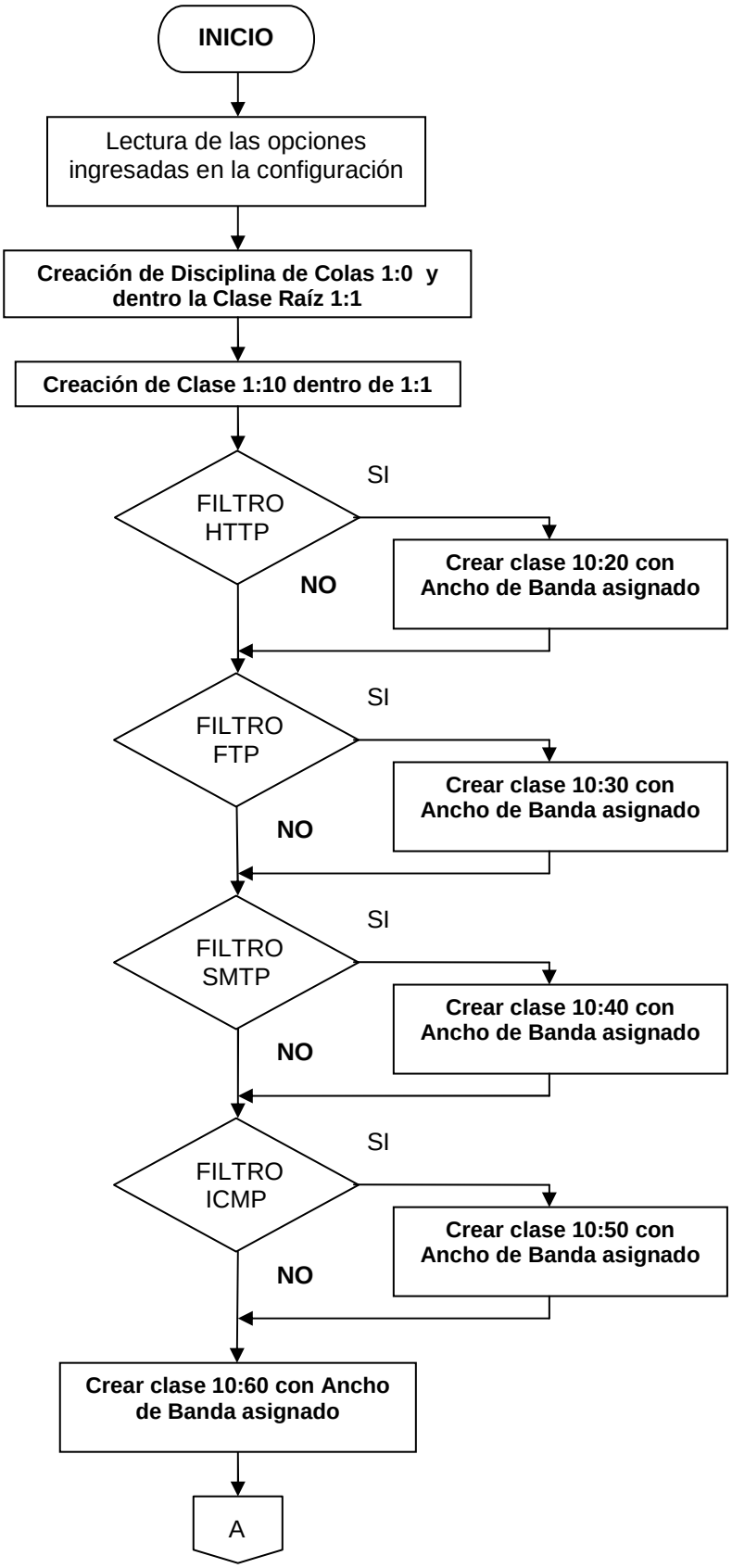


Figura 2.18.- División del enlace para el Control de Tráfico.

- Una vez creadas las clases, se aplican filtros a los paquetes para encaminarlos a su clase correspondiente, esto se puede hacer mediante la siguiente línea de comando **tc filter add dev <interfaz> parent 1: protocol ip u32 match ip dst <dirección IP> flowid 1:10.** La ejecución de ésta línea se la realiza así debido a que, es necesario utilizar filtros como el *u32* para la clasificación del tráfico. El filtro *u32* permite aplicar el criterio de dirección IP origen o destino del paquete. Además con la instrucción *flowid* se puede dirigir los paquetes a la clase que se desee. [7]
- Al final, una vez que se hayan aplicados los filtros al tráfico y creadas las clases, mediante el mismo comando **tc** se genera un reporte donde se detallan las clases creadas y las direcciones IP involucradas en el filtrado.

En la figura 2.19 se indica el diagrama de flujo del procesamiento de datos para el control de tráfico.


INICIO
Lectura de las opciones ingresadas en la configuración
Creación de Disciplina de Colas 1:0 y dentro la Clase Raíz 1:1
Creación de Clase 1:10 dentro de 1:1
FILTRO HTTP
SI
NO
Crear clase 10:20 con Ancho de Banda asignado
FILTRO FTP
SI
NO
Crear clase 10:30 con Ancho de Banda asignado
FILTRO SMTP
SI
NO
Crear clase 10:40 con Ancho de Banda asignado
FILTRO ICMP
SI
NO
Crear clase 10:50 con Ancho de Banda asignado
Crear clase 10:60 con Ancho de Banda asignado
A

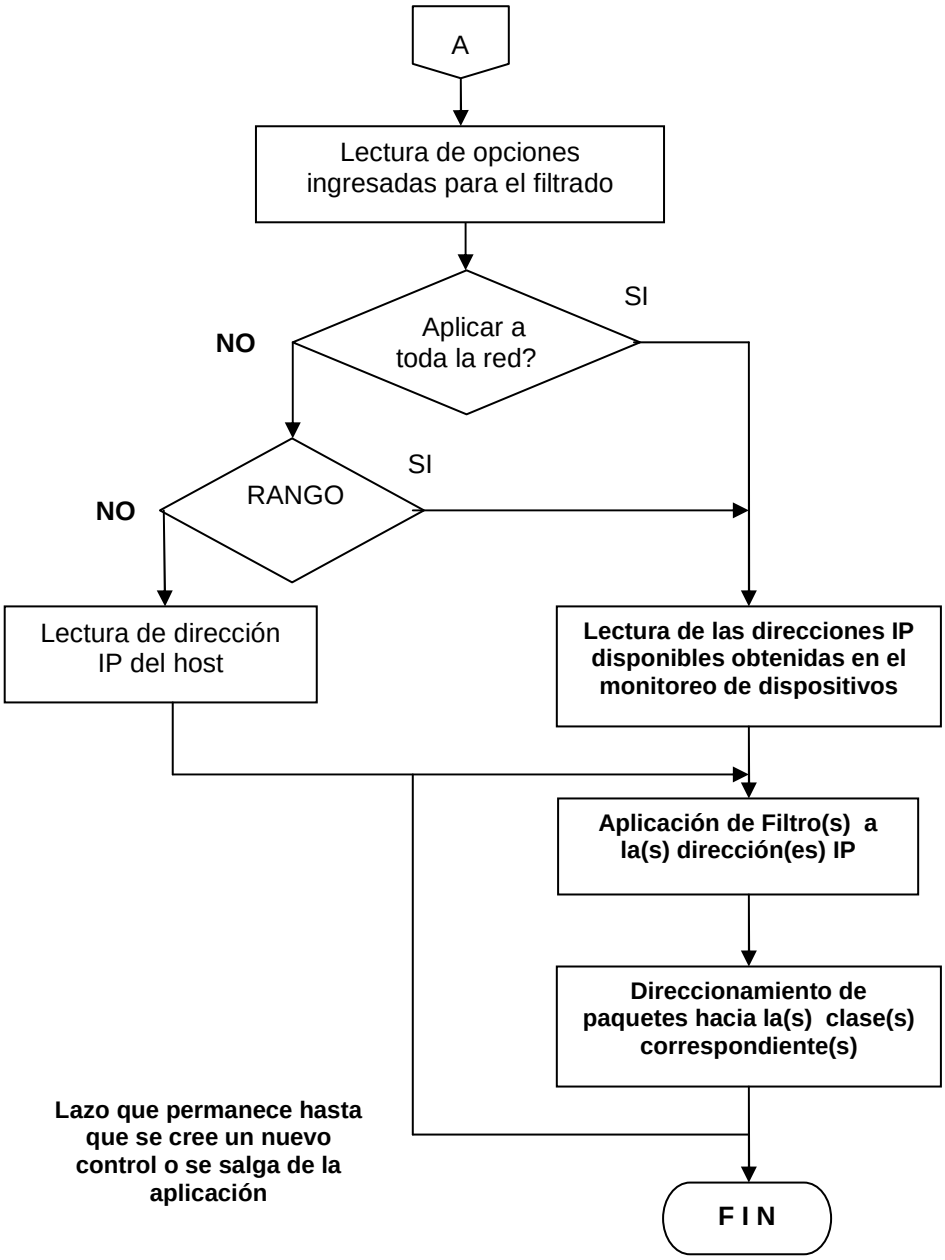


Figura 2.19.- Diagrama de Flujo del Procesamiento de datos para el Control de Tráfico.

2.6.3.3 Salida de datos para el Control de Tráfico

La interfaz gráfica presenta la opción de acceder al reporte creado al aplicar el control de tráfico, así como también la opción de ir observando las estadísticas

que guardan cada una de las clases de acuerdo al número de paquetes que cursan por cada una de ellas.

2.6.4 HERRAMIENTAS PRESENTES EN LA APLICACIÓN PARA MONITOREO Y CONTROL DE TRÁFICO DE DATOS

En la aplicación de monitoreo y control de tráfico, se pueden encontrar dentro de la opción *Herramientas* en la barra de menús, tres herramientas que a continuación se describen:

- **Actividad Local.-** Para ésta herramienta se utilizó el comando **netstat –aut**, cuya función es permitir al usuario observar un reporte de las conexiones activas y establecidas por el host local con Internet, mostrándose además del tipo de protocolo al que pertenecen, peticiones enviadas y recibidas, direcciones locales, direcciones remotas y estado de las conexiones.

- **Ping.-** Para ésta herramienta se empleó el comando **ping** con variantes tales como: número de paquetes y tamaño del paquete, opciones que pueden ser editadas por el usuario. Su función es tener una herramienta para enviar y recibir peticiones de eco para determinar si un destino está listo para recibir información.

- **Traceroute.-** Para ésta herramienta se utilizó el comando **traceroute** y su función es mostrar el camino que se necesita para llegar a una host remoto y trazar la ruta.

2.7 REQUERIMIENTOS DE SOFTWARE DE LA APLICACIÓN DE MONITOREO Y ADMINISTRACIÓN DE RED

Los requerimientos de software mínimos que deben estar instalados son los siguientes:

- El paquete de programas **Coreutils** que contiene un completo conjunto de utilidades básicas para el intérprete de comandos, de los cuales los utilizados para nuestra aplicación fueron: cat, chmod, cp, cut, date, df, dir, echo, expr, head, kill, mkdir, mv, pwd, rm, sleep, sort, touch, uname, ll y wc.
- Net-snmp-utils 5.1.2-11.
- Net-snmp 5.1.2-11.
- Nmap 3.70-1.
- Nmblookup 3.0.10-1.4E.
- Iproute2
- Tcpdump 3.8
- Sed 4.1.2
- Awk 3.1.3
- Grep 2.5.1
- Libcap 0.8.3
- Netstat 1.42
- Make 3.80
- Inetutils 1.4.2
- Traceroute 1.4a12

2.8 REQUERIMIENTOS DE HARDWARE

Las siguientes son características mínimas del host donde se debe instalar y ejecutar la aplicación desarrollada.

- Procesador Pentium IV de 1.6 GHz o superior.
- Memoria RAM de 256 MB.
- Espacio libre en disco duro de 2 GB.

BIBLIOGRAFÍA CAPÍTULO 2

[1] Manual incluido en la distribución del sistema operativo LINUX.

[2] osmosislatina**.**com
Comandos estándares.
http://www.osmosislatina.com/LINUX/comandos.jsp#generales

[3] escomposLINUX.org
Introducción a Samba.
http://www.escomposLINUX.org/lfs-es/blfs-es-5.1/server/samba3.html

[4] luxik.cdi.cz
HTB LINUX queuing discipline manual - user guide
http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm

[5] LINUX-cd.com.ar
Scripts básicos para bash
http://LINUX-cd.com.ar/manuales/LINUX-colegio/scripts-basicos-para-bash.html

[6] trolltech.com
Qt Overview
http://www.trolltech.com/products/qt/index.html

[7] greco.dit.upm.es
Control de tráfico utilizando LINUX.
greco.dit.upm.es/~david/tar/trabajos2002/05-Contro-Trafico-LINUX-Fernando-David-Gomez-res.pdf

CAPÍTULO 3

PRUEBAS DEL MONITOREO Y ADMINISTRACIÓN

Las pruebas se efectuaron en dos redes LAN ya establecidas, a través de obtención de datos de Monitoreo de dispositivos y Monitoreo de Tráfico.
Para esto se utilizan dos redes de prueba: LAN A y LAN B.

3.1 LIMITACIONES DE LA APLICACIÓN

A continuación se describen algunas limitaciones que tiene la aplicación de Monitoreo:

- No se puede determinar el nombre del dispositivo o host, cuando éste se encuentre en los siguientes casos:

  a) Sistema Operativo Windows y protocolo NetBIOS deshabilitado,
  b) Dispositivos administrables sin configuración SNMP, y
  c) Hosts LINUX con servicio SNMP apagado.

  Ésta información no se puede determinar, debido a que se envían paquetes de exploración que necesitan de estos servicios para obtener los resultados correspondientes. De él o los dispositivos que se encuentren en ésta situación, el reporte de los resultados acerca de ellos será mínimo.

- Para el Monitor de Tráfico, si el usuario no sabe de la utilización de un proxy y si éste es o no transparente, el análisis de tráfico HTTP se lo realiza a través del puerto 80, además si los números de puerto asignados originalmente a las aplicaciones HTTP, FTP, SMTP y SNMP fueron cambiados en el host donde se ejecuta la aplicación, el Monitoreo de Tráfico por puertos no entregará los resultados adecuados.

- Para la Administración del tráfico de datos, la ejecución del programa se la debe realizar en un host que esté sirviendo alguna aplicación tal como, HTTP, FTP, SMTP o ICMP a la red LAN, caso contrario, no se observará ningún efecto, ya que la función del programa es limitar el ancho de banda de salida desde la interfaz del servidor hacia los hosts clientes de la LAN. Además el usuario debe tomar en cuenta el ingreso o no del número de puerto proxy, ya que caso contrario, el control para el tráfico HTTP se lo realiza a través del puerto 80.
- Dentro del Monitoreo de dispositivos, los resultados son obtenidos y presentados de manera automática, a diferencia del Monitoreo de Tráfico en el cual, se debe detener la captura de paquetes para que luego de esto, se puedan desplegar los resultados en el espacio determinado para ello. Se facilita la interpretación de los resultados del Monitoreo de dispositivos a través de páginas ubicadas dentro de la misma ventana donde se clasifica la información en: Reporte General, Entorno Windows, Entorno LINUX, Dispositivos Administrables y Reporte de Errores(si se presentó el caso). Para el Monitoreo de Tráfico se presentan los resultados en forma de tabla de estadísticas generales y, en otra ventana se presenta el detalle de los paquetes capturados. La presentación de resultados de la Administración del tráfico de datos, brinda estadísticas de las clases y filtros creados, en donde además se puede ir refrescando la información que se va generando.

3.2 LA APLICACIÓN

El nombre de la aplicación desarrollada es ALCATRAZ, y su ventana principal se muestra en la figura 3.1.

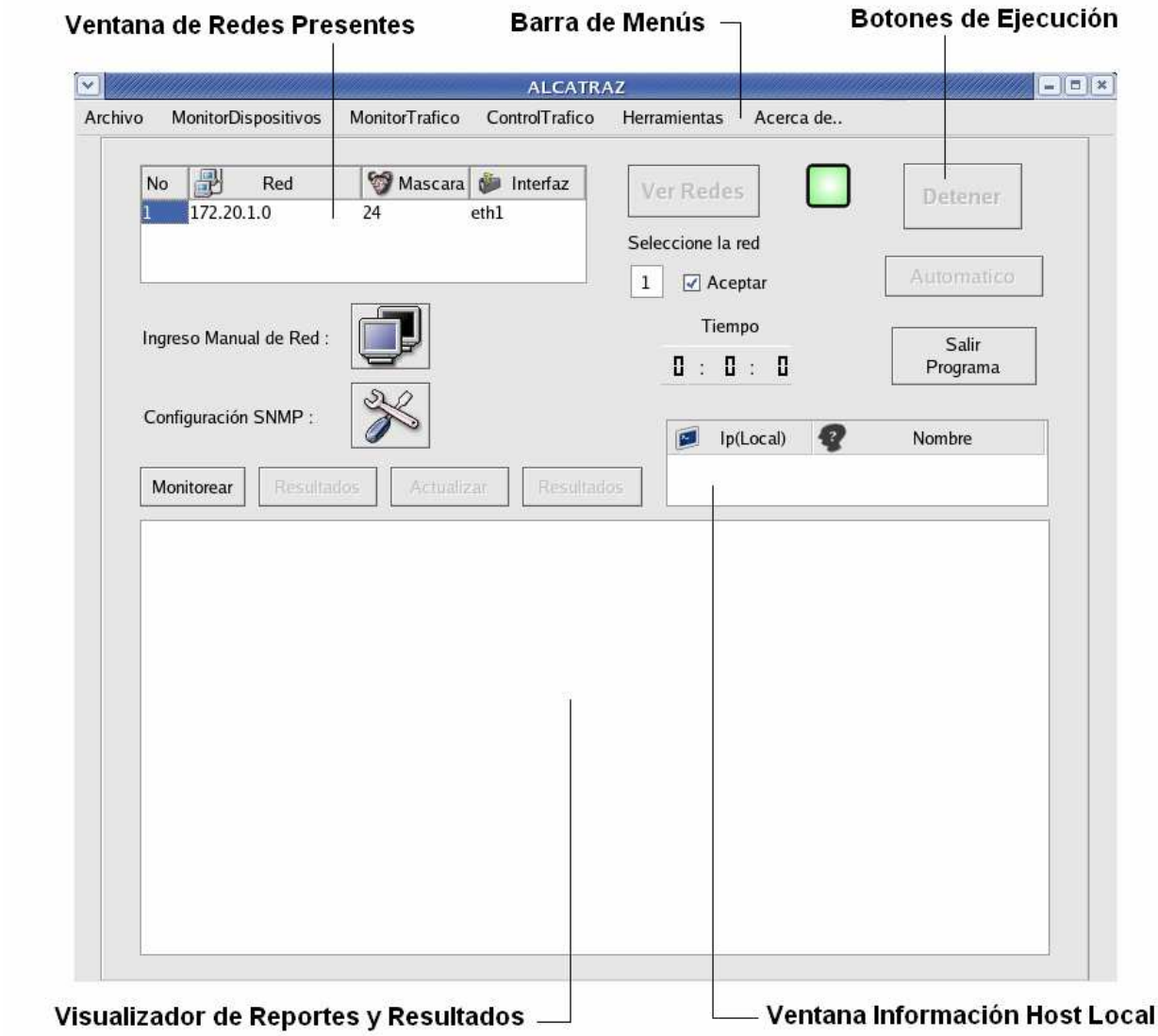


Figura 3.1.- Ventana Principal Aplicación ALCATRAZ.

En la barra de Menús se encuentran opciones como: *Archivo*, desde el cual se puede abrir reportes, guardar con otro nombre algún archivo abierto, imprimir y salir del programa.

En la figura 3.2 se muestra el menú Archivo con las diferentes opciones que ofrece para el Monitor de Tráfico.

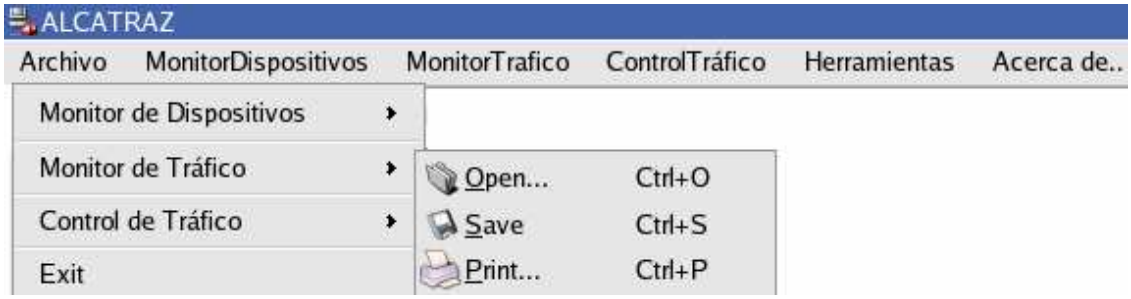


Figura 3.2.- Opciones del Menú Archivo.

En el submenú *MonitorDispositivos* de la barra de menús, se puede activar la ventana principal del programa para monitorear dispositivos desde la opción **Mostrar**; dentro del mismo submenú, con la opción **Ayuda**, se puede desplegar una serie de instrucciones o guía del programa.

En el submenú *MonitorTráfico*, la opción **Mostrar** nos lleva a la ventana donde se puede configurar las diferentes alternativas del programa para el monitoreo de tráfico; mientras que, la opción **Detalles** nos permite acceder a la ventana que contiene una lista detallada de los paquetes del último monitoreo realizado, o a su vez, para poder abrir y revisar archivos de la lista de reportes anteriores. Finalmente, la opción **Ayuda** despliega un conjunto de instrucciones para facilitar el uso del programa.

En la barra de Menús dentro de la opción *ControlTráfico*, se tiene las siguientes opciones: **Mostrar**, que permite acceder a la ventana de configuración y ejecución del programa para el Control de tráfico; **Detalles**, muestra en una ventana el contenido del detalle de los filtros y clases creadas. Y, como en los casos anteriores, la opción **Ayuda** despliega una serie de instrucciones para el uso de esta parte de la Aplicación.

El submenú *Herramientas* de la barra de menús, contiene herramientas útiles de red como: **Actividad Local**, para observar el estado de las conexiones TCP y UDP del host local; **Ping**, para comprobación de conectividad y **Traceroute** para trazar la ruta hacia un dispositivo remoto.

En la figura 3.3 se muestra el menú *Herramientas* con sus diferentes opciones.

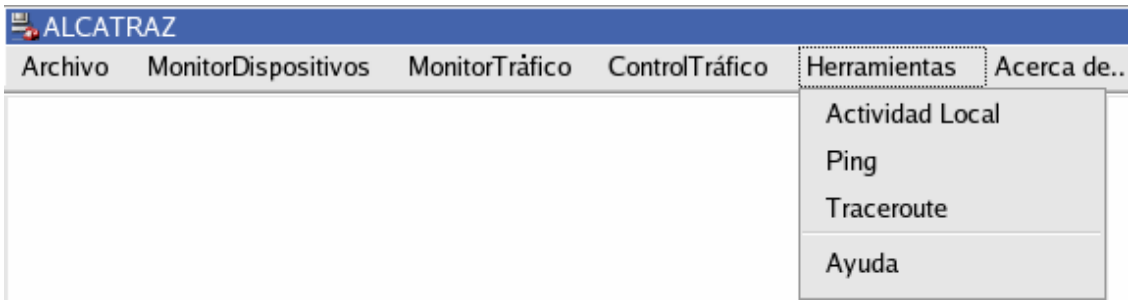


Figura 3.3.- Opciones Menú Herramientas.

3.2.1 MONITOR DE DISPOSITIVOS

Al momento de pulsar el botón *Ver Red*es, el programa detecta automáticamente las redes configuradas en el host con su respectiva interfaz de red. Una vez seleccionada la red, se habilita el botón con la opción de *Monitorear*. Al pulsar el botón *Monitorear,* la exploración se inicia y en pantalla se puede observar, un reloj que muestra el tiempo que se tarda en monitorear la red, además el botón con la opción *Detener*, el mismo que permite detener el monitoreo en forma secuencial para que se puedan observar los resultados capturados hasta ese momento. En la figura 3.4, se muestra lo indicado anteriormente.

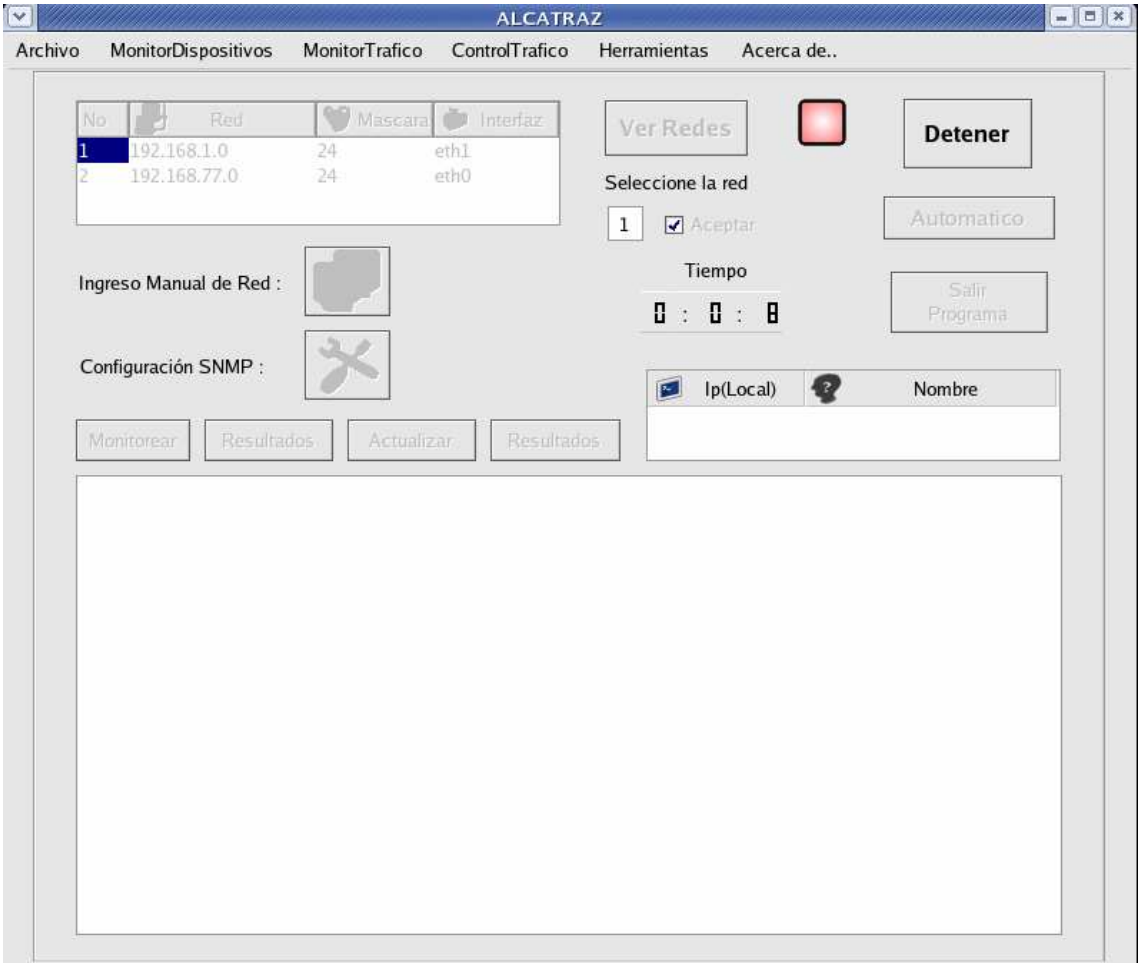


Figura 3.4.- Ejecución monitoreo de dispositivos.

Luego de terminado el proceso de captura y tratamiento de datos, los resultados del monitoreo de dispositivos aparecen automáticamente.

La clasificación de los resultados se muestra en distintas pantallas de acuerdo al tipo de dispositivo (host o dispositivo administrable) y al sistema operativo (Windows o LINUX).

La prueba realizada en la red LAN A, ilustra un ambiente combinado de dispositivos administrables y hosts en Windows como en LINUX.

La red en cuestión consta de aproximadamente 105 dispositivos que se encuentran unidos por switches administrables.

Parte del reporte general de dispositivos que el programa obtuvo, se detalla a continuación en la figura 3.5, donde se observan direcciones IP, direcciones MAC, fabricante de la NIC, tipo de dispositivo y sistema operativo.

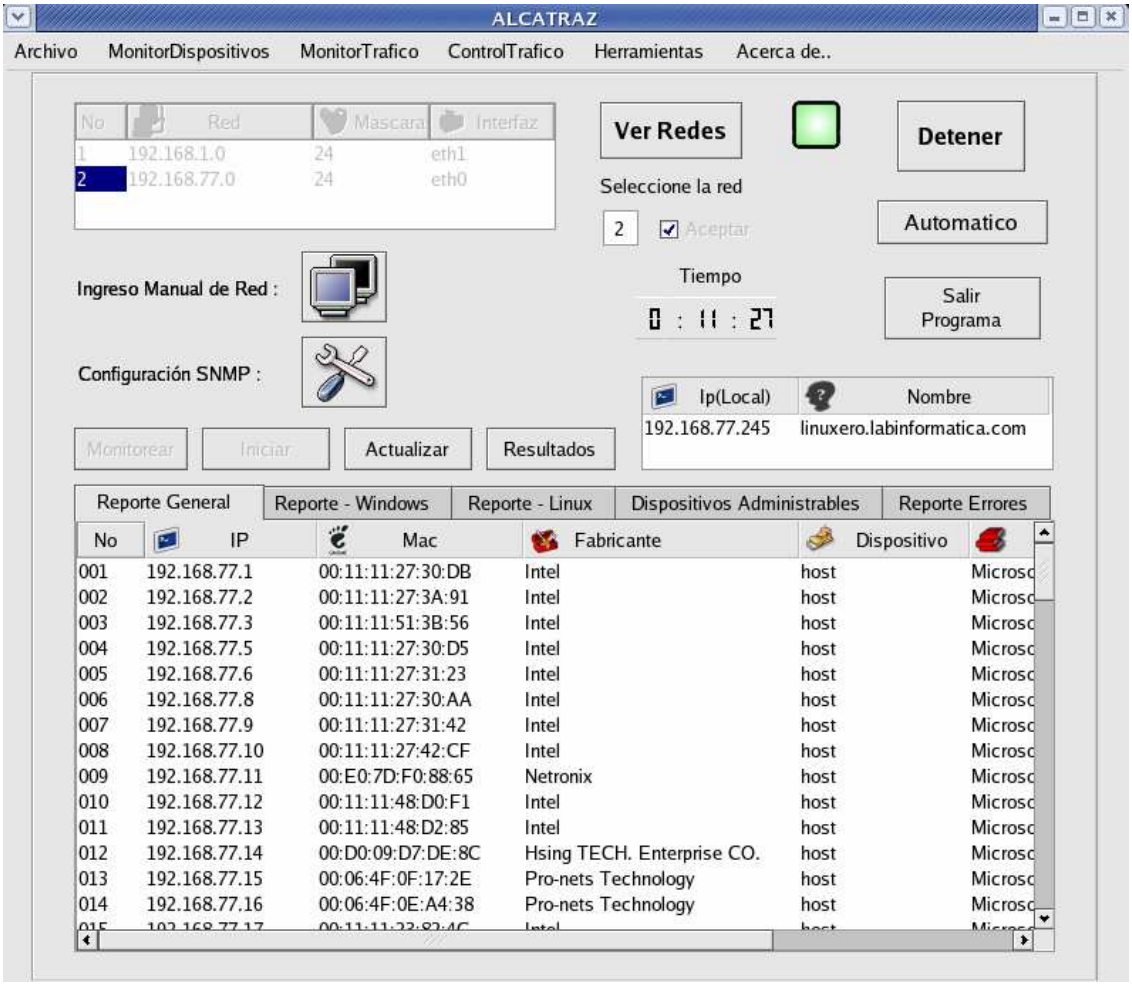


Figura 3.5.- Reporte General.

Los nombres y dominios de los hosts pertenecientes al entorno Windows se presentan en la ventana de Reporte-Windows, como lo muestra la figura 3.6.

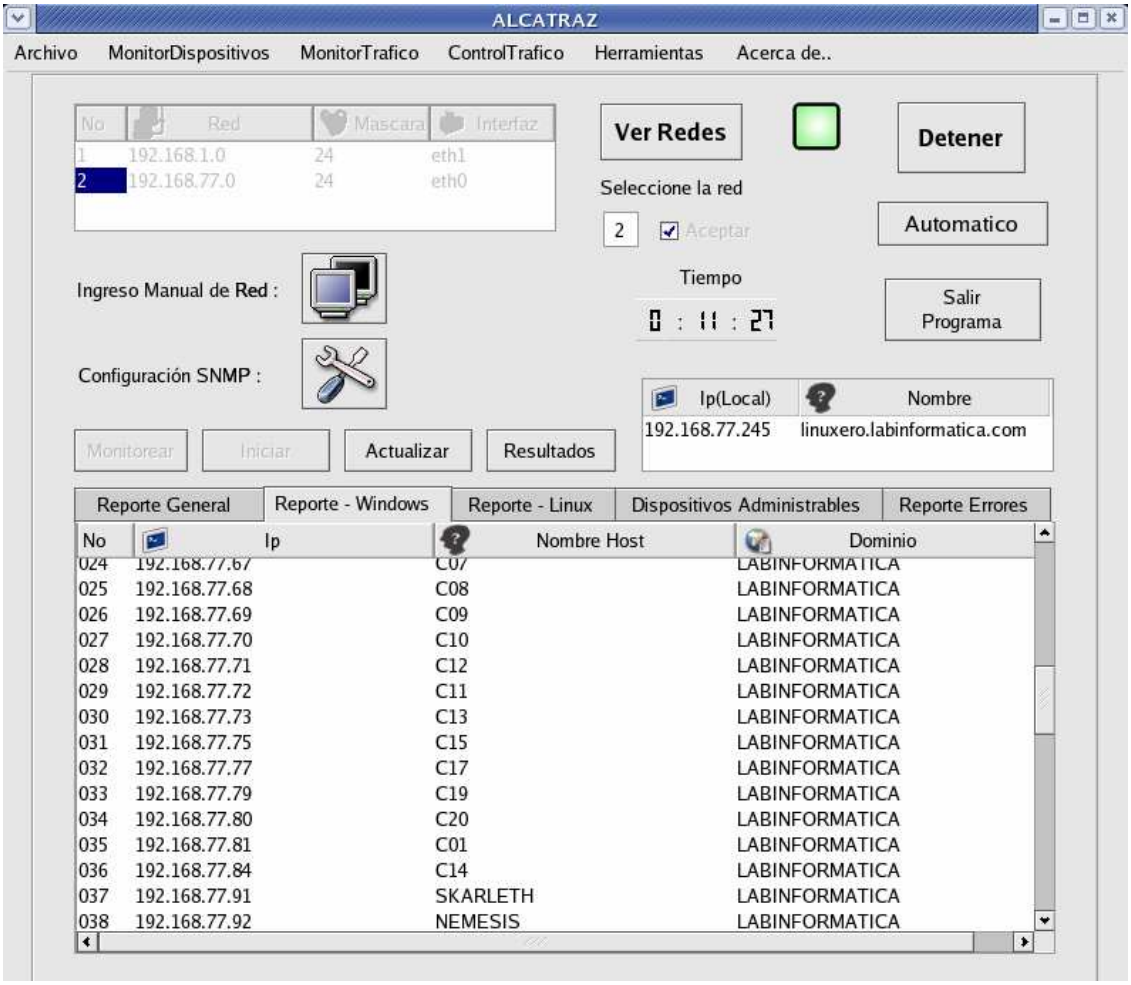


Figura 3.6.- Reporte Entorno Windows.

La ventana de Reporte-LINUX (figura 3.7), además de las direcciones IP de los hosts que funcionan con sistema operativo LINUX, también se muestran los nombres y dominios de los hosts en los cuales el servicio snmpd estuvo activado.

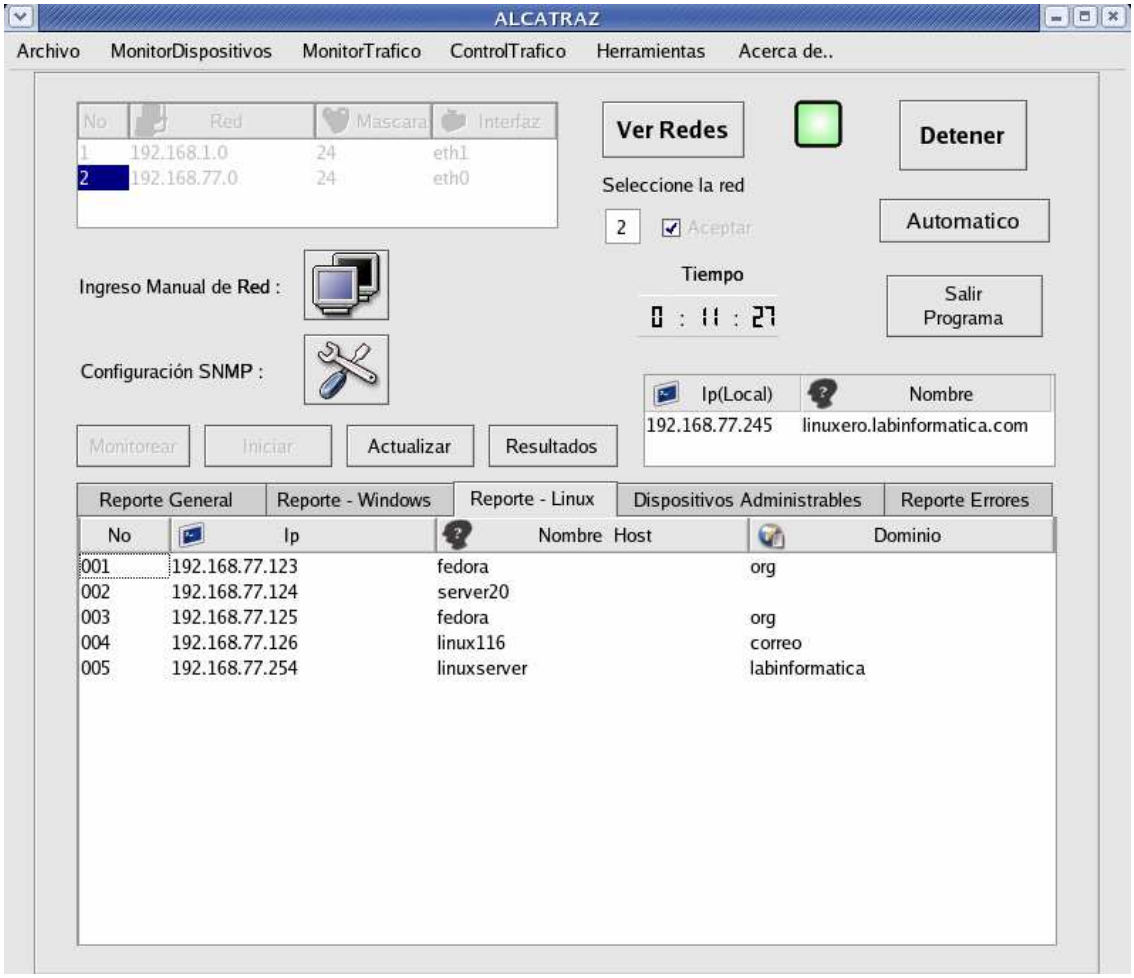


Figura 3.7.- Reporte Entorno LINUX.

La información que contiene el nombre y tipo de los dispositivos administrables se la encuentra en la ventana *Dispositivos Administrables* (figura 3.8).

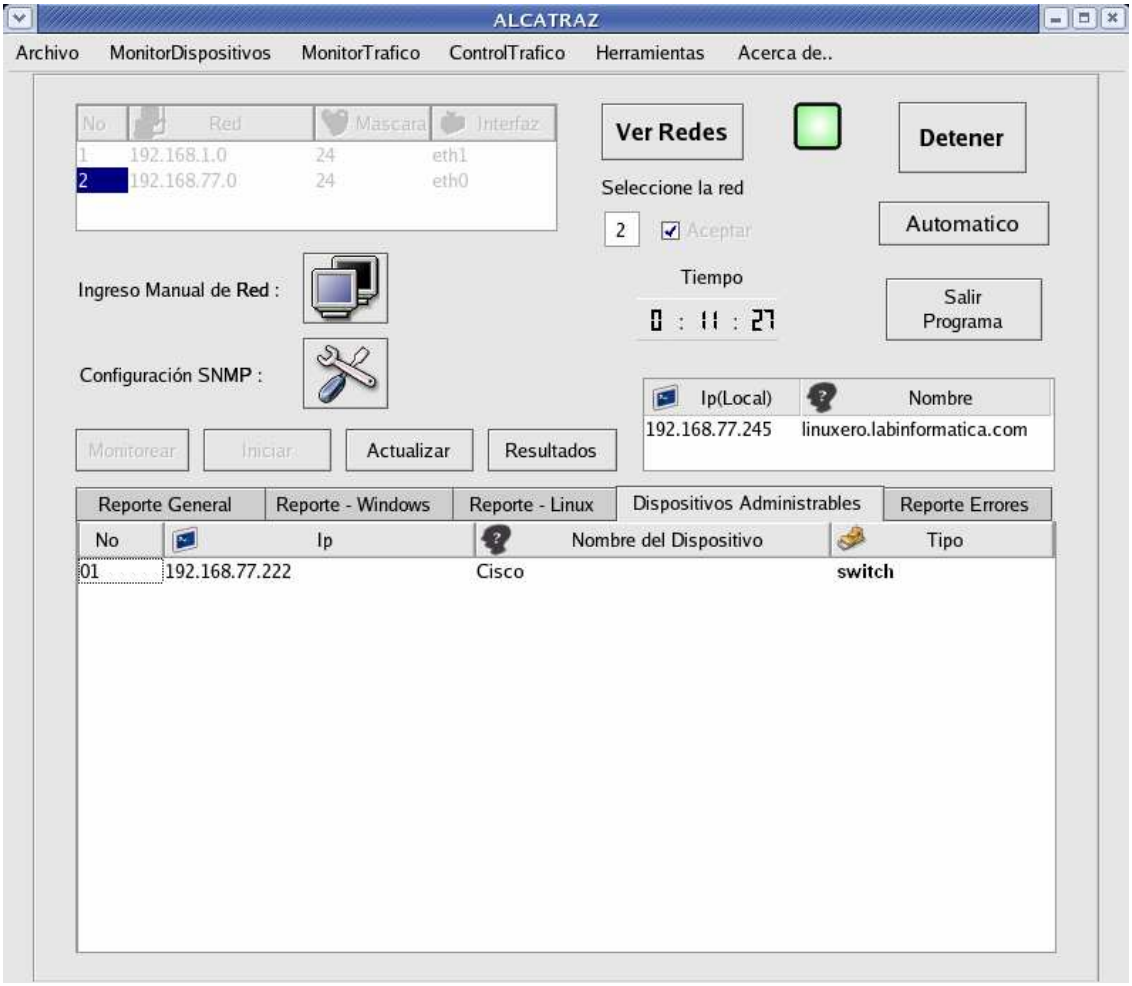


Figura 3.8.- Reporte de Dispositivos Administrables.

En la ventana *Reporte Errores* se listan los dispositivos con sus direcciones IP, nombre, dominio y problema. En la columna problema, se muestran posibles soluciones para las direcciones IP que no respondieron a una nueva prueba de conectividad realizada durante la actualización. En la figura 3.9, se muestra el resultado de una actualización con errores ocurrida en la red LAN A.

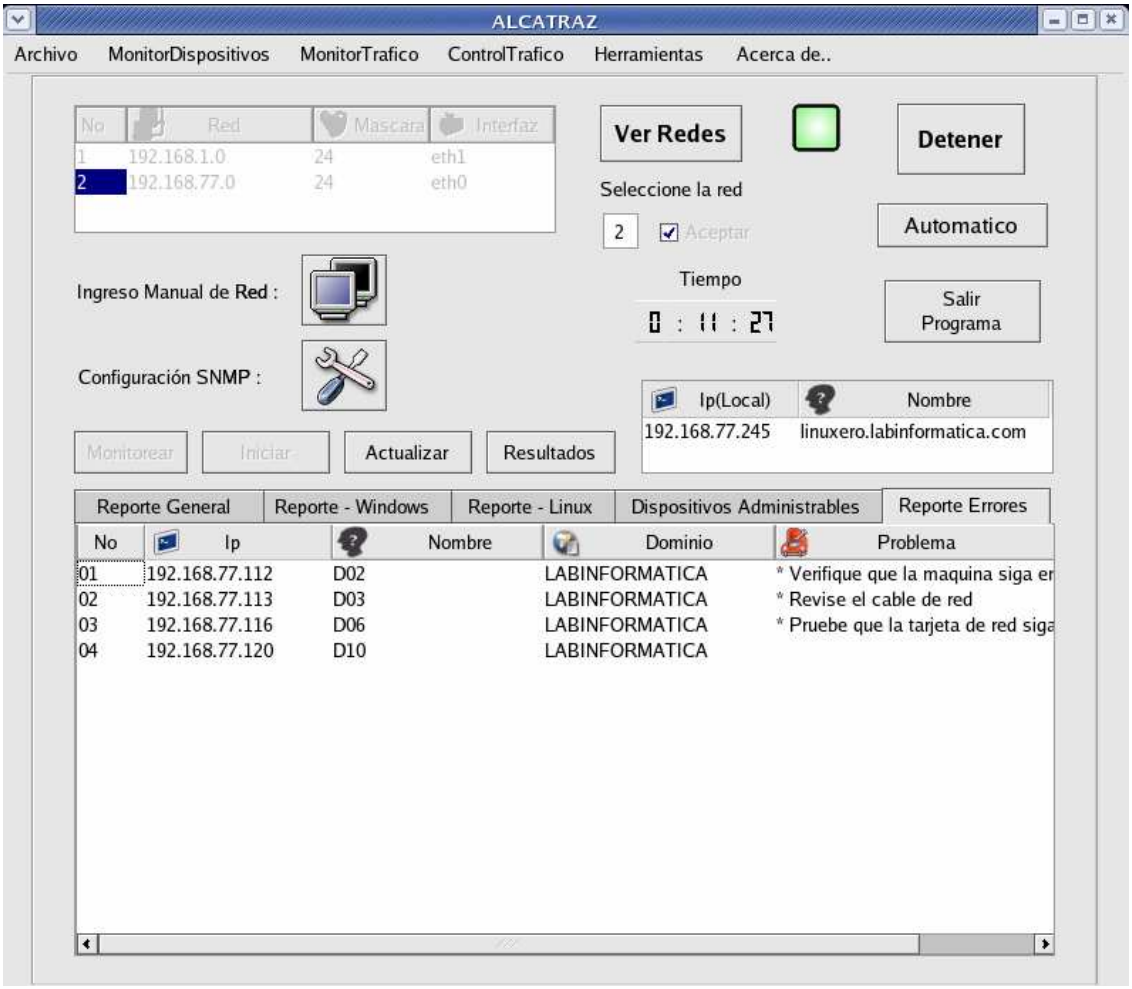


Figura 3.9.- Reporte de Errores.

La prueba del Monitoreo de dispositivos realizada en la red LAN B, indica un ambiente más amplio de dispositivos administrables, en comparación con la red LAN A.

Esta red presenta aproximadamente 126 dispositivos entre hosts y dispositivos administrables. La figura 3.10, muestra un extracto del Reporte General de los dispositivos encontrados.

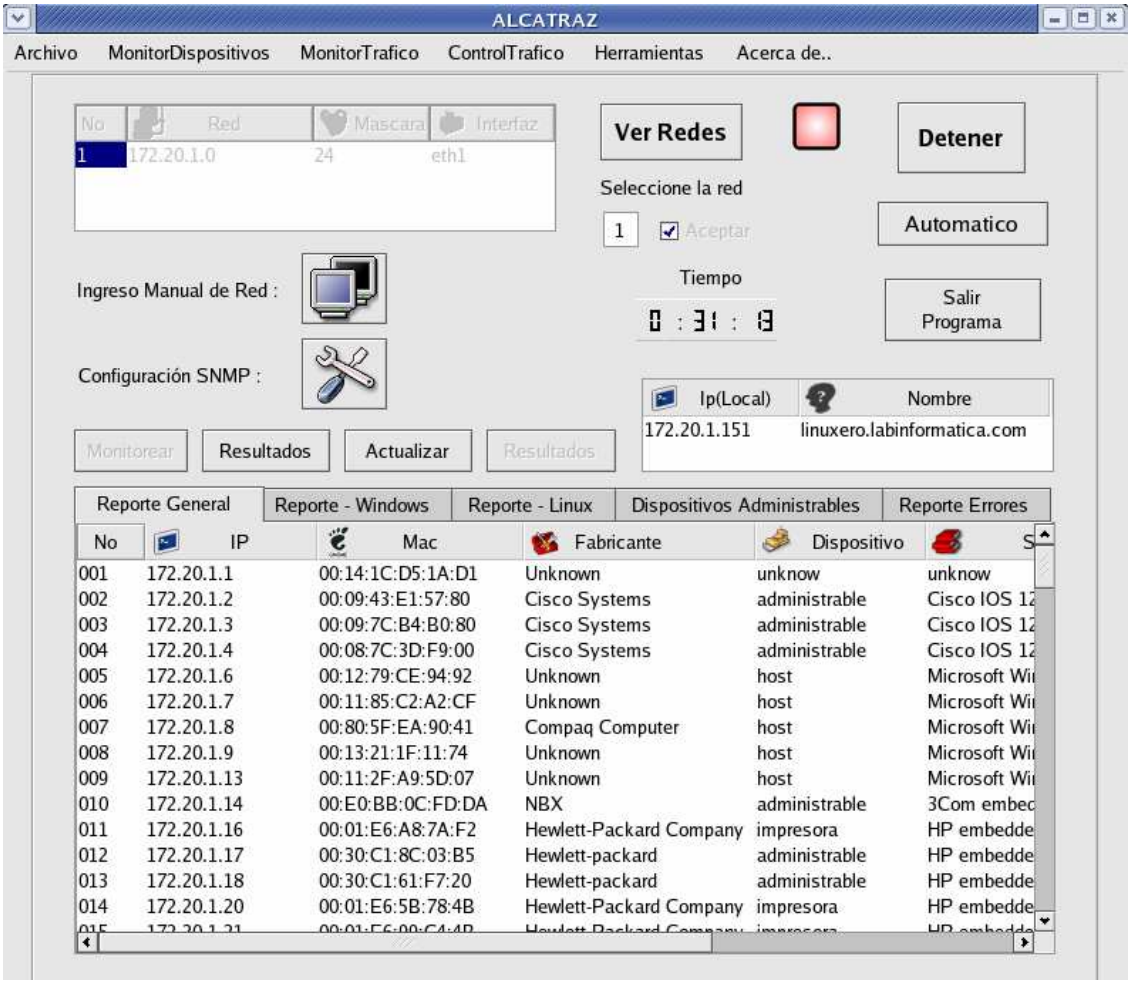


Figura 3.10.- Reporte General de dispositivos encontrados en la LAN B.

En la figura 3.11, se muestra el Reporte de hosts en entorno Windows presentes en la red LAN B.

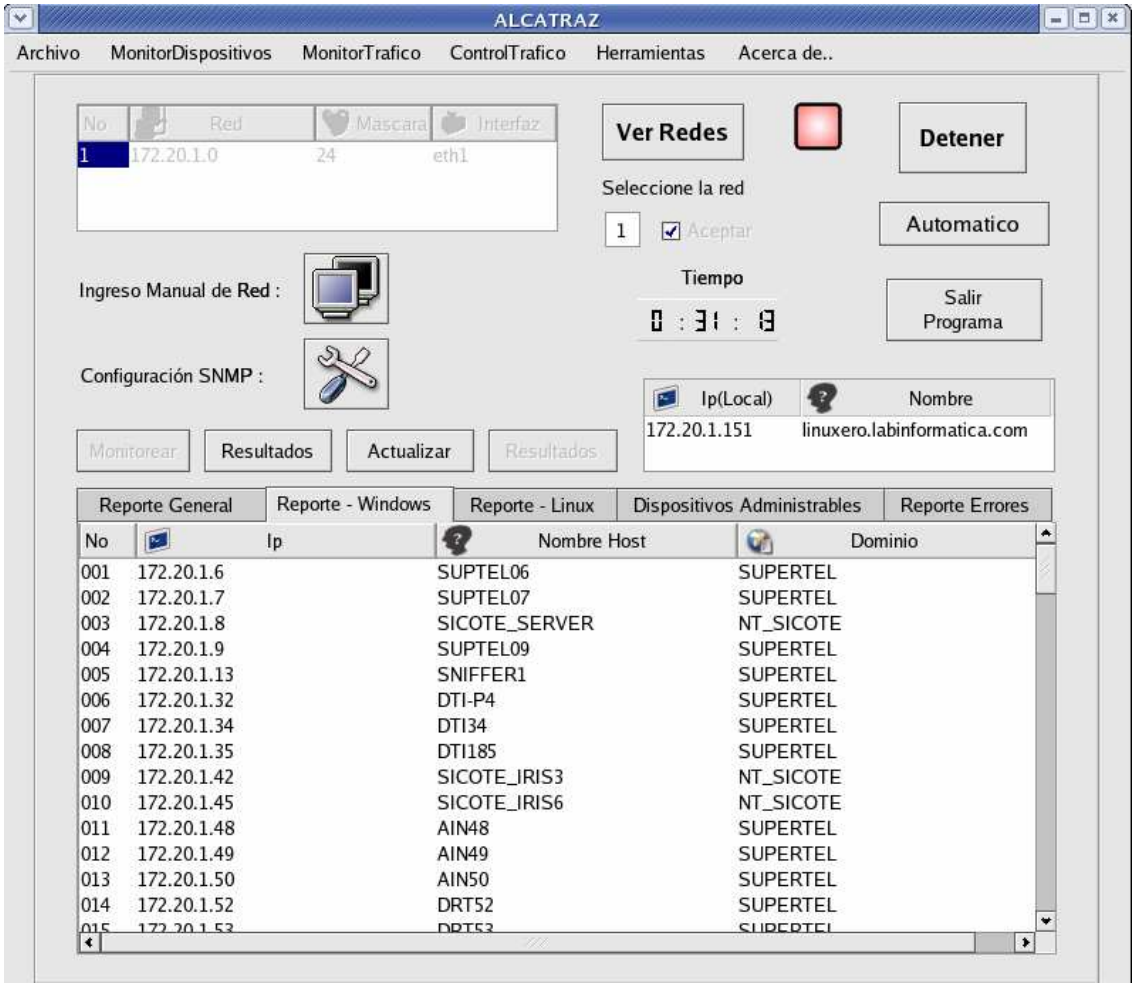


Figura 3.11.- Reporte Hosts Windows LAN B.

En la figura 3.12, se muestra el Reporte de dispositivos administrables presentes en la red LAN B.

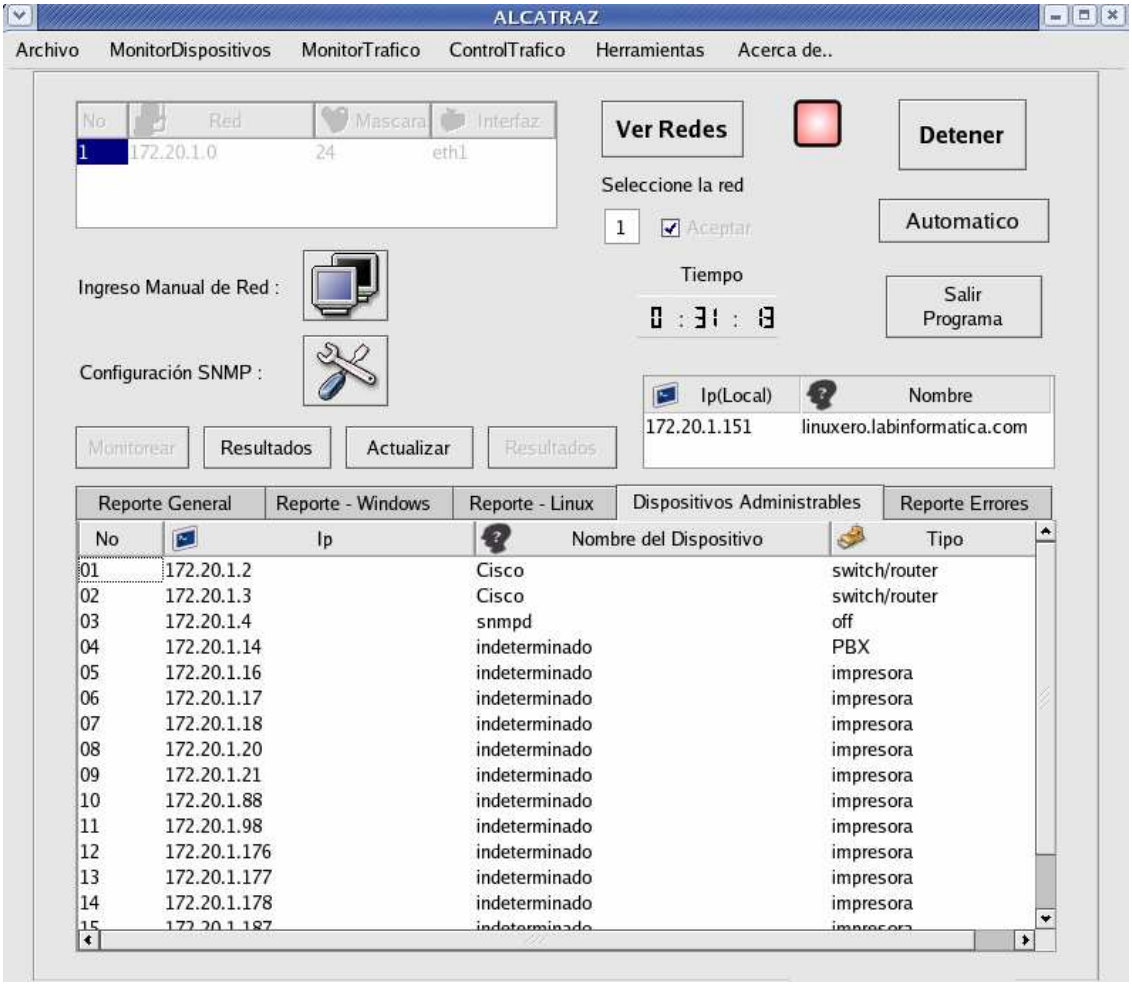


Figura 3.12.- Reporte de Dispositivos Administrables LAN B.

En esta prueba se puede observar que el programa encontró gran variedad de dispositivos administrables como: PBX (Private Branch Exchante), impresoras y dispositivos de capa 2 ó superior tales como switches o ruteadores.

Dispositivos en los cuales su tipo se señala como switch/ruteador se debe a que la información del IOS (Solo dispositivos CISCO) entregada por SNMP, no es suficiente para determinar el tipo al cual corresponde.

Como opción adicional dentro del Monitoreo de dispositivos se presenta la alternativa de automatizar las actividades, como se puede observar en la figura 3.13. Para automatizar se debe ingresar el período con el cual se realizará los siguientes monitoreos. Al hacer click en el botón *Iniciar,* se inicia el monitoreo con las opciones ingresadas anteriormente y se seguirán guardando los reportes en la carpeta correspondiente. Se puede detener la automatización haciendo click en el

botón *Detener*, de lo contrario, si se cierra la aplicación sin detener la automatización se dejarán procesos activos en el sistema.

Figura 3.13.- Automatización del Monitoreo de Dispositivos.

3.2.2 MONITOR DE TRÁFICO

Para verificar las tarjetas o interfaces de red conectadas al host, se hace click en *Sondear Tarjetas Activas* lo que desplegará en una lista los resultados de ésta acción y de dicha lista además, se podrá seleccionar una de las interfaces (si hay más de una). Antes de escoger el tipo de monitor de tráfico (protocolos o puertos), existe la posibilidad de elegir capturar el tráfico de entrada y salida del host local, luego, si se selecciona la opción de monitoreo de tráfico por puertos, en la ventana se activará la opción de utilización o no de proxy transparente, dentro de la cual si se escoge utilizarlo, en el programa aparece activo el botón *Iniciar,* desde el cual se puede ejecutar la captura de paquetes. Cuando se escoge no utilizar proxy transparente, el usuario tiene que ingresar el número del puerto para iniciar la captura de paquetes.

Al elegir el tipo de monitor de tráfico por protocolos, no hace falta configuraciones

adicionales y en la ventana aparece activo el botón *Iniciar*, como se muestra en la figura 3.14.

Figura 3.14.- Opciones de configuración previa ejecución del Monitor de Tráfico.

Durante la captura se puede observar dentro de la aplicación un reloj que muestra el tiempo durante el cual se está ejecutando el monitoreo, también aparece el botón *Detener* que termina la captura y en forma secuencial detiene todos los procesos para que al final se puedan observar los resultados capturados durante el monitoreo, como se muestra en la figura 3.15 y 3.16.

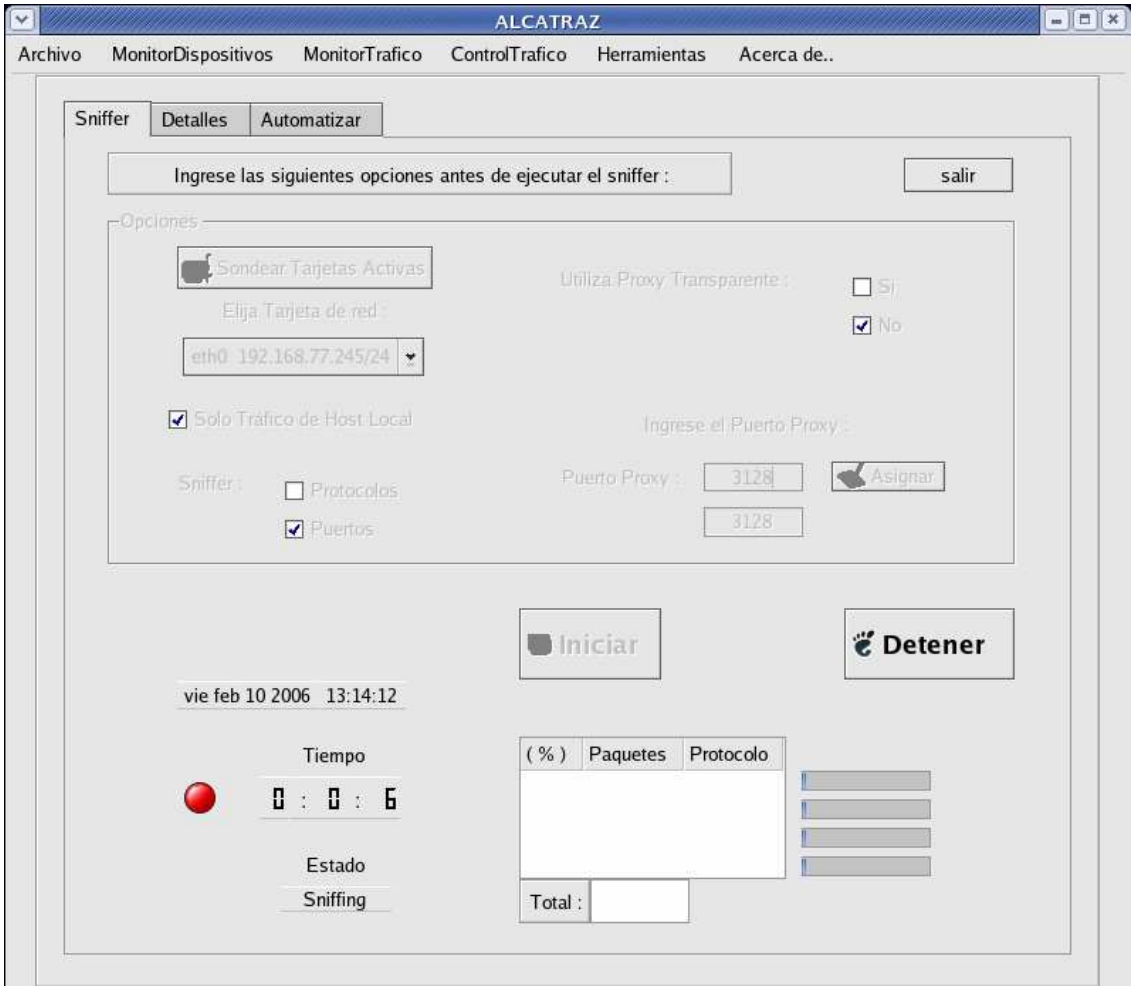


Figura 3.15.- Ejecución Monitoreo de Tráfico.

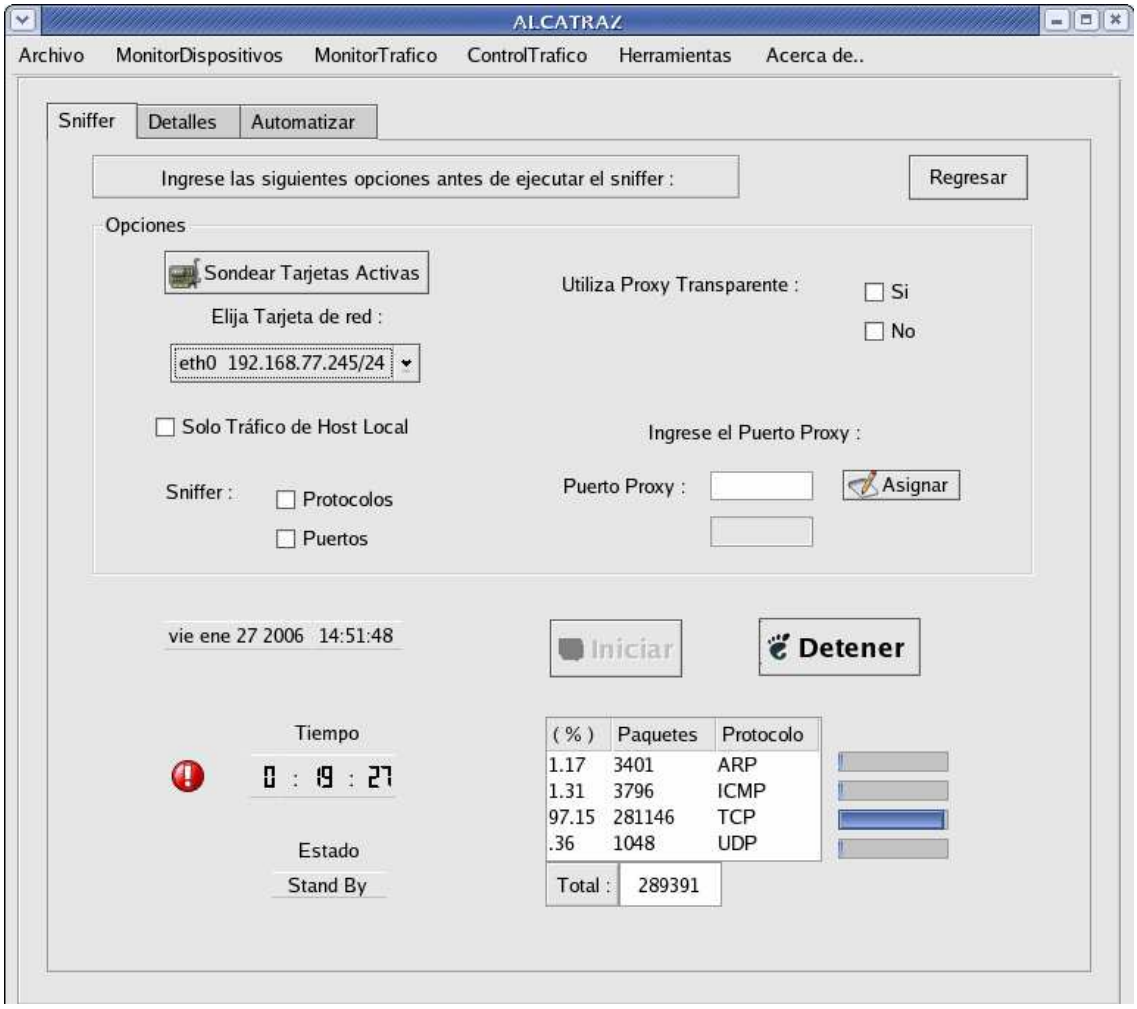


Figura 3.16.- Resultados desplegados en el Monitoreo de Tráfico.

En la prueba que se realizó en la red LAN A se obtuvieron resultados que se describen a continuación.

La figura 3.16 muestra la estadística general del monitoreo de tráfico por protocolos, el mismo que duró aproximadamente 19 minutos y en el que se capturó un total de 289391 paquetes, que en su mayoría corresponden al protocolo TCP debido a que sobre este protocolo, trabajan aplicaciones como HTTP, FTP, SMTP y otros que en el momento del monitoreo pudieron haber estado presentes. Los datagramas correspondientes al protocolo UDP son en porcentaje bajo, debido a que hay pocos paquetes NetBIOS o uso de DNS. El resultado de información ARP se debe a la presencia de paquetes de pregunta y respuesta de direcciones MAC entre dispositivos dentro de la red LAN. Finalmente, respecto a la estadística del protocolo ICMP, se debe en gran parte a

paquetes de preguntas y respuestas de eco entre los dispositivos presentes en la red.

El detalle de los paquetes capturados se puede observar dentro del monitor de tráfico en la ventana *Detalles*, como se muestra en la figura 3.17.

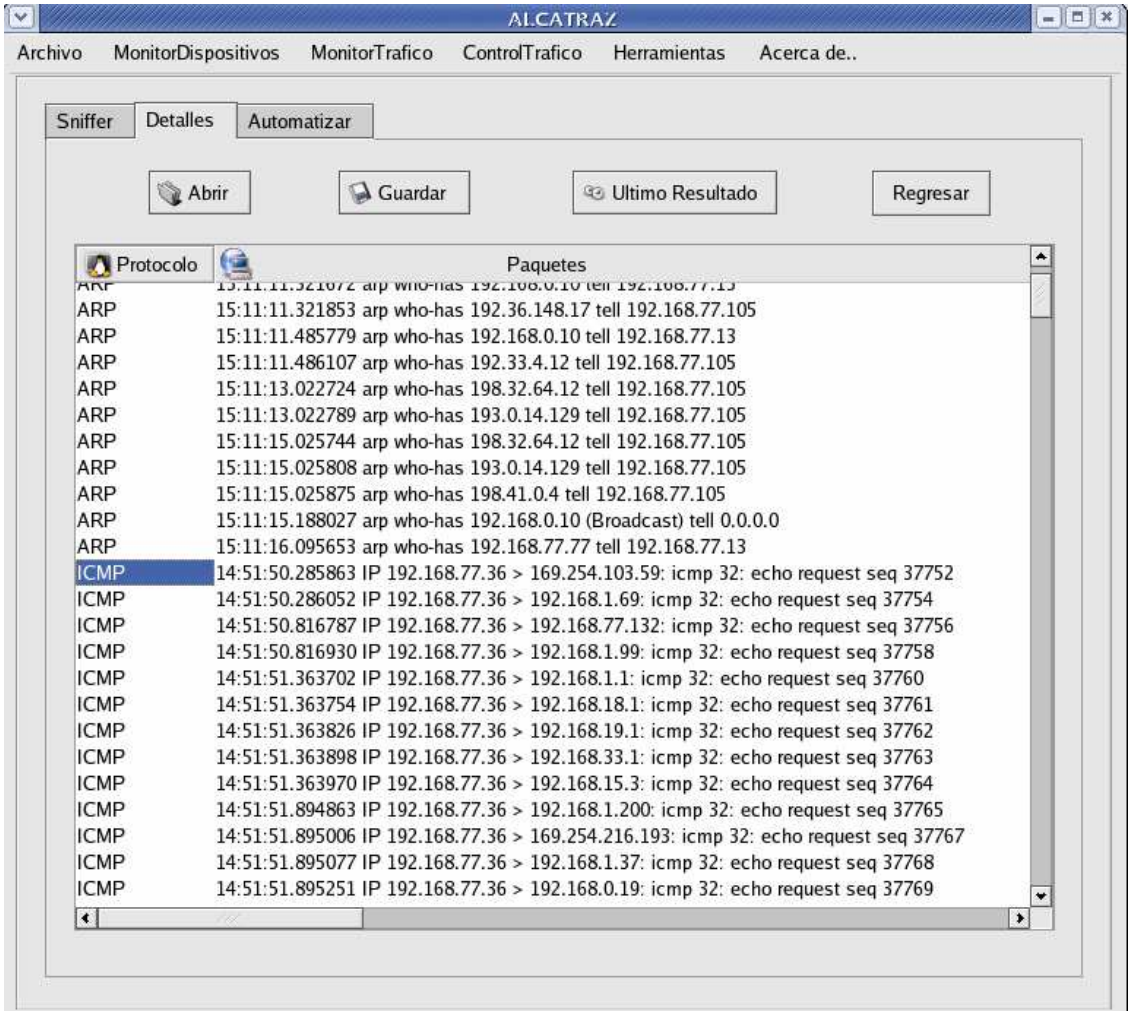


Figura 3.17.- Detalle de paquetes ARP e ICMP capturados en la red LAN A.

A continuación se toma como referencia los resultados de la figura 3.17 para interpretar dos de los paquetes más comunes presentes dentro del protocolo ARP:

arp who-has 192.168.77.245 tell 192.168.77.222

- ***Interpretación:*** El dispositivo con dirección IP 192.168.77.222, realiza una petición tipo brodcast de la dirección MAC perteneciente a la dirección IP 192.168.77.245.

arp reply 192.168.77.245 is-at 00:03:47:ef:a1:69

- ***Interpretación:*** El dispositivo con dirección IP 192.168.77.245, responde entregando su dirección MAC: 00:03:47:ef:a1:69.

Para la interpretación de los paquetes ICMP de la figura 3.16, se toman las siguientes muestras:

IP 192.168.77.254 > 192.168.77.245: icmp 80: 192.168.77.254 udp port domain unreachable

- ***Interpretación:*** El dispositivo con dirección IP 192.168.77.254, envía un mensaje de error, que en este caso es de puerto UDP inalcanzable, a la dirección IP 192.168.77.245.

IP 192.168.77.13 > 192.168.77.132: icmp 32: echo request seq 32651

- ***Interpretación:*** El dispositivo con dirección IP 192.168.77.13, realiza una petición de eco a la dirección IP 192.168.77.132.

En la figura 3.18 se observa otra parte del detalle de los paquetes capturados en la prueba de monitoreo por protocolos en la red LAN A.

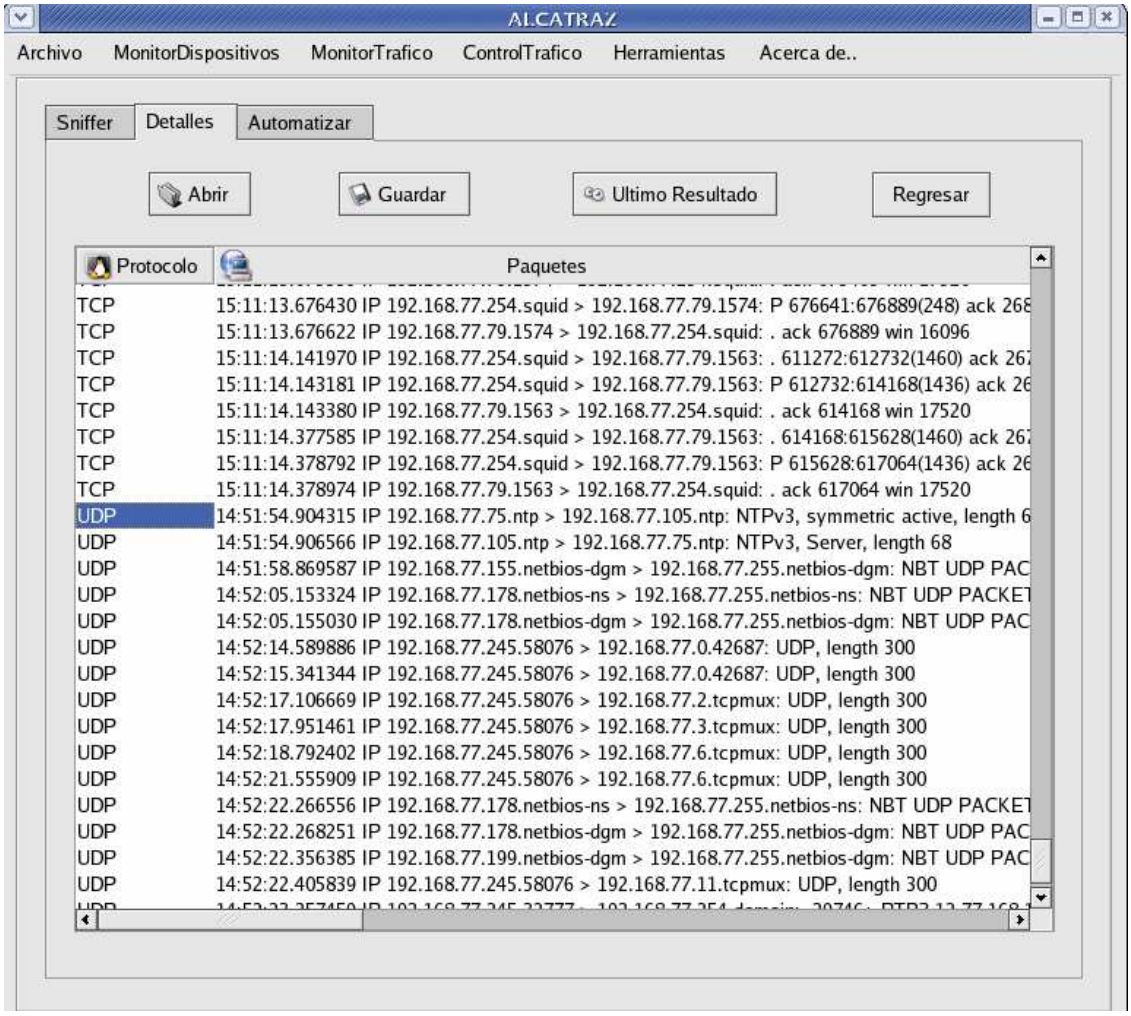


Figura 3.18.- Detalle de paquetes TCP y UDP capturados en la red LAN A.

Para la interpretación de los paquetes pertenecientes a TCP de la figura 3.18, se pueden tomar las siguientes muestras:

IP 192.168.77.245.58077 > 192.168.77.0.635: S 930368803:930368803(0) win 4096

- ***Interpretación:*** El puerto origen 58077 del host con dirección IP 192.168.77.245, envía un paquete tipo SYN para verificar actividad en el puerto 635 del host con dirección IP 192.168.77.0, además no contiene

datos (930368803: 930368803 (0)) e indica que la disponibilidad de la ventana es de 4096 bytes.

IP 192.168.77.254.squid > 192.168.77.79.1549: P 1448:2896(1448) ack 1 win 6432
IP 192.168.77.79.1549 > 192.168.77.254.squid: . ack 2896 win 17520

- ***Interpretación:*** El puerto squid del host con dirección IP 192.168.77.254, envía un paquete tipo PUSH para poner datos en el puerto 1549 del host con dirección IP 192.168.77.79, el tamaño de los datos fue de 1448 bytes. En el segundo paquete, se muestra la respuesta que la dirección IP 192.168.77.79, realizó a través del puerto 1549.

Finalmente, para la interpretación de los paquetes pertenecientes al protocolo UDP, de la figura 3.18 se pueden tomar las siguientes muestras:

IP 192.168.77.75.ntp > 192.168.77.105.ntp: NTPv3, symmetric active, length 68

- ***Interpretación:*** El puerto NTP (network time protocol) del host con dirección IP 192.168.77.75, envía un datagrama UDP al puerto NTP del host con dirección IP 192.168.77.105, con longitud de datos de 68 bytes.

IP 192.168.77.245.32783 > 192.168.77.101.snmp: GetNextRequest(28) .1.3.6.1.2.1.1.5.0

- ***Interpretación:*** El puerto origen 32783 del host con dirección IP 192.168.77.245, envía una pregunta al puerto snmp del host con dirección IP 192.168.77.101, con datos de 28 bytes de tamaño.

IP 192.168.77.245.32777 > 192.168.77.254.domain: 29746+ PTR? 12.77.168.192.in-addr.arpa. (44)

- ***Interpretación:*** El puerto 32777 del host con dirección IP 192.168.77.245, envía una pregunta al servidor de dominio de nombres por la dirección IP 192.168.77.254, con datos de 44 bytes de tamaño.

En la prueba realizada en la red LAN B, se puede observar los resultados del monitoreo de tráfico por tipo de protocolos de capa aplicación, a través del monitoreo de puertos.

A continuación se detalla la prueba realizada:

En la figura 3.19 se muestra que durante el monitoreo, se capturó un total de 521 paquetes en alrededor de ocho minutos.

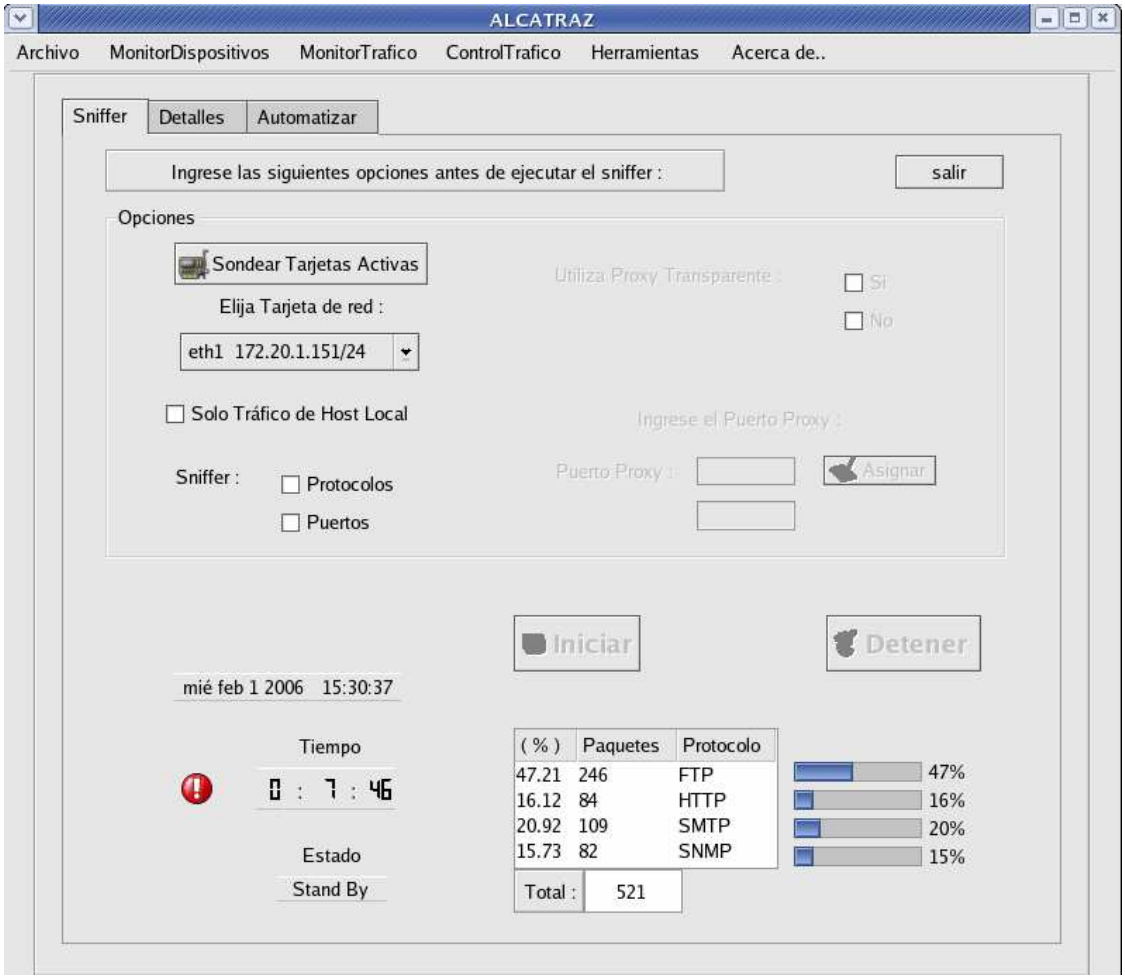


Figura 3.19.- Estadística de paquetes HTTP, FTP, SMTP y SNMP capturados en la red LAN B.

En las figuras 3.20, 3.21 y 3.22, se muestran partes del detalle de paquetes capturados durante el monitoreo en la red LAN B.

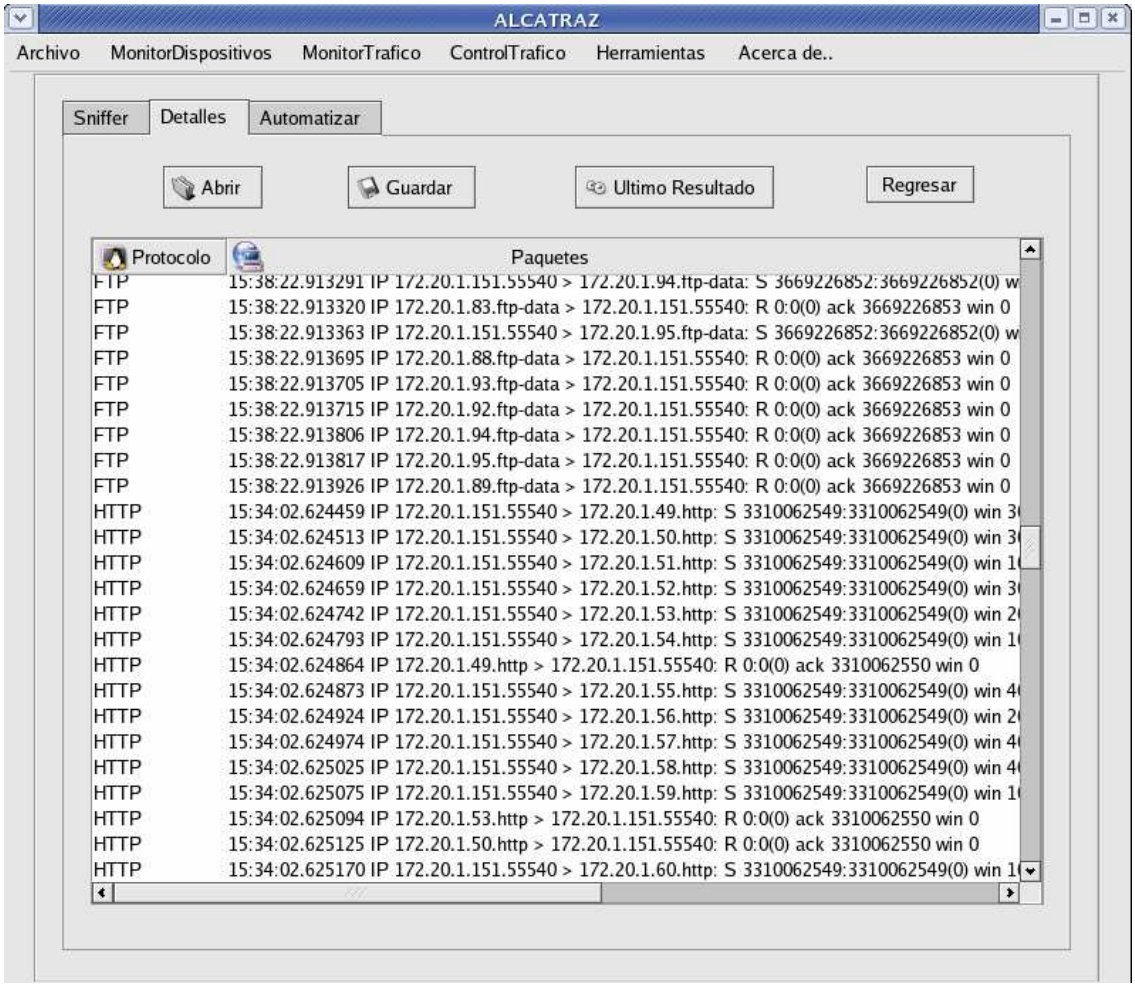


Figura 3.20.- Detalle de paquetes FTP y HTTP capturados en la red LAN B.

Para la interpretación de paquetes del protocolo FTP de la figura 3.20, se han tomado como muestra los siguientes ejemplos:

IP 172.20.1.151.55540 > 172.20.1.57.ftp: S 3310062549:3310062549(0) win 1024
IP 172.20.1.57.ftp > 172.20.1.151.55540: R 0:0(0) ack 3310062550 win 0

- ***Interpretación:*** El puerto 55540 del host con dirección IP 172.20.1.151, envía un paquete sin datos y del tipo SYN al puerto ftp del host con dirección IP 172.20.1.57, para verificar el estado de dicho puerto. El

tamaño de la ventana fue de 1024 bytes. El host con dirección IP 172.20.1.57, responde con un paquete TCP tipo R (Reset) y ventana de 0 bytes, con lo cual dice que no tiene ningún proceso activo en ese puerto.

IP 172.20.1.151.55540 > 172.20.1.58.ftp-data: S 3310062549:3310062549(0) win 3072
IP 172.20.1.58.ftp-data > 172.20.1.151.55540: R 0:0(0) ack 3310062550 win 0

- ***Interpretación:*** Aquí, se presenta el mismo caso de la interpretación anterior, es decir, el tipo de intercambio o conversación es la misma, pero la diferencia está en que el puerto de comunicación es el ftp-data.

Para la interpretación de los paquetes HTTP mostrados en la figura 3.21 se pueden extraer los siguientes ejemplos:

IP 172.20.1.151.55540 > 172.20.1.49.http: S 3310062549:3310062549(0) win 3072
IP 172.20.1.49.http > 172.20.1.151.55540: R 0:0(0) ack 3310062550 win 0

- ***Interpretación:*** El puerto 55540 del host con dirección IP 172.20.1.151 envía un paquete sin datos y del tipo SYN al puerto http del host con dirección IP 172.20.1.49 para verificar el estado de dicho puerto. El tamaño de la ventana fue de 1024 bytes. El host con dirección IP 172.20.1.49 responde con un paquete TCP tipo R (Reset) y ventana de 0 bytes, con lo cual dice que no tiene ningún proceso activo en ese puerto.

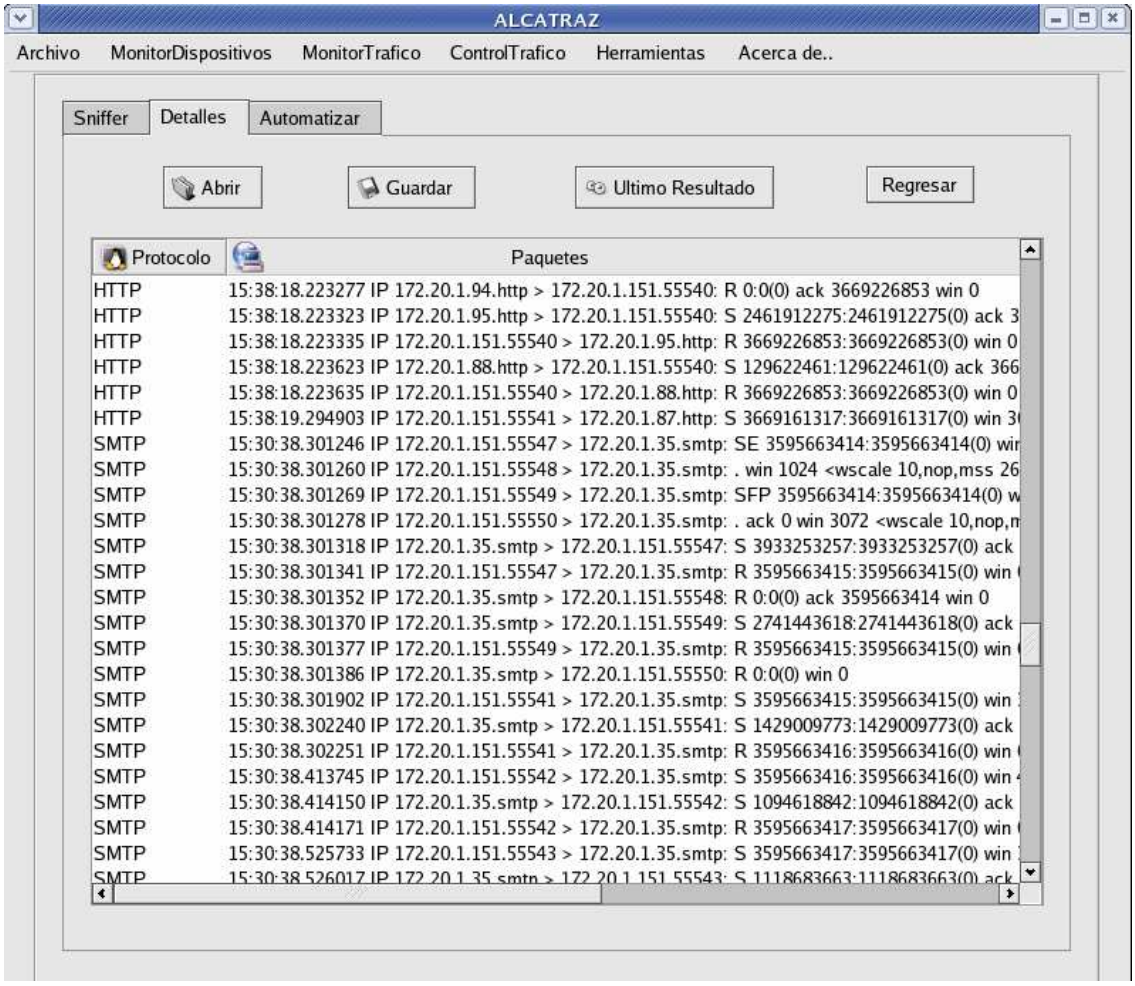


Figura 3.21.- Detalle de paquetes HTTP y SMTP capturados en la red LAN B.

Para la interpretación de los paquetes SMTP capturados, de la figura 3.21 se extraen los siguientes ejemplos:

IP 172.20.1.95.smtp > 172.20.1.151.55540: S 2461954120:2461954120(0) ack 3669226853 win 16616 <mss 1460>
IP 172.20.1.151.55540 > 172.20.1.95.smtp: R 3669226853:3669226853(0) win 0

- ***Interpretación:*** El puerto smtp del host con dirección IP 172.20.1.95, envía un paquete sin datos y del tipo SYN al puerto 55540 del host con dirección IP 172.20.1.151, para verificar el estado de dicho puerto. El tamaño de la ventana fue de 16616 bytes y tamaño máximo por segmento de datos de 1460 bytes. El host con dirección IP 172.20.1.151, responde con un

paquete TCP tipo R (Reset) y ventana de 0 bytes, con lo cual dice que no tiene ningún proceso activo en ese puerto.

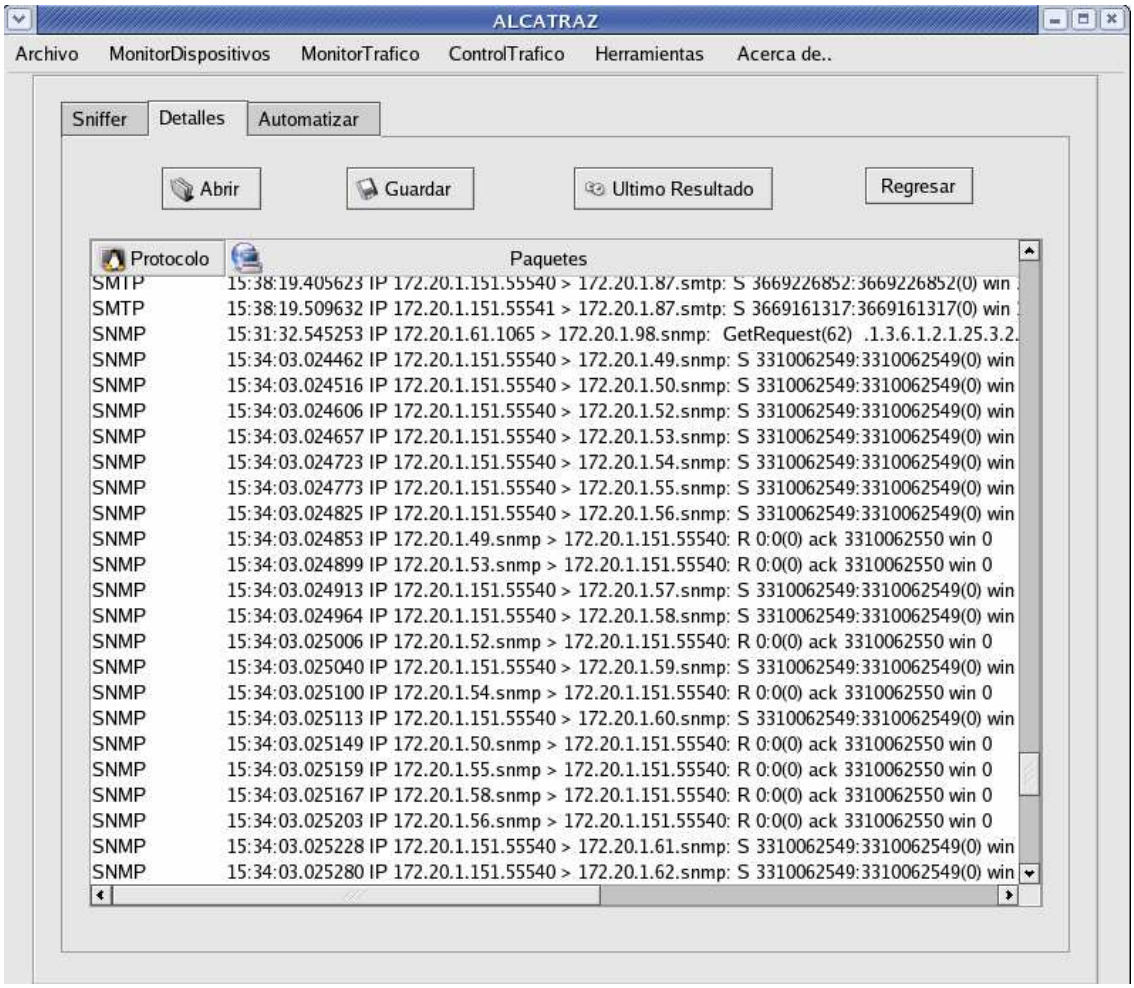


Figura 3.22.- Detalle de paquetes SNMP capturados en la red LAN B.

Finalmente para la interpretación de paquetes del protocolo SNMP de la figura 3.22 se toman los siguientes ejemplos:

IP 172.20.1.55.1089 > 172.20.1.98.snmp: GetRequest(62) .1.3.6.1.2.1.25.3.2.1.5.1 .1.3.6.1.2.1.25.3.5.1.1.1 .1.3.6.1.2.1.25.3.5.1.2.1

- ***Interpretación:*** El puerto 1089 del host con dirección IP 172.20.1.55, envía un paquete de 62 bytes al puerto snmp de la dirección IP 172.20.1.98, en el cual, realiza tres preguntas utilizando números OID al árbol de MIBs.

IP 172.20.1.151.55540 > 172.20.1.49.snmp: S 3310062549:3310062549(0) win 1024
IP 172.20.1.49.snmp > 172.20.1.151.55540: R 0:0(0) ack 3310062550 win 0

- ***Interpretación:*** El puerto 55540 del host con dirección IP 172.20.1.151 envía un paquete sin datos y del tipo SYN al puerto snmp del host con dirección IP 172.20.1.49 para verificar el estado de dicho puerto El tamaño de ventana fue de 1024 bytes. El host con dirección IP 172.20.1.49 responde con un paquete TCP tipo R (Reset) y ventana de 0 bytes con lo cual dice que no tiene ningún proceso activo en ese puerto.

Como opción adicional dentro del Monitoreo de Tráfico, se presenta la alternativa de automatizar las actividades pero habiendo antes configurado las opciones de la ventana *Sniffer*, como se puede observar en la figura 3.23.
Los requisitos para la automatización son: Ingreso del intervalo después del cual se realizará la siguiente monitoreo y también el tiempo de duración de cada captura. Una vez que se cumplan los requisitos, al hacer click en el botón *Automatizar* se puede programar los siguientes monitoreos y sólo se detendrá desde ésta misma ventana haciendo clic en el botón *Detener*, de lo contrario si se cierra la aplicación sin realizar este paso, se dejarán procesos activos en el sistema.

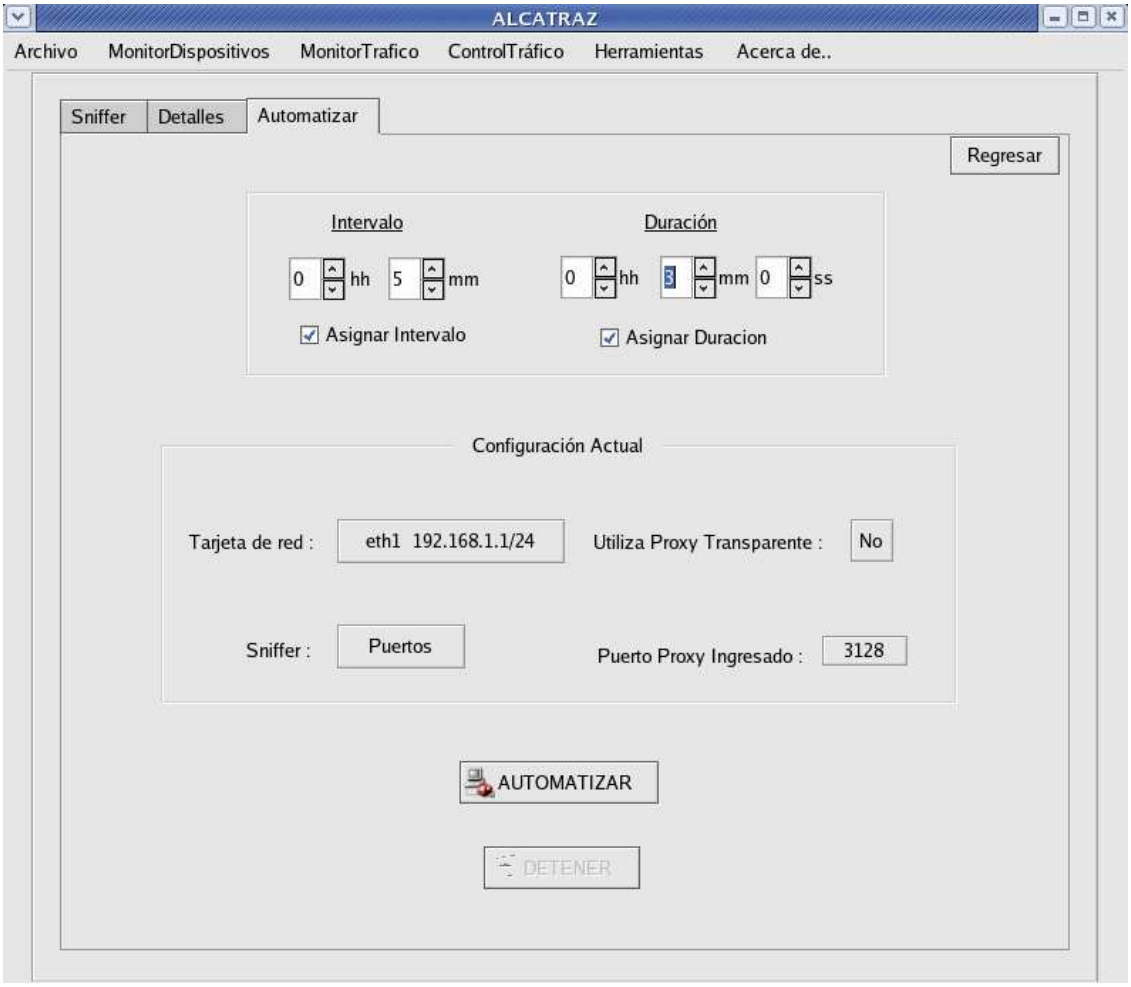


Figura 3.23.- Ventana para Automatizar el Monitoreo de Tráfico.

3.2.3 ADMINISTRACIÓN DEL TRÁFICO DE DATOS

Para verificar las tarjetas o interfaces de red conectadas al host, se hace click en *Tarjetas Activas,* lo que desplegará en una lista los resultados de ésta acción y de dicha lista además se podrá seleccionar una de las interfaces (si hay más de una).

Luego, se debe escoger de una la lista el valor y unidad del ancho de banda total disponible en la interfaz. Esto se lo puede observar en la figura 3.24.

Figura 3.24.- Sección principal para ejecución del Control de Tráfico.

En la figura 3.24, también se observa que se puede ingresar el puerto proxy, útil en los casos donde el administrador de red conozca que las peticiones para el protocolo HTTP se las hace a través de un puerto distinto al puerto 80.

Luego, se habilita un cuadro de diálogo, donde se pueden escoger los protocolos que se desea filtrar y dentro de los cuales se permite ingresar el ancho de banda parcial con la respectiva unidad (Kbps o Mbps), como lo muestra la figura 3.25.

Además en ésta sección, se presenta la opción de editar el ancho de banda asignado para el tráfico distinto al seleccionado en lista de protocolos, el cual no podrá ser menor a 1Kbps.

Figura 3.25.- Sección de ingreso de parciales de ancho de banda para el Control de Tráfico.

Luego, de haber ingresado los valores de ancho de banda, se debe escoger a qué direcciones IP se van a aplicar los filtros (figura 3.26), en donde si se elige la opción *Toda la red,* se aplicarán los filtros a todas las direcciones IP que hayan sido detectadas durante el monitoreo de dispositivos realizado para esa red. Para el caso en el que se elija ingresar un rango de direcciones, éstas serán tomadas desde el mismo conjunto anterior de direcciones IP.
Finalmente, existe la posibilidad de filtrar el tráfico de una sola dirección IP que también deberá estar presente en el conjunto de direcciones IP monitoreadas.

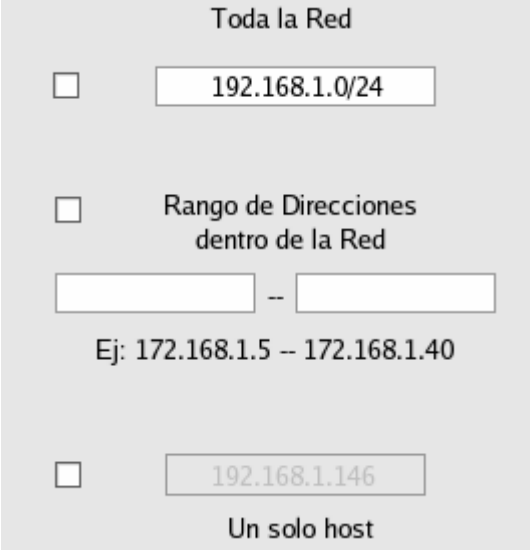


Figura 3.26.- Sección ingreso de direcciones IP para el Control de Tráfico.

Las pruebas de Administración de tráfico de datos, se muestran en detalle a partir de la figura 3.27, en donde, para una demostración visual de la aplicación de los filtros a una sola dirección IP, se utilizó el programa PRTG (Paessler Router Traffic Grapher), el cual es software para Windows y se lo utiliza para el monitoreo y clasificación del uso de ancho de banda, éste provee de una interfaz gráfica para la lectura del monitoreo del uso de ancho de banda, en el presente proyecto se utilizó para monitorear a través del protocolo SNMP los puertos del switch al que se conectan los hosts (servidor y clientes) y obtener en modo gráfico, el tráfico que cursa en cada uno de los puertos.

En la figura 3.27 se muestra la conexión de la LAN implementada para la prueba.

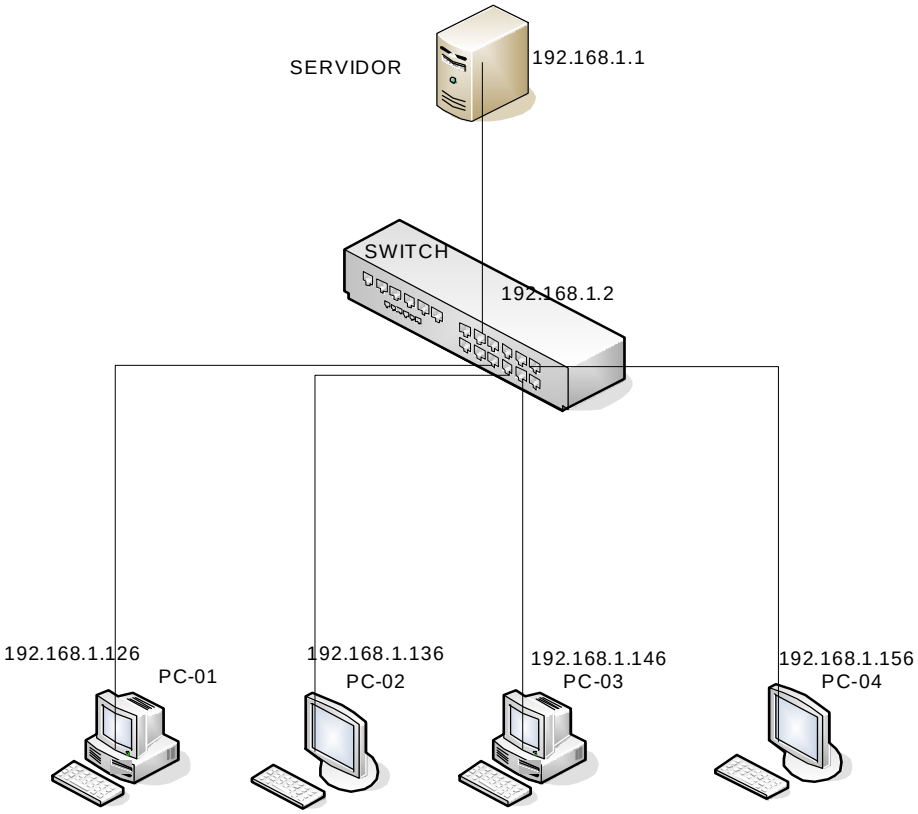


Figura 3.27.- LAN implementada para la Administración de Tráfico de datos.

Las conexiones realizadas para la implementación de la LAN fueron de la siguiente manera:

- En el puerto FastEthernet 0/1 se conectó el servidor,
- Al puerto FastEthernet 0/2 se conectó la PC-01,
- Al puerto FastEthernet 0/3 se conectó la PC-02,
- Al puerto FastEthernet 0/4 se conectó la PC-04 y finalmente
- Al puerto FastEthernet 0/6 se conectó el host al cual se le aplicó el filtrado con el software creado y el monitoreo con la ayuda del programa PRTG.

Las direcciones IP fueron establecidas de la forma que se muestra en la figura 3.27; para el interés de la prueba los puertos monitoreados fueron el FastEthernet 0/1, ya que en él está conectado el servidor y el puerto FastEthernet 0/6, donde está la PC-03 cuya dirección IP se seleccionó para el filtrado.

En la figura 3.28, se muestra una primera prueba con las siguientes asignaciones:

- Ancho de banda total de 100 Mbps,
- Filtro para protocolo FTP con ancho de banda de 20 Mbps, y
- Ancho de banda para tráfico no clasificado de 1 Kbps.

Figura 3.28.- Opciones ingresadas con filtro FTP para un solo host.

En la figura 3.29, se muestra el gráfico entregado por PRTG del resultado del filtro FTP a 20 Mbps.

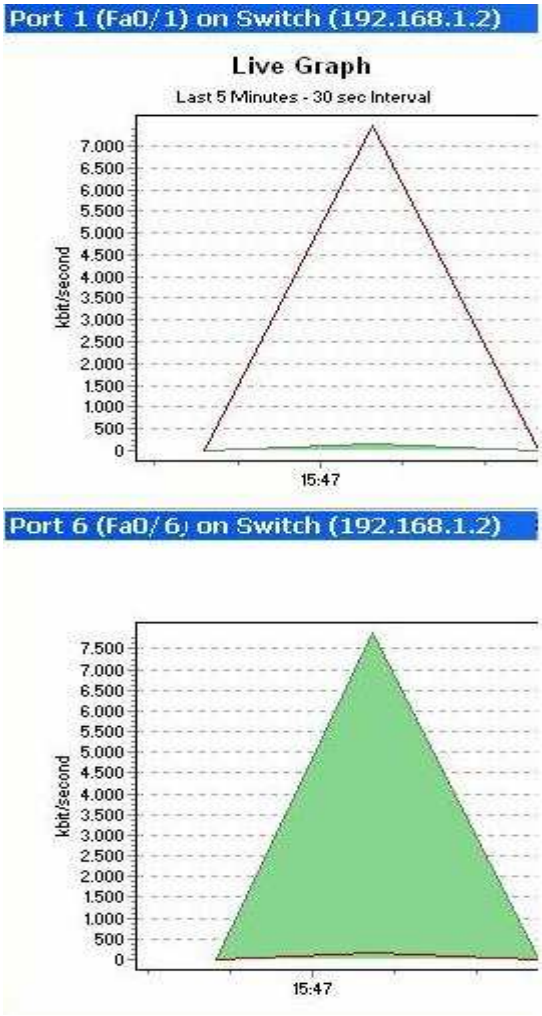


Figura 3.29.- Resultado de filtro FTP a 20 Mbps mostrado por PRTG.

A continuación se muestra los resultados obtenidos en la ventana de ejecución de comandos de Windows del host PC-03(figura 3.27), durante la aplicación del filtro para protocolo FTP con ancho de banda de 20 Mbps, en donde se puede ver que se descargó del servidor un archivo de aproximadamente 20 MB en 11.25 segundos a una velocidad de casi 20 Mbps.

Los resultados son los siguientes:

ftp> get home.tar.gz c:\home.proof

200 PORT command successful. Consider using PASV.

150 Opening BINARY mode data connection for home.tar.gz (27065298 bytes).

226 File send OK.

ftp: 27065298 bytes recibidos en 11,25 segundos 2406,66 a KB/s.

En la figura 3.30 mediante el programa PRTG, se muestra el resultado del uso de ancho de banda con filtro para el protocolo FTP y con un ancho de banda de 2 Mbps.

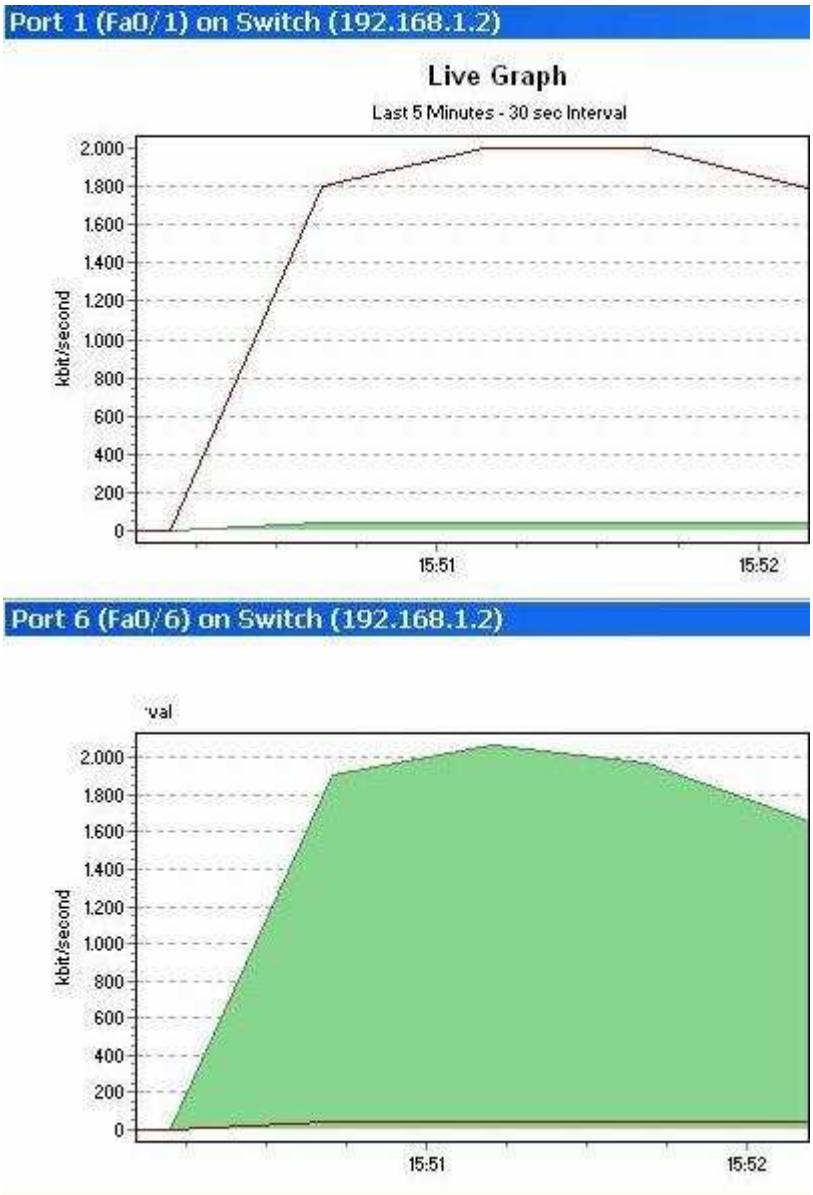


Figura 3.30.- Resultado de filtro FTP a 2 Mbps mostrado por PRTG.

A continuación se muestra los resultados obtenidos en la ventana de ejecución de comandos de Windows del host PC-03(figura 3.27), durante la aplicación del filtro para protocolo FTP con ancho de banda de 2 Mbps, se descargó el archivo del servidor en 112.88 segundos a una velocidad de casi 2 Mbps.

Los resultados obtenidos son los siguientes:

```
 ftp> get home.tar.gz c:\home.proof
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for home.tar.gz (27065298 bytes).
226 File send OK.
ftp: 27065298 bytes recibidos en 112,88 segundos 239,77 a KB/s.
```

En la figura 3.31, se observa que existe un pico que supera el límite de 1 Mbps asignado para el protocolo FTP , esto se debe a que además de los paquetes FTP, también cursaron paquetes ARP por el canal. Por lo tanto, el tráfico de paquetes ARP, al ser no clasificado, utilizó la clase de 1 Kbps asignado para éste tipo de tráfico. De los resultados obtenidos en la ventana de ejecución de comandos de Windows del host PC-03(figura 3.27) se puede ver que en ésta ocasión, se descargó el archivo del servidor en 225.90 segundos a una velocidad de casi 1 Mbps.

Los resultados obtenidos son:

```
ftp> get home.tar.gz c:\home.proof
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for home.tar.gz (27065298 bytes).
226 File send OK.
ftp: 27065298 bytes recibidos en 225,90 segundos 119,81 a KB/s.
```

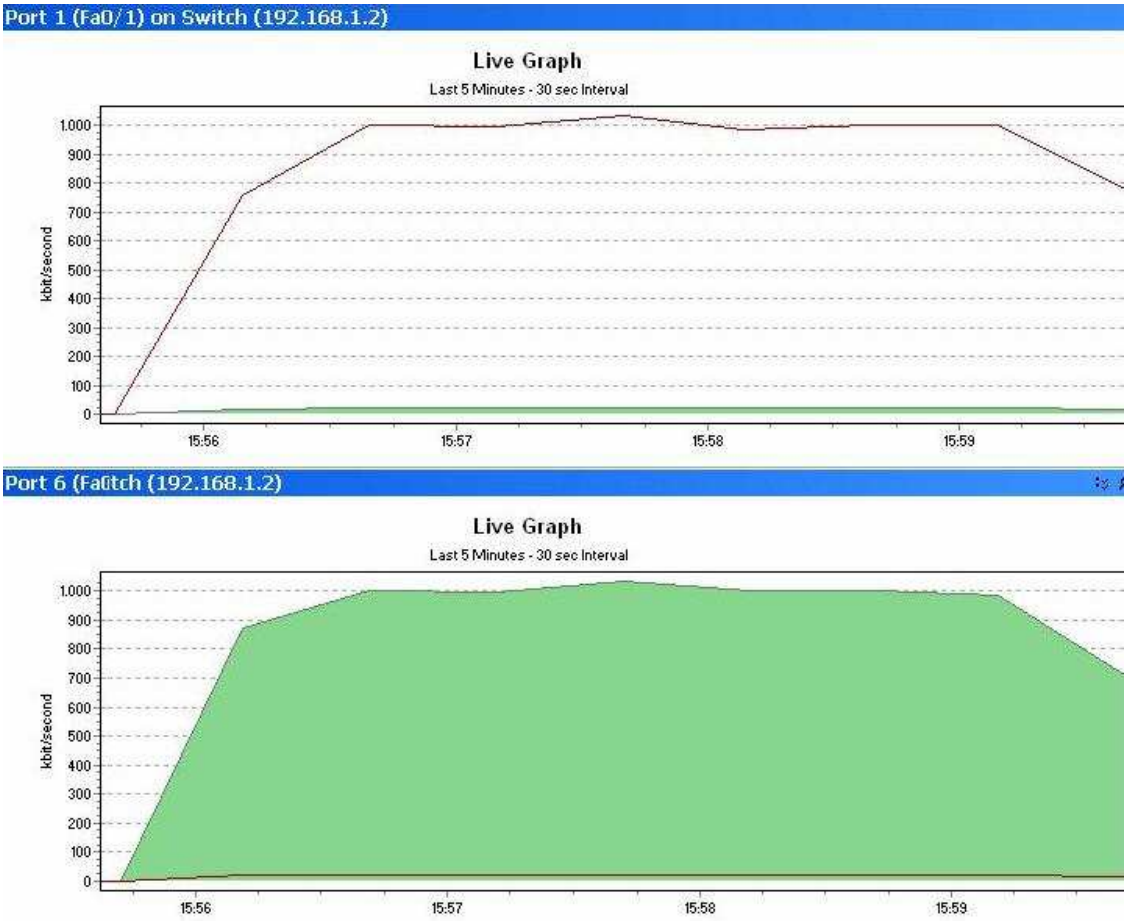


Figura 3.31.- Resultado de filtro FTP a 1 Mbps mostrado por PRTG.

Adicionalmente, se realizó una prueba con filtros para los protocolos HTTP y FTP configurados con un ancho de banda de 1 Mbps y 5 Mbps respectivamente; la figura 3.32 muestra el resultado obtenido por el programa PRTG de dicha prueba.

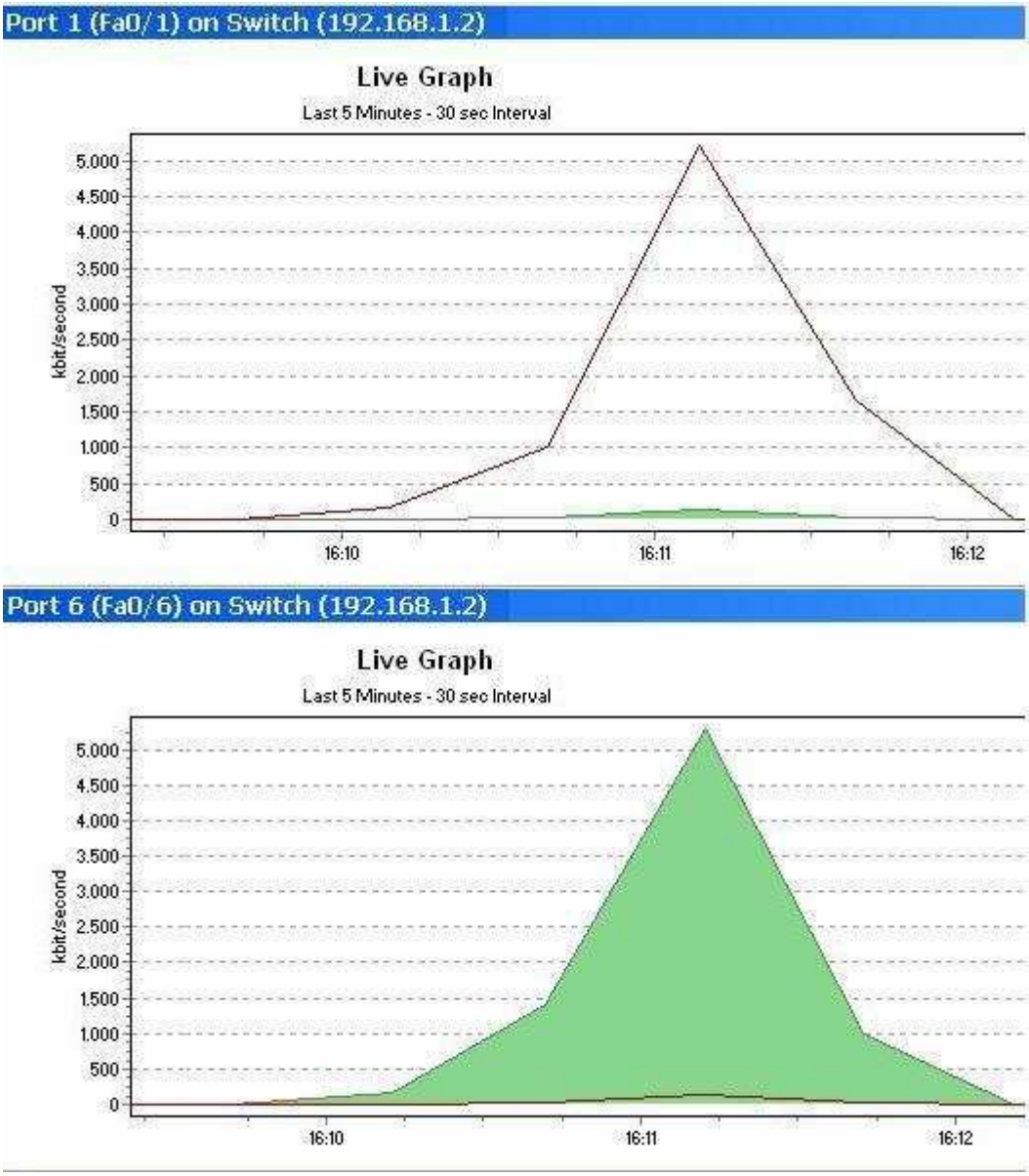


Figura 3.32.- Resultado de filtros HTTP a 1 Mbps y FTP a 5 Mbps mostrado por PRTG.

En la figura 3.32 se observa que el límite de es de 6 Mbps, y éste se debe a la suma de los anchos de banda asignados para cada protocolo, al cual además se le tiene que agregar el 1 Kbps del tráfico no clasificado. De los resultados obtenidos en la ventana de ejecución de comandos de Windows del host PC-03, se puede ver que se descargó del servidor el archivo en 45.08 segundos a una velocidad de casi 5 Mbps.

Los resultados obtenidos son los siguientes:

ftp> get home.tar.gz c:\home.proof
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for home.tar.gz (27065298 bytes).
226 File send OK.
ftp: 27065298 bytes recibidos en 45,08 segundos 600,45 a KB/s.

Los resultados del filtro para el protocolo HTTP, muestran que al sobrecargar al servidor con peticiones que superan el ancho de banda asignado para dicha aplicación, se obtiene lo que se muestra en la figura 3.33:

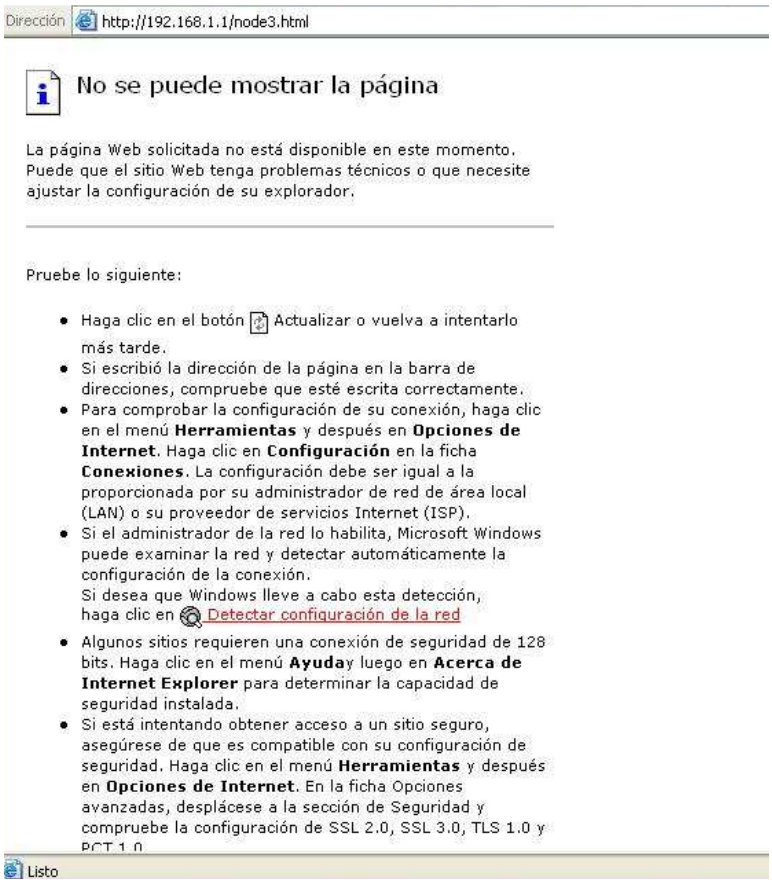


Figura 3.33.- Resultado de superar el límite en aplicación HTTP.

3.2.4 HERRAMIENTAS ADICIONALES

Se crearon herramientas que ayudan al monitoreo de red, como son preguntas y respuestas de eco, verificación de ruta hacia otro dispositivo y actividad de la conexiones locales del host con la red.

Ping

Herramienta que permite enviar y recibir peticiones de eco para determinar si el destino está listo para recibir información. En la figura 3.34 se muestra las opciones de esta herramienta:

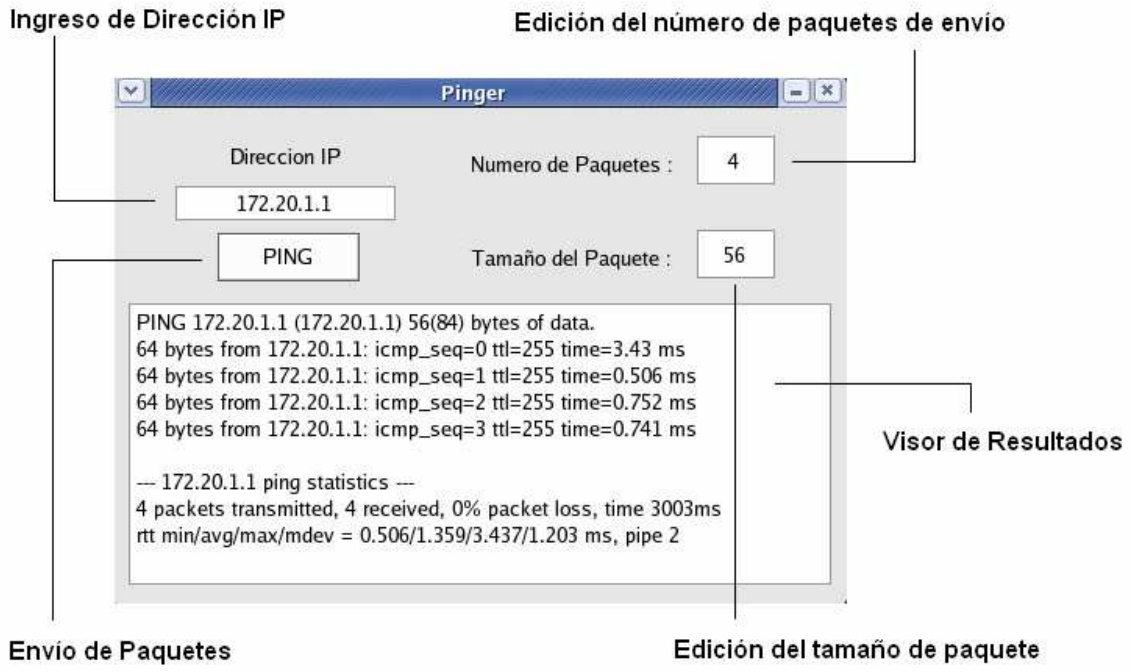


Figura 3.34.- Herramienta Ping de la Aplicación ALCATRAZ.

TraceRouter

Herramienta que permite determinar el número de saltos que un dispositivo realiza y el camino que traza para alcanzar una dirección IP específica. En la figura 3.35 se muestra las opciones de esta herramienta.

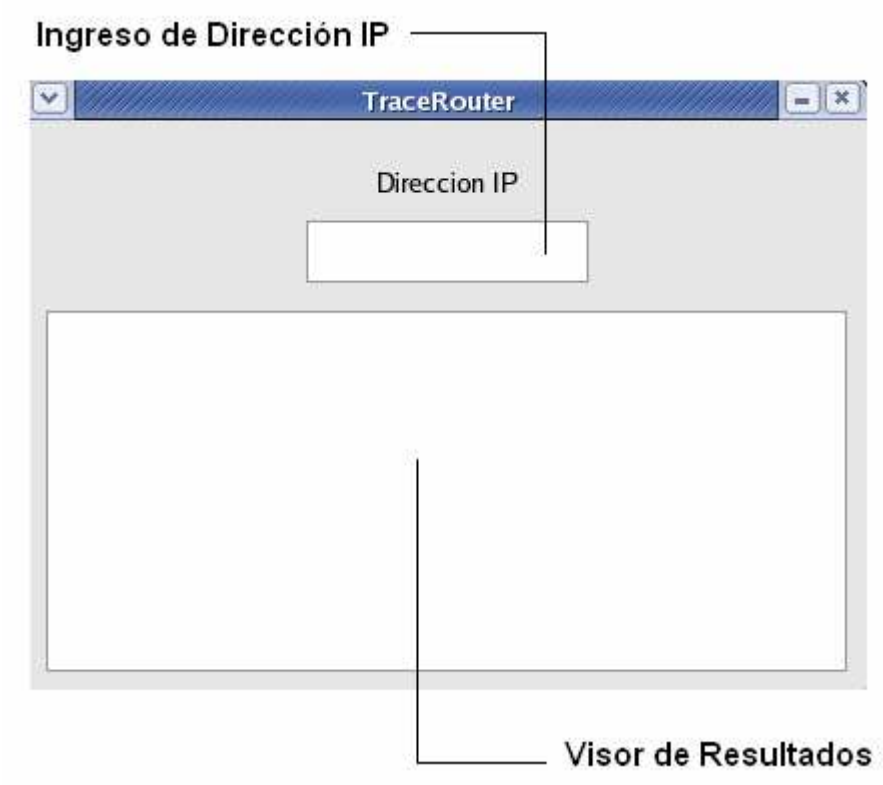


Figura 3.35.- Herramienta TraceRouter de la Aplicación ALCATRAZ

Actividad Local

Ésta herramienta permite observar las estadísticas brindadas por el comando *netstat,* mediante el cual se puede tener en detalle, el estado de las conexiones TCP y UDP del host con la red.

CAPÍTULO 4

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

- La administración de red tiene por objetivo proporcionar herramientas automatizadas y manuales de administración al usuario de red, para que éste pueda detectar posibles fallas o degradaciones en el desempeño de la misma. Le permite contar con estrategias de administración para optimizar la infraestructura existente y mejorar el rendimiento de aplicaciones y servicios.
- Muchas instituciones mantienen estructuras de comunicaciones complejas, debido a la necesidad de mantenerse comunicados con sus filiales; por lo que precisan de una buena administración de red que permita un mejor manejo y control de los elementos que la conforman.
- En LINUX se dispone de comandos como NMAP que con ayuda de varios protocolos como ICMP, nos facilitan de gran forma la detección de dispositivos activos presentes en la red. NMAP además nos presenta el listado de direcciones MAC, el tipo y sistema operativo del dispositivo.
- El principal problema de LINUX, es la dificultad de configuración. ALCATRAZ es una interfaz de configuración gráfica, que permite un sencillo control de todas las funcionalidades de la aplicación.
- La principal ventaja de la aplicación ALCATRAZ es su integración. Unir en un mismo paquete, funcionalidades como administración de ancho de banda, monitor de dispositivos y monitor de tráfico.
- Dentro del programa ALCATRAZ, en la parte de Administración de Trafico de Datos, requiere que el usuario tenga un mínimo conocimiento de los conceptos básicos del manejo de ancho de banda y de cómo éste puede afectar al desempeño de la red LAN.

- Para poder establecer una buena política de uso de ancho de banda se debe realizar un seguimiento del mismo. ALCATRAZ provee un historial detallado del tráfico cursado en la red, permitiendo establecerse con estos datos una política de consumo del ancho de banda, basada en argumentos reales.
- Con los datos del ancho de banda que se dispone en la red LAN, con ALCATRAZ se dispone de una herramienta para ajustar el consumo de ancho de banda de acuerdo a parámetros tales como: IP fuente, IP destino y tipo de protocolo.
- Gracias a las facilidades que presta el programar en SH, se puede manipular los comandos de administración de redes que tiene LINUX, para luego, tratar los datos obtenidos, clasificar la información y posteriormente presentarla en forma que sea fácilmente entendida por el usuario del programa ALCATRAZ.
- En Qt/Designer, se facilita la creación de aplicaciones gráficas. Ya que se presenta la facilidad de implementar señales y eventos entre botones, líneas de edición y demás elementos. Se puede conectar tantas señales como se desee a un solo evento, y una señal puede ser conectado a tantos eventos como se quiera, incluso, se puede conectar una señal a otra señal.
- SNMPv.3 posee la misma estructura básica de las versiones anteriores (SNMPv.1 y SNMPv.2), hecho que ha facilitado su desarrollo y avance, además de su comprensión. Lo que busca esta nueva versión, es cubrir las deficiencias de las otras versiones, enriqueciendo todos sus componentes para ofrecer nuevas capacidades de seguridad y facilidades de administración.
- Si en los dispositivos administrables y hosts (entorno LINUX) no tienen configurado, instalado y activado el servicio de SNMP, no se podrá obtener de éstos toda la información requerida para mostrar en pantalla.

RECOMENDACIONES

- No es recomendable la utilización de un único host como servidor para diferentes aplicaciones (FTP, MAIL, DNS, WEB, etc). Esto, para evitar que todo el tráfico sea dirigido hacia un solo host y éste se vea congestionado por la cantidad de información que debe manejar.
- Que la documentación entregada de QT/Designer y de la programación en SH, se tome en cuenta como documento base para futuros Proyectos.
- Que en el momento de elaborar una aplicación en Qt/Designer, se utilice la opción inicial QWizard, por cuanto permite agregar elementos sin mayor complejidad.
- Para las señales de click o de ratón es recomendable utilizar pushbuttons, mientras que, para señales desde teclado es recomendable utilizar líneas de edición, aunque las señales pueden actuar de la misma forma en cualquiera de los elementos.
- Se recomienda la programación en SH por ser de fácil comprensión y facilita la detección de errores en caso de existirlos, además que hay una gran cantidad de información disponible en Internet con manuales y muy accesibles.

GLOSARIO

ARP. (Address Resolution Protocol). El protocolo de resolución de direcciones es responsable de convertir la dirección de protocolo de alto nivel (direcciones IP) a direcciones de red físicas. El protocolo IP debe conocer la dirección física del dispositivo destino para que la capa de enlace pueda proceder con la transmisión de paquetes IP.

ext2. Sistema de Ficheros Extendido 2. Permite hasta 256 caracteres en los nombres de los ficheros y tamaños de estos de hasta 4 Terabytes.

FTP .Protocolo de Transferencia de Archivos *(File Transfer Protocol)***.** Es uno de los diversos protocolos de capa aplicación, Es el ideal para transferir grandes bloques de datos por la red.

GNU. Es un proyecto que ha desarrollado un sistema completo de software libre llamado “GNU” (GNU No es Unix) que es compatible con Unix. El proyecto GNU no está limitado a sistemas operativos, si no a proporcionar un amplio espectro de software esto incluye software de aplicación.

Gopher. Es un sistema de entrega de información distribuido. Utilizando gopher se puede acceder a información local o bien a acceder a servidores de información gopher de todo el mundo.

HTTP. Protocolo de transferencia de hipertexto (HTTP, HyperText Transfer Protocol) es el protocolo usado en cada transacción de la Web (WWW World Wide Web). El hipertexto es el contenido de las páginas web, y el protocolo de transferencia es el sistema mediante el cual se envían las peticiones de acceso a una página web, y su respuesta, remitiendo la información que se verá en pantalla.

ICMP. El Protocolo IP utiliza el protocolo de mensajes de control Internet "ICMP Internet Control Message Protocol" para informar de los errores que pueden ocurrir durante el enrutamiento de paquetes IP. ICMP es realmente una parte integral del protocol IP.

Inodo. En informática, especialmente en sistemas operativos de la familia Unix (Solaris, HP-UX, AIX, GNU/LINUX etc),un inodo o Index Node es un apuntador a sectores específicos de disco duro en los cuales se encuentra la información del archivo, un inodo también contiene información de permisos, propietarios y grupos a los cuales pertenece el archivo.

IOS. Sistema de Entrada y Salida (Input Output System). Es el sistema operativo de switches y ruteadores. Al igual que un host, un ruteador o switch no puede funcionar sin un sistema operativo, es decir, el hardware no puede realizar ninguna función.

NetBIOS. Protocolo de red originalmente creado para redes locales de computadoras IBM PC. NetBIOS engloba un conjunto de protocolos de nivel de sesión, que proveen 3 tipos de servicios: Servicio de nombres, Servicio de paquetes y Servicio de sesión.

Nodo. Punto final de la conexión de red o una unión que es común para dos o más líneas de una red. Los nodos pueden ser procesadores, controladores o hosts. Los nodos, que varían en cuanto al enrutamiento y a otras aptitudes funcionales; pueden estar interconectados mediante enlaces y sirven como puntos de control en la red. La palabra nodo a veces se utiliza de forma genérica para hacer referencia a cualquier entidad que tenga acceso a una red y frecuentemente se utiliza de modo indistinto con la palabra dispositivo.

NTP. (Protocolo de Tiempo de Red), Protocolo desarrollado sobre el TCP que garantiza la precisión de la hora local, con referencia a los relojes de radio y

atómicos ubicados en la Internet. Este protocolo puede sincronizar los relojes distribuidos en milisegundos durante períodos de tiempo prolongados.

PBX. (Private Branch Exchante). de Private Business eXchange. Es una central telefónica que es utilizada para negocios privados. En comparación a una compañía telefónica.

PDU. Es la unidad de datos de protocolo de las capas del modelo ISO/OSI.

POSIX (Portable Operating System Interface). Estándar que describe una condición para la fuente del software, debe ser un sistema operativo compatible POSIX, es decir, que sea un sistema portable y ser recompilado con poca dificultad.

PPP. (Point-to-Point Protocol). El protocolo Punto A Punto es un protocolo más reciente y robusto que SLIP, pero cumplen funciones similares.

Proxy. El término *proxy* hace referencia a un programa o dispositivo que realiza una acción en representación de otro. La finalidad más habitual es la del *servidor proxy*, que sirve para permitir el acceso a Internet a todos los equipos de una organización cuando sólo se puede disponer de un único equipo conectado, esto es, una única dirección IP.

Ruteador. Un ruteador es un tipo especial de computador. Cuenta con una CPU, memoria, bus de sistema y distintas interfaces de entrada/salida. Sin embargo, los ruteadores están diseñados para cumplir algunas funciones muy específicas como conectar y permitir la comunicación entre dos redes y determinan la mejor ruta para la transmisión de datos a través de las redes conectadas.

SLIP. (Serial Line Internet Protocol) permite la transmisión de paquetes IP sobre líneas seriales (líneas telefónicas). La información es empaquetada y transmitida en paquetes IP. Trabaja sobre TCP/IP.

SMTP. Simple Mail Transfer Protocol (SMTP), o protocolo simple de transferencia de correo electrónico. Protocolo de red basado en texto utilizado para el intercambio de mensajes de correo electrónico entre computadoras y/o distintos dispositivos (PDA's, Celulares, etc).

Socket. Similar a un puerto TCP/IP. Es un identificador para un servicio particular, en un nodo particular de la red, lo que realmente significa es que es la combinación de un puerto y una dirección IP. Un par de sockets conectados proveen transferencia de información entre hosts remotos.

Switch. Un switch (en castellano "interruptor" o "conmutador") es un dispositivo de interconexión de redes de hosts/computadoras que opera en la capa 2 (nivel de enlace de datos) del modelo OSI (Open Systems Interconection). Un switch interconecta dos o más segmentos de red, pasando datos de una red a otra, de acuerdo con la dirección MAC de destino de los datagramas en la red.

Telnet. Terminal de red. Aplicación que permite que un usuario se conecte a otro dispositivo en cualquier parte de la red y actúe como un terminal del mismo.

Throughput. Tasa de transferencia, rendimiento o throughput, se refiere a la tasa efectiva de bits.

UDP. (Protocolo de Datagrama de usuario). Es un protocolo no orientado a conexión. No proporciona fiabilidad ni mecanismos de control de flujo. No proporcionan procedimientos de recuperación de errores. Protocolos del nivel de aplicación, como el Protocolo de Transferencia de Datos Trivial (TFTP) y la Llamada de Procedimiento Remoto (RPC) utilizan UDP.

UUCP. Unix to Unix Copy. En origen es una utilidad de copia de archivos de Unix para enviar archivos de un dispositivo a otro por línea serial. Actualmente se utiliza también para transferir correo electrónico y grupos de noticias.

XML. Lenguaje Extensible basado en Marcas (Extensible Markup Language). Es un estándar abierto para describir datos. Permite al desarrollador de páginas web definir marcas especiales.

ANEXOS

ANEXO A : PROGRAMACIÓN EN SH

Programación en BASH shell

NOMBRE

bash - (GNU Bourne-Again Shell)

COPYRIGHT

Bash is Copyright (C) 1989-2002 by the Free Software Foundation, Inc.

DESCRIPCIÓN

Bash es un interprete de lenguaje de comandos sh-compatible que ejecuta comandos leídos desde la entrada Standard o desde un archivo. Bash también incorpora características de uso de las shells Korn y C (ksh y csh).

Bash esta adaptado conforme a la implementación de el IEEE POSIX Shell y Herramientas de especificación (IEEE Working Group 1003.2).

La orden echo

Puede usarse para visualizar mensajes, muestra sus argumentos en el terminal, que es el dispositivo de salida Standard. Sin argumento produce una línea vacía y por defecto agrega una nueva línea al final de la salida.

Por ejemplo:

```
j4nusx@xeon:~$ echo Hola BSDeros
Hola        BSDeros
j4nusx@xeon:~$_
```

Nota: La cadena de argumentos puede tener cualquier numero de caracteres. Sin embargo si la cadena contiene algún meta carácter deberá escribirse entre comillas.

Variables de Shell

Las Variables en la Shell se escriben generalmente con mayúsculas.
No hay espacios blancos a uno y otro lado del signo igual.
Y son visualizadas con el signo dollar $.

```
SISTEMA=sco
MSG="mi sistema operativo de código cerrado"
j4nusx@xeon:~$ echo $HOME
```

Eliminación de los significados especiales de los meta caracteres
Los meta caracteres tiene significados especiales para el shell. A veces, se requiere inhibir esos significados. El shell le proporciona un conjunto de caracteres que anula el significado de los meta caracteres. Este proceso de anular el significado especial de los meta caracteres se denomina escape.
(\ ) Slash invertido: Utilizado para indicar que el carácter que le sigue se interpreta como un carácter alfa-numérico ordinario.

```
j4nusx@xeon:~$ echo \"\'\<\>\?\&\$\\
"'<>?&$\
j4nusx@xeon:~$
```

Las dobles comillas ("): Puedes usar las dobles comillas para anular el significado de la mayoría de los caracteres especiales. Cualquier carácter especial entre un par de dobles comillas pierde su significado especial, excepto el signo de dollar $. (Utilice el slash invertido para eliminar sus significados especiales.).

```
j4nusx@xeon:~$ echo "*"
*
j4nusx@xeon:~$
```

Ahora inténtalo sin las comillas y podrás visualizar otro resultado.

Otro ejemplo pero con el signo de dollar $:

```
j4nusx@xeon:~$ echo "$HOME"
/home/j4nusx
j4nusx@xeon:~$
```

Las comillas sencillas ('): Las comillas sencillas funcionan de manera análoga a las dobles comillas. Cualquier carácter especial entre un par de comillas simples pierde su significado especial, excepto la comilla simple.

```
j4nusx@xeon:~$ echo '* $HOME ? & " '
* $HOME ? & "
j4nusx@xeon:~$
```

La comilla de acento grave (`): Esta distinción es muy importante. El shell interpreta dentro de los signos de acento grave como una orden ejecutable.

```
j4nusx@xeon:~$ echo La fecha actual es `date`
La fecha actual es Sat May 17 09:22:49 CDT 2003
j4nusx@xeon:~$
```

La orden read
Realizada principalmente para interactuar con el usuario, por medio de la entrada standart

```
j4nusx@xeon:~$ read YOSOY
Edwin Plauchu                <--- texto tipiado por
mi. j4nusx@xeon:~$ echo $YOSOY
Edwin Plauchu                <--- mostrando el contenido de la variable
capturada j4nusx@xeon:~$
```

Logica en la Shell
0 -true
1 -false
Bourne Again Shell es poseedor de una lógica inversa. Cuando una operación unix se termina con éxito el estado de la variable $? será 0.

```
j4nusx@xeon:~$ cat loco.txt
cat: loco: No such file or directory
j4nusx@xeon:~$ echo $?
1
j4nusx@xeon:~$
```

La construccion if -then

```
if [condición]
then
        órdenes
         ....
        última   orden
fi
```

La sentencia if finaliza con la palabra reservada fi (if escrito al revés).
El sangrado no es necesario, pero hace lucir al código más elegante.
Nota: Los corchetes que están alrededor de las condiciones son necesarios y deben estar rodeados por espacios en blanco.

Verdadero o Falso: La orden Test
La orden test que es interna al shell evalúa la expresión que se le da como argumento y devuelve verdadero si la expresión asi lo es. Test puede usarse de dos maneras:

```
if  test  "$VARIABLE"=valor
   then
                o
if  ["$VARIABLE"=valor]
   then
```

La construcción if -then -else

```
if [condición]
   then
      ordenes_en_caso_de_condicion_verdadera
      ....
else
      ordenes_en_caso_de_condicion_falsa
....
fi
```

La construccion if -then -elif

```
if [condicion_1]
  then
    ordenes
    ....
elif [condicion_2]
  then
    ordenes
    ....
elif [condicion_3]
  then
    ordenes
    ....
else
    ordenes
    .... fi
```

El Bucle while

```
    while [condicion]
do
          ordenes
          mas ordenes
      done
```

ejemplo:

```
CONTADOR=0
 while [  $CONTADOR -lt 10 ]; do
      echo El contador es $CONTADOR
      let CONTADOR=CONTADOR+1
 done
```

El Bucle until

```
until [condicion]
       do
ordenes
          mas ordenes
       done
```

ejemplo:

```
    CONTADOR=20
    until [   $CONTADOR  -lt  10  ];  do  echo
CONTADOR $CONTADOR
        let CONTADOR-=1
    done
```

Estructura for - in

```
    for variable in (lista de valores)
```

```
        do
            ordenes
      mas       ordenes..
done
ejemplo:

    for VARIABLE in `/etc/rc.d/rc.*`
        do
echo "$VARIABLE start"   # Arranca todos mis demonios
        done
```

Estructura select

```
select variable in  (lista de valores)
   do
      lista de ordenes
 done
```

ejemplo:

```
select VARIABLE in `ls`
        do
      echo "Cadena escogida $VARIABLE "
            echo "Numero de respuesta $REPLY"
            break      # Rompe el ciclo
        done
```

La línea leída es salvada en la variable REPLY. La lista es ejecutada después de cada selección, hasta que se aplique el comando *break* o un EOF.

Funciones

Como en casi todo lenguaje de programación, puede utilizar funciones para agrupar trozos de código de una manera más lógica, o practicar el divino arte de la recursión. Declarar una función es sólo cuestión de escribir function mi_func { mi_código }. Llamar a la función es como llamar a otro programa, sólo hay que escribir su nombre.

Evaluación aritmética

Pruebe esto en la línea de comandos (o en una shell):

```
echo 1 + 1
```

Si esperaba ver '2', quedará desilusionado. ¿Qué hacer si quiere que BASH evalúe unos números? La solución es ésta:

```
echo $((1+1))
```

Esto producirá una salida más 'lógica'. Esto se hace para evaluar una expresión aritmética. También puede hacerlo de esta manera:

```
echo $[1+1]
```

Si necesita usar fracciones, u otras matemáticas, puede utilizar bc para evaluar expresiones aritméticas.

Si ejecuta "echo $[3/4]" en la línea de comandos, devolverá 0, porque bash sólo utiliza enteros en sus respuestas. Si ejecuta "echo 3/4|bc -l", devolverá 0.75.

Capturando la salida de un comando

Este pequeño script muestra todas las tablas de todas las bases de datos (suponiendo que tenga MySQL instalado). Considere también cambiar el comando 'mysql' para que use un nombre de usuario y clave válidos.

```
#!/bin/bash
        DBS=`mysql -uroot  -e"show databases"`
        for b in $DBS ;
        do
                mysql -uroot -e"show tables from $b"
done
```

Las Órdenes Lógicas

Las Órdenes lógicas efectúan operaciones sobre órdenes de UNIX.

Condición AND

```
    comando1 && comando2
```

comando2 es ejecutado si, y solo si, comando1 retorna un estado de salida cero.

Condición OR

```
            comando1 || comando2
```

comando2 es ejecutado si y solo si comando1 retorna un estado de salida distinto de cero.

Construcción case

La estructura case escoge entre varias alternativas posibles.

```
case $OPCION
   patron1)
     ordenes
      ;;
   patron2)
      ordenes
      ;;
   *)
    ordenes
      ;;
```

El * hace coincidencia con cualquier patrón

Categorías de Comprobación de la Orden test de Bash

Sintaxis test numérico

test expresion_1 operador_logico expresion_2

INTEGER1 -eq INTEGER2
INTEGER1 es igual a INTEGER2

INTEGER1 -ge INTEGER2
INTEGER1 mayor que o igual a INTEGER2

INTEGER1 -gt INTEGER2
INTEGER1 es mayor que INTEGER2

INTEGER1 -le INTEGER2
INTEGER1 es menor que o igual a INTEGER2

INTEGER1 -lt INTEGER2
INTEGER1 es menor que INTEGER2

INTEGER1 -ne INTEGER2
INTEGER1 es distinto de INTEGER2

Sintaxis test cadenas comparación

test expresion_1 operador_logico expresion_2

STRING1 = STRING2
Las cadenas son iguales

STRING1 != STRING2
Las cadenas son diferentes

Sintaxis test cadenas comprobación

test operador "$VARIABLE"

Comprobación de cadenas	
-z	Prueba si la cadena esta vacía
-n	Comprueba el valor de una cadena
str	Verifica que no sea una cadena nula

Sintaxis test archivo comparación

FILE1 -ef FILE2
FILE1 y FILE2 tienen mismo manejador y números de inodo

FILE1 -nt FILE2
FILE1 es mas nuevo (fecha de modificación) que FILE2

FILE1 -ot FILE2
FILE1 es más viejo que FILE2

Sintaxis test archivo comprobación

test operador "$FILE"

Prueba de Archivos	
-f $FILE	El fichero existe y es un archivo regular.
-L $FILE	El nombre de fichero es un vinculo simbólico
-s $FILE	El fichero no esta vació
-r $FILE	El fichero se puede leer.
-w $FILE	El fichero puede ser modificado y se puede escribir en el.
-x $FILE	El fichero es ejecutable
-d $FILE	El nombre de fichero es un directorio
-c $FILE	El nombre de archivo hace referencia a un dispositivo de carácter
-b $FILE	El nombre de archivo hace referencia a un dispositivo de bloque
-O $FILE	El archivo existe y es propietario del mismo
-p $FILE	El archivo existe y es fue nombrado tubería
-S $FILE	El archivo es un socket
-u $FILE	El archivo existe y tiene activado el bit set-user-ID
-t [FD]	El descriptor de archivo FD (stdout por default) esta abierto sobre una terminal
-e $FILE	El archivo existe
-k $FILE	El archivo existe y tiene activado el bit sticky.
-G $FILE	El archivo existe y es propietario efectivo por ID de grupo.

Operaciones Lógicas con Expresiones

! EXPRESSION
NOT lógico

EXPRESSION1 -a EXPRESSION2
AND lógico

EXPRESSION1 -o EXPRESSION2
OR lógico

Variables especiales de Shell	
$#	Contiene el numero de parámetros de la línea de orden
$$	Contiene el numero PID (ID del proceso) del proceso en ejecución
$?	Contiene el estado de salida de la ultima orden
$0	Contiene el nombre del guión, tal como se escribe en la línea de orden
$@ o $.	Contiene todos los parámetros de la línea de orden
$1 , $2 .. $9	Las Variables especiales $1, $2, ... $9. Contienen los argumentos del 1 al 9, respectivamente. Se ignoran los argumentos de la línea de orden posteriores al 9.

Sustitución de Parámetros

El shell proporciona la posibilidad para sustituir parámetros, lo que permite comprobar su valor y cambiarlo de acuerdo a una opción especificada. Esto es útil en la programación de shell, cuando necesita verificar si una variable es igual a algo. Por ejemplo, cuando emite una orden read en un guión, necesita asegurarse de que el usuario ha introducido alguna cosa antes de realizar ninguna acción.
El formato consiste en un signo de dólar ($), un conjunto de llaves ({y}), una variable, dos puntos (:), un carácter y una palabra de la forma siguiente:

${variable:*caracter de opción* palabra}

El carácter opción determina lo que hay que hacer con la palabra. Los cuatro caracteres de opción se especifican mediante los signos + - = ?. Estas cuatro opciones funcionaran de manera diferente, dependiendo si la variable esta vacía o no.

Una variable esta vacía, solo si su valor es una cadena vacía.

${parámetro}: Colocando la variable (parámetro) dentro de las llaves evita que se origine conflicto con el carácter que sigue al nombre de la variable. El ejemplo siguiente clarifica esta cuestion.
Suponga que desea cambiar el nombre de un archivo llamado plauchu, especificado en la variable denominada ARCHIVO a plauchuX.

```
j4nusx@xeon:~$ echo $ARCHIVO
plauchu
j4nusx@xeon:~$ mv $ARCHIVO $ARCHIVOX
Usage: mv [-fi] source-file
j4nusx@xeon:~$
```

Esta orden no funciona porque el shell considera que $ARCHIVOX es el nombre de la variable que no existe.

```
j4nusx@xeon:~$ mv $ARCHIVO ${ARCHIVO}X
j4nusx@xeon:~$ ls pl*
plauchuX
```

Esta orden funciona porque el shell considera que $ARCHIVO es el nombre de la variable y sustituye su valor, en este caso plauchu.

${parametro:-cadena} La opción -(guión) significa que si la variable relacionada (parámetro) tiene asignado un valor y no esta vacía (no nula), se usa su valor. Si no es así, es decir, si la variable esta vacía (nula) o no tiene asignado valor, se sustituye su valor
con cadena.

Por ejemplo:

```
bash-2.05b$ VAR=
bash-2.05b$ echo ${VAR:-/etc/X11/XF86Config}
```

```
/etc/X11/XF86Config
bash-2.05b$ echo $VAR
bash-2.05b$
```

La variable VAR permanece como una variable vacía.

${parametro:-cadena}: La opción + es opuesta a la opción -.

```
bash-2.05b$ HELPME="help me"
bash-2.05b$ echo ${HELPME:+"Ayuda va en
camino"} Ayuda va en camino
bash-2.05b$ echo $HELPME
help me
bash-2.05b$
```

La variable HELPME permanece inalterada.

${parámetro:=cadena}: La opción = significa que si a la variable relacionada (parámetro) no tiene asignado un valor o esta vacía (nula), se sustituye su valor con cadena. Si no, la variable no esta vacía y su valor permanecerán inalterados. Por ejemplo:

```
bash-2.05b$ MSEG=
bash-2.05b$ echo ${MSEG:="Hola estoy
aquí"} Hola estoy aquí
bash-2.05b$ echo
$MSEG Hola estoy aquí
bash-2.05b$
```

La cena puede ser una cadena con espacios en blanco entre comillas. Funciona mientras se coloquen entre comillas.
El valor de MSEG se cambia y deja de ser una variable vacía.

${parametro:?cadena}: La opción ? significa que si la variable relacionada (parámetro) tiene asignado un valor y no esta vacía, entonces se sustituye su valor. Si no, si la variable esta vacía se imprime la palabra y se sale del guión actual. Si se omite cadena y se muestra el mensaje prefijado parameter null or not set.
Por ejemplo:

```
bash-2.05b$ MSEG=
bash-2.05b$ echo ${MSEG:?"Error"}
bash: MSEG: Error
bash-2.05b$
```

El shell evalúa la variable MSEG que esta vacía. De forma que la opción ? provocara la sustitución de la cadena Error, que pasa a la orden echo para visualizarla

ANEXO B : PROGRAMACIÓN EN QT/DESIGNER

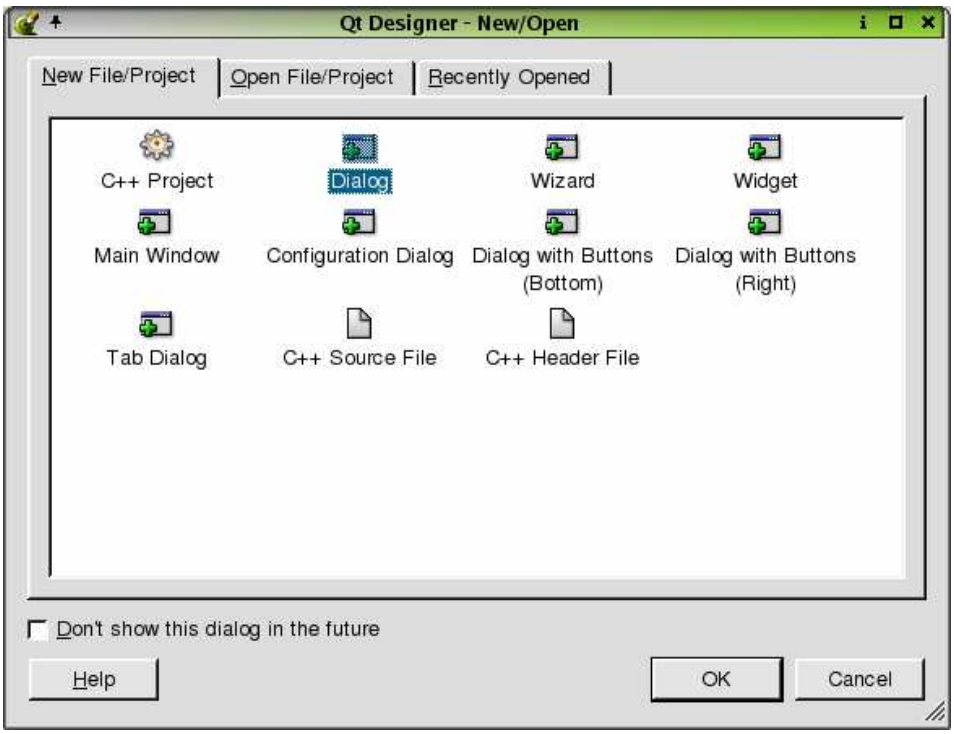
Qt Designer Manual

STARTING AND EXITING *QT DESIGNER*

To start *Qt Designer* under Windows click the **Start** button and click **Programs|Qt X.x.x|Designer**. (X.x.x is the Qt version number, e.g. 3.1.0.) If you're running a Unix or Linux operating system you can either double click the *Qt Designer* icon or enter designer & in an xterm.

When *Qt Designer* starts, it shows the *New/Open* dialog. If you prefer to not have this dialog appear the next time you open *Qt Designer*, check the "Don't show this dialog in the future" checkbox.

For this example, click **Cancel** to skip over the dialog.



When you've finished using *Qt Designer* click **File|Exit**; you will be prompted to save any unsaved changes. Help is available by pressing **F1** or from the **Help** menu.

To get the most benefit from the tutorial chapters we recommend that you start *Qt Designer* now and create the colortool application as you read. Most of the work involves using *Qt Designer*'s menus, dialogs and editors. We also suggest that as you work through this manual you enter the code directly using *Qt Designer*'s code editor. You can cut and paste the code from the on-line version of this manual or

copy it from the example source code.

When you start *Qt Designer*, by default, you will see a menu bar and various toolbars at the top. On the left is the widget Toolbox. Click the toolbox's buttons to reveal a particular set of tools. On the right there are three windows: the first is the Project Overview window, the second is the Object Explorer window, and the third is the Properties Editor/Signal Handlers window. The Project Overview window lists the files and images associated with the project; to open any form (.ui file), or the code associated with it (in the .ui.h file), simply single click it. The Object Explorer window lists the current form's widgets and members. The Properties Editor/Signal Handlers window is used to view and change the properties of forms and widgets. We will cover the use of *Qt Designer*'s windows, dialogs, menu options and tools as we create the example application.

CREATING THE PROJECT

Our colortool application is going to be a standard C++ application, so we need to create a C++ project and add our files and code to this project.

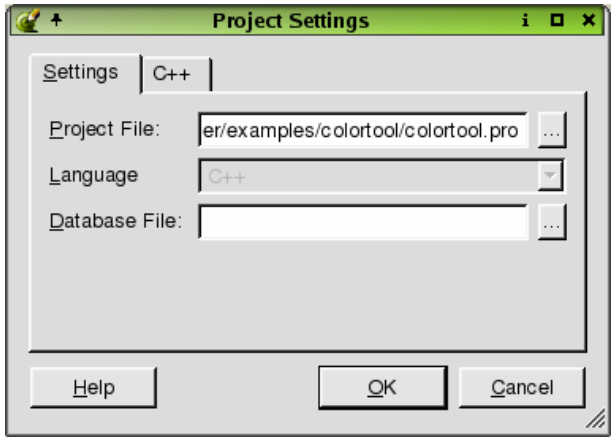
Creating a Project

Whenever you create a new application we recommend that you create a project file and open the project rather than individual .ui files. Using a project has the advantage that all the forms you create for the project are available via a single mouse click rather than having to be loaded individually through file open dialogs. An additional benefit of using project files is that they allow you to store all your images in a single file rather than duplicate them in each form in which they appear. See The Designer Approach chapter's Project management section for detailed information on the benefits of using project files.

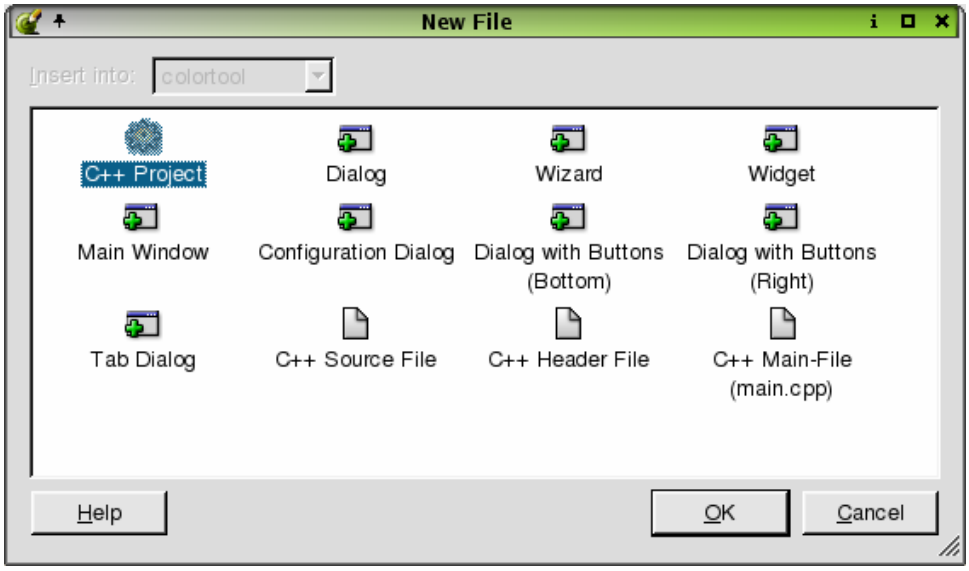
Project files use the .pro suffix and are used by the qmake tool to create makefiles for the relevant target platforms.

Create a new project as follows:

1. Click **File|New** to invoke the *New File* dialog.
2. Click "C++ Project" to create a C++ project, then click **OK** to invoke the *Project Settings* dialog.
3. Click the ellipsis button to the right of the Project File line edit to invoke the *Save As* dialog. Use this dialog to navigate to where you want to create the new project, ideally creating a new folder for it (e.g. called "colortool"), using the **Create New Folder** toolbar button.
4. Enter "colortool.pro" as the file name, then click **Save**. The project's name will now be "colortool"; click **OK** to close the *Project Settings* dialog.



5. Click **File|Save** to save the project.



The *New File* dialog is used to create all the files that can be used in a *Qt Designer* project. This includes C++ source files, an automatically generated main.cpp file (if you are in a project), and a variety of forms based on pre-defined templates. (You can create your own templates too.)

For the colortool application we want to start with a main window form. When we create this form, *Qt Designer* will present a wizard which we can use to automatically create menu and toolbar options and automatically create the relevant signal/slot connections. For every menu option or toolbar button, *Qt Designer* will create a single QAction (see the Actions and Action Groups sidebar).

Actions and Action Groups

An *action* is an operation that the user initiates through the user interface, for example, saving a file or changing some text's font weight to bold.

We often want the user to be able to perform an action using a variety of means. For example, to save a file we might want the user to be able to press **Ctrl+S**, or to click the **Save** toolbar button or to click the **File|Save** menu option. Although the

means of invoking the action are all different, the underlying operation is the same and we don't want to duplicate the code that performs the operation. In Qt we can create an action (a QAction object) which will call the appropriate function when the action is invoked. We can assign an accelerator, (e.g. **Ctrl+S**), to an action. We can also add an action to a menu and to a toolbar.

If the action has an on/off state, e.g. bold is on or off, when the user changes the state, for example by clicking a toolbar button, the state of everything associated with the action, e.g. menu items and toolbar buttons, is updated.

Some actions should operate together like radio buttons. For example, if we have left align, center align and right align actions, only one should be 'on' at any one time. An *action group* (a QActionGroup object) is used to group a set of actions together. If the action group's exclusive property is TRUE then only one of the actions in the group can be on at any one time. If the user changes the state of an action in an action group where exclusive is TRUE, everything associated with the actions in the action group, e.g. menu items and toolbar buttons, is updated.

Qt Designer can create actions and action groups visually, assign accelerators to them, and associate them with menu items and toolbar buttons.

CREATING THE MAIN WINDOW

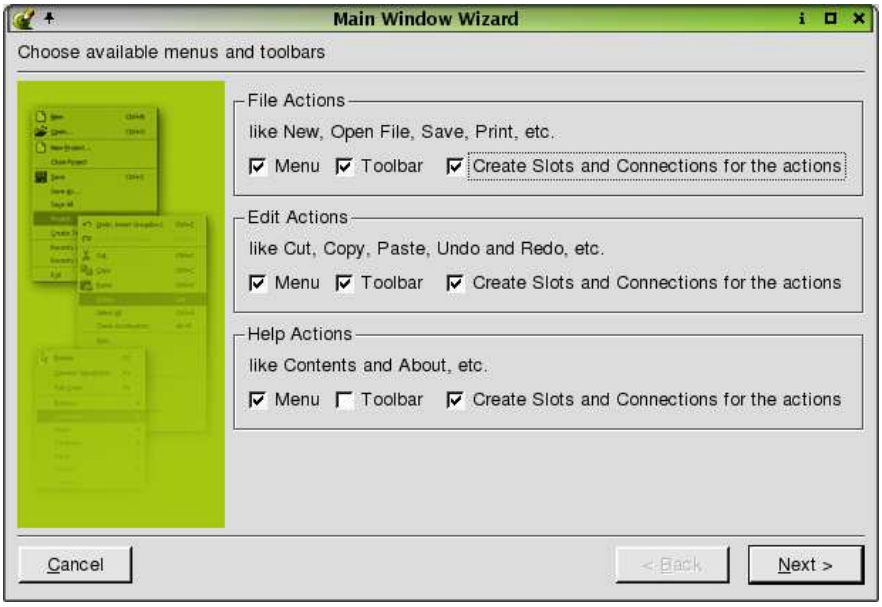
We will use the Main Window Wizard to build a main window. The wizard allows us to create actions as well as a menu bar and a toolbar through which the user can invoke the actions. We will also create our own actions, menus and toolbar buttons, and add a main widget to the main window.

Click **File|New** to invoke the *New File* dialog, click "Main Window" to create a main window form, then click **OK**. A new QMainWindow form will be created and the *Main Window Wizard* will pop up.

Using the Main Window Wizard

1. The *Choose available menus and toolbars* page appears first. It presents three categories of default actions, File Actions, Edit Actions and Help Actions. For each category you can choose to have *Qt Designer* create menu items, toolbar buttons, and signal/slots connections for the relevant actions. You can always add or delete actions, menu items, toolbar buttons, and connections later.

   We will accept the defaults for File Actions and for the Edit Actions, i.e. have menu items, toolbar buttons and the relevant connections created. In fact we'll be changing the Edit actions considerably later on, but it is still convenient to create them now. We won't have any Help Actions on the toolbar so uncheck the Help Action's Toolbar checkbox. Click **Next** to move on to the next wizard page.

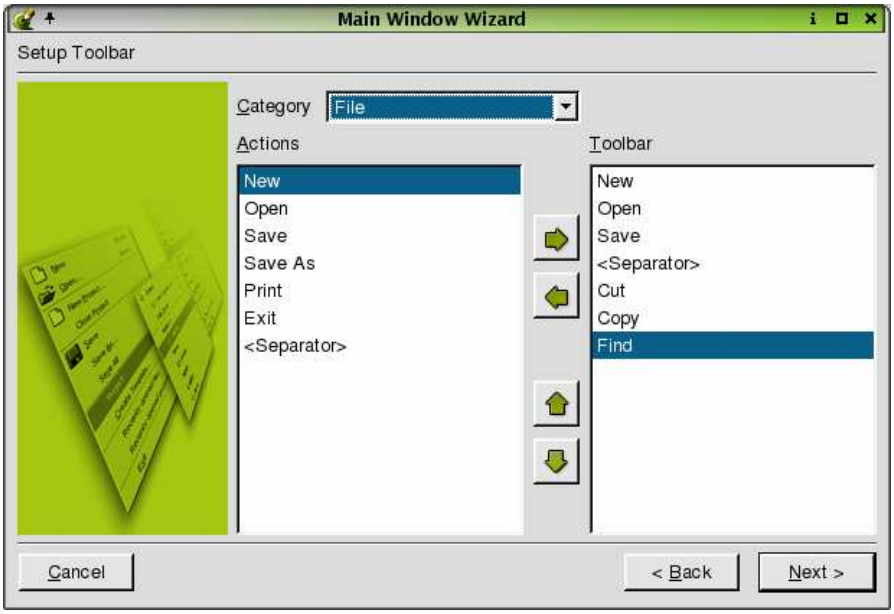


Main Window Wizard- Choosing menus and toolbars

2. The *Setup Toolbar* wizard page is used to populate a toolbar with actions from each of the default action categories. The Category combobox is used to select which set of actions you wish to choose from. The Actions list box lists the actions available for the current category. The Toolbar listbox lists the toolbar buttons you want to create. The blue left and right arrow buttons are used to move actions into or out of the Toolbar list box. The blue up and down arrow buttons are used to move actions up and down within the Toolbar list box. Note that the '<Separator>' item in the Actions list box may be moved to the Toolbar list box as often as required and will cause a separator to appear in the finished toolbar.

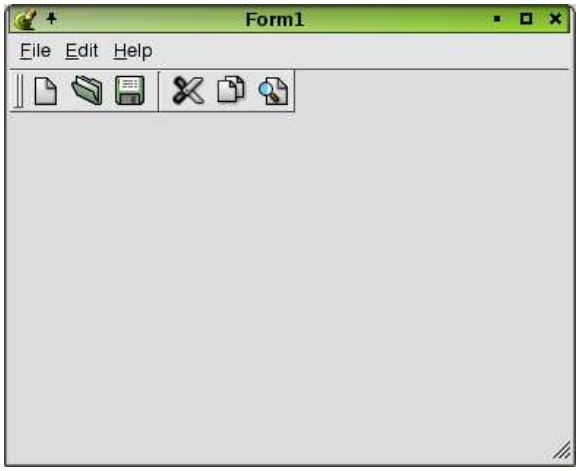
   Copy the New, Open, and Save Actions to the Toolbar list box. Copy a <Separator> to the Toolbar list box. Change the Category to Edit and copy the Cut, Copy, and Find actions to the Toolbar list box. Click **Next** and then click **Finish**.

   Click **File|Save** and save the form as mainform.ui.



Main Window Wizard- Setting up the toolbar

If you preview the form (**Ctrl+T**) the File and Edit menus will be available and you'll be able to drag the toolbar either into an independent window of its own, or to dock it to the left, right, bottom, or top of the window. The menus and toolbars are not yet functional, but we will rectify this as we progress. You leave preview mode by clicking the form's Close box (or the platform-specific equivalent).



Previewing the Form

Now that we've created the form we will need to change some of its properties. (See the Using the Property Editor sidebar.)

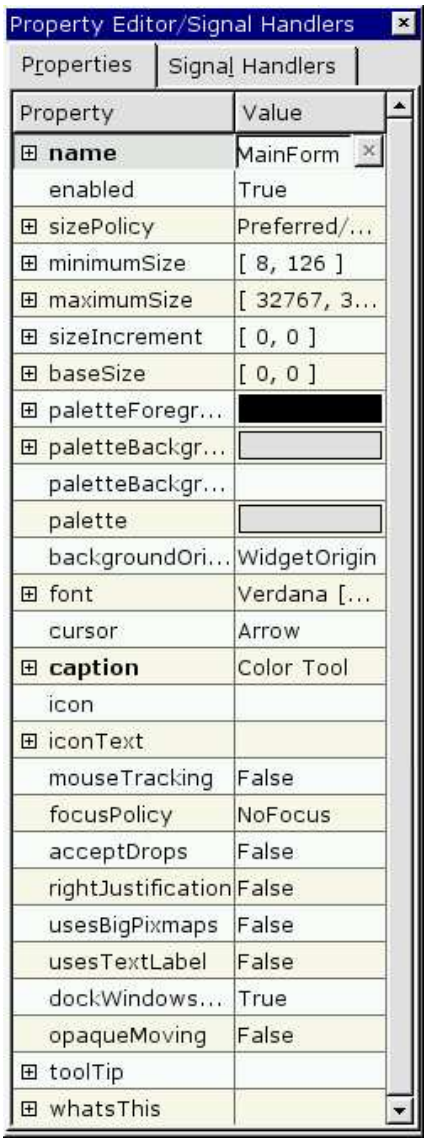
Using the Property Editor

The Property Editor has two columns, the Property column which lists property names and the Value column which lists the property values. Some property names have a plus sign '+' in a square to their left; this indicates that the property name is the collective name for a set of related properties. Click a form or widget to make the Property Editor show the form or widget's properties.

For example, click the *sizePolicy* property's plus sign; you will see four properties

appear indented below sizePolicy: hSizeType, vSizeType, horizontalStretch and verticalStretch. These properties are edited in the same way as any other properties.

If you want to change the same property to the same value for a whole set of widgets, (e.g. to give them all a common cursor, tooltip, colors, etc.), **Click** one of the widgets, then **Shift+Click** the others to select them all. (Alternatively, click the first widget's name in Object Explorer, then **Shift+Click** all the others in Object Explorer: this technique is especially useful for forms with lots of nested widgets and layouts.) The properties they have in common will be shown in the property editor, and any change made to one property will be made to that same property for all the selected widgets.



Property Editor

Some properties have simple values, for example, the *name* property has a text value, the *width* property (within *minimumSize*) has a numeric value. To change a text value click the existing text and type in your new text. To change a numeric value click the value and either type in a new number, or use the spin buttons to increase or decrease the existing number until it reaches the value you want.

Some properties have a fixed list of values, for example the *mouseTracking* property is boolean and can take the values True or False. The *cursor* property also has a fixed list of values. If you click the cursor property or the *mouseTracking* property the value will be shown in a drop down combobox; click the down arrow to see what values are available. Some properties have complex sets of values or special values; for example the *font* property and the *iconSet* property. If you click the font property an ellipsis button (**...**) will appear; click this button and a *Select Font* dialog will pop up which you can use to change any of the font settings. Other properties have ellipsis buttons which lead to different dialogs depending on what settings the property can have. For example, if you have a lot of text to enter for a *text* property you could click the ellipsis button to invoke the *Multi-line Edit* dialog.

The names of properties which have changed are shown in bold. If you've changed a property but want to revert it to its default value click the property's value and then click the red 'X' button to the right of the value. Some properties have an *initial* value, e.g. 'TextEdit1', but no default value; if you revert a property that has an initial value but no default value (by clicking the red 'X') the value will become empty unless the property, e.g. name, is not allowed to be empty.

The property editor fully supports Undo and Redo (**Ctrl+Z** and **Ctrl+Y**, also available from the **Edit** menu).

Setting the Form's Properties and Actions

Click the form to make all of its properties appear in the Property Editor. Change the form's *name* to "MainForm" and its *caption* to "Color Tool".

Now we'll need to delete some actions that the main window wizard created but that are not relevant to our application.

Click the Object Explorer's Members tab. Right click the filePrint() slot, then click Delete from the popup menu. In the same way delete the editUndo(), editRedo() and editPaste() slots. Later we'll see how to create new slots when we add further functionality to the application.

We also need to delete these actions in the Action Editor window. Right click the filePrintAction action, then click Delete Action from the popup menu. In the same way delete the editUndoAction, editRedoAction and editPasteAction actions.

Finally, we need to delete those separators in the form's menu that have become redundant because they separated actions that we've now deleted.

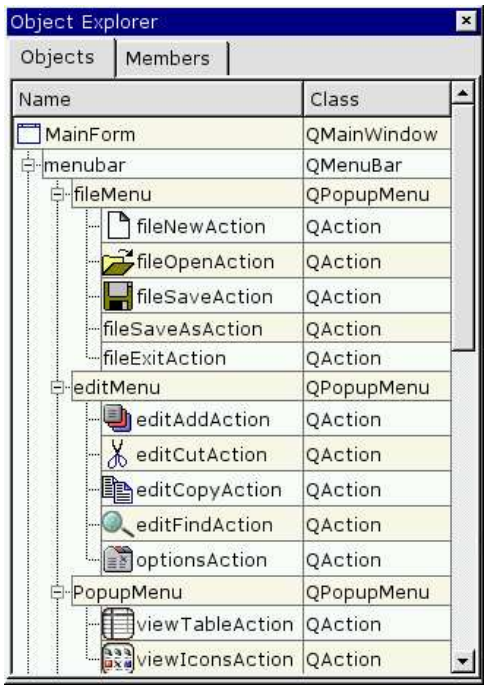
Click the form's **File** menu. (Note, we're clicking our newly created form's **File** menu, not *Qt Designer*'s **File** menu!) There are *two* separators above the Exit menu option (the **File|Print** option was in-between until we deleted it). Click one of these separators, then press the **Delete** key. Don't worry if you miss and delete a menu option by accident: if you delete the wrong thing click **Edit|Undo** to undelete. The form's **Edit** menu has a redundant separator at the top (the undo and redo options were there). Delete this separator in the same way. Again, don't worry if you delete a menu option by mistake, just press **Ctrl+Z** to undo.

Click **File|Save** to save the form.

The form can now be previewed by clicking **Preview|Preview Form** (or press **Ctrl+T**).

The Object Explorer

View the Object Explorer window by clicking **Window|Views|Object Explorer**. The Object Explorer has two tabs, the Objects tab which shows the object hierarchy, and the Members tab which shows the members you have added to the form. Click the name of a widget in the Objects tab to select the widget and show its properties in the Property Editor. It is easy to see and select widgets in the Object Explorer which is especially useful for forms that have many widgets or which use layouts. Multiple widgets can be selected by **Click**ing the first one then **Shift+Click**ing the others.



Object Explorer

In the original version of *Qt Designer* if you wanted to provide code for a form you had to subclass the form and put your code in the subclass. This version fully supports the subclassing approach, but now provides an alternative: placing your code directly into forms. Writing code in *Qt Designer* is not quite the same as subclassing, for example you cannot get direct access to the form's constructor or destructor. If you need code to be executed by the constructor create a slot called void init(); if it exists it will be called from the constructor. Similarly, if you need code to be executed before destruction create a slot called void destroy(). You can also add your own class variables which will be put in the generated constructor's code, and you can add forward declarations and any includes you require. To add a variable or declaration, right click the appropriate item, e.g. Class Variables, then click **New** then enter your text, e.g. QString m_filename. If one or more items exist, right click to pop up a menu that has New, Edit and Delete options. If you want to enter multiple items, e.g. multiple include files or multiple data members, it is easiest to right click in the relevant section, then click **Edit** to invoke an Edit dialog. To edit code, just click the name of a function to invoke the code editor. Code editing and creating slots are covered later in the chapter.

If you subclass the form you create your own .cpp files which can contain your own constructor, destructor, functions, slots, declarations and variables as your

requirements dictate. (See Subclassing for more information.)

Adding Custom Actions

We want to provide the user with actions that are specific to our application. We want to provide the ability to switch between the two views we will be offering, and allow the user to add colors and set their preferred options. We'll prepare the way by creating a new menu for the view options and by adding a separator to the toolbar.

Click "new menu" on the menu bar and type "&View" over the text. The & (ampersand) causes the following character to be underlined and to become an Alt-accelerator (i.e., in this case Alt+V will pop up the View menu).

The Menu Bar

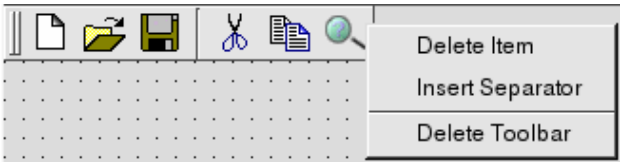
Duplicate Accelerators

In an application that has dialogs with lots of widgets it is easy to accidentally duplicate accelerators. *Qt Designer* provides the **Edit|Check Accelerators** menu option (**Alt+R**) which will highlight any two or more widgets which have the same accelerators, making it easy to spot the problem if it occurs.

Drag the View menu to the left of the "Help" menu and release it there. (A vertical red line indicates its position.)

Dragging the View Menu Item

We could create a new toolbar for the View menu items, but instead we'll put a separator at the end of the existing toolbar and add the View options after the separator. Right click the right-most toolbar button ("Find"), then click **Insert Separator**. Alternatively, we could have created an entirely new toolbar. See Creating and Populating Toolbars for more information on doing this.



Insert Separator in the Toolbar Menu

Creating and Populating Toolbars

A new toolbar is created by clicking to the right of the existing toolbars, then clicking **Add Toolbar**. The new toolbar is empty and is visible only by its *toolbar handle*. (Toolbar handle's are usually represented as a gray area containing either two thick vertical lines or with many small pits).

Toolbar Handle

Actions are added to toolbars simply by dragging them from the Action Editor to the toolbar, and dropping them on the toolbar in the position we want them. (The position is indicated by a vertical red line.)

Dragging the Action Group to the Toolbar

All the actions in an action group are added to a toolbar in one go, simply by dragging the action group from the Action Editor and dropping it on the toolbar.

Since toolbar buttons normally only show an image, all actions that are to be used in toolbars should have their *iconSet* property set to a suitable image.

Toolbar buttons and separators (usually represented as indented vertical gray lines), can be dragged and dropped into new positions in the toolbar at any time. Separators can be inserted by right clicking a toolbar button and clicking **Insert Separator**. Toolbar buttons and separators can be deleted by right clicking them and then clicking Delete Item. Toolbars can be deleted by right clicking their toolbar handle and then clicking Delete Toolbar.

If you preview an application you'll find that all the toolbars can be dragged to different docking points (top, left, right and bottom of a QMainWindow or subclass), or dragged out of the application as independent tool windows.

Adding Widgets to the Toolbar

Sometimes a simple button is insufficient for our needs. For example, if we wanted the user to be able to choose a font name and font size from the toolbar we might want to provide a direct means rather than having a toolbar button launch a font dialog.

It is perfectly feasible to add *ComboBox*es and *SpinBox*es to toolbars. For example, a *ComboBox* could be used to list the available font names and the *SpinBox* used to select a font size.

Although you can put any widget into a toolbar we recommend that widgets which can be associated with an action should *not* be added to the toolbar directly. For these widgets, i.e. menu items, toolbar buttons and lists of items, you should create an action (drop down action for a list of items), associate the action with the widget, and add the action to the toolbar. Widgets that can sensibly be inserted directly into a toolbar are *ComboBox*es, *SpinBox*es and *Line Edit*s.

Adding the Options Action

Right click the first action in the Action Editor, then click **New Action**. The Property Editor now shows the new action's properties. Change the action's *name* property to "optionsAction". Click the ellipsis button on the *iconSet* property to pop up the *Choose an Image* dialog. Click the **Add** button to invoke the *Choose Images...* dialog. Navigate to /tools/designer/examples/colortool/images; click the designer_tabwidget.png image. Click **Open** to use it and then click **OK** once you are in the *Choose an Image* dialog. Change the *text* property to "Options" and change the *menuText* property to "&Options...".

Click the Options action in the Action Editor and drag it to the Edit menu. The Edit menu will pop up; drag the Options action down the menu (a horizontal red line indicates its position), and drop it at the end after the "Find" item.

Alternative Approach to Adding the Options Action

Click Edit on the menu bar and then click "new item", located after the Find menu item. Type "&Options" over "new item" to rename it and press **Enter**. Move the arrow key to the space to the left of the Options menu item (the pixmap field) and press **Enter**. The *Choose an Image* dialog pops up. Click the **Add** button to invoke the *Choose Images...* dialog. Navigate to /tools/designer/examples/colortool/images; click the designer_tabwidget.png image. Click **Open** to use the image and then click **OK** once you are in the *Choose an Image* dialog. The pixmap now appears next to the Options item in the menu.

The options action ought to be visually separated from the other Edit menu options. Click the form's Edit menu, then click and drag the "new separator" item to the space above the Options item.

Since we also want to make this option available from the toolbar, click the Options action in the Action Editor and drag it to the toolbar. Drop it to the right of the magnifying glass (Find) toolbar button (after the separator); a horizontal red line indicates its position during the drag.

We'll connect and code this action later.

Adding the Add Action

Right click the first action in the Action Editor, then click **New Action**. Change the action's *name* property to "editAddAction". Change its *iconSet* property to designer_widgetstack.png. Change the *text* property to "Add" and the *menuText* property to "&Add...". Change the *accel* property to "Ctrl+A" (press **CTRL+A** and the key combination will automatically appear in the field).

Click the Add action and drag it to be the first item in the Edit menu. (Drag it to the edit menu and drop it when the horizontal red line is above the "Cut" menu item.)

Alternative Approach to Adding the Add Action

Click Edit on the menu bar and then click "new item", located after the Find menu item. Type "&Add" over "new item" to rename it and press **Enter**. Move the arrow key to the space to the left of the Add item and press **Enter**. The *Choose an Image* dialog pops up.Click the **Add** button to invoke the *Choose Images...* dialog. Navigate to /tools/designer/examples/colortool/images; click the designer_tabwidget.png image. Click **Open** to use the image and then click **OK** once you are in the *Choose an Image* dialog. The pixmap now appears next to the Options item in the menu. Finally, move the arrow key to the space to the right of the Add item and and press "Ctrl+A". The accelerator key combination now appears next to the Add menu item.

Tidying Up

We're going to use "Cut" for deleting colors, so we'll change the user-visible name to "Delete" to make its meaning clearer. Click the editCutAction in the Action Editor to make its properties appear in the Property Editor. Change its *text* property to "Delete" and change its *menuText* property to "&Delete".

A side-effect of the above change is that **Alt+C** (originally used for "Cut") is now unused. Click the editCopyAction action in the Action Editor, and change its *menuText* property to "&Copy".

Alternative Approach to Renaming Actions

To change the name of the Cut action to "Delete", click Edit on the menu bar and then click "Cut". Type "&Delete" over "Cut" and press enter.

To change name of the Copy action to "&Copy", click Edit on the menu bar and then click "Copy". Type "&Copy" over "Copy" and press enter.

We can always check to see if there are any accelerator conflicts by clicking **Edit|Check Accelerators** (or **Alt+R**).

Adding an Action Group

We want to provide the user with a choice of views, but since they can only use one view at a time we need to ensure that the menu options and toolbar buttons they use to switch between views always stay in sync. We don't have to write any code to achieve this: we simply put the relevant actions in an action group and let Qt take care of the details.

Right click an action in the Action Editor, then click **New Action Group**. The action group's properties are now showing in the Property Editor. Change the action group's *name* property to "viewActionGroup", and change its *text* property to "View". We want the action group to be *exclusive*, i.e. for only one of its actions to be "on" at any one time; but there's no need to set the *exclusive* property since it defaults to True which is what we want.

We'll now create the view actions. The process is virtually the same as for actions that are not in an action group; the only difference is that when we right click to pop up the context menu, we *must* right click the relevant action group, not just anything in the Action Editor.

Right click the viewActionGroup, then click **New Action**. Change this action's *name* property to "viewTableAction". Set its *toggleAction* property to True and set its *on* property to True. We want it to be a toggle action because either the user is using this view (it is "on") or another view (it is "off"). We set this action to "on" because it will be the default view. Change its *iconSet* property to designer_table.png. Change the *text* property to "View Table" and the *menuText* property to "View &Table". Change the *accel* property to "Ctrl+T", and set the *toolTip* property to "View Table (Ctrl+T)". When the user clicks the **View** menu and hovers the mouse over the "View Table" option the tool tip will appear in the status bar. Similarly when the user hovers the mouse over the "View Table" toolbar button, the tool tip text will appear both in the status bar and in a temporary yellow label next to the toolbar button.

Right click the viewActionGroup, then click **New Action**. Change this action's *name* property to "viewIconsAction". Set its *toggleAction* property to True. Change its *iconSet* property to designer_iconview.png. Change the *text* property to "View Icons" and the *menuText* property to "View &Icons". Set the *accel* property to "Ctrl+I" and change the *toolTip* property to "View Icons (Ctrl+I)".

Note that the Action Editor window is dockable, so if you don't want it to float freely you can drag it to one of *Qt Designer*'s dock areas (top, left, right, bottom of the main window) if preferred.

Using an Action Group

Now that we've created the view actions we need to make them available to the

user.

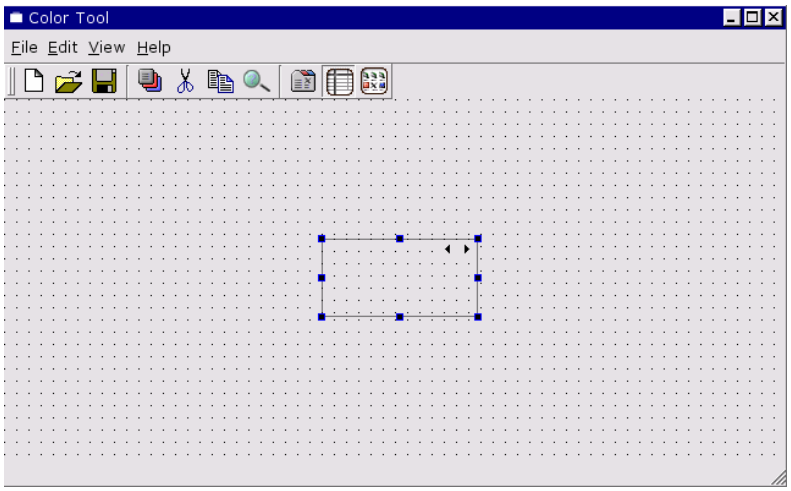
Click the viewActionGroup action group in the Action Editor, and drag it to the View menu; drop it on this menu (when the horizontal red line appears beneath the View menu). Because we dragged the action group, *all* its actions (in our case the viewTableAction and viewIconsAction) are added to the relevant menu. We'll also make the view actions available on the toolbar. Click the viewActionGroup once again, and drag it to the toolbar; drop it the right of the separator at the far right of the toolbar, and drop it on the toolbar's edge. (Again, a vertical red line will indicate the position.)

Don't forget that you can preview to see things in action with **Ctrl+T**, and to click **File|Save** (or press **Ctrl+S**) regularly! If you preview now you will find that if you click the view toolbar buttons and menu options that both the toolbar buttons and the menu items automatically stay in sync.

CREATING THE MAIN WIDGET

Most main-window style applications consist of a menu bar, a toolbar, a status bar and a central widget. We've already created a menu bar and toolbar, and since we've created a QMainWindow (via the main window wizard), we also have a status bar. Widgets commonly used as an application's main widget are QListView (which provides a tree view), QTable and QTextEdit. Since we want to provide our users with two different views of the same data, we'll use a QWidgetStack as our main widget. The QWidgetStack has no visual representation of its own; you place one or more widgets on each QWidgetStack "page", as if each page was a form in its own right, and then provide the user with some mechanism for switching between pages. (This is similar in principle to using a QTabWidget.) We want to provide our users with two views: a tabular view that lists colors and their names, and an icon-based view that shows color swatches. In our example we only place a single widget on each QWidgetStack page; but this merely reflects the application's design -- we could have placed any number of widgets on each page.

Click the Toolbox's Containers button, then click WidgetStack. Click approximately in the middle of the form to place the widget stack. Change the widget stack's *name* property to "colorWidgetStack".

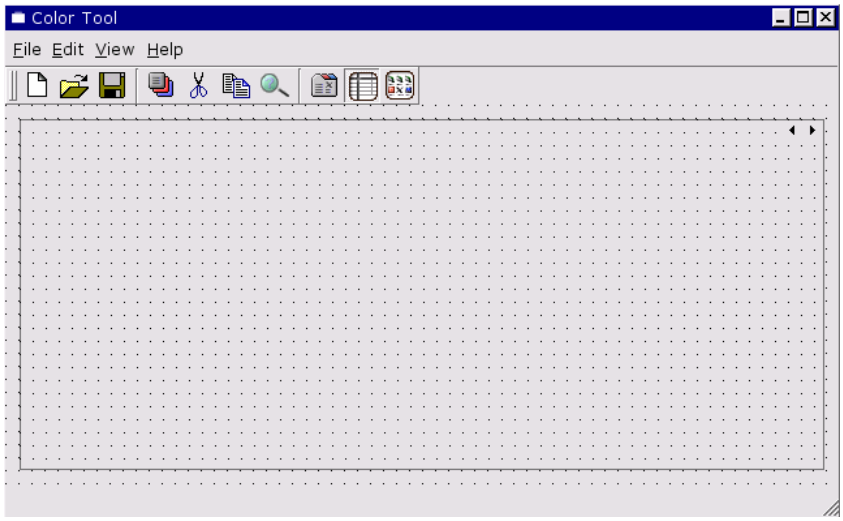


Widget Placement

When placing widgets on forms using *Qt Designer*, you only need to place things in *approximately* the right place. And there is no need to worry about the size of the widgets placed. If, for example, you place a label and then change its text so that the text doesn't fit, this doesn't matter. The reason we don't have to care about precise positions and sizes is that *Qt Designer* uses Qt's layout classes to lay out forms automatically: we just have to select sets of widgets and tell *Qt Designer* how they should be laid out in relation to each other, e.g. vertically, one above the other, or horizontally, side by side, or in a grid, and *Qt Designer* will lay them out and size them appropriately.

In this chapter we only make the most minimal use of *Qt Designer*'s layout facilities. We make more use of layouts and provide more information in chapter two, where we create several dialogs.

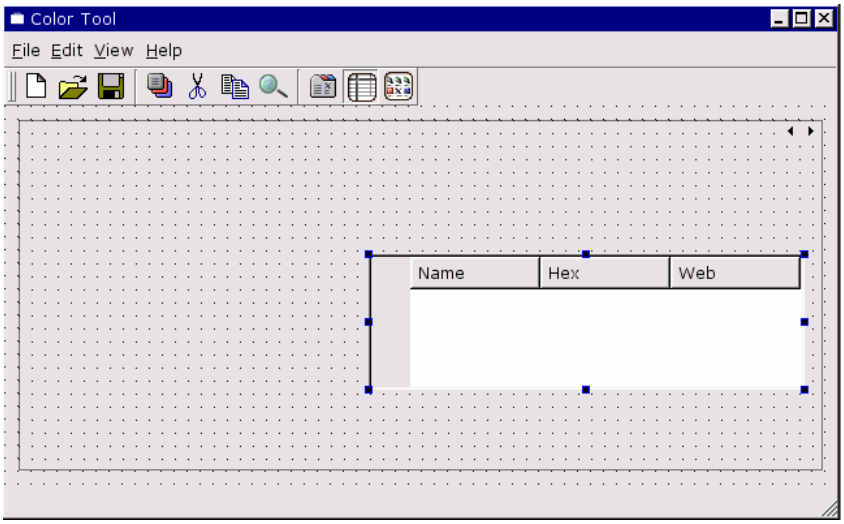
Click the form itself, then click the **Lay Out Vertically** toolbar button. The widget stack now fills the entire form. We're now ready to populate the widget stack's pages with widgets.



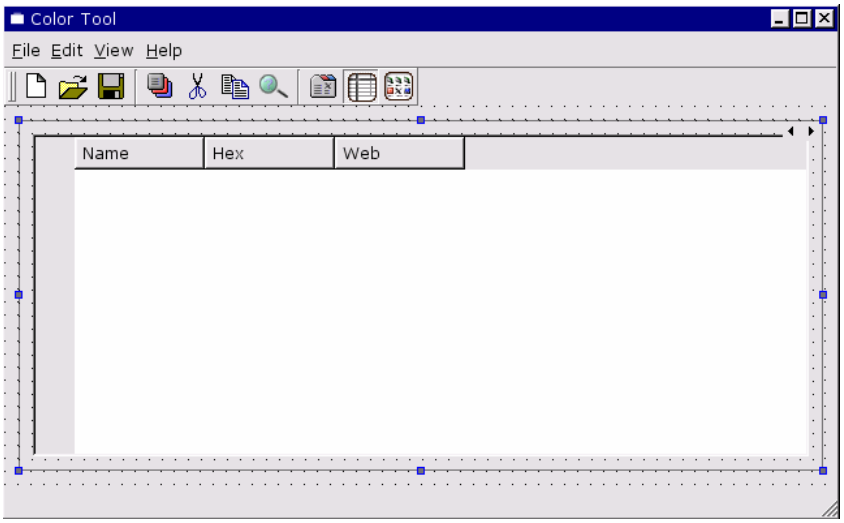
Click the Toolbox's Views button. Click Table, then click approximately in the middle of the widget stack. Change the table's *name* property to "colorTable", change its *numRows* property to "0", and its *readOnly* property to "True".

If you right click a widget to pop up its context menu, in most cases the first item will be an "Edit" option. The Table widget is no different in this respect, and its "Edit" option leads to a dialog through which columns and rows can have their titles changed, etc.

Right click the table, then click **Edit...** to invoke the *Edit Table* dialog. Change the Label for column 1 to "Name". Click "2" in the Columns list so that column 2's label is shown in the Label line edit. Change column 2's label to "Hex". In the same way change column 3's label to "Web". (The reference section provides full information on this dialog.) Click **OK** to close the dialog.

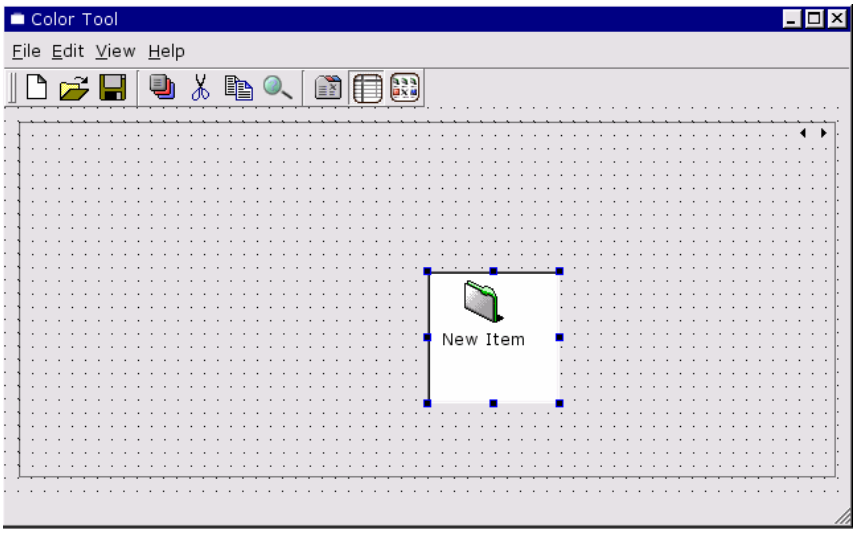


Click the widget stack, then click the **Lay Out Vertically** toolbar button. The table now fits inside the widget stack, and will resize with the widget stack (which in turn will resize with the form: try clicking **Ctrl+T** to preview and resize the previewed form).



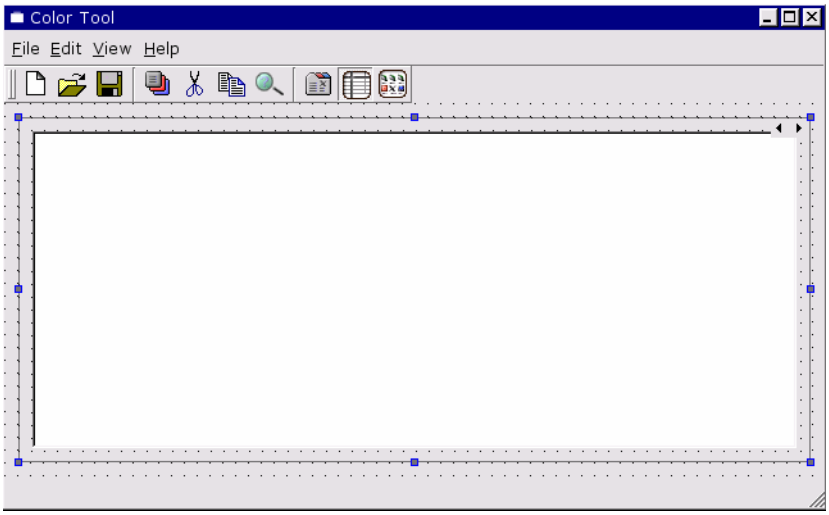
Click the "WStackPage" object in Object Explorer. Change its *name* property to "tablePage".

We're now ready to create the next page. Right click the widget stack, then click **Add Page** on the context menu. The table has "disappeared", or rather the new widget stack page obscures the first widget stack page which contains the table. Click IconView in the Toolbox, then click approximately in the middle of the widget stack. Change the IconView's *name* property to "colorIconView" and change its *resizeMode* property to "Adjust". We want our color swatches to appear in neat columns so change the *gridX* property to "100".



It is often useful to create IconView items during design, but it isn't appropriate for our application. Right click the IconView to pop up its context menu, then click **Edit...** to invoke the *Edit IconView* dialog. Click **Delete Item** to delete the default item, then click **OK**.

Click the widget stack, then click the **Lay Out Vertically** toolbar button. The icon view now fits inside the widget stack.



Click the "WStackPage" object in Object Explorer. Change its name to "iconsPage".

Right click the widget stack, then click **Previous Page**.

That completes the user interface design for our application's main window. Note that if you preview the form clicking the "View" menu options and toolbar buttons has no effect. This is because we haven't written any code to be executed when the actions triggered by these menu options and toolbar buttons occur. We'll write the necessary code in the next section.

WRITING THE CODE

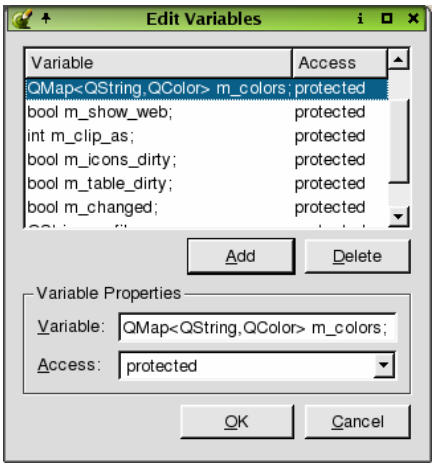
There are two approaches that can be taken when it comes to writing code for forms designed with *Qt Designer*. The original approach is to create a subclass of every form you create and put all your code in the subclass. Since Qt 3.0, *Qt*

Designer has provided an alternative: you can write your code directly in *Qt Designer* using the code editor. See The Designer Approach for a comparative review. For this example we will write all the code inside *Qt Designer*; for an example of the subclassing approach see Subclassing and Dynamic Dialogs.

Before we launch into writing code we need to create some form variables. For example, we need to keep track of whether a view needs updating (because the user loaded a new set of colors, or added or deleted colors in the other view).

Adding Member Variables

Click Object Explorer's Members tab. Right click "Class Variables" (towards the bottom), then click **Edit**. The *Edit Class Variables* dialog appears. Click the **Add** button, and type in "QMap<QString,QColor> m_colors". We will use this map to relate user color names to colors. Click the **Add** button again, and type in "bool m_changed". We'll use this variable to keep track of whether the data has changed or not; this is useful for offering the user a prompt to save unsaved changes when they exit or open a new file, for example.



In the same way add "QString m_filename" so that we can keep track of the file the user has open. Add "bool m_table_dirty" and "bool m_icons_dirty". If the user adds a color when viewing the table we'll mark the icons as 'dirty' so that the icon view will be updated if the user changes to view the icons, and vice versa. Add "bool m_show_web" -- we'll use this to record whether or not the user wants a column in the table to indicate which colors are web colors. Add "int m_clip_as" -- we'll use this to choose what to put on the clipboard when the user clicks **File|Copy**. We'll keep a pointer to the global clipboard, so add "QClipboard *clipboard". Finally add "QStringList m_comments". This is used for loading and saving color files and is explained later.

You should now have the following variables:

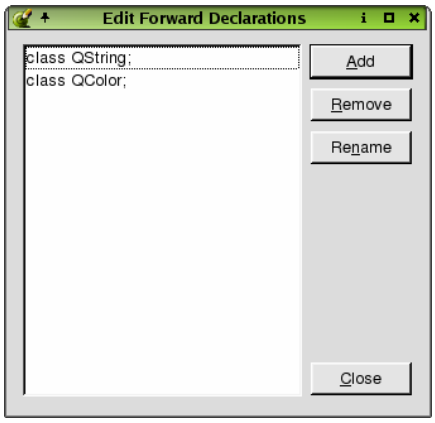
- QMap<QString,QColor> m_colors;
- bool m_changed;
- QString m_filename;
- bool m_table_dirty;
- bool m_icons_dirty;
- bool m_show_web;

- int m_clip_as;
- QClipboard *clipboard;
- QStringList m_comments;

Press **Enter**, to confirm the last variable, then click **OK** to close the dialog. All the variables now appear in Object Explorer's Members tab.

Adding Forward Declarations

Some of the variables we've created are of classes that need forward declarations. Right click Forward Declarations (in Object Explorer's Members tab), then click **Edit**. This pops up the *Edit Forward Declarations* dialog. This dialog works the same way as the *Edit Class Variables* dialog that we've just used. Add the following forward declarations: "class QString;" and "class QColor;". Close the dialog and the forward declarations appear in Object Explorer.

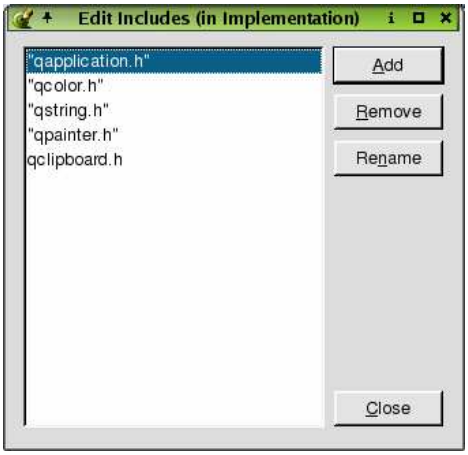


You should now have the following forward declarations:

- class QString;
- class QColor;

Adding Includes

Our form will also need some included files. Includes may be added in the declaration, or (for preference) in the implementation. Right click "Includes (in Implementation)", then click **Edit**. Use the dialog that pops up to enter "qcolor.h" and "qstring.h". Since we're going to use the clipboard we'll need access to the global clipboard object via QApplication, so also add "qapplication.h" and "qclipboard.h". We'll also be doing some drawing (e.g. the color swatches), so add "qpainter.h" too, then close the dialog.



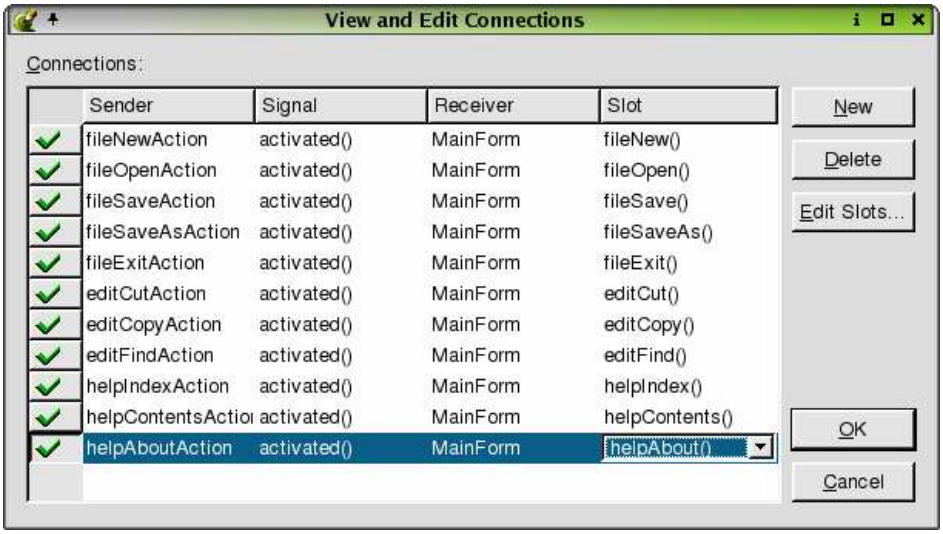
When entering include files you can include double quotes or angle brackets if you wish; if you don't use either *Qt Designer* will put in double quotes automatically.

You should now have added the following includes (in implementation):

- "qcolor.h"
- "qstring.h"
- "qapplication.h"
- "qclipboard.h"
- "qpainter.h"

Signals and Slots Connections

Most of the signals and slots connections were created automatically by the main window wizard when we created the main form. We have added some new actions since then, and we need to ensure that they are connected to slots so that we can code their behavior.



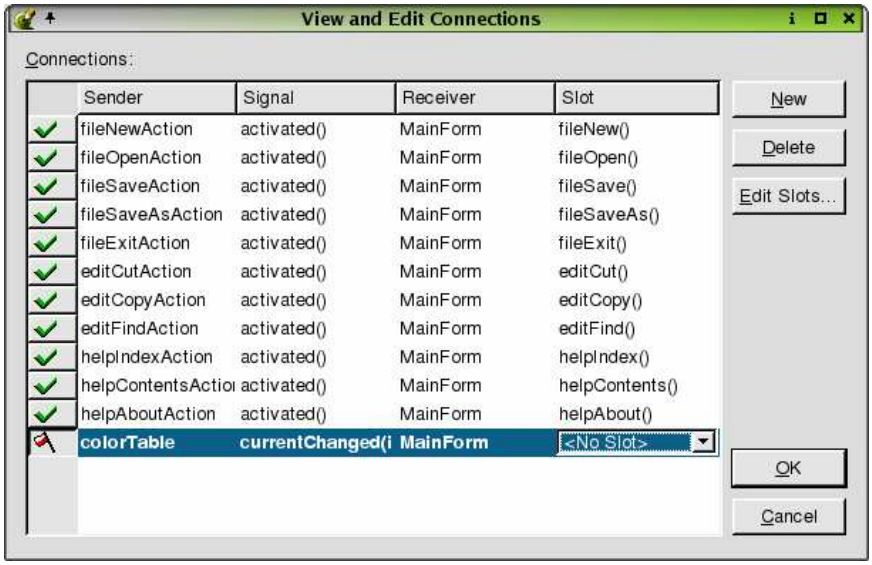
Creating Signals and Slots Connections

Click **Edit|Connections** to invoke the *View and Edit Connections* dialog.

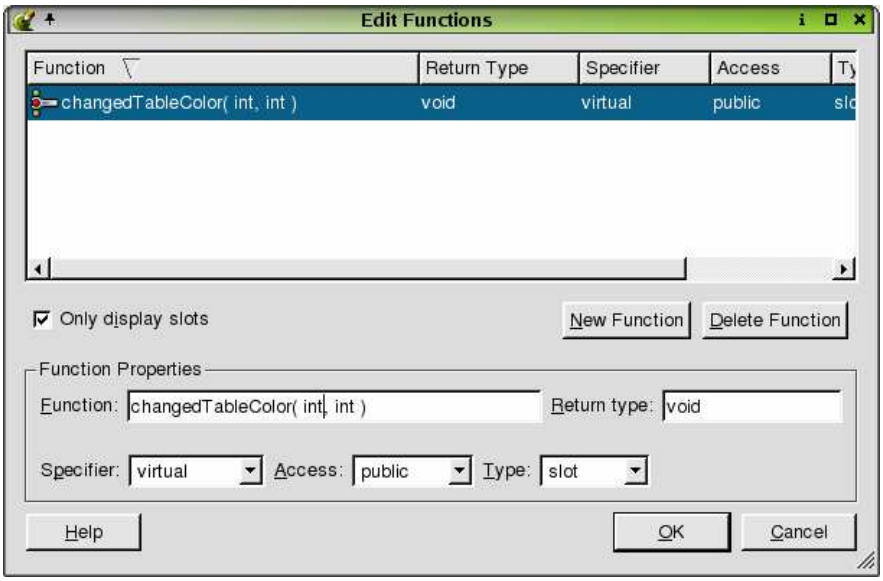
The use of this dialog usually follows the same pattern. We click **New** to create a new connection, then we select the Sender widget, the sender's Signal and the Receiver (usually the form). If we want to use a pre-defined slot, we select that

slot; otherwise we click **Edit Slots...** create a new slot on-the-fly, and select the newly created slot. (The old method of clicking and dragging to create connections is still supported, but the new method is a lot faster and easier, especially for creating lots of connections in one go.)

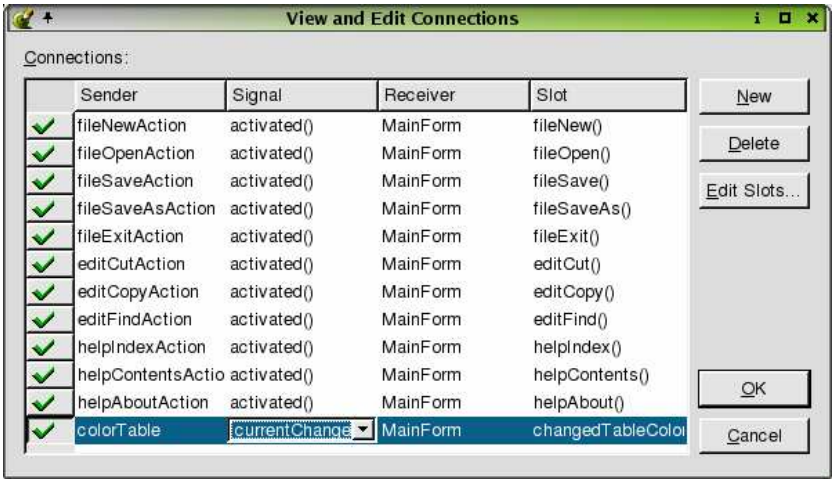
We want to update the status bar so that the user can see information about the color they're on. Click **Edit|Connections** to invoke the *View and Edit Connections* dialog. Click **New** to create a new connection. Change the Sender to "colorTable" and the Signal to "currentChanged(int,int)". Change the Receiver to "MainForm".



We want to connect to our own custom slot which we haven't yet created. Click the **Edit Slots...** button to invoke the *Edit Functions* dialog. Click **New Function** and change the slot name to "changedTableColor(int,int)". Click **OK** to close the dialog.



Now change the Slot in the *View and Edit Connections* dialog to our newly created "changedTableColor(int,int)" slot.



Click **New** to create a new connection. Change the Sender to "colorIconView" and the Signal to "currentChanged(QIconViewItem*)". Change the Receiver to "MainForm". Click the **Edit Slots...** button to invoke the *Edit Functions* dialog. Click **New Function** and change the slot name to "changedIconColor(QIconViewItem*)". Click **OK** to close the dialog. Now change the Slot in the *View and Edit Connections* dialog to "changedIconColor(QIconViewItem*)".

Now we can implement our changedTableColor() and changedIconColor() slots to update the status bar with details about the current color.

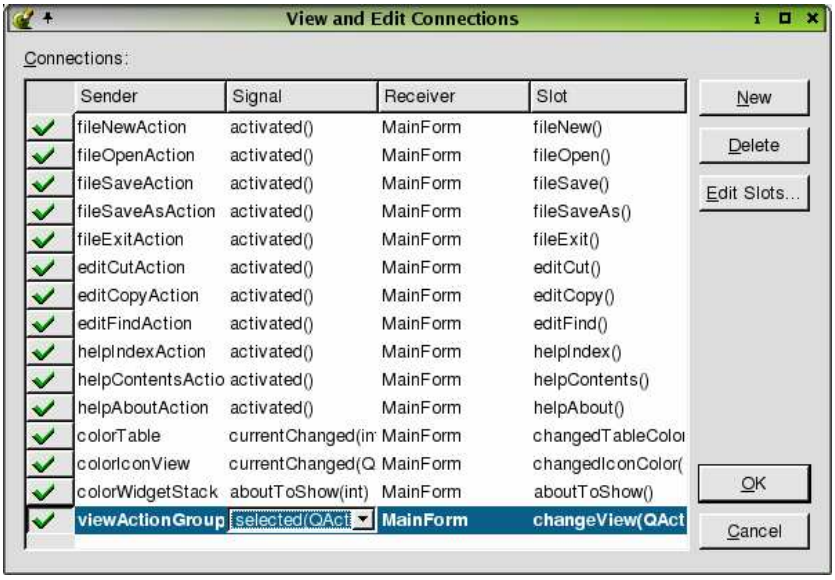
We also want to ensure that when the user changes view, the colors shown in the view are correct. For example, if the user deleted a color in the table view and changed to the icon view, we must ensure that the icon view does not show the deleted color.

Click **New** to create a new connection. Change the Sender to "colorWidgetStack", the Signal to "aboutToShow(int)", and the Receiver to "MainForm". Create a new slot called "aboutToShow()" and make this the Slot that the widget stack's "aboutToShow(int)" signal connects to. The signal includes the ID of the widget that is about to be shown; but we don't need it so we create a slot that doesn't take any parameters.

Once crucial piece of functionality is to allow the user to switch between views. We could connect each of the view actions separately, but it is more convenient (and easier to extend) if we connect the action group as a whole.

Create a new connection with the "viewActionGroup" as the Sender. Change the Signal to "selected(QAction*)" and change the Receiver to "MainForm". Create a slot called "changeView(QAction*)" and make this the slot that the signal connects to.

Click **OK** to close the *View and Edit Connections* dialog. We are now ready to write the code.



Creating main.cpp

Now that we've entered some of the code it would be nice to build and run the application to get a feel for the progress we've made. To do this we need to create a main() function. In Qt we typically create a small main.cpp file for the main() function. We can ask *Qt Designer* to create this file for us.

Click **File|New** to invoke the *New File* dialog. Click "C++ Main-File", then click OK. The *Configure Main-File* dialog appears, listing the all the forms in the project. We've only got one form, "MainForm", so it is already highlighted. Click **OK** to create a main.cpp file that loads our MainForm.

```
#include <qapplication.h>
#include "mainform.h"

int main( int argc, char ** argv )
{
    QApplication a( argc, argv );
    MainForm *w = new MainForm;
    w->show();
    return a.exec();
}
```

When *Qt Designer* generates a main.cpp file it includes this line:

```
a.connect( &a, SIGNAL( lastWindowClosed() ), &a, SLOT( quit() ) );
```

If we left this code as-is, the user could by-pass our own termination code by clicking the main window's close (X) button. Since we want to give the user the option to save any unsaved changes we need to ensure that we intercept any attempt to close the application. To achieve this we delete the connection and add a new slot, closeEvent() which will intercept attempts to close the application and call our fileExit() function.

Click main.cpp in the Project Overview window. The file will appear in an editing window. Delete the connect line.

Click mainform.ui.h in the Project Overview window; (you may need to click mainform.ui first to reveal mainform.ui.h). Right click "fileExit()" in Object Explorer's Members list (under Slots, public), then click **Goto Implementation**. Add the

following slot above the fileExit() slot:

```
void MainForm::closeEvent( QCloseEvent * )
{
   fileExit();
}
```

Now, whatever the user clicks to close the application, our fileExit() slot will be called. We'll code the fileExit() slot right now:

```
void MainForm::fileExit()
{
      QApplication::exit( 0 );
}
```

This ensures that our application will cleanly terminate. Later we'll revise this function to give the user the opportunity to save any unsaved data.

Building and Running

We now have some code in the application and a main.cpp containing the main() function, so we should be able to compile, link and run the application.

Click **File|Save** to ensure that all our work is saved to disk. Open a console (e.g. an xterm or DOS window), change directory to where you have saved the colortool project, and run qmake to generate a Makefile:

```
qmake -o Makefile colortool.pro
```

Now make the project (run nmake on Windows, make on other platforms). Providing you commented out the "findForm" and "loadSettings" lines in the init() function, the program should build. (If it doesn't build see the Troubleshooting section.)

Once the make has finished, run the program. You still can't change views since we haven't written the code for that yet, but it does create a default set of colors. You can terminate the application by clicking the close (X) button or by clicking **File|Exit**.

Editing the Code: Updating the Status Bar

We want to show information about the current color in the status bar, and we want to ensure that when the user changes their view or loads in a color file, the relevant view is updated.

aboutToShow()

```
void MainForm::aboutToShow()
{
   populate();
}
```

We could have made populate() a slot and connected directly to it. We've used the indirection because it's clearer and in a real application there would probably be more to do in this slot.

changedTableColor()

```
void MainForm::changedTableColor( int row, int )
{
   changedColor( colorTable->text( row, COL_NAME ) );
}
```

We connected to this slot so that we'd know whenever the user moved or clicked in the table view. We call the changedColor() function (which we'll see in a moment)

with the name of the current color. Note that we don't care about the column argument, so we could have left it out. Don't forget to name the changedTableColor parameter to "int row".

changedIconColor()

```
void MainForm::changedIconColor( QIconViewItem *item )
{
    changedColor( item->text() );
}
```

This slot is connected for the same purpose as changedTableColor(), above. It also calls changedColor() with the name of the current color. (If you're cutting and pasting the code don't forget to name the QIconViewItem parameter "item".)

changedColor()

This is a function that we need to write from scratch. Simply enter its code into *Qt Designer*'s code editor and it will automatically appear in Object Explorer's Members tab (under Functions, public).

By default any function that it typed directly into the code editor becomes a public function. To change this, right click the function's name in Object Explorer's Members list, and click **Properties** to invoke the *Edit Functions* dialog. This dialog can be used to change various attributes of the function, including changing it into a slot.

```
void MainForm::changedColor( const QString& name )
{
    QColor color = m_colors[name];
    int r = color.red();
    int g = color.green();
    int b = color.blue();
    statusBar()->message( QString( "%1 \"%2\" (%3,%4,%5)%6 {%7 %8 %9}" ).
                          arg( name ).
                          arg( color.name().upper() ).
                          arg( r ).arg( g ).arg( b ).
                          arg( isWebColor( color ) ? " web" : "" ).
                          arg( r / 255.0, 1, 'f', 3 ).
                          arg( g / 255.0, 1, 'f', 3 ).
                          arg( b / 255.0, 1, 'f', 3 )
                          );
}
```

This function looks up the color name in the colors map and retrieves the color the name refers to. It then displays the name, hex value and whether the color is a web color in the status bar.

Note that QMainWindow only creates a status bar if you actually use one. Since we haven't used one up until now we've had no problem, but if we were to try compiling we'd get an error because we're now using a status bar but haven't declared the relevant header. Click Object Explorer's Members tab and add a "qstatusbar.h" to the "Includes (In Implementation)" section. (Right click "Includes (In Implementation)", click **New**, enter "qstatusbar.h" then press **Enter**.)

You should now have added the following declaration to your includes (in implementation):

- "qstatusbar.h"

Try saving (press **Ctrl+S**), making and running the application. Move to different

colors and see the status bar indicating the color you are on. (If it doesn't build see the Troubleshooting section.)

Changing Views

Up to now we have not yet been able to see the icon view in action because there's been no code in place to switch views. We'll address this issue now.

We have already created a changeView() slot that is called when the user clicks one of the view toolbar buttons or menu options, so we just need to write in the code.

```
void MainForm::changeView(QAction* action)
{
    if ( action == viewTableAction )
        colorWidgetStack->raiseWidget( tablePage );
    else
        colorWidgetStack->raiseWidget( iconsPage );
}
```

(If you're cutting and pasting the code don't forget to name the QAction parameter "action".)

Editing the Code: File Handling

Since the X Consortium has already defined a file format for relating colors to color names we will use their format rather than creating one specially for the application. This has the advantage that we will be able to read and write rgb.txt, and that our format will be familiar to many users.

fileNew()

```
void MainForm::fileNew()
{
    if ( okToClear() ) {
        m_filename = "";
        m_changed = FALSE;
        m_table_dirty = TRUE;
        m_icons_dirty = TRUE;
        clearData( FALSE );
    }
}
```

This function doesn't load or save any data; it simply checks to see if it is okay to clear the existing data (with the call to okToClear() which we'll look at next), and if it is okay, it initializes the form.

okToClear()

Before we can create a new set of colors, or load an existing set, we must check to see if there are any unsaved changes. If there are, we must give the user the opportunity of saving their data. That's what this function does.

```
bool MainForm::okToClear()
{
    if ( m_changed ) {
        QString msg;
        if ( m_filename.isEmpty() )
            msg = "Unnamed colors ";
        else
            msg = QString( "Colors '%1'\n" ).arg( m_filename );
        msg += QString( "has been changed." );
        int ans = QMessageBox::information(
                    this,
```

```
                "Color Tool -- Unsaved Changes",
                msg, "&Save", "Cancel", "&Abandon",
                0, 1 );
        if ( ans == 0 )
            fileSave();
        else if ( ans == 1 )
            return FALSE;
    }

    return TRUE;
}
```

If the data has changed (m_changed is TRUE), we present the user with a message box offering the option of saving their data, or cancelling the current operation (e.g. not loading a new file, or not creating a new set of colors), or abandoning their changes and continuing. We make the **Save** button the default button (pressed by **Enter**) and the **Cancel** button the escape button (pressed by **Esc**).

Since we're using a QMessageBox we need to include the relevant header. (Right click "Includes (in Implementation)", then click **New**. Type "qmessagebox.h" and press **Enter**.)

You should now have added the following declaration to your includes (in implementation):

- "qmessagebox.h"

fileOpen()

```
void MainForm::fileOpen()
{
    if ( ! okToClear() )
        return;

    QString filename = QFileDialog::getOpenFileName(
                    QString::null, "Colors (*.txt)", this,
                    "file open", "Color Tool -- File Open" );
    if ( ! filename.isEmpty() )
        load( filename );
    else
        statusBar()->message( "File Open abandoned", 2000 );
}
```

If it isn't okay to clear the data (i.e. the user has unsaved changes and clicked **Cancel** in the message box popped up by okToClear()), we simply return. Otherwise we ask the user for a filename using one of QFileDialog's static functions, and if we got the filename we attempt to load the file.

Since we're using a QFileDialog we need to include the relevant header. (Right click "Includes (in Implementation)", then click **New**. Type "qfiledialog.h" and press **Enter**.)

You should now have added the following declaration to your includes (in implementation):

- "qfiledialog.h"

load()

```
void MainForm::load( const QString& filename )
{
    clearData( FALSE );
```

```
    m_filename = filename;
    QRegExp regex( "^\\s*(\\d+)\\s+(\\d+)\\s+(\\d+)\\s+(\\S+.*)$" );
    QFile file( filename );
    if ( file.open( IO_ReadOnly ) ) {
        statusBar()->message( QString( "Loading '%1'..." ).
                              arg( filename ) );
        QTextStream stream( &file );
        QString line;
        while ( ! stream.eof() ) {
            line = stream.readLine();
            if ( regex.search( line ) == -1 )
                m_comments += line;
            else
                m_colors[regex.cap( 4 )] = QColor(
                                    regex.cap( 1 ).toInt(),
                                    regex.cap( 2 ).toInt(),
                                    regex.cap( 3 ).toInt() );
        }
        file.close();
        m_filename = filename;
        setCaption( QString( "Color Tool -- %1" ).arg( m_filename ) );
        statusBar()->message( QString( "Loaded '%1'" ).
                              arg( m_filename ), 3000 );
        QWidget *visible = colorWidgetStack->visibleWidget();
        m_icons_dirty = ! ( m_table_dirty = ( visible == tablePage ) );
        populate();
        m_icons_dirty = ! ( m_table_dirty = ( visible != tablePage ) );
        m_changed = FALSE;
    }
    else
        statusBar()->message( QString( "Failed to load '%1'" ).
                              arg( m_filename ), 3000 );
}
```

Before loading new data, we clear out any existing data. The format of an rgb.txt file is:

RED WHITESPACE GREEN WHITESPACE BLUE WHITESPACE NAME

Where RED, GREEN and BLUE are decimal numbers in the range 0..255 taking up three characters padded with leading spaces where necessary. The WHITESPACE between the colors is usually a single space, and between BLUE and the NAME two tabs. The NAME may include whitespace. For example:

```
  0 191 255          deep sky blue
176  48  96          maroon
199  21 133           medium violet red
```

The file may also include comment lines; these begin with '!' for example.

There are numerous approaches we could have taken to parsing these files, but we've opted for a simple regular expression (regex). The regex is more "liberal" regarding the whitespace in the input than the format demands.

If a line matches the regex we create a new entry in the m_colors QMap, setting its text to be the name of the color (regex.cap(4)), and its value to be a new QColor created from the red, green and blue values. Lines that don't match the regex are treated as comments and are stored in the m_comments string list. (When we save the file we write all the comments out first even if they appeared in the middle of the file.)

Once we've populated the m_colors map we mark the visible view as "dirty" and call populate() to update it. We then mark the visible view as not dirty and the non-visible view as dirty. This ensures that when user changes the view, the view they switch to will be updated. We could have simply marked both views as dirty and updated them both, but it is more efficient to update "lazily", after all the user may only ever use one view, so why waste their time updating the other one.

Since we're using QFile and QRegExp we need to include the relevant headers. (Right click "Includes (in Implementation)", then click **New**. Type "qfile.h" and press **Enter**. Repeat this process to add "qregexp.h".)

You should now have added the following declarations to your includes (in implementation):

- "qfile.h"
- "qregexp.h"

The Regular Expression

The regex we've used can be broken up into the following pieces:

```
Regex:    ^ \\s* (\\d+) \\s+ (\\d+) \\s+ (\\d+) \\s+ (\\S+.*) $
Pieces:   A B    C      D    C      D    C      D    E        F
Captures:        cap(1)      cap(2)      cap(3)      cap(4)
```

Piece A says the regex must match from the beginning of the string, and piece F says the regex must match to the end of the string: so the regex must match the whole string or not match at all. The 'B' piece matches zero or more whitespaces (i.e. any leading whitespace), and the D pieces match one or more whitespaces (i.e. the gaps between each number). The 'C' pieces match one or more digits, i.e. the numbers. Piece E matches one or more non-whitespace followed by anything else, i.e. the name of the color.

The parentheses are used to *capture* the parts of the match that they enclose. The captured parts are numbered from 1.

For more information on regexes see the QRegExp documentation.

fileSaveAs()

```
void MainForm::fileSaveAs()
{
    QString filename = QFileDialog::getSaveFileName(
                        QString::null, "Colors (*.txt)", this,
                        "file save as", "Color Tool -- File Save As" );
    if ( ! filename.isEmpty() ) {
        int ans = 0;
        if ( QFile::exists( filename ) )
            ans = QMessageBox::warning(
                        this, "Color Tool -- Overwrite File",
                        QString( "Overwrite\n'%1'?" ).
                            arg( filename ),
                        "&Yes", "&No", QString::null, 1, 1 );
        if ( ans == 0 ) {
            m_filename = filename;
            fileSave();
            return;
        }
    }
    statusBar()->message( "Saving abandoned", 2000 );
```

```
}
```

If the user attempts to save data that has been edited but not saved previously, or if they want to save some existing data under a new name, this slot is called. The user is presented with a standard file dialog which they can use to choose a filename. If the filename already exists they are given the option of continuing (overwriting) or cancelling. If the filename doesn't exist or does but the user has elected to continue the m_filename member is set and fileSave() is called.

fileSave()

```
void MainForm::fileSave()
{
    if ( m_filename.isEmpty() ) {
        fileSaveAs();
        return;
    }

    QFile file( m_filename );
    if ( file.open( IO_WriteOnly ) ) {
        QTextStream stream( &file );
        if ( ! m_comments.isEmpty() )
            stream << m_comments.join( "\n" ) << "\n";
        QMap<QString,QColor>::ConstIterator it;
        for ( it = m_colors.constBegin(); it != m_colors.constEnd(); ++it ) {
            QColor color = it.data();
            stream << QString( "%1 %2 %3\t\t%4" ).
                      arg( color.red(), 3 ).
                      arg( color.green(), 3 ).
                      arg( color.blue(), 3 ).
                      arg( it.key() ) << "\n";
        }
        file.close();
        setCaption( QString( "Color Tool -- %1" ).arg( m_filename ) );
        statusBar()->message( QString( "Saved %1 colors to '%2'" ).
                              arg( m_colors.count() ).
                              arg( m_filename ), 3000 );
        m_changed = FALSE;
    }
    else
        statusBar()->message( QString( "Failed to save '%1'" ).
                              arg( m_filename ), 3000 );

}
```

If there is no current filename we call fileSaveAs(); that function will call this one if the user provides a filename.

We write out any comment lines first. This means that a file that we load and then save may not be the same (e.g. if the original had comments scattered throughout, since our saved version will have all the comments at the beginning). We then iterate over every color in the m_colors map, writing them out in the rgb.txt file format.

fileExit()

```
void MainForm::fileExit()
{
    if ( okToClear() ) {
        QApplication::exit( 0 );
    }
```

```
}
```

This is the second revision of this function. Now we only exit if the user has had the opportunity to save any unsaved changes. (We'll make a third and final version of this function later, when we deal with saving user settings.)

Try making and running the program. If you have rgb.txt on your system try loading it and saving it under a new name for testing purposes. If you don't have this file, save the standard colors and use those. In the next section we'll cover adding and deleting colors so that you can create your own color files. (If it doesn't build see the Troubleshooting section.)

Editing the Code: The Edit Options

Adding a new color, finding a color and handling user options all require custom dialogs, so we'll defer them until chapter three when we deal with dialogs.

editCut()

```
void MainForm::editCut()
{
    QString name;
    QWidget *visible = colorWidgetStack->visibleWidget();
    statusBar()->message( QString( "Deleting '%1'" ).arg( name ) );

    if ( visible == tablePage && colorTable->numRows() ) {
        int row = colorTable->currentRow();
        name = colorTable->text( row, 0 );
        colorTable->removeRow( colorTable->currentRow() );
        if ( row < colorTable->numRows() )
            colorTable->setCurrentCell( row, 0 );
        else if ( colorTable->numRows() )
            colorTable->setCurrentCell( colorTable->numRows() - 1, 0 );
        m_icons_dirty = TRUE;
    }
    else if ( visible == iconsPage && colorIconView->currentItem() ) {
        QIconViewItem *item = colorIconView->currentItem();
        name = item->text();
        if ( colorIconView->count() == 1 )
            colorIconView->clear();
        else {
            QIconViewItem *current = item->nextItem();
            if ( ! current )
                current = item->prevItem();
            delete item;
            if ( current )
                colorIconView->setCurrentItem( current );
            colorIconView->arrangeItemsInGrid();
        }
        m_table_dirty = TRUE;
    }

    if ( ! name.isNull() ) {
        m_colors.remove( name );
        m_changed = TRUE;
        statusBar()->message( QString( "Deleted '%1'" ).arg( name ), 5000 );
    }
    else
        statusBar()->message( QString( "Failed to delete '%1'" ).arg( name ), 5000 );
}
```

If the user is viewing the table view we delete the current row. We set the new current cell to be the one following the deleted row, or if the one we deleted was last, its predecessor. We mark the *other* view (the icon view) as dirty, to make sure that it is updated if the user switches views. Similarly, if the user is viewing the icon view, we make the next (or previous if there is no next) item current and delete the one they were on. We then mark the table view as dirty. If we deleted a color (i.e. there was a current color in one of the views), we remove it from the m_colors map and mark the data as changed.

editCopy()

```
void MainForm::editCopy()
{
    QString text;
    QWidget *visible = colorWidgetStack->visibleWidget();

    if ( visible == tablePage && colorTable->numRows() ) {
        int row = colorTable->currentRow();
        text = colorTable->text( row, 0 );
    }
    else if ( visible == iconsPage && colorIconView->currentItem() ) {
        QIconViewItem *item = colorIconView->currentItem();
        text = item->text();
    }
    if ( ! text.isNull() ) {
        QColor color = m_colors[text];
        switch ( m_clip_as ) {
            case CLIP_AS_HEX: text = color.name(); break;
            case CLIP_AS_NAME: break;
            case CLIP_AS_RGB:
                text = QString( "%1,%2,%3" ).
                    arg( color.red() ).
                    arg( color.green() ).
                    arg( color.blue() );
                break;
        }
        clipboard->setText( text );
        statusBar()->message( "Copied '" + text + "' to the clipboard" );
    }
}
```

In this function we retrieve the name of the color from the current table row (or current icon, depending on the view). We then set a QString to the text we want to copy into the clipboard and copy it.