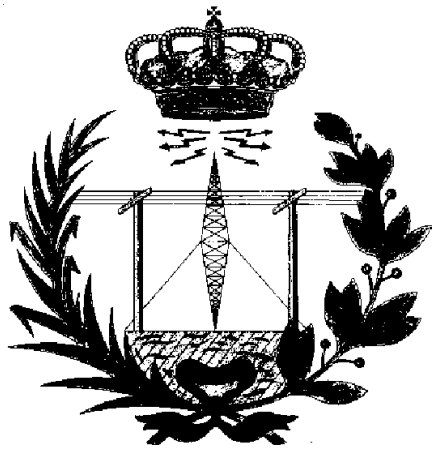
INGENIERÍA DE TELECOMUNICACIÓN

ESCUELA SUPERIOR DE INGENIEROS

UNIVERSIDAD DE SEVILLA

PROYECTO FIN DE CARRERA



Implementación de un Monitor y Analizador Gráfico de Red en el Entorno Gnome

por Juan Toledo Cota

Proyecto dirigido por Jon Tombs

Sevilla, Julio de 2001

a Mónica, por sobrellevar una
relación de muchos a muchos.
(Ella, los ordenadores y yo)

Tabla de contenidos

Lista de figuras

Agradecimientos

Jorge Chávez. Las horas interminales en su despacho no tienen precio. No hay mejor manera de aprender sobre ordenadores y tecnología que divertirse en su tecno parque temático.

Jon Tombs, mi gurú local de Unix. ¿Cuánta gente puede presumir de haber aprendido Linux de alguien que figura en el archivo AUTHORS del kernel?

Daniel López Ridruejo, por ser la fuente de inspiración para iniciar este proyecto.

A mis padres, por enseñarme a aprender.

A todos mis amigos, por no abandonarme cuando no modulo.

La Free Software Foundation, por el mágnifico conjunto de código que donan a la humanidad (y por supuesto también por el ordenador que me subvencionaron para seguir desarrollando EtherApe)

Miguel de Icaza, por iniciar el proyecto Gnome y dar un toque de español a un mundo tan dominado por el inglés.

A tantos que han contribuido a hacer del software libre lo que es: Linus Torvalds, Richard Stallman, Peter Mattis, Spencer Kimball, Josh MacDonald, Federico Mena Quintero, Damon Chaplin, Gerald Combs, Laurent Deniel, ESR, etc.

A todos los usuarios de EtherApe, y particularmente a los que han dedicado tantas horas para ayudar a depurar el programa: David Pollard, Jim Howard, etc.

```
#include "README.thanks"
```

Y a Mónica no sólo se lo dedico. También se lo agradezco.

Introducción

En los últimos años hemos vivido el desarrollo explosivo de la informática. Pero más importante si cabe que la universalización del acceso a los ordenadores ha sido el desarrollo de las redes de ordenadores, y en particular la fulminante penetración que Internet ha protagonizado a todos los niveles de la sociedad.

En el aspecto tecnológico, el hecho de que el acceso a todo tipo de datos remotos sea una realidad ubicua ha dado lugar al surgimiento casi diario de nuevas tecnologías que hacen uso de las nuevas posibilidades. Esta aceleración del progreso tecnológico ha traído consigo su propio vocablo: se habla del "Internet Time", dando a entender que las cosas cambian mucho más rápido de lo que la industria estaba acostumbrada. Las empresas han tenido que aprender a reinventarse a sí mismas casi cada año, a riesgo de quedarse en la cuneta desbancadas por nuevos competidores surgidos de la nada.

Afortunadamente este progreso hacia un medio de comunicación global extremadamente barato no sólo beneficia a las empresas. Los ciudadanos de a pie pueden ahora ponerse en contacto con gentes del otro lado del planeta de manera casi inmediata. Esto posibilita la creación de "comunidades virtuales", grupos de gente con aficiones comunes por raras que sean que no están limitados por su esparcimiento geográfico.

Las nuevas tecnologías que van surgiendo traen también otra consecuencia. En muchos casos los sistemas que se van poniendo en marcha crecen en complejidad y es difícil mantener una idea clara del conjunto que se está construyendo. Se echa en falta personal cualificado que pueda mantener la maquinaria en marcha, y muchas de las herramientas que facilitarían el trabajo están aún por diseñarse.

Aquí es donde las consecuencias sociales de Internet entran para echar una mano. Algunas de las comunidades virtuales que se han creado integran a profesionales de la tecnología que disfrutan de su trabajo. Pueden crear de manera independiente a las grandes empresas, y altruistamente generan soluciones a diferentes problemas que afectan a la sociedad.

Esa es la semilla de la que parte EtherApe. A partir de un problema, la necesidad de una herramienta de diagnóstico rápido de red, la sinergia del movimiento del software libre es capaz de generar una solución en un corto espacio de tiempo que puede competir de tú a tú con otros productos comerciales.

El trabajo no surge solo: sigue haciendo falta el esfuerzo de individuos dedicados. La diferencia está en la repercusión que ese esfuerzo individual tiene en el conjunto de la sociedad.

Monitorización de redes

> *Para cualquier actividad que se vaya a realizar son necesarias herramientas que ayuden a depurar el proceso, o que ayuden al mantenimiento una vez que esté puesto en marcha. En los trabajos que involucran redes de ordenadores, estas herramientas son los monitores de red, también conocidos como sniffers.*

Hoy en día las redes de ordenadores están muy extendidas. Desde el PC familiar conectado por módem a un proveedor gratuito de servicios de Internet, hasta el ejército de ordenadores de grandes empresas bancarias, todos son ejemplos de sistemas en red.

Potencialmente cada uno de los ordenadores de una red puede establecer una comunicación con otro. El gran número de ordenadores que intervienen es uno de los niveles de complejidad del problema, pero sólo el primero. Cada nodo puede mantener más de una conversación a la vez, y cada una de estas conversaciones puede tener un objetivo distinto.

El objetivo de estas conversaciones es variado. Algunas tendrán como finalidad el ayudar a la consecución final del producto o servicio por el que se montó la red, por ejemplo, la consulta de la página web de una agencia de viajes para la compra de un billete de avión. Otras, sin embargo, tienen lugar únicamente con el objetivo de mantener el sistema en marcha, como el intercambio constante de información que los encaminadores de las distintas redes tienen entre sí.

En un sistema tan complejo como puede ser una red informática muchas cosas pueden ir mal. Un servidor web mal configurado podría estar redirigiendo a un navegador a una página no deseada, un nodo defectuoso de la red puede generar tráfico hasta anular la capacidad de transporte de la red, o un encaminador mal programado puede estar "dejando caer" información, o enviándola al lugar erróneo.

Además de todo lo que pueda ocurrir fortuitamente, cualquier recurso que se pone a disposición de unos usuarios puede acabar siendo abusado. Un empleado que monopoliza el ancho de banda de una empresa, tanto si es para usos lícitos como si lo es por motivos lúdicos, puede reducir de manera importante la funcionalidad de la red. Además están los peligros externos. Individuos que por diversión o por motivos económicos intentan introducirse en los sistemas de una empresa, o una universidad, saltándose las posibles medidas de seguridad que se hayan establecido.

Es por todo esto que el gestor de la red necesita de una herramienta que le ayude a analizar lo que está ocurriendo. De por sí un cable no presenta un aspecto diferente si está cargado hasta el límite de capacidad o si no está siendo utilizado. Inferir que hay un problema usando las aplicaciones diarias de la red puede no ser obvio a primera vista. Y desde luego un intruso que no quiera ser detectado no lo será a menos que hagamos una búsqueda activa.

Así pues, los objetivos que persiguen los monitores de red pueden resumirse en:

- Análisis de la eficiencia del sistema
- Diagnosis de posibles problemas
- Identificación de amenazas de seguridad

Arquitectura de EtherApe

EtherApe es la solución al problema del diagnóstico rápido de una red. El principal objetivo de la arquitectura adoptada es el de obtener resultados lo más rápidamente posible. La tecnología en que se basa (Gnome) y el método de trabajo (software libre) han sido de gran ayuda para la consecución del objetivo marcado. Pero no basta con eso. Se ha tomado un esfuerzo considerable en separar las funciones del programa en módulos casi independientes. De esta manera también se ayuda a mejorar la mantenibilidad y depurabilidad del código.

1 Objetivos de diseño

La idea que da lugar a EtherApe es poder reproducir los resultados de Etherman en las plataformas más accesibles hoy día. Esta afirmación genérica se traduce en detalle en:

- Representación gráfica de la red

  Se requiere una representación intuitiva de lo que sucede en la red. Los nodos se identifican con círculos y las conversaciones con líneas que unen aquellos. Se introduce información adicional en el tamaño y el color de los elementos

- Representación en tiempo real

  EtherApe pretende ser un primer recurso en la diagnosis de problemas. La información se presenta en tiempo real y la persona responsable puede decidir qué pasos ulteriores tomar tras un vistazo rápido.

- Portabilidad

  Etherman estaba limitado a las plataformas a las que los desarrolladores originales tenían acceso.

- Perdurabilidad del código

  En contraposición a Etherman, EtherApe debe poder estar disponible por largo tiempo, independientemente de que cambien las plataformas subyacentes o que el desarrollador principal abandone su puesto. La licencia GPL asegura este punto.

Estos objetivos condicionan la elección entre las posibles alternativas que estén disponibles para la implementación. Después de presentar la solución adoptada se discutirán las posibles opciones y la razón que motiva tal decisión.

2 Esquema general

EtherApe es un programa diseñado para correr bajo plataformas Unix genéricas. El código fuente en C es portable a diversas implementaciones que cumplan con la norma POSIX.

EtherApe ha sido compilado y ejecutado con éxito al menos en las siguientes combinaciones de sistemas operativos y procesadores:

- Linux

  Debian, RedHat, Mandrake, Slackware, Suse, etc.

  Probado con procesadores x86, Alpha, Motorola 680x0 y PowerPC

- FreeBSD
- NetBSD
- Solaris

  Al menos con las versiones 7 y 8

Para conseguir su objetivo, EtherApe saca partido de un conjunto de bibliotecas de libre distribución, que son las que verdaderamente definen el límite de la portabilidad del programa.

En el siguiente esquema está representada la arquitectura de EtherApe, con el conjunto de bloques funcionales que la componen.

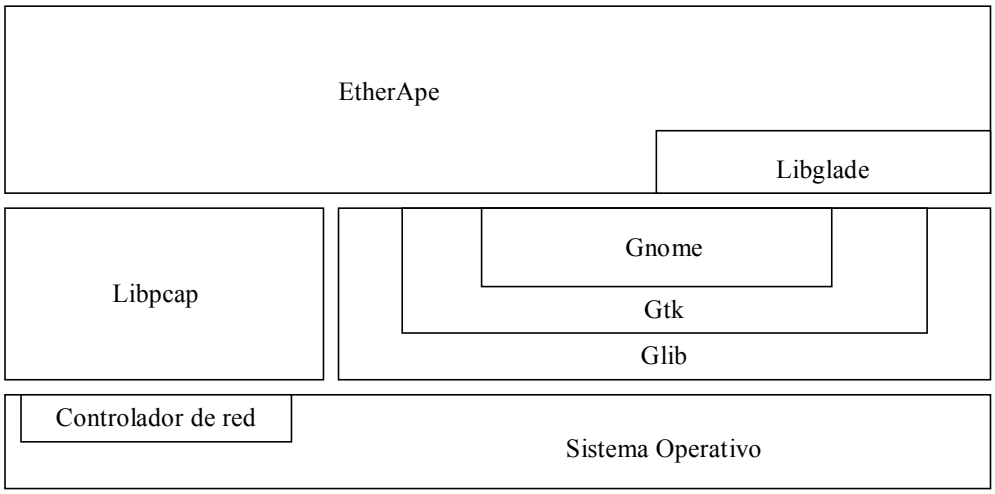


Figura 1 – Diagrama de bloques de EtherApe

Tal y como puede apreciarse en el esquema, el programa no interactúa directamente con el sistema operativo en ningún caso, sino que deja que sean las bibliotecas las que cumplan esa función.

También salta a la vista el hecho de que la interacción con el sistema operativo (y por ende, con el hardware) se realiza por dos caminos distintos que están bien diferenciados. Por un lado el sistema que se encargará de leer el tráfico de la red, y por otro el encargado de presentar los datos y el resto de la interfaz de usuario.

Nótese que en cualquier caso este es un esquema simplificado, y que cada uno de los elementos trae consigo sus propias dependencias. Ausencias notables son la biblioteca estándar de C, y las bibliotecas del sistema X-Windows.

3 Dependencias externas

Una de las máximas siempre presentes en el desarrollo de software libre (del software en general, pero como veremos más adelante es un aspecto fundamental del software libre) es el esfuerzo por no reinventar la rueda.

Este concepto alude al hecho de que demasiado a menudo un desarrollador tenderá a rechazar código no realizado por él mismo bien por desconfianza, bien por pereza a la hora de estudiar APIs (application program interface, interfaz de programación de aplicaciones) ajenos.

Sin embargo utilizar adecuadamente los recursos disponibles trae consigo un conjunto de beneficios:

- Disminución del tiempo de desarrollo

  Si bien será necesario adaptar el programa para asegurar que se ajusta a la interfaz de la biblioteca que vayamos a usar, ese tiempo siempre es menor que el que se necesita para programar desde el principio esa funcionalidad

- Aumento de la generalidad

  La biblioteca que vayamos a utilizar es un proyecto en sí mismo y sus creadores probablemente se habrán encargado de hacerlo funcionar en más entornos de los que nosotros tengamos inicialmente planeados

- Disminución del tiempo de depuración

  Por el mismo motivo que el punto anterior, los encargados de mantener esa biblioteca habrán tenido oportunidad de probar más casos patológicos, aumentando la fiabilidad del código.

  Por otro lado es cierto que no siempre es apropiado usar bibliotecas. Puede ocurrir que no haya bibliotecas existentes que se ajusten exactamente a nuestras necesidades. Aunque este no es el caso de EtherApe, una ventaja de usar bibliotecas que sean software libre cuando ocurre esto es que podemos partir del código existente, y añadir a la biblioteca sólo lo que falta hasta servir nuestras necesidades.

  Por todos estos motivos a lo largo del desarrollo de EtherApe se ha hecho un esfuerzo consciente para investigar qué código estaba ya disponible que pudiera ayudar en la implementación, y de usarlo en lo posible.

3.1 Libpcap

Libpcap es la biblioteca que nos permite obtener una copia de los paquetes que circulan por la red. Libpcap empezó siendo parte del monitor de red Tcpdump, pero con el paso del tiempo sus creadores la segregaron y la distribuyeron como una biblioteca independiente para que otros proyectos pudieran sacarle partido fácilmente.

Algunas de las características que hacen Libpcap enormemente interesante son:

- Independiente de plataforma

  A lo largo de los años en los que Tcpdump se ha ido desarrollando (la versión inicial de este programa data de 1991), el número de plataformas que soportaba ha ido creciendo sin cesar hasta convertirse prácticamente en una herramienta universal.

Libpcap ha heredado esta característica, y permite a los programas que enlazan con esta biblioteca usar un único API para extraer paquetes de la red desde casi cualquier sistema operativo: Ultrix, SunOS, VMS, Linux, HP-UX, FreeBSD, NetBSD, etc. Recientemente incluso el entorno Win32 está soportado, ampliando aún más si cabe sus dominios.

- Independiente de medio físico

  Libpcap no sólo es capaz de entender las especificidades de muchos sistemas operativos. También es capaz de abstraer las diferencias entre los distintos medios de transporte y proporciona la trama que se haya recibido sea cual sea el caso. Además, usando la función adecuada, informa a la aplicación que está por encima de cuál es el medio físico que se está usando, de modo que pueda prever cómo serán que formato tendrán las tramas que capture

- Lectura y escritura de archivos de captura

  Además de recoger paquetes "en vivo" directamente de un dispositivo de red, Libpcap es capaz de recuperar un volcado de tráfico hecho en un archivo, probablemente con otra herramienta que haya use Libpcap. En el caso de EtherApe, esto nos permite depurar problemas de usuarios utilizando como datos de ejemplo exactamente el mismo tráfico que ven ellos en su red.

  Otra ventaja adicional para la depuración es que cuando se lee de un archivo de datos pasamos del entorno aleatorio de una red en uso real a una entrada de datos que es siempre la misma. Si el programa comete un error al intentar procesarla podemos repetir el proceso cuantas veces sea necesario hasta identificar la causa del problema.

- Filtrado de tráfico

  Cuando una aplicación pide a la biblioteca Libpcap que inicie una captura, puede también pedir que sólo le muestre los paquetes que cumplan unos determinados criterios. Esta funcionalidad es extremadamente útil para aislar problemas en redes que están muy cargadas, donde pueden estar manteniéndose docenas de "conversaciones" diferentes en un momento dado.

  La sintaxis que definen los filtros es muy extensa, y entre otras cosas es posible seleccionar tráfico usando criterios de nodo de origen o destino, protocolo que se esté usando, puerto al que va destinado un paquete, etc.

Por todo lo anterior el uso de Libpcap es una decisión obvia a la hora de implementar cualquier monitor de red. De hecho, el número de paquetes basados en esta biblioteca es bastante alto. A modo de ejemplo podemos citar Ethereal, ya discutido más arriba; Nessus un programa auditor de seguridad; divine, que escucha los mensajes de ARP para sugerir configuraciones de red para ordenadores portátiles; Sniffit, otro analizador de red, pero con funciones específicas para volcar el contenido textual de las conversaciones (muy práctico por tanto para los piratas informáticos), etc.

El único inconveniente que se le podría achacar a Libpcap es que el hecho de que sea capaz de filtrar el tráfico usando tal diversidad de criterios implica que la misma biblioteca está de hecho analizando la pila de protocolos de cada paquete que recibe. En muchos casos eso se traduce en que se duplica el trabajo que hace el programa: la pila de protocolos se analiza dos veces, una en la biblioteca, y otra en la aplicación que soporta.

Sin embargo durante el desarrollo y el uso de EtherApe no se ha apreciado que esto suponga ningún inconveniente grave.

3.2 Glib

Glib es una biblioteca de portabilidad y de utilidades para sistemas Unix y Windows. Al igual que ocurre con Libpcap, Glib se escindió como biblioteca independiente a partir de Gtk+. Podríamos decir que Glib es un intento de estandarizar una extensión de la biblioteca estándar de C. Veámoslo con más detalle.

3.2.1 Portabilidad

Gtk+ se creó para ser un sustituto adecuado de la biblioteca de componentes gráficos Motif. Una de los objetivos a alcanzar era que el sistema fuera portable a muchas arquitecturas, salvando las distancias que separan a cada una de las implementaciones de C en cada plataforma. Hoy día es Glib la biblioteca que consigue este objetivo, y no sólo da soporte a Gtk+, sino a una gran variedad de otras bibliotecas y aplicaciones.

Veamos una lista de características que hacen que las aplicaciones que usan Glib sean más portables.

- Definición de tipos portables.

  Un problema del estándar C es que los tipos `int`, o `short int`, por ejemplo, no tienen exactamente la misma dimensión en todas las

plataformas. Depende de cómo esté definido para la arquitectura subyacente.

Por ese motivo, cuando se pretende hacer un programa portable el programador a de crear código condicional según la plataforma en la que se vaya a compilar, de manera que pueda tener confianza en el tamaño de los tipos que está utilizando.

Si se usa Glib, se deja ese trabajo a la biblioteca. Glib define una serie de tipos inambiguos, tales como gint8, guint8, gint16, guint16, gint32, guint32, gint64, guint64. Por otro lado también se añaden tipos que no están en el estándar a pesar de ser muy comunes como gboolean, gsize, gssize; y otros para simplificar el lenguaje como gpointer, gconstpointer, guchar, guint, gushort, gulong.

EtherApe saca partido de estos tipos en muchas situaciones. Por ejemplo, a la hora de definir variables que han de contener la una dirección IP (cuatro octetos) se usa el tipo `guint32`; o cuando se necesita un puntero a una zona de memoria que alberga una trama (es decir, un conjunto arbitrario de octetos), se utiliza `guint8 *`.

- Carga de código en tiempo de ejecución.

  Tanto Windows como muchas versiones de Unix soportan la incorporación de objetos compilados en tiempo de ejecución (las conocidas DLL, o de manera genérica, objetos compilados en Unix).

  Sin embargo el procedimiento para hacer uso de esa funcionalidad es dependiente de cada sistema operativo. Glib alivia el problema introduciendo un API portable, tanto para que el programa pueda averiguar si esa función está presente en la plataforma, como para ponerla en marcha.

- Canales de entrada y salida

  En C básico se habla de entrada y salida estándar, del error estándar y de descriptores genéricos de archivo. Sin embargo los sistemas operativos tienen características que van más allá de estos, como los *pipes* y los *sockets*, y trabajar con ellos requiere añadir complejidad al programa, además de utilizar mecanismos que no son portables.

  En particular, hacer lecturas o escrituras asíncronas implica diseñar el ciclo principal del programa alrededor de ellas, y es incómodo tener que estar pendiente en cada momento cómo se comportará tal o cual sistema operativo.

Mediante Glib se simplifica el proceso. Cómo veremos más adelante Glib aporta una implementación estandarizada del ciclo principal de un programa, y de esta manera puede incorporar también de manera natural en procesado de canales de entrada y salida, en sus múltiples encarnaciones, y tanto en su modo síncrono como asíncrono.

Esta característica es fundamental para EtherApe, que funciona a partir de dos flujos asíncronos independientes: la aparición de tramas de datos en la interfaz de red que se está leyendo y la interacción de usuario con la interfaz del programa.

3.2.2 Utilidades

Casi a la vez que se desarrollan los lenguajes de programación van apareciendo técnicas de programación que se comprueba son útiles. A pesar de que muchas de estas técnicas no son realmente parte del lenguaje en cuestión son tan básicas y tan necesarias que en muchas ocasiones su enseñanza se hace de manera conjunta.

Estructuras de datos como las listas enlazadas, los árboles binarios balanceados, o las tablas de hash son básicas para cualquier programa que deba almacenar información, y sin embargo por lo común cada programador debe hacer un reimplementación para cada programa que vaya a realizar, si bien lo más común era que al cabo del tiempo se creara su propia biblioteca de utilidades.

Por otro lado procedimientos como un bucle principal con soporte para eventos asíncronos, funciones que ayuden a mostrar mensajes de depuración o de registro, o que ayuden a minimizar los problemas de asignación y corrupción de memoria en C son también extremadamente comunes, si bien en muchos casos puesto que su uso ya no es tan perentorio acaba por ocurrir que los programadores lo dan de lado y da como resultados programas menos robustos y flexibles.

Glib pretende dar una solución a estos dos problemas. Estandarizando una solución el programador puede olvidarse de los detalles de estas tareas "mundanas" y pensar solamente en lo que es específico a su programa. Un conjunto de especialistas mantiene Glib, y se encarga de asegurarse de que la implementación de estos algoritmos genéricos es lo más eficiente y robusta posible.

Algunos de las utilidades que Glib aporta y de las que EtherApe hace uso extensivo son:

- El ciclo principal de eventos

  La mayoría de los programas que se hacen hoy en día no se corresponden con el modelo de antaño. En lugar de ejecutar un programa que realizará un único cálculo o función, hoy los programas se acercan más a las arquitecturas cliente/servidor.

  Los servidores de red se mantienen en segundo plano hasta que un cliente hace una llamada, y sólo entonces se despierta el proceso para realizar algún trabajo. Pero también ocurre lo mismo en las interfaces de usuario. Un programa como un procesador de texto está esencialmente parado hasta que un usuario pulsa una tecla o activa un mando con el ratón.

  Si bien las bibliotecas de componentes gráficos suelen proveer de algún mecanismo para permitir que el programa que deba soportar se comporte de esta manera, Glib ofrece la oportunidad de sustentar este esquema en cualquier tipo de programa, no sólo los que está orientados a una IGU. Así, pueden usarlo por ejemplo también servidores de red o bien bibliotecas de mayor nivel que se encarguen del sistema gráfico.

  Como ya se comentó antes EtherApe hace uso del ciclo principal de eventos para gestionar tanto la llegada de tramas de red como del tratamiento de la interfaz de usuario.

- Estructuras complejas de datos

  Glib proporciona un amplio abanico de estructuras de datos complejas, que se ajustan a cualquier uso. Tablas, listas simple y doblemente enlazadas, árboles balanceados binarios, tablas de hash, e incluso estructuras de uso mucho menos común como árboles n-arios, quarks (asociación bidireccional entre una cadena y un número entero) y tablas indizadas (tablas que pueden ser indizadas en un número arbitrario de columnas)

  Puesto que la biblioteca no puede saber qué uso exactamente se le va a dar a cada estructura, cada elemento almacena un puntero, en lugar de una estructura específica. EtherApe, por ejemplo, debe almacenar el conjunto de nodos que ha ido escuchando en la red. Para ello se crea una estructura que guarda información sobre un nodo, y se almacena un puntero a la clave que identifica a ese nodo en un árbol balanceado binario junto con el puntero a la estructura creada.

  Glib también proporciona funciones específicas para gestionar estas estructuras complejas de datos, ahorrando al programador de tener que estar pendiente de reservas de memoria, por ejemplo. Además puesto que Glib se

encarga de una buena parte de la gestión memoria, es capaz de optimizar su uso para minimizar el número real de peticiones de asignación que se hacen al sistema operativo, y de este modo acelerar la ejecución del programa.

- Mensajes de depuración y registro

  Glib incorpora un sistema flexible para el tratamiento de mensaje de depuración y registro. Usando funciones como `g_debug`, `g_info`, `g_warning`, o `g_critical` el programador separa la información que el programa debe registrar de la implementación del registro o la presentación de esa información.

  Por defecto la salida se hace por el estándar error dependiendo por ejemplo del valor de la variable de entorno DEBUG (tal y como hace EtherApe), pero también es posible derivar su procesado a otras funciones, de manera que se pueda hacer que la salida se presente en una ventana de la interfaz gráfica de usuario, o enviarlo al sistema de registro de mensajes del sistema operativo (syslogd, en los sistemas GNU/Linux)

El número de utilidades que incorpora Glib es aún mayor: funciones para la gestión de hilos, la gestión de cachés de memoria, el tratamiento robusto de cadenas o la medida de tiempos son algunas de ellas. Aquí sólo se han presentado algunos de los ejemplos de los que EtherApe se beneficia.

3.3 Gtk+

Gtk+ son las siglas de Gimp Toolkit. En 1996 dos estudiantes de Berkeley publicaron la primera versión de un programa de retoque fotográfico como software libre, llamado Gimp. En su primera encarnación Gimp usaba la biblioteca de componentes gráficos propietaria Motif, el estándar de la época, y pronto estuvo clara la necesidad de desarrollar una biblioteca libre que estuviera a la par de las necesidades técnicas.

Así surgió Gtk+, una biblioteca de componentes gráficos que desde entonces ha evolucionado técnicamente y se ha hecho muy popular, superando su objetivo inicial de dar soporte únicamente a Gimp.

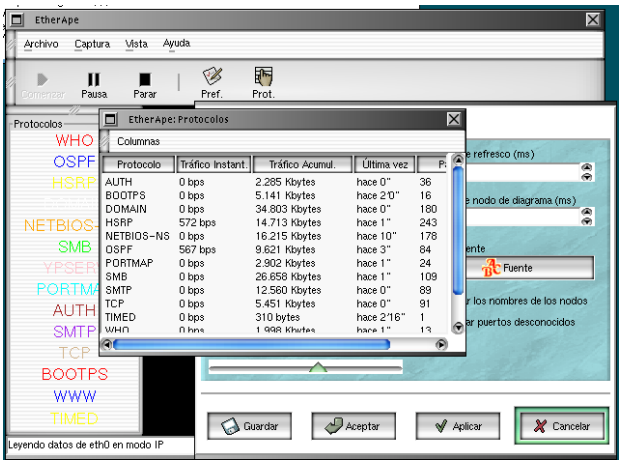
A pesar de que Gtk+ es una biblioteca escrita en C y preparada para ser usada por programas en C, Gtk+ incorpora el concepto de objetos mediante un ingenioso uso de las estructuras. Así pues los componentes se derivan unos de otros, mantienen las propiedades de sus padres y pueden ser tratados como cualquiera de sus ancestros.

De hecho la implementación interfaz de usuario como un conjunto de objetos es el paradigma que prevalece hoy en día y por tanto Gtk+ ha debido adoptar una estructura que lo soportara.

Gtk+ incorpora una larguísima lista de componentes gráficos de entre los cuales el programador de la aplicación puede elegir: botones, entradas de texto, ventanas de diálogo, árboles jerárquicos, etiquetas, menús, barras de progreso, etcétera. En este sentido Gtk+ es un gran paso adelante con respecto al estado anterior de la situación, en la que los programas que quisieran tener una presentación gráfica debían elegir entre pagar una licencia de Motif para tener un sistema moderno y eficaz o programar directamente usando las bibliotecas de bajo nivel del sistema X-Windows.

Gtk+ posee una característica que lo hace muy atractivo para los usuarios modernos. Una vez diseñado una interfaz de usuario, el aspecto final en pantalla no está totalmente delimitado, sino que el usuario tiene libertad para escoger la apariencia de los controles. Esto que parece banal hoy en día tiene una gran importancia entre el público a la hora de decidir qué programa van a usar. El programa no sólo ha de ser funcional. También ha de ser bonito.

Si bien no ha sido este el motivo fundamental para la elección como la biblioteca de componentes gráficos de EtherApe, sí es un aliciente añadido bastante interesante.



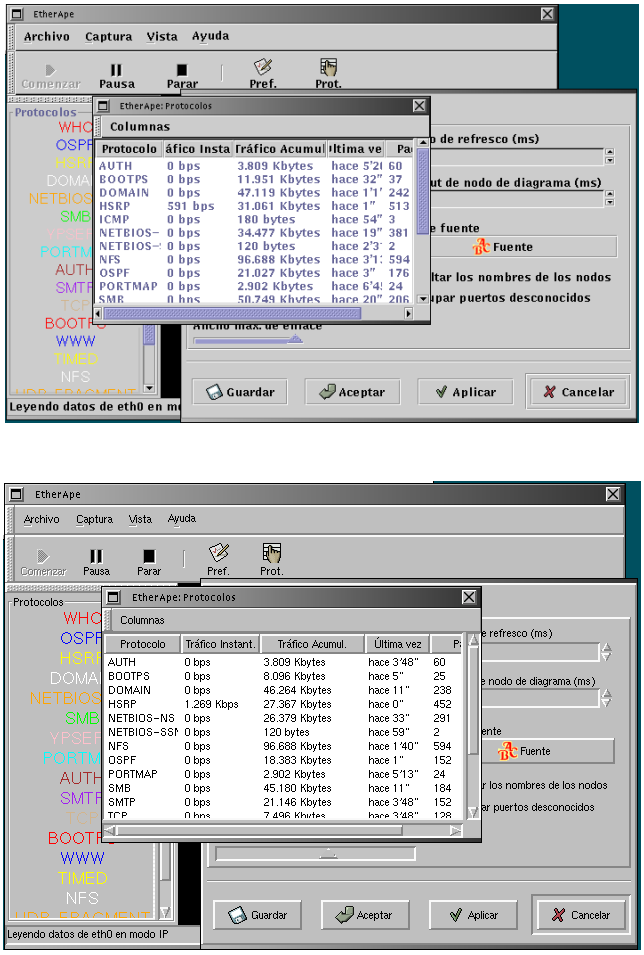


Figura 2 – Estilos de Gtk+ aplicados a EtherApe

Gtk+ tiene una estructura que lo hace fácilmente portable. Si bien en su encarnación original Gtk+ tenía como objetivo servir de puente entre la aplicación y las bibliotecas del sistema de ventanas de X, en realidad en grueso de la biblioteca no depende de X en absoluto, sino que se basa en otra biblioteca propia que se encarga de acceder a las primitivas gráficas: la biblioteca Gdk.

Puesto que los puntos de contacto con el sistema X-Windows están tan bien definidos, en los últimos tiempo ha sido posible hacer una versión de Gdk (y por tanto de Gtk+) que funciona de manera nativa en los entornos Win32. A pesar de que la versión de Gtk+ para Windows todavía no es estable, el hecho de que no sea técnicamente imposible implica que a no muy largo plazo las aplicaciones originalmente programadas para correr en entornos Unix usando Gtk+ podrían ser también aplicaciones de Windows con muy pocos cambios.

Actualmente Gtk+ es junto con QT una de las dos bibliotecas de componentes gráficos distribuidas como software libre con más aceptación. La diferencia

técnica fundamental entre las dos es que mientras que Gtk+ es una biblioteca en C, QT es una biblioteca de objetos de C++.

3.4 Gnome

3.4.1 Nota histórica

Además de la diferencia entre los lenguajes de Gtk+ y QT, hay otra distinción histórica que mantiene a los usuarios de cada una de las bibliotecas a bastante distancia.

En 1998, más o menos a la vez que la versión 1.0 de Gtk+ hacía su aparición, un conjunto de programadores en Alemania inicia un intento de desarrollar un entorno integrado de usuario (un equivalente en software libre a CDE para plataformas Unix, o la interfaz de usuario de Windows) basado en una reciente biblioteca de componentes que en aquel momento era técnicamente más avanzada que Gtk+, pero que adolecía de "problemas de licencia". Era (y es) el entorno KDE.

Si bien QT era una biblioteca gratuita, no era software libre en sí mismo, y por tanto muchas figuras del software libre argüían que KDE hipotecaba su futuro a los designios de la compañía que desarrollaba la biblioteca QT.

Surgió entonces un movimiento alternativo para desarrollar otro entorno integrado de usuario, pero éste había de estar basado enteramente en software libre, y en particular se eligió Gtk+ como la biblioteca de componentes gráficos que habría de sustentarlos. Este segundo entorno es el conocido como Gnome.

Con el paso de los años la disputa original acabó por carecer de sentido cuando Trolltech, la compañía creadora de QT distribuyó su biblioteca bajo la licencia GPL. Sin embargo para entonces tanto KDE como Gnome habían alcanzado ya una masa crítica de desarrolladores y hoy en día siguen evolucionando como dos alternativas independientes.

Figura 3 – El escritorio Gnome

3.4.2 Definición de Gnome

A primera vista es difícil aclarar qué es exactamente Gnome, y esto es así porque Gnome es en realidad un conjunto de cosas agrupadas bajo un mismo nombre.

- Un entorno integrado de usuario

  También conocido como el escritorio Gnome. Son un conjunto de aplicaciones que hacen que el contacto inicial de un usuario no técnico con el ordenador sea una experiencia relativamente amigable.

  Aquí se incluyen los menús de acceso a los programas, las barras de tareas, los pequeños programas que los usuarios han aprendido a esperar que estén siempre disponibles, como una calculadora, etc.

  El objetivo es lograr un escritorio atractivo y que las aplicaciones tengan un funcionamiento similar entre todas ellas para hacer más intuitivo el trabajo del usuario.

- Aplicaciones de productividad

  El núcleo de desarrolladores de Gnome no sólo pretende programar las pequeñas aplicaciones (por comparación, digamos las aplicaciones que vendrían por defecto al instalar Windows 98), sino que también están las

aplicaciones más complejas que se necesitan hoy en día en cualquier entorno productivo.

Por ejemplo, Gnumeric es la respuesta del entorno Gnome al programa de hoja de cálculo Excel, pero también hay programas para hacer procesamiento de textos, presentaciones, tratamiento de imágenes, etc.

- Una plataforma de desarrollo

  Para alcanzar los objetivos que se pretenden, Gnome se ayuda de una serie de bibliotecas que añaden funcionalidad a los programas. Una de estas bibliotecas se construye sobre Gtk+ para crear componentes gráficos más complejos (por ejemplo un selector de archivos que mantiene un historial), pero muchas más son independientes de la interfaz gráfica.

  Hay, por ejemplo, funciones para procesar los argumentos de la línea de comandos, para gestionar los documentos de ayuda en línea, soporte para una arquitectura de componentes sobre CORBA, y una capa de abstracción de sistemas de archivos (útil para leer imágenes de una cámara fotográfica como si fuera un sistema local, por ejemplo).

Si bien EtherApe no pertenece de manera formal al proyecto Gnome (una distinción que puede definirse como meramente administrativa), si que intenta cumplir todas las guías de diseño para programas de ese entorno, y desde luego hace uso extensivo de las facilidades que proporciona la plataforma de desarrollo.

3.4.3 Características de interés

En muchos casos no es indispensable hacer una aplicación dependiente de Gnome para que pueda cumplir su objetivo fundamental. De hecho cualquier aplicación que use las bibliotecas de Gnome provoca una cascada de dependencias que en muchos casos puede complicar su uso a los usuarios finales. Por este motivo no es raro encontrar aplicaciones que tengan dos versiones diferentes. Una más ligera que sólo dependa de Gtk+, y otra más amigable para el usuario pero también con un bagaje de dependencias mucho más amplio que sí usan las bibliotecas que proporciona Gnome.

En el caso de EtherApe una característica en particular decidió la balanza a favor de la inclusión de Gnome: el componente *canvas*. Este componente simplifica en sobremanera la presentación del diagrama de la red, e incluirlo ayudaba a conseguir un producto funcional en un lapso de tiempo mucho más corto.

Una vez que se decide usar Gnome, se aprovecha la circunstancia para sacar partido de otras características que ofrece el entorno. Así pues, algunas de estas son:

- El componente *canvas*.

  Como ya se ha dicho este componente es fundamental a la hora de la presentación del diagrama. La versión inicial de EtherApe que no utilizaba Gnome debía "pintar" en un componente de dibujo cada uno de los elementos diagrama en cada paso. Igualmente era necesario utilizar una técnica de doble memoria intermedia para evitar el parpadeo de la pantalla, además de necesitar código adicional para asegurar un refresco cada vez que cualquier otro elemento del sistema de ventanas obstruía parcialmente el diagrama.

  El canvas evita todos estos problemas, porque no es simplemente un área de dibujo, sino un objeto que es capaz de almacenar primitivas de dibujo y sus características aliviando al programador de las complicadas tareas gráficas. Básicamente al canvas se le ordena mantener en pantalla un círculo de un radio y color determinados en un punto determinado y el componente se encarga de todo lo demás. Las características pueden ser modificadas con posterioridad y el canvas se actualizará automáticamente.

- Documentación en línea

  La segunda característica más visible que hace evidente que EtherApe es una aplicación Gnome es la presencia de documentación en línea.

  Gnome define una serie de estándares sobre cómo debe organizarse la documentación de una aplicación, y provee además de una interfaz de programación que servirá luego para lanzar el visor de la ayuda.

  La documentación se escribe utilizando un lenguaje de estructurado de marcadores (SGML), la implementación dedicada a la documentación denominada Docbook. A partir de un único archivo de texto utilizando este lenguaje es posible generar múltiples formatos de salida: postscript, HTML, XML, texto llano, TeX, etc.

```
<!-- ============= Introduction ============================== -->
  <sect1 id="intro">
    <title>Introduction</title>

    <para>

    <application>EtherApe</application> is a graphical network monitor
      for Unix modeled after etherman. Featuring ether, ip and tcp
      modes, it displays network activity graphically. Hosts and links
      change in size with traffic. Protocols are color coded. It
      supports ethernet, fddi, ppp and slip devices. It can filter
      traffic to be shown, and can read traffic from a file as well as
      live from the network.
    </para>

    <para>
      <application>EtherApe</application> is also a tool for gathering
      network statistics, and the data can be presented in a number of
      different ways.
    </para>

    <para>
      <application>EtherApe</application>
    </para>

    <para>
      To run <application>EtherApe</application>, select
      <menuchoice>
          <guisubmenu>Applications</guisubmenu>
          <guimenuitem>EtherApe</guimenuitem>
      </menuchoice>
      from the <guimenu>Main Menu</guimenu>, or type
      <command>MYGNOMEAPP</command> on the command line.
  </para>

<!--
    <para>
      <application>EtherApe</application> is included in the
      <filename>GNOME-PACKAGE</filename> package, which is part of the
      GNOME desktop environment. This document describes version
      &version; of <application>EtherApe</application>.
    </para>
-->
  </sect1>
```

Figura 4 – Ejemplo de código Docbook del fichero de ayuda

Para la presentación en línea de la ayuda se usa el formato HTML, utilizando un visor de HTML hecho a medida. En el futuro se espera cambiar a un formato XML

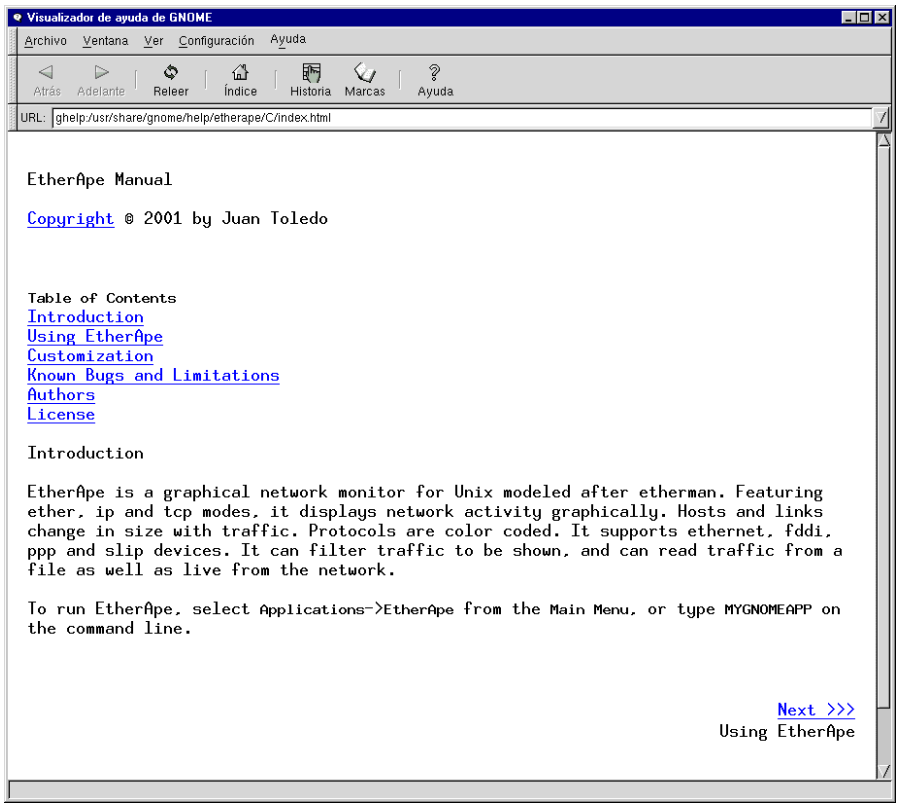


Figura 5 – Visor de ayuda de Gnome

- Almacenamiento de preferencias

  Gnome incluye un conjunto de funciones de biblioteca que permite almacenar información en un archivo específico para a aplicación.

  Su uso más inmediato es el guardar el conjunto de variables que componen las preferencias de la aplicación, de modo que una vez que el usuario ha configurado el programa a su gusto no tenga que volver a hacerlo la siguiente vez que arranque el programa.

  El funcionamiento es parecido al del registro de un sistema Windows, salvo que la información se guarda con formato ASCII en un archivo de texto específico para cada usuario en lugar de hacerlo en binario en un archivo genérico del sistema.

  De esta manera es más fácil detectar errores y administrar el sistema, a la vez que es más robusto ante la corrupción de archivos.

- Almacenamiento de historiales

  Uno de los componentes gráficos avanzados de Gnome es la entrada de texto con almacenamiento de historial. Cada vez que se introduce una entrada nueva se añade a la lista desplegable que incorpora el componente.

Además, estas entradas se van almacenado en el mismo archivo de configuración que se mencionó en el punto anterior, pero mientras que guardar las preferencias hay que hacer una llamada explícita a la biblioteca, en el caso de los historiales es un proceso automático.

EtherApe utiliza entradas de este tipo en el diálogo de carga de ficheros de captura y en la entrada de filtros de captura. La capacidad de un programa de mantener cierta memoria sobre el uso que se hace de él es altamente valorada por los usuarios, ya que obvia la necesidad de repetir pasos que ya se hayan dado en el pasado.

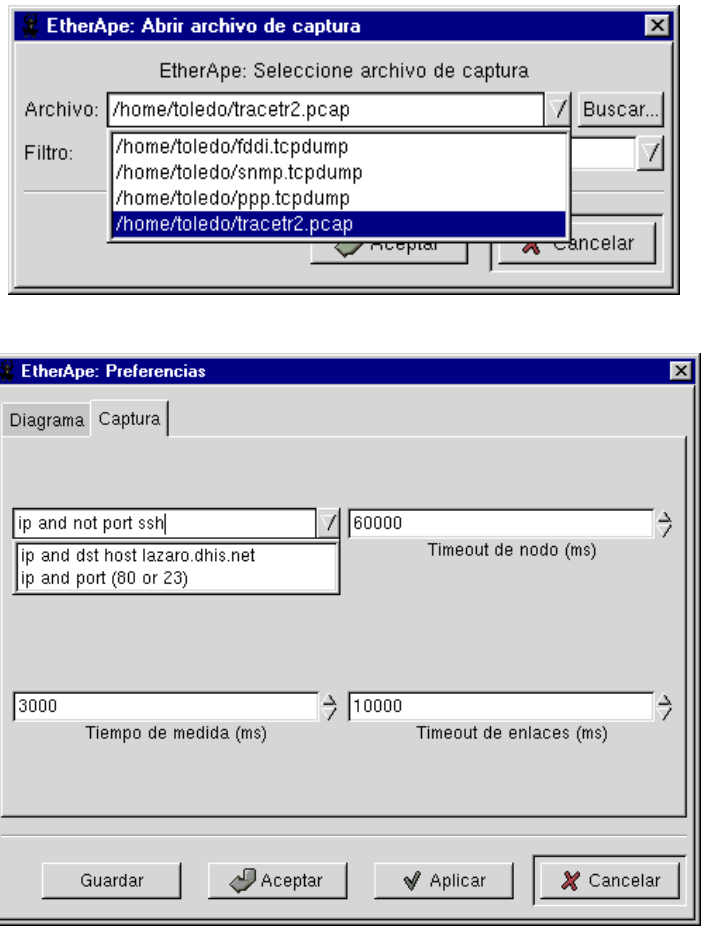


Figura 6 – Entradas de texto con historial

3.5 Glade/Libglade

Puesto que Gtk+ y Gnome son bibliotecas de C, la manera de diseñar y poner en marcha la interfaz de usuario es mediante llamadas a funciones. Este es un sistema muy poco intuitivo, tedioso y que induce fácilmente a errores.

Por cada elemento de la interfaz gráfica hay que especificar dónde se coloca, cuáles son sus atributos por defecto y cuáles y cómo son los atributos que ha de gestionar

En los Entornos Integrados de Desarrollo (IDE) modernos, el problema se resuelve mediante programas accesorios que permiten "dibujar" una interfaz de usuario y asignar propiedades a los elementos. En un paso posterior el IDE escribe el código que implementa esa interfaz, para que el programador pueda integrarlo a su programa.

```
diagram_only_toggle =
    gtk_toggle_button_new_with_label (_("Click to toggle"));
  gtk_widget_ref (diagram_only_toggle);
  gtk_object_set_data_full (GTK_OBJECT (diag_pref), "diagram_only_toggle",
                                diagram_only_toggle,
                                (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (diagram_only_toggle);
  gtk_box_pack_start (GTK_BOX (vbox11), diagram_only_toggle, FALSE, FALSE, 2);
  gtk_tooltips_set_tip (tooltips, diagram_only_toggle,
                            _("Toggle whether text is shown on the diagram"),
                            NULL);

  label30 = gtk_label_new (_("No text"));
  gtk_widget_ref (label30);
  gtk_object_set_data_full (GTK_OBJECT (diag_pref), "label30", label30,
                                (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (label30);
  gtk_box_pack_start (GTK_BOX (vbox11), label30, FALSE, FALSE, 0);
  gtk_label_set_justify (GTK_LABEL (label30), GTK_JUSTIFY_LEFT);

  vbox5 = gtk_vbox_new (FALSE, 0);
  gtk_widget_ref (vbox5);
  gtk_object_set_data_full (GTK_OBJECT (diag_pref), "vbox5", vbox5,
                                (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (vbox5);
```

Figura 7 – Ejemplo de código C generado por Glade

3.5.1 Glade

El programa que cumple esta función en el entorno Gnome es Glade. Mediante Glade es posible diseñar prácticamente toda la interfaz de usuario de un programa: las ventanas, los diálogos, los menús... Como si se tratara de un programa de dibujo, Glade presentan una caja de herramientas donde se encuentran todos los componentes gráficos que se pueden utilizar, tanto de Gtk+ como de Gnome.

Una vez especificado diseñada la interfaz, el programa procede a escribir los archivos en C. Por un lado están las funciones que harán que aparezca en pantalla la IGU. Por otro lado el esqueleto de las funciones que se habrán de ejecutar cuando se active alguno de los procedimientos de los componentes gráficos.

Realmente Glade va más allá, y de hecho puede escribir un programa completo que puede ser compilado y ejecutado. Incluso si el programa no necesita de procedimientos no relacionados con IGU (algo excepcionalmente raro, pero un programa que deba simplemente plantar en pantalla una serie de avisos), el programa estará completo sin necesidad de haber escrito una sola línea de código.

Sin embargo Glade también tiene algunos problemas. Conforme el programa que se está desarrollando evoluciona es más que probable que se habrán de añadir nuevas ventanas y controles. Glade fuerza a que todo el código que genera se almacene únicamente en dos archivos: interface.c para la generación de la IGU y callbacks.c para que el programador complete las funciones de rellamada de la interfaz.

Para un programa que va adquiriendo cierta envergadura este es un sistema engorroso, porque impide establecer cierto orden en la estructura del programa, separando las distintas partes que componen la interfaz. Por otro lado el algoritmo que debe detectar cuando se ha implementado ya una de las funciones de rellamada no es perfecto, y en algunos casos nos encontramos con un callbacks.c que define dos veces la misma función.

3.5.2 Libglade

Libglade aporta una ingeniosa solución al problema anterior, y representa un buen ejemplo de innovación aportado por el mundo del software libre.

Los programas que usan Libglade separan radicalmente la interfaz de usuario del resto del programa. Ni una sola línea de código se dedica a la implementación de la IGU, sino que se genera automáticamente en tiempo de ejecución a partir de un archivo de definición en formato XML generado por Glade.

Así pues, cuando EtherApe hace la llamada apropiada a Libglade, esta biblioteca carga el archivo etherape.glade y presenta la interfaz de usuario. Cuando se activa alguna función de la IGU que requiera de una función de rellamada es Libglade la que se encarga de efectuarla, sabiendo cuál es la función a la que debe llamar puesto que está especificada en etherape.glade.

Por otro lado, existen funciones que permiten al programa principal acceder a los componentes gráficos que ha generado Libglade, de modo que se puedan modificar en detalle sus propiedades en tiempo de ejecución, si es que eso es necesario.

Mediante este sistema se resuelve el problema de la estructura del código, puesto que lo que antes iba en interface.c se hace innecesario; y dado que las funciones

que antes iban en callbacks.c pueden estar definidas en cualquier parte del programa con tal de que sean funciones globables accesibles por la biblioteca en tiempo de ejecución.

Pero además Libglade hace posible algo verdaderamente sorprendente: ya no es necesario volver a compilar la aplicación para cambiar la interfaz de usuario, basta con editar con Glade el archivo de definición de la interfaz. Esto da lugar a ciclos de desarrollo aún más rápidos de los que permite Glade por sí mismo, pero también permite que sean personas diferentes las que trabajen en la interfaz y el código, o que los usuarios personalicen a su gusto la apariencia del programa con una increíble flexibilidad sin tener que cambiar en absoluto el programa.

Y por si fuera poco todo eso se consigue sin tener que hacer uso de ningún interprete. Todo sigue siendo código C compilado, con lo cual no se pierde nada en velocidad de ejecución

```
<widget>
          <class>GtkMenuBar</class>
          <name>menubar1</name>
          <shadow_type>GTK_SHADOW_NONE</shadow_type>

          <widget>
            <class>GtkMenuItem</class>
            <name>file1</name>
            <stock_item>GNOMEUIINFO_MENU_FILE_TREE</stock_item>

            <widget>
              <class>GtkMenu</class>
              <name>file1_menu</name>

              <widget>
                <class>GtkPixmapMenuItem</class>
                <name>open</name>
                <signal>
                    <name>activate</name>
                    <handler>on_open_activate</handler>
                </signal>
                <stock_item>GNOMEUIINFO_MENU_OPEN_ITEM</stock_item>
              </widget>

...
```

Figura 8 – Ejemplo de código XML del fichero etherape.glade

3.6 Ethereal

Hasta ahora se han tratado las bibliotecas más importantes con las que enlaza EtherApe para poder ser compilado. Habíamos comentado que utilizar bibliotecas externas ayuda a crear el programa más rápidamente, además de obtener un programa más robusto y fiable (siempre que las bibliotecas están bien hechas, claro está).

Sin embargo el hecho de que EtherApe sea software libre abre las puertas a un extensísimo conjunto de recursos de programación que no necesariamente tiene por qué estar disponible en forma de una biblioteca: hablamos del mismo código fuente de cualquier otro programa que sea software libre.

A pesar de que EtherApe es un programa relativamente novedoso (es la primera implementación de software libre de un monitor gráfico de red), no todos los algoritmos que debe implementar son un campo nuevo. En particular, al igual que ocurre con EtherApe, Ethereal debe identificar cuál es la pila de protocolos que transporta cada paquete para cumplir su objetivo de un análisis detallado.

Así pues, puesto que el código de Ethereal está expuesto a todo aquel que quiera detenerse a examinarlo, constituye una referencia casi más útil que los estándares que definen a los protocolos, puesto que expresa algorítmicamente el modo de identificarlos.

EtherApe obtiene un provechoso partido de esta situación, incorporando directamente a su código algunos de los procedimientos desarrollados originalmente para este otro programa. Si bien en la mayor parte de los casos no es posible copiar y pegar directamente puesto que las estructuras de datos que se manejan no son idénticas, sí que se convierte en una tarea mucho más sencilla añadir funcionalidad en este sentido.

No sólo se ha aprovechado la identificación de protocolos, también problemas menores como la representación textual de direcciones ethernet o IP a partir de su representación numérica son problemas comunes por los que resultaría inútil perder el tiempo en reimplementar.

Por otro lado hay que destacar que este comportamiento dista mucho de ser parasitismo. Muy al contrario es una simbiosis por la que todo el mundo gana. EtherApe reduce su tiempo desarrollo y aumenta en funcionalidad, y a su vez el conjunto de los usuarios de EtherApe (entre los que se incluyen los desarrolladores de Ethereal) disponen de una nueva herramienta con la que hacer su trabajo.

Puesto que la funcionalidad de Ethereal está embebida directamente en el código EtherApe, este programa no es una dependencia propiamente dicha. Es decir, no es necesario disponer de éste para poder utilizar aquel. Sin embargo, puesto que su contribución es fundamental para el éxito del proyecto se ha decidido incluirlo aquí.

3.7 Licencias del código externo

Siempre que se vaya a usar código producido externamente se ha de estar en cumplimiento de las licencias correspondientes para encontrarnos dentro de la legalidad.

En el caso de proyectos comerciales tradicionales esto implica pagar una cantidad acordada para adquirir el derecho de uso y poder enlazar la correspondiente biblioteca.

Sin embargo EtherApe se ha producido usando enteramente software libre, y por tanto las licencias de sus dependencias lo reflejan. Libpcap se distribuye bajo la licencia BSD, Glib, Gtk+, las bibliotecas de Gnome y la biblioteca estándar de C se encuentran bajo la licencia LGPL, y Ethereal se distribuye con licencia GPL.

La licencia BSD se resume básicamente en que no hay límites en el uso del software que se distribuye con ella, con tal de que en la documentación del trabajo que se produzca contenga una mención al producto en el que se basa.

La licencia LGPL ("Lesser General Public License") es común a muchas bibliotecas de software libre. Lo que indica es que la biblioteca es en sí misma software libre, pero no se limita el uso a cualquier tipo de programa que quiera usar la funcionalidad que aporta.

Como ya se ha indicado más arriba, la licencia GPL es un tipo de licencia que permite el uso ilimitado del software, pero obliga a que trabajos derivados estén asimismo también recogidos bajo la misma licencia.

Así pues queda claro que sólo el uso de código de Ethereal fuerza a que EtherApe sea también software libre bajo licencia GPL. Sin embargo a juicio del autor hay poco que ganar manteniendo el programa como software propietario, y muchas ventajas que perder.

4 Estructura interna

EtherApe es una aplicación, que como casi todas las aplicaciones basadas en una IGU no siguen un camino lineal, sino que están controlada por eventos.

Tres son los eventos principales a los que EtherApe debe responder

- Llegada de una trama a una interfaz de red

  Cuando esto ocurre el programa analiza el paquete en cuestión y actualiza todas las estructuras de datos que estén relacionadas con dicho paquete.

- Interacción con la interfaz de usuario

  En cualquier momento el usuario puede parar o iniciar una captura, cambiar la interfaz de red de donde leer los datos o finalizar el programa totalmente. En cualquiera de estos casos se debe actuar de acuerdo con el comando que se ha ejecutado.

- Vencimiento del temporizador de refresco

  El diagrama principal de red se actualiza cada cierto tiempo, fijado por el usuario. Las ventanas de información estadísticas también deben ser refrescadas, aunque lo hacen con menos frecuencia.

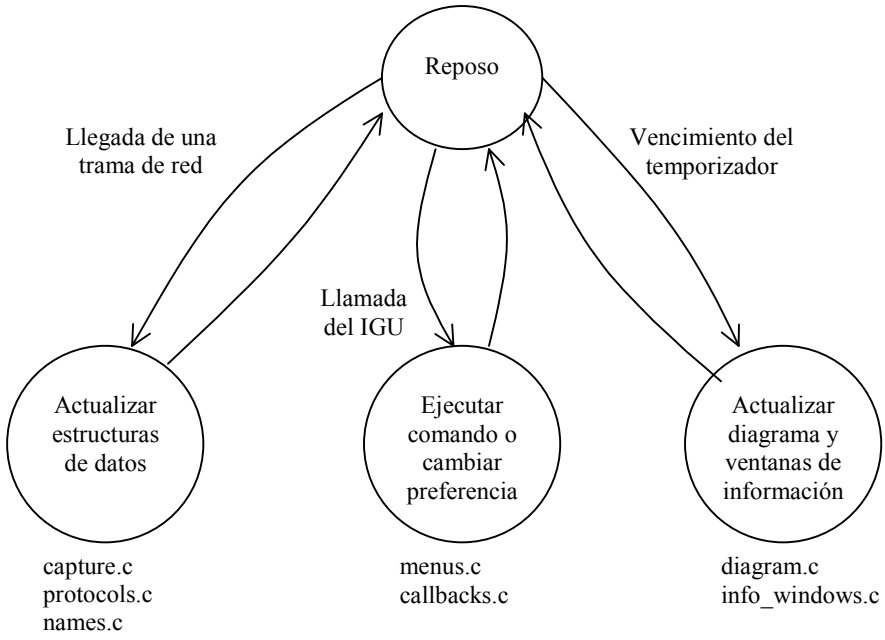


Figura 9 – Diagrama de eventos de EtherApe

En la figura se refleja también en qué parte del código se implementa la funcionalidad que requiere cada uno de los eventos.

Motor de captura

Para cumplir los objetivos que se han planteado EtherApe debe ser capaz de escuchar *la red y* entender *lo que se está diciendo. La información acumulada se almacenará para su posterior presentación.*

1 Captura portable de paquetes

1.1 Adquisición de tramas

Lo mínimo indispensable para poder hacer cualquier análisis del tráfico en la red es poder adquirir información de lo que allí está ocurriendo. Se trata de conseguir que el programa obtenga una copia de cada paquete que llega a la interfaz de red.

Sin embargo por lo general las tarjetas de red están configuradas para atender exclusivamente el tráfico que le pueda concernir, esto es, aquél que le está dirigido directamente y el tráfico de broadcast. Esto se hace para minimizar la carga que se pudiera provocar en el nodo al tratar de procesar tramas que serán, en la práctica, inútiles. Si se quiere escuchar todo el tráfico, no sólo el descrito antes, se ha de configurar la interfaz de red en lo que se llama "modo promiscuo". Eso permite al sistema operativo tener acceso a todas las tramas recibidas, y así también, por tanto, a las aplicaciones que las reclamen.

Como se puede imaginar este proceso es característico del sistema operativo que se vaya a utilizar, y también depende de la interfaz de red. La manera de resolver este problema de modo portable es utilizando la biblioteca Libpcap

Utilizando las funciones adecuadas se logra que Libpcap haga una copia de cada trama y nos proporciones un puntero a su copia. EtherApe hace un tratamiento de ese paquete y almacena sólo la información que le es interesante.

1.2 Integración en el bucle de eventos

Para que el bucle principal haga una llamada a las funciones de procesado de paquetes cuando llega uno, se ha de registrar de alguna manera en el bucle de eventos.

De manera general, Gdk (y través de él, Gtk+) permiten registrar funciones que serán llamadas cuando se cumplan ciertas condiciones en un descriptor de archivo.

En particular, cuando inicializamos libpcap para hacer una captura podemos pedir un descriptor de archivo que usaremos en la llamada de `gdk_input_add` para poder hacer el tratamiento de cada nuevo paquete que llegue

2 Identificación de nodos

La primera tarea a realizar cuando se analiza una trama es averiguar quienes son los nodos origen y destino de la comunicación. De esta manera el tráfico que transporte esta trama será agregado a los nodos correspondientes y quedará reflejado en el gráfico.

2.1 Definición de identidad

Lo interesante en este punto es que no hay una única definición para nodo origen y destino en una trama en particular. Usando la terminología OSI, cada nivel de la pila de un nodo está manteniendo su propia conversación con otro nivel equivalente en la pila de otro nodo, que ni siquiera tiene que ser el mismo.

Así, por ejemplo, tomemos el ejemplo de un navegador de internet que está recibiendo una página web.

Si lo miramos al nivel de área local, todo el tráfico se intercambia entre el PC cliente y el encaminador de esa red local. Desde el punto de vista de redes IP, se está estableciendo una comunicación punto a punto ente el PC cliente y el nodo remoto que alberga el servidor web. Pero se puede ir más allá. En la práctica, para ver una página web es necesario hacer más de una consulta. Al menos una para descargar el documento HTML y otra por cada imagen que tenga la página. Para cada una de estas comunicaciones se establece una conexión TCP distinta, todas ellas dirigidas al puerto 80 del servidor, pero cada una iniciada desde un puerto distinto del PC cliente.

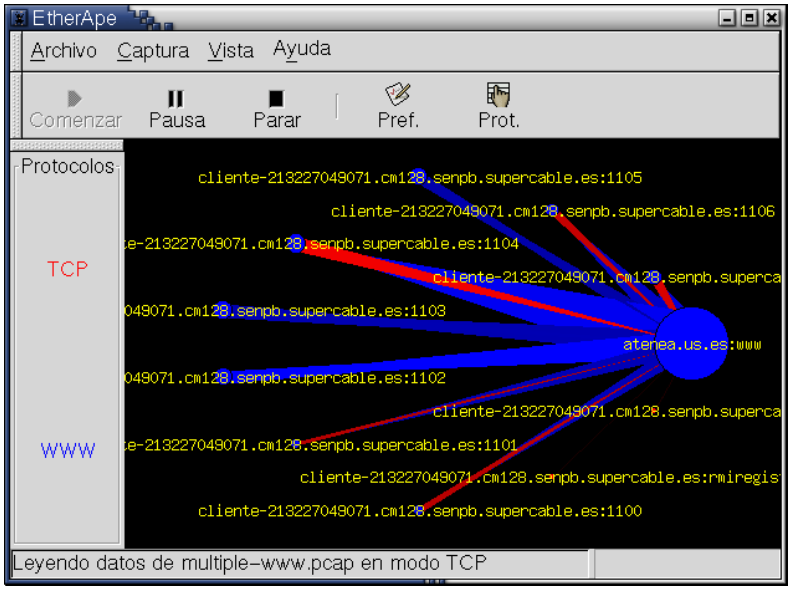


Figura 10 – Modo TCP: cada puerto es un nodo

Queda claro por tanto que en cada momento hay que tomar una decisión con respecto a como se define la identidad de los nodos. Es posible ordenar a EtherApe que actúe de una manera u otra, a través de lo que se han denominado "modos". Así pues, se puede elegir entre los modos Ethernet, IP, o TCP, para el caso anterior. Pero también existen los modos FDDI y Token Ring para redes que no están basadas en tecnología Ethernet.

Nótese que en ciertos tipos de topología, como en las redes punto a punto PPP no existe el concepto de identidades al nivel de capa de enlace, puesto que sólo puede haber un origen y un destino y cada uno sabe quién es el otro. Por eso EtherApe inhabilita los modos de capa de enlace cuando está leyendo datos de una interfaz PPP o SLIP.

2.2 Casos soportados

Ahora veremos más específicamente cómo se definen los nodos en cada modo en particular

- Ethernet

  El identificador de nodo se corresponde con la dirección hardware de la interfaz de red. En la cabecera de los tres tipos posibles de comunicación Ethernet (802.3, ETH-II y 802.2) siempre aparece la pareja de direcciones que identifican el origen y el destino. Cada una de esas direcciones es un conjunto de seis octetos.

- FDDI y Token Ring

  Si bien en estos dos casos hay información adicional, igualmente se encuentra la pareja de direcciones hardware que coincide en tamaño, aunque la posición de esta información esté situada de manera diferente a como lo está en las tramas Ethernet.

- IP

  Una vez localizada la cabecera IP en la trama, no hay más que leer las direcciones de origen y de destino, que están también siempre presentes. En este caso cada una de ellas es un conjunto de 4 octectos.

  Hay que hacer notar que si bien todas las tramas que se capturen en una interfaz de red deben albergar una cabecera que se corresponde con el nivel dos de ese medio físico, no ocurre lo mismo con los protocolos de nivel superior. Este es el caso de IP y TCP.

  Antes de intentar buscar una identidad IP a un nodo hay que asegurarse de que realmente lo que tenemos es tráfico IP. Para ello lo que se hace es que cada vez que se está utilizando uno de estos modos el programa agrega automáticamente un filtrado a la interfaz que elimina las tramas que no correspondan a ese tipo de tráfico.

- TCP

  Este es un ejemplo más complejo de definición de identidad. En los casos anteriores lo único que se hacía era copiar directamente las direcciones de origen y destino que se encontraban en las cabeceras correspondientes.

  En el caso de TCP, la cabecera sólo indica los puertos de origen y de destino, pero estos por sí solos no identifican los extremos de la comunicación. Así pues el programa crea una identificación concatenando los cuatros octetos de la dirección IP con los dos octetos adicionales del puerto TCP.

  De este modo en modo TCP cada nodo está definido por una clave de seis octetos, y la heurística que se necesita para comparar nodos entre sí es ligeramente más complicada, puesto que hay que mirar información en dos cabeceras distintas.

3 Análisis de la pila de protocolos

Además de la representación gráfica de la información, la segunda característica que hace que EtherApe se destaque entre otras alternativas para monitorizar una

red es su capacidad de hacer un análisis completo de la pila de protocolos de cada paquete.

Es común que otras aplicaciones distingan entre varios tipos fundamentales de tráfico, separando por ejemplo el tráfico IP del IPX, o distinguiendo entre ICMP, UDP y TCP en el tráfico IP. EtherApe va mucho más allá tratando de distinguir completamente para qué aplicación concreta se está enviando cada paquete que circula por a red.

Como se comentó antes, es aquí donde el estudio del código fuente de Ethereal ha sido instrumental. Ethereal debe reconocer cada protocolo y presentar de manera legible cada uno de los campos que componen la cabecera de todos los protocolos.

Puesto que EtherApe sólo debe identificar los protocolos, no destripar su contenido, basta con replicar los algoritmos de identificación.

3.1 Sistemas de identificación

No todos los casos son iguales a la hora de identificar un protocolo. El sistema varía según la altura de la pila de protocolos en la que nos encontremos, y en algunos casos es prácticamente imposible implementar un esquema que nos dé la certeza sobre el protocolo que se está usando.

- Protocolos de nivel de enlace

  Cuando se pide a libpcap que inicie una captura sobre una interfaz de red o un archivo de captura, libpcap devuelve una estructura que contiene información sobre la captura que se va a realizar. Entre otras cosas un campo de esa estructura nos indica qué medio físico soporta la red que vamos a monitorizar.

  En la mayoría de los casos cada medio físico utiliza sólo un protocolo de nivel de enlace. De esta manera se puede identificar directamente la presencia (o ausencia en su caso) del protocolo de nivel 2.

  En las redes ethernet sí hay tres posibilidades distintas, que se corresponden con las tres variedades de tramas ethernet que existen, 802.2, 802.3 y Ethernet-II. Sin embargo hay métodos para identificar cada caso unívocamente, sin posibilidad de error, y por tanto no es un problema.

  Si es más importante identificarlos en lo que respecta al análisis de los niveles superiores, puesto que la heurística es diferente en cada caso.

- Identificación definida por estándar

  Este sistema es igualmente sencillo en los casos en que los protocolos tienen reservados un campo para identificar el protocolo de nivel superior que transportan.

  Ejemplos de este caso son las tramas Ethernet-II (que pueden transportar IP e IPX, por ejemplo) o las cabeceras de IP (ICMP, TCP, UDP, IGMP, GRE, OSPF, EGP, etc.)

- Puertos TCP y UDP asignados

  En las cabeceras de TCP y UDP no hay un estándar que defina taxativamente el tipo de tráfico que transportan. Sin embargo el IANA (Internet Assigned Numbers Authority) sí ha hecho un esfuerzo de normalización para ayudar a la interoperabilidad de las aplicaciones de red.

  Los 65536 puertos posibles están divididos en tres grupos: puertos de sistema (0-1023), puertos registrados (1024-49151) y puertos dinámicos o privados (49152-65535).

  En los dos primeros casos IANA publica una lista de servicios que pueden estar escuchando en cada puerto. En problema es que si bien es casi seguro que una conexión TCP a un puerto 80 sea de HTTP, no es forzosamente cierto, sobre todo con los puertos mayores que 1024, que pueden ser abiertos por cualquier usuario sin privilegios.

- Heurística

  Si los mecanismos anteriores no funcionan todavía es posible ir probando si la estructura del paquete se corresponde con algún protocolo conocido. Este comportamiento, que sí está implementado en Ethereal, aún no lo está en EtherApe.

3.2 Identificación de conversaciones

En algunos casos, una vez que se ha establecido una comunicación entre dos entidades en nodos distantes, ambos mantienen un estado que hace innecesario referir explícitamente en cada paquete algunos parámetros de la comunicación.

Esto ocurre por ejemplo en los mensajes de respuesta de los protocolos basados en RPC. Sólo en la petición se indica cuál es el servicio que se va a utilizar, pero en la respuesta no se hace, puesto que se considera implícito.

Igualmente ocurre con el funcionamiento pasivo del protocolo FTP. En un momento dado el servidor indica al cliente que ha abierto un puerto adicional

aleatorio (no registrado) para atender exlusivamente una transferencia de archivos con ese cliente. En este caso indentificar tramas sucesivas por el número de puerto puede dar lugar a error.

EtherApe resuelve estos problemas porque mientras que analiza los paquetes está buscando paquetes que indiquen el inicio de una conversación. Si los encuentra, almacena los parámetros de esa conversacion, y luego puede usar esos datos para tratar de indetificar el protocolo de subsiguientes tramas.

4 Extracción de nombres

Hasta ahora hemos estado tratando las identidades de los nodos como un concepto interno destinado a localizar el origen y el destino del tráfico que se cursa. Sin embargo también es necesario informar al usuario de esos mismo detalles, pero si es posible hacerlo de una manera que sea más comprensible para una persona que un conjunto de octetos.

4.1 Definición de nombre

Para este proyecto un nombre es un pareja de cadenas alfanuméricas que se asocian a un protocolo. La primera cadena es la representación numérica de una identidad. La segunda es la representación más fácilmente reconocible para el usuario de esa misma identidad. Esta definición general es compleja, y se entenderá mejor lo que se quiere decir con un ejemplo.

Para una comunicación entre dos nodos IP, la identificación que se usa es un conjunto de cuatro octetos. EtherApe guarda esta identificación internamente en una palabra de 32 bits, digamos *0xC193A094*. La versión **numérica** del nombre para este caso es *193.147.160.148*, mientras que la versión **resuelta** es *www.esi.us.es*.

4.2 Multiplicidad de nombres

Mientras que para un modo en particular un nodo sólo tiene una única identidad, ese mismo nodo puede estar relacionado con un extenso número de nombres, por dos motivos distintos.

Por un lado es posible asignar un nombre al nodo casi a cada nivel de la pila de protocolos. Por ejemplo, para un ordenador conectado a una red ethernet hay que pensar el la dirección ethernet, en la dirección IP, y en la identificación propia de un nodo de una red Microsoft para compartir archivos.

Por otro lado, incluso para un mismo protocolo es probable que un mismo nodo se identifique con más de un nombre. Así, por ejemplo, trabajando en el modo IP, ese nodo está relacionado con múltiples nombres TCP (combinación de dirección IP y puerto TCP)

Por todo ello a la hora de diseñar las estructuras de datos se ha tenido en cuenta esta situación para dotarlas de la suficiente flexibilidad para poder almacenar toda esta información con éxito.

4.3 Resolución de nombres

La coversión entre el identificador y la pareja de nombres que le corresponde es específica para cada protocolo. Veremos algunos de los casos.

- Direcciones Ethernet

  El nombre numérico se representa utilizando el formato más común: separando cada octeto representado en hexadecimal por dos puntos

  El nombre resuelto se obtiene a partir de tablas de traducción si es que estas están instaladas en el sistema. EtherApe busca en primer lugar el archivo /etc/ethers para intentar traducir la dirección a un nombre reconocible. Si no está disponible busca a continuación /etc/manuf, que es un archívo genérico que relacion los tres primeros octectos de la dirección con el fabricante de la tarjeta.

- Direcciones IP

  El nombre numérico es el habitual. Cuatro números decimales separados por un punto.

  La resolución completa del nombre tiene sus propios problemas añadidos. Es posible pedir al sistema que nos informe del nombre que se corresponde con una direción IP, pero esta función bloquea hasta que se obtiene un resultado.

  Si se hiciera directamente, entonces el programa se quedaría congelado cada vez que apareciera una nueva dirección IP, y esto es lo contrario de lo que se pretende con el programa: la capacidad de poder echar un vistazo de manera inmediata a lo que ocurre en la red.

  El modo de resolverlo es introduciendo funciones que resuelvan asíncronamente. De ese modo EtherApe no resuelve completamente el nombre hasta que el servidor de DNS no está preparado para ello. Esta funcionalidad está implementada en dns.c, y es otro ejemplo de las ventajas

del software libre, puesto que es una adaptación directa de la implementación original que se hizo para mtr[1].

- Direcciones TCP

  Es un caso casi idéntico al anterior. A la dirección IP obtenida se le agregan dos puntos y el número de puerto. En la versión resuelta se consulta el número de puerto en el fichero de sistema /etc/services para buscar el nombre del servicio.

- Nombre NetBIOS o SMB

  En los protocolos que usan NetBIOS, el nombre de cada una de las estaciones que participan en el protocolo se explicita en cada trama, y únicamente es necesario extraerla. El único inconviente es que en el caso de NetBIOS el nombre no se transmite como texto en claro, sino que está codificado y es necesario traducirlo.

  Ese mismo nombre se transmite en ocasiones para ciertas funciones del protocolo SMB. En este caso el nombre se transmite en claro, así que es más fácil recuperarlo.

  Además del nombre, en ambos casos el último octecto es un código que se utiliza para especificar la función que tiene ese nodo dentro de la red. Puesto que en SMB un nodo puede tener más de una función (cliente, servidor, servidor de nombre, comprobación de identificaciones) el mismo nodo tiene un conjunto de nombres SMB asociadas.

  En este caso no es posible dar una representación estrictamente numérica, así que se limita al nombre sin tener en cuenta el código de función.

  En la representación resuelta sí se tiene en cuenta ese código, y se representa textualmente su significado.

[1] El programa mtr es una versión avanzada del clásico traceroute. Mantiene un estadística constante sobre cada punto de una ruta, y no bloquea mientras está intentado resolver el nombre de cada encaminador. La página web de mtr está en http://www.bitwizard.nl/mtr/

Interfaz de usuario

La peculiaridad que caracteriza a EtherApe es la representación gráfica en tiempo real de los datos capturados. Un diagrama fácilmente reconocible es el objetivo fundamental del programa. Sin embargo no es la única información que EtherApe presenta al usuario, también hay estadísticas de datos agregados. Todo esto, junto con las opciones de configuración de EtherApe es lo que define la interfaz con el usuario.

1 El diagrama de red

El diagrama de red es el componente más destacado de la interfaz de usuario, así como el objetivo fundamental del programa. A partir de los datos capturados se presenta en pantalla un diagrama que traduce de manera gráfica el tráfico en la red.

Los círculos representan a cada uno de los nodos identificados y las líneas que los unen es el tráfico cursado entre ellos. El tamaño de los elementos varía según el ancho de banda que se esté usando, y el color de los elementos se traduce en el protocolo más utilizado, bien por el nodo, bien por un enlace en particular.

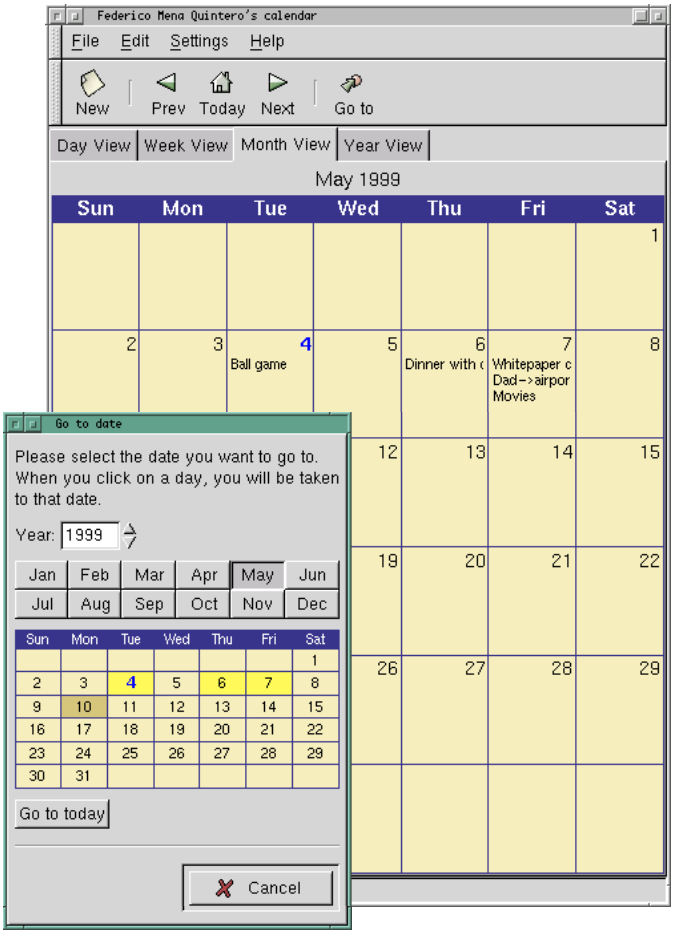
1.1 El componente canvas de Gnome

El motivo fundamental por el que EtherApe es una aplicación del entorno Gnome es por la facilidad que su componente canvas aporta a la implementación del diagrama.

The GNOME canvas is a high-level engine for creating structured graphics. This means the programmer can insert graphical items like lines, rectangles, and text into the canvas, and refer to them later for further manipulation. The programmer does not need to worry about repainting these items or generating events for them; the canvas automatically takes care of these operations.

El canvas de Gnome es un motor de alto nivel para la creación de gráficos estrucutrados. Esto quiere decir que el programador pueder insertar elementos gráficos como líneas, rectángulos y texto dentro del canvas, y referirse luego a elos para manipularlos. El programador no debe preocuparse con el redibujado de los elementos, o de la generación de eventos asociados, puesto que el canvas lo hace automáticamente.

Además de los elementos descritos, el canvas permite que se creen nuevos elementos, y agrupar un conjunto de elementos para tratarlo como uno sólo. No es de extrañar por tanto que el canvas tenga un enorme éxito entre los programadores de aplicaciones para Gnome.



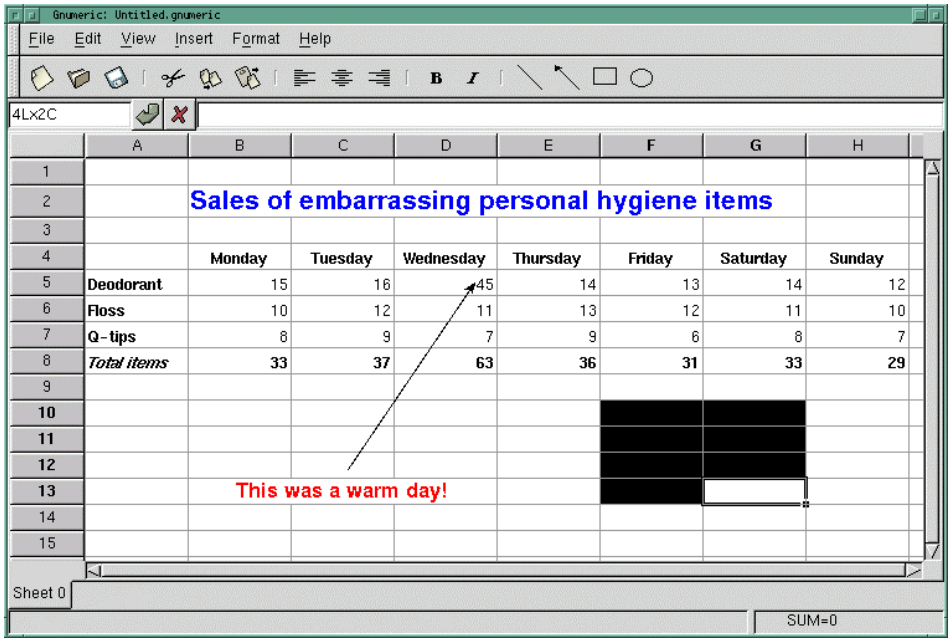


Figura 11 – Ejemplos de canvas: Calendar y Gnumeric

El canvas implementa conceptos complejos, como la multiplicidad de sistemas de referencia para localizar los elementos o la técnica del "doble buffering" para evitar parpadeos.

1.2 Elementos que componen el diagrama

Desde el punto de vista del usuario sólo son dos los tipos de elementos del diagrama: los nodos y los enlaces. Sin embargo su estructura interna es ligeramente más compleja.

1.2.1 Nodos

En realidad un nodo está formado por un elemento grupo que engloba a un elemento elipse y a un elemento de texto.

Las propiedades del conjunto que se manipulan después de su creación son la posición del conjunto, el color del círculo, el radio del círculo y el color del texto. Adicionalmente también es posible hacer invisible por completo el texto.

La utilidad del elemento grupo es la de tomarlo como origen de referencia para el posicionamiento de la elipse y el texto. De este modo basta con redefinir la posición del elemento grupo con respecto del canvas para mover el conjunto.

En cada refresco del diagrama se revisan las propiedades de cada nodo, y si alguna de ellas ha cambiado se pone en cola un refresco del diagrama (de hecho

espera a haber actualizado todos los nodos y enlaces antes de redibujar el canvas).

En cada pase se comprueba cuál es la cantidad de tráfico a medir y el protocolo principal del nodo de captura asociado para actualizar el radio y color respectivamente de cada nodo de canvas.

Asimismo se actualiza el texto identificativo con el del nombre más representativo que el nodo tenga especificado en cada momento. Se comprueba además cual es el color de texto designado en ese momento y si se ha indicado que el elemento de texto debe hacerse invisible.

1.2.2 Enlaces

Si se analiza el diagrama con anteción se observará que los enlaces no son rectángulos que unen dos nodos. En lugar de eso se verá que son triángulos, donde un nodo está colocado en el centro de la base y otro en el vértice opuesto.

Esto es así porque durante la captura se ha utilizado una estructura enlace para *cada sentido* de la comunicación. De esta manera se mantiene una estadística más detallada sobre las carácterísticas de la comunicación y se hace aparente en qué sentido se está moviendo más informacion en el diagrama.

En cada actualización la posición del triángulo se actualiza siguiendo como referencia la posición de los elementos grupo de los nodos asociados, y el color se corrige para indicar el protocolo más utilizado.

En el diagrama de ejemplo se observa claramente la distinción entre los dos sentidos de una comunicación. En este caso el cliente está recuperando una página web de un servidor remoto. Se aprecia cómo hay mucho mayor cantidad de tráfico descendente. Por otro lado el color nos indica que mientras que del servidor recibimos mayoritariamente tráfico describiendo la página web (protocolo HTTP), el cliente envía sobre todo tráfico TCP vacío de contenido, que este caso se corresponde con los asentimientos asociados a cualquier conexión TCP.

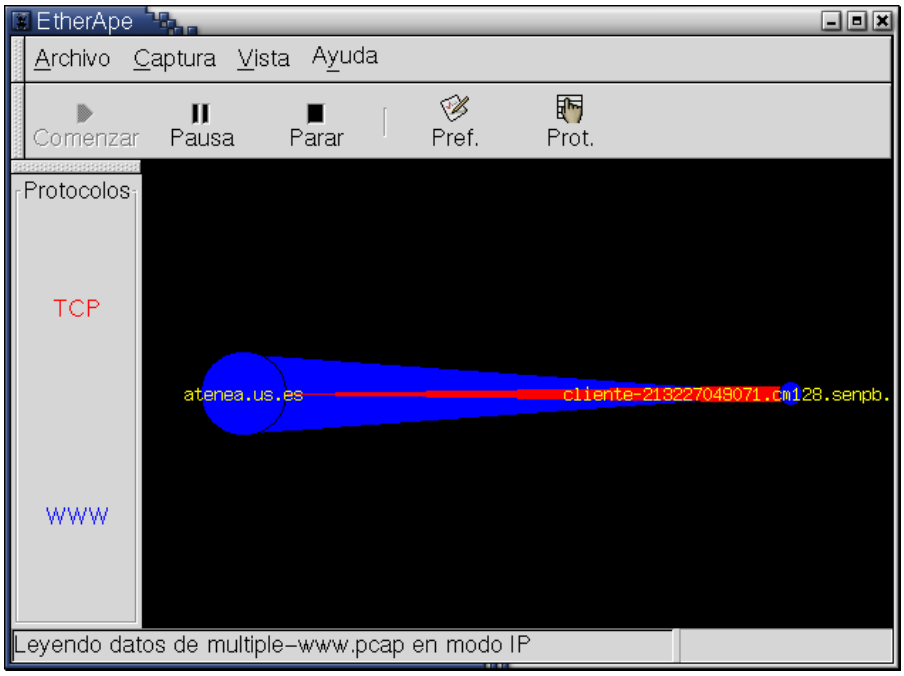


Figura 12 – Ejemplo de comunicación asimétrica

1.2.3 La Leyenda

La leyenda se genera automáticamente conforme se capturan paquetes, asignando los colores disponibles a los protocolos identificados siguiendo una estructura round-robin.

En este momento este es uno de los principales problemas de la interfaz de usuario de EtherApe, puesto que en cada ejecución un mismo protocolo puede tomar un color diferente, y pasado un tiempo se habrán identificado tantos protocolos que la asignación de colores no será unívoca.

Este problema es nuevo en los programas de la clase de EtherApe, puesto que implementaciones anteriores no eran capaces de reconocer tan variedad de protocolos, y en consecuencia era sencillo hacer una asignación estática de colores.

Una posibilidad sería asignar un conjunto cerrado de colores a los protocolos que se consideren más importantes (por ejemplo, HTTP, DNS, NFS), dejando la atribución de los demás a una asignación circular. Sin embargo esto tiene el inconveniente de que cada usuario considerará principales un conjunto diferente de protocolos, con lo que probablemente no habremos arreglado nada.

Así pues la situación se mantendrá hasta que en una revisión posterior del programa se haya desarrollado una solución que permita al usuario seleccionar complentamente cómo desee que actúe el programa, leyendo sus preferencias

bien de un archivo de configuración, bien mediante un diálogo hecho expresamente a ese efecto.

1.3 Estructuras de datos

Lo más importante a destacar es el hecho de que las variables que se usan en el diagrama duplican en muchos casos a las variables que se utilizan durante la captura.

Esta decisión se tomó bien temprano en el diseño, desde el momento que quedó patente que el sistema de captura era lo suficientemente genérico como para ser de utilidad no solo para el sistema de representacion que utiliza EtherApe, sino para otras posibles herramientas de análisis de red.

Por este motivo, previendo que un futuro el motor de captura se convierta en una biblioteca independiente el resto del programa trata de comportarse como si de hecho ya lo fuera, de modo que se facilite una más que probable transición en el futuro.

Así pues nos encontramos con dós árboles binarios adicionales, canvas_nodes y canvas_links, que soportan el grueso de la información que controla el diagrama

2 Las ventanas de información estadística

Conforme el programa fue evolucionando, el motor de captura se iba haciendo más potente, almacenando más información sobre las características de tráfico de la red de lo que es posible representar en un diagrama.

El diagrama cumple su función de “foto” de lo más importante del estado actual mostrándonos por ejemplo el nombre más representativo o el protocolo más usado. De esta manera se ignora una gran cantidad de información que se ha ido recopilando y para la que hay que buscar algún modo de presentación.

Es este un trabajo que aún no está terminado. Aún queda información registrada a la que no es posible acceder, pero con lo que ya está implementado se sientan las bases para una sistema genérico de representación de información.

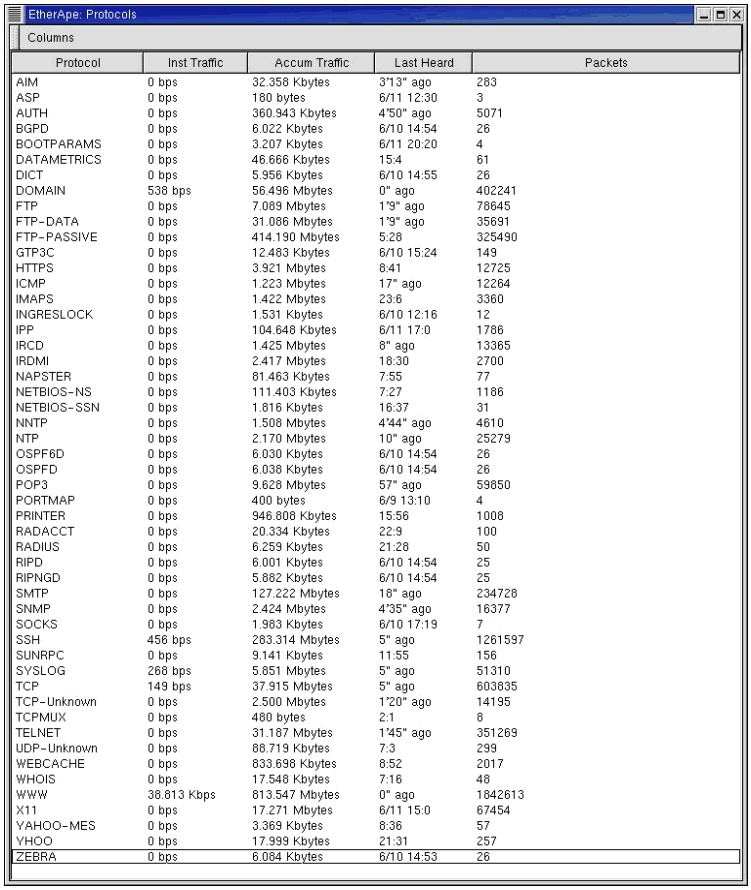
2.1 Ventana de protocolos

El primer resultado de este esfuerzo por ampliar el conjunto de información que presenta el programa es la ventana de protocolos. Aquí se presenta la informacion que no es específica de ningún nodo, sino que engloba todo el tráfico que se ha escuchado en la red.

Para cada paquete que escucha, el motor de captura analiza su pila de protocolos, y luego añade la información pertinente de ese paquete a lista global de protocolos que se va guardando.

Analizando esta información es posible presentar la tabla que se muestra en la ventana de protocolos, donde se incluyen los siguientes campos:

- *Nombre*. Es posible agregar todos los paquetes que utilizan un puerto TCP o UDP no identificado bajo sendos nombres genéricos TCP-Unknown y UDP-Unknown. La segunda opción es identificarlo con el número de puerto, e.g. TCP-2454.
- *Tráfico instantáneo*. Se calcula haciendo media a lo largo de un periodo definido por el "Tiempo de medida" (seleccionable por el usuario).
- *Tráfico acumulado*. Cantidad de tráfico acumulada desde que se inició la medida. Se resetea cuando se para una captura.
- *Última vez*. Índica en qué momento fue escuchado ese protocolo por última vez. Hasta 10 minutos da un valor relativo (hace 2'35"), luego un valor absoluto. Cuando han pasado más de 24 horas indica también el día.
- *Paquetes*. Proporciona una métrica diferente del impacto de un protocolo en la red. En muchas ocasiones un protocolo que esté generando una tormenta de broadcast no está mandando gran cantidad de tráfico, sino gran cantidad de paquetes que pueden ser pequeños. El que los paquetes sen más o menos grandes no afecta al hecho de que todas las estaciones están procesando todos los paquetes que se hayan recibido, y ese es el problema.



Protocol	Inst Traffic	Accum Traffic	Last Heard	Packets
AIM	0 bps	32.358 Kbytes	3"13" ago	283
ASP	0 bps	180 bytes	6/11 12:30	3
AUTH	0 bps	360.943 Kbytes	4"50" ago	5071
BGPD	0 bps	6.022 Kbytes	6/10 14:54	26
BOOTPARAMS	0 bps	3.207 Kbytes	6/11 20:20	4
DATAMETRICS	0 bps	46.666 Kbytes	15:4	61
DICT	0 bps	5.956 Kbytes	6/10 14:55	26
DOMAIN	536 bps	56.496 Mbytes	0" ago	402241
FTP	0 bps	7.089 Mbytes	1"9" ago	78645
FTP-DATA	0 bps	31.086 Mbytes	1"9" ago	35691
FTP-PASSIVE	0 bps	414.190 Mbytes	5:28	325490
GTP3C	0 bps	12.483 Kbytes	6/10 15:24	149
HTTPS	0 bps	3.921 Mbytes	8:41	12725
ICMP	0 bps	1.223 Mbytes	17" ago	12264
IMAPS	0 bps	1.422 Mbytes	23:6	3360
INGRESLOCK	0 bps	1.531 Kbytes	6/10 12:16	12
IPP	0 bps	104.648 Kbytes	6/11 17:0	1786
IRCD	0 bps	1.425 Mbytes	8" ago	13365
IRDMI	0 bps	2.417 Mbytes	18:30	2700
NAPSTER	0 bps	81.463 Kbytes	7:55	77
NETBIOS-NS	0 bps	111.403 Kbytes	7:27	1186
NETBIOS-SSN	0 bps	1.816 Kbytes	16:37	31
NNTP	0 bps	1.508 Mbytes	4"44" ago	4610
NTP	0 bps	2.170 Mbytes	10" ago	25279
OSPF6D	0 bps	6.030 Kbytes	6/10 14:54	26
OSPFD	0 bps	6.038 Kbytes	6/10 14:54	26
POP3	0 bps	9.628 Mbytes	57" ago	59850
PORTMAP	0 bps	400 bytes	6/9 13:10	4
PRINTER	0 bps	946.808 Kbytes	15:56	1008
RADACCT	0 bps	20.334 Kbytes	22:9	100
RADIUS	0 bps	6.259 Kbytes	21:28	50
RIPD	0 bps	6.001 Kbytes	6/10 14:54	25
RIPNGD	0 bps	5.882 Kbytes	6/10 14:54	25
SMTP	0 bps	127.222 Mbytes	18" ago	234728
SNMP	0 bps	2.424 Mbytes	4"35" ago	16377
SOCKS	0 bps	1.983 Kbytes	6/10 17:19	7
SSH	456 bps	283.314 Mbytes	5" ago	1261597
SUNRPC	0 bps	9.141 Kbytes	11:55	156
SYSLOG	268 bps	5.851 Mbytes	5" ago	51310
TCP	149 bps	37.915 Mbytes	5" ago	603835
TCP-Unknown	0 bps	2.500 Mbytes	1"20" ago	14195
TCPMUX	0 bps	480 bytes	2:1	8
TELNET	0 bps	31.187 Mbytes	1"45" ago	351269
UDP-Unknown	0 bps	88.719 Kbytes	7:3	299
WEBCACHE	0 bps	833.698 Kbytes	8:52	2017
WHOIS	0 bps	17.548 Kbytes	7:16	48
WWW	38.813 Kbps	813.547 Mbytes	0" ago	1842613
X11	0 bps	17.271 Mbytes	6/11 15:0	67454
YAHOO-MES	0 bps	3.369 Kbytes	8:36	57
YHOO	0 bps	17.999 Kbytes	21:31	257
ZEBRA	0 bps	6.084 Kbytes	6/10 14:53	26

Figura 13 – Ventana de protocolos

La tabla se implementa usando el componete GTK-Clist (de lista en columnas). Una ventaja de este componente es que permite ordenar la tabla de muchas maneras diferentes, si se proporciona en cada caso una función que compare los datos de dos filas arbitrarias.

De este modo se han programado 5 funciones de comparación diferentes, de modo que haciendo click en la cabecera de cada columna la tabla queda ordenada para ese campo en particular.

2.2 Ventana de información de nodo

Si bien es posible pedirle a EtherApe que utilice seis medidas distintas para calcular el radio del nodo, en ningún caso es posible ver toda esa información a la vez.

Por otro lado en el diagrama siempre se muestra el nombre resuelto, si es que esta disponible, y hay casos en los que resulta útil saber ese nombre no resuelto.

Para mostrar esta información debe hacerse doble click sobre un nodo en diagrama. El código de gestión de eventos de diagrama reconoce este hecho y abre la ventana correspodiente.

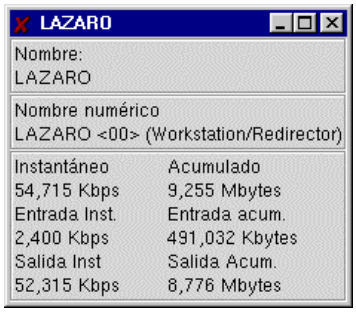


Figura 14 – Ventana de información de nodo

EtherApe sabe en realidad mucho más sobre un nodo de lo que aparece en esta ventana sencilla. En sucesivas versiones debería ser posible desplegar su árbol completo de protocolos conocidos, y mostrar asimismo todos los nombres que ha usado.

En estos momentos es posible tener acceso a partede esa información, usando los mecanismos de depuracion a través de la salida estándar de la consola. A continuación se presenta un ejemplo de sesión. La salida a partir de la línea “NODE LAZARO INFORMATION” es la información buscada, que sale como resultado de llamar la ventana de información de ese nodo desde el interfaz gráfico.

```
lazaro:~# export DEBUG=INFO
lazaro:~# /usr/bin/etherape
INFO: Dispositivos de captura disponibles: eth0 eth1 lo
INFO: Live device eth0 opened for capture. pcap_fd: 5
INFO: Link type is Ethernet
INFO: Diagrama iniciado
INFO: Reading TCP and UDP services from /etc/services
INFO: DDP protocols not supported in rtmp  1/ddp    # Routing Table Maintenance
Protocol
INFO: DDP protocols not supported in nbp  2/ddp    # Name Binding Protocol
INFO: DDP protocols not supported in echo  4/ddp    # AppleTalk Echo Protocol
INFO: DDP protocols not supported in zip  6/ddp    # Zone Information Protocol
INFO: Nodes: 23. Canvas nodes: 15
INFO: NODE LAZARO INFORMATION
INFO: Protocol level 1 information
INFO:   Protocol ETH_II
INFO:           Name: LAZARO-RJ45
INFO: Protocol level 2 information
INFO:   Protocol IP
INFO:           Name: cliente-213227048220.cm128.senpb.supercable.es
INFO: Protocol level 3 information
INFO:   Protocol TCP
INFO:           Name: cliente-213227048220.cm128.senpb.supercable.es:1372
cliente-213227048220.cm128.senpb.supercable.es:1371
cliente-213227048220.cm128.senpb.supercable.es:1370
cliente-213227048220.cm128.senpb.supercable.es:1369
cliente-213227048220.cm128.senpb.supercable.es:1368
cliente-213227048220.cm128.senpb.supercable.es:netbios-ssn
cliente-213227048220.cm128.senpb.supercable.es:1367
cliente-213227048220.cm128.senpb.supercable.es:1365
cliente-213227048220.cm128.senpb.supercable.es:1366
cliente-213227048220.cm128.senpb.supercable.es:ssh
INFO:   Protocol UDP
INFO: Protocol level 4 information
INFO:   Protocol X11
INFO:   Protocol RPC
INFO:   Protocol SSH
INFO:   Protocol UNKNOWN
INFO:   Protocol NETBIOS-SSN
INFO:           Name: LAZARO
INFO:   Protocol NTP
INFO:   Protocol DOMAIN
INFO:   Protocol POP3
INFO: Protocol level 5 information
INFO:   Protocol UNKNOWN
INFO:   Protocol X11
INFO:   Protocol SMB
INFO:   Protocol UDP-Unknown
INFO:   Protocol NTP
```

Figura 15 – Información de nodo en consola

Puede observarse cómo para cada nivel de la pila de protocolos se muestran los protocolos utilizados y cada uno de los nombres utilizados a este nivel (Ethernet: LAZARO-RJ45, IP: cliente-213227048220.cm128.senpb.supercable.es, NetBIOS: LAZARO, etc.) .

La versión final que se haga en la interfaz gráfica deberá también incluir la misma información presente en la ventana de protocolos.

2.3 Ventana de información de protocolo

A partir de la ventana de protocolos es posible presentar una ventana específica para un único protocolo.

En estos momentos su utilidad es limitada, puesto que no da más información que lo que aparece en la misma ventana de protocolos. En todo caso su valor reside pemitir al usuario ceñirse al protocolo en el que está interesado liberar espacio de pantalla para otros usos.

Figura 16 – Ventana de información de protocolo

El espacio de la parte inferior de la ventana está reservado para dar una información detallada sobre todos los nodos que han utilizado ese protocolo, de manera simétrica a lo que deberá ocurrir con la ventana de información de nodo.

3 Elementos activos de la interfaz de usuario

Son elementos activos a aquellos que están presentes para permitir que usuario. Si bien tanto el diagrama como la ventana de protocolos cumplen esta definición puesto que hacer doble click hace que aparezcan nuevas ventanas, nos centraremos en aquellos que están expresamente pensados para cumplir esta función.

3.1 La barra de menús

Los sistemas de menús son el modo más común de interactuar con una aplicación gráfica.

Gnome define un estándar de cómo se han de distribuir los menús para hacer que para un usuario novato usar aplicaciones de este entorno se más intuitivo. Ejemplos de esto es el menú *Archivo*, que ha de estar el primero a la izquierda, o el menú *Ayuda*, que es el último por la derecha, pero no está alineado a la derecha.

Veamos con detalle las distintas posibilidades.

3.1.1 El menú Archivo

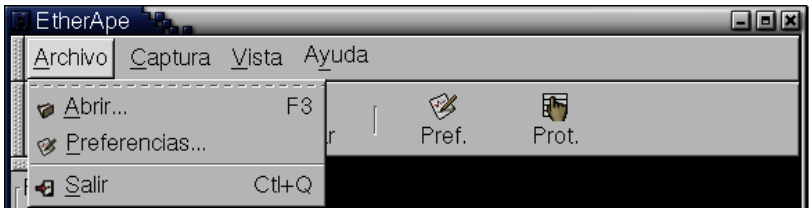


Figura 17 – El menú Archivo

- *Abrir*. Abre la ventana de selección de archivo para cargar un archivo de captura. Una vez seleccionado finaliza la captura actual y comienza leer del archivo.
- *Preferencias*. Abre el diálogo de preferencias.
- *Salir*. Finaliza la captura en curso y cierra la aplicación. Es importante asegurarse de que se han cerrado correctamente los procesos de libpcap, o se corre el riesgo de dejar la interfaz de captura en modo promíscuo.[2]

Por ser el menú más común, es el que más especificado está para las aplicaciones Gnome. El comando Salir debe estar aquí presente, y si existe el comando Abrir, también.

De hecho los tres comandos de este menú son Stock Items, pertenecen al estándar Gnome. Entre otras cosas, esto hace que los comandos estén automáticamente traducidos a cualquier idioma.

3.1.2 El menú Captura

Desde aquí se controlan los parámetros principales de la captura.

[2] Es importante asegurarse de que no se va a dejar la interfaz de red en modo promíscuo, por ello la aplicación es capaz de interceptar la señal de interrupción (que se produce cuando el usuario hace Control-C desde la consola, por ejemplo) para asegurarse de que libpcap tiene la oportunidad de cerrar de manera limpia.

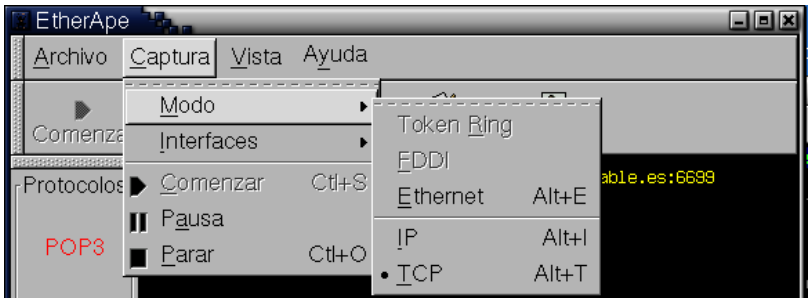


Figura 18 – Submenú de modo

- *Submenú de modo*. Permite seleccionar cualquiera de los modos de catptura disponibles en la interfaz de captura que esté activa en ese momento. Cuando se cambia de modo se reinicializa el estado del programa, puesto que la definición de los nodos cambia y los datos ya no pueden seguir agregándose.

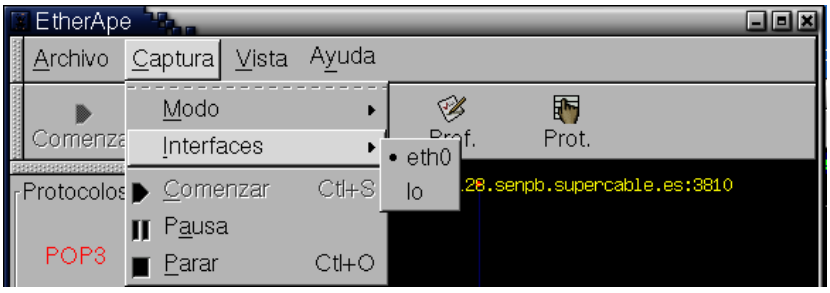


Figura 19 – Submenú de interfaces

- *Submenú de interfaces*. Mediante el uso de funciones de servicio de libpcap, EtherApe detecta todos los interfaces del sistema con los cuales es posible hacer una captura en vivo. Cambiar la interfaz también finaliza la captura anterior y resetea las estadísticas.
- *Comenzar*. Empezar a leer datos de la interfaz de lectura o de un archivo de captura.
- *Pausa*. Cuando se está leyendo de una interfaz en vivo, este comando congela el diagrama, pero se siguen procesando los paquetes que llegan de modo que cuando se siga adelante mediante el comando Comenzar los datos estadísticos se actualicen y sigan siendo válidos.

  Cuando se lee de un archivo de captura, se congelan tanto el diagrama como la captura. Al final del archivo de captura el programa se pone en pausa automáticamente para permitir analizar las ventanas de estadísticas.

- *Parar*. Finaliza la captura actual y libera la interfaz de red, si se estaba usando. Después de parar el programa mantiene su estado: la interfaz de captura o el fichero de lectura, y el modo que se esté utilizando.

3.1.3 El menú Vista

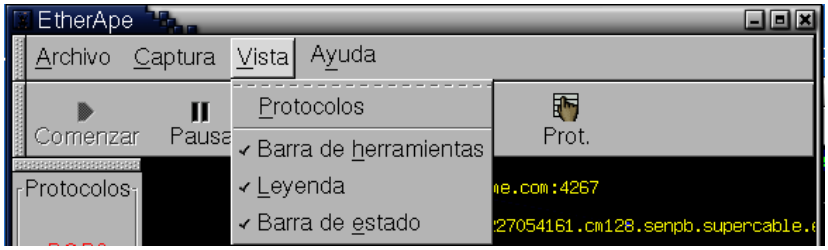


Figura 20 – Menú Vista

- *Protocolos*. Abre la ventana de protocolos
- *Barra de herramientas*. Permite seleccionar si la barra de herramientas está o no presente, de modo que se pueda hacer más hueco para el diagrama propiamente dicho.
- *Leyenda*. Ditto, para la tabla de la leyenda.
- *Barra de estado*. Ditto, para la barra de estado.

3.1.4 El menú de Ayuda

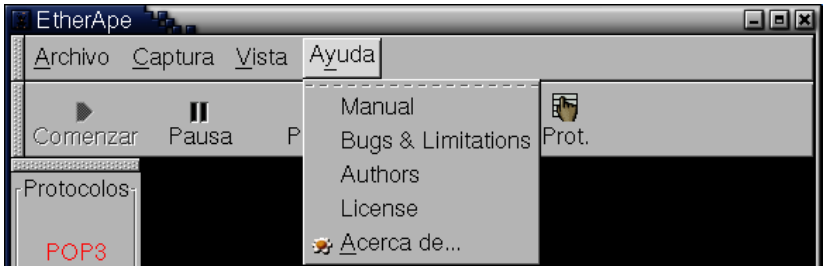


Figura 21 – Menú de Ayuda

El estándar Gnome especifican que todas las aplicaciones deben incluir un menú de ayuda, que incluya como mínimo enlaces a la ayuda en línea y el comando Acerca de...

- *Páginas de ayuda*. Estos comandos de menú no se crean de la misma manera de los demás, sino que las rutinas de inicialización de Gnome las añaden automáticamente a partir del fichero

/usr/share/gnome/help/etherape/C/topic.dat, que se distribuye junto con la aplicación.

De esta manera se facilita el que los encargados de la documentación puedan trabajar independientemente de los programadores.

- *Acerca de...* Este comando está igualmente estandarizado por Gnome, así como el cuadro diálogo que se presenta al activarlo.

3.2 El diálogo de preferencias

EtherApe permite configurar muchos de los parámetros que definen su funcionamiento. El diálogo de preferencias se divide en dos hojas, una de las cuales controla de qué manera se hace la captura, mientras que la otra configura la presentación de los datos en pantalla.

3.2.1 Parámetros del diagrama

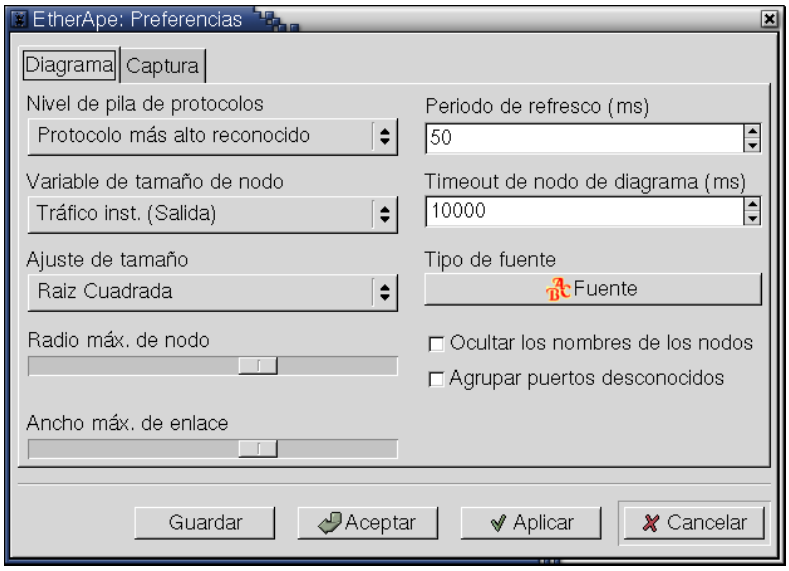


Figura 22 – Diálogo de preferencias del diagrama

- *Nivel de pila de protocolos*. Define el conjunto de protocolos que se distinguirán en pantalla. Los posibles valores son niveles fijos de la pila (nivel 2, nivel 3, nivel 4, etc.) o bien presentar siempre el protocolo de nivel más alto reconocido.

  Hay que tener en cuenta que la selección de niveles fijos no se ajusta exactamente con la definición de niveles en un sistema OSI, simplemente con el orden en que se van encontrado para cada medio físico. Para una red Ethernet sí produce el valor esperado, pero para FDDI, por ejemplo, el nivel

2 es FDDI, pero el 3 puede ser LLC, mientras que IP va en este caso en el cuarto nivel.

Cuando se usa el ajuste de "Nivel más alto reconocido", se mezclan protocolos de varios niveles, pero suele resultar ser lo más útil, y es el valor por defecto.

- *Variable de tamaño de nodo*. El radio del nodo puede determinarse a partir de seis variables diferentes. Suma del tráfico de entrada y salida, tráfico de entrada y tráfico de salida, tomados como valores instantáneos o como acumulados.

  Para los valores instantáneos se utiliza la variable "Tiempo de medida" que se configura en las preferencias de capturas. El acumulado es a partir de la última vez que se reseteó la captura.

  Variando este parámetro pueden analizarse hechos interesantes, tales como averiguar quién ha consumido más ancho ancho de banda, o quién ha servido más tráfico, o bien quién está monopolizando la red en este momento. El valor por defecto es "tráfico instantáneo de salida"

- *Ajuste de tamaño*. Por su naturaleza, las comunicaciones de red suelen funcionar por ráfagas. Esto es, un nodo puede permanecer largo tiempo callado y repentinamente comenzar a enviar información. Por otro lado, diferentes protocolos manejan cantidades muy dispares de tráfico. Por lo general protocolos que usen puentes o routers para su configuración automática utilizan paquetes de menos de 200 bytes, mientras que típicamente un protocolo para compartir archivos en una red local intentará copar el ancho de banda máximo.

  Así pues, si el ancho de los nodos se relaciona de manera lineal con el tráfico, para un ajuste que sirva para comparar protocolos que usen poco ancho de banda provocará que cuando aparezca tráfico de otro tipo la pantalla se rellene completamente por el tamaño que han alcanzado el radio de los nodos implicads. Si se ajusta a la inversa, los protocolos de menor impacto se hacen prácticamente invisibles.

  El modo de mantener una visión de conjunto es utilizar una función de ajuste, que se controla con este parámetro. Es posible utilizar una función linea, logarítmica o de raiz cuadrada. Este ajuste afecta tanto al cálculo del radio de los nodos como al del ancho de los enlaces.

- *Radio de nodo / Ancho de enlace*. Una vez decidida la función de ajuste, este parámetro permite aplicar una corrección multiplicativa al valor

calculado. De este modo tenemos aún más control sobre cómo queremos que distintos niveles de tráfico aparezcan en pantalla.

Este ajuste es necesario puesto que no es lo mismo analizar el tráfico de un enlace punto a punto sobre un módem que el de una línea dedicada T3.

- *Periodo de refresco*. Es el tiempo que pasa entre sucesivas actualizaciones del diagrama. Permite definir al usuario el nivel de equilibrio deseadoentre validez del diagrama y uso de la CPU.

  Hay que tener en cuenta que en cada refresco no sólo se redibuja el diagrama, sino que es necesario procesar las estructuras de datos de captura para asegurar que vamos a utilizar datos válidos. Por ejemplo los paquetes o nodos demasiado antigos se descartan en este momento, puesto que hacerlo de manera contínua supone un esfuerzo de procesado innecesario.

  También aquí hay que permitir flexibilidad, puesto que el análisis de una red que transporte mucho más tráfico requerirá de mucho más tiempo de procesado y el periodo de refresco se habrá de ajustar al alza en consecuencia.

- *Expiración de nodo de diagrama*. Se utiliza para mantener en pantalla sólo los nodos que esté activos (hayan intervenido en tráfico de algún tiempo) dentro del lapso definido.

  Hay que distinguir entre tiempo tiempo de expiración de nodo de diagrama y el de captura, siendo el primero siempre menor que el segundo. Cuando un nodo expira del diagrama únicamente desaparece allí (se elimina el canvas_node correspondiente), pero el motor de captura sigue guardando la información relativa a este nodo.

  Esto se hace así por dos motivos. El más importante es el de guardar la estadística de este nodo durante más tiempo, de modo que cuando vuelva a estar activo esa información se agregue a la que ya está recogida. Así, por ejemplo, si se hubiera eliminado completamente, a la siguiente aparición el proceso de recogida de nombres tendría que comenzar desde el principio.

  Por otro lado cabe la posibilidad de que el usuario mantenga un interés especial por un nodo y mantenga abierta su ventana de información de nodo. Aunque el nodo se quede inactivo por un tiempo es necesario refrescar la información de la ventana, y eso se hace a partir de los datos guardados en el nodo de captura.

- *Tipo de fuente*. Define la fuente a utilizar para superponer el nombre de los nodos en el diagrama.

- *Ocultar los nombres de los nodos*. En algunos casos puede ser innecesario y hasta molesto identificar cada uno de los nodos del diagrama para observar el comportamiento global de la red. Esta opción permite eliminar los nombres.

  Por otro lado no se puede negar el valor lúdico de la presentación en pantalla de EtherApe, y eliminar los nombres contribuye a crear un diagrama más actractivo para los usuarios que sólo quieran entrenerse mirando el monitor.

- *Agrupar puertos desconocidos*. Si bien esta opción afecta de manera importante a la presentación, en realidad define cómo se realiza la captura, y en versiones sucesivas se trasladará al panel correspondiente.

  Mediante este parámetro se puede decidir si paquetes dirigidos a puertos TCP o UDP no reconocidos se agrupan dentro de protocolos genéricos TCP-Unknown y UDP-Unkown, o bien se crea un protocolo nuevo por cada puerto TCP-Port 3364, o UDP-Port 6734.

  Mientras que en el primer caso se está perdiendo información debido al agrupamiento, si el programa está analizando una red con protocolos que abren puertos arbitrarios y cuyo funcionamiento no está implementado, el número de protocolos reconocidos podría crecer sin control hasta hacer el digrama bastante difícil de reconocer.

  En cualquier caso es objetivo del programa la eliminación de puertos no reconocidos. Su presencia se considera como un caso todavía no implementado.

3.2.2 Parámetros de captura

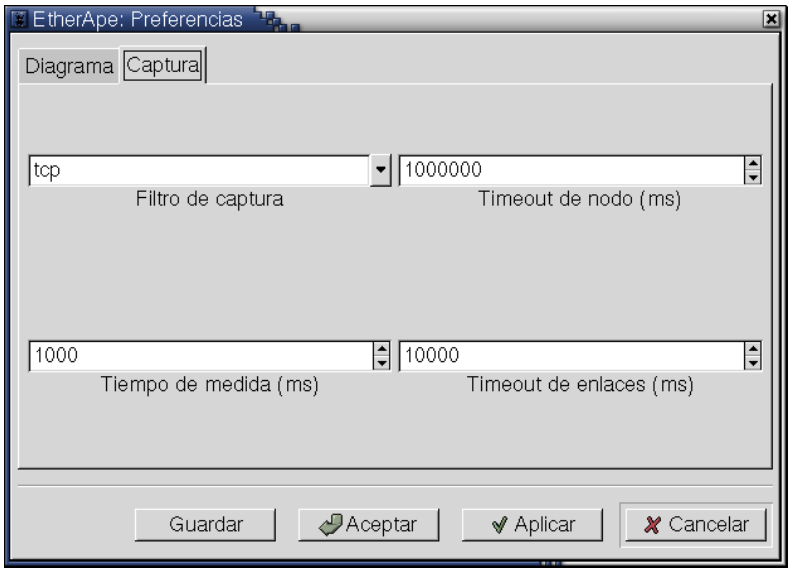


Figura 23 – Diálogo de preferencias de captura

- *Filtro de captura*. Permite limitar el tipo de tráfico que EtherApe va a procesar. Esta funcionalidad la proporciona la biblioteca libpcap, y por tanto la sintaxis a utilizar es la que esta exige.

  Hay que recordar que cuando se trabaja en un modo específico se debe asegurar que sólo se trata de procesar nodos cuyas identidades se admitan en ese modo. Para conseguirlo el programa actualiza automáticamente el filtro de captura utilizado para ajustarse al modo actual.

- *Expiración de nodo de captura*. Toda la información recogida sobre los nodos que lleven todo este tiempo inactivos será deshechada.

  Si al explicar el tiemo de expiración de nodo de diagrama justificábamos la necesidad de recordar esta información, es igualmente importante olvidarla cuando sea necesario.

  EtherApe está concebido como una herramienta de primera necesidad, y como tal no es raro encontrar administradores de sistemas que la dejan corriendo en una consola ininterrumpidamente durante semanas. Durante el funcionamiento normal de una red habrá nodos que estarán activos casi la totalidad del tiempo (un servidor local, el encaminador), y otros que podrían aparecer una sola vez y no volver a hacerlo más (el servidor de una página web que se visita ocasionalmente). Si el programa no deshechara ningún tipo de información sus requerimientos de memoria crecerían hasta agotar los recursos del ordenador donde está corriendo.

Así pues este parámetro permite que el programa siga corriendo indefinidamente, ajustado el equilibrio entre información almacenada y las necesidades de memoria.

- *Tiempo de medida*. Lapso de tiempo de utiliza el programa para calcular todos los valores instantáneos que se vayan a presentar.

  Puesto que este valor es ajustable, el usuario puede decidir si está más interesado en calcular una media a largo plazo o en observar el comportamiento más aproximado al instantáneo.

- *Expiración de enlaces*. Juega el papel que el tiempo de expiración de nodos, salvo que para los enlaces.

  Sin embargo, puesto que de los enlaces no se guarda información que pueda tener un interés adicional en el futuro es innecesario contar con el equivalente de expiración de enlaces de diagrama.

3.2.3 Aplicabilidad y permanencia de los parámetros

Una característica imprescindible para hacer que un programa sea amigable a los ojos de los usuarios es dotarles de cierta inteligencia. Eso significa por ejemplo el que sean capaz de recordar automáticamente valores anteriores del filtro de captura, o qué archivos de captura se han abierto recientemente.

Igualmente es importante permitir al usuario almacenar sus preferencias para recuperarlas en la siguiente sesión. Gnome aporta funciones de biblioteca que facilitan este trabajo, manteniendo automáticamente el archivo donde se almacenan.

Utilizando el botón de Guardar puede hacer que los cambios hechos en el diálogo de preferencias durante la sesión actual se almacenen para la siguiente.

Hay que notar que modificar los parámetros no funciona igual para todos. Accionar algunos de ellos implica un cambio inmediato en el programa (como por ejemplo cambiar el factor de multiplicación de radio de nodo), mientras que para hacer efectivos otros es necesario pulsar el botón Aplicar (por ejemplo, para ver el cambio de tipo de fuente en los nombres)

El botón Aceptar cierra el diálogo de preferencias aplicando los cambios automáticamente, mientras que Cancelar también cierra el diálogo, pero sin aplicar los cambios.

Procedimiento de desarrollo

Una parte importante de la elaboración de EtherApe ha sido el esfuerzo dedicado a que el programa siga los estándares de desarrollo comúnmente utilizados por la comunidad del software libre, en lugar de imponer un estilo propio. A pesar de que esto requiere un esfuerzo inicial nada despreciable, esta labor no tarda en dar sus frutos.

1 Introducción

El éxito de un proyecto depende de muchos factores. No basta con contar con un desarrollador de cierta calidad técnica. Es necesario mirar más allá para poder decidir cuáles son las herramientas adecuadas para realizar el trabajo, por ejemplo.

Pero cuando se trata de software de libre distribución hay que tener en cuenta que para lograr que se materialicen algunas de sus ventajas además hay que seguir sus propias reglas. Pongamos un ejemplo: Si pretendes que otros usuarios que utilizan una plataforma diferente a la tuya comprueben si tu código funciona en su entorno, debes al menos asegurarte de una lista de cosas:

- Que el usuario pueda conocer de la existencia de tu programa
- Que le sea posible obtener una copia con facilidad
- Que el programa haga al menos un esfuerzo razonable de funcionar en sistemas no probados
- Que el usuario disponga de medios eficaces para comunicarse con el programador e informar de su experiencia.

Asi pues, es importante recordar que tan importante como crear el programa es asegurar un procedimiento de desarrollo que le permita integrarse con la comunidad del software libre y sacar partido de ella.

2 Estándar GNU de programación

A lo largo de los años en que se ha ido distribuyendo software libre, el formato en que cada paquete se distribuye se ha ido formalizando hasta asentarse en un estándar.

Este estándar describe un conjunto de procedimientos que si son implementados permiten a un usuario o programador que acaba de acceder al programa conocer a priori de qué manera trabajar con él.

Algunos aspectos que pueden parecer triviales como la nomenclatura del paquete de distribución, la estructura de directorios en que se descomprime, y algunos archivos de presencia indispensable; realmente contribuyen a hacer más fácil el trabajo a las personas que no han intervenido directamente en el desarrollo del programa, puesto que se ven ante un esquema conocido.

Desde el punto de vista del usuario, si tiene delante un paquete que sigue el estándar GNU, sabe que con un alto grado de probabilidad todo va a ir bien con más que ejecutar una serie aprendida de comandos. En el caso de etherape, sería

```
lazaro$ tar xvfz etherape-0.7.8.tar.gz
lazaro$ cd etherape-0.7.8
lazaro$ ./configure
lazaro$ make install
```

Primer se descomprime el paquete. Luego se ejecuta el script llamado configure, que analiza el entorno de compilación actual, y por último se compila el paquete completo y se procede a la instalación.

Cualquiera que haya tratado de compilar un programa que tenga un cierto número de dependencias de software y que no siga este esquema sabrá valorar en su justa medida su simplicidad y su potencia.

2.1 Autoconf

En este sentido Autoconf es una herramienta indispensable. Su objetivo es facilitar la compilación de un mismo programa en plataformas diferentes, así como normalizar el proceso de ajuste de la compilación en esas arquitecturas.

El resultado final de utilizar esta herramienta será el script configure, que se encargará de realizar esas tareas.

Para producir este script el programador dispone de un lenguaje específico con el que puede definir los requerimientos de su programa. Se puede, por ejemplo, requerir que se diponga de la bibliotecas de resolución de nombres, o de ciertas ficheros de cabecera estándar.

Autoconf generará un script que será capaz de descubrir en qué tipo de plataforma está corriendo y adaptarse, de forma que tenga mayores probabilidades de averiguar si se cumplen los requerimientos, y de ser así, adaptar el proceso de compilación a las especificidades de la plataforma.

Por otro lado, es común que paquetes de bibliotecas que el programador vaya a utilizar, pongan a su disposición macros para ser utilizadas junto con autoconf, de modo que automáticamente se pueda verificar si están instaladas en esa plataforma.

2.2 Automake

La manera más habitual de guiar el proceso de compilación de un programa es utilizando un Makefile, un archivo que define los objetivos a construir y las dependencias que deben cumplirse antes de poder se construidos.

Automake es una herramienta que permite facilitar enormemente el proceso de generación de estos archivos, a la vez que se integra con autoconf para mejorar el resultado conjunto.

Así, cuando se usa esta herramienta el programador sólo debe indicar, por ejemplo, cuáles son los archivos que contienen el código fuente, sin ni siquiera preocuparse por las bibliotecas de las que depende el programa.

El script configure se encargará de descubrir esas dependencias, y, con ayuda de la información generada por automake, construir un archivo Makefile específico para la plataforma en que se esté ejecutando.

Además, al utilizar automake el archivo Makefile generado resulta ser muy funcional, integrando automáticamente multitud de objetivos que son a la vez muy útiles y esperados por los usuarios.

Así, por ejemplo, el usuario sabe que *make install* compilará el ejecutable y copiará todos los archivos necesarios en el sistema. Pero también dispone de *make uninstall*, que automáticamente borrará los archivos que se hubieran copiado, en caso de que quisiera deshacerse del programa, o *make clean*, para borrar los archivos generados en la compilacón.

El programador también saca partido, al disponer de objetivos que le són tan útiles como *make dist*, que genera automáticamente un archivo de distribución que sigue todas las normas GNU.

3 Traducciones: Gettext

Cuando se distribuye un programa de software libre a través de internet, el mercado potencial de usuarios no está restringido a ningún área geográfica.

El software puede serle de utilidad a cualquiera, y es bueno asegurar el mayor número de usuarios posibles para contribuir a que los mecanismos beneficiosos del software libre hagan efecto.

En este sentido hay que tener en cuenta que no todo el mundo habla el idioma que el programador ha elegido para su programa (que no tiene por qué ser el suyo propio), y por tanto es de interés encontrar un mecanismo que permita de manera sencilla la traducción de los mensajes al usuario.

Este mecanismo se llama gettext. Gettext cumple dos funciones: por un lado es el código que ajusta los mensajes a un idioma específico en tiempo de ejecución, y por otro son herramientas y esquemas de trabajo para facilitar la traducción de los programas.

Así, los programas se distribuyen junto con los catálogos de mensajes en diferentes idiomas, y se compilan junto con el código que los utiliza si es que la plataforma de destino no incorpora ya la biblioteca que cumple esa función.

El trabajo de la traducción se divide en dos frentes separados. Por un lado el programador marca cuáles son las cadenas que se presentan al usuario y que por tanto será necesario traducir. El marcaje supone un trabajo mínimo, puesto solo que consiste en pasar las cadenas por la función de biblioteca _().

Por otro lado los traductores ejecutan una serie de programas que generan archivos de traducción para cada idioma que deberán editar. A partir de estos archivos de texto que ya están traducidos se generarán archivos binarios con los catálogos de mensajes.

Gettext no es el único sistema que existe para la traducción de programas, pero es el estándar de facto para los programas de software libre. Esto implica que al implementar este sistema dentro de EtherApe, cualquier persona con experiencia puede contribuir una traducción en muy poco tiempo.

Así ha ocurrido de hecho, y es gracias a la colaboración desinteresada de dos usuarios que EtherApe está disponible en francés y holandés, además de las versiones en inglés y español aportadas por el autor.

4 Gestión de codigo: CVS

Cuando se desarrolla un programa de cierta entidad es necesario utilizar una herramienta que permita mantener un historial de su evolución. Esta es la función principal de CVS (siglas Concurrent Versioning System).

Utilizando CVS es posible seguir los cambios que se van introduciendo, y deshacerlos también, si es necesario. También permite marcar un momento en el tiempo del código, de modo que sea posible recuperar el proyecto completo tal y como estaba en un momento particular, o cuando se distribuyó una versión en particular.

Cuando se usa a fondo, es posible construir un árbol de desarrollo con varias ramas paralelas. Esto permite, por ejemplo, iniciar un cambio importante en el código sin perder la estabilidad de la versión anterior. En esta situación aún es posible incorporar correciones de errores en la versión estable, y más adelante integrar la versión inestable con el tronco principal.

Pero además CVS cumple una función fundamental en el esquema de desarrollo de software libre. CVS es un sistema concurrente que soporta autenticación. De esta manera es posible mantener un equipo de personas trabajando sobre la misma base de código, sin necesidad de estar haciendo llegar parches a una persona encargada específicamente del mantenimiento de la versión oficial.

Esto tiene un enorme valor para un proyecto de software libre. Cualquier herramienta que facilite el trabajo a contribuyentes de código en perspectiva es una ventaja que merece la pena poner en marcha.

En el caso de EtherApe, por ejemplo, los traductores tienen acceso directo a los archivos que le corresponden, y la persona encargada de preparar el paquete debian también edita directamente los archivos de configuración específicos.

Cuando llega el momento de sacar una nueva distribución, todos los cambios están ya incorporados, y el proceso se simplifica.

Si además se permite acceder al código en el servidor de CVS de forma anónima, se da la oportunidad a los usuarios que así lo deseen de probar lo antes posible las últimas modificaciones.

5 Promoción del código

Uno de los aspectos más importantes del software libre es asegurar el máximo de usuarios posibles.

Cuantos más usuarios haya más rápidamente se localizarán los errores. Cuanto antes se corrijan los errores mejor reputación tendrá el programa y más personas lo usarán. A mayor número de usuarios, mayor el número de ellos con conocimientos y el interés suficientes para mejorar el programa. Y así sigue el ciclo.

Es un ciclo de realimentación positiva, pero los beneficios son mayores haciendo un esfuerzo por hacer que la maquinaria funcione suavemente.

El primer paso es disponer de un sitio en el WWW para distribuir el programa e incluir toda la información posible para ayudar a los usuarios o a las personas que puedan contribuir de alguna manera al desarrollo del código.

En este punto pretendemos "vender" el producto, y así es como se debe trabajar. Cuanto más profesional sea la página web, mejores expectativas tendrán los posibles usuarios y más probable será que se lancen a probarlo.

De cualquier manera, no basta con poner una página web. Para que el público dirija su navegador a un sitio en concreto primero debe conocer de su existencia, así que también hay que hacer publicidad.

La manera más efectiva de hacer publicidad es hacer saber las páginas que se encargan de seguir todo el software disponible de la existencia de nuestro programa. El sitio por excelencia es Freshmeat (www.freshmeat.net), que a diario consultan muchos miles de personas para comprobar qué novedades hay.

Puesto que nuestra aplicación utiliza Gnome, también se ha utilizado su página web al efecto para promocionar EtherApe.

"Release early, release often", es una máxima del software libre. Cuanto más se avance el programa y más versiones intermedias se publiquen, mayor será su exposición, y el número de personas que habrán tenido oportunidad de hablar de él.

Figura 24 – Página web de EtherApe

6 Sourceforge

Algunos de los procedimientos de los que se han hablado requieren una cierta cantidad de recursos. Un sitio web requiere contratar espacio en un servidor y pagar por el ancho de banda utilizado.

Encontrar una máquina accesible a internet donde poder instalar y administrar un servidor de CVS es todavía más difícil, y lo mismo ocurre con las listas de correo.

Afortunadamente la iniciativa empresarial hace tiempo que empezó a ver los beneficios que tiene para todo el mudo el software libre. Una de las empresas más grandes dedicadas a hacer negocios exclusivamente con Linux, VA, puso a disposición de los desarrolladores de software libre el material necesario para contrarestar la carestía de recursos. Este proyecto se llama Sourceforge, y a pesar de no contar aún ni con dos años de antigüedad se ha convertido en un pilar reconocido del fomento del software libre.

Sourceforge cede gratuitamente una extensa lista de recursos y servicios, entre los que se encuentran.

- Sitio web

  Espacio en disco duro y ancho de banda suficientes para cubrir las necesidades del proyecto más exitoso.

- Gestión de listas de correo

  No sólo proporcionan las máquinas que hacen que el sistema funcione. Además liberan al desarrollador de la carga de su mantenimiento, proveyéndole de un sencillo interfaz para su gestión.

- Servidor de CVS

  Al igual que ocurre con las listas de correo, mantener un servidor de CVS no es ninguna tarea trivial. Los técnicos de Sourceforge lo hacen posible para miles de proyectos diferentes. Además también permite hojear el código que se guarda en el servidor a través de un navegador de WWW.

- Acceso shell y granjas de compilación

  El desarrollador puede acceder directamente a una multitud de máquinas para satisfacer cualquier necesidad que tenga. Un grupo de sistemas heterogéneos está disponible para compilar código, ya sea porque la potencia de los sistemas personales de los desarrolladores no es suficiente, como para poder comprobar directamente la portabilidad de su código en las distintas plataformas.

- Sistemas de seguimientos de errores.

  El contacto con los usuarios es fundamental para poder sacar partido al primer hecho básico del software libre: la facilidad con que el software es probado en multitud de entornos y circunstancias diferentes.

  Sourceforge proporciona un sistema a través de sus páginas web para permitir hacer un seguimiento de los errores. El seguimiento es público, de manera que se disminuye la duplicidad en los informes y un mayor número de usuarios se beneficia de los consejos que pueda dar el desarrollador.

En definitiva Sourceforge es una herramienta fantástica, que hace posible aumentar la productividad del desarrollador liberándole de las tareas que podríamos llamar administrativas.

La página principal nos confirma que en el momento de escribir estas líneas EtherApe no es sino uno de los 23,501 proyectos registrados. No todos acabarán teniendo éxito, pero indudablemente lo tienen mucho más fácil gracias a este servicio.

Futuras líneas de trabajo

Como ocurre con cualquier proyecto de software libre, no es posible marcar el final de la vida de un programa. Cualquier persona que tenga interés puede seguir añadiéndole caracterísiticas o eliminado errores de programación.

Sin embargo el autor no ha abandonado todavía y hay varias líneas en las que todavía queda por hacer, algunas de las cuales ya se han comentado.

- Selección de colores

  Crear una interfaz gráfica adecuada para que el usuario pueda decidir cómo se asignan los colores a los diferentes programas. Las preferencias deben poder ser grabadas.

- Completar las ventanas de estadísticas

  Permitir analizar los nodos que usan un protocolo determinado, y al mismo tiempo ver qué protocolos usa un nodo específico.

- Fijar preferencias relativas a nodos individuales

  Cada nodo tendría su juego de preferencias, en principio heredadas de las globales para nodos nuevos, pero que luego podrían ser cambiadas por el usuario y guardadas para una sesión posterior.

  Se podría fijar entonces individualmente el tiempo de medida, por ejemplo.

- Selección manual de nombre preferente

  Permitir al usuario seleccionar qué tipo de nombre prefiere en pantalla para los nodos, a ser posible de manera individual para cada uno.

- Volcado a base de datos

  Cierto tipo de usuarios puede estar interesado en utilizar los datos recogidos por EtherApe en otros estudios. Debería ser posible volcar esa información, a ser posible con una interfaz genérica a sistemas de bases de datos.

- Convertir el motor de captura en un proyecto independiente

  No todos los usuarios quieren o pueden instalar Gnome en sus sistemas para utilizar EtherApe. Convirtiendo el motor de captura en una biblioteca que se

distribuyera independientemente se facilitaría mucho el desarrollo de versiones alternativas. Por ejemplo una versión que sólo dependa de GTK+, u otro que funcione en modo texto.

Todas estas mejoras irán llegando con el tiempo, lo que no se puede asegurar es el orden en que lo harán, puesto que en muchas ocasiones son los usuarios los que fijan las prioridades, y estan no tienen por qué coincidir con las del desarrollador.

Conclusión

El proyecto que comenzó como ejercicio para probar las virtudes del software libre ha acabado teniendo mucho más éxito del esperado.

Despues el punto de vista técnico EtherApe supera en muchos aspectos a Etherman, en gran medida gracias a la enorme ayuda que supone poder basarse en tanta cantidad de software de calidad.

Pero en el camino EtherApe ha cosechado un reconocimiento que no era esperado.

Tan sólo dos meses después de comenzar el proyecto el autor fue invitado a dar una de la sesiones en el congreso de Expo Linux 2000, celebrado en Madrid. A raiz de aquella presentación la Free Software Foundation decidión patrocinar el desarrollo financionado un nuevo equipo informático.

Durante semanas el programa estuvo entre los 10 más descargados de los miles de proyectos disponibles en Sourceforge. En los 427 días que el proyecto lleva en línea la página web ha tenido 379.885 visitas, y el programa ha sido descargado 62.757 veces.

Estas cifras no son indicativas de el número real de usuarios del programa, puesto que paulatinamente un mayor número de sitios web hacían copias del programa para ofrecerlo directamente a sus usuarios, tales como Tucows o Icewalkers.

Pero probablemente mucho más importante haya sido el efecto que ha tenido el que la mayoría de las compañías dedicadas a distribuir software para Linux en CD hayan incluido EtherApe entre los programas ofertados. Debian, RedHat, Mandrake, Conectiva y Linux/PPC son sólo algunas de ellas. El sistema operativo FreeBSD también da la opción de instalarlo directamente.

El autor ha tenido la oportunidad de escribirse directamente con más de 200 usuarios distintos. Algunos tan distinguidos como la NASA, que utilizó EtherApe para monitorizar el uso de red de equipos destinados a volar en la Estacion Espacial Internacional, o una compañía dedicada a diseñar sistemas de aviónica.

No cabe duda que mantener un proyecto que es utilizado 24 horas al día, 7 días a la semana en entornos de producción fuerza al desarrollador a cuidar los detalles y que el resultado sea bueno. Pero el hecho adicional de que el código está libremente disponible, y de que de hecho se pretende hacerlo legible para fomentar las contribuciones externas implica que no sólo se cuida el resultado, sino también la maquinaria.

El objetivo de reproducir el funcionamiento de Etherman no sólo se ha alcanzado. La licencia GPL bajo la que se distribuye EtherApe implica que el programa ya quedará abandonado, y que en el futuro sólo le cabe mejorar.

Bibliografía

- Radia Perlman

  Interconnections: Bridges and Routers. Addison-Wesley Publishing Company

- Havoc Pennington

  GTK+/Gnome Application Development. New Riders Publishing. http://developer.gnome.org/doc/GGAD/

- Eric S. Raymond

  The Cathedral & the Bazaar. O'Reilly. http://www.tuxedo.org/~esr/writings/cathedral-bazaar/

- Internet Assigned Numbers Authority

  Port numbers. http://www.iana.org/assignments/port-numbers

Apéndice A: Texto de la licencia GPL

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Preamble
The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION
0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program

or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)
These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.
Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)
The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.
If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS
How to Apply These Terms to Your New Programs
If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the program's name and an idea of what it does.
Copyright (C) yyyy name of author

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type `show w'. This is free software, and you are welcome
to redistribute it under certain conditions; type `show c'
for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Apéndice B: Código Fuente

globals.h

```
/* EtherApe
 * Copyright (C) 2001 Juan Toledo
 * $Id: globals.h,v 1.56 2001/07/06 10:14:00 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#ifndef ETHERAPE_GLOBALS_H
#define ETHERAPE_GLOBALS_H

#include <gnome.h>
#include <sys/time.h>
#include <pcap.h>
#include <glade/glade.h>

#define ETHERAPE_GLADE_FILE "etherape.glade"
#define STACK_SIZE 5                    /* How many protocol levels to keep
                                         * track of (+1) */

/* Enumerations */

/* Since gdb does understand enums and not defines, and as
 * way to make an easier transition to a non-pcap etherape,
 * I define my own enum for link type codes */
typedef enum
{
  L_NULL = DLT_NULL,                    /* no link-layer encapsulation */
  L_EN10MB = DLT_EN10MB,                /* Ethernet (10Mb) */
  L_EN3MB = DLT_EN3MB,                  /* Experimental Ethernet (3Mb) */
  L_AX25 = DLT_AX25,                    /* Amateur Radio AX.25 */
  L_PRONET = DLT_PRONET,                /* Proteon ProNET Token Ring */
  L_CHAOS = DLT_CHAOS,                  /* Chaos */
  L_IEEE802 = DLT_IEEE802,              /* IEEE 802 Networks */
  L_ARCNET = DLT_ARCNET,                /* ARCNET */
  L_SLIP = DLT_SLIP,                    /* Serial Line IP */
  L_PPP = DLT_PPP,                      /* Point-to-point Protocol */
  L_FDDI = DLT_FDDI,                    /* FDDI */
  L_ATM_RFC1483 = DLT_ATM_RFC1483,      /* LLC/SNAP encapsulated atm */
  L_RAW = DLT_RAW,                      /* raw IP */
  L_SLIP_BSDOS = DLT_SLIP_BSDOS,        /* BSD/OS Serial Line IP */
  L_PPP_BSDOS = DLT_PPP_BSDOS           /* BSD/OS Point-to-point Protocol */
}
link_type_t;

typedef enum
{
  /* Beware! The value given by the option widget is dependant on
   * the order set in glade! */
  LINEAR = 0,
  LOG = 1,
  SQRT = 2
}
size_mode_t;

typedef enum
{
  INST_TOTAL = 0,
  INST_INBOUND = 1,
  INST_OUTBOUND = 2,
  ACCU_TOTAL = 3,
  ACCU_INBOUND = 4,
  ACCU_OUTBOUND = 5
}
```

```
node_size_variable_t;

typedef enum
{
 DEFAULT = -1,
 ETHERNET = 0,
 FDDI = 1,
 IEEE802 = 2,
 IP = 3,
 IPX = 4,
 TCP = 5,
 UDP = 6
}
apemode_t;

/* Flag to indicate whether a packet belongs to a node or a link */
/* TODO This has to go, it should be a packet property */
enum packet_belongs
{
 NODE = 0, LINK = 1, PROTOCOL = 2
};

/* Flag used in node packets to indicate whether this packet was
 * inbound or outbound for the parent node */
typedef enum
{
 INBOUND = 0, OUTBOUND = 1
}
packet_direction;

/* Possible states of capture status */
enum status_t
{
 STOP = 0, PLAY = 1, PAUSE = 2
};

/* Capture structures */

typedef struct
{
 guint8 *node_id;
 GString *name;
 GString *numeric_name;
 gboolean solved;
 gdouble accumulated;
 gdouble n_packets;
}
name_t;

typedef struct
{
 guint8 *node_id;          /* pointer to the node identification
                            * could be an ether or ip address*/
 GString *name;            /* String with a readable default name of the node */
 GString *numeric_name;    /* String with a numeric representation of the id */
 guint32 ip_address;       /* Needed by the resolver */
 GString *numeric_ip;      /* Ugly hack for ethernet mode */
 gdouble average;          /* Average bytes in or out in the last x ms */
 gdouble average_in;       /* Average bytes in in the last x ms */
 gdouble average_out;      /* Average bytes out in the last x ms */
 gdouble accumulated;      /* Accumulated bytes */
 gdouble accumulated_in;   /* Accumulated incoming bytes */
 gdouble accumulated_out;  /* Accumulated outcoming bytes */
 gdouble aver_accu;        /* Accumulated bytes in the last x ms */
 gdouble aver_accu_in;     /* Accumulated incoming bytes in the last x ms */
 gdouble aver_accu_out;    /* Accumulated outcoming bytes in the last x ms */
 gdouble n_packets;        /* Number of total packets received */
 struct timeval last_time; /* Timestamp of the last packet to be added */
 GList *packets;           /* List of packets sizes in or out and
                                                  * its sizes. Used to calculate average
                                                  * traffic */
 gchar *main_prot[STACK_SIZE + 1];     /* Most common protocol for the node */
 GList *protocols[STACK_SIZE + 1];     /* It's a stack. Each level is a list of
                                                          * all protocols heard at that
level */

}
node_t;

/* Link information */
typedef struct
{
 guint8 *link_id;                      /* pointer to guint8 containing src and
                                        * destination nodes_id's of the link */
 gdouble average;
 gdouble accumulated;
 guint n_packets;
 struct timeval last_time;     /* Timestamp of the last packet added */
 GList *packets;                       /* List of packets heard on this link */
 gchar *main_prot[STACK_SIZE + 1];     /* Most common protocol for the link */
 GList *protocols[STACK_SIZE + 1];     /* It's a stack. Each level is a list of
                                * all protocols heard at that level */
```

```
  gchar *src_name;
  gchar *dst_name;
}
link_t;

/* Information about each packet heard on the network */
typedef struct
{
  guint size;                   /* Size in bytes of the packet */
  struct timeval timestamp;     /* Time at which the packet was heard */
  gchar *prot;                  /* Packet protocol tree */
  guint ref_count;              /* How many structures are referencing this
                                 * packet. When the count reaches zero the packet
                                 * is deleted */
  guint8 *src_id;               /* Source and destination ids for the packet */
  guint8 *dst_id;               /* Useful to identify direction in node_packets */
}
packet_t;

/* Information about each protocol heard on a link */
typedef struct
{
  gchar *name;                  /* Name of the protocol */
  gdouble average;              /* Average bytes in or out in the last x ms */
  gdouble aver_accu;            /* Accumulated bytes in the last x ms */
  gdouble accumulated;          /* Accumulated traffic in bytes for this protocol */
  guint n_packets;              /* Number of packets containing this protocol */
  GList *packets;               /* List of packets that used this protocol */
  GdkColor color;               /* The color associated with this protocol. It's here
                                 * so that I can use the same structure and lookup functions
                                 * in capture.c and diagram.c */
  GList *node_names;            /* Has a list of all node names used with this
                                 * protocol (used in node protocols) */
  GList *node_ids;              /* Has a list of all the nodes that have used this
                                 * protocol (used in the global protocols list) */
  struct timeval last_heard;    /* The last at which this protocol carried traffic */
}
protocol_t;

/* Diagram structures */

typedef struct
{
  guint8 *canvas_node_id;
  node_t *node;
  GnomeCanvasItem *node_item;
  GnomeCanvasItem *text_item;
  GnomeCanvasGroup *group_item;
  GdkColor color;
  gboolean is_new;
  gboolean shown;               /* True if it is to be displayed. */
}
canvas_node_t;

typedef struct
{
  guint8 *canvas_link_id;
  link_t *link;
  GnomeCanvasItem *link_item;
  GdkColor color;
}
canvas_link_t;

GTree *canvas_nodes;        /* We don't use the nodes tree directly in order to
                             * separate data from presentation: that is, we need to
                             * keep a list of CanvasItems, but we do not want to keep
                             * that info on the nodes tree itself */
GTree *canvas_links;        /* See above */
gboolean already_updating;  /* True while an instance of update_diagram is running */


/* Variables */

GladeXML *xml;
GtkWidget *app1;                        /* Pointer to the main app window */
GtkWidget *diag_pref;                   /* Pointer to the diagram configuration window */
struct timeval now;                     /* Set both at each packet capture and
                                         * in each redraw of the diagram */
GTree *nodes;                           /* Has all the nodes heard on the network */
GTree *links;                           /* Has all links heard on the net */
GList *protocols[STACK_SIZE + 1];       /* It's a stack. Each level is a list of
                                         * all protocols heards at that level */
gdouble n_packets;                      /* Number of total packets received */
gdouble n_mem_packets;                  /* Number of packets currently in memory */

link_type_t linktype;                   /* Type of device we are listening to */
guint node_id_length;                   /* Length of the node_id key. Depends
                                         * on the mode of operation */
guint l3_offset;                        /* Offset to the level 3 protocol data
                                         * Depends of the linktype */
enum status_t status;                   /* Keeps capture status (playing, stopped, paused) */
gboolean end_of_file;                   /* Marks that the end of the offline file
```

```
                                        * has been reached */

/* Genereral settings */

gchar *input_file;                      /* Capture file to read from */
gboolean numeric;                       /* Whether dns lookups are performed */
gboolean dns;                           /* Negation of the above. Is used by dns.c */
gint diagram_timeout;                   /* Descriptor of the diagram timeout function
                                         * (Used to change the refresh_period in the callback */
apemode_t mode;                         /* Mode of operation. Can be
                                         * ETHERNET, IP, UDP or TCP */

/* Diagram settings */

gboolean diagram_only;      /* Do not use text on the diagram */
gboolean group_unk;         /* Whether to display as one every unkown port protocol */
gboolean nofade;            /* Do not fade unused links */
gboolean stationary;        /* Use alternative algorith for node placing */
guint32 refresh_period;     /* Time between diagram refreshes */
gdouble node_radius_multiplier;         /* used to calculate the radius of the
                                         * displayed nodes. So that the user can
                                         * select with certain precision this
                                         * value, the GUI uses the log10 of the
                                         * multiplier */
gdouble link_width_multiplier;          /* Same explanation as above */
size_mode_t size_mode;                  /* Default mode for node size and
                                         * link width calculation */
node_size_variable_t node_size_variable;    /* Default variable that sets the node
                                             * size */
gchar *node_color, *link_color, *text_color;/* Default colors
                                             * TODO do we need link_color anymore? */
gchar *fontname;                        /* Font to be used for text display */
gboolean need_reposition;   /* Force a diagram relayout */
guint stack_level;                      /* Which level of the protocol stack
                                         * we will concentrate on */
gint node_limit;                        /* Max number of nodes to show. If <0 it's not
                                         * limited */
gdouble gui_node_timeout_time;          /* After this time has passed with no traffic
                                         * for a node, it disappears from the diagram */
GList *legend_protocols;


/* Capture settings */

gdouble averaging_time;                 /* Microseconds of time we consider to
                                         * calculate traffic averages */
gdouble link_timeout_time;  /* After this time
                             * has passed with no traffic in a
                             * link, it disappears */
gdouble node_timeout_time;  /* After this time has passed
                             * with no traffic in/out a
                             * node, it is deleted from memory */
gchar *interface;           /* Network interface to listen to */
gchar *filter;              /* Pcap filter to be used */


/* Global functions declarations */

/* From main.c */
void cleanup (int signum);

/* From capture.c */
gchar *init_capture (void);
gboolean start_capture (void);
gboolean pause_capture (void);
gboolean stop_capture (void);
void cleanup_capture (void);
gint set_filter (gchar * filter, gchar * device);
void update_nodes (void);
void update_links (void);
void update_protocols (void);
node_t *ape_get_new_node (void);        /* Returns a new node that hasn't been heard of */
struct timeval substract_times (struct timeval a, struct timeval b);
gint node_id_compare (gconstpointer a, gconstpointer b);
gint link_id_compare (gconstpointer a, gconstpointer b);
gint protocol_compare (gconstpointer a, gconstpointer b);
gchar *ip_to_str (const guint8 * ad);
gchar *ether_to_str (const guint8 * ad);
gchar *ether_to_str_punct (const guint8 * ad, char punct);
void dump_node_info (node_t * node);

/* From protocols.c */
gchar *get_packet_prot (const guint8 * packet, guint len);

/* From names.c */
void get_packet_names (GList ** protocols,
                       const guint8 * packet,
                       guint16 size,
                       const gchar * prot_stack, packet_direction direction);

/* From diagram.c */
guint update_diagram (GtkWidget * canvas);
```

```
void init_diagram (void);
void destroying_timeout (gpointer data);
void destroying_idle (gpointer data);
void set_appbar_status (gchar * str);
void delete_gui_protocols (void);
gchar *traffic_to_str (gdouble traffic, gboolean is_speed);

/* From menus.c */
void init_menus (void);
void fatal_error_dialog (const gchar * message);
void update_history (GnomeEntry * gentry, const gchar * str,
                                        gboolean is_fileentry);
void gui_start_capture (void);
void gui_pause_capture (void);
gboolean gui_stop_capture (void);               /* gui_stop_capture might fail. For instance,
                                                 * it can't run if diagram_update is running */

/* From info_windows.c */
void display_protocols_window (void);
void create_node_info_window (canvas_node_t * canvas_node);
guint update_info_windows (void);
void update_protocols_window (void);
void update_node_info_windows (void);

/* From conversations.c */
void add_conversation (guint32 src_address, guint32 dst_address,
                                         guint16 src_port, guint16 dst_port, gchar * data);
gchar *find_conversation (guint32 src_address, guint32 dst_address,
                                                 guint16 src_port, guint16 dst_port);
void delete_conversations (void);


/* Macros */


#define g_my_debug(format, args...)      g_log (G_LOG_DOMAIN, \

G_LOG_LEVEL_DEBUG, \
                                                                        format,
##args)
#define g_my_info(format, args...)      g_log (G_LOG_DOMAIN, \

G_LOG_LEVEL_INFO, \
                                                                        format,
##args)
#define g_my_critical(format, args...)      g_log (G_LOG_DOMAIN, \

G_LOG_LEVEL_CRITICAL, \
                                                                        format,
##args)

#define IS_OLDER(diff, timeout)       ((diff.tv_sec > timeout/1000) | \
                                                                        (diff.tv_usec/1000 >=
timeout))



/* Pointer versions of ntohs and ntohl.  Given a pointer to a member of a
 * byte array, returns the value of the two or four bytes at the pointer.
 */
#define pntohs(p)  ((guint16)                \
                                        ((guint16)*((guint8 *)p+0)<<8|  \
                                                (guint16)*((guint8 *)p+1)<<0))

#define pntohl(p)  ((guint32)*((guint8 *)p+0)<<24|  \
                                        (guint32)*((guint8 *)p+1)<<16|  \
                                        (guint32)*((guint8 *)p+2)<<8|   \
                                        (guint32)*((guint8 *)p+3)<<0)

#define pletohs(p)  ((guint16)                \
                                        ((guint16)*((guint8 *)(p)+1)<<8|  \
                                                (guint16)*((guint8 *)(p)+0)<<0))


/* Takes the hi_nibble value from a byte */
#define hi_nibble(b) ((b & 0xf0) >> 4)

#endif
```

main.h

```
/* Etherape
 * Copyright (C) 2000 Juan Toledo
 * $Id: main.h,v 1.9 2001/07/06 10:14:00 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */


#include <signal.h>

static GLogLevelFlags debug_mask;
static void (*oldhandler) (int);

static gboolean quiet = FALSE;

static void load_config (gchar * prefix);
static gboolean get_version_levels (const gchar * version_string,
                                    guint * major, guint * minor,
                                    guint * patch);
static gint version_compare (const gchar * a, const gchar * b);
static void set_debug_level (void);

static void session_die (GnomeClient * client, gpointer client_data);

static gint
save_session (GnomeClient * client, gint phase, GnomeSaveStyle save_style,
              gint is_shutdown, GnomeInteractStyle interact_style,
              gint is_fast, gpointer client_data);
static void
log_handler (gchar * log_domain,
             GLogLevelFlags mask, const gchar * message, gpointer user_data);
```

main.c

```
/* EtherApe
 * Copyright (C) 2001 Juan Toledo
 * $Id: main.c,v 1.84 2001/07/06 10:14:00 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include "globals.h"
#include "main.h"


int
main (int argc, char *argv[])
{
  gchar *mode_string = NULL;
  GtkWidget *widget;
  GnomeClient *client;
  gchar *cl_filter = NULL, *cl_interface = NULL, *cl_input_file = NULL;
  poptContext poptcon;

  struct poptOption optionsTable[] = {
    {"mode", 'm', POPT_ARG_STRING, &mode_string, 0,
     N_("mode of operation"), N_("<ethernet|fddi|ip|tcp>")},
    {"interface", 'i', POPT_ARG_STRING, &cl_interface, 0,
     N_("set interface to listen to"), N_("<interface name>")},
    {"filter", 'f', POPT_ARG_STRING, &cl_filter, 0,
     N_("set capture filter"), N_("<capture filter>")},
```

```
    {"infile", 'r', POPT_ARG_STRING, &cl_input_file, 0,
     N_("set input file"), N_("<file name>")},
    {"numeric", 'n', POPT_ARG_NONE, &numeric, 0,
     N_("don't convert addresses to names"), NULL},
    {"diagram-only", 'd', POPT_ARG_NONE, &diagram_only, 0,
     N_("don't display any node text identification"), NULL},
    {"no-fade", 'F', POPT_ARG_NONE, &nofade, 0,
     N_("do not fade old links"), NULL},
    {"stationary", 's', POPT_ARG_NONE, &stationary, 0,
     N_("don't move nodes around"), NULL},
    {"node_limit", 'l', POPT_ARG_INT, &node_limit, 0,
     N_("limits nodes displayed"), N_("<number of nodes>")},
    {"quiet", 'q', POPT_ARG_NONE, &quiet, 0,
     N_("Don't show warnings"), NULL},
    {"node-color", 'N', POPT_ARG_STRING, &node_color, 0,
     N_("set the node color"), N_("color")},
    {"link-color", 'L', POPT_ARG_STRING, &link_color, 0,
     N_("set the link color"), N_("color")},
    {"text-color", 'T', POPT_ARG_STRING, &text_color, 0,
     N_("set the text color"), N_("color")},

   POPT_AUTOHELP {NULL, 0, 0, NULL, 0}
  };


#ifdef ENABLE_NLS
  bindtextdomain (PACKAGE, PACKAGE_LOCALE_DIR);
  textdomain (PACKAGE);
#endif

  /* We set the window icon to use */
  if (!getenv ("GNOME_DESKTOP_ICON"))
    putenv ("GNOME_DESKTOP_ICON=" PIXMAPS_DIR "/etherape.png");


  /* We initiate the application and read command line options */
  gnome_init_with_popt_table ("EtherApe", VERSION, argc, argv, optionsTable,
                              0, NULL);


  /* We obtain application parameters
   * First, absolute defaults
   * Second, values saved in the config file
   * Third, whatever given in the command line */

  /* Absolute defaults */
  numeric = 0;
  mode = IP;
  dns = 1;
  filter = NULL;
  status = STOP;
  refresh_period = 800;                 /* ms */
  node_color = g_strdup ("brown");
  link_color = g_strdup ("tan");        /* TODO I think link_color is
                                                        * actually never used
anymore,
                                                        * is it? */
  text_color = g_strdup ("yellow");
  node_limit = -1;


  set_debug_level ();

  /* Config file */
  load_config ("/Etherape/");

  /* Command line */
  poptcon = poptGetContext ("Etherape", argc, argv, optionsTable, 0);
  while (poptGetNextOpt (poptcon) > 0);

  if (cl_filter)
    {
      if (filter)
        g_free (filter);
      filter = g_strdup (cl_filter);
    }

  if (cl_interface)
    {
      if (interface)
        g_free (interface);
      interface = g_strdup (cl_interface);
    }

  if (cl_input_file)
    {
      if (input_file)
        g_free (input_file);
      input_file = g_strdup (cl_input_file);
    }
```

```
/* dns is used in dns.c as opposite of numeric */
dns = !numeric;

/* Find mode of operation */
if (mode_string)
  {
    if (strstr (mode_string, "ethernet"))
                mode = ETHERNET;
    else if (strstr (mode_string, "fddi"))
                mode = FDDI;
    else if (strstr (mode_string, "ip"))
                mode = IP;
    else if (strstr (mode_string, "tcp"))
                mode = TCP;
    else if (strstr (mode_string, "udp"))
                mode = UDP;
    else
                g_warning (_
                           ("Unrecognized mode. Do etherape --help for a list of modes"));
  }

/* Glade */

glade_gnome_init ();
xml = glade_xml_new (GLADEDIR "/" ETHERAPE_GLADE_FILE, NULL);
if (!xml)
  {
    g_error (_("We could not load the interface! (%s)"),
                    GLADEDIR "/" ETHERAPE_GLADE_FILE);
    return 1;
  }
glade_xml_signal_autoconnect (xml);


app1 = glade_xml_get_widget (xml, "app1");
diag_pref = glade_xml_get_widget (xml, "diag_pref");

/* Sets controls to the values of variables and connects signals */
init_diagram ();


/* Session handling */
client = gnome_master_client ();
gtk_signal_connect (GTK_OBJECT (client), "save_yourself",
                                    GTK_SIGNAL_FUNC (save_session), argv[0]);
gtk_signal_connect (GTK_OBJECT (client), "die",
                                                GTK_SIGNAL_FUNC (session_die), NULL);
  gtk_widget_show (app1);


  /*
   * Signal handling
   * Catch SIGINT and SIGTERM and, if we get either of them, clean up
   * and exit.
   * XXX - deal with signal semantics on various platforms.  Or just
   * use "sigaction()" and be done with it?
   */
  signal (SIGTERM, cleanup);
  signal (SIGINT, cleanup);
#if !defined(WIN32)
  if ((oldhandler = signal (SIGHUP, cleanup)) != SIG_DFL)/* Play nice with nohup */
    signal (SIGHUP, oldhandler);
#endif


  /* With this we force an update of the diagram every x ms
   * Data in the diagram is updated, and then the canvas redraws itself when
   * the gtk loop is idle. If the CPU can't handle the set refresh_period,
   * then it will just do a best effort */

  widget = glade_xml_get_widget (xml, "canvas1");
  diagram_timeout = g_timeout_add_full (G_PRIORITY_DEFAULT,
                                                                        refresh_period,
                                                                        (GtkFunction)
update_diagram,
                                                                        widget,
                                                                        (GDestroyNotify)
destroying_timeout);

  /* This other timeout makes sure that the info windows are updated */
  g_timeout_add (500, (GtkFunction) update_info_windows, NULL);

  init_menus ();

  gui_start_capture ();


  /* MAIN LOOP */
  gtk_main ();
  return 0;
}                                                       /* main */
```

```
/* loads configuration from .gnome/Etherape */
static void
load_config (char *prefix)
{
  gboolean u;
  gchar *config_file_version;

  gnome_config_push_prefix (prefix);

  config_file_version =
    gnome_config_get_string_with_default ("General/version=0.5.4", &u);
  diagram_only =
    gnome_config_get_bool_with_default ("Diagram/diagram_only=FALSE", &u);
  group_unk =
    gnome_config_get_bool_with_default ("Diagram/group_unk=TRUE", &u);
  stationary
    = gnome_config_get_bool_with_default ("Diagram/stationary=FALSE", &u);
  /* Not yet, since we can't force fading
     nofade = gnome_config_get_bool_with_default ("Diagram/nofade=FALSE", &u);
   */
  node_timeout_time =
    gnome_config_get_float_with_default
    ("Diagram/node_timeout_time=3600000.0", &u);
  gui_node_timeout_time =
    gnome_config_get_float_with_default
    ("Diagram/gui_node_timeout_time=60000.0", &u);
  if (nofade)
    link_timeout_time =
      gnome_config_get_float_with_default
      ("Diagram/link_timeout_time=5000.0", &u);
  else
    link_timeout_time =
      gnome_config_get_float_with_default
      ("Diagram/link_timeout_time=20000.0", &u);
  averaging_time =
    gnome_config_get_float_with_default ("Diagram/averaging_time=3000.0", &u);
  node_radius_multiplier =
    gnome_config_get_float_with_default
    ("Diagram/node_radius_multiplier=0.0005", &u);
  if (u)
    node_radius_multiplier = 0.0005;            /* This is a bug with gnome_config */
  link_width_multiplier =
    gnome_config_get_float_with_default
    ("Diagram/link_width_multiplier=0.0005", &u);
  if (u)
    link_width_multiplier = 0.0005;
  mode = gnome_config_get_int_with_default ("General/mode=-1", &u);/* DEFAULT */
  if (mode == IP || mode == TCP)
    refresh_period =
      gnome_config_get_int_with_default ("Diagram/refresh_period=3000", &u);
  else
    refresh_period =
      gnome_config_get_int_with_default ("Diagram/refresh_period=800", &u);

  size_mode = gnome_config_get_int_with_default ("Diagram/size_mode=0", &u);      /* LINEAR */
  node_size_variable = gnome_config_get_int_with_default ("Diagram/node_size_variable=2", &u);  /* INST_OUTBOUN
  stack_level =
    gnome_config_get_int_with_default ("Diagram/stack_level=0", &u);
  if ((stack_level != 0)
      && (version_compare (config_file_version, "0.5.4") < 0))
    g_warning (_("Stack Level is not set to Topmost Recognized Protocol. "
                 "Please check in the preferences dialog that this is what "
                 "you really want"));
  fontname =
    gnome_config_get_string_with_default
    ("Diagram/fontname=-*-*-*-*-*-*-*-*-140-*-*-*-*-iso8859-1", &u);

  g_free (config_file_version);
  gnome_config_pop_prefix ();
}                                                       /* load_config */

static gint
version_compare (const gchar * a, const gchar * b)
{
  guint a_mj, a_mi, a_pl, b_mj, b_mi, b_pl;

  g_assert (a != NULL);
  g_assert (b != NULL);

  /* TODO What should we return if there was a problem? */
  g_return_val_if_fail ((get_version_levels (a, &a_mj, &a_mi, &a_pl)
                         && get_version_levels (b, &b_mj, &b_mi, &b_pl)), 0);
  if (a_mj < b_mj)
    return -1;
  else if (a_mj > b_mj)
    return 1;
  else if (a_mi < b_mi)
    return -1;
  else if (a_mi > b_mi)
    return 1;
```

```
  else if (a_pl < b_pl)
    return -1;
  else if (a_pl > b_pl)
    return 1;
  else
    return 0;
}

static gboolean
get_version_levels (const gchar * version_string,
                    guint * major, guint * minor, guint * patch)
{
  gchar **tokens;

  g_assert (version_string != NULL);

  tokens = g_strsplit (version_string, ".", 0);
  g_return_val_if_fail ((tokens
                         && tokens[0] && tokens[1] && tokens[2]
                         && sscanf (tokens[0], "%d", major)
                         && sscanf (tokens[1], "%d", minor)
                         && sscanf (tokens[2], "%d", patch)), FALSE);
  g_strfreev (tokens);
  return TRUE;
}



/* saves configuration to .gnome/Etherape */
/* It's not static since it will be called from the GUI */
void
save_config (char *prefix)
{
  gnome_config_push_prefix (prefix);
  gnome_config_set_bool ("Diagram/diagram_only", diagram_only);
  gnome_config_set_bool ("Diagram/group_unk", group_unk);
  gnome_config_set_bool ("Diagram/nofade", nofade);
  gnome_config_set_float ("Diagram/node_timeout_time", node_timeout_time);
  gnome_config_set_float ("Diagram/gui_node_timeout_time",
                          gui_node_timeout_time);
  gnome_config_set_float ("Diagram/link_timeout_time", link_timeout_time);
  gnome_config_set_float ("Diagram/averaging_time", averaging_time);
  gnome_config_set_float ("Diagram/node_radius_multiplier",
                          node_radius_multiplier);
  gnome_config_set_float ("Diagram/link_width_multiplier",
                          link_width_multiplier);
#if 0                                   /* TODO should we save this? */
  gnome_config_set_int ("General/mode", mode);
#endif
  gnome_config_set_int ("Diagram/refresh_period", refresh_period);
  gnome_config_set_int ("Diagram/size_mode", size_mode);
  gnome_config_set_int ("Diagram/node_size_variable", node_size_variable);
  gnome_config_set_int ("Diagram/stack_level", stack_level);
  gnome_config_set_string ("Diagram/fontname", fontname);
  gnome_config_set_string ("General/version", VERSION);

  gnome_config_sync ();
  gnome_config_pop_prefix ();

  g_my_info (_("Preferences saved"));

}                                       /* save_config */


static void
set_debug_level (void)
{
  gchar *env_debug;
  env_debug = g_getenv ("DEBUG");

  debug_mask = (G_LOG_LEVEL_MASK & ~(G_LOG_LEVEL_DEBUG |
G_LOG_LEVEL_INFO));

  if (env_debug)
    {
      if (!strcmp (env_debug, "INFO"))
        debug_mask = (G_LOG_LEVEL_MASK & ~G_LOG_LEVEL_DEBUG);
      else if (!strcmp (env_debug, "DEBUG"))
        debug_mask = G_LOG_LEVEL_MASK;
    }

  if (quiet)
    debug_mask = 0;

  g_log_set_handler (NULL, G_LOG_LEVEL_MASK, (GLogFunc) log_handler, NULL);
  g_my_debug ("debug_mask %d", debug_mask);
}

static void
log_handler (gchar * log_domain,
             GLogLevelFlags mask, const gchar * message, gpointer user_data)
{
```

```
  if (mask & debug_mask)
    g_log_default_handler (NULL, mask, message, user_data);
}

/* the gnome session manager may call this function */
static void
session_die (GnomeClient * client, gpointer client_data)
{
  g_message ("in die");
  gtk_main_quit ();
}                                               /* session_die */

/* the gnome session manager may call this function */
static gint
save_session (GnomeClient * client, gint phase, GnomeSaveStyle save_style,
              gint is_shutdown, GnomeInteractStyle interact_style,
              gint is_fast, gpointer client_data)
{
  gchar **argv;
  guint argc;

  /* allocate 0-filled, so it will be NULL-terminated */
  argv = g_malloc0 (sizeof (gchar *) * 4);
  argc = 1;

  argv[0] = client_data;

  g_message ("In save_session");
#if 0
  if (message)
    {
      argv[1] = "--message";
      argv[2] = message;
      argc = 3;
    }
#endif

  gnome_client_set_clone_command (client, argc, argv);
  gnome_client_set_restart_command (client, argc, argv);

  return TRUE;
}                                               /* save_session */


/*
 * Quit the program.
 * Makes sure that the capture device is closed, or else we might
 * be leaving it in promiscuous mode
 */
void
cleanup (int signum)
{
  cleanup_capture ();
  gtk_exit (0);
}
```

capture.h

```
/* Etherape
 * Copyright (C) 2000 Juan Toledo
 * $Id: capture.h,v 1.55 2001/05/11 09:52:51 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <pcap.h>
#include "globals.h"

#define MAXSIZE 200
#define PCAP_TIMEOUT 250

/*
 * LOCAL ENUMERATIONS
 */


/* Used on some functions to indicate how to operate on the node info
 * depending on what side of the comm the node was at */
typedef enum
{
  SRC = 0,
  DST = 1
}
create_node_type_t;

static pcap_t *pch;                     /* pcap structure */
static struct pcap_pkthdr phdr;

static guint32 ms_to_next;    /* Used for offline mode to store the amount
                               * of time that we have to wait between
                               * one packet and the next */
gint pcap_fd;                 /* The file descriptor used by libpcap */
gint capture_source;          /* It's the input tag or the timeout tag,
                               * in online or offline mode */

static gint dns_fd = 0;       /* The file descriptor used by dns.c */
static GList *new_nodes = NULL;  /* List that contains every new node not yet
                                  * acknowledged by the main app with
                                  * ape_get_new_node */

/* Local funtions declarations */
static guint get_offline_packet (void);
static void cap_t_o_destroy (gpointer data);
static void packet_read (guint8 * packet, gint source,
                         GdkInputCondition condition);

static guint8 *get_node_id (const guint8 * packet,
                            create_node_type_t node_type);
static guint8 *get_link_id (const guint8 * packet);

static node_t *create_node (const guint8 * packet, const guint8 * node_id);
static link_t *create_link (const guint8 * packet, const guint8 * link_id);

static void add_node_packet (const guint8 * packet,
                             packet_t * packet_info,
                             const guint8 * node_id,
                             packet_direction direction);
static void add_link_packet (const guint8 * packet,
                             packet_t * packet_info, const guint8 * link_id);
void add_protocol (GList ** protocols, const gchar * stack,
                   struct pcap_pkthdr phdr, packet_t * packet_info);

static gint update_node (guint8 * node_id, node_t * node, gpointer pointer);
static gint update_link (guint8 * link_id, link_t * link, gpointer pointer);
static gboolean update_protocol (protocol_t * protocol);


static void update_node_names (node_t * node);
static void set_node_name (node_t * node, gchar * preferences);

static void update_packet_list (GList * packets, gpointer parent,
                                enum packet_belongs belongs_to);
static gboolean check_packet (GList * packets, GList ** packet_l_e,
                              gpointer parent,
                              enum packet_belongs belongs_to);
```

```
static void forget_node_from_protocols (guint8 * node_id);
static gchar *get_main_prot (GList * packets,
                             GList ** protocols, guint level);
static gint prot_freq_compare (gconstpointer a, gconstpointer b);
static gint names_freq_compare (gconstpointer a, gconstpointer b);
gchar *print_mem (const guint8 * ad, guint length);
static void dns_ready (gpointer data, gint fd, GdkInputCondition cond);
```


capture.c

```
/* EtherApe
 * Copyright (C) 2001 Juan Toledo
 * $Id: capture.c,v 1.140 2001/07/06 10:14:00 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <gnome.h>
#include <ctype.h>
#include <netinet/in.h>
#include <pcap.h>

#include "globals.h"
#include "capture.h"
#include "dns.h"
#include "eth_resolv.h"

/*
 * FUNCTION DEFINITIONS
 */

/* Sets up the pcap device
 * Sets up the mode and related variables
 * Sets up dns if needed
 * Sets up callbacks for pcap and dns
 * Creates nodes and links trees */
gchar *
init_capture (void)
{

  guint i = STACK_SIZE;
```

```
  gchar *device;
  gchar ebuf[300];
  gchar *str = NULL;
  gboolean error = FALSE;
  static gchar errorbuf[300];
  static gboolean data_initialized = FALSE;


  if (!data_initialized)
    {
      nodes = g_tree_new (node_id_compare);
      links = g_tree_new (link_id_compare);
      while (i + 1)
                  {
                   protocols[i] = NULL;
                   i--;
                  }
      if (!numeric)
                  {
                   dns_open ();
                   dns_fd = dns_waitfd ();
                   g_my_debug ("File descriptor for DNS is %d", dns_fd);
                   gdk_input_add (dns_fd,
                                  GDK_INPUT_READ, (GdkInputFunction) dns_ready,
NULL);
                  }
      else
                  dns_fd = 0;

      status = STOP;
      data_initialized = TRUE;

      n_packets = n_mem_packets = 0;
    }

  device = interface;
  if (!device && !input_file)
    {
      device = g_strdup (pcap_lookupdev (ebuf));
      if (device == NULL)
                  {
                   sprintf (errorbuf, _("Error getting device: %s"), ebuf);
                   return errorbuf;
                  }
      /* TODO I should probably tidy this up, I probably don't
       * need the local variable device. But I need to reset
       * interface since I need to know whether we are in
       * online or offline mode later on */
      interface = device;
    }


  end_of_file = FALSE;
  if (!input_file)
    {
      if (!
                  ((pcap_t *) pch =
                   pcap_open_live (device, MAXSIZE, TRUE, PCAP_TIMEOUT, ebuf)))
                  {
                   sprintf (errorbuf,
                                  _("Error opening %s : %s - perhaps you need to be root?"),
                                  device, ebuf);
                   return errorbuf;
                  }
      pcap_fd = pcap_fileno (pch);
      g_my_info (_("Live device %s opened for capture. pcap_fd: %d"), device,
                                  pcap_fd);
    }
  else
    {
      if (device)
                  {
                   sprintf (errorbuf,
                                  _("Can't open both %s and device %s. Please choose one."),
                                  input_file, device);
                   return errorbuf;
                  }
      if (!((pcap_t *) pch = pcap_open_offline (input_file, ebuf)))
                  {
                   sprintf (errorbuf, _("Error opening %s : %s"), input_file, ebuf);
                   return errorbuf;
                  }
      g_my_info (_("%s opened for offline capture"), input_file);

    }


  linktype = pcap_datalink (pch);

  /* l3_offset is equal to the size of the link layer header */

  switch (linktype)
```

```
{
case L_EN10MB:
  g_my_info (_("Link type is Ethernet"));
  if (mode == DEFAULT)
                mode = IP;
  if (mode == FDDI)
                error = TRUE;
  l3_offset = 14;
  break;
case L_RAW:                             /* The case for PPP or SLIP, for instance */
  g_my_info (_("Link type is RAW"));
  if (mode == DEFAULT)
                mode = IP;
  if ((mode == ETHERNET) || (mode == FDDI) || (mode == IEEE802))
                error = TRUE;
  l3_offset = 0;
  break;
case L_FDDI:                            /* We are assuming LLC async frames only */
  g_my_info (_("Link type is FDDI"));
  if (mode == DEFAULT)
                mode = IP;
  if ((mode == ETHERNET) || (mode == IEEE802))
                error = TRUE;
  l3_offset = 21;
  break;
case L_IEEE802:
  /* As far as I know IEEE802 is Token Ring */
  g_my_info (_("Link type is Token Ring"));
  if (mode == DEFAULT)
                mode = IP;
  if ((mode == ETHERNET) || (mode == FDDI))
                error = TRUE;
  l3_offset = 22;
  break;
case L_NULL:                            /* Loopback */
  g_my_info (_("Link type is NULL"));
  if (mode == DEFAULT)
                mode = IP;
  if ((mode == ETHERNET) || (mode == FDDI) || (mode == IEEE802))
                error = TRUE;
  l3_offset = 4;
  break;
default:
  sprintf (errorbuf, _("Link type not yet supported"));
  return errorbuf;
}
```

```
/* TODO Shouldn't we free memory somwhere because of the strconcat? */
switch (mode)
  {
  case IP:
    if (filter)
                str = g_strconcat ("ip and ", filter, NULL);
    else
                {
                  g_free (filter);
                  str = g_strdup ("ip");
                }
    break;
  case TCP:
    if (filter)
                str = g_strconcat ("tcp and ", filter, NULL);
    else
                {
                  g_free (filter);
                  str = g_strdup ("tcp");
                }
    break;
  case UDP:
    if (filter)
                str = g_strconcat ("udp and ", filter, NULL);
    else
                {
                  g_free (filter);
                  str = g_strdup ("udp");
                }
    break;
  case DEFAULT:
  case ETHERNET:
  case IPX:
  case FDDI:
  case IEEE802:
    if (filter)
                str = g_strdup (filter);
    break;
  }
g_free (filter);
filter = str;
str = NULL;

if (filter)
  set_filter (filter, device);
```

```
 if (error)
   {
    sprintf (errorbuf, _("Mode not available in this device"));
    return errorbuf;
   }

 switch (mode)
   {
  case ETHERNET:
    node_id_length = 6;
    break;
  case FDDI:
    node_id_length = 6;
    break;
  case IEEE802:
    node_id_length = 6;
    break;
  case IP:
    node_id_length = 4;
    break;
  case TCP:
    node_id_length = 6;
    break;
  default:
    sprintf (errorbuf, _("Ape mode not yet supported"));
    return errorbuf;
   }


 return NULL;
}                                                   /* init_capture */

/* TODO make it return an error value and act accordingly */
/* Installs a filter in the pcap structure */
gint
set_filter (gchar * filter, gchar * device)
{
 gchar ebuf[300];
 static bpf_u_int32 netnum, netmask;
 static struct bpf_program fp;
 gboolean ok = 1;


 if (!pch)
   return 1;

 /* A capture filter was specified; set it up. */
 if (device && (pcap_lookupnet (device, &netnum, &netmask, ebuf) < 0))
   {
    g_warning (_
                                ("Can't use filter:  Couldn't obtain netmask info (%s)."),
                                ebuf);
    ok = 0;
   }
 if (ok && (pcap_compile (pch, &fp, filter, 1, netmask) < 0))
   {
    g_warning (_("Unable to parse filter string (%s)."), pcap_geterr (pch));
    ok = 0;
   }
 if (ok && (pcap_setfilter (pch, &fp) < 0))
   {
    g_warning (_("Can't install filter (%s)."), pcap_geterr (pch));
   }

 return 0;
#if 0
 /* TODO pending to be more general, since we want to be able
  * to change the capturing device in runtime. */
 if (!current_device)
  current_device = g_strdup (device);

 /* A capture filter was specified; set it up. */
 if (current_device
     && (pcap_lookupnet (current_device, &netnum, &netmask, ebuf) < 0))
   {
    g_warning (_
                                ("Can't use filter:  Couldn't obtain netmask info (%s)."),
                                ebuf);
    ok = 0;
   }
 if (ok && (pcap_compile (pch, &fp, filter, 1, netmask) < 0))
   {
    g_warning (_("Unable to parse filter string (%s)."), pcap_geterr (pch));
    ok = 0;
   }
 if (ok && (pcap_setfilter (pch, &fp) < 0))
   {
    g_warning (_("Can't install filter (%s)."), pcap_geterr (pch));
   }
#endif
```

```
}

gboolean
start_capture (void)
{

 if ((status != PAUSE) && (status != STOP))
   {
    g_warning (_("Status not PAUSE or STOP at start_capture"));
    return FALSE;
   }

 /*
  * See pause_capture for an explanation of why we don't always
  * add the source
  */
 if (interface && (status == STOP))
   {
    g_my_debug (_("Starting live capture"));
    capture_source = gdk_input_add (pcap_fd,
                                    GDK_INPUT_READ,
                                    (GdkInputFunction) packet_read,
NULL);
   }
 else
   {
    g_my_debug (_("Starting offline capture"));
    capture_source = g_timeout_add_full (G_PRIORITY_DEFAULT,
                                         1,
                                         (GtkFunction)
get_offline_packet,
                                         NULL,
                                         (GDestroyNotify)
cap_t_o_destroy);
   }

 status = PLAY;
 return TRUE;
}                                       /* start_capture */

gboolean
pause_capture (void)
{
 if (status != PLAY)
   g_warning (_("Status not PLAY at pause_capture"));

 if (interface)
   {
    /* Why would we want to miss packets while pausing to
     * better analyze a moment in time? If we wanted
     * to do this it should be optional at least.
     * In order for this to work, start_capture should only
     * add the source if the pause was for an offline capture */
#if 0
    g_my_debug (_("Pausing live capture"));
    gdk_input_remove (capture_source);      /* gdk_input_remove does not
                                                                * return an
error code */
#endif
   }
 else
   {
    g_my_debug (_("Pausing offline capture"));
    if (!end_of_file)
                {
                 if (!g_source_remove (capture_source))
                   {
                    g_warning (_("Error while trying to pause capture"));
                    return FALSE;
                   }
                }
   }

 status = PAUSE;
 return TRUE;

}


gboolean
stop_capture (void)
{
 struct pcap_stat ps;

 if ((status != PLAY) && (status != PAUSE))
   {
    g_warning (_("Status not PLAY or PAUSE at stop_capture"));
    return FALSE;
   }

 if (interface)
   {
```

```
    g_my_debug (_("Stopping live capture"));
    gdk_input_remove (capture_source);      /* gdk_input_remove does not
                                                                    * return an
error code */
   }
  else
   {
    g_my_debug (_("Stopping offline capture"));
    if (!end_of_file)
                {
                 if (!g_source_remove (capture_source))
                   {
                    g_warning (_
                                                ("Error while removing capture source in
stop_capture"));
                    return FALSE;
                   }
                }
   }

  status = STOP;

  /* With status in STOP, all protocols, nodes and links will be deleted */
  update_protocols ();
  update_nodes ();
  update_links ();

  /* Free conversations */
  delete_conversations ();

  /* Free the list of new_nodes */
  g_list_free (new_nodes);
  new_nodes = NULL;

  if (filter)
    {
     g_free (filter);
     filter = NULL;
    }

  /* Clean the buffer */
  if (!interface)
    get_offline_packet ();

  /* Close the capture */
  pcap_stats (pch, &ps);
  g_my_debug ("libpcap received %d packets, dropped %d. EtherApe saw %g",
                    ps.ps_recv, ps.ps_drop, n_packets);
  n_packets = 0;
  pcap_close (pch);
  g_my_info (_("Capture device stopped or file closed"));

  return TRUE;
}                                                   /* stop_capture */

/*
 * Makes sure we don't leave any open device behind, or else we
 * might leave it in promiscous mode
 */
void
cleanup_capture (void)
{
  if (status != STOP)
    stop_capture ();
}


/* This is a timeout function used when reading from capture files
 * It forces a waiting time so that it reproduces the rate
 * at which packets were coming */
static guint
get_offline_packet (void)
{
  static guint8 *packet = NULL;
  static struct timeval last_time = { 0, 0 }, this_time, diff;

  if (status == STOP)
    {
     packet = NULL;
     last_time.tv_usec = last_time.tv_sec = 0;
     return FALSE;
    }

  if (packet)
    packet_read (packet, 0, GDK_INPUT_READ);

  packet = (guint8 *) pcap_next (pch, &phdr);
  if (!packet)
    end_of_file = TRUE;

  if (last_time.tv_sec == 0 && last_time.tv_usec == 0)
    {
```

```
    last_time.tv_sec = phdr.ts.tv_sec;
    last_time.tv_usec = phdr.ts.tv_usec;
   }

  this_time.tv_sec = phdr.ts.tv_sec;
  this_time.tv_usec = phdr.ts.tv_usec;

  diff = substract_times (this_time, last_time);
  ms_to_next = diff.tv_sec * 1000 + diff.tv_usec / 1000;

  last_time = this_time;

  return FALSE;
}                                                   /* get_offline_packet */


static void
cap_t_o_destroy (gpointer data)
{

  if ((status == PLAY) && !end_of_file)
    capture_source = g_timeout_add_full (G_PRIORITY_DEFAULT,
                                                                ms_to_next,
                                                                (GtkFunction)
get_offline_packet,
                                                                data,
                                                                (GDestroyNotify)
cap_t_o_destroy);

}                                                   /* capture_t_o_destroy */


/* This function is called everytime there is a new packet in
 * the network interface. It then updates traffic information
 * for the appropriate nodes and links */
static void
packet_read (guint8 * packet, gint source, GdkInputCondition condition)
{
  guint8 *link_id = NULL;
  node_t *node = NULL;
  packet_t *packet_info = NULL;
  gchar *prot = NULL;

  /* Redhat's phdr.ts is not a timeval, so I can't
   * just copy the structures */

  /* Get next packet only if in live mode */
  if (source)
    {
      packet = (guint8 *) pcap_next (pch, &phdr);
      now.tv_sec = phdr.ts.tv_sec;
      now.tv_usec = phdr.ts.tv_usec;
    }
  else
    gettimeofday (&now, NULL);

  if (!packet)
    return;

  /* Get a string with the protocol tree */
  prot = get_packet_prot (packet, phdr.len);

  /* We create a packet structure to hold data */
  packet_info = g_malloc (sizeof (packet_t));
  packet_info->size = phdr.len;
  packet_info->timestamp = now;
  packet_info->prot = g_strdup (prot);
  packet_info->ref_count = 0;

  /* If there is no node with that id, create it. Otherwise
   * just use the one available */
  if ((node = g_tree_lookup (nodes, get_node_id (packet, SRC))))
    packet_info->src_id = node->node_id;
  else
    packet_info->src_id =
      g_memdup (get_node_id (packet, SRC), node_id_length);
  if ((node = g_tree_lookup (nodes, get_node_id (packet, DST))))
    packet_info->dst_id = node->node_id;
  else
    packet_info->dst_id =
      g_memdup (get_node_id (packet, DST), node_id_length);

  n_packets++;
  n_mem_packets++;

  add_protocol (protocols, prot, phdr, packet_info);

  /* Add this packet information to the src and dst nodes. If they
   * don't exist, create them */
  add_node_packet (packet, packet_info, packet_info->src_id, OUTBOUND);
  add_node_packet (packet, packet_info, packet_info->dst_id, INBOUND);
```

```
  link_id = get_link_id (packet);
  /* And now we update link traffic information for this packet */
  add_link_packet (packet, packet_info, link_id);
#ifndef LEAK
  return;
#endif


}                                                       /* packet_read */


/* Returns a pointer to a set of octects that define a link for the
 * current mode in this particular packet */
static guint8 *
get_node_id (const guint8 * packet, create_node_type_t node_type)
{
  static guint8 *node_id = NULL;

  if (node_id)
    {
      g_free (node_id);
      node_id = NULL;
    }

  switch (mode)
    {
    case ETHERNET:
      if (node_type == SRC)
                node_id = g_memdup (packet + 6, node_id_length);
      else
                node_id = g_memdup (packet, node_id_length);
      break;
    case FDDI:
      if (node_type == SRC)
                node_id = g_memdup (packet + 7, node_id_length);
      else
                node_id = g_memdup (packet + 1, node_id_length);
      break;
    case IEEE802:
      if (node_type == SRC)
                node_id = g_memdup (packet + 8, node_id_length);
      else
                node_id = g_memdup (packet + 2, node_id_length);
      break;
    case IP:
      if (node_type == SRC)
                node_id = g_memdup (packet + l3_offset + 12, node_id_length);
      else
                node_id = g_memdup (packet + l3_offset + 16, node_id_length);
      break;
    case TCP:
      node_id = g_malloc (node_id_length);
      if (node_type == SRC)
                {
                  guint16 port;
                  g_memmove (node_id, packet + l3_offset + 12, 4);
                  port = ntohs (*(guint16 *) (packet + l3_offset + 20));
                  g_memmove (node_id + 4, &port, 2);
                }
      else
                {
                  guint16 port;
                  g_memmove (node_id, packet + l3_offset + 16, 4);
                  port = ntohs (*(guint16 *) (packet + l3_offset + 22));
                  g_memmove (node_id + 4, &port, 2);
                }
      break;
    default:
      g_error (_("Reached default in get_node_id"));
    }

  return node_id;
}                                                       /* get_node_id */

/* Returns a pointer to a set of octects that define a link for the
 * current mode in this particular packet */
static guint8 *
get_link_id (const guint8 * packet)
{
  static guint8 *link_id = NULL;
  guint16 port;

  if (link_id)
    {
      g_free (link_id);
      link_id = NULL;
    }


  switch (mode)
    {
```

```
    case ETHERNET:
      link_id = g_malloc (2 * node_id_length);
      g_memmove (link_id, packet + 6, node_id_length);
      g_memmove (link_id + 6, packet, node_id_length);
      break;
    case FDDI:
      link_id = g_malloc (2 * node_id_length);
      g_memmove (link_id, packet + 7, node_id_length);
      g_memmove (link_id + 6, packet + 1, node_id_length);
      break;
    case IEEE802:
      link_id = g_malloc (2 * node_id_length);
      g_memmove (link_id, packet + 8, node_id_length);
      g_memmove (link_id + 6, packet + 2, node_id_length);
      break;
    case IP:
      link_id = g_memdup (packet + l3_offset + 12, 2 * node_id_length);
      break;
    case TCP:

      link_id = g_malloc (2 * node_id_length);
      g_memmove (link_id, packet + l3_offset + 12, 4);
      port = ntohs (*(guint16 *) (packet + l3_offset + 20));
      g_memmove (link_id + 4, &port, 2);
      g_memmove (link_id + 6, packet + l3_offset + 16, 4);
      port = ntohs (*(guint16 *) (packet + l3_offset + 22));
      g_memmove (link_id + 10, &port, 2);
      break;
    default:
      g_error (_("Unsopported ape mode in get_link_id"));
    }
  return link_id;
}                                                   /* get_link_id */


/* We update node information for each new packet that arrives in the
 * network. If the node the packet refers to is unknown, we
 * create it. */
static void
add_node_packet (const guint8 * packet,
                 packet_t * packet_info,
                 const guint8 * node_id, packet_direction direction)
{
  node_t *node;
  node = g_tree_lookup (nodes, node_id);
  if (node == NULL)
    node = create_node (packet, node_id);

  /* We add a packet to the list of packets to/from that host which we want
   * to account for */
  packet_info->ref_count++;
  node->packets = g_list_prepend (node->packets, packet_info);

  /* We update the node's protocol stack with the protocol
   * information this packet is bearing */
  add_protocol (node->protocols, packet_info->prot, phdr, NULL);

  /* We update node info */
  node->accumulated += packet_info->size;
  if (direction == INBOUND)
    node->accumulated_in += packet_info->size;
  else
    node->accumulated_out += packet_info->size;
  node->aver_accu += packet_info->size;
  if (direction == INBOUND)
    node->aver_accu_in += packet_info->size;
  else
    node->aver_accu_out += packet_info->size;
  node->last_time = now;
  node->n_packets++;

  /* If this is the first packet we've heard from the node in a while,
   * we add it to the list of new nodes so that the main app know this
   * node is active again */
  if (node->n_packets == 1)
    new_nodes = g_list_prepend (new_nodes, node);

  /* Update names list for this node */
  get_packet_names (node->protocols, packet, packet_info->size,
                                        packet_info->prot, direction);



  /* update_node (node->node_id, node, NULL); */

}                                                   /* add_node_packet */


/* Save as above plus we update protocol aggregate information */
static void
add_link_packet (const guint8 * packet, packet_t * packet_info,
                                    const guint8 * link_id)
```

```
{
  link_t *link;

  link = g_tree_lookup (links, link_id);
  if (!link)
    link = create_link (packet, link_id);

  /* We add the packet structure the list of
   * packets that this link has */
  packet_info->ref_count++;
  link->packets = g_list_prepend (link->packets, packet_info);

  /* We update the link's protocol stack with the protocol
   * information this packet is bearing */
  add_protocol (link->protocols, packet_info->prot, phdr, NULL);

  /* We update link info */
  link->accumulated += packet_info->size;
  link->last_time = now;
  link->n_packets++;

  /* update_link (link->link_id, link, NULL); */

}                                                   /* add_link_packet */

/* Allocates a new node structure, and adds it to the
 * global nodes binary tree */
static node_t *
create_node (const guint8 * packet, const guint8 * node_id)
{
  node_t *node = NULL;
  guint i = STACK_SIZE;

  node = g_malloc (sizeof (node_t));

  /* We have already allocated memory for the id when we created the
   * packet. We will use that, and will free it when the node disappears
   * and not with the packet */
#if 0
  node->node_id = g_memdup (node_id, node_id_length);
#endif
  node->node_id = node_id;

  node->name = NULL;
  node->numeric_name = NULL;
  /* TODO remove these two, shouldn't be used anymore */
  /* We initialize the ip_address, although it won't be
   * used in many cases */
  node->ip_address = 0;
  node->numeric_ip = NULL;

  node->name = g_string_new (print_mem (node_id, node_id_length));
  node->numeric_name = g_string_new (print_mem (node_id, node_id_length));

  node->average = node->average_in = node->average_out = 0;
  node->n_packets = 0;
  node->accumulated = node->accumulated_in = node->accumulated_out = 0;
  node->aver_accu = node->aver_accu_in = node->aver_accu_out = 0;

  node->packets = NULL;
  while (i + 1)
    {
      node->protocols[i] = NULL;
      node->main_prot[i] = NULL;
      i--;
    }

  g_tree_insert (nodes, node->node_id, node);          /* Add it to the main tree of nodes */

  g_log (G_LOG_DOMAIN, G_LOG_LEVEL_DEBUG,
                  _("Creating node: %s. Number of nodes %d"),
                  node->name->str, g_tree_nnodes (nodes));

  return node;
}                                                   /* create_node */


/* Allocates a new link structure, and adds it to the
 * global links binary tree */
static link_t *
create_link (const guint8 * packet, const guint8 * link_id)
{
  link_t *link;
  guint i = STACK_SIZE;
  node_t *node;

  link = g_malloc (sizeof (link_t));

  link->link_id = g_memdup (link_id, 2 * node_id_length);
  link->average = 0;
  link->n_packets = 0;
  link->accumulated = 0;
```

```
  link->packets = NULL;
  link->src_name = NULL;
  link->dst_name = NULL;
  while (i + 1)
    {
      link->protocols[i] = NULL;
      link->main_prot[i] = NULL;
      i--;
    }
  g_tree_insert (links, link->link_id, link);
  node = g_tree_lookup (nodes, link_id);
  link->src_name = g_strdup (node->name->str);
  node = g_tree_lookup (nodes, (link_id + node_id_length));
  link->dst_name = g_strdup (node->name->str);

  g_log (G_LOG_DOMAIN, G_LOG_LEVEL_DEBUG,
                  _("Creating link: %s-%s. Number of links %d"),
                  link->src_name, link->dst_name, g_tree_nnodes (links));

  return link;
}                                                   /* create_link */


/* Callback function everytime a dns_lookup function is finished */
static void
dns_ready (gpointer data, gint fd, GdkInputCondition cond)
{
  dns_ack ();
}


/* For a given protocol stack, it updates the both the global
 * protocols and specific node and link list */
void
add_protocol (GList ** protocols, const gchar * stack,
                        struct pcap_pkthdr phdr, packet_t * packet)
{
  GList *protocol_item = NULL;
  protocol_t *protocol_info = NULL;
  gchar **tokens = NULL;
  guint i = 0;
  guint8 *src_id = NULL, *dst_id = NULL;
  gchar *protocol_name;

  tokens = g_strsplit (stack, "/", 0);

  if (packet)
    {
      src_id = packet->src_id;
      dst_id = packet->dst_id;
    }

  for (; i <= STACK_SIZE; i++)
    {
      if (group_unk && strstr (tokens[i], "TCP-Port"))
                  protocol_name = "TCP-Unknown";
      else if (group_unk && strstr (tokens[i], "UDP-Port"))
                  protocol_name = "UDP-Unknown";
      else
                  protocol_name = tokens[i];

      /* If there is yet not such protocol, create it */
      if (!(protocol_item = g_list_find_custom (protocols[i],

                  protocol_name,

                  protocol_compare)))
                  {
                    protocol_info = g_malloc (sizeof (protocol_t));
                    protocol_info->name = g_strdup (protocol_name);
                    protocol_info->accumulated = 0;
                    protocol_info->aver_accu = 0;
                    protocol_info->average = 0;
                    protocol_info->n_packets = 0;
                    protocol_info->node_names = NULL;
                    protocol_info->node_ids = NULL;
                    protocol_info->packets = NULL;
                    protocols[i] = g_list_prepend (protocols[i], protocol_info);
                  }
      else
                  protocol_info = protocol_item->data;

      protocol_info->last_heard = now;
      protocol_info->accumulated += phdr.len;
      protocol_info->aver_accu += phdr.len;
      protocol_info->n_packets++;

      /* We add a packet to the list of packets that used this protocol */
      if (packet)
                  {
                    packet->ref_count++;
                    protocol_info->packets =
```

```
                g_list_prepend (protocol_info->packets, packet);
              }

    /* For protocols belonging to the global protocols list, take note of nodes that are using
     * this protocol */
    if (src_id)
              {
               if (!g_list_find (protocol_info->node_ids, src_id))
                protocol_info->node_ids =
                  g_list_prepend (protocol_info->node_ids, src_id);
              }
    if (dst_id)
              {
               if (!g_list_find (protocol_info->node_ids, src_id))
                protocol_info->node_ids =
                  g_list_prepend (protocol_info->node_ids, src_id);
              }
  }
 g_strfreev (tokens);
 tokens = NULL;
}                                                       /* add_protocol */


/* Calls update_node for every node. This is actually a function that
 shouldn't be called often, because it might take a very long time
 to complete */
void
update_nodes (void)
{
 guint n_nodes_before, n_nodes_after;

 do
   {
    n_nodes_before = g_tree_nnodes (nodes);
    g_tree_traverse (nodes, (GTraverseFunc) update_node, G_IN_ORDER, NULL);
    n_nodes_after = g_tree_nnodes (nodes);
   }
 while (n_nodes_before != n_nodes_after);

}                                                       /* update_nodes */

/* Deletes all data from a node, and possibly the node itself */
static gint
update_node (guint8 * node_id, node_t * node, gpointer pointer)
{
 struct timeval diff;
 GList *protocol_item = NULL;
 protocol_t *protocol_info = NULL;
 guint i = STACK_SIZE;

 if (node->packets)
  update_packet_list (node->packets, (gpointer) node, NODE);


 if (node->n_packets == 0)
  {

    diff = substract_times (now, node->last_time);

#if 0
    /* Delete node if we stop the capture */
    if (status == STOP)
#endif
#if 1
                /* Remove node if node is too old or if capture is stopped */
                if ((IS_OLDER (diff, node_timeout_time) && node_timeout_time)
                   || (status == STOP))
#endif
                 {

                  g_log (G_LOG_DOMAIN, G_LOG_LEVEL_DEBUG,
                                 _("Removing node: %s. Number of nodes %d"),
                                  node->name->str, g_tree_nnodes (nodes) - 1);

                  /* First thing we do is delete the node for the list of new_nodes,
                   * if it's there */
                  new_nodes = g_list_remove (new_nodes, node);

                  g_string_free (node->name, TRUE);
                  node->name = NULL;
                  g_string_free (node->numeric_name, TRUE);
                  node->numeric_name = NULL;
                  if (node->numeric_ip)
                   {
                              g_string_free (node->numeric_ip, TRUE);
                              node->numeric_ip = NULL;
                   }
                  for (; i + 1; i--)
                   if (node->main_prot[i])
                              {
                               g_free (node->main_prot[i]);
                               node->main_prot[i] = NULL;
```

```
                }
i = 0;
while (i <= STACK_SIZE)
  {

                while (node->protocols[i])
                  {
                    protocol_item = node->protocols[i];
                    protocol_info = protocol_item->data;

                    if (!protocol_info->accumulated)
                      {
                                GList *name_item = NULL;
                                name_t *name;
                                g_free (protocol_info->name);
                                protocol_info->name = NULL;

                                while (protocol_info->node_names)
                                  {
                                    name_item = protocol_info->node_names;
                                    name = name_item->data;
                                    g_free (name->node_id);
                                    g_string_free (name->name, TRUE);
                                    g_string_free (name->numeric_name, TRUE);
                                    protocol_info->node_names =
                                      g_list_remove_link (protocol_info->node_names,
                                                                                name_item);
                                    g_free (name);
                                    g_list_free (name_item);
                                  }

                                node->protocols[i] =
                                 g_list_remove_link (node->protocols[i],
                                                                        protocol_item);
                                g_free (protocol_info);
                                g_list_free (protocol_item);
                      }
                  }
                i++;
  }

g_free (node);
g_tree_remove (nodes, node_id);

/* Remove all mentions to this node in the globals protocols list */
forget_node_from_protocols (node_id);
                    g_free (node_id);
                    node = NULL;
                    return TRUE;            /* I've checked it's not safe to traverse
                                             * while deleting, so we return TRUE to stop
                                             * the traversion (Does that word exist? :-) */
                  }
                else
                  {
                   /* The packet list structure has already been freed in
                    * check_packets */
                    node->packets = NULL;
                    node->average = node->average_in = node->average_out = 0.0;
                  }
    }

  return FALSE;
}                                                       /* update_node */

/* Remove all mentions to this node in the globals protocols list */
static void
forget_node_from_protocols (guint8 * node_id)
{
 GList *protocol_item = NULL;
 protocol_t *protocol = NULL;
 guint i = 0;

 for (; i <= STACK_SIZE; i++)
   {
    protocol_item = protocols[i];
    while (protocol_item)
                {
                 protocol = protocol_item->data;
                 protocol->node_ids = g_list_remove (protocol->node_ids, node_id);
                 protocol_item = protocol_item->next;
                }
   }

}                                                       /* forget_node_from_protocols */

/* Returns a node from the list of new nodes or NULL if there are no more
 * new nodes */
node_t *
ape_get_new_node (void)
{
 node_t *node = NULL;
```

```
 GList *old_item = NULL;

 if (!new_nodes)
   return NULL;

 node = new_nodes->data;
 old_item = new_nodes;

 /* We make sure now that the node hasn't been deleted since */
 /* TODO Sometimes when I get here I have a node, but a null
  * node->node_id. What gives? */
 while (node && node->node_id && !g_tree_lookup (nodes, node->node_id))
   {
    g_my_debug
                ("Already deleted node in list of new nodes, in ape_get_new_node");

    /* Remove this node from the list of new nodes */
    new_nodes = g_list_remove_link (new_nodes, new_nodes);
    g_list_free_1 (old_item);
    if (new_nodes)
                node = new_nodes->data;
    else
                node = NULL;
    old_item = new_nodes;
   }

 if (!new_nodes)
   return NULL;

 /* Remove this node from the list of new nodes */
 new_nodes = g_list_remove_link (new_nodes, new_nodes);
 g_list_free_1 (old_item);

 return node;
}                                                   /* ape_get_new_node */


/* Calls update_link for every link. This is actually a function that
 shouldn't be called often, because it might take a very long time
 to complete */
void
update_links (void)
{
 guint n_links_before, n_links_after;

 do
   {
    n_links_before = g_tree_nnodes (links);
    g_tree_traverse (links, (GTraverseFunc) update_link, G_IN_ORDER, NULL);
    n_links_after = g_tree_nnodes (links);
   }
 while (n_links_before != n_links_after);

}                                                   /* update_links */

static gint
update_link (guint8 * link_id, link_t * link, gpointer pointer)
{
 struct timeval diff;
 guint i = STACK_SIZE;

 if (link->packets)
   update_packet_list (link->packets, (gpointer) link, LINK);

 diff = substract_times (now, link->last_time);

 if (link->n_packets == 0)
   {


#if 0
     if (status == STOP)
#endif
#if 1
                /* Remove link if it is too old or if capture is stopped */
                if ((IS_OLDER (diff, link_timeout_time) && link_timeout_time)
                    || (status == STOP))
#endif
                  {
                   for (; i + 1; i--)
                     if (link->main_prot[i])
                                {
                                 g_free (link->main_prot[i]);
                                 link->main_prot[i] = NULL;
                                }
                   g_free (link->src_name);
                   link->src_name = NULL;
                   g_free (link->dst_name);
                   link->dst_name = NULL;
                   g_free (link);
                   g_tree_remove (links, link_id);
                   g_log (G_LOG_DOMAIN, G_LOG_LEVEL_DEBUG,
```

```
                                _("Removing link. Number of links %d"),
                                g_tree_nnodes (links));

                        g_free (link_id);
                        link = NULL;

                        return TRUE;            /* I've checked it's not safe to traverse
                                                 * while deleting, so we return TRUE to stop
                                                 * the traversion (Does that word exist? :-) */
                      }
                    else
                      {
                       /* The packet list structure has already been freed in
                        * check_packets */
                       link->packets = NULL;
                       link->accumulated = 0;
                       while (i + 1)
                         {
                                link->protocols[i] = NULL;
                                i--;
                         }
                      }
  }

 return FALSE;

}                                                       /* update_link */

/* Update the values for the instantaneous traffic values of the global protocols
 * Delete them  if that's the case */
void
update_protocols (void)
{
 GList *item = NULL;
 protocol_t *protocol = NULL;
 guint i = 0;

 for (; i <= STACK_SIZE; i++)
   {
    item = protocols[i];
    while (item)
                {
                 protocol = item->data;
                 item = item->next;
                 /* If update_protocol returns false, the protocol has been deleted */
                 if (!update_protocol (protocol))
                   protocols[i] = g_list_remove (protocols[i], protocol);
                }

    /* This should be unnecesary now, shouldn't it? */
    if (status == STOP)
                {
                 g_list_free (protocols[i]);
                 protocols[i] = NULL;
                }
  }
}                                                       /* update_protocols */

/* Update the values for the instantaneous traffic values of a particular
 * protocol, which could be either one of the globals or one that belongs
 * to a single node. Returns TRUE for regular updating, and FALSE if the
 * protocol has been deleted */

gboolean
update_protocol (protocol_t * protocol)
{
 update_packet_list (protocol->packets, (gpointer) protocol, PROTOCOL);

 if (protocol->aver_accu == 0)
  protocol->packets = NULL;

 if (status == STOP)
  {
   g_list_free (protocol->node_ids);
   if (protocol->name)
                g_free (protocol->name);
   g_free (protocol);

   return (FALSE);
  }

 return TRUE;
}                                                       /* update_protocol */

/* This function is called to discard packets from the list
 * of packets beloging to a node or a link, and to calculate
 * the average traffic for that node or link */
void
update_packet_list (GList * packets, gpointer parent,
                                enum packet_belongs belongs_to)
{
 struct timeval difference;
```

```
guint i = STACK_SIZE;
node_t *node = NULL;
link_t *link = NULL;
protocol_t *protocol = NULL;
gdouble usecs_from_oldest; /* usecs since the first valid packet */
GList *packet_l_e = NULL; /* Packets is a list of packets.
                           * packet_l_e is always the latest (oldest)
                           * list element */
packet_t *packet = NULL;

if (!packets || !parent)
  return;

packet_l_e = g_list_last (packets);

/* Going from oldest to newer, delete all irrelevant packets */
while (check_packet (packet_l_e, &packet_l_e, parent, belongs_to));

/* TODO Move all this below to update_node and update_link */
/* If there still is relevant packets, then calculate average
 * traffic and update names*/

if (packet_l_e)
  {
    packet_l_e = g_list_last (packets);
    packet = (packet_t *) packet_l_e->data;

    node = (node_t *) parent;
    link = (link_t *) parent;
    protocol = (protocol_t *) parent;

    difference = substract_times (now, packet->timestamp);
    usecs_from_oldest = difference.tv_sec * 1000000 + difference.tv_usec;

    /* average in bps, so we multiply by 8 and 1000000 */
    switch (belongs_to)
                {
                case NODE:

                  node->average = 8000000 * node->aver_accu / usecs_from_oldest;
                  node->average_in = 8000000 * node->aver_accu_in / usecs_from_oldest;
                  node->average_out =
                    8000000 * node->aver_accu_out / usecs_from_oldest;
                  while (i + 1)
                    {
                      if (node->main_prot[i])
                        g_free (node->main_prot[i]);
                      node->main_prot[i]
                        = get_main_prot (packets, node->protocols, i);
                      i--;
                    }
                  update_node_names (node);
                  break;
                case LINK:
                  link->average = 8000000 * link->accumulated / usecs_from_oldest;
                  /* We look for the most used protocol for this link */
                  while (i + 1)
                    {
                      if (link->main_prot[i])
                        g_free (link->main_prot[i]);
                      link->main_prot[i]
                        = get_main_prot (packets, link->protocols, i);
                      i--;
                    }
                  break;
                case PROTOCOL:
                  protocol->average =
                    8000000 * protocol->aver_accu / usecs_from_oldest;
                  break;
                default:
                  g_my_critical
                    ("belongs_to not NODE, LINK or PROTOCOL in update_packet_list");
                }

  }
}                                                   /* update_packet_list */

/* Sets the node->name and node->numeric_name to the most used of
 * the default name for the current mode */
static void
update_node_names (node_t * node)
{
  guint i = STACK_SIZE;
  GList *protocol_item = NULL;
  protocol_t *protocol = NULL;

  /* TODO Check if it's while i or while i+1.
   * Then fix it in other places */
  while (i + 1)
    {
      if (node->protocols[i])
                {
```

```
                protocol_item = protocols[i];
                for (; protocol_item; protocol_item = protocol_item->next)
                  {
                    protocol = (protocol_t *) (protocol_item->data);
                    protocol->node_names
                            = g_list_sort (protocol->node_names, names_freq_compare);
                  }
              }
    i--;
  }

 switch (mode)
  {
  case ETHERNET:
    set_node_name (node,
                                "ETH_II,SOLVED;802.2,SOLVED;803.3,SOLVED;"
                                "NETBIOS-DGM,n;NETBIOS-SSN,n;IP,n;"
                                "IPX-SAP,n;ARP,n;" "ETH_II,n;802.2,n;802.3,n");
    break;
  case FDDI:
    set_node_name (node,
                                "FDDI,SOLVED;NETBIOS-DGM,n;NETBIOS-
SSN,n;IP,n;ARP,n;FDDI,n");
    break;
  case IEEE802:
    set_node_name (node,
                                "IEEE802,SOLVED;NETBIOS-DGM,n;NETBIOS-
SSN,n;IP,n;ARP,n;IEEE802,n");
    break;
  case IP:
    set_node_name (node, "NETBIOS-DGM,n;NETBIOS-SSN,n;IP,n");
    break;
  case TCP:
    set_node_name (node, "TCP,n");
    break;
  default:
    break;
  }
}                                                       /* update_node_names */

static void
set_node_name (node_t * node, gchar * preferences)
{
 GList *name_item = NULL, *protocol_item = NULL;
 name_t *name = NULL;
 protocol_t *protocol = NULL;
 gchar **prots, **tokens;
 guint i = 0;
 guint j = STACK_SIZE;
 gboolean cont = TRUE;

 prots = g_strsplit (preferences, ";", 0);
 for (; prots[i] && cont; i++)
  {
    for (j = STACK_SIZE; j && cont; j--)    /* We don't do level 0,
                                             * which has the topmost prot */
                {
                 tokens = g_strsplit (prots[i], ",", 0);
                 protocol_item = g_list_find_custom (node->protocols[j],
                                                                    tokens[0],
protocol_compare);
                 if (protocol_item)
                   {
                     protocol = (protocol_t *) (protocol_item->data);
                     name_item = protocol->node_names;
                     if (!strcmp (protocol->name, tokens[0]) && name_item)
                                {
                                 name = (name_t *) (name_item->data);
                                 /* If we require this protocol to be solved and it's not,
                                  * the we have to go on */
                                 if (strcmp (tokens[1], "SOLVED") || name->solved)
                                   {
                                     if (node->name)
                                                if (strcmp (node->name->str, name->name->str))
                                                  g_my_debug ("Switching node name from %s to %s",
                                                              node->name->str, name->name->str);

                                     g_string_assign (node->name, name->name->str);
                                     g_string_assign (node->numeric_name,
                                                                    name->numeric_name->str);
                                     cont = FALSE;
                                   }
                                }
                   }
                 g_strfreev (tokens);
                 tokens = NULL;

                }
  }
 g_strfreev (prots);
 prots = NULL;
}                                                       /* set_node_name */
```

```
/* Finds the most commmon protocol of all the packets in a
 * given link (only link, by now) */
static gchar *
get_main_prot (GList * packets, GList ** protocols, guint level)
{
  protocol_t *protocol;
  /* If we haven't recognized any protocol at that level,
   * we say it's unknown */
  if (!protocols[level])
    return NULL;
  protocols[level] = g_list_sort (protocols[level], prot_freq_compare);
  protocol = (protocol_t *) protocols[level]->data;
  return g_strdup (protocol->name);
}                                                   /* get_main_prot */

/* Make sure this particular packet in a list of packets beloging to
 * either o link or a node is young enough to be relevant. Else
 * remove it from the list */
/* TODO This whole function is a mess. I must take it to pieces
 * so that it is more readble and maintainable */
static gboolean
check_packet (GList * packets, GList ** packet_l_e,
                        gpointer parent, enum packet_belongs belongs_to)
{

  struct timeval result;
  double time_comparison;
  guint i = 0;
  node_t *node = NULL;
  link_t *link = NULL;
  protocol_t *protocol = NULL;
  GList *protocol_item = NULL;
  protocol_t *protocol_info = NULL;
  gchar **tokens = NULL;
  packet_t *packet = NULL;
  packet_direction direction;
  static packet_direction last_lo_direction = INBOUND;
  gchar *protocol_name = NULL;

  packet = (packet_t *) packets->data;

  if (!packet)
    {
      g_warning (_("Null packet in check_packet"));
      return FALSE;
    }

result = substract_times (now, packet->timestamp);

/* If this packet is older than the averaging time,
 * then it is removed, and the process continues.
 * Else, we are done.
 * For links, if the timeout time is smaller than the
 * averaging time, we use that instead */

if ((belongs_to == NODE) || (belongs_to == PROTOCOL))
  if (node_timeout_time)
    time_comparison = (node_timeout_time > averaging_time) ?
                averaging_time : node_timeout_time;
  else
    time_comparison = averaging_time;
else if (link_timeout_time)
  time_comparison = (link_timeout_time > averaging_time) ?
    averaging_time : link_timeout_time;
else
  time_comparison = averaging_time;

node = (node_t *) parent;
link = (link_t *) parent;
protocol = (protocol_t *) parent;

/* If this packet is too old, we discard it */
/* We also delete all packets if capture is stopped */
if (IS_OLDER (result, time_comparison) || (status == STOP))
  {

    switch (belongs_to)
                {
                case NODE:
                  /* Find the direction of the packet */
                  if (!memcmp (packet->dst_id, packet->src_id, node_id_length))
                    {
                      /* This is an evil case that can happen with the loopback
                       * device, and I can only think of a hack to try to solve
                       * it */
                      if (last_lo_direction == INBOUND)
                                direction = OUTBOUND;
                      else
                                direction = INBOUND;
                      last_lo_direction = direction;
```

```
  }
 else if (!memcmp (packet->dst_id, node->node_id, node_id_length))
  direction = INBOUND;
 else if (!memcmp (packet->src_id, node->node_id, node_id_length))
  direction = OUTBOUND;
 else
  g_my_critical ("Packet does not belong to node in check_packet!");

 /* Substract this packet's length to the accumulated */
 node->aver_accu -= packet->size;
 if (direction == INBOUND)
  node->aver_accu_in -= packet->size;
 else
  node->aver_accu_out -= packet->size;
 /* Decrease the number of packets */
 node->n_packets--;
 /* If it was the last packet in the queue, set
  * average to 0. It has to be done here because
  * otherwise average calculation in update_packet list
  * requires some packets to exist */
 if (!node->aver_accu)
  node->average = 0;

 /* We remove protocol aggregate information */
 tokens = g_strsplit (packet->prot, "/", 0);
 while ((i <= STACK_SIZE) && tokens[i])
  {
   if (group_unk && strstr (tokens[i], "TCP-Port"))
                protocol_name = "TCP-Unknown";
   else if (group_unk && strstr (tokens[i], "UDP-Port"))
                protocol_name = "UDP-Unknown";
   else
                protocol_name = tokens[i];

   protocol_item = g_list_find_custom (node->protocols[i],
protocol_name,
protocol_compare);
   if (!protocol_item)
                {
                 g_my_critical
                  ("Protocol not found while removing protocol information from
node in check_packet");
                 break;
                }
   protocol_info = protocol_item->data;
   protocol_info->accumulated -= packet->size;
   i++;
  }
 g_strfreev (tokens);
 tokens = NULL;
 break;

case LINK:
 /* See above for explanations */
 link->accumulated -= packet->size;
 link->n_packets--;
 if (!link->accumulated)
  link->average = 0;

 /* We remove protocol aggregate information */
 tokens = g_strsplit (packet->prot, "/", 0);
 while ((i <= STACK_SIZE) && tokens[i])
  {
   if (group_unk && strstr (tokens[i], "TCP-Port"))
                protocol_name = "TCP-Unknown";
   else if (group_unk && strstr (tokens[i], "UDP-Port"))
                protocol_name = "UDP-Unknown";
   else
                protocol_name = tokens[i];

   protocol_item = g_list_find_custom (link->protocols[i],
protocol_name,
protocol_compare);
   if (!protocol_item)
                {
                 g_my_critical
                  ("Protocol not found while removing protocol information from
link in check_packet");
                 break;
                }
   protocol_info = protocol_item->data;
   protocol_info->accumulated -= packet->size;

   if (!protocol_info->accumulated)
                {
                 g_free (protocol_info->name);
                 protocol_info->name = NULL;
```

```
                                link->protocols[i] =
                                  g_list_remove_link (link->protocols[i], protocol_item);
                                g_free (protocol_info);
                                g_list_free (protocol_item);
                               }
                      i++;
                     }
                    g_strfreev (tokens);
                    tokens = NULL;
                    break;
                  case PROTOCOL:
                    protocol->aver_accu -= packet->size;
                    if (!protocol->aver_accu)
                      protocol->average = 0;
                    break;
                  }

    packet->ref_count--;

    if (!packet->ref_count)
                  {
                    if (packet->prot)
                      {
                        g_free (packet->prot);
                        packet->prot = NULL;

                      }
                    /* the pointers to the src_id
                     * and dst_id are actually copies
                     * of the ones held by the nodes,
                     * and those will be freed with
                     * the node */
#if 0
                    g_free (packet->src_id);
                    g_free (packet->dst_id);
#endif
                    g_free (packet);
                    packets->data = packet = NULL;

                    n_mem_packets--;
                  }

    /* TODO I have to come back here and make sure I can't make
     * this any simpler */
    if (packets->prev)
                  {
                    GList *item;
                    packets = packets->prev;
                    item = packets->next;
                    g_list_remove_link (packets, item);
                    g_list_free_1 (item);
                    *packet_l_e = packets;
                    return (TRUE);              /* Old packet removed,
                                                 * keep on searching */
                  }
    else
                  {
                    packets->data = NULL;
                    *packet_l_e = NULL;
                    g_list_free (packets);
                    /* Last packet removed,
                     * don't search anymore
                     */
                  }
   }

  return FALSE;                                 /* Last packet searched
                                                 * End search */
}                                               /* check_packet */

/* Comparison function used to order the (GTree *) nodes
 * and canvas_nodes heard on the network */
gint
node_id_compare (gconstpointer a, gconstpointer b)
{
  int i;

  i = node_id_length - 1;

  g_assert (a != NULL);
  g_assert (b != NULL);


  while (i)
    {
      if (((guint8 *) a)[i] < ((guint8 *) b)[i])
                  {
                    return -1;
                  }
      else if (((guint8 *) a)[i] > ((guint8 *) b)[i])
                  return 1;
      i--;
```

```
    }

  return 0;
}                                                   /* node_id_compare */

/* Comparison function used to order the (GTree *) links
 * and canvas_links heard on the network */
gint
link_id_compare (gconstpointer a, gconstpointer b)
{
  int i;

  i = 2 * node_id_length - 1;

  g_return_val_if_fail (a != NULL, 1);          /* This shouldn't happen.
                                                 * We arbitrarily passing 1 to
                                                 * the comparison */
  g_return_val_if_fail (b != NULL, 1);


  while (i)
    {
      if (((guint8 *) a)[i] < ((guint8 *) b)[i])
                    {
                      return -1;
                    }
      else if (((guint8 *) a)[i] > ((guint8 *) b)[i])
                    return 1;
      i--;
    }

  return 0;
}                                                   /* link_id_compare */

/* Comparison function used to compare two link protocols */
gint
protocol_compare (gconstpointer a, gconstpointer b)
{
  g_assert (a != NULL);
  g_assert (b != NULL);

  return strcmp (((protocol_t *) a)->name, (gchar *) b);
}

/* Comparison function to sort protocols by their accumulated traffic */
static gint
prot_freq_compare (gconstpointer a, gconstpointer b)
{
  protocol_t *prot_a, *prot_b;

  g_assert (a != NULL);
  g_assert (b != NULL);

  prot_a = (protocol_t *) a;
  prot_b = (protocol_t *) b;

  if (prot_a->accumulated > prot_b->accumulated)
    return -1;
  if (prot_a->accumulated < prot_b->accumulated)
    return 1;
  return 0;
}                                                   /* prot_freq_compare */

/* Comparison function to sort protocols by their accumulated traffic */
static gint
names_freq_compare (gconstpointer a, gconstpointer b)
{
  name_t *name_a, *name_b;

  g_assert (a != NULL);
  g_assert (b != NULL);

  name_a = (name_t *) a;
  name_b = (name_t *) b;

  if (name_a->accumulated > name_b->accumulated)
    return -1;
  if (name_a->accumulated < name_b->accumulated)
    return 1;
  return 0;
}                                                   /* names_freq_compare */


/* Returns a timeval structure with the time difference between to
 * other timevals. result = a - b */
struct timeval
substract_times (struct timeval a, struct timeval b)
{
  struct timeval result;

  /* Perform the carry for the later subtraction by updating Y. */
  if (a.tv_usec < b.tv_usec)
```

```
    {
     int nsec = (b.tv_usec - a.tv_usec) / 1000000 + 1;
     b.tv_usec -= 1000000 * nsec;
     b.tv_sec += nsec;
    }
  if (a.tv_usec - b.tv_usec > 1000000)
    {
     int nsec = (a.tv_usec - b.tv_usec) / 1000000;
     b.tv_usec += 1000000 * nsec;
     b.tv_sec -= nsec;
    }

  result.tv_sec = a.tv_sec - b.tv_sec;
  result.tv_usec = a.tv_usec - b.tv_usec;

  return result;


}                                                   /* substract_times */


/* Next three functions copied directly from ethereal packet.c
 * by Gerald Combs */

/* Output has to be copied elsewhere */
/* TODO should I dump this funtion now that I have dns_lookup? */
gchar *
ip_to_str (const guint8 * ad)
{
  static gchar str[3][16];
  static gchar *cur;
  gchar *p;
  int i;
  guint32 octet;
  guint32 digit;

  if (cur == &str[0][0])
    {
     cur = &str[1][0];
    }
  else if (cur == &str[1][0])
    {
     cur = &str[2][0];
    }
  else
    {
     cur = &str[0][0];
    }
  p = &cur[16];
  *--p = '\0';
  i = 3;
  for (;;)
    {
     octet = ad[i];
     *--p = (octet % 10) + '0';
     octet /= 10;
     digit = octet % 10;
     octet /= 10;
     if (digit != 0 || octet != 0)
                *--p = digit + '0';
     if (octet != 0)
                *--p = octet + '0';
     if (i == 0)
                break;
     *--p = '.';
     i--;
    }
  return p;
}                                                   /* ip_to_str */

/* (toledo) This function I copied from capture.c of ethereal it was
 * without comments, but I believe it keeps three different
 * strings conversions in memory so as to try to make sure that
 * the conversions made will be valid in memory for a longer
 * period of time */

/* Places char punct in the string as the hex-digit separator.
 * If punct is '\0', no punctuation is applied (and thus
 * the resulting string is 5 bytes shorter)
 */

gchar *
ether_to_str_punct (const guint8 * ad, char punct)
{
  static gchar str[3][18];
  static gchar *cur;
  gchar *p;
  int i;
  guint32 octet;
  static const gchar hex_digits[16] = "0123456789abcdef";

  if (cur == &str[0][0])
```

```
    {
      cur = &str[1][0];
    }
  else if (cur == &str[1][0])
    {
      cur = &str[2][0];
    }
  else
    {
      cur = &str[0][0];
    }
  p = &cur[18];
  *--p = '\0';
  i = 5;
  for (;;)
    {
      octet = ad[i];
      *--p = hex_digits[octet & 0xF];
      octet >>= 4;
      *--p = hex_digits[octet & 0xF];
      if (i == 0)
                  break;
      if (punct)
                  *--p = punct;
      i--;
    }
  return p;
}                                                   /* ether_to_str_punct */


/* Wrapper for the most common case of asking
 * for a string using a colon as the hex-digit separator.
 */
gchar *
ether_to_str (const guint8 * ad)
{
  return ether_to_str_punct (ad, ':');
}                                                   /* ether_to_str */


gchar *
print_mem (const guint8 * ad, guint length)
{
  static gchar str[3][50];
  static gchar *cur;
  char punct = ':';
  gchar *p;
  int i;
  guint32 octet;
  static const gchar hex_digits[16] = "0123456789abcdef";

  if (cur == &str[0][0])
    {
      cur = &str[1][0];
    }
  else if (cur == &str[1][0])
    {
      cur = &str[2][0];
    }
  else
    {
      cur = &str[0][0];
    }
  p = &cur[18];
  *--p = '\0';
  i = length - 1;
  for (;;)
    {
      octet = ad[i];
      *--p = hex_digits[octet & 0xF];
      octet >>= 4;
      *--p = hex_digits[octet & 0xF];
      if (i == 0)
                  break;
      if (punct)
                  *--p = punct;
      i--;
    }
  return p;
}                                                   /* print_mem */


void
dump_node_info (node_t * node)
{

  GList *protocol_item = NULL, *name_item = NULL;
  protocol_t *protocol = NULL;
  name_t *name = NULL;
  guint i = 1;
```

```
 if (!node)
   return;

 if (node->name)
   g_my_info ("NODE %s INFORMATION", node->name->str);

 for (; i <= STACK_SIZE; i++)
   {
    if (node->protocols[i])
                 {
                  g_my_info ("Protocol level %d information", i);
                  protocol_item = node->protocols[i];
                  while (protocol_item)
                    {
                     protocol = protocol_item->data;
                     g_my_info ("\tProtocol %s", protocol->name);
                     if ((name_item = protocol->node_names))
                                 {
                                  GString *names = NULL;
                                  while (name_item)
                                    {
                                     if (!names)
                                                 names = g_string_new ("");
                                     name = name_item->data;
                                     names = g_string_append (names, name->name->str);
                                     names = g_string_append (names, " ");
                                     name_item = name_item->next;
                                    }
                                  g_my_info ("\t\tName: %s", names->str);
                                  g_string_free (names, TRUE);
                                 }
                     protocol_item = protocol_item->next;
                    }
                 }
   }

}                                                    /* dump_node_info */
```


protocols.h

```
/* Etherape
 * Copyright (C) 2000 Juan Toledo
 * $Id: protocols.h,v 1.19 2001/05/13 18:56:55 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include "prot_types.h"

#define IS_PORT(p) ( (src_service && src_service->number==p) \
                                 || (dst_service && dst_service->number==p) )
#define LINESIZE 1024

/* Enums */

enum rpc_type
{
 RPC_CALL = 0,
 RPC_REPLY = 1
};

enum rpc_program
{
 BOOTPARAMS_PROGRAM = 1,
 MOUNT_PROGRAM = 100005,
 NFS_PROGRAM = 100003,
 NLM_PROGRAM = 100021,
 PORTMAP_PROGRAM = 100000,
 STAT_PROGRAM = 100024,
 YPBIND_PROGRAM = 100007,
 YPSERV_PROGRAM = 100004,
```

```
  YPXFR_PROGRAM = 100069
};

/* Variables */

static GTree *tcp_services = NULL;
static GTree *udp_services = NULL;
static guint offset = 0;

/* These are used for conversations */
static guint32 global_src_address;
static guint32 global_dst_address;
static guint16 global_src_port;
static guint16 global_dst_port;

/* Functions declarations */

static void get_eth_type (void);
static void get_fddi_type (void);
static void get_ieee802_type (void);
static void get_eth_II (etype_t etype);
static void get_eth_802_3 (ethhdrtype_t ethhdr_type);

static void get_llc (void);
static void get_ip (void);
static void get_ipx (void);
static void get_tcp (void);
static gint tcp_compare (gconstpointer a, gconstpointer b);
static void get_udp (void);
static gint udp_compare (gconstpointer a, gconstpointer b);

static void get_netbios (void);
static void get_netbios_ssn (void);
static void get_netbios_dgm (void);

static void get_ftp (void);
static gboolean get_rpc (gboolean is_udp);
static void load_services (void);
static guint16 choose_port (guint16 a, guint16 b);
static void append_etype_prot (etype_t etype);
```


protocols.c

```
/* EtherApe
 * Copyright (C) 2001 Juan Toledo
 * $Id: protocols.c,v 1.52 2001/05/14 00:27:45 toledo Exp $
 *
 * This file is mostly a rehash of algorithms found in
 * packet-*. of ethereal
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include "globals.h"
#include "protocols.h"
#include <ctype.h>
#include <string.h>

static GString *prot;
static const guint8 *packet;
guint capture_len = 0;

gchar *
get_packet_prot (const guint8 * p, guint len)
{
  gchar **tokens = NULL;
  gchar *top_prot = NULL;
  gchar *str = NULL;

  guint i = 0;

  g_assert (p != NULL);

  if (prot)
```

```
  {
   g_string_free (prot, TRUE);
   prot = NULL;
  }
prot = g_string_new ("");

packet = p;
capture_len = len;

switch (linktype)
  {
  case L_EN10MB:
    get_eth_type ();
    break;
  case L_FDDI:
    prot = g_string_new ("FDDI");
    get_fddi_type ();
    break;
  case L_IEEE802:
    prot = g_string_append (prot, "Token Ring");
    get_ieee802_type ();
    break;
  case L_RAW:                                   /* Both for PPP and SLIP */
    prot = g_string_append (prot, "RAW/IP");
    get_ip ();
    break;
  case L_NULL:
    prot = g_string_append (prot, "NULL/IP");
    get_ip ();
    break;
  default:
    break;
  }

/* TODO I think this is a serious waste of CPU and memory.
   * I think I should look at other ways of dealing with not recognized
   * protocols other than just creating entries saying UNKNOWN */
/* I think I'll do this by changing the global protocols into
 * a GArray */

if (prot)
  {
   tokens = g_strsplit (prot->str, "/", 0);
   while ((tokens[i] != NULL) && i <= STACK_SIZE)
                i++;
   g_strfreev (tokens);
    tokens = NULL;
  }
 for (; i <= STACK_SIZE; i++)
  prot = g_string_append (prot, "/UNKNOWN");

 tokens = g_strsplit (prot->str, "/", 0);
 for (i = 0; (i <= STACK_SIZE) && strcmp (tokens[i], "UNKNOWN"); i++)
  top_prot = tokens[i];

 g_assert (top_prot != NULL);
 str = g_strdup_printf ("%s/", top_prot);
 prot = g_string_prepend (prot, str);
 g_free (str);
 str = NULL;
 g_strfreev (tokens);
 tokens = NULL;

 /* g_message ("Protocol stack is %s", prot->str); */
 return prot->str;
}                                               /* get_packet_prot */


static void
get_eth_type (void)
{
 etype_t etype;
 ethhdrtype_t ethhdr_type = ETHERNET_II;        /* Default */

 etype = pntohs (&packet[12]);


 if (etype <= IEEE_802_3_MAX_LEN)
  {

   /* Is there an 802.2 layer? I can tell by looking at the first 2
    *      bytes after the 802.3 header. If they are 0xffff, then what
    *      follows the 802.3 header is an IPX payload, meaning no 802.2.
    *      (IPX/SPX is they only thing that can be contained inside a
    *      straight 802.3 packet). A non-0xffff value means that there's an
    *      802.2 layer inside the 802.3 layer */
   if (packet[14] == 0xff && packet[15] == 0xff)
                {
                 ethhdr_type = ETHERNET_802_3;
                }
   else
                {
```

```
                ethhdr_type = ETHERNET_802_2;
              }

    /* Oh, yuck.  Cisco ISL frames require special interpretation of the
     *       destination address field; fortunately, they can be recognized by
     *       checking the first 5 octets of the destination address, which are
     *       01-00-0C-00-00 for ISL frames. */
    if (packet[0] == 0x01 && packet[1] == 0x00 && packet[2] == 0x0C
                && packet[3] == 0x00 && packet[4] == 0x00)
              {
                /* TODO Analyze ISL frames */
                prot = g_string_append (prot, "ISL");
                return;
              }
  }

  if (ethhdr_type == ETHERNET_802_3)
    {
      prot = g_string_append (prot, "802.3-RAW");
      return;
    }

  if (ethhdr_type == ETHERNET_802_2)
    {
      prot = g_string_append (prot, "802.3");
      get_eth_802_3 (ethhdr_type);
      return;
    }

  /* Else, it's ETHERNET_II */

  prot = g_string_append (prot, "ETH_II");
  get_eth_II (etype);
  return;

}                                                   /* get_eth_type */

static void
get_eth_802_3 (ethhdrtype_t ethhdr_type)
{

  offset = 14;

  switch (ethhdr_type)
    {
    case ETHERNET_802_2:
      prot = g_string_append (prot, "/LLC");
      get_llc ();
      break;
    case ETHERNET_802_3:
      get_ipx ();
      break;
    default:
    }
}                                                   /* get_eth_802_3 */

static void
get_fddi_type (void)
{
  prot = g_string_append (prot, "/LLC");
  /* Ok, this is only temporary while I truly dissect LLC
   * and fddi */
  if ((packet[19] == 0x08) && (packet[20] == 0x00))
    {
      prot = g_string_append (prot, "/IP");
      get_ip ();
    }

}                                                   /* get_fddi_type */

static void
get_ieee802_type (void)
{
  /* As with FDDI, we only support LLC by now */
  prot = g_string_append (prot, "/LLC");

  if ((packet[20] == 0x08) && (packet[21] == 0x00))
    {
      prot = g_string_append (prot, "/IP");
      get_ip ();
    }

}

static void
get_eth_II (etype_t etype)
{
  append_etype_prot (etype);

  if (etype == ETHERTYPE_IP)
    get_ip ();
  if (etype == ETHERTYPE_IPX)
```

```
    get_ipx ();

  return;
}                                                       /* get_eth_II */

static void
get_llc (void)
{
#define SAP_SNAP 0xAA
#define XDLC_I          0x00  /* Information frames */
#define XDLC_U          0x03 /* Unnumbered frames */
#define XDLC_UI         0x00/* Unnumbered Information */
#define XDLC_IS_INFORMATION(control) \
    (((control) & 0x1) == XDLC_I || (control) == (XDLC_UI|XDLC_U))

  sap_type_t dsap, ssap;
  gboolean is_snap;
  guint16 control;

  dsap = *(guint8 *) (packet + offset);
  ssap = *(guint8 *) (packet + offset + 1);

  is_snap = (dsap == SAP_SNAP) && (ssap == SAP_SNAP);

  /* SNAP not yet supported */
  if (is_snap)
    return;

  /* To get this control value is actually a lot more
   * complicated than this, see xdlc.c in ethereal,
   * but I'll try like this, it seems it works for my pourposes at
   * least most of the time */
  control = *(guint8 *) (packet + offset + 2);

  if (!XDLC_IS_INFORMATION (control))
    return;

  offset += 3;

  switch (dsap)
    {
    case SAP_NULL:
      prot = g_string_append (prot, "/LLC-NULL");
      break;
    case SAP_LLC_SLMGMT:
      prot = g_string_append (prot, "/LLC-SLMGMT");
      break;
    case SAP_SNA_PATHCTRL:
      prot = g_string_append (prot, "/PATHCTRL");
      break;
    case SAP_IP:
      prot = g_string_append (prot, "/IP");
      break;
    case SAP_SNA1:
      prot = g_string_append (prot, "/SNA1");
      break;
    case SAP_SNA2:
      prot = g_string_append (prot, "/SNA2");
      break;
    case SAP_PROWAY_NM_INIT:
      prot = g_string_append (prot, "/PROWAY-NM-INIT");
      break;
    case SAP_TI:
      prot = g_string_append (prot, "/TI");
      break;
    case SAP_BPDU:
      prot = g_string_append (prot, "/BPDU");
      break;
    case SAP_RS511:
      prot = g_string_append (prot, "/RS511");
      break;
    case SAP_X25:
      prot = g_string_append (prot, "/X25");
      break;
    case SAP_XNS:
      prot = g_string_append (prot, "/XNS");
      break;
    case SAP_NESTAR:
      prot = g_string_append (prot, "/NESTAR");
      break;
    case SAP_PROWAY_ASLM:
      prot = g_string_append (prot, "/PROWAY-ASLM");
      break;
    case SAP_ARP:
      prot = g_string_append (prot, "/ARP");
      break;
    case SAP_SNAP:
      /* We are not supposed to reach this point */
      g_warning ("Reached SNAP while checking for DSAP in get_llc");
      prot = g_string_append (prot, "/LLC-SNAP");
      break;
    case SAP_VINES1:
```

```
      prot = g_string_append (prot, "/VINES1");
      break;
    case SAP_VINES2:
      prot = g_string_append (prot, "/VINES2");
      break;
    case SAP_NETWARE:
      get_ipx ();
      break;
    case SAP_NETBIOS:
      prot = g_string_append (prot, "/NETBIOS");
      get_netbios ();
      break;
    case SAP_IBMNM:
      prot = g_string_append (prot, "/IBMNM");
      break;
    case SAP_RPL1:
      prot = g_string_append (prot, "/RPL1");
      break;
    case SAP_UB:
      prot = g_string_append (prot, "/UB");
      break;
    case SAP_RPL2:
      prot = g_string_append (prot, "/RPL2");
      break;
    case SAP_OSINL:
      prot = g_string_append (prot, "/OSINL");
      break;
    case SAP_GLOBAL:
      prot = g_string_append (prot, "/LLC-GLOBAL");
      break;
    }

}                                                   /* get_llc */

static void
get_ip (void)
{
  guint16 fragment_offset;
  iptype_t ip_type;
  ip_type = packet[l3_offset + 9];
  fragment_offset = pntohs (packet + l3_offset + 6);
  fragment_offset &= 0x0fff;

  /*This is used for conversations */
  global_src_address = pntohl (packet + l3_offset + 12);
  global_dst_address = pntohl (packet + l3_offset + 16);

  offset = l3_offset + 20;

  switch (ip_type)
    {
    case IP_PROTO_ICMP:
      prot = g_string_append (prot, "/ICMP");
      break;
    case IP_PROTO_TCP:
      if (fragment_offset)
                prot = g_string_append (prot, "/TCP_FRAGMENT");
      else
                {
                  prot = g_string_append (prot, "/TCP");
                  get_tcp ();
                }
      break;
    case IP_PROTO_UDP:
      if (fragment_offset)
                prot = g_string_append (prot, "/UDP_FRAGMENT");
      else
                {
                  prot = g_string_append (prot, "/UDP");
                  get_udp ();
                }
      break;
    case IP_PROTO_IGMP:
      prot = g_string_append (prot, "/IGMP");
      break;
    case IP_PROTO_GGP:
      prot = g_string_append (prot, "/GGP");
      break;
    case IP_PROTO_IPIP:
      prot = g_string_append (prot, "/IPIP");
      break;
#if 0                                   /* TODO How come IPIP and IPV4 share
the same number? */
    case IP_PROTO_IPV4:
      prot = g_string_append (prot, "/IPV4");
      break;
#endif
    case IP_PROTO_EGP:
      prot = g_string_append (prot, "/EGP");
      break;
    case IP_PROTO_PUP:
      prot = g_string_append (prot, "/PUP");
```

```
    break;
  case IP_PROTO_IDP:
    prot = g_string_append (prot, "/IDP");
    break;
  case IP_PROTO_TP:
    prot = g_string_append (prot, "/TP");
    break;
  case IP_PROTO_IPV6:
    prot = g_string_append (prot, "/IPV6");
    break;
  case IP_PROTO_ROUTING:
    prot = g_string_append (prot, "/ROUTING");
    break;
  case IP_PROTO_FRAGMENT:
    prot = g_string_append (prot, "/FRAGMENT");
    break;
  case IP_PROTO_RSVP:
    prot = g_string_append (prot, "/RSVP");
    break;
  case IP_PROTO_GRE:
    prot = g_string_append (prot, "/GRE");
    break;
  case IP_PROTO_ESP:
    prot = g_string_append (prot, "/ESP");
    break;
  case IP_PROTO_AH:
    prot = g_string_append (prot, "/AH");
    break;
  case IP_PROTO_ICMPV6:
    prot = g_string_append (prot, "/ICPMPV6");
    break;
  case IP_PROTO_NONE:
    prot = g_string_append (prot, "/NONE");
    break;
  case IP_PROTO_DSTOPTS:
    prot = g_string_append (prot, "/DSTOPTS");
    break;
  case IP_PROTO_VINES:
    prot = g_string_append (prot, "/VINES");
    break;
  case IP_PROTO_EIGRP:
    prot = g_string_append (prot, "/EIGRP");
    break;
  case IP_PROTO_OSPF:
    prot = g_string_append (prot, "/OSPF");
    break;
  case IP_PROTO_ENCAP:
    prot = g_string_append (prot, "/ENCAP");
    break;
  case IP_PROTO_PIM:
    prot = g_string_append (prot, "/PIM");
    break;
  case IP_PROTO_IPCOMP:
    prot = g_string_append (prot, "/IPCOMP");
    break;
  case IP_PROTO_VRRP:
    prot = g_string_append (prot, "/VRRP");
    break;
  default:
    prot = g_string_append (prot, "/IP_UNKNOWN");
  }

 return;
}

static void
get_ipx ()
{
 ipx_socket_t ipx_dsocket, ipx_ssocket;
 guint16 ipx_length;
 ipx_type_t ipx_type;

 /* Make sure this is an IPX packet */
 if ((offset + 30 > capture_len) || *(guint16 *) (packet + offset) != 0xffff)
  return;

 prot = g_string_append (prot, "/IPX");

 ipx_dsocket = pntohs (packet + offset + 16);
 ipx_ssocket = pntohs (packet + offset + 28);
 ipx_type = *(guint8 *) (packet + offset + 5);
 ipx_length = pntohs (packet + offset + 2);

 switch (ipx_type)
  {
   /* Look at the socket with these two types */
  case IPX_PACKET_TYPE_PEP:
  case IPX_PACKET_TYPE_IPX:
    break;
  case IPX_PACKET_TYPE_RIP:
    prot = g_string_append (prot, "/IPX-RIP");
    break;
```

```
  case IPX_PACKET_TYPE_ECHO:
    prot = g_string_append (prot, "/IPX-ECHO");
    break;
  case IPX_PACKET_TYPE_ERROR:
    prot = g_string_append (prot, "/IPX-ERROR");
    break;
  case IPX_PACKET_TYPE_SPX:
    prot = g_string_append (prot, "/IPX-SPX");
    break;
  case IPX_PACKET_TYPE_NCP:
    prot = g_string_append (prot, "/IPX-NCP");
    break;
  case IPX_PACKET_TYPE_WANBCAST:
    prot = g_string_append (prot, "/IPX-NetBIOS");
    break;
  }

if ((ipx_type != IPX_PACKET_TYPE_IPX) && (ipx_type != IPX_PACKET_TYPE_PEP)
    && (ipx_type != IPX_PACKET_TYPE_WANBCAST))
  return;

if ((ipx_dsocket == IPX_SOCKET_SAP) || (ipx_ssocket == IPX_SOCKET_SAP))
  prot = g_string_append (prot, "/IPX-SAP");
else if ((ipx_dsocket == IPX_SOCKET_ATTACHMATE_GW)
                    || (ipx_ssocket == IPX_SOCKET_ATTACHMATE_GW))
  prot = g_string_append (prot, "/ATTACHMATE-GW");
else if ((ipx_dsocket == IPX_SOCKET_PING_NOVELL)
                    || (ipx_ssocket == IPX_SOCKET_PING_NOVELL))
  prot = g_string_append (prot, "/PING-NOVELL");
else if ((ipx_dsocket == IPX_SOCKET_NCP) || (ipx_ssocket == IPX_SOCKET_NCP))
  prot = g_string_append (prot, "/IPX-NCP");
else if ((ipx_dsocket == IPX_SOCKET_IPXRIP)
                    || (ipx_ssocket == IPX_SOCKET_IPXRIP))
  prot = g_string_append (prot, "/IPX-RIP");
else if ((ipx_dsocket == IPX_SOCKET_NETBIOS)
                    || (ipx_ssocket == IPX_SOCKET_NETBIOS))
  prot = g_string_append (prot, "/IPX-NetBIOS");
else if ((ipx_dsocket == IPX_SOCKET_DIAGNOSTIC)
                    || (ipx_ssocket == IPX_SOCKET_DIAGNOSTIC))
  prot = g_string_append (prot, "/IPX-DIAG");
else if ((ipx_dsocket == IPX_SOCKET_SERIALIZATION)
                    || (ipx_ssocket == IPX_SOCKET_SERIALIZATION))
  prot = g_string_append (prot, "/IPX-SERIAL.");
else if ((ipx_dsocket == IPX_SOCKET_ADSM)
                    || (ipx_ssocket == IPX_SOCKET_ADSM))
  prot = g_string_append (prot, "/IPX-ADSM");
else if ((ipx_dsocket == IPX_SOCKET_EIGRP)
                    || (ipx_ssocket == IPX_SOCKET_EIGRP))
  prot = g_string_append (prot, "/EIGRP");
else if ((ipx_dsocket == IPX_SOCKET_WIDE_AREA_ROUTER)
                    || (ipx_ssocket == IPX_SOCKET_WIDE_AREA_ROUTER))
  prot = g_string_append (prot, "/IPX W.A. ROUTER");
else if ((ipx_dsocket == IPX_SOCKET_TCP_TUNNEL)
                    || (ipx_ssocket == IPX_SOCKET_TCP_TUNNEL))
  prot = g_string_append (prot, "/IPX-TCP-TUNNEL");
else if ((ipx_dsocket == IPX_SOCKET_UDP_TUNNEL)
                    || (ipx_ssocket == IPX_SOCKET_UDP_TUNNEL))
  prot = g_string_append (prot, "/IPX-UDP-TUNNEL");
else if ((ipx_dsocket == IPX_SOCKET_NWLINK_SMB_SERVER)
                    || (ipx_ssocket == IPX_SOCKET_NWLINK_SMB_SERVER)
                    || (ipx_dsocket == IPX_SOCKET_NWLINK_SMB_NAMEQUERY)
                    || (ipx_ssocket == IPX_SOCKET_NWLINK_SMB_NAMEQUERY)
                    || (ipx_dsocket == IPX_SOCKET_NWLINK_SMB_REDIR)
                    || (ipx_ssocket == IPX_SOCKET_NWLINK_SMB_REDIR)
                    || (ipx_dsocket == IPX_SOCKET_NWLINK_SMB_MAILSLOT)
                    || (ipx_ssocket == IPX_SOCKET_NWLINK_SMB_MAILSLOT)
                    || (ipx_dsocket == IPX_SOCKET_NWLINK_SMB_MESSENGER)
                    || (ipx_ssocket == IPX_SOCKET_NWLINK_SMB_MESSENGER)
                    || (ipx_dsocket == IPX_SOCKET_NWLINK_SMB_BROWSE)
                    || (ipx_ssocket == IPX_SOCKET_NWLINK_SMB_BROWSE))
  prot = g_string_append (prot, "/IPX-SMB");
else if ((ipx_dsocket == IPX_SOCKET_SNMP_AGENT)
                    || (ipx_ssocket == IPX_SOCKET_SNMP_AGENT)
                    || (ipx_dsocket == IPX_SOCKET_SNMP_SINK)
                    || (ipx_ssocket == IPX_SOCKET_SNMP_SINK))
  prot = g_string_append (prot, "/SNMP");
else
  g_my_debug ("Unknown IPX ports %d, %d", ipx_dsocket, ipx_ssocket);

}                                                   /* get_ipx */

static void
get_tcp (void)
{

  tcp_service_t *src_service, *dst_service, *chosen_service;
  tcp_type_t src_port, dst_port, chosen_port;
  guint8 th_off_x2;
  guint8 tcp_len;
  gchar *str;

  if (!tcp_services)
```

```
  load_services ();

global_src_port = src_port = pntohs (packet + offset);
global_dst_port = dst_port = pntohs (packet + offset + 2);
th_off_x2 = *(guint8 *) (packet + offset + 12);
tcp_len = hi_nibble (th_off_x2) * 4;          /* TCP header length, in bytes */

offset += tcp_len;

if (offset >= capture_len)
  return;                                     /* Continue only if there is room
                                                 * for data in the packet */

src_service = g_tree_lookup (tcp_services, &src_port);
dst_service = g_tree_lookup (tcp_services, &dst_port);

/* Check whether this packet belongs to a registered conversation */

if ((str = find_conversation (global_src_address, global_dst_address,
                                                        src_port, dst_port)))
  {
    prot = g_string_append (prot, str);
    return;
  }

/* It's not possible to know in advance whether an UDP
 * packet is an RPC packet. We'll try */
/* False means we are calling rpc from a TCP packet */
if (get_rpc (FALSE))
  return;

if (IS_PORT (TCP_NETBIOS_SSN))
  {
    get_netbios_ssn ();
    return;
  }
else if (IS_PORT (TCP_FTP))
  {
    get_ftp ();
    return;
  }

chosen_port = choose_port (src_port, dst_port);

if (!src_service && !dst_service)
  {
    prot = g_string_append (prot, "/TCP-Port ");
    str = g_strdup_printf ("%d", chosen_port);
    prot = g_string_append (prot, str);
    g_free (str);
    return;
  }

 if (!dst_service || ((src_port == chosen_port) && src_service))
   chosen_service = src_service;
 else
   chosen_service = dst_service;

#if 0
 /* Give priority to registered ports */
 if ((src_port < 1024) && (dst_port >= 1024))
   chosen_service = src_service;
 else if ((src_port >= 1024) && (dst_port < 1024))
   chosen_service = dst_service;
 else if (!dst_service)          /* Give priority to dst_service just because */
   chosen_service = src_service;
 else
   chosen_service = dst_service;
#endif

 str = g_strdup_printf ("/%s", chosen_service->name);
 prot = g_string_append (prot, str);
 g_free (str);
 str = NULL;
 return;
}                                                       /* get_tcp */

static void
get_udp (void)
{
 udp_service_t *src_service, *dst_service, *chosen_service;
 udp_type_t src_port, dst_port, chosen_port;
 gchar *str;

 if (!udp_services)
   load_services ();

 global_src_port = src_port = pntohs (packet + offset);
 global_dst_port = dst_port = pntohs (packet + offset + 2);

 offset += 8;
```

```
  /* TODO We should check up the size of the packet the same
   * way it is done in TCP */

  src_service = g_tree_lookup (udp_services, &src_port);
  dst_service = g_tree_lookup (udp_services, &dst_port);

  /* It's not possible to know in advance whether an UDP
   * packet is an RPC packet. We'll try */
  if (get_rpc (TRUE))
    return;

  if (IS_PORT (UDP_NETBIOS_NS))
    {
      get_netbios_dgm ();
      return;
    }

  chosen_port = choose_port (src_port, dst_port);

  if (!dst_service && !src_service)
    {
      prot = g_string_append (prot, "/UDP-Port ");
      str = g_strdup_printf ("%d", chosen_port);
      prot = g_string_append (prot, str);
      g_free (str);
      return;
    }

  if (!dst_service || ((src_port == chosen_port) && src_service))
    chosen_service = src_service;
  else
    chosen_service = dst_service;

#if 0
  /* Give priority to registered ports */
  if ((src_port < 1024) && (dst_port >= 1024))
    chosen_service = src_service;
  else if ((src_port >= 1024) && (dst_port < 1024))
    chosen_service = dst_service;
  else if (!dst_service)          /* Give priority to dst_service just because */
    chosen_service = src_service;
  else
    chosen_service = dst_service;
#endif

  str = g_strdup_printf ("/%s", chosen_service->name);
  prot = g_string_append (prot, str);
  g_free (str);
  str = NULL;
  return;
}                                                   /* get_udp */

static gboolean
get_rpc (gboolean is_udp)
{
  enum rpc_type msg_type;
  enum rpc_program msg_program;
  gchar *rpc_prot = NULL;

  /* Determine whether this is an RPC packet */

  if ((offset + 24) > capture_len)
    return FALSE;                           /* not big enough */

  if (is_udp)
    {
      msg_type = pntohl (packet + offset + 4);
      msg_program = pntohl (packet + offset + 12);
    }
  else
    {
      msg_type = pntohl (packet + offset + 8);
      msg_program = pntohl (packet + offset + 16);
    }

  if (msg_type != RPC_REPLY && msg_type != RPC_CALL)
    return FALSE;

  prot = g_string_append (prot, "/RPC");

  switch (msg_type)
    {
    case RPC_REPLY:
      /* RPC_REPLYs don't carry an rpc program tag */
      /* TODO In order to be able to dissect what is it's
       * protocol I'd have to keep track of who sent
       * which call */
      if (!(rpc_prot = find_conversation (global_dst_address, 0,
                                         global_dst_port, 0)))
        return FALSE;
      prot = g_string_append (prot, rpc_prot);
      return TRUE;
```

```
  case RPC_CALL:
   switch (msg_program)
                {
                case BOOTPARAMS_PROGRAM:
                  rpc_prot = g_strdup ("/BOOTPARAMS");
                  break;
                case MOUNT_PROGRAM:
                  rpc_prot = g_strdup ("/MOUNT");
                  break;
                case NLM_PROGRAM:
                  rpc_prot = g_strdup ("/NLM");
                  break;
                case PORTMAP_PROGRAM:
                  rpc_prot = g_strdup ("/PORTMAP");
                  break;
                case STAT_PROGRAM:
                  rpc_prot = g_strdup ("/STAT");
                  break;
                case NFS_PROGRAM:
                  rpc_prot = g_strdup ("/NFS");
                  break;
                case YPBIND_PROGRAM:
                  rpc_prot = g_strdup ("/YPBIND");
                  break;
                case YPSERV_PROGRAM:
                  rpc_prot = g_strdup ("/YPSERV");
                  break;
                case YPXFR_PROGRAM:
                  rpc_prot = g_strdup ("/YPXFR");
                  break;
                default:
                  return FALSE;
                }
   prot = g_string_append (prot, rpc_prot);

   /* Search for an already existing conversation, if not, create one */
   if (!find_conversation (global_src_address, 0, global_src_port, 0))
                add_conversation (global_src_address, 0,
                                                global_src_port, 0, rpc_prot);

   g_free (rpc_prot);
   return TRUE;
  default:
   return FALSE;
  }
 return FALSE;
}                                                       /* get_rpc */

/* This function is only called from a straight llc packet,
 * never from an IP packet */
void
get_netbios (void)
{
 guint16 hdr_len;

 /* Check that there is room for the minimum header */
 if (offset + 5 > capture_len)
   return;

 hdr_len = pletohs (packet + offset);

 /* If there is any data at all, it is SMB (or so I understand
  * from Ethereal's packet-netbios.c */

 if (offset + hdr_len < capture_len)
   prot = g_string_append (prot, "/SMB");

}                                                       /* get_netbios */

void
get_netbios_ssn (void)
{
#define SESSION_MESSAGE 0
 guint8 mesg_type;

 prot = g_string_append (prot, "/NETBIOS-SSN");

 mesg_type = *(guint8 *) (packet + offset);

 if (mesg_type == SESSION_MESSAGE)
   prot = g_string_append (prot, "/SMB");

 /* TODO Calculate new offset whenever we have
  * a "dissector" for a higher protocol */
 return;
}                                                       /* get_netbions_ssn */

void
get_netbios_dgm (void)
{
 guint8 mesg_type;
```

```
  prot = g_string_append (prot, "/NETBIOS-DGM");

  mesg_type = *(guint8 *) (packet + offset);

  /* Magic numbers copied from ethereal, as usual
   * They mean Direct (unique|group|broadcast) datagram */
  if (mesg_type == 0x10 || mesg_type == 0x11 || mesg_type == 0x12)
    prot = g_string_append (prot, "/SMB");

  /* TODO Calculate new offset whenever we have
   * a "dissector" for a higher protocol */
  return;
}                                                   /* get_netbios_dgm */

void
get_ftp (void)
{
  gchar *mesg = NULL;
  guint size = capture_len - offset;
  gchar *hi_str = NULL, *low_str = NULL;
  guint hi_byte, low_byte;
  guint16 server_port;
  guint i = 0;


  prot = g_string_append (prot, "/FTP");


  if ((offset + 3) > capture_len)
    return;                                     /* not big enough */

  if ((gchar) packet[offset] != '2'
      || (gchar) packet[offset + 1] != '2'
      || (gchar) packet[offset + 2] != '7')
    return;

  /* We have a passive message. Get the port */
  mesg = g_malloc (size + 1);
  memcpy (mesg, packet + offset, size);
  mesg[size] = '\0';

  g_my_debug ("Found FTP passive command: %s", mesg);

  g_my_debug ("FTP Token: %s", strtok (mesg, "(,)"));
  for (i = 1; i <= 4; i++)
    g_my_debug ("FTP Token: %s", strtok (NULL, "(,)"));

  g_my_debug ("FTP Token: %s", hi_str = strtok (NULL, "(,)"));
  if (!hi_str || !sscanf (hi_str, "%d", &hi_byte))
    return;
  g_my_debug ("FTP Token: %s", low_str = strtok (NULL, "(,)"));
  if (!low_str || !sscanf (low_str, "%d", &low_byte))
    return;

  server_port = hi_byte * 256 + low_byte;

  g_my_debug ("FTP Hi: %d. Low: %d. Passive port is %d", hi_byte, low_byte,
              server_port);

  /* A port number zero means any port */
  add_conversation (global_src_address, global_dst_address,
                    server_port, 0, "/FTP-PASSIVE");

  g_free (mesg);

  return;
}

/* Comparison function to sort tcp services port number */
static gint
tcp_compare (gconstpointer a, gconstpointer b)
{
  tcp_type_t port_a, port_b;

  port_a = *(tcp_type_t *) a;
  port_b = *(tcp_type_t *) b;

  if (port_a > port_b)
    return 1;
  if (port_a < port_b)
    return -1;
  return 0;
}                                                   /* tcp_compare */

/* Comparison function to sort udp services port number */
static gint
udp_compare (gconstpointer a, gconstpointer b)
{
  udp_type_t port_a, port_b;

  port_a = *(tcp_type_t *) a;
  port_b = *(tcp_type_t *) b;
```

```
  if (port_a > port_b)
    return 1;
  if (port_a < port_b)
    return -1;
  return 0;
}                                                   /* udp_compare */


/* TODO this is probably this single piece of code I am most ashamed of.
 * I should learn how to use flex or yacc and do this The Right Way (TM)*/
static void
load_services (void)
{
  FILE *services = NULL;
  gchar *line;
  gchar **t1 = NULL, **t2 = NULL;
  gchar *str;
  tcp_service_t *tcp_service;
  udp_service_t *udp_service;
  guint i;
  char filename[PATH_MAX];

  tcp_type_t port_number;      /* udp and tcp are the same */

  strcpy (filename, CONFDIR "/services");
  if (!(services = fopen (filename, "r")))
    {
      strcpy (filename, "/etc/services");
      if (!(services = fopen (filename, "r")))
                  {
                    g_my_critical (_
                                                   ("Failed to open %s. No TCP or UDP services will be
recognized"),
                                                   filename);
                    return;
                  }
    }

  g_my_info (_("Reading TCP and UDP services from %s"), filename);

  tcp_services = g_tree_new ((GCompareFunc) tcp_compare);
  udp_services = g_tree_new ((GCompareFunc) udp_compare);

  line = g_malloc (LINESIZE);

  while (fgets (line, LINESIZE, services))
    {
      if (line[0] != '#' && line[0] != ' ' && line[0] != '\n'
                  && line[0] != '\t')
                  {
                    gboolean error = FALSE;

                    if (!g_strdelimit (line, " \t\n", ' '))
                      error = TRUE;

                    if (error || !(t1 = g_strsplit (line, " ", 0)))
                      error = TRUE;
                    if (!error && t1[0])
                      g_strup (t1[0]);

                    for (i = 1; t1[i] && !strcmp ("", t1[i]); i++);


                    if (!error && (str = t1[i]))
                      if (!(t2 = g_strsplit (str, "/", 0)))
                        error = TRUE;

                    if (error || !t2[0])
                      error = TRUE;

                    /* TODO The h here is not portable */
                    if (!sscanf (t2[0], "%hd", &port_number) || (port_number < 1))
                      error = TRUE;

                    if (error || !t2[1])
                      error = TRUE;

                    if (error
                        || (g_strcasecmp ("udp", t2[1]) && g_strcasecmp ("tcp", t2[1])
                                    && g_strcasecmp ("ddp", t2[1])))
                      error = TRUE;

                    if (error)
                      g_warning (_("Unable to  parse line %s"), line);
                    else
                      {
#if DEBUG
                        g_my_debug ("Loading service %s %s %d", t2[1], t1[0],
                                                        port_number);
#endif
                        if (!g_strcasecmp ("tcp", t2[1]))
```

```
                {
                 tcp_service = g_malloc (sizeof (tcp_service_t));
                 tcp_service->number = port_number;
                 tcp_service->name = g_strdup (t1[0]);
                 g_tree_insert (tcp_services,
                                &(tcp_service->number), tcp_service);
                }
            else if (!g_strcasecmp ("udp", t2[1]))
                {
                 udp_service = g_malloc (sizeof (udp_service_t));
                 udp_service->number = port_number;
                 udp_service->name = g_strdup (t1[0]);
                 g_tree_insert (udp_services,
                                &(udp_service->number), udp_service);
                }
            else
                g_my_info (_("DDP protocols not supported in %s"), line);

          }

        g_strfreev (t2);
        t2 = NULL;
        g_strfreev (t1);
        t1 = NULL;

      }
  }

 g_free (line);
 line = NULL;
 fclose (services);
}                                               /* load_services */

/* Given two port numbers, it returns the
 * one that is a privileged port if the other
 * is not. If not, just returns the lower numbered */
static guint16
choose_port (guint16 a, guint16 b)
{
 guint ret;

 if ((a < 1024) && (b >= 1024))
   ret = a;
 else if ((a >= 1024) && (b < 1024))
   ret = b;
 else if (a <= b)
   ret = a;
 else
   ret = b;

 return ret;

}                                               /* choose_port */

static void
append_etype_prot (etype_t etype)
{
 switch (etype)
   {
   case ETHERTYPE_IP:
     prot = g_string_append (prot, "/IP");
     break;
   case ETHERTYPE_ARP:
     prot = g_string_append (prot, "/ARP");
     break;
   case ETHERTYPE_IPv6:
     prot = g_string_append (prot, "/IPv6");
     break;
   case ETHERTYPE_X25L3:
     prot = g_string_append (prot, "/X25L3");
     break;
   case ETHERTYPE_REVARP:
     prot = g_string_append (prot, "/REVARP");
     break;
   case ETHERTYPE_ATALK:
     prot = g_string_append (prot, "/ATALK");
     break;
   case ETHERTYPE_AARP:
     prot = g_string_append (prot, "/AARP");
     break;
   case ETHERTYPE_IPX:
     break;
   case ETHERTYPE_VINES:
     prot = g_string_append (prot, "/VINES");
     break;
   case ETHERTYPE_TRAIN:
     prot = g_string_append (prot, "/TRAIN");
     break;
   case ETHERTYPE_LOOP:
     prot = g_string_append (prot, "/LOOP");
     break;
   case ETHERTYPE_PPPOED:
```

```
    prot = g_string_append (prot, "/PPPOED");
    break;
  case ETHERTYPE_PPPOES:
    prot = g_string_append (prot, "/PPPOES");
    break;
  case ETHERTYPE_VLAN:
    prot = g_string_append (prot, "/VLAN");
    break;
  case ETHERTYPE_SNMP:
    prot = g_string_append (prot, "/SNMP");
    break;
  case ETHERTYPE_DNA_DL:
    prot = g_string_append (prot, "/DNA-DL");
    break;
  case ETHERTYPE_DNA_RC:
    prot = g_string_append (prot, "/DNA-RC");
    break;
  case ETHERTYPE_DNA_RT:
    prot = g_string_append (prot, "/DNA-RT");
    break;
  case ETHERTYPE_DEC:
    prot = g_string_append (prot, "/DEC");
    break;
  case ETHERTYPE_DEC_DIAG:
    prot = g_string_append (prot, "/DEC-DIAG");
    break;
  case ETHERTYPE_DEC_CUST:
    prot = g_string_append (prot, "/DEC-CUST");
    break;
  case ETHERTYPE_DEC_SCA:
    prot = g_string_append (prot, "/DEC-SCA");
    break;
  case ETHERTYPE_DEC_LB:
    prot = g_string_append (prot, "/DEC-LB");
    break;
  case ETHERTYPE_MPLS:
    prot = g_string_append (prot, "/MPLS");
    break;
  case ETHERTYPE_MPLS_MULTI:
    prot = g_string_append (prot, "/MPLS-MULTI");
    break;
  case ETHERTYPE_LAT:
    prot = g_string_append (prot, "/LAT");
    break;
  case ETHERTYPE_PPP:
    prot = g_string_append (prot, "/PPP");
    break;
  case ETHERTYPE_WCP:
    prot = g_string_append (prot, "/WCP");
    break;
  case ETHERTYPE_3C_NBP_DGRAM:
    prot = g_string_append (prot, "/3C-NBP-DGRAM");
    break;
  case ETHERTYPE_ETHBRIDGE:
    prot = g_string_append (prot, "/ETHBRIDGE");
    break;
  case ETHERTYPE_UNK:
    prot = g_string_append (prot, "/ETH_UNKNOWN");
  }

  return;
}                                                    /* append_etype_prot */
```

names.h

```
/* Etherape
 * Copyright (C) 2000 Juan Toledo
 * $Id: names.h,v 1.14 2001/05/14 00:27:17 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include "globals.h"
#include "dns.h"
#include "eth_resolv.h"
#include "names_netbios.h"

typedef void (p_func_t) (void);

typedef struct
{
  gchar *prot;
  p_func_t *function;
}
prot_function_t;

static const guint8 *p;
static const guint8 *id;
static guint16 offset;
static guint16 packet_size;
static guint8 level;
static guint id_length;
static packet_direction dir;
static GList **prot_list;
static name_t *name;
static gchar **tokens = NULL;
static GTree *prot_functions = NULL;
static prot_function_t *next_func = NULL;

static void get_eth_name (void);
static void get_raw_name (void);
static void get_null_name (void);
static void get_fddi_name (void);
static void get_ieee802_name (void);
static void get_llc_name (void);
static void get_arp_name (void);
static void get_ip_name (void);
static void get_ipx_name (void);
static void get_udp_name (void);
static void get_tcp_name (void);
static void get_ipxsap_name (void);
static void get_nbipx_name (void);
static void get_nbss_name (void);
static void get_nbdgm_name (void);
static void add_name (gchar * numeric, gchar * resolved, gboolean solved);
static gint id_compare (gconstpointer a, gconstpointer b);

#define KNOWN_PROTS 17

static prot_function_t prot_functions_table[KNOWN_PROTS + 1] = {
  {"ETH_II", get_eth_name},
  {"802.2", get_eth_name},
  {"802.3", get_eth_name},
  {"ISL", get_eth_name},
  {"RAW", get_raw_name},
  {"NULL", get_null_name},
  {"FDDI", get_fddi_name},
  {"Token Ring", get_ieee802_name},
  {"LLC", get_llc_name},
  {"ARP", get_arp_name},
  {"IP", get_ip_name},
  {"IPX", get_ipx_name},
  {"TCP", get_tcp_name},
  {"UDP", get_udp_name},
  {"IPX-SAP", get_ipxsap_name},
  {"IPX-NetBIOS", get_nbipx_name},
  {"NETBIOS-SSN", get_nbss_name},
  {"NETBIOS-DGM", get_nbdgm_name}
};
```

names.c

```
/* EtherApe
 * Copyright (C) 2001 Juan Toledo
 * $Id: names.c,v 1.32 2001/05/14 00:27:17 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <gnome.h>
#include <netinet/in.h>
#include "names.h"

void
get_packet_names (GList ** protocols,
                  const guint8 * packet,
                  guint16 size,
                  const gchar * prot_stack, packet_direction direction)
{
  g_assert (protocols != NULL);
  g_assert (packet != NULL);

  if (!prot_functions)
    {
      guint i = 0;
      prot_functions = g_tree_new ((GCompareFunc) strcmp);
      for (; i <= KNOWN_PROTS; i++)
        g_tree_insert (prot_functions,
                       prot_functions_table[i].prot,
                       &(prot_functions_table[i]));
    }

  p = packet;
  packet_size = size;
  dir = direction;
  prot_list = protocols;
  offset = 0;
  level = 1;                    /* Level 0 is for the topmost. Not valid */
  tokens = g_strsplit (prot_stack, "/", 0);

  next_func = g_tree_lookup (prot_functions, tokens[level]);
  if (next_func)
    next_func->function ();

  g_strfreev (tokens);
  tokens = NULL;
}                               /* get_packet_names */


/* Raw is used for ppp and slip. There is actually no information,
 * so we just jump to the next protocol */
static void
get_raw_name (void)
{

  level++;

  next_func = g_tree_lookup (prot_functions, tokens[level]);
  if (next_func)
    next_func->function ();

}                               /* get_raw_name */

/* Null is used for loopback. There is actually no information,
 * so we just jump to the next protocol */
/* TODO Are we so sure there is actually no information?
 * Then what are those four bytes? */
static void
get_null_name (void)
{

  level++;
  offset += 4;

  next_func = g_tree_lookup (prot_functions, tokens[level]);
  if (next_func)
    next_func->function ();

}                               /* get_null_name */
```

```
static void
get_eth_name (void)
{
  gchar *numeric = NULL, *solved = NULL;
  gboolean found_in_ethers = FALSE;

  if (dir == INBOUND)
    id = p + offset;
  else
    id = p + offset + 6;

  id_length = 6;

  numeric = ether_to_str (id);
  solved = get_ether_name (id);

  /* get_ether_name will return an ethernet address with
   * the first three numbers substituted with the manufacter
   * if it cannot find an /etc/ethers entry. If it is so,
   * then the last 8 characters (for example ab:cd:ef) will
   * be the same, and we will note that the name hasn't
   * been solved */

  if (numeric && solved)
    found_in_ethers = strcmp (numeric + strlen (numeric) - 8,
                              solved + strlen (solved) - 8);

  if (found_in_ethers)
    add_name (numeric, solved, TRUE);
  else
    add_name (numeric, solved, FALSE);

  level++;
  offset += 14;

  next_func = g_tree_lookup (prot_functions, tokens[level]);
  if (next_func)
    next_func->function ();

}                                               /* get_eth_name */

static void
get_ieee802_name (void)
{
  gchar *numeric = NULL, *solved = NULL;
  gboolean found_in_ethers = FALSE;

  if (dir == INBOUND)
    id = p + offset + 2;
  else
    id = p + offset + 8;

  id_length = 6;

  numeric = ether_to_str (id);
  solved = get_ether_name (id);

  /* get_ether_name will return an ethernet address with
   * the first three numbers substituted with the manufacter
   * if it cannot find an /etc/ethers entry. If it is so,
   * then the last 8 characters (for example ab:cd:ef) will
   * be the same, and we will note that the name hasn't
   * been solved */

  if (numeric && solved)
    found_in_ethers = strcmp (numeric + strlen (numeric) - 8,
                              solved + strlen (solved) - 8);

  if (found_in_ethers)
    add_name (numeric, solved, TRUE);
  else
    add_name (numeric, solved, FALSE);

  level++;
  offset += 14;

  next_func = g_tree_lookup (prot_functions, tokens[level]);
  if (next_func)
    next_func->function ();

}                                               /* get_ieee802_name */

static void
get_fddi_name (void)
{
  gchar *numeric = NULL, *solved = NULL;
  gboolean found_in_ethers = FALSE;

  if (dir == INBOUND)
    id = p + offset + 1;
```

```
  else
    id = p + offset + 7;

  id_length = 6;

  numeric = ether_to_str (id);
  solved = get_ether_name (id);

  /* get_ether_name will return an ethernet address with
   * the first three numbers substituted with the manufacter
   * if it cannot find an /etc/ethers entry. If it is so,
   * then the last 8 characters (for example ab:cd:ef) will
   * be the same, and we will note that the name hasn't
   * been solved */

  if (numeric && solved)
    found_in_ethers = strcmp (numeric + strlen (numeric) - 8,
                              solved + strlen (solved) - 8);

  if (found_in_ethers)
    add_name (numeric, solved, TRUE);
  else
    add_name (numeric, solved, FALSE);

  level++;
  offset += 13;

  next_func = g_tree_lookup (prot_functions, tokens[level]);
  if (next_func)
    next_func->function ();

}                                               /* get_fddi_name */

/* LLC is the only supported FDDI link layer type */
static void
get_llc_name (void)
{

  level++;
  /* TODO IMPORTANT
   * We must decode the llc header to calculate the offset
   * We are just doing assumptions by now */

  if ((linktype == L_FDDI) || (linktype == L_IEEE802))
    offset += 8;
  else if (linktype == L_EN10MB)
    offset += 3;
  else
    return;

  next_func = g_tree_lookup (prot_functions, tokens[level]);
  if (next_func)
    next_func->function ();

}                                               /* get_llc_name */

static void
get_arp_name (void)
{
  guint16 protocol_type;
  guint8 hardware_len, protocol_len;
  const guint8 *id;
#define ARPTYPE_IP 0x0800

  /* We can only tell the IP address of the asking node.
   * Most of the times the callee will be the broadcast
   * address */
  if (dir == INBOUND)
    return;

  /* We only know about IP ARP queries */
  protocol_type = pntohs ((p + offset + 2));
  if (protocol_type != ARPTYPE_IP)
    return;

  hardware_len = *(guint8 *) (p + offset + 4);
  protocol_len = *(guint8 *) (p + offset + 5);

  id = p + offset + 8 + hardware_len;

  if (mode == ETHERNET)
    add_name (ip_to_str (id), dns_lookup (pntohl (id), FALSE), TRUE);
  else
    add_name (ip_to_str (id), dns_lookup (pntohl (id), TRUE), TRUE);


  /* ARP doesn't carry any other protocol on top, so we return
   * directly */
}                                               /* get_arp_name */

static void
get_ip_name (void)
```

```
{

 if (dir == INBOUND)
   id = p + offset + 16;
 else
   id = p + offset + 12;

 id_length = 4;

 if (numeric)
   add_name (ip_to_str (id), ip_to_str (id), FALSE);
 else
   {
    if (mode == ETHERNET)
                add_name (ip_to_str (id), dns_lookup (pntohl (id), FALSE), TRUE);
    else
                add_name (ip_to_str (id), dns_lookup (pntohl (id), TRUE), TRUE);
   }

 /* TODO I don't like the fact that the gdk_input for dns.c is not
  * called in this function, because it means that it can't be used
  * as a library */

 level++;
 offset += 20;

 next_func = g_tree_lookup (prot_functions, tokens[level]);

 if (next_func)
   next_func->function ();

}                                                       /* get_ip_name */

static void
get_ipx_name (void)
{
 level++;
 offset += 30;

 next_func = g_tree_lookup (prot_functions, tokens[level]);
 if (next_func)
   next_func->function ();
}

static void
get_tcp_name (void)
{
 guint8 *id_buffer;
 guint8 th_off_x2;
 guint8 tcp_len;
 guint16 port;
 GString *numeric_name, *resolved_name;
 gchar *str;


 id_length = 6;

 id_buffer = g_malloc (id_length);

 if (dir == OUTBOUND)
   {
    g_memmove (id_buffer, p + offset - 8, 4);
    port = ntohs (*(guint16 *) (p + offset));
    g_memmove (id_buffer + 4, &port, 2);
   }
 else
   {
    g_memmove (id_buffer, p + offset - 4, 4);
    port = ntohs (*(guint16 *) (p + offset + 2));
    g_memmove (id_buffer + 4, &port, 2);
   }

 /* TODO I should substitute these for a g_snprintf
  * and save myself so many allocations */

 numeric_name = g_string_new (ip_to_str (id_buffer));
 numeric_name = g_string_append_c (numeric_name, ':');
 str = g_strdup_printf ("%d", *(guint16 *) (id_buffer + 4));
 numeric_name = g_string_append (numeric_name, str);
 g_free (str);
 str = NULL;

 resolved_name = g_string_new (dns_lookup (pntohl (id_buffer), TRUE));
 resolved_name = g_string_append_c (resolved_name, ':');
 resolved_name = g_string_append (resolved_name,
                                                  get_tcp_port (*(guint16 *)
                                                                (id_buffer +
4)));

 id = id_buffer;
 add_name (numeric_name->str, resolved_name->str, TRUE);
```

```
  g_free (id_buffer);
  id_buffer = NULL;
  g_string_free (numeric_name, TRUE);
  numeric_name = NULL;
  g_string_free (resolved_name, TRUE);
  resolved_name = NULL;

  level++;

  th_off_x2 = *(guint8 *) (p + offset + 12);
  tcp_len = hi_nibble (th_off_x2) * 4;          /* TCP header length, in bytes */
  offset += tcp_len;

  next_func = g_tree_lookup (prot_functions, tokens[level]);
  if (next_func)
    next_func->function ();

}                                                       /* get_tcp_name */


/* TODO I still have to properly implement this. Right now it's just
 * a placeholder to get to the UDP/NETBIOS-DGM */
static void
get_udp_name (void)
{

  level++;
  offset += 8;

  next_func = g_tree_lookup (prot_functions, tokens[level]);
  if (next_func)
    next_func->function ();

}                                                       /* get_udp_name */


/* TODO SET UP THE id's FOR THIS NETBIOS NAME FUNCTIONS */
static void
get_ipxsap_name (void)
{
  guint16 sap_type;
  gchar *name;

  sap_type = pntohs (p + offset);

  /* we want responses */
  if (sap_type != 0x0002)
    return;

  name = (gchar *) (p + offset + 4);

  g_my_debug ("Sap name %s found", name);

  add_name (name, name, TRUE);

}                                                       /* get_ipxsap_name */

static void
get_nbipx_name (void)
{

}                                                       /* get_ipxsap_name */


static void
get_nbss_name (void)
{
  /* TODO this really shouldn't be definened both here and in
   * names_netbios.h */
#define SESSION_REQUEST 0x81
#define MAXDNAME        1025            /* maximum domain name length */
#define NETBIOS_NAME_LEN  16

  guint i = 0;
  guint8 mesg_type;
  guint16 length;
  gchar *numeric_name = NULL;
  gchar name[(NETBIOS_NAME_LEN - 1) * 4 + MAXDNAME];
  guint name_len;
  int name_type;                /* TODO I hate to use an int here, while I have been
                                   * using glib data types all the time. But I just don't
                                   * know what it might mean in the ethereal implementation */


  mesg_type = *(guint8 *) (p + offset);
  length = pntohs ((p + offset + 2));

  offset += 4;

  if (mesg_type == SESSION_REQUEST)
    {
      name_len = ethereal_nbns_name (p, offset, offset, name, &name_type);
```

```
    if (dir == OUTBOUND)
                ethereal_nbns_name (p, offset + name_len, offset + name_len, name,
                                                    &name_type);


    /* We just want the name, not the space padding behind it */

    for (; i <= (NETBIOS_NAME_LEN - 2) && name[i] != ' '; i++);
    name[i] = '\0';

    /* Many packages will be straight TCP packages directed to the proper
     * port which first byte happens to be SESSION_REQUEST. In those cases
     * the name will be illegal, and we will not add it */
    if (strcmp (name, "Illegal"))
                {
                 numeric_name =
                  g_strdup_printf ("%s %s (%s)", name, name + NETBIOS_NAME_LEN - 1,
                                                    get_netbios_host_type (name_type));

                 add_name (numeric_name, name, TRUE);
                 g_free (numeric_name);
                }

  }


  level++;
  offset += length;

  next_func = g_tree_lookup (prot_functions, tokens[level]);
  if (next_func)
    next_func->function ();
}                                                       /* get_nbss_name */

static void
get_nbdgm_name (void)
{
  guint8 mesg_type;
  gchar *numeric_name = NULL;
  gchar name[(NETBIOS_NAME_LEN - 1) * 4 + MAXDNAME];
  gboolean name_found = FALSE;
  int name_type;
  int len;
  guint i = 0;

  mesg_type = *(guint8 *) (p + offset);

  offset += 10;

  /* Magic numbers copied from ethereal, as usual
   * They mean Direct (unique|group|broadcast) datagram */
  if (mesg_type == 0x10 || mesg_type == 0x11 || mesg_type == 0x12)
    {
     offset += 4;
     len = ethereal_nbns_name (p, offset, offset, name, &name_type);

     if (dir == INBOUND)
                ethereal_nbns_name (p, offset + len, offset + len, name, &name_type);
     name_found = TRUE;

    }
  else if (mesg_type == 0x14 || mesg_type == 0x15 || mesg_type == 0x16)
    {
     if (dir == INBOUND)
                {
                 len = ethereal_nbns_name (p, offset, offset, name, &name_type);
                 name_found = TRUE;
                }
    }

  /* We just want the name, not the space padding behind it */

  for (; i <= (NETBIOS_NAME_LEN - 2) && name[i] != ' '; i++);
  name[i] = '\0';

  /* The reasing here might not make sense as in the TCP case, but it
   * doesn't hurt to check that we don't have an illegal name anyhow */
  if (strcmp (name, "Illegal") && name_found)
    {
     numeric_name =
                g_strdup_printf ("%s %s (%s)", name, name + NETBIOS_NAME_LEN - 1,
                                                    get_netbios_host_type (name_type));

     add_name (numeric_name, name, TRUE);
     g_free (numeric_name);
    }


}                                                       /* get_nbdgm_name */
```

```
static void
add_name (gchar * numeric_name, gchar * resolved_name, gboolean solved)
{
 GList *protocol_item = NULL;
 protocol_t *protocol = NULL;
 GList *name_item = NULL;

 /* Find the protocol entry */
 protocol_item = g_list_find_custom (prot_list[level],
                                     tokens[level], protocol_compare);
 protocol = (protocol_t *) (protocol_item->data);
 name_item = protocol->node_names;

 /* Have we heard this address before? */
 name_item = g_list_find_custom (name_item, id, id_compare);
 if (name_item)
   name = name_item->data;
 else
   name = NULL;

 if (!name)                                 /* First time name */
   {
    name = g_malloc (sizeof (name_t));
    name->node_id = g_memdup (id, id_length);
    name->n_packets = 0;
    name->accumulated = 0;
    name->numeric_name = NULL;
    name->name = NULL;
    protocol->node_names = g_list_prepend (protocol->node_names, name);
   }

 if (!name->numeric_name)
   name->numeric_name = g_string_new (numeric_name);
 else
   g_string_assign (name->numeric_name, numeric_name);

 if (!name->name)
   {
    if (numeric)
                name->name = g_string_new (numeric_name);
    else
                name->name = g_string_new (resolved_name);
   }
 else
   {
    if (numeric)
                g_string_assign (name->name, numeric_name);
    else
                g_string_assign (name->name, resolved_name);
   }

 if (numeric)
   name->solved = FALSE;
 else
   name->solved = solved;

 name->n_packets++;
 name->accumulated += packet_size;

}                                                       /* add_name */

/* Comparison function used to compare two node ids */
static gint
id_compare (gconstpointer a, gconstpointer b)
{
 g_assert (a != NULL);
 g_assert (b != NULL);

 return memcmp (((name_t *) a)->node_id, (guint8 *) b, id_length);
}                                                       /* id_compare */
```

diagram.h

```
/* Etherape
 * Copyright (C) 2000 Juan Toledo
 * $Id: diagram.h,v 1.25 2001/05/07 21:50:04 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include "globals.h"


static gboolean is_idle = FALSE;
static guint displayed_nodes;

static guint prot_color_index = 0;
static guint known_protocols[STACK_SIZE + 1];

struct popup_data
{
  GtkWidget *node_popup;
  canvas_node_t *canvas_node;
};

/* Function definitions */

static void check_new_protocol (protocol_t * protocol, GtkWidget * canvas);
static gint check_new_node (guint8 * ether_addr,
                            node_t * node, GtkWidget * canvas);
static gint update_canvas_nodes (guint8 * ether_addr,
                                 canvas_node_t * canvas_node,
                                 GtkWidget * canvas);
static gboolean display_node (node_t * node);
static void limit_nodes (void);
static gint add_ordered_node (guint8 * node_id,
                              canvas_node_t * canvas_node,
                              GTree * ordered_nodes);
static gint check_ordered_node (gdouble * traffic, canvas_node_t * node,
                                guint * count);
static gint traffic_compare (gconstpointer a, gconstpointer b);
static gint reposition_canvas_nodes (guint8 * ether_addr,
                                     canvas_node_t * canvas_node,
                                     GtkWidget * canvas);
static gint check_new_link (guint8 * ether_link,
                            link_t * link, GtkWidget * canvas);
static gint update_canvas_links (guint8 * ether_link,
                                 canvas_link_t * canvas_link,
                                 GtkWidget * canvas);
static gchar *get_prot_color (gchar * name);
static gdouble get_node_size (gdouble average);
static gdouble get_link_size (gdouble average);
static gint link_item_event (GnomeCanvasItem * item,
                             GdkEvent * event, canvas_link_t * canvas_link);
static gint node_item_event (GnomeCanvasItem * item,
                             GdkEvent * event, canvas_node_t * canvas_node);
#if 0
static guint popup_to (struct popup_data *pd);
#endif
```

diagram.c

```
/* EtherApe
 * Copyright (C) 2001 Juan Toledo
 * $Id: diagram.c,v 1.112 2001/06/22 08:41:08 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <gnome.h>

#include "math.h"

#include "globals.h"
#include "diagram.h"
#include "preferences.h"

/* It updates controls from values of variables, and connects control
 * signals to callback functions */
void
init_diagram ()
{
  GtkWidget *widget;
  GtkSpinButton *spin;
  GtkStyle *style;
  GtkWidget *canvas;
  GdkColor color;

  /* Creates trees */
  canvas_nodes = g_tree_new (node_id_compare);
  canvas_links = g_tree_new (link_id_compare);


  /* Updates controls from values of variables */
  widget = glade_xml_get_widget (xml, "node_radius_slider");
  gtk_adjustment_set_value (GTK_RANGE (widget)->adjustment,
                            log (node_radius_multiplier) / log (10));
  gtk_signal_emit_by_name (GTK_OBJECT (GTK_RANGE (widget)->adjustment),
                           "changed");
  widget = glade_xml_get_widget (xml, "link_width_slider");
  gtk_adjustment_set_value (GTK_RANGE (widget)->adjustment,
                            log (link_width_multiplier) / log (10));
  gtk_signal_emit_by_name (GTK_OBJECT (GTK_RANGE (widget)->adjustment),
                           "changed");
  spin = GTK_SPIN_BUTTON (glade_xml_get_widget (xml, "averaging_spin"));
  gtk_spin_button_set_value (spin, averaging_time);
  spin = GTK_SPIN_BUTTON (glade_xml_get_widget (xml, "refresh_spin"));
  gtk_spin_button_set_value (spin, refresh_period);
  spin = GTK_SPIN_BUTTON (glade_xml_get_widget (xml, "gui_node_to_spin"));
  gtk_spin_button_set_value (spin, gui_node_timeout_time);
  spin = GTK_SPIN_BUTTON (glade_xml_get_widget (xml, "node_to_spin"));
  gtk_spin_button_set_value (spin, node_timeout_time);
  spin = GTK_SPIN_BUTTON (glade_xml_get_widget (xml, "link_to_spin"));
  gtk_spin_button_set_value (spin, link_timeout_time);

  widget = glade_xml_get_widget (xml, "diagram_only_toggle");
  gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (widget), diagram_only);
  widget = glade_xml_get_widget (xml, "group_unk_check");
  gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (widget), group_unk);

  widget = glade_xml_get_widget (xml, "size_mode_menu");
  gtk_option_menu_set_history (GTK_OPTION_MENU (widget), size_mode);
  widget = glade_xml_get_widget (xml, "node_size_optionmenu");
  gtk_option_menu_set_history (GTK_OPTION_MENU (widget), node_size_variable);
  widget = glade_xml_get_widget (xml, "stack_level_menu");
  gtk_option_menu_set_history (GTK_OPTION_MENU (widget), stack_level);
  widget = glade_xml_get_widget (xml, "filter_gnome_entry");
  gnome_entry_load_history (GNOME_ENTRY (widget));
  widget = glade_xml_get_widget (xml, "file_filter_entry");
  gnome_entry_load_history (GNOME_ENTRY (widget));
  widget = glade_xml_get_widget (xml, "fileentry");
  widget = gnome_file_entry_gnome_entry (GNOME_FILE_ENTRY (widget));
  gnome_entry_load_history (GNOME_ENTRY (widget));

  /* Connects signals */
```

```
  widget = glade_xml_get_widget (xml, "node_radius_slider");
  gtk_signal_connect (GTK_OBJECT (GTK_RANGE (widget)->adjustment),
                      "value_changed",
                      GTK_SIGNAL_FUNC
                      (on_node_radius_slider_adjustment_changed), NULL);
  widget = glade_xml_get_widget (xml, "link_width_slider");
  gtk_signal_connect (GTK_OBJECT (GTK_RANGE (widget)->adjustment),
                      "value_changed",
                      GTK_SIGNAL_FUNC
                      (on_link_width_slider_adjustment_changed), NULL);
  widget = glade_xml_get_widget (xml, "averaging_spin");
  gtk_signal_connect (GTK_OBJECT (GTK_SPIN_BUTTON (widget)->adjustment),
                      "value_changed",
                      GTK_SIGNAL_FUNC
                      (on_averaging_spin_adjustment_changed), NULL);
  widget = glade_xml_get_widget (xml, "refresh_spin");
  gtk_signal_connect (GTK_OBJECT (GTK_SPIN_BUTTON (widget)->adjustment),
                      "value_changed",
                      GTK_SIGNAL_FUNC
                      (on_refresh_spin_adjustment_changed),
                      glade_xml_get_widget (xml, "canvas1"));
  widget = glade_xml_get_widget (xml, "node_to_spin");
  gtk_signal_connect (GTK_OBJECT (GTK_SPIN_BUTTON (widget)->adjustment),
                      "value_changed",
                      GTK_SIGNAL_FUNC
                      (on_node_to_spin_adjustment_changed), NULL);
  widget = glade_xml_get_widget (xml, "gui_node_to_spin");
  gtk_signal_connect (GTK_OBJECT (GTK_SPIN_BUTTON (widget)->adjustment),
                      "value_changed",
                      GTK_SIGNAL_FUNC
                      (on_gui_node_to_spin_adjustment_changed), NULL);
  widget = glade_xml_get_widget (xml, "link_to_spin");
  gtk_signal_connect (GTK_OBJECT (GTK_SPIN_BUTTON (widget)->adjustment),
                      "value_changed",
                      GTK_SIGNAL_FUNC
                      (on_link_to_spin_adjustment_changed), NULL);
  widget = glade_xml_get_widget (xml, "size_mode_menu");
  gtk_signal_connect (GTK_OBJECT (GTK_OPTION_MENU (widget)->menu),
                      "deactivate",
                      GTK_SIGNAL_FUNC (on_size_mode_menu_selected), NULL);
  widget = glade_xml_get_widget (xml, "node_size_optionmenu");
  gtk_signal_connect (GTK_OBJECT (GTK_OPTION_MENU (widget)->menu),
                      "deactivate",
                      GTK_SIGNAL_FUNC (on_node_size_optionmenu_selected),
                      NULL);
  widget = glade_xml_get_widget (xml, "stack_level_menu");
  gtk_signal_connect (GTK_OBJECT (GTK_OPTION_MENU (widget)->menu),
                                  "deactivate",
                                  GTK_SIGNAL_FUNC (on_stack_level_menu_selected), NULL);

  /* Sets canvas background to black */
  canvas = glade_xml_get_widget (xml, "canvas1");
  gdk_color_parse ("black", &color);
  gdk_colormap_alloc_color (gtk_widget_get_colormap (canvas), &color, TRUE,
                                                      TRUE);
  style = gtk_style_new ();
  style->bg[GTK_STATE_NORMAL] = color;
  gtk_widget_set_style (canvas, style);

  gtk_style_set_background (canvas->style, canvas->window, GTK_STATE_NORMAL);

  /* Initialize the known_protocols table */
  delete_gui_protocols ();

  /* Set the already_updating global flag */
  already_updating = FALSE;
}                                                   /* init_diagram */


void
destroying_timeout (gpointer data)
{
/* g_message ("A timeout function has been destroyed"); */
  diagram_timeout = g_idle_add_full (G_PRIORITY_DEFAULT,
                                     (GtkFunction) update_diagram,
                                     data, (GDestroyNotify) destroying_idle);
  is_idle = TRUE;
}

void
destroying_idle (gpointer data)
{
/*   g_message ("An idle function has been destroyed"); */
  diagram_timeout = g_timeout_add_full (G_PRIORITY_DEFAULT,
                                        refresh_period,
                                        (GtkFunction) update_diagram,
                                        data,
                                        (GDestroyNotify) destroying_timeout);
  is_idle = FALSE;
}
```

```
/* Refreshes the diagram. Called each refresh_period ms
 * 1. Checks for new protocols and displays them
 * 2. Updates nodes looks
 * 3. Updates links looks
 */
guint
update_diagram (GtkWidget * canvas)
{
  GString *status_string = NULL;
  guint n_links = 0, n_links_new = 1;
  guint n_nodes_before = 0, n_nodes_after = 1;
  static struct timeval last_time = { 0, 0 }, diff;
  guint32 diff_msecs;
  node_t *new_node = NULL;

  if (status == PAUSE)
    return FALSE;

  if (end_of_file && status != STOP)
    gui_pause_capture ();

  /*
   * It could happen that during an intensive calculation, in order
   * to update the GUI and make the application responsive gtk_main_iteration
   * is called. But that could also trigger this very function's timeout.
   * If we let it run twice many problems could come up. Thus,
   * we are preventing it with the already_updating variable
   */

  if (already_updating)
    {
      g_my_debug ("update_diagram called while already updating");
      return FALSE;
    }

  already_updating = TRUE;


  gettimeofday (&now, NULL);


  /* We search for new protocols */
  while (known_protocols[stack_level] !=
                    g_list_length (protocols[stack_level]))
    g_list_foreach (protocols[stack_level], (GFunc) check_new_protocol,
                                        canvas);
```

```
  /* Deletes all nodes and updates traffic values */
  /* TODO To reduce CPU usage, I could just as well update each specific
   * node in update_canvas_nodes and create a new timeout function that would
   * make sure that old nodes get deleted by calling update_nodes, but
   * not as often as with diagram_refresh_period */
  update_nodes ();

  /* Check if there are any new nodes */
#if 0
  g_tree_traverse (nodes, (GTraverseFunc) check_new_node, G_IN_ORDER, canvas);
#endif
  while ((new_node = ape_get_new_node ()))
    check_new_node (new_node->node_id, new_node, canvas);

  /* Update nodes look and delete outdated canvas_nodes */
  do
    {
      n_nodes_before = g_tree_nnodes (canvas_nodes);
      g_tree_traverse (canvas_nodes,
                       (GTraverseFunc) update_canvas_nodes,
                       G_IN_ORDER, canvas);
      n_nodes_after = g_tree_nnodes (canvas_nodes);
    }
  while (n_nodes_before != n_nodes_after);

  /* Limit the number of nodes displayed, if a limit has been set */
  /* TODO check whether this is the right function to use, now that we have a more
   * general display_node called in update_canvas_nodes */
  limit_nodes ();

  /* Reposition canvas_nodes */

  if (need_reposition)
    {
      g_tree_traverse (canvas_nodes,
                       (GTraverseFunc) reposition_canvas_nodes,
                       G_IN_ORDER, canvas);
      need_reposition = 0;
    }


  /* Delete old capture links and update capture link variables */
  update_links ();

  /* Check if there are any new links */
```

```
g_tree_traverse (links, (GTraverseFunc) check_new_link, G_IN_ORDER, canvas);


/* Update links look
 * We also delete timedout links, and when we do that we stop
 * traversing, so we need to go on until we have finished updating */

do
  {
   n_links = g_tree_nnodes (canvas_links);
   g_tree_traverse (canvas_links,
                                    (GTraverseFunc) update_canvas_links,
                                    G_IN_ORDER, canvas);
   n_links_new = g_tree_nnodes (canvas_links);
  }
while (n_links != n_links_new);

/* Update protocol information */
update_protocols ();

/* With this we make sure that we don't overload the
 * CPU with redraws */

if ((last_time.tv_sec == 0) && (last_time.tv_usec == 0))
  last_time = now;

/* Force redraw */
while (gtk_events_pending ())
  gtk_main_iteration ();

gettimeofday (&now, NULL);
diff = substract_times (now, last_time);
diff_msecs = diff.tv_sec * 1000 + diff.tv_usec / 1000;
last_time = now;

status_string = g_string_new (_("Number of nodes: "));
g_string_sprintfa (status_string, "%d", n_nodes_after);

g_string_sprintfa (status_string,
                                    _(". Refresh Period: %d"), (int) diff_msecs);

g_string_sprintfa (status_string,
                                    ". Total Packets %g, packets in memory: %g", n_packets,
                                    n_mem_packets);
if (is_idle)
  status_string = g_string_append (status_string, _(". IDLE."));
  else
    status_string = g_string_append (status_string, _(". TIMEOUT."));
  g_my_debug (status_string->str);
  g_string_free (status_string, TRUE);
  status_string = NULL;

#if 0
  g_message ("Total Packets %g, packets in memory: %g", n_packets,
                        n_mem_packets);
#endif

  already_updating = FALSE;

  if (!is_idle)
    {
      if (diff_msecs > refresh_period * 1.2)
                    {
/*                    g_message ("Timeout about to be removed"); */
                      return FALSE;                 /* Removes the timeout */
                    }
    }
  else
    {
      if (diff_msecs < refresh_period)
                    {
/*                    g_message ("Idle about to be removed"); */
                      return FALSE;                 /* removes the idle */
                    }
    }

  return TRUE;                                      /* Keep on calling this function */

}                                                                       /* update_diagram */


/* Checks whether there is already a legend entry for each known
 * protocol. If not, create it */
static void
check_new_protocol (protocol_t * protocol, GtkWidget * canvas)
{
  GList *protocol_item = NULL;
  protocol_t *legend_protocol = NULL;
  GtkWidget *prot_table;
  GtkWidget *label;
  GtkArg args[2];
  GtkStyle *style;
```

```
  gchar *color_string;
  guint n_rows, n_columns;
  static gboolean first = TRUE;

  args[0].name = "n_rows";
  args[1].name = "n_columns";

  /* First, we check whether the diagram already knows about this protocol,
   * checking whether it is shown on the legend. */
  /*  g_message ("Looking for %s", protocol->name); */
  if ((protocol_item = g_list_find_custom (legend_protocols,
                                           protocol->name,
protocol_compare)))
    {
      g_my_debug ("Protocol %s found in legend protocols list",
                  protocol->name);
      legend_protocol = (protocol_t *) (protocol_item->data);
      protocol->color = legend_protocol->color;
      return;
    }

  g_my_debug ("Not found. Creating item");

  /* It's not, so we build a new entry on the legend */
  /* First, we add a new row to the table */
  prot_table = glade_xml_get_widget (xml, "prot_table");
  gtk_object_getv (GTK_OBJECT (prot_table), 2, args);
  n_rows = args[0].d.int_data;
  n_columns = args[0].d.int_data;
  /* Glade won't let me define a 0 row table
   * I feel this is ugly, but it's late and I don't feel like
   * cleaning this up :-) */
  if (!first)
    n_rows++;

  first = FALSE;

  /* Then we add the new label widgets */
  label = gtk_label_new (protocol->name);
  gtk_widget_ref (label);
  /* I'm not really sure what this exactly does. I just copied it from
   * interface.c, but what I'm trying to do is set the name of the widget */
  gtk_object_set_data_full (GTK_OBJECT (app1), protocol->name, label,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (label);
  gtk_label_set_justify (GTK_LABEL (label), GTK_JUSTIFY_CENTER);
  gtk_table_attach (GTK_TABLE (prot_table), label,
                    0, 1, n_rows - 1, n_rows,
                    (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                    (GtkAttachOptions) (GTK_EXPAND), 0, 0);
  gtk_table_resize (GTK_TABLE (prot_table), n_rows + 1, n_columns - 1);
  gtk_container_queue_resize (GTK_CONTAINER (app1));


  color_string = get_prot_color (protocol->name);
  g_my_debug ("%s in %s", protocol->name, color_string);
  gdk_color_parse (color_string, &(protocol->color));
  gdk_colormap_alloc_color (gtk_widget_get_colormap (label), &protocol->color,
                            TRUE, TRUE);

  style = gtk_style_new ();
  style->fg[GTK_STATE_NORMAL] = protocol->color;
  gtk_widget_set_style (label, style);
  gtk_style_unref (style);

  /* We create a legend protocol and add it to the list, to keep count */
  legend_protocol = g_malloc (sizeof (protocol_t));
  legend_protocol->name = g_strdup (protocol->name);
  legend_protocol->color = protocol->color;
  legend_protocols = g_list_prepend (legend_protocols, legend_protocol);

  known_protocols[stack_level]++;

}                                                   /* check_new_protocol */

/* Frees the legend_protocols list and empties the table of protocols */
void
delete_gui_protocols (void)
{
  GList *item = NULL;
  protocol_t *protocol = NULL;
  GtkWidget *prot_table = NULL;
  guint i = 0;

  item = legend_protocols;

  while (item)
    {
      protocol = item->data;
      g_free (protocol->name);
```

```
    g_free (protocol);
    item = item->next;
   }

  g_list_free (legend_protocols);
  legend_protocols = NULL;
  prot_color_index = 0;
  for (; i <= STACK_SIZE; i++)
   known_protocols[i] = 0;

  prot_table = glade_xml_get_widget (xml, "prot_table");

  item = gtk_container_children (GTK_CONTAINER (prot_table));

  while (item)
   {
    gtk_container_remove (GTK_CONTAINER (prot_table),
                          (GtkWidget *) item->data);
    item = item->next;
   }

}                                       /* delete_gui_protocols */

/* For a given protocol, returns the color string that should be used
 * Right now it's just assigning colors from a list until it runs
 * out, but this behavious will change */
static gchar *
get_prot_color (gchar * name)
{
  gchar *colors[] = { "red", "blue", "yellow", "white", "orange", "green",
    "pink", "cyan", "brown", "purple", "tan"
  };

  gchar *color = NULL;

  color = colors[prot_color_index];

  prot_color_index++;
  if (prot_color_index == 11)
   prot_color_index = 0;

  return color;

}                                       /* get_prot_color */
```

```
/* Checks if there is a canvas_node per each node. If not, one canvas_node
 * must be created and initiated */
static gint
check_new_node (guint8 * node_id, node_t * node, GtkWidget * canvas)
{
  canvas_node_t *new_canvas_node;
  GnomeCanvasGroup *group;

  if (!node || !node->node_id)
   return FALSE;

  if (display_node (node) && !g_tree_lookup (canvas_nodes, node_id))
   {
    group = gnome_canvas_root (GNOME_CANVAS (canvas));

    new_canvas_node = g_malloc (sizeof (canvas_node_t));
    new_canvas_node->canvas_node_id = g_memdup (node_id, node_id_length);
    new_canvas_node->node = node;

    group = GNOME_CANVAS_GROUP (gnome_canvas_item_new (group,
                                                       gnome_canvas_group_get_type
                                                       (), "x", 100.0, "y",
                                                       100.0, NULL));
    gtk_object_ref (GTK_OBJECT (group));

    new_canvas_node->node_item
            = gnome_canvas_item_new (group,
                                     GNOME_TYPE_CANVAS_ELLIPSE,
                                     "x1", 0.0,
                                     "x2", 0.0,
                                     "y1", 0.0,
                                     "y2", 0.0,
                                     "fill_color",
                                     node_color,
                                     "outline_color",
                                     "black", "width_pixels", 0, NULL);
    gtk_object_ref (GTK_OBJECT (new_canvas_node->node_item));
    new_canvas_node->text_item =
            gnome_canvas_item_new (group, GNOME_TYPE_CANVAS_TEXT, "text",
                                   node->name->str,
                                   "x", 0.0,
                                   "y", 0.0,
                                   "anchor", GTK_ANCHOR_CENTER,
                                   "font", fontname,
                                   "fill_color", text_color, NULL);
    gtk_object_ref (GTK_OBJECT (new_canvas_node->text_item));
```

```
    new_canvas_node->group_item = group;

    gnome_canvas_item_raise_to_top (GNOME_CANVAS_ITEM
                                    (new_canvas_node->text_item));
    gtk_signal_connect (GTK_OBJECT (new_canvas_node->group_item), "event",
                        (GtkSignalFunc) node_item_event, new_canvas_node);

    g_tree_insert (canvas_nodes,
                   new_canvas_node->canvas_node_id, new_canvas_node);
    g_log (G_LOG_DOMAIN, G_LOG_LEVEL_DEBUG,
           _("Creating canvas_node: %s. Number of nodes %d"),
           new_canvas_node->node->name->str, g_tree_nnodes (canvas_nodes));

    new_canvas_node->is_new = TRUE;
    new_canvas_node->shown = TRUE;
    need_reposition = TRUE;
   }

  return FALSE;                                 /* False to keep on traversing */
}                                               /* check_new_node */


/* - updates sizes, names, etc */
static gint
update_canvas_nodes (guint8 * node_id, canvas_node_t * canvas_node,
                     GtkWidget * canvas)
{
  node_t *node;
  gdouble node_size;
  GtkArg args[1];
  GList *protocol_item;
  protocol_t *protocol = NULL;
  static clock_t start = 0;
  clock_t end;
  gdouble cpu_time_used;

  /* We don't need this anymore since now update_nodes is called in update_diagram */
#if 0
  node = canvas_node->node;

  /* First we check whether the link has timed out */
  node = update_node (node);
#endif
#if 1
  node = g_tree_lookup (nodes, node_id);
#endif
```

```
  /* Remove node if node is too old or if capture is stopped */
  if (!node || !display_node (node))
    {
      if (canvas_node->group_item)
                {
                  gtk_object_destroy (GTK_OBJECT (canvas_node->group_item));
                  gtk_object_unref (GTK_OBJECT (canvas_node->group_item));
                  canvas_node->group_item = NULL;
                }
      if (canvas_node->node_item)
                {
                  gtk_object_destroy (GTK_OBJECT (canvas_node->node_item));
                  gtk_object_unref (GTK_OBJECT (canvas_node->node_item));
                  canvas_node->node_item = NULL;
                }
      if (canvas_node->text_item)
                {
                  gtk_object_destroy (GTK_OBJECT (canvas_node->text_item));
                  gtk_object_unref (GTK_OBJECT (canvas_node->text_item));
                  canvas_node->text_item = NULL;
                }

      g_tree_remove (canvas_nodes, node_id);
      g_my_debug ("Removing canvas_node. Number of nodes %d",
                  g_tree_nnodes (canvas_nodes));
      g_free (node_id);
      g_free (canvas_node);
      need_reposition = TRUE;
      return TRUE;                      /* I've checked it's not safe to traverse
                                         * while deleting, so we return TRUE to stop
                                         * the traversion (Does that word exist? :-) */
    }

  if (node->main_prot[stack_level])
    {
      protocol_item = g_list_find_custom (protocols[stack_level],
                                          node->main_prot[stack_level],
                                          protocol_compare);
      if (protocol_item)
                protocol = protocol_item->data;
      else
                g_warning (_("Main node protocol not found in update_canvas_nodes"));
    }

  switch (node_size_variable)
```

```
  {
  case INST_TOTAL:
    node_size = get_node_size (node->average);
    break;
  case INST_INBOUND:
    node_size = get_node_size (node->average_in);
    break;
  case INST_OUTBOUND:
    node_size = get_node_size (node->average_out);
    break;
  case ACCU_TOTAL:
    node_size = get_node_size (node->accumulated);
    break;
  case ACCU_INBOUND:
    node_size = get_node_size (node->accumulated_in);
    break;
  case ACCU_OUTBOUND:
    node_size = get_node_size (node->accumulated_out);
    break;
  default:
    node_size = get_node_size (node->average_out);
    g_warning (_("Unknown value or node_size_variable"));
  }


if (protocol)
  canvas_node->color = protocol->color;

gnome_canvas_item_set (canvas_node->node_item,
                                                "x1", -node_size / 2,
                                                "x2", node_size / 2,
                                                "y1", -node_size / 2,
                                                "y2", node_size / 2,
                                                "fill_color_gdk", &(canvas_node->color), NULL);

/* We check the name of the node, and update the canvas node name
 * if it has changed (useful for non blocking dns resolving) */
/*TODO why is it exactly that sometimes it is NULL? */
if (canvas_node->text_item)
  {
    args[0].name = "text";
    gtk_object_getv (GTK_OBJECT (canvas_node->text_item), 1, args);
    if (strcmp (args[0].d.string_data, node->name->str))
                {
                  gnome_canvas_item_set (canvas_node->text_item,
                                                                "text", node->name->str, NULL);
                  gnome_canvas_item_request_update (canvas_node->text_item);
                }

      /* Memprof is telling us that we have to free the string */
      g_free (args[0].d.string_data);
    }

  /* Processor time check. If too much time has passed, update the GUI */
  end = clock ();
  cpu_time_used = ((gdouble) (end - start)) / CLOCKS_PER_SEC;
  if (cpu_time_used > 0.05)
    {
      /* Force redraw */
      while (gtk_events_pending ())
                gtk_main_iteration ();
      start = end;
    }
  return FALSE;                 /* False means keep on calling the function */

}                                                               /* update_canvas_nodes */


/* Returns whether the node in question should be displayed in the
 * diagram or not */
static gboolean
display_node (node_t * node)
{
  struct timeval diff;

  if (!node)
    return FALSE;

  diff = substract_times (now, node->last_time);

  /* There are problems if a canvas_node is deleted if it still
   * has packets, so we have to check that as well */

  /* Remove canvas_node if node is too old */
  if (IS_OLDER (diff, gui_node_timeout_time) && gui_node_timeout_time
      && !node->n_packets)
    return FALSE;

#if 1
  if ((gui_node_timeout_time == 1) && !node->n_packets)
    g_my_critical ("Impossible situation in display node");
#endif
```

```
  return TRUE;
}                                                       /* display_node */


/* Sorts canvas nodes with the criterium set in preferences and sets
 * which will be displayed in the diagram */
static void
limit_nodes (void)
{
  static GTree *ordered_nodes = NULL;
  static guint limit;
  displayed_nodes = 0;                          /* We'll increment for each node we don't
                                                 * limit */

  if (node_limit < 0)
    {
      displayed_nodes = g_tree_nnodes (canvas_nodes);
      return;
    }

  limit = node_limit;

  ordered_nodes = g_tree_new (traffic_compare);

  g_tree_traverse (canvas_nodes, (GTraverseFunc) add_ordered_node, G_IN_ORDER,
                                        ordered_nodes);
  g_tree_traverse (ordered_nodes, (GTraverseFunc) check_ordered_node,
                                        G_IN_ORDER, &limit);
  g_tree_destroy (ordered_nodes);
  ordered_nodes = NULL;
}                                                       /* limit_nodes */

static gint
add_ordered_node (guint8 * node_id, canvas_node_t * node,
                                        GTree * ordered_nodes)
{
  g_tree_insert (ordered_nodes, node->node, node);
  g_my_debug ("Adding ordered node. Number of nodes: %d",
                        g_tree_nnodes (ordered_nodes));
  return FALSE;                                         /* keep on traversing */
}                                                       /* add_ordered_node */

static gint
check_ordered_node (gdouble * traffic, canvas_node_t * node, guint * count)
{
  /* TODO We can probably optimize this by stopping the traversion once
   * the limit has been reached */
  if (*count)
    {
      if (!node->shown)
                        need_reposition = TRUE;
      node->shown = TRUE;
      displayed_nodes++;
      (*count)--;
    }
  else
    {
      if (node->shown)
                        need_reposition = TRUE;
      node->shown = FALSE;
    }
  return FALSE;                                         /* keep on traversing */
}                                                       /* check_ordered_node */

/* Comparison function used to order the (GTree *) nodes
 * and canvas_nodes heard on the network */
static gint
traffic_compare (gconstpointer a, gconstpointer b)
{
  node_t *node_a, *node_b;

  g_assert (a != NULL);
  g_assert (b != NULL);

  node_a = (node_t *) a;
  node_b = (node_t *) b;

  if (node_a->average < node_b->average)
    return 1;
  if (node_a->average > node_b->average)
    return -1;

  /* If two nodes have the same traffic, we still have
   * to distinguish them somehow. We use the node_id */

  return (node_id_compare (node_a->node_id, node_b->node_id));

}                                                       /* traffic_compare */


/* Called from update_diagram if the global need_reposition
```

```
 * is set. It rearranges the nodes*/
/* TODO I think I should update all links as well, so as not having
 * stale graphics if the diagram has been resized */
static gint
reposition_canvas_nodes (guint8 * ether_addr, canvas_node_t * canvas_node,
                                                GtkWidget * canvas)
{
 static gfloat angle = 0.0;
 static guint node_i = 0, n_nodes = 0;
 gdouble x = 0, y = 0, xmin, ymin, xmax, ymax, text_compensation = 50;
 gdouble x_rad_max, y_rad_max;
 gdouble oddAngle = angle;

 if (!canvas_node->shown)
   {
    gnome_canvas_item_hide (canvas_node->node_item);
    gnome_canvas_item_hide (canvas_node->text_item);
    return FALSE;
   }

 gnome_canvas_get_scroll_region (GNOME_CANVAS (canvas),
                                                        &xmin, &ymin, &xmax, &ymax);
 if (!n_nodes)
   {
    n_nodes = node_i = displayed_nodes;
    g_my_debug ("Displayed nodes = %d", displayed_nodes);
   }

 xmin += text_compensation;
 xmax -= text_compensation;/* Reduce the drawable area so that
                            * the node name is not lost
                            * TODO: Need a function to calculate
                            * text_compensation depending on font size */
 x_rad_max = 0.9 * (xmax - xmin) / 2;
 y_rad_max = 0.9 * (ymax - ymin) / 2;

 /* TODO I've done all the stationary changes in a hurry
  * I should review it an tidy up all this stuff */
 if (stationary)
   {
    if (canvas_node->is_new)
                {
                 static guint count = 0, base = 1;
                 gdouble angle = 0;

                 if (count == 0)
                   {
                    angle = M_PI * 2.0f;
                    count++;
                   }
                 else
                   {

                    if (count > 2 * base)
                                {
                                 base *= 2;
                                 count = 1;
                                }
                    angle = M_PI * (gdouble) count / ((gdouble) base);
                    count += 2;
                   }
                 x = x_rad_max * cos (angle);
                 y = y_rad_max * sin (angle);
                }

   }
 else
   {

    if (n_nodes % 2 == 0)      /* spacing is better when n_nodes is odd and Y is linear */
                oddAngle = (angle * n_nodes) / (n_nodes + 1);
    if (n_nodes > 7)
                {
                 x = x_rad_max * cos (oddAngle);
                 y = y_rad_max * asin (sin (oddAngle)) / (M_PI / 2);
                }
    else
                {
                 x = x_rad_max * cos (angle);
                 y = y_rad_max * sin (angle);
                }
   }

 if (!stationary || canvas_node->is_new)
   {
    /* g_message ("%g %g", x, y); */
    gnome_canvas_item_set (GNOME_CANVAS_ITEM (canvas_node->group_item),
                                                        "x", x, "y", y, NULL);
    canvas_node->is_new = FALSE;
   }

 /* We update the text font */
```

```
  gnome_canvas_item_set (canvas_node->text_item, "font", fontname, NULL);

  if (diagram_only)
    {
      gnome_canvas_item_hide (canvas_node->text_item);
    }
  else
    {
      gnome_canvas_item_show (canvas_node->text_item);
      gnome_canvas_item_request_update (canvas_node->text_item);
    }

  gnome_canvas_item_show (canvas_node->node_item);
  gnome_canvas_item_request_update (canvas_node->node_item);


  node_i--;

  if (node_i)
    angle += 2 * M_PI / n_nodes;
  else
    {
      angle = 0.0;
      n_nodes = 0;
    }

  return FALSE;
}                                                   /* reposition_canvas_nodes */


/* Goes through all known links and checks whether there already exists
 * a corresponding canvas_link. If not, create it.*/
static gint
check_new_link (guint8 * link_id, link_t * link, GtkWidget * canvas)
{
  canvas_link_t *new_canvas_link;
  GnomeCanvasGroup *group;
  GnomeCanvasPoints *points;
  guint i = 0;

  GtkArg args[2];
  args[0].name = "x";
  args[1].name = "y";

  if (!g_tree_lookup (canvas_links, link_id))
    {
      group = gnome_canvas_root (GNOME_CANVAS (canvas));

      new_canvas_link = g_malloc (sizeof (canvas_link_t));
      new_canvas_link->canvas_link_id = g_memdup (link_id,
                                                  2 *
node_id_length);
      new_canvas_link->link = link;

      /* We set the lines position using groups positions */
      points = gnome_canvas_points_new (3);

      for (; i <= 5; i++)
                points->coords[i] = 0.0;

      new_canvas_link->link_item
                = gnome_canvas_item_new (group,
                                         gnome_canvas_polygon_get_type (),
                                         "points", points, "fill_color", "tan",
NULL);
      gtk_object_ref (GTK_OBJECT (new_canvas_link->link_item));


      g_tree_insert (canvas_links,
                     new_canvas_link->canvas_link_id, new_canvas_link);
      gnome_canvas_item_lower_to_bottom (new_canvas_link->link_item);

      gnome_canvas_points_unref (points);

      gtk_signal_connect (GTK_OBJECT (new_canvas_link->link_item), "event",
                          (GtkSignalFunc) link_item_event, new_canvas_link);


      g_log (G_LOG_DOMAIN, G_LOG_LEVEL_DEBUG,
                _("Creating canvas_link: %s-%s. Number of links %d"),
                link->src_name, link->dst_name, g_tree_nnodes (canvas_links));

    }

  return FALSE;
}                                                   /* check_new_link */


/* - calls update_links, so that the related link updates its average
 *   traffic and main protocol, and old links are deleted
```

```
 * - caculates link size and color fading */
static gint
update_canvas_links (guint8 * link_id, canvas_link_t * canvas_link,
                                        GtkWidget * canvas)
{
  link_t *link;
  GnomeCanvasPoints *points;
  canvas_node_t *canvas_node;
  GList *protocol_item;
  protocol_t *protocol = NULL;
  GtkArg args[2];
  gdouble link_size, versorx, versory, modulus;
  struct timeval diff;
  guint32 scaledColor;
  gdouble scale;

  link = canvas_link->link;

#if 0
  /* First we check whether the link has timed out */
  link = update_link (link);
#endif
  link = g_tree_lookup (links, link_id);

  if (!link)
    {
      /* TODO VERY IMPORTANT *
       * Make the same destroying procedure as in the canvas_nodes
       * Else we are prone to have memory corruption */
      gtk_object_destroy (GTK_OBJECT (canvas_link->link_item));
      gtk_object_unref (GTK_OBJECT (canvas_link->link_item));

      g_tree_remove (canvas_links, link_id);
      g_my_debug ("Removing canvas link. Number of links %d",
                                        g_tree_nnodes (canvas_links));
      g_free (link_id);
      g_free (canvas_link);
      return TRUE;              /* I've checked it's not safe to traverse
                                 * while deleting, so we return TRUE to stop
                                 * the traversion (Does that word exist? :-) */
    }


  /* TODO do we really have to check for link in this two? */
  if (link)
    diff = substract_times (now, link->last_time);

  if (link && link->main_prot[stack_level])
    {
      protocol_item = g_list_find_custom (protocols[stack_level],
                                                                        link-
>main_prot[stack_level],
                                                                        protocol_compare);
      if (protocol_item)
                protocol = protocol_item->data;
      else
                g_warning (_("Main link protocol not found in update_canvas_links"));
    }
  args[0].name = "x";
  args[1].name = "y";

  points = gnome_canvas_points_new (3);

  /* If either source or destination has disappeared, we hide the link
   * until it can be show again */
  /* TODO: This is a dirty hack. Redo this again later by properly
   * deleting the link */

  /* We get coords for the destination node */
  canvas_node = g_tree_lookup (canvas_nodes, link_id + node_id_length);
  if (!canvas_node || !canvas_node->shown)
    {
      gnome_canvas_item_hide (canvas_link->link_item);
      gnome_canvas_points_unref (points);
      return FALSE;
    }
  gtk_object_getv (GTK_OBJECT (canvas_node->group_item), 2, args);
  points->coords[0] = args[0].d.double_data;
  points->coords[1] = args[1].d.double_data;

  /* We get coords from source node */
  canvas_node = g_tree_lookup (canvas_nodes, link_id);
  if (!canvas_node || !canvas_node->shown)
    {
      gnome_canvas_item_hide (canvas_link->link_item);
      gnome_canvas_points_unref (points);
      return FALSE;
    }
  gtk_object_getv (GTK_OBJECT (canvas_node->group_item), 2, args);

  versorx = -(points->coords[1] - args[1].d.double_data);
  versory = points->coords[0] - args[0].d.double_data;
```

```
  modulus = sqrt (pow (versorx, 2) + pow (versory, 2));
  link_size = get_link_size (link->average) / 2;

  points->coords[2] = args[0].d.double_data + (versorx / modulus) * link_size;
  points->coords[3] = args[1].d.double_data + (versory / modulus) * link_size;
  points->coords[4] = args[0].d.double_data - (versorx / modulus) * link_size;
  points->coords[5] = args[1].d.double_data - (versory / modulus) * link_size;

  /* TODO What if there never is a protocol?
   * I have to initialize canvas_link->color to a known value */
  if (protocol)
    canvas_link->color = protocol->color;

  if (nofade)
    {
      guint32 color;
      color = (((int) (canvas_link->color.red) & 0xFF00) << 16) |
                    (((int) (canvas_link->color.green) & 0xFF00) << 8) |
                    ((int) (canvas_link->color.blue) & 0xFF00) | 0xFF;
      gnome_canvas_item_set (canvas_link->link_item,
                                                  "points", points,
                                                  "fill_color_rgba", color, NULL);
    }
  else
    {
      /* scale color down to 10% at link timeout */
      scale =
                    pow (0.10,
                        (diff.tv_sec * 1000.0 +
                         diff.tv_usec / 1000) / link_timeout_time);
      scaledColor =
                    (((int) (scale * canvas_link->color.red) & 0xFF00) << 16) |
                    (((int) (scale * canvas_link->color.green) & 0xFF00) << 8) |
                    ((int) (scale * canvas_link->color.blue) & 0xFF00) | 0xFF;
      gnome_canvas_item_set (canvas_link->link_item, "points", points,
                                                  "fill_color_rgba", scaledColor, NULL);
    }

  /* If we got this far, the link can be shown. Make sure it is */
  gnome_canvas_item_show (canvas_link->link_item);

  gnome_canvas_points_unref (points);

  return FALSE;

}                                                   /* update_canvas_links */
```

```
/* Returs the radius in pixels given average traffic and size mode */
static gdouble
get_node_size (gdouble average)
{
  gdouble result = 0.0;
  switch (size_mode)
    {
    case LINEAR:
      result = average + 1;
      break;
    case LOG:
      result = log (average + 1);
      break;
    case SQRT:
      result = sqrt (average + 1);
      break;
    }
  return (double) (5 + node_radius_multiplier * result);
}

/* Returs the width in pixels given average traffic and size mode */
static gdouble
get_link_size (gdouble average)
{
  gdouble result = 0.0;
  switch (size_mode)
    {
    case LINEAR:
      result = average + 1;
      break;
    case LOG:
      result = log (average + 1);
      break;
    case SQRT:
      result = sqrt (average + 1);
      break;
    }
  return (double) (1 + link_width_multiplier * result);
}                                                   /* get_link_size */


/* Called for every event a link receives. Right now it's used to
 * set a message in the appbar */
```

```
static gint
link_item_event (GnomeCanvasItem * item, GdkEvent * event,
                 canvas_link_t * canvas_link)
{
  static GnomeAppBar *appbar;
  gchar *str;

  if (!appbar)
    appbar = GNOME_APPBAR (glade_xml_get_widget (xml, "appbar1"));

  switch (event->type)
    {

    case GDK_ENTER_NOTIFY:
      if (canvas_link && canvas_link->link
                && canvas_link->link->main_prot[stack_level])
              str =
                g_strdup_printf (_("Link main protocol: %s"),
                                 canvas_link->link->main_prot[stack_level]);
      else
              str = g_strdup_printf (_("Link main protocol unknown"));
      gnome_appbar_push (appbar, str);
      g_free (str);
      break;
    case GDK_LEAVE_NOTIFY:
      gnome_appbar_pop (appbar);
      break;
    default:
      break;
    }

  return FALSE;
}                                               /* link_item_event */


/* Called for every event a node receives. Right now it's used to
 * set a message in the appbar and launch the popup timeout */
static gint
node_item_event (GnomeCanvasItem * item, GdkEvent * event,
                 canvas_node_t * canvas_node)
{

  gdouble item_x, item_y;
  static GnomeAppBar *appbar;
#if 0
  gchar *str;
  static gint popup = 0;
  static struct popup_data pd = { NULL, NULL };
#endif

  if (!appbar)
    appbar = GNOME_APPBAR (glade_xml_get_widget (xml, "appbar1"));

  /* This is not used yet, but it will be. */
  item_x = event->button.x;
  item_y = event->button.y;
  gnome_canvas_item_w2i (item->parent, &item_x, &item_y);

  switch (event->type)
    {

      /* I am finally going to get rid of the hideous popup! */
#if 0
    case GDK_ENTER_NOTIFY:
      pd.canvas_node = canvas_node;
      popup = gtk_timeout_add (1000, (GtkFunction) popup_to, &pd);
      str = g_strdup_printf ("%s (%s)",
                                        canvas_node->node->name->str,
                                        canvas_node->node->numeric_name->str);
      gnome_appbar_push (appbar, str);
      g_free (str);
      break;
    case GDK_LEAVE_NOTIFY:
      if (popup)
                {
                  gtk_timeout_remove (popup);
                  popup = 0;
                  pd.canvas_node = NULL;
                  pd.node_popup = NULL;
                }
      gnome_appbar_pop (appbar);
      break;
#endif
    case GDK_2BUTTON_PRESS:
      if (!canvas_node || !canvas_node->node)
                return FALSE;
      create_node_info_window (canvas_node);
      break;
    default:
      break;
    }
```

```
  return FALSE;

}                                                   /* node_item_event */


#if 0
/* This function is the one that sets ups and displays the node
 * pop up windows */
static guint
popup_to (struct popup_data *pd)
{

  GladeXML *xml_popup;
  GtkLabel *label;
  gchar *str;

  xml_popup = glade_xml_new (GLADEDIR "/" ETHERAPE_GLADE_FILE, "node_popup");
  glade_xml_signal_autoconnect (xml);
  pd->node_popup = glade_xml_get_widget (xml_popup, "node_popup");

  /* TODO Why is not the signal connection being set up automatically?
   * I don't know, and so I have to do it on my own while I investigate
   * the problem */
  gtk_widget_set_events (pd->node_popup, GDK_POINTER_MOTION_MASK);
  gtk_signal_connect (GTK_OBJECT (pd->node_popup), "motion_notify_event",
                      GTK_SIGNAL_FUNC (gtk_widget_destroy), NULL);

  label = (GtkLabel *) glade_xml_get_widget (xml_popup, "name");

  /* This function may be called even before the node has a name
   * If that happens, return */
  if (!(pd->canvas_node) || !(pd->canvas_node->node)
      || !(pd->canvas_node->node->name)
      || !(pd->canvas_node->node->numeric_name))
    return FALSE;

  if (mode == ETHERNET && pd->canvas_node->node->ip_address
      && pd->canvas_node->node->numeric_ip)
    {
      str = g_strdup_printf ("%s (%s, %s)",
                             pd->canvas_node->node->name->str,
                             pd->canvas_node->node->numeric_ip->str,
                             pd->canvas_node->node->numeric_name->str);
      gtk_label_set_text (label, str);
      g_free (str);
    }
  else
    {
      str = g_strdup_printf ("%s (%s)",
                             pd->canvas_node->node->name->str,
                             pd->canvas_node->node->numeric_name->str);
      gtk_label_set_text (label, str);
      g_free (str);
    }


  label = (GtkLabel *) glade_xml_get_widget (xml_popup, "accumulated");
  str =
    g_strdup_printf ("Acummulated bytes: %g",
                     pd->canvas_node->node->accumulated);
  gtk_label_set_text (label, str);
  g_free (str);

  label = (GtkLabel *) glade_xml_get_widget (xml_popup, "average");
  str = g_strdup_printf ("Average bps: %g", pd->canvas_node->node->average);
  gtk_label_set_text (label, str);
  g_free (str);

  gtk_widget_show (GTK_WIDGET (pd->node_popup));

/*  gtk_object_unref(GTK_OBJECT(xml_popup)); */
  return FALSE;                                     /* Only called once */

}                                                   /* popup_to */
#endif

/* Pushes a string into the appbar status area */

void
set_appbar_status (gchar * str)
{
  static GnomeAppBar *appbar = NULL;
  static gchar *status_string = NULL;

  if (status_string)
    g_free (status_string);

  status_string = g_strdup (str);

  if (!appbar)
```

```
   appbar = GNOME_APPBAR (glade_xml_get_widget (xml, "appbar1"));

 gnome_appbar_pop (appbar);
 gnome_appbar_push (appbar, status_string);

}                                                   /* set_appbar_status */

gchar *
traffic_to_str (gdouble traffic, gboolean is_speed)
{
 static gchar *str = NULL;

 if (str)
   g_free (str);

 if (is_speed)
   {
    if (traffic > 1000000)
                str = g_strdup_printf ("%.3f Mbps", traffic / 1000000);
    else if (traffic > 1000)
                str = g_strdup_printf ("%.3f Kbps", traffic / 1000);
    else
                str = g_strdup_printf ("%.0f bps", traffic);
   }
 else
   {
    /* Debug code for sanity check */
    if (traffic && traffic < 1)
                g_warning ("Ill traffic value in traffic_to_str");

    if (traffic > 1024 * 1024)
                str = g_strdup_printf ("%.3f Mbytes", traffic / 1024 / 1024);
    else if (traffic > 1024)
                str = g_strdup_printf ("%.3f Kbytes", traffic / 1024);
    else
                str = g_strdup_printf ("%.0f bytes", traffic);
   }

 return str;
}                                                   /* traffic_to_str */
```


menus.h

```
/* EtherApe
 * Copyright (C) 2001 Juan Toledo
 * $Id: menus.h,v 1.8 2001/07/06 10:14:00 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <gtk/gtk.h>
#include "globals.h"
#include "util.h"

GList *interface_list = NULL;                   /* A list of all usable interface */

static GnomeUIInfo help_submenu[] = {
  GNOMEUIINFO_HELP ("etherape"),
  GNOMEUIINFO_END
};

static gboolean in_start_capture;

void on_open_activate (GtkMenuItem * menuitem, gpointer user_data);
void on_file_cancel_button_clicked (GtkButton * button, gpointer user_data);
void on_file_combo_entry_changed (GtkEditable * editable, gpointer user_data);
void on_file_ok_button_clicked (GtkButton * button, gpointer user_data);

void on_interface_radio_activate (gchar * gui_device);
void on_mode_radio_activate (GtkMenuItem * menuitem, gpointer user_data);
```

```
void on_start_menuitem_activate (GtkMenuItem * menuitem, gpointer user_data);
void on_stop_menuitem_activate (GtkMenuItem * menuitem, gpointer user_data);

void on_toolbar_check_activate (GtkCheckMenuItem * menuitem,
                                                    gpointer user_data);
void on_legend_check_activate (GtkCheckMenuItem * menuitem,
                                                  gpointer user_data);
void on_status_bar_check_activate (GtkCheckMenuItem * menuitem,
                                                     gpointer user_data);

void on_about1_activate (GtkMenuItem * menuitem, gpointer user_data);
static void set_active_interface (void);
```


menus.c

```
/* EtherApe
 * Copyright (C) 2001 Juan Toledo
 * $Id: menus.c,v 1.19 2001/07/06 10:14:00 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include "menus.h"

#if 0
/* This is here just as an example in case I need it again */
void
on_file1_activate (GtkMenuItem * menuitem, gpointer user_data)
{
 GtkWidget *messagebox;
 GladeXML *xml_messagebox;
 xml_messagebox =
   glade_xml_new (GLADEDIR "/" ETHERAPE_GLADE_FILE, "messagebox1");
 messagebox = glade_xml_get_widget (xml_messagebox, "messagebox1");
 if (!xml)
   {
    g_error (_("We could not load the interface! (%s)"),
                        GLADEDIR "/" ETHERAPE_GLADE_FILE);
    return;
   }
 gtk_widget_show (messagebox);
 gtk_object_unref (GTK_OBJECT (xml_messagebox));
}
#endif

void
```

```
init_menus (void)
{
 GtkWidget *widget = NULL, *item = NULL;
 gint err;
 gchar err_str[300];
 GList *interfaces;
 GSList *group = NULL;
 GString *info_string = NULL;

 /* It seems libglade is not acknowledging the "Use gnome help" option in the
  * glade file and so it is not automatically adding the help items in the help
  * menu. Thus I must add it manually here */

 widget = glade_xml_get_widget (xml, "help1_menu");
 gnome_app_fill_menu ((GtkMenuShell *) widget, help_submenu,
                                        gtk_accel_group_get_default (), TRUE, 1);

 interface_list = get_interface_list (&err, err_str);

 interfaces = interface_list;

 if (!interfaces)
   {
    g_my_info (_("No suitables interfaces for capture have been found"));
    return;
   }

 widget = glade_xml_get_widget (xml, "interfaces_menu");

 info_string = g_string_new (_("Available interfaces for capture:"));

 /* Set up a hidden dummy interface to set when there is no active
  * interface */
 item = gtk_radio_menu_item_new_with_label (group, "apedummy");
 group = gtk_radio_menu_item_group (GTK_RADIO_MENU_ITEM (item));
 gtk_menu_append (GTK_MENU (widget), item);

 /* Set up the real interfaces menu entries */
 while (interfaces)
   {
    item =
                gtk_radio_menu_item_new_with_label (group,
                                                    (gchar *) (interfaces-
>data));
    group = gtk_radio_menu_item_group (GTK_RADIO_MENU_ITEM (item));
    gtk_menu_append (GTK_MENU (widget), item);
    gtk_widget_show (item);
    gtk_signal_connect_object (GTK_OBJECT (item), "activate",
                                                    GTK_SIGNAL_FUNC
                                                    (on_interface_radio_activate),
                                                    (gpointer) g_strdup (interfaces->data));
    g_string_append (info_string, " ");
    g_string_append (info_string, (gchar *) (interfaces->data));
    interfaces = interfaces->next;
   }


 g_my_info (info_string->str);

}                                                   /* init_menus */

/* FILE MENU */

void
on_open_activate (GtkMenuItem * menuitem, gpointer user_data)
{
 static GtkWidget *open_file_dialog = NULL;
 if (!open_file_dialog)
   open_file_dialog = glade_xml_get_widget (xml, "open_file_dialog");
 gtk_widget_show (open_file_dialog);
}                                                   /* on_open_activate */

void
on_file_cancel_button_clicked (GtkButton * button, gpointer user_data)
{
 GtkWidget *open_file_dialog;
 open_file_dialog = glade_xml_get_widget (xml, "open_file_dialog");
 gtk_widget_hide (open_file_dialog);
}                                                   /* on_file_cancel_button_clicked */

/* Sets up the new input capture file */
void
on_file_combo_entry_changed (GtkEditable * editable, gpointer user_data)
{
 gchar *str;
 str = gtk_editable_get_chars (editable, 0, -1);
 if (input_file)
   g_free (input_file);

 input_file = g_strdup (str);
 g_free (str);
```

```
}                                                   /* on_file_combo_entry_changed */

void
on_file_ok_button_clicked (GtkButton * button, gpointer user_data)
{
 GtkWidget *widget;
 gchar *str = NULL;

 if (status != STOP)
   if (!gui_stop_capture ())
     return;

 if (interface)
   {
     g_free (interface);
     interface = NULL;
   }

 widget = glade_xml_get_widget (xml, "file_combo_entry");
 str = gtk_editable_get_chars (GTK_EDITABLE (widget), 0, -1);

 widget = glade_xml_get_widget (xml, "open_file_dialog");
 gtk_widget_hide (widget);


 if (input_file)
   g_free (input_file);
 input_file = g_strdup (str);
 if (str)
   g_free (str);

 widget = glade_xml_get_widget (xml, "fileentry");
 update_history (GNOME_ENTRY
                 (gnome_file_entry_gnome_entry (GNOME_FILE_ENTRY (widget))),
                 input_file, TRUE);

 gui_start_capture ();

}                                                   /* on_file_ok_button_clicked */


/* Capture menu */

void
on_interface_radio_activate (gchar * gui_device)
{
 g_assert (gui_device != NULL);

 if (interface && !strcmp (gui_device, interface))
   return;

 if (in_start_capture)
   return;                                          /* Disregard when called because
                                                     * of interface look change from
                                                     * start_capture */


 if (status != STOP)
   if (!gui_stop_capture ())
     return;

 if (input_file)
   g_free (input_file);
 input_file = NULL;

 if (interface)
   g_free (interface);
 interface = g_strdup (gui_device);

 gui_start_capture ();

 g_my_info (_("Capture interface set to %s in GUI"), gui_device);
}                                                   /* on_interface_radio_activate */

void
on_mode_radio_activate (GtkMenuItem * menuitem, gpointer user_data)
{
 apemode_t new_mode = DEFAULT;

 g_assert (user_data != NULL);

 if (in_start_capture)
   return;                                          /* Disregard when called because
                                                     * of interface look change from
                                                     * start_capture */

 g_my_debug ("Initial mode in on_mode_radio_activate %s",
             (gchar *) user_data);

 if (!strcmp ("IEEE802", user_data))
   new_mode = IEEE802;
 else if (!strcmp ("FDDI", user_data))
   new_mode = FDDI;
```

```
else if (!strcmp ("Ethernet", user_data))
  new_mode = ETHERNET;
else if (!strcmp ("IP", user_data))
  new_mode = IP;
else if (!strcmp ("TCP", user_data))
  new_mode = TCP;
else if (!strcmp ("UDP", user_data))
  new_mode = UDP;
else
  {
    g_my_critical ("Unsopported mode in on_mode_radio_activate");
    exit (1);
  }

if (new_mode == mode)
  return;

/* I don't know why, but this menu item is called twice, instead
 * of once. This forces me to make sure we are not trying to set
 * anything impossible */

g_my_debug ("Mode menuitem active: %d",
                    GTK_CHECK_MENU_ITEM (menuitem)->active);

switch (linktype)
  {
  case L_NULL:
  case L_RAW:
    if ((new_mode == ETHERNET) || (new_mode == FDDI)
                  || (new_mode == IEEE802))
                return;
    break;
  case L_EN10MB:
    if ((new_mode == FDDI) || (new_mode == IEEE802))
                return;
    break;
  case L_FDDI:
    if (new_mode == ETHERNET || (new_mode == IEEE802))
                return;
    break;
  case L_IEEE802:
    if (new_mode == ETHERNET || (new_mode == FDDI))
                return;
    break;
  default:
  }
  if (status != STOP)
    if (!gui_stop_capture ())
      return;
  mode = new_mode;
  g_my_info (_("Mode set to %s in GUI"), (gchar *) user_data);
  gui_start_capture ();

}                                               /* on_mode_radio_activate */

void
on_start_menuitem_activate (GtkMenuItem * menuitem, gpointer user_data)
{
  gui_start_capture ();

}                                               /* on_start_menuitem_activate */

void
on_pause_menuitem_activate (GtkMenuItem * menuitem, gpointer user_data)
{
  gui_pause_capture ();

}                                               /* on_pause_menuitem_activate */

void
on_stop_menuitem_activate (GtkMenuItem * menuitem, gpointer user_data)
{

  gui_stop_capture ();

}                                               /* on_stop_menuitem_activate */



/* View menu */



void
on_toolbar_check_activate (GtkCheckMenuItem * menuitem, gpointer user_data)
{
  GnomeDock *dock;
  GtkWidget *widget;

  dock = GNOME_DOCK (glade_xml_get_widget (xml, "dock1"));
  widget = glade_xml_get_widget (xml, "dock_toolbar");
  if (menuitem->active)
```

```
    gtk_widget_show (widget);
  else
    gtk_widget_hide (widget);

  gtk_container_queue_resize (GTK_CONTAINER (dock));
}                                               /* on_toolbar_check_activate */

void
on_legend_check_activate (GtkCheckMenuItem * menuitem, gpointer user_data)
{
  GnomeDock *dock;
  GtkWidget *widget;

  dock = GNOME_DOCK (glade_xml_get_widget (xml, "dock1"));
  widget = glade_xml_get_widget (xml, "dock_legend");
  if (menuitem->active)
    gtk_widget_show (widget);
  else
    gtk_widget_hide (widget);

  gtk_container_queue_resize (GTK_CONTAINER (dock));

}                                               /* on_legend_check_activate */

void
on_status_bar_check_activate (GtkCheckMenuItem * menuitem, gpointer user_data)
{
  GtkWidget *widget;
  widget = glade_xml_get_widget (xml, "appbar1");
  if (menuitem->active)
    gtk_widget_show (widget);
  else
    gtk_widget_hide (widget);

  gtk_container_queue_resize (GTK_CONTAINER (app1));
}                                               /* on_status_bar_check_activate */


/* Help menu */


void
on_about1_activate (GtkMenuItem * menuitem, gpointer user_data)
{
  GtkWidget *about;
  GladeXML *xml_about;

  xml_about = glade_xml_new (GLADEDIR "/" ETHERAPE_GLADE_FILE, "about2");
  if (!xml_about)
    {
      g_error (_("We could not load the interface! (%s)"),
                        GLADEDIR "/" ETHERAPE_GLADE_FILE);
      return;
    }
  about = glade_xml_get_widget (xml_about, "about2");
  gtk_object_destroy (GTK_OBJECT (xml_about));

  gtk_widget_show (about);

}                                               /* on_about1_activate */


/* Helper functions */


/* Saves the history of a gnome_entry making sure there are no duplicates */
void
update_history (GnomeEntry * gentry, const gchar * str, gboolean is_fileentry)
{
  GList *entry_items = NULL;
  gboolean is_new = TRUE, normal_case = TRUE;

  /* The combo fills up even if there are duplicate values. I think
   * this is a bug in gnome and might be fixed in the future, but
   * right now I'll just make sure the history is fine and reload that*/

  /* TODO There really should be a better way of getting all values from
   * the drop down list than this casting mess :-( */

  /* The file entry introduces a copy of the file name at the end of the
   * list if the browse button has been used. There goes another hack to
   * work around it */
  entry_items = GTK_LIST (GTK_COMBO (gentry)->list)->children;
  while (entry_items)
    {
      gchar *history_value;
      history_value = GTK_LABEL (GTK_BIN (entry_items->data)->child)->label;
      if (!strcmp (history_value, str) && !is_fileentry
```

```
                && (entry_items->next != NULL))
              is_new = FALSE;
    normal_case = !(is_fileentry && !(entry_items->next));
    entry_items = g_list_next (entry_items);
   }

 if (is_new)
   {
    g_my_info ("New entry history item: %s", str);
    if (normal_case)
              gnome_entry_prepend_history (gentry, TRUE, str);
    gnome_entry_save_history (gentry);
   }

 gnome_entry_load_history (gentry);
}                                                    /* update_history */

/* Sets up the GUI to reflect changes and calls start_capture() */
void
gui_start_capture (void)
{
 GtkWidget *widget;
 gchar *errorbuf = NULL;
 GString *status_string = NULL;

 if ((status == PAUSE) && end_of_file)
  if (!gui_stop_capture ())
    return;

 if (status == STOP)
  if ((errorbuf = init_capture ()) != NULL)
    {
              fatal_error_dialog (errorbuf);
              return;
    }

 if (status == PLAY)
   {
    g_warning (_("Status not STOP or PAUSE at gui_start_capture"));
    return;
   }

 if (!start_capture ())
  return;

 in_start_capture = TRUE;
```

```
/* Enable and disable control buttons */
widget = glade_xml_get_widget (xml, "stop_button");
gtk_widget_set_sensitive (widget, TRUE);
widget = glade_xml_get_widget (xml, "stop_menuitem");
gtk_widget_set_sensitive (widget, TRUE);
widget = glade_xml_get_widget (xml, "start_button");
gtk_widget_set_sensitive (widget, FALSE);
widget = glade_xml_get_widget (xml, "start_menuitem");
gtk_widget_set_sensitive (widget, FALSE);
widget = glade_xml_get_widget (xml, "pause_button");
gtk_widget_set_sensitive (widget, TRUE);
widget = glade_xml_get_widget (xml, "pause_menuitem");
gtk_widget_set_sensitive (widget, TRUE);

/* Enable and disable mode buttons */

switch (linktype)
  {
  case L_NULL:
  case L_RAW:
    widget = glade_xml_get_widget (xml, "ieee802_radio");
    gtk_widget_set_sensitive (widget, FALSE);
    widget = glade_xml_get_widget (xml, "fddi_radio");
    gtk_widget_set_sensitive (widget, FALSE);
    widget = glade_xml_get_widget (xml, "ethernet_radio");
    gtk_widget_set_sensitive (widget, FALSE);
    break;
  case L_FDDI:
    widget = glade_xml_get_widget (xml, "fddi_radio");
    gtk_widget_set_sensitive (widget, TRUE);
    widget = glade_xml_get_widget (xml, "ethernet_radio");
    gtk_widget_set_sensitive (widget, FALSE);
    widget = glade_xml_get_widget (xml, "ieee802_radio");
    gtk_widget_set_sensitive (widget, FALSE);
    break;
  case L_EN10MB:
    widget = glade_xml_get_widget (xml, "fddi_radio");
    gtk_widget_set_sensitive (widget, FALSE);
    widget = glade_xml_get_widget (xml, "ethernet_radio");
    gtk_widget_set_sensitive (widget, TRUE);
    widget = glade_xml_get_widget (xml, "ieee802_radio");
    gtk_widget_set_sensitive (widget, FALSE);
    break;
  case L_IEEE802:
    widget = glade_xml_get_widget (xml, "fddi_radio");
```

```
    gtk_widget_set_sensitive (widget, FALSE);
    widget = glade_xml_get_widget (xml, "ethernet_radio");
    gtk_widget_set_sensitive (widget, FALSE);
    widget = glade_xml_get_widget (xml, "ieee802_radio");
    gtk_widget_set_sensitive (widget, TRUE);
    break;
  default:
  }

/* Set active mode in GUI */

switch (mode)
  {
  case IEEE802:
    widget = glade_xml_get_widget (xml, "ieee802_radio");
    break;
  case FDDI:
    widget = glade_xml_get_widget (xml, "fddi_radio");
    break;
  case ETHERNET:
    widget = glade_xml_get_widget (xml, "ethernet_radio");
    break;
  case IP:
    widget = glade_xml_get_widget (xml, "ip_radio");
    break;
  case TCP:
    widget = glade_xml_get_widget (xml, "tcp_radio");
    break;
  case UDP:
    widget = glade_xml_get_widget (xml, "udp_radio");
    break;
  default:
  }
gtk_check_menu_item_set_active (GTK_CHECK_MENU_ITEM (widget), TRUE);

/* Set the interface in GUI */
set_active_interface ();

/* Sets the appbar */

status_string = g_string_new (_("Reading data from "));

if (input_file)
  g_string_append (status_string, input_file);
else if (interface)
  g_string_append (status_string, interface);
else
  g_string_append (status_string, _("default interface"));

switch (mode)
  {
  case IEEE802:
    g_string_append (status_string, _(" in Token Ring mode"));
    break;
  case FDDI:
    g_string_append (status_string, _(" in FDDI mode"));
    break;
  case ETHERNET:
    g_string_append (status_string, _(" in Ethernet mode"));
    break;
  case IP:
    g_string_append (status_string, _(" in IP mode"));
    break;
  case TCP:
    g_string_append (status_string, _(" in TCP mode"));
    break;
  case UDP:
    g_string_append (status_string, _(" in UDP mode"));
    break;
  default:
  }

set_appbar_status (status_string->str);
g_string_free (status_string, TRUE);

in_start_capture = FALSE;

g_my_info (_("Diagram started"));
}                                                    /* gui_start_capture */


void
gui_pause_capture (void)
{
 GtkWidget *widget;
 GString *status_string = NULL;

 if (status == PAUSE)
   {
    g_warning (_("Status not PLAY at gui_pause_capture"));
    return;
```

```
  }

 /*
  * Make sure the data in the info windows is updated
  * so that it is consistent
  */
 update_info_windows ();

 if (!pause_capture ())
  return;

 widget = glade_xml_get_widget (xml, "stop_button");
 gtk_widget_set_sensitive (widget, TRUE);
 widget = glade_xml_get_widget (xml, "stop_menuitem");
 gtk_widget_set_sensitive (widget, TRUE);
 widget = glade_xml_get_widget (xml, "start_button");
 gtk_widget_set_sensitive (widget, TRUE);
 widget = glade_xml_get_widget (xml, "start_menuitem");
 gtk_widget_set_sensitive (widget, TRUE);
 widget = glade_xml_get_widget (xml, "pause_button");
 gtk_widget_set_sensitive (widget, FALSE);
 widget = glade_xml_get_widget (xml, "pause_menuitem");
 gtk_widget_set_sensitive (widget, FALSE);


 /* Sets the appbar */

 status_string = g_string_new (_("Paused"));

 set_appbar_status (status_string->str);
 g_string_free (status_string, TRUE);

 g_my_info (_("Diagram paused"));

}                                                    /* gui_pause_capture */

/* Sets up the GUI to reflect changes and calls stop_capture() */
gboolean
gui_stop_capture (void)
{
 GtkWidget *widget;
 GString *status_string = NULL;

 /*
  * gui_stop_capture needs to call update_diagram in order to
  * delete all canvas_nodes and nodes. But since a slow running
  * update_diagram will yield to pending events, gui_stop_capture
  * might end up being called below another update_diagram. We can't
  * allow two simultaneous calls, so we fail
  */

 if (already_updating)
  return FALSE;

 if (status == STOP)
  {
   g_warning (_("Status not PLAY or PAUSE at gui_stop_capture"));
   return FALSE;
  }

 if (!stop_capture ())
  return FALSE;

 widget = glade_xml_get_widget (xml, "start_button");
 gtk_widget_set_sensitive (widget, TRUE);
 widget = glade_xml_get_widget (xml, "start_menuitem");
 gtk_widget_set_sensitive (widget, TRUE);
 widget = glade_xml_get_widget (xml, "stop_button");
 gtk_widget_set_sensitive (widget, FALSE);
 widget = glade_xml_get_widget (xml, "stop_menuitem");
 gtk_widget_set_sensitive (widget, FALSE);
 widget = glade_xml_get_widget (xml, "pause_button");
 gtk_widget_set_sensitive (widget, FALSE);
 widget = glade_xml_get_widget (xml, "pause_menuitem");
 gtk_widget_set_sensitive (widget, FALSE);

 /* Delete and free protocol information */
 delete_gui_protocols ();

 /* By calling update_diagram, we are forcing node_update
  * and link_update, thus deleting all nodes and links since
  * status=STOP. Then the diagram is redrawn */
 widget = glade_xml_get_widget (xml, "canvas1");
 update_diagram (widget);

 /* Now update info windows */
 update_info_windows ();

 /* Sets the appbar */

 status_string = g_string_new (_("Ready to capture from "));
```

```
  if (input_file)
    g_string_append (status_string, input_file);
  else if (interface)
    g_string_append (status_string, interface);
  else
    g_string_append (status_string, _("default interface"));

  set_appbar_status (status_string->str);
  g_string_free (status_string, TRUE);

  g_my_info (_("Diagram stopped"));

  return TRUE;
}                                                   /* gui_stop_capture */

void
fatal_error_dialog (const gchar * message)
{
  GtkWidget *error_messagebox;

  /* TODO I am not very sure I am not leaking with this, but this is
   * extraordinary and would hardly be noticeable */

  error_messagebox = gnome_message_box_new (message,
                        GNOME_MESSAGE_BOX_ERROR,
                        GNOME_STOCK_BUTTON_OK, NULL);
  gtk_widget_show (error_messagebox);

}                                                   /* fatal_error_dialog */

void
set_active_interface ()
{
  GtkWidget *widget;
  GList *menu_items = NULL;
  gchar *label;

  widget = glade_xml_get_widget (xml, "interfaces_menu");

  menu_items = GTK_MENU_SHELL (widget)->children;

  while (menu_items)
    {
      widget = (GtkWidget *) (menu_items->data);

      if (input_file)
                {
                  gtk_check_menu_item_set_active (GTK_CHECK_MENU_ITEM (widget), TRUE);
                  return;
                }

      label = GTK_LABEL (GTK_BIN (widget)->child)->label;

      if (!strcmp (label, interface))
                gtk_check_menu_item_set_active (GTK_CHECK_MENU_ITEM (widget), TRUE);

      menu_items = menu_items->next;
    }

#if 0
  GList *entry_items = NULL;
  gboolean is_new = TRUE, normal_case = TRUE;

  /* The combo fills up even if there are duplicate values. I think
   * this is a bug in gnome and might be fixed in the future, but
   * right now I'll just make sure the history is fine and reload that*/

  /* TODO There really should be a better way of getting all values from
   * the drop down list than this casting mess :-( */

  /* The file entry introduces a copy of the file name at the end of the
   * list if the browse button has been used. There goes another hack to
   * work around it */
  entry_items = GTK_LIST (GTK_COMBO (gentry)->list)->children;
  while (entry_items)
    {
      gchar *history_value;
      history_value = GTK_LABEL (GTK_BIN (entry_items->data)->child)->label;
      if (!strcmp (history_value, str) && !is_fileentry
                  && (entry_items->next != NULL))
                is_new = FALSE;
      normal_case = !(is_fileentry && !(entry_items->next));
      entry_items = g_list_next (entry_items);
    }
#endif

}                                                   /* set_active_interface */
```

preferences.h

```
/* Etherape
 * Copyright (C) 2000 Juan Toledo
 * $Id: preferences.h,v 1.5 2001/05/05 01:05:16 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <gtk/gtk.h>
#include "globals.h"
#include "math.h"

/* Extern functions */
extern void save_config (gchar * prefix);

void on_preferences1_activate (GtkMenuItem * menuitem, gpointer user_data);
void on_averaging_spin_adjustment_changed (GtkAdjustment * adj);
void on_refresh_spin_adjustment_changed (GtkAdjustment * adj,
                                         GtkWidget * canvas);
void on_node_radius_slider_adjustment_changed (GtkAdjustment * adj);
void on_link_width_slider_adjustment_changed (GtkAdjustment * adj);
void on_node_to_spin_adjustment_changed (GtkAdjustment * adj);
void on_gui_node_to_spin_adjustment_changed (GtkAdjustment * adj);
void on_link_to_spin_adjustment_changed (GtkAdjustment * adj);
void on_font_button_clicked (GtkButton * button, gpointer user_data);
void on_ok_button1_clicked (GtkButton * button, gpointer user_data);
void on_cancel_button1_clicked (GtkButton * button, gpointer user_data);
void on_apply_button1_clicked (GtkButton * button, gpointer user_data);
void on_size_mode_menu_selected (GtkMenuShell * menu_shell, gpointer data);
void on_node_size_optionmenu_selected (GtkMenuShell * menu_shell,
                                       gpointer data);
void on_stack_level_menu_selected (GtkMenuShell * menu_shell, gpointer data);
void on_save_pref_button_clicked (GtkButton * button, gpointer user_data);
void on_diagram_only_toggle_toggled (GtkToggleButton * togglebutton,
                                     gpointer user_data);
void on_filter_entry_changed (GtkEditable * editable, gpointer user_data);
void on_apply_pref_button_clicked (GtkButton * button, gpointer user_data);
void on_ok_pref_button_clicked (GtkButton * button, gpointer user_data);
void on_cancel_pref_button_clicked (GtkButton * button, gpointer user_data);
```

preferences.c

```
/* EtherApe
 * Copyright (C) 2001 Juan Toledo
 * $Id: preferences.c,v 1.9 2001/05/12 11:17:18 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include "preferences.h"

void
on_preferences1_activate (GtkMenuItem * menuitem, gpointer user_data)
{
  GtkEditable *entry;
  gint position = 0;

  entry = GTK_EDITABLE (glade_xml_get_widget (xml, "filter_entry"));
  gtk_editable_delete_text (entry, 0, -1);
  if (filter)
    gtk_editable_insert_text (entry, filter, strlen (filter), &position);
  else
    gtk_editable_insert_text (entry, "", 0, &position);
  gtk_widget_show (diag_pref);
  gdk_window_raise (diag_pref->window);
}

void
on_node_radius_slider_adjustment_changed (GtkAdjustment * adj)
{

  node_radius_multiplier = exp ((double) adj->value * log (10));
  g_log (G_LOG_DOMAIN, G_LOG_LEVEL_DEBUG,
                        _("Adjustment value: %g. Radius multiplier %g"),
                        adj->value, node_radius_multiplier);

}

void
on_link_width_slider_adjustment_changed (GtkAdjustment * adj)
{

  link_width_multiplier = exp ((double) adj->value * log (10));
  g_log (G_LOG_DOMAIN, G_LOG_LEVEL_DEBUG,
                        _("Adjustment value: %g. Radius multiplier %g"),
                        adj->value, link_width_multiplier);

}

void
on_averaging_spin_adjustment_changed (GtkAdjustment * adj)
{
  averaging_time = adj->value;                  /* Control and value in ms */
}

void
on_refresh_spin_adjustment_changed (GtkAdjustment * adj, GtkWidget * canvas)
{
  refresh_period = adj->value;
  /* When removing the source (which could either be an idle or a timeout
   * function, I'm also forcing the callback for the corresponding
   * destroying function, which in turn will install a timeout or idle
   * function using the new refresh_period. It might take a while for it
   * to settle down, but I think it works now */
  g_source_remove (diagram_timeout);
}

void
on_node_to_spin_adjustment_changed (GtkAdjustment * adj)
{
  node_timeout_time = adj->value;               /* Control and value in ms */
}                                               /* on_node_to_spin_adjustment_changed */

void
on_gui_node_to_spin_adjustment_changed (GtkAdjustment * adj)
{
  gui_node_timeout_time = adj->value;           /* Control and value in ms */
}                                               /* on_gui_node_to_spin_adjustment_changed */
```

```
void
on_link_to_spin_adjustment_changed (GtkAdjustment * adj)
{
 link_timeout_time = adj->value;              /* Control and value in ms */
}

void
on_font_button_clicked (GtkButton * button, gpointer user_data)
{
 static GtkWidget *fontsel = NULL;
 if (!fontsel)
   fontsel = glade_xml_get_widget (xml, "fontselectiondialog1");
 gtk_font_selection_dialog_set_font_name (GTK_FONT_SELECTION_DIALOG
                                                                    (fontsel), fontname);
 gtk_widget_show (fontsel);
}


void
on_ok_button1_clicked (GtkButton * button, gpointer user_data)
{
 GtkWidget *fontsel;
 gchar *str;
 fontsel = glade_xml_get_widget (xml, "fontselectiondialog1");
 str =
   gtk_font_selection_dialog_get_font_name (GTK_FONT_SELECTION_DIALOG
                                                                    (fontsel));
 if (str)
   {
    if (fontname)
                g_free (fontname);
    fontname = g_strdup (str);
    g_free (str);
    need_reposition = TRUE;
   }

 gtk_widget_hide (fontsel);
}


void
on_cancel_button1_clicked (GtkButton * button, gpointer user_data)
{
 GtkWidget *fontsel;
 fontsel = glade_xml_get_widget (xml, "fontselectiondialog1");
 gtk_widget_hide (fontsel);
}                                            /* on_cancel_button1_clicked */


void
on_apply_button1_clicked (GtkButton * button, gpointer user_data)
{
 GtkWidget *fontsel;
 gchar *str;

 fontsel = glade_xml_get_widget (xml, "fontselectiondialog1");
 str =
   gtk_font_selection_dialog_get_font_name (GTK_FONT_SELECTION_DIALOG
                                                                    (fontsel));
 if (str)
   {
    if (fontname)
                g_free (fontname);
    fontname = g_strdup (str);
    g_free (str);
    need_reposition = TRUE;
   }
}                                            /* on_apply_button1_clicked */

void
on_size_mode_menu_selected (GtkMenuShell * menu_shell, gpointer data)
{
 GtkWidget *active_item;

 active_item = gtk_menu_get_active (GTK_MENU (menu_shell));
 /* Beware! Size mode is an enumeration. The menu options
  * must much the enumaration values */
 size_mode = g_list_index (menu_shell->children, active_item);
}                                            /* on_size_mode_menu_selected */

void
on_node_size_optionmenu_selected (GtkMenuShell * menu_shell, gpointer data)
{
 GtkWidget *active_item;

 active_item = gtk_menu_get_active (GTK_MENU (menu_shell));
 /* Beware! Size mode is an enumeration. The menu options
  * must much the enumaration values */
 node_size_variable = g_list_index (menu_shell->children, active_item);

}                                            /* on_node_size_optionmenu_selected */
```

```
void
on_stack_level_menu_selected (GtkMenuShell * menu_shell, gpointer data)
{
 GtkWidget *active_item;

 active_item = gtk_menu_get_active (GTK_MENU (menu_shell));
 stack_level = g_list_index (menu_shell->children, active_item);

 delete_gui_protocols ();

}                                               /* on_stack_level_menu_selected */

void
on_save_pref_button_clicked (GtkButton * button, gpointer user_data)
{
 save_config ("/Etherape/");
}                                               /* on_save_pref_button_clicked */


void
on_diagram_only_toggle_toggled (GtkToggleButton * togglebutton,
                                                gpointer user_data)
{

 diagram_only = gtk_toggle_button_get_active (togglebutton);
 need_reposition = TRUE;
}                                               /* on_diagram_only_toggle_toggled */

void
on_group_unk_check_toggled (GtkToggleButton * togglebutton,
                                            gpointer user_data)
{
 enum status_t old_status = status;

 if ((status == PLAY) || (status == PAUSE))
   gui_stop_capture ();

 group_unk = gtk_toggle_button_get_active (togglebutton);

 if (old_status == PLAY)
   gui_start_capture ();
}                                               /* on_group_unk_check_toggled */

void
on_ok_pref_button_clicked (GtkButton * button, gpointer user_data)
{
 on_apply_pref_button_clicked (button, NULL);
 gtk_widget_hide (diag_pref);
}                                               /* on_ok_pref_button_clicked */

void
on_apply_pref_button_clicked (GtkButton * button, gpointer user_data)
{
 GtkWidget *widget = NULL;

 widget = glade_xml_get_widget (xml, "filter_entry");
 on_filter_entry_changed (GTK_EDITABLE (widget), NULL);
 widget = glade_xml_get_widget (xml, "filter_gnome_entry");

 update_history (GNOME_ENTRY (widget), filter, FALSE);
}                                               /* on_apply_pref_button_clicked */

void
on_cancel_pref_button_clicked (GtkButton * button, gpointer user_data)
{
 gtk_widget_hide (diag_pref);
}                                               /* on_cancel_pref_button_clicked */

/* Makes a new filter */
void
on_filter_entry_changed (GtkEditable * editable, gpointer user_data)
{
 gchar *str;
 /* TODO should make sure that for each mode the filter is set up
  * correctly */
 str = gtk_editable_get_chars (editable, 0, -1);
 if (filter)
   g_free (filter);
 filter = g_strdup (str);
 g_free (str);
 /* TODO We should look at the error code from set_filter and pop
  * up a window accordingly */
 set_filter (filter, NULL);
}                                               /* on_filter_entry_changed */
```

info_windows.h

```
/* Etherape
 * Copyright (C) 2000 Juan Toledo
 * $Id: info_windows.h,v 1.5 2001/05/10 09:25:58 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

typedef struct
{
  guint8 *node_id;
  GtkWidget *window;
}
node_info_window_t;

typedef struct
{
  gchar *prot_name;
  GtkWidget *window;
}
prot_info_window_t;

GList *node_info_windows = NULL;
GList *prot_info_windows = NULL;

static GList *info_protocols = NULL;

static guint prot_clist_sort_column = 0;
static gboolean prot_clist_reverse_sort = FALSE;


static void update_prot_info_windows (void);

gboolean on_prot_table_button_press_event (GtkWidget * widget,
                                           GdkEventButton * event,
                                           gpointer user_data);
static void update_node_info_window (node_info_window_t * node_info_window);
static void update_prot_info_window (prot_info_window_t * prot_info_window);
static gint node_info_compare (gconstpointer a, gconstpointer b);
static gint prot_info_compare (gconstpointer a, gconstpointer b);

void on_node_info_delete_event (GtkWidget * node_info, gpointer user_data);
void on_prot_info_delete_event (GtkWidget * node_info, gpointer user_data);
static void prot_clist_button_clicked (GtkButton * button,
                                       gpointer func_data);
static gint prot_window_compare (GtkCList * clist, gconstpointer p1,
                                 gconstpointer p2);
static void create_prot_info_window (protocol_t * protocol);

static gchar *timeval_to_str (struct timeval tv);
```

info_windows.c

```
/* EtherApe
 * Copyright (C) 2001 Juan Toledo
 * $Id: info_windows.c,v 1.10 2001/06/22 08:39:30 toledo Exp $
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include "globals.h"
#include "info_windows.h"
#include <math.h>

guint
update_info_windows (void)
{
 static GtkWidget *protocols_window = NULL;

 if (!protocols_window)
   protocols_window = glade_xml_get_widget (xml, "protocols_window");

 if (status == PAUSE)
   return TRUE;

 gettimeofday (&now, NULL);

 /* Update the protocols window if it is being displayed */
 if (GTK_WIDGET_VISIBLE (protocols_window))
   update_protocols_window ();

 update_node_info_windows ();
 update_prot_info_windows ();

 return TRUE;                                   /* Keep on calling this function */

}                                               /* update_info_windows */

void
update_protocols_window (void)
{
 GList *item = NULL;
 protocol_t *legend_protocol = NULL, *info_protocol = NULL, *protocol = NULL;
 static struct timeval last_updated = { 0, 0 };
 struct timeval diff;
 static GtkWidget *prot_clist = NULL;
 static gboolean clist_initiated = FALSE;
 GtkWidget *widget = NULL;
 gchar *row[5] = { NULL, NULL, NULL, NULL }, *str = NULL;
 gint i;

 if (!prot_clist)
   prot_clist = glade_xml_get_widget (xml, "prot_clist");

 if (!clist_initiated)
   {
    /* set the sorting function */
    gtk_clist_set_compare_func (GTK_CLIST (prot_clist),
                                prot_window_compare);
    /* set the callbacks for the column titles */
    for (i = 0; i < GTK_CLIST (prot_clist)->columns; i++)
                {
                 widget =
                   (gtk_clist_get_column_widget (GTK_CLIST (prot_clist), i))->parent;
                 gtk_signal_connect (GTK_OBJECT (widget), "clicked",
                                     GTK_SIGNAL_FUNC (prot_clist_button_clicked),
                                     GINT_TO_POINTER (i));
                }

    clist_initiated = TRUE;
   }

 gtk_clist_freeze (GTK_CLIST (prot_clist));

 /* Add any new protocols in the legend */
 item = legend_protocols;
```

```
 while (item)
   {
    legend_protocol = item->data;
    if (!g_list_find_custom (info_protocols, legend_protocol->name,
                                                protocol_compare))
                  {
                   protocol = g_malloc (sizeof (protocol_t));
                   protocol->name = g_strdup (legend_protocol->name);
                   protocol->last_heard.tv_sec = protocol->last_heard.tv_usec = 0;
                   info_protocols = g_list_prepend (info_protocols, protocol);
                   row[0] = protocol->name;
                   row[1] = row[2] = row[3] = row[4] = "";
                   gtk_clist_prepend (GTK_CLIST (prot_clist), row);
                   gtk_clist_set_row_data (GTK_CLIST (prot_clist), 0, protocol);
                  }
    item = item->next;
   }

 /* Delete protocols not in the legend */
 for (i = GTK_CLIST (prot_clist)->rows - 1; i >= 0; i--)
   {
    protocol = gtk_clist_get_row_data (GTK_CLIST (prot_clist), i);
    if (!g_list_find_custom (legend_protocols, protocol->name,
                                                protocol_compare))
                  {
                   info_protocols = g_list_remove (info_protocols, protocol);
                   g_free (protocol->name);
                   g_free (protocol);
                   gtk_clist_remove (GTK_CLIST (prot_clist), i);
                  }
   }

 /* Update rows */
 for (i = GTK_CLIST (prot_clist)->rows - 1; i >= 0; i--)
   {
    if (!
                   (info_protocol =
                    gtk_clist_get_row_data (GTK_CLIST (prot_clist), i)))
                  {
                   g_my_critical
                     ("Unable to extract protocol structure from table in
update_protocols_window");
                   return;
                  }

    if (!
                   (item =
                    g_list_find_custom (protocols[stack_level], info_protocol->name,
                                                protocol_compare)))
                  {
                   g_my_critical
                     ("Global protocol not found in update_protocols_window");
                   return;
                  }

    protocol = item->data;
    diff =
                  substract_times (info_protocol->last_heard, protocol->last_heard);
    if ((diff.tv_usec < 0) || (diff.tv_sec < 0))
                  {
                   info_protocol->accumulated = protocol->accumulated;
                   info_protocol->last_heard = protocol->last_heard;
                   info_protocol->n_packets = protocol->n_packets;

                   str = traffic_to_str (protocol->accumulated, FALSE);
                   gtk_clist_set_text (GTK_CLIST (prot_clist), i, 2, str);

                   str = g_strdup_printf ("%d", info_protocol->n_packets);
                   gtk_clist_set_text (GTK_CLIST (prot_clist), i, 4, str);
                   g_free (str);

                  }
    /*str = ctime (&(protocol->last_heard.tv_sec)); */
    info_protocol->average = protocol->average;
    str = traffic_to_str (protocol->average, TRUE);
    gtk_clist_set_text (GTK_CLIST (prot_clist), i, 1, str);
    str = timeval_to_str (protocol->last_heard);
    gtk_clist_set_text (GTK_CLIST (prot_clist), i, 3, str);

   }

  gtk_clist_sort (GTK_CLIST (prot_clist));

  gtk_clist_thaw (GTK_CLIST (prot_clist));

#if 0
  /* Clean the list */
  prot_clist = glade_xml_get_widget (xml, "prot_clist");
  gtk_clist_clear (GTK_CLIST (prot_clist));
  /* Fill with data from legend_protocols */
  /* substract_times */
  item = legend_protocols;
```

```
  while (item)
    {
     protocol = item->data;
     capture_item = g_list_find_custom (protocols[stack_level],
                                                                protocol->name,
protocol_compare);
     protocol = capture_item->data;
     row[0] = protocol->name;
     row[1] = g_strdup (traffic_to_str (protocol->average, TRUE));
     row[2] = g_strdup (traffic_to_str (protocol->accumulated, FALSE));
     gtk_clist_prepend (GTK_CLIST (prot_clist), row);
     g_free (row[1]);
     g_free (row[2]);
     item = item->next;
    }

#endif
  last_updated = now;
}                                                   /* update_protocols_window */

void
create_node_info_window (canvas_node_t * canvas_node)
{
 GtkWidget *window;
 node_info_window_t *node_info_window;
 GladeXML *xml_info_window;
 GtkWidget *widget;
 GList *list_item;
 /* If there is already a window, we don't need to create it again */
 if (!(list_item =
              g_list_find_custom (node_info_windows,
                                  canvas_node->canvas_node_id, node_info_compare)))
   {
    xml_info_window =
              glade_xml_new (GLADEDIR "/" ETHERAPE_GLADE_FILE, "node_info");
    if (!xml_info_window)
              {
               g_error (_("We could not load the interface! (%s)"),
                        GLADEDIR "/" ETHERAPE_GLADE_FILE);
               return;
              }
    glade_xml_signal_autoconnect (xml_info_window);
    window = glade_xml_get_widget (xml_info_window, "node_info");
    gtk_widget_show (window);
    widget = glade_xml_get_widget (xml_info_window, "node_info_name_label");
    gtk_object_set_data (GTK_OBJECT (window), "name_label", widget);
    widget =
              glade_xml_get_widget (xml_info_window,
                                    "node_info_numeric_name_label");
    gtk_object_set_data (GTK_OBJECT (window), "numeric_name_label", widget);
    widget = glade_xml_get_widget (xml_info_window, "node_info_average");
    gtk_object_set_data (GTK_OBJECT (window), "average", widget);
    widget = glade_xml_get_widget (xml_info_window, "node_info_average_in");
    gtk_object_set_data (GTK_OBJECT (window), "average_in", widget);
    widget =
              glade_xml_get_widget (xml_info_window, "node_info_average_out");
    gtk_object_set_data (GTK_OBJECT (window), "average_out", widget);
    widget =
              glade_xml_get_widget (xml_info_window, "node_info_accumulated");
    gtk_object_set_data (GTK_OBJECT (window), "accumulated", widget);
    widget =
              glade_xml_get_widget (xml_info_window, "node_info_accumulated_in");
    gtk_object_set_data (GTK_OBJECT (window), "accumulated_in", widget);
    widget =
              glade_xml_get_widget (xml_info_window, "node_info_accumulated_out");
    gtk_object_set_data (GTK_OBJECT (window), "accumulated_out", widget);
    gtk_object_destroy (GTK_OBJECT (xml_info_window));
    node_info_window = g_malloc (sizeof (node_info_window_t));
    node_info_window->node_id =
              g_memdup (canvas_node->canvas_node_id, node_id_length);
    gtk_object_set_data (GTK_OBJECT (window), "node_id",
                                    node_info_window->node_id);
    node_info_window->window = window;
    node_info_windows =
              g_list_prepend (node_info_windows, node_info_window);
   }
 else
   node_info_window = (node_info_window_t *) list_item->data;
 update_node_info_window (node_info_window);
 if (canvas_node && canvas_node->node)
   {
    g_my_info ("Nodes: %d. Canvas nodes: %d", g_tree_nnodes (nodes),
                        g_tree_nnodes (canvas_nodes));
    dump_node_info (canvas_node->node);
   }

}                                                   /* create_node_info_window */


void
update_node_info_windows (void)
{
```

```
  GList *list_item = NULL, *remove_item;
  node_info_window_t *node_info_window = NULL;
  static struct timeval last_time = {
    0, 0
  }
  , diff;
  diff = substract_times (now, last_time);
  /* Update info windows at most twice a second */
  if (refresh_period < 500)
    if (!(IS_OLDER (diff, 500)))
      return;
  list_item = node_info_windows;
  while (list_item)
    {
      node_info_window = (node_info_window_t *) list_item->data;
      if (status == STOP)
                    {
                      g_free (node_info_window->node_id);
                      gtk_widget_destroy (GTK_WIDGET (node_info_window->window));
                      remove_item = list_item;
                      list_item = list_item->next;
                      node_info_windows =
                        g_list_remove_link (node_info_windows, remove_item);
                    }
      else
                    {
                      update_node_info_window (node_info_window);
                      list_item = list_item->next;
                    }
    }

  last_time = now;
  return;
}                                                   /* update_node_info_windows */

/* It's called when a node info window is closed by the user
 * It has to free memory and delete the window from the list
 * of windows */
void
on_node_info_delete_event (GtkWidget * node_info, gpointer user_data)
{
  GList *item = NULL;
  guint8 *node_id = NULL;
  node_info_window_t *node_info_window = NULL;
  node_id = gtk_object_get_data (GTK_OBJECT (node_info), "node_id");
  if (!node_id)
    {
      g_my_critical (_("No node_id in on_node_info_delete_event"));
      return;
    }

  if (!(item =
                g_list_find_custom (node_info_windows, node_id, node_info_compare)))
    {
      g_my_critical (_("No node_info_window in on_node_info_delete_event"));
      return;
    }

  node_info_window = item->data;
  g_free (node_info_window->node_id);
  gtk_widget_destroy (GTK_WIDGET (node_info_window->window));
  node_info_windows = g_list_remove_link (node_info_windows, item);
}                                                   /* on_node_info_delete_event */

/* It's called when a prot info window is closed by the user
 * It has to free memory and delete the window from the list
 * of windows */
void
on_prot_info_delete_event (GtkWidget * prot_info, gpointer user_data)
{
  GList *item = NULL;
  guint8 *prot_name = NULL;
  prot_info_window_t *prot_info_window = NULL;
  prot_name = gtk_object_get_data (GTK_OBJECT (prot_info), "prot_name");
  if (!prot_name)
    {
      g_my_critical (_("No prot_name in on_prot_info_delete_event"));
      return;
    }

  if (!(item =
                g_list_find_custom (prot_info_windows, prot_name, prot_info_compare)))
    {
      g_my_critical (_("No prot_info_window in on_prot_info_delete_event"));
      return;
    }

  prot_info_window = item->data;
  g_free (prot_info_window->prot_name);
  gtk_widget_destroy (GTK_WIDGET (prot_info_window->window));
  prot_info_windows = g_list_remove_link (prot_info_windows, item);
}                                                   /* on_prot_info_delete_event */
```

```
static void
update_node_info_window (node_info_window_t * node_info_window)
{
 guint8 *node_id = NULL;
 node_t *node = NULL;
 GtkWidget *window = NULL;
 GtkWidget *widget = NULL;
 node_id = node_info_window->node_id;
 window = node_info_window->window;
 if (!(node = g_tree_lookup (nodes, node_id)))
   {
    widget =
                gtk_object_get_data (GTK_OBJECT (window), "numeric_name_label");
    gtk_label_set_text (GTK_LABEL (widget), _("No info available"));
    widget = gtk_object_get_data (GTK_OBJECT (window), "average");
    gtk_label_set_text (GTK_LABEL (widget), "X");
    widget = gtk_object_get_data (GTK_OBJECT (window), "average_in");
    gtk_label_set_text (GTK_LABEL (widget), "X");
    widget = gtk_object_get_data (GTK_OBJECT (window), "average_out");
    gtk_label_set_text (GTK_LABEL (widget), "X");
    widget = gtk_object_get_data (GTK_OBJECT (window), "accumulated");
    gtk_label_set_text (GTK_LABEL (widget), "X");
    widget = gtk_object_get_data (GTK_OBJECT (window), "accumulated_in");
    gtk_label_set_text (GTK_LABEL (widget), "X");
    widget = gtk_object_get_data (GTK_OBJECT (window), "accumulated_out");
    gtk_label_set_text (GTK_LABEL (widget), "X");
    gtk_container_queue_resize (GTK_CONTAINER (node_info_window->window));
    return;
   }

 gtk_window_set_title (GTK_WINDOW (window), node->name->str);
 widget = gtk_object_get_data (GTK_OBJECT (window), "name_label");
 gtk_label_set_text (GTK_LABEL (widget), node->name->str);
 widget = gtk_object_get_data (GTK_OBJECT (window), "numeric_name_label");
 gtk_label_set_text (GTK_LABEL (widget), node->numeric_name->str);
 widget = gtk_object_get_data (GTK_OBJECT (window), "average");
 gtk_label_set_text (GTK_LABEL (widget),
                     traffic_to_str (node->average, TRUE));
 widget = gtk_object_get_data (GTK_OBJECT (window), "average_in");
 gtk_label_set_text (GTK_LABEL (widget),
                     traffic_to_str (node->average_in, TRUE));
 widget = gtk_object_get_data (GTK_OBJECT (window), "average_out");
 gtk_label_set_text (GTK_LABEL (widget),
                     traffic_to_str (node->average_out, TRUE));
 widget = gtk_object_get_data (GTK_OBJECT (window), "accumulated");
 gtk_label_set_text (GTK_LABEL (widget),
                     traffic_to_str (node->accumulated, FALSE));
 widget = gtk_object_get_data (GTK_OBJECT (window), "accumulated_in");
 gtk_label_set_text (GTK_LABEL (widget),
                     traffic_to_str (node->accumulated_in, FALSE));
 widget = gtk_object_get_data (GTK_OBJECT (window), "accumulated_out");
 gtk_label_set_text (GTK_LABEL (widget),
                     traffic_to_str (node->accumulated_out, FALSE));
 gtk_container_queue_resize (GTK_CONTAINER (node_info_window->window));
}                                                   /* update_node_info_window */

void
toggle_protocols_window (void)
{
 static GtkWidget *protocols_check = NULL;
 if (!protocols_check)
   protocols_check = glade_xml_get_widget (xml, "protocols_check");
 gtk_menu_item_activate (GTK_MENU_ITEM (protocols_check));
}                                                   /* toggle_protocols_window */

void
on_protocols_check_activate (GtkCheckMenuItem * menuitem, gpointer user_data)
{
 static GtkWidget *protocols_window = NULL;
 if (!protocols_window)
   protocols_window = glade_xml_get_widget (xml, "protocols_window");
 if (menuitem->active)
   {
    gtk_widget_show (protocols_window);
    gdk_window_raise (protocols_window->window);
    update_protocols_window ();
   }
 else
   gtk_widget_hide (protocols_window);
}                                                   /* on_protocols_check_activate */

void
on_prot_column_view_activate (GtkCheckMenuItem * menuitem, gpointer user_data)
{
 static GtkWidget *prot_clist = NULL;
 guint column;

 if (!prot_clist)
   prot_clist = glade_xml_get_widget (xml, "prot_clist");

 if (!sscanf ((gchar *) user_data, "%d", &column))
```

```
    {
     g_warning ("Unable to decode column in on_prot_column_view_activate");
     return;
    }

  gtk_clist_set_column_visibility (GTK_CLIST (prot_clist),
                                                        column, menuitem->active);

}                                                   /* on_prot_column_view_activate */

/* Displays the protocols window when the legend is double clicked */
gboolean
on_prot_table_button_press_event (GtkWidget * widget,
                                                        GdkEventButton * event, gpointer
user_data)
{
  switch (event->type)
    {
    case GDK_2BUTTON_PRESS:
      toggle_protocols_window ();
      return TRUE;
    default:
    }

  return FALSE;
}                                                   /* on_prot_table_button_press_event */

static void
prot_clist_button_clicked (GtkButton * button, gpointer func_data)
{
  guint column = GPOINTER_TO_INT (func_data);

  /* If clicked a second time, reverse the order of sorting */
  if (column == prot_clist_sort_column)
    {
      if (prot_clist_reverse_sort == TRUE)
                prot_clist_reverse_sort = FALSE;
      else
                prot_clist_reverse_sort = TRUE;
    }

  prot_clist_sort_column = column;

  update_protocols_window ();
}                                                   /* prot_clist_button_clicked */
```

```
void
on_prot_clist_select_row (GtkCList * clist,
                                                        gint row,
                                                        gint column, GdkEvent * event, gpointer user_data)
{
  protocol_t *protocol = NULL;

  if (!event)
    return;

  switch (event->type)
    {
    case GDK_2BUTTON_PRESS:
      protocol = gtk_clist_get_row_data (clist, row);
      create_prot_info_window (protocol);
    default:
    }
}                                                   /* on_prot_clist_select_row */

static void
create_prot_info_window (protocol_t * protocol)
{
  GtkWidget *window;
  prot_info_window_t *prot_info_window;
  GladeXML *xml_info_window;
  GtkWidget *widget;
  GList *list_item;

  /* If there is already a window, we don't need to create it again */
  if (!(list_item =
                g_list_find_custom (prot_info_windows,
                                                        protocol->name, prot_info_compare)))
    {
      xml_info_window =
                glade_xml_new (GLADEDIR "/" ETHERAPE_GLADE_FILE, "prot_info");
      if (!xml_info_window)
                {
                  g_error (_("We could not load the interface! (%s)"),
                                GLADEDIR "/" ETHERAPE_GLADE_FILE);
                  return;
                }
      glade_xml_signal_autoconnect (xml_info_window);
      window = glade_xml_get_widget (xml_info_window, "prot_info");
      gtk_widget_show (window);
      widget = glade_xml_get_widget (xml_info_window, "prot_info_name_label");
      gtk_object_set_data (GTK_OBJECT (window), "name_label", widget);
```

```
    widget =
                glade_xml_get_widget (xml_info_window, "prot_info_last_heard_label");
    gtk_object_set_data (GTK_OBJECT (window), "last_heard_label", widget);
    widget = glade_xml_get_widget (xml_info_window, "prot_info_average");
    gtk_object_set_data (GTK_OBJECT (window), "average", widget);
    widget =
                glade_xml_get_widget (xml_info_window, "prot_info_accumulated");
    gtk_object_set_data (GTK_OBJECT (window), "accumulated", widget);
    gtk_object_destroy (GTK_OBJECT (xml_info_window));

    prot_info_window = g_malloc (sizeof (prot_info_window_t));
    prot_info_window->prot_name = g_strdup (protocol->name);
    gtk_object_set_data (GTK_OBJECT (window), "prot_name",
                                            prot_info_window->prot_name);
    prot_info_window->window = window;
    prot_info_windows =
                g_list_prepend (prot_info_windows, prot_info_window);
  }
 else
  prot_info_window = (prot_info_window_t *) list_item->data;

 update_prot_info_window (prot_info_window);

}                                                   /* create_prot_info_window */


void
update_prot_info_windows (void)
{
 GList *list_item = NULL, *remove_item;
 prot_info_window_t *prot_info_window = NULL;

 list_item = prot_info_windows;
 while (list_item)
  {
   prot_info_window = (prot_info_window_t *) list_item->data;
   if (status == STOP)
                {
                 g_free (prot_info_window->prot_name);
                 gtk_widget_destroy (GTK_WIDGET (prot_info_window->window));
                 remove_item = list_item;
                 list_item = list_item->next;
                 prot_info_windows =
                   g_list_remove_link (prot_info_windows, remove_item);
                }
   else
                {
                 update_prot_info_window (prot_info_window);
                 list_item = list_item->next;
                }
  }

 return;
}                                                   /* update_prot_info_windows */


static void
update_prot_info_window (prot_info_window_t * prot_info_window)
{
 guint8 *prot_name = NULL;
 protocol_t *prot = NULL;
 GtkWidget *window = NULL;
 GtkWidget *widget = NULL;
 GList *item = NULL;
 prot_name = prot_info_window->prot_name;

 window = prot_info_window->window;
 if (!
    (item =
     g_list_find_custom (info_protocols, prot_name, protocol_compare)))
  {
   widget = gtk_object_get_data (GTK_OBJECT (window), "average");
   gtk_label_set_text (GTK_LABEL (widget), "X");
   widget = gtk_object_get_data (GTK_OBJECT (window), "accumulated");
   gtk_label_set_text (GTK_LABEL (widget), "X");
   gtk_container_queue_resize (GTK_CONTAINER (prot_info_window->window));
   return;
  }

 prot = item->data;

 gtk_window_set_title (GTK_WINDOW (window), prot->name);
 widget = gtk_object_get_data (GTK_OBJECT (window), "name_label");
 gtk_label_set_text (GTK_LABEL (widget), prot->name);
 widget = gtk_object_get_data (GTK_OBJECT (window), "last_heard_label");
 gtk_label_set_text (GTK_LABEL (widget), timeval_to_str (prot->last_heard));
 widget = gtk_object_get_data (GTK_OBJECT (window), "average");
 gtk_label_set_text (GTK_LABEL (widget),
                                        traffic_to_str (prot->average, TRUE));
 widget = gtk_object_get_data (GTK_OBJECT (window), "accumulated");
 gtk_label_set_text (GTK_LABEL (widget),
                        traffic_to_str (prot->accumulated, FALSE));
```

```
  gtk_container_queue_resize (GTK_CONTAINER (prot_info_window->window));

}                                                   /* update_prot_info_window */


/* Comparison function used to sort the clist */
static gint
prot_window_compare (GtkCList * clist, gconstpointer p1, gconstpointer p2)
{
  gint ret;
  gdouble t1, t2;
  struct timeval time1, time2, diff;

  protocol_t *prot1, *prot2;
  prot1 = ((const GtkCListRow *) p1)->data;
  prot2 = ((const GtkCListRow *) p2)->data;

  switch (prot_clist_sort_column)
    {
    case 0:
      ret = strcmp (prot1->name, prot2->name);
      break;
    case 1:
      t1 = prot1->average;
      t2 = prot2->average;
      if (t1 == t2)
                ret = 0;
      else if (t1 < t2)
                ret = -1;
      else
                ret = 1;
      break;
    case 2:
      t1 = prot1->accumulated;
      t2 = prot2->accumulated;
      if (t1 == t2)
                ret = 0;
      else if (t1 < t2)
                ret = -1;
      else
                ret = 1;
      break;
    case 3:
      time1 = prot1->last_heard;
      time2 = prot2->last_heard;
      diff = substract_times (time1, time2);
      if ((diff.tv_sec == 0) && (diff.tv_usec == 0))
                ret = 0;
      else if ((diff.tv_sec < 0) || (diff.tv_usec < 0))
                ret = -1;
      else
                ret = 1;
      break;
    case 4:
      if (prot1->n_packets == prot2->n_packets)
                ret = 0;
      else if (prot1->n_packets < prot2->n_packets)
                ret = -1;
      else
                ret = 1;
      break;
    default:
      ret = 0;
    }

  if (prot_clist_reverse_sort)
    ret = -ret;

  return ret;
}                                                   /* prot_window_compare */

/* Comparison function used to compare node_info_windows */
static gint
node_info_compare (gconstpointer a, gconstpointer b)
{
  g_assert (a != NULL);
  g_assert (b != NULL);
  return memcmp (((node_info_window_t *) a)->node_id, (guint8 *) b,
                                        node_id_length);
}

/* Comparison function used to compare prot_info_windows */
static gint
prot_info_compare (gconstpointer a, gconstpointer b)
{
  g_assert (a != NULL);
  g_assert (b != NULL);
  return strcmp (((prot_info_window_t *) a)->prot_name, (guint8 *) b);
}                                                   /* prot_info_compare */

static gchar *
timeval_to_str (struct timeval last_heard)
```

```
{
 static gchar *str = NULL;
 struct timeval diff;
 struct tm broken_time;

 if (str)
   g_free (str);

 diff = substract_times (now, last_heard);
 if (!localtime_r ((time_t *) & (last_heard.tv_sec), &broken_time))
   {
     g_my_critical ("Time conversion failed in timeval_to_str");
     return NULL;
   }

 if (diff.tv_sec <= 60)
   {
     /* Meaning "n seconds" ago */
     str = g_strdup_printf (_("%d\" ago"), (int) diff.tv_sec);
   }
 else if (diff.tv_sec < 600)
   {
     /* Meaning "m minutes, n seconds ago" */
     str =
                 g_strdup_printf (_("%d'%d\" ago"),
                                  (int) floor ((double) diff.tv_sec / 60),
                                  (int) diff.tv_sec % 60);
   }
 else if (diff.tv_sec < 3600 * 24)
   {
     str =
                 g_strdup_printf ("%d:%d", broken_time.tm_hour, broken_time.tm_min);
   }
 else
   {
     /* Watch out! The first is month, the second day of the month */
     str = g_strdup_printf (_("%d/%d %d:%d"),
                                                 broken_time.tm_mon, broken_time.tm_mday,
                                                 broken_time.tm_hour, broken_time.tm_min);
   }

 return str;
}                                                       /* timeval_to_str */
```