Xniff: Analizador gráfico de red.

Estudiante: Daniel Lerch Hostalot
ETIS

Consultor: David Carrera Pérez
18 de Junio del 2004

Dedicatoria:

A mi querida Elisabet que ha agotado la poca paciencia que le quedaba mientras acababa éste proyecto.

Xniff es una analizador gráfico de red desarrollado como proyecto final de carrera de la Ingeniería Técnica en Informática de Sistemas, en la Universitat Oberta de Catalunya.

Éste analizador gráfico o sniffer esta diseñado para funcionar sobre el sistema operativo GNU/Linux, aunque el desarrollo ha sido realizado mediante las APIs estándar de UNIX, por lo que debería funcionar correctamente en la mayoría de sistemas de ésta familia.

Xniff identifica los protocolos Ethernet, ARP, IP, ICMP, TCP y UDP, aunque captura todo tipo de paquetes. Además dispone de un sistema de volcado en hexadecimal y diversas herramientas de análisis.

Xniff funciona únicamente de forma gráfica, utilizando para ello las librerías GTK+ (Librerías base del entorno GNOME).

Ha sido desarrollado utilizando el lenguaje de programación C.

ÍNDICE

ÍNDICE DE FIGURAS

Estructura de la memoria

1. Análisis de redes: el Sniffer

1.1 ¿Qué es un Sniffer?

El término sniffer proviene del inglés “sniffer dog” (perro rastreador). En informática se utiliza éste término para referirse a la herramienta utilizada para el análisis de redes. Un sniffer es, por lo tanto, una herramienta capaz de capturar toda la información transmitida por una red.

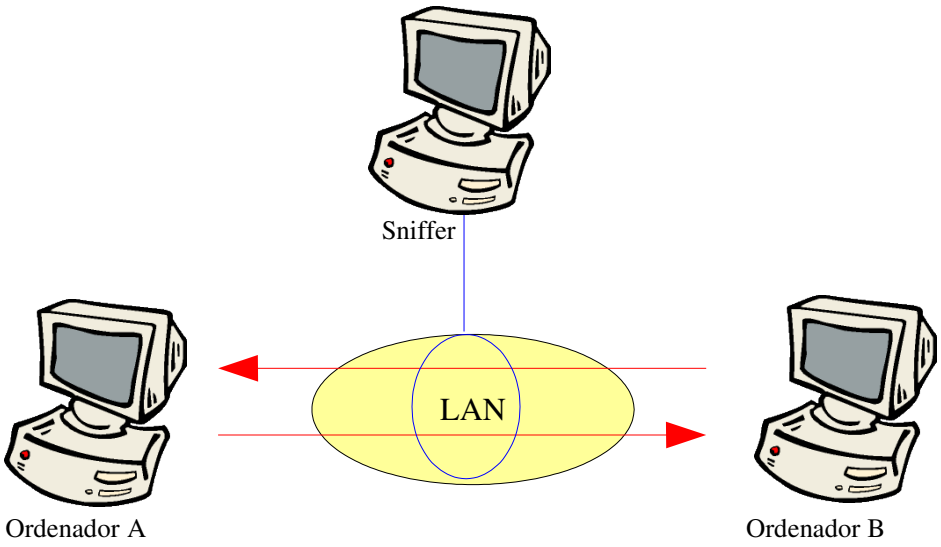


Figura 1.1.1: Un sniffer captura la transmisión entre el ordenador A y el ordenador B.

Dada la gran cantidad de paquetes que pueden pasar por una red en una fracción de tiempo, es habitual que los sniffers dispongan de capacidades avanzadas de filtrado. Éstas les ofrecen la posibilidad de capturar únicamente los paquetes deseados.

Algunos sniffers incluyen, además, la posibilidad de realizar un análisis posterior de la información capturada. Normalmente estos sniffers son gráficos.

1.2 Sniffers Open Source

Los sniffers han sido unas herramientas muy utilizadas desde el inicio de las redes de computadoras. Por este motivo existen una cantidad enorme de sniffers tanto comerciales como sniffers Open Source gratuitos. De todos estos, con toda seguridad, el más conocido es TCPDump. Una herramienta del mundo UNIX con unos cuantos años de experiencia.

1.2.1 TCPDump y PCAP:

TCPDump puede ser descargada desde su página principal http://www.tcpdump.org.

TCPDump está escrito sobre la librería de captura de paquetes PCAP. Ésta librería es multiplataforma y ha servido para llevar los sniffers open source a otros sistemas operativos como Windows.

Una característica muy interesante de esta librería y consecuentemente de TCPDump es el sistema de filtrado de paquetes del que dispone, ya que soporta un lenguaje sencillo que permite configurar todo tipo de combinaciones de filtrado.

Es importante destacar que la mayoría de sniffers Open Source están basados en la librería PCAP.

1.2.2 Ethereal

Ethereal es probablemente el sniffer gráfico más utilizado en entornos UNIX. Está basado en la librería de red PCAP, comentada en el apartado anterior y dispone de capacidad de análisis para una cantidad sorprendente de protocolos.

Puede descargarse desde http://www.ethereal.com.

Ethereal es un sniffer muy útil para los usuarios conocedores de TCPDump que buscan un entorno gráfica para el análisis de paquetes. Ethereal no dispone de un sistema gráfico de filtrado, por lo que a los usuarios que no conozcan TCPDump les resultara poco práctico, pues el sistema de filtrado esta basado en un lenguaje utilizado por PCAP.

1.2.3 Dsniff

Los Sniffers son una herramienta ideal para encontrar y solucionar problemas en la red. Nos permiten averiguar si ésta sufre un ataque, si esta saturada, si una máquina envía paquetes extraños ...

Pero sin duda, los que más provecho han encontrado en esta herramienta han sido los crackers (intrusos, pirátas informáticos, ...). La posibilidad de acceder al contenido de la red hace posible, entre otros ataques, la captura de contraseñas o información confidencial.

Dsniff es un suite de herramientas relacionadas con las seguridad informática que permiten entre otras cosas: ataques MITM a SSH y SSL, espionaje de conexiones HTTP, ataques ARP poisoning, etc.

La herramienta que nos interesa del paquete Dsniff es un sniffer también llamado Dsniff. Éste sniffer se dedica única y exclusivamente a la captura de contraseñas que pasan por la red.

Más información relacionada con los sniffers y las seguridad informática en el apartado 1.4.

1.2.4 Otros

Existen otros sniffers no menos impotantes que los anteriores como: Sniffit (http://reptile.rug.ac.be/~coder/sniffit/sniffit.html) o Snort (http://www.snort.org) que además es un detector de intrusos. Cada uno de ellos con ciertas características que resultarán más o menos importantes en función de la utilización que se le pretenda dar al sniffer.

1.3 Captura de paquetes en sistemas UNIX

Existen tres posibilidades principales para la captura de paquetes de red en sistemas UNIX:

1. Uso de una librería de acceso a la red como libpcap (http://www.tcpdump.org).

Libpcap es una librería de captura de paquetes muy extendida en sistemas UNIX. Dispone de características tan interesantes como su disponibilidad en múltiples plataformas o su sistema de filtros avanzados.

2. La API del kernel del sistema operativo.

Esta API es la utilizada en el mismo kernel para el desarrollo del sistema de red. Como inconveniente principal hay que destacar que puede depender de la versión del kernel o sistema operativo.

3. La API estándar de UNIX.

API disponible en todos los sistemas operativos de la familia UNIX.

Probablemente la mejor opción para el desarrollo de un sniffer profesional es el uso de la librería PCAP. Ésta librería dispone de todo lo necesario para el desarrollo de un sniffer: Filtros de paquetes avanzados, sistema de captura de paquetes sencillo, formatos propios de volcado ... Su uso es muy sencillo y en pocas líneas de código podrías desarrollarse un sniffer completo.

El único inconveniente de esta librería es la enorme abstracción que realiza del sistema de red, por otra parte necesario para una librería multiplataforma. Éste hecho hace que deje de ser interesante su uso en proyectos en los que se desea un acceso total a la red y a la API del sistema operativo.

1.4 Los sniffers y la seguridad informática

1.4.1 El sniffer como herramienta de ataque pasivo

Como se ha introducido en el apartado 1.2.3 Dsniff, los sniffers están muy relacionados con la seguridad informática. Un buen sniffer es una de las herramientas básicas de todo cracker (intruso, pirata informático, ...).

Cuando un intruso consigue acceder a una red después de haber comprometido un máquina, uno de los siguientes pasos suele ser dejar un sniffer que vaya capturando el tráfico de red. Ésto le proporcionará al intruso contraseñas y otra información confidencial. Además le permitirá descubrir otras máquinas en la red y su función en la misma.

1.4.2 La importancia del cifrado en las comunicaciones

En la actualidad, cada vez se está haciendo más necesario el cifrado de las comunicaciones: Substituir protocolos como telnet o ftp por protocolos cifrados como SSH, HTTP por HTTPS, configurar VPNs, etc.

Acceder al tráfico de red puede resultar más sencillo de lo que parece inicialmente. Y una vez se tiene acceso al tráfico de red, si la comunicación no está cifrada, el acceso total al sistema es solo cuestión de tiempo.

Las redes Wireless son un ejemplo de necesidad de cifrado en las comunicaciones. Cualquiera con un ordenador portátil provisto de una tarjeta de red inalámbrica, puede capturar el tráfico de esa red.

Éstos sniffers suelen representar un grave problema de seguridad en las redes de computadoras y no se dispone de una forma sencilla de detectarlos.

La posibilidad de la detección de un sniffer es un tema que ha sido ampliamente estudiado:

Si se dispone acceso a la maquina que se supone que tiene un sniffer instalado, su detección es sencilla. Basta con observar las interfaces de red en busca de una que esté en modo promiscuo.

Si no se dispone de acceso a la maquina, la tarea se vuelve sumamente dificil. Dado que el sniffer realiza un ataque pasivo a la red, es decir no interactúa con la misma, solo se limita a capturar paquetes.

2. Análisis y diseño de Xniff

2.1 Análisis de requisitos

2.1.1 Introducción

Inicialmente, Xniff debe funcionar en el sistema operativo GNU/Linux aunque el desarrollo se realizará con APIs estándar de UNIX y GNOME. Por lo tanto, debería funcionar en cualquier sistema operativo de tipo UNIX con un entorno de escritorio GNOME, como por ejemplo FreeBSD, NetBSD ...

El lenguaje de programación C ha sido el elegido para el desarrollo por ser el más extendido en el mundo UNIX y permitir un acceso total al API del sistema operativo.

La opción elegida para el desarrollo de Xniff es la API estándar UNIX. No se ha elegido la API del kernel de Linux porque el programa resultaría muy poco portable. Tampoco se ha elegido la librería libpcap aún y ser probablemente la mejor opción porque representa un abstracción demasiado elevada de la captura de paquetes. En este proyecto resulta interesante disponer de un acceso total al API de red y todas sus posibilidades.

Para el desarrollo de la interfaz gráfica de en GNU/Linux existen dos posibilidades principales: Las librerías Qt utilizadas por el proyecto KDE (http://www.kde.org) y Las librerías GTK+ utilizadas por el proyecto GNOME (http://www.gnome.org). Las librerías Qt son las librerías gráficas utilizadas en el entorno de escritorio KDE. Están diseñadas con tecnología OO y suelen programarse en C++. Las librerías GTK+ son las librerías gráficas utilizadas en el entorno de escritorio GNOME. Están diseñadas con tecnología OO aunque desarrolladas en C.

Las librería elegidas para el proyecto son las librerías GTK+, más adecuadas para la programación en C, lenguaje del proyecto Xniff.

2.1.2 Requisitos del sistema de captura

A nivel de captura de paquetes, Xniff debe cumplir los siguientes requisitos:

1. Establecimiento de la interfaz en modo promiscuo:

Para una correcta captura de todos los paquetes de la red, Xniff debe ser capaz de establecer la interfaz en modo promiscuo. Es decir, a la escucha de todo el trafico de red.

2. Captura de captura de paquetes Ethernet, ARP, ICMP, IP, TCP y UDP:

Dado que la cantidad de protocolos de red es inagotable se ha decido establecer unos protocolos para su análisis con Xniff. Estos son ARP, ICMP, IP, TCP y UDP.

3. Captura de paquetes a nivel de aplicación:

Xniff deberá ser capaz de mostrar el contenido de los paquetes de nivel de aplicación, aunque no se analice su contenido.

2.1.3 Requisitos de la interfaz gráfica

A continuación se listan los requisitos que debería cumplir Xniff a nivel de interfaz gráfica:

1. Interfaz de usuario intuitiva:

La interfaz gráfica de Xniff deberá ser intuitiva, al estilo de las típicas aplicaciones gráficas de GNOME.

2. Sistema de filtrado de paquetes sencillo:

Una de las dificultades que se han visto en el punto 1.2.2 referentes al sniffer Ethereal era el uso del sistema de fintrado de paquetes. Xniff deberá tener un sistema de filtrado gráfico sencillo, que aunque no disponga de la potencia de la librería PCAP permita utilizar el sniffer a un usuario desconocedor de este tipo de filtros.

3. Filtrado de cadenas ASCII o hexadecimales:

Un sistema de filtrado que se echa de menos en algunos sniffers es el filtrado de cadenas ASCII o cadenas hexadecimales. Sería interesante que en Xniff estuviese disponible.

4. Sistema de volcado hexadecimal-ASCII:

Una utilidad interesante de algunos sniffers como TCPDump y Ethereal es su sistema de volcado de paquetes en hexadecimal y ASCII que permite analizar al detalle el contenido. Xniff deberá incorporar un sistema similar.

4. Conversor entero-hexadecimal:

Lo sniffers tradicionales suelen usarse junto a una calculadora mediante la cual se realizan conversiones entre valores hexadecimales y números enteros. Xniff deberá proporcionar un sistema que realice la conversión.

5. Sistema de análisis de paquetes:

Deberá proporcionarse un sistema de análisis del paquete: búsqueda de campos, identificación de los mismos ...

2.2 Diseño del sniffer

2.2.1 APIs utilizadas

Como se ha indicado en el apartado de análisis, las APIs utilizadas por Xniff son GTK+ (como API gráfica) y la API estándar de UNIX. A continuación se comentan algunos aspectos de las mismas.

Sobre la API de UNIX y la captura de paquetes:

La API de UNIX dispone de diversas funciones para la captura de paquetes. Aún y ser una API en teoría estándar para todos los UNIX, existen algunas diferencias menores en algunos aspectos. En este proyecto nos centraremos en el funcionamiento en el sistema operativo GNU/Linux, pero hay que remarcar que podría funcionar en otros UNIX con unas modificaciones mínimas.

Inicialmente, centraremos nuestra atención en la llamada al sistema `socket()` que nos permite la creación de un socket con capacidad de captura de todos los paquetes de la red:

```
sock = socket(AF_INET, SOCK_PACKET, htons(0x3))
```

El parámetro `htons(0x3)` debería poner la tarjeta en modo promiscuo, pero en algunos sistemas no lo hace, por lo que será necesario realizar otras operaciones. Para ello necesitaremos crear un socket y establecer el flag `IFF_PROMISC` con la función `ioctl()`.

```
ifr.ifr_flags |= IFF_PROMISC;
ioctl(sd, SIOCSIFFLAGS, &ifr);
```

Una vez la interfaz está en modo promiscuo y se ha creado el socket de captura de paquetes, únicamente tendremos que leer el tráfico de red con la llamada `read()`.

```
char buffer[BUFFER_SIZE];
len = read(sock, buffer, BUFFER_SIZE);
```

Finalmente, mediante las estructuras de cabeceras adecuadas podremos acceder al contenido del paquete:

```
#include <net/ethernet.h>
struct ether_header *eth = (struct ether_header *) buffer;

struct arp_hdr {
   unsigned short int hardware;
   unsigned short int protocol;
   char hw_addr_len;
   char proto_addr_len;
   unsigned short operation;
   char src_addr[6];
   char src_ip[4];
   char dst_addr[6];
   char dst_ip[4];
};
struct arp_hdr *arp =
(struct arp_hdr *)( buffer + sizeof(struct ether_header));

#include <netinet/ip.h>
struct iphdr *ip =
(struct iphdr*) (buffer + sizeof(struct ether_header));

#include <netinet/ip_icmp.h>
struct icmphdr *icmp =
(struct icmphdr*) (buffer + sizeof(struct ether_header));

#include <netinet/tcp.h>
struct tcphdr *tcp = (struct tcphdr *)
( buffer + sizeof(struct ether_header) + sizeof(struct iphdr));

#include <netinet/udp.h>
struct udphdr *udp = (struct udphdr *)
( buffer + sizeof(struct ether_header) + sizeof(struct iphdr));
```

Por ejemplo, podríamos acceder al la versión del paquete IP con:

```
ip->version
```

Sobre GTK+:

GTK+ es una completa API para el desarrollo de interfaces de usuario desarrollada en C mediante técnicas de orientación a objetos. Es la librería sobre la que está construido el proyecto GNOME.

GTK+ dispone de una amplia colección de Widgets como botones, listas, checkbox, etc para el uso en las interfaces de usuario. Además se complementa con la librería GLIB que le proporciona un marco de trabajo independiente de plataforma con funciones para el manejo de errores, escritura de mensajes en pantalla, estructuras de datos (listas, árboles, tablas de hash,..), tipos de datos, etc.

Veamos, para empezar, el uso de GTK+ en la creación de una ventana sencilla:

```
#include <gtk/gtk.h>
GtkWidget *main_window;
gtk_init (&argc, &argv);
main_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
gtk_widget_show (main_window);
gtk_main ();
```

Inicialmente hemos creado un Widget que define la ventana principal. A continuación se inicializa GTK+ con los argumentos recibidos de main. Se crea la ventana y se hace visible. Finalmente se entra en gtk_main() un bucle infinito que se dedica a recoger los eventos lanzados sobre la ventana y a procesarlos.

Veamos ahora como empaquetar Widgets y mostrarlos en una ventana. A continuación se muestra un fragmento de código del archivo “xniff_interface_controls.c” que muestra como crear una tabla e introducir dentro algunos Widgets:

```
/* TABLE de tres filas y dos columnas */
GtkWidget *table1 = gtk_table_new (3, 2, FALSE);
gtk_widget_show (table1);
gtk_box_pack_start (GTK_BOX (main_vbox), table1, TRUE, TRUE, 0);

/* Insertamos la etiqueta Packet Field */
GtkWidget *label_packetfield = gtk_label_new ("Packet Field:");
gtk_widget_show (label_packetfield);
gtk_table_attach (GTK_TABLE (table1), label_packetfield, 0, 1, 0, 1,
                  (GtkAttachOptions) (GTK_FILL),
                  (GtkAttachOptions) (0), 0, 0);
gtk_label_set_justify (GTK_LABEL (label_packetfield), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (label_packetfield), 0, 0.5);
gtk_misc_set_padding (GTK_MISC (label_packetfield), 10, 0);

/* Insertamos la etiqueta Value */
GtkWidget *label_value = gtk_label_new ("Value:");
gtk_widget_show (label_value);
gtk_table_attach (GTK_TABLE (table1), label_value, 0, 1, 1, 2,
                  (GtkAttachOptions) (GTK_FILL),
                  (GtkAttachOptions) (0), 0, 0);
gtk_label_set_justify (GTK_LABEL (label_value), GTK_JUSTIFY_LEFT);
gtk_misc_set_alignment (GTK_MISC (label_value), 0, 0.5);
gtk_misc_set_padding (GTK_MISC (label_value), 10, 0);
```

Hemos visto que la creación de una ventana es relativamente sencillo. Aún así a medida que se empiezan a añadir más Widgets y la aplicación va creciendo (gran número de ventanas, gran cantidad de Widgets, etc) la gestión del código se complica.

2.2.2 Estructura del código fuente

En el directorio "src/" del proyecto podemos encontrar el código fuente de la aplicación. A continuación se describe con más o menos detalle la organización del código y la función de cada uno de los archivos fuente,

Archivo de proyecto Makefile:

Makefile: Archivo de compilación del proyecto.

Archivos de cabecera:

xniff.h: Cabecera principal. Se ha creado únicamente una cabecera para hacer más sencilla la gestión del código. En ella pueden encontrarse las variables globales y los prototipos de todas las funciones.

xniff_headers.h: Se ha creado un archivo especial por si era necesario personalizar las cabeceras de los paquetes. Solo ha sido necesario en los paquetes ARP.

Archivos fuente:

xniff_main.c: Archivo principal en el que se encuentra la función main() y desde donde se crea la ventana principal.

xniff_callbacks.c: Éste archivo contiene todas las funciones que se ejecutan a partir de un evento. Cada una de estas funciones se encarga de llamar a otras funciones situadas en otros archivos en función del tipo de evento recibido.

xniff_capture.c: Aquí se agrupan las funciones relacionadas con la captura de paquetes. En el se pone la interfaz en modo promiscuo, se capturan los paquetes, se aplican los filtros, etc.

xniff_init.c: En este archivo se ha incluido la inicialización de variables de la aplicación.

xniff_interface.c: Archivo que crea la interfaz de la ventana principal.

xniff_interface_clist.c: Crea una lista de columnas donde se mostrarán al usuario los paquetes que se van capturando.

xniff_interface_controls.c: Crea un panel de control que permite al usuario interactuar con el volcado del paquete.

xniff_interface_filter.c: Crea la ventana de configuración de los filtros.

xniff_interface_menu.c: Crea los menús de la aplicación.

xniff_interface_show_app_protocol.c: Crea una ventana con el contenido del protocolo de nivel de aplicación.

xniff_interface_showmessage.c: Crea un mensaje. Utilizado para mostrar mensajes al usuario.

xniff_interface_text.c: Crea el cuadro de texto dedicado al volcado del paquete.

xniff_interface_toolbar.c: Crea la barra de herramientas.

2.2.3 Arquitectura

El siguiente diagrama muestra el funcionamiento de Xniff en un diagrama de bloques:

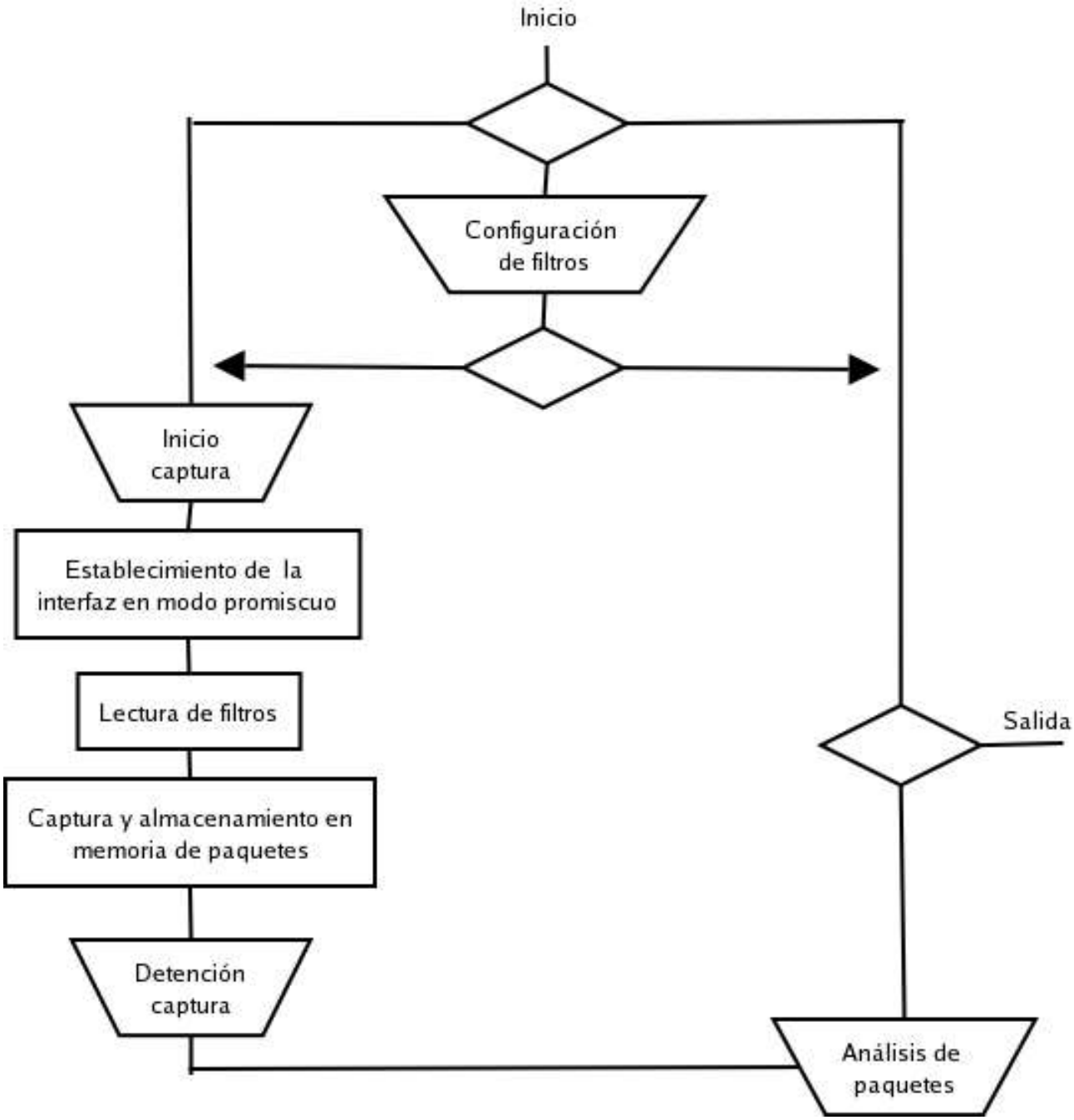


Figura 2.2.3: Diagrama de bloques

3. Implementación

3.1 Fases de implementación

3.1.1 Introducción

A continuación se describen las fases en las que se ha dividido la implementación del proyecto. Se ha dividido el proceso de implementación en tres fases. La primera está relacionada con la interfaz gráfica. La segunda con el módulo de captura de paquetes y la tercera con el módulo de filtrado.

Dado que Xniff es un sniffer totalmente gráfico, cabe destacar, que todas las fases contienen un importante desarrollo de interfaz.

El código completo de Xniff contiene unas 4500 lineas aproximadamente, por lo que no se incluirá en éste documento. Aún así, para hacer más esclarecedor el desarrollo seguido se han incluido algunos pedazos de código considerado relevante.

3.1.2 Fase1: Interfaz gráfica I

Existe una herramienta conocida como GLADE que a partir de un entorno visual en el que se diseña la interfaz a base de clics de ratón genera el código fuente automáticamente. Este aplicativo resulta especialmente útil en proyectos con una interfaz gráfica sencilla.

El problema con el que nos encontramos es que solo permite hacer un número limitado de cosas. Cuando se desea desarrollar mediante técnicas más avanzadas no queda más remedio que entrar a modificar el código fuente que GLADE ha generado. Ésto resulta problemático, dado que el código generado es desordenado y poco intuitivo.

Se ha decidido prescindir de GLADE por éste último motivo. Xniff dispone de una interfaz gráfica compleja aunque no muy grande, por lo que resulta más sencillo desarrollar todo el código a mano.

Ventana principal:

Inicialmente se ha desarrollado una ventana principal dividida en tres paneles, una barra de herramientas y un menú de aplicación.

La barra de herramientas y el menú permitirán el acceso a las diferentes funcionalidades del aplicativo.

De los tres paneles, un panel principal contiene una lista de columnas que mostrará los paquetes capturados. Los dos paneles restantes contendrán, el primero un volcado hexadecimal/ascii del paquete y el segundo un panel de controls que permitirá al usuario interactuar con el volcado.

El desarrollo de esta parte ha sido trivial, aunque un poco laborioso. A continuación una muestra del resultado:

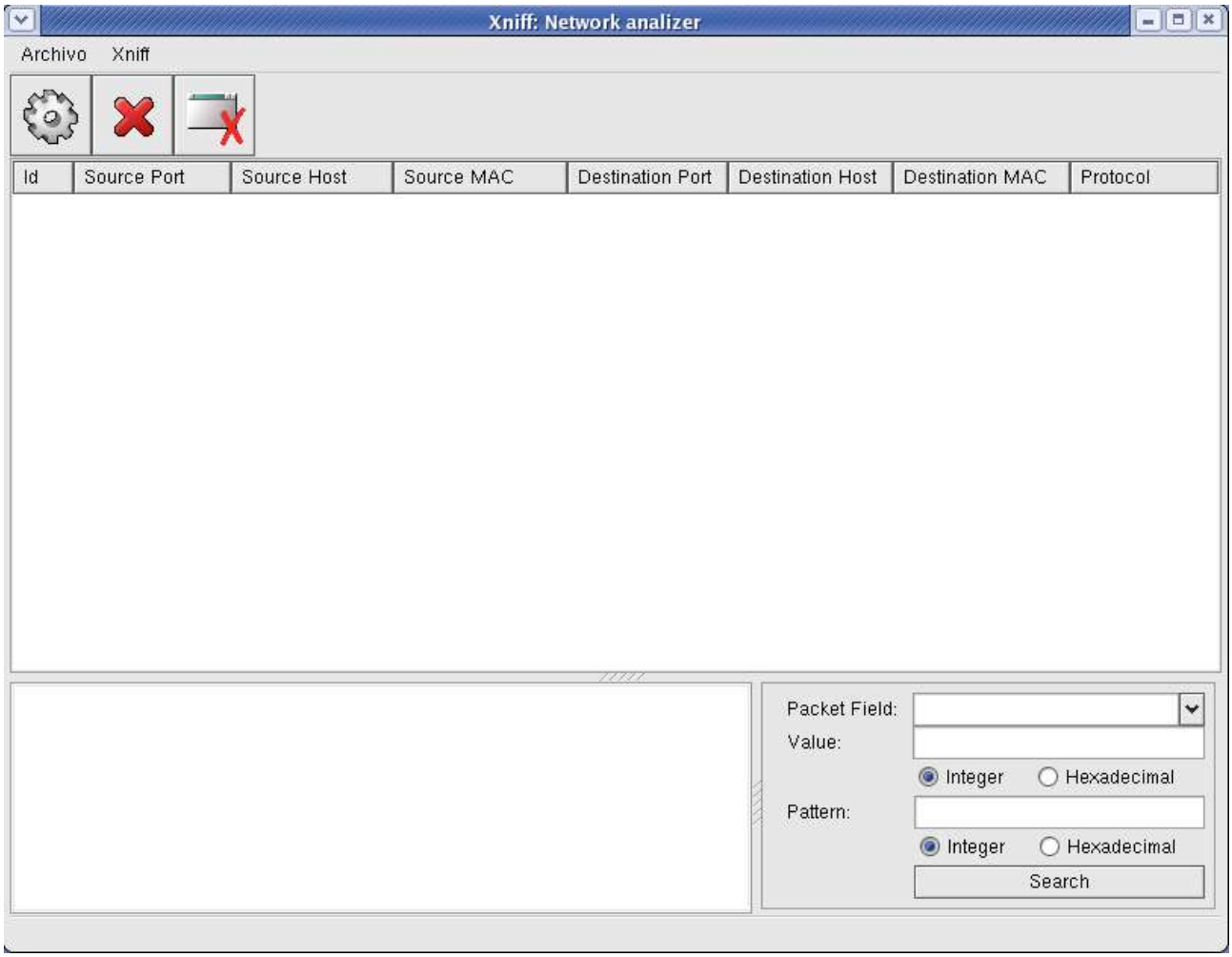


Figura 3.1.2: Ventana principal de Xniff

3.1.3 Fase2: Módulo de captura de paquetes

El módulo de captura de paquetes puede dividirse en cuatro partes: La primera consiste en el establecimiento de la interfaz de red en modo promiscuo. La segunda consiste en la captura de paquetes. La tercera en el acceso al contenido del paquete. Y la cuarta en el enlace entre la captura de paquetes y la parte gráfica de la aplicación.

Modo promiscuo:

Para poder capturar todos los paquetes de la red es necesario poner la interfaz en modo promiscuo. Para esto es necesario acceder al sistema con `ioctl()`.

```
int sd;
struct ifreq ifr;
char *interface = “eth0”;

/* Crea el socket */
if ((sd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
{
   perror("socket");
   return FALSE;
}
/* Interfaz elegida */
strncpy(ifr.ifr_name, interface, IFNAMSIZ);
if (ioctl(sd, SIOCGIFFLAGS, &ifr) == -1)
{
   xniff_show_message ("Alert!", "No such device or inteface");
   return FALSE;
}
/* Pone la interfaz en modo promiscuo */
ifr.ifr_flags |= IFF_PROMISC;
if (ioctl(sd, SIOCSIFFLAGS, &ifr) == -1)
{
   xniff_show_message ("Alert!", "Promisc mode error");
   return FALSE;
}
/* Cierra el socket */
close(sd);
```

El nombre de la interfaz deberá estar parametrizado para que el usuario pueda modificarlo.

Captura de paquetes:

Para capturar paquetes es necesario abrir un socket con las opciones necesarios para un acceso a todos los protocolos. A continuación hay que entrar en un bucle que se dedique a leer los paquetes que van pasando por la red. En Xniff esto se ha hecho de la siguiente manera:

```
/* Socket de captura de todos los paquetes. (modo promiscuo) */
if((sock = socket(AF_INET, SOCK_PACKET, htons(0x3))) == -1) {

   perror("socket()");
   exit(EXIT_FAILURE);
}
while ((xniff_semaphore)&&(cont<MAX_PACKETS)) {
   len = read(sock, xniff_packet, XNIFF_PACKET_SIZE);
   /* ... */
}
```

Donde `xniff_semaphore` es una variable que controla si Xniff está en modo captura o está parado.

Acceso al contenido del paquete:

Para acceder al contenido del paquete es necesario utilizar las estructuras de cabeceras disponibles en UNIX o crear estructuras propias. En Xniff utilizan las cabeceras de UNIX Ethernet, ICMP, IP, TCP y UDP. Pero la cabecera ARP ha sido reescrita y se ha incluido en el archivo “xniff_headers.h”.

```
/* Cabecera ARP */
struct arp_hdr {
   unsigned short int hardware;
   unsigned short int protocol;
   char hw_addr_len;
   char proto_addr_len;
   unsigned short operation;
   char src_addr[6];
   char src_ip[4];
   char dst_addr[6];
   char dst_ip[4];
};
```

Para acceder al contenido de los paquetes es preciso mapear los buffers con la estructura ya mencionada. En Xniff éste código se encuentra en el archivo "xn iff_capture.c" y es como sigue:

```
/* Cabecera ETH */
struct ether_header *eth = (struct ether_header *) xniff_packet;

/* Cabecera ARP */
struct arp_hdr *arp =
(struct arp_hdr *)(xniff_packet + sizeof(struct ether_header));


/* Cabecera IP */
struct iphdr *ip =
(struct iphdr*) (xniff_packet + sizeof(struct ether_header));

/* Cabecera ICMP */
struct icmphdr *icmp =
(struct icmphdr*) (xniff_packet + sizeof(struct ether_header));

/* Cabecera TCP */
struct tcphdr *tcp = (struct tcphdr *)
   (xniff_packet + sizeof(struct ether_header) + sizeof(struct iphdr));

/* Cabecera UDP */
struct udphdr *udp = (struct udphdr *)
   (xniff_packet + sizeof(struct ether_header) + sizeof(struct iphdr));
```

Enlace con la interfaz gráfica:

En el archi "xn iff.h" se define lo siguiente:

```
#define MAX_PACKETS 10000
#define XNIFF_PACKET_SIZE  1024
char xniff_packets_list[MAX_PACKETS][XNIFF_PACKET_SIZE];
```

Estas variables son globales y forman el enlace entre el módulo de captura y el entorno gráfico.

Cuando se realiza la captura de paquetes, éstos se guardan en `xniff_packets_list` desde donde los diferentes módulos gráficos pueden acceder cuando sea necesario.

3.1.4 Fase3: Interfaz gráfica II

Panel principal:

El objetivo del panel principal es mostrar al usuario los paquetes que se van capturando. Debe refrescarse en tiempo real, es decir a medida que se van capturando los paquetes debe mostrarlos por pantalla.

Ésto ha causado ciertos problemas inicialmente, dado que al entrar en un bucle de captura de paquetes, la gran cantidad de tráfico de red saturaba el aplicativo, que consecuentemente no respondía suavemente a los eventos.

La solución ha sido añadir al bucle la instrucción siguiente:

```
gtk_main_iteration ();
```

gtk_main_interation() procesa los eventos recibidos. Así para cada iteración del bucle el procesado de eventos evita que la aplicación se bloquee.

Por otra parte, el panel principal dispone de la posibilidad de ser ordenado por columnas mediante un simple clic de ratón. Ésto ha sido posible gracias a la función `gtk_clist_set_sort_column()`. Veamos la función de ordenación del archivo xniff_callbacks.c:

```
void xniff_on_clist_column_click(GtkCList *clist, gint column) {

   /* Por defecto ordenado de forma ascendiente */
   static gboolean sort_ascending = TRUE;

   /* Ordena el clist */
   if (sort_ascending) {
      gtk_clist_set_sort_type(GTK_CLIST(main_clist), GTK_SORT_ASCENDING);
      sort_ascending = FALSE;
   }
   else {
      gtk_clist_set_sort_type(GTK_CLIST(main_clist), GTK_SORT_DESCENDING);
      sort_ascending = TRUE;
   }

   gtk_clist_set_sort_column (GTK_CLIST (clist), column);

   if ( (column==0)||(column==1)||(column==2)||(column==4) )
      gtk_clist_set_compare_func (GTK_CLIST (clist), arithmetic_compare);
   else
      gtk_clist_set_compare_func (GTK_CLIST (clist), NULL);

   gtk_clist_sort (GTK_CLIST (clist));
   gtk_clist_set_auto_sort(GTK_CLIST(main_clist), TRUE);
}
```

Ventana de volcado del paquete:

Cuando se selecciona un paquete capturado del panel principal se muestra su contenido en hexadecimal y ASCII en la ventana de volcado.

El primer problema al desarrollar esta parte ha sido la fuente por defecto del sistema. Ésta no resultaba uniforme por lo que era imposible que el texto quedase alineado. Por lo tanto en este panel ha sido necesario cambiar la fuente. Una fuente adecuada es la courier.

```
GdkFont *f = gdk_font_load("-adobe-courier-medium-r-*-*-*-120-*-*-m-*-*-*");
```

Otro problema destacable consiste en la disposición del código hexadecimal y ASCII a partir de un buffer con el paquete capturado. El algoritmo utilizado ha sido el siguiente:

```
for (i=0; i<packet_len; i=i+2)
{
   /* Excribe líneas de 16 bytes */
   if (i%16==0) {
   gtk_text_insert (GTK_TEXT(l_text), f, NULL, NULL, "    ", 4);
   gtk_text_insert (GTK_TEXT(l_text), f, NULL, NULL, ascii, strlen(ascii));
   gtk_text_insert (GTK_TEXT(l_text), f, NULL, NULL, "\n    ", 5);
   memset(ascii,'\0',17);
   }

   /* Códigos Hexadecimales */
   char str[8];
   memset (str, '\0', 8);
   snprintf (str, 8, "%.2X%.2X ",
      (unsigned char)xniff_packets_list[id][i],
      (unsigned char)xniff_packets_list[id][i+1] );

   gtk_text_insert (GTK_TEXT(l_text), f, NULL, NULL, str, strlen(str));

   /* Códigos ASCII  */
   char a = '.';
   char b = '.';
   if (isprint( (unsigned char)xniff_packets_list[id][i] ))
      a = (unsigned char)xniff_packets_list[id][i];

   if (isprint( (unsigned char)xniff_packets_list[id][i+1] ))
      b = (unsigned char)xniff_packets_list[id][i+1];

   snprintf (str, 8, "%c%c", a, b);
   strncat (ascii, str, 8);
```

```
   }
```

Cabe destacar que las operaciones disponibles sobre un Widget GtkText son muy limitadas lo que obliga a realizar algunas "chapuzas" para poder disponer el código de forma diferente a lo habitual.

Panel de control:

El panel de control es la parte que permite al usuario interactuar con el contenido del paquete capturado.

Inicialmente se ha desarrollado el sistema de búsqueda de campos en el paquete. Su funcionamiento es sencillo, cuando el usuario selecciona un paquete del panel principal, el panel de control actualiza la lista de campos de la lista en función del paquete seleccionado. Veamos como ejemplo como se añaden los campos de un paquete TCP a la lista:

```
   if (ip->protocol == 6)
   {
      combo_items = g_list_append (combo_items, XN_TCP_SRC_PORT);
      combo_items = g_list_append (combo_items, XN_TCP_DST_PORT);
      combo_items = g_list_append (combo_items, XN_TCP_SEQ);
      combo_items = g_list_append (combo_items, XN_TCP_ACK);
      combo_items = g_list_append (combo_items, XN_TCP_D_R_FLAG);
      combo_items = g_list_append (combo_items, XN_TCP_WINDOW);
      combo_items = g_list_append (combo_items, XN_TCP_CHECKSUM);
      combo_items = g_list_append (combo_items, XN_TCP_URG);
   }

   /* ... */
   gtk_combo_set_popdown_strings (GTK_COMBO(combo_fields), combo_items);
```

El panel de control también permite pasar valores de entero a hexadecimal y viceversa. La función utilizada és:

```
void xniff_on_radio_value_clicked (GtkWidget *widget, gpointer data)
{
   /* Estado de los botones */
   gboolean value_int   = GTK_TOGGLE_BUTTON (radio_value_int)->active;
   gboolean value_hex   = GTK_TOGGLE_BUTTON (radio_value_hex)->active;

   /* Contenido del entry */
   char *text = gtk_entry_get_text (GTK_ENTRY(entry_value));
```

```
   char buffer[256];
   memset (buffer, '\0', 256);

   /* Pasa a entero */
   if (value_int) {

      if (xniff_radio_value_status == XN_RADIO_STATUS_INT)
         return;

      if (xniff_radio_value_status == XN_RADIO_STATUS_HEX) {

         /* Pasamos de HEX a INT */
         snprintf (buffer, sizeof(buffer), "%d", hextoi(text));
         gtk_entry_set_text (GTK_ENTRY(entry_value), buffer);
      }
   }
   /* Pasa a hexadecimal */
   if (value_hex){

      if (xniff_radio_value_status == XN_RADIO_STATUS_INT) {

         /* Pasamos de INT a HEX */
         snprintf (buffer, sizeof(buffer), "%X", atoi(text));
         gtk_entry_set_text (GTK_ENTRY(entry_value), buffer);
      }
      if (xniff_radio_value_status == XN_RADIO_STATUS_HEX)
         return;
   }
   /* Guardamos el nuevo estado */
   if (value_int) xniff_radio_value_status = XN_RADIO_STATUS_INT;
   if (value_hex) xniff_radio_value_status = XN_RADIO_STATUS_HEX;
}
```

Finalmente, existe la opción búsqueda que localiza ciertos valores entrados por el usuario en la ventan de volcado del paquete. Al encontrar el valor este queda seleccionado. Veamos una imagen de ejemplo:

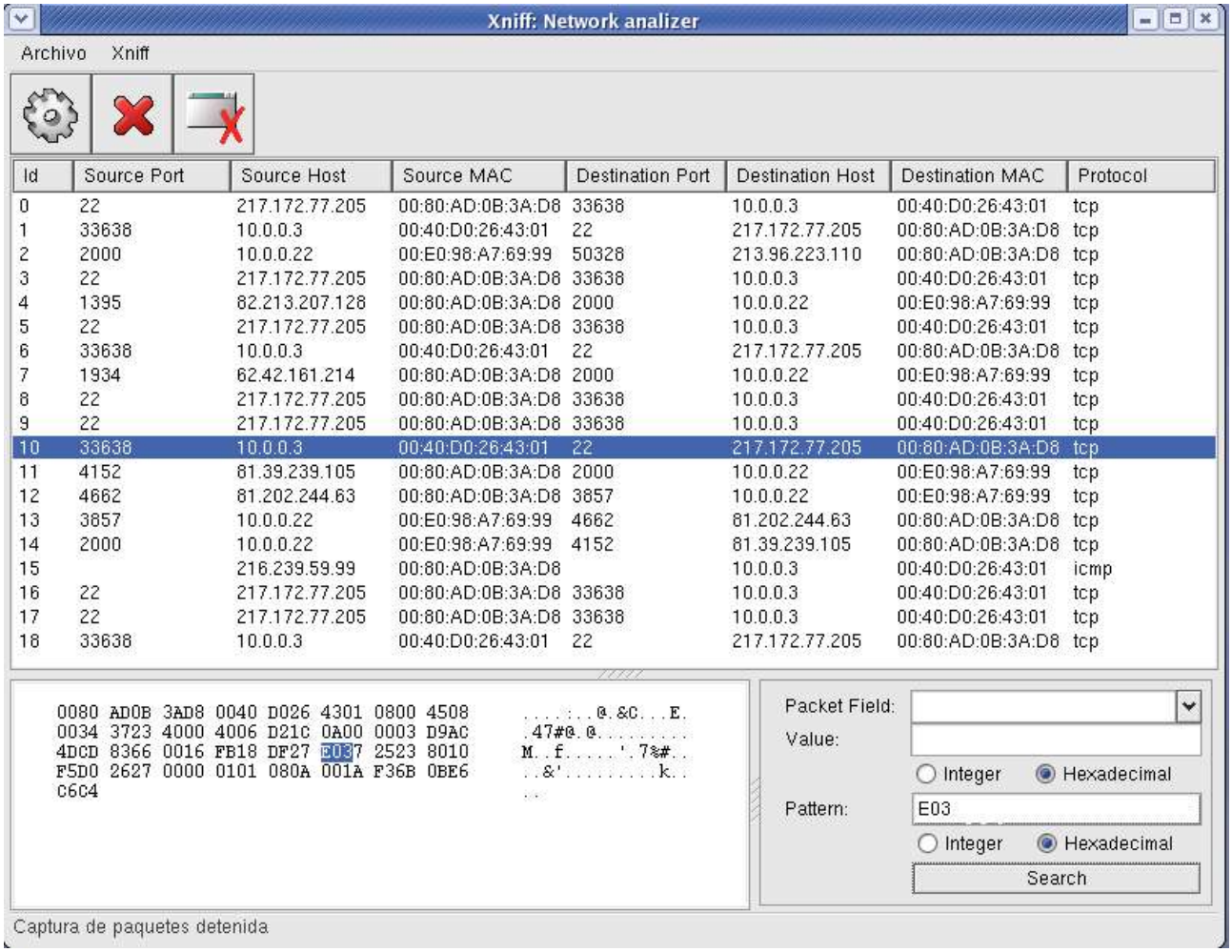


Figura 3.1.4: Búsqueda de una cadena hexadecimal

3.1.5 Fase4: Módulo de filtrado de paquetes

En el archivo xniff_interface_filter.c se crea una ventan de configuración de filtros. Cuando el usuario modifica esta configuración se establecen toda una serie variables que identifican cada campo. Posteriormente, cuando se realiza la captura de paquetes, el módulo encargado, lee estas variables y aplica los filtros. Ésto ocurre en el archivo "x niff_capture.c".

Como ejemplo veamos como se filtra la dirección IP:

```
if (strcmp(packet_filter->ip_src,"")!=0)
        if (strcmp(packet_filter->ip_src,
            inet_ntoa(*(struct in_addr *)&ip->saddr))!=0) continue;
```

3.2 Evaluación del sniffer

La única forma de testear la aplicación ha sido trabajando con ella y probando todo tipo de protocolos. A parte de lo laborioso que haya podido resultar el hecho de tener que ir capturando paquetes y comparar el contenido con otros sniffers ampliamente probados, el test general del sniffer ha resultado más o menos trivial. Se han probado los protocolos que eran objetivo: Ethernet, ARP, ICMP, IP, TCP y UDP. Y han probado otros protocolos para verificar que no causaban problemas.

Ha sido necesario realizar algunas modificaciones menores para acabar de ajustar correctamente Xniff y actualmente parece que funciona sin problemas.

3. Conclusiones

Después de dedicar la mayor parte de mi tiempo libre al desarrollo de esto proyecto he llegado a la conclusión de que aunque se han hecho muchas cosas, todavía quedan muchas más por hacer. Uno de los principales huecos en Xniff consiste en la detección de protocolos de nivel de aplicación. Es increíble ver como sniffers como Ethereal son capaces de detectar casi todos los protocolos de la red. Aún y así, creo que Xniff ha conseguido un nivel muy aceptable considerando el número de créditos de la asignatura.

Después de probar varios sniffers en Linux he podido observar que la mayoría se basan en lo mismo: Una interfaz gráfica (o no) basada en la librería PCAP. Es probable que ésta sea la mejor opción, pero no habría aprendido tanto sobre el análisis del tráfico de red y las APIs de UNIX si la hubiese utilizado en el desarrollo de Xniff.

A parte del desarrollo de los módulos de red me parece preciso recalcar la gran calidad de las librerías GTK+. Un aspecto que me ha llamado gratamente la atención ha sido la posibilidad de ejecutar Xniff en máquinas remotas con una interfaz local; posible gracias a GTK+ y GNOME.

4. Agradecimientos

- A Richard Stallman por iniciar el movimiento que nos ha hecho disfrutar de tan maravilloso software.

- A Linus Benedict Torvalds y a todos los hackers que han contribuido en el desarrollo de Linux.

- A Sun Microsystems por liberar StarOffice y a todos los desarrolladores de OpenOffice que han hecho posible trabajar con normalidad en un Desktop Linux. Ésta memoria no podría haber sido escrita con tanta facilidad en Linux sin OpenOffice.

- A todos los hackers que contribuyen al movimiento Open Source.

Apéndice A: Análisis de cabeceras

A.1. Cabeceras Ethernet

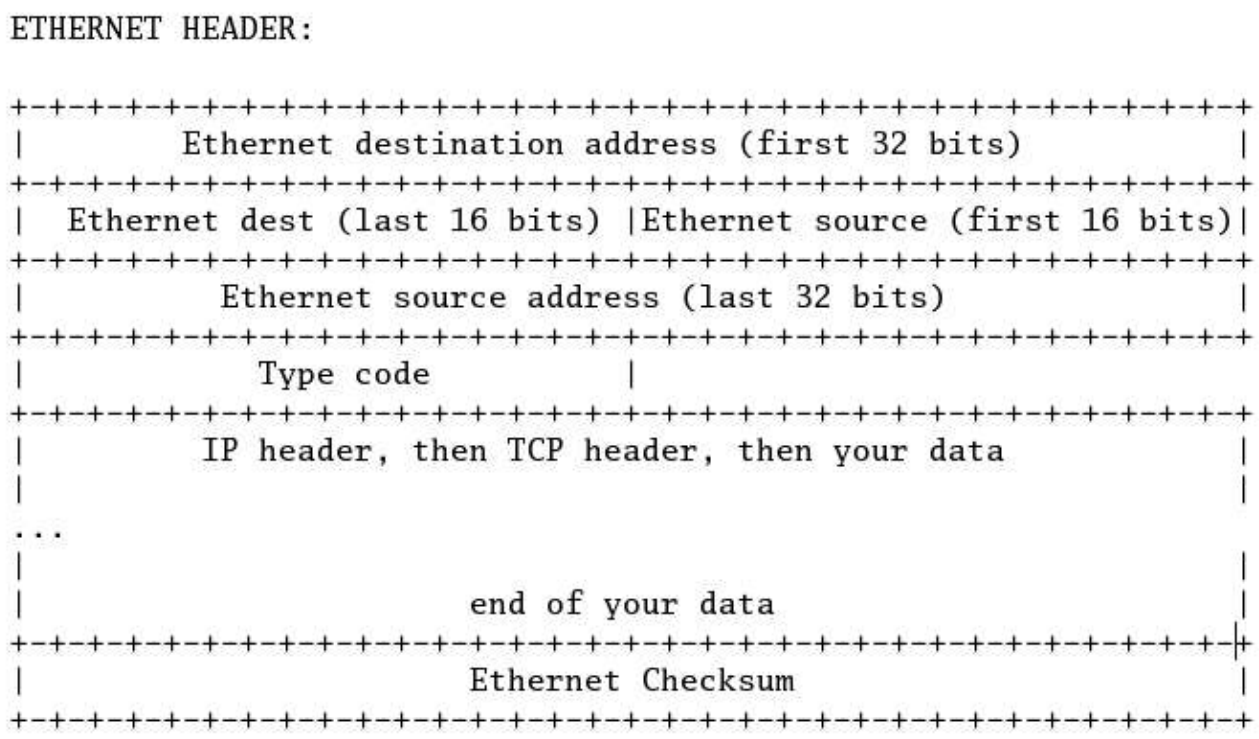


Figura A.1: Cabecera ethernet

Al realizar la captura de paquetes Xniff muestra en la ventana principal las direcciones ethernet origen y destino. Para analizar con más detalle los campos ethernet es necesario acceder al panel de control.

Mediante el panel de control Xniff permite identificar los siguientes campos:

- Dirección ethernet origen.
- Dirección ethernet destino.
- Tipo de paquete ethernet.

Además nos permite el acceso al código hexadecimal y ascii del paquete.

A.2. Cabeceras ARP

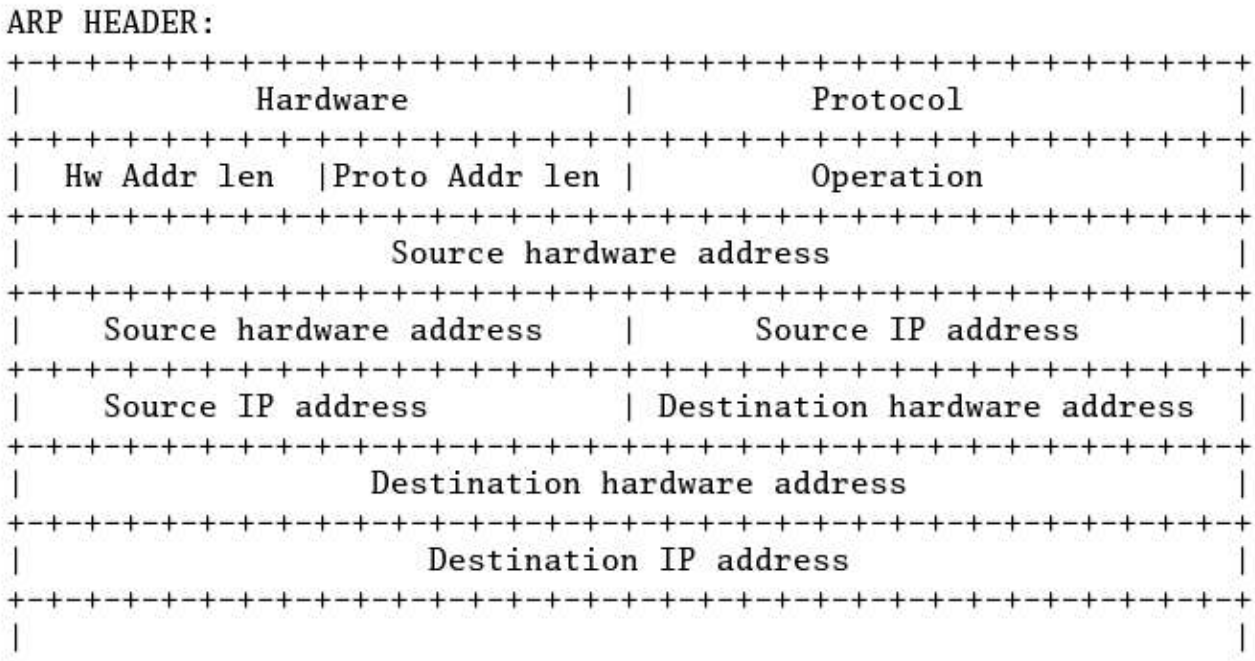


Figura A.2: Cabecera ARP

En la ventana principal del sniffer, los únicos campos que pertenecen al paquete ARP son la dirección IP origen y la dirección IP destino. Además el identificador de protocolo muestra que se trata de un paquete de este tipo. Los campos de dirección ethernet origen y ethernet destino deben coincidir con los campos "source hardware address" y "destination hardware address".

A través del panel de control podemos identificar los siguientes campos:

- hardware

- Protocol

- Operation

Además nos permite el acceso al código hexadecimal y ascii del paquete.

A.3. Cabeceras IP

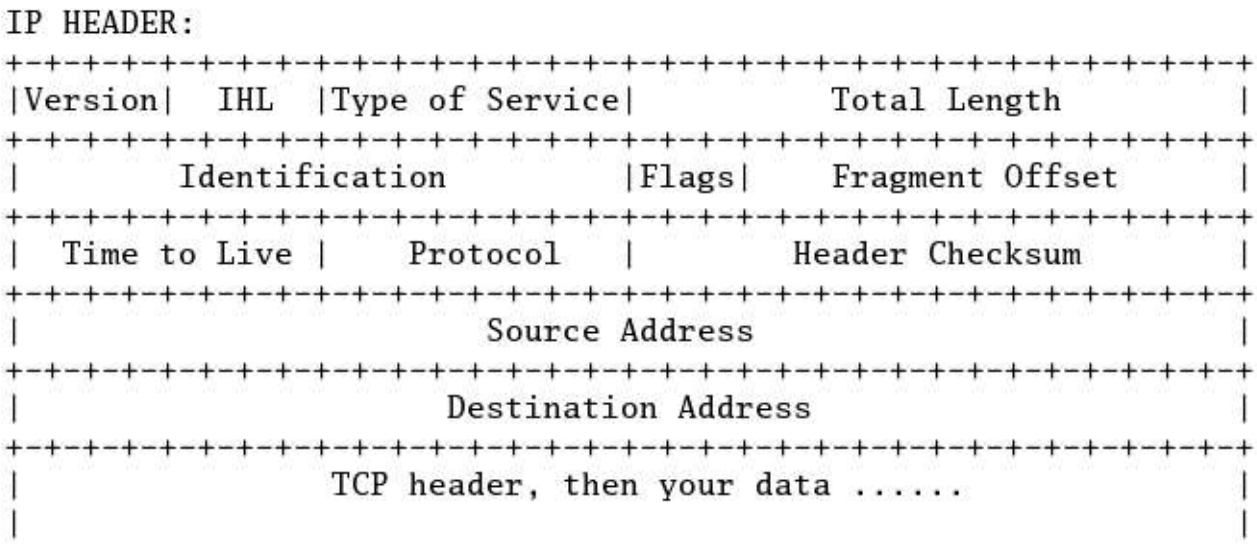


Figura A.3: Cabecera IP

En la ventana principal de Xniff podemos visualizar las IPs origen y destino. El resto de los campos y el contenido del paquete en hexadecimal y ascii es accesible desde el panel de control.

El siguiente listado de campos es identificado por Xniff:

- Versión.
- IHL
- ToS
- Total Lenght
- Identification
- Flags & Fragment offset
- TTL
- Protocol
- header checksum
- Source address
- Destination address

A.4. Cabeceras ICMP

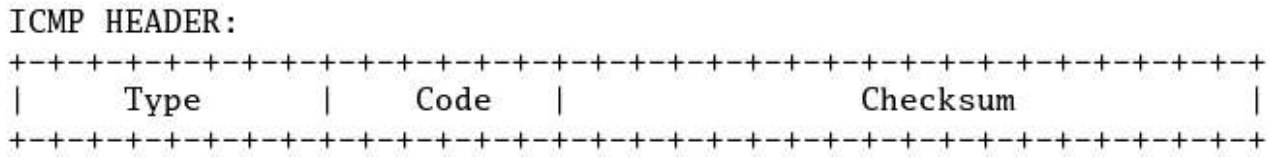


Figura A.4: Cabecera ICMP

En la ventana principal del sniffer no se muestra ningún campo de los paquetes ICMP. Únicamente el indicador de protocolo muestra que se trata de un paquete ICMP.

Los paquetes que reconoce Xniff desde el panel de control son:

- Type
- Code
- Checksum

A.5. Cabeceras TCP

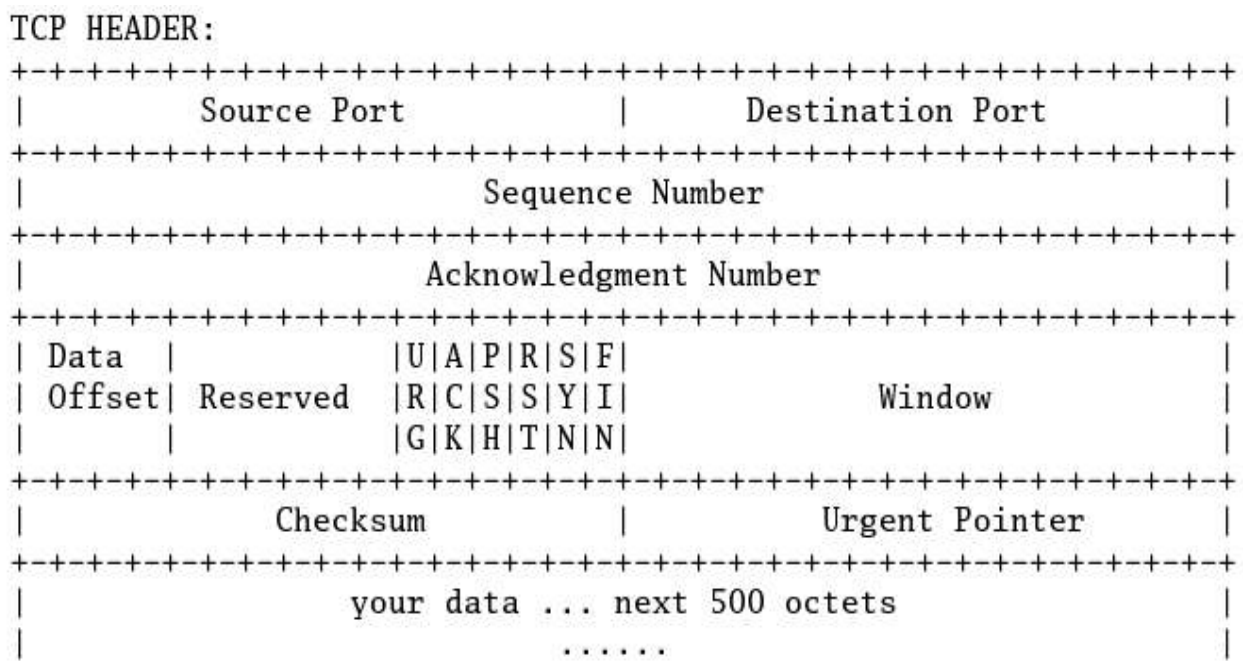


Figura A.5: Cabecera TCP

Desde la ventana principal del sniffer se pueden ver los puertos origen y destino. El resto de campos es accesible desde el panel de control.

La lista de campos reconocidos por Xniff es la siguiente:

- Source port
- Destination port
- Sequence number
- Acknowledgment number
- Flags
- Window
- Checksum
- Urg pointer

A.6. Cabeceras UDP

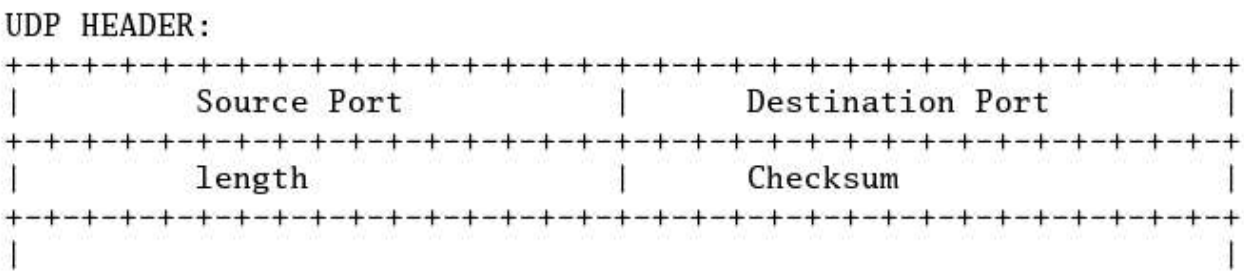


Figura A.6: Cabecera UDP

Desde la ventana principal del sniffer se pueden ver los puertos origen y destino. El resto de campos es accesible desde el panel de control.

La lista de campos reconocidos por Xniff es la siguiente:

- Source port
- Destination port
- Lenght
- Checksum

Apéndice B: Manual del usuario

B.1. Introducción

En los siguientes puntos se describen las partes en las que está dividida la aplicación y como funciona cada una de ellas.

El aspecto general de la aplicación es el siguiente:

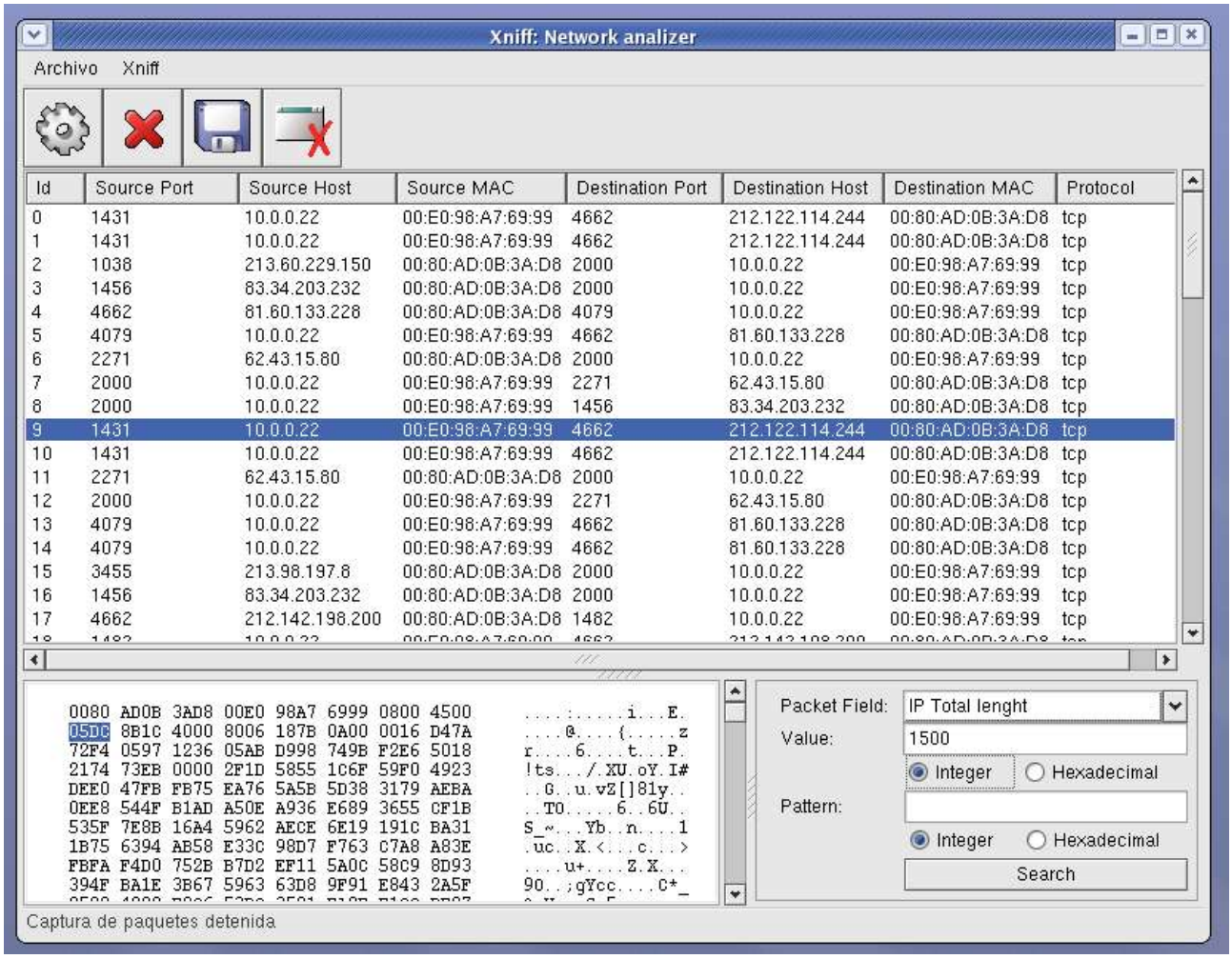


Figura B.1: Ventana principal de Xniff

En la imagen anterior podemos distinguir entre el menú principal, la barra de herramientas, un panel principal donde se muestra la captura de paquetes y dos subpaneles en la parte inferior. En el panel izquierdo se muestra un volcado del paquete mientras que en el panel derecho disponemos de un panel de control que permite interactuar con el volcado

hexadecimal/ascii.

B.2. Menú, teclas aceleradoras y barra de herramientas

Para acceder a las diferentes funcionalidades del aplicativo existen tres posibilidades:

- El menú.
- Las teclas aceleradoras.
- La barra de herramientas.

A continuación una captura muestra el aspecto del menú y la barra de herramientas:

Figura B.2: Barra de herramientas de Xniff

Entre los iconos disponibles en la barra de herramientas los más importantes son los dos primeros de la izquierda. El primero, la rueda dentada, inicia la captura de paquetes. El segundo, la cruz roja, finaliza las capturas.

Xniff dispone de cuatro teclas aceleradoras o atajos. Éstos se listan a continuación:

- Ctrl + i: Inicia la captura de paquetes.
- Ctrl + p: Par las captura de paquetes.
- Ctrl + f: Muestra una ventana desde donde se pueden configurar los filtros de paquetes.
- Ctrl + x: Cierra la aplicación.

B.3. Panel principal: Lista de paquetes capturados

En el panel principal se muestra la lista de paquetes capturados con el contenido de sus campos más importantes: host y puerto origen, host y puerto destino, dirección ethernet origen y destino, protocolo ...

Estos paquetes pueden ordenarse haciendo clic en la cabecera de cada columna. Por ejemplo, pulsando sobre "Destination Host" ordenaría los paquetes por la dirección del host destino.

Vea la siguiente figura.

Id	Source Port	Source Host	Source MAC	Destination Port	Destination Host	Destination MAC	Protocol
8	2000	10.0.0.22	00:E0:98:A7:69:99	1456	83.34.203.232	00:80:AD:0B:3A:D8	tcp
30	2000	10.0.0.22	00:E0:98:A7:69:99	1456	83.34.203.232	00:80:AD:0B:3A:D8	tcp
42	2000	10.0.0.22	00:E0:98:A7:69:99	1456	83.34.203.232	00:80:AD:0B:3A:D8	tcp
68	3971	10.0.0.22	00:E0:98:A7:69:99	25000	83.34.167.121	00:80:AD:0B:3A:D8	tcp
46	2000	10.0.0.22	00:E0:98:A7:69:99	4278	83.33.115.11	00:80:AD:0B:3A:D8	tcp
53	2000	10.0.0.22	00:E0:98:A7:69:99	4278	83.33.115.11	00:80:AD:0B:3A:D8	tcp
29	4921	10.0.0.22	00:E0:98:A7:69:99	4662	82.213.201.198	00:80:AD:0B:3A:D8	tcp
72	4921	10.0.0.22	00:E0:98:A7:69:99	4662	82.213.201.198	00:80:AD:0B:3A:D8	tcp
5	4079	10.0.0.22	00:E0:98:A7:69:99	4662	81.60.133.228	00:80:AD:0B:3A:D8	tcp
13	4079	10.0.0.22	00:E0:98:A7:69:99	4662	81.60.133.228	00:80:AD:0B:3A:D8	tcp
14	4079	10.0.0.22	00:E0:98:A7:69:99	4662	81.60.133.228	00:80:AD:0B:3A:D8	tcp
22	4079	10.0.0.22	00:E0:98:A7:69:99	4662	81.60.133.228	00:80:AD:0B:3A:D8	tcp
34	4079	10.0.0.22	00:E0:98:A7:69:99	4662	81.60.133.228	00:80:AD:0B:3A:D8	tcp
35	4079	10.0.0.22	00:E0:98:A7:69:99	4662	81.60.133.228	00:80:AD:0B:3A:D8	tcp
40	4079	10.0.0.22	00:E0:98:A7:69:99	4662	81.60.133.228	00:80:AD:0B:3A:D8	tcp
44	4079	10.0.0.22	00:E0:98:A7:69:99	4662	81.60.133.228	00:80:AD:0B:3A:D8	tcp
48	4079	10.0.0.22	00:E0:98:A7:69:99	4662	81.60.133.228	00:80:AD:0B:3A:D8	tcp
49	4079	10.0.0.22	00:E0:98:A7:69:99	4662	81.60.133.228	00:80:AD:0B:3A:D8	tcp
51	4079	10.0.0.22	00:E0:98:A7:69:99	4662	81.60.133.228	00:80:AD:0B:3A:D8	tcp

Figura B.3: Panel principal de Xniff

Cuando se inicia la captura el panel principal se va actualizando a medida que se reciben los paquetes.

B.4. Panel secundario: Análisis del paquete seleccionado

Cuando se selecciona un paquete en el panel principal, automáticamente se escribe en el panel secundario un volcado en código hexadecimal y ASCII.

Puede utilizarse el panel de control (siguiente apartado) para analizar dicho código hexadecimal.

Ejemplo de paquete ICMP:

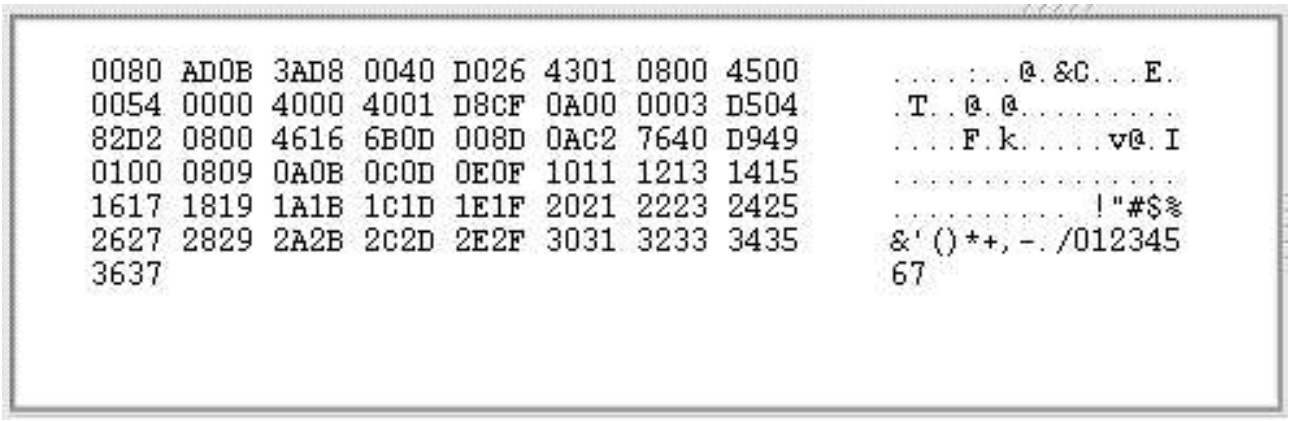


Figura B.4: Panel de volcado de paquetes

B.5. Panel de control

El panel de control permite realizar búsquedas en el volcado hexadecimal del paquete.

En el siguiente ejemplo buscamos el tipo de un paquete ICMP:

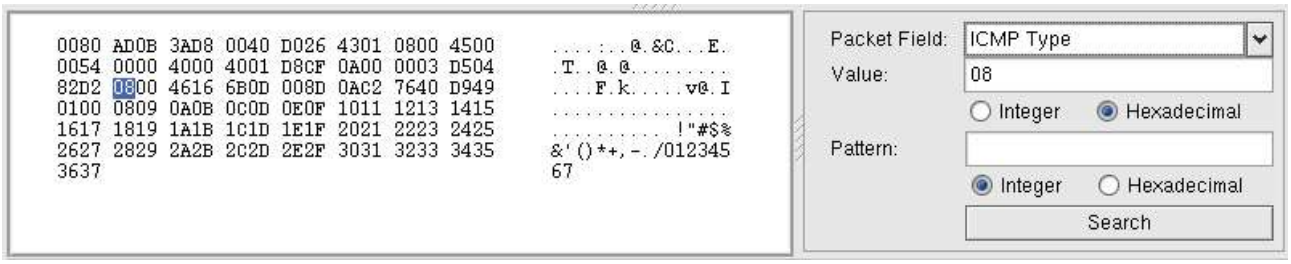


Figura B.5: Uso del panel de control

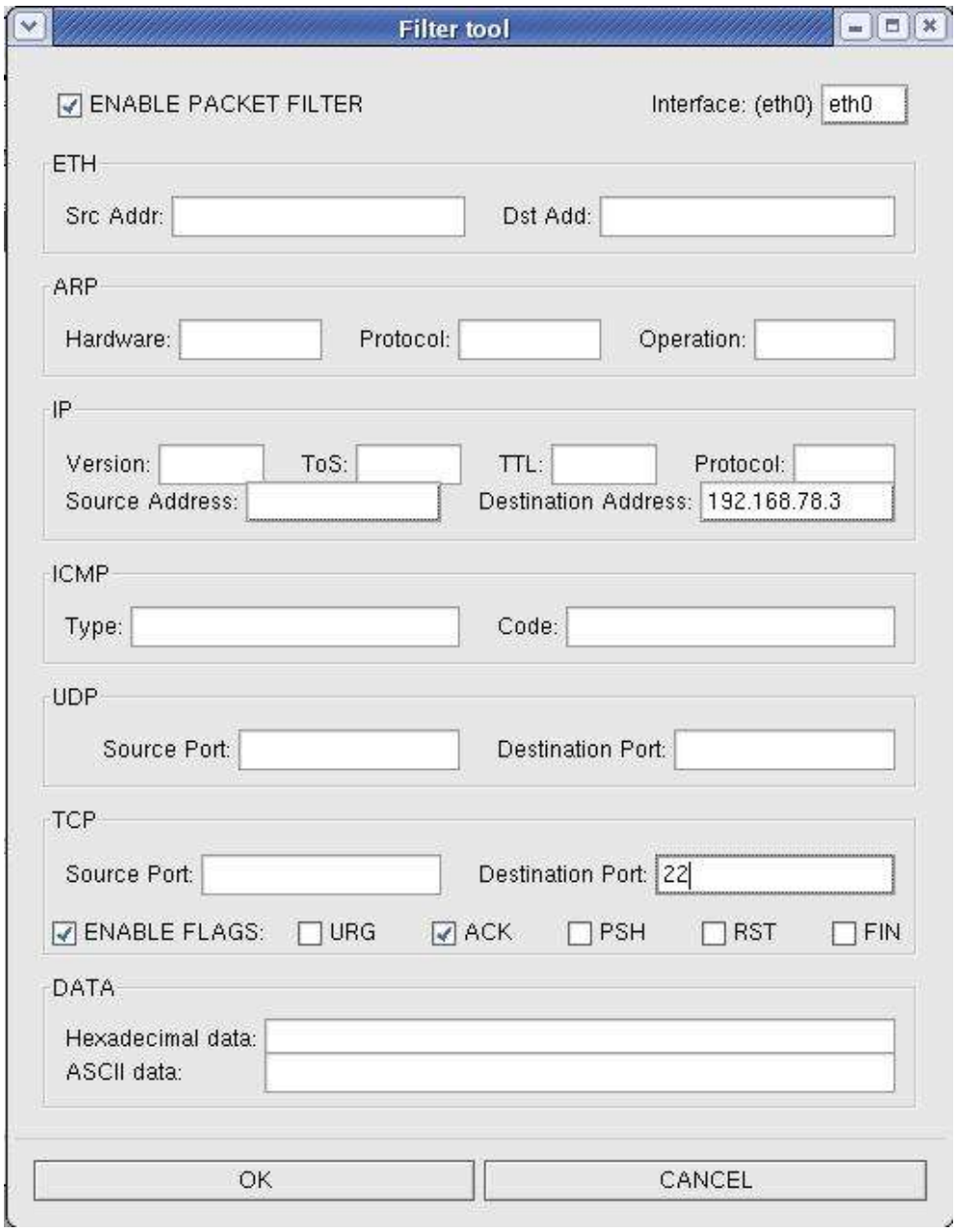
El resultado podemos pasarlo a entero utilizando los botones de radio de la parte inferior.

El panel de control también permite buscar patrones en el código hexadecimal mediante el botón "search" y pasarlos de hexadecimal a entero o viceversa.

B.6. Filtrado de paquetes

Desde el menú o mediante la combinación Ctrl+f se puede acceder al filtro de paquetes. Desde éste podemos especificar las condiciones que deben cumplir los paquetes capturados.

En el siguiente gráfica vemos un ejemplo del sistema de filtrado:



El sistema de filtrado dispone de diferentes opciones que se describen a continuación:

1. Habilitación del sistema de filtrado.
2. Interfaz de red utilizada.
4. Protocolo Ethernet.
5. Protocolo ARP.
6. Protocolo IP.
7. Protocolo ICMP.
8. Protocolo UDP.
9. Protocolo TCP.
10. Filtrado de datos.

Apéndice C: Manual de instalación

La instalación de Xniff es muy sencilla y se describe en los siguientes paso:

Xniff se distribuye en un paquete llamado "xniff.tar.gz". És te paquete está empaquetado con la herramienta tar y comprimido con la herramienta gzip. Puede descomprimirse y desempaquetarse con el siguiente comando:

$ tar -xvfz xniff.tar.fz

El comando anterior creará un directorio "x niff" que contiene a su vez dos directorios. Un directorio "doc/" con documentación varia y un directorio "src/" con el código de xniff.

Para compilar el código entre en el directorio "sr c/" y escriba "make":

$ cd xniff

$ make

Una vez compilado, puede instalar el ejecutable con:

$make install

Finalmente, puede ejecutar xniff con:

$xniff

Apéndice D: Licencia de Xniff

Se ha optado por la licencia GPL. Puede encontrar un copia en inglés y en castellano en los siguientes enlaces:

Licencia original: http://www.gnu.org/copyleft/gpl.html

Licencia en castellano: http://es.gnu.org/Licencias/gples.html

Bibliografía

Libros:

Programación de Socket Linux

Autor: Sean Walton

Editorial: Prentice Hall

ISBN: 84-205-3121-9

UNIX Programación avanzada

Autor: Fco. Manuel Márquez

Editorial: Ra-ma

ISBN: 84-7897-239-0

Desarrollo de aplicaciones Linux con GTK+ y GDK

Autor: Eric Harlow

Editorial: Prentice Hall

ISBN: 84-8322-196-9

C manual de referencia. 4Ed

Autor: Herbert Schildt

Editorial: Mc Graw-Hill

ISBN: 84-481-2895-8

Enlaces:

google: http://www.google.com

GNOME: http://www.gnome.org

GTK+: http://www.gtk.org

Ethereal: http://www.etehreal.com

TCPDump: http://www.tcpdump.org