UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

"DISEÑO Y DESARROLLO DE UN ANALIZADOR DE PROTOCOLOS DE RED DE DATOS"

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A:

HERIBERTO GARCÍA LEDEZMA

DIRECTORA DE TESIS: M.C. MA. JAQUELINA LÓPEZ BARRIENTOS

México, D.F. **Febrero del 2009.**

Agradecimientos

Gracias por esta vida…

A mi madre, Yolanda, y a mis hermanos, Yuridia y Abimael, gracias por su amor y comprensión, por creer en mí y por su apoyo en todos los sentidos. Ustedes son mi principal fuente de inspiración y de fuerza.

Gracias tía Juana por tu amor, gracias Simón por impulsar mi superación.

Gracias a todos mis amigos, ustedes que sin que tengamos la misma sangre han llegado a ser parte de mi vida y familia.

Gracias Maestra Jaquelina por guiarme y apoyarme en esta tesis, por todos sus consejos y por su amistad.

Gracias a la UNAM y a la Facultad de Ingeniería por darme esta formación académica y todas las demás experiencias que tuve en sus instalaciones.

Me lo contaron y lo olvidé; lo vi y lo entendí; lo hice y lo aprendí.
Confucio.

Todo lo que somos es el resultado de lo que hemos pensado;
está fundado en nuestros pensamientos y está hecho de nuestros pensamientos.
Siddhartha Gautama

Uno mismo hace el mal, uno mismo lo sufre; uno mismo se aparta del mal, uno mismo se purifica. Pureza e impureza son cosas de uno mismo, nadie puede purificar a otro.
Siddhartha Gautama

Índice

Índice I

Introducción 1

Capítulo 1. Antecedentes generales 7

1.1 Redes de computadoras actuales 7
1.1.1 Características generales de una red de computadoras 8
1.1.2 Conexión y transmisión de información 12
1.1.2.1 Tipos de comunicación 14
1.1.2.2 Métodos de transferencia y control de transmisión 14
1.1.3 Tecnologías de red 15
1.1.3.1 Ethernet 16
1.1.3.1.1 CSMA/CD 16
1.2 Arquitectura de red 18
1.3 Identificadores de red 23
1.3.1 Direcciones de red IPv4 24

Capítulo 2. Protocolos soportados por el analizador 29

2.1 Protocolos TCP/IP 29
2.1.1 Protocolo Internet, IP 31
2.1.1.1 Fragmentación 33
2.1.2 Protocolo de Control de Transmisión, TCP 34
2.1.3 Protocolo de Mensajes de Control de Internet, ICMP 39
2.1.3.1 Descripción de los tipos de mensajes ICMP 40
2.1.3.1.1 Mensaje de Destino Inaccesible 41
2.1.3.1.2 Mensaje de Tiempo Superado 41
2.1.3.1.3 Mensaje de problema de parámetros 41
2.1.3.1.4 Mensaje de Disminución del Tráfico desde el Origen 41
2.1.3.1.5 Mensaje de Redirección 42

2.1.3.1.6 Mensaje de Echo o de Echo Reply 42
2.1.3.1.7 Mensaje de Solicitud y de Respuesta de Marca de Tiempo 42
2.1.3.1.8 Mensaje de Solicitud de Información o de Respuesta de Información 43
2.1.4 Protocolo de Datagramas de Usuario, UDP 43
2.1.5 Protocolo de Resolución de Direcciones, ARP 44

2.2 Formatos del frame Ethernet 46
2.2.1 Ethernet II 47
2.2.2 Ethernet Novell RAW 802.3 47
2.2.3 Ethernet IEEE 802.3 47
2.2.4 Ethernet IEEE 802.3 SNAP 48

Capítulo 3. Análisis y diseño 51

3.1 Analizadores de protocolos 51
3.1.1 Características generales 52
3.2 Planteamiento del problema 52
3.2.1 Bitácoras de información 53
3.3 Diseño del analizador de protocolos. 55
3.3.1 Descripción básica de la aplicación del modelo de proceso 57
3.4. Descripción del entorno de desarrollo 60
3.4.1 Sistema operativo 60
3.4.2 Dependencia de archivos 61
3.4.3 Lenguajes de programación utilizados 62
3.4.3.1 Compilador de C 63
3.4.3.2 Compilador de java 65
3.4.4 Datos técnicos del hardware y sistema operativo de desarrollo 65

Capítulo 4. Desarrollo 67

4.1 Descripción general de la programación 67
4.2 Descripción de códigos fuente en lenguaje C 72
4.2.1 Archivo *Netbender.c* 73
4.2.1.1 Función main(int argc, char **argv) 73
4.2.1.2 Función leeFlujo(int ptcp, int pudp, int picmp, int parp, int pantalla, char modo, char *interfaz, char *archivoBytes) 74
4.2.1.3 Función int creaFormato(char *archivoOrigen, char *archivoDestino) 82
4.2.2 Archivo paquetesTCP.c 84
4.2.2.1 Función int paquetesTCP(unsigned char *buffEth, unsigned char *buff, char *archivoBytes, int tamEth, char *tiempo, int Tmillitm, int noFrame, int pantalla) 84
4.2.3 Archivo paquetesICMP.c 88
4.2.3.1 Función int paquetesICMP(unsigned char *buffEth, unsigned char *buff, char *archivoBytes, int tamEth, char *tiempo, int Tmillitm, int noFrame, int pantalla) 88
4.2.4 Archivo paquetesUDP.c 91
4.2.4.1 Función int paquetesUDP(unsigned char *buffEth, unsigned char *buff, char *archivoBytes, int tamEth, char *tiempo, int Tmillitm, int noFrame, int pantalla) 91
4.2.5 Archivo paquetesARP.c 92
4.2.5.1 Función int paquetesARP(unsigned char *buffEth, unsigned char *buff, char *archivoBytes, int tamEth, char *tiempo, int Tmillitm, int noFrame, int pantalla) 92

4.2.6 Archivos sumacontrolIP.c, sumacontrolTCP.c, sumacontrolICMP.c y Archivo sumacontrolUDP.c 94
4.2.6.1 Función unsigned short sumacontrolIP(void *datosTCP, int tam, void *datosIP) 95
4.2.6.2 Función unsigned short sumacontrolTCP(void *datosTCP, int tam, void *datosIP) 95
4.2.6.2 Función unsigned short sumacontrolICMP(void *datosICMP, int tam) 96
4.2.6.3 Función unsigned short sumacontrolUDP(void *datosUDP, int tam, void *datosIP) 97
4.2.7 Archivos xmlTCP.c, xmlICMP.c, xmlUDP.c y xmlARP.c 98
4.2.7.1 Función int xmlTCP(unsigned char *buffEth, int noFrame, char *aptfecha, char *apthora, int tamanioTotal, char *archivoDestino) 98
4.2.7.2 Función int xmlICMP(unsigned char *buffEth, int noFrame, char *aptfecha, char *apthora, int tamanioTotal, char *archivoDestino) 100
4.2.7.3 Función int xmlUDP(unsigned char *buffEth, int noFrame, char *aptfecha, char *apthora, int tamanioTotal, char *archivoDestino) 101
4.2.7.4 Función int xmlARP(unsigned char *buffEth, int noFrame, char *aptfecha, char *apthora, int tamanioTotal, char *archivoDestino) 103
4.2.8 Archivos textoTCP.c, textoICMP.c, textoUDP.c, textoARP.c 104
4.2.8.1 Funciones textoTCP(), textoICMP(), textoUDP(), textoARP() 104
4.2.9 Archivos de encabezado 105
4.3 Descripción del código fuente analizaFlujo.java 106
4.3.1 Detalles del constructor analizaFlujo 106
4.3.2 Detalles de los métodos 106
4.3.2.1 Método parseaFlujo 106
4.3.2.2 Método imprimeEstadisticas 107
4.3.2.3 Método main 107

Capítulo 5. Instalación y forma de uso 109

5.1 Instalación del analizador de protocolos 109
5.2 Uso del analizador de protocolos 110
5.3 Ejemplo de ejecución y salida 111

Capítulo 6. Mantenimiento y expectativas a futuro 119

6.1 Consideraciones para el uso del analizador de protocolos 119
6.1.1 Sobre el sistema operativo 119
6.1.2 Sobre la edición de códigos fuente y compilación 120
6.1.3 Sobre la información generada 120
6.2 Estado actual de la aplicación 121
6.3 Propuestas de puntos por mejorar 121

Conclusiones 123

Anexos 127

ANEXO I. Clasificación de redes por cobertura 127
ANEXO II. Topologías de red 129
ANEXO III. Subredes 133
ANEXO IV. Dispositivos de red 136
ANEXO V. Conceptos de desempeño de red 139
ANEXO VI. IP option numbers 143
ANEXO VII. TCP option numbers 144
ANEXO VIII. ICMP type numbers 145
ANEXO IX. Definición del Tipo de Documento para la bitácora XML 149
ANEXO X. Manpage para el programa Netbender 155
ANEXO XI. Manpage para el programa analizaFlujo 158
ANEXO XII. Manpage para frame.dtd 160

Glosario 161

Referencias 167

Introducción

Uno de los aspectos claves para la evolución de la humanidad en todos los ámbitos ha sido la comunicación, ya que, entre otros puntos, ha permitido conocer los avances logrados en distintos lugares y en distintos tiempos favoreciendo la integración de estos conocimientos y su mejor aprovechamiento.

El proceso de comunicación en su forma más básica requiere de por lo menos 3 componentes: un emisor de la información (fuente del mensaje), un medio para transmitirla (canal de comunicación) y un receptor (destinatario); y dependiendo de los mecanismos para llevar a cabo la transmisión de conocimientos o de cualquier idea en general intervienen nuevos elementos que tienen como fin mejorar el proceso comunicativo en diversos aspectos, desde la fidelidad de lo que expresa la fuente, pasando por la rapidez de su propagación, y contemplando puntos de confidencialidad, entre muchos otros que dependen del tipo de información.

A través del tiempo la humanidad ha encontrado algunos obstáculos que dificultan el proceso comunicativo entre diversos grupos, principalmente por la forma en como expresan sus ideas de forma gráfica y hablada. Para solucionar éste y otros conflictos, cada grupo se ha visto obligado a comprender distintos lenguajes, a usar traductores o a desarrollar equivalencias entre los distintos lenguajes. En el campo tecnológico se ha presentado algo similar por el uso de distintas técnicas de comunicación y en consecuencia se ha optado por generar normas que regulan tanto la forma física de comunicación, en lo que se refiere a los medios, como a la forma lógica, como por ejemplo los procedimientos para negociar conexiones entre dispositivos; es decir, se han desarrollado estándares para facilitar el flujo de los datos. Es claro que se facilita más la comunicación entre más elementos utilizan el

mismo conjunto de normas para comunicarse, ya que con esto se reduce en tiempo y elementos el proceso de comunicación al no verse en la necesidad de codificar en distintos formatos el mensaje emitido.

Por otra parte, no sólo varían las características de los elementos que posibilitan la comunicación en cuanto a alcances y procedimientos para cumplir sus objetivos, sino que también cambian en cuanto a cantidad. De esta forma se presentan canales de comunicación distintos y receptores que posteriormente se convierten en retransmisores (o fuentes intermediarias de información) a lo largo de todo el proceso comunicativo. Con este procedimiento de transmisión se genera una red de información, que se va tejiendo por los receptores y transmisores que intervienen para que una fuente inicial y original comunique algo a un destinatario en particular. Bajo este ambiente de necesidad de comunicación surgieron redes postales, redes telegráficas, telefónicas y de datos, entre muchas otras. De acuerdo con lo anterior una definición adecuada para red de computadoras es:

> "*Conjunto de elementos interconectados entre sí, mejor conocidos como nodos, con el propósito de intercambiar información y compartir recursos de software y hardware.*"

En este caso un nodo puede ser tanto una computadora como cualquier dispositivo activo (switches, routers, etc.) que intervenga en la comunicación de datos.

En los años posteriores a la segunda guerra mundial, los propósitos principales que se tenían para las redes de computadoras eran de tipo militar y por bastante tiempo continuó así. Alrededor del año 1973 el departamento de defensa de los Estados Unidos inició un programa tecnológico con el fin de lograr una comunicación efectiva entre redes de computadoras con diferentes características, del cual surgieron varios protocolos de comunicación que en conjunto se conocen como la suite de protocolos TCP/IP (Transmisión Control Protocol/Internet Protocol), y que dio inicio a lo que se conoce como Internet en la actualidad. Para la década de 1980 a 1990, con el inicio de las computadoras personales, la interconexión de redes tomó un enfoque distinto debido a que en los centros de trabajo donde se empezaron a utilizar estas microcomputadoras cada vez fue más necesario compartir información entre estos dispositivos para incrementar la eficiencia. Algo similar sucedió en los últimos 10 años del siglo XX, sin embargo los lugares donde se popularizó el uso de redes de computadoras ya no fueron sólo los centros de trabajo, sino que se incluyó a los hogares, principalmente con el uso de la Internet, que básicamente se puede considerar como una red de redes a nivel mundial y que basa su popularidad en la cantidad y variedad de servicios que ofrece: desde servicios de correo electrónico, servicios de hospedaje para sitios Web y servicios de compra y de conversaciones en tiempo real, por mencionar algunos.

Prácticamente en todo sector de la industria y negocios las labores se relacionan de una u otra forma con el manejo de información a través de una red de datos y lógicamente el cuidado de estos datos es lo más importante. Es común la presencia de redes de comunicación de datos en el sector de educación, en el de investigación, en el de comercio, en el de salud, etc., ya que con ellas se puede manejar eficientemente cantidades enormes de información tanto por su tamaño como por su importancia. Estas redes pueden ser desde pequeñas del tipo LAN (Local Area Network) en las que las personas que se encargan de

ellas, administradores de red, conocen con *suficiente* detalle a los elementos que intervienen en la transmisión de la información y tienen control sobre ellos, hasta redes extensas como las WAN's (Wide Area Network) donde la cantidad de elementos que sirven como intermediarios en el proceso de comunicación es bastante grande y no está al alcance de un solo administrador el poder verificar su correcto funcionamiento y que se le dé un adecuado uso.

En la administración de redes actualmente se tienen definidos tres puntos indispensables: la disponibilidad, la confidencialidad, y la integridad de la información y por lo tanto se deben establecer controles rígidos que garanticen que se cumplan. En la parte que se refiere a la transferencia de datos se debe cuidar especialmente la integridad y la confidencialidad y es por ello que día a día se presentan nuevos mecanismos que se encargan de mejorar las características del medio de comunicación para no perder información ni que ésta se vea alterada; y también se desarrollan metodologías y tecnologías que permiten un mayor grado de seguridad de la información que se transmite para que no sea robada o interferida y sólo puedan acceder a ella los individuos adecuados a través de técnicas de autenticación difíciles de violar.

Es sorprendente el avance tecnológico con el que se cuenta hoy en día y el índice de desarrollo que se tiene; sin embargo es muy frecuente el escuchar de fraudes cometidos con ayuda de la tecnología, de hechos ilícitos conseguidos en sitios de Internet y a través de él, y esto se debe precisamente a su carácter público y a que no se tiene total control sobre las redes por las que circula nuestra información, pues cualquier persona prácticamente puede acceder a Internet desde cualquier lugar con que se cuente con un proveedor de este servicio. Una solución a este problema sería el conocer cómo son administradas las redes intermediarias en el proceso de transmisión de información y asegurarse que siguieran normas de seguridad bien definidas y que nadie las quebrantara; sin embargo, esta situación es imposible, así que sólo queda a los administradores de red tomar medidas en cuanto al horizonte sobre el que tienen dominio. Para lo anterior se debe considerar que los posibles ataques pueden venir desde redes externas o incluso de la misma red interna (intranet).

Existe una gran variedad de razones por las que se realiza un ataque a una red que compromete alguno o los tres puntos esenciales ya mencionados; desde ataques que se realizan por el simple hecho de saber que se puede romper la seguridad informática de alguna organización, pasando por la detección de debilidades para eliminarlas, y hasta las que están claramente impulsadas por el robo de información valiosa para una institución. De igual forma existen múltiples métodos para realizar intrusiones a un sistema, desde los muy elaborados y que llevan mucho tiempo hasta las que se realizan por usuarios que sólo ocupan herramientas diseñadas por terceras personas sin conocer el funcionamiento del programa que están ejecutando.

Contrariamente a lo que se pudiera pensar, el porcentaje de ataques externos se encuentra alrededor del 30% del total, y en consecuencia el 70% restante se debe a ataques internos a la red por usuarios malintencionados o simplemente descuidados según el *Verizon's 2008 Data Breach Investigations Report*. En general los ataques externos y/o internos pueden llegar a causar pérdidas millonarias. Ante esto se han establecido mecanismos de prevención e identificación de anomalías en el tráfico de la red. Entre los métodos más

comunes se encuentra el uso de *firewalls* o *cortafuegos*, que mediante el establecimiento de ciertas reglas seleccionan el tráfico de red que es permitido, denegado o rechazado. Otra de las aplicaciones a las que se recurre frecuentemente es la de los IDS (siglas en inglés de Sistema Detector de Intrusos). Básicamente los IDS son programas que detectan accesos no autorizados (o su intento), de acuerdo a políticas de seguridad de red que se establecen, a un sistema mediante el análisis del tráfico de red y la comparación de los datos obtenidos con firmas de ataques conocidos, y comportamientos sospechosos como pueden ser peticiones de conexión repetidas, escaneo de puertos o paquetes de información malformados.

Desde otro punto de vista, con el paso del tiempo también se han diseñado aplicaciones con el propósito de evaluar la eficiencia de una red en aspectos de seguridad y desempeño. Entre los muy variados programas se encuentran los que se clasifican como *analizadores de protocolos*; utilidad que se puede utilizar desde distintos enfoques: ya sea para conseguir información manejada por aplicaciones que se ejecutan en los nodos de red y utilizarla para posteriormente violar normas de seguridad de la red aprovechando algún *bug* del software de red, del sistema operativo o de su configuración; o por el contrario, para detectar las vulnerabilidades de la red y de su configuración y corregirlas y saber en qué grado se es potencial víctima de una intrusión o un ataque. Con los *analizadores de protocolos* se obtiene información del tipo de protocolos utilizados en la red, del origen y destino de los datos en cuanto a puertos y direcciones IP, y otros datos útiles que permiten conocer la situación de una red desde diversos aspectos, como por ejemplo el tipo de información que circula en ella y las aplicaciones que se ejecutan en los nodos que la integran.

Para asentar los fines del presente trabajo se definen los objetivos general y particulares de la siguiente forma:

Objetivo general.

Generar una aplicación que reúna las características esenciales de un analizador de protocolos.

Objetivos particulares.

1. Diseñar y desarrollar una aplicación de software que sea capaz de monitorear el tráfico de red en tiempo real de los protocolos más comunes de redes diseñadas bajo el modelo TCP/IP (Transmision Control Protocol / Internet Protocol) para detectar:

- Vulnerabilidades de configuración de los nodos de red.
- Paquetes de información no deseados o dañinos.
- Qué aplicaciones se están ejecutando dentro de una red de datos.
- Direcciones IPs y puertos origen y destino con más tráfico.
- Direcciones físicas de tarjeta de red activas.
- Fechas y horas, etc.

2. Recabar la información a través de bitácoras de acuerdo a un formato que reúna las características más importantes del tráfico de red, para su posterior interpretación contemplando diversos parámetros.

3. Elaborar un trabajo que reúna información útil para la administración de redes de datos en cuanto al manejo de la información con respecto a los protocolos de transmisión de información digital más comunes de los englobados en TCP/IP.

4. Presentar una metodología de diseño para la realización de un analizador de protocolos considerando las características de una red específica.

Así, se busca que el trabajo presentado sirva como punto inicial para el desarrollo de aplicaciones referentes a seguridad en redes de datos.

La tesis se enfoca en primera instancia en presentar un panorama general del campo de las redes de datos a través del capítulo 1; en tanto que en el capítulo 2, se describe la estructura de las tramas y de los paquetes que es capaz de interpretar la aplicación, así como su comportamiento lógico a nivel general para la transmisión de información.

En el capítulo 3 se pretende explicar la metodología usada para el desarrollo y la estructura básica que debe seguir un software de este tipo en lo que se refiere a tareas generales a realizar. En este tercer capítulo también se especifican los recursos con los cuales se trabajó y que marcan al mismo tiempo los requerimientos básicos de ejecución. El capítulo 4 es el más técnico de todos ya que se enfoca en describir la organización del programa a través de los códigos fuente y su relación, así como la descripción de los bloques de código fuente indispensables para su funcionamiento. Se puede considerar al capítulo quinto como un manual básico de instalación y de uso, aunque realmente donde se explica con profundidad como se debe de ejecutar el analizador de protocolos de acuerdo a lo requerido por el usuario es en los anexos X, XI y XIII.

Por último, en el capítulo número 6 se describe el estado actual de la aplicación, así como las consideraciones a tomar por el usuario para usarlo y modificarlo en caso de requerirlo e incrementar su funcionalidad.

Como parte final del trabajo escrito se presentan las conclusiones hechas sobre el tema de tesis; los anexos necesarios para comprender de mejor forma el contenido de los capítulos y el software; un glosario de términos técnicos empleados, y la sección de fuentes de información, tanto bibliográficas como electrónicas consultadas.

Capítulo 1
Antecedentes generales

1.1 Redes de computadoras actuales

Una red de computadoras es un tipo particular de red de datos y en consecuencia su interacción con otros tipos de redes es muy grande e importante ya que posibilita en gran medida su buen funcionamiento. Esta relación es lo que permite el ritmo de vida que se presenta en nuestros días en las ciudades y lo que posibilita en gran medida el desarrollo de lugares donde se tienen diversos tipos de carencias.

Está en la esencia del ser humano la comunicación, y con el paso del tiempo la tecnología ha originado diversas formas para llevarla a cabo. Desde la primera transmisión de un mensaje por medio del telégrafo en el año de 1844 a través de una línea de comunicación, creada por Samuel Morse (ver figura 1.1), se puede considerar el inicio de la era de la electrónica en el campo de las comunicaciones. Por un poco más de 30 años el lenguaje empleado para las comunicaciones fue el *código Morse*, hasta que Emil Baudot diseñó el código de longitud constante en el que el número de elementos que conformaban cada símbolo transmitido era el mismo y por lo tanto la longitud para representar cada carácter se volvió constante. Un punto más de importancia del código de Baudot, a pesar de presentar mejoras con el paso del tiempo, radica en que sentó algunas de las bases para el diseño del *código ASCII*.

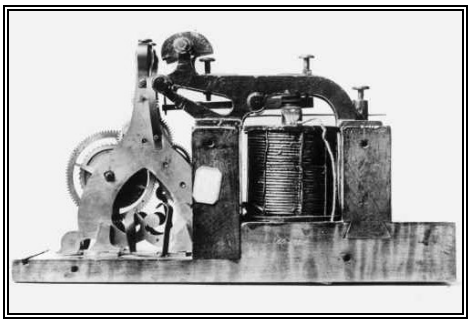

Figura 1.1. *Telégrafo de Samuel Morse*

En el año de 1877 el científico escocés nacionalizado estadounidense Alexander Graham Bell construyó el teléfono al descubrir que la voz se podía transmitir utilizando corriente continua. Después de esto, el uso del teléfono se popularizó rápidamente a niveles que a finales de la primera década del siglo XX el uso del teléfono se había difundido por casi todo el territorio de Estados Unidos. Con el paso del tiempo a estos inventos se le sumaron otros descubrimientos que permitieron el rápido avance de la transmisión de datos, incluyendo la *conmutación de mensajes* y el avance correspondiente a cada una de las generaciones de las computadoras. El siglo pasado, principalmente en su segunda mitad, marcó el inicio de la revolución tecnológica en el campo de las telecomunicaciones con la expansión de redes telegráficas y telefónicas y la generación de redes de radio, televisión, y de comunicaciones personales inalámbricas.

Hoy en día, prácticamente desde el inicio hasta el fin de las actividades diarias de una persona se presenta la influencia de las redes de datos. Desde la señal de radio o televisión de los noticiarios matutinos, redes de vigilancia de tráfico vial, el control de la red del tren subterráneo, la red de información en un banco, los trámites realizados en el gobierno, en una escuela o en una empresa, las conversaciones telefónicas inalámbricas y por medios guiados, y todos los servicios disponibles por la Internet: transacciones, publicaciones electrónicas, servicios de mensajería, transmisiones de video y audio, etc.

1.1.1 Características generales de una red de computadoras

Para dar una definición formal de una red de computadoras y sus características es conveniente establecer primero el significado de un término más general, de esta forma se presenta el siguiente concepto:

Red de datos

> Una red de datos es un conjunto de dispositivos electrónicos capaces de comunicarse entre sí ya sea por un medio físico o por un medio no guiado como el aire, y que tienen el objetivo de transmitir información y en algunos casos modificarla o tratarla de alguna forma para que se manejable por algún otro dispositivo o usuario final.

De acuerdo con esta definición las redes telefónicas, de radio y televisión son redes de datos, también conocidas como redes de telecomunicaciones, y dependiendo de las características de los servicios que ofrecen se clasifican en distintos tipos. Las redes recién mencionadas se caracterizan por tener dos puntos en común, el primero de ellos es que cada una se ocupa de transmitir un específico tipo de datos, es decir, cada una de estas redes sólo transmite ya sea señales de radio, o datos para video, o señales telefónicas. Una red de televisión transmite solamente datos que tienen sentido para receptores capaces de interpretar esta clase de señal, al igual que una red de radio sólo maneja señales de radio. Y con lo anterior llegamos al segundo punto en común de estas redes de telecomunicaciones: los dispositivos que ocupan. Cada tipo de red utiliza dispositivos de propósito especial, televisores, aparatos de radio, teléfonos, etc., y a pesar de que existen casos en que en un mismo aparato es capaz de recibir y mostrar al usuario final distintos tipos de señales, los dispositivos electrónicos que ocupan para interpretar las señales de cada red son distintos.

Así como estas características identifican a estas redes de datos, son las que definen a lo que son las redes de computadoras aunque desde diferente perspectiva. Mientras esas redes son específicas para un tipo de datos, una red de computadoras se diferencia por su generalidad en cuanto a los tipos de información que es capaz de manejar. Este tipo de red se diseña con hardware de propósito general y no está optimizada para algo en particular, como por ejemplo el tipo de información, por lo menos en su definición principal. De esta forma el concepto de red de computadoras es el siguiente:

Red de computadoras

> Conjunto de dispositivos electrónicos, conocidos en general como nodos de red, que se comunican entre sí a través de medios de diferentes características para compartir distinto tipo de información y procesarla de forma segura, y compartir recursos de software y de hardware.

De esta definición se pueden resaltar algunos aspectos:

- Conectividad y comunicación. Los usuarios de las redes de computadoras pueden comunicarse entre sí ya sea con individuos específicos o con grupos de personas en general por métodos como correo electrónico, conversaciones en tiempo real, publicaciones en sitios de Internet y transmisiones de audio y/o video, por mencionar algunos.
- Información compartida. Por medio de este tipo de red es posible transmitir información de distinto tipo entre nodos, con lo que se agilizan trámites y actividades en general.
- Recursos de hardware compartidos. Se posibilita que varios usuarios utilicen un dispositivo físico en tiempos distintos o aparentemente al mismo tiempo. Como por ejemplo una impresora conectada a una red, espacio en disco duro o el procesador de un *servidor*.
- Recursos de software compartidos. El compartir recursos de software es imprescindible para compartir recursos de hardware, sin embargo, esto también tiene su valor por sí mismo. Desde máquinas con *sistemas operativos multiusuario*,

y una gran gama de servidores (Web, de bases de datos, de algún servicio de conexión remota, o de algún otro tipo de aplicación).

- Seguridad y mejor administración de datos. La seguridad de la información se presenta en distintos niveles de acuerdo a la configuración de la red de computadoras; sin embargo los avances conseguidos en este campo son bastante buenos gracias a la cantidad de herramientas de software y de hardware disponibles hoy en día. Permite una administración de la información más dinámica y eficiente pues es posible manejar la información de forma centralizada o de manera distribuida, con los beneficios y desventajas que cada una de estas formas ofrece; y mantener respaldos actualizados de forma más rápida que con los métodos tradicionales. Sin mencionar las ventajas en el ahorro del espacio físico.
- Mejor distribución de tareas. Desde varios puntos de vista se puede tomar este punto, ya sea referente a la organización de las actividades de un sector o a la distribución de trabajo de procesamiento, con lo que tareas de determinado tipo se pueden facilitar al distribuir el trabajo entre varias computadoras.

Por otro lado, una red de datos de este tipo presenta desventajas o puntos débiles, algunos de los más comunes se mencionan a continuación.

- Costos de la infraestructura de la red. Como en muchos otros casos, el obtener un mejor desempeño de algún producto, servicio o tarea dependiendo de la inversión de esfuerzo y dinero que se haga. Estos costos, el monetario principalmente, se concentran en los elementos necesarios para la red: diseño, medios de comunicación, hardware y software.
- Costos de la administración de la red. De acuerdo a la cantidad de servicios que proporciona una determinada red será el volumen de la administración que se requiera. En una red de computadoras se deben cuidar bastantes aspectos para obtener un desempeño adecuado y aunque es verdad que algunas tareas de la administración se facilitan con el avance tecnológico, este mismo desarrollo genera nuevas funciones que administrar o diferentes formas de hacerlo. Así que son necesarios los servicios de uno o varios administradores, lo que implica el pago de cada uno de ellos.
- Distribución de información indeseable. Debido a que cada computadora está conectada por lo menos a un dispositivo de red más, es posible que reciba información no solicitada por el usuario de forma directa y que puede deteriorar el funcionamiento de dicha máquina. Tal tipo de información puede ser desde aplicaciones que proporcionan o promocionan uno o varios servicios, hasta aplicaciones como los virus de computadora que incluso pueden llegar a afectar el rendimiento de la red de las más variadas formas, como la denegación de servicios o pérdida de datos.

Una red de datos está compuesta físicamente de nodos de red y de los medios a través de los cuales se comunican (también conocidos como enlaces o *links* por su nombre en inglés), pero además es necesario contar con software que permita la configuración de la red. De forma general se describen los elementos de una red de computadoras.

Dispositivos o nodos de red. Dispositivos electrónicos que permiten el procesamiento de información ya sea para los usuarios finales (nodos finales) o para otros dispositivos, actuando de esta forma como intermediarios en la transmisión de información entre nodos finales.

Host. Se denomina host a una computadora conectada a una red y plenamente identificada a través de una dirección específica a dicha red.

Medio de transmisión o de comunicación. Termino que se aplica al medio a través del cual los datos son transmitidos entre nodos de red. Por ejemplo cables de cobre.

Interfaz de red. Se refiere al dispositivo que permite la conexión de un nodo de red a un medio o enlace de comunicación.

Sistema operativo de red. Un sistema operativo es software que permite usar distintas aplicaciones, pero también administrar recursos de hardware de forma directa o por medio de software extra (controladores) para algunos elementos. De esta manera, un sistema operativo de red es software capaz de administrar recursos de hardware y software necesarios para el funcionamiento de una red de datos.

Software para servicios de red. En esta categoría se encuentran aplicaciones necesarias para actividades específicas. Software que preste algún servicio (*servidor*) y software que utilice dicho servicio (*cliente*).

Actualmente existen muchas herramientas que permiten realizar diseños de red de forma esquemática con gráficos predeterminados y añadir otros puntos relevantes; sin embargo, de forma básica cualquier nodo de red sin importar si es un host u otro dispositivo se representa con una circunferencia, y el medio físico de conexión entre nodos con una línea, con se muestra en la figura 1.2:

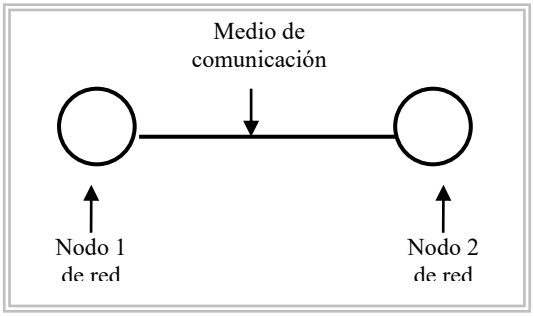

Figura 1.2. *Esquema básico de 2 nodos conectados por un medio de transmisión.*

Por otra parte, es común representar a una red de datos de forma gráfica como una nube ya que no se conoce su estructura o no es necesario especificar dispositivos de red conectados entre sí, ni su cantidad o tipo de configuración. En la figura 1.3 se ilustran 3 redes interconectadas por algunos nodos de red.

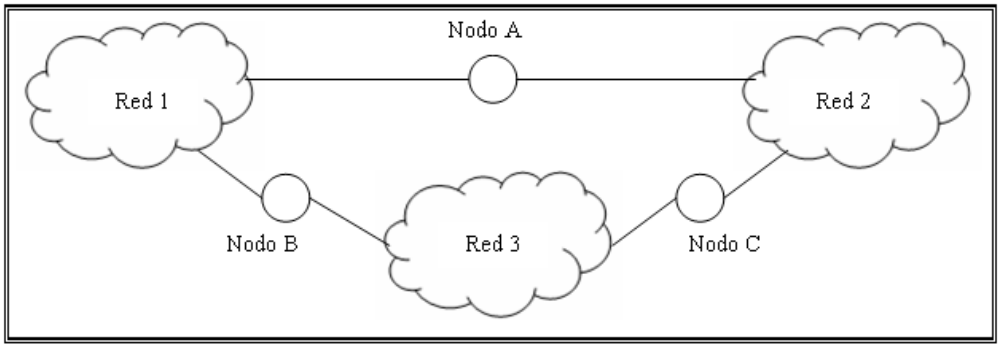

Figura 1.3. *Interconexión de las redes 1, 2 y 3, por los nodos de red A, B y C*

1.1.2 Conexión y transmisión de información

La conexión entre nodos es uno de los aspectos esenciales de una red de computadoras y depende del uso que se le destine a la red el nivel de conectividad. En forma simple, la conectividad se refiere a la forma en como están conectados los nodos de la red. Un bajo nivel de conectividad se presenta cuando la red está conformada por pocos nodos, por ejemplo 2 computadoras conectadas a través de un solo medio de conexión, *cable UTP* por decir alguno. Por otra parte, mientras más nodos compartan el medio de comunicación el nivel de conectividad aumenta. Básicamente existen dos tipos de conectividad, los cuales se representan en la figura 1.4 y se definen a continuación.

Conectividad punto a punto

Cada nodo de la red está conectado a través de un enlace de comunicación a sólo un nodo más.

Conectividad multipunto o de acceso múltiple

Cada nodo de red está conectado a un enlace que comparten dos o más nodos, con lo cual cada uno puede comunicarse con más de un nodo.

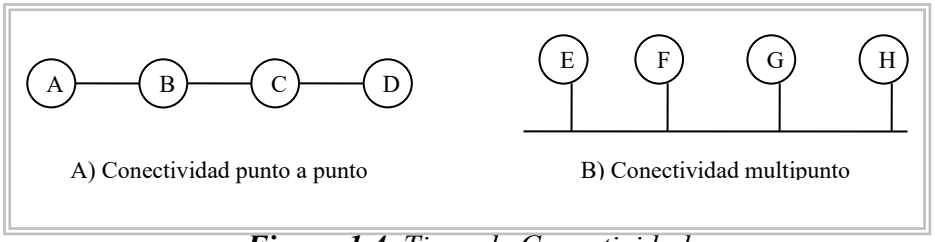

Figura 1.4. *Tipos de Conectividad*

En una red pueden presentarse ambos tipos de conectividad, y la comunicación entre dos nodos no es necesariamente de forma directa sino que puede darse a través de un nodo intermediario que se encarga de retransmitir el mensaje que le llega. Cuando el funcionamiento de una red es de esta forma, se dice que se trata de una *red conmutada* (*switched network*). De las redes conmutadas, las más conocidas son las de circuito

conmutado (*circuit switched*), en las cuales necesariamente se debe comprobar la existencia del nodo destino y para ello se establece un circuito de prueba formado por enlaces de comunicación y nodos intermediarios; y en caso de que esté disponible dicho nodo final los datos se transmiten a través del circuito de prueba. Otro tipo de red es la que se conoce como red de conmutación de paquetes (*packet switched*), y en la cual la información se divide en piezas (paquetes de datos) que son enviados a un nodo final a través de los nodos de la red por distintas rutas. Los dispositivos intermediarios almacenan el paquete recibido en su memoria interna y después lo envían a otro nodo para que haga lo mismo o envíe al nodo destino. Estos tipos de redes se ejemplifican en las figuras 1.5, para el *circuit switched*, y 1.6 para el *packet switched*.

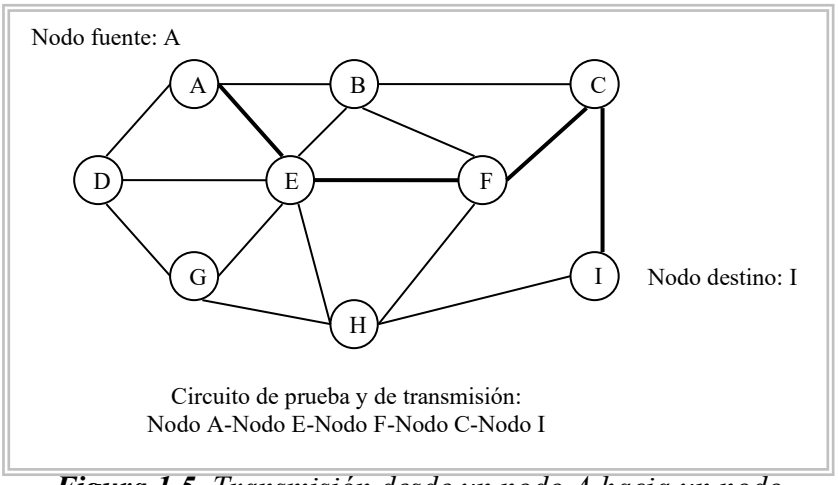

Figura 1.5. *Transmisión desde un nodo A hacia un nodo*

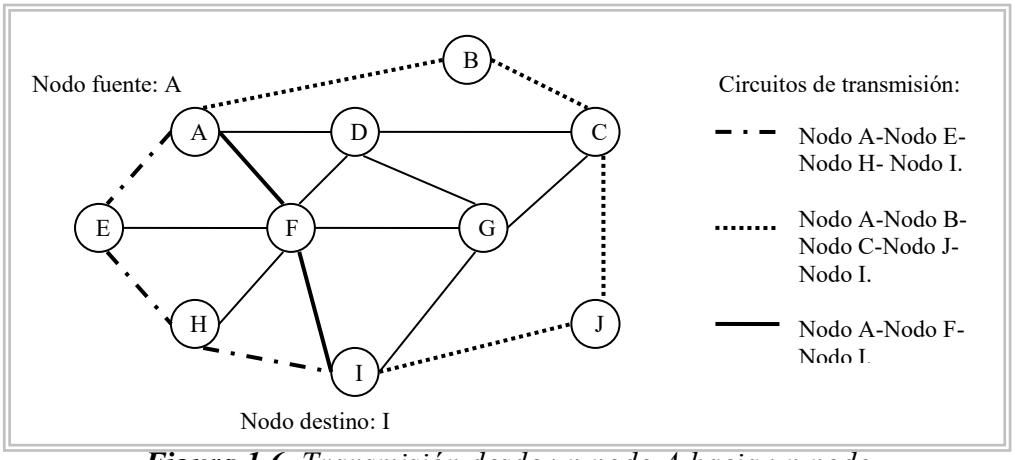

Figura 1.6. *Transmisión desde un nodo A hacia un nodo*

La red de circuito conmutado tiene como ventaja el asegurarse de la existencia de nodo destino, sin embargo durante esta fase de verificación se consume un tiempo más que en la de conmutación de paquetes. Y esta última tiene como ventaja el tener menor costo para implementarse y facilitar servicios digitales de comunicación. Ejemplo típico de una red *circuit switched* es una red telefónica; y de una red *packet switched* es una red de área local.

1.1.2.1 Tipos de comunicación

Existen tres tipos básicos de comunicación entre dos nodos para las redes de datos.

Comunicación simplex. La comunicación se da en un solo sentido, de un nodo que proporciona un servicio a un nodo que lo recibe. El nodo destinatario no hace una petición del servicio, sino que siempre que está activado se mantiene a la escucha de la señal emitida por el nodo transmisor aún y cuando no haya señal transmitida. Este tipo de comunicación se da por ejemplo en las transmisiones comerciales de radio.

Comunicación half duplex. La comunicación se lleva a cabo en ambos sentidos pero en diferentes tiempos para ello se establece un turno para cada nodo que desee transmitir información. Un ejemplo es la comunicación con dispositivos del tipo *walkie-talkie*.

Comunicación full duplex. La comunicación es posible en ambos sentidos y al mismo tiempo entre los dos nodos que participan en la transmisión de información. Un ejemplo de este tipo de comunicación son las conversaciones telefónicas.

Las redes de computadoras son capaces de enviar información por la comunicación de tipo full duplex.

1.1.2.2 Métodos de transferencia y control de transmisión

Modos de transferencia es como comúnmente se conoce a la transmisión de datos dependiendo de a cuantos destinatarios se dirija la información. Cuando los datos son enviados a un solo nodo, el modo de transferencia empleado es *Unicast*, sin importar si la comunicación es de tipo simplex, half duplex, o full duplex. El modo de transferencia de tipo *Broadcast* se presenta cuando la información transmitida tiene como destinatario a cada uno de los nodos que integran la red de datos. El tercero y último es el *Multicast* y se presenta cuando los datos se envían a más de un nodo de la red pero no a todos los que la conforman, es decir, a sólo una cierta parte de la red.

Como se mencionó anteriormente una red de computadoras ofrece distintos tipos de servicios a sus usuarios y dichos servicios son proporcionados por los procesos (*servidores*) de algún software. El *control de transmisión* de una red de computadoras se refiere a la forma en que se distribuyen esta clase de procesos en la red, mediante la instalación y ejecución de este software en los nodos de red. De esta manera, el control de transmisión se puede realizar de dos formas.

Control de transmisión centralizado. Con esta configuración los *servidores* para distintas aplicaciones residen en un solo equipo, presentando como punto a favor un menor costo en lo que se refiere a equipo de cómputo donde residan estos servicios. Este término también se aplica cuando la comunicación de la red depende de un dispositivo único al ser el que distribuye la información entre los demás nodos, de tal forma que se centraliza su control. Tiene como punto débil la dependencia de un único nodo para algún tipo de servicio. De forma esquemática este tipo de control se representa en la

figura 1.7, en la que un nodo A reúne 2 servicios que ocupan todos los hosts de la red (Nodos C, D y E).

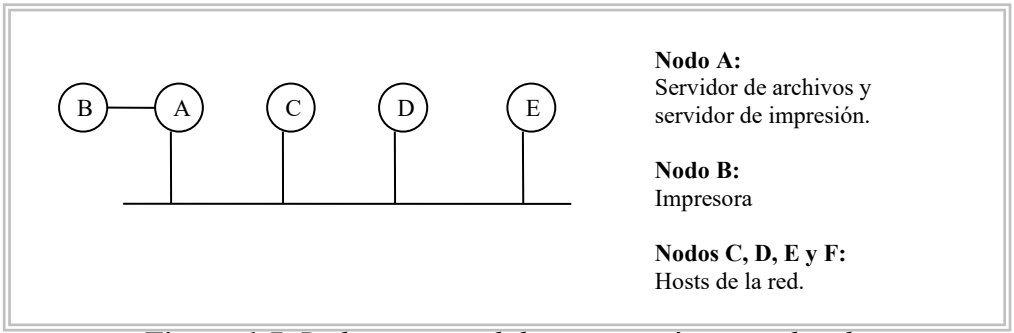

Figura 1.7. *Red con control de transmisión centralizado.*

Control de transmisión distribuido. Se realiza una distribución de los *servidores* de la red en varios nodos con lo que se evita la dependencia de solamente un nodo. Sin embargo, esto trae consigo un mayor esfuerzo de administración e inversión para los equipos. En la figura 1.8 se ilustra la idea anterior.

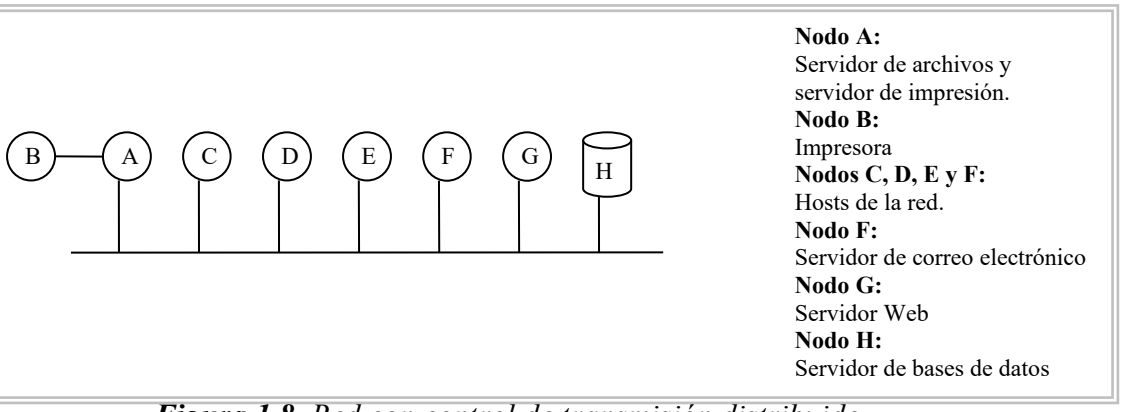

Figura 1.8. *Red con control de transmisión distribuido.*

1.1.3 Tecnologías de red

Una tecnología de red es una norma que establece características tanto físicas como lógicas que deben seguir las configuraciones de red para tener un funcionamiento de buena calidad. Hace referencia a los medios de transmisión, a los dispositivos de red que se ocupan, conectores, disposición de los elementos de red, fases de comunicación de los hosts y métodos de monitoreo de la red, detección y solución de errores. Estas normas de red las establecen organismos internacionales como la *IEEE*, Institute of Electric and Electronics Engineers. A continuación se tratarán los puntos esenciales del estándar de red más común para redes de área local diseñadas con medios guiados y que ocupan algunas de las topologías de red descritas en el anexo II de este trabajo. Una revisión de las características generales de las redes clasificadas por cobertura se encuentra en el anexo I.

1.1.3.1 Ethernet

Este estándar se aplica para las topologías de bus y de estrella; las diferencias en cada caso son generadas en principio por el medio de transmisión empleado. El estándar 802.3 es el más popular para configuraciones de redes LAN.

Ethernet, como tecnología de red, fue desarrollado alrededor del año 1975 por investigadores del Xerox Palo Alto Research Center, que se basaron en un proyecto llamado Aloha de la Universidad de Hawai. Este primer conjunto de reglas hacia referencia al *cable coaxial* como único medio de transmisión y a la topología de bus, pero con el paso del tiempo fue modificándose y complementándose hasta obtener el estándar actual. Existen varios apartados generales de esta tecnología, diferenciados claramente por el medio de transmisión, cada uno de estos 3 a su vez se divide en varias especificaciones.Los nombres que identifican a cada especificación siguen siguientes reglas de sintaxis. El primer número que se presenta indica el ancho de banda en Mbps de cada una. La palabra base señala que la transmisión de datos es de forma digital o en banda base. Los caracteres que le siguen refieren el medio de transmisión usado y caracerísticas propias de cada especificación. En caso de ser sólo un número refiere a la cantidad de cientos de metros que el segmento de red se puede extender sin usar algún repetidor de la señal; una letra indica el medio de transmisión usado por la Ethernet, T para cable de par trenzado y F para cable de fibra óptica. Si no se indica una letra se considera que el medio de transmisión es cable coaxial.

Así, algunas de las epecificaciones son:

- 10Base2 utiliza cable coaxial, tiene un ancho de banda de 10 Mbps y se puede extender hasta 200 metros.
- 10Base5 utiliza cable coaxial, tiene un ancho de banda de 10 Mbps y se puede extender hasta 500 metros.
- 10BaseT utiliza cable de par trenzado, transmite a 10 Mbps y la longitud máxima sin ocupar repetidores es de cien metros.
- 100BaseTX utiliza cable de par trenzado y la longitud máxima sin ocupar repetidoers es de cien metros. Tansmite a 100 Mbps.
- 100BASE-FX trasmite a 100 Mbps a través de fibra óptica.
- 1000BASE-T: 1 Gbit/s over Category 5e copper cabling.
- 1000Base-SX para fibra óptica multimodo.
- 1000Base-LX para fibra óptica monomodo.

Cuando una Ethernet utiliza cable de par trenzado la configuración física no es en forma de bus sino es en forma de estrella, y el dispositivo central es un repetidor al que se conectan los hosts, que también puede estar conectado a otro repetidor.

1.1.3.1.1 CSMA/CD

CSMA/CD son las siglas de *Carrier Sense Multiple Access/Collision Detect* y es un algoritmo que se ocupa para controlar el acceso a una red Ethernet, es decir para determinar la forma en como se transmite la información. Entre los puntos principales de este control

de acceso al medio se encuentra el formato en que se transmiten los datos, ya que éstos no se envían de forma independiente sino que se adjunta otra información necesaria para una comunicación exitosa; a este conjunto de información se le denomina *Frame Ethernet* y tiene la siguiente estructura:

- Un campo de 64 bits llamado Preámbulo que permite al nodo destino sincronizarse con la señal.
- Dos campos de 48 bits, uno para especificar una dirección que identifique al nodo destino, y el otro para indicar la dirección del nodo que transmite el frame.
- Un campo de 16 bits para indicar el tipo de protocolo que debe manipular el frame en el nodo destino.
- Un campo de longitud variable que contiene la información codificada en forma de bits.
- Un campo de *Cyclic Redundancy Check*, que se usa para determinar errores de transmisión.

A los campos de direcciones fuente y destino y al de tipo se les llama en conjunto cabecera del frame, lo cual se ilustra en la figura 1.9.

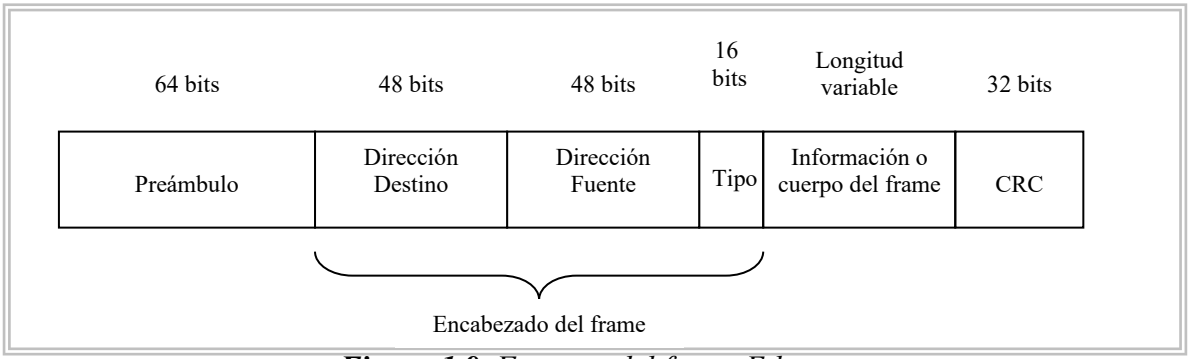

***Figura 1.9**. Formato del frame Ethernet.*

Como mínimo el campo de datos debe ser de 46 bytes. Las direcciones fuente y destino recién mencionadas identifican a los hosts que están involucrados, sin embargo éstas no pertenecen en un sentido estricto a los nodos sino a la interfaz que le permite a cada uno conectarse a la red. La dirección de cada interfaz se forma con 48 bits y para un fácil manejo se expresa como 6 grupos, cada uno con 2 números en base hexadecimal. Un ejemplo de una dirección MAC o Ethernet, como se les conoce generalmente, es la siguiente:

00-0A-E6-22-B3-23

Que en base binaria es:

00000000-00001010-11100110-00100010-10110011-00100011

Cada interfaz tiene una dirección MAC única ya que a cada empresa fabricante se le asigna un número en binario de 24 bits diferente al de los demás y que sirve como prefijo a las direcciones MAC de las interfaces que elabora para completar los 48 bits de longitud.

Todos los frames que circulan por el bus son revisados por todas las interfaces de red conectadas a él, pero sólo la interfaz con la dirección a la que va dirigido lo transmite al host en el cual está instalado. Cada interfaz es capaz de reconocer que un frame va dirigido hacia ella cuando tiene su dirección MAC en el campo de dirección destino, cuando el campo de dirección destino tiene una dirección de broadcast (todos los bits ajustados a 1), o cuando tiene una dirección de multicast (primer bit ajustado a 1, pero no los demás); o cuando la interfaz está configurada en modo *promiscuo*, lo que significa que interpretará que todos los frames van dirigidos hacia su dirección aún cuando no sea así.

Uno de los puntos que indica el CSMA/CD es que para que un nodo pueda enviar datos tiene que revisar primero que el bus de datos se encuentre libre de cualquier otra transmisión. En caso de que detecte que hay un frame ocupando el bus no enviará su frame al bus hasta que sea desocupado. Debido a este censo de disponibilidad del bus es el nombre de *Carrier Sense.* Debido a que dos nodos o más pueden detectar en un determinado momento que el bus de datos está libre, puede darse el caso de que envíen al bus al mismo tiempo su frame de datos con lo que se provocará una colisión de datos y en ese caso cada interfaz la detectará debido a la transmisión del campo de preámbulo de su frame y en lugar de continuar transmitiendo el resto del frame transmitirá una secuencia de 32 bits llamada secuencia de interferencia (*jammming sequence*). De esta forma cada nodo transmitirá como mínimo 96 bits de datos, 64 del campo de preámbulo y 32 de la secuencia de interferencia.

1.2 Arquitectura de red

En una red de computadoras se llevan a cabo muchas tareas para permitir la comunicación entre los nodos. Dichas actividades se realizan con el propósito de controlar los datos desde las aplicaciones que los generan o los ocupan hasta los dispositivos de hardware que transmiten la información por los enlaces de comunicación; y consisten en la verificación de la integridad de la información, su codificación a un formato especial, distribución a los procesos de cómputo adecuados o establecimiento, manejo y terminación de una comunicación, por mencionar algunos.

La arquitectura de red se refiere a la manera en como se llevan a cabo todas las actividades de red del tipo mencionado, es decir, especifica el tratamiento de la información en todo el proceso de comunicación.

Para diseñar el funcionamiento de una red se tiene que conocer o establecer cada etapa que se llevará a cabo durante la comunicación y con base en ello hacer una abstracción; cada elemento realizará una tarea y será capaz de interactuar con otros elementos de una abstracción general. De cada abstracción se establece un *nivel* o una *capa* que realizará tareas de un solo ámbito, con lo cual se divide el problema de comunicación en varios elementos manejables de forma más fácil que si sólo se tuviera un elemento general para realizar todo el procedimiento y que permite mayor flexibilidad para realizar modificaciones e incluso añadir nuevas funcionalidades. En cada capa, al conjunto de operaciones que da un tratamiento especial a la información, la interpreta y permite su

intercambio entre capas de la arquitectura de red se le llama *protocolo*. Cada nodo tiene una arquitectura de red que puede ser la misma del nodo con quien se está comunicando o puede ser diferente debido a que implementan un protocolo de distintas formas, sin embargo deben ser capaces de intercambiar información.

Un protocolo establece dos interfaces de comunicación, la primera es una *Interfaz de Servicio* y permite comunicarse con otras capas que necesiten de su servicio dentro del mismo nodo, es decir, define los procedimientos que se llevarán a cabo en la capa de acuerdo a las necesidades de otros elementos en la arquitectura de red. La segunda interfaz se conoce como *Peer Interface* (Interfaz de igualdad), que marca las reglas de comunicación, lenguaje y significado de la información entre capas del mismo tipo (en realidad de la misma capa cuando dos nodos trabajan con la misma arquitectura de red). De esta forma, una capa presta sus servicios a la capa superior de su misma estructura de red y se comunica con una capa igual a ella que se presenta en el otro host. A excepción de la capa de la estructura de red que abarca aspectos de hardware, todas las demás capas se intercomunican con su correspondiente en el otro nodo de forma indirecta, trasmitiendo su información a la capa de nivel inferior hasta llegar a la capa de hardware. Esta interacción entre capas se da por la necesidad de la comunicación entre los protocolos existentes en cada una y dependiendo de la arquitectura de red la cantidad de protocolos varía. La siguiente imagen (Figura 1.10) describe algunos puntos de los ya mencionados. En ese caso la interfaz de igualdad consistir de un enlace punto a punto entre los 2 hosts o de una red intermediaria.

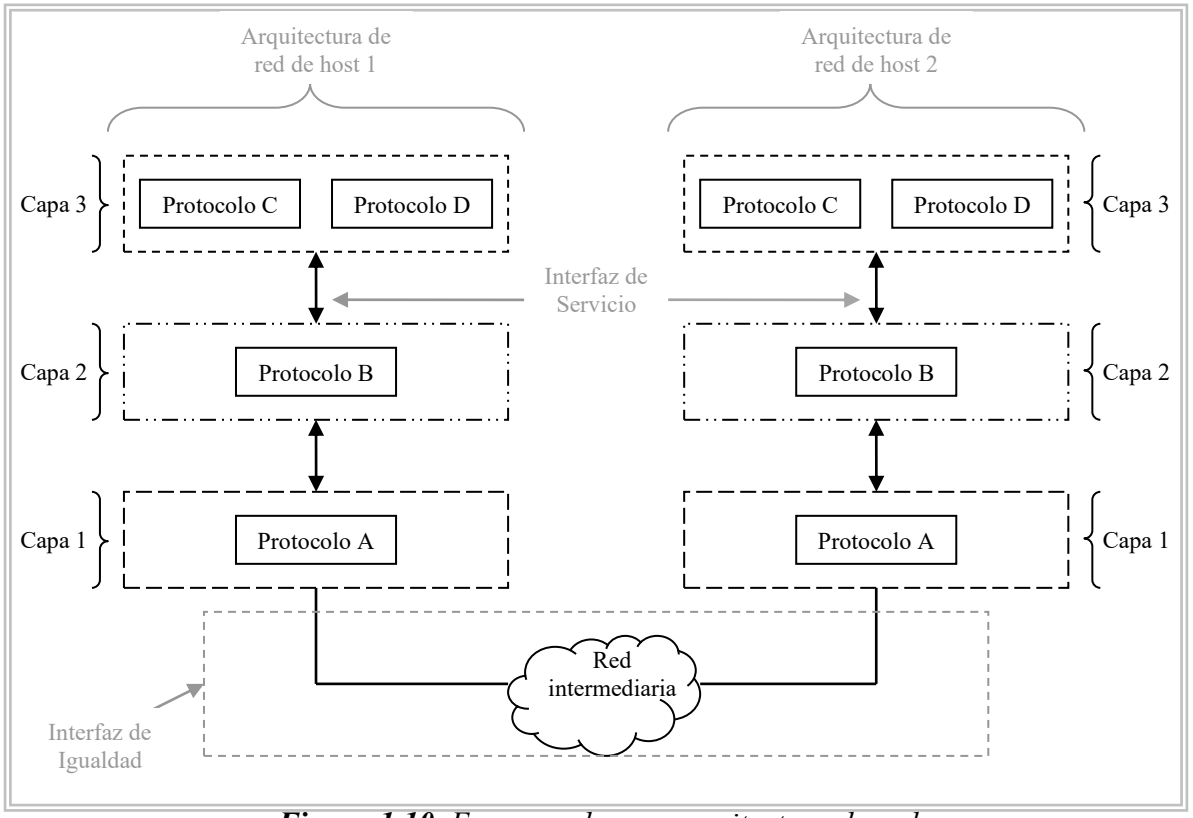

***Figura 1.10**. Esquema de una arquitectura de red.*

En una arquitectura se llama ***Gráfica de Protocolos*** al esquema que representa a todos los protocolos de cada capa y a las posibles interacciones entre cada capa; mientras que al conjunto de protocolos utilizados para llevar a cabo una comunicación entre host se le llama ***Pila de Protocolos*** (Protocol Stack). En la figura 1.11 se muestra una gráfica de protocolos donde se resalta la pila de protocolos formada por el enlace *Protocolo G-Protocolo E-Protocolo C-Protocolo A*.

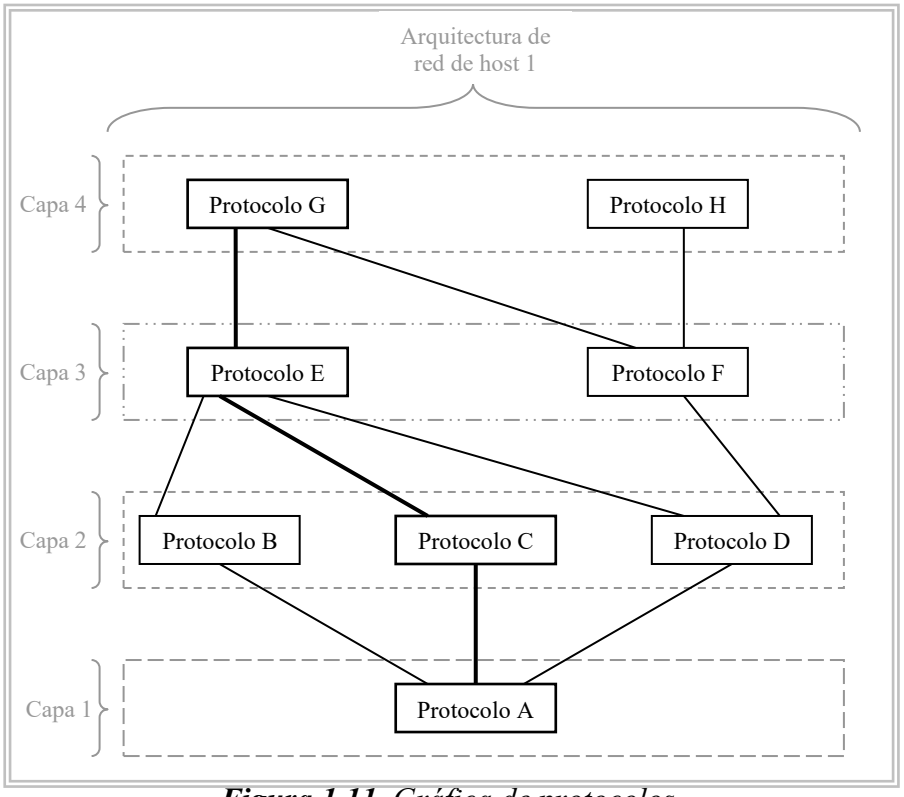

Figura 1.11*. Gráfica de protocolos.*

Para transmitir los datos de un host se lleva a cabo un procedimiento en cada nivel de la gráfica de protocolos, determinado por la pila de protocolos en cuestión; sin embargo siempre se presenta algo en particular en cada nivel que permite la comunicación entre capas correspondientes de la arquitectura de red en el host fuente y en el host destino. En el host que envía la información se lleva a cabo un encapsulamiento en cada capa de los datos que recibe, es decir, de la capa inmediata anterior. En el host origen los datos fluyen de la capa más alta a la más baja y en el host destino, el que recibe los datos transmitidos, sucede de forma contraria, el flujo es del nivel más bajo al más alto. Es claro que ambos host *intercambian los papeles* para alternarse como host fuente y destino para llevar a cabo la comunicación. El protocolo más alto de la pila de protocolos recibe los datos a transmitir pero antes de pasarla al protocolo de la capa inmediata inferior le añade información con el propósito de indicar al protocolo correspondiente en la misma capa, pero del host destino, como manejar dicho mensaje. Así una capa interactúa con su correspondiente en el otro nodo con esta información de control que puede consistir desde unos pocos bytes hasta algunas docenas de bytes; cuando esos datos se adjuntan al principio de los datos, mejor conocido como *cuerpo del mensaje* o *payload*, se le llama *encabezado* (header) y cuando se coloca al final se conoce como *trailer*. Para ejemplificar lo anterior se puede considerar a la

pila de protocolos *Protocolo G-Protocolo E-Protocolo C-Protocolo A* de la figura 1.11; en dicho caso el protocolo G recibe un mensaje para transmitir y le añade información de control como encabezado para después pasarlo a la capa 3 para ser tratado por el protocolo E que también le antepone información que sólo se puede interpretar en la misma capa del nodo destino, ya que sólo tiene sentido para el protocolo E. Sucesivamente esta acción se presenta hasta e inclusive en la capa más baja de la pila y entonces es enviado el mensaje por el enlace de transmisión. Una vez en el host destino cada capa leerá la información de control correspondiente, determinada por el protocolo que se haya utilizado y ejecutará lo indicado por ella antes de pasar el mensaje; ya sin la cabecera o *trailer* el mensaje será enviado a la capa superior donde un protocolo interpretará la información anexada por la misma capa en el host origen, y pasará el mensaje a la capa inmediata superior, lo cual se repetirá hasta llegar a una aplicación de software a la que se entregará el mensaje tal cual fue emitido inicialmente. Un proceso de comunicación de este tipo se ilustra en la figura 1.12.

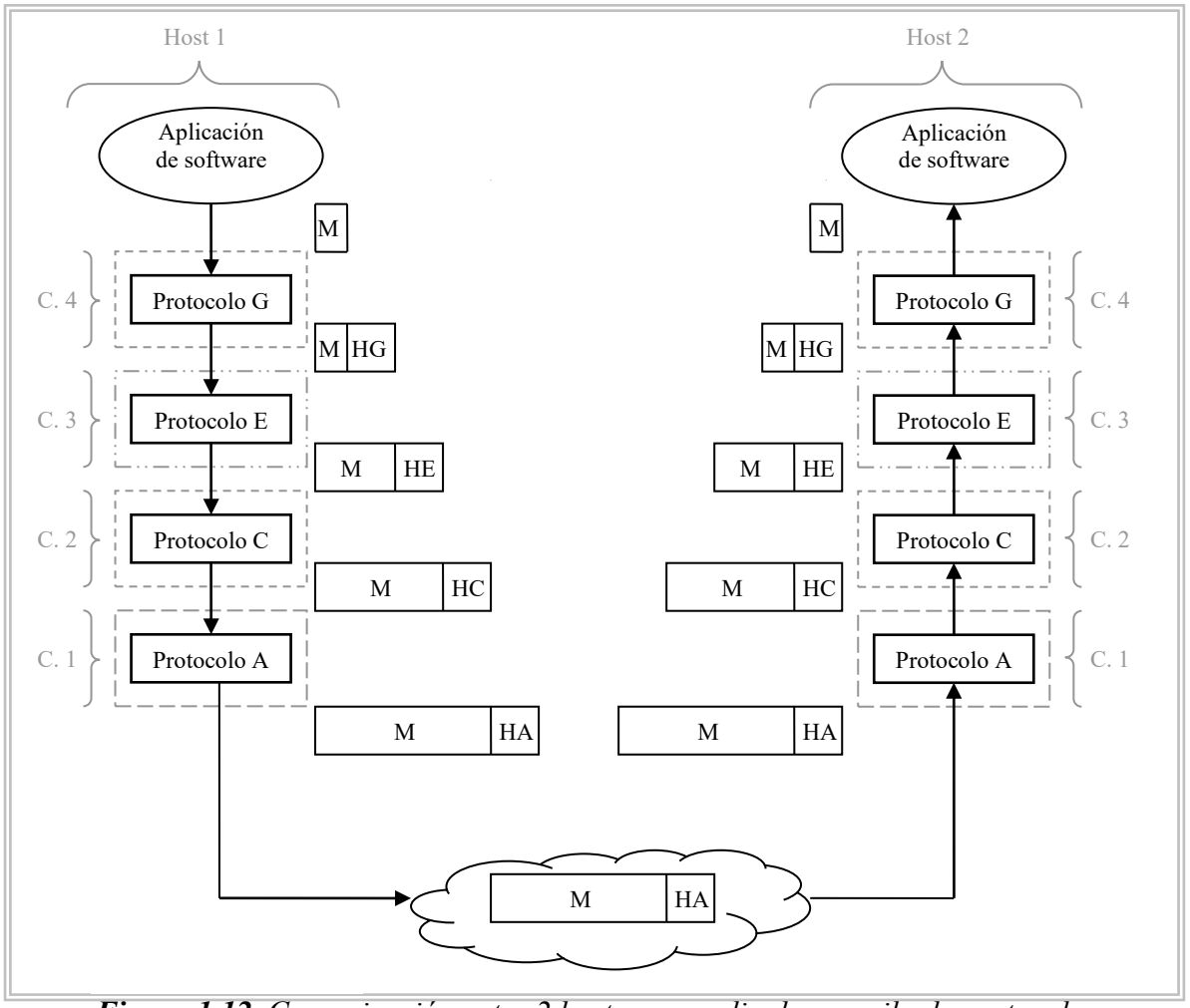

***Figura 1.12**. Comunicación entre 2 hosts por medio de una pila de protocolos.*

En la figura 1.12 se puede ver que conforme el mensaje pasa a través de las capas, el protocolo agrega un encabezado (HG, HE, HC y HA de los protocolos G, E, C y A, respectivamente); en cada capa se toma como mensaje a los datos que recibe incluyendo las cabeceras o *trailers* agregados anteriormente. Esta información extra a la información original tiene varias utilidades, que dependen del protocolo que la haya agregado; y una de ellas es para determinar a qué protocolo de la capa superior, en caso de presentarse más de uno, se debe de pasar la información, o a qué aplicación deben destinarse los datos en el host destino. De forma general se conoce como *clave de demultiplexaje* (*demultiplexing key*) a los bytes que permiten realizar dicha tarea. Algunos protocolos usan un campo de 8 bits, otros de 16 y en algunos casos se puede tener un campo de 32 bits; de igual manera pueden tener un campo de demultiplexaje que se utiliza en los dos hosts, o dos campos, uno para usarse en cada host.

Existen muchas arquitecturas de red que pueden permitir la comunicación entre dos aplicaciones de distintos nodos, cada una ofreciendo distintas características; sin embargo se ha optado por usar estándares para uniformizar los métodos de comunicación. La Organización Internacional de Estándares (*Internacional Standars Organization, ISO*), el Grupo de tareas de Ingeniería de Internet (*Internet Engineering Task Force, IETF*) son organizaciones reconocidas que definen la normatividad para algunas arquitecturas de red. El modelo OSI (*Open Systems Interconection*) o Arquitectura OSI es el estándar definido por la ISO para la interconexión entre nodos de red. Se conforma por siete capas, de las cuales las 3 inferiores se implementan en cada nodo de red ya sea un host o un switch. La más alta es la *Capa de Aplicación* que involucra a aplicaciones software que manejan, generan o presentan la información transmitida por la red; la *Capa de Presentación*, básicamente¸ refiere al formato con el cual se manejarán los datos a transmitir, por ejemplo longitud de un número entero en bits, tipo de codificación (ASCII, EBCDIC, etc.), u orden de los datos para enviarlos. La capa 5, *Capa de Sesión*, lleva a cabo tareas que permiten los flujos de información entre los host finales, es decir, administra desde el inicio hasta su desenlace la conexión lógica definiendo métodos de comunicación. La capa 4 se conoce como *Capa de Transporte* lleva a cabo métodos de corrección de errores y de control de flujo de la información; en esta capa los datos transmitidos entre nodos finales se conocen como *mensaje*. Los métodos de estas 4 capas (de Transporte, de Sesión, de Presentación y de Aplicación) se llevan a cabo sólo en los host finales. La Capa Física es la capa más baja de la arquitectura de red y en ella se definen aspectos para la codificación de los datos en flujos de bits, representados por niveles de voltaje. La Capa de Datos, capa 2, define conjuntos de información llamados *frames* y lleva a cabo métodos de detección de errores como FCS (Frame Check Sequence) y análisis de *Cyclic Redundancy Check* (CRC). Esta capa se implementa en las interfaces de red de los nodos. La tercera capa se conoce como *Capa de Red* y realiza tareas de conmutación, por lo que la información intercambiada entre nodos, a este nivel, se denomina *paquete* en lugar de frame. En la figura 1.13 representa un esquema del modelo OSI.

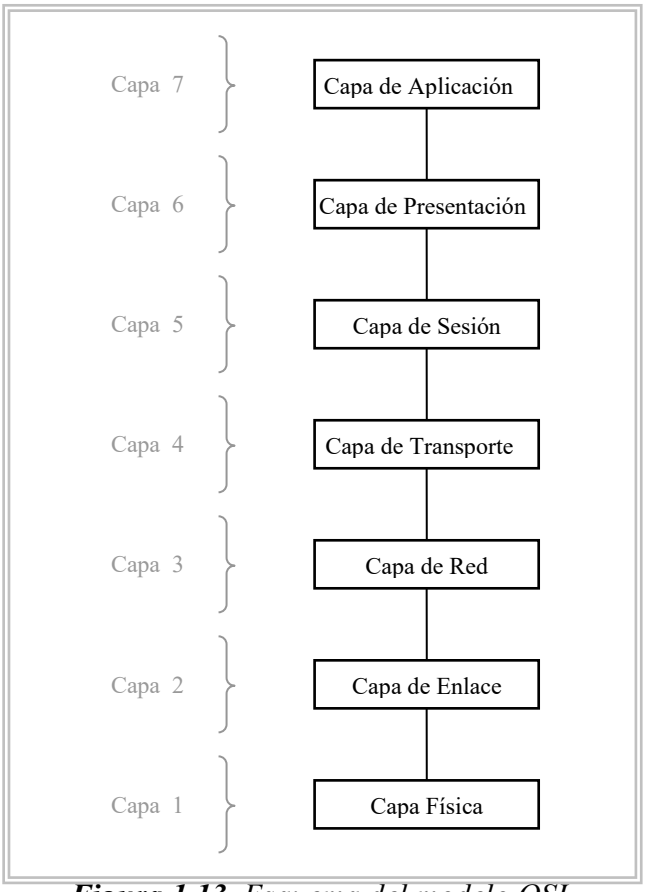

***Figura 1.13**. Esquema del modelo OSI.*

1.3 Identificadores de red

Esta sección tiene como propósito describir de manera general los mecanismos que identifican a los nodos durante la fase de comunicación y que permiten establecer claramente quienes son los elementos que conforman la red de datos. En la sección 1.1.3.2 al hablar del método de acceso al medio CSMA/CD ya se abordó un tipo específico de identificadores de red, el de las direcciones de las interfaces de red (MAC Address). Sin embargo esta forma para identificar a los nodos sólo se puede utilizar para redes LAN que ocupen la tecnología Ethernet.

En las redes WAN el mecanismo dominante para identificar a los nodos de red es el de la tecnología IP, basada en el protocolo IP y que en el modelo OSI se ejecuta en la capa de red. La versión que predomina de este estándar es la 4, por lo que se le llama IPv4, sin embargo poco a poco durante la reciente década se ha estado migrando a la versión 6. Aunque una dirección IP realmente reside en una interfaz de red no es el mismo caso de una dirección MAC, ya que no está grabada en un chip y por lo tanto no es dependiente de dicha interfaz, sino que una dirección IP puede ser asignada a cualquier interfaz de red y por ende al nodo al que conecta a la red, siempre y cuando no haya dentro de la misma red otra interfaz que la esté usando en ese momento.

1.3.1 Direcciones de red IPv4

Una dirección de red IPv4 se compone de 32 bits de datos separados en 4 grupos de 8 bits cada uno, denominados bytes. La estructura de las direcciones IP es jerárquica ya que cada byte representa una jerarquía en la red; cada dirección IP de esta versión se compone de 2 partes, una que identifica a la red de la que forma parte dicha dirección y otra que identifica específicamente al host. La parte que identifica al host tiene como característica principal que es única dentro de la red. Para facilitar el manejo de las direcciones IP comúnmente cada octeto se traduce al equivalente decimal y se separa de los demás por medio de un punto; de tal forma que una dirección IP que en formato binario sea:

10101101110111011101110111010101

al separar los bits en grupos de 8 se puede escribir como:

173.221.221.213

Existen 5 clases de red de acuerdo al estándar IPv4 y difieren entre sí debido a la estructura del primer byte.

En una dirección IP de una red Clase A el primer octeto inicia (de izquierda a derecha) con un bit ajustado a un valor cero, y los demás bits pueden tomar cualquier valor. Una red clase B tiene la característica de que el primer byte de cada una de sus direcciones IP inicia con sus 2 primeros bits ajustados a *10* y los otros 6 bits pueden tomar cualquier valor; para una dirección IP de red clase C los 3 primeros bits del primer byte son forzosamente *110*. Las direcciones IP de los otros 2 tipos de red se ocupan con fines distintos a los de las 3 primeras clases ya que las de Clase D se usan para labores de *multicast* y las de clase E se diseñaron para usarse en algún futuro. Las direcciones IP de clase D tienen los primeros 4 bits del primer byte ajustados de la siguiente manera *1110*, mientras que las direcciones de clase E los tienen ajustados a un valor *1111*. Para cada una de estas clases se define una *máscara de red* o *Netmask*, la cual tiene el formato de una dirección IPv4. En notación decimal las máscaras de red para las clases A, B y C son *255.0.0.0*, *255.255.0.0* y *255.255.255.0* respectivamente.

La siguiente figura ilustra la estructura de las direcciones IP de acuerdo a la clase de red a la que pertenecen:

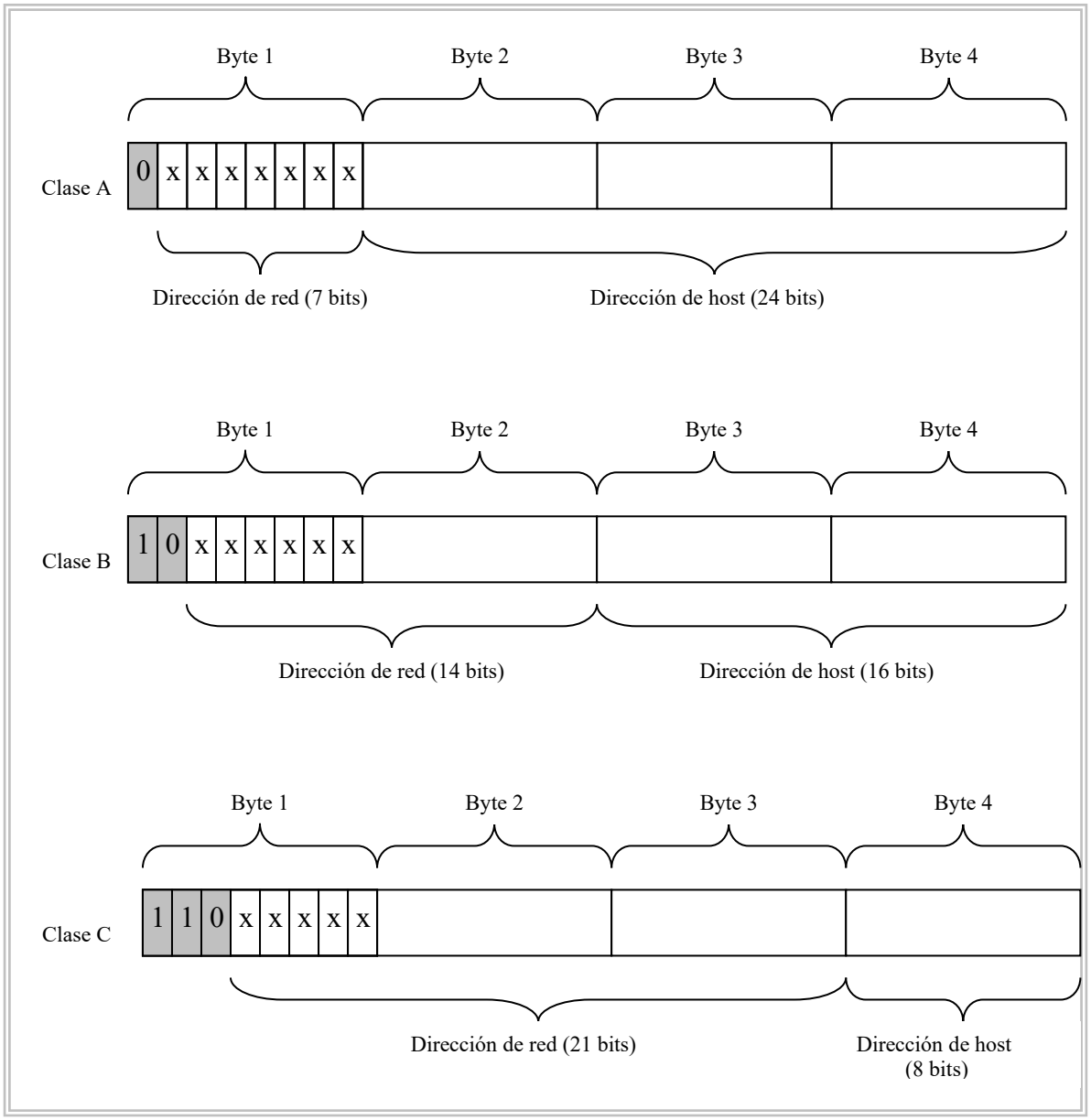

***Figura 1.14**. Direcciones de red de Clase A, B y C.*

En la tabla 1.1 se presenta el rango de direcciones para cada clase de red sin hacer distinción de direcciones IP reservadas.

Rango de direcciones	Clase de red
0.0.0.0-127.255.255.255	A
128.0.0.0-191.255.255.255	B
192.0.0.0-223.255.255.255	C
224.0.0.0-239.255.255.255	D
240.0.0.0-255.255.255.255	E

***Tabla 1.1.** Rango de direcciones para las clases de redes IPv4.*

De acuerdo a la cantidad de bits de la dirección de red clase A se podrían tener 2^7 posibles direcciones, es decir 128 redes clase A, cada una puede albergar hasta una cantidad de 2^{24}-2 hosts pues se reservan 2 direcciones, una para identificar a la red y otra dirección de broadcast. Por ejemplo la dirección 101.0.0.0 identifica a toda la red clase A y la dirección 101.255.255.255 es la dirección de broadcast. En el caso de las redes clase B cada una puede tener 2^{16} -2 hosts, debido a la reserva de la dirección de broadcast y a la que identifica a la red; en total puede haber hasta 2^{14} direcciones de red clase B. En el caso de las redes clase C puede haber hasta 2^{21} redes clase C, cada una con 2^8-2 hosts.

Existen rangos de direcciones que también tienen un uso especial, éstos se muestran en la tabla 1.2.

Rango de direcciones	Uso	Clase de red	Direcciones posibles
0.0.0.0 - 0.255.255.255	Dirección Cero	A	16,777,216
10.0.0.0 - 10.255.255.255	Direcciones IP no homologadas	A	16,777,216
127.0.0.0 - 127.255.255.255	Dirección Localhost Loopback	A	16,777,216
169.254.0.0 - 169.254.255.255	Zeroconf	B	65,536
172.16.0.0 – 172.31.255.255	Direcciones IP no homologadas	B	1,048,576
192.0.2.0 - 192.0.2.255	Documentación y ejemplos	C	256
192.88.99.0 - 192.88.99.255	IPv6 to IPv4 relay Anycast	C	256
192.168.0.0 - 192.168.255.255	Direcciones IP no homologadas	C	65,536
198.18.0.0 – 198.19.255.255	Pruebas para dispositivos de red	C	131,072
224.0.0.0 - 239.255.255.255	Multicast	D	268,435,456
240.0.0.0 - 255.255.255.255	Reservado	E	268,435,456

Tabla 1.2. *Direcciones IPv4 reservadas.*

Las direcciones IP para Zeroconf se utilizan para realizar una rápida configuración de una red sin realizar procedimientos elaborados. Entre las tareas que realiza Zeroconf se encuentran seleccionar una dirección IP para los elementos de red, descubrir qué nodo se identifica por algún nombre en particular y descubrir dónde se encuentran servicios, como el de impresión. El fin de utilizar direcciones *IPv6 to IPv4 relay Anycast,* también conocido como *6to4* es lograr la transmisión de paquetes IPv6 a través de una red IPv4, y para ello se realiza una encapsulación de dicha información en paquetes con el formato IPv4. Algo que es importante notar de la tabla anterior es que para las clases de redes A, B y C se cuenta con rangos de direcciones IPs privadas que tienen como fin asignarse a nodos dentro de una subred, de tal forma que varias redes privadas pueden disponer de dichas direcciones y en consecuencia repetirse de una red a otra; en este caso los nodos se comunican con el exterior de la subred por medio de una dirección IP que no es privada, es decir, una dirección IP homologada.

Inicialmente se tenía la idea de que las redes clase A correspondieran con las redes WAN, las redes clase B con las redes MAN y las redes clase C con las redes LAN, sin embargo, con esta clasificación no se satisfacieron los requerimientos de organización debido a la cantidad de hosts que cada una podía albergar, y hubo que definir nuevos métodos de planeación para adecuar las direcciones IPs a la demanda real. Una de estas técnicas es conocida como *subneting*, la cual se explica en el anexo III.

Capítulo 2
Protocolos soportados por el analizador

2.1 Protocolos TCP/IP

TCP e IP son las siglas en inglés de Protocolo de Control de Transmisión y Protocolo de Internet, respectivamente; pero al mencionar protocolos TCP/IP se hace referencia a un conjunto de protocolos que permiten una comunicación entre nodos de red.

La arquitectura de red de TCP/IP se compone de 4 capas; y de la misma forma que en el modelo OSI cada nivel añade a los datos que recibe información de control (encabezado o *header*) para comunicarse con su capa correspondiente pero en el host final. Un esquema de esta arquitectura se muestra en la figura 2.1.

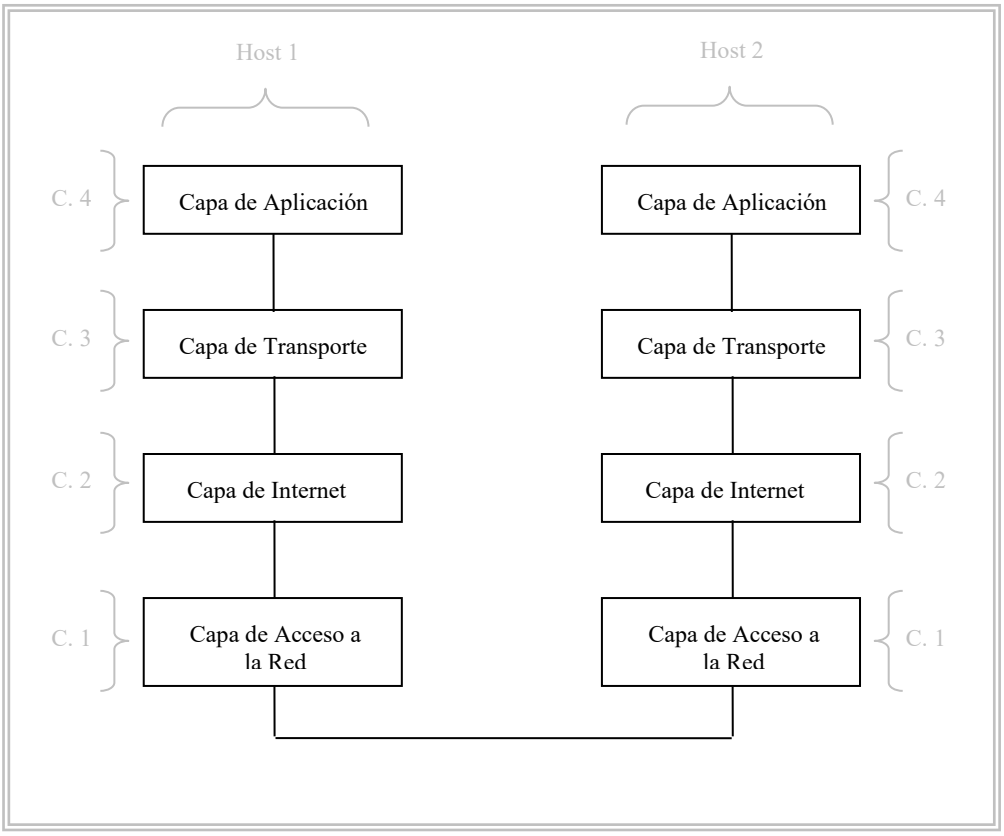

***Figura 2.1**. Esquema del modelo TCP/IP.*

La capa 1 establece como transmitir los datagramas IP a través de la red y por ello abarca protocolos de acceso a la red que van cambiando y apareciendo conforme se generan nuevas tecnologías. Si se compara con las capas del modelo OSI, realiza las tareas de los tres primeros niveles, es decir, de las capas Física, de Enlace y de Red. En la capa de Internet se ejecutan 2 protocolos, el IP y el ICMP (*Internet Control Message Protocol*). Algunas de las tareas que se realizan en este nivel son:

- Estructuración del datagrama IP y definición del esquema de direccionamiento.
- Ruteo de datagramas.
- Fragmentación de datagramas y su ensamble.
- Generación y envío de mensajes de control de flujo en la red de los datagramas.

La siguiente capa es la de Transporte Host a Host, o simplemente de Transporte, y en ella se presentan los protocolos TCP y UDP (User Datagram Protocol), el primero de ellos orientado a conexión. El uso de alguno de ellos depende del tipo de aplicación. En la capa de Aplicación se incluyen los procesos que manejan los datos a enviar o datos transmitidos; algunas de estas aplicaciones son: Telnet, FTP, SMTP, y HTTP, o servicios como DNS y NFS.

2.1.1 Protocolo Internet, IP

El Internet Protocol (IP) se encarga de realizar las tareas de comunicación de datos en forma de datagramas en la Internet. Sus especificaciones técnicas se presentan en el *RFC791*. Las funciones de direccionamiento y fragmentación del Protocolo Internet se llevan a cabo a través de módulos de programación en cada nodo de red ya sea un host, un switch, un gateway, etc. La implementación de dichos módulos debe cumplir características descritas en el estándar del IP para satisfacer los requerimientos de la comunicación de Internet, pero la forma de llevarlos a cabo varia de un equipo a otro ya sea por la configuración empleada por el administrador de dicho dispositivo o por el equipo de hardware mismo.

Cada datagrama se maneja de forma independiente a los demás sin importar si forma parte de una serie de datagramas que en conjunto conforman uno fragmentado; y cada uno contiene información sobre cómo procesarlo en el módulo de Internet al cual llega. Dicha información se puede clasificar en 4 mecanismos:

1. Tipo de servicio: De forma general, estos parámetros indican características de transmisión del datagrama y rutas a seguir en cuanto a redes o nodos.
2. Tiempo de vida: El tiempo durante el cual un datagrama será válido en la red por la que transita, este valor se expresa generalmente en segundos, pero también se ve disminuido en una unidad en cada nodo por el que es procesado a pesar de no haber transcurrido un segundo en esa actividad.
3. Opciones. Éstas se utilizan en casos especiales en que es necesario realizar alguna actividad extra durante la transferencia del datagrama, algunas de ellas son marcas de tiempo, seguimiento de rutas específicas, etc.
4. Suma de control. De cada datagrama se realiza una operación de control para verificar que los datos recibidos son los correctos y no sufrieron modificaciones durante su envío.

Esta suma de verificación es lo más aproximado a un mecanismo de control ya que el Protocolo Internet no lleva a cabo otro mecanismo de integridad de datos ni de control de flujo.

El datagrama IP es una serie de bits agrupados en palabras con un tamaño específico de acuerdo a la información que deben contener. Se forma mediante un encabezado que contiene información de control del datagrama. La figura 2.2 representa al encabezado de un datagrama IP.

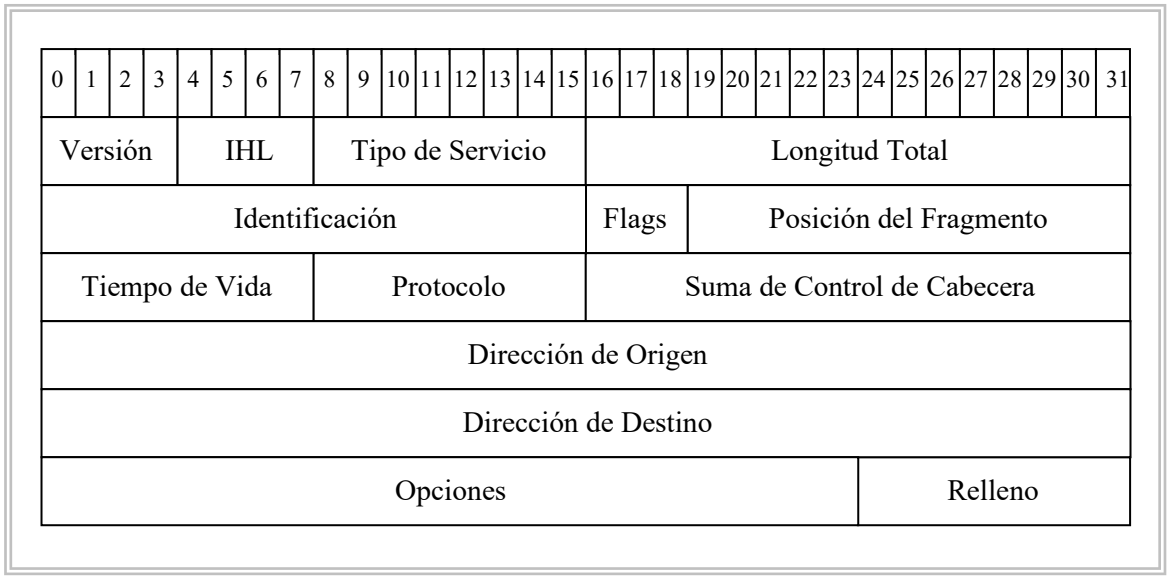

Figura 2.2. *Formato de Cabecera de un Datagrama Internet*

El campo *Versión* indica la versión del Protocolo Internet que normatiza su estructura; *IHL* es la abreviatura de Internet Header Length (Longitud de la Cabecera Internet) y es el campo que indica el tamaño del encabezado en palabras de 32 bits. El campo que corresponde al *Tipo de servicio* se utiliza en redes que permiten establecer un grado de urgencia para la entrega de datagramas; de igual forma permite informar sobre un trato especial en cuanto a enrutamiento. El campo señalado como de *Longitud total* se usa para indicar el tamaño total del datagrama en palabras de 8 bits, incluyendo los datos de información.

Los 16 bits del campo *Identificación* se utilizan para asignar a cada datagrama un valor único para realizar el re ensamble de los fragmentos en caso de ser necesario. Existen tres bits que se conocen como campo de *Flags* o banderas, por su traducción al español, el bit más significativo siempre es cero porque está reservado, el segundo bit más significativo se conoce como bandera *Don't Fragment* e indica que el datagrama no debe ser fragmentado bajo ninguna circunstancia si está ajustado a un valor de 1; la tercera bandera es conocida como *More Fragment* y si tiene un valor de 1 significa que todavía hay más fragmentos con un identificador superior al de dicho datagrama y que junto con él forman parte de un datagrama mayor que fue dividido. El campo *Posición del Fragmento* se ocupa al momento de unir los datagramas para formar el que fue dividido ya que indica a que parte del original pertenece.

El *Tiempo de vida* se refiere al tiempo durante el cual un datagrama será válido en la red como se explicó anteriormente, mientras que el de *Protocolo* especifica el protocolo que manejará el datagrama en un nivel posterior de la arquitectura de red. El campo de *Suma de control de* cabecera, mejor conocido como *Checksum* se calcula como el complemento a uno de 16 bits de la suma de los complementos a uno de todas las palabras de 16 bits de la cabecera. En el cálculo de la suma de control, el propio campo suma de control se considera formado por ceros. Los campos *Dirección de origen* y *Dirección destino* identifican a los

host finales que mantienen la comunicación, cada uno se compone de 32 bits. El campo *Opciones* se utiliza para indicar alguna operación extra referente a los datos ya sea para llevar a cabo un registro o para la transmisión del mensaje. El campo denominado *Relleno* es utilizado para darle al encabezado del datagrama un formato adecuado para su envío por la red, ya que rellena con ceros los bits necesarios para completar un tamaño múltiplo de 32 bits. Una lista de las opciones IP y su correspondiente *RFC* se indica en el documento denominado *ip-parameters* publicado por la *IANA* (Internet Assigned Numbers Authority) y que se presenta como el anexo VI en este trabajo.

Existen dos casos para el formato de una opción IP, en el primero de ellos se tiene un solo octeto para el tipo de opción, y en el segundo existen los campos tipo de opción, la longitud de la opción y el correspondiente a los datos que lleva.

El octeto Tipo de opción se compone de 3 campos: Un bit para el indicador de copiado que señala si la opción se debe copiar en todos los fragmentos en los que se divide el paquete (valor de 1); 2 bits para la clase de opción; y 5 bits para el número de opción.

Los valores posibles para la clase de opción son:

0 = control
1 = reservado para uso futuro
2 = depuración y medida
3 = reservado para uso futuro

El direccionamiento de los datagramas en el IP se realiza a través de direcciones de red asignadas a los nodos y a una correspondencia con direcciones de red local, la cual reside en la interfaz de red. El formato de estas direcciones se revisó en el capítulo anterior en el apartado *1.3.1 Direcciones de red IPv4*.

2.1.1.1 Fragmentación

Consiste en dividir un datagrama en varios de tal forma que puedan ser transmitidos, el límite de tamaño de cada datagrama está dado por el MTU (Maximum Transfer Unit) de la red. No todo datagrama puede ser fragmentado aunque sea mayor que el MTU especificado y esto se debe a que en las opciones del encabezado lleva indicada la bandera *Don´t Fragment*; en este caso será descartado si sobrepasa el tamaño máximo de transferencia.

Los métodos para fragmentar un datagrama son variados, pero cada uno de ellos debe de realizar procedimientos básicos para conservar las características funcionales de éstos, como por ejemplo copiar la información del encabezado del datagrama original en los encabezados de cada fragmento; dividir el campo de datos original necesariamente en múltiplos de 8 octetos para cada datagrama nuevo, excepto para el último; ajustar el valor del campo identificador de cada fragmento, etc. El re ensamble de los fragmentos se realiza tomando en cuenta el campo *Identificador*, el campo de *Dirección de origen*, el campo de *Dirección destino* y el campo de *Protocolo* del datagrama IP. Cada módulo de internet

deber ser capaz de recibir datagramas de entre 68 octetos como mínimo (60 octetos de cabecera y 8 octetos de datos), y 576 octetos como máximo. Las palabras que se modifican en los datagramas al fragmentarse son:

- El campo de *Opciones* en caso de presentarse, ya que no es necesario adjuntar todas las opciones en todos los encabezados de los fragmentos resultantes.
- La bandera *More Fragment*, pues cuando no se ha fragmentado el datagrama tiene un valor de cero, pero al dividirse esto cambia a un valor de 1 en cada parte hasta que se genera el último de ellos.
- El campo, *Posición del fragmento* (*Fragment Offset*).
- El campo que indica la longitud de la cabecera.
- El campo que indica la longitud total del datagrama (del fragmento).
- El campo Checksum, ya que se tienen que tomar en cuenta las variaciones en el encabezado y las opciones definidas para ese fragmento.

2.1.2 Protocolo de Control de Transmisión, TCP

TCP es un protocolo orientado a conexión a diferencia del IP, ya que proporciona mecanismos para llevar a cabo una comunicación fiable entre los procesos de dos hosts. Este protocolo se encuentra sobre la capa de red y por eso ocupa los servicios que proporciona un protocolo como el IP; también se comunica con protocolos de nivel superior y/o aplicaciones, a los que para fines prácticos se les conoce como *usuario local*. La transferencia de datos se realiza mediante su empaquetamiento en segmentos que son manejados a través del sistema de Internet.

TCP lleva a cabo una *Transferencia básica de datos* pues cuenta con mecanismos para decidir cuando enviar datos y cuando no hacerlo, así mismo cuenta con un método para asegurarse que todos los segmentos almacenados en un buffer sean enviados de forma inmediata. A través de números de secuencia asignados a cada segmento identifica los datos repetidos, los ordena, recupera fragmentos erróneos o solicita nuevamente datos que han expirado el tiempo máximo de llegada. También regula el flujo de datos mediante la especificación de un rango de identificadores de segmentos que se espera recibir, y lleva a cabo multiplexamiento en cuanto a conexiones con varios host, debido a que identifica cada conexión a través de una pareja de *conectores* o *sockets*. A la combinación de información del estado para cada flujo de datos entre dos hosts, con la pareja de sockets, los números de secuencia y el tamaño de la ventana de transmisión recibe el nombre de *Conexión*. Así cuando dos procesos quieran comunicarse primero deben establecer una conexión. Para establecer una conexión, el proceso envía datos al modulo TCP del host donde reside, el cual los empaqueta en segmentos y a su vez los pasa al módulo IP, o uno que realice tareas similares, para empaquetar los segmentos TCP en datagramas que puedan ser transmitidos a través de la red; para esto último se lleva a cabo otro empaquetamiento del datagrama a un formato de la red local. Durante todo el trayecto el paquete puede ser fragmentado, o desenvuelto a distintos niveles por nodos intermedios como gateways para conocer una ruta específica a seguir o conocer otros datos necesarios para la transferencia de los datos. En el host final, después de haber ensamblado los fragmentos en caso de ser necesario, se

desenvuelve el paquete hasta obtener el segmento TCP original para mandarlo al módulo TCP del nodo final que lo procesará y enviará al proceso correspondiente.

De forma similar a como se utilizan direcciones para identificar cada nodo en la red, los protocolos de transporte como el TCP y el UDP usan identificadores conocidos como números de puerto para cada aplicación con la que se comunican y que se ejecuta en el nodo donde reside el módulo que ejecuta dicho protocolo. De acuerdo al tipo protocolo que se ocupe en la capa de transporte los números de puerto pueden cambiar aunque se trate de la misma aplicación; sin embargo a aplicaciones de red muy utilizadas, conocidas como *well-known services* (*servicios bien conocidos*), se les ha asignado un número de puerto fijo y que no varia entre protocolos de transporte (a menos de que la aplicación en cuestión permita configurarse para trabajar en otro número de puerto). Los números de protocolo que van desde el 1 hasta el 255 son los reservados para los *well-known services* y por lo tanto se conocen como *well-known ports* (*puertos bien conocidos*), los números de puerto que van del 256 al 1024 son destinados para servicios específicos de UNIX; los puertos entre el número 1025 al 65535 no tienen alguna aplicación relacionada y se conocen como *dynamically allocated port* (*puertos dinámicamente asignados*). Este último tipo de puertos se utiliza por el módulo TCP para atender varias peticiones a través de un servicio de red; lo que se consigue con la siguiente metodología: cuando un host requiere comunicarse con otro a través una aplicación de red utiliza un puerto no asociado a ninguna aplicación (*dynamically allocated port*) como puerto del host origen y como puerto destino ocupa el *puerto bien conocido* (en caso de que lo tenga) del servicio de red con el que requiere comunicarse y en la petición de conexión se envía dicha información; por su parte el host destino ocupa como su puerto origen al *well-known port* de la aplicación de red y como puerto destino al *dynamically allocated port* que eligió el otro host. A la combinación de la dirección IP del host con el número de puerto origen se le conoce como *Socket* y la combinación del sockets del host origen con el socket del host destino identifica una conexión que es única dentro de la red.

El formato de la cabecera de un segmento TCP se muestra en la figura 2.3.

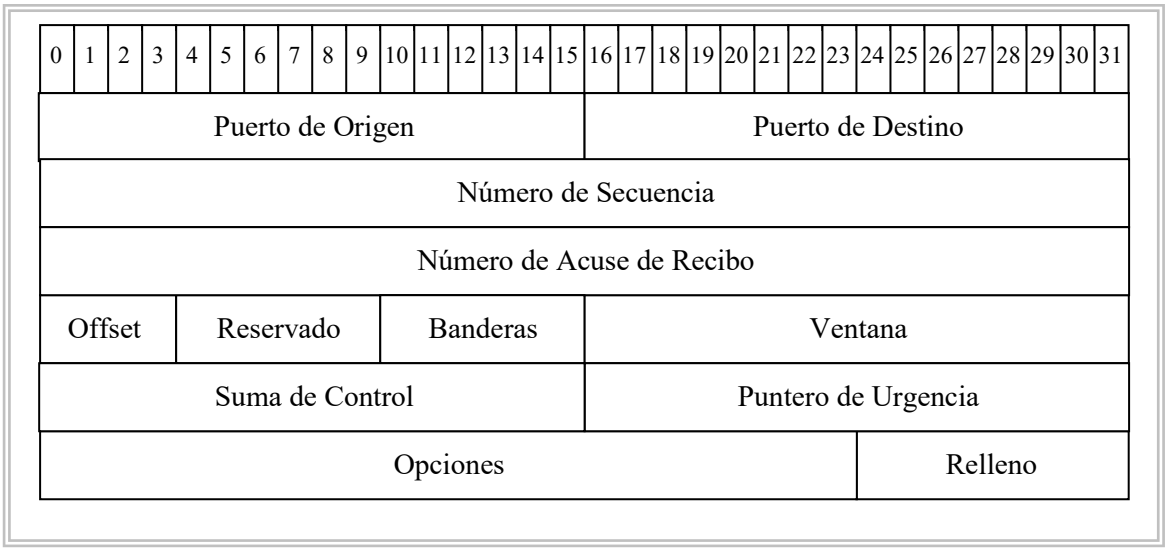

Figura 2.3. *Formato de cabecera de un segmento TCP.*

El encabezado de un datagrama se compone de 12 grupos de datos. En la primera palabra de 32 bits se indican el *Puerto de Origen* y el *Puerto Destino* de la aplicación de red, cada campo con una longitud de 16 bits; la segunda palabra se ocupa para indicar el Número *de Secuencia* del primer octeto de datos que lleva el segmento y que sirve como identificador del segmento. El *Número de Acuse de Recibo* en un campo de 32 bits e indica el número de secuencia del siguiente segmento que se espera recibir. El campo *Offset* (4 bits) funge como un apuntador a la palabra de 32 bits donde comienzan los datos. El campo denominado *Reservado* (6 bits) no tiene un uso definido, generalmente se le da el valor de 0. El campo identificado como *Banderas* se compone de 6 bits, cada uno asignado a una bandera que dependiendo del valor que tenga, 1 ó 0, indican que algún campo del segmento debe tomarse en cuenta o no, o que debe realizarse alguna acción de control, las cuales se presentan en el siguiente orden:

- Bandera URG (Urgent). Refiere al campo denominado Puntero de Urgencia.
- Bandera ACK (Acknowledge). Indica si se debe considerar el valor del campo del Número de Acuse de Recibo.
- Bandera PSH (Push). Su valor indica si se debe enviar inmediatamente los datos almacenados en un buffer.
- Bandera RST (Reset). Indicación para reiniciar una conexión.
- Bandera Syn (Synchronize). Este bit indica si el segmento tiene una función de sincronización de la conexión.
- Bandera Fin. Indica que ya no hay más datos que enviar.

El campo *Ventana* (16 bits) es un indicador de la cantidad de octetos, en cuanto al campo de datos, que se está dispuesto a recibir a partir de que envía el segmento; esto es con el fin de regular el flujo de información y ser capaz de procesar un límite máximo de información. El campo *Suma de Control* se utiliza para comprobar la integridad del segmento en cuanto a datos válidos y se calcula como el complemento a 1 de 16 bits de la suma de los complementos a 1 de todas las palabras de 16 bits del encabezado y texto; si un segmento tiene una cantidad impar de octetos de encabezado y texto, el último octeto se rellena con ceros a la derecha para formar una palabra de 16 bits. En esta operación los bits del campo *Suma de Control* se consideran ajustados a ceros y también se considera una pseudocabecera de 12 octetos que contiene la dirección de origen, la dirección de destino, el protocolo y la longitud del segmento TCP en octetos para verificar la validez de los campos de direccionamiento, además de un octeto de bits ajustados a un valor de cero. Aquí la longitud del segmento se calcula como la longitud de la cabecera más la longitud de los datos. Los bits del *Puntero de Urgencia* indican el número de secuencia del octeto al que seguirán los datos urgentes. El campo *Opciones* no tiene una longitud establecida pero siempre es un múltiplo de 8 bits; existen 2 formatos para el tipo de opción: uno con un sólo octeto que indica el tipo de opción, y el otro con 3 octetos como mínimo, uno para indicar el tipo de opción, el segundo para especificar la longitud de la opción y a partir del tercero los datos necesarios para procesar la opción. Con el campo de Relleno (bits ajustados a cero) se asegura que los datos del segmento comiencen en una posición múltiplo de 32 bits. La lista de opciones TCP es publicada por la IANA bajo el nombre de *tcp-parameters* y se incluye como el anexo VII al final de este trabajo.

El modulo TCP genera lo que se conoce como Bloque de Control de Transmisión, TCB por sus siglas en inglés, para almacenar valores referentes a la conexión que está manteniendo; entre dicha información se encuentran números de secuencia que identifican a los segmentos de los que todavía no se recibe acuse de recibo, número de secuencia para la siguiente transmisión, tamaño de la ventana actual, número de secuencia enviado para la actualización de la ventana y su número de acuse, número de secuencia inicial, número de secuencia del segmento que se espera recibir, tamaño de la ventana de recepción y número de secuencia de recepción inicial. La implementación de un módulo TCP debe contemplar varios estados para establecer una conexión y llevarla a cabo, dichas fases son las siguientes:

- LISTEN. El módulo TCP espera una solicitud de conexión.
- SYN-SENT. Se espera una solicitud de conexión (segmento con bandera SYN ajustada 1) concordante a una solicitud de conexión hecha previamente.
- SYN-RECEIVED. Estado en el que se ha recibido una solicitud de conexión aceptable y/o estado después de haber enviado un segmento con un acuse de recibo a una solicitud de conexión y en el que también se hace una solicitud de conexión.
- ESTABLISHED. Se presenta cuando los dos host han pasado de la parte de sincronización mediante segmentos con la bandera SYN activa y la conexión se ha establecido por completo.
- FIN-WAIT-1. Se espera una solicitud para finalizar la conexión, o la espera de un acuse de recibo de una solicitud para finalizar la conexión.
- FIN-WAIT-2. Espera una solicitud para finalizar la conexión.
- CLOSE-WAIT. Estado en el que el módulo TCP espera una solicitud para finalizar la conexión por parte del usuario local.
- CLOSING. Espera del acuse de recibo de una solicitud para terminar la conexión.
- LAST-ACK. Espera del acuse de recibo de una solicitud para finalizar conexión la cual incluye un acuse de recibo para una solicitud de finalización hecha por el host remoto.
- TIME-WAIT. Espera durante un tiempo suficiente para asegurar que el módulo TCP del otro host recibió el acuse de recibo de su solicitud de finalización de la conexión.
- CLOSED. Estado en el que no existe ninguna conexión.

Estos estados se clasifican en sincronizados y no sincronizados; éste último se refiere a los estados SYN-SENT, SYN-RECEIVED, mientras que los estados sincronizados son los 9 restantes.

Antes de empezar a transmitir datos es necesario que los hosts que participan en la comunicación se sincronicen, lo que sirve para fijar números de secuencia iniciales de datos para los segmentos TCP. La forma básica de sincronización se conoce como *Three-Way Handshake* o *Acuerdo en Tres Pasos*. En una primera etapa, un primer segmento enviado, el módulo TCP del host que desea entablar una comunicación envía un segmento con la bandera SYN activa y un número de secuencia indicado en la segunda palabra de 32 bits

(un número *X* por ejemplo); en la segunda fase, el host que recibió el segmento anteriormente dicho contesta con un segmento TCP con la bandera ACK activa y especifica en la tercera palabra de 32 bits el valor incrementado en 1 (*X+1*) del número de secuencia recibido (con lo cual se verifica que se recibió el segmento inicial), pero al mismo tiempo al tener también la bandera SYN activada señala cuál va a ser su número de secuencia inicial (un número *Y*, por ejemplo). En el tercer y último paso de sincronización el primer host responde con un segmento con la bandera ACK activa y un Número de Acuse de Recibo mayor en una unidad al Número de Secuencia Recibido (*Y+1*) y un Número de Secuencia propio correspondiente con el solicitado por el otro host, que en ejemplo sería de *X+1*. Los segmentos subsecuentes serían para intercambiar información de la aplicación de red. En la figura 2.4 se ilustra este proceso, incluyendo los estados que toma cada host al momento de enviar o recibir el mensaje.

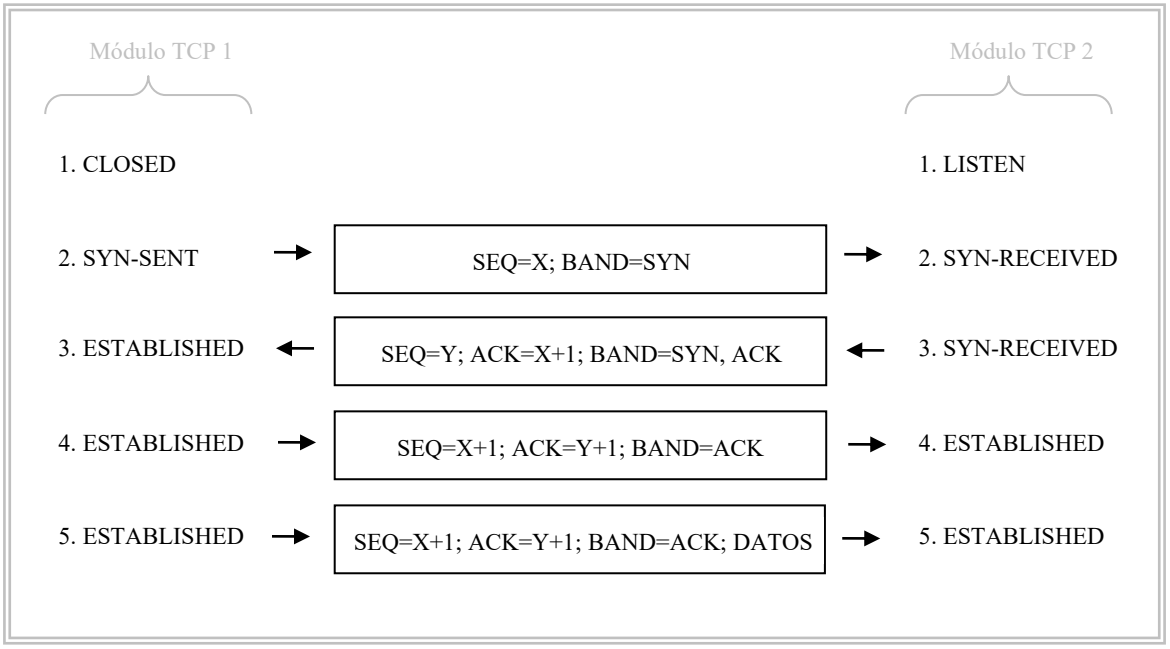

Figura 2.4. *Establecimiento de una conexión con el método Three-Way Handshake.*

Existen muchos otros escenarios para que dos hosts se sincronicen, los cuáles dependen del estado en que se encuentre cada uno en el momento en que quiera establecer la comunicación. El comportamiento a llevar a cabo se explica detalladamente en el RFC 793. El TCP es un protocolo de conmutación de paquetes y en consecuencia algunos segmentos pueden llegar retrasados e incluso pertenecer a una conexión anteriormente llevada a cabo; por esta razón contempla métodos para detectar situaciones anómalas y reiniciar la comunicación a través de la bandera reset y nuevos números de sincronización. Como método de prevención, antes de reiniciar una comunicación con un host se espera un intervalo de tiempo igual al máximo tiempo en que un segmento permanece en la red antes de caducar.

La finalización de una conexión también se realiza de forma controlada, y también existen varias situaciones en las que se puede presentar; a continuación se muestra la forma básica para terminar la conexión. El usuario local (aplicación en el host local) decide terminar la conexión y hace una petición de cierre al módulo de TCP local, que enviará un segmento con la bandera de FIN activada y a partir de ese momento cambiará a estado FIN-WAIT-1 y todos los segmentos que se encuentren en el buffer serán retransmitidos hasta que se reciba su acuse de recibido. Un módulo TCP que recibe un segmento de solicitud para finalización enviará el acuse correspondiente, pero sólo enviará un segmento con la bandera FIN activa hasta que su usuario local le indique una llamada *Close*. La figura 2.5 representa este proceso.

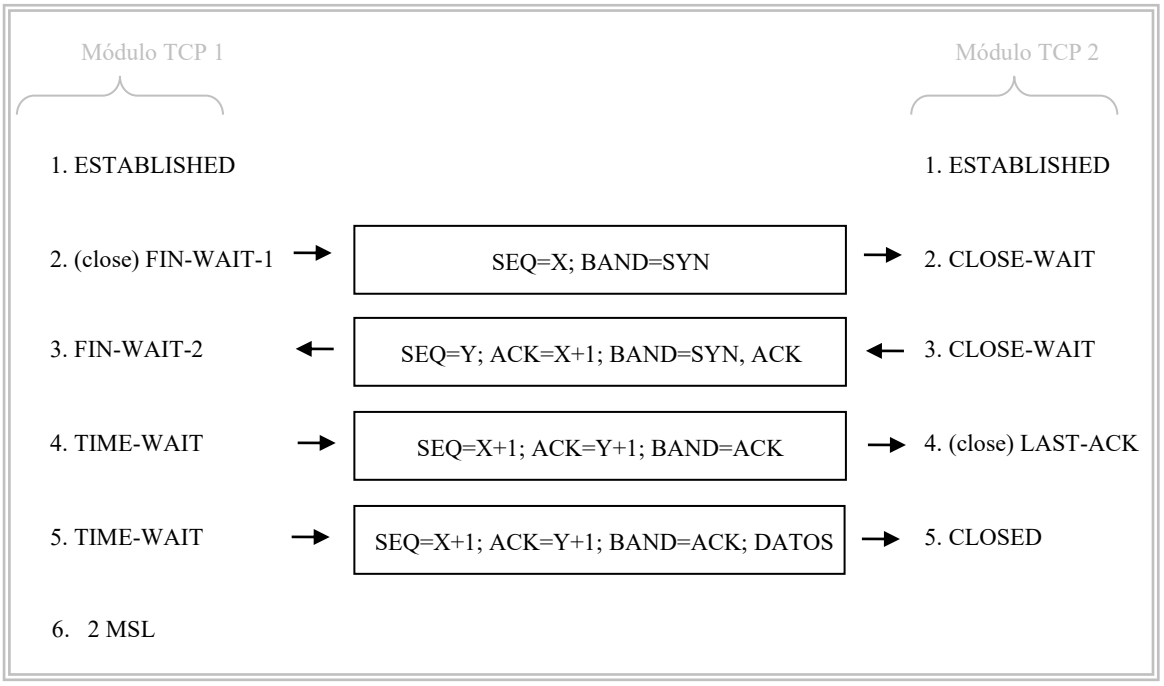

Figura 2.5. *Finalización de una conexión entre dos hosts.*

En el último paso del esquema anterior se espera un tiempo equivalente a 2 MSL con el fin de que los segmentos de esa conexión que pudieron quedar perdidos en la red sean descartados al terminar su tiempo de vida y en encarnaciones posteriores de la conexión no haya errores. Es necesario destacar que el usuario que hace la llamada CLOSE puede continuar recibiendo paquetes hasta que se entere que su contraparte ha cerrado también la conexión.

2.1.3 Protocolo de Mensajes de Control de Internet, ICMP

El *Internet Control Message Protocol*, se utiliza para informar sobre el estado de error de los datagramas de Internet a través de los circuitos de transmisión. Estos mensajes se envían al host origen del datagrama y son generados por los nodos que son intermediarios entre los host finales e incluso por el host destino. El manejo de mensajes

ICMP se debe implementar en el módulo IP de cada host y ocupan el mecanismo de este protocolo para ser comunicados; sin embargo no se envían mensajes ICMP referentes a mensajes ICMP.

Cuando un datagrama es fragmentado sólo se generan mensajes de Control de Internet para el fragmento 0. Cada mensaje ICMP empieza desde el primer octeto de datos del datagrama IP, y su formato generalmente es el siguiente:

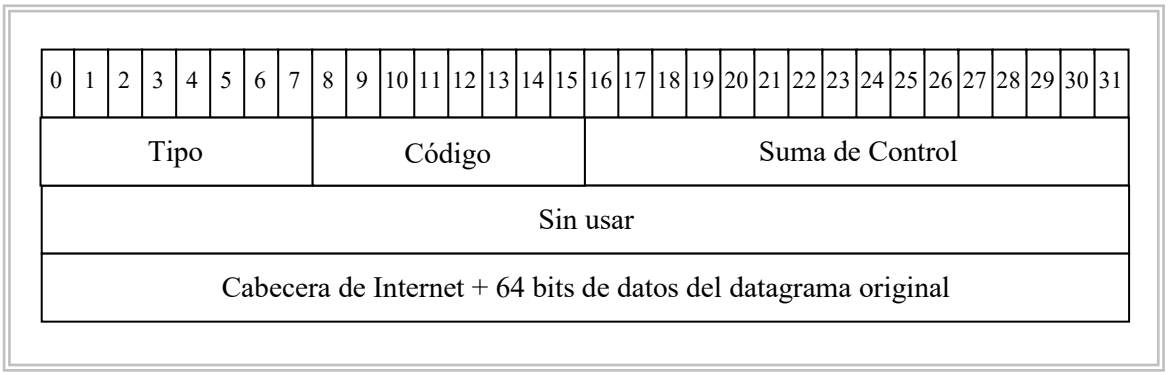

Figura 2.6. *Formato general de un mensaje ICMP.*

Algunos bits que forman el campo que no tiene uso en ocasiones se usan para especificar alguna información en particular, pero ello depende del tipo de mensaje. El campo de suma de control en cada tipo de mensaje se refiere al complemento a uno de 16 bits de la suma de los complementos a 1 de 16 bits de los campos que lo integran; esta suma puede ir cambiando conforme se reenvía el mensaje y tiene un valor inicial de 0. El campo que contiene la cabecera de Internet más los 64 primeros bits de datos del datagrama original se utiliza para asociar el mensaje al proceso correspondiente a dicha información. Estos 64 bits contienen los números de puerto que ocupa el protocolo de un nivel más arriba en la arquitectura de red. De forma general, para enviar mensajes ICMP algunos campos de la cabecera del datagrama IP deben tener un valor en específico, estos valores son los siguientes:

- Versión=4
- Tipo de Servicio =0
- Protocolo=1

2.1.3.1 Descripción de los tipos de mensajes ICMP

En el RFC 792 se describe el funcionamiento del ICMP y se presentan los tipos de mensaje que soportaba en un inicio, sin embargo estos se han incrementado con el correr del tiempo. Ésta sección describe el funcionamiento de los tipos de mensaje ICMP descritos en dicho RFC. La IANA a través del documento *icmp-parameters* publica los números de tipo de opción para paquetes ICMP; este documento se presenta en el apartado de ANEXOS con el punto número VIII.

2.1.3.1.1 Mensaje de Destino Inaccesible

El valor del campo de Tipo es 3 y el campo Código puede tomar los siguientes valores:

- 0 = Red inaccesible.
- 1 = Host inaccesible.
- 2 = Protocolo inaccesible.
- 3 = Puerto inaccesible.
- 4 = Se necesitaba fragmentación pero la bandera DF estaba activada.
- 5 = Fallo en la ruta de origen.

Esta clase de mensajes se produce cuando algún Gateway no puede encontrar la red a la que va dirigida el datagrama o cuando el host destino no puede direccionarlo al módulo del protocolo o puerto al que va dirigido debido que no están activos. También se pueden generan cuando es necesario fragmentar el datagrama IP pero lleva activa la bandera *Don't Fragment*. Los mensajes con código 0, 1, 4 y 5 son producidos por gateways y los que tienen códigos 2 y 3 por hosts.

2.1.3.1.2 Mensaje de Tiempo Superado

El campo Tipo tiene un valor de 11, y los valores para el campo Código son los siguientes:

- 0 = Tiempo de vida superado en tránsito.
- 1 = Tiempo de re ensamblaje de fragmentos superado.

Los mensajes con código 0 se generan por dispositivos intermedios entre el host origen y el host destino cuando el tiempo de vida del datagrama llega a un valor de 0; y con un código 1 cuando un host no puede re ensamblar el datagrama en un tiempo máximo establecido debido a que no cuenta con todas las partes. Si el fragmento 0 no está disponible no se genera el mensaje ICMP.

2.1.3.1.3 Mensaje de problema de parámetros

El campo tipo lleva un valor de 12 y el de código un valor de 0. De la segunda palabra de 32 bits del mensaje ICMP se ocupan los 8 primeros bits para indicar un Puntero y el resto de los bits de esa palabra se quedan sin usar. El puntero indica el octeto del datagrama original dónde se detecta el error. Al haber un error no se puede procesar el encabezado y el datagrama es desechado. Este mensaje se puede generar por un host o por un gateway.

2.1.3.1.4 Mensaje de Disminución del Tráfico desde el Origen

Tiene un valor de 4 en el campo de Tipo de mensaje y un valor de 0 en el de código. Cuando en un gateway se satura el espacio en el buffer de recepción de datagramas al

procesarlos puede empezar a descartar datagramas y enviar mensajes ICMP de esta clase al host que los envió; el host final también puede enviar este tipo de mensajes si de igual forma no puede procesarlos con la suficiente rapidez. Cuando se recibe un mensaje de disminución de tráfico el host debe disminuir el ritmo de generación de tráfico al destino especificado hasta que deje de recibirlos, y puede aumentarlo gradualmente hasta que vuelva a recibir este tipo de mensajes. En algunas ocasiones un host o un gateway puede enviar estos mensajes cuando está próximo a saturarse su buffer de entrada.

2.1.3.1.5 Mensaje de Redirección

El valor del campo Tipo es de 5, y los posibles valores de campo Código son:

- 0 = Redirigir datagramas debido a la red.
- 1 = Redirigir datagramas debido al host.
- 2 = Redirigir datagramas debido al tipo de Servicio y la red.
- 3 = Redirigir datagramas debido al tipo de Servicio y al host.

La segunda palabra de 32 bits funge como un indicador de la dirección de la pasarela o gateway al cual el host debe redirigir el tráfico. Este re direccionamiento se debe a que una pasarela que recibe un mensaje y lo re envía a otra, identifica que el host tiene acceso a dicho nodo y entonces mediante este tipo de mensajes indica que el host puede seguir ese camino más corto.

2.1.3.1.6 Mensaje de Echo o de Echo Reply

Se diferencia de los demás tipos de mensajes debido a que no tiene el campo de la Cabecera de Internet más los primeros 64 bits de datos del datagrama original. La segunda palabra de 32 bits se divide en dos campos de 16 bits cada uno; un campo Identificador y un campo Número de Secuencia. Ambos se utilizan para realizar una correspondencia entre los mensajes de solicitud y de respuesta. El campo Identificador puede utilizarse como un puerto para identificar una sesión y el Número de Secuencia se incrementaría con cada nueva petición de eco enviada. De esta forma al contestar con Echo Reply los valores (que serán los mismos que los de la solicitud) servirán para realizar la correspondencia con la petición.

Si el campo Tipo tiene un valor de 8 se trata de un mensaje de *Echo* y si tiene un valor de 0 es un mensaje de *respuesta de Echo*. El código siempre tiene un valor de 0.

2.1.3.1.7 Mensaje de Solicitud y de Respuesta de Marca de Tiempo

Los campos por los que se conforma estos mensajes son: Tipo, Código (siempre ajustado a un valor de 0), Suma de Control, Identificador (16 bits), Número de Secuencia (16 bits), Marca de Tiempo de Origen, Marca de Tiempo de Recepción, Marca de Tiempo

de Transmisión. Para cada marca de tiempo se destinan 32 bits; los demás campos tienen el tamaño acostumbrado para mensajes ICMP. El campo Identificador se ocupa para reconocer los mensajes correspondientes con sus respuestas; de igual forma se ocupa el Número de Secuencia. La marca de tiempo es la cantidad de milisegundos que han pasado desde la medianoche UT hasta el instante en que se manipula el mensaje. Si no se puede establecer dicho valor, el bit más significativo de la marca de tiempo se ajusta a uno. Cuando se hace una solicitud de información el campo Código lleva un valor de 13 y cuando es una respuesta lleva un valor de 14.

2.1.3.1.8 Mensaje de Solicitud de Información o de Respuesta de Información

Los campos que lo integran son los siguientes: Tipo, que puede tomar el valor de 15 para solicitar información y el valor de 16 para indicar que se trata de una respuesta; el campo Código que lleva un valor de 0, Suma de Control, un campo Identificador para identificar respuestas con su solicitud correspondiente, y un campo de Número de Secuencia que se utiliza también con fines de correspondencia de solicitudes con respuestas. Este tipo de mensajes puede ser generado por un host o un Gateway, y se utiliza para identificar a la red en la que se encuentra el nodo que lo emite.

2.1.4 Protocolo de Datagramas de Usuario, UDP

El objetivo de este protocolo es permitir el envío de mensajes con un mínimo de reglas de transmisión entre programas de aplicación. No es un protocolo orientado a conexión. El formato del mensaje es el mostrado en la figura 2.7.

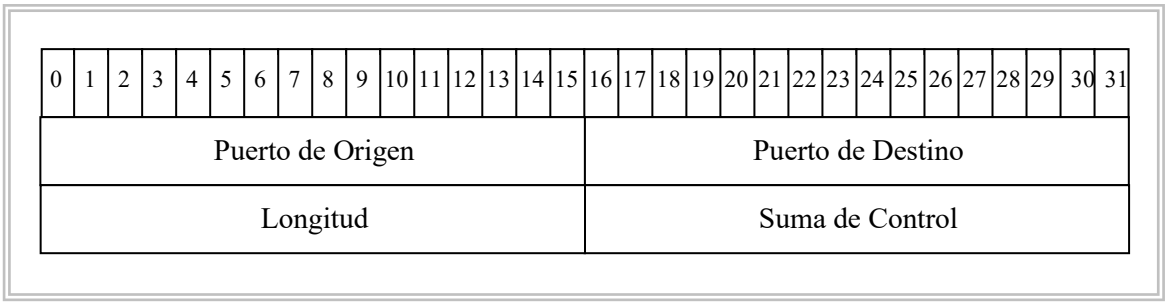

Figura 2.7. *Formato general de un mensaje UDP*

Si la cantidad de datos a enviar es pequeña el asegurarse con otros protocolos una transmisión confiable puede resultar mucho más trabajoso que retransmitir el conjunto entero de datos con UDP. Algunas aplicaciones implementan sus propios métodos para la entrega confiable de los datos y aquellas que requieren un modelo de comunicación de *solicitud-respuesta* cuando no se recibe una confirmación simplemente reenvían una solicitud y no requieren del servicio de un protocolo como el TCP. Algunas de las aplicaciones que lo utilizan son el TFTP (*Trivial File Transfer Protocol*) y las que realizan tareas de DNS (*Domain Name System*).

El campo *Puerto de Origen* es opcional y si no se necesita se le asigna un valor de 0, se asume que a ese puerto se debe dirigir la respuesta en caso de que no se especifique otro. El campo *Puerto de Destino* especifica el puerto que utiliza la aplicación remota con el que se desea comunicar la aplicación del host local. El campo *Longitud* indica la longitud total en octetos del datagrama, incluyendo la cabecera y los datos. La *Suma de Control* se calcula como el complemento a uno de 16 bits de la suma de los complementos a uno de las palabras que conforman el paquete y las que resultan de la combinación de una pseudo-cabecera que se integra por la dirección de origen y la dirección destino, cada una de 32 bits; un campo de 8 bits ajustados a cero, el protocolo (8 bits) y la longitud UDP (16 bits).

2.1.5 Protocolo de Resolución de Direcciones, ARP

El ARP (Address Resolution Protocol) establece el mecanismo para obtener la dirección MAC del host con una determinada dirección IP. La identificación de los hosts que se encuentran en redes Ethernet se realiza con base en direcciones MAC; pero cuando una aplicación de red necesita comunicarse con un proceso en otro nodo lo hace a través de direcciones propias de otro protocolo de comunicación, como el IP. Sin embargo, en las capas inferiores (capa de Enlace y Física) se debe encontrar una correspondencia de esas direcciones que manejan las aplicaciones con direcciones que sólo se reconocen en este nivel de la arquitectura.

Una trama ARP se compone por los siguientes campos: *Espacio de direcciones de hardware*, que especifica el tipo de hardware, por ejemplo Ethernet, o alguna red de paquetes de difusión; un campo para indicar el tipo de protocolo que ocupa la aplicación de red que se llama *Espacio de direcciones de protocolo*. Un campo para indicar la longitud en bytes de la dirección de hardware y uno para indicar la longitud en bytes de la dirección de protocolo; un campo *Código de operación* para saber si se trata de una solicitud de información o de una respuesta. Y 4 campos de longitud variable, *Dirección de hardware y Dirección de protocolo del host fuente* y *Dirección de hardware y Dirección de protocolo del host destino.*

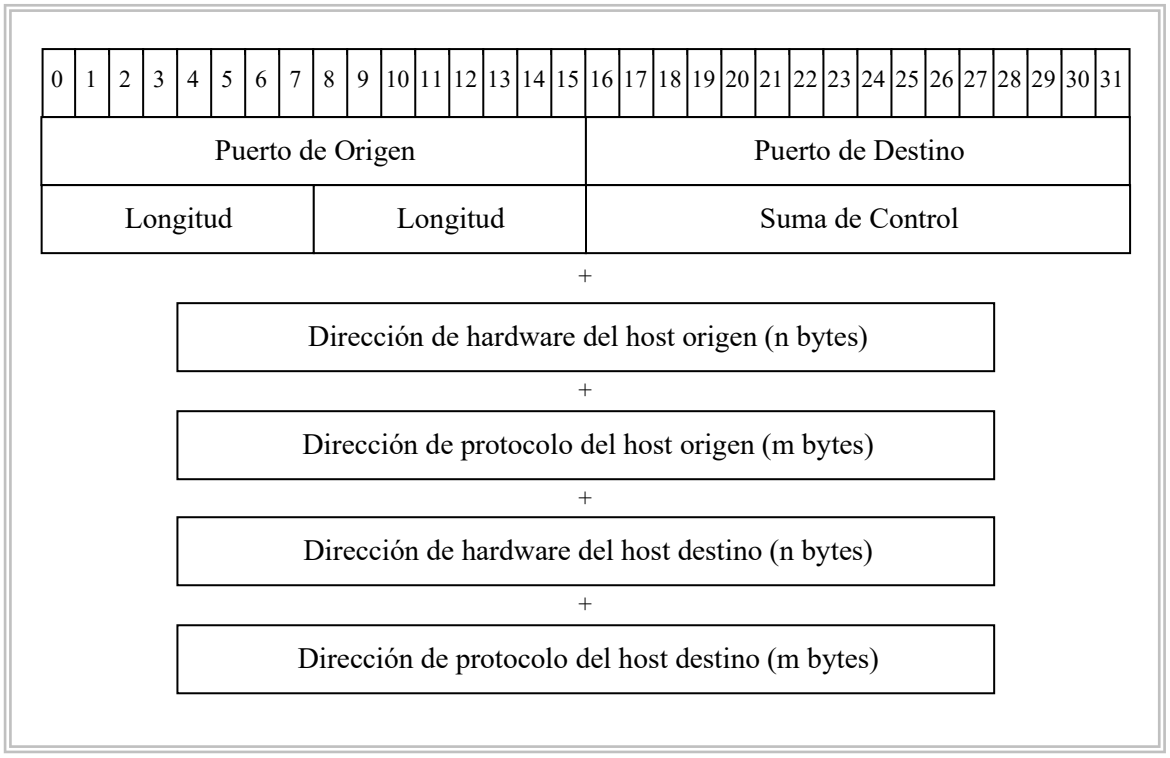

Figura 2.8. *Formato para una trama del ARP*

En la figura 2.8 las variables *n* y *m* representan el valor de los campos *Longitud de dirección de hardware* y *Longitud de dirección de protocolo*, respectivamente.

Una aplicación de red hace una solicitud para transmitir datos al controlador de hardware, pero sólo proporciona una dirección de protocolo de red y su tipo para identificar al destinatario; entonces el controlador de hardware consulta al módulo de resolución de direcciones para conocer la dirección MAC a la cual mandar los datos. En cada host se tiene una memoria caché ARP que para cada entrada tiene 3 registros: *tipo de protocolo*, *dirección de protocolo*, y *dirección MAC*, y que el módulo de resolución de direcciones consulta para responder al controlador de hardware. Si existe una entrada referente a dicha dirección de protocolo y tipo de protocolo, la dirección MAC se proporciona como respuesta y el controlador de hardware envía el paquete. En caso de que no exista dicha información en la caché ARP el datagrama proporcionado por el módulo de red es desechado, lo que se informa a la aplicación de red para que lo intente retransmitir en una posterior ocasión, y se genera un paquete Ethernet con un campo *Tipo* de *Resolución de dirección*; el campo de *Espacio de Direcciones* se establece como Ethernet (valor de 1); el campo de *Espacio de Direcciones de Protocolo* se ajusta al tipo de protocolo de la dirección de red proporcionada (valor de 2048 para IP); al campo de la *Longitud de la dirección* de hardware se le da un valor de 6, y al campo de la *Longitud de protocolo* un valor de acuerdo a la longitud de la dirección de red, que para el Protocolo Internet sería de 4. Como el paquete que se genera es para realizar una solicitud de la dirección MAC, el campo *Código de Operación* adquiere un valor de solicitud (valor de 1), y los campos de *Dirección de hardware de origen y Dirección de protocolo de origen* se ajustan a los valores del host. El campo correspondiente a la *Dirección de hardware del host* destino

queda sin especificarse debido a que esa es la información que se está solicitando. Al campo *Dirección de protocolo* del host destino se le da el valor de la dirección proporcionada por la aplicación de red o si es conveniente la dirección de broadcast para actualizar las direcciones MAC de todos los equipos.

El equipo que recibe la solicitud de dirección MAC realiza las siguientes operaciones:

1. Revisa si tiene el tipo de hardware indicado en el espacio de direcciones de la solicitud, Ethernet por ejemplo.
2. Revisa si es capaz de comprender el tipo de protocolo especificado en el campo de espacio de direcciones de protocolo.
3. Ajusta una bandera a falso.
4. Revisa si tiene en su caché ARP una entrada correspondiente al *tipo de protocolo* y a la *dirección de protocolo del host* solicitante y si es así actualiza el campo correspondiente a la dirección MAC y cambia el valor de la bandera mencionada anteriormente verdadero.
5. Revisa si tiene la dirección de protocolo indicada es la que tiene asignada, y si es así continua con los siguientes pasos.
6. Si no tiene en su caché ARP una entrada correspondiente al *tipo de protocolo* y a la *dirección de protocolo del host* solicitante (bandera con un valor falso), los agrega junto con la dirección MAC del solicitante.
7. Si el campo *Código de operación* indica que se trata de una solicitud intercambia los valores de direcciones de hardware y protocolo del emisor con los de destino, y lo configura como respuesta. Después envía el paquete a la nueva dirección de hardware destino.

La máquina que hizo la solicitud inicial recibe la respuesta, realiza la actualización en el ARP caché con los datos recibidos y desecha el paquete.

2.2 Formatos del frame Ethernet

A través del tiempo el frame Ethernet ha tenido modificaciones en cuanto a su formato, sin embargo las diferentes estructuras se aproximan a lo presentado en el capítulo uno en el apartado 1.1.3.2. Actualmente se ocupan 4 versiones (Ethernet II, Ethernet Novell RAW 802.3, Ethernet IEEE 802.3 y Ethernet IEEE 802.3 SNAP), sin embargo para la transmisión de cada una de ellas se contempla al inicio el campo *Preámbulo* de 64 bits y al final el campo de comprobación de redundancia cíclica de 32 bits. Este preámbulo comienza con una secuencia de 62 bits con valores alternados 1 y 0 (comenzando con 1), y termina con dos bits ajustados a un valor de 1. Cabe mencionar que el último byte del preámbulo se conoce como *SFD* (Start Frame Delimiter).

Un aspecto que marca una diferencia clara entre las versiones utilizadas es un campo de 2 bytes llamado *Tipo/Longitud* ya que si tiene un valor superior a la longitud máxima permitida para el frame el campo se debe manejar como *Ethertype*, es decir, el tipo de protocolo que debe manipular la información contenida en el campo de datos en el nodo

destino en una capa superior de la arquitectura de red. El valor máximo que puede tomar este campo para interpretarse como longitud es de 1514 (suma de la longitud de todos los campos exceptuando el preámbulo y el CRC) a pesar de que los valores destinados para especificar los protocolos comienzan en el 1536.

2.2.1 Ethernet II

Este formato de frame se compone por 4 campos: 6 bytes para especificar la Dirección Ethernet Destino, 6 bytes para indicar la Dirección Ethernet Origen, 2 bytes para el campo *Ethertype*, y un campo de datos con una longitud entre 46 y 1500 bytes (ver figura 2.9).

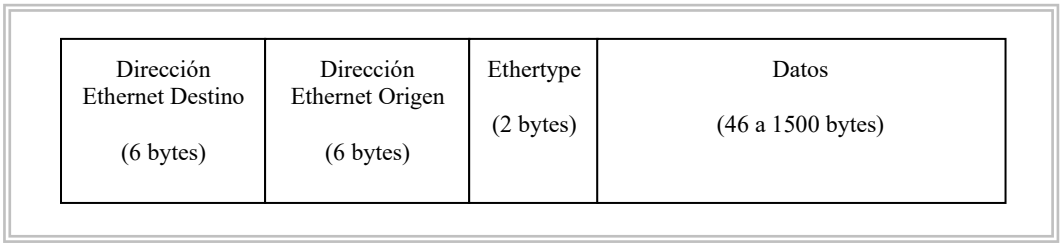

Figura 2.9. *Formato del frame Ethernet II.*

2.2.2 Ethernet Novell RAW 802.3

Este tipo de frame se utiliza para transportar *paquetes IPX*, inicia con un campo de Dirección Ethernet Destino y con un campo de Dirección Ethernet Origen, cada uno de 6 bytes, enseguida se presenta un campo de 2 bytes para especificar la longitud del frame. Posteriormente se encuentra un campo conocido como *IPX header* de 3 bytes pero con los dos primeros ajustados a un valor de 0xFF; y por último está el campo de datos con una longitud variable entre 43 y 1497 bytes. Lo anterior se muestra en la figura 2.10.

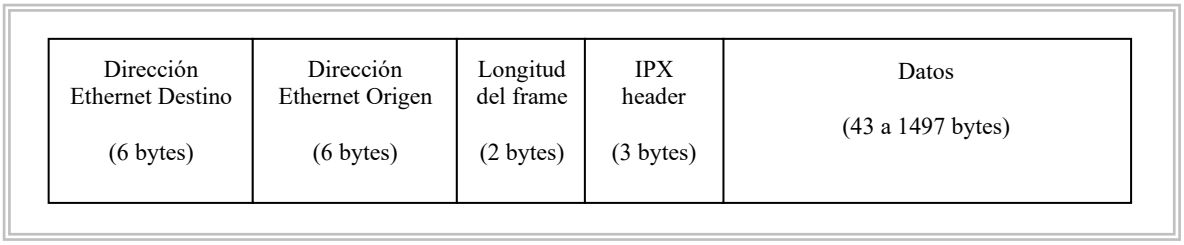

Figura 2.10. *Formato del frame Ethernet Novell RAW 802.3.*

2.2.3 Ethernet IEEE 802.3

El estándar IEEE 802.3 contempla dos campos de 6 bytes cada uno para indicar las Direcciones Ethernet Destino y Origen, respectivamente, después especifica un campo de 2 bytes para señalar la longitud del frame, y enseguida se establecen 3 bytes de control:

DSAP (Destination Service Access Point), *SSAP* (Source Service Access Point) y CONTROL, los cuales en conjunto se conocen como *encabezado 802.2*. Después continúa el campo de datos con una longitud variable entre 43 y 1497 bytes. Esta estructura se muestra en la figura 2.11.

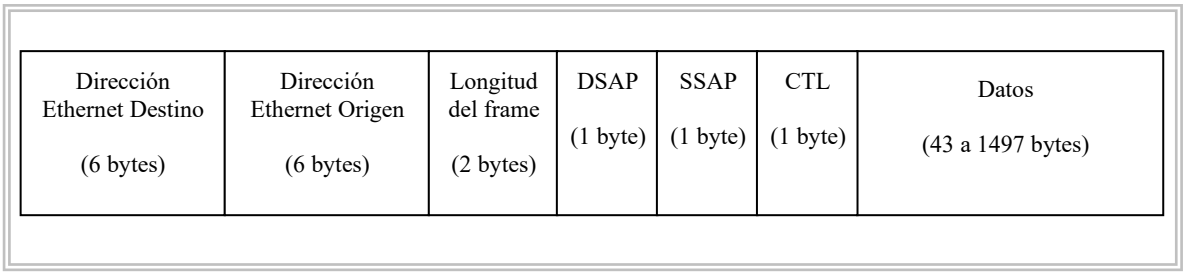

Figura 2.11. *Formato del frame Ethernet 802.3*

2.2.4 Ethernet IEEE 802.3 SNAP

SNAP son las siglas de Sub-Network Access Protocol. Es una extensión del estándar Ethernet 802.3 de la IEEE que se realizó con el fin de poder encapsular paquetes IP y ARP dentro del frame 802.3. El frame se integra por los siguientes campos: Dirección Ethernet Origen (6 bytes); Dirección Ethernet Destino (6 bytes); un campo de 2 bytes para especificar la longitud del frame; los campos DSAP y SSAP (ajustados a un valor de 0xAA); el campo CONTROL ajustado aun valor de 0x03; un campo de 3 bytes conocido como *Protocol ID* donde se indica el código del fabricante de la tarjeta de red la mayoría de las veces o se establece a un valor de cero; un campo Ethertype de 2 bytes; y por último el campo de datos que puede medir desde 38 a 1492 bytes. La figura 2.12 ejemplifica lo anterior.

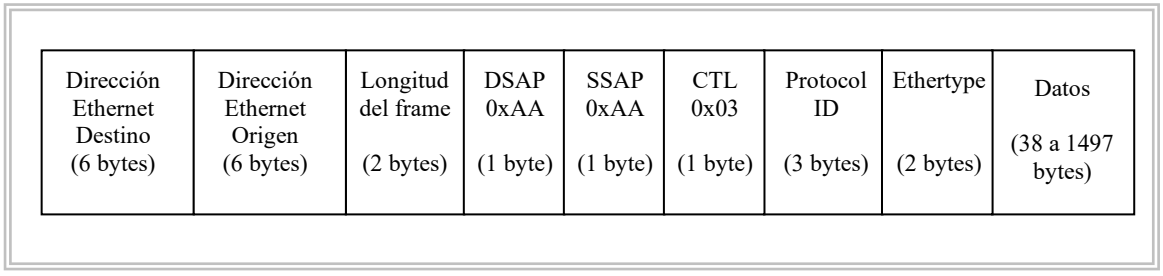

Figura 2.12. *Formato del frame Ethernet 802.3 SNAP*

La importancia de distinguir los diferentes tipos de frame para el analizador de protocolos radica en poder reconocer cuando se presenta cada uno de ellos y saber interpretarlo de forma adecuada. Sobre todo porque hay que considerar que solamente las versiones Ethernet II e IEEE 802.3 SNAP están diseñadas para el transporte de los protocolos de la familia TCP/IP. Algo similar sucede con los paquetes IP, TCP, ICMP, UDP y ARP, ya que para capturarlos e interpretarlos con esta herramienta es necesario conocer las posibles estructuras que pueden tomar cada uno de ellos.

Para estos protocolos a nivel de red de la arquitectura TCP/IP sólo se presentan y explican las opciones expresadas en los primeros RFC que los definen, sin embargo, lo destacable es que las demás opciones surgidas a lo largo del tiempo se ajustan al formato descrito en esos documentos y, por supuesto, en este capítulo.

Para conocer la lista de los RFC que explican las opciones completas para estos paquetes se puede acudir a los anexos VI, VII y VIII.

Capítulo 3
Análisis y diseño

3.1 Analizadores de protocolos

Inicialmente los analizadores de protocolos eran un tipo de hardware y con el tiempo fueron desarrollándose en forma de software, como son la mayoría actualmente; y básicamente son herramientas que permiten obtener datos ordenados sobre el tráfico que presenta la red sobre la que se ejecutan. Para esto realizan la captura de tramas de datos de la red (Ethernet, Token Ring, entre otros) y las interpretan para presentar la información en tiempo real o en un momento posterior a su registro.

La utilidad de tener datos sobre el tráfico radica en poder realizar un diagnóstico sobre el estado de la red y poder identificar ciertos *puntos débiles* en su funcionamiento (como la generación excesiva de tramas) o simplemente recabar información sobre lo que se transporta, sobre el uso de determinados protocolos, o el estado de los equipos que los conforman. Los analizadores de protocolos también sirven para comprender el funcionamiento de los protocolos que soporten y su interacción o el comportamiento de la red ante determinadas situaciones.

Para capturar las tramas se necesita configurar el dispositivo de red del host donde está instalado el analizador de protocolos para que actúe en *modo promiscuo*, con lo cual realizará la escucha de todas las tramas que le sea posible que circulen por la red aún y cuando no estén dirigidas a dicho nodo.

3.1.1 Características generales

La elección de un analizador de protocolos para realizar tareas de supervisión depende de las características de la red (tecnología que ocupa, tamaño y topología, por mencionar algunas), de la capacidad de procesamiento del host donde se va a instalar y el sistema operativo con el que cuenta. Un factor muy importante para dicha elección es el grupo de protocolos que es capaz de interpretar. Generalmente, mientras más características tenga un software de este tipo mayor es su costo, en lo que respecta a los que no son software libre; la mayoría de ellos son muy buenos no sólo por sus funciones sino también por sus atributos añadidos como atención el línea, actualizaciones disponibles y ambientes de operación. Por otra parte también existen muchos que se distribuyen bajo licencia *GPL* (General Public License) y que cuentan con un gran soporte de protocolos y funciones extras debido a la participación de programadores de diversas partes del mundo, además de permitir la alteración de su código y configuración más personalizada.

De forma general cada analizador de protocolos es capaz de:

- Capturar y clasificar tramas de acuerdo al tipo de protocolo.
- Establecer filtros para la captura y para mostrar la información.
- Presentar estadísticas sobre el tráfico de la red.
- Mostrar datos en tiempo real o en momentos posteriores.
- Generar un archivo de tipo *log* con los datos recabados.

3.2 Planteamiento del problema

Con base en los objetivos de esta tesis se definen las siguientes características del analizador de protocolos a desarrollar.

En una red de datos la cantidad de protocolos que se pueden utilizar para transmitir información es muy grande y contemplarlos en un programa de este tipo es una tarea ardua, sin embargo para los fines de este trabajo se decidió que el analizador de protocolos se enfocaría a una muestra representativa de los protocolos no en cuanto a cantidad sino a la forma en como trabajan en la arquitectura de red más utilizada. A causa de esto se optó por los protocolos descritos en el ***capítulo 2: Protocolos soportados por el analizador.***

Para cada uno de los protocolos elegidos se debe recabar necesariamente la información de la estructura del paquete en cuestión y datos generales que describan características del tráfico de red. Para el frame Ethernet los datos generales a registrar son:

- Número de frame, relativo al flujo de red leído.
- Fecha y hora de recepción del frame.
- Tamaño del frame en bytes.
- Interpretación del campo Ethertype (0x0800=IP y 0x0806=ARP).
- Tipo de frame Ethernet.

- Dirección Ethernet origen y dirección Ethernet destino.
- Valor del campo Ethertype.
- Valor de los campos DSAP, SSAP y CTL.
- Valor del campo Protocol ID.

Cabe aclarar que algunos de estos datos se presentan sólo en determinado tipo de frame Ethernet.

3.2.1 Bitácoras de información

Para analizar el tráfico de red capturado es necesario traducir los datos mencionados recientemente a un lenguaje entendible por el administrador y almacenarlos en bitácoras de información. Se tienen contempladas varias formas para mostrar estos datos, cada una de ellas incluyendo los datos ya descritos de cada protocolo que contempla el analizador.

El primer formato y el más inmediato es a través de la salida estándar del sistema, es decir, el monitor. En este caso se agrupa la información en bloques presentando primero los datos generales del frame Ethernet, enseguida, si el campo Ethertype corresponde al Protocolo Internet se presenta un bloque de información de dicho paquete, después un área para presentar los datos del protocolo encapsulado dentro del IP (TCP, ICMP o UDP). Después, si se presentan, se destina un espacio para las opciones de cada paquete u datos extra según sea el protocolo. Si el paquete transportado es TCP se imprimen byte por byte los datos que contiene y en caso de que alguno de ellos no sea un carácter imprimible en su lugar se muestra un punto. En el caso de los paquetes UDP también se imprimen los datos extra en formato carácter y su correspondiente representación hexadecimal. Por otro lado, si el campo Ethertype indica que el frame transporta un paqueta ARP, se imprimen los datos de cada campo de este paquete. Al final de presentar los datos de cada paquete se muestra una línea divisoria que lo señala. La funcionalidad de presentar los datos de esta forma es poder ver el flujo de datos en la red en tiempo real y poder hacer un diagnóstico rápido de cuestiones generales como por ejemplo la cantidad de paquetes transmitidos o, combinado con herramientas del mismo sistema operativo, hacer algo un poco más elaborado. Aparte de esto, también es posible ver posteriormente dicha salida de texto en pantalla a través de un archivo intermedio que se crea al instante de realizar la captura y que almacena el flujo de datos tal cual se recibe.

De las formas para mostrar la información al usuario las anteriores son las más comprensibles ya que se indica brevemente a qué se refiere cada dato presentado. Desde este punto de vista es bueno tener el registro en memoria secundaria, sin embargo no es necesario programar funciones para escribir todo este formato en un archivo de texto pues esta función se puede realizar a través de un redireccionamiento de la salida estándar a un archivo de ese tipo. También se optó por este comportamiento ya que almacenar esta información a través de un archivo exige tener espacio suficiente en disco duro teniendo en cuenta que no sólo se almacenaría el flujo de bytes leído sino también los bytes que le dan estructura al formato.

El otro formato para presentar la información del flujo de bytes es a través de un archivo XML. Aprovechando la estructura de etiquetas para formar este tipo de archivos se puede almacenar los datos de cada frame Ethernet de forma ordenada, con lo cual se refleja muy bien la estructura de la información recibida. Otra de las ventajas de tener la información en este formato es que se puede visualizar a través de un sencillo editor de texto o con un explorador web que añade la función de retraer o expandir los nodos del documento. Por otra parte, al tener todos los datos bajo un formato ordenado que identifique el tipo de dato a través de etiquetas permite realizar una lectura selectiva de la información y realizar tareas de análisis. La estructura de los documentos XML se rige por archivos conocidos como *Document Type Definition* (DTD), que son una especie de plantilla donde con *expresiones regulares* se indica el orden y número de ocurrencias de las etiquetas válidas. El DTD diseñado para las bitácoras XML se encuentra en el ANEXO IX de este trabajo.

Además de capturar el flujo de bytes, interpretarlos y mostrarlos se pretende presentar estadísticas que proporcionen aspectos básicos para análisis del tráfico de red. Los puntos a cubrir son:

- Presentar cada dirección Ethernet origen identificada en el flujo de bytes con su correspondiente dirección IP y cuantas veces se presentó esta dupla en el tráfico de red.
- Presentar cada dirección Ethernet destino identificada en el flujo de bytes con su correspondiente dirección IP y cuántas veces se presentó esta dupla en el tráfico de red.
- Indicar cuál es la dupla de direcciones origen (Ethernet-IP) que más se presenta en el tráfico de red.
- Indicar cuál es la dupla de direcciones destino (Ethernet-IP) que más se presenta en el tráfico de red.
- Señalar la cantidad de direcciones Ethernet origen y destino del flujo de bytes capturado.
- Señalar la cantidad de direcciones IP origen y destino del flujo de bytes capturado.
- Cantidad total de frames.
- Cantidad total de paquetes ARP.
- Cantidad total de paquetes IP.
- Cantidad total de paquetes TCP.
- Cantidad total de paquetes ICMP.
- Cantidad total de paquetes UDP.
- Cantidad de paquetes IP con opciones.
- Cantidad de paquetes TCP con opciones.
- Cantidad de paquetes ICMP con opciones.
- Cantidad de paquetes UDP con datos extra.
- Cantidad de paquetes IP con errores.
- Cantidad de paquetes TCP con errores.
- Cantidad de paquetes ICMP con errores.
- Cantidad de paquetes UDP con errores.

3.3 Diseño del analizador de protocolos.

El punto inicial para desarrollar el analizador de protocolos es programar un *snnifer* para capturar el tráfico de toda la red. Para establecer una comunicación entre dos hosts es necesario generar una conexión mediante sockets en cada uno, sin embargo en este caso sólo es necesario generar el socket del lado donde se ejecute el analizador de protocolos pero indicando como una de sus características la escucha de tráfico en modo promiscuo. En lenguaje C para linux un socket se genera mediante la función *int socket(int dominio, int tipo, int protocolo)* definida a través de los archivos *types.h* y *socket.h*. El parámetro que recibe llamado *dominio* se refiere a un *dominio de comunicaciones* que indica la familia de protocolos que se usará para la comunicación, algunos de los valores aceptados son: *PF_UNIX* para comunicación local, *PF_INET* que corresponde a protocolos de Internet IPv4 y *PF_PACKET* que se usa como interfaz para protocolos de bajo nivel y que para este caso es el más apropiado ya que los demás tipos sólo proporcionan información de los paquetes a nivel de los protocolos que refieren, y no de capas inferiores de la arquitectura de red y de la cual también es necesario para el análisis del flujo de red. El argumento *tipo* de la función socket se usa para indicar cómo se van a manejar los protocolos en la comunicación, por ejemplo, si el tipo es *SOCK_STREAM* se proporciona flujos de bytes basados en una conexión bidireccional secuenciada y confiable; si se indica *SOCK_DGRAM* se admiten mensajes no confiables, sin conexión y de una longitud máxima fija; y si se le da un valor de *SOCK_RAW* se tiene un acceso directo a los protocolos de red, que es lo conveniente para el analizador. El argumento *protocolo* indica el protocolo de red que se usará con el socket y acorde al objetivo de la aplicación en desarrollo el valor que se proporcione no debe limitar la captura de paquetes de uno solo sino debe incluir a todos los protocolos de la arquitectura sobre la que se trabaja. El valor que permite este funcionamiento es 0x0003 como se define en el archivo *if_ether.h* que pertenece a un grupo de cabeceras de lenguaje C para Linux.

Una vez que se ha creado el extremo de la comunicación para escuchar en modo promiscuo lo siguiente es establecer la forma de recibir las tramas, es decir, definir el comportamiento ante la llegada o no de información. En lenguaje C las funciones que permiten recibir información actúan en un inicio como *bloqueantes*, es decir, mientras no se reciba algún dato en el socket el programa que lleva a cabo la conexión se quedará esperando y no podrá recibir información a través de otro socket a pesar de que sí haya datos destinados a este último. Esto es importante en el caso de que se tenga un socket destinado a cada protocolo a escuchar, sin embargo, como el socket definido para esta aplicación es capaz de capturar frames que contengan cualquiera de los protocolos definidos no es estrictamente necesario establecer una forma de recepción de frames *no bloqueante*.

La siguiente fase a cubrir es la que se refiere a la interpretación de la estructura del flujo de bytes, es decir, reconocer a qué datos corresponden los bytes capturados y así poder ordenarlos. Al recibir los frames a través del tipo de socket definido los primeros bytes corresponden al encabezado Ethernet y en consecuencia los primeros 12 bytes están plenamente identificados y de acuerdo al valor de los bytes 13 y 14 se determina el tipo de frame Ethernet del que se trata y en consecuencia cómo se deben interpretar los datos que le siguen, es decir, la identificación del tipo de datos que contiene el flujo es de forma

progresiva y de lo general a lo específico. De esta forma se identifica que protocolo transporta el frame y cómo interpretar los campos que lo integran de acuerdo a lo indicado en el RFC correspondiente.

La etapa anterior se lleva a cabo al mismo tiempo que la fase donde se define el formato para la salida estándar, ya que conforme se va interpretando el significado de los bytes es necesario corroborar que tengan un valor razonable a través de su representación en forma de caracteres.

Al tener la representación del flujo de bytes para cada uno de los protocolos soportados el siguiente punto a abordar es la generación de las bitácoras de información. En este caso necesariamente se tienen que almacenar los datos en memoria secundaria, sin embargo es conveniente hacerlo de forma tal que el espacio ocupado sea reducido. Por esto se decidió guardar el flujo de bytes tal cual se recibía y hacer por separado un módulo para su posterior lectura, escritura en formato de texto en la pantalla de nueva cuenta, o la escritura del archivo XML.

Originalmente, al recibir cada frame del tráfico de red se hace su interpretación y escritura en pantalla, sin embargo, este procedimiento no se puede aplicar para leer el archivo en el cual se guardó el flujo ya que se tiene consecutivamente los bytes de cada frame y no hay una forma de diferenciarlos tal como *la recepción de cada frame*. De forma similar no se contaba con datos sobre la fecha y hora de llegada así que se decidió añadir al flujo de bytes datos que realizarían la tarea de describir esta y otra información de cada frame capturado con el propósito de facilitar y agilizar su procesamiento. Así, por ejemplo, se escribe antes de los bytes de la trama el tipo de protocolo que contiene (tcp, icmp, udp o arp) para ya no tener que leer del frame esta información y entonces determinar el procedimiento a realizar, sino directamente hacerlo al conocer el tipo de paquete. El procedimiento para la generación del archivo XML es muy parecido al que imprime en pantalla la interpretación de los datos capturados, solamente se cambia el descriptor del archivo de salida estándar por uno referente a un archivo para escribir y en lugar de imprimir la descripción del dato, éste se encierra entre las etiquetas adecuadas. Por otra parte, en lugar de que el descriptor del archivo del que se lee se refiera al socket, como se hace al capturar los datos, se especifica que indique al archivo en el cual se guardó el flujo de bytes.

Hasta este punto se cubre la primera de dos fases en cuanto a funciones establecidas para el analizador, faltando sólo su integración para manejar las características a través de un único programa. En los sistemas operativos GNU/Linux las herramientas y comandos que se ejecutan a través del shell reciben información sobre las características de ejecución por medio de banderas u opciones, con el fin de personalizar la salida de la aplicación según las necesidades. De este modo, es conveniente que el analizador de protocolos se pueda ejecutar indicando opciones para la captura y presentación de datos, así como para la generación de archivos. Estas características se definen en la parte de integración de las funciones para establecer que módulos se deben ejecutar de acuerdo a los intereses del usuario. Las banderas definidas para la fase de captura del flujo de red son las mostradas en la tabla 3.1.

Bandera	Función
p	Bandera de protocolos. Sirve para indicar la lista de protocolos separados por espacios o en su defecto la palabra *def* que indicará los 5 definidos de inicio: ip, tcp, icmp, udp y arp. Si sólo se indica el protocolo ip se leerán paquetes con los protocolos tcp, icmp y udp, ya que los 3 ocupan al protocolo IP como medio de transporte; para especificar solamente alguno de ellos bastará con indicar su nombre.
i	Interfaz de red. Especifica la interfaz de red por la que se realizará la captura de paquetes. Si no se indica alguna interfaz el valor predeterminado es eth0.
m	Bandera para establecer el modo de escucha de la interfaz de red, ya sea en forma promiscua o en forma como normalmente funciona.
v	Vista de información sobre el flujo capturado en pantalla. Bandera que indica que se despliegue en la salida estándar una representación de los paquetes capturados e información referente. Si esta bandera no se indica no se mostrará alguna información en el monitor.
f	Archivo. Esta bandera recibe como argumento el nombre del archivo donde se desea se haga el volcado del flujo de bytes capturado. Si no se indica esta bandera los datos capturados se guardarán en un archivo cuyo nombre será la palabra *flujo* concatenado con la fecha en formato *ddmmaaaa* (2 dígitos para el día, 2 dígitos para el mes y 4 dígitos para el año) y hora en que se ejecutó el programa, por ejemplo: *flujo08082008-01:22:20.*
x	Bandera para crear un archivo XML. Recibe como primer argumento el nombre del archivo del que se va leer el flujo de bytes capturado y como segundo el nombre del archivo de tipo XML se quiere generar.
t	Bandera para mostrar en pantalla con un formato de texto la información de los frames capturados y almacenados previamente en el archivo donde se hizo el registro del flujo capturado. Recibe como argumento el nombre del archivo del que se va leer el flujo de bytes capturado.

Tabla 3.1. *Opciones de ejecución del Analizador de protocolos.*

La siguiente etapa de desarrollo es la generación de estadísticas a partir de archivos XML. En este caso la programación se decidió hacer en lenguaje Java para aprovechar algunos objetos definidos en su *API* y algunos de sus métodos. Básicamente en este módulo lo que se hace es la lectura de un archivo renglón por renglón e identificar etiquetas XML y contenido característico por medio de expresiones regulares, y así realizar una cuenta de los elementos correspondientes a los datos que interesan para presentar estadísticas.

3.3.1 Descripción básica de la aplicación del modelo de proceso

La estrategia de desarrollo para esta aplicación es la conocida como *Modelo Incremental* y que de forma simple se conforma por varias etapas del *Modelo Lineal*; al final de cada secuencia se obtiene un resultado que marca un nivel de avance reconocible y funcional del software. Un diagrama que ejemplifica este proceso es el de la figura 3.1.

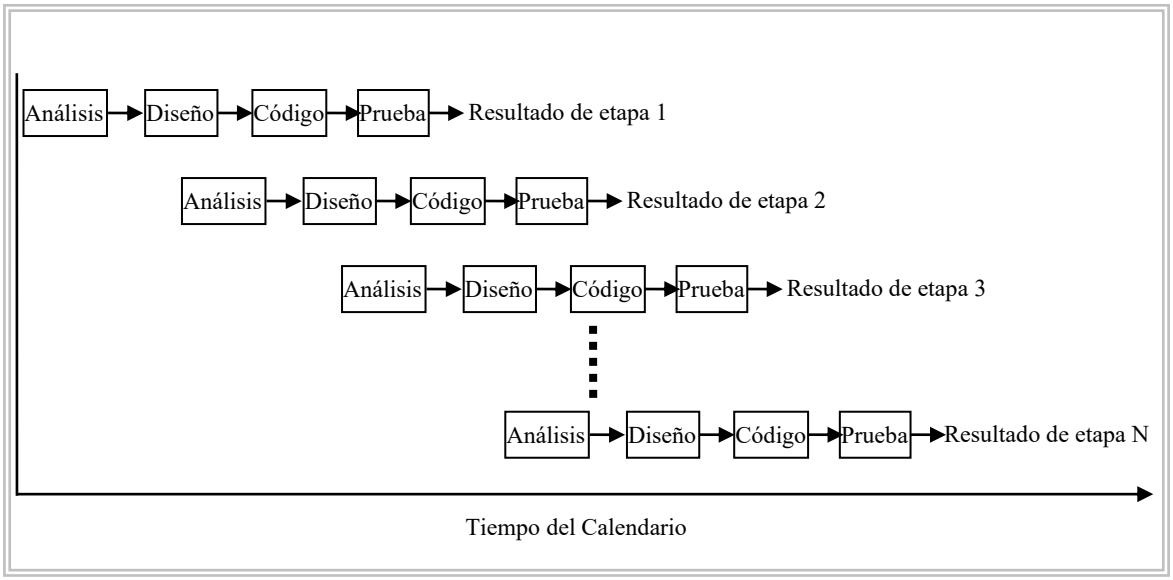

Figura 3.1. *Modelo incremental.*

Para realizar la programación de la aplicación se utilizó un enfoque progresivo definiendo primero el código para las tareas más generales, pero no por ello menos importantes, y que constituyen el punto de partida para el funcionamiento de la aplicación. De esta forma, la primer meta a cumplir fue la escucha de frames Ethernet dejando para el siguiente paso la identificación de los campos que los componen y su identificación. Una vez clasificados los tipos de frames Ethernet la siguiente labor fue la identificación del tipo de paquete de red que contiene para tomar en cuenta sólo los protocolos establecidos para soportar.

En una etapa inicial del análisis de protocolos se eligió trabajar con la interpretación de los paquetes IP para después abordar lo correspondiente a los paquetes ARP; así se procedió al reconocimiento de los campos que integran la cabecera de los paquetes del Protocolo Internet y en consecuencia conocer el protocolo del paquete transportado, además de reconocer cada una de las opciones posibles e indicar los campos que las conforman. De los protocolos que se encapsulan dentro del protocolo de Internet solamente 3 son los soportados en esta aplicación, y también en este caso se estableció uno para empezar la interpretación, el TCP, quedando el ICMP y el UDP pendientes. Para la interpretación de los paquetes TCP se inició por lo más general, es decir, saber leer la cabecera y enseguida cada una de las opciones posibles y por último los datos que transporta. Acabado este análisis se realizó un procedimiento similar con el Protocolo de Control de Mensajes y con el Protocolo de Datagramas de Usuario. Completado este módulo se procedió a realizar el correspondiente para paquetes ARP con el que se reconocería los campos que lo conforman y su contenido.

A cada uno de estos módulos se les fue agregando funcionalidad a través de código fuente para realizar la escritura de los datos recibidos en un archivo y no ya sólo la escucha. Posteriormente se diseñó código, basado en los que realiza la escucha de datos, para leer la información recabada y generar archivos XML descriptivos de los frames recibidos.

Después de esta etapa se desarrolló el módulo para presentar los datos de los frames, almacenados en un archivo previamente, en formato de texto en la salida estándar.

La siguiente tarea fue permitir parámetros de ejecución para la captura de frames y almacenarlos de acuerdo a su contenido. La última etapa de desarrollo consistió en generar un código para presentar estadísticas de un flujo de red capturado.

De acuerdo a la importancia de los avances obtenidos se establecen los resultados que marcan el fin de cada una de las etapas, dichos resultados que en conjunto conformaron toda la aplicación son:

1. Creación de un sniffer y escucha de frames Ethernet.
2. Interpretación de los campos de los diferentes tipos de trama ethernet y su clasificación; así como la presentación en pantalla de estos datos.
3. Identificación de paquetes IP, lectura e interpretación de su cabecera y presentación de esta información en un formato de texto simple en pantalla.
4. Identificación de paquetes TCP, lectura e interpretación de su cabecera y presentación de esta información en formato de texto en pantalla.
5. Identificación de paquetes ICMP, lectura e interpretación de su cabecera y presentación de esta información en formato de texto en pantalla.
6. Identificación de paquetes UDP, lectura e interpretación de su cabecera y presentación de esta información en formato de texto en pantalla.
7. Identificación de paquetes ARP, lectura e interpretación de su cabecera y presentación de esta información en un formato de texto simple en pantalla.
8. Escritura de los bytes que conforman cada frame capturado que transporta un paquete IP y a su vez un paquete TCP o ICMP o UDP en un archivo de texto, antecedidos por un encabezado descriptivo de los datos.
9. Escritura de los bytes que conforman cada frame capturado que transporta un paquete ARP en un archivo de texto, antecedidos por un encabezado descriptivo de los datos.
10. Generación de un archivo XML representativo de un flujo de red a partir de un archivo de texto que contiene frames capturados con un encabezado informativo de su contenido.
11. Presentación en pantalla en formato de texto simple de los campos que componen cada uno de los frames capturados y almacenados previamente en un archivo de texto, le cual funge como fuente de información.
12. Ejecución del sniffer indicando opciones específicas para la captura de datos, para el almacenamiento de datos y para la presentación de información.
13. Lectura de un archivo XML con los datos del flujo de red capturado y generación de estadísticas descriptivas de los frames y paquetes de red capturados.

3.4. Descripción del entorno de desarrollo

La revisión del entorno de desarrollo tiene gran importancia debido a que el analizador de protocolos está orientado a valerse de características propias del ambiente donde se encuentre instalado. En este apartado se describirán de forma general las características de las *herramientas* con las que se desarrolló este software con el fin de dejar en claro los requerimientos esenciales para su funcionamiento y dar a conocer bajo qué circunstancias se elaboró y que en caso de modificarse posteriormente por los usuarios se tenga en cuenta las *dependencias* que tiene con respecto a la estructura del sistema operativo y demás software utilizado.

3.4.1 Sistema operativo

Tanto el sistema operativo de desarrollo como el elegido para ejecutar el analizador de protocolos es *GNU/LINUX* el cual se puede considerar como sistema operativo de red. Surgió con la idea de tener un software de tipo UNIX por considerar que sus características esenciales eran buenas y a ellas se les podría añadir nuevas ideas sin dejarlas disfuncionales; y que se pudiera compartir con personas que fueran capaces de usarlo y/o adaptarlo a sus necesidades libremente sin verse restringidos por las reglas de confidencialidad establecidas como en el software que no es libre o atenidos a la corrección de errores por el propietario de los programas. Actualmente es usado como plataforma de servidores y como sistema operativo para oficina y para el hogar.

Comúnmente se dice que los sistemas GNU/Linux organizan la información a través de una estructura jerárquica de directorios conocida como de árbol, y es una buena concepción para entender como localizar archivos en el sistema operativo y para el fin de este documento, sin embargo hay que tener claro que la organización y localización real de los datos es más compleja.

Esta estructura de árbol parte de un elemento o nodo raíz representado por una diagonal */,* el cual es un directorio a partir del cual se desprenden ramas que en su extremo contienen ya sea un directorio del que se pueden generar más ramas, o un tipo de archivo. Para GNU/Linux, los archivos son sólo secuencias de bytes sin importar el tipo de información que contenga: de texto, información de imagen, códigos ejecutables e incluso los dispositivos de salida y de entrada los considera como archivos, y su nombre se limita a 256 caracteres. Sin embargo existe otra clasificación que los divide en los siguientes tipos:

- **Archivos regulares.** Contienen datos normales, por ejemplo archivos de texto, archivos de entrada a programas o la salida de uno de estos, e incluso los mismos archivos ejecutables.

- **Directorios.** Archivos que contienen la lista de otros archivos.

- **Archivos especiales.** Aquellos que se mapean a dispositivos de entrada/salida y a dispositivos de almacenamiento. De estos hay dos clases los de *tipo bloque* y los de *tipo caracter*. Los primeros de ellos son los que corresponden a dispositivos a través de los cuales el sistema mueve datos en forma de bloque, como por ejemplo discos duros, cdroms o regiones de memoria. Los de tipo carácter son aquellos a través de los que se transmite información byte por byte.

- **Links.** Los *links* son archivos que mapean a un conjunto de datos existentes. Cada archivo de tipo link se puede considerar como una forma de acceder al contenido de un archivo ya existente a través de otro nombre, el nombre del link.

- **Sockets.** Tipo especial de archivo que se utiliza para identificar una conexión de red y el flujo de información correspondiente.

- **Pipes.** Son una forma de comunicación entre procesos sin usar la semántica propia de los sockets de red.

Se conoce como *ruta de localización absoluta* de un directorio a la descripción de su ubicación en la estructura de archivos describiéndola desde el directorio raiz y separando los nombres de directorios por una diagonal, por ejemplo, la ruta */usr/local/bin/glib-config* indica que el archivo *glib-config* se encuentra en el directorio *bin* que se encuentra en el directorio *local* y este a su vez está dentro del direcorio *usr* el cual se encuentra en el directorio /.

Existen directorios propios del sistema operativo y que tienen un uso específico de acuerdo al tipo de archivos que contienen, la gran mayoría se presentan en todas las distribuciones y algunos son específicos a una sola.

3.4.2 Dependencia de archivos

Fueron varios archivos de tipo cabecera usados para la programación en lenguaje C. Fue indispensable utilizar alguno de ellos debido a que refieren a funcionalidades para el manejo de conexiones de red que el mismo sistema operativo utiliza; mientras que de otros se pudo omitir su uso y crear funciones propias, sin embargo fueron utilizados para concordar con la forma de manejar el tráfico de red por GNU/LINUX. Este tipo de cabeceras se encuentran en la ruta */usr/include* y en la tabla 3.2 se presenta una descripción general de los que fueron utilizados.

Archivo cabecera	Descripción
stdio.h	Entrada/Salida estándar con buffer. Como su nombre lo dice proporciona funcionalidad para el manejo de las entradas y salidas estándar. Define macros, expresiones constantes enteras positivas y negativas, constantes de tipo puntero a NULL, constantes de tipo cadena, apuntadores a archivos y datos definidos a través de la expresión *typedef*.
stdlib.h	Definiciones de librerías estándar. Contiene funciones relacionadas a la asignación de memoria, control de procesos y conversiones entre otras. Define macros y tipos de datos a través de typedef.
string.h	Define funciones para el manejo de cadenas.
time.h	Define funciones para el manejo de tiempos y fechas.
sys/timeb.h	Definiciones adicionales a time.h para tiempos y fechas.
sys/types.h	Definiciones de tipos de datos.
sys/socket.h	Define datos y funciones para el manejo de sockets.
fcntl.h	Define solicitudes y argumentos usados por las funciones fcntl() y open().
unistd.h	Define constantes simbólicas estándar y tipos de datos.
locale.h	Define macros para ajustar valores correspondientes a una locación en específico.
net/ethernet.h	Define una cabecera para el manejo de frames Ethernet. Su código incluye al archivo *linux/if_ether.h*.
netinet.h/ip.h	Define una cabecera para el manejo de paquetes IP.
netinet/in.h	Definiciones para la familia del Protocolo Internet. Establece tipos de datos. macros y estructuras para el manejo del Protocolo Internet en cuanto a programación.
ctype.h	Define prototipos de funciones que evalúan tipos de caracteres.
netinet/ip_icmp.h	Define estructuras para el manejo de la cabecera de paquetes icmp y constantes propias de este protocolo.
netinet/udp.h	Define estructuras para el manejo de la cabecera de paquetes UDP.
net/if_arp.h	Define estructuras para el manejo de la cabecera de paquetes arp y constantes propias de este protocolo.

Tabla 3.2. *Archivos de cabecera utilizados en la programación del analizador de protocolos.*

3.4.3 Lenguajes de programación utilizados

En un principio se pensó realizar el analizador de protocolos con un solo lenguaje y en consecuencia ocupar solo un estilo de programación, sin embargo, conforme se fue desarrollando el programa se optó por utilizar 2 *paradigmas de programación*.

En la programación estructurada se tiene un enfoque orientado hacia la máquina, sin embargo también tiene como objetivo realizar una programación de fácil comprensión.

Implica un alto grado de estructuración debido a que para realizar alguna tarea se divide todo el proceso en procedimientos de menor complejidad y una vez desarrollada cada parte del programa se procede a su integración con las demás.

En cada una de las estructuras que componen al programa las tareas se llevan a cabo a través de operaciones de secuencia, selección e iteración, y se definen con base a *qué es lo que hace cada una* y *cómo lo hace.*

El objetivo de la *programación orientada a objetos* es facilitar el desarrollo de aplicaciones dando al programador una visión más cercana a lo que es el mundo real para tratar las tareas que debe realizar el programa. Básicamente, una aplicación se compone por un conjunto de *Objetos* que interactúan entre ellos a través de *Métodos* para realizar diversas tareas con base en sus *Atributos* (datos del objeto); cada uno de estos objetos es una abstracción de un actor que realizaría alguna labor para conseguir el fin de la aplicación. Cada objeto se define a través de un fragmento de código denominado clase, y está perfectamente definido en cuanto a sus atributos y métodos que puede llevar a cabo, por lo cual, este paradigma de programación permite una mayor modularidad del sistema y en consecuencia su fácil modificación y mejoramiento. Entre las características más destacables de este paradigma se encuentran la *Herencia de Clases*, que se refiere a la capacidad de compartir métodos y/o atributos a clases denominadas hijas; el *Principio de ocultación* que radica en permitir el acceso a determinados métodos o atributos de un objeto sólo a objetos con cierto nivel de permiso; *Polimorfismo*, que refiere a la capacidad de realizar tareas distintas identificadas por el mismo nombre a través de objetos diferentes; y la sobre escritura de métodos que permite a una clase hija realizar uno de los métodos de la clase padre de forma diferente a como lo haría esta última.

3.4.3.1 Compilador de C

El lenguaje de programación C originalmente se pensó para la programación del sistema operativo UNIX, pero con el paso del tiempo se ha ocupado para desarrollar diversas aplicaciones. Fue diseñado en el año 1969 en los Laboratorios Bell por Dennis Ritchie y Ken Thompson, pero su primera estandarización se realizó hasta el año de 1989 por el Instituto Nacional Americano de Estándares (ANSI por sus siglas en inglés). Un año después, la Organización Internacional de Estándares (ISO) modificó un poco el *ANSI C* y definió el estándar ISO/IEC 9899:1990; para el año 1999 publicó el estándar ISO 9899:1999, mejor conocido como *C99* y que se adoptó como el nuevo estándar ANSI en el año 2000.

El diseño del estándar ANSI para el lenguaje de programación C tiene como uno de sus principales objetivos la portabilidad de los programas, es decir, que funcionen de forma correcta independientemente de la plataforma donde se desee ejecutar siempre y cuando la implementación del lenguaje sea compatible; sin embargo, es común que los diseñadores de los compiladores o de los sistemas operativos soporten o añadan elementos extras al estándar para obtener ciertos beneficios en cuanto a programación de aplicaciones y en consecuencia la compatibilidad no se presenta como se desearía.

GNU/Linux está programado en lenguaje C, sin embargo no se trata de un estándar ISO ni ANSI por modificar algunas definiciones en la estructura de los archivos de tipo encabezado, añadir nuevos o prescindir de otros; sin que esto signifique que sean lenguajes de programación radicalmente diferentes, ya que aspectos esenciales del lenguaje C, como la sintaxis, son prácticamente los mismos. Por esta razón al lenguaje C para desarrollo de aplicaciones en este sistema operativo comúnmente se le llama *C para Linux.*

Para el ambiente de desarrollo elegido existen varios compiladores de C, como el cc (*C Compiler*) que también está disponible en plataformas UNIX. El compilador de C utilizado para el analizador de protocolos desarrollado es el que forma parte de la *Colección de Compiladores de GNU* (GCC, por sus siglas en inglés) y que es parte del *GNU Project*. GCC se puede incluir como parte de la instalación inicial de las distribuciones de este sistema operativo y también su instalación y/o actualización por una versión más reciente se puede hacer una vez instalado GNU/LINUX; es capaz de compilar códigos fuente escritos en C, C++, Objective-C, Fortran, Java y Ada, además de incluir librerías para realizar dichas tareas.

Con el compilador para lenguaje C de GCC se pueden realizar las 4 etapas de compilación paso a paso o todas juntas a través de una sola instrucción. Dichas fases son las siguientes:

- **Preprocesado.** Interpretación de las directivas al preprocesado y eliminación de comentarios.
- **Compilación.** Transforma el código fuente que es resultado de la etapa de preprocesado en lenguaje ensamblador correspondiente al procesador de la máquina.
- **Ensamblado.** Convierte el programa en lenguaje ensamblador a código objeto, que es un *código máquina* que se puede ejecutar directamente por el procesador.
- **Enlazado.** Fase dónde el código objeto del programa compilado se reúne con otros códigos objeto de algunas funciones propias del lenguaje y existentes en las bibliotecas del sistema. Como resultado proporciona el programa ejecutable.

Este compilador de lenguaje C permite realizar la etapa de enlazado de forma dinámica o estática. Ésta primera forma de enlazado significa que el código de las funciones existentes que requiere el programa en C se cargará en memoria al momento de ejecución desde las bibliotecas del sistema, con lo cual se obtiene un programa ejecutable de tamaño pequeño pero con dependencias de dichas bibliotecas. El enlazado estático refiere a crear un programa ejecutable donde los códigos binarios de las funciones de las bibliotecas de C que requiere el programa se unen a su código binario y con lo cual se obtiene un ejecutable de mayor tamaño que el obtenido con el enlace dinámico.

La versión del compilador utilizado es la 4.1.1, y en su versión actual (4.3.0) aún no soporta completamente las características indicadas en la norma ANSI para el lenguaje C, pero sí la gran mayoría de ellas.

3.4.3.2 Compilador de java

Existen varios compiladores para programas escritos en java, algunos optimizados para determinado hardware y/o sistema operativo o software, como por ejemplo el *Eclipse Compiler for Java* que es un compilador de código abierto usado por Eclipse o el GCJ que es parte del GCC. Cada compilador de java debe tener la función de convertir el código fuente que se le proporciona a un código de tipo *bytecode*, el cual tiene como característica ser independiente de la plataforma de software y de hardware sobre la que se ejecute por ser interpretado por la *máquina virtual de* java, la cual junto con una *Interfaz de Programación de Aplicaciones* (API, por sus siglas en inglés) conforma la plataforma de Java.

Como no son necesarios requerimientos especiales para la compilación del código fuente para el analizador de protocolos se trabajó con el *Java programming language compiler* (javac) que es el compilador proporcionado por *Sun Microsystems* como parte del *Java Depelopment Kit* (JDK) para construir aplicaciones. El software de desarrollo para la parte de java es la versión 1.6.0_01 de JDK obtenida desde el sitio web oficial de java. La versión mínima del JDK para poder utilizar el programa escrito en java es la 1.5.

3.4.4 Datos técnicos del hardware y sistema operativo de desarrollo

El programa fue desarrollado en equipo con las siguientes características técnicas de acuerdo a lo indicado por el sistema operativo:

- Procesador Pentium III a 996.776 MHz con 16 Kilobytes de memoria caché nivel 1, 216 Kilobytes de memoria caché nivel 2.
- Memoria RAM de 512 Megabytes.
- Bus de datos a 133 Mhz.

En cuanto a las características del sistema operativo GNU/Linux se tiene:

- Fedora Core 6 como distribución.
- Versión de kernel 2.6.18-1.2798.fc6.
- 512 Megabytes de memoria swap.

El programa hace uso de 20 Megabytes de memoria al reservar esa cantidad para almacenar directamente el frame en turno, en consecuencia tendrá un buen desempeño en una computadora con 128 Megabytes de memoria RAM, que actualmente es la cantidad mínima de memoria disponible para la gran mayoría de las computadoras en uso.

Capítulo 4
Desarrollo

4.1 Descripción general de la programación

El analizador de protocolos se integra por 2 programas, el primero de ellos lleva el nombre de *Netbender*, está programado en lenguaje C para Linux y realiza las tareas de captura del flujo de red, creación del archivo donde se guarda dicha secuencia de bytes, impresión en pantalla de información descriptiva de los datos de cada frame al tiempo que se capturan o posteriormente a partir del archivo ya mencionado, y creación de un documento con la representación de los datos de los frames capturados a través de etiquetas XML. El segundo programa se llama *analizaFlujo* y está escrito en lenguaje Java; tiene como propósito la generación de estadísticas del flujo de red capturado a partir del archivo XML que lo contiene.

En los diagramas de flujo de la figura 4.1 (partes a, b y c) y de la figura 4.2 se muestran las tareas generales que hace cada uno de los programas y el orden en que se llevan a cabo.

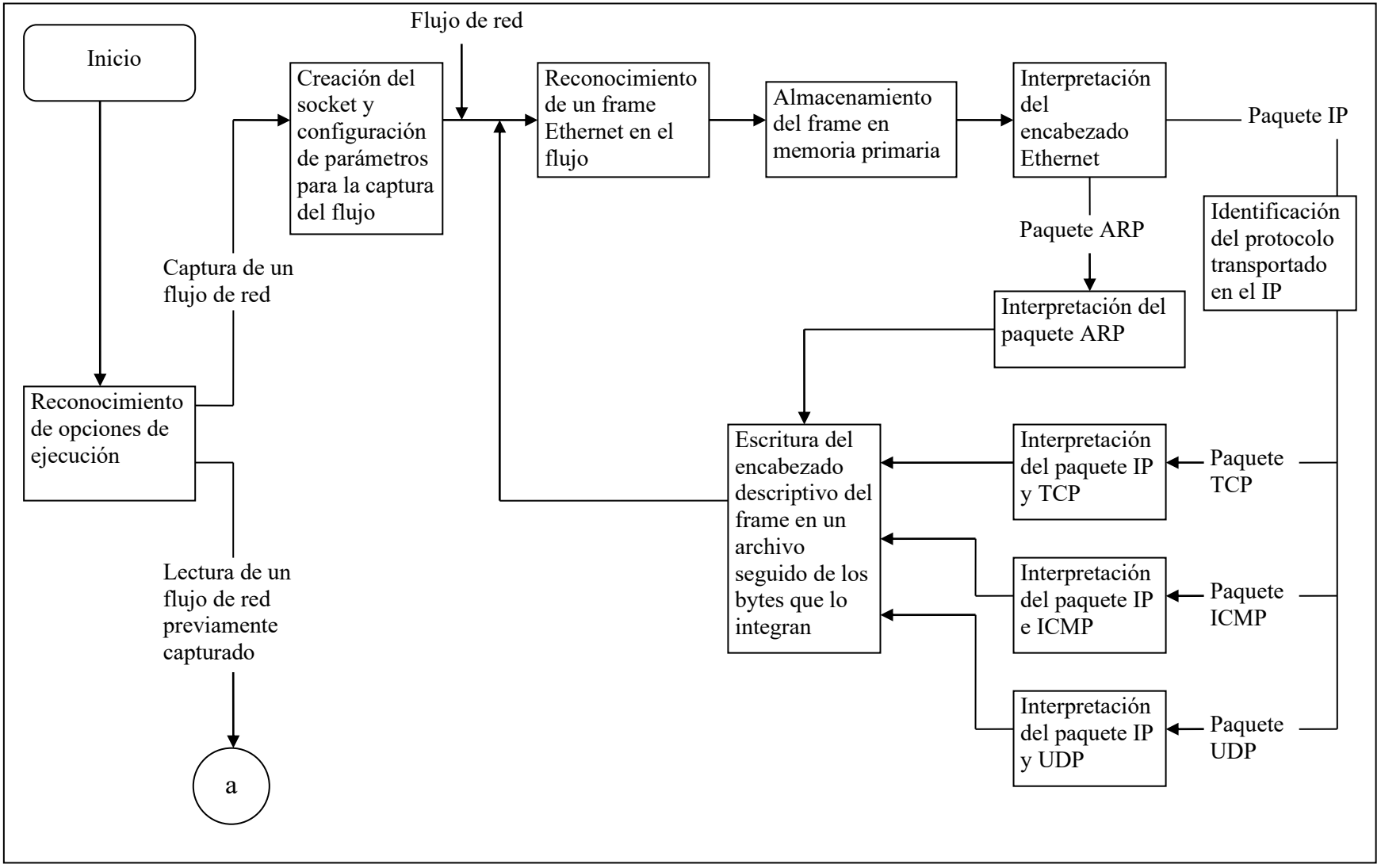

Figura 4.1. *Diagrama de flujo para el programa Netbender, parte a.*

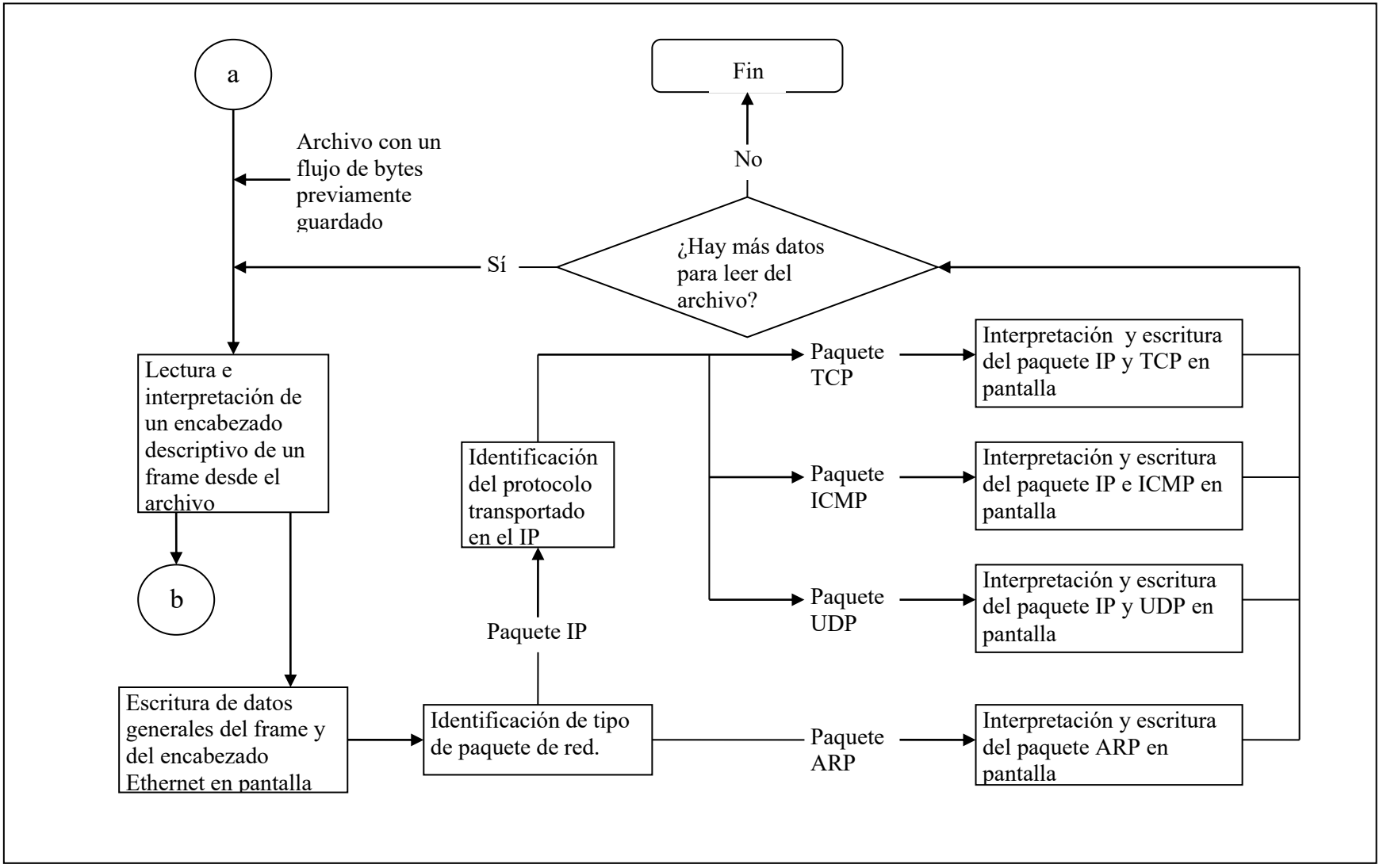

Figura 4.1. *Diagrama de flujo para el programa Netbender, parte b.*

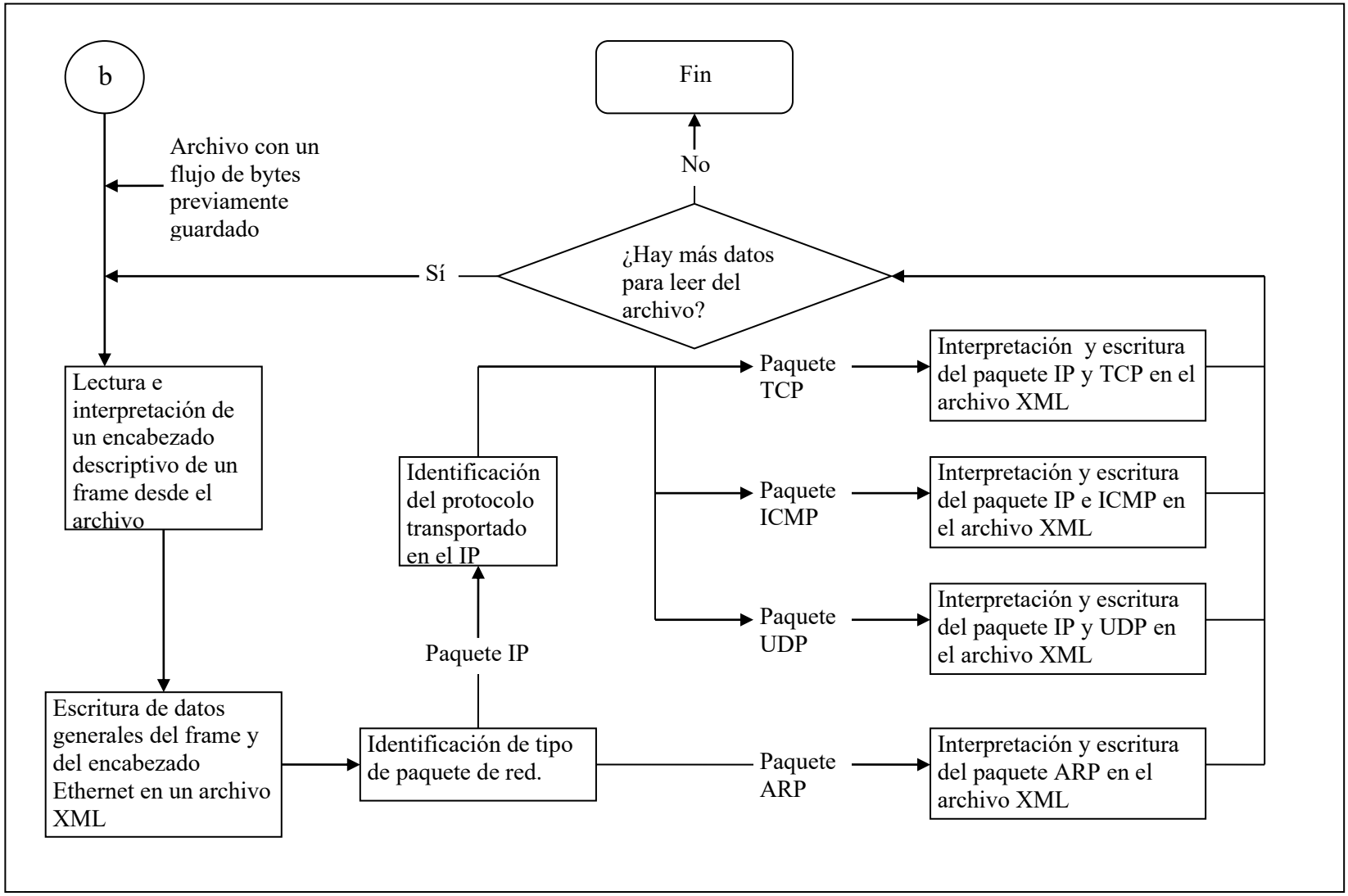

Figura 4.1. *Diagrama de flujo para el programa Netbender, parte c.*

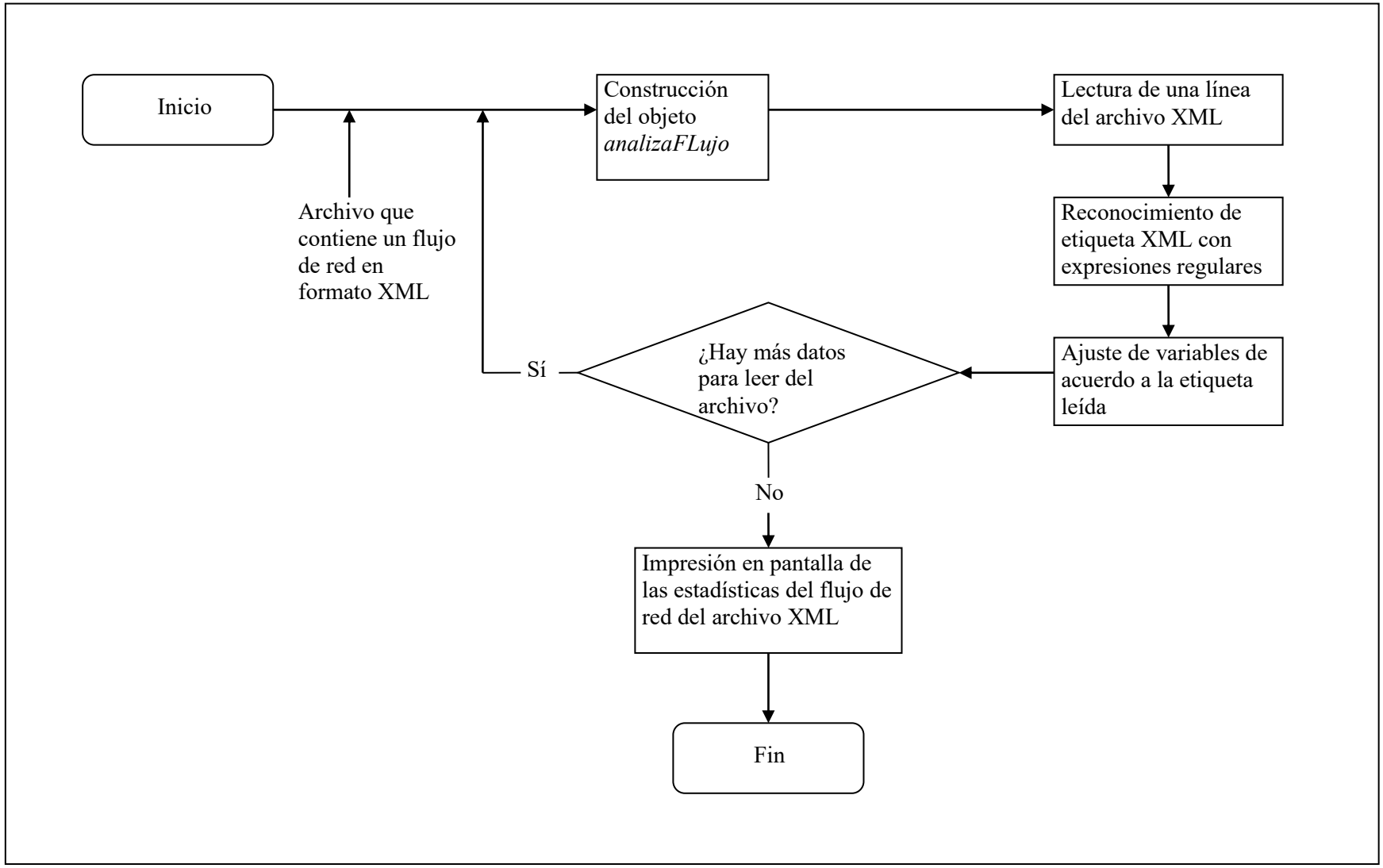

Figura 4.2. *Diagrama de flujo para el programa analizaFlujo.*

El programa Netbender es el archivo ejecutable resultado de la compilación de varios códigos fuente, cada uno de los cuales contiene distintas funciones que se encargan de cierta tarea. Esta organización del código fuente permite hacerlo más comprensible porque cada archivo se enfoca a un ámbito de procedimientos específicos a realizar, lo cual proporciona una dependencia entre las funciones reducida solamente al tipo de datos que reciben y proporciona como salida, lo que facilita su modificación en caso de ser necesario ya que el usuario no tendrá que revisar todos los archivos para ubicar el código fuente de alguna tarea en particular y minimiza en riesgo de alterar código accidentalmente y dejar el programa con mal funcionamiento. Los archivos que se ocupan para generar el Netbender son:

- paquetesTCP.c
- paquetesICMP.c
- paquetesUDP.c
- paquetesARP.c
- xmlTCP.c
- xmlICMP.c
- xmlUDP.c
- xmlARP.c
- textoTCP.c
- textoICMP.c
- textoUDP.c
- textoARP.c

Incluidos en el programa principal a través de los archivos de tipo cabecera llamados: paquetesTCP.h, paquetesICMP.h, paquetesUDP.h, paquetesARP.h, xmlTCP.h, xmlICMP.h, xmlUDP.h, xmlARP.h, textoTCP.h, textoICMP.h, textoUDP.h y textoARP.h que indican el prototipo de cada función almacenada en los archivos de código fuente mencionado. Además de los archivos indicados en el apartado *3.4.2 Dependencia de archivos*.

Por otra parte el código fuente *analizaFlujo.java* importa las siguientes clases del API de java:

- java.io.BufferedReader
- java.io.File
- java.io.FileReader
- java.io.FileNotFoundException
- java.io.IOException
- java.util.ArrayList
- java.lang.String.*

4.2 Descripción de códigos fuente en lenguaje C

En esta sección se explican los segmentos de código cuya comprensión no es tan fácil como el resto de acuerdo a la lógica del programa y a que utilizan funciones no tan frecuentes en una programación que no tiene que ver con manejo de flujos de red, además de abordar tareas que refieren a cuestiones indispensables para el funcionamiento del analizador de protocolos.

Para cubrir los segmentos de códigos no explicados se describe el uso que se les dará a cada una de las variables ocupadas con el fin de que el mismo alumno o usuario del programa pueda deducir por su propia cuenta el funcionamiento de dichos bloques de código fuente.

4.2.1 Archivo *Netbender.c*

En este código fuente se incluye la función *main()* de la aplicación y en consecuencia a partir de él comienza la ejecución del analizador de protocolos. En este archivo se incluyen instrucciones para establecer la secuencia de tareas a realizar a partir de los argumentos proporcionados o identificar errores en los parámetros de ejecución. A continuación se describen con detenimiento segmentos de código importantes de este archivo.

4.2.1.1 Función main(int argc, char **argv)

Esta función básicamente se encarga de leer las banderas de ejecución para el programa e identificar cada uno de sus argumentos, además de encontrar errores de sintaxis en la forma en como se proporcionan las opciones.

Las variables que se ocupan en esta función se indican en la tabla 4.1.

Variable	Descripción
int i	Variable de tipo contador para realizar iteraciones.
int protocolos	Variable que indica si ya se han leído todos los argumentos para la bandera -p (valor de 1) y en consecuencia ya no seguir en el ciclo correspondiente a esa tarea.
int masOpc	Si adquiere un valor de 0 indica que hasta la iteración actual para identificar el tipo de argumento pasado al programa ya se ha clasificado alguno de ellos como de tipo bandera u opción y registrado sus argumentos respectivos. En consecuencia se debe comparar hasta la siguiente iteración el argumento actual para saber si es de tipo bandera.
char tiempo[20]	Arreglo que almacena bajo un formato específico la fecha y la hora en que se ejecuta el programa para formar parte del nombre del archivo donde se guardará el flujo de bytes capturado.
struct timeb T	Estructura donde se almacena la fecha y hora actuales a través de la función *ftime(struct timeb *pt)*.
struct tm *td	Apuntador a una estructura de tipo *tm*, la cual sirve para almacenar una representación de tiempo separada en distintos campos (segundos, minutos, horas, etc.).
char archivob[22]	Arreglo que almacena el nombre del archivo donde se guardará el flujo de bytes capturado.
int tcp, int udp, int icmp, int arp	Variables que adquieren el valor de 1 para indicar en la lógica del programa que se tiene que capturar paquetes de los protocolos tcp, udp, icmp y arp, respectivamente. Su valor de inicio es de 0.
int pantalla	Inicialmente tiene un valor de 0. Toma un valor de 1 para indicar en la lógica del programa que se muestre en pantalla una representación en formato de texto de los datos capturados.

char *interfaz	Apuntador de tipo carácter que indica el nombre del dispositivo de red por el que se realizará la captura de paquetes. Se inicializa con la cadena eth0.
char *archivoBytes	Apuntador de tipo carácter que señala al arreglo que contiene el nombre del archivo donde se guardará el flujo de bytes capturado.
char *archivoOrigen	Apuntador de tipo carácter que refiere al nombre del archivo desde el cual se leerán datos en caso de ejecutarse el programa con las banderas -x o -t. Se inicializa con un espacio en blanco.
char *archivoDestino	Apuntador al nombre del archivo en el que se guardarán los datos del flujo de red en formato XML. Se inicializa con un espacio en blanco.
char modo	Variable de tipo carácter que si se ajusta a un valor de "P" indica en la lógica del programa que la escucha a través de la interfaz de red sea en modo promiscuo; y si se ajusta a un valor de "N" indica que se realice la escucha de forma habitual a como se hace en la tarjeta de red. De inicio se le da un valor de "P".
int valcreaFormato	Variable que almacena el valor regresado por la función *creaFormato()*.
int valleeFlujo	Variable que almacena el valor regresado por la función *leeFlujo()*.

Tabla 4.1. *Variables utilizadas en la función main().*

Una vez identificados dichos parámetros de ejecución y ajustado las variables correspondientes ejecuta la función *creaFormato()* o *leeFlujo()*.

4.2.1.2 Función leeFlujo(int ptcp, int pudp, int picmp, int parp, int pantalla, char modo, char *interfaz, char *archivoBytes)

Esta función se encarga de escuchar el tráfico de red, interpretar y mostrar en pantalla la información correspondiente al encabezado Ethernet, así como de llamar a la función adecuada según el tipo de paquete de red transportado para seguir interpretando y mostrando sus datos. Como tareas esenciales que realiza están: crear el socket y configurarlo para realizar la escucha del tráfico de red, configurar la interfaz de red a utilizar en modo promiscuo o en modo común, recibir cada frame del flujo de red, clasificarlo y llamar a otras funciones para seguir manipulándolo.

Las variables que ocupa son las especificadas en la tabla 4.2.

Variable	Descripción
int ptcp, int pudp, int picmp, int parp	Variables que indican si se debe realizar la escucha de paquetes tcp, icmp, udp y arp, respectivamente.
int pantalla	Variable que indica si se debe imprimir en pantalla la información capturada de cada paquete con texto descriptivo.

char modo	Variable de tipo carácter que si se ajusta a un valor de "P" indica en la lógica del programa que la escucha a través de la interfaz de red sea en modo promiscuo; y si se ajusta a un valor de "N" indica que se realice la escucha de forma habitual a como se hace en la tarjeta de red.
char *interfaz	Apuntador de tipo carácter que indica el nombre del dispositivo de red por el que se realizará la captura de paquetes. Se inicializa con la cadena eth0.
char *archivoBytes	Apuntador de tipo carácter que señala al arreglo que contiene el nombre del archivo donde se guardará el flujo de bytes capturado.
char *loc	Apuntador a la cadena de caracteres que indica la localización establecida del sistema.
int fd	Variable destinada para almacenar el descriptor del socket creado para la captura del tráfico de red.
int fl	Variable para almacenar las banderas de control del descriptor de socket.
int funPaq	Variable para almacenar el valor devuelto por las funciones para el manejo de los distintos tipos de paquetes de red del flujo leído.
unsigned char buffEth[20000]	Arreglo de tipo carácter para almacenar el frame Ethernet en turno desde el flujo de bytes capturado.
unsigned char *buffeth	Apuntador hacia el primer elemento del arreglo donde se almacena el frame leído.
unsigned char *buff	Apuntador a la dirección de memoria localizada después de lo que ocupará el encabezado del frame Ethernet en el arreglo buffEth.
char tiempo[20]	Arreglo de caracteres para almacenar la fecha y hora en la que se recibe en frame Ethernet en turno desde el flujo de bytes.
struct ifreq datosInterfaz	Es una estructura que almacena información sobre el dispositivo de red ocupado. Para mayor información se puede consultar el manual del Programador de Linux para *netdevice*.
int indiceInterfaz	Almacena el índice identificador del dispositivo de red perteneciente a la tarjeta a través de la que se realizará la escucha del tráfico de red.
struct timeb T	Estructura donde se almacena la fecha y hora actuales a través de la función *ftime(struct timeb *pt)*. Para mayor información se puede revisar el manual del Programador de Linux de dicha función.
struct tm *td	Apuntador a una estructura de tipo *tm*, la cual sirve para almacenar una representación de tiempo separada en distintos campos (segundos, minutos, horas, etc.). Para mayor información revisar el manual del Programador de Linux de la función *localtime*.
struct ether_header *eth	Apuntador a una estructura *ether_header* que contiene como miembros a los campos del encabezado Ethernet que ocupan los primeros 14 bytes de un frame de este tipo. Estructura definida en */net/ethernet.h*.

struct iphdr *ip	Apuntador a una estructura *iphdr* que contiene como miembros a cada uno de los campos que conforman un paquete IP. Estructura definida en */netinet/ip.h*.
struct arphdr *arp	Apuntador a una estructura *arphdr* que contiene como miembros a cada uno de los campos que conforman un paquete ARP. Estructura definida en */net/if_arp.h*.
int tamEth	Almacena el tamaño del encabezado Ethernet del frame en cuestión.
int noFrame	Variable para contabilizar el número de frames leídos.
unsigned short *temp2B	Variable para almacenar temporalmente datos de 2 bytes.
struct sockaddr_ll dir	Estructura que almacena información de la cabecera del nivel de enlace para la arquitectura de red TCP/IP. Para mayor información revisar el manual del Programador de Linux de *packet*.

Tabla 4.2. *Variables utilizadas en la función leeFlujo().*

Para crear el socket que representa el extremo de la comunicación por el que se realiza la captura del flujo de bytes se ocupa la siguiente línea:

```
socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
```

En este caso el dominio de comunicaciones (familia de protocolos que se usará para el socket) es el identificado por *PF_PACKET* y que refiere a una interfaz para protocolos de bajo nivel, lo cual permite recibir o enviar paquetes tal cual son emitidos (paquetes RAW) en la capa de enlace. Con el argumento *SOCK_RAW* se permite un acceso directo a los protocolos de red de los paquetes RAW a través del socket. El último argumento para esta función indica el protocolo que se capturará y que debe ser expresado en bytes en orden de red para lo cual se ocupa la función *htons()*. Este protocolo se especifica a través de un número entero, pero comúnmente se indica en notación hexadecimal o por medio de una variable declarada través de la directiva define en el archivo *if_ether.h*. La variable *ETH_P_ALL* tiene el valor de 0x0003 e indica que se capturen todos los protocolos.

Con las líneas siguientes se hace que el apuntador *eth* de tipo *ether_header* apunte al inicio del arreglo *buffEth*, y que los apuntadores *ip* y *arp*, a estructuras de tipo *iphdr* y *arphdr* respectivamente apunten inicialmente a *buff*, que es un apuntador a donde inician los datos del datagrama Ethernet (inicio del paquete IP o del paquete ARP).

```
struct ether_header *eth = (struct ether_header*) buffEth;
struct iphdr *ip = (struct iphdr*) buff;
struct arphdr *arp = (struct arphdr*) buff;
```

Con la línea:

```
strcpy(datosInterfaz.ifr_name, interfaz);
```

se almacena el valor de la variable interfaz en el miembro *ifr_name* de la estructura *datosInterfaz*.

La función *ioctl()* manipula parámetros propios de distintos dispositivos; si hay un error en su ejecución devuelve como un valor de -1. Con la línea:

```
ioctl (fd, SIOCGIFINDEX, &datosInterfaz);
```

se obtiene el índice de la interfaz asociada con *datosInterfaz* y se almacena en *datosInterfaz.ifr_ifindex*. Más referencias de esta función se pueden encontrar en las páginas de manual (*manpage*) de *ioctl(int d, int solicitud, ...)* y *netdevice*.

Con la siguiente línea se copia el valor de *datosInterfaz.ifr_ifindex* en *indiceInterfaz*.

```
memcpy ((void *) &indiceInterfaz, (void *) &datosInterfaz.ifr_ifindex,1);
```

Con la siguiente línea se obtiene la palabra de banderas activas del dispositivo y se almacena en *datosInterfaz.ifr_flags* para después ajustarlas y realizar la escucha de datos en modo promiscuo o en modo normal.

```
ioctl (fd, SIOCGIFFLAGS, &datosInterfaz);
```

Para ajustar los valores de dichas banderas se hace uso de máscaras de bits especificadas en la página de manual de *netdevice*. De esta forma, con la línea

```
datosInterfaz.ifr_flags |= IFF_PROMISC;
```

se hace una operación OR de la palabra, obtenida previamente, que indica las banderas actuales con la máscara *IFF_PROMISC* para añadir la funcionalidad de escucha en modo promiscuo. El paso final es establecer este nuevo valor de banderas en la estructura de la interfaz a través de la línea:

```
ioctl (fd, SIOCSIFFLAGS, &datosInterfaz);
```

La palabra de banderas activas en un estado inicial, es decir, sin modificación previa, tiene un valor decimal de 4163 (valor obtenido experimentalmente); razón por la cual para realizar una escucha en modo Normal basta con asignar ese valor a *datosInterfaz.ifr_flags* directamente y sin necesidad de realizar una operación OR con alguna bandera, además de que no existe una máscara de bits con cual hacer una operación de bits para regresarla a un estado inicial.

En la lógica del programa lo siguiente es asociar el socket a una interfaz de red en específico.

Para esto se ocupan los campos *sll_protocol* y *sll_ifindex* de una estructura de tipo *sockaddr_ll* que identifica al dispositivo que se utilizará.

```
dir.sll_family = PF_PACKET;
dir.sll_protocol = htons(0x0003);
dir.sll_ifindex = indiceInterfaz;
```

Con las sentencias anteriores se inicializan los miembros que interesan de la estructura *dir*. Enseguida se realiza el enlazado del socket con la interfaz de red a través de la instrucción:

```
bind(fd,(struct sockaddr *)&dir,sizeof(struct sockaddr_ll));
```

Como argumentos recibe el descriptor del socket (*fd*), los datos de la interfaz con cual enlazarlo, y la longitud en bytes de la estructura que representa dicha interfaz. La función regresa un valor de -1 en caso de error.

Otra forma para realizar en enlazado es ajustando opciones del socket a través de la función *setsockopt()* con la siguiente sentencia:

```
setsockopt(fd,SOL_SOCKET,SO_BINDTODEVICE,interfaz,strlen(interfaz));
```

Se proporcionan como argumentos el descriptor del socket, el nivel en el cual se va a configurar proporcionando el número de protocolo sobre el que se trabajará para niveles superiores o con la palabra *SOL_SOCKET* para indicar que se trabajará en el nivel del conector. Posteriormente se indica la opción del conector a configurar y que en nuestro caso es la que permite asociarlo con un dispositivo, es decir, la opción *SO_BINDTODEVICE*; después el nombre del dispositivo y por último el tamaño del buffer al que señala el apuntador de la interfaz. Sin embargo, este método sólo es funcional cuando se manejan protocolos pertenecientes a la familia *PF_INET*.

Con las siguientes líneas se configura el socket de forma no bloqueante.

```
fl = fcntl(fd, F_GETFL);
fcntl(fd, F_SETFL, fl | O_NONBLOCK);
```

La primera de ellas guarda en *fl* las banderas del descriptor del socket, y la segunda las establece como no bloqueantes a través de una operación OR entre su estado inicial y el valor de la variable *O_NONBLOCK* establecida por medio de la directiva define en el archivo *bits/fcntl.h* incluido por medio del archivo *fcntl.h*, cabecera del código fuente principal. La función *fcntl(int fd, int cmd)* regresa un valor de -1 en caso de error y cero en caso de un funcionamiento adecuado para los comandos *F_GETFL* y *F_SETFL*.

Hasta este momento el socket está configurado para captar paquetes de red y ya puede realizar esa tarea, sólo hay que indicar qué hacer cada que se reciba un nuevo frame Ethernet. Estas indicaciones deben proporcionarse para cada frame que se recibe con el fin de empezar a interpretarlo y darle el manejo correspondiente a su contenido, en consecuencia esas instrucciones se ponen en un ciclo infinito y se repetirán hasta que se indique su interrupción por medio del teclado. La sentencia que genera este ciclo infinito es:

```
for (;;)
{
      ...
}
```

En el ciclo infinito mencionado se recibe cada frame del flujo de red y se procede a su manejo, para ello los bytes capturados se almacenan en un arreglo de caracteres y se manejan a través de apuntadores. Lo anterior se consigue con la línea:

```
if(recvfrom(fd, buffEth, sizeof(buffEth), 0, NULL, NULL)>0);
```

La función *recvfrom(int s, void *buf, size_t lon, int banderas, struct sockaddr *origen, socklen_t *longDestino)* devuelve como resultado de su ejecución la cantidad de bytes recibidos y se maneja como argumento de una sentencia *if(condición)* para que cada vez que se reciban datos en el socket se ejecute el bloque de instrucciones que se indican dentro de sus llaves. Como la escucha de datos es en modo promiscuo no es necesario proporcionar a la función *recvfrom(int s, void *buf, size_t lon, int banderas, struct sockaddr *origen, socklen_t *longDestino)* los argumentos que indican desde donde se reciben los datos y la longitud de la estructura que indica ese origen y por eso se establecen a un valor de *NULL*. El campo de las banderas se ajusta a un valor de 0 ya que no es necesario indicar alguna de las opciones disponibles para el fin que se busca.

Como parte de las tareas a realizar en caso de la recepción de algún frame está el clasificarlo según su tipo y para ello se revisa inicialmente el valor indicado en el campo *ethertype* del frame, el cual se encuentra en formato de bytes de red. La localización de este campo dentro del frame depende de su clasificación de acuerdo a lo visto en el capítulo *2. Protocolos soportados por el analizador*, y en consecuencia su manejo en la lógica del programa se hace por medio del apuntador *temp2B*.

Si este campo indica un frame de tipo Ethernet II (variable *ether_type* de la estructura *eth* mayor o igual a 1536) se ajusta el valor del apuntador *buff* al byte 14 del arreglo donde se almacena el frame.

```
 buff=&buffEth[14];
```

Y se reposiciona el valor del apuntador eth.

```
eth = (struct ether_header*) buffEth;
```

Enseguida se revisa si el frame contiene un paquete de tipo IP o ARP a través del campo Ethertype con las líneas:

```
if(ntohs(eth->ether_type)==2048)
```

y

```
if(ntohs(eth->ether_type)==2054)
```

respectivamente.

Si el paquete contenido en el frame es IP (ethertype con un valor de 2048), se ajusta el

valor del apuntador *ip* de acuerdo a donde inicie el paquete en el arreglo donde se almacenó el frame.

```
ip = (struct iphdr*) buff;
```

Con la línea siguiente a través del valor de la variable *protocol* de la estructura ip se revisa si el datagrama recibido contiene el protocolo que está indicado a leer según las opciones de ejecución para contabilizarlo por medio de la variable *noFrame*.

```
if(   ((ptcp==1)&&(ip->protocol==6)) ||
      ((pudp==1)&&(ip->protocol==17))||
      ((picmp==1)&&(ip->protocol==1))||
      ((ptcp==0)&&(pudp==0)&&(picmp==0)&&(parp==0))
  )
```

La combinación ptcp==0, pudp==0, picmp==0, parp==0 se da cuando solo se indica la bandera -v en las opciones generales de ejecución.

Si el paquete contenido en el frame es ARP y está indicada la lectura de paquetes ARP del flujo de red:

```
if(   (ntohs(eth->ether_type)==2054) &&
      (  (parp==1)||((ptcp==0)&&(pudp==0)&&(picmp==0)&&(parp==0))  )
   )
```

entonces se ajusta el valor del apuntador *arp* de acuerdo a donde inicie el paquete en el arreglo donde se almacenó el frame.

```
arp = (struct arphdr*) buff;
```

Y se incrementa el valor de la variable que contabiliza los números de frames capturados.

Si el valor de *ether_type* en la estructura *eth* no es mayor o igual a 1536, se revisa si se trata de un frame Ethernet IEEE 802.3 SNAP (campos *DSAP* y *SSAP* con un valor de 0xAA y el campo *Control* con un valor de 0x03) con la línea:

```
if((buffEth[14]==170)&&(buffEth[15]==170)&&(buffEth[16]==3))
```

En caso afirmativo se ajusta el valor del apuntador buff al byte 22 del arreglo donde se almacena el frame ya que el encabezado de este tipo de frame tiene una longitud de 22 bytes. Después se revisa si el frame lleva un paquete IP o ARP pero ahora a través de un apuntador al byte 20 del arreglo que contiene los datos recibidos ya que a partir de allí se encuentra el campo Ethertype en los frames 802.3 SNAP:

```
temp2B=&buffEth[20];
if(ntohs(*temp2B)==2048)      , para paquetes IP, o
if(ntohs(*temp2B)==2054)      , para paquetes ARP.
```

Una vez identificado esto, se realiza un procedimiento parecido al descrito para frames Ethernet II.

Si el frame no cumple tampoco con las normas para ser de tipo Ethernet IEEE 802.3 SNAP se revisa si los elementos con índices 14 y 15 tienen un valor de 0xFF, lo cual indica que se trata de un frame Ethernet Novell RAW 802.3. En caso de que no sea ninguno de los 3 tipos de frame mencionados entonces se trata de un frame Ethernet IEEE 802.3, el cual no está diseñado para transportar paquetes IP o ARP.

Nuevamente con el campo Ethertype manejado a través de *temp2B* se revisa si el protocolo transportado por el frame, ya sea Ethernet II u 802.3 SNAP, es IP. En caso afirmativo se lee el campo del paquete IP que señala el protocolo que transporta a su vez y se indaga si en las opciones de ejecución está indicado capturar paquetes del tipo de protocolo encontrado.

Esta verificación de la siguiente forma:

```
if(   (ip->protocol==6) &&
      ( (ptcp==1)||((ptcp==0)&&(pudp==0)&&(picmp==0)&&(parp==0)) )
   )
```

para el protocolo TCP,

```
if(   (ip->protocol==1) &&
      ( (picmp==1)||((ptcp==0)&&(pudp==0)&&(picmp==0)&&(parp==0)) )
   )
```

para el protocolo ICMP, y

```
if(   (ip->protocol==17) &&
      ( (pudp==1)||((ptcp==0)&&(pudp==0)&&(picmp==0)&&(parp==0)) )
   )
```

para el protocolo UDP.

En caso de que el protocolo transportado por el frame sea ARP también se verifica si está indicado capturar ese tipo de paquetes y para ello se ocupa:

```
if(   (ip->protocol==17) &&
      ( (parp==1)||((ptcp==0)&&(pudp==0)&&(picmp==0)&&(parp==0)) )
   )
```

Para cada una de esas sentencias condicionales, si el resultado es afirmativo se llama a otra función para continuar con el manejo del paquete de forma específica al protocolo transportado.

4.2.1.3 Función int creaFormato(char *archivoOrigen, char *archivoDestino)

La tarea que realiza esta función es determinar el formato en el cual se mostraran los datos del flujo, es decir, determina si los datos del flujo capturado con anterioridad se escribirán en un archivo XML o si se presentarán en pantalla con formato de texto. Dependiendo de esto se llaman a funciones que se encargan de realizar dichas tareas para cada uno de los tipos de paquetes de red identificados. Las variables que ocupa esta función se indican en la tabla 4.3

Variable	Descripción
char *archivoOrigen	Apuntador al nombre del archivo desde el cual se leerán datos en caso de ejecutarse el programa con las banderas -x o -t.
char *archivoDestino	Apuntador que señala al nombre del archivo en el que se guardarán los datos del flujo de red en formato XML.
char *loc	Apuntador de tipo carácter que almacena la cadena que indica la localización establecida del sistema.
FILE *daF, FILE *daX	Descriptores de archivo para los archivos origen y destino respectivamente.
char fecha[11], char hora[13]	Variables donde se almacena la fecha y hora, respectivamente, del frame leído.
char *aptfecha, char *apthora	Apuntadores para manejar la información de los arreglos fecha y hora.
int posicionFinal=0	Se usa para situar el indicador de posición del archivo origen y saber su tamaño.
tamanioTotal	Indica el tamaño en bytes del frame leído.
int contador	Variable de tipo contador para realizar iteraciones.
char paqueteLeido[5]	Variable donde se almacena el tipo de paquete leído (tcp, udp, icmp, arp).
unsigned char buffEth[20000]	Arreglo de tipo carácter para almacenar el frame Ethernet en turno desde el flujo de bytes capturado.
unsigned char *buffeth	Apuntador hacia el primer elemento donde se almacena el frame leído.
int noFrame	Variable para contabilizar el número de frames leídos.
int funPaq	Variable para almacenar el valor devuelto por las funciones para el manejo de los distintos tipos de paquetes de red del flujo leído.

Tabla 4.3. *Variables utilizadas en la función creaFormato().*

La función *fseek(int flujo, long desplazamiento, int posiciónInicial)* se encarga de mover el indicador de posición del archivo apuntado por el descriptor proporcionado. De este modo, con la instrucción:

```
fseek(daF,0,SEEK_END);
```

se mueve el indicador al final del archivo desde el que van a obtener los datos con un desplazamiento de 0 desde dicho punto; y con el código:

```
fseek(daF,0,SEEK_SET);
```

se mueve el indicador de posición al inicio del archivo con un desplazamiento de 0 desde dicho punto para después empezar a leerlo.

Después, se hace una comparación de *archivoDestino* con un espacio en blanco, si hay coincidencia entonces fue indicado la opción -x al programa principal para crear un archivo XML a partir de un archivo con el flujo capturado. Si se cumple esa condición se abre el archivo XML en modo escritura posicionándose el indicador de posición al final, y si no existe se crea:

```
daX=fopen(archivoDestino,"a");
```

Cuando ya se ha escrito el encabezado del archivo XML, a través de un ciclo se empieza a recorrer el archivo del flujo de bytes reconociendo los datos de cada frame. Para cada iteración, en primer lugar se lee la línea que contiene información sobre el frame almacenado y enseguida se escribe esta información con las etiquetas XML respectivas en el archivo destino. Las sentencias correspondientes a esas tareas son:

```
fscanf(daF, "%d %s %s %d %s", &noFrame, aptfecha, apthora, &tamanioTotal,
tipoPaquete);
```

y

```
fprintf(daX,   "<frameEth>\n\t<infoFrame>\n\t\t<numeroF>%d</numeroF>\n\t\t
<fechaF>%s</fechaF>\n\t\t<horaF>%s</horaF>\n\t\t<tamanioF>%d</tamanioF>\n
\t\t<tipoPaquete>%s</tipoPaquete>\n",    noFrame,    aptfecha,    apthora,
tamanioTotal, tipoPaquete)
```

Antes de empezar a leer propiamente los datos del frame se debe recorrer el indicador de posición del archivo un lugar después de la posición actual para pasar a la línea que ya contiene dicha información. Para esto se ocupa la instrucción

```
fseek(daF,1,SEEK_CUR);
```

Mediante el siguiente ciclo:

```
for(contador=0;contador<tamanioTotal;contador++)
{
      buffEth[contador]=fgetc(daF);
      printf("%#x ",buffEth[contador]);
}
```

Se recorre cada byte del frame actual y lo guarda como elemento del arreglo buffEth para después pasar un apuntador a estos datos como parte de los argumentos de las funciones xmlTCP, xmlICMP, xmlUDP y xmlARP según el tipo de paquete de red identificado de cada frame. Estas funciones se encargan de complementar el proceso de escritura de los datos leidos en formato XML de cada frame. Cuando se sale del ciclo que escribe cada frame en formato XML solo resta escribir la etiqueta de cierre principal.

En caso de que no se identifique que se ejecutó el programa con la bandera -x las sentencias que siguen se orientan a presentar en pantalla los datos del archivo origen en formato de texto. En este caso prácticamente se siguen las mismas instrucciones que en el caso de la escritura del archivo XML pero en lugar de ocupar sentencias para escribir en un archivo se utilizan funciones para escribir en pantalla; y en lugar de llamar a las funciones xmlTCP, xmlICMP, xmlUDP y xmlARP para el trato específico de los frames se llaman a las funciones textoTCP, textoICMP, textoUDP y textoARP.

4.2.2 Archivo paquetesTCP.c

Se trata de un código fuente que se integra sólo de una función llamada *paquetesTCP()*, la cual se invoca desde el archivo *Netbender.c* cuando se identifica en el flujo de red leído un paquete TCP, con el propósito de darle el tratamiento adecuado según los argumentos de ejecución recibidos.

4.2.2.1 Función int paquetesTCP(unsigned char *buffEth, unsigned char *buff, char *archivoBytes, int tamEth, char *tiempo, int Tmillitm, int noFrame, int pantalla)

Esta función se encargar de interpretar los bytes del frame capturado que corresponden a un paquete IP y TCP, y guardar en un archivo todos los bytes que componen al frame Ethernet capturado antecedidos de un encabezado descriptivo. Las variables que ocupa esta función se indican en la tabla 4.4.

Variable	Descripción
unsigned char *buffEth	Apuntador al arreglo donde se almacenan los bytes del frame capturado.
unsigned char *buff	Apuntador a la dirección de memoria localizada después de lo que ocupará el encabezado del frame Ethernet en el arreglo buffEth.
char *archivoBytes	Apuntador de tipo carácter que señala al arreglo que contiene el nombre del archivo donde se guardará el flujo de bytes capturado.
int tamEth	Tamaño en bytes del frame Ethernet.
char *tiempo	Apuntador al arreglo donde se almacena la fecha y hora en la que se recibe el frame Ethernet en turno desde el flujo de bytes.
int Tmillitm	Milésimas del segundo en que se capturó el frame.
int noFrame	Número de frame capturado en el flujo de red.
int pantalla	Variable que indica si se debe imprimir en pantalla la información capturada de cada paquete con texto descriptivo.
FILE *daB	Descriptor para el archivo que contiene el flujo de bytes almacenado.
int tamanioTotal	Tamaño total del frame.
unsigned short michecksum	Variable donde se guarda la suma de control calculada.
int contador	Variable de tipo contador para ciclos.

int inicioDatos	Variable donde se almacena el número de byte donde inician los datos del paquete TCP en el frame.
int i	Variable de tipo contador para ciclos.
int opcionesIP=0	Variable que sirve como índice para recorrer los bytes de las opciones del paquete IP en el arreglo que contiene los datos del paquete IP.
int opcionesTCP=0	Variable que sirve como índice para recorrer los bytes de las opciones del paquete TCP en el arreglo que contiene los datos del paquete IP.
int temp	Variable de tipo temporal para varios usos.
unsigned short *temp2B	Variable temporal para almacenar palabras de 2 bytes ocupadas en la estructura de paquetes de red.
unsigned int *temp4B	Variable temporal para almacenar palabras de 4 bytes ocupadas en la estructura de paquetes de red.
struct iphdr *ip	Apuntador a una estructura *iphdr* que contiene como miembros a cada uno de los campos que conforman un paquete IP. Estructura definida en /netinet/ip.h.
struct tcphdr *tcp	Apuntador a una estructura *tcphdr* que contiene como miembros a cada uno de los campos que conforman un paquete TCP. Estructura definida en */netinet/tcp.h*. Esta estructura empieza enseguida del encabezado ip y por eso se define a partir de buff más el tamaño de una estructura de tipo iphdr.

Tabla 4.4. *Variables utilizadas en la función paquetesTCP().*

Con la línea

```
michecksum=sumacontrolIP(ip,20);
```

se calcula la suma de control del paquete IP a través de la función *sumacontrolIP()* que recibe como argumentos un apuntador a la dirección de memoria donde comienza la estructura de tipo *iphdr* que es donde comienzan los datos del encabezado del paquete; y el tamaño en bytes del encabezado que siempre es de 20 bytes. Para la suma de control del paquete TCP se llama a la función *sumacontrolTCP()* que recibe como argumentos el apuntador a la zona de memoria donde comienza el paquete TCP, la longitud del paquete TCP y un apuntador a la dirección de memoria donde comienza el paquete IP.

Una vez impresos los datos de las cabeceras IP y TCP se revisa si el paquete TCP tiene opciones con la sentencia *if(condición)* de la forma siguiente:

```
if(tcp->doff-5>0)
```

ya que *tcp->doff* contiene el valor del campo *offset* del paquete, el cual es un apuntador a la palabra de 32 bits donde comienzan los datos TCP y que tiene un valor de 5 cuando no existen opciones TCP. En caso de que haya opciones se procede a su interpretación y para ello se utiliza un ciclo cuyas instrucciones se ejecutan mientras la variable *opcionesTCP*, que inicialmente se ajusta al número del byte donde comienzan las opciones del paquete, no sobrepase la cantidad de bytes que ocupan la cabecera IP más la cabecera TCP. Lo anterior se define por:

```
opcionesTCP=(ip->ihl+5)*4;
while(opcionesTCP<((ip->ihl)*4+(tcp->doff)*4))
{
      ...
}
```

Dentro de este ciclo las opciones se identifican a través de la función *switch(expresión)* que revisa el tipo de opción indicada por la posición actual en el apuntador *buff* manejado como arreglo y teniendo como índice a la variable *opcionesTCP*. De cada opción se imprime el tipo y su longitud, y una vez dado el manejo adecuado a los datos que contiene se ajusta el valor de la variable *opcionesTCP* para que como índice apunte al elemento del arreglo *buff* donde se inicia la siguiente opción.

Después de la opción 0, que sólo se presenta hasta el final de todas las opciones y que tiene una longitud de un byte, sólo se pueden presentar bytes como relleno para completar la palabra de 32 bits, y para recorrerlos se ocupa un ciclo que itera desde cero hasta la cantidad de bytes restantes de opciones TCP.

```
for(temp=0;temp<((ip->ihl)*4+(tcp->doff)*4)-opcionesTCP;temp++)
{
            printf("%#x",buff[opcionesTCP+temp]);
}
```

En la opción 5 para recorrer los datos extras, en la condición a verificar de la sentencia *for(inicialización, condición, sentencia),* a la variable *temp* se le restan 2 para no contemplar los bytes de tipo y longitud y el resultado se divide entre 8 porque se tiene que tomar en cuenta que los datos se presentan en grupos de 8 bytes (2 campos de 4 bytes).

En la opción 27 para imprimir los datos del campo *QS Nonce (Con corrimiento a la izquierda 2 posiciones)* es necesario una serie de corrimiento de bits porque el campo es de 30 bits y se requiere imprimir byte por byte, bajo la consideración de que el "byte" más significativo es el que estaría incompleto (por ello se completan los 8 bits con ceros a la izquierda); entonces su valor se obtiene en la variable *temp* con la sentencia:

```
temp= (buff[opcionesTCP+4]>>2);
```

Y para los siguientes datos se toman los 6 bits más significativos del octeto en turno y los 2 bits menos significativos del octeto anterior y se hace una operación OR para completar el octeto.

```
temp= (buff[opcionesTCP+4]<<6) | (buff[opcionesTCP+5]>>2);
```

Lo cual se hace hasta recorrer todos los datos; por último se imprime el campo *R* que tiene dos bits de longitud.

Para saber si en el paquete del Protocolo Internet hay opciones, al campo que indica la longitud del encabezado IP se le resta la longitud estándar en palabras de 32 bits que

ocupan los campos de un encabezado IP sin opciones y se revisa si el resultado es mayor a cero.

```
if(ip->ihl-5>0)
```

La variable *opcionesIP* se inicializa a un valor de 20 porque a partir de esa posición en el arreglo que contiene los datos del paquete IP se encuentran las opciones en caso de presentarse.

En la opción IP de tipo 131 se debe considerar que el valor del apuntador, que indica el octeto donde comienza la siguiente dirección origen a procesar, es relativo a esta opción y su menor valor es 4. En consecuencia para mostrar estos datos, el índice del arreglo consiste del valor de la variable *opcionesIP* más el valor del apuntador menos una unidad. Esto se ejemplifica en la figura 4.3.

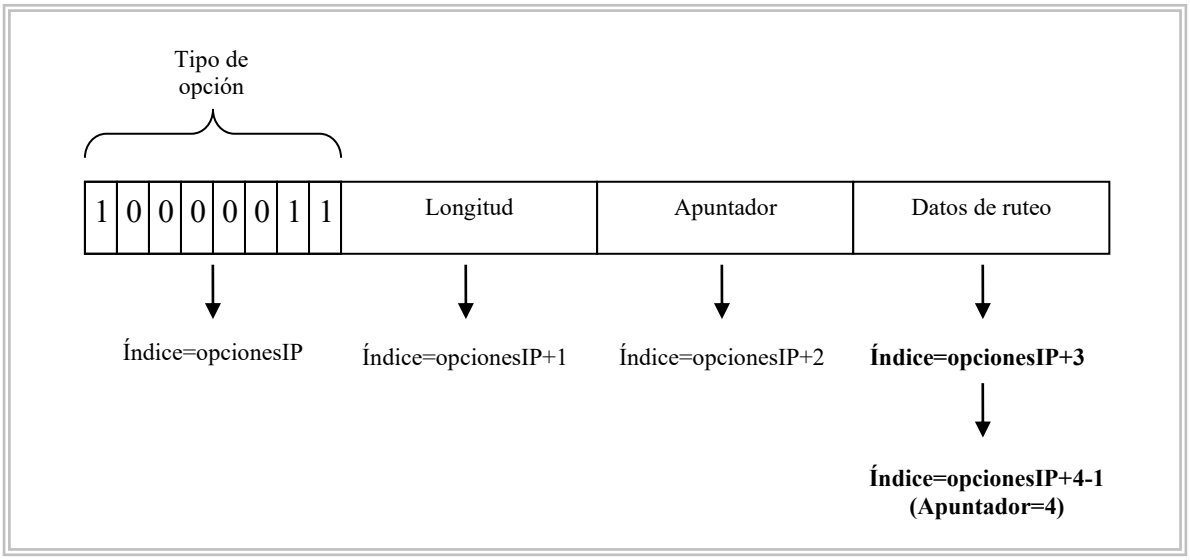

***Figura 4.3**. Ubicación de los datos para la opción IP de tipo 131.*

Para la opción 7 del protocolo Internet, en el ciclo para mostrar las direcciones almacenadas el incremento es de 4 para siempre empezar con el primer octeto de la siguiente dirección IP almacenada. Si no hay direcciones almacenadas entonces el apuntador señala la dirección de memoria donde se debe almacenar la siguiente dirección, la cual es después del apuntador (pointer=4).

Para mostrar los datos de la opción IP de tipo 68 se debe contemplar que estos empiezan a partir del octeto número 5 y que el valor del apuntador es el número de octetos desde el comienzo de esta opción hasta el fin de las marcas de tiempo existentes, y señala al espacio de memoria donde se debe almacenar la dirección IP y después su marca de tiempo o sólo la marca de tiempo, cada una de 4 bytes. En los índices del arreglo buff para mostrar la dirección IP y la marca de tiempo se resta una unidad porque el apuntador es un valor relativo a esta opción. Es decir:

```
      temp4B=&buff[opcionesIP+i-1];
```

y

```
      temp4B=&buff[opcionesIP+i+4-1]; o temp4B=&buff[opcionesIP+i+3];
```

Para la opción IP de tipo 25 se realiza un procedimiento similar que para la opción de tipo 27 del paquete TCP.

Una vez identificadas las opciones IP y TCP se escriben en un archivo todos los bytes que componen el frame Ethernet a través de un ciclo que recorre todo el arreglo *buffEth*. Sin embargo antes de los bytes de frame se escribe una cabecera de información compuesta por el número de frame que ocupa en el flujo de red, la fecha y la hora de su captura, el tamaño total y el tipo de protocolo transportado como datos del paquete IP, es decir, tcp.

4.2.3 Archivo paquetesICMP.c

Este archivo se ejecuta cada vez que se lee del flujo de red un paquete ICMP para manejarlo de acuerdo a su estructura característica. La única función que contiene se llama desde la función *leeflujo()* en el archivo Netbender.c.

4.2.3.1 Función int paquetesICMP(unsigned char *buffEth, unsigned char *buff, char *archivoBytes, int tamEth, char *tiempo, int Tmillitm, int noFrame, int pantalla)

Esta función se encargar de interpretar los bytes del frame capturado que corresponden a un paquete IP e ICMP, además de guardar en un archivo todos los bytes que componen al frame Ethernet capturado antecedidos de un encabezado descriptivo. Las variables que ocupa esta función se muestran en la tabla 4.5.

Variable	Descripción
unsigned char *buffEth	Apuntador al arreglo donde se almacenan los bytes del frame capturado.
unsigned char *buff	Apuntador a la dirección de memoria localizada después de lo que ocupará el encabezado del frame Ethernet en el arreglo buffEth.
char *archivoBytes	Apuntador de tipo carácter que señala al arreglo que contiene el nombre del archivo donde se guardará el flujo de bytes capturado.
int tamEth	Tamaño en bytes del frame Ethernet.
char *tiempo	Apuntador al arreglo donde se almacena la fecha y hora en la que se recibe el frame Ethernet en turno desde el flujo de bytes.
int Tmillitm	Milésimas del segundo en que se capturó el frame.
int noFrame	Número de frame capturado en el flujo de red.
int pantalla	Variable que indica si se debe imprimir en pantalla la información capturada de cada paquete con texto descriptivo.

FILE *daB	Descriptor para el archivo que contiene el flujo de bytes almacenado.
int tamanioTotal	Tamaño total del frame.
unsigned short michecksum	Variable donde se guarda la suma de control calculada.
int contador	Variable de tipo contador para ciclos.
int opcionesIP=0	Variable que sirve como índice para recorrer los bytes de las opciones del paquete IP en el arreglo que contiene los datos del paquete IP.
int temp	Variable de tipo temporal para varios usos.
int i	Variable de tipo contador para ciclos.
unsigned short *temp2B	Variable temporal para almacenar palabras de 2 bytes ocupadas en la estructura de paquetes de red.
unsigned int *temp4B	Variable temporal para almacenar palabras de 4 bytes ocupadas en la estructura de paquetes de red.
struct iphdr *ip	Apuntador a una estructura *iphdr* que contiene como miembros a cada uno de los campos que conforman un paquete IP. Estructura definida en */netinet/ip.h*.
struct icmphdr *icmp	Apuntador a una estructura *icmphdr* que contiene como miembros a cada uno de los campos que conforman un paquete ICMP. Estructura definida en */netinet/ip_icmp.h*. Esta estructura empieza enseguida del encabezado ip y por eso se define a partir de buff más el tamaño de una estructura de tipo iphdr.

Tabla 4.5. *Variables utilizadas en la función paquetesICMP().*

Si el valor de la variable *pantalla* es 1 se imprimen los datos en la salida estándar, primero atendiendo a los del paquete IP y después a los del paquete ICMP. La forma en como se interpreta el paquete IP es la misma en como se hace en la función *paquetesTCP()*.

Para manejar los paquetes ICMP, en caso de que se tenga que imprimir información en pantalla, se revisa mediante la sentencia *switch(expresión)* el valor del campo tipo del paquete:

```
switch(icmp->type)
```

De acuerdo al dato obtenido, el paquete puede tener varios valores posibles para el campo código y en consecuencia se ocupa nuevamente la sentencia *switch(expresión)* para determinarlo. También se leen los demás campos que componen al paquete de acuerdo al tipo identificado y se calcula la suma de control ICMP con la función *sumacontrolICMP()* que recibe como parámetro el apuntador a la dirección de memoria donde inician los datos del paquete y su longitud total.

En algunos paquetes es necesario revisar el campo que inicialmente no tiene un uso específico y para ello se ocupa la variable *temp* como índice del arreglo que contiene los datos del paquete IP. La localización de este campo se obtiene con la sentencia:

```
temp=(((ip->ihl+1)*4));
```

Es decir, en palabras de 32 bits, la longitud del encabezado IP más una palabra que conforman los campos Tipo, Código y Suma de Control, lo cual se multiplica por cuatro para obtener el valor en bytes.

Para leer los datos del campo de *Cabecera de Internet más 64 bits de datos anexados* se hace un posicionamiento parecido y para saber si existen dichos datos en el paquete se hace la comparación siguiente:

```
if((ntohs(ip->tot_len)-(ip->ihl+2)*4)>=28)
```

A la longitud total del paquete IP se le restan los bytes que ocupan el encabezado IP más las dos primeras palabras de 32 bits del encabezado icmp y si este valor es mayor a 28 (cantidad de bytes de la cabecera de internet más 64 bits de datos anexados) significa que sí se presenta ese campo de forma completa. Después se realiza la comparación con 32, 34. 36 y 38 bytes para saber si existen más datos anexos y seguir recorriéndolos. La forma de interpretar dichos datos es como se hace para una cabecera de un paquete IP seguida de una cabecera TCP.
En algunos tipos de paquete ICMP los datos extras no corresponden a la cabecera de internet y los 64 bits de datos anexados sino a datos cuya interpretación depende del datagrama ICMP y para revisar si existe dicha información se hace la comparación:

```
if((ntohs(ip->tot_len)-(ip->ihl+2)*4)>0)
```

y se recorren a través de un ciclo hasta llegar a la longitud total del paquete IP imprimiendo el carácter correspondiente a cada byte o su valor en hexadecimal si no tiene representación ASCII.

```
for(i=temp;i<ntohs(ip->tot_len);i++)
{
	if(isprint(buff[i])!=0)
	{
		printf("%c",buff[i]);
	}
	else
	{
		printf("%#x",buff[i]);
	}
}
```

Habiendo identificado todas las opciones IP e ICMP se escriben en un archivo todos los bytes que componen el frame Ethernet como en el caso de la función *paquetesTCP()*. También se escribe la cabecera compuesta por el número de frame que ocupa en el flujo de red, la fecha y la hora de su captura, el tamaño total y el tipo de protocolo transportado como datos del paquete IP, es decir, icmp.

4.2.4 Archivo paquetesUDP.c

Como su nombre lo indica, este código fuente se refiere a las instrucciones necesarias para tratar un paquete UDP cada vez que se lee uno del flujo de bytes del tráfico de red. Dichas tareas se agrupan en la función denominada *paquetesUDP()*, la cual se describe a continuación.

4.2.4.1 Función int paquetesUDP(unsigned char *buffEth, unsigned char *buff, char *archivoBytes, int tamEth, char *tiempo, int Tmillitm, int noFrame, int pantalla)

Esta función se encarga de interpretar los bytes del frame capturado que corresponden a un paquete IP y UDP, además de guardar en un archivo todos los bytes que componen al frame Ethernet antecedidos de un encabezado descriptivo. Las variables que ocupa se señalan en la tabla 4.6.

Variable	Descripción
unsigned char *buffEth	Apuntador al arreglo donde se almacenan los bytes del frame capturado.
unsigned char *buff	Apuntador a la dirección de memoria localizada después de lo que ocupará el encabezado del frame Ethernet en el arreglo *buffEth*.
char *archivoBytes	Apuntador de tipo carácter que señala al arreglo que contiene el nombre del archivo donde se guardará el flujo de bytes capturado.
int tamEth	Tamaño en bytes del frame Ethernet.
char *tiempo	Apuntador al arreglo donde se almacena la fecha y hora en la que se recibe el frame Ethernet en turno desde el flujo de bytes.
int Tmillitm	Milésimas del segundo en que se capturó el frame.
int noFrame	Número de frame capturado en el flujo de red.
int pantalla	Variable que indica si se debe imprimir en pantalla con texto descriptivo la información capturada de cada paquete.
FILE *daB	Descriptor para el archivo que contiene el flujo de bytes almacenado.
int tamanioTotal	Tamaño total del frame.
unsigned short michecksum	Variable donde se guarda la suma de control calculada.
int contador	Variable de tipo contador para ciclos.
int opcionesIP=0	Variable que sirve como índice para recorrer los bytes de las opciones del paquete IP en el arreglo que contiene los datos del paquete IP.
int temp, temp2	Variables de tipo temporal para varios usos.
int i	Variable de tipo contador para ciclos.
unsigned short *temp2B	Variable temporal para almacenar palabras de 2 bytes ocupadas en la estructura de paquetes de red.

unsigned int *temp4B	Variable temporal para almacenar palabras de 4 bytes ocupadas en la estructura de paquetes de red.
struct iphdr *ip	Apuntador a una estructura *iphdr* que contiene como miembros a cada uno de los campos que conforman un paquete IP. Estructura definida en */netinet/ip.h*.
struct udphdr *udp	Apuntador a una estructura *udphdr* que contiene como miembros a cada uno de los campos que conforman un paquete ICMP. Estructura definida en */netinet/udp.h*. Esta estructura empieza enseguida del encabezado ip y por eso se define a partir de *buff* más el tamaño de una estructura de tipo *iphdr*.

Tabla 4.6. *Variables utilizadas en la función paquetesUDP().*

Al igual que sucede con las funciones paquetesTCP() y paquetesICMP() si está indicado imprimir en pantalla información de los paquetes recibidos, se hace uso de la estructura de tipo *iphdr* definida para los paquetes IP; mientras que para el caso de los paquetes udp se usa una estructura de tipo *udphdr*. Para calcular la suma de control de los paquetes UDP se usa la función sumacontrolUDP() que recibe como argumentos el apuntador a la estructura que contiene los datos del paquetes udp, la longitud del paquete udp, y el apuntador a la estructura de tipo ip.

Para saber si el paquete UDP tiene datos extra se resta a la longitud total del paquete IP la longitud en bytes del encabezado IP y los 8 bytes que ocupa el encabezado de un paquete UDP; si el resultado es mayor a cero entonces por consecuencia existen bytes extra del paquete UDP. En caso de presentarse, estos datos extra se imprimen en formato hexadecimal y en formato carácter a través de ciclos, iniciando desde el byte que se ubica enseguida del encabezado UDP y llegando hasta esa ubicación más la cantidad de bytes de datos extra detectados.

Por último también se lleva a cabo para cada frame detectado la escritura de un encabezado descriptivo y los datos que contiene en un archivo.

4.2.5 Archivo paquetesARP.c

Este archivo contiene código fuente dedicado a interpretar paquetes ARP por medio de la función *paquetesARP()*, la cual es la única de la que se integra.

4.2.5.1 Función int paquetesARP(unsigned char *buffEth, unsigned char *buff, char *archivoBytes, int tamEth, char *tiempo, int Tmillitm, int noFrame, int pantalla)

La labor que hace esta función es reconocer los campos que componen un paquete ARP e imprimir su contenido en pantalla con un formato adecuado e indicando brevemente

a qué se refieren. Además, escribe a un archivo el flujo de bytes del frame ethernet que contiene al paquete ARP en cuestión antecedido por un encabezado con información útil para el tratamiento posterior de dicha secuencia de datos. Las variables que se utilizan en esta función se muestran en la tabla 4.7.

Variable	Descripción
unsigned char *buffEth	Apuntador al arreglo donde se almacenan los bytes del frame capturado.
unsigned char *buff	Apuntador a la dirección de memoria localizada después de lo que ocupará el encabezado del frame Ethernet en el arreglo buffEth.
char *archivoBytes	Apuntador de tipo carácter que señala al arreglo que contiene el nombre del archivo donde se guardará el flujo de bytes capturado.
int tamEth	Tamaño en bytes del frame Ethernet.
char *tiempo	Apuntador al arreglo donde se almacena la fecha y hora en la que se recibe el frame Ethernet en turno desde el flujo de bytes.
int Tmillitm	Milesimas del segundo en que se capturó el frame.
int noFrame	Número de frame capturado en el flujo de red.
int pantalla	Variable que indica si se debe imprimir en pantalla la información capturada de cada paquete con texto descriptivo.
FILE *daB	Descriptor para el archivo que contiene el flujo de bytes almacenado.
int tamanioTotal	Tamaño total del frame.
int contador	Variable de tipo contador para ciclos.
int tamanioArp	Variable que guarda el tamaño del paquete ARP capturado.
int temp	Variable de tipo temporal para varios usos.
int opcionesIP=0	Variable que sirve como índice para recorrer los bytes de las opciones del paquete IP en el arreglo que contiene los datos del paquete IP.
unsigned int *temp4B;	Variable temporal para almacenar palabras de 4 bytes ocupadas en la estructura de paquetes de red.
struct arphdr *arp	Apuntador a una estructura *arphdr* que contiene como miembros a cada uno de los campos que conforman un paquete ARP. Estructura definida en */net/if_arp.h*.

Tabla 4.7. *Variables utilizadas en la función paquetesARP().*

El tamaño de un paquete ARP se obtiene al multiplicar por 2 la suma de la longitud de direcciones de hardware más la longitud de direcciones de protocolo ya que cada uno incluye 2 direcciones de hardware y 2 de protocolo; para después sumar 8, que es la cantidad de bytes que ocupan los demás campos del paquete: *Espacio de direcciones de hardware*=2 bytes, *Espacio de direcciones de protocolo*=2 bytes, *Largo en bytes para cada dirección de hardware*=1 byte, *Largo en bytes para cada dirección de protocolo*=1 byte, *Código de operación*=2 bytes.

Se muestran en pantalla los datos del paquete ARP sólo si está indicado hacerlo (variable *pantalla* con un valor de 1). Para imprimir los bytes de la dirección de hardware origen se leen n bytes enseguida de los 8 bytes que ocupan el encabezado de un paquete ARP, donde n es la longitud en bytes de la dirección de hardware transportada.

```
for(temp=0;temp<arp->ar_hln;temp++)
{
      printf("%x",buff[8+temp]);
      if(temp<arp->ar_hln-1)
      {
            printf(":");
      }
}
```

Si la longitud de cada dirección de protocolo es distinta de 4, se imprime cada byte que la conforma separado de los demás por un guión medio. Los bytes que se toman corresponden a los que se ubican después de los 8 bytes del encabezado arp más los bytes que conforman la dirección de hardware origen.

```
for(temp=0;temp<arp->ar_pln;temp++)
{
      printf("%x-",buff[8+arp->ar_hln+temp]);
}
```

En caso de que la longitud de cada dirección de protocolo sea de 4 bytes, se realiza la impresión de la dirección origen en formato de dirección IP.

```
printf("%s",inet_ntoa(*temp4B));
```

Para mostrar la información de las direcciones destino se realiza algo similar a lo indicado para direcciones origen, modificando solamente la localización de los bytes a imprimir.

Después, para cada frame recibido se escribe en un archivo un encabezado descriptivo, como para los paquetes TCP, ICMP y UDP, seguido de los bytes que lo componen.

4.2.6 Archivos sumacontrolIP.c, sumacontrolTCP.c, sumacontrolICMP.c y Archivo sumacontrolUDP.c

En este apartado se incluye la descripción de las funciones que se encargan de realizar la suma de control (o *checksum*) para cada uno de los paquetes a los cuales se les aplica de acuerdo a la estructura que representa a cada tipo de ellos y que da soporte este software. La suma de control se realiza con el fin de que el analizador de protocolos apunte si existen contrariedades en lo que cada paquete contiene y *lo que dice que contiene*.

4.2.6.1 Función unsigned short sumacontrolIP(void *datosTCP, int tam, void *datosIP)

Función que se localiza en el archivo *sumacontrolIP.c* y que se encarga de calcular la suma de control para un paquete IP. Recibe como parámetros un apuntador a la zona de memoria donde inicia el paquete IP mencionado, y el tamaño del encabezado IP. La tabla 4.8 indica las demás variables que ocupa.

Variable	Descripción
unsigned int suma=0x00000000;	Variable de 4 bytes para almacenar la suma temporal de las palabras de 16 bits.
unsigned short sumactl;	Variable que contiene la suma de control del paquete.
unsigned short *ptr;	Apuntador a la zona de memoria donde inicia el paquete IP.

Tabla 4.8. *Variables locales de la función sumacontrolIP().*

El cálculo se hace a través de un ciclo que recorre en grupos de dos bytes, a través del apuntador *ptr*, los 20 bytes totales de la cabecera del paquete para sumarlos y guardarlos en la variable suma. Al momento de llegar al byte número 10 se suma el valor de cero porque en los bytes 10 y 11 del paquete se encuentra el campo Checksum que se debe considerar con ese valor.

La suma de control se calcula como el complemento a uno de 16 bits de la suma de los complementos a uno de todas las palabras de 16 bits de la cabecera. Lo cual se hace con la siguiente línea de código:

```
sumactl=~(((suma>>16)+(suma & 0xFFFF)) & 0xFFFF);
```

Cada palabra de 16 bits se suma y el resultado se guarda en una palabra de 32 bits para contemplar el acarreo en caso de presentarse, después los primeros 16 bits de este resultado parcial se suman con el acarreo y sólo se toman los primeros 16 bits de la palabra. Por último se obtiene el complemento a uno.

4.2.6.2 Función unsigned short sumacontrolTCP(void *datosTCP, int tam, void *datosIP)

Calcula la suma de control para un paquete TCP y se encuentra en el archivo *sumacontrolTCP.C*. Recibe como argumentos un apuntador a la zona de memoria donde inician los bytes del paquete TCP, el tamaño del paquete TCP, y un apuntador a la zona de memoria donde inicia el paquete IP almacenado. Las variables locales a la función se muestran en la tabla 4.9.

Variable	Descripción
unsigned int suma=0x00000000	Variable de 4 bytes para almacenar la suma temporal de las palabras de 16 bits.
unsigned short sumactl	Variable que contiene la suma de control del paquete.
unsigned short *ptr	Apuntador a la zona de memoria donde inicia el paquete TCP.
int c	Variable que sirve como índice para ciclos.
unsigned short *ptr2	Apuntador a la zona de memoria donde inicia el paquete IP.

Tabla 4.9. *Variables locales de la función sumacontrolTCP().*

El paquete TCP se recorre por grupos de 2 bytes sumando su valor al contenido de la variable suma, excepto cuando se llega al byte 16 que es donde se localiza el campo Checksum y en consecuencia el valor sumado es 0. En caso de que haya una cantidad de octetos impar (el índice c es menor en una unidad al tamaño total) sólo se toman los 8 bits que corresponden a la información correcta de los bytes señalados por el apuntador:

```
suma+=(ntohs(*(ptr++)&(0x00FF)))
```

Después de sumar los datos del encabezado y datos del paquete TCP se suman los bytes de la pseucabecera:

```
suma+=ntohs(*(ptr2+6));
suma+=ntohs(*(ptr2+7));
suma+=ntohs(*(ptr2+8));
suma+=ntohs(*(ptr2+9));
suma+=ntohs(*(ptr2+4))&0x00FF;
suma+=tam;
```

Líneas que corresponden a la suma del primer octeto de la dirección origen, del segundo octeto de la dirección origen, del primer octeto de la dirección destino, del segundo octeto de la dirección destino, del valor del campo protocolo, y de la longitud del paquete TCP, respectivamente.

Después los primeros 16 bits del resultado parcial se suman con el acarreo y sólo se toman los primeros 16 bits de la palabra. Enseguida se obtiene el complemento a uno.

4.2.6.2 Función unsigned short sumacontrolICMP(void *datosICMP, int tam)

Esta función es la única que compone al archivo sumacontrolICMP.c y su tarea es calcular la suma de control de paquetes ICMP. Recibe como argumentos un apuntador al inicio de los datos del paquete ICMP y su tamaño en bytes. Las variables declaradas dentro del cuerpo de la función se describen en la tabla 4.10.

Variable	Descripción
unsigned int suma=0x00000000	Variable de 4 bytes para almacenar la suma temporal de las palabras de 16 bits.
unsigned short sumactl	Variable que contiene la suma de control del paquete.
unsigned short *ptr	Apuntador a la zona de memoria donde inicia el paquete ICMP.
int c	Variable que sirve como índice para ciclos.

Tabla 4.10. *Variables locales utilizadas en la función sumacontrolICMP().*

El cálculo se hace a través de un ciclo que recorre en grupos de dos bytes, a través del apuntador *ptr*, la cantidad total de bytes del paquete para sumarlos y guardarlos en la variable suma. Al momento de llegar al byte número 2 se suma el valor de cero porque en los bytes 2 y 3 del paquete se encuentra el campo *Checksum* que se debe considerar con ese valor.

La suma de control se calcula como el complemento a uno de 16 bits de la suma de los complementos a uno de los bytes que integran el paquete tomado en palabras de 16 bits.

4.2.6.3 Función unsigned short sumacontrolUDP(void *datosUDP, int tam, void *datosIP)

Calcula la suma de control de un paquete UDP. Los parámetros que recibe son un apuntador a la dirección de memoria donde está guardado el paquete UDP, su tamaño en bytes, y un apuntador a la dirección de memoria donde inicia el paquete IP del cual forma parte. La tabla 4.11 señala las variables ocupadas en esta función.

Variable	Descripción
unsigned int suma=0x00000000	Variable de 4 bytes para almacenar la suma temporal de las palabras de 16 bits.
unsigned short sumactl	Variable que contiene la suma de control del paquete.
unsigned short *ptr	Apuntador a la zona de memoria donde inicia el paquete UDP.
unsigned short *ptr2	Apuntador a la zona de memoria donde inicia el paquete IP.

Tabla 4.11. *Variables locales de la función sumacontrolUDP().*

Antes de empezar a sumar los datos del paquete UDP en palabras de 16 bits se suman los campos que componen a la pseudocabecera:

```
suma+=ntohs(*(ptr2+6));
suma+=ntohs(*(ptr2+7));
suma+=ntohs(*(ptr2+8));
suma+=ntohs(*(ptr2+9));
suma+=0x00;
suma+=ntohs(*(ptr2+4))&0x00FF;
suma+=tam;
```

Las 2 primeras líneas para la dirección de red origen, las segundas 2 para la dirección de red destino, después un campo de 8 bits ajustado a un valor de cero, el byte que indica el protocolo y por último el tamaño del paquete UDP. Enseguida se inicia el ciclo para sumar los bytes del paquete en palabras de 2 bytes a excepción del campo *Cheksum* que se encuentra partir el byte número 6. Por último se hace la suma de los primeros 16 bits del resultado parcial más el acarreo y de esto se toman los primeros 16 bits para después hacer el complemento a uno:

```
sumactl=~(((suma>>16)+(suma & 0xFFFF)) & 0xFFFF);
```

Esta función se encuentra en el archivo llamado *sumacontrolUPD.c*.

4.2.7 Archivos xmlTCP.c, xmlICMP.c, xmlUDP.c y xmlARP.c

Estos archivos contienen funciones que se utilizan para crear archivos de tipo XML representativos de un flujo de bytes leido y almacenado previamente. Cada uno de estos archivos se integra por una sola función que se invoca desde el código fuente del archivo *Netbender.c* de acuerdo al tipo de paquete identificado por medio del encabezado correspondiente y que antecede a cada frame; todo esto de acuerdo a lo que se define en la estructura para este tipo de archivo de tipo bitácora.

4.2.7.1 Función int xmlTCP(unsigned char *buffEth, int noFrame, char *aptfecha, char *apthora, int tamanioTotal, char *archivoDestino)

Esta función se encarga de leer los bytes almacenados de un frame desde un archivo, interpretarlos y escribir cada uno de los campos que lo integran en formato XML en un archivo. Específicamente se encarga de escribir frames que contienen paquetes TCP y se encuentra en el código fuente del archivo *xmlTCP.c*. Las variables que ocupa se muestran en la tabla 4.12.

Variable	Descripción
unsigned char *buffet	Apuntador a la dirección de memoria donde han sido guardados los bytes del frame capturado.
int noFrame	Posición que ocupa el frame en el flujo de red.
char *aptfecha	Apuntador a la zona de memoria donde está guardada la fecha de captura del frame.
char *apthora	Apuntador a la zona de memoria donde está guardada la hora de captura del frame.
int tamanioTotal	Tamaño total de frame Ethernet.
char *archivoDestino	Apuntador que indica el nombre del archivo XML donde se deben escribir los datos frame.
char *loc	Apuntador de tipo carácter que almacena la cadena de caracteres que indica la localización establecida del sistema.

FILE *daXML	Descriptor de archivo para manejar el archivo XML destino.
int contador	Variable de tipo índice para recorrer elementos de arreglos.
unsigned char *buff	Apuntador a la dirección de memoria localizada después de lo que ocupará el encabezado del frame Ethernet en el apuntador *buffEth* manejado como arreglo.
struct iphdr *ip	Apuntador a una estructura *iphdr* que contiene como miembros a cada uno de los campos que conforman un paquete IP. Estructura definida en */netinet/ip.h*.
struct tcphdr *tcp	Apuntador a una estructura *tcphdr* que contiene como miembros a cada uno de los campos que conforman un paquete TCP. Estructura definida en */netinet/tcp.h*. Esta estructura empieza enseguida del encabezado ip y por eso se define a partir de *buff* más el tamaño de una estructura de tipo *iphdr*.
struct ether_header *eth	Apuntador a una estructura *ether_header* que contiene como miembros a los campos del encabezado *Ethernet* que ocupan los primeros 14 bytes de un frame de este tipo.
unsigned short *temp2B	Variable temporal para almacenar palabras de 2 bytes ocupadas en la estructura de paquetes de red.
unsigned int *temp4B	Variable temporal para almacenar palabras de 4 bytes ocupadas en la estructura de paquetes de red.
unsigned short michecksum;	Variable donde se guarda la suma de control calculada.
int opcionesIP=0	Variable que sirve como índice para recorrer los bytes de las opciones del paquete IP en el arreglo que contiene los datos del paquete IP.
int opcionesTCP=0	Variable que sirve como índice para recorrer los bytes de las opciones del paquete TCP en el arreglo que contiene los datos del paquete IP.
int temp=0	Variable de tipo temporal para varios usos.
int i	Variable de tipo índice para recorrer arreglos.
int inicioDatos=0	Variable que indica el número de bytes donde inician los datos del paquete TCP.

Tabla 4.12. *Variables utilizadas en la función xmlTCP().*

Se hace una identificación del tipo de frame ethernet que se está leyendo tal cual se hizo en el archivo Netbender.c en la función leeFlujo(), pero ya no se revisa si el tipo de protocolo de red que contiene es ARP. Los datos que componen el encabezado del frame Ethernet identificado no se imprimen con una descripción en texto de su contenido si no encerrados entre la etiquetas XML correspondiente al tipo de información.

Después se verifica que se trate de un paquete IP que contiene un paquete TCP con la línea:

```
if((ntohs(*temp2B)==2048)&&(ip->protocol==6))
```

Las acciones que refieren a la impresión de la información contenida en los paquetes IP y

TCP se realizan de forma similar a como fue señalado en la función paquetesTCP() con la diferencia que la salida no es la pantalla sino el archivo indicado por el descriptor *daXML* a través de la función *fprintf(FILE *flujo, const char *formato, ...)* y que no hay descripción en formato de texto de cada campo sino por medio de etiquetas XML.

Para presentar los datos del paquete TCP se comprueba si el byte tiene una representación ASCII imprimible. En caso afirmativo se revisa si el byte en cuestión corresponde a un carácter no permitido como elemento de texto para las etiquetas de tipo ELEMENT en la sintaxis de documentos XML (", &, ', <, >) y entonces imprimir la secuencia de escape correspondiente (", &, ', < y >); si el byte no tiene su representación en texto de cualquiera de esos caracteres se imprime normalmente. Si el byte no tiene representación de carácter ASCII se imprime su código hexadecimal.

4.2.7.2 Función int xmlICMP(unsigned char *buffEth, int noFrame, char *aptfecha, char *apthora, int tamanioTotal, char *archivoDestino)

Función localizada en el archivo *xmlICMP.c* y que se encarga de leer desde un archivo los bytes de un frame que contiene un paquete IP y un paquete ICMP, los interpreta y escribe cada uno de los campos que lo integran en formato XML en otro archivo. En la tabla 4.13 se describen las variables que ocupa.

Variable	Descripción
unsigned char *buffEth	Apuntador a la dirección de memoria donde han sido guardados los bytes del frame capturado.
int noFrame	Posición que ocupa el frame en el flujo de red.
char *aptfecha	Apuntador a la zona de memoria donde está guardada la fecha de captura del frame.
char *apthora	Apuntador a la zona de memoria donde está guardada la hora de captura del frame.
int tamanioTotal	Tamaño total de frame Ethernet.
char *archivoDestino	Apuntador que indica el nombre del archivo XML donde se deben escribir los datos frame.
FILE *daXML	Descriptor de archivo para manejar el archivo XML destino.
int contador	Variable de tipo índice para recorrer elementos de arreglos.
unsigned char *buff	Apuntador a la dirección de memoria localizada después de lo que ocupará el encabezado del frame Ethernet en el apuntador *buffEth* manejado como arreglo.
unsigned short michecksum	Variable donde se guarda la suma de control calculada.
int opcionesIP=0	Variable que sirve como índice para recorrer los bytes de las opciones del paquete IP en el arreglo que contiene los datos del paquete IP.
unsigned short *temp2B	Variable temporal para almacenar palabras de 2 bytes ocupadas en la estructura de paquetes de red.

unsigned int *temp4B	Variable temporal para almacenar palabras de 4 bytes ocupadas en la estructura de paquetes de red.
struct ether_header *eth	Apuntador a una estructura *ether_header* que contiene como miembros a los campos del encabezado Ethernet que ocupan los primeros 14 bytes de un frame de este tipo. Estructura definida en */net/ethernet.h*.
struct iphdr *ip	Apuntador a una estructura *iphdr* que contiene como miembros a cada uno de los campos que conforman un paquete IP. Estructura definida en */netinet/ip.h*.
struct icmphdr *icmp	Apuntador a una estructura *icmphdr* que contiene como miembros a cada uno de los campos que conforman un paquete ICMP.
int temp=0	Variable de tipo temporal para varios usos.
int i	Variable de tipo índice para recorrer arreglos

Tabla 4.13. *Variables utilizadas en la función xmlICMP()*

Para la identificación del tipo de frame Ethernet se realiza el mismo procedimiento que el de la función xmlTCP().

Las acciones que refieren a la impresión de la información contenida en los paquetes IP e ICMP se realizan de forma similar a como fue señalado en la función paquetesICMP() con la diferencia que la salida no es la pantalla sino el archivo indicado por el descriptor *daXML* a través de la función *fprintf(FILE *flujo, const char *formato, ...)* y que la descripción de cada campo no es en formato de texto simple sino en formato XML. Para escribir los datos de texto de etiquetas XML de tipo ELEMENT para alguna de las opciones ICMP, si alguno de los caracteres no tiene representación ASCII se escribe la representación hexadecimal del byte. En caso contrario se escribe el carácter correspondiente a menos de que sea un carácter no permitido para el contenido de texto de esa etiqueta (", &, ', <, >), y en cuyo caso se escribe la secuencia de escape correspondiente.

4.2.7.3 Función int xmlUDP(unsigned char *buffEth, int noFrame, char *aptfecha, char *apthora, int tamanioTotal, char *archivoDestino)

Se localiza en el archivo *xmlUDP.c* diseñado para esta apliación. Se encarga de leer desde un archivo los bytes de un frame que contiene un paquete IP y un paquete UDP, lo interpreta y escribe cada uno de los campos que lo integran en formato XML en otro archivo. Las variables que recibe como argumentos y las que se definen dentro del cuerpo de la función son las que se indican en la tabla 4.14.

Variable	Descripción
unsigned char *buffEth	Apuntador a la dirección de memoria donde han sido guardados los bytes del frame capturado.
int noFrame	Posición que ocupa el frame en el flujo de red.

char *aptfecha	Apuntador a la zona de memoria donde está guardada la fecha de captura del frame.
char *apthora	Apuntador a la zona de memoria donde está guardada la hora de captura del frame.
int tamanioTotal	Tamaño total de frame Ethernet.
char *archivoDestino	Apuntador que indica el nombre del archivo XML donde se deben escribir los datos frame.
char *loc	Apuntador a la cadena de caracteres que indica la localización establecida del sistema.
FILE *daXML	Descriptor de archivo para manejar el archivo XML destino.
int contador	Variable de tipo índice para recorrer elementos de arreglos.
unsigned char *buff	Apuntador a la dirección de memoria localizada después de lo que ocupará el encabezado del frame Ethernet en el apuntador *buffEth* manejado como arreglo.
unsigned short michecksum	Variable donde se guarda la suma de control calculada.
int opcionesIP=0	Variable que sirve como índice para recorrer los bytes de las opciones del paquete IP en el arreglo que contiene los datos del paquete IP.
int temp=0	Variable de tipo temporal para varios usos.
int i	Variable de tipo índice para recorrer arreglos.
int temp2=0	Variable de tipo temporal para varios usos.
unsigned short *temp2B	Variable temporal para almacenar palabras de 2 bytes ocupadas en la estructura de paquetes de red.
unsigned int *temp4B	Variable temporal para almacenar palabras de 4 bytes ocupadas en la estructura de paquetes de red.
struct ether_header *eth	Apuntador a una estructura *ether_header* que contiene como miembros a los campos del encabezado Ethernet que ocupan los primeros 14 bytes de un frame de este tipo.
struct iphdr *ip	Apuntador a una estructura *iphdr* que contiene como miembros a cada uno de los campos que conforman un paquete IP. Estructura definida en */netinet/ip.h*.
struct udphdr *udp	Apuntador a una estructura *udphdr* que contiene como miembros a cada uno de los campos que conforman un paquete UDP.

Tabla 4.14. *Variables utilizadas en la función xmlUDP().*

Se hace la identificación del tipo de frame Ethernet tal como en la función xmlTCP(). Las acciones que refieren a la impresión de la información de los paquetes IP y UDP se realizan de forma similar a como fue señalado en la función paquetesTCP() con la diferencia que la salida no es la pantalla sino el archivo indicado por el descriptor *daXML* a través de la función fprintf() y que no hay descripción en formato de texto de cada campo sino por medio de etiquetas XML.

Para escribir los datos extras de paquetes UDP en formato XML se revisa si alguno de los caracteres no tiene representación ASCII para escribir la representación hexadecimal del byte. En caso contrario se escribe el carácter correspondiente a menos de que sea un carácter no permitido para contenido de texto de esa etiqueta (", &, ', <, >), y en cuyo caso se escribe la secuencia de escape correspondiente.

4.2.7.4 Función int xmlARP(unsigned char *buffEth, int noFrame, char *aptfecha, char *apthora, int tamanioTotal, char *archivoDestino)

Función que se encarga de leer desde un archivo los bytes de un frame que contiene un paquete IP y un paquete UDP, los interpreta y escribe cada uno de los campos que lo integran en formato XML en otro archivo. Las variables que ocupa se describen en la tabla 4.15.

Variable	Descripción
unsigned char *buffEth	Apuntador a la dirección de memoria donde han sido guardados los bytes del frame capturado.
int noFrame	Posición que ocupa el frame en el flujo de red.
char *aptfecha	Apuntador a la zona de memoria donde está guardada la fecha de captura del frame.
char *apthora	Apuntador a la zona de memoria donde está guardada la hora de captura del frame.
int tamanioTotal	Tamaño total de frame Ethernet.
char *archivoDestino	Apuntador que indica el nombre del archivo XML donde se deben escribir los datos frame.
char *loc	Apuntador que señala la cadena de caracteres que indica la localización establecida del sistema.
FILE *daXML	Descriptor de archivo para manejar el archivo XML destino.
int temp=0	Variable de tipo temporal para varios usos.
unsigned short *temp2B	Variable temporal para almacenar palabras de 2 bytes ocupadas en la estructura de paquetes de red.
unsigned int *temp4B	Variable temporal para almacenar palabras de 4 bytes ocupadas en la estructura de paquetes de red.
unsigned char *buff	Apuntador a la dirección de memoria localizada después de lo que ocupará el encabezado del frame Ethernet en el apuntador *buffEth* manejado como arreglo.
struct ether_header *eth	Apuntador a una estructura *ether_header* que contiene como miembros a los campos del encabezado Ethernet que ocupan los primeros 14 bytes de un frame de este tipo.
struct arphdr *arphdr	Apuntador a una estructura *arphdr* que contiene como miembros a cada uno de los campos que conforman un paquete ARP.

Tabla 4.15. *Variables utilizadas en la función xmlARP().*

Se hace la identificación del tipo de frame Ethernet tal como en la función xmlTCP(). Las acciones que refieren a la impresión de la información de los paquetes ARP se realizan de forma similar a como fue señalado en la función paquetesARP() con la diferencia que la salida no es la pantalla sino el archivo indicado por el descriptor *daXML* a través de la función *fprintf(FILE *flujo, const char *formato, ...)* y que la de cada campo no es en formato de texto sino por medio de etiquetas XML.

4.2.8 Archivos textoTCP.c, textoICMP.c, textoUDP.c, textoARP.c

Estos archivos contienen funciones cuyo objetivo es leer los datos de frames capturados previamente en un archivo y escribir información de cada uno en la pantalla.

4.2.8.1 Funciones textoTCP(), textoICMP(), textoUDP(), textoARP()

Para realizar las tareas de lectura y escritura en pantalla de los datos mencionados, se realiza un procedimiento es muy similar al de las funciones xmlTCP(), xmlICMP(), xmlUDP() y xmlARP(), sólo que los datos no se imprimen con etiquetas XML sino con un formato parecido al presentado cuando se muestran los datos al momento de captura del frame con algunos datos extras. Además ya no es necesario mostrar secuencias de escape para los casos donde se tenga que presentar el texto correspondiente a algunos bytes, sólo se verifica si existe una representación de caracter ASCII para imprimirlos tal cual o sino sólo mostrar su valor hexadecimal.

Las definiciones de las funciones que realizan estas labores son:

```
int textoTCP(unsigned char *buffEth, int noFrame, char *aptfecha, char
*apthora, int tamanioTotal);

int textoICMP(unsigned char *buffEth, int noFrame, char *aptfecha,
char *apthora, int tamanioTotal);

int textoUDP(unsigned char *buffEth, int noFrame, char *aptfecha, char
*apthora, int tamanioTotal);

int textoARP(unsigned char *buffEth, int noFrame, char *aptfecha, char
*apthora, int tamanioTotal);
```

Siendo invocadas desde el archivo *Netbender.c* y en específico desde la función *creaFormato(char *archivoOrigen, char *archivoDestino)*. Cada una actúa según el tipo de paquete leído desde el archivo fuente, *textoTCP()* para paquetes TCP, *textoICMP()* para paquetes ICMP, *textoUDP()* para paquetes UDP y *textoARP()* para paquetesARP. Las variables son las referidas por la tabla 4.16.

Variable	Descripción
unsigned char *buffEth	Apuntador a la dirección de memoria donde han sido guardados los bytes del frame capturado.
int noFrame	Posición que ocupa el frame en el flujo de red.
char *aptfecha	Apuntador a la zona de memoria donde está guardada la fecha de captura del frame.
char *apthora	Apuntador a la zona de memoria donde está guardada la hora de captura del frame.
int tamanioTotal	Tamaño total del frame Ethernet.
char *archivoDestino	Apuntador que indica el nombre del archivo XML donde se deben escribir los datos frame.

Tabla 4.16. *Variables utilizadas en la funciones textoTCP(), textoICMP(), textoUDP(), textoARP()*

4.2.9 Archivos de encabezado

Los siguientes archivos tipo *header* (ver tabla 4.17) contienen los prototipos para las diferentes funciones en lenguaje C realizadas para el analizador de protocolos.

Archivo de tipo encabezado	Descripción
paquetesTCP.h	Prototipo para la función *paquetesTCP()*
paquetesICMP.h	Prototipo para la función *paquetesICMP()*
paquetesUDP.h	Prototipo para la función *paquetesUDP()*
paquetesARP.h	Prototipo para la función *paquetesARP()*
xmlTCP.h	Prototipo para la función *xmlTCP()*
xmlICMP.h	Prototipo para la función *xmlICMP()*
xmlUDP.h	Prototipo para la función *xmlUDP()*
xmlARP.h	Prototipo para la función *xmlARP()*
textoTCP.h	Prototipo para la función *textoTCP()*
textoICMP.h	Prototipo para la función *textoICMP()*
textoUDP.h	Prototipo para la función *textoUDP()*
textoARP.h	Prototipo para la función *textoARP()*
sumacontrolIP.h	Prototipo para la funcion *sumacontrolIP()*
sumacontrolTCP.h	Prototipo para la funcion *sumacontrolTCP()*
sumacontrolICMP.h	Prototipo para la funcion *sumacontrolICMP()*
sumacontrolUDP.h	Prototipo para la funcion *sumacontrolUDP()*

Tabla 4.17. *Archivos de tipo encabezado realizados para el analizador de protocolos*

4.3 Descripción del código fuente analizaFlujo.java

A continuación se presentan extractos de la documentación generada con la herramienta *javadoc* de Sun Microsystems para el código *analizaFlujo.java*.
La clase analizaFlujo genera estadísticas a partir de un archivo en formato XML que se apega a la sintaxis marcada en el Document Type Definition llamado frame.dtd perteneciente a este conjunto de herramientas (Netbender y analizaFlujo) que conforman el analizador de protocolos.

4.3.1 Detalles del constructor analizaFlujo

```
public analizaFlujo(java.lang.String archivo)
             throws java.io.FileNotFoundException
```

Construye un objeto analizaFlujo e inicializa el atributo *archXML* con el archivo XML a analizar y el atributo *texto* para almacenar un flujo de entrada de máximo 55 caracteres, esto es porque de acuerdo a lo definido en el archivo frame.dtd una linea del archivo XML no puede exceder esa cantidad. Añade al final de los ArrayList *indiceMaxOrigen* e *indiceMaxDestino* el elemento 0.

Parameters:

`archivo` - La ruta del archivo XML a analizar.

Throws:

`java.io.FileNotFoundException` - Si el archivo a leer no existe en la ruta especificada.

4.3.2 Detalles de los métodos

Para realizar operaciones sobre el objeto analizaFlujo se diseñaron 3 métodos, incluyendo al método main; estos métodos llevan por nombre una descripción de la tarea que realizan. En seguida se describe cada uno de ellos.

4.3.2.1 Método parseaFlujo

```
public void parseaFlujo()
                throws java.io.IOException
```

Lee línea por línea el archivo XML y de cada una de ellas a través de expresiones regulares identifica las etiquetas correspondientes a la información de la cual se quiere presentar estadísticas.

Para datos como cantidad de frames, cantidad de paquetes ARP, IP, TCP, UDP e ICMP, cantidad de paquetes con opciones (IP, TCP e ICMP), cantidad de paquetes UDP con datos extras y cantidad de paquetes (IP, TCP, ICMP y UDP) con errores, sólo basta con reconocer cada que se presenta la etiqueta adecuada e incrementar la variable contador correspondiente.

En el caso de las direcciones origen (Ethernet e IP), destina un ArrayList para almacenar cada una de ellas previamente habiendo identificado si la dupla no ha sido registrada o en caso contrario sólo incrementa el elemento de otro ArrayList que corresponde a dicha dupla y que señala cuantas veces se ha presentado; para el caso de direcciones destino se realiza el mismo procedimiento.

Throws:

`java.io.IOException` - Si una excepción de Entrada/Salida ocurre al intentar leer una línea del archivo XML.

4.3.2.2 Método imprimeEstadisticas

```
public void imprimeEstadisticas()
                        throws java.io.IOException
```

Imprime en pantalla la lista de direcciones origen y destino, las direcciones origen y destino que más ocurren, información sobre direcciones Ethernet e IP, e información sobre los paquetes de red.

Obtiene las duplas de direcciones que más ocurren, haciendo un recorrido de las listas que las almacenan y si su correspondiente marcador de ocurrencias es mayor al almacenado de otra dupla, el ArrayList que contiene los índices de las duplas que más se repiten se vacía y se añade el índice de la dupla identificada; si el marcador de ocurrencias es igual al de las duplas almacenadas sólo se añade a la lista el índice de la nueva dupla.

Throws:

`java.io.IOException` - Si una excepción de Entrada/Salida ocurre al intentar cerrar el buffer de lectura para líneas del archivo XML.

4.3.2.3 Método main

```
public static void main(java.lang.String[] args)
```

Recibe como único argumento la ruta del archivo XML del cual se van a generar las estadísticas del flujo de red.

Parameters:

`args` - Argumentos pasados desde el shell.

Realizar esta programación ha sido una tarea larga y en algunos casos complicado sobre todo por encontrar, manipular, y en algunos casos diseñar, las funciones adecuadas para llevar a cabo los procedimientos de acuerdo a lo que se pretende. Queda claro que para comprender la estructura y el funcionamiento de toda la herramienta no basta con sólo leer este capítulo (y el siguiente), sino que es necesario revisar detenidamente el código proporcionado para conocer como opera cada uno de los archivos ejecutables.

Es importante hacer notar que cada código fuente en lenguaje C está relacionado con los demás, sin embargo la labor que hace cada uno está claramente definida, iniciando por el nombre que se les da, razón por la cual es fácil encontrar y modificar las instrucciones en caso de ser necesario y para tareas que no tengan que ver con los formatos predefinidos para la generación de los archivos que almacenan el flujo de bytes capturados o el archivo XML, lo cual afectaría directamente a la lógica de análisis y conllevaría cambiar también la forma de interpretar ese tipo de bitácoras para las tareas establecidas.

Capítulo 5
Instalación y forma de uso

5.1 Instalación del analizador de protocolos

El software final es proporcionado en una carpeta encapsulada y comprimida bajo el nombre de *analizadorNetbender-1.0.tar.gz*. En consecuencia, para su instalación, el primer paso es descomprimir el archivo, en este caso se ejemplifica con el comando de shell *gunzip*:

[root@celeste local]# gunzip analizadorNetbender-1.0.tar.gz

y después desencapsularlo con:

[root@celeste local]# tar -xvf analizadorNetbender-1.0.tar

O hacer ambas tareas con:

[root@celeste local]# tar -zxvf analizadorNetbender-1.0.tar.gz

Como resultado se obtiene una carpeta llamada *analizadorNetbender-1.0*, en la cual se encuentra un archivo llamado *Makefile* que contiene las opciones de compilación e instalación del programa. Las líneas que se pueden modificar son:

dirDestino = /bin
dirManDestino8 = /usr/local/man/man8
dirManDestino5 = /usr/local/man/man5

Las cuales indican la ruta de instalación del archivo ejecutable *Netbender* y la de los archivos de tipo *manpage* (páginas de manual) *Netbender.8*, *analizaFlujo.8* y *frame.dtd.5*.

Para iniciar la compilación basta con ejecutar el comando *make* desde esa carpeta y si no hay algún error en la salida lo siguiente es ejecutar el comando *make* otra vez pero con la opción *install*. Las líneas que resumen esto son:

[root@celeste analizadorNetbender-1.0]# make

y

[root@celeste analizadorNetbender-1.0]# make install

Dentro del directorio *analizadorNetbender-1.0* se encuentra la carpeta *codigos_java* que contiene el código *analizaFlujo.java*, el archivo de bytecode *analizaFlujo.class*, y una carpeta con la documentación generada a través de javadoc de esta clase java. Como el bytecode de java es independiente de la plataforma donde se ejecute gracias a la máquina virtual no es necesario recompilar el código pero puede hacerse si así se desea de la siguiente forma:

[root@celeste codigos_java]# javac analizaFlujo.java

5.2 Uso del analizador de protocolos

Para conocer la forma de uso de los programas *Netbender* y *analizaFlujo* se pueden revisar las páginas de manual respectivas provistas junto con los archivos ejecutables desde el intérprete de comandos:

[root@celeste analizadorNetbender-1.0]# man Netbender

y

[root@celeste analizadorNetbender-1.0]# man analizaFlujo

Las cuales también se presentan como los puntos 10 y 11 de la sección de anexos.

5.3 Ejemplo de ejecución y salida

Se pretende capturar en modo promiscuo a través de la tarjeta de red eth1 el flujo concordante con los protocolos TCP y ARP, presentando en pantalla una salida inmediata; así como almacenar el flujo en un archivo llamado flujo00. Así, la primer forma de ejecutar el programa es:

[root@celeste analizadorNetbender-1.0]# Netbender -m P -i eth1 -p tcp arp -v -f flujo00

De la salida en pantalla se muestran a continuación las líneas que indican información de los 3 primeros frames.

```
Número de frame: 1      Fecha y hora: 21/08/2008 17:13:21.423
Ethernet II
Dirección Ethernet Origen: 0:50:da:59:20:6e     Dirección Ethernet Destino: 0:c:db:ac:1c:0
Protocolo (Ethertype): 0x800

>>> Datos del paquete IP <<<                    >>> Datos del paquete TCP <<<
Versión IP: 4                                   Puerto de origen: 22
IHL: 5 (palabras de 32 bits)                    Puerto de destino: 33787
Tipo de servicio: 0                             Número de secuencia:   4252576793
Longitud Total: 48 (palabras de 8 bits)         Número de acuse:    461521013
Identificador del paquete: 0            Posición de los datos (palabras de 32 bits): 7
Banderas: DF=1, MF=0                            URG: 0, ACK: 1, PSH: 0
Posición del fragmento: 0                       RST: 0, SYN: 1, FIN: 0
Tiempo de vida: 64                              CWR: 0, ECN-ECHO: 0
Valor del campo "protocolo": 6                  Tamaño de la ventana: 5840
Valor del checksum: 0x3ac1 (0x3ac1)             Valor del checksum: 0x629c (0x629c)
Dirección fuente: 132.248.59.3                  Puntero de urgencia: 0
Dirección Destino: 58.97.5.171

--> Opciones TCP: 2 palabras de 32 bits <--
->Tipo de opción: 2: Máximo tamaño de segmento
        Longitud de la opción: 4
        Datos de la opción: 1460
->Tipo de opción: 1: Sin operación
        Longitud de la opción: 1
->Tipo de opción: 1: Sin operación
        Longitud de la opción: 1
->Tipo de opción: 4: Permiso para selección de acuses
        Longitud de la opción: 2

--> Paquete IP sin opciones <--

*******************************************   Fin de los datos del paquete
*******************************************

Número de frame: 2      Fecha y hora: 21/08/2008 17:13:21.430
Ethernet II
Dirección Ethernet Origen: 0:50:da:59:20:6e     Dirección Ethernet Destino: 0:c:db:ac:1c:0
Protocolo (Ethertype): 0x800

>>> Datos del paquete IP <<<                    >>> Datos del paquete TCP <<<
Versión IP: 4                                   Puerto de origen: 22
IHL: 5 (palabras de 32 bits)                    Puerto de destino: 33787
Tipo de servicio: 0                             Número de secuencia:   4252576793
Longitud Total: 48 (palabras de 8 bits)         Número de acuse:    461521013
Identificador del paquete: 0            Posición de los datos (palabras de 32 bits): 7
Banderas: DF=1, MF=0                            URG: 0, ACK: 1, PSH: 0
Posición del fragmento: 0                       RST: 0, SYN: 1, FIN: 0
Tiempo de vida: 64                              CWR: 0, ECN-ECHO: 0
```

```
Valor del campo "protocolo": 6                  Tamaño de la ventana: 5840
Valor del checksum: 0x3ac1 (0x3ac1)             Valor del checksum: 0x629c (0x629c)
Dirección fuente: 132.248.59.3                  Puntero de urgencia: 0
Dirección Destino: 58.97.5.171

--> Opciones TCP: 2 palabras de 32 bits <--
->Tipo de opción: 2: Máximo tamaño de segmento
        Longitud de la opción: 4
        Datos de la opción: 1460
->Tipo de opción: 1: Sin operación
        Longitud de la opción: 1
->Tipo de opción: 1: Sin operación
        Longitud de la opción: 1
->Tipo de opción: 4: Permiso para selección de acuses
        Longitud de la opción: 2

--> Paquete IP sin opciones <--

*****************************************   Fin de los datos del paquete
*****************************************

Número de frame: 3      Fecha y hora: 21/08/2008 17:13:21.453
Ethernet II
Dirección Ethernet Origen: 0:c:db:ac:1c:0       Dirección Ethernet Destino: 0:f:fe:b1:67:b9
Protocolo (Ethertype): 0x800

>>> Datos del paquete IP <<<                    >>> Datos del paquete TCP <<<
Versión IP: 4                                   Puerto de origen: 80
IHL: 5 (palabras de 32 bits)                    Puerto de destino: 1036
Tipo de servicio: 0                             Número de secuencia:   2267864298
Longitud Total: 1500 (palabras de 8 bits)               Número de acuse:   1865956366
Identificador del paquete: 59707                Posición de los datos (palabras de 32 bits):
5
Banderas: DF=1, MF=0                            URG: 0, ACK: 1, PSH: 0
Posición del fragmento: 0                       RST: 0, SYN: 0, FIN: 0
Tiempo de vida: 50                              CWR: 0, ECN-ECHO: 0
Valor del campo "protocolo": 6                  Tamaño de la ventana: 6432
Valor del checksum: 0xda16 (0xda16)             Valor del checksum: 0xcf7a (0xcf7a)
Dirección fuente: 90.183.101.10                 Puntero de urgencia: 0
Dirección Destino: 132.248.59.16

--> Paquete TCP sin opciones <--

--> Paquete IP sin opciones <--

--> Los datos enviados en formato de caracter son:
.2..A}..!....^..5...r`B.&KQ.._N..KT....x...L...Et..v.;Z.1.....Qo.~.\y.............|..>p1.h..
e.t...'qP.Q..1..... ~K.......o5;S.fX...8...-
.K...$.I._.7%y....y+!..n..c..........T...f..7N.9.8u=C..a......._S..........IF,..?....G..K...
B].@..M..K.V...=J.L!..Lm[..N........<..L<.....I.KH..2.z...h)
Xf."..v.n..y/...w.V.....!..P.C..._....s.@U.Y..0*...D..mz.0s.v.Y.....6.&f...1...d.......rj
......`ll.G........".C..3...Y'.!Y8C..h1.("D...A..=....+...C......&hr....y<Gv._.<r...!..)....
...........;...v..2.......^.........N.nJ.\.r.x.}f0...a.\p7.r{...|.3.....i.....#.U......^....
......]zxk2....lgBO.l....N....0..:.".RF...j....kI>2..~..r=..`.s..H..1j*....q..Z\.f.iE...u...
....V')..(.y...Jp..b.....H./.<%.t.Z.....t*...j..e0.TR.......C..D.s@......!A..cR....#h...&...
X.B!.ds.f....cw.7..LuxXM.)....ZpQ...g.faQw....H.5.T77.86.+.6.......M....;r.z.N.D.E+..ok....w
.<..kX...T....uK......zkg.......%...?.Y..v.bi..
,...$..G..g..?....d)..qoL..;..2&T.r.mC...?.$r.mu)..u.l..[..~.;dm....."........kr.1......"..g6
\...&...EE.:H'.~hZ...@,..2...#..h..=...R.\'.@O.V...d7y$.{..."...}.UU=7m7...U...T..........YJ.
.n.5....K.;..L.J..h.......1)...e1MP..9..2..t2.G.T..K..uptt5"h......}4Xk[of.(.$i....y.:....L.
1\..f.....v|.....5..i....:..*-r.....n(...z....._n...4O1.u-
Xf.6..9...S&.......p....L.0.n......qRJ=Udc...\z....dx.<).Cw....N......3._{).......^...].C...
....K........._]MQ.....(>.l.hV..>.F.....&.h.f?Y....'.1.!...P~..]`.&...0....g.*.<..D....c...u
.Gj.4i.U.....o^u...2....qe.0X-.G.G_i.......$..a.@......
*****************************************   Fin de los datos del paquete
*****************************************
```

Al leer el contenido del archivo flujo00 con el editor de texto *vi* se pueden notar los encabezados creados para cada frame capturado (resaltados en negritas en esta ocasión) seguidos de los bytes que lo integran. De este archivo se presentan las líneas correspondientes también a los 3 primeros frames:

1 21/08/2008 17:13:21.423 62 tcp
^@^LÛ¬^\^@^@PÚY
n^H^@E^@^@0^@^@@^@@^F:Á<84>ø;^C:a^E«^@^V<83>ûýy,^Y^[<82>@up^R^VÐb<9c>^@^@^B^D^E´^A^A^D^B**2**
21/08/2008 17:13:21.430 62 tcp
^@^LÛ¬^\^@^@PÚY
n^H^@E^@^@0^@^@@^@@^F:Á<84>ø;^C:a^E«^@^V<83>ûýy,^Y^[<82>@up^R^VÐb<9c>^@^@^B^D^E´^A^A^D^B**3**
21/08/2008 17:13:21.453 1514 tcp
^@^Oþ±g¹^@^LÛ¬^\^@^H^@E^@^EÜé;@^@2^FÚ^VZ·e
<84>ø;^P^@P^D^L<87>,Ûêo8<^NP^P^Y Ïz^@^@Á2ÐßA}õë!ç^U«ä^ã´5õ^F¬r`B<85>&KQà^G_N-
^WKT<8f>í·^]xÂ²ÿLµ»<8d>Et^V^BvÃ;ZÓ1^O<9d>Ú<8c>øQo^Y~^S\y^Kßí<99>^^<89>þîá¸^^^?<85>|µÓ>p1³hä¥
e¯téý³'qP£Q<83><8b>1là¸<8c>^Q ~KÑÝØ^K¶þÅo5;SÿfX<8a>õú8°¦<97>-^\K°^P^^$¡I^D_ö7%yÆ^T<8e>
y+!Å^[nÇc¨¬^^ª¥þæÌ÷<9e>TÞ<80>°f®¯7N<9e>9£8u=C¡<91>aòÄÈ<99>Ùò¡_SÂÀ
ª^UË¯¨<8d>ÅIF,Á<90>?<9c>´<94>óGß<85>K<95>^@úB]<84>@ïÐM
ÊK¶Vª^_<9b>=J¬L!ÛúLm[ÿ<96>N<99>^^<96>«<92>¯ôÁ<ÆËL<Æ^^¬½ÕIæKHÃ<98>2òz°
·h) Xfí"êÄvîn^Fðy/^QÁæw<91>VÇ»^T<9e>²!ãùP CÞëÓ_^G×<93>¢sÞ@U<86>YÕó0*û^SÙDß<8b>mz<9f>0s
v°Y^T<92>Ô°Ë6Ð&fÌÇ1ü^\ûdâª¿×£<9f>^]rj
¦Ñ×áÄý`ll^CG¸^V^\ëâÌóá"^?Cµ^U3âÍ½Y'ù!Y8CÃ¢h1<8c>("D^MÈ^HA^A¢=ê<99><8d>å+-
<95>®C¿ß¤Þú¸&hrç¶<81>Ëy<Gv<90>_<93><r^QÀÓ!<92>ä)<8e>
ÈÐ^[<96>Û¦<9c>É^Dç^X¦£;¥^C^Wv<96>õ2<88>^D<89>¶îÜÇ^ñØð<87>¼ä^C^C¨N÷nJ \¾räx^\}f0Ò»§a
\p7¦r{<84><98>Ù|<9d>3Æ<96>^S^]Èi^SÏ¼µÛ#^FU¡³ô^CË^<92>¾<8d><92>^[Å^R¾-
<90>]zxk2<9b>Åú<96>lgBOÅl.Í¬µN<9c>ûå<8f>0<9e><84>:Õ"÷RF<86>^C^TjÚ<9a>ÒõkI>2<9b>þ~þ<8a>r=^^<9
6>`ãsÊ<93>H¦¬1j*ÝÅ^N<87>q^R^]Z\øf^YiE¼®òu^Ký<92>Ö§<84>¡V')<8a>^N(<8f>y¨é<88>Jp<90>ôbì
£^DþH^M/ñ<%^FtìZ<95>^^<93>-
<9a>t*«Ò^_j<88>öe0ñTR°Å<90>Çª±«C<91><9c>D<80>s@^N^NÇ^U^Tá!A^^<84>cRíùùç#h^B¬Þ&ÈÍ^AXïB!<81>ds
^Bfñ<9b>^^¤cw³7ãôLuxXMâ)Î^]^G<82>ZpQÍ^Lòg<9b>faQw<8d>¼È<93>H°5<9e>T77Þ86ª+<89>6<8b>^W¢^E·^T<
88>M<81>½±´;r¨z<86>NæD^LE+^Fäok¥¯<94><8b>wå<åñkXý^^©T<95>é-
·uK¼¸^F¹<9f>^?zkg<9b>Ð÷^O¾<9d>^_%^M^Q^_?åY<84><94>vÞbi¾<83>
,Ôã<91>$ËÔGØ¦g¶^R?<88>Ýð^^d)^UÚqoL<8d><9b>;×
2&TÌrämC<8e>Ä<94>?É$rämu)^ZÜuØlÊÕ[^H^C~à;dm
Ö¬ö<9e>"x^Eô<9c>ñ<90>^Zkr<8e>1<86>è^Y
ã×"¶^Vg6\^@<80>^S&ÇÞ^NEE×:H'<9b>~hZ^_<9a><88>@,<93>
2^Kø#×©hô^E=ò<99>ÛRµ\'§@O
V¹^P^Td7y$^L{<99>¾<9e>"±ÂÀ}<91>UU=7m7½Û^VUù<91>¥T<93>ÔµÌ¸<91>Á^LÅ^HYJï^Kn<9f>5½®<93>ÍK°;<94>
ÝLªJÞÃhÈñ¨^K²Í<8d>1)Ò<82>Þe1MPïÐ9©<99>2^G<94>t2åG
T×ÔK¥<84>uptt5"h×<86><98>^@¢Õ}4Xk[ofÖ(¸$i§´
^^yö:ûÐþ¡LÈ1\ÝÍf<88>^Lµþ¡v|êÖ^Z^\æ5ñÿiÆ<94><99>¬:¼¯*-r^WÇ<96>ØØn(¬¸Îz^QÙÂê²_n<8f>»õ4O1úu-
Xf.6²Ç9<8b>¡ÏS&^@¿<96>ÊÛ<83>^?p^C<92><8d><94>LÅ0<8e>n¢<81>¬¢<8d><8d>qRJ=Udc^LÒ©\z^Z^^<9b>Ädx
û<)úCwå Þ¦N<93>Áë®Ç^L3á_{)«<88>^E^RÍø®^Ö ê]<94>CïÌ¿ù<95>ìªK<9c>Ð¬á<93>^?<9d>ü<81>_]MQ^Cüø^Yà
(>^Rl¥hV^X¬><9e>Fý^TÑ^^^A&óh·f?Y©<80>¬<80>'¼1^]!<8c><9b>¸P~³<9f>]`û&Ñ<80>²0^Pª«<87>gí*<9a><à
´DÑÖ<81>^^cÝË^Fu¢Gj<87>4ißUæ<86><93>ì^No^u^ËëÛ2<8d>·£^XqeÞ0X-
°G<99>G_iÅ^S<9b>ÑØ<80><83>$½Ùaÿ@°ÔÒ<97>À^Q

Cabe recordar que no todos los bytes tienen una representación de carácter ASCII y que la salida anterior corresponde a una forma propia de interpretar esos bytes por el editor de texto empleado.

Siguiendo con este ejemplo, para crear un archivo xml que tenga la representación de los datos capturados en el archivo flujo00 se ejecuta la siguiente línea desde el shell del sistema:

[root@celeste analizadorNetbender-1.0]# Netbender -x flujo00 flujo00.xml

El código XML generado correspondiente a los 2 primeros frames es el siguiente:

```
<!-- flujo00.xml -->
<!DOCTYPE person SYSTEM "frame.dtd">
<flujo>
<frameEth>
        <infoFrame>
                <numeroF>1</numeroF>
                <fechaF>21/08/2008</fechaF>
                <horaF>17:13:21.423</horaF>
                <tamanioF>62</tamanioF>
                <tipoPaquete>tcp</tipoPaquete>
                <tipoF>Ethernet II</tipoF>
        </infoFrame>
        <encabezadoFrame>
                <dirEthOrigen>0:50:da:59:20:6e</dirEthOrigen>
                <dirEthDestino>0:c:db:ac:1c:0</dirEthDestino>
                <ethertype>0x800</ethertype>
        </encabezadoFrame>
        <datosFrame>
                <paqIP>
                        <encabezadoIP>
                                <versionIP>4</versionIP>
                                <IHL>5 (palabras de 32 bits)</IHL>
                                <tipoServ>0</tipoServ>
                                <longTotalIP>48 (palabras de 8 bits)</longTotalIP>
                                <idPaq>0</idPaq>
                                <bandDF>1</bandDF>
                                <bandMF>0</bandMF>
                                <posFrag>0</posFrag>
                                <TTL>64</TTL>
                                <protocol>6</protocol>
                                <FCSIP>0x3ac1 (0x3ac1)</FCSIP>
                                <dirIPOrigen>132.248.59.3</dirIPOrigen>
                                <dirIPDestino>58.97.5.171</dirIPDestino>
                        </encabezadoIP>
                        <paqTCP>
                                <encabezadoTCP>
                                        <puertoOrigTCP>22</puertoOrigTCP>
                                        <puertoDestTCP>33787</puertoDestTCP>
                                        <numSec>-42390503</numSec>
                                        <numAcuse>461521013</numAcuse>
                                        <posicionDat>Palabra de 32 bits no: 7</posicionDat>
                                        <bandURG>0</bandURG>
                                        <bandACK>1</bandACK>
                                        <bandPSH>0</bandPSH>
                                        <bandRST>0</bandRST>
                                        <bandSYN>1</bandSYN>
                                        <bandFIN>0</bandFIN>
                                        <bandCRW>0</bandCRW>
                                        <bandECN-ECHO>0</bandECN-ECHO>
                                        <tamVent>5840</tamVent>
                                        <FCSTCP>0x629c (0x629c)</FCSTCP>
                                        <puntUrg>0</puntUrg>
                                </encabezadoTCP>
                                <opcionesTCP>
                                        <pal32Bits>2 palabras de 32 bits</pal32Bits>
                                        <tipoOpc>2: Máximo tamaño de segmento</tipoOpc>
                                        <longOpc>4</longOpc>
                                        <opTCP2>
                                                <datosDeOpcion>1460</datosDeOpcion>
                                        </opTCP2>
                                        <tipoOpc>1: Sin operación</tipoOpc>
                                        <longOpc>1</longOpc>
                                        <tipoOpc>1: Sin operación</tipoOpc>
                                        <longOpc>1</longOpc>
                                        <tipoOpc>4: Permiso para selección de
acuses</tipoOpc>
                                        <longOpc>2</longOpc>
                                </opcionesTCP>
                        </paqTCP>
                </paqIP>
        </datosFrame>
```

```
</frameEth>
<frameEth>
        <infoFrame>
                <numeroF>2</numeroF>
                <fechaF>21/08/2008</fechaF>
                <horaF>17:13:21.430</horaF>
                <tamanioF>62</tamanioF>
                <tipoPaquete>tcp</tipoPaquete>
                <tipoF>Ethernet II</tipoF>
        </infoFrame>
        <encabezadoFrame>
                <dirEthOrigen>0:50:da:59:20:6e</dirEthOrigen>
                <dirEthDestino>0:c:db:ac:1c:0</dirEthDestino>
                <ethertype>0x800</ethertype>
        </encabezadoFrame>
        <datosFrame>
                <paqIP>
                        <encabezadoIP>
                                <versionIP>4</versionIP>
                                <IHL>5 (palabras de 32 bits)</IHL>
                                <tipoServ>0</tipoServ>
                                <longTotalIP>48 (palabras de 8 bits)</longTotalIP>
                                <idPaq>0</idPaq>
                                <bandDF>1</bandDF>
                                <bandMF>0</bandMF>
                                <posFrag>0</posFrag>
                                <TTL>64</TTL>
                                <protocol>6</protocol>
                                <FCSIP>0x3ac1 (0x3ac1)</FCSIP>
                                <dirIPOrigen>132.248.59.3</dirIPOrigen>
                                <dirIPDestino>58.97.5.171</dirIPDestino>
                        </encabezadoIP>
                        <paqTCP>
                                <encabezadoTCP>
                                        <puertoOrigTCP>22</puertoOrigTCP>
                                        <puertoDestTCP>33787</puertoDestTCP>
                                        <numSec>-42390503</numSec>
                                        <numAcuse>461521013</numAcuse>
                                        <posicionDat>Palabra de 32 bits no: 7</posicionDat>
                                        <bandURG>0</bandURG>
                                        <bandACK>1</bandACK>
                                        <bandPSH>0</bandPSH>
                                        <bandRST>0</bandRST>
                                        <bandSYN>1</bandSYN>
                                        <bandFIN>0</bandFIN>
                                        <bandCRW>0</bandCRW>
                                        <bandECN-ECHO>0</bandECN-ECHO>
                                        <tamVent>5840</tamVent>
                                        <FCSTCP>0x629c (0x629c)</FCSTCP>
                                        <puntUrg>0</puntUrg>
                                </encabezadoTCP>
                                <opcionesTCP>
                                        <pal32Bits>2 palabras de 32 bits</pal32Bits>
                                        <tipoOpc>2: Máximo tamaño de segmento</tipoOpc>
                                        <longOpc>4</longOpc>
                                        <opTCP2>
                                                <datosDeOpcion>1460</datosDeOpcion>
                                        </opTCP2>
                                        <tipoOpc>1: Sin operación</tipoOpc>
                                        <longOpc>1</longOpc>
                                        <tipoOpc>1: Sin operación</tipoOpc>
                                        <longOpc>1</longOpc>
                                        <tipoOpc>4: Permiso para selección de
acuses</tipoOpc>
                                        <longOpc>2</longOpc>
                                </opcionesTCP>
                        </paqTCP>
                </paqIP>
        </datosFrame>
</frameEth>
```

Si se deseara obtener nuevamente una salida en formato de texto del flujo de red previamente capturado basta con ejecutar el programana Netbender de la siguiente forma:

[root@celeste analizadorNetbender-1.0]# Netbender -t flujo00

Y con esto la salida en pantalla, de los primeros 3 frames, es:

```
Número de frame: 1      Fecha y hora: 21/08/2008 17:13:21.423   Tamaño del frame: 62    Tipo
de paquete: tcp
Ethernet II
Dirección Ethernet Origen: 0:50:da:59:20:6e     Dirección Ethernet Destino: 0:c:db:ac:1c:0
Protocolo (Ethertype): 0x800

>>> Datos del paquete IP <<<                    >>> Datos del paquete TCP <<<
Versión IP: 4                                   Puerto de origen: 22
IHL: 5 (palabras de 32 bits)                    Puerto de destino: 33787
Tipo de servicio: 0                             Número de secuencia: -42390503
Longitud Total: 48 (palabras de 8 bits)         Número de acuse: 461521013
Identificador del paquete: 0            Posición de los datos (palabras de 32 bits): 7
Banderas: DF=1, MF=0                            URG: 0, ACK: 1, PSH: 0
Posición del fragmento: 0                       RST: 0, SYN: 1, FIN: 0
Tiempo de vida: 64                              CWR: 0, ECN-ECHO: 0
Valor del campo "protocolo": 6                  Tamaño de la ventana: 5840
Valor del checksum: 0x3ac1 (0x3ac1)             Valor del checksum: 0x629c (0x629c)
Dirección fuente: 132.248.59.3                  Puntero de urgencia: 0
Dirección Destino: 58.97.5.171

--> Opciones TCP: 2 palabras de 32 bits <--
->Tipo de opción: 2: Máximo tamaño de segmento
        Longitud de la opción: 4
        Datos de la opción: 1460
->Tipo de opción: 1: Sin operación
        Longitud de la opción: 1
->Tipo de opción: 1: Sin operación
        Longitud de la opción: 1
->Tipo de opción: 4: Permiso para selección de acuses
        Longitud de la opción: 2

--> Paquete IP sin opciones <--

*****************************************   Fin de los datos del paquete
*****************************************

Número de frame: 2      Fecha y hora: 21/08/2008 17:13:21.430   Tamaño del frame: 62    Tipo
de paquete: tcp
Ethernet II
Dirección Ethernet Origen: 0:50:da:59:20:6e     Dirección Ethernet Destino: 0:c:db:ac:1c:0
Protocolo (Ethertype): 0x800

>>> Datos del paquete IP <<<                    >>> Datos del paquete TCP <<<
Versión IP: 4                                   Puerto de origen: 22
IHL: 5 (palabras de 32 bits)                    Puerto de destino: 33787
Tipo de servicio: 0                             Número de secuencia: -42390503
Longitud Total: 48 (palabras de 8 bits)         Número de acuse: 461521013
Identificador del paquete: 0            Posición de los datos (palabras de 32 bits): 7
Banderas: DF=1, MF=0                            URG: 0, ACK: 1, PSH: 0
Posición del fragmento: 0                       RST: 0, SYN: 1, FIN: 0
Tiempo de vida: 64                              CWR: 0, ECN-ECHO: 0
Valor del campo "protocolo": 6                  Tamaño de la ventana: 5840
Valor del checksum: 0x3ac1 (0x3ac1)             Valor del checksum: 0x629c (0x629c)
Dirección fuente: 132.248.59.3                  Puntero de urgencia: 0
Dirección Destino: 58.97.5.171

--> Opciones TCP: 2 palabras de 32 bits <--
->Tipo de opción: 2: Máximo tamaño de segmento
        Longitud de la opción: 4
```

```
        Datos de la opción: 1460
->Tipo de opción: 1: Sin operación
        Longitud de la opción: 1
->Tipo de opción: 1: Sin operación
        Longitud de la opción: 1
->Tipo de opción: 4: Permiso para selección de acuses
        Longitud de la opción: 2

--> Paquete IP sin opciones <--

*****************************************   Fin de los datos del paquete
*****************************************

Número de frame: 3      Fecha y hora: 21/08/2008 17:13:21.453   Tamaño del frame: 1514  Tipo
de paquete: tcp
Ethernet II
Dirección Ethernet Origen: 0:c:db:ac:1c:0       Dirección Ethernet Destino: 0:f:fe:b1:67:b9
Protocolo (Ethertype): 0x800

>>> Datos del paquete IP <<<                    >>> Datos del paquete TCP <<<
Versión IP: 4                                   Puerto de origen: 80
IHL: 5 (palabras de 32 bits)                    Puerto de destino: 1036
Tipo de servicio: 0                             Número de secuencia: -2027102998
Longitud Total: 1500 (palabras de 8 bits)               Número de acuse: 1865956366
Identificador del paquete: 59707                Posición de los datos (palabras de 32 bits):
5
Banderas: DF=1, MF=0                            URG: 0, ACK: 1, PSH: 0
Posición del fragmento: 0                       RST: 0, SYN: 0, FIN: 0
Tiempo de vida: 50                              CWR: 0, ECN-ECHO: 0
Valor del campo "protocolo": 6                  Tamaño de la ventana: 6432
Valor del checksum: 0xda16 (0xda16)             Valor del checksum: 0xcf7a (0xcf7a)
Dirección fuente: 90.183.101.10                 Puntero de urgencia: 0
Dirección Destino: 132.248.59.16

--> Paquete TCP sin opciones <--

--> Paquete IP sin opciones <--

--> Los datos enviados en formato de caracter son:
.2..A}..!....^..5...r`B.&KQ.._N..KT....x...L...Et..v.;Z.1.....Qo.~.\y.............|..>p1.h..
e.t...'qP.Q..1..... ~K.......o5;S.fX...8...-
.K...$.I._.7%y....y+!..n..c..........T...f..7N.9.8u=C..a......._S..........IF,..?....G..K...
B].@..M..K.V...=J.L!..Lm[..N........<..L<.....I.KH..2.z...h)
Xf."..v.n..y/...w.V.....!..P.C..._....s.@U.Y..0*...D..mz.0s.v.Y.....6.&f...1...d.......rj
......`ll.G........".C..3...Y'.!Y8C..h1.("D...A..=....+...C......&hr....y<Gv._.<r...!..)....
..........;...v..2.......^.........N.nJ.\.r.x.}f0...a.\p7.r{...|.3.....i.....#.U......^....
......]zxk2....lgBO.l....N....0..:.".RF...j....kI>2..~..r=..`.s..H..1j*....q..Z\.f.iE...u...
....V')..(.y...Jp..b.....H./.<%.t.Z.....t*...j..e0.TR.......C..D.s@......!A..cR....#h...&...
X.B!.ds.f....cw.7..LuxXM.)....ZpQ...g.faQw....H.5.T77.86.+.6.......M....;r.z.N.D.E+..ok....w
.<..kX...T....uK......zkg.......%...?.Y..v.bi..
,...$..G..g..?....d)..qoL..;..2&T.r.mC...?.$r.mu)..u.l..[..~.;dm....."........kr.1......"..g6
\...&...EE.:H'.~hZ...@,..2..#..h..=...R.\'.@O.V...d7y$.{..."...}.UU=7m7...U...T..........YJ.
.n.5....K.;..L.J..h.......1)...e1MP..9..2..t2.G.T..K..uptt5"h......}4Xk[of.(.$i....y.:....L.
1\..f.....v|.....5..i....:..*-r.....n(...z....._n...4O1.u-
Xf.6..9...S&.......p....L.0.n......qRJ=Udc...\z....dx.<).Cw....N......3._{)........^...].C...
....K.........]MQ.....(>.l.hV..>.F.....&.h.f?Y....'.1.!...P~..]`.&...0....g.*.<..D....c...u
.Gj.4i.U.....o^u...2....qe.0X-.G.G_i.......$..a.@......
*****************************************   Fin de los datos del paquete
*****************************************
```

Ahora, para obtener un resumen estadístico del flujo de red capturado a partir del archivo flujo00.xml se ejecuta el programa analizaFlujo de la forma:

[root@celeste codigos_java]# java analizaFlujo flujo00.xml

Ésto bajo la consideración de que el archivo XML a procesar se encuentra en el mismo directorio que el archivo de tipo bytecode analizaFlujo.class. La salida en pantalla en este caso es la siguiente:

```
>>> Lista de direcciones origen:
        Ethernet origen: 0:50:da:59:20:6e e IP origen: 132.248.59.3 se presentan 2
ocasion(es)
        Ethernet origen: 0:c:db:ac:1c:0 e IP origen: 90.183.101.10 se presentan 4
ocasion(es)
        Ethernet origen: 0:f:fe:b1:67:b9 e IP origen: 132.248.59.16 se presentan 1
ocasion(es)

>>> Lista de direcciones destino:
        Ethernet destino: 0:c:db:ac:1c:0 e IP destino: 58.97.5.171 se presentan 2
ocasion(es)
        Ethernet destino: 0:f:fe:b1:67:b9 e IP destino: 132.248.59.16 se presentan 4
ocasion(es)
        Ethernet destino: 0:c:db:ac:1c:0 e IP destino: 90.183.101.10 se presentan 1
ocasion(es)

>>> Las direcciones origen que más ocurren son:
        Ethernet origen: 0:c:db:ac:1c:0 e IP origen: 90.183.101.10 se presentan 4
ocasion(es)

>>> Las direcciones destino que más ocurren son:
        Ethernet destino: 0:f:fe:b1:67:b9 e IP destino: 132.248.59.16 se presentan 4
ocasion(es)

>>> Resumen sobre direcciones Ethernet e IP:
        Cantidad de direcciones Ethernet origen: 3
        Cantidad de direcciones Ethernet destino: 3
        Cantidad de direcciones IP origen: 3
        Cantidad de direcciones IP destino: 3

>>> Resumen sobre paquetes de red:
        Cantidad total de paquetes: 7
        Cantidad de paquetes ARP: 0
        Cantidad de paquetes IP: 7
        Cantidad de paquetes TCP: 7
        Cantidad de paquetes ICMP: 0
        Cantidad de paquetes UDP: 0
        Cantidad de paquetes con opciones IP: 0
        Cantidad de paquetes con opciones TCP: 2
        Cantidad de paquetes con opciones ICMP: 0
        Cantidad de paquetes UDP con datos extra: 0
        Cantidad de paquetes IP con errores: 0
        Cantidad de paquetes TCP con errores: 0
        Cantidad de paquetes ICMP con errores: 0
        Cantidad de paquetes UDP con errores: 0
```

Capítulo 6
Mantenimiento y expectativas a futuro

6.1 Consideraciones para el uso del analizador de protocolos

Existen 3 aspectos esenciales a considerar sobre el uso del analizador de protocolos y que el usuario debería conocer para utilizar eficientemente esta herramienta de software. Estos requerimientos no son cuestiones difíciles de conocer ya que en cuanto cuestiones técnicas esta información se adquiere durante el curso de la carrera de Ingeniería en Computación. A continuación se detalla cada uno de estos puntos.

6.1.1 Sobre el sistema operativo

El analizador de protocoles está diseñado para ejecutarse en un sistema operativo GNU/LINUX, así que el usuario deberá estar familiarizado con su uso por lo menos en lo que refiere a tareas básicas realizadas a través del intérprete de comandos, como por ejemplo:

- Conocimiento de la estructura de directorios y desplazamiento a través de ella.
- Creación, copia, movimiento, borrado y edición de archivos de tipo regular y de tipo directorio.
- Asignación de permisos de ejecución.
- Compresión y descompresión de archivos.
- Configuración básica de los dispositivos de red.

Además de esto, deberá haber leído las páginas de manual proporcionadas.

6.1.2 Sobre la edición de códigos fuente y compilación

La programación del analizador de protocolos está enfocada a ser simple y explicativa por sí misma para que los usuarios sean capaces de modificar o agregar bloques funcionales siguiendo la lógica presentada de acuerdo a sus necesidades. Sin embargo, es necesario que cuenten con conocimientos del lenguaje de programación utilizado. Para el código en lenguaje C se debe saber el uso de:

- Sentencias de control de flujo (*if*, *else*, *for*, *while*, *switch*).
- Arreglos y apuntadores.
- Funciones desde un mismo y desde distintos archivos.
- Funciones básicas para cadenas.
- Funciones de manipulación de archivos.
- Datos de tipo estructura.

Mientras que para modificar el código en lenguaje Java es necesario conocer bases de la programación orientada a objetos y cuestiones sencillas del API de Java, como el manejo de excepciones y algunos métodos simples para el manejo de expresiones regulares.

Para realizar la compilación como es indicado en el capítulo anterior es necesario tener instalado la utilidad *make* de GNU que sirve para controlar la generación de archivos ejecutables, y el compilador *javac* del JDK que corresponda al API 1.5 o superior.

6.1.3 Sobre la información generada

Si se desea leer el contenido de los archivos que contienen flujos de bytes capturados se debe tener en cuenta que el editor de texto va a mostrar una representación propia para los bytes almacenados que no tengan un carácter ASCII asociado a su valor. Además, si un archivo XML de los generados a partir de un flujo de bytes capturado previamente es de un tamaño mayor a 8 Megabytes no es recomendable intentar ver su contenido a través de un explorador web, a menos que se tenga un equipo con muy buenas características de hardware (sobre todo memoria RAM y bus de datos). Por otra parte, es recomendable colocar el archivo *frame.dtd* proporcionado como parte del analizador de

protocolos, o una copia de él, en el directorio donde se encuentre el archivo XML a visualizar a través de un explorador web, ya que algunos de ellos necesariamente lo requieren para mostrar el contenido de este tipo de archivos.

En cuestiones generales es claro que mientras mejores sean las características del procesador mejores serán los resultados en lo que refiere a tiempos de ejecución.

6.2 Estado actual de la aplicación

Desde el punto de vista que corresponde al desarrollo del tema de tesis el analizador de protocolos se puede considerar una aplicación completa debido a que cumple con los requisitos establecidos para este trabajo. Los programas elaborados como parte de la aplicación se pueden considerar una fuente de información muy útil para la comprensión de la estructura de los frames Ethernet y de los paquetes de red que contienen. Además de la claridad conseguida sobre este mismo aspecto con las diferentes salidas generadas.

Sin embargo, si se toma desde otro enfoque, el propio de una herramienta de administración de red, se tiene una aplicación que cumple sólo con aspectos básicos e incluso se queda corta por la cantidad de protocolos que puede interpretar en comparación con la variedad de protocolos que se presentan en una red de área local.

6.3 Propuestas de puntos por mejorar

Existen muchos aspectos para mejorar del analizador de protocolos, que van desde lo simple hasta los que presentan ciertas dificultades en cuanto a tiempo de desarrollo. Iniciando con los puntos básicos a mejorar se puede mencionar una reorganización del formato de texto para la salida en pantalla con el fin de dar una mejor apariencia.

Por otra parte, se pueden corregir algunos términos empleados para expresar los datos, ya que en algunas ocasiones se hace la traducción de las frases desde el inglés al español intentando adecuarla al entorno en el que se enmarca, lo que no garantiza la total correspondencia a la frase adecuada. En otros casos se conserva el nombre en inglés tal cual se señala en el RFC donde se explica.

De forma similar se puede cambiar la forma de presentar la información para algunas opciones de paquetes para que dicho dato sea más fácil de interpretar, en específico aquellos datos que se componen por varios bytes y de los que se imprime su valor en formato hexadecimal byte por byte o al contrario, todos los bytes a través de un solo valor, y cuya razón para hacerlo es solamente el sentido común de acuerdo al tipo de campo. Para hacer esto es necesario revisar a fondo lo expuesto en los RFC's adecuados y confirmar o corregir el manejo dado para estos campos.

También es posible reducir más el código fuente mediante la inclusión de tareas similares en una función que, para los casos necesarios, marque las diferencias pertinentes. En caso concreto, el código que maneja la información de las opciones de los paquetes IP para impresión en pantalla de información o creación de archivos con extensión XML y que se presenta en varios archivos se puede presentar en una sola función para referirla en cada caso.

Para obtener estadísticas del flujo de red es necesario generar previamente el archivo XML ya que con su estructura se facilita dicha tarea; sin embargo, sería conveniente obtener los datos que nos interesan a partir de la lectura del archivo que contiene el flujo de bytes conforme se va interpretando como cuando se genera el código XML o como cuando se imprime en pantalla en formato de texto la interpretación de cada campo de los frames, y con ello evitar el uso de memoria secundaria para almacenar archivos.

Como ya se ha mencionado, los programas que componen el analizador de protocolos cumplen cabalmente con lo planeado, sin embargo, esto no quiere decir que no tenga aspectos que mejorar y sobre todo añadir. Lo que salta a la vista en primer instancia es la generación de una Interfaz Gráfica de Usuario, o *GUI* por sus siglas en inglés y con ello tener un ambiente más amigable de ejecución, pero no sólo eso sino también con ello modificar la forma en como se realizan algunas de las tareas mencionadas. Como se encuentra actualmente la aplicación es necesario valerse de editores de texto o exploradores web para ver el contenido de los diferentes tipos de archivos, pero con la generación de una GUI propia se podría tener el entorno para visualizar este tipo de información, y en el caso de los archivos XML que presentan cierta dificultad para revisarse si su tamaño es muy grande, se podría idear una forma a través de la interfaz gráfica para leerlo poco a poco y por partes, con lo que no sería necesario cargar en memoria primaria toda la fuente de información de un solo momento sino hacerlo de manera progresiva según se vaya requiriendo para mostrar en pantalla. Otro de los beneficios que se podría tener con esa interfaz sería un mecanismo para reordenar los datos que al usuario le interese ver, por ejemplo, todos los paquetes de una dirección origen en específico, todos los paquetes que conformen un flujo en particular identificado a través de números de secuencia, los paquetes que tengan un error en su estructura o los frames que cumplan con cualquier otro filtro especificado.

Conclusiones

La arquitectura de red TCP/IP es la más aceptada y usada para la implementación de redes de computadoras debido a que proporciona soporte para una gran variedad de operaciones. Esto se debe a que la estructura de cada protocolo que la conforma es amplia lo cual es consecuencia de la gran cantidad de opciones que permiten o a la vasta serie de valores para cada uno de los campos que integran cada paquete. Afortunadamente esto que, para su comprensión, parecería en principio algo muy complicado, no lo es tanto porque los campos que integran a cada una de las opciones tienen una formación estandarizada.

Conocer cada una de los valores definidos para cada campo de cada protocolo de esta arquitectura tiene cierta dificultad, pero es necesario si se desea implementar aplicaciones que los soporten, como es el caso del analizador de protocolos desarrollado en este trabajo. Para la aplicación programada, el considerar cada una de estas opciones para los paquetes de red soportados fue una tarea que ocupó bastante tiempo sobre todo porque la mayoría de las opciones se definen en fuentes distintas a donde se hace la descripción inicial del protocolo en cuestión y hubo que buscar el correspondiente *RFC* y leer en algunos casos gran parte de documentos largos para interpretar de forma adecuada los significados de las opciones. En el capítulo 2 se hace referencia a la estructura de los paquetes de cada protocolo que soporta esta aplicación, sin embargo, hay que tener en cuenta que lo descrito en ese apartado no cubre cada una de las opciones existentes para cada uno de ellos, sino que sólo trata de los temas establecidos a través del RFC inicial de cada especificación; lo cual no quiere decir que el programa no soporte todas y cada una de las opciones para los paquetes de datos. Estas extensiones que se fueron agregando a través del tiempo a cada especificación se pueden consultar en los RFCs respectivos, de los cuales se indica la fuente correspondiente en los anexos VI, VII y VIII de este trabajo escrito. En el caso de los frames Ethernet es de innegable importancia el conocer como se compone cada tipo de trama sobre todo porque hay que hacer la diferencia entre las que pueden contener paquetes de protocolos correspondientes a la suite TCP/IP y las que no. Evidentemente, el realizar las investigaciones no sólo fue necesario para redactar la parte de la tesis destinada a explicar el funcionamiento de los protocolos de la suite TCP/IP elegidos, sino para conformar una base de conocimientos fidedigna e indispensable para realizar este tipo de aplicaciones de red. Por otra parte, lo anteriormente descrito me sirvió para comprender de

forma más amplia y correcta los mecanismos de intercambio de información en una red de datos.

El primer diseño del analizador de protocolos sólo contemplaba los puntos esenciales que la aplicación debe reunir, lo cuales se mantuvieron intactos durante el desarrollo del programa. Sin embargo conforme se fue trabajando, como en muchos otros procesos de software, fue necesario añadir otros tantos para cubrir las necesidades que surgieron con el paso del tiempo para cumplir algunas de las metas planteadas. Establecer estos puntos básicos que conformaron *el esqueleto* de la aplicación fue esencial para observar el avance gradual de la programación, pero sobre todo para no perder el camino durante la fase de programación.

Utilizar el *Modelo incremental* para la generación del software fue una muy buena elección entre las técnicas disponibles ya que permitió fijar metas intermedias de desarrollo e ir acoplando los módulos generados. Esta forma de diseño fue la base de desarrollo, sin embargo se podría decir que realmente no apliqué solo esta metodología sino que la combiné con el *Modelo en espiral* por las características de evaluación de eficiencia de cada modulo hecho debido a que de cada uno de ellos no generé una versión definitiva en el primer intento, es decir, esa primer célula de la aplicación que cumplía las especificaciones básicas la fui mejorando poco a poco reiniciando otra vez las etapas de *análisis*, *diseño*, *código* y *prueba* pertenecientes al sistema incremental para conseguir la funcionalidad total requerida. Por otro lado, para la sección correspondiente a la generación de estadísticas emplee de forma simple el *Modelo de ensamblaje de componentes* debido a que para esta parte ocupé programación orientada a objetos, definiendo una clase y 2 métodos, además del método main, para manipular el objeto creado para el análisis del flujo de red; esto aparte de auxiliarme de clases definidas en el *API* de *Java*. Con esto se comprueba que, aunque cada metodología existente para programación es buena e intenta reglamentar con la mayor eficiencia cada paso a seguir para obtener un software de buena calidad en ocasiones es necesario hacer ciertas modificaciones a las etapas propuestas y en ocasiones combinarlas con las propuestas en otros modelos.

Con la descripción de las características del ambiente de desarrollo (sistema operativo, lenguajes de programación y compiladores) se logran establecer los recursos básicos para la elaboración y para el funcionamiento del analizador, es decir, la guía que señala a través de módulos las tareas generales pero necesarias para realizar un análisis de este tipo. Esto es importante ya que si se pretendiera hacer un programa similar utilizando ya sea un sistema operativo diferente o lenguaje de programación distinto, seguramente cambiaría la forma de realizar cada función, pero en sí, la estructura elemental y funcional debería mantenerse casi intacta.

Para los usuarios que pretendan comprender la forma en como realiza cada tarea el analizador de protocolos recomiendo leer el capítulo cuarto, ya que a pesar de no ser una documentación técnica formal y completa de los códigos fuente, proporciona una ayuda importante para entender los procedimientos que a mi parecer pudieran ser los menos fáciles de comprender. Para redactar este capítulo me fue indispensable el haber realizado comentarios en el mismo código fuente al momento de realizar la programación, ya que en algunos casos debido a la extensión de cada archivo era algo engorroso recordar el

significado de determinada información mostrada o la causa de porqué alguna tarea se había programado en específico de esa forma.

El software se proporciona de una forma muy practica ya que la carpeta que contiene los códigos fuente, así como la documentación compatible con el mecanismo de páginas de documentación del sistema operativo y los archivos generados con *javadoc*, está encapsulada y comprimida bajo el formato *tar.gz*, como se hace con una gran cantidad de software manejada en las distribuciones de GNU/Linux y en consecuencia conocida por los usuarios. Algo similar sucede con los pasos de instalación, ya que al manejarse a través de un archivo de tipo *Makefile* este proceso se simplifica a la ejecución de 2 instrucciones desde el intérprete de comandos. En general, se puede decir que esta aplicación se ajusta a los parámetros estándar de presentación, instalación y documentación, esto último con las páginas de manual que incluye de los programas ejecutables y el del DTD utilizado, con lo que una persona con experiencia a nivel usuario en el sistema operativo predefinido como plataforma, puede instalarlo y utilizarlo de forma sencilla. Para proporcionar al usuario el software en estos formatos y generar la documentación de las páginas de manual para el sistema operativo, así como para generar la documentación en formato HTML para el programa en Java tuve que investigar como se realizaban dichas labores. Anteriormente había realizado alguna documentación similar, sin embargo no recordaba con precisión la serie de pasos a ejecutar o la sintaxis de los documentos fuente necesarios. Así, se puede decir que generar la documentación me ayudó a refirmar estos conocimientos que requerí para algún proyecto de la carrera.

Esta aplicación de software es capaz de registrar en tiempo real el tráfico de una red de área local en lo que respecta a los protocolos IP, TCP, ICMP, UDP y ARP englobados en frames Ethernet II y Ethernet IEEE 802.3 SNAP, así como reconocer frames Ethernet IEEE 802.3 y Ethernet Novell RAW 802.3 e indicar que se ha presentado uno de este tipo dentro del flujo de datos; con lo cual se cubre lo planteado en el primer objetivo particular mencionado en la introducción de esta tesis.

Con los 2 tipos de bitácoras (en formato XML y flujo de bytes con encabezados descriptivos de cada paquete) que se generan y que es posible escribirlas en memoria secundaria se obtiene un registro exacto del flujo de red que se escucha, ya sea en modo promiscuo o en modo común. Con los dos tipos de bitácoras es posible revisar la información previamente almacenada ya sea a través de editores de texto y exploradores web para el caso de archivos XML, y a través de la salida estándar en el caso del archivo que almacena el flujo de bytes. De esta forma se consigue lo indicado en el segundo objetivo general de este trabajo.

Considero que este trabajo escrito junto con los códigos fuente desarrollados son una muy buena fuente de conocimiento para aquellas personas que busquen comprender el funcionamiento de los protocolos de red que se abordan así como la interacción entre ellos y los mecanismos para su transmisión a través de la capa de acceso a la red según la arquitectura TCP/IP, todo esto acotado a los límites establecidos para generar el programa. De igual forma, es un buen punto de partida para los interesados en desarrollar aplicaciones relacionadas con el monitoreo de tráfico de red, ya que se logra describir los pasos

necesarios para la elaboración de software de tipo sniffer, y para la organización de la información recabada del los paquetes de datos.

Se pretende que una vez dado el visto bueno al analizador de protocolos, esta aplicación de software esté al alcance de la comunidad estudiantil a través del laboratorio de redes y seguridad de la facultad de ingeniería de la UNAM o por medio de su sitio web: *http://redyseguridad.fi-p.unam.mx/Redes y Seguridad.htm*, así como en los laboratorios de computación del edificio Ing. Luis G. Valdés Vallejo de esta misma institución. De igual forma, ponerlo al alcance de una mayor comunidad de programadores en el sitio *http://sourceforge.net/* debido a que es uno de los principales lugares virtuales en Internet para compartir software libre.

A través de estos logros se cumplen los objetivos particulares descritos en la introducción y que conforman el objetivo general de la tesis: *Generar una aplicación que reúna las características esenciales de un analizador de protocolos*.

Como conclusión general puedo decir que haber realizado el analizador de protocolos me permitió reafirmar y poner en práctica conocimientos que me fueron dados durante las clases de la carrera, despejar muchas dudas que tenía con respecto a algunos temas del área de redes de datos y sobre todo adquirir nuevos conocimientos. Aunque ya había participado en el desarrollo de software, generar esta aplicación me permitió conocer no solo la parte de programación sino involucrarme profundamente en cuestiones de definición de requerimientos y diseño a partir de cero, así como darme cuenta de los contratiempos generales y particulares que se presentan al elaborar de un proyecto de software de este tipo.

Anexos

ANEXO I. Clasificación de redes por cobertura

Una forma simple y bastante popular de clasificar a las redes es basándose en su extensión física, es decir, a través del área por la que se extienden sus nodos de red. Aunado a esto, la clasificación por cobertura también involucra a las tecnologías que ocupan las redes y en consecuencia los dispositivos de red y enlaces de comunicación.

Por otra parte existe un término que va un poco más allá de la extensión física de la red, se trata de *Intranet*. El concepto de Intranet es algo difuso, sin embargo se puede interpretar como una red perteneciente a una organización o institución sin importar la extensión física de dicha red. Incluso una intranet puede contener a varias *LANs* o *MANs*, definiciones que se verán a continuación.

I.I Redes de área local

Las redes de área local, mejor conocidas como LANs (Local Area Networks) son las que ocupan desde unos pocos metros cuadrados, como la red de una sala de cómputo o un laboratorio, hasta unos cuantos cientos de metros, máximo 1 [km], como por ejemplo la red que conecta a unos cuantos edificios. Ocupan como enlaces de comunicación a medios guiados, ya sea cable coaxial, cable de par trenzado o fibra óptica; y en menor medida medios aéreos.

I.II Redes de área metropolitana

Se clasifica como MANs (Metropolitan Area Networks) a aquellas que por su extensión abarcan algunas decenas de kilómetros. Se integran por algunas redes LAN. En la actualidad el término MAN ha perdido presencia y en su lugar se ocupa el concepto de WAN.

I.III Redes de área amplia

Una red de área amplia se conoce como WAN debido a las iniciales de su traducción al idioma inglés, Wide Area Network. En un inicio se denominó como red de área amplia a aquella que permitía comunicarse a nodos que se encontraban alejados cientos de kilómetros o más. Sin embargo hoy en día también se ocupa para describir a redes de tipo MAN. Ocupa enlaces de comunicación terrestres o guiados.

Regularmente una red WAN se conforma por nodos de red que no necesariamente son hosts, sino dispositivos que retransmiten la señal y con lo cual se consigue la extensión física de esta red.

En la clasificación de redes por cobertura se hace una diferencia entre las redes que ocupan medios de comunicación como cables y las que ocupan medios aéreos o inalámbricos. Y de esta forma se presentan las redes WLAN (Wireless LAN), que tienen las características de una red LAN a excepción del medio de comunicación, ya que estas redes ocupan medios de radio frecuencia.

De igual manera algunas veces es ocupado el termino Wireless WAN para redes WAN inalámbricas.

ANEXO II. Topologías de red

Topología en un sentido estricto se refiere al estudio de una zona geográfica, sin embargo en al ámbito de las redes de datos, en específico el de las de computadoras, se refiere a la forma en como están conectados físicamente los nodos. Existen varios tipos de topologías de red, algunas de ellas ocupan una tecnología en particular y otras pueden diseñarse con varias tecnologías o estándares de comunicación.

II.I Topología de bus

En una topología de bus se tiene un medio de transmisión principal, conocido como bus, al cual se conectan todos los nodos de la red. Este medio de comunicación es cable coaxial y la interfaz de red de cada nodo se conecta a él por medio de un conector BNC comúnmente conocido como tipo T. En esta red la información que envía un nodo es transmitida a lo largo del bus y llega a cada nodo de la red, cada uno determinará si esa información está dirigida a él y si es así la aceptará y procesará, en caso contrario no la recibirá. Es necesario conectar a cada extremo del cable coaxial un elemento llamado *terminador*, que no es más que una alta resistencia para evitar efectos reflectivos de la señal que llega a esa punta del cable.

Debido a las características de transmisión del cable coaxial una red de esta topología no puede exceder de los 500 metros de longitud en cuanto al bus de datos (por lo menos no sin un dispositivo que refuerce la señal); y la separación entre conectores BNC que transmiten la señal a los nodos debe ser por lo menos de 2.5 [m] entre sí. Los dispositivos recientemente mencionados, se llaman repetidores y como ya se expresó, sirven para prolongar la señal del bus hasta otros 500 [m]. En total se pueden tener hasta 4 dispositivos que refuercen la señal y tener una red en bus de hasta 2500 [m] de longitud y con una capacidad máxima de 1024 hosts conectados a la red. La organización de estos repetidores debe ser tal que no haya más de 4 entre 2 nodos de la red. Lo anterior se muestra en las figuras II.I y II.II.

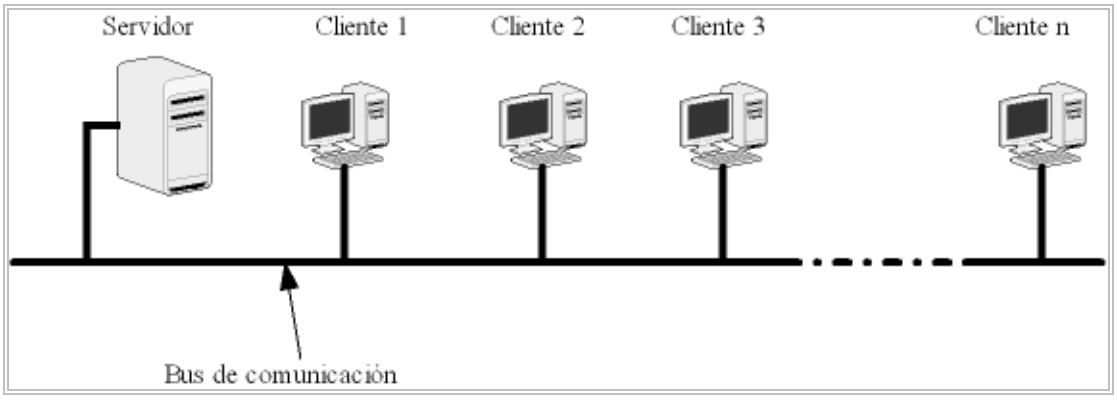

Figura II.I. *Topología de Bus*

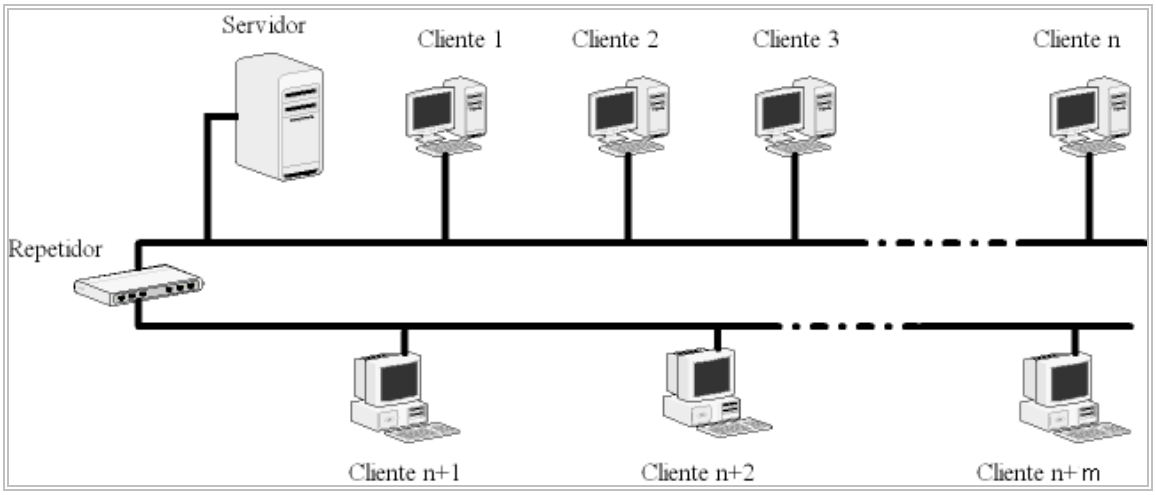

Figura II.II. *Topología de Bus con un repetidor*

La tecnología que ocupa se denomina *Ethernet* con *CSMA/CD* como método de acceso al medio. Otras características generales son:

- Topología utilizada para redes LAN.
- Facilidad para añadir nuevas estaciones a la red.
- Económica.
- Soporta hasta varias decenas de nodos.
- Tiempo de respuesta mayor mientras más nodos tenga la red.
- En caso de que el bus presente una interrupción la comunicación queda restringida a los nodos de la parte del bus donde se generó la información a transmitir.
- Usa codificación Manchester.

II.II Topología de estrella

En estas redes se tiene un dispositivo central que se encarga de controlar el flujo de información y al cual se encuentran conectados los nodos. Dependiendo de la tecnología de este elemento, la información se puede enviar a cada nodo de la red que esté conectado a él o comunicarse sólo al nodo al cual va dirigido. Actualmente utiliza como medio de transmisión al cable de par trenzado, en su mayoría UTP. Al igual que las redes en bus, también ocupa la tecnología Ethernet, pero con un método de acceso al medio diferente. Un diagrama de esta topología se muestra en la figura II.III.

Otros puntos importantes son:

- Se implementa en redes LAN.
- El control de transmisión es centralizado en lo que se refiere al flujo de información, es decir, el funcionamiento de la red depende del funcionamiento del dispositivo que controla el tráfico.

- La cantidad de nodos depende de la cantidad de puertos físicos del dispositivo central. Esto se puede incrementar mediante la conexión *en cascada* o *serial* de dispositivos que controlan el tráfico de red.
- La conexión de nodos de red es sencilla.

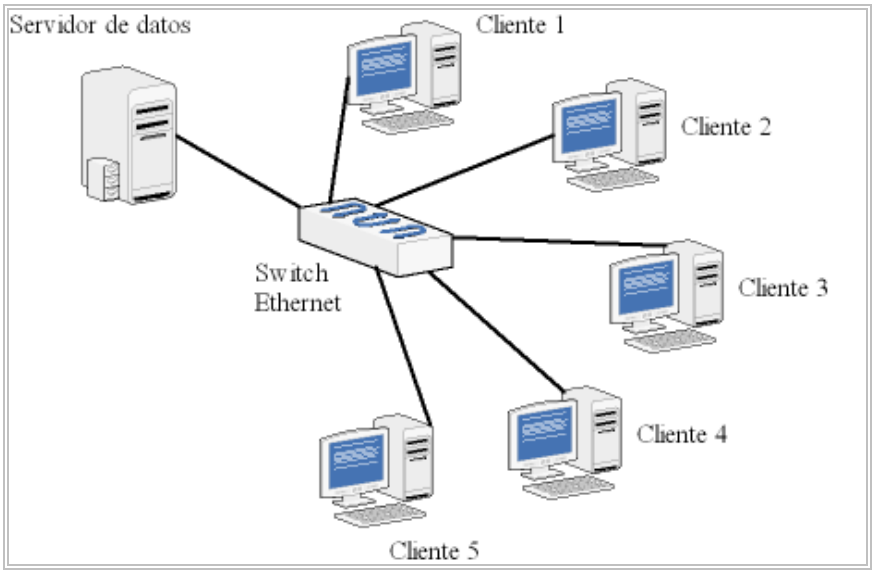

Figura II.III. *Topología de Estrella*

II.III Topología de anillo

Recibe su nombre debido a que la conexión entre los nodos se asimila a un anillo, cada uno de los cuales se conecta por medio de un cable coaxial o fibra óptica. Los paquetes de información se transfieren en un solo sentido del anillo a pesar de que el nodo destino se encuentre junto al emisor pero en sentido contrario. Utiliza la tecnología *Token ring* y el método de acceso al medio *Token passing*. La figura II.IV ejemplifica la topología de anillo.

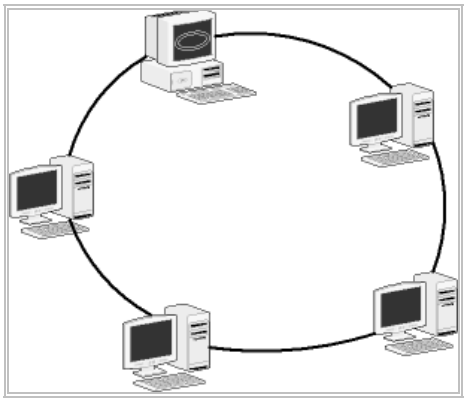

Figura II.IV. *Topología de Anillo*

Esta topología fue diseñada por IBM para redes LAN ocupando cable STP como medio de transmisión y podía albergar hasta un poco más de 260 nodos, pero en el estándar actual (IEEE 802.5) esta cantidad se limita a 250. En las primeras redes de anillo si un nodo de red fallaba la comunicación no podía continuar ya que el anillo se rompía, pero actualmente se ocupan dispositivos electromecánicos que solucionan este problema ya que al dejar de funcionar el nodo, automáticamente se cierra el circuito del anillo por un relevador; cuando el nodo vuelve a funcionar el relevador vuelve a su estado original. Este funcionamiento se ilustra esquemáticamente en la figura II.V. Los dispositivos *MSAU*, Multi Station Access Unit, también llamados MAU, contienen varios de estos relevadores en su interior y sirven para conectar varios nodos del anillo, con lo que físicamente se tiene una topología de estrella que funciona lógicamente como un anillo (ver figura II.VI).

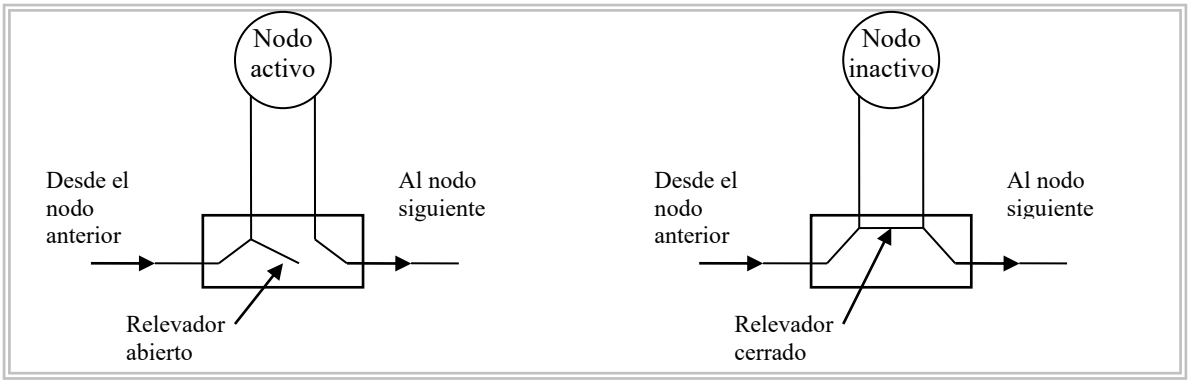

***Figura II.V**. Funcionamiento del relevador para topología de anillo.*

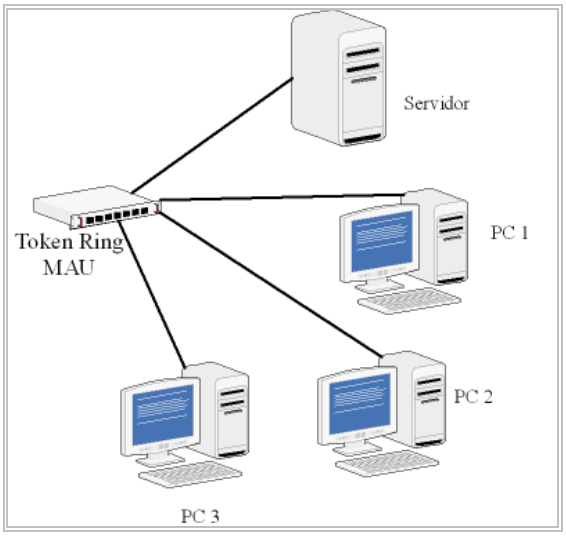

***Figura II.VI**. Topología de anillo con MAU (Multi Access Unit).*

Otros puntos que la describen son:

- En el estándar actual para redes de anillo no se especifica un medio de transmisión en particular.
- En redes LAN trabaja a anchos de banda de 4 y 16 Mbps.
- Todos los nodos pueden ver la información que viaja por el anillo.

ANEXO III. Subredes

Una manera de identificar a una red como clase A, B o C es revisar el rango de direcciones IP que ocupa para sus nodos de acuerdo a la clasificación vista anteriormente; sin embargo, es recomendable revisar la *máscara de red* (*Netmask*) que tiene configurada ya que a través de ella puede ser administrada como si fuera una clase de red distinta sin importar el rango de direcciones que ocupe. La única limitante se da en la escalabilidad para ser manejada como una clase de red que tenga capacidad para mayor número de nodos, es decir, una red clase A puede ser manejada como una red clase B, pero no a la inversa; incluso una red puede segmentarse en subredes asignándoles una máscara de red que no corresponde a las redes A, B y C. El beneficio de esto se ve reflejado en la administración de la red, ya que al subdividir una red se puede obtener un manejo más fácil de las máquinas que conforma a la subred pues la cantidad de host disminuye con respecto a la red original y cada red puede utilizarse de acuerdo a fines específicos (por ejemplo, investigación, desarrollo, usuarios en general, etc,) además de que cada subred es lógicamente independiente de las otras.

Una máscara de red tiene la estructura de una dirección IP, es decir, también se compone de 4 octetos de bits. A una red de clase A le corresponde la máscara de red *255.0.0.0*, a una red clase B la máscara de red *255.255.0.0*, y para una red clase C se tiene la máscara de red *255.255.255.0*. A grandes rasgos, la máscara de red se ocupa por algunos dispositivos de red, conocidos como *routers*, para conocer *a donde* enviar un paquete de datos que tiene como destino un hosts que no pertenece a la red en la que se encuentra dicho dispositivo y por ello también se puede ocupar para configurar una red en varias subredes.

Como se describe en la sección *1.3.1 Direcciones de Red IPv4*, una dirección IP se compone por 2 partes, una que identifica a la red y otra que identifica al host, y la forma para identificar la red a la que pertenece la dirección de red de un host es realizando una multiplicación bit a bit de la dirección del host con la máscara de red, por ejemplo, un host puede estar configurado bajo la dirección IP 132.248.69.30 y con una máscara de red 255.255.0.0, y aplicando una multiplicación bit a bit entre ellas se tiene:

1 0 0 0 0 1 0 0 . 1 1 1 1 1 0 0 0. 0 1 0 0 0 1 0 1. 0 0 0 1 1 1 1 0 → 132.248. 69 .30
1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 .0 0 0 0 0 0 0 0. 0 0 0 0 0 0 0 0 → 255.255. 0 . 0
---
1 0 0 0 0 1 0 0 . 1 1 1 1 1 0 0 0. 0 1 0 0 0 1 0 1. 0 0 0 0 0 0 0 0 → 132. 248 . 0 . 0

De esta operación se obtiene como resultado la dirección de red 132.248.0.0. Teniendo en cuenta el rango dentro del cual cae la dirección del host (128.0.0.0-191.255.255.255) y a la clasificación de redes en clases A, B y C, se pudo saber de antemano que se trataba de una dirección de red clase B y por lo tanto los bits que representan la dirección de red son los que integran los 2 primeros octetos y que en representación decimal son 132.248.

Sin embargo, existen casos en los que la máscara de red difiere a las ya mencionadas y en ese caso la dirección de red a la que pertenece el host cambia. Por ejemplo, una red clase C

puede *segmentarse* o dividirse en varias redes aplicando máscaras de red distintas a la 255.255.255.0. Si se considera una dirección IP *X.X.X.Y*, que está dentro del rango 192.0.0.0-223.255.255.255, donde inicialmente los octetos identificados con el carácter *X* corresponden a la dirección de red y el octeto indicado con el carácter *Y* corresponde a la dirección del host y se ocupa una máscara de red del tipo:

1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 .1 1 1 1 1 1 1 1. 1 0 0 0 0 0 0 0

Que en formato decimal se representa como 255.255.255.128 y después se aplica una multiplicación bit a bit entre ellas se va a obtener un resultado diferente a *X.X.X.0* que sería el esperado con una máscara 255.255.255.0. Ejemplificando un poco más, si el carácter Y representara al número 152, al hacer la operación para conocer la dirección de red de ese host se tendría:

```
X X X X X X X X . X X X X X X X X . X X X X X X X X . 1 0 0 1 1 0 0 0
1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 0 0 0 0 0 0 0
-----------------------------------------------------------------------------------
X X X X X X X X . X X X X X X X X . X X X X X X X X . 1 0 0 0 0 0 0 0
```

Este resultado en notación decimal es *X.X.X.128*. Ahora, si el carácter Y representa al número 17 al hacer la misma operación se obtiene:

```
X X X X X X X X . X X X X X X X X . X X X X X X X X . 0 0 0 1 0 0 0 1
1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 0 0 0 0 0 0 0
-----------------------------------------------------------------------------------
X X X X X X X X . X X X X X X X X . X X X X X X X X . 0 0 0 0 0 0 0 0
```

Lo cual representa una dirección de red X.X.X.0.

De estos resultados se puede ver que con la máscara de red 255.255.255.128 sólo se pueden obtener 2 direcciones de red, ya que con direcciones IP donde la Y sea un número mayor a 127 se obtendrá la dirección de red X.X.X.128, y cuando Y sea menor a 128 la dirección de red será X.X.X.0. En conclusión, con la máscara de red manejada, una red de clase C se divide en 2 redes, una que contiene las direcciones IPs que van desde X.X.X.0 a X.X.X.127 y otra que contiene las direcciones IPs en el rango X.X.X.128 a X.X.X.255. De cada una de estas dos redes, la dirección IP menor es la dirección que identifica a la red y la dirección IP mayor es la dirección de broadcast de dicha subred.

En la tabla III.I se muestran algunas máscaras de red para redes clase C y las subredes que originan:

Máscara de red	Cantidad de subredes	IPs inferiores de las subredes	IPs superiores de las subredes
255.255.255.0	1	x.x.x.0	x.x.x.255
255.255.255.128	2	x.x.x.0 x.x.x.128	x.x.x.127 x.x.x.255
255.255.255.192	4	x.x.x.0 x.x.x.64 x.x.x.128 x.x.x.192	x.x.x.63 x.x.x.127 x.x.x.191 x.x.x.255
255.255.255.224	8	x.x.x.0 x.x.x.32 x.x.x.64 x.x.x.96 x.x.x.128 x.x.x.160 x.x.x.192 x.x.x.224	x.x.x.31 x.x.x.63 x.x.x.95 x.x.x.127 x.x.x.159 x.x.x.191 x.x.x.223 x.x.x.255

Tabla III.I. Algunas máscaras de red para redes Clase C

Algo similar sucede para el subneting en redes clase A y clase B. Por ejemplo una red clase B *X.X.Y.Y* con una máscara de red 255.255.128.0 se generan 2 subredes, una con la dirección de red X.X.0.0 y otra con una dirección de red X.X.128.0. Un caso que es importante mencionar es que con una máscara de red 255.255.255.0 para una red de clase B se consiguen 254 redes, cada una equivalente a una red de clase C.

ANEXO IV. Dispositivos de red

Una red de datos se compone por nodos con los que los usuarios finales tienen estrecho contacto, como por ejemplo computadoras de escritorio, computadoras portátiles e impresoras, entre otros; sin embargo existen algunos elementos de la red que sólo los administradores de la red deben de utilizar y sólo en ciertas ocasiones en las que es necesario configurarlas. Este tipo especial de nodos tienen varias tareas pero en general todos ellos se encargan de controlar el flujo de los paquetes de datos enviados por nodos finales. Este anexo describe las características generales de estos dispositivos de red así como algunas de las técnicas más comunes para llevar a cabo dicho control de información.

IV.I Repetidor

Un repetidor es un dispositivo que permite regenerar una señal y se emplea cuando la distancia del medio de transmisión es tal que las señales transmitidas por los nodos sufren atenuación; por ejemplo una red con topología de bus puede ocupar repetidores cada 500[m] con el fin de extender su alcance; sin embargo sólo puede ocupar hasta 4 repetidores y alcanzar una longitud máxima de 2500[m].

IV.II Hub

Es un dispositivo que trabaja en el nivel físico de la arquitectura de red y se utiliza para redes que ocupan la tecnología Ethernet ya sea con una topología de bus, donde el Hub representa el bus principal o con topología de estrella con cables de par trenzado, ya que la información que recibe la reenvía a cada nodo conectado a él. En un principio, para el diseño físico de la red se tenía que tomar en cuenta el no superar una cantidad de 4 hubs entre 2 nodos por limitaciones físicas en lo que se refiere a la transmisión de los bits; pero ahora este contratiempo se ha superado en algunos modelos.

Dos hubs se pueden conectar entre sí de dos formas con lo cual se obtienen características distintas, cuando se conectan a través de puertos destinados para los nodos de red se considera que entre esos dos hubs se tiene el mismo segmento físico y se conserva la limitante de la extensión para el medio de transmisión, sea cable de par trenzado o cable coaxial. La otra manera es conectado cada hub a un canal (*backbone*) general utilizando un puerto especial y en consecuencia no se ocupa un puerto de red para nodos finales y se evita la limitante de la distancia del medio de transmisión.

IV.III Switch

Un switch es un dispositivo de red que funciona en la capa de red, y su principal tarea es dirigir al nodo adecuado, que puede ser un nodo final o un nodo que sea capaz de encontrarlo, los paquetes de información que recibe de los nodos que se le conectan. A la

acción de recibir paquetes y enviarlos al nodo adecuado se le llama *switching* o *forwarding*, lo cual es la principal tarea de la capa de red.

En redes con topología de estrella funge como nodo central pero a diferencia de un *hub* da a cada host conectado a él la capacidad de transmitir sus datos ocupando todo el ancho de banda correspondiente al medio de transmisión.

Existen 3 formas generales de llevar a cabo el forwarding, la primera es un método no orientado a conexión y es por medio de datagramas, mientras que la segunda sí es un método orientado a conexión y se conoce como *circuito virtual*. La última forma y la menos empleada se conoce como *source routing*.

IV.III.I Switching por datagramas

Cada paquete de datos contiene como parte de su información la dirección del host que lo emitió y la dirección del host a donde debe llegar. Mediante una tabla de ruteo configurada en el switch se conoce el puerto a donde se debe de enviar un paquete de información que tiene como destino un nodo en específico. Al proceso de la creación de la tabla de ruteo y su llenado se conoce como *routing*.

IV.III.II Switching por medio de circuitos virtuales

Se lleva a cabo en dos etapas, una de configuración de la conexión y otra de transferencia de paquetes. En la primera fase se establece el circuito entre los nodos finales, el cual si se lleva a cabo con la configuración de un administrador de red se conoce como *circuito virtual permanente*, PVC por sus siglan en inglés, ya que se supone que por un largo tiempo los hosts involucrados van a ocupar ese circuito para comunicarse; o si los circuitos se generan alternativamente por los hosts se les conoce como *switched virtual circuit* (SVC). Los SVC pueden ser borrados dinámicamente también y se generan por medio de señalizaciones entre el switch y los hosts.

Para formar un PVC en la tabla del switch en cada entrada se deben configurar 4 valores: un valor para la interfaz de entrada o puerto, un valor para el identificador del circuito virtual de entrada y que depende del hosts al que se quiera enviar el paquete para ese nodo origen en específico, un valor para la interfaz de salida y un valor para el identificador del circuito virtual de salida. De esta forma se tiene una única combinación de valores para cada posible combinación de hosts finales.

IV.III.III Switching por medio de source routing

La principal característica de esta metodología es que cada host fuente debe conocer toda la información de la topología de la red en que se encuentra. Con esto, cada host origen indica en la cabecera del paquete la ruta, a través de los puertos de los switches de la red, que se debe seguir para llegar al nodo destino. Esta técnica tiene como principal inconveniente que la longitud de la cabecera debe ser variable dependiendo de la estructura de la red, y por lo tanto es muy poco usada.

IV.IV Puente

El *Puente* o *Bridge*, también se conoce como *Switch LAN,* ya que su forma de trabajo es muy parecida a la de los switchs comunes, sólo que filtra tráfico de redes y no de hosts. Un puente se usa para interconectar 2 redes LAN que comparten un medio de transmisión para todos sus hosts, como redes Ethernet. El motivo para conectar redes Ethernet por medio de un bridge y no por medio de un repetidor es que este dispositivo funciona como un host más de la red y por lo tanto no interfiere para la organización en cuanto a las limitaciones físicas para este tipo de redes.

Un puente escucha todo el tráfico de las redes que interconecta y envia al puerto adecuado el frame de información que tiene como destino un host que no se encuentra en la Ethernet donde está el nodo origen. La decisión para distribuir paquetes se toma con base en una tabla de direccionamiento localizada en el bridge. A las redes que se interconectan por medio de un Puente se les conoce en conjunto como LAN Extendida.

En algunos casos la tabla de direccionamiento se puede generar manualmente pero no es conveniente debido a los frecuentes cambios en los equipos que integran una red; así que éstas entradas en el bridge se generan de forma automática, ya que al recibir un paquete de datos en algunos de sus puertos registra la dirección del nodo que lo envió, dirección MAC por ejemplo, para saber a qué puerto dirigirse en posteriores ocasiones para encontrarlo.

Algo importante en el funcionamiento de los bridges es que trabajan a nivel de la capa de enlace en una arquitectura de red y por lo tanto el uso del encabezado de red del frame limita a conectar redes que tengan el mismo tipo de formato para direcciones entre ellas.

IV.V Ruteador

Los ruteadores son dispositivos que actúan sobre la capa de red y permiten la transmisión de información entre redes que ocupen diferentes tecnologías de forma interna, por ejemplo redes Ethernet con Token Ring, pero que utilicen el mismo protocolo en este nivel, por ejemplo el protocolo IPv4 o AppleTalk.

Cuando un nodo requiere enviar información a otro que no se encuentra dentro de la misma red física y por lo tanto no es posible localizarlo a través de la dirección MAC, el paquete de información se envía a un ruteador de la red origen, el cual a través de un analisis de la dirección de red del encabezado y revisión de sus tablas de ruteo decide reenviar al paquete a otro ruteador que tiene la posibilidad de localizar ya sea al nodo destino o a otro ruteador con esta característica.

La información de dicha tabla puede ser ingresada manualmente por el administrador del *router*, pero en la mayoría de los casos se hace de forma automática a través de algoritmos de ruteo. El uso de los ruteadores permite subdividir la red y con ello lograr una mejor distribución del tráfico.

ANEXO V. Conceptos de desempeño de red

V.I Ancho de Banda

El ancho de banda de una red se define como la cantidad de información que puede ser transmitida sobre ésta en un periodo de tiempo. El ancho de banda puede verse desde dos enfoques, uno que se refiere al enlace físico y otro que involucra a la *lógica proceso a proceso del canal*. Con respecto al enlace físico el ancho de banda indica cuántos datos pueden transmitirse por él en determinado tiempo, por ejemplo puede decirse que un enlace tiene un ancho de banda de 10 Mbps (Mega bits por segundo), que en otras palabras significa que puede transmitir alrededor de 10 millones de bits en cada segundo o un bit cada 0.1 [μs]. Desde el otro punto de vista, el ancho de banda involucra además aspectos que tienen que ver con cuántas veces se pueden realizar las operaciones para manipular cada bit que es recibido, es decir, cuánto tiempo tarda el procesar cada bit que llega al nodo.

V.II Latencia

Este término, también conocido como *delay*, se aplica al tiempo que toma a un mensaje viajar de un lado de la red a otro. Por ejemplo, una red punto a punto puede tener una latencia de 1 [ms], lo cual quiere decir que le toma 1 [ms] a un mensaje llegar de un nodo al otro. Un término muy ligado al de latencia es el *Round-trip-time* (*RTT*) que indica cuanto toma enviar un mensaje desde un punto de la red a otro y recibir una respuesta de regreso.

La latencia se calcula de la siguiente forma:

Latencia = Tiempo de Propagación + Tiempo de Transmisión + Tiempo de espera

donde:

Tiempo de Propagación = Longitud del medio de transmisión / Velocidad de la luz en el medio de transmisión

Tiempo de Transmisión = Tamaño del mensaje / Ancho de banda del medio de transmisión

Se hace mención de la velocidad de la luz en el medio de transmisión porque esta varia dependiendo del medio de transmisión (2×10^8 [m/s] en fibra óptica a diferencia de 3×10^8 [m/s] en el vacio). De las relaciones anteriores se puede ver que la latencia depende del tiempo que toma transmitir una unidad de datos, lo cual depende del ancho de banda y del tamaño del paquete en el que los datos son transmitidos; así como también del *tiempo de encolamiento* en la red o del *tiempo de espera* por almacenamiento de información en algunos nodos antes de reenvíala a otros.

V.III Relación Ancho de Banda y Latencia

Una medida muy importante en el desempeño de las redes de datos es el que se obtiene de la multiplicación del ancho de banda del enlace por su latencia debido a que permite conocer cuantos bits pueden transmitirse por el link desde un nodo fuente a un nodo destino hasta antes que se obtenga una respuesta del host destino. Esta medida se podría interpretar como el volumen de un cilindro en el que el área de la base es el ancho de banda y la altura es el delay, tal y como se ilustra en la figura V.I.

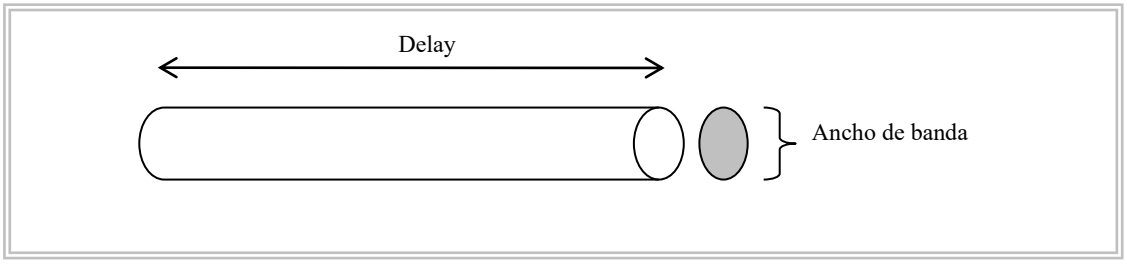

Figura V.I. *Representación gráfica del producto Ancho de Banda x Latencia.*

Realmente lo que se debe multiplicar es el ancho de banda por el RTT, ya que se debe toma en cuenta que la respuesta del host destino regresa por el mismo link.

Tanto el ancho de banda como la latencia son importantes pero se debe de dirigir la atención a una u otra dependiendo de cuestiones que dependen de la etapa de la comunicación que se esté llevando a cabo. Cuando los mensajes que se intercambian son de tamaño pequeño, por decir unos cuantos kilobytes, el ancho de banda del enlace no marca una gran diferencia en cuanto al desempeño de la red porque este generalmente es suficiente para transmitir el mensaje de datos sin fragmentarlo. Por ejemplo, si el mensaje tiene un tamaño de 10 kilobytes, en dos redes con un ancho de banda de 10 Mbps y otro de 100 Mbps el tiempo de transmisión, de acuerdo a las formulas mencionadas anteriormente es de 1[ms] y de 0.1 [ms]. Es decir, la diferencia de los tiempos de transmisión del mensaje es imperceptible porque todos los datos se transmiten de una sola vez. Este tipo de mensajes generalmente se intercambian entre hosts durante las negociaciones para establecer una conexión.

Caso distinto es cuando se envía mensajes de tamaño mucho mayor, por decir unos cuantos Megabytes, ya que el ancho de banda importa mucho más que la latencia o el RTT. Como ejemplo se pueden suponer dos redes con un mismo RTT pero con anchos de banda de 1 Mbps y de 100 Mbps a través de las que se debe enviar un archivo de 10 MB. Si el RTT es de 100 [ms] se necesitarán alrededor de 100 ciclos de RTT para transmitir todo el archivo ya que el producto ancho de banda x delay da un valor de 0.1 Mb para este caso. Mientras que para el otro enlace el resultado de esta multiplicación es alrededor de 100 Mbps con lo que se necesitaría con un ciclo de RTT sería más que suficiente para transmitir dichos datos ya que sólo ocuparían alrededor de 1/10 del total de bits que puede contener el enlace.

V.IV Throughput

Este término se puede entender como la cantidad real de datos que la red transmite en un enlace por unidad de tiempo, mientras que el ancho de banda indica el máximo de datos que se podrían transmitir por el link. Teóricamente el *throughput* se calcula de la siguiente forma:

Throughput= Tamaño de archivo/Tiempo de transferencia

y en este caso el tiempo de transferencia refiere al tiempo necesario para establecer la comunicación y el tiempo de transmisión.

Tiempo de transferencia= RTT + 1/Ancho de banda x Tamaño del archivo

Por ejemplo, si se quisiera enviar un archivo de 1 MB a través de un enlace con un ancho de banda de 1 Gbps que tenga un RTT de 100 [ms] el tiempo de transmisiòn sería de 8 [ms], con lo que el throughput serìa de 74.1 Mbps.

Estos nodos capaces de redireccionar paquetes de información, en su mayoría son elementos conocidos como *switch* y por lo general no sólo atienden a un único dispositivo, sino que varios nodos o *hosts* se interconectan con ellos. Debido a esto deben ser capaces de manejar el tráfico de paquetes de información que le son enviados para dirigirlos a través del medio de comunicación con el que se conectan a otros nodos, principalmente cuando se comunican con otro dispositivo que realiza una función similar. A este manejo o administración del tráfico de paquetes y direccionamiento a través de solamente un enlace se llama *multiplexaje*. Existen diversas formas de multiplexaje e incluso en un misma clase de enlace compartido se ocupan diferentes algoritmos para destinar el medio de transmisión a los paquetes de datos que llegan de otros hosts. En el multiplexaje por división de tiempo (*Time Divison Multiplexing, TDM*), el switch destina un tiempo determinado a transmitir los paquetes de un host y al concluir este transmite la información que recibió de otro nodo distinto, y así lo hace sucesivamente hasta volver a transmitir los paquetes del primer nodo mencionado durante otro periodo de tiempo. Otro método de multiplexaje es por división de frecuencias (*Frequency Division Multiplexing, FDM*) y consiste en transmitir por el mismo enlace de comunicación la información que llega desde diferentes nodos al mismo tiempo, pero cada uno a distintas frecuencias.

Sin importar el tipo de multiplexaje que se lleve a cabo en un dispositivo de este tipo, se tienen dos limitaciones en cuanto al volumen de datos que se transmite, y que se intentan disminuir con los algoritmos para la asignación de tiempos de atención a cada nodo. Uno de ellos es cuando se destina un tiempo de atención a un nodo de red y este no tiene información que enviar y por lo tanto durante ese tiempo el enlace no transporta información y en el caso de FDM una frecuencia no es ocupada. El segundo punto del que depende la eficiencia del multiplexaje se debe a la cantidad de señales que puede transmitir; en el caso de TDM no es recomendable alterar la cantidad de tiempo que se destina a cada

nodo que se conecta al dispositivo de multiplexado ni añadir nuevas frecuencias de transmisión en el caso del FDM.

Algunos de los algoritmos utilizados para la asignación de tiempos en el multiplexaje son el *FIFO* y la técnica *round robin*. El *statistical multiplexing* que tiene como fin el lograr una mejor distribución de los espacios de tiempo a los flujos de datos a transmitir se basa en la demanda del medio de transmisión; es decir, si durante el tiempo que destina para transmitir la información de un nodo no hay flujo de información cambia el derecho a usar el enlace a un flujo de datos correspondiente a otro nodo. Si el dispositivo no recibe datos de cualquier otro nodo para transmitir entonces permite que continúe la transmisión de paquetes del nodo que lo está usando.

ANEXO VI. IP option numbers

(last updated 2007-02-15)

The Internet Protocol (IP) has provision for optional header fields identified by an option type field. Options 0 and 1 are exactly one octet which is their type field. All other options have their one octet type field, followed by a one octet length field, followed by length-2 octets of option data. The option type field is sub-divided into a one bit copied flag, a two bit class field, and a five bit option number. These taken together form an eight bit value for the option type field. IP options are commonly refered to by this value.

Copy	Class	Number	Value	Name		Reference
0	0	0	0	EOOL	- End of Options List	[RFC791,JBP]
0	0	1	1	NOP	- No Operation	[RFC791,JBP]
1	0	2	130	SEC	- Security	[RFC1108]
1	0	3	131	LSR	- Loose Source Route	[RFC791,JBP]
0	2	4	68	TS	- Time Stamp	[RFC791,JBP]
1	0	5	133	E-SEC	- Extended Security	[RFC1108]
1	0	6	134	CIPSO	- Commercial Security	[???]
0	0	7	7	RR	- Record Route	[RFC791,JBP]
1	0	8	136	SID	- Stream ID	[RFC791,JBP]
1	0	9	137	SSR	- Strict Source Route	[RFC791,JBP]
0	0	10	10	ZSU	- Experimental Measurement	[ZSu]
0	0	11	11	MTUP	- MTU Probe	[RFC1191]*
0	0	12	12	MTUR	- MTU Reply	[RFC1191]*
1	2	13	205	FINN	- Experimental Flow Control	[Finn]
1	0	14	142	VISA	- Expermental Access Control	[Estrin]
0	0	15	15	ENCODE	- ???	[VerSteeg]
1	0	16	144	IMITD	- IMI Traffic Descriptor	[Lee]
1	0	17	145	EIP	- Extended Internet Protocol	[RFC1385]
0	2	18	82	TR	- Traceroute	[RFC1393]
1	0	19	147	ADDEXT	- Address Extension	[Ullmann IPv7]
1	0	20	148	RTRALT	- Router Alert	[RFC2113]
1	0	21	149	SDB	- Selective Directed Broadcast	[Graff]
1	0	22	150		- Unassigned (Released 18 October 2005)	
1	0	23	151	DPS	- Dynamic Packet State	[Malis]
1	0	24	152	UMP	- Upstream Multicast Pkt.	[Farinacci]
0	0	25	25	QS	- Quick-Start	[RFC4782]
0	0	30	30	EXP	- RFC3692-style Experiment (**)	[RFC4727]
0	2	30	94	EXP	- RFC3692-style Experiment (**)	[RFC4727]
1	0	30	158	EXP	- RFC3692-style Experiment (**)	[RFC4727]
1	2	30	222	EXP	- RFC3692-style Experiment (**)	[RFC4727]

[Note, an asterisk (*) denotes an obsoleted IP Option Number.]
[Note, two astarisks (**) It is only appropriate to use these values in explicitly-configured experiments; they MUST NOT be shipped as defaults in implementations. See RFC 3692 for details.]

ANEXO VII. TCP option numbers

(last updated 2007-02-15)

The Transmission Control Protocol (TCP) has provision for optional header fields identified by an option kind field. Options 0 and 1 are exactly one octet which is their kind field. All other options have their one octet kind field, followed by a one octet length field, followed by length-2 octets of option data.

Kind	Length	Meaning	Reference
0	-	End of Option List	[RFC793]
1	-	No-Operation	[RFC793]
2	4	Maximum Segment Size	[RFC793]
3	3	WSOPT - Window Scale	[RFC1323]
4	2	SACK Permitted	[RFC2018]
5	N	SACK	[RFC2018]
6	6	Echo (obsoleted by option 8)	[RFC1072]
7	6	Echo Reply (obsoleted by option 8)	[RFC1072]
8	10	TSOPT - Time Stamp Option	[RFC1323]
9	2	Partial Order Connection Permitted	[RFC1693]
10	3	Partial Order Service Profile	[RFC1693]
11		CC	[RFC1644]
12		CC.NEW	[RFC1644]
13		CC.ECHO	[RFC1644]
14	3	TCP Alternate Checksum Request	[RFC1146]
15	N	TCP Alternate Checksum Data	[RFC1146]
16		Skeeter	[Knowles]
17		Bubba	[Knowles]
18	3	Trailer Checksum Option	[Subbu & Monroe]
19	18	MD5 Signature Option	[RFC2385]
20		SCPS Capabilities	[Scott]
21		Selective Negative Acknowledgements	[Scott]
22		Record Boundaries	[Scott]
23		Corruption experienced	[Scott]
24		SNAP	[Sukonnik]
25		Unassigned (released 12/18/00)	
26		TCP Compression Filter	[Bellovin]
27	8	Quick-Start Response	[RFC4782]
28-252		Unassigned	
253	N	RFC3692-style Experiment 1 (*)	[RFC4727]
254	N	RFC3692-style Experiment 2 (*)	[RFC4727]

(*) It is only appropriate to use these values in explicitly-configured experiments; they MUST NOT be shipped as defaults in implementations. See RFC 3692 for details.

TCP ALTERNATE CHECKSUM NUMBERS

Number	Description	Reference
0	TCP Checksum	[RFC-1146]
1	8-bit Fletchers's algorithm	[RFC-1146]
2	16-bit Fletchers's algorithm	[RFC-1146]
3	Redundant Checksum Avoidance	[Kay]

ANEXO VIII. ICMP type numbers

(last updated 2007-08-02)

Registries included below:
- Code Fields
- ICMP Type Numbers
- ICMP Extension Objects Classes

The Internet Control Message Protocol (ICMP) has many messages that are identified by a "type" field.

Type	Name	Reference
0	Echo Reply	[RFC792]
1	Unassigned	[JBP]
2	Unassigned	[JBP]
3	Destination Unreachable	[RFC792]
4	Source Quench	[RFC792]
5	Redirect	[RFC792]
6	Alternate Host Address	[JBP]
7	Unassigned	[JBP]
8	Echo	[RFC792]
9	Router Advertisement	[RFC1256]
10	Router Solicitation	[RFC1256]
11	Time Exceeded	[RFC792]
12	Parameter Problem	[RFC792]
13	Timestamp	[RFC792]
14	Timestamp Reply	[RFC792]
15	Information Request	[RFC792]
16	Information Reply	[RFC792]
17	Address Mask Request	[RFC950]
18	Address Mask Reply	[RFC950]
19	Reserved (for Security)	[Solo]
20-29	Reserved (for Robustness Experiment)	[ZSu]
30	Traceroute	[RFC1393]
31	Datagram Conversion Error	[RFC1475]
32	Mobile Host Redirect	[David Johnson]
33	IPv6 Where-Are-You	[Bill Simpson]
34	IPv6 I-Am-Here	[Bill Simpson]
35	Mobile Registration Request	[Bill Simpson]
36	Mobile Registration Reply	[Bill Simpson]
37	Domain Name Request	[RFC1788]
38	Domain Name Reply	[RFC1788]
39	SKIP	[Markson]
40	Photuris	[RFC2521]
41	ICMP messages utilized by experimental mobility protocols such as Seamoby	[RFC4065]
42-255	Reserved	[JBP]

Many of these ICMP types have a "code" field. Here we list the types again with their assigned code fields.

```
Type    Name                                        Reference
----    ------------------------                    ---------
  0     Echo Reply                                   [RFC792]
        Codes
            0  No Code

  1     Unassigned                                      [JBP]

  2     Unassigned                                      [JBP]

  3     Destination Unreachable                      [RFC792]
      Codes
          0  Net Unreachable
          1  Host Unreachable
          2  Protocol Unreachable
          3  Port Unreachable
          4  Fragmentation Needed and Don't Fragment was Set
          5  Source Route Failed
          6  Destination Network Unknown
          7  Destination Host Unknown
          8  Source Host Isolated
          9  Communication with Destination Network is
             Administratively Prohibited
          10  Communication with Destination Host is
               Administratively Prohibited
          11  Destination Network Unreachable for Type of Service
          12  Destination Host Unreachable for Type of Service
          13  Communication Administratively Prohibited      [RFC1812]
          14  Host Precedence Violation                      [RFC1812]
          15  Precedence cutoff in effect                    [RFC1812]


  4     Source Quench                                        [RFC792]
        Codes
          0  No Code

  5     Redirect                                             [RFC792]

        Codes
            0  Redirect Datagram for the Network (or subnet)
            1  Redirect Datagram for the Host
            2  Redirect Datagram for the Type of Service and Network
            3  Redirect Datagram for the Type of Service and Host

  6     Alternate Host Address                               [JBP]

        Codes
            0  Alternate Address for Host

  7     Unassigned                                           [JBP]

  8     Echo                                                 [RFC792]

        Codes
            0  No Code
  9     Router Advertisement                                 [RFC1256]
```

```
       Codes
           0  Normal router advertisement
          16  Does not route common traffic                [RFC2002]


10     Router Selection                                     [RFC1256]

       Codes
           0  No Code

11     Time Exceeded                                        [RFC792]

       Codes
           0  Time to Live exceeded in Transit
           1  Fragment Reassembly Time Exceeded

12     Parameter Problem                                    [RFC792]

       Codes
           0  Pointer indicates the error
           1  Missing a Required Option                    [RFC1108]
           2  Bad Length


13     Timestamp                                            [RFC792]

       Codes
           0  No Code

14     Timestamp Reply                                      [RFC792]

       Codes
           0  No Code

15     Information Request                                  [RFC792]

       Codes
           0  No Code

16     Information Reply                                    [RFC792]

       Codes
           0  No Code

17     Address Mask Request                                 [RFC950]

       Codes
           0  No Code

18     Address Mask Reply                                   [RFC950]

       Codes
           0  No Code

19     Reserved (for Security)                              [Solo]

20-29  Reserved (for Robustness Experiment)                 [ZSu]
```

30	Traceroute	[RFC1393]
31	Datagram Conversion Error	[RFC1475]
32	Mobile Host Redirect	[David Johnson]
33	IPv6 Where-Are-You	[Bill Simpson]
34	IPv6 I-Am-Here	[Bill Simpson]
35	Mobile Registration Request	[Bill Simpson]
36	Mobile Registration Reply	[Bill Simpson]
39	SKIP	[Markson]
40	Photuris	[RFC2521]

Codes
- 0 = Bad SPI
- 1 = Authentication Failed
- 2 = Decompression Failed
- 3 = Decryption Failed
- 4 = Need Authentication
- 5 = Need Authorization

41-252	Unassigned	
253	RFC3692-style Experiment 1 (*)	[RFC4727]
254	RFC3692-style Experiment 2 (*)	[RFC4727]

(*) It is only appropriate to use these values in explicitly-configured experiments; they MUST NOT be shipped as defaults in implementations. See RFC 3692 for details.

ANEXO IX. Definición del Tipo de Documento para la bitácora XML

Los archivos DTD (siglas en inglés de *Document Type Definition*) indican la sintaxis para describir y restringir la estructura lógica de un documento XML. Contienen varios tipos de declaración además de comentarios e instrucciones de procesado.

La declaración DOCTYPE es la que contiene todas las demás declaraciones de un DTD y asocia el documento con el conjunto de declaraciones, debe tener el mismo nombre que el elemento raíz. Este elemento puede contener declaraciones internas al archivo donde se encuentre, o puede referir a declaraciones en otros archivos, como es el caso para el analizador de protocolos. Esta referencia se hace a través de *identificadores de sistema* e *identificadores públicos*, el primero es un *URI* de la localización del recurso en la organización del sistema y el segundo es un identificador independiente de la locación del archivo DTD. La sintaxis para este elemento es:

<!DOCTYPE nombre PUBLIC *identificadorPúblico identificadorDeSistema*>

o

<!DOCTYPE nombre SYSTEM *identificadorDeSistema*>

La declaración ELEMENT define un elemento a través de un nombre y específica su modelo de contenido: la palabra clave ANY que indica que cualquier hijo es permitido dentro del elemento; la palabra EMPTY que significa que el elemento no podrá contener otros elementos; el específico grupo de elementos hijo que contendrá encerrados entre paréntesis y separados por comas o el símbolo " | " que representa el operador OR para indicar que sólo uno de los hijos listados es permitido. La cantidad de veces que los hijos de un elemento pueden aparecer se puede manejar a través de los siguientes modificadores:

*	Cero o más veces
+	Una o más veces
?	Cero o una vez

Si no se indican modificadores el hijo o grupo de hijos deben aparecer exactamente una vez en la posición indicada. En el modelo de contenido se pueden crear gupos de hijos dentro de la lista de hijos permitidos a través de la anidación de paréntesis. Si el modelo de contenido es (#PCDATA) sólo se permite texto dentro del elemento.

La sintaxis para esta declaración es:

<!ELEMENT nombre *modelo_de_contenido*>

Para el diseño del documento DTD se tuvo en cuenta la estructura de los frames Ethernet y de los paquetes pra cada uno de los protocolos soportados, así como la información que se deseaba tener sobre el flujo recibido. A continuación se presenta el Document Type Definition para los archivos XML manejado por el analizador de protocolos.

Archivo frame.dtd

```
<!-- frame.dtd -->
<!ELEMENT flujo (frameEth)+>
<!ELEMENT frameEth (infoFrame|(infoFrame, encabezadoFrame)|(infoFrame, encabezadoFrame, datosFrame))>
<!ELEMENT infoFrame (numeroF, fechaF, horaF, tamanioF, tipoPaquete, tipoF)>
<!ELEMENT encabezadoFrame (dirEthOrigen, dirEthDestino, (EthII|IEEESNAP|IEEE))>
<!ELEMENT EthII (ethertype)>
<!ELEMENT IEEESNAP (longitud, dsap, ssap, control, protocolID, ethertype)>
<!ELEMENT IEEE (longitud, dsap, ssap, control)>
<!ELEMENT datosFrame (paqIP|paqARP)>
<!ELEMENT paqIP (encabezadoIP, opcionesIP*, (paqTCP|paqICMP|paqUDP))>
<!ELEMENT encabezadoIP (versionIP, IHL, tipoServ, longTotalIP, idPaq, bandDF, bandMF, posFrag, TTL, protocol, (FCSIP|FCSEIP), dirIPOrigen, dirIPDestino)>
<!ELEMENT opcionesIP (pal32Bits, tipoOpc, copOpcFrag, claseOpc, numeOpc, longOpc, (opIP0|opIP130|opIP132|opIP7|opIP136|opIP68|opIP133|opIP134|opIP10|opIP11|opIP12|opIP205|opIP142|opIP15|opIP144|opIP145|opIP82|opIP147|opIP148|opIP149|opIP150|opIP151|opIP152|opIP25|opIP30|opIP94|opIP158|opIP222)?)>
<!ELEMENT opIP0 (cantBytesRelleno)>
<!ELEMENT opIP130 (acercaDeOpciones, seguridad, compartimentos, manejoRestricciones,codControlTransmision)>
<!ELEMENT opIP131 (valorApuntador, dirApuntada)>
<!ELEMENT opIP137 (valorApuntador, dirApuntada)>
<!ELEMENT opIP7 (valorApuntador, dirAlmacAntValApunt*, espacioDireccion)>
<!ELEMENT opIP136 (identificadorFlujo)>
<!ELEMENT opIP68 (modIPsinRegMarcasTiempo, valorIndicadores, valorApuntador, (octetosApuntados?, marcaTiempo)* )>
<!ELEMENT opIP133 (codInfoAdiciSegur, infoAdiciSegur)>
<!ELEMENT opIP134 (datosDeOpcion)>
<!ELEMENT opIP10 (datosDeOpcion)>
<!ELEMENT opIP11 (datosDeOpcion)>
<!ELEMENT opIP12 (datosDeOpcion)>
<!ELEMENT opIP205 (datosDeOpcion)>
<!ELEMENT opIP142 (datosDeOpcion)>
<!ELEMENT opIP15 (datosDeOpcion)>
<!ELEMENT opIP144 (datosDeOpcion)>
<!ELEMENT opIP145 (datosDeOpcion)>
<!ELEMENT opIP82 (numeroId, OutboundHopCount, ReturnHopCount, IpGenPaqOutbound)>
<!ELEMENT opIP147 (datosDeOpcion)>
<!ELEMENT opIP148 (datosDeOpcion)>
<!ELEMENT opIP149 (datosDeOpcion)>
<!ELEMENT opIP150 (datosDeOpcion)>
<!ELEMENT opIP151 (datosDeOpcion)>
<!ELEMENT opIP152 (datosDeOpcion)>
<!ELEMENT opIP25 (func?, rateRequest?, (TTLpaqInicioRapido|octetoNoUsado)?, QSNonceCorriIzq2Pos, R)>
<!ELEMENT opIP30 (datosDeOpcion)>
<!ELEMENT opIP94 (datosDeOpcion)>
<!ELEMENT opIP158 (datosDeOpcion)>
<!ELEMENT opIP222 (datosDeOpcion)>
<!ELEMENT paqTCP (encabezadoTCP, opcionesTCP*, datosPaqCar?, alertaPaqIP?)>
<!ELEMENT encabezadoTCP (puertoOrigTCP, puertoDestTCP, numSec, numAcuse, posicionDat, bandURG, bandACK, bandPSH, bandRST, bandSYN, bandFIN, bandCRW, bandECN-ECHO, tamVent, (FCSTCP|FCSETCP), puntUrg)>
```

<!ELEMENT opcionesTCP (pal32Bits, tipoOpc, longOpc, (opTCP0|opTCP2|opTCP3|opTCP5?|opTCP6|opTCP7|opTCP8|opTCP10|opTCP11|opTCP12|opTCP13|opTCP14|opTCP15|opTCP16|opTCP17|opTCP18|opTCP19|opTCP20|opTCP149|opTCP150|opTCP151|opTCP152|opTCP25|opTCP30|opTCP94|opTCP158|opTCP222|opTCPdefault)*)>
<!ELEMENT opTCP0 (cantBytesRelleno, bytesRelleno?)>
<!ELEMENT opTCP2 (datosDeOpcion)>
<!ELEMENT opTCP3 (Shift-cntCorriDer)>
<!ELEMENT opTCP5 (limIzqBloque*, limDerBloque*)>
<!ELEMENT opTCP6 (datosDeOpcion)>
<!ELEMENT opTCP7 (datosDeOpcion)>
<!ELEMENT opTCP8 (marcaTiempoOrigen, marcaTiempoRespuesta)>
<!ELEMENT opTCP10 (bitBandInicio, bitBandFin, campoAjuste)>
<!ELEMENT opTCP11 (datosDeOpcion)>
<!ELEMENT opTCP12 (datosDeOpcion)>
<!ELEMENT opTCP13 (datosDeOpcion)>
<!ELEMENT opTCP14 (sumaControlTCP|algFletcher8Bits|algFletcher16Bits)>
<!ELEMENT opTCP15 (datosDeOpcion)>
<!ELEMENT opTCP16 (datosDeOpcion)>
<!ELEMENT opTCP17 (datosDeOpcion)>
<!ELEMENT opTCP18 (datosDeOpcion)>
<!ELEMENT opTCP19 (firmaMD5Hex)>
<!ELEMENT opTCP20 (datosDeOpcion)>
<!ELEMENT opTCP21 (datosDeOpcion)>
<!ELEMENT opTCP22 (datosDeOpcion)>
<!ELEMENT opTCP23 (datosDeOpcion)>
<!ELEMENT opTCP24 (datosDeOpcion)>
<!ELEMENT opTCP26 (datosDeOpcion)>
<!ELEMENT opTCP27 (CuatroBitsReserv, rateRequest, difTTL_IP-PaqIniRap, QSNonceCorriIzq2Pos, R)>
<!ELEMENT opTCP253 (datosDeOpcion)>
<!ELEMENT opTCP254 (datosDeOpcion)>
<!ELEMENT opTCP254 (datosDeOpcion)>
<!ELEMENT datosPaqCar (#PCDATA)>
<!ELEMENT paqICMP (tipoICMP, nomTipoICMP, codigoICMP, nomCodICMP?, (FCSICMP|FCSEICMP)?, (caseICMP3|caseICMP11|caseICMP12|caseICMP4|caseICMP5|caseICMP0|caseICMP8|caseICMP13|caseICMP14|caseICMP15|caseICMP16|caseICMP17|caseICMP18|caseICMP9|caseICMP10|caseICMP30|caseICMP31|caseICMP37|caseICMP38|caseICMP40|caseICMP41)?)>
<!ELEMENT caseICMP3 (reservICMP, numInt64bits, (alerta|cabeInt64bits))>
<!ELEMENT caseICMP11 (reservICMP, numInt64bits, (alerta|cabeInt64bits))>
<!ELEMENT caseICMP12 (octetoErroneo, reservICMP, numInt64bits, (alerta|cabeInt64bits))>
<!ELEMENT caseICMP4 (reservICMP, numInt64bits, (alerta|cabeInt64bits))>
<!ELEMENT caseICMP5 (dirEnvTrafico, numInt64bits, (alerta|cabeInt64bits))>
<!ELEMENT caseICMP0 (id, numSec, numBytesExtra, bytesExtra?)>
<!ELEMENT caseICMP8 (id, numSec, numBytesExtra, bytesExtra?)>
<!ELEMENT caseICMP13 (id, numSec, marcTiempoOrig, marcTiempoRecep, marcTiempoTrans)>
<!ELEMENT caseICMP14 (id, numSec, marcTiempoOrig, marcTiempoRecep, marcTiempoTrans)>
<!ELEMENT caseICMP15 (id, numSec)>
<!ELEMENT caseICMP16 (id, numSec)>
<!ELEMENT caseICMP17 (id, numSec, mascaraRed)>
<!ELEMENT caseICMP18 (id, numSec, mascaraRed)>
<!ELEMENT caseICMP9 (cantDirRutdrs, Pal32BytRutdr, segValDirRutdr, infoRutdr+)>
<!ELEMENT caseICMP10 (reservICMP)>
<!ELEMENT caseICMP30 (id, reservICMP, OutboundHopCount, ReturnHopCount, octetosPorSeg, linkSalidaMTU)>
<!ELEMENT caseICMP31 (octetoErroneo, numInt64bits, (alerta|cabeInt64bits))>
<!ELEMENT caseICMP37 (id, numSec)>

<!ELEMENT caseICMP38 (id, numSec, TTL, nombresDados)>
<!ELEMENT caseICMP40 (reservICMP, valorApuntador, numInt64bits, (alerta|cabeInt64bits))>
<!ELEMENT caseICMP41 (subtipo, reservICMP, opcionesICMP)>
<!ELEMENT cabeInt64bits (versionIP, IHL, tipoServ, longTotalIP, idPaq, bandDF, bandMF, posFrag, TTL, protocol, FCS, dirIPOrigen, dirIPDestino, puertoOrigTCP, puertoDestTCP, numSec, numAcuse, posicionDat, bandCRW, bandECN-ECHO, bandURG, bandACK, bandPSH, bandRST, bandSYN, bandFIN, tamVent, FCS, puntUrg)>
<!ELEMENT infoRutdr (dirIPRutdr, nivelPref)>
<!ELEMENT paqUDP (puertoOrigUDP, puertoDestUDP, longitud, (FCSUDP|FCSEUDP)?, datosExtraUDP?)>
<!ELEMENT datosExtraUDP (numBytesExtra, datosExtraHex, datosExtraCar)?>
<!ELEMENT paqARP (espDirHardware, espDirProtocol, longDirHardware, longDirProtocolo, codigoOper, dirHardwareOrig, dirProtocolOrig, dirHardwareDest, dirProtocolDest)>
<!ELEMENT acercaDeOpciones (seguridad, compartimentos, manejoRestricciones,codControlTransmision)>
<!ELEMENT numeroF (#PCDATA)>
<!ELEMENT fechaF (#PCDATA)>
<!ELEMENT horaF (#PCDATA)>
<!ELEMENT tamanioF (#PCDATA)>
<!ELEMENT tipoPaquete (#PCDATA)>
<!ELEMENT tipoF (#PCDATA)>
<!ELEMENT dirEthOrigen (#PCDATA)>
<!ELEMENT dirEthDestino (#PCDATA)>
<!ELEMENT ethertype (#PCDATA)>
<!ELEMENT longitud (#PCDATA)>
<!ELEMENT dsap (#PCDATA)>
<!ELEMENT ssap (#PCDATA)>
<!ELEMENT control (#PCDATA)>
<!ELEMENT protocolID (#PCDATA)>
<!ELEMENT versionIP (#PCDATA)>
<!ELEMENT IHL (#PCDATA)>
<!ELEMENT tipoServ (#PCDATA)>
<!ELEMENT longTotalIP (#PCDATA)>
<!ELEMENT idPaq (#PCDATA)>
<!ELEMENT bandMF (#PCDATA)>
<!ELEMENT bandDF (#PCDATA)>
<!ELEMENT posFrag (#PCDATA)>
<!ELEMENT TTL (#PCDATA)>
<!ELEMENT protocol (#PCDATA)>
<!ELEMENT FCS (#PCDATA)>
<!ELEMENT FCSIP (#PCDATA)>
<!ELEMENT FCSTCP (#PCDATA)>
<!ELEMENT FCSICMP (#PCDATA)>
<!ELEMENT FCSUDP (#PCDATA)>
<!ELEMENT dirIPOrigen (#PCDATA)>
<!ELEMENT dirIPDestino (#PCDATA)>
<!ELEMENT pal32Bits (#PCDATA)>
<!ELEMENT tipoOpc (#PCDATA)>
<!ELEMENT copOpcFrag (#PCDATA)>
<!ELEMENT claseOpc (#PCDATA)>
<!ELEMENT numeOpc (#PCDATA)>
<!ELEMENT longOpc (#PCDATA)>
<!ELEMENT cantBytesRelleno (#PCDATA)>
<!ELEMENT bytesRelleno (#PCDATA)>
<!ELEMENT seguridad (#PCDATA)>
<!ELEMENT compartimentos (#PCDATA)>
<!ELEMENT manejoRestricciones (#PCDATA)>

<!ELEMENT codControlTransmision (#PCDATA)>
<!ELEMENT valorApuntador (#PCDATA)>
<!ELEMENT dirApuntada (#PCDATA)>
<!ELEMENT dirAlmacAntValApunt (#PCDATA)>
<!ELEMENT espacioDireccion (#PCDATA)>
<!ELEMENT identificadorFlujo (#PCDATA)>
<!ELEMENT octetosApuntados (#PCDATA)>
<!ELEMENT marcaTiempo (#PCDATA)>
<!ELEMENT modIPsinRegMarcasTiempo (#PCDATA)>
<!ELEMENT valorIndicadores (#PCDATA)>
<!ELEMENT codInfoAdiciSegur (#PCDATA)>
<!ELEMENT infoAdiciSegur (#PCDATA)>
<!ELEMENT datosDeOpcion (#PCDATA)>
<!ELEMENT numeroId (#PCDATA)>
<!ELEMENT OutboundHopCount (#PCDATA)>
<!ELEMENT ReturnHopCount (#PCDATA)>
<!ELEMENT IpGenPaqOutbound (#PCDATA)>
<!ELEMENT func (#PCDATA)>
<!ELEMENT rateRequest (#PCDATA)>
<!ELEMENT TTLpaqInicioRapido (#PCDATA)>
<!ELEMENT octetoNoUsado (#PCDATA)>
<!ELEMENT QSNonceCorriIzq2Pos (#PCDATA)>
<!ELEMENT R (#PCDATA)>
<!ELEMENT puertoOrigTCP (#PCDATA)>
<!ELEMENT puertoDestTCP (#PCDATA)>
<!ELEMENT numSec (#PCDATA)>
<!ELEMENT numAcuse (#PCDATA)>
<!ELEMENT posicionDat (#PCDATA)>
<!ELEMENT bandURG (#PCDATA)>
<!ELEMENT bandACK (#PCDATA)>
<!ELEMENT bandPSH (#PCDATA)>
<!ELEMENT bandRST (#PCDATA)>
<!ELEMENT bandSYN (#PCDATA)>
<!ELEMENT bandFIN (#PCDATA)>
<!ELEMENT bandCRW (#PCDATA)>
<!ELEMENT bandECN-ECHO (#PCDATA)>
<!ELEMENT tamVent (#PCDATA)>
<!ELEMENT puntUrg (#PCDATA)>
<!ELEMENT Shift-cntCorriDer (#PCDATA)>
<!ELEMENT limIzqBloque (#PCDATA)>
<!ELEMENT limDerBloque (#PCDATA)>
<!ELEMENT marcaTiempoOrigen (#PCDATA)>
<!ELEMENT marcaTiempoRespuesta (#PCDATA)>
<!ELEMENT bitBandInicio (#PCDATA)>
<!ELEMENT bitBandFin (#PCDATA)>
<!ELEMENT campoAjuste (#PCDATA)>
<!ELEMENT sumaControlTCP (#PCDATA)>
<!ELEMENT algFletcher8Bits (#PCDATA)>
<!ELEMENT algFletcher16Bits (#PCDATA)>
<!ELEMENT firmaMD5Hex (#PCDATA)>
<!ELEMENT CuatroBitsReserv (#PCDATA)>
<!ELEMENT rateRequest (#PCDATA)>
<!ELEMENT difTTL_IP-PaqIniRap (#PCDATA)>
<!ELEMENT opTCPdefault (#PCDATA)>
<!ELEMENT tipoICMP (#PCDATA)>
<!ELEMENT nomTipoICMP (#PCDATA)>

<!ELEMENT codigoICMP (#PCDATA)>
<!ELEMENT nomCodICMP (#PCDATA)>
<!ELEMENT FCSE (#PCDATA)>
<!ELEMENT FCSEIP (#PCDATA)>
<!ELEMENT FCSETCP (#PCDATA)>
<!ELEMENT FCSEICMP (#PCDATA)>
<!ELEMENT FCSEUDP (#PCDATA)>
<!ELEMENT reservICMP (#PCDATA)>
<!ELEMENT numInt64bits (#PCDATA)>
<!ELEMENT alerta (#PCDATA)>
<!ELEMENT octetoErroneo (#PCDATA)>
<!ELEMENT dirEnvTrafico (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT numSec (#PCDATA)>
<!ELEMENT numBytesExtra (#PCDATA)>
<!ELEMENT bytesExtra (#PCDATA)>
<!ELEMENT mascaraRed (#PCDATA)>
<!ELEMENT cantDirRutdrs (#PCDATA)>
<!ELEMENT Pal32BytRutdr (#PCDATA)>
<!ELEMENT segValDirRutdr (#PCDATA)>
<!ELEMENT dirIPRutdr (#PCDATA)>
<!ELEMENT nivelPref (#PCDATA)>
<!ELEMENT OutboundHopCount (#PCDATA)>
<!ELEMENT ReturnHopCount (#PCDATA)>
<!ELEMENT octetosPorSeg (#PCDATA)>
<!ELEMENT linkSalidaMTU (#PCDATA)>
<!ELEMENT nombresDados (#PCDATA)>
<!ELEMENT subtipo (#PCDATA)>
<!ELEMENT opcionesICMP (#PCDATA)>
<!ELEMENT puertoOrigUDP (#PCDATA)>
<!ELEMENT puertoDestUDP (#PCDATA)>
<!ELEMENT longitud (#PCDATA)>
<!ELEMENT datosExtraHex (#PCDATA)>
<!ELEMENT datosExtraCar (#PCDATA)>
<!ELEMENT espDirHardware (#PCDATA)>
<!ELEMENT espDirProtocol (#PCDATA)>
<!ELEMENT longDirHardware (#PCDATA)>
<!ELEMENT longDirProtocolo (#PCDATA)>
<!ELEMENT codigoOper (#PCDATA)>
<!ELEMENT dirHardwareOrig (#PCDATA)>
<!ELEMENT dirProtocolOrig (#PCDATA)>
<!ELEMENT dirHardwareDest (#PCDATA)>
<!ELEMENT dirProtocolDest (#PCDATA)>

ANEXO X. Manpage para el programa Netbender

Netbender(8) Herramientas de administración Netbender(8)

NOMBRE

Netbender - Analizador de protocolos para la arquitectura de red TCP/IP.

SINOPSIS

```
Netbender [ -p protocolos ] [ -i interfaz ]
          [ -f nombreArchivoFlujoBytes ] [ -m modoEscucha ] [ -v ]

Netbender [ -x nombreArchivoFlujoBytes nombreArchivoXml ]

Netbender [ -t nombreArchivoFlujoBytes ]
```

DESCRIPCIÓN

Analizador de protocolos para la arquitectura de red TCP/IP con soporte para IP, TCP, ICMP, UDP y ARP.

Este programa guarda el flujo de red capturado tal cual se reciben los bytes con un pequeño encabezado descriptivo de cada frame en el archivo nombreArchivoFlujoBytes o en el archivo nombreArchivoXml con formato XML.

La palabra protocolos indicada en la sinopsis refiere a una lista de los protocolos que se capturarán del flujo de red. Los argumentos soportados para esta opción son:

ip Corresponde al protocolo IP y en consecuencia a TCP, UDP e ICMP.

tcp Corresponde al protocolo TCP.

icmp Corresponde al protocolo ICMP.

udp Corresponde al protocolo UDP.

arp Corresponde al protocolo ARP.

def Corresponde a los 5 tipos de protocolos soportados: IP, TCP, ICMP, UDP y ARP.

La palabra interfaz especifica la interfaz de red por la que se realizará la escucha del tráfico de red.

El argumento modoEscucha señala si se hará la escucha del tráfico de red en modo promiscuo y en cuyo caso tendrá un valor de P , o en modo normal para lo que su valor será N.

OPCIONES

-p Captura el flujo de red correspondiente con los protocolos dados como argumentos enseguida de esta opción. Si no se especifica esta bandera se capturará el tráfico de los 5 protocolos soportados.

-i Captura el flujo de red sólo por la interfaz indicada. Si se omite esta opción se realizará la captura a través de la interfaz eth0.

-f Guarda el flujo de red capturado en el archivo nombreArchivoFlujoBytes. Si no se indica esta bandera el flujo se guardará en un archivo llamado *flujo* concatenado con la fecha y hora en que se ejecuta el programa, por ejemplo "flujo30062008-01:22:20".

-m Indica si se realizará la escucha del tráfico de red en modo promiscuo o en modo normal.

-v Presenta en pantalla los datos capturados en tiempo real indicando a través de texto simple en que consiste cada campo. El uso de esta opción ocasiona una captura no integral de los frames debido a los recursos de memoria ram y de procesamiento que ocupa.

-x Lee los datos guardados previamente en nombreArchivoFlujoBytes y crea el archivo con formato XML llamado nombreArchivoXml.

-t Lee los datos guardados previamente en nombreArchivoFlujoBytes y muestra en pantalla los paquetes leídos indicando con un formato de texto en que consiste cada campo.

EJEMPLOS

Netbender -p ip -f miflujo01

Realiza la captura en modo promiscuo, a través de la interfaz de red predefinida (eth0), de frames Ethernet que contienen paquetes IP y los guarda en el archivo miflujo01.

Netbender -p arp tcp

Realiza la captura de frames Ethernet en modo promiscuo a través de la interfaz eth0 que contienen paquetes arp y frames Ethernet que contienen paquetes IP que transportan paquetes TCP y los guarda en un archivo cuyo nombre se forma por la concatenación de la palabra flujo y la fecha y hora en que se ejecuta el programa.

Netbender -i eth1 -f miflujo02 -v

Realiza la captura en modo promiscuo de frames Ethernet que contiene los 5 protocolos soportados a través de la interfaz de red eth1 y los guarda en un archivo llamado miflujo02, además de mostrar en pantalla los datos de cada frame indicando con texto el significado de cada campo.

Netbender -x miflujo01 miflujo01.xml

Crea un archivo con formato XML llamado miflujo01.xml a partir de un flujo de red capturado y almacenado previamente en el archivo miflujo01.

OBSERVACIONES

Con respecto al uso de la opción -v se recomienda sólo utilizarla cuando el flujo de red a capturar no se prevé que sea tan amplio como el de una red completa ya que su uso limita la captura de frames por los recursos de memoria ram y de procesamiento que utiliza, con lo cual no se

hace una lectura tan rápida como para identificar y almacenar todos los paquetes que existen en el flujo de datos. Una opción a esto es primero hacer la captura del flujo de red sin mostrar información de los paquetes en tiempo real y después mediante la opción -t imprimir en pantalla los datos de los paquetes con indicaciones de los campos en texto simple pero más comprensible que etiquetas XML como es el otro formato en que se almacenar y presentar la información.

VEÁSE TAMBIÉN

analizaflujo(8) , frame.dtd(5)

AUTOR

Heriberto García <hega07@gmail.com>

Version 1.0 **Junio del 2008** **Netbender(8)**

ANEXO XI. Manpage para el programa analizaFlujo

analizaFlujo(8) Herramientas de administración analizaFlujo(8)

NOMBRE

analizaFlujo - Generador de estadísticas de un flujo de red almacenado previamente.

SINOPSIS

java analizaFlujo nombreArchivoXml

DESCRIPCIÓN

Programa escrito en lenguaje Java que genera estadísticas a partir del archivo nombreArchivoXml con formato XML creado por la aplicación Netbender y que se apega a la sintaxis marcada en el *Document Type Definition* llamado *frame.dtd* propio de este conjunto de herramientas para análisis de tráfico de red.

Dicho archivo XML contiene datos capturados previamente de un flujo de red a través de la aplicación ya mencionada.

Las estadísticas que se obtienen a través de este programa son:

1. Cada dirección Ethernet origen identificada en el flujo de bytes con su correspondiente dirección IP y cuántas veces se presentó esta dupla en el tráfico de red.

2. Cada dirección Ethernet destino identificada en el flujo de bytes con su correspondiente dirección IP y cuántas veces se presentó esta dupla en el tráfico de red.

3. La dupla de direcciones origen (Ethernet-IP) que más se presentó en el tráfico de red.

4. La dupla de direcciones destino (Ethernet-IP) que más se presentó en el tráfico de red.

5. La cantidad de direcciones Ethernet, origen y destino, del flujo de bytes capturado.

6. La cantidad de direcciones IP, origen y destino, del flujo de bytes capturado.

7. Cantidad total de frames.

8. Cantidad total de paquetes ARP.

9. Cantidad total de paquetes IP.

10. Cantidad total de paquetes TCP.

11. Cantidad total de paquetes ICMP.

12. Cantidad total de paquetes UDP.

13. Cantidad de paquetes IP con opciones.

14. Cantidad de paquetes TCP con opciones.

15. Cantidad de paquetes ICMP con opciones.

16. Cantidad de paquetes UDP con datos extra.

17. Cantidad de paquetes IP con errores.

18. Cantidad de paquetes TCP con errores.

19. Cantidad de paquetes ICMP con errores.

20. Cantidad de paquetes UDP con errores.

OPCIONES

Este programa no tiene opciones de ejecución.

OBSERVACIONES

Para compilar el código fuente de este programa y ejecutarlo es necesario tener instalado el Java Development Kit al menos en su versión 1.5.

VEÁSE TAMBIÉN

Netbender(8) , frame.dtd(5)

AUTOR

Heriberto García <hega07@gmail.com>

Version 1.0 Junio del 2008 analizaFlujo(8)

ANEXO XII. Manpage para frame.dtd

frame.dtd(5) **frame.dtd** **frame.dtd(5)**

NOMBRE

frame.dtd

DESCRIPCIÓN

frame.dtd es un archivo Document Type Definition que indica la estructura de los archivos XML que contienen los frames de un flujo de red capturado y formateado a través de la aplicación Netbender.

VEÁSE TAMBIÉN

analizaFlujo(8) , Netbender(8)

AUTOR

Heriberto García <hega07@gmail.com>

Version 1.0 **Junio del 2008** **frame.dtd(5)**

Glosario

AppleTalk.

Sistema de comunicaciones de red (ahora obsoleto) que interconecta workstations, impresoras, módems compartidos y otras computadoras que actúan como servidores de archivos y de impresión. Entre sus características principales se encuentran: estar integrado en todos los MAC OS, proveer direccionamiento dinámico con lo que se facilita la instalación y configuración de equipos, y fácil exploración de recursos.

BNC.

Siglas de *Bayonette Neil-Concelman*. Termino que hace referencia a los adaptadores que utiliza el cable coaxial para conectarse a las interfaces de red diseñadas para ese medio de transmisión. Los más comunes son los conocidos como conectores *BNC tipo T.* Existen conectores BNC que permiten unir 2 segmentos de cable coaxial para extender la red.

Bytecode.

Código intermedio entre el código fuente de un programa y el código máquina. Tiene como fin eliminar la dependencia entre el código ejecutable de un programa y el hardware sobre el que fue compilado a través de su interpretación por una *máquina virtual*.

Cable coaxial.

Medio de transmisión que se compone por un conductor interior de cobre que está rodeado por una capa aislante de polietileno generalmente de forma continua. Sobre dicho aislante se encuentra un segundo conductor de cobre estañado o de plomo en forma de malla que tiene como propósito reducir las interferencias electromagnéticas. Por último se presenta la capa exterior también de polietileno y de color negro. Existen varios tipos de cable coaxial que se diferencian por el tipo de transmisión que pueden realizar y por su *impedancia*.

Cable de par trenzado.

Medio de transmisión que consiste de una más parejas de hilos de cobre, cada uno de un milímetro de diámetro y recubierto por plástico. Cada pareja de cables se trenza con

el objetivo de reducir interferencias electromagnéticas y efectos capacitivos. Para las redes de computadoras se emplean dos tipos de cable de par trenzado: UTP y STP. Para cada tipo de cable de par trenzado existen categorías que definen características en cuanto a ancho de banda.

Cable STP.

STP son las siglas de *Shielded Twisted Pair*. Es un cable de par trenzado de 4 parejas que se cubren por una capa de aluminio para disminuir aún más los efectos electromagnéticos de dispositivos eléctricos y electrónicos cercanos. Fue ideado por IBM para ser usado en uno de los diseños de red que ocupaban en un principio. Sobre la capa de aluminio se tiene una cubierta de PVC. La distancia máxima entre dos nodos conectados por STP es de 150 [m]. Su ancho de banda inicialmente era de 4 Mbps.

Cable UTP.

USP son las siglas de *Unshielded Twisted Pair*. La cubierta exterior es de PVC, y se forma por 8 cables (4 pares) que tienen como única protección a interferencias externas el trenzado de cada pareja, por lo que la máxima distancia entre nodos conectados por este tipo de cable es de 100 [m]. Sin embargo estas características son suficientes para los requerimientos que se tienen hoy en día, sin mencionar que el precio es menor que el del cable STP.

Categorías para cable de par trenzado.

El cable de par trenzado se clasifica en categorías incrementando su rendimiento en cuanto a ancho de banda mientras mayor categoría tenga. El número de categoría corresponde a la cantidad de trenzados por pie que tiene el cable. El ancho de banda correspondiente a cada categoría de cable UTP se presenta en la tabla G.1.

Cable UTP/STP	Ancho de banda permitido	Cantidad de trenzados por pie
Categoría 3	16 Mbps	3
Categoría 4	20 Mbps	4
Categoría 5	100 Mbps	5
Categoría 6	1 Gbps	6

G.1. *Categorías del cable UTP.*

Código ASCII.

Siglas del *American Standard Code for Information Interchange* (Código Americano Estándar para Intercambio de Información). Se trata de un código de 128 caracteres que utiliza 7 bits, define 33 caracteres no imprimibles y que en su mayoría son caracteres de control. Es usado para intercambiar información entre sistemas procesadores de datos, sistemas de comunicaciones de datos u equipo asociado.

Código Morse.

Es una codificación de caracteres (letras, números, signos de puntuación y caracteres especiales) a través de secuencias de elementos cortos y largos para transmitir información a través de un telégrafo. Dicha secuencia de elementos se integra por puntos para los elementos cortos, rayas para los elementos largos y espacios entre estos elementos.

Comunicación por microondas.

Las señales de microondas se ocupan para intercomunicar redes a grandes distancias. Dichas señales se transmiten con frecuencias que van entre los 890 [MHz] y los 20 [GHz] y por consecuencia la señal viaja prácticamente en línea recta así que las antenas receptoras deben estar alineadas. Este tipo de señales son afectadas por la curvatura de la tierra desviándola de su trayectoria inicial por lo que se debe tomar en cuenta al calcular la distancia entre receptores; mientras menor sea la frecuencia el alcance es mayor. Son afectadas por interferencias electromagnéticas pero son inmunes a la lluvia por eso se ocupan ampliamente en exteriores. Se utilizan para transmisiones telefónicas, de fax y video.

Conmutación de mensajes.

Método de comunicación que consiste en enviar un mensaje completo, es decir sin fragmentar, desde un nodo origen a un nodo intermedio que lo almacena en una lista de mensajes entrantes para re enviarlo cuando sea su turno ya sea a otro nodo intermedio o al nodo destino.

Cyclic Redundancy Check (CRC).

Se conoce por este nombre al resultado de una función o la función en sí que permite detectar errores en los datos recibidos. Sólo recibe como parámetros los datos (que pueden ser en cualquier cantidad) sobre los cuales se va a efectuar la operación y retorna como resultado un valor de longitud fija.

Ethertype.

Nombre del campo en un frame Ethernet y cuyo valor indica el tipo de protocolo o contexto al que corresponden los datos que lleva dicha trama. También se le da este nombre al valor que indica el protocolo en cuestión.

Fibra óptica.

Medio de comunicación cuyo principio de transmisión es por medio de luz. A través de un núcleo de fibra de vidrio se propagan uno o varios rayos de luz sobre los que se codifican los datos a transmitir. La fibra óptica se compone de un conductor de fibra de vidrio que va desde 10[μm] a 200[μm] de diámetro y se recubre por otra capa también de fibra pero que tiene un diámetro entre 125[μm] y 380[μm], la cual es envuelta por una capa de 125 [μm] de diámetro también de fibra pero con menor índice de refracción de tal forma que el rayo de luz transmitido sea reflejado en el interior del núcleo al incidir sobre su superficie con un ángulo mayor al ángulo crítico. Con esto se consigue que el rayo de luz al ser reflejado una y otra vez dentro de la fibra llegue al otro extremo. La capa siguiente del cable es de plástico flexible y la exterior es de silicón.

FCS.

Siglas de *Frame Check Sequence*. Se refiere al campo que se añade al final de un frame Ethernet y que contiene la suma de control del mismo para que el host receptor pueda identificar un error en el envío de los datos al recalcular esta suma hacer la comparación.

GPL.

Acronimo de *General Public License*. Significa Licencia Pública General y es una licencia que tiene como objetivo garantizar que el usuario sea libre de compartir y modificar un programa que es clasificado como software libre, y que las versiones modificadas o la original no puedan ser registradas como propiedad de algún particular y se prohíba las libertades a los usuarios sobre dicho programa.

IANA.

Siglas de Internet *Assigned Numbers Authority* (Autoridad de Asignación de Números de Internet). Es la organización responsable para la coordinación global de nombres de dominio en lo que respecta al nivel más alto, direccionamiento IP, y otros recursos de protocolos de Internet.

IETF.

Siglas de *Internet Engineering Task Force*. La IETF es una gran comunidad abierta de diseñadores de red, operadores, vendedores e investigadores concernientes a la evolución de la arquitectura de Internet y la operación de Internet sin problemas.

Interfaz de Programación de Aplicaciones.

Es un conjunto de funciones procedimientos o métodos que sobre un sistema operativo, librería o servicio soporta peticiones hechas por distintos software.

Impedancia.

Medida de la oposición a una corriente alterna que se presenta en un circuito. Se integra por la resistencia, la reactancia inductiva y la reactancia capacitiva, éstas dos últimas se encuentran 90 grados fuera de fase con respecto a la resistencia y en consecuencia la impedancia es un número complejo con la resistencia como la parte real y las inductancias como la parte imaginaria.

IPX.

Siglas de *Internetwork Packet Exchange* (Intercambio de Paquetes de Interred). Es un protocolo NetWare que provee un manejo de mensajes no orientado a conexión.

Java Development Kit.

Conjunto de aplicaciones de software diseñados por S*un Microsystems* que permiten desarrollar, probar y ejecutar apliaciones de software escritas en lenguaje Java.

Máquina virtual java.

Es un software que permite ejecutar archivos de tipo bytecode generados por un compilador de código fuente en java. Es una interfaz entre el bytecode y el hardware con lo cual permite ejecutar un programa de la misma forma independientemente del la arquitectura de la computdora y del sistema operativo con el que cuenta.

MAC OS.

Significa Sistema Operativo Macintosh y se refiere al sistema operativo desarrollado por Apple para las computadoras Macintosh.

NetWare.

Un Sistema operativo de red de Novell.

NFS.

Acronimo de *Network File System* (Sistema de Archivos de Red). Es un software que permite a computadoras montar una partición de disco en una máquina remota como si fuera un disco local. Lo cual permite compartir archivos a través de una red de forma rápida y segura.

Paradigma de programación.

Se refiere a la forma o enfoque que se sigue para la programación: estructurado, orientado a objetos, lógico, o funcional, por mencionar algunos.

RFC.

Siglas de *Resquest For Comments* (Solicitud de comentarios). Se refiere a una serie de documentos técnicos y organizacionales acerca de Internet, incluyendo especificaciones técnicas y documentos de políticas aprobadas por la IETF.

Software libre.

Se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. De modo más preciso, se refiere a cuatro libertades de los usuarios del software. La libertad de usar el programa, con cualquier propósito. La libertad de estudiar cómo funciona el programa, y adaptarlo a tus necesidades; la libertad de distribuir copias, con lo que puedes ayudar a tu vecino; la libertad de mejorar el programa y hacer públicas las mejoras a los demás, de modo que toda la comunidad se beneficie.

Snnifer.

Software que permite leer y registrar el contenido de paquetes del tráfico de una red digital de datos a pesar de no estar dirigidos al host donde está instalado el programa. También permite decodificar el contenido de cada paquete interceptado.

VI.

Editor de texto creado originalmente para sistemas UNIX y que actualmente es software libre. Se ocupa para editar archivos de código fuente y archivos de texto plano.

XML.

Siglas de Extensible Markup Language (Lenguaje de Marcas Extensible). Básicamente XML permite definir definir otros lenguajes de a cuerdo a necesidades particulares.

Referencias

Referencias bibliográficas

Andrew S., Tanembaum. *Redes de computadoras*. Cuarta edición. Pearson Educación de México, S.A. de C.V. 2003

Comer, Douglas E. *Internetworking with TCP/IP. Volume I. Principles and architecture.k* Prentice Hall, Inc. 1995.

Petearson, Larry L. y Davie, Bruce S.. *Computer Networks. A system Approach.* Tercera edición, Morgan Kaufmann Publishers, 2003.

Pressman, Roger S. *Ingeniería del Software. Un enfoque práctico.* Cuarta edición. McGraw-HILL/INTERAMERICADA DE ESPAÑA, S.A.U. 1998.

Stallings, William. *Data and Computer Communications*. Sexta edición, Prentice Hall, 1999.

Referencias de Internet

A la fecha del 4 de noviembre del 2008 se verificó que las siguientes direcciones Web todavía estuvieran en línea y correspondieran con la información consultada en el momento que se utilizaron como fuente de información para este trabajo. Las fechas indicadas en cada fuente corresponden a la fecha de edición o actualización de la información en cuestión.

http://www.theserverside.net/books/addisonwesley/EssentialXML/downloads/EssentialXML.zip. Libro electrónico. Skonnard, Aaron y Gudgin, Martin. *Essential XML Quick Reference*. Addison-Wesley. 2001.

http://es.tldp.org/Tutoriales/doc-tutorial-recomendaciones-seguridad/doc-tutorial-recomendaciones-seguridad.pdf. Libro electrónico. Bravo Estrada, Diego. *Recomendaciones de seguridad en sistemas distribuidos de cómputo (V0.11).*

http://es.tldp.org/Manuales-LuCAS/IAR/intro-admon-redes-v1.1.pdf. Libro electrónico. Hedrick. Charles L. *Introducción a la Administración de una Red Local basada en Internet.* Rutgers University. 1999.

http://www.rediris.es/cert/doc/segtcpip/Seguridad_en_TCP-IP_Ed1.pdf. Libro electrónico. Siles Peláez, Raul. *Análisis de seguridad de la familia de protocolos TCP/IP y sus servicios asociados.* Edición I. 2002.

http://docs.linux.cz/programming/other/ALP/advanced-linux-programming.pdf. Libro electrónico. Mitchell, Mark; Oldman, Jeffrey y Samuel, Alex. *Advanced Linux Programming*. 2001.

http://es.tldp.org/Tutoriales/PROG-SOCKETS/prog-sockets.pdf. Libro electrónico. BracaMan. *Programación Básica de Sockets en Unix para Novatos*.

http://sopa.dis.ulpgc.es/ii-dso/leclinux/ipc/sockets/LEC_SOCKETS.pdf. *Comunicación entre Procesos - Sockets*. Universidad de Las Palmas de Gran Canaria.

http://www.hispasec.com/directorio/laboratorio/articulos/EspiasEnRedesEthernet/06.html. HISPASEC SISTEMAS. Seguridad y Tecnologías de la información. *Detección de espias en redes ethernet: Implementación de la herramienta.* 2007.

http://blog.txipinet.com/2006/11/05/49-curso-de-programacion-en-c-para-gnu-linux-viii. txipi:blog. *Curso de programación en C para GNU/Linux (VIII)*. 2005.

http://ditec.um.es/laso/docs/tut-tcpip/3376fm.html. *Tutorial y descripción técnica de TCP/IP.*

http://hea-www.harvard.edu/~fine/Tech/addrinuse.html. Bind: Address Already in Use.

http://www.zonavirus.com/datos/articulos/148/Detecci%C3%B3n_esp%C3%ADas_redes_ethernet_LINUX.asp. Mejias Arenas, José Carlos. *Detección de espías en redes ethernet (LINUX).* 2005.

http://www.ctv.es/USERS/carles/PROYECTO/cap4/cap4.html. Munyoz Baldó. Carles Xavier. *LA INTERFAZ SOCKET.*

http://valkiriak.galeon.com/CONTENIDO/PROGRAMACION/pg03.htm. *Programacion " Sockets".*

http://linux.die.net/man/2/fcntl. *fcntl(2) - Linux man page.*

http://it.aut.uah.es/enrique/docencia/ii/redes/documentos/IOavanzada.htm. Conceptos Avanzados en socket.

http://www.arrakis.es/~dmrq/beej/index.html. "Beej" Hall, Brian. *Guía Beej de Programación en Redes.* 2001.

http://www.eslinux.com/articulos/8591/programacion-sockets-lenguaje-c. Programacion de sockets en lenguaje C. Ariel. 2004.

http://www.esimez.ipn.mx/acadcompu/apuntes_notas%20breves/apuntadores.pdf. Libro electrónico. Jiménez Pacheco, Javier. *APUNTADORES*.

http://www.itq.edu.mx/vidatec/maestros/sis/mlopez/Tutorial/apunt.htm. Apuntadores y Arreglos. El Instituto Tecnológico de Querétaro Mexico. 1999.

http://fmc.axarnet.es/redes/tema_13.htm. Secciones: Análisis del problema, Herramientas para diagnóstico.

http://www.gnu.org. GNU Operating System.

http://www.fsf.org. Free Software Foundation.

http://encarta.msn.com/media_461518096_761571562_-1_1/Telegraph.html. Picture and Sound Clip. Telegraph.

http://www.iana.org/assignments/ethernet-numbers. Ether Types.

http://www.iana.org/assignments/icmp-parameters. Icmp Type Numbers.

http://www.iana.org/assignments/ip-parameters. Ip Option Numbers.

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

http://www.iana.org/assignments/tcp-parameters. Transmission Control Protocol (TCP) Option Numbers.

http://www.rfc-editor.org/rfc/rfc768.txt. User Datagram Protocol.

http://www.rfc-editor.org/rfc/rfc791.txt. Internet Protocol.

http://www.rfc-editor.org/rfc/rfc792.txt. Internet Control Message Protocol.

http://www.rfc-editor.org/rfc/rfc793.txt. Transmission Control Protocol.

http://www.rfc-editor.org/rfc/rfc826.txt. An Ethernet Address Resolution Protocol.

http://www.rfc-editor.org/rfc/rfc903.txt. A Reverse Address Resolution Protocol.

http://www.rfc-editor.org/rfc/rfc943.txt. Assigned Numbers.

http://www.rfc-editor.org/rfc/rfc948.txt. Two Methods For The Transmission Of Ip Datagrams Over Ieee 802.3 Networks.

http://www.rfc-editor.org/rfc/rfc950.txt. Internet Standard Subnetting Procedure.

http://www.ietf.org/rfc/rfc1042.txt. A Standard for the Transmission of IP Datagrams over IEEE 802 Networks.

http://ipsysctl-tutorial.frozentux.net/other/rfc1072.txt. TCP Extensions for Long-Delay Paths.

http://tools.ietf.org/rfc/rfc1108.txt. Security Options for the Internet Protocol.

http://www.isi.edu/in-notes/rfc1122.txt. Requirements for Internet Hosts -- Communication Layers.

http://www.rfc-editor.org/rfc/rfc1146.txt. TCP Alternate Checksum Options.

http://www.ietf.org/rfc/rfc1191.txt. Path MTU Discovery.

http://www.ietf.org/rfc/rfc1256.txt. ICMP Router Discovery Messages.

http://www.ietf.org/rfc/rfc1323.txt. TCP Extensions for High Performance.

http://www.rfc-editor.org/rfc/rfc1385.txt. EIP: The Extended Internet Protocol. A Framework for Maintaining Backward Compatibility.

http://www.ietf.org/rfc/rfc1393.txt. Traceroute Using an IP Option.

http://www.rfc-editor.org/rfc/rfc1475.txt. TP/IX: The Next Internet.

http://www.isi.edu/in-notes/rfc1644.txt. T/TCP -- TCP Extensions for Transactions. Functional Specification.

http://www.rfc-editor.org/rfc/rfc1693.txt. An Extension to TCP : Partial Order Service.

http://www.rfc-editor.org/rfc/rfc1788.txt. ICMP Domain Name Messages.

http://www.ietf.org/rfc/rfc1812.txt. Requirements for IP Version 4 Routers.

http://www.isi.edu/in-notes/rfc2018.txt. TCP Selective Acknowledgment Options.

http://www.rfc-editor.org/rfc/rfc2113.txt. IP Router Alert Option.

http://www.ietf.org/rfc/rfc2385.txt. Protection of BGP Sessions via the TCP MD5 Signature Option.

http://www.rfc-editor.org/rfc/rfc2521.txt. ICMP Security Failures Messages.

http://www.rfc-editor.org/rfc/rfc4065.txt. Instructions for Seamoby and Experimental Mobility Protocol IANA Allocations.

http://www.rfc-editor.org/rfc/rfc4727.txt. Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers.

http://www.ietf.org/rfc/rfc4782.txt. Quick-Start for TCP and IP.