ESCUELA POLITECNICA NACIONAL

ESCUELA DE INGENIERIA

GENERADOR - COLECTOR DE PAQUETES TCP

PROYECTO PREVIO A LA OBTENCION DEL TITULO DE INGENIERO EN ELECTRONICA Y TELECOMUNICACIONES

MARCO ANTONIO VASQUEZ HERRERA

DIRECTOR: Msc. CARLOS EGAS

Quito, Diciembre 2001

DECLARACION

Yo Marco Antonio Vásquez Herrera, declaro que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

La Escuela Politécnica Nacional, puede hacer uso de los derechos correspondientes a este trabajo, según lo establecido por la Ley, Reglamento de Propiedad Intelectual y por la normatividad institucional vigente.

Marco Antonio Vásquez Herrera

CERTIFICACION

Certifico que el presente trabajo fue desarrollado por Marco Antonio Vásquez Herrera, bajo mi supervisión.

Ing. Carlos Egas

DIRECTOR DE PROYECTO

AGRADECIMIENTO

A Dios mi Señor que ha pesar de mis errores nunca me ha dejado solo, y me dio la fortaleza y voluntad para seguir adelante.
Quiero agradecer también a personas que en el camino te dan ánimo y apoyo para seguir adelante sin esperar nada a cambio, en especial al Ing. Fausto Fuentes, a la Sra. Mariana de Coronel y al Ing. Carlos Egas.

La fé mueve montañas.

DEDICATORIA

A mis padres, entrego ante ellos esta obra con amor, respeto y humildad; pues gracias a su sacrificio y buen ejemplo soy un hombre de bien.
A mi esposa e hijos, que ahora son la razón de mi existencia.
A mis hermanos, familiares y amigos.

MARCO ANTONIO.

RESUMEN.

La finalidad del proyecto que pongo a consideración es el de dotar de un medio que permita comprobar lo recibido en el proceso de aprendizaje de redes y a la vez de un ejemplo aunque sencillo de como trabajan los programas o aplicaciones de monitoreo de redes.

Comprender como funcionan los monitores de red y capturar paquetes que circulan por una red, para analizar su cabecera; siendo de particular interés los de protocolo TCP. Como resultado de la investigación se encuentra en Internet una arquitectura abierta para la captura de paquetes llamada WinPcap, desarrollada en el Politécnico de Torino-Italia y que puede trabajar bajo Windows. De WinPcap se utiliza una biblioteca con funciones que permite la captura de paquetes a bajo nivel. Se desarrolla una sencilla aplicación de software en Visual C++ 6.0 por medio de la cual se llama a algunas de la mencionadas funciones, que permiten capturar tramas ethernet, almacenarlas en un buffer, analizar el campo protocolo de los paquetes IP para filtrar aquellos cuyo protocolo sea TCP y mostrar la estructura de sus cabeceras.

Esta aplicación ha sido probada en una red LAN con tecnología ethernet, compuesta por computadoras con Windows 98, obteniendo resultados satisfactorios que permiten profundizar en el entendimiento de los protocolos TCP/IP.

La estructura de este proyecto esta dada por los siguientes capítulos:

El lector encontrará el planteamiento de las bases teóricas en el Capítulo 1; en el que se revisan brevemente la pila de protocolos TCP/IP, el modelo ISO/OSI, dando énfasis a la *estructura de la cabecera del paquete TCP*. En otro apartado de este capítulo se revisa también sin mayor profundidad la API de Windows Sockets (Winsock) poniendo mayor interés en el uso de los *sockets básicos*, y como núcleo de desarrollo de esta aplicación se presenta WinPcap, que con sus

APIs de alto nivel (Libpcap) y de bajo nivel (Packet.dll), explicando las estructuras y funciones de esta última.

Los requerimientos del programa de aplicación para la generación y captura de paquetes en la red, así como los pasos para utilizar la librería WinPcap en el Visual C++ son descritos en el Capítulo 2, además aquí se muestran los diagramas de Clase y de Objetos de la aplicación y se da una explicación bastante detallada sobre el desarrollo de la misma.

Los fundamentos del lenguaje de programación Visual C++, se muestran en el Anexo A de este trabajo, indicando como sugerencia utilizar la ayuda y ejemplos de programas, para el desarrollo de aplicaciones, pues mucho de lo que se necesita hacer alguien ya lo hizo y solo se debe entender y utilizar de acuerdo a los requerimientos que se tenga.

En el Capítulo 3 se hace una presentación de figuras capturadas de la pantalla de las computadoras en las que se utilizó la aplicación tanto como generador así como colector de paquetes; estas son las pruebas y resultados de este trabajo, y se hace un comentario sobre las mismas.

Las Conclusiones y Recomendaciones son presentadas en el Capitulo 4,siendo la más importante el hecho de que existe en Internet una nueva herramienta para el monitoreo de redes "ANALYZER", con librerías que se pueden utilizar para desarrollar nuevas herramientas o utilidades de software para redes, con una personalización de las aplicaciones de acuerdo a las características requeridas.

PRESENTACION.

El presente proyecto de titulación tiene como finalidad construir un ambiente teórico práctico, para el afianzamiento del aprendizaje y entrenamiento la familia de protocolos TCP/IP, en ambientes de redes LAN y WAN.

Como productos de esta proyecto, el lector encontrará el planteamiento de las bases teóricas de protocolos de redes (TCP/IP); así como del lenguaje de programación Visual C++ que es la plataforma de desarrollo de la aplicación que tiene como finalidad capturar los paquetes de datos directamente de la tarjeta o adaptador de red, para luego identificar los datos capturados (información de cabecera), filtrar los paquetes TCP recibidos. Además genera paquetes de prueba, que sirven para comprobar el buen funcionamiento de la aplicación desarrollada y así capturar ese paquete que circula por la red de comunicaciones utilizada.

Se difunde también una descripción de la librería WINPCAP, desarrollada por el Politécnico de Torino, que es útil para desarrollar aplicaciones de prueba y monitoreo de redes, misma que fue desarrollada basándose en la conocida librería LIBPCAP de la universidad de Berkeley, en la cual también tienen origen los sockets que luego permiten el aparecimiento de la API Winsock.

Se hace notar de ciertas limitaciones de la API WinSock que al usar sockets básicos y definir a TCP como el protocolo a usar, no es funcional.

Además aprovechando que se manipulan los datos a bajo nivel (tramas ethernet), se muestra las cabeceras IP de los paquetes.

Cabe señalar que este tema es parte de un conjunto de proyectos de titulación propuestos tomando como base la API de Winsock, con el objetivo final de integrar las diferentes aplicaciones en una sola herramienta para análisis y administración de redes.

CONTENIDO.

CAPITULO 1.

GENERADOR Y COLECTOR DE PAQUETES TCP.

1.1 INTRODUCCION..... 1

1.2 FUNDAMENTO TEÓRICO..... 3

1.2.1 GENERALIDADES DE LA PILA DE PROTOCOLOS TCP/IP..... 3

1.2.2 MODELO DE REFERENCIA ISO/OSI VERSUS TCP/IP..... 6

1.2.2.1 Modelo de Referencia ISO/OSI..... 6

1.2.2.2 Modelo De Referencia TCP/IP..... 9

1.2.3 ENCAPSULAMIENTO..... 11

1.2.4 ESTRUCTURA DEL DATAGRAMA O PAQUETE IP..... 12

1.2.5 PROTOCOLO DE CONTROL DE TRANSPORTE(TCP)..... 16

1.2.5.1 Funciones de TCP..... 16

1.2.5.2 Estructura del Paquete TCP..... 17

1.2.5.3 Banderas de la Cabecera TCP..... 19

1.2.5.4 Conexión en el Protocolo TCP..... 21

1.2.6 REVISIÓN DE LA API WINDOWS SOCKETS..... 22

1.2.6.1 Creación de un socket..... 25

1.2.6.2 Sockets básicos 27

1.2.7 NUEVAS APIs PARA MONITOREO DE REDES..... 31

1.2.7.1 Revisión de Antecedentes..... 31

1.2.7.2 Introducción 32

1.2.7.3 Descripción..... 33

1.2.7.4 La estructura de la pila de captura..... 34

1.2.7.5 Estructuras de los datos..... 38

1.2.7.6 Las Funciones 42

1.3 HERRAMIENTA DE PROGRAMACION..... 54

CAPITULO 2

DESARROLLO DEL PROGRAMA DE APLICACION

2.1 REQUERIMIENTOS DEL PROGRAMA DE APLICACION. 55
2.2 PROCEDIMIENTO PARA UTILIZAR LA LIBRERIA PACKET.DLL..... 56
2.3 DIAGRAMAS DE CLASES Y OBJETOS DE LA APLICACIÓN..... 57
2.4 CREACIÓN DEL PROYECTO..... 58
2.5 DISEÑO DE LA INTERFAZ DE LA APLICACIÓN..... 62
2.6 GENERADOR DE PAQUETES (MODO ENVIAR)..... 63
2.7 COLECTOR DE PAQUETES (MODO RECIBIR)..... 66
2.8 VER CABECERA IP..... 73

CAPITULO 3.

PRUEBAS Y RESULTADOS

3.1 PAQUETE ENVIADO..... 75
3.2 PAQUETES RECIBIDOS..... 77
3.3 COMENTARIOS..... 80

CAPITULO 4

CONCLUSIONES Y RECOMENDACIONES..... 85

BIBLIOGRAFIA..... 88

ANEXO A

ENTORNO VISUAL C++

A.1 FUNDAMENTOS DE PROGRAMACIÓN EN VISUAL C++ 6.0....A1
A.2 VENTAJAS....A4
A.3 ESTRUCTURA DE UN OBJETO....A4
A.4 POLIMORFISMO....A5
A.5 CLASES....A5
A.6 ELEMENTOS DEL LENGUAJE....A6
A.7 OPERADORES....A7
A.8 SENTENCIAS DE CONTROL....A7
A.9 INICIO DE UN PROYECTO EN VISUAL C++6.0....A9

ANEXO B

MANUAL DE USUARIO.

ANEXO C

CODIGO FUENTE DEL PROGRAMA.

CAPITULO 1.

GENERADOR Y COLECTOR DE PAQUETES TCP.

1.1 INTRODUCCION.

Ahora que la humanidad vive en un mundo cada día más globalizado gracias al gran desarrollo y evolución de los medios de comunicación. En la actualidad es casi instantánea la difusión e intercambio de la información, de diversa índole, sea esta económica, deportiva, cultural, noticiosa, etc. En este gran campo de las comunicaciones, se tiene como medio fundamental la *Internet,* la red de redes que ha sido determinante, en este proceso de globalización.

[♣]A finales de los años 60, la Agencia de Proyectos Avanzados de Investigación del Departamento de Defensa de los Estados Unidos o ARPA, posteriormente llamada DARPA, comenzó una asociación con universidades y otros organismos de investigación, para el desarrollo de nuevas tecnologías de comunicación de datos.

En 1969 comenzó a funcionar la primera red de conmutación de paquetes con 4 nodos, ARPANET (Advanced Research Projects Agency Network). En 1975 la Agencia de Comunicaciones para la Defensa DCA(Defense Comunications Agency), asumió la responsabilidad del funcionamiento de la red, que ya cubría Estados Unidos de costa a costa.

Los protocolos iniciales de ARPANET eran lentos y solian sufrir de frecuentes problemas. En 1974 Vinton G Cerf y Robert E Kahn[♠] propusieron en un articulo el diseño de un nuevo núcleo de protocolos, esto supuso la base para el desarrollo del Protocolo de Internet (IP) y del Protocolo de Control de Transmisión (TCP).

[♣]TCP/IP Capl Sidnie Feit McGrawHill 1998

[♠] A Protocol for Packet Network Interconections, IEEE Transuetions of Comunicatios, mayo 1974

A partir de 1980, se tardo 3 años en convertir los host de ARPANET que eran menos de 100, al nuevo grupo de protocolos.

[♣]Y es entre 1977 y 1984, que se desarrolla el modelo de referencia para interconexión de sistemas abiertos OSI (Open System Interconection)

En el inicio las redes de computadoras tenían varios inconvenientes como los que se señalan a continuación:

- La interconexión de los equipos trabajaba solamente con equipos del mismo fabricante.
- Se tenía un número limitado de redes de área local y de área extendida.
- En ocasiones resultaba demasiado complejo la utilización de varios tipos de software para aplicaciones y dispositivos.
- Esta falta de flexibilidad no permitía que redes existentes se pudieran conectar de forma sencilla y barata.

Dicha situación cambió con el aparecimiento del conjunto de protocolos TCP/IP (protocolo de control de transporte / protocolo de Internet). El Departamento de Defensa de los Estados Unidos (DoD) creó el modelo TCP/IP porque necesitaba una red que pudiera sobrevivir a cualquier circunstancia y este modelo se transformo en el estándar a partir del cual se desarrollo Internet.

TCP/IP desarrolló varias aplicaciones importantes como:

- Acceso desde cualquier terminal a cualquier host.(Telnet)
- La capacidad de copiar archivos entre computadoras.(FTP)
- El intercambio de correo electrónico entre usuarios de Internet.(SMTP)
- La impresión remota.
- WEB (World Wide Web)

La cantidad de usuarios de red con protocolos TCP/IP es impresionante y sigue creciendo rápidamente, se están desarrollando nuevos servicios que

[♣] Cap2. PROGRAMACION EN INTERNET, K Jamsa/K Cope McGrawHill 1996.

se integran modularmente a los productos de TCP/IP, hay que tener en cuenta también que el ambiente de programación que ha crecido con mayor rapidez es el de Microsoft Windows, el segmento de usuarios de Internet que crece con mayor rapidez, es el de quienes se conectan a ella desde sus PC basadas en Windows. Miles de organizaciones se han conectado a Internet, ello ha ocasionado algunos problemas en una familia de protocolos diseñados originalmente para uso interno de los militares y el gobierno americano. Se están tratando estos problemas y se ha centrado la atención en la evolución de mecanismos más eficientes para direcciones, enrutamiento y para encontrar soluciones robustas a los problemas de seguridad.

1.2 FUNDAMENTO TEÓRICO.

1.2.1 GENERALIDADES DE LA PILA DE PROTOCOLOS TCP/IP.

Un *Protocolo* es un conjunto de reglas que hacen que la comunicación en una red sea más eficiente, y determina el formato y la transmisión de datos. Por ejemplo, IP contiene un conjunto de reglas para encaminar datos, mientras que TCP tiene reglas para la entrega confiable datos.

La información que viaja a través de una red se conoce como *paquete, datos o paquete de datos.* Un *paquete de datos* es una unidad de información lógicamente agrupada que se desplaza entre los sistemas de computación.

Una *pila de protocolos* es un conjunto de protocolos en capas que funcionan para proveer comunicación entre aplicaciones. Por ejemplo, TCP, IP y Ethernet constituyen una pila de protocolos.

Host es una computadora que ejecuta aplicaciones y tiene uno o varios usuarios.

Ruteador es el que encamina o da ruta a los datos por una red.

Una *internet* es un conjunto de redes LAN (red de área local) o WAN (red de área extendida), conectadas por ruteadores. La *Internet*[♠] (con I mayúscula) es una internet muy especial que conecta a miles de redes.

La expresión *nodo de red* se usa para referirse a una entidad de comunicaciones en una red sin especificar si se trata de un host, un ruteador u otro dispositivo, por ejemplo un puente (bridge).

Una *red de área local* (LAN) es una red de comunicación de datos cuyo objetivo es dar servicio a un área relativamente pequeña; normalmente de unos pocos kilómetros cuadrados (dependiendo del medio de transmisión).

Una *red de área extendida* (WAN) cubre una área geográfica mayor y normalmente se construye utilizando líneas telefónicas dedicadas y servicios de conmutación de paquetes.

Un *enlace* es un término más general, y puede ser cualquier medio, de área local o extendida, que utilizan los nodos para comunicarse mediante un protocolo de nivel de enlace.

La palabra *octeto* se utiliza para referirse a 8 bits, de igual manera suele utilizarse la palabra *byte*.

Algunas computadoras almacenan los datos guardando primero el byte más significativo, es lo que se llama el estilo *Big Endian*[♠] de representación de datos. Otras computadoras guardan primero el byte menos significativo, es el estilo *Little Endian*[♥]. El protocolo de Internet fue escrito con normas Big Endian.

[♠♥] TCP/IP Cap1 Sidnie Feit McGrawHill 1998

La petición de comentarios RFC (Request For Comments) 793, es la que trata, acerca de los estándares y normas que rigen el protocolo de control de transporte TCP, sin ser esta la única.

TCP/IP tiene algunas características únicas que justifican su durabilidad.

a) La arquitectura TCP/IP agrupa a bancos de redes, creando una red mayor llamada una internet. Para un usuario, una internet aparece, simplemente, como una única red compuesta por todos los host conectados a cualquiera de los nodos que la forman.

b) Los protocolos TCP/IP se diseñaron para ser independientes del hardware o de su sistema operativo, así como de las tecnologías de los medios y los enlaces de datos. Se requería que los protocolos fuesen robustos, sobreviviendo altas tasas de error en la red, con capacidad de enrutamiento adaptativo transparente en el caso de que se perdiesen nodos o enlaces.

c) La capacidad de la familia de protocolos TCP/IP para conectar de redes locales y extendidas heterogéneas le convierten en un buen *integrador*.

d) El protocolo de control de transporte TCP, que es parte de la familia de protocolos TCP/IP, es un protocolo de transmisión de datos *confiable*, orientado a conexión.

e) Los sistemas que utilizan TCP/IP normalmente disponen de una interfaz de programación de comunicaciones para los desarrolladores de software.

La mayoría utilizan una interfaz de programación de sockets de Berkeley definida inicialmente para el sistema operativo UNIX.

La interfaz de programación de sockets incluye:

- Rutinas básicas para crear, transmitir y recibir los mensajes independientes que se usan en las comunicaciones de UDP, no orientadas a conexión.
- Rutinas para establecer una conexión de TCP, enviar, recibir datos y cerrar la conexión.

Para lograr un intercambio confiable de datos entre computadoras se deben realizar los siguientes procedimientos:

- Empaquetar los datos.
- Determinar el camino que deben seguir.
- Trasmitirlos por el medio físico.
- Regular la tasa de transferencia de acuerdo con el ancho de banda disponible y la capacidad del receptor para absorber los datos.
- Ensamblar los datos entrantes para que se mantengan en la secuencia correcta y no haya pérdida de tramas.
- Comprobar los datos entrantes para ver si hay tramas repetidas.
- Notificar al emisor de que los datos se han recibido correctamente.
- Entregar los datos a la aplicación correcta.
- Manejar errores y problemas.

Como resultado se tiene un software de comunicaciones complejo. Con un modelo en capas resulta más sencillo agrupar funciones relacionadas e implementar el software de comunicaciones de forma modular.

1.2.2 MODELO DE REFERENCIA ISO/OSI VERSUS TCP/IP.

1.2.2.1 Modelo De Referencia ISO/OSI

Se hace una breve reseña del modelo de referencia para interconexión de sistemas abiertos ISO/OSI (Open Systems Interconnection of the International Organization for Standardization) de la Organización Internacional de Normalización. Las siet capas del modeloOSI son:

- Capa 7: La capa de aplicación.
- Capa 6: La capa de presentación.
- Capa 5: La capa de sesión.

- ❖ Capa 4: La capa de transporte.
- ❖ Capa 3: La capa de red.
- ❖ Capa 2: La capa de enlace de datos.
- ❖ Capa 1: La capa física.

A continuación se describen rápidamente que funciones realiza cada una de las siete capas del modelo ISO/OSI.

- La capa *aplicación* suministra servicios de red a las aplicaciones del usuario. Algunos ejemplos son: los navegadores de Web, el correo electrónico, la transferencia de archivos y la emulación de terminales. En resumen la **capa aplicación suministra procesos de red a aplicaciones.**

- La capa *presentación* garantiza que la información que envía la capa de aplicación de un sistema pueda ser leída por la capa de aplicación de otro. De ser necesario la capa presentación traduce entre varios formatos de datos utilizando un formato común.. Por ejemplo es aquí donde se comprime datos, o encripta datos, o cambia a formato ASCii o EBCDIC. En resumen la **capa presentación maneja la representación de los datos**.

- La capa *sesión* establece, administra y finaliza las sesiones entre dos hosts que se están comunicando. También sincroniza el dialogo entre las capas de presentación de los dos hosts y administra su intercambio de datos. En resumen la **capa sesión maneja la comunicación entre hosts**.

- La capa *transporte* segmenta los datos originados en el host emisor y los reensambla en una corriente de datos dentro del sistema del host receptor. Tiene a cargo la confiabilidad del transporte de datos mediante la detección y recuperación de fallas, establecer, mantener y terminar los circuitos virtuales, y el control de flujo de la información.

En resumen la **capa transporte maneja las conexiones de extremo a extremo**.

- La capa de *red* proporciona conectividad y selección de ruta entre dos sistemas de hosts que pueden estar ubicados en redes geográficamente distintas. En resumen la **capa red maneja el direccionamiento y el enrutamiento**.

- La capa de *enlace de datos* se ocupa del direccionamiento físico, la topología de la red, el acceso a la red, la notificación de errores, entrega ordenada de tramas y control de flujo. En resumen la **capa de enlace de datos permite la transferencia confiable de los datos a través de los medios**.

  La *capa de enlace* incluye una interfaz de hardware y dos módulos de protocolo: el *protocolo de resolución de direcciones* ARP(Addresss Resolution Protocol) y el *protocolo de resolución de direcciones inverso* RARP(Reverse Addresss Resolution Protocol). ARP mediante broadcast para una dirección IP encuentra la correspondiente dirección física (MAC).

  La capa de enlace maneja el intercambio de datos entre la capa física y la de red. Los datos se organizan en unidades llamadas tramas (paquetes), cada trama tiene una cabecera que incluye una dirección origen y destino e información de control y un campo conocido como cola que se usa para la detección de errores.

- Las características tales como niveles de voltaje, temporizaron de cambios de voltaje, velocidad de datos físicos y otros atributos similares se definen a través de las especificaciones de la *capa física*. En resumen la **capa física define características eléctricas, mecánicas y funcionales de los medios**.

1.2.2.2 Modelo De Referencia TCP/IP.

El modelo de referencia TCP/IP y la pila de protocolos TCP/IP hacen que sea posible la comunicación entre dos computadoras, desde cualquier parte del mundo a casi la velocidad de la luz.

El modelo TCP/IP tiene cuatro capas: la capa de aplicación, la capa de transporte, la capa de internet y la capa de red. es importante observar que algunas de las capas del modelo TCP/IP poseen el mismo nombre que las capas del modelo OSI. No confunda las capas de los dos modelos, porque por ejemplo la capa aplicación tiene diferentes funciones en cada modelo.

Los diseñadores de TCP/IP crearon una **capa aplicación** que maneja protocolos de alto nivel, aspectos de representación, codificación y control de dialogo; y esta realiza la funciones de las tres capas superiores del modelo OSI.

La **capa transporte** se refiere a los aspectos de calidad de servicio con respecto a la confiabilidad, el control de flujo y la corrección de errores. Uno de sus protocolos, el protocolo para control de la transmisión (TCP), ofrece maneras flexibles y de alta calidad para crear comunicaciones de red confiables, sin problemas de flujo y con un nivel de error bajo. TCP es un protocolo orientado a conexión. Mantiene un dialogo entre el origen y el destino mientras empaqueta la información de la capa de aplicación en unidades llamadas segmentos. Orientado a conexión no significa que el circuito exista entre las computadoras que se están comunicando (esto sería una conmutación de circuito). Significa que los segmentos de la capa viajan de un lado a otro entre dos hosts para comprobar que la conexión exista lógicamente para un determinado periodo. Esto se conoce como una conmutación de paquetes. Además de TCP la capa transporte tiene el protocolo de datagrama de usuario (UDP), este último no es orientado a conexión.

El propósito de la **capa de internet** es enviar paquetes origen desde cualquier red y que estos lleguen a su destino independientemente de la ruta y de las redes que se utilizaron para llegar hasta allí. El protocolo especifico que rige esta capa se denomina Protocolo Internet (IP). En esta capa se produce la determinación de la mejor ruta y la conmutación de paquetes.

El nombre de esta capa es muy amplio y se presta a confusión. También se denomina capa de Host a Red. Es la capa que se ocupa de todos los aspectos que requiere un paquete IP para realizar realmente un enlace físico y luego realizar otro enlace físico. Esta capa incluye los detalles de tecnología de LAN y WAN y todos los detalles de la capa física y de enlace de datos del modelo OSI.

En la figura 1.1 se muestran las capas de TCP/IP y OSI.

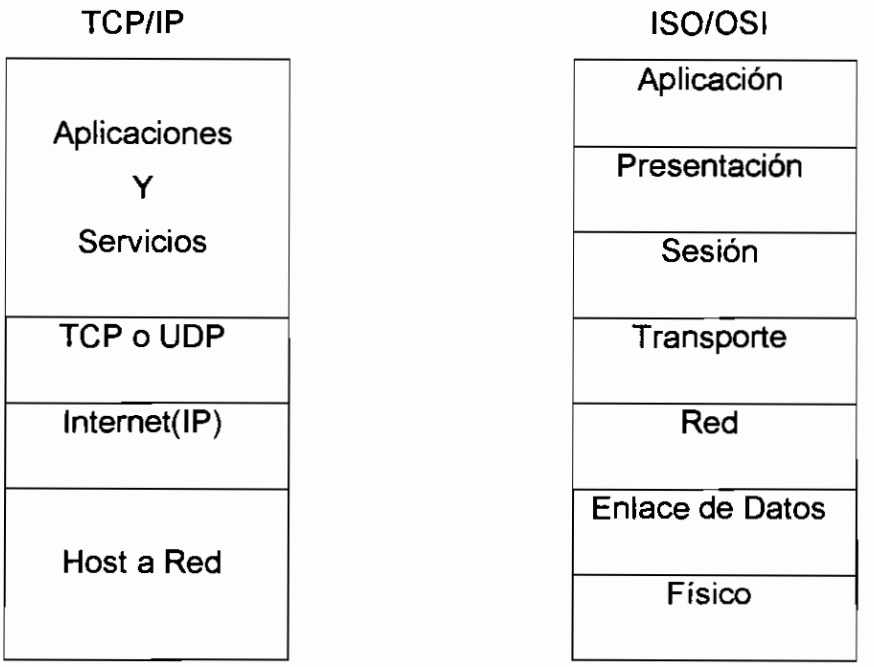


Figura.1.1 Capas de TCP/IP y de OSI.

Las conexiones de red son orientadas a conexión o sin conexión. Un protocolo orientado a conexión debe establecer un enlace con otra

aplicación antes de que exista cualquier comunicación, no puede transportar datos hasta que se establezca una conexión.

Un protocolo sin conexión no establece un enlace antes de transmitir mensajes. Como resultado cada mensaje que utiliza un protocolo sin conexión debe contener toda la información necesaria para su entrega; el protocolo de datagramas de usuario UDP y el protocolo Internet IP son protocolos sin conexión.

El *protocolo de Internet* realiza funciones de la capa de red. IP encamina datos entre sistemas. Los datos pueden atravesar un enlace único o pueden reenviarse por varios enlaces de una internet. Los datos se transportan en unidades llamadas datagramas, un *datagrama* tiene una cabecera IP que contiene información acerca de direcciones de la capa; los ruteadores examinan la dirección de destino de la cabecera IP para dirigir los datagramas a su destino. La capa IP es una capa no orientada a conexión ya que cada datagrama se encamina de forma independiente IP no garantiza una entrega confiable, ni en secuencia de los mismos.

1.2.3 ENCAPSULAMIENTO.

Como se puede observar en la figura 1.2 el módulo de aplicación puede encapsular los *datos del usuario* en un *mensaje de la aplicación*. El diseño de un programa determina si se debe utilizar o no este mensaje. En muchos casos el programa puede formatear los datos del usuario, para que los use un protocolo de la red digamos TCP; el módulo TCP formatea los datos de la aplicación en un *segmento TCP*, el cual incluye los datos de la aplicación junto con el encabezado de información TCP que necesita el protocolo. Conforme la información pasa por el módulo IP en la capa de red, el software de red formatea el segmento TCP para convertirlo en un *datagrama (o paquete) IP*. El controlador de Ethernet formatea los datos del módulo IP y los coloca dentro de una trama Ethernet.

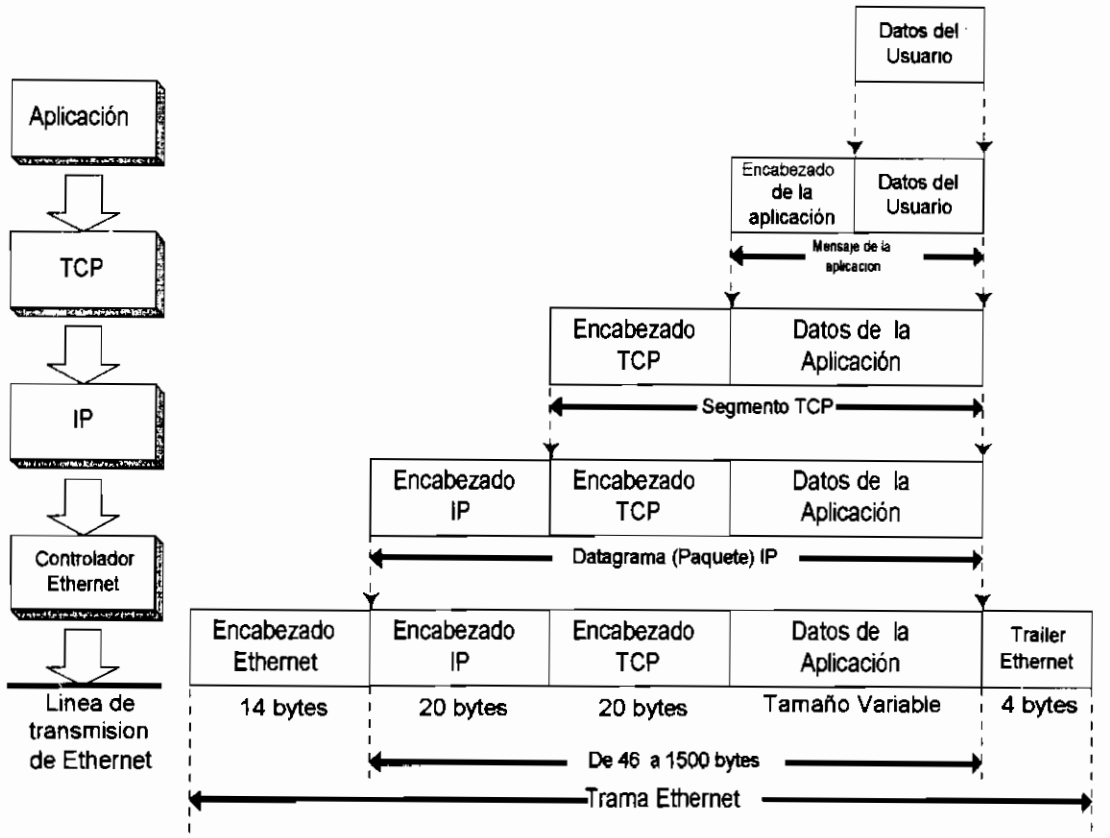


Figura.1.2 Encapsulamiento de datos utilizando TCP en una red ethernet.

Es importante recordar ya que será de utilidad en el futuro que el encabezado ethernet tiene una longitud de 14 bytes.

1.2.4 ESTRUCTURA DEL DATAGRAMA O PAQUETE IP.

La cabecera de un datagrama IP está formada por cinco o más palabra de 32 bits. El tamaño máximo de una cabecera es de 15 palabras, es decir 60 bytes, pero en la práctica la mayoría de las cabeceras de los datagramas tienen el tamaño mínimo de cinco palabras, veinte octetos o 20 bytes de longitud. Como se aprecia en la figura.1.3

A continuación se revisa brevemente qué indica cada campo de la cabecera de un paquete o datagrama IP.

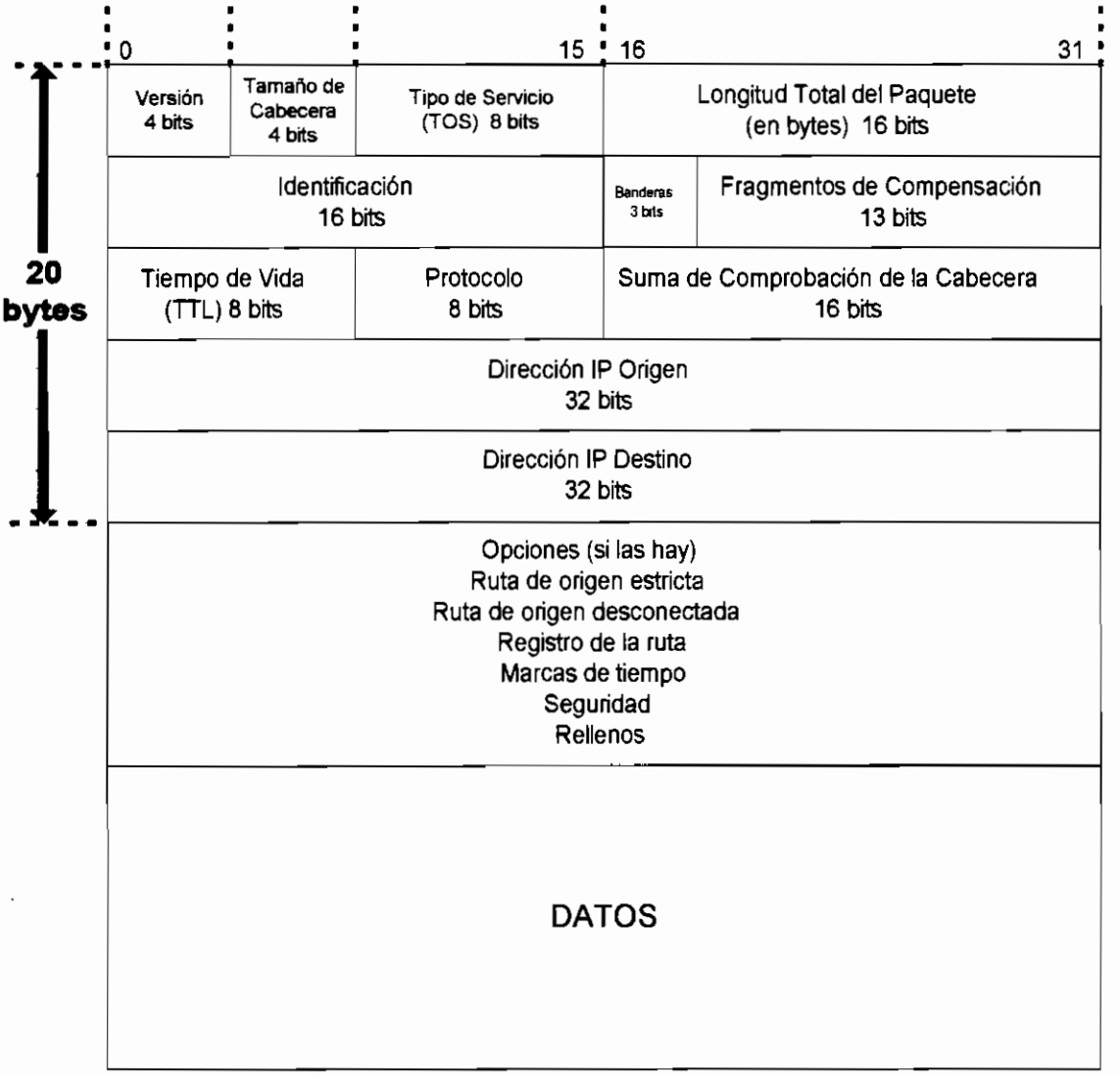


Figura.1.3 Estructura de un datagrama IP

- El primer campo da el número de *versión*, actualmente estamos tratando la versión 4.
- El campo HLEN (header length) da el *tamaño de la cabecera IP* en palabras de 32 bits , si no hay opciones el tamaño de la cabecera es de 5 palabras.
- El campo *tipo de servicio* (TOS) define las prioridades del paquete IP en cuanto a velocidad, capacidad de transferencia, costo y/o confiabilidad.

- El campo *longitud total del paquete* o datagrama mide su tamaño en octetos o bytes. En esta medida incluye tanto la cabecera como los datos del datagrama y puede tener valores de hasta 65.535 (2^{16} –1) octetos. Pero hay que tener en cuenta que cada tecnología de red específica el tamaño máximo del paquete que acepta, puede llamar a esta limitación unidad de transferencia máxima MTU (Maximum Transfer Unit); las especificaciones de ethernet limitan la transferencia a 1500 bytes.
- El campo *identificación* de 16 bits, en este número permite al host destino reconocer los fragmentos que pertenecen a un mismo datagrama.
- En general, las computadoras anfitrión o host destino utilizan los campos identificación, *banderas* y *fragmentos de compensación* para reordenar los paquetes IP fragmentados.
- El campo *tiempo de vida* especifica cuánto tiempo puede vivir el paquete en la red; la mayoría de los sistemas ajustan el campo de TTL a 30 segundos. Si el campo TTL llega a cero antes de que el paquete alcance su destino, TCP/IP inmediatamente destruye este último.
- Los campos más importantes de la cabecera son los de *dirección IP de destino de 32 bits* (que permite a IP enrutar su datagrama), dirección IP de origen de 32 bits y protocolo.
- El campo de 8 bits *protocolo* es de particular interés ya que éste indica que tipo de protocolo creó los datos encapsulados en el paquete, por ejemplo, si el campo protocolo contiene el valor 6, se sabrá que el paquete IP contiene un segmento TCP.
- El campo *suma de comprobación de la cabecera*[*], es un campo de 16 bits que contiene una suma de control que se calcula con los campos de la cabecera de IP. La suma de control hay que actualizarla según se reenvía el datagrama ya que el campo tiempo de vida cambia en cada ruteador; el resto de valores también pueden cambiar debido a la

[*] La *suma de control* es el complemento a uno de 16 bits de la suma con complemento a uno de todas las palabras de 16 bits de la cabecera. Antes del calculó el campo suma de control se pone a cero (0).

fragmentación o debido a los valores que se escriben en los campos opcionales.

Existen hasta 40 bytes extra en la cabecera de IP que pueden llevar una o más *opciones*. Las opciones que se incluyen en los datagramas las eligen las aplicaciones de origen. Es bastante raro usar estas opciones. Las opciones son:

- Uso de una ruta estricta.
- Ruta de origen desconectada.
- Crear un registro de la ruta.
- Marcas de tiempo (Timestamp).
- Seguridad básica del departamento de defensa.
- Seguridad extendida del departamento de defensa.
- Sin operación (bits de relleno entre opciones).
- Fin de la lista de opciones (bits de relleno al final, para completar 32 bits).

Lecturas recomendadas: protocolo IP se define en la RFC 791. En la RFC 1122 se especifican actualizaciones, correcciones y requisitos de conformidad. En la RFC 1812 se detallen los requisitos de los ruteadores de la versión 4 de IP y se explican detalles sobre el funcionamiento de estos ruteadores. Las RFC 1071, 1141 y 1624 tratan sobre el cálculo de la suma de control Internet.

1.2.5 PROTOCOLO DE CONTROL DE TRANSPORTE (TCP).

El protocolo Internet IP es simple para que la capa de red se centre en una importante función, el encaminamiento de los datos desde el emisor hasta el destino. Es tarea de TCP asegurar que los datos se entregan *fiablemente, en secuencia y sin confusiones o errores.*

Como principales características del protocolo de control transporte TCP se indican las siguientes:

- TCP proporciona un servicio de entrega de datos confiable, asegura esta confiabilidad incluyendo una suma de comprobación en los segmentos TCP e intercambiando mensajes de confirmación entre los módulos TCP.
- Las ventanas deslizantes permiten a los módulos TCP enviar múltiples segmentos sin tener que esperar mensajes de confirmación.
- TCP utiliza una ventana deslizante para incrementar el ancho de banda de la red y administrar el control de flujo.
- Los módulos TCP utilizan un saludo inicial en tres pasos* para establecer las conexiones TCP y un saludo final en dos pasos para terminarlas.

  Los programas cierran las conexiones TCP, cuando alguno de los dos lados envía un mensaje con la bandera FIN, y como segundo paso el otro lado le contesta también enviando una bandera FIN.

1.2.5.1 Funciones de TCP.

En la siguiente lista se resumen las principales funciones de TCP:

- Asociar puertos con conexiones
- Establecer conexiones usando un acuerdo en tres pasos.
- Realizar un arranque lento para evitar sobrecargar la red.
- Dividir los datos en segmentos para su transmisión.
- Numerar los datos (octetos).
- Manejar los segmentos entrantes duplicados.

---

*Ver pag 21 **Conexión en el Protocolo TCP.**

- Calcular las sumas de control.
- Regular el flujo datos usando las ventanas de envío y recepción.
- Terminar las conexiones de manera ordenada.
- Abortar conexiones.
- Marcar datos urgentes.
- Confirmación positiva con retransmisión.
- Cálculo de los plazos de retransmisión.
- Reducir el tráfico cuando la red se congestiona.
- Indicar los segmentos que llegan en desorden.
- Comprobar si las ventanas de recepción están cerradas.

1.2.5.2 Estructura del Paquete TCP.

La figura 1.4 muestra la estructura de un segmento TCP y, aunque se muestra en capas, se debe entender que el encabezado es simplemente un flujo serial de datos de al menos veinte bytes de longitud. En los siguientes párrafos se describen los campos que comprenden el encabezado TCP.

- El *puerto origen* y el *puerto destino*, ambos campos de 16 bits, identifican de manera efectiva las aplicaciones enviadas y recibidas (o protocolos de aplicación). Los números de los campos puerto origen y destino, más la dirección IP origen y destino (en el encabezado IP), se combinan para crear una identificación única de cada conexión TCP. Puede llamar socket a cada lado de la conexión TCP.
- El campo *número de secuencia*, de 32 bits, identifica el primer byte de datos en el área de datos del segmento TCP. TCP identifica el byte por su compensación (offset) relativa del inicio del flujo datos, es decir a cada byte del flujo de datos se le identifica por un número de secuencia.
- El campo *confirmación*, de 32 bits, identifica el siguiente byte de datos que espera recibir la conexión del flujo de datos. Por ejemplo, si el último byte recibido tiene el número de secuencia de 1000, TCP envía el número 1001 como confirmación.

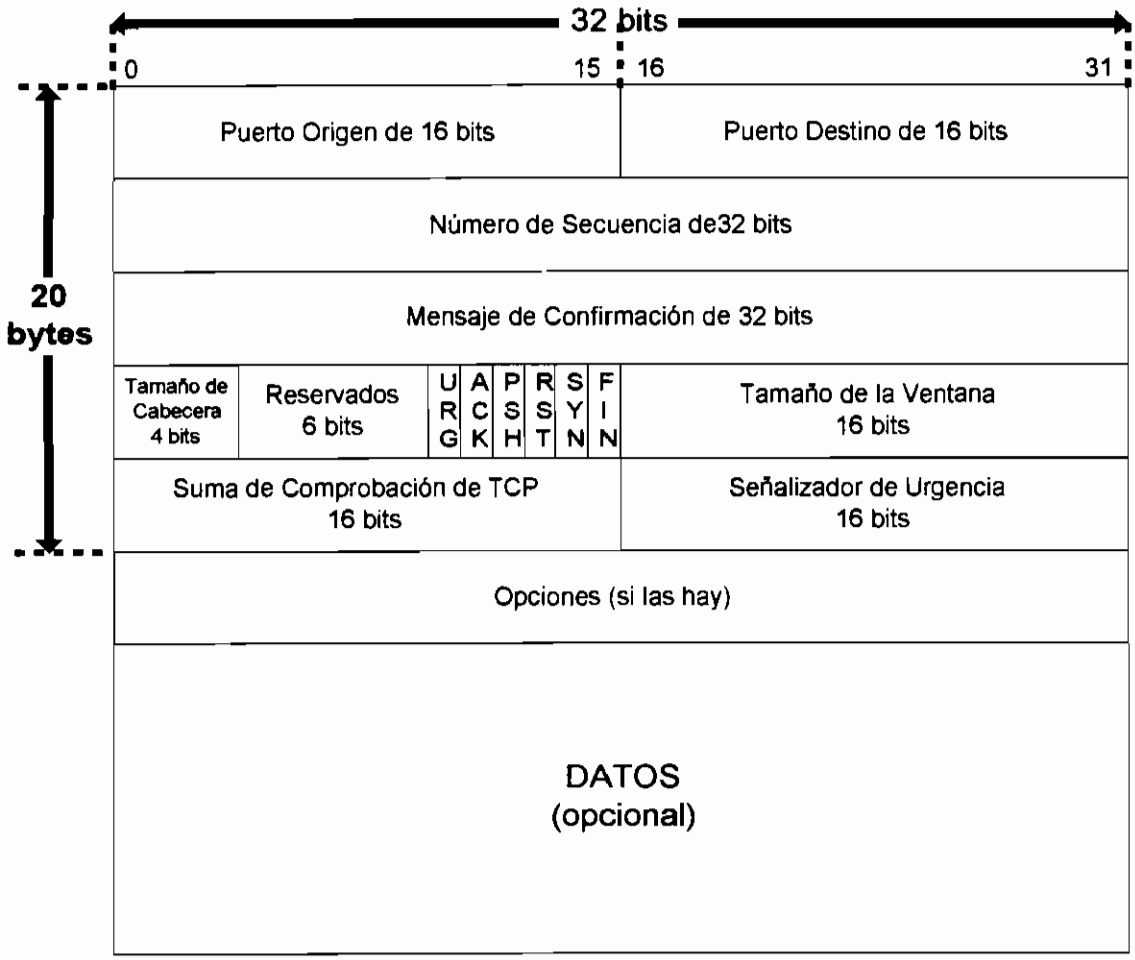


Figura.1.4 Estructura de un segmento TCP.

- Al igual que el encabezado IP, el campo longitud o *tamaño de cabecera TCP* utiliza 4 bits para especificar el tamaño del encabezado TCP en palabras de 32 bits, también al igual que la cabecera IP, la de TCP es casi siempre de veinte bytes de longitud. El área de datos comienza inmediatamente después del encabezado TCP. Al examinar el campo longitud del encabezado, los módulos receptores TCP pueden calcular el inicio del área de datos como si fueran cuatro veces más bytes (palabras de 32 bits) desde el inicio del segmento TCP.

1.2.5.3 Banderas de la Cabecera TCP.

El encabezado TCP incluye seis campos de banderas de un bit, a continuación se da una breve descripción de cada bandera.

1. URG. Esta bandera se encarga de decir al módulo TCP receptor que el campo indicador de urgencia indica los datos urgentes (el módulo TCP debe procesarlos antes que cualquier otro).
2. ACK. Esta bandera informa al módulo TCP receptor que el campo número de confirmación contiene números de confirmación válidos. Como sabe, esta bandera ayuda a TCP a asegurar la confiabilidad de los datos.
3. PSH. Esta bandera indica al módulo TCP receptor que envíe de inmediato el segmento de datos a su aplicación destino. Casi siempre los buffer del módulo TCP ingresan los datos. Así, TCP no espera para enviar el segmento de datos a su aplicación destino hasta que el buffer alcanza un umbral específico. La bandera PSH indica al módulo TCP que no debe llevar al buffer el segmento de datos. Por ejemplo una aplicación Telnet por lo común establece esta bandera y, al hacerlo, fuerza a TCP a pasar los impulsos del teclado del usuario al servidor de Telnet. Esto ayuda a eliminar los retrasos producidos al presentar los caracteres que se reciben de vuelta al emisor, la mayoría de los usuarios de Telnet desean ver lo que escriben cuando escriben.
4. RST. Esta solicita al módulo TCP receptor que le restablezca la conexión, TCP envía mensaje con la bandera RST cuando detecta problemas en la conexión. La mayoría de las aplicaciones simplemente terminan al recibir esta bandera. Sin embargo, puede utilizarla para diseñar programas mejorados que se recuperan de colisiones de hardware o software.
5. SYN. Esta bandera indicada al módulo TCP receptor que sincronice los números de secuencia. Como sabe, TCP utiliza esta bandera para decir al módulo TCP receptor que el emisor se prepara para transmitir un nuevo flujo de datos.

6. FIN. Esta bandera dice al módulo TCP receptor que el emisor término de transmitir datos. La bandera FIN solo cierra el flujo datos en la dirección en la que viaja. El módulo TCP receptor también debe enviar un mensaje con la bandera FIN para completar el cierre de la conexión.

- El campo *tamaño de ventana*, de 16 bits, informa al módulo TCP receptor el número de bytes que el emisor está dispuesto a aceptar. TCP utiliza ventanas deslizantes de tamaño variable para mejorar la capacidad de transferencia y optimizar el ancho de banda de la red. El valor de este campo especifica el tamaño de la ventana deslizante. Lo común es que sea de varios miles de bytes.

- El campo *suma de comprobación TCP* incluye en sus cálculos los datos TCP. Se necesita que el emisor calcule e incluya la suma de comprobación en este campo; así mismo, requiere que los módulos TCP receptores revisen las sumas de comprobación cuando reciben los datos.

- El campo *indicador de urgencia*, de 16 bits, especifica una localización de byte en el área datos. El propósito de la bandera URG y del indicador de urgencia es informar al módulo TCP receptor que existe algún tipo de *datos urgentes* y señalarle donde se localizan. Sin embargo, nadie ha definido adecuadamente el término de datos urgentes, ni la responsabilidad del módulo TCP receptor respecto al manejo de estos. Quizá lo más significativo es el hecho de que lo representado por esta localización de byte (incluso su ubicación) es tema de gran debate. Por el momento, tal vez convenga que se limite al usar el modo urgente de TCP.

- Al igual que el encabezado IP, el de TCP incluye el campo no obligatorio *opciones*. Durante las negociaciones iniciales de dos lados de una conexión TCP, los módulos TCP emplean, por lo común, el campo opciones, con la opción tamaño máximo del segmento. El

tamaño máximo del segmento de TCP es similar a la unidad de transferencia máxima MTU de la capa física: define el tamaño máximo de mensaje que puede aceptar el módulo TCP. TCP optimiza el ancho de banda de la red incrementando la capacidad de transferencia. La opción tamaño máximo de segmento permite a los módulos TCP anunciar el tamaño mayor del segmento o mensaje que espera recibir. Los módulos TCP sólo pueden utilizar esta opción en un mensaje que contenga establecida la bandera SYN. No obstante esta bandera no es una opción negociable. Uno de los lados de la conexión TCP anuncia al otro que espera a un tamaño máximo de segmento de cierto valor. Si un módulo TCP no transmite un tamaño máximo del segmento, TCP supone, por omisión, un tamaño máximo de 536 bytes*.

1.2.5.4 Conexión en el Protocolo TCP.

Antes de que TCP establezca una conexión oficial debe ocurrir un "saludo inicial" en tres pasos:

1. El módulo TCP del lado cliente pide una conexión TCP enviando una solicitud de sincronización y un número de secuencia inicial.
2. El módulo TCP en el lado servidor confirma la solicitud de conexión y, al mismo tiempo, solicita que el lado cliente se sincronice con su número de secuencia inicial.
3. El módulo TCP de lado cliente confirma la solicitud de sincronización del lado servidor.

Tras este saludo inicial en tres pasos, ambos lados de la conexión TCP tienen toda de información que necesitan para identificar los datos en el canal de comunicación: los números de secuencia y de confirmación.
En otras palabras, han sincronizado sus números de secuencia y confirman la sincronización.

Las conexiones TCP son full-duplex, de tal manera que los datos viajan en ambas direcciones al mismo tiempo, es decir, el flujo de datos en una

*TCP/IP Cap10 Sidnie Feit McGrawHill 1998

dirección es independiente del flujo de datos en la otra. Gracias a la capacidad full-duplex de TCP, cada lado de una conexión TCP debe mantener 2 números de secuencia, uno para cada dirección en la que circulan los datos. Por lo tanto se requiere que los módulos TCP terminen el flujo de datos en ambas direcciones antes de cerrar la conexión TCP. Cuando TCP hace un *medio cierre*[*], concluye el flujo de datos solo en una dirección.

La norma original de TCP está definida en la RFC 793. En la RFC 1122 se especifican actualizaciones, correcciones y requisitos de conformidad.

1.2.6 REVISIÓN DE LA API WINDOWS SOCKETS.

Una *interfaz de programa de aplicación* (API, Aplication Program Interface) es simplemente un grupo de funciones que los programadores utilizan a fin de desarrollar programas de aplicación para un ambiente de cómputo específico. Por ejemplo, para que los desarrolladores de software que desean crear aplicaciones para Windows, Microsoft proporciona una extensa colección de rutinas de programación: la API de Windows.

El socket es una representación abstracta de un extremo (punta) de la comunicación por red. La interfaz de socket es una API para redes TCP/IP, es decir, define una variedad de funciones o rutinas que permiten a los programadores desarrollar aplicaciones para utilizarlas en redes TCP/IP.

A fin de emplear una interfaz de socket para la comunicación en red, los programas siguen un proceso sencillo de cuatro pasos:

1. El programa crea un socket.
2. Luego lo configura para usarlo, es decir, el programa conecta el socket a un anfitrión remoto o lo enlaza a un puerto de protocolo local.
3. El programa transmite o recibe datos por medio de los sockets conforme lo solicite la aplicación.
4. Por último, cuando termina, el programa cierra el socket.

---

[*] Cap5. PROGRAMACION EN INTERNET. K Jamsa/K Cope McGrawHill 1996.

La estructura de datos del socket incluye elementos para almacenar valores de los parámetros de la función socket; entre ellos están cuatro elementos como son: dirección IP local, dirección IP remota, puerto local y puerto remoto. La función socket reserva un espacio y crea la estructura de datos del socket pero deja vacíos los elementos para las direcciones; además devuelve un identificador de socket que puede utilizar en el programa para configurar el mismo. Para asociar un socket con direcciones de red específicas debe llamar a otras funciones dentro de la API de sockets.

La API Windows sockets (mas conocida como Winsock) se deriva de la API de la interfaz de sockets de Berkeley. Windows Sockets implementa la interfaz de sockets como una biblioteca de enlace dinámico (DLL, Dynamic Link Library), que se sitúa entre la pila de protocolos TCP/IP y las aplicaciones del usuario, esto es, Winsock maneja la interfaz con los protocolos TCP/IP.

En Windows, una *biblioteca de enlace dinámico(DLL)*, proporciona un modo estándar para agregar nuevas funcionalidades que pueden emplear los programadores durante su ejecución. Una biblioteca de enlace dinámico es un módulo de código ejecutable que Windows puede cargar cuando se le requiera (conforme los programas necesiten usar las funciones que contiene esa biblioteca).Windows también puede descargar una DLL cuando los programas ya no necesiten los módulos de código que almacena la biblioteca. El diseño de la mayoría de las DLL de Windows permite que múltiples programas las usen al mismo tiempo.

La API Winsock proporciona una colección (o biblioteca) de funciones que pueden utilizar los programas para lograr tareas específicas. La especificación de Winsock organiza la biblioteca en tres grupos:

- Las funciones de los socket de Berkeley incluidas en la API.
- Funciones de base de datos permiten que estos programas obtengan información de Internet acerca de los nombres de dominio, servicios de comunicaciones y protocolos.

♦ Las extensiones específicas de Windows a las rutinas de los sockets de Berkeley.

Una función de bloqueo[1] evita que su programa llame a cualquiera otra función de Winsock hasta que la propia función de bloqueo termine sus operaciones en la red. Una función de no bloqueo termina de inmediato su operación, o regresa con un mensaje de error; en otras palabras, una función de no bloqueo[2] no espera hasta que la operación concluya.

A continuación se mencionan las funciones estilo Berkeley que pueden bloquear en la API Winsock. Estas *son accept, closesocket, recv, recvfrom, select, send, sendto.*

Las funciones estilo Berkeley que no bloquean en la API Winsock son: *bind, getpeername, getsockname, getsockopt, htonl, htons, inet_addr, inet_ntoa, ioctlsocket, listen, ntohl, ntohs, setsockopt, shutdown, socket.*

Las funciones de *base de datos* de Winsock permiten que sus programas obtengan información de Internet sobre nombres de dominio, servicios de comunicación y protocolo, son: *gethostbyaddr, gethostbyname, getprotobyname, getprotobynumber, getservbyname, getservbyport.*

La API Winsock incluye versiones asíncronas (específicas de Windows) de todas las funciones de base de datos, excepto de la función *gethostbyname.* Los desarrolladores de Winsock diseñaron las funciones asíncronas para aprovechar las facilidades basadas en mensajes inherentes a Windows.

Los desarrolladores de Winsock diseñaron versiones asíncronas especiales de ciertas funciones de sockets para que los programadores pudieran aprovechar el despliegue de mensajes dentro de Windows, enseguida se mencionan estas funciones de extensión específica de Microsoft Windows incluidas en la API Winsock, excepto las funciones de base de datos

[1] [2] Cap7.Cap8 PROGRAMACION EN INTERNET. K Jamsa/K Cope McGrawHill 1996.

asíncronas: WSAAsyncSelect, WSACancelAsyncRequest, WSACancelBlockingCall, WSACleanup, WSAStartup, WSAGetLastError, WSAIsBlocking, WSASetBlockingHook, WSASetLastError, WSAUnhookBlockingHook.

1.2.6.1 Creación de un socket.

A fin de usar un socket para la comunicación en red, en un programa se debe primero crearlo. Para hacerlo, el programa llama a la función *socket*. La siguiente instrucción muestra un ejemplo de una llamada a la función socket:

socket_handle = socket (protocol_family, socket_type, protocol);

Esta función tiene tres parámetros: el primero hace referencia a la familia de protocolos de Internet que para el caso de TCP/IP es constante simbólica **PF_INET**. El segundo parámetro requerido por la función socket especifica qué tipo de comunicación desea, es decir que el tipo de socket que puede ser orientado a conexión o no orientado a conexión los posibles valores son representados por las constantes simbólicas: **SOCK_ STREAM** para flujo de bytes, **SOCK_DGRAM** para datagramas; la interfaz de sockets también define un tercer tipo llamado *socket básico (raw socket)* **SOCK_RAW** que permite a un programa utilizar los mismos protocolos de nivel inferior que la red utiliza comúnmente, como son IP e ICMP (Internet Control Message Protocol) "Protocolo de Internet de Mensajes de control".

El tercer parámetro le permite a la función *socket* especificar que protocolo usará en el socket. Para especificar el protocolo TCP debe usar la constante simbólica **IPPROTO_TCP**.

Cada socket requiere cinco piezas: información las direcciones IP para el anfitrión local y remoto, un puerto para los procesos locales y remotos y un protocolo para utilizarlo a través de la conexión. En la siguiente tabla.1.1 se muestran las funciones de la API Winsock que un programa utiliza para configurar un socket para la comunicación en red.

Uso del Socket	Información Local	Información Remota
Cliente Orientado a Conexión	Una llamada a la función *connect* almacena información local y remota la estructura de datos del socket	
Servidor Orientado a conexión	bind	Listen Accept
Cliente Sin conexión	bind	Sento
Servidor Sin conexión	bind	Recvfrom

Tabla 1.1 Funciones de WinSock para configurar la comunicación en Red.

Después de que un programa configura un socket puede utilizarlo para comunicación en la red. El proceso de comunicación incluyen enviar y recibir información. La interfaz de socket incluye funciones para hacer ambas tareas.

Funciones	**Descripción**
Send	Transmite información a través de un socket de conexión, usando banderas especiales para controlar el comportamiento del socket.
Sendto	Transmite información a una dirección anfitrión especificada en una estructura de dirección de un socket, usando un buffer sencillo para mensajes.
Recv	Recibe información de un socket conectado, usando banderas especiales para controlar el comportamiento de el socket.
Recvfrom	Recibe información de un socket y, de manera opcional, graba la dirección de la red del anfitrión que está enviando, usando un buffer sencillo de mensajes.

Tabla.1.2 Funciones de la API WinSock para Entrada/Salida de Red.

La principal diferencia entre las funciones de la tabla.1.2 es que las funciones *send* y *recv* solo se pueden usar en *sockets conectados*. Las funciones *sendto* y *recvfrom* pueden especificar direcciones de red, esto es, los programas pueden usar estas funciones en *sockets no conectados*. Además, sus programas pueden usar las funciones *sendto* y *recvfrom* en *sockets conectados*.

Winsock no restringe los identificadores de socket a valores positivos. En la especificación de Winsock define cómodamente a **socket** como un tipo datos sin signo. Winsock emplea la constante **INVALID_SOCKET** para identificar a los sockets inválidos, el uno negativo se interpreta como un tipo de información no asignada.

Winsocks define también una constante nueva **SOCKET_ERROR** (a la que define como -1), como el valor que identifica los errores de socket, que pueden ser reportados por la función WSAGetLastError.

1.2.6.2 Sockets básicos

Un socket básico permite que los programas "brinquen"[♠] la capa de transporte TCP\IP y tengan acceso a protocolos de nivel inferior, como IP e ICMP(protocolo de Internet de mensajes de control). Por lo tanto, deben hacer funciones de la capa de transporte, como el encapsulamiento de información, para la capa IP. En general, las operaciones de los socket básicos ocurren de la siguiente manera. Primero, el programa usa la función *socket* de Winsock para crear un socket básico pero, en vez de especificar SOCK_STREAM o SOCK_DGRAM para el tipo de comunicación, especifica **SOCK_RAW**. Además, el programa también debe especificar un protocolo específico.

Después, la implementación de Winsock añade un encabezado IP a cualquier información escrita en el socket. En el campo protocolo del

[♠] Cap14, PROGRAMACION EN INTERNET, K Jamsa/K Cope McGrawHill 1996.

encabezado IP añadido, la implementación de Winsock almacena el valor del protocolo que el programa especificó cuando creó el socket.

Cuando la implementación de Winsock recibe cualquier información para el protocolo especificado, Winsock pasa una copia (incluyendo el encabezado IP del datagrama) a todos los procesos que crean un socket básico para ese protocolo. Por ejemplo, si un programa crea un socket básico para el protocolo ICMP, este programa recibirá una copia de todos los paquetes de información de ICMP que llegan. El programa examina cada una de estas copias para determinar qué paquetes le pertenecen. En general, la capa de transporte haría esta operación pero, debido a que este programa usa un socket básico para saltar la capa de transporte ya no está disponible para asociar la información entrante con sockets o puertos de protocolo específicos.

Para crear un sockets básico debe hacer un esfuerzo de programación mayor que cuando crea sockets estándar. Para comenzar, debe definir sus propias estructuras de información para almacenar los datos del encabezado del datagrama, así como llenar estructuras con los valores correctos. Para crear con exactitud las estructuras de información requeridas, debe tener cabal comprensión del protocolo subyacente y su estructura de paquetes. En otras palabras, si omite campos, almacena los valores equivocados en un campo, o define un campo de un tamaño equivocado, el programa basado en sockets básicos fallará. Por otro lado, después de entender en la operación del protocolo y de información asociada, se puede usar un socket básico sin mucha dificultad. Debido a las definiciones de la estructura y a las instrucciones de asignación que requiere un socket básico, se debe escribir más líneas de código que las que escribiría para un socket estándar. Pero escribir líneas para un socket básico no es más difícil que para un socket estándar, sólo debe poner mucha atención a los detalles. Dicho de otro modo, es necesario asegurarse de que se esta definiendo la estructura de información de

acuerdo a la especificación del protocolo y verificar que se almacenan los valores correctos en ellas.

Los protocolos usan sumas de comprobación y códigos de redundancia cíclica para detectar la corrupción de datos. Las computadoras crean la suma de comprobación agregando el valor binario de cada carácter alfanumérico en un bloque de información. Para revisar si existe corrupción en los datos, la computadora receptora calcula una suma de comprobación nueva y la compara con la que calculó el módulo que trasmitió el mensaje. Si no coinciden existe un error. El algoritmo básico es bastante sencillo: el programa combina bytes adyacentes de información para formar valores enteros de 16 bits, luego suma estos valores y después calcula el complemento a uno de la suma. Este último es la suma de comprobación de Internet, la cual puede almacenar el programa en el campo suma de comprobación del protocolo.

Con un socket básico, los programas usan protocolos sin conexión que entregan los datos mediante datagramas.
La siguiente instrucción es de particular interés y muestra el prototipo de Winsock para la función *sendto*:
Result = sendto (socket_handle, message_buffer, buffer_length, special_flags, socket_address_structure, address_structure_length);

Esta función requiere de seis parámetros:

El primero *socket_handle* es el descriptor del socket que se obtiene luego de crear el socket, identifica el socket que debe usarse.

El segundo parámetro *message_buffer* es un buffer de mensajes donde se almacena la información o dato quieren enviarse.

El tercer parámetro *buffer_length* es la longitud del buffer y dice a Winsock el máximo número de bytes que pueden almacenar el buffer, es el tamaño de la información contenida en el buffer de envío.

El cuarto parámetro *special_flags* son banderas especiales, winsock proporciona dos banderas especiales que pueden usar para este parámetro la constante simbólica MSG_OOB[1] como valor de bandera si se necesita enviar información fuera de banda al puerto de protocolo (recuerde, que la información fuera de banda es información urgente que se debe procesar de inmediato.); la constante simbólica MSG_DONTROUTE como valor de la bandera si no se desea que la información esté sujeta a enrutamiento, no obstante se puede especificar un valor 0 (cero), en cuyo caso sólo copia información de la cola de datos entrantes de la capa de transporte al buffer de mensajes del programa.

El quinto parámetro *socket_address_structure* , estructura de la dirección del socket identifica la dirección destino. Antes de que se llame a la función *sendto* deben almacenar la información del anfitrión remoto en una estructura de datos de socket, y luego pasa un apuntador a esta estructura de dirección como quinto parámetro.

El sexto parámetro address_structure_length, longitud de la estructura de la dirección especifica la longitud en bytes de la dirección destino.

[1] Cap8 PROGRAMACION EN INTERNET, K Jamsa/K Cope McGrawHill 1996.

1.2.7 NUEVAS APIs PARA MONITOREO DE REDES.

1.2.7.1 Revisión de Antecedentes

Las aplicaciones para análisis de redes confían en un conjunto apropiado de primitivas para capturar paquetes, monitorear la red y más. Mientras casi todas las variantes de UNIX tienen módulos kernel que soportan al menos la captura de paquetes, las capacidades de Windows no lo hacen. Hay algunas APIs disponibles, cada una tiene su propio módulo kernel; sin embargo, estas sufren de severas limitaciones. Por Ejemplo la API *Netmon* no está disponible libremente y su extensibilidad es muy limitada; más aún ésta no permite el envío de paquetes.

El *driver del filtro IP* está disponible solamente para Windows 2000 y no soporta otros protocolos mas que IP; lo que permite el control de la caída de paquetes pero no permite el monitoreo y generación de los mismos.

PCAUSA [que esta disponible en www.rawether.net] ofrece un producto comercial que provee una interface para captura de paquetes incluye un filtro compatible BPF. Sin embargo la interfaz es de muy bajo nivel y no provee funciones abstractas, como la generación de filtros.

El crecimiento y difusión de Windows para tareas que tradicionalmente eran realizadas por estaciones de trabajo UNIX, hacen que el no disponer de estas características no sea un problema ignorable, ya que limita el número y la calidad de los análisis, herramientas de seguridad en dicha plataforma. Los esfuerzos por tener una arquitectura abierta se enfoquen a la creación de una arquitectura poderosa y extensible de bajo nivel para el análisis de redes en la plataforma Win32, dio como resultado la API llamada WinPcap.

Esta arquitectura es el primer sistema abierto[1] para captura de paquetes en Win32 y cierra una importante brecha entre UNIX y Windows. Más aún

[1] http://netgroup-serv.polito.it/winpcap/docs/defaul.htm

WinPcap pone en primer lugar el rendimiento y así es capaz de soportar la demanda de la mayoría de las aplicaciones.

El filtrado de paquetes es hecho por un componente en modo kernel (para seleccionar paquetes) y una librería en modo usuario (para entregar los paquetes a las aplicaciones). Este último componente provee una interfaz estándar para acceso de bajo nivel a la red y permite a los programadores evitar la programación en el nivel kernel. WinPcap incluye un controlador optimizado en modo kernel, llamado filtro de paquetes Netgroup (NPF) siglas en inglés de Netgroup Packet Filter, y un conjunto de librerías de nivel de usuario que son compatibles con libpcap. NPF es parte de la pila de protocolos e interactúa con el Sistema Operativo como cualquier otro protocolo de red.

El objetivo primario de la API compatible Libpcap [desarrollado en la universidad de Berkeley] fue crear un conjunto de funciones de plataforma cruzada para la captura de paquetes. WinPcap hace fácil exportar aplicaciones UNIX a Win32 y habilita a la vez un gran conjunto de programas a ser usados en Win32, luego de una simple recompilación. Más aún, por la importancia del monitoreo del tráfico WinPcap provee un sistema de llamadas altamente específico para ésto.

1.2.7.2 Introducción

PACKET.DLL es una biblioteca o librería de enlace dinámico que hace de interfaz entre las aplicaciones a nivel de usuario y el controlador de captura de paquetes. La DLL implementa un conjunto de funciones que hacen más simple la comunicación con el controlador. Esto evita usar llamadas al sistema o IOCTLs (controles de entrada/salida) en programas de usuario. Es más, proporciona funciones para manejar adaptadores de red, leer y escribir paquetes de red, configurar buffer y filtros en el controlador, y más. Hay dos versiones de PACKET.DLL: la primera trabaja en Windows 95/98, la segunda en Windows NT/2000. Las dos versiones poseen la misma interfaz de programación, haciendo fácil de escribir

aplicaciones de captura independientes del sistema operativo. Usando la API PACKET.DLL, la misma aplicación puede correrse en Windows 95, 98 NT y 2000 sin ninguna modificación. A continuación se describe cómo usar **PACKET.DLL**, dando los detalles de las funciones y estructuras de los datos exportados por la DLL.

Si se está escribiendo una aplicación de captura y no se tiene ningún requerimiento particular de bajo nivel, se recomienda que se usen las funciones de wpcap.dll, que es un superset de la librería de captura de paquetes (libpcap), en lugar de la API PACKET.DLL; wpcap.dll usa las funciones de PACKET.DLL también, pero proporciona un ambiente de programación más poderoso, inmediato y fácil para usar. Con wpcap.dll operaciones como capturar un paquete, crear un filtro de captura o guardar en un archivo una descarga se ejecutan con seguridad; y su uso es intuitivo. Libpcap puede proporcionar todas las funciones estándar necesitadas por un monitor normal de la red (sniffer). Es más, los programas escritos para usar libpcap, se compilan fácilmente en UNIX debido a la compatibilidad entre Win32 y versiones de UNIX de esta biblioteca.

Sin embargo, la **API PACKET.DLL** ofrece algunas posibilidades que no son dadas por libpcap. Libpcap fue escrito para ser exportable y ofrecer una API de captura dependiente del sistema por consiguiente no puede aprovecharse de todas las posibilidades ofrecidas por el controlador. En este caso se requerirán algunas funciones de **PACKET.DLL**.

1.2.7.3 Descripción

WinPcap es una arquitectura para captura de paquetes y análisis de redes TCP/IP para plataformas Win32. Esta incluye un filtro de paquetes a nivel kernel, una biblioteca de enlace dinámico de bajo nivel (**packet.dll**), una librería de alto nivel independiente del sistema (Wpcap.dll basada en libpcap).

Packet.dll es una API que puede ser usada para accesar directamente a las funciones del controlador **packet.dll**, ofreciendo una interfaz de programación independiente del sistema operativo Windows de Microsoft. *Controlador* es sólo un módulo de software que proporciona una interfaz con una pieza de hardware especifica en este caso la tarjeta de red.
Wpcap.dll exporta un conjunto de primitivas de captura de alto nivel que son compatibles con el libpcap, la famosa librería de captura de UNIX. Estas funciones permiten capturar paquetes independientemente del sistema operativo y del tipo de hardware de la red.

WinPcap.dll exporta dos conjuntos de llamadas:

- Un conjunto de funciones de bajo nivel del controlador **packet.dll**, usada para enviar y recibir paquetes en modo "básico"(raw).
- Un conjunto de funciones de alto nivel para captura de paquetes que es un super-conjunto de las funciones de la librería libpcap de UNIX.

En adelante se llamará a la **API Packet Driver** o **Packet.dll** como el primer conjunto de funciones, mientras **wpcap.dll** o **libpcap** se refiere a una API más abstracta que es equivalente a una librería libpcap exportada de UNIX.

Los requisitos mínimos para compilar el controlador de dispositivo son los siguientes:
Sistema operativo: Windows 95/98
Driver Developer Kit (DDK) for Windows 95
Software Development Kit (SDK)
Visual C++ 6.0

1.2.7.4 La estructura de la pila de captura

Para capturar paquetes transferidos por una red, una aplicación de captura necesita interactuar directamente con el hardware de la red. Por esta razón el sistema operativo debe ofrecer un conjunto de primitivas de captura para comunicar y recibir directamente datos del adaptador de red.

La meta de estas primitivas es básicamente capturar los paquetes de la red (escondiendo la interacción con el adaptador de la red), y los transfiere a las llamadas de los programas. De ésta manera es dependiente del sistema, y la aplicación es bastante diferente en los varios sistemas operativos.

La sección de captura de paquetes del kernel debe ser rápida y eficaz porque debe poder también capturar paquetes en LANs de gran velocidad con tráfico pesado, limitando la pérdidas de paquetes y usando una cantidad pequeña de recursos del sistema. También debe ser general y flexible para ser usado por diferentes tipos de aplicaciones (analizadores, monitores de red, aplicaciones de prueba de red).

La aplicación de captura a nivel de usuario recibe paquetes del sistema, los interpreta, procesa, y los muestra al usuario de una manera comprensible.

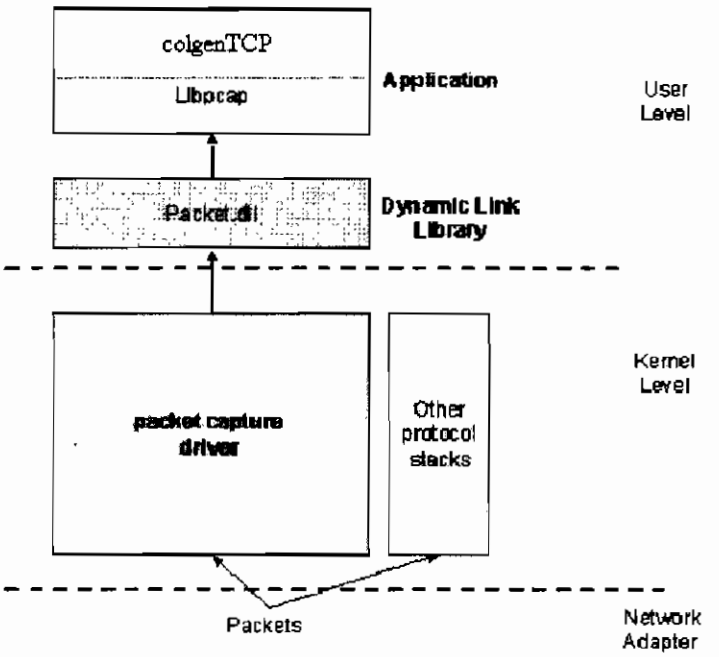


Figura. 1.5 Estructura de la pila de captura.

Debe ser fácil usar, independiente del sistema, modular y extensible para apoyar el mayor número de protocolos. Es más debe permitir aumentar el

número de protocolos decodificados de manera simple. Estos características son esenciales debido al alto número de protocolos de la red actualmente disponibles y la velocidad con que ellos cambian, así la integridad y expancibilidad son importantes.

Al nivel más bajo esta el adaptador de red. Se usa para capturar los paquetes que circulan en la red. Durante una captura el adaptador de la red trabaja normalmente en un modo particular (*'el modo promiscuo '*)[1] eso le obliga a que acepte todos los paquetes y no sólo los dirigidos a él.

El controlador de Captura de paquete es el módulo del software de más bajo nivel de la pila de captura. Es la parte que trabaja al nivel del kernel y actúa directamente con el adaptador de la red para obtener los paquetes. Proporciona a las aplicaciones que un conjunto de funciones para leer y escribir datos de la red al nivel de enlace de datos.

PACKET.DLL trabaja al nivel de usuario, pero está separado del programa de captura, como se puede observar en la figura 1.5.
Es una biblioteca de enlace dinámica que aísla el programa de captura del controlador proporcionando una interfaz de captura independiente del sistema.

La biblioteca o librería pcap, o libpcap, es una biblioteca estática que es usada para la captura de paquetes por parte de las aplicaciones. Usa los servicios exportados por PACKET.DLL, y proporciona a las aplicaciones un nivel más alto y una interfaz captura poderosa. Notar que esta estáticamente enlazada, es decir es parte de la aplicación que lo usa.

La interfaz del usuario es la parte más alta del programa de la captura. Maneja la interacción con el usuario y despliega el resultado de una captura.

---

[1] Develpment of on Architecture for Packet Capture, and Network Traffic Analysis. Loris Degioanni,Politecnico Di Torino (Turin, Italy, Mar 2000).

La estructura básica del controlador se muestra en la figura.1.6.
Las flechas hacia arriba en la figura representan el flujo de paquetes desde una red de computadoras a la aplicación de captura. La flecha más grande entre el buffer kernel y la aplicación indica que más de un paquete puede pasar entre estas dos entidades en una sola llamada de lectura del sistema. Las flechas hacia el fondo de la figura indican el camino de los paquetes de la aplicación a la red. La flecha gruesa entre el buffer del kernel y la aplicación indica que, en particular las situaciones, más de un paquete puede pasar entre estas dos entidades en una sola llamada de escritura del sistema.

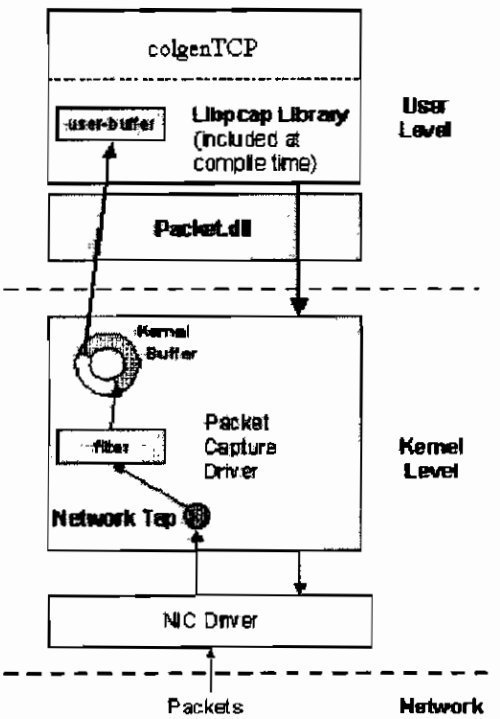


Figura.1.6 Estructura del driver

Además se ha incluido la posibilidad de escribir paquetes que pueden pasarse a través del uso directo de la biblioteca de enlace dinámico PACKET.DLL .

La estructura mostrada en Figura 1.6 es una descripción simplificada del controlador de paquete o Packet Driver.

1.2.7.5 Estructuras de los datos

Estructuras de los datos definidas en **packet32.h** son:

- PACKET
- ADAPTER
- bpf_insn
- bpf_program
- bpf_hdr
- bpf_stat
- NetType

Los dos primeros son controladores específicos de paquetes, mientras que los otros fueron originalmente definidos en **libpcap**. Este segundo conjunto de estructuras se usa para hacer operaciones tales como la configuración de un filtro o interpretación de los datos que vienen del controlador. El controlador de dispositivo de hecho usa la misma sintaxis de BPF para comunicarse con las aplicaciones, así que la estructura usada es la misma. Una estructura extensa, PACKET_IOD_DATA, se define en el archivo **ntddpack.h**

La estructura PACKET

La estructura PACKET tiene los campos siguientes:

- PVOID *Buffer*
- UINT *Length*
- PVOID *Next*
- UINT *ulBytesReceived*

Buffer es un puntero a un buffer que contienen los datos del paquete;

Length indica el tamaño de este buffer.

UlBytesReceived indica la longitud de la porción del buffer de que contiene datos válidos.

La estructura ADAPTER

Esta estructura describe un adaptador de red. Tiene cuatro campos:

- HANDLE *hFile*
- TCHAR *SymbolicLink*
- int *NumWrites*
- HANDLE *ReadEvent*

hFile es un puntero al manejador del controlador, una aplicación necesita conocer su manejador para comunicarse directamente con el controlador de red. En todo caso, este procedimiento se ha discontinuado porque PACKET.DLL ofrece a un conjunto de funciones para hacerlo.

SymbolicLink es una cadena de caracteres que contiene el nombre del adaptador de red actualmente abierto.

NumWrites es para uso interno y debe ser considerado opaco por el usuario.

ReadEvent es un evento de la notificación asociado con las llamadas leídas en el adaptador. Puede pasarse como parámetro a las funciones Win32 estándar (como WaitForSingleObject o WaitForMultipleObjects) para esperar hasta que el buffer del controlador contenga algunos datos sin realizar una llamada de lectura, y es particularmente útil en aplicaciones de GUI que necesitan atender concurrentemente a varios eventos. En Windows NT/2000 la función PacketSetMinToCopy () puede usarse para definir la cantidad mínima de datos en el buffer del kernel que causará el evento señalado.

La estructura PACKET_OID_DATA

Esta estructura se usa para comunicarse con el adaptador de la red a través de la consulta OID y un conjunto de funciones. Tiene tres campos:

- ULONG *Oid*
- ULONG *Length*
- UCHAR *Data*

Oid es un identificador numérico que indica el tipo de consulta y conjunto de funciones que se ejecutarán en el adaptador a través de la función de PacketRequest. Se definen posibles valores en el archivo incluido

ntddndis.h. Puede usarse, por ejemplo para recuperar el estado de los contadores de errores en el adaptador, sus direcciones MAC (Media Access Control), la lista de los grupos multicast definidos en él, y más.
El campo *Length* indica la longitud del campo *Data* que contiene la información pasada o recibida del adaptador.

La estructura bpf_insn

Esta estructura contiene una sola instrucción para la máquina-registro de BPF. Se usa para enviarle un programa del filtro al controlador. Tiene los campos siguientes:

- USHORT *code*
- UCHAR *jt*
- UCHAR *jf*
- int *k*

El campo *code* indica el tipo de la instrucción y los modos de direccionamiento.
Los campos *jt* y *jf* son usados por las instrucciones de salto condicional y son los desplazamientos de la próxima instrucción a los objetivos verdaderos y falsos.
El campo *k* es un campo genérico usado para varios propósitos.

La estructura bpf_program

Esta estructura apunta a un programa de filtro BPF, y es usado por la función de PacketSetBPF para configurar un filtro en el controlador. Tiene dos campos:

- UINT *bf_len*
- struct bpf_insn **bf_insns*;

El campo *bf_len* indica la longitud del programa.
Bf_insns es un puntero a la primera instrucción del programa.
La función PacketSetBPF configura un nuevo filtro en el controlador a través de una llamada IOCTL con el control *code* configurado a pBIOCSETF; una estructura bpf_program se pasa al controlador durante esta llamada.

La estructura bpf_hdr

Esta estructura define la cabecera usada por el controlador para entregar un paquete a la aplicación. La cabecera se encapsula con los bytes del paquete capturado, y se usa para mantener información como la longitud y marcas de tiempo (timestamp) para cada paquete. Es el mismo usado por BPF en UNIX. La estructura del bpf_hdr tiene los campos siguientes:

- struct timeval *bh_tstamp*
- UINT *bh_caplen*
- UINT *bh_datalen*
- USHORT *bh_hdrlen*

bh_tstamp guarda las marcas de tiempo (timestamp) asociadas con el paquete capturado. El timestamp tiene el mismo formato usado por BPF y se guarda en una estructura TimeVal que tiene dos campos:

tv_sec: fecha de la captura en el formato de tiempo estándar de UNIX (número de segundos desde 1/1/1970)

tv_usec: microsegundos de la captura.

Bh_caplen indica la longitud de porción capturada.

Bh_datalen es la longitud original del paquete.

Bh_hdrlen es la longitud de la cabecera que encapsula el paquete.

La estructura bpf_stat

Esta estructura es usada por el controlador para devolver las estadísticas de una sesión de captura. Tiene dos campos:

UINT *bs_recv*

UINT *bs_drop*

bs_recv indica el número de paquetes que el controlador recibió del adaptador de la red desde el principio de la captura. Este valor incluye los paquetes perdidos por el filtro, y debe ser una cuenta de los paquetes transmitidos por la red a la que el adaptador esta conectado.

Bs_drop indica el número de paquetes que el controlador perdió desde el principio de la captura. Básicamente, un paquete se pierde cuando el buffer del controlador está lleno. En esta situación que el paquete puede no guardarse y el controlador lo rechaza.

Nota: *bs_drop* no toma en consideración los paquetes que están perdidos cuando la función de captura del controlador es demasiado lenta, es decir cuando el tiempo pasó para ejecutar la captura es más largo que el tiempo entre dos paquetes (esto puede pasar si el filtro es muy complejo, si la red es demasiado rápida o el procesador es demasiado lento). En esta situación la captura no se ejecuta, así que el controlador no registra que se ha perdido un paquete.

La estructura NetType

Esta estructura es usada por la función de PacketGetNetType para recibir del controlador la información sobre el tipo de adaptador actual. Esta tiene dos campos:

UINT *LinkType*

UINT *LinkSpeed*

Linktype indica el tipo del adaptador de red actual (vea PacketGetNetType para más información).

Linkspeed indica la velocidad de la red en Bits por segundo.

1.2.7.6 Las Funciones

PACKET.DLL proporciona un conjunto completo de funciones que pueden usarse para enviar y recibir un paquete, consulta y configura los parámetros de un adaptador de red, abre y cierra un adaptador, maneja la asignación dinámica de las estructuras PACKET, configura un filtro BPF, cambia el tamaño del buffer del controlador y finalmente recupera las estadísticas de la sesión de captura.

Las funciones disponibles son:

- PacketGetAdapterNames
- PacketOpenAdapter

- PacketCloseAdapter
- PacketAllocatePacket
- PacketInitPacket
- PacketFreePacket
- PacketReceivePacket
- PacketSetMinToCopy
- PacketSendPacket
- PacketResetAdapter
- PacketSetHwFilter
- PacketRequest
- PacketSetBuff
- PacketSetBpf
- PacketGetStats
- PacketGetNetType
- PacketSetReadTimeout
- PacketSetMode
- PacketSetNumWrites
- PacketGetNetInfo

ULONG PacketGetAdapterNames(PTSTR pStr, PULONG BufferSize)

Normalmente, ésta es la primera función que debe usarse para comunicarse con el controlador. Esta devuelve los nombres de los adaptadores instalados en el sistema, en el buffer de usuario asignado *pStr*. Después de los nombres de los adaptadores, el *pStr* contiene una cadena de caracteres que describen cada uno de ellos.

BufferSize es la longitud del buffer

Advertencia: el resultado de esta función se obtiene consultando el registro del sistema operativo, por consiguiente el formato del resultado en Windows NT/2000 es diferente de uno en Windows 95/98/ME. Windows 9x usa el método de código ASCII para guardar una cadena de caracteres, mientras Windows NTx usa (normalmente) UNICODE. Después de una llamada a PacketGetAdapterNames en Windows 95x, el *pStr* contiene una

cadena de caracteres de ASCII con los nombres de los adaptadores separados por un solo ASCII "\0", un doble "\0", seguidos por las descripciones de los adaptadores separadas por un solo ASCII "\0". La cadena de caracteres se termina por un doble " \0".

En Windows NTx, el *pStr* contiene los nombres de los adaptadores, en formato UNICODE , separado por un solo UNICODE "\0" (es decir 2 ASCII "\0"), un doble UNICODE "\0", seguidos por las descripciones de los adaptadores, en formato ASCII, separadas por un solo ASCII "\0". La cadena de caracteres es terminada por un ASCII doble "\0". Por ejemplo, en Windows 98 con caracteres ASCII se tiene un byte tras otro:

83'**S**' 73'**I**' 83'**S**' 78'**N**' 73'**I**' 67'**C**' 0'' 80'**P**' 80'**P**' 77'**M**' 65'**A**' 67'**C**' 0'' 0'' 83'**S**'.

LPADAPTER PacketOpenAdapter(LPTSTR AdapterName)

Esta función recibe una cadena de caracteres que contiene el nombre del adaptador a abrir y retorna el puntero a un objeto del ADAPTADOR (ADAPTER) propiamente inicializado. Los nombres de los adaptadores pueden ser obtenidos llamando la función PacketGetAdapterNames.

Nota: como ya se dijo, en Windows 95 el controlador de captura trabaja con código ASCII, y en Windows NT con UNICODE. Por consiguiente, *AdapterName* debe ser una cadena de caracteres ASCII en Windows 95, y una cadena de caracteres UNICODE en Windows NT. Esta diferencia no es un problema si la cadena de caracteres apuntada por *AdapterName* se obtuvo a través de la función PacketGetAdapterNames, porque devuelve los nombres de los adaptadores en el formato apropiado. Los problemas pueden presentarse en Windows NT cuando la cadena de caracteres se obtiene de funciones C ANSI como "scanf", porque ellas usan el código ASCII. Éste puede ser un problema importante al convertir aplicaciones de línea de comando desde UNIX. Para evitarlo se incluye en Windows NT una rutina PacketOpenAdapter para convertir cadena de caracteres ASCII a UNICODE. PacketOpenAdapter en Windows NT acepta el ASCII y el formato de UNICODE. Si una cadena de caracteres ASCII se recibe, es convertida a UNICODE por PACKET.DLL antes de pasarlo al controlador.

LPPACKET PacketAllocatePacket(void)

Asigna memoria a una estructura PACKET y retorna un puntero a la misma. La estructura retornada debe ser inicializada adecuadamente llamando la función de PacketInitPacket.

Advertencia: El campo Buffer de la estructura PACKET no es configurado por esta función. El buffer debe ser asignado por el programador, y debe ser asociado a la estructura PACKET con una llamada a la función PacketInitPacket.

VOID PacketInitPacket(LPPACKET lpPacket, PVOID Buffer, UINT Length)

Inicializa una estructura **PACKET**. Hay tres parámetros de entrada:

- **LpPacket** es un puntero a la estructura a ser inicializada.
- **Buffer** es un puntero al buffer de usuario asignado que contendrá los datos capturados
- **Length** es la longitud del buffer. Éste es el tamaño máximo del buffer que puede ser transferido del controlador a la aplicación usando una solo lectura.

Nota: El tamaño del buffer asociado con la estructura **PACKET** es un parámetro que puede influir sensiblemente en la performance del proceso de captura, puesto que este buffer contiene los paquetes recibidos desde el controlador. El controlador es capaz de devolver a la aplicación varios paquetes capturados usando una sola llamada de lectura (vea la función packetreceivepacket), y el número de paquetes transferidos a la aplicación en una sola llamada sólo está limitado por el tamaño del buffer asociado con la estructura **PACKET** usada para llevar a cabo la lectura. Por consiguiente poniendo un buffer grande con la PacketInitPacket se pueden disminuir el número de llamadas al sistema, mejorando la velocidad de la captura.

VOID PacketFreePacket(LPPACKET lpPacket)

Esta función libera la estructura **PACKET** apuntada por *lpPacket*.

Advertencia: El campo del Buffer de la estructura **PACKET** no es liberada por esta función y debe ser explícitamente liberada por el programador.

BOOLEAN PacketReceivePacket(LPADAPTER AdapterObject,
LPPACKET lpPacket, BOOLEAN Sync)

Esta función realiza la captura de un conjunto de paquetes. Tiene los parámetros de la entrada siguientes:

- **AdapterObject** es un puntero a una estructura ADAPTER que identifica el adaptador de red del que deben capturarse los paquetes.
- **LpPacket** es un puntero a una estructura PACKET que contendrá los paquetes.
- **Sync** es una bandera que indica si el funcionamiento se hará de una manera síncrona o asíncrona. Este parámetro está obsoleto y es ignorado por recientes versiones de PACKET.DLL, porque el acceso al controlador siempre es síncrono.

El número de paquetes recibido con esta función es variable. Realmente depende del número de paquetes guardado en el buffer del controlador, en el tamaño de estos paquetes y en el tamaño del buffer asociado con el parámetro lpPacket. La siguiente figura muestra el formato usado por el controlador para enviar paquetes a la aplicación.

Figura .1.7 Método usado para codificar los paquetes

Los paquetes se guardan en el buffer asociado con la estructura PACKET de *lpPacket*. Cada paquete tiene una cabecera que consiste en una estructura *bpf_hdr* que define su longitud y guarda sus marcas de tiempo (timestamp). Un campo del relleno se usa para alinear en palabras (WORD) los datos en el buffer (para aumentar la velocidad de las copias). Los campos *bh_datalen* y *bh_hdrlen* de la estructura bpf_hdr se deben usar para extraer los paquetes del buffer.

Es posible poner un intervalo de tiempo de espera en llamadas de lectura con la función PacketSetReadTimeout. En este caso la llamada vuelve aun cuando ningún paquete se haya capturado, si el tiempo de espera configurado por esta función expira.

BOOLEAN PacketSetMinToCopy(LPADAPTER AdapterObject,int nbytes)

Esta función puede usarse para definir la cantidad mínima de datos en el buffer del kernel que causará que el controlador libere una captura (es decir en un PacketReceivePacket) en ejecución.

Nbytes especifica este valor en bytes.

En la presencia de un valor grande para esta variable, el kernel (módulo central del sistema operativo) espera por la llegada de varios paquetes antes de copiar los datos al usuario. Esto garantiza un número bajo de llamadas del sistema, es decir el uso del procesador es bajo, o sea mejor rendimiento que es una buena configuración para las aplicaciones como los monitores de red (sniffers). Viceversa, un pequeño valor significa que el kernel copiará los paquetes tan pronto la aplicación esté lista a recibirlos. Esto se sugiere para las aplicaciones de tiempo real (como por ejemplo, un redirector de ARP) que necesita el mejor tiempo de respuesta del kernel.

Nota: esta función sólo tiene efecto en Windows NT/2000. El controlador para Windows 95/98/ME no ofrece esta posibilidad de modificar la cantidad de datos para liberar (desbloquear) una lectura, por consiguiente esta llamada se lleva a cabo bajo estos sistemas solamente por compatibilidad.

BOOLEAN PacketResetAdapter(LPADAPTER AdapterObject)

Restablece (resetea) el adaptador recibido como parámetro de la entrada. Retorna VERDADERO si la operación se realiza con éxito.

BOOLEAN PacketSetHwFilter(LPADAPTER AdapterObject, ULONG Filter)

Esta función pone un filtro del hardware en los paquetes entrantes. Se declaran las constantes que definen los filtros en el archivo ntddndis.h. Los parámetros de la entrada son: el adaptador en el cual el filtro debe definirse, y el identificador del filtro. El valor retornado es VERDAD si la operación tuvo éxito. Aquí esta una lista de los filtros más útiles:

- NDIS_PACKET_TYPE_PROMISCUOUS: pone el adaptador en modo promiscuo. Cada paquete entrante es aceptado por el adaptador.
- NDIS_PACKET_TYPE_DIRECTED: se aceptan sólo paquetes dirigidos al adaptador de la computadora.
- NDIS_PACKET_TYPE_BROADCAST: sólo los paquetes de difusión se aceptan.
- NDIS_PACKET_TYPE_MULTICAST: se aceptan sólo los paquetes del multicast que pertenecen a los grupos de los que este adaptador es miembro.
- NDIS_PACKET_TYPE_ALL_MULTICAST: cada paquete del multicast se acepta.
- NDIS_PACKET_TYPE_ALL_LOCAL: todos los paquetes locales, es decir:

  NDIS_PACKET_TYPE_DIRECTED+
  NDIS_PACKET_TYPE_BROADCAST+
  NDIS_PACKET_TYPE_MULTICAST

BOOLEAN PacketRequest(LPADAPTER AdapterObject, BOOLEAN Set, PPACKET_OID_DATA OidData)

Esta función se usa para realizar una consulta/configuración de la operación en el adaptador apuntado por *AdapterObject*. Con esta función es posible obtener o definir varios parámetros del adaptador de red, como

la dimensión de los buffers internos, la velocidad del enlace o el contador de paquetes corruptos.

El segundo parámetro **Set,** especifica si el funcionamiento es una configuración (set=TRUE) o una consulta (set=FALSE).

El tercer parámetro es un puntero a una estructura PACKET_OID_DATA (vea la sección estructura de datos). El valor del retorno es verdad si la función se completa sin errores. Se declaran las constantes que definen los funcionamientos en el archivo ntddndis.h.

NOTA: no todos los adaptadores de red llevan a cabo todas las funciones de consulta/configuración. Hay un conjunto de funciones obligatorias de OID que se dan para estar presente en todos los adaptadores, y un conjunto de funciones facultativas, no provistas por todos los adaptadores. Si usted usa una función facultativa, tenga cuidado para adjuntarlo en una declaración *if* para verificar el resultado.

BOOLEAN PacketSetBuff(LPADAPTER AdapterObject, int dim)

Esta función se usa para poner a un nuevo tamaño el buffer del controlador asociado con el adaptador apuntado por *AdapterObject*. ***Dim*** es la nueva dimensión en bytes. La función vuelve VERDADERO si se completó con éxito, FALSO si no hay suficiente memoria para asignar al nuevo buffer. Cuando una nueva dimensión es configurada, los datos en el buffer viejo se desechan y los paquetes que se guardaron en él se pierden.

Nota: la dimensión del buffer del controlador afecta fuertemente el performance del proceso de captura. Una aplicación de captura necesita hacer operaciones en cada paquete mientras el CPU es compartido con otras tareas. Por consiguiente la aplicación podría no trabajar a velocidad de la red durante tráfico pesado, picos o ráfagas, sobre todo en presencia de alta carga en el CPU debido a otras aplicaciones. Este problema es más notable en máquinas más lentas. El controlador, por otro lado, corre en

modo kernel y se escribe explícitamente para capturar paquetes, así que es muy rápido y normalmente no pierde paquetes. Por consiguiente, un buffer adecuado en el controlador es capaz de guardar los paquetes mientras la aplicación está ocupada, compensando así la lentitud de la aplicación y evitando la pérdida de paquetes durante picos, ráfagas o actividad alta de la red. El tamaño del buffer se pone a 0 (cero) cuando se abre una instancia del controlador: el programador debe recordar ponerlo a un valor apropiado.

Libpcap llama esta función y configura el tamaño del buffer a 1MB al principio de una captura. Por consiguiente programas escritos normalmente usando libpcap no necesitan afrontar este problema.

BOOLEAN PacketSetBpf(LPADAPTER AdapterObject, struct bpf_program *fp)
Esta función asocia un nuevo filtro BPF[1] al adaptador *AdapterObject*. El filtro, apuntado por *fp*, es un conjunto de instrucciones que la máquina-registro (motor) de BPF del controlador ejecutará en cada paquete entrante. Esta función retorna VERDAD (TRUE) si el controlador es configurado exitosamente, FALSO si ocurre un error o si el programa del filtro no es aceptado. El controlador realiza un chequeo en cada nuevo filtro para evitar la caída del sistema debido a programas fraudulentos o gusanos, y rechaza filtros inválidos.

Un filtro puede ser creado automáticamente usando la función pcap_compile de libpcap. Esta función convierte un filtro de texto con la sintaxis de WinDump (vea el manual de WinDump para más detalles) en un programa de BPF. Si usted no quiere usar libpcap, pero usted necesita saber el código de un filtro, usted puede ejecutar WinDump con los parámetros –d o –dd o –ddd para obtener el pseudocodigo del filtro.

[1] Los detalles sobre los filtros de BPF pueden encontrarse en [McCanne y Jacobson 1993]

BOOLEAN PacketGetStats(LPADAPTER AdapterObject, struct bpf_stat *s)

Con esta función, el programador puede saber el valor de dos variables internas del controlador:

- El número de paquetes que han sido recibidos por el adaptador *AdapterObject*, empezando en el momento en el que se abrió con PacketOpenAdapter.
- El número de paquetes recibidos por el adaptador pero que han sido dejados caer por el kernel. Un paquete se deja caer cuando la aplicación a nivel de usuario no está lista para recogerlos y el buffer del kernel asociado con el adaptador está lleno.

Los dos valores son copiados por el controlador en una estructura bpf_stat proporcionada por la aplicación. Estos valores son útiles para saber el estado de la red y el comportamiento de la sesión de captura.

BOOLEAN PacketGetNetType(LPADAPTER AdapterObject,NetType *type)

Retorna el tipo del adaptador apuntado por *AdapterObject* en una estructura NetType. El *LinkType* del parámetro de tipo puede ser configurado por esta función a uno de los valores siguientes:

- NdisMedium802_3: Ethernet (802.3)
- NdisMedium802_5: Token Ring (802.5)
- NdisMediumFddi: FDDI
- NdisMediumWan: WAN
- NdisMediumLocalTalk: LocalTalk
- NdisMediumDix: DIX
- NdisMediumAtm: ATM
- NdisMediumArcnetRaw: ARCNET (raw)
- NdisMediumArcnet878_2: ARCNET (878.2)
- NdisMediumWirelessWan: Various types of NdisWirelessXxx media.

Los controladores de captura BPF al momento soportan:

NdisMediumWan,

NdisMedium802_3,

NdisMedium802_5,

NdisMediumFddi,

NdisMediumArcnet878_2 y

NdisMediumAtm.

El campo *LinkSpeed* indica la velocidad de la red en bits por segundo.

El valor del retorno es VERDAD (TRUE) si la operación se realiza con éxito.

BOOLEAN PacketSetReadTimeout(LPADAPTER AdapterObject, int timeout)

Esta función configura el valor del tiempo de espera de lectura con el adaptador asociado *AdapterObject*. ***Timeout*** indica el nuevo tiempo de espera de captura (lectura) en milisegundos después del cual PacketReceivePacket () retornará (aun si ningún paquete ha sido capturado por el controlador). Poniendo ***timeout*** a 0 significa que no hay tiempo de espera, es decir PacketReceivePacket () nunca retorna si ningún paquete llega. Un ***timeout*** de –1 causa que siempre PacketReceivePacket () retorne inmediatamente.

Esta función también trabaja si el adaptador está trabajando en modo de estadística, y puede usarse para poner el intervalo de tiempo entre dos informes o reportes estadísticos.

BOOLEAN PacketSetMode (LPADAPTER AdapterObject, int mode)

Esta función pone al adaptador AdapterObject en el modo ***mode***, que puede tener dos posibles valores:

- MODE_CAPT: modo de la captura normal. Configuración por defecto luego de la llamada a PacketOpenAdapter.
- MODE_STAT: modo estadístico, un modo del funcionamiento particular del controlador de captura BPF que puede ser usado para realizar estadísticas en tiempo real del tráfico de la red. El controlador no captura nada cuando esta en modo estadístico y se limita a contar el número de paquetes y la cantidad de bytes que satisfacen el filtro de BPF definido por el usuario. Estos contadores pueden ser obtenidos por la aplicación con la función estándar PacketReceivePacket, y se reciben a intervalos regulares, cada vez que timeout expira. El valor predefinido de timeout es 1 segundo, pero puede ponerse a cualquier otro valor

(con una precisión de 1 ms) con la función de PacketSetReadTimeout. Los contadores se encapsulan en una estructura bpf_hdr antes de pasarse a la aplicación. Las capturas en este modo tienen un impacto muy bajo con el performance del sistema.

Una aplicación que quiere usar modo estadístico debe realizar las operaciones siguientes:

- Abrir el adaptador;
- Configurarlo en modo estadístico con PacketSetMode;
- Poner un filtro que define los paquetes que serán contados, con PacketSetBpf.
- Poner el intervalo de tiempo correcto con PacketSetReadTimeout;
- Recibir los resultados con PacketReceivePacket. Esta función devuelve el número de paquetes y la cantidad de bytes capturados que satisfacen el filtro en el último intervalo de tiempo. Estos valores son enteros de 64 bits y se encapsulan en una estructura bpf_hdr. Por consiguiente los datos retornados por PacketReceivePacket serán de 34 bytes longitud, y tendrán el formato que se muestra en la FIGURA 1.8.

Struct timeval **bh_tstamp**
UINT **bh_caplen**=16
UINT **bh_datalen**=16
USHORT **bh_hdrlen**=18
LARGE_INTEGER **PacketsAccepted**
LARGE_INTEGER **BytesAccepted**

Figura.1.8 Estructura de los paquetes capturados en el buffer

NOTA: si la interfaz está trabajando en modo estadístico, no hay ninguna necesidad del buffer kernel porque los valores estadísticos se calculan sin guardar ningún dato en él. Así que su dimensión puede ponerse a 0 con PacketSetBuff.

BOOLEAN PacketSetNumWrites(LPADAPTER AdapterObject,int nwrites)
Esta función pone a *nwrites* el número de veces de una solo escritura en el adaptador *AdapterObject* debe repetirse. Vea PacketSendPacket para más detalles.

BOOLEAN PacketGetNetInfo(LPTSTR AdapterName, PULONG netp, PULONG maskp)
Retorna las direcciones IP (en *netp*) y la mascara de red (en *maskp*) del adaptador cuyo nombre es especificado por *AdapterName*.

1.3 HERRAMIENTA DE PROGRAMACION.

Se utiliza un poderoso y potente Front End o entorno de programación integrado como es el Visual C++, este permite utilizar la *programación orientada a objetos* que tiene sus complicaciones con el manejo y uso del concepto de punteros, pero en todo caso útil y efectivo para nuestros propósitos. Una introducción a los fundamentos de programación de este lenguaje se presentan en el ANEXO A de este trabajo.

CAPITULO 2

DESARROLLO DEL PROGRAMA DE APLICACION

Como se indica en el capítulo anterior es en la sección del *Anexo A* donde se explica brevemente el entorno de programación Visual C++; se empieza a crear la aplicación con la ayuda de los WIZARD de programación. Pero antes se revisan cuales son los requerimientos del programa.

2.1 REQUERIMIENTOS DEL PROGRAMA DE APLICACION

Se busca crear una aplicación con las siguientes características:

- Que funcione bajo Windows 98.
- Debe tener dos modos de funcionamiento,
- Un modo para enviar datos (generador de paquetes).
- Un modo para recibir o capturar datos (colector de paquetes)
- Permitir ingresar el nombre del host o anfitrión remoto al que se quiere enviar el paquete.
- Mostrar datos predeterminados de un paquete a enviarse a la red
- Permitir ingresar nuevos valores en los campos del encabezado, cuando está en modo de envío.
- Mediante un botón CheckSum calcula de la suma de comprobación del paquete enviarse.
- Un botón de comando permitirá enviar el paquete una y otra vez si se lo sigue presionando.
- Por medio de un menú permitir ver el paquete enviado.
- Que permita seleccionar el adaptador de red de la computadora por medio de la cual se van a capturar los paquetes
- Que mediante un menú permita ver alguno de los seis primeros paquetes.
- Permitir ver la cabecera IP, que encapsuló al último paquete visto.
- Cuando se estén visualizando los paquetes solo se habilite el botón OK o Salir para retornar al programa principal.

- Los campos datos y opciones se han definido por simplicidad, del tipo Dword.
- Los valores permitidos para el campo longitud de cabecera están entre 5 y 15.
- Los valores permitidos para los campos de banderas son cero (0) o uno (1).
- Los campos aceptarán solo caracteres numéricos, que estén dentro de los limites de valores permitidos.
- El botón ayuda desplegará un documento básico sobre el uso de la aplicación, y una breve descripción de la estructura del Paquete TCP.

2.2 PROCEDIMIENTO PARA UTILIZAR LA LIBRERIA PACKET.DLL

Para utilizar la librería externa PACKET.DLL de WinPcap se deben seguir los siguientes pasos:

- Del archivo comprimido WPdpack.zip, se extrae el archivo ejecutable WinPcap.exe y se lo ejecuta.
- Del archivo comprimido WPcapsrc.zip, se extrae el directorio ...\WinPcap\Common, en éste se encuentran los archivos: packet32.h,packet.lib, etc.
- En la barra de herramientas: ....Tools\Options de visual C++, se selecciona la tabulación "directories", seleccionar:
- En el ListBox "Include Files" y "Library Files", añadir el directorio: ...\WinPcap\Common.

Ahora Visual C++ puede utilizar la nueva librería como un recurso de dependencia externa.

2.3 DIAGRAMA DE CLASES DE LA APLICACIÓN

A continuación se presenta el diagrama de clases de la aplicación ColgenTCP.

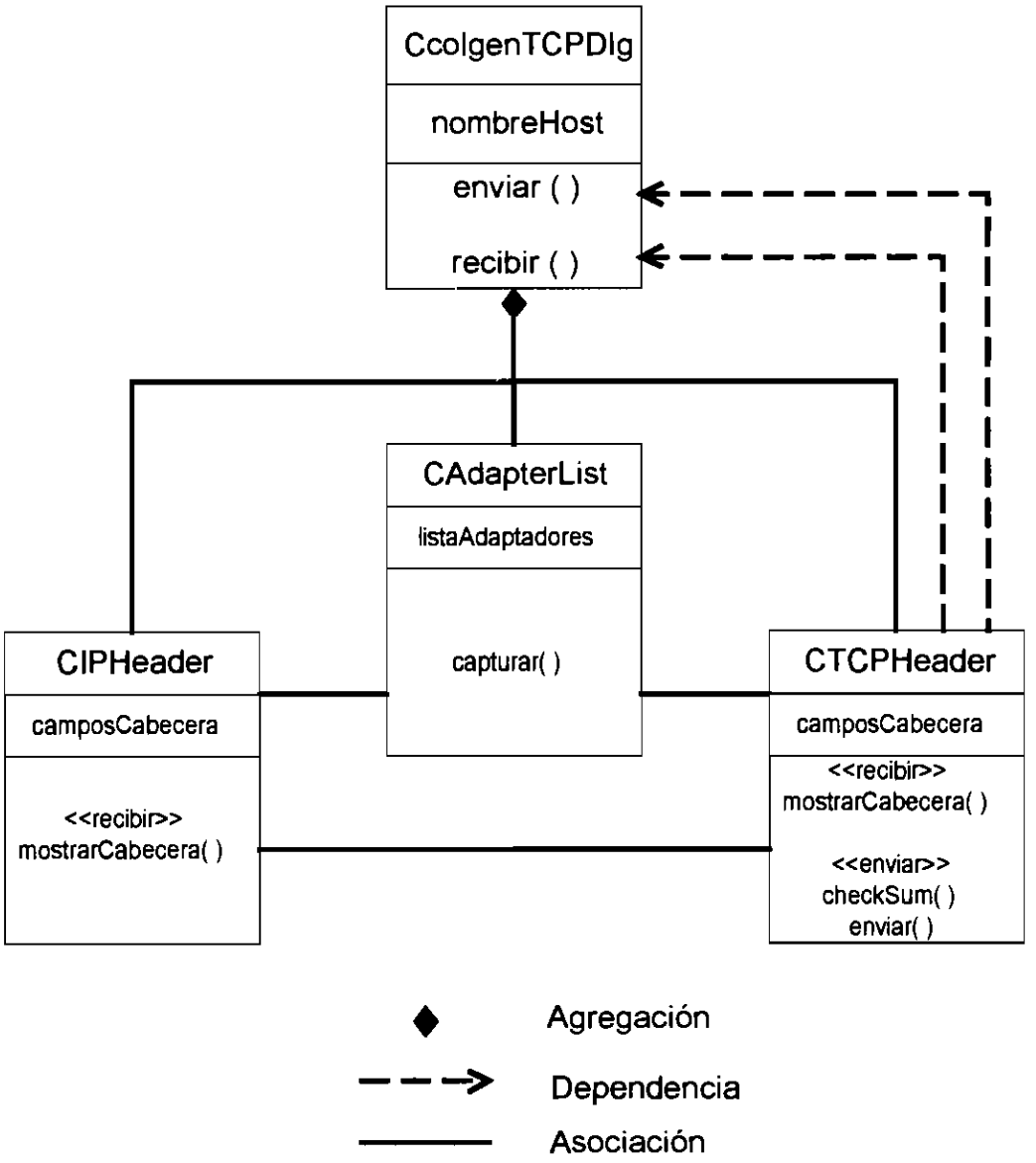


Figura 2.1 Diagrama de Clases

La visión actual de desarrollo de software toma una perspectiva orientada a objetos, en la cual el principal bloque de construcción es el objeto o clase. Para explicarlo sencillamente un *objeto* es una cosa; una *clase* es una descripción de un conjunto de objetos similares.

Gráficamente una clase se representa como un rectángulo, que normalmente incluye su nombre atributo y operaciones.

Una *dependencia* es una relación de uso que declara que un cambio en la especificación de un elemento puede afectar a otro elemento que la utiliza , pero no necesariamente a la inversa. Gráficamente se representa como una línea discontinua dirigida hacia el elemento del cual depende.

Una *asociación* es una relación estructural que especifica que los objetos de un elemento están conectados con los objetos de otro. Dada una asociación entre dos clases se puede navegar desde un objeto de una clase hasta un objeto de la otra clase, y viceversa. Gráficamente se representa como una línea continua.

La *agregación* es un tipo especial de asociación, que representa una relación estructural entre un todo y sus partes. Se representa agregando un rombo a la asociación en la parte del todo.

Como se puede observar en la figura 2.1 tenemos el diagrama de clases para la aplicación ColgenTCP.

2.4 CREACIÓN DEL PROYECTO

En primer lugar se realiza la creación del espacio de trabajo para el proyecto. En el menú se da un click en File | New, que abré el siguiente cuadro de diálogo en el cual en la ficha proyectos en el campo nombre de proyecto escribimos **colgenTCP**, y en la ventana de la izquierda

escogemos la opción MFC AppWizard (exe), como se indica en la figura 2.2.

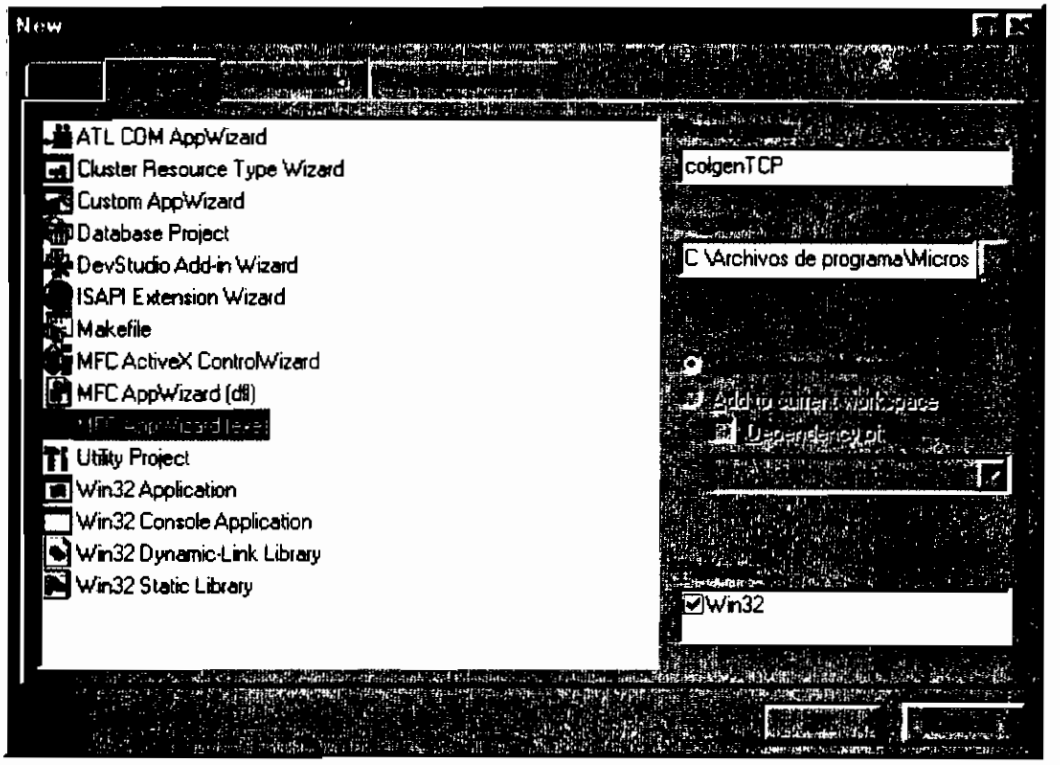


Figura.2.2 Cuadro de diálogo para nuevo proyecto.

Al dar un click en el botón OK inicia el AppWizard, que tiene varios pasos:
En el primer paso se escoge la opción: Dialog Based
En el segundo paso además de las opciones por defecto se habilitan las opciones para ayuda y para Windows sockets.
En los siguientes dos pasos se escogen las opciones por defecto y se da click en el botón finalizar. Entonces aparece un diálogo con las características del nuevo proyecto.

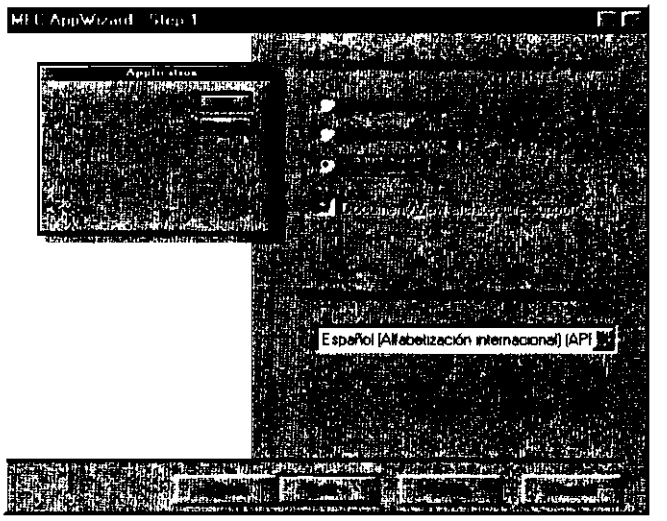


Figura.2.3 Primer Paso Crear Proyecto.

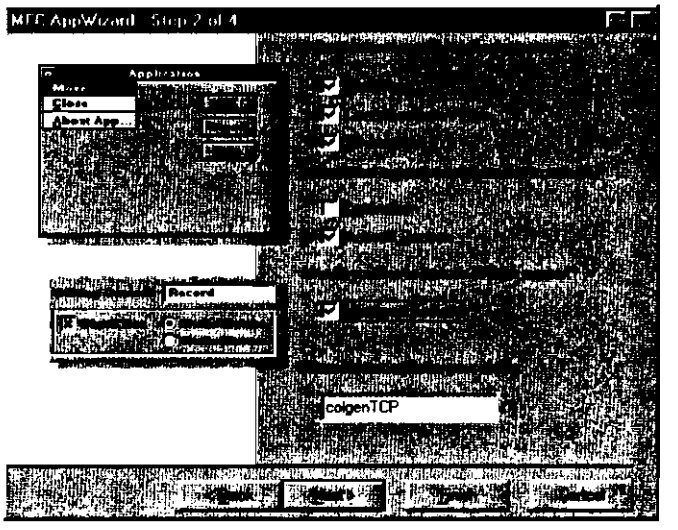


Figura.2.4 Segundo Paso Crear Proyecto.

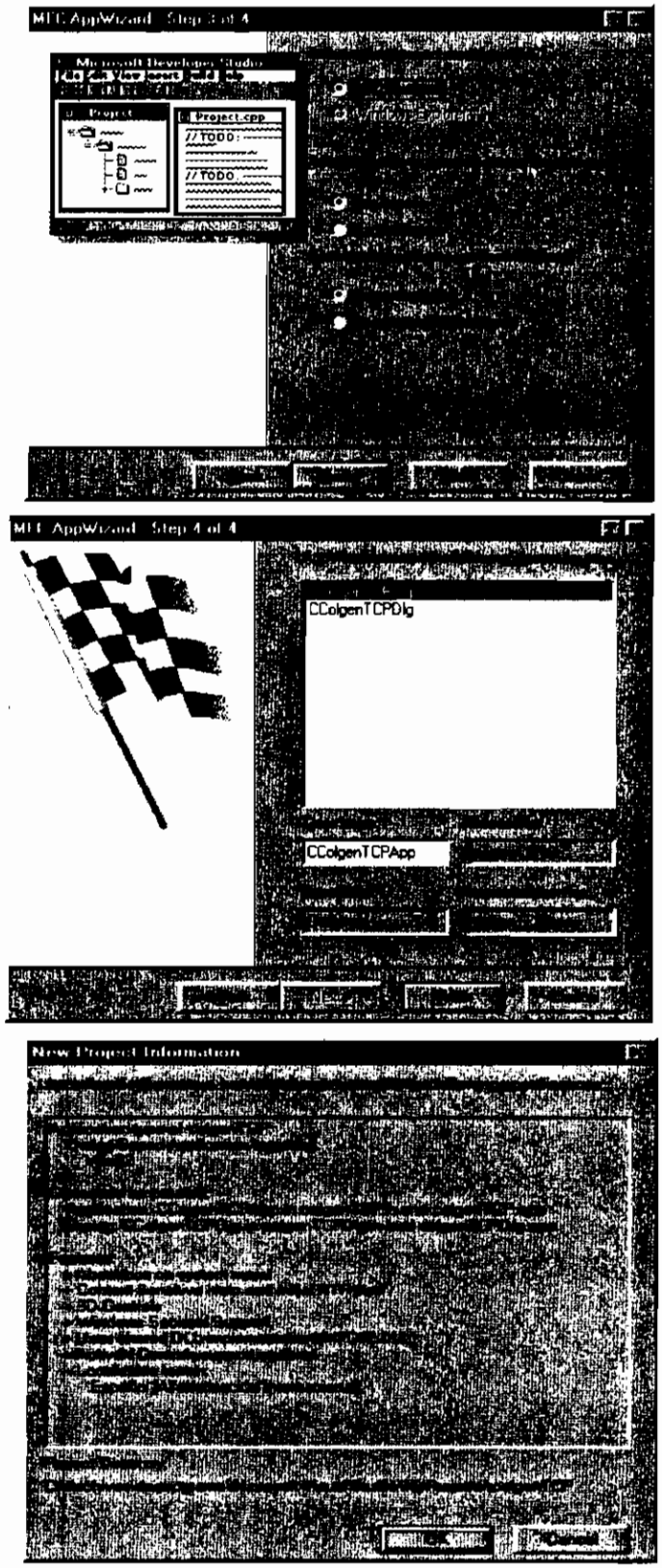


Figura.2.5 Secuencia Diálogos para nuevo proyecto.

2.5 DISEÑO DE LA INTERFAZ DE LA APLICACIÓN.

Luego se procede al diseño de la ventana de la aplicación colgenTCP.

Se ha modificado la caja de diálogo principal de tal manera que se tiene un control groupbox que contiene dos radiobutton y permite seleccionar el modo de operación del programa ya sea como generador o como colector. Se tienen además 3 command button, uno de los cuales sirve para ejecutar ya sea el envío o recepción de paquetes; el otro botón se usa para a cancelar y cerrar la aplicación; finalmente el tercer botón al ser presionado despliega una ayuda básica sobre el uso del programa.

En la interfaz además se tiene un control static text, que indica que en el editbox adjunto se debe ingresar el **nombre del anfitrión,** y otro que señala que en el listbox inferior se desplegarán los datagramas IP que encapsulen paquetes TCP.

Finalmente se tiene un menú, que permite **Ver** el paquete enviado, o los primeros seis paquetes TCP recibidos.

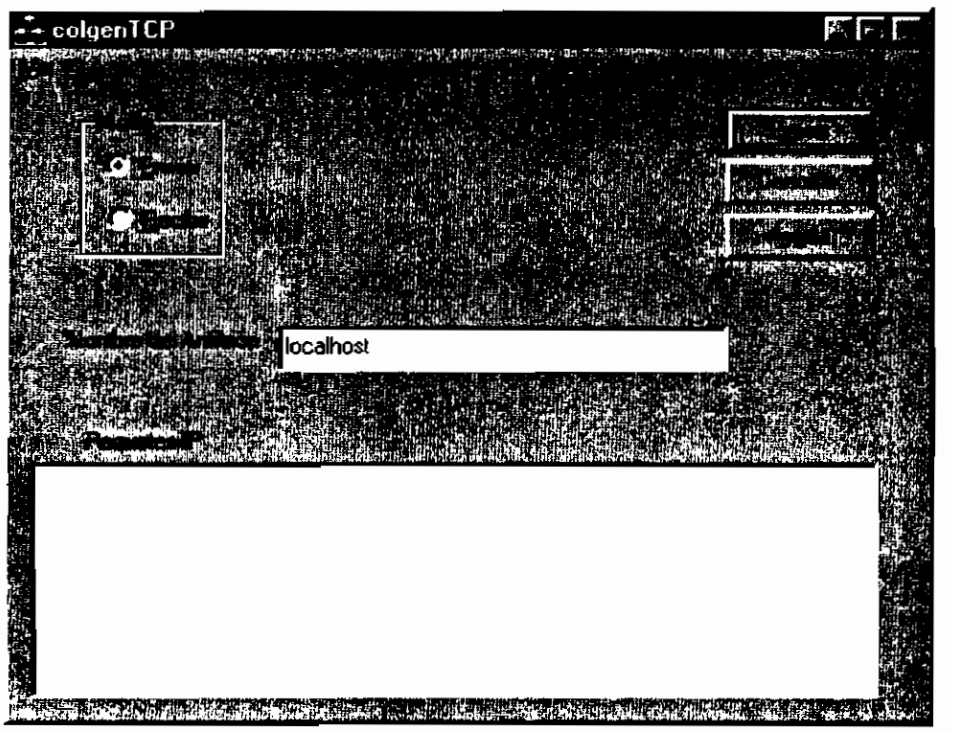


Figura 2.6 Interfaz principal de la aplicación

Para agregar funcionalidad y código a los controles y menú del diálogo se utiliza el Class Wizard de Visual C++, como se explicó ya en el capítulo anterior.

En general se observa una interfaz que permite al programa trabajar en uno de dos modos ya sea como generador, en cuyo caso su función es *enviar* un paquete la red; o sea como colector, en cuyo caso la función del programa es capturar o *recibir* paquetes que circulen por la red.

A continuación se describe cada modo de operación del programa.

2.6 GENERADOR DE PAQUETES (MODO ENVIAR).

A continuación se presenta el diagrama de flujo para el programa funcionando en modo de envío o generador de paquetes.

El programa cuando inicia, de manera predeterminada está en modo de envío o generador de paquete. Como se muestra en el diagrama de flujo, en primer lugar se resuelve la dirección del host o anfitrión destino, para lo cual se utiliza la función *GetIPAddress que toma como parámetro o argumento el nombre del host y devuelve su dirección IP si lo encuentra o en caso contrario da un mensaje de error "anfitrión no alcanzable".*

Figura 2.7 Mensaje que devuelve *GetIPAddress*

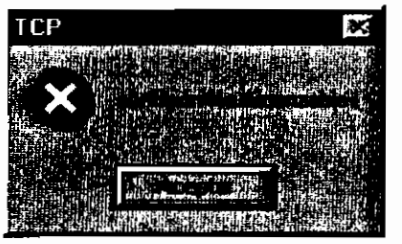


Figura 2.8 Mensaje que devuelve la aplicación

La función *GetIPAddress* utiliza la función **gethostbyname**(lpstrAddr) de visual C++, y ha sido explicada en detalle en varias tesis relacionadas con interfaz o API Winsock.

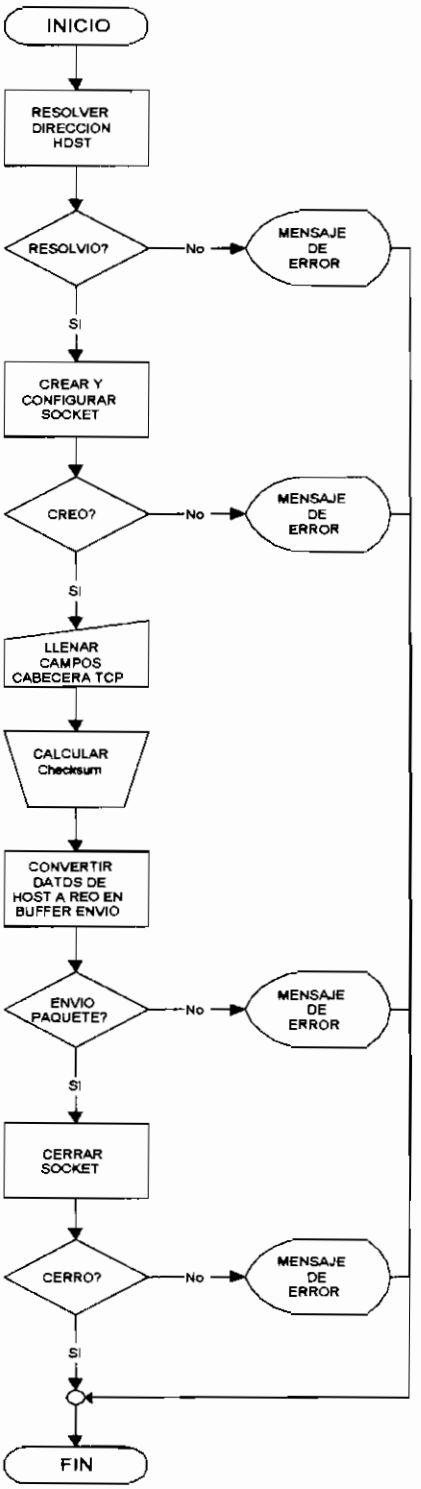


Figura 2.9 Diagrama de flujo para el generador de paquetes.

A continuación se utiliza la función socket para crear un socket básico, como ya se explicó esta función toman tres parámetros: la familia direcciones de Internet, el tipo de socket, y el protocolo usarse; como se indica en la línea siguiente:

TcpSocket = socket (PF_INET, SOCK_RAW,IPPROTO_IP)

En caso de no poder crear el socket se despliega el siguiente mensaje de error "no puedo crear un socket básico".
Se crea e inicializa el buffer de envío, luego se especifica el buffer de envío como una estructura. Se llama a la clase **CTCPHeader**, que es un diálogo que muestra la estructura de un paquete TCP, llena con valores predeterminados en sus campos o que permiten ingresar nuevos valores en dichos campos. Luego se debe presionar el botón **cheksum**, que calcula en la suma de comprobación de acuerdo al algoritmo ya explicado en el marco teórico. Se utiliza las funciones htons y htonl, para convertir el ordenamiento de bytes de computadora a red*.

Por medio de la función *sendto* se realiza el envío del paquete datos, esta función se explicó en detalle en el marco teórico en la parte correspondiente a sockets básicos.
Si la función devuelve un mensaje de error, se despliega el mensaje de error " no se pudo enviar el paquete".

Finalmente se utiliza la función *closesocket* para cerrar el socket básico, y de no poder hacerlo se despliega un mensaje de error.

Una vez enviado el paquete, el menú **Ver** de la ventana de diálogo principal permite visualizar el paquete enviado en el cuadro de diálogo con estructura correspondiente a un paquete TCP.

---

* Ver BIG ENDIAN y LITTLE ENDIAN pag. 4

La clase **CTCPHeader** como ya se ha dicho permite al usuario ingresar los valores en los campos correspondientes de la estructura de un paquete TCP y además calcular la suma de comprobación (ChkSum). En esta clase se inicializan las variables asociadas con los diferentes campos así como se realizan las conversiones adecuadas de ordenamiento de bytes para enviar o desplegar adecuadamente los datos. La interfaz es como la gráfica que se muestra a continuación.

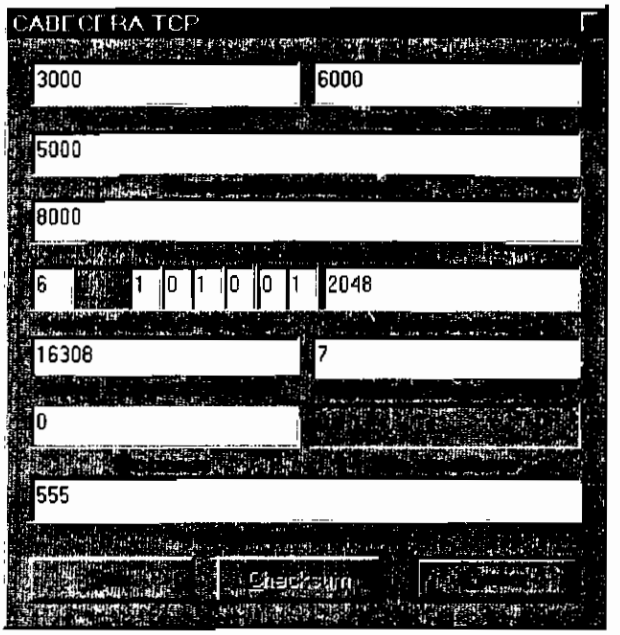


Figura 2.10 Ventana para ingreso de Valores/Cálculo CheckSum.

2.7 COLECTOR DE PAQUETES (MODO RECIBIR)

Para el siguiente módulo se utiliza las librerías de la **API PACKET.DLL**
Se indica al programa que trabaje en modo de recepción o como colector de paquetes dando un click con él ratón en el radio button con etiqueta **Recibir,** este evento cambia también el rótulo del botón **Enviar** a **Recibir**, a continuación se presenta el diagrama de flujo del programa trabajando como un colector de paquetes.

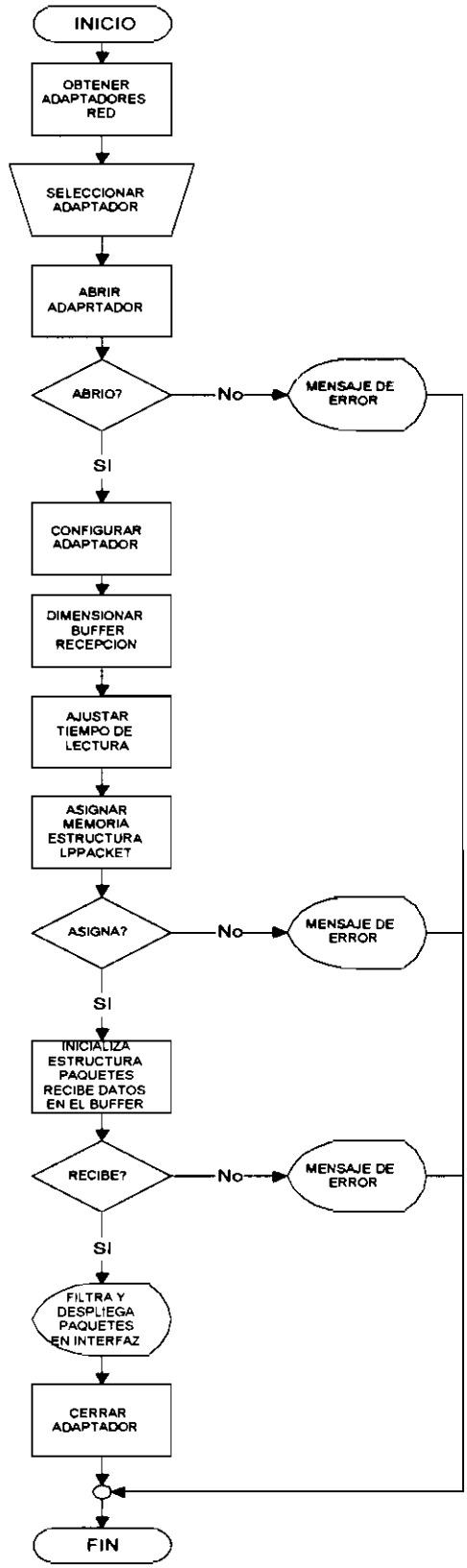


Figura 2.11 Diagrama de flujo para el generador de paquetes.

Primeramente se inicializan algunas variables y punteros, así como también se encera el buffer de recepción y los buffer AdapterList; además se define e inicializa el puntero **lpAdapter** a la estructura **ADAPTER,** ya explicada en el capítulo anterior.

Mediante la función:

PacketGetAdapterNames(AdapterNamea,&AdapterLength**)**

de la API packet.dll se obtienen los adaptadores de red instalados en un computadora para luego copiarlos en los buffer AdapterList y desplegarlos en un ListBox de una nueva caja de diálogo de la **CAdapterList**; en donde el usuario escoge el adaptador a utilizarse en la captura de paquetes.

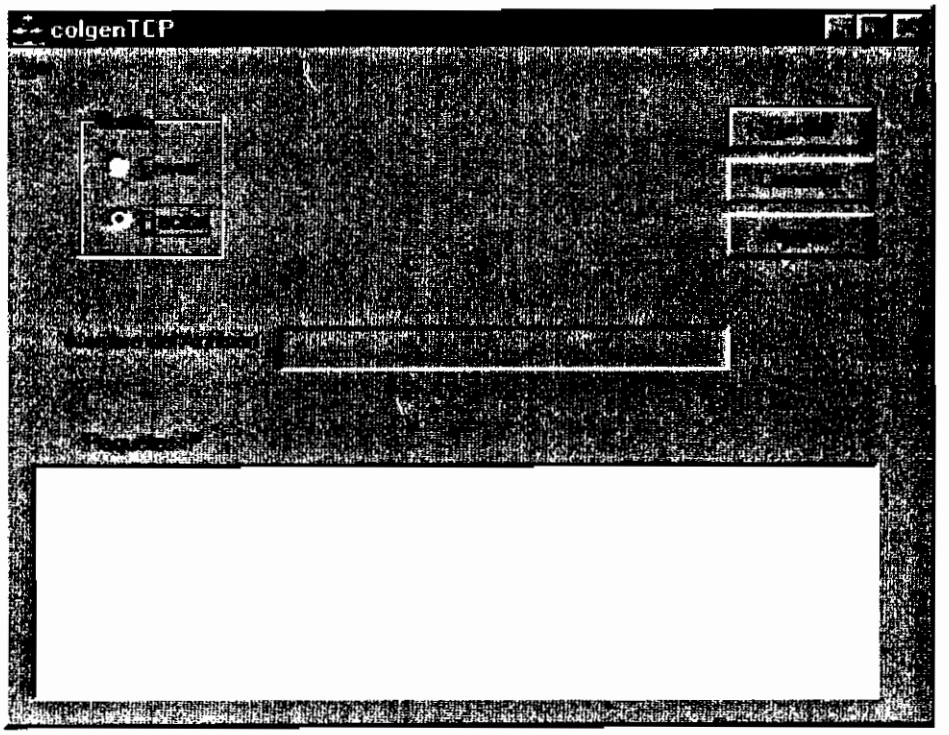


Figura 2.12 Programa como Colector Paquetes

Utilizando la función: **PacketOpenAdapter(**AdapterList[Open]**),** se abre el adaptador seleccionado para la captura de paquetes y de no ser posible se despliega mensajes de error: "**N**o es posible abrir el controlador del adaptador de red"

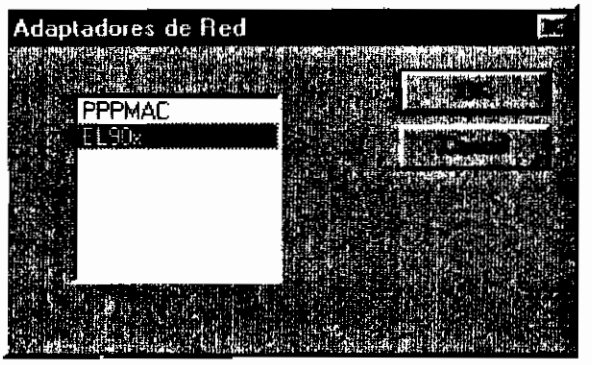


Figura 2.13 Dialogo para seleccionar adaptador de red

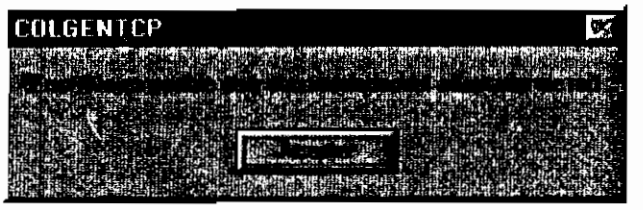


Figura 2.14 Message Box indica error en apertura de adaptador

Si la operación de apertura del adaptador se ha realizado con éxito, se procede a configurar el adaptador de red en modo promiscuo con la siguiente función:

PacketSetHwFilter(lpAdapter,NDIS_PACKET_TYPE_PROMISCUOUS)
Se configura el tamaño del buffer del controlador del adaptador de red a 512K, que es un tamaño razonable ni muy grande ni muy pequeña; para lo cual se utiliza la siguiente función: **PacketSetBuff (** lpAdapter, 512000 **)**
Además se configura el tiempo de lectura del adaptador a un segundo con la función siguiente: **PacketSetReadTimeout (** lpAdapter,1000 **)** que es más que suficiente para capturar datos en una red de 10 MBPS.
A continuación asignamos memoria a la estructura **PACKET,** apuntada por el puntero **lpPacket**; esto lo hacemos utilizando la función:

lpPacket = **PacketAllocatePacket( );** en caso de no poder asignar memoria a la estructura desplegamos el siguiente mensaje de error: "Falla en asignar estructura PACKET"

Sí ha sido exitosa la asignación de memoria a la estructura procedemos a inicializar la misma con la siguiente función:

PacketInitPacket(lpPacket, (char*)buffer, 256000 **)**

Finalmente se realiza la captura de paquetes con la siguiente instrucción:

PacketReceivePacket(lpAdapter, lpPacket, TRUE**)**

Y si esto no se da despliega el siguiente mensaje de error: "Error: Falla en la recepción de paquetes"

Una vez que se tienen los paquetes capturados en el buffer se procede a filtrar los que son de interés en este caso TCP. El siguiente de diagrama de flujo facilita la comprensión del proceso de filtrado.

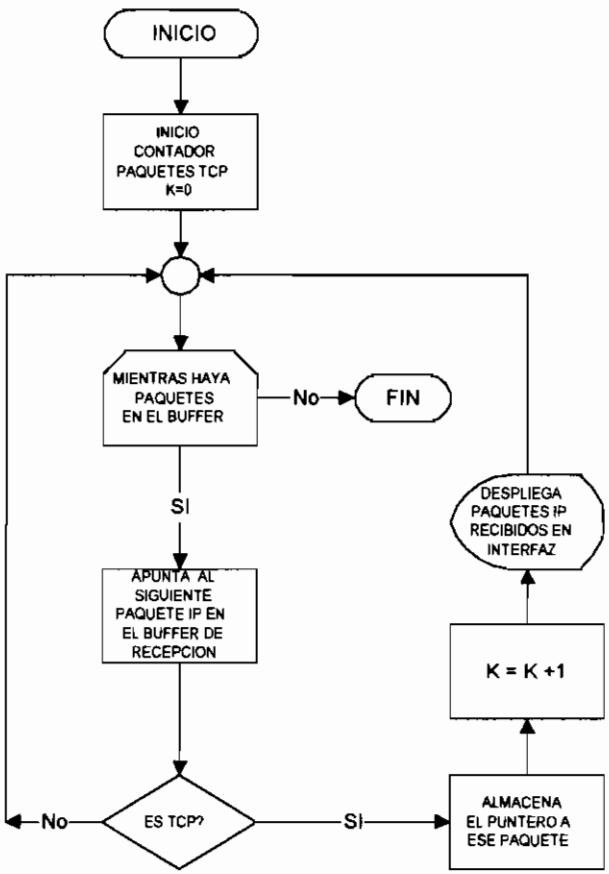


Figura 2.15 Filtro de paquetes TCP

La función que implementa este filtro es:

void CColgenTCPDlg::**PrintPackets**(LPPACKET lpPacket)

Se inicializa y encera el contador paquetes TCP. En primer lugar se determina el puntero al buffer donde se guardan los paquetes capturados; en el caso de una red ethernet los paquetes van a ser tramas ethernet, entonces sabemos que una trama ethernet tiene por cabecera 14 bytes y así determinamos el puntero al paquete o datagramas IP versión 4 que llamamos **pIpHeader,** luego analizamos el encabezado IP en el campo protocolo, y si éste tiene valor 6 entonces éste es un paquete TCP y guardamos este puntero en un vector pPacket[k] = pIpHeader; para poder recuperarlos posteriormente, y mostrarlo en un diálogo que presenté la estructura del paquete TCP.

Este proceso lo realizamos para todos los paquetes capturados en el buffer. Adicionalmente en este proceso de filtrado aprovechamos también para desplegar en el List Box de diálogo de interfaz principal los paquetes que cumplen con el filtro, para esto se utiliza la función **wsprintf** y el método m_ctlListData.**AddString**(listsecenv)

Mientras que para determinar el puntero paquete TCP se utiliza la siguiente expresión:

pTcpHeader = (TcpHeader *)((LPBYTE)pIpHeader+((pIpHeader->ip_verlen & 0x0f)*4))

Que no es otra cosa sino apuntar al inicio de los datos del datagrama IP.

Al final es necesario cerrar el adaptador esto se lo hace con la siguiente función: **PacketCloseAdapter(** lpAdapter **)**

Ahora que se han capturado y filtrado los paquetes TCP, es importante que podamos visualizar la estructura de varios de ellos; se ha considerado presentar los seis primeros paquetes capturados.

A continuación se muestra el diagrama de flujo de las rutinas utilizadas para visualizar los datos. (Figura 2.16 Visualizar Paquetes TCP)

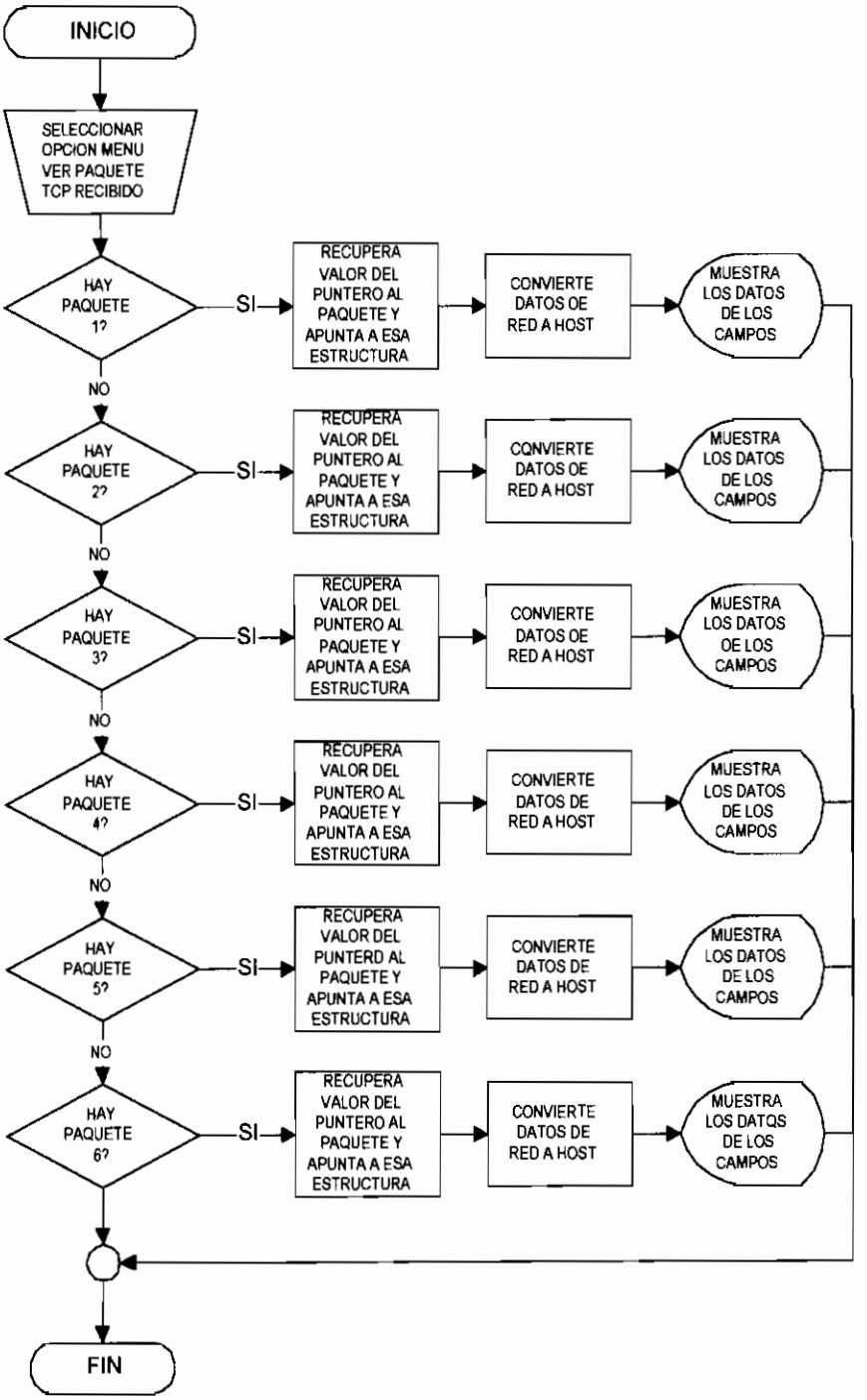


Figura 2.16 Visualizar Paquetes TCP

En él menú Ver del diálogo principal, en la opción **Cabecera de Paquete TCP Recibido,** se selecciona el paquete 1,2,3,4,5 o 6 para visualizar su estructura.

En el programa principal colgenTCPDlg.cpp, mediante condicional **if** se determina que número (k) de paquete se quiere ver, entonces se recupera el puntero a dicho paquete con la instrucción: **pIpHeader = pPacket[k]** luego se llama a la clase **CTCPHeader**, para el despliegue de los datos.
La clase **CTCPHeader** es un diálogo donde se presenta la estructura del paquete TCP como se muestra en la figura 2.17

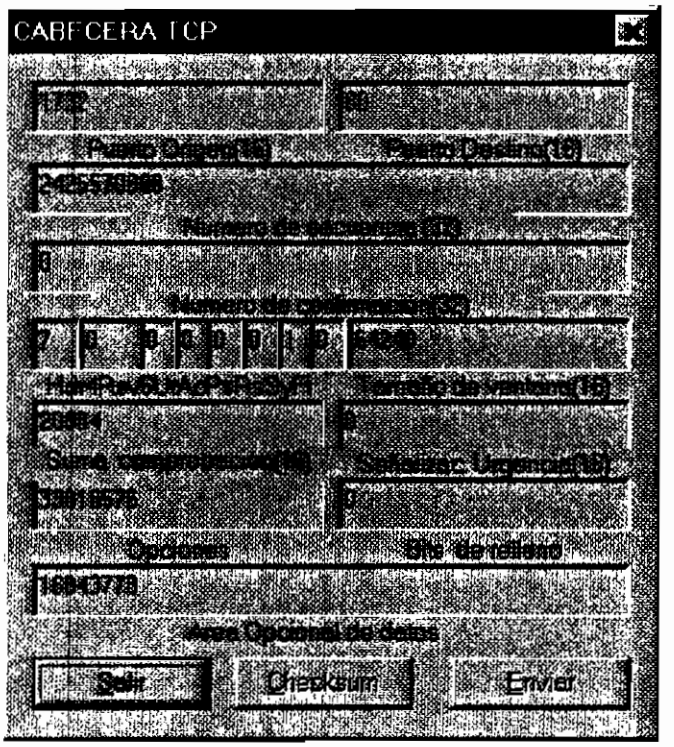


Figura 2.17 Dialogo de CTCPHeader.

2.8 VER CABECERA IP.

Adicionalmente en el menú Ver se ha incorporado a la opción, ver cabecera del paquete IP, ésta función llama a la clase **CIPHeader,** debe ser utilizada inmediatamente luego de un paquete TCP para ver la estructura del paquete IP en que esta encapsulado.

La clase **CIPHeader** define la estructura del datagrama IP y despliega una ventana de diálogo como la siguiente.

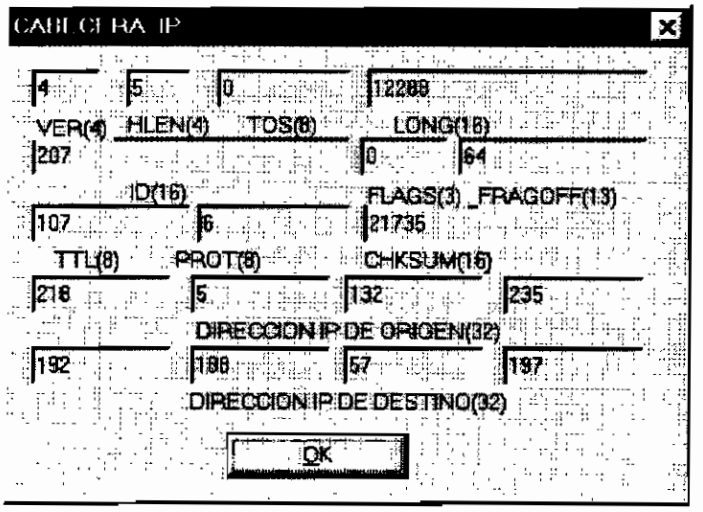


Figura 2.18 Dialogo muestra cabecera IP.

En el siguiente capítulo se da una explicación más detallada y gráfica sobre el uso del programa.

En lo que respecta al detalle de las instrucciones del programa, el código fuente de las diversas clases utilizadas en esta aplicación se muestran en el ANEXO C.

CAPITULO 3.

PRUEBAS Y RESULTADOS

A continuación se muestra, una secuencia de gráficos que son pantallas capturadas del uso del programa colgenTCP.

La figura 3.1 muestra la interfaz principal del programa.

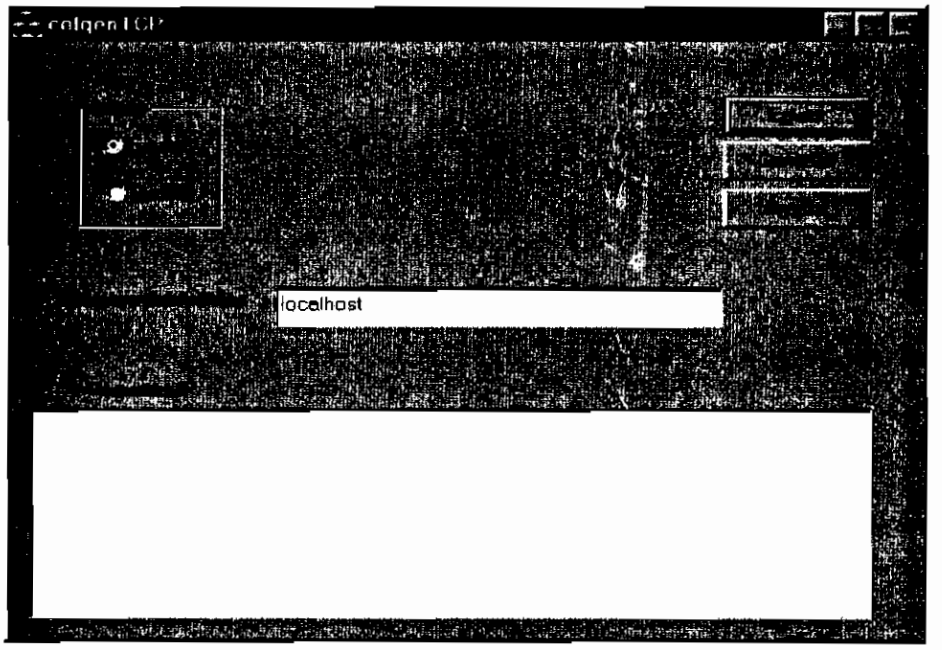


Figura 3.1 Interfaz principal del programa.

3.1 PAQUETE ENVIADO

Se selecciona el modo de envío, que además es el que viene por defecto seleccionado. Se da click en el botón enviar y se despliega la siguiente pantalla para el ingreso de datos. Esta ventana aparece con una serie de datos predeterminados, y calculada la suma de comprobación para esos datos. Si se desea puede dar click en el botón Enviar(OK) de la ventana y enviar un paquete con estos datos, cada vez que se da un click se envia un paquete al anfitrión determinado en la interfaz principal de ColgenTCP.

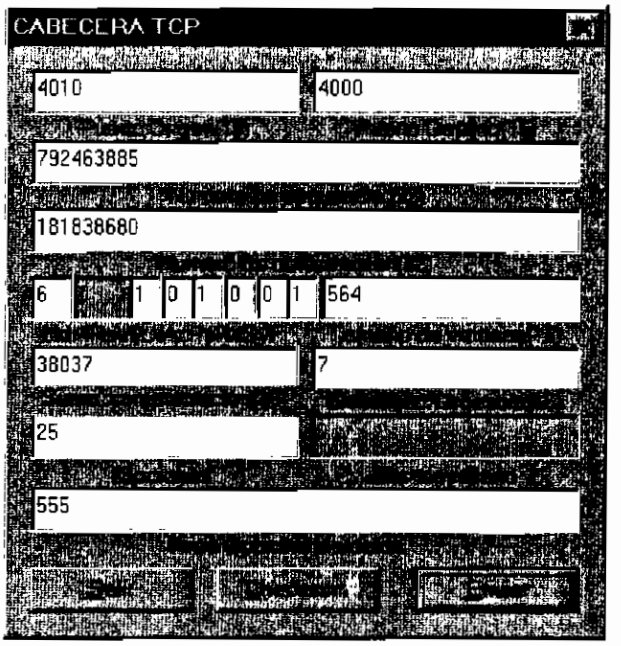


Figura 3.2 Dialogo muestra Datos por defecto.

Si se modifican los datos, por ejemplo el puerto de destino a 7001 y se calcula el Checksum. Se tiene la siguiente ventana.

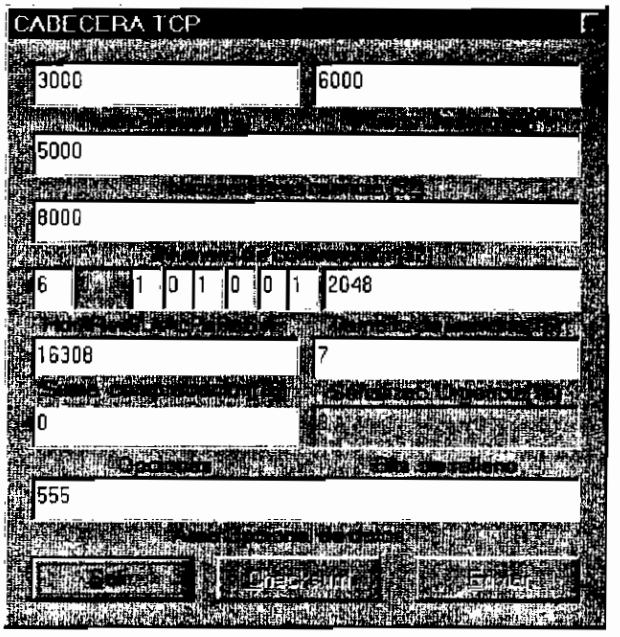


Figura 3.3 Modifico Datos y Calulo Checksum.

Finalmente se presiona el botón Enviar(OK) para enviar el paquete.

3.2 PAQUETES RECIBIDOS.

Se selecciona ahora la opción de recibir.

Luego de seleccionar el adaptador de red y capturar los paquetes se tiene la interfaz mostrada por la figura 3.4.

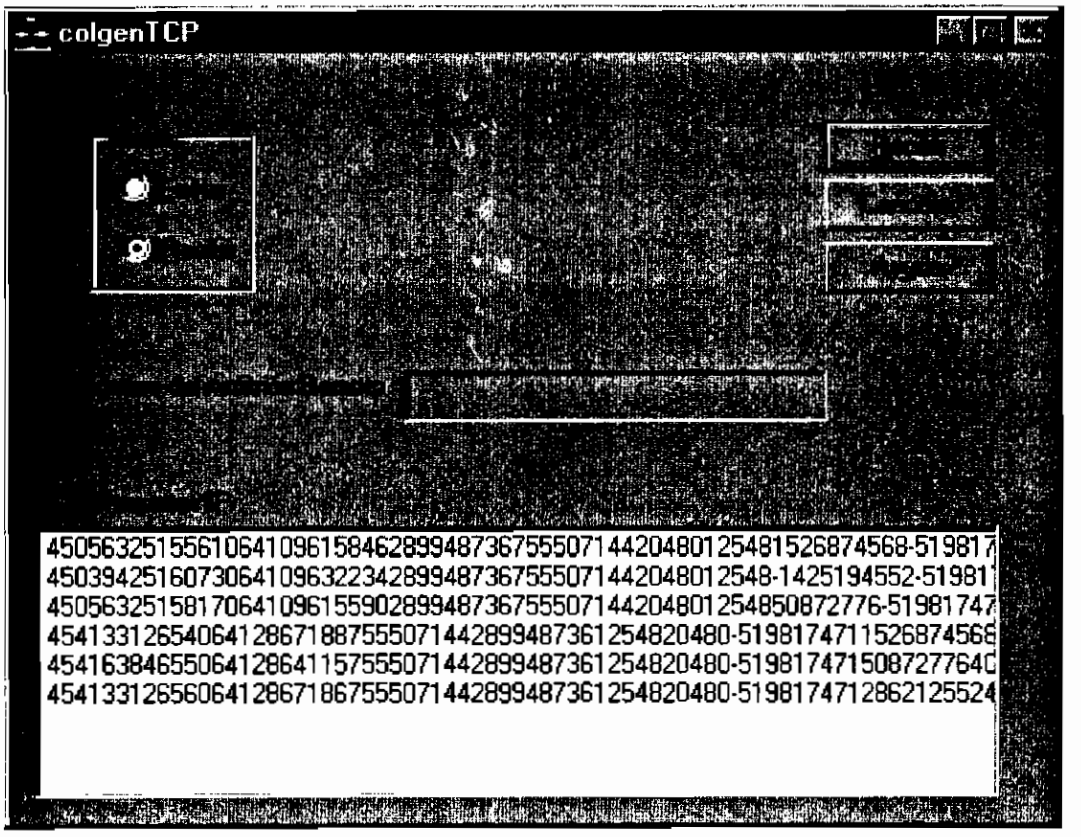


Figura 3.4 Interfaz Principal muestra paquetes capturados.

Como se puede observar en la figura anterior se listan seis paquetes TCP que se han capturado en el buffer en esa ocasión.

A continuación se muestra el primer paquete TCP recibido, y el paquete IP que lo contiene:

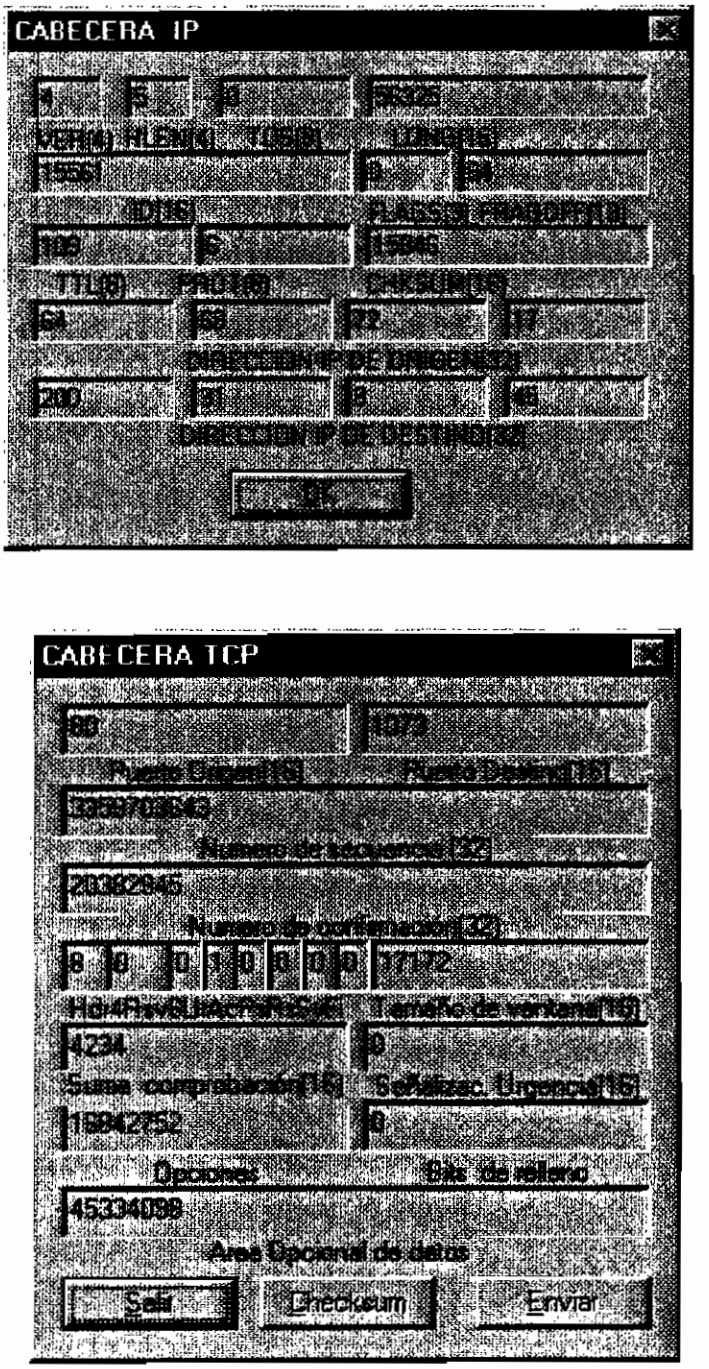
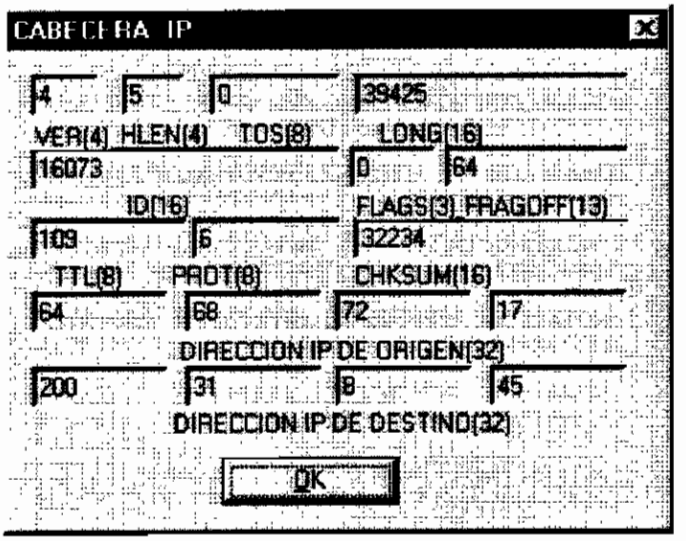


Figura 3.5 Primer Paquete Capturado

Veamos el segundo paquete TCP recibido, y el paquete IP que lo contiene:



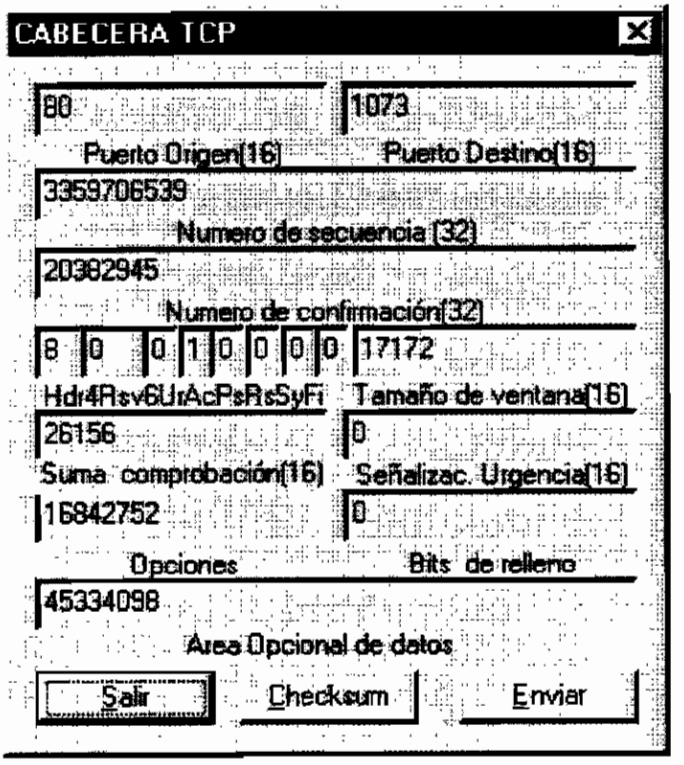
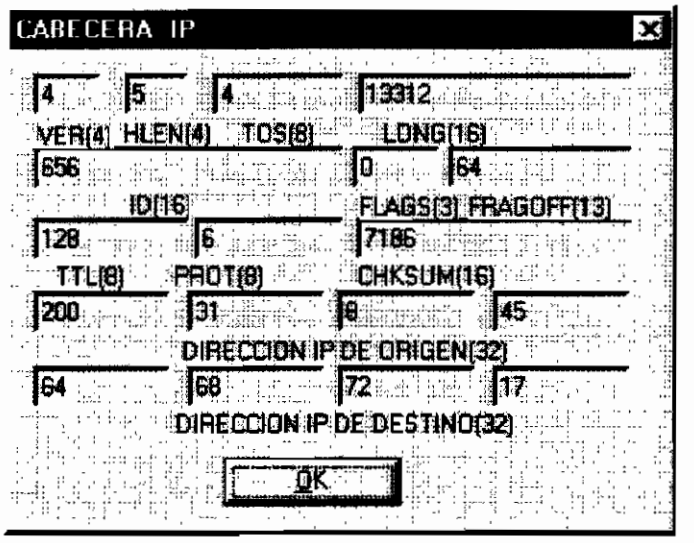


Figura 3.6 Segundo Paquete Capturado

Veamos el sexto paquete TCP recibido, y el paquete IP que lo contiene:



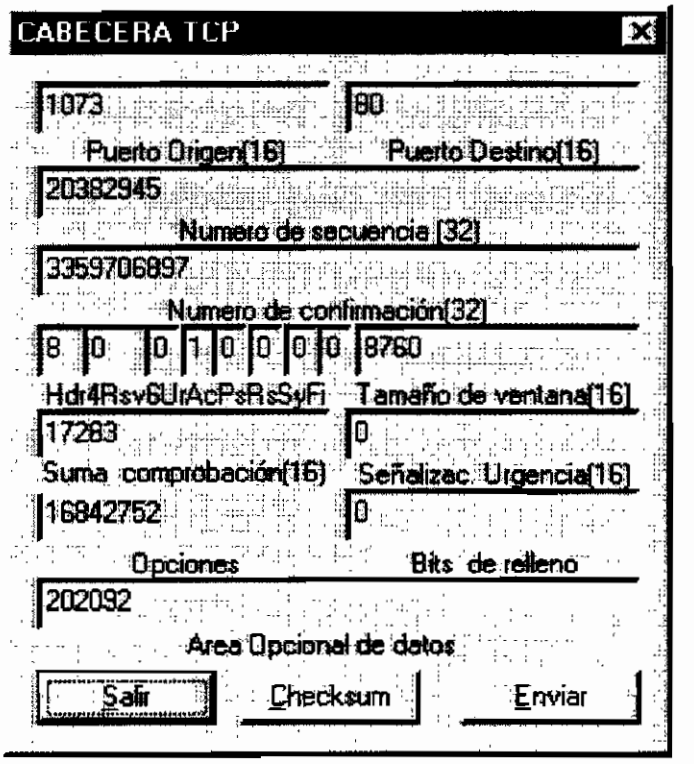


Figura 3.7 Sexto Paquete Capturado.

3.3 COMENTARIOS

Como se puede ver en las figuras 3.5, 3.6, 3.7 la cabecera IP de todos estos paquetes en el campo correspondiente al tipo protocolo tienen el número 6, que es el valor correspondiente al protocolo TCP que es el tipo de paquetes que se permiten pasar por el filtro de esta aplicación.
Otros campos de importancia para analizar qué función tiene el paquete TCP además de la longitud cabecera, son las banderas que en el caso de estos paquetes indican que son de confirmación.

Otra situación de particular interés se da cuando en una máquina se genera el paquete y en otra se lo captura. En cuyo caso al analizar los campos de direcciones en las cabeceras IP, efectivamente corresponden a las direcciones IP de las máquinas involucradas en esta prueba.

La figura 3.3 nos muestra el paquete enviado y la figura 3.8 este paquete recibido en otra computadora. Notar que esta ventana no permite modificar los valores de sus campos.

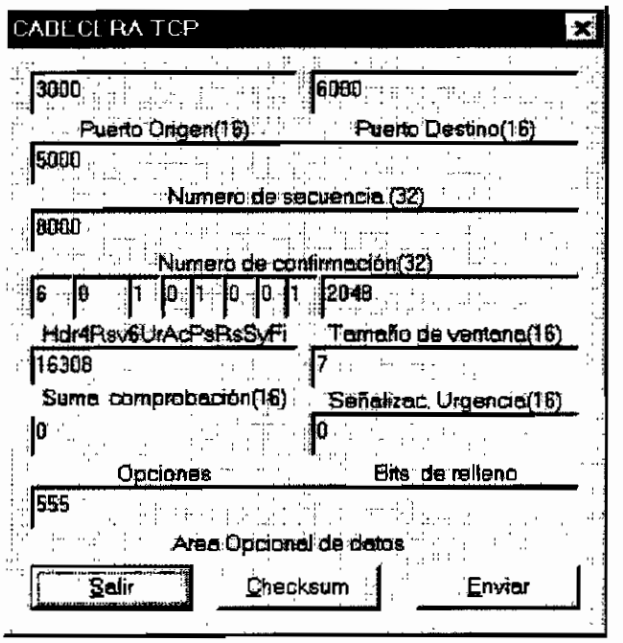


Figura 3.8 Paquete Enviado-Recibido

En las figuras 3.9 y 3.10 se muestran la cabecera IP del paquete recibido, así como la interfaz principal, donde se visualiza que se enviaron y recibieron varios paquetes.

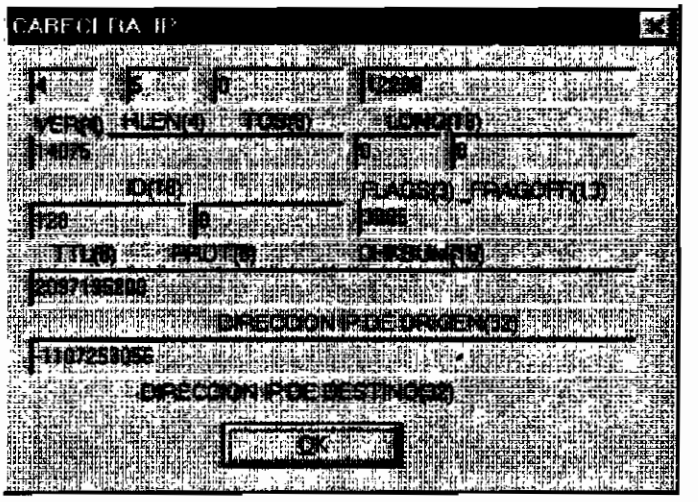


Figura 3.9 Cabecera paquete IP recibido

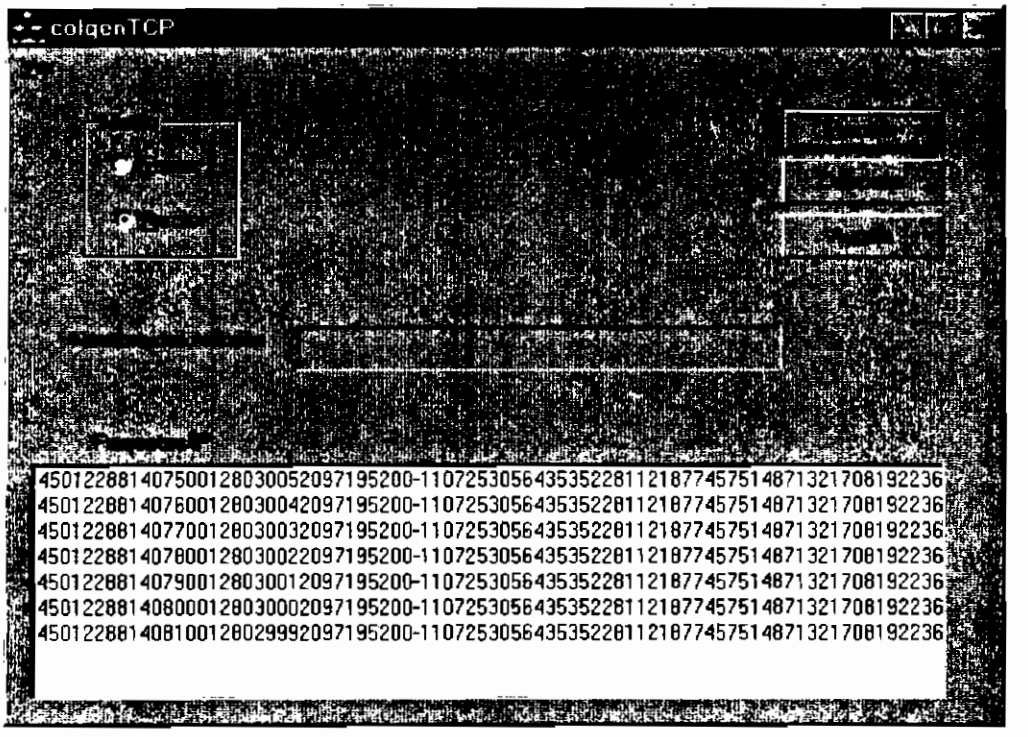


Figura 3.10 Interfaz de ColgenTCP muestra paquete enviado-recibido.

Finalmente como método de comprobación se utiliza el software de monitoreo de redes del Politécnico de Torino "ANALYZER", en la figura 3.11 se muestra la cabecera IP del paquete, mientras que la figura 3.12 se muestra la cabecera del paquete TCP.

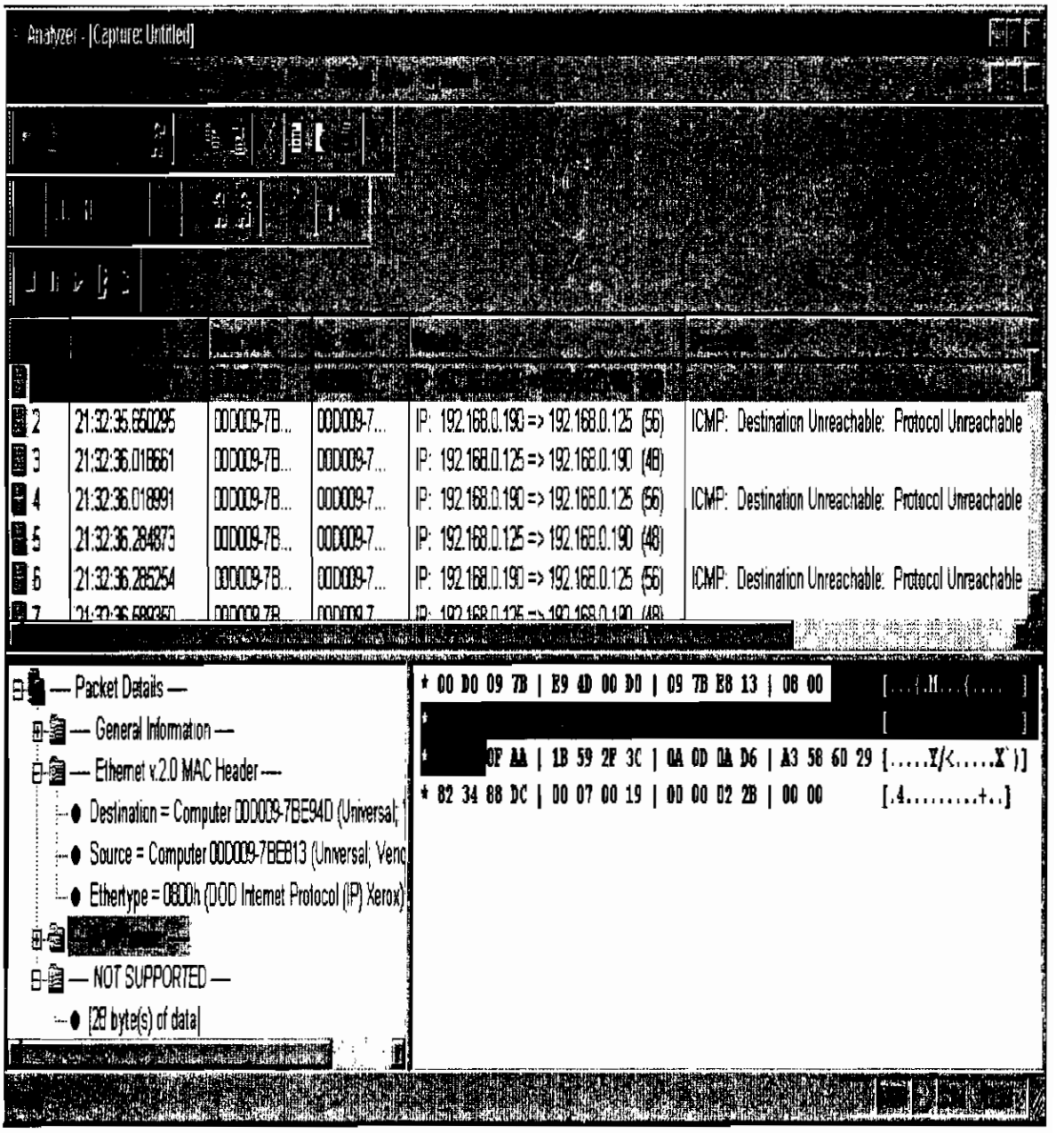


Figura 3.11 Analyzer muestra cabecera paquete IP capturado.

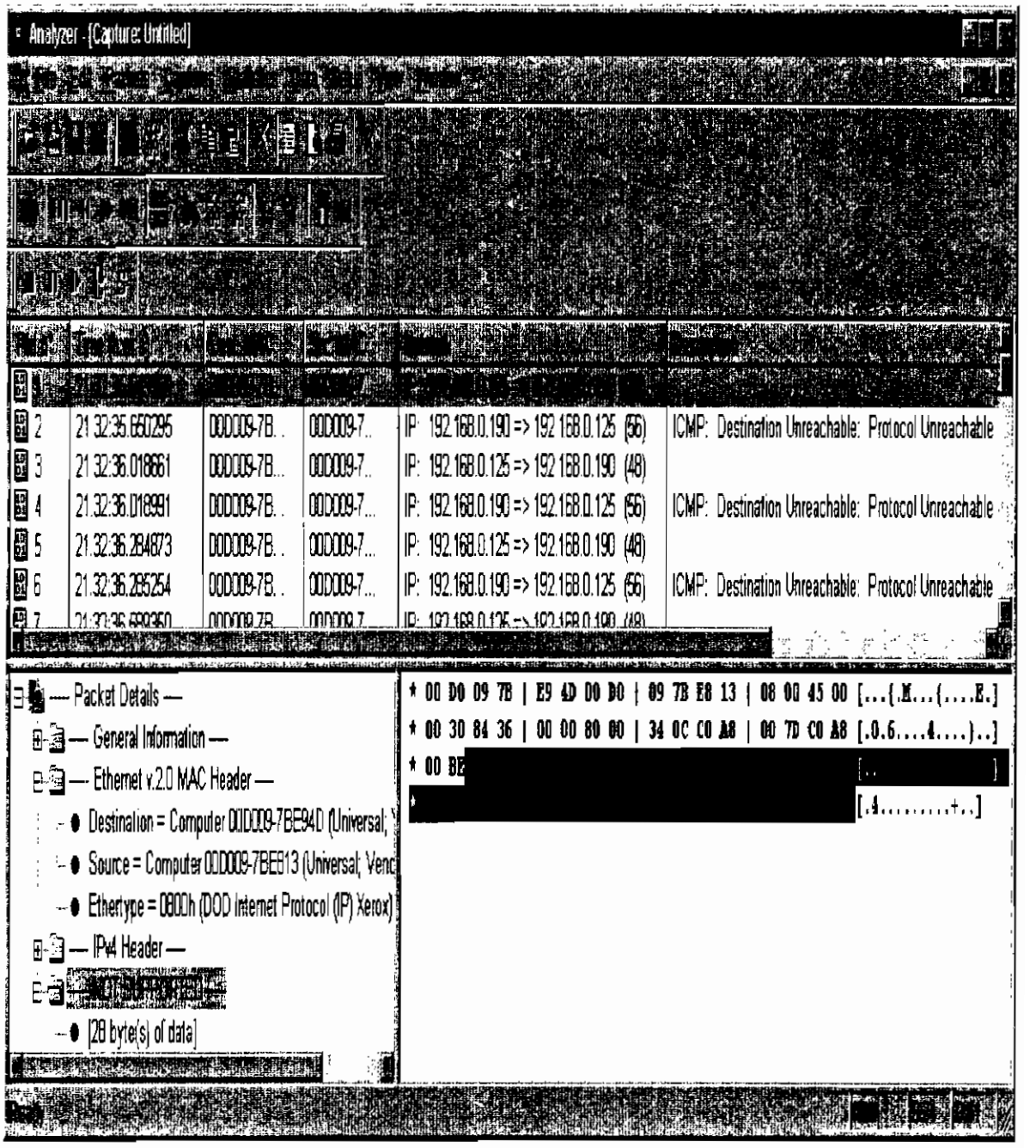


Figura 3.12 Analyzer muestra contenido de paquete IP capturado.

Para prueba por ejemplo el valor hexadecimal **0F AA** cuyo equivalente decimal es el numero **4010,** que efectivamente corresponde al valor del campo puerto origen; y el numero **1B 59** con equivalente decimal **7001,** que también corresponde al valor del campo puerto destino.

CAPITULO 4

CONCLUSIONES Y RECOMENDACIONES.

- Un socket básico permite que los programas salten la capa de transporte TCP/IP y tengan acceso a protocolos de nivel inferior, como IP e ICMP. Por lo tanto, se deben realizar funciones de la capa de transporte, como el encapsulamiento de información, para la capa IP.

- El programa usa un socket básico para saltar la capa de transporte y ésta ya no está disponible para asociar la información entrante con sockets o puertos de protocolo específicos.

- El socket puede ser orientado a conexión o no orientado a conexión, esto se se define en el segundo parámetro de la función socket los posibles valores son representados por las constantes simbólicas: **SOCK_ STREAM** para flujo de bytes, **SOCK_DGRAM** para datagramas; la interfaz de sockets también define un tercer tipo llamado *socket básico* **SOCK_RAW** que permite a un programa utilizar los mismos protocolos de nivel inferior que la red utiliza comúnmente, como son IP e ICMP.

- En el campo protocolo del encabezado IP añadido, la API Winsock almacena el valor del protocolo que el programa especifica el momento de crear el socket. El tercer parámetro le permite a la función *socket* especificar que protocolo usará en el socket, para especificar el protocolo TCP debe usar la constante simbólica **IPPROTO_TCP**.

  Con un socket básico, los programas usan protocolos sin conexión que entregan los datos mediante datagramas.

- Por todo lo mencionado anteriormente y además se sabe que un protocolo orientado a conexión debe **establecer un enlace** con otra aplicación antes de que exista cualquier comunicación, no puede comunicarse es decir transportar datos hasta que se establezca una conexión. Siendo TCP un protocolo orientado conexión. A pesar de que en el programa compila sin errores de sintaxis la instrucción en la que se crea un socket básico especificando su protocolo como TCP y crea dicho socket, sin embargo no hemos establecido una conexión virtual, por lo que este socket no tramiten ni recibe nada.

- La implementación de Winsock cuando utiliza sockets básicos añade un encabezado IP a cualquier información escrita en el socket, es decir envía un datagrama a la capa de red, esto se puede hacer si definimos el tipo de protocolo a usar como IP, estableciendo el tercer parámetro de la función socket como **IPPROTO_IP.**

- Sabiendo que la información que se transmite en la red no es sino un flujo de unos y ceros. Se ha procedido a encapsular un paquete con estructura **TCP** en un datagrama IP, la sutil diferencia que no permite que el programa de monitoreo de red "ANALYZER" lo reconozca como un paquete TCP, es el hecho de que en el campo protocolo de la cabecera IP tenemos el valor cero (0) en lugar del valor 6 que corresponde a TCP, esto debido a a las razones arriba ya explicadas.

- En un grupo de discusión de Internet con respecto a los socket básicos con TCP, se realiza el siguiente comentario: "quieren jugar a ser Dios, reinventando el protocolo **TCP**". Lo cual nos confirma que los socket básicos con TCP no son una buena pareja.

  La API Winsock no es una interfaz que nos permita monitorear y generar todos los tipos de paquetes que circulan en una red.

- En la búsqueda de una solución para el monitoreo de redes, se encontró al "ANALYZER", cuyo desarrollo se fundamenta en la API **WinPcap*** y en la API **PACKET.DLL** desarrollado en el Politécnico de Torino (Italia).

- WinPcap es una arquitectura para captura de paquetes para análisis de redes TCP\IP en plataformas Win32. Esta incluye un filtro de paquetes a nivel kernel(módulo central del sistema), una biblioteca o librería de enlace dinámico de bajo nivel (**packet.dll**), una librería de alto nivel independiente del sistema (Wpcap.dll basada en libpcap).

- En este proyecto de titulación se utilizan ciertas funciones de la API **PACKET.DLL** para la captura de los paquetes en un buffer, mientras que el filtrado de los paquetes se hace al nivel de aplicación, existiendo una manera más eficiente que es el filtrado en la misma tarjeta o adaptador de red. Si ser este el objetivo de la tesis, sin embargo se hace mención de este particular.

- Aunque aquí se ha utilizado la librería **PACKET.DLL** solamente para la captura de paquetes, cabe indicar que también se la puede utilizar, para generar paquetes o tramas ethernet. Posiblemente de esta manera se puede resolver la sutil diferencia arriba mencionada, pero el costo es mayor, pues el programa debería encapsular el datagrama IP que contiene nuestro paquete TCP en una trama ethernet.

- Finalmente cabe señalar que con estas nuevas interfaces de aplicación la **API Packet Driver** o **Packet.dll** (como el conjunto de funciones de bajo nivel), y **wpcap.dll** o **libpcap** (como el conjunto de funciones de alto nivel) se podrán desarrollar nuevas y mejores herramientas para el monitoreo y administración de redes.

---

* basada en LIBPCAP desarrollada en la universidad de Berkeley

BIBLIOGRAFIA.

1. **JAMSA Kris**, PROGRAMACION EN INTERNET, Mc Graw Hill 1996
2. **TANENBAUM Andrew**, REDES DE COMPUTADORAS, Mc Graw Hill 1998, 3ra edición.
3. **FEIT Sidnie**, TCP/IP ARQUITECTURA PROTOCOLOS E IMPLEMENTACION, Mc Graw Hill 1998.
4. **PAPAS Chris, MURRAY William**, MANUAL DE REFERENCIA VISUAL C++ 6.0, Mc Graw Hill 1999
5. **CHAPMAN Davis**, APRENDIENDO VISUAL C++ 6 EN 21 DIAS, Prentice Hall 1999.
6. **FIE**,APUNTES DE TELEMATICA, EPN-FIE 1998.
7. **RFC** (Request For Comments) 793, ACERCA DE TCP
8. **RFC 1071,** COMPUTING THE INTERNET CHECKSUM, 01-SEP-1988.
9. **LORIS DEGIOANNI**, DEVELOPMENT OF AN ARCHITECTURE FOR PACKET CAPTURE AND NETWORK TRAFFIC ANALYSIS, Politecnico di Torino Marzo 2000.
10. **http://netgroup-serv.polito.it/winpcap**

ANEXO A
ENTORNO VISUAL C++

Una vez diseñado el programa. Es tiempo para crear el proyecto en Visual C++ con sus archivos iniciales y propiedades del proyecto. Después de eso, se puede empezar agregando características a la interfaz de usuario, agregando código para llevar a cabo la funcionalidad del programa, y depurarlo conforme se avanza.

En esta sección se revisará los elementos del lenguaje de programación que nos ayudará en el desarrollo de la aplicación, empezando por el entorno de desarrollo de Visual C++ 6.0 al que se adjuntarán ciertas herramientas y sentencias de lenguaje C++.

A.1 FUNDAMENTOS DE PROGRAMACIÓN EN VISUAL C++ 6.0

Visual C++ 6.0 es una evolución como lenguaje de programación C, donde el ambiente de visual C++6.0 tiene una interfaz de trabajo amigable al usuario, que se caracteriza porque, cada vez que se ejecuta un programa, este entorno genera al mismo tiempo los códigos objeto y los ejecutables. El ambiente de programación está dividido en 4 secciones o paneles:

A.1.1 EL PANEL WORKSPACE

Iniciado el programa, ubicado en el lado izquierdo de la pantalla se nos presenta el panel workspace, o espacio de trabajo de la aplicación en desarrollo con tres tabulaciones o pestañas distintas, pudiendo a través de este panel navegar sobre estos diversos componentes:

- Class View donde nos permite manipular los componentes de la aplicación al nivel de clase C++
- Resource View, permite ubicar y editar cada uno de los recursos de la aplicación como botones, iconos y menús de la ventana de diálogo; y,

- File View nos muestra todos los archivos de la aplicación en desarrollo.

A.1.2 EL PANEL OUTPUT

El panel Output, que no está visible mientras no se esté compilando la aplicación. Nos muestra instrucciones del progreso del compilador, advertencias, mensajes de error, y es el lugar donde el depurador de visual C++ despliega todas las variables con sus valores actuales, de acuerdo a su avance a través del código en la compilación, aparece en la parte inferior de la pantalla. Si se lo cierra, éste se abrirá automáticamente mientras visual C++ requiera desplegar algún mensaje.

A.1.3 EL ÁREA DE EDICIÓN

Es el lugar donde se desarrolla toda la edición sea diseño de cuadros de diálogo, edición misma cuando se utilice Visual C++ o códigos fuente en C++;por lo tanto ocupa la mayor parte de la pantalla. También en este cuadro se desplegará el editor de iconos que se lo utiliza cuando requiera editar iconos personalizados para sus aplicaciones.

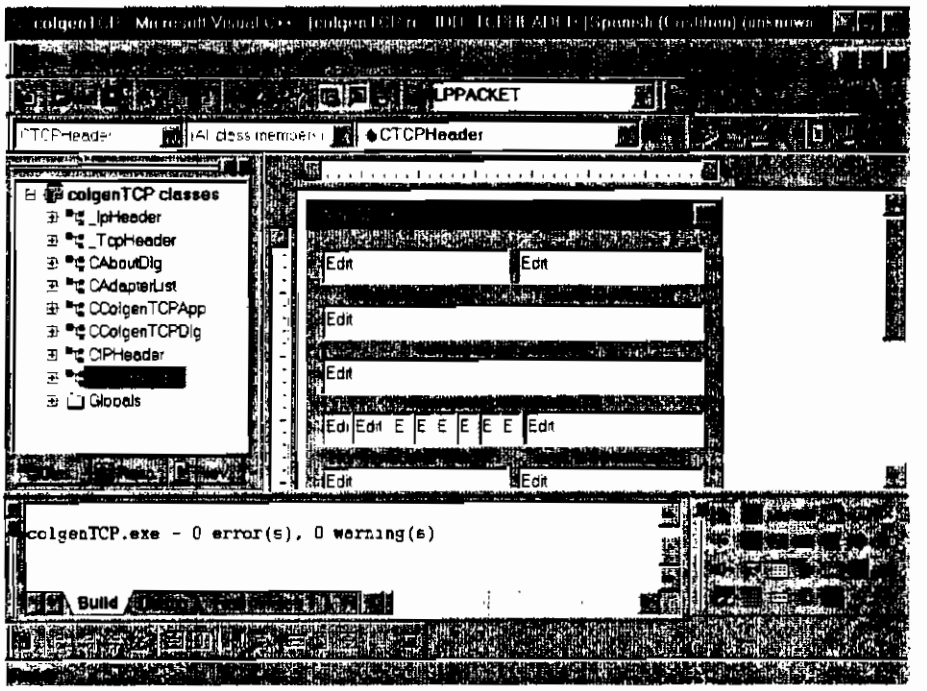


Figura.A.1. Entorno de Trabajo Visual C++

A.1.4 BARRA DE MENÚS

Cuando se ejecuta Visual C++ aparecen tres barras de menús, las que se pueden personalizar así como el panel workspace, output o la ventana de edición; al adquirir más destreza en el desarrollo de aplicaciones, sin embargo las que aparecen inicialmente son:

- La barra de herramientas estándar que contiene las herramientas comunes utilizadas en la mayoría de aplicaciones de Windows como son las opciones de Guardar, copiar, pegar muy útiles; así como otras funciones propias de Visual.
- Barra de Wizard, que desplegada muestra una gran cantidad de acciones sin tener que abrir la clase Wizard.
- Minibarra Build que se la utiliza para ejecutar la construcción, ejecución y comandos de depuración.

Se puede organizar de mejor manera el ambiente de trabajo sea en el número de barras de herramientas a mostrar por el developer studio o por la ubicación misma del conjunto de herramientas en el ambiente de trabajo. Para ocultar o desplegar los diversos grupos de herramientas basta con dar un click con el botón derecho del ratón sobre el área de las barras de herramientas, apareciendo un menú contextual que permite activar o desactivar los diferentes grupos.

En cuanto a la ubicación basta con arrastrar cada uno de los grupos haciendo click en la doble barra mostrada en el extremo de cada uno de los grupos.

Visual C++ utiliza como base al lenguaje de programación C o C++, que predomina en el mundo de las minicomputadoras de sistema UNIX y computadoras personales, en los sistemas operativos, intérpretes, compiladores, editores de texto, programas de ensamblado, gestores de bases de datos, etc.

A.2 VENTAJAS

Eficiencia.- Aprovechamiento óptimo de las características de hardware es decir los programas en C tienden a ser mas compactos y se ejecutan con mayor rapidez.

Exportabilidad.- Fácil de adaptar con leves modificaciones a nuevos modelos computacionales.

P*otencia*.- En C están escritos casi todos los compiladores e intérpretes, así como sistemas operativos como UNIX; utilizado también en ingeniería de manejo de pórticos, comunicaciones, etc.

Flexibilidad.- Posee control tanto sobre el lenguaje ensamblador y las ventajas del lenguaje de alto nivel.

En visual C++ se utiliza la *programación orientada a objetos* que es superior *a* la programación estructurada.

La programación orientada a objetos se basa en la idea natural de la existencia llena de un mundo de objetos, donde la resolución de problemas se realiza en términos de objetos, un lenguaje se dice que se basa en objetos si soporta objetos como característica fundamental del mismo.

El elemento principal de la programación orientada a objetos es el objeto, el mismo que se puede definir como un conjunto complejo de datos y programas que poseen una estructura y forman parte de una organización.

Esta definición especifica varias propiedades importantes de los objetos. En primer lugar, un objeto no es un dato simple, sino que contiene en su interior cierto número de componentes bien estructurados. En segundo lugar cada objeto no es un ente aislado sino que forma parte de una organización jerárquica o de otro tipo.

A.3 ESTRUCTURA DE UN OBJETO

Un objeto puede dividirse como una especie de cápsula dividida en tres partes:

1. Relaciones
2. Propiedades
3. Métodos

Cada uno de estos componentes desempeña un papel totalmente independiente:

Las *relaciones* permiten que los objetos se inserten en la organización y están formadas esencialmente por punteros a otros objetos.

Las *propiedades* distinguen un objeto determinado de los restantes que forman parte de la misma organización y tienen valores dependiendo de la propiedad de que se trate. Las propiedades de los objetos pueden ser heredadas a sus descendientes en la organización.

Los *métodos* son las operaciones que pueden realizarse sobre el objeto, que normalmente estarán incorporados en forma de programas (código) que el objeto es capaz de ejecutar y que también pone a disposición de sus descendientes a través de la herencia.

A.4 POLIMORFISMO

Una de las características fundamentales de la programación orientada a objetos es el polimorfismo, es decir la posibilidad de construir varios métodos con el mismo nombre, pero con relación a la clase que pertenece cada uno, con comportamientos diferentes. Esto conlleva a la habilidad de enviar un mismo mensaje a objetos de clases diferentes; por ejemplo, un mensaje "+" a un objeto entero significaría suma, mientras que para un objeto string significaría concatenación ("pegar" strings uno seguido del otro).

A.5 CLASES

El concepto de clase está ligado al de objeto; las clases son para los objetos algo similar de lo que son los tipos para las variables. De hecho la declaración de una clase es en muchos lenguajes una declaración de tipo y los objetos de esta clase se obtienen igual que el resto de variables de sus tipos. En general una clase es un conjunto de atributos mientras que en programación una clase es un conjunto de objetos que comparte estructura y comportamiento común.

A.6 ELEMENTOS DEL LENGUAJE

Los datos en lenguaje C++ pueden representarse mediante constantes o variables

TIPO		TAMAÑO EN BITS	RANGO MINIMO
CARÁCTER	Char	8	-128 a127
	Unsigned char	8	0 a 255
	Signed	8	-128 a 127
ENTERO	Int	16	-32768 a32767
	unsigned int	16	0 a 65535
	signed int	16	-32768 a32767
	short int	16	-32768 a32767
	unsigned short int	16	0 a 65535
	signed short int	16	-32768 a32767
	long int	32	-2147483648 a 2147483647
	signed long int	32	-2147483648 a 2147483647
	unsigned long int	32	0 a 4294967295
PUNTO FLOTANTE	Float	32	±(3.4 e-38 a 3.4 e+38) 6 dígitos de precisión
	Double	64	±(1.7 e-308 a 1.7 e+308) 10 dígitos de precisión
	long double	80	±(3.4 e-4932 a 3.4 e+4932) 10 dígitos de precisión
SIN TIPO	Void	0	sin valor

Tabla 1.3. Tipos de datos en Lenguaje C

A.6.1 CONSTANTES

Son datos preseleccionados en tiempo de compilación del programa, cuyos valores se mantienen sin alterar durante la ejecución del mismo.

Proveen ciertas ventajas como, se puede modificar su valor en un solo sitio o que el nombre de la constante define claramente su valor ya que el compilador es capaz de identificar el tipo de la constante por el aspecto que tiene.

A.6.2 VARIABLES

Es una posición de memoria con nombre, en donde se guarda su valor, el mismo que puede ser modificado durante la ejecución del programa. En C se debe declarar las variables previamente antes de usarlas, porque en la compilación del programa asigne el espacio de memoria para las variables de acuerdo al tipo declarado.

A.6.3 TIPOS DE DATOS

Define el conjunto de valores que puede tener una variable junto con un conjunto de operaciones que se pueden realizar sobre esta variable. Los tipos de datos comunes se enumeran en la tabla 1.3

Todos los demás tipos de datos que se basan en los enumerados y por la forma de almacenamiento en la memoria del computadora.

A.7 OPERADORES

La tabla 1.4 resume los operadores existentes para programación en lenguaje C o C++.

A.8 SENTENCIAS DE CONTROL

Son las piezas con las que se construye un programa clasificándose en:

1. La sentencia de declaración. Sirve para establecer los nombres y el tipo de variables y hace que la computadora reserve posiciones de memoria para cada una de ellas.
2. La sentencia de asignación. Sirve para asignar valores a las variables y está formada por un nombre de variable, seguido de operador de asignación =, de la expresión y de un punto y coma;.
3. La sentencia de función. Es una llamada a función que se encarga de realizar una tarea de acuerdo a como fue diseñada la función.

TIPO	OPERACIÓN	OPERADOR
Asignación	Igualdad	=
	Suma	+=
	Resta	-=
	Multiplicación	*=
	División	/=
	División parte entera	%=
Aritmético	Suma	+
	Reta	-
	Multiplicación	*
	División	/
	Parte entera	%
Relación	Menor que	<
	Mayor que	>
	Igual a	==
	No es igual a	!=
	Menor o igual a	<=
	Mayor o igual a	>=
Lógicos	AND	&&
	OR	||
	NOT	!

Tabla 1.4 Operadores de lenguaje C y C++

4. Las sentencias de control. Son aquellas que deciden que sentencias deben ejecutarse o cuales sentencias deben repetirse.
5. La sentencia nula no hace nada.

La sentencia puede ser compuesta, es decir pueden estar agrupadas y encerradas entre llaves realizando una tarea determinada.

A.9 INICIO DE UN PROYECTO EN VISUAL C++6.0

Usando los WIZARD de programación (AppWizard) para MFC, no sólo se beneficia del código pre-escrito que ellos proporcionan, sino también del código fuente pre-generado que se usa para llamar a MFC que maneja muchas tareas de la programación rutinarias. Se puede, por supuesto, escribir un programa sin usar un WIZARD, pero los WIZARD le ahorran tanto tiempo que perdería de no usarlos.

ClassWizard como ayudante de un programador, ClassWizard hace más fáciles ciertas tareas rutinarias como crear nuevas clases, definir manejadores de mensajes, sobreponer funciones virtuales, y adquirir datos de los controles en una caja de diálogo. ClassWizard también hace fácil agregar propiedades, métodos, y eventos a los objetos para su automatización. ClassWizard trabaja con programas que usan la librería "Microsoft Foundation Class (MFC)".

En el Visual C++ 6.0 para iniciar un proyecto se debe seguir los pasos indicados a continuación.

A.9.1 CREACIÓN DEL ESPACIO DE TRABAJO PARA EL PROYECTO

Es crear directorios, donde se almacenarán el código fuente así como directorios en donde se almacenarán diversos archivos de configuración para la construcción del programa. Este se crea de la siguiente manera:

- Seleccionar File|New como se muestra en la figura A.2.
- En la ficha Projects se selecciona el Asistente MFC App Wizard(exe).
- En el campo Project Name escribimos el nombre que identificará al proyecto.
- Al Aceptar (OK) se creará un directorio donde se almacenará la aplicación y se ejecuta el Asistente que realizará varias preguntas.

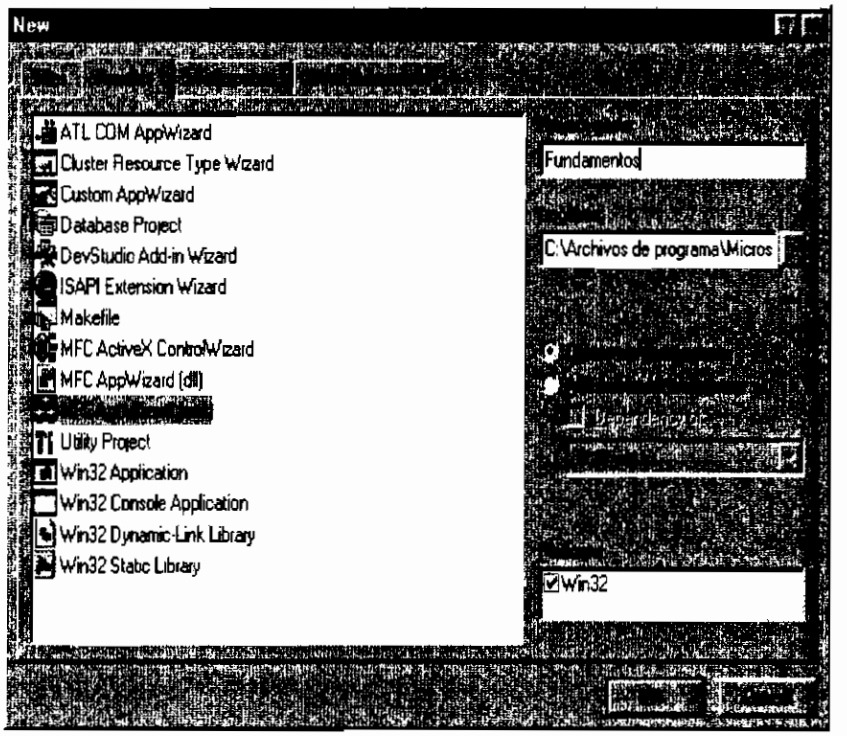


Figura.A.2 Dialogo para crear Nuevo Proyecto

A.9.2 CONFIGURACIÓN DEL APPWIZARD PARA CREAR LA INTERFAZ DE LA APLICACIÓN

El asistente AppWizard hará algunas preguntas acerca del tipo de aplicación a construirse:

- Si la aplicación se basará en diálogo, documento simple o múltiples documentos.
- Si desea algunas características a incluir, como ayudas o soporte para Winsock.
- Pide el nombre de la aplicación que se mostrará en la barra azul, común en una ventana de Windows.

- Si desea que aparezcan comentarios en la edición o incluir la biblioteca de la MFC como DLL; a lo cual es recomendable aceptar lo que el asistente marca como por defecto.
- Muestra las clases que el asistente crea, a lo cual se debe aceptar e inmediatamente el asistente procede a construir.

A.9.3 DISEÑO DE LA VENTANA DE LA APLICACIÓN

Este diseño se lo realiza ubicándose en el panel Workspace, debiendo seguir los siguientes pasos para poder modificar el diálogo a la necesidad:

- En la ficha Resources, se encuentra la ventana de diálogo creada por defecto mediante el App Wizard y que lleva el nombre que se escogió al expandir el árbol de recursos.
- Se debe colocar en el cuadro de diálogo los controles necesarios, arrastrándolos desde la barra de componentes, orientados a la aplicación.
- Al hacer clic derecho en cada recurso de la caja de diálogo se puede cambia propiedades como nombres e identificaciones. Esta propiedad ayuda a personalizar nombres para una rápida identificación.

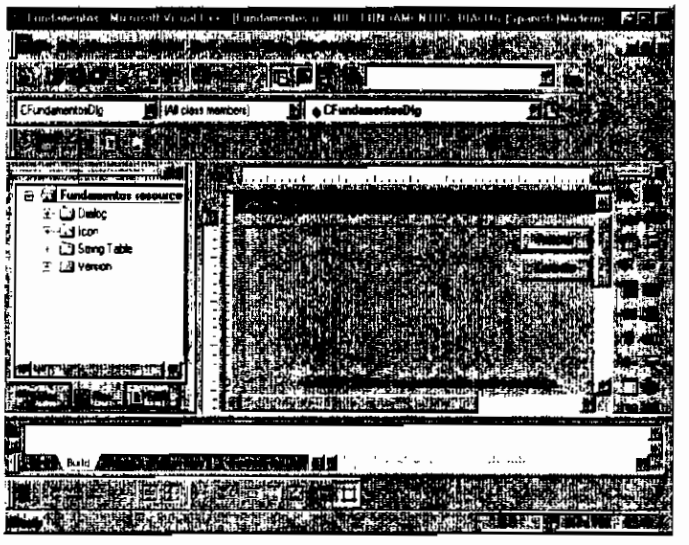


Figura A.3. Diseño de la Ventana de Dialogo

A.9.4 EDICIÓN DEL CÓDIGO DEL PROGRAMA

Utilizando el ClassWizard se puede agregar funcionalidad a una aplicación, para lo que debemos seguir los pasos siguientes:

- Para agregar funcionalidad a un botón de ejecución se debe dar clic con el botón derecho del Mouse, sobre este y selecciona la clase Wizard desde el menú contextual.
- La ventana de la clase Wizard mostrará marcado el identificador del botón, para lo que se debe proceder en la ficha a añadir una función. A esta acción el asistente requerirá que se le dé un nombre así como presenta la opción a escoger si la ejecución se dará al dar un clic sobre el botón o doble clic.
- Agregada la función, aparecerá en la lista Member Functions y se procede a editar el código haciendo clic sobre el botón Edit Code.

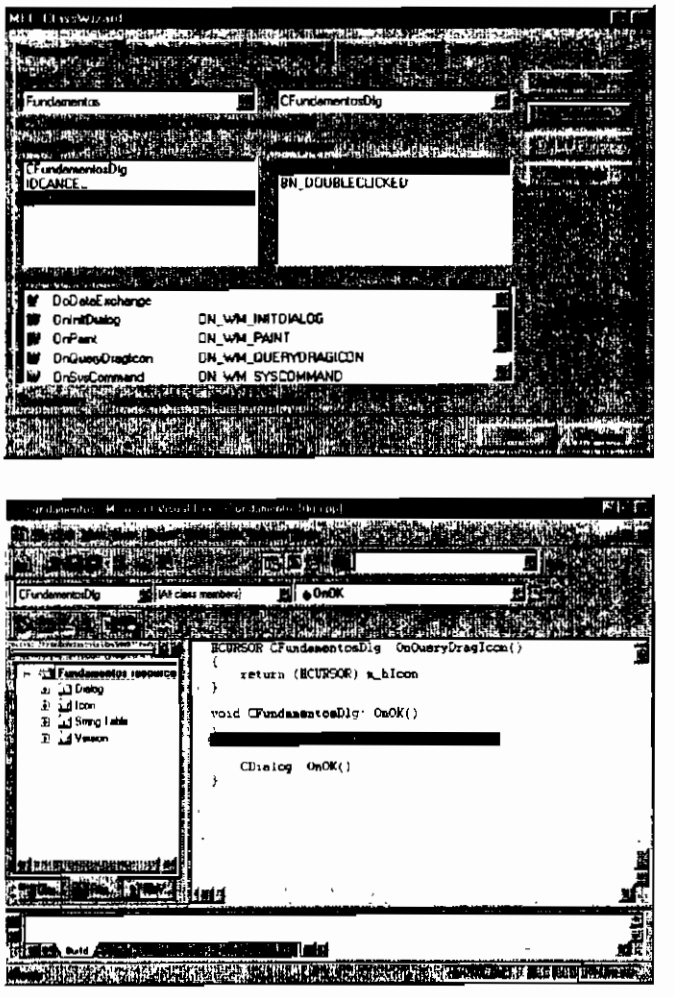


Figura A.4 Edición del código de programa

MANUAL DE USUARIO.

CONTENIDO

1. Introducción
2. Modo de operación
3. Trabajando en modo de envío
4. Trabajando en modos de recepción
5. Como visualizar paquete(s).
6. Requerimientos.

1. INTRODUCCIÓN

Este es un programa cuya interfaz gráfica lo hace sencillo de utilizar. Sin embargo usuario debe tener conciencia de lo que está haciendo para que utilice este programa, y esto supone un conocimiento previo y con cierto grado de profundidad de la pila de protocolos TCP/IP.

2. MODO DE OPERACIÓN

En general se observa una interfaz sencilla que permite al programa a trabajar en uno de dos modos ya sea como generador, en cuyo caso su función es *enviar* un paquete la red; o sea como colector, en cuyo caso la función del programa es capturar o *recibir* paquetes que circulen por la red.

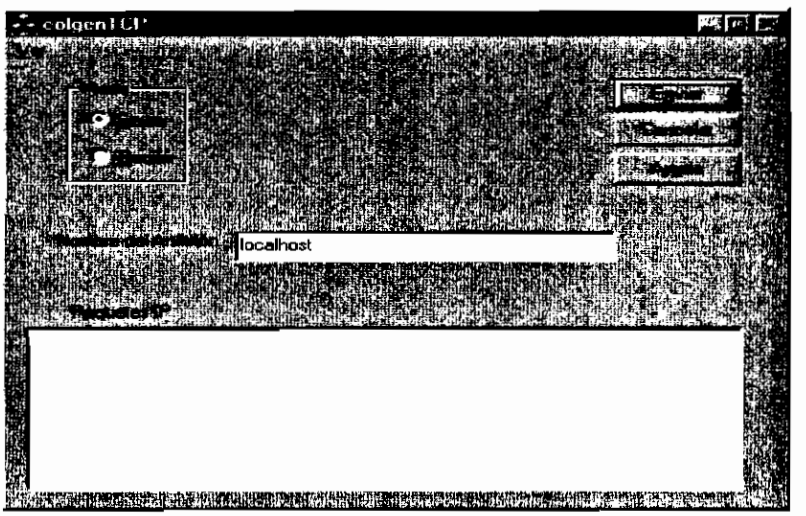


Figura.B.1 Interfaz Principal.

Con el ratón debe dar click en una de las opciónes Enviar o Recibir, que están agrupadas en el recuadro Modo.

3. TRABAJANDO EN MODO DE ENVÍO

Una vez que ha seleccionado modo-enviar, debe hacer click en el botón Enviar, a continuación aparece la siguiente caja de diálogo:

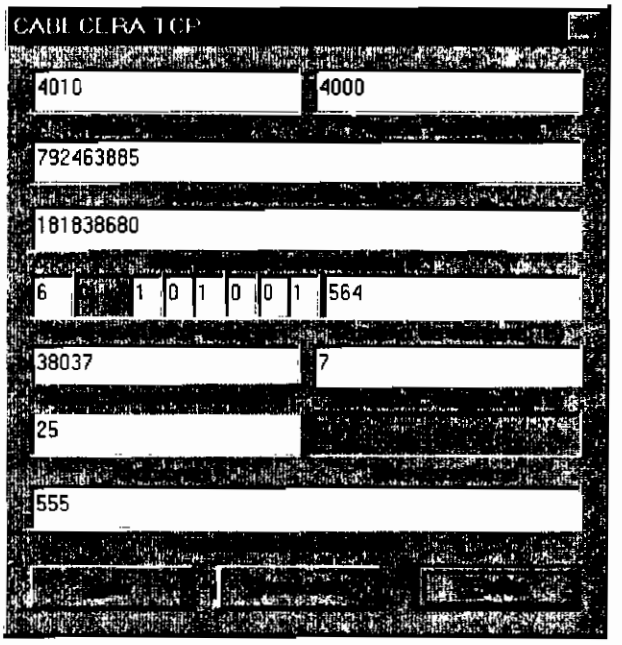


Figura B.2 Ventana para Ingreso de Datos

Para enviar un paquete inmediatamente haga click en el botón **CheckSum**, este calcula la suma de comprobación del paquete y finalmente haga click en el botón **OK** para enviar el paquete. Esta que es la estructura del encabezado de un paquete TCP, aparece con valores predeterminados por el programa, pero este diálogo permite al usuario modificar los valores, y poner los que el mismo creyera conveniente.

Estos valores de usuario debe ingresarlos con buen criterio, por lo que en a continuación se da una breve descripción de dichos campos.

El *puerto origen* y el *puerto destino*, ambos campos de 16 bits, identifican de manera efectiva las aplicaciones que envían y reciben los paquetes.

El campo *número de secuencia*, de 32 bits, identifica el primer byte de datos en el área de datos del segmento TCP. TCP identifica el byte por su

compensación (offset) relativa del inicio del flujo datos, es decir a cada byte del flujo de datos se le identifica por un número de secuencia.

El campo *confirmación*, de 32 bits, identifica el siguiente byte de datos que espera recibir la conexión del flujo de datos. Por ejemplo, si el último byte recibido tiene el número de secuencia de 1000, TCP envía el número 1001 como confirmación.

El campo longitud o *tamaño de cabecera TCP* utiliza 4 bits para especificar el tamaño del encabezado TCP en palabras de 32 bits, casi siempre de veinte bytes de longitud. El área de *datos* comienza inmediatamente después del encabezado TCP. Al examinar el campo longitud del encabezado, los módulos receptores TCP pueden calcular el inicio del área de datos como si fueran cuatro veces más bytes (palabras de 32 bits) desde el inicio del segmento TCP.

Banderas.

El encabezado TCP incluye seis campos de banderas de un bit (sólo toma el valor 1 ó 0) a continuación se da una breve descripción de cada bandera.

URG. Esta bandera se encarga de decir al módulo TCP receptor que el campo indicador de urgencia indica los datos urgentes (el módulo TCP debe procesarlos antes que cualquier otros).

ACK. Esta bandera informa al módulo TCP receptor que el campo número de confirmación contiene números de confirmación válidos. Como sabe, esta bandera ayuda a TCP a asegurar la confiabilidad de los datos.

PSH. Esta bandera solicita un empujón. Es decir, dice el módulo TCP receptor que envíe de inmediato el segmento de datos a su aplicación destino. Casi siempre los buffer del módulo TCP ingresan los datos. Así, TCP no espera para enviar el segmento de datos a su aplicación destino hasta que el buffer alcanza un umbral específico. La bandera PSH indica al módulo TCP que no debe llevar al buffer el segmento de datos. Por ejemplo una aplicación Telnet por lo común establece esta bandera y, al hacerlo, fuerza a TCP a pasar los impulsos del teclado del usuario al servidor de Telnet. Esto ayuda a eliminar los retrasos producidos al presentar los caracteres que se reciben de vuelta al emisor, la mayoría de los usuarios de Telnet desean ver lo que escriben cuando lo escriben.

RST. Esta solicita al módulo TCP receptor que le restablezca la conexión, TCP envía mensaje con la bandera RST cuando detecta problemas en la conexión. La mayoría de las aplicaciones simplemente terminan al recibir esta bandera. Sin embargo, puede utilizarla para diseñar programas mejorados que se recuperan de colisiones de hardware o software.

SYN. Esta bandera indicada al módulo TCP receptor que sincronice los números de secuencia. Como sabe, TCP utiliza esta bandera para decir al módulo TCP receptor que el emisor se prepara para transmitir un nuevo flujo de datos.

FIN. Esta bandera dice al módulo TCP receptor que el emisor término de transmitir datos. La bandera FIN solo cierra el flujo datos en la dirección en la que viaja. El módulo TCP receptor también debe enviar un mensaje con la bandera FIN para completar el cierre de la conexión.

El campo *tamaño de ventana*, de 16 bits, informa al módulo TCP receptor el número de bytes que el emisor está dispuesto a aceptar. Como aprendió, TCP utiliza una ventanas deslizantes de tamaño variable para mejorar la capacidad de transferencia y optimizar el ancho de banda de la red. El valor de este campo especifica el tamaño de la ventana deslizante. Lo común es que sea de varios miles de bytes.

El campo *suma de comprobación TCP* incluye en sus cálculos los datos TCP. TCP necesita que el emisor calcule e incluya la suma de comprobación en este campo; asimismo, requiere que los módulos TCP receptores revisen las sumas de comprobación cuando reciben los datos.

El campo *indicador de urgencia*, de 16 bits, especifica una localización de byte en el área datos. El propósito de la bandera URG y del indicador de urgencia es informar al módulo TCP receptor que existe algún tipo de *datos urgentes* y señalarle donde se localizan. Sin embargo, nadie ha definido adecuadamente el término de datos urgentes, ni la responsabilidad del módulo TCP receptor respecto al manejo de estos. Quizá lo más significativo es el hecho de que lo representado por esta localización de byte (incluso su ubicación) es tema de gran debate. Por el momento, tal vez convenga que se limite al usar el modo urgente de TCP.

Los módulos TCP emplean, por lo común, el campo opciones con la opción tamaño máximo del segmento. El tamaño máximo del segmento de TCP es similar a la unidad de transferencia máxima MTU de la capa física: define el tamaño máximo de mensaje que puede aceptar el módulo TCP. Como sabe, TCP optimiza el ancho de banda de la red incrementando la capacidad de transferencia. La opción tamaño máximo de segmento permite a los módulos TCP anunciar el tamaño mayor del segmento o mensaje que espera recibir. Los módulos TCP sólo pueden utilizar esta opción en un mensaje que contenga establecida la bandera SYN. No obstante esta bandera no es una opción negociable. Uno de los lados de la conexión TCP anuncia al otro que espera a un tamaño máximo de segmento de cierto valor. Si un módulo TCP no transmite un tamaño máximo del segmento, TCP supone, por omisión, un tamaño máximo de 536 bytes.

4. TRABAJANDO EN MODO DE RECEPCIÓN.

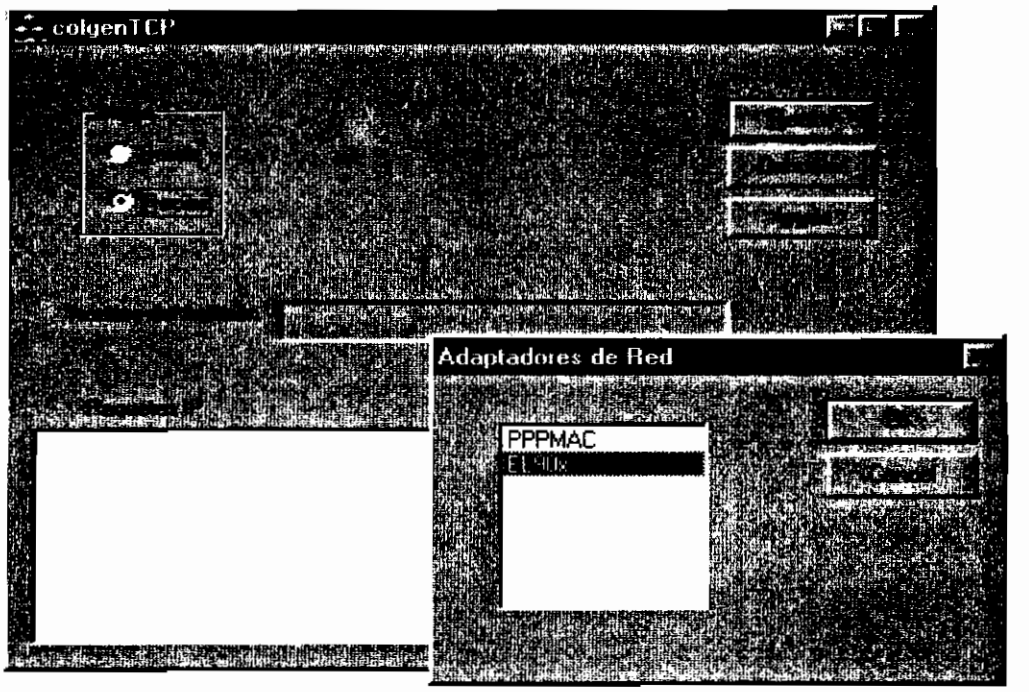


Figura B.3 Dialogo para seleccionar adaptador de red

Una vez que ha seleccionado trabajar en el modo Recibir, dando un click con él ratón en el botón con etiqueta **Recibir** aparece una caja de diálogo que le pregunta por medio de que adaptador o tarjeta de red quiere usted capturar los paquetes; para esto Ud. da un click en el adaptador seleccionado y finalmente hace click en el boton **OK** para capturar los paquetes.

Si hay paquetes capturados éstos aparecen en Lista de cuadro la interfaz principal, como en el siguiente ejemplo.

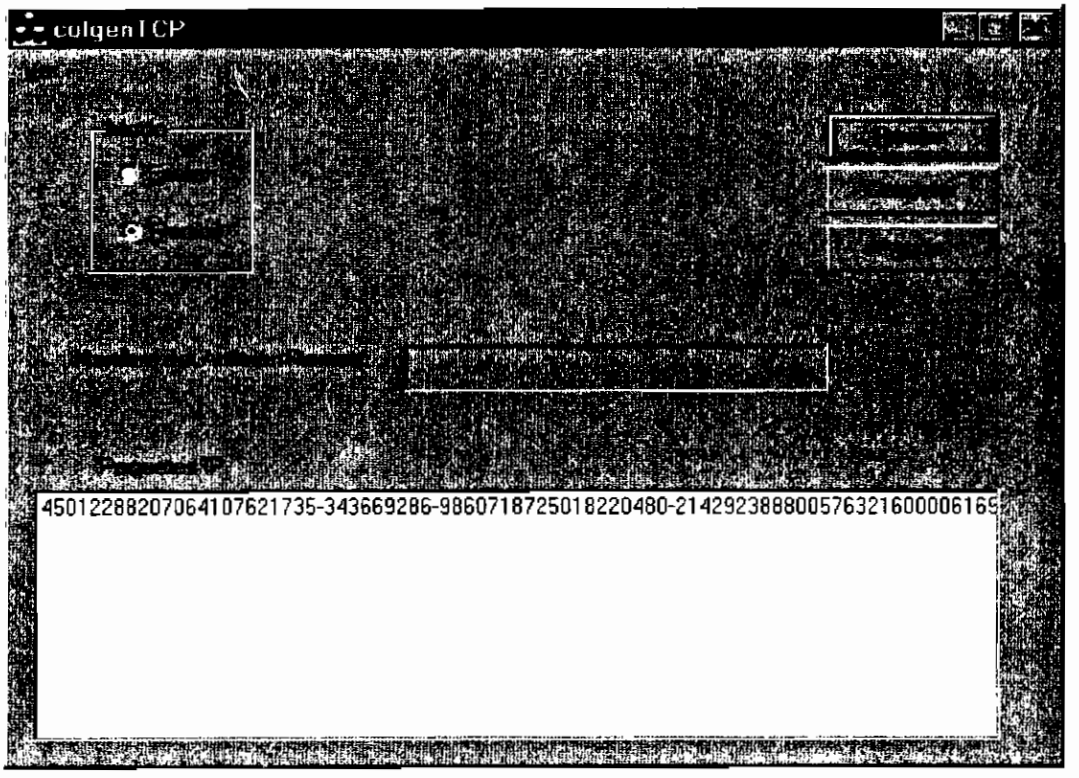


Figura B.4 Interfaz Principal lista paquetes capturados

Si en la lista no hay paquete (datos) hay que intentar la captura nuevamente.

Para mostrar el paquete capturado se utiliza en el menú Ver.

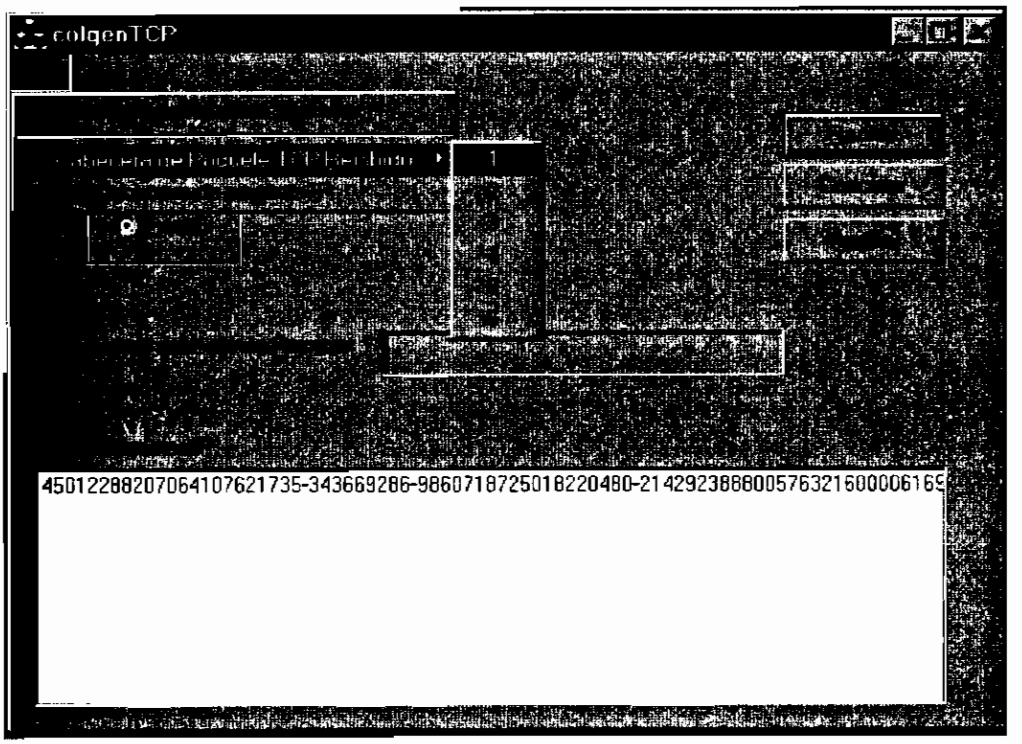


Figura B.5 Seleccionar del menu para ver el paquete 1.

Que despliega lo siguiente:

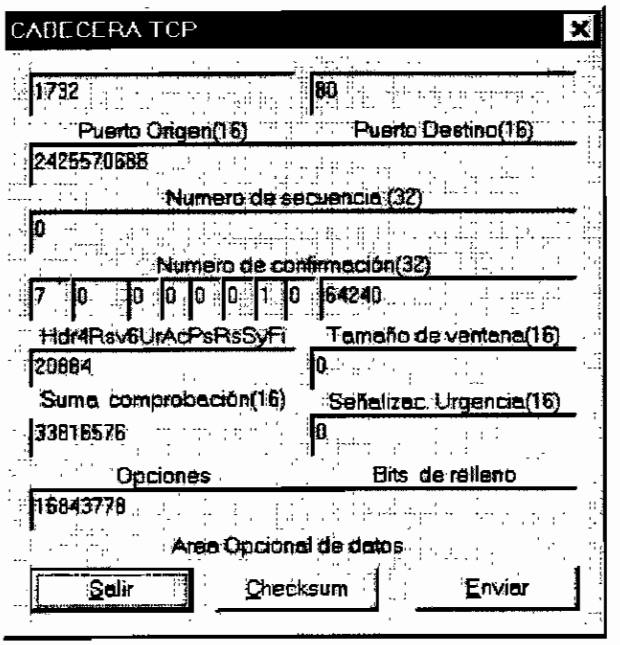


Figura B.6 Cabecera Paquete TCP 1.

Una vez visualizado el paquete TCP, se puede adicionalmente en el menú **Ver** visualizar el paquete o datagrama IP que lo contiene.

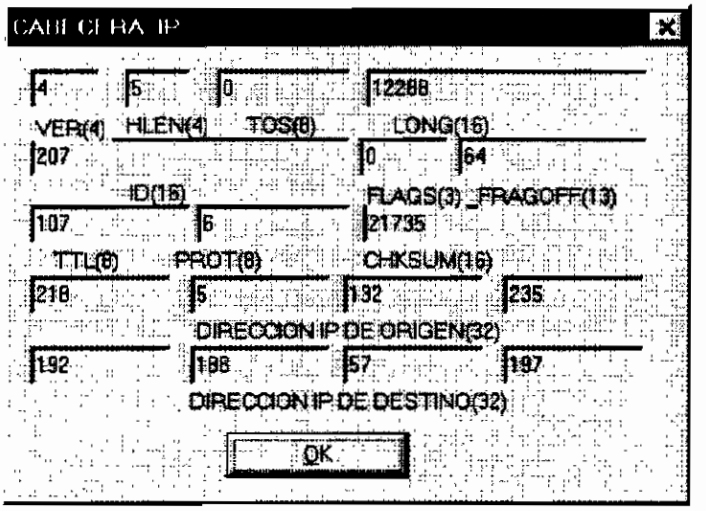


Figura B.7 Cabecera Paquete IP 1.

Y este proceso se sigue para visualizar cualquiera de los seis paquetes capturados si es que los hay.

De ser necesario se puede capturar estos cuadros de diálogo como cualquier otro en Windows utilizando las teclas PRINT SCREEN (imprimir pantalla) o ALT+PRINT SCREEN, para luego pegarlo en cualquier documento por ejemplo un archivó de WORD.

6. REQUERIMIENTOS.

Se recomienda en cuanto a Hardware se refiere lo siguiente:

- Pentium II (mínimo Pentium MMX)
- Memoria Ram 64MB (mínimo 32MB)
- Al menos 20MB libres en el disco duro
- Tener Instalado Visual C++ versión 6.0

```
// colgenTCPDlg.cpp : implementation file
//

#include "stdafx.h"
#include "colgenTCP.h"
#include "colgenTCPDlg.h"
#include "AdapterList.h"


//////////////////////////////////////////////////////////////////////////
#include       <afxpriv.h>

//////////////////////////////////////////////////////////////////////////

#include "IPHeader.h"
#include "TCPHeader.h"


#include <packet32.h>
#include <windows.h>
#include <stdio.h>
#include <conio.h>



#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////***************
#define SIMULTANEOU_READS 10
#define MAX_ETHERNET_FRAME_SIZE 1514

#define Max_Num_Adapter 10

// Prototypes
////////*************

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        //{{AFX_MSG(CAboutDlg)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
                // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CColgenTCPDlg dialog

CColgenTCPDlg::CColgenTCPDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CColgenTCPDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CColgenTCPDlg)
        m_iSend = -1;
        m_NomHost = _T("");
        //}}AFX_DATA_INIT
        // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CColgenTCPDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CColgenTCPDlg)
        DDX_Control(pDX, IDC_LISTDATA, m_ctlListData);
        DDX_Control(pDX, IDEJECT, m_ctlEject);
        DDX_Radio(pDX, IDC_RSEND, m_iSend);
        DDX_Text(pDX, IDC_NOMHOST, m_NomHost);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CColgenTCPDlg, CDialog)
        //{{AFX_MSG_MAP(CColgenTCPDlg)
        ON_WM_SYSCOMMAND()
        ON_WM_DESTROY()
        ON_WM_PAINT()
        ON_WM_QUERYDRAGICON()
        ON_BN_CLICKED(IDEJECT, OnEject)
        ON_WM_KEYDOWN()
        ON_BN_CLICKED(IDC_RSEND, OnRsend)
        ON_BN_CLICKED(IDC_RECEIVE, OnReceive)
        ON_COMMAND(ID_VER_PAQUETEENVIADO, OnVerPaqueteenviado)
        ON_COMMAND(ID_VER_CABECERADEPAQUETETCPRECIBIDO_1, OnVerCabeceradepaquetetcprecibido1)
        ON_COMMAND(ID_VER_CABECERADEPAQUETETCPRECIBIDO_2, OnVerCabeceradepaquetetcprecibido2)
        ON_COMMAND(ID_VER_CABECERADEPAQUETETCPRECIBIDO_3, OnVerCabeceradepaquetetcprecibido3)
        ON_COMMAND(ID_VER_CABECERADEPAQUETETCPRECIBIDO_4, OnVerCabeceradepaquetetcprecibido4)
        ON_COMMAND(ID_VER_CABECERADEPAQUETETCPRECIBIDO_5, OnVerCabeceradepaquetetcprecibido5)
        ON_COMMAND(ID_VER_CABECERADEPAQUETETCPRECIBIDO_6, OnVerCabeceradepaquetetcprecibido6)
        ON_COMMAND(ID_VER_CABECERAIP, OnVerCabeceraip)
        ON_COMMAND(ID_VER_VERCABECERADEPAQUETEIP, OnVerVercabeceradepaqueteip)
        ON_COMMAND(ID_VER_CABECERADEPAQUETEENVIADO, OnVerCabeceradepaqueteenviado)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CColgenTCPDlg message handlers

BOOL CColgenTCPDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        // Add "About..." menu item to system menu.

        // IDM_ABOUTBOX must be in the system command range.
        ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);

        CMenu* pSysMenu = GetSystemMenu(FALSE);
        if (pSysMenu != NULL)
        {
                CString strAboutMenu;
                strAboutMenu.LoadString(IDS_ABOUTBOX);
                if (!strAboutMenu.IsEmpty())
                {
                        pSysMenu->AppendMenu(MF_SEPARATOR);
                        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
                }
        }

        // Set the icon for this dialog.  The framework does this automatically
        //  when the application's main window is not a dialog
```

```
        SetIcon(m_hIcon, TRUE);                         // Set big icon
        SetIcon(m_hIcon, FALSE);                    // Set small icon

        // TODO: Add extra initialization here
        //////////*****************************
        CAdapterList m_sSetPointer;
        m_sSetPointer.SetParent(this);
        m_iSend =0;// Iniciamos la variable para función de envio
        verpr = 0;
        verpe = 0;
        verptcp = 0;
        m_NomHost = "localhost";
        UpdateData(FALSE);//Actualizamos el diálogo

        //////////*****************************

        return TRUE;  // return TRUE  unless you set the focus to a control
}

void CColgenTCPDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
                CAboutDlg dlgAbout;
                dlgAbout.DoModal();
        }
        else
        {
                CDialog::OnSysCommand(nID, lParam);
        }
}

void CColgenTCPDlg::OnDestroy()
{
        WinHelp(0L, HELP_QUIT);

//      PacketFreePacket(lpPacket);

        CDialog::OnDestroy();
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CColgenTCPDlg::OnPaint()
{
        if (IsIconic())
        {
                CPaintDC dc(this); // device context for painting

                SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

                // Center icon in client rectangle
                int cxIcon = GetSystemMetrics(SM_CXICON);
                int cyIcon = GetSystemMetrics(SM_CYICON);
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;

                // Draw the icon
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
                CDialog::OnPaint();
        }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CColgenTCPDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}
//////////////////////////////////////////////////////////////////////

void CColgenTCPDlg::OnReceive()
{
        // TODO: Add your control notification handler code here
```

```
        UpdateData(TRUE);       //Se debe actualizar el botón marcado

        m_NomHost  = "";        //Elimino dirección alguna

        GetDlgItem(IDC_NOMHOST)->EnableWindow(FALSE);//Deshabilitamos la variable

        UpdateData(FALSE);//Actualizo la ventana

        m_ctlEject.SetWindowText ("&Recibir");////Modifico rótulo de botón
        }

    ////////////////////////////////////////////////////////////////////////

    void CColgenTCPDlg::OnRsend()
    {
        // TODO: Add your control notification handler code here

        UpdateData(TRUE);       //Se debe actualizar el boton marcado

        GetDlgItem(IDC_NOMHOST)->EnableWindow(TRUE);//Habilito la variable
        m_NomHost  = "localhost";//Predetermino uan dirección de Host
        UpdateData(FALSE);//Retorno datos a la caja de edición.

        m_ctlEject.SetWindowText ("E&nviar");//Modifico rótulo de botón
        }

    ////////////////////////////////////////////////////////////////////////


    void CColgenTCPDlg::OnEject()
    {
        // TODO: Add your control notification handler code here
        ////////////**************************

        UpdateData(TRUE);
        CListBox                *pListBox = (CListBox *)GetDlgItem(IDC_LISTDATA);
        pListBox->ResetContent();  ////borra listbox
        UpdateData(FALSE);


        if (m_iSend == ENVIO)//        El RButon esta en modo de envio?
        {
    ////////////////////////////////////////////////////////**************
                mode = 1;
                verpe = 0;
                verpr = 0;

                LPBYTE pbuff;

                pbuff = (LPBYTE)TcpSendPacket;
                                //Especifico el buffer de envío como una estructura
                memset (pbuff,0,100);//Lleno de datos temporalmente

                UpdateData(TRUE); //Actualiza datos desde la ventana

                lpNomHost = T2A((LPTSTR)((LPCTSTR)m_NomHost));
                                                //Expresa en cadena de caracteres
                if (CColgenTCPDlg::GetIPAddress (lpNomHost)!= INADDR_NONE)
                                                //Resuelve la dirección ingresada
                {

                        if (TcpSocket != INVALID_SOCKET)
                        {
                                closesocket(TcpSocket);
                        }

                        TcpSocket = socket (PF_INET, SOCK_RAW,IPPROTO_IP );
                                        //Crea un socket básico y especifica a ICMP
                                        //como el protocolo a utilizar
                        if (TcpSocket == INVALID_SOCKET)

                        {
                                MessageBox("No puedo crear un socket básico !", "TCP"
                                , MB_OK|MB_ICONSTOP);
                                //Desplegamos un mensaje de no poder crear socket básico

                                return;
                        }
```

```
                pTcpHeader = (LPTcpHeader)TcpSendPacket;
                        //Especifico el buffer de envío como una estructura
                memset (pTcpHeader, 'E', TcpHeaderLength);
                        //Lleno de datos temporalmente
                memset (pTcpHeader, 0, TcpHeaderLength);
                        //Pongo en cero los datos

                // Llenado previo de datos Tcp en un buffer de envío


                pTcpHeader->TcpSrcPort = htons(4010);
                pTcpHeader->TcpDestPort= htons(4000);
                pTcpHeader->TcpSeqNum = htonl(792463885);
                pTcpHeader->TcpAckNum = htonl(181838680);
                pTcpHeader->TcpHeadChkFlgs = htons(24617);
                pTcpHeader->TcpWndSize=htons(564);
                pTcpHeader->TcpChksum=htons(0);
                pTcpHeader->TcpUrgPoint=htons(7);
                pTcpHeader->TcpOptions=htons(25);
                pTcpHeader->TcpData=htonl(555);
                pTcpHeader->TcpChksum=(Chksum((LPWORD)pTcpHeader,TcpHeaderLength ));


        // Llamo a desplegar datos

                CTCPHeader varCTCPHeader;
                varCTCPHeader.SetParent(this);
                varCTCPHeader.DoModal();

                int ResultClose = closesocket(TcpSocket);
                if (ResultClose == SOCKET_ERROR)
                {
                        MessageBox("No se puede cerrar el socketet básico !","TCP",
                        MB_OK|MB_ICONSTOP);
                        //Desplegamos un mensaje de no poder crear socket básico

                        return;
                }

                return;
        }

        MessageBox("Anfitrión No Alcanzable !","TCP",
        MB_OK|MB_ICONSTOP);
        //Desplegamos un mensaje de no poder crear socket básico
        return ;

    }


////////////////////////////////////////////////////////**************
    if (m_iSend == RECEPCION)

    {
        memset (buffer,0,256000);
        for (i=0;i<10;i++)
        {
                memset(AdapterList[i],0,1024);

        }
        //define a pointer to an ADAPTER structure e inicialico
        lpAdapter = 0;
        AdapterNum=0;
        AdapterLength=512;

        i=0;
        {
                PacketGetAdapterNames(AdapterNames,&AdapterLength);
                tempa=AdapterNames;
                tempa1=AdapterNames;
                while ((*tempa!='\0')||(*(tempa-1)!='\0'))
                {
                        if (*tempa=='\0')
                        {
                                memcpy(AdapterList[i],tempa1,tempa-tempa1);
                                tempa1=tempa+1;
                                i++;
                        }
                        tempa++;
                }
                AdapterNum=i;
```

```
                        CAdapterList varCSelectAdapter;
                        varCSelectAdapter.SetParent(this);
                        varCSelectAdapter.DoModal();
                        if (m_Adaptern==20)
                        {
                                return;
                        }
                        if ((Open = varCSelectAdapter.nIndex)==-1)
                        {
                                return;
                        }

                        lpAdapter =   PacketOpenAdapter(AdapterList[Open]);
                        if (!lpAdapter || (lpAdapter->hFile == INVALID_HANDLE_VALUE))
                        {
                                dwErrorCode=GetLastError();
                                MessageBox(
                                "Error: No es posible abrir el controlador del adaptador de red");
                                return;
                        }
                        // set the network adapter in promiscuous mode
                        PacketSetHwFilter(lpAdapter,NDIS_PACKET_TYPE_PROMISCUOUS);
                        // set a 512K buffer in the driver
                        PacketSetBuff(lpAdapter,512000);
                        // set a 1 second read timeout
                        PacketSetReadTimeout(lpAdapter,1000);
                        //allocate and initialize a packet structure that will be used to
                        //receive the packets.
                        if((lpPacket = PacketAllocatePacket())==NULL)
                        {
                                MessageBox(" Falla en asignar estructura PACKET");
                                return ;
                        }
                        PacketInitPacket(lpPacket,(char*)buffer,256000);
                                if(PacketReceivePacket(lpAdapter,lpPacket,TRUE)==FALSE)
                                {
                                        MessageBox("Error: Falla en la recepción de paquetes");
                                        return ;
                                }
                                LPBYTE lpDataEth;
                                lpDataEth = (LPBYTE)lpPacket->Buffer;
                                CColgenTCPDlg::PrintPackets(lpPacket);
                        //print the capture statistics
                        PacketGetStats(lpAdapter,&stat);
                        //      printf("\n\n%d packets received.\n%d Packets lost",
                        //                                      stat.bs_recv,stat.bs_drop);

                        PacketCloseAdapter(lpAdapter);// close the adapter and exit
                        verpr = 1;
                }
        }


}


// this function prints the content of a block of packets received from the driver

void CColgenTCPDlg::PrintPackets(LPPACKET lpPacket)
{

ulBytesReceived = lpPacket->ulBytesReceived;
buf =(char*) lpPacket->Buffer;
off=0;

hdr=(struct bpf_hdr *)(buf+off);
tlen1=hdr->bh_datalen;
tlen=hdr->bh_caplen;

char listsecenv[120];


        k =0;
        while(off<ulBytesReceived)
        {

                hdr=(struct bpf_hdr *)(buf+off);
                tlen1=hdr->bh_datalen;
                tlen=hdr->bh_caplen;

                off+=hdr->bh_hdrlen;
```

```
			pChar =(char*)(buf+off);

			off=Packet_WORDALIGN(off+tlen);

			pIpHeader = (IpHeader *)(pChar+14);

			if((pIpHeader->ip_proto == 6)||(pIpHeader->ip_proto == 0))
			{
				pPacket[k] =  pIpHeader;
				k=k+1;
				pTcpHeader = (TcpHeader *)((LPBYTE)pIpHeader+((pIpHeader->ip_verlen & 0x0f)*4));

	wsprintf(listsecenv,"%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d",
				(pIpHeader->ip_verlen & 0xf0)>>4,
					(pIpHeader->ip_verlen & 0x0f),
					pIpHeader->ip_tos,
					pIpHeader->ip_len,pIpHeader->ip_id,
					((pIpHeader->ip_fragoff & 0xE000)>>13),
					(pIpHeader->ip_fragoff & 0x1FFF),
					pIpHeader->ip_ttl,
					pIpHeader->ip_proto,
					pIpHeader->ip_chksum,
					pIpHeader->ip_src_addr.S_un.S_addr,
					pIpHeader->ip_dst_addr.S_un.S_addr,

					pTcpHeader->TcpSrcPort,
					pTcpHeader->TcpDestPort,
					pTcpHeader->TcpSeqNum,
					pTcpHeader->TcpAckNum,
					(pTcpHeader->TcpHeadChkFlgs & 0xF000),
					(pTcpHeader->TcpHeadChkFlgs & 0x0FC0),
					(pTcpHeader->TcpHeadChkFlgs & 0x0020),
					(pTcpHeader->TcpHeadChkFlgs & 0x0010),
					(pTcpHeader->TcpHeadChkFlgs & 0x0008),
					(pTcpHeader->TcpHeadChkFlgs & 0x0004),
					(pTcpHeader->TcpHeadChkFlgs & 0x0002),
					(pTcpHeader->TcpHeadChkFlgs & 0x0001),
					pTcpHeader->TcpWndSize,
					pTcpHeader->TcpChksum,
					pTcpHeader->TcpUrgPoint,
					pTcpHeader->TcpOptions,
					pTcpHeader->TcpData);

				m_ctlListData.AddString(listsecenv);
			}

	}
}

void CColgenTCPDlg::SetAdaptern(DWORD Adaptern)
{
//Establece el apuntador de la variable miembro
	m_Adaptern = Adaptern;
}

void CColgenTCPDlg::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
	// TODO: Add your message handler code here and/or call default
	return;
	CDialog::OnKeyDown(nChar, nRepCnt, nFlags);
}



void CColgenTCPDlg::OnVerPaqueteenviado()
{
	// TODO: Add your command handler code here

}
unsigned long CColgenTCPDlg::GetIPAddress(LPSTR lpstrAddr)
{

	memset(&TcpSockAddr,0,sizeof(TcpSockAddr));
						//Ponemos en cero la estructura SOCKADDR_IN
	unsigned long addr = inet_addr(lpstrAddr);
	//Convierte la dirección IP en una cadena en notación decimal

	hostdest = NULL;
```

```
        if (addr == INADDR_NONE)
                        //Consultamos si la cadena correspondia a una dirección
        {

                hostdest = gethostbyname(lpstrAddr);//No corresponde a una dirección ,
                                                    //entonces resuelve el nombre del anfitrión

                if (hostdest)
                        //Si es obtiene la dirección IP como un entero largo sin signo
                {
                        memcpy(&(TcpSockAddr.sin_addr),hostdest->h_addr,hostdest->h_length);
                                        //Copiamos la dirección IP desde la estructura de
                                                //información del anfitrión en internet
                        TcpSockAddr.sin_family = hostdest->h_addrtype;
                }

                else

                {
                        MessageBox("No puedo obtener la dirección IP !",
                        lpstrAddr, MB_OK|MB_ICONINFORMATION);
                        //Desplegamos un mensaje de no poder obtener la dirección del anfitrión
                        return INADDR_NONE;
                }

        }

        else

        {
                TcpSockAddr.sin_addr.s_addr = addr;
                //Recuperamos la dirección IP de la cadena en notación decimal
                TcpSockAddr.sin_family = AF_INET;
                //LLenanmos la estructura SOCKADDR_IN necesaria para llenar TCB
        }
        return TcpSockAddr.sin_addr.s_addr;
                                //Retorna la dirección del anfitrión en notacion decimal
}


void CColgenTCPDlg::OnVerCabeceradepaquetetcprecibido1()
{
        // TODO: Add your command handler code here
        if ((verpr == 1)&&(k>0))
        {
                        pIpHeader = pPacket[0];
                        CTCPHeader varCTCPHeader;
                        varCTCPHeader.SetParent(this);
                        varCTCPHeader.DoModal();
                        verptcp=1;      //comprueba que primero se vea paquete TCP
        }
}

void CColgenTCPDlg::OnVerCabeceradepaquetetcprecibido2()
{
        // TODO: Add your command handler code here
if ((verpr == 1)&&(k-1>0))
        {
                        pIpHeader = pPacket[1];
                        CTCPHeader varCTCPHeader;
                        varCTCPHeader.SetParent(this);
                        varCTCPHeader.DoModal();


        }
}

void CColgenTCPDlg::OnVerCabeceradepaquetetcprecibido3()
{
        // TODO: Add your command handler code here
        if ((verpr == 1)&&(k-2>0))
        {
                        pIpHeader = pPacket[2];
                        CTCPHeader varCTCPHeader;
                        varCTCPHeader.SetParent(this);
                        varCTCPHeader.DoModal();

        }

}
```

```
void CColgenTCPDlg::OnVerCabeceradepaquetetcprecibido4()
{
        // TODO: Add your command handler code here
        if ((verpr == 1)&&(k-3>0))
        {
                        pIpHeader = pPacket[3];
                        CTCPHeader varCTCPHeader;
                        varCTCPHeader.SetParent(this);
                        varCTCPHeader.DoModal();


        }
}

void CColgenTCPDlg::OnVerCabeceradepaquetetcprecibido5()
{
        // TODO: Add your command handler code here
        if ((verpr == 1)&&(k-4>0))
        {
                        pIpHeader = pPacket[4];
                        CTCPHeader varCTCPHeader;
                        varCTCPHeader.SetParent(this);
                        varCTCPHeader.DoModal();

        }
}

void CColgenTCPDlg::OnVerCabeceradepaquetetcprecibido6()
{
        // TODO: Add your command handler code here
        if ((verpr == 1)&&(k-5>0))
        {
                        pIpHeader = pPacket[5];
                        CTCPHeader varCTCPHeader;
                        varCTCPHeader.SetParent(this);
                        varCTCPHeader.DoModal();

        }
}


void CColgenTCPDlg::OnVerCabeceraip()
{
        // TODO: Add your command handler code here
                if (verpr == 1)
        {
                        CIPHeader varCIPHeader;
                        varCIPHeader.SetParent(this);
                        varCIPHeader.DoModal();

        }

}

void CColgenTCPDlg::OnVerVercabeceradepaqueteip()
{
        // TODO: Add your command handler code here
        if ((verpr == 1)&&(verptcp == 1))
        //comprueba que primero se vea paquete recibido TCP
        {
                        CIPHeader varCIPHeader;
                        varCIPHeader.SetParent(this);
                        varCIPHeader.DoModal();

        }
        return;
}

WORD CColgenTCPDlg::Chksum(LPWORD lpwTcpData, WORD wDataLength)
{
        ////////////////////////
        //AQUI INICIA MI CODIGO
        ////////////////////////
        long    lSum;                           // Almacena la sumatoria
        WORD    wOddByte;                       // Byte sobre la izquierda de la sumatoria
        WORD    wAnswer;                        // Suma de comprobación con complemento a 1's

        lSum = 0L;

        while (wDataLength > 1)
                {
```

```
                lSum += *lpwTcpData++;
                wDataLength -= 2;
        }

        // Maneja el byte impar si es necesario y asegura que la mitad superior sea cero
        if (wDataLength == 1)
        {
                wOddByte = 0;
                *((LPBYTE) &wOddByte) = *(LPBYTE)lpwTcpData; // Un byte solamente
                lSum += wOddByte;
        }

        // Agrega los extras de los 16 bits a los 16 bits inferiores
        lSum = (lSum >> 16) + (lSum & 0xffff);
                                // Agrega los 16 bits superiores a los 16 bits inferiores
        lSum += (lSum >> 16);           // Agrega el extra
        wAnswer = (WORD)~lSum;          // complemento a 1's, entonces trunca a 16 bits


        return(wAnswer);

        ////////////////////////
        //AQUI TERMINA MI CODIGO
        ////////////////////////
}

void CColgenTCPDlg::Enviar()  //funcion para el envio del paquete
{
//Envio

        int offset = 0;//Se debe calcular el offset de datos ingresados

        int iTcpBufferSize = TcpHeaderLength + offset;
        int ResultSend = sendto (TcpSocket,(LPSTR)TcpSendPacket,iTcpBufferSize,0,
                (LPSOCKADDR)&TcpSockAddr, sizeof (TcpSockAddr));

        if (ResultSend == SOCKET_ERROR)
                {
                        MessageBox("No se pudo enviar el paquete TCP!", "TCP",

                                                                MB_OK|MB_ICONSTOP);

                        //Desplegamos un mensaje de no poder crear socket básico
                        int icmpSocketError = WSAGetLastError();

                }
        verpe = 1;      //se envio el paquete

}

void CColgenTCPDlg::OnVerCabeceradepaqueteenviado()
{
        // TODO: Add your command handler code here
        if ((verpe == 1)&&(verpr == 0)) //evalua si se ha enviado algun paquete
        {
                CTCPHeader varCTCPHeader;
                varCTCPHeader.SetParent(this);
                varCTCPHeader.DoModal();//despliega paquete enviado
        }
        return;
}
```

```
// AdapterList.cpp : implementation file
//

#include "stdafx.h"
#include "colgenTCP.h"
#include "AdapterList.h"
//////**********
//#include "colgenTCPDlg.h"
/////***********
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAdapterList dialog


CAdapterList::CAdapterList(CWnd* pParent /*=NULL*/)
	: CDialog(CAdapterList::IDD, pParent)
{
	//{{AFX_DATA_INIT(CAdapterList)
	//}}AFX_DATA_INIT
}


void CAdapterList::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAdapterList)
	DDX_Control(pDX, IDC_LISTADAPTERS, m_ctlListAdapters);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CAdapterList, CDialog)
	//{{AFX_MSG_MAP(CAdapterList)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAdapterList message handlers

BOOL CAdapterList::OnInitDialog()
{
	CDialog::OnInitDialog();

	// TODO: Add extra initialization here

	//////////**********************


		CString s2,s3,s4 ;
		CColgenTCPDlg* lpColgenTCPDlg = (CColgenTCPDlg*)m_pWnd;
		CColgenTCPDlg lpColgenTCPDlg1;
		int i;
		for (i=0;i<lpColgenTCPDlg->AdapterNum;i++)
		{
			m_ctlListAdapters.AddString(lpColgenTCPDlg->AdapterList[i]);

			UpdateData(FALSE);
		}

	//////////***************

	return TRUE;  // return TRUE unless you set the focus to a control
		      // EXCEPTION: OCX Property Pages should return FALSE
}

void CAdapterList::SetParent(CDialog* pWnd)
{
//Establece el apuntador de la variable miembro
	m_pWnd = pWnd;
}

void CAdapterList::OnOK()
{
	// TODO: Add extra validation here
```

```
        nIndex = m_ctlListAdapters.GetCurSel();
        if (nIndex == -1)
        {

                #define IDS_PLEASE_SELECT_COLOR 1
                CString str;
                str.LoadString(1);
                MessageBox("Seleccione un adaptador de la lista");
                return;
        }

                color = m_ctlListAdapters.GetItemData(nIndex);

                CColgenTCPDlg varCColgenTCPDlg;
                varCColgenTCPDlg.SetAdaptern(nIndex);


        CDialog::OnOK();
}

void CAdapterList::OnCancel()
{
        // TODO: Add extra cleanup here
        nIndex = 20;
        CDialog::OnCancel();
}
```

```
// TCPHeader.cpp : implementation file
//

#include "stdafx.h"
#include "ColgenTCP.h"
#include "TCPHeader.h"
#include "ColgenTCPDlg.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CTCPHeader dialog


CTCPHeader::CTCPHeader(CWnd* pParent /*=NULL*/)
	: CDialog(CTCPHeader::IDD, pParent)
{
	//{{AFX_DATA_INIT(CTCPHeader)
	m_iSrcP = 0;
	m_iDestP = 0;
	m_iAck = 0;
	m_iBFill = 0;
	m_iChkSum = 0;
	m_iFin = 0;
	m_iOpt = 0;
	m_iPsh = 0;
	m_iRst = 0;
	m_iSyn = 0;
	m_iUrg = 0;
	m_iWndSize = 0;
	m_iUrgP = 0;
	m_iAckN = 0;
	m_iSeqN = 0;
	m_iDataOpt = 0;
	m_iTcpHl = 0;
	m_iReserv = 0;
	//}}AFX_DATA_INIT
}


void CTCPHeader::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CTCPHeader)
	DDX_Text(pDX, IDC_SRCP, m_iSrcP);
	DDV_MinMaxInt(pDX, m_iSrcP, 0, 65535);
	DDX_Text(pDX, IDC_DESTP, m_iDestP);
	DDV_MinMaxInt(pDX, m_iDestP, 0, 65535);
	DDX_Text(pDX, IDC_ACK, m_iAck);
	DDV_MinMaxInt(pDX, m_iAck, 0, 1);
	DDX_Text(pDX, IDC_BFILL, m_iBFill);
	DDX_Text(pDX, IDC_CHKSUM, m_iChkSum);
	DDV_MinMaxInt(pDX, m_iChkSum, 0, 65535);
	DDX_Text(pDX, IDC_FIN, m_iFin);
	DDV_MinMaxInt(pDX, m_iFin, 0, 1);
	DDX_Text(pDX, IDC_OPT, m_iOpt);
	DDV_MinMaxInt(pDX, m_iOpt, 0, 65535);
	DDX_Text(pDX, IDC_PSH, m_iPsh);
	DDV_MinMaxInt(pDX, m_iPsh, 0, 1);
	DDX_Text(pDX, IDC_RST, m_iRst);
	DDV_MinMaxInt(pDX, m_iRst, 0, 1);
	DDX_Text(pDX, IDC_SYN, m_iSyn);
	DDV_MinMaxInt(pDX, m_iSyn, 0, 1);
	DDX_Text(pDX, IDC_URG, m_iUrg);
	DDV_MinMaxInt(pDX, m_iUrg, 0, 1);
	DDX_Text(pDX, IDC_WNDSIZE, m_iWndSize);
	DDV_MinMaxInt(pDX, m_iWndSize, 0, 65535);
	DDX_Text(pDX, IDC_URGP, m_iUrgP);
	DDV_MinMaxInt(pDX, m_iUrgP, 0, 65535);
	DDX_Text(pDX, IDC_ACKN, m_iAckN);
	DDV_MinMaxDWord(pDX, m_iAckN, 0, 4294967295);
	DDX_Text(pDX, IDC_SEQN, m_iSeqN);
	DDV_MinMaxDWord(pDX, m_iSeqN, 0, 4294967295);
	DDX_Text(pDX, IDC_DATAOPT, m_iDataOpt);
```

```
        DDV_MinMaxDWord(pDX, m_iDataOpt, 0, 4294967295);
        DDX_Text(pDX, IDC_TCPHL, m_iTcpHl);
        DDV_MinMaxInt(pDX, m_iTcpHl, 5, 15);
        DDX_Text(pDX, IDC_RESERV, m_iReserv);
        DDV_MinMaxInt(pDX, m_iReserv, 0, 65);
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CTCPHeader, CDialog)
        //{{AFX_MSG_MAP(CTCPHeader)
        ON_BN_CLICKED(IDC_BCHK, OnBchk)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTCPHeader message handlers

BOOL CTCPHeader::OnInitDialog()
{
        CDialog::OnInitDialog();

        // TODO: Add extra initialization here

        GetDlgItem(IDC_BCHK)->EnableWindow(FALSE);//desabilita checksum

        GetDlgItem(IDOK)->EnableWindow(FALSE);
                                                        //desabilita envio
        UpdateData(FALSE);

        CColgenTCPDlg* lpColgenTCPDlg = (CColgenTCPDlg*)m_pWnd;

        if (lpColgenTCPDlg->mode == 1)        //esta en modo envio

        {
                if (lpColgenTCPDlg->verpe == 0)       //evalua si envio paquete
                {
                        GetDlgItem(IDC_BCHK)->EnableWindow(TRUE);
                                                        //habilita checksum
                        GetDlgItem(IDOK)->EnableWindow(TRUE);
                                                        //habilita envio
                }
                LPTcpHeader pTcpHeader;
                pTcpHeader =(TcpHeader *)(lpColgenTCPDlg->pTcpHeader);


                        m_iSrcP = ntohs(pTcpHeader->TcpSrcPort);
                        m_iDestP =ntohs(pTcpHeader->TcpDestPort);
                        m_iSeqN = ntohl(pTcpHeader->TcpSeqNum);
                        m_iAckN = ntohl(pTcpHeader->TcpAckNum);
                        m_iTcpHl =(ntohs(pTcpHeader->TcpHeadChkFlgs) & 0xF000)>>12;
                        m_iReserv =(ntohs(pTcpHeader->TcpHeadChkFlgs) & 0x0FC0)>>6;
                        m_iUrg =(ntohs(pTcpHeader->TcpHeadChkFlgs) & 0x0020)>>5;
                        m_iAck =(ntohs(pTcpHeader->TcpHeadChkFlgs) & 0x0010)>>4;
                        m_iPsh =(ntohs(pTcpHeader->TcpHeadChkFlgs) & 0x0008)>>3;
                        m_iRst =(ntohs(pTcpHeader->TcpHeadChkFlgs) & 0x0004)>>2;
                        m_iSyn =(ntohs(pTcpHeader->TcpHeadChkFlgs) & 0x0002)>>1;
                        m_iFin =ntohs(pTcpHeader->TcpHeadChkFlgs) & 0x0001;
                        m_iWndSize = ntohs(pTcpHeader->TcpWndSize);
                        m_iChkSum =ntohs( pTcpHeader->TcpChksum);
                        m_iUrgP =ntohs(pTcpHeader->TcpUrgPoint);
                        m_iOpt =ntohs(pTcpHeader->TcpOptions);
                        m_iDataOpt =ntohl(pTcpHeader->TcpData);
                        UpdateData(FALSE);
        }
        else    //esta en modo recibir
        {
        CColgenTCPDlg* lpColgenTCPDlg = (CColgenTCPDlg*)m_pWnd;
        LPTcpHeader pTcpHeader;
//      pTcpHeader = (TcpHeader *)(pIpHeader+((pIpHeader->ip_verlen & 0x0f)*4));

        pTcpHeader =(LPTcpHeader)((LPBYTE)(lpColgenTCPDlg->pIpHeader)+
                ((lpColgenTCPDlg->pIpHeader->ip_verlen & 0x0f)*4));//(LPIpHeader)m_pIpBuffer;


                        m_iSrcP = ntohs(pTcpHeader->TcpSrcPort);
                        m_iDestP =ntohs( pTcpHeader->TcpDestPort);
                        m_iSeqN = ntohl(pTcpHeader->TcpSeqNum);
                        m_iAckN = ntohl(pTcpHeader->TcpAckNum);
                        m_iTcpHl =(ntohs(pTcpHeader->TcpHeadChkFlgs) & 0xF000) >>12;
```

```
                        m_iReserv =(ntohs(pTcpHeader->TcpHeadChkFlgs) & 0x0FC0)>>6;
                        m_iUrg =(ntohs(pTcpHeader->TcpHeadChkFlgs) & 0x0020)>>5;
                        m_iAck =(ntohs(pTcpHeader->TcpHeadChkFlgs) & 0x0010)>>4;
                        m_iPsh =(ntohs(pTcpHeader->TcpHeadChkFlgs) & 0x0008)>>3;
                        m_iRst =(ntohs(pTcpHeader->TcpHeadChkFlgs) & 0x0004)>>2;
                        m_iSyn =(ntohs(pTcpHeader->TcpHeadChkFlgs) & 0x0002)>>1;
                        m_iFin =(ntohs(pTcpHeader->TcpHeadChkFlgs) & 0x0001);
                        m_iWndSize =ntohs( pTcpHeader->TcpWndSize);
                        m_iChkSum =ntohs( pTcpHeader->TcpChksum);
                        m_iUrgP =ntohs(pTcpHeader->TcpUrgPoint);
                        m_iOpt =ntohl(pTcpHeader->TcpOptions);
                        m_iDataOpt =ntohl(pTcpHeader->TcpData);

                        UpdateData(FALSE);



                        GetDlgItem(IDC_SRCP)->EnableWindow(FALSE);
                        GetDlgItem(IDC_DESTP)->EnableWindow(FALSE);
                        GetDlgItem(IDC_SEQN)->EnableWindow(FALSE);
                        GetDlgItem(IDC_ACKN)->EnableWindow(FALSE);
                        GetDlgItem(IDC_TCPHL)->EnableWindow(FALSE);
                        GetDlgItem(IDC_RESERV)->EnableWindow(FALSE);
                        GetDlgItem(IDC_URG)->EnableWindow(FALSE);
                        GetDlgItem(IDC_ACK)->EnableWindow(FALSE);
                        GetDlgItem(IDC_PSH)->EnableWindow(FALSE);
                        GetDlgItem(IDC_RST)->EnableWindow(FALSE);
                        GetDlgItem(IDC_SYN)->EnableWindow(FALSE);
                        GetDlgItem(IDC_FIN)->EnableWindow(FALSE);
                        GetDlgItem(IDC_WNDSIZE)->EnableWindow(FALSE);
                        GetDlgItem(IDC_CHKSUM)->EnableWindow(FALSE);
                        GetDlgItem(IDC_URGP)->EnableWindow(FALSE);
                        GetDlgItem(IDC_OPT)->EnableWindow(FALSE);
                        GetDlgItem(IDC_BFILL)->EnableWindow(FALSE);
                        GetDlgItem(IDC_DATAOPT)->EnableWindow(FALSE);
                        //GetDlgItem(IDC_BCHK)->EnableWindow(FALSE);


        }
        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
void CTCPHeader::SetParent(CDialog* pWnd)
{
//Establece el apuntador de la variable miembro
        m_pWnd = pWnd;
}


void CTCPHeader::OnOK()
{
        // TODO: Add extra validation here
        CColgenTCPDlg* lpColgenTCPDlg = (CColgenTCPDlg*)m_pWnd;

        //LPTcpHeader pTcpHeader;
        if (lpColgenTCPDlg->mode == 1)     //modo envio

        {

        CColgenTCPDlg* lpColgenTCPDlg = (CColgenTCPDlg*)m_pWnd;
        LPTcpHeader pTcpHeader;
        pTcpHeader =(LPTcpHeader)lpColgenTCPDlg->pTcpHeader;

                        UpdateData(TRUE);
                        pTcpHeader->TcpSrcPort =htons(m_iSrcP);
                        pTcpHeader->TcpDestPort=htons(m_iDestP);
                        pTcpHeader->TcpSeqNum=htonl(m_iSeqN);
                        pTcpHeader->TcpAckNum=htonl(m_iAckN);
                        pTcpHeader->TcpHeadChkFlgs = htons((m_iTcpHL<<12)+(m_iReserv<<6)
                                        +(m_iUrg<<5)+(m_iAck<<4)+(m_iPsh<<3)+(m_iRst<<2)
                                        +(m_iSyn<<1)+(m_iFin));
                        pTcpHeader->TcpWndSize=htons(m_iWndSize);
                        pTcpHeader->TcpChksum=0;
                        pTcpHeader->TcpUrgPoint=htons(m_iUrgP);
                        pTcpHeader->TcpOptions=htons(m_iOpt);
                        pTcpHeader->TcpData=htonl(m_iDataOpt);


                        pTcpHeader->TcpChksum=(Chksum((LPWORD)pTcpHeader,TcpHeaderLength));

                        GetDlgItem(IDC_BCHK)->EnableWindow(FALSE);
```

```
                        UpdateData(FALSE);


        }

        lpColgenTCPDlg->Enviar();

        //CDialog::OnOK();//como comentario no cierra el dialogo
                                //pero no se si manda paquetes.
                                //manda con boton CANCELAR


void CTCPHeader::OnBchk()
{
        // TODO: Add your control notification handler code here

        CColgenTCPDlg* lpColgenTCPDlg = (CColgenTCPDlg*)m_pWnd;
        LPTcpHeader pTcpHeader;
        pTcpHeader =(LPTcpHeader)lpColgenTCPDlg->pTcpHeader;

                        UpdateData(TRUE);
                        pTcpHeader->TcpSrcPort =htons(m_iSrcP);
                        pTcpHeader->TcpDestPort=htons(m_iDestP);
                        pTcpHeader->TcpSeqNum=htonl(m_iSeqN);
                        pTcpHeader->TcpAckNum=htonl(m_iAckN);
                        pTcpHeader->TcpHeadChkFlgs = htons((m_iTcpHl<<12)+(m_iReserv<<6)
                                +(m_iUrg<<5)+(m_iAck<<4)+(m_iPsh<<3)+(m_iRst<<2)
                                +(m_iSyn<<1)+(m_iFin));
                        pTcpHeader->TcpWndSize=htons(m_iWndSize);
                        pTcpHeader->TcpChksum=0;
                        pTcpHeader->TcpUrgPoint=htons(m_iUrgP);
                        pTcpHeader->TcpOptions=htons(m_iOpt);
                        pTcpHeader->TcpData=htonl(m_iDataOpt);


                        pTcpHeader->TcpChksum=(Chksum((LPWORD)pTcpHeader,TcpHeaderLength ));
                        m_iChkSum = ntohs(pTcpHeader->TcpChksum);
                        UpdateData(FALSE);


}

WORD CTCPHeader::Chksum(LPWORD lpwTcpData, WORD wDataLength)

        //////////////////////
        //AQUI INICIA MI CODIGO
        //////////////////////
        long    lSum;                           // Almacena la sumatoria
        WORD    wOddByte;                       // Byte sobre la izquierda de la sumatoria
        WORD    wAnswer;                        // Suma de comprobación con complemento a 1's

        lSum = 0L;


        while (wDataLength > 1)
                {
                        lSum += *lpwTcpData++ ;
                        wDataLength -= 2;
                }

                // Maneja el byte impar si es necesario y asegura que la mitad superior sea cero
                if (wDataLength == 1)
                {
                        wOddByte   0;
                        *((LPBYTE) &wOddByte) = *(LPBYTE)lpwTcpData; // Un byte solamente
                        lSum += wOddByte;
                }

                // Agrega los extras de los 16 bits a los 16 bits inferiores
                lSum = (lSum >  16) + (lSum & 0xffff);
                                // Agrega los 16 bits superiores a los 16 bits inferiores
                lSum += (lSum >> 16);  // Agrega el extra
                wAnswer = (WORD)~lSum; // complemento a 1's, entonces trunca  a 16 bits


                return(wAnswer);

                //////////////////////
                //AQUI TERMINA MI CODIGO
                //////////////////////
}
```

```
// IPHeader.cpp : implementation file
//

#include "stdafx.h"
#include "colgenTCP.h"
#include "IPHeader.h"
#include "ColgenTCPDlg.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CIPHeader dialog


CIPHeader::CIPHeader(CWnd* pParent /*=NULL*/)
	: CDialog(CIPHeader::IDD, pParent)
{
	//{{AFX_DATA_INIT(CIPHeader)
	m_iChkSumIP = 0;
	m_iDestIP = 0;
	m_iFlagsIP = 0;
	m_iFragOffIP = 0;
	m_iHLenIP = 0;
	m_iLongFullIP = 0;
	m_iProtIP = 0;
	m_iSrcIP = 0;
	m_iTosIP = 0;
	m_iTtlIP = 0;
	m_iVerIP = 0;
	m_iIdIP = 0;
	m_isrcip1 = 0;
	m_isrcip2 = 0;
	m_isrcip3 = 0;
	m_isrcip4 = 0;
	m_idestip1 = 0;
	m_idestip2 = 0;
	m_idestip3 = 0;
	m_idestip4 = 0;
	//}}AFX_DATA_INIT
}


void CIPHeader::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CIPHeader)
	DDX_Text(pDX, IDC_CHKSUMIP, m_iChkSumIP);
	DDV_MinMaxInt(pDX, m_iChkSumIP, 0, 65535);
	DDX_Text(pDX, IDC_DESTIP, m_iDestIP);
	DDX_Text(pDX, IDC_FLAGSIP, m_iFlagsIP);
	DDX_Text(pDX, IDC_FRAGOFFIP, m_iFragOffIP);
	DDX_Text(pDX, IDC_HLENIP, m_iHLenIP);
	DDX_Text(pDX, IDC_LONGFULLIP, m_iLongFullIP);
	DDX_Text(pDX, IDC_PROTIP, m_iProtIP);
	DDX_Text(pDX, IDC_SRCIP, m_iSrcIP);
	DDX_Text(pDX, IDC_TOSIP, m_iTosIP);
	DDX_Text(pDX, IDC_TTLIP, m_iTtlIP);
	DDX_Text(pDX, IDC_VERIP, m_iVerIP);
	DDX_Text(pDX, IDC_ID, m_iIdIP);
	DDX_Text(pDX, IDC_SRCIP1, m_isrcip1);
	DDX_Text(pDX, IDC_SRCIP2, m_isrcip2);
	DDX_Text(pDX, IDC_SRCIP3, m_isrcip3);
	DDX_Text(pDX, IDC_SRCIP4, m_isrcip4);
	DDX_Text(pDX, IDC_DESTIP1, m_idestip1);
	DDX_Text(pDX, IDC_DESTIP2, m_idestip2);
	DDX_Text(pDX, IDC_DESTIP3, m_idestip3);
	DDX_Text(pDX, IDC_DESTIP4, m_idestip4);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CIPHeader, CDialog)
	//{{AFX_MSG_MAP(CIPHeader)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
/////////////////////////////////////////////////////////////////////////////
// CIPHeader message handlers

BOOL CIPHeader::OnInitDialog()
{
	CDialog::OnInitDialog();

	// TODO: Add extra initialization here

	CColgenTCPDlg* lpColgenTCPDlg = (CColgenTCPDlg*)m_pWnd;
	LPIpHeader pIpHeader;

	pIpHeader = (IpHeader *)(lpColgenTCPDlg->pIpHeader);

			m_iVerIP = ((pIpHeader->ip_verlen & 0xf0)>>4);
			m_iHLenIP = (pIpHeader->ip_verlen & 0x0f);
			m_iTosIP = pIpHeader->ip_tos;
			m_iLongFullIP = pIpHeader->ip_len;
			m_iIdIP =pIpHeader->ip_id;

			m_iFlagsIP = ((pIpHeader->ip_fragoff & 0xE000)>>13);
			m_iFragOffIP = (pIpHeader->ip_fragoff & 0x1FFF);
			m_iTtlIP =pIpHeader->ip_ttl;
			m_iProtIP =pIpHeader->ip_proto;
			m_iChkSumIP =pIpHeader->ip_chksum;
			m_iSrcIP =pIpHeader->ip_src_addr.S_un.S_addr;
			m_isrcip1=(m_iSrcIP&(0x000000ff));
			m_isrcip2=((m_iSrcIP&(0x0000ff00))>>8);
			m_isrcip3=((m_iSrcIP&(0x00ff0000))>>16);
			m_isrcip4=((m_iSrcIP&(0xff000000))>>24);
			m_iDestIP =pIpHeader->ip_dst_addr.S_un.S_addr;
			m_idestip1=(m_iDestIP&(0x000000ff));
			m_idestip2=((m_iDestIP&(0x0000ff00))>>8);
			m_idestip3=((m_iDestIP&(0x00ff0000))>>16);
			m_idestip4=((m_iDestIP&(0xff000000))>>24);
			UpdateData(FALSE);


			GetDlgItem(IDC_VERIP)->EnableWindow(FALSE);
			GetDlgItem(IDC_CHKSUMIP)->EnableWindow(FALSE);
			GetDlgItem(IDC_DESTIP1)->EnableWindow(FALSE);
			GetDlgItem(IDC_DESTIP2)->EnableWindow(FALSE);
			GetDlgItem(IDC_DESTIP3)->EnableWindow(FALSE);
			GetDlgItem(IDC_DESTIP4)->EnableWindow(FALSE);
			GetDlgItem(IDC_FLAGSIP)->EnableWindow(FALSE);
			GetDlgItem(IDC_FRAGOFFIP)->EnableWindow(FALSE);
			GetDlgItem(IDC_HLENIP)->EnableWindow(FALSE);
			GetDlgItem(IDC_IL)->EnableWindow(FALSE);
			GetDlgItem(IDC_LONGFULLIP)->EnableWindow(FALSE);
			GetDlgItem(IDC_SRCIP1)->EnableWindow(FALSE);
			GetDlgItem(IDC_SRCIP2)->EnableWindow(FALSE);
			GetDlgItem(IDC_SRCIP3)->EnableWindow(FALSE);
			GetDlgItem(IDC_SRCIP4)->EnableWindow(FALSE);
			GetDlgItem(IDC_TOSIP)->EnableWindow(FALSE);
			GetDlgItem(IDC_PROTIP)->EnableWindow(FALSE);
			GetDlgItem(IDC_TTLIP)->EnableWindow(FALSE);

	return TRUE;  // return TRUE unless you set the focus to a control
			// EXCEPTION: OCX Property Pages should return FALSE
}
void CIPHeader::SetParent(CDialog* pWnd)
{
//Establece el apuntador de la variable miembro
	m_pWnd = pWnd;
}
```

```
// colgenTCPDlg.h : header file
//

#if !defined(AFX_COLGENTCPDLG_H__D6900186_8F38_11D5_875C_000102BF481C__INCLUDED_)
#define AFX_COLGENTCPDLG_H__D6900186_8F38_11D5_875C_000102BF481C__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include <packet32.h>
#include "AdapterList.h"      // Added by ClassView

#define         ENVIO                   0
#define         RECEPCION               1

////////////////////////////////////////////////////////////////////
//
                                //
//                                              IP Header
                        //
//                     Implementation of RFC791 IP Header                    //
//
                                //
////////////////////////////////////////////////////////////////////

typedef struct _IpHeader
{
        BYTE ip_verlen;                          // Longitud de la versión y
encabezado
        BYTE ip_tos;                        // Tipo de servicio
        WORD ip_len;                        // Longitud total del paquete
        WORD ip_id;                              // Identificación del
datagrama
        WORD ip_fragoff;                    // Offset del fragmento
        BYTE ip_ttl;                        // Tiempo de vida
        BYTE ip_proto;                      // Protocolo
        WORD ip_chksum;                          // Suma de comprobación

        IN_ADDR ip_src_addr;           // Dirección fuente
        IN_ADDR ip_dst_addr;           // Dirección destino
//      BYTE ip_data[1];                    // Longitud del área de datos

} IpHeader;

typedef IpHeader FAR * LPIpHeader;

#define IpHeaderLength sizeof(IpHeader)

////////////////////////////////////////////////////////////////////
//
                                //
//                                       TCP header
                        //
//                     Implementation of RFC793 TCP Header                    //
//
                                //
////////////////////////////////////////////////////////////////////

typedef struct _TcpHeader
{
  WORD             TcpSrcPort;
  WORD             TcpDestPort;
  DWORD            TcpSeqNum;            // Type sub code
  DWORD            TcpAckNum;
  WORD             TcpHeadChkFlgs;
  WORD             TcpWndSize;
  WORD             TcpChksum;
  WORD             TcpUrgPoint;
  WORD             TcpOptions;// Not standard field in header, but reserved nonetheless
  DWORD            TcpData;// Not standard field in header, but reserved nonetheless
} TcpHeader;

typedef TcpHeader FAR * LPTcpHeader;
#define TcpHeaderLength sizeof(TcpHeader)
//////////*//////////////////////////////////////////////////////*****************



//////////////////////////////////////////////////////////////////////////////
```

```
// CColgenTCPDlg dialog

class CColgenTCPDlg : public CDialog
{
// Construction
public:
        void Enviar();
        WORD Chksum(LPWORD lpwTcpData, WORD wDataLength);
        int verpr;
        int verpe;
        int verptcp;
        ////////**************
        //              unsigned char buffer[256000];  // buffer to hold the data coming from the driver
        t
        char        AdapterList[10][1024];
        LPIpHeader      pPacket[6];


        int i,k;
        LPADAPTER lpAdapter;

        DWORD      dwErrorCode;
        DWORD dwVersion;
        DWORD dwWindowsMajorVersion;
        //unicode strings (winnt)
        WCHAR           AdapterName[512]; // string that contains a list of the network adapters
        WCHAR           *temp,*templ;
        //ascii strings (win95)
        char            AdapterNamea[512]; // string that contains a list of the network adapters
        char            *tempa,*templa;
        int                     AdapterNum,Open;
        ULONG           AdapterLength;

        int                                     mode,m;

        LPSTR                           lpNomHost;
        LPHOSTENT                       hostdest;       //Estructura de información del anfitrión en
internet
        SOCKET                          TcpSocket;              //Manejador de socket (descriptor)
        SOCKADDR_IN                     TcpSockAddr;    //Estructura de dirección del socket de
envio
        SOCKADDR_IN                     rcvSockAddr;
        BYTE                            TcpSendPacket[100];
        LPTcpHeader                     pTcpHeader,pTcpHeaderv;


        char        AdapterList2[1024];
        struct bpf_stat stat;

        // obtain the name of the adapters installed on this machine
                LPPACKET   lpPacket;
                ULONG   il, j, ulLines, ulen, ulBytesReceived;
        char    *pChar, *pLine, *base;
        char    *buf;
        u_int off;
        u_int tlen,tlenl;
        struct bpf_hdr *hdr;
        LPIpHeader pIpHeader,pIpHeaderv;



        CString adapternamelist;
        DWORD m_Adaptern;
void PrintPackets(LPPACKET lpPacket);
void SetAdaptern(DWORD Adaptern);
unsigned long GetIPAddress(LPSTR lpstrAddr);


        /////////*******************

        CColgenTCPDlg(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
        //{{AFX_DATA(CColgenTCPDlg)
        enum { IDD = IDD_COLGENTCP_DIALOG };
        CListBox        m_ctlListData;
        CButton m_ctlEject;
        int             m_iSend;
        CString m_NomHost;
        //}}AFX_DATA
```

```
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CColgenTCPDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);     // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        HICON m_hIcon;
                unsigned char buffer[256000];
                        // buffer to hold the data coming from the driver


        // Generated message map functions
        //{{AFX_MSG(CColgenTCPDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnDestroy();
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        afx_msg void OnEject();
        afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
        afx_msg void OnRsend();
        afx_msg void OnReceive();
        afx_msg void OnVerPaqueteenviado();
        afx_msg void OnVerCabeceradepaquetetcprecibido1();
        afx_msg void OnVerCabeceradepaquetetcprecibido2();
        afx_msg void OnVerCabeceradepaquetetcprecibido3();
        afx_msg void OnVerCabeceradepaquetetcprecibido4();
        afx_msg void OnVerCabeceradepaquetetcprecibido5();
        afx_msg void OnVerCabeceradepaquetetcprecibido6();
        afx_msg void OnVerCabeceraip();
        afx_msg void OnVerVercabeceradepaqueteip();
        afx_msg void OnVerCabeceradepaqueteenviado();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
private:


//      CAdapterList m_sSetPointer;
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_COLGENTCPDLG_H__D6900186_8F38_11D5_875C_000102BF481C__INCLUDED_)
```

```
// colgenTCP.h : main header file for the COLGENTCP application
//

#if !defined(AFX_COLGENTCP_H__D6900184_8F38_11D5_875C_000102BF481C__INCLUDED_)
#define AFX_COLGENTCP_H__D6900184_8F38_11D5_875C_000102BF481C__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CColgenTCPApp:
// See colgenTCP.cpp for the implementation of this class
//

class CColgenTCPApp : public CWinApp
{
public:
	CColgenTCPApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CColgenTCPApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CColgenTCPApp)
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_COLGENTCP_H__D6900184_8F38_11D5_875C_000102BF481C__INCLUDED_)
```

```
#if !defined(AFX_ADAPTERLIST_H__D6900192_8F38_11D5_875C_000102BF481C__INCLUDED_)
#define AFX_ADAPTERLIST_H__D6900192_8F38_11D5_875C_000102BF481C__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// AdapterList.h : header file
//
#include "ColgenTCPDlg.h"
/////////////////////////////////////////////////////////////////////////////
// CAdapterList dialog

class CAdapterList : public CDialog
{
// Construction
public:
                CDialog* m_pWnd;
        void SetParent(CDialog *pWnd);
        int nIndex;
                DWORD color;
        CAdapterList(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CAdapterList)
        enum { IDD = IDD_DIALOG1 };
        CListBox        m_ctlListAdapters;
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAdapterList)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CAdapterList)
        virtual BOOL OnInitDialog();
        virtual void OnOK();
        virtual void OnCancel();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
private:
//      CDialog* m_pWnd;
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_ADAPTERLIST_H__D6900192_8F38_11D5_875C_000102BF481C__INCLUDED_)
```

```
#if !defined(AFX_TCPHEADER_H__08223A40_AA89_11D5_866D_00D009A28426__INCLUDED_)
#define AFX_TCPHEADER_H__08223A40_AA89_11D5_866D_00D009A28426__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// TCPHeader.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CTCPHeader dialog

class CTCPHeader : public CDialog
{
// Construction
public:
        WORD Chksum(LPWORD lpwTcpData, WORD wDataLength);

        void SetParent(CDialog* pWnd);
        CTCPHeader(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CTCPHeader)
        enum { IDD = IDD_TCPHEADER };
        int             m_iSrcP;
        int             m_iDestP;
        int             m_iAck;
        int             m_iBFill;
        int             m_iChkSum;
        int             m_iFin;
        int             m_iOpt;
        int             m_iPsh;
        int             m_iRst;
        int             m_iSyn;
        int             m_iUrg;
        int             m_iWndSize;
        int             m_iUrgP;
        DWORD   m_iAckN;
        DWORD   m_iSeqN;
        DWORD   m_iDataOpt;
        int             m_iTcpHl;
        int             m_iReserv;
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CTCPHeader)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CTCPHeader)
        virtual BOOL OnInitDialog();
        virtual void OnOK();
        afx_msg void OnBchk();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
private:
        CDialog* m_pWnd;

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_TCPHEADER_H__08223A40_AA89_11D5_866D_00D009A28426__INCLUDED_)
```

```
#if !defined(AFX_IPHEADER_H__08223A41_AA89_11D5_866D_00D009A28426__INCLUDED_)
#define AFX_IPHEADER_H__08223A41_AA89_11D5_866D_00D009A28426__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// IPHeader.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CIPHeader dialog

class CIPHeader : public CDialog
{
// Construction
public:
	CDialog* m_pWnd;
	void SetParent(CDialog *pWnd);
	CIPHeader(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CIPHeader)
	enum { IDD = IDD_IPHEADER };
	int		m_iChkSumIP;
	int		m_iDestIP;
	int		m_iFlagsIP;
	int		m_iFragOffIP;
	int		m_iHLenIP;
	int		m_iLongFullIP;
	int		m_iProtIP;
	int		m_iSrcIP;
	int		m_iTosIP;
	int		m_iTtlIP;
	int		m_iVerIP;
	int		m_iIdIP;
	int		m_isrcip1;
	int		m_isrcip2;
	int		m_isrcip3;
	int		m_isrcip4;
	int		m_idestip1;
	int		m_idestip2;
	int		m_idestip3;
	int		m_idestip4;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CIPHeader)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CIPHeader)
	virtual BOOL OnInitDialog();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_IPHEADER_H__08223A41_AA89_11D5_866D_00D009A28426__INCLUDED_)
```

```
/*++

Copyright (c) 1997-1998  Microsoft Corporation

Module Name:

    basetsd.h

Abstract:

    Type definitions for the basic sized types.

Author:

    Jeff Havens (jhavens)   23-Oct-1997

Revision History:

--*/

#ifndef _BASETSD_H_
#define _BASETSD_H_

#ifdef __cplusplus
extern "C" {
#endif

//
// The following types are guaranteed to be signed and 32 bits wide.
//

typedef int LONG32, *PLONG32;
typedef int INT32, *PINT32;

//
// The following types are guaranteed to be unsigned and 32 bits wide.
//

typedef unsigned int ULONG32, *PULONG32;
typedef unsigned int DWORD32, *PDWORD32;
typedef unsigned int UINT32, *PUINT32;

//
// The INT_PTR is guaranteed to be the same size as a pointer.  Its
// size with change with pointer size (32/64).  It should be used
// anywhere that a pointer is cast to an integer type. UINT_PTR is
// the unsigned variation.
//
// HALF_PTR is half the size of a pointer it intended for use with
// within strcuture which contain a pointer and two small fields.
// UHALF_PTR is the unsigned variation.
//

#ifdef _WIN64

typedef __int64 INT_PTR, *PINT_PTR;
typedef unsigned __int64 UINT_PTR, *PUINT_PTR;

#define MAXINT_PTR (0x7fffffffffffffffI64)
#define MININT_PTR (0x8000000000000000I64)
#define MAXUINT_PTR (0xffffffffffffffffUI64)

typedef unsigned int UHALF_PTR, *PUHALF_PTR;
typedef int HALF_PTR, *PHALF_PTR;

#define MAXUHALF_PTR (0xffffffffUL)
#define MAXHALF_PTR (0x7fffffffL)
#define MINHALF_PTR (0x80000000L)

#pragma warning(disable:4311)   // type cast truncation

#if !defined(__midl)
__inline
unsigned long
HandleToUlong(
    void *h
    )
{
    return((unsigned long) h );
}
```

```
__inline
unsigned long
PtrToUlong(
    void  *p
    )
{
    return((unsigned long) p );
}

__inline
unsigned short
PtrToUshort(
    void  *p
    )
{
    return((unsigned short) p );
}

__inline
long
PtrToLong(
    void  *p
    )
{
    return((long) p );
}

__inline
short
PtrToShort(
    void  *p
    )
{
    return((short) p );
}
#endif
#pragma warning(3:4311)   // type cast truncation

#else


typedef long INT_PTR, *PINT_PTR;
typedef unsigned long UINT_PTR, *PUINT_PTR;

#define MAXINT_PTR (0x7fffffffL)
#define MININT_PTR (0x80000000L)
#define MAXUINT_PTR (0xffffffffUL)

typedef unsigned short UHALF_PTR, *PUHALF_PTR;
typedef short HALF_PTR, *PHALF_PTR;

#define MAXUHALF_PTR 0xffff
#define MAXHALF_PTR 0x7fff
#define MINHALF_PTR 0x8000

#define HandleToUlong( h ) ((ULONG) (h) )
#define PtrToUlong( p ) ((ULONG) (p) )
#define PtrToLong( p ) ((LONG) (p) )
#define PtrToUshort( p ) ((unsigned short) (p) )
#define PtrToShort( p ) ((short) (p) )

#endif

//
// SIZE_T used for counts or ranges which need to span the range of
// of a pointer.  SSIZE_T is the signed variation.
//

typedef UINT_PTR SIZE_T, *PSIZE_T;
typedef INT_PTR SSIZE_T, *PSSIZE_T;

//
// The following types are guaranteed to be signed and 64 bits wide.
//

typedef __int64 LONG64, *PLONG64;
typedef __int64 INT64, *PINT64;


//
```

```
// The following types are guaranteed to be unsigned and 64 bits wide.
//

typedef unsigned __int64 ULONG64, *PULONG64;
typedef unsigned __int64 DWORD64, *PDWORD64;
typedef unsigned __int64 UINT64, *PUINT64;

#ifdef __cplusplus
}
#endif

#endif // _BASETSD_H_
```

```
/*++ BUILD Version: 0004    // Increment this if a change has global effects
   Copyright (c) 1990-1993  Microsoft Corporation
   Module Name:
   devioctl.h
   Revision History:
   -- */
// begin_winioctl
#ifndef _DEVIOCTL_
#define _DEVIOCTL_
// begin_ntddk begin_nthal begin_ntifs
//
// Define the various device type values.  Note that values used by Microsoft
// Corporation are in the range 0-32767, and 32768-65535 are reserved for use
// by customers.
//
#define DEVICE_TYPE ULONG
#define FILE_DEVICE_BEEP                0x00000001
#define FILE_DEVICE_CD_ROM              0x00000002
#define FILE_DEVICE_CD_ROM_FILE_SYSTEM  0x00000003
#define FILE_DEVICE_CONTROLLER          0x00000004
#define FILE_DEVICE_DATALINK            0x00000005
#define FILE_DEVICE_DFS                 0x00000006
#define FILE_DEVICE_DISK                0x00000007
#define FILE_DEVICE_DISK_FILE_SYSTEM    0x00000008
#define FILE_DEVICE_FILE_SYSTEM         0x00000009
#define FILE_DEVICE_INPORT_PORT         0x0000000a
#define FILE_DEVICE_KEYBOARD            0x0000000b
#define FILE_DEVICE_MAILSLOT            0x0000000c
#define FILE_DEVICE_MIDI_IN             0x0000000d
#define FILE_DEVICE_MIDI_OUT            0x0000000e
#define FILE_DEVICE_MOUSE               0x0000000f
#define FILE_DEVICE_MULTI_UNC_PROVIDER  0x00000010
#define FILE_DEVICE_NAMED_PIPE          0x00000011
#define FILE_DEVICE_NETWORK             0x00000012
#define FILE_DEVICE_NETWORK_BROWSER     0x00000013
#define FILE_DEVICE_NETWORK_FILE_SYSTEM 0x00000014
#define FILE_DEVICE_NULL                0x00000015
#define FILE_DEVICE_PARALLEL_PORT       0x00000016
#define FILE_DEVICE_PHYSICAL_NETCARD    0x00000017
#define FILE_DEVICE_PRINTER             0x00000018
#define FILE_DEVICE_SCANNER             0x00000019
#define FILE_DEVICE_SERIAL_MOUSE_PORT   0x0000001a
#define FILE_DEVICE_SERIAL_PORT         0x0000001b
#define FILE_DEVICE_SCREEN              0x0000001c
#define FILE_DEVICE_SOUND               0x0000001d
#define FILE_DEVICE_STREAMS             0x0000001e
#define FILE_DEVICE_TAPE                0x0000001f
#define FILE_DEVICE_TAPE_FILE_SYSTEM    0x00000020
#define FILE_DEVICE_TRANSPORT           0x00000021
#define FILE_DEVICE_UNKNOWN             0x00000022
#define FILE_DEVICE_VIDEO               0x00000023
#define FILE_DEVICE_VIRTUAL_DISK        0x00000024
#define FILE_DEVICE_WAVE_IN             0x00000025
#define FILE_DEVICE_WAVE_OUT            0x00000026
#define FILE_DEVICE_8042_PORT           0x00000027
#define FILE_DEVICE_NETWORK_REDIRECTOR  0x00000028
#define FILE_DEVICE_BATTERY             0x00000029
#define FILE_DEVICE_BUS_EXTENDER        0x0000002a
#define FILE_DEVICE_MODEM               0x0000002b
#define FILE_DEVICE_VDM                 0x0000002c
#define FILE_DEVICE_MASS_STORAGE        0x0000002d
//
// Macro definition for defining IOCTL and FSCTL function control codes.  Note
// that function codes 0-2047 are reserved for Microsoft Corporation, and
// 2048-4095 are reserved for customers.
//
#define CTL_CODE( DeviceType, Function, Method, Access ) (                 \
    ((DeviceType) << 16) | ((Access) << 14) | ((Function) << 2) | (Method) \
)
//
// Define the method codes for how buffers are passed for I/O and FS controls
//
#define METHOD_BUFFERED                 0
#define METHOD_IN_DIRECT                1
#define METHOD_OUT_DIRECT               2
#define METHOD_NEITHER                  3
//
// Define the access check value for any access
//
//
```

```
// The FILE_READ_ACCESS and FILE_WRITE_ACCESS constants are also defined in
// ntioapi.h as FILE_READ_DATA and FILE_WRITE_DATA. The values for these
// constants *MUST* always be in sync.
//
#define FILE_ANY_ACCESS                 0
#define FILE_READ_ACCESS          ( 0x0001 ) // file & pipe
#define FILE_WRITE_ACCESS         ( 0x0002 ) // file & pipe
// end_ntddk end_nthal end_ntifs
#endif                          // _DEVIOCTL_
// end_winioctl
```

```
/*++ BUILD Version: 0001              // Increment this if a change has global effects
   Copyright (c) 1990-1993  Microsoft Corporation
   Module Name:
   ntddndis.h
   Abstract:
   This is the include file that defines all constants and types for
   accessing the Network driver interface device.
   Author:
   Steve Wood (stevewo) 27-May-1990
   Revision History:
   Adam Barr (adamba)          04-Nov-1992          added the correct values for NDIS 3.0.
   Jameel Hyder (jameelh)      01-Aug-95            added Pnp IoCTLs and structures
   Kyle Brandon (kyleb) 09/24/96              added general co ndis oids.
   -- */
#ifndef _NTDDNDIS_
#define _NTDDNDIS_
//
// Device Name - this string is the name of the device.  It is the name
// that should be passed to NtOpenFile when accessing the device.
//
// Note:  For devices that support multiple units, it should be suffixed
//                with the Ascii representation of the unit number.
//
#define DD_NDIS_DEVICE_NAME "\\Device\\UNKNOWN"
//
// NtDeviceIoControlFile IoControlCode values for this device.
//
// Warning:  Remember that the low two bits of the code specify how the
//                      buffers are passed to the driver!
//
#define _NDIS_CONTROL_CODE(request,method) \
                    CTL_CODE(FILE_DEVICE_PHYSICAL_NETCARD, request, method, FILE_ANY_ACCESS)
#define IOCTL_NDIS_QUERY_GLOBAL_STATS _NDIS_CONTROL_CODE( 0, METHOD_OUT_DIRECT )
#define IOCTL_NDIS_QUERY_ALL_STATS          _NDIS_CONTROL_CODE( 1, METHOD_OUT_DIRECT )
#define IOCTL_NDIS_ADD_DEVICE               _NDIS_CONTROL_CODE( 2, METHOD_BUFFERED )
#define IOCTL_NDIS_DELETE_DEVICE            _NDIS_CONTROL_CODE( 3, METHOD_BUFFERED )
#define IOCTL_NDIS_TRANSLATE_NAME           _NDIS_CONTROL_CODE( 4, METHOD_BUFFERED )
#define IOCTL_NDIS_ADD_TDI_DEVICE           _NDIS_CONTROL_CODE( 5, METHOD_BUFFERED )
#define IOCTL_NDIS_NOTIFY_PROTOCOL          _NDIS_CONTROL_CODE( 6, METHOD_BUFFERED )
#define IOCTL_NDIS_GET_LOG_DATA                     _NDIS_CONTROL_CODE( 7, METHOD_OUT_DIRECT )
//
// NtDeviceIoControlFile InputBuffer/OutputBuffer record structures for
// this device.
//
//
// This is the type of an NDIS OID value.
//
typedef ULONG NDIS_OID, *PNDIS_OID;
//
// IOCTL_NDIS_QUERY_ALL_STATS returns a sequence of these, packed
// together (no padding is required since statistics all have
// four or eight bytes of data).
//
typedef struct _NDIS_STATISTICS_VALUE {
    NDIS_OID Oid;
    ULONG DataLength;
    UCHAR Data[1];              // variable length

} NDIS_STATISTICS_VALUE, *PNDIS_STATISTICS_VALUE;

//
// Structure used by TRANSLATE_NAME IOCTL
//
typedef struct _NET_PNP_ID {
    ULONG ClassId;
    ULONG Token;
} NET_PNP_ID, *PNET_PNP_ID;

typedef struct _NET_PNP_TRANSLATE_LIST {
    ULONG BytesNeeded;
    NET_PNP_ID IdArray[ANYSIZE_ARRAY];
} NET_PNP_TRANSLATE_LIST, *PNET_PNP_TRANSLATE_LIST;

//
// Structure used to define a self-contained variable data structure
//
typedef struct _NDIS_VAR_DATA_DESC {
    USHORT Length;              // # of octects of data

    USHORT MaximumLength;       // # of octects available
```

```
    LONG Offset;                // Offset of data relative to the descriptor
} NDIS_VAR_DATA_DESC, *PNDIS_VAR_DATA_DESC;

//
// Object Identifiers used by NdisRequest Query/Set Information
//
//
// General Objects
//
#define OID_GEN_SUPPORTED_LIST                  0x00010101
#define OID_GEN_HARDWARE_STATUS                 0x00010102
#define OID_GEN_MEDIA_SUPPORTED                 0x00010103
#define OID_GEN_MEDIA_IN_USE                    0x00010104
#define OID_GEN_MAXIMUM_LOOKAHEAD               0x00010105
#define OID_GEN_MAXIMUM_FRAME_SIZE              0x00010106
#define OID_GEN_LINK_SPEED                      0x00010107
#define OID_GEN_TRANSMIT_BUFFER_SPACE           0x00010108
#define OID_GEN_RECEIVE_BUFFER_SPACE            0x00010109
#define OID_GEN_TRANSMIT_BLOCK_SIZE             0x0001010A
#define OID_GEN_RECEIVE_BLOCK_SIZE              0x0001010B
#define OID_GEN_VENDOR_ID                       0x0001010C
#define OID_GEN_VENDOR_DESCRIPTION              0x0001010D
#define OID_GEN_CURRENT_PACKET_FILTER           0x0001010E
#define OID_GEN_CURRENT_LOOKAHEAD               0x0001010F
#define OID_GEN_DRIVER_VERSION                  0x00010110
#define OID_GEN_MAXIMUM_TOTAL_SIZE              0x00010111
#define OID_GEN_PROTOCOL_OPTIONS                0x00010112
#define OID_GEN_MAC_OPTIONS                     0x00010113
#define OID_GEN_MEDIA_CONNECT_STATUS            0x00010114
#define OID_GEN_MAXIMUM_SEND_PACKETS            0x00010115
#define OID_GEN_VENDOR_DRIVER_VERSION           0x00010116
#define OID_GEN_XMIT_OK                         0x00020101
#define OID_GEN_RCV_OK                          0x00020102
#define OID_GEN_XMIT_ERROR                      0x00020103
#define OID_GEN_RCV_ERROR                       0x00020104
#define OID_GEN_RCV_NO_BUFFER                   0x00020105
#define OID_GEN_DIRECTED_BYTES_XMIT             0x00020201
#define OID_GEN_DIRECTED_FRAMES_XMIT            0x00020202
#define OID_GEN_MULTICAST_BYTES_XMIT            0x00020203
#define OID_GEN_MULTICAST_FRAMES_XMIT           0x00020204
#define OID_GEN_BROADCAST_BYTES_XMIT            0x00020205
#define OID_GEN_BROADCAST_FRAMES_XMIT           0x00020206
#define OID_GEN_DIRECTED_BYTES_RCV              0x00020207
#define OID_GEN_DIRECTED_FRAMES_RCV             0x00020208
#define OID_GEN_MULTICAST_BYTES_RCV             0x00020209
#define OID_GEN_MULTICAST_FRAMES_RCV            0x0002020A
#define OID_GEN_BROADCAST_BYTES_RCV             0x0002020B
#define OID_GEN_BROADCAST_FRAMES_RCV            0x0002020C
#define OID_GEN_RCV_CRC_ERROR                   0x0002020D
#define OID_GEN_TRANSMIT_QUEUE_LENGTH           0x0002020E
#define OID_GEN_GET_TIME_CAPS                   0x0002020F
#define OID_GEN_GET_NETCARD_TIME                0x00020210
//
//      These are connection-oriented general OIDs.
//      These replace the above OIDs for connection-oriented media.
//
#define OID_GEN_CO_SUPPORTED_LIST               0x00010101
#define OID_GEN_CO_HARDWARE_STATUS              0x00010102
#define OID_GEN_CO_MEDIA_SUPPORTED              0x00010103
#define OID_GEN_CO_MEDIA_IN_USE                 0x00010104
#define OID_GEN_CO_LINK_SPEED                   0x00010105
#define OID_GEN_CO_VENDOR_ID                    0x00010106
#define OID_GEN_CO_VENDOR_DESCRIPTION           0x00010107
#define OID_GEN_CO_DRIVER_VERSION               0x00010108
#define OID_GEN_CO_PROTOCOL_OPTIONS             0x00010109
#define OID_GEN_CO_MAC_OPTIONS                  0x0001010A
#define OID_GEN_CO_MEDIA_CONNECT_STATUS         0x0001010B
#define OID_GEN_CO_VENDOR_DRIVER_VERSION        0x0001010C
#define OID_GEN_CO_MINIMUM_LINK_SPEED           0x0001010D
#define OID_GEN_CO_GET_TIME_CAPS                0x00010201
#define OID_GEN_CO_GET_NETCARD_TIME             0x00010202
//
//      These are connection-oriented statistics OIDs.
//
#define OID_GEN_CO_XMIT_PDUS_OK                 0x00020101
#define OID_GEN_CO_RCV_PDUS_OK                  0x00020102
#define OID_GEN_CO_XMIT_PDUS_ERROR              0x00020103
#define OID_GEN_CO_RCV_PDUS_ERROR               0x00020104
#define OID_GEN_CO_RCV_PDUS_NO_BUFFER           0x00020105
```

```
#define OID_GEN_CO_RCV_CRC_ERROR                0x00020201
#define OID_GEN_CO_TRANSMIT_QUEUE_LENGTH        0x00020202
#define OID_GEN_CO_BYTES_XMIT                   0x00020203
#define OID_GEN_CO_BYTES_RCV                    0x00020204
#define OID_GEN_CO_BYTES_XMIT_OUTSTANDING       0x00020205
#define OID_GEN_CO_NETCARD_LOAD                 0x00020206
//
// These are objects for Connection-oriented media call-managers and are not
// valid for ndis drivers. Under construction.
//
#define OID_CO_ADD_PVC                          0xFF000001
#define OID_CO_DELETE_PVC                       0xFF000002
#define OID_CO_GET_CALL_INFORMATION             0xFF000003
#define OID_CO_ADD_ADDRESS                      0xFF000004
#define OID_CO_DELETE_ADDRESS                   0xFF000005
#define OID_CO_GET_ADDRESSES                    0xFF000006
#define OID_CO_ADDRESS_CHANGE                   0xFF000007
#define OID_CO_SIGNALING_ENABLED                0xFF000008
#define OID_CO_SIGNALING_DISABLED               0xFF000009
//
// 802.3 Objects (Ethernet)
//
#define OID_802_3_PERMANENT_ADDRESS             0x01010101
#define OID_802_3_CURRENT_ADDRESS               0x01010102
#define OID_802_3_MULTICAST_LIST                0x01010103
#define OID_802_3_MAXIMUM_LIST_SIZE             0x01010104
#define OID_802_3_MAC_OPTIONS                   0x01010105
//
//
#define NDIS_802_3_MAC_OPTION_PRIORITY          0x00000001
#define OID_802_3_RCV_ERROR_ALIGNMENT           0x01020101
#define OID_802_3_XMIT_ONE_COLLISION            0x01020102
#define OID_802_3_XMIT_MORE_COLLISIONS          0x01020103
#define OID_802_3_XMIT_DEFERRED                 0x01020201
#define OID_802_3_XMIT_MAX_COLLISIONS           0x01020202
#define OID_802_3_RCV_OVERRUN                   0x01020203
#define OID_802_3_XMIT_UNDERRUN                 0x01020204
#define OID_802_3_XMIT_HEARTBEAT_FAILURE        0x01020205
#define OID_802_3_XMIT_TIMES_CRS_LOST           0x01020206
#define OID_802_3_XMIT_LATE_COLLISIONS          0x01020207
//
// 802.5 Objects (Token-Ring)
//
#define OID_802_5_PERMANENT_ADDRESS             0x02010101
#define OID_802_5_CURRENT_ADDRESS               0x02010102
#define OID_802_5_CURRENT_FUNCTIONAL            0x02010103
#define OID_802_5_CURRENT_GROUP                 0x02010104
#define OID_802_5_LAST_OPEN_STATUS              0x02010105
#define OID_802_5_CURRENT_RING_STATUS           0x02010106
#define OID_802_5_CURRENT_RING_STATE            0x02010107
#define OID_802_5_LINE_ERRORS                   0x02020101
#define OID_802_5_LOST_FRAMES                   0x02020102
#define OID_802_5_BURST_ERRORS                  0x02020201
#define OID_802_5_AC_ERRORS                     0x02020202
#define OID_802_5_ABORT_DELIMETERS              0x02020203
#define OID_802_5_FRAME_COPIED_ERRORS           0x02020204
#define OID_802_5_FREQUENCY_ERRORS              0x02020205
#define OID_802_5_TOKEN_ERRORS                  0x02020206
#define OID_802_5_INTERNAL_ERRORS               0x02020207
//
// FDDI Objects
//
#define OID_FDDI_LONG_PERMANENT_ADDR            0x03010101
#define OID_FDDI_LONG_CURRENT_ADDR              0x03010102
#define OID_FDDI_LONG_MULTICAST_LIST            0x03010103
#define OID_FDDI_LONG_MAX_LIST_SIZE             0x03010104
#define OID_FDDI_SHORT_PERMANENT_ADDR           0x03010105
#define OID_FDDI_SHORT_CURRENT_ADDR             0x03010106
#define OID_FDDI_SHORT_MULTICAST_LIST           0x03010107
#define OID_FDDI_SHORT_MAX_LIST_SIZE            0x03010108
#define OID_FDDI_ATTACHMENT_TYPE                0x03020101
#define OID_FDDI_UPSTREAM_NODE_LONG             0x03020102
#define OID_FDDI_DOWNSTREAM_NODE_LONG           0x03020103
#define OID_FDDI_FRAME_ERRORS                   0x03020104
#define OID_FDDI_FRAMES_LOST                    0x03020105
#define OID_FDDI_RING_MGT_STATE                 0x03020106
#define OID_FDDI_LCT_FAILURES                   0x03020107
#define OID_FDDI_LEM_REJECTS                    0x03020108
#define OID_FDDI_LCONNECTION_STATE              0x03020109
#define OID_FDDI_SMT_STATION_ID                 0x03030201
#define OID_FDDI_SMT_OP_VERSION_ID              0x03030202
```

```
#define OID_FDDI_SMT_HI_VERSION_ID                  0x03030203
#define OID_FDDI_SMT_LO_VERSION_ID                  0x03030204
#define OID_FDDI_SMT_MANUFACTURER_DATA              0x03030205
#define OID_FDDI_SMT_USER_DATA                      0x03030206
#define OID_FDDI_SMT_MIB_VERSION_ID                 0x03030207
#define OID_FDDI_SMT_MAC_CT                         0x03030208
#define OID_FDDI_SMT_NON_MASTER_CT                  0x03030209
#define OID_FDDI_SMT_MASTER_CT                      0x0303020A
#define OID_FDDI_SMT_AVAILABLE_PATHS            0x0303020B
#define OID_FDDI_SMT_CONFIG_CAPABILITIES        0x0303020C
#define OID_FDDI_SMT_CONFIG_POLICY                  0x0303020D
#define OID_FDDI_SMT_CONNECTION_POLICY              0x0303020E
#define OID_FDDI_SMT_T_NOTIFY                       0x0303020F
#define OID_FDDI_SMT_STAT_RPT_POLICY            0x03030210
#define OID_FDDI_SMT_TRACE_MAX_EXPIRATION   0x03030211
#define OID_FDDI_SMT_PORT_INDEXES                   0x03030212
#define OID_FDDI_SMT_MAC_INDEXES                    0x03030213
#define OID_FDDI_SMT_BYPASS_PRESENT                 0x03030214
#define OID_FDDI_SMT_ECM_STATE                      0x03030215
#define OID_FDDI_SMT_CF_STATE                       0x03030216
#define OID_FDDI_SMT_HOLD_STATE                     0x03030217
#define OID_FDDI_SMT_REMOTE_DISCONNECT_FLAG 0x03030218
#define OID_FDDI_SMT_STATION_STATUS                 0x03030219
#define OID_FDDI_SMT_PEER_WRAP_FLAG                 0x0303021A
#define OID_FDDI_SMT_MSG_TIME_STAMP                 0x0303021B
#define OID_FDDI_SMT_TRANSITION_TIME_STAMP  0x0303021C
#define OID_FDDI_SMT_SET_COUNT                      0x0303021D
#define OID_FDDI_SMT_LAST_SET_STATION_ID        0x0303021E
#define OID_FDDI_MAC_FRAME_STATUS_FUNCTIONS 0x0303021F
#define OID_FDDI_MAC_BRIDGE_FUNCTIONS           0x03030220
#define OID_FDDI_MAC_T_MAX_CAPABILITY           0x03030221
#define OID_FDDI_MAC_TVX_CAPABILITY                 0x03030222
#define OID_FDDI_MAC_AVAILABLE_PATHS            0x03030223
#define OID_FDDI_MAC_CURRENT_PATH                   0x03030224
#define OID_FDDI_MAC_UPSTREAM_NBR                   0x03030225
#define OID_FDDI_MAC_DOWNSTREAM_NBR                 0x03030226
#define OID_FDDI_MAC_OLD_UPSTREAM_NBR           0x03030227
#define OID_FDDI_MAC_OLD_DOWNSTREAM_NBR             0x03030228
#define OID_FDDI_MAC_DUP_ADDRESS_TEST           0x03030229
#define OID_FDDI_MAC_REQUESTED_PATHS            0x0303022A
#define OID_FDDI_MAC_DOWNSTREAM_PORT_TYPE   0x0303022B
#define OID_FDDI_MAC_INDEX                          0x0303022C
#define OID_FDDI_MAC_SMT_ADDRESS                    0x0303022D
#define OID_FDDI_MAC_LONG_GRP_ADDRESS           0x0303022E
#define OID_FDDI_MAC_SHORT_GRP_ADDRESS              0x0303022F
#define OID_FDDI_MAC_T_REQ                          0x03030230
#define OID_FDDI_MAC_T_NEG                          0x03030231
#define OID_FDDI_MAC_T_MAX                          0x03030232
#define OID_FDDI_MAC_TVX_VALUE                      0x03030233
#define OID_FDDI_MAC_T_PRI0                         0x03030234
#define OID_FDDI_MAC_T_PRI1                         0x03030235
#define OID_FDDI_MAC_T_PRI2                         0x03030236
#define OID_FDDI_MAC_T_PRI3                         0x03030237
#define OID_FDDI_MAC_T_PRI4                         0x03030238
#define OID_FDDI_MAC_T_PRI5                         0x03030239
#define OID_FDDI_MAC_T_PRI6                         0x0303023A
#define OID_FDDI_MAC_FRAME_CT                       0x0303023B
#define OID_FDDI_MAC_COPIED_CT                      0x0303023C
#define OID_FDDI_MAC_TRANSMIT_CT                    0x0303023D
#define OID_FDDI_MAC_TOKEN_CT                       0x0303023E
#define OID_FDDI_MAC_ERROR_CT                       0x0303023F
#define OID_FDDI_MAC_LOST_CT                        0x03030240
#define OID_FDDI_MAC_TVX_EXPIRED_CT                 0x03030241
#define OID_FDDI_MAC_NOT_COPIED_CT                  0x03030242
#define OID_FDDI_MAC_LATE_CT                        0x03030243
#define OID_FDDI_MAC_RING_OP_CT                     0x03030244
#define OID_FDDI_MAC_FRAME_ERROR_THRESHOLD  0x03030245
#define OID_FDDI_MAC_FRAME_ERROR_RATIO              0x03030246
#define OID_FDDI_MAC_NOT_COPIED_THRESHOLD   0x03030247
#define OID_FDDI_MAC_NOT_COPIED_RATIO           0x03030248
#define OID_FDDI_MAC_RMT_STATE                      0x03030249
#define OID_FDDI_MAC_DA_FLAG                        0x0303024A
#define OID_FDDI_MAC_UNDA_FLAG                      0x0303024B
#define OID_FDDI_MAC_FRAME_ERROR_FLAG           0x0303024C
#define OID_FDDI_MAC_NOT_COPIED_FLAG            0x0303024D
#define OID_FDDI_MAC_MA_UNITDATA_AVAILABLE  0x0303024E
#define OID_FDDI_MAC_HARDWARE_PRESENT           0x0303024F
#define OID_FDDI_MAC_MA_UNITDATA_ENABLE             0x03030250
#define OID_FDDI_PATH_INDEX                         0x03030251
#define OID_FDDI_PATH_RING_LATENCY                  0x03030252
#define OID_FDDI_PATH_TRACE_STATUS                  0x03030253
```

```
#define OID_FDDI_PATH_SBA_PAYLOAD                       0x03030254
#define OID_FDDI_PATH_SBA_OVERHEAD                      0x03030255
#define OID_FDDI_PATH_CONFIGURATION                     0x03030256
#define OID_FDDI_PATH_T_R_MODE                                  0x03030257
#define OID_FDDI_PATH_SBA_AVAILABLE                     0x03030258
#define OID_FDDI_PATH_TVX_LOWER_BOUND                   0x03030259
#define OID_FDDI_PATH_T_MAX_LOWER_BOUND                 0x0303025A
#define OID_FDDI_PATH_MAX_T_REQ                                 0x0303025B
#define OID_FDDI_PORT_MY_TYPE                           0x0303025C
#define OID_FDDI_PORT_NEIGHBOR_TYPE                     0x0303025D
#define OID_FDDI_PORT_CONNECTION_POLICIES     0x0303025E
#define OID_FDDI_PORT_MAC_INDICATED                     0x0303025F
#define OID_FDDI_PORT_CURRENT_PATH                      0x03030260
#define OID_FDDI_PORT_REQUESTED_PATHS                   0x03030261
#define OID_FDDI_PORT_MAC_PLACEMENT                     0x03030262
#define OID_FDDI_PORT_AVAILABLE_PATHS                   0x03030263
#define OID_FDDI_PORT_MAC_LOOP_TIME                     0x03030264
#define OID_FDDI_PORT_PMD_CLASS                                 0x03030265
#define OID_FDDI_PORT_CONNECTION_CAPABILITIES           0x03030266
#define OID_FDDI_PORT_INDEX                                     0x03030267
#define OID_FDDI_PORT_MAINT_LS                                  0x03030268
#define OID_FDDI_PORT_BS_FLAG                           0x03030269
#define OID_FDDI_PORT_PC_LS                                     0x0303026A
#define OID_FDDI_PORT_EB_ERROR_CT                       0x0303026B
#define OID_FDDI_PORT_LCT_FAIL_CT                       0x0303026C
#define OID_FDDI_PORT_LER_ESTIMATE                      0x0303026D
#define OID_FDDI_PORT_LEM_REJECT_CT                     0x0303026E
#define OID_FDDI_PORT_LEM_CT                            0x0303026F
#define OID_FDDI_PORT_LER_CUTOFF                        0x03030270
#define OID_FDDI_PORT_LER_ALARM                                 0x03030271
#define OID_FDDI_PORT_CONNNECT_STATE                    0x03030272
#define OID_FDDI_PORT_PCM_STATE                                 0x03030273
#define OID_FDDI_PORT_PC_WITHHOLD                       0x03030274
#define OID_FDDI_PORT_LER_FLAG                                  0x03030275
#define OID_FDDI_PORT_HARDWARE_PRESENT                  0x03030276
#define OID_FDDI_SMT_STATION_ACTION                     0x03030277
#define OID_FDDI_PORT_ACTION                            0x03030278
#define OID_FDDI_IF_DESCR                                       0x03030279
#define OID_FDDI_IF_TYPE                                        0x0303027A
#define OID_FDDI_IF_MTU                                                 0x0303027B
#define OID_FDDI_IF_SPEED                                       0x0303027C
#define OID_FDDI_IF_PHYS_ADDRESS                        0x0303027D
#define OID_FDDI_IF_ADMIN_STATUS                        0x0303027E
#define OID_FDDI_IF_OPER_STATUS                                 0x0303027F
#define OID_FDDI_IF_LAST_CHANGE                                 0x03030280
#define OID_FDDI_IF_IN_OCTETS                           0x03030281
#define OID_FDDI_IF_IN_UCAST_PKTS                       0x03030282
#define OID_FDDI_IF_IN_NUCAST_PKTS                      0x03030283
#define OID_FDDI_IF_IN_DISCARDS                                 0x03030284
#define OID_FDDI_IF_IN_ERRORS                           0x03030285
#define OID_FDDI_IF_IN_UNKNOWN_PROTOS                   0x03030286
#define OID_FDDI_IF_OUT_OCTETS                                  0x03030287
#define OID_FDDI_IF_OUT_UCAST_PKTS                      0x03030288
#define OID_FDDI_IF_OUT_NUCAST_PKTS                     0x03030289
#define OID_FDDI_IF_OUT_DISCARDS                        0x0303028A
#define OID_FDDI_IF_OUT_ERRORS                                  0x0303028B
#define OID_FDDI_IF_OUT_QLEN                            0x0303028C
#define OID_FDDI_IF_SPECIFIC                            0x0303028D
//
// WAN objects
//
#define OID_WAN_PERMANENT_ADDRESS                       0x04010101
#define OID_WAN_CURRENT_ADDRESS                                 0x04010102
#define OID_WAN_QUALITY_OF_SERVICE                      0x04010103
#define OID_WAN_PROTOCOL_TYPE                           0x04010104
#define OID_WAN_MEDIUM_SUBTYPE                                  0x04010105
#define OID_WAN_HEADER_FORMAT                           0x04010106
#define OID_WAN_GET_INFO                                        0x04010107
#define OID_WAN_SET_LINK_INFO                           0x04010108
#define OID_WAN_GET_LINK_INFO                           0x04010109
#define OID_WAN_LINE_COUNT                                      0x0401010A
#define OID_WAN_GET_BRIDGE_INFO                                 0x0401020A
#define OID_WAN_SET_BRIDGE_INFO                                 0x0401020B
#define OID_WAN_GET_COMP_INFO                           0x0401020C
#define OID_WAN_SET_COMP_INFO                           0x0401020D
#define OID_WAN_GET_STATS_INFO                                  0x0401020E
//
// LocalTalk objects
//
#define OID_LTALK_CURRENT_NODE_ID                       0x05010102
#define OID_LTALK_IN_BROADCASTS                                 0x05020101
```

```
#define OID_LTALK_IN_LENGTH_ERRORS                          0x05020102
#define OID_LTALK_OUT_NO_HANDLERS                           0x05020201
#define OID_LTALK_COLLISIONS                                0x05020202
#define OID_LTALK_DEFERS                                          0x05020203
#define OID_LTALK_NO_DATA_ERRORS                            0x05020204
#define OID_LTALK_RANDOM_CTS_ERRORS                         0x05020205
#define OID_LTALK_FCS_ERRORS                                0x05020206
//
// Arcnet objects
//
#define OID_ARCNET_PERMANENT_ADDRESS                0x06010101
#define OID_ARCNET_CURRENT_ADDRESS                          0x06010102
#define OID_ARCNET_RECONFIGURATIONS                         0x06020201
//
// TAPI objects
//
#define OID_TAPI_ACCEPT                                                   0x07030101
#define OID_TAPI_ANSWER                                                   0x07030102
#define OID_TAPI_CLOSE                                              0x07030103
#define OID_TAPI_CLOSE_CALL                                         0x07030104
#define OID_TAPI_CONDITIONAL_MEDIA_DETECTION 0x07030105
#define OID_TAPI_CONFIG_DIALOG                                      0x07030106
#define OID_TAPI_DEV_SPECIFIC                               0x07030107
#define OID_TAPI_DIAL                                               0x07030108
#define OID_TAPI_DROP                                               0x07030109
#define OID_TAPI_GET_ADDRESS_CAPS                           0x0703010A
#define OID_TAPI_GET_ADDRESS_ID                                     0x0703010B
#define OID_TAPI_GET_ADDRESS_STATUS                         0x0703010C
#define OID_TAPI_GET_CALL_ADDRESS_ID                0x0703010D
#define OID_TAPI_GET_CALL_INFO                                      0x0703010E
#define OID_TAPI_GET_CALL_STATUS                            0x0703010F
#define OID_TAPI_GET_DEV_CAPS                               0x07030110
#define OID_TAPI_GET_DEV_CONFIG                                     0x07030111
#define OID_TAPI_GET_EXTENSION_ID                           0x07030112
#define OID_TAPI_GET_ID                                                   0x07030113
#define OID_TAPI_GET_LINE_DEV_STATUS                0x07030114
#define OID_TAPI_MAKE_CALL                                          0x07030115
#define OID_TAPI_NEGOTIATE_EXT_VERSION                      0x07030116
#define OID_TAPI_OPEN                                               0x07030117
#define OID_TAPI_PROVIDER_INITIALIZE                0x07030118
#define OID_TAPI_PROVIDER_SHUTDOWN                          0x07030119
#define OID_TAPI_SECURE_CALL                                0x0703011A
#define OID_TAPI_SELECT_EXT_VERSION                         0x0703011B
#define OID_TAPI_SEND_USER_USER_INFO                0x0703011C
#define OID_TAPI_SET_APP_SPECIFIC                           0x0703011D
#define OID_TAPI_SET_CALL_PARAMS                            0x0703011E
#define OID_TAPI_SET_DEFAULT_MEDIA_DETECTION 0x0703011F
#define OID_TAPI_SET_DEV_CONFIG                                     0x07030120
#define OID_TAPI_SET_MEDIA_MODE                                     0x07030121
#define OID_TAPI_SET_STATUS_MESSAGES                0x07030122
//
// ATM Connection Oriented Ndis
//
#define OID_ATM_SUPPORTED_VC_RATES                          0x08010101
#define OID_ATM_SUPPORTED_SERVICE_CATEGORY          0x08010102
#define OID_ATM_SUPPORTED_AAL_TYPES                         0x08010103
#define OID_ATM_HW_CURRENT_ADDRESS                          0x08010104
#define OID_ATM_MAX_ACTIVE_VCS                                    0x08010105
#define OID_ATM_MAX_ACTIVE_VCI_BITS                         0x08010106
#define OID_ATM_MAX_ACTIVE_VPI_BITS                         0x08010107
#define OID_ATM_MAX_AAL0_PACKET_SIZE                0x08010108
#define OID_ATM_MAX_AAL1_PACKET_SIZE                0x08010109
#define OID_ATM_MAX_AAL34_PACKET_SIZE               0x0801010A
#define OID_ATM_MAX_AAL5_PACKET_SIZE                0x0801010B
#define OID_ATM_SIGNALING_VPIVCI                            0x08010201
#define OID_ATM_ASSIGNED_VPI                                0x08010202
#define OID_ATM_ACQUIRE_ACCESS_NET_RESOURCES 0x08010203
#define OID_ATM_RELEASE_ACCESS_NET_RESOURCES 0x08010204
#define OID_ATM_ILMI_VPIVCI                                       0x08010205
#define OID_ATM_DIGITAL_BROADCAST_VPIVCI            0x08010206
#define OID_ATM_GET_NEAREST_FLOW                            0x08010207
#define OID_ATM_ALIGNMENT_REQUIRED                          0x08010208
//
//      ATM specific statistics OIDs.
//
#define OID_ATM_RCV_CELLS_OK                                0x08020101
#define OID_ATM_XMIT_CELLS_OK                               0x08020102
#define OID_ATM_RCV_CELLS_DROPPED                           0x08020103
#define OID_ATM_RCV_INVALID_VPI_VCI                         0x08020201
#define OID_ATM_CELLS_HEC_ERROR                                   0x08020202
#define OID_ATM_RCV_REASSEMBLY_ERROR                0x08020203
```

```
//
// PCCA (Wireless) object
//
//
// All WirelessWAN devices must support the following OIDs
//
#define OID_WW_GEN_NETWORK_TYPES_SUPPORTED          0x09010101
#define OID_WW_GEN_NETWORK_TYPE_IN_USE                      0x09010102
#define OID_WW_GEN_HEADER_FORMATS_SUPPORTED         0x09010103
#define OID_WW_GEN_HEADER_FORMAT_IN_USE                     0x09010104
#define OID_WW_GEN_INDICATION_REQUEST               0x09010105
#define OID_WW_GEN_DEVICE_INFO                                      0x09010106
#define OID_WW_GEN_OPERATION_MODE                           0x09010107
#define OID_WW_GEN_LOCK_STATUS                                      0x09010108
#define OID_WW_GEN_DISABLE_TRANSMITTER                      0x09010109
#define OID_WW_GEN_NETWORK_ID                               0x0901010A
#define OID_WW_GEN_PERMANENT_ADDRESS                0x0901010B
#define OID_WW_GEN_CURRENT_ADDRESS                          0x0901010C
#define OID_WW_GEN_SUSPEND_DRIVER                           0x0901010D
#define OID_WW_GEN_BASESTATION_ID                           0x0901010E
#define OID_WW_GEN_CHANNEL_ID                               0x0901010F
#define OID_WW_GEN_ENCRYPTION_SUPPORTED                     0x09010110
#define OID_WW_GEN_ENCRYPTION_IN_USE                0x09010111
#define OID_WW_GEN_ENCRYPTION_STATE                         0x09010112
#define OID_WW_GEN_CHANNEL_QUALITY                          0x09010113
#define OID_WW_GEN_REGISTRATION_STATUS                      0x09010114
#define OID_WW_GEN_RADIO_LINK_SPEED                         0x09010115
#define OID_WW_GEN_LATENCY                                          0x09010116
#define OID_WW_GEN_BATTERY_LEVEL                            0x09010117
#define OID_WW_GEN_EXTERNAL_POWER                           0x09010118
//
// Network Dependent OIDs - Mobitex:
//
#define OID_WW_MBX_SUBADDR                                          0x09050101
// OID 0x09050102 is reserved and may not be used
#define OID_WW_MBX_FLEXLIST                                         0x09050103
#define OID_WW_MBX_GROUPLIST                                0x09050104
#define OID_WW_MBX_TRAFFIC_AREA                                     0x09050105
#define OID_WW_MBX_LIVE_DIE                                         0x09050106
#define OID_WW_MBX_TEMP_DEFAULTLIST                         0x09050107
//
// Network Dependent OIDs - Pinpoint:
//
#define OID_WW_PIN_LOC_AUTHORIZE                            0x09090101
#define OID_WW_PIN_LAST_LOCATION                            0x09090102
#define OID_WW_PIN_LOC_FIX                                          0x09090103
//
// Network Dependent - CDPD:
//
#define OID_WW_CDPD_SPNI                                            0x090D0101
#define OID_WW_CDPD_WAS1                                            0x090D0102
#define OID_WW_CDPD_AREA_COLOR                                      0x090D0103
#define OID_WW_CDPD_TX_POWER_LEVEL                          0x090D0104
#define OID_WW_CDPD_EID                                                     0x090D0105
#define OID_WW_CDPD_HEADER_COMPRESSION                      0x090D0106
#define OID_WW_CDPD_DATA_COMPRESSION                0x090D0107
#define OID_WW_CDPD_CHANNEL_SELECT                          0x090D0108
#define OID_WW_CDPD_CHANNEL_STATE                           0x090D0109
#define OID_WW_CDPD_NEI                                                     0x090D010A
#define OID_WW_CDPD_NEI_STATE                               0x090D010B
#define OID_WW_CDPD_SERVICE_PROVIDER_IDENTIFIER     0x090D010C
#define OID_WW_CDPD_SLEEP_MODE                                      0x090D010D
#define OID_WW_CDPD_CIRCUIT_SWITCHED                0x090D010E
#define OID_WW_CDPD_TEI                                                     0x090D010F
#define OID_WW_CDPD_RSSI                                            0x090D0110
//
// Network Dependent - Ardis:
//
#define OID_WW_ARD_SNDCP                                            0x09110101
#define OID_WW_ARD_TMLY_MSG                                         0x09110102
#define OID_WW_ARD_DATAGRAM                                         0x09110103
//
// Network Dependent - DataTac:
//
#define OID_WW_TAC_COMPRESSION                                      0x09150101
#define OID_WW_TAC_SET_CONFIG                               0x09150102
#define OID_WW_TAC_GET_STATUS                               0x09150103
#define OID_WW_TAC_USER_HEADER                                      0x09150104
//
// Network Dependent - Metricom:
//
```

```
#define OID_WW_MET_FUNCTION                                     0x09190101
//
// IRDA objects
//
#define OID_IRDA_RECEIVING                                      0x0A010100
#define OID_IRDA_TURNAROUND_TIME                        0x0A010101
#define OID_IRDA_SUPPORTED_SPEEDS                       0x0A010102
#define OID_IRDA_LINK_SPEED                                     0x0A010103
#define OID_IRDA_MEDIA_BUSY                                     0x0A010104
#define OID_IRDA_EXTRA_RCV_BOFS                                 0x0A010200
#define OID_IRDA_RATE_SNIFF                                     0x0A010201
#define OID_IRDA_UNICAST_LIST                           0x0A010202
#define OID_IRDA_MAX_UNICAST_LIST_SIZE                  0x0A010203
#define OID_IRDA_MAX_RECEIVE_WINDOW_SIZE        0x0A010204
#define OID_IRDA_MAX_SEND_WINDOW_SIZE           0x0A010205
//
// Medium the Ndis Driver is running on (OID_GEN_MEDIA_SUPPORTED/
// OID_GEN_MEDIA_IN_USE).
//
typedef enum _NDIS_MEDIUM {
    NdisMedium802_3,
    NdisMedium802_5,
    NdisMediumFddi,
    NdisMediumWan,
    NdisMediumLocalTalk,
    NdisMediumDix,              // defined for convenience, not a real medium
    NdisMediumArcnetRaw,
    NdisMediumArcnet878_2,
    NdisMediumAtm,
    NdisMediumWirelessWan,
    NdisMediumIrda,
    NdisMediumMax               // Not a real medium, defined as an upper-bound
} NDIS_MEDIUM, *PNDIS_MEDIUM;

//
// Hardware status codes (OID_GEN_HARDWARE_STATUS).
//
typedef enum _NDIS_HARDWARE_STATUS {
    NdisHardwareStatusReady,
    NdisHardwareStatusInitializing,
    NdisHardwareStatusReset,
    NdisHardwareStatusClosing,
    NdisHardwareStatusNotReady
} NDIS_HARDWARE_STATUS, *PNDIS_HARDWARE_STATUS;

//
// this is the type passed in the OID_GEN_GET_TIME_CAPS request
//
typedef struct _GEN_GET_TIME_CAPS {
    ULONG Flags;                // Bits defined below

    ULONG ClockPrecision;
} GEN_GET_TIME_CAPS, *PGEN_GET_TIME_CAPS;

#define READABLE_LOCAL_CLOCK                            0x000000001
#define CLOCK_NETWORK_DERIVED                           0x000000002
#define CLOCK_PRECISION                                                 0x000000004
#define RECEIVE_TIME_INDICATION_CAPABLE                 0x000000008
#define TIMED_SEND_CAPABLE                                      0x000000010
#define TIME_STAMP_CAPABLE                                      0x000000020
//
//
// this is the type passed in the OID_GEN_GET_NETCARD_TIME request
//
typedef struct _GEN_GET_NETCARD_TIME {
    ULONG ReadTime;
} GEN_GET_NETCARD_TIME, *PGEN_GET_NETCARD_TIME;

//
// Defines the attachment types for FDDI (OID_FDDI_ATTACHMENT_TYPE).
//
typedef enum _NDIS_FDDI_ATTACHMENT_TYPE {
    NdisFddiTypeIsolated = 1,
    NdisFddiTypeLocalA,
    NdisFddiTypeLocalB,
    NdisFddiTypeLocalAB,
    NdisFddiTypeLocalS,
    NdisFddiTypeWrapA,
    NdisFddiTypeWrapB,
    NdisFddiTypeWrapAB,
    NdisFddiTypeWrapS,
```

```
    NdisFddiTypeCWrapA,
    NdisFddiTypeCWrapB,
    NdisFddiTypeCWrapS,
    NdisFddiTypeThrough
} NDIS_FDDI_ATTACHMENT_TYPE, *PNDIS_FDDI_ATTACHMENT_TYPE;

//
// Defines the ring management states for FDDI (OID_FDDI_RING_MGT_STATE).
//
typedef enum _NDIS_FDDI_RING_MGT_STATE {
    NdisFddiRingIsolated = 1,
    NdisFddiRingNonOperational,
    NdisFddiRingOperational,
    NdisFddiRingDetect,
    NdisFddiRingNonOperationalDup,
    NdisFddiRingOperationalDup,
    NdisFddiRingDirected,
    NdisFddiRingTrace
} NDIS_FDDI_RING_MGT_STATE, *PNDIS_FDDI_RING_MGT_STATE;

//
// Defines the Lconnection state for FDDI (OID_FDDI_LCONNECTION_STATE).
//
typedef enum _NDIS_FDDI_LCONNECTION_STATE {
    NdisFddiStateOff = 1,
    NdisFddiStateBreak,
    NdisFddiStateTrace,
    NdisFddiStateConnect,
    NdisFddiStateNext,
    NdisFddiStateSignal,
    NdisFddiStateJoin,
    NdisFddiStateVerify,
    NdisFddiStateActive,
    NdisFddiStateMaintenance
} NDIS_FDDI_LCONNECTION_STATE, *PNDIS_FDDI_LCONNECTION_STATE;

//
// Defines the medium subtypes for WAN medium (OID_WAN_MEDIUM_SUBTYPE).
//
typedef enum _NDIS_WAN_MEDIUM_SUBTYPE {
    NdisWanMediumHub,
    NdisWanMediumX_25,
    NdisWanMediumIsdn,
    NdisWanMediumSerial,
    NdisWanMediumFrameRelay,
    NdisWanMediumAtm,
    NdisWanMediumSonet,
    NdisWanMediumSW56K
} NDIS_WAN_MEDIUM_SUBTYPE, *PNDIS_WAN_MEDIUM_SUBTYPE;

//
// Defines the header format for WAN medium (OID_WAN_HEADER_FORMAT).
//
typedef enum _NDIS_WAN_HEADER_FORMAT {
    NdisWanHeaderNative,      // src/dest based on subtype, followed by NLPID
     NdisWanHeaderEthernet    // emulation of ethernet header
} NDIS_WAN_HEADER_FORMAT, *PNDIS_WAN_HEADER_FORMAT;

//
// Defines the line quality on a WAN line (OID_WAN_QUALITY_OF_SERVICE).
//
typedef enum _NDIS_WAN_QUALITY {
    NdisWanRaw,
    NdisWanErrorControl,
    NdisWanReliable
} NDIS_WAN_QUALITY, *PNDIS_WAN_QUALITY;

//
// Defines the state of a token-ring adapter (OID_802_5_CURRENT_RING_STATE).
//
typedef enum _NDIS_802_5_RING_STATE {
    NdisRingStateOpened = 1,
    NdisRingStateClosed,
    NdisRingStateOpening,
    NdisRingStateClosing,
    NdisRingStateOpenFailure,
    NdisRingStateRingFailure
} NDIS_802_5_RING_STATE, *PNDIS_802_5_RING_STATE;

//
// Defines the state of the LAN media
```

```
//
typedef enum _NDIS_MEDIA_STATE {
    NdisMediaStateConnected,
    NdisMediaStateDisconnected
} NDIS_MEDIA_STATE, *PNDIS_MEDIA_STATE;

//
// The following is set on a per-packet basis as OOB data with NdisClass802_3Priority
//
typedef ULONG Priority_802_3; // 0-7 priority levels
//
//      The following structure is used to query OID_GEN_CO_LINK_SPEED and
//      OID_GEN_CO_MINIMUM_LINK_SPEED.  The first OID will return the current
//      link speed of the adapter.  The second will return the minimum link speed
//      the adapter is capable of.
//

typedef struct _NDIS_CO_LINK_SPEED {
    ULONG Outbound;
    ULONG Inbound;
} NDIS_CO_LINK_SPEED,

*PNDIS_CO_LINK_SPEED;
//
// Ndis Packet Filter Bits (OID_GEN_CURRENT_PACKET_FILTER).
//
#define NDIS_PACKET_TYPE_DIRECTED                   0x0001
#define NDIS_PACKET_TYPE_MULTICAST                  0x0002
#define NDIS_PACKET_TYPE_ALL_MULTICAST              0x0004
#define NDIS_PACKET_TYPE_BROADCAST                  0x0008
#define NDIS_PACKET_TYPE_SOURCE_ROUTING             0x0010
#define NDIS_PACKET_TYPE_PROMISCUOUS                0x0020
#define NDIS_PACKET_TYPE_SMT                        0x0040
#define NDIS_PACKET_TYPE_ALL_LOCAL                  0x0080
#define NDIS_PACKET_TYPE_MAC_FRAME                  0x8000
#define NDIS_PACKET_TYPE_FUNCTIONAL                 0x4000
#define NDIS_PACKET_TYPE_ALL_FUNCTIONAL             0x2000
#define NDIS_PACKET_TYPE_GROUP                      0x1000
//
// Ndis Token-Ring Ring Status Codes (OID_802_5_CURRENT_RING_STATUS).
//
#define NDIS_RING_SIGNAL_LOSS                       0x00008000
#define NDIS_RING_HARD_ERROR                        0x00004000
#define NDIS_RING_SOFT_ERROR                        0x00002000
#define NDIS_RING_TRANSMIT_BEACON                   0x00001000
#define NDIS_RING_LOBE_WIRE_FAULT                   0x00000800
#define NDIS_RING_AUTO_REMOVAL_ERROR                0x00000400
#define NDIS_RING_REMOVE_RECEIVED                   0x00000200
#define NDIS_RING_COUNTER_OVERFLOW                  0x00000100
#define NDIS_RING_SINGLE_STATION                    0x00000080
#define NDIS_RING_RING_RECOVERY                     0x00000040
//
// Ndis protocol option bits (OID_GEN_PROTOCOL_OPTIONS).
//
#define NDIS_PROT_OPTION_ESTIMATED_LENGTH    0x00000001
#define NDIS_PROT_OPTION_NO_LOOPBACK         0x00000002
#define NDIS_PROT_OPTION_NO_RSVD_ON_RCVPKT   0x00000004
//
// Ndis MAC option bits (OID_GEN_MAC_OPTIONS).
//
#define NDIS_MAC_OPTION_COPY_LOOKAHEAD_DATA  0x00000001
#define NDIS_MAC_OPTION_RECEIVE_SERIALIZED   0x00000002
#define NDIS_MAC_OPTION_TRANSFERS_NOT_PEND   0x00000004
#define NDIS_MAC_OPTION_NO_LOOPBACK          0x00000008
#define NDIS_MAC_OPTION_FULL_DUPLEX          0x00000010
#define NDIS_MAC_OPTION_EOTX_INDICATION      0x00000020
#define NDIS_MAC_OPTION_RESERVED             0x80000000
//
//      NDIS MAC option bits for OID_GEN_CO_MAC_OPTIONS.
//
#define NDIS_CO_MAC_OPTION_DYNAMIC_LINK_SPEED 0x00000001
#ifdef  IRDA
//
// The following is set on a per-packet basis as OOB data with NdisClassIrdaPacketInfo
// This is the per-packet info specified on a per-packet basis
//
typedef struct _NDIS_IRDA_PACKET_INFO {
    UINT ExtraBOFs;
    UINT MinTurnAroundTime;
} NDIS_IRDA_PACKET_INFO, *PNDIS_IRDA_PACKET_INFO;
```

```
#endif
#ifdef  WIRELESS_WAN
//
// Wireless WAN structure definitions
//
//
// currently defined Wireless network subtypes
//
typedef enum _NDIS_WW_NETWORK_TYPE {
    NdisWWGeneric,
    NdisWWMobitex,
    NdisWWPinpoint,
    NdisWWCDPD,
    NdisWWArdis,
    NdisWWDataTAC,
    NdisWWMetricom,
    NdisWWGSM,
    NdisWWCDMA,
    NdisWWTDMA,
    NdisWWAMPS,
    NdisWWInmarsat,
    NdisWWpACT
} NDIS_WW_NETWORK_TYPE;

//
// currently defined header formats
//
typedef enum _NDIS_WW_HEADER_FORMAT {
    NdisWWDIXEthernetFrames,
    NdisWWMPAKFrames,
    NdisWWRDLAPFrames,
    NdisWWMDC4800Frames
} NDIS_WW_HEADER_FORMAT;

//
// currently defined encryption types
//
typedef enum _NDIS_WW_ENCRYPTION_TYPE {
    NdisWWUnknownEncryption = -1,
    NdisWWNoEncryption,
    NdisWWDefaultEncryption
} NDIS_WW_ENCRYPTION_TYPE, *PNDIS_WW_ENCRYPTION_TYPE;

//
// OID_WW_GEN_INDICATION_REQUEST
//
typedef struct _NDIS_WW_INDICATION_REQUEST {
    NDIS_OID Oid;              // IN

    UINT uIndicationFlag;      // IN

    UINT uApplicationToken;    // IN OUT

    HANDLE hIndicationHandle; // IN OUT

    INT iPollingInterval;      // IN OUT

    NDIS_VAR_DATA_DESC InitialValue;  // IN OUT

    NDIS_VAR_DATA_DESC OIDIndicationValue;   // OUT - only valid after indication

    NDIS_VAR_DATA_DESC TriggerValue;  // IN

} NDIS_WW_INDICATION_REQUEST, *PNDIS_WW_INDICATION_REQUEST;

#define OID_INDICATION_REQUEST_ENABLE                0x0000
#define OID_INDICATION_REQUEST_CANCEL                0x0001
//
// OID_WW_GEN_DEVICE_INFO
//
typedef struct _WW_DEVICE_INFO {
    NDIS_VAR_DATA_DESC Manufacturer;
    NDIS_VAR_DATA_DESC ModelNum;
    NDIS_VAR_DATA_DESC SWVersionNum;
    NDIS_VAR_DATA_DESC SerialNum;
} WW_DEVICE_INFO, *PWW_DEVICE_INFO;

//
// OID_WW_GEN_OPERATION_MODE
//
typedef INT WW_OPERATION_MODE;        //  0 = Normal mode
```

```
                                                        //  1 = Power saving mode
                                                        // -1 = mode unknown
//
// OID_WW_GEN_LOCK_STATUS
//
typedef INT WW_LOCK_STATUS;    //  0 = unlocked
                                                        //  1 = locked
                                                        // -1 = unknown lock status
//
// OID_WW_GEN_DISABLE_TRANSMITTER
//
typedef INT WW_DISABLE_TRANSMITTER;   //  0 = transmitter enabled
                                                        //  1 = transmitter disabled
                                                        // -1 = unknown value
//
// OID_WW_GEN_NETWORK_ID
//

typedef NDIS_VAR_DATA_DESC WW_NETWORK_ID;
//
// OID_WW_GEN_PERMANENT_ADDRESS
//
typedef NDIS_VAR_DATA_DESC WW_PERMANENT_ADDRESS;
//
// OID_WW_GEN_CURRENT_ADDRESS
//
typedef struct _WW_CURRENT_ADDRESS {
    NDIS_WW_HEADER_FORMAT Format;
    NDIS_VAR_DATA_DESC Address;
} WW_CURRENT_ADDRESS, *PWW_CURRENT_ADDRESS;

//
// OID_WW_GEN_SUSPEND_DRIVER
//
typedef BOOLEAN WW_SUSPEND_DRIVER;    // 0 = driver operational
                                                                    // 1 = driver suspended
//
// OID_WW_GEN_BASESTATION_ID
//

typedef NDIS_VAR_DATA_DESC WW_BASESTATION_ID;
//
// OID_WW_GEN_CHANNEL_ID
//
typedef NDIS_VAR_DATA_DESC WW_CHANNEL_ID;
//
// OID_WW_GEN_ENCRYPTION_STATE
//
typedef BOOLEAN WW_ENCRYPTION_STATE;  // 0 = if encryption is disabled
                                                        // 1 = if encryption is enabled
//
// OID_WW_GEN_CHANNEL_QUALITY
//
typedef INT WW_CHANNEL_QUALITY;       //  0 = Not in network contact,
                                      // 1-100 = Quality of Channel (100 is highest quality).
                                      // -1 = channel quality is unknown
//
// OID_WW_GEN_REGISTRATION_STATUS
//

typedef INT WW_REGISTRATION_STATUS;   //  0 = Registration denied
                                      //  1 = Registration pending
                                      //  2 = Registered
                                      // -1 = unknown registration status
//
// OID_WW_GEN_RADIO_LINK_SPEED
//

typedef UINT WW_RADIO_LINK_SPEED;     // Bits per second.
//
// OID_WW_GEN_LATENCY
//

typedef UINT WW_LATENCY;      //  milliseconds
//
// OID_WW_GEN_BATTERY_LEVEL
//

typedef INT WW_BATTERY_LEVEL; //  0-100 = battery level in percentage
                      //      (100=fully charged)
                      // -1 = unknown battery level.
```

```
//
// OID_WW_GEN_EXTERNAL_POWER
//

typedef INT WW_EXTERNAL_POWER;          //   0 = no external power connected
                                        //   1 = external power connected
                                        //  -1 = unknown
//
// OID_WW_MET_FUNCTION
//

typedef NDIS_VAR_DATA_DESC WW_MET_FUNCTION;
//
// OID_WW_TAC_COMPRESSION
//
typedef BOOLEAN WW_TAC_COMPRESSION;     // Determines whether or not network level compression
                                        // is being used.
//
// OID_WW_TAC_SET_CONFIG
//

typedef struct _WW_TAC_SETCONFIG {
    NDIS_VAR_DATA_DESC RCV_MODE;
    NDIS_VAR_DATA_DESC TX_CONTROL;
    NDIS_VAR_DATA_DESC RX_CONTROL;
    NDIS_VAR_DATA_DESC FLOW_CONTROL;
    NDIS_VAR_DATA_DESC RESET_CNF;
    NDIS_VAR_DATA_DESC READ_CNF;
} WW_TAC_SETCONFIG, *PWW_TAC_SETCONFIG;

//
// OID_WW_TAC_GET_STATUS
//
typedef struct _WW_TAC_GETSTATUS {
    BOOLEAN Action;             // Set = Execute command.

    NDIS_VAR_DATA_DESC Command;
    NDIS_VAR_DATA_DESC Option;
    NDIS_VAR_DATA_DESC Response;        // The response to the requested command
    // - max. length of string is 256 octets.

} WW_TAC_GETSTATUS, *PWW_TAC_GETSTATUS;

//
// OID_WW_TAC_USER_HEADER
//
typedef NDIS_VAR_DATA_DESC WW_TAC_USERHEADER;       // This will hold the user header - Max. 64
octets.
//
// OID_WW_ARD_SNDCP
//

typedef struct _WW_ARD_SNDCP {
    NDIS_VAR_DATA_DESC Version;         // The version of SNDCP protocol supported.

    INT BlockSize;              // The block size used for SNDCP

    INT Window;                         // The window size used in SNDCP

} WW_ARD_SNDCP, *PWW_ARD_SNDCP;

//
// OID_WW_ARD_TMLY_MSG
//
typedef BOOLEAN WW_ARD_CHANNEL_STATUS;      // The current status of the inbound RF Channel.
//
// OID_WW_ARD_DATAGRAM
//

typedef struct _WW_ARD_DATAGRAM {
    BOOLEAN LoadLevel;          // Byte that contains the load level info.

    INT SessionTime;            // Datagram session time remaining.

    NDIS_VAR_DATA_DESC HostAddr;        // Host address.

    NDIS_VAR_DATA_DESC THostAddr;       // Test host address.

} WW_ARD_DATAGRAM, *PWW_ARD_DATAGRAM;

//
```

```
// OID_WW_CDPD_SPNI
//
typedef struct _WW_CDPD_SPNI {
    UINT SPNI[10];              //10 16-bit service provider network IDs

    INT OperatingMode;          // 0 = ignore SPNI,
    // 1 = require SPNI from list,
    // 2 = prefer SPNI from list.
    // 3 = exclude SPNI from list.

} WW_CDPD_SPNI, *PWW_CDPD_SPNI;

//
// OID_WW_CDPD_WASI
//
typedef struct _WW_CDPD_WIDE_AREA_SERVICE_ID {
    UINT WASI[10];              //10 16-bit wide area service IDs

    INT OperatingMode;          // 0 = ignore WASI,
    // 1 = Require WASI from list,
    // 2 = prefer WASI from list
    // 3 = exclude WASI from list.

} WW_CDPD_WIDE_AREA_SERVICE_ID, *PWW_CDPD_WIDE_AREA_SERVICE_ID;

//
// OID_WW_CDPD_AREA_COLOR
//
typedef INT WW_CDPD_AREA_COLOR;
//
// OID_WW_CDPD_TX_POWER_LEVEL
//
typedef UINT WW_CDPD_TX_POWER_LEVEL;
//
// OID_WW_CDPD_EID
//
typedef NDIS_VAR_DATA_DESC WW_CDPD_EID;
//
// OID_WW_CDPD_HEADER_COMPRESSION
//
typedef INT WW_CDPD_HEADER_COMPRESSION;             //  0 = no header compression,
                                                    //  1 = always compress headers,
                                                    //  2 = compress headers if MD-IS does
                                                    // -1 = unknown
//
// OID_WW_CDPD_DATA_COMPRESSION
//
typedef INT WW_CDPD_DATA_COMPRESSION; // 0  = no data compression,
                                                    // 1  = data compression enabled
                                                    // -1 =  unknown
//
// OID_WW_CDPD_CHANNEL_SELECT
//

typedef struct _WW_CDPD_CHANNEL_SELECT {
    UINT ChannelID;             // channel number

    UINT fixedDuration;                 // duration in seconds

} WW_CDPD_CHANNEL_SELECT, *PWW_CDPD_CHANNEL_SELECT;

//
// OID_WW_CDPD_CHANNEL_STATE
//
typedef enum _WW_CDPD_CHANNEL_STATE {
    CDPDChannelNotAvail,
    CDPDChannelScanning,
    CDPDChannelInitAcquired,
    CDPDChannelAcquired,
    CDPDChannelSleeping,
    CDPDChannelWaking,
    CDPDChannelCSDialing,
    CDPDChannelCSRedial,
    CDPDChannelCSAnswering,
    CDPDChannelCSConnected,
    CDPDChannelCSSuspended
} WW_CDPD_CHANNEL_STATE, *PWW_CDPD_CHANNEL_STATE;

//
// OID_WW_CDPD_NEI
//
```

```
typedef enum _WW_CDPD_NEI_FORMAT {
    CDPDNeiIPv4,
    CDPDNeiCLNP,
    CDPDNeiIPv6
} WW_CDPD_NEI_FORMAT, *PWW_CDPD_NEI_FORMAT;
typedef enum _WW_CDPD_NEI_TYPE {
    CDPDNeiIndividual,
    CDPDNeiMulticast,
    CDPDNeiBroadcast
} WW_CDPD_NEI_TYPE;
typedef struct _WW_CDPD_NEI {
    UINT uNeiIndex;
    WW_CDPD_NEI_FORMAT NeiFormat;
    WW_CDPD_NEI_TYPE NeiType;
    WORD NeiGmid;              // group member identifier, only
    // meaningful if NeiType ==
    // CDPDNeiMulticast

    NDIS_VAR_DATA_DESC NeiAddress;
} WW_CDPD_NEI;

//
// OID_WW_CDPD_NEI_STATE
//
typedef enum _WW_CDPD_NEI_STATE {
    CDPDUnknown,
    CDPDRegistered,
    CDPDDeregistered
} WW_CDPD_NEI_STATE, *PWW_CDPD_NEI_STATE;
typedef enum _WW_CDPD_NEI_SUB_STATE {
    CDPDPending,               // Registration pending
     CDPDNoReason,             // Registration denied - no reason given
     CDPDMDISNotCapable,       // Registration denied - MD-IS not capable of
    //  handling M-ES at this time
     CDPDNEINotAuthorized,     // Registration denied - NEI is not authorized to
    //  use this subnetwork
     CDPDInsufficientAuth,     // Registration denied - M-ES gave insufficient
    //  authentication credentials
     CDPDUnsupportedAuth,      // Registration denied - M-ES gave unsupported
    //  authentication credentials
     CDPDUsageExceeded,              // Registration denied - NEI has exceeded usage
    //  limitations
     CDPDDeniedThisNetwork     // Registration denied on this network, service
    //  may be obtained on alternate Service Provider
    //  network
} WW_CDPD_NEI_SUB_STATE;
typedef struct _WW_CDPD_NEI_REG_STATE {
    UINT uNeiIndex;
    WW_CDPD_NEI_STATE NeiState;
    WW_CDPD_NEI_SUB_STATE NeiSubState;
} WW_CDPD_NEI_REG_STATE, *PWW_CDPD_NEI_REG_STATE;

//
// OID_WW_CDPD_SERVICE_PROVIDER_IDENTIFIER
//
typedef struct _WW_CDPD_SERVICE_PROVIDER_ID {
    UINT SPI[10];              //10 16-bit service provider IDs

    INT OperatingMode;         // 0 = ignore SPI,
    // 1 = require SPI from list,
    // 2 = prefer SPI from list.
    // 3 = exclude SPI from list.

} WW_CDPD_SERVICE_PROVIDER_ID, *PWW_CDPD_SERVICE_PROVIDER_ID;

//
// OID_WW_CDPD_SLEEP_MODE
//
typedef INT WW_CDPD_SLEEP_MODE;
//
// OID_WW_CDPD_TEI
//
typedef ULONG WW_CDPD_TEI;
//
// OID_WW_CDPD_CIRCUIT_SWITCHED
//
typedef struct _WW_CDPD_CIRCUIT_SWITCHED {
    INT service_preference;    // -1 = unknown,
    //  0 = always use packet switched CDPD,
    //  1 = always use CS CDPD via AMPS,
    //  . = always use CS CDPD via PSTN,
```

```
    //  3 = use circuit switched via AMPS only
    //      when packet switched is not available.
    //  4 = use packet switched only when circuit
    //   switched via AMPS is not available.
    //  5 = device manuf. defined service
    //   preference.
    //  6 = device manuf. defined service
    //   preference.

    INT service_status;              // -1 = unknown,
    //  0 = packet switched CDPD,
    //  1 = circuit switched CDPD via AMPS,
    //  2 = circuit switched CDPD via PSTN.

    INT connect_rate;          //  CS connection bit rate (bits per second).
    //  0 = no active connection,
    // -1 = unknown
    //  Dial code last used to dial.

    NDIS_VAR_DATA_DESC dial_code[20];

    UINT sid;                  //  Current AMPS system ID

    INT a_b_side_selection;    // -1 = unknown,
    //  0 = no AMPS service
    //  1 = AMPS "A" side channels selected
    //  2 = AMPS "B" side channels selected

    INT AMPS_channel;          // -1= unknown
    //  0 = no AMPS service.
    //  1-1023 = AMPS channel number in use

    UINT action;               //  0 = no action
    //  1 = suspend (hangup)
    //  2 = dial

    //  Default dial code for CS CDPD service
    //  encoded as specified in the CS CDPD
    //  implementor guidelines.
    NDIS_VAR_DATA_DESC default_dial[20];

    //  Number for the CS CDPD network to call
    //   back the mobile, encoded as specified in
    //   the CS CDPD implementor guidelines.
    NDIS_VAR_DATA_DESC call_back[20];

    UINT sid_list[10];         //  List of 10 16-bit preferred AMPS
    //   system IDs for CS CDPD.

    UINT inactivity_timer;     //  Wait time after last data before dropping
    //   call.
    //  0-65535 = inactivity time limit (seconds).

    UINT receive_timer;              //  secs. per CS-CDPD Implementor Guidelines.

    UINT conn_resp_timer;      //  secs. per CS-CDPD Implementor Guidelines.

    UINT reconn_resp_timer;    //  secs. per CS-CDPD Implementor Guidelines.

    UINT disconn_timer;              //  secs. per CS-CDPD Implementor Guidelines.

    UINT NEI_reg_timer;              //  secs. per CS-CDPD Implementor Guidelines.

    UINT reconn_retry_timer;   //  secs. per CS-CDPD Implementor Guidelines.

    UINT link_reset_timer;     //  secs. per CS-CDPD Implementor Guidelines.

    UINT link_reset_ack_timer;       //  secs. per CS-CDPD Implementor Guidelines.

    UINT n401_retry_limit;     //  per CS-CDPD Implementor Guidelines.

    UINT n402_retry_limit;     //  per CS-CDPD Implementor Guidelines.

    UINT n404_retry_limit;     //  per CS-CDPD Implementor Guidelines.

    UINT n405_retry_limit;     //  per CS-CDPD Implementor Guidelines.

} WW_CDPD_CIRCUIT_SWITCHED, *WW_PCDPD_CIRCUIT_SWITCHED;
typedef UINT WW_CDPD_RSSI;
//
// OID_WW_PIN_LOC_AUTHORIZE
```

```
//
typedef INT WW_PIN_AUTHORIZED;          // 0  = unauthorized
                                        // 1  = authorized
                                        // -1 = unknown
//
// OID_WW_PIN_LAST_LOCATION
// OID_WW_PIN_LOC_FIX
//

typedef struct _WW_PIN_LOCATION {
    INT Latitude;               // Latitude in hundredths of a second

    INT Longitude;              // Longitude in hundredths of a second

    INT Altitude;               // Altitude in feet

    INT FixTime;                // Time of the location fix, since midnight,  local time (of the
    // current day), in tenths of a second

    INT NetTime;                // Current local network time of the current day, since midnight,
    // in tenths of a second

    INT LocQuality;             // 0-100 = location quality

    INT LatReg;                         // Latitude registration offset, in hundredths of a second

    INT LongReg;                // Longitude registration offset, in hundredths of a second

    INT GMTOffset;              // Offset in minutes of the local time zone from GMT

} WW_PIN_LOCATION, *PWW_PIN_LOCATION;

//
// The following is set on a per-packet basis as OOB data with NdisClassWirelessWanMbxMailbox
//
typedef ULONG WW_MBX_MAILBOX_FLAG;      // 1 = set mailbox flag, 0 = do not set mailbox flag
//
// OID_WW_MBX_SUBADDR
//

typedef struct _WW_MBX_PMAN {
    BOOLEAN ACTION;             // 0 = Login PMAN,  1 = Logout PMAN

    UINT MAN;
    UCHAR PASSWORD[8];          // Password should be null for Logout and indications.
    // Maximum length of password is 8 chars.

} WW_MBX_PMAN, *PWW_MBX_PMAN;

//
// OID_WW_MBX_FLEXLIST
//
typedef struct _WW_MBX_FLEXLIST {
    INT count;                  //  Number of MAN entries used.
    // -1=unknown.

    UINT MAN[7];                //  List of MANs.

} WW_MBX_FLEXLIST;

//
// OID_WW_MBX_GROUPLIST
//
typedef struct _WW_MBX_GROUPLIST {
    INT count;                  //  Number of MAN entries used.
    // -1=unknown.

    UINT MAN[15];               //  List of MANs.

} WW_MBX_GROUPLIST;

//
// OID_WW_MBX_TRAFFIC_AREA
//
typedef enum _WW_MBX_TRAFFIC_AREA {
    unknown_traffic_area,       // The driver has no information about the current traffic area.
     in_traffic_area,           // Mobile unit has entered a subscribed traffic area.
     in_auth_traffic_area,      // Mobile unit is outside traffic area but is authorized.
     unauth_traffic_area        // Mobile unit is outside traffic area but is un-authorized.
} WW_MBX_TRAFFIC_AREA;
```

```
//
// OID_WW_MBX_LIVE_DIE
//
typedef INT WW_MBX_LIVE_DIE;  //  0 = DIE last received
                                                        //  1 = LIVE last received
                                                        // -1 = unknown
//
// OID_WW_MBX_TEMP_DEFAULTLIST
//

typedef struct _WW_MBX_CHANNEL_PAIR {
    UINT Mobile_Tx;
    UINT Mobile_Rx;
} WW_MBX_CHANNEL_PAIR, *PWW_MBX_CHANNEL_PAIR;
typedef struct _WW_MBX_TEMPDEFAULTLIST {
    UINT Length;
    WW_MBX_CHANNEL_PAIR ChannelPair[1];
} WW_MBX_TEMPDEFAULTLIST, *WW_PMBX_TEMPDEFAULTLIST;

#endif                          // WIRELESS_WAN
#endif                          // _NTDDNDIS_
```

ntddpack.h

```
#ifndef __NTDDPACKET
#define __NTDDPACKET 1
#include "devioctl.h"
/*#include <packon.h> */
struct _PACKET_OID_DATA {
    ULONG Oid;
    ULONG Length;
    UCHAR Data[1];
};

typedef struct _PACKET_OID_DATA PACKET_OID_DATA, *PPACKET_OID_DATA;

/*#include <packoff.h> */
#define FILE_DEVICE_PROTOCOL            0x8000
#define IOCTL_PROTOCOL_QUERY_OID        CTL_CODE(FILE_DEVICE_PROTOCOL, 0 , METHOD_BUFFERED,
FILE_ANY_ACCESS)
#define IOCTL_PROTOCOL_SET_OID          CTL_CODE(FILE_DEVICE_PROTOCOL, 1 , METHOD_BUFFERED,
FILE_ANY_ACCESS)
#define IOCTL_PROTOCOL_STATISTICS       CTL_CODE(FILE_DEVICE_PROTOCOL, 2 , METHOD_BUFFERED,
FILE_ANY_ACCESS)
#define IOCTL_PROTOCOL_RESET            CTL_CODE(FILE_DEVICE_PROTOCOL, 3 , METHOD_BUFFERED,
FILE_ANY_ACCESS)
#define IOCTL_PROTOCOL_READ             CTL_CODE(FILE_DEVICE_PROTOCOL, 4 , METHOD_BUFFERED,
FILE_ANY_ACCESS)
#define IOCTL_PROTOCOL_WRITE            CTL_CODE(FILE_DEVICE_PROTOCOL, 5 , METHOD_BUFFERED,
FILE_ANY_ACCESS)
#define IOCTL_PROTOCOL_MACNAME          CTL_CODE(FILE_DEVICE_PROTOCOL, 6 , METHOD_BUFFERED,
FILE_ANY_ACCESS)
#define IOCTL_OPEN                      CTL_CODE(FILE_DEVICE_PROTOCOL, 7 , METHOD_BUFFERED,
FILE_ANY_ACCESS)
#define IOCTL_CLOSE                     CTL_CODE(FILE_DEVICE_PROTOCOL, 8 , METHOD_BUFFERED,
FILE_ANY_ACCESS)

#endif
```

```
/*
 * Copyright (c) 1999, 2000
 *      Politecnico di Torino.  All rights reserved.
 *packet32.h
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that: (1) source code distributions
 * retain the above copyright notice and this paragraph in its entirety, (2)
 * distributions including binary code include the above copyright notice and
 * this paragraph in its entirety in the documentation or other materials
 * provided with the distribution, and (3) all advertising materials mentioning
 * features or use of this software display the following acknowledgement:
 * ``This product includes software developed by the Politecnico
 * di Torino, and its contributors.'' Neither the name of
 * the University nor the names of its contributors may be used to endorse
 * or promote products derived from this software without specific prior
 * written permission.
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */

#ifndef __PACKET32
#define __PACKET32

#include <ntddpack.h>
#include <ntddndis.h>

#include <winsock.h>

//working modes
#define MODE_CAPT 0
#define MODE_STAT 1

//ioctls
#define pBIOCSETBUFFERSIZE 9592
#define pBIOCSETF 9030
#define pBIOCGSTATS 9031
#define pBIOCSRTIMEOUT 7416
#define pBIOCSMODE 7412
#define pBIOCSWRITEREP 7413
#define pBIOCSMINTOCOPY 7414
#define pBIOCSETOID 2147483648
#define pBIOCQUERYOID 2147483652
#define pATTACHPROCESS 7117
#define pDETACHPROCESS 7118
#define pBIOCEVNAME 7415

#define pBIOCSTIMEZONE 7471

//Network type structure

typedef struct NetType
{
        UINT LinkType;
        UINT LinkSpeed;
}NetType;


// Alignment macros.  Packet_WORDALIGN rounds up to the next
// even multiple of Packet_ALIGNMENT.
#define Packet_ALIGNMENT sizeof(int)
#define Packet_WORDALIGN(x) (((x)+(Packet_ALIGNMENT-1))&~(Packet_ALIGNMENT-1))


//some definitions from libpcap
#ifndef BPF_MAJOR_VERSION

struct bpf_program {
        UINT bf_len;
        struct bpf_insn *bf_insns;
};
struct bpf_insn {
        USHORT code;
        UCHAR  jt;
        UCHAR  jf;
        int k;
};
struct bpf_stat {
        UINT bs_recv;           /* number of packets received */
        UINT bs_drop;           /* number of packets dropped */
```

```
};

struct bpf_hdr {
        struct timeval bh_tstamp;     /* time stamp */
        UINT    bh_caplen;      /* length of captured portion */
        UINT    bh_datalen;     /* original length of packet */
        USHORT          bh_hdrlen;      /* length of bpf header (this struct
                                           plus alignment padding) */
};

#endif

#define         DOSNAMEPREFIX   TEXT("Packet_")
#define         MAX_LINK_NAME_LENGTH    64
#define         NMAX_PACKET 65535

typedef struct _ADAPTER  {
                                        HANDLE hFile;
                            TCHAR  SymbolicLink[MAX_LINK_NAME_LENGTH];
                                        int NumWrites;
                                        HANDLE ReadEvent;
                                        UINT ReadTimeOut;
                                      }  ADAPTER, *LPADAPTER;

typedef struct _PACKET {
                                       HANDLE       hEvent;
                           OVERLAPPED   OverLapped;
                           PVOID        Buffer;
                           UINT         Length;
                                        UINT         ulBytesReceived;
                                        BOOLEAN      bIoComplete;
                                     }  PACKET, *LPPACKET;

#ifdef __cplusplus
extern "C" {
#endif

BOOLEAN PacketSetMinToCopy(LPADAPTER AdapterObject,int nbytes);
BOOLEAN PacketSetNumWrites(LPADAPTER AdapterObject,int nwrites);
BOOLEAN PacketSetMode(LPADAPTER AdapterObject,int mode);
BOOLEAN PacketSetMaxLookaheadsize (LPADAPTER AdapterObject);
BOOLEAN PacketSetReadTimeout(LPADAPTER AdapterObject,int timeout);
BOOLEAN PacketSetBpf(LPADAPTER AdapterObject,struct bpf_program *fp);
BOOLEAN PacketGetStats(LPADAPTER AdapterObject,struct bpf_stat *s);
BOOLEAN PacketSetBuff(LPADAPTER AdapterObject,int dim);
BOOLEAN PacketGetNetType (LPADAPTER AdapterObject,NetType *type);
LPADAPTER PacketOpenAdapter(LPTSTR AdapterName);
BOOLEAN PacketSendPacket(LPADAPTER AdapterObject,LPPACKET pPacket,BOOLEAN Sync);
LPPACKET PacketAllocatePacket(void);
LPPACKET PacketAllocateNPacket(UINT n);
VOID PacketInitPacket(LPPACKET lpPacket,PVOID  Buffer,UINT  Length);
VOID PacketFreePacket(LPPACKET lpPacket);
BOOLEAN PacketResetAdapter(LPADAPTER AdapterObject);
BOOLEAN PacketWaitPacket(LPADAPTER AdapterObject,LPPACKET lpPacket);
BOOLEAN PacketReceiveNPacket(LPADAPTER AdapterObject,LPPACKET headLPacket,UINT n,UINT
length,BYTE* buffer,BOOLEAN Sync);
BOOLEAN PacketReceivePacket(LPADAPTER AdapterObject,LPPACKET lpPacket,BOOLEAN Sync);
VOID PacketCloseAdapter(LPADAPTER lpAdapter);
BOOLEAN PacketSetHwFilter(LPADAPTER AdapterObject,ULONG Filter);
BOOLEAN PacketGetAdapterNames(PTSTR pStr,PULONG  BufferSize);
BOOLEAN PacketGetNetInfo(LPTSTR AdapterName, PULONG netp, PULONG maskp);
BOOLEAN PacketRequest(LPADAPTER  AdapterObject,BOOLEAN Set,PPACKET_OID_DATA  OidData);
VOID PacketSetNextPacket(LPPACKET lpPacket, LPPACKET next);
VOID PacketSetLengthBuffer(LPPACKET lpPacket, UINT dim);
VOID PacketSetLengthPacket(LPPACKET lpPacket, UINT numBytes);
LPPACKET PacketGetNextPacket(LPPACKET lpPacket);

#ifdef __cplusplus
}
#endif

#endif //__PACKET32
```