INDICE

DEDICATORIA 3
AGRADECIMIENTOS 4
RESUMEN 5

1 INTRODUCCION. 7

1.1 ANTECEDENTES....7
1.2 PLANTEAMIENTO DEL PROBLEMA 7
1.3 OBJETIVOS 8
1.3.1 OBJETIVO GENERAL 8
1.3.2 OBJETIVO ESPECIFICO. 9
1.4 ORGANIZACION DEL TRABAJO DE INVESTIGACION. 10
1.5 LISTA DE FIGURAS 11

2. SEGURIDAD INFORMATICA 15

2.1 SEGURIDAD INFORMATICA 15
2.1.1 PUERTOS.....16
2.1.1.1 ESCANEO DE PUERTOS 17
2.1.1.2 INICIANDO UNA CONEXION. 18
2.1.1.3 FINALIZANDO UNA CONEXION 18
2.1.2 TECNICAS DE ESCANEO. 19
2.1.2.1 TCP Connect. 20
2.1.2.2 TCP SYN. 20
2.1.2.3 TCP FIN....21
2.1.2.4 UDP SCAN. 22
2.1.2.5 ACK SCAN. 22
2.1.2.6 NULL SCAN. 23
2.1.2.7 Xmas SCAN. 23
2.1.2.8 SYN/ACK SCAN. 24
2.1.2.9 ZOMBIE SCAN. 24
2.1.3 BARRERAS DE PROTECCION....25
2.1.3.1 FIREWALLS 26
2.1.3.2 IDS....28
2.1.3.3 PROXY....31
2.1.3.3.1 PROXY WEB. 32
2.1.3.3.2 PROXY SERVER....33
2.1.4 ROUTERS Y ENRUTAMIENTO. 34
2.1.5 NAT Y PAT. 37
2.1.6 AUDITORIA DE LA RED 38
2.1.6.1 SPOOFING....38
2.1.6.1.1. SPOOFING ARP. 39
2.1.6.1.2 SPOOFING RARP....40
2.1.6.1.3 SPOOFING ICMP. 41
2.1.6.2 SNIFFER. 41
2.1.6.3 DoS. 42
2.2 SEGURIDAD INALAMBRICA....44
2.2.1 PUNTOS DE ACCESO (AP)....45
2.2.2 PROTOCOLO DE APLICACIONES INALAMBRICAS (WAP)....46
2.2.3 AISLAMIENTO EQUIVALENTE DE REDES INALAMBRICAS (WEP)....47
2.2.4 ACCESO PROTEGIDO A WI-FI (WPA). 49
2.2.5 PROTOCOLO DOMINANTE TEMPORAL DE LA INTEGRIDAD (TKIP). 50
2.2.6 CODIGO DE INTEGRIDAD DEL MENSAJE (MIC)....51
2.3 SEGURIDAD BLUETOOTH. 53

2.3.1 ESTANDAR Y ESPECIFICACIONES DE BLUETOOTH.....54
2.3.2 TOPOLOGIA DE RED BLUETOOTH 55
2.3.3 CAPAS DEL PROTOCOLO BLUETOOTH.....58
2.3.4 AUDITORIA BLUETOOTH.....62
2.3.4.1 BLUESNARF. 63
2.3.4.2 BLUEBUG.....64
2.3.4.3 HELOMOTO. 65
2.3.4.4 BLUELINE. 65

3. SOCKETS DE COMUNICACIONES. 68

3.1 QUE ES EL SOCKET. 68
3.2 TIPOS DE SOCKETS. 69
3.3 ENCAPSULACION DE LOS DATOS. 70
3.4 ESTRUCTURAS DE DATOS DEL SOCKET. 71
3.5 DIRECCIONES IP. 72
3.6 FUNCIONES PARA SOCKETS. 74
3.6.1 FUNCION SOCKET(). 74
3.6.2 FUNCION CLOSE(). 75
2.6.3 FUNCION SHUTDOWN(). 76
3.6.4 FUNCION BIND(). 76
3.6.5 FUNCION LISTEN(). 77
3.6.6 FUNCION ACCEPT(). 79
3.6.7 FUNCION CONNECT(). 81
3.6.8 FUNCION SEND(). 82
3.6.9 FUNCION RECV(). 84
3.6.10 FUNCION SENDTO(). 86
3.6.11 FUNCION RECVFROM(). 87
3.6.12 FUNCION SELECT(). 88
3.6.13 FUNCION GETPEERNAME(). 91
3.6.14 FUNCION GETHOSTNAME(). 94
3.6.15 FUNCION GETDOMAINNAME(). 95
3.6.16 FUNCION GETHOSTID(). 96
3.6.17 FUNCION GETSOCKNAME(). 97

4. PROTOCOLOS DE COMUNICACION. 100

4.1 PROTOCOLO DE INTERNET (IP) 100
4.2 PROTOCOLO DE MENSAJES DE CONTROL EN INTERNET (ICMP). 103
4.3 PROTOCOLO DE DATAGRAMAS DE USUARIOS (UDP). 105
4.4 PROTOCOLO DE CONTROL DE TRANSMICION (TCP). 107
4.5 PROTOCOLO RAW. 110

5. RESULTADOS. 112

5.1 SISTEMA PARA COMUNICACION DE REDES LAN, INALAMBRICAS Y BLUETOOTH. 112
5.2 OBJETIVOS Y REQUERIMIENTOS DEL SISTEMA. 113
5.3 COMUNICACIÓN Y LOGICA DEL SISTEMA. 114
5.4 SERVIDOR Y CLIENTE. 115
5.5 FINES COMERCIALES. 120

REFERENCIAS BIBLIOGRAFICAS. 122

DEDICATORIA

Este proyecto de tesis lo dedico de forma principal y especial a Dios, porque es el quien me ha brindado la paciencia, el conocimiento y la fuerza que día a día me ha brindado para continuar con mi sueño, sin importar los obstáculos que se me han presentado en la vida en formas distintas pero que han tenido el objetivo de dejar mis sueños como algo inalcanzable, gracias a dios todo fue posible.

AGRADECIMIENTOS

A mi amada esposa, mi madre que siempre me ha soportado, a los amigos que me han apoyado con este proyecto, mi director de tesis (Dr. Aldo Luis Mendez Perez) por darme la oportunidad de dirigir mi proyecto, mis asesores (Ing. Luis Antonio Gracia Garza y Dr. Gerardo Romero Galvan) por la paciencia y especialmente a la Universidad Autónoma de Tamaulipas en su Unidad Académica Reynosa-Rodhe por brindarme parte de mis conocimientos los cuales me sirven para emprender proyectos y objetivos futuros.

RESUMEN

La realización de un sistema para administración, monitorización y servicios para redes LAN, Inalámbrica y Bluetooth mediante software basado en Linux.

Cumpliendo las demandas de seguridad más a la vanguardia incorporadas para protección de la información, usuarios y la red en su totalidad, con la finalidad de formar una red interactiva y controlable entre los modos de comunicación ya mencionados. Destinando como punto principal los siguientes aspectos entre redes de comunicación:

- Intercomunicación entre redes LAN, Inalámbrica y Bluetooth mediante un software cliente.
- Intercambio de información de cualquier tipo (audio, video, archivos, imágenes, etc...) entre redes LAN, Inalámbrica y Bluetooth.
- Mensajería y conversaciones en tiempo real entre redes.
- Motorización y control de la información que se puede intercambiar entre redes.
- Auditoria de las redes de forma automatizada o manual, con el fin de mantener una red lo más segura posible.
- Actualizaciones de software mediante una conexión a internet.
- Respaldos y registros estadísticos del estado de la red, la saturación y el retardo.

También tomando en cuenta la disminución de costos en la utilización de servicios en dispositivos móviles y estáticos que utilizan la tecnología bluetooth, lo cual conlleva la incorporación de elementos publicitarios con el fin de brindar al usuario otros beneficios cotidianos por servicio que se le brinde, los cuales pueden ser de igual manera controlables, administrables y ajustables a las demandas y reacciones del usuario.

INTRODUCCION

1

1 INTRODUCCION. 7
1.1 ANTECEDENTES 7
1.2 PLANTEAMIENTO DEL PROBLEMA 7
1.3 OBJETIVOS 8
1.3.1 OBJETIVO GENERAL 8
1.3.2 OBJETIVO ESPECIFICO. 9
1.4 ORGANIZACION DEL TRABAJO DE INVESTIGACION. 10
1.5 LISTA DE FIGURAS 11

1 INTRODUCCION.

1.1 ANTECEDENTES

En los distintos planteles educativos a nivel universitario y empresas a gran escala, la utilización de equipos de cómputo conectados mediante un dispositivo de red (concentrador) son de gran necesidad para el trabajo en equipo y un mejor manejo de la información, ya que se tiene todo en forma compartida por medio de los equipos de la red. Conforme al avance de las tecnologías surgen las redes inalámbricas, las cuales incorporan un punto extra adicional a la redes alámbricas (LAN, Local Area Network), ya que permite tener equipos conectados a la red de forma inalámbrica, brindando una mejor movilidad y facilidad de conexión, sin necesidad de estar frente a un PC Desktop como lo es comúnmente y no solo termina en un PC o un Laptop, sino también ahora el uso de tecnologías móviles pequeñas como son las Palms y Celulares, son de gran utilidad para portar información de forma fácil y rápida, brindado así un dispositivo de uso doble, que sería: comunicación telefónica y almacenador de información, que para esto se da uso de la tecnología Bluetooth, permitiendo el compartimiento y búsqueda de dispositivos Bluetooth en una distancia determinada.

Muchas facilidades se han incorporado con los avances en las redes de comunicaciones, pero también han surgido modos de infiltración los cuales hacen los datos seguros, como algo no seguro y disponible a miradas indiscretas o no deseadas. De forma paralela los dispositivos que utilizan tecnología Bluetooth (celulares, palms, etc.) han quedado fuera de la comunicación para con las redes LAN e inalámbricas.

1.2 PLANTEAMIENTO DEL PROBLEMA

Como se ha mencionado en las redes de comunicación o conexión se experimentan formas de compartimiento, manejo y creación de información, pero no obstante se sigue excluyendo la red Bluetooth como línea básica de comunicación entre PCs – laptops y/o celulares, en la actualidad no existe aplicación gratuita o comercial que implemente o permita el montaje de redes a una mejor interacción, la cual permita a dispositivos Bluetooth, inalámbricos y Ethernet, disfrutar de un ambiente de trabajo seguro y confiable.

Por lo tanto se concluye que es necesario la creación de una aplicación de software para conjugar las redes LAN, inalámbricas con redes Bluetooth, con el fin de aportar una herramienta que incremente la facilidad de trabajo entre equipos de distintas redes y aporte una nueva forma de comunicar dispositivos diferentes a lo que son equipos de cómputo, aprovechando las distintas tecnologías que poseen un medio de comunicación Bluetooth. También es una visión o forma cómoda por software para comunicar redes bluetooth con otras redes, en base a la falta de dispositivos de hardware tipo concentradores que comuniquen redes bluetooth.

A su vez tanto el documento como la aplicación tiene fines de crecimiento futuro en base a los distintos servicios que se pueden implementar entre las distintas redes de comunicaciones, cuestiones de publicidad en servicios masivos con un gran número de usuarios activos, implementación de cuestiones de seguridad para equipos de tecnología pequeña (palms, celulares, etc.) los cuales aun carecen de privacidad mediante el medio bluetooth.

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

Desarrollar una aplicación de red a gran escala que permita acortando y facilitando la comunicación de tecnologías LAN, inalámbrica y Bluetooth, constituido por una aplicación servidor y cliente, cada una trabajando en conjunto para brindar distintos servicios lucrativos y no lucrativos dentro de las redes de comunicaciones. Estableciendo como prioridad principal la preservación de datos, seguridad de la información, fiabilidad de uso y facilidad de manejo del software, intentando establecerse como una aplicación que intente cumplir con las normativas de red establecida en sus distintos protocolos de comunicaciones usados y permita el crecimiento de los servicios para los usuarios.

Para los usuarios una herramienta de trabajo que permita el flujo de información en momentos determinantes de forma rápida, segura y compatible para cualquier tipo de tecnología que posea un dispositivo de comunicación LAN, inalámbrico o Bluetooth. Y respecto a su compatibilidad, también siendo utilizable para los sistemas operativos Windows y sistemas operativos Linux, siendo así un software multiplataforma y disponible para uso entre redes Windows, redes Linux o entre ambas.

1.3.2 OBJETIVO ESPECIFICO.

En base a las redes de comunicaciones que se tienen actualmente y con más auge (LAN, inalámbrico y Bluetooth), que este documento ilustre e incite a un estudio más profundo en las áreas de redes de comunicaciones, pero en un mayor enfoque hacia la programación de la red y como resultado una aplicación de software que proporcione los siguientes servicios:

- Programa Servidor que proporcione panel de control, con las siguientes características:
  - Agregar/modificar/eliminar usuarios con privilegios de uso.
  - Establecer delimitaciones en transferencias de archivos (dimensión).
  - Establecer delimitaciones en tiempos de transferencias repetitivas por PC/usuario (envío simultáneo).
  - Establecer delimitaciones en nombres de equipos (desktop, laptop, palms o celulares).
  - Monitor de red en su totalidad.
  - Monitor de red por tipo de comunicación (LAN, inalámbrico y Bluetooth).
  - Visualización de equipos activos e inactivos (tabla).
  - Visualización de estado de la red por paquetes.
  - Envío de mensajes a equipos visualizados.
  - Escaneo en busca de equipos por tipo de red.
  - Consola de comandos AT para equipos Bluetooth.
  - Manejabilidad remota del servidor.
  - Recuperación de passwords.
  - Chat.

- Programa cliente que proporcione ventana con características especiales:
  - Ventana ligada al inicio de sesión.
  - Listado de equipos conectados a la red.
  - Listado de equipos por tipo de red.
  - Escaneo/búsqueda de equipos por red o en toda la red.
  - Visualización del estado de dispositivos/equipos de la red.
  - Transferencia de archivos entre dispositivos o equipos de la red.
  - Envió de mensajes.
  - Visualización
  - Acceso remoto a equipos.

1.4 ORGANIZACION DEL TRABAJO DE INVESTIGACION.

CAPITULO I – INTRODUCCION.

En este capitulo se brinda una descripción detallada de la finalidad de este documento de tesis, para la realización de una aplicación de software la cual permita la intercomunicación de las redes de comunicación LAN (Local Area Netowork), Inalámbrica y Bluetooth.

CAPITULO II – SEGURIDAD INFORMATICA.

Seguridad Informática.

En este capitulo se haba de forma general sobre las redes de comunicación de área local (LAN, Local Area Network) y se da un enfoque mayor a las cuestiones de auditoria de redes LAN, proporcionando el conocimiento de los métodos de intrusión básicos con fines de preservación de la seguridad LAN.

Seguridad Inalámbrica.

En este capitulo se describe a grandes rasgos los distintos métodos de encriptación de paquetes, brindados por los routers para privatización de la información, también se investigan los algoritmos para encriptación de información.

Seguridad Bluetooth.

En este capitulo se habla de una de las tecnologías de comunicación mas recientes, proporcionando información sobre las distintas problematicas de seguridad que presenta la tecnología Blueooth y sus formas de intrusión con fines de brindar mayor seguridad a los dispositivos que utilicen tecnología bluetooth.

CAPITULO III – SOCKETS DE COMUNICACIÓN.

En este capitulo se mencionan las distintas funciones existentes para implementación en códigos de lenguaje C para programación de aplicaciones para redes de comunicaciones. Se detallan las características que presentan los sockets de comunicaciones, formas de capturar paquetes y modificacion de paquetes de distintos protocolos.

CAPITULO IV – PROTOCOLOS DE COMUNICACIÓN.

En este capitulo se habla de los protocolos de comunicaciones que utilizan las distintas aplicaciones para intercambio de información en redes de comunicaciones LAN, Inalámbricas y Bluetooth. Se describen algunos de los protocolos mas utilizados, conformación del paquete y descripciones de cada una de sus tramas.

CAPITULO V – RESULTADOS.

En este capitulo es el resultado de los demás capítulos estudiados plasmados en una aplicación propuesta, se muestran algunas imágenes de la aplicación, se conocen sus objetivos a fines, funcionamiento y resultados obtenidos.

1.5 LISTA DE FIGURAS

CAPITULO 2.1 SEGURIDAD INFORMATICA
FIGURA 2.1 Estableciendo una sincronización de conexión. . . . 18
FIGURA 2.2 Finalización de una conexión. . . . 18
FIGURA 2.3 Escaneo de tipo TCP Connect. . . . 20
FIGURA 2.4 Escaneo de tipo TCP SYN. . . . 21
FIGURA 2.5 Escaneo de tipo TCP FIN. . . . 21
FIGURA 2.6 Escaneo de tipo UDP SCAN. . . . 22
FIGURA 2.7 Escaneo de tipo ACK SCAN. . . . 23
FIGURA 2.8 Escaneo de tipo NULL SCAN. . . . 24
FIGURA 2.9 Implementación de un firewall, ilustrando trafico protegido y no protegido. . . . 26
FIGURA 2.10 Implementación de Firewall con DMZ. . . . 28
FIGURA 2.11 Implementación de IDS (sistema para detección de intrusos). . . . 29
FIGURA 2.12 Implementación de un IDS basado en Host. . . . 30
FIGURA 2.13 Implementación de IDS basado en Red. . . . 30
FIGURA 2.14 Implementación de IDS Distribuido. . . . 31
FIGURA 2.15 Ilustración de Proxy Web. . . . 33
FIGURA 2.16 Proxy Server con servicio variado. . . . 34
FIGURA 2.17 Descripción de Router D-Link. . . . 35
FIGURA 2.18 lustración de una red con utilización de Router. . . . 36
FIGURA 2.19 Ilustración de Ataque Spoofing ARP “Man in the Middle”. . . . 40

CAPITULO 2.2 SEGURIDAD INALAMBRICA
FIGURA 2.20 Descripción de Router D-Link para inalámbrica. . . . 45
FIGURA 2.21 Pila de Protocolos WAP y WEB. . . . 46
FIGURA 2.22 Servicios a tecnologías móviles mediante internet. . . . 47
FIGURA 2.23 Proceso de cifrado WEP. . . . 48
FIGURA 2.24 Panel de Configuración de Métodos de Cifrado para redes inalámbricas. . . . 48
FIGURA 2.25 Panel de Configuración de Métodos de Cifrado WPA para redes inalámbricas. . . . 49
FIGURA 2.26 Estructura de Encriptación del Protocolo TKIP. . . . 50

CAPITULO 2.3 SEGURIDAD BLUETOOTH
FIGURA 2.27 Logotipo Bluetooth. . . . 53
FIGURA 2.28 Formación del símbolo Bluetooth. . . . 54
FIGURA 2.29 Topología Bluetooth. . . . 56
FIGURA 2.30 Dispositivos de Informática Bluetooth. . . . 56
FIGURA 2.31 Dispositivos de Audio Bluetooth. . . . 57
FIGURA 2.32 Dispositivos de Automóvil Bluetooth. . . . 57
FIGURA 2.33 Dispositivos de Entrada y Salida Bluetooth. . . . 57
FIGURA 2.34 Dispositivos Móviles Bluetooth. . . . 58
FIGURA 2.35 Modelo Visual de una Piconet. . . . 58
FIGURA 2.36 Capas del Protocolo Bluetooth. . . . 59
FIGURA 2.37 Gestor de Enlace. . . . 60
FIGURA 2.38 Clases de Servicios en un paquete bluetooth. . . . 61
FIGURA 2.39 Tabla de algunas empresas que implementan bloques de comandos at. . . . 62
FIGURA 2.40 Archivos del perfil OBEX. . . . 63

CAPITULO 4 PROTOCOLOS DE COMUNICACIÓN
FIGURA 4.1 Esquema de Cabecera IP. . . . 101
FIGURA 4.2 Paquete ICMP. . . . 104
FIGURA 4.3 Paquete UDP. . . . 106
FIGURA 4.4 Paquete TCP. . . . 107

CAPITULO 5 RESULTADOS
FIGURA 5.1 Logotipo Aryax. . . . 112
FIGURA 5.2 Red de Comunicaciones en Aryax. . . . 114

FIGURA 5.3 Presentacion de Aryax. 118
FIGURA 5.4 Panel LAN. 119
FIGURA 5.5 Panel Inalambrico. 119
FIGURA 5.6 Panel Blueooth. 120

SEGURIDAD INFORMATICA

SEGURIDAD INFORMATICA

2

2. SEGURIDAD INFORMATICA 15
2.1 SEGURIDAD INFORMATICA 15
2.1.1 PUERTOS. 16
2.1.1.1 ESCANEO DE PUERTOS 17
2.1.1.2 INICIANDO UNA CONEXION. 18
2.1.1.3 FINALIZANDO UNA CONEXION 18
2.1.2 TECNICAS DE ESCANEO. 19
2.1.2.1 TCP Connect. 20
2.1.2.2 TCP SYN. 20
2.1.2.3 TCP FIN. 21
2.1.2.4 UDP SCAN. 22
2.1.2.5 ACK SCAN. 22
2.1.2.6 NULL SCAN. 23
2.1.2.7 Xmas SCAN. 23
2.1.2.8 SYN/ACK SCAN. 24
2.1.2.9 ZOMBIE SCAN. 24
2.1.3 BARRERAS DE PROTECCION 25
2.1.3.1 FIREWALLS 26
2.1.3.2 IDS 28
2.1.3.3 PROXY 31
2.1.3.3.1 PROXY WEB. 32
2.1.3.3.2 PROXY SERVER 33
2.1.4 ROUTERS Y ENRUTAMIENTO. 34
2.1.5 NAT Y PAT. 37
2.1.6 AUDITORIA DE LA RED 38
2.1.6.1 SPOOFING 38
2.1.6.1.1. SPOOFING ARP. 39
2.1.6.1.2 SPOOFING RARP. 40
2.1.6.1.3 SPOOFING ICMP. 41
2.1.6.2 SNIFFER. 41
2.1.6.3 DoS. 42

2. SEGURIDAD INFORMATICA

2.1 SEGURIDAD INFORMATICA

La seguridad informática es punta principal al iniciar desde un simple proyecto computacional ya sea software o hardware, hasta la implementación de aplicaciones, redes, o cualquier cosa que pueda ser escalón para atentar contra la seguridad.

La información es una de las cuestiones principales que resguarda el tema de seguridad informática, asumiendo que toda información en cualquier cantidad se desee tener a la mano sin importar distancias, esto conlleva a la utilización de equipos de cómputo conectados a Internet, donde pueden ser comprometidos sin una previa revisión del mismo sistema o red.

Para llevar acabo la implementación de las mejores medidas de seguridad, instalaciones premeditadas y aseguradas, es necesario que cada administrador de red no solo se enfoque en conocer las licencias de un software, o bien en conocer el funcionamiento del mismo software, sino que vaya un poco más haya, teniendo conocimiento de las distintas técnicas de intrusión que pueden dejar colapsado un equipo en momentos donde la información es demandada a gran escala o donde sea necesario tener una rapidez de la información dejando como punto principal el conocimiento no solo de técnicas de intrusión, sino también de los distintos protocolos que se utilizan para comunicación entre enrutadores (routers) y protocolos utilizados para comunicación entre las aplicaciones de la red.

En el caso de utilización de motores de código ya programados, es recomendable estar a la vanguardia de las últimas actualizaciones de software que proporcione el creador del mismo, ya que cualquier orificio de seguridad puede ser ocasionado no solo por el sistema operativo, sino que entran en juego las distintas aplicaciones de software con problemas en su código fuente, archivos que se reciben por parte de personas desconocidas o por correo electrónico (e-mail), que posteriormente son ejecutadas con fines personales y terminan siendo códigos fraudulentos y dañinos escondidos en cualquier aplicación, imagen, canciones, etc.

Es necesario tomar las medidas necesarias previas al conocimiento obtenido en el área de seguridad informática, ya que esto permitirá tener consciencia plena de cualquier movimiento realizado en equipos de cómputo, redes de computadoras y hasta servidores que atiende múltiples peticiones y que pueden ser comprometidas para fines de robo de la información.

2.1.1 PUERTOS.

Un puerto es un número de 16 bits que comprende del 0 al 65535, el cual sirve para que las aplicaciones se puedan comunicar e intercambiar información. Existen dos tipos de puertos que son:

1) Los puertos privados o reservados, son los que están establecidos por defecto para un uso definido, el uso definido puede ser una aplicación ya existen con el Sistema Operativo que se utiliza para comunicación entre procesos, otras aplicaciones, protocolos o transportar datos entre capas. Estos puertos comprenden del 0 al 1024.

2) Los puertos libres o sin uso definido, son puertos que comprende aproximadamente del 1205-65535, y están disponibles para uso libre por las demás aplicaciones, o bien pueden ser establecidos manualmente por el usuario para uso en distintas aplicaciones. Los estados de un puerto están indicados en 3 formas, que son:

a) Abierto. Estado que establece un puerto disponible para recepción de datos, los datos pueden ser enviados en distintos formatos o tipos, esto depende del protocolo utilizado en dicho puerto.

b) Cerrado. El puerto está cerrado y por lo tanto los paquetes o información enviada a dicho puerto serán rechazados.

c) Bloqueado o en silencio. El puerto permanecerá activo siempre y cuando la aplicación la habilite para aceptación de paquetes, conexiones o información de protocolos. Mientras que esté desactivado, el puerto rechazara cualquier intento de conexión o envió de información.

Es importante destacar que así como cada máquina puede tener sus puertos en los distintos estados (dependiendo el uso que le dará la aplicación), esta misma máquina debe tener una dirección IP (Internet protocol) la cual le servirá de identificación para poder ser accedida o consultada por otros equipos dentro de la red, dicho de una forma más simple: *"cada máquina poseerá su dirección IP y puerto abierto para intercomunicarse"*, técnicamente cada máquina invocara al equipo de la siguiente manera:

10.10.10.10:2020 :: 10.10.10.10 es la dirección IP y 2020 el numero de puerto.

En sistemas operativos Linux los puertos pueden ser asignados de una forma más rápida, ya que se tiene a disposición los siguientes archivos de configuración:

'/etc/protocols' .
Es donde se guardan los tipos de protocolos a usar, su orden es el siguiente: nombre de protocolo, número de protocolo que le identifica, nombre final del protocolo. De esta forma se tendrá un mejor conocimiento del tipo de protocolo a utilizar en nuestro puerto.

'/etc/inetd.conf'.
En este archivo se configurará de forma detallada el nombre que se establecerá para el programa a ejecutar, así el archivo de configuración *'/etc/protocols'* podrá identificarlo de forma rápida. El orden de configuración de este archivo es el siguiente: nombre de aplicación, flujo de socket a utilizar, protocolo, bandera, usuario que iniciará el programa, dirección donde está el programa.

***'/etc/services'*.**
Archivo principal donde estarán listados los puertos que se tendrán disponibles para utilización por las distintas aplicaciones, es aquí donde se definirá de forma manual los puertos que deseemos usar para nuestros programas o bien de forma personal. De esta forma el sistema podrá asignar la aplicación deseada al puerto establecido y utilizando el protocolo deseado para la comunicación. Su forma de configuración es: nombre de la aplicación, número de puerto, protocolo.

Para sistemas operativos Windows, la configuración de los puertos es de forma más visual, utilizado la ventana de configuración de firewall, en el apartado del panel de control, así mismo los detalles de la configuración se ajustan con simples clicks y la lógica es la misma, consta de: puerto, programa ejecutable .exe, comentario. El tipo de socket y protocolo a utilizar lo resolverá automáticamente el sistema operativo o bien haciéndole una consulta a la aplicación misma que utilizará el puerto.

2.1.1.1 ESCANEO DE PUERTOS

El escaneo de puertos es una forma para comprobar el estado de los puertos de una computadora, dependiendo el tipo de protocolo a utilizar es la técnica de escaneo a utilizar. Es importante mencionar que se debe tener un conocimiento previo de la constitución de cada uno de los paquetes a utilizar en el escaneo, por ejemplo: TCP (Protocolo de Control de Transmisión), UDP (Protocolo de Datagrama de Usuario), ICMP (Protocolo de Mensajes de Control en Internet), etc.

Para conocer el escaneo de puertos es indispensable la forma en que un par de computadoras pueden iniciar una conexión para intercambio de datos, chat, entre otros, dependiendo el tipo de protocolo a utilizar.

Como ejemplo principal se visualiza la forma de iniciación de conexión entre dos computadoras utilizando el protocolo TCP, como veremos en este protocolo se hacen uso de sus banderas (flags) para:

- Iniciar una conexión.
- Empezar a enviar datos.
- Avisar de finalización de conexión.
- Avisar de finalización de conexión forzosa (desconexión del cliente sin previo aviso de bandera).

2.1.1.2 INICIANDO UNA CONEXION.

A un intento de conexión, el cliente envía el bit de Bandera **SYN** activado (1), el cliente recepciona el paquete TCP y visualización el bit en **SYN**, el cual indica que se desea realizar una sincronización de conexión, el servidor contesta al cliente con un nuevo paquete activando las banderas de **SYN** y **ACK** indicando que se acepta la sincronización de la conexión y este mismo es una Respuesta a la petición. El cliente debe contestar a su vez al servidor la respuesta que ha recepcionado los 2 bits (**SYN** y **ACK**), para esto solo emite un paquete TCP con el bit activado en la bandera **ACK**. En este momento ya estamos conectados. Observar la Figura 2.1 como ejemplo del proceso.

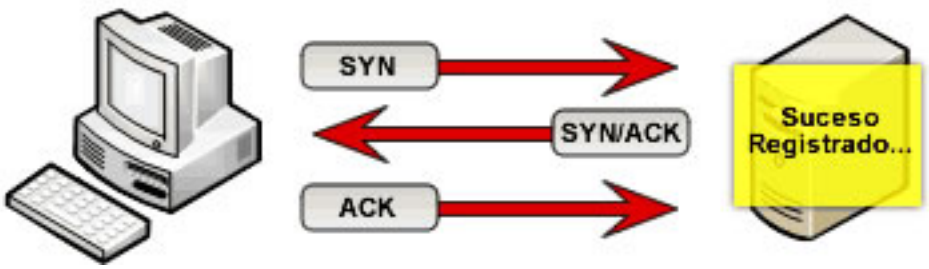

FIGURA 2.1: Estableciendo una sincronización de conexión.

Es importante mencionar que cuando se está intentando sincronizar una conexión a un servidor y enviamos un paquete TCP con el bit activado en bandera **SYN** únicamente, este paquete forzará al servidor asignar un espacio en la pila **TCP/IP** donde estará el número de confirmación almacenado, el servidor esperará algunos segundos a que el cliente le envíe la respuesta de aceptación del paquete, es decir, un paquete TCP con el bit de bandera activado en **ACK** solamente.

2.1.1.3 FINALIZANDO UNA CONEXION

El cliente debe emitir un paquete TCP con el bit activado en la bandera **FIN**, el cual indica el deseo de la finalización de una conexión, el servidor al revivir este paquete responderá con un nuevo paquete TCP con el bit de bandera activado en **ACK** correspondiente a una respuesta exitosa al paquete recepcionado anteriormente (**FIN**), posteriormente el servidor responde con un paquete TCP con el bit activado en bandera **FIN**, a su vez el servidor espera a que el cliente le responda con un paquete TCP con el bit activado en bandera **ACK**, desde este momento la conexión ha finalizado. Observar la representación del proceso en la Figura 2.2.

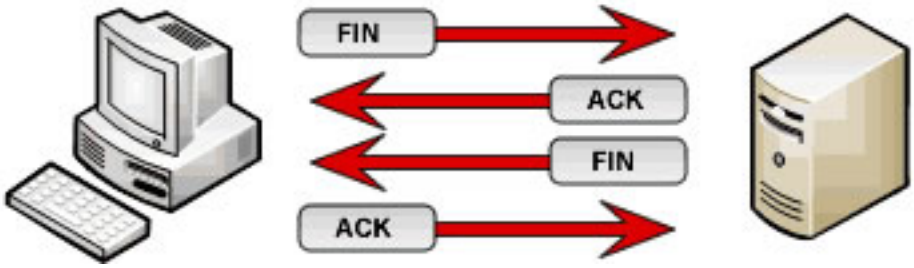

FIGURA 2.2: Finalización de una conexión.

En casos de que se estén transmitiendo datos, y se emita una paquete **FIN**, el receptor emitirá una respuesta **ACK**, esperará la finalización de transacción de datos y posteriormente se emitirá un paquete **FIN** por el receptor y un **ACK** por el emisor.

2.1.2 TECNICAS DE ESCANEO.

Las técnicas de escaneo son métodos para auditar el estado de un puerto en una computadora para posteriormente detectar fallas de funcionamiento con fines de protección de los datos y del programa que utiliza dicho puerto, existen distintas técnicas las cuales se especializan en la manipulación del paquete desde lo más bajo de la red, explicado de una forma más simple: "*manipulación del paquete mediante programación, realizando modificaciones en banderas para observar el comportamiento del servidor"*.

Muchas de estas técnicas son de gran utilidad para comprobar el estado verdadero de un puerto, debido a que muchas aplicaciones utilizan los estados de un puerto de distintas maneras, todas ellas enfocadas en su propia forma de trabajar o de preservación de datos. Es importante destacar este tema, ya que eso no solo puede englobar aplicaciones que utilicen puertos, sino también para otros tipos de cuestiones que pongan en riego la información privada o primordial, como pueden ser: servidores para transacciones de cuentas, información vital para el sistema, transmisión de información importante, etc.

Es imprescindible tener una buena configuración, un buen firewall que cuide los puertos y una aplicación que se ajuste a las necesidades y tenga sus estándares de encriptación para asegurar la información que se manejará.

2.1.2.1 TCP Connect.

Es un modo de escaneo que se logra mediante la llamada a connect() (SYN) en la programación de socket, realizando esta llamada a cada uno de los puertos de una computadora, tomando como respuesta lo siguiente:

- Si se acepta la conexión, entonces el puerto esta abierto.
- Si se retorna un aviso de cierre de conexión (bit **RST**), indica que el puerto está cerrado.
- Si no se tiene respuesta, entonces el puerto está en estado de silencio.

Este es el tipo de escaneo más rápido, ya que se pueden crear tantos sockets se deseen, donde cada socket será un intento de conexión a otra computadora.

La desventaja de esta técnica es que resulta ser muy delatadora o llamativa, ya que llamar a la función connect() indica realizar una conexión completa (véase explicación arriba), por lo cual genera un valor en el registro de la computadora y ésta contendrá nuestros datos de la conexión. La Figura 2.3 muestra el proceso.

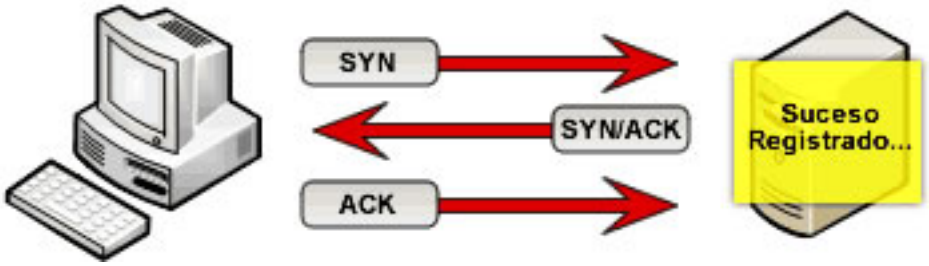

FIGURA 2.3: Escaneo de tipo TCP Connect.

2.1.2.2 TCP SYN.

Esta técnica nos permite escanear los puertos de una computadora sin realizar la conexión por completo, ya que su funcionamiento es:

1- Enviar un paquete SYN para fingir la conexión y espera una respuesta ACK.
2- Si servidor enviara un RST, esto nos indica que el puerto está cerrado.
3- Si el servidor no contesta, entonces el puerto está en modo silencio.
4- Si se obtuvo un paquete SYN/ACK o RST, no se dará una contestación (ACK) como sería lo esperado por el servidor.
5- Enviamos directamente un paquete RST, para que el servidor no complete el intento de conexión y así mismo el servidor no registrará este suceso.

La ventaja es que no genera un registro en la computadora que intentamos conectar, a menos que exista algún firewall o IDS cuidando a dicha computadora, mientras será indetectable. Como punto importante cabe mencionar que para lanzar un escaneo de este tipo se requiere de privilegios de administrador o root ya que usan sockets RAW, además de ser un poco lento. En la Figura 2.4 se ilustra el proceso.

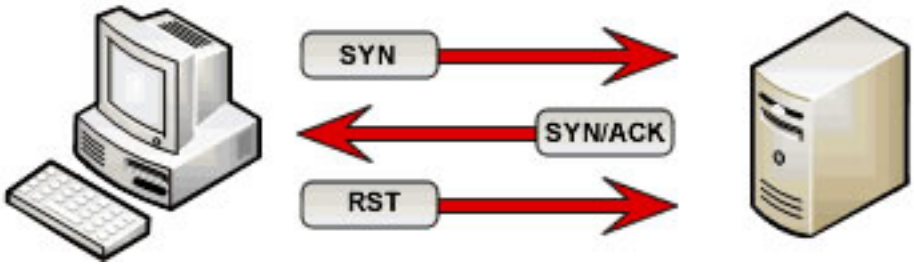

FIGURA 2.4: Escaneo de tipo TCP SYN.

2.1.2.3 TCP FIN.

Es una de las técnicas más sigilosas, ya que se apoya en una particularidad de los estándares de **TCP/IP** que dice:

'Al recibir un paquete FIN en un puerto cerrado, se debe responder con un paquete RST'...

Esta técnica se realiza enviando un paquete **TCP** con el bit de bandera **FIN** activado, a un puerto de una computadora, si éste responde con un paquete **TCP** con el bit de bandera **RST** activado, quiere decir que el puerto esta cerrado, en caso contrario el puerto puede estar abierto o en silencio. Observar la Figura 2.5 para su comprensión.

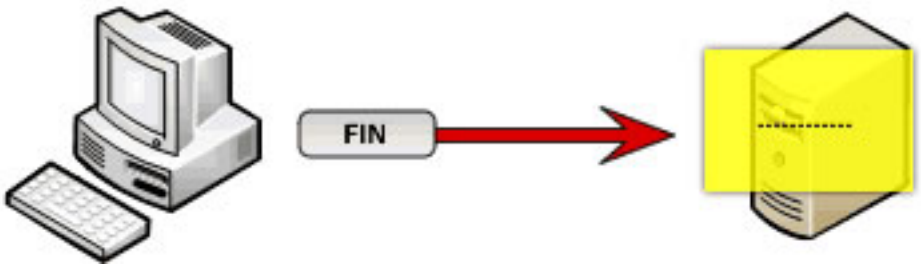

FIGURA 2.5: Escaneo de tipo TCP FIN.

Un punto muy débil de este tipo de técnica es que, cierta compañía de software que ignora en algunas partes los estándares de informática, tal como Microsoft ®, por ejemplo en los sistemas operativos Windows un puerto cerrado ignora los paquetes **FIN**, por lo cual escanear un Sistema Operativo de este tipo nos generara una lista inmensa de puertos abiertos, aunque en realidad estén cerrados o en silencio.

Como ventaja principal es que muchos firewalls no detectan este tipo de escaneo, a no ser algún IDS bien configurado.

2.1.2.4 UDP SCAN.

Esta técnica es mucho más simple, ya que solo consiste en formar un paquete **UDP** vacío y enviarlo al puerto, el servidor contestará con un paquete **UDP de tipo 3** (destino inalcanzable), lo cual indica que el puerto está cerrado, así que en caso de que esté abierto o en silencio, el servidor no responderá. En la Figura 2.6 se muestra el proceso.

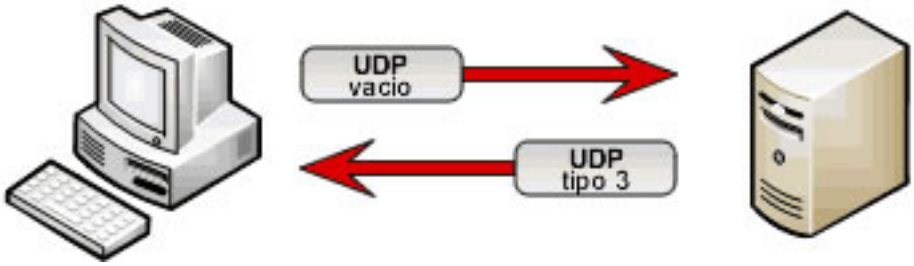

FIGURA 2.6: Escaneo de tipo UDP SCAN.

Esta técnica de escaneo puede ser un poco más lenta, dependiendo el Sistema Operativo que posea la computadora a escanear, según el **RFC 1812-Reqirements for IP Versión 4** recomienda limitar la capacidad de generación de mensajes **ICMP** de error, en base a esto, en sistemas operativos **Linux** (/usr/include/Linux/icmp.h) establece un máximo de 20 mensajes de error por segundo, y en sistemas operativos **Windows** no existe una limitación fijada, así que esta técnica es muy rápida sobre este Sistema Operativo.

2.1.2.5 ACK SCAN.

ACK Scan a diferencia de las demás técnicas, nos permite detectar con exactitud el puerto que se encuentre en modo silencio, utilizando esta técnica como apoyo secundario o utilización de esta misma después de una técnica ya mencionada, resulta ser muy poderosa para determinar si el puerto se encuentra abierto o en silencio. A su vez también puede ser como apoyo para escanear computadoras que se encuentres tras un Firewall de Router, ya que este no acepta intentos de conexión (**SYN**).

Su funcionamiento es basado en el envío de paquetes ACK con números de secuencia y confirmación de forma aleatoria, cuando el puerto reciba este paquete y el puerto esté abierto o cerrado responderá con un paquete **RST**, pero si no se obtiene respuesta entonces el puerto esta en modo silencio. Observar su representación en la Figura 2.7.

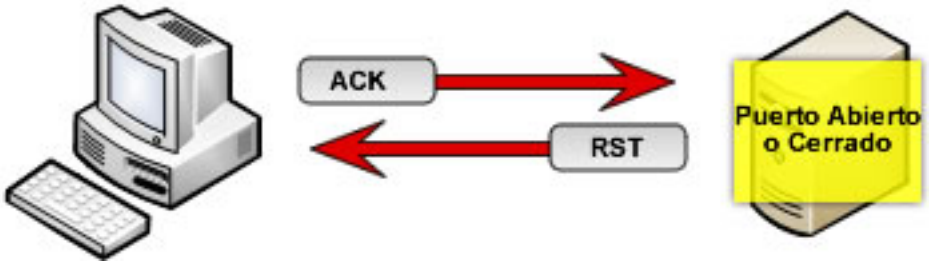

FIGURA 2.7: Escaneo de tipo ACK SCAN.

2.1.2.6 NULL SCAN.

Este escaneo es similar al **TCP FIN**, con la particularidad de que el cliente debe emitir un paquete TCP con todas las banderas desactivadas, así se debe observar lo siguiente del lado del servidor:

- Si el servidor responde con un **RST**, entonces el puerto está cerrado.
- Si no responde, el puerto está abierto o en silencio.

La única ventaja que tiene este escaneo, es que ciertos firewalls vigilan los intentos de conexión (**SYN**) y finalización de conexión (**FIN**), de modo que resulta necesario este escaneo cuando se presente un problema de este tipo. La Figura 2.8 muestra el proceso de una mejor manera.

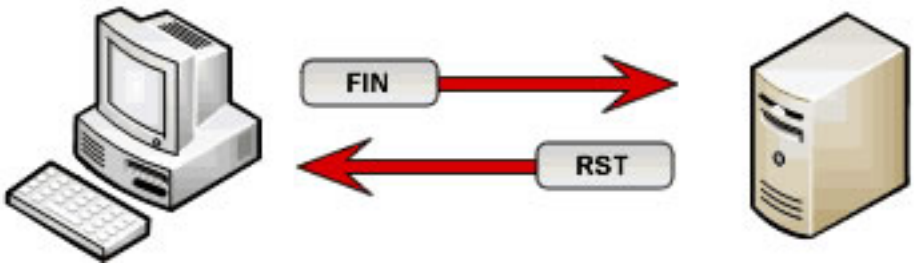

FIGURA 2.8: Escaneo de tipo NULL SCAN.

2.1.2.7 Xmas SCAN.

En este escaneo se debe activar las banderas **FIN/URG/PSH** y esperan como respuesta:

- Si el servidor responde con un **RST**, el puerto está cerrado.
- Si no responde, entonces está abierto o en silencio.

Las desventajas son que el escaneo solo se hace a computadoras que no estén tras un firewall que vigile paquetes **SYN** y **FIN**.

2.1.2.8 SYN/ACK SCAN.

El escaneo SYN/ACK es muy potente frente a equipos que están tras un firewall o IDS sencillo, ya que este paquete solo engaña al servidor avisándole que hubo un error en la transacción (nunca estuvo conectado), así mismo surgen estos efectos:

- Si el servidor responde con un **RST**, el puerto está cerrado.
- Si no responde, el puerto está abierto o en silencio.

2.1.2.9 ZOMBIE SCAN.

Esta técnica se realiza para no ser detectada por el servidor que deseemos escanear, o mejor dicho, que nuestra IP no llegue a ser detectada por el servidor, y a cambio de esto se registre un IP que no es la nuestra. Es necesario contar con un Host Zombie, es decir, una computadora conectada a Internet y que su tráfico sea muy bajo, o en su defecto nulo.

Primero realizamos al Host Zombie un **ping** que nos permita ver el campo **ID** de la cabecera **IP** y el resultado sería:

Código:

```
50 bytes from 10.10.10.10: seq=1 ttl=64 id=+1 win=0 time=96 ms
50 bytes from 10.10.10.10: seq=2 ttl=64 id=+1 win=0 time=88 ms
50 bytes from 10.10.10.10: seq=3 ttl=64 id=+1 win=0 time=92 ms
```

Como se ilustra arriba, el campo **ID** está en 1, podemos decir que el tráfico es nulo.

El siguiente paso es realizar un paquete con el IP falseado (spoofing) que contendrá la IP Origen del Host Zombie, de este modo el intento de conexión se realizará hacia el servidor, con una IP origen del Host Zombie, de modo que surgirán los siguientes efectos:

- Si el servidor envía un SYN/ACK, indica que el puerto está abierto en Servidor, y al recibir este paquete el Host Zombie retornará un paquetes RST, he aquí donde se genera un pequeño trafico de datos.
- Si el servidor envía un RST/ACK, indica que el puerto está cerrado y al recibir este paquete el Host Zombie, no realiza ninguna contestación, es decir que el trafico sigue nulo.
- Si el servidor no realiza contestación, entonces el puerto estará en silencio, y el Host Zombie estará aún con tráfico nulo.

Este modo de operar es un estándar establecido dentro de la implementación de la pila TCP/IP.

A continuación vemos de nuevo el estado de ID, en el ping que mantuvimos constante después de enviar el paquete falseado.

Código:

```
50 bytes from 10.10.10.10: seq=10 ttl=64 id=+1 win=0 timw=96 ms
50 bytes from 10.10.10.10: seq=11 ttl=64 id=+2 win=0 timw=80 ms
50 bytes from 10.10.10.10: seq=12 ttl=64 id=+3 win=0 timw=92 ms
50 bytes from 10.10.10.10: seq=13 ttl=64 id=+2 win=0 timw=96 ms
50 bytes from 10.10.10.10: seq=14 ttl=64 id=+1 win=0 timw=80 ms
50 bytes from 10.10.10.10: seq=15 ttl=64 id=+1 win=0 timw=92 ms
```

Como se detalla, el puerto del servidor está abierto, ya que el Host Zombie (10.10.10.10) tuvo un incremento en el campo **ID**, esto indica que hubo una interacción entre el Servidor y el Host Zombie. En caso que estuviese cerrado o en silencio el puerto del servidor, el estado ID de Host Zombie (10.10.10.10) sería 1.

2.1.3 BARRERAS DE PROTECCION

Las barreras de protección son las que nos brindan la seguridad mediante software o hardware, éstas trabajan cubriendo cierta área de privacidad o enfocadas a resguardar la información y buscar patrones maliciosos que lleguen a corromper la seguridad del sistema.

Existen barreras de protección enfocadas a cada tipo de problemática en cuestiones de seguridad, en redes de cualquier tipo se maneja como protocolo base por lo general el modelo TCP/IP, ya que la comunicación dentro de la red es segmentada y pasada por un flujo, en donde dicho flujo tiene su propio estándar de protocolo, dependiendo el tipo de flujo, es el tipo de protocolo a utilizar. Los flujos TCP/IP (son los más utilizados) viajan por la red y pueden transportar información de distintos tipos, esta información pueden ser desde: archivos segmentados, comandos, órdenes para gestión de red, passwords, firmas digitales, datos encriptados, entre otros. Esta información viaja a través de un puerto en específico, donde otra aplicación espera en ese mismo puerto para recibir dichos datos y realizar una actividad.

Tanto los paquetes, como los puertos juegan un papel importante en la comunicación entre redes, ya que el medio principal para comunicación es un puerto y éste se encuentra disponible a transportar información sin ninguna limitación, a su vez se tienen dos grandes desventajas que pueden quebrantar la seguridad en una red entera o un equipo, y éstas serían:

1- Los puertos transportan información que puede haber sido falseada, además cada aplicación puede abrir un puerto sin dar aviso al usuario.

2- Las aplicaciones que abren puertos para comunicarse, pueden tener errores de programación que al ejecutar algún comando, línea o un simple exceso de información

(desbordamiento de buffer), pueden dejar un hueco en el sistema para poder realizar operaciones ilegales en un sistema de cómputo o una red entera.

Así que tanto las aplicaciones como los puertos son parte importante a vigilar para cerrar la entrada a intrusos en la red, he aquí donde aplicaciones de software o hardware cumplen el objetivo de monitorear, revisar, analizar, tapar, controlar y alertar sobre cuestiones que lleguen a comprometer a un sistema o red.

Estas aplicaciones de software o hardware son los firewalls, IDSs y proxys, donde cada uno cumple una tarea distinta y dependiendo el tamaño de una red, es la forma en que se implementa, de una forma más simple, es a libre decisión si se implementa en equipos particulares o en equipos que realizan otras actividades en la red. Cada tipo de barrera de protección cumple una tarea específica o trabajan en cooperación mutua para mantener una red segura donde la información sea privada y no exista el mínimo error en la inseguridad de la red.

2.1.3.1 FIREWALLS

Los firewalls son barreras que actúan principalmente como centinelas de seguridad, regulando el tráfico entre los puertos estableciendo reglas, las cuales son: aceptar tráfico en puertos determinados, ignorar tráfico en puertos establecidos, redirigir el tráfico que llegue a puertos establecidos y restringir servicios que los usuarios deseen acceder mediante la red. Un firewall proporciona pocas ventajas de seguridad para redes corporativas/empresariales, ya que solo actúa como una alarma para el tráfico de la red, generando registros de información sobre sucesos ocurridos en los puertos.

Estos firewalls están conformados como tipo software o hardware, visto de una mejor forma un firewall puede ser un router, una computadora que mantiene la red (servidor) o una computadora en específico (host), ambas con el fin de mantener una información al día sobre los movimientos o sucesos obtenidos en la comunicación entre equipos mediante sus puertos. Observar la Figura 2.9 para una mejor comprensión.

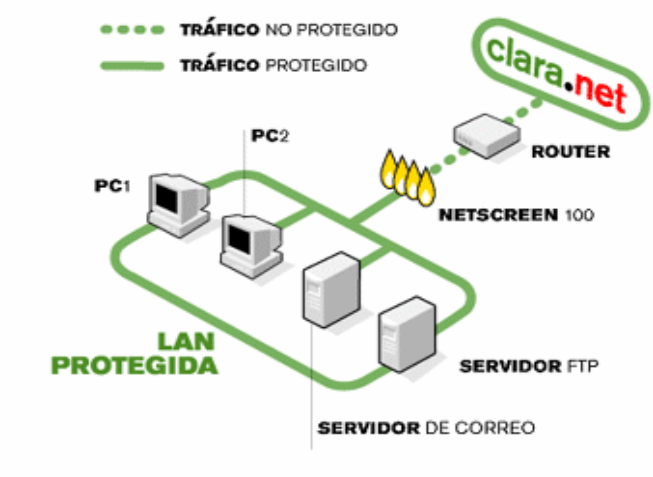

FIGURA 2.9: Implementación de un firewall, ilustrando trafico protegido y no protegido.

Es importante conocer algunos conceptos que se manejan dentro de las configuraciones de firewalls, donde se puede implementar tres tipos sobre una red, las cuales son:

1) Zona Desmilitarizada. Es la configuración de un firewall con sus respectivas denegaciones de servicios para los usuarios, y estableciendo un conjunto de redes o un solo host que tendrá privilegiado el tráfico libre, el cual actuará como un buffer de recepción para no comprometer la Intranet. De este modo se establecerá una zona desmilitarizada o libre para paquetes de entrada y salida en un host o red específica.

2) Host Bastion. Es un equipo de computadora que se sitúa entre el router de la red corporativa e Internet, este host bastion recibirá los paquetes de entrada y examinará sus puertos de destino, consultando con sus reglas de redireccionamiento tomará las decisiones necesarias para aceptar, denegar o redirigir el tráfico entre la Intranet e Internet.

3) Gateway o Puerta de Enlace. Es un router el cual se encarga de aceptar, denegar y redireccionar el trafico de la red, también posee sus directivas para denegación de servicios en puertos específicos El router como ventaja principal lo posee integrado ya por defecto.

Para implementar el firewall se puede hacer en tres formas:

1) Firewall de red basado en host. Es un equipo que actuará como firewall entre la Intranet e Internet.

2) Firewall basado en host. Es un firewall que actuará por equipos, y estará instalado en cada equipo. Ejm: ZoneAlarm, Kerio, etc.

3) Firewall basado en routers. Es el firewall del router.

4) Firewall de equipos. Es el firewall interno que tienen los dispositivos de hardware y son configurables mediante el mismo sistema operativo o bien el software que proporciona el fabricante al instalar el controlador.

Es recomendable posicionar un firewall en el modo que más convenga para tenerlo como centinela para nuestros paquetes de entrada y salida, obteniendo una mejor seguridad en los puertos que deseemos cuidar o deseemos ocupar para nuestras aplicaciones, ya que manteniendo bloqueadas las entradas por puertos se delimitaran a los usuarios a utilizar solamente las aplicaciones que utilicen dichos puertos asignados en el firewall de: router, equipos, host o la red mediante un host. Teniendo en cuenta que un firewall no será punto principal para preservar a un 99% la seguridad, sino que se estarán resguardando los puertos, vigilando los paquetes y evitando miradas o intrusiones indiscretas, ya que un firewall no podrá detectar ni atacar a los que estén intentando acceder ilegalmente mediante los puertos abiertos, de este modo resultaría inútil un firewall. Observar la Figura 2.10.

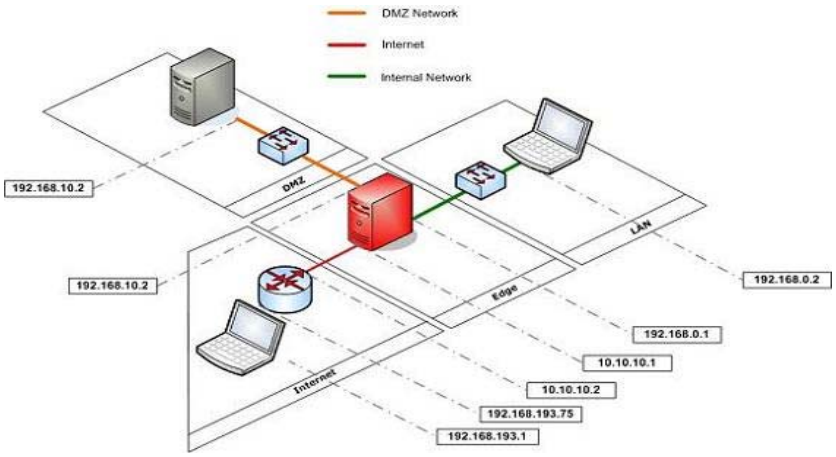

FIGURA 2.10: Implementación de Firewall con DMZ.

2.1.3.2 IDS

Un IDS como sus siglas indican, es un sistema para detección de intrusos. Los IDS son una variante de los firewalls, los cuales sirven como refuerzo para los paquetes que entran por los puertos abiertos o permitidos por el firewall, asumiendo que dichos paquetes entrantes tienen código o configuraciones inyectadas en el paquete, los cuales pueden realizar efectos dañinos al software que se encuentra recibiendo los paquetes, o bien pueden ser códigos que permitan abrir agujeros de seguridad (bugs) para permitir la entrada ilegal o vista indiscreta, dejando así al sistema indefenso y comprometido. A continuación en la Figura 2.11 se aprecia un ejemplo.

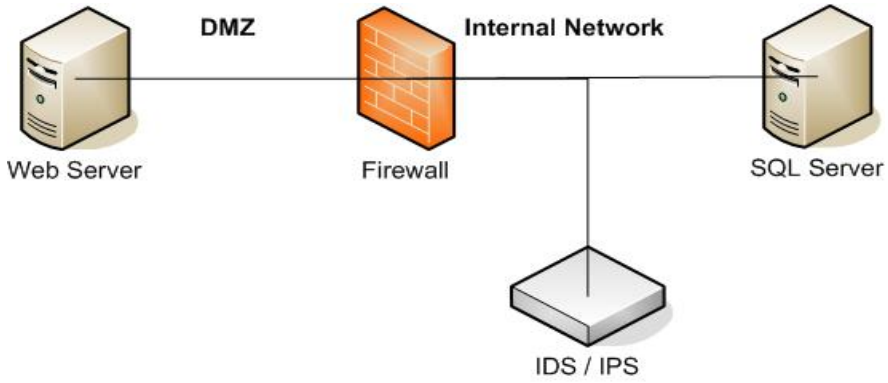

FIGURA 2.11: Implementación de IDS (sistema para detección de intrusos).

La tarea de un IDS es detectar patrones sospechosos o códigos maliciosos en los paquetes que están circulando por la red o bien entrando a un host, en base a estos patrones manda un alerta y realiza una acción para suprimir el problema, posteriormente realizar la protección del equipo lo cual puede llegar a desconectar de la red el equipo o bien la red entera, esto dependiendo de la magnitud del problema, también contrarresta el ataque y da aviso detallado sobre las huellas o ubicación del intruso para tomar medidas legales a dicha intrusión ilegal.

Para que un IDS realice todas las actividades mencionadas, éste posee un modo de configuración basado por reglas (parecido al firewall) donde estas reglas indican los patrones que deben ser cumplidos en paquetes y si encontrase patrones distintos se le indica la acción a realizar, posteriormente se generará un registro detallado de las anomalías detectadas en el transcurso del día o conforme se desee revisar el registro, un IDS también posee el modo de aviso como alerta, para evitar estar visualizando el registro por sucesos sospechosos, el IDS alertará sobre los problemas que encuentre conforme sucedan.

Algunas ventajas que ofrece IDS se enuncian a continuación:

- Asegura la red mediante un sofisticado mecanismo basado por reglas, patrones de comportamiento y alertas.
- Detección de intrusos y contraataque a estos mismos.
- Exploración de puertos en busca de aperturas no establecidas previamente por el usuario o administrador del sistema.
- Alertas sobre desbordamientos de memorias surgidos repentinamente ocasionado por el software mismo instalado y que puede ser puerta de entrada para intrusos.
- Los ataques detectados son registrados y congelados, esto puede llegar hasta desconectar el equipo, los equipos o la red entera, dependiendo como lo determine el IDS.

Como se puede observar los IDS son sistemas sofisticados que pueden realizar búsquedas indeterminadas sobre anomalías y brindar servicios de seguridad, privacidad y detección de forma confiable, es por eso que los IDS se dividen en tres tipos:

1) HIDS :: IDS en host. Es un IDS que se encuentra vigilando a un único equipo de cómputo, es decir que solo vigilará y analizará los paquetes que van dirigidos y que provengan de ese mismo equipo. La Figura 2.12 representa un ejemplo.

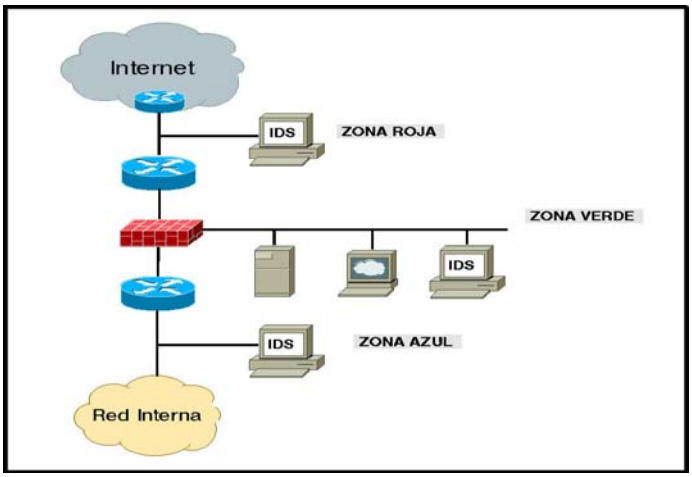

FIGURA 2.12: Implementación de un IDS basado en Host.

2) NIDS :: IDS en red. Es un IDS que por defecto es el servidor y se encontrará vigilando todo el segmento de la red en busca de patrones distintos para procesar un análisis en todos los paquetes que viajan hacia los equipos o provenientes de ellos mismos. Observar la Figura 2.13 para su comprensión.

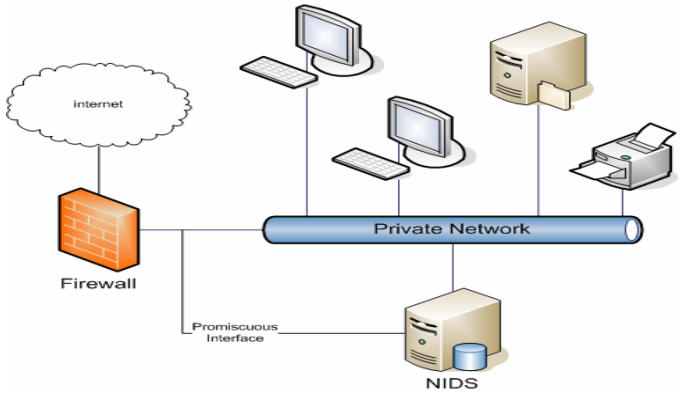

FIGURA 2.13: Implementación de IDS basado en Red.

3) DIDS :: IDS Distribuido. En particular es una de las mejores formas de montar un IDS, más amplio, más confiable y ajustable a las distintas necesidades que surgen en la empresa/corporativo, ya que este tipo de IDS está conformado por un sistema Cliente-Servidor. Esto se lleva acabo asignando equipos de cómputo únicamente para trabajar como IDS cliente en distintas partes de la red (en caso que sea muy grande), posteriormente estableciendo un equipo de cómputo como base principal (servidor) donde se almacenarán

todas las alertas, acciones anormales, detecciones, registros, etc. Cada uno de los clientes se encargará de vigilar sus equipos o su segmento de red, y cada suceso visualizado es volcado hacia la base central que contendrá toda la información sobre los demás IDSs. Este método se acomoda también para redes del tipo VPN (Virtual Private Network, Red Privada Virtual)

En conclusión se recomienda montar sistemas de firewall por router o por host, para proteger los puertos y solo establecer los puertos que se van a utilizar, posteriormente optar por la implementación de algún tipo de IDS que más convenga para tener mejor resguardada la red, evitando que la información sea usurpada o se vea en peligro de ser observada por personas no deseadas. Observar la Figura 2.14.

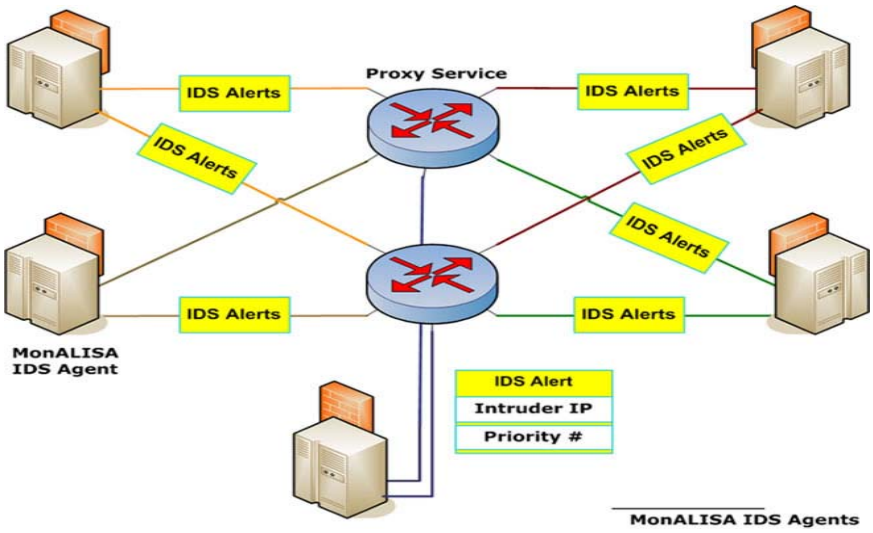

FIGURA 2.14: Implementación de IDS Distribuido.

2.1.3.3 PROXY

Es un servicio que brinda una computadora, ya sea cliente o servidor (como se desee implementar) con el fin de interponerse entre nosotros y la computadora a la que deseemos conectarnos, mandar información o visitar, a su vez se puede establecer que sirve como pasarela para llegar a un lugar. En la actualidad un proxy no solo puede servir como pasarela, sino que puede ser un buen firewall que se interpone entre nosotros y puede tomar una decisión (dependiendo su configuración) para poder hacernos llegar al destino o no, y lo mismo viceversa.

Existen dos tipos de proxys, los proxys que se especializan en recibir y contestar peticiones del tipo web y del tipo información, que sirven para transportar información de un

sitio a otro. Esto con sus respectivas limitaciones que vendría conformando una forma de filtro para saber que información podrá pasar por alto y cual no.

A diferencia de un navegador normal, los proxys utilizan un puerto adicional (distinto al 80, puerto de Internet), por donde se manejarán para pasar información y determinan la forma en que se contestará, claro está, dependiendo el tipo de servicio que se proporcione por el proxy.

Un Proxy es una buena barrera de protección a nivel website (enviar páginas web) y nivel acceso a recursos compartidos de forma anónima o utilizando una sola dirección IP como salida.

2.1.3.3.1 PROXY WEB.

Los proxy web son aplicaciones que proporcionan servicio de filtraje de sitios web, son utilizados para poder filtrar cierto contenido permitido por la configuración que se le haya dado al servidor proxy web. El filtro actúa conforme a un conjunto de cadenas de caracteres que deseemos omitir, en dado caso nos regula el tipo de páginas que se podrán visualizar y se delimita al usuario el acceso a sitios web restringidos, o bien, que el proxy web tenga configurado apto para el usuario.

Un proxy web tiene un uso mayor bajo instituciones educativas donde se proporciona acceso a internet de forma “libre”, de esta forma se controla el acceso a las páginas los alumnos entran, poniendo reglas donde no se permitan contenidos del tipo pornográfico, hacking, contenido no educativo o cualquier cosa que genere tráfico inútil en la red.

La configuración de un proxy no es solo para delimitar contenido dentro de una consulta a sitios web, sino también genera registros de acceso a sitios donde es casi imposible obtener información sobre sus contenidos, es decir que es imposible realizar las comparaciones de los filtros/reglas establecidas previamente. En base a ese registro el administrador puede visualizar las páginas con mayor acceso por los usuarios y poder tomar una decisión determinante conforme al contenido que se muestra en dicho sitio web con mayor acceso.

El funcionamiento de un proxy web es el siguiente:

1- El usuario pide al servidor una consulta de visualización de página web.
2- El servidor realiza la consulta al sitio web y espera que se le devuelva como respuesta dicha página.
3- El servidor pasa por un filtro el contenido de la página que se le dio como respuesta y toma una decisión
4- Si el contenido fue aceptado por el servidor, la página es visualizada. En caso contrario se muestra el mensaje que se configure, este puede ser: “Página no encontrada” o “Página censurada”.

Como forma dinámica, el servidor proxy web también resguarda las cookies sobre consultas a paginas web ya realizadas por otros usuarios de la red, en caso que otro usuario consulte el mismo web que otro, el servidor proxy web no perderá tiempo ni recursos de la red para enviar una petición de pagina web, sino que de forma automática le envía al usuario el contenido del sitio web o en su defecto la censuracion de dicha pagina. Este método es muy recomendable para evitar saturación o uso inútil de la red cuando los usuarios desean realizar peticiones o consultas a sitios web iguales.

Por el lado de la privacidad y seguridad, un proxy web no solo sirve tanto para filtrar, delimitar, sino que a su vez puede estar configurado de dos maneras:

a) Anónimo. El servidor proxy web enviará peticiones de páginas web y ocultará la dirección IP de la máquina cliente que realizó la consulta al servidor proxy y este a su vez al sitio web. Este método resguarda la identidad (dirección IP) del cliente.

b) No anónimo. El servidor proxy web enviará la petición de la página web y cuando el sitio web desee obtener información de la máquina que lo está consultando, el servidor proxy web contestará con los datos de la dirección del cliente que realizó la petición original. Este método tiene la desventaja de no resguardar la privacidad del cliente.

Tanto el modo anónimo y no anónimo, proporcionan un servicio rápido y confiable cuando el Servidor proxy web tiene con anterioridad el contenido de dicho web solicitado. La Figura 2.15 muestra un mejor ejemplo.

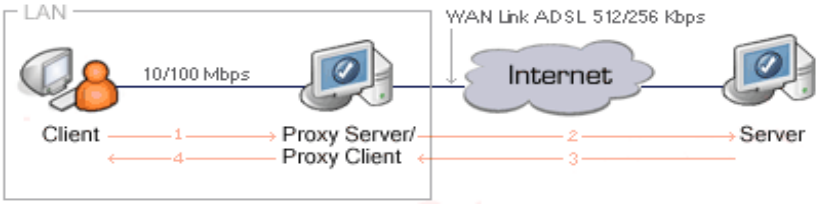

FIGURA 2.15: Ilustracion de Porxy Web.

2.1.3.3.2 PROXY SERVER

Los proxy server son de forma parecida al proxy Web, solo que el proxy Web se enfoca al servicio de página web, los proxy server no solo pueden alimentar el servicio de páginas web, sino que también sirven para proporcionar más recursos a los que se deseen acceder o conectar otro cliente de la red, de esta forma poder realizar una respuesta más rápida a un recurso ya solicitado con anterioridad.

Los proxy server son intermediario entre el cliente e Internet, y sirven para cuando dentro de la red solo se cuenta con una dirección a Internet para salir, y todas las aplicaciones

o recursos que requiere obtener están en una computadora fuera de la compañía o bien dentro de ella, el servidor proxy puede alimentar las peticiones y permitir acceso a Internet a los equipos con una sola conexión

Las ventajas de un servidor proxy es que puede limitar los accesos a Internet a los clientes deseados y solo dar acceso a los que se configuren aptos para este servicio, así mismo se puede denegar la ejecución de aplicaciones que requieren conectar a otros servidores, ejemplo claro es: MSN Messenger, un proxy server puede denegar el acceso hacia Internet a aplicaciones previamente establecidas.

El ahorro de trabajo de carga en los clientes disminuye, ya que se puede utilizar como almacenador de datos, los cuales van a ser llamados por la aplicación directamente al proxy server. Esto sin ocupar recursos del cliente y dejando más recursos para trabajar con otras aplicaciones.

Así como el proxy server realice las tareas más pesadas, esto puede ser contrario a lo que se desea obtener, ya que el exceso de trabajo realizado, o bien la carga de peticiones en la red y procesamiento de datos pueden disminuir el rendimiento del mismo proxy server. La Figura 2.16 ilustra la explicación.

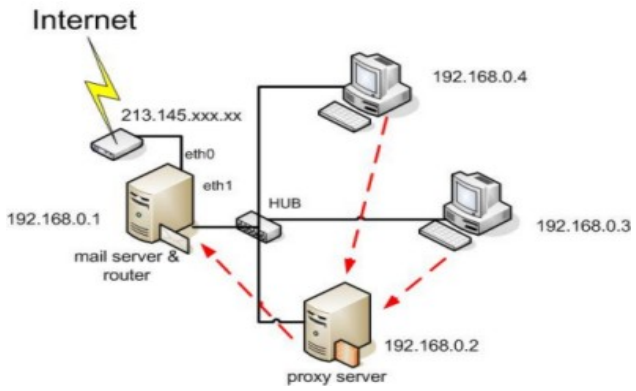

FIGURA 2.16: Proxy Server con servicio variado.

2.1.4 Routers y enrutamiento.

El router es un dispositivo de hardware utilizado para la comunicación entre: routers y/o hosts. A diferencia de los otros dispositivos de hardware para redes el router posee las siguientes características:

- Entrada WAN para el servicio de Internet y posteriormente los equipos conectados serán capaces de acceder a dicho servicio, esto también dependiendo de la configuración que

tenga el router.

- Entrada LAN, consiste en un número específico de puertos ethernet para la conexión de los clientes.
- Panel de configuración del router, el administrador podrá configurar o limitar los servicios a como sea necesario.
- Firewall incluido, la mayoría de los routers poseen su propio firewall en donde se puede configurar equipos de forma particular o bien zonas desmilitarizadas (DMZ).
- Posee una tabla de enrutamiento, mostrando los equipos que se encuentran activos o pasivos en la red.
- Log de conexiones y desconexiones como estadísticas para un mayor conocimiento sobre equipos que están trabajando o acceden al router de una u otra forma.
- Configuraciones de administrador para implementar puentes de red hacia otros puntos.

En la Figura 2.17 se puede apreciar un router para un mejor conocimeitno.

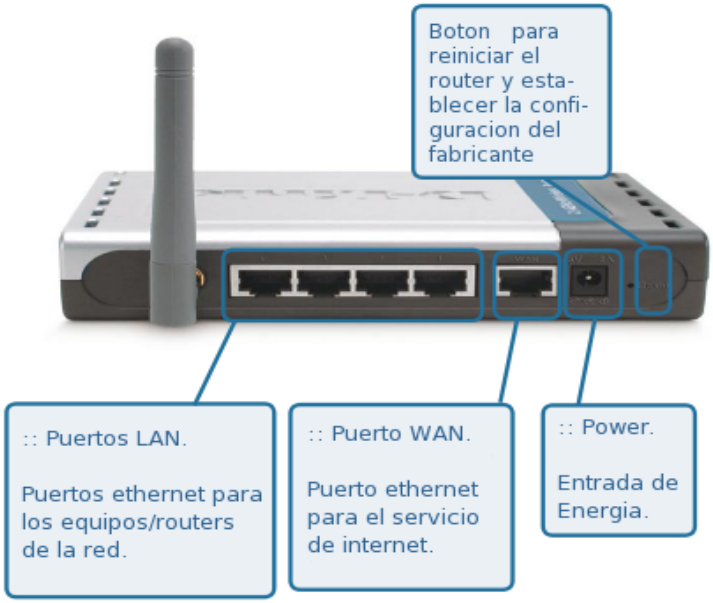

FIGURA 2.17: Descripción de Router D-Link.

El enrutamiento es la forma en que se transportará un paquete de un nodo a otro dentro o fuera de la red. La tarea principal es tomar el paquete a enrutar, encapsularlo dentro del router utilizando el protocolo de ruteo que se requiera y realizar el envío del paquete al equipo o bien al router siguiente.

El fin de la comunicación entre routers es para actualizar sus tablas de direcciones según los equipos que se tenga en cada LAN, de este modo utilizando algún protocolo de enrutamiento se realizan los envíos de paquetes de forma más directa, tratando de evitar contratiempos o peticiones de búsqueda en otros puntos de la red.

Existen dos tipos de enrutamiento para lograr acabo la actualización de tablas de ruteo entre routers, estos son:

1) Ruteo estático. El administrador debe actualizar su propia lista de direcciones cada vez que la topología de la red o los equipos conectados al routers cambien, así mismo el modo estático especifica que los equipos se conectarán mediante una dirección IP estática, la cual debe ser establecida a su vez por el administrador. Esto resulta muy difícil si la red fuese muy amplia y estuviera en crecimiento constante, pero es una buena técnica para preservar la seguridad y saber el conocimiento exacto de equipos conectados y que se pueden conectar a la red, en base a esto poder realizar análisis de fallos en busca de problemas dentro de la red.

2) Ruteo dinámico. El administrador configura el router para que actualice su tabla de enrutamiento de forma automática, de este modo los equipos pueden obtener su dirección IP de forma automática (dhcp) sin problema alguno. Este método es muy eficaz y evita contratiempos en caso que la red estuviese en crecimiento constante, pero reduce el rendimiento de la red ya que el router realizará trabajo extra al realizar actualizaciones constantemente.

Una vez especificada la forma de enrutamiento se continúa con la configuración del tipo de protocolo para el ruteo de paquetes en la red. Los tipos de protocolos para el enrutamiento son:

- **RIP ::** *Protocolo de Ruteamiento por Vector Distancia.* Determina la dirección (vector) y la distancia a cualquier enlace de la red.

- **IGRP ::** *Protocolo de Ruteamiento de Gateway Interior.* Protocolo de enrutamiento patentado por Cisco.

- **OSPF ::** *Protocolo de Ruteamiento de Estado del Enlace.* Obtención de la ruta más corta, recreando la topología de la red e implementando un algoritmo para la obtención de la ruta más corta.

- **EIGRP ::** *Protocolo de Ruteamiento de Gateway Interior Mejorado.* Combina aspectos de RIP y OSPF para determinar el envío del paquete, también patentado por Cisco.

En la Figura 2.18 se ilustra un ejemplo.

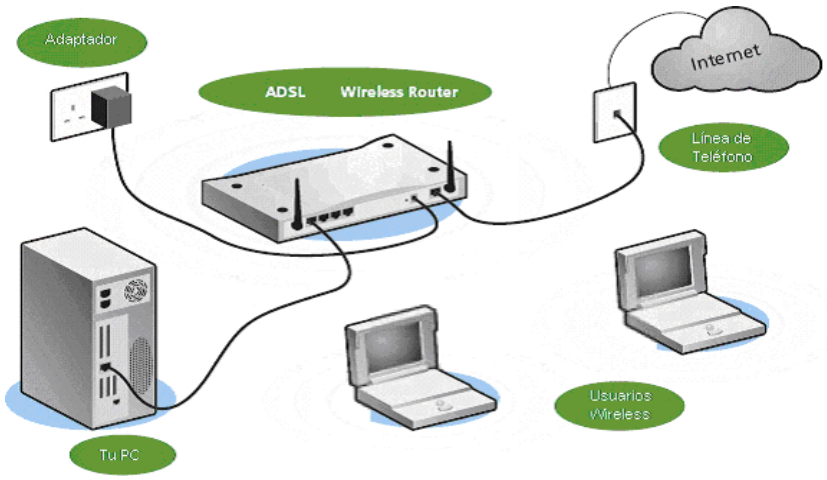

FIGURA 2.18: lustración de una red con utilización de Router.

2.1.5 NAT Y PAT.

La NAT (Network Adress Traduction) que significa traducción de direcciones de red, permite el enmascaramiento de las direcciones IP bajo una única dirección IP, que sería la dirección asignada por el proveedor de servicio de Internet (ISP, Proveedor del Servicio de Internet). Esto es de gran utilidad ya que permite salir a Internet con una única IP pública y tener la IP de la red Internet segura.

A continuación un ejemplo de 2 computadoras realizando una búsqueda en google en la Tabla 1:

IP Interna	Puerto Origen	IP Publica	Consultando a:	Datos
PC1 - 192.168.1.100	1010	200.68.100.78	google.com	????
PC2 - 192.168.1.101	1011	200.68.100.78	google.com	????

En el ejemplo se visualiza que tanto el equipo PC1 y PC2 realizan una misma consulta a google.com, enviando su consulta por el puerto 1010 y 1011 respectivamente, el servidor de google recibe dos consultas de una misma dirección IP (200.68.100.78), y revisando la tabla NAT visualiza que son consultas realizadas por distintos puertos, el servidor de google.com contesta a la petición por el puerto donde se origino actualmente la consulta y de este modo varios equipos pueden realizar consultas al mismo servidor sin que este se confunda.

PAT (Port Adress Traduction) que significa traducción de direcciones de puerto, este se utiliza cuando más de un equipo realiza una consulta al mismo servidor, brinda la traducción del puerto a otro (cambiando el puerto de recepción), envía la consulta por un puerto y espera respuesta por otro puerto especificado en la tabla NAT, de esta forma se sigue preservando la dirección Interna del equipo, se realiza su petición y tanto el servidor como el router podrán dirigir los paquetes de un sitio a otro.

A continuación un ejemplo de implementación de PAT, cuando ambos equipos realizan consultas a un mismo servidor utilizando un mismo puerto.

IP Interna	Puerto Origen	PAT	IP Publica	Consultando a:	Datos
PC1 - 192.168.1.100	1010	1010	200.68.100.78	google.com	????
PC2 - 192.168.1.101	1010	1011	200.68.100.78	google.com	????

Al realizar la consulta y generar el paquete, el router realiza una revisión de la tabla NAT y se toma con la generación de dos paquetes que tienen un mismo puerto de origen y que van a una misma dirección IP (google.com), es aquí donde el router implementa PAT sobre la NAT, para traducir el puerto a un nuevo puerto, esto lo realiza modificando el paquete directamente y cambiando su puerta de origen a una nueva que encuentre disponible, este suceso se registra en su tabla NAT y envía la consulta.

2.1.6 AUDITORIA DE LA RED

En toda red de comunicaciones se requiere una total seguridad, privacidad y confiabilidad de los datos, bien sea datos existentes en una computadora, o que se encuentren viajando por la red, estableciendo así un sistema de cómputo o una red, casi segura. Cada administrador de red debe tener en cuenta que la seguridad no se basa principalmente a nivel de sistema operativo, sino que influyen varios factores los cuales pueden poner en riesgo el equipo de cómputo, la red o parte de ella, esto recae principalmente en las aplicaciones de software que se tiene trabajando a nivel de sistema operativo, es decir aplicaciones fuera del alcance o administración del sistema operativo, he aquí principalmente donde recaen los problemas de seguridad o también llamados *'bugs'*.

Para tomar las medidas primordiales para brindar seguridad en la red, es necesario conocer las virtudes y debilidades tanto del sistema operativo, software que se tenga trabajando, protocolos que se utilicen para comunicación, protocolos que se utilicen para ruteo, técnicas de escaneo, formas de falseo para visualizar el comportamiento que se tiene en el equipo comprometido (en forma de prueba) o bien la red entera.

Estos conocimientos forman parte de la auditoría de la red, posicionándose así como una forma de comprobar la estabilidad, fiabilidad y rendimiento de la red.

Para iniciar con la auditoría de la red es primordial conocer de manera intermedia el funcionamiento de los distintos protocolos de ruteamiento que pueden ser objetivo principal para ataques iniciales, captura de paquetes e intervenciones a equipos o redes, las cuales son sostenidas, encaminadas y reconocidas por los distintos Routers establecidos en distintos puntos de la red.

2.1.6.1 SPOOFING

Consiste en falsear información de paquetes, inyectando información al paquete formado o paquetes capturados con el fin de engañar a un objetivo (víctima), la técnica de spoofing tiene a su vez fines interesantes para auditar la capacidad de seguridad que posee un servidor. Ya que utilizando alguna técnica de spoofing se puede comprobar la fidelidad de la red o alguna IDS (sistema de detección de intrusos).

Las distintas técnicas de spoofing se basan en los protocolos que más se suelen utilizar en la red, en una comunicación host a host, host a router y de router a router. Estos protocolos suelen ser ARP (protocolo de resolución de direcciones), ICMP (protocolo de mensajes de control de Internet), RARP (protocolo de resolución de direcciones inversa).

A continuación se explican los protocolos para un mejor entendimiento en las técnicas de spoofing:

a) ARP (Protocolo de resolución de direcciones). *"Establece las direcciones IP y MAC de la estación destino para y se almacenan en la tabla ARP para ser utilizada en la encapsulación de datos y envíos a dicho destino"*[1]. El Protocolo ARP consiste en establecer la dirección IP y dirección MAC dentro de la tabla ARP para realizar envíos de información a dicha dirección establecida en la tabla ARP, en caso que no se tenga una dirección dentro de la tabla ARP el dispositivo de red enviará un paquete de petición de ARP para descubrir la dirección MAC hacia donde se encapsularán los paquetes y serán enviados. Al enviar dicho paquete de petición ARP, todos los equipos de la red recibirán esta petición y responderá solo el equipo destino que se le está requiriendo su dirección MAC.

b) RARP (Protocolo de resolución de direcciones inversa). *"Protocolo que se utiliza para resolver la dirección IP a partir de una dirección MAC"* [4]. Este protocolo tiene un funcionamiento parecido a ARP, solo que la diferencia es que resuelve la dirección IP apartir de una dirección MAC, de este modo se puede realizar búsquedas de direcciones IP teniendo a mano las direcciones MAC de equipos de la red. Como desventajas, todas las peticiones de difusión para resolver direcciones IP, se realizan mediante el envío de direcciones MAC, pero estas son enviadas directamente a un servidor y no a los routers ni equipos de la red, por lo tanto es necesario contar con un servidor RARP para obtener este servicio [8].

c) ICMP (Protocolo de control de mensajes de Internet). *"Se transportan en datagramas y se utilizan para enviar mensajes de error y de control"* [1]. Los paquetes ICMP son utilizados de forma constante y excesiva por toda la red, ya que son muy rápidos y proporcionan información rápida sobre el suceso a alguna actividad, suspensión de algún servicio, error en algún proceso, desconexión, etc. Los distintos mensajes ICMP están descritos en la sección *4.2 Protocolo de Mensajes de Control en Internet (ICMP)*, se explica tanto los tipos de mensajes y su estructura para fines de programación. Por mencionar algún tipo de mensaje importante, los ICMP se suelen utilizar cuando algún equipo está fuera de la red, en alguna interrupción o al realizar ping a algún equipo, la dirección de algún router no se localiza, petición de cambio de dirección, los mensajes ICMP de este tipo proporcionan un mensaje siguiente: *"Destino Inalcanzable", "Eco", "Respuesta de Eco", "Redireccionar", "Apagado del Origen", "Respuesta de Información", etc.*

2.1.6.1.1. SPOOFING ARP.

Este tipo de falsificación (spoofing) es el que nos lleva a ser un intermediario entre los datos de una computadora origen y una computadora destino, es decir que toda información que se envíen entre ellos pasará primero por nosotros, es lo que con lleva como muchos dicen: *"Hombre dentro del Medio" (Man in the middle).*

Para conocer esta técnica es necesario saber los pasos que realiza una computadora para enviar un paquete a un destino bajo el protocolos de ruteo y Ethernet, sabiendo que el protocolo ARP es el encargado de traducir la dirección MAC apartir de la dirección IP que se tenga y Ethernet es un protocolo basado únicamente en direcciones MAC. A continuación una explicación muy simple:

- La computadora origen envía una petición ARP-Request a la dirección Broadcast pidiendo la MAC de la dirección IP que tenemos para enviarle los datos.
- El router recibe la petición ARP-Request y se la envía a todas las computadoras.
- La computadora poseedora de la dirección IP responderá con un paquete ARP-Replay, la cual tendrá su dirección MAC.
- Posteriormente tanto routers como hosts generan una tabla ARP haciendo relación IP-MAC para envíos de paquetes de forma futura.
- Finalmente cuando se desee enviar un paquete a un equipo que ya este registrado en nuestra tabla ARP, se realiza el envío empaquetando la dirección mac.

Conociendo la forma establecida como trabaja el protocolo ARP y sus tablas ARP en routers y/o host's, las máquinas atacantes pueden hacer envíos masivos de paquetes ARP-Replay, estableciendo su propia MAC como dirección valida y segura para envío de paquetes, de esta forma se logra posicionar entre el atacante y la máquina destino. La Figura 2.19 ilustra la explicación.

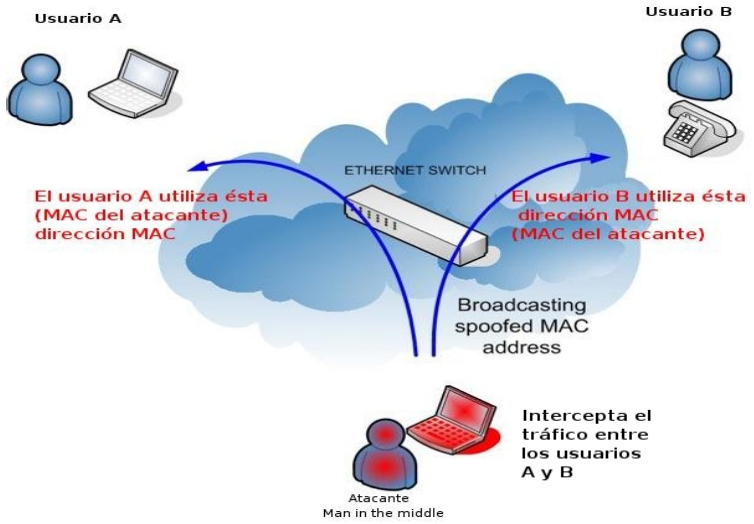

FIGURA 2.19: Ilustración de Ataque Spoofing ARP “Man inthe Middle”.

2.1.6.1.2 SPOOFING RARP.

De la misma forma que el spoofing ARP, RARP es un protocolo que se basa en la traducción de direcciones IP a partir de una dirección MAC conocida, pero con la desventaja que es necesario el montaje de un servidor RARP para implementación de este protocolo.

Por lo tanto un atacante puede hacer mal utilización del protocolo rarp para realizar spoofing RARP con solo enviar paquetes mal formados RARP especificando su dirección IP, de esta forma la tabla RARP se llenará de esta dirección IP y la tomará como una dirección IP

fiable y confiable, por lo tanto el tráfico se verá redireccionado a la computadora del atacante. A partir de ahí el atacante spoofer puede decidir realizar el sniffeo de los datos y reenviar los datos a su origen verdadero o no hacerlo.

2.1.6.1.3 SPOOFING ICMP.

El protocolo ICMP es un protocolo para control de errores e información sobre consultas y respuestas a distintas actividades o sucesos en la red, una de las actividades que sería muy interesante observar es cuando el router realiza los cambios de ruta en su dirección para que los equipos puedan enviar sus paquetes, en este momento de forma automática el router puede hacer envío de paquetes ICMP para que los equipos de la red actualicen su tabla ARP y obtengan la dirección de la nueva ruta hacia dónde enviarán sus paquetes. En este proceso el ruter envía una petición de cambio de ruta, en este momento la dirección máquina realiza sus cambios de ruta de forma normal.

El atacante para realizar spoofing ICMP, realiza un paquete ICMP mal formado con información falsa, en donde le avisa a un equipo de la red sobre el cambio de ruta para envío de paquetes, como si el mismo equipo atacante fuese el router, el host realiza los cambios en su tabla y todo el trafico que salga del host sera enviado al atacante.

2.1.6.2 SNIFFER.

Sniffer es un derivado de snif (oler), que es una peculiaridad de los animales por oler cosas, o seguir el rastro de algo mediante su olfato, teniendo en cuenta que puede olfatear cualquier cosa todo esto delimitado por la capacidad de su nariz. En término informático sniffer se denomina al enfoque de la visualización de paquetes que transitan en la red para poder observar y tomar alguno de ellos (equivalente a disparar), posteriormente el tráfico sniffeado (capturado) puede ser desmantelado y observado cuidadosamente, así mismo capturar más hasta formar el paquete original.

Para realizar sniffer es necesario tener en cuenta 2 cosas muy importantes:

1- Para realizar sniffing se puede establecer la tarjeta de red o cualquier dispositivo de red en modo promiscuo, de esta forma se puede capturar el tráfico de la red. De forma contraria a esto, algunos fabricantes impiden el modo promiscuo a algunos dispositivos, dejando casi imposible el sniffeo de paquetes.

2- Para realizar sniffing se puede utilizar la programación de sockets, los cuales dan acceso al nivel más bajo para la captura de tráfico mediante funciones más amigables y permite implementar estas funciones en aplicaciones visuales las cuales pueden ser como una forma

de detección de tramas para fines de información de datos sospechosos o bien comprobación de la red.

Un sniffer no solo es una forma de atentar la privacidad de la información que viaja por la red, sino también es una forma educativa de como viaja la información, como se forma un paquete, como se desarma un paquete, pensando así en la realización de aplicaciones más dedicadas a la seguridad, como puede ser un sistema de detección de intrusos, que analice los paquetes en busca de patrones o reglas sospechosas previamente establecidas por el usuario y nos ayude a determinar en dado caso la intrusión de código o falseo de direcciones con fines de robo de la información [9].

2.1.6.3 DoS.

DoS o denegación de servicios (Denied Of Services) consiste en atacar los puertos de un servidor o computadora, saturándolos de flujos de información y ocupando de forma rápida todo su ancho de banda, acortando los recursos y servicio del servidor mismo, ya que mediante este ataque el servidor no tendrá el espacio ni el tiempo necesario de atender a los usuarios (prestarles el servicio que proporcione) de este modo se llega acabo la denegación de un servicio.

Los ataques DoS son una herramienta muy útil para comprobar la capacidad de conexiones que puede soportar un servidor, brindando una forma estadística de la cantidad de flujo que puede permitir y hasta que nivel de pueden ver colapsados los distintos servicios que brinde dicho servidor.

Los DoS's tienen como principal objetivo iniciar una sincronización de conexión con un servidor a un determinado servicio que proporcione, posteriormente esta petición formará parte de la pila TCP/IP del servidor (ocupando desde aquí un espacio), posteriormente el servidor esperará a la respuesta del usuario (sobre aceptación de la conexión), el servidor puede esperar de 1, 2 o 3 minutos a que el usuario responda, en caso que se sobrepase el tiempo, el servidor rechaza la conexión y libera el espacio de la pila. Mediante aplicaciones maliciosas o de auditoría de red, se pueden falsear (spoofear) las dirección IP de origen, haciendo múltiples intentos de conexión desde un mismo equipos, así mismo el servidor esperar las respuestas de direcciones falseadas (spoofeadas), esto realizado en mayor escala llega a saturar el ancho de banda y colapsar los servicios que brinda el servidor.

Existe un ataque en mejor capacidad llamado DDoS, Denegación de Servicios Distribuido, es igual que el DoS pero desde este punto el usuario auditor (o atacante) utiliza varios equipos remotos para realizar múltiples intentos de conexiones utilizando direcciones falseadas (spoofeadas) desde cada uno de los equipos [10].

SEGURIDAD INFORMATICA

2

2.2 SEGURIDAD INALAMBRICA 44
2.2.1 PUNTOS DE ACCESO (AP) 45
2.2.2 PROTOCOLO DE APLICACIONES INALAMBRICAS (WAP) 46
2.2.3 AISLAMIENTO EQUIVALENTE DE REDES INALAMBRICAS (WEP). 47
2.2.4 ACCESO PROTEGIDO A WI-FI (WPA). 49
2.2.5 PROTOCOLO DOMINANTE TEMPORAL DE LA INTEGRIDAD (TKIP). 50
2.2.6 CODIGO DE INTEGRIDAD DEL MENSAJE (MIC) 51

2.2 SEGURIDAD INALAMBRICA

La seguridad inalámbrica es un punto importante que merece la atención debida, teniendo en cuenta que la utilización de equipos portátiles crean la necesidad de establecer un método de comunicación entre la red local (LAN) y la red inalambrica, ya sea por cuestiones de trabajo, de diversión o personal.

Para la estructuración o implantación de una red inalámbrica es necesario considerar un concentrador para ser el punto de acceso (AP) y la implementación de un método de seguridad para tener un canal seguro en la comunicación, posteriormente un método de encriptación para proteger la información que viajará por el canal y así evitar miradas indiscretas a la información, equipos y posibles intentos de intrusión a las redes que se tengan interconectadas.

La seguridad inalámbrica se basa principalmente en los métodos de encriptación de la información utilizando algoritmos para resguardar los datos. Estos algoritmos son implementados por el método de seguridad utilizado (en este documento se describe el método de encriptación TKIP). Algunos algoritmos son:

- TKIP (Temporal Key Integrity Protocol). Genera claves temporales para evitar intrusiones mediante re-envío de mensajes y anida un código de integridad del mensaje para proteger la información y determinar su fidelidad.
- CCMP (Counter-Mode/Cipher Block Chaining Message Authentication Code Protocol).
- WRAP (Wireless Robust Authenticated Protocol).

Los métodos de seguridad para afianzar el canal son los encargados de asegurar la conexión entre el punto de acceso (AP) y el cliente conectado, de esta forma se previene de usuarios indeseados o que no estén dentro de la red inalámbrica, se mantiene una sincronización segura, los datos son confidenciales y están fuera del alcance de los intrusos y en caso que la red inalámbrica esté conectada a la red local, se mantiene la seguridad entre ambas redes. Los métodos de seguridad son:

- WEP (Wires Equivalent Privacy).
- WPA (Wi-Fi Privacy Access). Es una mejor de WEP, estableciendo un vector inicial de 48 bits, incorporación de encriptación por algoritmo TKIP. Pero WEP se puede implementar en dos modalidades que son:
  - WPA-PSK (Wi-Fi Privacy Access-Pre-Shared Key). Estableciendo una Clave como método de acceso, pero siguiendo con los mismos procesos de encriptación de WPA. Este método es para facilitar el método de implementación de WPA.
  - WPA-RADIUS (Wi-Fi Privacy Access – RADIUS). Este método es más complicado de implementar ya que requiere de un servidor RADIUS el cual establecerá claves de acceso entre los AP y sus usuarios, de esta forma se tendrá una red inalámbrica aún más protegida, segura y administrable. Los métodos de encriptación son los establecidos en WPA, más los métodos de claves generadas por el servidor RADIUS y el usuario.

De esta forma combinando los métodos de encriptación que se tienen soportados y la seguridad que se implementara en la red inalámbrica, se tendrá un mejor resguardo de la

información, otras redes locales y la tranquilidad que se estará trabajando en una conexión segura donde los datos o cualquier cosa que se transfiera estará asegurada, así que una buena elección y un buen montaje de seguridad en redes inalámbricas serán incapie para alcanzar una mejor forma de trabajo sin robo de información o perdida de confianza para desarrollo de proyectos privados.

2.2.1 PUNTOS DE ACCESO (AP)

Los Puntos de Acceso (AP-Access Points) son solo uno o varios dispositivos de hardware con capacidad de brindar servicio para redes inalámbricas. Estos dispositivos son concentradores de red como switch, hub o routers que permiten la intercomunicación entre redes, teniendo como principal característica una o un par de antenas que son las que se encargarán de enviar/recibir las ondas en alguno de los canales que posee o se tenga configurado en el concentrador.

Los puntos de acceso aparte de intercomunicar redes inalámbricas, también pueden comunicar la red LAN, ya que los concentradores inalámbricos también poseen entrada ethernet para formar redes LAN, y así poder tener en constante comunicación ambas redes inalámbricas y LAN.

Existen delimitaciones en las redes inalámbricas, y éstas se basan principalmente en que un punto de acceso tiene un ancho determinado para la comunicación de las redes inalámbricas, de este modo el envío masivo de información o cualquier actividad que ocupe un ancho de banda, disminuirá el rendimiento y en algunos casos la misma conectividad del punto de acceso con el equipo inalámbrico. Otro de los factores más comunes es la distancia del punto de acceso, este solo puede abarcar una distancia determinada, la cual está especificada o se rige mediante la potencia que posea la antena del concentrador [17]. La Figura 2.20 muestra un concentrador inalámbrico.

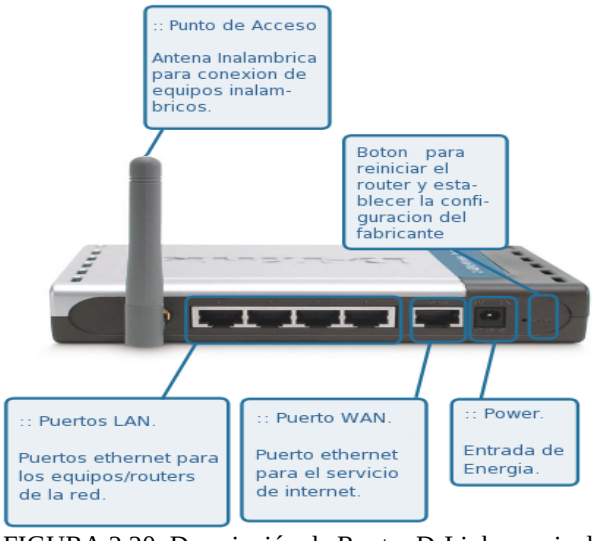

FIGURA 2.20: Descripción de Router D-Link para inalámbrica.

2.2.2 PROTOCOLO DE APLICACIONES INALAMBRICAS (WAP)

El protocolo de aplicaciones inalámbricas (WAP-Wireless Application Protocol) es un estándar abierto internacional para aplicaciones que utilizan las comunicaciones inalámbricas, por ejemplo los accesos a Internet desde un teléfono móvil [5].

La tecnología WAP consiste en un entorno de aplicación y una pila de protocolos para aplicaciones y servicios accesible mediante terminales móviles (por ejemplo los celulares), esto permite que cualquier terminal móvil pueda tener acceso a servicios de manera limitante, ya que la tecnología telefónica ofrece grandes ventajas en sus equipos como serían: almacenamiento de información, pequeña cantidad de RAM para procesamiento, caché interna, todas sus características de manera limitada ya que esto se deriva por el tamaño y peso del equipo. Es por eso que las tecnologías WAP permiten estandarizar los equipos y servicios para brindar a cualquier celular la mejor de las prestaciones conforme sean las capacidades del mismo equipo telefónico. Observar la Figura 2.21.

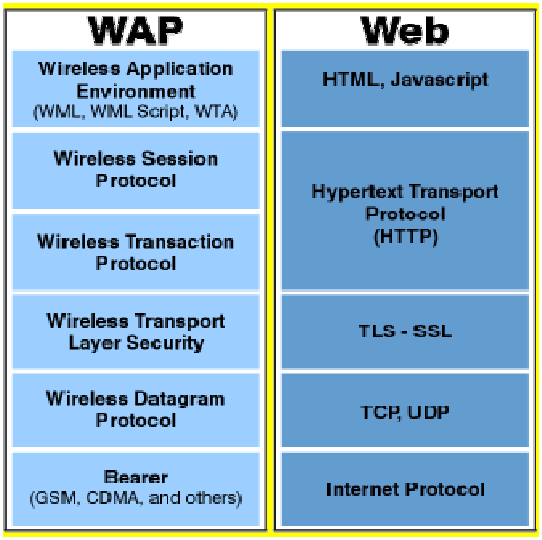

FIGURA 2.21: Pila de Protocolos WAP y WEB.

WAP en su versión 1 definida en 1999, mostró al mundo su lenguaje en el que se estandarizaba para proporcionar los distintos servicios a tecnologías móviles, éste tuvo como nombre WML (Wireless Markup Lenguage), pero a pesar de esto aún existían puntos débiles a cubrir en vista que el estándar WAP y su lenguaje WML no eran totalmente compatibles con Internet debido al tipo de servicios que puede proporcionar un servidor con una potencia variable contra un equipo móvil con capacidades distintas.

Los avances en la implementación de algoritmos en la pila de WAP han ido avanzando hasta el año 2004 cuando surge WAP 2.0, siendo una reingeniería de WAP versión 1, utilizando XHTML-MP (Mobile Profile) como lenguaje de presentación de contenidos, incorpora mejoras al soporte de gráficos y en cuanto a los protocolos usados en la capa de transporte se utiliza TCP y en la aplicación HTTP, alcanzando a cubrir la mayoría de los protocolos usados en Internet. La Figura 2.22 ilustra la explicación.

FIGURA 2.22: Servicios a tecnologías móviles mediante internet.

2.2.3 AISLAMIENTO EQUIVALENTE DE REDES INALAMBRICAS (WEP).

WEP como sus siglas dicen Wireless Equivalent Privacy (Aislamiento Equivalente de Redes Inalámbricas) es el sistema de cifrado incluido en el estándar IEEE 802.11 como protocolo para redes inalámbricas que permite cifrar la información que se transmite [6].

WEP es un método de cifrado de la información utilizando cifrado a nivel 2, se encuentra basado en el algoritmo de cifrado RC4 el cual implementa claves de 64 bits (40 bits más 24 bits del vector de iniciación IV) o de 124 bits (104 bits más 20 bits del vector de iniciación IV). Su principal funcionamiento consiste en dos pasos cifrar las tramas antes de ser enviadas al punto de acceso (AP) apoyándose en el algoritmo RC4 y posteriormente en el algoritmo de chequeo de integridad del mensaje para comprobación de tramas al ser recibidas.

El funcionamiento del algoritmo RC4 se apoya en el flujo de las semillas (seed), donde dicha semilla es un número aleatorio generado para cifrar los mensajes realizando operaciones XOR, el único inconveniente que posee este tipo de cifrado o algoritmo, es que se debe tener una semilla (número aleatorio) distinta para cada mensaje a cifrar, cabe mencionar que el paquete entero es cifrado con la clave WEP [18]. Observar la Figura 2.23 como ejemplo.

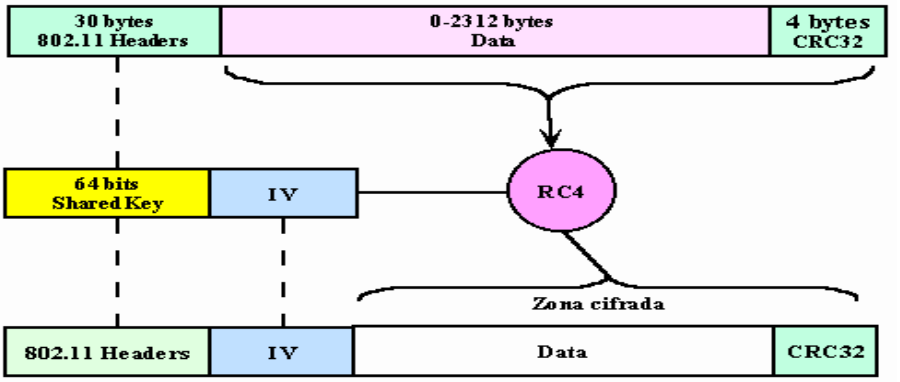

FIGURA 2.23: Proceso de cifrado WEP.

Actualmente el cifrado WEP es uno de los más utilizados en redes inalámbricas como protección a nuestra red, pero la realidad es que el cifrado WEP es uno de los más fáciles de crackear o saltar, ya que utiliza la misma clave para cifrar todos los mensajes, por lo tanto cualquier router con cifrado WEP puede ser atentado y poder gozar de los servicios que proporcione la red, por ejemplo: Internet. El proceso de audición de una red inalámbrica con cifrado WEP se realiza utilizando la aplicación aircrack [19]. De forma general el modo más básico para hacerse de una clave WEP es realizando dos pasos que son: Sniffing de Packetes y utilizar algún WEP-Cracker, es decir se deben capturar algunos paquetes de la red inalámbrica (no es necesario estar conectado o tener un IP), posteriormente se procede a desempaquetar la información para intentar sacar la clave WEP, entre más grande sea la clave WEP, mayor deben de ser los paquetes interceptados. Observar Figura 2.24.

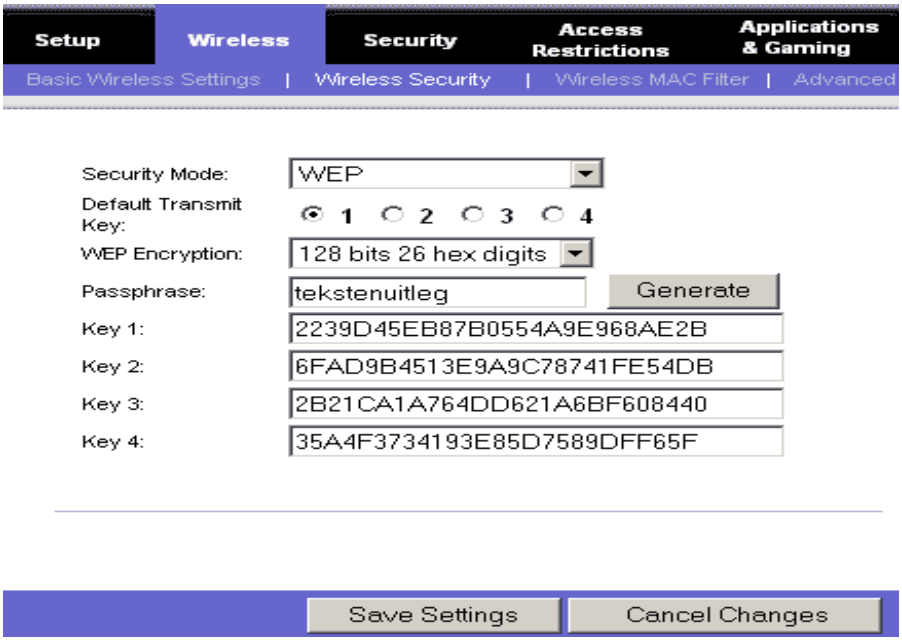

FIGURA 2.24: Panel de Configuración de Métodos de Cifrado para redes inalambricas.

2.2.4 ACCESO PROTEGIDO A WI-FI (WPA).

WPA que significa Wi-Fi Protected Access (Acceso Protegido a Wi-Fi) es un sistema para proteger las redes inalámbricas creado para corregir las deficiencias del sistema previo (WEP) e implementa la mayoría del estándar IEEE 802.11i [6]. Evidentemente la evolución o mejora del cifrado WEP es WPA basándose en métodos de autentificación y cifrado nuevo, el cual congela los puntos débiles que poseía WEP respecto a su vector de inicialización (VI) el cual podría ser retomado en las tramas para obtención de la clave WEP por fuerza bruta, de modo que WPA es el reforzamiento de WEP con sus distintas desventajas.

Para utilización del WPA como método de seguridad en nuestra red inalámbrica debe tener en cuenta que la mayoría de los equipos antiguos o mejor dicho, con las primeras tarjetas inalámbricas del mercado, no soportan este tipo de cifrado, también que a la fecha siguen en existencia estas tarjetas que no brindan soporte a WPA, la prueba principal está en el precio de los productos.

WPA incorpora un reforzamiento en generación de claves de 128 bits y un vector inicial de 48 bits, inicialmente WPA se diseñó como un sistema de protección de redes inalámbricas protegido mediante autentificación forzosa, para esto se utilizó el protocolo RADIUS, brindando así un usuario y clave para cada equipo que se conecte al punto de acceso (AP). Posteriormente se incorpora un método menos complicado y más factible para implementar WPA en oficinas y casa, para esto se dio uso de una clave pre-compartida (PSK, Pre-Shared Key) continuando en ambas modalidades de configuración la utilización del algoritmo RC4 para el cifrado de la información.

A continuación una pequeña lista de las mejoras de WPA:

- Implementación de un código de integridad del mensaje (MIC) [16].
- Evita el ataque de repetición (replay attack), ya que incluye un contador de tramas.
- En base al mayor tamaño de las claves, implementación de MIC y el contador de tramas, resulta más difícil la intrusión a la red inalámbrica.
- Al detectar dos intentos de ataques durante un minuto, las redes WPA se desconectan durante 60 segundos. La Figura 2.25 ilustra la explicación.

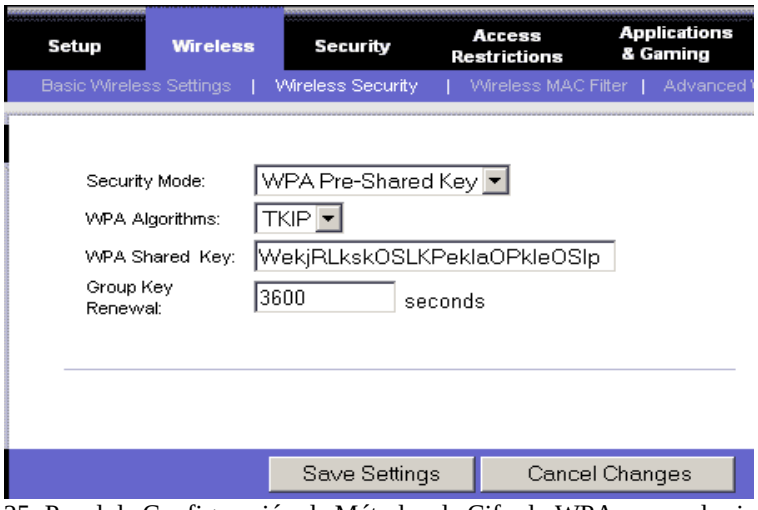

FIGURA 2.25: Panel de Configuración de Métodos de Cifrado WPA para redes inalambricas.

Actualmente WPA a pasado a evolucionar como WPA2 adoptando por completo el estándar IEEE 802.11i [20] pero aún no se extiende como soporte incluido en todos los dispositivos de hardware, sus respectivos fabricantes aún están en implementación del mismo.

2.2.5 PROTOCOLO DOMINANTE TEMPORAL DE LA INTEGRIDAD (TKIP).

TKIP o Protocolo Dominante Temporal de la Integridad (Temporal Key Integrity Protocol), también conocido como hashing de clave WEP WPA, basado para brindar una mejor protección al protocolo WEP el cual puede ser quebrado de una manera fácil, con el algoritmo que utiliza TKIP para encriptación de los datos se puede tener una red segura y sin la necesidad de cambiar de hardware para soportar otros protocolos.

El funcionamiento de TKIP se basa en generar una clave temporal (hashing), esta misma clave es compartida entre los equipos de la red inalámbrica y los puntos de acceso (AP), posteriormente TKIP utiliza el hashing y la MAC del cliente para combinarlos, así mismo le agrega la clave del vector de inicialización de 16 octetos generado y con esto cifra los datos, esta clave se reemplazara después de cada 10,000 paquetes. Una de las cuestiones similares a WEP, es que TKIP utiliza RC4 para cifrar el mensaje.

Con los métodos mencionados TKIP proporciona una mejor forma de brindar seguridad a las redes inalámbricas utilizando claves distintas después de un determinado numero de paquetes, así mismo evita el reemplazo del hardware (solo el firmware) para trabajar con este protocolo [15]. Observar la Figura 2.26.

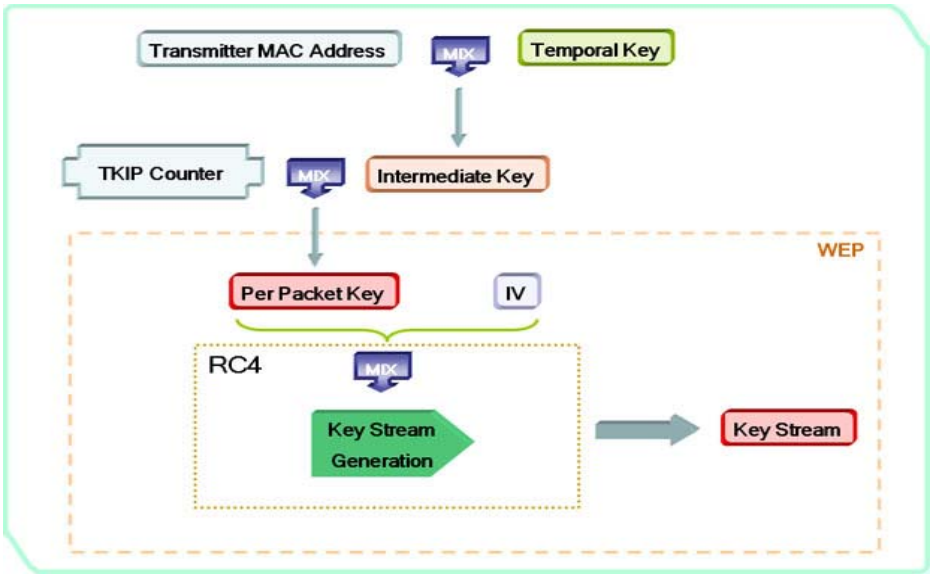

FIGURA 2.26: Estructura de Encriptación del Protocolo TKIP.

2.2.6 CODIGO DE INTEGRIDAD DEL MENSAJE (MIC)

MIC como su siglas lo indica es Código de la Integridad del Mensaje (Message Integrity Code), y está fuertemente implementado o enlazado con el protocolo TKIP ya que viene a formar un complemento para reforzar la encriptación del mensaje y así brindar una clave más fuerte y más difícil de quebrar. Con esto se obtiene una mayor seguridad en redes inalámbricas mediante la utilización de MIC.

El algoritmo MIC también es conocido como algoritmo Michael, el cual se implementa sobre redes inalámbricas para determinar la fidelidad del mensaje, siguiendo el método de encriptación TKIP, este mismo genera la clave MIC y se la anida al paquete [16].

SEGURIDAD INFORMATICA

2

2.3 SEGURIDAD BLUETOOTH. 53
2.3.1 ESTANDAR Y ESPECIFICACIONES DE BLUETOOTH 54
2.3.2 TOPOLOGIA DE RED BLUETOOTH 55
2.3.3 CAPAS DEL PROTOCOLO BLUETOOTH 58
2.3.4 AUDITORIA BLUETOOTH 62
2.3.4.1 BLUESNARF. 63
2.3.4.2 BLUEBUG. 64
2.3.4.3 HELOMOTO. 65
2.3.4.4 BLUELINE. 65

2.3 SEGURIDAD BLUETOOTH.

Bluetooth es una de las tecnologías con una mayor utilización día a día que los demás medios de comunicación, ya que se encuentra presente en la mayoría de los dispositivos electrónicos móviles y estáticos, como son por lo general en celulares, palms, manos libres, módulos usb, periféricos para computadoras, entre otros. Teniendo un gran impacto por la facilidad de compartición de elementos entre usuarios mediante un celular (dispositivo móvil), ahorrando utilización de cables y evitando estar en una posición fija para transferir los datos, también brinda un gran apoyo para facilitar la utilización del celular en momentos que se está realizando otra tarea, este dispositivo es el manos libres que se conecta al celular para atender las llamadas por medio de una orden de voz.

El dispositivo Bluetooth fue realizado como una tecnología al alcance de los usuarios, estableciéndose como una tecnología flexible y barata, teniendo una especificación muy definida para su incorporación en equipos móviles y estáticos, formando un dispositivo con poco consumo de energía, brindando transferencia de información de forma punto a punto o multipunto y una pila de protocolos muy amplia para adaptarse en varias formas de uso: voz, archivos, audio, video, etc. Uno de los puntos más débiles de Bluetooth es su seguridad. La Figura 2.27 muestra el logotipo Bluetooth.

FIGURA 2.27: Logotipo Bluetooth.

2.3.1 ESTANDAR Y ESPECIFICACIONES DE BLUETOOTH

Bluetooth es el nuevo estándar para comunicación de dispositivos electrónicos, estos van desde dispositivos pequeños como son: celulares, palms, manos libres (audífonos), impresoras, ratones, teclados, pastillas USB, micrófonos, entre otros, que permiten la transmisión de voz y datos mediante un enlace radiofrecuencia a una distancia especifica dependiendo del alcance del dispositivo bluetooth.

El proyecto Bluetooth surge en 1994 por la compañía Ericsson Mobile Communications sobre un dispositivo que funcionase como medio de comunicación entre dispositivos móviles, utilizando un enlace de radiofrecuencia y un bajo consumo de energía para dispositivos que utilizan baterías.

Bluetooth está conformado por dos símbolos que son:

- Las Ruinas Nordicas Hagalaz ᚼ
- Berkana ᛒ

La Figura 2.28 muestra la mezcla de los símbolos.

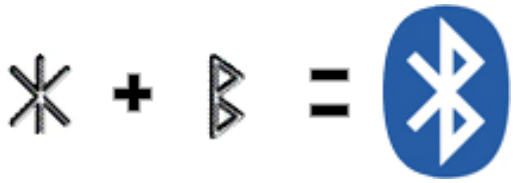

FIGURA 2.28: Formación del símbolo Bluetooth.

La especificación Bluetooth está sostenida por el grupo de trabajo IEEE 802.15.1 y se realizo con el fin de brindar un dispositivo electrónico de poco consumo eléctrico y para comunicar equipos de cómputo, celulares, dispositivos de hardware en entornos de comunicaciones móviles y estáticos. A continuación se presenta las especificaciones técnicas de bluetooth:

- Trabaja a una frecuencia de radio de 2.4GHz a 2.8GHz.
- Emplea la técnica de salto de frecuencia (FHSS, Frecuency Hopping Spread Spectrum), que consiste en dividir la banda en 79 canales a una longitud de 1 Mhz y realizar 1,600 saltos.
- La potencia de transmisión varía según la versión del núcleo bluetooth:
  - Versión 1.1: 723.1 Kbps.
  - Versión 1.2: 1 Mbps.
  - Versión 2.0+EDR: de 2.1 Mbps a 3 Mbps.
- La potencia de transmisión se divide en tres clases de productos:
  - Clase 1: 100 mW/20 dBm, con un rango de 100m.
  - Clase 2: 2.5 mW/4 dBm, con un rango de 10m.
  - Clase 3: 1 mW/0 dBm, con un rango de 1m.
- Utiliza protocolo de banda base para hacer una mejor transmisión de voz y datos.
- Los tipos de enlaces soportados para voz y datos son:
  - Enlace asíncrono sin conexión (ACL, Asynchronous Connectionless).

- Enlace síncrono orientado a conexión (SCO, Synchronous Connection-Orineted).

- En 1999 se ratificó la versión 1.0 de bluetooth, de modo que las mejoras fueron las siguientes:
  - Versión 1.1:
    - Soluciona errores de la versión 1.0.
    - Añade indicadores de calidad de señal recibida (RSSI).
  - Versión 1.2:
    - Implementa el salto de frecuencia (Adaptive Frequency Hopping).
    - Implementa tipo de enlace para aplicaciones de audio extended Synchronous Connections (eSCO) que mejora la calidad de voz.
    - Mejoras en el Host Coltroller Interface (HCI) para una sincronización más rápida.
  - Versión 2.0:
    - Nueva versión compatible con anteriores 1.x.
    - Incorpora tecnología Enhanced Data Rate (EDR) que incrementa la velocidad de transmisión hasta 3 Mbps.
    - Reducción de consumo de energía.

2.3.2 TOPOLOGIA DE RED BLUETOOTH

Bluetooth es un estándar que trabaja a una frecuencia de 2.4GHz disponible a nivel mundial, no requiere una licencia de operación y evita interferencias con otros dispositivos de comunicación. Los medios de comunicación bluetooth brindan el servicio de comunicación de forma estática y móvil siempre y cuando se encuentre dentro del radio de cobertura de la frecuencia o señal del dispositivo, también permite realizar sincronización de conexiones punto a punto y multipunto con sus respectivas normas.

El modelo de comunicación que se utiliza para sincronización de uno o varios dispositivos sobre comunicaciones bluetooth se basa en establecer un dispositivo maestro para realizar las conexión punto a punto o multipunto entre dispositivos de comunicación, en el cual el dispositivo maestro realiza la sincronización desde el punto de vista del reloj de tiempo y los saltos de frecuencia que realiza de canal en canal para sincronizar los dispositivos. Los dispositivos que se sincronizan con el maestro son llamados esclavos, los cuales pueden ser de 1 a 8 dispositivos esclavos, a este modelo de red se le denomina: Piconet [21].

La topología bluetooth se basa en la conformación de una o varias piconet, donde la unión de varias piconets se le denomina Scatternet [22], cabe mencionar que entre piconets cada uno de los dispositivos esclavos pueden ser participes en las distintas piconets mediante el enlace maestro de cada piconet, para este proceso se utiliza una Multiplexación por División de Tiempo (TDM) [23] la cual permite que cada dispositivo esclavo de una piconet tendrá acceso de presencia en forma secuencial en una piconet a la vez. Observar el ejemplo de la Figura 2.29.

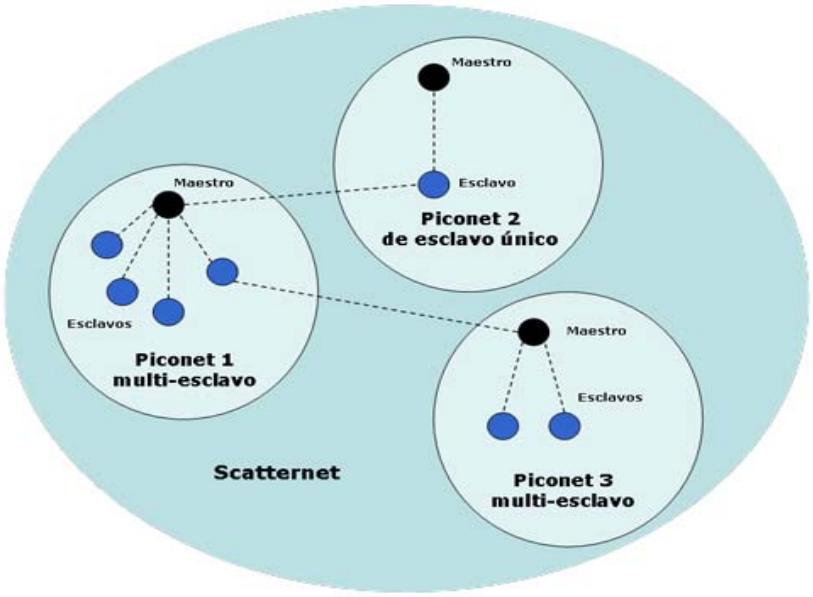

FIGURA 2.29: Topología Bluetooth.

Bluetooth es una tecnología que se incorpora y en la actualidad aún se sigue incorporando a distintos dispositivos móviles o estáticos, formando así una amplia gama de productos al alcance del usuario y de uso cotidiano. Los cuales brinden un servicio de comunicación e intercambio de información entre dispositivos bluetooth gracias al perfil OBEX FTP, el cual permite la transferencia de archivos entre móviles disponibles en el radio de cobertura disponible.

Los archivos que se pueden intercambiar entre dispositivos bluetooth utilizando el perfil OBEX FTP, va desde archivos de audio, texto, sincronización agendas, sincronización entre dispositivo y computadora, etc. Todo esto y más brindan los distintos productos existentes en el mercado que incorporan la tecnología bluetooth.

A continuación se listan en forma de categorías las distintas gamas de productos existentes:

- *Dispositivos de informática bluetooth.*
  Productos de hardware para sistemas de computo los cuales permitan la sincronización entre un ordenador y un sistema móvil que posea la tecnología bluetooth, estos van desde equipos portátiles que incorporan bluetooth integrado, pastillas USB para equipos de cómputo y pasarelas (gateways) de acceso a otras redes. Observar Figura 2.30.

FIGURA 2.30: Dispositivos de Informática Bluetooth.

- *Dispositivos de audio bluetooth.*
  Productos de uso personal para comunicación bluetooth, utilizados con más frecuencia para comunicaciones telefónicas y escuchar música, los cuales son auriculares para estero, auriculares para equipos de computo, manos libres y audífonos. Ver Figura 2.31.

FIGURA 2.31: Dispositivos de Audio Bluetooth.

- *Dispositivos de automóvil Bluetooth.*
  Productos utilizados para el sector automovilístico los cuales tienen impacto en sistemas de navegación para envío de coordenadas mediante un dispositivo manos libres utilizando voz, estos productos son sistemas integrados, manos libres, gps. Observar Figura 2.32.

FIGURA 2.32: Dispositivos de Automóvil Bluetooth.

- *Dispositivos de entrada y salida Bluetooth.*
  Productos disponibles para equipos de cómputo los cuales permiten la comunicación de periféricos con una computadora y viceversa, estos pueden ser teclados, ratones, impresoras, etc. Ver Figura 2.33.

FIGURA 2.33: Dispositivos de Entrada y Salida Bluetooth.

- *Dispositivos móviles Bluetooth.*
  Productos móviles de uso personal para comunicación telefónica, también para compartición de archivos, imágenes música y publicidad (para empresas), esto pueden ser celulares, palms, gphone's, smart phone's, pda's. Observar Figura 2.34.

FIGURA 2.34: Dispositivos Móviles Bluetooth.

La amplia gama de productos ya mencionados por categoría pueden ser sincronizados entre sí para formar piconets en cualquier parte del mundo, también se pueden realizar sincronización entre piconets para conformar las scatternets, todo esto con un fin de comunicación para intercambio de información de cualquier tipo utilizando un dispositivo móvil o estático sin necesidad de cables, utilizando solamente un radio de cobertura y alguna frecuencia que este estandarizada. A continuación se visualiza una piconet entre dispositivos. La Figura 2.35 ilustra la explicación.

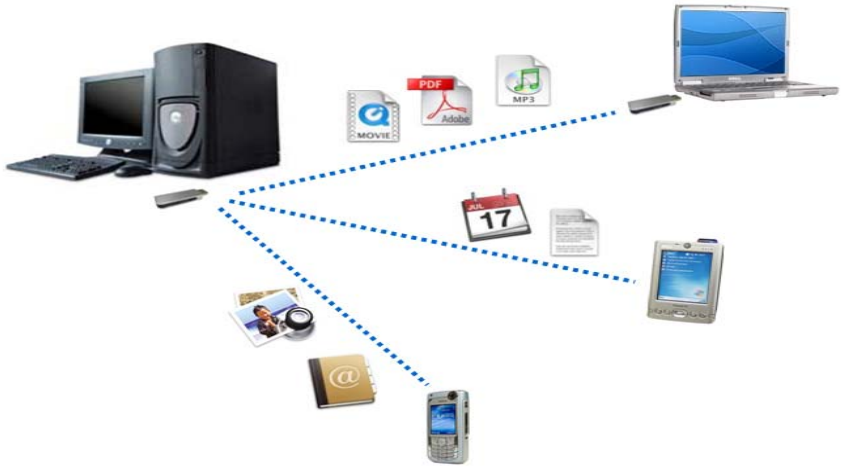

FIGURA 2.35: Modelo Visual de una Piconet.

2.3.3 CAPAS DEL PROTOCOLO BLUETOOTH

La pila de protocolos de bluetooth está basada en el modelo de referencia OSI y utiliza una arquitectura de protocolos que permiten dividir las funciones de red en un conjunto de niveles, pensado para que cada fabricante pueda realizar su propio método de interoperabilidad con esta tecnología.

La pila de protocolos Bluetooth se divide en dos zonas, donde ambas son necesarias para la comunicación. Así mismo es importante saber que ambas zonas están comunicadas mediante la interface de controlador de host (HCI), esto se observa Figura 2.18:

- *Hardware Bluetooth.*El fin del hardware bluetooth es brindar la comunicación y el envío de la información mediante la interface de radio.

- *Software Bluetooth.*Encargado de las capas superiores de enlace y aplicación, es decir, el conjunto de protocolos establecidos por el fabricante mediante una aplicación.

La Figura 2.36 ilustra la capas de protocolos bluetooth.

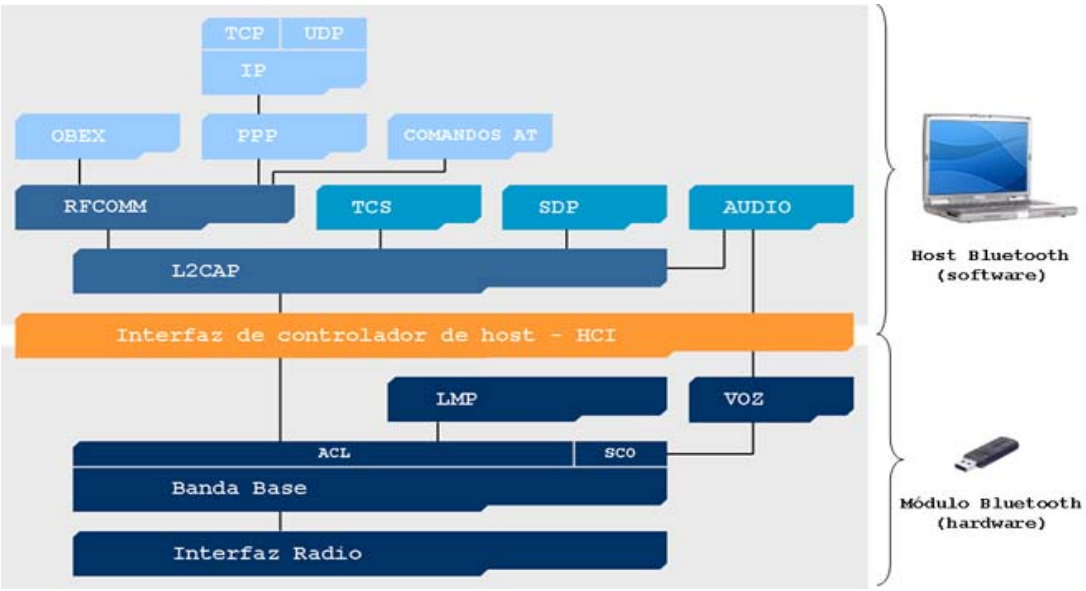

FIGURA 2.36: Capas del Protocolo Bluetooth.

A continuación se describirá cada una de las categorías de las capas de bluetooth.

Capa de Banda Base e Interfaz de Radio.
Son las que realizan el enlace físico mediante la radiofrecuencia [24] en un modo de enlace síncrono (SCO, Synchronous Connection-Oriented) o asíncrono (ACL, Asynchronous Connectionless) entre ambos dispositivos bluetooth, dentro de una piconet realizando las tareas de modulación y demodulación de los datos en la señal.

Capa de Protocolo de Gestión de Enlace (LMP).
La capa de protocolo de gestión de enlace (LMP, Link Manager Protocol) es el encargado de manejar diversas cuestiones de los dispositivo bluetooth que van desde la configuración del mismo hasta el control y negociación de paquetes. En seguida se listan los puntos que cubre esta capa:

- Configuración de los dispositivos bluetooth.
- Cuando ambos dispositivos bluetooth están en un mismo radio, la capa LMP realiza una serie de intercambios de mensajes para establecer el enlace entre ambos dispositivos.
- Dentro del intercambio de mensajes para inicializar un enlace, se incluyen mecanismos de seguridad como cifrado de datos para empezar una sesión bluetooth.

- Administra los niveles de energía y ciclos de trabajo de los dispositivos de radio.
- Utiliza el Gestor de Enlace (módulo de software) para descubrir otros gestores de enlaces remotos y empezar una comunicación.

El gestor de enlace es un módulo de software dentro de un microprocesador que se sitúa dentro del dispositivo bluetooth, y éste se comunica con la capa de protocolo de gestión de enlace para determinar el tipo de enlace que se utilizará en la comunicación, tipo de tramas y establecer la identificación de un dispositivo. Entre gestores de enlace intercambian datos del tipo unidades de datos de protocolo (PDU, Protocol Data Unit) [25] los cuales tienen un nivel más alto de prioridad que los demás paquetes, de esta forma no se interrumpe el trafico de L2CAP. Observar la Figura 2.37.

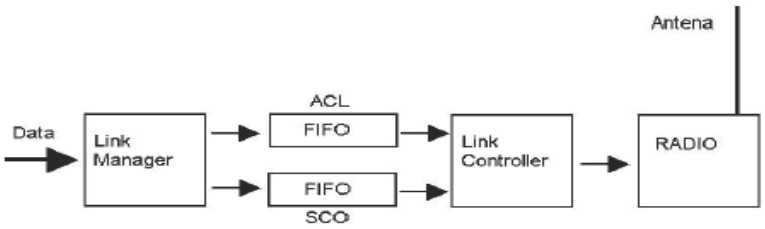

FIGURA 2.37: Gestor de Enlace.

Capa de Interfaz de Controlador de Host (HCI).
La capa de interfaz de controlador de host (HCI, Host Control Interface) actúa como comunicador o frontera entre las capas de protocolo relativas al hardware (hardware bluetooth) y relativas al software (software bluetooth) proporcionando un conjunto de comandos para la comunicación entre el dispositivo y el firmware del módulo bluetooth, sin importar que sean de distinto fabricante.

Esta capa brinda la facilidad para descubrir dispositivos cercas del radio de cobertura, para esto utiliza paquetes de consulta llamados: 'inquiry' que funcionan de la siguiente manera:

- Si el dispositivo está en modo visible (discoverable) estará en modo inquiry_scan y disponible para intercambiar mensajes, al recibir un mensaje cambiara su estado a inquiry_response y enviara una respuesta a host origen con su dirección MAC y otros parámetros.
- Si el dispositivo está en modo no visible (non discoverable) estará en modo inquiry_response y estará imposibilitado para responder mensajes al host origen.

A igual que los dispositivos de red que utilizan su dirección de acceso al medio (MAC) a nivel de red, los dispositivos bluetooth también poseen una dirección de acceso al medio (MAC), pero comúnmente denominada como: BD_ADDR.

Capa de Protocolo de Adaptación y Control del Enlace Lógico (L2CAP).
La capa de protocolo de adaptación y control del enlace lógico (L2CAP, Logical Link Control and Adaptation Protocol) sirve como multiplexor de protocolos para que la banda base pueda comunicarse con los protocolos de una capa superior, como son: RFCOMM, SDP, TCS.

También proporciona los servicios de segmentación, los cuales constan de dividir los paquetes a tamaños más pequeños y poder enviarlos a la banda base, recomposición de paquetes de banda base a paquetes L2CAP y la Calidad del Servicio vigilando que se cumplan los contratos de calidad establecidos y control de los recursos utilizados por los protocolos. L2CAP está definida únicamente para funcionar en modos de transmisión asíncrona sin conexión (ACL), solo entre dos dispositivos.

Capa de Protocolo de Descubrimiento de Servicios (SDP).
La capa de protocolo de descubrimiento de servicios (SDP) realiza búsquedas de servicios disponibles por dispositivos bluetooth para intercambiar información entre aplicaciones comunes, enumerando estos servicios para una selección rápida por el usuario.

Un servicio que brinde un dispositivo bluetooth puede ser cualquier aplicación para comunicación, control o manejo de información, estos pueden ser en forma de servicios para hardware o software. Existen servicios genéricos establecidos, están incorporados en los paquetes que se transfieren entre dispositivos bluetooth mediante un campo de tipo Service Class y estos son:

- Positioning (location identification).
- Networking (net, ad-hoc, etc...).
- Rendering (printing, speakers, etc...).
- Capturing (scanners, microphones, etc...).
- Object Transfer (v-inbox, v-folder).
- Audio (speakers, microphones, headset service, etc...).
- Telephony (cordless telephony, modem, headset services, etc...).
- Information (web-server, wap-server, etc...).

La Figura 2.38 muestra una mejor explicación.

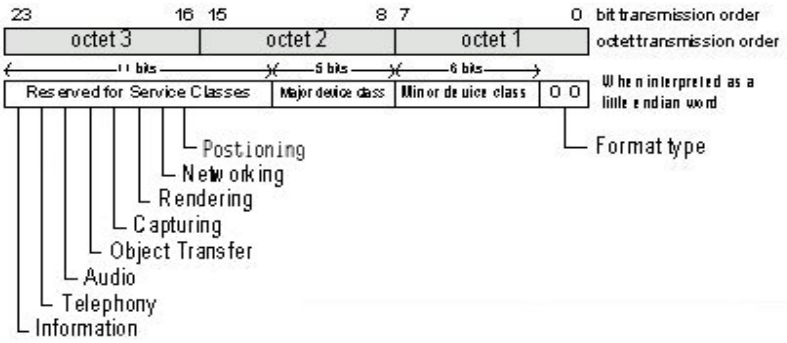

FIGURA 2.38: Clases de Servicios en un paquete bluetooth.

Capa RFCOMM.
Capa de protocolo RFCOMM (Radio Frecuency COMMunication) permite la emulación de los puertos serie RS-232 sobre el protocolo L2CAP proporcionando la capacidad de transporte de información hacia los servicios de niveles superiores que utilizan el cable serie como mecanismo de comunicación.

Comandos AT.

Son instrucciones codificadas que permiten la comunicación entre el hombre y un dispositivo móvil mediante comandos y una terminal. Los comandos AT fueron desarrollados en 1977 por Dennis Hayes como una interface de comunicación con modems para poder realizarle configuraciones y proporcionarle instrucciones a realizar.

Existen en la actualidad dispositivos como bluetooth que incorporar la utilización de comandos AT para manipular el dispositivo. Los niveles de manipulación están divididos por bloques los cuales incorporan varios aspectos para manipulación de servicios de algún dispositivo bluetooth, estos niveles son:

- *Bloque Básico de Comandos AT.*
  Permite manipulación del dispositivo realizando: desvío de llamadas, realización de llamadas con voz y datos, obtener información del dispositivo, modelos de la terminal, nivel de batería, calidad de señal, etc.

- *Bloque de Comandos AT referido a agendas.*
  Permite la manipulación de la agenda, que va desde modificar, eliminar y agregar nuevos contactos a la lista de contactos almacenados en la memoria del dispositivo o en una memoria individual, manipulación de las listas de llamadas recibidas, perdidas o realizadas y desvío de llamadas.

- *Bloque de Comandos AT referido a SMS's.*
  Permite la manipulación de Mensajes que están en la memoria del dispositivo o en alguna memoria individual, brindando los servicios de: visualización de mensajes, modificación de mensajes, eliminación de mensajes, escribir mensajes, y enviar mensajes. Ver Figura 2.39.

Modelo	Soporta		
	Bloque básico	Bloque Agenda	Bloque SMS
Nokia™ 6820	✓		✓
Sony-Ericsson™ T68i	✓	✓	
Nokia™ 6600	✓		

FIGURA 2.39: Tabla de algunas empresas que implementan bloques de comandos at.

2.3.4 AUDITORIA BLUETOOTH

Bluetooth es un estándar creado con el fin de que otros fabricantes pudieran incorporar esta tecnología a sus productos y establecer su propia capa de protocolos, formando así un método de comunicación adaptable a equipos pequenos que requieran poco consumo de energía y proporcionar una variedad inmensa de servicios. En la actualidad las pilas de protocolos Bluetooth más conocidas son Widcomm, Toshiba Bluetooth Stack, Microsoft Windows XP Bluetooth y IVT BlueSoleil Stack, mientas que los Sistemas Operativos Linux disponen de las pilas de protocolos Bluetooth BlueZ, OpenBT y Affix, de Nokia [29].

A pesar de ser una de las tecnologías nuevas en el mercado, no es una tecnología de comunicación exenta a fallos en seguridad, particularmente en dispositivos móviles y estáticos. En bluetooth las cuestiones de inseguridad son verdaderamente alarmantes ya que ponen en riesgo la privacidad del usuario y algunas cuestiones monetarias referente a realización de llamadas sin conocimiento del usuario propietario.

Uno de los escenarios más tentados a una intrusión a dispositivos móviles bluetooth son los sitios públicos (escuelas, bancos, etc.), donde sin limitación alguna puede haber equipos de cómputo manejadas con personas mal intencionadas intentando irrumpir en los servicios que proporcionan los dispositivos móviles mediante un enlace bluetooth.

Para dar una solución a esto, muchas compañías han agregado métodos de autentificación y autorización a servicio que brinda un dispositivo móvil mediante una lista segura de conexión, donde el usuario deberá registrar los equipos y establecer una clave única comúnmente llamada PIN, para que de esta manera se pueda considerar como segura el intercambio de información o prestación de algún servicio, a todo este proceso se le nombra emparejamiento.

No obstante sobre estas especificaciones incorporadas a los equipos actuales móviles, siguen surgiendo problemas de seguridad por parte de algunas malas implementaciones respecto a privilegios de servicios, ya que algunos servicios se basan principalmente en el método de autorización y estos a su vez tienen como referencia segura la lista de conexiones o equipos seguros mediante su dirección MAC.

2.3.4.1 BLUESNARF.

Se basa en la extracción de archivos del dispositivo móvil bluetooth mediante la vulnerabilidad del Perfil de Carga de Objetos (OBEX Object Push) el cual proporciona el servicio de carga y descarga de información entre dispositivos bluetooth.

La vulnerabilidad se basaba principalmente en la falta de seguridad para con la autentificación y autorización al momento de pedir que se transfiriera información entre dispositivos bluetooth, este acceso era libre y se podría disponer de la información del dispositivo, algunos archivos disponibles se muestran en la Figura 2.40.

Filename	Description	Supported operations
Device Info		
telecom/devinfo.txt	Information hardware version, software version, serial number, etc. Character sets	GET
telecom/rtc.txt	The Real Time Clock Object contains the current date and time of the device	GET/PUT
Phone Book		
telecom/pb.vcf	Level 2 access (Access entire phonebook database)	GET/PUT
telecom/pb/luid/.vcf	Add new entry	PUT
telecom/pb/0.vcf	Own business card	GET/PUT
telecom/pb/info.log	Supported properties and memory info	GET
telecom/pb/luid/ ###.log	Change log	GET
telecom/pb/luid/ cc.log	Change counter	GET
Calendar		
telecom/cal.vcs	Level 2 access	GET/PUT
telecom/cal/luid/.vcs	Add new entry	PUT
telecom/cal/info.log	Supported properties and memory info	GET

FIGURA 2.40: Archivos del perfil OBEX.

En la actualidad el Perfil de Carga de Objetos sigue activo en todos los dispositivos móviles bluetooth pero implementa un mecanismo de autentificación y autorización de acceso y este se realiza agregando el dispositivo móvil a la lista de dispositivos de confianza para comunicación.

La obtención de estos archivos se puede llevar acabo mediante la aplicación obexftp para Linux, la cual proporciona el servicio de extracción de archivos desde dispositivos móviles bluetooth [26].

2.3.4.2 BLUEBUG.

Esta vulnerabilidad nos permite conectarnos a un canal RFCOMM del dispositivo móvil bluetooth y empezar a realizar comando AT en la terminal, con esto el atacante podrá realizar operaciones como:

- Gestión de agenda: eliminar, editar, agregar contactos.
- Gestión de mensajes: crear y enviar, eliminar mensajes.
- Gestión de llamadas: visualizar y eliminar llamadas perdidas, recibidas o realizadas, y realizar llamadas de voz a otros dispositivos móviles.
- Información: obtener información del dispositivo móvil por completo.

El principal problema de esta vulnerabilidad es la realización de llamadas sin aviso alguno hacia otros dispositivos, ya que esto podría repercutir en el usuario propietario. Para su solución se realizaron implantaciones de seguridad emitiendo una autorización y autentificación al canal, forzando a los usuarios o atacantes agregar el dispositivo móvil a la lista de dispositivos confiables.

Para lograr la conexión a un canal RFCOMM se debe utilizar la aplicación rfcomm proporcionada por las librerías BlueZ para Sistemas Operativos Linux, posteriormente que se abre un canal RFCOMM se debe utilizar la aplicación *cu* que está integrada en el paquete Taylor UUCP, esta aplicación nos permite iniciar una sesión de comandos AT en algún canal rfcomm abierto [27].

2.3.4.3 HELOMOTO.

Este tipo de ataque solo afecta a equipos Motorola en sus modelos v80, 500, 600 los cuales poseen una implementación incorrecta en la lista de dispositivos de confianza de los modelos ya mencionados, permitiendo a un atacante acceder a perfiles que requieran autorización, pero no autentificación.

El ataque se lleva acabo cuando el atacante intenta un envío de una tarjeta de visitas vCard, posteriormente el dispositivo móvil automáticamente agrega al atacante en la lista de dispositivos de confianza, desde este momento el atacante puede realizar una conexión y acceder a perfiles que requieran autorización, uno de gran importancia es el Perfil de Pasarela de Voz (Voice Gateway Profile) en donde el atacante podrá empezar a realizar comandos AT dentro de este perfil. Este proceso se realiza con una utilería llamada helomoto, realizada con las librerías BlueZ [28].

2.3.4.4 BLUELINE.

Esta vulnerabilidad afecta a los dispositivos Motorola PEBL U2 y RAZR V3 y el fin principal es realizar una conexión al Perfil de Pasarela de Voz (Voice Gateway Profile) para realizar comandos AT en la terminal móvil.

Los dispositivos móviles ya mencionados evitan la vulnerabilidad helomoto de cierto modo que los atacantes que intenten realizar alguna conexión sean rechazados a menos que estén dentro de la lista de dispositivos de confianza, pero una de las problemáticas en estos dispositivos móviles es que los dispositivos que estén dentro de la lista de dispositivos que tuvieron una conexión anterior dentro de la lista del historia pueden realizar envíos de información al equipos atacado, donde por cada comando realizado el usuario tendrá que notificar su autorización.

Para llevar acabo este ataque, se debe utilizar la aplicación helomoto con el fin de abrir una conexión obexftp y esta sea denegada por el dispositivo móvil atacado, pero desde este momento el dispositivo móvil atacado registra la conexión dentro del historial de conexiones, en este momento se debe realizar una conexión con la aplicación hciconfig proporcionada por

las librerías BlueZ en linux, y en forma de engaño se debe realizar una alteración del mensaje a mostrar en pantalla del usuario para que este acepte la conexión

Una ves aceptado el mensaje malicioso, el atacante utilizando la aplicación rfcomm realizara una conexión al canal del Perfil de Pasarela de Voz y esta sera aceptada, ya que este perfil no requiere autorización, solo autentificación y podrá realizar comandos AT en la terminal.

SOCKETS DE COMUNICACIONES

3

3. SOCKETS DE COMUNICACIONES. 68
3.1 QUE ES EL SOCKET. 68
3.2 TIPOS DE SOCKETS. 69
3.3 ENCAPSULACION DE LOS DATOS. 70
3.4 ESTRUCTURAS DE DATOS DEL SOCKET. 71
3.5 DIRECCIONES IP. 72
3.6 FUNCIONES PARA SOCKETS. 74
3.6.1 FUNCION SOCKET(). 75
3.6.2 FUNCION CLOSE(). 76
2.6.3 FUNCION SHUTDOWN(). 76
3.6.4 FUNCION BIND(). 76
3.6.5 FUNCION LISTEN(). 78
3.6.6 FUNCION ACCEPT(). 79
3.6.7 FUNCION CONNECT() 81
3.6.8 FUNCION SEND(). 82
3.6.9 FUNCION RECV(). 84
3.6.10 FUNCION SENDTO(). 86
3.6.11 FUNCION RECVFROM(). 87
3.6.12 FUNCION SELECT(). 88
3.6.13 FUNCION GETPEERNAME(). 91
3.6.14 FUNCION GETHOSTNAME(). 94
3.6.15 FUNCION GETDOMAINNAME(). 95
3.6.16 FUNCION GETHOSTID(). 96
3.6.17 FUNCION GETSOCKNAME(). 97

3. SOCKETS DE COMUNICACIONES.

3.1 QUE ES EL SOCKET.

Un socket es un canal de comunicaciones por donde se pueden interconectar dos computadoras o más, con el fin de intercambiar datos los cuales son utilizados por la aplicación del usuario, estos sockets no solo pueden comunicar aplicaciones, sino también son un medio para llegar a un dispositivo de hardware.

Los sockets no son un sistema de hardware, ni un aparato casero; sólo son llamadas que se realizan a una determinada función del lenguaje en que se está programando, y este como contestación retorna un valor entero asociado a un archivo, el cual puede usarse para comunicarse con:

- Aplicaciones de red.
- Conexiones de red.
- Terminales.
- Archivos FIFO.
- Archivo en el disco duro local.

Es importante resaltar que en este documento se visualizaran códigos para plataformas Linux, ya que facilitan el manejo y programación de los sockets, también existe gran diferencia entre los archivos de Windows y los descriptores de archivos en Linux. Los archivos en Windows son apuntadores de archivos dentro de una estructura y utilizan la biblioteca estándar de ANSI C, los archivos en Unix son descriptores de archivos que utilizan una tabla de descriptores de archivos y funciones establecidas en la biblioteca estándar de E/S de Unix.

Posteriormente utilizando las funciones adecuadas para comunicarse con el socket, se pondrá en disputa la fiabilidad e integridad de los datos a enviar al socket y recibir del mismo.

Las aplicaciones de Internet, así como:

- MSN Messenger.
- LimeWire
- Juegos de PC On-Line.
- Aplicaciones para utilización en Red.
- Servicio FTP.

Utilizan un socket de comunicaciones el cual está asociado a un número de puerto de nuestra computadora, por donde se están intercambiando datos para el funcionamiento del programa. Pero estos tipos de socket tienen un principal objetivo, ya que dichos datos que se intercambian pueden ser recibidos por alguna otra aplicación que esté en escucha en el mismo puerto, he aquí donde entra la fiabilidad, confiabilidad y protección de datos. En donde cada aplicación que utiliza un socket de comunicaciones incorpora su propio sistema de encriptación/desencriptación de datos (protección de datos).

Los sockets no solo pueden ser observados (escuchar en el mismo puerto), sino que también se pueden programar sockets en modo bloqueado, de esta forma solo permite que una sola aplicación este en escucha en dicho socket (puerto de comunicaciones), pero también pueden programarse sockets en modo no bloqueado, por si necesitamos que otros escuchen en el mismo socket (puerto de comunicaciones).

Por Ejemplo:

Sockets Bloqueados:
- Aplicaciones particulares en red.
- Otras aplicaciones de uso personal o que transporta datos confidenciales.

Sockets No Bloqueados:
- Aplicaciones para navegación Web.
- Aplicaciones para conversaciones (Chat).
- Aplicaciones para comunicación de programas para intercambio de archivos (P2P).
- Puertos de utilización continua por distintos dispositivos de hardware o software.

3.2 TIPOS DE SOCKETS.

El socket es un número entero asociado a un archivo, el cual nos permitirá comunicarnos con el puerto al que estamos escuchando, pero no solo basta crear un socket, ya que existen tipos de sockets los cuales sirven para comunicarnos con distintos tipos de sockets. Para un mejor entendimiento, si tenemos un socket A en escucha en otra máquina y queremos comunicarnos con dicho socket A, se tiene que realizar un sockets de tipo A para comunicarnos con dicho socket.

Existen varios tipos de sockets, por mencionar algunos son:

- DARPA :: Sockets de Internet.
- Nombres de ruta en un nodo local :: Sockets de Unix.
- Direcciones CCITT X.25 :: Sockets X.25.

En este documento se tratan los DARPA :: Sockets de Internet, y dentro de DARPA existen tipos de socket para el envío de datos, los cuales establecen el estado del socket. Algunos tipos de socket en DARPA son:

- SOCK_DGRAM :: Socket de Datagramas.
- SOCK_STREAM :: Sockets de Flujo.
- RAW SOCKETS :: Sockets Originales/a nivel capa de red.

Los sockets de flujo (SOCK_STREAM) son sockets orientados a la conexión, proporcionas 2 vías para comunicación de datos, un flujo para el envío y un flujo para la

recepción de los datos, brindando una mejor calidad y preservación de los datos, ya que permite que la información llegue en el mismo orden en el que se envío.

Los sockets de flujo utilizan un tipo de protocolo para mantener la calidad de los datos, este protocolo es el Protocolo de Control de Transmisiones TCP (ver RFC-793) el cual asegura que los datos lleguen secuencialmente y libres de errores. Posteriormente utiliza el Protocolo de Internet, IP, (ver RFC-791) para el enrutamiento en Internet, pero aquí no se responsabiliza de la integridad de los datos.

Los sockets de datagramas (SOCK_DGRAM) son sockets orientados a la no conexión, ya que no son como los sockets de flujo que mantienen un flujo abierto de conexión, estos solo sirven para envío y recepción rápida de datos, realizando la siguiente función:

- Construcción del paquete.
- Inserción de encabezado IP con la información del destino.
- Mensaje enviado.

Estos tipos de sockets son generalmente usados para el envío de paquete por paquete, en donde el sockets de datagrama al enviar un paquete al receptor, espera como respuesta un paquete ACK, de lo contrario re-envía el paquete hasta esperar respuesta.

Los sockets de datagrama utilizan un protocolo para el envío de datos, este es el Protocolo de Datagramas de Usuario UDP (ver RFC-768).

3.3 ENCAPSULACION DE LOS DATOS.

Después de la creación del sockets, hay que saber como se envía la información, para esto solo es cuestión ver el modelo OSI, el cual tiene las siguientes capas:

- Aplicación
- Presentación
- Sesión
- Transporte
- Red
- Enlace
- Físico

En donde cada capa realiza su propia tarea, desde la capa de aplicación a la de transporte se genera el paquete de datos a enviar por el socket, posteriormente se encapsula el paquete con una dirección de internet IP en la capa de red, en seguida se re-encapsula el paquete pegandole una dirección de hardware (MAC) y finalmente la codificación de los datos para el envío al destino.

Todos estos pasos no se realizan de modo manual en la programación a nivel de sockets, estos pasos son realizados por programas externos al nuestro para facilitar el envío de

los datos. Solo es cuestión del usuario realizar llamadas a las siguientes funciones para el envío y recepción de los datos:

- send();
- recv();

La función send() envía los datos, en donde el kernel construye la capa de transporte y de Internet, después el hardware se encarga de la Capa de Acceso al Medio (MAC).

La función recv() realiza extrae las partes del paquete y las guarda en un buffer.

3.4 ESTRUCTURAS DE DATOS DEL SOCKET.

En la programación de sockets, toda información referente al tipo de socket a realizar e información de dirección, se guardara en una estructura de datos, la cual es:

Código:

```
struct sockaddr
        {
        unsigned short sa_family;    //direccion de familia, AF_xxx
        char sa_data[14];    //14 bits para el protocolo de direccion
        }
```

sa_family :: es el tipo familia de socket a realizar, el más usual es: AF_INET es una familia que puede ser utilziado para comunicarse con sockets realizados en plataformas Windows o Linux.

sa_data[14] :: contiene la dirección destino y el puerto al que se asociará el socket.

Para trabajar con la estructura **struct sockaddr** los programadores crearon una estructura paralela, la cual es:

Código:

```
struct sockaddr_in
        {
        shot int sin_family;     //familia de direccion
        unsigned short int sin_port;     //numero de puerto
        struct in_addr sin_addr;     //direccion de internet
        unsigned char sin_zero[8];     //mismo tamano que struct sockaddr
        }
```

sin_family :: familia de dirección, es idéntica a sa_family.

sin_port :: puerto al que se le va a asociar un descriptor de archivo para interactuar con él, este debe ser en un ser dado en un tipo de orden de byte, el cual es red de orden de byte (Network Byte Order).

sin_addr :: dirección IP del destino, la cual debe estar en red de orden de byte (Network Byte Order).

sin_zero[8] :: variable que se debe rellenar con ceros, los cuales indican el mismo tamaño de la estructura **struct sockaddr**.

Esta estructura sirve para facilitar un mejor manejo de los datos. Cabe resaltar que **sin_addr** y **sin_port** deben estar en red de orden de byte (Network Byte Order), ya que estos son encapsulados junto con el mensaje a enviar, en cambio **sin_family** es utilizado por el kernel para establecer el tipo de familia del socket de direcciones, así que se debe dejar como está directamente AF_INET, que es host de orden de byte (Host Byte Order).

Existen dos tipos de orden de byte, que son:

- Byte más importante primero (llamado un octeto).
- Menos importante primero.

Host de orden de byte (Host Byte Order) y red de orden de byte (Network Byte Order), o también llamado Big-Endian Byte Order al NBO, son las formas del orden de byte en que las máquinas guardan sus números internamente para uso en encapsulamiento, procesamiento, etc.

Existen dos tipos en que se pueden convertir el Orden de Byte, estos son: short (2 bytes) y long (4 bytes), para ellos están las funciones que hacen posible esta conversión y éstas son:

htons() :: De Host a Red Corto (Host to Network Short).
htonl() :: De Host a Red Largo (Host to Network Long).
ntohs() :: De Red a Host Corto (Network to Host Short).
ntohl() :: De Red a Host Largo (Network to Host Long).

Se recomienda por una mejor portabilidad, todos los bytes se pongan en NBT, así no importa la versión, antigüedad de la máquina, el código funcionará para cualquiera.

3.5 DIRECCIONES IP.

Las direcciones IP son un conjunto de números delimitados por un punto, los cuales son conocidos como el identificador que representa a cada máquina dentro de una Red, estas direcciones se utilizan para llamar o comunicarnos con una máquina dentro de la red. Visualmente el usuario conoce la IP como por ejemplo:

10 . 9 . 8 . 7

La cual se divide:

10 . 9 :: Red
8 :: Subred
7 :: host

Esto sería de forma visual, pero las máquinas manejan dicha dirección IP, como un conjunto de bits delimitados por un punto, en donde la dirección mencionada equivaldría a:

10 . 9 . 8 . 7
00001010 . 00001001 . 00001000 . 00000111

Este conjunto de bits se conoce como dirección de red (Nework).

En la programación de sockets existen las funciones:

Código:

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int inet_addr( const char *addr );
```

***addr** :: es la dirección IP en forma de cadena y es devuelta en el formato de bits.

Código:

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/in.h>

//ASCII to Network
inet_aton( const char *addr, struct in_addr *inp );
```

cont char *addr :: direccion IP en formato de cadena.
struct in_addr *inp :: puntero a **struct sockaddr_in** donde se guardará la informacion.

La función anterior devuelve la dirección IP en formato de bits, almacenando esta información en el segundo argumento.

Código:

```
#include <sys/socket.h>
#include <inet/in.h>
#include <arpa/inet.h>

//Network to ASCII
inet_ntoa( struct in_addr *inp );
```

struct in_addr *inp :: puntero a **struct sockaddr_in** donde es almacenada la dirección en formato de bits.

La función anterior devuelve la dirección en formato de bits almacenada en la estructura, en modo ASCII para visualización en formato de Dirección IP.

A continuación se presenta un ejemplo de uso de las funciones mostradas:

Código 001.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PUERTO 1414

int main()
        {
        struct sockaddr_in test;

        test.sin_family= AF_INET; //familia de direccion
        test.sin_port= htons(PUERTO);  //de host a red (NBO)
        //forma 1
        //test.sin_addr.s_addr= inet_addr( "127.0.01" ); //loopback
        //forma2
        inet_aton( "127.0.01", &test.sin_addr );

        memset( &(test.sin_zero), '\0', 8 ); //rellenamos

        system( "clear" );
        printf( "Mi IP es: %s", inet_ntoa(test.sin_addr) );
        getchar();
        return 0;
        }
```

3.6 FUNCIONES PARA SOCKETS.

En esta parte se estudian las distintas funciones para la programación de sockets para redes en plataformas Unix y Windows, el principal objetivo de cada función que se mencione es:

- Realización del socket.
- Asociación del socket.
- Simplificar el proceso.
- Encapotamiento
- Conversión de direcciones.
- Comunicación
- Enlace.
- Envío y recepción de datos.

Mediante el uso de éstas mismas, todo el trabajo se pasa al kernel del sistema y éste automatizará las distintas actividades.

3.6.1 FUNCION SOCKET().

Función que proporciona un descriptor de archivo para la comunicación con el socket. En caso de error retorna un -1 y rellena la variable “errno” con un valor de error.

Código:

```
#include <sys/socket.h>
#include <sys/types.h>

int socket( int familia, int tipo, int protocolo );
```

int familia :: Familia de direcciones.

int tipo :: tipo de sockets a realizar.

int protocol :: se recomienda 0 para que la función escoja el protocolo correcto.

A continuación se presenta un código de ejemplo para su uso:

Código 002.c (reducido):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PUERTO 1414

int main()
        {
        struct sockaddr_in servidor;
        int sockfd; //socket file desciptor

        //rellenamos
        servidor.sin_family= AF_INET; //familia
        servidor.sin_port= htons(PUERTO); //NBO
        inet_aton( "127.0.0.1", &servidor.sin_addr );
        memset( &servidor.sin_zero, '\0', 8 );

        system( "clear" );

        //creamos socket y obtenemos descriptor
        if( (sockfd=socket( AF_INET, SOCK_STREAM, 0 )) == -1 )
                {
                fprintf( stderr, "Problemas para Crear Socket :: %s", strerror(errno) );
                exit(1);
                }

        printf( "SOCKET ..... [OK]" );

        printf( "\n\nFin de la Ilustracion para uso de SOCKET()...." );

        close(sockfd); //cerramos socket
        getchar();
        return 0;
        }
```

3.6.2 FUNCION CLOSE().

Función para realizar el cierre de un socket. En caso de error retorna -1.

Código:

```
#include <sys/socket.h>
#include <sys/types.h>

int close( int sockfd );
```

int sockfd :: Descriptor de archivo.

2.6.3 FUNCION SHUTDOWN().

Función que realiza un cambio de estado en el socket.

Código:

```
#include <sys/socket.h>
#include <sys/types.h>

int shutdown( int sockfd, int accion );
```

int sockfd :: Descriptor de archivo.

int acción :: Valor que indicará la acción a realizar con el socket. Los posibles valores son:
0 :: Recibir, está deshabilitado.
1 :: Enviar, está deshabilitado.
2 :: Enviar y Recibir, están deshabilitados (equivalente a **close()**).

3.6.4 FUNCION BIND().

Función que asocia el socket con el puerto indicado y utiliza el descriptor realizado por **socket()** para llamadas entrantes. En caso de error retorna un -1 y rellena a "errno" con un valor de error.

Código:

```
#include <sys/socket.h>
#include <sys/types.h>

int bind( int sockfd, struct sockaddr_in *servidor, int longitud );
```

int sockfd :: descriptor de archivo socket realizado con la función **socket()**.

struct sockaddr_in *servidor :: puntero a la **struct sockaddr_in** que contendrá toda la información de tu dirección.

int longitud :: longitud de **struct sockaddr_in *servidor**.

A continuación se presenta un código de ejemplo para su forma de uso:

Código 002.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PUERTO 1414

int main()
	{
	struct sockaddr_in servidor;
	int sockfd; //socket file desciptor

	//rellenamos
	servidor.sin_family= AF_INET; //familia
	servidor.sin_port= htons(PUERTO); //NBO
	inet_aton( "127.0.0.1", &servidor.sin_addr );
	memset( &servidor.sin_zero, '\0', 8 );

	system( "clear" );

	//creamos socket y obtenemos descriptor
	if( (sockfd=socket( AF_INET, SOCK_STREAM, 0 )) == -1 )
		{
		fprintf( stderr, "Problemas para Crear Socket :: %s", strerror(errno) );
		exit(1);
		}

	printf( "SOCKET ..... [OK]" );

	//asociamos socket con el puerto
	if( bind( sockfd, (struct sockaddr *)&servidor, sizeof(struct sockaddr) ) == -1 )
		{
		fprintf( stderr, "\nProblemas para Asociar Socket :: %s", strerror(errno) );
		exit(1);
		}

	printf( "\nBIND ..... [OK]" );

	printf( "\n\nFin de la Ilustracion para uso de BIND()...." );

	close(sockfd); //cerramos socket
	getchar();
	return 0;
	}
```

3.6.5 FUNCION LISTEN().

Función que nos permite escuchar en el puerto hasta recibir una conexión entrante, posteriormente interactuar con dicha conexión. Esta función nos permite establecer un límite de conexiones entrantes en cola. En caso de error retorna un -1 y establece un valor en la variable "errno".

Código:

```
#include <sys/socket.h>
#include <sys/types.h>

int listen( int sockfd, int limite );
```

int sockfd :: descriptor de archivo.

int limite :: número de clientes límite que tendremos en cola/espera.

A continuación se muestra un código de ejemplo para su uso:

Código 003.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PUERTO 1414
#define LIMITE 10

int main()
        {
        int sockfd;
        struct sockaddr_in servidor;

        servidor.sin_family= AF_INET; //familia de direcciones
        servidor.sin_port= htons(PUERTO); //NBO
        inet_aton( "127.0.0.1", &servidor.sin_addr ); //ip en formato de Red
        memset( &servidor.sin_zero, '\0', 8 ); //rellenamos

        //Creamos socket y obtenemos descriptor
        if( (sockfd=socket( AF_INET, SOCK_STREAM, 0 ))==-1 )
                {
                fprintf( stderr, "Problemas para Crear Socket :: %s", strerror(errno) );
                exit(1);
                }
        else
                printf( "SOCKET ..... [OK]" );

        //asociamos socket
        if( bind( sockfd, (struct sockaddr *)&servidor, sizeof(struct sockaddr) )==-1 )
                {
                fprintf( stderr, "\nProblemas para Asociar Socket :: %s", strerror(errno) );
                exit(1);
                }
        else
                printf( "\nBIND ..... [OK]" );

        //escuchamos en el socket hasta recivir conexion
        if( listen( sockfd, LIMITE )==-1 )
                {
                fprintf( stderr, "\nProblemas para Escuchar en el Socket :: %s", strerror(errno) );
                exit(1);
                }
```

```
        else
                printf( "\nLISTEN ..... [OK]" );

        close(sockfd);

        printf( "\n\nPusa para salir.." );
        getchar();
        return 0;
        }
```

3.6.6 FUNCION ACCEPT().

Función que realiza la aceptación de una llamada entrante que fue puesta en cola por la función **listen()**, posteriormente al aceptar dicha conexión entrante se retorna un nuevo descriptor de archivo socket para interactuar únicamente con dicho cliente (socket). En caso de error retorna -1 y establece un valor en la variable “errno”.

Código:

```
#include <sys/socket.h>
#include <sys/types.h>

int accept( int nuevofd, void *addr, int *longitud );
```

int nuevofd :: Descriptor de archivo nuevo.

void *addr :: Puntero a **struct sockaddr** donde guardaremos los datos del cliente.

int *longitud :: Puntero a la longitud de **struct sockaddr** del cliente.

A continuación se visualiza un código de ejemplo para su uso:

Código 005.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PUERTO 1414
#define LIMITE 10

int main()
	{
	int sockfd, clientfd, size_c=0;
	struct sockaddr_in servidor;
	struct sockaddr_in cliente;

	servidor.sin_family= AF_INET;
	servidor.sin_port= htons(PUERTO);
	inet_aton( "127.0.0.1", &servidor.sin_addr );
	memset( &servidor.sin_zero, '\0', 8 );

	system( "clear" );
	if( (sockfd=socket( AF_INET, SOCK_STREAM, 0 ))==-1 )
		{
		fprintf( stderr, "Problemas para Crear Socket :: %s", strerror(errno) );
		exit(1);
		}
	else printf( "SOCKET ..... [OK]" );

	if( bind( sockfd, (struct sockaddr *)&servidor, sizeof(struct sockaddr) )==-1 )
		{
		fprintf( stderr, "\nProblemas para Asociar Socket :: %s", strerror(errno) );
		exit(1);
		}
	else printf( "\nBIND ..... [OK]" );

	if( listen( sockfd, LIMITE )==-1 )
		{
		fprintf( stderr, "\nProblmas para Escuchar en puerto :: %s", strerror(errno) );
		exit(1);
		}
	else printf( "\nLISTEN ..... [OK]" );

	size_c= sizeof(struct sockaddr);

	if( (clientfd=accept( sockfd, (struct sockaddr *)&cliente, &size_c ))==-1 )
		{
		fprintf( stderr, "\nProblemas para Aceptar Cliente :: %s", strerror(errno) );
		exit(1);
		}
	else printf( "\nACCEPT ..... [OK]" );

	close(clientfd);
	close(sockfd);

	printf( "\n\nFin de la aplicacion..." );
	getchar();
	return 0;
	}
```

3.6.7 FUNCION CONNECT()

Función que realiza la conexión hacia un socket que se encuentre en escucha, antes de usarse dicha función debe realizarse un socket y obtener un descriptor de archivo para el socket al que se desea conectar, posteriormente realizar la conexión entre sockets. En caso de error retorna un -1 y establece un valor de error en la variable "errno". También es importante mencionar que esta función depende mucho del socket del servidor, ya que éste debe tener su socket en escucha (función **listen()**) para una conexión exitosa.

Código:

```
#include <sys/socket.h>
#include <sys/types.h>

int connect( int sockfd, struct sockaddr *cliente, int longitud );
```

int sockfd :: descriptor de archivo.

struct sockaddr *cliente :: puntero a **struct sockaddr** la cual contendrá nuestros datos.

int longitud :: longitud de **struct sockaddr**.

A continuación se presenta un código de ejemplo para su uso:

Código 004.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PUERTO 1414

int main()
	{
	int sockfd;
	struct sockaddr_in cliente;

	cliente.sin_family= AF_INET; //familia de direcciones
	cliente.sin_port= htons(PUERTO); //NBO
	inet_aton( "127.0.0.1", &cliente.sin_addr ); //ip en formato de Red
	memset( &cliente.sin_zero, '\0', 8 ); //rellenado de variable

	system( "clear" );

	//creamos socket para obtener descriptor
	if( (sockfd=socket( AF_INET, SOCK_STREAM, 0 ))==-1)
		{
		fprintf( stderr, "Problemas para Crear Socket :: %s", strerror(errno) );
		exit(1);
		}
	else
		printf( "\nSOCKET ..... [OK]" );

	if( connect( sockfd, (struct sockaddr *)&cliente, sizeof(struct sockaddr) ) == -1 )
		{
		fprintf( stderr, "\nProblemas para Conectar al Socket :: %s", strerror(errno) );
		exit(1);
		}
	else
```

```
                printf( "\nCONNECT ..... [OK]" );

        printf( "\n\nPulsa para salir...." );
        getchar();
        return 0;
        }
```

3.6.8 FUNCION SEND().

Es la función para enviar datos sobre sockets de flujo (SOCK_STREAM) o sockets de datagramas (SOCK_DGRAM) conectados. Para envío de datos en sockets de datagramas (SOCK_DGRAM) desconectados utiliza la función **sendto()**. En caso de error retorna -1 y establece un valor en la variable "errno". En caso de éxito retorna el número de bytes enviados.

Código:

```
#include <sys/socket.h>
#include <sys/types.h>

int send( int sockfd, const void *buffer, int longitud, int bandera );
```

int sockfd :: Descriptor de archivo.

const void *buffer :: puntero a la variable que contendrá los datos.

int *longitud :: longitud del buffer que contendrá los datos.

int bandera :: Como recomendación 0 para que tome la mejor decisión en el envío de los datos. Pero a continuación algunos valores de bandera disponibles:

MSG_CONFIG :: Indica que esperará mensaje de confirmación de los datos recibidos con éxito que se enviaron por SEND(). Solo válido para **SOCK_DGRAM** y **SOCK_RAW**.

MSG_DONTROUTE :: Indica que no se utilizará la pasarela (gateway) para enviar los paquetes, solo enviará los datos directamente al host conectado a la red. Utilizado generalmente para diagnóstico o programas de ruteo.

MSG_DONTWAIT :: Habilita la operación de no-bloqueo, si la aplicación fue bloqueada en el intento, éste retorna un valor de error **EAGAIN**.

MSG_EOR :: Termina una grabación soportada solo por sockets de tipo **SOCK_SEQPAKET**.

MSG_MORE :: Indica que tiene más datos para enviar. Bandera utilizada por sockets TCP.

MSG_NOSIGNAL :: Solicita que no se envía **SIGPIPE** en caso de error en conectores orientados a la conexión cuando el otro conector rompa la conexión.

MSG_OOB :: Envía los datos en fuera-de-orden (**out-of-band**) en conectores de tipo **SOCK_STREAM**.

A continuación se muestra un código de ejemplo para su uso:

Código 006_c.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PUERTO 1414
#define N 1024

//prototipos
void inicializar( char *x );

void inicializar( char *x )
	{
	int i;

	for( i=0; i<N; i++ )
		x[i]='\0';
	}

int main()
	{
	int sockfd, size_c=0;
	struct sockaddr_in servidor;
	char buf[N];

	servidor.sin_family= AF_INET;
	servidor.sin_port= htons(PUERTO);
	inet_aton( "127.0.0.1", &servidor.sin_addr );
	memset( &servidor.sin_zero, '\0', 8 );

	system( "clear" );
	if( (sockfd=socket( AF_INET, SOCK_STREAM, 0 ))==-1 )
		{
		fprintf( stderr, "Problemas para Crear Socket :: %s", strerror(errno) );
		exit(1);
		}
	else printf( "SOCKET ..... [OK]" );

	size_c= sizeof(struct sockaddr);

	if( connect( sockfd, (struct sockaddr *)&servidor, sizeof(struct sockaddr))==-1 )
		{
		fprintf( stderr, "\nProblemas para Aceptar Cliente :: %s", strerror(errno) );
		exit(1);
		}
	else printf( "\nCONNECT ..... [OK]" );

	inicializar(buf);

	printf( "\n\nMensjae a Enviar: " );
	fgets( buf, sizeof(buf), stdin );
	buf[strlen(buf)-1]='\0';

	if( send( sockfd, buf, sizeof(buf), 0 )==-1 )
		{
		fprintf( stderr, "\bProblemas para envar datos :: %s", strerror(errno) );
		exit(1);
		}
	else
		{
		printf( "\nSEND ..... [OK]" );
```

```
                printf( "\n\n\t****************** MENSAJE **************\n" );
                printf( "\t\t%s", buf );
                printf( "\n\t***********************************************" );
                }

        close(sockfd);
        printf( "\n\nFin de la aplicacion..." );
        getchar();
        return 0;
        }
```

3.6.9 FUNCION RECV().

Función para recepción de datos sobre sockets de flujo (SOCK_STREAM) o sockets de datagramas (SOCK_DGRAM) conectados, para recepción de datos sobre sockets de datagramas (SOCK_DGRAM) desconectados utilizar la función **recfrom()**. En caso de error retorna -1 y fija la variable errno con un valor y encaso de éxito retorna en número de bytes recibidos

Código:

```
#include <sys/socket.h>
#include <sys/types.h>

int recv( int clientefd, void *buffer, int longitud, int bandera );
```

int clientefd :: Descriptor de archivo.

void *buffer :: Puntero a la variable donde guardaremos los datos recibidos.

int longitud :: longitud de la variable buffer.

int bandera :: Como recomendación 0 para que tome la mejor decisión en la recepción de los datos. Algunos valores de bandera disponibles son las siguientes:

MSG_DONTWAIT :: Habilita la operación de no-bloqueo. Si la aplicación ha sido bloqueada, retorna un valor de **EAGAIN**.

MSG_ERRQUEUE :: Indica que los errores que estén en cola deben ser recibidos desde la cola de errores de conectores. El mensaje se suministrará en una estructura tipo **struct sock_extended_err** y se le debe dar un buffer de suficientes dimensiones para su almacenamiento.

MSG_OOB :: Indica la recepción de datos de fuera-de-banda (**out-of-band**) y no se recibirán en el flujo de datos normal.

MSG_PEEK :: Indica que la operación de recepción devuelva datos del principio de la cola de recepción sin quitarlos de allí

MSG_TRUNC :: Retorna la longitud real del paquete.

MSG_WAITALL :: Indica que la aplicación se bloquee hasta que se satisfaga la operación por completo.

Código 006.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PUERTO 1414
#define LIMITE 10
#define N 1024

//prototipos
void inicializar( char *x );

void inicializar( char *x )
	{
	int i;

	for( i=0; i<N; i++ )
		x[i]='\0';
	}

int main()
	{
	int sockfd, clientfd, size_c=0, bytes_r=0;
	struct sockaddr_in servidor;
	struct sockaddr_in cliente;
	char buf[N];

	servidor.sin_family= AF_INET;
	servidor.sin_port= htons(PUERTO);
	inet_aton( "127.0.0.1", &servidor.sin_addr );
	memset( &servidor.sin_zero, '\0', 8 );

	system( "clear" );
	if( (sockfd=socket( AF_INET, SOCK_STREAM, 0 ))==-1 )
		{
		fprintf( stderr, "Problemas para Crear Socket :: %s", strerror(errno) );
		exit(1);
		}
	else printf( "SOCKET ..... [OK]" );

	if( bind( sockfd, (struct sockaddr *)&servidor, sizeof(struct sockaddr) )==-1 )
		{
		fprintf( stderr, "\nProblemas para Asociar Socket :: %s", strerror(errno) );
		exit(1);
		}
	else printf( "\nBIND ..... [OK]" );

	if( listen( sockfd, LIMITE )==-1 )
		{
		fprintf( stderr, "\nProblemas para Escuchar en el Socket :: %s", strerror(errno) );
		exit(1);
		}
	else printf( "\nLISTEN ..... [OK]" );

	size_c= sizeof(struct sockaddr);

	if( (clientfd= accept( sockfd, (struct sockaddr *)&cliente, &size_c ))==-1 )
		{
		fprintf( stderr, "\nProblemas para Aceptar Cliente :: %s", strerror(errno) );
		exit(1);
		}
	else printf( "\nACCEPT ..... [OK]" );

	inicializar(buf);

	if( (bytes_r=recv( clientfd, buf, sizeof(buf), 0 ))==-1 )
		{
```

```
                fprintf( stderr, "\bProblemas para recivir datos :: %s", strerror(errno) );
                exit(1);
                }
        else
                {
                //buf[strlen(buf)]='\0';

                printf( "\nRECV ..... [OK]" );
                printf( "\n\n\t********** MENSAJE **********\n" );
                printf( "\t\t%s", buf );
                printf( "\n\t************************************************" );

                printf( "\nBytes Recividor: %i", bytes_r );
                printf( "\nLongitud del Buffer: %i", strlen(buf) );
                }

        close(clientfd);
        close(sockfd);
        printf( "\n\nFin de la aplicacion.." );
        getchar();
        return 0;
        }
```

3.6.10 FUNCION SENDTO().

Función para el envío de datos sobre sockets de tipo sockets de datagrama (SOCK_DGRA) desconectados, en caso de error retorna un -1 y establece un valor en la variable “errno”. En caso de éxito retorna en número de bytes enviados.

Código:

```
#include <sys/socket.h>
#include <sys/types.h>

int sendto( int sockfd, const void *buffer, int longitud, unsigned int bandera, const struct sockaddr *servidor, int longitud_servidor );
```

int sockfd :: Descriptor de archivo.

const void *buffer :: Puntero a la variable que almacena los datos a enviar.

int longitud :: tamaño de los datos a enviar.

unsigned int bandera :: Como recomendación 0 para que tome la mejor opción para el envío de los datos. A continuación algunos valores de bandera disponibles:

MSG_CONFIG :: Indica que esperará mensaje de confirmación de los datos recibidos con éxito que se enviaron por SEND(). Solo válido para **SOCK_DGRAM** y **SOCK_RAW**.

MSG_DONTROUTE :: Indica que no se utilizará la pasarela para enviar los paquetes, solo enviará los datos directamente al host conectado a la red. Utilizado generalmente para diagnóstico o programas de ruteo.

MSG_DONTWAIT :: Habilita la operación de no-bloqueo, si la aplicación fue bloqueada en el intento, éste retorna un valor de error **EAGAIN**.

MSG_EOR :: Termina una grabación soportada solo por sockets de tipo **SOCK_SEQPAKET**.

MSG_MORE :: Indica que tiene más datos para enviar. Bandera utilizada por sockets TCP.

MSG_NOSIGNAL :: Solicita que no se envía **SIGPIPE** en caso de error en conectores orientados a la conexión cuando el otro conector rompa la conexión

MSG_OOB :: Envía los datos en fuera-de-orden (**out-of-band**) en conectores de tipo **SOCK_STREAM**.

const struct sockaddr *servidor :: puntero a la estructura que contiene los datos del servidor.

int longitud_servidor :: longitud de la estructura.

3.6.11 FUNCION RECVFROM().

Función para la recepción de datos sobre sockets de tipo Sockets de Datagrama (SOCK_DGRAM) desconectados, en caso de error retorna un -1 y establece un valor en la variable "errno". En caso de éxito retorna en número de bytes recibidos.

Código:

```
#include <sys/socket.h>
#include <sys/types.h>

int recvfrom( int sockfd, void *buffer, int longitud, int bandera, struct sockaddr *servidor, int longitud_servidor );
```

int sockfd :: Descriptor de archivo.

void *buffer :: puntero a la variable que almacenará los datos recibidos.

int longitud :: tamaño del buffer disponible.

int bandera :: Como recomendación 0 para que tome la mejor opción en la recepción de los datos. Algunos valores de bandera disponibles son los siguientes:

MSG_DONTWAIT :: Habilita la operación de no-bloqueo. Si la aplicación ha sido bloqueada, retorna un valor de **EAGAIN**.

MSG_ERRQUEUE :: Indica que los errores que estén en cola deben ser recibidos desde la cola de errores de conectores. El mensaje se suministrará en una estructura tipo **struct sock_extended_err** y se le debe dar un buffer de suficientes dimensiones para su almacenamiento.

MSG_OOB :: Indica la recepción de datos de fuera-de-banda (**out-of-band**) no se recibirán en el flujo de datos normal.

MSG_PEEK :: Indica que la operación de recepción devuelva datos del principio de la cola de recepción sin quitarlos de allí

MSG_TRUNC :: Retorna la longitud real del paquete.

MSG_WAITALL :: Indica que la aplicación se bloquee hasta que se satisfaga la operación por completo.

struct sockaddr *servidor :: puntero a la estructura que contiene los datos del servidor.

int longitud_servidor :: tamaño de la estructura.

3.6.12 FUNCION SELECT().

Función que proporciona un método para vigilar un conjunto de descriptores de archivo, indicando que él o los descriptores de archivos están listos para las siguientes tres opciones:

- La lectura puede realizarse sin bloqueo.
- La escritura puede realizarse sin bloqueo.
- La existencia de una operación excepcional pendiente (indica la presencia de un dato fuera de banda durante la comunicación por la red).

Código:

```
#include <time.h>
#include <sys/types.h>

int select( int totalfds, fd_set *lecturafds, fd_set *escriturafds, fd_set *excepcionfds, struct timeval *tiempomaximo );
```

int totalfds :: Número del bit que pertenece al descriptor de archivo mayor, debe darse el valor del bit mayor más uno.

fd_set *lecturafds :: Conjunto de descriptores a vigilar para operaciones de lectura.

fd_set *escriturafds :: Conjunto de descriptores a vigilar para operaciones de escritura.

fd_set *excepcionfds :: Conjunto de descriptores a vigilarse para condiciones excepcionales.

struct timeval *tiempomaximo :: valor de tiempo que forzara a select retornar los bits disponibles en cada uno de sus operaciones.

En caso de éxito borra los bits de cada uno de los **fd_set** que no estén disponibles para su operación, dejando solamente los que están listos. En caso de error retorna un -1 y establece un valor de error en la variable “errno”.

Existen macros para manejar un conjunto de bits/descriptores o bien, ir agregando bits/descriptores a un número de bits ya existentes. Para ello existen las funciones siguientes:

Código:

```
#include <time.h>
#include <sys/types.h>

void FD_SET( int sockfd, fd_set *fds );
void FD_CLR( int sockfd, fd_set *fds, );
void FD_ZERO( fd_set *fds );
int FD_ISSET( int sockfd, fd_set *fds );
```

void FD_SET :: Guarda el descriptor de archivo (**int sockfd**) dentro del conjuntos de desciptores **fd_set *fds**.

void FD_CLR :: Borra el descriptor de archivo (**int sockfd**) contenido en el conjunto de descriptores **fd_set *fds**.

void FD_ZERO :: Vacia el contenedor de descriptores **fd_set *fds**.

int FD_ISSET :: Determina si el descriptor de archivo **int sockfd** existe dentro del contenedor de descriptores de archivos **fd_set *fds**.

A continuación se muestra un código de ejemplo (cliente y servidor) para su uso:

Código 007.c:

```
void *enviar_datos( void *sock )
	{
	char buf[N];
	int clientds, i;
	clientds= (int)sock;

	for(;;)
		{
		memset( &buf, '\0', N );
		printf( "\nEscribe: " );
		fgets( buf, N, stdin );

		if( buf=="salir" )
			pthread_exit(0);
		else
			{
			buf[strlen(buf)-1]='\0';
			if( send( clientds, buf, strlen(buf), 0 )==-1 )
				fprintf( stderr, "\nProblema al Enviar Datos :: %s", strerror(errno) );
			}
		}
	}

void *recivir_datos( void *sock )
	{
	char buf[N];
	int clientds, bytes_r=0;
	clientds= (int)sock;

	do
		{
		memset( &buf, '\0', N );
		bytes_r=0;

		if( (bytes_r=recv( clientds, buf, N, 0 ))==-1 )
			fprintf( stderr, "\nProblemas para Recivir Datos :: %s", strerror(errno) );
		else
			{
			if( bytes_r==0 )
				{
				printf( "--El Cliente Forzo el Cierre --\n" );
```

```
                                        pthread_exit(0);
                                        }
                                else if( bytes_r==-1 )
                                        {
                                        printf( "-- El Cliente Salio ---\n" );
                                        pthread_exit(0);
                                        }
                                else
                                        printf( "%s\n\r", buf );
                                }
                        }while( bytes_r!=0 && bytes_r!=-1 );
                }

void inicia_servidor( void )
                {
                int sockds, clientds, bytes_r=0;
                struct sockaddr_in servidor;
                struct sockaddr_in cliente;
                int size_c=0, aux=1;
                pthread_t hilo_s, hilo_r;

                servidor.sin_family= AF_INET;
                servidor.sin_port= htons(PUERTO);
                inet_aton( "127.0.0.1", &servidor.sin_addr );
                memset( &servidor.sin_zero, '\0', 8 );

                if( (sockds=socket( AF_INET, SOCK_STREAM, 0 ))==-1 )
                        fprintf( stderr, "\nProblemas para Crear Socket :: %s", strerror(errno) );
                else
                        {
                        if( setsockopt( sockds, SOL_SOCKET, SO_REUSEADDR, &aux, sizeof(int) )==-1 )
                                fprintf( stderr, "\nProblemas para Reutilizar Direccion :: %s", strerror(errno) );

                        if( bind( sockds, (struct sockaddr *)&servidor, sizeof(struct sockaddr) )==-1 )
                                fprintf( stderr, "\nProblemas para Asociar Socket :: %s", strerror(errno) );
                        else
                                {
                                printf( "\nEsperando Cliente.....\n" );
                                if( listen( sockds, LIMITE )==-1 )
                                        fprintf( stderr, "\nProblemas para Escuchar en el Socket :: %s", strerror(errno) );
                                else
                                        {
                                        size_c= sizeof(struct sockaddr);
                                        if( (clientds= accept( sockds, (struct sockaddr *)&cliente, &size_c ))==-1 )
                                                fprintf( stderr, "\nProblemas para Aceptar Cliente :: %s", strerror(errno) );
                                        else
                                                {
                                                if( pthread_create( &hilo_r, NULL, recivir_datos, (void *)clientds )!=0 )
                                                        fprintf( stderr, "\nProblemas para crear Hilo de Recepcion :: %s",
strerror(errno) );
                                                if( pthread_create( &hilo_s, NULL, enviar_datos, (void *)clientds )!=0 )
                                                        fprintf( stderr, "\nProblemas para crear Hilo de Teclado :: %s",
strerror(errno) );

                                                pthread_join( hilo_r, NULL );
                                                pthread_join( hilo_s, NULL );
                                                }
                                        close(clientds);
                                        }
                                }
                        close(sockds);
                        }
                }
```

3.6.13 FUNCION GETPEERNAME().

Función que proporciona información sobre quién está conectado a nuestro socket o bien del otro lado del socket donde estamos invocando dicha función En caso de error retorna -1 y establece un valor en la variable "errno", en caso de éxito retorna la dirección IP en formato de red y se debe utilizar la función **inet_ntoa()** para interpretarla de forma numérica con puntos como delimitante.

Código:

```
#include <unistd.h>

int getpeername( int sockfd, struct sockaddr *addr, int *longitud );
```

int sockfd :: Descriptor de archivo.

struct sockaddr *addr :: puntero a la estructura donde se guarda la información

int *longitud :: dimensión del puntero *addr.

A continuación un código de ejemplo para su uso:

Código 008.c (recortado):

```
//SERVIDOR

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define N 1024 //buffer 1MB
#define PUERTO 1414 //puerto
#define LIMITE 10 //solo 10 en cola

int main()
        {
        int servfd, clientfd, size_c=0, size_a=0, bytes_r=0;
        char buf[N];
        struct sockaddr_in servidor;
        struct sockaddr_in cliente;
        struct sockaddr_in auxiliar; //para obtener informacion al vuelo

        servidor.sin_family= AF_INET;
        servidor.sin_port= htons(PUERTO);
        inet_aton( "127.0.0.1", &servidor.sin_addr );
        memset( &servidor.sin_zero, '\0', 8 );
        memset( &buf, '\0', N );

        system( "clear" );
        if( (servfd=socket( AF_INET, SOCK_STREAM, 0 ))==-1 )
                {
                fprintf( stderr, "Problemas para crear Socket :: %s", strerror(errno) );
                exit(1);
                }
        else    printf( "SOCKET ..... [OK]" );

        if( bind( servfd, (struct sockaddr *)&servidor, sizeof(struct sockaddr) )==-1 )
                {
                fprintf( stderr, "\nProblemas para Asociar Socket :: %s", strerror(errno) );
                exit(1);
```

```
		}
	else printf( "\nBIND ..... [OK]" );

	if( listen( servfd, LIMITE )==-1 )
		{
		fprintf( stderr, "\nProblemas para Escuchar en Socket :: %s", strerror(errno) );
		exit(1);
		}
	else printf( "\nLISTEN ..... [OK]" );

	size_c= sizeof(struct sockaddr);
	if( (clientfd=accept( servfd, (struct sockaddr *)&cliente, &size_c ))==-1 )
		{
		fprintf( stderr, "\nProblemas para Aceptar Cliente :: %s", strerror(errno) );
		exit(1);
		}
	else
		{
		if( send( clientfd, "\n--> Conexion Establecida\0", 26, 0 )==-1 )
			fprintf( stderr, "\nProblemas para Enviar :: %s", strerror(errno) );
		else
			{
			if( (bytes_r=recv( clientfd, buf, sizeof(buf), 0 ))==-1 )
				fprintf( stderr, "\nProblemas para Recivir :: %s", strerror(errno) );
			else
				{
				//mensaje de bienvenida
				printf( "\n\n%s-\n", buf );

				//A continuacion descomentar para activar servicio

				//*-------------------------------------------------------------------------*
				//|		funcion getpeername();
		|
				//|		Proporciona informacion sobre el cliente del otro lado del socket. |
				//*-------------------------------------------------------------------------*
				//-->INICIO de funcion.
				//
				printf( "\nObteniendo IP del cliente..." );

				size_a= sizeof( struct sockaddr );
				if( getpeername( clientfd, (struct sockaddr *)&auxiliar, &size_a )==-1 )
					{
					fprintf( stderr, "\nProblemas para obtener IP del Cliente :: %s", strerror(errno) );
					}
				else
					printf( "\nIP Cliente: %s", inet_ntoa(auxiliar.sin_addr) );
				//
				//--> FIN de funcion.
				}
			}
		close(clientfd);
		}

	close(servfd);

	printf( "\n\nPulsa para salir..." );
	getchar();
	return 0;
	}
```

Codigo 008_c.c (recortado):

```
//CLIENTE

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PUERTO 1414
#define N 1024

int main()
	{
	int sockfd, bytes_r=0, size_c=0, size_a=0;
	struct sockaddr_in cliente;
	struct sockaddr_in auxiliar;
	char buf[N];

	cliente.sin_family= AF_INET;
	cliente.sin_port= htons(PUERTO);
	inet_aton( "127.0.0.1", &cliente.sin_addr );
	memset( &cliente.sin_zero, '\0', 8 );
	memset( &buf, '\0', N );

	system( "clear" );
	if( (sockfd=socket( AF_INET, SOCK_STREAM, 0 ))==-1 )
		{
		fprintf( stderr, "Problemas para Crear Socket :: %s", strerror(errno) );
		exit(1);
		}
	else printf( "SOCKET ..... [OK]" );

	if( connect( sockfd, (struct sockaddr *)&cliente, sizeof(struct sockaddr) )==-1 )
		{
		fprintf( stderr, "\nProblema para Conectar al Socket :: %s", strerror(errno) );
		exit(1);
		}
	else
		{
		printf( "\nCONNECT ..... [OK]" );

		if( (bytes_r=recv( sockfd, buf, sizeof(buf), 0)) == -1 )
			fprintf( stderr, "\nProblemas para Recivir :: %s", strerror(errno) );
		else
			{
			if( send( sockfd, "\n--> Conexion Extablecida\0", 26, 0 )==-1 )
				fprintf( stderr, "\nProblemas para Enviar :: %s", strerror(errno) );
			else
				{
				//mensaje de bienvenida
				printf( "\n\n%s\n", buf );

				//A continuacion descomentar para activar servicio

				//*-------------------------------------------------------------------------*
				//|		funcion getpeername();
		|
				//|		Proporciona informacion sobre el cliente del otro lado del socket. |
				//*-------------------------------------------------------------------------*
				//-->INICIO de funcion.
				//
				printf( "\nObteniendo IP del cliente..." );

				size_a= sizeof( struct sockaddr );
				if( getpeername( sockfd, (struct sockaddr *)&auxiliar, &size_a )==-1 )
					{
					fprintf( stderr, "\nProblemas para obtener IP del Cliente :: %s", strerror(errno) );
```

```
                                        }
                                else
                                        printf( "\nIP Servidor: %s", inet_ntoa(auxiliar.sin_addr) );
                                //
                                //--> FIN de funcion.
                                }
                        }

                close(sockfd);
                }
        printf( "\n\nPulsa para salir..." );
        getchar();
        return 0;
        }
```

3.6.14 FUNCION GETHOSTNAME().

Función que proporciona el nombre de la computadora. En caso de error retorna un -1 y establece un valor en la variable “errn”, en caso de éxito retorna un 0.

Código:

```
#include <unistd.h>

int gethostname( char *buf, size_t dimension );
```

char *buf :: puntero que apuntará al nombre del host.

size_t dimensión :: tamaño de la variable que contendrá el nombre del host.

A continuación un código de ejemplo para su uso:

Código 008.c (recortado):

```
//SERVIDOR

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>

int main()
        {
        system( "clear" );
        char nombre[50];
        int size_n=0;

        printf( "\nObteniendo Nombre del Cliente...." );

        size_n= sizeof(nombre);
        if( gethostname( nombre, size_n )==-1 )
                fprintf( stderr, "\nProblemas para Obtener Nombre del Cliente :: %s", strerror(errno) );
        else
                printf( "\nNombre del Cliente es: %s", nombre );

        printf( "\n\nPulsa para salir..." );
        getchar();
        return 0;
        }
```

3.6.15 FUNCION GETDOMAINNAME().

Función que proporciona el nombre del dominio de la computadora. En caso de error retorna un -1 y establece un valor en la variable “errno”, en éxito retorna un 0.

Código:

```
#include <unistd.h>

int getdomainname( char *buf, size_t longitud );
```

char *buf :: puntero que contendrá los datos.

size_t longitud :: dimensión del puntero que contendrá los datos.

A continuación un código de ejemplo para su uso:

Código 008.c(recortado):

```
//SERVIDOR

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd>

int main()
	{
	system( "clear" );
	char nombre[50];
	int size_n=0;

	printf( "\nObteniendo Nombre de Dominio del Cliente..." );

	size_n= sizeof(nombre);
	if( getdomainname( nombre, size_n )==-1 )
		fprintf( stderr, "\nProblemas para Obtener Nombre de Dominio del Cliente :: %s", strerror(errno) );
	else
		printf( "\nNombre de Dominio del Cliente es: %s", nombre );
	}

	printf( "\n\nPulsa para salir..." );
	getchar();
	return 0;
	}
```

3.6.16 FUNCION GETHOSTID().

Función que proporciona el número de identificación de la computadora. En caso de error retorna un -1 y establece un valor en la variable “errno”, en caso de éxito retorna el número de identificación de 32 bits.

Código:

```
#include <unistd.h>

long int gethostid( void );
```

A continuación un código de ejemplo para su uso:

Código 008.c(recortado):

```
//SERVIDOR

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define N 1024 //buffer 1MB
#define PUERTO 1414 //puerto
#define LIMITE 10 //solo 10 en cola

int main()
	{
	int servfd, clientfd, size_c=0, bytes_r=0;
	char buf[N];
	struct sockaddr_in servidor;
	struct sockaddr_in cliente;

	servidor.sin_family= AF_INET;
	servidor.sin_port= htons(PUERTO);
	inet_aton( "127.0.0.1", &servidor.sin_addr );
	memset( &servidor.sin_zero, '\0', 8 );
	memset( &buf, '\0', N );

	system( "clear" );
	int id_h=0;

	printf( "\nObteniendo ID: " );

	if( (id_h=gethostid())==-1 )
		fprintf( stderr, "\nProblemas para obtener id :: %s", strerror(errno) );
	else
		printf( "\nEl ID es: %i", id_h );

	printf( "\n\nPulsa para salir..." );
	getchar();
	return 0;
	}
```

3.6.17 FUNCION GETSOCKNAME().

Función que proporciona el nombre actual de la conexión al socket, en caso de error retorna un -1 y establece un valor en la variable “errno”, en éxito un cero.

Código:

```
#include <socket.h>

int getsockname( int sockfd, struct sockaddr *nombre, socklen_t *longitud );
```

int sockfd :: Descriptor de archivo.

struct sockaddr *nombre :: Apuntador que contendrá los datos.

socklen_t *longitud :: Dimensión del apuntador.

A continuación un código de ejemplo para su uso:

Código 008.c (recortado):

```
//SERVIDOR

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define N 1024 //buffer 1MB
#define PUERTO 1414 //puerto
#define LIMITE 10 //solo 10 en cola

int main()
	{
	int servfd, clientfd, size_c=0, bytes_r=0;
	char buf[N];
	struct sockaddr_in servidor;
	struct sockaddr_in cliente;

	servidor.sin_family= AF_INET;
	servidor.sin_port= htons(PUERTO);
	inet_aton( "127.0.0.1", &servidor.sin_addr );
	memset( &servidor.sin_zero, '\0', 8 );
	memset( &buf, '\0', N );

	system( "clear" );
	if( (servfd=socket( AF_INET, SOCK_STREAM, 0 ))==-1 )
		{
		fprintf( stderr, "Problemas para crear Socket :: %s", strerror(errno) );
		exit(1);
		}
	else	printf( "SOCKET ..... [OK]" );

	if( bind( servfd, (struct sockaddr *)&servidor, sizeof(struct sockaddr) )==-1 )
		{
		fprintf( stderr, "\nProblemas para Asociar Socket :: %s", strerror(errno) );
		exit(1);
		}
	else printf( "\nBIND ..... [OK]" );

	if( listen( servfd, LIMITE )==-1 )
		{
```

```
		fprintf( stderr, "\nProblemas para Escuchar en Socket :: %s", strerror(errno) );
		exit(1);
		}
	else printf( "\nLISTEN ..... [OK]" );

	size_c= sizeof(struct sockaddr);
	if( (clientfd=accept( servfd, (struct sockaddr *)&cliente, &size_c ))==-1 )
		{
		fprintf( stderr, "\nProblemas para Aceptar Cliente :: %s", strerror(errno) );
		exit(1);
		}
	else
		{
		printf( "\nACCEPT ..... [OK]" );
		if( send( clientfd, "\n--> Conexion Establecida\0", 26, 0 )==-1 )
			fprintf( stderr, "\nProblemas para Enviar :: %s", strerror(errno) );
		else
			{
			if( (bytes_r=recv( clientfd, buf, sizeof(buf), 0 ))==-1 )
				fprintf( stderr, "\nProblemas para Recivir :: %s", strerror(errno) );
			else
				{
				//mensaje de bienvenida
				printf( "\n\n%s\n", buf );

				//*---------------------------------------------------------------------------*
				//|		funcion getsockname();
		|
				//|		Proporciona informacion sobre el cliente del otro lado del socket. |
				//*---------------------------------------------------------------------------*
				//-->INICIO de funcion.
				//
				struct sockaddr_in nombre;
				int size_n=0;

				printf( "\nObteniendo nombre del socket..." );

				size_n= sizeof(struct sockaddr);
				if( getsockname( clientfd, (struct sockaddr *)&nombre, size_n )==-1 )
					fprintf( stderr, "\nProblemas para obtener nombre del socket :: %s", strerror(errno) );
				else
					printf( "\nNombre del Socket es: %s", nombre );
				//
				//--> FIN de funcion.
				}
			}
		close(clientfd);
		}

	close(servfd);

	printf( "\n\nPulsa para salir..." );
	getchar();
	return 0;
	}
```

PROTOCOLOS DE COMUNICACION

4

4. PROTOCOLOS DE COMUNICACION 100
4.1 PROTOCOLO DE INTERNET (IP) 100
4.2 PROTOCOLO DE MENSAJES DE CONTROL EN INTERNET (ICMP). 103
4.3 PROTOCOLO DE DATAGRAMAS DE USUARIOS (UDP). 105
4.4 PROTOCOLO DE CONTROL DE TRANSMICION (TCP) 107
4.5 PROTOCOLO RAW 110

4. PROTOCOLOS DE COMUNICACION

Estos protocolos son estándares establecidos y documentados en sus RFC numerados en base a su creación de documento y también contando con actualizaciones en su estándar y documento *[30]*.

La función de los distintos protocolos existentes es brindar una forma de comunicación entre equipos de cómputo, pero cada tipo de protocolo brinda distintas formas de formación de un paquete, así también sus niveles de seguridad en base a la fiabilidad de los datos, formas de conexión, estructuración de sus capas, así mismo cada protocolo podrá diferenciarse de los demás.

En los distintos protocolos existen tres datos que nunca deben de faltar, que son: dirección origen, dirección destino y datos. Apartir de esto, los campos agregados al paquete son los que identificarán el tipo de protocolo que se utiliza según el estándar que lleve al organizar los campos y la información que utiliza.

Existen protocolos ya estandarizados a nivel universal como son: TCP, UDP, IP, ICMP y RAW, estos son utilizados para la comunicación en muchas aplicaciones de red, pero cada una tiene sus pros y sus contras. En seguida se verá una mejor explicación y formación de estos paquetes, pero se debe resaltar que para realizar o manipularlos, es necesario tener conocimiento de la programación en red (Sockets), así mismo se podrá hasta estandarizar nuestro propio protocolo.

4.1 PROTOCOLO DE INTERNET (IP)

El Protocolo de Internet (IP), es un protocolo orientado a lano conexión, realizando operaciones de comunicación y conmutación de paquetes, estos paquetes son conocidos como bloques o datagramas los cuales se retransmiten hasta que el destino responda a la recepción del paquete datagrama. Los datagramas pueden ser un medio de comunicación no fiable, ya que no garantiza el éxito de la recepción y no tienen un orden de llegada, por lo que el origen deberá armar los paquetes decepcionados.

A continuación se presenta el esquema de la cabecera IP en la Figura 4.1.

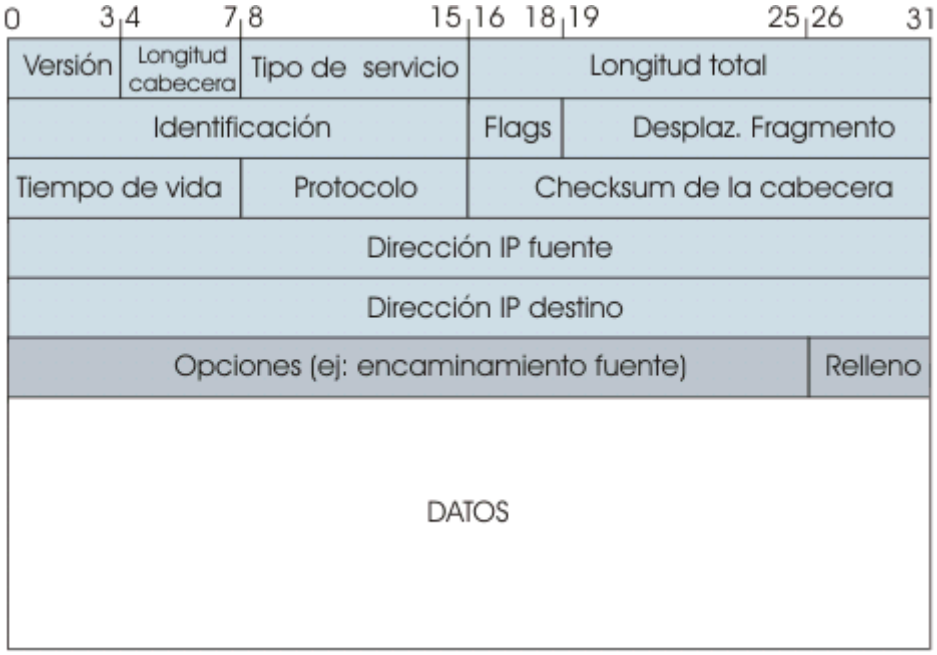

FIGURA 4.1: Esquema de Cabecera IP.

Junto al esquema, ahora veamos la definición de la estructura IP previamente rellenada para ilustrarse con el esquema de arriba.

Código:

```
#typedef unsigned int uint;
#typedef unsigned char uchar;

struct ip_packet
        {
        uint version:4;    //version de 4 bits Ipv4
        uint header_len:4;    //tamano de cabecera en words
        uint serve_type:8;    //como servir el paquete
        uint packet_len:16;    //tamano total del paquete en bytes
        uint ID:16;    //ID del paquete
        uint __reserved:1;    //siempre cero
        uint dont_frag:1;    //permitir fragmentacion
        uint more_frags:1;    //continuacion de fragmentos
        uint frag_offset:13;    //ayudar a la recomposicion
        uint time_to_live:8;    //numero de saltos de router permitidos
        uint protocolo:6;    //tipo de protocolo (TCP)
        uint header_chksum:16;    //suma de comprobacion de la cabecera
        uint IPv4_source:32;    //Direccion IP del Origen
        uint IPv4_dest:32;    //Direcion IP del Destino
        uchar options[];    //opcional, hasta 40 bytes
        uchar data[];    //datos del mensaje hasta 64kb
        }
```

Campo Versión.

Establece un número de versión del protocolo IP, algunos valores posibles se muestran en la Tabla 1.

Tabla 1. Versiones del protocolo IP

Versión	Tipo de protocolo
4	IPv4.
5	Modo de Datagrama IP de Flujo (IP Experimental).

6	IPv6.
7	TP/IX (el “próximo” protocolo de Internet).
8	El Protocolo de Internet “P”.
9	TUBA.

Campo Longitud de Cabecera.
Indica al receptor la longitud de la cabecera utilizando words de 32 bits. Desde 0–50 words (60 bytes).

Campo Tipo de Servicio.
Indica como administrar el paquete, posee dos subcampos, campo de precedencia (ignorado en muchos sistemas) y campo de TOS. TOS tiene cuatro opciones, las cuales pueden ser: retardo mínimo, rendimiento máximo, fiabilidad máxima, costo mínimo. Si no se selecciona ningún tipo de servicio, se tomará como administración normal.

Campo ID.
El subsistema IP proporciona un ID único de 16 bits, si se alcanzara el número máximo, el subsistema IP reutiliza los IDs por medio de la hora del sistema. El ID ayuda a recomponer paquetes fragmentados.

Campo DF – Dont Frag.
Si se encuentra activado (1) indica que el mensaje debe ser fragmentado, si no (0) que debe ser aceptado completamente o rechazado.

Campo MF – More Frags.
Indica la continuación de fragmentos del paquete, cuando se reciba un paquete con esta bandera en 0, es señal que no habrá más fragmentos.

Campo Frag Offset.
Es un campo de 13 bits e indica la zona a la que pertenece este fragmento, ya que los paquetes en muchas rutas de la red pueden ser fragmentados y llegar en tiempo distintos, este campo tiene un valor el cual se multiplica por ocho y se obtiene la posición para su recomposición del paquete. Entonces cada paquete es un múltiplo de 8 excepto el último. En caso que el subsistema IP no pueda obtener todos los fragmentos del paquete en un lapso de tiempo especifico, descarta el paquete y envía un mensaje de error al origen.

Campo TTL – Time To Live
Anteriormente este campo servía para contar/establecer el tiempo de vida que podía perdurar un paquete durante su transito por la red, actualmente ahora sirve para indicar el número de saltos que debe dar un paquete durante su tránsito por la red, en cada salto esta bandera es decrementada, así mismo cada salto puede ser dado en un host o router y al llegar a ser 0 antes de llegar a su destino, este re-envía un mensaje de error al origen. Este campo es de 8 bits y permite 255 saltos.

Campo Protocolo.
Es un valor de protocolo asignado, cada protocolo tiene su valor, por mencionar algunos en la Tabla 2:

Tabla 2. Descripción de protocolos.

Número de protocolo	Tipo de protocolo
0	IP.
1	ICMP.
6	TCP.
17	UDP.

Para una mejor referencia ver *'/etc/protocols'* en los sistemas operativos Linux y para todos los archivos mencionados con las mismas caracteristicas.

En la Tabla 2 el número de protocolo indica al subsistema IP y nos indica cómo tratar cada paquete entrante.

Campo Opciones.
Son opciones que pueden pasarse con cada paquete, que van desde información de enrutamiento, marca de tiempo, medidas de seguridad, registros de enrutamiento y alarmas de caminos. Este campo puede ser hasta de 40 bytes.

Campo Datos.
En esta parte se incluye el mensaje y puede ser hasta de 64 bytes.

Información adicional sobre los campos y funcionamientos se puede leer el documento *RFC*-791 o bien [11]**.** Los que utilicen sistemas operativos Linux, pueden tomar mayor referencia o información sobre sus banderas y programación, ver */usr/include/linux/ip.h.*

4.2 PROTOCOLO DE MENSAJES DE CONTROL EN INTERNET (ICMP).

Es un protocolo orientado a la no conexión, utilizando el envío de datagramas como forma en su paquete, el proceso general es enviar un paquete sin establecer una conexión y envía el paquete.

Una de las particularidades de este protocolo es que, se utiliza para dar informes de errores y mensajes de control, por lo general un paquete ICMP nunca es fragmentado por un router, ya que se tiende a utilizar este tipo de paquetes para avisar de un control, error o estado, que por lo normal no llega a ser tan grande.

A continuación veamos la forma de un paquete ICMP en la Figura 4.2.

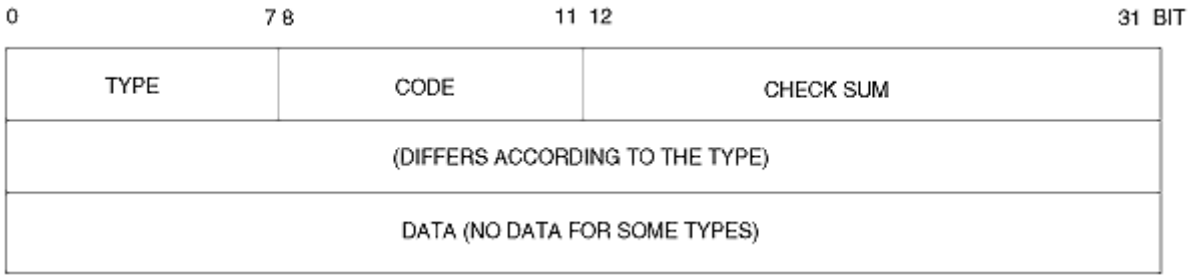

FIGURA 4.2: Paquete ICMP.

Junto a este esquema ahora veamos la estructura *icmp_header* para su uso en programación de sockets.

Código:

```
#typedef unsigned short int uint;
#typedef unsigned char uchar;

struct icmp_header
        {
        uint type;    //tipo de error
        uint code;    //codigo de error
        uchar checksum;    //suma de comprobacion
        uchar msg[];    //datos del mensaje
        }
```

Campo Tipo.
Indica el tipo de mensaje ICMP [2] en la Tabla 3.

Tabla 3. Tabla de Numeros de Mensaje ICMP.

0	Echo replay. (Respuesta de Eco)	11	Time Exceeded. (Tiempo Excedido)
1	Reservado.	12	Parameter problem. (Problema de Parametro)
2	Reservado.	13	Timestamp. (Marca de Tiempo)
3	Destination unreachable. (Destino Inalcanzable)	14	Timestamp replay. (Respuesta de Marca de Tiempo)
4	Source quench. (Apagado de Origen)	15	Information request. (Peticion de Inf.)
5	Redirect. (Redireccionar)	16	Information replay. (Respuesta de Inf.)
6	Direccion Alterna de Host.	17	Netmask reques. (Peticion de Mascara de Red)
7	Reservado.	18	Netmask replay. (Respuesta de Mascara de Red)
8	Echo. (Eco)	19	Reservado para Seguridad.
9	Anuncion de Router.	20-29	Reservado para Experimentos de Robustes.
10	Solicitud de Router.	30	Traceroute. (Trazar Ruta)

Campo código.
Referido al código que corresponde al tipo de mensaje (ver Tabla 3).

Campo suma de comprobación.
Número que permite comprobar la integridad de la cabecera y los datos. Se calcula mediante la suma en complemento a uno de 16 bits de la suma en complemento a uno del paquete ICMP.

Campo Especial.
El contenido de este campo puede variar dependiendo el tipo de mensaje ICMP construido, éste puede ser:

Campo Especial de Datos.
El contenido de este campo puede variar dependiendo el tipo de mensaje ICMP construido, éste puede ser:

Información adicional sobre los campos y funcionamientos se puede leer el documento *RFC*-792 o bien *[12]***.** Los que utilicen sistemas operativos Linux, pueden tomar mayor referencia o información sobre sus banderas y programación, ver */usr/include/linux/icmp.h*.

4.3 PROTOCOLO DE DATAGRAMAS DE USUARIOS (UDP).

Es un protocolo orientado a la no conexión que utiliza paquetes tipo datagramas, no es necesario que se haya realizado previamente una sincronización de conexión (tipo TCP) para enviar datos, solo es necesario crear el paquete UDP y enviarlo al destino.

Cuando se crea un paquete UDP, el núcleo del sistema lo único que realiza es establecer los 3 datos más importantes, que son: dirección origen, dirección destino y datos, lo pone en la red y no se preocupa por la fiabilidad del paquete, es decir no le da importancia si el paquete fue recibido por el recepto o no. Así que esto pasa a formar uno de los puntos débiles de paquetes UDP, ya que este protocolo resulta ser poco fiable al transmitir paquetes.

Una ventaja muy buena que aportan los paquetes UDP son su velocidad de transmisión en la red, cuando un paquete es bien monitoreado o rastreado este suele ser una forma de comunicación muy veloz.

Para que el protocolo UDP sea un poco más fiable es necesario tomar en cuenta algunas cuestiones que son:

- **Dividir el paquete ::** si el paquete es muy grande, entonces dividirlo en fragmentos más pequeños y asignarles un número, así el equipo receptor podrá re-ensamblar el paquete entero.
- **Seguir rastro del paquete ::** como anteriormente se asignó un número a cada paquete, brindar la posibilidad de que, por cada paquete que se transmita, el emisor

debería pedir un acuse de recibo por cada paquete enviado (ACK), de este modo ayudaría a que el equipo receptor podrá recuperar ciertas partes del paquete.

- **Comprobar legitibilidad ::** por cada paquete recibido, el receptor deberá realizar una operación para comprobar los datos, ya sea utilizando una suma de comprobación (checksum) o una comprobación de redundancia cíclica (CRC). De este modo podrá pedir que se le retransmita el paquete en caso que esté con error.
- **Utilizar tiempos de espera ::** consiste en que si el paquete enviado sobrepasa el tiempo asignado, el receptor puede enviar un paquete de petición de paquetes UDP.

A continuación el esquema gráfico de un paquete UDP en la Figura 4.3.

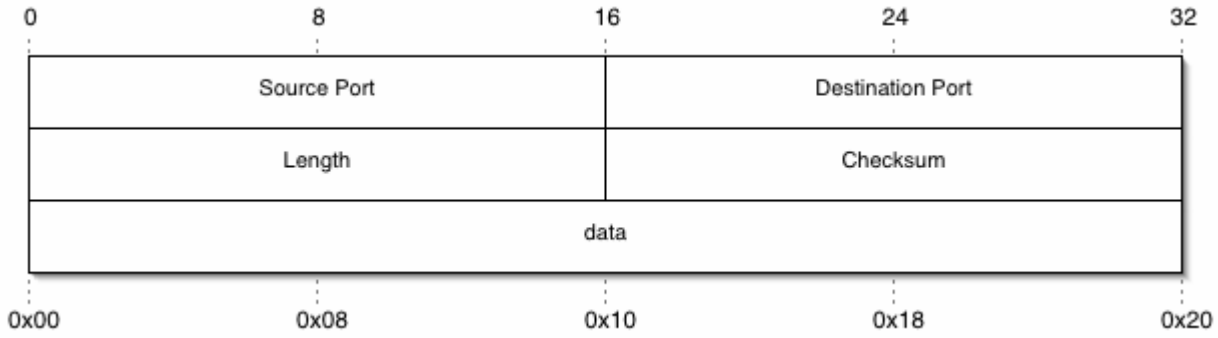

FIGURA 4.3: Paquete UDP.

Ésta es su estructura para programación de sockets,

Codigo:

```
#typedef unsigned short int uint;
#typedef unsigned char uchar;

struct udp_header
        {
        uint src_port;    //puerto de origen
        uint dst_port;    //puerto destino
        uint length;    //longitud del paquete
        uint checksum;    //suma de comprobacion
        uchar data[];    //datos
        }
```

Campo source port.
Es el número de puerto de dónde proviene el paquete UDP.

Campo destination port.
Es el número de puerto a dónde llegará el paquete UDP.

Campo longitud.
Es la longitud del paquete en bytes.

Campo suma de comprobación.
Es un número que permite comprobar que la cabecera de datos no ha llegado con error al destino. Se calcula mediante la suma en complemento a uno de 16 bits de una cabecera

especial que contiene los siguientes datos: dirección IP Origen, dirección IP Destino, campo reservado con valor 0, protocolo y tamaño del paquete.

Campo Datos.
Es la información del paquete.

Información adicional sobre los campos y funcionamientos se puede leer el documento *RFC*-768 o bien [13]**.** Los que utilicen Sistemas Operativos Linux, pueden tomar mayor referencia o información sobre sus banderas y programación, ver */usr/include/linux/udp.h.*

4.4 PROTOCOLO DE CONTROL DE TRANSMICION (TCP)

El Protocolo de Control de Transmisión es un protocolo orientado a la conexión, y requiere que ambas computadoras estén conectadas y listas para intercambiar datos. La forma principal de TCP consta en que un host dentro de la red envía un paquete a otra PC, este paquete es garantizado por TCP (su recepción), ya que por cada paquete enviado, se esperará por una respuesta de satisfacción de recepción del paquete. Así mismo, TCP toma cada paquete y le adjunta un número único.

Cuando el Destino recibe los datos, TCP introduce una suma de comprobación la cual determina la integridad de los datos, en caso de tener errores, el receptor (destino) da un aviso de error y solicita la re-transmisión del paquete, a su vez si el receptor no ha realizado un aviso de recepción exitosa de los datos, el subsistema TCP vuelve a re-enviar el paquete sin la intervención del mismo programa.

TCP para garantizar el envío de paquete, también utiliza limitaciones en el envío de sus paquetes, el cual consta en reducir la dimensión de sus paquetes, estableciéndolo a una dimensión determinada la cual no sea un problema al momento de ser fragmentado por algunos routers. Esta dimensión puede ir de 536 bytes hasta 1,500 bytes, para implementar algún tamaño en específico es necesario establecerlo manualmente, para eso realice un sockets de programación y establezca el valor con **MSS** (tamaño de segmento máximo).

A continuación se presenta el esquema TCP en la Figura 4.4.

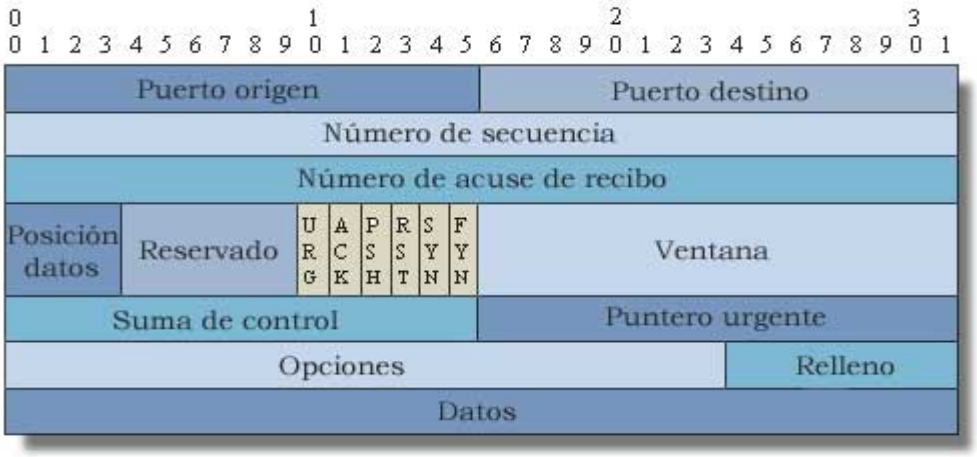

FIGURA 4.4: Paquete TCP.

A su vez, esta es la estructura para la programación de sockets,

Código:

```
#typedef unsigned char ui8;

struct tcp_header
        {
        ui16 src_port;    //puerto de origen
        ui16 dst_post;    //puerto destino
        ui32 seq_num;     //numero de secuencia
        ui32 ack_num;     //numero de acuse de recivo
        uint data_off:4;    //offset de datos
        uint __res:6;    //reservado
        uint urg_flag;    //bandera URG
        uint ack_flag;    //bandera ACK
        uint push_flag;    //bandera PUSH
        uint rst_flag;    //bandera RST
        uint syn_flag;    //bandera SYN
        uint fin_flag;    //bandera FIN
        ui16 window;    //numero de bytes que se permiten recivir
        ui16 checksum;    //suma de comprobacion
        ui16 urg_pos;    //ultimo bytes de un mensaje urgente
        ui8 options[];    //opciones TCP
        ui8 __padding[];    //para alineacion de datos (relleno)
        uchar data[];    //datos
        }
```

Campo Puerto de Origen :: src_port
Indica el puerto de origen.

Campo Puerto de Destino :: dst_port
Indica el puerto de destino.

Campo Número de Secuencia :: seq_num
Número de secuencia para el flujo de continuación de datos y a su vez para ordenar el paquete.

Campo Número de Acuse de Recibo :: acq_num
Número para el acuse de recibo, cada vez que se recibe una parte del paquete éste tiene un número de secuencia, se le debe contestar al host con un paquete que lleve activado el número de acuse de recibo para continuar con el flujo de recepción de datos.

Campo Posición de Datos :: data_off
Como el tamaño de un paquete no se puede medir con exactitud, el campo de posición de datos, nos indica en dónde inician los datos del paquete.

Campo Reservado :: __res
Espacio reservado que siempre está en cero.

Campo URG :: urg_flag
Bandera que activa la opción de URGENTE, al estar activado (con 1), indica que el paquete debe ser prioridad principal y será tomado primero antes que todos los paquetes en cola.

Campo ACK :: ack_flag

Bandera que activa la opción de acuse de recibo, la cual se utiliza para indicar que los datos anteriores fueron recibidos con éxito y el número de confirmación sea tomado en cuenta por la pila TCP/IP.

Campo PSH :: push_flag
Bandera que activa la opción de realizar, esta bandera se toma como una indicación para el vaciado del buffer de transmisión o recepción.

Campo RST :: rst_flag
Bandera que activa la opción de limpiar (Reset), indica que ha ocurrido algún problema con la sincronización de la conexión, por lo tanto es necesario reiniciar la conexión.

Campo SYN :: syn_flag
Bandera que activa la opción de sincronización, indica que se desea iniciar una conexión.

Campo FIN :: fin_flag
Bandera que activa la opción de finalización, indica que se desea finalizar una conexión.

Campo Ventana :: window
Este campo indica el tamaño de datos que será capaz de recibir en la próxima transmisión.

Campo Suma de Comprobación :: checksum
Es un número que se calcula mediante una operación aritmética binaria (suma en complemento a uno de 16 bits) e indica la integridad de la cabecera y sus datos.

Campo Puntero de Urgencia :: urg_pos
Incia apartir de dónde se terminan los datos urgentes en un paquete, ya que TCP permite combinar en un paquete, datos normales y datos urgentes, para eso utiliza data_offset y urg_pos, para determinar dónde están cada tipo de datos. Es importante destacar que si la bandera URG está desactivada, este campo es ignorado.

Campo Opciones :: options
Campo que indica a las conexiones nuevas al tamaño máximo de los segmentos que se podrán recibir. Es importante destacar que el campo ventana solo permite variar el tamaño de segmento dependiendo el congestionamiento existente, pero este campo indica un tamaño máximo del segmento en la sesión.

Información adicional sobre los campos y funcionamientos se puede leer el documento *RFC-793 o* bien [14]**.** Los que utilicen Sistemas Operativos Linux, pueden tomar mayor referencia o información sobre sus banderas y programación, ver */usr/include/linux/tcp.h.*

4.5 PROTOCOLO RAW

RAW, más que un protocolo con reglas y estándares, es la realización de un sockets para creación de paquetes de cualquier tipo de Protocolo desde 0, es decir, con RAW podremos crear paquetes sin contenido o bien ir anidándole contenido para después posicionarlo en la red. La programación o creación de paquetes RAW son también utilizadas para acceder a cualquier parte de un paquete recibido o formado por algún otro protocolo.

La utilización de paquetes RAW son de ayuda para implementar o crear nuevos protocolos con el fin de tener un uso o funcionamiento particular en la aceptación de paquetes, también permiten crear nuestras propias reglas o estándares para creación del mismo paquete.

Existe una gran delimitación con los paquetes RAW, ya que estos solo pueden ser formados por usuarios con ID 0, es decir, usuarios tipo Root (en sistemas operativos Linux), por lo tanto queda limitado estrictamente para usuarios del sistemas o grupo, a sí mismo, la ejecución de programas que implementan creación o modificación de paquetes queda igualmente restringido (ya que utilizan RAW).

Ya se mencionó que se obtiene una gran limitación con este tipo de aplicaciones o forma de programación que implementa paquetes o programación RAW, pero también tiene una gran ventaja a nivel administración o desarrollo de software para redes, y se basa principalmente en que, todo programador que utilice o implemente protocolo RAW, podrá tomar todos los tipos de paquetes que se reciban en el sistema, así es, no es necesario ser experto de la aplicación o bien tener derechos, RAW está más abajo que las capas OSI, es una forma de poder ver todos los paquetes y en un momento dado ocupar dicha información para funciones de monitoreo o estadística.

A continuación una breve reseña de programación para implementar sockets RAW en la Tabla 4.

Tabla 4. Modos de apertura de un socket.

SOCK_RAW	Utilícese esta opción para creación de un sockets, ejm: *socket(AF_INET, SOCK_RAW, PROTOCOLO);*
IP_HDRINCL	Permite re-construir la información de cabecera del paquete RAW, ejm: *setsockopt(socketds, SOL_IP, IP_HDRINCL, &valor, sizeof(valor));*, donde '*valor=1;*'.
IP_TTL	Permite cambiar el numero de saltos de un paquete, ejm: *setsockopt(sockfd, SOL_IP, IP_TTL, &valor, sizeof(valor);*, donde *'valor=1;'*.
TCP_MAXSEG	Permite cambiar el tamaño de los segmentos recepcionados en el flujo TCP, este valor debe ser menor a 540bytes (tamaño determinado por TCP como máximo), ejm: *setsockopt(sockfd, SOL_TCP, TCP_MAXSEG, &valor, sizeof(valor));*, donde '*valor=1*'.

RESULTADOS

5

5.1 SISTEMA PARA COMUNICACION DE REDES LAN, INALAMBRICAS Y BLUETOOTH. 112
5.2 OBJETIVOS Y REQUERIMIENTOS DEL SISTEMA. 113
5.3 COMUNICACIÓN Y LOGICA DEL SISTEMA. 114
5.4 SERVIDOR Y CLIENTE. 115
5.5 FINES COMERCIALES. 120

5. RESULTADOS.

Tras la investigación y recopilación de conocimientos en los temas plasmados en este documento que son:

- Seguridad LAN.
- Seguridad inalámbrica.
- Seguridad Bluetooth.
- Programación del socket.
- Protocolos de Comunicación.

Acompañados de los resultados obtenidos tras la practica y entendimiento de los mismos, se culmina con la codificación de una aplicación para comunicación de redes LAN, inalámbrica y Bluetooth la cual proporcione una diversidad de servicios al usuario y propietario en la utilización de esta aplicación.

El sistema para comunicación de redes LAN, inalámbrica y Bluetooth propuesto en este trabajo de investigación denominado “Aryax” (ver Figura 5.1) es una aplicación para comunicación de redes que permita la convivencia entre ambas tecnologías de comunicación ya mencionadas tiene una visión amplia no solo para el área de informática y redes, sino que contemple la posibilidad de incorporar cuestiones de publicidad, venta utilizando la red en tiempo real y transferencia de archivos entre usuario y software además de los servicios que se tendrán por omisión.

5.1 SISTEMA PARA COMUNICACION DE REDES LAN, INALAMBRICAS Y BLUETOOTH.

Aryax es aun aplicación que se compone de un servidor y un cliente, donde el servidor es parte principal para la comunicación con los clientes, generación de estadísticas mediante el tráfico capturado, auditoría de informática, delimitación de servicios para los clientes, entre otros. Este sistema trabaja bajo el sistema operativo Linux, ya que es un sistema operativo más confiable que Microsoft © Windows, posee un número muy diminutivo en cuestiones de errores del sistema y no existen amenazas de algún virus (tales como los troyanos o exploits) que expongan la información que se resguarde en el sistema. El cliente inicialmente también está programado para trabajar bajo un ambiente Linux, pero se preve la posibilidad de implementarse un cliente para usuarios del sistema operativos Microsoft © Windows. En la Figura 5.1 podemos apreciar el logotipo de la aplicación Aryax.

FIGURA 5.1: Logotipo Aryax.

5.2 OBJETIVOS Y REQUERIMIENTOS DEL SISTEMA.

Los objetivos de Aryax son los siguientes:

1. Ser la primera herramienta para transferencia de archivos Bluetooth bajo el sistema operativo Linux.
2. Ser la primera aplicación que permita la intercomunicación entre las tecnologías de comunicación LAN, inalámbrica y Bluetooth de una forma sencilla.
3. Intercambio de información entre redes de comunicación, esto puede ser: audio, imágenes, video, archivos, mensajes, chat, entre otros, en tiempo real.
4. Escaneo de dispositivos conectados a la red LAN, inalámbrica y Bluetooth.
5. Generación de registros de dispositivos de red conectados para evitar escaneo constantemente.
6. Generación de registros por cada intercambio de información realizada entre dispositivos de la red.
7. Delimitación en el intercambio de información para los clientes.
8. Establecer anuncios de publicidad entre mensajes intercambiables entre cualquier dispositivo de red hacia los dispositivos móviles.
9. Envío de publicidad personalizada a dispositivos de la red.
10. Auditoría de seguridad en la red.
11. Compartimiento de información.
12. Creación de sesiones de chat auditables y moderadas.
13. Acortar distancias en intercambio de información mediante la utilización de la red.

Los requerimientos del sistema para la instalación de Aryax son:

A. Requerimientos del software servidor:
    1. Sistema operativo Linux.
    2. 256MB de RAM para permitir el trabajo con multi-hilos.
    3. Procesador de 1.0 Ghz mínimo para permitir el trabajo con multi-hilos.
    4. Teclado, ratón, monitor.
    5. Dispositivo de hardware para comunicación a la red que se desee.

B. Requerimientos del software cliente:
    1. Sistema operativo Linux.
    2. 256MB de RAM para permitir el trabajo con multi-hilos.
    3. Procesador de 1.0 Ghz mínimo para permitir el trabajo con multi-hilos.
    4. Gestor de Bases de Datos MySQL.
    5. Dispositivo de hardware para comunicación a la red que se desee.
    6. Teclado, ratón, monitor.
    7. Pasarela (gateway) para acceso a otras redes (Bluetooth).
    8. Router para permitir acceso a dispositivos inalámbricos.
    9. Conmutador (switch) para concentrar LAN.

5.3 COMUNICACIÓN Y LOGICA DEL SISTEMA.

Para proporcionar los servicios que brinda Aryax, es necesario principalmente la utilización de los dispositivos de comunicación para red, como serian: switch, router y gateway.

El switch es un concentrador de red que permite la interconexión de equipos de computo mediante un dispositivo de hardware (NIC, Network Interface Card) y un medio físico (Cable UTP), con estos requerimientos se realiza una red LAN lista para intercambiar información.

El router tiene las mismas características que el switch, solo que éste tiene las particularidades de permitir la conexión a dispositivos inalámbricos y una entrada especial para Internet, en base a esto se puede proporcionar el servicio de Internet a los equipos conectados mediante cable y la conexión inalámbrica.

La pasarela es un dispositivo de hardware para redes, que se utiliza para conectar varios tipos de red, comúnmente se puede denominar como “dispositivo para accesar a otras redes”, el fin de este dispositivo de hardware es unir las redes LAN e inalámbrica con Bluetooth.

Teniendo interconectados los tres dispositivos ya mencionados con un servidor y la instalación de aryax como servidor, se puede iniciar la conexión y búsqueda de dispositivos para realizar cualquier comunicación o intercambio de información.

La lógica principal está en los concentradores, que son los encargados de dirigir el tráfico y redireccionar los paquetes que entran o salen, posteriormente si son paquetes entrantes hacia cualquier otro dispositivo, primeramente se debe pasar dicho paquete al sistema servidor, en donde aryax creará una reseña en su base de datos la cual servirá para realizar cuestiones estadísticas sobre aspectos de servicios, utilización, rendimiento, tráfico, saturación, etc. Una vez registrado el movimiento, se libera el paquete y se le envía a su destinatario. En la Figura 5.2 se puede visualizar una red de comunicaciones Aryax como ejemplo.

FIGURA 5.2: Red de Comunicaciones en Aryax.

5.4 SERVIDOR Y CLIENTE.

Una vez cumplidos los requerimientos mencionados en el capítulo 5.2 tanto para servidores como los clientes, se inicia el intercambio de información donde cada uno realizará las siguientes operaciones:

- El sistema servidor.
    - Redes LAN.
        1. Visualización de equipos conectados dentro de la red LAN con la posibilidad de envío de información y escaneo de nuevos equipos.
        2. Los equipos mostrados de forma opaca, son equipos que estuvieron conectados a la red y se encuentran temporalmente desconectados.
        3. El símbolo de "+" mostrado a la izquierda de los íconos que identifican a los equipos como conectados, brindan mayor información sobre el equipo seleccionado, esta información puede ser: estado, sistema operativo, versión del sistema operativo, nombre del equipo, nombre de la sesión, IP, archivos compartidos, servidor ftp, servidor http, servidor ssh.
        4. Publicidad permitida a mostrar para esta red.
    - Redes inalámbricas.
        1. Visualización de equipos conectados dentro de la red inalámbrica con la posibilidad de envío de información y escaneo de nuevos equipos.
        2. Los equipos mostrados de forma opaca, son equipos que estuvieron conectados a la red y se encuentran temporalmente desconectados.
        3. El símbolo de "+" mostrado a la izquierda de los íconos que identifican a los equipos como conectados, brindan mayor información sobre el equipo seleccionado, esta información puede ser: estado, sistema operativo, versión del sistema operativo, nombre del equipo, nombre de la sesión, IP, archivos compartidos, servidor ftp, servidor http, servidor ssh.
        4. Publicidad permitida a mostrar para esta red.
    - Redes Bluetooth.
        1. Visualización de equipos conectados dentro de la red Bluetooth con la posibilidad de envío de información y escaneo de nuevos equipos.
        2. Los equipos mostrados de forma opaca, son equipos que estuvieron conectados a la red y se encuentran temporalmente desconectados.
        3. El símbolo de "+" mostrado a la izquierda de los íconos que identifican a los equipos como conectados, brindan mayor información sobre el equipo seleccionado, esta información puede ser: estado, sistema operativo, versión del sistema operativo, nombre del equipo, nombre de la sesión, IP, archivos compartidos, servidor ftp, servidor http, servidor ssh.
        4. Publicidad permitida a mostrar para esta red.
    - Auditoría.
        1. Permite auditar a cualquiera de las redes de comunicación.
        2. Selección del tipo de auditoría para cualquiera de las redes y dependiendo el tipo de red.

3. Auditar equipos en particular dentro de una de las redes.

- Chat.
  1. Realización de canales de chat.
  2. Se establece un administrador por default (servidor aryax) y administrador original (creador del chat).
  3. Se delimita el número de usuarios por el creador del canal.
  4. Se establece el tipo de canal: privado (solo servidor aryax lo visualiza) o público (los usuarios no lo pueden visualizar, solo ser invitados).
  5. Búsqueda de canales.
  6. Inicio de sesión a algún canal o múltiples canales.
  7. Envío de información a usuarios del canal.
- Administración.
  1. Establecimiento de delimitaciones para la red LAN.
  2. Establecimiento de delimitaciones para la red inalámbrica.
  3. Establecimiento de delimitaciones para la red Bluetooth.
  4. Establecimiento de delimitaciones para el servicio de chat.
  5. Establecimiento de delimitaciones para el envío de información en alguna red o chat.
  6. Establecimiento de delimitaciones para el número de envíos o recepción por día u hora en alguna red o chat.
  7. Tipo de archivos permitidos a enviar o recibir.
  8. Medio de comunicación a establecer para la comunicación (NIC, inalámbrica, Bluetooth).
  9. Nombre del servidor Aryax, contraseña.
  10. Configuración del correo electrónico para el envío automático de anomalías detectadas.
  11. Establecimiento de ciclos de tiempo para la visualización de publicidad por los usuarios cliente.
  12. Establecer cada cuantos minutos realizar escaneos de equipos en todas las redes.
- Estadísticas,
  1. Estadísticas de tráfico en tiempo real por tipo de red.
  2. Estadísticas de tráfico en tiempo real por la red completa.
  3. Estadísticas de tráfico por equipo.
  4. Estadísticas de tráfico de red por periodos (horarios, días, meses).
  5. Sitios más visitados.
  6. Tipos de archivos más enviados.
  7. Horas críticas.
  8. Número de equipos conectados por: hora, día, meses.
  9. Número Clics realizados a la publicidad.

- El sistema cliente.
  - Redes LAN.

    1. Visualización de equipos conectados dentro de la red LAN con la posibilidad de envío de información y escaneo de nuevos equipos.

2. Los equipos mostrados de forma opaca, son equipos que estuvieron conectados a la red y se encuentran temporalmente desconectados.
3. El símbolo de "+" mostrado a la izquierda de los íconos que identifican a los equipos como conectados, brindan mayor información sobre el equipo seleccionado, esta información puede ser: estado, sistema operativo, versión del sistema operativo, nombre del equipo, nombre de la sesión, IP, archivos compartidos, servidor ftp, servidor http, servidor ssh.

- Redes inalámbricas.

1. Visualización de equipos conectados dentro de la red LAN con la posibilidad de envío de información y escaneo de nuevos equipos
2. Los equipos mostrados de forma opaca, son equipos que estuvieron conectados a la red y se encuentran temporalmente desconectados.
3. El símbolo de "+" mostrado a la izquierda de los íconos que identifican a los equipos como conectados, brindan mayor información sobre el equipo seleccionado, esta información puede ser: estado, sistema operativo, versión del sistema operativo, nombre del equipo, nombre de la sesión, IP, archivos compartidos, servidor ftp, servidor http, servdor ssh.

- Redes Bluetooth.

1. Visualización de equipos conectados dentro de la red LAN con la posibilidad de envío de información y escaneo de nuevos equipos
2. Los equipos mostrados de forma opaca, son equipos que estuvieron conectados a la red y se encuentran temporalmente desconectados.
3. El símbolo de "+" mostrado a la izquierda de los íconos que identifican a los equipos como conectados, brindan mayor información sobre el equipo seleccionado, esta información puede ser: estado, sistema operativo, versión del sistema operativo, nombre del equipo, nombre de la sesión, IP, archivos compartidos, servidor ftp, servidor http, servidor ssh.

- Chat.

1. Realización de canales de chat.
2. Se establece un administrador por default (servidor aryax) el cual moderará los canales creados por los demás usuarios dentro de alguna de las redes.
3. Se delimitará el número de usuarios por el creador del canal y se puede establecer un número máximo de usuarios para todos los que deseen crear canales.
4. Se delimitará el número de canales permitidos a crear.
5. Se establece el tipo de canal: privado o público, donde el servidor aryax podra visualizar cualquier tipo de canal.
6. Delimitados al número de tipos de canales a crear.
7. Búsqueda de canales.
8. Inicio de sesión a algún canal o múltiples canales.

9. Envío de información a usuarios del canal.

- Configuración.
  1. Medio de comunicación a establecer para la comunicación (NIC, inalámbrica, Bluetooth).
  2. Nombre del equipo dentro de la red aryax.
  3. Usuario y contraseña para inicio de sesión a la red aryax.
  4. Establecer archivos a compartir.
  5. Archivos que serán visibles en determinadas redes.
  6. Servicios que tendrán disponibles (http, ssh, ftp).

Cada vez que los clientes realicen una búsqueda de equipos dentro de la red LAN, inalámbrica o Bluetooth, solamente se realizará una consulta al servidor el cual contestará con los equipos que se encuentren en esa red a buscar. Para evitar la saturación y pérdida de tiempo en escaneo de equipos por parte del servidor al momento de ser consultado, el servidor realizará escaneos automáticos en cierto tiempo, esto podría ser cada cierto número de minutos, de esta forma se evita estar escaneando equipo cada instante. Todos los resultados obtenidos por el servidor en cuestiones de: escaneo, transferencias de información, configuraciones de los usuarios en sus clientes aryax, etc. Serán guardados en la base de datos del servidor aryax para evitar la utilización de bases de datos en los equipos cliente.

En las Figuras 5.3, 5.4, 5.5 y 5.6 se muestra: entrada al sistema aryax, panel LAN, panel Inalámbrica y panel Blueooth.

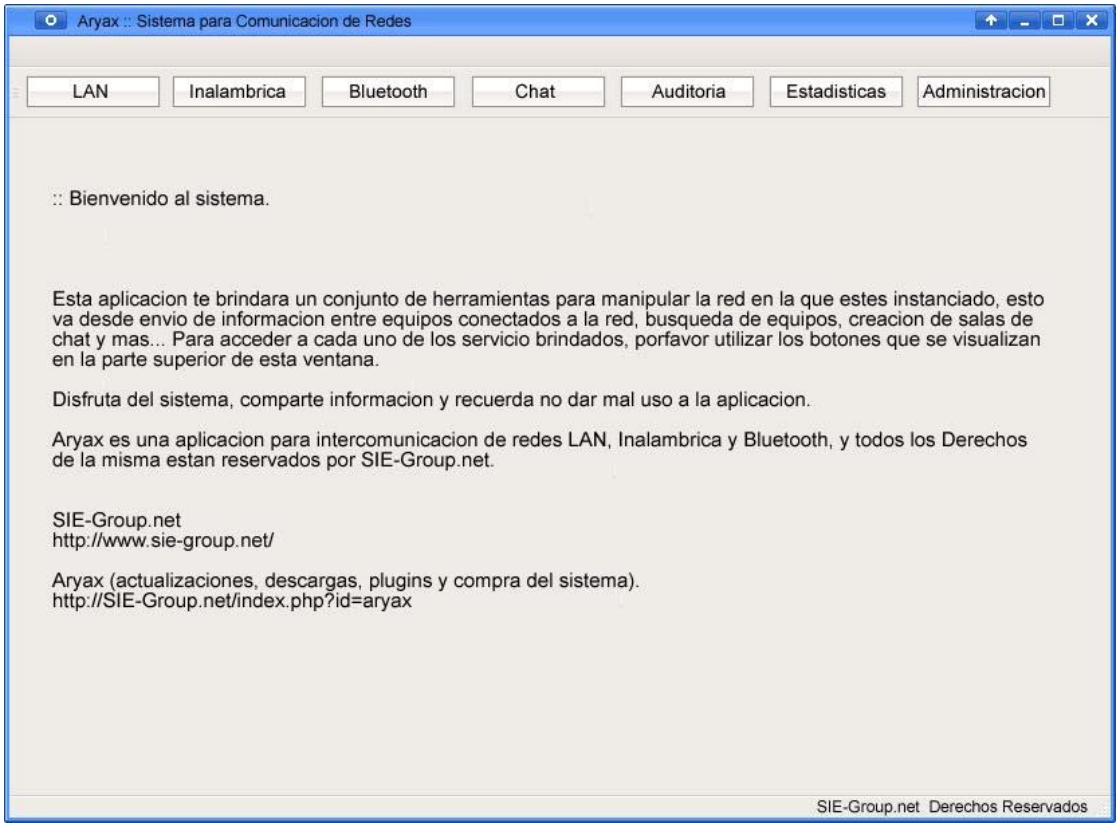

FIGURA 5.3: Presentacion Aryax.

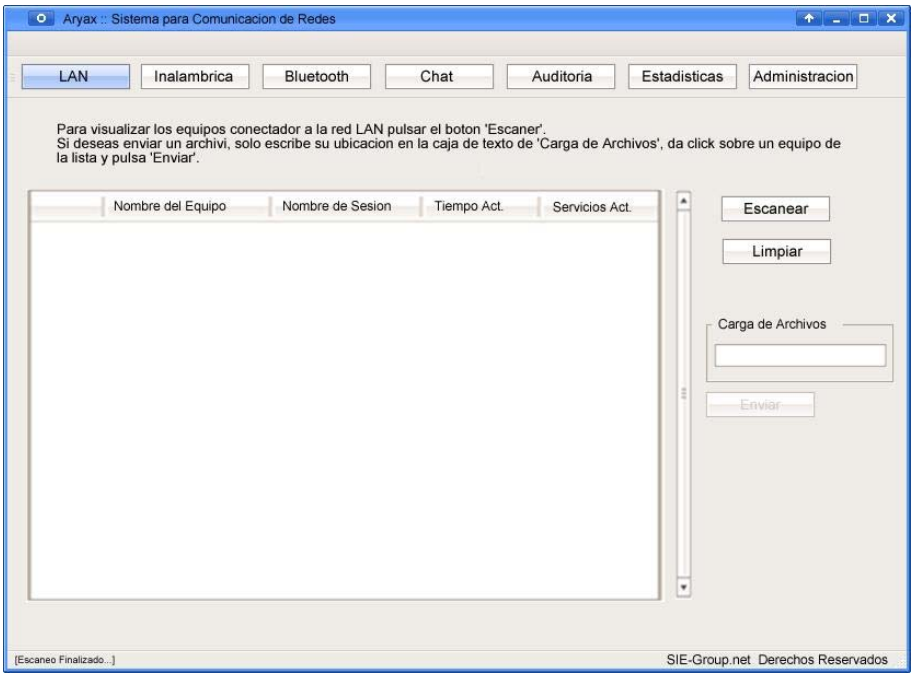

FIGURA 5.4: Panel LAN.

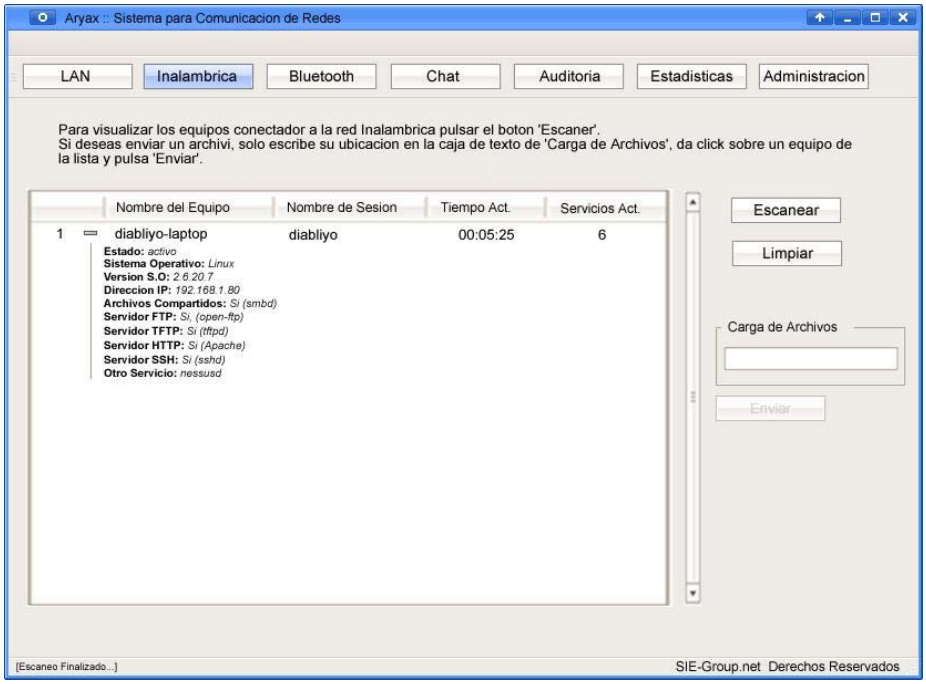

FIGURA 5.5: Panel Inalámbrico.

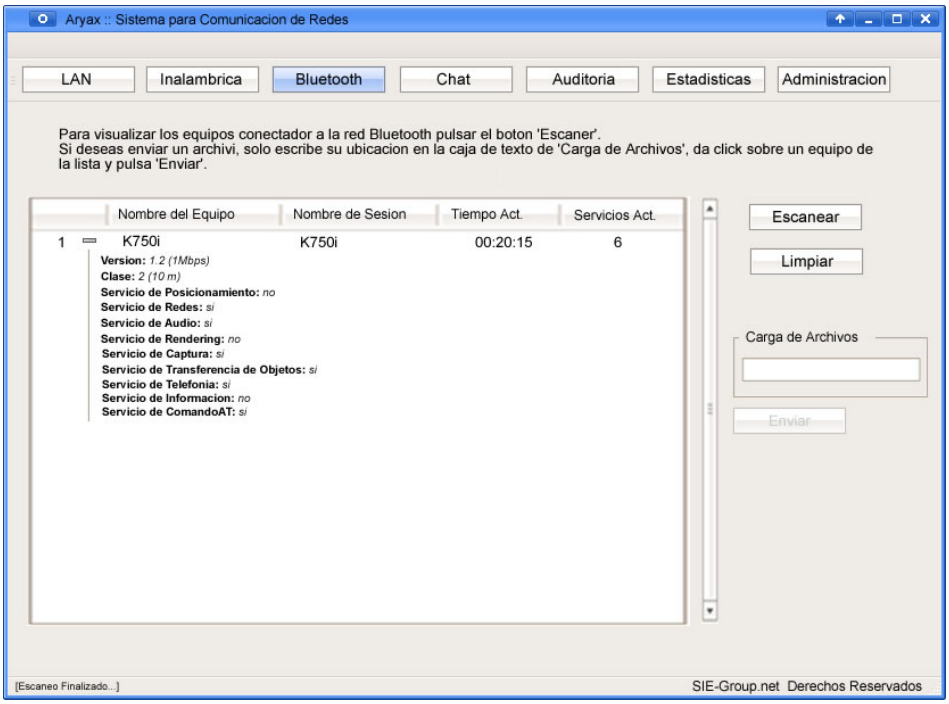

FIGURA 5.6: Panel Bluetooth.

5.5 FINES COMERCIALES.

La licencia del software aryax aún es un tema abierto debido a los servicios que proporciona y la facilidad de obtener ingresos mediante la publicidad que se puede incorporar a las formas de comunicación que utilizan los usuarios.

Aryax es una aplicación de software creada para trabajar bajo ambiente Linux con fines de comunicación y comercio electrónico, enfocado en la publicidad, anuncios, ventas de articulos, etc. Cualquiera materia que pueda ser vendida a una persona, esto mediante los anuncios configurables por el administrador, así mismo este servicio de publicidad no tiene costo alguno para el que adquiere esta aplicación, queda a decisión propia el número de anuncios publicitarios y cuestiones de pago por anuncios mostrados en alguna forma de comunicación.

Sabiendo que se puede obtener un beneficio al servicio que proporciona el sistema, esto evita dudas al momento de comprar la aplicación Aryax, debido a que la misma eficiencia del sistema, lógica y formas comerciales a manipular, crean más que una herramienta de trabajo y lo posicionan como una forma de mercado electrónico el cual hace que la compra del software sea saldada por el trabajo del mismo.

CONCLUSIONES

Los estudios realizados en este documento de investigación permiten un mejor conocimiento para reconocer los modos de comunicación y modelos utilizados en el viaje de la información de una red a otra, siendo que se conocen las ventajas y desventajas existentes en las redes de comunicación y poder realizar un modelo de protección para cualquier red de comunicación estudiada en este documento. De igual manera conjugado con conocimientos avanzados en programación se realizo la codificación de una aplicación que mezcla todos los conocimientos descritos en este documento.

Conforme a los resultados arrojados en el estudio realizado por la aplicación y funcionamiento del servicio que proporciona el mismo programa, se determinó que la aplicación “Aryax” podrá ser un buen modelo de comunicación para redes LAN, inalámbrica y Bluetooth, ya que incorpora una variedad de paneles de configuración los cuales pueden ser ajustables a las limitaciones que se requieran en la red entera para un mejor uso de la misma (según se desee). Contando con elementos publicitarios adheridos al gusto por los manejadores del software, con fines de obtener un ingreso para solventar el servicio que proporciona dicho software.

Los modos de comunicación brindados para equipos móviles que utilicen el software cliente “Aryax” podrán experimentar maneras rápidas y sencillas de intercambio de información entre redes Inalámbricas, LAN y Bluetooth, sin costo alguno en la utilización del servicio de forma inicial, estos parámetros pueden ser moderados por el administrador del software.

Todos los equipos que se encuentran dentro de la red LAN e inalámbrica, que utilicen el software cliente “Aryax” podrán experimentar formas rápidas y sencillas para intercambiar información entre redes LAN, inalámbricas y Bluetooth, sin costo alguno en la utilización del servicio de forma inicial, estos parámetros pueden ser moderados por el administrador del software.

Como visión futura, el sistema para comunicación de redes LAN, Inalámbricas y Bluetooth “Aryax”, deja camino abierto para ser una herramienta con un crecimiento constante, ya que como se había mencionado con anterioridad el sistema Aryax solo esta disponible para trabajar bajo un ambiente Linux y los servicios que proporciona son propuestas iniciales para tener un mejor impacto. De modo que Aryax aun es un sistema que necesita implementarse en sistemas operativos Windows e ínsita a los desarrolladores de software y principiantes a seguir con proyectos cada vez más innovadores y con mejores servicios accesible para todas las plataformas.

REFERENCIAS BIBLIOGRAFICAS

1 Cisco Systems. (ano, mes dia). *Guía de Segundo Ano CCNA 3 y 4* (Tercera Edición). [Libro]. *Volumen 3 y 4 en una sola edicion*. ISBN 8420540803, Inc. ciscopress.com.

2 Wikipedia.org. (2006, abril). *ICMP Protocolo de Mensajes de Control de Internet* (Edición Única). [Online]. *Volumen Único*. Disponibilidad: http://es.wikipedia.org/wiki/Internet_Control_Message_Protocol

3 Wikipedia.org. (2006, abril). *ARP Protocolo de Resolución de Dirección*. (Edición Única). [Online]. *Volumen Único.* Disponibilidad: http://es.wikipedia.org/wiki/Address_Resolution_Protocol

4 Wikipedia.org. (2006, abril). *RARP Protocolo de Resolución de Direcciones Inversa*. (Edición Única). [Online]. *Volumen Único.* Disponibilidad: http://es.wikipedia.org/wiki/Reverse_Address_Resolution_Protocol

5 Wikipedia.org. (2006, abril). *WAP Protocolo de Aplicaciones Inalambricas*. (Edición Única). [Online]. *Volumen Único.* Disponibilidad: http://es.wikipedia.org/wiki/Wireless_Application_Protocol

6 Wikipedia.org. (2006, abril). *WEP Wired Equivalent Privacy*. (Edición Única). [Online]. *Volumen Único.* Disponibilidad: http://es.wikipedia.org/wiki/Wired_Equivalent_Privacy

7 Wikipedia.org. (2006, abril). *WPA Acceso Protegido a Wi-Fi*. (Edición Única). [Online]. V*olumen Único.* Disponibilidad: http://es.wikipedia.org/wiki/WPA

8 RFC-Editor. *RFC-903 RARP*. (Edición RFC). [Online]. *Volumen 903*. Disponibilidad: http://tools.ietf.org/html/rfc903

9 Wikipedia.org. (2006, abril). *Tipos de Sniffers*. (Edición Única). [Online]. *Volumen Único.* Disponibilidad: http://es.wikipedia.org/wiki/Tipos_de_Sniffer

10 Wikipedia.org. (2006, abril). *DoS Denegación de Servicios*. (Edición Única). [Online]. *Volumen Único.* Disponibilidad: http://es.wikipedia.org/wiki/Ataque_de_denegaci%C3%B3n_de_servicio

11 RFC-Editor. RFC-0791 *Internet Protocol IP*. (Edición RFC). [Online]. *Volumen 0791*. Disponibilidad: http://www.rfc-es.org/rfc/rfc0791-es.txt

12 RFC-Editor. *RFC-0792 Protocolo de Mensajes de Control de Internet ICMP*. (Edición RFC). [Online]. *Volumen 0792*. Disponibilidad: http://www.rfc-es.org/rfc/rfc0792-es.txt

13 RFC-Editor. *RFC-0768 Protocolo de Datagramas de Usuario UDP*. (Edición RFC). [Online]. *Volumen 0768*. Disponibilidad: http://www.rfc-es.org/rfc/rfc0768-es.txt

14 RFC-Editor. *RFC-0793 Protocolo de Control de Transmicion TCP*. (Edición RFC). [Online]. *Volumen 0793*. Disponibilidad: http://www.rfc-es.org/rfc/rfc0793-es.txt

15 Wikipedia.org. (2006, abril). *Protocolo Dominante Temporal de la Integridad (TKIP-Temporal Key Integrity Protocol).* (Edición Única). [Online]. *Volumen Único.* Disponibilidad: http://es.wikipedia.org/wiki/TKIP

16 Codigo de Integridad del Mensaje (MIC-Message Integrity Code) http://www.tech-faq.com/lang/es/mic-message-integrity-check.shtml

17 Wikipedia.org. (2006, abril). *Puntos de Acceso (AP)*. (Edición Única). [Online]. *Volumen Único*. Disponibilidad: http://es.wikipedia.org/wiki/Punto_de_acceso

18 Wikipedia.org. (2006, abril). *Cifrado RC4*. (Edición Única). [Online]. *Volumen Único*. Disponibilidad: http://es.wikipedia.org/wiki/RC4

19 AirCrack-Ng.org. *Manual de AirCrack-Ng*. (Edición Única). [Online]. *Volumen Único*. Disponibilidad: http://aircrack-ng.org/doku.php

20 Wikipedia.org. (2006, abril). *Estandar IEEE 802.11*. (Edición Única). [Online]. *Volumen Único*. Disponibilidad: http://es.wikipedia.org/wiki/IEEE_802.11

21 Wikipedia.org. (2006, abril). *Picoten*. (Edición Única). [Online]. *Volumen Único*. Disponibilidad: http://es.wikipedia.org/wiki/Piconet

22 Zona Bluetooth. *Que es Bluetooth*. (Edición Única). [Online]. *Volumen Único*. Disponibilidad: http://www.zonabluetooth.com/que_es_bluetooth2.htm

23 Wikipedia.org. (2006, abril). *Multiplexación por División de Tiempo*. (Edición Única). [Online]. *Volumen Único*. Disponibilidad: http://es.wikipedia.org/wiki/Multiplexaci%C3%B3n_por_divisi%C3%B3n_de_tiempo

24 Wikipedia.org. (2006, abril). *Radiofrecuencia*. (Edición Única). [Online]. *Volumen Único*. Disponibilidad: http://es.wikipedia.org/wiki/Radiofrecuencia

25 Wikipedia.org. (2006, abril). *Unidades de Datos de Protocolo (PDU)*. (Edición Única). [Online]. *Volumen Único*. Disponibilidad: http://es.wikipedia.org/wiki/Protocol_Data_Unit

26 OpenObex. *User Guide ObexFTP*. (Edición Única). [Online]. *Volumen Único*. Disponibilidad: http://sourceforge.net/projects/openobex/

27 Airs.com. *User Guide Taylor UUCP*. (Edición Única). [Online]. *Volumen Único*. Disponibilidad: http://www.airs.com/ian/uucp.html

28 Trifinite.org. *Helomoto*. (Software). [Online]. Aplicación *Descargable*. Disponibilidad: http://trifinite.org/Downloads/helomoto.tgz

29 BlueZ.org. *User Guide BlueZ Programming*. (Edición Única). [Online]. *Volumen Único*. Disponibilidad: http://www.bluez.org/

30 RFC-es.org. Documentación RFC. (Edición Única). [Online]. *Volumen Único*. Disponibilidad: http://www.rfc-es.org/