Universidad Tecnológica de la Mixteca

Tema de Tesis:
Emulador de Ruteador IP

Que para obtener el título de
Ingeniero en Electrónica
presenta:

Víctor Francisco Martínez Silva.

Acatlima, Huajuapam de León, Oaxaca.

A:

Víctor Hugo Ferra Medina ,
Mis padres, hermanas y amigos.

Agradezco a:

Mis padres, mis hermanas, mis amigos
M. C. José Antonio Moreno
Mis profesores

por su apoyo, amor, cariño
y el haberme permitido
compartirles mi vida

Contenido

Introducción

En los últimos años la Internet ha tenido un gran desarrollo, inesperado para sus mismo creadores. Sin embargo, esta tecnología ha logrado ofrecer un medio para que miles de usuarios de distintas entidades conectadas compartan información entre todos ellos de una forma fácil y segura; incorporando una amplia variedad de tecnologías subyacentes de red.

Su estudio a profundidad es pertinente. Es por ello que con este trabajo de investigación se ha desarrollado una herramienta para que cualquier estudiante de redes pueda tener una mejor perspectiva de la arquitectura, desempeño y funcionalidad de una red de redes virtual.

La emulación de un ruteador IP a través de un programa, que pueda ejecutarse en una computadora personal (sistema operativo Windows 95 o mayor) conectada a una red local (Ethernet), mostrará algunas de las características del Protocolo Internet (IP), Protocolo de Mensajes de Control Internet (ICMP), Protocolo de Asociación de Direcciones (ARP) y el Protocolo de Información de Ruteo (RIP) pertenecientes a la Suite de Protocolos TCP/IP.

En los dos primeros capítulos se muestran algunas características de la Suite de Protocolos TCP/IP y el algoritmo de ruteo. El tercer capítulo trata acerca del paradigma cliente-servidor y la interface de programación sobre redes para Microsoft Windows, comúnmente llamada Winsock, la cual está basada en la paradigma "socket" de Berkeley Software Distribution (BSD) de la Universidad de California en Berkeley. El cuarto capítulo presenta cómo se puede realizar la emulación de dichos protocolos y por lo tanto una red de redes virtual con el programa Emulador Server, los requisitos para ello y algunos ejemplos.

La herramienta fue diseñada de manera que al ejecutarse no modifique la configuración de la computadora; es decir no se altere la dirección IP ni ningún otro dato correspondiente; ya que las direcciones de las redes a emular el programa las considera como virtuales, por la implementación del protocolo IP en el nivel de aplicación, con ayuda del protocolo UDP de la capa de transporte. En otras palabras, se creó un pseudo IP que permite crear una red de redes virtual en una pequeña red local para poder observar de una manera real los procesos de ruteo y tareas de los demás protocolos relacionadas con éste.

Capítulo 1
La Suite de Protocolos TCP/IP

1.1 Protocolos Internet TCP/IP

La tecnología de la Agencia de Proyectos de Investigación Avanzada (ARPA) [1] engloba un grupo de estándares de red que especifican cómo se comunican las computadoras entre sí, así como los requisitos para interconectar redes y para rutear los paquetes de información. Conocido como el grupo de protocolos Internet TCP/IP, es utilizado para comunicar cualquier grupo de redes interconectadas.

La Tecnología TCP/IP es la base para una red de redes global que conecta hogares, universidades, corporaciones, laboratorios, etc. en todo el mundo. La red de redes resultante permite que todos los usuarios de las diferentes entidades conectadas compartan información entre todos ellos tan fácilmente como si estuvieran en un cuarto contiguo. La tecnología TCP/IP en la Internet muestra cómo puede incorporar una amplia variedad de tecnologías subyacentes de red. Además, proporciona reglas para la comunicación, así como los detalles referentes a los formatos de los mensajes; también describen cómo responde una computadora cuando llega un mensaje y especifican de qué manera una computadora maneja un error u otras condiciones anormales. Además de permitir la comunicación por computadora de forma independiente de cualquier hardware de red de cualquier marca.

El hacer a un lado los detalles de bajo nivel de la comunicación nos ayuda a mejorar la productividad de muchas maneras. Primeramente nos permite manejar solo abstracciones de protocolos de un nivel más elevado, no necesitando aprender o recordar todos los detalles sobre la configuración del hardware en particular. Por lo tanto como segunda ventaja, los programas realizados pueden ejecutarse en cualquier máquina y arquitectura, sin cambiar o ser reconfigurados. Por último ya que los programas son independientes del hardware subyacente, pueden proporcionar comunicación directa entre un par arbitrario de máquinas. Los programadores no necesitan hacer versiones especiales de software de aplicación para mover y traducir datos entre cada par de máquinas posibles.

Desde el punto de vista del usuario, una red de redes TCP/IP aparece como un grupo de programas de aplicación que utilizan la red para llevar a cabo tareas útiles de comunicación. La mayoría de los usuarios que accesan a la Internet lo hacen al correr programas de aplicación sin entender la tecnología TCP/IP, la estructura de la red de redes subyacente o incluso sin entender el camino que siguen los datos hacia su destino. Sólo los programadores que crean los programas de aplicación de red necesitan ver a la red de redes como una red, así como entender parte de la tecnología.

Los servicios de aplicación de Internet [2] más populares y difundidos son:

- Correo electrónico.- El correo electrónico permite que un usuario envíe o reciba mensajes cortos de texto (memorandos, cartas, etc.) con otros usuarios.

- Transferencia de archivos.- Esta aplicación permite que los usuarios envíen o reciban archivos arbitrariamente grandes de programas o de datos.

- Acceso remoto.- El acceso remoto permite que un usuario se conecte a una máquina remota, desde su computadora, y establezca una sesión interactiva.

- Web.- World Wide Web es un sistema de hipermedios que permite al usuario acceder a información por medio de Browsers a algún webserver a través del protocolo HTTP para poder interactuar con imágenes, sonido, vídeo y otros sitios o webservers.

Por otra parte, un programador que crea programas de aplicación que utilizan protocolos TCP/IP tiene una visión totalmente diferente de una red de redes. En el nivel de red, una red de redes proporciona dos grandes tipos de servicios [2] que todos los programas de aplicación utilizan:

- Servicio sin conexión de entrega de paquetes.- La entrega sin conexión es una abstracción del servicio que la mayoría de las redes de conmutación de paquetes ofrece. Simplemente significa que una red de redes TCP/IP rutea mensajes pequeños de una máquina a otra, basándose en la información de dirección que contiene cada mensaje. Debido a que el servicio sin conexión rutea cada paquete por separado, no garantiza una entrega confiable y en orden. Como por lo general se introduce directamente en el hardware subyacente, el servicio sin conexión es muy eficiente en cuanto a tiempo y a sobreflujo. Algo muy importante es que tener una entrega de paquetes sin conexión como la base de todos los servicios de red de redes, hace que los protocolos TCP/IP sean adaptables a un amplio rango de hardware de red.

- Servicio de transporte de flujo confiable.- Debido a que la mayor parte de las aplicaciones necesitan mucho más que sólo la entrega de paquetes; ya que necesitan recuperarse de manera automática de los errores de transmisión, paquetes perdidos o fallas de conmutadores intermedios a lo largo del camino entre el transmisor y el receptor; el servicio de transporte confiable resuelve dichos problemas. Permite que una aplicación en una computadora establezca una "conexión" con una aplicación en otra computadora, para después enviar un gran volumen de datos a través de la conexión como si ésta fuera permanente y directa del hardware. Se establece entre las computadoras un circuito virtual de comunicación. Debajo de todo esto, por supuesto, los protocolos de comunicación dividen el flujo de datos en pequeños mensajes y los envían, uno tras otro, esperando que el receptor proporcione un acuse de recibo de la recepción.

Con todo ello; nos podemos dar cuenta que la tecnología TCP/IP tiene como objetivo primordial: esconder los detalles del hardware subyacente de red a la vez que proporciona servicios universales de comunicación, dando como resultado una abstracción de alto nivel que proporciona la estructura para todas las decisiones en cuanto a diseño.

1.2 Arquitectura de una red de redes TCP/IP

El concepto de una red de redes o internet es muy poderoso. Elimina la noción sobre comunicaciones de los detalles de las tecnologías de red y oculta los rasgos de bajo nivel al usuario. De manera más importante, controla todas las decisiones sobre diseño de software y explica cómo manejar las direcciones físicas y las rutas.

Todo ello se basa en una interconexión en el nivel de red. La cual proporciona un mecanismo que entrega en tiempo real paquetes de datos, desde su fuente original hasta su destino final. El manejo de pequeñas cantidades de datos en vez de archivos o grandes mensajes tiene muchas ventajas:

- El esquema se proyecta directamente hacia el hardware subyacente de red, haciéndolo extremadamente eficiente.
- La interconexión de nivel de red separa de los programas de aplicación, las actividades de comunicación de datos, permitiendo que computadoras intermedias manejen el tráfico de red sin "entender" las aplicaciones que lo utilizan.
- Utilizar conexiones de red mantiene flexible a todo el sistema, haciendo posible la construcción de instalaciones de comunicación con propósitos generales.
- El esquema permite que los administradores de red agreguen nuevas tecnologías de red al modificar o agregar una pieza sencilla de software nuevo de nivel de red, mientras los programas de aplicación permanecen sin cambios.

La interconexión y la transmisión de datos en una red de redes se establecen físicamente mediante los ruteadores. Los ruteadores son máquinas que se encargan de unir una red con otra y de transferir paquetes de datos entre ellas. Es por ello que un ruteador necesita conocer la topología de la red de redes más allá de las redes que interconectan. Se puede pensar entonces que los ruteadores, que deben saber cómo rutear paquetes hacia su destino, son grandes máquinas con suficiente memoria para guardar información sobre cada máquina dentro de la red de redes a la que se conectan. Sin embargo hemos de ver que en las redes de redes TCP/IP, los ruteadores son máquinas pequeñas y con poca memoria, ya que los ruteadores solo utilizan la red de destino y no el host o computadora destino cuando rutean un paquete. Por lo tanto la cantidad de información que necesita guardar un ruteador es proporcional al número de redes dentro de otra red, no al número de computadoras.

Además de los ruteadores que interconectan redes físicas, se necesita software en cada computadora o host para permitir que los programas de aplicación utilicen la red de redes como si ésta fuera una sola red física real. En la figura 1.1 podemos ver el papel de los ruteadores en una red de redes.

Este concepto abstracto sobre sistemas de comunicación para una interconexión universal de nivel de red se conoce como enlace de redes (internetworking) [2]. En la figura 1.2 podemos observar el punto de vista del usuario sobre la Internet y su estructura de interconexión como una red de redes.

Podemos concluir entonces que los protocolos TCP/IP para redes de redes definen una abstracción de "red" que trata de manera igual a todas las redes, ya sea una red de área local (LAN) Ethernet, una red de área amplia (WAN) como la columna vertebral ANSNET o un enlace punto a punto entre dos máquinas. Obteniendo así una sola red

virtual extensa que abarca a todas las redes interconectadas por ruteadores, dejando a un lado el hardware subyacente de las redes.

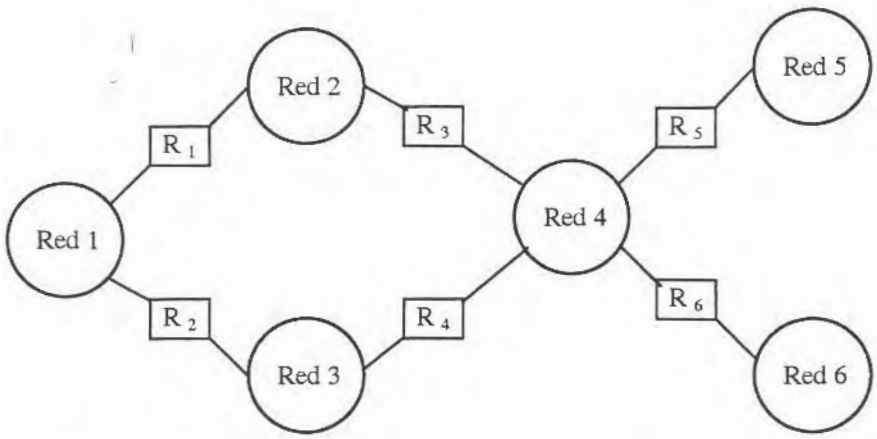


Figura 1.1. Red de redes, interconectadas por ruteadores.

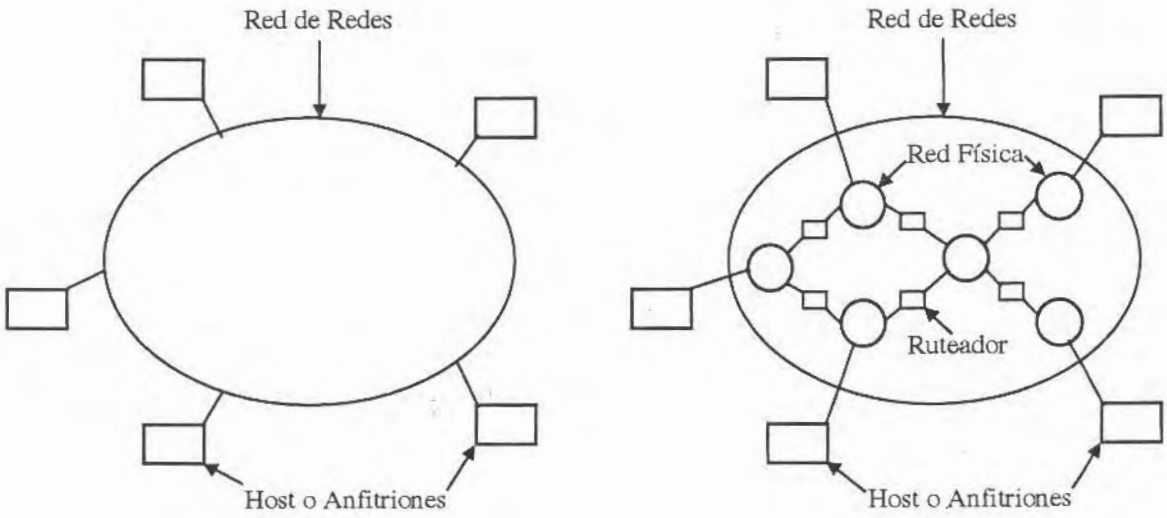


Figura 1.2 (a) Una sola red virtual, punto de vista del usuario, y (b) estructura física de la red de redes

1.3 Direcciones Internet

Cada tecnología de hardware de red define un mecanismo de direccionamiento que las computadoras utilizan para especificar el destino de cada paquete. A cada computadora conectada a una red se le asigna una dirección única, conocida como dirección física de red.

Siguiendo nuestra filosofía de comunicaciones como un servicio universal, necesitamos un método aceptado de manera global para identificar cada computadora conectada a la red de redes; no importándonos los detalles de cualquier tecnología de hardware. Los diseñadores del TCP/IP eligieron un esquema análogo al direccionamiento en las redes físicas, en el que cada computadora o anfitrión en la red de redes tiene asignada una dirección de número entero de 32 bits, llamada su dirección de red de redes o dirección IP [3]. La parte inteligente del direccionamiento en una red de redes es que los números enteros son seleccionados con cuidado para hacer eficiente el ruteo. De manera específica, una dirección IP codifica la identificación de la red a la que se conecta el anfitrión, así como la identificación de un anfitrión único en esa red.

Cada dirección es un par (netid, hostid), en donde netid identifica una red y hostid una máquina dentro de la red. Con base a esta definición se puede determinar el tipo de dirección IP según los tres bits de orden, de los que son necesarios sólo dos bits para distinguir entre los tres tipos primarios que se definen (ver figura 1.3) [4]. Las direcciones tipo A, que se utilizan para las pocas redes que tienen más de 2^{16} hosts, asignan 7 bits al campo netid y 24 bits al campo hostid. Las direcciones tipo B, que se utilizan para redes de tamaño mediano que tienen entre 2^8 y 2^{16} anfitriones, asignan 14 bits al campo netid y 16 bits al hostid. Por último, las direcciones tipo C, que tienen menos de 2^8 hosts, asignan 21 bits al campo netid y sólo 8 bits al hostid. Obsérvese que las direcciones IP se han definido de tal forma que es posible extraer rápidamente los campos hostid o netid. Los ruteadores, que utilizan el campo netid de una dirección para decidir a dónde enviar un paquete, dependen de una extracción eficiente para lograr una velocidad alta. Las redes tipo D son direcciones de multidifusión y las redes tipo E están reservadas para uso posterior.

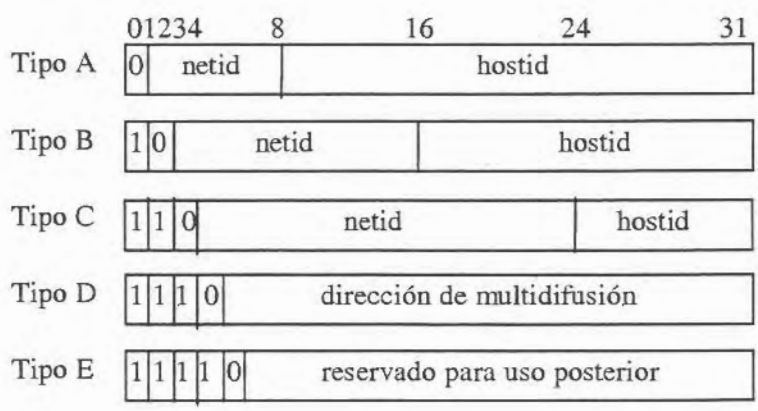


Figura 1.3 Los cinco tipos de direcciones de Internet (IP)

Las direcciones IP se escriben como cuatro enteros decimales separados por puntos, en donde cada entero proporciona el valor de un octeto de la dirección IP, por ejemplo:

10100010 00100100 10111100 01111111

se escribe

162.36.188.127

Por lo tanto, la relación entre los tipos de direcciones IP y los números decimales es la siguiente:

Tipo	Dirección más baja	Dirección más alta
A	0.1.0.0	126.0.0.0
B	128.0.0.0	191.255.0.0
C	192.0.1.0	223.255.255.0
D	224.0.0.0	239.255.255.255
E	240.0.0.0	247.255.255.255

Figura 1.4 Tipos de redes con sus respectivos rangos de direcciones IP en decimales con punto

Para permitir el intercambio de datos binarios entre máquinas, los protocolos TCP/IP requieren del ordenamiento estándar de octetos para los enteros dentro de los campos del protocolo. Un host debe convertir todos los datos binarios de su forma interna a un orden estándar de octetos de red antes de enviar un paquete y debe hacer la conversión de orden de octeto de red al orden interno cuando reciba paquetes.

Cuando dos máquinas necesitan comunicarse entre sí sus direcciones IP deben ser transformadas en sus direcciones físicas correspondiente al hardware subyacente de comunicación que están utilizando [6]. Este es un problema conocido como problema de asociación de direcciones y ha sido resuelto de muchas maneras; una de ellas es mediante el Protocolo de Asociación de Direcciones (ARP) [5], basado en una asociación dinámica muy sencilla y fácil de mantener. Cuando un host quiere mandar un paquete de datos a cualquier máquina o anfitrión con una dirección IP ya definida, primeramente manda un mensaje a toda la red preguntando cuál es la máquina con esa dirección IP, para que ella le responda con su dirección física y así poderle mandar el mensaje. Se entiende que todas las máquinas conectadas a la red van a recibir el mensaje de solicitud, pero sólo la máquina que reconozca su dirección con la de búsqueda va a responder la solicitud. Podemos resumir que el protocolo ARP permite que un host encuentre la dirección física de cualquier máquina conectada a la misma red física con sólo tener la dirección IP de su objetivo.

El ARP tiene además una memoria intermedia donde almacena las direcciones IP y físicas de las máquinas que va consultando el anfitrión, esto para usos posteriores. Antes de enviar una solicitud ARP, el anfitrión busca primero la dirección IP objetivo en su memoria intermedia.

Cada vez que un host trasmite una solicitud ARP, manda su dirección IP y física para que los receptores actualicen su información en su memoria intermedia.

El ARP oculta el direccionamiento físico subyacente de red, al permitir que se asigne una dirección IP arbitraria a cada máquina.

De esta manera podemos pensar en ARP como parte del sistema físico de red y no como parte de los protocolos de red de redes.

De forma inversa existe un protocolo llamado RARP (Protocolo Inverso de Asociación de Direcciones) [7], que nos sirve cuando queremos encontrar la dirección IP de una máquina y solo tenemos la dirección física. El protocolo RARP utiliza el direccionamiento físico de la red para obtener la dirección de red de redes de la máquina. El mecanismo RARP proporciona la dirección de hardware físico de la máquina de destino para identificar de manera única el procesador y trasmite por difusión la solicitud RARP. Los servidores en la red (en muchas ocasiones dedicados; es decir, servidores RARP) reciben el mensaje, buscan la dirección en sus tablas o la transforman y responder al transmisor. Dado esto la máquina solicitante obtiene su dirección IP, la guarda en memoria y no vuelve a utilizar RARP hasta que inicia de nuevo.

Como alternativa a RARP existe el protocolo de arranque BOOTP [8] que utiliza el UDP, lo que hace posible extender el proceso de arranque a través de un ruteador; además de permitir a una máquina determinar una dirección de ruteador, una dirección (archivo) de servidor y el nombre de un programa que la computadora deberá correr. Todo ello permite a los administradores establecer la configuración de una base de datos que transforma un nombre genérico como "Unix" en un nombre de archivo completamente caracterizado que contiene la imagen de memoria apropiada para el hardware del cliente.

Al BOOTP le sucede el Dynamic Host Configuration Protocol (DHCP) [9] que permite que un servidor localice direcciones IP automáticamente o dinámicamente. Lo cual es muy útil y necesario en ambientes con redes inalámbricas, cuyas computadoras pueden conectarse y desconectarse rápidamente.

1.4 Protocolo Internet (IP)

Conceptualmente, una red de redes TCP/IP proporciona tres conjuntos de servicios arreglados jerárquicamente [2]. Primeramente tenemos un servicio de entrega sin conexión que nos brinda el fundamento sobre el cual se apoya el servicio de transporte confiable, como segundo servicio, y establece una plataforma de alto nivel de la cual dependen las aplicaciones (última capa). En la figura 1.5 podemos ver estas tres capas conceptuales de los servicios de la Internet.

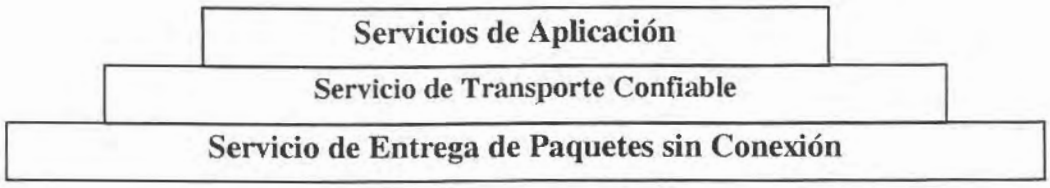


Figura 1.5 Capas conceptuales de los servicios de la Internet

El servicio de entrega de paquetes sin conexión está definido por el Protocolo Internet [10], conocido comúnmente por sus iniciales IP. Técnicamente, el servicio se define como un sistema de entrega de paquetes sin conexión y con el mejor esfuerzo, análogo al servicio proporcionado por el hardware de red Ethernet que opera con un paradigma de entrega con el mejor esfuerzo. El servicio se conoce como no confiable porque la entrega no está garantizada. Los paquetes se pueden perder, duplicar, retrasar o entregar sin orden, pero el servicio no detectará estas condiciones ni informará al emisor o al receptor. El servicio es llamado sin conexión dado que cada paquete es tratado de manera independiente de todos los demás. Una secuencia de paquetes que se envían de una computadora a otra puede viajar por diferentes rutas, algunos de ellos pueden perderse mientras otros se entregan. Por último, se dice que el servicio trabaja con base en una entrega con el mejor esfuerzo porque el software de red de redes hace un serio intento por entregar los paquetes.

El protocolo IP proporciona tres definiciones importantes. Primero, define la unidad básica para la transferencia de datos utilizada a través de una red de redes TCP/IP. Es decir, especifica el formato exacto de todos los datos que pasarán a través de una red de redes TCP/IP. Segundo, el software IP realiza la función de ruteo, seleccionando la ruta por la que los datos serán enviados. Tercero, el IP incluye un conjunto de reglas que le dan forma a la idea de entrega de paquetes no confiable. Las reglas caracterizan la forma en que los host y ruteadores deben procesar los paquetes, cómo y cuándo se deben generar los mensajes de error y las condiciones bajo las cuales los paquetes pueden ser descartados.

La unidad de transferencia básica utilizada en una red de redes TCP/IP se llama datagrama Internet, o bien, datagrama IP. Como una trama común de red física, un datagrama se divide en dos áreas: el encabezado y los datos. El encabezado contiene la dirección IP de la fuente y del destino, también contiene un campo de tipo que identifica el contenido del datagrama, además de la versión del protocolo IP, la longitud del encabezado, el tipo de servicio, la longitud total del datagrama, el tiempo de vida del datagrama, el protocolo de alto nivel que se utilizó para crear el mensaje que se está transportando en

el área de datos, el desplazamiento de fragmentación, un conjunto de banderas, una suma de verificación del encabezado y un campo de opciones IP que sirven generalmente para pruebas de red o depuración. La figura 1.6 muestra el arreglo de los campos dentro del datagrama.

El tamaño de un datagrama puede ser variable, no obstante tiene un máximo de 65,535 octetos, ya que solo se asignan 16 bits al campo de longitud total; sin embargo en protocolos recientes este límite puede modificarse.

La relación entre las tramas de las redes físicas y los datagramas se encuentra precisamente en el tamaño. Un datagrama viaja dentro de una trama física haciendo que la transmisión a través de la red física sea eficiente, esta idea es conocida como encapsulación. Un datagrama puede viajar a través de muchos tipos de redes físicas conforme se mueve a través de una red de redes hacia su destino final. Cada hardware de red tiene un límite superior fijo para la cantidad de datos que pueden transferirse en una trama física, conocido como la unidad de transferencia máxima de una red (maximum transfer unit, o MTU). Ya que el límite superior es muy variable como tecnologías de redes subyacentes hay, sería un poco complicado ajustar los datagramas a las restricciones de las redes físicas; por lo tanto es mejor hacer una selección a un tamaño de datagrama más conveniente desde el principio y establecer una forma para dividir datagramas en pequeños fragmentos cuando el datagrama necesita viajar a través de una red que tiene una MTU pequeña; a este proceso se le conoce como fragmentación y por lo general se da en el ruteador a lo largo del trayecto entre la fuente del datagrama y su destino final.

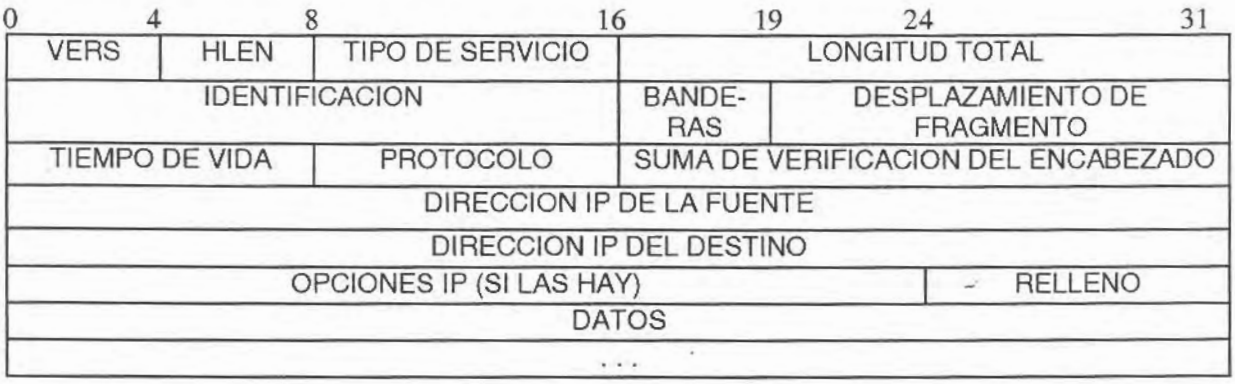


Figura 1.6 Formato de un datagrama IP

El tamaño de cada fragmento se selecciona de manera que cada uno de éstos pueda transportarse a través de la red subyacente en una sola trama. Además, dado que el IP representa el desplazamiento de datos en múltiplos de 8 octetos, el tamaño del fragmento debe seleccionarse de modo que sea un múltiplo de 8. Es menester mencionar que al seleccionar el múltiplo de 8 octetos más cercano a la MTU de la red no es usual dividir el datagrama en fragmentos de tamaños iguales; los últimos fragmentos por lo general son más cortos que los otros. Antes de que puedan procesarse en su lugar de destino, los fragmentos se deben reensamblar para producir una copia completa del datagrama original. El protocolo IP no limita los datagramas a un tamaño pequeño, ni garantiza que los datagramas grandes serán entregados sin fragmentación. Los campos IDENTIFI-

CACIÓN, BANDERAS Y FRAGMENTACION, controlan la fragmentación y el reensamblado de los datagramas.

Hay cuatro opciones en el campo de OPCIONES IP que son muy útiles e interesantes. La primera es la opción de ruteo no estricto de fuente, se utiliza para rutear un datagrama a través de una trayectoria específica; es de gran utilidad cuando se quiere verificar una determinada ruta hacia una red. La segunda es la opción de registro de ruta, y nos sirve para registrar el trayecto de una ruta; todas las redes por las cuales tenga que pasar un datagrama para llegar a su destino, serán registradas. La tercera opción es el ruteo estricto de fuente, se utiliza para establecer la ruta de un datagrama en un trayecto específico. La diferencia con el ruteo no estricto de fuente es que este último permite múltiples saltos de redes entre direcciones sucesivas de la lista de direcciones IP en la ruta a seguir y el ruteo estricto de fuente no, si algún ruteador no puede seguir la secuencia de direcciones producirá un error. Por último está la opción de sello de tiempo, se usa para registrar sellos de hora a lo largo de una ruta.

El algoritmo de ruteo se verá en el capítulo 2 y el manejo de errores, procesamiento de datagramas y la eliminación en las siguientes secciones.

1.5 Mensajes de Error y Control

Un datagrama viaja de ruteador en ruteador hasta que llega a uno que lo pueda entregar directamente a su destino final. Si un ruteador no puede rutear o entregar un datagrama, o si el ruteador detecta una condición anormal que afecta su capacidad para direccionarlo, necesita informar a la fuente original para que evite o corrija el problema.

Para permitir que los ruteadores o anfitriones (hosts, entiéndase cualquier computadora conectada a la red) en una red de redes reporten los errores o proporcionen información sobre circunstancias inesperadas, los diseñadores agregaron a los protocolos TCP/IP un mecanismo de mensajes de propósito especial, conocido como Protocolo de Mensajes de Control Internet (ICMP) [11] [12] [17].

El ICMP proporciona comunicación entre el software del Protocolo Internet en una máquina y el mismo software en otra.

Es menester mencionar que cuando un datagrama causa un error, el ICMP sólo reporta la condición del error a la fuente original del datagrama, y ya es tarea de ésta relacionar el error con un programa de aplicación individual o tomar alguna otra acción para corregir el problema.

Dado que el ICMP solo reporta los problemas a la fuente original, no se puede utilizar para informar los problemas a los ruteadores intermedios. Si en el camino un ruteador se equivoca y direcciona el mensaje hacia otro ruteador, este último no podrá reportar la anomalía al ruteador que generó el error, ya que sólo se enviará un reporte a la fuente original.

El motivo por el cual los errores se comunican solo a la fuente original es claro; ya que un datagrama solo cuenta con la dirección IP fuente y destino, y no todas las direcciones IP de las redes que va recorriendo. Además, como los ruteadores pueden establecer y cambiar sus propias tablas de ruteo, no existe un conocimiento global de las rutas. Por lo tanto, cuando un datagrama llega a un ruteador, es imposible conocer el camino que siguió para llegar hasta ahí, y si llegara a suceder un error no sería posible reportarlo a todas las máquinas que recorrió el mensaje. En vez de ello, el ruteador utiliza el ICMP para informar a la fuente que ocurrió un problema.

El formato de un mensaje ICMP se muestra en la siguiente figura:

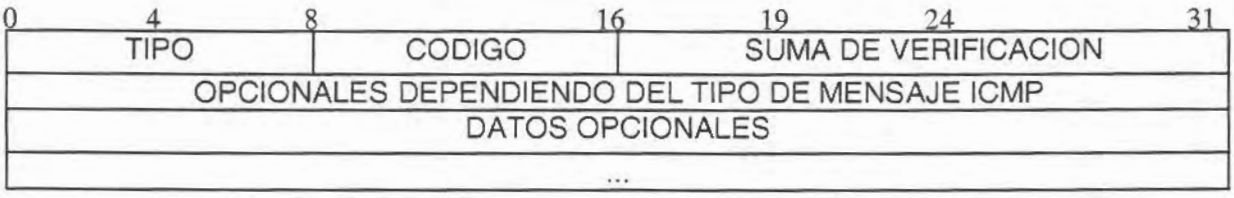


Figura 1.7 Formato general de un mensaje ICMP

El campo TIPO define el significado del mensaje así como su formato. Los tipos son:

Número	Significado
0	Respuesta de Eco
3	Destino inaccesible
4	Disminución de origen
5	Redireccionar (cambiar de ruta)
8	Solicitud de Eco
11	Tiempo excedido para un datagrama
12	Problema de parámetros en un datagrama
13	Solicitud de timestamp
14	Respuesta de timestamp
15	Solicitud de información (obsoleto)
16	Respuesta de información (obsoleto)
17	Solicitud de máscara de dirección
18	Respuesta de máscara de dirección

Figura 1.8 Tipos de mensajes ICMP

Los mensajes ICMP que reportan errores siempre incluyen el encabezado y los primeros 64 bits de datos del datagrama que causó el problema. El motivo de regresar más que el encabezado del datagrama únicamente es para permitir que el receptor determine de manera más precisa qué protocolo(s) y qué programa de aplicación es responsable del datagrama.

En nuestra aplicación se manejan los tipos:

- Destino inaccesible
- Redireccionar (cambiar de ruta)
- Tiempo excedido para un datagrama
- Solicitud de máscara de dirección
- Respuesta de máscara de dirección

Dentro del tipo destino inaccesible, los mensajes que se generan son:

- Red inaccesible
- Anfitrión inaccesible
- Red de destino desconocida
- Anfitrión de destino desconocido

1.6 Estratificación por capas de TCP/IP

Los protocolos son estándares que especifican cómo se representan los datos cuando son transferidos de una máquina a otra. También, especifican cómo se da la transferencia, cómo se detectan los errores y cómo se envían los acuses de recibo. Los sistemas complejos de comunicación de datos no utilizan un solo protocolo para manejar todas las tareas de transmisión, sino que requieren de un conjunto de protocolos cooperativos, a veces llamados familia de protocolos o conjuntos de protocolos.

Para simplificar el diseño y la implantación de los protocolos, los problemas de comunicación se transfieren hacia subproblemas que se pueden resolver de manera independiente. Cada problema se asigna a un protocolo por separado.

Visualicemos los módulos del software de protocolo en una máquina como un pila vertical constituida por capas [10]. Cada capa tiene la responsabilidad de manejar una parte del problema o una determinada tarea en la comunicación. En la figura 1.9 podemos apreciar esto.

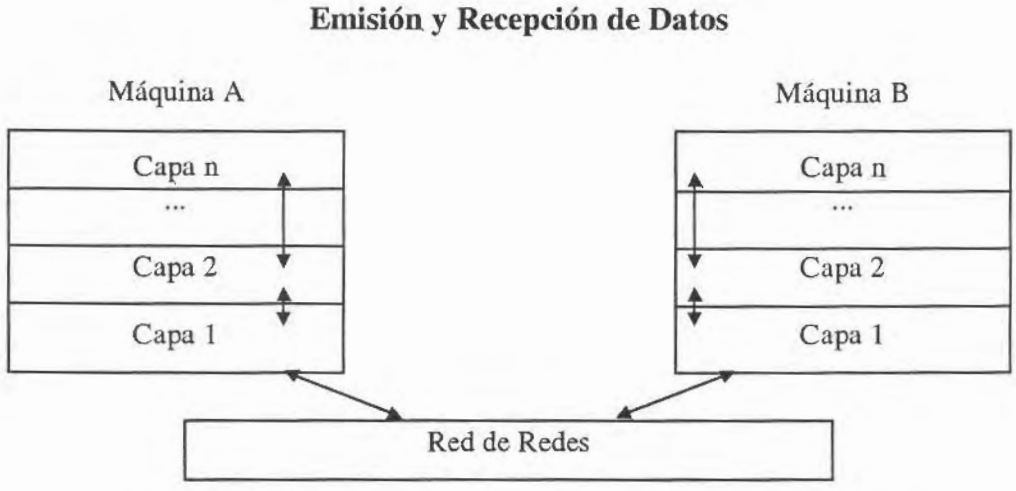


Figura 1.9 Organización del software de protocolo por módulos

Podemos darnos cuenta que al enviar un mensaje desde un programa de aplicación, por ejemplo en la máquina A, hacia un programa de aplicación en la máquina B, significa transferir el mensaje hacia abajo, por las capas sucesivas del software de protocolo en la máquina A, transferir el mensaje a través de la red y, luego transferir el mensaje hacia arriba, a través de las capas sucesivas del software de protocolo en la máquina B.

La idea de la estratificación por capas es fundamental porque proporciona una estructura conceptual para el diseño de protocolos. En este modelo por capas, cada capa maneja una parte de los problemas de comunicación y generalmente se asocian a un protocolo. Los protocolos siguen el principio de la estratificación por capas, el cual establece que la implantación del software de la capa n en la máquina B recibe exactamente la implementación del software de la capa n en la fuente de la máquina A. Con ello podemos establecer un modelo conceptual del software de protocolos TCP/IP [10] de la siguiente manera:

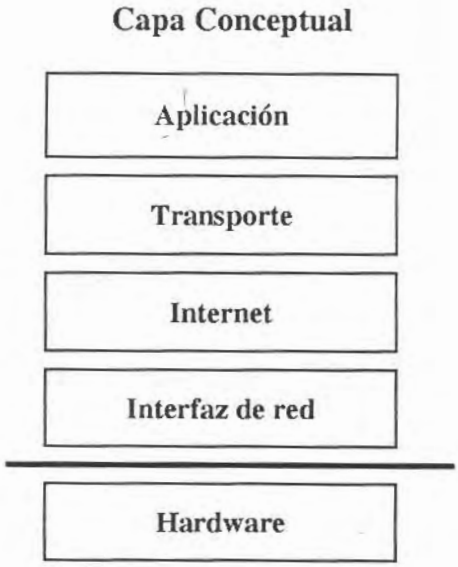


Figura 1.10 Capa conceptual del Software TCP/IP

La capa de aplicación es el nivel más alto y a través de ella el usuario puede acceder a los servicios disponibles que ofrece la Internet. La aplicación interactúa con uno de los protocolos de nivel de transporte para enviar o recibir datos. Cada programa de aplicación selecciona el tipo de transporte necesario, el cual puede ser una secuencia de mensajes individuales o un flujo continuo de octetos. El programa de aplicación pasa los datos en la forma requerida hacia el nivel de transporte para su entrega.

La capa de transporte proporciona la comunicación entre un programa de aplicación y otro. Este tipo de comunicación se conoce frecuentemente como comunicación punto a punto. La capa de transporte regula el flujo de información. Puede también proporcionar un transporte confiable, asegurando que los datos lleguen sin errores y en secuencia.

En la capa de Internet se maneja la comunicación de una computadora a otra. Esto se lleva a cabo cuando se acepta una solicitud para enviar un paquete desde la capa de transporte, junto con una identificación de la máquina, hacia la que se debe de enviar el paquete. Encapsula el paquete en un datagrama IP, llena el encabezado del datagrama, utiliza un algoritmo de ruteo para determinar si puede entregar el datagrama directamente o si debe de enviarlo a un ruteador y pasar el datagrama hacia la interfaz de red apropiada para su transmisión. También se maneja la entrada de datagramas, se verifica su validez y se hace uso de un algoritmo de ruteo para decidir si el datagrama debe procesarse de manera local o debe ser transmitido. Por último, la capa Internet envía los mensajes ICMP de error y control necesarios y maneja todos los mensajes ICMP entrantes.

El software TCP/IP de nivel inferior consta de una capa de interfaz de red responsable de aceptar los datagramas IP y transmitirlos hacia una red específica.

De una forma más detallada y en comparación con el modelo ISO (Reference Model of Open System Interconnection, Modelo de Referencia de Interconexión de sistemas abiertos) el modelo de estratificación por capas de TCP/IP de Internet [10] [13] es el siguiente:

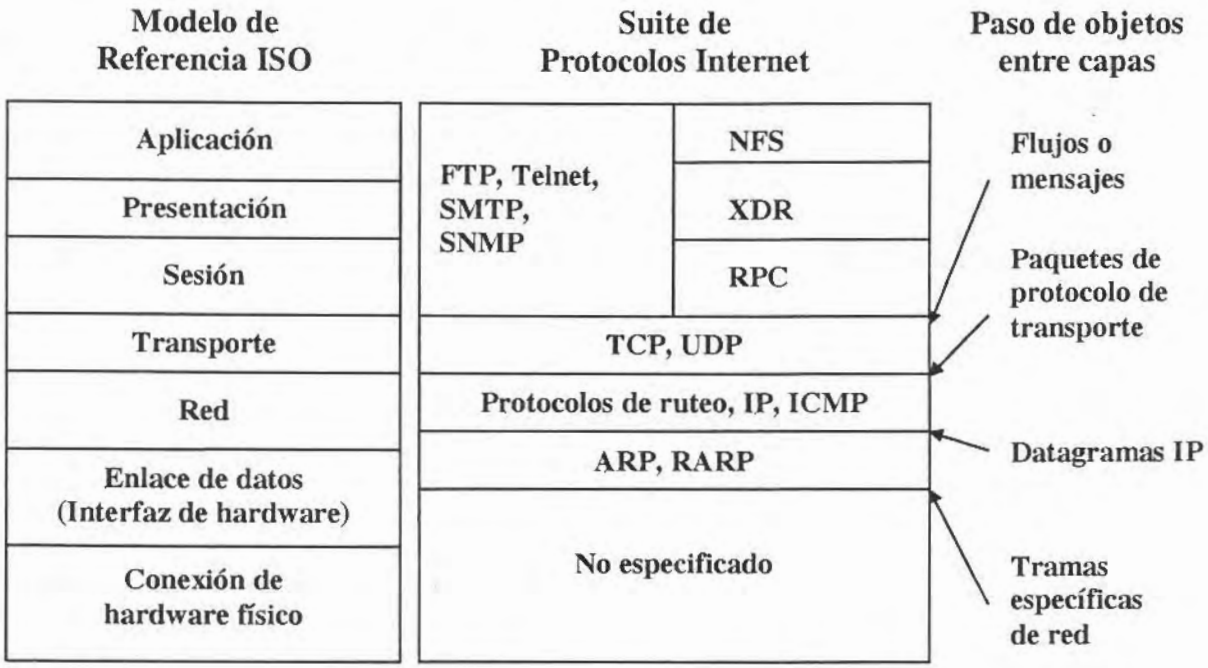


Figura 1.11 Modelo de Referencia ISO, Suite de Protocolos Internet y la forma en que los objetos pasan entre capas

Los protocolos de comunicación utilizan técnicas de multiplexado y desmultiplexado a través de la jerarquía de capas. Cuando envía un mensaje, la computadora fuente incluye bits extras que codifican el tipo de mensaje, el programa de origen y los protocolos utilizados. Finalmente, todos los mensajes son colocados dentro de tramas de red para transferirse y combinarse en flujos de paquetes. En el extremo de recepción, la máquina destino y en particular la interfaz de red demultiplexa la trama con base al tipo.

En la práctica, el software de protocolo utiliza el multiplexado y el desmultiplexado para distinguir entre varios protocolos dentro de una capa dada, haciendo el software de protocolo más complejo que como lo sugiere el modelo de estratificación por capas.

1.7 Protocolo de datagrama de usuario (UDP)

Como se mencionó anteriormente la capa de transporte proporciona la comunicación entre un programa de aplicación y otro. Luego entonces, puede parecer natural decir que un programa o proceso es el destino final de un mensaje. Sin embargo, especificar que un proceso en particular en una máquina específica es el destino final para un datagrama es un poco confuso y siendo posible que la mayoría de los sistemas operativos de computadoras permiten que varios programas de aplicación se ejecuten al mismo tiempo, la idea se hace aún más difícil.

En vez de pensar en un proceso como destino final, imaginaremos que cada máquina contiene un grupo de puntos abstractos de destino, llamados puertos de protocolo. Cada puerto de protocolo se identifica por medio de un número entero positivo. El sistema operativo local proporciona un mecanismo de interfaz síncrono que los procesos utilizan para especificar o accesar un puerto; es decir, cada vez que se acceda a un puerto, los procesos se detienen mientras dura la operación.

Para comunicarse con un puerto externo, un transmisor necesita saber tanto la dirección IP de la máquina destino como el número de puerto de protocolo del destino dentro de la máquina. Cada mensaje debe llevar el número del puerto de destino de la máquina a la que se envía, así como el número de puerto de origen de la máquina fuente a la que se deben direccionar las respuestas. Por lo tanto, es posible que cualquier proceso que recibe un mensaje conteste al transmisor.

El Protocolo de Datagrama de Usuario (UDP) [14] proporciona el mecanismo primario que utilizan los programas de aplicación para enviar datagramas a otros programas de aplicación, mediante el uso de puertos de protocolos que ayudan a distinguir los muchos programas que se pueden llegar a ejecutar en una máquina. Esto es, además de los datos, cada mensaje UDP contiene tanto el número de puerto destino como el número de puerto de origen, haciendo posible que el software UDP en el destino entregue el mensaje al receptor correcto y que éste envíe una respuesta.

El UDP utiliza el IP para transportar un mensaje de una máquina a otra dando como resultado un servicio de entrega de paquetes, sin conexión y no confiable. Esto significa que no emplea acuses de recibo para asegurarse de que llegan mensajes, no ordena los mensajes entrantes, ni proporciona retroalimentación para controlar la velocidad a la que fluye la información entre las máquinas. Por lo tanto, los mensajes UDP se pueden perder, duplicar o llegar sin orden.

El formato de los mensajes UDP es el siguiente:

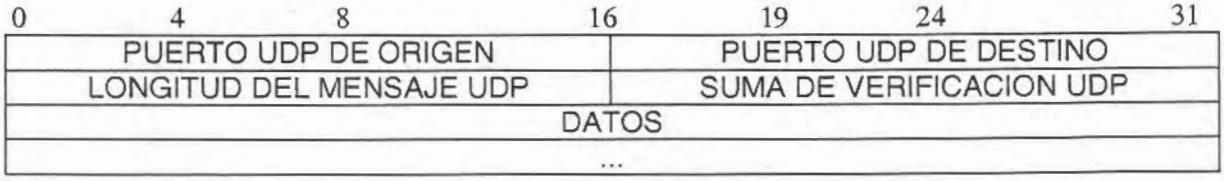


Figura 1.12 Formato de un datagrama UDP

La capa IP sólo es responsable de transferir datos entre dos máquinas dentro de la red de redes, mientras que la capa UDP solamente es responsable de diferenciar entre varias fuentes o destinos dentro de un anfitrión.

El UDP es un protocolo sencillo en el sentido de que no aumenta de manera significativa la semántica del IP.

1.8 Protocolo de Control de Transmisión (TCP)

En el nivel más alto, los programas de aplicación a menudo necesitan enviar grandes cantidades de datos de una computadora a otra. Si utilizáramos el UDP para ello nuestra transmisión correría el riesgo de no llevarse a cabo con éxito, ya que teniendo un sistema sin conexión y no confiable, indiscutiblemente que seríamos susceptibles a muchos errores. Es por eso que los diseñadores trataron de hacer un protocolo que proporcionara un servicio confiable y fuera capaz de soportar las transmisiones de grandes cantidades de datos con un alto nivel de seguridad. Este protocolo es conocido como TCP (Transfer Control Protocol, Protocolo de Control de Transferencia) [18] [12], por sus siglas en inglés.

El TCP brinda una orientación de flujo; es decir, cuando dos programas de aplicación o procesos transfieren grandes volúmenes de datos, pensamos en los datos como un flujo de bits, divididos en octetos de 8 bits (bytes), y el servicio de entrega de flujo en la máquina de destino pasa al receptor exactamente la misma secuencia de octetos que le pasa el transmisor en la máquina de origen.

Otra de las características del TCP es que la transferencia de flujo se lleva a cabo como una llamada telefónica; es decir, antes de poder empezar la transferencia, los programas de aplicación, transmisor y receptor interactúan con sus respectivos sistemas operativos, informándose de la necesidad de realizar una transferencia de flujo; así una aplicación realiza una llamada que la otra tiene que aceptar. Cuando se ha autorizado la petición y los extremos están listos, los módulos de protocolo informan a los programas de aplicación que se estableció una conexión y que la transferencia puede comenzar, en este momento se establece un circuito virtual entre ellas; ya que no existe un circuito dedicado de hardware, aunque así lo parezca, pero sí de software. Durante la transferencia, el software de protocolo en las dos máquinas continúa comunicándose para verificar que los datos se reciban correctamente y si en dado caso hay una falla, ambos lados lo detecten y lo reporten a los programas de aplicación.

Para hacer eficiente la transferencia y minimizar el tráfico de red, las implantaciones utilizan una memoria intermedia en donde recolectan datos suficientes de un flujo para llenar un datagrama razonablemente largo antes de transmitirlo a través de una red de redes. Así si un programa de aplicación genera un flujo de un octeto a la vez, la transferencia a través de una red de redes puede ser sumamente eficiente. Por el contrario, si el programa genera bloques de datos muy largos, el software de protocolo puede dividir cada bloque en partes pequeñas para su transmisión.

Es importante entender que el servicio de flujo TCP no está obligado a formar flujos estructurados de datos. Los programas de aplicación deben entender el contenido del flujo y ponerse de acuerdo sobre su formato antes de iniciar una conexión. Para ello el TCP proporciona una transferencia de datos concurrente en ambas direcciones, conocida como full duplex. La cual consiste, desde el punto de vista de un proceso de aplicación, en dos flujos independientes que se mueven en direcciones opuestas, sin ninguna interacción aparente. La ventaja de esta conexión es que el software subyacente de protocolo puede enviar en datagramas información de control de flujo al origen, llevando datos en la dirección opuesta. Este procedimiento de carga, transporte y descarga reduce el tráfico de red.

La unidad de transferencia entre el software TCP de dos máquinas se conoce como segmento. Su formato se presenta en la figura 1.13.

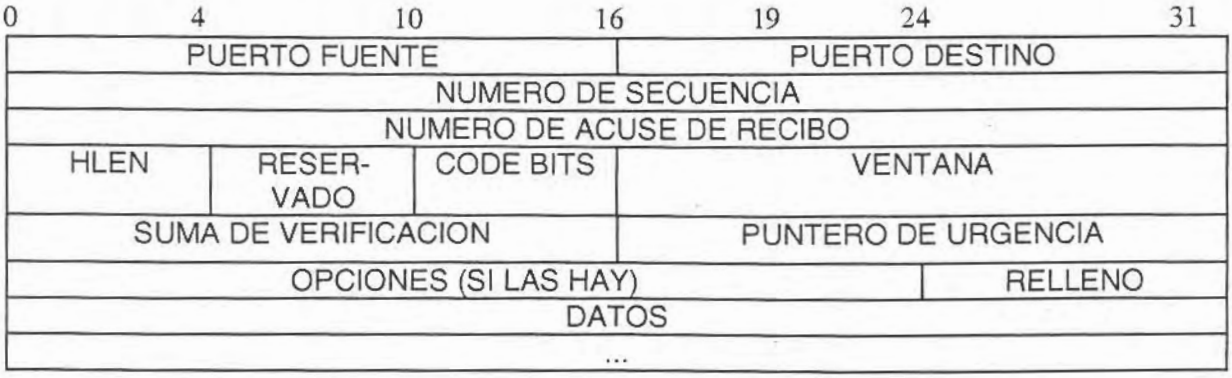


Figura 1.13 Formato de un segmento TCP

Los campos PUERTO FUENTE y PUERTO DESTINO al igual que el UDP son los números enteros de los puertos que identifican la conexión con la aplicación.

El campo NUMERO DE SECUENCIA identifica la posición de los datos del segmento en el flujo de datos del transmisor. El campo NUMERO DE ACUSE DE RECIBO identifica el número de octetos que la fuente espera recibir después. El campo HLEN especifica la longitud del encabezado del segmento, medida en múltiplos de 32 bits. El campo etiquetado como CODE BITS sirve para determinar el propósito y contenido del segmento.

Cuando una conexión se establece por primera vez, el receptor TCP asigna un búfer de K octetos y utiliza el campo VENTANA, en los segmentos de acuse de recibo, para anunciar el tamaño disponible del búfer al emisor.

Aunque el TCP es un protocolo orientado al flujo, algunas veces es importante que el programa en un extremo de la conexión envíe datos fuera de banda, sin esperar a que el programa en el otro extremo de la conexión consuma los octetos que ya están en el flujo. El campo PUNTERO DE URGENCIA se utiliza para identificar a los datos fuera de banda.

Al TCP se le asocia comúnmente como parte del grupo protocolos Internet TCP/IP, pero en realidad es un protocolo independiente de propósitos generales que se puede adaptar para utilizarlo con otros sistemas de entrega.

Capítulo 2
Protocolos y Algoritmos de Ruteo

2.1 Algoritmos de Ruteo de Datagramas IP

En este capítulo se tratará de forma más detallada el papel que juegan los ruteadores en una red de redes, los algoritmos y los protocolos que utilizan para rutear los paquetes de datos.

Como se mencionó anteriormente la interconexión y la transmisión de datos en una red de redes se establece físicamente mediante los ruteadores [16] [2] [13]. Es decir, para poder tener acceso a una red de manera externa se debe hacer por medio de un ruteador. Por lo tanto los ruteadores deben saber cómo rutear paquetes hacia su destino. Se mencionó también que los ruteadores solo utilizan la red de destino y no el host o computadora destino cuando rutean un paquete.

Para poder explicar cómo se lleva a cabo el ruteo de un mensaje, dividiremos el ruteo en dos partes: entrega directa y entrega indirecta. La entrega directa es la transmisión de un datagrama desde una máquina a través de una sola red física hasta otra; es decir, dos máquinas solamente pueden llevar a cabo la entrega directa si ambas se conectan directamente (valga la redundancia) al mismo sistema subyacente de transmisión física. La entrega directa es la base de toda la comunicación en una red de redes. Por otro lado la entrega indirecta ocurre cuando el destino no es una red conectada directamente, lo que obliga al transmisor a pasar el datagrama a un ruteador para su entrega.

La transmisión de un datagrama IP entre dos máquinas dentro de una sola red física no involucra ruteadores. La máquina fuente antes de mandar un datagrama substrae el prefijo específico de red de la dirección IP destino y la compara con la porción de red de su propia dirección IP. Si son iguales, significa que la entrega del mensaje es directa. Entonces la máquina transmisora encapsula el datagrama dentro de una trama física, transforma la dirección IP de destino en una dirección física de hardware y envía la trama resultante directamente a su destino.

La entrega indirecta se da precisamente cuando la porción de red de la dirección IP destino es distinta de la máquina transmisora, ésta debe enviar el datagrama hacia el ruteador más cercano, para que éste, si puede entregarlo directamente a su destino lo haga, de lo contrario lo envíe a otro ruteador que posiblemente lo entregue directamente a su destino final. De esta manera los ruteadores en una red de redes TCP/IP llegan a formar una estructura cooperativa e interconectada.

Para poder saber a dónde deben enviar cada datagrama los ruteadores, se valen de una tabla de ruteo Internet o tabla de ruteo IP, que se encuentra almacenada en su memoria, y contiene información sobre posibles destinos y cómo alcanzarlos. Esta tabla también es utilizada por los hosts que deben decidir hacia qué ruteador deben mandar cada mensaje para que llegue a su destino final.

Por lo regular la información que contiene la tabla de ruteo está distribuida en pares: la dirección IP de una red destino y la dirección IP del siguiente ruteador en el camino hacia dicha red. Con esto último sólo se especifica un paso a lo largo del camino hacia la red de destino, por lo tanto el ruteador que contiene dicha tabla no conoce el camino completo hacia el destino final.

Es menester recalcar que cada registro en una tabla de ruteo apunta hacia un ruteador que se pueda alcanzar a través de una sola red. Esto es, que todos los ruteadores

listados en la tabla de ruteo de la máquina anfitrión o ruteador deben residir en las redes con las que la máquina anfitrión se conecta de manera directa.

Por otra parte, esta manera de ordenar la información nos permite almacenar solamente la porción de red de una dirección de destino en vez de toda la dirección del host, consecuentemente el ruteo es más eficiente y mantiene reducidas las tablas de ruteo.

Además de esto las tablas de ruteo de los hosts o anfitriones deben permanecer con el mínimo de información; ya que la idea es obligar a los hosts a que deleguen la mayor parte de sus funciones de ruteo a los ruteadores.

Supongamos que tenemos cuatro redes interconectadas como se muestra en la figura 2.1. La tabla de ruteo del ruteador R1 sería como se muestra en la figura 2.2.

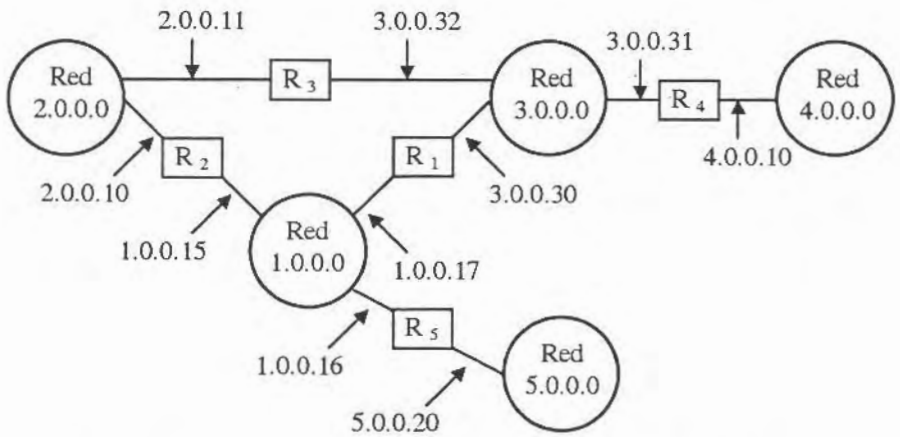


Figura 2.1 Red de redes interconectada por 3 ruteadores

Hacia la Red:	Rutear a:
1.0.0.0	Entrega Directa
3.0.0.0	Entrega Directa
2.0.0.0	3.0.0.32
4.0.0.0	3.0.0.31
5.0.0.0	1.0.0.16

Figura 2.2 Tabla de ruteo en R_1

Nótese que hacia la red 2.0.0.0 hay dos caminos: el primero es por el ruteador 3.0.0.32 y el segundo, por el ruteador o gateway 1.0.0.15. ¿ Cómo decide el ruteador R_1 que ruta tomar para direccionar los mensajes para dicha red?. Esta pregunta se responderá en las siguientes secciones donde se explica la forma de actualizar la información y el intercambio de las tablas de ruteo entre ruteadores; ya que en tiempo real los sistemas son susceptibles a fallas y cambios de diversa índole. Por lo que los ruteadores constan-

temente están verificando la viabilidad de las rutas para poder decidir en caso de que se presente que ruta es la más óptima en cuanto a velocidad y disponibilidad de entrega.

En muchos casos la red está interconectada por un solo ruteador, como la red 5.0.0.0 o la 4.0.0.0 del ejemplo anterior; o bien, muchos registros de la tabla de ruteo están asociados a un ruteador. En ambas situaciones se utiliza una ruta predeterminada; esto con la finalidad de mantener reducido el tamaño de las tablas de ruteo. Por lo tanto, si en la búsqueda de ruteador para el manejo de un mensaje no se llegara a encontrar, los paquetes son direccionados a la ruta asignada por omisión.

Aunque se ha dicho que todo el ruteo está basado en redes y no en hosts individuales, en forma práctica es permitido que se especifiquen rutas por anfitrión como caso especial. Esto con la finalidad de tener un mayor control de la red local, hacer comprobaciones, depuraciones de conexiones de red o tablas de ruteo y para controlar el acceso por razones de seguridad.

Resumiendo el algoritmo de ruteo IP es el siguiente:

RutaDatagrama(Datagrama, Tabla de Ruteo)

1. - Extraer la dirección IP de destino, D, del datagrama y computar el prefijo de red, N;
2. - Si N corresponde a cualquier dirección de red directamente conectada entregar el datagrama al destino D sobre dicha red. (Esto comprende la transformación de D en una dirección física, encapsular el datagrama y enviar la trama).
   De otra forma, si la tabla contiene una ruta con anfitrión específico para D, enviar el datagrama al salto siguiente especificado en la tabla;
   de otra forma, si la tabla contiene una ruta para la red N, enviar el datagrama al salto siguiente especificado en la tabla;
   de otra forma, si la tabla contiene una ruta asignada por omisión, enviar el datagrama al ruteador asignado por omisión especificado en la tabla;
   de otra forma, declarar un error de ruteo;

2.2 Ruteo de Subred y Superred

En muchos lugares, frecuentemente se utilizan pequeñas redes físicas locales con una sola dirección de red IP asignada [17] [12]. Esto con la finalidad de tener un mayor control de administración.

Para poder manejar la información, los ruteadores internos saben que la red está compuesta por muchas redes o subredes físicas. Los ruteadores externos no lo saben, ellos solamente conocen la dirección de la red en su totalidad; es decir, al conjunto de redes físicas la reparan como una sola red.

El TCP/IP permite la asignación de direcciones de subred a cada red física a través del manejo de máscaras – números de 32 bits, como las direcciones IP -. De la misma forma en que se divide la porción de red y anfitrión como se hace de forma estándar a las direcciones IP, se lleva a cabo la separación entre subred y host. Por ejemplo, si se tuviera una red tipo B y quisiéramos dividirla en subredes tipo C, lo que haríamos sería establecer una máscara compuesta por 24 bits puestos en uno y los 8 restantes en cero. Así los unos indicarían la porción de red o subred y los ceros el host:

Dirección Tipo B	100.15.0.0
Máscara de Subred	255.255.255.0
Direcciones Tipo C	100.15.1.0 - 100.15.255.0

Los ruteadores externos solo tendrían la dirección 100.15.0.0; mientras que los ruteadores locales sabrían que hay 255 redes. Esto hace que el algoritmo de ruteo sea distinto:

Ruta_IP_Datagrama(Datagrama, Tabla de Ruteo)

1. Extraer la dirección IP de destino, ID, del datagrama:
2. Computar la dirección IP de la red de destino, IN;
3. Si N corresponde a cualquier dirección de red conectada, enviar el datagrama a su destino a través de dicha red. (Esto comprende la transformación de ID en una dirección física, encapsular el datagrama y enviar de la trama).
   De otra forma,
   para cada registro en la tabla de ruteo hacer lo siguiente:
   Dejar que N sea el bitwise-and de ID y de la máscara de subred
   Si N es igual al campo de dirección de red del registro, entonces rutear el datagrama a la dirección especificada de salto al siguiente
   fin-de-ciclo
   Si no se encuentra correspondencia, declarar un error de ruteo;

Para el manejo de máscaras, el TCP/IP es muy flexible; ya que permite que el direccionamiento de subred se haga de forma arbitraria. Sin embargo, en la práctica se recomienda que las localidades utilicen máscaras contiguas y que las redes físicas con la misma dirección IP tengan la misma máscara. Todo ello para que las tablas de ruteo no sean ambiguas.

Por el contrario el direccionamiento de superred utiliza muchas direcciones IP para una solo ámbito o lugar. El ruteo en este caso se vuelve un poco complicado y principalmente las tablas de ruteo aumentan de tamaño considerablemente; ya que en vez de tener un registro para cada lugar, contiene muchos registros por cada uno.

Este problema se ha resuelto mediante el Ruteo sin tipo de inter-dominio (CIDR). El cual consiste en formar grupos en potencias de dos de direcciones IP contiguas y una máscara de bit que indica el tamaño del grupo. Los ruteadores que funcionan bajo este algoritmo utilizan un paradigma de correspondencia mayor para seleccionar la ruta. En la búsqueda no necesita la dirección corresponder a un valor binario.

2.3 Sistemas Autónomos

Hasta este momento solamente se ha mencionado la manera en que se direccionan los paquetes de información hacia su destino. El cual está basado en una tabla de ruteo. Pero, ¿cómo es que se obtiene dicha tabla?. En la práctica las redes de comunicación están expuestas a muchos cambios, ¿de qué manera repercuten en las tablas de ruteo?. Para poder responder a estas preguntas analicemos primeramente la forma en que está organizada una red de redes.

Cuando hablamos de una red, por pequeña que esta sea, tiene una estructura y una administración; tanto en el hardware como en el software. Visualizando una red de redes, indiscutiblemente que se encuentra estructurada y administrada para su buen funcionamiento. En el ruteo es indispensable.

Dividamos la red de redes en subgrupos de redes, que cada subgrupo intercambie información internamente sobre sus redes y gateways existentes. Cada subgrupo tendrá un ruteador que intercambiará información con cada ruteador de cada subgrupo, sobre las redes y gateways que existen en su grupo.

A cada subgrupo se le conoce con el nombre de sistema autónomo [16] [13] y a su ruteador como ruteador núcleo.

Esta forma de administración se lleva a cabo en la Internet. Los motivos son diversos, los más importantes son mayor rapidez en la actualización de la tabla de ruteo, con ello queremos dar a entender que cada ruteador debe, continuamente, intercambiar la información de su tabla de ruteo a todos los ruteadores vecinos (conectados directamente al mismo sistema subyacente de transmisión física). También se tiene un mayor control en fallas y cambios. La información que se maneja entre los ruteadores en mínima y consistente.

Así, de manera general, es como cada ruteador obtiene su tabla de ruteo y cómo la restaura ante posibles cambios.

Formalmente, un sistema autónomo es un grupo de redes y ruteadores controlados por una sola autoridad administrativa. Los ruteadores dentro de un sistema autónomo son libres de seleccionar sus propios mecanismos de exploración, propagación, validación y verificación de la consistencia de las rutas. A estos mecanismos se le conocen como Protocolos de Ruteo Interno (IGP). Examinaremos uno de ellos: RIP.

Los ruteadores núcleo, también tienen normas y mecanismos para poder explorar, propagar, validar y verificar la consistencia de las rutas. A estos se le conocen como Protocolos de Ruteo Externo.

2.4 Protocolo de Información de Ruteo

El RIP es un protocolo basado en el algoritmo Bellman-Ford (vector distancia), que consiste en establecer una lista con todas las direcciones IP de las redes conocidas, junto con el primer gateway de la ruta hacia dicha red, la red física que se utiliza, la distancia al destino, esto es el número de ruteadores que debe pasar para llegar a la red destino mencionada y el tiempo que debe pasar para recibir la actualización de la tabla.

Con base en esto, cada ruteador o host que tiene implementado RIP tiene en su tabla de ruteo por lo menos los siguientes datos:

- La dirección IP de la red destino.
- Distancia o costo total que toma en enviar un datagrama desde la fuente hacia el destino.
- La dirección IP del siguiente ruteador a través de la ruta al destino final.
- Una bandera que indica que la información del ruteador ha sido cambiada recientemente.
- Varios temporizadores asociados con la ruta propuesta.

El RIP divide las máquinas participantes en activas y pasivas (silenciosas). Las máquinas activas comunican sus rutas a las demás máquinas. Las silenciosas no lo hacen. Los ruteadores asumen el papel de activos; a menos que algún ruteador perdiera contacto con todos, podría alguna red considerarlo como pasivo. Los hosts son declarados como pasivos. Los ruteadores cada 30 segundos envían un mensaje de respuesta (response) a sus ruteadores vecinos con sus rutas y destinos disponibles. Los host solo escuchan y actualizan sus rutas con base a estos mensajes. En la versión 2.0 del RIP, los hosts ya no escuchan los mensajes de actualización, ya que éstos son entregados por multidifusión.

En la figura 2.3 se muestra el formato de los mensajes RIP.

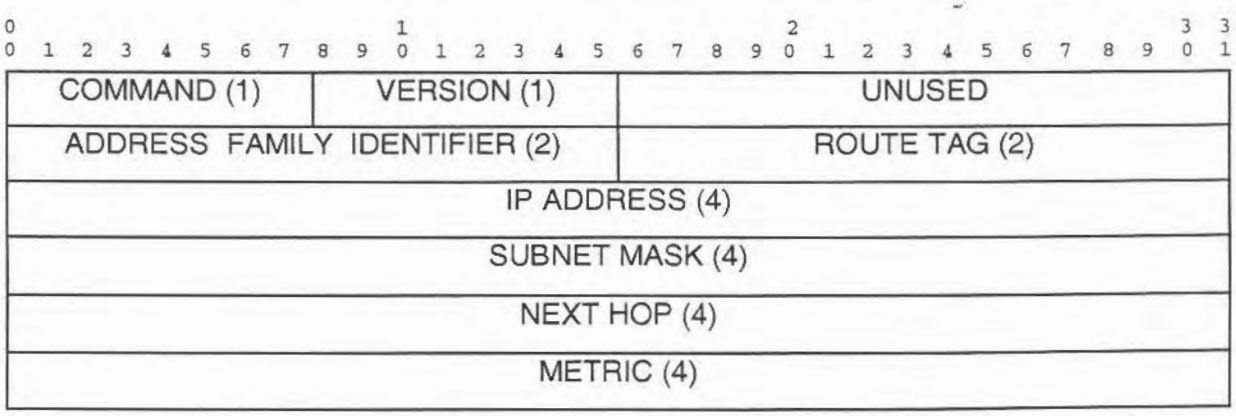


Figura 2.3 Formato de un mensaje RIP

Donde COMMAND (1 octeto) especifica el propósito del datagrama:

1 Request	Solicitud para información parcial o total de la tabla de ruteo.
2 Response	Respuesta o mensaje con información parcial o total de la tabla de ruteo del emisor.

El campo VERSION (1 octeto) especifica el número de la versión del RIP, actualmente 2; es usado para verificar e interpretar el mensaje de forma correcta. El campo UNUSED (2 octetos) debe ser ignorado. El campo ADDRESS FAMILY IDENTIFIER (2 octetos) indica que tipos de direcciones están especificadas en cada entrada. Para las direcciones IP es el número 2. El campo ROUTE TAG es un atributo asignado a cada ruteador con la finalidad de separar los ruteadores RIP internos (aquellos que solo utilizan RIP) de los ruteadores RIP externos (aquellos que emplean además de RIP otros protocolos).

Los siguientes cuatro campos muestran:

- La dirección IP del destino
- Su máscara de subred
- La dirección del ruteador salto al siguiente
- El número de hops o saltos hacia dicha red, es decir el costo total o la métrica

La máscara de subred, como se mencionó anteriormente, sirve para separar la porción del hostid de la del netid, en este caso la subred. Si este campo es cero, entonces la máscara de subred no debe ser incluida en el registro de la tabla. El ruteador asume que la máscara a utilizar es la suya.

La información interna de cualquier red no debe ser conocida por las demás redes. El manejo de máscaras de subred no se aplica a ruteadores que manejen la versión RIP-1. La emulación está basada en la versión RIP-2 definida en el RFC 2453.

Cuando el campo NEXT HOP contiene el valor de 0.0.0.0, significa que el ruteador del salto al siguiente es el que mandó el mensaje RIP, de lo contrario los mensajes hacia la red indicada deben ser enviados hacia la dirección que se indica. El objetivo de este campo es solo informativo.

Normalmente los mensajes request son mandados por difusión. Los host no responden a éste, solo los ruteadores presentes en la red.

Un mensaje request es procesado registro por registro. Se busca el destino en la tabla de ruteo del ruteador que lo recibe, si hay alguna ruta alternativa a dicho destino, entonces el costo es copiado en el campo METRIC del datagrama. Por el contrario, si no existe alguna ruta entonces el costo total del mensaje request toma el valor de infinito.

Ya que se han buscado todos los destinos, el campo COMMAND cambia a response y se manda el datagrama al solicitante. Si no hay ningún registro en el mensaje, no se responde a este. Solo si, exactamente existe un registro, con el campo ADDRESS

FAMILY puesto en cero y la métrica de infinito (16). Significa que solicitan toda la tabla de ruteo. Esto sucede cuando un ruteador inicializa su funcionamiento por primera vez. Por lo regular los mensajes request son generados por algún software de monitoreo o diagnóstico.

Es menester mencionar que si la petición es para toda la tabla de ruteo, el proceso de salida incluye la actualización de horizonte separado (split horizont update), el cual consiste en no mandar en el mensaje response los destinos por los que puedo llegar a través del ruteador que lo solicita. Por otra parte, si la petición es para registros con destinos específicos, la actualización de horizonte separado no se realiza, ya que los datos que se mandan son utilizados para otros propósitos y no intervienen en la información para el ruteo. En cambio si un ruteador solicita toda la tabla de ruteo es porque inicia su actividad y necesita actualizar su tabla con las rutas existentes, por eso es necesaria la actualización de horizonte separado, para evitar la convergencia lenta.

Como se dijo los mensajes response pueden incluir todo o parte de la tabla de ruteo y son generados, ya sea por que:

- se ha recibido una petición request
- se actualiza (cada 30 segundos) con los demás ruteadores
- se ha recibido alguna modificación en alguna ruta o destino (trigger-update)

Tienen como límite 25 registros por mensaje y en su elaboración se aplica la actualización de horizonte separado; excepto, como se mencionó anteriormente, cuando se genera por una petición con registros de destinos específicos. En el proceso trigger-update además del split horizont update, solo se agregan al mensaje los registros que hayan cambiado en su información, como se indica más abajo, en un intervalo de 1 a 5 segundos, para evitar colisiones en la red, generando un exceso de mensajes.

Los registros de las tablas de ruteo tienen un tiempo de vida (timeout) de 180 segundos, si no son actualizados entonces inicia el proceso de borrado, que consiste en asignarle la métrica de infinito a la ruta en la tabla de ruteo, inicializar un temporizador de 120 segundos (garbage-collection), colocar la bandera ruta como ruta cambiada y activar el proceso trigger-update.

El uso del temporizador garbage-collection permite que los demás ruteadores se enteren de la inaccesibilidad de la ruta (poison-reverse) para evitar la convergencia lenta. Si se llega al termino del tiempo, entonces se borra por completo el registro de la tabla de ruteo.

Cuando se recibe un mensaje response, se debe verificar primeramente que el mensaje proviene de algún ruteador válido, es decir de un ruteador vecino.

Ya que se ha comprobado su autenticidad los registros son procesados uno por uno de la siguiente manera:

1. Es válido el registro, si la métrica del destino esta entre el intervalo 1 y 16. Si la dirección destino es distinta de direcciones IP de clase D, E, por difusión, 127 (loopback) o cero.

   Otro caso, se descarta el registro y se pasa al siguiente registro.

2. Ya que se valida el registro, se incrementa la métrica en uno, si sobrepasa el valor de 16 se asigna 16.
3. Se busca una ruta disponible (que exista una ruta y que su métrica sea distinta de infinito) para el destino en la tabla de ruteo.
4. Si no se ha encontrado se agrega la ruta a la tabla de ruteo, siempre y cuando la nueva métrica sea distinta que infinito, de lo contrario no se agrega, no tiene sentido agregar rutas que son inaccesibles. Para agregar una ruta a la tabla de ruteo se hace lo siguiente:

    - Se ajusta Tddestino con Dddestino
    - Se ajusta Tmetrica con Nmetrica
    - Se coloca en TNextHop la dirección del ruteador que mandó el mensaje response
    - Se inicializa el temporizador timeout para la ruta. Si el temporizador garbage-collection está corriendo para esta ruta, se debe parar
    - Se coloca la bandera ruta actualizada
    - Se activa el proceso trigger-update

    Si se halló una ruta disponible para el destino entonces:

    1. Se comparan TNextHop con la dirección del ruteador que mandó el mensaje. Si son iguales, se reinicializa el temporizador timeout.

    2. Se comparan las métricas. Si TNextHop es igual que la dirección del ruteador que mandó el mensaje y Nmetrica es distinta que Tmetrica o bien, si Nmetrica es menor que Tmetrica entonces se hace lo siguiente:

        - Se substituye la Tmetrica por Nmetrica
        - Se ajusta TNextHop, si es necesario
        - Se coloca la bandera ruta actualizada
        - Se activa el proceso trigger-update
        - Si Nmetrica es infinita entonces comienza el proceso de borrado, en otro caso se reinicializa el temporizador timeout. Nótese que si Nmetrica es infinita, solo se inicia el proceso de borrado solo si no ha sido inicializado, de lo contrario no se hace nada

    3. Si Tmetrica y Nmetrica son iguales y TNextHop a presentado retardos en su actualización, es decir, ha pasado más de la mitad del tiempo que es permitido para recibir la actualización de la ruta, entonces la ruta actual es substituida por la nueva ruta.

Como se puede ver el RIP está limitado a redes que tienen un diámetro de red de 15 saltos, por lo que en sistemas autónomos grandes su implementación es inapropiada. Aunque la convergencia lenta no es mucho problema con las heurísticas aplicadas, en redes lentas puede ser problema, ya que se pueden forma ciclos de ruteo, los cuales tardarían mucho tiempo, además de consumo de recursos (ancho de banda), en resolverse. El RIP es un protocolo que depende de las métricas y en situaciones que se requieran el uso de rutas alternativas es inútil su uso.

Capítulo 3
Sockets

3.1 Modelo Cliente-Servidor

El patrón de interacción primario que se da entre las aplicaciones de cooperación se conoce como paradigma cliente-servidor [2]. La interacción cliente-servidor forma la base de la mayor parte de la comunicación por redes y es fundamental ya que nos ayuda a comprender las bases sobre las que están construidos los algoritmos distribuidos.

El término servidor se aplica a cualquier programa que ofrece un servicio que se puede obtener en una red. Un servidor acepta la petición desde la red, realiza el servicio y devuelve el resultado al solicitante. En el caso de los servicios más sencillos, cada petición llega en un solo datagrama IP y el servidor devuelve una respuesta en otro datagrama.

Un programa ejecutable se convierte en un cliente cuando manda una petición a un servidor y espera una respuesta. Debido a que el modelo cliente-servidor es de extensión conveniente y natural en la comunicación de interproceso en una sola máquina, es fácil construir programas que utilicen el modelo para interactuar.

Los servidores pueden ejecutar tareas simples o complejas. Por ejemplo, un servidor hora del día simplemente devuelve la hora actual cuando un cliente manda un paquete al servidor. Un servidor de archivo recibe las peticiones para realizar las operaciones de almacenaje o recuperación de datos de un archivo; el servidor realiza la operación y devuelve el resultado.

Los servidores se suelen implantar como aplicaciones de programas (proceso usuario). La ventaja de implantar los servidores como programas de aplicación es que pueden ejecutarse en cualquier sistema computacional que soporte la comunicación TCP/IP. De este modo, el servidor de un servicio en particular puede ejecutarse en un sistema de tiempo compartido junto con otros programas o en una computadora personal. Los servidores múltiples pueden ofrecer el mismo servicio y ejecutarse en la misma máquina o en múltiples máquinas. De hecho, comúnmente se duplican copias de un servidor dado en máquinas físicamente independientes para incrementar la disponibilidad o mejorar la ejecución. Si el propósito principal de una computadora es apoyar un programa servidor en particular, el término "servidor" se puede aplicar tanto a la computadora como el programa servidor. De este modo, podemos escuchar frases como "la máquina N es nuestro servidor de archivos".

Nuestro programa de aplicación representa este paradigma, ya que en ocasiones es cliente o servidor.

3.2 Sockets

Como preámbulo para poder comprender el papel que juega el Winsock en el sistema operativo Windows 95, analizaremos primeramente que es un socket bajo UNIX [2].

En la sección anterior se vio el modo de interacción entre aplicaciones. En capítulos anteriores se mostraron algunos rasgos de la suite de protocolos TCP/IP. En esta sección conoceremos la interfaz que existe entre los programas de aplicación y el software de protocolo llamada socket (Winsock para Windows).

UNIX se diseñó originalmente como un sistema de tiempo compartido para computadoras de un solo procesador que sigue un paradigma denominado open-read-write-close. Se trata de un sistema operativo orientado al proceso, en el que cada programa de aplicación se ejecuta como un proceso de nivel de usuario. Un programa de aplicación interactúa con el sistema operativo haciendo llamadas de sistema. Desde el punto de vista del programador, las llamadas de sistema se ven y comportan exactamente igual que las demás llamadas de procedimientos. Toman argumentos y devuelven uno o más resultados. Los argumentos pueden ser valores (por ejemplo, una operación de enteros) o punteros a objetos en el programa de aplicación (como un búfer que ha de ser llenado de caracteres).

Antes de que un proceso de usuario pueda ejecutar operaciones de E/S, llama a open para especificar el archivo o dispositivo que se va a usar y obtiene el permiso. La llamada a open devuelve un pequeño entero descriptor de archivo que el proceso utiliza cuando ejecuta las operaciones de E/S en el archivo o dispositivo abierto. Una vez que se ha abierto un objeto, el proceso de usuario hace una o más llamadas a read o write para transferir datos. Read transfiere datos dentro del proceso de usuario; write transfiere datos del proceso de usuario al archivo o dispositivo. Tanto read como write toman tres argumentos que especifican el descriptor de archivo que se ha de usar, la dirección de un búfer y el número de octetos que se han de transferir. Luego de completar todas las operaciones de transferencias, el proceso de usuario llama a close para informar al sistema operativo que ha de terminar de usar el objeto.

De primera instancia esto bastó para implantar TCP/IP bajo UNIX; sin embargo, la interacción entre los procesos del usuario y los protocolos de red era aún más compleja que las interacciones entre los procesos del usuario y las instalaciones convencionales de E/S. En particular, la interfaz de protocolos debía permitir a los programadores crear un código de servidor que esperara las conexiones pasivamente, así como también un código cliente que formara activamente las conexiones. Además, los programas de aplicación que mandaban datagramas podían especificar la dirección de destino junto con cada datagrama en lugar de destinos enlazados en el momento en que llamaban a open. Esto ocasionó que el paradigma open-read-write-close sufriera muchos cambios para servir como interfaz entre los programas de aplicación y protocolos.

A esta abstracción se le conoce como socket. En UNIX los sockets forman parte del sistema operativo. En otros sistemas operativos se utilizan las bibliotecas de rutinas para proporcionar la interfaz socket. Un socket proporciona un punto final para la comunicación. Al igual que con el acceso a archivos, los programas de aplicación requieren que el sistema operativo cree un socket cuando se necesita. El sistema operativo devuelve un entero pequeño que utiliza el programa de aplicación para hacer

referencia al socket recientemente creado. La diferencia principal entre los descriptores de archivos y los descriptores de socket es que el sistema operativo en laza un descriptor archivo a un archivo o dispositivo especifico cuando la aplicación llama a open, pero puede crear sockets sin enlazarlos a direcciones de destino específicas. La aplicación puede elegir abastecer una dirección de destino cada vez que utiliza el socket (es decir, cuando se envían datagramas) o elegir enlazar la dirección de destino a un socket y evadir la especificación de destino repetidamente (es decir, cuando se hace una conexión TCP).

El Winsock implementa esta abstracción socket para Windows. Las bibliotecas de rutinas están basadas en la E/S de red de BSD de UNIX.

3.3 Windows Sockets y OWLSock

Windows Sockets, comúnmente llamado Winsock, enmarcan la interface de programación sobre redes para Microsoft Windows, el cual está basado en la paradigma "socket" popularizado en Berkeley Software Distribution (BSD) de la Universidad de California en Berkeley. Esta abarca tanto las rutinas socket de Berkeley como un grupo de extensiones Windows diseñadas para que el programador tome ventaja del manejo de eventos original de Windows.

Winsock tiene la intención de proporcionar una sola interface de programación (API) para desarrollo de aplicaciones de redes. Además, en el entorno de una versión particular de Windows, ésta define una interface binaria (ABI) tal que una aplicación escrita para Windows Sockets API pueda trabajar en cualquier protocolo e implementación de cualquier red física. Con ello se definen librerías llamadas y asociadas a semánticas las cuales los desarrolladores pueden programar para cualquier marca de software de red e implementación.

Windows Sockets está definido y documentado para usar API junto con la Suite TCP/IP. Concretamente, Windows Sockets soporta tanto conexiones TCP y UDP.

Con base en Winsock surge OWLSock (Object Windows Library Sockets), librerías de clases de Borland [19]. Estas encapsulan y simplifican la mayoría de características de Winsock 1.1 [18]. OWLSock encapsula:

- Direcciones
- Direcciones Internet
- Sockets
- Stream Sockets
- Datagram Sockets
- Acceso de Host con el manejo de bases de datos
- Servicios de bases de datos
- Errores
- Reporte de errores

Es compatible con Win16, Win32s y Win32. También proporciona compatibilidad con modalidades de sockets síncronas (blocking) y asíncronas (non-blocking). Las siguientes tablas listan las clases OWLSock con su correspondiente estructura Winsock y las funciones Winsock con sus correspondientes funciones OWLSock.

Estructuras equivalentes OWLSock:

Winsock Structure	**OWLSock Structure**
Sockaddr	TSocketAddress or sockaddr
Sockaddr_in	TINetSocketAddress
Hostent	THostEntry or hostent
Protoent	Protoent only
Servent	TServiceEntry or servent
WSAData	TSocketInfo or WSAData

Funciones equivalentes OWLSock:

Winsock Function	OWLSock Class::Function
accept()	TStreamSocket::Accept(TStreamSocket& socket)
	TStreamSocket::Accept(SOCKET& socket, sockaddr& sAddress)
bind()	int TSocket::BindSocket(TSocketAddress& boundSocketAddress)
	int TSocket::BindSocket()
closesocket()	int TSocket::CloseSocket()
connect()	TStreamSocket::Connect()
getpeername()	int TSocket::GetPeerAddress(TSocketAddress& socketAddress, int& nAddressLength, SOCKET& socket)
	int TSocket::GetPeerAddress(TSocketAddress& socketAddress, int& nAddressLength)
getsockname()	int TSocket::GetMyAddress(TSocketAddress& socketAddress, int& nAddressLength, SOCKET& socket)
	int TSocket::GetMyAddress(TSocketAddress& socketAddress, int& nAddressLength)
getsockopt()	int TSocket::SetBroadcastOption(BOOL bBroadcast);
	int TSocket::SetDebugOption(BOOL bDebug);
	int TSocket::SetLingerOption(BOOL bLinger, u_short nLingerTime=0);
	int TSocket::SetRouteOption(BOOL bRoute);
	int TSocket::SetKeepAliveOption(BOOL bKeepAlive);
	int TSocket::SetOOBOption(BOOL bSendOOBDataInline);
	int TSocket::SetReceiveBufferOption(int nReceiveBufferSize);
	int TSocket::SetSendBufferOption(int nSendBufferSize);
	int TSocket::SetReuseAddressOption(BOOL bAllowReuseAddress);
htonl()	::htonl() only
htons()	::htons() only
inet_addr()	u_long TINetSocketAddress::ConvertAddress(char* szAddress)
inet_ntoa()	char* TINetSocketAddress::ConvertAddress(u_long lAddress)
ioctlsocket()	unsigned long TSocket::GetDriverWaitingSize()
listen()	TStreamSocket::Listen()
ntohl()	::ntohl() only
ntohs()	::ntohs() only
recv()	int TStreamSocket::ReceiveQueue(DataQueue& queue, int nFlags)
	TStreamSocket::DoReadNotification(const SOCKET& /*socket*/, int nError)
	TStreamSocket::DoOOBNotification(const SOCKET& /*s*/, int nError)
recvfrom()	int TDatagramSocket::ReceiveQueue(DataQueue& queue, int nFlags)

	TDatagramSocket::DoReadNotification(const SOCKET& /*socket*/, int nError)
select()	::select() only
send()	TStreamSocket::AttemptDataSend(DataQueue& queue, int nFlags)
	int TStreamSocket::Write(char* chData, unsigned long lLength, int nFlags, short bBecomeOwnerOfData, short bCopyData)
	TStreamSocket::DoWriteNotification(const SOCKET& /*s*/, int nError)
sendto()	TDatagramSocket::AttemptDataSend(DataQueue& queue, int nFlags)
	int TDatagramSocket::Write(SocketAddress& thePeerSocketAddress, char* chData, unsigned long lLength, bBecomeOwnerOfData, short bCopyData)
	int TDatagramSocket::Write(char* chData, unsigned long Llength, short bBecomeOwnerOfData, short bCopyData)
	TDatagramSocket::DoWriteNotification(const SOCKET& /*s*/, int nError)
setsockopt()	int TSocket::GetBroadcastOption(BOOL& bBroadcast);
	int TSocket::GetDebugOption(BOOL& bDebug);
	int TSocket::GetLingerOption(BOOL& bLinger, u_short& nLingerTime);
	int TSocket::GetRouteOption(BOOL& bRoute);
	int TSocket::GetKeepAliveOption(BOOL& bKeepAlive);
	int TSocket::GetOOBOption(BOOL& bSendOOBDataInline);
	int TSocket::GetReceiveBufferOption(int& nReceiveBufferSize);
	int TSocket::GetSendBufferOption(int& nSendBufferSize);
	int TSocket::GetReuseAddressOption(BOOL& bAllowReuseAddress);
htonl()	::htonl()
shutdown()	TSocket::ShutDownSocket
socket()	TSocket::CreateSocket()
gethostbyaddr()	int THostInfoManager::GetHostInfo(HostEntry*& Hentry, const TsocketAddress& sAddress)
	int THostInfoManager::GetHostInfoAsync(HANDLE& hTheHostRequest, TSocketAddress& sAddress)
	int THostInfoManager::GetHostInfoAsync(HWND hwndNotify, UINT nMessage, HANDLE& hTheHostRequest, TsocketAddress& sAddress)
gethostname()	int THostInfoManager::GetHostName(char* szName, int nNameLength)
gethostbyname()	int THostInfoManager::GetHostInfo(THostEntry*& hEntry, const char* szName)
int HostInfoManager::GetHostAddress(char* szHostAddress, const char* szHostName)	int THostInfoManager::GetHostAddress(TSocketAddress& sAddress, const char* szHostName)
getprotobyname()	::getprotobyname() only
getprotobynumber()	::getprotobynumber() only

getservbyname()	int TServiceManager::GetService(ServiceEntry*& sEntry, char* szName, const char* szProtocol)
	int TServiceManager::GetServiceAsync(HANDLE& hService, char* szName, const char* szProtocol)
	int TServiceManager::GetServiceAsync(HWND hwndNotify, UINT nMessage, HANDLE& hService, char* szName, const char* szProtocol)
getservbyport()	int TServiceManager::GetService(ServiceEntry*& sEntry, int nPort, const char* szProtocol)
	int TServiceManager::GetServiceAsync(HANDLE& hService, int nPort, const char* szProtocol)
	int TServiceManager::GetServiceAsync(HWND hwndNotify, UINT nMessage, HANDLE& hService, int nPort, const char* szProtocol)
WSAAsyncGetHostByAddr()	int THostInfoManager::GetHostInfoAsync(HANDLE& hTheHostRequest, TSocketAddress& sAddress)
	int THostInfoManager::GetHostInfoAsync(HWND hwndNotify, UINT nMessage, HANDLE& hTheHostRequest, TSocketAddress& sAddress)
WSAAsyncGetHostByName()	int THostInfoManager::GetHostInfoAsync(HANDLE& hTheHostRequest, char* szName)
	int THostInfoManager::GetHostInfoAsync(HWND hwndNotify, UINT nMessage, HANDLE& hTheHostRequest, char* szName)
WSAAsyncGetProtoByName()	::WSAGetProtoByName() only
WSAAsyncGetProtoByNumber()	::WSAGetProtoByNumber() only
WSAAsyncGetServByName()	int TServiceManager::GetServiceAsync(HANDLE& hService, char* szName, const char* szProtocol)
	int TServiceManager::GetServiceAsync(HWND hwndNotify, UINT nMessage, HANDLE& hService, char* szName, const char* szProtocol)
WSAAsyncGetServByPort()	int TServiceManager::GetServiceAsync(HANDLE& hService, int nPort, const char* szProtocol)
	int TServiceManager::GetServiceAsync(HWND hwndNotify, UINT nMessage, HANDLE& hService, int nPort, const char* szProtocol)
WSAAsyncSelect()	void TSocket::StartAcceptNotification()
	void TSocket::StartRegularNotification()
	void TSocket::CancelNotification()
WSACancelAsyncRequest()	int TServiceManager::CancelService(HANDLE hService)
	int THostInfoManager::CancelHostRequest(HANDLE hTheHostRequest)
WSACancelBlockingCall()	::WSACancelBlockingCall() only
WSACleanup()	::TSocketManager::ShutDown()
WSAGetLastError()	int TSocket::GetLastError()
	int THostInfoManager::GetLastError()
	int TServiceManager::GetLastError()

	int TSocketManager::GetLastError ()
WSAIsBlocking()	::WSAIsBlocking() only
WSASetBlockingHook()	::WSASetBlockingHook() only
WSASetLastError()	::WSASetLastError() only
WSAStartup()	::TSocketManager::Startup()
WSAUnhookBlockingHook()	::WSAUnhookBlockingHook() only

Capítulo 4
Ruteador Server

4.1 Objetivo

El objetivo de este trabajo de investigación fue desarrollar una herramienta para que cualquier alumno que estudie redes tenga una mejor perspectiva de la arquitectura, desempeño y funcionalidad de la Internet (red de redes virtual de computadoras) y la Suite de protocolos TCP/IP, mediante un programa que emule el funcionamiento de un ruteador IP.

El programa fue diseñado de manera que tuviera una ejecución libre e independiente de la configuración de red en la computadora; es decir, que al ejecutarse no modificara ningún dato correspondiente a las propiedades IP de la máquina (dirección IP, máscara de subred, etc.), tan solo los consultara; ya que las direcciones de las redes a emular, el programa las considera como virtuales, por la implementación del protocolo IP en el nivel de aplicación con ayuda del protocolo UDP de la capa de transporte. En otras palabras, se creó un pseudo IP que permitiera crear una red de redes virtual en una pequeña red local para poder observar de una manera real los procesos de ruteo y tareas de los demás protocolos relacionadas con éste.

4.2 Justificación

Como se mencionó anteriormente, los protocolos TCP y el IP proporcionan reglas para la comunicación, así como los detalles referentes a los formatos de los mensajes; proveen una interconexión en el nivel de red a través de los ruteadores, describen cómo responde una computadora cuando llega un mensaje y especifican de qué manera una computadora maneja un error u otras condiciones anormales. Además de permitir la comunicación por computadora de forma independiente de cualquier hardware de red de cualquier marca.

A primera vista los conceptos y la arquitectura, incluyendo la estratificación por capas, puede parecer fácil. A primera instancia, ya que en la realidad es un poco más complicado de lo que parece. En los primeros capítulos se describen a grandes rasgos las características del TCP/IP, el algoritmo y los protocolos de ruteo. La abstracción de estos conceptos es cualidad que sobresale. Si se quiere tener una mejor comprensión en su estudio, el recurso visual y práctico es determinante. Poder experimentar sobre dichos conceptos sería un poco complicado, a menos que se tuviera una red de redes. Aún así sería difícil, ya que para poder hacer pruebas tendríamos que tener toda la red a nuestro cargo. Por otra parte la rapidez con que suceden todos los eventos es demasiada. Definitivamente tendríamos muchos problemas para poder observarlos.

Surge entonces la idea de realizar todas las pruebas que se desee, observar el desempeño, simular situaciones de error, etc., en una pequeña red de computadoras (5 o 10 por ejemplo). Cada computadora podría asumir el papel de host de alguna red o ruteador y con ello crear una red de redes virtual como la Internet.

El primer paso fue emular el proceso de ruteo: implementando el algoritmo y un protocolo interno de ruteo (RIP). Además, el Protocolo Internet (IP), el ARP y el ICMP. Con ello podremos observar e interactuar con los procedimientos y eventos que ocurren en una red de redes dentro de un sistema autónomo.

El sistema ha sido hecho pensando en un crecimiento, por ello, más adelante podrían implementarse mas funciones; como un protocolo más de ruteo interno o externo, o bien, enfatizar algún procedimiento en especial.

4.3 Metodología

Para poder llevar a cabo la emulación de Ruteador IP se utilizó programación orientada a objetos, lenguaje C++, compilador Borland ver. 5.0.

El enfoque inicial en el cual se sustentó el desarrollo del programa, fue el mismo que el de la Internet: ocultar los detalles subyacentes de red; en este caso, representado por la dirección IP de la computadora donde se ejecutaría Ruteador Server. Todo ello para poder crear una red de redes virtual dentro de una pequeña red; además, que el programa pudiera ejecutarse en cualquier máquina no importando su dirección IP, sin modificar ningún dato de configuración de la computadora para no causar problemas dentro de la red experimental.

Todo esto dio como resultado un nuevo modelo conceptual basado en la abstracción misma del modelo conceptual del software TCP/IP de la Internet (ver la figura 4.1). Donde las capas de Aplicación y Transporte representan ahora la capa de Internet y la capa de Internet e Interfaz de Red son la capa de Interfaz de Red. Este modelo nos permite utilizar cualquier dirección IP en la computadora y crear una diversidad de redes para poder ver de manera real el funcionamiento de un ruteador y el papel que juega dentro de una red de redes como la Internet. Además de algunas características del protocolo IP, ICMP y ARP; ya que para el buen funcionamiento se implantaron todos estos protocolos; obteniendo una capa de Internet e Interfaz de red virtual.

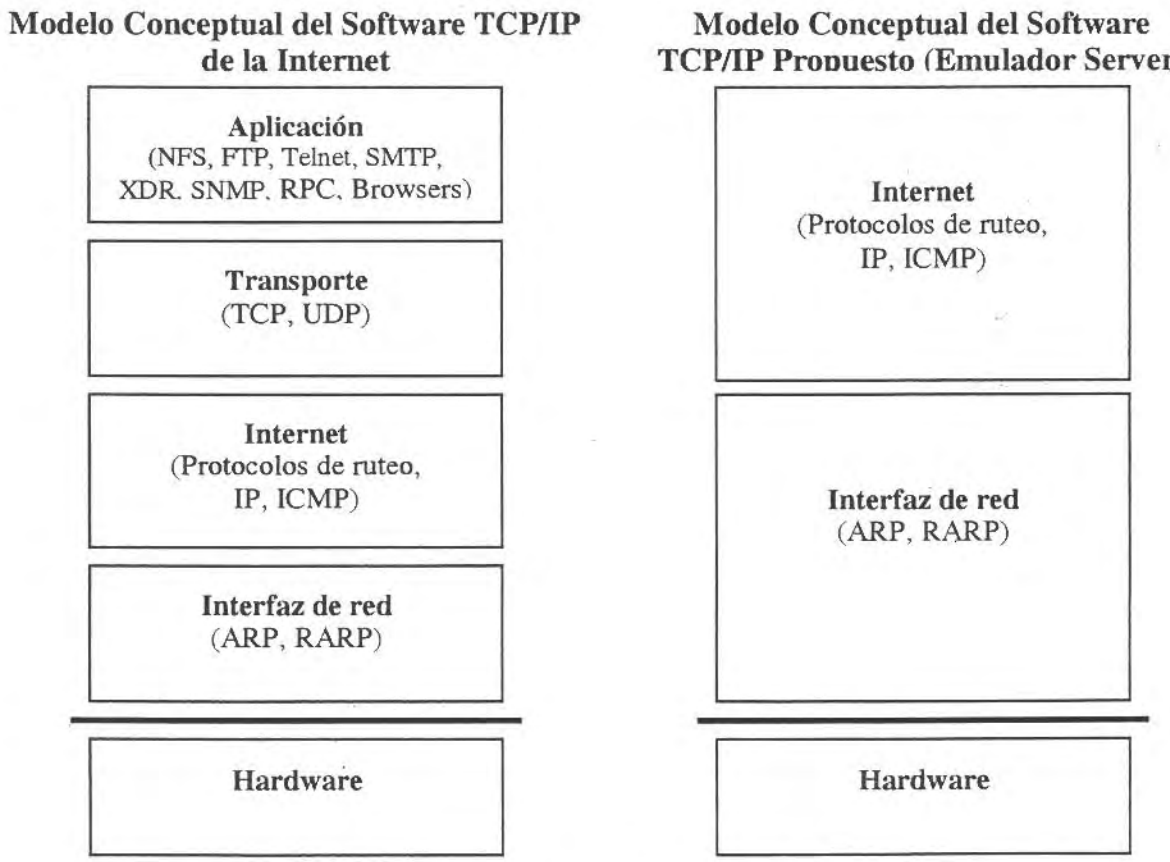


Figura 4.1 Modelo conceptual del Software TCP/IP y el propuesto.

Para poder implementar el protocolo IP se creó un pseudo datagrama IP dentro del datagrama UDP. Se eligió UDP por la característica en común que tienen con el protocolo IP: sistema de entrega sin conexión y con el mejor esfuerzo. Pseudo IP para poder hacer a un lado la dirección IP de la máquina y poder asignarle cualquier dirección IP (dirección IP virtual) y crear toda una red de redes virtual dentro de una LAN experimental. En la figura 4.2 se observa dicho encapsulado.

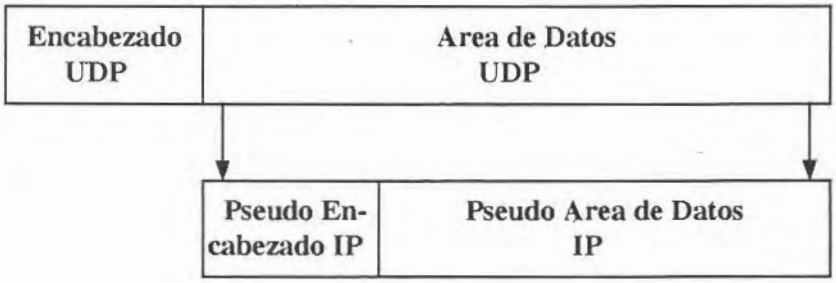


Figura 4.2 Pseudo Datagrama IP encapsulado en un datagrama IP para su transmisión en una red de redes virtual.

De esta manera la transmisión de los datagramas IP virtuales se hace independiente y libre de la dirección IP real de la máquina; ya que la dirección virtual de los anfitriones de la red de redes a emular va contenida en el pseudo encabezado IP.

También se establece el formato para poder encapsular los pseudo datagramas ICMP (ver figura 4.3).

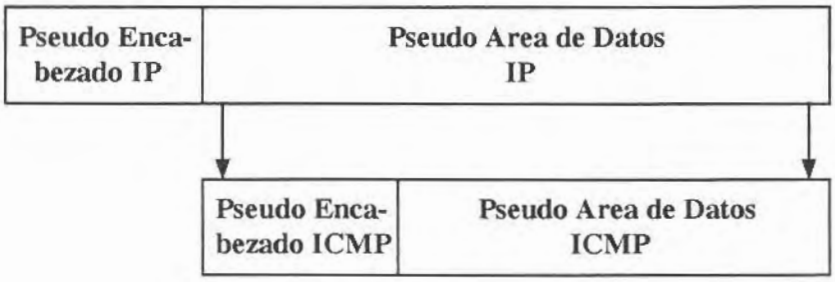


Figura 4.3 Pseudo Datagrama ICMP encapsulado en el pseudo datagrama IP.

De la misma forma se establece el encapsulado de datagramas ARP, esto en la figura 4.4. Para los mensajes RIP, el diseño consistió en encapsular dentro del pseudo datagrama IP, un pseudo datagrama UDP y dentro de él, el pseudo mensaje RIP (ver figura 4.5).

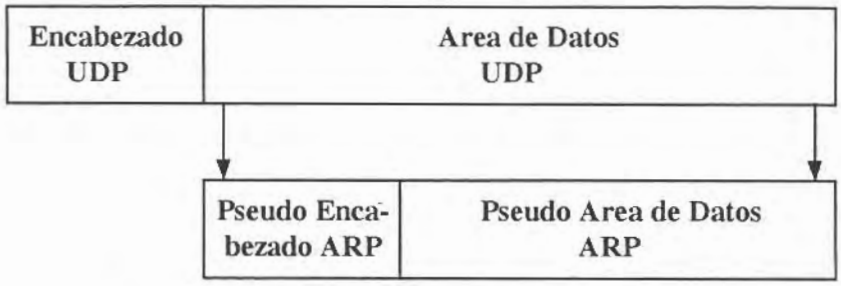


Figura 4.4 Pseudo Datagrama ARP encapsulado en el datagrama UDP.

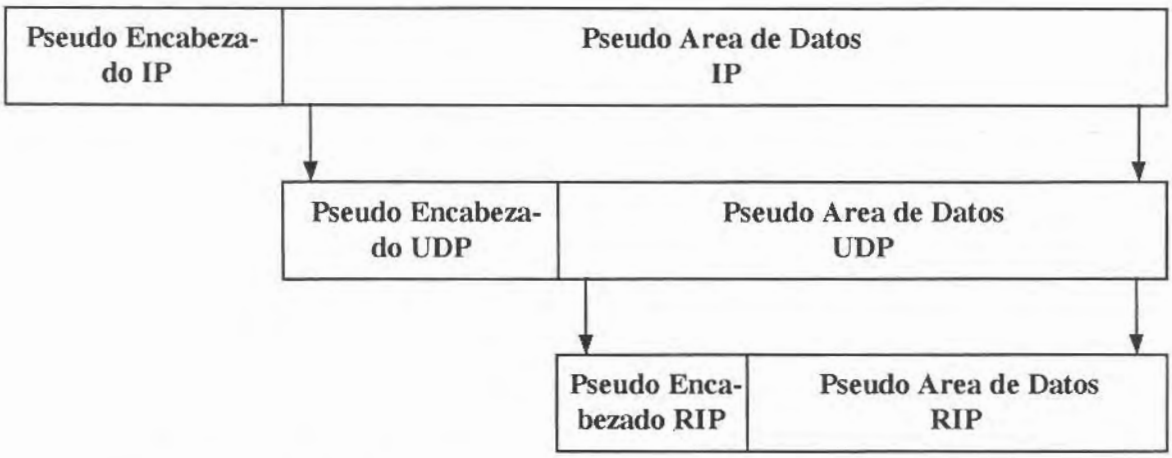


Figura 4.5 Pseudo Datagrama RIP encapsulado en el pseudo datagrama UDP, dentro del pseudo datagrama IP.

La implementación de todos los protocolos estuvo sustentada en los RFC'S correspondientes.

Por otra parte el modelado de las clases y los objetos se basó en el esquema de la figura 4.6. Hubo la necesidad de definir una clase que se encargara de controlar los eventos en el intercambio de datos entre las distintas clases que definen las propiedades IP, tanto del host y ruteador con las clases de emulación de ambos (monitoreo).

La clase Tabla de Ruteo inicial define el objeto encargado en la configuración de una tabla de Ruteo inicial y Ruteador Predeterminado. La clase Inicialización Socket define las funciones para la inicialización del socket UDP. Esta clase hereda a la clase Monitor Host y Monitor Ruteador.

Los eventos están predeterminados a los tipos de mensajes que recibe el anfitrión o ruteador al momento de la ejecución:

- Mensajes Request, Response (Protocolo RIP).
- Mensajes de petición y respuesta de dirección IP (Protocolo ARP).

- Mensajes de redireccionamiento, tiempo excedido, solicitud y respuesta de máscara, ruteo inaccesible, red o destino desconocido, etc. (Protocolo ICMP).

También en el manejo de eventos por temporizadores, estos definidos en los protocolos RIP y ARP.

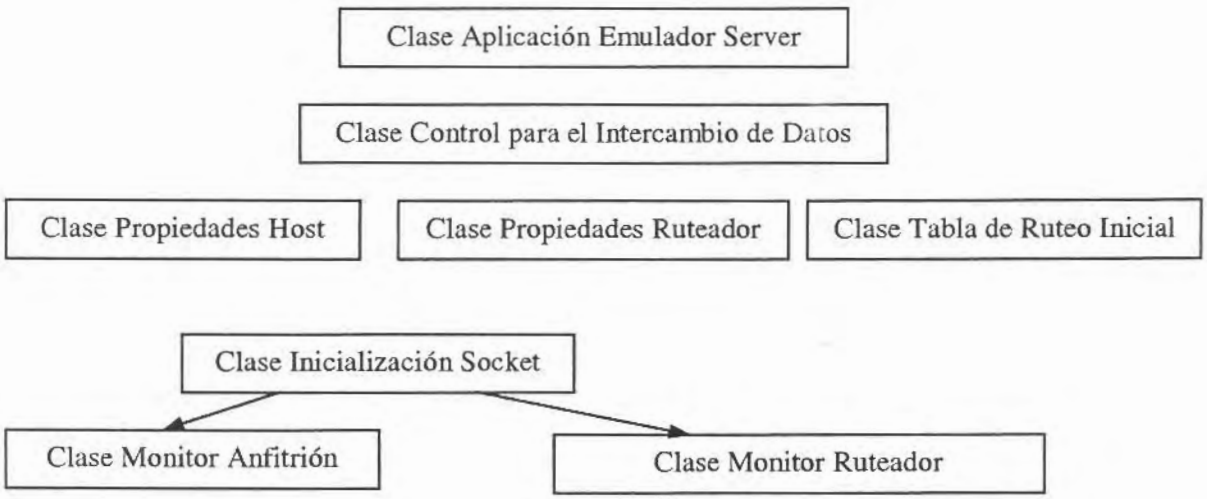


Figura 4.6 Esquema básico de las clases definidas para la aplicación Emulador Server.

En la figura 4.7 se puede observar el intercambio de información que se sucede entre los distintos módulos definidos en la figura 4.6.

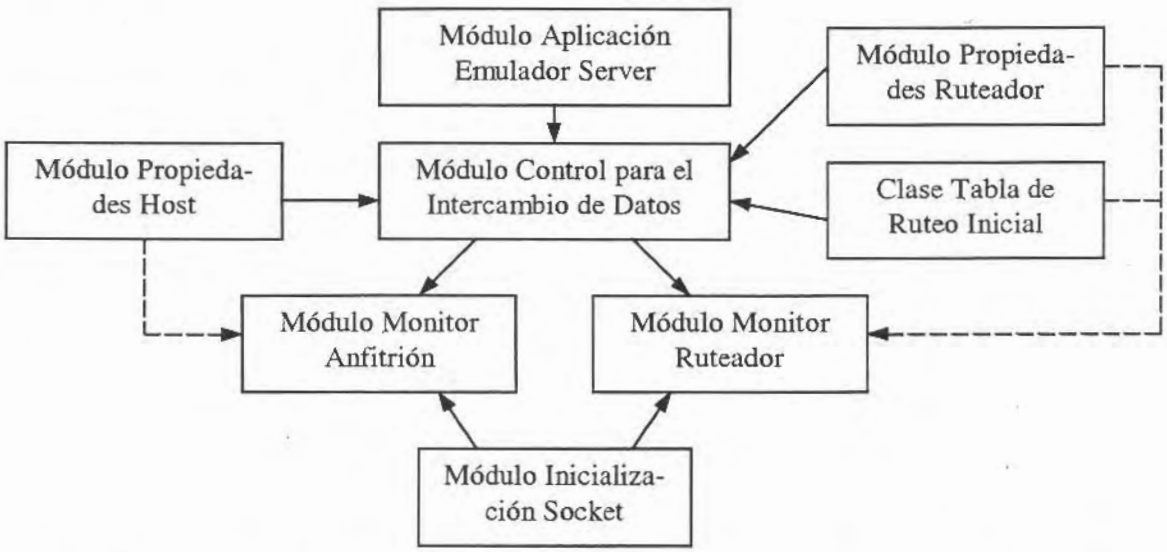


Figura 4.7 Intercambio de datos entre los módulos del esquema básico de diseño.

4.4 Emulación

Es importante destacar que los proceso emulados se suscitan en la Internet internamente, en la capa IP:

- algoritmo de ruteo en redes y subredes (manejo de máscaras)
- intercambio de información de tablas de ruteo con base el protocolo RIP
- manejo de errores (protocolo ICMP) relacionados con el ruteo
- obtención de direcciones máquina de la capa de enlace (ARP)

Requisitos del sistema:

- Una red local (LAN) Ethernet.
- Computadoras 486 o mayor
- 8 Mb. RAM de preferencia 16 Mb.
- 4 Mb. espacio en disco duro.
- Resolución en video de 800 x 600 pixeles.
- Sistema Windows 95
- Winsock ver 1.1 o mayor.
- Software TCP/IP con su dirección IP definida en cada máquina participante.

Instalación de Emulador Server, sólo se tiene que ejecutar el archivo por lotes: **Instalar** del disco 1 de instalación. Si se desea se puede compartir la carpeta donde residen los programas para ejecutarlo desde otras máquinas.

Operación: El proceso de emulación y manejo del programa es muy sencillo. Cada máquina en la red experimental podrá asumir el papel de host o ruteador. Para ello se deben configurar los datos (dirección IP, máscara de subred, etc.); esto se lleva a cabo con el comando del menú principal **Anfitrión** y la opción **Propiedades**, si se desea que la máquina emule un host y el comando **Ruteador**, opción **Configuración** si la máquina asume el papel de ruteador.

Siguiendo la filosofía de ocultar el hardware subyacente de red y teniendo en cuenta el concepto de Internet, el programa asume que la dirección IP real de la computadora es su dirección máquina y que la nueva dirección que se asignará es la dirección IP, como se explicó en secciones anteriores. La dirección IP real se oculta y sólo se trabaja con la dirección IP que se declaró arbitrariamente durante la emulación. Para aclarar más, supóngase que la red experimental donde se llevará a cabo la emulación, tiene la dirección 200.200.200.0. Tres máquinas con direcciones 200.200.200.20, 200.200.200.21, 200.200.200.22, como se muestra en la figura 4.8 y se quiere emular la red de redes mostrada en la figura 4.9.

Ahora, supóngase que la máquina M1 será el ruteador R1 de la red de redes a emular, es decir, que M1 tendrá asignada las direcciones 1.0.0.17 y 3.0.0.30. Para poder configurar el ruteador se ejecuta en M1 el programa **Ruteador Server** y con el comando **Ruteador**, **Configuración** del menu principal se asignan las direcciones 1.0.0.17 y 3.0.0.30 con las máscaras de red y las métricas. Al ejecutar dicho comando aparecerá la ventana **Propiedades IP** (ver fig. 4.10).

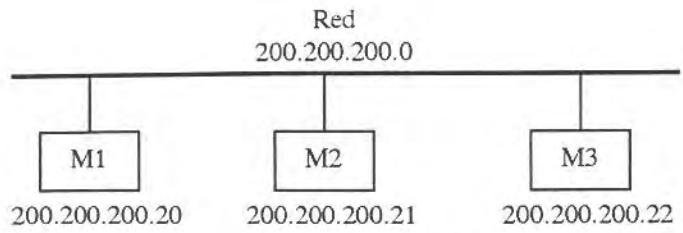


Figura 4.8. Red local donde se llevará a cabo la emulación.

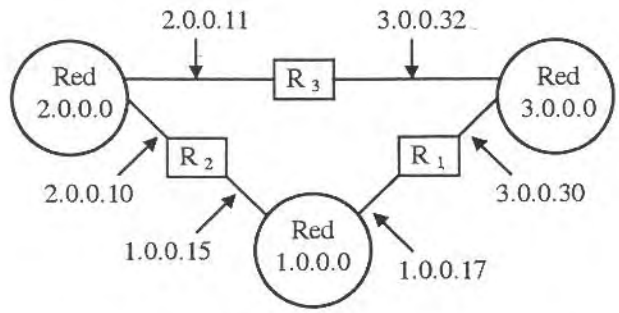


Figura 4.9. Red de redes a emular.

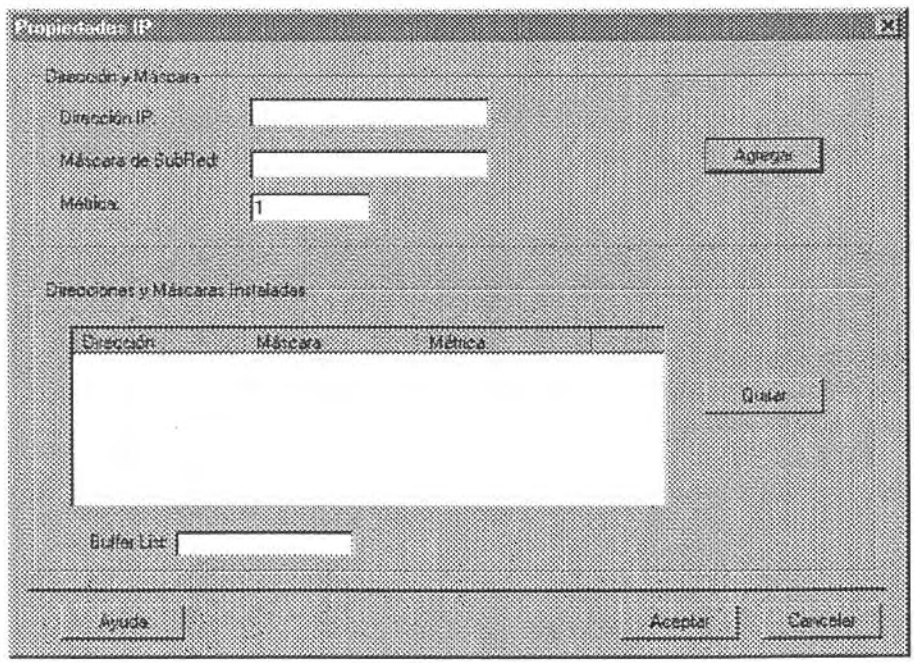


Figura 4.10. Ventana de definición de las propiedades IP del Ruteador a emular.

Se insertan los datos. Primeramente la dirección 1.0.0.17 con máscara de subred 255.255.0.0 y métrica 3, se presiona el botón **Agregar** para instarlar la dirección y los demás datos. Si hay alguna equivocación en algún parámetro se puede quitar dicha dirección de la lista de **Direcciones y Máscaras Instaladas** seleccionando la dirección con el mouse y presionando el botón **Quitar** e insertar nuevamente.

Después se agrega la dirección 3.0.0.30 con máscara de subred 255.255.255.0 y métrica 4 de la misma manera expuesta. Finalmente, los datos quedarían como se muestran en la figura 4.11.

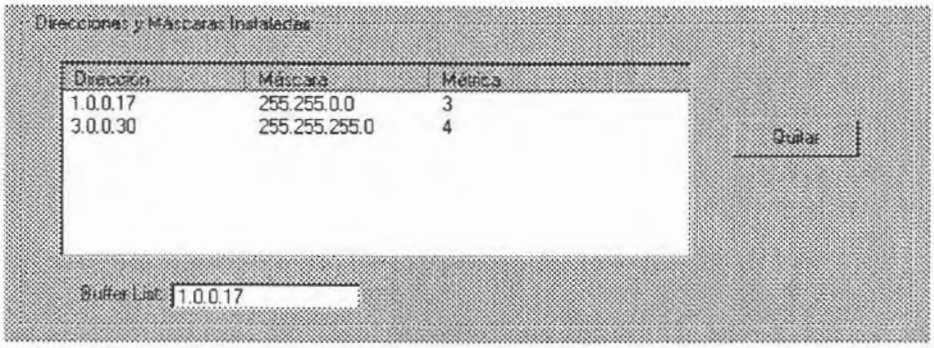


Figura 4.11. Definición de las propiedades de R1 en M1.

El siguiente paso es activar el ruteador con el comando **Ruteador, Monitor**. Aparecerá la ventana **Monitor de Ruteo** (ver fig. 4.12) donde se podrá observar y ejecutar las funciones de ruteo expuestas para M1 emulando a R1.

Con ello M1 emulará además de R1, la red 1.0.0.0 y 3.0.0.0 ya que el ruteador es parte de ambas redes como un host. Obsérvese en la tabla de ruteo (figura 4.15) que la dirección IP real (200.200.200.20) quedará oculta ya que solo se trabajará con las nuevas direcciones asignadas. Este es punto clave en el funcionamiento de la Internet, red de redes virtual: se hacen a un lado los detalles subyacentes de red y el host se conoce solo por su dirección IP, en este caso la dirección IP real. En la figura 4.13 se puede ver.

M1 ahora se conocerá como R1 en nuestra emulación. Además, la asignación de direcciones IP virtuales no afecta en nada la dirección real. Todo ello gracias a la implementación del protocolo IP en la capa de Aplicación (ver figura 4.1).

Siguiendo la emulación, cuando un ruteador es activado debe mandar mensajes *request* iniciales por difusión a las redes a las que pertenece, de acuerdo con el RIP, para recibir de los ruteadores vecinos sus tablas de ruteo para actualizar la suya. En el **Registro de Eventos** se puede observar (ver figura. 4.14). Como R1 está solo, no recibe ninguna respuesta.

Nótese en la figura 4.15 que aunque la tabla de ruteo está dividida en dos listas en realidad es una sola. El motivo de la división es para lograr una mayor comprensión. La

primera lista muestra las redes locales, es decir, las redes a las que está conectado R1 y en la otra los destinos y rutas a los que tiene acceso. Hasta ahora ninguno.

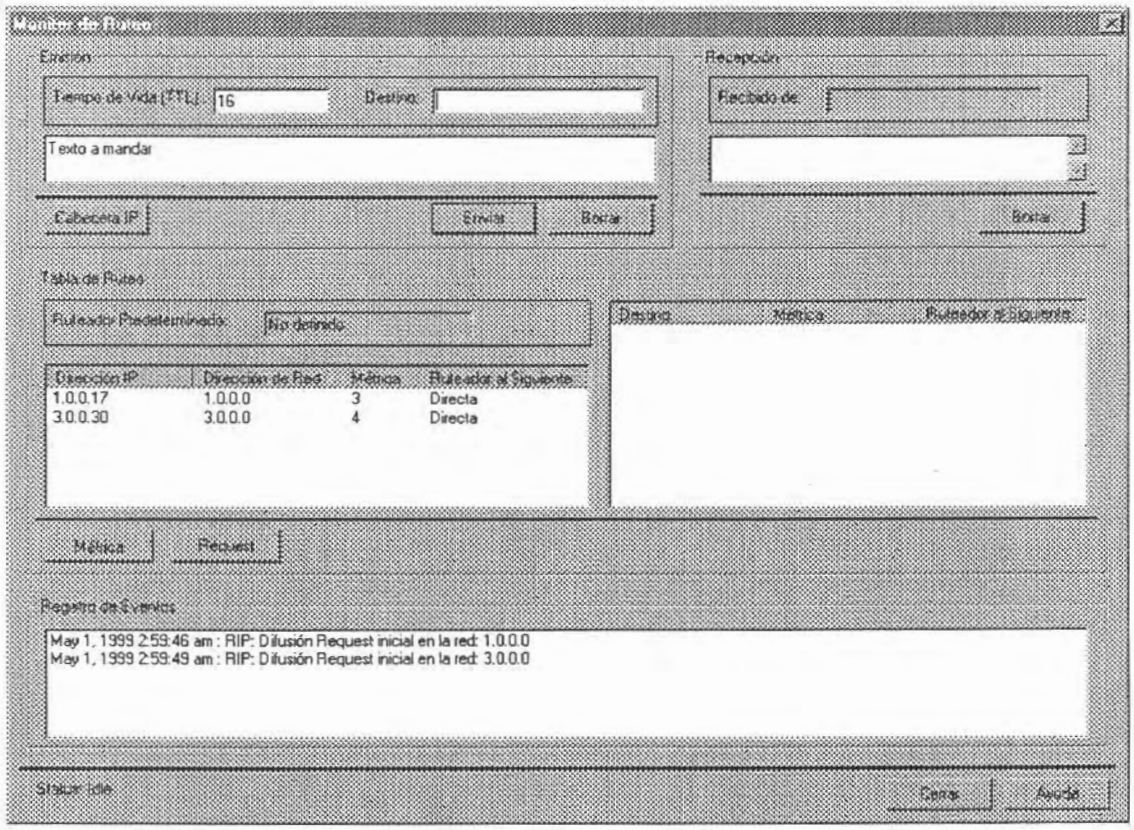


Figura 4.12. Pantalla de monitoreo del Ruteador a simular.

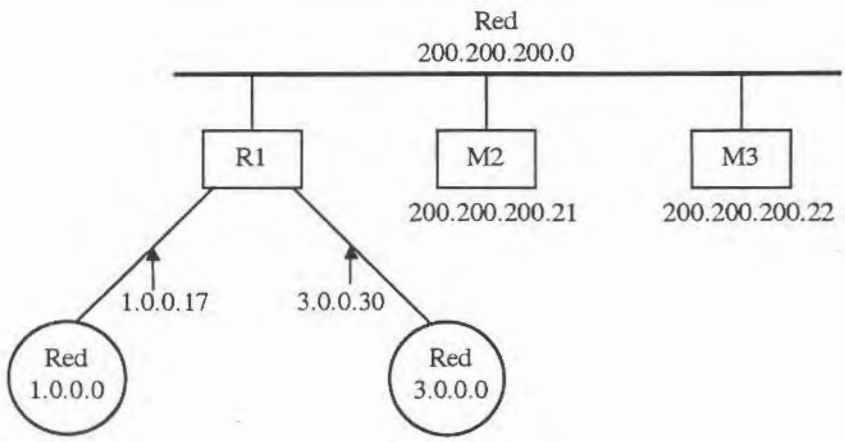


Figura 4.13. Ruteador R1 activado y con ello las redes 1.0.0.0 y 3.0.0.0.

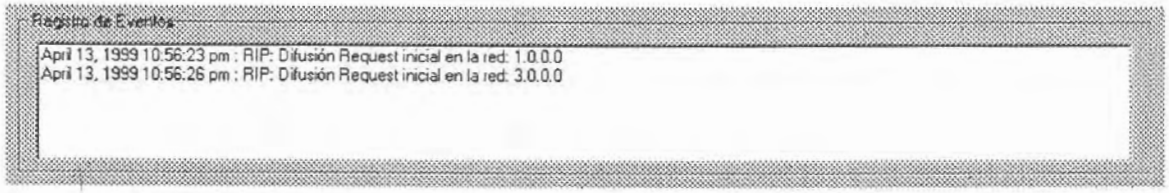


Figura 4.14. Registro de Eventos durante la emulación del ruteador

Figura 4.15. Tabla de Ruteo en R1. Ambas listas forman la tabla de ruteo.

El siguiente paso es encender el ruteador R2 en M2. Los datos quedarían como se observa en la figura 4.16. Aquí se ha hecho un cambio en la métrica para la red 1.0.0.0 con respecto al primer ruteador, se ha asignado el valor de 5. Más adelante se verá porqué.

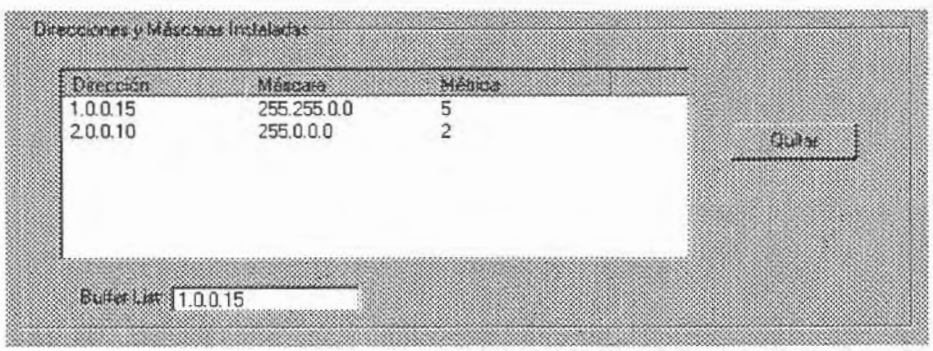


Figura 4.16. Propiedades de R2 en M2.

Cuando se activa R2 envía mensajes *request* iniciales e inmediatamente R1 le responde con su tabla de ruteo (ver figuras 4.17 y 4.18).

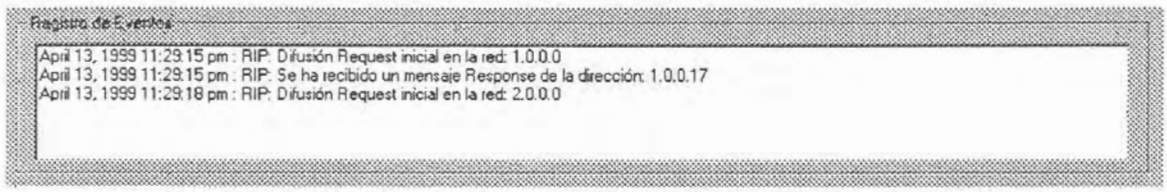


Figura 4.17. R2 envía un request inicial a la red 1.0.0.0
e inmediatamente R1 (1.0.017) le responde.

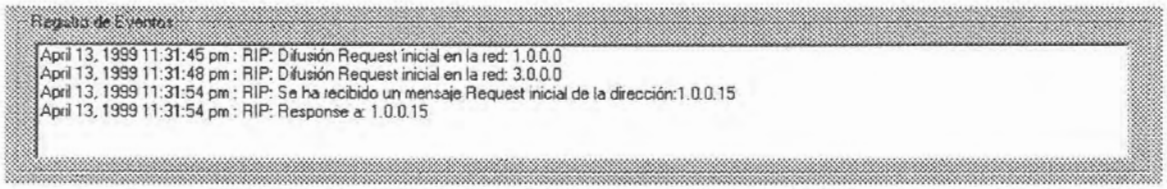


Figura 4.18. R1 recibe de R2 (1.0.0.15) el request inicial
e inmediatamente le responde.

La tabla de R2 quedaría como en la figura 4.19.

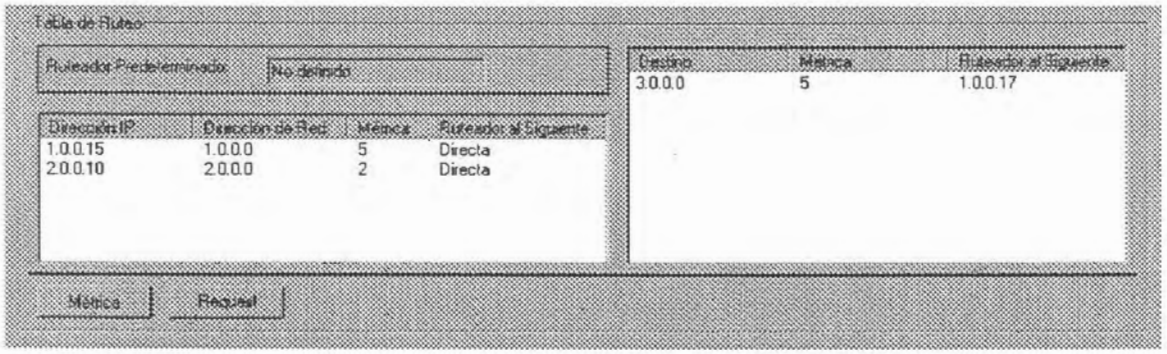


Figura 4.19. Tabla de Ruteo en R2.

En este momento R2 inicia el proceso *Trigger-Update*, pero no manda ningún mensaje, ya que sólo tiene de ruteador vecino a R1 y por la heurística *split-horizont* no puede anunciarle la nueva ruta que tiene; ya que por R1 llega a ella. Es hasta el mensaje *response update* que tiene que mandar R2 cada 30 segundos, (ver fig. 4.20) cuando R1 actualiza su tabla de ruteo con el destino 2.0.0.0 (ver figuras 4.21 y 4.22). Al hacerlo inicializa el proceso *Trigger-Update* y sucede lo mismo como pasó con R2.

Ahora se activará R3. Para ello se agrega primeramente la dirección 2.0.0.11 con métrica de 3, esto para que a la primera red que mande la difusión sea la 2.0.0.0. Los datos se verían como en la figura 4.23.

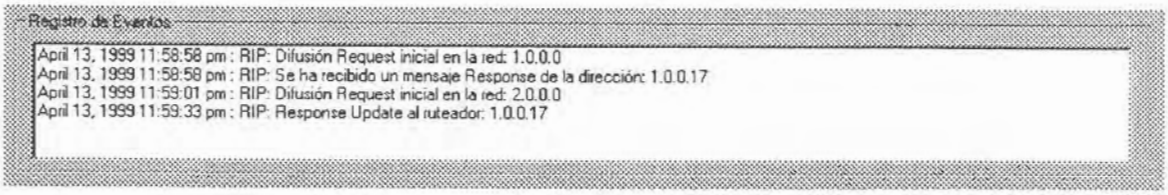


Figura 4.20. Después de 30 segundos R2 envía un Response Update a R1 (1.0.017).

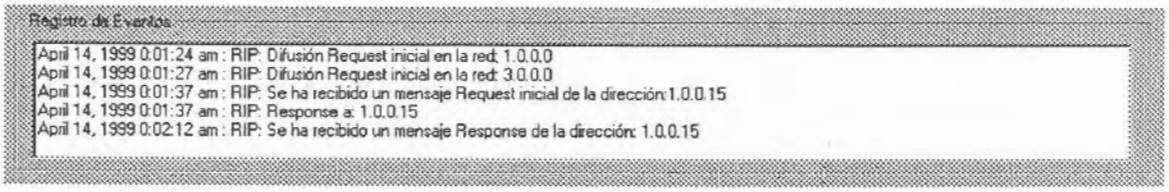


Figura 4.21. R1 recibe un Response de R2 (1.0.0.15).

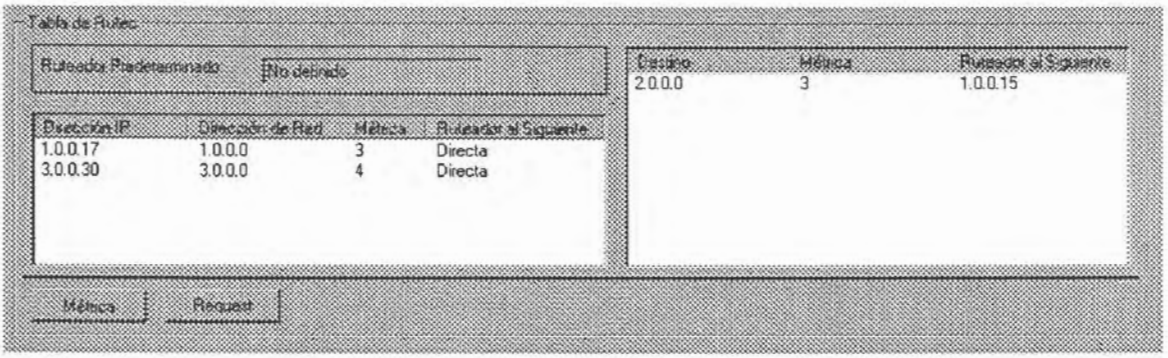


Figura 4.22. Tabla de Ruteo en R1 después de la actualización recibida por R2.

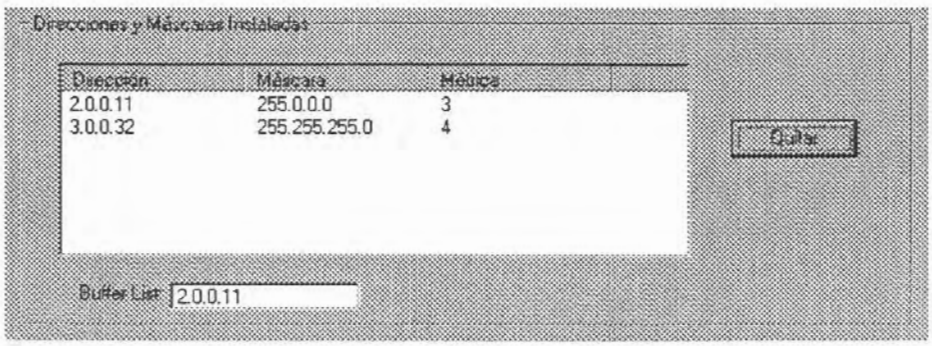


Figura 4.23. Propiedades de R3 en M3.

Al mandar las difusiones R3, el primer ruteador que le responde es R2. La tabla quedaría como se observa en la figura 4.24.

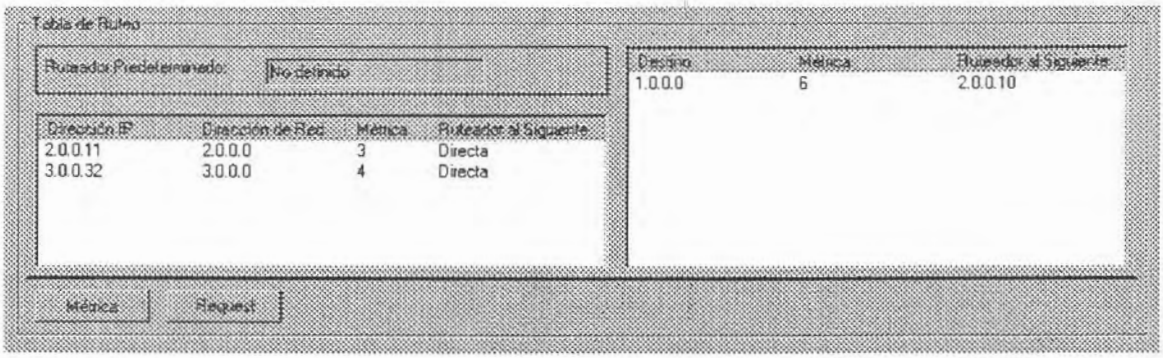


Figura 4.24. Tabla de Ruteo en R3 cuando recibe notificación (Response) de R2.

Cuando R3 recibe la notificación de R1, la ruta hacia el destino 1.0.0.0 cambia; ya que la métrica que ofrece R1 es menor, 4. Véase la figura 4.25.

Figura 4.25. Tabla de Ruteo en R3 después de recibir notificación (Response) de R1.

La red de redes propuesta emulada en nuestra red experimental se vería como en la figura 4.26.

Si se apagara R1, esto se hace cerrando la caja de diálogo **Monitor de Ruteo**, R2 y R3 seguirían mandando sus mensajes de actualización a R1; ya que ellos no saben que R1 está apagado (ver fig. 4.27). Esta es una desventaja del RIP: cuando un ruteador ha dejado de funcionar completamente los ruteadores vecinos no pueden conocer este hecho.

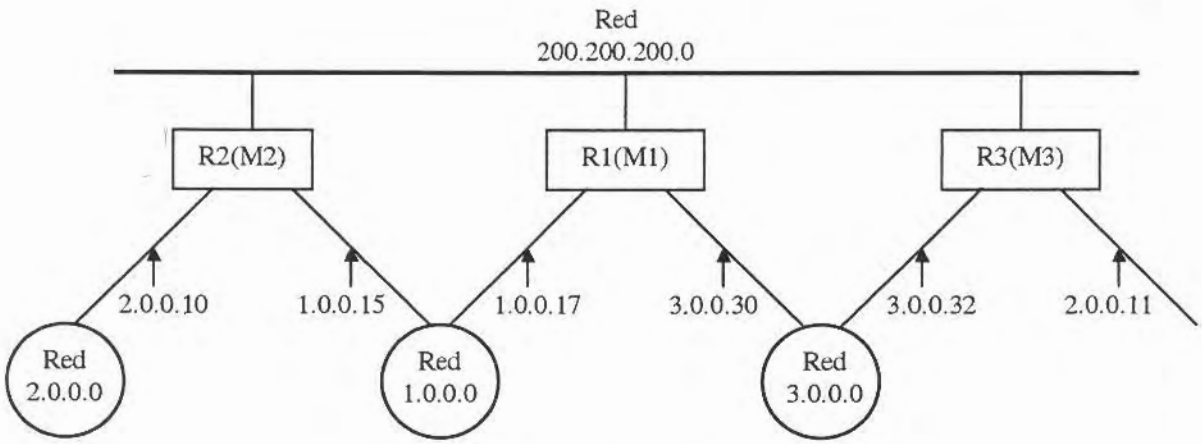
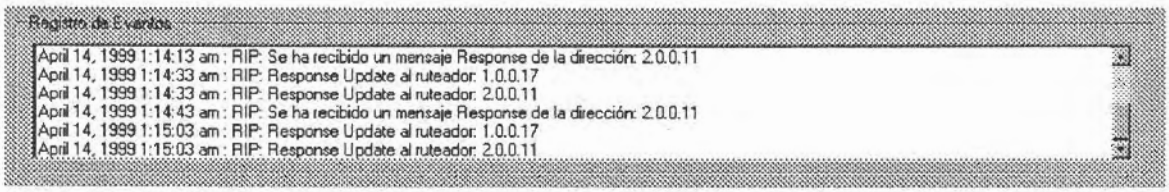
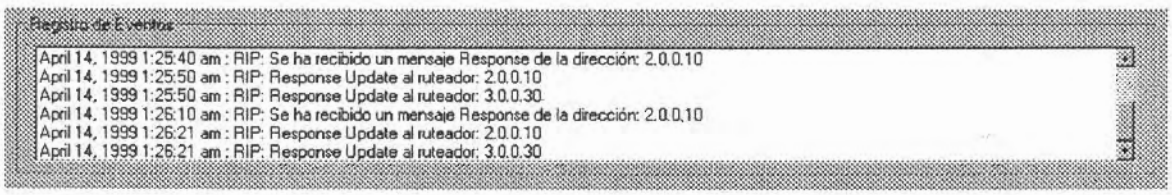


Figura 4.26. Emulación de la red de redes ejemplo en la red experimental.



(a)



(b)

Figura 4.27. R2 (a) y R3 (b) envían mensajes Response Update a R1 (1.0.0.17 y 3.0.0.30) aunque éste ya no esté activo.

Pasados 90 segundos y al no recibir actualización R2 de R1 para el destino 3.0.0.0 y al recibir R2 un mensaje *response update* de R3, cambiará la ruta hacia el destino 3.0.0.0 por R3 (ver fig. 4.28), esto ocurre por la heurística implantada para métricas iguales con rutas distintas: si ha pasado más de la mitad del tiempo permitido para una actualización de la ruta actual, y se recibe una notificación de alguna ruta hacia tal destino se cambia. Esto para no esperar a que el *timeout* termine.

Pasados 180 segundos la métrica hacia el destino 1.0.0.0 en R3 cambia a infinito e inicia el proceso de borrado (ver fig. 4.29).

Figura 4.28. Tabla de Ruteo en R2 después de recibir un mensaje de actualización de R2 para la red 3.0.0.0.

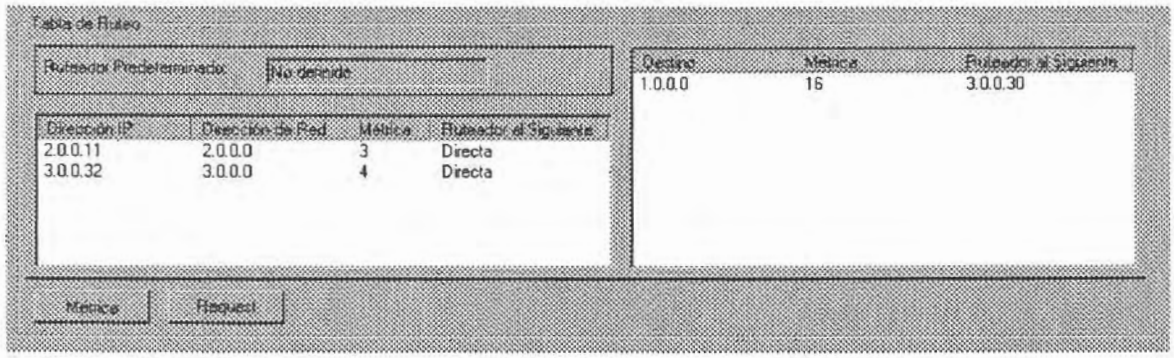


Figura 4.29. La red destino 1.0.0.0 toma la métrica de infinito (inaccesible) en la Tabla de Ruteo de R3.

Aquí la heurística mencionada anteriormente no funciona; ya que la métrica propuesta por R2 es mayor que la actual. Sin embargo al recibir notificación R3 de R2 cambia la ruta y la métrica por la de R2. En la figura 4.30 podemos observar el cambio.

Si se vuelve a conectar R1 entonces la tabla de R3 volvería a ser como lo era antes ya que R1 ofrece una métrica mejor hacia la red 1.0.0.0. En R2 no pasa nada ya que la métrica hacia la red 3.0.0.0 es igual para ambas rutas.

Ahora se activará en nuestra red de redes un anfitrión (H1) en la red 3.0.0.0. Se utilizará otra máquina, con dirección 3.0.0.5 y que su ruteador predeterminado sea R3. Se puede agregar también R1 pero va a tomar como predeterminado al primer ruteador que se agregue a la lista de **Gateways Instalados**. Todo esto se hace en el comando **Anfitrión, Propiedades** (ver fig. 4.31).

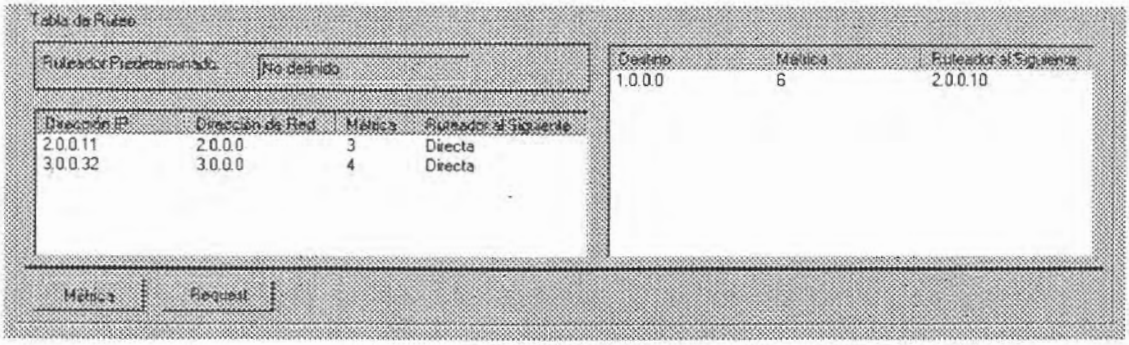
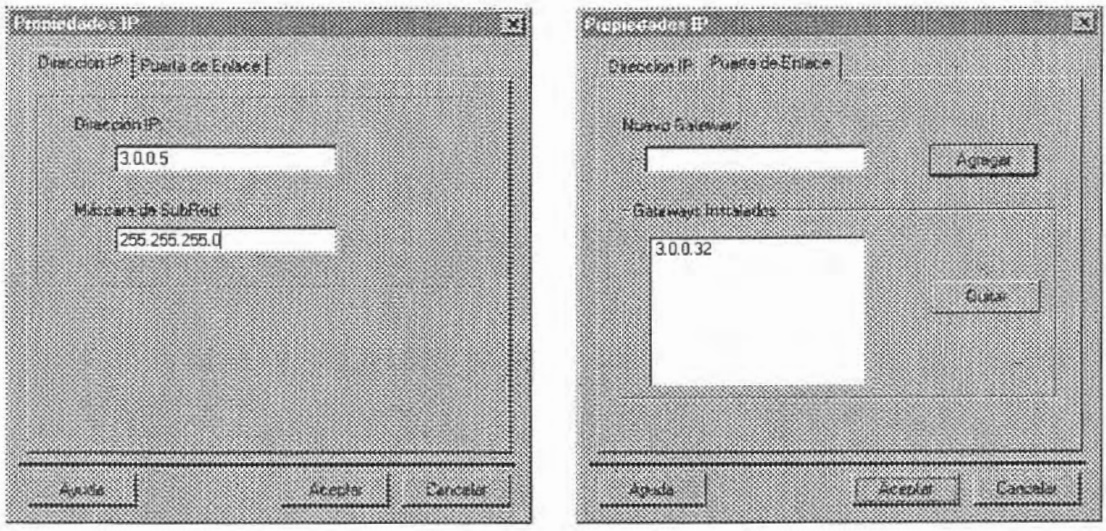


Figura 4.30. Tabla de Ruteo en R3. La red destino 1.0.0.0 actualizada por R2.



(a) (b)

Figura 4.31. Ventana para la definición de las propiedades del Anfitrión (Host) a emular. Dirección y máscara (a). Ruteador predeterminado (b)

Se activa mediante el comando **Anfitrión, Monitor**. Véase que en el cuadro **Ruteador Predeterminado** estará la dirección 3.0.0.30 (fig. 4.32). Al igual que el monitor del ruteador, el anfitrión tiene el botón **Cabecera IP**. Con él se podrá observar el formato de un datagrama IP en forma binaria. Envíese un mensaje a la dirección 1.0.0.20. Antes de mandar el mensaje presione dicho botón y obsérvese el formato (ver figura 4.33).

Aunque el host no esté encendido, esto no implica que se lleve a cabo su entrega. Recordese que el IP proporciona un sistema de entrega de paquetes sin conexión y con el mejor esfuerzo, por lo cual y junto con el ARP, van a tratar de entregar el mensaje pero al no haber acuse de recibo no se notifica la entrega. Por otra parte, el manejo de este error corresponde a capas más arriba, aplicación, y aunque nuestro programa corresponda a ésta, no olvidemos que en su ejecución corresponde a la capa IP.

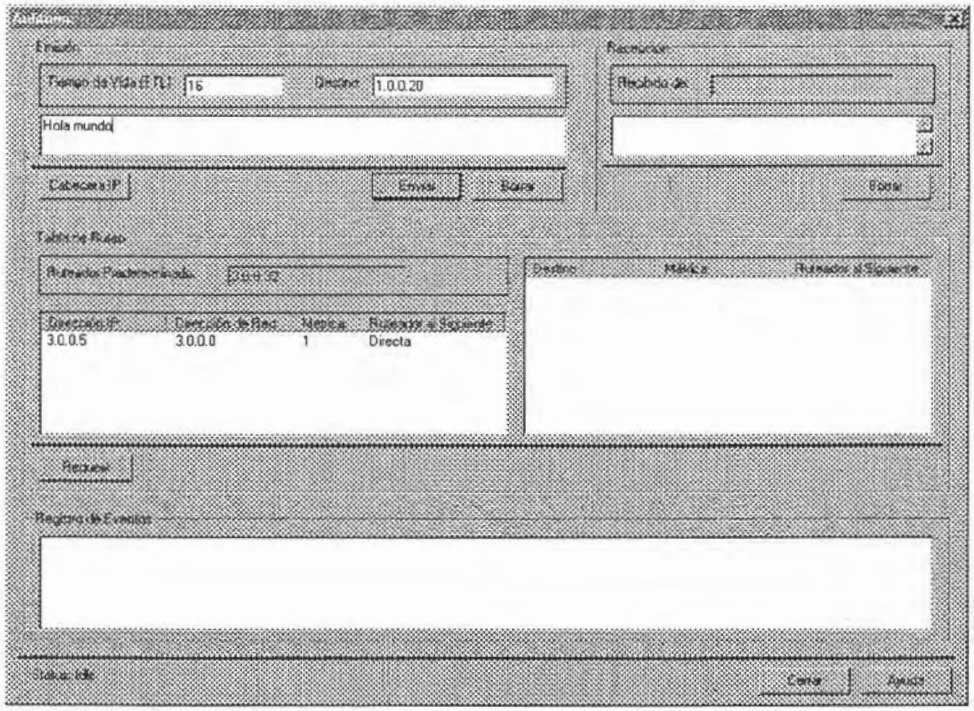


Figura 4.32. Ventana de H1 (3.0.0.5).

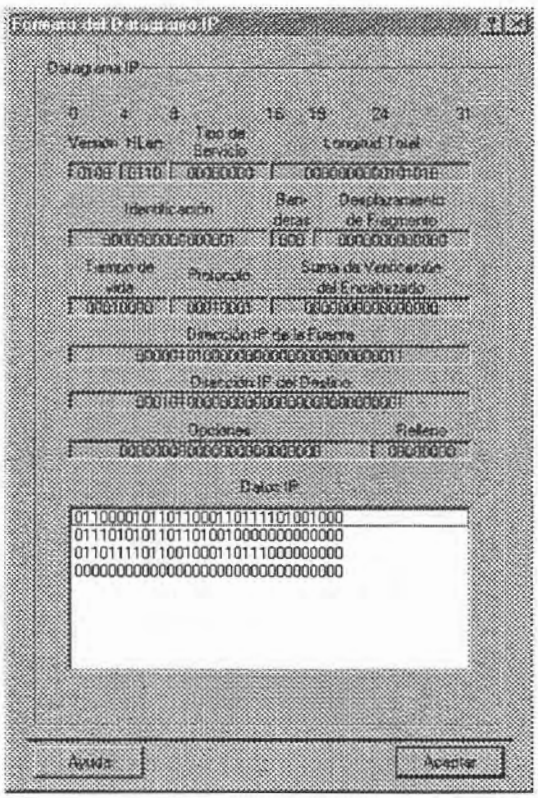
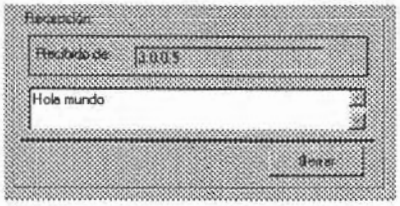
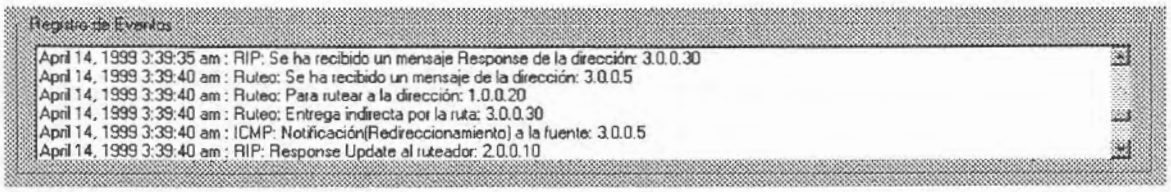


Figura 4.33. Formato del datagrama UDP/IP binario.

Al mandar el mensaje H1, presionando el botón **Enviar**, se puede observar como se lleva a cabo el ruteo. El mensaje es mandado hacia R3, el cual lo entrega a R2 para su entrega directa; pero al entregar R3 el mensaje a R2 se da cuenta que H1 debió mandarlo por R1, ya que éste lo puede entregar directamente. Esto genera por parte de R3 un mensaje ICMP de redireccionamiento para H1 (ver fig. 4.34), el cual al recibirlo genera un mensaje ICMP de solicitud de máscara hacia R1 (ver figuras 4.35 y 4.36) para poder agregarlo a su tabla de ruteo, como se muestra en la figura 4.37. La solicitud de la métrica no se da. Nosotros nos tomamos la libertad de hacer esto para informar y darnos cuenta de este parámetro, sin embargo dase cuenta que no es necesario que el host conozca la métrica, ya que el objetivo es delegar la mayoría de funciones de ruteo a los ruteadores. De esta manera los host actualizan su tabla de ruteo. Véase que en la versión 2.0 del RIP los host ya no pueden escuchar los mensajes *response* para actualizar sus tablas de ruteo, porque estos se entregan por multidifusión.



(a)



(b)

Figura 4.34. (a) Recepción de R3 del mensaje mandado por H1. (b) Se rutea a la dirección 1.0.0.20 indirectamente por la ruta 3.0.0.30 y se genera una notificación ICMP (redireccionamiento) a H1 (3.0.0.5).

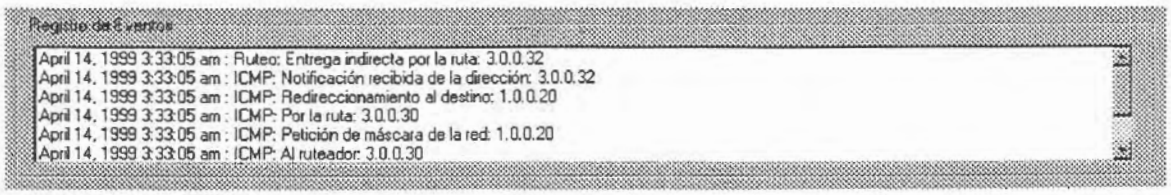
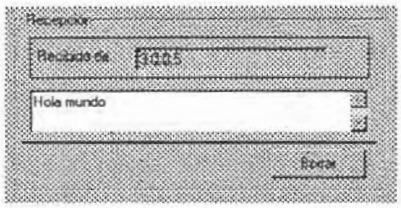
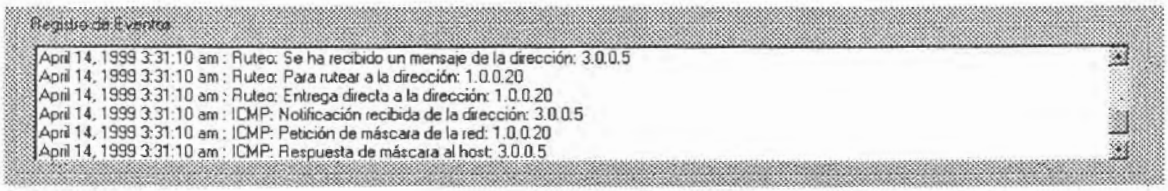


Figura 4.35. H1 recibe notificación ICMP (redireccionamiento) de R3 (3.0.0.32) y genera una petición de máscara de subred para el destino 1.0.0.20 a R1 (3.0.0.30).

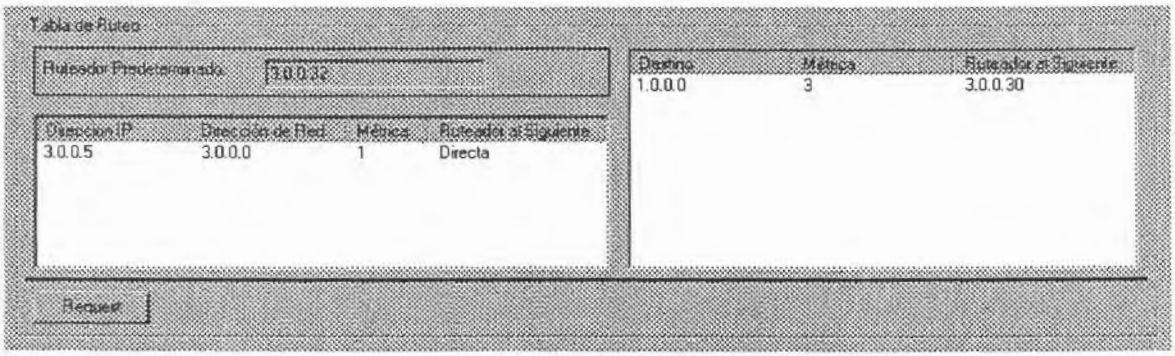


(a)

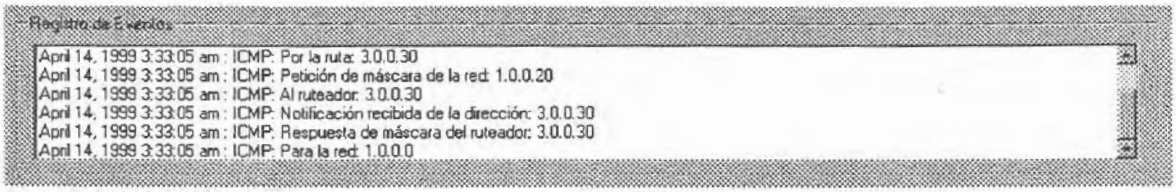


(b)

Figura 4.36. (a) Recepción de R1 (3.0.0.30) del mensaje mandado por H1. (b) Se rutea a la dirección 1.0.0.20 directamente, se recibe petición de máscara para el destino 1.0.0.20 y se genera la respuesta de máscara a la fuente H1 (3.0.0.5).



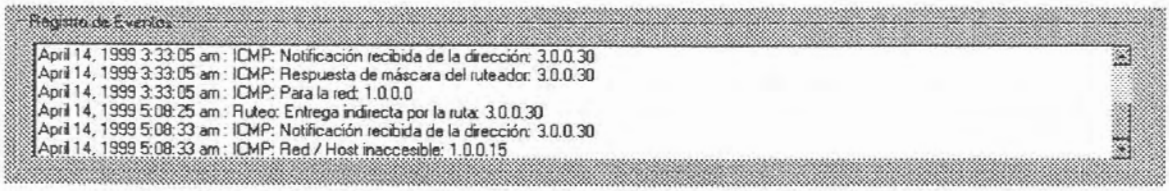
(a)



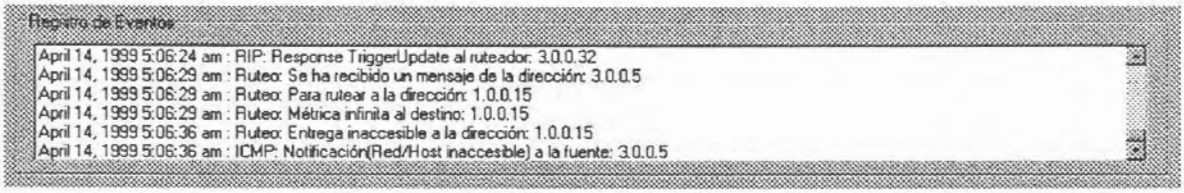
(b)

Figura 4.37. (a) Tabla de Ruteo resultante en H1. (b) Se obtiene la máscara para el destino 1.0.0.20.

Cámbiese la métrica a infinito en R1 para la red 1.0.0.0. Esto se hace eligiendo la red en la primera lista de la tabla de ruteo y presionando el botón **Métrica**. Se inicia el proceso *Trigger-Update* y si no hay algún cambio más en el intervalo de 1 a 5 segundos, R1 manda un mensaje *response* a R3. Ahora envíese un mensaje desde H1 a la dirección 1.0.0.15 (R2). Al recibir dicho mensaje R1 generará un mensaje ICMP de red inaccesible para H1, ya que su métrica es infinita (ver figura 4.38). Uno supondría que si no puede entregarlo directamente lo puede mandar hacia R3 para su entrega. Es válido, pero para RIP no lo es y no porque sea una operación no válida, sino que no es capaz de manejar rutas alternativas para posibles situaciones como éstas.



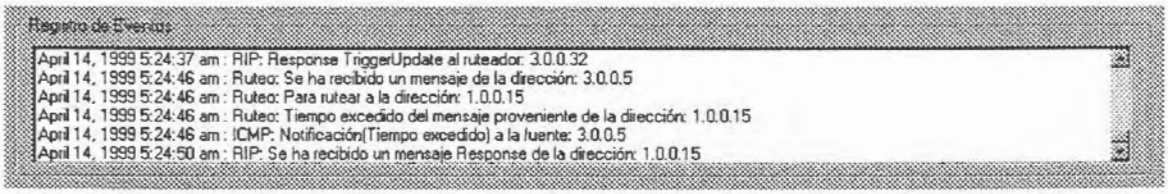
(a)



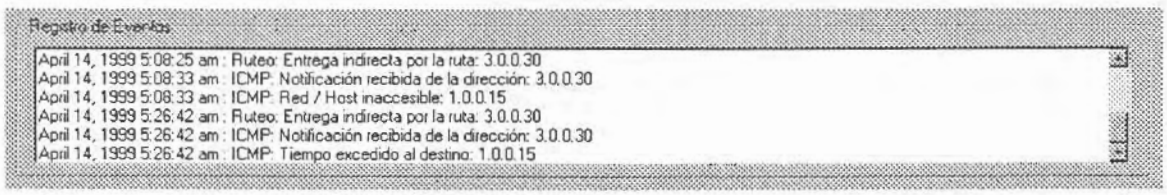
(b)

Figura 4.38. (a) Envío de H1 por R1 y notificación de destino inaccesible.
(b) Recepción de R1 desde H1 hacia el destino 1.0.0.15, destino inaccesible, notificación (destino inaccesible) a la fuente H1.

Cámbiese la métrica a un valor válido y envíese un mensaje hacia R2 desde H1 pero con el tiempo de vida de 1. Cuando llegue el mensaje a R1 este generará un mensaje ICMP para H1 de tiempo de vida excedido (véase fig. 4.39).



(a)



(b)

Figura 4.39. (a) Envío de H1 por R1 y notificación de tiempo excedido del mensaje. (b) Recepción de R1 desde H1 hacia el destino 1.0.0.15, tiempo excedido del mensaje, notificación (tiempo excedido) a la fuente H1.

Reemplácese la métrica a infinito en R1 para la red 1.0.0.0. Presione en H1 el botón **Request**, con esto generamos una petición *request* para R1 de monitoreo (ver fig. 4.40) y verifique que la métrica ha cambiado de valor para la red 1.0.0.0 con respecto a la que nosotros tenemos (ver fig. 4.41). Este tipo de petición solamente se utiliza para monitorear las tablas de ruteo, ya que no intervienen en la información de ruteo contenida en las tablas de los ruteadores.

Nótese que RIP maneja las subredes como redes independientes. Para mayor información y referencias consulte el RFC 2453 en el apéndice.

Para poder hacer más pruebas se cuenta con la opción de poder dar al inicio a cada ruteador una tabla de ruteo inicial (**Ruteador, Tabla de Ruteo**) y con ello manejar diversas situaciones de contenido y principalmente la propagación y el manejo de errores. Además se puede manejar ruteadores predeterminados (véase figura 4.42). También se pueden manejar rutas a anfitriones específicos, sólo se tiene que utilizar la máscara con todos unos (255.255.255.255). Hay muchas situaciones que se pueden emular, la herramienta está abierta a todas ellas.

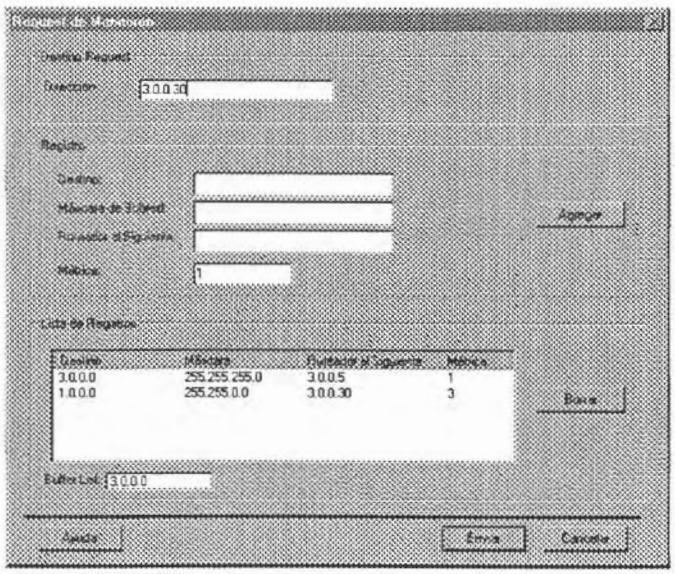


Figura 4.40. Request de monitoreo de H1 a R1 (3.0.0.30).

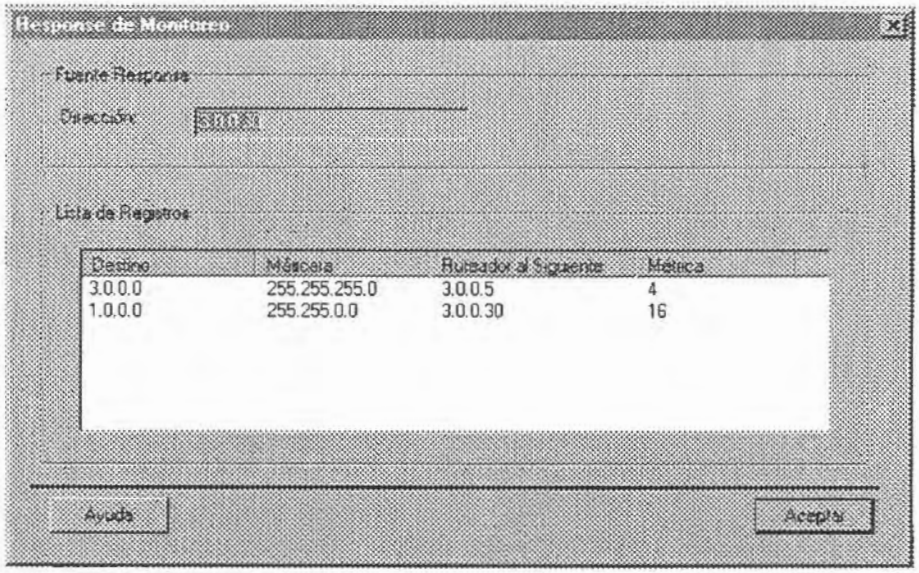


Figura 4.41. Response de monitoreo enviado por R1 para H1.

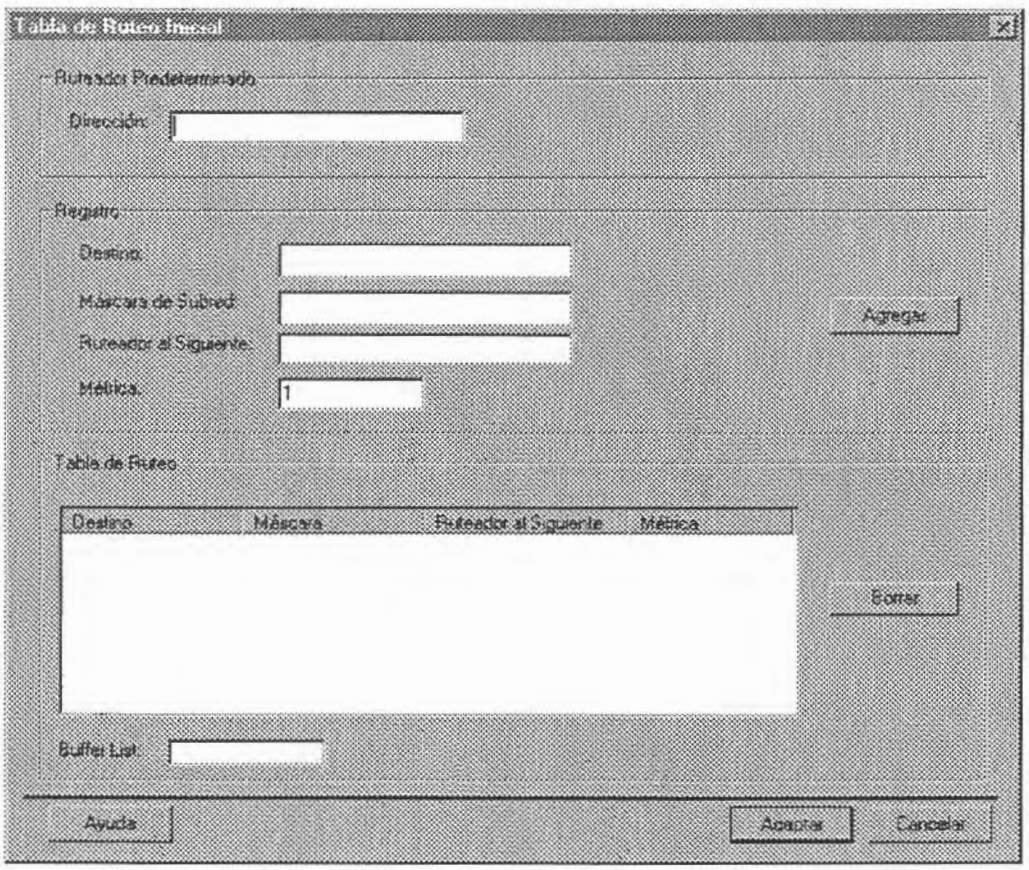


Figura 4.42. Ventana para asignar un ruteador predeterminado y una tabla de ruteo inicial al ruteador a emular.

Conclusión

Conclusión

El protocolo Internet es parte importante en la creación de una red de redes virtual. Con ello se mostró que el desarrollo de programas bajo esta tecnología permite hacer a un lado los detalles del hardware subyacente de red.

Nuestra emulación permite visualizar de forma más detallada los procesos referentes al funcionamiento de la capa IP. Podemos observar la operabilidad, el algoritmo de ruteo, el funcionamiento del protocolo interno RIP y principalmente el manejo de errores.

Sin duda alguna que los elementos visuales permiten una mejor comprensión. Con ello el programa Emulador Server aprovecha y da al alumno una mejor visión de la arquitectura, funcionamiento y diseño de la Internet.

Se realizaron pruebas con los alumnos de décimo semestre de la carrera de electrónica para comprobar la operabilidad del programa. Los alumnos pudieron experimentar con el protocolo IP, ICMP, ARP y RIP. Hubo una mejor comprensión y aprendizaje.

Después de realizar las pruebas con los alumnos, se les aplicó una pequeña evaluación. Los resultados mostraron un mejoría con respecto a una evaluación anterior al uso del programa.

Durante las pruebas surgieron dudas de los alumnos con respecto al protocolo RIP, las cuales fueron disipadas con la emulación de situaciones que relacionaban las interrogantes producidas.

Conforme se realizaban las prácticas las preguntas iban siendo más específicas sobre los procesos de ruteo, notificaciones ICMP y el protocolo IP.

No hubo dificultades y obstáculos al respecto.

Se pudo observar que el programa invitaba al alumno a profundizar más sobre los temas tratados y con ello tener una visión clara de los procesos mencionados.

Apéndice

Apéndice

Network Working Group G. Malkin
Request for Comments: 2453 Bay Networks
Obsoletes: 1723, 1388 November 1998
STD: 56
Category: Standards Track

RIP Version 2

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

This document specifies an extension of the Routing Information Protocol (RIP), as defined in [1], to expand the amount of useful information carried in RIP messages and to add a measure of security.

A companion document will define the SNMP MIB objects for RIP-2 [2]. An additional document will define cryptographic security improvements for RIP-2 [3].

Acknowledgements

I would like to thank the IETF RIP Working Group for their help in improving the RIP-2 protocol. Much of the text for the background discussions about distance vector protocols and some of the descriptions of the operation of RIP were taken from "Routing Information Protocol" by C. Hedrick [1]. Some of the final editing on the document was done by Scott Bradner.

Table of Contents

1. Justification

With the advent of OSPF and IS-IS, there are those who believe that RIP is obsolete. While it is true that the newer IGP routing protocols are far superior to RIP, RIP does have some advantages. Primarily, in a small network, RIP has very little overhead in terms of bandwidth used and configuration and management time. RIP is also very easy to implement, especially in relation to the newer IGPs.

Additionally, there are many, many more RIP implementations in the field than OSPF and IS-IS combined. It is likely to remain that way for some years yet.

Given that RIP will be useful in many environments for some period of time, it is reasonable to increase RIP's usefulness. This is especially true since the gain is far greater than the expense of the change.

2. Current RIP

The current RIP-1 message contains the minimal amount of information necessary for routers to route messages through a network. It also contains a large amount of unused space, owing to its origins.

The current RIP-1 protocol does not consider autonomous systems and IGP/EGP interactions, subnetting [11], and authentication since implementations of these postdate RIP-1. The lack of subnet masks is a particularly serious problem for routers since they need a subnet mask to know how to determine a route. If a RIP-1 route is a network route (all non-network bits 0), the subnet mask equals the network mask. However, if some of the non-network bits are set, the router cannot determine the subnet mask. Worse still, the router cannot determine if the RIP-1 route is a subnet route or a host route. Currently, some routers simply choose the subnet mask of the interface over which the route was learned and determine the route type from that.

3. Basic Protocol

3.1 Introduction

RIP is a routing protocol based on the Bellman-Ford (or distance vector) algorithm. This algorithm has been used for routing computations in computer networks since the early days of the ARPANET. The particular packet formats and protocol described here are based on the program "routed," which is included with the Berkeley distribution of Unix. In an international network, such as the Internet, it is very unlikely that a single routing protocol will used for the entire network. Rather, the network will be organized as a collection of Autonomous Systems (AS), each of which will, in general, be administered by a single entity. Each AS will have its own routing technology, which may differ among AS's. The routing protocol used within an AS is referred to as an Interior Gateway Protocol (IGP). A separate protocol, called an Exterior Gateway Protocol (EGP), is used to transfer routing information among the AS's. RIP was designed to work as an IGP in moderate-size AS's. It is not intended for use in more complex environments. For information on the context into which RIP-1 is expected to fit, see Braden and Postel [6].

RIP uses one of a class of routing algorithms known as Distance Vector algorithms. The earliest description of this class of algorithms known to the author is in Ford and Fulkerson [8]. Because of this, they are sometimes known as Ford-Fulkerson algorithms. The term Bellman-Ford is also used, and derives from the fact that the formulation is based on Bellman's equation [4]. The presentation in this document is closely based on [5]. This document contains a protocol specification. For an introduction to the mathematics of routing algorithms, see [1]. The basic algorithms used by this protocol were used in computer routing as early as 1969 in the ARPANET. However, the specific ancestry of this protocol is within the Xerox network protocols. The PUP protocols [7] used the Gateway Information Protocol to exchange routing information. A somewhat updated version of this protocol was adopted for the Xerox Network Systems (XNS) architecture, with the name Routing Information Protocol [9]. Berkeley's routed is largely the same as the Routing Information Protocol, with XNS addresses replaced by a more general address format capable of handling IPv4 and other types of address, and with routing updates limited to one every 30 seconds. Because of this similarity, the term Routing Information Protocol (or just RIP) is used to refer to both the XNS protocol and the protocol used by routed.

RIP is intended for use within the IP-based Internet. The Internet is organized into a number of networks connected by special purpose gateways known as routers. The networks may be either point-to-point links or more complex networks such as Ethernet or token ring. Hosts and routers are presented with IP datagrams addressed to some host. Routing is the method by which the host or router decides where to send the datagram. It may be able to send the datagram directly to the destination, if that destination is on one of the networks that are directly connected to the host or router. However, the interesting case is when the destination is not directly reachable. In this case, the host or router attempts to send the datagram to a router that is nearer the destination. The goal of a routing protocol is very simple: It is to supply the information that is needed to do routing.

3.2 Limitations of the Protocol

This protocol does not solve every possible routing problem. As mentioned above, it is primary intended for use as an IGP in networks of moderate size. In addition, the following specific limitations are be mentioned:

- The protocol is limited to networks whose longest path (the network's diameter) is 15 hops. The designers believe that the basic protocol design is inappropriate for larger networks. Note that this statement of the limit assumes that a cost of 1 is used for each network. This is the way RIP is normally configured. If the system administrator chooses to use larger costs, the upper bound of 15 can easily become a problem.

- The protocol depends upon "counting to infinity" to resolve certain unusual situations. (This will be explained in the next section.) If the system of networks has several hundred networks, and a routing loop was formed involving all of them, the resolution of the loop would require either much time (if the frequency of routing updates were limited) or bandwidth (if updates were sent whenever changes were detected). Such a loop would consume a large amount of network

bandwidth before the loop was corrected. We believe that in realistic cases, this will not be a problem except on slow lines. Even then, the problem will be fairly unusual, since various precautions are taken that should prevent these problems in most cases.

- This protocol uses fixed "metrics" to compare alternative routes. It is not appropriate for situations where routes need to be chosen based on real-time parameters such a measured delay, reliability, or load. The obvious extensions to allow metrics of this type are likely to introduce instabilities of a sort that the protocol is not designed to handle.

3.3. Organization of this document

The main body of this document is organized into two parts, which occupy the next two sections:

A conceptual development and justification of distance vector algorithms in general.

The actual protocol description.

Each of these two sections can largely stand on its own. Section 3.4 attempts to give an informal presentation of the mathematical underpinnings of the algorithm. Note that the presentation follows a "spiral" method. An initial, fairly simple algorithm is described. Then refinements are added to it in successive sections. Section 3.5 is the actual protocol description. Except where specific references are made to section 3.4, it should be possible to implement RIP entirely from the specifications given in section 3.5.

3.4 Distance Vector Algorithms

Routing is the task of finding a path from a sender to a desired destination. In the IP "Internet model" this reduces primarily to a matter of finding a series of routers between the source and destination networks. As long as a message or datagram remains on a single network or subnet, any forwarding problems are the responsibility of technology that is specific to the network. For example, Ethernet and the ARPANET each define a way in which any sender can talk to any specified destination within that one network. IP routing comes in primarily when messages must go from a sender on one network to a destination on a different one. In that case, the message must pass through one or more routers connecting the networks. If the networks are not adjacent, the message may pass through several intervening networks, and the routers connecting them. Once the message gets to a router that is on the same network as the destination, that network's own technology is used to get to the destination.

Throughout this section, the term "network" is used generically to cover a single broadcast network (e.g., an Ethernet), a point to point line, or the ARPANET. The critical point is that a network is treated as a single entity by IP. Either no forwarding decision is necessary (as with a point to point line), or that forwarding is done in a manner that is transparent to IP, allowing IP to treat the entire network as a single fully-connected system (as with an Ethernet or the ARPANET). Note that the term "network" is used in a somewhat different way in discussions of IP addressing. We are using the term "network" here to refer to subnets in cases where subnet addressing is in use.

A number of different approaches for finding routes between networks are possible. One useful way of categorizing these approaches is on the basis of the type of information the routers need to exchange in order to be able to find routes. Distance vector algorithms are based on the exchange of only a small amount of information. Each entity (router or host) that participates in the routing protocol is assumed to keep information about all of the destinations within the system. Generally, information about all entities connected to one network is summarized by a single entry, which describes the route to all destinations on that network. This summarization is possible because as far as IP is concerned, routing within a network is invisible. Each entry in this routing database includes the next router to which datagrams destined for the entity should be sent. In addition, it

includes a "metric" measuring the total distance to the entity. Distance is a somewhat generalized concept, which may cover the time delay in getting messages to the entity, the dollar cost of sending messages to it, etc. Distance vector algorithms get their name from the fact that it is possible to compute optimal routes when the only information exchanged is the list of these distances. Furthermore, information is only exchanged among entities that are adjacent, that is, entities that share a common network.

Although routing is most commonly based on information about networks, it is sometimes necessary to keep track of the routes to individual hosts. The RIP protocol makes no formal distinction between networks and hosts. It simply describes exchange of information about destinations, which may be either networks or hosts. (Note however, that it is possible for an implementor to choose not to support host routes. See section 3.2.) In fact, the mathematical developments are most conveniently thought of in terms of routes from one host or router to another. When discussing the algorithm in abstract terms, it is best to think of a routing entry for a network as an abbreviation for routing entries for all of the entities connected to that network. This sort of abbreviation makes sense only because we think of networks as having no internal structure that is visible at the IP level. Thus, we will generally assign the same distance to every entity in a given network.

We said above that each entity keeps a routing database with one entry for every possible destination in the system. An actual implementation is likely to need to keep the following information about each destination:

- address: in IP implementations of these algorithms, this will be the IP address of the host or network.

- router: the first router along the route to the destination.

- interface: the physical network which must be used to reach the first router.

- metric: a number, indicating the distance to the destination.

- timer: the amount of time since the entry was last updated.

In addition, various flags and other internal information will probably be included. This database is initialized with a description of the entities that are directly connected to the system. It is updated according to information received in messages from neighboring routers.

The most important information exchanged by the hosts and routers is carried in update messages. Each entity that participates in the routing scheme sends update messages that describe the routing database as it currently exists in that entity. It is possible to maintain optimal routes for the entire system by using only information obtained from neighboring entities. The algorithm used for that will be described in the next section.

As we mentioned above, the purpose of routing is to find a way to get datagrams to their ultimate destinations. Distance vector algorithms are based on a table in each router listing the best route to every destination in the system. Of course, in order to define which route is best, we have to have some way of measuring goodness. This is referred to as the "metric".

In simple networks, it is common to use a metric that simply counts how many routers a message must go through. In more complex networks, a metric is chosen to represent the total amount of delay that the message suffers, the cost of sending it, or some other quantity which may be minimized. The main requirement is that it must be possible to represent the metric as a sum of "costs" for individual hops.

Formally, if it is possible to get from entity i to entity j directly (i.e., without passing through another router between), then a cost, d(i,j), is associated with the hop between i and j. In the normal case where all entities on a given network are considered to be the same, d(i,j) is the same for all destinations on a given network, and represents the cost of using that network. To get the metric of a complete route, one just adds up the costs of the individual hops that make up the route. For the purposes of this memo, we assume that the costs are positive integers.

Let D(i,j) represent the metric of the best route from entity i to entity j. It should be defined for every pair of entities. d(i,j) represents the costs of the individual steps. Formally, let d(i,j) represent the cost of going directly from entity i to entity j. It is infinite if i and j are not immediate neighbors. (Note that d(i,i) is infinite. That is, we don't consider there to be a direct connection from a node to itself.) Since costs are additive, it is easy to show that the best metric must be described by

$$D(i,i) = 0, \qquad \text{all } i$$
$$D(i,j) = \min_k [d(i,k) + D(k,j)], \quad \text{otherwise}$$

and that the best routes start by going from i to those neighbors k for which d(i,k) + D(k,j) has the minimum value. (These things can be shown by induction on the number of steps in the routes.) Note that we can limit the second equation to k's that are immediate neighbors of i. For the others, d(i,k) is infinite, so the term involving them can never be the minimum.

It turns out that one can compute the metric by a simple algorithm based on this. Entity i gets its neighbors k to send it their estimates of their distances to the destination j. When i gets the estimates from k, it adds d(i,k) to each of the numbers. This is simply the cost of traversing the network between i and k. Now and then i compares the values from all of its neighbors and picks the smallest.

A proof is given in [2] that this algorithm will converge to the correct estimates of D(i,j) in finite time in the absence of topology changes. The authors make very few assumptions about the order in which the entities send each other their information, or when the min is recomputed. Basically, entities just can't stop sending updates or recomputing metrics, and the networks can't delay messages forever. (Crash of a routing entity is a topology change.) Also, their proof does not make any assumptions about the initial estimates of D(i,j), except that they must be non-negative. The fact that these fairly weak assumptions are good enough is important. Because we don't have to make assumptions about when updates are sent, it is safe to run the algorithm asynchronously. That is, each entity can send updates according to its own clock. Updates can be dropped by the network, as long as they don't all get dropped. Because we don't have to make assumptions about the starting condition, the algorithm can handle changes. When the system changes, the routing algorithm starts moving to a new equilibrium, using the old one as its starting point. It is important that the algorithm will converge in finite time no matter what the starting point. Otherwise certain kinds of changes might lead to non-convergent behavior.

The statement of the algorithm given above (and the proof) assumes that each entity keeps copies of the estimates that come from each of its neighbors, and now and then does a min over all of the neighbors. In fact real implementations don't necessarily do that. They simply remember the best metric seen so far, and the identity of the neighbor that sent it. They replace this information whenever they see a better (smaller) metric. This allows them to compute the minimum incrementally, without having to store data from all of the neighbors.

There is one other difference between the algorithm as described in texts and those used in real protocols such as RIP: the description above would have each entity include an entry for itself, showing a distance of zero. In fact this is not generally done. Recall that all entities on a network are normally summarized by a single entry for the network. Consider the situation of a host or router G that is connected to network A. C represents the cost of using network A (usually a metric of one). (Recall that we are assuming that the internal structure of a network is not visible to IP, and thus the cost of going between any two entities on it is the same.) In principle, G should get a

message from every other entity H on network A, showing a cost of 0 to get from that entity to itself. G would then compute C + 0 as the distance to H. Rather than having G look at all of these identical messages, it simply starts out by making an entry for network A in its table, and assigning it a metric of C. This entry for network A should be thought of as summarizing the entries for all other entities on network A. The only entity on A that can't be summarized by that common entry is G itself, since the cost of going from G to G is 0, not C. But since we never need those 0 entries, we can safely get along with just the single entry for network A. Note one other implication of this strategy: because we don't need to use the 0 entries for anything, hosts that do not function as routers don't need to send any update messages. Clearly hosts that don't function as routers (i.e., hosts that are connected to only one network) can have no useful information to contribute other than their own entry D(i,i) = 0. As they have only the one interface, it is easy to see that a route to any other network through them will simply go in that interface and then come right back out it. Thus the cost of such a route will be greater than the best cost by at least C. Since we don't need the 0 entries, non- routers need not participate in the routing protocol at all.

Let us summarize what a host or router G does. For each destination in the system, G will keep a current estimate of the metric for that destination (i.e., the total cost of getting to it) and the identity of the neighboring router on whose data that metric is based. If the destination is on a network that is directly connected to G, then G simply uses an entry that shows the cost of using the network, and the fact that no router is needed to get to the destination. It is easy to show that once the computation has converged to the correct metrics, the neighbor that is recorded by this technique is in fact the first router on the path to the destination. (If there are several equally good paths, it is the first router on one of them.) This combination of destination, metric, and router is typically referred to as a route to the destination with that metric, using that router.

4.ne The method so far only has a way to lower the metric, as the existing metric is kept until a smaller one shows up. It is possible that the initial estimate might be too low. Thus, there must be a way to increase the metric. It turns out to be sufficient to use the following rule: suppose the current route to a destination has metric D and uses router G. If a new set of information arrived from some source other than G, only update the route if the new metric is better than D. But if a new set of information arrives from G itself, always update D to the new value. It is easy to show that with this rule, the incremental update process produces the same routes as a calculation that remembers the latest information from all the neighbors and does an explicit minimum. (Note that the discussion so far assumes that the network configuration is static. It does not allow for the possibility that a system might fail.)

To summarize, here is the basic distance vector algorithm as it has been developed so far. (Note that this is not a statement of the RIP protocol. There are several refinements still to be added.) The following procedure is carried out by every entity that participates in the routing protocol. This must include all of the routers in the system. Hosts that are not routers may participate as well.

- Keep a table with an entry for every possible destination in the system. The entry contains the distance D to the destination, and the first router G on the route to that network. Conceptually, there should be an entry for the entity itself, with metric 0, but this is not actually included.

- Periodically, send a routing update to every neighbor. The update is a set of messages that contain all of the information from the routing table. It contains an entry for each destination, with the distance shown to that destination.

- When a routing update arrives from a neighbor G', add the cost associated with the network that is shared with G'. (This should be the network over which the update arrived.) Call the resulting distance D'. Compare the resulting distances with the current routing table entries. If the new distance D' for N is smaller than the existing value D, adopt the new route. That is, change the table entry for N to have metric D' and router G'. If G' is the router from which the existing route came, i.e., G' = G, then use the new metric even if it is larger than the old one.

3.4.1 Dealing with changes in topology

The discussion above assumes that the topology of the network is fixed. In practice, routers and lines often fail and come back up. To handle this possibility, we need to modify the algorithm slightly.

The theoretical version of the algorithm involved a minimum over all immediate neighbors. If the topology changes, the set of neighbors changes. Therefore, the next time the calculation is done, the change will be reflected. However, as mentioned above, actual implementations use an incremental version of the minimization. Only the best route to any given destination is remembered. If the router involved in that route should crash, or the network connection to it break, the calculation might never reflect the change. The algorithm as shown so far depends upon a router notifying its neighbors if its metrics change. If the router crashes, then it has no way of notifying neighbors of a change.

In order to handle problems of this kind, distance vector protocols must make some provision for timing out routes. The details depend upon the specific protocol. As an example, in RIP every router that participates in routing sends an update message to all its neighbors once every 30 seconds. Suppose the current route for network N uses router G. If we don't hear from G for 180 seconds, we can assume that either the router has crashed or the network connecting us to it has become unusable. Thus, we mark the route as invalid. When we hear from another neighbor that has a valid route to N, the valid route will replace the invalid one. Note that we wait for 180 seconds before timing out a route even though we expect to hear from each neighbor every 30 seconds. Unfortunately, messages are occasionally lost by networks. Thus, it is probably not a good idea to invalidate a route based on a single missed message.

As we will see below, it is useful to have a way to notify neighbors that there currently isn't a valid route to some network. RIP, along with several other protocols of this class, does this through a normal update message, by marking that network as unreachable. A specific metric value is chosen to indicate an unreachable destination; that metric value is larger than the largest valid metric that we expect to see. In the existing implementation of RIP, 16 is used. This value is normally referred to as "infinity", since it is larger than the largest valid metric. 16 may look like a surprisingly small number. It is chosen to be this small for reasons that we will see shortly. In most implementations, the same convention is used internally to flag a route as invalid.

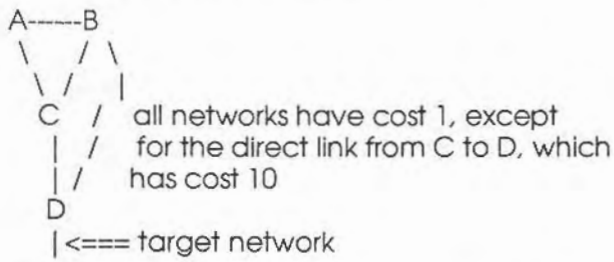
3.4.2 Preventing instability

The algorithm as presented up to this point will always allow a host or router to calculate a correct routing table. However, that is still not quite enough to make it useful in practice. The proofs referred to above only show that the routing tables will converge to the correct values in finite time. They do not guarantee that this time will be small enough to be useful, nor do they say what will happen to the metrics for networks that become inaccessible.

It is easy enough to extend the mathematics to handle routes becoming inaccessible. The convention suggested above will do that. We choose a large metric value to represent "infinity". This value must be large enough that no real metric would ever get that large. For the purposes of this example, we will use the value 16. Suppose a network becomes inaccessible. All of the immediately neighboring routers time out and set the metric for that network to 16. For purposes of analysis, we can assume that all the neighboring routers have gotten a new piece of hardware that connects them directly to the vanished network, with a cost of 16. Since that is the only connection to the vanished network, all the other routers in the system will converge to new routes that go through one of those routers. It is easy to see that once convergence has happened, all the routers will have metrics of at least 16 for the vanished network. Routers one hop away from the original neighbors would end up with metrics of at least 17; routers two hops away would end up with at least 18, etc. As these metrics are larger than the maximum metric value, they are all set to 16. It

is obvious that the system will now converge to a metric of 16 for the vanished network at all routers.

Unfortunately, the question of how long convergence will take is not amenable to quite so simple an answer. Before going any further, it will be useful to look at an example (taken from [2]). Note that what we are about to show will not happen with a correct implementation of RIP. We are trying to show why certain features are needed. In the following example the letters correspond to routers, and the lines to networks.



Each router will have a table showing a route to each network.

However, for purposes of this illustration, we show only the routes from each router to the network marked at the bottom of the diagram.

D: directly connected, metric 1
B: route via D, metric 2
C: route via B, metric 3
A: route via B, metric 3

Now suppose that the link from B to D fails. The routes should now adjust to use the link from C to D. Unfortunately, it will take a while for this to this to happen. The routing changes start when B notices that the route to D is no longer usable. For simplicity, the chart below assumes that all routers send updates at the same time. The chart shows the metric for the target network, as it appears in the routing table at each router.

```
time ------>

D: dir, 1   dir, 1   dir, 1   dir, 1  ...  dir, 1   dir, 1
B: unreach  C,   4   C,   5   C,   6       C,  11   C,  12
C: B,   3   A,   4   A,   5   A,   6       A,  11   D,  11
A: B,   3   C,   4   C,   5   C,   6       C,  11   C,  12

dir = directly connected
unreach = unreachable
```

Here's the problem: B is able to get rid of its failed route using a timeout mechanism, but vestiges of that route persist in the system for a long time. Initially, A and C still think they can get to D via B. So, they keep sending updates listing metrics of 3. In the next iteration, B will then claim that it can get to D via either A or C. Of course, it can't. The routes being claimed by A and C are now gone, but they have no way of knowing that yet. And even when they discover that their routes via B have gone away, they each think there is a route available via the other. Eventually the system converges, as all the mathematics claims it must. But it can take some time to do so. The worst case is when a network becomes completely inaccessible from some part of the system. In that case, the metrics may increase slowly in a pattern like the one above until they finally reach infinity. For this reason, the problem is called "counting to infinity".

You should now see why "infinity" is chosen to be as small as possible. If a network becomes completely inaccessible, we want counting to infinity to be stopped as soon as possible. Infinity must be large enough that no real route is that big. But it shouldn't be any bigger than required. Thus the choice of infinity is a tradeoff between network size and speed of convergence in case counting to infinity happens. The designers of RIP believed that the protocol was unlikely to be practical for networks with a diameter larger than 15.

There are several things that can be done to prevent problems like this. The ones used by RIP are called "split horizon with poisoned reverse", and "triggered updates".

3.4.3 Split horizon

Note that some of the problem above is caused by the fact that A and C are engaged in a pattern of mutual deception. Each claims to be able to get to D via the other. This can be prevented by being a bit more careful about where information is sent. In particular, it is never useful to claim reachability for a destination network to the neighbor(s) from which the route was learned. "Split horizon" is a scheme for avoiding problems caused by including routes in updates sent to the router from which they were learned. The "simple split horizon" scheme omits routes learned from one neighbor in updates sent to that neighbor. "Split horizon with poisoned reverse" includes such routes in updates, but sets their metrics to infinity.

If A thinks it can get to D via C, its messages to C should indicate that D is unreachable. If the route through C is real, then C either has a direct connection to D, or a connection through some other router. C's route can't possibly go back to A, since that forms a loop. By telling C that D is unreachable, A simply guards against the possibility that C might get confused and believe that there is a route through A. This is obvious for a point to point line. But consider the possibility that A and C are connected by a broadcast network such as an Ethernet, and there are other routers on that network. If A has a route through C, it should indicate that D is unreachable when talking to any other router on that network. The other routers on the network can get to C themselves. They would never need to get to C via A. If A's best route is really through C, no other router on that network needs to know that A can reach D. This is fortunate, because it means that the same update message that is used for C can be used for all other routers on the same network. Thus, update messages can be sent by broadcast.

In general, split horizon with poisoned reverse is safer than simple split horizon. If two routers have routes pointing at each other, advertising reverse routes with a metric of 16 will break the loop immediately. If the reverse routes are simply not advertised, the erroneous routes will have to be eliminated by waiting for a timeout. However, poisoned reverse does have a disadvantage: it increases the size of the routing messages. Consider the case of a campus backbone connecting a number of different buildings. In each building, there is a router connecting the backbone to a local network. Consider what routing updates those routers should broadcast on the backbone network. All that the rest of the network really needs to know about each router is what local networks it is connected to. Using simple split horizon, only those routes would appear in update messages sent by the router to the backbone network. If split horizon with poisoned reverse is used, the router must mention all routes that it learns from the backbone, with metrics of 16. If the system is large, this can result in a large update message, almost all of whose entries indicate unreachable networks.

In a static sense, advertising reverse routes with a metric of 16 provides no additional information. If there are many routers on one broadcast network, these extra entries can use significant bandwidth. The reason they are there is to improve dynamic behavior. When topology changes, mentioning routes that should not go through the router as well as those that should can speed up convergence. However, in some situations, network managers may prefer to accept somewhat slower convergence in order to minimize routing overhead. Thus implementors may at their option implement simple split horizon rather than split horizon with poisoned reverse, or they may provide a configuration option that allows the network manager to choose which behavior to use. It is also

permissible to implement hybrid schemes that advertise some reverse routes with a metric of 16 and omit others. An example of such a scheme would be to use a metric of 16 for reverse routes for a certain period of time after routing changes involving them, and thereafter omitting them from updates.

The router requirements RFC [11] specifies that all implementation of RIP must use split horizon and should also use split horizon with poisoned reverse, although there may be a knob to disable poisoned reverse.

3.4.4 Triggered updates

Split horizon with poisoned reverse will prevent any routing loops that involve only two routers. However, it is still possible to end up with patterns in which three routers are engaged in mutual deception.. For example, A may believe it has a route through B, B through C, and C through A. Split horizon cannot stop such a loop. This loop will only be resolved when the metric reaches infinity and the network involved is then declared unreachable. Triggered updates are an attempt to speed up this convergence. To get triggered updates, we simply add a rule that whenever a router changes the metric for a route, it is required to send update messages almost immediately, even if it is not yet time for one of the regular update message. (The timing details will differ from protocol to protocol. Some distance vector protocols, including RIP, specify a small time delay, in order to avoid having triggered updates generate excessive network traffic.) Note how this combines with the rules for computing new metrics. Suppose a router's route to destination N goes through router G. If an update arrives from G itself, the receiving router is required to believe the new information, whether the new metric is higher or lower than the old one. If the result is a change in metric, then the receiving router will send triggered updates to all the hosts and routers directly connected to it. They in turn may each send updates to their neighbors. The result is a cascade of triggered updates. It is easy to show which routers and hosts are involved in the cascade. Suppose a router G times out a route to destination N. G will send triggered updates to all of its neighbors. However, the only neighbors who will believe the new information are those whose routes for N go through G. The other routers and hosts will see this as information about a new route that is worse than the one they are already using, and ignore it. The neighbors whose routes go through G will update their metrics and send triggered updates to all of their neighbors. Again, only those neighbors whose routes go through them will pay attention. Thus, the triggered updates will propagate backwards along all paths leading to router G, updating the metrics to infinity. This propagation will stop as soon as it reaches a portion of the network whose route to destination N takes some other path.

If the system could be made to sit still while the cascade of triggered updates happens, it would be possible to prove that counting to infinity will never happen. Bad routes would always be removed immediately, and so no routing loops could form.

Unfortunately, things are not so nice. While the triggered updates are being sent, regular updates may be happening at the same time. Routers that haven't received the triggered update yet will still be sending out information based on the route that no longer exists. It is possible that after the triggered update has gone through a router, it might receive a normal update from one of these routers that hasn't yet gotten the word. This could reestablish an orphaned remnant of the faulty route. If triggered updates happen quickly enough, this is very unlikely. However, counting to infinity is still possible.

The router requirements RFC [11] specifies that all implementation of RIP must implement triggered update for deleted routes and may implement triggered updates for new routes or change of routes. RIP implementations must also limit the rate which of triggered updates may be trandmitted. (see section 3.10.1)

3.5 Protocol Specification

RIP is intended to allow routers to exchange information for computing routes through an IPv4-based network. Any router that uses RIP is assumed to have interfaces to one or more networks, otherwise it isn't really a router. These are referred to as its directly- connected networks. The protocol relies on access to certain information about each of these networks, the most important of which is its metric. The RIP metric of a network is an integer between 1 and 15, inclusive. It is set in some manner not specified in this protocol; however, given the maximum path limit of 15, a value of 1 is usually used. Implementations should allow the system administrator to set the metric of each network. In addition to the metric, each network will have an IPv4 destination address and subnet mask associated with it. These are to be set by the system administrator in a manner not specified in this protocol.

Any host that uses RIP is assumed to have interfaces to one or more networks. These are referred to as its "directly-connected networks". The protocol relies on access to certain information about each of these networks. The most important is its metric or "cost". The metric of a network is an integer between 1 and 15 inclusive. It is set in some manner not specified in this protocol. Most existing implementations always use a metric of 1. New implementations should allow the system administrator to set the cost of each network. In addition to the cost, each network will have an IPv4 network number and a subnet mask associated with it. These are to be set by the system administrator in a manner not specified in this protocol.

Note that the rules specified in section 3.7 assume that there is a single subnet mask applying to each IPv4 network, and that only the subnet masks for directly-connected networks are known. There may be systems that use different subnet masks for different subnets within a single network. There may also be instances where it is desirable for a system to know the subnets masks of distant networks. Network- wide distribution of routing information which contains different subnet masks is permitted if all routers in the network are running the extensions presented in this document. However, if all routers in the network are not running these extensions distribution of routing information containing different subnet masks must be limited to avoid interoperability problems. See sections 3.7 and 4.3 for the rules governing subnet distribution.

Each router that implements RIP is assumed to have a routing table. This table has one entry for every destination that is reachable throughout the system operating RIP. Each entry contains at least the following information:

- The IPv4 address of the destination.

- A metric, which represents the total cost of getting a datagram from the router to that destination. This metric is the sum of the costs associated with the networks that would be traversed to get to the destination.

- The IPv4 address of the next router along the path to the destination (i.e., the next hop). If the destination is on one of the directly-connected networks, this item is not needed.

- A flag to indicate that information about the route has changed recently. This will be referred to as the "route change flag."

- Various timers associated with the route. See section 3.6 for more details on timers.

The entries for the directly-connected networks are set up by the router using information gathered by means not specified in this protocol. The metric for a directly-connected network is set to the cost of that network. As mentioned, 1 is the usual cost. In that case, the RIP metric reduces to a simple hop-count. More complex metrics may be used when it is desirable to show preference for some networks over others (e.g., to indicate of differences in bandwidth or reliability).

To support the extensions detailed in this document, each entry must additionally contain a subnet mask. The subnet mask allows the router (along with the IPv4 address of the destination) to

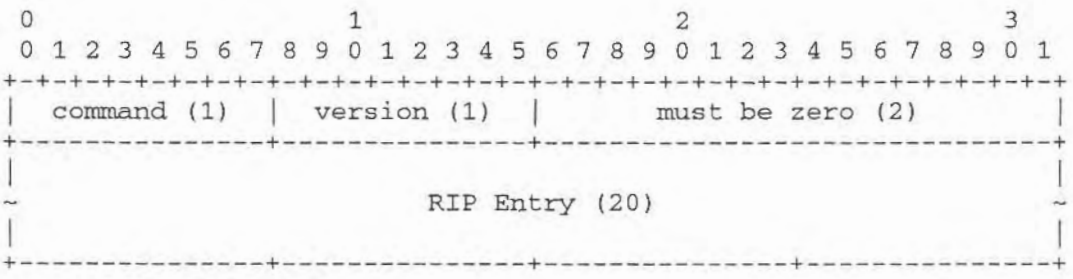
identify the different subnets within a single network as well as the subnets masks of distant networks. Implementors may also choose to allow the system administrator to enter additional routes. These would most likely be routes to hosts or networks outside the scope of the routing system. They are referred to as "static routes." Entries for destinations other than these initial ones are added and updated by the algorithms described in the following sections.

In order for the protocol to provide complete information on routing, every router in the AS must participate in the protocol. In cases where multiple IGPs are in use, there must be at least one router which can leak routing information between the protocols.

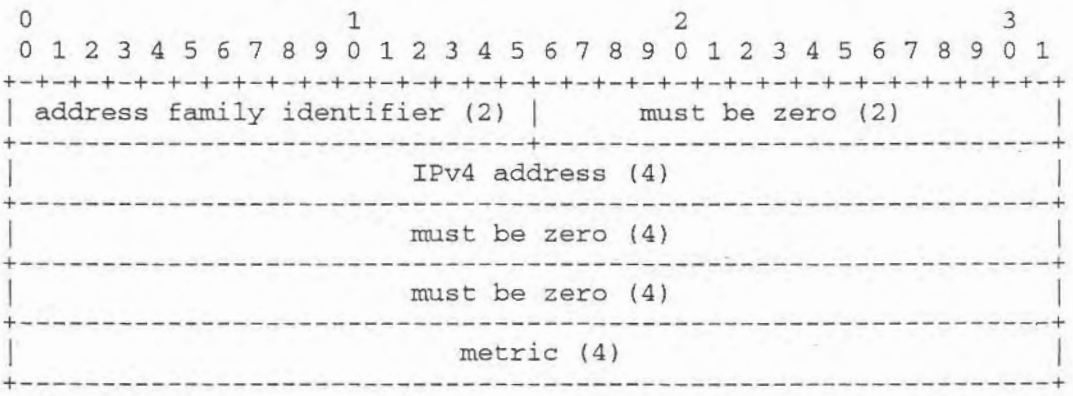
3.6 Message Format

RIP is a UDP-based protocol. Each router that uses RIP has a routing process that sends and receives datagrams on UDP port number 520, the RIP-1/RIP-2 port. All communications intended for another routers's RIP process are sent to the RIP port. All routing update messages are sent from the RIP port. Unsolicited routing update messages have both the source and destination port equal to the RIP port. Update messages sent in response to a request are sent to the port from which the request came. Specific queries may be sent from ports other than the RIP port, but they must be directed to the RIP port on the target machine.

The RIP packet format is:



There may be between 1 and 25 (inclusive) RIP entries. A RIP-1 entry has the following format:



Field sizes are given in octets. Unless otherwise specified, fields contain binary integers, in network byte order, with the most- significant octet first (big-endian). Each tick mark represents one bit.

Every message contains a RIP header which consists of a command and a version number. This section of the document describes version 1 of the protocol; section 4 describes the version 2 extensions. The command field is used to specify the purpose of this message. The commands implemented in version 1 and 2 are:

1 - request A request for the responding system to send all or part of its routing table.

2 - response A message containing all or part of the sender's routing table. This message may be sent in response to a request, or it may be an unsolicited routing update generated by the sender.

For each of these message types, in version 1, the remainder of the datagram contains a list of Route Entries (RTEs). Each RTE in this list contains an Address Family Identifier (AFI), destination IPv4 address, and the cost to reach that destination (metric).

The AFI is the type of address. For RIP-1, only AF_INET (2) is generally supported.

The metric field contains a value between 1 and 15 (inclusive) which specifies the current metric for the destination; or the value 16 (infinity), which indicates that the destination is not reachable.

3.7 Addressing Considerations

Distance vector routing can be used to describe routes to individual hosts or to networks. The RIP protocol allows either of these possibilities. The destinations appearing in request and response messages can be networks, hosts, or a special code used to indicate a default address. In general, the kinds of routes actually used will depend upon the routing strategy used for the particular network. Many networks are set up so that routing information for individual hosts is not needed. If every node on a given network or subnet is accessible through the same routers, then there is no reason to mention individual hosts in the routing tables. However, networks that include point-to-point lines sometimes require routers to keep track of routes to certain nodes. Whether this feature is required depends upon the addressing and routing approach used in the system. Thus, some implementations may choose not to support host routes. If host routes are not supported, they are to be dropped when they are received in response messages (see section 3.7.2).

The RIP-1 packet format does not distinguish among various types of address. Fields that are labeled "address" can contain any of the following:

host address subnet number network number zero (default route)

Entities which use RIP-1 are assumed to use the most specific information available when routing a datagram. That is, when routing a datagram, its destination address must first be checked against the list of node addresses. Then it must be checked to see whether it matches any known subnet or network number. Finally, if none of these match, the default route is used.

When a node evaluates information that it receives via RIP-1, its interpretation of an address depends upon whether it knows the subnet mask that applies to the net. If so, then it is possible to determine the meaning of the address. For example, consider net 128.6. It has a subnet mask of 255.255.255.0. Thus 128.6.0.0 is a network number, 128.6.4.0 is a subnet number, and 128.6.4.1 is a node address. However, if the node does not know the subnet mask, evaluation of an address may be ambiguous. If there is a non-zero node part, there is no clear way to determine whether the address represents a subnet number or a node address. As a subnet number would be useless without the subnet mask, addresses are assumed to represent nodes in this situation. In order to avoid this sort of ambiguity, when using version 1, nodes must not send subnet routes to nodes that cannot be expected to know the appropriate subnet mask. Normally hosts only know the subnet masks for directly-connected networks. Therefore, unless special provisions have been made, routes to a subnet must not be sent outside the network of which the subnet is a part. RIP-2 (see section 4) eliminates the subnet/host ambiguity by including the subnet mask in the routing entry.

This "subnet filtering" is carried out by the routers at the "border" of the subnetted network. These are routers which connect that network with some other network. Within the subnetted network,

each subnet is treated as an individual network. Routing entries for each subnet are circulated by RIP. However, border routers send only a single entry for the network as a whole to nodes in other networks. This means that a border router will send different information to different neighbors. For neighbors connected to the subnetted network, it generates a list of all subnets to which it is directly connected, using the subnet number. For neighbors connected to other networks, it makes a single entry for the network as a whole, showing the metric associated with that network. This metric would normally be the smallest metric for the subnets to which the router is attached.

Similarly, border routers must not mention host routes for nodes within one of the directly-connected networks in messages to other networks. Those routes will be subsumed by the single entry for the network as a whole.

The router requirements RFC [11] specifies that all implementation of RIP should support host routes but if they do not then they must ignore any received host routes.

The special address 0.0.0.0 is used to describe a default route. A default route is used when it is not convenient to list every possible network in the RIP updates, and when one or more closely-connected routers in the system are prepared to handle traffic to the networks that are not listed explicitly. These routers should create RIP entries for the address 0.0.0.0, just as if it were a network to which they are connected. The decision as to how routers create entries for 0.0.0.0 is left to the implementor. Most commonly, the system administrator will be provided with a way to specify which routers should create entries for 0.0.0.0; however, other mechanisms are possible. For example, an implementor might decide that any router which speaks BGP should be declared to be a default router. It may be useful to allow the network administrator to choose the metric to be used in these entries. If there is more than one default router, this will make it possible to express a preference for one over the other. The entries for 0.0.0.0 are handled by RIP in exactly the same manner as if there were an actual network with this address. System administrators should take care to make sure that routes to 0.0.0.0 do not propagate further than is intended. Generally, each autonomous system has its own preferred default router. Thus, routes involving 0.0.0.0 should generally not leave the boundary of an autonomous system. The mechanisms for enforcing this are not specified in this document.

3.8 Timers

This section describes all events that are triggered by timers.

Every 30 seconds, the RIP process is awakened to send an unsolicited Response message containing the complete routing table (see section 3.9 on Split Horizon) to every neighboring router. When there are many routers on a single network, there is a tendency for them to synchronize with each other such that they all issue updates at the same time. This can happen whenever the 30 second timer is affected by the processing load on the system. It is undesirable for the update messages to become synchronized, since it can lead to unnecessary collisions on broadcast networks. Therefore, implementations are required to take one of two precautions:

- The 30-second updates are triggered by a clock whose rate is not affected by system load or the time required to service the previous update timer.

- The 30-second timer is offset by a small random time (+/- 0 to 5 seconds) each time it is set. (Implementors may wish to consider even larger variation in the light of recent research results [10])

There are two timers associated with each route, a "timeout" and a "garbage-collection" time. Upon expiration of the timeout, the route is no longer valid; however, it is retained in the routing table for a short time so that neighbors can be notified that the route has been dropped. Upon expiration of the garbage-collection timer, the route is finally removed from the routing table.

The timeout is initialized when a route is established, and any time an update message is received for the route. If 180 seconds elapse from the last time the timeout was initialized, the route is considered to have expired, and the deletion process described below begins for that route.

Deletions can occur for one of two reasons: the timeout expires, or the metric is set to 16 because of an update received from the current router (see section 3.7.2 for a discussion of processing updates from other routers). In either case, the following events happen:

- The garbage-collection timer is set for 120 seconds.

- The metric for the route is set to 16 (infinity). This causes the route to be removed from service.

- The route change flag is set to indicate that this entry has been changed.

- The output process is signalled to trigger a response.

Until the garbage-collection timer expires, the route is included in all updates sent by this router. When the garbage-collection timer expires, the route is deleted from the routing table.

Should a new route to this network be established while the garbage- collection timer is running, the new route will replace the one that is about to be deleted. In this case the garbage-collection timer must be cleared.

Triggered updates also use a small timer; however, this is best described in section 3.9.1.

3.9 Input Processing

This section will describe the handling of datagrams received on the RIP port. Processing will depend upon the value in the command field.

See sections 4.6 and 5.1 for details on handling version numbers.

3.9.1 Request Messages

A Request is used to ask for a response containing all or part of a router's routing table. Normally, Requests are sent as broadcasts (multicasts for RIP-2), from the RIP port, by routers which have just come up and are seeking to fill in their routing tables as quickly as possible. However, there may be situations (e.g., router monitoring) where the routing table of only a single router is needed. In this case, the Request should be sent directly to that router from a UDP port other than the RIP port. If such a Request is received, the router responds directly to the requestor's address and port.

The Request is processed entry by entry. If there are no entries, no response is given. There is one special case. If there is exactly one entry in the request, and it has an address family identifier of zero and a metric of infinity (i.e., 16), then this is a request to send the entire routing table. In that case, a call is made to the output process to send the routing table to the requesting address/port. Except for this special case, processing is quite simple. Examine the list of RTEs in the Request one by one. For each entry, look up the destination in the router's routing database and, if there is a route, put that route's metric in the metric field of the RTE. If there is no explicit route to the specified destination, put infinity in the metric field. Once all the entries have been filled in, change the command from Request to Response and send the datagram back to the requestor.

Note that there is a difference in metric handling for specific and whole-table requests. If the request is for a complete routing table, normal output processing is done, including Split Horizon (see section 3.9 on Split Horizon). If the request is for specific entries, they are looked up in the

routing table and the information is returned as is; no Split Horizon processing is done. The reason for this distinction is the expectation that these requests are likely to be used for different purposes. When a router first comes up, it multicasts a Request on every connected network asking for a complete routing table. It is assumed that these complete routing tables are to be used to update the requestor's routing table. For this reason, Split Horizon must be done. It is further assumed that a Request for specific networks is made only by diagnostic software, and is not used for routing. In this case, the requester would want to know the exact contents of the routing table and would not want any information hidden or modified.

3.9.2 Response Messages

A Response can be received for one of several different reasons:

- response to a specific query
- regular update (unsolicited response)
- triggered update caused by a route change

Processing is the same no matter why the Response was generated.

Because processing of a Response may update the router's routing table, the Response must be checked carefully for validity. The Response must be ignored if it is not from the RIP port. The datagram's IPv4 source address should be checked to see whether the datagram is from a valid neighbor; the source of the datagram must be on a directly-connected network. It is also worth checking to see whether the response is from one of the router's own addresses. Interfaces on broadcast networks may receive copies of their own broadcasts/multicasts immediately. If a router processes its own output as new input, confusion is likely so such datagrams must be ignored. Once the datagram as a whole has been validated, process the RTEs in the Response one by one. Again, start by doing validation. Incorrect metrics and other format errors usually indicate misbehaving neighbors and should probably be brought to the administrator's attention. For example, if the metric is greater than infinity, ignore the entry but log the event. The basic validation tests are:

- is the destination address valid (e.g., unicast; not net 0 or 127)
- is the metric valid (i.e., between 1 and 16, inclusive)

If any check fails, ignore that entry and proceed to the next. Again, logging the error is probably a good idea.

Once the entry has been validated, update the metric by adding the cost of the network on which the message arrived. If the result is greater than infinity, use infinity. That is,

metric = MIN (metric + cost, infinity)

Now, check to see whether there is already an explicit route for the destination address. If there is no such route, add this route to the routing table, unless the metric is infinity (there is no point in adding a route which is unusable). Adding a route to the routing table consists of:

- Setting the destination address to the destination address in the RTE

- Setting the metric to the newly calculated metric (as described above)

- Set the next hop address to be the address of the router from which the datagram came

- Initialize the timeout for the route. If the garbage-collection timer is running for this route, stop it (see section 3.6 for a discussion of the timers)

- Set the route change flag

- Signal the output process to trigger an update (see section 3.8.1)

If there is an existing route, compare the next hop address to the address of the router from which the datagram came. If this datagram is from the same router as the existing route, reinitialize the timeout. Next, compare the metrics. If the datagram is from the same router as the existing route, and the new metric is different than the old one; or, if the new metric is lower than the old one; do the following actions:

- Adopt the route from the datagram (i.e., put the new metric in and adjust the next hop address, if necessary).

- Set the route change flag and signal the output process to trigger an update

- If the new metric is infinity, start the deletion process (described above); otherwise, re-initialize the timeout

If the new metric is infinity, the deletion process begins for the route, which is no longer used for routing packets. Note that the deletion process is started only when the metric is first set to infinity. If the metric was already infinity, then a new deletion process is not started.

If the new metric is the same as the old one, it is simplest to do nothing further (beyond re-initializing the timeout, as specified above); but, there is a heuristic which could be applied. Normally, it is senseless to replace a route if the new route has the same metric as the existing route; this would cause the route to bounce back and forth, which would generate an intolerable number of triggered updates. However, if the existing route is showing signs of timing out, it may be better to switch to an equally-good alternative route immediately, rather than waiting for the timeout to happen. Therefore, if the new metric is the same as the old one, examine the timeout for the existing route. If it is at least halfway to the expiration point, switch to the new route. This heuristic is optional, but highly recommended.

Any entry that fails these tests is ignored, as it is no better than the current route.

3.10 Output Processing

This section describes the processing used to create response messages that contain all or part of the routing table. This processing may be triggered in any of the following ways:

- By input processing, when a Request is received (this Response is unicast to the requestor; see section 3.7.1)

- By the regular routing update (broadcast/multicast every 30 seconds) router.

- By triggered updates (broadcast/multicast when a route changes)

When a Response is to be sent to all neighbors (i.e., a regular or triggered update), a Response message is directed to the router at the far end of each connected point-to-point link, and is broadcast (multicast for RIP-2) on all connected networks which support broadcasting. Thus, one Response is prepared for each directly- connected network, and sent to the appropriate address (direct or broadcast/multicast). In most cases, this reaches all neighboring routers. However, there are some cases where this may not be good enough. This may involve a network that is not a broadcast network (e.g., the ARPANET), or a situation involving dumb routers. In such cases, it may be necessary to specify an actual list of neighboring routers and send a datagram to each one explicitly. It is left to the implementor to determine whether such a mechanism is needed, and to define how the list is specified.

3.10.1 Triggered Updates

Triggered updates require special handling for two reasons. First, experience shows that triggered updates can cause excessive load on networks with limited capacity or networks with many routers on them. Therefore, the protocol requires that implementors include provisions to limit the frequency of triggered updates. After a triggered update is sent, a timer should be set for a random interval between 1 and 5 seconds. If other changes that would trigger updates occur before the timer expires, a single update is triggered when the timer expires. The timer is then reset to another random value between 1 and 5 seconds. A triggered update should be suppressed if a regular update is due by the time the triggered update would be sent.

Second, triggered updates do not need to include the entire routing table. In principle, only those routes which have changed need to be included. Therefore, messages generated as part of a triggered update must include at least those routes that have their route change flag set. They may include additional routes, at the discretion of the implementor; however, sending complete routing updates is strongly discouraged. When a triggered update is processed, messages should be generated for every directly-connected network. Split Horizon processing is done when generating triggered updates as well as normal updates (see section 3.9). If, after Split Horizon processing for a given network, a changed route will appear unchanged on that network (e.g., it appears with an infinite metric), the route need not be sent. If no routes need be sent on that network, the update may be omitted. Once all of the triggered updates have been generated, the route change flags should be cleared. If input processing is allowed while output is being generated, appropriate interlocking must be done. The route change flags should not be changed as a result of processing input while a triggered update message is being generated.

The only difference between a triggered update and other update messages is the possible omission of routes that have not changed. The remaining mechanisms, described in the next section, must be applied to all updates.

3.10.2 Generating Response Messages

This section describes how a Response message is generated for a particular directly-connected network:

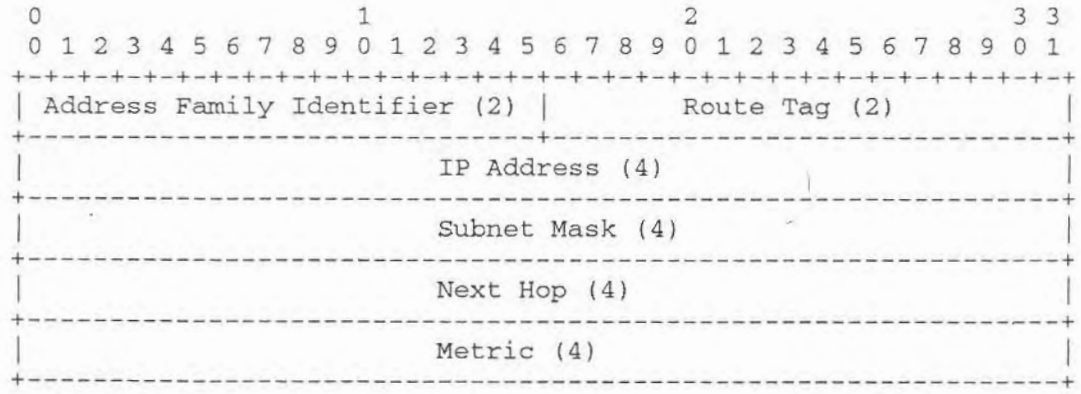
Set the version number to either 1 or 2. The mechanism for deciding which version to send is implementation specific; however, if this is the Response to a Request, the Response version should match the Request version. Set the command to Response. Set the bytes labeled "must be zero" to zero. Start filling in RTEs. Recall that there is a limit of 25 RTEs to a Response; if there are more, send the current Response and start a new one. There is no defined limit to the number of datagrams which make up a Response.

To fill in the RTEs, examine each route in the routing table. If a triggered update is being generated, only entries whose route change flags are set need be included. If, after Split Horizon processing, the route should not be included, skip it. If the route is to be included, then the destination address and metric are put into the RTE. Routes must be included in the datagram even if their metrics are infinite.

4. Protocol Extensions

This section does not change the RIP protocol per se. Rather, it provides extensions to the message format which allows routers to share important additional information.

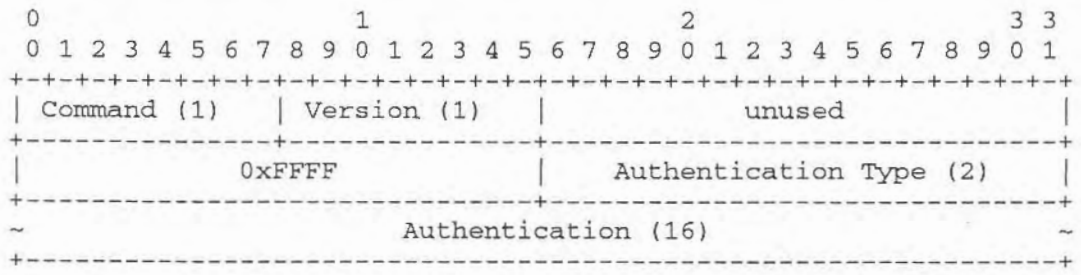
The same header format is used for RIP-1 and RIP-2 messages (see section 3.4). The format for the 20-octet route entry (RTE) for RIP-2 is:



The Address Family Identifier, IP Address, and Metric all have the meanings defined in section 3.4. The Version field will specify version number 2 for RIP messages which use authentication or carry information in any of the newly defined fields.

4.1 Authentication

Since authentication is a per message function, and since there is only one 2-octet field available in the message header, and since any reasonable authentication scheme will require more than two octets, the authentication scheme for RIP version 2 will use the space of an entire RIP entry. If the Address Family Identifier of the first (and only the first) entry in the message is 0xFFFF, then the remainder of the entry contains the authentication. This means that there can be, at most, 24 RIP entries in the remainder of the message. If authentication is not in use, then no entries in the message should have an Address Family Identifier of 0xFFFF. A RIP message which contains an authentication entry would begin with the following format:



Currently, the only Authentication Type is simple password and it is type 2. The remaining 16 octets contain the plain text password. If the password is under 16 octets, it must be left-justified and padded to the right with nulls (0x00).

4.2 Route Tag

The Route Tag (RT) field is an attribute assigned to a route which must be preserved and readvertised with a route. The intended use of the Route Tag is to provide a method of separating "internal" RIP routes (routes for networks within the RIP routing domain) from "external" RIP routes, which may have been imported from an EGP or another IGP.

Routers supporting protocols other than RIP should be configurable to allow the Route Tag to be configured for routes imported from different sources. For example, routes imported from EGP or BGP should be able to have their Route Tag either set to an arbitrary value, or at least to the number of the Autonomous System from which the routes were learned.

Other uses of the Route Tag are valid, as long as all routers in the RIP domain use it consistently. This allows for the possibility of a BGP-RIP protocol interactions document, which would describe methods for synchronizing routing in a transit network.

4.3 Subnet mask

The Subnet Mask field contains the subnet mask which is applied to the IP address to yield the non-host portion of the address. If this field is zero, then no subnet mask has been included for this entry.

On an interface where a RIP-1 router may hear and operate on the information in a RIP-2 routing entry the following rules apply:

1) information internal to one network must never be advertised into another network,
2) information about a more specific subnet may not be advertised where RIP-1 routers would consider it a host route, and
3) supernet routes (routes with a netmask less specific than the "natural" network mask) must not be advertised where they could be misinterpreted by RIP-1 routers.

4.4 Next Hop

The immediate next hop IP address to which packets to the destination specified by this route entry should be forwarded. Specifying a value of 0.0.0.0 in this field indicates that routing should be via the originator of the RIP advertisement. An address specified as a next hop must, per force, be directly reachable on the logical subnet over which the advertisement is made.

The purpose of the Next Hop field is to eliminate packets being routed through extra hops in the system. It is particularly useful when RIP is not being run on all of the routers on a network. A simple example is given in Appendix A. Note that Next Hop is an "advisory" field. That is, if the provided information is ignored, a possibly sub-optimal, but absolutely valid, route may be taken. If the received Next Hop is not directly reachable, it should be treated as 0.0.0.0.

4.5 Multicasting

In order to reduce unnecessary load on those hosts which are not listening to RIP-2 messages, an IP multicast address will be used for periodic broadcasts. The IP multicast address is 224.0.0.9. Note that IGMP is not needed since these are inter-router messages which are not forwarded.

On NBMA networks, unicast addressing may be used. However, if a response addressed to the RIP-2 multicast address is received, it should be accepted.

In order to maintain backwards compatibility, the use of the multicast address will be configurable, as described in section 5.1. If multicasting is used, it should be used on all interfaces which support it.

4.6 Queries

If a RIP-2 router receives a RIP-1 Request, it should respond with a RIP-1 Response. If the router is configured to send only RIP-2 messages, it should not respond to a RIP-1 Request.

5. Compatibility

RFC [1] showed considerable forethought in its specification of the handling of version numbers. It specifies that RIP messages of version 0 are to be discarded, that RIP messages of version 1 are to be discarded if any Must Be Zero (MBZ) field is non-zero, and that RIP messages of any version greater than 1 should not be discarded simply because an MBZ field contains a value other than

zero. This means that the new version of RIP is totally backwards compatible with existing RIP implementations which adhere to this part of the specification.

5.1 Compatibility Switch

A compatibility switch is necessary for two reasons. First, there are implementations of RIP-1 in the field which do not follow RFC [1] as described above. Second, the use of multicasting would prevent RIP-1 systems from receiving RIP-2 updates (which may be a desired feature in some cases). This switch should be configurable on a per-interface basis.

The switch has four settings: RIP-1, in which only RIP-1 messages are sent; RIP-1 compatibility, in which RIP-2 messages are broadcast; RIP-2, in which RIP-2 messages are multicast; and "none", which disables the sending of RIP messages. It is recommended that the default setting be either RIP-1 or RIP-2, but not RIP-1 compatibility. This is because of the potential problems which can occur on some topologies. RIP-1 compatibility should only be used when all of the consequences of its use are well understood by the network administrator.

For completeness, routers should also implement a receive control switch which would determine whether to accept, RIP-1 only, RIP-2 only, both, or none. It should also be configurable on a per-interface basis. It is recommended that the default be compatible with the default chosen for sending updates.

5.2 Authentication

The following algorithm should be used to authenticate a RIP message. If the router is not configured to authenticate RIP-2 messages, then RIP-1 and unauthenticated RIP-2 messages will be accepted; authenticated RIP-2 messages shall be discarded. If the router is configured to authenticate RIP-2 messages, then RIP-1 messages and RIP-2 messages which pass authentication testing shall be accepted; unauthenticated and failed authentication RIP-2 messages shall be discarded. For maximum security, RIP-1 messages should be ignored when authentication is in use (see section 4.1); otherwise, the routing information from authenticated messages will be propagated by RIP-1 routers in an unauthenticated manner.

Since an authentication entry is marked with an Address Family Identifier of 0xFFFF, a RIP-1 system would ignore this entry since it would belong to an address family other than IP. It should be noted, therefore, that use of authentication will not prevent RIP-1 systems from seeing RIP-2 messages. If desired, this may be done using multicasting, as described in sections 4.5 and 5.1.

5.3 Larger Infinity

While on the subject of compatibility, there is one item which people have requested: increasing infinity. The primary reason that this cannot be done is that it would violate backwards compatibility. A larger infinity would obviously confuse older versions of rip. At best, they would ignore the route as they would ignore a metric of 16. There was also a proposal to make the Metric a single octet and reuse the high three octets, but this would break any implementations which treat the metric as a 4-octet entity.

5.4 Addressless Links

As in RIP-1, addressless links will not be supported by RIP-2.

6. Interaction between version 1 and version 2

Because version 1 packets do not contain subnet information, the semantics employed by routers on networks that contain both version 1 and version 2 networks should be limited to that of version

1. Otherwise it is possible either to create blackhole routes (i.e., routes for networks that do not exist) or to create excessive routing information in a version 1 environment.

Some implementations attempt to automatically summarize groups of adjacent routes into single entries, the goal being to reduce the total number of entries. This is called auto-summarization.

Specifically, when using both version 1 and version 2 within a network, a single subnet mask should be used throughout the network. In addition, auto-summarization mechanisms should be disabled for such networks, and implementations must provide mechanisms to disable auto-summarization.

7. Security Considerations

The basic RIP protocol is not a secure protocol. To bring RIP-2 in line with more modern routing protocols, an extensible authentication mechanism has been incorporated into the protocol enhancements. This mechanism is described in sections 4.1 and 5.2. Security is further enhanced by the mechanism described in [3].

Appendix A

This is a simple example of the use of the next hop field in a rip entry.

Assume that IR1, IR2, and IR3 are all "internal" routers which are under one administration (e.g. a campus) which has elected to use RIP-2 as its IGP. XR1, XR2, and XR3, on the other hand, are under separate administration (e.g. a regional network, of which the campus is a member) and are using some other routing protocol (e.g. OSPF). XR1, XR2, and XR3 exchange routing information among themselves such that they know that the best routes to networks N1 and N2 are via XR1, to N3, N4, and N5 are via XR2, and to N6 and N7 are via XR3. By setting the Next Hop field correctly (to XR2 for N3/N4/N5, to XR3 for N6/N7), only XR1 need exchange RIP-2 routes with IR1/IR2/IR3 for routing to occur without additional hops through XR1. Without the Next Hop (for example, if RIP-1 were used) it would be necessary for XR2 and XR3 to also participate in the RIP-2 protocol to eliminate extra hops.

References

[1] Hedrick, C., "Routing Information Protocol", STD 34, RFC 1058, Rutgers University, June 1988.
[2] Malkin, G., and F. Baker, "RIP Version 2 MIB Extension", RFC 1389, January 1993.
[3] Baker, F., and R. Atkinson, "RIP-II MD5 Authentication", RFC 2082, January 1997.
[4] Bellman, R. E., "Dynamic Programming", Princeton University Press, Princeton, N.J., 1957.
[5] Bertsekas, D. P., and Gallaher, R. G., "Data Networks", Prentice-Hall, Englewood Cliffs, N.J., 1987.
[6] Braden, R., and Postel, J., "Requirements for Internet Gateways", STD 4, RFC 1009, June 1987.
[7] Boggs, D. R., Shoch, J. F., Taft, E. A., and Metcalfe, R. M., "Pup: An Internetwork Architecture", IEEE Transactions on Communications, April 1980.
[8] Ford, L. R. Jr., and Fulkerson, D. R., "Flows in Networks", Princeton University Press, Princeton, N.J., 1962.
[9] Xerox Corp., "Internet Transport Protocols", Xerox System Integration Standard XSIS 028112, December 1981.

[10] Floyd, S., and V. Jacobson, "The synchronization of Periodic Routing Messages," ACM Sigcom '93 symposium, September 1993.
[11] Baker, F., "Requirements for IP Version 4 Routers." RFC 1812, June 1995.

Author's Address Gary Scott Malkin Bay Networks 8 Federal Street Billerica, MA 01821 Phone: (978) 916-4237 EMail: gmalkin@baynetworks.com

Bibliografía

Bibliografía

[1] Request for Comments 1160. NRI. May 1990.

[2] Comer, D.E. 1996. Redes Globales de Información con Internet y TCP/IP Principios básicos, protocolos y Arquitectura. 3ª. Ed. Prentice-Hall, Hispanoamericana, S.A.

[3] Request for Comments 1700. Network Working Group. J. Reynolds y J. Postel. STD 2. ISI. October 1994. Category Standards Track.

[4] Request for Comments 1166. Network Working Group. S. Kirkpatrick, M. Stahl y M. Recker. SRI-NIC. July 1990.

[5] Request For Comments 826. Network Working Group. David C. Plummer. (DCP@MIT-MC). November 1982.

[6] Request For Comments 814. NAME, ADDRESSES, PORTS, AND ROUTES. David D. Clark. MIT Laboratory for Computer Science. Computer Systems and Communications Group. July, 1982.

[7] Request for Comments 903. Network Working Group. A Reverse Address Resolution Protocol. Finlayson R., Mann T., Mogul J. y Theimer M. Computer Science Department Stanford University. June 1984

[8] Request for Comments 1542. Network Working Group. Clarifications and Extensions for the Bootstrap Protocol. W. Wimer. Carnegie Mellon University. October 1993. Category Standards Track.

[9] Request for Comments 2131. Network Working Group. Dynamic Host Configuration Protocol. R. Droms. Bucknell University. March 1997. Category Standards Track.

[10] Request for Comments 791. Internet Protocol, Darpa Internet Program, Protocol Specification. Information Sciences Institute University of Southern California for Defense Advanced Research Projects Agency, Information Processing Techniques Office. September 1981.

[11] Request for Comments 792. Network Working Group. Internet Control Message Protocol, Darpa Internet Program Protocol Specification. J. Postel. ISI. September 1981.

[12] Request for Comments 1122. Network Working Group. Requirements for Internet Hosts -- Communication Layers. Braden, R. Internet Engineering Task Force. October 1989.

[13] Cisco Internetworking Library. Cisco Systems, Inc. Copyright 1994.

[14] Request for Comments 768. User Datagram Protocol. Postel , J. ISI. August 1980.

[15] Request for Comments 793. Transmission Control Protocol, Darpa Internet Program, Protocol Specification. Information Sciences Institute University of Southern California for Defense Advanced Research Projects Agency Information Processing Techniques Office. September 1981.

[16] Request for Comments 1812. Network Working Group. Requirements for IP Version 4 Routers. Baker, F. Cisco Systems. June 1995. Category Standards Track.

[17] Request for Comments 950. Network Working Group. Internet Standard Subnetting Procedure. Mogul, J. (Stanford), Postel , J. (ISI). August 1985.

[18] Windows Sockets. An Open Interface for Network Programming under Microsoft Windows Version 1.1. Hall, M., Towfiq, M., Arnold, G., Treadwell, D. y Sanders, H. Copyright January 1993.

[19] OWLSock Winsock class library for Borland's OWL (Object Windows Library) Documentation Version 1.0. Pedriana, P. Copyright May 1995.