UNIVERSIDAD NACIONAL DEL LITORAL

FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

REDES Y COMUNICACIONES DE DATOS II

Resumen de Parcial

Autores:
Mario ROSALES
Franco SANTELLÁN
Carlos GENTILE

E-mail:
mariorosales941@gmail.com
franco.santellan@hotmail.com
csgentile@gmail.com

MAYO, 2016

Índice

PRIMER PARCIAL: LA CAPA DE RED

La capa de red se encarga de llevar los paquetes desde el origen hasta el destino. Llegar al destino puede requerir muchos saltos por enrutadores intermedios. Para lograr su cometido, la capa de red debe conocer la topología de la subred de comunicación (es decir, el grupo de enrutadores) y elegir las rutas adecuadas a través de ella; también debe tener cuidado al escoger las rutas para no sobrecargar algunas de las líneas de comunicación y los enrutadores y dejar inactivos a otros. Por último, cuando el origen y el destino están en redes diferentes, ocurren nuevos problemas. La capa de red es la encargada de solucionarlos.

1. Aspectos de Diseño de la Capa de Red

1.1. Conmutación de paquetes de almacenamiento y reenvío

Un *host* transmite al enrutador más cercano un paquete que tiene por enviar, ya sea en su propia LAN o a través de un enlace punto a punto con la empresa portadora. El paquete se almacena ahí hasta que haya llegado por completo, a fin de que la suma de verificación pueda comprobarse. Después se reenvía al siguiente enrutador de la ruta hasta que llegue al *host* de destino, donde se entrega.

1.2. Servicios proporcionados a la capa de transporte

La capa de red proporciona servicios a la capa de transporte en la interfaz capa de red/capa de transporte. Una pregunta importante es qué tipo de servicios proporciona la capa de red a la capa de transporte. Los servicios de la capa de red se han diseñado con los siguientes objetivos en mente:

- Los servicios deben ser independientes de la tecnología del enrutador.
- La capa de transporte debe estar aislada de la cantidad, tipo y topología de los enrutadores presentes.
- Las direcciones de red disponibles para la capa de transporte deben seguir un plan de numeración uniforme, aun a través de varias LANs y WANs.

La discusión se centra en determinar si la capa de red debe proporcionar **servicio orientado** o **no orientado** a la conexión.

Un bando (representado por la comunidad de Internet) alega que la tarea del enrutador es mover bits de un lado a otro, y nada más. Desde su punto de vista, la subred es inherentemente inestable, sin importar su diseño. Por lo tanto, los hosts deben aceptar este hecho y efectuar ellos mismos el control de errores (es decir, detección y corrección de errores) y el control de flujo. Este punto de vista conduce directamente a la conclusión de que el servicio de red no debe ser orientado a la conexión. En particular, no debe efectuarse ningún ordenamiento de paquetes ni control de flujo, pues de todos modos los hosts lo van a efectuar y probablemente se ganaría poco haciéndolo dos veces. Además, cada paquete debe llevar la dirección de destino completa, porque cada paquete enviado se transporta de manera independiente de sus antecesores, si los hay.

El otro bando (representado por las compañías telefónicas) argumenta que la subred debe proporcionar un servicio confiable, orientado a la conexión. Desde este punto de vista, la calidad del servicio es el factor dominante, y sin conexiones en la subred, tal calidad es muy difícil de alcanzar, especialmente para el tráfico de tiempo real como la voz y el vídeo.

Si se ofrece el servicio no orientado a la conexión, los paquetes se colocan individualmente en la subred y se enrutan de manera independiente. No se necesita una configuración avanzada. En este contexto, por lo general los paquetes se conocen como **datagramas** y la subred se conoce como **subred de datagramas**. Si se utiliza el servicio orientado a la conexión, antes de poder enviar cualquier paquete de datos, es necesario establecer una ruta del enrutador de origen al de destino. Esta conexión se conoce como **CV (circuito virtual)** y la subred se conoce como **subred de circuitos virtuales**.

1.3. Implementación del servicio no orientado a la conexión

A continuación veamos cómo funciona una subred de datagramas. Suponga que el proceso *P1* de la *figura* 1 tiene un mensaje largo para *P2*. Supongamos que el mensaje es cuatro veces más largo que el tamaño máximo de paquete, por lo que la capa de red tiene que dividirlo en cuatro paquetes, 1, 2, 3 y 4, y envía cada uno de ellos a la vez al enrutador A mediante algún protocolo punto a punto.

Cada enrutador tiene una tabla interna que le indica a dónde enviar paquetes para cada destino posible. Cada entrada de tabla es un par que consiste en un destino y la línea de salida que se utilizará para ese destino. Sólo se pueden utilizar líneas

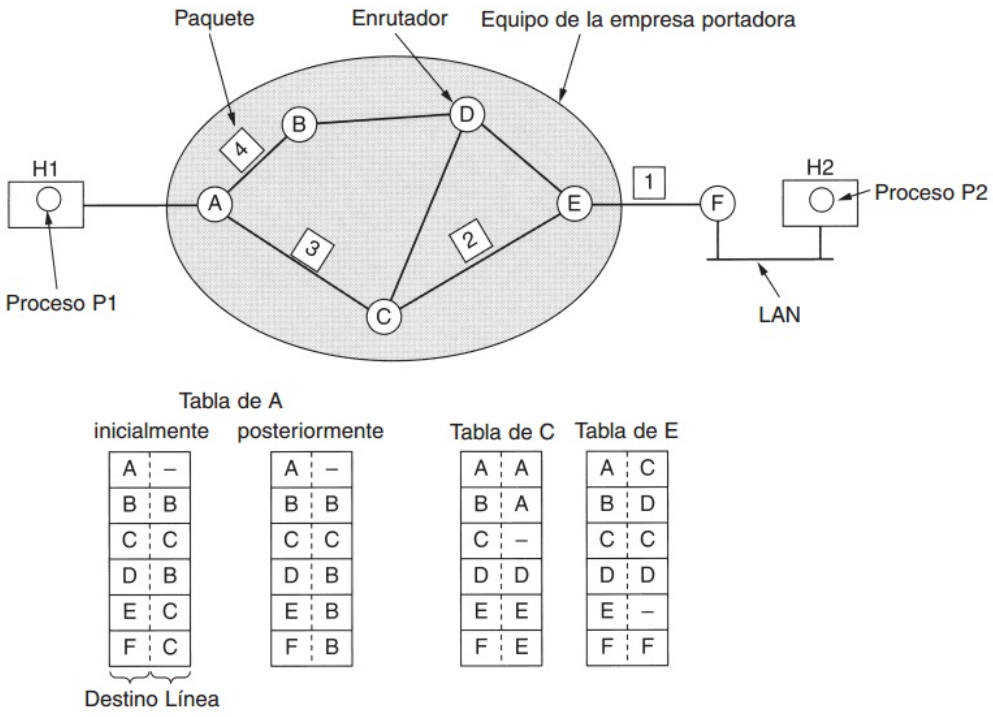


Figura 1: Enrutamiento dentro de una subred de datagramas.

conectadas directamente. Por ejemplo, en la *figura* 1, A sólo tiene dos líneas de salida —a B y C—, por lo que cada paquete entrante debe enviarse a uno de estos enrutadores, incluso si el destino final es algún otro enrutador. En la figura, la tabla de enrutamiento inicial de A se muestra abajo de la leyenda ”inicialmente”.

Conforme los paquetes 1, 2 y 3 llegaron a A, se almacenaron unos momentos (para comprobar sus sumas de verificación). Después cada uno se reenvió a C de acuerdo con la tabla de A. Posteriormente, el paquete 1 se reenvió a E y después a F. Cuando llegó a F, se encapsuló en una trama de capa de enlace de datos y se envió a H2 a través de la LAN. Los paquetes 2 y 3 siguieron la misma ruta. Sin embargo, pasó algo diferente con el paquete 4. Cuando llegó a A, se envió al enrutador B, aunque también estaba destinado a F. Por alguna razón, A decidió enviar el paquete 4 por una ruta diferente a la de los primeros tres paquetes.

Tal vez se enteró de que había alguna congestión de tráfico en alguna parte de la ruta ACE y actualizó su tabla de enrutamiento, como se muestra bajo la leyenda ”posteriormente”. El algoritmo que maneja las tablas y que realiza las decisiones de enrutamiento se conoce como **algoritmo de enrutamiento**.

1.4. Implementación del servicio orientado a la conexión

Para servicio orientado a la conexión necesitamos una subred de circuitos virtuales. Veamos cómo funciona. El propósito de los circuitos virtuales es evitar la necesidad de elegir una nueva ruta para cada paquete enviado, como en la *figura* 1. En su lugar, cuando se establece una conexión, se elige una ruta de la máquina de origen a la de destino como parte de la configuración de conexión y se almacena en tablas dentro de los enrutadores. Esa ruta se utiliza para todo el tráfico que fluye a través de la conexión, exactamente de la misma forma en que funciona el sistema telefónico. Cuando se libera la conexión, también se termina el circuito virtual. Con el servicio orientado a la conexión, cada paquete lleva un identificador que indica a cuál circuito virtual pertenece.

Considerando la situación que se muestra en la *figura* 2. En ésta, el host *H1* ha establecido una conexión 1 con el host *H2*. Se recuerda como la primera entrada de cada una de las tablas de enrutamiento. La primera línea de la tabla A indica que si un paquete tiene el identificador de conexión 1 viene de *H1*, se enviará al enrutador C y se le dará el identificador de conexión 1. De manera similar, la primera entrada en C enruta el paquete a E, también con el identificador de conexión 1.

Ahora consideremos lo que sucede si *H3* también desea establecer una conexión con *H2*. Elige el identificador de conexión 1 (debido a que está iniciando la conexión y a que ésta es su única conexión) y le indica a la subred que establezca el circuito virtual. Esto nos lleva a la segunda fila de las tablas. Observe que aquí surge un problema debido a que aunque A sí puede saber con facilidad cuáles paquetes de conexión 1 provienen de *H1* y cuáles provienen de *H3*, C no puede hacerlo. Por esta

razón, A asigna un identificador de conexión diferente al tráfico saliente para la segunda conexión. Con el propósito de evitar conflictos de este tipo, los enrutadores requieren la capacidad de reemplazar identificadores de conexión en los paquetes salientes. En algunos contextos a esto se le conoce como **conmutación de etiquetas**.

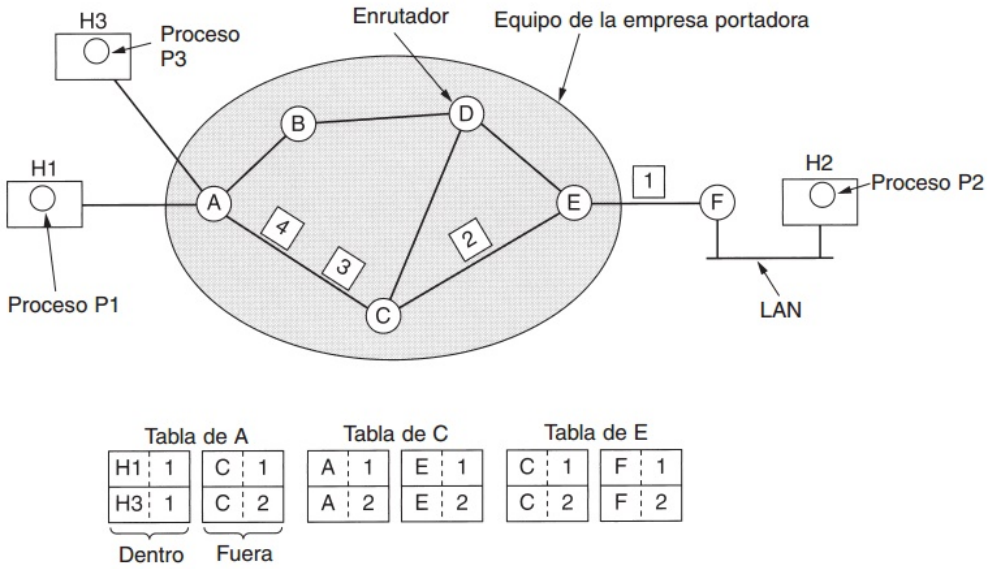


Figura 2: Enrutamiento dentro de una subred de circuitos virtuales.

1.5. Comparación entre las subredes de circuitos virtuales y las de datagramas

Dentro de la subred hay varios pros y contras entre los circuitos virtuales y los datagramas. Uno de ellos tiene que ver con el espacio de memoria del enrutador y el ancho de banda. Los circuitos virtuales permiten que los paquetes contengan números de circuito en lugar de direcciones de destino completas. Si los paquetes suelen ser bastante cortos, una dirección de destino completa en cada paquete puede representar una sobrecarga significativa y, por lo tanto, ancho de banda desperdiciado. El precio que se paga por el uso interno de circuitos virtuales es el espacio de tabla en los enrutadores.

Otro punto por considerar es el del tiempo de configuración contra el tiempo de análisis de la dirección. El uso de circuitos virtuales requiere una fase de configuración, que consume tiempo y recursos. Sin embargo, determinar lo que hay que hacer con un paquete de datos en una subred de circuitos virtuales es fácil: el enrutador simplemente usa el número de circuito para buscar en una tabla y encontrar hacia dónde va el paquete. En una subred de datagramas se requiere un procedimiento más complicado para localizar el destino del paquete.

Otra cuestión es la cantidad requerida de espacio de tabla en la memoria del enrutador. Una subred de datagramas necesita tener una entrada para cada destino posible, mientras que una subred de circuitos virtuales sólo necesita una entrada por cada circuito virtual. Sin embargo, esta ventaja es engañosa debido a que los paquetes de configuración de conexión también tienen que enrutarse, y a que utilizan direcciones de destino, de la misma forma en que lo hacen los datagramas.

Los circuitos virtuales tienen algunas ventajas en cuanto a la calidad del servicio ya que evitan congestiones en la subred, pues los recursos (por ejemplo, búferes, ancho de banda y ciclos de CPU) pueden reservarse por adelantado al establecerse la conexión. Una vez que comienzan a llegar los paquetes, estarán ahí el ancho de banda y la capacidad de enrutamiento necesarios. En una subred de datagramas es más difícil evitar las congestiones.

Los circuitos virtuales también tienen un problema de vulnerabilidad. Si se cae un enrutador y se pierde su memoria, todos los circuitos virtuales que pasan por él tendrán que abortarse,aunque se recupere un segundo después. Por el contrario, si se cae un enrutador de datagramas, sólo sufrirán los usuarios cuyos paquetes estaban encolados en el enrutador en el momento de la falla y, dependiendo de si ya se había confirmado o no su recepción, tal vez ni siquiera todos ellos. La pérdida de una línea de comunicación es fatal para los circuitos virtuales que la usan, pero puede compensarse fácilmente cuando se usan datagramas.

2. Algoritmos de Enrutamiento

La función principal de la capa de red es enrutar paquetes de la máquina de origen a la de destino. En la mayoría de las subredes, los paquetes requerirán varios saltos para completar el viaje. La única excepción importante son las redes de difusión, pero aun aquí es importante el enrutamiento si el origen y el destino no están en la misma red. Los algoritmos que eligen las rutas y las estructuras de datos que usan constituyen un aspecto principal del diseño de la capa de red.

El **algoritmo de enrutamiento** es aquella parte del software de la capa de red encargada de decidir la línea de salida por la que se transmitirá un paquete de entrada. Si la subred usa datagramas de manera interna, esta decisión debe tomarse cada vez que llega un paquete de datos, dado que la mejor ruta podría haber cambiado desde la última vez. Si la subred usa circuitos virtuales internamente, las decisiones de enrutamiento se toman sólo al establecerse un circuito virtual nuevo. En lo sucesivo, los paquetes de datos simplemente siguen la ruta previamente establecida. Este último caso a veces se llama enrutamiento de sesión, dado que una ruta permanece vigente durante toda la sesión de usuario.

Algunas veces es útil distinguir entre el **enrutamiento**, que es el proceso consistente en tomar la decisión de cuáles rutas utilizar, y el **reenvío**, que consiste en la acción que se toma cuando llega un paquete. Se puede considerar que un enrutador realiza **dos procesos** internos. Uno de ellos maneja cada paquete conforme llega, buscando en las tablas de enrutamiento la línea de salida por la cual se enviará. Este proceso se conoce como reenvío. El otro proceso es responsable de llenar y actualizar las tablas de enrutamiento. Es ahí donde entra en acción el algoritmo de enrutamiento.

Sin importar si las rutas para cada paquete se eligen de manera independiente o sólo cuando se establecen nuevas conexiones, hay ciertas propiedades que todo algoritmo de enrutamiento debe poseer:

- **exactitud**
- **sencillez**
- **robustez**
- **estabilidad**
- **equidad**
- **optimización**

La exactitud y la sencillez apenas requieren comentarios, pero la necesidad de robustez puede ser menos obvia a primera vista. Una vez que una red principal entra en operación, cabría esperar que funcionara continuamente durante años sin fallas a nivel de sistema. Durante ese periodo habrá fallas de hardware y de software de todo tipo. Los hosts, enrutadores y líneas fallarán en forma repetida y la topología cambiará muchas veces. El algoritmo de enrutamiento debe ser capaz de manejar los cambios de topología y tráfico sin requerir el aborto de todas las actividades en todos los hosts y el reinicio de la red con cada caída de un enrutador.

La estabilidad también es una meta importante del algoritmo de enrutamiento. Existen algoritmos de enrutamiento que nunca alcanzan el equilibrio, sin importar el tiempo que permanezcan operativos. Un algoritmo estable alcanza el equilibrio y lo conserva. La equidad y la optimización pueden parecer algo obvias, pero resulta que con frecuencia son metas contradictorias.

Antes de que podamos siquiera intentar encontrar el punto medio entre la equidad y la optimización, debemos decidir qué es lo que buscamos optimizar. Un candidato obvio es la minimización del retardo medio de los paquetes, pero también lo es el aumento al máximo de la velocidad real de transporte de la red. Además, estas dos metas también están en conflicto, ya que la operación de cualquier sistema de colas cerca de su capacidad máxima implica un retardo de encolamiento grande. Como término medio, muchas redes intentan minimizar el número de saltos que tiene que dar un paquete, puesto que la reducción de la cantidad de saltos reduce el retardo y también el consumo de ancho de banda, lo que a su vez mejora la velocidad real de transporte.

Los algoritmos de enrutamiento pueden agruparse en dos clases principales:

- **Algoritmos No Adaptativos (Enrutamiento Estático):** Los algoritmos no adaptativos no basan sus decisiones de enrutamiento en mediciones o estimaciones del tráfico y la topología actuales. En cambio, la decisión de qué ruta se usará para llegar de I a J se toma por adelantado, fuera de línea, y se carga en los enrutadores al arrancar la red. Este procedimiento se conoce como enrutamiento estático.
- **Algoritmos Adaptativos (Enrutamiento Dinámico):** Los algoritmos adaptativos cambian sus decisiones de enrutamiento para reflejar los cambios de topología y, por lo general también el tráfico. Los algoritmos adaptativos difieren en el lugar de donde obtienen su información (localmente), el momento de cambio de sus rutas (cada ΔT *seg*) y la métrica usada para la optimización (distancia o número de saltos).

2.1. Principio de optimización

Antes de entrar en algoritmos específicos, puede ser útil señalar que es posible hacer un postulado general sobre las rutas óptimas sin importar la topología o el tráfico de la red. Este postulado se conoce como principio de optimización, y establece que si el enrutador J está en ruta óptima del enrutador I al enrutador K, entonces la ruta óptima de J a K también está en la misma ruta.

Los criterios que se aplican para establecer la métrica de ruta óptima suelen ser:

- Minimizar:
    1. Número de saltos
    2. Congestión en los enlaces
    3. Retardo de los enlaces
- Maximizar:
    1. Ancho de Banda de los enlaces
    2. Fiabilidad de los enlaces (minimizar (BER)

Como consecuencia directa del principio de optimización, podemos ver que el grupo de rutas óptimas de todos los orígenes a un destino dado forman un árbol con raíz en el destino. Tal árbol se conoce como **árbol sumidero**, y se ilustra en la *figura* 3. Observe que un árbol sumidero no necesariamente es único; pueden existir otros árboles con las mismas longitudes de rutas. La meta de todos los algoritmos de enrutamiento es descubrir y utilizar los árboles sumideros de todos los enrutadores. Puesto que un árbol sumidero ciertamente es un árbol, no contiene ciclos, por lo que cada paquete será entregado en un número de saltos finito y limitado.

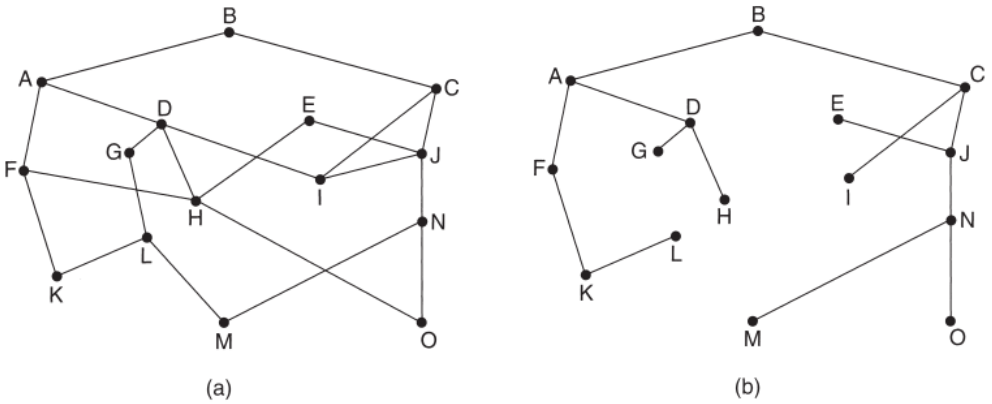


Figura 3: (a) Una subred. (b) Árbol sumidero para el enrutador B.

2.2. Enrutamiento por la ruta más corta

La idea es armar un grafo de la subred, en el que cada nodo representa un enrutador y cada arco del grafo una línea de comunicación (con frecuencia llamada enlace). Para elegir una ruta entre un par dado de enrutadores, el algoritmo simplemente encuentra en el grafo la ruta más corta entre ellos. El concepto de **ruta más corta** merece una explicación. Una manera de medir la longitud de una ruta es por la cantidad de saltos. Usando esta métrica, las rutas ABC y ABE de la *figura* 4 tienen la misma longitud. Otra métrica es la distancia geográfica en kilómetros, en cuyo caso ABC es claramente mucho mayor que ABE.

Sin embargo, también son posibles muchas otras métricas además de los saltos y la distancia física. Por ejemplo, cada arco podría etiquetarse con el retardo medio de encolamiento y transmisión de un paquete de prueba estándar, determinado por series de prueba cada hora. Con estas etiquetas en el grafo, la ruta más corta es la más rápida, en lugar de la ruta con menos arcos o kilómetros.

En el caso más general, las etiquetas de los arcos podrían calcularse como una función de la distancia, ancho de banda,

tráfico medio, costo de comunicación, longitud media de las colas, retardo medio y otros factores. Cambiando la función de ponderación, el algoritmo calcularía la ruta ”más corta” de acuerdo con cualquiera de varios criterios, o una combinación de ellos.

Se conocen varios algoritmos de cálculo de la ruta más corta entre dos nodos de un grafo. Éste se debe a *Dijkstra*. Cada nodo se etiqueta (entre paréntesis) con su distancia al nodo de origen a través de la mejor ruta conocida. Inicialmente no se conocen rutas, por lo que todos los nodos tienen la etiqueta infinito. A medida que avanza el algoritmo y se encuentran rutas, las etiquetas pueden cambiar, reflejando mejores rutas. Una etiqueta puede ser tentativa o permanente. Inicialmente todas las etiquetas son tentativas. Una vez que se descubre que una etiqueta representa la ruta más corta posible del origen a ese nodo, se vuelve permanente y no cambia más.

Para ilustrar el funcionamiento del algoritmo de etiquetado, observe el grafo ponderado no dirigido de la *figura* 4 *(a)*, donde las ponderaciones representan, por ejemplo, distancias. Queremos encontrar la ruta más corta posible de A a D. Comenzamos por marcar como permanente el nodo A, indicado por un círculo rellenado. Después examinamos, por turno, cada uno de los nodos adyacentes a A (el nodo de trabajo), reetiquetando cada uno con la distancia desde A. Cada vez que reetiquetamos un nodo, también lo reetiquetamos con el nodo desde el que se hizo la prueba, para poder reconstruir más tarde la ruta final. Una vez que terminamos de examinar cada uno de los nodos adyacentes a A, examinamos todos los nodos etiquetados tentativamente en el grafo completo y hacemos permanente el de la etiqueta más pequeña, como se muestra en la *figura* 4 *(b)*. Éste se convierte en el nuevo nodo de trabajo. Esto se repite hasta llegar al nodo D.

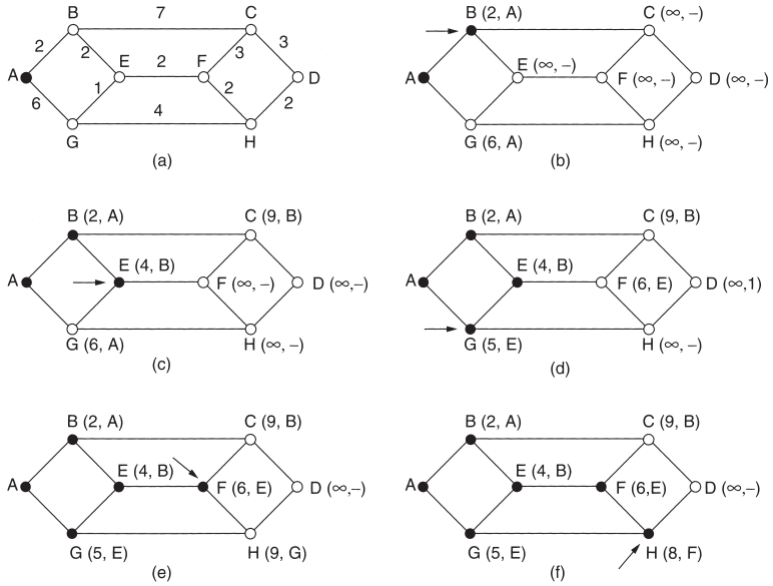


Figura 4: Los primeros cinco pasos del cálculo de la ruta más corta de A a D. Las flechas indican el nodo de trabajo.

2.3. Inundación

Otro algoritmo estático es la inundación, en la que cada paquete de entrada se envía por cada una de las líneas de salida, excepto aquella por la que llegó. La inundación evidentemente genera grandes cantidades de paquetes duplicados; de hecho, una cantidad infinita a menos que se tomen algunas medidas para limitar el proceso. Una de estas medidas es integrar un contador de saltos en el encabezado de cada paquete, que disminuya con cada salto, y el paquete se descarte cuando el contador llegue a cero. Lo ideal es inicializar el contador de saltos a la longitud de la ruta entre el origen y el destino. Si el emisor desconoce el tamaño de la ruta, puede inicializar el contador al peor caso, es decir, el diámetro total de la subred.

Una técnica alterna para ponerle diques a la inundación es llevar un registro de los paquetes difundidos, para evitar enviarlos una segunda vez. Una manera de lograr este propósito es hacer que el enrutador de origen ponga un número de secuencia en cada paquete que recibe de sus hosts. Cada enrutador necesita una lista por cada enrutador de origen que indique los números de secuencia originados en ese enrutador que ya ha visto. Si un paquete de entrada está en la lista, no se difunde.

Para evitar que la lista crezca sin límites, cada lista debe incluir un contador, *k*, que indique que todos los números de secuencia hasta *k* ya han sido vistos. Cuando llega un paquete, es fácil comprobar si es un duplicado; de ser así, se descarta. Es más, no se necesita la lista completa por debajo de *k*, pues *k* la resume efectivamente.

Una variación de la inundación, un poco más práctica, es la **inundación selectiva**. En este algoritmo, los enrutadores no

envían cada paquete de entrada por todas las líneas, sino sólo por aquellas que van aproximadamente en la dirección correcta. Por lo general, no tiene mucho caso enviar un paquete dirigido al oeste a través de una línea dirigida al este.

La inundación no es práctica en la mayoría de las aplicaciones, pero tiene algunos usos. Por ejemplo, en aplicaciones militares, donde grandes cantidades de enrutadores pueden volar en pedazos en cualquier momento, es altamente deseable la excelente robustez de la inundación.

2.4. Enrutamiento por vector de distancia

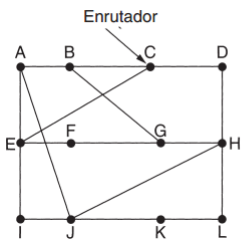
Las redes modernas de computadoras por lo general utilizan algoritmos de enrutamiento dinámico en lugar de los estáticos antes descritos, pues los algoritmos estáticos no toman en cuenta la carga actual de la red. Los algoritmos de enrutamiento por vector de distancia operan haciendo que cada enrutador mantenga una tabla (es decir, un vector) que da la mejor distancia conocida a cada destino y la línea que se puede usar para llegar ahí. Estas tablas se actualizan intercambiando información con los vecinos.

En el enrutamiento por vector de distancia, cada enrutador mantiene una tabla de enrutamiento indizada por, y conteniendo un registro de, cada enrutador de la subred. Esta entrada comprende dos partes: la línea preferida de salida hacia ese destino y una estimación del tiempo o distancia a ese destino. La métrica usada podría ser la cantidad de saltos, el retardo de tiempo en milisegundos, el número total de paquetes encolados a lo largo de la ruta, o algo parecido.

Se supone que el enrutador conoce la "distancia" a cada uno de sus vecinos. Si la métrica es de saltos, la distancia simplemente es un salto. Si la métrica es la longitud de la cola, el enrutador simplemente examina cada cola. Si la métrica es el retardo, el enrutador puede medirlo en forma directa con paquetes especiales de ECO que el receptor simplemente marca con la hora y lo regresa tan rápido como puede.

Supongamos que el retardo se usa como métrica y que el enrutador conoce el retardo a cada uno de sus vecinos. Una vez cada *T mseg*, cada enrutador envía a todos sus vecinos una lista de sus retardos estimados a cada destino. También recibe una lista parecida de cada vecino. Imagine que una de estas tablas acaba de llegar del vecino X, siendo X_i la estimación de X respecto al tiempo que le toma llegar al enrutador i. Si el enrutador sabe que el retardo a X es de m *mseg*, también sabe que puede llegar al enrutador i a través de X en $X_i + m$ *mseg*. Efectuando este cálculo para cada vecino, un enrutador puede encontrar la estimación que parezca ser la mejor y usar esa estimación, así como la línea correspondiente, en su nueva tabla de enrutamiento.

Este proceso de actualización se ilustra en la *figura* 5. En la parte *(a)* se muestra una subred. En las primeras cuatro columnas de la parte *(b)* aparecen los vectores de retardo recibidos de los vecinos del enrutador J. A indica tener un retardo de 12 *mseg* a B, un retardo de 25 *mseg* a C, un retardo de 40 *mseg* a D, etc. Suponga que J ha medido o estimado el retardo a sus vecinos A, I, H y K en 8, 10, 12 y 6 *mseg*, respectivamente.



(a)

	A	I	H	K	Nuevo retardo estimado desde J	Línea
A	0	24	20	21	8	A
B	12	36	31	28	20	A
C	25	18	19	36	28	I
D	40	27	8	24	20	H
E	14	7	30	22	17	I
F	23	20	19	40	30	I
G	18	31	6	31	18	H
H	17	20	0	19	12	H
I	21	0	14	22	10	I
J	9	11	7	10	0	–
K	24	22	22	0	6	K
L	29	33	9	9	15	K

Retardo JA es de 8 | Retardo JI es de 10 | Retardo JH es de 12 | Retardo JK es de 6

Vectores recibidos de los cuatro vecinos de J

Nueva tabla de enrutamiento para J

(b)

Figura 5: (a) Subred. (b) Entrada de A, I, H, K y la nueva tabla de enrutamiento de J.

Considere la manera en que J calcula su nueva ruta al enrutador G. Sabe que puede llegar a A en 8 *mseg*, y A indica ser capaz de llegar a G en 18 *mseg*, por lo que J sabe que puede contar con un retardo de 26 *mseg* a G si reenvía a través de A los paquetes destinados a G. Del mismo modo, J calcula el retardo a G a través de I, H y K en $41(31 + 10)$, $18(6 + 12)$ y $37(31 + 6)$ *mseg*, respectivamente. El mejor de estos valores es el 18, por lo que escribe una entrada en su tabla de

enrutamiento indicando que el retardo a G es de 18 *mseg*, y que la ruta que se utilizará es vía H. Se lleva a cabo el mismo cálculo para los demás destinos, y la nueva tabla de enrutamiento se muestra en la última columna de la figura.

El problema de la cuenta hasta infinito

El enrutamiento por vector de distancia funciona en teoría, pero tiene un problema serio en la práctica: aunque llega a la respuesta correcta, podría hacerlo lentamente. En particular, reacciona con rapidez a las buenas noticias, pero con lentitud ante las malas. Para ver la rapidez de propagación de las buenas noticias, considere la subred de cinco nodos (lineal) de la *figura* 6, en donde la métrica de retardo es el número de saltos. Suponga que A está desactivado inicialmente y que los otros enrutadores lo saben. En otras palabras, habrán registrado como infinito el retardo a A.

Al activarse A, los demás enrutadores saben de él gracias a los intercambios de vectores. En el momento del primer intercambio, B se entera de que su vecino de la izquierda tiene un retardo de 0 hacia A. B crea entonces una entrada en su tabla de enrutamiento, indicando que A está a un salto de distancia hacia la izquierda. Los demás enrutadores aún piensan que A está desactivado. En este punto, las entradas de la tabla de enrutamiento de A se muestran en la segunda fila de la *figura* 6 *(a)*. Durante el siguiente intercambio, C se entera de que B tiene una ruta a A de longitud 1, por lo que actualiza su tabla de enrutamiento para indicar una ruta de longitud 2, pero D y E no se enteran de las buenas nuevas sino hasta después. Como es evidente, las buenas noticias se difunden a razón de un salto por intercambio. En una subred cuya ruta mayor tiene una longitud de *N* saltos, en un lapso de *N* intercambios todo mundo sabrá sobre las líneas y enrutadores recientemente revividos.

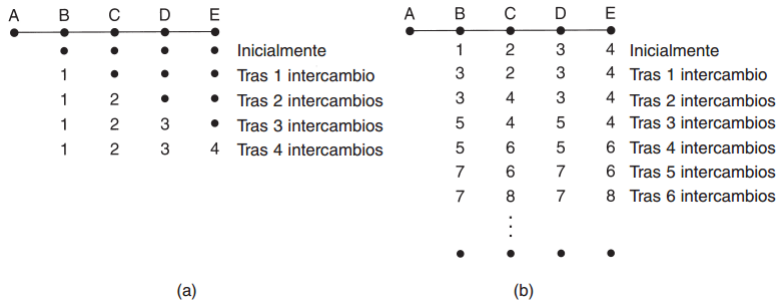


Figura 6: El problema de la cuenta hasta infinito.

Ahora consideremos la situación de la *figura* 6 *(b)*, en la que todas las líneas y enrutadores están activos inicialmente. Los enrutadores B, C, D y E tienen distancias a A de 1, 2, 3 y 4, respectivamente. De pronto, A se desactiva, o bien se corta la línea entre A y B, que de hecho es la misma cosa desde el punto de vista de B. En el primer intercambio de paquetes, B no escucha nada de A. Afortunadamente, C dice: "No te preocupes. Tengo una ruta a A de longitud 2". B no sabe que la ruta de C pasa a través de B mismo. Hasta donde B sabe, C puede tener 10 líneas, todas con rutas independientes a A de longitud 2. Como resultado, B ahora piensa que puede llegar a A por medio de C, con una longitud de ruta de 3. D y E no actualizan sus entradas para A en el primer intercambio.

En el segundo intercambio, C nota que cada uno de sus vecinos indica tener una ruta a A de longitud 3. C escoge una de ellas al azar y hace que su nueva distancia a A sea de 4, como se muestra en la tercera fila de la *figura* 6 *(b)*. Los intercambios subsecuentes producen la historia mostrada en el resto de la *figura*.

A partir de esta figura debe quedar clara la razón por la que las malas noticias viajan con lentitud: ningún enrutador jamás tiene un valor mayor en más de una unidad que el mínimo de todos sus vecinos. Gradualmente, todos los enrutadores elevan cuentas hacia el infinito, pero el número de intercambios requerido depende del valor numérico usado para el infinito. Por esta razón, es prudente hacer que el infinito sea igual a la ruta más larga, más 1. Este problema se conoce como el **problema de la cuenta hasta el infinito**, lo cual no es del todo sorprendente.

2.5. Enrutamiento por estado del enlace

Dos problemas principales causaron la desaparición del enrutamiento por vector de distancia. Primero, debido a que la métrica de retardo era la longitud de la cola, no tomaba en cuenta el ancho de banda al escoger rutas. Un segundo problema: es que el algoritmo con frecuencia tardaba demasiado en converger (el problema de la cuenta hasta el infinito). Por estas razones, el algoritmo fue reemplazado por uno completamente nuevo, llamado **enrutamiento por estado del enlace**.

El concepto en que se basa el enrutamiento por estado del enlace es sencillo y puede enunciarse en cinco partes. Cada enrutador debe:

- Descubrir a sus vecinos y conocer sus direcciones de red.
- Medir el retardo o costo para cada uno de sus vecinos.
- Construir un paquete que indique todo lo que acaba de aprender.
- Enviar este paquete a todos los demás enrutadores.
- Calcular la ruta más corta a todos los demás enrutadores.

Conocimiento de los vecinos

Cuando un enrutador se pone en funcionamiento, su primera tarea es averiguar quiénes son sus vecinos; esto lo realiza enviando un paquete HELLO especial a cada línea punto a punto. Se espera que el enrutador del otro extremo regrese una respuesta indicando quién es. Estos nombres deben ser globalmente únicos puesto que, cuando un enrutador distante escucha después que tres enrutadores están conectados a F, es indispensable que pueda determinar si los tres se refieren al mismo F. Cuando se conectan dos o más enrutadores mediante una LAN, la situación es ligeramente más complicada. Una manera de modelar la LAN es considerarla como otro nodo.

Medición del costo de la línea

El algoritmo de enrutamiento por estado del enlace requiere que cada enrutador sepa, o cuando menos tenga una idea razonable, del retardo a cada uno de sus vecinos. La manera más directa de determinar este retardo es enviar un paquete ECHO especial a través de la línea y una vez que llegue al otro extremo, éste debe regresarlo inmediatamente. Si se mide el tiempo de ida y vuelta y se divide entre dos, el enrutador emisor puede tener una idea razonable del retardo.

Un aspecto interesante es si se debe tomar en cuenta la carga al medir el retardo. Para considerar la carga, el temporizador debe iniciarse cuando el paquete ECHO se ponga en la cola. Para ignorar la carga, el temporizador debe iniciarse cuando el paquete ECHO alcance el frente de la cola. Pueden citarse argumentos a favor de ambos métodos. La inclusión de los retardos inducidos por el tráfico en las mediciones implica que cuando un enrutador puede escoger entre dos líneas con el mismo ancho de banda, una con carga alta continua y otra sin ella, considerará como ruta más corta la de la línea sin carga. Esta selección resultará en un mejor desempeño.

Construcción de los paquetes de estado del enlace

Una vez que se ha recabado la información necesaria para el intercambio, el siguiente paso es que cada enrutador construya un paquete que contenga todos los datos. El paquete comienza con la identidad del emisor, seguida de un número de secuencia, una edad y una lista de vecinos. Se da el retardo de vecino. En la *figura* 7 *(a)* se da un ejemplo de subred, y los retardos se muestran como etiquetas en las líneas. En la *figura* 7 *(b)* se muestran los paquetes de estado del enlace de los seis enrutadores.

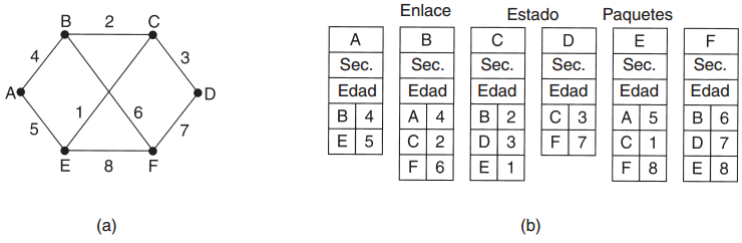


Figura 7: (a) Subred. (b) Paquetes de estado del enlace para esta subred.

Es fácil construir los paquetes de estado del enlace. La parte difícil es determinar cuándo construirlos. Una posibilidad es construirlos de manera periódica, es decir, a intervalos regulares. Otra posibilidad es construirlos cuando ocurra un evento significativo, como la caída o la reactivación de una línea o de un vecino, o el cambio apreciable de sus propiedades.

Distribución de los paquetes de estado del enlace

La parte más complicada del algoritmo es la distribución confiable de los paquetes de estado del enlace. A medida que se distribuyen e instalan los paquetes, los enrutadores que reciban los primeros cambiarán sus rutas. En consecuencia, los distintos enrutadores podrían estar usando versiones diferentes de la topología, lo que puede conducir a inconsistencias, ciclos, máquinas inalcanzables y otros problemas.

Primero describiremos el algoritmo básico de distribución y luego lo refinaremos. La idea fundamental es utilizar inundación para distribuir los paquetes de estado del enlace. A fin de mantener controlada la inundación, cada paquete contiene un número de secuencia que se incrementa con cada paquete nuevo enviado. Los enrutadores llevan el registro de todos los

pares (enrutador de origen, secuencia) que ven. Cuando llega un paquete de estado del enlace, se verifica contra la lista de paquetes ya vistos. Si es nuevo, se reenvía a través de todas las líneas, excepto aquella por la que llegó. Si es un duplicado, se descarta. Si llega un paquete con número de secuencia menor que el mayor visto hasta el momento, se rechaza como obsoleto debido que el enrutador tiene datos más recientes.

Este algoritmo tiene algunos problemas, pero son manejables. Primero, si los números de secuencia vuelven a comenzar, reinará la confusión. La solución aquí es utilizar un número de secuencia de 32 bits. Segundo, si llega a caerse un enrutador, perderá el registro de su número de secuencia. Si comienza nuevamente en 0, se rechazará como duplicado el siguiente paquete. Tercero, si llega a corromperse un número de secuencia y se escribe 65.540 en lugar de 4 (un error de 1 bit), los paquetes 5 a 65.540 serán rechazados como obsoletos, dado que se piensa que el número de secuencia actual es 65.540.

La solución a todos estos problemas es incluir la **edad** de cada paquete después del número de secuencia y disminuirla una vez cada segundo. Cuando la edad llega a cero, se descarta la información de ese enrutador. Los enrutadores también decrementan el campo de edad durante el proceso inicial de inundación para asegurar que no pueda perderse ningún paquete y sobrevivir durante un periodo indefinido (se descarta el paquete cuya edad sea cero).

Algunos refinamientos de este algoritmo lo hacen más robusto. Una vez que un paquete de estado del enlace llega a un enrutador para ser inundado, no se encola para transmisión inmediata. En vez de ello, entra en un área de almacenamiento donde espera un tiempo breve. Si antes de transmitirlo entra otro paquete de estado del enlace proveniente del mismo origen, se comparan sus números de secuencia. Si son iguales, se descarta el duplicado. Si son diferentes, se desecha el más viejo.

Cálculo de las nuevas rutas

Una vez que un enrutador ha acumulado un grupo completo de paquetes de estado del enlace, puede construir el grafo de la subred completa porque todos los enlaces están representados. De hecho, cada enlace se representa dos veces, una para cada dirección. Los dos valores pueden promediarse o usarse por separado. Ahora puede ejecutar localmente el algoritmo de Dijkstra para construir la ruta más corta a todos los destinos posibles. Los resultados de este algoritmo pueden instalarse en las tablas de enrutamiento, y la operación normal puede reiniciarse.

Para una subred con n enrutadores, cada uno de los cuales tiene k vecinos, la memoria requerida para almacenar los datos de entrada es proporcional a kn. En las subredes grandes éste puede ser un problema. También puede serlo el tiempo de cómputo. Sin embargo, en muchas situaciones prácticas, el enrutamiento por estado del enlace funciona bien.

Sin embargo, problemas con el hardware o el software pueden causar estragos con este algoritmo (lo mismo que con otros). Por ejemplo, si un enrutador afirma tener una línea que no tiene, u olvida una línea que sí tiene, el grafo de la subred será incorrecto. Si un enrutador deja de reenviar paquetes, o los corrompe al hacerlo, surgirán problemas. Por último, si al enrutador se le acaba la memoria o se ejecuta mal el algoritmo de cálculo de enrutamiento, surgirán problemas. A medida que la subred crece en decenas o cientos de miles de nodos, la probabilidad de falla ocasional de un enrutador deja de ser insignificante. Lo importante es tratar de limitar el daño cuando ocurra lo inevitable.

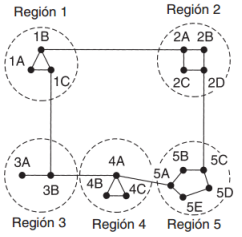
2.6. Enrutamiento jerárquico

A medida que crece el tamaño de las redes, también lo hacen, de manera proporcional, las tablas de enrutamiento del enrutador. Las tablas que siempre crecen no sólo consumen memoria del enrutador, sino que también se necesita más tiempo de CPU para examinarlas y más ancho de banda para enviar informes de estado entre enrutadores. En cierto momento, la red puede crecer hasta el punto en que ya no es factible que cada enrutador tenga una entrada para cada uno de los demás enrutadores, por lo que el enrutamiento tendrá que hacerse de manera jerárquica, como ocurre en la red telefónica.

Cuando se utiliza el enrutamiento jerárquico, los enrutadores se dividen en lo que llamaremos **regiones**, donde cada enrutador conoce todos los detalles para enrutar paquetes a destinos dentro de su propia región, pero no sabe nada de la estructura interna de las otras regiones. Cuando se interconectan diferentes redes, es natural considerar cada una como región independiente a fin de liberar a los enrutadores de una red de la necesidad de conocer la estructura topológica de las demás.

En la *figura* 8 se da un ejemplo cuantitativo de enrutamiento en una jerarquía de dos niveles con cinco regiones. La tabla de enrutamiento completa para el enrutador 1A tiene 17 entradas, como se muestra en la *figura* 8 *(b)*. Si el enrutamiento es jerárquico, como en la *figura* 8 *(c)*, hay entradas para todos los enrutadores locales, igual que antes, pero las demás regiones se han condensado en un solo enrutador, por lo que todo el tráfico para la *región 2* va a través de la línea 1B-2A, pero el resto del tráfico remoto viaja por la línea 1C-3B. El enrutamiento jerárquico redujo la tabla de 17 entradas a 7. A medida que crece la razón entre la cantidad de regiones y el número de enrutadores por región, aumentan los ahorros de espacio de tabla.

Desgraciadamente, estas ganancias de espacio no son gratuitas. Se paga un precio, que es una longitud de ruta mayor. Por ejemplo, la mejor ruta de 1A a 5C es a través de la *región 2* pero con el enrutamiento jerárquico, todo el tráfico a la *región 5*



(a)

Tabla completa para 1A

Dest.	Línea	Saltos
1A	–	–
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

(b)

Tabla jerárquica para 1A

Dest.	Línea	Saltos
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

(c)

Figura 8: Enrutamiento jerárquico.

pasa por la *región 3*, porque eso es mejor para la mayoría de los destinos de la *región 5*.

Cuando una red se vuelve muy grande, surge una pregunta interesante: ¿cuántos niveles debe tener la jerarquía? El número óptimo de niveles para una subred de *N* enrutadores es de $ln(N)$, y se requieren un total de $e * ln(N)$ entradas por enrutador.

2.7. Enrutamiento por difusión

En algunas aplicaciones, los hosts necesitan enviar mensajes a varios otros hosts o a todos los demás. El envío simultáneo de un paquete a todos los destinos se llama difusión; se han propuesto varios métodos para llevarla a cabo.

Un método de difusión que no requiere características especiales de la subred es que el origen simplemente envíe un paquete distinto a todos los destinos. El método no sólo desperdicia ancho de banda, sino que también requiere que el origen tenga una lista completa de todos los destinos.

La inundación es otro candidato obvio. Aunque ésta es poco adecuada para la comunicación punto a punto ordinaria, para difusión puede merecer consideración seria, especialmente si no es aplicable ninguno de los métodos descritos a continuación. El problema de la inundación como técnica de difusión es el mismo que tiene como algoritmo de enrutamiento punto a punto: genera demasiados paquetes y consume demasiado ancho de banda.

Un tercer algoritmo es el enrutamiento multidestino. Con este método, cada paquete contiene una lista de destinos o un mapa de bits que indica los destinos deseados. Cuando un paquete llega al enrutador, éste revisa todos los destinos para determinar el grupo de líneas de salida que necesitará. El enrutador genera una copia nueva del paquete para cada línea de salida que se utilizará, e incluye en cada paquete sólo aquellos destinos que utilizarán la línea. En efecto, el grupo de destinos se divide entre las líneas de salida. Después de una cantidad suficiente de saltos, cada paquete llevará sólo un destino, así que puede tratarse como un paquete normal.

Un cuarto algoritmo de difusión usa explícitamente el árbol sumidero para el enrutador que inicia la difusión, o cualquier otro árbol de expansión adecuado. El árbol de expansión es un subgrupo de la subred que incluye todos los enrutadores pero no contiene ciclos. Si cada enrutador sabe cuáles de sus líneas pertenecen al árbol de expansión, puede copiar un paquete de entrada difundido en todas las líneas del árbol de expansión, excepto en aquella por la que llegó. Este método utiliza de manera óptima el ancho de banda, generando la cantidad mínima de paquetes necesarios para llevar a cabo el trabajo. El único problema es que cada enrutador debe tener conocimiento de algún árbol de expansión para que este método pueda funcionar.

Nuestro último algoritmo de difusión es un intento de aproximar el comportamiento del anterior, aun cuando los enrutadores no saben nada en lo absoluto sobre árboles de expansión. La idea, llamada reenvío por ruta invertida (reverse path *forwarding*), es excepcionalmente sencilla una vez planteada:

Cuando llega un paquete difundido a un enrutador, éste lo revisa para ver si llegó por la línea normalmente usada para enviar paquetes al origen de la difusión. De ser así, hay excelentes posibilidades de que el paquete difundido haya seguido la mejor ruta desde el enrutador y, por lo tanto, sea la primera copia en llegar al enrutador. Si éste es el caso, el enrutador reenvía copias del paquete a todas las líneas, excepto a aquella por la que llegó. Sin embargo, si el paquete difundido llegó por una línea diferente de la preferida, el paquete se descarta como probable duplicado.

La ventaja principal del reenvío por ruta invertida es que es razonablemente eficiente y fácil de implementar. No requiere que

los enrutadores conozcan los árboles de expansión ni tiene la sobrecarga de una lista de destinos o de un mapa de bits en cada paquete de difusión, como los tiene el direccionamiento multidestino. Tampoco requiere mecanismos especiales para detener el proceso, como en el caso de la inundación (ya sea un contador de saltos en cada paquete y un conocimiento previo del diámetro de la subred, o una lista de paquetes ya vistos por origen).

3. Algoritmos de Control de Congestión

Cuando hay demasiados paquetes presentes en la subred (o en una parte de ella), hay una degradación del desempeño. Esta situación se llama **congestión**. Cuando la cantidad de paquetes descargados en la subred por los *hosts* está dentro de su capacidad de conducción, todos se entregan (excepto algunos por errores de transmisión) y la cantidad entregada es proporcional al número enviado. Sin embargo, a medida que aumenta el tráfico, los enrutadores ya no pueden manejarlo y comienzan a perder paquetes. Esto tiende a empeorar las cosas. Con mucho tráfico, el desempeño se desploma por completo y casi no hay entrega de paquetes.

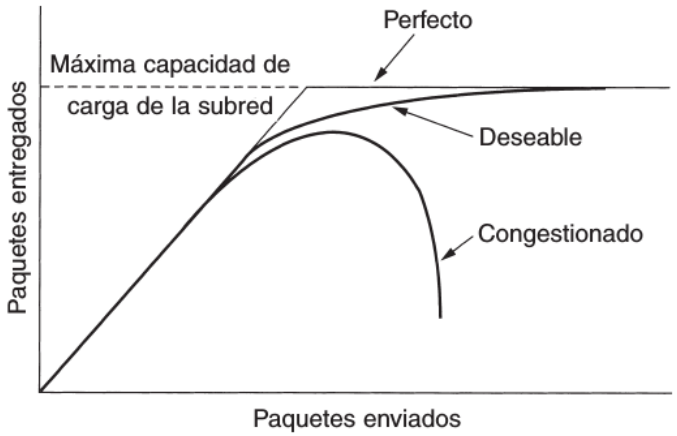


Figura 9: Cuando se genera demasiado tráfico, ocurre congestión y se degrada marcadamente el desempeño.

La congestión puede ocurrir por varias razones. Si de manera repentina comienzan a llegar cadenas de paquetes por tres o cuatro líneas de entrada y todas necesitan la misma línea de salida, se generará una cola. Si no hay suficiente memoria para almacenar a todos los paquetes, algunos de ellos se perderán. La adición de memoria puede ayudar hasta cierto punto: si los enrutadores tienen una cantidad infinita de memoria, la congestión empeora en lugar de mejorar, ya que para cuando los paquetes llegan al principio de la cola, su temporizador ha terminado (repetidamente) y se han enviado duplicados. Todos estos paquetes serán debidamente reenviados al siguiente enrutador, aumentando la carga en todo el camino hasta el destino. Los procesadores lentos también pueden causar congestión.

La actualización de las líneas sin cambiar los procesadores, o viceversa, por lo general ayuda un poco, pero con frecuencia simplemente desplaza el cuello de botella. Además, actualizar sólo parte de un sistema simplemente mueve el cuello de botella a otra parte.

Diferencia entre el control de la congestión y el control de flujo

El control de congestión se ocupa de asegurar que la subred sea capaz de transportar el tráfico ofrecido. Es un asunto global, en el que interviene el comportamiento de todos los *hosts*, todos los enrutadores, el proceso de almacenamiento y reenvío dentro de los enrutadores y todos los demás factores que tienden a disminuir la capacidad de transporte de la subred.

En contraste, el control de flujo se relaciona con el tráfico punto a punto entre un emisor dado y un receptor dado. Su tarea es asegurar que un emisor rápido no pueda transmitir datos de manera continua a una velocidad mayor que la que puede absorber el receptor. El control de flujo casi siempre implica una retroalimentación directa del receptor al emisor, para indicar al emisor cómo van las cosas en el otro lado.

3.1. Principios generales del control de congestión

Soluciones de ciclo abierto

Intentan resolver el problema mediante un buen diseño, para asegurarse en primer lugar de que no ocurra. Una vez que el sistema está en funcionamiento, no se hacen correcciones a medio camino.

Las herramientas para llevar a cabo control de ciclo abierto incluyen decidir cuándo aceptar tráfico nuevo, decidir cuándo descartar paquetes, y cuáles, y tomar decisiones de calendarización en varios puntos de la red. Todas tienen en común el hecho de que toman decisiones independientemente del estado actual de la red.

Soluciones de ciclo cerrado

Se basan en el concepto de un ciclo de retroalimentación. Este método tiene tres partes cuando se aplica al control de congestión:

1. Monitorear el sistema para detectar cuándo y dónde ocurren congestiones.

   Es posible utilizar varias métricas para monitorear la subred en busca de congestiones. Las principales son:

   - Porcentaje de paquetes descartados debido a falta de espacio de búfer.
   - Longitud promedio de las colas.
   - Cantidad de paquetes para los cuales termina el temporizador y se transmiten de nueva cuenta.
   - Retardo promedio de los paquetes.
   - Desviación estándar del retardo de paquete.

   En todos los casos, un aumento en las cifras indica un aumento en la congestión.
2. Pasar esta información a lugares en los que puedan llevarse a cabo acciones.

   Esto es: la transferencia de información relativa a la congestión desde el punto en que se detecta hasta el punto en que puede hacerse algo al respecto. La manera más obvia es que el enrutador que detecta la congestión envíe un paquete al origen del tráfico, anunciando el problema. Por supuesto, estos paquetes adicionales aumentan la carga. Existen otras opciones. Por ejemplo, en cada paquete puede reservarse un bit o campo para que los enrutadores lo llenen cuando la congestión rebase algún umbral. Cuando un enrutador detecta este estado congestionado, llena el campo de todos los paquetes de salida, para avisar a los vecinos.
3. Ajustar la operación del sistema para corregir el problema.

   La presencia de congestión significa que la carga es (temporalmente) superior (en una parte del sistema) a la que pueden manejar los recursos. Vienen a la mente dos soluciones: aumentar los recursos o disminuir la carga. Sin embargo, a veces no es posible aumentar la capacidad, o ya ha sido aumentada al máximo. Entonces, la única forma de combatir la congestión es disminuir la carga. Existen varias maneras de reducir la carga, como negar el servicio a algunos usuarios, degradar el servicio para algunos o todos los usuarios y obligar a los usuarios a programar sus solicitudes de una manera más predecible.

3.2. Políticas de prevención de congestión

Los sistemas de ciclo abierto están diseñados para reducir al mínimo la congestión desde el inicio, usando políticas adecuadas en varios niveles (ver Figura 10).

Capa	Políticas
Transporte	• Política de retransmisión • Política de almacenamiento en caché de paquetes fuera de orden • Política de confirmaciones de recepción • Política de control de flujo • Determinación de terminaciones de temporizador
Red	• Circuitos virtuales vs. datagramas en la subred • Política de encolamiento y servicio de paquetes • Política de descarte de paquetes • Algoritmo de enrutamiento • Administración de tiempo de vida del paquete
Enlace de datos	• Política de retransmisiones • Política de almacenamiento en caché de paquetes fuera de orden • Política de confirmación de recepción • Política de control de flujo

Figura 10: Políticas relacionadas con la congestión.

3.3. Control de congestión en subredes de circuitos virtuales

Una de las técnicas que se usa ampliamente para evitar que empeoren las congestiones que ya han comenzado es el **control de admisión**: una vez que se ha detectado la congestión, no se establecen circuitos virtuales nuevos hasta que ha desaparecido el problema. Por lo tanto, fallan los intentos por establecer conexiones nuevas de capa de transporte. Permitir el acceso a más usuarios simplemente empeoraría las cosas.

Un método alterno es permitir el establecimiento de nuevos circuitos virtuales, pero enrutando cuidadosamente los circuitos nuevos por otras rutas que no tengan problemas. Ver por ejemplo la Figura 11.

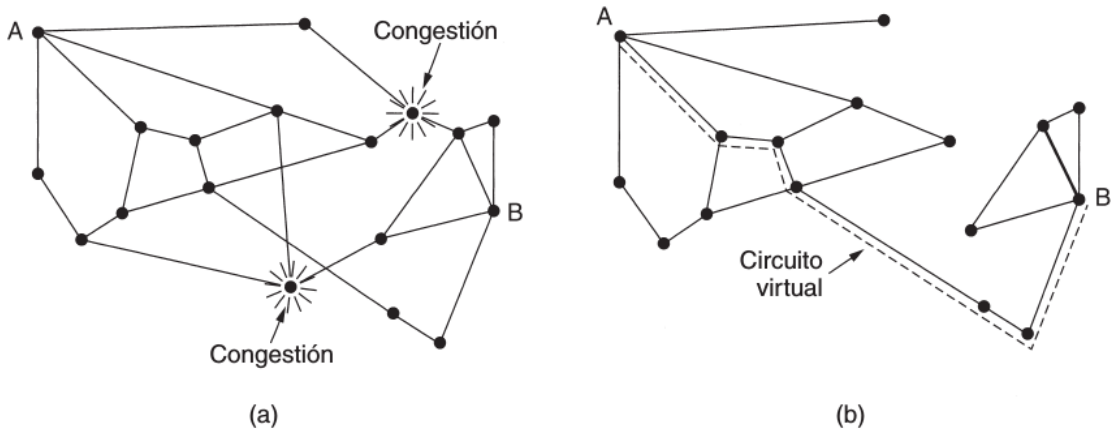


Figura 11: (a) Subred congestionada. (b) Subred redibujada que elimina la congestión. También se muestra un circuito virtual de A a B.

Otra estrategia es negociar un acuerdo entre el host y la subred cuando se establece un circuito virtual. Este arreglo normalmente especifica el volumen y la forma del tráfico, la calidad de servicio requerida y otros parámetros. Para cumplir con su parte del acuerdo, la subred por lo general reservará recursos a lo largo de la ruta cuando se establezca el circuito. Estos recursos pueden incluir espacio en tablas y en búfer en los enrutadores y ancho de banda en las líneas. La consecuencia del control de congestión es un ancho de banda sin utilizar (desperdiciado).

3.4. Control de congestión en subredes de datagramas

Cada enrutador puede monitorear fácilmente el uso de sus líneas de salida y de otros recursos, por lo que puede elaborar estrategias de control.

El bit de advertencia

Cuando el paquete llega a su destino, la entidad transportadora copia el bit especial (que señala el estado de advertencia) en la siguiente *ACK* que se regresa al origen. El origen entonces reduce el tráfico. Mientras el enrutador esté en el estado de advertencia, el bit de advertencia se mantiene activo, y el origen continúa disminuyendo su tasa de transmisión.

Paquetes reguladores

En este método, el enrutador regresa un paquete regulador al host de origen, proporcionándole el destino encontrado en el paquete. El paquete original se etiqueta (se activa un bit del encabezado) de manera que no genere más paquetes reguladores. Cuando el host de origen obtiene el paquete regulador, se le pide que reduzca en un porcentaje X el tráfico enviado al destino especificado.

3.5. Desprendimiento de carga

Cuando ninguno de los métodos anteriores elimina la congestión, los enrutadores pueden optar por el **desprendimiento de carga**: si el enrutador no puede manejar las cargas, elimina los paquetes y que las capas superiores se encarguen de enviarlos nuevamente.

Detección temprana aleatoria

Es bien sabido que tratar con la congestión después de que se detecta por primera vez es más efectivo que dejar que dañe el trabajo y luego tratar de solucionarlo. Esta observación conduce a la idea de descartar paquetes antes de que se ocupe todo el

espacio de búfer. Un algoritmo popular para realizar esto se conoce como **RED** (*Random Early Detection*: Detección Temprana Aleatoria). El objetivo es hacer que los enrutadores se deshagan de los paquetes antes de que la situación sea irremediable. Para determinar cuándo comenzar a descartarlos, los enrutadores mantienen un promedio móvil de sus longitudes de cola. Cuando la longitud de cola promedio en algunas líneas sobrepasa un umbral, se dice que la línea está congestionada y se toma alguna medida.

3.6. Control de fluctuación

En aplicaciones como la transmisión continua de audio y vídeo no importa gran cosa si los paquetes tardan 20 o 30 mseg en ser entregados, siempre y cuando el tiempo de tránsito (retardo) sea constante. La variación (es decir, la desviación estándar) en el retardo de los paquetes se conoce como fluctuación. Una fluctuación alta, por ejemplo cuando unos paquetes tardan en llegar a su destino 20 mseg y otros 30 mseg, resultará en una calidad desigual del sonido o la imagen. Podría ser aceptable entonces lograr una solución donde la mayoría de los paquetes se entregue con un retardo que esté entre 24.5 y 25.5 mseg.

La fluctuación puede limitarse calculando el tiempo de tránsito esperado para cada salto en la ruta. Cuando un paquete llega a un enrutador, éste lo examina para saber qué tan adelantado o retrasado está respecto a lo programado. Esta información se almacena en el paquete y se actualiza en cada salto. Si el paquete está adelantado, se retiene durante el tiempo suficiente para regresarlo a lo programado; si está retrasado, el enrutador trata de sacarlo rápidamente.

En algunas aplicaciones, como el vídeo bajo demanda, la fluctuación puede eliminarse almacenando los datos en el búfer del receptor y después obteniéndolos de dicho búfer en lugar de utilizar la red en tiempo real. Sin embargo, para otras aplicaciones, especialmente aquellas que requieren interacción en tiempo real entre personas como la telefonía y videoconferencia en Internet, el retardo inherente del almacenamiento en el búfer no es aceptable.

4. Calidad del Servicio

4.1. Requerimientos

El **flujo** es el conjunto de paquetes que van de un origen a un destino. Se puede caracterizar por cuatro parámetros principales (*CRAF*):

- Confiabilidad
- Retardo
- Ancho de banda
- Fluctuación

En conjunto determinan la **QoS** (*Quality of Service*) que el flujo requiere.

En una red orientada a la conexión, todos los paquetes que pertenezcan a un flujo siguen la misma ruta; en una red no orientada a la conexión, pueden seguir diferentes rutas.

Aplicación	Confiabilidad	Retardo	Fluctuación	Ancho de banda
Correo electrónico	Alta	Bajo	Baja	Bajo
Transferencia de archivos	Alta	Bajo	Baja	Medio
Acceso a Web	Alta	Medio	Baja	Medio
Inicio de sesión remoto	Alta	Medio	Media	Bajo
Audio bajo demanda	Baja	Bajo	Alta	Medio
Vídeo bajo demanda	Baja	Bajo	Alta	Alto
Telefonía	Baja	Alto	Alta	Bajo
Videoconferencia	Baja	Alto	Alta	Alto

Figura 12: Qué tan rigurosos son los requerimientos de *QoS*.

4.2. Técnicas para alcanzar buena calidad de servicio

Ninguna técnica proporciona *QoS* eficiente y confiable de una manera óptima. En su lugar, se ha desarrollado una variedad de técnicas, con soluciones prácticas que con frecuencia combinan múltiples técnicas.

Sobreaprovisionamiento

Proporciona la suficiente capacidad de enrutador, espacio en búfer y ancho de banda como para que los paquetes fluyan con facilidad, a cambio de mayor coste de inversión.

Almacenamiento en búfer

Los flujos pueden almacenarse en el búfer en el lado receptor antes de ser entregados. Tiene las siguientes consecuencias en los requerimientos:

- Confiabilidad y ancho de banda: no les afecta.
- Retardo: se incrementa.
- Fluctuación: atenúa o mejora.

Modelado de tráfico

Consiste en regular la tasa promedio (y las ráfagas) de la transmisión de los datos. Cuando se establece una conexión, el usuario y la subred acuerdan un cierto patrón de tráfico para ese circuito: forman un **acuerdo de nivel de servicio**.

El modelado de tráfico reduce la congestión. Para saber si el cliente está cumpliendo con el acuerdo se hace una supervisión de tráfico. Aceptar una forma de tráfico y supervisarlo más tarde es más fácil en las subredes de circuitos virtuales que en las de datagramas. Se utiliza para vídeo y audio en tiempo real.

Algoritmo de cubeta con goteo

Sin importar la rapidez con que entra agua en la cubeta, el flujo de salida tiene una tasa constante, ρ, cuando hay agua en la cubeta, y una tasa de cero cuando la cubeta está vacía. Además, una vez que se llena la cubeta, cualquier agua adicional que entra se derrama por los costados y se pierde (es decir, no aparece en el flujo por debajo del agujero).

De manera conceptual, cada host está conectado a la red mediante una interfaz que contiene una cubeta con goteo (una cola interna finita). Si llega un paquete cuando la cola está llena, éste se descarta.

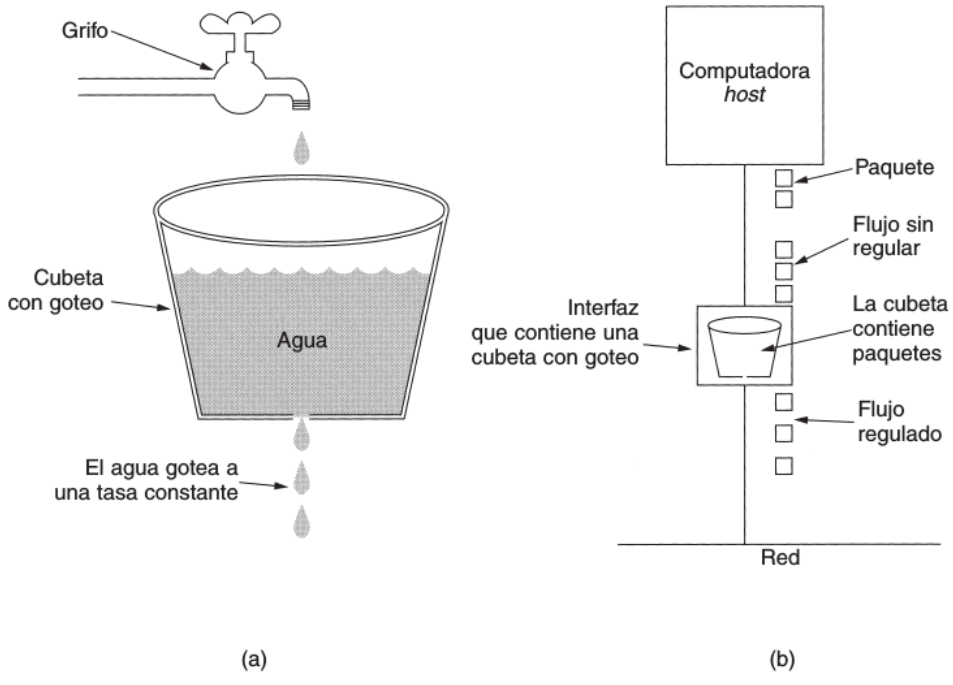


Figura 13: (a) Una cubeta con goteo, llena de agua. (b) Cubeta con goteo, llena de paquetes.

Convierte un flujo desigual de paquetes de los procesos de usuario dentro del host en un flujo continuo de paquetes hacia la red, moderando las ráfagas y reduciendo las posibilidades de congestión.

Algoritmo de cubeta con tokens

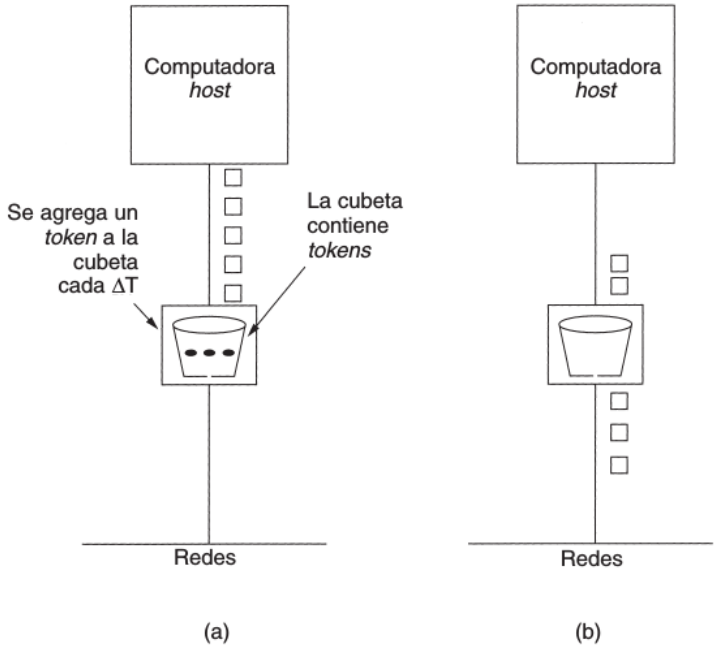


Figura 14: Algoritmo de cubeta con *tokens*. (a) Antes. (b) Después.

El algoritmo de cubeta con goteo impone un patrón de salida rígido a la tasa promedio, sin importar la cantidad de ráfagas que tenga el tráfico. Un algoritmo más flexible puede permitir que la salida se acelere un poco cuando llegan ráfagas grandes.

En esencia, lo que hace este algoritmo es permitir ráfagas, pero limitadas a una longitud máxima regulada. Para ello, funciona con un sistema de permisos (*tokens*). Los *hosts* inactivos acumulan permisos hasta el tamaño máximo de la cubeta, para enviar posteriormente ráfagas grandes. Esta propiedad proporciona irregularidad en el flujo de salida y da una respuesta más rápida a las ráfagas de entrada repentinas. Cuando se llena la cubeta se descartan *tokens*.

Reservación de recursos

Conociendo una ruta específica para una flujo, es posible reservar recursos a lo largo de esa ruta para asegurar que la capacidad necesaria esté disponible. Estos pueden ser del tipo:

1. **Ancho de banda**: no sobrecargar ninguna línea de salida.
2. **Espacio de búfer**: es posible reservar para que cierto flujo específico no tenga que competir con otros flujos por el espacio en búfer.
3. **Ciclos de CPU**: el enrutador sólo puede procesar cierta cantidad de paquetes por segundo.

Control de admisión

Está relacionada con la reservación de recursos. Cuando se desea crear un circuito nuevo, el origen envía una especificación de flujo por el circuito. Esta indica datos como la tasa de la cubeta con *tokens*, la tasa pico de datos, los tamaños máximos y mínimos de los paquetes, etc. Cada router que esta en el circuito revisa esta especificación y la modifica en caso de no poder garantizarla. Cuando llega al otro lado se puede establecer los parámetros.

Enrutamiento proporcional

Se divide el trafico de un flujo por varios rutas. Para definir la cantidad de paquetes que se envía por cada ruta se puede utilizar información local, como el ancho de banda de la linea.

Calendarización de paquetes

Controla el orden en que se envían los paquetes en un router. En vez de enviar los paquetes en el orden de llegada, se utilizan varias colas de llegada, y se hace *Round-Robin*[1] sobre estas para enviar los paquetes. Esto se conoce como **encolamiento justo**.

[1] Método para seleccionar todos los elementos en un grupo de manera equitativa y en un orden racional, normalmente comenzando por el primer elemento de la lista hasta llegar al último y empezando de nuevo desde el primer elemento. Una forma sencilla de entender el Round-robin es imaginar una secuencia para "tomar turnos". En operaciones computacionales, un método para ejecutar diferentes procesos de manera concurrente, para la utilización equitativa de los recursos del equipo, es limitando cada proceso a un pequeño período, y luego suspendiendo este proceso para dar oportunidad a otro proceso y así

Para mejorarlo, en vez de hacer un *Round-Robin*, se puede ordenar los paquetes que están al frente de cada cola de acuerdo a su tamaño, enviando primero los mas pequeños. También se le puede agregar un peso a cada cola. El nombre del algoritmo que implementa esta modificación es **encolamiento justo ponderado**.

4.3. Conmutación de etiquetas y MPLS

Consiste en agregar una etiqueta en frente de cada paquete y realizar enrutamiento con base en ella y no con base en la dirección de destino. Hacer que la etiqueta sea un índice de una tabla provoca que encontrar la línea correcta de salida sea una simple cuestión de buscar en una tabla.

MPLS

En algún momento, la IETF comenzó a estandarizar la idea de la conmutación de etiquetas bajo el nombre MPLS (*Multi Protocol Layer Switch*: Conmutación de Etiquetas Multiprotocolo). Se describe en el RFC 3031.

Algunos distinguen entre *enrutamiento* y *conmutación*: el enrutamiento es el proceso de buscar una dirección de destino en una tabla para saber a dónde enviar los paquetes hacia ese destino. En contraste, la conmutación utiliza una etiqueta que se toma de un paquete como un índice en una tabla de reenvío (aunque no hay universalidad en los criterios adoptados para estas definiciones).

El primer problema es en dónde colocar la etiqueta. Debido a que los paquetes IP no fueron diseñados para circuitos virtuales, en el encabezado IP no hay ningún campo disponible para los números de tales circuitos. Por esta razón, se tuvo que agregar un nuevo encabezado MPLS enfrente del encabezado IP. En una línea de enrutador a enrutador que utiliza PPP como protocolo de tramas, el formato de trama, incluyendo los encabezados PPP, MPLS, IP y TCP, es como se muestra en la Figura 15.

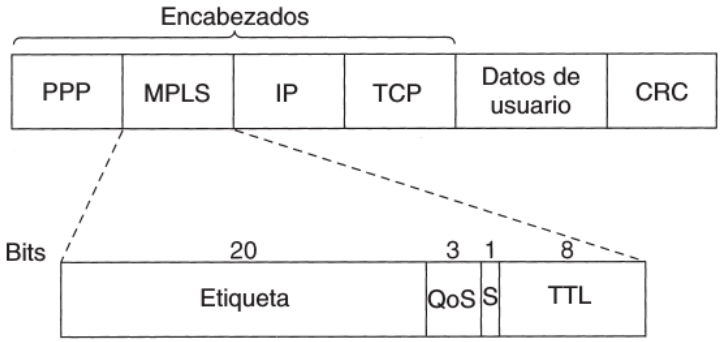


Figura 15: Transmisión de un segmento TCP que utiliza IP, MPLS y PPP.

El encabezado tiene cuatro campos:

- Etiqueta: contiene los índices.
- QoS: indica la clase de servicio.
- S: se relaciona con colocar en una pila múltiples etiquetas en redes jerárquicas. Si vale uno es la última etiqueta añadida.
- TTL: evita el ciclo infinito en caso de que haya inestabilidad en enrutamiento.

Se puede decir que es un protocolo de capa 2 y $1/2$.

5. La Capa de Red de Internet

En la capa de red, la Internet puede verse como un conjunto de subredes, o sistemas autónomos interconectados. No hay una estructura real, pero existen varias redes dorsales principales. Éstas se construyen a partir de líneas de alto ancho de banda y enrutadores rápidos. Conectadas a las redes dorsales hay redes regionales (de nivel medio), y conectadas a estas redes regionales están las LANs de muchas universidades, compañías y proveedores de servicios de Internet.

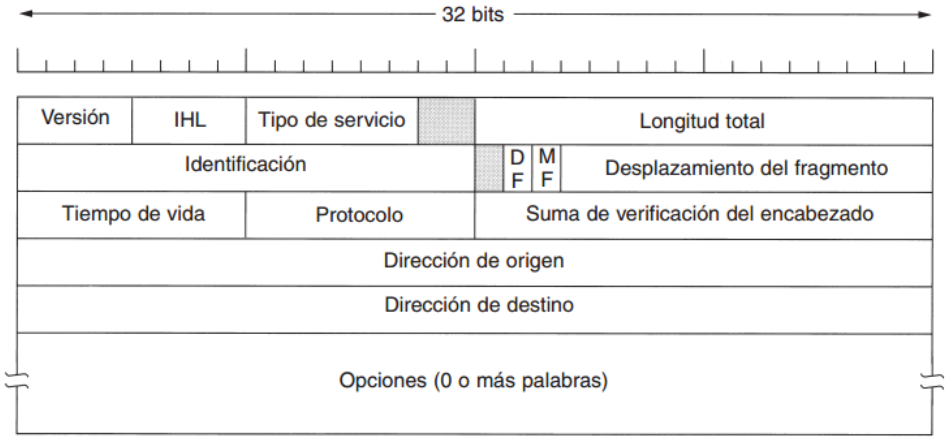
El pegamento que mantiene unida a Internet es el protocolo de capa de red, IP (Protocolo de Internet). A diferencia de la mayoría de los protocolos de capa de red anteriores, éste se diseñó desde el principio con la interconexión de redes en mente. Una buena manera de visualizar la capa de red es la siguiente. Su trabajo es proporcionar un medio de mejor esfuerzo

sucesivamente. A esto se le denomina comúnmente como Planificación Round-Robin.

(es decir, sin garantía) para el transporte de datagramas del origen al destino, sin importar si estas máquinas están en la misma red, o si hay otras redes entre ellas.

5.1. El Protocolo IP

Un datagrama IP consiste en una parte de encabezado y una parte de texto. El encabezado tiene una parte fija de 20 bytes y una parte opcional de longitud variable. Se transmite en orden de *bigendian*: de izquierda a derecha, comenzando por el bit de orden mayor del campo de Versión.



El campo de *Versión* lleva el registro de la versión del protocolo al que pertenece el datagrama. Al incluir la versión en cada datagrama, es posible hacer que la transición entre versiones se lleve meses, o incluso años, ejecutando algunas máquinas la versión vieja y otras la versión nueva. Dado que la longitud del encabezado no es constante, se incluye un campo en el encabezado, IHL, para indicar la longitud en palabras de 32 bits. El valor mínimo es de 5, cifra que se aplica cuando no hay opciones. El valor máximo de este campo de 4 bits es 15, lo que limita el encabezado a 60 bytes y, por lo tanto, el campo de Opciones a 40 bytes.

El campo de *Tipo de servicio* es uno de los pocos campos que ha cambiado su significado (levemente) durante años. Su propósito aún es distinguir entre las diferentes clases de servicios. Son posibles varias combinaciones de confiabilidad y velocidad.

Originalmente, el campo de 6 bits contenía (de izquierda a derecha) un campo de *Precedencia* de tres bits y tres banderas, D, T y R. El campo de *Precedencia* es una prioridad, de 0 (normal) a 7 (paquete de control de red). Los tres bits de bandera permiten al host especificar lo que le interesa más del grupo retardo (delay), velocidad real de transporte (throughput), confiabilidad (reliability). En teoría, estos campos permiten a los enrutadores tomar decisiones entre, por ejemplo, un enlace satelital de alto rendimiento y alto retardo o una línea arrendada con bajo rendimiento y poco retardo. En la práctica, los enrutadores actuales ignoran por completo el campo de Tipo de servicio, a menos que se les indique lo contrario.

La *Longitud total* incluye todo el datagrama: tanto el encabezado como los datos. La longitud máxima es de 65.535 bytes. Actualmente este límite es tolerable, pero con las redes futuras de gigabits se requerirán datagramas más grandes.

El campo de *Identificación* es necesario para que el host de destino determine a qué datagrama pertenece un fragmento recién llegado. Todos los fragmentos de un datagrama contienen el mismo valor de *Identificación*.

A continuación viene un bit sin uso y luego dos campos de 1 bit. *DF* significa no fragmentar (Don't Fragment); es una orden para los enrutadores de que no fragmenten el datagrama, porque el destino es incapaz de juntar las piezas de nuevo. *MF* significa más fragmentos. Todos los fragmentos excepto el último tienen establecido este bit, que es necesario para saber cuándo han llegado todos los fragmentos de un datagrama.

El *Desplazamiento del fragmento* indica en qué parte del datagrama actual va este fragmento. Todos los fragmentos excepto el último del datagrama deben tener un múltiplo de 8 bytes, que es la unidad de fragmentos elemental. Dado que se proporcionan 13 bits, puede haber un máximo de 8.192 fragmentos por datagrama, dando una longitud máxima de datagrama de 65.536 bytes, uno más que el campo de Longitud total.

El campo de *Tiempo de vida* es un contador que sirve para limitar la vida de un paquete. Se supone que este contador cuenta el tiempo en segundos, permitiendo una vida máxima de 255 seg; debe disminuirse en cada salto y se supone que disminuye muchas veces al encolarse durante un tiempo grande en un enrutador. En la práctica, simplemente cuenta los saltos. Cuando el contador llega a cero, el paquete se descarta y se envía de regreso un paquete de aviso al host de origen. Esta característica

evita que los datagramas vaguen eternamente, algo que de otra manera podría ocurrir si se llegan a corromper las tablas de enrutamiento.

Una vez que la capa de red ha ensamblado un datagrama completo, necesita saber qué hacer con él. El campo de *Protocolo* indica el protocolo de las capas superiores al que debe entregarse el paquete. **TCP** es una posibilidad, pero también está **UDP** y algunos más.

La *Suma de verificacióndel encabezado* verifica solamente el encabezado. Tal suma de verificación es útil para la detección de errores generados por palabras de memoria erróneas en un enrutador.

La *Dirección de origen* y *la Dirección de destino* indican el número de red y el número de host.

El campo de *Opciones* se diseñó para proporcionar un recurso que permitiera que las versiones subsiguientes del protocolo incluyeran información no presente en el diseño original, para permitir que los experimentadores prueben ideas nuevas y para evitar la asignación de bits de encabezado a información pocas veces necesaria. Las opciones son de longitud variable. Originalmente se definieron cinco opciones, como se lista en la siguiente figura.

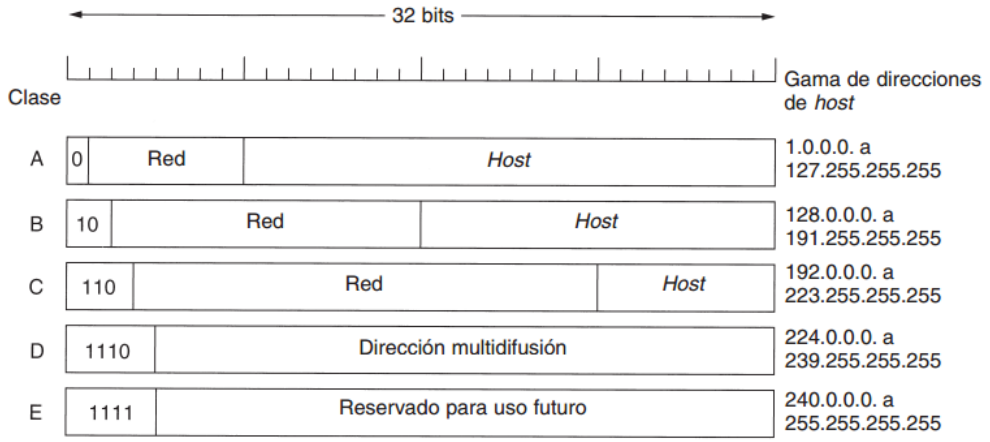
Opción	Descripción
Seguridad	Especifica qué tan secreto es el datagrama
Enrutamiento estricto desde el origen	Indica la ruta completa a seguir
Enrutamiento libre desde el origen	Da una lista de los enrutadores que no deben evitarse
Registrar ruta	Hace que cada enrutador agregue su dirección IP
Marca de tiempo	Hace que cada enrutador agregue su dirección y su marca de tiempo

5.2. Direcciones IP

Cada *host* y enrutador de Internet tiene una dirección IP, que codifica su número de red y su número de *host*. La combinación es única: no hay dos máquinas que tengan la misma dirección IP.

Todas las direcciones IP son de 32 bits de longitud y se usan en los campos de *Dirección de origen* y de *Dirección de destino* de los paquetes IP. En importante mencionar que una dirección IP realmente no se refiere a un host. En realidad se refiere a una interfaz de red, por lo que si un host está en dos redes, debe tener dos direcciones IP. Sin embargo, en la práctica, la mayoría de los hosts se encuentran en una red y, por lo tanto, tienen una dirección IP.

Por varias décadas, las direcciones IP se dividieron en cinco categorías, las cuales se listan a continuación. Esta asignación se ha llamado **direccionamiento con clase** (*classful addressing*).



Las direcciones de red, que son números de 32 bits, generalmente se escriben en notación decimal con puntos. En este formato, cada uno de los 4 bytes se escribe en decimal, de 0 a 255. La dirección IP menor es $0.0.0.0$ y la mayor $255.255.255.255$. Los valores 0 y -1(todos $1s$) tienen significado especial. El valor 0 significa esta red o este host. El valor -1 se usa como dirección de difusión para indicar todos los hosts de la red indicada.

La dirección IP $0.0.0.0$ es usada por los hosts cuando están siendo arrancados, pero no se usa después. Las direcciones IP con 0 como número de red se refieren a la red actual. Estas direcciones permiten que las máquinas se refieran a su propia red sin saber su número (pero tiene que saber su clase para saber cuántos $0s$ hay que incluir). La dirección que consiste

solamente en $1s$ permite la difusión en la red local, por lo común una LAN. Las direcciones con un número de red propio y solamente unos en el campo de host permiten que las máquinas envíen paquetes de difusión a LANs distantes desde cualquier parte de Internet. Por último, todas las direcciones de la forma $127.xx.yy.zz$ se reservan para direcciones locales de prueba (*loopbacks*). Los paquetes enviados a esa dirección no se colocan en el cable; se procesan localmente y se tratan como paquetes de entrada. Esto permite que los paquetes se envíen a la red local sin que el transmisor conozca su número.

Subredes

Como hemos visto, todos los hosts de una red deben tener el mismo número de red. Esta propiedad del direccionamiento IP puede causar problemas a medida que crezcan las redes. El problema es la regla de que una sola dirección de clase A, B o C haga referencia a una red, no a una colección de LANs. Conforme más y más organizaciones se encontraron en esta situación, se hizo un pequeño cambio al sistema de direccionamiento para manejar tal situación. La solución a este problema es permitir la división de una red en varias partes para uso interno, pero aún actuar como una sola red ante el mundo exterior.

Cuando un paquete entra en el enrutador principal, ¿cómo sabe a cuál subred pasarlo (Ethernet)? Básicamente, en lugar de tener una sola dirección de clase B con 14 bits para el número de red y 16 bits para el número de host, algunos bits se eliminan del número de host para crear un número de subred.

Para implementar subredes, el enrutador principal necesita una máscara de subred que indique la división entre el número de red + el número de subred y el host. Las máscaras de subred también se pueden escribir en notación decimal con puntos, o agregando a la dirección IP una diagonal seguida del número de bits usados para los números de red y subred.

Para ver el funcionamiento de las subredes, es necesario explicar la manera en que se procesan los paquetes IP en un enrutador. Cada enrutador tiene una tabla en la que se lista cierto número de direcciones IP (red, 0) y cierto número de direcciones IP (esta red, host). El primer tipo indica cómo llegar a redes distantes. El segundo tipo indica cómo llegar a redes locales. La interfaz de red a utilizar para alcanzar el destino, así como otra información, está asociada a cada tabla.

Cuando llega un paquete IP, se busca su dirección de destino en la tabla de enrutamiento. Si el paquete es para una red distante, se reenvía al siguiente enrutador de la interfaz dada en la tabla; si es para un host local, se envía directamente al destino. Si la red no está en la tabla, el paquete se reenvía a un enrutador predeterminado con tablas más extensas. Este algoritmo significa que cada enrutador sólo tiene que llevar el registro de otras redes y hosts locales (no de pares red-host), reduciendo en gran medida el tamaño de la tabla de enrutamiento.

Al introducirse subredes, se cambian las tablas de enrutamiento, agregando entradas con forma de (esta red, subred, 0) y (esta red, esta subred, host). Por lo tanto, un enrutador de la subred *k* sabe cómo llegar a todas las demás subredes y a todos los hosts de la subred *k*; no tiene que saber los detalles sobre los hosts de otras subredes. De hecho, todo lo que se necesita es hacer que cada enrutador haga un AND booleano con la máscara de subred de la red para deshacerse del número de host y buscar la dirección resultante en sus tablas (tras determinar de qué clase de red se trata).

CIDR—Enrutamiento interdominios sin clases

El problema, en pocas palabras, es que Internet se está quedando rápidamente sin direcciones de IP. En teoría, existen cerca de dos mil millones de direcciones, pero la práctica de organizar el espacio de direcciones por clases desperdicia millones de ellas. En particular, el verdadero villano es la red clase B. Para la mayoría de las organizaciones, una red clase A, con 16 millones de direcciones, es demasiado grande, y una red clase C, de 256 direcciones, es demasiado pequeña. Una red clase B, con 65536, es la adecuada. En realidad, una dirección clase B es demasiado grande para la mayoría de las organizaciones. Hay estudios que demuestran que más de la mitad de todas las redes clase B tienen menos de 50 hosts.

Actualmente, el almacenamiento de medio millón de entradas tal vez es posible, aunque caro en los enrutadores. Además, diversos algoritmos de enrutamiento requieren que cada enrutador transmita sus tablas periódicamente (por ejemplo, protocolos de vector de distancia). Cuanto más grandes sean las tablas, más probabilidad habrá de que algunas partes se pierdan en el camino, dando pie a datos incompletos en el otro lado y posiblemente a inestabilidades de enrutamiento.

El problema de las tablas de enrutamiento podría haberse resuelto usando una jerarquía más. Por ejemplo, podría haber servido hacer que cada dirección IP tuviera un campo de país, estado, ciudad, red y host. Entonces cada enrutador sólo necesitaría saber cómo llegar a los países, a los estados o provincias de su propio país, a las ciudades de su estado o provincia y a las redes de su ciudad. Por desgracia, esta solución requeriría bastante más de 32 bits para las direcciones de IP y usaría ineficientemente las direcciones.

En resumen, la mayoría de las soluciones resuelven un problema pero crean uno nuevo. La solución que se implementó y que dio a Internet un respiro es el **CIDR (Enrutamiento Interdominios sin Clases)**. El concepto básico del CIDR es asignar las direcciones IP restantes en bloques de tamaño variable, independientemente de las clases.

Con CIDR, este algoritmo sencillo ya no funciona. En cambio, cada entrada de tabla de enrutamiento se extiende para darle una máscara de 32 bits. De esta manera, ahora hay una sola tabla de enrutamiento para todas las redes que consten de

un arreglo de tres variables (dirección IP, máscara de subred, línea saliente). Cuando llega un paquete, primero se extrae su dirección de destino IP. Luego (conceptualmente) se analiza la tabla de enrutamiento entrada por entrada, enmascarando la dirección de destino y comparándola con la entrada de la tabla buscando una correspondencia. Es posible que coincidan entradas múltiples (con diferentes longitudes de máscara de subred), en cuyo caso se usa la máscara más larga. De esta manera, si hay una coincidencia para una máscara $/20$ y una máscara $/24$, se usa la entrada $/24$.

NAT - Traducción de Dirección de Red

La idea básica de NAT es asignar una sola dirección IP a cada compañía (o a lo sumo, un número pequeño) para el tráfico de Internet. Dentro de la compañía, cada computadora tiene una dirección IP única que se usa para enrutar el tráfico interno. Sin embargo, cuando un paquete sale de la compañía y va al ISP, se presenta una traducción de dirección. Para hacer posible este esquema los tres rangos de direcciones IP se han declarado como privados. Las compañías pueden usarlos internamente cuando lo deseen. La única regla es que ningún paquete que contiene estas direcciones puede aparecer en la propia Internet. Los tres rangos reservados son:

10.0.0.0	–	10.255.255.255/8	(16.777.216 *hosts*)
172.16.0.0	–	172.31.255.255/12	(1.048.576 *hosts*)
192.168.0.0	–	192.168.255.255/16	(65.536 *hosts*)

Los diseñadores de NAT observaron que la mayoría de paquetes IP lleva cargas útiles de TCP o UDP: los dos tienen encabezados que contienen un puerto de origen y un puerto de destino. Los puertos son enteros de 16 bits que indican dónde empieza y dónde acaba la conexión TCP. Estos puertos proporcionan el campo requerido para hacer que NAT funcione.

Cuando un proceso quiere establecer una conexión TCP con un proceso remoto, se conecta a un puerto TCP sin usar en su propia máquina. Éste se conoce como puerto de origen y le indica a TCP dónde enviar paquetes entrantes que pertenecen a esta conexión. El proceso también proporciona un puerto de destino para decir a quién dar los paquetes en el lado remoto. Los puertos 0-1023 se reservan para servicios bien conocidos. Por ejemplo, 80 es el puerto usado por los servidores Web, para que los clientes remotos puedan localizarlos. Cada mensaje TCP saliente contiene un puerto de origen y un puerto de destino. Juntos, estos puertos sirven para identificar los procesos que usan la conexión en ambos extremos.

Es necesario reemplazar el *Puerto de origen* porque podría ocurrir que ambas conexiones de las máquinas 10.0.0.1 y 10.0.0.2 usaran el puerto 5000, por ejemplo, así que el Puerto de origen no basta para identificar el proceso de envío.

Objeciones:

1. NAT viola el modelo arquitectónico de IP que establece que cada dirección IP identifica una sola máquina globalmente. Toda la estructura del software de Internet se basa en este hecho. Con NAT, las miles de máquinas pueden usar (y lo hacen) la dirección 10.0.0.1.
2. NAT cambia a Internet de una red sin conexión a un tipo de red orientada a la conexión.
3. NAT viola la regla más fundamental de los protocolos de capas: la capa k no puede hacer ninguna suposición de en qué capa $k + 1$ ha puesto el campo de carga útil.
4. En Internet no se exige que los procesos utilicen TCP o UDP.
5. Algunas aplicaciones insertan direcciones IP en el cuerpo del texto. El receptor extrae estas direcciones y las usa. Puesto que NAT no sabe nada sobre estas direcciones, no las puede reemplazar, de modo que fallará cualquier intento de usarlas en el extremo remoto. El **FTP** (*File Transfer Protocol*: Protocolo de Transferencia de Archivos) estándar funciona de esta manera y puede fallar.
6. Debido a que el campo Puerto de origen de TCP es de 16 bits, a lo sumo se pueden asignar 65,536 máquinas hacia una dirección IP. En realidad, el número es ligeramente menor porque los primeros 4096 puertos se reservan para usos especiales. Sin embargo, si hay varias direcciones IP disponibles, cada una puede manejar 61,440 máquinas.

5.3. Protocolos de Control en Internet

Además del IP que se usa para transferencia de datos, Internet tiene algunos protocolos de control que se usan en la capa de redes, como **ICMP**, **ARP**, **RARP**, **BOOTP** y **DHCP**.

Protocolo de Mensajes de Control en Internet

Los enrutadores supervisan estrechamente el funcionamiento de Internet. Cuando ocurre algo inesperado, el **Protocolo de Mensajes de Control en Internet** (**ICMP**) informa del evento. Hay definidos alrededor de una docena de tipos de mensajes ICMP. Los más importantes se listan en la Figura 16.

ARP - Protocolo de Resolución de Direcciones Aunque en Internet cada máquina tiene una (o más) direcciones IP,

Tipo de mensaje	Descripción
Destination unreachable	El paquete no se pudo entregar
Time exceeded	Campo de tiempo de vida = 0
Parameter problem	Campo de encabezado no válido
Source quench	Paquete regulador
Redirect	Enseña a un enrutador sobre geografía
Echo	Pregunta a una máquina si está viva
Echo reply	Sí, estoy viva
Timestamp request	Misma que solicitud de eco, pero con marca de tiempo
Timestamp reply	Misma que respuesta de eco, pero con marca de tiempo

Figura 16: Los principales tipos de mensaje ICMP.

en realidad éstas no pueden usarse para enviar paquetes porque el hardware de capa de enlace de datos no entiende las direcciones de Internet. La pregunta ahora es: ¿cómo se convierten direcciones IP en direcciones de capa de enlace de datos, como Ethernet? El protocolo utilizado para hacer esta pregunta y obtener la respuesta se llama Protocolo de Resolución de Direcciones (ARP).

La ventaja de usar ARP en lugar de archivos de configuración es la sencillez. El gerente de sistemas sólo tiene que asignar a cada máquina una dirección IP y decidir respecto de las máscaras de subred. ARP hace el resto.

RARP, BOOTP y DHCP

ARP resuelve el problema de encontrar qué dirección Ethernet corresponde a una dirección IP dada. A veces se tiene que resolver el problema inverso: dada una dirección Ethernet, ¿cuál es la dirección IP correspondiente? En particular, este problema ocurre cuando se inicializa una estación de trabajo sin disco. Dicha máquina recibirá normalmente la imagen binaria de su sistema operativo desde un servidor de archivos remoto. ¿Pero cómo aprende su dirección IP?

La primera solución inventada fue usar el **RARP** (*Reverse Address Resolution Protocol*: **Protocolo de Resolución de Dirección Inverso**). Este protocolo permite que una estación de trabajo recientemente inicializada transmita su dirección Ethernet y diga: “Mi dirección Ethernet de 48 bits es $14.04.05.18.01.25$. ¿Alguien allá afuera conoce mi dirección IP?” El servidor RARP ve esta solicitud, busca la dirección Ethernet en sus archivos de configuración y devuelve la dirección IP correspondiente.

Una desventaja de RARP es que usa una dirección de destino de todos los 1s (de difusión limitada) para llegar al servidor RARP. Sin embargo, dichas difusiones no las envían los enrutadores, por lo que se necesita un servidor RARP en cada red. Para resolver este problema, se inventó un protocolo de arranque alternativo llamado **BOOTP**. A diferencia de RARP, el BOOTP usa mensajes UDP que se envían a través de los enrutadores. También proporciona información adicional a una estación de trabajo sin disco, incluso la dirección IP del servidor de archivos que contiene la imagen de memoria, la dirección IP del enrutador predeterminado y la máscara de subred que debe usar.

Un problema serio con BOOTP es que requiere configuración manual de tablas para relacionar una dirección IP con una dirección Ethernet. Cuando se agrega un nuevo host a una LAN, no se puede usar el BOOTP hasta que un administrador le haya asignado una dirección IP e introducido manualmente sus direcciones IP y Ethernet en las tablas de configuración de BOOTP. Para eliminar este paso conducente a errores, el BOOTP se extendió y se le dio un nuevo nombre: **DHCP** (**Protocolo de Configuración de Host Dinámico**). DHCP permite asignación de dirección IP manual y automática.

Como RARP y BOOTP, DHCP se basa en la idea de un servidor especial que asigna direcciones IP a hosts que las requieren. Este servidor no necesita estar en la misma LAN que el host solicitante. Puesto que el servidor DHCP no se puede alcanzar por difusión, se necesita un agente de retransmisión DHCP en cada LAN. Para encontrar su dirección IP, una máquina inicializada recientemente difunde un paquete DHCP DISCOVER. El agente de retransmisión DHCP de su LAN intercepta todas las difusiones DHCP. Cuando encuentra un paquete DHCP DISCOVER, envía el paquete mediante unidifusión al servidor DHCP, posiblemente en una red distante. La única pieza de información que el agente de retransmisión necesita es la dirección IP del servidor DHCP.

5.4. OSPF - Protocolos de Enrutamiento de Puerta de Enlace Interior

Un algoritmo de enrutamiento dentro de un sistema autónomo se llama **protocolo de puerta de enlace interior (IGP)**; un algoritmo para enrutamiento entre sistemas autónomos se llama **protocolo de puerta de enlace exterior (EGP)**.

El protocolo de puerta de enlace interior original de Internet era un protocolo de vector de distancia (RIP). Funcionó bien en sistemas pequeños, pero no así conforme los sistemas autónomos fueron más grandes. También padeció el problema de la cuenta hasta el infinito y la convergencia generalmente lenta, por lo que se reemplazó en mayo de 1979 por un protocolo de estado del enlace. Ese sucesor, llamado **OSPF (Abrir Primero la Ruta más Corta)**, se volvió una norma en 1990. Ahora la mayoría de vendedores de enrutadores lo apoyan, y se ha convertido en el protocolo de puerta de enlace interior principal.

Dada la larga experiencia con otros protocolos de enrutamiento, el grupo que diseñó el nuevo protocolo tenía una larga lista de requisitos que cumplir. Primero, el algoritmo se tenía que publicar en la literatura abierta, de ahí la "O" inicial de OSPF. Segundo, el nuevo protocolo tenía que apoyar una variedad de métrica de distancia, como la distancia física, retardo, etcétera. Tercero, tenía que ser un algoritmo dinámico, uno que se adaptara automática y rápidamente a los cambios de topología.

Cuarto, y esto era nuevo para OSPF, tenía que apoyar el enrutamiento con base en el tipo de servicio. El nuevo protocolo tenía que poder dirigir el tráfico en tiempo real de una manera y el resto del tráfico de una manera diferente. Quinto, y relacionado con el anterior, el nuevo protocolo tenía que balancear la carga, dividiéndola en líneas múltiples. La mayoría de los protocolos anteriores enviaba todos los paquetes por la mejor ruta.

Sexto, se necesitó apoyo para los sistemas jerárquicos. En 1988 Internet había crecido tanto que no se podía esperar que ningún enrutador conociera toda la topología. Se tuvo que diseñar el nuevo protocolo de enrutamiento para que ningún enrutador tuviera que conocerla. Séptimo, se requirió una pizca de seguridad para impedir que los estudiantes bromistas engañaran a los enrutadores enviándoles falsa información de enrutamiento. Finalmente, se necesitó una previsión para tratar con enrutadores que se conectaban a Internet mediante un túnel. Los protocolos anteriores no manejaron bien este aspecto.

OSPF soporta tres tipos de conexiones y redes:

- Las líneas punto a punto exactamente entre dos enrutadores.
- Redes de multiacceso con difusión (por ejemplo, la mayoría de las LANs).
- Redes de multiacceso sin difusión (por ejemplo, la mayoría de las WANs de paquetes conmutados).

OSPF funciona resumiendo la colección de redes reales, enrutadores y líneas en un grafo dirigido en el que a cada arco se asigna un costo (distancia, retardo, etcétera). Entonces calcula la ruta más corta con base en los pesos de los arcos. Una conexión en serie entre dos enrutadores se representa por un par de arcos, uno en cada dirección. Sus pesos pueden ser diferentes. Una red de multiacceso se representa con un nodo para la red en sí más un nodo para cada enrutador. Los arcos del nodo de la red a los enrutadores tienen peso 0 y se omiten del grafo.

OSPF distingue cuatro clases de enrutadores:

- Enrutadores internos que están totalmente dentro de un área.
- Enrutadores de límite de área que conectan dos o más áreas.
- Enrutadores de la red dorsal que están en la red dorsal.
- Enrutadores fronterizos de sistemas autónomos que se comunican con los enrutadores de otros sistemas autónomos.

OSPF trabaja intercambiando información entre enrutadores adyacentes, que no es lo mismo que entre enrutadores vecinos. En particular, es ineficaz tener cualquier enrutador en la LAN que se comunica con cualquier otro enrutador en la LAN. Para evitar esta situación, se elige un enrutador como **enrutador designado**. Se dice que es **adyacente** a todos los demás enrutadores en su LAN, e intercambia información con ellos. Los enrutadores vecinos que no son adyacentes no intercambian información entre sí.

5.5. IPv6

Si bien el CIDR y el NAT pueden durar unos pocos años más, todo mundo se da cuenta que los días del IP en su forma actual (Ipv4) están contados. Con la explosión del interés por la Internet que comenzó a mediados de la década de 1990, es muy posible que en el próximo milenio será usada por un grupo mucho más grande de gente, especialmente gente con necesidades diferentes. Por una parte, millones de personas con computadoras portátiles inalámbricas podrían usarla para mantenerse en contacto con sus bases. Por otra, con la inminente convergencia de las industrias de la computación, las comunicaciones y el entretenimiento, tal vez no pase mucho tiempo ante de que todos los teléfonos y los televisores del mundo estén en un nodo Internet, produciendo mil millones de máquinas utilizadas para recibir audio y vídeo bajo demanda. En estas circunstancias, se hizo evidente que el IP tenía que evolucionar y volverse más flexible.

Al ver en el horizonte estos problemas, la IETF comenzó a trabajar en 1990 en una versión nueva del IP, una que nunca se quedaría sin direcciones, resolvería varios otros problemas y sería más flexible y eficiente también. Sus metas principales eran:

- Manejar miles de millones de hosts, aún con asignación de espacio de direcciones ineficiente.

- Reducir el tamaño de las tablas de enrutamiento.
- Simplificar el protocolo, para permitir a los enrutadores el procesamiento más rápido de los paquetes.
- Proporcionar mayor seguridad (verificación de autenticidad y confidencialidad) que el IP actual.
- Prestar mayor atención al tipo de servicio, especialmente con datos en tiempo real.
- Ayudar a la multidifusión permitiendo la especificación de alcances.
- Posibilitar que un host sea móvil sin cambiar su dirección.
- Permitir que el protocolo evolucione.
- Permitir que el protocolo viejo y el nuevo coexistan por años.

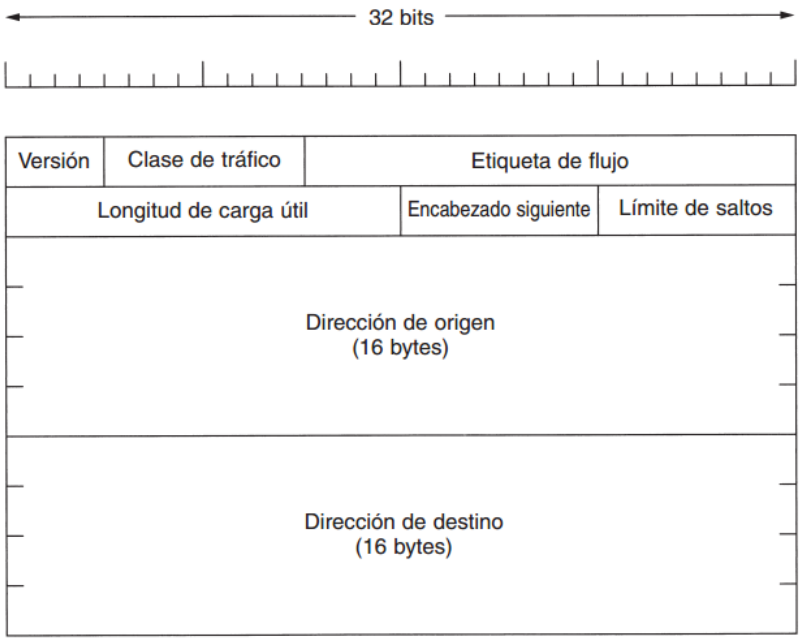
El **IPv6** cumple los objetivos bastante bien: mantiene las buenas características del IP, descarta y reduce las malas, y agrega nuevas donde se necesitan. En general, IPv6 no es compatible con IPv4, pero es compatible con todos los demás protocolos Internet, incluidos TCP, UDP, ICMP, IGMP, OSPF, BGP y DNS, a veces con algunas pequeñas modificaciones (principalmente para manejar direcciones más grandes).

Por principio, y lo más importante, el IPv6 tiene direcciones más grandes que el IPv4; son de 16 bytes de longitud, lo que resuelve el problema que se buscaba resolver: proporcionar una cantidad prácticamente ilimitada de direcciones Internet. La segunda mejora principal del IPv6 es la simplificación del encabezado, que contiene sólo 7 campos. Este cambio permite a los enrutadores procesar con mayor rapidez los paquetes y mejorar, por tanto, la velocidad real de transporte. También estudiaremos pronto el encabezado. La tercera mejora importante fue el mejor apoyo de las opciones. Este cambio fue esencial con el nuevo encabezado, pues campos que antes eran obligatorios ahora son opcionales.

Una cuarta área en la que el IPv6 representa un avance importante es la seguridad. La IETF tenía infinidad de historias sobre preadolescentes precoces que usaban sus computadoras personales para meterse en bancos e instalaciones militares por todas partes de Internet. Por último, se ha puesto mayor atención en la calidad del servicio.

El encabezado principal del IPv6

El encabezado del IPv6 se muestra en la figura siguiente. El campo de *Versión* siempre es 6 para el IPv6 (y de 4 para el IPv4). Durante el periodo de transición del IPv4 al IPv6, que probablemente se llevará una década, los enrutadores podrán examinar este campo para saber el tipo de paquete que tienen.



El campo *Clase de tráfico* se usa para distinguir entre los paquetes con requisitos diferentes de entrega en tiempo real. Un campo diseñado para este propósito ha estado en el IP desde el principio, pero los enrutadores lo han implementado sólo esporádicamente.

El campo de *Etiqueta de flujo* aún es experimental, pero se usará para permitir a un origen y a un destino establecer una pseudoconexión con propiedades y requisitos particulares. El flujo puede establecerse por adelantado, dándole un identificador. Cuando aparece un paquete con una Etiqueta de flujo diferente de cero, todos los enrutadores pueden buscarla en sus tablas internas para ver el tipo de tratamiento especial que requiere. En efecto, los flujos son un intento de tener lo mejor de

ambos mundos: la flexibilidad de una subred de datagramas y las garantías de una subred de circuitos virtuales.

El campo de *Longitud de carga útil* indica cuántos bytes siguen al encabezado de 40 bytes El nombre de campo se cambió de Longitud total en el IPv4 porque el significado cambió ligeramente: los 40 bytes del encabezado ya no se cuentan como parte de la longitud, como antes.

El campo *Encabezado siguiente* revela el secreto. La razón por la que pudo simplificarse el encabezado es que puede haber encabezados adicionales (opcionales) de extensión. Este campo indica cuál de los seis encabezados de extensión (actualmente), de haberlos, sigue a éste.

El campo de *Límite de saltos* se usa para evitar que los paquetes vivan eternamente. En la práctica es igual al campo de Tiempo de vida del IPv4, es decir, un campo que se disminuye en cada salto. En teoría, en el IPv4 era un tiempo en segundos, pero ningún enrutador lo usaba de esa manera, por lo que se cambió el nombre para reflejar la manera en que se usa realmente.

Luego vienen los campos de *Dirección de origen* y *Dirección de destino.* Por último, el campo de *Suma de verificación* desaparece, porque su cálculo reduce en gran medida el desempeño.

Encabezados de extensión

Con todo, algunos de los campos faltantes del IPv4 en ocasiones son necesarios, por lo que el IPv6 introdujo el concepto de encabezado de extensión (opcional). Estos encabezados pueden usarse para proporcionar información extra, pero codificada de una manera eficiente. Hay seis tipos de encabezados de extensión definidos actualmente los cuales se muestran a continuación:

Encabezado de extensión	Descripción
Opciones salto por salto	Información diversa para los enrutadores
Opciones de destino	Información adicional para el destino
Enrutamiento	Ruta total o parcial a seguir
Fragmentación	Manejo de fragmentos de datagramas
Autenticación	Verificación de la identidad del emisor
Carga útil de seguridad encriptada	Información sobre el contenido encriptado

Referencias

[1] A. S. Tanenbaum, *Redes de Computadoras*, 4th ed., 2003.

[2] Apuntes de Cátedra, *Redes y Comunicaciones de Datos II*, 2016.