Redes y Comunicaciones de Datos II

Cristian Escudero
Resumen

April 26, 2013

1 La Capa de Red

La **capa de red** (CdR) se encarga de llevar los paquetes desde el origen hasta el destino, lo que puede requerir muchos saltos por enrutadores intermedios. Contrasta con la **capa de enlace de datos** (CED), cuya función es la de mover tramas de un extremo del cable al otro. La CdR es la capa más baja que maneja la transmisión de **extremo a extremo**.

La CdR debe:

- Conocer la **topología** de la *subred* de comunicación y elegir las rutas adecuadas a través de ella.
- Tener cuidado al escoger las rutas para no sobrecargar algunas de las líneas de comunicación y los enrutadores y dejar inactivos a otros.
- Cuando el *origen* y el *destino* están en redes diferentes, se encarga de solucionar los problemas que surjan.

1.1 Aspectos de diseño de la capa de Red

1.1.1 Servicios proporcionados a la capa de Transporte

Objetivos de los servicios proporcionados por la CdR:

1. Deben ser independientes de la tecnología del enrutador.
2. La **capa de transporte** (CdT) debe estar **aislada** de la cantidad, tipo y topología de los enrutadores presentes.
3. Las direcciones de red disponibles para la CdT deben seguir un plan de numeración uniforme, aun a través de varias LANs y WANs.

1.1.2 Implementación del servicio no orientado a la conexión

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. Los paquetes son referidos como **datagramas** y la subred se conoce como **subred de datagramas**.

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 **circuito virtual** (CV), y la subred se conoce como **subred de CVs**.

El algoritmo que maneja las tablas y que realiza las decisiones de enrutamiento se conoce como **algoritmo de enrutamiento**.

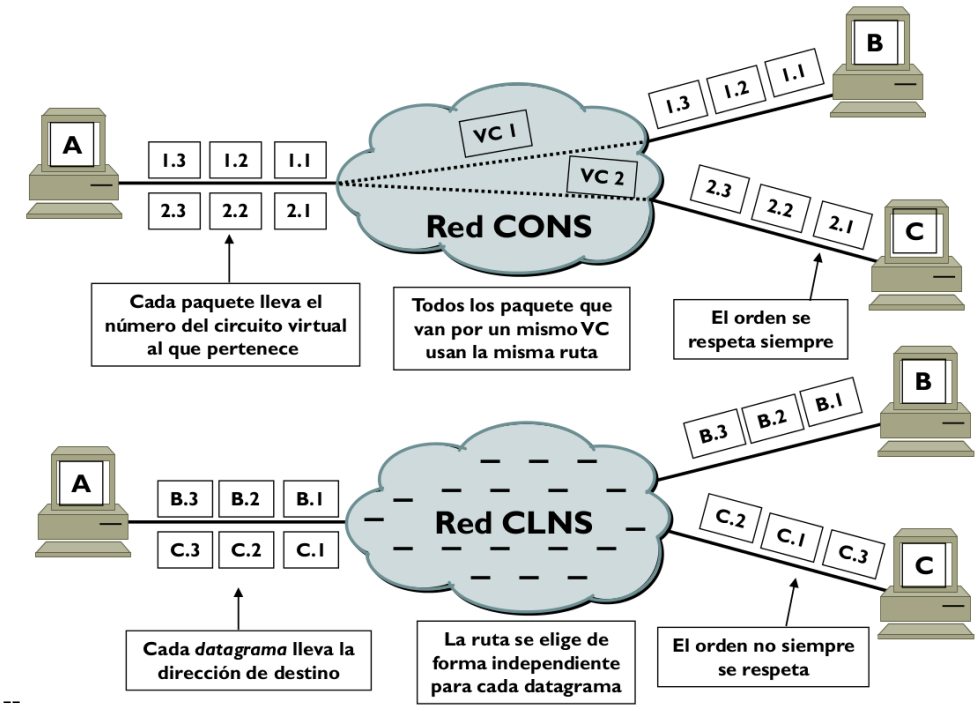


Figure 1: Diferencia gráfica entre servicios orientados y no orientados en la CdR.

1.1.3 Implementación del servicio orientado a la conexión

Necesitamos una subred de CVs para evitar la necesidad de elegir una nueva ruta para cada paquete enviado. 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. Cuando se libera la conexión, también se termina el CV. Cada paquete lleva un **identificador** que indica a cuál CV pertenece.

Los enrutadores requieren la capacidad de reemplazar identificadores de conexión en los paquetes salientes (**conmutación de etiquetas**).

1.1.4 Comparación entre las subredes de CVs y las de datagramas

Dentro de la subred hay varios pros y contras entre los CVs y los datagramas (ver **Tabla 1**):

Espacio de memoria del enrutador y el ancho de banda: Los CVs permiten que los paquetes contengan números de CV 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 CVs es el espacio de tabla en los enrutadores. La mejor elección desde el punto de vista **económico** depende del **costo relativo** entre los circuitos de comunicación y la memoria de los enrutadores.

Vulnerabilidad: En los CVs, si se cae un enrutador y se pierde su memoria, todos los CVs 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 CVs que la usan, pero puede compensarse fácilmente cuando se usan datagramas. Éstos también permiten que los enrutadores equilibren el tráfico a través de la subred, ya que las rutas pueden cambiarse a lo largo de una secuencia larga de transmisiones de paquetes.

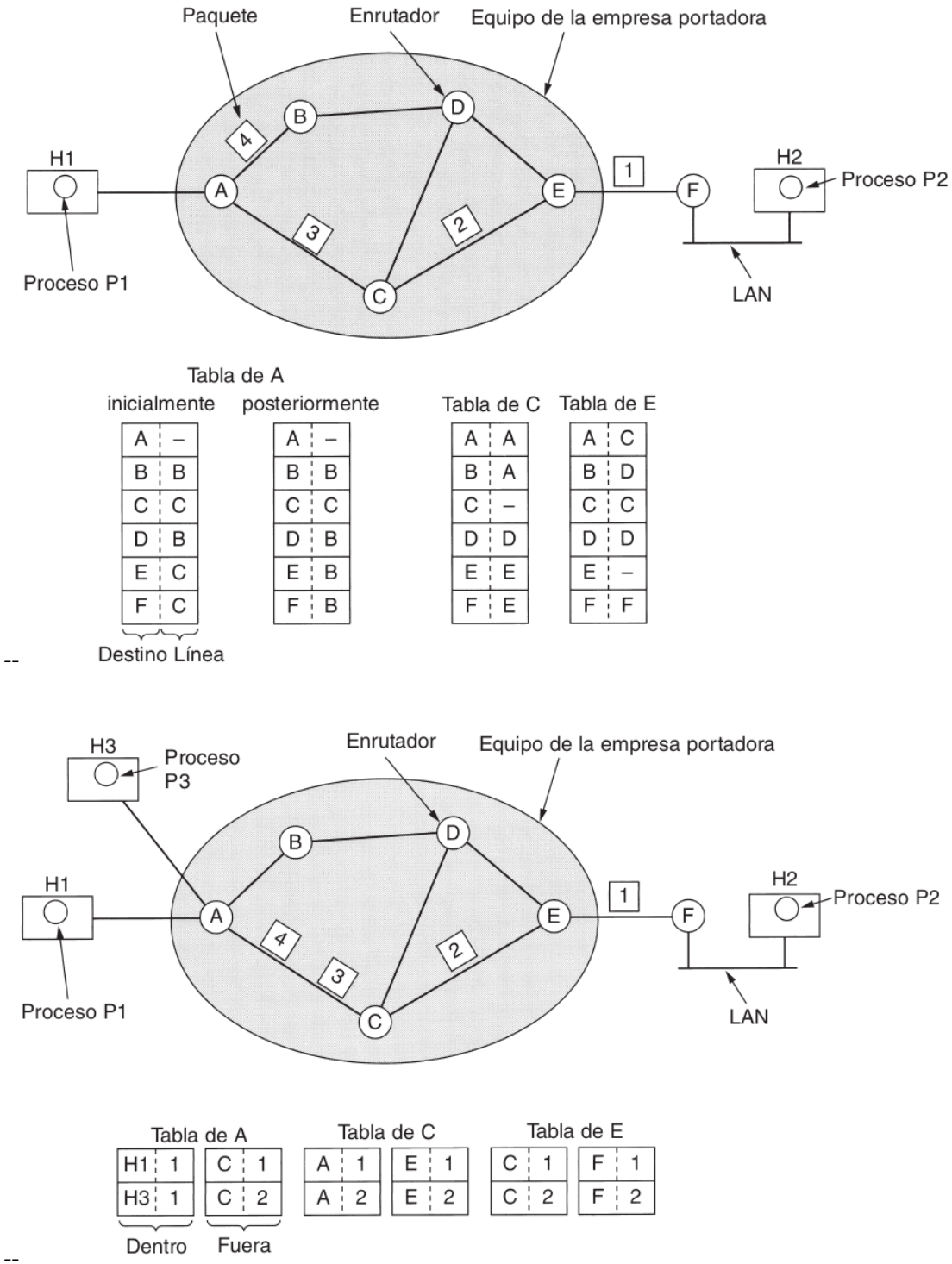


Figure 2: Enrutamiento dentro de una: subred de datagramas (*arriba*), subred de CVs (*abajo*).

1.2 Algoritmos de Enrutamiento

El **algoritmo de enrutamiento** es aquella parte del software de la CdR encargada de decidir la línea de salida por la que se transmitirá un paquete de entrada. Si la subred de manera interna **usa**:

- **Datagramas**: esta decisión debe tomarse cada vez que llega un paquete de datos, dado que la mejor ruta podría haber cambiado desde la última vez;
- **CVs**: las decisiones de enrutamiento se toman sólo al establecerse un CV nuevo. En lo sucesivo, los paquetes de datos simplemente siguen la ruta previamente establecida.

Asunto	Subred de datagramas	Subred de CVs
Configuración del circuito	No necesaria, pero requiere un procedimiento más complicado para localizar el destino del paquete.	Requiere una fase de configuración, que consume tiempo y recursos.
Direccionamiento	Cada paquete contiene la dirección de origen y de destino.	Cada paquete contiene un número de CV corto.
Información de estado	Los enrutadores no contienen información de estado de las conexiones.	Cada CV requiere espacio de tabla del enrutador por conexión.
Enrutamiento	Cada paquete se enruta de manera independiente.	Ruta escogida cuando se establece el CV; todos los paquetes siguen esta ruta.
Efecto de fallas del enrutador	Ninguno, excepto para paquetes perdidos durante una caída.	Terminan todos los CVs que pasan a través del enrutador.
QoS y Control de Congestión	Difícil de llevar a cabo.	Fácil si se pueden asignar suficientes recursos por adelantado para cada CV.

Table 1: Comparación de las subredes de datagramas y de CVs.

Se puede considerar que un enrutador realiza dos procesos internos:

Enrutamiento: proceso de tomar la decisión de cuáles rutas utilizar. Es responsable de llenar y actualizar las tablas de enrutamiento.

Reenvío: acción que se toma cuando llega un paquete. Maneja cada paquete conforme llega, buscando en las tablas de enrutamiento la línea de salida por la cual se enviará.

Hay ciertas propiedades que todo *algoritmo de enrutamiento* debe poseer:

- **Exactitud.**
- **Sencillez.**
- **Robustez.** 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.
- **Estabilidad.** Debe alcanzar el equilibrio y conservarlo.
- **Equidad.**
- **Optimización.**

La *equidad* y la *optimización* resulta que con frecuencia son metas contradictorias. Antes de poder encontrar un punto medio entre ambos, debemos decidir **qué es lo que buscamos optimizar**. 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:

No adaptativos (o estáticos). Las decisiones de enrutamiento se toman en base a información recopilada con anterioridad (*horas, días o meses*). Normalmente el cálculo de la ruta es costoso y se realiza de forma centralizada. Por eso una vez fijada la ruta raramente se cambia. No toman en cuenta la carga actual de la red.

Adaptativos (o dinámicos). Cambian sus decisiones de enrutamiento para reflejar los cambios de topología y, por lo general también el tráfico (en tiempo real). Difieren entre sí en:

- el lugar de donde obtienen su información;
- el momento de cambio de sus rutas;
- la métrica usada para la optimización.

1.2.1 Principio de optimización

Este 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:
  - + número de "saltos";
  - + congestión de los enlaces;
  - + retardo de los enlaces.
- Maximizar:
  - + ancho de banda de los enlaces;
  - + fiabilidad de los enlaces (minimizar BER).
- Una determinada combinación ponderada de todos los anteriores según los gustos del usuario.

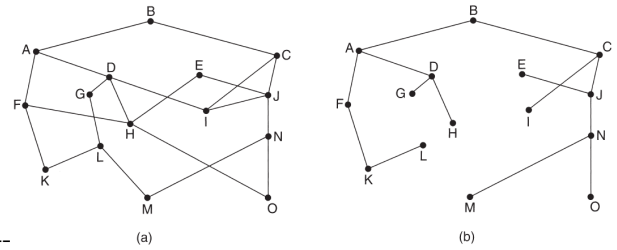


Figure 3: (a) Una subred. (b) Árbol sumidero para el enrutador B.

Arbol sumidero: árbol formado por el grupo de rutas óptimas de todos los orígenes a un destino dado, con raíz en el destino. No contiene ciclos, por lo que cada paquete será entregado en un número de saltos finito y limitado.

La meta de todos los algoritmos de enrutamiento es descubrir y utilizar los árboles sumideros de todos los enrutadores.

1.2.2 Enrutamiento por la ruta más corta (algoritmo estático)

Para elegir una ruta entre un par dado de enrutadores, el algoritmo simplemente encuentra en el grafo la **ruta más corta** entre ellos. Como determina esta ruta depende de la métrica utilizada. Ver **Figura 5-7** de [TANEM], página 354, para un ejemplo gráfico del algoritmo.

1.2.3 Inundación (algoritmo estático)

Cada paquete de entrada se envía por cada una de las líneas de salida, excepto aquella por la que llegó. Hay que poner medidas para controlar el tráfico infinito de un paquete (por ejemplo, utilizar número de secuencia y un tiempo de vida según un contador de saltos).

Una variación del algoritmo es la **inundación selectiva**: 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, a menos que la topología sea extremadamente peculiar y que el enrutador esté seguro de este hecho.

La inundación no es práctica en la mayoría de las aplicaciones, pero tiene algunos usos (en el área militar por ejemplo, ante posibles pérdidas múltiples de enrutadores). Siempre escoge la ruta más corta posible, porque escoge en paralelo todas las rutas posibles. En consecuencia, ningún otro algoritmo puede producir un retardo más corto.

1.2.4 Enrutamiento por vector de distancia (algoritmo dinámico)

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 paquetes de información con los vecinos, que indican la distancia a cada posible destino. Se supone que el enrutador conoce la "*distancia*" a cada uno de sus vecinos.

Su principal virtud es la sencillez del algoritmo, que permite hacer los cálculos con poca CPU y poca memoria en el enrutador. Es utilizado por el protocolo **RIP** (*Routing Information Protocol*).

El problema de la cuenta hasta infinito:

Este enrutamiento 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. La **esencia del problema** consiste en que cuando el enrutador X indica a Y que tiene una ruta en algún lugar, Y no tiene forma de saber si él mismo está en esa ruta.

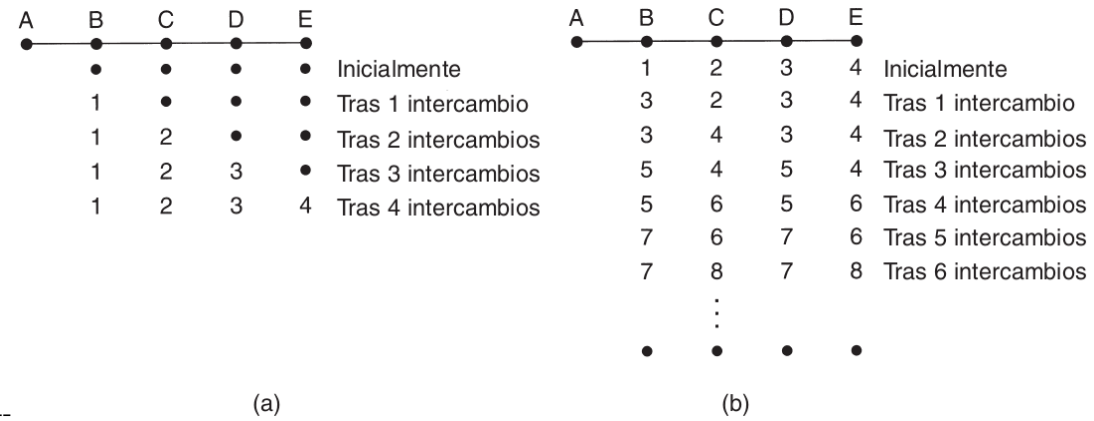


Figure 4: El problema de la cuenta hasta infinito. Los números indican la distancia en saltos hacia el enrutador A. En el caso (a), el enrutador A se activa; en el caso (b), todos los enrutadores están activados y de repente el enrutador A se desactiva.

El enrutamiento por vector de distancia es útil cuando se utiliza una métrica sencilla, como por ejemplo el número de saltos, donde el problema de la cuenta hasta infinito es más fácil de resolver.

1.2.5 Enrutamiento por estado del enlace

Dos problemas principales causaron la **desaparición** del *enrutamiento por vector de distancia*: no tomaba en cuenta el ancho de banda al escoger las rutas, y el problema de la cuenta hasta el infinito. Por eso, fue reemplazado por este nuevo tipo de enrutamiento, más fiable y eficiente.

El concepto en el que se basa es sencillo y puede enunciarse en cinco partes. Cada enrutador debe:

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

Toda la topología y todos los retardos se miden experimentalmente y se distribuyen a cada enrutador.

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**.

Cuando se conectan dos o más enrutadores mediante una LAN, esta se puede modelar considerandola como otro nodo.

Medición del costo de la línea:

El algoritmo requiere que cada enrutador sepa aproximadamente 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. Al medir el tiempo de ida y vuelta y dividirlo entre dos, el enrutador emisor puede tener una idea razonable del retardo.

La prueba puede llevarse a cabo varias veces y usarse el promedio para mejores resultados. Esto asume de forma implícita que los retardos son **simétricos**.

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.

Para evitar **oscilaciones** en la selección de la mejor ruta (si se *considera* la carga, ver **figura 5-12** de [TANEM], página 362), podría ser adecuado dividir la carga entre múltiples líneas, con una fracción conocida de la carga viajando sobre cada una de ellas.

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 ha de determinar **cuándo** construirlos: de manera periódica a intervalos regulares; ó cuando ocurra un evento significativo (caída o reactivación de una línea o de un vecino, por ejemplo).

Distribución de los paquetes de estado del enlace:

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.

Pueden presentarse algunos problemas:

1. Los números de secuencia vuelven a comenzar. *Solución*: usar un número de secuencia de `32 bits`.
2. 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.
3. 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 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.

Para hacerlo **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. 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.

1.2.6 Enrutamiento jerárquico

Problema: los algoritmos de enrutamiento no son **escalables**. La cantidad de información intercambiada aumenta de forma **no lineal** con el tamaño de la red. En la misma medida aumentan la complejidad de los cálculos, los requerimientos de CPU y memoria en los 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.

Solución: crear **regiones** (niveles *jerárquicos*), 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. Solo algunos enrutadores de cada región comunican con el exterior. Las rutas son menos óptimas, pero se reduce la información de enrutamiento.

¿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.

1.2.7 RIP - Routing Information Protocol

Protocolo de puerta de enlace interna, que utiliza el protocolo de enrutamiento por **vector de distancia**.

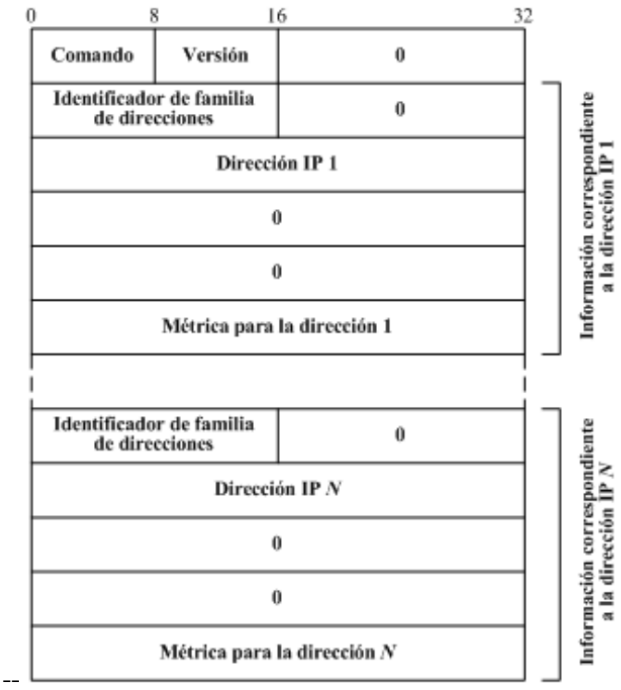


Figure 5: Formato del paquete RIP.

Modo de operación:

1. Envía un mensaje a todos sus vecinos, utilizando el puerto 520, pidiendo una copia de la tabla del vecino. El vecino responde con su tabla completa.
2. Cuando esta activo, envía su tabla por *broadcast*, cada 30 seg.
3. ¿Una métrica cambia? $\Rightarrow$ la difunde.

1.2.8 OSPF - Open Shortest Path First

Utiliza el protocolo de enrutamiento por **estado de enlace**. Debe cumplir los siguientes requisitos:

- Ser abierto (*open*).
- Ser un algoritmo dinámico.
- Apoyar una variedad de métricas.
- Apoyar el enrutamiento con base en el tipo de servicio.
- Balancear la carga, dividiéndola en líneas múltiples.
- Implementar un esquema de seguridad.

Las categorías utilizadas para **determinar la métrica** son las siguientes:

- Normal (`TOS 0`). Métrica por defecto.
- Minimizar el costo monetario (`TOS 2`).
- Maximizar la confiabilidad (`TOS 4`).
- Maximizar el rendimiento real (`TOS 8`).
- Minimizar el retardo (`TOS 15`).

Los **paquetes** que utiliza son:

- `Hello`: Descubre quiénes son los vecinos.
- `Link State Update`: Proporciona los costos del emisor a sus vecinos.
- `Link State Ack`: Confirma la recepción de la actualización del estado.
- `Database Description`: Anuncia qué actualizaciones tiene el emisor.
- `Link State Request`: Solicita información del socio.

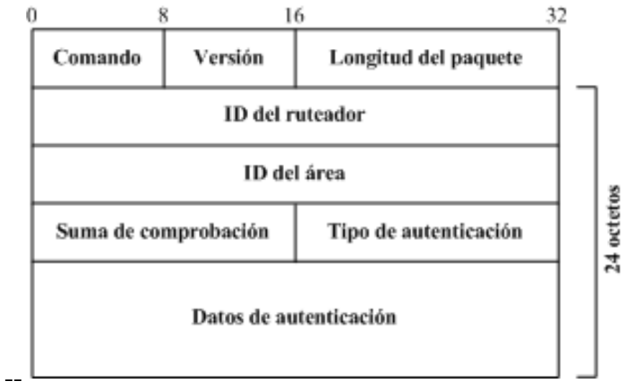


Figure 6: Formato del paquete OSPF.

1.3 Algoritmos de Control de Congestión

La **congestión** se produce cuando, al haber demasiados paquetes presentes en la subred (o en una parte de ella), hay una degradación del desempeño. Con mucho tráfico puede que haya incluso una pérdida completa de la capacidad de entregar paquetes (ver **Figura 7**).

La congestión puede ocurrir por varias razones. Si de manera repentina comienzan a llegar cadenas de paquetes hacia un **enrutador con varias puertas y una sola salida**, se generará una cola. Si no hay suficiente memoria para almacenar a todos los paquetes, algunos de ellos se descartarán. Incluso con una cantidad infinita de memoria la congestión empeora, ya que para cuando los paquetes llegan al principio de la cola, su temporizador ha terminado (repetidamente) y se han enviado duplicados.

Los **procesadores lentos**, las líneas de **poco ancho de banda** y la **inestabilidad** de enrutadores también pueden causar congestión.

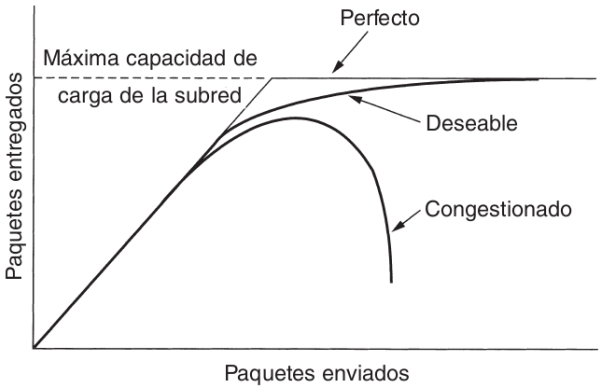


Figure 7: Cuando se genera demasiado tráfico, ocurre congestión y se degrada marcadamente el desempeño.

Control de congestión y control de flujo:

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.

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. Implica normalmente una **retroalimentación** directa del receptor al emisor.

1.3.1 Principios generales del control de congestión

Las redes de computadoras pueden analizarse desde el punto de vista de una **teoría de control**, que conduce a dividir en **dos grupos** a todas las soluciones:

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. Se clasifican entre los que **actúan en el origen** ó en **el destino**. Todas toman decisiones **independientemente** del estado actual de la red.

Ciclo cerrado. Se basan en el concepto de un **ciclo de retroalimentación**. Tiene tres partes:

1. Monitorear el sistema.
2. Enviar esta información monitoreada.
3. Ajustar la operación del sistema.

Los algoritmos de *ciclo cerrado* se dividen en dos categorías:

- **Explícitos.** Regresan paquetes desde el punto de congestión para avisar al origen.
- **Implícitos.** El origen deduce la existencia de una congestión haciendo observaciones locales.

Métricas utilizadas para monitorear la subred en busca de congestiones:

- 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 nuevamente.
- 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**. La presencia de congestión significa que la carga es, *temporalmente*, superior (en una parte del sistema) a la que pueden manejar los recursos.

1.3.2 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 8**).

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

Figure 8: Políticas relacionadas con la congestión.

1.3.3 Control de congestión en subredes de CVs

Control de admisión:

Una vez que se ha detectado la congestión, **no se establecen CVs nuevos** hasta que ha desaparecido el problema.

Un método alterno es **permitir la creación de nuevos CVs**, pero enrutando cuidadosamente los circuitos nuevos por otras rutas que no tengan problemas (**Figura 9**).

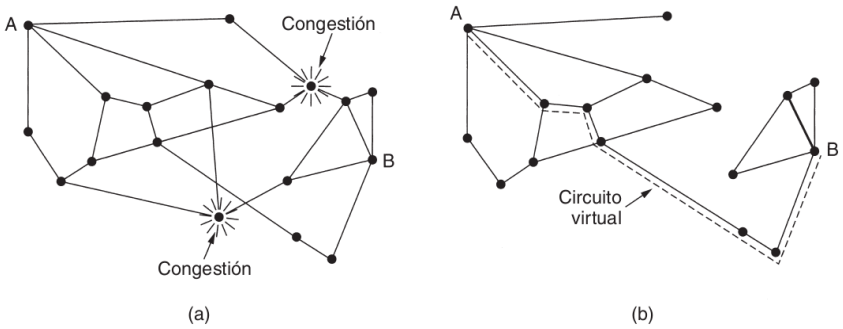


Figure 9: (a) Subred congestionada. (b) Subred redibujada que elimina la congestión.

Otra estrategia es **negociar un acuerdo** entre el *host* y la subred cuando se establece un CV. Para cumplir con su parte del acuerdo, la subred reservará recursos a lo largo de la ruta cuando se establezca el circuito. La disponibilidad de todos los recursos necesarios hace poco probable la congestión.

1.3.4 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[1] 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.

Paquetes reguladores de salto por salto

A altas velocidades o distancias grandes, el envío de un paquete regulador a los *hosts* de origen no funciona bien porque la **reacción es muy lenta**. Este método hace que el paquete regulador ejerza su efecto en **cada salto** que da.

Proporciona un alivio rápido al punto de congestión, a expensas de usar más búferes ascendentes. De esta manera puede cortarse la congestión en la raíz, sin que se pierdan paquetes (para representación gráfica, ver **figura 5-28** de [TANEM], página 393).

1.3.5 Desprendimiento de carga

Si el enrutador no puede manejar las cargas, descarta los paquetes y que las capas superiores se encarguen de enviarlos nuevamente.

Un enrutador abrumado por paquetes simplemente puede escoger paquetes al azar para desprenderse de ellos. Un método alterno es utilizar el descarte según las aplicaciones que se estén ejecutando: en la **transferencia de archivos** vale más un paquete viejo que uno nuevo; en **multimedia** es al revés.

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**

[1] Se activa un `bit` del encabezado.

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.

1.3.6 Control de fluctuación

La **variación** en el retardo de los paquetes se conoce como **fluctuación**. Cuando tenemos una fluctuación **alta** (10[mseg] de diferencia entre paquetes, por ejemplo), se produce un degradado de calidad en las aplicaciones de *streaming* en tiempo real.

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.

1.4 Calidad del Servicio

El **flujo** es el conjunto de paquetes que van de un origen a un destino. Se puede caracterizar por cuatro parámetros principales: **confiabilidad**, **retardo**, **fluctuación** y **ancho de banda**. En conjunto determinan la QoS 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

Figure 10: Qué tan rigurosos son los requerimientos de calidad del servicio.

1.4.1 Técnicas para alcanzar una buena calidad de servicio

Ninguna 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 se 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:

- **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 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 CVs que en las de datagramas.

Se utiliza para video y audio en tiempo real.

Algoritmo de cubeta con goteo:

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.

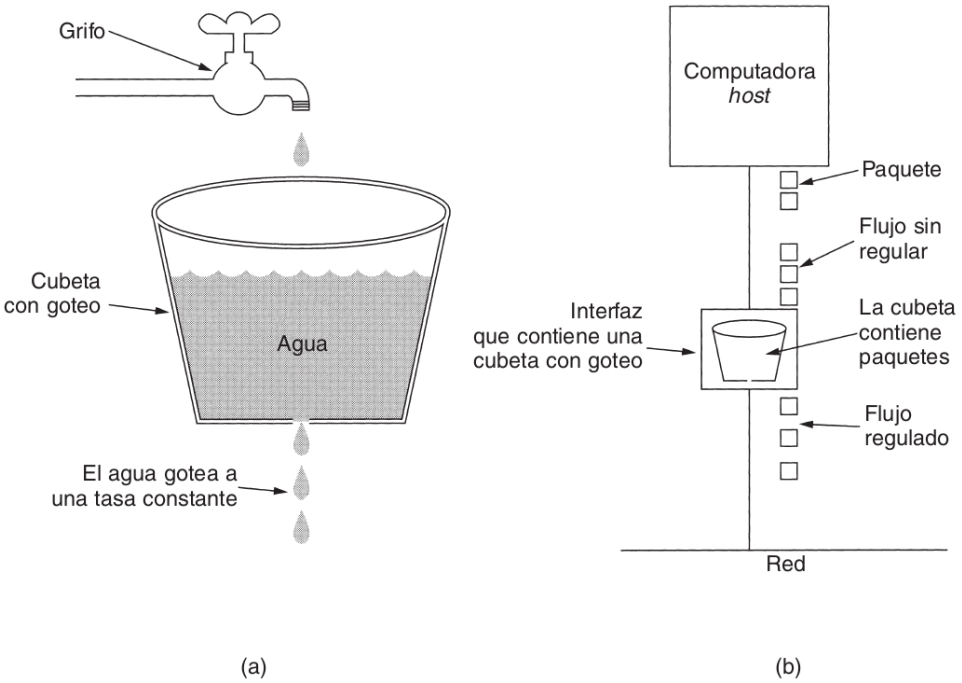


Figure 11: (a) Una cubeta con goteo, llena de agua. (b) Cubeta con goteo, llena de paquetes.

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*:

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 solo puede procesar cierta cantidad de paquetes por segundo.

Calendarización de paquetes:

Si un enrutador maneja múltiples flujos, existe el peligro de que un flujo acapare mucha de su capacidad y limite a los otros flujos.

Encolamiento justo. Los enrutadores tienen colas separadas para cada línea de salida, una por flujo. Cuando una línea se queda inactiva, el enrutador explora las diferentes colas de manera circular, y toma el primer paquete de la siguiente cola. De esta forma, con n *hosts* compitiendo por una línea de salida dada, cada *host* obtiene la oportunidad de enviar uno de n paquetes. El envío de más paquetes no mejorará esta fracción.

Problema: Proporciona más ancho de banda a los *hosts* que utilizan paquetes más grandes que a los que utilizan paquetes más pequeños.

Solución: Utilizar exploración circular `byte` por `byte` en vez de por paquete.

Encolamiento justo ponderado. Permite dar a ciertos *hosts* más ancho de banda que a otros, usando un sistema de prioridades.

1.4.2 Conmutación de etiquetas y MPLS (MultiProtocol Label Switching

Para desarrollar mejores métodos de reenvío, se puede agregar una **etiqueta** en frente de cada paquete y realizar el 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.

Enrutamiento: proceso de buscar una dirección de destino en una tabla para saber a dónde enviar los paquetes hacia ese destino.

Conmutación utiliza una etiqueta que se toma de un paquete como un índice en una tabla de reenvío.

Debido a que los paquetes IP no fueron diseñados para CVs, se tuvo que agregar un nuevo encabezado MPLS en frente del encabezado IP.

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.
- **TTL:** evita el ciclo infinito en caso de que haya inestabilidad en enrutamiento.

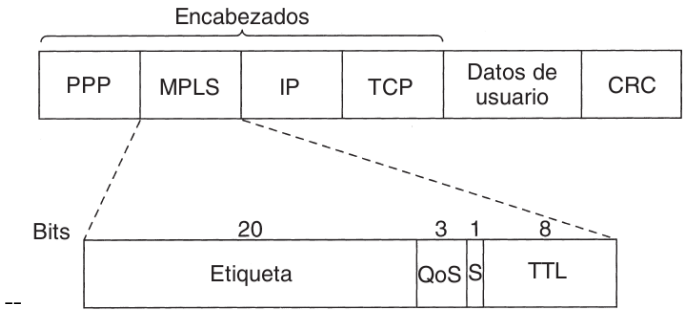


Figure 12: Transmisión de un segmento TCP que utiliza IP, MPLS y PPP.

Debido a que los encabezados MPLS no son parte del paquete de la CdR o de la trama de la CED, MPLS es en gran medida independiente de ambas capas: es **multiprotocolo**.

1.5 La Capa de Red de Internet

Los principios que guiaron su diseño en el pasado son (del más al menos importante):

1. **Asegúrese de que funciona.** No termine el diseño o estándar hasta que múltiples prototipos se hayan comunicado entre sí de manera exitosa.
2. **Mantenga la simplicidad.** Cuando tenga dudas, utilice la solución más simple.

Aplicación	Web (HTTP)	Transf. fich. (FTP)	e-mail (SMTP)	Resol. nombres (DNS)	Vídeo streaming	Telefonía
Transporte	TCP (Transmission Control Prot.)			UDP (User Datagram Prot.)		
Red	IP (Internet Protocol)					
Enlace	Ethernet		WiFi	ADSL	CATV	
Física	Cable o Fibra (1-1000 Mbps)		Radio 2,4 ó 5 GHz (1-54 Mbps)	Cable telefónico (0,5-25 Mbs)	Cable coaxial 50 Ω (30-40 Mbps)	

Figure 13: Situación de los protocolos de Internet en el modelo de capas.

3. **Elija opciones claras.** Si hay varias maneras para realizar la misma tarea, elija sólo una.
4. **Aproveche la modularidad.** Si las circunstancias requieren que un módulo o capa cambie, los otros no se verán afectados.
5. **Tenga en cuenta la heterogeneidad.** El diseño de la red debe ser simple, general y flexible.
6. **Evite las opciones y parámetros estáticos.**
7. **Busque un buen diseño; no es necesario que sea perfecto.** Para manejar algún caso especial, deje que esa parte la resuelvan las personas interesadas en él.
8. **Sea estricto cuando envíe y tolerante cuando reciba.**
9. **Piense en la escalabilidad.**
10. **Considere el desempeño y el costo.**

1.5.1 El Protocolo IP

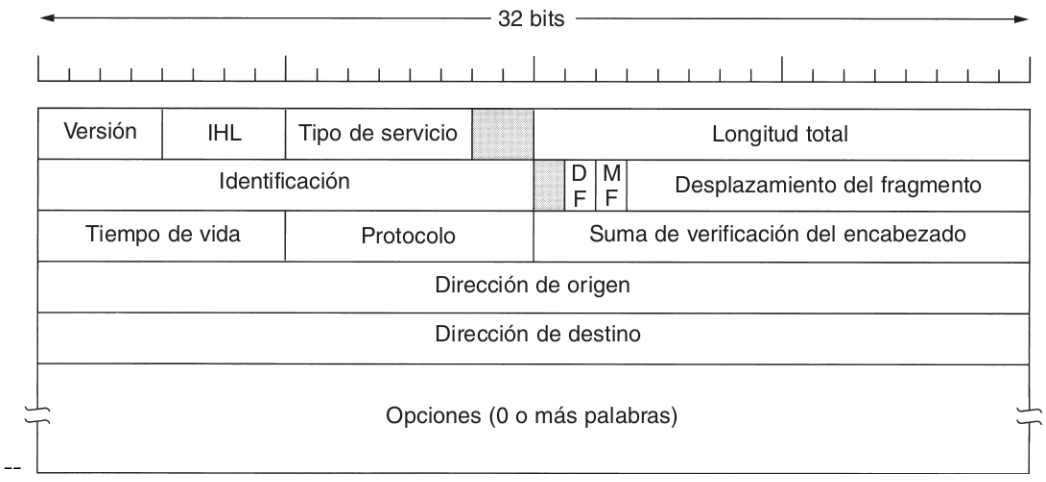


Figure 14: El encabezado de IPv4 (Protocolo Internet).

El formato de 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. Los campos son:

- **Versión:** siempre vale 4.
- **Longitud Cabecera (IHL, *Internet Header Length*):** en palabras de `32 bits` (rango 5-15).
- **Tipo de Servicio:** para *QoS*.
- **Longitud total:** expresada en octetos, incluye la cabecera (rango 20-65535)

- **Campos de Fragmentación:** Identificación (a qué datagrama pertenece), DF (*Don't Fragment*), MF (*More Fragment*), Desplazamiento del Fragmento (en que parte del datagrama va).
- **Tiempo de vida (TTL):** cuenta saltos hacia atrás, se descarta cuando es cero (rango 0-255).
- **Protocolo:** indica a que protocolo pertenecen los datos.
- **Checksum:** sirve para comprobar la integridad de la cabecera (pero no de los datos).
- **Direcciones de origen y destino:** De `32 bits`, inalteradas durante el TTL del paquete.
- **Opciones:** si las hay su longitud debe ser múltiplo de 4 octetos.

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

Figure 15: Algunas de las opciones del datagrama IP.

1.5.2 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. En realidad un dirección IP se refiere a una **interfaz de red**, no a un *host* en particular.

Las direcciones IP tienen una longitud de 4 `bytes` (32 `bits`) y suelen representarse como cuatro números decimales separados por puntos (por ejemplo, 192.168.0.100). El rango máximo va de 0.0.0.0 a 255.255.255.255, aunque algunas direcciones están reservadas:

- La dirección IP 0.0.0.0 es usada por los *hosts* solo cuando están siendo arrancados.
- Las direcciones IP con 0 como número de red se refieren a la red actual (pueden referirse a su propia red sin saber su número).
- La dirección que consiste solamente en 1s (255) permite la difusión en la red local.
- Las direcciones con un número de red propio y 255 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*).

Todos los *hosts* de Internet tienen entonces direcciones entre 1.0.0.1 a 223.255.255.254. La asignación de direcciones IP puede hacerse:

- Por configuración local en el propio equipo.
- Mediante un protocolo de asignación de direcciones desde un servidor.
- Utilizando direcciones locales del enlace.

Cada interfaz ha de tener asignada una **máscara**, que indique la longitud del prefijo de red. También se le asigna a cada *host* un **enrutador por defecto** (*default gateway*), que esté en la misma LAN.

Subredes:

Nace de permitir la división de una red en varias partes para uso interno, pero aún actuar como una sola red ante el mundo exterior, para solventar el problema del crecimiento de cantidad de redes. Para hacerlo, algunos `bits` se eliminan del número de *host* para crear un número de **subred**.

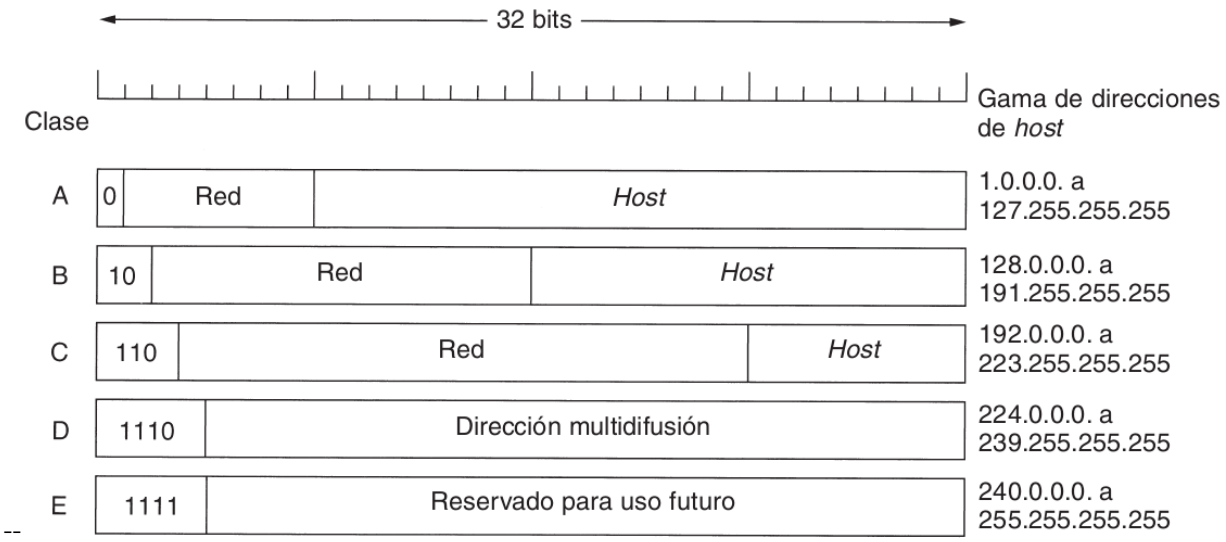


Figure 16: Formatos de dirección IP en **direccionamiento con clase**. La clase establece donde se sitúa la separación red/*host*.

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*. Se suele usar una notación "/X" para indicar que la máscara de subred tiene una longitud de X `bits`.

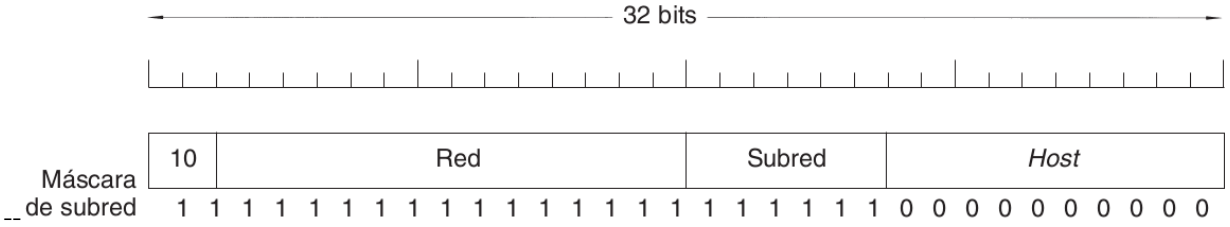


Figure 17: Una red clase B dividida en 64 subredes.

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.

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.

La **división de redes** reduce entonces espacio en la tabla de enrutamiento creando una jerarquía de tres niveles, que consiste en red, subred y *host*.

CIDR (Classless Inter-Domain Routing):

El concepto básico es asignar las direcciones IP en bloques de tamaño variable, **independientemente de las clases**. Este nuevo esquema se aplica al todo el rango libre de direcciones de las antiguas clases A, B y C (pero no a D, *multicast*, ni E, *reservado*), y hace más complicado el reenvío.

En el nuevo algoritmo implementado por CIDR, 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.

1.5.3 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 CdR:

- **ICMP (Internet Control Message Protocol)**: mensajes de error y situaciones anómalas.
- **ARP**: Resolución de direcciones MAC.
- **RARP, BOOTP, DHCP**: Resolución de direcciones IP.
- **IGMP**: Gestión de grupos *multicast*.

ICMP - Protocolo de Mensajes de Control de Internet:

Permite reportar diversas incidencias o situaciones excepcionales que pueden producirse en el envío de un datagrama. Todos los mensajes ICMP se envían en datagramas IP (1 en el campo *protocolo*). Generalmente los mensajes ICMP incluyen como datos la **cabecera** y los primeros bytes de datos del paquete que ha provocado el mensaje ICMP.

Mensaje	Significado
Destination Unreachable (Destino inaccesible)	Red, *host*, protocolo o puerto inaccesible o desconocido. Datagrama demasiado grande que tiene prohibida la fragmentación (`bit DF` puesto).
Source Quench (Apagar la fuente)	Ejerce **control de flujo** sobre el emisor en casos de congestión. No se utiliza. Actualmente se realiza el control de congestión de forma implícita, sin envío de mensajes explícitos.
Echo Request, y *Echo Reply*	Sirve para comprobar la accesibilidad de la IP remota (comando usado: `ping`).
Time Exceeded (Tiempo excedido)	Datagrama descartado por agotamiento del `TTL` (comando usado: `traceroute`).
Redirect (Cambio de ruta)	El enrutador nos sugiere un camino mejor que el que estamos utilizando (más corto).

ARP - Protocolo de Resolución de Direcciones:

¿Cómo se convierten direcciones IP en direcciones de CED? El **Protocolo de Resolución de Direcciones (ARP)** lo que hace es preguntar al resto de su LAN quién posee la dirección IP destino, y la máquina destino le responde con su dirección de CED. Cuando la dirección destino se encuentra fuera de la LAN, el ARP de la máquina origen lo buscará usando el enrutador.

La ventaja de usar ARP en lugar de archivos de configuración -que relacionan direcciones IP con direcciones de CED- es la **sencillez**.

RARP, BOOTP y DHCP:

RARP (*Reverse ARP*). A partir de una dirección Ethernet, devuelve la dirección IP correspondiente.

BOOTP (*Bootstrap Protocol*). Reemplazo de RARP. Usa mensajes UDP que envía a través de los enrutadores. Además proporciona otra información adicional además de la dirección IP.

DHCP (*Dynamic Host Configuration Protocol*). BOOTP extendido. Permite asignación de IP manual y automática (BOOTP sólo permite manual).

2 La Capa de Transporte

Es el corazón de toda la jerarquía de protocolos. Su tarea es la de proporcionar un transporte de datos **confiable** y **económico** de la máquina de origen a la máquina de destino, independientemente de la red o redes físicas en uso.

2.1 El Servicio de Transporte

Servicios proporcionados a las capas superiores:

La meta fundamental es proporcionar un servicio **eficiente**, **confiable** y **económico** a sus usuarios (normalmente procesos de la CdA). Para ello, utiliza los servicios proporcionados por la CdR. El HW o SW de la CdT que se encarga del trabajo se llama **entidad de transporte**.

Ofrece dos tipos de servicios, los cuales son muy parecidos a sus homónimos en la CdR: **orientado** y **no orientado a la conexión.**

La diferencia entre los servicios proporcionados por la CdR y por la CdT es en **dónde se ejecuta el código de transporte**; en el primero es en los **enrutadores**, mientras que en el segundo es en los ***hosts***.

Esencialmente, la existencia de la CdT hace posible que el **servicio de transporte** sea **más confiable** que el **servicio de red** subyacente, ya que puede detectar y compensar paquetes perdidos y datos alterados.

La CdT cumple la función clave de **aislar** a las capas superiores de la tecnología, el diseño y las imperfecciones de la subred. De esta manera, se hace la distinción: las cuatro capas del 1 al 4 pueden verse como el **proveedor del servicio de transporte**, y el resto de las capas superiores son el **usuario del servicio de transporte**.

2.1.1 Primitivas del servicio de transporte

La **esencia** del **servicio de transporte orientado a la conexión** es ocultar las imperfecciones del servicio de red para que los procesos usuarios puedan dar por hecho simple mente la existencia de un flujo de `bits` libre de errores.

Una diferencia entre los servicios de red y de transporte es a quién están dirigidos: el primero lo usan únicamente las entidades de transporte; al segundo lo usan muchos programas (por lo que debe ser adecuado y fácil de usar).

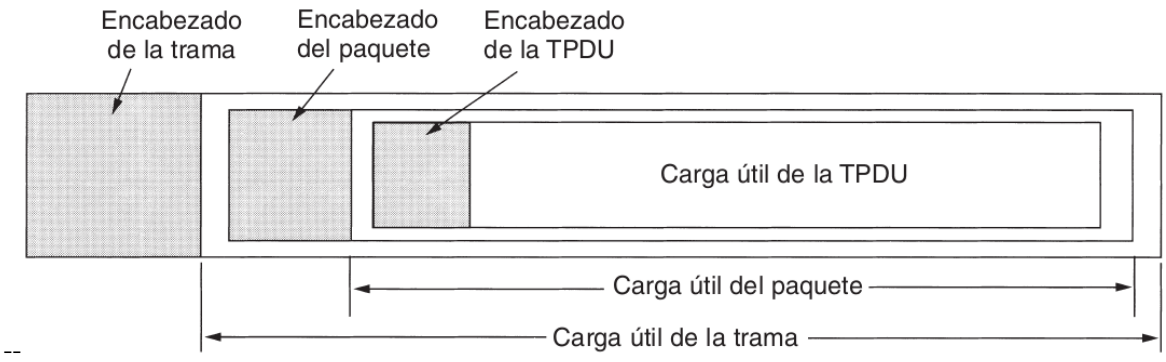


Figure 18: Anidamiento de las TDPUs, los paquetes y las tramas.

Primitiva	Paquete enviado	Significado
LISTEN	(ninguno)	Se bloquea hasta que algún proceso intenta la conexión
CONNECT	CONNECTION REQ.	Intenta activamente establecer una conexión
SEND	DATA	Envía información
RECEIVE	(ninguno)	Se bloquea hasta que llega un paquete DATA
DISCONNECT	DISCONNECTION REQ.	Este lado quiere liberar la conexión

Figure 19: Primitivas de un servicio de transporte sencillo.

TDPU (Unidad de Datos del Protocolo de Transporte) hace referencia a los mensajes enviados de una entidad de transporte a otra.

Proceso en capa de transporte:

1. El servicio ejecuta una primitiva `LISTEN`.
2. Un cliente quiere conectarse a un servidor y ejecuta una primitiva `CONNECT`.
3. El `CONNECT` invoca a una TDPU `CONNECTION REQUEST` al servidor.
4. El servidor envía una TDPU `CONNECTION ACCEPTED`.
5. Ahora los datos se envían usando las primitivas `SEND` y `RECEIVE`.
6. Cuando ya no se necesitan una conexión se emite un TDPU `DISCONNECT`.

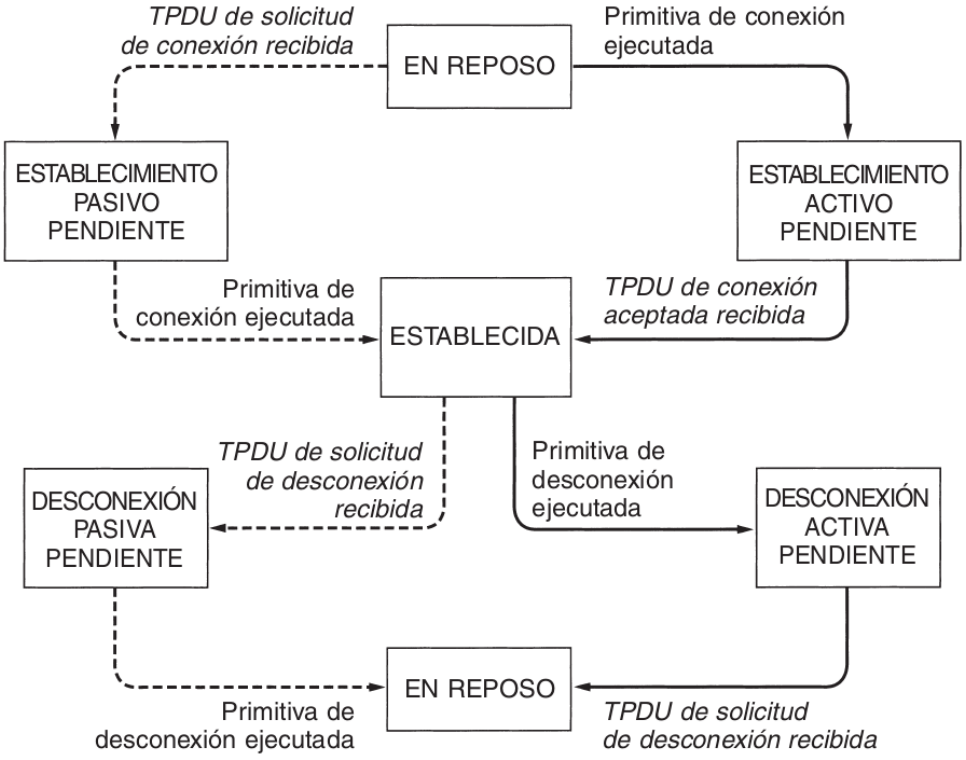


Figure 20: Diagrama de estado de un esquema sencillo de manejo de conexiones. Las transiciones escritas en cursivas son causadas por llegadas de paquetes. Las líneas continuas muestran la secuencia de estados del cliente. Las líneas punteadas muestran la secuencia de estados del servidor.

2.2 Elementos de los Protocolos de Transporte

El servicio de transporte se implementa mediante un **protocolo de transporte** entre las dos entidades de transporte. Este se encarga del control de errores, la secuenciación y el control de flujo, entre otros

aspectos.

Diferencias entre los servicios protocolos de cada capa:

Capa de Enlace	Capa de Transporte
No es necesario direccionamiento.	Se requiere saber la dirección destino.
Establecimiento de la conexión sencillo.	Establecimiento de la conexión complejo.
No existe capacidad de almacenamiento entre los enrutadores.	Existe la posibilidad que los paquetes queden almacenados en la subred durante un tiempo finito.
Se necesitan búferes.	Se necesitan búferes pero de mayor capacidad.

2.2.1 Direccionamiento

El método que normalmente se emplea es definir direcciones de transporte en las que los procesos pueden estar a la escucha de solicitudes de conexión. En Internet, estos puntos terminales se denominan **puertos**; acá usamos el término genérico **TSAP** (**Punto de Acceso al Servicio de Transporte**). Los puntos terminales análogos de la CdR se llaman **NSAP** (**Punto de Acceso al Servicio de Red**) (*las direcciones IP son ejemplos de NSAPs*).

Como en algunas redes cada computadora tiene un solo NSAP, los TSAPs sirven para distinguir los múltiples puntos terminales de transporte que comparten un NSAP.

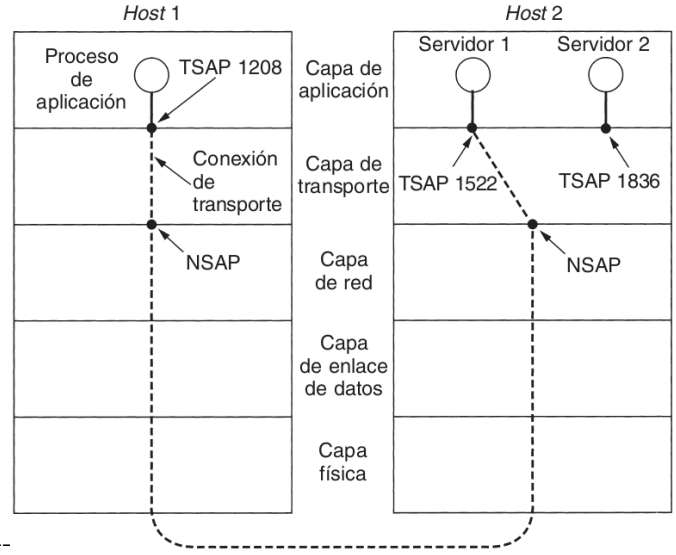


Figure 21: TSAPs, NSAPs y conexiones de transporte.

Protocolo inicial de conexión:

En lugar de que cada servidor concebible escuche en un TSAP bien conocido, cada máquina que desea ofrecer servicio a usuarios remotos tiene un **servidor de procesos** especial que actúa como *proxy* de los servidores de menor uso. Este servidor escucha en un grupo de puertos al mismo tiempo, esperando una solicitud de conexión.

En un esquema alterno existe un proceso especial llamado **servidor de nombres**. Para encontrar la dirección TSAP correspondiente a un nombre de servicio dado, el usuario establece una conexión con el servidor de nombres (que escucha en un TSAP bien conocido). Entonces el usuario envía un mensaje especificando el nombre del servicio, y el servidor de nombres devuelve la dirección TSAP. Luego el usuario libera la conexión con el servidor de nombres y establece una nueva con el servicio deseado.

2.2.2 Establecimiento de una conexión

El establecimiento de una conexión se complica dada la **existencia de duplicados retrasados**. Si podemos asegurar que ningún paquete viva más allá de cierto tiempo conocido, el problema se vuelve algo más manejable.

El tiempo de vida de un paquete puede restringirse a un máximo conocido usando:

1. Un diseño de subred restringido.
2. Colocar un contador de saltos en cada paquete.

3. Marcar el tiempo en cada paquete.

Acuerdo de tres vías:

Este protocolo de establecimiento no requiere que ambos lados comiencen a transmitir con el mismo número de secuencia, por lo que puede usarse con otros métodos de sincronización distintos del método de reloj global.

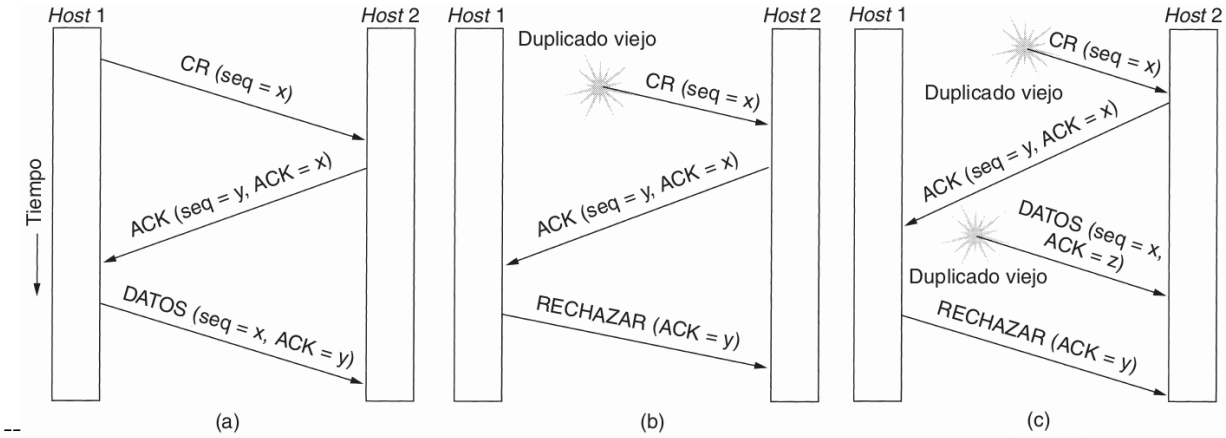


Figure 22: Tres escenarios para establecer una conexión usando un acuerdo de tres vías. CR significa `CONNECTION REQUEST`. (a) Operación normal. (b) `CONNECTION REQUEST` duplicada vieja que aparece de la nada. (c) `CONNECTION REQUEST` duplicada y `ACK` duplicada.

El procedimiento normal de establecimiento al iniciar el *host 1* (ver **Figura 22**). El *host 1* escoge un número de secuencia, x, y envía al *host 2* una TDPU `CONNECTION REQUEST` que lo contiene. El *host 2* responde con una TDPU `CONNECTION ACCEPTED` confirmando la recepción de x y anunciando su propio número de secuencia inicial, y. Por último, el *host 1* confirma la recepción de la selección de un número de secuencia inicial del *host 2* en la primera TDPU de datos que envía.

2.2.3 Liberación de una conexión

Existen dos formas de liberar una conexión:

- **Liberación asimétrica.** Cuando una parte cuelga, se interrumpe la conexión. Es abrupta y puede resultar en la pérdida de datos (ver **Figura 23**.
- **Liberación simétrica.** Trata la conexión como dos conexiones unidireccionales distintas, y requiere que cada una se libere por separado. Es ideal cuando cada proceso tiene una cantidad fija de datos por enviar y sabe con certidumbre cuándo los ha enviado. Sin embargo, tiene el **problema de los dos ejércitos**.

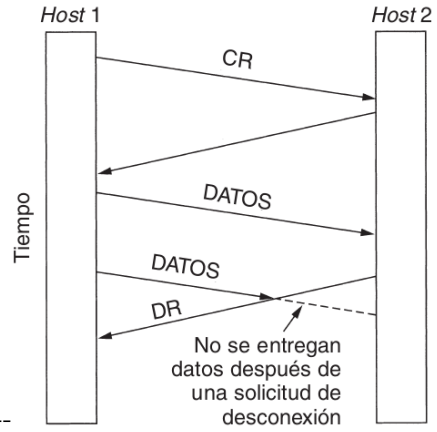


Figure 23: Desconexión abrupta con pérdida de datos (asimétrica).

Problema de los dos ejércitos:

Esencialmente, no importa cuántas rondas de confirmación se lleven a cabo, no hay forma de garantizar que ambos *hosts* coincidan en que la **solicitud de desconexión** ha sido entregada y que no se haya alterado o perdido alguna de las solicitudes entre medio.

Para tratar de solucionarlo en parte se agregan **temporizadores** que expiran luego de cierto tiempo (ver **Figura 24**). Aunque este protocolo puede fallar si se pierden tanto la `DR` y se suceden N retransmisiones: el emisor se dará por vencido y liberará la conexión, mientras el otro lado nunca se enterará de esto y mantendrá la conexión (situación de **conexión semiabierta**).

Hay soluciones que arreglan algunos de estos problemas, pero generan otros: *no hay un protocolo perfecto.*

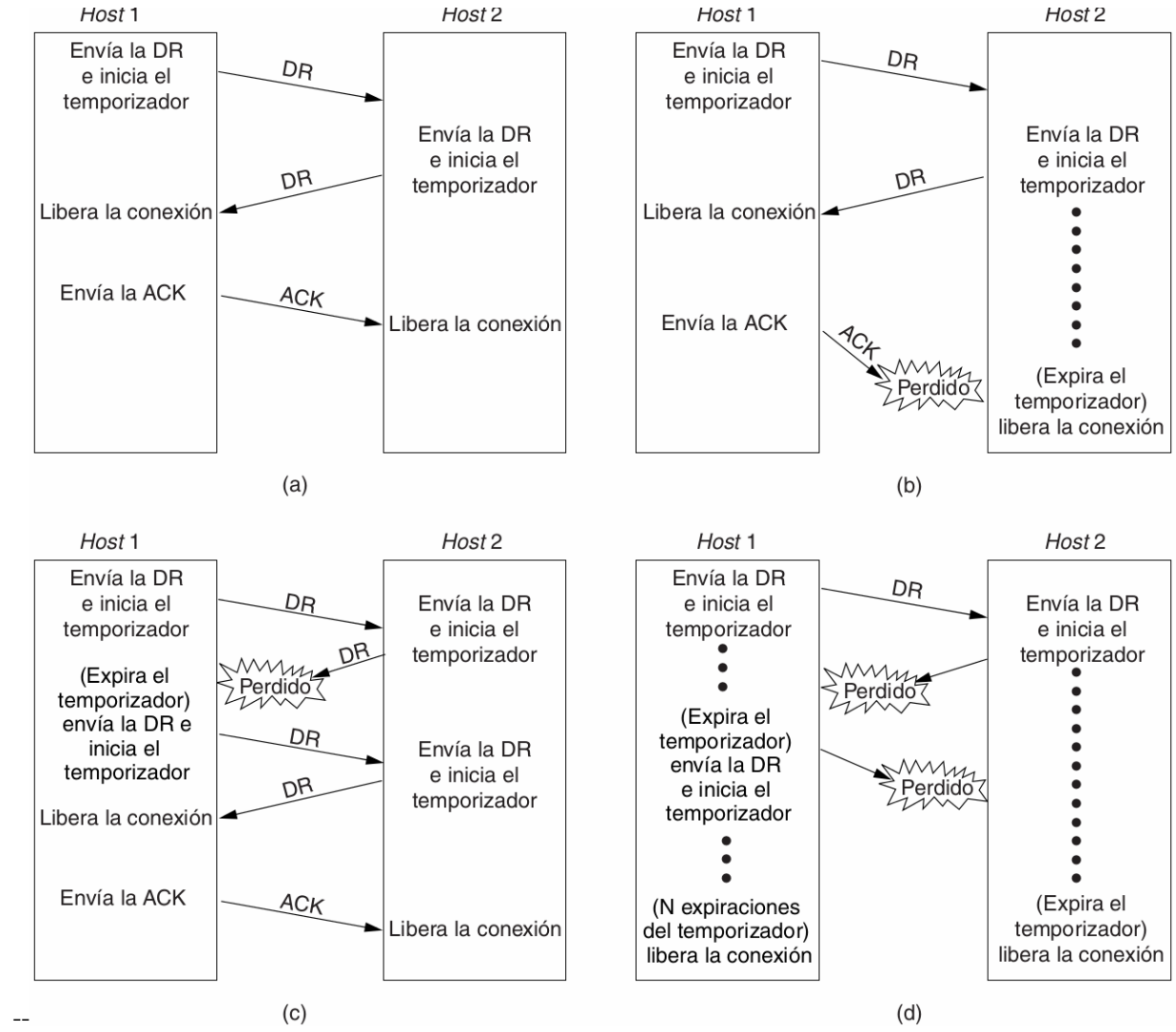


Figure 24: Cuatro escenarios de un protocolo para liberar una conexión. (a) Caso normal del acuerdo de tres vías. (b) Pérdida de la última `ACK`. (c) Respuesta perdida. (d) Respuesta perdida y pérdida de las `DRs` subsecuentes.

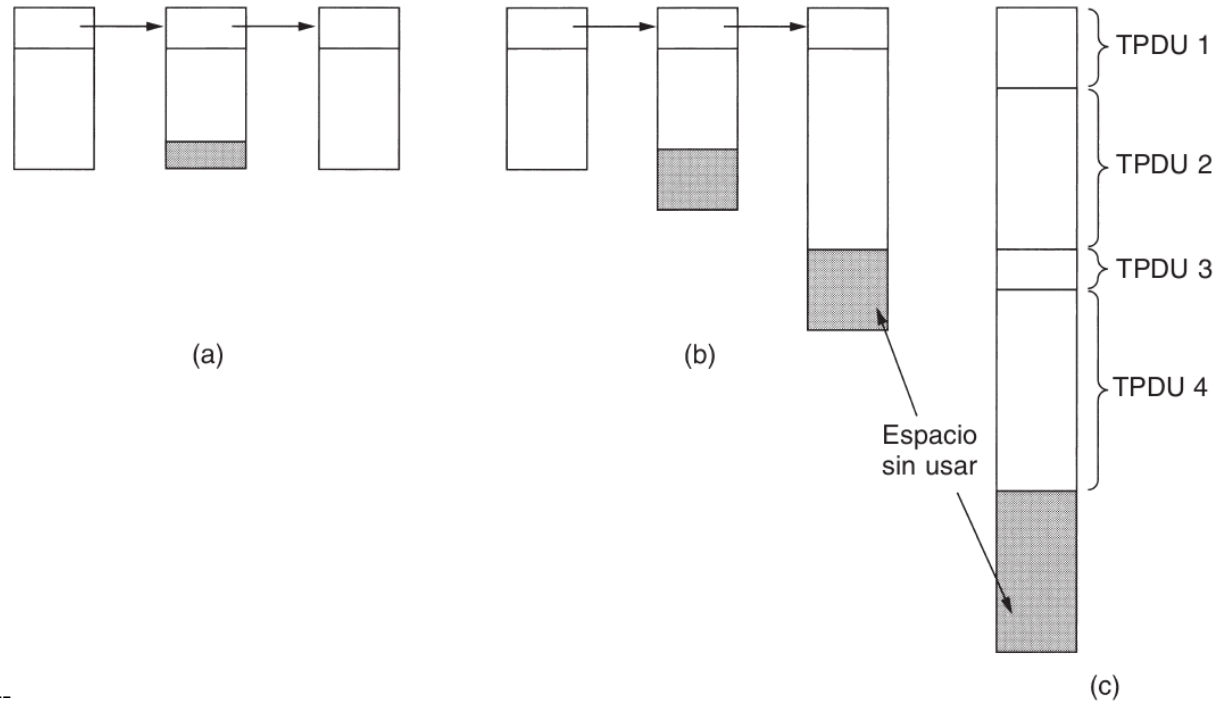
2.2.4 Control de flujo y almacenamiento en búfer

En algunos sentidos el problema del **control de flujo** en la CdT es igual que en la CED, pero en otros es diferente. Ambas capas se requiere una **ventana corrediza** u otro esquema en cada conexión para evitar que un emisor rápido desborde a un receptor lento. La diferencia principal es que un enrutador por lo regular tiene relativamente pocas líneas, y que un *host* puede tener numerosas conexiones.

Si el servicio de red es **no confiable**, el emisor debe almacenar en búfer todas las TDPUs enviadas, igual que en la CED.

Si el servicio de red es **confiable**, se pueden hacer otros arreglos. El receptor no puede garantizar que se aceptará cada TDPU que llegue, por lo que tendrá el emisor que usar búferes. Además el emisor no puede confiar en la ACK de la CdR porque esto sólo significa que ha llegado la TDPU, no que ha sido aceptada. La cuestión está en determinar el tamaño de los búferes en el receptor. En la **Figura 25** pueden verse distintos acercamientos.

El **equilibrio óptimo** entre el almacenamiento en búfer en el origen y en el destino depende del tipo



--

Figure 25: (a) Búferes encadenados de **tamaño fijo**: ideal al tener TDPUs del mismo tamaño, ineficiente en otro caso. (b) Búferes encadenados de **tamaño variable**: mejor uso de memoria a coste de mayor complejidad de administración. (c) Un gran búfer **circular** por conexión: eficiente solo en carga alta.

de tráfico transportado por la conexión: para un tráfico en ráfagas de bajo ancho de banda, es mejor mantener búferes en el emisor; para tráfico continuo de alto ancho de banda, es mejor hacerlo en el receptor.

Manejando la asignación dinámica de buffer:

Además de necesitar una **ventana de tamaño variable**, se usa el siguiente procedimiento de asignación:

1. El emisor solicita una cierta cantidad de búferes.
2. El receptor otorga tantos búferes como puede. Cada vez que el emisor envía una TDPU, disminuye su asignación, deteniéndose por completo al llegar la asignación a cero.
3. El receptor entonces incorpora tanto las ACKs como las asignaciones de búfer al tráfico de regreso.

Para evitar una situación de **bloqueo irreversible** al perderse una TDPU de control, cada *host* debe enviar periódicamente una TDPU de control con la ACK y estado de búferes de cada conexión.

Nota: Si llegase un momento en que el espacio de búfer dejase de limitar el flujo máximo, se necesitará un mecanismo basado por otro lado en la **capacidad de carga** de la subred. Como por ejemplo, de ventana corrediza cuyo tamaño de ventana emisora depende de este parámetro.

2.2.5 Multiplexión

Existen dos tipos de multiplexión en la CdT:

Hacia arriba. Cuando llega un TDPU, se necesita algún mecanismo para saber a cual proceso asignarla.

Hacia abajo. Si un usuario necesita más ancho de banda del que le puede proporcionar un CV, una alternativa es abrir múltiples conexiones de red y distribuir el tráfico entre ellas.

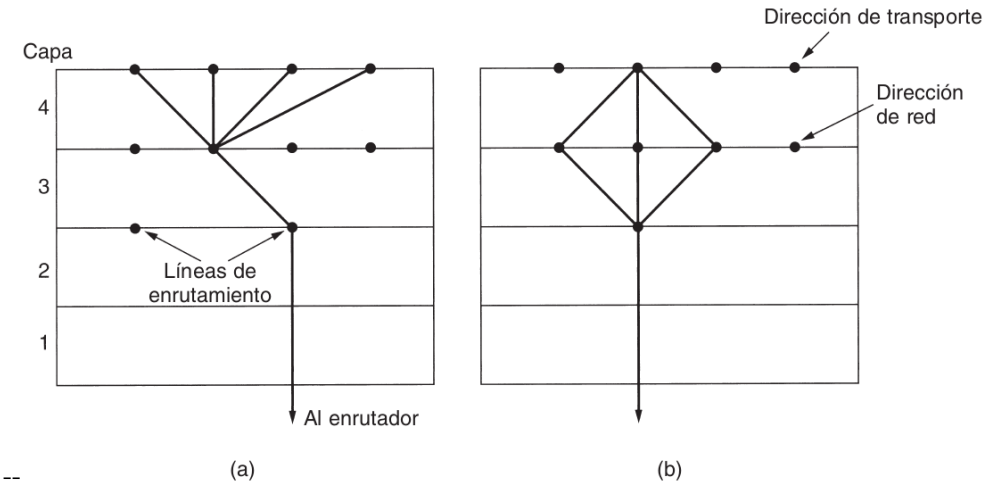


Figure 26: (a) Multiplexión hacia arriba. (b) Multiplexión hacia abajo.

2.3 Los Protocolos de Transporte de Internet: UDP

Internet tiene dos protocolos principales en la CdT:

- **UDP (Protocolo de Datagrama de Usuario)**, no orientado a la conexión.
- **TCP (Protocolo de Control de Transmisión)**, orientado a la conexión.

2.3.1 Introducción a UDP

Proporciona una forma para que las aplicaciones envíen datagramas IP encapsulados sin tener que establecer una conexión. Es un servicio sencillo, pero **no confiable**.

Se utiliza en los siguientes casos:

- El intercambio de mensajes es muy escaso. *Ejemplo: consultas al DNS.*
- La aplicación es en tiempo real. *Ejemplo: Videoconferencias, VoIP.*
- Los mensajes se envían regularmente y no importa si se pierde alguno. *Ejemplo: SNMP.*
- Se envía trafico *broadcast/multicast.*
- Tráfico de P2P o IPTV.

UDP transmite **segmentos** que consisten en un encabezado de 8 `bytes` seguido por la carga útil. Agregamos los campos de **puertos de origen y destino** al formato IP puro. Con ellos, entrega los segmentos de manera correcta.

El **puerto de origen** se necesita principalmente cuando debe enviarse una respuesta al origen. Al copiar el campo *Puerto de origen* del segmento que llega en el campo *Puerto de destino* del segmento que sale, el proceso que envía la respuesta puede especificar cuál proceso de la máquina emisora va a obtenerlo.

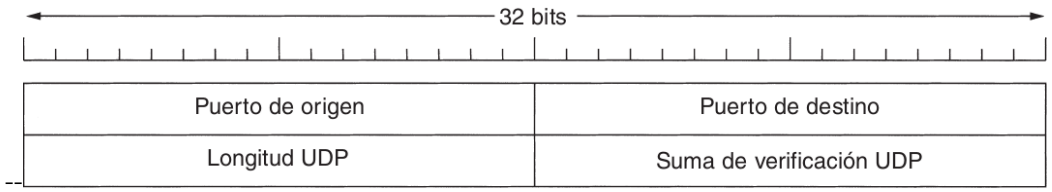


Figure 27: Encabezado UDP.

El campo *Longitud UDP* incluye el encabezado de 8 bytes y los datos. El campo *Suma de verificación UDP* es opcional y se almacena como 0 si no se calcula (si la calidad del servicio no importa).

UDP proporciona una interfaz al protocolo IP con la característica agregada de desmultiplexar varios procesos utilizando los puertos, y efectúa opcionalmente una comprobación de errores. **No realiza**:

- Conexión/Desconexión.
- Retransmisión de datos perdidos.
- Control de Flujo/Congestión.

2.4 Los Protocolos de Transporte de Internet: TCP

La mayoría de las aplicaciones de Internet se necesita una entrega en secuencia **confiable**. UDP no puede proporcionar esto, en su lugar se utiliza TCP.

2.4.1 Introducción a TCP

Se diseño específicamente para proporcionar un flujo de `bytes` **confiable** de extremo a extremo a través de una interred **no confiable**. TCP tiene un diseño que se adapta de manera dinámica a las propiedades de la interred[2] y que se sobrepone a muchos tipos de fallas. Los paquetes TCP se llaman **segmentos**.

La capa IP no proporciona ninguna garantía de que los datagramas se entregarán de manera apropiada, por lo que corresponde a TCP proporcionar la confiabilidad que la mayoría de los usuarios desean y que IP no proporciona.

2.4.2 El modelo del servicio TCP

El servicio TCP se obtiene al hacer que tanto el servidor como el cliente creen puntos terminales, llamados ***sockets***. Cada *socket* posee el número de dirección IP del *host* y un número que es local a ese *host*, llamado **puerto**.

Los números de puerto menores que 1024 se llaman **puertos bien conocidos** y se reservan para servicios estándar. Para evitar que haya muchos *daemons* de procesos inactivos la mayor parte del tiempo, se tiene un sólo *daemon* llamado **inetd (Internet's daemon)** que se conecta a sí mismo a múltiples puertos y gestiona las conexiones de los *daemons* de los otros procesos.

Todas las conexiones TCP son de **dúplex total** y de **punto a punto**. Algunas otras funciones que TCP lleva adelante son:

- Conexión/Desconexión.
- Control de Congestión/Flujo.
- Control de errores, retransmitiendo segmentos perdidos o erróneos. Elimina duplicados.
- Gestión del intercambio de datos de forma eficiente en la red.

Servicio	Puerto	TCP	UDP
DayTime	13		X
FTP	21	X	
SSH	22	X	
TelNet	23	X	
SMTP	25	X	
Domain (DNS)	53	X	X
BOOTP	67		X
TFTP	69		X
HTTP	80	X	
POP3	110	X	
NTP	123		X
SNMP	161		X
LDAP	389		X
HTTPS	443	X	
SIP	5060		X

Figure 28: Algunos puertos asignados.

2.4.3 El Protocolo TCP

La entidad TCP emisora y la receptora intercambian datos en forma de segmentos. Un **segmento** consiste en un encabezado TCP fijo de `20 bytes` (más una parte opcional) seguido de cero o más `bytes` de datos. El *software* de TCP decide el tamaño de los segmentos; puede acumular datos de varias

[2] Una **interred** difiere de una sola red debido a que diversas partes podrían tener diferentes topologías, anchos de banda, retardos, tamaños de paquete y otros parámetros.

escrituras para formar un segmento, o dividir los datos de una escritura en varios segmentos. Hay dos límites que restringen el tamaño de segmento:

1. Cada segmento, incluido el encabezado TCP, debe caber en la carga útil de ≈`64Kb` del IP.
2. Cada red tiene una **unidad máxima de transferencia (MTU)** (generalmente de `1500 bytes`) y cada segmento debe caber en la MTU.

El protocolo básico usado por las entidades TCP es el protocolo de **ventana corrediza**: Cuando llega un segmento al destino, la entidad TCP receptora devuelve un segmento que contiene un número de `ACK` igual al siguiente número de secuencia que espera recibir. Si el temporizador del emisor expira antes del `ACK` el emisor envía de nuevo el segmento.

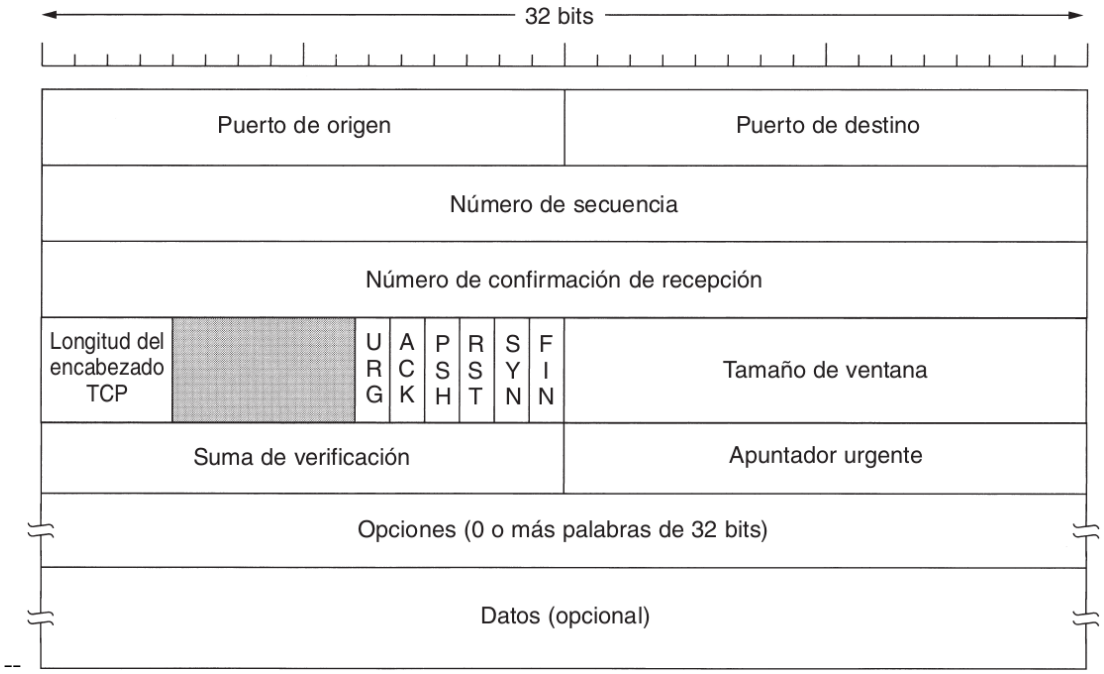


Figure 29: Encabezado TCP.

2.4.4 El encabezado del segmento TCP

Contiene los siguientes campos:

- **Puerto origen y destino**. Identifican los puntos terminales locales de la conexión.
- **Numero de secuencia y de** `ACK`. El segundo especifica el siguiente `byte` esperado.
- **Longitud del encabezado**. Indica la cantidad de palabras de `32 bits` contenidas en el encabezado TCP.
- **URG**. Apuntador urgente. Interrumpe la aplicación receptora y le obliga a que lea los datos urgentes.
- **ACK**. Se establece en 1 si la confirmación es válida.
- **PSH**. Indica datos que se deben transmitir inmediato (no que queden almacenados).
- **RST**. Restablece una conexión.
- **SYN**. Para estabecer una conexión. Se usa para denotar `CR` y `CA`.
- **FIN**. Para liberar una conexión.
- **Tamaño de ventana**. Indica la cantidad de `bytes` que pueden enviarse comenzando por el `byte` cuya recepción se ha confirmado.
- **Suma de verificación**. Agrega confiabilidad. Es una suma de verificación del encabezado, los datos y el *pseudoencabezado conceptual*.
- **Opciones**. Ofrece una forma de agregar características extras no cubiertas por el encabezado normal. *Ejemplo: Carga útil máxima que esta dispuesta a aceptar.*

2.4.5 Establecimiento de una conexión TCP

Se establecen usando el **acuerdo de tres vías**:

- El servidor espera pasivamente una conexión entrante ejecutando las primitivas `LISTEN` y `ACCEPT`, especificando o no el origen.
- El cliente ejecuta una primitiva `CONNECT` especificando la dirección y el puerto IP. Esta primitiva envía un segmento TCP con el `bit SYN` encendido y el `bit ACK` apagado.
- Si el servidor le rechaza, le envía un segmento con el `RST` encendido.
- Si el servidor acepta, le envía un segmento de ACK.

2.4.6 Liberación de una conexión TCP

Para liberar una conexión, cualquiera de las partes puede enviar un segmento TCP con el `bit FIN` establecido, lo que significa que no tiene más datos por transmitir. Al confirmarse la recepción del `FIN`, ese sentido se apaga. Sin embargo, puede continuar un flujo de datos indefinido en el otro sentido. Cuando ambos sentidos se han apagado, se libera la conexión.

Se usan cuatro segmentos (o tres si hay *piggybacking*). Se usan temporizadores para evitar el problema de los dos ejercitos.

2.4.7 Política de transmisión del TCP

Cuando la ventana es de 0, el emisor normalmente no puede enviar segmentos, salvo en dos situaciones:

1. Datos urgentes para enviar.
2. El emisor envia un segmento de `1 byte` para que el receptor reanuncie el siguiente `byte` esperado y el tamaño de la ventana.

Para mejorar el desempeño del TCP, se debe lograr que el emisor no envíe segmentos pequeños, y que el receptor no los pida.

2.4.8 Control de congestión en TCP

El primer paso del manejo de la congestión es su detección. Para manejar la congestión TCP maneja dinámicamente el tamaño de las ventanas, asumiendo que existen dos problemas potenciales: **capacidad de la red** y **capacidad del receptor**.

Para solucionarlos, cada emisor mantiene dos ventanas:

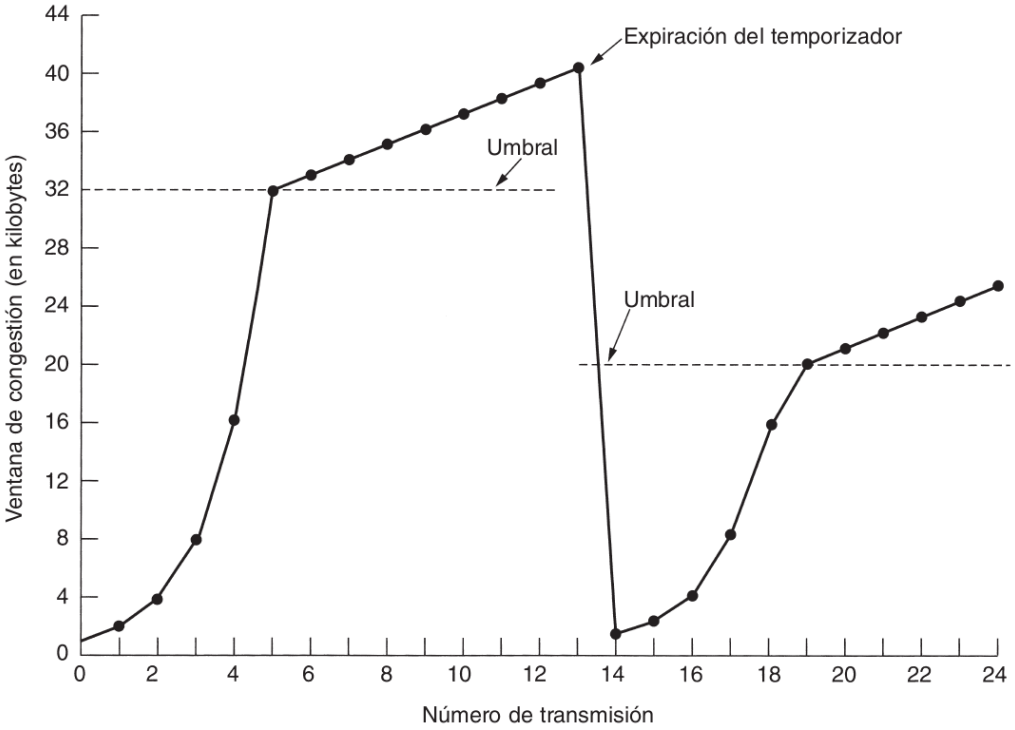
- La ventana que ha otorgado el receptor.
- La **ventana de congestión**.

La cantidad de `bytes` que pueden enviarse es la cifra menor de las dos ventanas.

Algoritmo de arranque lento:

Al establecer una conexión, el emisor asigna a la ventana de congestión el **tamaño de segmento máximo** usado por la conexión; entonces envía un segmento máximo. Si se recibe la ACK de este segmento antes de que expire el temporizador, el emisor agrega el equivalente en `bytes` de un segmento a la ventana de congestión para hacerla de dos segmentos de tamaño máximo, y envía dos segmentos. A medida que se confirma cada uno de estos segmentos, se aumenta el tamaño de la ventana de congestión en un segmento máximo. La ventana de congestión sigue creciendo **exponencialmente** hasta ocurrir una expiración del temporizador o alcanzar el tamaño de la ventana receptora.

Al ocurrir una expiración del temporizador, se establece el **umbral** en la mitad de la ventana de congestión actual, y la ventana de congestión se restablece a un segmento máximo. Luego se usa el *arranque lento* para determinar lo que puede manejar la red, excepto que el crecimiento exponencial termina al



--

Figure 30: Ejemplo de algoritmo de arranque lento.

alcanzar el umbral. A partir de este punto, las transmisiones exitosas aumentan linealmente la ventana de congestión (en un segmento máximo por ráfaga) en lugar de uno por segmento. En efecto, este algoritmo está suponiendo que probablemente es aceptable recortar la ventana de congestión a la mitad, y luego aumentarla gradualmente a partir de ahí.

2.4.9 Administración de temporizadores del TCP

El TCP usa conceptualmente varios temporizadores:

Temporizador de retransmisión. Se inicia al enviarse un segmento. Si la ACK del segmento llega antes de expirar el temporizador, éste se detiene. Si, por otra parte, el temporizador termina antes de llegar la ACK, se retransmite el segmento. Este temporizador es ajustado constantemente con base de mediciones continuas del desempeño de la red.

Temporizador de persistencia. Es para evitar que el emisor y receptor se queden esperando en base a una falla de entrega de paquetes (**bloqueo irreversible**). Al expirar, el receptor envía un sondeo al receptor pidiéndole el estado del tamaño de su ventana.

Temporizador de seguir con vida. Cuando una conexión ha estado inactiva durante demasiado tiempo, el temporizador expira y comprueba el estado del otro lado: si no recibe respuesta, termina la conexión.

Temporizador que usa *Timed Wait*. Se usa durante el cierre. Opera durante el doble del tiempo máximo de vida de paquete para asegurar que, al cerrarse una conexión, todos los paquetes creados por ella hayan desaparecido.