TCP/IP Illustrated - Richard Stevens (Resumen)

CAPÍTULO 3. IP: PROTOCOLO DE INTERNET

Es un protocolo **no confiable** (no asegura que los datagramas lleguen a destino) que proporciona un servicio de **mejor esfuerzo**. Cuando ocurre un error con un datagrama, IP simplemente descarta lo descarta y retorna un mensaje ICMP al origen. Es un protocolo **sin conexión** (sin estado): no mantiene información sobre los datagramas, por lo que pueden ser entregados fuera de orden. Dos datagramas iguales, serán tratados como diferentes.

Cabecera de IP

El tamaño normal de un datagrama IP es 20 bytes.

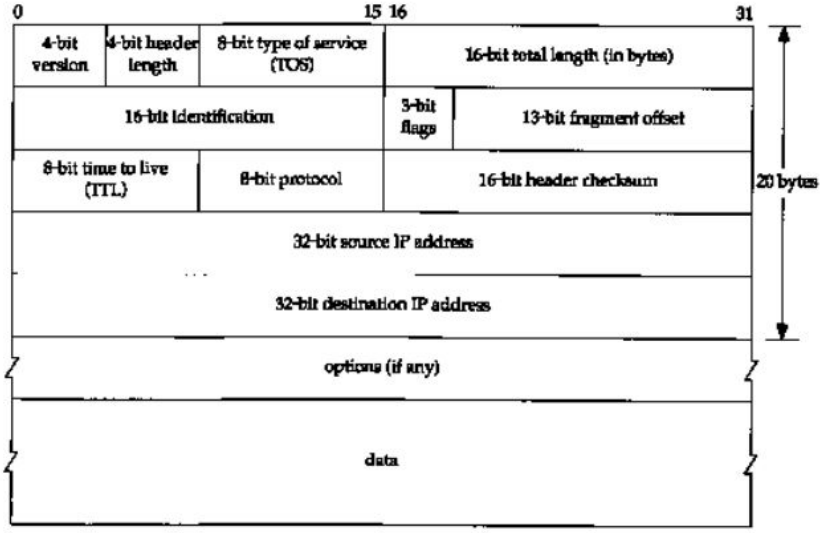


Figure 3.1 IP datagram, showing the fields in the IP header.

El bit más significativo comienza en 0 y termina en 32 (valor es 31 a la derecha). Los 4 bytes (32 bits) se transmiten en el siguiente orden: Bits 0-7 primeros, luego 8-15, 16-23 y finalmente 24-31 (ordenamiento **big endian** o **network byte**).

Campos de Header

1. **Versión**: Versión de IP, IPv4.
2. **Header Length** (Tamaño de cabecera): Número de palabras de 32 bits en la cabecera, incluyendo opciones. Como es de 4 bits (15 decimal), limita la cabecera en (15*32)/8 (60 bytes). El valor normal (cuando no hay opciones) es (5*32)/8 (20 bytes).
3. **Type of Service (TOS)**: Dispone de 3 bits de precedencia, 4 **bits TOS**, y un bit sin uso que debe ser 0. Los bits TOS son: **minimizar delay**, **maximizar throughput**, **maximizar confiabilidad** y **minimizar costo**. Todos los bits en 0, implican un servicio “normal”. Solo uno de esos bits puede ser prendido. La opción de minimizar delay es útil para aplicaciones interactivas (donde se transfieren pequeñas cantidades de datos), por ejemplo Telnet y Rlogin. FTP, por otra parte, buscará máximo throughput. SNMP buscará máxima confiabilidad. Nota: TOS **no está soportado** en la mayoría de implementaciones actuales de TCP/IP.

4. **Total Length** es el tamaño total del datagrama IP en bytes. Con este campo y el de tamaño de cabecera, es posible conocer dónde comienza y termina el datagrama IP. Al ser de 16 bits, el tamaño máximo de un datagrama IP es 65532 bytes. El campo cambia cuando un datagrama está fragmentado. *A pesar de que es posible enviar un datagrama de 65532 bytes, la mayoría de las capas de enlace* ***fragmentarán****. Por otra parte un host no está obligado a recibir un datagrama más grande que 576 bytes. Como TCP divide los datos de usuario en segmentos, no afecta a TCP. Sin embargo, con UDP los datos de usuario se limitan a 576 bytes. De todas formas, la mayoría de implementaciones de hoy día (en especial las que soportan NFS, Network File System) permiten datagramas IP a poco más de 8192 bytes.* Se requiere el campo tamaño total, ya que algunas capas de enlace (ej: Ethernet) realizan **padding** (relleno) a pequeños frames para que cumplan un tamaño mínimo. A pesar de que el tamaño mínimo de trama Ethernet es 46 bytes, el datagrama IP puede ser menor: sin el tamaño total, la capa IP no sabría cuánto de una trama Ethernet es realidad un datagrama IP.
5. **Identificación**: Identifica a cada datagrama de **forma unívoca**. El RFC 791 (Postel) dice que este campo debe ser elegido por la capa superior a IP. Esto significa que dos datagramas consecutivos (uno enviado por TCP y otro por UDP), puede tener el mismo campo de identificación. Sin embargo la mayoría de implementaciones se hacen sobre IP, por cada datagrama enviado.
6. **Time to live (TTL)**: Establece una **cota superior**, limitando la cantidad de routers por los que el datagrama puede pasar. Es inicializado con un valor, generalmente 32 o 64, y decrementado en 1 por cada router que maneja el datagrama. Cuando el **campo alcanza el 0**, el **datagrama se descarta** y el origen se notifica con un mensaje **ICMP**. Previene que los paquetes queden “atrapados” en bucles de enrutamiento.
7. **Protocolo**: Identifica el protocolo de capa superior.
8. **Suma de comprobación de cabecera (Header Checksum)**: Únicamente calculado sobre el header IP. ICMP, IGMP, UDP y TCP tienen checksums en sus respectivas cabeceras para cubrir su (propia) cabecera y datos. Se recalcula cada vez que algún nodo cambia algún campo (ejemplo el TTL). *Para computar el checksum se suma complemento a 1 cada palabra de 16 bits de la cabecera, y haciendo complemento a 1 del valor resultante.* Luego, el receptor también lo calcula y compara. Si hay un error, IP descarta el datagrama si generar un mensaje de error, siendo tarea de las capas superiores ocuparse de la retransmisión.
9. **IP origen y destino**: Datos de emisor y receptor.
10. **Opciones**: Es una lista de tamaño variable. Las opciones son:
    - **Seguridad y manejo de restricciones**: no clasificado hasta máximo secreto.
    - **Registro de ruta**: Registra itinerario de un datagrama. Cuando un nodo recibe el paquete, escribe su dirección IP en la posición indicada por el puntero y lo incrementa en 4 (siempre que sea menor que el tamaño de opción). La opción no se copia en caso de fragmentación.
    - **Marca de tiempo** (Timestamp).
    - **Enrutamiento desde el origen no estricto** (Loose Source Routing): Ruta establecida por el origen, con nodos intermedios alternativos.

- **Enrutamiento desde el origen estricto**: El paquete debe seguir estrictamente el camino especificado.

Estas opciones son raramente utilizadas y no todos los hosts la soportan. El campo de opciones tiene un límite de 32 bits. Se rellena con 0 si es necesario, asegurando que la cabecera sea siempre sea múltiplo de 32 (requerido para el campo de longitud de cabecera).

Ruteo IP

El ruteo IP es simple, especialmente para un host. Si el destino está directamente conectado o en la misma red, el datagrama IP se envía **directamente** a destino. De otro modo, el host envía el datagrama a un **router** por defecto, para que entregue el datagrama a destino. *Aclaración: un **host** no rutea datagramas de una interfaz a otra, mientras que un **router** sí. Un host que contiene embebida una funcionalidad de router, no puede reenviar a datagrama a menos que esto se explicite.*

La capa IP posee una **tabla de ruteo** en memoria que se utiliza para decidir dónde enviar un datagrama. Cuando se recibe un datagrama, IP primero chequea si la dirección es local o no. Si es local, el datagrama se entrega al módulo de protocolo especificado en la cabecera IP (destino). Si el datagrama no está destinado para dicha capa IP (no es local) y si el host está configurado como router, se reenvía el paquete (de otro modo, el datagrama se descarta).

Cada tabla de ruteo contiene:

- **IP destino**: Puede ser un host o una red.
- **IP del siguiente salto a router** o red directamente conectada. No es el destino final, pero toma los datagramas y los envía al destino.
- **Flags**: Especifica si una IP es de red o de host, o si es un router o una interfaz directamente conectada.
- **Interfaz de red**.

El enrutamiento IP se realiza en base a **saltos**. IP no conoce la ruta completa para cualquier destino (excepto que esté directamente conectada). Todo lo que provee el enrutamiento IP es la dirección del salto al siguiente router, a donde se debe enviar el datagrama.

El ruteo IP realiza las siguientes acciones (en orden):

1. Busca una dirección IP (en su tabla) que coincida con la IP destino. Si se encuentra, envía el paquete al siguiente salto o interfaz directamente conectada.
2. Busca en la tabla de ruteo la entrada que más coincida con el network ID y envía el paquete allí.
3. En última instancia, envía el datagrama a la entrada especificada como “default”.

Otras características:

- Si el datagrama no se puede entregar, se genera el error “**host unreachable**” o “**network unreachable**” (ICMP) a la aplicación origen.
- La posibilidad de especificar las redes en las tablas de enrutamiento es una permite “**resumir**” las tablas de rutas.

- Cada host obtiene la **dirección MAC**, del siguiente salto o red directamente conectada, mediante protocolo ARP, y pasa el paquete a la capa inferior.

Puntos a tener en cuenta:

1. Algunos routers usan la ruta default para redes que estén fuera del alcance local.
2. La IP destino en el datagrama jamás cambia.
3. Diferentes capas de enlace (protocolos) pueden ser usados en cada enlace.

Dirección de subred

Se utiliza para no desaprovechar direcciones IP. Por ejemplo, en las de clase A y B, es posible conectar muchísimos hosts, sin embargo, rara vez se utilizará un bloque entero. Por ello, se opta por la división.

Figure 3.5 Subnetting a class B address.

El **subneteo** esconde los detalles internos de una organización a los routers externos.
Las redes clase A y B son las más sub-neteadas (por la cantidad de subredes y host disponibles).

Máscara de subred: Necesaria para diferenciar network ID de host ID. Con la dirección IP y la máscara de subred, es posible determinar si un datagrama IP está destinado a:

1. Un host dentro de la subred.
2. Un host en otra subred.
3. Un host en una red diferente.

Algunos comandos

- **ifconfig**
  - Comando para consultar o configurar una interfaz de red TCP/IP.
  - La interfaz de **loopback** es considerada una interfaz de red.
- **netstat**: Provee información acerca de las interfaces.

CIDR (ruteo sin clases) propone una solución al desuso de direcciones causado por una estructura de ruta no jerárquica y plana.

CAPÍTULO 4. ARP: ADDRESS RESOLUTION PROTOCOL

ARP provee un **mapeo dinámico** entre dos formas de direccionamiento diferentes: IP, de 32 bits y direcciones de protocolo de enlace, para el caso Ethernet (MAC, de 48 bits). ARP convierte direcciones lógicas a físicas (de hardware). Es dinámico, ocurre de forma automática (no es tarea del administrador de red).
RARP, por otra parte, se utiliza para sistemas sin disco rígido, que requieren conocer su dirección lógica (a diferencia de ARP requiere configuración manual).

Para obtener la **dirección física** de una **IP**, el módulo ARP envía un broadcast a toda la red Ethernet (**ARP request**), con la IP destino y el mensaje: **“si tu eres dueño de esta IP,**

respóndeme con tu dirección física". Si algún host es dueño la dirección IP, responde con esta dirección y la de hardware.
La fundamentación detrás de ARP, es que kernel (por ejemplo, el driver Ethernet) debe conocer el **destino físico**, pero **TCP/IP trabaja con direcciones lógicas**. ARP provee este **mapeo dirección lógica-dirección física** (que puede utilizar varias tecnologías).

Nota: Los enlaces punto a punto no necesitan utilizar ARP. Cuando estos links son configurados el kernel sólo debe ser informado de la dirección IP de cada extremo.

Caché ARP: se realiza en cada host. El comando **arp -a** permite verificarla. Los tiempos de vida son de 20 minutos para entradas completas y de 3 minutos para incompletas. Cada tiempo de vida se resetea a 0, cuando se utiliza (los requisitos del RFC, sin embargo, especifican que la entrada debería descartarse igualmente).

Formato de paquete ARP

Paquete solicitud/respuesta ARP

Tipo de hardware	2 bytes
Tipo de protocolo	2 bytes
Longitud dirección de hardware en bytes (x) / Longitud dirección de protocolo en bytes (y)	2 bytes
Código de operación	2 bytes
Dirección hardware del emisor	x bytes
Dirección IP del emisor	y bytes
Dirección hardware del receptor	x bytes
Dirección IP del receptor	y bytes

1. **Frame type**, especifica que tipo de datos se transmiten. Para un ARP request o reply, es 0x0806.
2. **Tipo de hardware** especifica el tipo de dirección física. Valor es 1 para Ethernet.
3. **Prot** especifica el tipo de protocolo lógico. 0x0800 para IP.
4. **Longitud de hardware y de protocolo**, especifican el tamaño en bytes de la dirección física y de protocolo. Para IP sobre Ethernet, se tienen 6 y 4, respectivamente.
5. **Código de operación** indica si se trata de un ARP request (1), ARP reply (2), RARP request (3) o RARP reply (4). El campo es obligatorio,ya que en frame type se usa el mismo identificador para un ARP request y reply.

Existe duplicación: Las direcciones de hardware están disponibles tanto en la cabecera, como en el cuerpo.

¿Qué pasa si ARP solicita un host no existente? Se produce un time-out con el error "unable to connect to remote host". La tabla ARP se rellena con la dirección IP y el valor "incomplete" para la dirección física. Previo a ello, ARP realiza una serie de reintentos.

Proxy ARP (ARP hack o **ARP promiscuo)**
Consiste en que un router envíe su propia dirección ante consultas ARP de un host que se encuentra en una interfaz, si el host destino está en alguna otra. Se puede usar para esconder dos direcciones físicas de hosts entre sí o para aquellos hosts que implementen viejos estándares de TCP/IP o Ethernet.

ARP gratuito: En ocasiones (por ejemplo, cuando se cambia una dirección IP), es útil que un host envíe un **ARP request con su propia dirección IP** a los demás, con dos propósitos: Que los hosts **actualicen su caché**, o para **detectar direcciones IP duplicadas**. En el último caso, si un host responde el ARP request, luego, el emisor del paquete ARP gratuito, puede actuar en consecuencia, y generar una advertencia de IP duplicada (en algunas implementaciones, el host receptor, es quien, directamente, envía esta advertencia al darse cuenta que la dirección IP origen y la suya son las mismas).

Envenenamiento ARP: http://blackspiral.org/docs/arp_spoofing.html
El protocolo ARP tiene carencias que facilitan el uso ilegítimo para recibir tráfico ajeno debido a:

- **Ausencia de autenticación**: Un host no puede determinar de ningún modo la autenticidad de los paquetes ARP.
- **Cachés sujetas a alteraciones externas**: Es posible modificar las entradas de caché ARP, tan sólo construyendo y enviando una petición o respuesta adecuada.
- **Actualización de las cachés por iniciativa externa**: Con ARP gratuito, una máquina puede actualizar las cachés ARP del resto en cualquier momento.

El protocolo ARP es sensible a amenazas del tipo man-in-the-middle.

Comandos ARP

- **-a** para visualizar las entradas en la caché.
- **-d** para eliminar un entrada.
- **-s hostname address** agrega una entrada permanente (a menos que se especifique), siendo hostname la IP y address la dirección Ethernet. **pub** al final de un **-s** indica que el host puede responder un ARP request que consulte por dicha dirección IP.

En efecto, ARP es el protocolo básico en las implementaciones TCP/IP, que (normalmente) hace su trabajo sin ayuda de un administrador de red. La caché es fundamental.

CAPÍTULO 5. RARP: REVERSE ADDRESS RESOLUTION PROTOCOL

Se utiliza en sistemas que **no poseen memoria persistente propia**. Estos sistemas no conocen su dirección lógica (en principio), pero si su dirección de hardware. Aunque el concepto es simple, la implementación puede ser difícil.

Formato de paquete RARP: Es similar al paquete ARP. La única diferencia es que el **frame type** cambia en caso de RARP (0x8035), y la **op-field** utilizada es 3 (para RARP request) y 4 (para RARP reply).

Se necesita un servidor RARP: La función de servidor RARP la provee un proceso de usuario, no es parte del kernel de una implementación de TCP/IP. Un problema es que las peticiones RARP se transmiten como tramas Ethernet, que no son forwardeadas por los routers. Además, se responde con **mínima información**: únicamente la dirección del sistema (y no otros datos interesantes como la máscara, puerta de enlace, etc). Otro problema son los servidores redundantes, que incrementan el tráfico en la red (por las respuestas). *Una alternativa es **BOOTP**, que funciona con UDP*.

CAPÍTULO 6: ICMP - INTERNET CONTROL MESSAGE PROTOCOL

ICMP (protocolo de mensajes de control de Internet) se considera parte de la capa IP (los mensajes se encapsulan en datagramas). ICMP comunica **mensajes de error** y **cuestiones que requieren atención**.

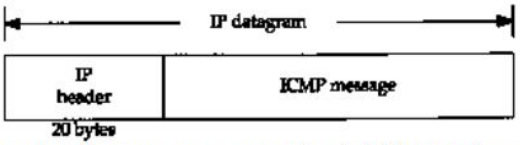


Figure 6.1 ICMP messages encapsulated within an IP datagram.

Mensaje ICMP

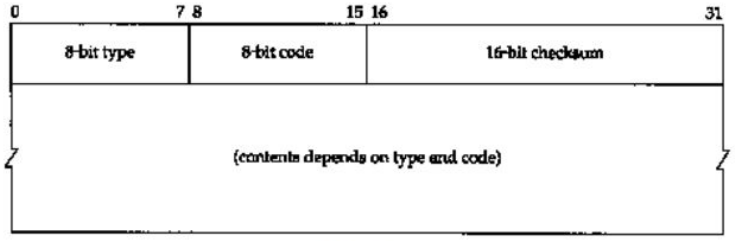


Figure 6.2 ICMP message.

Los primeros 4 bytes (fijos), tienen el mismo formato para todos los mensajes. El contenido, depende del mensaje en sí. Existen 15 tipos de valores para el campo **tipo** (c/u identifica un mensaje ICMP). El campo **checksum** cubre el mensaje ICMP entero (se utiliza el mismo algoritmo que para IP).

Tipos de mensaje ICMP: Los tipos de mensaje se determinan con el campo **tipo** y el **código de mensaje ICMP**. Puede tratarse de un query, o un error. Un mensaje ICMP de error no se puede generar a causa de otro de error o de un datagrama de difusión o multidifusión: no puede ser una dirección con 0 (ceros), una dirección loopback, una de broadcast o multicast (esto último previene las tormentas de broadcast).

type	code	Description	Query	Error
0	0	echo reply (Ping reply. Chapter 7)	*	
3		destination unreachable:		
	0	network unreachable (Section 9.3)		*
	1	host unreachable (Section 9.3)		*
	2	protocol unreachable		*
	3	port unreachable (Section 6.5)		*
	4	fragmentation needed but don't-fragment bit set (Section 11.6)		*
	5	source route failed (Section 8.5)		*
	6	destination network unknown		*
	7	destination host unknown		*
	8	source host isolated (obsolete)		*
	9	destination network administratively prohibited		*
	10	destination host administratively prohibited		*
	11	network unreachable for TOS (Section 9.3)		*
	12	host unreachable for TOS (Section 9.3)		*
	13	communication administratively prohibited by filtering		*
	14	host precedence violation		*
	15	precedence cutoff in effect		
4	0	source quench (elementary flow control. Section 11.11)		*
5		redirect (Section 9.5):		
	0	redirect for network		*
	1	redirect for host		*
	2	redirect for type-of-service and network		*
	3	redirect for type-of-service and host		*
8	0	echo request (Ping request. Chapter 7)	*	
9	0	router advertisement (Section 9.6)	*	
10	0	router solicitation (Section 9.6)	*	
11		time exceeded:		
	0	time-to-live equals 0 during transit (Traceroute, Chapter 8)		*
	1	time-to-live equals 0 during reassembly (Section 11.5)		*
12		parameter problem:		
	0	IP header bad (catchall error)		*
	1	required option missing		*
13	0	timestamp request (Section 6.4)	*	
14	0	timestamp reply (Section 6.4)	*	
15	0	information request (obsolete)	*	
16	0	information reply (obsolete)	*	
17	0	address mask request (Section 6.3)	*	
18	0	address mask reply (Section 6.3)	*	

Figure 6.3 ICMP message types.

Mensajes interesantes de ICMP

- **Request and reply de máscara**: Diseñado para sistemas sin disco para obtener su máscara de red (similar a RARP). Una alternativa es BOOTP.
- **Request and reply de Timestamp**: Request de fecha y hora actuales. El valor recomendado debe ser en milisegundos desde la medianoche, en UTC (una alternativa “más saludable” es la utilización de daytime o NTP).
- **Puerto inalcanzable**: Cuando el destino no corresponde a un proceso en escucha, TCP o UDP almacenan los puertos origen y destino dentro de los primeros 8 bytes del mensaje ICMP, para generar el error a la aplicación.

CAPÍTULO 7: PING

Tiene por objetivo verificar si un host es **alcanzable**. Para ello, el programa envía un ICMP “echo request” al host, esperando una respuesta. **Es el test de conectividad básico entre dos sistemas que corren bajo TCP/IP.**

Nota: Como, por seguridad, en algunos casos, ciertos tipos de mensaje se filtran, ping no es una herramienta 100% confiable.

Encabezado*:*

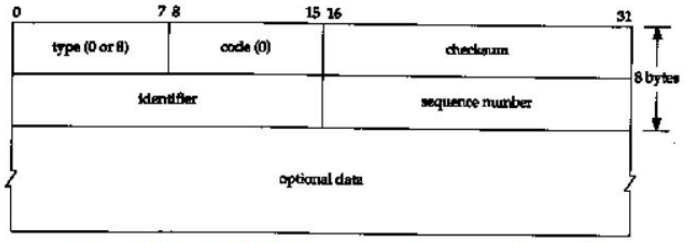


Figure 7.1 Format of ICMP message for echo request and echo reply.

La mayoría de implementaciones setea el identificador en el mismo que el **process ID** que inicia la acción, permitiendo **múltiples instancias** de ping. Por otra parte, el **número de secuencia** comienza en 0 y se incrementa por cada nuevo echo request enviado. Históricamente la petición de ping operó, por defecto, enviando una solicitud, cada segundo. Hoy día, la mayoría de implementaciones (Linux), envía una sola petición. Se debe especificar el comportamiento anterior con **-s**.

Ejemplo de output:

```
bsdi % ping svr4
PING svr4 (140.252.13.34): 56 data bytes
64 bytes from 140.252.13.34: icmp_seq=0 ttl=255 time=0 ms
64 bytes from 140.252.13.34: icmp_seq=1 ttl=255 time=0 ms
64 bytes from 140.252.13.34: icmp_seq=2 ttl=255 time=0 ms
64 bytes from 140.252.13.34: icmp_seq=3 ttl=255 time=0 ms
64 bytes from 140.252.13.34: icmp_seq=4 ttl=255 time=0 ms
64 bytes from 140.252.13.34: icmp_seq=5 ttl=255 time=0 ms
64 bytes from 140.252.13.34: icmp_seq=6 ttl=255 time=0 ms
64 bytes from 140.252.13.34: icmp_seq=7 ttl=255 time=0 ms
^? type interrupt key to stop
--- svr4 ping statistics ---
8 packets transmitted, 8 packets received, 0% packet loss
round-trip min/avg/max = 0/0/0 ms
```

La herramienta ping también puede ser útil para **calcular el RTT**. Los mensajes ICMP se visualizan cuando llega una respuesta (que pueden ser o no en el orden enviado). *Para el ejemplo dado (LAN), todos los mensajes han llegado en el orden que los request fueron enviados. Para el caso de una WAN, puede existir pérdida de paquetes y un RTT mayor, y existe mayor probabilidad que los mensajes no se visualicen en el orden enviado.*

Opción record route (RR)

Esta opción permite que cada router por donde pasa el datagrama, imprima su ruta dentro de una lista dentro del datagrama. Sin embargo, la mayoría de sistemas no implementan esta funcionalidad. Otro problema es la limitación impuesta por header length header de IP (4 bits), lo que limita el encabezado IP a 60 bytes. Dado que el tamaño fijo de un encabezado IP es 30 bytes y RR usa 3 bytes para el overhead, el resultado es 60-20-3=37 bytes, disponibles para la lista. Si este resultado se divide por 4 (tamaño de una dirección IP), se tienen 9 direcciones posibles para la opción de Record route.

Encabezado:

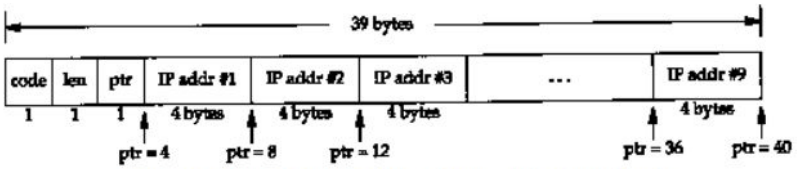


Figure 7.3 General format of record route option in IP header.

- **Code**, especifica la opción de IP. Para RR, el valor es 7.
- **Len** es el total de números de bytes de RR, que en este caso es 39
- **Ptr** es un puntero que indica donde almacenar la siguiente IP. Su valor mínimo es 4 (puntero de la primer IP). Por cada IP grabada, el valor aumenta a 8, 12, 16, hasta 36. Luego de 9 direcciones, ptr se vuelve 40, lo que indica que la lista está llena.

Un router (por definición **multihomed**), ¿**qué dirección IP almacena en la lista**? El RFC 791, específica que debe grabarse la interfaz de salida. En un ICMP echo reply, se graba la dirección de entrada. Si se realiza un ping con RR activo (con la opción -R), se mostrarán las variaciones de rutas entre ambos extremos.

Opción Timestamp

Es similar a la RR. **Encabezado**:

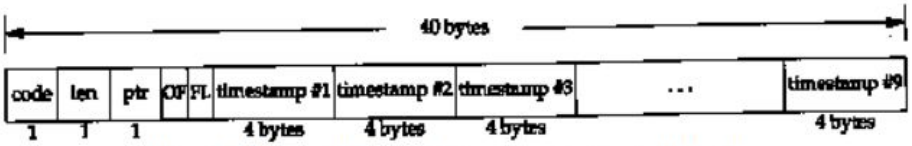


Figure 7.7 General format of timestamp option in IP header.

- El código **0x44** se utiliza para la opción timestamp.
- **OF** es el campo de overflow.
- **FL** refiere a las banderas:
  - 0: Cada router sólo debe grabar su timestamp.
  - 1: Cada router debe grabar la IP y timestamp (sólo hay lugar para 4 pares).
  - 3: El emisor inicializa la lista de opciones con cuatro pares de IP y 0 en los valores de timestamp. Un router grabará su timestamp sólo si la siguiente IP en la lista coincide.

*Si un router no puede agregar su timestamp (porque no hay espacio) se incrementa el campo **overflow**. El valor para el timestamp, es el número en milisegundos pasada la medianoche, en UTC (similar a ICMP timestamp request & reply). En comparación a la opción RR, los problemas aumentan con la opción timestamp. Si se graban las IP y timestamp, solo se pueden almacenar 4 de estos pares (y grabar sólo los timestamps no sería de utilidad, ya que no se sabría que router lo originó). Quizás lo interesante sería utilizar la opción 3 (que permite que el emisor pre-grabe las IPs de routers, y que estos ingresen sus timestamp), sin embargo sigue siendo bastante limitado en su uso.*

CAPÍTULO 8: TRACEROUTE

Aunque no hay garantía que dos datagramas consecutivos sigan la misma ruta, la mayoría del tiempo lo hacen. La opción **RR** (anterior apartado) **se utiliza muy poco** ya que no todos los routers la soportan. Otra limitante es el poco espacio disponible para grabar las rutas.

Traceroute utiliza ICMP y el campo **TTL** de un encabezado IP. El TTL es un campo de 8 bits que se inicializa en algún valor (recomendado 64, con un máximo de 255). Cada router que maneja el datagrama debe **decrementar el TTL** en uno (el RFC 1009 especifica que si el datagrama se retiene en un router por más de un segundo, debe decrementar el TTL por el total de segundos, sin embargo, en la gran mayoría de implementaciones, el TTL actúa como contador de saltos). El **propósito** principal del **TTL** es **prevenir que un datagrama termine en un loop infinito**, estableciendo un límite de saltos. **Cuando un router recibe un datagrama cuyo TTL es 0 o 1, no se forwardea**.

Nota: un host destino que recibe un datagrama con valor 0 o 1, debe encaminarlo a la aplicación, ya que este no rutea. El ruteador, en cambio, debe descartarlo y generar un mensaje ICMP de "tiempo excedido" al emisor.

La clave para traceroute, es que el mensaje ICMP de "tiempo excedido" contiene la dirección IP del router que lo generó. De modo que **traceroute trabaja de la siguiente manera**:

1. Envía un datagrama con **TTL 1** al destino.
2. El primer router **decrementa el TTL**, descarta el datagrama y envía un mensaje ICMP de **TTL excedido**. Esto permite identificar el primer router.
3. Luego, el emisor, envía un datagrama con **TTL 2**, continuando así **hasta que el datagrama alcance el destino**.
4. Sin embargo, como **el destino no genera un mensaje ICMP**, es necesario establecer otra estrategia: En esta instancia, Traceroute genera **datagramas UDP**, con un **puerto poco utilizado** (mayor a 30000), lo que hace improbable que una aplicación use ese puerto. Esto causa el mensaje ICMP "puerto inalcanzable". Todo lo que traceroute debe hacer es diferenciar los mensajes ICMP recibidos.

Nota: el programa traceroute debe ser capaz de setear el campo TTL para el datagrama saliente. No todas las interfaces TCP/IP soportan esto, pero la mayoría de sistemas actuales sí (aunque puede requerir ciertos privilegios de super usuario).

Ejemplo de traceroute en LAN:

```
svr4 % traceroute slip
traceroute to slip (140.252.13.65), 30 hops max. 40 byte
packets
1 bsdi (140.252.13.35) 20 ms 10 ms 10 ms
2 slip (140.252.13.65) 120 ms 120 ms 120 ms
```

En este caso, el output de traceroute, indica la dirección IP del destino y el TTL (que no excederá el valor 30). Los 40 bytes corresponden a 20 de IP, 8 de UDP y 12 de datos de usuario (estos 12 bytes contienen un número de secuencia, incrementado cada vez que el datagrama se envía, una copia del TTL de salida, y el tiempo en el que el datagrama se

envió). Para cada valor de TTL **se envían 3 datagramas**, y por cada uno se calcula el **RTT**. Si no se recibe una respuesta para ninguno de los tres datagramas, dentro de los 5 segundos, la salida se completa con un asterisco, y se envía el siguiente. El TTL se calcula en el emisor. Como los tiempos a cada router son totales, para calcular el tiempo de salto, es útil restar el RTT actual, respecto al paquete anterior. Por otra parte, el puerto destino UDP empieza en un número, por ej. 33435 y se incrementa por cada datagrama enviado.

Ejemplo de traza de traceroute:
El siguiente ejemplo de traza, está realizado sobre LAN. Sobre WAN, la variación de RTT puede ser mucho mayor.

```
1   0.0             arp who-has bsdi tell svr4
2   0.000586        arp reply bsdi is-at 0:0:c0:6f:2d:40
    (0.0006)
3   0.003067        svr4.42804 > slip.33435; udp 12 [ttl 1]
    (0.0025)
4   0.004325        bsdi > svr4: icmp: time exceeded in-
    (0.0013)        transit
5   0.069810        svr4.42804 > slip.33436: udp 12 [ttl 1]
    (0.0655)
6   0.071149        bsdi > svr4: icmp: time exceeded in-
    (0.0013)        transit
7   0.085162        svr4.42804 > slip.33437: udp 12 [ttl 1]
    (0.0140)
8   0.086375        bsdi > svr4: icmp: time exceeded in-
    (0.0012)        transit
9   0.118608        svr4.42804 > slip.33438: udp 12
    (0.0322)
10  0.226464        slip > svr4: icmp: slip udp port 33438
    (0.1079)        unreachable
11  0.287296        svr4.42804 > slip.33439: udp 12
    (0.0608)
12  0.395230        slip > svr4: icmp: slip udp port 33439
    (0.1079)        unreachable
13  0.409504        svr4.42804 > slip.33440: udp 12
    (0.0143)
14  0.517430        slip > svr4: icmp: slip udp port 33440
    (0.1079)        unreachable
```

Figure 8.1 `tcpdump` output for `traceroute` example from `svr4` to `slip`.

El mensaje "*time exceeded in transit*", refiere a los routers. El número de puerto origen se calcula teniendo en cuenta el PID del proceso.

Puntos a denotar de traceroute

1. No hay garantía de que que dos datagramas consecutivos sigan la misma ruta (la topología de red puede cambiar durante un traceroute).
2. No hay garantía de que los datagramas ICMP devueltos, tomen la misma ruta que el datagrama envíado: el RTT puede no ser 100% confiable.
3. La IP devuelta por el mensaje ICMP, es la interface por el cual el datagrama UDP arribó (lo que difiere con la opción de Record Route de IP, que graba la interfaz de salida). Por ello, la ruta generada de A->B, no será la misma que la generada de B->A. Ejemplo:
   Traceroute de svr4 a slip:

```
svr4 % traceroute slip
traceroute to slip (140.252.13.65), 30 hops max. 40 byte
packets
1 bsdi (140.252.13.35) 20 ms 10 ms 10 ms
2 slip (140.252.13.65) 120 ms 120 ms 120 ms
```

Traceroute de slip a svr4:

```
slip % traceroute svr4
traceroute to svr4 (140.252.13.34), 30 hops max, 40 byte
packets
1 bsdi (140.252.13.66) 110 ms 110 ms 110 ms
2 svr4 (140.252.13.34) 110 ms 120 ms 110 ms
```

Por ello, es útil imprimir, junto a las direcciones IP, el nemónico asociado (lo que requiere configurar un servicio DNS reverso en el host que ejecuta traceroute). Los nombres además, pueden cambiar, si las interfaces se configuran con nombres diferentes.

Opción de enrutamiento basado en el origen

El enrutamiento IP suele ser dinámico, ya que cada router toma las decisiones en tiempo real (durante el ruteo). Sin embargo, se puede especificar la ruta desde el origen, teniendo en cuenta dos estilos:

- **Estricto**: El emisor especifica el camino **exacto** que el datagrama debe seguir. Si un router no encuentra a otro en el siguiente salto, generará un mensaje ICMP con el mensaje “source route failed”.
- **Alternativo**: El emisor especifica el camino, pero el datagrama puede pasar entre diversas direcciones entre dos IP del camino.

Encabezado:

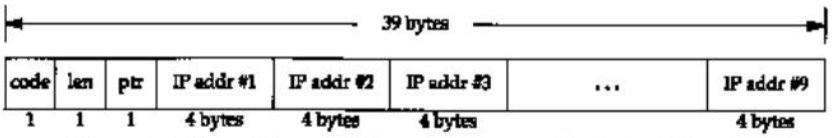
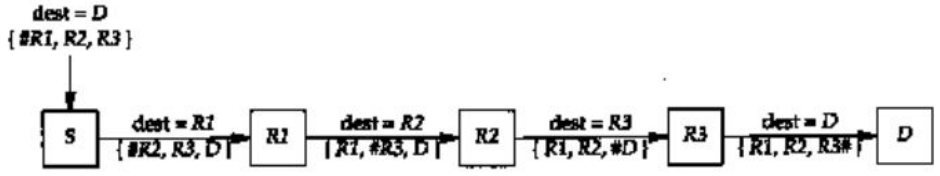


Figure 8.6 General format of the source route option in the IP header.

- El header es similar al usado en RR (también con un máximo de 9 direcciones).
- El código **0x83** se usa para encaminamiento alternativo y **0x89**, para encaminamiento estricto.
- **Len** y **ptr** tienen similar funcionalidad a RR.
- Las **opciones de enrutamiento** también se denominan “source and record route”, puesto que la lista de direcciones IP se actualiza durante el camino por los routers. El host origen genera una ruta (con la IP destino al final de la ruta), y cada router especifica el **nuevo destino** en el encabezado IP, según el valor del puntero, que se incrementa en 4.



Nota: La opción traceroute -g obliga a seguir una ruta no estricta, pero indefectiblemente deberá pasar por los routers especificados. Con loose source routing, se puede obtener un RTT más aproximado.

CAPÍTULO 9: RUTEO IP

El ruteo es una de las funciones más importantes de IP. Los datagramas a ser ruteados pueden ser generados en el host local o en algún otro. Estos últimos deben estar configurados para actuar como routers, ya que de otro modo, serán descartados.

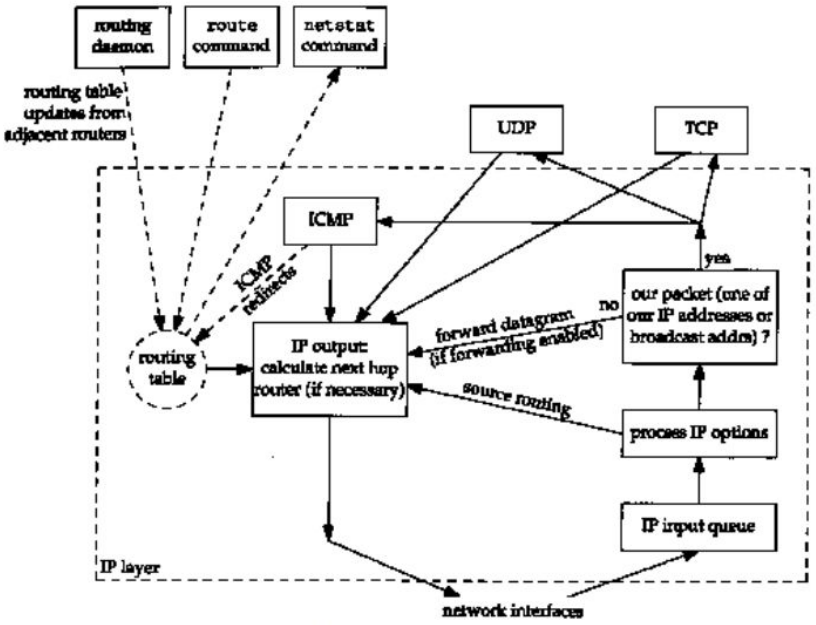


Figure 9.1 Processing done at the IP layer.

Principios de ruteo: IP utiliza una tabla de rutas para el proceso. Para la búsqueda dentro de esta, IP lleva a cabo las siguientes acciones (**en orden**):

1. Busca un matching con la dirección de host.
2. Busca un matching con la dirección de red.
3. Busca la entrada default (network ID 0).

Es importante diferenciar entre **mecanismo de ruteo** y **política de ruteo**. En el primer caso, el router verifica su tabla de ruta, para evaluar por cuál interfaz enviar el paquete, lo cual difiere de una política de ruteo, que es un conjunto de reglas que permite decidir qué rutas deben grabarse en la tabla. **IP realiza el mecanismo de ruteo**, mientras que un **demonio de encaminamiento provee la política de rutas**.

Tabla de enrutamiento simple:

```
svr4 % netstat -
rn
Routing tables
Destination     Gateway       Flags Refcnt Use       Interface
140.252.13.65   140.252.13.35 UGH   0      0         emd0
127.0.0.1       127.0.0.1     UH    1      0         lo0
default         140.252.13.33 UG    0      0         emd0
140.252.13.32   140.252.13.34 U     4      25043     emd0
```

El comando **netstat -rn** permite visualizar las tablas de rutas: La opción **-r** permite (en sí) ver la tabla de rutas (ya que netstat es útil también para visualizar conexiones TCP) y la opción **-n** permite imprimir las IP en formato numérico, en lugar de nombres.

Respecto al output, la primera línea de la salida indica que 140.252.13.35 es el gateway para llegar a la red 140.252.13.65.

Las flags mostradas, pueden tomar los siguientes valores:

- **U**: La ruta está ok.
- **G**: La ruta también es una puerta de enlace. Si esta bandera no está definida, el destino está conectado directamente.
- **H**: La ruta es a un host, es decir a una dirección de host completa (si la bandera no está definida es la ruta a una red).
- **D**: La ruta fue creada por una redirección.
- **M**: La ruta fue modificada por una redirección.

Notas:

- La flag G es importante, ya que permite diferenciar entre **ruteo indirecto** y **directo**. En el ruteo directo, tanto la dirección IP, como MAC de un paquete especifican la ruta final. En el ruteo indirecto, la dirección MAC específica la dirección de la puerta de enlace y la IP el destino final.
- La columna de **referencia de conteo** (refcnt), indica el número de usos activo para cada ruta. Refiere a las conexiones TCP que actualmente pasan por esa ruta.
- La columna **use** muestra el número de paquetes que han sido enviados por la ruta.
- La columna **interface**, indica el nombre de la interfaz.
- Un host puede tener una o más rutas por defecto. Esto significa que es posible enviar paquetes a una de estas, si una red o host no es encontrado. Normalmente se especifica con las flags UG.

Nota: El RFC especifica que la capa IP puede soportar varias rutas por defecto (las cuales suelen elegirse en round-robin). Algunas implementaciones, sin embargo, no soportan esto.

La entrada del final (especificada únicamente con la U), sin la H, indica que 140.252.13.32, es una red con la porción de host seteada a 0 (una ruta directamente conectada). En este caso, la máscara, puede inferirse a partir de un AND lógico entre la dirección del host y esta.

La complejidad de una tabla de rutas, depende de la topología de las redes a las que el host tiene acceso:

1. El caso más simple (pero menos interesante), es el de un host que no está conectado a ninguna red. Los protocolos TCP/IP se pueden utilizar para comunicarse consigo mismo. La tabla de ruta consiste en una entrada para la interfaz de **loopback**.
2. Si un host está conectado a una LAN, solo tendrá dos entradas: una para la interfaz **loopback** y otra para la **LAN**.

3. Otro caso ocurre cuando otras redes son alcanzables a través de un router (vía la entrada **default**).
4. El último caso consiste en agregar entradas a redes específicas (por ejemplo a determinados servidores).

*Nota: Cuando un host quiere **auto-enviarse** un datagrama, puede utilizar su hotsname, su IP, el loopack-name o la loopback-IP.*

Ejemplo de comunicación de un host consigo mismo:

```
ftp svr4 ftp
ftp 140.252.13.34
ftp localhost
ftp 127.0.0.1
```

En los dos primeros casos, el envío se realiza a través del driver Ethernet (en base al matching con la dirección IP 140.252.13.35) . En los otros dos, se realiza una entrega directa a la interfaz de loopback basado en la dirección IP (127.0.0.1, presente en la tabla). El paquete se envía al **driver loopback**, pero con **diferentes decisiones de ruteo**.

Inicializando una tabla de rutas

Para iniciar una tabla de rutas, primero deben incluirse las rutas a las redes directamente conectadas. Ejemplo:

```
route add default sun 1
route add slip bsdi 1
```

El tercer argumento, en ambos casos (default y slip), son los destinos. El cuarto argumento es el router y el argumento final es la métrica. Si la métrica es mayor a 0, se establece (automáticamente) la bandera G.

Otro ejemplo de tabla de ruta:

```
sun % netstat -rn
Routing tables
```

Destination	Gateway	Flags	Refcnt	Use	Interface
140.252.13.65	140.252.13.35	UGH	0	171	le0
127.0.0.1	127.0.0.1	UH	1	766	lo0
140.252.1.183	140.252.1.29	UH	0	0	sl0
default	140.252.1.183	UG	1	2955	sl0
140.252.13.32	140.252.13.33	U	8	99551	le0

En este caso, en la tercera línea, a diferencia de la tabla mostrada anteriormente, hay una ruta directa (sin la flag G) a un host, que corresponde a un link punto a punto.

Sin rutas al destino: Si no hay matching con ninguna dirección de host y red, y tampoco existe un default GW, se generará el mensaje "**host unreacheable**" o "**network unreacheable**". Si el error se genera en el emisor, se retorna un error **error a la capa de aplicación**, si se genera durante el reenvío de un datagrama, se genera un **mensaje ICMP**.

Mensajes ICMP de errores de host y redes inalcanzables
El mensaje “host inalcanzable” es generado por un router que recibe un datagrama que no puede reenviar. En algunos casos, los errores de hosts o redes inalcanzables se dan en los **routers de backbone**, donde algunos no tienen entradas por defecto y existe un **conocimiento “casi completo”** de la red.

Hosts ¿Forwardear o no forwardear? En la mayoría de implementaciones derivadas de Berkeley, para que un host rutee, se debe setear el valor la variable de kernel **ip-forwarding** (o nombre similar) a uno distinto de 0 (por defecto establecido de esta manera).
Mensajes ICMP de redirección
Un router genera un mensaje ICMP de redirección al emisor, en caso de que un datagrama deba ser envíado a través de un router diferente.

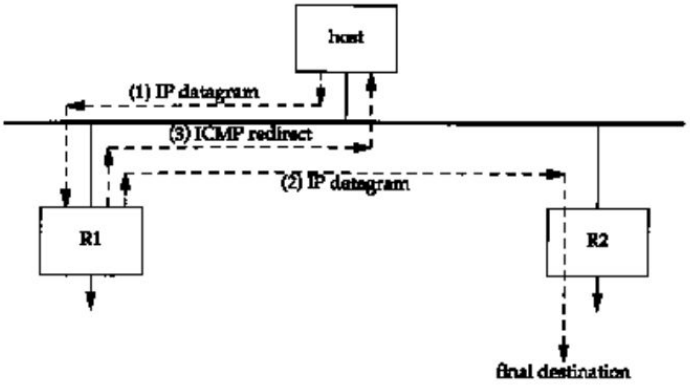


Figure 9.3 Example of an ICMP redirect.

Ejemplo:
1. En el anterior diagrama, el host envía un datagrama IP a R1 (router por defecto).
2. R1 recibe el datagrama y examina su tabla de rutas y determina que R2 es el siguiente salto. Cuando envía el datagrama a R2, R1 detecta que está enviando el datagrama por **la misma interfaz por la que llegó** (esto es clave para generar la redirección).
3. R1 genera un **ICMP redirect** al host, para que envíe los futuros datagramas a **ese destino** vía R2, en lugar de R1.

Nota: *El mensaje ICMP de redirección no sería generado si el datagrama fuera envíado por una interfaz diferente a la que llegó.*

La redirección tiene por **objetivo** que un host con un **mínimo conocimiento de rutas**, pueda construir una mejor tabla de rutas. Un host puede empezar con una ruta por defecto, y actualizar su tabla de enrutamiento con los redirect. *Cuando una ruta se agrega por redirect, se especifica con la flag D*.

Formato del mensaje de redirección:

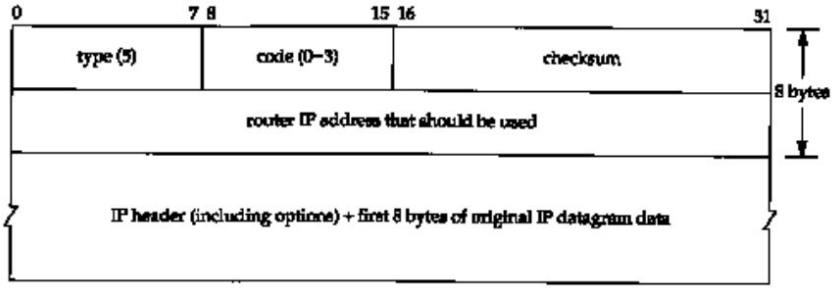


Figure 9.4 ICMP redirect message.

Existen 4 tipos de mensajes redirect, según el código:

- **0**: Redirección para red.
- **1**: Redirección para host.
- **2**: Redirección para Type of service y red.
- **3**: Redirección para Type of service y host.

Notas:

- **Es importante aclarar que las redirecciones se generan únicamente por los routers y no por los hosts** (ya que se tratan de mensajes ICMP).
- Debido a la presencia de subredes, los routers suelen envían re-direcciones de host, y no de red (por la porción de subnet-id).

Mensajes ICMP de descubrimiento de ruta

De forma inicial, se debe configurar la entrada default. Una alternativa para agregar rutas es aprovechar los **mensajes ICMP**. Existe, además, una solicitud ICMP que permite mediante broadcast o multicast, actualizar las tablas de rutas tomando las respuestas recibidas.

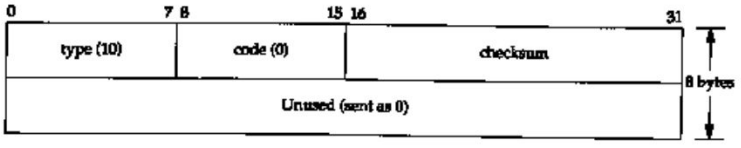


Figure 9.6 Format of ICMP router solicitation message.

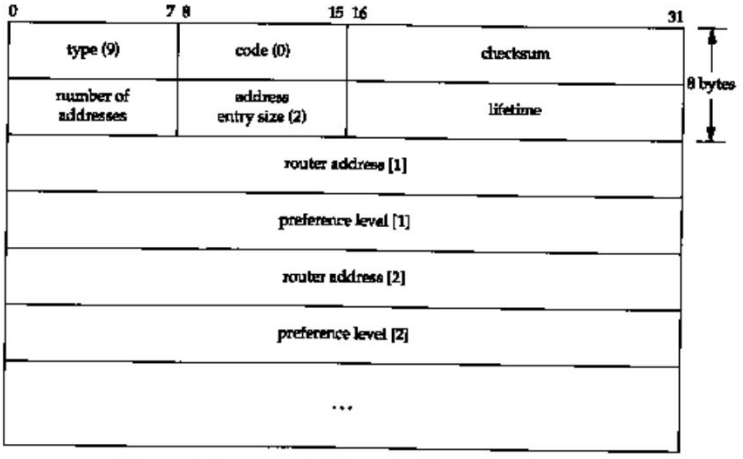


Figure 9.7 Format of ICMP router advertisement message.

En un único mensaje, un router puede anunciar múltiples direcciones (para ello utiliza el campo **número de direcciones**). **Address entry size** es un número de 32 bits, y es siempre 2. **Lifetime** es el número de segundos en el que el anuncio se considera válido. El valor **preferencia** indica el nivel de prioridad del router para ser utilizado como default GW. Otro uso de este campo, es en caso de la presencia de varios routers debido a subnetting: obviamente, el de primer nivel, deberá ser establecido con máxima prioridad.

Operación en un router: Un router **transmite avisos periódicamente** en todas las interfaces, mediante broadcast o multicast, en un intervalo de 450 a 600 segundos (10 min máximo), con un lifetime de 30 minutos. Se puede establecer el lifetime en 0, para indicar que la interfaz está deshabilitada. Además, un router puede recibir **solicitudes** de un host.

Operación en host: En principio, transmite **3 solicitudes**, con 3 segundos de diferencia cada una. Tan pronto como se recibe una respuesta, las solicitudes cesan. Un host también puede escuchar los cambios anunciados por routers adyacentes. Como los anuncios se suponen como máximo, cada 10 minutos, el lifetime de un router, pocas veces expirará.
Implementación: Los mensajes de descubrimiento de rutas, normalmente se generan por un proceso de usuario.

CAPÍTULO 10: PROTOCOLOS DE RUTEO DINÁMICO

Las rutas estáticas se crean cuando una interfaz se configura, y se agregan con el comando route, o se crean con un ICMP redirect. Estas rutas tienen sentido en redes pequeñas, en caso de que haya un “único punto” para conectar a otras redes o en caso de que no existan rutas redundantes.

Ruteo dinámico: El ruteo dinámico ocurre cuando los routers comentan a los routers adyacentes, información sobre las redes directamente conectadas. Los routers se comunican utilizando un **protocolo de ruteo**. El proceso de un router que se comunica con sus vecinos, se llama **demonio de ruteo** y actualiza el núcleo de la tabla de ruteo con información recibida de sus vecinos. El uso de un protocolo dinámico no cambia la forma en la que el kernel realiza el **mecanismo de ruteo**: lo único que cambia, es que las rutas se agregan y eliminan dinámicamente, todo el tiempo.

- El demonio de ruteo tiene políticas, para elegir qué rutas colocar en la **tabla** de rutas. Si este encuentra múltiples rutas a destino, el demonio elige (de alguna manera), la mejor. Si el demonio se encuentra con que un enlace está caído, puede eliminar las rutas afectadas, o agregar rutas alternativas.
- Normalmente, cada **sistema autónomo** (administrador de una red, por ej. una corporación o una universidad), utiliza un protocolo de ruteo determinado para comunicarse dentro de este, denominado **IGP (interior o intradomain gateway protocol)**. Los protocolos más conocidos son **RIP (Routing Information Protocol)** y **OSPF (Open Shortest Path First Protocol)**, un intento de reemplazo de RIP**.** Cada sistema autónomo intercambia sus rutas utilizando protocolos **EGPs** (Exterior Gateway protocols). Actualmente se utiliza BDP (Border Gw protocol).

RIP: Routing Information Protocol

Los mensajes RIP se intercambian utilizando datagramas UDP.

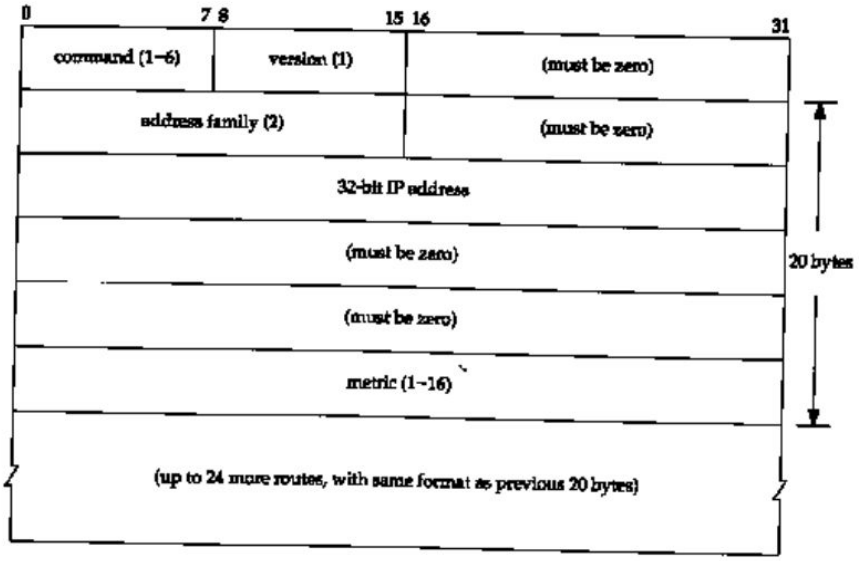


Figure 10.3 Format of a RIP message.

El campo **comando** es 1 si se trata de una petición y 2 si es una respuesta. Los 20 bytes inferiores refieren a la **dirección de familia** (siempre 2 para IP), una dirección IP, y una **métrica asociada** (normalmente, un contador de saltos). El tamaño de un mensaje RIP es menor a 512 bytes, por lo que se pueden advertir un máximo de 25 rutas, mediante el formato de 20 bytes (20*25+4=504). Debido a este límite, para enviar una tabla de ruta entera, se suelen requerir múltiples mensajes.

Operación normal: RIP utiliza el puerto 520 (UDP).

- **Inicialización**: Envía un paquete de petición a cada interfaz (normalmente, vía broadcast), solicitando las tablas de ruta. El comando de request es un comando con un 1, pero con address family en 0, donde la métrica se setea en 16: es un **request especial** que solicita una tabla de rutas **completa** al otro extremo.
- **Request recibido**: Si se recibe un **special request**, entonces se envía la tabla entera. De otra forma, se procesa cada entrada del request. Si el router dispone de una ruta a la dirección especificada, se setea la métrica, de otra forma, se configura en 16 (que indica "**infinito**", por lo que no existe ruta al destino).
- **Response recibido**: El receptor compara sus entradas con las recibidas. Ciertas rutas pueden ser añadidas, existentes pueden ser modificadas y otras eliminadas.
- **Actualizaciones regulares**: Cada 30 segundos, los routers envían sus **tablas de rutas completas**.
- **Actualizaciones disparadas**: Ocurren cuando la métrica de una ruta cambia. Sólo se envían las entradas necesarias.

Cada ruta tiene un **timeout** asociado (3 minutos). Si el timeout expira, la entrada se setea a infinito (16) y se marca para eliminación (lo que significa que el router que debería avisar cada 30 segundos, no lo está haciendo). La eliminación de una ruta de la tabla, se posterga otros 60 segundos, para asegurar la **propagación de la invalidación de la ruta**.

Métricas: La métrica usada en RIP son los saltos. El hop count para todas las interfaces conectadas es 1. Cada router suma 1, al pasar el paquete por otra interfaz. El máximo de saltos es 15. Un salto de 16 indica ruta inalcanzable.

Problemas

- RIP **no posee conocimiento de direccionamiento de subredes**.
- **Alto tiempo de convergencia**, usualmente minutos.
- Presencia de **loops** (evitables seteando el hop count a 16).

RIP v2: Extensión de RIP. No cambia el protocolo, pero agrega información adicional en los campos etiquetados como “must be zero”. Se agrega un **identificador de proceso** (que permite correr múltiples instancias de RIP), y **route tags**, para soportar protocolos de ruteo externos y **máscara de subred**. Además, soporta **multicasting** o broadcasting, lo que **reduce la carga de host a los que no les interesan los mensajes RIP**.

OSPF: Open Short Path First

Es una alternativa a RIP, que supera sus limitaciones. Es un protocolo de **estado de enlace**, opuesto a RIP que es un protocolo de vector de distancia (los mensajes que RIP envía están basados en los vectores recibidos por sus vecinos). Un protocolo de estado de enlace, no intercambia distancias con los vecinos, sino que prueba a cada uno de ellos, y luego comparte esta información con los demás vecinos: por ello, un protocolo de estado de enlace converge (estabiliza) mucho más rápido que uno de vector de distancia. OSPF, además, usa IP directamente (No usa UDP, ni TCP). Otras ventajas respecto a RIP:

1. **Más de una entrada en la tabla**, para un destino, según el tipo de servicio.
2. Interfaces se asociadas con un **costo** basado en throughput, RTT, confiabilidad, etc.
3. Si existen dos rutas con casi los mismos pesos, OSPF realiza **balanceo de carga** (distribuye el tráfico equitativamente).
4. Soporta **subredes**.
5. Los enlaces **punto a punto no necesitan direcciones IP** (son **redes no numeradas**), lo cual ayuda a destinar el uso de estas IP (no utilizadas) para otros casos.
6. Esquema de **autenticación simple**: Se puede especificar una contraseña en texto plano (similar al esquema RIP v2).
7. **Multicasting**: Reduce la carga en los sistemas que no participan de OSPF (aunque implementado también por RIPv2).

BGP: Border Gateway Protocol

Los protocolos de **enrutamiento externo**, se utilizan para intercambiar información entre diferentes sistemas autónomos. Uno de los mayores objetivos de BGP es reducir el tráfico en las redes de tránsito.

CIDR: Classless Interdomain Routing

Previene tablas de ruta demasiado grandes. El concepto básico es la **sumarización de rutas**. Por ejemplo, si se asignan 16 direcciones IPs a un solo sitio, o si están conectadas al mismo router, estas 16 direcciones se podrían resumir como **una única entrada** en la tabla de rutas. De este modo el ruteo comienza a tener una **organización jerárquica**.

Características necesarias para llevar a cabo la sumarización:

1. Las direcciones deben compartir los **mismos bits de orden superior**.
2. Los algoritmos de enrutamiento deben conocer las **máscaras de las IP**.

CIDR utiliza la técnica del **match más largo**, es decir, el que coincide en su mayor cantidad de bits, dentro de la máscara de 32 bits. El término "**sin clase**" refiere a que las decisiones de ruteo ahora se basan en operaciones de máscara, y **si una dirección es clase A, B o C, no importa**.

CAPÍTULO 11: UDP - USER DATAGRAM PROTOCOL

UDP es un protocolo de transporte simple, orientado al **datagrama**. Tiene como limitante el tamaño de un datagrama IP, lo cual es diferente de TCP (orientado al stream, lo que permite que una aplicación escriba una cantidad arbitraria de datos).

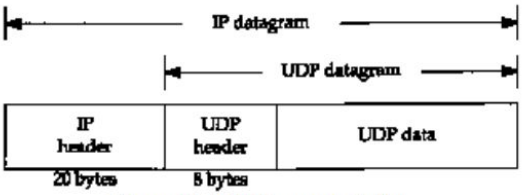


Figure 11.1 UDP encapsulation.

UDP **no da confiabilidad**: no garantiza que los paquetes lleguen a destino. Sólo proporciona **multiplexación** en puertos, y **checksum opcional**. Por otra parte, la aplicación necesita preocuparse acerca del tamaño del datagrama IP resultante: si excede la MTU de la red, será **fragmentado**.

Encabezado UDP:

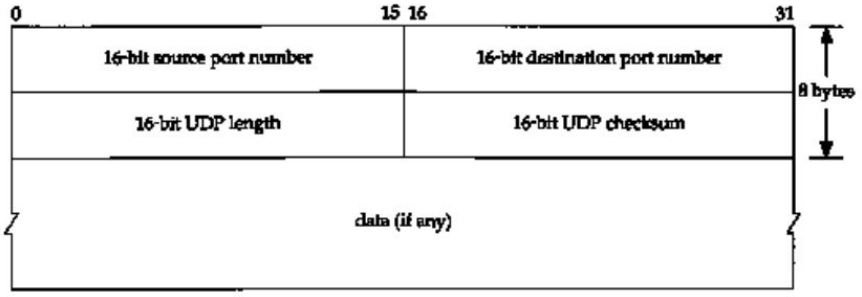


Figure 11.2 UDP header.

Los **números de puerto** identifican los **procesos origen y destino**. TCP y UDP usan los puertos destinos para demultiplexar datos entrantes desde IP. Los puertos **TCP y UDP** se multiplexan en **capas de transporte independientes**. Si un protocolo de capa de aplicación brinda **dos tipos de servicio** (confiable o no confiable), suelen utilizarse los **mismos números de puerto**. *Esto es por conveniencia y no es requerido por los protocolos*. El campo UDP length es el tamaño de la cabecera más los datos. El valor mínimo es 8 bytes (datagrama sin datos).

Checksum UDP: Al igual que TCP, cubre la cabecera y los datos (IP solo cubre el header). Con UDP es opcional, sin embargo, con TCP es obligatorio. Además, TCP y UDP incluyen una **pseudo-cabecera** en la suma de comprobación (con los campos de dirección de IP). El propósito es **asegurar que los datos hayan llegado al destino correcto** (aunque esto representa una violación del protocolo).

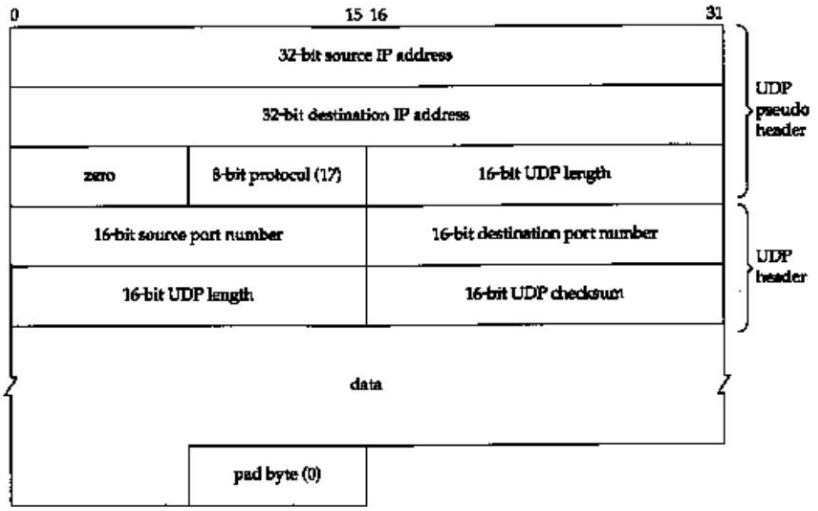


Figure 11.3 Fields used for computation of UDP checksum.

Notas:

- Se necesita un **padding** para computar el checksum, y así lograr una cantidad par de datos en el datagrama UDP.
- Si la **suma de comprobación es 0**, el emisor no ha computado checksum (sin embargo siempre está habilitada, por RFC). En otro caso, si el receptor detecta un error de checksum, el datagrama se descarta sin generar un mensaje de error.

Fragmentación IP

Si el datagrama a enviar, es más grande que la MTU, IP realizará **fragmentación** en el host emisor, o en un router intermedio. Sólo se **re-ensamblará en el destino**, en capa IP. El objetivo es que la fragmentación sea transparente para el protocolo de transporte, sin embargo, es posible una **degradación de rendimiento**, por el **tiempo de reensamblado**.

Es posible que un datagrama ya fragmentado, vuelva a ser fragmentado (más de una vez), por lo que es importante mantener, en el encabezado, la información necesaria para el posterior reensamblado. Para ello, se utiliza el campo **identificación**, que contiene un **único valor para cada datagrama IP que el emisor transmite** (este número se copia en cada fragmento de un datagrama dado). El bit "**More Fragments**", se activa para cada fragmento, excepto para el fragmento final. El **offset** determina el offset del actual fragmento en relación al datagrama original. Cuando un datagrama se fragmenta, el campo **total length** pasa a ser del tamaño del fragmento. Otro bit es el de "**don´t fragment**". Si está en 1, el datagrama no deberá ser fragmentado: Si el datagrama lo requeriría, se generaría un mensaje ICMP: "*Fragmentation needed but don't fragment bit set".*

Cuando un datagrama se fragmenta, cada sub-datagrama se convierte en un nuevo paquete, que se rutea independientemente, lo que produce que los **fragmentos puedan llegar en desorden**. Sin embargo, existe suficiente información para reensamblar correctamente en el receptor.

Aunque la fragmentación luce transparente, es una característica no deseable: si un fragmento se pierde, **todo el segmento del que era parte ese fragmento debe ser retransmitido** (por TCP, o la capa de aplicación, en caso de UDP). Como la fragmentación se puede realizar en routers intermedios, no es posible saber a ciencia cierta como se ha fragmentado el datagrama en el camino. Esto es razón suficiente para evitar **la fragmentación**. En UDP es fácil generar fragmentación, no así en TCP, ya que en gracias al MSS, es posible evitarlo.

Ejemplo de fragmentación:

```
bsdi % sock -u -i -nl -w1471 svr4 discard
bsdi % sock -u -i -nl -w1472 svr4 discard
bsdi % sock -u -i -nl -w1473 svr4 discard
bsdi % sock -u -i -nl -w1474 svr4 discard
```

```
1  0.0                bsdi-1112 > svr4.discard: udp 1471
2  21.008303          bsdi.1114 > svr4.discard: udp 1472
   (21.0083)
3  50.449704          bsdi.1116 > svr4.discard: udp 1473 (frag
   (29.4414)          26304:1480@0+)
4  50.450040 (        bsdi > svr4: (frag 26304:1@1480)
   0.0003)
5  75.328650          bsdi.1118 > svr4.discard: udp 1474 (frag
   (24.8786)          26313:1480@0+)
6  75.328982 (        bsdi > svr4: (frag 26313:2@1480)
   0.0003)
```

Figure 11.7 Watching fragmentation of UDP datagrams.

En el caso dado, los primeros dos datagramas (líneas 1 y 2), caben en tramas Ethernet y no son fragmentados. Sin embargo, en los frames 3 y 4 (datagrama 3) y 5, 6 (datagrama 4), si existe fragmentación. La salida frag 26304 (líneas 3 y 4), y frag 26313 (líneas 5 y 6), especifica el valor de identificación IP. El siguiente valor (1480), es el tamaño, excluyendo la cabecera IP. El primer fragmento en ambos datagramas contiene 1480 bytes de datos (8 de UDP y 1472 de datos). Los 20 de IP darían 1500 bytes exactamente. El segundo fragmento del primer datagrama (línea 4), contiene 1 byte de datos restantes. El segundo fragmento del segundo datagrama (línea 6), contiene 2 bytes extras de datos fragmentados. La fragmentación requiere que la porción de datos del fragmento sea múltiplo de 8 (en este caso 1480 es múltiplo de 8). El signo “+”, luego de cada fragmento, implica que existen más fragmentos. El primer fragmento comienza en 0 (líneas 4 y 4), mientras que el segundo comienza en el byte de offset 1480 (líneas 4 y 6). Es importante notar, que los números de puerto UDP no se imprimen en los fragmentos, ya que estos se encuentran encapsulados en el primero. La figura a continuación (como ejemplo al datagrama 3, del esquema 11.7, anterior), puede mostrar esto con más claridad:

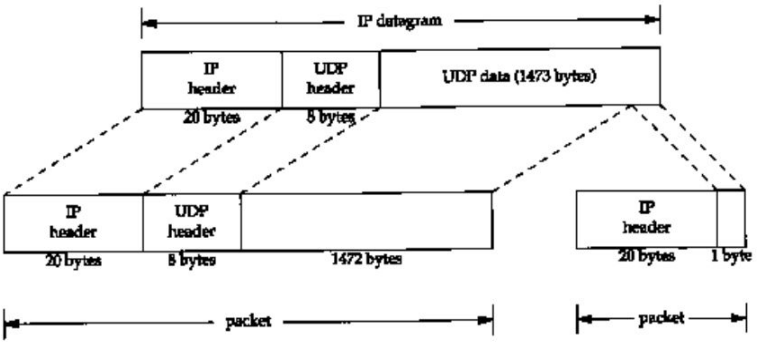


Figure 11.8 Example of UDP fragmentation.

ICMP: Unreachable Error (Fragmentation Required)

Se puede generar un mensaje ICMP "unreachable error" cuando un router recibe un datagrama con la flag DF (don't fragment) encendida, pero que requiere fragmentación (si un router no puede proporcionar este mensaje ICMP, el TTL se setea en 0). Este error se puede utilizar por el emisor, ya que incluye el valor de la MTU, lo que es útil para determinar la MTU más chica en el camino origen-destino (técnica se denomina **descubrimiento de MTU de camino (path MTU discovery)**), implementada mediante un ping que incrementa el tamaño de los paquetes hasta que haya fragmentación.

Determinando el MTU del camino utilizando traceroute: A pesar de que la mayoría de sistemas no soportan MTU discovery, se puede simular con una modificación a la versión de traceroute, con el envío de paquetes con la **flag DF activa**. El tamaño del primer paquete enviado no deberá superar la MTU de la interfaz de salida, y cada vez que se reciba el mensaje ICMP (de no fragmentación), y si el router envía la MTU dentro del mensaje ICMP, se utilizará este valor. En otro caso, se probará reduciendo el tamaño de paquete (como existen un tamaño de MTU probables, es conveniente que el programa se mueva al valor inmediatamente más chico).

Ejemplo (en este caso el router no retorna la MTU de la interfaz de salida):

```
sun % traeeroute.pmtu slip
traceroute to slip (140.252.13.65), 30 hops max
outgoing MTU = 1500
1 bsdi (140.252.13.35) 15 ms 6 ms 6 ms
2 bsdi (140.252.13.35) 6 ms
fragmentation required and DF set, trying new MTU = 1492
fragmentation required and DF set, trying new MTU = 1006
fragmentation required and DF set, trying new MTU = 576
fragmentation required and DF set, trying new MTU = 552
fragmentation required and DF set, trying new MTU = 544
fragmentation required and DF set, trying new MTU = 512
fragmentation required and DF set, trying new MTU = 508
fragmentation required and DF set, trying new MTU = 296
2 slip (140.252.13.65) 377 ms 377 ms 377 ms
```

Si se dispone del MTU, la salida se reduce a:

```
sun % traceroute.pmtu slip
traceroute to slip (140.252.13.65), 30 hops max
outgoing MTU = 1500
1 bsdi (140.252.13.35) 53 ms 6 ms 6 ms
2 bsdi (140.252.13.35) 6 ms
fragmentation required and DF set, next hop MTU = 296
2 slip (140.252.13.65) 377 ms 378 ms 377 ms
```

Hoy en día, el valor estándar de MTU para las LANS es de 1500 bytes (Ethernet).

Path MTU Discovery con UDP: ¿Qué sucede cuando una aplicación escribe datagramas muy grandes para un enlace intermedio? En este caso, el emisor tratará de realizar fragmentación (evitandola en los routers), y utilizando la flag **DF** (Don't fragment). Luego, en base a los mensajes ICMP, reducirá el tamaño de cada fragmento generado. Pueden darse dos casos:

1. **El mensaje ICMP de un router no retorna con la MTU del próximo salto**: En este caso el origen irá "probando" con la reducción del tamaño de los fragmentos. Luego de una serie de intentos, seteará la flag DF en 0, y enviará fragmentos del tamaño de la última MTU probada (aunque puede que no sea la correcta).
2. **El mensaje ICMP de un router retorna la MTU del próximo salto**: En este caso, el origen seteará el tamaño de los fragmentos en la MTU. Si el origen logra fragmentar un paquete en la mínima MTU posible de un enlace intermedio, se generará una ventaja en relación a que si lo hicieran los dispositivo intermedios: En tal caso, se generarían menor cantidad de fragmentos.

Ejemplo: Dada la MTU mínima de 296 bytes y generados datagramas de 650 bytes, si el origen, en base a mensajes ICMP, realiza una última prueba generando paquetes de 576 como máximo, se generarían 4 fragmentos: 2 por el emisor, uno de 576 (544 de datos, por encabezados de 20 bytes de IP y 8 bytes de UDP. Nota: no se genera de 548, para que sea múltiplo de 8) y otro de 106 bytes. El primer fragmento debería volver a ser fragmentado en el router intermedio de 296, lo que generaría 3 fragmentos extras: teniendo en cuenta el segundo fragmento del origen, serían en total 4 fragmentos. *Si el origen obtendría información sobre la MTU mínima (296), sólo se generarían 3 fragmentos: 650/(296-20 de header IP)=2.35.*

Problemas entre UDP y ARP: Si la caché ARP está vacía, varios fragmentos desde un origen, podrían generar varios ARP request en un lapso corto de tiempo (ya que no se envían en orden). Para evitar una **inundación por ARP**, en ciertas implementaciones, sólo el último datagrama espera un ARP reply, y los demás se descartan. Esto produce que un protocolo de aplicación deba dar cuenta de ello (ya que UDP no proporciona fiabilidad).

Tamaño máximo de datagrama UDP: El máximo de un datagrama IP es 65535 bytes (impuesto por los 16 bits de la cabecera IP). Con un header IP de 20 bytes y uno UDP de 8 bytes, la cantidad máxima de datos de usuario es 65507 bytes (la mayoría de implementaciones, sin embargo, proveen menos). Muchas veces, esto no es aplicable en la práctica, ya que los tamaños de buffers son menores. Normalmente se sugiere que los datagramas sean menores a 512 bytes (lo que además debería evitar fragmentación).

Truncado de datagrama: Como en las interfaces UDP se especifica un tamaño máximo de datos a leer cada vez, ¿qué sucede si la aplicación recibe un datagrama más grande de lo que puede leer? La mayoría de implementaciones (Berkeley, por ejemplo), truncan el datagrama, descartando cualquier exceso de datos. Otras implementaciones permiten que el resto se lea en subsecuentes operaciones de lecturas.

ICMP: Source Quench error (Error de sofocación de origen): Puede suceder cuando un router o host envía un mensaje ICMP indicando que la cantidad de datagramas recibidos supera la velocidad a la que pueden ser procesados (normalmente porque el origen posee un ancho de banda muy grande, e inunda a algún receptor, ya que no hay control de congestión). Un sistema intermedio no está obligado a generar dicho error.

Diseño de servidor UDP: A pesar de que el diseño de un cliente es simple, el de un servidor puede ser complejo: mientras que un cliente se conecta para transmitir algo, un servidor debe estar siempre a la escucha. En un servidor, los paquetes de un cliente se identifican en base a la tupla dirección ip:puerto origen e ip: puerto destino.

Cola de entrada UDP: Los datagramas UDP se encolan a medida que llegan. Esta cola se vacía a medida que la aplicación realiza lecturas. Es importante aclarar que la cola de UDP de FIFO, mientras que por ejemplo en ARP es LIFO.

Wildcards

En un servidor UDP, se pueden utilizar **wildcards** (los datagramas UDP se aceptan desde cualquier interfaz local). Ejemplo: *.7777. Si en cambio, se setea una IP (ej. 140.285.1.29), los datagramas enviadas a otra IP (ej. 140.282.1.35) y al puerto 7777, serán descartados. En este último caso, sería posible iniciar distintos servicios en el mismo puerto, pero con diferente IP (con wildcard, se inicia el mismo servicio para un conjunto de IPs).

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	(state)
udp	0	0	*.8888	*.*	
udp	0	0	140.252.13.63.8888	*.*	
udp	0	0	127.0.0.1.8888	*.*	
udp	0	0	140.252.13.33.8888	*.*	
udp	0	0	140.252.1.29.8888	*.*	

La opción **SO_REUSEADDR**, permite este comportamiento. Para el caso dado, el comodín tomará solo aquellos paquetes que no estén cubiertos por los siguientes 4 sockets. Nota: algunas implementaciones no permiten tal comportamiento, mostrando el error "*Address already in use"*.

Restricción de IPs de emisores

Local Address	Foreign Address	Description
localIP.lport	*foreignIP.fport*	restricted to one client
localIP.lport	*.*	restricted to datagrams arriving on one local interface: *localIP*
. lport	*.*	receives all datagrams sent to *lport*

Varios destinatarios por puerto: A pesar de que no se especifica en el RFC, la mayoría de implementaciones sólo permite una aplicación por vez en escucha para una determinada dirección IP y puerto. Aunque algunas implementaciones no permitan la reutilización de direcciones, esto si es posible en sistemas que soporten multicasting (donde múltiples extremos pueden utilizar la misma IP y puerto UDP). En la mayoría de implementaciones Berkeley (superior a 4.4) esto debe especificarse con **SO_REUSEPORT**, lo que permite múltiples endpoints en el mismo puerto. Cuando llega un datagrama con una dirección multicast, entonces, una copia se entrega a cada uno de ellos. Cuando llega un datagrama a una dirección unicast, solo una copia se entrega a uno de los endpoints (a cuál, lo decide la implementación).

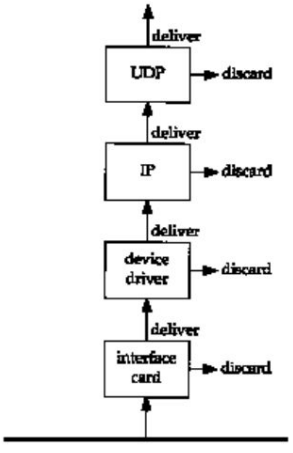
CAPÍTULO 12: BROADCASTING Y MULTICASTING

Broadcasting y multicasting solo aplican a UDP, donde enviar un mensaje a múltiples destinatarios tiene sentido (TCP es orientado a la conexión).

Tipos de direcciones

- Una dirección de destino especifica una única interfaz denominada **unicast**.
- Una dirección de **broadcast**, se utiliza en el protocolo ARP y RARP.
- Una dirección de **multicast** se ubica entre en unicast y el broadcast: el frame es envíado a un conjunto de host en base a un grupo multicast.

Para entender el broadcast y la multidifusión es necesario entender que, en el primer caso, el filtro toma lugar en cada host, cada vez que una trama pasa en el cable. Normalmente, la interfaz de red recibe los paquetes que tienen su interfaz física o la dirección de broadcast. Adicionalmente, las interfaces pueden ser puestas en modo promiscuo para recibir una copia de cualquier frame.



Broadcasting: **Durante un broadcast, cada protocolo realiza un filtrado**. Si la dirección de red es 01:00:00:00:00:00 (o ff:ff:ff:ff:ff:ff, en el caso de Ethernet), o matchea con la dirección física, el paquete se pasa al controlador Ethernet, que descarta el paquete si el checksum es incorrecto. Posteriormente IP puede realizar un filtrado (basado en la interfaz origen y destino). UDP y TCP pueden realizar un filtrado basado en el puerto. El problema **es el procesamiento que tiene lugar en un host que NO está interesado**. *Por ejemplo, si hay 50 hosts compartiendo un cable, pero sólo 20 están interesados en recibir datos de una aplicación, el procesamiento de los 30 hosts restantes representa una carga para los hosts.* El **multicast** intenta reducir la carga en los hosts que no están interesados en la aplicación.

Nota: En caso de que un host posea más de una interfaz (sea multi-homed), y realice un broadcast, dependiendo la implementación, un paquete se enviará por todas, o sólo una.

La dirección de broadcast límite es 255.255.255.255. Se puede usar como dirección destino, cuando un host no conoce su máscara de subred o incluso su dirección IP. Un datagrama con una dirección de broadcast nunca se forwardea por un router. Tipos de difusión:

- **Dirigida a una red**: Las direcciones de broadcast para redes directamente conectadas tienen un host ID con todos los bits en 1. Una red directamente conectada tiene la dirección netid.255.255.255, donde netid es la red directamente conectada.
- **Dirigida a una subred**: La dirección de difusión a subnet es el host ID con unos, pero un subnet id específico. Esto requiere conocer la máscara.
- **A todas las subnets**: Requiere conocimiento de la máscara del destino.

Multicasting: Provee dos servicios para una aplicación:

1. Entrega a múltiples destinos.
2. Solicitud de un servidor por parte de un cliente.

Grupos de direcciones Multicast

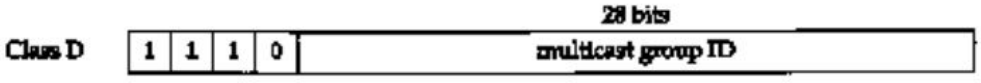


Figure 12.2 Format of a class D IP address.

Una dirección de grupo multicast es la combinación de los 4 bits de alto orden de 1110 y el ID de multicast. El conjunto de host que escuchan en una IP multicast particular se denomina **host group** (grupo de host). Existen grupos “predefinidos” por IANA, por ejemplo 224.0.0.1 hace referencia a “todos los sistemas en esta subred”, 224.0.0.2 significa “todos los routers en esta subnet”. La dirección multicast 224.0.1.1 es para NTP, 224.0.0.9 es para RIP-2 y 224.0.1.2 para SGI’s (Silicon Graphics).

Convirtiendo una dirección multicast en una dirección ethernet

Las direcciones ethernet para multidifusión están en el intervalo 01:00:5e:00:00:00 a 01:00:5e:7f:ff:Ff (el 01, como se mencionó anteriormente, indica broadcast, los restantes

refieren a la dirección multicast). Sin embargo, el controlador Ethernet, debe seguir realizando el filtrado.

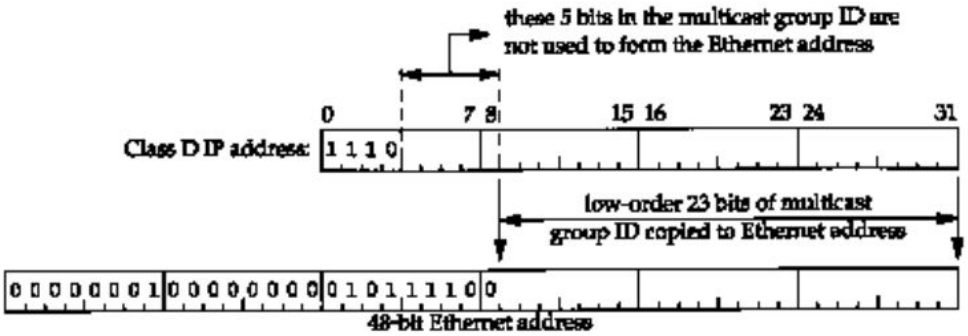


Figure 12.3 Mapping of a class D IP address into Ethernet multicast address.

El multicast en una sola red física es simple. El proceso emisor especifica una dirección IP que es de multicast, y el controlador convierte esta a una dirección Ethernet. Los procesos receptores deben notificar a IP que desean recibir datagramas multicast, y este a su vez al controlador (esto se denomina “unirse a un grupo de recepción”).
Resumen: *El broadcast es el envío paquetes a todos los host de una red/subred (los routers no pueden encaminar broadcasts). El multicast es el envío de paquetes a un* ***conjunto*** *(subconjunto del total) de hosts en una red. Cada capa puede descartar el paquete por diferentes razones. Hay 4 tipos de direcciones de broadcast: limitadas, dirigidas a la red, dirigidas a una subred, y dirigidas a todas las subredes conectadas (el más común es el dirigido a una subred).*

Una dirección de clase D se denomina grupo de multicast. Se convierte a una dirección Ethernet, colocando sus 23 bits inferiores en una dirección Ethernet fija. El mapeo no es único, lo que requiere filtros adicionales por uno de los módulos de protocolo.

CAPÍTULO 14. DNS: DOMAIN NAME SERVER

DNS es una **base de datos distribuida** que utilizan las aplicaciones TCP/IP para **mapear nombres de hosts a direcciones IP**. Se usa el término distribuido ya que **no hay un punto único de información**: cada sitio (departamento universitario, campus, compañía, etc), mantiene su propia base de datos y corre un programa servidor que otros clientes pueden consultar. Desde el punto de vista de la aplicación, el acceso DNS es a través de un **resolver**, que no es parte del kernel del sistema operativo, como si lo son los protocolos TCP/IP por debajo de la capa de aplicación, quienes “no saben” de la existencia de DNS. Por otra parte, en UNIX, es posible invocar al resolver, mediante la función gethostbyname(name), que dado un nombre de host, retorna su IP y gethostbyaddr(IP), que toma una IP y retorna una serie de nombres. El resolver puede contactar uno o más **servidores de nombres (name servers)** para realizar el mapeo. DNS es importante ya que una aplicación debe consultar los nombres de dominio, antes de abrir una conexión TCP o enviar datagramas sobre UDP.

Postulado básico: DNS es un espacio de nombre jerárquico (similar al sistema de archivos de UNIX).

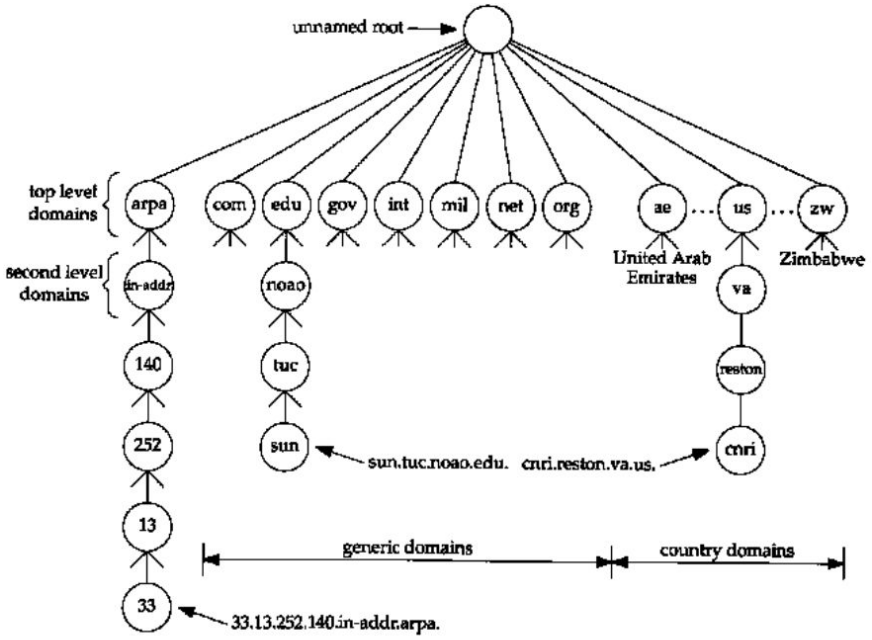


Figure 14.1 Hierarchical organization of the DNS.

Cada nodo dispone de una **etiqueta** de hasta 63 caracteres (no se diferencian mayúsculas de minúsculas). El nodo root, es especial, y no posee etiqueta. El **nombre de dominio** de cualquier nodo es una sucesión de labels separados por punto, empezando en un nodo hasta la raíz. Un nombre de dominio que finaliza con un punto se denomina **nombre de dominio absoluto** o **nombre de dominio completo (fully qualified domain name, FQDN)**. Ejemplo: **sun.tuc.noao.edu.**. Si el nombre de dominio no finaliza con punto, puede asumirse que necesita ser completado. Los **TDL (Top level domains)** se dividen en tres:

1. **arpa** es un dominio especial usado para mapear IP a direcciones de nombres.
2. **7 dominios de 3 caracteres**, llamados dominios genéricos (dominios organizaciones, ej: com, edu, gov, int, mil, net, org).
3. **Dominios de 2 caracteres** basados en los códigos de país (country domains, ejemplo: ar).

Muchos países forman dominios de segundo nivel, similares a los dominios genéricos. Ejemplo: com.ar.

Una característica importante de DNS es la **delegación de responsabilidad**. Mientras una entidad superior (NIC), mantiene los dominios genéricos, delega los inferiores a entidades inferiores, por zonas. Una **zona** es un subárbol de DNS y está **administrada de forma separada**. Los dominios de segundo nivel pueden, además, dividir su zona en otras más pequeñas. Por ejemplo, una universidad puede dividir sus zonas en departamentos. Una vez que se delega una zona, le corresponde a su administrador generar múltiples servidores de nombre para esa zona. A cada nuevo servidor se le asigna un nombre y una dirección IP.
Se dice que un servidor de nombres tiene **autoridad** para una o varias zonas. El administrador de una zona debe proporcionar un servidor de nombres primario y otros secundarios, que deben ser redundantes, para evitar un único punto de fallo.

Transferencia de zona

Las zonas de servidores primarios, se cargan en los secundarios. Cuando un servidor secundario obtiene información de una zona primaria, se lleva a cabo un proceso denominado **transferencia de zona**. Cuando se agrega un nuevo servidor, el administrador debe agregar la información apropiada (nombre y direcciones IP, como mínimo), al archivo que utiliza la zona primaria. Los servidores secundarios realizan consultas al primario (normalmente cada 3 horas), para verificar si debe llevarse a cabo una transferencia de zona, en el caso de que existan nuevos datos.

¿Qué hace un servidor de nombres cuando no contiene la información consultada?

Normalmente contacta otro servidor de nombres (de esto se trata la naturaleza distribuida de DNS). Como no todos los servidores de nombres conocen a los otros, cada servidor debe saber cómo comunicarse con un **root server** (básicamente, debe conocer la IP de alguno de ellos). Si un servidor de nombres consulta a un root server, sobre un dominio de nivel 2, el root server, otorgará los dominios de dicho nivel al solicitante. Luego, si el cliente necesita consultar a un dominio de nivel 3, consultará al servidor correspondiente de nivel 2 (información obtenida del root). De este modo, la consulta DNS es un **proceso iterativo**.

Formato de mensaje DNS

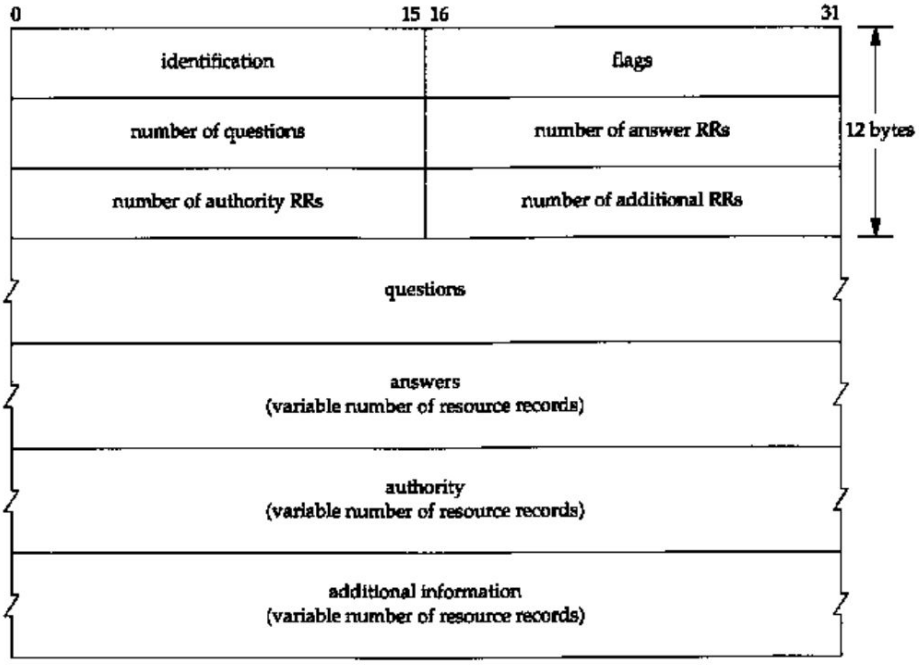


Figure 14.3 General format of DNS queries and responses.

El mensaje tiene 12 bytes fijos, seguido por 4 campos de tamaño variable. Al campo **identificación** lo setea el cliente, y permite que este matchee responses y requests.

El campo flag de 16 bits, está dividido en los siguientes subcampos:

Figure 14.4 *flags* field in the DNS header.

- **QR**, de 1 bit: 0 indica que el mensaje es una query, 1 que es un response.
- **opcode**, de 4 bits: 0 (el valor normal), indica una query estándar. 1 indica una query inversa y 2 indica bad request.
- **AA**: Indica respuesta autorizada, “authoritative answer”.
- **TC**: Significa truncado, en caso de que el mensaje UDP excede los 512 bytes.
- **RD**: Recursion desired, o recursión deseada en query. Indica al servidor al que se le envía el paquete, que debe resolver la query por el mismo (**consulta recursiva**). Si el bit no está activo, y la respuesta no es autoritativa, el servidor consultado retornará una lista de servidores a los cuales es posible contactar para averiguar la respuesta (**consulta iterativa**).
- **RA**: Recursion available (recursión disponible), indica, si está en 1, que el servidor soporta recursión.
- **zero**, es un campo de 3 bits, que debe establecerse en 0.
- **rcode**, es un campo de 4 bits, con el código de retorno. Los valores comunes son 0 (no error) y 3 (name error). Este último solo puede ser generado por el servidor de nombres con autoridad, e indica que el nombre especificado en la consulta no existe.

Los restantes 4 16 bits especifican el número de entradas en los 4 campos de tamaño variable que completan el registro. Para una query el “número de preguntas” normalmente es 1, y los otros tres son 0. Similarmente, para una respuesta, el campo número de respuesta es de al menos 1, y los campos restantes 0.

Porción DNS de query

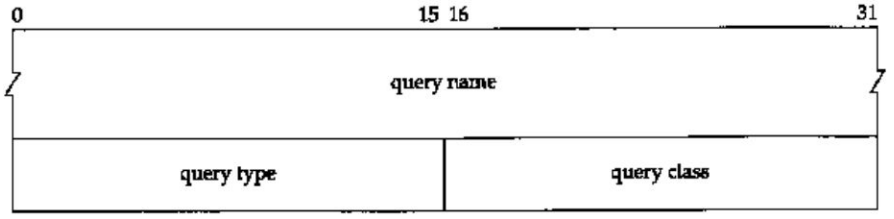


Figure 14.5 Format of *question* portion of DNS query message.

query name refiere al nombre que se desea averiguar. Es una secuencia de una varias etiquetas. Cada label comienza con un campo de 1 byte que indica el número de bytes a continuación. Cada nombre finaliza con el byte 0 (un label de tamaño 0, el del root). Cada campo de tamaño de label está limitado de 0 a 63 (no se utiliza padding). Ejemplo del nombre de dominio **gemini.tuc.noao.edu**:

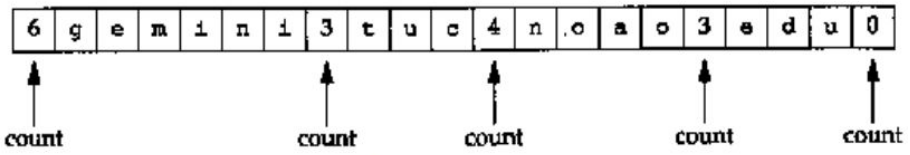


Figure 14.6 Representation of the domain name `gemini.tuc.noao.edu`.

Cada **query** y **response** (este último, llamado resource record), tiene un tipo. Hay 20 diferente, pero algunos son obsoletos:

Name	Numeric value	Description	*type?*	*query type?*
A	1	IP address	*	*
NS	2	name server	*	*
CNAME	5	canonical name	*	*
PTR	12	pointer record	*	*
HINFO	13	host info	*	*
MX	15	mail exchange record	*	*
AXFR	252	request for zone transfer		*
* or ANY	255	request for all records		*

Figure 14.7 *type* and *query type* values for DNS questions and responses.

La query más común es la de tipo A, que indica que se desea averiguar la IP de un nombre. Una query/response PTR indica que se solicita/responde una IP correspondiente a un nombre. El campo **query class** es siempre 1 (significa Internet address, IP).

Porción de registro de recurso DNS (respuesta)
Los 3 campos finales en un mensaje DNS (respuesta, autoridad e información adicional), tienen un formato similar denominado “resource record” o RR.

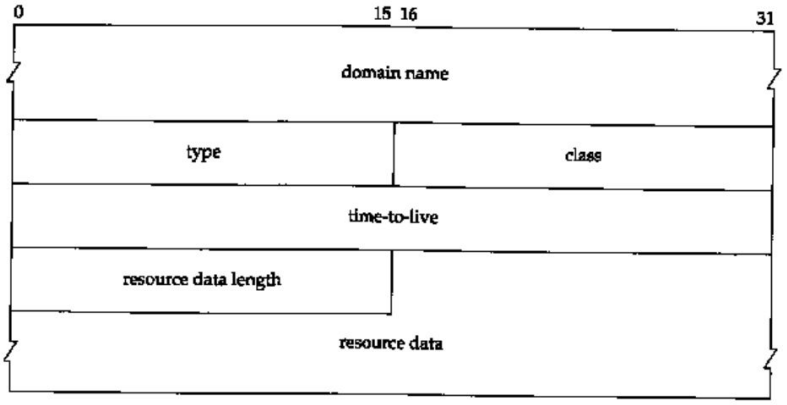


Figure 14.8 Format of DNS resource record.

Domain name indica el nombre al cual corresponden los RR (idem formato que un query name). **Tipo** especifica el tipo de código RR (mismo formato que RR, 1 para IP). **TTL** define el número de segundos que un RR se puede guardar en caché. El TTL más común es 2 días. **El tamaño de recursos de datos** especifica la cantidad de resource data. El formato de los datos depende del tipo. Para un tipo 1 (recurso tipo A), el tamaño de dato de recurso es una IP de 4 bytes.

Ejemplo:

```
1  0.0                    140.252.1.29.1447 > 140.252.1.54.53: 1+ A?
                          gemini.tuc.noao.edu. (37)
2  0.290820 (0.2908)      140.252.1.54.53 > 140.252.1.29.1447: 1* 2/0/0 A
                          140.252.1.11 (69)
```

Figure 14.10 `tcpdump` output for name server query of the hostname `gemini.tuc.noao.edu`.

En la línea 1, 1+ indica que el campo identificación es 1, y el + indica **recursión deseada** (**por defecto el resolver realiza una query por recursión**). A? indica una query de tipo A (se solicita una dirección IP). El signo de pregunta indica query. El resolver, además añade un punto final al query name, que indica nombre de dominio absoluto. El tamaño de los datos de usuario es de 37 bytes (12 bytes de tamaño fijo, 21 bytes del query name y 4 bytes del query type y query class). En la línea 2, 1* indica el número de identificación seguido de una flag AA (Authoritative answer). El número 2/0/0 muestra el número de RR en el final de los 3 campos de tamaño variable en el response: 2 respuestas RRs, 0 authority RRs y 0 RRs adicionales. La IP muestra la respuesta de tipo A. Se obtienen 2 respuestas ya que el host es multihomed.

El detalle final es que el tamaño de datos UDP es de 69 bytes, esto se da ya que:

1. La consulta también se retorna en la respuesta.
2. Como puede haber muchas repeticiones de nombres de dominio, se utiliza un método de compresión.

Pointer Queries

Las consultas de tipo puntero retornan un **nombre dada una dirección IP**. La sección arpa y su subsección in-addr permiten realizar este tipo de consultas. Por ejemplo, si se tiene acceso al dominio nao.edu, su porción en in-addr.arpa correspondería a 140.252. Los siguientes subniveles podrían dar lugar a la resolución de una ip final. En el caso, para sun, como los nombres se leen de abajo hacia arriba, la IP quedaría conformada por 33.13.252.140 (33.13.252.140.in-addr.arpa). Por ello, las respuestas de las pointers query, se leen de arriba hacia abajo (resolución inversa).

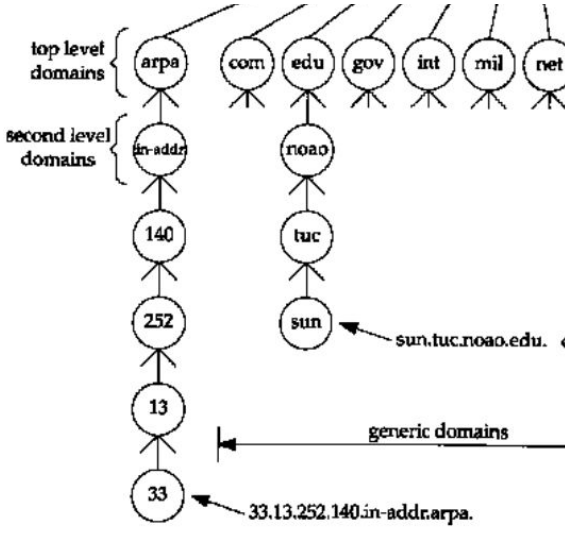


Figure 14.1 Hierarchical organizatio

Si no habría una rama separada para las direcciones, las resoluciones de IP a Host, podrían tomar tiempo (y aunque esto resulte confuso, es una manera inteligente de paliar el problema). Desde el punto de vista de la aplicación, esto es transparente, ya que es solucionado por el resolver.

Nota: FQDN (Full qualified domain name) es un nombre que incluye el del equipo y el de su dominio asociado. Por ejemplo, si una computadora se llama serv1, y el dominio es

bar.com, el FQDN será serv1.bar.com (Normalmente se agrega un punto al final). La longitud máxima de un FQDN es de 255 bytes, restringido a 63 bytes en cada etiqueta.

Comprobación de suplantación de nombre de host
En ciertos casos, para evitar la suplantación de nombre de hosts, luego que se recibe la respuesta a una consulta de un servidor, se realiza una consulta inversa con el nombre recibido (que devolverá el nombre de la IP resuelta). Si las respuestas coinciden, no hay suplantación.

Resource Records (Nombres de recursos)

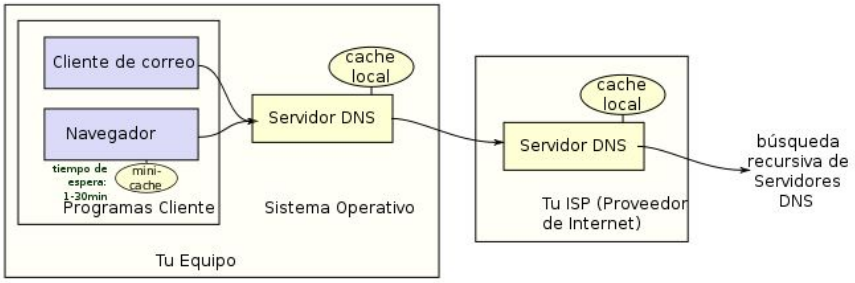
- **A**: Define una dirección IP. Se almacena como un valor binario de 32 bits.
- **PTR**: Utilizado para consultas de puntero. La dirección IP se representa como un nombre de dominio (secuencia de etiquetas) en el dominio in-addr.arpa.
- **CNAME (alias)**: Significa nombre canónico. Se representa como un nombre de dominio (secuencia de labels).
- **HINFO**: Dos cadenas de caracteres que especifican la CPU y el sistema operativo. No todos lo proveen.
- **MX**: Mail exchange records. Cada registro MX tiene asignado un valor de preferencia de 16 bits, lo que permite la existencia de más de 1 registro MX (en caso de que exista más de un servidor). Si existen múltiples registros MX para un destino, se utilizan en round-robin, empezando con el de menor valor de preferencia. Si un intento es fallido, se pasa al siguiente.
- **NS**: Name server record. Especifican el servidor de nombres con autoridad para un dominio.

Caché: Para reducir el tráfico DNS en Internet, todos los servidores emplean una caché. De modo que si, diversas aplicaciones realizan consultas DNS, todas estas se almacenan en el servidor local.

¿UDP o TCP?: Normalmente, se utiliza UDP, pero cuando el servidor retorna la flag TC activada (“truncated”), significa que el mensaje de respuestas excede los 512 bytes, por lo que el resolver realiza nuevamente la consulta, pero utilizando TCP. TCP también se utiliza cuando un servidor secundario realiza una transferencia de zona.

En definitiva, DNS es una parte esencial para cualquier host conectado a Internet, pero usado también en redes privadas. Las aplicaciones contactan a los resolvers para convertir nombres de dominio a direcciones IP y viceversa. **Los resolvers contactan un servidor DNS local, y este servidor puede contactarse con otros servidores root.** Todos los mensajes DNS (consultas y respuestas) tienen el mismo formato de mensaje. En las respuestas, se suelen realizar optimizaciones (como el retorno de RRs adicionales), para evitar que el solicitante realice consultas “extras”.

Arquitectura gráfica de DNS



CAPÍTULO 17. TCP: TRANSMISSION CONTROL PROTOCOL

TCP provee un servicio totalmente diferente a UDP, otorgando:

- Servicio **orientado a la conexión**: Dos hosts (cliente y servidor), deben establecer primero una conexión TCP.
- **Confiabilidad**: Los datos de aplicación se dividen en trozos a enviar, según la consideración de TCP (UDP genera un datagrama del tamaño en cuestión). La unidad de protocolo se llama **segmento**, y se inicia un timer, esperando confirmación del mismo. Si la confirmación no se recibe, el segmento se **retransmite**. TCP implementa timeout adaptativo y estrategias en la retransmisión. Si TCP recibe datos de otro endpoint, envía confirmación (no inmediatamente, sino que con un retraso de unos segundos). TCP mantiene un checksum sobre el header y los datos. Si en el otro extremo el checksum calculado es distinto al recibido, se descarta el segmento y no se confirma (aguardando retransmisión). TCP actúa sobre IP, que envía los datagramas sin orden. Cuando TCP recibe los datos, debe pasarlos en el orden correcto a la aplicación. TCP también debe descartar segmentos duplicados. También se provee **control de flujo** (ya que cada host tiene un buffer finito). *Esto previene que un host más rápido sature a otro más lento*.
- Servicio de **stream**: Un extremo puede leer los datos en la secuencias de bytes que crea conveniente. Si el emisor escribió primero 10 bytes, luego 20 bytes y finalmente 50 bytes, el receptor puede leer esos 80 bytes en bloques de a 20 bytes. TCP no tiene idea si los datos intercambiados son ASCII, binarios, etc. La interpretación de este stream de bytes la dan las aplicaciones de capa superior.

Encabezado TCP

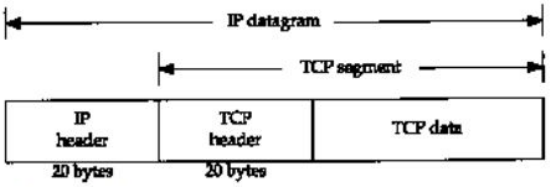


Figure 17.1 Encapsulation of TCP data in an IP datagram.

Figure 17.2 shows the format of the TCP header. Its normal size is 20 bytes, unless options are present.

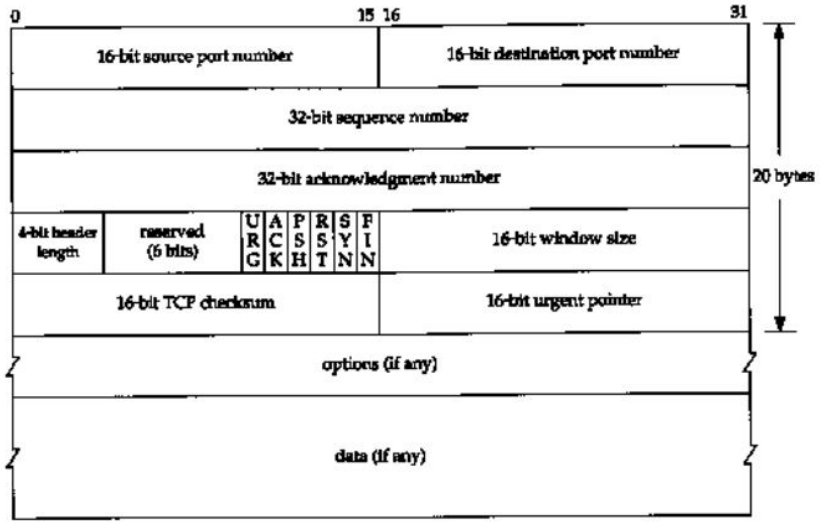


Figure 17.2 TCP header.

- Los **números de puerto** identifican las aplicaciones emisor/receptor. La combinación de una dirección IP y su correspondiente puerto se denomina **socket**.
- El **número de secuencia** identifica el número de byte del stream. Como es de 32 bits, al llegar a 2^32-1, se establece (nuevamente) en 0.
    - Cuando se envía un paquete **SYN**, este contiene un **ISN** (initial sequence number), elegido por el host. El número de secuencia envíado en un SYN, debe ser ISN+1 (el flag SYN consume un número de secuencia). La flag **FIN** también consume un número de secuencia.
- El **número de confirmación**, contiene el próximo número que el emisor de la confirmación espera recibir. Esto es número de secuencia + 1. Este campo es válido solo si la flag **ACK** se encuentra encendida.
    - El envío de un ACK no incrementa el nro. de secuencia.
    - En la mayoría de implementaciones, una vez que la conexión está establecida, por más que no se esté confirmando un segmento o por más que se envíe un ACK, los números de secuencia y confirmación se envían igualmente (con nro. de secuencia y confirmación del segmento n-1). Otra cuestión a denotar, es que la flag ACK siempre está encendida (lo cual permite confirmar paquetes sin necesidad de enviar un segmento con tan solo un ACK).
- Como TCP es **full-duplex**, cada extremo debe mantener sus números de secuencia y confirmación. TCP es descrito como un **protocolo de ventanas deslizantes sin ACK selectivos o negativos**. TCP (en principio) carece de reconocimiento selectivo, ya que al recibir un número de confirmación “x” entiende que se han recibido todos los segmentos hasta ese número. Por otra parte, el envío de confirmaciones duplicadas puede ayudar a dar cuenta al receptor que hay paquetes perdidos.
- La **longitud de cabecera** otorga un tamaño de palabras de 32 bits. Como es de 4 bits (1+2+4+8), el tamaño máximo es (15*32)/8=60 bytes. Si no se especifican opciones, el tamaño normal es de 20 bytes.

- Es posible seleccionar 6 flags para TCP. Más de una puede estar activa al mismo tiempo:
    - **URG**: Indica que **urgent pointer** es válido.
    - **ACK**: Indica que **número de confirmación** es válido.
    - **PSH**: Indica al receptor que debe **pasar los datos a la aplicación lo más rápido posible**.
    - **RST**: **Resetea** la conexión.
    - **SYN**: **Sincroniza números de secuencia** para iniciar conexión.
    - **FIN**: Indica **fin** que el emisor ya no enviará más datos.
- El **tamaño de ventana** (para **control de flujo**) se anuncia por cada extremo, y es el número de bytes, comenzando por el especificado en el **número de acuse de recibo** que el receptor está dispuesto a aceptar. Es un campo de 16 bits, por lo que se limita a 65535 bytes. En las opciones, se pueden seleccionar otros escalas a la ventana, para proporcionar ventanas más grandes.
- La **suma de comprobación** cubre tanto la cabecera como los datos TCP. Es calculada y almacenada por el emisor, y verificada en el receptor.
- El **puntero urgente** es válido si la la flag **URG** está encendida. Es la cantidad de bytes desde el número de secuencia, hasta la cual los datos son “urgentes”.
- La sección de **datos** es opcional. Un header sin datos, es un ACK.
- **Relleno**: Se utiliza para asegurarse que la cabecera acaba con un tamaño múltiplo de 32 bits.

Opciones: MSS

El campo **MSS** (Maximum segment size), de la sección opciones, denota el tamaño máximo de segmento que cada emisor espera recibir. Cada uno de los extremos suele especificar esta opción en el primer segmento intercambiado (con el SYN). Normalmente se utiliza el tamaño más pequeño. Es util para evitar fragmentación y suele corresponderse con el MTU más pequeño. **Problemas con el MSS**: Si es muy pequeño la gran proporción entre las cabeceras y los datos hará que se produzca un uso ineficiente del ancho de banda. Si es demasiado grande y la MTU es pequeña, se producirá fragmentación. Como un fragmento no se puede confirmar o retransmitir de forma independiente, todos los fragmentos deberán llegar correctamente o se tendrá que retransmitir todo el datagrama. Teóricamente, el tamaño óptimo de segmento ocurre cuando los datagramas IP llevan segmentos lo más grande posibles sin que haya necesidad de fragmentarlos. Un MSS grande indica que se pueden envíar más datos por segmento, amortizando el costo de cabeceras IP y TCP. Normalmente se intenta que el MSS + las cabeceras IP y TCP, no superen el tamaño máximo de trama. Para Ethernet esto implica 1460 bytes. En 802.3 el MSS puede ser hasta 1452 bytes (Coaxial 10BASE5). Si los extremos están en una red local, puede anunciarse un tamaño de 1460 bytes. Esto es deducible (en cierta medida) por los prefijos de red de emisor/receptor. En redes de hace algunos años atrás, si el destino era “no local”, el MSS podía descender a 536.

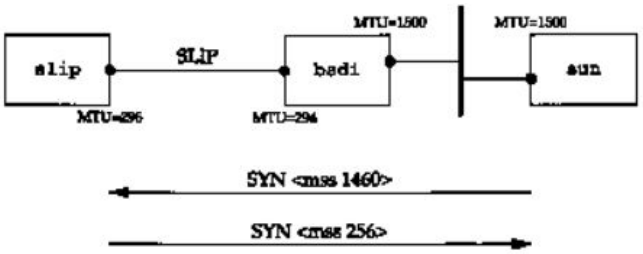


Figure 18.8 TCP connection from `sun` to `slip` showing MSS values.

Suponiendo la red presentada, SLIP debería anunciar un MSS de 256 y SUN un MSS de 1460. Sin embargo, SUN se "ajustará" a SLIP y no enviará segmentos más grandes que 296. Esto funciona bien, si en el medio no hay enlaces de menor tamaño, donde habrá fragmentación. En ciertos, casos, la única forma de evitar esto es usando el mecanismo de descubrimiento de MTU.

Otras opciones de cabecera

- **Escalado de ventana**: El campo "ventana" de la cabecera TCP tiene longitud 16 bits (64 Kbytes máximo), tamaño suficiente para las primeras redes, pero, no para las actuales. Para superar esta limitación, se propuso dicha opción que consiste en **tres octetos**: un tipo, una longitud, y un valor de desplazamiento. Esencialmente el valor de desplazamiento especifica un factor de escalado que se le aplica al valor de la ventana. La opción puede ser negociada durante el Tree-way-handshake o especificada para cada segmento. En este último caso, el factor de escalado de la ventana varía de un segmento a otro.
- **Acuse de recibo selectivos**: Esta opción debe ser implementada por ambos extremos, lo que se comprueba en la fase de establecimiento de la conexión
- **Timestamp**: Útil computar el tiempo que tarda en viajar un paquete entre los extremos de la conexión. Para TCP, el tiempo medio que tarde un paquete en llegar al otro extremo determinará cuánto tendrá que esperar antes de una retransmisión.

CAPÍTULO 18. TCP: INICIO Y FIN DE CONEXIÓN

El inicio de conexión es necesario antes de la transferencia de los datos. Normalmente se establece mediante un protocolo sin conexión, como lo es UDP. Una conexión TCP se identifica por una cuadrupla <IP local: puerto local>,<IP destino: puerto destino>

```
1  0.0                 svr4.1037 > bsdi.discard: S
                       1415531521:1415531521(0)
                       win 4096 <mss 1024>

2  0.002402            bsdi.discard > svr4.1037: S
   (0.0024)            1823083521:1823083521(0)
                       ack 1415531522 win 4096 <mss 1024>
3  0.007224            svr4.1037 > bsdi.discard: ack 1823083522
   (0.0048)            win 4096
4  4.155441            svr4.1037 > bsdi.discard: F
   (4.1482)            1415531522:1415531522(0)
                       ack 1823083522 win 4096
5  4.156747            bsdi.discard > svr4.1037: . ack 1415531523
   (0.0013)            win 4096
```

```
6   4.158144 (0.0014)    bsdi.discard > svr4.1037: F 1823083522:1823083522(0) ack 1415531523 win 4096
7   4.180662 (0.0225)    svr4.1037 > bsdi.discard: . ack 1823083523 win 4096
```

Figure 18.1 `tcpdump` output for TCP connection establishment and termination.

Referencia: source>destination: flags.

Análisis: En la línea 1, el campo 1415531521:1415531521 (0), especifica (antes de los dos puntos) el número de secuencia inicial y “final” (luego de los dos puntos), que confirmará el receptor, y el número de bytes de datos que es 0 (en este caso el receptor confirmará nro de secuencia + 1). En la línea 2, 1415531522, muestra el número de confirmación, que únicamente se muestra si el flag ACK es verdadero. El campo win 4096 en todas las líneas, muestra el tamaño de ventana anunciado por el emisor. En estos ejemplos, donde no se intercambian datos, se mantiene en 4096 (valor por defecto). El campo final <mss 1024> muestra la opción MSS especificada por el emisor. El emisor no recibirá segmentos TCP más grandes que este.

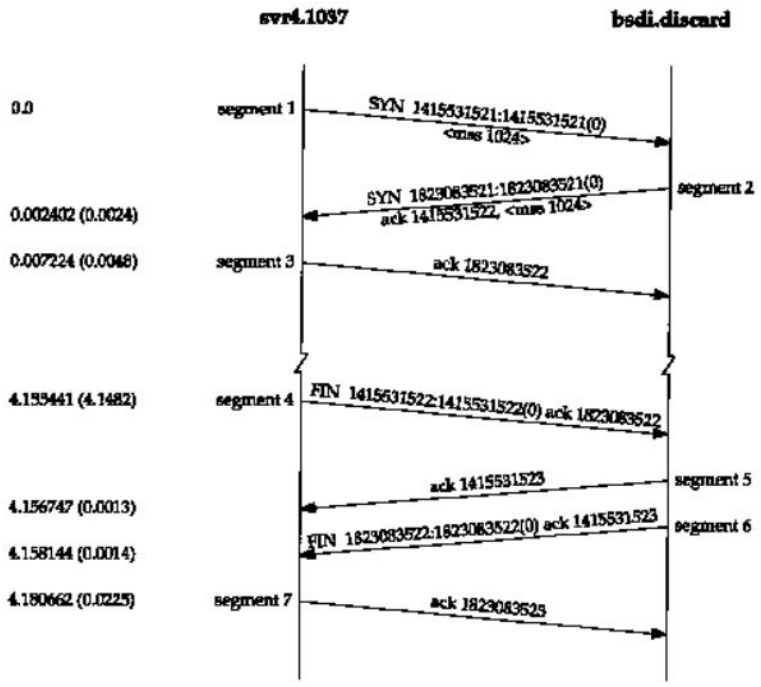


Figure 18.3 Time line of connection establishment and connection termination.

Análisis del gráfico

- **Inicio de conexión**: En los primeros tres segmentos se observa el Three-Way Handshake. El lado que envía el SYN, ejecuta una **apertura activa**, mientras el lado que recibe el SYN realiza una **apertura pasiva**. El MSS de cada lado se selecciona durante el SYN. El número de secuencia intercambiado al principio constituye el ISN (nro. inicial), que es un contador (general) de 32 bits incrementado cada 4 microsegundos con el fin de prevenir que los paquetes que quedaron retrasados en la red interfieran con una nueva conexión (el número de secuencia rota cada 4.55

horas). Cuando TCP inicia una nueva conexión, utiliza, como inicial, el número de secuencia inicial calculado. Las implementaciones pueden cambiar.

- **Fin de conexión**: Se utilizan 4 segmentos. En principio, la conexión se cierra por la mitad (primeros dos segmentos). Como la conexión es full-duplex, cada uno deberá aplicar el **protocolo de fin**. La flag FIN sólo cierra la conexión en un extremo (half-close), por lo que el otro podrá seguir enviando datos. De todas formas, a efectos prácticos, la conexión se finaliza cuando uno de los extremos ejecuta el **FIN activo** (el extremo que se ve "obligado" a cerrar la conexión, ejecuta un FIN pasivo). Cada uno de estos segmentos aumenta en uno el número de secuencia.

Timeout en establecimiento de conexión: Luego de una serie de reintentos de envío de SYN, la conexión se establecerá como fuera de tiempo. Esto indicará que el servidor es inaccesible (que está caído, por ejemplo). Dependiendo de la implementación, variará el ISN e intervalo en cada intento

TCP: Cierre en una sola dirección (Half-Close)

Se da cuando **sólo uno de los dos extremos cierra la conexión**, mientras que el otro la deja abierta (en dirección al extremo que originalmente envió la flag FIN). El envío de la flag FIN indica que el extremo no enviará más datos, pero que aún puede recibir. *Nota: Normalmente en las aplicaciones de socket, se termina la conexión en ambas direcciones mediante la llamada a "close()".*

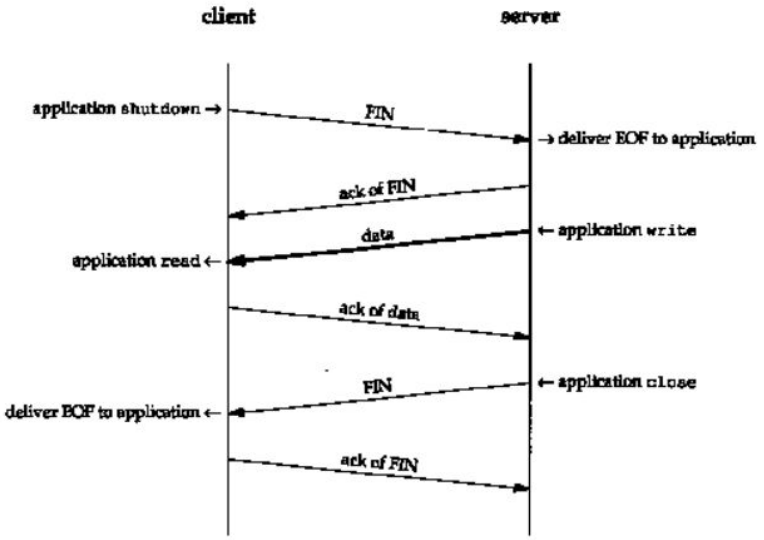
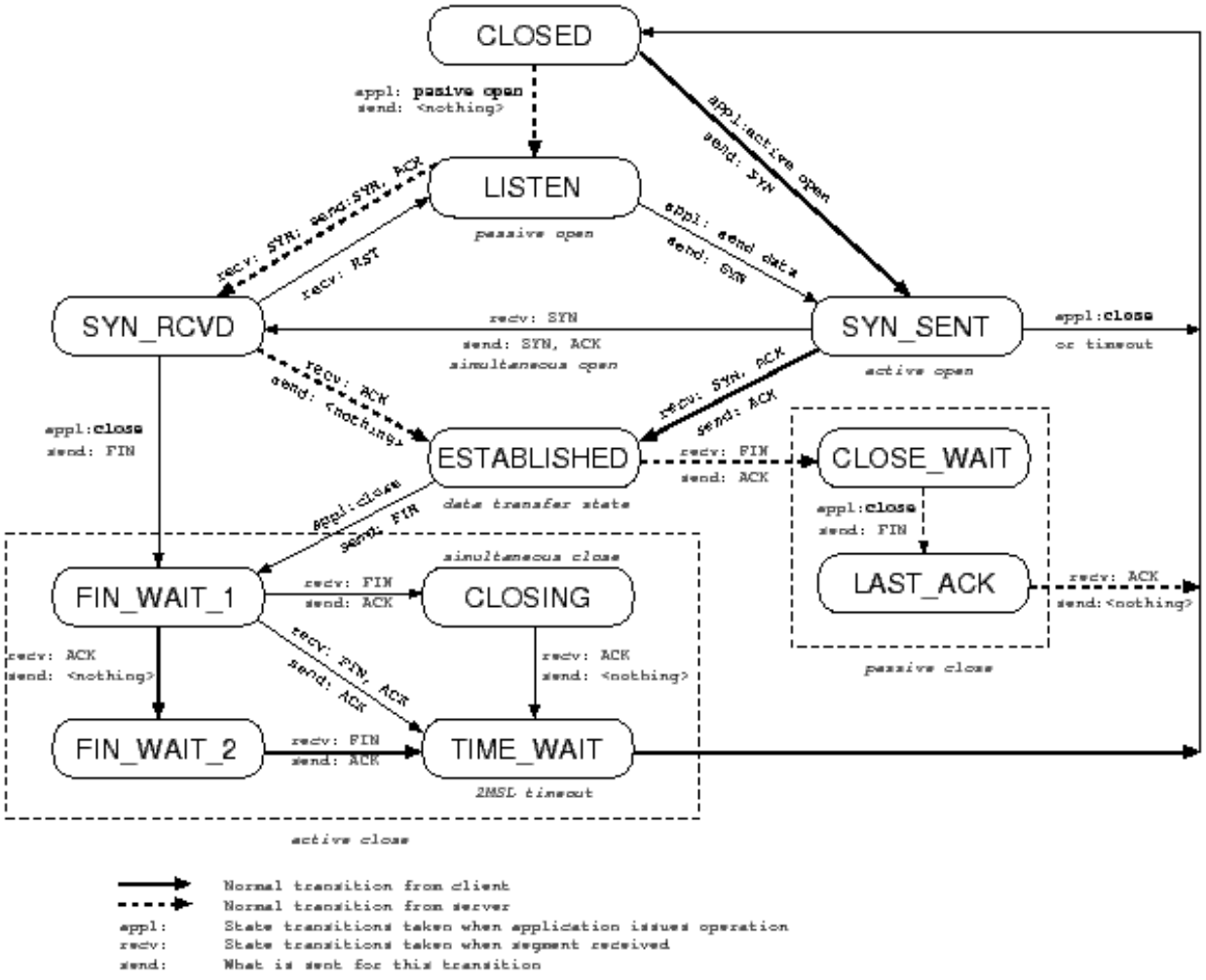


Figure 18.10 Example of TCP's half-close.

Diagrama de transición de estados



En total existen 11 estados, y son idénticos a los mostrados en la ejecución del comando **netstat**. **CLOSED** realmente no es un estado, pero es un punto de partida y finalización imaginario. Un servidor puede estar “esperando” que los clientes se conecten a él. La espera corresponde a un estado de “**LISTEN**”. En realidad se dice que el servidor está “escuchando” en un puerto. Cuando un **servidor** se ejecuta pasa de estado “CLOSED” a estado “LISTEN” esperando que le lleguen segmentos TCP con el flag SYN activado (apertura pasiva). Sin embargo, cuando se ejecuta un **cliente**, éste iniciará directamente la conexión con el servidor enviándole un segmento SYN (apertura activa).

Notas:

- La transición de estado de LISTEN a SYN_SENT es legal pero no está soportada en algunas implementaciones (Berkeley, por ejemplo).
- La transición de SYN_RCVD a LISTEN es válida solo si anteriormente se ha ingresado a a LISTEN desde SYN_RECV, y no desde SYN_SENT.

Descripción de estados:

- **CLOSED** : No hay conexión activa ni pendiente.
- **LISTEN**: El servidor espera una una apertura.

- **SYN RCVD**: Llegó una solicitud de conexión; espera ACK.
- **SYN SENT**: La aplicación comenzó una conexión.
- **ESTABLISHED**: Estado normal de transferencia de datos.
- **FIN WAIT 1**: La aplicación inicia fin de conexión.
- **FIN WAIT 2**: La contraparte está de acuerdo en finalizar.
- **TIMED WAIT**: Espera a que todos los paquetes mueran.
- **CLOSING**: Ambos lados intentaron cerrar simultáneamente.
- **CLOSE WAIT**: El otro lado inició un cierre.
- **LAST ACK**: Espera a que se reciban los últimos paquetes y el ACK de FIN.

Ejemplo gráfico de apertura y cierre

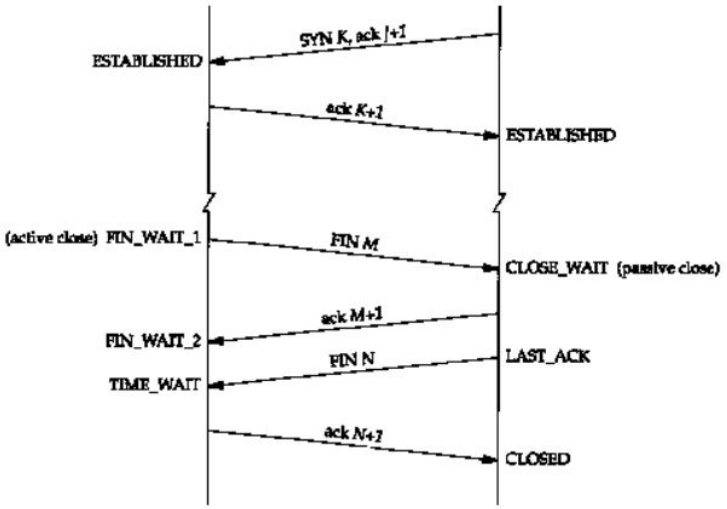


Figure 18.13 TCP states corresponding to normal connection establishment and termination.

Nota: En el presente gráfico se asume que el cliente ha realizado una apertura activa.

2MSL Wait State

El estado **TIME_WAIT**, también llamado **2MSL** (Maximum segment lifetime), es la cantidad máxima de tiempo de vida de cualquier segmento antes de ser descartado y está limitado por el **TTL** de IP. El RFC 793 (Postel) especifica el MSL en dos minutos, pero puede ser de 30 segundos o un minuto. Esto temporizador asegura que una nueva conexión, no reciba segmentos destinados a una anterior. Tener en cuenta, que además, para paliar esto, los números de secuencia van rotando.

La regla es: Cuando TCP inicia un cierre activo y envía el ACK final, la conexión estará en **TIME_WAIT dos veces el valor del MSL**. Mientras el temporizador esté activo, el **par de sockets** (emisor y destino), **no puede ser reutilizado**. Las implementaciones de Berkeley, además, son más estrictas, evitando re-utilizar ese puerto efímero en una futura conexión. Algunas implementaciones de APIs, permiten reutilizar el socket durante el tiempo del 2MSL Timer, mediante la opción SO_REUSEADDR (encarnación de conexión) que puede reutilizar una nueva instancia de la conexión. De todas formas, TCP puede no permitir que el número de puerto sea utilizado para una conexión en espera. **Todos los segmentos recibidos durante el 2MSL se descartan.**

Es normal que un **cliente** realice el cierre activo y entre en estado **TIME_WAIT**. El servidor, en este caso, no pasa por este estado (normalmente realiza el cierre pasivo). Si el servidor realiza un cierre activo al cliente, y este último se reinicia de inmediato, la nueva instancia de conexión del cliente no podrá volver a utilizar el mismo número de puerto local: *Sin embargo este no es un problema, ya que los clientes suelen utilizar puertos efímeros*, contrario a los servidores, que usan puertos bien conocidos. De este modo, a causa del 2MSL Wait State, el servidor estaría imposibilitado de utilizar dicho puerto de 1 a 4 minutos (2*MSL), por encontrarse en estado estado TIME_WAIT.

Nota: *Si el servidor es quien realiza el cierre activo, al cliente se le permite reutilizar al puerto. Esto se considera una violación de protocolo, presente en la mayor parte de implementaciones de Berkeley.*

Tiempo de silencio

Si los host se crashean, la espera 2MSL no surte efecto. En este caso, no se habrían retenido conocimiento de los últimos números de secuencia de cada conexión activa. En este escenario, las nuevas conexiones, sí podrían confundirse con anteriores. Por ello, el RFC 793 (TCP), establece que se debería retrasar la emisión de cualquier segmento TCP de una nueva conexión que utilice tuplas de IP y puertos existentes antes del cuelgue, al menos durante al menos el MSL. Sin embargo, la mayoría de implementaciones tienden a violar esta restricción, asumiendo el riesgo.

Estado FIN WAIT 2

En este estado, el host ha enviado el FIN, que ha sido reconocido por el otro extremo, por lo que también se encuentra a la espera de un FIN. Cuando se recibe este último FIN, se pasa al estado TIME_WAIT. Sin embargo, cuando este extremo se encuentra en estado FIN WAIT 2, el otro extremo (quien debe realizar el cierre pasivo), se encuentra en estado CLOSE_WAIT (y podría permanecer así, hasta que la aplicación decida envíar el FIN, por lo la conexión podría quedar “entre-abierta” en una espera infinita). En algunas implementaciones se evita esta “espera infinita”, haciendo que que quien inició el fin de conexión pase del estado FIN_WAIT_2 a CLOSED, siempre y cuando no se reciban datos en un tiempo de 10 minutos 75 segundos (lo cual, de todas formas, es una violación de protocolo).

Segmentos de reinicio

En general el BIT de **RST** (reset) se envía cuando se recibe un segmento que no es propio de la conexión actual (sino que de una anterior). Las causas de generar un paquete con este bit pueden ser varias: Intento de conexión a un puerto no existente (en UDP se utiliza ICMP), aborto de una conexión, respuesta ante conexiones semi-abiertas.

```
                    bsdi.1087 > svr4.20000: S
1   0.0             297416193:297416193(0)
                    win 4096 <mss 1024> [tos 0x10]
2   0.003771        svr4.20000 > bsdi.1087: R 0:0(0)
    (0.0038)        ack 297416194 win 0
```

Figure 18.14 Reset generated by attempt to open connection to nonexistent port.

Los números de secuencia del receptor, se establecen en 0, y el ACK e en ISN+1.

Abortando una conexión: Protocolarmente, la conexión debe finalizarse con la flag FIN, pero también puede utilizarse RST ("**liberación abortiva**"),que permite que los datos en la **cola de la aplicación se desechen** inmediatamente. La API utilizada por la aplicación, debiera tener una alternativa para generar un abort(), en vez de un close().

Ejemplo de conexión (-L especifica que la conexión deberá cerrarse con RST):

```
bsdi % sock -LO svr4 8888   this is the client; server shown later
hello, world                type one line of input that's sent to other end
^D                          type end-of-file character to terminate client
```

1	0.0	bsdi.1099 > svr4.8888: S 671112193:671112193(0) <mss 1024>
2	0.004975 (0.0050)	svr4.8888 > bsdi.1099; S 3224959489:3224959489(0) ack 671112194 <mss 1024>
3	0.006656 (0.0017)	bsdi.1099 > svr4.8888: . ack 1
4	4.833073 (4.8264)	bsdi.1099 > svr4.8888: P 1:14(13) ack 1
5	5.026224 (0.1932)	svr4.8888 > bsdi.1099: . ack 14
6	9.527634 (4.5014)	bsdi.1099 > svr4.8888: R 14:14(0) ack 1

Figure 18.15 Aborting a connection with a reset (RST) instead of a FIN.

El segmento RST provoca el cierre, sin recibir respuesta alguna desde el otro lado. Cuando el receptor recibe el RST, se puede observar el siguiente mensaje de error:

```
svr4 % sock -s 8888          run as server, listen on-port 8888
hello, world                 this is what the client sent over
read error: Connection reset by peer
```

Detección de conexiones abiertas en una sola dirección (half-open connections)
Una conexión entreabierta, ocurre cuando un extremo ha tenido un cuelgue inesperado (o similar), sin que el otro se haya enterado. Por ejemplo, si un cliente Telnet se apaga sin cerrar la conexión, el servidor podría dejar su parte de la conexión abierta (esperando que el cliente transmita datos). Si el cliente volviera a iniciar la conexión en otro puerto, la conexión anterior, del lado del servidor, podría seguir "viva". Otro ejemplo, es si un servidor se reiniciara mientras un cliente Telnet no transmite datos. Suponiendo que el servidor se reinicia antes de que el cliente tenga que volver a transmitir y, posteriormente, el cliente realiza una transmisión, el servidor, enviará un segmento con la flag de RST activa.

Apertura simultánea
Aunque es muy poco probable, es posible que dos extremos realicen una apertura pasiva en el mismo instante de tiempo.

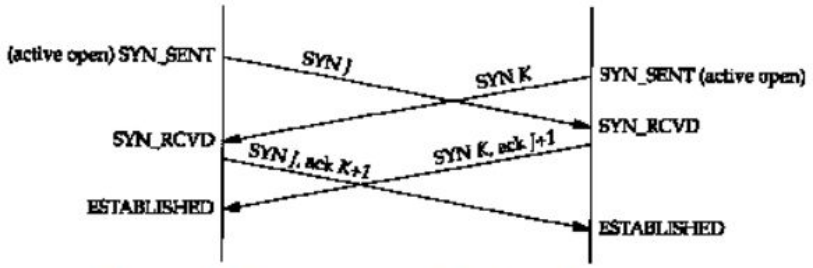


Figure 18.17 Segments exchanged during simultaneous open.

Afortunadamente el diseño de TCP permite que se pueden manejar correctamente las aperturas simultáneas. Esto no resultaría un problema, más allá de los SYN innecesarios**.** Este tipo de apertura requiere el intercambio de 4 segmentos (uno más que en el Three-way handshake). La salvedad es que, en este caso, ambos extremos actuarían, tanto como cliente y servidor. **Nota: En algunas implementaciones esto podría NO funcionar, y se podría terminar intercambiando un infinito número de segmentos SYN-ACK.**

Cierre simultáneo:

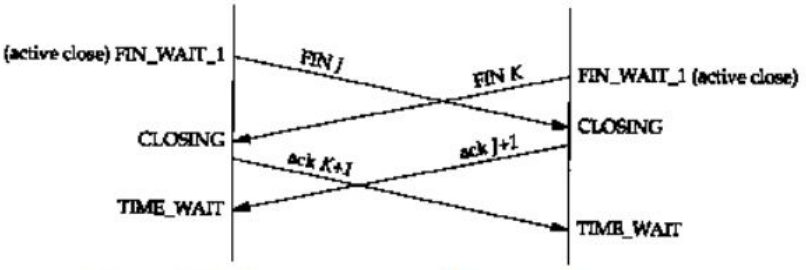
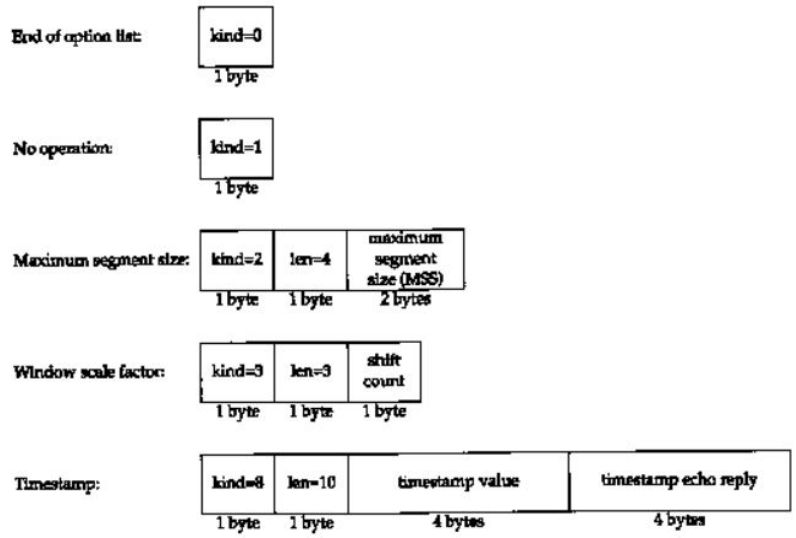


Figure 18.19 Segments exchanged during simultaneous close.

Durante un cierre simultáneo, ambos extremos pasan al estado de TIME_WAIT. En este caso, a diferencia de la apertura simultánea, la cantidad de segmentos intercambiados es la misma que en un cierre común. La única diferencia es que ambos extremos pasan por el estado CLOSING (dado únicamente en estos casos de cierre simultáneo), en espera de la confirmación correspondiente.

Opciones TCP:
Las opciones originales definidas son “Fin de opciones”, “No operation” y “MSS” (presente en casi todos los segmentos SYN. Los nuevos RFC, definen, otras opciones más:



Todas las opciones comienzan con un byte de tipo (kind), que especifica el tipo de opción, y ocupa un byte. La opción NOP permite un pad, para que los campos sean múltiplo de 4. Por ejemplo, durante un segmento SYN, las opciones intercambiadas podrían ser las siguientes: *<mss 512,nop,wscale 0,nop,nop,timestamp 146647 0>: en este caso, la razón del primer "nop", es para paddear los 3 bytes de wscale, luego los dos posteriores, se utilizan para el pad del timestamp (1+1+4+4=10+2=12, que es múltiplo de 4)*. Los tipos 4, 5, 6 y 7 son opciones de ACK selectivo (aún en discusión) y eco, este último reemplazado con la opción timestamp.

Diseño de servidor TCP

La mayoría de servidores TCP son concurrentes: Cuando existe una nueva conexión, el servidor acepta la conexión e invoca un nuevo proceso para manejar al cliente.

Comando netstat: Muestra las conexiones activas. La bandera -a especifica que se deben mostrar todas las conexiones, no solo las establecidas. -n imprime las IP en decimal punteado, en lugar de tratar de usar DNS para convertirlas en nombre, y además imprime los números de puerto (ej, 23), en lugar de los nombres de servicio (ej. Telnet). La opción -f inet solo muestra los endpoints TCP y UDP.

```
sun % netstat -a -n -f inet
Active Internet connections (including servers)
Proto  Recv-Q  Send-Q   Local Address   Foreign Address (state)

tcp    0       0        *.23            *.*             LISTEN
```

La dirección local se muestra como *.23, donde el asterisco se denomina **carácter comodín** (wildcard character). Esto significa que si llega una solicitud de conexión, será aceptada por cualquier interfaz local. Si el host es multihomed, podría especificarse una dirección IP de las asignadas a cada interfaz.

```
Proto Recv-Q Send-Q Local Address    Foreign Address    (state)
tcp   0      0      140.252.13.33.23 140.252.13.65.1029 ESTABLISHED
tcp   0      0      *.23             *.*                LISTEN
```

La IP presente en local address, corresponde a la interfaz en la que llegó la solicitud de conexión. La segunda conexión en estado LISTEN, indica que el puerto 23 se encuentra a la escucha de nuevas conexiones. Si llegara una nueva solicitud de conexión de un mismo cliente, TCP las trataría como dos distintas, al estar en puertos separados (puertos efímeros para el cliente). Este es un ejemplo de la **multiplexación TCP**.

```
Proto Recv-Q Send-Q Local Address    Foreign Address    (state)
tcp   0      0      140.252.13.33.23 140.252.13.65.1030 ESTABLISHED
tcp   0      0      140.252.13.33.23 140.252.13.65.1029 ESTABLISHED
```

Por otra parte, los endpoints en estado ESTABLISHED no pueden recibir segmentos SYN, y los endpoints en estado LISTEN, no pueden recibir segmentos de datos.

Ejemplo en multihomed (observar las diferencias en local address):

```
Proto Recv-Q Send-Q Local Address    Foreign Address    (state)
tcp   0      0      140.252.1.29.23  140.252.1.32.34603 ESTABLISHED
tcp   0      0      140.252.13.33.23 140.252.13.65.1030 ESTABLISHED
tcp   0      0      140.252.13.33.23 140.252.13.65.1029 ESTABLISHED
tcp   0      0      *.23             *.*                LISTEN
```

Restricción de direcciones IP locales: Cuando el servidor no utiliza un wildcard, y especifica una dirección IP (o nombre de host), se restringe la escucha a únicamente esa IP, descartando conexiones a otras interfaces (la aplicación nunca se entera de esto, ya que es tarea del núcleo TCP).

Restricción de direcciones extranjeras: En TCP (y UDP) es posible, además de seleccionar un IP y puerto locales, un IP y puerto “externo” el cual será el único del que se aceptarán datos. Desafortunadamente muchas de las API, no proporcionan una manera de hacer esto. El servidor debería implementar dicha funcionalidad en capa de aplicación.

En resumen:

Dirección local	**Dirección externa**	**Descripción**
IP local.Puerto l.	IP ext. Puerto externo	Restringido a un cliente (normalmente no soportado)
IP local.Puerto l.	*.*	Restringido a conexiones en la interfaz de “IP local”
*.Puerto local	*.*	Recibe todas las conexiones a “Puerto local”

Cola de solicitudes de conexiones entrantes

¿Cómo maneja TCP las conexiones entrantes si la aplicación TCP está ocupada?

1. TCP puede aceptar muchas conexiones (mediante Three-way handshake), sin que hayan sido "aceptadas" por la aplicación (este número no tiene nada que ver con el número de conexiones máximas establecidas, o concurrentes).
2. La aplicación puede especificar el límite de esta cola, comúnmente llamada **backlog** (0-5). Muchas aplicaciones especifican el máximo valor de 5.
3. Si hay espacio en la cola, el módulo TCP (o mediante API de capa de aplicación) puede confirmar el SYN y completar la conexión. La aplicación servidor no ve esta conexión hasta que se recibe el tercer segmento de un three-way handshake. De modo que el cliente, pensará que el servidor está listo para recibir datos: si esto pasa, el servidor TCP encolará los datos entrantes.
4. Sin espacio en la cola para una nueva conexión, **TCP ignorará el SYN recibido.** Ni siquiera enviará el segmento de RST. Eventualmente, el cliente entrará en time out. Ignorando el SYN, el cliente entienda que (quizás) luego se podrá conectar.

CAPÍTULO 19: TCP - FLUJO DE DATOS INTERACTIVO

El tráfico de TCP puede ser **masivo** (en bloques o Bulk) para el caso de FTP y Mail o **interactivo**, para el caso de Telnet y Rlogin. Los datos en una transferencia masiva, intentan cubrir el MSS, mientras que en el trafico interativo pueden existir paquetes de menos de 10 bytes.

Entrada interactiva: Rlogin genera un paquete por cada tecla pulsada. Además, genera otros tres paquetes más: el ack y eco del servidor, y finalmente el ack del cliente, a diferencia de Telnet (que envía caracteres por línea, lo cual reduce la carga de la red).

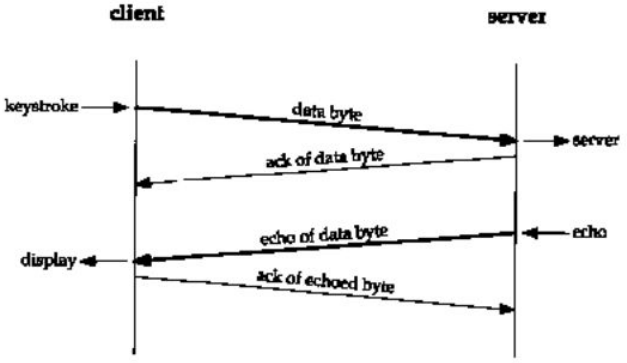


Figure 19.1 One possible way to do remote echo of interactive keystroke.

Normalmente, de igual modo, los segmentos 2 y 3 se combinan en uno solo. Esto se denomina **retraso de la confirmación**: Habitualmente TCP no envía los ACK en cuanto se reciben, sino que los retrasa, **esperando datos** que vayan en la misma dirección del ACK. Esto se denomina ACK sobre datos (**piggyback**).

La mayoría de implementaciones utiliza de 1 a 200 ms de delay, siendo posible utilizar hasta 500 ms. **El delay trabaja de la siguiente manera**: Si el timer está en 100 ms (u otro valor) y un ACK llega para ser enviado, el retraso sólo será de 100 ms respecto a los 200 ms originales (200 mls-100 mls). El **ACK se enviará en caso de expiración del timer o tan**

pronto como haya datos. Esto es ideal en RLogin, donde el servidor, además de enviar un ACK, debe enviar un echo.

Algoritmo de Nagle

Por más que se transmita 1 byte de datos, se necesitan en total 41 bytes para el paquete: 1 byte de datos, 20 bytes para el header IP y otros 20 bytes para el header TCP. Estos pequeños paquetes (**tinygrams**), no suelen ser problema en las LANs, pero si en las WANs, donde pueden causar **congestión**. Una solución a esto, es el **Algoritmo de Nagle**, que sostiene que una conexión TCP puede tener **únicamente un tinygram no reconocido**. Este algoritmo es **autosincronizado**: TCP puede recoger otras pequeñas cantidades de datos, durante la espera del ACK. Esto produce que se envíen menos segmentos (con un tamaño máximo del MSS). Este algoritmo no tiene trascendencia en las LANs, donde los tiempos de respuesta son sumamente rápidos. Esto cambia cuando el RTT aumenta (en WAN o redes lentas).

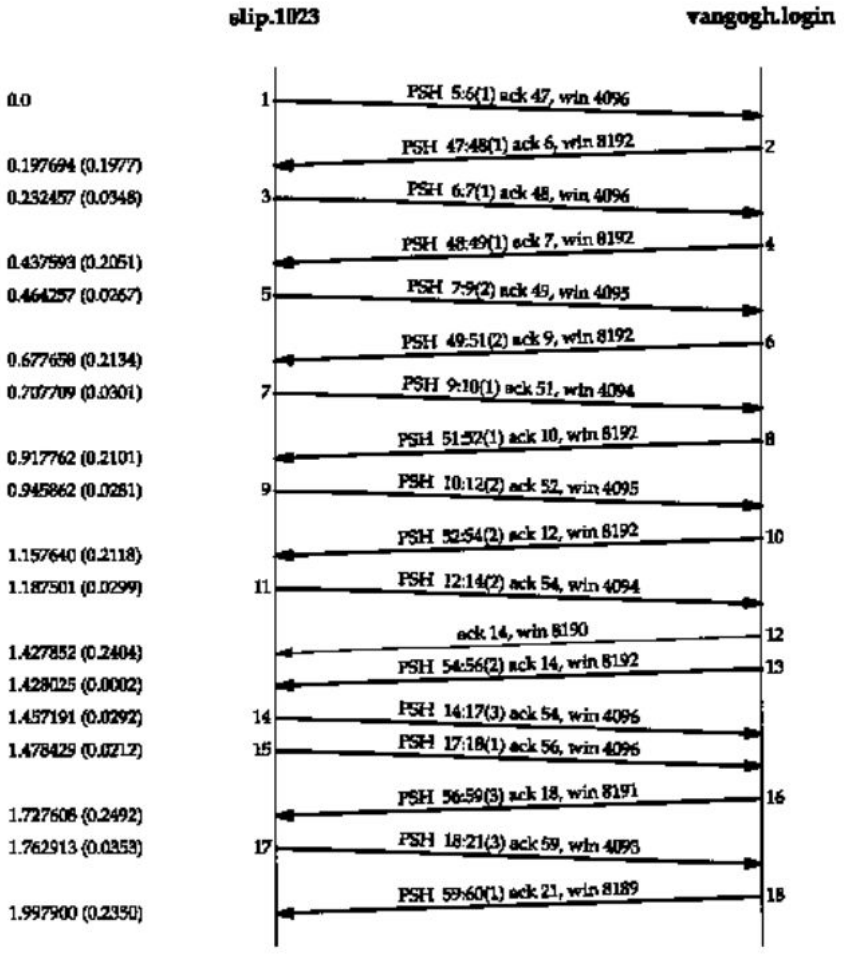


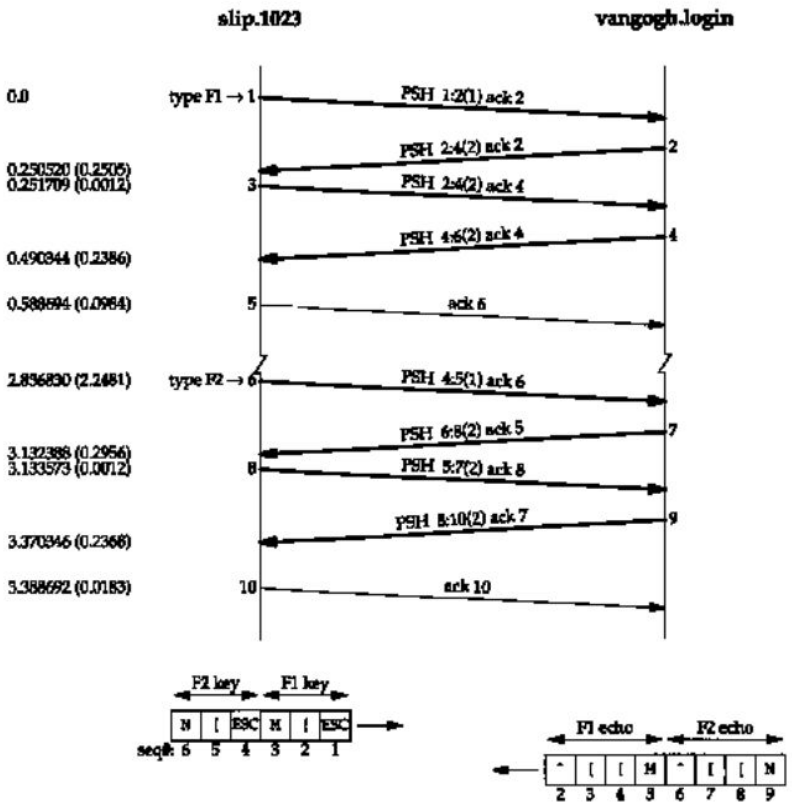
Figure 19.4 Dataflow using `rlogin` between `slip` and `vangogh.cs.berkeley.edu`.

Se puede observar el algoritmo de Nagle en los segmentos mayores a 1 byte (se envían 9 segmentos en total, en lugar de 16). Los segmentos 14 y 15 parecen contradecir el algoritmo de Nagle, sin embargo el segmento 14 es en respuesta al ACK recibido del segmento 12, y el segmento 15 contiene el ACK del segmento 13. El segmento 12 podría tratarse de un ACK retardado, envíado por expiración del temporizador en el servidor. En el

segmento 17 se tiene que TCP envía 3 bytes, recibe confirmación de los 3, pero sólo 1 byte de echo ¿Qué significa esto? Que **TCP puede confirmar los segmentos antes de que sean leídos por la aplicación**. Una afirmación de que no se han leído los datos es que la diferencia entre ventanas entre 16 y 18 es 8191-8189=2 bytes.

Deshabilitando el algoritmo de Nagle: Hay veces en la que es útil deshabilitar el algoritmo de Nagle. Por ejemplo en el escritorio remoto, pequeños mensajes (movimientos de mouse), deben ser entregados sin demora (en tiempo real), para proveer a los usuarios interactividad al realizar ciertas operaciones. Otro ejemplo está en que las teclas de función especiales (F1, F2, …) generan múltiples bytes de datos (que normalmente comienzan con el carácter ASCII escape). Si se envía un primer byte, y para el envío de los restantes se espera un ACK, el usuario interactivo tendrá notables retrasos. La API de sockets utiliza la opción **TCP_NODELAY** para deshabilitar el algoritmo de Nagle. El RFC de TCP establece que se debe implementar el algoritmo de Nagle, pero debe haber forma de deshabilitarlo en conexiones individuales.

Ejemplo con algoritmo de Nagle activo:



La razón de que el echo ocupa dos bytes (segmento 2) es porque el carácter ASCII es encodeado en 2 bytes: ^ y [.

Ejemplo con algoritmo de Nagle inactivo:

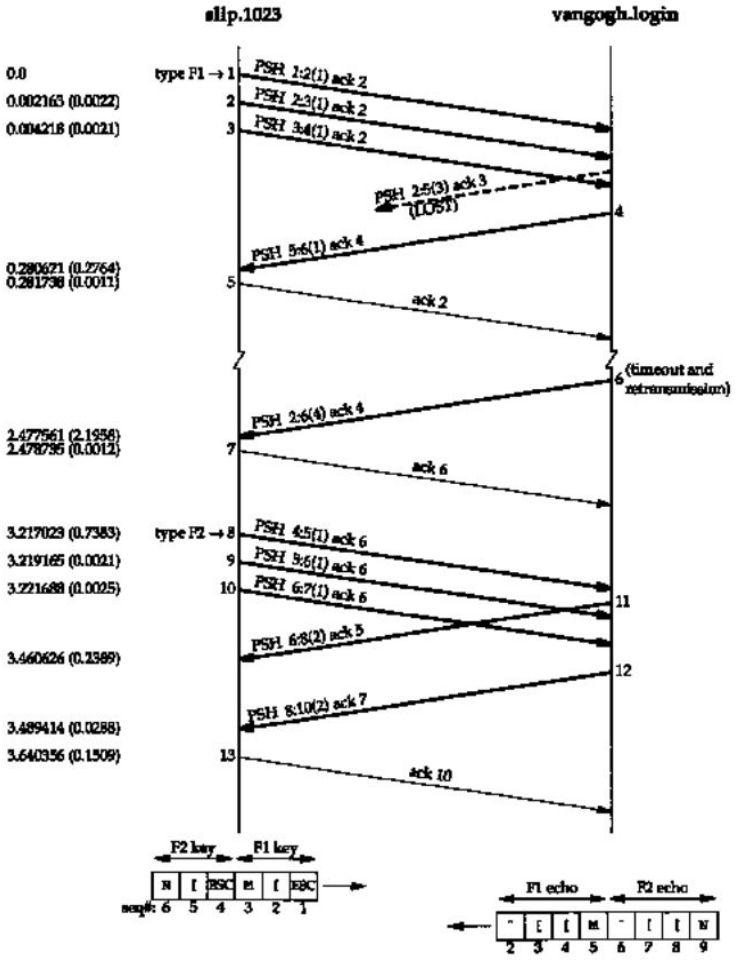


Figure 19.8 Time line for Figure 19.7 (Nagle algorithm disabled).

Con la deshabilitación del algoritmo de Nagle, **ya no hay delay**. Además puede verse que el ack 3 y los primeros 3 bytes de eco se pierden, a lo que el emisor da cuenta por la recepción del byte 5:6. Por ello el receptor del ECO responde con un ack 2, indicando que espera los bytes 2 en adelante a lo que el emisor del ECO, vuelve a enviar los cuatro bytes en un proceso denominado **re-empaquetado**. Por otra parte, el segmento 11, envía en dos bytes el eco del primer byte, y el 12 envía el eco de los segmentos 9 y 10.

Anuncios de tamaño de ventana

En la figura 19.4 **slip** anuncia una ventana de 4096 bytes y **vangogh** una de 8196 bytes. La mayoría de segmentos de la figura tienen uno de esos dos valores. En el segmento 5, sin embargo, se anuncia una ventana de 4095 bytes: esto significa que hay un byte en el buffer TCP (cliente Rlogin), que la aplicación deberá leer. El el siguiente segmento, se anuncia una ventana de 4094 bytes, lo que significa que aún quedan 2 bytes a leer. El servidor normalmente anuncia una ventana de 8192 bytes, ya que los datos se envían luego que el servidor lee los datos del cliente (ya que debe realizar un echo).

CAPÍTULO 20: TCP - FLUJO DE DATOS MASIVO

En el protocolo **S&W** no se envía un nuevo bloque de datos hasta recibir un ACK. En Sliding Windows, es posible transmitir más de un paquete antes de detener la transmisión, aún en la espera de un ACK. Esto permite una transferencia de datos más rápida ya que el emisor no debe parar y esperar por cada bloque de datos enviado.

Flag PUSH

La flag PUSH indica al receptor TCP que debe pasar estos datos (y los que se encuentren en el buffer, hasta la recepción de dicho segmento), **lo más rápido posible al proceso receptor** (aplicación), sin esperar datos adicionales. *Por ejemplo, se podría establecer la flag PSH cuando el cliente envía comandos. De todas formas, hoy en día, gran parte de implementaciones ya envían la flag de PSH activa. En otras implementaciones, se envía la flag PSH cuando se vacía el buffer de envío del emisor, mientras que en otras, la FLAG directamente se ignora ya que no retrasa la entrega de datos a la aplicación.*

Flujo de datos normal

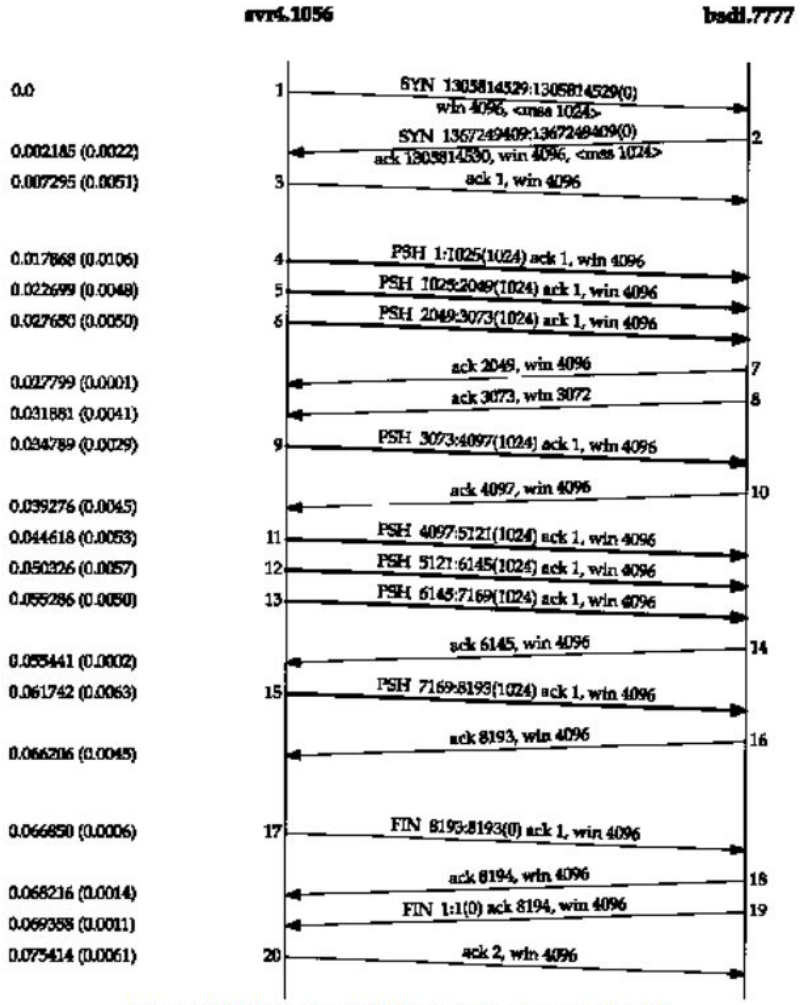


Figure 20.1 Transfer of 8192 bytes from `svr4` to `bsdi`.

El emisor primero transmite 3 segmentos de datos (4 a 6). El siguiente segmento (7) confirma los primeros dos segmentos únicamente (esto es porque el número de secuencia

confirmado es 2049 no 3073). Esto indica la expiración del temporizador de ACK retardado. En el segmento 8 se anuncia una ventana de 3072 bytes, lo cual indica que hay 1024 datos no leídos por la aplicación. En los segmentos se puede observar que cuando llega una conexión se marca como un ACK retardado. El segmento 14, por ejemplo, confirma los segmentos 11 y 12 y el 16 el 13 y el 15, ya que **los ACKs son acumulativos**. *En este caso, el cliente envía el indicador PUSH en los ocho segmentos. Esto se debe a que el cliente realiza ocho escrituras de 1024 bytes y en cada una vacía el buffer de envío.*

Emisor rápido, receptor lento

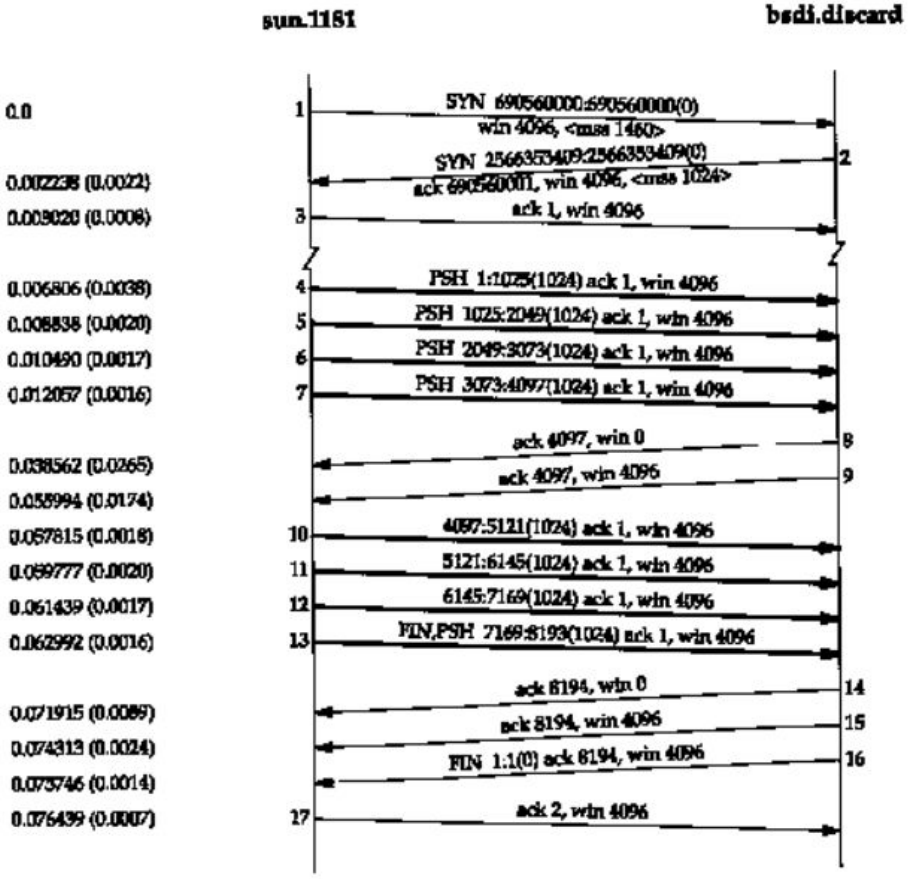


Figure 20.3 Sending 8192 bytes from a fast sender to a slow receiver.

En este caso, el emisor transmite 4 segmentos (4-7), uno tras otro, hasta llenar la ventana del receptor. El emisor, entonces, se detiene y espera un ACK. El receptor envía un ACK, pero la ventana anunciada es 0. Esto significa que se han recibido los datos, pero no hay más espacio en los buffers TCP. Otro ACK, denominado **actualización de ventana** es enviado 17.4 ms más tarde, anunciando al receptor que ahora puede recibir otros 4096 bytes. En el final segmento 14, se indica que se confirma hasta el byte 8192 de datos y la flag FIN (8193), por ello el ACK indica que se espera el número 8194. *En este caso el segmento PUSH se visualiza en los segmentos 4 a 7, porque inmediatamente se vacia el buffer del emisor. Para enviar los segmentos 10 a 13, TCP debió esperar a que se abra la ventana del receptor. Cuando esto sucede, se envían los 4 segmentos, pero sólo el del final tiene la flag PUSH activa.*

Ventanas deslizantes

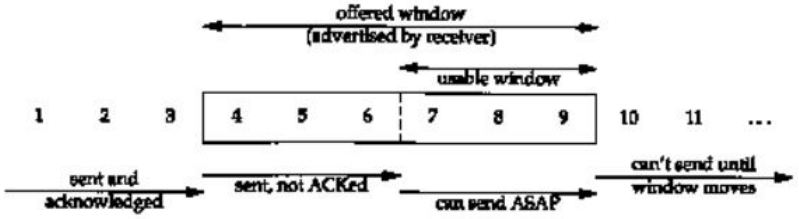


Figure 20.4 Visualization of TCP sliding window.

La ventana anunciada por el receptor se denomina **ventana ofrecida** y cubre los bytes 4 al 9. De esta forma, el emisor de datos calcula la **ventana usable** (en base a la ventana ofrecida), mediante los números de secuencia que el receptor aún no ha confirmado. Esta ventana indica el número de datos que el emisor puede enviar inmediatamente. A medida que el receptor reconoce datos, la ventana se mueve a la derecha. Tres términos se utilizan para describir el movimiento de la ventana:

1. La ventana se **cerrará** de izquierda a derecha, cuando el emisor reciba un ACK.
2. La ventana se **abrirá** cuando el borde derecho se mueve a la derecha, lo cual permite que más datos sean enviados. Esto sucede cuando el proceso del otro extremo lee los datos reconocidos, liberando espacio del buffer.
3. El receptor puede **encoger** el borde derecho, moviendo la ventana a la izquierda. El RFC desaconseja esto.

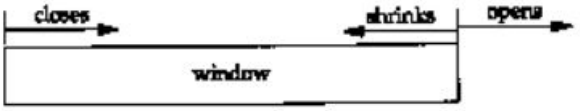


Figure 20.5 Movement of window edges.

Si el borde izquierdo alcanza el borde derecho, se produce el fenómeno **ventana 0.** Esto evita el envío de datos.

Tamaño de ventana

El tamaño de ventana (buffer) de envío y recepción puede ser especificado en la mayoría de APIs. Ejemplo de ventana: Se setean bsdi en 6144 y sun en 8192

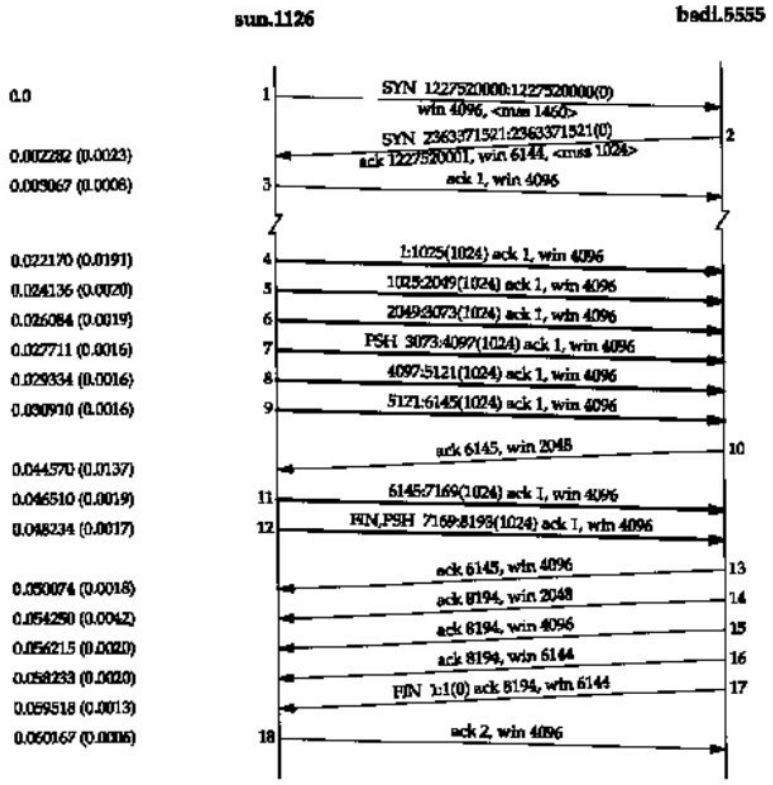


Figure 20.7 Data transfer with receiver offering a window size of 6144 bytes.

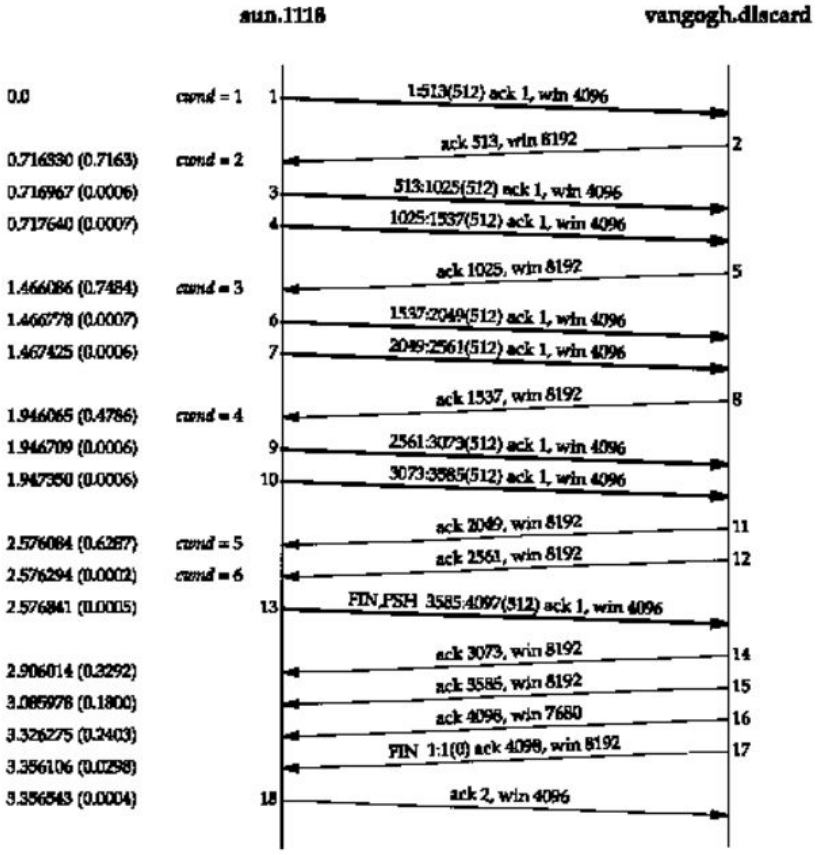
La ventana ofrecida por el receptor es de 6144 bytes en el segmento 2. Por tanto, el cliente envía 6 segmentos de inmediato (segmentos 4-9) y se detiene (exactamente 6144 bytes). El segmento 10 reconoce todos los datos, pero anuncia una ventana de 2048, probablemente porque la aplicación del receptor no ha leído más que esta cantidad de datos. Los segmentos 11 y 12 completan la transferencia de datos desde el cliente. El segmento 12, además, contiene la FLAG de fin encendida. El segmento 13 contiene el mismo número de confirmación que el segmento 10, pero anuncia una ventana más grande. Los segmentos 17 y 18 realizan un cierre normal. *En este caso, se envía la flag PSH activa en el séptimo segmento, ya que se supone que este llenará el buffer del receptor (a pesar de que se especifica una sola lectura de 8192 bytes, TCP realiza 2 escrituras). También es destacable las actualizaciones del tamaño de ventana en los segmentos 15 y 16 (a pesar de que no son necesarios, ya que se ha envíado la flag FIN). De todos modos,* ***las implementaciones suelen enviar esta actualización cuando el tamaño de ventana se incrementa 2 veces el MSS o al menos en un 50% del total de la ventana****. Esto tiene el fin de evitar el* ***síndrome de ventana tonta****.*

Slow Start

Aumenta exponencialmente el envío de paquetes al receptor, hasta detectar síntomas de congestión en la red. Slow start añade otra ventana al emisor TCP: La **ventana de congestión** (o congestion window o **cwnd**). Cuando comienza la conexión, cwnd se inicia en un segmento (con el MSS anunciado por el otro extremo). Cada vez que se recibe un ACK, la cwnd se incrementa en un segmento (como cwnd se expresa en bytes, incrementa la cantidad de bytes del MSS). Esto da como resultado un aumento exponencial del tamaño de la cwnd.

El emisor, entonces, transmite el **mínimo entre la ventana de congestión y la ventana anunciada por el emisor**. La **ventana de congestión es un control de flujo impuesto por el emisor**, mientras que la **ventana anunciada, es un control de flujo impuesto por el receptor**. *En un cierto punto, si un encaminador intermedio comenzaría a descartar paquetes, esto indicaría al emisor de que la cwnd es demasiado grande y debe reducirla.*

Ejemplo:



Rendimiento en transferencia masiva

Como usualmente se tiene que el espacio (de tiempo) entre dos segmentos, y sus correspondientes ACKs, por parte del receptor, es casi el mismo (exceptuando casos de retardos en cola), se dice que TCP es **auto-temporizado** (self-clocking). El estado ideal de una red, se da cuando hay una cantidad de segmentos de datos viajando desde emisor a receptor, a la vez que hay una cantidad idéntica ACKs “volviendo”.

Bandwidth-Delay product (BDP) - Producto retardo ancho de banda

Indica con cuantos bits el emisor puede llenar un canal. La capacidad del canal se puede calcular como: $capacidad\ (bits) = ancho\ de\ banda\ (bits/seg) * RTT(segundos)$. Un aumento de RTT a 2*RTT aumenta un doble el BDP, ya que se requiere mayor cantidad de bits para que el emisor llene el canal:

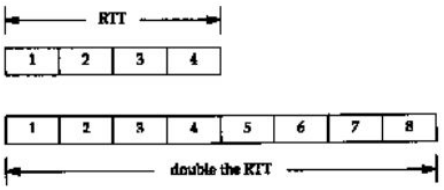
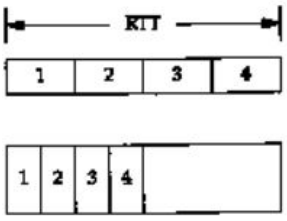


Figure 20.11 Doubling the RTT doubles the capacity of the pipe.

Similarmente, un aumento de BW a 2*BW, aumenta el doble la capacidad de transmisión, requiriendo más datos para llenar el canal (aunque estos se entregan más rápido), que en caso de que aumente el RTT:



Congestión:
La congestión puede ocurrir cuando los datos se envían desde un canal “grande” (LAN) a uno más “pequeño” (WAN) o en routers, cuya capacidad de salida es menor que la suma de las entradas.

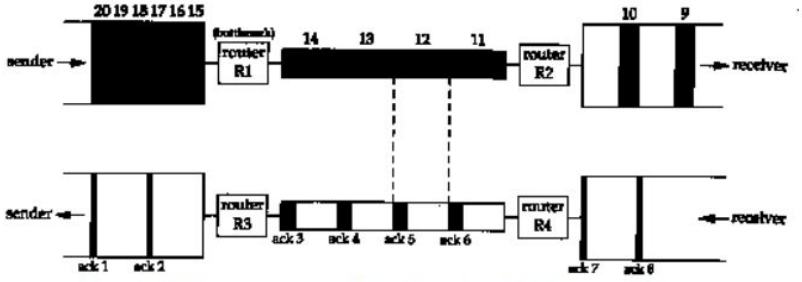


Figure 20.13 Congestion caused by a bigger pipe feeding a smaller pipe.

En la figura 20.13 R1 se etiqueta como el “cuello de botella”, ya que es el punto donde se causa congestión. R1 y R3 pueden, en la mayoría de casos son el mismo router, pero en el ejemplo se suponen distintos. A pesar de que la LAN a la derecha de R1 es bastante rápida, la WAN a su izquierda no es tan rápida, cuando llegan los datos a la LAN receptora, se reciben más “espaciados” en comparación a como fueron enviados. De esta forma, los ACKs llegan con este mismo espaciamiento (auto-temporización). La separación de los ACKs se corresponderá con el ancho de banda del enlace más lento.
En la figura 20.13, además se supone que no se usa slow start, y que los segmentos son enviados tan rápido como la LAN podría llevarlos (se asume, además una ventana anunciada por el receptor de 20 segmentos). También se asume que R1 puede almacenar los 20 segmentos en su buffer (realmente esto no está garantizado, y un router podría empezar a descartar paquetes, lo cual se evita con ***congestion avoidance****).*

Modo urgente

TCP permite que un emisor avise que **dentro de un flujo de datos existen "datos urgentes"**. Cuando el extremo de recepción recibe estos datos, debe decidir qué hacer. Esto se logra seteando dos campos del header TCP: La bandera URG y el puntero urgente (de 16 bits), que con un offset positivo, indica el último byte de datos urgentes (no hay forma de especificar el inicio). **Los datos en modo URG se envían primero a la aplicación**, sin importar su secuencia en el flujo de datos, y se envían aunque la ventana anunciada por el receptor haya sido 0.

En principio, la especificación original dió una doble interpretación respecto a si el puntero refería al último byte de datos urgente, o al byte siguiente a este. Posteriormente, se especificó que el puntero apunta al último byte de datos urgente. Sin embargo, ciertas implementaciones de antes del RFC emplean la segunda metodología ya obsoleta.

TCP debe informar a la aplicación cuando se ha activado el puntero de urgencia, si es que no hay otro ya pendiente en dicha conexión o si el puntero urgente avanza en el flujo de datos. Algunas implementaciones incorrectas llaman el modo urgente de TCP como datos **fuera de banda**. La realidad es que si la aplicación realmente quisiera separar datos del canal, utilizaría una segunda conexión TCP (que es mucho más simple que la complejidad que traería la implementación de esto para la aplicación).

Los usos más comunes del puntero de urgencia, en Telnet y Rlogin se dan para el aviso de pulsación de teclas de interrupción, o para el caso de FTP, cuando un usuario interactivo aborta una transferencia. Sin embargo, en ciertos casos, su uso puede generar confusión.

¿Qué ocurre si el emisor entra en modo urgente varias veces antes de que el proceso receptor procese todos los datos hasta el primer puntero urgente? El puntero urgente simplemente avanza en el flujo de datos, y su posición previa en el receptor se pierde. Sólo hay un puntero urgente en el receptor, y su valor es sobreescrito cada vez que llega un nuevo puntero urgente del otro extremo.

CAPÍTULO 21: TCP - TIMEOUT Y RETRANSMISIÓN

TCP provee un transporte confiable. Una de las formas de confiabilidad, es que un extremo confirme todo lo que recibe. Sin embargo, los segmentos de datos y los ACK pueden perderse. TCP se encarga de esto estableciendo un tiempo de espera al enviar los datos, y si los datos no se reconocen cuando el tiempo expira, los retransmite. Un elemento crítico es **¿Cómo se determina el intervalo de tiempo de espera y con qué frecuencia se produce una retransmisión?**

TCP maneja 4 temporizadores diferentes para la conexión:

1. **Timer de retransmisión**, empleado cuando se espera un reconocimiento del otro extremo.
2. **Timer de persistencia**, para transmitir información acerca del tamaño de ventana aún cuando el otro extremo haya cerrado su ventana de recepción.

3. **Timer keepalive**, para detectar cuando el otro extremo de una conexión se ha desconectado o reiniciado.
4. **Timer 2MSL**, que mide el tiempo que una conexión debe permanecer en TIME_WAIT.

Time out y retransmisión
Varias implementaciones típicas de TCP miden el RTT de los segmentos y utilizan estas mediciones para estimar el tiempo del timer de retransmisión. Respecto a las retransmisiones, TCP utiliza un algoritmo de **retraso (back-off) exponencial**. Normalmente el primer "timeout" se produce 1,5 segundos después de la primera transmisión , a partir del cual cada nueva retransmisión duplica el valor anterior del "timeout", hasta una cota superior de 64 segundos.

*Nota: En algunas implementaciones el primer timeout, en vez de ocurrir 1,5 segundos luego de la transmisión, ocurre en un rango de 0 a 500 ms antes (por ejemplo en 1 segundo), debido a una implementación de un timer de time-out de 500 ms también utilizado en otros temporizadores de TCP. La implementación TCP de Berkeley (BSD) establece tiempos de 500 ms por cada tick. La primera vez que se pone en marcha el primer temporizador, puede iniciar en un rango de 0 a 500 ms. Por ejemplo, teniendo en cuenta que se inicia con tres ticks=0.500*1+0.500*1+0.500*1=1,5 segundos, pero como bien se mencionó el primer tick podría iniciar "en la mitad" del timer.*

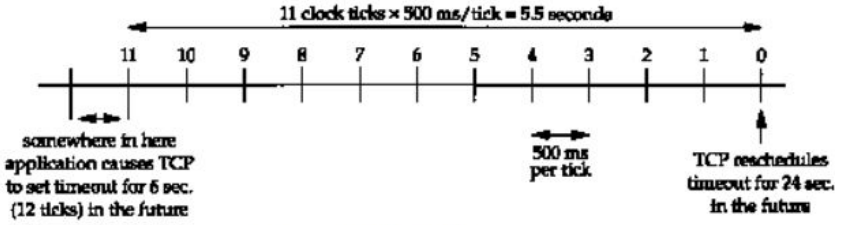


Figure 18.7 TCP 500-ms timer.

Medición de RTT (también ver Kurose)
Para los timeouts y la retransmisión es esencial la medición del RTT, el cual es variable (como el tráfico de red). Por ello TCP debe realizar un seguimiento y modificar el tiempo de retransmisión cuando hay variabilidad. El temporizador debe mantenerse mayor que el valor del RTT, ya que si es menor, se producirá una **expiración prematura** y con ello, **retransmisiones innecesarias**. Sin embargo, si es **demasiado largo**, traería como consecuencia una **reacción lenta ante la pérdida de segmentos**.

Algoritmo de retransmisión adaptativo
TCP debe modificar su timer cuando recibe un ACK de un determinado número de secuencia, ignorando las retransmisiones y reconocimientos acumulativos (a fin de evitar interpretaciones erróneas del RTT). El RTT estimado se calcula de la siguiente manera:
$R = \alpha R + (1-\alpha)M$, *siendo* M *el valor medio del RTT,* α *un suavizado y* R *el RTT estimado.*
En la fórmula, se recomienda un valor de 0.9 para α. El nuevo R' será el resultado de R (el RTT estimado) ponderado en un 90%, más un 10% de la nueva muestra de RTT recibida.

Finalmente el valor del tiempo de retransmisión (RTO, retransmission timeout value), será seteado como:
$RTO = R\beta,\ siendo\ \beta\ un\ factor\ de\ varianza\ de\ retardo,\ de\ valor\ recomendado\ 2.$

Jacobson, sin embargo, recalca que el problema de este algoritmo es no poder "mantenerse" al día del RTT, cuando existen amplias variaciones, dando lugar a retransmisiones prematuras. Por ello, se vuelve necesario calcular la varianza de las mediciones del RTT como:

$$RRTDesv = (1 - \beta) * RTTDesv + \beta * |RTTMuestra - RTTEstimado|$$

Siendo RTTMuestra-RTTEstimado, el margen de error: $Err = RTTMuestra - RTTEstimado$
El valor recomendado de β es 0,25. Finalmente $RTO = R + 4 * RTTDesv$

Algoritmo de Karn
Puede ocurrir un problema si se intentan tomar las **retransmisiones para medir el RTT estimado**. Si se retransmite un segmento, y luego llega un ACK, ¿corresponde a la primera transmisión o a la segunda? Esto se denomina **problema de ambigüedad de retransmisión**. Karn establece que cuando ocurre una retransmisión no se debe actualizar RTO, **a menos hasta que se reciba un ACK de un segmento no retransmitido (o acumulativo)**. *La mayoría de implementaciones de Berkeley no miden el tiempo de varios paquetes en simultáneo, sino que de a uno por vez.*

Cuando se envía un segmento a medir, se almacena su número de secuencia, y cuando este retorna el ACK, se miden nuevamente el RTT y la desviación suavizados.

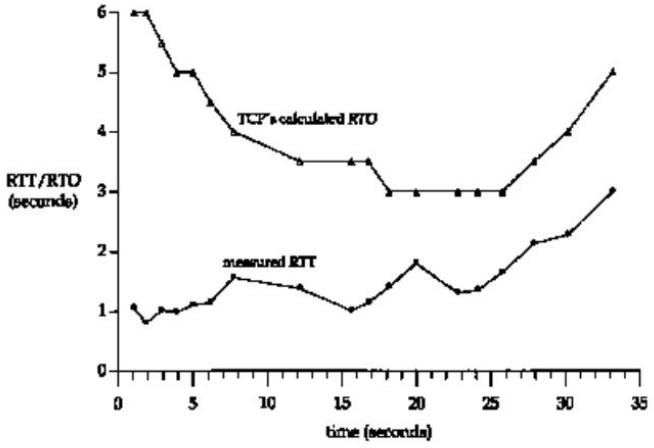


Figure 21.4 Measured RTT and TCP's calculated RTO for example.

Slow Start
Descrito anteriormente, aumenta en uno la cantidad de segmentos a transmitir conforme se reciben ACKs.

Congestión

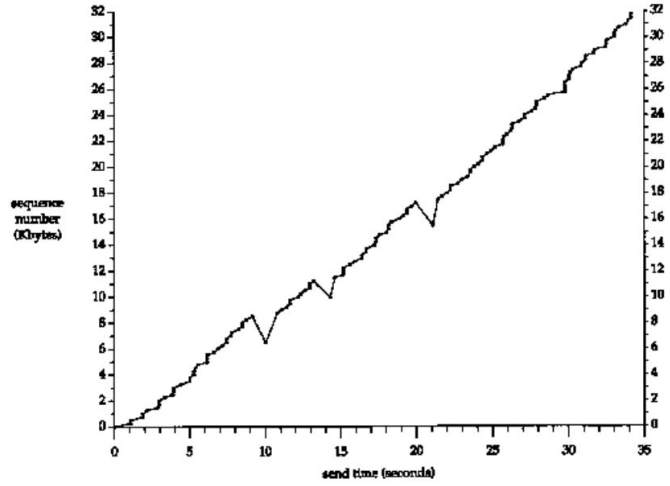


Figure 21.6 Sending of 32768 bytes of data from `slip` to `vangogh`.

En la Figura 21.6, puede observarse la transmisión de segmentos de datos, con su correspondiente número de secuencia (en eje de las y) y el tiempo en que se realizó el envío. Los movimientos hacia abajo y hacia arriba representan las retransmisiones (en los tiempos 10, 14 y 21).

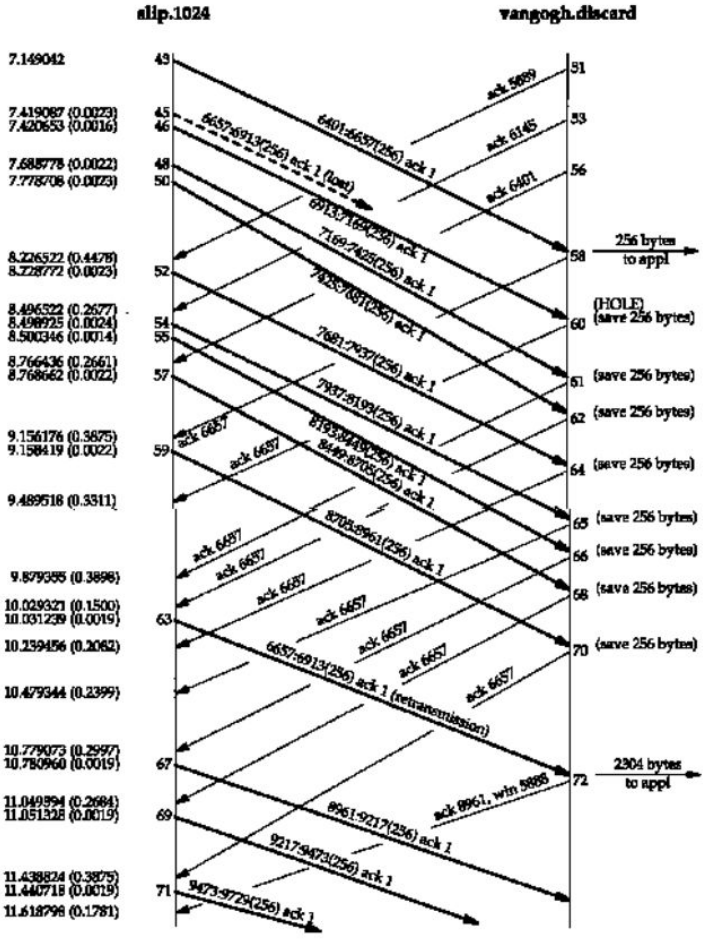


Figure 21.7 Packet exchange for retransmission around the 10-second mark.

Suponiendo que **slip** siempre anuncia una ventana de 4096 bytes y **vangogh** una de 8192 bytes, en esta transmisión, el segmento 45 (6657) se ha perdido o ha llegado dañado. Luego de sucesivas transmisiones del emisor, el receptor sólo responde que no tiene el

segmento 6657. En el tercer de ACK duplicado (segmento 62), el emisor realiza una retransmisión del segmento 6657 (segmento 63). Las implementaciones derivadas de Berkeley cuentan el número de ACKs duplicados recibidos, y cuando un tercero es recibido, se asume que el segmento se ha perdido, y retransmite sólo un segmento, a partir de ese número de secuencia. Este es el algoritmo de **fast retransmit** (retransmisión rápida) de Jacobson, que luego evoluciona a **fast recovery**. Luego de la retransmisión rápida, a partir del segmento 67, la transmisión continúa desde el segmento enviado antes de la retransmisión.
En este caso, la implementación TCP, hace uso de una suerte de "**retransmisión selectiva**", en base al ACK del segmento faltante. Antes de la recepción del segmento 60, el emisor deja un "agujero" por si recibe el segmento, y guarda los consecutivos aunque recibidos "fuera de orden". Algo a destacar es que el emisor, luego de la retransmisión rápida, no espera el ACK, sino que sigue enviando datos. Obsérvese que la ventana anunciada en el segmento 72 es 5888: Esto significa que los 2304 bytes confirmados en este segmento, no han sido leídos por el proceso de usuario.

Congestion Avoidance (Evitación/Anulación de congestión)
Slow start es útil para iniciar el flujo de datos a través de la conexión, pero en algún punto algún router intermedio puede comenzar a descartar paquetes. Congestion Avoidance es la forma de hacer frente a la pérdida de paquetes. Este algoritmo tiene la **presunción que la pérdida causada por daños en el paquete es mínima**, por lo que la pérdida de un paquete estaría más que relacionada con la congestión. Hay dos indicadores de congestión: Un **timeout** y la **recepción de tres ACKs duplicados**.

Slow start y congestion avoidance se implementan en conjunto. Para ello se requieren dos variables a mantener en cada conexión: una **ventana de congestión**, cwnd, y un umbral de slow start (**threshold**), ssthresh. Los algoritmos operan de la siguiente manera:

1. Inicialmente se setea **cwnd** a uno y **ssthresh** (slow start threshold) a 65535 bytes.
2. TCP nunca enviará más que el mínimo entre **cwnd** y la ventana anunciada por el receptor: min(cwnd, rwnd). *Congestion avoidance es el control de flujo impuesto por el emisor, y se basa en la congestión de red percibida, mientras que la ventana anunciada es impuesta por el receptor, y se basa en la cantidad de buffer disponible del mismo para la conexión*.
3. Cuando ocurre congestión (indicado por timeout o la recepción de 3 ACKs duplicados) la mitad de la actual ventana de congestión (es decir min(cwnd, rwnd)). se almacena en **ssthresh**. Adicionalmente, si la congestión se denota por un timeout, se toma un método más "agresivo" y se setea **cwnd** a 1 segmento.
4. Cuando los datos se reconocen por el otro extremo, se incrementa **cwnd**, pero el camino para el incremento depende si se está realizando **slow start** o **congestion avoidance**.
5. Si **cwnd** es menor igual a **ssthresh**, se iniciará con slow start, en otro caso con congestion avoidance. Slow start se ejecuta hasta el umbral especificado (es decir donde ha ocurrido la congestión), y se continúa con **congestion avoidance**.

Mientras que slow start inicia **cwnd** con un segmento y la incrementa con cada ACK recibido, congestion avoidance incrementa la ventana en MSS*MSS/cwnd (o 1/cwnd si se

toma MSS como la unidad), con cada ACK recibido, por lo que se trata de un incremento aditivo, con decremento multiplicativo, en comparación a la exponencialidad de slow start (crecimiento exponencial). En pocas palabras, slow start incrementará cwnd a cwnd*2 por cada RTT, mientras que congestion avoidance incrementará cwnd en 1 MSS por RTT. **Es posible afirmar, entonces, que la ventana se incrementará en 1 MSS cuando recién se reciban una cantidad ACKs igual a cwnd (suponiendo cwnd medido en unidad y no en bytes).**

Nota: El término slow start no es del todo correcto, de hecho la transmisión de paquetes puede ser lenta a causa de la congestión, pero la tasa de inyección a paquetes crece exponencialmente con slow start. Dicha tasa de aumento no se ralentiza hasta ssthresh, que es cuando congestion avoidance entra en juego.

Ejemplo:

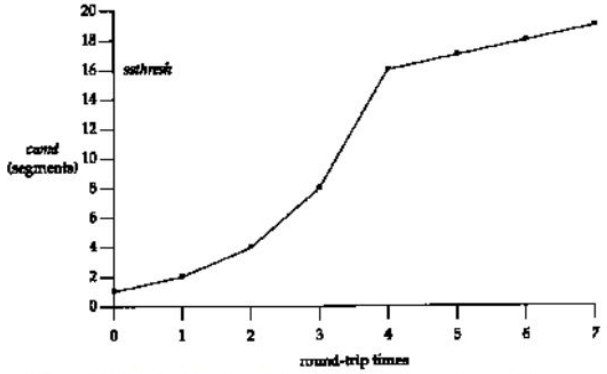


Figure 21.8 Visualization of slow start and congestion avoidance.

Nota: Visualización en unidades de segmentos. En esta figura, se asume que la congestión ha ocurrido cuando cwnd=32*MSS (ssthresh=16), que es hasta donde cwnd aumenta de forma exponencial. Desde el punto del umbral, el crecimiento es lineal, con un máximo de 1 por RTT.

Fast Recovery: Fast retransmit + Congestion avoidance

Cuando TCP recibe segmentos fuera de orden, genera ACKs duplicados, haciendo referencia al segmento faltante por cada nuevo segmento recibido (hasta recibir el segmento). Dado que no es posible saber (a ciencia cierta) si un ACK duplicado es causado por un segmento perdido, o porque los segmentos han llegado desordenados a destino, por ejemplo, dado el caso de que llegue primero #0 y luego #2, y que el receptor envíe ACK #1, para solicitar la retransmisión de este segmento, puede ocurrir que antes de que al emisor le llegue la solicitud de retransmisión, el segmento llegue #1 (desordenado) al receptor (a causa de retrasos en la red). También puede ocurrir que el receptor haya tardado en efectuar el reordenamiento de los segmentos, y antes envíe solicitud de retransmisión del segmento faltante. En cambio, si se reciben 3 o más ACKs, esto indica (con cierta seguridad) que el segmento ha sido perdido. En este caso, se optará por aplicar **fast recovery**: el segmento se enviará antes de que temporizador de pérdida de segmento expire. Luego, se ejecutará **congestion avoidance**, a diferencia de fast retransmit, donde se ejecutaría slow start. Este algoritmo es denominado **fast recovery** (o recuperación rápida).

La recepción de **3 ACKs duplicados**, indica que el otro extremo está recibiendo segmentos. El emisor, entonces, en vez de optar por una reducción “agresiva” de la velocidad de transmisión, entrando en slow start, opta por entrar en fast recovery.

1. Cuando se recibe un tercer ACK duplicado, se setea el **ssthresh** a la mitad de la **cwnd** actual. Se retransmite el segmento perdido y se setea **cwnd** a **sstresh más 3** veces el MSS.
2. Cada vez que se recibe otro ACK duplicado, se incrementa **cwnd** en el MSS y se transmite un paquete (si lo permite el nuevo valor de cwnd).
3. Cuando llega el ACK que reconoce nuevos datos, se setea **cwnd** a **ssthresh** (el valor seteado en el paso 1, es decir el valor de la mitad de la cwnd cuando se produjo la congestión).

El algoritmo de retransmisión rápida apareció por primera vez en la versión 4.3BSD de Tahoe, pero era seguido por slow start. El algoritmo de fast recovery (fast retransmit+congestion avoidance), apareció por primera vez en la versión 4.3BDS Reno.

Nota: Suponiendo que una congestión nunca se produce, el valor de la ventana de congestión estará limitada por la ventana anunciada por el control de flujo del receptor. Si el timeout ocurre en el SYN, **ssthresh** es seteado al mínimo 2*MSS.

Ejemplo de congestión:
Suponiendo MSS de 256:

Segment#	Action			Variable	
(Figure 21.2)	Send	Receive	Comment	*cwnd*	*ssthresh*
	SYN		initialize	256	65535
	SYN	SYN, ACK	timeout retransmit	256	512
	ACK	-	-	-	-
1	1:257(256)				
2		ACK 257	slow start	512	512
3	257:513(256)				
4	513:769(256)				
5		ACK 513	slow start	768	512
6	769:1025(256)				
7	1025:1281(256)				
8		ACK 769	cong. avoid	885	512
9	1281:1537(256)				
10		ACK 1025	cong. avoid	991	512
11	1537:1793(256)				
12	-	ACK 1281	cong. avoid	1089	512

Figure 21.9 Example of congestion avoidance.

En el ejemplo, como ocurre un timeout del SYN, ssthresh se setea a 512 bytes (2MSS), y cwnd a 1 MSS (256 bytes), entrando en la fase de slow start. Slow start se mantiene hasta la recepción del ACK 513 (debido a que cwnd (ACK-1) <= ssthresh). Cuando se recibe ACK 769, como cwnd > ssthresh, se inicia en modo de congestion avoidance. El nuevo valor de cwnd estará dado por: cwnd = cwnd + (MSS*MSS/cwnd+MSS/8), donde el incremento es igual 1/cwnd en caso de que se hable de segmentos como unidades y no como bytes. Nota: En este caso se incluye el término MSS/8 (incorrectamente) para dicha implementación.

En el primer paso: cwnd=768+256*256/768+256/8 = 885 (redondeo entero).
Cuando se recibe el ACK 1025, se calcula: cwnd=885+256*256/885+256/8=991

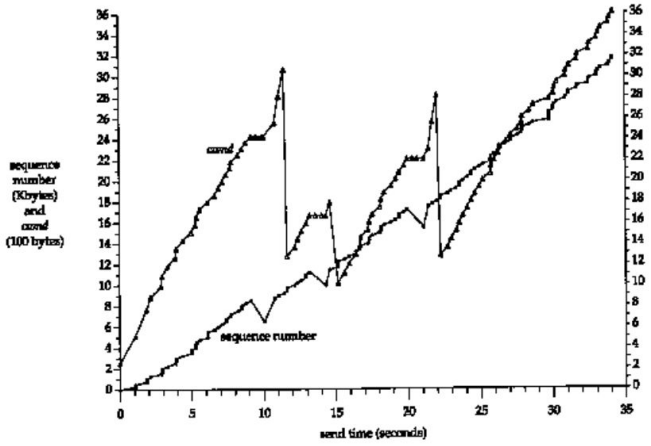


Figure 21.10 Value of *cwnd* and send sequence number while data is being transmitted.

El incremento aditivo en cwnd continua hasta la primera retransmisión. En el caso de la figura, es imposible notar la diferencia visual entre el incremento exponencial y el aditivo, ya que la fase de slow start es muy rápida.

Métricas por ruta

La mayoría de implementaciones TCP, permite que cuando se finaliza una conexión se almacenen los siguientes datos de una conexión: RTT suavizado, desviación media suavizada y umbral de arranque lento. La cantidad de “datos necesarios” es 16 ventanas de datos, lo que permite al filtro de RTT suavizado converger en un 5% del valor correcto. Cuando se inicia una nueva conexión, se utilizan estas métricas almacenadas anteriormente.

Errores ICMP

¿Cómo maneja TCP ciertos errores ICMP? Los más comunes son origen lleno, host inalcanzable y red inalcanzable. Las implementaciones Berkeley más comunes, manejan los errores de la siguiente manera:

1. Un error de origen lleno (source quench) produce que la ventana de congestión (cwnd) se setee a un segmento, para iniciar slow start, sin embargo el threshold, no cambiará.
2. La recepción de host o red inalcanzable, se ignora, ya que estos errores se consideran transitorios (puede que un router intermedio se haya caído). Luego de cierto tiempo, el error se notifica.

Re-paquetización

Cuando se produce un timeout TCP y hay que re-transmitir, no se retransmite el segmento idéntico otra vez. En su lugar, TCP realiza la re-paquetización, enviando un segmento grande (de como máximo MSS), con datos no confirmados durante la espera del ACK.

Resumen: *TCP calcula el tiempo de ida y vuelta y luego utiliza estas mediciones para realizar un seguimiento de un estimador de RTT suavizado y un estimador de desviación*

media suavizada. Estos dos estimadores se utilizan para calcular el siguiente valor de tiempo de retransmisión. Muchas implementaciones sólo miden un solo RTT por ventana. El algoritmo de Karn elimina el problema de la ambigüedad de la retransmisión al impedirnos medir el RTT cuando un paquete se pierde.

CAPÍTULO 22: TCP - TIMER DE PERSISTENCIA

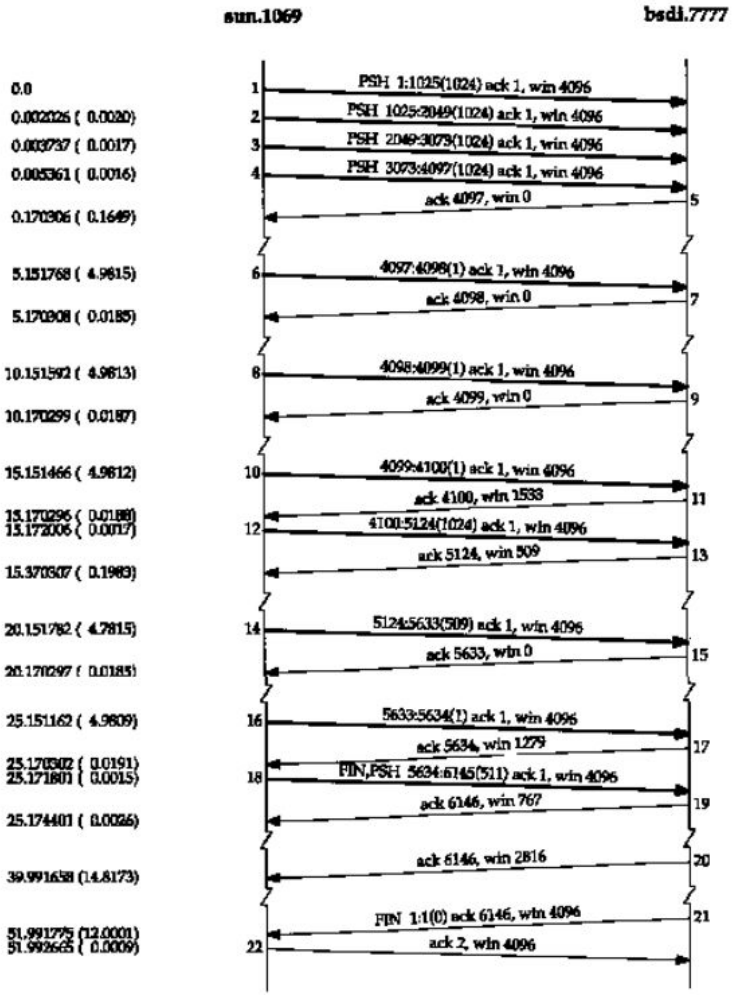
El receptor realiza control de flujo especificando la cantidad de datos que está dispuesto a aceptar (**rwnd**). Cuando la ventana se vuelve 0 (cero), el emisor detiene la transferencia de datos, hasta que rwnd>0.

TCP debe manejar el caso en que se pierda un segmento, de anuncio de nueva ventana (es importante destacar que TCP no confirma los ACK si el segmento que llegó del otro extremo no contiene datos). Si se pierde un acuse de recibo de este tipo, **el receptor podría quedar esperando la apertura de la ventana, mientras que el emisor el envío de datos**. Para prevenir este "deadlock" el emisor usa un **temporizador de persistencia**, que permite consultar el estado de la ventana al emisor. Estos segmentos del emisor se llaman **sondeos de ventana**.

El temporizador de persistencia utiliza un **algoritmo de backoff** (luego de cada sondeo de ventana se duplica el tiempo de espera para el próximo sondeo). El primer timeout se inicia a los 1.5 segundos para una conexión LAN típica. De todas formas el temporizador de persistencia está limitado de 5 a 60 segundos (por lo que si el valor es menor a 5, se establece en 5, y si es mayor a 60, se establece en 60). El segmento que sondea la ventana contiene 1 byte de datos (a TCP se le permite enviar un byte, aunque la ventana del otro extremo esté cerrada). Naturalmente, **el sondeo fuerza un ACK por parte del receptor**.

Síndrome de ventana tonta: El **síndrome de ventana tonta (silly window syndrome, SWS)**, produce que se intercambien **pequeños segmentos de datos en vez de segmentos de tamaño completo** (ya que la ventana del receptor se llena rápido). Esto se produce cuando el receptor **anuncia pequeños tamaños de ventanas** y porque básicamente, emisor y receptor operan a velocidades muy diferentes. Para evitar esto, ambos extremos llevan a cabo acciones:

1. El **receptor no anuncia tamaños pequeños de ventana**: sólo en caso de que el aumento ocurra en un tamaño igual o mayor al MSS, o al menos de la mitad del tamaño de ventana de recepción del receptor (lo que sea menor).
2. El **emisor evita este problema**, transmitiendo solo (a) **cuando se pueda transmitir un segmento completo**, (b) cuando se puede **enviar al menos la mitad de datos que la ventana anunciada** o (c) transmitiendo **cuando se pueda enviar todo lo que se tiene** (o no se está esperando un ACK), o cuando el **algoritmo de Nagle está deshabilitado para la conexión**. La condición (b) es útil para hosts que anuncian tamaños de ventanas pequeños que un MSS. La condición (c), previene el envío de segmentos pequeños, cuando hay datos sin acuse de recibo y el algoritmo de Nagle está activo, lo que también es útil en caso de que la aplicación realice escrituras pequeñas (p. ej, más pequeñas que un MSS). La condición (a) establece qué tamaño de segmento es pequeño (< 1 MSS).



- Se puede observar cómo se evita el síndrome de ventana tonta, en los segmentos 10 y 11: El emisor envía datos ya que la ventana es mayor a 1 MSS, y en frames anteriores, el emisor envía una ventana de tamaño 0, por más que haya leído datos (en la prueba se asume que realiza pequeñas lecturas de 256 bytes).
- A pesar de en el frame 13 la ventana anunciada es de 509 bytes, el receptor no puede anunciar 0, ya que violaría el principio que la ventana de TCP no puede achicarse de derecha a izquierda (tener en cuenta que la ventana anunciada anteriormente fue de 1533). Como se puede observar, luego de esto, el emisor no transmite inmediatamente, pero como el temporizador de persistencia expira, envía datos por 509 bytes, para ver si un ACK confirma un tamaño de ventana mayor. Luego de esto, por más que el receptor tenga espacio en la ventana, anuncia un tamaño de 0.
- A partir del frame 19 (confirmación de cierre de conexión), el emisor no realiza más transmisiones.
- El ACK 20 tiene el fin de anunciar un nuevo tamaño de ventana (se anuncian 2049 más, en comparación a los 767 anteriores): El receptor anuncia su ventana, ya que el tamaño ha aumentado al menos la mitad de su tamaño de buffer de recepción (4096).
- La lectura final por parte de la aplicación ocurre a los 51.99 segundos, que es cuando el receptor envía una notificación de cierre de conexión.

Resumen: *El **timer de persistencia** se utiliza **cuando un extremo anuncia una ventana de tamaño 0**. El emisor va verificando nuevos tamaños de ventana (utilizando un algoritmo de backoff). Es importante, evitar, además, el síndrome de ventana tonta, tanto del lado emisor como receptor.*

CAPÍTULO 23: TCP - TIMER KEEP-ALIVE

Una conexión TCP puede mantenerse a pesar de que los extremos no transmitan datos, o que los sistemas intermedios, caigan, se reinicien, etc (siempre que hayan rutas alternativas). Esto significa que el nivel aplicación de un cliente o servidor debe disponer de temporizadores para finalizar la conexión, por lo que es útil saber si un cliente aún tiene activa la conexión TCP. El timer **keep alive** se encarga de ello.

Este timer no es parte de la especificación TCP. **RFC da tres razones para no usarlo**: (1) Puede **abortar conexiones** temporalmente caídas (por ejemplo, si justo durante la prueba, se crashea un router, el host pensará que fue el extremo quien crasheó), (2) consume **ancho de banda innecesario** y (3) tienen un **costo**, en una red que “cobra” por paquete. El timer keep-alive se cuestiona por ser una **técnica de polling** (operación de consulta constante que degrada el rendimiento). En teoría, debiera ser implementado en capa de aplicación.

El timer keep-alive está especialmente destinado para los servidores, a los que las conexiones TCP le consumen recursos. Por ejemplo, si en una sesión RLogin, el cliente se desconecta sin una correcta desautenticación, el servidor podría permanecer esperando un comando, en una conexión abierta “por la mitad”. El timer keep-alive se suele configurar en los servidores, para detectar estas conexiones half-open. Si no hay actividad por 2 horas, el servidor envía una prueba de segmento al cliente. Se puede presentar uno de cuatro estados:

1. El **cliente funciona correctamente**, por lo que responde la petición (este caso es transparente para la aplicación, ya que esta nunca se entera de la prueba). El servidor TCP resetea el timer keep-alive, para volver a consultar en las próximas 2 horas. El timer se volverá a resetear si hay tráfico.
2. El **servidor está caído**. Si no recibe una respuesta, el servidor prueba cada 75 segundos, 10 veces. Si no hay respuesta alguna, entonces el servidor considera que el cliente ya no existe y finaliza la conexión. En este caso TCP emitirá el mensaje “connection timed out” a la capa de aplicación.
3. El **cliente se reinició**. Aquí el servidor recibe una respuesta, pero un RESET, lo que causa que se termine la conexión. Se mostrará el msg “connection reset by peer”.
4. El **cliente está activo, pero es inalcanzable** por el servidor. Este es el mismo caso que en el punto (2), ya que el servidor no puede distinguir entre ambas situaciones. En ciertos casos, esto puede originar un mensaje ICMP. Normalmente esto se debe a errores temporales en la conexión.

Aunque el valor del timer keep-alive se puede alterar, RFC recomienda que no sea menor a 2 hs. Hay que tener en cuenta que (habitualmente) este cambio afecta a todos los usuarios de una implementación TCP de un host.

Implementación: Se envía un segmento con el último número de secuencia -1. El envío de un número de secuencia "incorrecto" obliga a responder la llamada keep-alive. **El response contendrá el número de secuencia en realidad esperado**. *Nota: algunas implementaciones (antiguas), no responden a las pruebas keepalive, si el segmento no contiene datos. De modo que, algunos sistemas se configuran para enviar "datos basura". Esto no tiene efecto alguno sobre los datos de aplicación, ya que, como este segmento se recibió con anterioridad, se descarta por el receptor. En las implementaciones primero se prueba el envío de un segmento sin datos, y si no hay respuesta, se prueba el envío de 1 byte de datos.*

CAPÍTULO 24: TCP - FUTUROS Y PERFORMANCE

TCP tiene limitaciones en el rendimiento, por ejemplo en el caso en que se use una MTU de 536. El mecanismo de **descubrimiento de MTU mínima** en un camino podría representar una mejora. Para establecer el siguiente tamaño de segmento, normalmente se utiliza el tamaño actual, restando el tamaño de cabeceras TCP e IP. Como los caminos intermedios entre 2 conexiones pueden cambiar, en intervalos de 10 minutos (recomendado por RFC), se deben probar nuevos valores mayores al tamaño de segmento actual. Es importante que el tamaño de segmento varíe en un máximo hasta el MSS anunciado por el otro extremo, o la MTU del enlace del host.

¿Paquetes grandes o paquetes pequeños? El problema de los paquetes pequeños es la cantidad de overhead generado. El problema de los paquetes grandes, es que la mayoría de dispositivos intermedios son "store-and-forward". El tiempo de procesamiento en cada nodo, añadiría tiempo extra en el análisis de cada paquete.

Enlaces grandes
Como recordatorio, el BDP es el tamaño del canal entre dos extremos (BW*RTT). Las redes con un gran BDP, también se pueden desaprovechar en TCP. Por ello, es que TCP implementa la opción de escalado de ventana y timestamp.

Algunos **problemas que se pueden encontrar en enlaces de gran capacidad**:

1. El tamaño de ventana TCP es un campo de 16 bits, por lo que está limitado a 65535 bytes. Teniendo en cuenta la siguiente tabla, se necesita un mayor tamaño para obtener el máximo rendimiento, utilizando la opción de escalado de ventana:

Network	Bandwidth (bits/sec)	Round-trip time (ms)	Bandwidth-delay product (bytes)
Ethernet LAN	10,000,000	3	3,750
T1 telephone line, transcontinental	1,544,000	60	11,580
T1 telephone line, satellite	1,544,000	500	95,500
T3 telephone line, transcontinental	45,000,000	60	337,500
gigabit, transcontinental	1,000,000,000	60	7,500,000

2. Los paquetes perdidos pueden reducir el rendimiento drásticamente (por los algoritmos de fast retransmit y fast recovery).

3. Algunas implementaciones TCP solo miden el RTT una vez por ventana (y no por cada segmento). La opción timestamp en algunos segmentos, permite medir más segmentos.
4. Los números de secuencia pueden rotar en tan solo cuestión de minutos. Por ejemplo, en redes Gigabit, decenas de miles de bytes pueden ser enviados en cuestión de segundos, por lo que el número se secuencia iniciará nuevamente en 0, lo que puede traer problemas de la identificación de paquetes (en caso de que por ejemplo, se hayan retransmitido algunos). En ciertos casos esto se maneja con el algoritmo de PAWS (protection against wrapped sequence numbers), que usa la opción Timestmap de TCP.

Escalado de ventana: Esta opción incrementa el tamaño de ventana de 16 a 32 bits. La opción puede definirse de 0 a 14. El máximo valor de una ventana de 14, vendría dado por 65535*(2^14). Esta opción solo puede definirse en un segmento SYN.

Timestamp: Permite al emisor calcular un RTT, por cada ACK recibido, en base al timestamp enviado por el receptor.

Otras modificaciones propuestas (experimentales) tienen que ver con el modo de transacción, para evitar el three-way handshake y los 4 segmentos de cierre de conexión.

Ejemplo de **performance TCP**, en relación a los datos de usuario, teniendo en cuenta: Red de 10 Mbits y overhead.

Field	Data #bytes	ACK #bytes
Ethernet preamble	8	8
Ethernet destination address	6	6
Ethernet source address	6	6
Ethernet type field	2	2
IP header	20	20
TCP header	20	20
user data	1460	0
pad (to Ethernet minimum)	0	6
Ethernet CRC	4	4
interpacket gap (9.6 microsec)	12	12
total	1538	84

Figure 24.9 Field sizes for Ethernet theoretical maximum throughput calculation.

Si se tiene en cuenta que el máximo de una ventana TCP es 65535 (sin usar el escalado de ventana), esto permite una ventana de 44 de 1460 segmentos. Si el receptor envía un ACK cada 22 segmentos, el cálculo vendría dado por:

$$rendimiento = 22 * 1460\ bytes\ /\ (22 * 1568 + 84) * 10000000\ bits/sec/8\ bits/byte = 1183667\ bytes/sec$$

Este es el límite teórico: Se tiene en cuenta que todos los ACKs llegan bien, que el emisor siempre puede transmitir los segmentos con el mínimo espaciado de Ethernet, etc. Los valores en la práctica siempre son menores a los teóricos.