UNIVERSIDAD NACIONAL DEL LITORAL

FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

REDES Y COMUNICACIONES DE DATOS II

RESUMEN
SEGUNDO PARCIAL

Autor:
Carlos GENTILE

E-mail:
csgentile@gmail.com

JUNIO, 2016

6 LA CAPA DE TRANSPORTE

6.1. EL SERVICIO DE TRANSPORTE

6.1.1. Servicios proporcionados a las capas superiores

La meta fundamental de la capa de transporte es proporcionar un servicio eficiente, confiable y económico a sus usuarios, que normalmente son procesos de la capa de aplicación. Para lograr este objetivo, la capa de transporte utiliza los servicios proporcionados por la capa de red. El hardware o software de la capa de transporte que se encarga del trabajo se llama **entidad de transporte**, la cual puede estar en el *kernel* (núcleo) del sistema operativo, en un proceso de usuario independiente, en un paquete de biblioteca que forma parte de las aplicaciones de red o en la tarjeta de red. En la Figura 6.1 se ilustra la relación (lógica) entre las capas de red, transporte y aplicación.

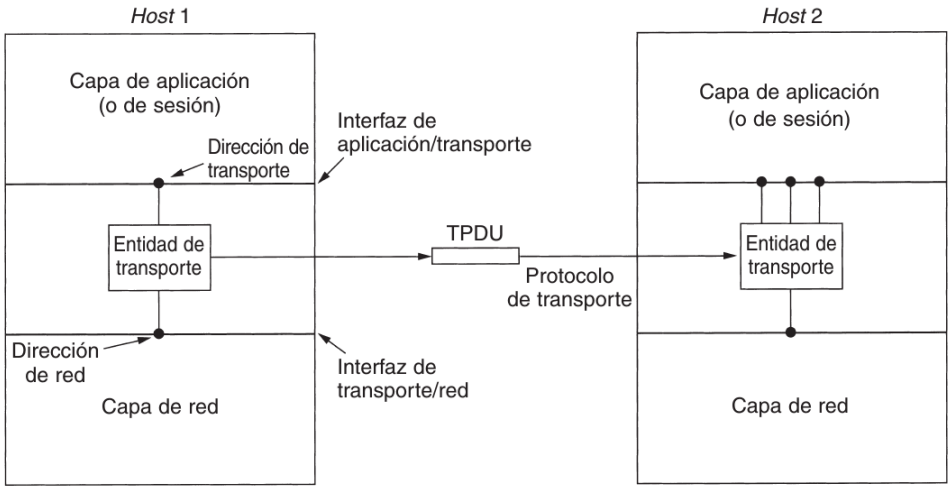

Figura 6.1: Las capas de red, transporte y aplicación.

Hay dos tipos de servicios de transporte: un **servicio orientado a la conexión** y otro **servicio no orientado a la conexión**. Si bien ambos servicios son similares a sus pares de la capa de red, hay un motivo por el cual se necesita una capa más: el código de transporte se ejecuta por completo en las máquinas de los usuarios, pero la capa de red, por lo general, se ejecuta en los enrutadores, los cuales son operados por la empresa portadora. Si la capa de red ofrece un servicio poco confiable, perdiendo paquetes con frecuencia o los enrutadores se caen de vez en cuando se generan inconvenientes, ya que los usuarios no tienen control sobre la capa de red. La única posibilidad es poner encima de la capa de red otra capa que mejore la calidad del servicio.

La existencia de la capa de transporte hace posible que el servicio de transporte sea más confiable que el servicio de red subyacente. La capa de transporte puede detectar y compensar paquetes perdidos y datos alterados. Más aún, gracias a la capa de transporte, es posible escribir programas de aplicación usando un conjunto estándar de primitivas, y que estos programas funcionen en una amplia variedad de redes sin necesidad de preocuparse por lidiar con diferentes interfaces de subred y transmisiones no confiables. En el mundo real esta capa cumple la función clave de aislar a las capas superiores de la tecnología, el diseño y las imperfecciones de la subred. Por esta razón suele hacerse la siguiente distinción:

- Capas 1 a 4: **proveedor** del servicio de transporte.
- Capas 5 a 7: **usuario** del servicio de transporte.

6.1.2. Primitivas del servicio de transporte

Para permitir que los usuarios accedan al servicio de transporte, la capa de transporte debe proporcionar algunas operaciones a los programas de aplicación, es decir, una interfaz del servicio de transporte. Cada servicio de transporte tiene su propia interfaz. El servicio de transporte es parecido al servicio de red, pero hay algunas diferencias importantes. La principal diferencia es que el propósito del servicio de red es modelar el servicio ofrecido por las redes reales, con todos sus problemas. Las redes reales pueden perder paquetes, por lo que el servicio de red generalmente no es confiable. En cambio, el servicio de transporte orientado a la conexión tiene por finalidad ocultar las imperfecciones del servicio de red para que los procesos usuarios puedan dar por hecho simplemente la existencia de un flujo de bits libre de errores, es decir, ofrecer un servicio confiable.

La capa de transporte también puede proporcionar un servicio no confiable (de datagramas). Hay algunas aplicaciones que se benefician del transporte no orientado a la conexión, como la computación cliente-servidor y la multimedia de flujo continuo.

La segunda diferencia reside en a quién está dirigido el servicio. El servicio de red lo usan únicamente las entidades de transporte. Pocos usuarios escriben sus propias entidades de transporte y, por lo tanto, pocos usuarios o programas llegan a ver los aspectos internos del servicio de red. En contraste, muchos programas ven las primitivas de transporte. En consecuencia, el servicio de transporte debe ser adecuado y fácil de usar.

Una interfaz sencilla para el servicio de transporte se muestra en las primitivas de la Figura 6.2. La esencia de lo que debe hacer una interfaz de transporte orientada a la conexión: permitir que los programas de aplicación establezcan, usen y liberen conexiones.

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

Figura 6.2: Primitivas de un servicio de transporte sencillo.

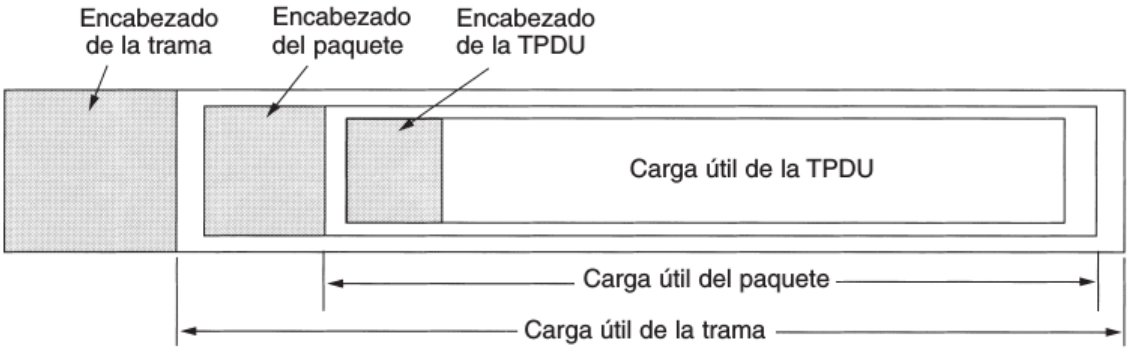

Figura 6.3: Anidamiento de las TPDUs, los paquetes y las tramas.

Los mensajes enviados de una entidad de transporte a otra se llaman **Unidad de Datos del Protocolo de Transporte (TPDU: *Transport Protocol Data Unit*)**. La carga útil de una TPDU y su encabezado se encuentran anidados adentro de un paquete de red y éste adentro de una trama de capa de enlace, como puede verse en la figura 6.3.

Para ilustrar el uso de las primitivas de transporte definidas antes tomamos por ejemplo un esquema Cliente-Servidor, en el cual un *host* solicita conexión a un servicio ofrecido por un servidor de aplicación. El funcionamiento puede resumirse así:

1. El servidor ejecuta una primitiva `LISTEN` y se bloquea hasta que aparezca un cliente.
2. Cuando un cliente solicita conectarse ejecuta una primitiva `CONNECT`, bloqueando al emisor y enviando una TPDU `CONNECTION REQUEST` al servidor.
3. La entidad de transporte del servidor recibe la solicitud de conexión, emite una TPDU `CONNECTION ACCEPTED` y se desbloquea.

4. Al llegar la TPDU `CONNECTION ACCEPTED` al cliente, éste se desbloquea y puede empezar la transferencia de datos usando las primitivas `SEND` y `RECEIVE`. La primera envía la información, mientras que la segunda bloquea al receptor esperando una TPDU `DATA`. Al terminar el envío de un lado se bloquea esperando la respuesta del otro y así sucesivamente. También se enviarán confirmaciones de recepción a la vez que las entidades de transporte llevan temporizadores y pueden efectuar retransmisiones.
5. Hay dos formas de finalizar la conexión:
   - **Desconexión asimétrica**: cualquiera de los dos usuarios de transporte puede emitir una primitiva `DISCONNECT`, que resulta en el envío de una TPDU `DISCONNECT` a la entidad de transporte remota. A su llegada, se libera la conexión.
   - **Desconexión simétrica**: cada parte se cierra por separado, independientemente de la otra. Cuando una de las partes emite una primitiva `DISCONNECT`, quiere decir que ya no tiene más datos por enviar, pero aún está dispuesta a recibir datos de la otra parte. En este modelo, una conexión se libera cuando ambas partes han emitido una primitiva `DISCONNECT`.

En la Figura 6.4 se presenta un diagrama de estado del establecimiento y liberación de una conexión con estas primitivas sencillas. Cada transición es activada por algún evento, ya sea una primitiva ejecutada por el usuario de transporte local o la llegada de un paquete. Suponemos que la confirmación de recepción de cada TPDU se realiza por separado, que se usa un modelo de desconexión simétrica, y que el cliente la realiza primero.

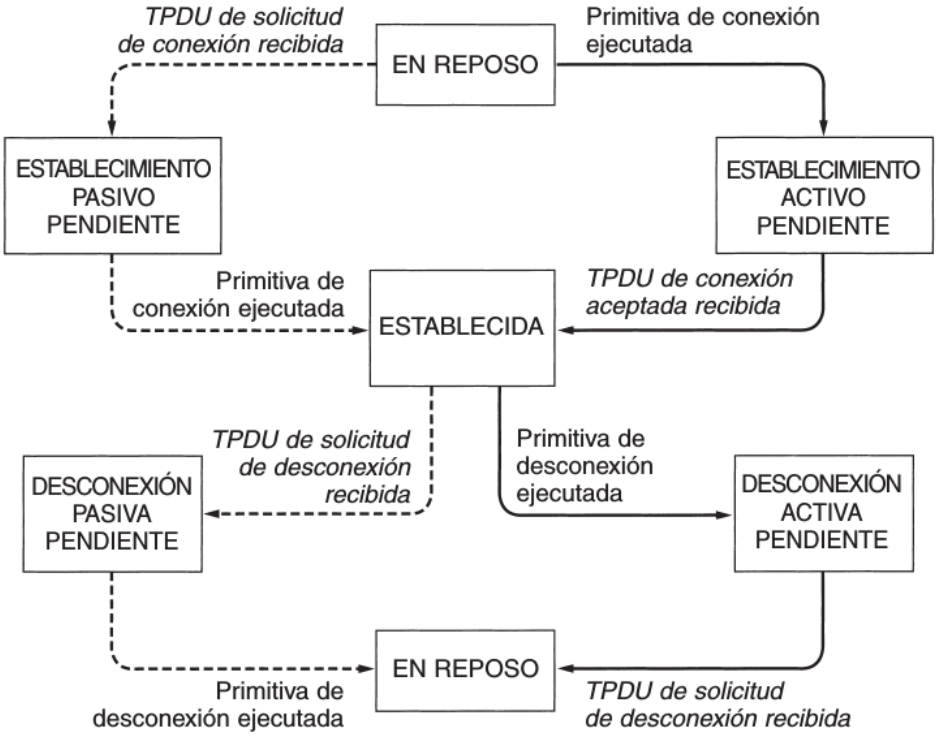

Figura 6.4: Diagrama de estado de un esquema sencillo de manejo de conexiones. Las transiciones escritas en cursivas son causadas por llegadas de paquetes. Las líneas continuas muestran la secuencia de estados del cliente. Las líneas punteadas muestran la secuencia de estados del servidor.

6.2. ELEMENTOS DE LOS PROTOCOLOS DE TRANSPORTE

El servicio de transporte se implementa mediante un **protocolo de transporte** entre las dos entidades de transporte. En ciertos aspectos, los protocolos de transporte se parecen a los protocolos de enlace de datos. Ambos se encargan del control de errores, la secuenciación y el control de flujo, entre otros aspectos. Sin embargo, existen diferencias significativas entre los dos, las cuales se resumen en el Cuadro 6.1.

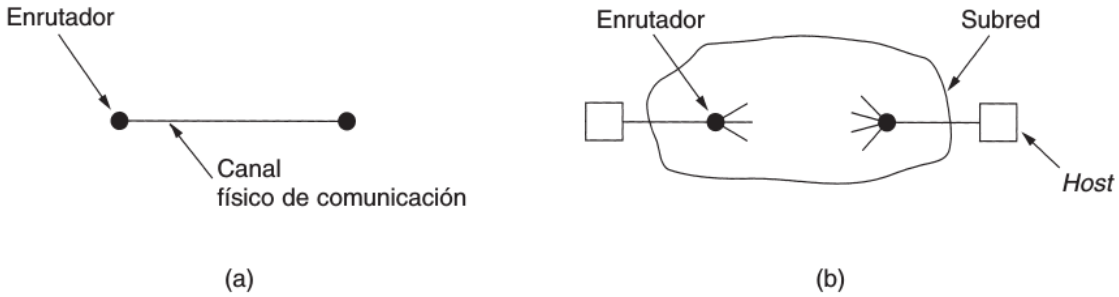

Figura 6.5: (a) Entorno de la capa de enlace de datos. (b) Entorno de la capa de transporte.

Capa de Enlace	Capa de Transporte
Dos enrutadores se comunican por un canal físico.	El canal físico es reemplazado por una subred completa.
No es necesario que un enrutador especifique el enrutador con el que quiere comunicarse; cada línea de salida especifica de manera única un enrutador en particular.	Se requiere el direccionamiento explícito de los destinos.
El proceso de establecimiento de una conexión a través del cable de la Figura 6.5(a) es sencillo: el otro extremo siempre está ahí (a menos que se caiga).	El establecimiento inicial de la conexión es más complicado.
No existe capacidad de almacenamiento (no confundir con búferes).	Hay una existencia potencial de capacidad de almacenamiento en la subred. Al enviar un enrutador una trama, ésta puede llegar o perderse, pero no puede andar de un lado a otro durante un tiempo, sin ser detectada y aparecer repentinamente 30 segundos después. Si la subred usa datagramas y enrutamiento adaptativo internamente, hay una probabilidad nada despreciable de que un paquete pueda almacenarse durante varios segundos y entregarse después. Las consecuencias de esta capacidad de almacenamiento de paquetes en la subred pueden ser desastrosas y requerir el uso de protocolos especiales.
Se requieren búferes y control de flujo. Algunos de los protocolos asignan una cantidad fija de búferes a cada línea de modo que, al llegar una trama, siempre hay un búfer disponible.	También se requieren búferes y control de flujo, pero la gran cantidad de conexiones que deben manejarse hace menos atractiva la idea de dedicar muchos búferes a cada una.

Cuadro 6.1: Comparación entre los protocolos de capa de enlace y capa de transporte.

6.2.1. Direccionamiento

Cuando un proceso (por ejemplo, un usuario) de aplicación desea establecer una conexión con un proceso de aplicación remoto, debe especificar a cuál se conectará. El método que normalmente se emplea es definir direcciones de transporte en las que los procesos pueden estar a la escucha de solicitudes de conexión. En Internet, estos puntos terminales se denominan **puertos**. Se utilizan dos términos genéricos para definir a los terminales de las capas de transporte y de red, respectivamente:

- **TSAP**: Punto de Acceso al Servicio de Transporte.
- **NSAP**: Punto de Acceso al Servicio de Red (e.g. direcciones IP).

En la Figura 6.6 se ilustra la relación entre el NSAP, el TSAP y la conexión de transporte. Los procesos de aplicación, tanto clientes como servidores, se pueden enlazar por sí mismos a un TSAP para establecer una conexión a un TSAP remoto. Estas conexiones se realizan a través de NSAPs en cada host, como se muestra. Como en algunas redes cada computadora tiene un solo NSAP, los TSAPs sirven para distinguir los múltiples puntos terminales de transporte que comparten un NSAP.

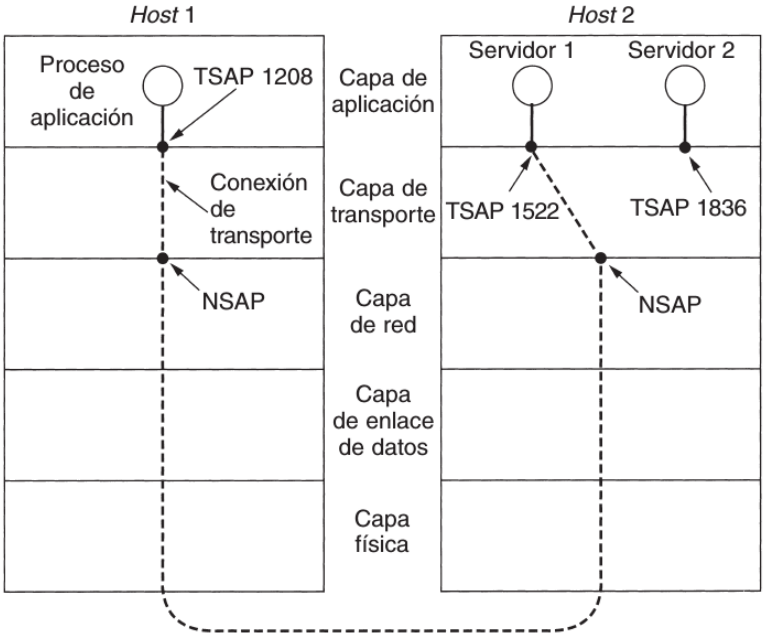

Figura 6.6: TSAPs, NSAPs y conexiones de transporte.

El **protocolo inicial de conexión** funciona de la siguiente manera: en lugar de que cada servidor escuche en un TSAP bien conocido, cada máquina que desea ofrecer servicio a usuarios remotos tiene un servidor de procesos especial que actúa como *proxy* de los servidores de menor uso. Este servidor escucha en un grupo de puertos al mismo tiempo, esperando una solicitud de conexión. Los usuarios potenciales de un servicio comienzan por emitir una solicitud `CONNECT`, especificando la dirección TSAP del servicio que desean. Si no hay ningún servidor esperándolos, consiguen una conexión al servidor de procesos. Tras obtener la solicitud entrante, el servidor de procesos genera el servidor solicitado, permitiéndole heredar la conexión con el usuario existente. El nuevo servidor entonces hace el trabajo requerido, mientras que el servidor de procesos vuelve a escuchar solicitudes nuevas.

Aunque el protocolo de conexión inicial funciona bien para aquellos servidores que pueden crearse conforme son necesarios, hay muchas situaciones en las que los servicios existen independientemente del servidor de procesos. Por ejemplo, un servidor de archivos necesita operar en un hardware especial (una máquina con un disco) y no puede simplemente crearse sobre la marcha cuando alguien quiere comunicarse con él. Para manejar esta situación, se usa con frecuencia un esquema alterno. En este modelo, existe un proceso especial llamado servidor de nombres, o a veces servidor de directorio. Para encontrar la dirección TSAP correspondiente a un nombre de servicio dado el usuario establece una conexión con el servidor de nombres (que escucha en un TSAP bien conocido). Entonces el usuario envía un mensaje especificando el nombre del servicio, y el servidor de nombres devuelve la dirección TSAP. Luego el usuario libera la conexión con el servidor de nombres y establece una nueva con el servicio deseado. En este modelo, al crearse un servicio nuevo, debe registrarse en el servidor de nombres, dando tanto su nombre de servicio (generalmente, una cadena ASCII) como la dirección de su TSAP. El servidor de nombres registra esta información en su base de datos interna.

6.2.2. Establecimiento de una conexión

A primera vista, parecería suficiente con que una entidad de transporte enviara una TPDU `CONNECTION REQUEST` al destino y esperar una respuesta `CONNECTION ACCEPTED`. El problema ocurre cuando la red puede perder, almacenar o duplicar paquetes o bien, por ejemplo, una subred muy congestionada donde las confirmaciones de recepción casi nunca regresan a tiempo, y cada paquete expira y se retransmite dos o tres veces. El principal problema es el eventual arribo de duplicados retrasados. Esto puede atacarse de varias maneras, ninguna de las cuales es muy satisfactoria.

Una es usar direcciones de transporte desechables. En este enfoque, cada vez que se requiere una dirección de transporte, se genera una nueva. Al liberarse la conexión, se descarta la dirección y no se vuelve a utilizar. Esta estrategia imposibilita el modelo de servidor de procesos descripto antes.

Otra posibilidad es dar a cada conexión un identificador de conexión (es decir, un número de secuencia que se incrementa con cada conexión establecida), seleccionado por la parte iniciadora, y ponerlo en cada TPDU, incluida la que solicita la conexión. Tras la liberación de una conexión, cada entidad de transporte podría actualizar una tabla que liste conexiones obsoletas como

pares `(entidad de transporte, identificador de conexión)`. Cada vez que entrara una solicitud de conexión, podría cotejarse con la tabla para saber si pertenece a una conexión previamente liberada. Por desgracia, este esquema tiene una falla básica: requiere que cada entidad de transporte mantenga una cierta cantidad de información histórica durante un tiempo indefinido. Si se cae una máquina y pierde su memoria, ya no sabrá qué identificadores de conexión usó.

Se necesita un enfoque diferente. En lugar de permitir que los paquetes vivan eternamente en la subred, se debe diseñar un mecanismo para eliminar a los paquetes viejos que aún andan vagando por ahí. Si se puede asegurar que ningún paquete viva más allá de cierto tiempo conocido, el problema se vuelve algo más manejable. El tiempo de vida de un paquete puede restringirse a un máximo conocido usando una de las técnicas mencionadas a continuación.

1. **Un diseño de subred restringido**. Incluye cualquier método que evite que los paquetes hagan ciclos y limite el retardo por congestión.
2. **Colocar un contador de saltos en cada paquete**. Consiste en inicializar el conteo de saltos con un valor apropiado y decrementarlo cada vez que se reenvíe el paquete. Se descarta cualquier paquete cuyo contador de saltos llega a cero.
3. **Marcar el tiempo en cada paquete**. Se requiere que cada paquete lleve la hora en la que fue creado, y que los enrutadores se pongan de acuerdo en descartar cualquier paquete que haya rebasado cierto tiempo predeterminado. Se requiere también que los relojes de los enrutadores estén sincronizados (difícil).

En la práctica, necesitaremos garantizar no sólo que el paquete está eliminado, sino que también lo están todas sus confirmaciones de recepción, para lo cual se espera un tiempo T, que es un múltiplo pequeño del máximo tiempo de vida de paquete. Pasado este tiempo, tras el envío de un paquete, se puede estar seguro de que tanto el paquete como las confirmaciones han desaparecido.

Método de Tomlinson

Para resolver el problema de una máquina que pierde toda la memoria acerca de su estado tras una caída, Tomlinson propuso equipar cada *host* con un reloj de hora del día con las siguientes características:

- Los relojes de los diferentes *hosts* no necesitan estar sincronizados.
- Cada reloj tiene la forma de un contador binario que se incrementa a sí mismo a intervalos uniformes.
- La cantidad de bits del contador debe ser igual o mayor que la cantidad de bits en los números de secuencia.
- Lo más importante: se supone que el reloj continúa operando aun ante la caída del *host*.

La idea básica es asegurar que nunca estén pendientes al mismo tiempo dos TPDUs de número idéntico. Cuando se establece una conexión, los k bits de orden menor del reloj se usan como número inicial de secuencia (también k bits). Por tanto cada conexión numera sus TPDUs con un número de secuencia inicial diferente y el reinicio de la numeración ocurre mucho tiempo después que los paquetes expiraron. Una vez que ambas entidades de transporte han acordado el número de secuencia inicial, puede usarse cualquier protocolo de ventana corrediza para el control de flujo de datos.

Acuerdo de tres vías

El método basado en reloj resuelve el problema del duplicado retrasado de las TPDUs de datos, pero para que este método resulte de utilidad, debe establecerse primero una conexión. Dado que las TPDUs de control también pueden retrasarse, está el problema potencial de lograr que ambos lados acuerden el número de secuencia inicial. Supongamos, por ejemplo, que se establecen conexiones haciendo que el *host* 1 envíe una TPDU `CONNECTION REQUEST` al *host* 2 con número de secuencia inicial y número de puerto de destino dados. El receptor (*host* 2), confirma entonces la recepción de esta solicitud enviando de regreso una TPDU `CONNECTION ACCEPTED`. Si la TPDU `CONNECTION REQUEST` se pierde, pero aparece con retardo una `CONNECTION REQUEST` duplicada en el *host* 2, se establecerá incorrectamente la conexión.

Para resolver este problema, Tomlinson desarrolló el **acuerdo de tres vías** (*three-way handshake*). Este protocolo de establecimiento no requiere que ambos lados comiencen a transmitir con el mismo número de secuencia, por lo que puede usarse con otros métodos de sincronización distintos del método de reloj global. El procedimiento normal de establecimiento al iniciar el *host* 1 se muestra en la Figura 6.7(a). El *host* 1 escoge un número de secuencia, x, y envía al *host* 2 una TPDU `CONNECTION REQUEST` que lo contiene. El *host* 2 responde con una TPDU `CONNECTION ACCEPTED` confirmando la recepción de x y anunciando su propio número de secuencia inicial, y. Por último, el *host* 1 confirma la recepción de la selección de un número de secuencia inicial del *host* 2 en la primera TPDU de datos que envía.

En la figura Figura 6.7(b), la primera TPDU es una `CONNECTION REQUEST` duplicada con retraso de una conexión vieja. Esta

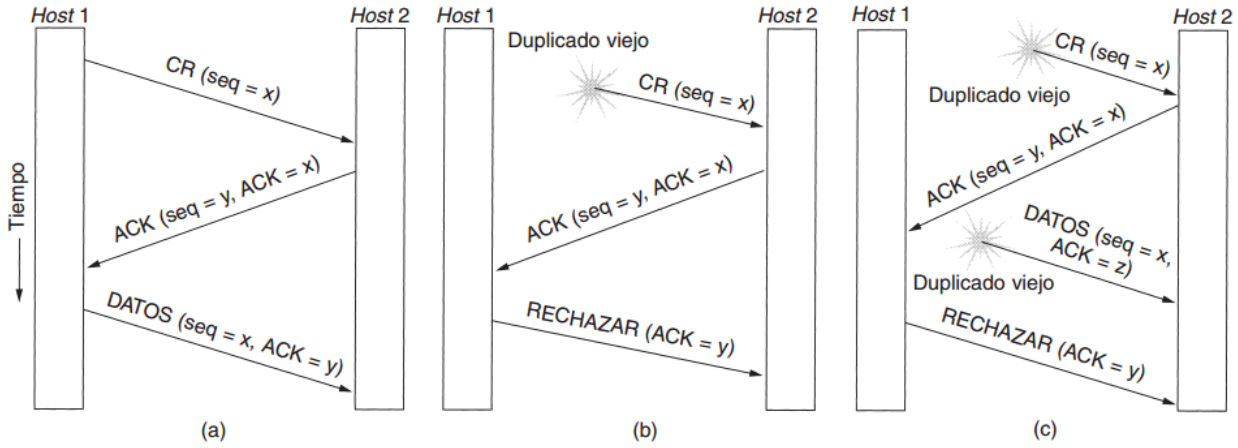

Figura 6.7: Tres escenarios para establecer una conexión usando un acuerdo de tres vías. CR significa `CONNECTION REQUEST`. (a) Operación normal. (b) `CONNECTION REQUEST` duplicada vieja que aparece de la nada. (c) `CONNECTION REQUEST` duplicada y `ACK` duplicada.

TPDU llega al *host* 2 sin el conocimiento del *host* 1. El *host* 2 reacciona a esta TPDU enviando al *host* 1 una TPDU *ACK*, solicitando de hecho la comprobación de que el *host* 1 en verdad trató de establecer una nueva conexión. Al rechazar el *host* 1 el intento de establecimiento de conexión del *host* 2, éste se da cuenta de que fue engañado por un duplicado con retardo y abandona la conexión. De esta manera, un duplicado con retardo no causa daño.

El peor caso ocurre cuando en la subred deambulan tanto una `CONNECTION REQUEST` retardada como una `ACK`. Este caso se muestra en la Figura 6.7(c). Como en el ejemplo previo, el *host* 2 recibe una `CONNECTION REQUEST` retrasada y la contesta. En este momento es crucial notar que el *host* 2 ha propuesto usar y como número de secuencia inicial para el tráfico del *host* 2 al *host* 1, sabiendo bien que no existen todavía TPDUs que contengan el número de secuencia y ni confirmaciones de recepción de y. Cuando llega la segunda TPDU retrasada al *host* 2, el hecho de que se confirmó la recepción de z en lugar de y indica al *host* 2 que éste también es un duplicado viejo.

6.2.3. Liberación de una conexión

Hay dos estilos de terminación de una conexión:

- **Liberación asimétrica**. Es la manera en que funciona el sistema telefónico: cuando una parte cuelga, se interrumpe la conexión. Es abrupta y puede resultar en la pérdida de datos. En la Figura 6.8, tras establecerse la conexión, el *host* 1 envía una TPDU que llega adecuadamente al *host* 2. Entonces el *host* 1 envía otra TPDU. Desgraciadamente, el *host* 2 emite una `DISCONNECT` antes de llegar la segunda TPDU. El resultado es que se libera la conexión y se pierden datos.

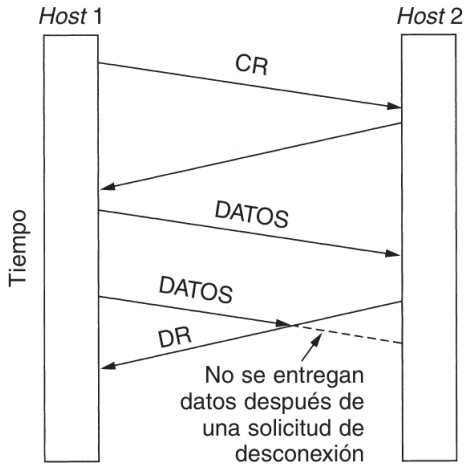

Figura 6.8: Desconexión abrupta con pérdida de datos.

- **Liberación simétrica**. Trata la conexión como dos conexiones unidireccionales distintas, y requiere que cada una se libere por separado. Aquí, un *host* puede continuar recibiendo datos aun tras haber enviado una TPDU `DISCONNECT`. La

liberación simétrica es ideal cuando cada proceso tiene una cantidad fija de datos por enviar y sabe con certidumbre cuándo los ha enviado. Podríamos pensar en un protocolo en el que el *host* 1 diga: "Ya terminé. ¿Terminaste también?" Si el *host* 2 responde: "Ya terminé también. Adiós", la conexión puede liberarse con seguridad. Por desgracia este protocolo no siempre funciona. El inconveniente se ilustra con el **problema de los 2 ejércitos**. Si el canal de comunicación entre las dos partes de la conexión es no confiable, entonces pueden no llegar a tiempo o perderse confirmaciones de solicitudes de desconexión. Dado que el emisor del mensaje final nunca puede estar seguro de su llegada, no se desconectará. Peor aún, el otro *host* sabe esto, por lo que no se desconectará tampoco. Si ninguna de las partes está preparada para desconectarse hasta estar convencida de que la otra está preparada para desconectarse también, nunca ocurrirá la desconexión. De todas formas, aunque este protocolo no es infalible, generalmente es adecuado.

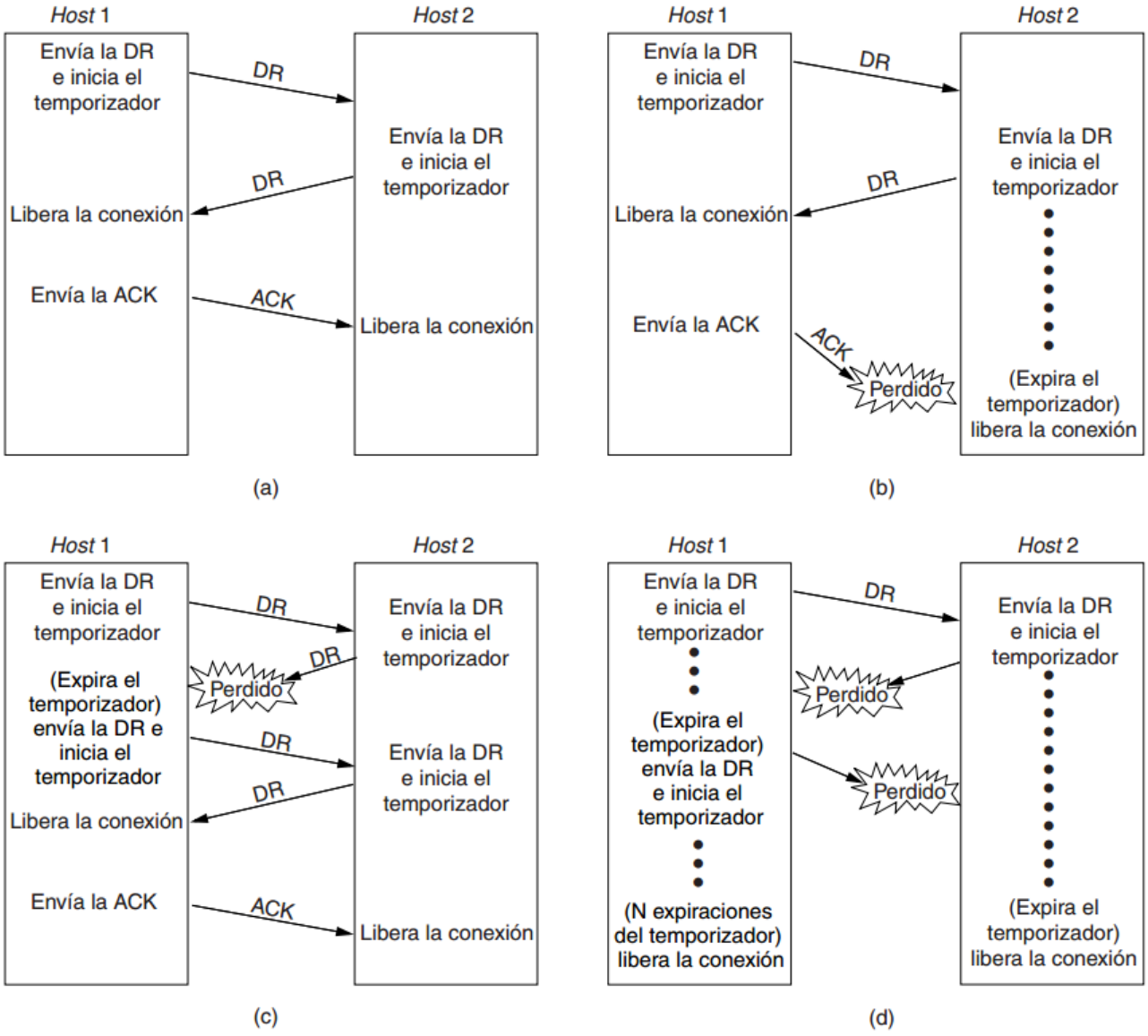

Figura 6.9: Cuatro escenarios de un protocolo para liberar una conexión. (a) Caso normal del acuerdo de tres vías. (b) Pérdida de la última ACK. (c) Respuesta perdida. (d) Respuesta perdida y pérdida de las DRs subsecuentes.

En la Figura 6.9(a) vemos el caso normal en el que uno de los usuarios envía una TPDU `DR` (`DISCONNECTION REQUEST`, solicitud de desconexión) a fin de iniciar la liberación de una conexión. Al llegar, el receptor devuelve también una TPDU `DR` e inicia un temporizador, para el caso de que se pierda su `DR`. Al llegar esta `DR`, el emisor original envía de regreso una TPDU `ACK` y libera la conexión. Finalmente, cuando la TPDU `ACK` llega, el receptor también libera la conexión. La liberación de una conexión significa que la entidad de transporte remueve la información sobre la conexión de su tabla de conexiones abiertas y avisa de alguna manera al dueño de la conexión (el usuario de transporte). Esta acción es diferente a aquélla en la que el usuario de transporte emite una primitiva `DISCONNECT`.

Si se pierde la última TPDU `ACK`, como se muestra en la Figura 6.9(b), el temporizador salva la situación. Al expirar el temporizador, la conexión se libera de todos modos.

Ahora consideremos el caso de la pérdida de la segunda DR. El usuario que inicia la desconexión no recibirá la respuesta esperada, su temporizador expirará y todo comenzará de nuevo. En la Figura 6.9(c) vemos la manera en que funciona esto, suponiendo que la segunda vez no se pierden TPDUs y que todas se entregan correctamente y a tiempo.

El último escenario, la Figura 6.9(d), es el mismo que en la Figura 6.9(c), excepto que ahora suponemos que todos los intentos repetidos de retransmitir la DR también fallan debido a la pérdida de TPDUs. Tras N reintentos, el emisor simplemente se da por vencido y libera la conexión. Mientras tanto, expira el temporizador del receptor y también se sale.

Aunque este protocolo generalmente es suficiente, en teoría puede fallar si se pierden la DR inicial y N retransmisiones. El emisor se dará por vencido y liberará la conexión, pero el otro lado no sabrá nada sobre los intentos de desconexión y seguirá plenamente activo. Esta situación origina una conexión semiabierta. Pudimos haber evitado este problema no permitiendo que el emisor se diera por vencido tras N reintentos, sino obligándolo a seguir insistiendo hasta recibir una respuesta. Sin embargo, si se permite que expire el temporizador en el otro lado, entonces el emisor continuará eternamente, pues nunca aparecerá una respuesta. Si no permitimos que expire el temporizador en el lado receptor, entonces el protocolo se atora en la Figura 6.9(b).

Otra manera de eliminar las conexiones semiabiertas es tener una regla que diga que, si no ha llegado ninguna TPDU durante cierta cantidad de segundos, se libera automáticamente la conexión. De esta manera, si un lado llega a desconectarse, el otro lado detectará la falta de actividad y también se desconectará. Por supuesto que si se pone en práctica esta regla, es necesario que cada entidad de transporte tenga un temporizador que se detenga y se reinicie con cada envío de una TPDU. Si expira este temporizador, se transmite una TPDU ficticia, simplemente para evitar que el otro lado se desconecte. Por otra parte, si se usa la regla de desconexión automática y se pierden demasiadas TPDU ficticias una tras otra en una conexión que de otro modo estaría en reposo, primero un lado y luego el otro se desconectarán automáticamente.

6.2.4. Control de flujo y almacenamiento en búfer

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

Si el servicio de red es no confiable, el emisor debe almacenar en búfer todas las TDPUs enviadas, igual que en la capa de enlace de datos. Si el servicio de red es confiable, se pueden hacer otros arreglos. El receptor no puede garantizar que se aceptará cada TDPU que llegue, por lo que el emisor tendrá que usar búferes. Además el emisor no puede confiar en la confirmación de recepción de la capa de red porque esto sólo significa que ha llegado la TDPU, no que ha sido aceptada. La cuestión está en determinar el tamaño de los búferes en el receptor. En la Figura 6.10 pueden verse los posibles acercamientos.

El **equilibrio óptimo** entre el almacenamiento en búfer en el origen y en el destino **depende del tipo de tráfico** transportado por la conexión:

- Para un tráfico en ráfagas de bajo ancho de banda, es mejor mantener búferes en el emisor.
- Para tráfico continuo de alto ancho de banda, es mejor hacerlo en el receptor.

A medida que se abren y cierran conexiones, y a medida que cambia el patrón del tráfico, el emisor y el receptor necesitan ajustar dinámicamente sus asignaciones de búferes. En consecuencia, el protocolo de transporte debe permitir que un *host* emisor solicite espacio en búfer en el otro extremo. Además de necesitar una ventana de tamaño variable, se usa el siguiente procedimiento de **asignación dinámica de búferes**:

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

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

Por otra parte, si llegase un momento en que el espacio de búfer dejase de limitar el flujo máximo, se necesitará un mecanismo basado en la capacidad de carga de la subred, como por ejemplo, de ventana corrediza cuyo tamaño de ventana emisora depende de este parámetro.

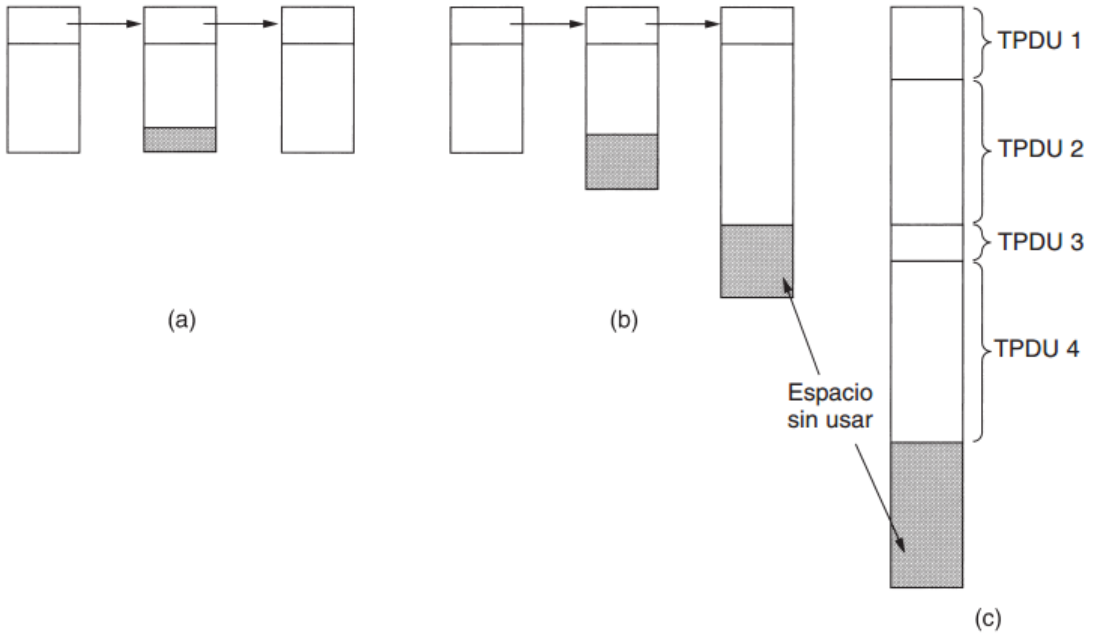

Figura 6.10: (a) Búferes encadenados de **tamaño fijo**: ideal al tener TDPUs del mismo tamaño, ineficiente en otro caso. (b) Búferes encadenados de **tamaño variable**: mejor uso de memoria a coste de mayor complejidad de administración. (c) Un gran búfer **circular** por conexión: eficiente solo en carga alta.

6.2.5. Multiplexión

En la capa de transporte puede surgir la necesidad de multiplexión por varias razones. Por ejemplo, si en un *host* sólo se dispone de una dirección de red, todas las conexiones de transporte de esa máquina tendrán que utilizarla. Cuando llega una TDPU, se necesita algún mecanismo para saber a cuál proceso asignarla. Esta situación, conocida como **multiplexión hacia arriba**, se muestra en la Figura 6.11(a). En esta figura, cuatro distintas conexiones de transporte utilizan la misma conexión de red (por ejemplo, dirección IP) al *host* remoto.

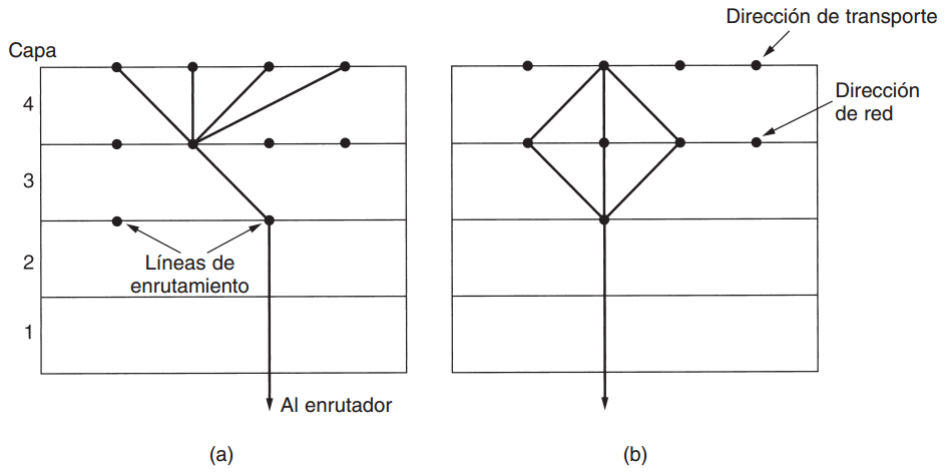

Figura 6.11: (a) Multiplexión hacia arriba. (b) Multiplexión hacia abajo.

La multiplexión también puede ser útil en la capa de transporte por otra razón. Por ejemplo, supongamos que una subred utiliza circuitos virtuales de manera interna y que impone una tasa máxima de datos a cada uno. Si un usuario necesita más ancho de banda del que le puede proporcionar un circuito virtual, una alternativa es abrir múltiples conexiones de red y distribuir el tráfico entre ellas en *round robin* (de manera circular), como se muestra en la Figura 6.11(b). Esto se denomina **multiplexión hacia abajo**. Con k conexiones de red abiertas, el ancho de banda efectivo se incrementa por un factor de k.

6.2.6. Recuperación de caídas

Si los *hosts* y los enrutadores están sujetos a caídas, la recuperación de éstas se vuelve un tema importante. Si la entidad de transporte está por entero dentro de los *hosts*, la recuperación de caídas de la red y de enrutadores es sencilla. Si la capa de red proporciona servicio de datagramas, las entidades de transporte esperan la pérdida de algunas TPDUs todo el tiempo, y saben cómo manejarla. Si la capa de red proporciona servicio orientado a la conexión, entonces la pérdida de un circuito virtual se maneja estableciendo uno nuevo y sondeando la entidad de transporte remota para saber cuáles TPDUs ha recibido y cuáles no. Estas últimas pueden retransmitirse.

Un problema más complicado es la manera de recuperarse de caídas del *host*. En particular, podría ser deseable que los clientes sean capaces de continuar trabajando cuando los servidores caen y se reinician rápidamente.

Sin importar cómo se programen el emisor y el receptor, siempre habrá situaciones en las que el protocolo no podrá recuperarse correctamente. El servidor puede programarse de una de dos maneras: mandar confirmación de recepción primero o escribir primero. El cliente puede programarse de cuatro maneras: siempre retransmitir la última TPDU, nunca retransmitir la última TPDU, retransmitir sólo en el estado S0 (ninguna TPDU pendiente de transmisión) o retransmitir sólo en el estado S1 (pendiente de transmisión sólo una TPDU). Esto da ocho combinaciones pero para cada combinación existe algún grupo de eventos que hacen fallar al protocolo.

En términos más generales, puede plantearse que la recuperación de una caída de capa N sólo puede hacerla la capa $N+1$, y sólo si la capa superior retiene suficiente información del estado. La capa de transporte puede recuperarse de fallas de la capa de red, siempre y cuando cada extremo de una conexión lleve el registro de dónde está.

6.4. LOS PROTOCOLOS DE TRANSPORTE DE INTERNET: UDP

Internet tiene dos protocolos principales en la capa de transporte, uno orientado y otro no orientado a la conexión. El protocolo no orientado a la conexión es **UDP**. El protocolo orientado a la conexión es **TCP**. UDP es básicamente IP con un encabezado corto.

6.4.1. Introducción a UDP

El conjunto de protocolos de Internet soporta un protocolo de transporte no orientado a la conexión: el **Protocolo de Datagramas de Usuario** (UDP: *User Datagram Protocol*). Este protocolo proporciona una forma para que las aplicaciones envíen datagramas IP encapsulados sin tener que establecer una conexión. UDP se describe en el `RFC 768`.

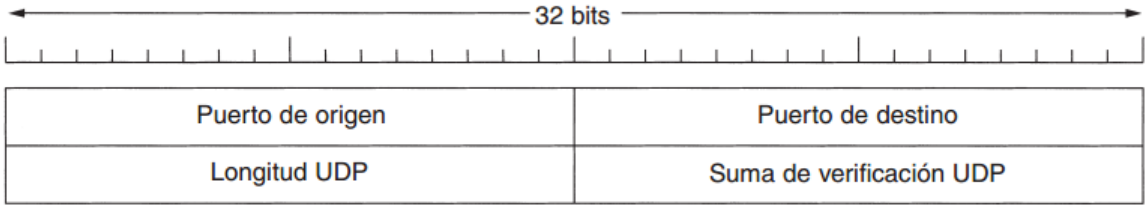

Figura 6.12: El encabezado UDP.

UDP transmite segmentos que consisten en un encabezado de 8 bytes seguido por la carga útil. En la Figura 6.12 se muestra tal encabezado. Los dos puertos sirven para identificar los puntos terminales dentro de las máquinas de origen y destino. Cuando llega un paquete UDP, su carga útil se entrega al proceso que está enlazado al puerto de destino. Este enlace ocurre cuando se utiliza la primitiva `BIND` o algo similar. El valor principal de contar con UDP en lugar de simplemente utilizar el IP puro es la adición de los puertos de origen y destino. Sin los campos de puerto, la capa de transporte no sabría qué hacer con el paquete. Con ellos, entrega los segmentos de manera correcta.

El encabezado UDP contiene los siguientes campos:

- **Puerto de origen**, el cual es puerto utilizado por el *host* emisor.
- **Puerto de destino**, el cual es el puerto utilizado por el *host* receptor.
- **Longitud UDP**, el cual incluye el encabezado de 8 bytes y los datos.
- **Suma de verificación UDP**: el cual es un campo opcional y se almacena 0 si no se utiliza (por ejemplo en aplicaciones

de digitalización de voz).

UDP explícitamente no realiza control de flujo, control de errores o retransmisión cuando se recibe un segmento erróneo. Todo lo anterior le corresponde a los procesos de usuario. Lo que sí realiza es proporcionar una interfaz al protocolo IP con la característica agregada de des-multiplexar varios procesos utilizando los puertos. Para aplicaciones que necesitan tener control preciso sobre el flujo de paquetes, control de errores o temporización, UDP es lo ideal. También es útil en las situaciones cliente-servidor. Con frecuencia, el cliente envía una solicitud corta al servidor y espera una respuesta corta. Si se pierde la solicitud o la respuesta, el cliente simplemente puede terminar y probar nuevamente. El código no sólo es simple, sino que se necesitan muy pocos mensajes (uno en cada dirección) en comparación con un protocolo que requiere una configuración inicial. Una aplicación que utiliza de esta manera a UDP es **Sistema de Nombres de Dominio** (**DNS**: *Domain Name System*): un programa que necesita buscar la dirección IP de algún *host*, por ejemplo, www.google.com, puede enviar al servidor DNS un paquete UDP que contenga el nombre de dicho *host*. El servidor responde con un paquete UDP que contiene la dirección IP del *host*. No se necesita configuración por adelantado ni tampoco liberación posterior. Sólo dos mensajes viajan a través de la red.

6.5. LOS PROTOCOLOS DE TRANSPORTE DE INTERNET: TCP

La mayoría de las aplicaciones de Internet se necesita una entrega en secuencia confiable. UDP no puede proporcionar esto, por lo que se necesita otro protocolo. Se llama **TCP**.

6.5.1. Introducción a TCP

TCP (Protocolo de Control de Transmisión) se diseño específicamente para proporcionar un flujo de bytes confiable de extremo a extremo a través de una interred no confiable. Una interred difiere de una sola red debido a que diversas partes podrían tener diferentes topologías, anchos de banda, retardos, tamaños de paquete y otros parámetros. TCP tiene un diseño que se adapta de manera dinámica a las propiedades de la interred y que se sobrepone a muchos tipos de fallas. TCP se definió formalmente en el `RFC 793`.

Cada máquina que soporta TCP tiene una entidad de transporte TCP, ya sea un procedimiento de biblioteca, un proceso de usuario o parte del kernel. En todos los casos, maneja flujos TCP e interactúa con la capa IP. Una entidad TCP acepta flujos de datos de usuario de procesos locales, los divide en fragmentos que no excedan los 64 KB (en la práctica, por lo general, 1460 bytes de datos que se ajustan en una sola trama Ethernet con los encabezados IP y TCP), y envía cada fragmento como un datagrama IP independiente. Cuando los datagramas que contienen datos TCP llegan a una máquina, se pasan a la entidad TCP, la cual reconstruye los flujos de bytes originales.

La capa IP no proporciona ninguna garantía de que los datagramas se entregarán de manera apropiada, por lo que corresponde a TCP terminar los temporizadores y retransmitir los datagramas conforme sea necesario. Los datagramas que llegan podrían hacerlo en el orden incorrecto; también corresponde a TCP re-ensamblarlos en mensajes en la secuencia apropiada. En resumen, TCP debe proporcionar la confiabilidad.

6.5.2. El modelo del servicio TCP

El servicio TCP se obtiene al hacer que tanto el servidor como el cliente creen puntos terminales, llamados *sockets*. Cada *socket* tiene un número (dirección), que consiste en la dirección IP del *host*, y un número de 16 bits, que es local a ese *host*, llamado puerto. Un puerto es el nombre TCP para un TSAP. Para obtener el servicio TCP, se debe establecer de manera explícita una conexión entre un *socket* en la máquina emisora y uno en la máquina receptora.

Un *socket* puede utilizarse para múltiples conexiones al mismo tiempo. En otras palabras, dos o más conexiones pueden terminar en el mismo *socket*. Las conexiones se identifican mediante los identificadores de *socket* de los dos extremos, es decir (*socket1*, *socket2*). No se utiliza ningún otro número de circuitos virtuales ni identificador.

Los números de puerto menores que 1024 se llaman **puertos bien conocidos** y se reservan para servicios estándar. Los puertos bien conocidos se especifican en www.iana.org pero cada *host* puede asignar los demás según sea necesario. La dirección de un puerto más la dirección IP de su *host* forman un punto terminal único de 48 bits.

Por ejemplo, cualquier proceso que desee establecer una conexión a un *host* para transferir un archivo utilizando FTP puede conectarse con el puerto 21 del host de destino para conectar su demonio (*daemon*) FTP. Lo que se hace generalmente es que un solo demonio, llamado `inetd` (demonio de Internet) en UNIX, se conecte a sí mismo a múltiples puertos y esperar la primera conexión entrante. Cuando eso ocurre, `inetd` bifurca un nuevo proceso y ejecuta el demonio apropiado en él, dejando

que ese demonio maneje la solicitud. De esta forma, los demonios distintos a `inetd` sólo están activos cuando hay trabajo para ellos. `inetd` consulta un archivo de configuración para saber cuál puerto utilizar. En consecuencia, el administrador del sistema puede configurar el sistema para tener demonios permanentes en los puertos más ocupados (por ejemplo, el puerto 80) e `inetd` en los demás.

Algunas características más de TCP es que todas las conexiones son de dúplex total (*full-duplex*) y de punto a punto. Además TCP no soporta la multidifusión ni la difusión. Cuando una aplicación pasa datos a TCP, éste decide si los envía inmediatamente o los almacena en el búfer. Sin embargo, algunas veces, la aplicación realmente necesita que los datos se envíen de inmediato. Para obtener los datos, las aplicaciones pueden utilizar el indicador `PUSH`, que es una señal para TCP de que no debe retrasar la transmisión.

Una última característica del servicio TCP son los **datos urgentes**. Cuando un usuario interactivo oprime las teclas `Supr` o `Ctrl+C` para interrumpir una operación remota que ha iniciado, la aplicación emisora coloca información de control en el flujo de datos y se la da a TCP junto con el indicador `URGENT`. Este evento ocasiona que TCP interrumpa el encolamiento de datos y transmita inmediatamente todo lo que tenga para esa conexión. Cuando el destino recibe los datos urgentes, se interrumpe la aplicación receptora a fin de que pueda detener lo que esté haciendo y que lea el flujo de datos para encontrar los datos urgentes. El final de los datos urgentes se marca para que la aplicación sepa cuándo terminan.

6.5.3. El protocolo TCP

Una característica clave de TCP es que cada byte de una conexión TCP tiene su propio número de secuencia de 32 bits. Éstos se utilizan para confirmaciones de recepción y para el mecanismo de ventana.

La entidad TCP emisora y la receptora intercambian datos en forma de segmentos. Un **segmento** consiste en un encabezado TCP fijo de 20 bytes (más una parte opcional) seguido de cero o más bytes de datos. El software de TCP decide el tamaño de los segmentos; puede acumular datos de varias escrituras para formar un segmento, o dividir los datos de una escritura en varios segmentos. Hay dos límites que restringen el tamaño de segmento. Primero, cada segmento, incluido el encabezado TCP, debe caber en la carga útil de 65,515 bytes del IP. Segundo, cada red tiene una **unidad máxima de transferencia (MTU)** y cada segmento debe caber en la MTU. En la práctica, la MTU es, generalmente, de 1500 bytes (el tamaño de la carga útil en Ethernet) y, por tanto, define el límite superior del tamaño de segmento.

El protocolo básico usado por las entidades TCP es el protocolo de ventana corrediza. Cuando un transmisor envía un segmento, también inicia un temporizador. Cuando llega el segmento al destino, la entidad TCP receptora devuelve un segmento (con datos, si existen, de otro modo sin ellos) que contiene un número de confirmación de recepción igual al siguiente número de secuencia que espera recibir. Si el temporizador del emisor expira antes de la recepción de la confirmación, el emisor envía de nuevo el segmento.

Pueden llegar segmentos fuera de orden, por ejemplo los bytes `3072-4095` podrían llegar pero no enviarse confirmación de recepción porque los bytes `2048-3071` no han aparecido aún. También pueden retardarse segmentos en tránsito durante tanto tiempo que el temporizador del emisor expira y los segmentos se retransmiten. Las retransmisiones podrían incluir rangos de bytes diferentes a los de la transmisión original, lo cual requiere una administración cuidadosa para llevar el control de los bytes que se han recibido correctamente en un momento determinado. Sin embargo, esto es factible ya que cada byte del flujo tiene su propio desplazamiento único.

6.5.4. El encabezado del segmento TCP

Cada segmento comienza con un encabezado de formato fijo de 20 bytes. El encabezado fijo puede ir seguido de opciones de encabezado. Tras las opciones, si las hay, pueden continuar hasta $65535 - 20 - 20 = 65495$ bytes de datos, donde los primeros 20 se refieren al encabezado IP y los segundos al encabezado TCP. Los segmentos sin datos son legales y se usan por lo común para confirmaciones de recepción y mensajes de control.

El control de flujo en el TCP se maneja usando una ventana corrediza de tamaño variable. Es válido un campo `Tamaño de ventana de = 0`, e indica que se han recibido los bytes hasta `Número de confirmación de recepción = -1`, inclusive, pero que el receptor actualmente necesita un descanso y quisiera no recibir más datos por el momento. El permiso para enviar puede otorgarse después enviando un segmento con el mismo `Número de confirmación de recepción` y un campo `Tamaño de ventana` distinto de cero.

En TCP, las confirmaciones de recepción y los permisos para enviar datos adicionales son completamente independientes. En efecto, un receptor puede decir: “He recibido bytes hasta k, pero por ahora no deseo más”. Esta independencia (de hecho, una ventana de tamaño variable) da flexibilidad adicional.

Campo	Longitud	Descripción
Puerto de origen	16 bits	Terminal emisor de la conexión.
Puerto de destino	16 bits	Terminal receptor de la conexión.
Número de secuencia	32 bits	Identificador asignado al segmento actual en la conexión.
Nro. de confirmación de recepción	32 bits	Especifica el siguiente byte esperado, no el último byte correctamente recibido.
Longitud del encabezado TCP	4 bits	Indica la cantidad de palabras de 32 bits contenidas en el encabezado TCP. Esta información es necesaria porque el campo de Opciones es de longitud variable, por lo que el encabezado también.
Campo sin uso	6 bits	- - - - -
URG	1 bit	Si se establece en 1 se está en uso el *apuntador urgente*.
ACK	1 bit	se establece en 1 para indicar que el Número de confirmación de recepción es válido. Si el ACK es 0, el segmento no contiene una confirmación de recepción, por lo que se ignora el campo de Número de confirmación de recepción.
PSH	1 bit	Indica datos que se deben transmitir de inmediato. Por este medio se solicita atentamente al receptor que entregue los datos a la aplicación a su llegada y no los almacene en búfer hasta la recepción de un búfer completo.
RST	1 bit	Se usa para restablecer una conexión que se ha confundido debido a una caída de *host* u otra razón; también sirve para rechazar un segmento no válido o un intento de abrir una conexión. Por lo general, si usted recibe un segmento con el bit RST encendido, tiene un problema entre manos.
SYN	1 bit	Se usa para establecer conexiones. La solicitud de conexión tiene `SYN = 1` y `ACK = 0` para indicar que el campo de confirmación de recepción incorporado no está en uso. La respuesta de conexión sí lleva una confirmación de recepción, por lo que tiene `SYN = 1` y `ACK = 1`. En esencia, el bit `SYN` se usa para denotar `CONNECTION REQUEST` y `CONNECTION ACCEPTED`, y el bit `ACK` sirve para distinguir entre ambas posibilidades.
FIN	1 bit	Se usa para liberar una conexión; especifica que el emisor no tiene más datos que transmitir. Sin embargo, tras cerrar una conexión, un proceso puede continuar recibiendo datos indefinidamente. Ambos segmentos, `SYN` y `FIN`, tienen números de secuencia y, por tanto, tienen la garantía de procesarse en el orden correcto.
Tamaño de ventana	16 bits	Indica la cantidad de bytes que pueden enviarse comenzando por el byte cuya recepción se ha confirmado.
Suma de verificación	16 bits	Es una suma de verificación del encabezado, los datos y el pseudo-encabezado conceptual cuya recepción se ha confirmado.
Apuntador urgente	16 bits	Sirve para indicar un desplazamiento en bytes a partir del número actual de secuencia en el que se encuentran datos urgentes. Este recurso sustituye los mensajes de interrupción. Este recurso es un mecanismo rudimentario para permitir que el emisor envíe una señal al receptor sin implicar al TCP en la razón de la interrupción.
Opciones	0 o más palabras de 32 bits	Ofrece una forma de agregar características extra no cubiertas por el encabezado normal.
Datos	- - - - -	Se utiliza para asegurarse que la cabecera acaba con un tamaño múltiplo de 32 bits.

Cuadro 6.2: Campos del segmento TCP.

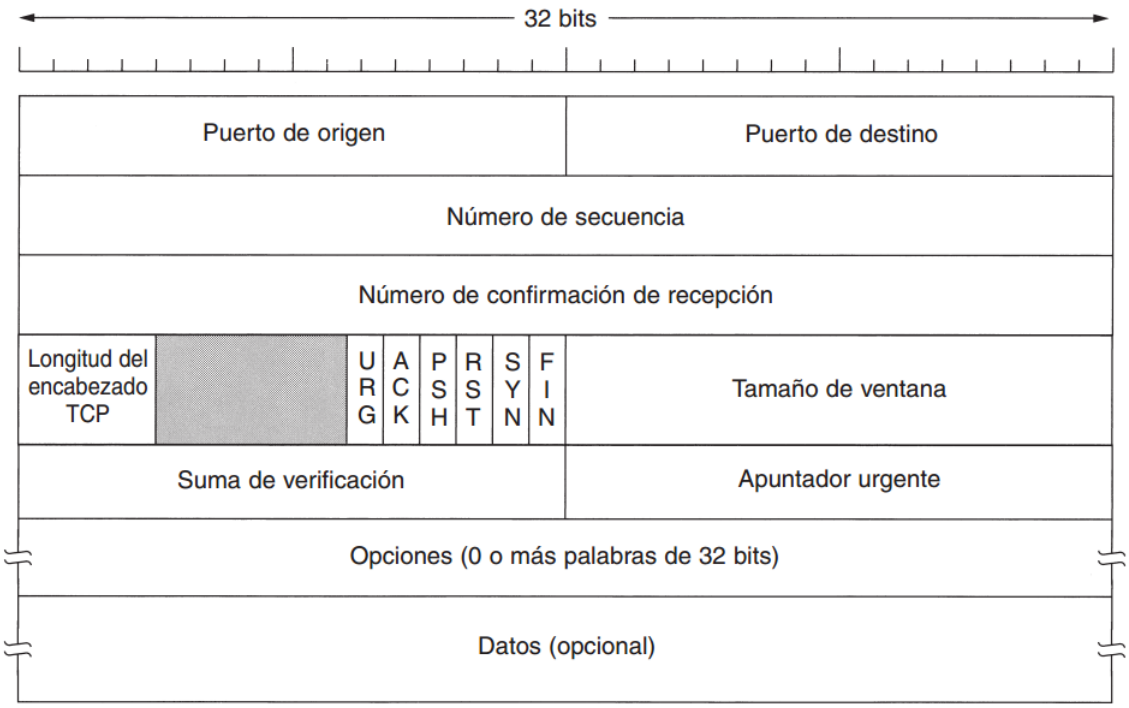

Figura 6.13: Encabezado TCP.

Al realizar el *checksum*, se establece el campo de `Suma de verificación` del TCP en cero, y se rellena el campo de datos con un byte cero adicional si la longitud es un número impar. El algoritmo de suma de verificación simplemente suma todas las palabras de 16 bits en complemento a 1 y luego obtiene el complemento a 1 de la suma. Como consecuencia, al realizar el cálculo el receptor con el segmento completo, incluido el campo de Suma de verificación, el resultado debe ser cero.

El pseudo-encabezado (ver Figura 6.14) contiene las direcciones IP de 32 bits de las máquinas de origen y de destino, el número de protocolo de TCP (6), y la cuenta de bytes del segmento TCP (incluido el encabezado). La inclusión del pseudo-encabezado en el cálculo de la suma de verificación TCP ayuda a detectar paquetes mal entregados, pero hacerlo viola la jerarquía de protocolos puesto que las direcciones de IP que contiene pertenecen a la capa IP, no a la capa TCP. UDP utiliza el mismo pseudo-encabezado para su suma de verificación.

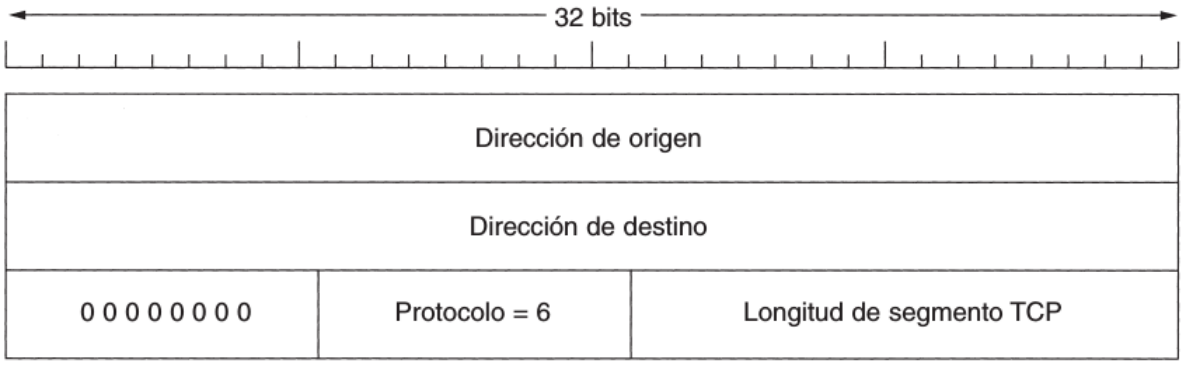

Figura 6.14: Pseudo-encabezado incluido en la suma de verificación del TCP.

El campo Opciones ofrece una forma de agregar características extra no cubiertas por el encabezado normal. La opción más importante es la que permite que cada *host* especifique la carga útil TCP máxima que está dispuesto a aceptar. El uso de segmentos grandes es más eficiente que el de segmentos pequeños, puesto que el encabezado de 20 bytes puede entonces amortizarse entre más datos, pero los hosts pequeños tal vez no puedan manejar segmentos muy grandes. Durante el establecimiento de la conexión, cada lado puede anunciar su máximo y ver el de su compañero. Si un *host* no usa esta opción, tiene una carga útil predeterminada de 536 bytes. Se requiere que todos los *hosts* de Internet acepten segmentos TCP de $536 + 20 = 556$ bytes. No es necesario que el tamaño máximo de segmento en ambas direcciones sea el mismo.

En las líneas con alto ancho de banda, alto retardo o ambas cosas, un tamaño de ventana más grande permitirá al emisor continuar enviando datos, pero como el campo de `Tamaño de ventana` es de 16 bits, es imposible expresar tal tamaño. Se

propusieron algunas mejoras entre ellas:

- `RFC 1323`: una opción de escala de ventana que permite al emisor y al receptor negociar un factor de escala de ventana. Este número da la posibilidad de que ambos lados desplacen el campo de Tamaño de ventana hasta 14 bits a la izquierda, permitiendo por tanto ventanas de hasta 2^{30} bytes.
- `RFC 1106`: repetición selectiva en lugar del protocolo de retroceso n. Si el receptor recibe un segmento malo y luego una gran cantidad de segmentos buenos, el temporizador del protocolo TCP normal expirará en algún momento y se retransmitirán todos los segmentos sin confirmación de recepción, incluidos los que se recibieron correctamente. El `RFC 1106` introdujo los **NAKs**, para permitir que el receptor solicite un segmento (o segmentos) específico. Tras recibirlo, puede enviar una confirmación de recepción de todos los datos que tiene en búfer, reduciendo de esta manera la cantidad de datos retransmitidos.

6.5.5. Establecimiento de una conexión TCP

En el TCP las conexiones se establecen usando el acuerdo de tres vías. Para establecer una conexión, un lado, digamos el servidor, espera pasivamente una conexión entrante ejecutando las primitivas `LISTEN` y `ACCEPT` y especificando cierto origen o bien nadie en particular.

El otro lado, digamos el cliente, ejecuta una primitiva `CONNECT` especificando la dirección y el puerto IP con el que se desea conectar, el tamaño máximo de segmento TCP que está dispuesto a aceptar y opcionalmente algunos datos de usuario (por ejemplo, una contraseña). La primitiva `CONNECT` envía un segmento TCP con el bit `SYN` encendido y el bit `ACK` apagado, y espera una respuesta.

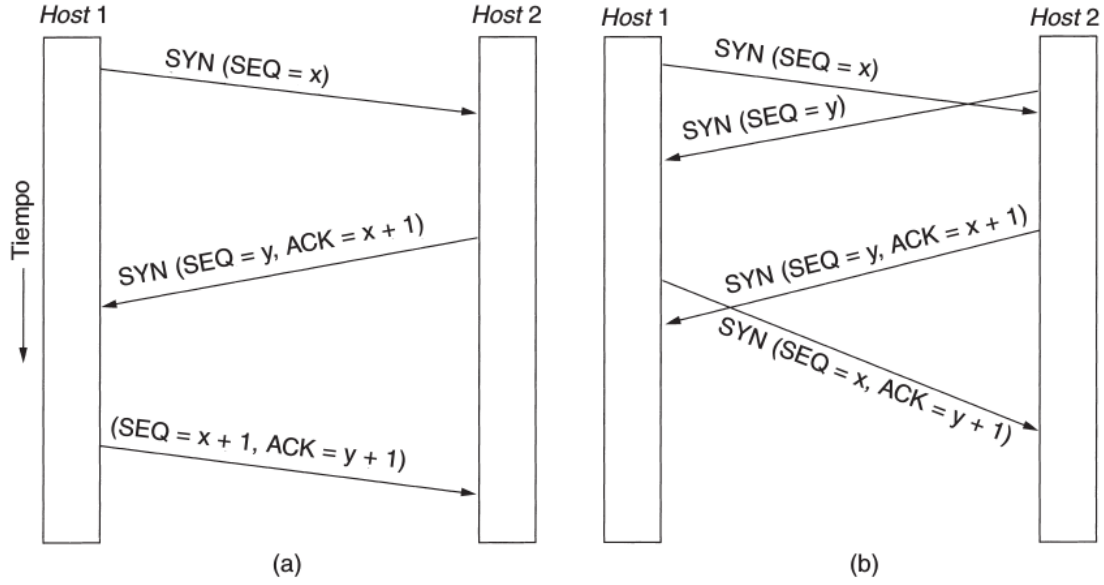

Figura 6.15: (a) Establecimiento de una conexión TCP en el caso normal. (b) Colisión de llamadas.

Al llegar el segmento al destino, la entidad TCP ahí revisa si hay un proceso que haya ejecutado un `LISTEN` en el puerto indicado en el campo de `Puerto de destino`. Si no lo hay, envía una respuesta con el bit `RST` encendido para rechazar la conexión.

Si algún proceso está escuchando en el puerto, ese proceso recibe el segmento TCP entrante y puede entonces aceptar o rechazar la conexión; si la acepta, se devuelve un segmento de confirmación de recepción.

En el caso en que dos *hosts* intentan simultáneamente establecer una conexión entre los mismos dos *sockets*, la secuencia de eventos es la que se ilustra en la Figura 6.15(b). El resultado de estos eventos es que sólo se establece una conexión, no dos, pues las conexiones se identifican por sus puntos terminales. Si el primer establecimiento resulta en una conexión identificada por `(x,y)`, al igual que en el segundo, sólo se hace una entrada de tabla, es decir, de `(x,y)`.

Para el número de secuencia inicial de una conexión se usa un esquema basado en reloj, con un pulso de reloj cada 4 μseg. Por seguridad adicional, al caerse un *host*, no podrá reiniciarse durante el tiempo máximo de paquete (120 seg) para asegurar que no haya paquetes de conexiones previas vagando por Internet.

6.5.6. Liberación de una conexión TCP

Aunque las conexiones TCP son dúplex total, para entender la manera en que se liberan las conexiones es mejor visualizarlas como un par de conexiones símplex. Cada conexión símplex se libera independientemente de su igual. Para liberar una conexión, cualquiera de las partes puede enviar un segmento TCP con el bit `FIN` establecido, lo que significa que no tiene más datos por transmitir. Al confirmarse la recepción del `FIN`, ese sentido se apaga. Sin embargo, puede continuar un flujo de datos indefinido en el otro sentido. Cuando ambos sentidos se han apagado, se libera la conexión. Normalmente se requieren cuatro segmentos TCP para liberar una conexión, un `FIN` y un `ACK` para cada sentido. Sin embargo, es posible que el primer `ACK` y el segundo `FIN` estén contenidos en el mismo segmento, reduciendo la cuenta total a tres.

Al igual que con las llamadas telefónicas en las que ambas partes dicen adiós y cuelgan el teléfono simultáneamente, ambos extremos de una conexión TCP pueden enviar segmentos `FIN` al mismo tiempo. La recepción de ambos se confirma de la manera normal, y se apaga la conexión. De hecho, en esencia no hay diferencia entre la liberación secuencial o simultánea por parte de los *hosts*.

Para evitar el problema de los dos ejércitos, se usan temporizadores. Si no llega una respuesta a un `FIN` en un máximo de dos tiempos de vida de paquete, el emisor del `FIN` libera la conexión. Tarde o temprano el otro lado notará que, al parecer, ya nadie lo está escuchando, y también expirará su temporizador. Aunque esta solución no es perfecta, en la práctica, pocas veces ocurren problemas.

6.5.7. Modelado de administración de conexiones TCP

Los pasos requeridos para establecer y liberar conexiones pueden representarse en una máquina de estados finitos con los 11 estados listados en la Figura 6.16. En cada estado son legales ciertos eventos. Al ocurrir un evento legal, debe emprenderse alguna acción. Si ocurren otros eventos, se informa un error.

Estado	Descripción
CLOSED	No hay conexión activa ni pendiente
LISTEN	El servidor espera una llamada
SYN RCVD	Llegó solicitud de conexión; espera ACK
SYN SENT	La aplicación comenzó a abrir una conexión
ESTABLISHED	Estado normal de transferencia de datos
FIN WAIT 1	La aplicación dijo que ya terminó
FIN WAIT 2	El otro lado acordó liberar
TIMED WAIT	Espera que todos los paquetes mueran
CLOSING	Ambos lados intentaron cerrar simultáneamente
CLOSE WAIT	El otro lado inició una liberación
LAST ACK	Espera que todos los paquetes mueran

Figura 6.16: Estados usados en la máquina de estados finitos de administración de conexiones TCP.

Cada conexión comienza en el estado `CLOSED` (cerrado) y deja ese estado cuando hace una apertura pasiva (`LISTEN`), o una apertura activa (`CONNECT`). Si el otro lado realiza la acción opuesta, se establece una conexión y el estado se vuelve `ESTABLISHED`. La liberación de la conexión puede iniciarse desde cualquiera de los dos lados. Al completarse, el estado regresa a `CLOSED`.

Al emitir una solicitud `CONNECT` una aplicación de la máquina cliente, la entidad TCP local crea un registro de conexión, lo marca para indicar que está en el estado `SYN SENT`, y envía un segmento `SYN`. Muchas conexiones pueden estar abiertas (o en proceso de apertura) al mismo tiempo como parte de varias aplicaciones, por lo que el estado es por conexión y se asienta en el registro de conexiones. Al llegar el `SYN + ACK`, el TCP envía al `ACK` final del acuerdo de tres vías y se conmuta al estado `ESTABLISHED`. Ahora pueden enviarse y recibirse datos.

Al terminar una aplicación, ejecuta una primitiva `CLOSE`, que causa que la entidad TCP local envíe un segmento `FIN` y espere el `ACK` correspondiente. Al llegar el `ACK`, se hace una transición al estado `FIN WAIT 2`, y ya está cerrado un sentido de la conexión. Cuando también cierra el otro lado, llega un `FIN`, para el cual se envía una confirmación de recepción. Ahora ambos

lados están cerrados, pero el TCP espera un tiempo igual al tiempo de vida máximo del paquete para garantizar que todos los paquetes de la conexión han sido eliminados, como protección en caso de la pérdida de una confirmación de recepción. Al expirar el temporizador, el TCP borra el registro de la conexión.

Desde el punto de vista del servidor. El servidor hace un `LISTEN` y se detiene a esperar la aparición de alguien. Al llegar un `SYN`, se envía una confirmación de recepción y el servidor pasa al estado `SYN RCVD`. Cuando llega la confirmación de recepción del `SYN` del servidor, el acuerdo de tres vías se ha completado y el servidor regresa al estado `ESTABLISHED`. Ahora puede ocurrir la transferencia de datos.

Cuando el cliente ha tenido suficiente, hace un `CLOSE`, que causa la llegada de un `FIN` al servidor. Entonces se envía una señal al servidor. Cuando éste también hace un `CLOSE`, se envía un `FIN` al cliente. Al llegar la confirmación de recepción del cliente, el servidor libera la conexión y elimina el registro de conexión.

6.5.8. Política de transmisión del TCP

La administración de ventanas en el TCP no está vinculada directamente a las confirmaciones de recepción como en la mayoría de los protocolos de enlace de datos. Por ejemplo, suponga que el receptor tiene un búfer de 4096 bytes, como se muestra en la Figura 6.17. Si el emisor envía un segmento de 2048 bytes que se recibe correctamente, el receptor enviará la confirmación de recepción del segmento. Sin embargo, dado que ahora sólo tiene 2048 bytes de espacio de búfer (hasta que la aplicación retire algunos datos de éste), anunciará una ventana de 2048 comenzando con el siguiente byte esperado.

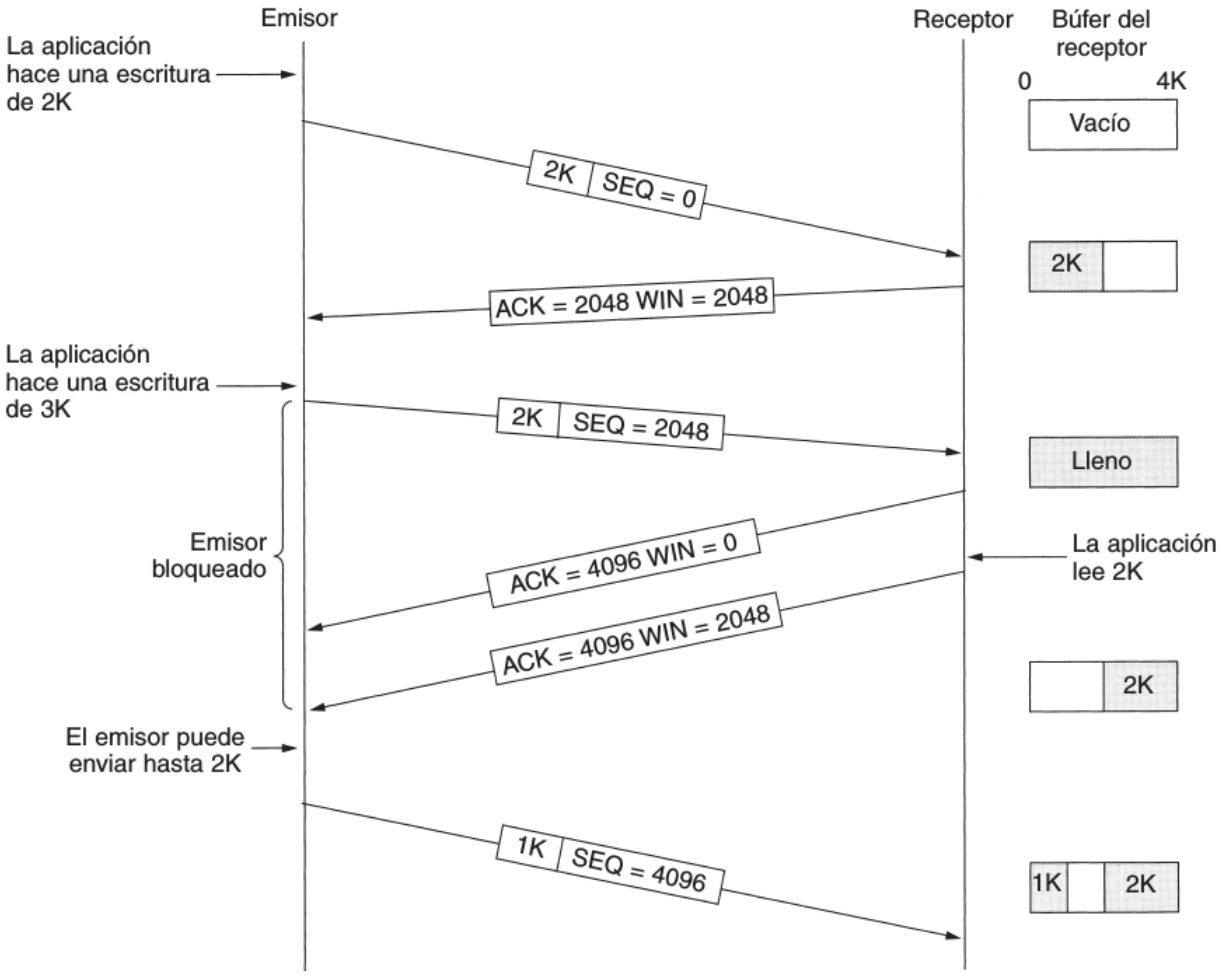

Figura 6.17: Administración de ventanas en TCP.

Ahora el emisor envía otros 2048 bytes, para los cuales el receptor envía la confirmación de recepción, pero la ventana anunciada es de 0. El emisor debe detenerse hasta que el proceso de aplicación del *host* receptor retire algunos datos del búfer, en cuyo momento el TCP puede anunciar una ventana más grande.

Cuando la ventana es de 0, el emisor normalmente no puede enviar segmentos, salvo en dos situaciones. Primera, pueden enviarse datos urgentes (por ejemplo, para permitir que el usuario elimine el proceso en ejecución en la máquina remota).

Segunda, el emisor puede enviar un segmento de 1 byte para hacer que el receptor reanuncie el siguiente byte esperado y el tamaño de la ventana. El estándar TCP proporciona explícitamente esta opción para evitar un bloqueo irreversible si llega a perderse un anuncio de ventana.

No se requiere que los emisores envíen los datos tan pronto reciba desde la aplicación, ni que los receptores se los pasen a la aplicación y envíen confirmación tan pronto estos lleguen. Estas libertades pueden explotarse para mejorar el desempeño del protocolo, por ejemplo:

- Retardando las confirmaciones de recepción y las actualizaciones de ventana durante 500 mseg con la esperanza de que lleguen algunos datos con los cuales viajar gratuitamente, recortando a la mitad la cuenta de paquetes y el uso de ancho de banda.
- **Algoritmo de Nagle**: al llegar al emisor datos 1 byte a la vez, se envía el primer dato y se almacena en búfer el resto hasta que llegue la confirmación de recepción del primero. Luego se envían todos juntos los caracteres almacenados en búfer y se vuelven a almacenar los que llegan de la aplicación. El algoritmo envía el segmento si han entrado suficientes datos para llenar la mitad de la ventana o la totalidad de un segmento.
- **Solución de Clark**: consiste en evitar que el emisor envié la actualización de la ventana hasta que no tenga la mitad del búfer vacío o hasta que no pueda manejar el tamaño de segmento máximo declarado, lo que sea mas pequeño. Esto evita el **problema de la ventana tonta**, que ocurre cuando la aplicación del lado receptor lee de a un byte a la vez.
- El receptor puede bloquear la primitiva `READ` hasta que haya suficientes datos en búfer. De esta forma se evita la sobrecarga en la transferencia de archivos por ejemplo.
- Otro problema del receptor es qué debe hacer con los segmentos fuera de orden. Pueden conservarse o descartarse, al albedrío del receptor. Por supuesto, las confirmaciones de recepción pueden enviarse sólo después de haber recibido todos los datos hasta el byte confirmado. Si el receptor recibe los segmentos 0, 1, 2, 4, 5, 6 y 7, puede enviar una confirmación de recepción de todos los bytes hasta el último byte del segmento 2, inclusive. Al expirar el temporizador del emisor, retransmitirá el segmento 3. Si el receptor tienen en búfer los segmentos 4 a 7, al recibir el segmento 3 puede enviar una confirmación de recepción de todos los bytes hasta el final del segmento 7.

6.5.9. Control de congestión en TCP

El primer paso del manejo de la congestión es su detección. Hoy día, la pérdida de paquetes por errores de transmisión es relativamente rara debido a que las troncales de larga distancia son de fibra. En consecuencia, la mayoría de las expiraciones de tiempo en Internet se deben a congestión. Todos los algoritmos TCP de Internet suponen que las expiraciones de tiempo son causadas por congestión y las revisan en busca de problemas.

Al establecerse una conexión, se tiene que seleccionar un tamaño de ventana adecuado. El receptor puede especificar una ventana con base en su tamaño de búfer. Si el emisor se ajusta a su tamaño de ventana, no ocurrirán problemas por desbordamiento de búferes en la terminal receptora, pero aún pueden ocurrir debido a congestión interna en la red.

El problema de la congestión se ilustra en la Figura 6.18. Un tubo grueso que conduce a un receptor de poca capacidad. Mientras el emisor no envíe más agua de la que puede contener la cubeta, no se perderá agua. El factor limitante no es la capacidad de la cubeta, sino la capacidad de conducción interna de la red. Si entra demasiada agua a alta velocidad, ésta retrocederá, perdiéndose algo (en este caso, por el desbordamiento del embudo).

La solución de Internet es aceptar que existen dos problemas potenciales (capacidad de la red y capacidad del receptor) y manejarlos por separado. Para ello, cada emisor mantiene dos ventanas: la ventana que ha otorgado el receptor y una segunda ventana, la ventana de congestión. Cada una refleja la cantidad de bytes que puede enviar el emisor. La cantidad de bytes que pueden enviarse es la cifra menor de las dos ventanas. Por tanto, la ventana efectiva es el mínimo de lo que el emisor piensa que es correcto y lo que el receptor piensa que está bien. Si el receptor dice “envía 8 KB” pero el emisor sabe que las ráfagas de más de 4 KB saturan la red, envía 4 KB. Por otra parte, si el receptor dice “envía 8 KB” y el emisor sabe que las ráfagas de hasta 32 KB pueden llegar sin problemas, envía los 8 KB solicitados.

Al establecer una conexión, el emisor asigna a la ventana de congestión el tamaño de segmento máximo usado por la conexión; entonces envía un segmento máximo. Si se recibe la confirmación de recepción de este segmento antes de que expire el temporizador, el emisor agrega el equivalente en bytes de un segmento a la ventana de congestión para hacerla de dos segmentos de tamaño máximo, y envía dos segmentos. A medida que se confirma cada uno de estos segmentos, se aumenta el tamaño de la ventana de congestión en un segmento máximo. Cuando la ventana de congestión es de n segmentos, si de todos los n se reciben confirmaciones de recepción a tiempo, se aumenta el tamaño de la ventana de congestión en la cuenta de bytes correspondiente a n segmentos. De hecho, cada ráfaga confirmada duplica la ventana de congestionamiento.

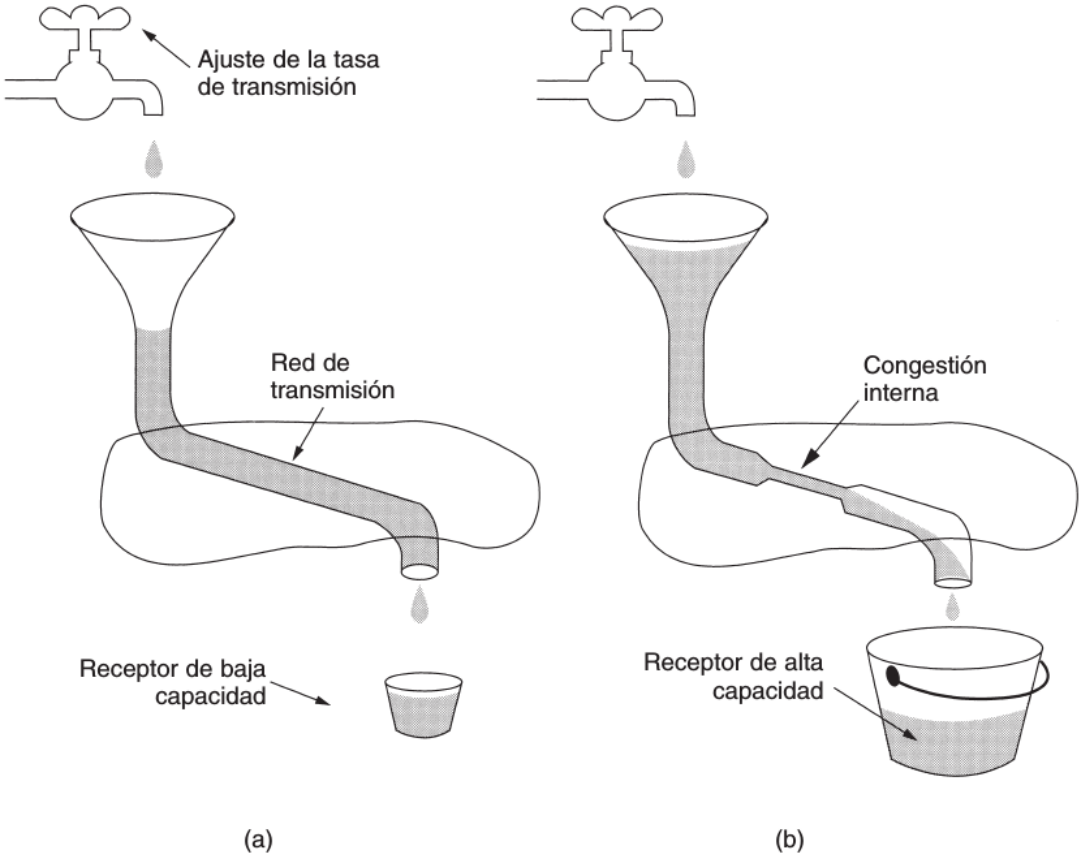

Figura 6.18: (a) Red rápida alimentando un receptor de baja capacidad. (b) Red lenta alimentando un receptor de alta capacidad.

La ventana de congestión sigue creciendo exponencialmente hasta ocurrir una expiración del temporizador o alcanzar el tamaño de la ventana receptora. La idea es que, si las ráfagas de 1024, 2048 y 4096 bytes funcionan bien, pero una ráfaga de 8192 produce una expiración del temporizador, la ventana de congestión debe establecerse en 4096 para evitar la congestión. Mientras el tamaño de la ventana de congestión permanezca en 4096, no se enviará una ráfaga de mayor longitud, sin importar la cantidad de espacio de ventana otorgada por el receptor. Este algoritmo se llama **arranque lento**.

El algoritmo de control de congestión de Internet, el cual usa un tercer parámetro, el **umbral**, inicialmente de 64 KB, además de las ventanas de recepción y congestión. Al ocurrir una expiración del temporizador, se establece el umbral en la mitad de la ventana de congestión actual, y la ventana de congestión se restablece a un segmento máximo. Luego se usa el arranque lento para determinar lo que puede manejar la red, excepto que el crecimiento exponencial termina al alcanzar el umbral. A partir de este punto, las transmisiones exitosas aumentan linealmente la ventana de congestión (en un segmento máximo por ráfaga) en lugar de uno por segmento. En efecto, este algoritmo está suponiendo que probablemente es aceptable recortar la ventana de congestión a la mitad, y luego aumentarla gradualmente a partir de ahí.

Como ilustración de la operación del algoritmo de congestión, véase la Figura 6.19. El tamaño máximo de segmento aquí es de 1024 bytes. Inicialmente, la ventana de congestión era de 64 KB, pero ocurre una expiración del temporizador, así que se establece el umbral en 32KB y la ventana de congestión en 1KB, para la transmisión 0. La ventana de congestión entonces crece exponencialmente hasta alcanzar el umbral (32 KB). A partir de entonces, crece linealmente.

La transmisión 13 tiene mala suerte y ocurre una expiración del temporizador. Se establece el umbral en la mitad de la ventana actual (ahora de 40 KB, por lo que la mitad es de 20 KB), e inicia de nuevo el arranque lento. Al llegar las confirmaciones de recepción de la transmisión 14, los primeros cuatro incrementan la ventana de congestión en un segmento máximo, pero después de eso el crecimiento se vuelve lineal nuevamente.

Si no ocurren más expiraciones del temporizador, la ventana de congestión continuará creciendo hasta el tamaño de la ventana del receptor. En ese punto, dejará de crecer y permanecerá constante mientras no ocurran más expiraciones del temporizador y la ventana del receptor no cambie de tamaño.

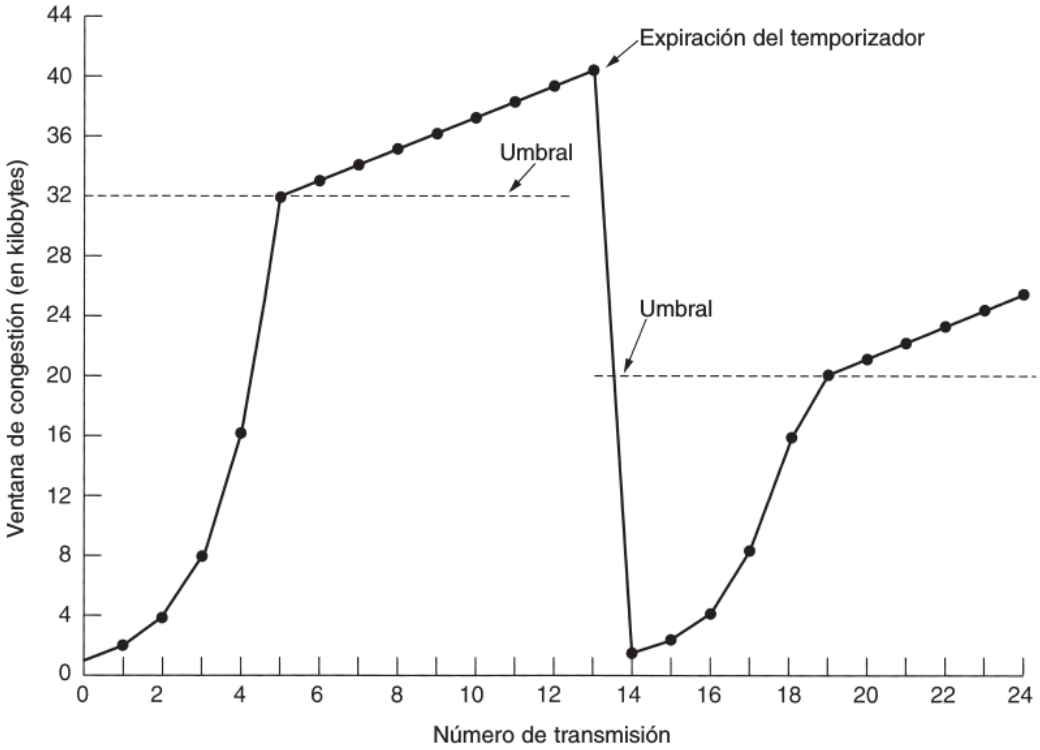

Figura 6.19: Ejemplo del algoritmo de congestión de Internet.

6.5.10. Administración de temporizadores del TCP

TCP utiliza varios temporizadores:

- **De retransmisión**: Para calcular el tiempo de expiración del temporizador de retransmisión se utiliza un algoritmo muy dinámico que continuamente esta actualizando el RTT (*Round-Trip Time*) con la siguiente formula:

$$RTT = \alpha RTT + (1 - \alpha)M$$

  donde M es el tiempo que demora un segmento en viajar hasta el receptor mas el tiempo que demora la confirmación en volver y α es un factor de amortiguamiento que determina el peso que se le da al valor anterior. Por lo común, $\alpha = 7/8$.

  Aun dado un buen valor de RTT, la selección de una expiración adecuada del temporizador de retransmisión no es un asunto sencillo. Normalmente el TCP usa βRTT, pero el truco es seleccionar β. En las implementaciones iniciales, β siempre era 2, pero la experiencia demostró que un valor constante era inflexible puesto que no respondía cuando subía la variación.

  Un problema que ocurre con la estimación dinámica de RTT es qué se debe hacer cuando expira el temporizador de un segmento y se envía de nuevo. Cuando llega la confirmación de recepción, no es claro si éste se refiere a la primera transmisión o a una posterior. Si se adivina mal se puede contaminar seriamente la estimación de RTT.

  Karn hizo una propuesta sencilla: no actualizar el RTT con ninguno de los segmentos retransmitidos. En cambio, se duplica la expiración del temporizador con cada falla hasta que los segmentos pasan a la primera. Este sistema se llama **algoritmo de Karn** y lo usan la mayoría de las implementaciones TCP.

- **De persistencia**: diseñado para evitar el siguiente bloqueo irreversible. El receptor envía una confirmación de recepción con un tamaño de ventana de 0, indicando al emisor que espere. Después, el receptor actualiza la ventana, pero se pierde al paquete con la actualización. Ahora, tanto el emisor como el receptor están esperando que el otro haga algo. Cuando ter mina el temporizador de persistencia, el emisor envía un sondeo al receptor. La respuesta al sondeo da el tamaño de la ventana. Si aún es de cero, se inicia el temporizador de persistencia nuevamente y se repite el ciclo. Si es diferente de cero, pueden enviarse datos.

- **De seguir con vida (*keepalive timer*)**: Cuando una conexión ha estado inactiva durante demasiado tiempo, el temporizador de seguir con vida puede expirar, haciendo que un lado compruebe que el otro aún está ahí. Si no se recibe

respuesta, se termina la conexión. Esta característica es motivo de controversias porque agrega sobrecarga y puede terminar una conexión saludable debido a una partición temporal de la red.

- **El tiempo de espera**: Se usa en el estado `TIMED WAIT` durante el cierre; opera durante el doble del tiempo máximo de vida de paquete para asegurar que, al cerrarse una conexión, todos los paquetes creados por ella hayan desaparecido.

7 LA CAPA DE APLICACIÓN

7.1. DNS - EL SISTEMA DE NOMBRES DE DOMINIO

Dada a la dificultad de memorizar direcciones de red, y de la inflexibilidad de manejar cambios en las direcciones de los servidores al enviar mensajes a IPs, se introdujeron los nombres ASCII, con el fin de separar los **nombres de máquina** de las **direcciones de máquina**. Sin embargo, la red sólo comprende direcciones numéricas, por lo que se requieren algunos mecanismos para convertir las cadenas ASCII de nuevo a direcciones de red.

Una alternativa que surgió fue la de mantener un servidor con un archivo `hosts.txt` que listaba todos los *hosts* y sus direcciones de IP, y que se enviaba periódicamente a todos los hosts de la red. Sin embargo, este método no sería-a práctico ante la presencia de miles de hosts, ya que no solo el tamaño del archivo crecería de manera considerable, sino que además ocurrirían conflictos constantes con los nombres de los hosts a menos de que dichos nombres se administraran de forma centralizada, algo impensable en una red internacional enorme. Para resolver estos problemas, se inventó el **DNS**.

El DNS se basa en un esquema jerárquico que permite asignar nombres, basándose en el concepto de dominio, utilizando para su gestión una base de datos (BD) distribuida. Para relacionar un nombre con una dirección IP, un programa de aplicación llama a un procedimiento de biblioteca llamado **resolvedor**, y le pasa el nombre como parámetro (consulta DNS). El resolvedor envía un paquete UDP a un servidor DNS local, que después busca el nombre y devuelve la dirección IP al resolvedor, que entonces lo remite al solicitante. Una vez que tiene la dirección IP, el programa puede establecer una conexión TCP con el destino, o enviarle paquetes UDP.

7.1.1. El espacio de nombres DNS

Conceptualmente, Internet se divide en 200 **dominios** de nivel superior, cada uno de los cuales abarca muchos *hosts*. Cada dominio se divide en **subdominios**, los cuales, a su vez, también se dividen, y así sucesivamente. Puede verse como un árbol, cuyas hojas representan los dominios que no tienen subdominios, y que además pueden contener un solo *host*, o miles (si representa una organización).

Los dominios de nivel superior se dividen en dos categorías: **genéricos** y **de país** (o *geográficos*).

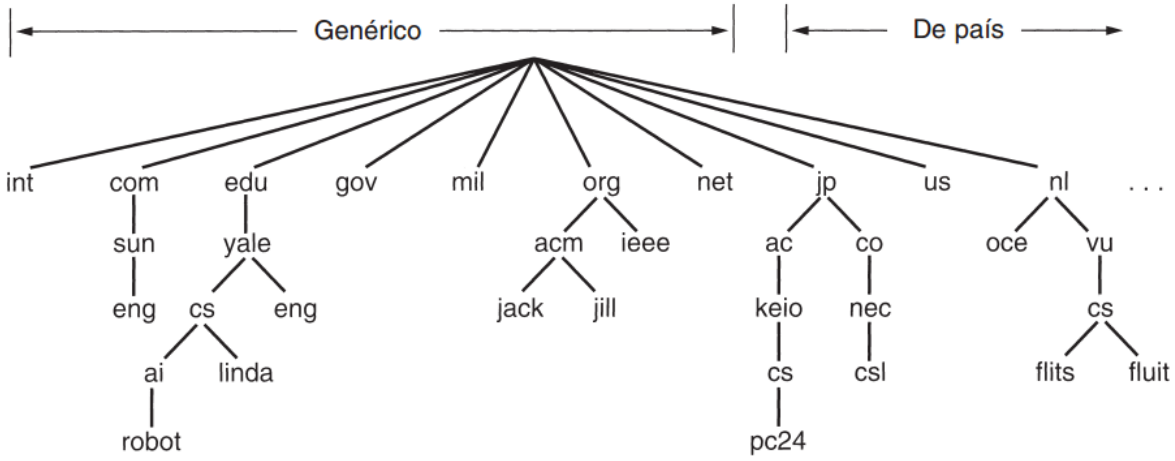

Figura 7.1: Parte del espacio de nombres de dominio de Internet.

Cada dominio se nombra por la ruta hacia arriba desde él a la **raíz** (*sin nombre*). Los componentes se separan con puntos (ej, "*eng.sun.com.*").

Los **nombres de dominio** pueden ser:

- **Absolutos:** terminan con punto (ej, "*eng.sun.com.*").

- **Relativos:** tienen que interpretarse en algún contexto para determinar de manera única su significado verdadero (ej, "*eng.sun.com*").

En ambos casos, un nombre de dominio hace referencia a un nodo específico del árbol y a todos los nodos por debajo de él, y no hacen distinción entre mayúsculas y minúsculas. Los nombres reflejan los **límites organizacionales**, no las redes físicas.

Para crear un nuevo dominio, se requiere el permiso del dominio en el que se incluirá, evitando así los conflictos de nombres y permitiendo a cada dominio llevar el registro de todos sus subdominios. Una vez que se ha creado y registrado un nuevo dominio, este puede crear subdominios, sin obtener el permiso de nadie más arriba en el árbol.

7.1.2. Registro de recursos

Cada dominio, sea un *host* individual o un dominio de nivel superior, puede tener un grupo de **registros de recursos** (RRs) asociados a él. En un *host* individual, el RR más común es simplemente su dirección IP, pero también existen muchos otros tipos de RRs. Cuando un resolvedor da un nombre de dominio al DNS, lo que recibe son los RRs asociados a ese nombre. Por lo tanto, DNS relaciona los dominios de nombres con los RRs.

Cada entrada en la tabla de un DNS contiene información, no sólo de las direcciones IP, si no de un RR, de cinco campos:

- `Nombre_dominio`: el dominio al que pertenece ese registro. Puede haber más de un registro por dominio. Si se omite, se usa por defecto el último nombre de domino indicado.
- `TTL`: indica la estabilidad del registro.
    * La información **altamente estable** tiene un valor grande (86.400 [seg], o sea, 1 día).
    * La información **volátil** recibe un valor pequeño (60 [seg]).
- `Clase`: para la información de Internet, siempre es `IN`. Si se omite, se toma el último valor indicado.
- `Tipo`: indica el tipo de registro de que se trata.
- `Valor`: puede ser un número, un nombre de dominio o una cadena ASCII. La semántica depende del tipo de registro.

Tipo	Significado	Descripción
`SOA`	*Start of Autority*	Inicio de autoridad, identificando el dominio o la zona. Fija una serie de parámetros para esta zona.
`A`	*Address*	Dirección IP de un *host* en 32 `bits`. Si este tiene varias direcciones IP, habrá un registro diferente por cada una de ellas.
`MX`	*Mail eXchanger*	Especifica el nombre del dominio que está preparado para aceptar correo electrónico.
`NS`	*Name Server*	El nombre de dominio se hace corresponder con el nombre de una computadora de confianza para el dominio o servidor de nombres.
`CNAME`	*Canonical Name*	Es un alias que se corresponde con el nombre canónico verdadero.
`PTR`	*Pointer*	Apuntador, hace corresponder una dirección IP con el nombre de un sistema. Usado para asociar {`dirección IP, nombre`}, y realizar de esa forma **búsquedas inversas**.
`HINFO`	*Host Info*	Información del *host*, tipo y modelo de computadora y SO, en ASCII.
`TXT`	*Text*	Texto ACII no interpretado. Permite agregar comentarios a la BD.
`WKS`	*Well-Known Services*	Servicios públicos. Puede listar los servicios de las aplicaciones disponibles en el ordenador.

Cuadro 7.1: Principales tipos de registro de recursos DNS.

7.1.3. Servidores de nombres

Un solo servidor de nombres podría contener toda la BD DNS y responder a todas las consultas dirigidas a ella, pero estaría tan sobrecargado que sería inservible. Más aún, si llegara a caerse, la Internet completa se vendría abajo. Para evitar los problemas asociados a tener una sola fuente de información, el espacio de nombres DNS se divide en **zonas** no traslapantes. Una zona es una parte contigua del **árbol de nombres** que se administra como una unidad.

Cuando un resolvedor tiene una consulta referente a un nombre de dominio, la pasa a uno de los servidores de nombres locales. Si el dominio que se busca cae bajo la jurisdicción del servidor de nombres, devuelve los registros de **recursos autorizados** -que provienen de la autoridad que administra el registro y, por lo tanto, siempre son correctos-. Los registros autorizados contrastan con los **registros en caché**, que podrían no estar actualizados.

Por otro lado, si el dominio es remoto y no hay información disponible localmente sobre el dominio solicitado, el servidor de nombres envía un mensaje de consulta al servidor de nombres de nivel superior en el que le solicita dicho dominio. Este método de consultas conoce como **consulta recursiva**, puesto que cada servidor que no tiene toda la información solicitada la busca en algún otro lado y luego la proporciona. Algunos servidores no implementan este método y siempre devuelven el nombre del siguiente servidor a intentar (**consulta iterativa**).

Cuando un cliente DNS no recibe una respuesta antes de que termine su temporizador, por lo general probará con otro servidor la siguiente vez, asumiendo que el servidor probablemente esté inactivo.

Otro método es el de **búsqueda inversa**, que dado una dirección IP, devuelve el nombre. Para evitar una búsqueda exhaustiva por todo el espacio de nombres de dominio, se utiliza un dominio especial llamado `in-addr.arpa`. Cuando un cliente DNS necesita averiguar el nombre de dominio asociado a la dirección IP `w.x.y.z` realiza una pregunta inversa a `z.y.x.w.in-addr.arpa`.

Nota: la inversión de los bytes es necesaria debido a que los nombres de dominio son más genéricos por la derecha, al contrario que ocurre con las direcciones IP.

Tipos de servidores:

1. **Primarios (*Primary Name Servers*)**: almacenan la información de su zona en una BD local. Son responsables de mantener la información actualizada y cualquier cambio debe ser notificado a este servidor.
2. **Secundarios (*Secundary Name Servers*)**: obtienen los datos de su zona desde otro servidor que tenga autoridad para esa zona. El proceso de copia de la información se denomina **transferencia de zona**.
3. **Maestros (*Master Name Servers*)**: transfieren las zonas a los servidores secundarios. Cuando un servidor secundario arranca busca un servidor maestro y realiza la *transferencia de zona*. Un servidor maestro para una zona puede ser a la vez un servidor primario o secundario de esa zona. Estos servidores extraen la información desde el servidor primario de la zona. Así se evita que los servidores secundarios sobrecargen al servidor primario con transferencias de zonas.
4. **Locales (*Caching-only servers*)**: no tienen autoridad sobre ningún dominio; se limitan a contactar con otros servidores para resolver las peticiones de los clientes DNS. Estos servidores mantienen una memoria caché con las últimas preguntas contestadas. Cada vez que un cliente DNS le formula una pregunta, primero consulta en su memoria caché. Si encuentra la dirección IP solicitada, se la devuelve al cliente; si no, consulta a otros servidores, apuntando la respuesta en su memoria caché y comunicando la respuesta al cliente.
5. **Raíz (“.”)**: se usa para consultar *hosts* externos, cuyas direcciones IP están presentes en un fichero de configuración del sistema y se cargan en el caché del DNS al iniciar el servidor. Proporcionan referencias directas a servidores de los dominios de segundo nivel (COM, EDU, geográficos, étc). Conocen a todos los servidores de dominios de primer nivel. Reciben consultas de servidores locales que no saben resolver un nombre. Hay trece servidores raíz a lo largo del mundo.

7.2. CORREO ELECTRÓNICO

Los primeros sistemas de correo electrónico simplemente consistían en protocolos de transferencia de archivos, conteniendo la primera línea del archivo la dirección del destinatario.

Algunas limitaciones que surgieron de este sistema:

- Envío a grupos.
- Sin notificación de entrega.
- Sin estructura interna de envío.

En 1982 se publicaron las propuestas de correo electrónico del ARPANET:

- `RFC 821`. Protocolo de transmisión SMTP.
- `RFC 822`. Formato de mensaje.

El CCITT elaboró su recomendación X.400 para reemplazar al RFC 822, pero su excesiva complejidad lo hizo desaparecer, como le sucedió a la mayoría de las aplicaciones OSI.

7.2.1. Arquitectura y servicios

Los sistemas de correo electrónico normalmente consisten en dos subsistemas:

- Los **agentes de usuario**, que permiten leer y enviar correo electrónico. Son programas locales para interactuar con el sistema de correo electrónico. El usuario debe proporcionar el mensaje, la dirección de destino (en un formato que el agente de usuario pueda manejar, como direcciones DNS de la forma *example@dns-address*) y, posiblemente, algunos otros parámetros.
- Los **agentes de transferencia de mensajes**, que mueven los mensajes del origen al destino. Son por lo común *daemons* del sistema que operan en segundo plano y mueven correo electrónico a través del sistema. Se clasifican a su vez en:
    * **De distribución.** (SMTP, ESMTP).
    * **De entrega final.** (POP3, IMAP).

Por lo general,los sistemas de correo electrónico desempeñan cinco funciones básicas:

1. La **redacción** se refiere al proceso de crear mensajes y respuestas.
2. La **transferencia** se refiere a mover mensajes del remitente al destinatario. Requiere establecer una conexión con el destino o alguna máquina intermedia, enviar el mensaje y liberar la conexión, todo de forma automática.
3. La **generación del informe** notifica que ocurrió con el mensaje: *¿se entregó, rechazó o se perdió?*
4. La **visualización** de los mensajes es necesaria para que la gente pueda leer su correo electrónico.
5. La **disposición** tiene que ver con lo que el destinatario hace con el mensaje una vez que lo recibe.

En los sistemas de correo electrónico, existe una clara distinción entre el **sobre** (RFC 821) y su **contenido** (RFC 822). El sobre encapsula el mensaje; contiene toda la información necesaria para transportar el mensaje, como dirección de destino, prioridad y nivel de seguridad, la cual es diferente del mensaje mismo. Los agentes de transporte del mensaje usan el sobre para enrutar.

El mensaje dentro del sobre contiene dos partes: el **encabezado**, que contiene información de control para los agentes de usuario, y el **cuerpo**, que es por completo para el destinatario humano.

7.2.2. El agente de usuario

Un agente de usuario normalmente es un programa (a veces llamado lector de correo) que acepta una variedad de comandos para redactar, recibir y contestar los mensajes, así como para manipular los buzones de correo.

- **Envío de correo**: El usuario debe proporcionar el mensaje, la dirección de destino y, posiblemente, algunos otros parámetros. El mensaje puede producirse con un editor de texto independiente o, posiblemente, un editor de texto incorporado en el agente de usuario. La dirección de destino debe estar en un formato que el agente de usuario pueda manejar. Muchos agentes de usuario esperan direcciones DNS de la forma `usuario@dirección-dns`. Existen otros formatos como el propuesto por **X.400**. Estas se componen de pares `atributo = valor`, separadas por diagonales. Por ejemplo:

  ```
  /C=US/ST=MASSACHUSETTS/L=CAMBRIDGE/PA=360 MEMORIAL DR./CN=KEN SMITH/
  ```

  Esta dirección especifica un PAÍS (C), ESTADO (S), LOCALIDAD (L), DIRECCIÓN PERSONAL (PA) y NOMBRE COMÚN (CN). Son posibles muchos otros atributos, de modo que puede enviarse correo electrónico a alguien cuyo nombre no se conoce, siempre y cuando se conozca un número suficiente de atributos.
- **Lectura de correo**: Por lo común, cuando se inicia un agente de usuario, buscará en el buzón del usuario el correo electrónico recibido, antes de presentar otra cosa en la pantalla. Después puede anunciar la cantidad de mensajes en el buzón o presentar un resumen de una línea de cada uno y esperar un comando.

7.2.3. Formato de mensajes

7.2.3.1. RFC 822

Los mensajes consisten en un sobre primitivo, algunos campos de encabezado, una línea en blanco y el cuerpo del mensaje. Cada campo de cabecera consiste en una sola línea de texto ASCII que contiene el nombre del campo, dos puntos (:), y, para la mayoría de los campos, un valor.

Encabezado	Significado
To:	Direcciones de correo electrónico de los destinatarios primarios
Cc:	Direcciones de correo electrónico de los destinatarios secundarios
Bcc:	Direcciones de correo electrónico para las copias ocultas
From:	Persona o personas que crearon el mensaje
Sender:	Dirección de correo electrónico del remitente
Received:	Línea agregada por cada agente de transferencia en la ruta
Return-Path:	Puede usarse para identificar una ruta de regreso al remitente

Figura 7.2: Campos de encabezado `RFC 822` relacionados con el transporte de mensajes.

Encabezado	Significado
Date:	Fecha y hora de envío del mensaje
Reply-To:	Dirección de corre electrónico a la que deben enviarse las contestaciones
Message-Id:	Número único para referencia posterior a este mensaje
In-Reply-To:	Identificador del mensaje al que éste responde
References:	Otros identificadores de mensaje pertinentes
Keywords:	Claves seleccionadas por el usuario
Subject:	Resumen corto del mensaje para desplegar en una línea

Figura 7.3: Campos de encabezado `RFC 822` usados por los agentes.

El `RFC 822` explícitamente indica que los usuarios pueden inventar cabeceras nuevas para uso privado siempre y cuando comiencen con la cadena `X-`.

7.2.3.2. MIME - Extensiones Multipropósito de Correo de Internet

La idea básica de MIME es continuar usando el formato `RFC 822`, pero agregar una estructura al cuerpo del mensaje y definir reglas de codificación para los mensajes no ASCII. De esa manera, nada cambia de la arquitectura anterior del `RFC 822`. Sólo afecta a los agentes de usuario, ya que para SMTP es totalmente transparente.

Encabezado	Significado
`MIME-Version`	Identifica la version de MIME. Si no existe se considera que el mensaje es texto normal en inglés.
`Content-Description`	Cadena de texto que describe el contenido. Esta cadena es necesaria para que el destinatario sepa si desea descodificar y leer el mensaje o no.
`Content-Id`	Identificador único, usa el mismo formato que el encabezado estándar `Message-Id`.
`Content-Transfer-Encoding`	Indica la manera en que está envuelto el cuerpo del mensaje. Existen cinco tipos básicos de **esquemas** de codificación de mensajes: `ASCII 7`, `ASCII 8`, *codificación binaria*, `base64`, y *codificación entrecomillada imprimible*.
`Content-Type`	Especifica la naturaleza del cuerpo del mensaje. Existen 7 tipos básicos (*text, image, audio, video, application, message, multipart*), cada uno de los cuales tiene uno o más subtipos. El *tipo* y el *subtipo* se separan mediante un carácter diagonal (/). Ejemplo: `video/mpeg`.

Cuadro 7.2: Encabezados de mensaje MIME.

7.2.4. Transferencia de mensajes

7.2.4.1. SMTP - Protocolo Simple de Transporte de Correo

SMTP es un protocolo ASCII sencillo **cliente/servidor**. El correo electrónico se entrega al hacer que la máquina de origen establezca una conexión TCP con el **puerto 25** de la máquina de destino. Escuchando en este puerto está un *daemon* de correo electrónico que habla con el SMTP. Este *daemon* acepta conexiones de entrada y copia mensajes de estas a los buzones adecuados. Si no puede entregarse un mensaje, se devuelve al remitente un informe de error que contiene la primera parte del mensaje que no pudo entregarse.

Después de establecer la conexión TCP con el puerto 25, el servidor comienza por enviar una línea de texto que proporciona su identidad e indica si está preparado o no para recibir correo:

- Si **no está dispuesto**, el cliente libera la conexión y lo intenta después.
- Si **está dispuesto**, el cliente anuncia de quién proviene el mensaje, y a quién está dirigido. Si existe el destinatario en el destino, el servidor da al cliente permiso para enviar el mensaje. A continuación el cliente envía el mensaje y el servidor confirma su recepción. Una vez que todo el correo electrónico ha sido intercambiado en **ambas direcciones**, se libera la conexión.

Aunque el protocolo SMTP está bien definido, pueden surgir algunos problemas:

- La longitud del mensaje en implementaciones viejas de SMTP está limitada a `64 Kb`.
- Si el cliente y el servidor tienen temporizaciones distintas, uno de ellos puede terminar mientras que el otro continúa trabajando, terminando inesperadamente la conexión.

Para superar algunos de estos problemas, se ha definido el **SMTP extendido** (**ESMTP**).

7.2.5. Entrega final

Problema: acceso no permanente a internet, por ende, no se puede enviar correo directo entre *hosts*.

Solución: que un agente de transferencia de mensajes en una máquina ISP acepte correo electrónico para sus clientes y lo almacene en sus buzones en una máquina ISP con acceso permanente.

Un cliente puede consultar su buzón en la máquina ISP a través de los protocolos **IMAP** y **POP3**.

El **POP3** (*Post Office Protocol* version 3) es un protocolo definido en la `RFC 1225` y actualizado en la `RFC 1939`. POP3 está diseñado para recibir correo; le permite a los usuarios con conexiones intermitentes o muy lentas (tales como las conexiones por módem), descargar su correo electrónico mientras tienen conexión y revisarlo posteriormente incluso estando desconectados.

Aunque algunos clientes de correo incluyen la opción de dejar los mensajes en el servidor, el funcionamiento general es:

1. Un cliente que utilice POP3 se conecta.
2. Se obtienen todos los mensajes.

3. Se almacenan los mensajes en la computadora del usuario como mensajes nuevos.
4. Se eliminan los mensajes del servidor
5. El cliente se desconecta.

Tiene comandos para que un usuario establezca una sesión (`USER` y `PASS`), la termine (`QUIT`), obtenga mensajes (`RETR`) y los borre (`DELE`). El protocolo mismo consiste en texto ASCII y se asemeja a SMTP. El objetivo del POP3 es obtener correo electrónico del buzón remoto y almacenarlo en la máquina local del usuario para su lectura posterior.

Es posible conectarse manualmente al servidor POP3 haciendo Telnet al **puerto 110**.

Muchas personas tienen una sola cuenta de correo electrónico en el trabajo o en la escuela y desean accederla desde el trabajo, desde la PC de su casa, desde su computadora portátil, etc. Aunque POP3 permite esto, debido a que descarga todos los mensajes almacenados en cada contacto, el resultado es que los mensajes de correo electrónico del usuario quedan esparcidos rápidamente en múltiples máquinas. Esta desventaja dio lugar a un protocolo de entrega final alternativo, el Protocolo de Acceso a Mensajes de Internet (IMAP: (*Internet Message Access Protocol*). Es un protocolo de aplicación que permite el acceso a mensajes almacenados en un servidor de Internet. Definido en la `RFC 1064` y actualizado en la `RFC 2060`.

La idea en que se basa IMAP es que el servidor de correo electrónico mantenga un depósito central al que puede accederse desde cualquier máquina. Por tanto, a diferencia del POP3, no copia el correo electrónico en la máquina personal del usuario dado que el usuario puede tener varias computadoras para consultar el correo, y observa si sus correos han sido leídos con anterioridad. Se utiliza el **puerto 143**.

Protocolo	Ventajas	Desventajas
IMAP 4	• Trabaja en modo de conexión permanente, por lo que avisa inmediatamente de la llegada de nuevo correo. • Transmite solo las cabeceras por lo que el usuario puede decidir su borrado inmediato. • La bajada del mensaje se produce solo cuando el usuario quiere leerlo. • El almacenamiento local del mensaje es opcional (una opción del cliente de correo). • Gestiona carpetas, plantillas y borradores en el servidor. • El almacenamiento de mensajes y carpetas en el servidor permite su uso desde múltiples dispositivos y de forma simultánea. • Permite la búsqueda de mensajes por medio de palabras claves. • Los mensajes se pueden etiquetar. El marcado queda en el servidor. • Se pueden crear carpetas compartidas con otros usuarios (depende del servidor).	• No todos los clientes de correo soportan la extensión IMAP IDLE (aviso de nuevos correos). • Necesita una transacción por cada correo que se quiera leer. • Hay un retraso en la aparición del mensaje en la pantalla del usuario, mientras se descarga. • Si se pierde la conexión, no se podrá ver el mensaje salvo si el cliente de correo lo haya almacenado en local. • Las carpetas, plantillas y borradores no podrán ser leídos usando POP (excepto la Bandeja de entrada).
POP3	• Los correos aparecen inmediatamente porque quedan residentes en el dispositivo (una vez descargados). • Entre servidor-cliente no se tienen que enviar tantas órdenes para la comunicación entre ellos. • Funciona adecuadamente si no se utiliza una conexión constante a Internet o a la red que contiene el servidor de correo.	• Sólo se conecta periódicamente cada X minutos para buscar por nuevo correo. • La conexión periódica provoca un aumento del tráfico y un retraso en la respuesta del cliente (esperar la descarga completa). • En cada conexión, se baja todos los correos nuevos, vayan a ser después leídos o no. • Los correos ocupan espacio local del dispositivo. • Por defecto, elimina los mensajes del servidor, haciendo imposible el acceso a ellos desde otro dispositivo.

Cuadro 7.3: Comparación entre IMAP 4 y POP3.

7.3. WORLD WIDE WEB

7.3.1. Panorama de la arquitectura

Desde el punto de vista del usuario, Web consiste en un enorme conjunto de documentos a nivel mundial, generalmente llamados páginas Web. Cada página puede contener vínculos (apuntadores) a otras páginas relacionadas en cualquier lugar del mundo. Los usuarios pueden seguir un vínculo haciendo clic en él, lo que los lleva a la página apuntada. Este proceso puede repetirse de manera indefinida.

Las páginas se ven mediante un programa llamado navegador. El navegador obtiene la página solicitada, interpreta el texto y los comandos de formateo que contienen, y despliega la página, adecuadamente formateada, en la pantalla.

7.3.1.1. El Cliente

En esencia, un navegador es un programa que puede desplegar una página Web y atrapar los clicks que se hacen en los elementos de la página desplegada. Cuando se selecciona un elemento, el navegador sigue el hipervínculo y obtiene la página seleccionada. Por lo tanto, el hipervínculo incrustado necesita una manera de nombrar cualquier página que se encuentre en Web. Las páginas se nombran utilizando `URLs` (Localizadores Uniformes de Recursos).

Para poder desplegar la nueva página, el navegador tiene que entender su formato. Para permitir que todos los navegadores entiendan todas las páginas Web, éstas se escriben en un lenguaje estandarizado llamado `HTML`, el cual las describe. No todas las páginas contienen `HTML`. Una página puede consistir en un documento con formato PDF, GIF, JPEG, MP3, etc. Para abrir estos archivos los navegadores utilizan plug-ins o aplicaciones auxiliares.

7.3.1.2. El Servidor

Un servidor Web generalmente recibe solicitudes de paginas y responde con el archivo solicitado. El proceso general es el siguiente:

- Resuelve el nombre de la página Web solicitada.
- Autentica al cliente, de ser necesario.
- Realiza control de acceso en el cliente.
- Realiza control de acceso en la página Web.
- Verifica el caché.
- Obtiene del disco la página solicitada si esta no estaba en cache.
- Determina el tipo MIME que se incluirá en la respuesta.
- Se encarga de diversos detalles.
- Regresa la respuesta al cliente.
- Realiza una entrada en el registro del servidor.

Como se puede ver, los servidores siempre utilizan cache para evitar el proceso de leer cada pagina del disco. Otras técnicas para que el servidor pueda responder mas solicitudes, es la utilización de varios subprocesos dentro de cada maquina y varios discos. Un subproceso recibe todos los pedidos y se los pasa a otro subproceso para que se encargue de responderlo. Todos estos comparten la misma memoria cache. También se puede utilizar varias maquinas en lugar de varios subprocesos dentro de la misma maquina. Esto se conoce como **granja de servidores**. La desventaja es que la memoria cache no se comparte y que es necesaria una maquina intermediaria para responder las solicitudes y delegar los trabajos.

7.3.1.3. Localizadores Uniformes de Recursos (URLs)

Un `URL` tiene tres partes:

- El nombre del protocolo (`http`).
- El nombre DNS de la máquina donde se localiza la página (`www.abcd.com`).
- Generalmente, el nombre del archivo que contiene la página (`productos.html`).

Este esquema de URL es abierto en el sentido de que es directo hacer que los navegadores utilicen múltiples protocolos para obtener diferentes tipos de recursos. De hecho, se han definido los URLs de varios otros protocolos comunes.

7.3.1.4. Sin estado y cookies

Cuando un cliente solicita una página Web, el servidor puede proporcionar información adicional junto con la página solicitada. Esta información puede incluir una ***cookie***, que es un pequeño archivo (o cadena, de a lo mucho 4 KB). Los navegadores almacenan *cookies* ofrecidas en un directorio de *cookies* en el disco duro de la máquina del cliente, a menos que el usuario las haya deshabilitado. Las *cookies* son simplemente archivos o cadenas, no programas ejecutables. En principio, una *cookie*

Nombre	Usado para	Ejemplo
http	Hipertexto (HTML)	http://www.cs.vu.nl/~ast/
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	Archivo local	file:///usr/suzanne/prog.c
news	Grupo de noticias	news:comp.os.minix
news	Artículo	news:AA0134223112@cs.utah.edu
gopher	Gopher	gopher://gopher.tc.umn.edu/11/Libraries
mailto	Envío de correo electrónico	mailto:JohnUser@acm.org
telnet	Inicio de sesión remota	telnet://www.w3.org:80

Figura 7.4: Algunos URLs comunes.

puede contener un virus, pero puesto que las *cookies* se tratan como datos, no hay una forma oficial de que los virus se ejecuten realmente y hagan daño.

Una cookie puede contener hasta cinco campo:

- **Dominio**: indica de dónde viene la *cookie*.
- **Ruta**: es la estructura del directorio del servidor que identifica qué partes del árbol de archivos del servidor podrían utilizar la *cookie*. Por lo general es /, lo que significa el árbol completo.
- **Contenido**: toma la forma `nombre = valor`. Este campo es donde se almacena el contenido de la *cookie*.
- **Expira**: especifica cuándo caduca la *cookie*.
- **Seguro**: puede establecerse para indicar que el navegador podría simplemente regresar la *cookie* a un servidor seguro. Esta característica se utiliza para comercio electrónico, actividades bancarias y otras aplicaciones seguras.

Justo antes de que un navegador solicite una página a un sitio Web, verifica su directorio de *cookies* para ver si el dominio al que está solicitando la página ya colocó alguna *cookie*. De ser así, todas las **cookies** colocadas por ese dominio se incluyen en el mensaje de solicitud. Cuando el servidor las obtiene, puede interpretarlas de la forma que desee.

7.3.2. HTTP - Protocolo de Transferencia de Hipertexto

El protocolo de transferencia utilizado en *World Wide Web* es HTTP. Especifica cuáles mensajes pueden enviar los clientes a los servidores y qué respuestas obtienen. Cada interacción consiste en una solicitud ASCII, seguida por una respuesta tipo MIME del `RFC 822`. Todos los clientes y servidores deben obedecer este protocolo. Se define en el `RFC 2616`.

7.3.2.1. Conexiones

La forma común en que un navegador contacta a un servidor es estableciendo una conexión TCP con el puerto 80 de la máquina del servidor, aunque este procedimiento no se requiere formalmente. El valor de utilizar TCP es que ni los navegadores ni los servidores tienen que preocuparse por los mensajes largos, perdidos o duplicados, ni por las confirmaciones de recepción.

En HTTP 1.0, una vez que se establecía la conexión, se enviaba una solicitud y se obtenía una respuesta. Después se liberaba dicha conexión. Este método era adecuado en un mundo en el que una página Web típica consistía por completo de texto HTML. Sin embargo, años más tarde la página Web promedio contenía una gran cantidad de iconos, imágenes, entre otras cosas, por lo que establecer una conexión TCP para transportar un solo icono era un método muy costoso.

A partir de HTTP 1.1 las conexiones son **persistentes**, es decir, una vez establecida la conexión, el cliente puede realizar varias solicitudes, sin la necesidad de establecer una conexión nueva para cada una. También es posible enviar solicitudes en canalización, es decir, enviar la solicitud 2 antes de que la respuesta a la solicitud 1 haya llegado.

7.3.2.2. Métodos

HTTP se ha hecho intencionalmente más general de lo necesario con miras a las aplicaciones orientadas a objetos futuras. Por esta razón, se soportan otras operaciones, llamadas **métodos**. Cada solicitud consiste en una o más líneas de texto ASCII, y la primera palabra de la primera línea es el nombre del método solicitado.

Método	Descripción
GET	Solicita la lectura de una pagina Web.
HEAD	Solicita la lectura del encabezado de una pagina. Se puede utilizar para obtener la fecha de la ultima modificación de la pagina, para indización por ejemplo.
PUT	Solicita el almacenamiento de una pagina. El cuerpo de la solicitud contiene la pagina, junto con datos de autenticación.
POST	Similar a PUT, solo que este método inserta los datos en algún sentido generalizado. Por ejemplo un mensaje a un grupo de noticias.
DELETE	Elimina la pagina. Se debe acompañar de datos de autenticación.
TRACE	Se utiliza en depuración. Indica al servidor que devuelva la ultima solicitud.
CONNECT	No se utiliza. Se reserva para un uso futuro.
OPTIONS	Proporciona una forma para que el cliente consulte al servidor sobre sus propiedades o las de un archivo específico.

Cuadro 7.4: Métodos del protocolo HTTP.

Cada solicitud obtiene una respuesta que consiste en una línea de estado, y posiblemente de información adicional (por ejemplo, toda o parte de una página Web). La línea de estado contiene un código de estado de tres dígitos que indica si la solicitud fue atendida, y si no, por qué. El primer dígito se utiliza para dividir las respuestas en cinco grupos mayores. Los significados de los códigos se enumeran en el Cuadro 7.4.

Código	Significado	Ejemplos
1xx	Información	100 = el servidor está de acuerdo en manejar la solicitud del cliente.
2xx	Éxito	200 = la solicitud es exitosa; 204 = no hay contenido.
3xx	Redirección	301 = página movida; 304 = la página en caché aún es válida.
4xx	Error del cliente	403 = página prohibida; 404 = página no encontrada.
5xx	Error del servidor	500 = error interno del servidor; 503 = trata más tarde.

Cuadro 7.5: Los grupos de respuesta del código de estado.

7.3.2.3. Encabezados del mensaje

A la línea de solicitud (por ejemplo, la línea con el método GET) le pueden seguir líneas adicionales que contienen más información. Éstas se llaman encabezados de solicitud. Esta información puede compararse con los parámetros de una llamada a procedimiento. Las respuestas también pueden tener encabezados de respuesta. Algunos encabezados pueden utilizarse en cualquier dirección. Los encabezados se listan en la Figura 7.5.

7.3.3. FTP - Protocolo de Transferencia de Archivos

FTP (*File Transfer Protocol*) es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP, basado en la arquitectura cliente-servidor. Desde un equipo cliente se puede conectar a un servidor para descargar archivos desde él o para enviarle archivos. El servicio FTP es ofrecido utilizando normalmente el puerto de red 20 (para el envío de datos, es decir, archivos) y el 21 (para el envió de ordenes y control).

Un problema básico de FTP es que está pensado para ofrecer la máxima velocidad en la conexión, pero no la máxima seguridad, ya que todo el intercambio de información, desde el *login* y *password* del usuario en el servidor hasta la transferencia de cualquier archivo, se realiza en texto plano sin ningún tipo de cifrado.

Encabezado	Tipo	Contenido
User-Agent	Solicitud	Información acerca del navegador y su plataforma
Accept	Solicitud	El tipo de páginas que el cliente puede manejar
Accept-Charset	Solicitud	Los conjuntos de caracteres que son aceptables para el cliente
Accept-Encoding	Solicitud	Las codificaciones de página que el cliente puede manejar
Accept-Language	Solicitud	Los idiomas naturales que el cliente puede manejar
Host	Solicitud	El nombre DNS del servidor
Authorization	Solicitud	Una lista de las credenciales del cliente
Cookie	Solicitud	Regresa al servidor una cookie establecida previamente
Date	Ambas	Fecha y hora en que se envió el mensaje
Upgrade	Ambas	El protocolo al que el emisor desea cambiar
Server	Respuesta	Información acerca del servidor
Content-Encoding	Respuesta	Cómo se codifica el contenido (por ejemplo, gzip)
Content-Language	Respuesta	El idioma natural utilizado en la página
Content-Length	Respuesta	La longitud de la página en bytes
Content-Type	Respuesta	El tipo MIME de la página
Last-Modified	Respuesta	La fecha y hora de la última vez que cambió la página
Location	Respuesta	Un comando para que el cliente envíe su solicitud a cualquier otro lugar
Accept-Ranges	Respuesta	El servidor aceptará solicitudes de rango de bytes
Set-Cookie	Respuesta	El servidor desea que el cliente guarde una cookie

Figura 7.5: Algunos encabezados de mensaje HTTP.

FTP permite copiar archivos de un sistema a otro, listar directorios, realizar tareas de gestión como eliminar archivos y cambiar nombres, etc. Existe una variante llamada TFTP (*Trivial FTP*) que es un protocolo de transferencia muy simple. A menudo se utiliza para transferir pequeños archivos entre ordenadores en una red, como cuando un terminal *X Window* o cualquier otro cliente ligero arranca desde un servidor de red.

7.3.4. Telnet (Terminal Networking)

Es un protocolo de red a otra máquina para manejarla remotamente. También es el nombre del programa informático que implementa el cliente. La máquina a la que se acceda debe tener un programa especial que reciba y gestione las conexiones. Se utiliza generalmente el **puerto 23**.

Sólo sirve para acceder en modo terminal, pero fue una herramienta muy útil para arreglar fallos a distancia, sin necesidad de estar físicamente en el mismo sitio que la máquina que los tenía. También se usaba para consultar datos a distancia, como datos personales en máquinas accesibles por red, información bibliográfica, etc.

Su mayor problema es de seguridad, ya que todos los nombres de usuario y contraseñas necesarias para entrar en las máquinas viajan por la red como texto plano. Por esta razón dejó de usarse, casi totalmente, cuando apareció y se popularizó SSH, que puede describirse como una versión cifrada de telnet.

8 SEGURIDAD EN REDES

CONCEPTOS GENERALES Y DEFINICIONES

Lograr la **seguridad informática** consiste en implementar un conjunto de mecanismos y protocolos que minimicen la vulnerabilidad de bienes (algo de valor) y recursos. Se entiende por vulnerabilidad a una debilidad que se puede explotar para violar un sistema o la información que contiene. En la implementación de métodos de seguridad deben considerarse cuestiones legales ya que, por ejemplo, en algunos países el uso de información cifrada está prohibido, mientras que para algunos gobiernos los temas referentes a seguridad son muy complejos y estos tratan de implantar reglas o estándares de cifrado (que ellos mismos puedan descifrar fácilmente). Se plantea entonces la dualidad *segridad vs. privacidad*.

Conceptos

Seguridad de una red: implica la seguridad de cada uno de los dispositivos de la red.

Hacker: tiene el objetivo de demostrar que algo no es seguro.

Cracker: utiliza sus ataques para sacar beneficio económico o perjudicar.

Lamer: utiliza herramientas existentes. No crea nada nuevo.

Amenaza o ataque: intento de sabotear una operación o la propia preparación para sabotearla (poner en compromiso). Los tipos de amenazas son:

- **Compromiso**: la entidad atacante obtiene el control de algún elemento interno de la red, por ejemplo utilizando cuentas con password triviales o errores del sistema.
- **Modificación**: la entidad atacante modifica el contenido de algún mensaje o texto.
- **Suplantación**: la entidad atacante se hace pasar por otra persona.
- **Reenvío**: la entidad atacante obtiene un mensaje o texto en tránsito y más tarde lo reenvía para duplicar su efecto.
- **Denegación de servicio**: la entidad atacante impide que un elemento cumpla su función.

Clasificación de problemas de seguridad

La mayoría de los problemas de seguridad son causados intencionalmente por gente maliciosa que intenta ganar algo o hacerle daño a alguien. Hacer segura una red comprende mucho más que simplemente mantener los programas libres de errores de programación. Las medidas para detener a los adversarios casuales tendrán poca eficacia contra los adversarios serios. Los problemas de seguridad de las redes pueden dividirse de forma general en cuatro áreas interrelacionadas:

1. **Secreto**: encargado de mantener la información fuera de las manos de usuarios no autorizados.
2. **Validación de identificación**: encargada de determinar la identidad de la persona/computadora con la que se esta hablando.
3. **Control de integridad**: encargado de asegurar que el mensaje recibido fue el enviado por la otra parte y no un mensaje manipulado por un tercero.
4. **No repudio**: encargado de asegurar la “firma” de los mensajes, de igual forma que se firma en papel una petición de compra/venta entre empresas.

La seguridad en la pila de protocolos

En la capa física podemos protegernos contra la intervención de las líneas de transmisión encerrando éstas en tubos sellados que contengan gas a alta presión. Cualquier intento de hacer un agujero en el tubo liberará un poco de gas, con lo cual la presión disminuirá y se disparará una alarma. Algunos sistemas militares usan esta técnica.

En la capa de enlace de datos, los paquetes de una línea punto a punto pueden encriptarse cuando se envíen desde una máquina y desencriptarse cuando lleguen a otra. Los detalles pueden manejarse en la capa de enlace de datos, sin necesidad de que las capas superiores se enteren de ello. Sin embargo, esta solución se viene abajo cuando los paquetes tienen que atravesar varios enrutadores, puesto que los paquetes tienen que desencriptarse en cada enrutador, dentro del cual son vulnerables a posibles ataques. No obstante, la **encriptación de enlace** (*link encryption*), como se llama a este método, puede agregarse fácilmente a cualquier red y con frecuencia es útil.

En la capa de red pueden instalarse *firewalls* para mantener adentro (o afuera) a los paquetes. La seguridad de IP también funciona en esta capa.

En la capa de transporte pueden encriptarse conexiones enteras, de extremo a extremo, es decir, proceso a proceso. Para lograr una máxima seguridad, se requiere seguridad de extremo a extremo.

Por último, los asuntos como la autenticación de usuario y el no repudio sólo pueden manejarse en la capa de aplicación.

Como puede verse, no se puede atribuir la seguridad por completo a una sóla capa o a un conjunto específico de capas. Casi toda la seguridad se basa en principios de criptografía, a excepción de la seguridad en la capa física.

8.1. CRIPTOGRAFÍA

8.1.1. Introducción a la criptografía

Cifrado: es una transformación carácter por carácter o bit por bit, sin importar la estructura lingüística del mensaje.

Código: reemplaza una palabra con otra palabra o símbolo.

Texto llano: son los mensajes por encriptar.

Clave: es el parámetro utilizado por una función encriptadora.

Texto cifrado: es el resultado del proceso de encriptación.

Intruso: escucha y copia con exactitud todo el texto cifrado. Sin embargo, a diferencia del destinatario original, el intruso no conoce la clave de desencriptación y no puede desencriptar con facilidad el texto cifrado. Hay dos tipos:

- **Intruso pasivo**: sólo escucha el canal de comunicación.
- **Intruso activo**: además de escuchar el canal de comunicación también puede registrar mensajes y reproducirlos posteriormente, inyectar sus propios mensajes y modificar los mensajes legítimos antes de que lleguen al destinatario.

Criptografía: se traduce del griego como “escritura secreta u oculta” (*kryptos* = oculto, *graphe* = escrito). Se divide generalmente en:

- **Criptografía clásica**: El algoritmo secreto. Se realizan cifrados por sustitución y transposición
- **Criptografía moderna**: El algoritmo es público. Se realizan cifrados en base a claves que se mantienen secretas.

Criptoanálisis: se encarga de descifrar los mensajes. Los intrusos utilizan estas técnicas.

Criptología: es la combinación de la creación de cifrados (criptografía) y el descifrado de mensajes (criptoanálisis).

Hasta la llegada de las computadoras, una de las principales restricciones de la criptografía había sido la capacidad del empleado encargado de la codificación para realizar las transformaciones necesarias, con frecuencia en un campo de batalla con poco equipo. Una restricción adicional ha sido la dificultad de cambiar rápidamente de un método de criptografía a otro, debido a que esto implica volver a capacitar a una gran cantidad de personas. Sin embargo, el peligro de que un empleado fuera capturado por el enemigo ha hecho indispensable la capacidad de cambiar el método de criptografía de manera instantánea, de ser necesario.

Utilizaremos $C = E_K(P)$ para indicar que la encriptación del texto llano P usando la clave K produce el texto cifrado C. Del

mismo modo, $P = D_K(C)$ representa la desencriptación de C para obtener el texto llano nuevamente. Por lo tanto,

$$D_K(E_K(P)) = P$$

donde E y D son funciones matemáticas de dos parámetros y se ha escrito uno de los parámetros (la clave) como subíndice, en lugar de como argumento, para distinguirlo del mensaje.

Una regla fundamental de la criptografía es que se debe suponer que el criptoanalista conoce el método general de encriptación y desencriptación usado. En otras palabras, el criptoanalista sabe con detalle cómo funciona el método de encriptación, E, y desencriptación, D.

La clave consiste, generalmente, en una cadena corta. En contraste con el método general, que tal vez se cambie cada cierto número de años, la clave puede cambiarse con la frecuencia requerida. Por lo tanto, nuestro modelo básico es un método general estable y conocido públicamente pero parametrizado por una clave secreta y que puede cambiarse con facilidad. La idea de que el criptoanalista conozca los algoritmos y que la naturaleza secreta se base principalmente en las claves se conoce como **principio de Kerckhoff**: *Todos los algoritmos deben ser públicos; sólo las claves deben ser secretas*. Tratar de mantener secreto el algoritmo se conoce como **seguridad por desconocimiento** es un método que nunca funciona.

Puesto que la parte secreta debe ser la clave, la longitud de ésta es un aspecto importante del diseño. Cuanto más grande sea la clave, mayor será el factor de trabajo que tendrá que enfrentar el criptoanalista. El factor de trabajo para descifrar el sistema mediante una búsqueda exhaustiva del espacio de clave crece exponencialmente con la longitud de la clave. El secreto radica en tener un algoritmo robusto (pero público) y una clave larga.

Desde el punto de vista del criptoanalista, el problema del criptoanálisis presenta tres variaciones principales:

1. **Sólo texto cifrado**: cuando el criptoanalista cuenta con cierta cantidad de texto cifrado pero no tiene texto llano.
2. **Texto llano conocido**: cuando el criptoanalista cuenta con texto cifrado y el texto llano correspondiente.
3. **Texto llano seleccionado**: cuando el criptoanalista tiene la capacidad de encriptar textos normales que él escoge

Principios criptográficos fundamentales:

- Introducir **redundancia** (relleno) en los mensajes que permita acotar la posibilidad de ataque y/o en su caso, detectar fácilmente el sabotaje. Por ejemplo CRC y *hash* de los mensajes.
- Introducir una **marca temporal** en los mensajes, lo que permite restringir los mensajes cifrados a un intervalo de tiempo, filtrando duplicaciones de mensajes viejos

8.1.2. Cifrados por sustitución

En un cifrado por sustitución, cada letra o grupo de letras se reemplazan por otra letra o grupo de letras para disfrazarla. Uno de los cifrados más viejos conocidos es el **cifrado de César**. En este método, a se vuelve D, b se vuelve E, c se vuelve F, . . . , y z se vuelve C. Por ejemplo, *ataque* se vuelve *DWDTXH*.

Una pequeña generalización del cifrado de César permite que el alfabeto de texto cifrado se desplace k letras, en lugar de siempre 3. En este caso, k se convierte en una clave del método general de alfabetos desplazados circularmente.

La siguiente mejora es hacer que cada uno de los símbolos del texto llano, digamos las 26 letras del abecedario (no incluimos la ñ) inglés, tengan una correspondencia con alguna otra letra. Por ejemplo:

Texto llano:	a b c d e f g h i j k l m n o p q r s t u v w x y z
Texto cifrado:	Q W E R T Y U I O P A S D F G H J K L Z X C V B N M

El sistema general de sustitución de símbolo por símbolo se llama **sustitución monoalfabética**, siendo la clave la cadena de 26 letras correspondiente al alfabeto completo. Para la clave anterior, el texto llano *ataque* se transformaría en el texto cifrado *QZQJXT*.

A primera vista, esto podría parecer un sistema seguro, porque, aunque el criptoanalista conoce el sistema general (sustitución letra por letra), no sabe cuál de las $26! \approx 4 \times 10^{26}$ claves posibles se está usando. En contraste con el cifrado de César, intentarlas todas, aun a 1 nseg por solución, una computadora tardaría 10^{10} años en probar todas las claves.

No obstante, si se cuenta con una cantidad aun pequeña de texto cifrado, puede descifrarse fácilmente. El ataque básico aprovecha las propiedades estadísticas de los lenguajes naturales. En inglés, por ejemplo, la e es la letra más común, seguida de t, o, a, n, i, etc. Las combinaciones de dos letras más comunes, o digramas, son th, in, er, re y an. Las combinaciones

de tres letras más comunes, o trigramas, son the, ing, and e ion.

Por ejemplo, Un criptoanalista que intenta descifrar un cifrado monoalfabético comenzaría por contar la frecuencia relativa de todas las letras del texto cifrado. Entonces podría asignar tentativamente la más común a la letra e y la siguiente más común a la letra t. Vería entonces los trigramas para encontrar uno común de la forma tXe, lo que sugerirá fuertemente que X es h y formar the. Adivinando las palabras comunes, digramas y trigramas, y conociendo los patrones probables de las vocales y consonantes, el criptoanalista construye un texto llano tentativo, letra por letra.

Otra estrategia es adivinar una palabra o frase probable.

8.1.3. Cifrados por transposición

Los cifrados por sustitución conservan el orden de los símbolos de texto llano, pero los disfrazan. Los **cifrados por transposición**, en contraste, reordenan las letras pero no las disfrazan. Por ejemplo, en la Figura 8.1 se presenta un cifrado de transposición la transposición columnar. La clave del cifrado es una palabra o frase que no contiene letras repetidas. En este ejemplo, la clave es *MEGABUCK*. El propósito de la clave es numerar las columnas, estando la columna 1 bajo la letra clave más cercana al inicio del alfabeto, y así sucesivamente. El texto llano se escribe horizontalmente, en filas, las cuales se rellenan para completar la matriz si es necesario. El texto cifrado se lee por columnas, comenzando por la columna cuya letra clave es la más baja.

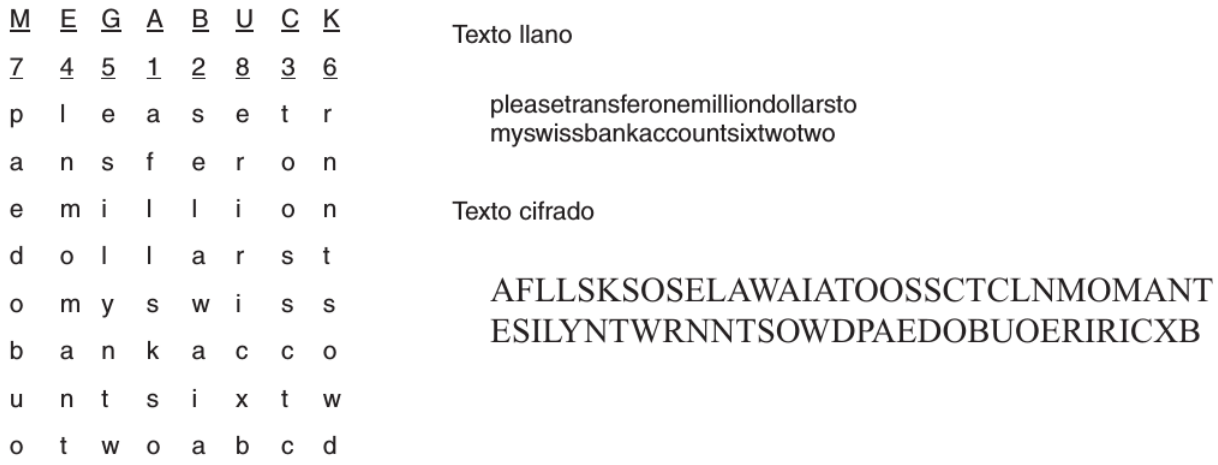

Figura 8.1: Cifrado por transposición. Ejemplo de transposición columnar.

Para descifrar un cifrado por transposición, el criptoanalista debe primero estar consciente de que está tratando con un cifrado de este tipo. Observando la frecuencia de E, T, A, O, I, N, etc., es fácil ver si se ajustan al patrón usual del texto llano. De ser así, es evidente que se trata de un cifrado por transposición, pues en tal cifrado cada letra se representa a sí misma y la distribución de frecuencia permanece intacta. El siguiente paso es adivinar la cantidad de columnas. En muchos casos, puede adivinarse una palabra o frase probable por el contexto del mensaje. Para cada longitud de clave, se produce un grupo diferente de digramas (o trigramas) en el texto cifrado. Buscando las diferentes posibilidades, el criptoanalista con frecuencia puede determinar fácilmente la longitud de la clave. El paso restante es ordenar las columnas.

8.1.4. Rellenos de una sola vez

La construcción de un cifrado inviolable en realidad es bastante sencilla. Primero se escoge una cadena de bits al azar como clave. Luego se convierte el texto llano en una cadena de bits, por ejemplo usando su representación ASCII. Por último, se calcula el OR EXCLUSIVO de estas dos cadenas, bit por bit. El texto cifrado resultante no puede descifrarse, porque en una muestra suficientemente grande de texto cifrado cada letra aparecerá con la misma frecuencia, lo mismo que cada digrama y trigrama, y así sucesivamente. Este método, conocido como relleno de una sola vez, es inmune a todos los ataques actuales y futuros sin importar cuánta potencia computacional tenga el intruso. La razón se deriva de una teoría de la información: simplemente no hay información en el mensaje debido a que todos los textos llanos posibles de una longitud dada son parecidos.

Para cada texto llano ASCII de una dada longitud de caracteres hay un relleno de una sola vez que lo genera. Eso es lo que queremos dar a entender cuando decimos que no hay información en el texto cifrado: es posible obtener cualquier mensaje

con la longitud correcta a partir de él.

En teoría, los rellenos de una sola vez son excelentes, pero en la práctica tienen varias desventajas. Para comenzar, la clave no se puede memorizar, por lo que tanto el emisor como el receptor deben cargar una copia escrita con ellos. Si cualquiera de ellos corre el peligro de ser capturado, es obvio que las claves escritas no son una buena idea. Además, la cantidad total de datos que se pueden transmitir está limitada por la cantidad de claves disponibles. Otro problema es la sensibilidad del método a los caracteres perdidos o insertados. Si el emisor y el receptor pierden la sincronización, de ahí en adelante todos los datos aparecerán distorsionados.

8.2. ALGORITMOS DE CLAVE SIMÉTRICA

La criptografía moderna usa las mismas ideas básicas que la criptografía tradicional (la transposición y la sustitución), pero su orientación es distinta. Tradicionalmente, los criptógrafos han usado algoritmos. Hoy día se hace lo opuesto: el objetivo es hacer el algoritmo de encriptación tan complicado y rebuscado que incluso si el criptoanalista obtiene cantidades enormes de texto cifrado a su gusto, no será capaz de entender nada sin la clave.

Se denominan **algoritmos de clave simétrica** a los utilizan la misma clave para encriptar y desencriptar. En particular, nos enfocaremos en los cifrados en bloques, que toman un bloque de n bits de texto llano como entrada y lo transforman utilizando la clave en un bloque de n bits de texto cifrado.

Los algoritmos criptográficos pueden implementarse ya sea en hardware (para velocidad) o en software (para flexibilidad). Las transposiciones y las sustitucio nes pueden implementarse mediante circuitos eléctricos sencillos. En la Figura 8.2(a) se muestra un dispositivo, conocido como **caja P** (la P significa permutación), que se utiliza para efectuar una transposición de una entrada de 8 bits. Si los 8 bits se designan de arriba hacia abajo como 01234567, la salida de esta caja P en particular es 36071245. Mediante el cableado interno adecuado, puede hacerse que una caja P efectúe cualquier transposición prácticamente a la velocidad de la luz, pues no se requiere ningún cálculo, sólo propagación de la señal. Este diseño sigue el principio de Kerckhoff: el atacante sabe que el método general está permutando los bits. Lo que no sabe es qué bit va en qué lugar, y esto es la clave.

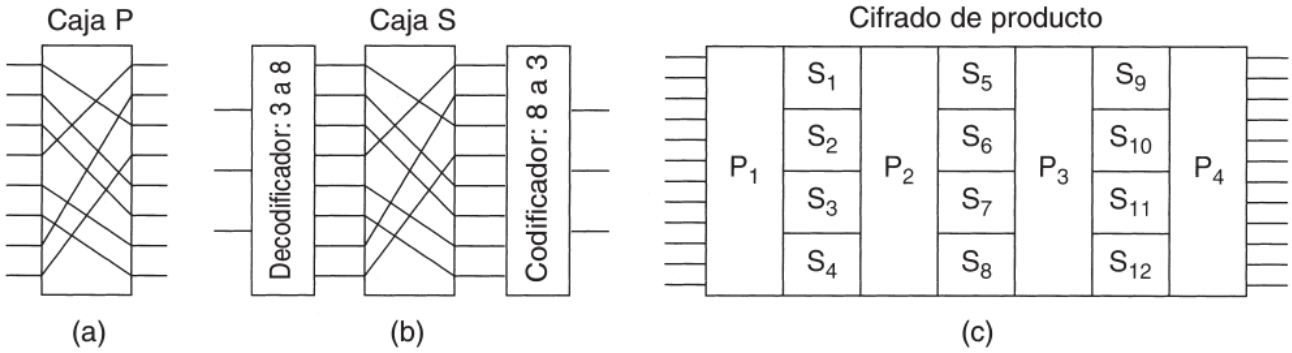

Figura 8.2: Elementos básicos del cifrado de producto. (a) Caja P. (b) Caja S. (c) Producto.

La sustitución se lleva a cabo mediante **cajas S**, como se muestra en la Figura 8.2(b). En este ejemplo, se ingresa un texto llano de 3 bits y sale un texto cifrado de 3 bits. La entrada de 3 bits selecciona una de las ocho líneas de salida de la primera etapa y la establece en 1; las demás líneas son 0. La segunda etapa es una caja P. La tercera etapa codifica en binario nuevamente la línea de entrada seleccionada. Con el alambrado que se muestra, si los ocho números octales 01234567 entraran uno tras otro, la secuencia de salida sería de 24506713. En otras palabras, se ha reemplazado el 0 por el 2, el 1 por el 4, etcétera. Nuevamente, puede lograrse cualquier sustitución mediante el alambrado adecuado de la caja P dentro de la caja S. Además, tal dispositivo puede integrarse en hardware y puede alcanzarse una gran velocidad debido a que los codificadores y decodificadores sólo tienen uno o dos retardos de puerta lógica y el tiempo de propagación a través de la caja P tal vez podría muy bien ser menor a 1 (retardo total del orden de nanosegundo).

La potencia real de estos elementos básicos sólo se hace aparente cuando ponemos en cascada una serie completa de cajas para formar un cifrado de producto, como se muestra en la Figura 8.2(c).

8.2.1. DES – Estándar de Encriptación de Datos

El cifrado DES (*Data Encryption Standard*) fue desarrollado por IBM a principios de los años '70. Se diseñó de forma que, fuera resistente a criptoanálisis y además sencillo para poder ser implementado en un circuito electrónico con la tecnología de ese tiempo.

El cifrado se compone de:

- Bloques de 64 bits (8 bytes). Produce 64 bits de texto cifrado.
- 19 etapas diferentes, donde la transposición final es la inversa de la inicial, y todas las etapas de iteración son funcionalmente iguales.
- Claves de 56 bits.

El descifrado se realiza con la misma clave que el cifrado, ejecutando los pasos en orden inverso.

La complejidad del algoritmo reside en la función $f()$ del bloque de iteración, que realiza operaciones lógicas (XOR), transposiciones, divisiones y duplicaciones. En cada una de las 16 iteraciones, se usa una clave diferente. Antes de iniciarse el algoritmo, se aplica una transposición de 56 bits a la clave. Antes de cada iteración, la clave se divide en dos unidades de 28 bits, cada una de las cuales se desplaza (gira) hacia la izquierda una cantidad de bits dependiente del número de iteración. Los elementos que formarán parte de la función $f()$ se derivan de esta última clave girada aplicándole otra transposición de 56 bits. Además en cada etapa de iteración, se extrae y permuta de los 56 bits un subgrupo de 48 bits diferente para la XOR de la función $f()$.

¿Es segudo DES? Dado un trozo pequeño de texto normal y el texto cifrado correspondiente, se puede encontrar la clave en unas horas con el hardware del DES, mediante una búsqueda exhaustiva del espacio de claves de 2^{56}, por lo tanto **DES no es seguro**.

Doble y Triple DES: El **Doble DES** consiste en ejecutar el DES 2 veces, con 2 claves de 56 bits distintas. Esto proporciona un espacio de claves de 2^{112}. Pero, se ha desarrollado un método de ataque llamado "encuentro a la mitad" que lo hace también vulnerable con 2^{57} operaciones. Ya en 1979, IBM se dio cuenta de que la longitud de la clave DES era muy corta y diseñó una forma de incrementarla de manera efectiva, utilizando cifrado triple (el **Triple DES**). Aquí se utilizan dos claves y tres etapas. En la primera etapa, el texto llano se encripta mediante DES de la forma usual con K_1. En la segunda etapa, DES se ejecuta en modo de desencriptación, utilizando K_2 como la clave. Por último, se realiza otra encriptación DES con K_1. La razón de que se usen dos claves es que incluso los criptógrafos más paranoicos coinciden en que por ahora 112 bits son suficientes para las aplicaciones comerciales. La razón para encriptar, desencriptar y luego encriptar de nuevo es la compatibilidad hacia atrás con los sistemas DES de una sola clave.

8.2.2. AES – Estándar de Encriptación Avanzada

El cifrado AES (*Advanced Encryption Standard*) o Rijndael es el sucesor de DES y T-DES, adoptado como estándar en el NIST (National Institute for Standars and Technology) de EEUU en el año 2000. El algoritmo surgió como resultado de una competición pública de desarrollo de un algoritmo que cumpliera con las siguientes reglas:

1. El algoritmo debe ser un cifrado de bloques simétricos.
2. Todo el diseño debe ser público.
3. Deben soportarse las longitudes de claves de 128, 192 y 256 bits.
4. Deben ser posibles las implementaciones tanto de software como de hardware.
5. El algoritmo debe ser público o con licencia en términos no discriminatorios.

Rijndael soporta longitudes de clave y tamaños de bloque de 128 a 256 bits en pasos de 32 bits. Las longitudes de clave y de bloque pueden elegirse de manera independiente. AES tiene dos variantes: un bloque de 128 bits con clave de 128 bits y un bloque de 128 bits con clave de 256 bits. Una clave de 128 bits da un espacio de claves de $2^{128} \approx 3 \times 10^{38}$ claves. Incluso si la NSA se las arregla para construir una máquina con mil millones de procesadores paralelos, cada uno de los cuales es capaz de evaluar una clave por picosegundo, a tal máquina le llevaría 10^{10} años buscar en el espacio de claves.

Al igual que el DES, Rijndael utiliza sustitución y permutaciones, así como múltiples rondas. El número de rondas depende del tamaño de clave y del tamaño de bloque, y es de 10 para las claves de 128 bits con bloques de 128 bits y aumenta hasta 14 para la clave o el bloque más grande. Sin embargo, a diferencia del DES, todas las operaciones involucran bytes completos, para permitir implementaciones eficientes tanto en hardware como en software.

8.3. ALGORITMOS DE CLAVE PÚBLICA

Históricamente el problema de distribución de claves siempre ha sido la parte débil de la mayoría de los criptosistemas. Sin importar lo robusto que sea un criptosistema, si un intruso puede robar la clave, el sistema no vale nada. Los criptólogos siempre daban por hecho que las claves de encriptación y desencriptación eran la misma.

En 1976, dos investigadores de la Universidad de Stanford, Diffie y Hellman (1976), propusieron una clase nueva de criptosistema, en el que las claves de encriptación y desencriptación eran diferentes y la clave de desencriptación no podía derivarse de la clave de encriptación. En su propuesta, el algoritmo de encriptación (con clave), E, y el algoritmo de desencriptación (con clave), D, tenían que cumplir con los tres requisitos siguientes:

1. $D(E(P)) = P$. Si aplicamos D a un mensaje cifrado, $E(P)$, obtenemos nuevamente el mensaje de texto llano original, P. Sin esta propiedad, el receptor legítimo no podría desencriptar el texto cifrado.
2. Es excesivamente difícil deducir D a partir de E.
3. E no puede descifrarse mediante un ataque de texto llano seleccionado. Este requisito es necesario porque los intrusos pueden experimentar a placer con el algoritmo. En estas condiciones, no hay razón para que una clave de encriptación no pueda hacerse pública.

El método funciona como sigue. Una persona, A, que quiera recibir mensajes secretos, primero diseña dos algoritmos, E_A y D_A, que cumplan los requisitos anteriores. El algoritmo de encriptación y la clave de A se hacen públicos, de ahí el nombre de criptografía de clave pública. Por ejemplo, A podría poner su clave pública en su página de inicio en Web. Utilizaremos la notación E_A para denotar el algoritmo de encriptación parametrizado por la clave pública de A. De manera similar, el algoritmo de desencriptación (secreto) parametrizado por la clave privada de A es D_A. Por otro lado, B hace lo mismo, haciendo pública E_B pero manteniendo secreta D_B.

Se supone que tanto la clave E_A, como la clave E_B, están en un archivo de lectura pública. Ahora, A toma su primer mensaje, P, calcula $E_B(P)$ y lo envía a B. B entonces lo desencripta aplicando su clave secreta D_B –es decir, calcula $D_B(E_B(P)) = P$–. Nadie más puede leer el mensaje encriptado, $E_B(P)$, porque se supone que el sistema de encriptación es robusto y porque es demasiado difícil derivar D_B de la E_B públicamente conocida. Para enviar una respuesta, R, B transmite $E_A(R)$. Ahora, A y B se pueden comunicar con seguridad.

La criptografía de clave pública requiere que cada usuario tenga dos claves: una clave pública, usada por todo el mundo para encriptar mensajes a enviar a ese usuario, y una clave privada, que necesita el usuario para desencriptar los mensajes. Consistentemente nos referimos a estas claves como claves públicas y privadas, respectivamente, y las distinguiremos de las claves secretas usadas en la criptografía convencional de clave simétrica.

8.3.1. El algoritmo RSA

La única dificultad estriba en que necesitamos encontrar algoritmos que realmente satisfagan estos tres requisitos. El algoritmo RSA es un buen método que cumple con estos requisitos, pero su mayor desventaja es que requiere claves de por lo menos 1024 bits para una buena seguridad (en comparación con los 128 bits de los algoritmos de clave simétrica), por lo cual es muy lento. El método se basa en ciertos principios de la teoría de los números y su uso puede resumirse como sigue:

1. Seleccionar dos números primos grandes, p y q (generalmente de 1024 bits).
2. Calcular $n = p \times q$ y $z = (p-1) \times (q-1)$.
3. Seleccionar un número primo con respecto a z, llamándolo d.
4. Encontrar e tal que $e \times d = 1 mod z$.

Para encriptar un mensaje, P, calculamos $C = P^e (mod n)$. Para desencriptar C, calculamos $P = C^d (mod n)$. Puede demostrarse que, para todos los P del intervalo especificado, las funciones de encriptación y desencriptación son inversas. Para ejecutar la encriptación, se necesitan e y n. Para llevar a cabo la desencriptación, se requieren d y n. Por tanto, la clave pública consiste en el par (e, n), y la clave privada consiste en (d, n).

La seguridad del método se basa en la dificultad para factorizar números grandes. Si el criptoanalista pudiera factorizar n (conocido públicamene), podría encontrar p y q y, a partir de éstos, z. Equipado con el conocimiento de z y de e, puede encontrar d usando el algoritmo de Euclides. Afortunadamente, se trata de un problema excesivamente difícil. Por ejemplo, la factorización de un número de 500 dígitos requiere 10^{25} años de tiempo de cómputo utilizando el mejor algoritmo conocido y una computadora con un tiempo de instrucción de 1 μseg.

8.5. ADMINISTRACIÓN DE CLAVES PÚBLICAS

La criptografía de clave pública hace posible que las personas que no comparten una clave común se comuniquen con seguridad. También posibilita firmar mensajes sin la presencia de un tercero confiable. Por último, los compendios de mensajes firmados hacen que verificar fácilmente la integridad de mensajes recibidos sea una realidad. Sin embargo, hay un problema que hemos pasado por alto: si A y B no se conocen entre sí, ¿cómo obtiene cada uno la clave pública del otro para iniciar el proceso de comunicación?

8.5.1. Certificados

Se ha propuesto la siguiente solución: certificar las claves públicas que pertenecen a las personas, empresas y otras organizaciones. Una organización que certifica claves públicas se conoce como **CA** (**Autoridad de Certificación**).

El trabajo fundamental de un certificado es enlazar una clave pública con el nombre de un personaje principal (individuo, empresa, etcétera). Los certificados mismos no son secretos ni protegidos. Por ejemplo, B podría decidir colocar su nuevo certificado en su sitio Web, con un vínculo en la página de inicio que diga: Haga clic aquí para obtener mi certificado de clave pública. El clic resultante podría regresar el certificado y el bloque de la firma (el *hash* SHA-1 firmado del certificado).

Un certificado también se puede utilizar para enlazar una clave pública a un atributo. Por ejemplo, un certificado podría decir si una persona tiene permisos para ver un contenido en particular, pero sin revelar su identidad. Por lo general, la persona que tiene el certificado podría enviarlo al sitio Web, al personaje principal o al proceso que se preocupa por los permisos. El sitio, el personaje principal o el proceso podrían generar a continuación un número aleatorio y encriptarlo con la clave pública del certificado. Si el dueño pudiera desencriptarlo y regresarlo, ésa sería una prueba de que el dueño tenía el atributo establecido en el certificado. De manera alternativa, el número aleatorio podría utilizarse para generar una clave de sesión para la conversación resultante.

8.5.2. X.509

Si todas las personas que desean algo firmado fueran a la CA con un tipo diferente de certificado, administrar todos los formatos diferentes pronto se volvería un problema. Para resolverlo se ha diseñado un estándar para certificados, el cual ha sido aprobado por la ITU. Dicho estándar se conoce como **X.509** y se utiliza ampliamente en Internet. La versión IETF del X.509 se describe en el `RFC 3280`. En esencia, el X.509 es una forma de describir certificados. Los campos principales en un certificado se listan en la Figura 8.3.

Campo	Significado
Versión	Cuál versión del X.509
Número de serie	Este número junto con el nombre de la CA identifican de manera única el certificado
Algoritmo de firma	El algoritmo que se utilizó para firmar el certificado
Emisor	El nombre X.500 de la CA
Validez	Las fechas de inicio y final del periodo de validez
Nombre del sujeto	La entidad cuya clave se está certificando
Clave pública	La clave pública del sujeto y el ID del algoritmo usado para generarla
ID del emisor	Un ID opcional que identifica de manera única al emisor del certificado
ID del sujeto	Un ID opcional que identifica de manera única al sujeto del certificado
Extensiones	Se han definido muchas extensiones
Firma	La firma del certificado (firmada por la clave privada de la CA)

Figura 8.3: Los campos básicos de un certificado X.509.

Por ejemplo, si *Bob* trabaja en el departamento de préstamos del Banco Monetario, su dirección X.500 podría ser:

$$/C = MX/O = BancoMonetario/OU = Prestamo/CN = Bob/$$

donde C corresponde al país, O a la organización, OU a la unidad organizacional y CN a un nombre común. Las CAs y otras entidades se nombran de forma similar. Un problema considerable con los nombres X.500 es que si alguien está tratando de contactar a $bob@bancomonetario.com$ y se le da un certificado con un nombre X.500, tal vez no sea obvio que el certificado se refiera al Bob que se busca. Por fortuna, a partir de la versión 3 se permiten los nombres DNS en lugar de los de X.500.

8.5.3. Infraestructuras de clave pública

El hecho de que una sola CA emita todos los certificados del mundo obviamente no funciona. Podría derrumbarse por la carga y también podría ser un punto central de fallas. Se ha desarrollado una forma diferente para certificar claves públicas. Tiene el nombre general **PKI** (**Infraestructura de Clave Pública**). Una PKI tiene múltiples componentes, entre ellos usuarios, CAs, certificados y directorios. Lo que una PKI hace es proporcionar una forma para estructurar estos componentes y definir estándares para los diversos documentos y protocolos.

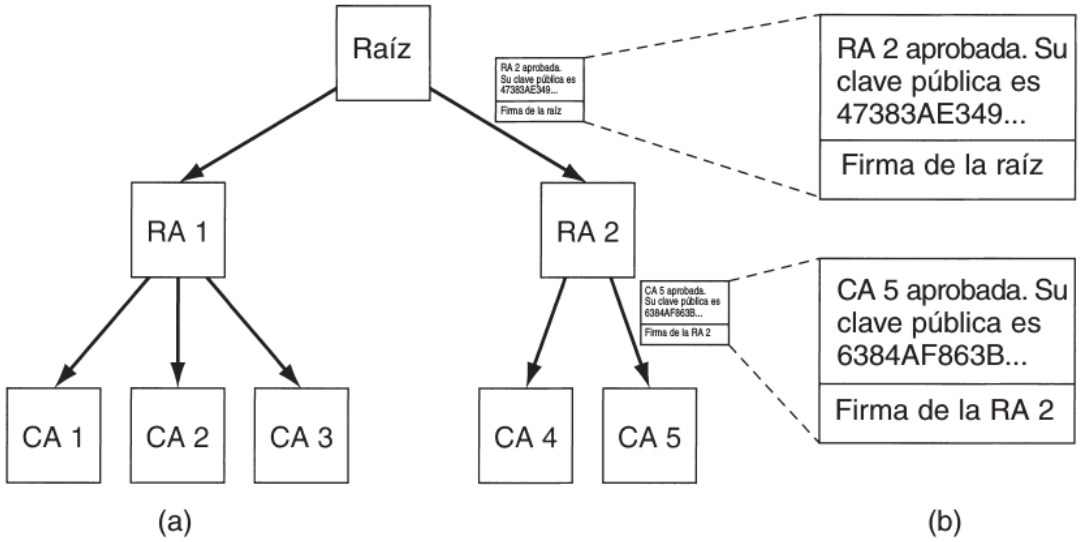

Figura 8.4: (a) Una PKI jerárquica. (b) Una cadena de certificados.

Una forma particularmente simple de PKI es una jerarquía de CAs, como se muestra en la Figura 8.4. En este ejemplo mostramos tres niveles, pero en la práctica podrían ser menos o más. La CA de nivel superior, la raíz, certifica a CAs de segundo nivel, a las que llamaremos **RAs** (**Autoridades Regionales**) debido a que podrían cubrir alguna región geográfica, como un país o un continente. Estas RAs, a su vez, certifican a los CAs reales, las cuales emiten los certificados X.509 a organizaciones e individuos. Cuando la raíz autoriza una nueva RA, genera un certificado X.509 donde indica que ha aprobado la RA, e incluye en él la nueva clave pública de la RA, la firma y se la proporciona a la RA. De manera similar, cuando una RA aprueba una CA, produce y firma un certificado que indica su aprobación y que contiene la clave pública de la CA.

Se supone que todos conocen la clave pública de la raíz.

Debido a que todos los certificados están firmados, se puede detectar con facilidad cualquier intento de alterar el contenido. Una cadena de certificados que va de esta forma a la raíz algunas veces se conoce como cadena de confianza o ruta de certificación. Por supuesto, aún tenemos el problema de quién va a ejecutar la raíz. La solución no es tener una sola raíz, sino tener muchas, cada una con sus propias RAs y CAs. De hecho, los navegadores modernos vienen precargados con claves públicas para aproximadamente 100 raíces, algunas veces llamadas anclas de confianza. De esta forma, es posible evitar tener una sola autoridad mundial confiable.

Pero ahora queda el problema de en qué forma el fabricante del navegador decide cuáles anclas de confianza propuestas son confiables y cuáles no. Queda a criterio del usuario confiar en el fabricante del navegador para elegir las mejores opciones y no simplemente aprobar todas las anclas de confianza deseando pagar sus cuotas de inclusión. La mayoría de los navegadores permiten que los usuarios inspeccionen las claves de la raíz (por lo general en la forma de certificados firmados por la raíz) y eliminen las que parezcan sospechosas.

Directorios

Otro problema de cualquier PKI es en dónde están almacenados los certificados (y sus cadenas hacia un ancla de confianza conocida). Una posibilidad es hacer que cada usuario almacene sus propios certificados. Si bien esto es seguro (es decir,

no hay forma de que los usuarios falsifiquen certificados firmados sin que esto se detecte), también es inconveniente. Una alternativa que se ha propuesto es utilizar DNS como un directorio de certificados. Otros prefieren dedicar servidores de directorio cuyo único trabajo sea manejar los certificados X.509. Tales directorios podrían proporcionar servicios de búsqueda utilizando propiedades de los nombres X.500

Revocaciones

En el mundo real, el otorgante de un certificado podría decidir revocarlo porque la persona u organización que lo posee ha abusado de alguna manera. También puede revocarse si la clave privada del sujeto se ha expuesto o, peor aún, si la clave privada de la CA está en peligro. Por lo tanto, una PKI necesita tratar el problema de la revocación.

Un primer paso en esta dirección es hacer que cada CA emita periódicamente una CRL (lista de revocación de certificados) que proporcione los números seriales de todos los certificados que ha revocado. Puesto que los certificados contienen fechas de vencimiento, la CRL sólo necesita contener los números seriales de los certificados que no han expirado. Una vez que pasa la fecha de vencimiento de un certificado, éste se invalida de manera automática, por lo que no hay necesidad de hacer una distinción entre los certificados que han expirado y los que fueron revocados. Ninguno de esos tipos de certificados puede utilizarse.

Se pueden generar las siguientes complicaciones:

- Tener que consultar una CRL antes de poder usar un certificado para saber si este es válido o no.
- Un certificado revocado podría regularizar su situación y habría que reinstalarlo. Esto elimina una de las mejores características de los certificados que es la de poder usarlos sin contactar a un CA.
- ¿Dónde se almacenarán los CRLs?
- ¿Qué tiempo de vida tendrán los CRLs?
- ¿Cómo se tratarán CRLs grandes?
- Etc.

8.6. SEGURIDAD EN LAS COMUNICACIONES

8.6.1. IPsec

El diseño **IPsec** (**Seguridad IP**), se describe en los `RFCs 2401`, `2402` y `2406`, entre otros. No todos los usuarios desean cifrado (pues éste es costoso computacionalmente). En lugar de hacerlo opcional, se decidió requerir cifrado todo el tiempo pero permitir el uso de un algoritmo nulo. Éste se describe en el `RFC 2410`.

El diseño IPsec completo es una estructura para servicios, algoritmos y granularidades múltiples. La razón para los servicios múltiples es que no todas las personas quieren pagar el precio por tener todos los servicios todo el tiempo, por lo que los servicios están disponibles bajo demanda. Los servicios principales son **confidencialidad, integridad de datos y protección contra ataques de repetición** (un intruso repite una conversación). Todos estos se basan en criptografía simétrica debido a que el alto rendimiento es crucial.

La razón de tener múltiples algoritmos es que un algoritmo que ahora se piensa es seguro puede ser violado en el futuro. Al hacer independiente al algoritmo IPsec, la estructura puede sobrevivir incluso si algún algoritmo particular es violado posteriormente. La razón de tener múltiples granularidades es para hacer posible la protección de una sola conexión TCP, todo el tráfico entre un par de hosts o todo el tráfico entre un par de enrutadores seguros, entre otras posibilidades.

IPsec, aunque se encuentra en la capa IP, es orientado a la conexión. Una “conexión” en el contexto de IPsec se conoce como **SA** (**asociación de seguridad**). Una SA es una conexión símplex entre dos puntos finales y tiene un identificador de seguridad asociado con ella. Si se necesita tráfico seguro en ambas direcciones, se requieren dos asociaciones de seguridad. Los identificadores de seguridad se transportan en paquetes que viajan en estas conexiones seguras y se utilizan para buscar claves y otra información relevante cuando llega un paquete seguro.

Técnicamente, IPsec tiene dos partes principales. La primera describe dos encabezados nuevos que pueden agregarse a paquetes para transportar el identificador de seguridad, datos de control de integridad, entre otra información. La otra parte, **ISAKMP** (**Asociación para Seguridad en Internet y Protocolo de Administración de Claves**), tiene que ver con el establecimiento de claves.

IPsec puede utilizarse en cualquiera de dos modos. En el **modo de transporte**, el encabezado IPsec se inserta justo des-

pués del encabezado IP. El campo Protocolo del encabezado IP se cambia para indicar que un encabezado IPsec sigue al encabezado IP normal (antes del encabezado TCP). El encabezado IPsec contiene información de seguridad, principalmente el identificador SA, un nuevo número de secuencia y tal vez una verificación de integridad del campo de carga.

En el **modo de túnel**, todo el paquete IP, encabezado y demás, se encapsula en el cuerpo de un paquete IP nuevo con un encabezado IP completamente nuevo. El modo de túnel es útil cuando un túnel termina en una ubicación que no sea el destino final. En algunos casos, el final del túnel es una máquina de puerta de enlace de seguridad, por ejemplo, un *firewall* de una empresa. En este modo, el *firewall* encapsula y desencapsula paquetes conforme pasan a través del *firewall*. Al terminar el túnel en esta máquina segura, las máquinas en la LAN de la empresa no tienen que estar consciente de IPsec. Sólo el *firewall* tiene que saber sobre él.

La desventaja del modo de túnel es que agrega un encabezado IP extra, por lo que se incrementa el tamaño del paquete en forma sustancial. En contraste, el modo de transporte no afecta tanto el tamaño del paquete.

8.6.2. Firewalls

La principal función de los *firewalls* (servidores de seguridad) es obligar a todo el trafico de entrada y salida de una red privada a que pase por un único punto. En este punto se coloca un *firewall* que controla todo lo que ingresa y sale.

Este *firewall* puede consistir en un enrutador que contenga tablas con orígenes y destinos (IPs y puertos) aceptados, orígenes y destinos prohibidos y reglas sobre lo que debe hacer con los paquetes que entran y salen. De esta forma se pueden bloquear determinados puertos (TELNET, USENET, etc.) y prohibir el intercambio de información con ciertas IP (e.g. de otros países).

El *firewall* también puede tener una **puerta de enlace de aplicación**. Esta trabaja en capa de aplicación. Por ejemplo, puede examinar cada mensaje que sale o entra (analizando los campos de encabezado, el tamaño del mensaje o incluso en el contenido). Aunque un *firewall* este perfectamente configurado, siempre tiene vulnerabilidades. Por ejemplo, un atacante puede falsificar la dirección de origen de los paquetes que envía para atravesar el *firewall*.

Un espía puede encriptar documentos para enviarlos desde el interior de la LAN hacia afuera. De esta forma la puerta de enlace no puede ver lo que se envía. También existen los ataques por denegación de servicio (DoS) y los ataques por denegación de servicio distribuidos (DDoS).

8.6.3. NAT (Network Address Translation)

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

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

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

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

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

Objeciones:

1. NAT viola el modelo arquitectónico de IP que establece que cada dirección IP identifica una sola máquina globalmente.

Toda la estructura del software de Internet se basa en este hecho. Con NAT, las miles de máquinas pueden usar (y lo hacen) la dirección 10.0.0.1.

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

8.6.4. PAT (Port Address Translation)

Es una característica del estándar NAT, que traduce conexiones TCP y UDP hechas por un *host* y un puerto en una red privada a otra dirección y puerto de la red publica (Internet). Permite que una sola dirección IP publica sea utilizada por varias máquinas de la intranet. Con PAT, una IP publica puede responder hasta a 64000 direcciones privadas. PAT Se apoya en el hecho de que el puerto de origen carece de importancia para la mayoría de los protocolos. Igual que NAT, se sitúa en la frontera entre la red interna y externa, y realiza cambios en la dirección del origen y del receptor en los paquetes de datos que pasan a través de ella. Los puertos (no las IP), se usan para designar diferentes hosts en la red privada.

Cuando un host del intranet manda un paquete hacia fuera, el servicio NAT remplaza la IP interna con la nueva IP del propio servicio tomada de un pool de direcciones publicas. Luego asigna a la conexión un puerto de la lista de puertos disponibles, inserta el puerto en el campo apropiado del paquete de datos y envía el paquete. El servicio NAT crea una entrada en su tabla de direcciones IP internas, puertos internos, IP externas y puertos externos. A partir de entonces, todos los paquetes que provengan del mismo hosts serán traducidos con los mismos puertos y dirección. El receptor del paquete utilizará los IP y puerto recibidos para responder. El traductor revisara la tabla de traducciones, colocara la nueva dirección y el nuevo puerto de destino en el paquete y éste será enviado por la red interna.

8.6.5. Proxy

Es un programa o dispositivo que realiza una acción en representación de otro, esto es, si una hipotética máquina `A` solicita un recurso a una `C`, lo hará mediante una petición a `B`; `C` entonces no sabrá que la petición procedió originalmente de `A`.

Su finalidad más habitual es la de servidor *proxy*, que consiste en interceptar las conexiones de red que un cliente hace a un servidor de destino. De ellos, el más famoso es el servidor *proxy web* (comúnmente conocido solo como *proxy*), pero también existen *proxies* para otros protocolos, como el *proxy* de FTP.

Hay dos tipos de *proxy* atendiendo a quien es el que quiere implementar la política del *proxy*:

- ***Proxy* local**: En este caso el que quiere implementar la política es el mismo que hace la petición. Suelen estar en la misma máquina que el cliente que hace las peticiones. Son muy usados para que el cliente pueda controlar el tráfico y pueda establecer reglas de filtrado que por ejemplo pueden asegurar que no se revela información privada.
- ***Proxy* externo**: El que quiere implementar la política del *proxy* es una entidad externa. Se suelen usar para implementar cacheos, bloquear contenidos, control del tráfico, etc.

Ventajas:

- **Control**: todo lo que se realiza pasa por el *proxy*.
- **Ahorro**: sólo el *proxy* esta preparado para realizar ciertos trabajos.
- **Velocidad**: por ejemplo utilizando caché en el *proxy*.
- **Filtrado**: el *proxy* decide qué dejar pasar y qué no.
- **Modificación**: un *proxy* puede falsificar información, o modificarla siguiendo un algoritmo.

- **Anonimato**: muchos usuarios pueden utilizar la identidad del *proxy*.

Desventajas:

- **Intromisión**.
- **Incoherencia**: ocasionada al usar caché.
- **Irregularidades**: generadas por esta “falsificación” de la identidad (en TCP/IP trae problemas).

Bibliografía

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

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