Servicios UDP

UDP es un protocolo de transporte ligero y simple que proporciona unos servicios mínimos. No está orientado a la conexión, por lo que no tiene lugar un procedimiento de negociación antes de que los dos procesos comiencen a comunicarse. UDP proporciona un servicio de transferencia de datos no fiable; es decir, cuando un proceso envía un mensaje a un socket UDP, el protocolo UDP no ofrece *ninguna* garantía de que el mensaje vaya a llegar al proceso receptor. Además, los mensajes que sí llegan al proceso receptor pueden hacerlo de manera desordenada.

UDP no incluye tampoco un mecanismo de control de congestión, por lo que el lado emisor de UDP puede introducir datos en la capa inferior (la capa de red) a la velocidad que le parezca. (Sin embargo, tenga en cuenta que la tasa de transferencia extremo a extremo real puede ser menor que esta velocidad, a causa de la capacidad de transmisión limitada de los enlaces intervinientes o a causa de la congestión.)

Servicios no proporcionados por los protocolos de transporte de Internet

Hemos organizado los posibles servicios del protocolo de transporte según cuatro parámetros: transferencia de datos fiable, tasa de transferencia, temporización y seguridad. ¿Cuáles de estos servicios proporcionan TCP y UDP? Ya hemos mencionado que TCP proporciona transferencia de datos fiable extremo a extremo. Y también sabemos que TCP se puede mejorar fácilmente en la capa de aplicación, con SSL, para proporcionar servicios de seguridad. Pero en esta breve descripción de TCP y UDP hemos omitido notoriamente hacer mención de las garantías relativas a la tasa de transferencia o la temporización, servicios que *no* proporcionan los protocolos de transporte de Internet de hoy día. ¿Significa esto que las aplicaciones sensibles al tiempo, como la telefonía por Internet, no se pueden ejecutar actualmente en Internet? Evidentemente, la respuesta es no: Internet lleva muchos años albergando aplicaciones sensibles al tiempo. Estas aplicaciones suelen funcionar bastante bien, porque han sido diseñadas para hacer frente a esta falta de garantías de la mejor forma posible. En el Capítulo 9 veremos algunos de estos trucos de diseño. No obstante, un diseño inteligente tiene sus limitaciones cuando el retardo es excesivo, o cuando la tasa de transferencia extremo a extremo es limitada. En resumen, actualmente Internet puede ofrecer servicios satisfactorios a las aplicaciones sensibles al tiempo, pero no puede proporcionar ninguna garantía de tasa de transferencia ni de temporización.

La Figura 2.5 enumera los protocolos de transporte utilizados por algunas aplicaciones populares de Internet. Podemos ver que aplicaciones como el correo electrónico, el acceso remoto a terminales, la Web y la transferencia de archivos utilizan TCP. Estas aplicaciones han elegido TCP principalmente porque este protocolo ofrece un servicio de transferencia de datos fiable, garantizando que todos los datos llegarán finalmente a su destino. Como las aplicaciones de telefonía por Internet (como Skype) suelen tolerar cierto grado de pérdidas, pero requieren una tasa de transferencia mínima para ser efectivas, los desarrolladores de aplicaciones de telefonía por Internet suelen preferir ejecutarlas sobre UDP, evitando así el mecanismo de control de congestión de TCP y la mayor sobrecarga (bits que no forman parte de la carga útil) que los paquetes de datos tienen en TCP. Pero como muchos cortafuegos están configurados para bloquear el tráfico UDP (o la mayor parte del mismo), las aplicaciones de telefonía por Internet suelen diseñarse para usar TCP como solución alternativa, cuando falla la comunicación a través de UDP.

2.1.5 Protocolos de la capa de aplicación

Acabamos de aprender que los procesos de red se comunican entre sí enviando mensajes a sus sockets. Pero, ¿cómo están estructurados estos mensajes? ¿Cuál es el significado de cada uno de los campos de estos mensajes? ¿Cuándo envían los procesos estos mensajes? Estas preguntas nos llevan al ámbito de los protocolos de la capa de aplicación. Un **protocolo de la capa de aplicación** define cómo los procesos de una aplicación, que se ejecutan en distintos sistemas terminales, se pasan los mensajes entre sí. En particular, un protocolo de la capa de aplicación define:

Aplicación	Protocolo de la capa de aplicación	Protocolo de transporte subyacente
Correo electrónico	SMTP [RFC 5321]	TCP
Acceso remoto a terminal	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
Transferencia de archivos	FTP [RFC 959]	TCP
Flujos multimedia	HTTP (p. ej. YouTube)	TCP
Telefonía por Internet	SIP [rfc 3261], RTP [RFC 3550] o propietario (p. ej. Skype)	UDP o TCP

Figura 2.5 ♦ Aplicaciones populares de Internet, sus protocolos de la capa de aplicación y sus protocolos de transporte subyacentes.

- Los tipos de mensajes intercambiados; por ejemplo, mensajes de solicitud y mensajes de respuesta.
- La sintaxis de los diversos tipos de mensajes, es decir, los campos de los que consta el mensaje y cómo se delimitan esos campos.
- La semántica de los campos, es decir, el significado de la información contenida en los campos.
- Las reglas para determinar cuándo y cómo un proceso envía mensajes y responde a los mismos.

Algunos protocolos de la capa de aplicación están especificados en documentos RFC y, por tanto, son de dominio público. Por ejemplo, el protocolo de la capa de aplicación para la Web, HTTP (*HyperText Transfer Protocol* [RFC 2616]), está disponible como un RFC. Si quien desarrolla un navegador web sigue las reglas dadas en el RFC que se ocupa de HTTP, el navegador podrá recuperar páginas web de cualquier servidor web que también se ajuste a las reglas de dicho RFC. Existen muchos otros protocolos de la capa de aplicación que son propietarios y que intencionadamente no están disponibles para todo el mundo. Por ejemplo, Skype utiliza protocolos de la capa de aplicación propietarios.

Es importante diferenciar entre aplicaciones de red y protocolos de la capa de aplicación. Un protocolo de la capa de aplicación es únicamente un elemento de una aplicación de red (¡aunque uno muy importante, desde nuestro punto de vista!). Veamos un par de ejemplos. La Web es una aplicación cliente-servidor que permite a los usuarios obtener documentos almacenados en servidores web bajo demanda. La aplicación Web consta de muchos componentes, entre los que se incluyen un estándar para los formatos de documentos (es decir, HTML), navegadores web (como Firefox y Microsoft Internet Explorer), servidores web (por ejemplo, servidores Apache y Microsoft) y un protocolo de la capa de aplicación. El protocolo de la capa de aplicación de la Web, HTTP, define el formato y la secuencia de los mensajes que se pasan entre el navegador web y el servidor web. Por tanto, HTTP es sólo una pieza (aunque una pieza importante) de la aplicación Web. Otro ejemplo sería una aplicación de correo electrónico Internet, la cual también está constituida por muchos componentes, entre los que se incluyen los servidores de correo que albergan los buzones de los usuarios; los clientes de correo (como Microsoft Outlook) que permiten a los usuarios leer y crear mensajes; un estándar para definir la estructura de los mensajes de correo electrónico y protocolos de la capa de aplicación que definen cómo se pasan los mensajes entre los servidores, cómo se pasan los mensajes entre los servidores y los clientes de correo y cómo se interpretan los contenidos de las cabeceras de los mensajes. El principal protocolo de la capa de aplicación para el correo electrónico es SMTP (*Simple Mail Transfer Protocol*, Protocolo simple de transferencia de correo) [RFC 5321].

Por tanto, el protocolo principal de la capa de aplicación para correo electrónico, SMTP, sólo es un componente (aunque un componente importante) de la aplicación de correo electrónico.

2.1.6 Aplicaciones de red analizadas en este libro

Todos los días se desarrollan nuevas aplicaciones de Internet, tanto de dominio público como propietarias. En lugar de abordar un gran número de aplicaciones de Internet a modo de enciclopedia, hemos decidido centrarnos en unas pocas aplicaciones dominantes e importantes. En este capítulo abordaremos cinco aplicaciones relevantes: la Web, el correo electrónico, el servicio de directorio, los flujos de vídeo y las aplicaciones P2P. En primer lugar veremos la Web, no sólo porque es una aplicación enormemente popular, sino porque también su protocolo de la capa de aplicación, HTTP, es sencillo y fácil de comprender. Después veremos el correo electrónico, que fue la primera aplicación de éxito en Internet. El correo electrónico es más complejo que la Web, en el sentido de que no utiliza uno sino varios protocolos de la capa de aplicación. Después del correo electrónico, abordaremos el sistema DNS, que proporciona un servicio de directorio a Internet. La mayoría de los usuarios no interactúan directamente con DNS; en su lugar, invocan indirectamente a DNS a través de otras aplicaciones (entre las que se incluyen las aplicaciones web, de transferencia de archivos y de correo electrónico). DNS ilustra de forma muy conveniente cómo puede implementarse en la capa de aplicación de Internet un elemento de la funcionalidad de red básica (la traducción entre nombres de red y direcciones de red). Después examinaremos las aplicaciones P2P de compartición de archivos y completaremos nuestro estudio de las aplicaciones analizando los flujos de vídeo a la carta, incluyendo la distribución de vídeo almacenado a través de redes de distribución de contenido. En el Capítulo 9 hablaremos más en detalle de las aplicaciones multimedia, incluyendo la de voz sobre IP y la vídeoconferencia.

2.2 La Web y HTTP

Hasta principios de la década de 1990, Internet era utilizada principalmente por investigadores, profesores y estudiantes universitarios para acceder a hosts remotos; para transferir archivos desde los hosts locales a los hosts remotos, y viceversa, y para recibir y enviar noticias y mensajes de correo electrónico. Aunque estas aplicaciones eran (y continúan siendo) extremadamente útiles, Internet era prácticamente desconocida fuera de las comunidades académica y de investigación. Fue entonces, a principios de la década de 1990, cuando una nueva aplicación importante apareció en escena: la World Wide Web [Berners-Lee 1994]. La Web fue la primera aplicación de Internet que atrajo la atención del público general. Cambió de manera dramática, y continúa cambiando, la forma en que las personas interactúan dentro y fuera de sus entornos de trabajo. Hizo que Internet pasará de ser una de entre muchas redes de datos, a ser prácticamente la única red de datos.

Quizá lo que atrae a la mayoría de los usuarios es que la Web opera *bajo demanda*. Los usuarios reciben lo que desean y cuando lo desean. Es muy diferente a la radio y la televisión, que fuerzan a los usuarios a sintonizar los programas cuando el proveedor de contenido tiene el contenido disponible. Además de estar disponible bajo demanda, la Web posee muchas otras características maravillosas que a todo el mundo le gustan y que todos valoran. Para cualquier persona, es tremendamente fácil publicar información en la Web (todo el mundo puede convertirse en editor con unos costes extremadamente bajos). Los hipervínculos y los motores de búsqueda nos ayudan a navegar a través de un océano de sitios web. Las fotografías y los vídeos estimulan nuestros sentidos. Los formularios, JavaScript, los applets de Java y muchos otros mecanismos nos permiten interactuar con las páginas y sitios. Y la Web y sus protocolos sirven como plataforma para YouTube, para el correo electrónico basado en la Web (como Gmail) y para la mayoría de las aplicaciones móviles de Internet, incluyendo Instagram y Google Maps.

2.2.1 Introducción a HTTP

El corazón de la Web lo forma el **Protocolo de transferencia de hipertexto** (**HTTP**, ***HyperText Transfer Protocol***), que es el protocolo de la capa de aplicación de la Web. Está definido en los documentos [RFC 1945] y [RFC 2616]. HTTP se implementa mediante dos programas: un programa cliente y un programa servidor. Ambos programas, que se ejecutan en sistemas terminales diferentes, se comunican entre sí intercambiando mensajes HTTP. HTTP define la estructura de estos mensajes y cómo el cliente y el servidor intercambian los mensajes. Antes de explicar en detalle HTTP, vamos a hacer un breve repaso de la terminología Web.

Una **página web** (también denominada documento web) consta de objetos. Un objeto es simplemente un archivo (como por ejemplo un archivo HTML, una imagen JPEG, un applet Java o un clip de vídeo) que puede direccionarse mediante un único URL. La mayoría de las páginas web están constituidas por un **archivo base HTML** y varios objetos referenciados. Por ejemplo, si una página web contiene texto HTML y cinco imágenes JPEG, entonces la página web contiene seis objetos: el archivo base HTML y las cinco imágenes. El archivo base HTML hace referencia a los otros objetos contenidos en la página mediante los URL de los objetos. Cada URL tiene dos componentes: el nombre de host del servidor que alberga al objeto y el nombre de la ruta al objeto. Por ejemplo, en el URL

```
http://www.unaEscuela.edu/unDepartamento/imagen.gif
```

`www.unaEscuela.edu` corresponde a un nombre de host y `/unDepartmento/imagen.gif` es el nombre de una ruta. Puesto que los **navegadores web** (como Internet Explorer y Firefox) implementan el lado del cliente de HTTP, en el contexto de la Web utilizaremos los términos *navegador* y *cliente* de forma indistinta. Los **servidores web**, que implementan el lado del servidor de HTTP, albergan los objetos web, siendo cada uno de ellos direccionable mediante un URL. Entre los servidores web más populares se incluyen Apache y Microsoft Internet Information Server.

HTTP define cómo los clientes web solicitan páginas web a los servidores y cómo estos servidores web transfieren esas páginas a los clientes. Más adelante veremos la interacción entre el cliente y el servidor en detalle, si bien la idea general se ilustra en la Figura 2.6. Cuando un usuario solicita una página web (por ejemplo, haciendo clic en un hipervínculo), el navegador envía al servidor mensajes de solicitud HTTP, pidiendo los objetos contenidos en la página. El servidor recibe las solicitudes y responde con mensajes de respuesta HTTP que contienen los objetos.

HTTP utiliza TCP como su protocolo de transporte subyacente (en lugar de ejecutarse por encima de UDP). El cliente HTTP primero inicia una conexión TCP con el servidor. Una vez que la conexión se ha establecido, los procesos de navegador y de servidor acceden a TCP a través de sus

Figura 2.6 ♦ Comportamiento solicitud-respuesta de HTTP.

interfaces de socket. Como se ha descrito en la Sección 2.1, en el lado del cliente la interfaz de socket es la puerta entre el proceso cliente y la conexión TCP; en el lado del servidor, es la puerta entre el proceso servidor y la conexión TCP. El cliente envía mensajes de solicitud HTTP a su interfaz de socket y recibe mensajes de respuesta HTTP procedentes de su interfaz de socket. De forma similar, el servidor HTTP recibe mensajes de solicitud a través de su interfaz de socket y envía mensajes de respuesta a través de la misma. Una vez que el cliente envía un mensaje a su interfaz de socket, el mensaje deja de estar en las manos del cliente y pasa "a las manos" de TCP. Recuerde, de la Sección 2.1, que TCP proporciona un servicio de transferencia de datos fiable a HTTP. Esto implica que cada mensaje de solicitud HTTP enviado por un proceso cliente llegará intacto al servidor; del mismo modo, cada mensaje de respuesta HTTP enviado por el proceso servidor llegará intacto al cliente. Esta es una de las grandes ventajas de una arquitectura en capas: HTTP no tiene que preocuparse por las pérdidas de datos o por los detalles sobre cómo TCP recupera los datos perdidos o los reordena dentro de la red. Ése es el trabajo de TCP y de los protocolos de las capas inferiores de la pila de protocolos.

Es importante observar que el servidor envía los archivos solicitados a los clientes sin almacenar ninguna información acerca del estado del cliente. Si un determinado cliente pide el mismo objeto dos veces en un espacio de tiempo de unos pocos segundos, el servidor no responde diciendo que acaba de servir dicho objeto al cliente; en su lugar, el servidor reenvía el objeto, ya que ha olvidado por completo que ya lo había hecho anteriormente. Dado que un servidor HTTP no mantiene ninguna información acerca de los clientes, se dice que HTTP es un **protocolo sin memoria del estado**. Debemos destacar también que la Web utiliza la arquitectura de aplicación cliente-servidor, descrita en la Sección 2.1. Un servidor web siempre está activo, con una dirección IP fija, y da servicio a solicitudes procedentes de, potencialmente, millones de navegadores distintos.

2.2.2 Conexiones persistentes y no persistentes

En muchas aplicaciones de Internet, el cliente y el servidor están en comunicación durante un periodo de tiempo amplio, haciendo el cliente una serie de solicitudes y respondiendo el servidor a dichas solicitudes. Dependiendo de la aplicación y de cómo se esté empleando, las solicitudes pueden hacerse una tras otra, periódicamente a intervalos regulares o de forma intermitente. Cuando esta interacción cliente-servidor tiene lugar sobre TCP, el desarrollador de la aplicación tiene que tomar una decisión importante: ¿debería cada par solicitud/respuesta enviarse a través de una conexión TCP *separada* o deberían enviarse todas las solicitudes y sus correspondientes respuestas a través de la *misma* conexión TCP? Si se utiliza el primer método, se dice que la aplicación emplea **conexiones no persistentes**; si se emplea la segunda opción, entonces se habla de **conexiones persistentes**. Con el fin de profundizar en esta cuestión de diseño, vamos a examinar las ventajas y desventajas de las conexiones persistentes en el contexto de una aplicación específica, en concreto HTTP, que puede utilizar ambos tipos de conexión. Aunque HTTP emplea conexiones persistentes de manera predeterminada, los clientes y servidores HTTP se pueden configurar para emplear en su lugar conexiones no persistentes.

HTTP con conexiones no persistentes

Sigamos los pasos que permiten transferir una página web desde un servidor a un cliente en el caso de conexiones no persistentes. Supongamos que la página consta de un archivo base HTML y de 10 imágenes JPEG, residiendo los 11 objetos en el mismo servidor. Supongamos también que el URL del archivo base HTML es:

```
http://www.unaEscuela.edu/unDepartmento/home.index
```

Lo que ocurre es lo siguiente:

1. El proceso cliente HTTP inicia una conexión TCP con el servidor `www.unaEscuela.edu` en el puerto número 80, que es el número de puerto predeterminado para HTTP. Asociados con la conexión TCP, habrá un socket en el cliente y un socket en el servidor.
2. El cliente HTTP envía un mensaje de solicitud HTTP al servidor a través de su socket. El mensaje de solicitud incluye el nombre de la ruta `/unDepartmento/home.index`. (Más adelante veremos con más detalle los mensajes HTTP.)
3. El proceso servidor HTTP recibe el mensaje de solicitud a través de su socket, recupera el objeto `/unDepartmento/home.index` de su medio de almacenamiento (RAM o disco), encapsula el objeto en un mensaje de respuesta HTTP y lo envía al cliente a través de su socket.
4. El proceso servidor HTTP indica a TCP que cierre la conexión TCP. (Pero TCP realmente no termina la conexión hasta que está seguro de que el cliente ha recibido el mensaje de respuesta en perfecto estado.)
5. El cliente HTTP recibe el mensaje de respuesta. La conexión TCP termina. El mensaje indica que el objeto encapsulado es un archivo HTML. El cliente extrae el archivo del mensaje de respuesta, examina el archivo HTML y encuentra las referencias a los 10 objetos JPEG.
6. Los cuatro primeros pasos se repiten entonces para cada uno de los objetos JPEG referenciados.

Cuando el navegador recibe la página web, la muestra al usuario. Dos navegadores distintos pueden interpretar (es decir, mostrar al usuario) una página web de formas ligeramente distintas. HTTP no tiene nada que ver con cómo un cliente interpreta una página web. Las especificaciones HTTP ([RFC 1945] y [RFC 2616]) únicamente definen el protocolo de comunicación entre el programa cliente HTTP y el programa servidor HTTP.

Los pasos anteriores ilustran el uso de las conexiones no persistentes, donde cada conexión TCP se cierra después de que el servidor envíe el objeto: la conexión no se mantiene (no persiste) para los restantes objetos. Observe que cada conexión TCP transporta exactamente un mensaje de solicitud y un mensaje de respuesta. Por tanto, en este ejemplo, cuando un usuario solicita la página web, se generan 11 conexiones TCP.

En los pasos descritos anteriormente, hemos sido intencionadamente vagos en lo que respecta a si el cliente obtiene las diez imágenes JPEG a través de diez conexiones TCP en serie o si algunas de dichas imágenes se obtienen a través de conexiones TCP en paralelo. De hecho, los usuarios pueden configurar los navegadores modernos para controlar el grado de paralelismo. En sus modos predeterminados, la mayoría de los navegadores abren entre 5 y 10 conexiones TCP en paralelo y cada una de estas conexiones gestiona una transacción solicitud-respuesta. Si el usuario lo prefiere, el número máximo de conexiones en paralelo puede establecerse en uno, en cuyo caso se establecerán diez conexiones en serie. Como veremos en el siguiente capítulo, el uso de conexiones en paralelo reduce el tiempo de respuesta.

Antes de continuar, vamos a realizar un cálculo aproximado para estimar la cantidad de tiempo que transcurre desde que un cliente solicita el archivo base HTML hasta que recibe dicho archivo completo. Para ello, definimos el **tiempo de ida y vuelta** (**RTT, *Round-Trip Time***), que es el tiempo que tarda un paquete pequeño en viajar desde el cliente al servidor y volver de nuevo al cliente. El RTT incluye los retardos de propagación de los paquetes, los retardos de cola en los routers y switches intermedios y los retardos de procesamiento de los paquetes (estos retardos se explican en la Sección 1.4). Consideremos ahora lo que ocurre cuando un usuario hace clic en un hipervínculo. Como se muestra en la Figura 2.7, esto hace que el navegador inicie una conexión TCP entre el navegador y el servidor web, lo que implica un proceso de "acuerdo en tres fases" (el cliente envía un pequeño segmento TCP al servidor, el servidor reconoce la recepción y responde con otro pequeño segmento TCP y, por último, el cliente devuelve un mensaje de reconocimiento al servidor). Las dos primeras partes de este proceso de acuerdo en tres fases tardan un periodo de tiempo igual a RTT. Después de completarse las dos primeras fases de la negociación, el cliente envía a la conexión TCP el mensaje de solicitud HTTP combinado con la tercera parte de la negociación (el mensaje de reconocimiento). Una vez que el mensaje de solicitud llega al servidor, este envía el

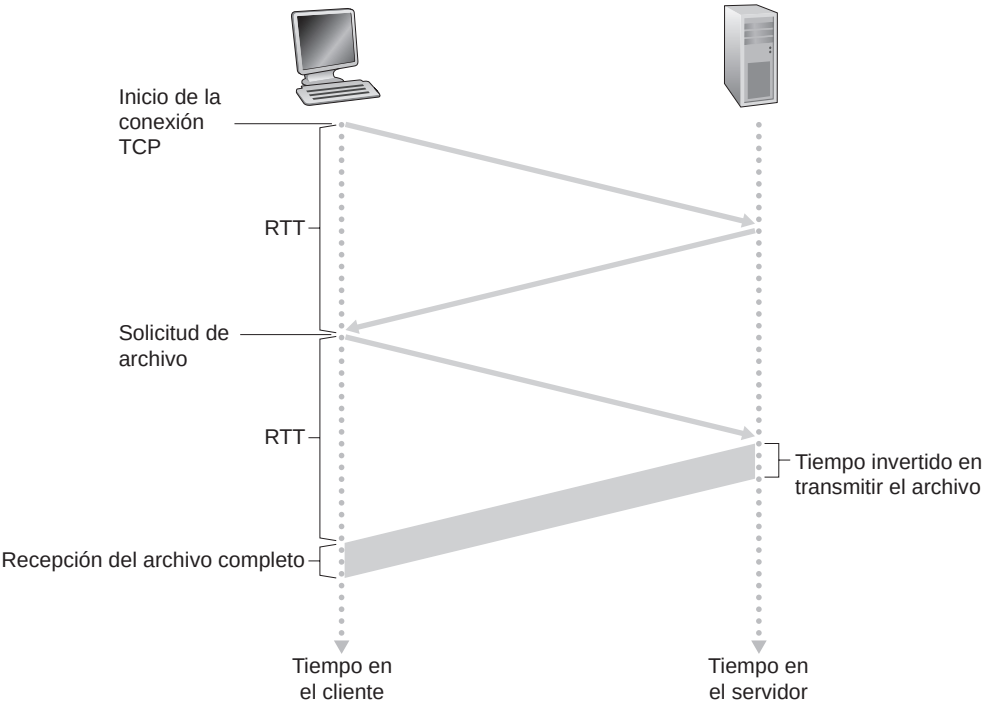

Figura 2.7 ♦ Cálculo aproximado del tiempo necesario para solicitar y recibir un archivo HTML.

archivo HTML a través de la conexión TCP. Este mensaje de solicitud/respuesta HTTP consume otro periodo de tiempo RTT. Luego el tiempo de respuesta total es aproximadamente igual a dos RTT más el tiempo de transmisión del archivo HTML por parte del servidor.

HTTP con conexiones persistentes

Las conexiones no persistentes presentan algunos inconvenientes. En primer lugar, tiene que establecerse y gestionarse una conexión completamente nueva *para cada objeto solicitado*. Para cada una de estas conexiones, deben asignarse buffers TCP y tienen que gestionarse variables TCP tanto en el cliente como en el servidor. Esto puede sobrecargar de forma significativa al servidor web, ya que puede estar sirviendo solicitudes de cientos de clientes distintos simultáneamente. En segundo lugar, como ya hemos explicado, cada objeto sufre un retardo de entrega de dos RTT: un RTT para establecer la conexión TCP y otro RTT para solicitar y recibir un objeto.

Con las conexiones persistentes de HTTP 1.1, el servidor deja la conexión TCP abierta después de enviar una respuesta. Las subsiguientes solicitudes y respuestas que tienen lugar entre el mismo cliente y el servidor pueden enviarse a través de la misma conexión. En concreto, una página web completa (en el ejemplo anterior, el archivo base HTML y las 10 imágenes) se puede enviar a través de una misma conexión TCP persistente. Además, varias páginas web que residan en el mismo servidor pueden enviarse desde el servidor a un mismo cliente a través de una única conexión TCP persistente. Estas solicitudes de objetos pueden realizarse una tras otra sin esperar a obtener las respuestas a las solicitudes pendientes (*pipelining*, procesamiento en cadena). Normalmente, el servidor HTTP cierra una conexión cuando no se ha utilizado durante cierto tiempo (un intervalo de fin de temporización configurable). Cuando el servidor recibe las solicitudes una tras otra, envía los objetos uno tras otro. El modo predeterminado de HTTP utiliza conexiones persistentes con procesamiento en cadena. Más recientemente, HTTP/2 [RFC 7540] amplía la funcionalidad

de HTTP 1.1, permitiendo entrelazar múltiples solicitudes y respuestas en la *misma* conexión, y proporcionando también un mecanismo para priorizar los mensajes de solicitud y respuesta HTTP dentro de dicha conexión. En los problemas de repaso de los Capítulos 2 y 3 compararemos cuantitativamente el rendimiento de las conexiones persistentes y no persistentes. Le animamos también a que consulte [Heidemann 1997; Nielsen 1997; RFC 7540].

2.2.3 Formato de los mensajes HTTP

Las especificaciones HTTP [RFC 1945; RFC 2616]; RFC 7540) incluyen las definiciones de los formatos de los mensajes HTTP. A continuación vamos a estudiar los dos tipos de mensajes HTTP existentes: mensajes de solicitud y mensajes de respuesta.

Mensaje de solicitud HTTP

A continuación se muestra un mensaje de solicitud HTTP típico:

```
GET /unadireccion/pagina.html HTTP/1.1
Host: www.unaEscuela.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

Podemos aprender muchas cosas si miramos en detalle este sencillo mensaje de solicitud. En primer lugar, podemos comprobar que el mensaje está escrito en texto ASCII normal, por lo que cualquier persona con conocimientos informáticos puede leerlo. En segundo lugar, vemos que el mensaje consta de cinco líneas, cada una de ellas seguida por un retorno de carro y un salto de línea. La última línea va seguida de un retorno de carro y un salto de línea adicionales. Aunque este mensaje en concreto está formado por cinco líneas, un mensaje de solicitud puede constar de muchas más líneas o tener tan solo una. La primera línea de un mensaje de solicitud HTTP se denomina **línea de solicitud** y las siguientes son las **líneas de cabecera**. La línea de solicitud consta de tres campos: el campo de método, el campo URL y el campo de la versión HTTP. El campo que especifica el método puede tomar diferentes valores, entre los que se incluyen GET, POST, HEAD, PUT y DELETE. La inmensa mayoría de los mensajes de solicitud HTTP utilizan el método GET. Este método se emplea cuando el navegador solicita un objeto, identificándose dicho objeto en el campo URL. En este ejemplo, el navegador está solicitando el objeto `/unadireccion/pagina.html`. El campo correspondiente a la versión se explica por sí mismo; en este ejemplo, el navegador utiliza la versión HTTP/1.1.

Analicemos ahora las líneas de cabecera de este ejemplo. La línea de cabecera `Host: www.unaescuela.edu` especifica el host en el que reside el objeto. Podría pensarse que esta línea de cabecera es innecesaria, puesto que ya existe una conexión TCP activa con el host. Pero, como veremos en la Sección 2.2.5, las cachés proxy web necesitan la información proporcionada por la línea de cabecera del host. Al incluir la línea de cabecera `Connection: close`, el navegador está diciendo al servidor que no desea molestarse en trabajar con conexiones persistentes, sino que desea que el servidor cierre la conexión después de enviar el objeto solicitado. La línea de cabecera `User-agent:` especifica el agente de usuario, es decir, el tipo de navegador que está haciendo la solicitud al servidor. En este caso, el agente de usuario es Mozilla/5.0, un navegador Firefox. Esta línea de cabecera resulta útil porque el servidor puede enviar versiones diferentes del mismo objeto a los distintos tipos de agentes de usuario (todas las versiones tienen la misma dirección URL). Por último, la línea de cabecera `Accept-language:` indica que el usuario prefiere recibir una versión en francés del objeto, si tal objeto existe en el servidor; en caso contrario, el servidor debe enviar la versión predeterminada. La línea de cabecera `Accept-language:` sólo es una de las muchas cabeceras de negociación del contenido disponibles en HTTP.

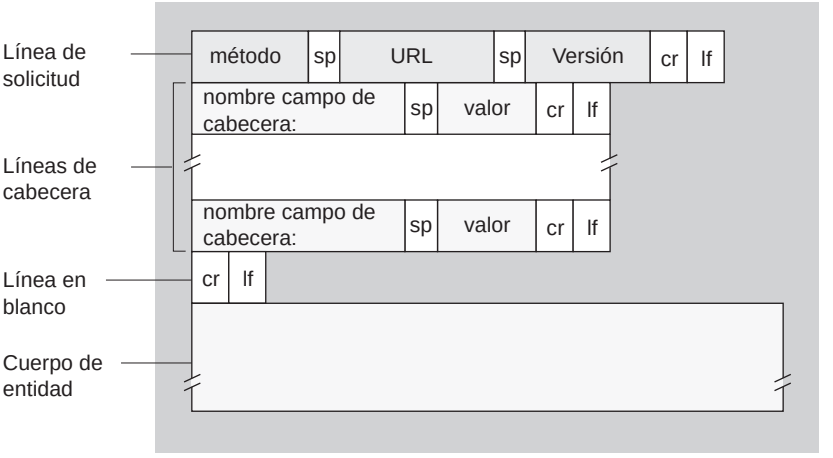

Figura 2.8 ♦ Formato general de un mensaje de solicitud HTTP.

Una vez visto un ejemplo, vamos a estudiar el formato general de un mensaje de solicitud, ilustrado en la Figura 2.8. Podemos comprobar que el formato general es muy similar al usado en el ejemplo anterior. Sin embargo, fíjese en que después de las líneas de cabecera (y el retorno de carro y el salto de línea adicionales) se incluye un "cuerpo de entidad". Este campo queda vacío cuando se utiliza el método `GET`, pero no cuando se usa el método `POST`. A menudo, un cliente HTTP utiliza el método `POST` cuando el usuario completa un formulario; por ejemplo, cuando especifica términos para realizar una búsqueda utilizando un motor de búsqueda. Con un mensaje `POST`, el usuario solicita también una página web al servidor, pero el contenido concreto de la misma dependerá de lo que el usuario haya escrito en los campos del formulario. Si el valor del campo de método es `POST`, entonces el cuerpo de la entidad contendrá lo que el usuario haya introducido en los campos del formulario.

No podemos dejar de mencionar que una solicitud generada con un formulario no necesariamente utiliza el método `POST`. En su lugar, a menudo los formularios HTML emplean el método `GET` e incluyen los datos de entrada (especificados en los campos del formulario) en el URL solicitado. Por ejemplo, si un formulario emplea el método `GET` y tiene dos campos, y las entradas a esos dos campos son `monos` y `bananas`, entonces el URL tendrá la estructura `www.unsitio.com/busquedadeanimales?monos&bananas`. Probablemente habrá visto direcciones URL ampliadas de este tipo, al navegar por la Web.

El método `HEAD` es similar al método `GET`. Cuando un servidor recibe una solicitud con el método `HEAD`, responde con un mensaje HTTP, pero excluye el objeto solicitado. Los desarrolladores de aplicaciones a menudo utilizan el método `HEAD` para labores de depuración. El método `PUT` suele utilizarse junto con herramientas de publicación web. Esto permite a un usuario cargar un objeto en una ruta específica (directorio) en un servidor web determinado. Las aplicaciones que necesitan cargar objetos en servidores web también emplean el método `PUT`. El método `DELETE` permite a un usuario o a una aplicación borrar un objeto de un servidor web.

Mensajes de respuesta HTTP

A continuación se muestra un mensaje de respuesta HTTP típico. Este mensaje podría ser la respuesta al ejemplo de mensaje de solicitud que acabamos de ver.

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
```

```
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(datos datos datos datos datos ...)
```

Examinemos detenidamente este mensaje de respuesta. Tiene tres secciones: una **línea de estado** inicial, seis **líneas de cabecera** y después el **cuerpo de entidad**. El cuerpo de entidad es la parte más importante del mensaje, ya que contiene el objeto solicitado en sí (representado por `datos datos datos datos datos ...`). La línea de estado contiene tres campos: el que especifica la versión del protocolo, un código de estado y el correspondiente mensaje explicativo del estado. En este ejemplo, la línea de estado indica que el servidor está utilizando HTTP/1.1 y que todo es correcto (OK); es decir, que el servidor ha encontrado y está enviando el objeto solicitado.

Veamos ahora las líneas de cabecera. El servidor utiliza la línea de cabecera `Connection: close` para indicar al cliente que va a cerrar la conexión TCP después de enviar el mensaje. La línea de cabecera `Date:` indica la hora y la fecha en la que se creó la respuesta HTTP y fue enviada por el servidor. Observe que esta línea no especifica la hora en que el objeto fue creado o modificado por última vez; es la hora en la que el servidor recupera el objeto de su sistema de archivos, inserta el objeto en el mensaje de respuesta y lo envía. La línea de cabecera `Server:` indica que el mensaje fue generado por un servidor web Apache; es análoga a la línea de cabecera `User-agent:` del mensaje de solicitud HTTP. La línea de cabecera `Last-Modified:` especifica la hora y la fecha en que el objeto fue creado o modificado por última vez. Esta línea `Last-Modified:`, que enseguida estudiaremos en detalle, resulta fundamental para el almacenamiento en caché del objeto, tanto en el cliente local como en los servidores de almacenamiento en caché de la red (también conocidos como servidores proxy). La línea de cabecera `Content-Length:` especifica el número de bytes del objeto que está siendo enviado. La línea `Content-Type:` indica que el objeto incluido en el cuerpo de entidad es texto HTML. (El tipo de objeto está indicado oficialmente por la línea de cabecera `Content-Type:` y no por la extensión del archivo.)

Una vez visto un ejemplo, vamos a pasar a examinar el formato general de un mensaje de respuesta, ilustrado en la Figura 2.9. Este formato general de mensaje de respuesta se corresponde con el del ejemplo anterior. Vamos a comentar algunas cosas más acerca de los códigos de estado y sus descripciones. El código de estado y su frase asociada indican el resultado de la solicitud. Algunos códigos de estado comunes y sus frases asociadas son:

- `200 OK:` La solicitud se ha ejecutado con éxito y se ha devuelto la información en el mensaje de respuesta.
- `301 Moved Permanently:` El objeto solicitado ha sido movido de forma permanente; el nuevo URL se especifica en la línea de cabecera `Location:` del mensaje de respuesta. El software cliente recuperará automáticamente el nuevo URL.
- `400 Bad Request:` Se trata de un código de error genérico que indica que la solicitud no ha sido comprendida por el servidor.
- `404 Not Found:` El documento solicitado no existe en este servidor.
- `505 HTTP Version Not Supported:` La versión de protocolo HTTP solicitada no es soportada por el servidor.

Nota de vídeo

Uso de Wireshark para investigar el protocolo HTTP.

¿Le gustaría ver un mensaje de respuesta HTTP real? ¡Esto es muy recomendable y además es muy fácil de hacer! En primer lugar, establezca una conexión Telnet con su servidor web favorito. A continuación, escriba un mensaje de solicitud de una línea para obtener algún objeto que esté almacenado en ese servidor. Por ejemplo, si tiene acceso a la línea de comandos (*prompt*), escriba:

```
telnet gaia.cs.umass.edu 80
GET /kurose_ross/interactive/index.php HTTP/1.1
Host: gaia.cs.umass.edu
```

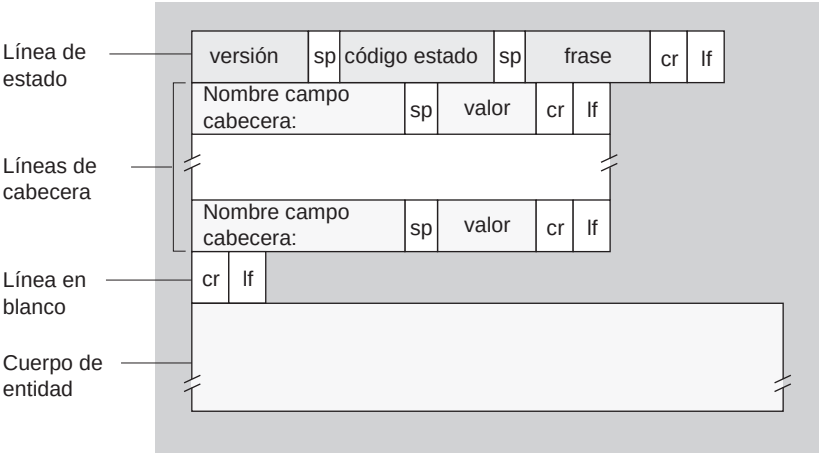

Figura 2.9 ♦ Formato general de un mensaje de respuesta HTTP.

(Pulse dos veces la tecla retorno de carro después de escribir la última línea.) De este modo se abre una conexión TCP en el puerto 80 del host `gaia.cs.umas.edu` y luego se envía el mensaje de solicitud HTTP. Debería ver un mensaje de respuesta que incluya el archivo base HTML de los problemas interactivos de repaso de este libro. Para ver simplemente las líneas del mensaje HTTP y no recibir el objeto, sustituya `GET` por `HEAD`.

En esta sección hemos visto una serie de líneas de cabecera que pueden utilizarse en los mensajes HTTP de solicitud y respuesta. La especificación de HTTP define un gran número de otras líneas de cabecera que pueden ser insertadas por los navegadores, servidores web y servidores de almacenamiento en caché de la red. Hemos cubierto únicamente una pequeña parte de la totalidad de líneas de cabecera disponibles. A continuación veremos unas pocas más y en la Sección 2.2.5 algunas otras, al hablar del almacenamiento en cachés web de la red. Puede leer una exposición enormemente clara y comprensible acerca del protocolo HTTP, incluyendo sus cabeceras y códigos de estado en [Krishnamurty 2001].

¿Cómo decide un navegador qué líneas de cabecera incluir en un mensaje de solicitud? ¿Cómo decide un servidor web qué líneas de cabecera incluir en un mensaje de respuesta? Un navegador generará líneas de cabecera en función del tipo y la versión del navegador (por ejemplo, un navegador HTTP/1.0 no generará ninguna línea de cabecera correspondiente a la versión 1.1), de la configuración del navegador que tenga el usuario (por ejemplo, el idioma preferido) y de si el navegador tiene actualmente en caché una versión del objeto, posiblemente desactualizada. Los servidores web se comportan de forma similar: existen productos, versiones y configuraciones diferentes, que influyen en las líneas de cabecera que se incluirán en los mensajes de respuesta.

2.2.4 Interacción usuario-servidor: cookies

Hemos mencionado anteriormente que un servidor HTTP no tiene memoria del estado de la conexión. Esto simplifica el diseño del servidor y ha permitido a los ingenieros desarrollar servidores web de alto rendimiento que pueden gestionar miles de conexiones TCP simultáneas. Sin embargo, para un sitio web, a menudo es deseable poder identificar a los usuarios, bien porque el servidor desea restringir el acceso a los usuarios o porque desea servir el contenido en función de la identidad del usuario. Para estos propósitos, HTTP utiliza cookies. Las cookies, definidas en [RFC 6265], permiten a los sitios seguir la pista a los usuarios. Actualmente, la mayoría de los sitios web comerciales más importantes utilizan cookies.

Como se muestra en la Figura 2.10, la tecnología de las cookies utiliza cuatro componentes: (1) una línea de cabecera de la cookie en el mensaje de respuesta HTTP; (2) una línea de cabecera de la cookie en el mensaje de solicitud HTTP; (3) el archivo de cookies almacenado en el sistema