unir
LA UNIVERSIDAD DE INTERNET

FORMACIÓN PROFESIONAL

UNIDAD FORMATIVA 4

Networking

Protocolos de comunicaciones

Índice

Protocolos de comunicaciones

Hemos sentado las bases sobre los modelos de redes de ordenadores, estudiando los modelos de referencia OSI y TCP/IP, y conocemos lo básico sobre los niveles más próximos al mundo físico, que son las capas físicas y de enlace de los modelos de referencia.

En esta sección vamos a estudiar con un poco más de detalle los protocolos que, sin duda, resulta fundamental conocer. Hablamos de protocolos a varios niveles, los niveles de los modelos de referencia sobre los que más se podría llegar a trabajar en un momento dado:

- El nivel de **red**, donde el protocolo por excelencia es el protocolo IP, responsable del direccionamiento y enrutamiento de paquetes en la práctica totalidad de redes conocidas, físicas o virtuales, *on-premise o cloud.*
- El nivel de **transporte**, responsable de las comunicaciones entre hosts y de dar servicio de conexión a todas las aplicaciones que usamos habitualmente a través de los protocolos TCP y UDP.
- El nivel de **aplicación**, dentro del cual existen una cantidad inabarcable de protocolos, de los cuales estudiaremos los más conocidos como HTTP, DNS o SSH.

Para terminar la lección, se hará una introducción a los **servicios web**, una tecnología que permite exponer cualquier tipo de servicio en forma de aplicación distribuida y que forma la base sobre la que se sustentan las arquitecturas de microservicios o las aplicaciones web modernas.

Objetivos

- Conocemos los detalles del protocolo de internet y cómo llegan los paquetes a los hosts.
- Se conocen las principales características de TCP y UDP.
- Nos hemos familiarizado con los principales protocolos de nivel de aplicación.
- Conocemos los fundamentos de los servicios web, basados en el protocolo HTTP.

Protocolos de nivel de red

El protocolo IP es el más ampliamente extendido pero no el único que existe en la capa de red, el nivel 3 del modelo de referencia OSI. Este nivel gestiona las opciones relacionadas con el host y el direccionamiento de red, la gestión de subredes e interconexión. También tiene la responsabilidad de enrutar los paquetes desde el origen hasta el destino, tanto dentro como fuera de una subred.

Dos subredes pueden tener direcciones diferentes, esquemas o tipos de direccionamiento no compatibles. Lo mismo ocurre con los protocolos: dos subredes pueden estar operando en diferentes protocolos que no son compatibles con el otro. La capa de red tiene la responsabilidad de encaminar los paquetes desde la fuente hasta destino, mapeando diferentes esquemas de direccionamiento y protocolos.

Direccionamiento

El direccionamiento es una de las principales tareas de la capa de red. Las direcciones de red son **lógicas**, es decir, son direcciones basadas en software que pueden ser cambiadas de acuerdo con las configuraciones que se especifiquen.

Una dirección de red puede identificar a un host (más concretamente, una interfaz de un host) o a una red completa. La dirección de red siempre se configura en una tarjeta de interfaz y casi siempre está mapeada por el sistema con la dirección MAC (dirección de hardware o de la capa 2) de la máquina.

Las direcciones IP se asignan de manera jerárquica, de manera que un host siempre reside bajo una red específica. El host que necesita comunicarse fuera de su subred necesita saber la dirección de la red de destino para iniciar la transmisión.

Cuando un host adquiere la dirección IP del host remoto, envía los paquetes a su **gateway** o puerta de acceso. El gateway será un router equipado con una tabla de rutas para poder decidir cómo enrutar el paquete al host de destino. Las tablas de rutas contienen la siguiente información:

- Dirección de la red de destino.
- Siguiente salto.

Al recibir un paquete, los enrutadores lo envían a su siguiente salto (es decir, uno de los enrutadores adyacentes) en dirección hacia el destino final. Lo mismo hará el siguiente enrutador y, eventualmente, el paquete de datos llegará a su destino.

El protocolo IP

El protocolo IP es el más ampliamente extendido en la capa de nivel 3. Actualmente hay dos versiones funcionando simultáneamente: **IPv4**, que ha gobernado el mundo de internet durante décadas, pero que se está quedando sin espacio de direcciones; e **IPv6**, que se creó para reemplazar IPv4 y se espera que también mitigue las limitaciones de IPv4. Otros ejemplos de protocolos de direccionamiento de red son:

- **IPX:** popularizado con las redes Novell en los años 80 y 90.
- **AppleTalk:** propietario de Apple y descontinuado desde 2009.

IPv4 sigue un esquema de direccionamiento de 32 bits utilizado como mecanismo de direccionamiento en el modelo TCP/IP. Las direcciones IP son jerárquicas, a diferencia de las direcciones Ethernet. Cada dirección de 32 bits se compone de una porción de **red** de longitud variable en los bits más significativos y una porción de **host** en los bits menos significativos. La porción de red tiene el mismo valor para todos los hosts en una red única, como una LAN Ethernet. Esto significa que una red corresponde a un bloque contiguo de espacio de direcciones IP. Este bloque se denomina *prefijo*.

172	.	16	.	254	.	1
10101100	.	00010000	.	11111110	.	00000001

255	.	255	.	0	.	0
11111111	.	11111111	.	00000000	.	00000000

172	.	16	.	0	.	0

Imagen 9. Dirección IP, su representación en octetos y bits, la máscara de red y la dirección de red.

Las direcciones IP se escriben en notación decimal con puntos. En este formato, cada uno de los 4 bytes se escribe en decimal, de 0 a 255. Los prefijos se escriben dando la dirección IP más baja en el bloque y el tamaño del bloque. Por convenio, el tamaño del prefijo en bits se escribe después de la dirección IP.

La longitud del prefijo corresponde a una máscara binaria de unos en la parte de la red, llamada **máscara de red**. Si se aplica un AND lógico entre la dirección IP y su máscara, se obtiene el bloque de red.

Por ejemplo, la dirección hexadecimal de 32 bits AC10FE01 se escribe como 172.16.254.1. Si además sabemos que pertenece a una red con un prefijo de 16 bits, el prefijo de red se indicaría como 172.16.0.0/16.

La ventaja clave de los prefijos es que los routers pueden reenviar paquetes basados únicamente en la porción de red de la dirección, siempre que cada una de las redes tenga un bloque de dirección único. La parte del host no es relevante para los routers porque todos los hosts en la misma red se enviarán en la misma dirección. Solo cuando los paquetes llegan a la red a la que están destinados se reenvían al host correcto.

Esto hace que las tablas de enrutamiento sean mucho más pequeñas de lo que serían de otro modo: mientras que el número de hosts en internet se acerca a los mil millones, los routers necesitan mantener rutas para alrededor de 300.000 prefijos.

Si bien el uso de una jerarquía permite escalar el enrutamiento de internet, tiene dos desventajas:

- Primero, la dirección IP de un host depende de dónde se encuentre en la red. Las direcciones Ethernet se pueden usar en cualquier parte del mundo, ya que se usan para direccionamiento en un medio común, pero cada dirección IP pertenece a una red específica, y los routers solo podrán entregar paquetes destinados a esa dirección a la red, ya que el direccionamiento es global.

- La segunda desventaja es que la jerarquía es un desperdicio de direcciones a menos que se administre cuidadosamente. Si las direcciones se asignan a redes en bloques demasiado grandes, habrá muchas direcciones asignadas que no estarán en uso. Esta distribución no importaría mucho si hubiera muchas direcciones para todos, pero no es el caso. IPv6 es la solución a esta escasez, pero hasta que se implemente ampliamente, habrá una gran presión para asignar direcciones IP de manera muy eficiente.

Subredes

Las direcciones IP están divididas en tres categorías principales:

- **Clase A**: utiliza el primer octeto para las direcciones de red y los últimos tres octetos para el direccionamiento del host.
- **Clase B**: utiliza los primeros dos octetos para las direcciones de red y los dos últimos para el direccionamiento del host.
- **Clase C**: utiliza los primeros tres octetos para las direcciones de red y el último para el direccionamiento del host.

Clase	Tamaño prefijo	#redes	#direcciones	Ejemplo de rango
A	8	128	16.777.216	10.0.0.0 - 10.255.255.255
B	16	16384	65.536	192.168.1.0 - 192.168.1.255
C	24	2097152	256	10.5.1.0 - 10.5.1.255

Tabla 1. Clases de subredes IP.

IPv4 también tiene espacios de direcciones bien definidos para ser utilizados como direcciones privadas (no enrutables en internet) y direcciones públicas (proporcionadas por los ISP y enrutables en internet).

CIDR	Rango de direcciones	# direcciones	Descripción
10.0.0.0/8	10.0.0.0 – 10.255.255.255	16.777.216	Una única red de clase A.
172.16.0.0/12	172.16.0.0 – 172.31.255.255	1.048.576	16 redes de clase B
192.168.0.0/16	192.168.0.0 – 192.168.255.255	65.536	256 redes de clase C

Tabla 2. Rangos de redes privadas.

Gracias a que IP permite la agregación jerárquica, las direcciones IP pueden estar contenidas en prefijos de diferentes tamaños. La misma dirección IP que un router trata como parte de un bloque /22 (que contiene 2^{10} direcciones) puede ser tratada por otro router como parte de un bloque /20 más grande (que contiene 2^{12} direcciones). Depende de cada router tener la información de prefijo correspondiente. Este diseño funciona a base de subredes y se llama CIDR (Classless Inter-Domain Routing, enrutamiento entre dominios sin clase). CIDR permite el diseño de jerarquía de red sin seguir el tamaño estricto de las clases A, B y C.

Protocolos asociados a IP

ARP o Address Resolution Protocol

Durante la comunicación, un host necesita la dirección de capa 2, conocida como MAC, del siguiente salto del paquete. Este salto puede ser el host de destino, si la comunicación no sale de la propia subred, o el gateway que el host ha decidido usar para salir de su subred si el destino está en una red remota. La dirección MAC está físicamente grabada en la tarjeta de interfaz de red de un equipo y nunca cambia. En un entorno virtualizado, una NIC virtual recibe una MAC del hipervisor. Esto puede provocar que haya MACs duplicadas, ya que la asignación es aleatoria, pero los hipervisores tienen mecanismos para evitar que las MACs se repitan en una misma red virtual. En cualquier caso, la NIC virtual de una VM no suele cambiar a lo largo de su vida.

Para conocer la dirección MAC del host remoto en una subred, el ordenador de origen envía un mensaje broadcast ARP preguntando qué host tiene dirección de IP. Debido a que es un mensaje broadcast, todos los hosts del segmento de red, que equivale a un dominio de broadcast, reciben este paquete y lo procesan. El paquete ARP contiene la dirección IP del host de destino al que el host de origen quiere contactar. Cuando un host recibe un paquete ARP que se le ha destinado, responde con su propia dirección MAC.

Una vez que el host recibe la dirección MAC de destino, puede comunicarse con el host remoto utilizando el protocolo de enlace. Este mapeo del MAC al IP se guarda en una caché de ambos hosts. Si en el futuro necesitan comunicarse, pueden directamente dirigirse a su caché ARP.

Internet Control Message Protocol (ICMP)

ICMP es un protocolo de diagnóstico y notificación de errores de red. Pertenece al protocolo IP y utiliza IP como protocolo de encapsulamiento. Después de construir el paquete ICMP, se **encapsula** en un paquete IP.

Los hosts usan ICMP para informar si se produce algún error en la red. Este ofrece opciones para informar de que contiene docenas de mensajes de diagnóstico y reporte de errores.

Los mensajes de ***ICMP-echo*** y ***ICMP-echo-reply*** son los mensajes ICMP más comúnmente utilizados para comprobar la accesibilidad de los hosts de extremo a extremo, utilidad expuesta en la mayoría de los sistemas operativos en la utilidad *ping*:

```
$ ping -v www.unir.net
PING d3bewhrf79fko.cloudfront.net (13.32.91.110): 56 data bytes
64 bytes from 13.32.91.110: icmp_seq=0 ttl=239 time=61.916 ms
64 bytes from 13.32.91.110: icmp_seq=1 ttl=239 time=8.146 ms
64 bytes from 13.32.91.110: icmp_seq=2 ttl=239 time=6.950 ms
64 bytes from 13.32.91.110: icmp_seq=3 ttl=239 time=6.073 ms
```

Cuando un host recibe una solicitud ICMP-echo, está obligado a enviar una respuesta ICMP-echo. No obstante, es habitual que muchos firewalls de red y de host descarten los mensajes de ping como **medida de seguridad**, por lo que el hecho de que un equipo no conteste a un ping no significa que no sea accesible. O también, como en el ejemplo anterior, que la respuesta venga de un servicio en la frontera de la red, ocultando lo que hay detrás.

NAT

Las direcciones IPv4 son **escasas**. Si un ISP dispone de un bloque /16, dándole 65.534 direcciones, el número de clientes que puede tener está limitado a ese número. Un mecanismo para resolver este problema es NAT (***Network Address Translation,*** o traducción de direcciones de red).

La idea básica detrás de NAT es que un ISP asignará a cada cliente una única dirección IP **pública** para navegar por la red y, dentro de la subred interna del cliente, cada equipo obtiene una dirección IP única que se utiliza para enrutar el tráfico local.

Ejemplo: en una red doméstica, esto se traduce en una IP para un teléfono móvil y otra para un ordenador, ambas en la subred de nuestro router, que tendrá asignada otra IP distinta por el proveedor.

Sin embargo, justo antes de que un paquete salga de la red privada y vaya al ISP, se realiza una traducción de la dirección IP interna única a la dirección IP pública compartida.

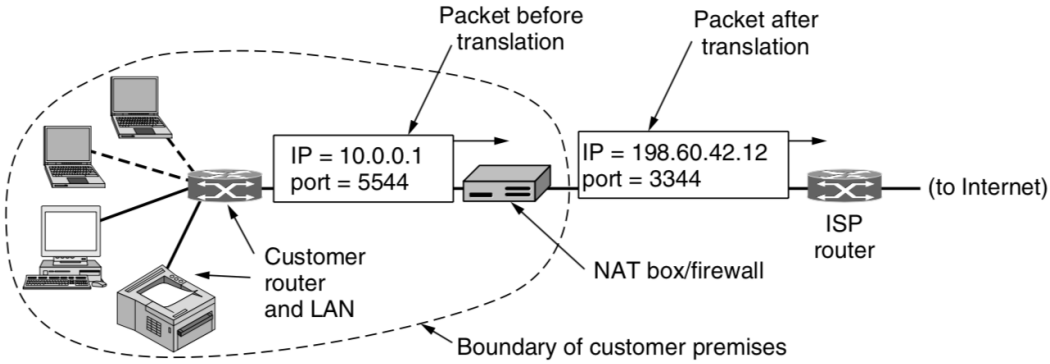


Imagen 10. Funcionamiento de NAT. [1]

Ahora viene el problema: cuando el paquete de respuesta vuelve al origen (por ejemplo, desde un servidor web), se dirige naturalmente a la dirección 198.60.42.12. Entonces ¿cómo sabe el dispositivo NAT con qué dirección interna reemplazarla? Las cabeceras IP **no soportan** NAT nativamente, por lo que la dirección interna no se puede incluir como parte del paquete. La solución en NAT es hacer uso de los puertos de origen y destino (el capítulo siguiente aclarará las dudas sobre los puertos TCP y UDP).

Cada vez que un paquete saliente llega al dispositivo NAT, la dirección de origen 10.x.y.z se reemplaza por la dirección IP pública del cliente. Además, el campo del puerto de origen TCP se reemplaza por un índice en una tabla de traducciones que el equipo NAT mantiene internamente. Esta entrada de la tabla contiene la dirección IP original y el puerto de origen original. Finalmente, las sumas de comprobación (los checksums) del encabezado IP y TCP **se vuelven a calcular** y se insertan en el paquete. Es necesario reemplazar el puerto de origen porque las conexiones de dos máquinas del cliente pueden utilizar el mismo puerto de origen sin por ello entrar en conflicto.

NAT resuelve el problema del escaso número de direcciones disponibles, pero recibe críticas en la industria por las siguientes razones:

- Viola la arquitectura de IP en la que cada dirección IP identifica de forma única una máquina en todo el mundo.

- Rompe el modelo de conectividad de extremo a extremo de internet, que dice que cualquier host puede enviar un paquete a cualquier otro host en cualquier momento. Dado que la tabla de traducciones de un equipo NAT se configura a partir de los paquetes salientes, los paquetes entrantes no pueden aceptarse hasta después de los salientes. Es decir, los equipos locales no son enrutables públicamente **a menos que hayan iniciado una conexión previamente.**

- De manera efectiva, un protocolo de red no orientado a conexión se convierte en un tipo peculiar de red orientada a la conexión. Si el equipo NAT falla y se pierde su tabla de mapeo, se destruyen todas sus conexiones TCP, aun cuando TCP está preparado para soportar la caída de equipos intermedios.

- NAT viola la regla **más fundamental** de la arquitectura de capas: la capa N puede no hacer suposiciones sobre lo que la capa N+1 ha puesto en el campo de carga útil. Este principio básico está ahí para mantener las capas independientes. Además, los procesos que quieren comunicarse a través de internet no están obligados a usar TCP o UDP. Sin embargo, NAT no soportará tráfico que use un protocolo diferente en la capa de transporte.

- Algunas aplicaciones usan múltiples conexiones TCP o puertos UDP en una misma conexión. Por ejemplo, FTP inserta direcciones IP y puertos TCP en el cuerpo del paquete, no en la cabecera, para que el receptor las extraiga y use (los detalles de FTP se estudiarán en el siguiente tema). Dado que NAT no tiene por qué analizar el tráfico de la aplicación, no puede reescribir las direcciones IP ni tenerlas en cuenta.

IPv6

El desgaste generado por IPv4 – más concretamente, por la falta de direcciones IP - dio nacimiento a una nueva generación de protocolo de internet: IPv6. Este asigna direcciones de 128 bits, proporcionando espacio de sobra para el futuro, para ser utilizado en todo el planeta.

IPv6 permite a los dispositivos adquirir una dirección IPv6 y comunicarse dentro de esa subred sin necesidad de depender de otro equipo que le asigne una dirección. Esta configuración automática elimina la necesidad de Dynamic Host Configuration Protocol (DHCP). De esta forma, incluso si el servidor DHCP de esa subred está inactivo o no existe, los hosts pueden comunicarse entre sí.

IPv6 está en fase de transición y se espera que reemplace completamente al IPv4 en los próximos años. En la actualidad, hay pocas redes que usen IPv6 nativamente sin depender de IPV4. Hay algunos mecanismos de transición disponibles para que las redes IPv6 puedan hablar fácilmente en IPv4:

- Implantación dual.

- Túnel.

- NAT-PT.

Protocolos de nivel de transporte

Todos los módulos y procedimientos de transporte o flujo de datos se encuentran dentro de la capa 4. Al igual que las anteriores, esta capa se comunica con su capa de transporte homónima del host remoto.

La capa de transporte ofrece una conexión entre pares y de extremo a extremo entre dos procesos en hosts remotos. La capa de transporte coge los datos de la capa superior (típicamente la capa de aplicación) y luego divide los datos en segmentos de menor tamaño, numera cada segmento y entrega esta información a la capa inferior para su reparto.

Funciones

- Es la primera que divide los datos suministrados por la capa de aplicación en unidades más pequeñas llamadas *segmentos*.
- En el caso de TCP, garantiza que los datos se reciban en la misma secuencia en la que fueron enviados.
- Realiza una entrega de extremo a extremo (end-to-end) entre hosts que pueden o no pertenecer a la misma subred.
- Todos los procesos del servidor que intentan comunicarse a través de la red están equipados con los puntos de acceso a los servicios de transporte (TSAP), también conocidos como *números de puerto de red*.

Comunicación de extremo a extremo

Un proceso en un host identifica a su anfitrión par en la red remota por medio de un número de puerto. Los puertos están muy bien definidos y un proceso sabe con antelación si un proceso está tratando de comunicarse con él. Por ejemplo, cuando un cliente DHCP desea comunicarse con un servidor DHCP remoto, siempre solicita el número de puerto 67. Cuando un cliente DNS desea comunicarse con un servidor de DNS remoto, siempre solicita el número de puerto 53.

La capa de transporte ofrece dos protocolos principales:

- **Protocolo de control de transmisión o TCP:** proporciona una comunicación fiable entre los dos hosts.
- **Protocolo de datagramas de usuario o UDP:** proporciona una comunicación poco fiable entre dos hosts.

TCP

El protocolo de control de transmisión o TCP es uno de los más importantes de la suite de protocolos de internet.

- Es un protocolo fiable, es decir, el receptor siempre envía una confirmación, ya sea positiva o negativa, sobre la recepción del paquete de datos al remitente, de manera que este último siempre está al tanto de si el paquete ha llegado en buenas condiciones o si hace falta reenviarlo.
- Asegura que los datos lleguen a destino en el mismo orden o secuencia en la que han sido enviados.
- Establece una conexión entre los hosts antes de iniciar el envío de los datos de la capa superior.
- Proporciona un mecanismo de comprobación de errores y recuperación.
- Proporciona una comunicación de extremo a extremo, control del flujo y calidad de servicio.
- Funciona en modo cliente/servidor, punto a punto.

Direccionamiento

La comunicación TCP entre dos hosts remotos se realiza mediante números de puerto. Los números de los puertos se establecen entre 0 a 65535, tal como permite el tamaño de 16 bits que tiene el campo en la cabecera. Se clasifican de la siguiente manera:

- Puertos del sistema: 0 – 1023.
- Puertos de usuario: 1024 – 49151.
- Puertos privados / dinámicos: 49152 – 65535.

Gestión de conexiones

La comunicación TCP funciona en el modelo de cliente/ servidor. El cliente inicia la conexión y el servidor lo acepta o lo rechaza. Se usa un protocolo de comunicación de tres vías para la gestión de la conexión, también conocido como *3-way handshake*.

Establecimiento

El cliente inicia la conexión y envía el segmento con un número de secuencia. El servidor lo reconoce con su propio número de secuencia y ACK del segmento del cliente, que es uno más que el número de secuencia del cliente. El cliente, después de recibir el ACK de su segmento, envía un acuse de recibo de la respuesta del servidor.

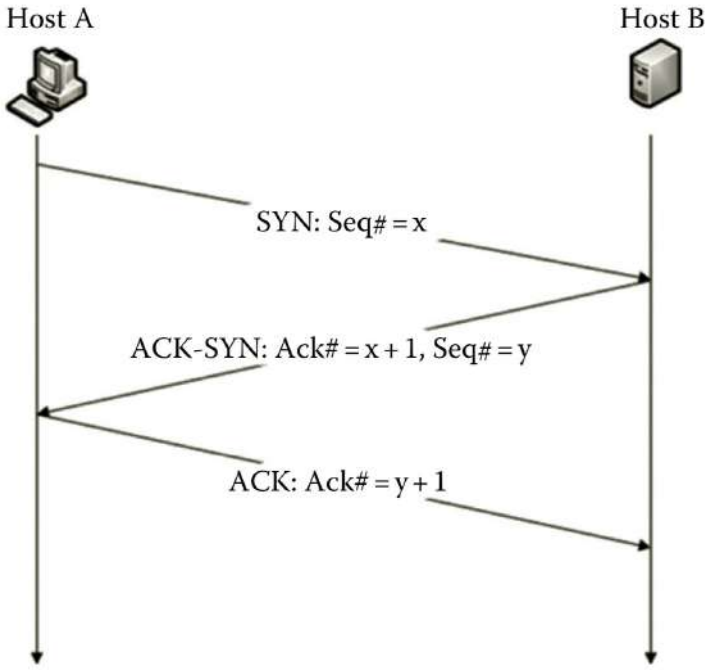


Imagen 12. Establecimiento de sesión TCP: 3-way handshake. [2]

Cualquiera de los servidores y clientes pueden enviar un segmento TCP con el indicador FIN establecido en 1. Cuando el extremo receptor responde dando un acuse de recibo FIN (ACK FIN), esa dirección de TCP, se cierra la comunicación y se libera la conexión.

Gestión del ancho de banda

TCP utiliza el concepto de **tamaño de ventana** para administrar la necesidad de ancho de banda. El tamaño de la ventana indica al remitente en el extremo remoto el número de datos en bytes que el receptor en este extremo puede recibir. TCP utiliza una fase de inicio lento asignando al tamaño de la ventana el valor 1 y aumentando el tamaño de la ventana de forma exponencial después de cada transferencia exitosa.

Por ejemplo, el cliente utiliza tamaño de ventana 2 y envía 2 bytes de datos. Cuando se recibe el acuse de recibo de este segmento, el tamaño de las ventanas se duplica a 4 y el próximo segmento enviado será de 4 bytes. Cuando se recibe el acuse de recibo del segmento de 4 bytes, el cliente establece el tamaño de las ventanas a 8 y así sucesivamente. Si se omite un acuse de recibo, es decir, se pierden datos en la red de tránsito o se recibe NACK, entonces el tamaño de la ventana se reduce a la mitad y la fase de arranque lento comienza nuevamente.

Control de errores y control de flujo

TCP utiliza números de puerto para saber qué proceso de aplicación necesita para transferir el segmento de datos. Junto con eso, utiliza números de secuencia para sincronizarse con el host remoto. Todos los segmentos de datos se envían y reciben con números de secuencia.

El remitente sabe qué último segmento de datos fue recibido por el receptor cuando obtiene ACK. El receptor conoce el último segmento enviado por el remitente refiriéndose al número de secuencia del paquete recibido recientemente.

Si el número de secuencia de un segmento recién recibido no coincide con el número de secuencia que el receptor esperaba, entonces se descarta y se envía como respuesta NACK. Si dos segmentos llegan con el mismo número de secuencia, se contrasta con el valor del timestamp TCP, a fin de tomar una decisión.

Multiplexación

Cuando un cliente TCP inicia una conexión con un servidor, siempre alude a un número de puerto definido, que indica el proceso de aplicación. La sesión TCP también define un puerto en el host de origen, en este caso escogido al azar de la lista de puertos privados que no están en uso. El hecho de que varios procesos puedan usar números de puertos diferentes sirve en efecto de multiplexación a nivel TCP: un cliente puede comunicarse con varios procesos de aplicación de un único host, incluso aunque ese host disponga de una única dirección de nivel 3 y de una única interfaz.

Timeouts

TCP utiliza diferentes tipos de temporizadores para controlar y gestionar diversas tareas:

- **Temporizador de validez (*keep alive*):** este temporizador se utiliza para comprobar la integridad y la validez de una conexión. Cuando expira el tiempo, el host envía una sonda para comprobar si la conexión todavía existe.

- **Temporizador de retransmisión:** este temporizador mantiene una sesión con estado de los datos enviados. Si el acuse de recibo de los datos enviados no se recibe dentro del tiempo de retransmisión, el segmento de datos se vuelve a enviar.

- **Temporizador de persistencia:** la sesión TCP puede ser pausada por cualquiera de los hosts enviando un tamaño de ventana 0. Para reanudar la sesión, un host debe enviar un tamaño de ventana más grande. Cuando expira este temporizador, el host reenvía su tamaño de ventana para que el otro extremo lo sepa. Este temporizador ayuda a evitar los estancamientos en la comunicación.

- **Tiempo de espera:** después de establecer una conexión, ambos hosts esperan un tiempo determinado para finalizar la conexión completamente. Esto es para asegurarse de que el otro extremo ha recibido su solicitud de terminación de conexión. El tiempo de espera máximo puede ser de hasta 240 segundos.

Recuperación ante los fallos

TCP es un protocolo muy fiable. Proporciona un número de secuencia a cada uno de los bytes enviados en un segmento. Proporciona, además, un mecanismo de retroalimentación mediante el que, cuando un host recibe un paquete, está obligado a enviar un mensaje de ACK al paquete que tiene el siguiente número de la secuencia, siempre y cuando no sea el último segmento.

Cuando un servidor TCP pierde la comunicación por una caída a lo largo de la ruta y reinicia su proceso, envía la transmisión TPDU a todos sus hosts. Estos pueden entonces enviar el último segmento de datos y seguir avanzando.

UDP

UDP es el protocolo más sencillo de la capa de transporte disponible para la suite TCP/IP. Se suele decir que UDP es poco fiable, pero realmente ofrece este servicio como tal: delega el control de entrega a la capa de aplicación para, a cambio, evitar la latencia inicial del 3-way handshake y el reenvío de paquetes si no es necesario.

En UDP, el receptor no genera un acuse de recibo del paquete recibido y, por ende, el remitente no lo espera. Esta limitación hace que este protocolo sea poco fiable a la vez que más fácil de procesar.

Requerimiento de UDP

¿Por qué es necesario un protocolo poco fiable para transportar datos? UDP se usa allí donde los paquetes de confirmación o ACK comparten una cantidad significativa de ancho de banda o añaden latencia. Por ejemplo, en el caso de la transmisión de vídeo, se envían miles de paquetes a los usuarios. Si hubiera que pedir acuse de recibo de todos los paquetes, se usaría una gran cantidad de ancho de banda. Además, si se pierde un paquete, puede no ser necesario reenviarlo: en un canal de audio es preferible no entregar parte del contenido (es decir, no reproducir un pequeño intervalo de tiempo) que detener la reproducción completamente y reproducirlo más tarde. De cara a la interacción en tiempo real de usuarios, UDP ofrece una mejor experiencia.

Este mecanismo del protocolo UDP garantiza una entrega fluida de paquetes e, incluso, si alguno de ellos se perdiera en la transmisión, como sucede a veces en vídeo o voz, el impacto no es grave.

Características

- UDP se utiliza cuando la garantía de entrega no es imprescindible o no tiene un impacto significativo.
- Elimina el retardo de establecimiento de conexión de TCP, lo que puede ser útil para aplicaciones sensibles a la latencia.
- Es una buena opción para los datos que fluyen en una única dirección.
- Es simple y adecuado para las comunicaciones basadas en consultas.
- No está orientado a la conexión.
- No proporciona el mecanismo de control de congestión.
- No garantiza la entrega ordenada de los datos.
- Se usa habitualmente en aplicaciones de streaming como VoIP y transmisión multimedia.

Otros protocolos de nivel 4

Aunque siempre se habla de TCP y UDP como *los protocolos* de la capa de transporte como si fueran los únicos disponibles en el mercado, en realidad, hay muchos más protocolos disponibles.

UDP-Lite (Lightweight User Datagram Protocol)

Es un protocolo no orientado a conexión que permite entregar una carga útil de datos potencialmente dañada a una aplicación. Esto es útil para sistemas en los que es preferible tomar las decisiones sobre la integridad de los datos en la propia capa de aplicación, donde se entiende la importancia de los bits, y no en la capa de transporte.

UDP-Lite se basa en UDP, pero a diferencia de este, donde todos o ninguno de los bits del paquete están protegidos por la suma de verificación (el checksum), UDP-Lite permite proteger con checksums parciales que solo cubren parte del paquete de datos y, por lo tanto, puede llegar a entregar paquetes que se hayan dañado parcialmente. Está diseñado para protocolos multimedia, como Voz sobre IP o flujos de vídeo, en los que recibir un paquete con una carga dañada es mejor que no recibir ningún paquete. Para el UDP y TCP, un solo bit con error invalidará la suma de verificación, lo que significa que se debe descartar todo el paquete. De alguna manera, los errores de bit se convierten en errores de paquete completo incluso cuando el daño a los datos es trivial. Para calcular la suma de comprobación, UDP-Lite utiliza el mismo algoritmo que UDP.

SCTP (Stream Control Transmission Protocol)

Es otro protocolo de la capa de transporte. Proporciona algunas de las características de UDP y TCP: está orientado a mensajes como UDP y garantiza la entrega y el orden de los mensajes con control de congestión como TCP. Se diferencia de esos protocolos en que proporciona múltiples rutas redundantes para aumentar la resistencia y la confiabilidad.

Cuando no hay soporte nativo de SCTP en el sistema operativo, es posible hacer un túnel SCTP sobre UDP. También es posible asignar llamadas de sistema de TCP a llamadas SCTP para que las aplicaciones existentes puedan usar SCTP sin modificación.

SCTP se ofrece como alternativa a TCP y UDP porque algunas aplicaciones necesitan una **transferencia garantizada** sin mantener el orden. El bloqueo de TCP hasta que los paquetes pueden ser entregados en orden causa retrasos innecesarios a estas aplicaciones. Para otras aplicaciones, la naturaleza orientada a un flujo de bytes de TCP dificulta el envío de datagramas individuales, pero si necesitan garantía de entrega, no pueden depender de UDP.

Protocolos de nivel de aplicación

Mientras que el número de protocolos de otros niveles es relativamente bajo, la cantidad de protocolos de aplicación está al nivel del número de aplicaciones disponibles. Por tanto, conocer en profundidad todos ellos es una tarea para los administradores más exigentes.

En este tema solo se analizarán algunos de ellos para ofrecer una idea general sobre cómo funcionan y qué tipo de tareas intentan resolver.

Aplicaciones *y protocolos* de aplicación

Existe una ambigüedad en la comprensión de la capa de aplicación y su protocolo. No todas las aplicaciones de usuario se pueden considerar elementos de la capa de aplicación, sino solo aquellas aplicaciones que interactúan con el sistema de comunicación subyacente, representado por la capa de transporte.

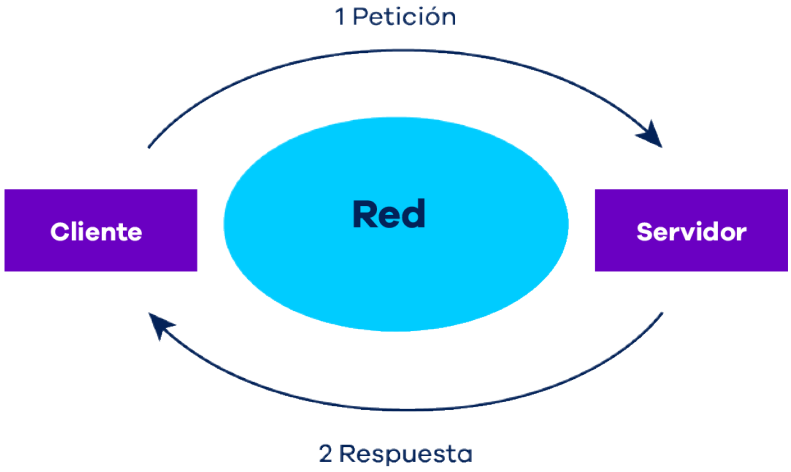
Por ejemplo, un editor de texto es una aplicación de usuario, pero no puede considerarse como un programa de la capa de aplicación. Por otro lado, un navegador web en realidad está usando el protocolo *Hyper Text Transfer Protocol* (HTTP) para interactuar con la red. El protocolo de la capa de aplicación es HTTP.

Otro ejemplo es *File Transfer Protocol* (FTP), que permite al usuario transferir archivos basados en texto o archivos binarios a través de la red. Un usuario puede utilizar este protocolo en cualquier programa de escritorio como FileZilla, CuteFTP o en línea de comandos. El protocolo de capa de aplicación es FTP, no las aplicaciones de usuario.

Modelo cliente-servidor

La manera de interactuar con los diferentes protocolos de aplicación más extendida es a través de un modelo llamado **cliente-servidor**. Tiene su origen en redes empresariales, en las que los ordenadores de los usuarios eran máquinas modestas con poca capacidad de almacenamiento, que se comunicaban a través de la red de la organización a máquinas centrales que almacenaban los datos u otorgaban funcionalidades de procesamiento de datos bajo demanda. Esta conexión se hacía a través de un software llamado *thin client*, algo similar a los escritorios virtuales.

Los **servidores** se suelen imaginar como máquinas potentísimas o enormes bases de datos, aunque en realidad no son más que los proveedores del servicio o de los datos que se desean consumir en un momento dado por los **clientes** usando como medio de comunicación uno o más protocolos de comunicaciones.



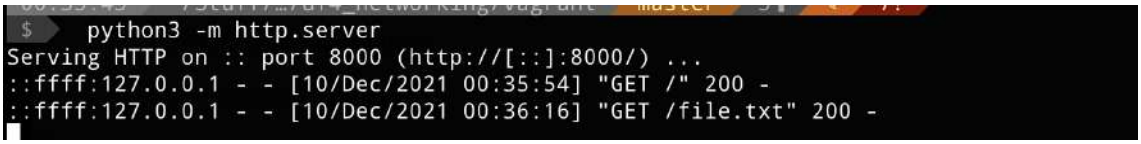
Volviendo al ejemplo de HTTP, el cliente podemos ser nosotros mismos accediendo a la web de un vendedor que, desde su servidor, nos ofrece varios tipos de servicios: listado, consulta de información, compra, etc.

Protocolo TELNET

TELNET (*TELecommunications NETwork*) es un protocolo de comunicación, genérico y bidireccional, con envío de información sin cabeceras extra ni cifrado de ningún tipo desde el cliente al servidor. Su uso habitual es para establecer una sesión de ***terminal virtual de red (NVT)*** a través de una red, de modo que se puedan enviar comandos al puerto destino donde el servidor los interpretará como sea necesario.

El puerto por defecto es el 23, aunque nada nos impide conectarnos al puerto que queramos, en donde puede estar escuchando cualquier servicio, y enviarle comandos: al enviarse los comandos tal cual, pueden ser interpretados por el servidor y retornar una respuesta en caso de tener éxito.

La aplicación *telnet* implementa este protocolo y podemos utilizarla en casi cualquier sistema operativo. Como ejemplo, vamos a levantar un servidor HTTP sencillo con Python y conectarnos a él usando *telnet* desde otro terminal:



En este caso, como HTTP es un protocolo de tipo texto, enviamos un comando GET <fichero> y obtenemos una respuesta válida.

Por esta razón, se puede emplear para ver si un puerto está abierto en un host, ya que en caso contrario recibiremos un mensaje *Connection Refused.*

TELNET **no es seguro**: envía toda su información *en claro* (sin cifrado de ningún tipo), por lo que cualquier analizador de tráfico puede interceptar lo que se envía. Por ello, el protocolo de inicio de sesión remoto por excelencia es *SSH* (*Secure shell),* donde se garantiza la confidencialidad de los datos mediante técnicas criptográficas.

Protocolo SSH

Es el protocolo por defecto para las conexiones seguras a sistemas remotos para tareas de administración y gestión mediante NVTs. Está encaminado a reemplazar a *telnet* y otras aplicaciones de login/copia de ficheros no seguras, como *rcp.* Por defecto, **SSH escucha el puerto 22**.

Desde 1999, la fundación BSD mantiene la suite de aplicaciones OpenSSH, que ofrecen gestión de claves, aplicaciones de conexión remota segura vía protocolo SSH y los demonios correspondientes al lado del servidor.

La seguridad de SSH se basa en dos factores:

- Establecimiento seguro de la conexión entre cliente y servidor, mediante técnicas de clave simétrica o clave pública. Estas técnicas serán introducidas convenientemente en la lección sobre seguridad en redes, pero basta decir por ahora que son **matemáticamente seguras**, siempre que se utilicen los algoritmos adecuados.
- Intercambio seguro de información, mediante cifrado de los datos en tránsito junto con verificación de los mensajes. Esto garantiza la integridad del mensaje recibido, así como la identidad del emisor.

Por estas razones, se puede usar *ssh* para crear un túnel seguro entre dos hosts y comunicarse mediante otros protocolos a través del mismo, ya de forma segura: el siguiente comando crea un túnel ssh en segundo plano, que conecta el puerto local 2020 con el puerto 8000 de la máquina remota.

```
$ ssh -l vagrant 192.168.33.10 -NfL 2020:127.0.0.1:8000
```

Para probar su utilidad, podríamos lanzar un servidor web dentro de esa máquina y hacer el mismo telnet del ejemplo anterior, pero al puerto 2020: de este modo, estaremos usando el protocolo **inseguro** *telnet* de forma segura.

Protocolo FTP

File Transfer Protocol (FTP) es el protocolo más utilizado para la transferencia de archivos en la red, o al menos lo era antes de la popularización de las redes de intercambio *peer-to-peer.* FTP utiliza TCP como capa de transporte y utiliza diferentes puertos en función del modo de funcionamiento:

- El cliente siempre inicia una conexión de control **al puerto TCP 21**.
- En modo **activo**, el cliente abre un socket en un puerto aleatorio e indica al servidor que inicie la transferencia de datos en ese puerto.
- En modo **pasivo**, es el servidor el que abre un socket adicional en un puerto aleatorio y le indica al cliente que lo use para sus transferencias.

El modo activo no funcionará en situaciones con firewalls o en las que el cliente está detrás de un NAT, ya que el puerto en el cliente no será accesible desde la red del servidor.

FTP también es un protocolo basado en texto. Los comandos se transmiten en el canal de control abierto al iniciar la sesión en el puerto 21, y el resultado de los comandos se devuelve bien en el puerto abierto por el servidor en modo pasivo o en el puerto abierto por el cliente en modo activo.

El ejemplo de la Imagen 6 muestra una conexión de control en la consola izquierda en la que se inicia sesión con los comandos USER y PASS. A continuación, se activa el modo pasivo con PASV, a lo que el servidor responde con una dirección IP y un puerto: los dos últimos números de la respuesta a PASV son el número de puerto, pero representado con el valor en decimal de 2 bytes, que en realidad son un número entero de 16 bits:

$$61340 = 239 * 2^8 + 156$$

El siguiente comando es LIST, para solicitar un listado de ficheros, y en la consola derecha se abre un canal TCP al puerto abierto por el servidor para este comando. Para recuperar el archivo hello.txt se activa de nuevo el modo pasivo, se ejecuta el comando RETR hello.txt y se abre otra conexión TCP, también en la consola de la derecha, donde aparece el contenido del fichero. Para terminar, se cierra la conexión de control con el comando QUIT.

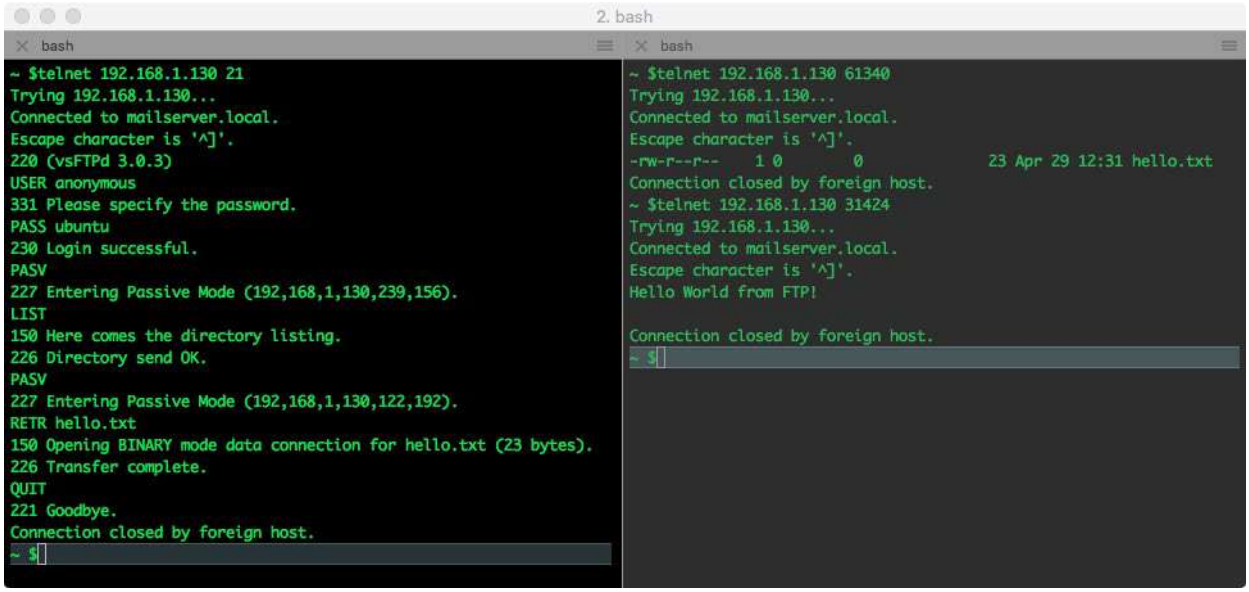


Imagen 6. Sesión FTP sobre telnet.

Aplicaciones

Hay múltiples clientes de FTP: Filezilla, CuteFTP, Cyberduck, etc. Los navegadores web suelen incluir un cliente FTP para poder recuperar ficheros a partir de enlace en las páginas web, aunque no ofrecen toda la funcionalidad de un cliente habitual.

Protocolo SMTP

El Protocolo de transferencia de correo simple (SMTP) se utiliza para el intercambio de mensajes de correo electrónico entre un cliente y un servidor o entre servidores. En un cliente de escritorio, un agente de la aplicación se encarga de gestionar el envío al servidor mediante SMTP. Mientras que el usuario final utiliza SMTP solo para enviar los correos electrónicos, los servidores normalmente utilizan SMTP tanto para enviar como para recibir correos. La recepción de correos en los clientes se suele llevar a cabo con POP o IMAP, como se verá en un capítulo posterior.

Los clientes usan el **puerto TCP 587** como destino para el envío de mensajes, mientras que los servidores usan el **puerto TCP 25** tanto para el envío como para la recepción.

SMTP es un protocolo basado en texto en el que el cliente y el servidor intercambian comandos y respuestas. Los comandos son cadenas de texto seguidas de parámetros, por ejemplo, *MAIL FROM:user@example.com*. Las respuestas empiezan por un código numérico y una descripción, por ejemplo, 250 Ok. Al ser un protocolo de texto es posible simular una sesión usando *telnet*. La Imagen 5 muestra un ejemplo en el que se usan los comandos HELO, MAIL FROM, RCPT TO, DATA y QUIT.

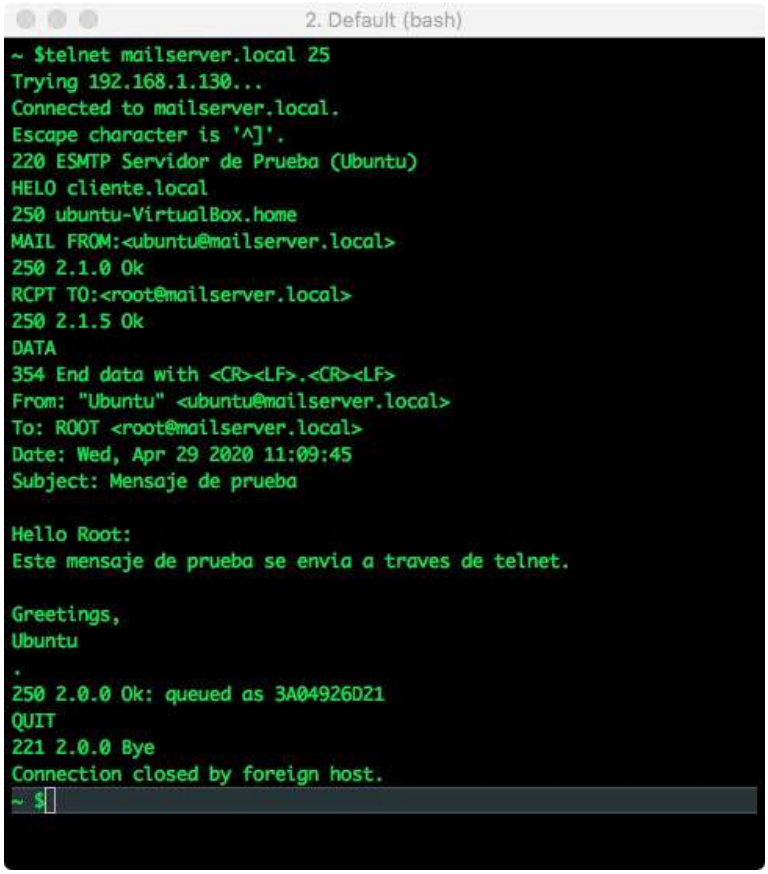


Imagen 5. Sesión SMTP usando telnet.

Una particularidad, que nos podíamos imaginar, es que al usar el puerto 25 y telnet la seguridad es ínfima: de hecho, si intentamos conectarnos con un servicio como Gmail usando SMTP, nos pedirá que iniciemos una sesión TLS y fallará (TLS es un protocolo que incorpora seguridad a las comunicaciones en la capa de aplicación, algo que se verá en las próximas lecciones).

Las aplicaciones de cliente implementan SMTP para el envío de correos y POP e IMAP para la recepción. Se ha extendido el uso de protocolos propietarios como Exchange ActiveSync, que ofrecen funcionalidades adicionales.

A nivel de servidor se pueden citar Postfix para Linux y Microsoft Exchange para Windows.

Protocolo DNS

El protocolo de nombres de dominio (***Domain Name System***, o DNS) facilita el mapeo entre nombres de equipos y direcciones IP. Los programas de software pueden trabajar con direcciones IP sin problema, pero para usuarios finales es más fácil recordar un nombre de equipo como *ftp.rediris.es* que su IP 130.206.13.2.

DNS también facilita las tareas de configuración, ya que el administrador del host ftp.rediris.es puede decidir cambiar la IP del equipo (por una tarea de mantenimiento o por un fallo en el equipo) y no tiene que notificar a todos los usuarios del cambio de IP: simplemente debe actualizar el registro en el servidor DNS y todos los clientes recibirán la nueva IP en la próxima consulta.

El volumen de nombres de equipo en internet es muy elevado, así que DNS trabaja con un modelo **jerárquico** basado en dominios y una base de datos distribuida. De este modo, si un equipo necesita resolver la dirección del equipo ftp.rediris.es, envía una solicitud DNS a su servidor DNS local. Este no tiene por qué conocer la dirección, pero enviará a su vez una solicitud a uno de los servidores raíz preguntando por el dominio, rediris.es. El servidor raíz contesta con la dirección de otro servidor DNS, el que aloja el dominio rediris.es. El servidor DNS local podrá entonces preguntar al servidor que aloja el dominio rediris.es por el nombre completo, ftp.rediris.es.

Los registros DNS no se limitan a nombres de equipo. Los tipos más relevantes de registros DNS se pueden consultar en la Tabla 2.

Registros DNS	
A	Dirección de un host
NS	Nombre de servidor DNS acreditado para esta zona
CNAME	Alias de un nombre de host
MX	Nombre de servidor de correo asociado al dominio
TXT	Cadena de texto

Tabla 2. Tipos de registros DNS.

Protocolo

El protocolo trabaja sobre UDP usando **el puerto 53 por defecto**. La elección de UDP se debe principalmente a razones de rendimiento. El número de peticiones DNS necesarias para resolver un único nombre es elevado y el contenido de cada petición es relativamente pequeño, así que el tamaño de las cabeceras TCP es comparable y añade una sobrecarga importante. Además, las peticiones DNS son una herramienta auxiliar antes de iniciar el protocolo deseado, por lo que es deseable una reducción en la latencia inicial. Aunque formalmente DNS no es confiable, no hay más que usar internet para comprobar que funciona como se espera.

Aunque las peticiones de resolución de nombres usan UDP como protocolo de transporte, los servidores DNS se comunican entre ellos mediante TCP en el puerto 53. Esta comunicación se usa para el intercambio y sincronización de zonas DNS. Este mecanismo permite que una actualización de un registro DNS se pueda replicar a otros servidores sin intervención manual.

Los sistemas operativos incorporan llamadas de sistema para la resolución de nombres sin necesidad de que las aplicaciones lo implementen independientemente. De manera práctica, es habitual usar las utilidades de línea de comandos *ping* y *host* para obtener la IP de un nombre de host.

Una utilidad avanzada es *dig*: permite ejecutar peticiones de diferentes tipos usando servidores DNS específicos (es decir, no los configurados por defecto en el sistema operativo), aunque se puede utilizar para más cosas. Veamos unos ejemplos:

```
# Dirección de correo de StackOverflow, según el DNS de google
$ dig  @8.8.8.8 mx stackoverflow.com

; <<>> DiG 9.11.3-1ubuntu1.8-Ubuntu <<>> @8.8.8.8 mx stackoverflow.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 40320
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;stackoverflow.com.                IN      MX

;; ANSWER SECTION:
stackoverflow.com.        300     IN      MX      10 alt3.aspmx.l.google.com.
stackoverflow.com.        300     IN      MX      1 aspmx.l.google.com.
stackoverflow.com.        300     IN      MX      5 alt1.aspmx.l.google.com.
stackoverflow.com.        300     IN      MX      10 alt4.aspmx.l.google.com.
stackoverflow.com.        300     IN      MX      5 alt2.aspmx.l.google.com.

;; Query time: 30 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Fri Dec 10 21:55:44 UTC 2021
;; MSG SIZE  rcvd: 161

# Mi dirección IP en formato corto, MUY ÚTIL como alias
$ dig +short myip.opendns.com @resolver1.opendns.com
79.59.48.102
```

En el mercado hay múltiples opciones de servidores DNS. Algunos de los habituales son BIND para equipos Unix y Linux y la implementación de DNS de Microsoft incorporada en Windows Server.

Protocolo HTTP

El protocolo quizá más conocido a este nivel, pues es el que nos permite navegar por diferentes páginas de internet, abriendo sus recursos en diferentes navegadores web o usando aplicaciones específicas para móvil que ofrecen servicios personalizados por este canal.

El protocolo de transferencia de hipertexto (*Hyper-Text transfer protocol*) no es más que otro protocolo de texto, que **por defecto escucha en el puerto 80** y que nos permite traernos un recurso determinado de un servidor estableciendo una conexión fiable a través de TCP, nada que no hayamos visto hasta ahora. El cómo lo hace, sin embargo, merece un poco de estudio.

- HTTP admite diferentes *verbos* o modos de petición: GET, POST, PUT, DELETE, HEAD y OPTIONS. Cada uno de estos modos indica al servidor que
  - le vamos a enviar parámetros extra de una manera determinada
  - y que esperamos recibir la respuesta, también, de una manera determinada.
- HTTP introduce las cabeceras *(Headers)* tanto a las peticiones como a las respuestas. Las cabeceras son campos de **metadatos**, algunos estándar como *Content-Type* (indica el formato de los datos que acompañan al cuerpo de un mensaje) o *Authorization* (cuyo valor corresponde con un modo de autenticación determinado con el servidor) y otras que se usan para indicar cierta información específica a los servicios, extendiendo así el *standard* HTTP.
- HTTP permite añadir parámetros a las peticiones, directamente en la URL (Uniform *resource locator)* o en el cuerpo del mensaje, en un formato especificado por la cabecera correspondiente.
- HTTP define unos códigos de respuesta que se pueden interpretar muy fácilmente por humanos, pero especialmente por máquinas: son números de tres cifras, donde 10x o 20x indican que todo ha ido bien, 30x indica algún problema que el cliente debe resolver, 40x indica algún problema en la petición hecha por el cliente y 50x indica un problema irrecuperable del lado del servidor. La lista completa puede consultarse aquí.

La potencia combinada de todas estas características la estudiaremos con más detalle en la sección correspondiente a los servicios Web.

Uniform Resource Locator (URL)

Usando una URL podemos **identificar** cualquier recurso de cualquier servidor, siempre que sepamos localizarlo (DNS se encarga de eso) y tengamos acceso a esa red. Una vez tenemos la IP del servidor, la URL representa un recurso único que este expone, ya sea un fichero de música, un documento *.pdf* o un documento HTML, a través de un protocolo determinado que dicho recurso también especifica (y que **no es** exclusivo de HTTP).

La estructura de una URL es la siguiente (las secciones entre **[corchetes]** son opcionales)

```
protocolo://[user:pass@]host[:puerto]/ruta/al/recurso[?param1=value1[&param2=value2…]][#fragmento]
```

- **protocolo**: define el protocolo de aplicación a usar para solicitar el recurso. Valores usuales son *http, https, ftp, mailto…* También pueden ser aplicaciones específicas, como *chrome://, spotify://…* Estas URLs suelen hacer que el recurso se maneje de una manera determinada al abrirlo con determinados dispositivos (como, por ejemplo, abrir Spotify en el móvil).
- **user/pass**: en caso de que el servicio necesite autenticación, se puede enviar directamente en la URL (ojo, no es seguro).
- **host**: el nombre canónico del servidor donde se aloja, cuya IP se consultará usando DNS.
- **puerto**: en caso de que no se use el puerto por defecto para el protocolo elegido, se especifica aquí dónde se escucha.

- **/ruta/al/recurso:** el path que usará el servidor para ubicar el recurso.
- **parámetros extra**: se pueden añadir parámetros extra a partir del símbolo ***?***, separados por ***ampersands (&).*** Estos parámetros serán empleados por el servicio para lo que decida, no forman parte de ningún protocolo.
- **fragmento:** Si se especifica, permite referenciar tan solo una parte del recurso. Por ejemplo, nos puede llevar a una parte concreta de una página web o a un instante concreto de un vídeo.

HyperText Markup Language (HTML)

La inmensa mayoría de los ficheros que se intercambian en internet son ficheros HTML. Los ficheros de tipo ML son *Lenguajes de Marcado*, que se caracterizan porque usan unas *marcas* o etiquetas para dotar de cierta semántica a sus contenidos, semántica que es interpretada por herramientas mucho mejor que por humanos.

```
<section>
 <article>
  <title>Ubuntu 21.10 is out!!</title>
  <description>blablablabla</description>
  <img src="images/stock/ubuntu_21_10.png" />
 </article>
</section>
```

Ejemplo de lenguaje de marcado.

Esto abrió la puerta a que se creasen unas aplicaciones llamadas ***navegadores web*** que, usando HTTP para recuperar recursos, interpretan esas etiquetas y *renderizan* su contenido en forma de imágenes, columnas, listas, etc. De hecho, usando HTML se pueden especificar recursos **extra** que solicitar a la hora de mostrar una web:

- Las hojas de estilo en cascada (*CSS, cascading style sheets)* que indican al navegador cómo y dónde debe mostrar cada elemento.
- Los scripts de navegador, ficheros JavaScript que se ejecutan cuando se descargan y permiten dotar de todo tipo de interacciones a cualquier web: anuncios específicos, tracking de usuarios, interacciones entre elementos… Las posibilidades son infinitas.

Dada la complejidad y el tedio de escribir estos documentos de forma manual, existen infinidad de maneras de generar HTML (en base a plantillas, de forma dinámica, estática…). La base sobre la que se sustenta internet son todos los motores y servidores web que generan, almacenan y emiten HTML en base a eventos.

El alcance de lo que se puede hacer con HTML, CSS y JavaScript está totalmente fuera del ámbito de este curso, pero resulta un campo de la Ingeniería Software en constante evolución, de mucha complejidad pero desde luego muy interesante, recomendable para cualquiera.

HTTPS

Como se ha repetido en varias ocasiones, HTTP no es seguro. Y el hecho de que sea un protocolo omnipresente y que se emplea para tantas operaciones sensibles es un problema de seguridad que lo hace inviable para su uso extendido.

Para paliar esto, se creó el protocolo ***HTTP Secure*** *(HTTPS)*, que es básicamente una versión segura de HTTP que corre sobre TLS (*Transport Layer Security*) para que las comunicaciones estén apropiadamente cifradas.

En esta sección no entraremos en detalles relativos a la seguridad, pero podemos adelantar que TLS proporciona los mecanismos de seguridad necesarios para que las comunicaciones sean ya confiables.

El puerto por defecto para HTTPS es el 443, y el esquema de las URLs cambia cuando se usa, comenzando estas por *https://.*

Servicios Web

Una arquitectura basada en ***web services*** es un modelo de interoperabilidad de sistemas donde una serie de servicios **distribuidos,** que exponen ciertas funcionalidades, se comunican entre sí utilizando el protocolo HTTP.

Cuando un sistema ofrece una interfaz HTTP mediante la cual se puede interactuar con determinados servicios internos, consultar datos, modificar estados y demás operaciones, se dice que **expone una API***,* del inglés *Application Programming Interface.*

Un caso concreto son los sistemas REST (de *Representational State Transfer*). Aunque no hay una definición concreta, se puede hablar de REST como un estilo de arquitectura con una serie de restricciones para el desarrollo de interfaces de aplicaciones [8], concretamente:

- Los recursos se identifican con un esquema de **URIs** (Uniform Resource Identifier), similar a las URLs de las páginas web.
- Los recursos tienen una representación en bytes y unos metadatos asociados.
- Solo unos pocos métodos son permitidos sobre los recursos y tienen el mismo significado sobre todos los recursos.
- Las interacciones no mantienen estado: cada petición termina completamente, bien con éxito o con fallo.
- El comportamiento idempotente de las operaciones es deseable.
- Entidades intermedias para proxy, caché u otras interacciones son deseables.

En la práctica, es habitual encontrar interfaces REST basadas en HTTP. Cuanto más se ciña una API al estándar HTTP, más fácil será que otras aplicaciones se integren con ella, ya que aprovechará el comportamiento de clientes HTTP ya existentes.

La razón de usar HTTP reside en la potencia del sistema de verbos, cabeceras y códigos de respuesta que integra el protocolo, junto con el concepto de URI para designar los recursos de un servicio.

Ejemplo de API REST

Imaginemos un servicio web en edixconsulting.com que,entre otras cosas, maneja datos de usuarios. Estos datos se pueden consultar y modificar a través de una API de usuarios, que podría estar accesible de este modo:

http://edixconsulting.example:1234/api/v1/users

Esta URL contiene un path (opcional, pero muy frecuente) que indica que no accedemos a la ruta habitual dentro de ese servidor, sino a una API. Y además, a la versión 1 de esa API, algo útil en el caso de que el servicio evolucione con el tiempo y decidamos migrar el esquema.

Por simplicidad, imaginemos que un usuario se puede representar con el siguiente JSON:

```
{
  "id": "juan",
  "age": 40,
  "email": "juan@edix.com",
  "last_login": "2021 Nov 27, 09:30"
}
```

Pues bien, usando una API REST, podríamos **consultar** una lista de usuarios o un usuario concreto, usando un endpoint *semántico:*

```
$ curl http://edixconsulting.com/api/v1/users
```

```
[
 {
  "id": "Juan",
  "age": 40,
  "email": "juan@edix.com",
  "last_login": "2021 Nov 27, 09:30"
 }, {
  "id": "antonio",
  "age": 31,
  "email": "antonio@edix.com",
  "last_login": "2021 Nov 01, 00:11"
 }
]
```

```
$ curl http://edixconsulting.com/api/v1/users/juan
```

```
{
  "id": "juan",
  "age": 40,
  "email": "juan@edix.com",
  "last_login": "2021 Nov 27, 09:30"
}
```

Podríamos ahora crear un nuevo usuario o modificar datos de uno existente. Para ello, lo apropiado sería aprovechar la potencia de HTTP, manteniendo el mismo endpoint pero cambiando el método HTTP por uno apropiado, según el protocolo:

```
$ curl -v -XPOST http://edixconsulting.com/api/v1/users/paco --data '{
 age: 27,
 email: "paco@edix.com"
}'

< HTTP/1.1 200 OK
< Date: Sat, 04 Dec 2021 12:22:55 GMT
< Content-Type: application/json

$ curl -v -XPUT http://edixconsulting.com/api/v1/users/paco --data '{
 age: 32
}'

< HTTP/1.1 200 OK
< Date: Sat, 04 Dec 2021 12:25:00 GMT
< Content-Type: application/json

$ curl http://edixconsulting.com/api/v1/users?name=paco
[
 {
  "id": "Paco",
  "age": 32,
  "email": "paco@edix.com",
  "last_login": ""
 }
]
```

Para terminar, se podría crear un endpoint para borrar usuarios, usando el verbo DELETE de HTTP en la URL correspondiente. Con esto queremos ilustrar que una API REST expone sus recursos de un modo organizado, por lo que los usuarios que interactúan con ella pueden esperar que se comporte de una determinada manera tan solo por la sintaxis.

Relaciones entre recursos

Sería muy sencillo ahora ampliar la API creando un nuevo tipo de recurso, por ejemplo, informes (reports). Imaginemos ahora que en la lógica interna de nuestra aplicación, cada usuario puede tener varios informes asociados (o ninguno). Lo que en bases de datos se conoce como una relación 1 a N entre entidades. Pues bien, si quisiéramos exponer esto como un recurso REST, podríamos considerar:

- http://edixconsulting.com/api/v1/users/paco/reports - Todos los informes pertenecientes al usuario 'paco'.
- http://edixconsulting.com/api/v1/reports - Todos los informes disponibles.
- http://edixconsulting.com/api/v1/reports?before_date=2021-11-10 - Una consulta a través de REST.

Autenticación

Hasta ahora hemos asumido que todos estos endpoints pueden ser accedidos por cualquiera, lo cual no es lo normal: querremos tener cierto control de quién accede a nuestra aplicación a través de HTTP y eso dependerá del control de acceso que hayamos implementado a nivel de servicio web.

HTTP nos proporciona varios mecanismos para enviar credenciales al servicio web:

Usuario y contraseña

Como ya hemos visto, podemos utilizar la URL para enviar, en claro, usuario y contraseña del servicio al que estamos accediendo:

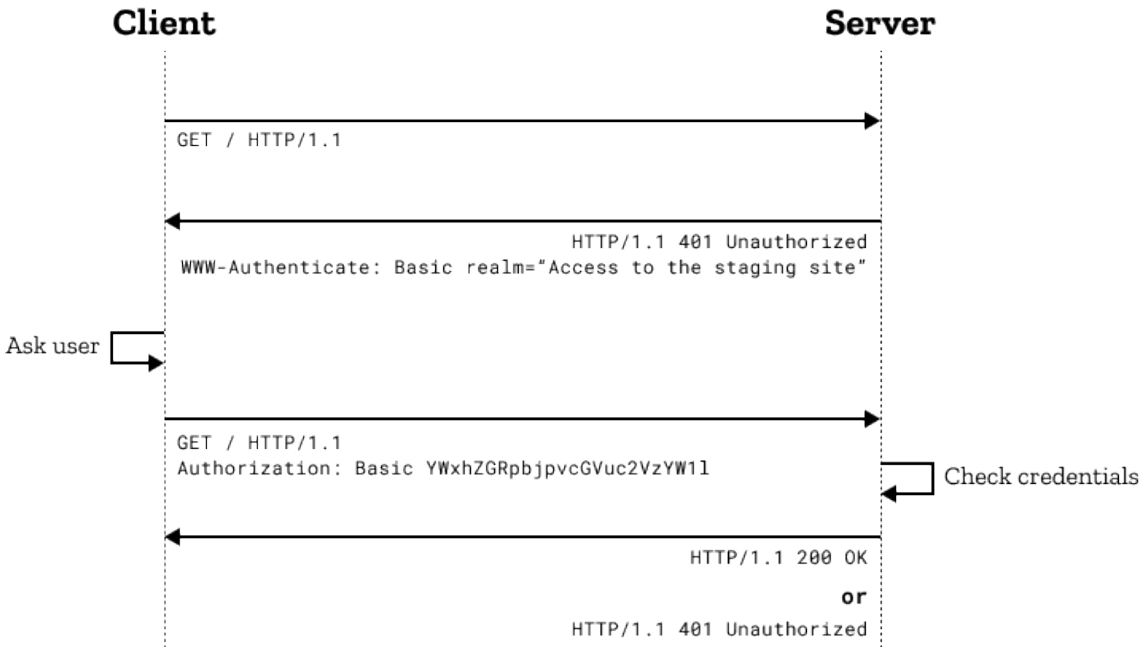
```
$ curl https://admin:1234@88.45.12.61:3030/api/v1/users
```

Como ya hemos visto, esta alternativa tiene el problema de la seguridad, ya que la URL solicitada quedará en los logs de acceso de los servidores web, del navegador, etc.

Sin embargo, no debemos olvidarnos de su existencia, ya que resulta muy útil en scripts o en Dockerfiles, donde podemos inyectar las credenciales como variables de entorno y conseguir limitar de ese modo la exposición de las mismas.

Autenticación básica

Podemos usar las cabeceras de autenticación del protocolo HTTP para enviar esas mismas credenciales. Es una alternativa bastante común porque, como veremos, existen diferentes maneras de enviarlas que lo hacen más seguro que en la URL.



Esquema básico de autenticación HTTP (Fuente: Mozilla)

Cuando un servidor requiere autenticación para un recurso, nos lo hará saber usando el código de respuesta 401 Unauthorized. Los navegadores web están preparados para preguntar al usuario por las credenciales, pero en una negociación automática entre máquinas será el cliente el que tenga que preparar la cadena de autenticación.

Dicha cadena será, para el caso más sencillo, la codificación en base64 de usuario y contraseña como parte valor de la cabecera HTTP 'Authorization'. Ojo, pues un valor **codificado** no significa **cifrado**: base64 no proporciona ninguna seguridad más allá de la ofuscación y, por tanto, no se debe confiar en este mecanismo para garantizar seguridad salvo que se envíe a través de un canal seguro.

Veamos un ejemplo usando httpbin.org, una conocida web para probar APIs REST:

```
# petición a un endpoint autenticado, respuesta 401
$ curl  -v  http://httpbin.org/basic-auth/admin/admin

* Trying 34.192.79.103...
* TCP_NODELAY set
* Connected to httpbin.org (34.192.79.103) port 80 (#0)
> GET /basic-auth/admin/admin HTTP/1.1
> Host: httpbin.org
> User-Agent: curl/7.64.1
> accept: application/json
>
< HTTP/1.1 401 UNAUTHORIZED
< Date: Sat, 11 Dec 2021 22:14:16 GMT
< Content-Length: 0
< Connection: keep-alive
< Server: gunicorn/19.9.0
< WWW-Authenticate: Basic realm="Fake Realm"
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Credentials: true
<
* Connection #0 to host httpbin.org left intact
* Closing connection 0

# envío de autorización vía cabecera HTTP + bash inline
$  curl http://httpbin.org/basic-auth/admin/admin" \
     -H "Authorization: Basic $(printf admin:admin | base64)"
{
  "authenticated": true,
  "user": "admin"
}
```

En este ejemplo hemos visto el caso más básico de autenticación, apropiadamente denominado *Basic*. Existen más tipos de autenticación en el protocolo HTTP. Puedes consultarlos aquí.

Referencias bibliográficas

[1] Telnet Protocol Specification, RFC 854. https://tools.ietf.org/html/rfc854.

[2] Tanenbaum, Andrew S. and Wetherall, David J. Computer Networks. Fifth Edition, Pearson New International ed. Boston [etc]: Pearson, 2014.

[3] Domain Names - Implementation and Specification, RFC 1035.

https://tools.ietf.org/html/rfc1035.

[4] DNS Zone Transfer Protocol (AXFR), RFC 5936.

https://tools.ietf.org/html/rfc5936#section-2.

[5] Simple Mail Transfer Protocol, RFC 2821. https://tools.ietf.org/html/rfc2821.

[6] Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, RFC 7231.

https://tools.ietf.org/html/rfc7231.

[7] HTTP response status codes.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status.

[8] A. Archip, C. Amarandei, P. Herghelegiu, C. Mironeanu and E. şerban. RESTful Web Services – A Question of Standards. 2018 22nd International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, 2018, pp. 677-682.

RESTful Web Services – A Question of Standards | IEEE Conference Publication

[9] Network Time Protocol (NTP), RFC 958. https://tools.ietf.org/html/rfc958.

[10] Hirsch, Frederick J. Introducing SSL and certificates using SSLeay. World Wide Web Journal 2.3 (1997): 141-173.

[11] SSL/TLS Strong Encryption: An Introduction.

http://httpd.apache.org/docs/2.2/ssl/ssl_intro.html

LA UNIVERSIDAD EN INTERNET | FORMACIÓN PROFESIONAL

PROEDUCA