Tutorial de TCP/IP

Javier Smaldone (http://blog.smaldone.com.ar)
22 de noviembre de 2006

Introducción

En este tutorial realizaremos la "*disección*" de una comunicación **TCP/IP** muy simple, con el fin de analizar qué ocurre a cada nivel.

A través de este sencillo experimento, podremos recorrer los conceptos fundamentales de las redes **TCP/IP**, para reafirmar la idea de que este protocolo es muy simple. El objetivo es lograr, sin demasiados conocimientos previos, una comprensión profunda de sus mecanismos.

La versión original (online) de este artículo, además de los comentarios de los lectores, puede encontrarse en: http://blog.smaldone.com.ar/2006/11/21/tutorial-sobre-tcpip/.

Licencia

Requisitos previos

No se requieren conocimientos de programación, aunque sí una base de conocimientos informáticos en general. Para continuar experimentando más allá de los expuesto aquí, se recomienda la utilización del programa **Wireshark** (antes conocido como **Ethereal**), disponible para **GNU/Linux**, **Microsoft Windows**, **Mac OS/X** y **Solaris** .

Quizás el requisito más importante sean las ganas de aprender, investigar, jugar y divertirse con redes **TCP/IP**.

Referencias

Cada vez que se introduzca un término relevante, el mismo contendrá un enlace a una página con mayor información, por lo general de la **Wikipedia** y, de ser posible, en español (aunque se recomienda ampliamente visitar su equivalente en inglés).

Un libro muy recomendable y claro (en inglés) es "*TCP/IP Illustrated Volume 1*" de W. Richard Stevens. Finalmente, la fuente de consulta definitiva, para conocer tanto los aspectos técnicos como la evolución historica de cada uno de los protocolos y normas, son los **Request for Comments** (**RFC**), algunos de los cuales han sido traducidos al español.

Nuestro experimento

Estableceremos una conexión **TCP/IP** entre un cliente y un servidor que intercambiarán información. Para ello, utilizaremos el servidor desarrollado en el tutorial sobre programación en redes (no es necesario que lo lea completamente, si no le interesa la programación, pero sería conveniente que revise los conceptos introductorios y el protocolo definido).

Ejecutamos entonces el servidor en el *host* **100.0.0.1**, utilizando el puerto **2222** (cualquiera de las versiones desarrolladas en el tutorial anterior sirve, ya que el protocolo es idéntico) y usamos el comando **telnet** para actuar como cliente desde el *host* **200.0.0.1** (el texto en *cursiva* es introducido por nosotros y el texto en **negrita** es la respuesta del servidor):

```
javier@200.0.0.1:~$ telnet 100.0.0.1 2222
Trying 100.0.0.1...
Connected to 100.0.0.1.
Escape character is '^]'.
Bienvenido.
salir
Adios.
Connection closed by foreign host.
javier@200.0.0.1:~$
```

Esto es todo. Ahora procederemos a analizar cómo se ha llevado a cabo esta comunicación a distintos niveles de abstracción.

Nivel de aplicación

A "*nivel de aplicación*" (lo que "*ven*" los programas), la comunicación se ha desarrollado de la siguiente manera:

1. El cliente se conecta al servidor.
2. El servidor envía el mensaje "**Bienvenido.**".
3. El cliente envía el comando "**salir**".
4. El servidor responde con el mensaje "**Adios.**".
5. El servidor cierra la conexión.
6. El cliente cierra la conexión.

Esto es prácticamente lo mismo que podemos observar de la salida del comando **telnet** utilizado, lo cual no es casual; intencionalmente a este nivel se ocultan todos los detalles de implementación, que aparecerán cuando analicemos los niveles inferiores.

Nivel de transporte

El protocolo utilizado a "*nivel de transporte*" es **TCP**. Este protocolo es el encargado de establecer la conexión y dividir la información en *paquetes*, garantizando que los mismos son entregados correctamente (sin pérdidas y en el orden apropiado).

Cabe resaltar aquí que el otro protocolo de transporte de **TCP/IP**, **UDP** no garantiza ni el arribo de todos los paquetes enviados, ni el orden en que estos llegan a destino. Por esto es mucho más simple, no incluyendo algunas de las características de **TCP** como números de secuencia y asentimientos.

A continuación analizaremos algunos aspectos del protocolo **TCP**.

Puertos y direcciones

El protocolo **TCP** se basa en **direcciones IP** para identificar los equipos (*hosts*) desde donde provienen y hacia donde se envían los paquetes.

Los **puertos** (*ports*) son valores numéricos (entre 0 y 65535) que se utilizan para identificar a los procesos que se están comunicando. En cada extremo, cada proceso interviniente en la comunicación utiliza un puerto único para enviar y recibir datos.

En conjunción, dos pares de puertos y direcciones IP identifican univocamente a dos procesos en una red **TCP/IP**.

Números de secuencia

TCP garantiza que la información es recibida en orden. Para ello, cada paquete enviado tiene un *número de secuencia*. Cada uno de los dos procesos involucrados mantiene su propia secuencia, que se inicia con un valor aleatorio y luego va incrementándose según la cantidad de bytes enviados.

Por ejemplo, si un paquete tiene número de secuencia x y contiene k bytes de datos, el número de secuencia del siguiente paquete emitido será $x + k$. (*Sí, el número de secuencia va contando la cantidad de bytes enviados por cada host*.)

Paquetes y acuses de recibo

TCP también asegura que toda la información emitida es recibida. Para ello, por cada paquete emitido, debe recibirse un asentimiento (en inglés "*acknowledgement*", abreviado **ACK**). Si pasado determinado tiempo no se recibe el **ACK** correspondiente, la información será retransmitida.

El **ACK** hace referencia al número de secuencia (que ha su vez involucra la cantidad de bytes enviados). Por ejemplo, para comunicar que se ha recibido correctamente el paquete cuyo número de secuencia es x, que contiene k bytes, se enviará un **ACK** con el valor $x + k$ (que coincide con el próximo número de secuencia a utilizar por parte del emisor del paquete en cuestión). Si el número de secuencia inicial es x, un valor de **ACK** t significa que el receptor ha recibido correctamente los primeros t-x bytes (en este sentido, el **ACK** es acumulativo).

El **ACK** no es un paquete especial, sino un campo dentro de un paquete **TCP** normal. Por esto, puede ocurrir que se envíe un paquete a solo efecto de asentir una determinada cantidad de bytes, o como parte de un paquete de otro tipo (por ejemplo, aprovechando el envío de nuevos datos, para comunicar la recepción de datos anteriores). De hecho, aunque ya se haya enviado un paquete exclusivamente de **ACK** con un valor *t*, si luego se envía un paquete de datos, puede repetirse en él el **ACK** con el mismo valor *t*, sin que esto confunda al emisor de los datos que se están asintiendo. (Por simplicidad, en nuestro ejemplo hemos eliminado esta información redundante).

Otros campos de un paquete TCP

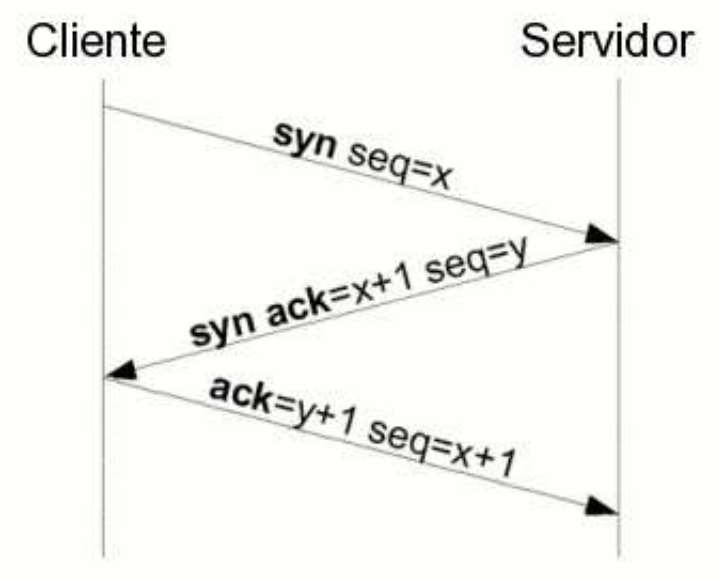
El protocolo **TCP** incorpora mecanismos tales como control de integridad de los datos (*checksum*), prevención de congestiones, entre otros, que no serán mencionados aquí por la simplicidad de este tutorial.

Inicio y fin de la conexión

Para dar comienzo a la conexión, el cliente envía un paquete **SYN** al puerto e **IP** en donde "*escucha*" el servidor, con un número de secuencia inicial aleatorio. Este último, responde con otro paquete **SYN**, con un número de secuencia inicial aleatorio y un **ACK** con el número de secuencia del paquete **SYN** recibido, más uno. El cliente envía un paquete con el **ACK** del **SYN** recibido, y una vez hecho esto la conexión se encuentra establecida y puede darse comienzo a la transmisión de datos (iniciada por cualquiera de las partes, según el protocolo de aplicación que utilicen).

La razón por la cual se intercambian números de secuencia aleatorios es para evitar que se confunda el inicio de dos conexiones diferentes y algunos ataques que se basan en falsear el comienzo de una conexión (*spoofing*).

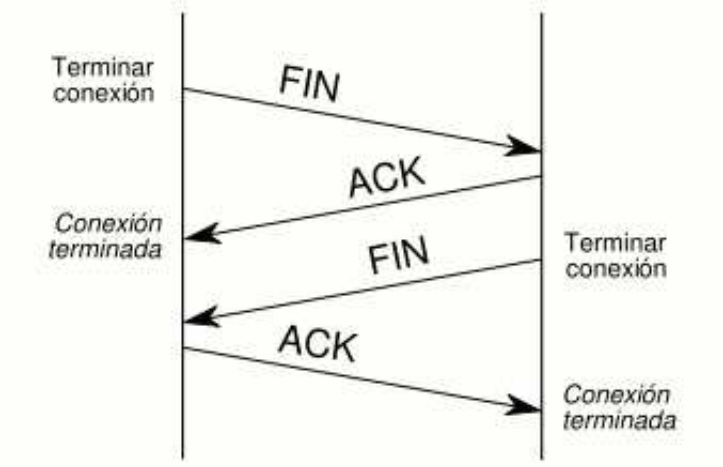
La siguiente figura ilustra el establecimiento de una conexión **TCP**, llamada "*negociación de tres pasos*" o "*3-way handshake*":



Para finalizar la conexión, uno de los dos procesos envía un paquete **FIN**, a lo que el otro responderá con un **ACK**. A su vez, el otro proceso puede enviar un paquete **FIN** (recibiendo también un **ACK**) y la conexión quedará cerrada definitivamente.

Nótese que el segundo proceso puede no enviar el paquete **FIN**. Esto significa que ese extremo de la conexión no se cerrará, pudiendo aún enviar datos a través de la misma. De lo contrario, en caso de desear terminar la conexión, puede combinar el **ACK** y el **FIN** en un solo paquete (esta es la situación más común).

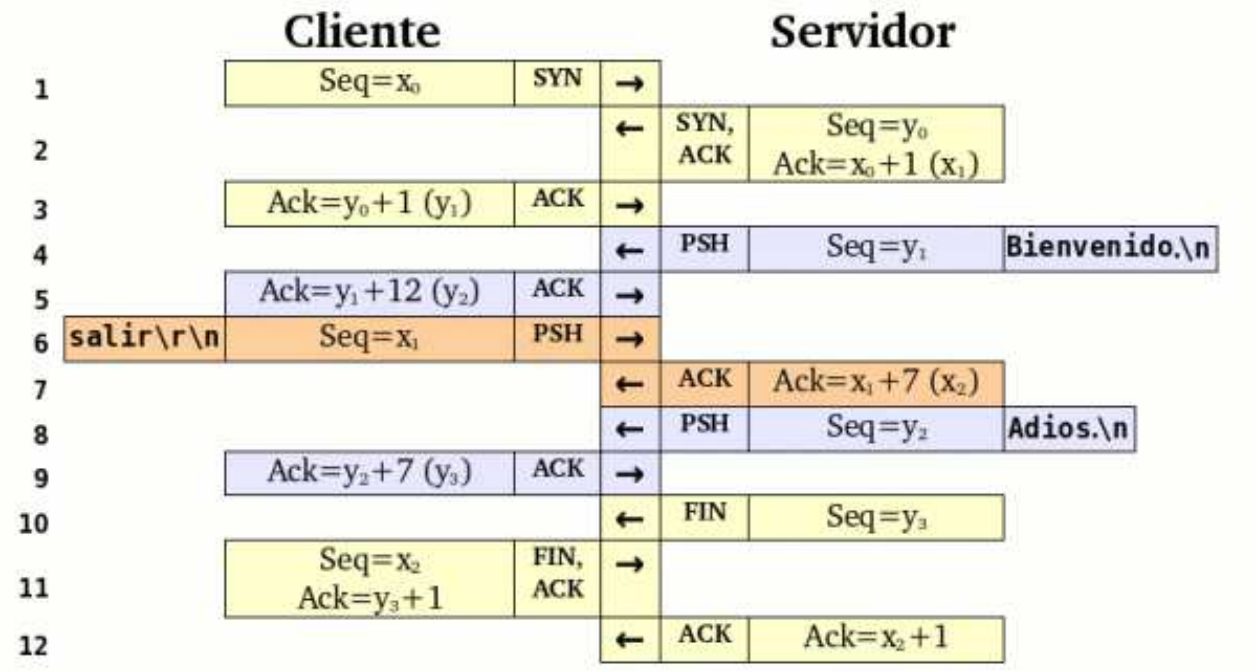
La siguiente figura ilustra la forma general de terminación de una conexión **TCP**:



Análisis a nivel de transporte

Habiendo revisado los conceptos más relevantes, analizaremos ahora la conexión realizada desde el punto de vista del protocolo **TCP**. Supondremos que el puerto utilizado por el cliente es **4683** (el del servidor, recordemos, es **2222**).

La siguiente figura muestra una visión simplificada de esta conexión:



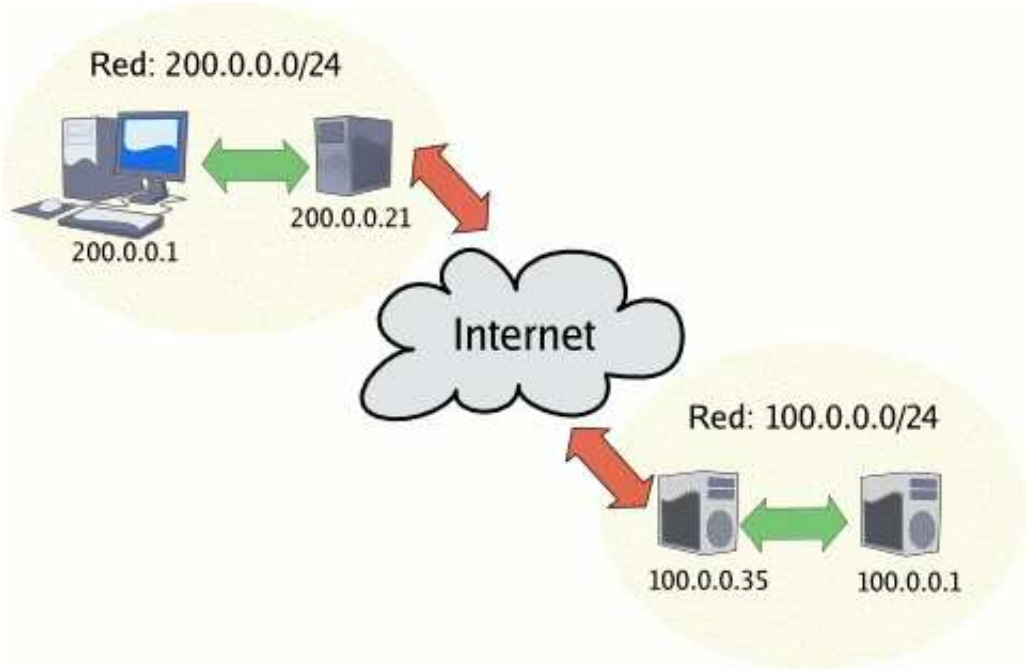
Veamos qué función cumple cada paquete:

1. El cliente envía un **SYN** al servidor, con número de secuencia x_0.
2. El servidor responde con un paquete **SYN**, con número de secuencia y_0 y un **ACK** con x_0+1 (en adelante, x_1).
3. El cliente envía un paquete **ACK** del **SYN** que acaba de recibir, con valor y_0+1 (en adelante, y_1). (A partir de este momento la conexión se encuentra establecida y puede comenzar el intercambio de datos entre las aplicaciones.)
4. El servidor envía un paquete **PSH** ("*push*") conteniendo la cadena "**Bienvenido.\n**" (12 bytes), con número se secuencia y_1. (El caracter "**\n**", *newline*, representa el fin de línea.)
5. El cliente envía un **ACK** por la correcta recepción del paquete anterior. El número de **ACK** es y_1+12 (que llamaremos y_2 y será el próximo número de secuencia utilizado por el servidor).
6. El cliente envía la cadena "**salir\r\n**", con número de secuencia x_1. (Los caracteres "**\r\n**" representan el fin de línea. Ver nota al final de la sección.)
7. El servidor envía el **ACK** correspondiente, con el valor x_1 más la longitud de "**salir\r\n**" (*7*), que llamaremos x_3.
8. El servidor envía la cadena "**Adios.\n**" (7 bytes), con número de secuencia y_2.
9. El cliente envía el **ACK** con el valor y_2+7 (en adelante, y_3).
10. El servidor cierra su lado de la conexión enviando un paquete **FIN**, con secuencia y_3.
11. El cliente, que también cierra su conexión, envía un paquete **FIN** con secuencia x_3, con el **ACK** del paquete anterior (y_3+1).
12. El servidor envía el **ACK** del anterior paquete **FIN** (con valor x_3+1), con lo cual la conexión finaliza.

Nota: *En este ejemplo podemos ver un error en el diseño del protocolo de aplicación, ya que el servidor representa los fines de línea con el caracter "**\n**" ("newline", código ASCII 10), en tanto que el cliente está usando los caracteres "**\r\n**" ("newline" y "carriage return", códigos ASCII 10 y 13, respectivamente). Esta última es la forma de representación más utilizada en aplicaciones **TCP/IP**.*

Nivel de red

Antes de realizar el análisis a "*nivel de red*" (protocolo **IP**) vamos a suponer que tenemos la siguiente topología:



El cliente se ejecuta en el *host* cuya dirección IP es **200.0.0.1**. El mismo está conectado a la red local **200.0.0.0/24** y su *gateway* ("*router*", "*enrutador*" o "*puerta de enlace*") es el *host* **200.0.0.21**.

La dirección de red local está compuesta por la *dirección de red* propiamente dicha, **200.0.0.0**, y la *máscara de red*, **24** (que también puede ser representada como **255.255.255.0**). Esto significa que los primeros **24** bits de cualquier dirección serán interpretados como identificador de la red, en tanto que los últimos **8** identificarán a cada *host* (recordemos que, aunque la notación usual es escribir las direcciones IP como cuatro números decimales separados por puntos, en realidad se componen de 32 bits).

Por ejemplo, en el contexto de esta red, la dirección **200.0.0.45** será considerada una dirección local (por coincidir los primeros **24** bits con los de la dirección de red). Esto significa que cualquier paquete destinado a dicha dirección IP, será entregado "*localmente*" (o sea, directamente a través del *protocolo de enlace*, como veremos más adelante).

En el caso de una dirección de destino "no-local", los paquetes serán entregados al *gateway* **200.0.0.21** (usando el *protocolo de enlace*), quien será luego el encargado de entregar el paquete al *host* de destino (si este perteneciera a alguna red local a la que dicho *gateway* esté conectado), o reenviarlo a través de otro

gateway (en caso contrario).

De la misma manera el servidor, cuya **dirección IP** es **100.0.0.1**, pertenece a la red **100.0.0.0/24**, cuyo *gateway* es **100.0.0.35**.

Análisis a nivel de red

A nivel IP no hay demasiado que agregar. Aquí se realiza el "*ruteo*" (o "*encaminamiento*") de los paquetes provenientes de la capa de transporte (que en este nivel se denominan "*datagramas*") desde la dirección IP de origen hasta la de destino (alternando estos roles **100.0.0.1** y **200.0.0.1** según sea el emisor el servidor o el cliente, respectivamente).

Un campo interesante que se añade es el **TTL** ("*tiempo de vida*" o "*time to live*"), que tiene un valor inicial en el *host* que produce el paquete (generalmente **64**) y luego es decrementado por cada *gateway* por el que pasa. Si un *gateway* recibe un paquete con el campo **TTL** en **cero**, el mismo es descartado y se genera un paquete del protocolo **ICMP** (usado para control y mensajes de error) dirigido a su emisor, indicando que dicho paquete ha excedido la cantidad máxima de saltos. Esta medida es implementada para evitar que, quizás por un error de ruteo, un paquete quede indefinidamente "dando vueltas" por la red.

En este ejemplo supondremos que el enrutamiento de paquetes es simple (estático), para facilitar las explicaciones. Existen casos complejos en donde se utiliza enrutamiento dinámico, en los cuales paquetes pertenecientes a la misma comunicación pueden enviarse por caminos distintos, alterando inclusive el orden en que llegan al destino (y hasta pudiendo producirse pérdidas).

Debemos tener en cuenta que estamos usando el llamado **IPv4** (IP versión 4), ya que también existe el **IPv6** (IP versión 6), cuya aplicación se está difundiendo rápidamente en Internet y que soluciona muchos inconvenientes que presenta el anterior.

Nivel de enlace

El "*nivel de enlace*" es el nivel más cercano al "*nivel físico*" y es totalmente independiente del protocolo **TCP/IP**. Existen gran variedad de tecnologías de este tipo; siendo las más comunes **Ethernet**, **WiFi**, **PPP**, **Frame Relay**, **ATM**, entre otras.

Supondremos el caso más común: una red Ethernet. Este protocolo se basa en el uso de direcciones de 6 bytes (48 bits), que no son "*ruteables*" (aquí no existen *gateways*), por lo cual se utiliza en redes de área local (**LANs**). Cada dispositivo **Ethernet** tiene asociada una dirección única (asignada por el fabricante del mismo), la que usualmente se denomina **MAC Address**.

Resolución de direcciones

Supongamos que el *host* **200.0.0.1** quiere enviar un paquete IP al *host* **200.0.0.33**. Como ambos están en la misma red local (los primeros 24 bits de sus direcciones coinciden), lo único que debe hacer es averiguar cuál es la dirección **Ethernet** de este último. Para ello, se utiliza el protocolo **ARP** ("*Address Resolution Protocol*").

El funcionamiento es muy simple. El host **200.0.0.1** envía un paquete (en terminología de **Ethernet** se denomina "*frame*") a la dirección **ff:ff:ff:ff:ff:ff** (las direcciones **Ethernet** se denotan con seis bytes en hexadecimal separados por dos puntos), que es la dirección de *broadcast* (que llega a todos los *hosts* de la red) preguntando quién tiene la dirección IP **200.0.0.33**. Dicha solicitud **ARP** tiene como origen la dirección **Ethernet** del emisor (supongamos, **00:30:b8:80:dd:11**).

El poseedor de esa dirección IP le responderá con otro *frame* con su dirección **Ethernet** como origen (supongamos, **01:4e:bb:a1:01:8b**). De ahora en más, cada vez que el *host* **200.0.0.1** quiera enviar un paquete IP al *host* **200.0.0.33**, enviará un *frame* **Ethernet** proveniente de la dirección **00:30:b8:80:dd:11** a la dirección **01:4e:bb:a1:01:8b**, conteniendo el paquete original. (La información sobre las direcciones **ARP** se almacenan en cada *host* en una tabla que tiene una duración de algunos minutos.)

Fragmentación

Puede ocurrir que el tamaño de los paquetes producidos por la capa de red (**IP**, en nuestro caso) sean de un tamaño mayor al máximo que puede transmitir el medio físico utilizado (**MTU** o "*unidad máxima de transferencia*". Por esto, puede ocurrir que los paquetes se *fragmenten*, para acomodarse a esta limitación.

Esta situación puede volver a presentarse a lo largo del camino "físico" que recorra la información, siendo responsabilidad de cada *gateway* el fragmentar y reensamblar los paquetes para preservar los datos.

Análisis a nivel de enlace

Volviendo ahora a nuestro experimento, el *host* donde se ejecuta la aplicación cliente (**200.0.0.1**) debe enviar paquetes IP al *host* en donde se ejecuta el servidor (**100.0.0.1**). Claramente, éste último no pertenece a su red local, por lo cual deberá enviarlo a través del *gateway*.

Para ello, usando el protocolo **ARP** averigua la dirección **Ethernet** del *gateway* (cuya dirección IP, recordemos, es **200.0.0.21**), que supondremos es **00:01:02:ed:41:61**. Una vez hecho esto, envía un frame **Ethernet** (con origen **00:30:b8:80:dd:11** y destino **00:01:02:ed:41:61**), conteniendo el paquete IP cuyo origen es **200.0.0.1** y con destino a **100.0.0.1**.

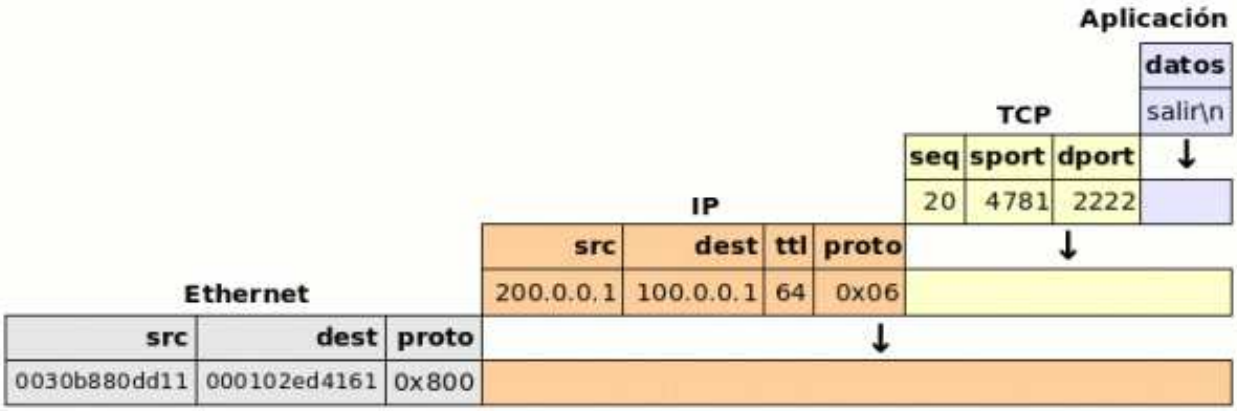
El *gateway* recibirá el *frame* (puesto que la dirección **Ethernet** de destino es la suya) y dentro de él encontrará un paquete IP dirigido a **100.0.0.1**. Decrementará el campo **TTL** y, en base a las reglas definidas en su "*tabla de enrutamiento*" (o "*tabla de ruteo*"), lo reenviará hacia el *gateway* correspondiente (usando el protocolo asociado al medio físico mediante el cual esté conectado con éste).

Una situación similar se presenta considerando los paquetes emitidos por el *host* **100.0.0.1** hacia **200.0.0.1**.

De esta manera, el mismo paquete IP va atravesando distintos *gateways* a través de distintos medios físicos (que involucran diferentes protocolos de enlace), hasta llegar al *host* de destino. Por ejemplo, el paquete original puede llegar al primer *gateway* a través de un *frame* **Ethernet**, ser enviado por este al *gateway* del proveedor de Internet a través de un paquete **PPP**, atravesar una red **ATM** en Internet, luego llegar por un enlace **Frame Relay** al *gateway* de la red de destino, y ser entregado en otro *frame* **Ethernet** al *host* de destino.

El camino de la información a través de las distintos niveles

La siguiente figura ilustra un ejemplo del camino que sigue la información desde la aplicación que la produce, a través de los distintos niveles o capas de cada protocolo.



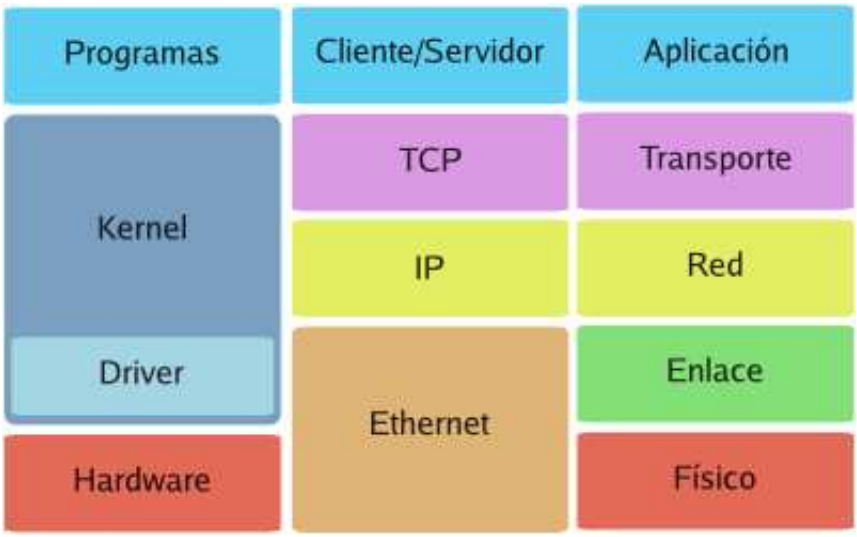
Nota: *Este ejemplo es una simplificación de la realidad. Se han omitido muchos otros datos que forman parte de cada protocolo (controles de error, delimitadores, indicadores de longitud, etc.), que no son relevantes en el contexto de este tutorial.*

1. **Nivel de aplicación**: La aplicación (en este caso, el programa cliente), escribe la cadena "**salir\n**".
2. **Nivel de transporte**: En la capa **TCP** forma un paquete agregando el número de secuencia (**20**), puerto de origen (**4781**) y puerto de destino (**2222**).
3. **Nivel de red**: En la capa **IP** se forma un paquete (*datagrama*) añadiendo la dirección IP origen (**200.0.0.1**), destino (**100.0.0.1**), el **TTL** (**64**) y un valor que identifica el protocolo del paquete encapsulado (**0x06**, valor hexadecimal que representa al protocolo **TCP**).
4. **Nivel de enlace**: En la capa **Ethernet** se forma un nuevo paquete (*frame*) agregando las direcciones **Ethernet** de origen (**00:30:b8:80:dd:11**) y destino (**00:01:02:ed:41:61**, la dirección del gateway, cuya IP es **200.0.0.21**). Se añade además el identificador del tipo de protocolo del paquete contenido (el valor **0x800** corresponde al protocolo **IP**).
5. **Nivel físico**: El *frame* formado es enviado a través del medio físico que vincula los *hosts* de la red local (típicamente, cable de par trenzado).

Como puede apreciarse, tanto el protocolo **IP** como **Ethernet** "*encapsulan*" a otros protocolos. Esto permite realizar distintas combinaciones, creando "*túneles*" (como en el caso del ampliamente difundido y utilizado **PPPoE**).

Software, modelo conceptual y hardware

El siguiente diagrama muestra la relación entre cada uno de los componentes lógicos analizados, su implementación y la división entre *hardware* y *software*.



Para finalizar

En este tutorial hemos recorrido cada uno de los principales componentes del protocolo **TCP/IP** y analizado su función a través de un ejemplo concreto (aunque simple).

Deliberadamente, hemos omitido algunos puntos importantes, como el sistema DNS (que posibilita la utilización de nombres en vez de direcciones IP), la asignación automática de direcciones IP (a través del protocolo DHCP), y algunos detalles sobre direccionamiento y enrutamiento IP. (Quizás algunos de ellos sean motivo de próximos tutoriales.)

Es el deseo del autor que a través de la lectura del presente tutorial se haya podido lograr un panorama general acerca del funcionamiento de las redes **TCP/IP**, posibilitando al lector su avance hacia temas (y, por qué no, experimentos) más complejos e interesantes.

Las críticas, sugerencias y comentarios (a través de la entrada en el blog del autor) son bienvenidos y agradecidos.