CURSO DE TCP/IP
INTRODUCCION

Si alguna vez has intentado comprender Internet, seguro que has acabado frente a un libro de TCP-IP. Y seguro que a la sexta página lo has dado por IMPOSIBLE!!! TCP-IP es el alma de la red, nosotros te ofrecemos un curso MUY ESPECIAL :)

1.Introducción a la introducción.

Probablemente muchos de vosotros esperabais con impaciencia un curso como este, así que espero que vuestra alegría, al ver que al fin nos metemos de lleno con el tema en esta revista, no se vea frustrada al descubrir que vuestro "profe" en este curso va a ser el mismo que mes tras mes os ha ido aburriendo soberanamente con la serie RAW. ;-)

Si es cierto que os parezco aburrido, en mi defensa he de alegar que la descripción detallada de un protocolo es algo aburrido de por sí y, aunque he hecho lo posible por hacer artículos amenos, cuando hay que ponerse serio, hay que ponerse serio.

Aprovecho esta ocasión para agradecer a Adhara (conocida como MariAn antes de digievolucionar) su aportación en este sentido, con las caricaturas e ilustraciones que ha aportado para amenizar en la medida de lo posible la serie RAW y que, por supuesto, también seguirá aportando en este curso que comienza.

Os plantearé el terrible dilema que he sufrido para poder comenzarlo. Para ayudarme a la hora de estructurar un poco las ideas he ojeado multitud de libros y cursos de TCP/IP para ver cómo abordaban el problema y poder explicarlo desde cero, que era mi gran reto.

Lamentablemente, en ninguno he encontrado lo que yo buscaba, una forma de introducir los conceptos de forma que alguien sin ningún conocimiento pueda no sólo aprenderse de memoria un montón de formatos de cabeceras, comandos de protocolos, y un glosario de términos; si no realmente llegar a hacer suyos los conceptos y comprenderlos de una forma totalmente natural.

Por supuesto, ya se que la mayoría de los lectores de esta revista tienen ya un nivel aceptable de conocimientos, pero he de enfocarlo no sólo para los lectores "veteranos", si no también para aquellos totalmente novatos que, sin duda, serán los que más se beneficien de este curso y que les abrirá las puertas para llegar a comprender en profundidad todo lo demás que se enseñe en la revista a partir de ahora.

Sin duda alguna, el TCP/IP es uno de los pilares de todo este jaleo en el que estamos metidos. ;-) Así que decidí coger al toro por los cuernos, y enfocar la cuestión desde un punto de vista diferente.

TCP/IP

TCP / IP: Transmission Control Protocol / Internet Protocol = Protocolo de Control de Transmisión / Protocolo de Internet

¿Cuántos cursos de TCP/IP empiezan contando el modelo **OSI (Open Systems Interconnection = Interconexión de Sistemas Abiertos)**? A los que hayáis seguido alguno de esos cursos publicados en Internet o en otras revistas, ¿os quedó claro desde el principio, por ejemplo, para qué servia la capa de sesión del modelo OSI? ¿No pensáis que quizá empezar abordando el problema planteando un modelo teórico tan complejo puede ser contraproducente? ¿No os quedaron

más dudas después de terminar el tema de introducción que antes de empezarlo? Seguro que la mayoría dejasteis el curso a la mitad (y otros ni siquiera pasasteis de las primeras páginas).

El empezar cualquier curso de TCP/IP hablando del modelo OSI parece que ha sido la solución estándar para solucionar el reto de mostrar el concepto de los protocolos por capas a gente totalmente nueva en el tema. Pero a mí personalmente nunca me ha parecido una buena idea, así que he tomado una medida muy arriesgada, y es intentar dar un nuevo enfoque a este reto.

No sé qué resultados tendrá mi enfoque, pero espero que al menos sea una alternativa para que aquellos que no terminan de captar los conceptos por los medios "clásicos" tengan aquí una segunda oportunidad.

¿Qué esperabais? ¿Que mi curso de TCP/IP fuese como todos los demás? ¡Para eso tenéis millones de libros sobre el tema! Lo que pretendemos dar en esta revista son nuevas visiones que no se pueden encontrar en las fuentes "convencionales".

El juzgar si nuestro enfoque es mejor que el convencional, ya depende de cada lector, y confío en que todos vosotros tendréis buen juicio para escoger la opción que para cada uno de vosotros resulte más adecuada.

Como veréis, mi enfoque intenta mostrar los conceptos puros, al margen de cualquier detalle técnico, mediante varios símiles con otros conceptos, bien conocidos por todos, de nuestra vida cotidiana.

Por supuesto, después de este primer artículo vendrán otros, y ahí si que veremos ya en profundidad los detalles técnicos, los cuales nos entrarán con muchísima más facilidad si previamente hemos dedicado un tiempo imprescindible a llegar al fondo de los conceptos.

2. El concepto fundamental de protocolo por capas

Empezaremos nuestro viaje metafórico por el mundo de los protocolos situándonos en un hospital, donde los doctores PyC y Scherzo hablan acerca de la próxima operación a corazón abierto que tendrán que llevar a cabo.

El doctor PyC tiene unas dudas acerca de la complicadísima operación, y acude al doctor Scherzo en busca de ayuda.

Dios mío, como haya entre los lectores algún médico o estudiante de medicina... ¡que me perdone por la increíble cantidad de estupideces que acabo de soltar! XD

Al margen de si tiene sentido o no la parrafada, supondremos que se trataba de una conversación perfectamente normal entre cirujanos.

Ahora vamos a plantearnos una serie de cuestiones sobre estas viñetas. En primer lugar, ¿qué han hecho básicamente PyC y Scherzo?

COMUNICARSE.

Ahora vamos a analizar un poco en qué ha consistido esa comunicación.

En primer lugar, lo que más nos llama la atención es el lenguaje técnico utilizado, que sólo es comprendido por los cirujanos.

Igual que los cirujanos tienen su propio lenguaje técnico, los informáticos también tienen el suyo, los arquitectos el suyo, y los abogados el suyo, todos ellos diferentes entre sí.

Pero, a pesar de que todos estos lenguajes técnicos sean diferentes, todos ellos se apoyan en una misma base, que es el idioma; en este caso, el castellano.

El lenguaje técnico de los cirujanos consiste únicamente en una serie de palabras y expresiones que permiten expresar los términos específicos que requieren los cirujanos para comunicarse en su trabajo. Por tanto, no es un lenguaje completo, ya que no posee una gramática propia que permita mantener una conversación sin apoyarse en un idioma básico, como puede ser el castellano.

Si, por ejemplo, eliminásemos de la parrafada del doctor PyC todo aquello que no formase parte exclusivamente del lenguaje técnico de los cirujanos, esta frase:

Se convertiría en ésta otra:

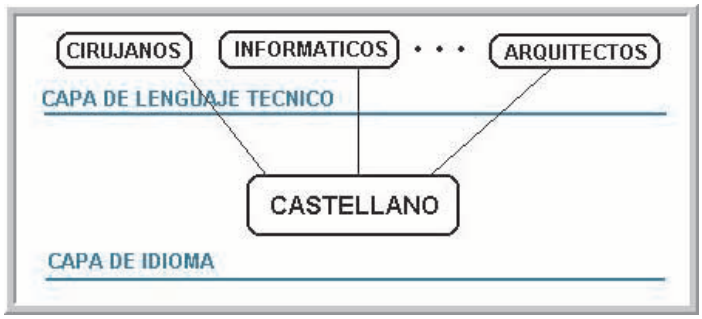
Por la cara que pone el doctor Scherzo podemos estar seguros de que utilizando tan sólo el lenguaje técnico, sin apoyarnos en la base que es el idioma castellano, es imposible que dos cirujanos se comuniquen.

Lo mismo que pasa con los cirujanos pasa con cualquier otro grupo de profesionales que utilicen su propio lenguaje técnico. Todos ellos apoyan toda su conversación en un idioma común, que puede ser el castellano, el inglés, o cualquier otro.

Por supuesto, para que dos profesionales se entiendan tienen que hablar no sólo el mismo lenguaje técnico, si no también el mismo idioma común.

Si el doctor PyC hablase Japonés, sin duda el doctor Scherzo habría puesto la misma cara de incomprensión.

Según lo que hemos visto hasta ahora, la comunicación entre los dos doctores funciona gracias a dos capas independientes: el idioma, y el lenguaje técnico.



¿Cuál es el motivo por el cual es esto así? Pues, si pensáis un poco, llegaréis vosotros mismos a la conclusión.

Imaginad que el lenguaje técnico de los cirujanos fuese un lenguaje completo, con sus fórmulas de saludos, despedidas, una gramática completa para construir las frases, palabras para expresar cualquier término común en cualquier comunicación (como los habituales: "me lo repita", "¡habla más despacio, que no me da tiempo a apuntarlo!", etc.), e incluso tuviese sus propios nombres, en lugar de los que tenemos en castellano (doctor Pyc, y doctor Scherzo). ¡Sería una completa locura!

Desde luego, no sería nada práctico que cualquier cirujano tuviese que aprender un idioma totalmente nuevo sólo para poder comunicarse con sus colegas.

Lo más práctico, y lo más lógico, es utilizar el recurso conocido por todos que es el idioma castellano, y simplemente ampliarlo con una serie de términos que permitan entrar en detalle en los conceptos manejados por los cirujanos.

Una vez comprendida la necesidad de comunicarse utilizando dos capas, vamos a entrar más en profundidad en la comunicación que vimos en las viñetas.

¿Qué otro medio común han utilizado el doctor Scherzo y el doctor Pyc para comunicarse? ¡El habla!
Si trasladásemos toda esa conversación a un papel, ¿no tendría el mismo sentido? ¿Y si la trasladásemos a una conversación telefónica? ¿O a una conversación por IRC (Internet Relay Chat)?

Los dos doctores se han apoyado en un medio físico común, que es la voz, pero perfectamente podrían haber mantenido la misma conversación a través de otro medio físico, como la escritura, o el teléfono.

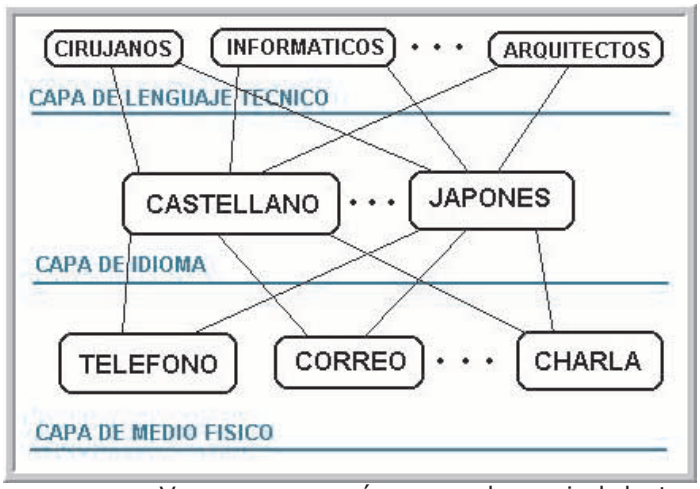
Tanto si esa conversación es hablada como si es escrita, seguiría utilizando tanto el lenguaje técnico de los cirujanos, como el idioma castellano. En nada cambiaría, salvo en el hecho de que el medio utilizado sería diferente.

Ahora bien, igual que un cirujano japonés no puede entenderse con un cirujano de Jaén, si el doctor PyC le hubiese soltado la parrafada al doctor Scherzo por correo, y éste le hubiese respondido a viva voz cuando recibiese la carta (es decir, que se lo habría contado a las paredes), tampoco habría sido posible una comunicación.

Ambos interlocutores tienen que compartir el mismo medio físico para comunicarse. Si un interlocutor está utilizando el teléfono, y el otro está respondiendo por escrito en un papel, jamás podrá haber una comunicación.

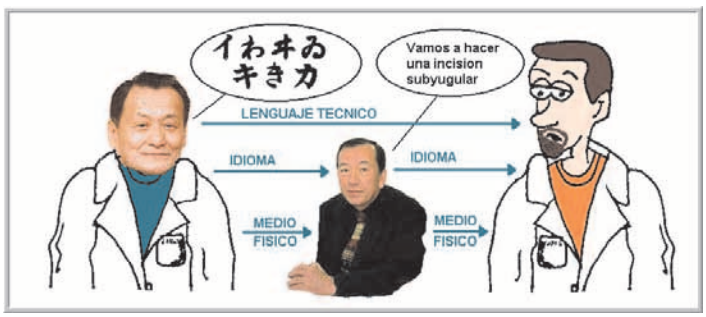
Por supuesto, tampoco sirve de nada que ambos hablen a viva voz, si cada uno está en un lugar diferente, donde no se puedan escuchar mutuamente.

Podemos considerar, por tanto, al medio físico como otra capa de la comunicación. En este caso, esta capa ya no existe por una conveniencia de hacer las cosas más fáciles, si no por una necesidad natural.



Vamos a ver qué ocurre ahora si el doctor Me-Iwa, de Japón, quiere hablar con el doctor PyC acerca de la operación de cardiopatía precarótida.

Por supuesto, no podrán comunicarse directamente, al hablar distintos idiomas, pero por suerte el doctor Me-Iwa tiene un intérprete que puede hablar tanto en castellano como en japonés, por lo que éste sería el escenario ahora:



Ahora meditemos un poco acerca de este nuevo personaje, el intérprete. Lo más probable es que este intérprete sea un estudiante de filología, o simplemente un estudiante de idiomas de academia pero, en cualquier caso, es poco probable que el intérprete sea un cirujano.

Pero, ¿es realmente necesario que el intérprete sepa algo de cirugía? El lenguaje técnico de los cirujanos al fin y al cabo no es más que una extensión del idioma, por lo que bastaría con que el intérprete simplemente conociese ambos idiomas para transmitir la conversación entre los dos interlocutores. Nos da exactamente igual que el intérprete no entienda ni papa de la conversación, ya que esa conversación no va dirigida a él, no es más que un mero intermediario. Él tiene que saber únicamente ambos idiomas: japonés y castellano.

Una vez que el intérprete ha traducido la frase "Vamos a hacer una incisión subyugular" ya será problema de los cirujanos entenderse entre ellos, ya que el intérprete no tiene ni idea de qué es una incisión subyugular, pero si que sabe traducir las palabras literalmente.

Por tanto, podríamos considerar al intérprete como un intermediario en la conversación que sólo llega a comprender hasta cierta capa de la comunicación pero que, aún así, es capaz de transmitir todo, confiando en que los interlocutores serán capaces de comprenderse una vez traducido el idioma.

Más adelante profundizaremos mucho más en la cuestión de los intermediarios en la comunicación, así que quedaos bien con esta idea. ;-)

3. Las capas de TCP/IP

¡Al fin vamos a ver la relación que tiene todo esto con el tema que nos interesa! Ya hemos comprendido la necesidad de estructurar la comunicación en diferentes capas, tanto por necesidad física, como por conveniencia para facilitar la comunicación (reducir su complejidad). Por supuesto, eso es algo que ocurre en todo tipo de comunicación y, por tanto, la comunicación entre máquinas no va a ser menos.

Por un momento, imaginemos que no hubiese capas en la comunicación por Internet.

Si yo tuviese que programar un cliente de correo electrónico (como por ejemplo el Outlook Express, tendría que idear desde cero toda una serie de funciones para interconectar al remitente y al destinatario, una serie de

funciones para asegurarme de que el mensaje llegue sin errores hasta el destino, una serie de funciones para identificar a ambas partes, etc., etc.

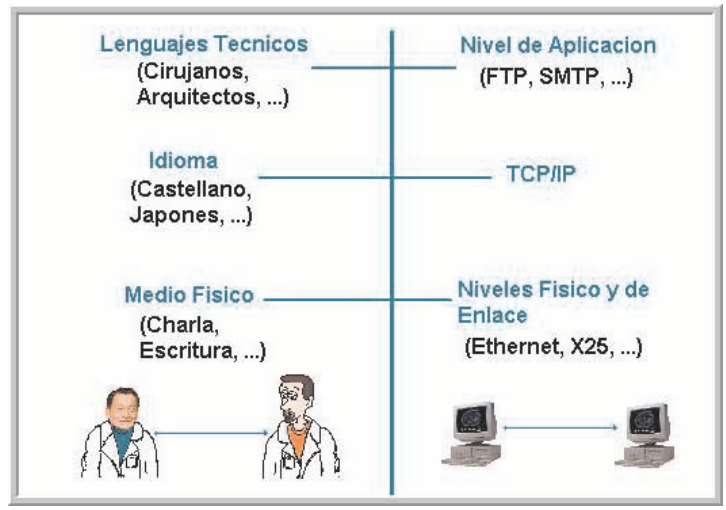
Si después me da por programar un cliente de FTP (si no sabes lo que es el FTP –File Transfer Protocol-, descárgate gratis el número 1 de esta revista desde www.hackxcrack.com), tendría que inventarme de nuevo y desde cero unas funciones para interconectar ambas partes (nuestro cliente con el servidor), unas funciones para asegurarme de que los ficheros y los comandos lleguen sin errores, una serie de funciones para identificar ambas partes, etc., etc.

El hacer las cosas de esta manera, no sólo sería una cantidad ingente de trabajo innecesario, si no que además dificultaría enormemente que los programas se entendiesen entre sí.

Si todas las aplicaciones que utilizan Internet tienen que realizar una serie de tareas comunes, como son la interconexión de las partes implicadas en la comunicación, la identificación de ambas partes, la corrección de errores, el ajuste de las velocidades de recepción y transmisión, etc., etc., ¿por qué no utilizar un lenguaje común para todas ellas?

Igual que todos los profesionales (cirujanos, informáticos, arquitectos...) utilizan el idioma castellano como base sobre la cual apoyan luego sus lenguajes técnicos propios, también las máquinas conectadas a Internet utilizan un mismo idioma común como base sobre la que luego apoyar cada lenguaje específico.

En este caso, el idioma común de las máquinas es el famoso TCP/IP, y los lenguajes técnicos que utiliza cada máquina apoyándose en TCP/IP son los que permiten las diferentes tareas, como transmitir ficheros (FTP), enviar correo (SMTP), mostrar páginas Web (HTTP), etc., etc.



Comparación entre las capas de la comunicación entre dos personas y la comunicación entre dos máquinas. Esta comparación no es muy precisa, así que la muestro sólo como una primera aproximación a la idea.

Los que hayáis seguido mi serie RAW desde el principio, comprenderéis ahora por qué una y otra vez repetía frases como: "protocolo que funciona sobre TCP/IP", o "protocolos por encima del nivel de TCP/IP", etc.

Por ejemplo, ¿tenéis a mano el número 14 de la revista, en el que hablaba sobre el protocolo **DNS**? Mirad el primer párrafo de ese artículo, y veréis que decía:
"Por primera vez en la ya veterana serie RAW, no se trata de un protocolo basado en TCP, ¡si no en el aún desconocido UDP!"

Esto sería lo mismo que decir: "este lenguaje que vamos a contar no se basa en el idioma castellano, si no en el idioma japonés".

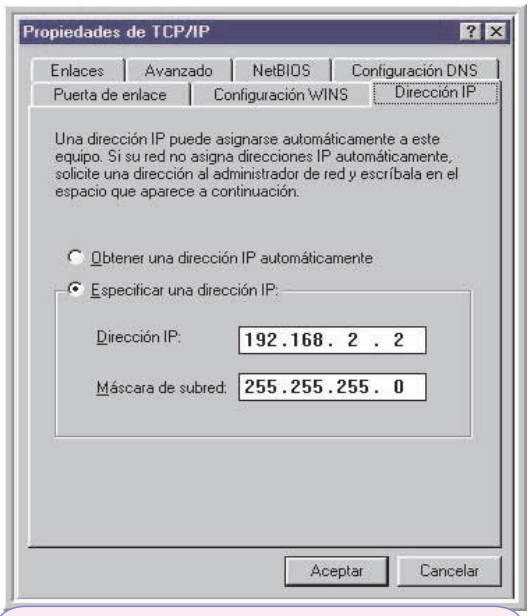
Ahora ya comprendemos la necesidad de separar por capas las comunicaciones entre máquinas para facilitar la comunicación, y reutilizar el trabajo ya hecho para no tener que reprogramarlo cada vez.

Y precisamente esas funciones que son utilizadas por muchos programas para que estos no tengan que reprogramarlas desde cero, son precisamente las funciones que te da la API de un **Sistema Operativo**.

Por ejemplo, un programa que funcione bajo Windows no tiene que preocuparse de saber cómo dibujar una ventana en la pantalla, si no que simplemente le dice al sistema "dibújame una ventana de estas características" y Windows hará el trabajo sucio por él.

Todavía recuerdo los tiempos en los que programaba aplicaciones gráficas en MS-DOS y me tenía que currar desde cero todo el interfaz... un auténtico infierno. Perdías más tiempo con el interfaz que con el propio programa.

Pues lo mismo que ocurre con las ventanas, que son una función común a todas las aplicaciones de Windows, también ocurre con las comunicaciones, que tienen una serie de funciones comunes a todas las aplicaciones de comunicaciones. Estas funciones comunes, que son las que proporciona el "idioma" TCP/IP, se ubican precisamente en el Sistema Operativo, para que sea él el que lidie con los detalles, igual que las ventanas las gestiona el Sistema Operativo, y es él el único que se preocupa de conocer los detalles para dibujarlas.



Configuración de TCP/IP en Windows.

Este es el motivo por el que, antes de conectar con Internet o con cualquier otra red, tenemos que configurar el protocolo TCP/IP en nuestro sistema.

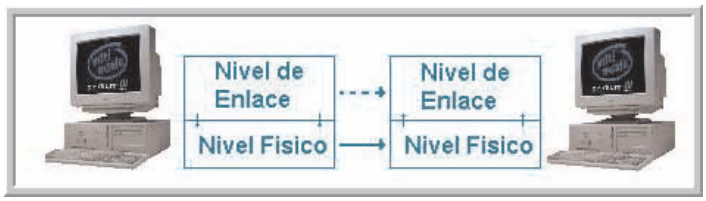
A lo largo del curso probablemente veremos cómo configurar correctamente el "idioma" TCP/IP con diferentes sistemas.

Por el momento, continuaremos con los conceptos sin entrar en ningún detalle. Ahora que ya habéis comprendido el concepto de capas, he de pediros que os olvidéis del ejemplo de los cirujanos, porque mi intención era únicamente que comprendieseis el concepto de capas, pero no mostrar metafóricamente cada capa del protocolo TCP/IP con su equivalente en el "mundo real", ya que las capas que forman TCP/IP no tienen prácticamente nada que ver con las capas que forman la comunicación entre dos cirujanos.

La única capa que sí que tienen en común tanto las máquinas como los cirujanos es la del **medio físico** ya que, como dije, esta capa no surge como una facilidad para la comunicación, si no que es una necesidad natural irremediable. Igual que dos cirujanos necesitan compartir un mismo medio para comunicarse, también han de hacerlo dos máquinas.

En el caso de las máquinas, hay que tener en cuenta no sólo la tecnología utilizada en los propios "cables" que interconectan las máquinas (que sería el medio físico) si no también el cómo están conectadas: cada cable conecta sólo dos máquinas, un sólo cable conecta a la vez a muchas máquinas entre sí, etc. Es decir, cómo se **enlazan** las máquinas entre sí. Ya veremos mucho más sobre esto más adelante, pero de momento nos quedaremos con la idea de que existe una segunda capa, conocida como **capa de enlace**.

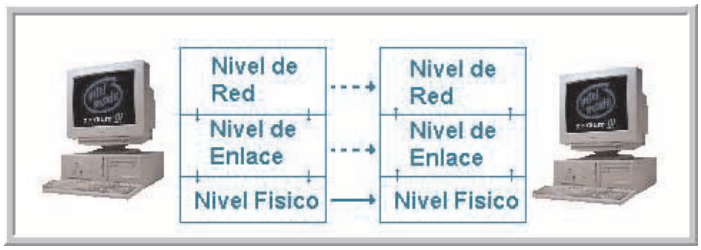
Para los impacientes, deciros que aquí es donde se ubicaría el famoso protocolo **ARP (Protocolo de Resolución de Direcciones)**, que es una parte de la capa de enlace utilizada normalmente en nuestros PCs.



Por encima del nivel de enlace, tenemos ya la primera capa que realmente nos sonará, que es la llamada **capa de red** y que, en nuestro caso (el caso de TCP/IP) es la capa llamada **IP**.

¿Cuál es la responsabilidad de esta capa? Pues principalmente una: conseguir que la comunicación llegue desde el origen hasta el destino. Ésta es la capa encargada de identificar a las diferentes máquinas, para que éstas puedan diferenciarse unas de otras, y mantener así comunicaciones separadas. Si no existiese esta capa, todos los datos de Internet llegarían a todas las máquinas conectadas a la red. No habría forma de ver una página Web, ya que no se podría diferenciar una de otra; no se podría enviar un e-mail, ya que no sabríamos cómo encontrar nuestro servidor de correo, etc., etc.

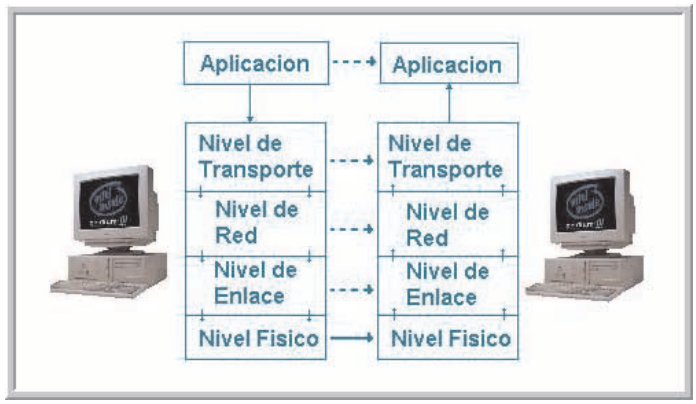
Supongo que esto no os sonará muy raro, teniendo en cuenta que las direcciones utilizadas en Internet se llaman precisamente **direcciones IP**. ¿Casualidad? ;-)



Pero, aún nos faltan muchas funciones comunes, ¿verdad? Como podéis adivinar, del resto de funciones se encarga la capa que está por encima de la capa IP que es, precisamente, la capa **TCP**.

En cualquier modelo (ya que TCP/IP no es el único modelo de protocolo por capas que existe) a esta capa se la conoce como **capa de transporte**.

Las funciones de esta capa son muchas, pero básicamente podríamos resumirlo en una: permitir el establecimiento de conexiones independientes y seguras, con todo lo que ello implica. Para comprender esto, así como para comprender mejor la capa IP, os muestro a continuación una serie de puntos que, de nuevo a base de símiles, os explicarán muchas de las funciones llevadas a cabo por cada capa.



3.1. La capa IP: La necesidad de direccional

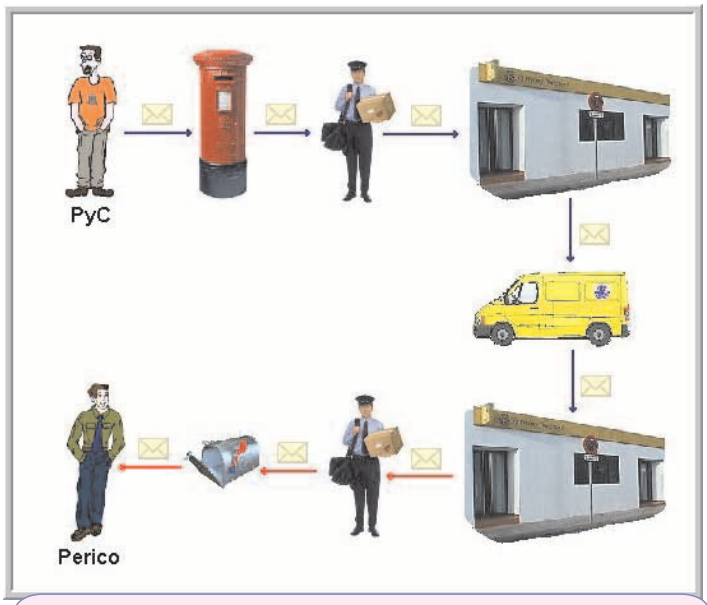
Para comprender la necesidad de la capa IP (Internet Protocol = Protocolo de Internet) presentaremos un nuevo escenario, bien conocido por todos, que es el **correo postal** de toda la vida.

En nuestro ejemplo, PyC, residente en Madrid, quiere enviar una carta a la siguiente dirección:

Perico Palotes
C/Pirulin. Nº12. 1º A.
35003 – Villapalotes (Huelva).

¿Cómo puede llegar hasta su destino la carta después de que PyC la eche en el buzón de su barrio? Todos conocemos bien el proceso. Primero, el cartero del barrio recoge todas las cartas del buzón, y las lleva a la oficina de correos más cercana. Una vez ahí, se ve que el destino de esta carta es Huelva y, por tanto, el sobre es transportado hasta esa provincia en un furgón de Correos. En esa provincia, el sobre se lleva a la oficina de correos

correspondiente. En esa oficina, comprueban la dirección, y llevan la carta hasta el buzón personal de Perico Palotes.



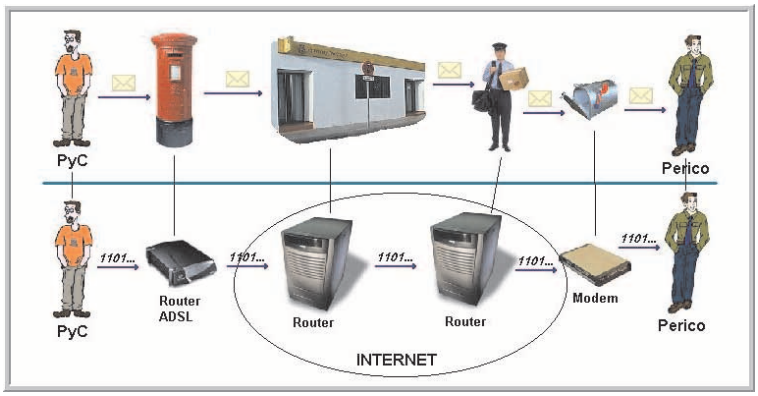
Funcionamiento básico del sistema de correo postal.

Gracias a las direcciones postales todo el sistema de Correos puede funcionar. Y no sólo gracias a la dirección del destinatario, si no también a la del remitente, que será una dirección con el mismo formato (nombre, calle, código postal, población, y provincia).

Gracias a la dirección del remitente se sabrá a quién informar si la carta no llega a su destino, y el destinatario podrá responder a la carta si lo desea.

Pues exactamente lo mismo ocurre con las direcciones de Internet. Aunque estas direcciones, las famosas IPs, aparentemente no consten de varios campos (nombre, calle, población, etc.), en realidad esos campos sí que existen, y están codificados dentro del propio número que forma la dirección IP.

Igual que existen las oficinas de correos, también existen una serie de máquinas dedicadas exclusivamente a transportar los "sobres" de Internet de una máquina a otra.
Estas máquinas mediadoras en la comunicación son conocidas habitualmente como routers. Estos routers, al igual que hacen las oficinas de Correos, se dedican a analizar las direcciones Ips para saber dónde tendrán que enviar el "sobre" para que, paso a paso, termine llegando a su destino.



Comparación entre los elementos del correo postal y los elementos de la comunicación de dos máquinas en Internet. Los furgones de correos, los carteros, y las oficinas de correos que forman la red postal, serían los routers que forman la red Internet. Igual que los carteros no abren las cartas, porque les basta sólo con ver el sobre, los routers tampoco ven el interior de nuestros paquetes, si no que miran sólo el equivalente al sobre para saber dónde mandarlos, como veremos más adelante.

3.2. La capa TCP (Transmission Control Protocol = Protocolo de Control de Transmisión) : La necesidad de las conexiones para tener una comunicación fiable

Volvamos ahora al escenario del hospital. En esta ocasión, el doctor PyC recibe un aviso de urgencia a través del servicio de megafonía del hospital.

Pero, ¿qué ocurriría si el doctor PyC no estuviese en ese momento en el hospital? ¿Y si estuviese en esos momentos escuchando música y no oyese el aviso? Cuando se lanza un aviso por megafonía se tiene cierta confianza en que el mensaje llegará a su destinatario, pero en realidad no se puede tener nunca esa certeza. Sueltas el mensaje al vacío, y no tienes forma de saber si al otro lado hay alguien escuchando. Esto es lo que se llama una comunicación no orientada a conexión.

Esta solución en muchos casos no es aceptable, pero en muchos otros sí que puede ser suficiente. Por ejemplo, en el caso de la megafonía del hospital, si el doctor PyC no ha acudido al aviso en un cierto tiempo, probablemente se le volverá a llamar. Existen otros casos de comunicación no orientada a conexión en los que directamente enviamos el mensaje con la esperanza de que llegue pero, si no llega, nos aguantamos y, a otra cosa, mariposa. Un ejemplo es el del envío de postales navideñas. Tú mandas un porrón, y si llegan o no a su destino es algo que en muchos casos nunca sabrás pero, lo que sin duda si que sabes, es que ni por asomo te vas a poner a reenviarlas por si acaso no han llegado.

Como en algunos casos la comunicación no orientada a conexión es suficiente, en las máquinas también es utilizada para algunos casos concretos. Cuando no necesitamos saber si nuestro interlocutor nos está escuchando, no necesitaremos utilizar un protocolo de transporte fiable, como es **TCP**, si no que nos bastará con utilizar un protocolo no orientado a conexión, que también os sonará bastante, y es el **UDP (Protocolo de Datagramas de Usuario)**.

Por tanto, UDP es también un protocolo de transporte e, igual que la mayoría de aplicaciones de comunicaciones utilizan como apoyo TCP/IP, también hay varias aplicaciones que en lugar de eso utilizan como apoyo UDP/IP.

Os remito de nuevo al número 14 de la revista, donde vimos un ejemplo de protocolo apoyado en UDP/IP, que es el protocolo DNS. A lo largo del curso ya veremos en profundidad el protocolo de transporte UDP.

Comentario de Hack x Crack...

Para aquellas mentes inquietas, apuntaremos un detalle.

El protocolo TCP podemos clasificarlo como "pesado" porque, al estar orientado a la conexión, consume muchos más recursos de red, es decir, el proceso para establecer una comunicación de este tipo está formada por muchos pasos.

El protocolo UDP podemos considerarlo como "ligero" porque, al no estar orientado a la conexión, consume pocos recursos de red, es decir, el proceso tiene muy pocos pasos.

El protocolo UDP es muy utilizado, por ejemplo, en la emisión de video por Internet o los programas de intercambio de archivos tipo eMule.

Para que se entienda, en la transmisión de video en tiempo real por Internet (por ejemplo), lo que interesa es que al receptor le llegue la máxima información posible en el menor espacio de tiempo posible, no importa si se pierden unos cuantos frames de película por el camino (es imperceptible), lo importante es que la película no se pare y se pueda ver fluida en tiempo real.

Sería absurdo que si el frame 7 no llega, se parase la película, pidiésemos de nuevo al emisor el frame 7, esperásemos su llegada, la comprobásemos y volviésemos a activar la película. Si pensamos que llegan entre 15 y 30 frames por segundo, bufff, estaríamos parando la película cada dos por tres... es mejor "despreciar" ese frame que no ha llegado y seguir con la "peli" :)

En el caso de los programas de intercambio de archivos tipo P2P (como el eMule, http://www.emule-project.net/), el tema se complica un poquito, pero solo un poquito.

Si estamos descargando un programa llamado "officexp.zip" (de 650MB) desde 7 usuarios a la vez, este nos llega en trocitos pequeños. Lo importante es que nos lleguen cuanto más trocitos mejor y en el menor espacio de tiempo. Pero claro, también es importante que no perdamos ningún trocito (o después el ZIP nos dará errores).

En este caso, podríamos pensar que es mejor utilizar TCP, puesto que nos asegura que llegan todos los trocitos; pero entonces estaríamos sobrecargando la red P2P con centenares de peticiones de comprobación y la descarga sería muy lenta. ¿Cómo resolvemos esto?

Pues trabajamos con UDP y hacemos que sea el programa P2P quien compruebe si faltan trocitos. En caso de faltar algún trozo se reclama y en caso de no faltar no se reclama.

PARALELISMO: Por si alguien no lo ha pillado, retomemos el caso del doctor y hagamos un paralelismo con el "mundo" P2P.

DOCTOR en un sistema TCP en un mundo P2P :)
El doctor (receptor) recibe por megafonía un mensaje (archivo) PERO el tipo del megáfono (emisor) es MUY EXIGENTE y OBLIGA al doctor (receptor) que confirme la correcta recepción de cada palabra (parte del archivo) que recibe.

El mensaje (archivo) a transmitir es: PRESÉNTESE INMEDIATAMENTE EN EL QUIRÓFANO.

1.- El tipo del megáfono (emisor) emite la primera palabra (primera parte del archivo): PRESÉNTESE

2.- El pobre doctor (receptor) va corriendo a un teléfono, llama al tipo del megáfono y le dice que ha recibido correctamente la palabra (trozo de archivo): PRESENTESE

3.- El tipo del megáfono (emisor) emite la segunda palabra (segunda parte del archivo): INMEDIATAMENTE

4.- El pobre doctor (receptor) va corriendo a un teléfono, llama al tipo del megáfono y le dice que ha recibido correctamente la palabra (trozo de archivo): INMEDIATAMENTE

5.- Y así seguiremos hasta que el doctor (receptor) confirma la llegada de la última palabra (trozo de archivo) QUIRÓFANO. En ese momento el doctor (receptor) une todas las palabras (trozos de archivo) y obtiene el mensaje (archivo): PRESÉNTESE INMEDIATAMENTE EN EL QUIRÓFANO.

Como podemos ver, la comunicación es 100% segura, el doctor confirma la llegada de cada palabra (trozo de archivo); pero han invertido mucho tiempo y muchas llamaditas de confirmación.

DOCTOR en un sistema UDP en un mundo P2P :)
El doctor (receptor) recibe por megafonía un mensaje (archivo) PERO el tipo del megáfono (emisor) es MUY DESPREOCUPADO y no le importa si el doctor (receptor) recibe o no el mensaje.

El mensaje (archivo) a transmitir es: PRESÉNTESE INMEDIATAMENTE EN EL QUIRÓFANO.

1.- El tipo del megáfono (emisor) emite la primera palabra (primera parte del archivo): PRESÉNTESE

2.- El doctor (receptor) en teoría recibe la primera palabra (primera parte del archivo): PRESÉNTESE

3.- El tipo del megáfono (emisor) emite la segunda palabra (segunda parte del archivo): INMEDIATAMENTE

4.- El doctor (receptor) en teoría recibe la segunda palabra (segunda parte del archivo): INMEDIATAMENTE

5.- Seguiríamos así hasta que el doctor (receptor) "en teoría" recibiese la última palabra (último trozo del archivo). En ese momento el doctor (receptor) uniría todas las palabras (trozos de archivo) obteniendo el mensaje completo (archivo).

Como podemos ver, la comunicación es mucho más rápida (nos ahorramos las confirmaciones) pero… ¿y si el doctor se ha perdido alguna palabra?... quizás se ha perdido la palabra INMEDIATAMENTE… si el doctor estaba tomándose un café quizá prefiere acabárselo antes de acudir al quirófano (y seguro que su paciente muere desangrado).

En el caso de emisión de video por Internet en tiempo real ya hemos dicho que no nos importaba perder unos cuantos frames, pero en el caso del P2P QUEREMOS TODOS los trocitos de archivo, queremos que el doctor no deje morir a su paciente… … ¿Cómo lo solucionamos si no queremos utilizar el sistema TCP?

Tenemos un problema: No queremos sobrecargar la red telefónica del hospital confirmando cada palabra que recibimos (TCP) pero queremos asegurarnos de recibir los mensajes completitos. Muy bien, pues en este caso tendremos que dotar a los intervinientes (el Sr. del megáfono y el doctor) de un poquito de inteligencia :)

El Sr. del megáfono (software emisor) y el doctor (software receptor) se reúnen y después de un buen rato discutiendo el problema llegan a una solución. Deciden utilizar el sistema UDP

"pero" cada palabra (trozo de archivo) emitida tendrá estará precedida de un número correlativo (número de control). Vamos a verlo.

DOCTOR en un sistema UDP en un mundo P2P (con control añadido).
El doctor (receptor) recibe por megafonía un mensaje (archivo) con un sistema previamente pactado :)

El mensaje (archivo) a transmitir es: PRESÉNTESE INMEDIATAMENTE EN EL QUIRÓFANO.

Según han pactado, el mensaje a transmitir será: UNOPRESÉNTESE DOSINMEDIATAMENTE TRESEN CUATROEL CINCOQUIRÓFANO.

1.- El tipo del megáfono (emisor) emite la primera palabra (primera parte del archivo): UNOPRESÉNTESE

2.- El doctor (receptor) en teoría recibe la primera palabra (primera parte del archivo): UNOPRESÉNTESE

3.- El tipo del megáfono (emisor) emite la segunda palabra (segunda parte del archivo): DOSINMEDIATAMENTE

4.- El doctor (receptor) en teoría recibe la segunda palabra (segunda parte del archivo): DOSINMEDIATAMENTE

5.- Seguiríamos así hasta que el doctor (receptor) "en teoría" recibiese la última palabra (último trozo del archivo). En ese momento el doctor (receptor/software receptor) comprobaría que tiene en su poder las palabras (trozos de archivo) y que no falta ninguna (se puede comprobar gracias a que tienen números correlativos).

Solo en caso de que faltase alguna palabra (trozo de archivo) el doctor llamaría por teléfono al emisor pidiéndole UNICAMENTE la palabra que le falta.

Como podemos ver, ahora la conexión sigue siendo del tipo UDP (cargamos poco la red); pero gracias a que hemos dotado al Sr. del megáfono y al doctor (software emisor y receptor) de "inteligencia" (software), hemos conseguido además bastante seguridad en la comunicación.

ACABANDO Y PUNTUALIZANDO:

Acabamos de aprender algo importantísimo que ya nunca deberíamos poder olvidar. Una cosa es el tipo de protocolo que estamos utilizando para nuestras conexiones (TCP o UDP e incluso ambos a la vez) y sus consecuencias sobre la red y, OTRA MUY DISTINTA, cómo programamos el software para mejorar el rendimiento de dichas conexiones.

Hemos visto que las carencias de seguridad del protocolo UDP (capa de transporte) han sido "salvadas" gracias a cómo hemos programado el software (nivel de aplicación).

PARA LOS QUISQUILLOSOS:
- Si, pero... ¿y si el doctor (software receptor) no recibe ninguna palabra (trocito de archivo) porque está dormido? Pues entonces dotamos de un poco más de inteligencia al programa para que la primera palabra (trocito de archivo) se haga por TCP (confirmacion aobligatoria) y el resto por UDP. De esta forma no se emitirán mas palabras (trocitos de archivo) por megafonía hasta que el doctor llame al Sr. del megáfono confirmando que la recibido la primera palabra.

- Si, pero... ¿y si el doctor (software receptor) no confirma la recepción de esa primera palabra (trocito de archivo)? Pues hacemos que el Sr. de megafonía (software emisor) envíe un mensaje al teléfono móvil del doctor cada 5 minutos durante 2 horas hasta que conteste.

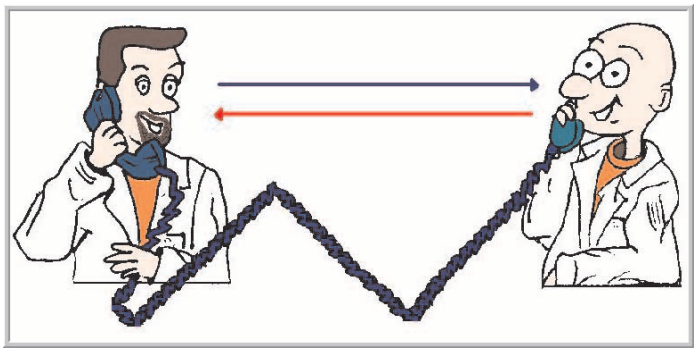
¿Y si a pesar de todo no contesta? Pues llamamos a otro doctor mientras el primero está dormido (en un P2P sería el equivalente a servir el archivo a otro cliente mientras el primero pasa a una lista de espera :)

La intención de esta extensa nota no es otra que ACERCAR ese extraño mundo de las capas OSI a la realidad, a programas que utilizamos diariamente y que no tenemos ni idea de cómo funcionan (por ejemplo la visualización de video en tiempo real y los P2P). Quizás ahora pensemos un poco más en lo que hay detrás de esas cosas que utilizamos mecánicamente sin pensar :)

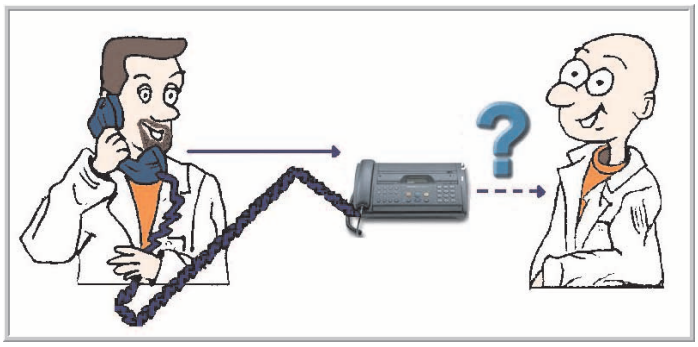
Olvidándonos ya de UDP, vamos a ver entonces qué es TCP, que es el que más nos interesa. A diferencia de las comunicaciones no orientadas a conexión, las orientadas a conexión son aquellas en las cuales hay un diálogo directo con el interlocutor. Es decir, no es ningún monólogo que sueltas con la esperanza de que alguien te escuche, si no que es una conversación entre dos o más

interlocutores, donde todos saben en todo momento si están siendo escuchados por los demás.

Como ejemplo de comunicación orientada a conexión tenemos el teléfono, donde en todo momento sabes si la persona con la que estás hablando está siguiendo el diálogo. Por el contrario, cuando hablas con un contestador automático telefónico, se trata precisamente de una comunicación no orientada a conexión.



Al hablar por teléfono mantenemos una conversación orientada a conexión, donde ambas partes saben que el otro está escuchando.



Al hablar con un contestador telefónico mantenemos una conversación no orientada a conexión, pues no sabemos si la otra parte llegará a escuchar nuestro mensaje.

La mayoría de servicios de comunicación entre máquinas requieren una comunicación orientada a conexión. Por ejemplo, si estas transfiriendo un fichero, normalmente necesitarás saber que éste está siendo recibido, si estás enviando un e-mail necesitarás saber que tu servidor de correo lo está recibiendo (aunque si llega al buzón del destinatario o no es ya un asunto aparte), si estás en un Chat necesitas saber que la persona o personas con las que hablas están conectadas en ese momento, y leyéndote.

Hasta donde la capa IP entiende, sólo existen sobres que circulan desde una dirección de remitente hacia una dirección de destinatario, pero en ningún momento existe un diálogo entre remitentes y destinatarios. Es en la capa TCP donde aparece este nuevo concepto, que engloba los sobres que para IP circulan por separado, en un único flujo de diálogo entre las dos partes.

En el caso de TCP, existen unos "sobres" especiales que lo único que hacen es decir "te estoy escuchando". Cada vez que un remitente envíe un sobre a un destinatario, éste responderá con un nuevo sobre, en el que simplemente dirá "te he escuchado".

Gracias a este mecanismo, se puede saber en todo momento si estamos hablando solos, o estamos en un diálogo.

3.3. La capa TCP: El tamaño de los paquetes.

Pero, ahora que sabemos que el protocolo TCP exige a los destinatarios que constantemente confirmen que han recibido lo que se les enviaba, ¿qué ocurre si en algún momento no se recibe esa confirmación? Pues el remitente, transcurrido un tiempo en el que no haya recibido confirmación, tendrá que reenviar el paquete y esperar de nuevo la confirmación, como en el ejemplo de la megafonía del hospital.

Pero, ¿qué ocurre entonces si el mensaje de megafonía era muy largo? Imaginemos que, en lugar de decir: "Doctor PyC, acuda a la sala de cirugía cardiovascular", el

mensaje dijese: "Doctor PyC, acuda a la sala de cirugía cardiovascular para atender una urgencia de cardiopatía precarótida en un paciente varón de 70 años, diabético, de grupo sanguíneo AB+, y cuyo color preferido es el fucsia". Si el Doctor PyC no respondiese a la primera llamada, habría que repetir toda la parrafada de nuevo.

No tiene sentido soltar parrafadas muy largas si no tienes la certeza de que estás siendo escuchado. Por eso, si lo que tienes que transmitir es muy largo, lo mejor es que lo vayas contando poco a poco, y esperando la confirmación de que cada parte ha sido escuchada.

Cuando hablamos por teléfono, normalmente no soltamos un rollo de varias horas sin parar (aunque los hay que si...), si no que estamos pendientes de que cada cierto tiempo nuestro sufrido interlocutor nos dé las confirmaciones de rigor como "si, si", o "aja", o "que te calles ya". Normalmente, si llevamos dos minutos seguidos hablando y no hemos escuchado un "aja" de nuestro interlocutor, nos mosqueamos bastante, y decimos "oye, ¿sigues ahí?".

En resumen, lo natural a la hora de transmitir mucha información es hacerlo en pequeños trozos, cada uno de los cuales confirmará su recepción por separado.

Lo mismo ocurre en la comunicación entre máquinas. Como TCP se encarga de enviar confirmaciones, es también el que se encarga de partir los paquetes muy grandes en paquetes más pequeños para que estas confirmaciones puedan llegar poco a poco, y no tener que retransmitir todo si no llegase la confirmación.

Esto nos permite, además, adaptarnos a la capacidad de nuestro interlocutor. Por ejemplo, si nos suscribiésemos a una enciclopedia por fascículos, y nos enviasen toda la colección de golpe, probablemente el cartero mandaría al garete a los tíos de Espasa, y les diría que los 20 volúmenes los iba a llevar hasta allí su simpática abuela. Los buzones de correos tienen un tamaño limitado y, si bien cada fascículo por separado cabe perfectamente en el buzón, la colección entera no cabría en ningún buzón. Lo mismo ocurre con las máquinas, que tienen un buzón de recepción de un tamaño limitado, y hemos de ajustarnos a esas limitaciones tecnológicas.

3.4. La capa TCP: Las conexiones simultáneas

Una de las principales funciones de la capa TCP es la de permitir que existan varios diálogos simultáneos entre dos interlocutores. Aquí no recurriré a más metáforas, si no que será más sencillo verlo directamente en nuestro campo de trabajo.

Si, por ejemplo, está PyC en MSN chateando con Scherzo, y a la vez le está enviando un archivo, ¿no estarán manteniendo dos diálogos simultáneos? Por un lado, están chateando, y por otro lado están enviando un archivo.

Suponiendo que un Chat en MSN funcionase mediante una conexión punto a punto (que no es así, como sabréis si habéis leído mi artículo sobre MSN, pero imaginaremos que sí), habría una serie de paquetes cuyo remitente sería PyC y cuyo destinatario sería Scherzo, pero de esos paquetes algunos serían parte del archivo que se está transfiriendo (que, por supuesto, estaría partido en trozos, tal y como vimos en el punto anterior), y otros serían parte de la conversación que mantienen PyC y Scherzo a través del Chat.

Para permitir que esto ocurra, el protocolo de transporte, TCP, tiene que tener algún sistema que identifique qué paquetes son del Chat, y qué paquetes son del archivo. Esto lo hace asignando un número a cada diálogo simultáneo y, según el número que haya en cada paquete, sabrá si éste forma parte del archivo, o del Chat.

Pues estos números mágicos de los que estoy hablando no son otros que los archiconocidos **PUERTOS**.

Un puerto es un campo del protocolo TCP que permite identificar el servicio al que va destinado cada paquete en una conexión entre dos máquinas.

Así, cada vez que una máquina reciba un paquete con el número de puerto 25, sabrá que ese paquete es un e-mail, cada vez que reciba un paquete con el número de puerto 21, sabrá que ese paquete es un comando de FTP, cada vez que reciba un paquete con el número de puerto 80 sabrá que es una conexión Web, etc., etc.

4. Ejemplo: Enviando un archivo.

Para recapitular todas las ideas mostradas a lo largo del artículo, termino con un ejemplo bastante completo que muestra paso a paso el envío de un archivo de PyC a Scherzo.

Estad muy atentos a cada paso, porque espero que este ejemplo os ayude a comprender mucho mejor todos los conceptos que necesitareis para seguir el resto del curso. Fijad también vuestra atención en todas las ilustraciones, pues muestran gráficamente toda la secuencia del ejemplo, y además los datos que aparezcan en las propias ilustraciones son también fundamentales.

A lo largo de la serie RAW os he explicado ya varios sistemas de transferencia de archivos (FTP, DCC, MSN,...). En este ejemplo usaremos, por ejemplo, una transferencia por FTP.

Antes de nada, vamos a ver cómo sería el proceso si sólo nos fijásemos en la capa de arriba, es decir, en la capa sobre la que he ido hablando mes tras mes en la serie RAW.

1. **PyC abre su servidor FTP**: pone un puerto en escucha, gracias a una función que da el sistema operativo que permite a cualquier aplicación realizar estas y otras funciones de TCP/IP.
2. **Scherzo abre una conexión con el servidor FTP de PyC**: el modo en que se abre esta conexión lo detallaremos a lo largo del curso, pero no en este artículo. De momento lo que sí que sabemos es que la responsable de abrir y mantener las conexiones es la capa TCP.
3. **Scherzo escoge el archivo que quiere bajar**: comandos CWD, CDUP, LIST,... todo esto ya lo vimos en los artículos sobre FTP de la serie RAW, y ahora no nos interesa mucho.
4. **Scherzo inicia la transferencia del archivo**: comandos PORT o PASV, y RETR o REST. También lo vimos en los artículos de la serie RAW, y tampoco nos interesa ahora.
5. **El archivo se transfiere desde el servidor de PyC hacia el cliente de Scherzo**: Aquí unos enanitos se encargan de llevar el archivo de una máquina a otra, cargando los datos en sacos que llevan a la espalda. Pero... ¡espera! ¡Si esto no es la serie RAW! En la serie RAW no me quedaba más remedio que deciros estas cosas, porque al llegar a este punto no podía daros más detalles, ya que más de una vez os mencioné que explicar lo que ocurre en estos momentos sería suficiente para llenar no sólo un artículo, si no una serie entera. ¡Y al fin ha llegado esa serie! Así que esperad unas cuantas líneas, que enseguida os explico cómo funcionan las cosas realmente. Tampoco quiero chafar la ilusión a nadie, así que si alguien no quiere dejar de creer en los enanitos que transportan paquetes de datos, que no siga leyendo! ;-)
6. **Finaliza la transferencia del archivo:** y a otra cosa, mariposa.

¿Para qué os he mostrado todos estos pasos que conocéis ya perfectamente (sobre todo si habéis seguido la serie RAW)? Pues sencillamente, para que veáis que entre los pasos 5 y 6 ocurren una gran cantidad de cosas que siempre hemos obviado, y que serán las que precisamente detalle en este ejemplo.

Nos olvidaremos, por tanto, del resto de pasos, y nos centraremos únicamente en lo que ocurre

desde que comienza la transferencia del archivo, hasta que ésta finaliza.
Algunos conceptos los simplificaré mucho, y otros incluso los obviaré, esperando así facilitar la comprensión de los conceptos fundamentales, aunque tenga que sacrificar parte de la verdad.

4.1. En el servidor FTP de PyC

Empezamos nuestro viaje por el mundo de los enanitos en el reino que nosotros conocemos, que es el del servidor FTP de PyC.

El servidor lo único que sabe es que tiene un archivo de 4500 KB que tiene que enviar al usuario Scherzo, que previamente hizo un login en el servidor.

El que programó el servidor FTP no tenía ni idea de TCP/IP, así que lo único que sabía era que el no tenía más que dar una orden al sistema operativo, y éste se encargaría de hacer todo el trabajo sucio. Así que el software del servidor simplemente llama a una función mágica del sistema operativo, parecida a esta:

EnviaFichero (fichero.doc, usuario)

Donde fichero.doc es el nombre del archivo que quiere enviar, y usuario es una variable que contiene un número que identifica al usuario Scherzo.

El valor de esa variable usuario no lo asignó el programa de FTP, si no que lo hizo el propio sistema operativo en el momento en que Scherzo se conectó al servidor.

Como es el sistema operativo el que maneja estos números mágicos que identifican a las conexiones (que, como veremos a lo largo del curso, se llaman sockets), el programa de FTP podrá pasarle este dato al sistema, y éste ya sabrá perfectamente lo que hacer con él.

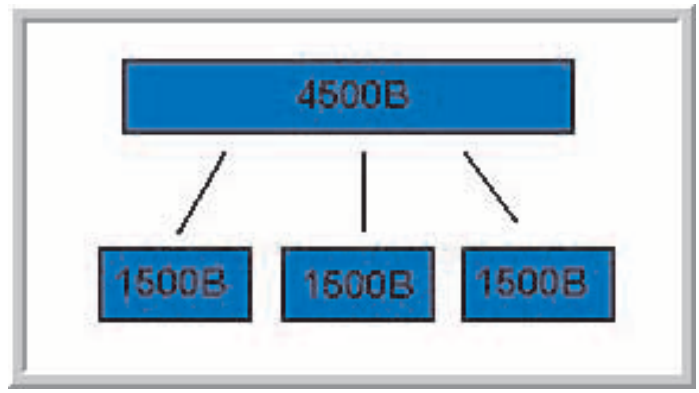
Una vez que el programa FTP llama a esta función mágica del sistema operativo, lo próximo que verá será la respuesta del sistema operativo **diciéndole "ale, ya está transmitido el archivo**", o bien cualquier error que haya podido ocurrir "**oye, no he podido enviarlo porque el usuario se ha desconectado**", etc.

Como esto es todo lo que ve el programa de FTP, tendremos que descender al oscuro reino del sistema operativo para comprender mejor qué es lo que ocurre desde que se llama a esa función hasta que el sistema operativo avisa de que el envío ha finalizado.

4.2. En el Sistema Operativo de PyC: La capa TCP.

Lo primero que hace la capa TCP nada más ver el archivo de 4500KB es decir: "buf, este chorizo es demasiado largo". ¿Qué criterio utiliza TCP para decidir si algo es demasiado largo? Es algo que veremos a lo largo del curso, pero de momento nos basta con saber que el tamaño máximo de un paquete es algo que determinó previamente la capa TCP según las limitaciones de la red.

Supongamos que ha determinado que el tamaño máximo de paquete son 1500KB. Tendrá que coger el archivo y partirlo en tres trozos, cada uno de 1500KB.



A cada trozo le asigna un número que determina la secuencia en la que han de unirse de nuevo los trozos. A este número se le llama precisamente número de secuencia.

Una peculiaridad del número de secuencia es que no se numera según el número de trozo,

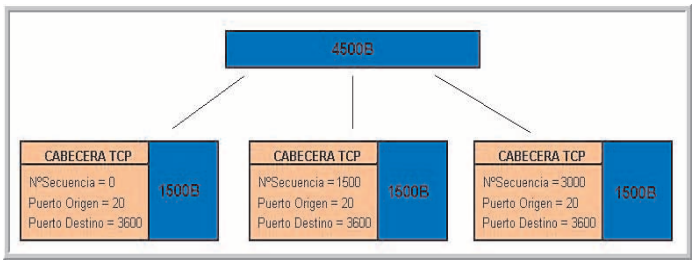
si no según el número de byte del archivo en el que empieza ese trozo. Así, el primer trozo tendrá un número de secuencia 0, el segundo un número de secuencia 1500, y el tercero un número de secuencia 3000.

Pero, ¿es éste el único dato que ha de asignar la capa TCP a cada trozo? Pues me temo que no, ya que sabemos bien que las responsabilidades de esta capa van más allá de simplemente partir los paquetes en bloques. Hablamos también de los números de puerto, así que tendrá que añadir a cada trozo los dos puertos implicados en la conexión. ¿Adiviná is cuáles son estos puertos?

Je, je... era una pregunta con trampa. Si habéis pensado que uno de los puertos era el 21, puerto asignado al servicio FTP, os habéis equivocado, aunque he de reconocer que lo he preguntado a mala leche. 0:-)

Si repasáis mejor el artículo de la serie RAW sobre FTP veréis que el puerto 21 sólo se utiliza para enviar comandos al servidor, pero no para la transferencia de archivos. Para ésta se utiliza otro puerto que es acordado previamente mediante el comando **PORT** o el comando **PASV**. Para los que no sepáis de qué hablo porque no conocéis el protocolo FTP, olvidaos de todo esto y quedaos simplemente con la idea de que tanto el puerto de PyC (origen) como el puerto de Scherzo (destino) son números que han acordado previamente, por ejemplo, el 20 y el 3600.

Por tanto, si añadimos a cada trozo los tres numerajos que hemos dicho (numero de secuencia, puerto de origen, y puerto de destino) nos queda el siguiente escenario:



Los tres paquetes que forman el archivo, con sus correspondientes cabeceras TCP. Fijaos bien en los campos que forman la cabecera.

La capa TCP ya ha terminado su trabajo inicial, y de momento se puede relajar un poco mandando los bloques a la capa IP, que sabrá qué hacer con ellos a continuación. Pero, a diferencia del programa de FTP, la capa TCP no se puede dormir en los laureles esperando a que la transferencia del archivo termine, si no que tendrá que seguir trabajando más adelante, tal y como iremos viendo.

4.3. En el sistema operativo de PyC: La capa IP.

En cuanto TCP llama a la capa IP, y le pasa los 3 bloques, ésta se pone en marcha. Su labor principal consiste en añadir a cada bloque las direcciones IP de PyC y Scherzo para que, una vez que los bloques estén flotando por el ciberespacio, todos aquellos mediadores por los que pasen sepan dónde llevarlos.

La dirección IP de PyC la conoce, por supuesto, el propio sistema operativo de PyC (bien porque la introdujimos nosotros manualmente al configurar nuestra LAN, bien porque el sistema la asignó automáticamente mediante DHCP, o bien porque se nos asignó una IP dinámica al conectar con nuestro ISP).

La dirección IP de Scherzo la puede obtener el sistema a partir de la variable usuario ya que, cuando Scherzo conectó con el programa FTP, el sistema operativo automáticamente asoció la IP de Scherzo a esa variable.

¿Tiene que añadir algo más la capa IP? ¡Pues si! Vamos a ver. ¿Para qué servía en la capa TCP meter los números de puerto en cada bloque? Si no metiésemos los números de puerto, cuando el bloque llegase al destinatario (Scherzo) éste no sabría qué hacer con él. Scherzo probablemente tendría abiertas en ese momento varias aplicaciones de comunicaciones, y cada paquete que le llegase tendría que indicar de algún modo a cuál de todas esas aplicaciones iba dirigido.

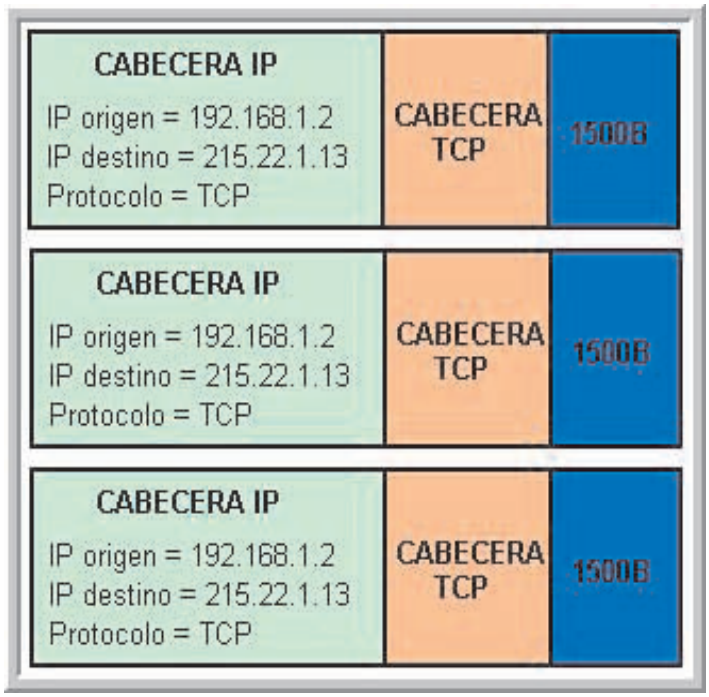
Pues algo parecido ocurre con la capa IP. Como ya vimos, no sólo existe TCP como protocolo

de transporte, si no también otros como UDP. Además, existen otra serie de protocolos que funcionan sobre IP, como ICMP, pero de eso no vamos a hablar ahora.

Lo que ahora nos interesa es que de alguna manera el bloque tiene que decir de alguna forma que es un bloque TCP, y no UDP. Para ello, la capa IP asigna mete un numerajo en cada bloque que identifica el protocolo de transporte desde el que le llegó ese bloque.

Como, en este caso, los bloques le llegaron a IP desde la capa TCP, la capa IP meterá en cada bloque un número que dirá que ése es un bloque TCP.

Por tanto, así nos queda el escenario ahora:



Los tres paquetes que forman el archivo, con sus cabeceras TCP e IP. Como podéis observar, la cabecera IP es igual para los tres paquetes.

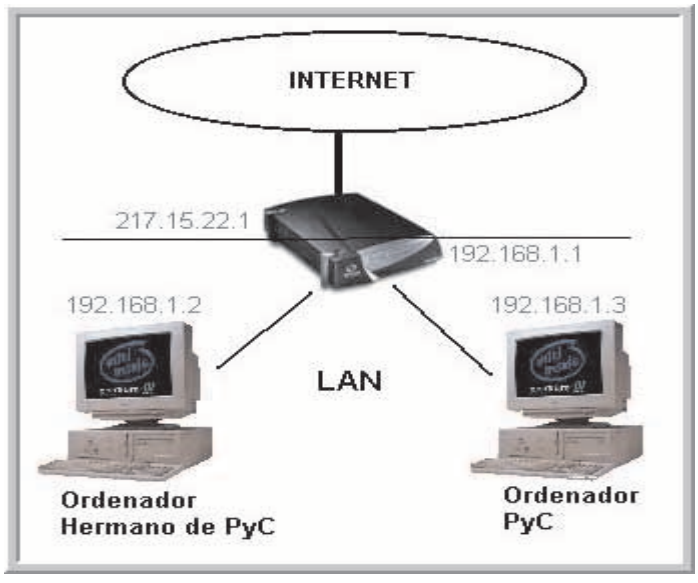
Una vez que los paquetes ya están listos, se los manda a la capa de abajo, la capa de enlace, de la que hemos hablado poco hasta el momento.

4.4. En el sistema operativo de PyC: la capa de enlace.

Hasta el momento hemos hablado poco de esta capa para evitar liaros más. Si bien las capas TCP e IP son iguales para todos los sistemas, la capa de enlace puede variar totalmente de un sistema a otro. Es aquí donde se encuentran las diferencias entre una conexión por módem, una conexión ADSL, y cualquier otro tipo de conexión.

Es decir, la capa de enlace depende de la tecnología utilizada para crear el enlace entre nuestra máquina e Internet.

Vamos a suponer que el enlace que tiene PyC es el más común hoy día, es decir, un enlace Ethernet. Éste es el que usamos si tenemos una tarjeta de red (bien con ADSL, o con cable, o con otras tecnologías).
Supongamos que PyC tiene en su casa una configuración muy habitual hoy día, que es la de una conexión a Internet a través de un router ADSL, con una pequeña LAN (red local) en casa compuesta por dos ordenadores: el suyo, y el de su hermano.



Esquema de la LAN de PyC, con las IPs de cada elemento. El router ADSL tiene 2 IPs: 192.168.1.1 para la LAN, y 217.15.22.1 de cara a Internet.

Como vemos en la ilustración, la única máquina que se comunica directamente con Internet es el router ADSL, y es éste el que tiene que encargarse de llevar todos los paquetes de PyC y de su hermano a Internet. Para ello, ambos ordenadores están conectados al router mediante un cable de red (si el router sólo tiene un conector RJ-45 tendría que haber en medio un switch, pero esa cuestión la obviamos por salirse del tema).

El problema aquí es que la comunicación en Ethernet es de tipo **broadcast**.

Esto significa que lo que circula por el cable llega a todas las máquinas que estén conectadas al mismo, y hay que idear alguna forma de hacer que sólo atienda a los datos la máquina interesada.

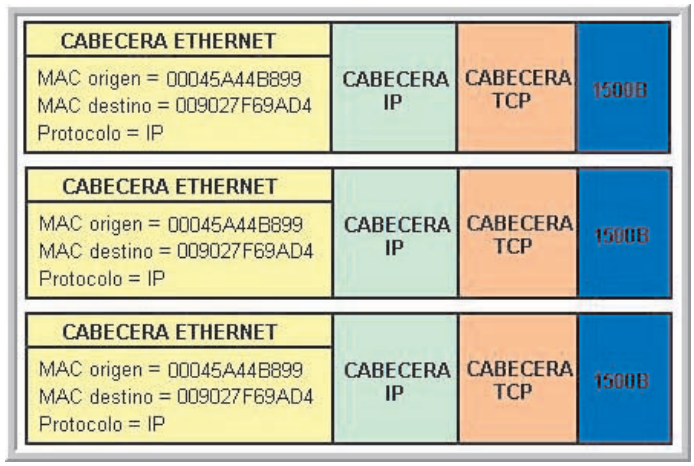
Este es precisamente el motivo por el que funciona un sniffer en una red local. Todos los datos de la red local circulan por el cable al que se conecta tu tarjeta de red, y sólo hay que "engañar" a la tarjeta de red para que te muestre todos los datos que hay en el cable, y no sólo los que van dirigidos a ella.

Pero lo que nos interesa aquí es conocer el mecanismo utilizado para distinguir unas máquinas de otras en una LAN.

Como podéis imaginar, esto se consigue asignando un número diferente a cada tarjeta de red de las máquinas conectadas a la LAN. Este número es único en el mundo para cada tarjeta de red.

Los fabricantes de dispositivos Ethernet tienen un acuerdo para que nunca se fabriquen dos tarjetas de red con el mismo número. Este número mágico es precisamente la famosa **dirección MAC (comúnmente conocida como dirección física)** de la que probablemente habréis oído hablar.

Al haber una MAC diferente por cada dispositivo Ethernet del mundo, las direcciones MAC tienen que ser lo suficientemente grandes como para que nunca tengan que repetirse. Estas direcciones, más largas que las direcciones IP, constan de 48 bits, por lo que teóricamente permiten identificar casi 300 Billones de dispositivos Ethernet diferentes. Entonces, ¿qué datos añadirá el nivel Ethernet a cada bloque que queremos transmitir? Pues, al igual que la capa IP, añadirá una dirección MAC de origen, y una dirección MAC de destino. Y, también igual que en la capa IP, tendrá que añadir un dato más, que es un identificador de la capa superior que le pasó los bloques, es decir, en este caso la capa IP.

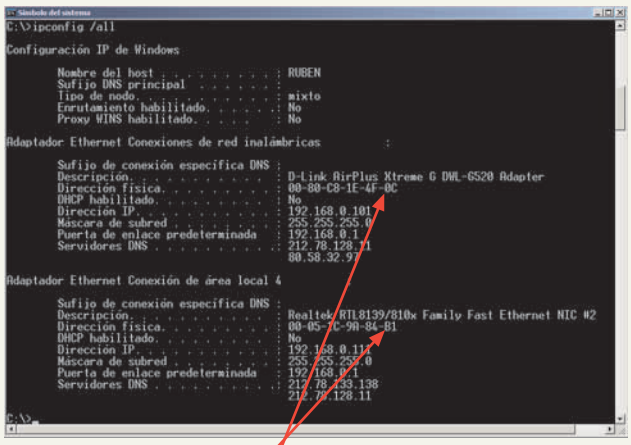


Los tres paquetes que forman el archivo, con sus cabeceras TCP, IP, y Ethernet. En ésta la MAC de origen será la de PyC, y la MAC de destino la del router ADSL de PyC, que será el próximo punto con el que habrá que enlazar la comunicación. El ordenador de PyC conoce la dirección MAC del router gracias al protocolo ARP, pero eso se sale del tema del artículo.

¿No sabes...

¿No sabes la MAC de tu tarjeta de Red? ¿de verdad?... bueno, bueno… tendrías que haber leído los anteriores números de esta revista :)

Abre una ventana de comandos (Menú inicio --> Todos los Programas --> Accesorios --> Símbolo del sistema). Escribe IPCONFIG /ALL. Pulsa enter y ZAS!!! Ahí tienes la MAC de tu/s tarjetas Ethernet (Tarjetas de Red).



En este caso podemos ver 2 tarjetas de red y cada una tiene una MAC (dirección física). La tarjeta D-Link AirPlus tiene la MAC 00-80-C8-1E-4F-0C y la Realtek tiene la MAC 00-05-1C-9A-84-B1

Como apunte interesante, aclarar que aunque la MAC es única en el mundo, existen Tarjetas de Red especiales y otros dispositivos (como algunos routers) que permiten "falsear" la MAC (poner la MAC que tu quieras ;). La mayoría de técnicos de Red tienen entre sus "herramientas de trabajo" una de estas tarjetas para comprobar el buen funcionamiento de los Sistema de Red.

4.5. En la tarjeta de red de PyC: La capa física

Una vez que el nivel de enlace ya ha hecho lo que tenía que hacer con los bloques, no tiene más que acceder ya directamente al hardware, en este caso la tarjeta de red, y los datos salen al fin de la máquina de PyC.

Por supuesto, existen muchas tecnologías diferentes en la capa física: RJ-45 (la que casi todos tenemos), coaxial, inalámbricas (muy de moda últimamente), etc., etc.

Una vez lanzados los paquetes a través del cable que une la máquina de PyC con el router ADSL estos, por supuesto, llegarán hasta el router.

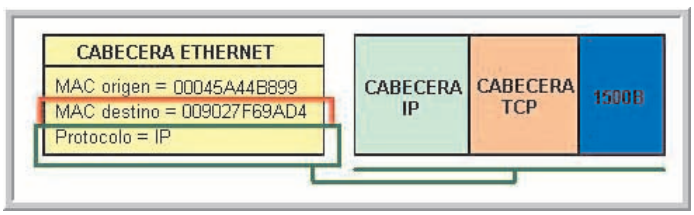
El primer paquete al fin sale del ordenador de PyC, hacia el router ADSL.

4.6. En el router ADSL de PyC: Las capas física y de enlace

Una vez que los datos llegan al router, éste detectará mediante hardware la presencia de datos en el cable, y enviará los bloques recibidos a su capa de nivel de enlace. Aquí es importante tener en cuenta que el nivel de enlace del router debe ser también Ethernet. De no ser así, la comunicación entre PyC y su router sería imposible.

Por tanto, una vez que los datos llegan a la capa Ethernet del router, éste analiza la cabecera Ethernet del paquete.

Al analizar la cabecera, descubre que la dirección MAC de destino era la suya, por lo que decide continuar el procesamiento del paquete, ya que le corresponde a él hacerlo. Para poder continuar el procesamiento debe saber a qué capa pasarle la pelota. Para ello analiza la cabecera Ethernet del paquete y descubre la capa de encima tiene que ser la capa IP. Así, el router elimina la cabecera Ethernet del paquete (ya que la capa IP es incapaz de comprenderla), y pasa el resto del paquete a la capa IP.



El router analiza la cabecera Ethernet, comprobando que la MAC de destino es la suya, y pasa el resto del paquete (sin la cabecera Ethernet) a la capa IP.

4.7. En el router ADSL de PyC: La capa IP

Una vez que la capa IP recibe el paquete, analiza su cabecera. Ahí encuentra la dirección IP de destino, que es la de Scherzo. El router tiene configurada internamente una tabla parecida a esta:

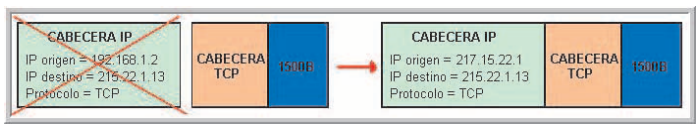
IP Routing Table

Type	Destination	Netmask	Gateway
Network	192.168.1.0	255.255.255.0	192.168.1.1
Network	217.15.22.64	255.255.255.0	217.15.22.1

Lo que vemos en esta tabla es que el router ADSL enviará al router de su ISP cualquier paquete que tenga como IP de destino una que no pertenezca a la LAN de PyC. Como la dirección IP de Scherzo no es una de las de la LAN de PyC, el paquete será enviado al router del ISP de PyC.

Una vez encontrado el siguiente mediador al que tiene que enviar el paquete, el trabajo del router ADSL termina aquí. No le interesa para nada lo que haya dentro del paquete, es decir, la cabecera TCP, y los datos del archivo que estamos enviando. Eso ya será problema del PC de Scherzo cuando reciba el paquete.

Lo único que tiene que hacer ahora el router ADSL es devolver el paquete a la capa de enlace para que ésta sepa qué hacer con él.



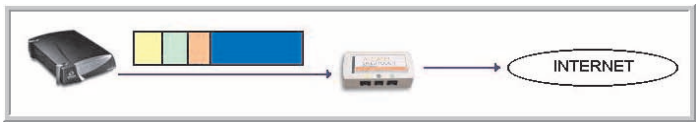
El router modifica la cabecera IP, sustituyendo la IP de LAN de PyC por la IP pública para que pueda salir ya al resto de la red Internet. Ya explicaremos esto mejor a lo largo del curso.

4.8. En el router adsl de PyC: De nuevo en la capa de enlace.

Pero... ¡Sorpresa! Con quien ha contactado la capa IP del router no es con la capa Ethernet, si no con otra capa de enlace diferente!!

Esto es debido a que la conexión entre el router ADSL y el router del ISP no es mediante un enlace Ethernet, si no mediante una tecnología de enlace ADSL.

No vamos a detallar aquí más capas de enlace, o buen barullo montaríamos, así que nos imaginaremos que esta capa añadirá sus propias direcciones y su identificador de la capa superior, y enviará el paquete a la capa física, que esta vez no será el cable Ethernet, si no el cable telefónico que va hacia el splitter.



El paquete ya puede salir a Internet, una vez reconstruidas las cabeceras IP y de enlace. La cabecera TCP y los datos del archivo (zona azul del paquete) se han mantenido intactas.

4.9. En el router del ISP de PyC

Una vez que el paquete llega al router del ISP, éste repetirá el proceso de recoger el paquete, analizar desde la capa de enlace si va dirigido a él, pasarlo a la capa IP, mirar su tabla de encaminamiento interna, y decidir a qué mediador enviar el paquete a continuación.

Imaginemos que ésta es una parte de la tabla de encaminamiento de este router:

IP Routing Table

Type	Destination	Netmask	Gateway
Network	215.0.0.0	255.0.0.0	215.12.133.2

En función de esta tabla, el router decidirá enviar el paquete a otro router, cuya IP es 215.12.133.2. Así, construirá una nueva cabecera de enlace, y reenviará el paquete por el cable físico adecuado.

4.10. Flotando en el ciberespacio

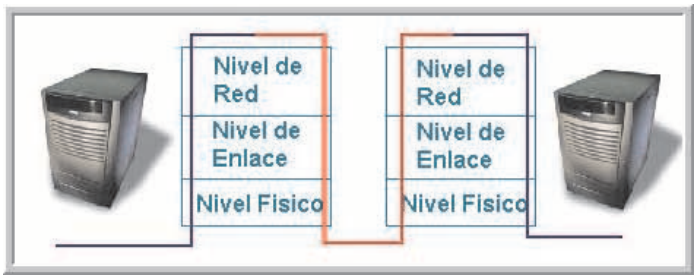
Este proceso de saltar de un router a otro buscando un camino que llegue hasta Scherzo puede constar de muchos pasos. Si queréis hacer la prueba vosotros mismos, tenéis un comando en vuestro sistema que os permite ver por qué routers van pasando vuestros paquetes al enviarlos a un destino concreto.

Ya os explicaré cómo funciona internamente este comando, pero de momento haced la prueba en una ventana MS-DOS (Ventana de Comandos), o una consola Linux:

tracert www.google.com

Cada línea es uno de los routers que forman el camino entre vosotros y la Web de Google.

En cada uno de estos routers se repetirá todo el proceso de analizar el paquete hasta la capa IP, e ir relanzándolo por los distintos cables que forman la auténtica telaraña de Internet.



Cada router por el que pase el paquete analizará sus cabeceras de enlace e IP, para ir retransmitiendo el paquete a cada punto que forma el camino entre el origen y el destino.

Cuando, a lo largo del curso, hable sobre la capa TCP, veremos que los paquetes no siguen siempre el mismo camino, e incluso pueden llegar desordenados a su destino, pero de momento no vamos a entrar en tanto detalle, que me falta ya espacio. ;-)

4.11. En el router ADSL de Scherzo

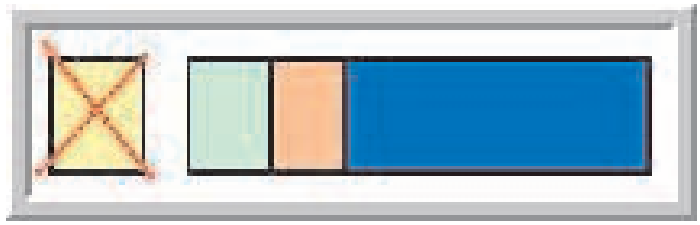
¡Al fin el paquete llegó hasta el último router del camino! Este, por supuesto, es el router ADSL de Scherzo. Éste analizará el paquete, y verá que en la capa IP tiene como dirección IP de destino la IP de Scherzo, así que ahora sólo le falta saber a cuál de los PCs de la LAN de Scherzo enviarlo.

Todos los PCs de la LAN de Scherzo comparten una misma IP de cara a Internet, que es la IP del router, y éste los diferencia sólo por su IP de LAN y por su dirección MAC. El cómo sabe el router a qué máquina enviar el paquete se sale ya del tema que puedo abarcar por hoy, así que nos creeremos que sabe dirigir el paquete a su destino dentro de la LAN.

Para ello, construye una nueva cabecera Ethernet (capa de enlace) que tiene como dirección MAC de destino la dirección MAC de Scherzo.

4.12. En el PC de Scherzo: la capa Ethernet.

Una vez que el paquete llega a Scherzo, su tarjeta de red empieza analizando la capa de enlace (Ethernet) y, al ver que la dirección MAC de destino es la suya, sabe que el paquete es para él. A continuación, lee el campo de la cabecera Ethernet que le dice que la siguiente capa que tiene que analizar el paquete es la capa IP, y le pasa la pelota a esta capa del sistema operativo.

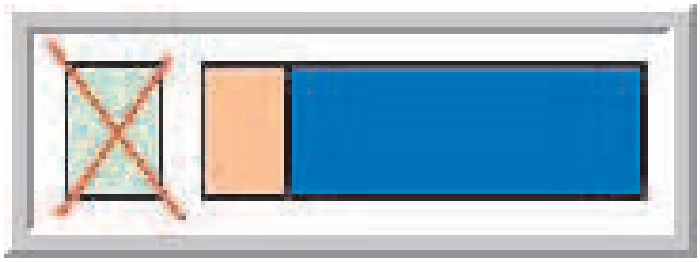


Una vez analizada la cabecera Ethernet, se elimina, y se pasa el resto del paquete a la capa IP.

4.13. En el sistema operativo de Scherzo: la capa IP

La capa IP de Scherzo analizará ahora no sólo la IP de destino (que es la de Scherzo), si no también la de origen (que es la de PyC), ya que tendrá que enviar sus respuestas a esa dirección. Una vez que se queda con estos dos datos, manda el paquete a la capa superior que, según la cabecera IP, es la capa TCP.

En todos los saltos que ha ido dando el paquete de un router a otro, ninguno ha analizado su cabecera TCP, ya que esto es sólo responsabilidad del destinatario final (Scherzo).



Una vez analizada la cabecera IP, se elimina, y se pasa el resto del paquete a la capa TCP.

4.14. En el sistema operativo de Scherzo: la capa TCP

La capa TCP de Scherzo cogerá el paquete, y verá que no es un paquete completo, si no sólo un trozo (recordemos que el archivo se partió en 3 trozos en la capa TCP de PyC).

La capa TCP de Scherzo tiene ahora dos responsabilidades: responder a PyC diciéndole que ha recibido el primer trozo, y mandar el paquete a la capa de arriba, es decir, a la aplicación cliente de FTP, que será la que sepa qué hacer con los datos contenidos en el paquete.

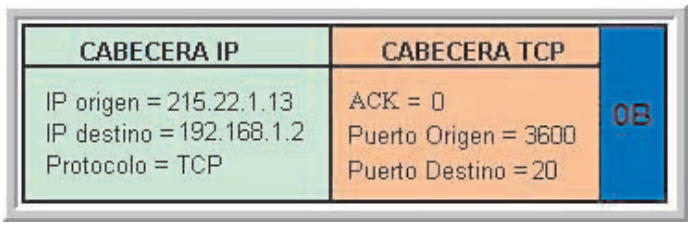
Con respecto al segundo punto, poco hay que decir. Simplemente, se eliminará la cabecera TCP y lo que quedará será un paquete de datos limpio y sin ninguna cabecera, que es lo que comprende la aplicación de PyC. La capa TCP esperará a tener el resto de trozos del archivo para poder reconstruirlo en el orden adecuado gracias a los números de secuencia.
Con respecto al primer punto, la capa TCP de Scherzo tiene que avisar a PyC de que ha recibido el primer trozo, así que comienza la construcción de un paquete totalmente nuevo.

Este paquete simplemente tendrá un aviso del tipo "oye, que te estoy escuchando, y he recibido ese paquete". Para ello, tiene una cabecera TCP especial que incluye, además del aviso de que está escuchando, un dato que es el último byte recibido del archivo. Como hemos recibido de momento sólo el primero de los tres trozos, habremos recibido hasta el byte 1500 del archivo. Por tanto, construimos una cabecera como esta:

El sistema de Scherzo construye un nuevo paquete para indicar a PyC que recibió el suyo. La cabecera TCP de este nuevo paquete constará de un campo ACK con el mismo valor que el número de secuencia del paquete al que quiere responder, e intercambiará los puertos de origen y de destino. El contenido del paquete (zona azul) estará vacío, ya que lo único importante aquí son las cabeceras.

4.15. En el sistema operativo de Scherzo: de vuelta en la capa IP

Esta cabecera la pasaremos a la capa IP, que conoce la IP de PyC, por lo que construye una cabecera IP adecuada para ese paquete:



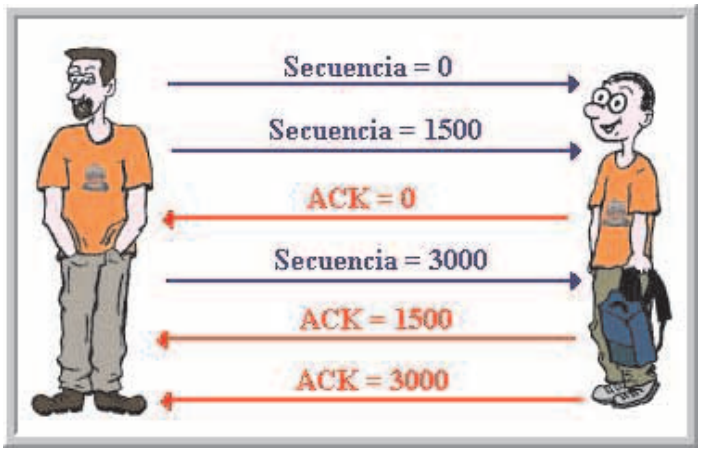
En la cabecera IP del nuevo paquete también se intercambian las IPs de origen y de destino.

4.16. ¡Vamos pa atrás!

Y así continúa el proceso, llevando el nuevo paquete de vuelta por el camino inverso al que siguió el paquete de PyC, hasta que el nuevo paquete llegue a PyC, éste analice su cabecera TCP, descubra que el primer paquete que envió llegó con éxito a su destino, y se despreocupe así ya al fin del paquete de marras.

Normalmente, no esperará a que Scherzo le vaya avisando de que recibe cada paquete, si no que enviará varios de golpe y, sólo si no recibe alguna de las respuestas, entonces reenviará aquel o aquellos de los que no ha recibido respuesta.

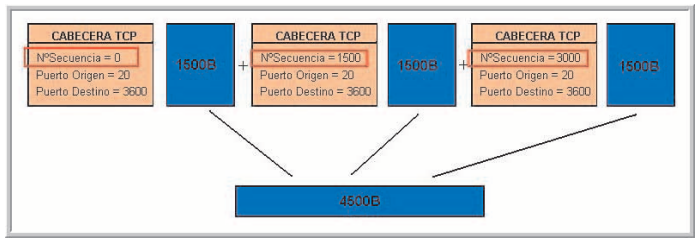
Por tanto, es probable que para cuando PyC haya recibido la primera respuesta de Scherzo, ya haya enviado también los otros dos paquetes que forman el archivo, ante los cuales también esperará la respuesta de Scherzo.



PyC envía los tres paquetes y espera un tiempo razonable a que le llegue la respuesta (ACK) de Scherzo diciendo que ha recibido cada uno de los paquetes

4.17. ¡Conseguido!

Una vez que Scherzo tiene ya los tres trozos del archivo, ya no tiene más que unirlos en el orden adecuado, el cual conoce gracias al número de secuencia de cada trozo, reconstruyendo así el archivo original, y mandándolo a la aplicación de FTP de Scherzo sin que ésta tenga ni idea de cómo ha llegado hasta allí ese archivo, confiando en que unos enanitos mágicos se habrán encargado del trabajo sucio. ;-)



Gracias al número de secuencia de cada paquete se puede reconstruir el archivo original, uniendo cada fragmento en el punto (posición en bytes) indicado por el número de secuencia.

Autor: PyC (Lco)

Ilustraciones: Adhara (Lco)