Tema 1: Redes de computadores e internet

1. ¿Qué es internet?

1.1 Descripción de los componentes esenciales

Hosts o sistemas terminales: dispositivos finales conectados a internet entre sí mediante una red de **enlaces de comunicaciones** y dispositivos de **conmutación de paquetes**.

Para acceder a la red, los hosts lo hacen a través de los **ISP**, que son proveedores de internet interconectados entre sí.

Los estándares de internet son desarrollados por el **IETF** (Internet Engineering Task Force) en documentos llamados **RFC**.

1.2 Descripción de los servicios

Una **aplicación distribuida** es aquella en la que varios host se intercambian datos.

1.3 ¿Qué es un protocolo?

Un **protocolo** define el formato y orden de los mensajes intercambiados entre dos o más entidades que se comunican, así como las acciones tomadas en la transmisión y/o recepción de un mensaje u otro suceso.

2. La frontera de la red

2.1 Programas cliente y servidor

Un programa **cliente** es aquel que se ejecuta en un host y solicita y recibe servicios de otro llamado **servidor** y que se ejecuta en otro host. En arquitecturas **P2P**, todos los host hacen funciones de **cliente** y **servidor**.

2.2 Redes de acceso

Acceso telefónico

El modem del usuario 'llama' al número de teléfono de un ISP y establece una conexión telefónica normal, transmitiendo datos por ella. Tiene las desventajas de ser muy lenta (56kb) y que no permite otras llamadas.

DSL

Utiliza también las líneas de teléfono, pero permite mayores velocidades y llamadas simultáneas mediante unos nodos DSLAM en las centrales de las compañías telefónicas.

Cable

Utiliza cable coaxial y permite mayores velocidades pico que DSL, pero es un medio de difusión compartido con los vecinos, y cuando muchos están conectados tiene velocidades más lentas.

FTTH

Utiliza fibra óptica y tiene las velocidades más altas.

Ethernet

Se usa normalmente en redes locales, y tiene velocidades altas (entre 100MBs y 10Gbps)

Wi-fi
Se usa para LANs inalámbricas.

Wi-Max
Es una mejora del Wi-Fi que permite comunicaciones rápidas y a larga distancia.

2.3 Medios físicos
Los medios físicos pueden ser guiados (la información va por un medio sólido, como un cable) o no guiados (por el aire o espacio).

Los más comunes son:

- **Cobre de par trenzado:** el medio más común y barato, puede llegar a 1Gbps
- **Cable coaxial**: rápido, utilizado por tele por cable, y es compartido.
- **Fibra óptica**: muy rápida, inmune a interferencias y pinchazos
- **Canales de radio terrestres:** pueden ser de área local (para redes locales) o de largo alcance (para móviles y tal)
- **Canales de radio vía satélite**: se emplean satélites geoestacionarios y de órbita baja terrestre para enviar los datos por el espacio.

3. El núcleo de la red

3.1 Conmutación de circuitos y conmutación de paquetes

Conmutación de circuitos
Se reservan los recursos necesarios para la comunicación entre dos host. Los canales se multiplexan por bandas de frecuencias (**FDM)** o por tiempo **(TDM)**.

Conmutación de paquetes
Los recursos se utilizan bajo petición y pueden tener que esperar. Decimos que la multiplexación es **estadística**.

3.2 Redes troncales de internet y proveedores ISP
En el nivel superior de la red tenemos a los **ISP de nivel 1**, que tienen altísimas velocidades de transmisión, están conectados todos con todos, proporcionan cobertura internacional, y están conectados a ISP de niveles inferiores.

Los **ISP de nivel 2** tienen cobertura regional o nacional, y depende de los de nivel 1 para llegar a todo el mundo. Por debajo de estos, puede haber ISP de niveles inferiores llamados **ISP de acceso** que proporcionan acceso a los clientes finales.

Los puntos entre los cuales se interconectan 2 ISP se llaman **Puntos de presencia**.

4. Retardos, pérdidas y tasa de transferencia en redes de conmutación de paquetes

4.1 Retardo en las redes de conmutación de paquetes
Llamamos **retardo nodal** a la suma de los 4 siguientes retardos:

Retardo de procesamiento

Es el tiempo requerido por los routers para examinar la cabecera del paquete y determinar dónde hay que enviarlo, así como comprobar errores y alguna otra tarea.

Retardo de cola

Es el tiempo que pasa un paquete en colas de salida de los routers mientras espera a ser transmitido.

Retardo de transmisión

Es el tiempo necesario para introducir todos los bits de un paquete por el enlace. Es igual a L/R donde R es la velocidad de transmisión del enlace y L el tamaño del archivo.

Retardo de propagación

Es el tiempo que tarda un bit, una vez está en el medio físico, en llegar hasta el final del enlace. Es igual a d/s donde s es la velocidad de propagación por el medio, y d la distancia.

4.2 Retardo de cola y pérdida de paquetes

Si llamamos *a* a la velocidad media a la que llegan paquetes a la cola de un router, la **intensidad de tráfico** en dicho router será La/R. Si La/R es mayor que la velocidad a la que pueden transmitirse paquetes hacia fuera (L/R generalmente), la cola aumentará y el retardo tenderá a infinito. Ahora bien, puede haber colas aunque la intensidad sea menor en media, siempre que coincidan muchos paquetes de golpe.

Se produce una pérdida de paquetes cuando llegan paquetes a un router con la cola llena.

4.3 Tasa de transferencia en las redes de computadoras

Llamamos tasa de transferencia de un archivo a F/T, donde F es el número de bits del archivo y T el tiempo que tarda en recibir el destino el archivo. Generalmente esta tasa se puede aproximar como F/min(Ri) donde Ri son las tasas de transferencia de los enlaces por el camino, ya que estos suelen ser el cuello de botella.

5. Capas de protocolos y sus modelos de servicio

5.1 Arquitectura en capas

La red está estructurada en capas. Cada capa ofrece un **modelo de servicio** a las que tiene por encima. Esta modularidad facilita el mantenimiento y surgimiento de nuevos protocolos en cada capa, pero a cambio provoca redundancias en los modelos de servicio (por ejemplo, se realizan sumas de comprobación de bits en varias capas) y alguna otra inconveniencia.

La **pila de capas o protocolos** de internet se compone de 5 capas:

Capa de aplicación

Aquí residen las aplicaciones de red y sus protocolos, como HTTP, SMTP, FTP o DNS. A la unidad de información enviada en dicha capa la llamamos **mensaje**.

Capa de transporte

Transporta los mensajes de aplicación a aplicación. Tiene dos protocolos, TCP y UDP, y la unidad mínima de información que envía la llamamos **segmento**.

Capa de red

Incluye el protocolo IP (entre otros) y facilita el envío de **datagramas** entre los distintos routers de la red.

Capa de enlace

Es la capa encargada de enviar la información entre routers consecutivos. Denominamos a sus paquetes **tramas**.

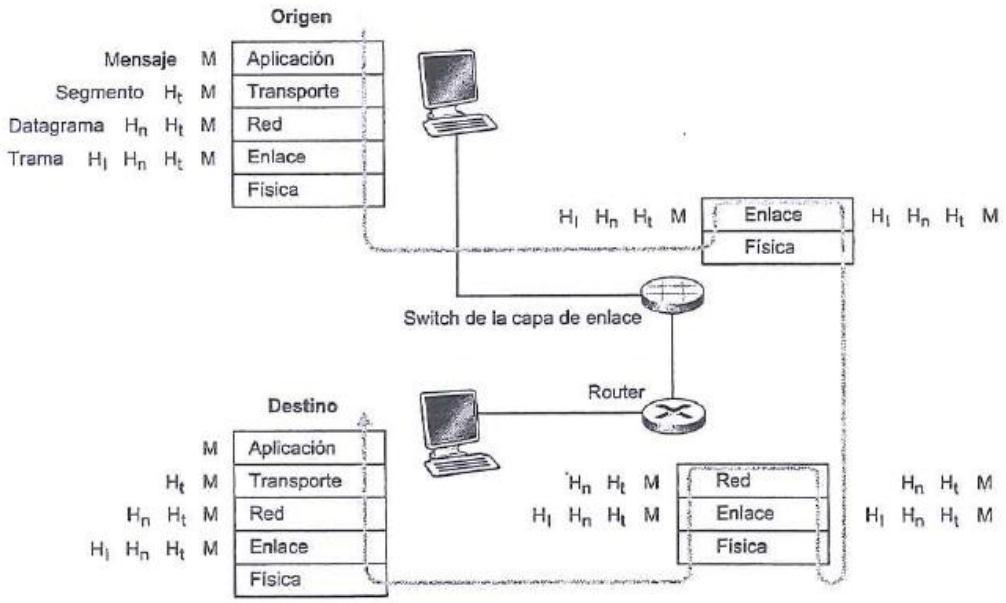
Capa física

Es la encargada de codificar bits en medios físicos mediante ondas, pulsos, etc.

Modelo OSI

Es un modelo deprecado que incluía, amén de estas cinco capas, una capa llamada **presentación** (que se encargaba de dar interpretación a los datos enviados) y otra de **sesión**, para sincronizar el intercambio de datos. Estas funcionalidades se incluyen ahora en la capa de aplicación, cuando fuere necesario.

5.2 Mensajes, segmentos, datagramas y tramas



Tema 2: La capa de aplicación

1. Principios de las aplicaciones de red

1.1 Arquitecturas de las aplicaciones de red

En la arquitectura **cliente-servidor** tenemos un host activo, **servidor**, que espera peticiones de los clientes.

En la arquitectura **P2P** los hosts (***peers***) se conectan por parejas de forma intermitente, actuando tanto de cliente como de servidor. Este tipo de arquitecturas se enfrentan a tres "retos": que los ISP residenciales ofrecen conexiones **asimétricas** (mucha más bajada que subida); mantener la **seguridad**, tarea difícil por su naturaleza distribuida y abierta, y dar **incentivos** a los usuarios para colaborar.

1.2 Procesos de comunicación

Llamamos **socket** a una interfaz (API) entre las aplicaciones de usuario y la red.

1.3 Servicios de transporte disponibles para las aplicaciones

Podemos clasificarlos en 4 grupos:

- **Transferencia de datos fiable:** que los datos se entreguen de forma íntegra (sin pérdidas ni corrupciones), en orden y sin duplicados.
- **Tasa de transferencia**: Que den una tasa de transferencia garantizada. Las apps **elásticas** son aquellas que se apañan con cualquier tasa.
- **Temporización**: que ofrezcan una latencia máxima garantizada.
- **Seguridad**: cifrar los datos transferidos

1.4 Servicios de transporte proporcionados por internet

TCP

- **Orientado a la conexión:** el cliente y servidor intercambian información de control de la capa de transporte antes de empezar a fluir los mensajes de la aplicación, para que ambos reserven recursos y se preparen. Dicha conexión es **full-duplex** porque ambos pueden enviar y recibir datos por la misma.
- **Fiable**
- **Controla la congestión:** limita cada conexión para que use una cuota equitativa.

UDP

Es un protocolo ligero y simple que no ofrece ninguno de los servicios. Es más rápido que TCP, pero en ocasiones lo bloquean los cortafuegos, y por ello tampoco se usa mucho.

2. DNS: Servicio de directorio de internet

2.1 Servicios proporcionados por DNS

DNS (*Domain name system)* es un protocolo de la capa de aplicación que se utiliza para convertir direcciones en formato legible por lo humanos en direcciones IP. DNS se ejecuta con UDP en el puerto 53. Proporciona, además de este servicio de traducción, los siguientes:

- **Alias de host**: traducción de direcciones sencillas (llamadas alias) a una dirección compleja (nombre canónico). Por ej., dkjfh.ej.caca.com -> alias: caca.com

- **Alias del servidor de correo:** ídem pero para servidores de correo
- **Distribución de carga:** se pueden asignar un conjunto de direcciones IP a un mismo nombre canónico de host. De esta manera, cuando alguien hace una petición DNS, cada vez obtendría una IP distinta, y así se distribuiría la carga de forma más eficiente.

2.2 Cómo funciona DNS

DNS podría realizarse mediante un host central en la red que tuviera todas las traducciones, pero esto tendría desventajas como dificultad de mantenimiento, tráfico excesivo o que sea un único punto de fallo. Por ello, en su lugar se usa una base de datos jerárquica y distribuida.

Tenemos tres tipos de servidores DNS:

- **Raíz:** son únicamente 13 a nivel global
- **TLD (dominio de nivel superior):** responsables de los .com, .es, …
- **Autoritativos:** están en todas las compañías con hosts accesibles públicamente

Cuando queremos hacer una consulta DNS, generalmente primero preguntamos a nuestro servidor de DNS configurado en el PC (que puede ser uno de la empresa o la uni, o del ISP). Después este se encarga por sí mismo de preguntar a un servidor raíz, el cual le refiere a un TLD, el cual le refiere a un autoritativo, el cual finalmente le da la traducción. Decimos que la consulta que hacemos al servidor primero de DNS es **recursiva** (le pasamos a él el trabajo) y las que él hace luego son **iterativas** (ya que no les va pasando el trabajo a los 3 servidores que consulta, solo les hace una pregunta).

Generalmente, para agilizar el proceso hay cachés de DNS intermedias.

2.3 Registros y mensajes DNS

Los registros DNS siguen el formato **(Nombre, Valor, Tipo, Tiempo de Vida** (para ser borrado de una caché) **).**

- Si **Tipo = A: Nombre** es un nombre de host y **Valor** la IP correspondiente
- Si **Tipo = NS: Nombre** es un dominio y **Valor** es la dirección del servidor autoritativo que sabrá traducir dicho dominio a IP.
- Si **Tipo = CNAME: Nombre** es un alias para el nombre canónico dado como **Valor**
- Si **Tipo = MX:** Es igual que CNAME pero para alias de correo.

De esto se desprende que los servidores autoritativos para un host tendrán sus registros de tipo A, mientras que todos los demás como mucho tendrán registros de tipo NS, así como registros de tipo A para poder llegar a los servidores autoritativos de dicho host.

3. Programación de *sockets* con TCP

Es importante saber que: el programa de servidor debe estar siempre escuchando con un socket de 'welcome', y que una vez recibe una petición de un cliente, crea para el mismo un socket **de conexión** único entre el cual se establece un servicio fiable de **flujo de bytes**.

4. Programación de *sockets* con UDP

En este caso es importante saber que no hay flujos (streams) de datos, ni creación de la conexión inicial. Lo único que se hace es enviar paquetes de uno en uno configurados todos con la ip y puerto del socket de destino.

Tema 3: La capa de transporte

1. La capa de transporte y sus servicios

Un protocolo de la capa de transporte proporciona una comunicación lógica entre procesos que se ejecutan en host distintos. Desde la perspectiva de la aplicación, es como si los host estuvieran conectados directamente.

1.1 La capa de transporte en internet

En internet, tanto UDP como TCP proporcionan los servicios de **multiplexación y demultiplexación** de la capa de transporte, así como la comprobación de la integridad de los datos mediante un campo de checksum en las cabeceras. Estos dos servicios son los únicos que proporciona UDP.

2. Multiplexación y demultiplexación

Llamamos **demultiplexación** en el host destino a entregar los datos en un segmento de la capa de transporte al socket adecuado. Llamamos **multiplexación** a recopilar todos los fragmentos en el host origen, encapsularlos con info de cabecera, y pasarlos a la capa de red.

Para multiplexar y demultiplexar, asignamos a cada socket un número de puerto de 16 bits. Los números 0-1023 se denominan **puertos bien conocidos** y están estandarizados y restringidos.

Sin conexión (UDP)

Para la demultiplexación, un socket queda identificado por la IP y puerto de **destino** del segmento. Dos segmentos de distintos host que se envíen a la misma IP y puerto acabarán en el mismo socket.

Con conexión (TCP)

En este caso, y como veíamos en el tema 2 sección 4ª, tenemos sockets de 'bienvenida', que se podrán identificar, como en UDP, solo con la IP y puerto de destino, y luego estos crean sockets de conexión para cada cliente. Con ello, tenemos que para un mismo puerto de destino en el host pueda haber un gran número de conexiones con clientes. Por ello, para poder demultiplexar a sockets de conexión, se utilizan las IP y puertos de **origen** y **destino**.

3. Transporte sin conexión: UDP

Ya hemos visto en el 2.1.4 y 3.1.1 los servicios que ofrece. ¿Por qué se usa?

- Mejor control sobre qué datos se envían y cuándo.
- Ahorro del retardo por establecimiento de conexión.
- Al no haber info de estado de conexión se soportan más clientes simultáneos
- Menor sobrecarga por reducido tamaño de cabecera (8B frente a 20B en TCP)

3.1 Estructura de los segmentos UDP

3.2 Suma de comprobación de UDP

1. El emisor calcula la suma de todas las palabras de 16 bit del segmento (menos la propia suma de comprobación, ya que la estamos calculando).
    a. Si hubiera desbordamientos, los sumamos por la derecha.
2. En el receptor se suman todas las palabras de 16 bit, incluyendo la suma, y debería dar todo 1's.

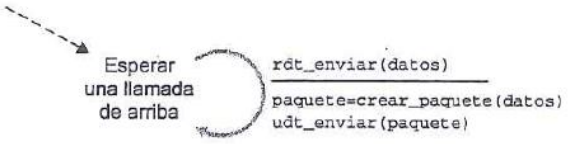
Es necesario recalcar que aunque se permita este mecanismo de comprobación de errores, no hay mecanismo de recuperación de los mismos en UDP (de ahí que no sea fiable).

4. Principios de un servicio de transferencia de datos fiable

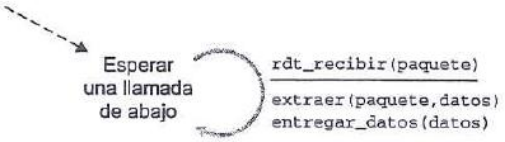
- `rdt_enviar()`: interfaz a la que llama la capa de aplicación del emisor para enviar datos de manera fiable
- `rdt_recibir()`: interfaz que será llamada en el receptor al llegar datos de manera fiable
- `entregar_datos()`: interfaz mediante la cual la capa de transporte del receptor envía los datos a la capa superior.
- `udt_enviar()`: interfaz para el envío de datos por el canal no seguro.

4.1 Construcción de un protocolo de transferencia de datos fiable

Rdt1.0 - Canal fiable

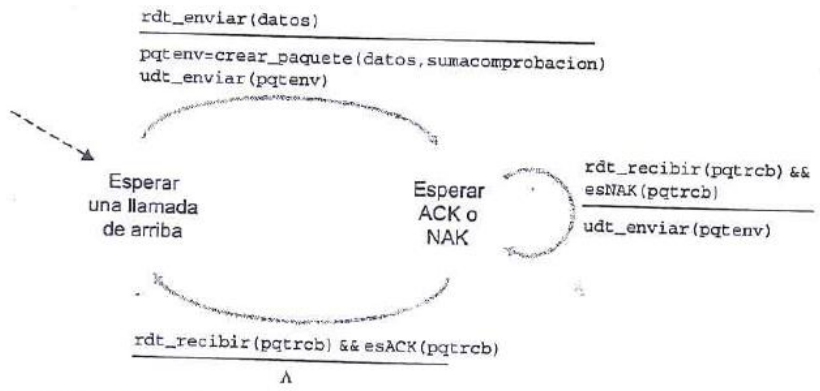


a. rdt1.0: lado emisor

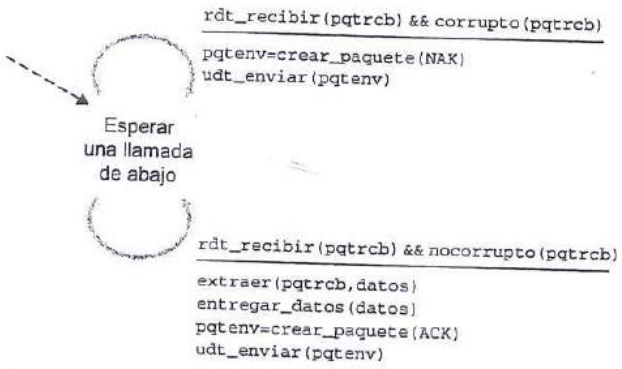


b. rdt1.0: lado receptor

Rdt2.0 – Canal con errores de bit



a. rdt2.0: lado emisor



b. rdt2.0: lado receptor

Este tiene un error fatal, ACK o NAK puede estar corrupto.

Rdt2.1 – Como el anterior pero corregido

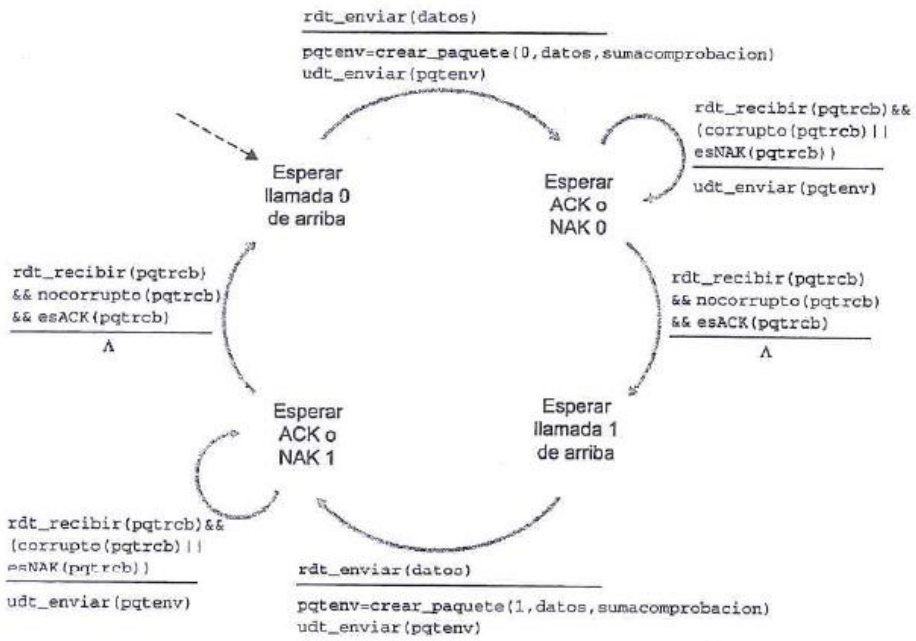


Figura 3.11 • Lado emisor de `rdt2.1`.

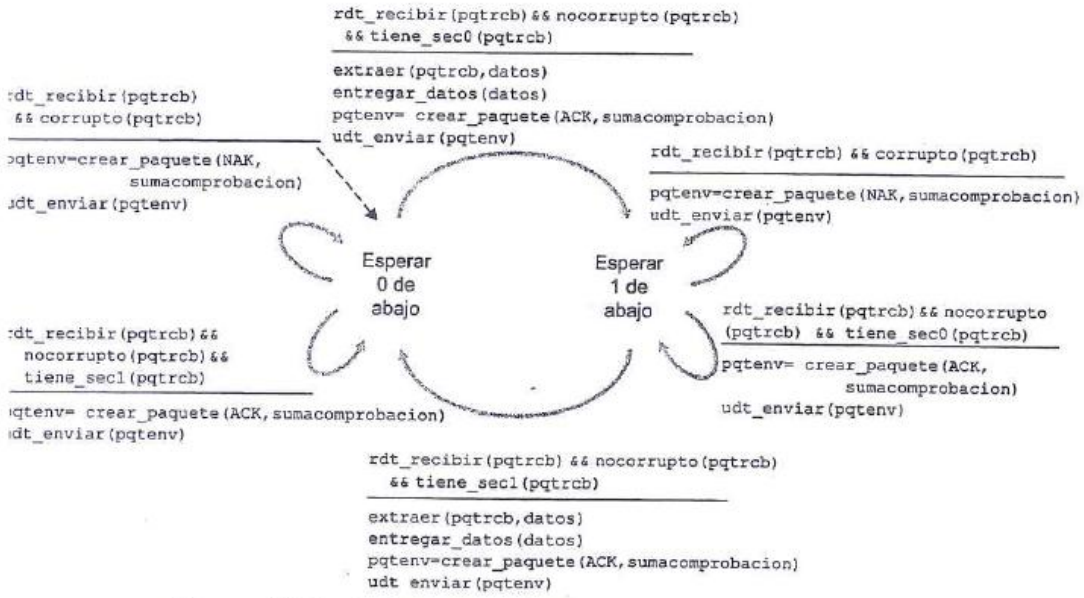


Figura 3.12 • Lado receptor de `rdt2.1`.

Rdt2.2 – Como el anterior pero ahorrándonos los NAK

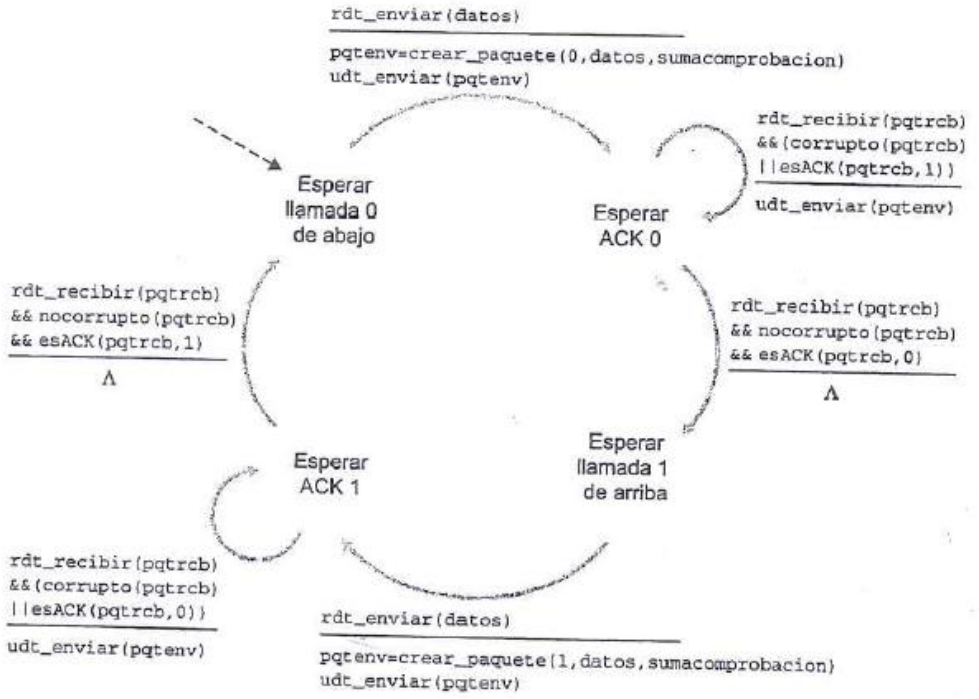


Figura 3.13 • Lado emisor de `rdt2.2`.

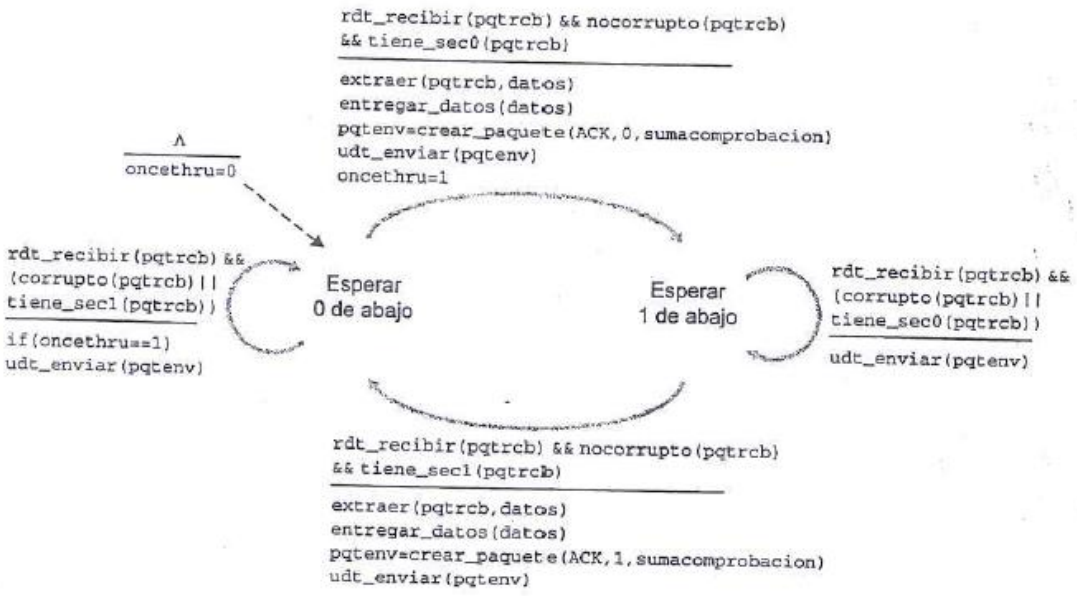


Figura 3.14 • Lado receptor de `rdt2.2`.

Rdt3.0 – Canal con errores de bit y pérdidas

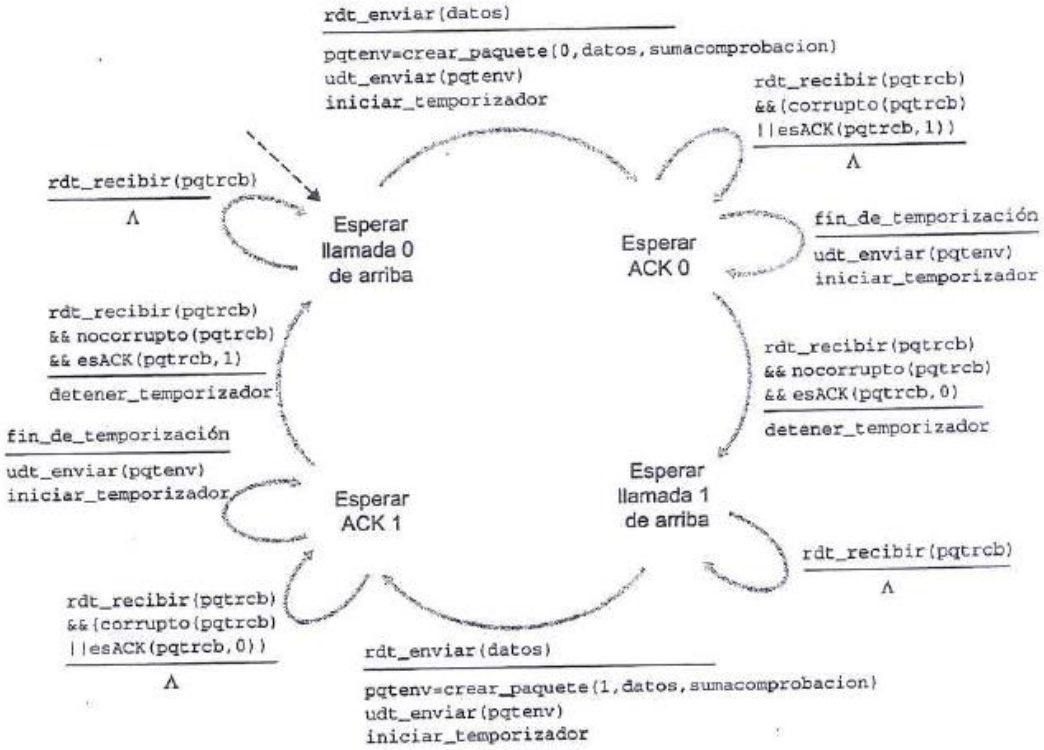


Figura 3.15 • Lado emisor de `rdt3.0`.

El receptor sería idéntico al visto en Rdt2.2

4.2 Protocolo de transferencia de datos fiable con procesamiento en cadena

En el Rdt3.0, se puede perder mucho tiempo esperando a recibir confirmaciones de cada segmento enviado. Por ello, en la realidad se utiliza el *pipelining* para enviar varios paquetes sin tener que esperar a que los anteriores hayan sido reconocidos. Para ello, utilizaremos buffers de recepción de paquetes en el receptor y números de secuencia más grandes que 0 o 1

4.3 Retroceder N (*Go-Back-N*)

En este protocolo se pueden tener como máximo N paquetes no reconocidos a la vez en una transmisión.

Cuando el emisor envía un paquete, inicia un temporizador para el mismo. Si el temporizador de un paquete se agota antes de que llegue su ACK, se reenvían todos los que había en la ventana en ese momento.

Cuando recibe un paquete en orden, el receptor envía un ACK del mismo al emisor. Cuando recibe uno en desorden, lo descarta y envía un ACK para el último recibido en orden.

Cuando el emisor recibe una confirmación, si esta es sobre alguno de los paquetes que tenía en aún sin confirmar, bien, los reconoce todos hasta él, y vuelve a transmitir paquetes hasta que haya N ‘en el aire’.

4.4 Repetición selectiva

En este caso los reconocimientos no son acumulativos, y cuando se producen sucesos de fin de temporización, solo se reenvía el paquete perdido.

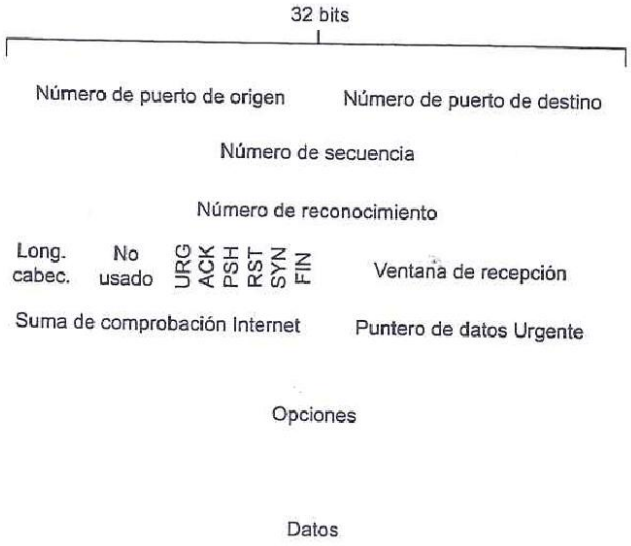
5. Transporte orientado a la conexión: TCP

5.1 La conexión TCP

La conexión TCP es **punto** a **punto**, es decir, no puede usarse para multidifusión.

Primero, mediante un *handshake* en **3 fases** se establece la conexión; segundo, el cliente pasa un flujo de datos a la capa de TCP y este se almacena en un **buffer de emisión**. Después estos datos se van colocando en segmentos (cuyo tamaño máximo denominamos **MSS,** y dentro del cual no se tienen en cuenta tamaño de cabeceras TCP ni IP), se pasan a la capa de red, y se envían. Finalmente en el destino, se van almacenando los datos en un **buffer de recepción**, y la aplicación correspondiente los va consumiendo.

5.2 Estructura del segmento TCP



El **número de secuencia** de un segmento es el número de byte al que corresponde el primer byte de su campo de datos dentro del flujo global enviado. El **número de reconocimiento** es el primer byte que se espera recibir. TCP proporciona reconocimientos acumulativos.

La **longitud de cabecera** cuenta palabras de 32 bits.

La **ventana de recepción** se verá en control de flujo. Los demás campos no son importantes.

5.3 Estimación del tiempo de ida y vuelta y fin de temporización

Para estimar el tiempo medio exponencialmente ponderado de ida y vuelta de datos en un momento dado, usamos:

$$RTTEstimado' = (1 - \alpha) \cdot RTTEstimado + \alpha \cdot RTTMuestra$$

Donde $RTTMuestra$ es el tiempo que transcurre entre que se envía un segmento y se recibe su reconocimiento, y nunca se calcula sobre segmentos retransmitidos; α es un valor entre 0 y 1 (recomendado: 0.125).

Para estimar una desviación típica del mismo usamos:

$$RTTDesv = (1 - \beta) \cdot RTTDesv + \beta \cdot |RTTMuestra - RTTEstimado|$$

Valor recomendado de β: 0.25.

Finalmente, utilizamos lo anterior para calcular un intervalo de tiempo para fin de temporización:

$$ITFT = RTTEstimado + 4 \cdot RTTDesv$$

5.4 Transferencia de datos fiable

TCP mantiene un único temporizador para estimar si se han perdido paquetes. Dicho temporizador se inicia –si no estuviese ya iniciado- al enviar un segmento. Si posteriormente se recibe un segmento que reconozca aquel con el que se inició el temporizador, pues reiniciamos el temporizador (si quedase algún segmento sin reconocer), y si no, cuando se agote, retransmitiremos el paquete con el que lo iniciamos, y reiniciamos el temporizador.

Duplicación del tiempo

Existe una modificación por la cual cuando se agota un temporizador habiéndose perdido el paquete, tomamos el intervalo de fin de temporización como dos veces el anterior.

Retransmisión rápida

Si observamos que recibimos paquetes con el mismo número de reconocimiento, es que hay unos paquetes que han adelantado a otro (cuyo número de secuencia será igual al de reconocimiento que recibimos). Si se reciben tres ACK duplicados (esto es, 4 iguales), no esperamos a que caduque el temporizador sino que reenviamos ya el paquete presuntamente perdido.

5.5 Control de flujo

Consiste en adaptar la velocidad de transmisión del emisor para no desbordar el buffer del receptor. Para ello, el receptor devuelve en los segmentos de reconocimiento el espacio libre que le queda en el buffer, a través del campo **ventana de recepción**, de la cabecera TCP.

Puede darse el caso de que el buffer se llene del todo, y por tanto el emisor deje de enviar paquetes, y con ello el receptor deje de enviar ACKs de paquetes recibidos. En ese caso, para forzar a que se envíe la ventana de recepción, cada cierto tiempo el receptor envía segmentos con un byte de datos y dicho valor de la ventana en la cabecer.

5.6 Gestión de conexión TCP

Establecimiento de la conexión

1. **SYN:** el cliente manda un paquete sin datos, con el bit SYN activo y un número de secuencia aleatorio.
2. **SYN-ACK**: el servidor recibe el SYN y asigna buffers y variables TCP. Después envía otro mensaje sin datos al cliente, con el bit SYN activo, número de reconocimiento del segmento anterior, y número de secuencia aleatorio.
3. **ACK:** el cliente asigna sus recursos TCP y confirma el SYN anterior. Además, puede aprovechar para enviar datos.

Cierre de la conexión

En este caso simplemente, primero uno de los dos host manda un mensaje con un bit FIN, y el otro lo reconoce. Después este segundo hace lo mismo. Una vez ambas cosas se han realizado, quedan liberados todos los recursos.

6. Principios del control de congestión

6.1 Las causas y costes de la congestión

Mirar si eso en el libro. Son solo ejemplos…

7. Mecanismo de control de congestión de TCP

En TCP, los emisores bajarán su velocidad cuando perciban que existe congestión en la red. En concreto, mantendrán el número de paquetes enviados y no reconocidos por debajo siempre del mínimo entre **VentanaRecepcion** (como vimos ya en control de flujo) y **VentanaCongestion** (nueva variable que usamos para representar la congestión).

La **VentanaCongestion** es una variable que va controlando cada emisor TCP, y que no se incluye en los paquetes enviados. Irá aumentando cuando veamos que no hay congestión, y disminuirá si hay congestión (la cual se detecta o por 3ack o por fin de temporización).

Hay tres fases distintas de crecimiento o decrecimiento de esta variable:

Arranque lento

Al iniciar una conexión TCP, el valor de la ventana de congestión se inicializa normalmente a 1 MSS, y se va incrementando en 1MSS más cada vez que recibamos una confirmación de paquete. Con ello, la velocidad va aumentando exponencialmente, ya que tendremos, en el primer instante de tiempo 1MSS, en el segundo instante (cuando llegue el paquete reconocido, esto es, RTT), 2MSS, en el tercer instante (llegarán dos paquetes) aumentamos a 4MSS, etc.

- Si se agota el **temporizador** de un paquete no reconocido, damos valor a otra variable de TCP llamada **umbral** = **VentanaCongestion/2**
- Si nos llegan 3 ACKS duplicados, retransmitimos y pasamos a la fase de recuperación rápida
- Si el valor de **VentanaCongestion** es igual a **Umbral**, pasamos a evitación de la congestión

Evitación de la congestión

En esta fase aumentamos el valor de VentanaCongestion en 1MSS cada periodo de tiempo RTT.

- Cuando hay un **fin de temporizador**, hacemos lo mismo que en el caso anterior y pasamos a arranque lento
- Cuando llegan 3 ACK duplicados, hacemos lo mismo que en arranque lento

Recuperación rápida

La recuperación rápida en realidad solo se realiza en el protocolo TCP Reno. Sin embargo ya explicaré, como si parte de esta fase se tratase, lo que hace otro protocolo, TCP Tahoe.

- **Tahoe**: Pone **Umbral** = **VentanaCongestion/2**, **VentanaCongestion=1MSS**, y pasamos a arranque lento
- **Reno:** Pone **Umbral = VentanaCongestion/2**, **VentanaCongestion=Umbral** y pasamos a evitación de la congestión.

Tema 4: La capa de red

1. Introducción

1.1 Reenvío y enrutamiento

Llamamos **reenvío** a pasar un paquete de un enlace de entrada a uno de salida, dentro de un router. Llamamos **enrutamiento** a determinar la ruta o camino óptimo entre routers para ir desde el origen al destino. Los routers tienen una llamada **tabla de reenvío** que es utilizan para la función con el mismo nombre, y que se configura mediante **protocolos de enrutamiento** para así conseguir también llevar a cabo la segunda función.

Los **routers** son mecanismos de **conmutación de paquetes,** al igual que los **switches**. La diferencia es que los primeros se basan en el valor de la capa de red, y los segundos en el de la de enlace.

1.2 Modelos de servicio de red

Posibles servicios a nivel de paquete enviado:

- Entrega garantizada
- Retardo limitado

A nivel de flujo de paquetes:

- Entrega en orden
- Ancho de banda mínimo garantizado
- Fluctuación máxima garantizada
- Seguridad

Ahora bien, la capa de red de internet, **IP**, no ofrece ninguno (es *best effort*).

2. Redes de circuitos virtuales y datagramas

Las redes de **circuitos** proporcionan un servicio de conexión previa y las de **datagramas** no.

2.1 Redes de circuitos virtuales

Hay diversas arquitecturas (como ATM y Frame Relay) de redes de circuitos, pero en ellas, los circuitos siempre constan de una **ruta** entre host y destino, **números de VC** para cada enlace y **entradas para la tabla de reenvío** correspondientes en cada router.

Existen tres fases en el funcionamiento de un circuito:

1. **Configuración del VC**: La capa de transporte contacta con la de red y le indica la dirección de destino. La capa de red determina la ruta y asigna números de VC para todos los enlaces del camino. Se añaden entradas a la tabla de reenvío de todos los router del camino con los números de VC recién creados.
2. **Transferencia de datos**
3. **Terminación del VC:** se borran las entradas de la ruta de las tablas de reenvío.

2.2 Redes de datagramas

En estas redes cada vez que se desea enviar un paquete, se marca en este la dirección de destino y luego los routers de la red se encargan de enviarlo. Para ello, utilizan tablas de reenvío que, según rangos de IP o coincidencia con el prefijo más largo, miran a qué interfaz debe ser enviado el paquete.

3. El interior de un router

Un router cuenta con cuatro componentes: puertos de entrada (que se suelen agrupar en tarjetas de línea), puertos de salida, entramado de comunicación entre los de entrada y los de salida y procesador de enrutamiento, que ejecuta los protocolos de enrutamiento.

3.1 Puertos de entrada

Son los que conectan a la capa física y de enlace con la de red del router. En muchos casos, para ahorrar tiempo eligiendo el puerto de salida, cada puerto de entrada suele guardar una copia de las tablas de reenvío para evitar cuellos de botella.

Para encontrar rápidamente en esa tabla la interfaz de salida correspondiente a un paquete, para que no se produzcan colas, se usan técnicas como **búsqueda en árbol, memorias direccionables por contenido** (algo así como hashmaps) o **cachés** de últimas direcciones usadas.

Una vez sabemos el puerto de salida correspondiente, lo enviamos por el entramado de comunicación hasta el mismo, salvo si dicho entramado está ocupado. En ese caso, el paquete se queda en una cola en su puerto de entrada.

3.2 Entramado de conmutación

Como ya hemos visto, es por donde pasan los paquetes desde los puertos de entrada a los de salida. Puede realizarse con diversas tecnologías:

- **Vía memoria:** se copia el paquete del puerto de entrada a una memoria intermedia, y luego de esta al puerto de salida
- **Vía bus:** se transfieren por un bus (en el que solo puede ir un paquete a la vez) y es más rápido que lo anterior.
- **Vía red de interconexión:** se usa un *crossbar switch* con 2n buses (donde n: número de puertos de entrada = número puertos salida) que permite enviar hasta n paquetes a la vez, sin que haya repetidos puertos de entrada o salida.

3.3 Puertos de salida

Encolan los paquetes que les llegan por el entramado y los transmiten a la capa de red. Tienen los llamados **planificadores de paquetes** que les ayudan a determinar qué paquete transmiten primero al enlace. Pueden transmitir el primero en llegar (FCFS) o usar otros métodos como colas ponderadas equitativas (WFQ).

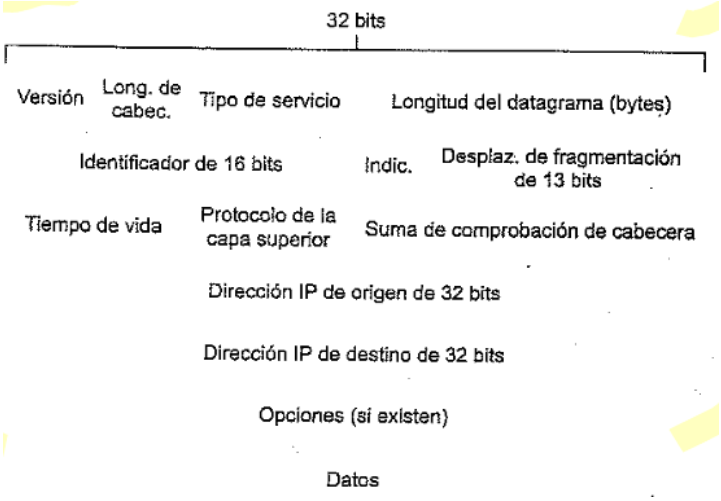
3.4 ¿Dónde se crean colas?

Se pueden crear colas en puertos de entrada y de salida. Si una cola supera el tamaño del buffer, se pierden paquetes. Sea v la velocidad del entramado de comunicación, l la velocidad de entrada de paquetes y n el número de puertos de entrada (igual al núm de puertos de salida, para simplificar), tenemos que:

- Para asegurar que no haya colas en los puertos de entrada, $v \geq l \cdot n$
- Como todos los n paquetes que entran a la vez pueden ir al mismo enlace de salida, para que no haya colas en el mismo la velocidad de salida deberá ser $s \geq v \cdot n = l \cdot n^2$
- Un caso especial de bloqueo es el **HOL**. Si usamos una red de interconexión en el entramado, podemos enviar simultáneamente n paquetes, pero si todos van a puertos distintos. Ahora bien, si un puerto de entrada tiene un paquete destinado a un puerto de salida ocupado, este se bloquea. El problema es que este paquete *cabeza de línea* bloquea a todos los que vienen tras él, que es el llamado bloqueo **HOL**.

4. Protocolo de Internet (IP): reenvío y direccionamiento en Internet

4.1 Formato de los datagramas



- La **longitud de cabecera** cuenta grupos de 32 bits
- El **tipo de servicio** hace referencia a cosas como 'urgente', 'entrega fiable'.
- La segunda fila es para **fragmentación**.
- La **suma de cabecera** es como la vista en el tema anterior.

Fragmentación

La cantidad máxima de datos que se puede transmitir en un enlace se conoce como **MTU**. Esta impone un límite a los datagramas IP, y por tanto a veces toca fragmentarlos.

Cuando se crea un datagrama, se identifica con el **id de 16 bit** de la cabecera. Al llegar a un enlace por el que no cabe, el router de turno lo fragmenta en datagramas más pequeños, y estos se rellenan con los mismos campos que el original, salvo el bit **indicador**, que se pone a 1 en todos los fragmentos salvo el último, y el **desplaz. De fragm.,** que indica, en múltiplos de 8 bytes, a qué byte corresponde el primer dato del fragmento actual, con respecto al flujo total. Este desplazamiento de 8 en 8 bytes restringe los tamaños de parte de datos de los datagramas a múltiplos de 8 bytes.

4.2 Direccionamiento IPV4

Cada interfaz de host y router tiene su propia dirección IP de 32 bit. Cuando especificamos una dirección de una **subred** (red de computadores contenida en internet), lo hacemos con una dirección IP y una máscara de subred, que indica cuántos bits de la IP determinan la dirección de subred.

$X.X.X.X/M$ -> Los M primeros bits de la IP definen la subred, el resto, los host internos.

La estrategia de asignación de IPs en internet se llama **CIDR**, y está definida para ser jerárquica, y el organismo que las gestiona se denomina ICANN.

Hay dos IP siempre reservadas en cada subred: aquella con todo 1's (dirección de difusión) y otra con todo 0's (dirección de subred).

DCHP

Normalmente, las IP de los routers se suelen configurar **manualmente** por los administradores, sin embargo, se suele dejar la configuración de los hosts a DHCP, que lo hace automáticamente.

DCHP es un protocolo cliente servidor que asigna a los hosts direcciones IP dentro de una subred. Dichas direcciones pueden ser la misma siempre que el host se conecte, o variables. Además, también proporciona información adicional a los host, tal como la máscara de subred, puerta de enlace y servidor DNS primario. Es un proceso de cuatro pasos:

1. **Descubrimiento del servidor:** el nuevo host envía un paquete UDP al puerto 67 con IP de origen todo 0s y de destino todo 1s.
2. **Ofertas del servidor**: el o los servidores envían varias ofertas de IPs de nuevo a la dirección de destino todo 1s y puerto 68.
3. **Solicitud DHCP**: el cliente elige una de las ofertas, enviando un paquete con origen todo 0s y destino todo 1s con la oferta dada.
4. **ACK DHCP**: el servidor contesta con un ACK a ese paquete.

NAT

Como las IPv4 son pocas en comparación con el número de hosts que hay en la actualidad, y como los números de puerto sobran, se ha inventado un truquillo para aprovechar los puertos para sacar más direcciones: **NAT**.

Consiste en una tabla en el router que hace una correspondencia entre pares de (Ip interna de un host de la red, puerto) con pares de (Ip externa de la red, puerto). Si no se entiende mirar ejemplo, página 341 del libro.

Muchos puristas de la **IETF** se quejan de NAT porque obliga a los routers a tocar la capa de transporte, y usa los puertos para lo que no son. Además, dificulta el P2P ya que las IP de los hosts son variables. Para usar programas P2P ha de haber un servidor centralizado fuera de NAT (y con IP fija y conocida por los usuarios del programa) que almacene las IPS de los *peers*.

4.3 Protocolo de mensajes de control de internet (ICMP)

ICMP es un protocolo que no se clasifica bien ni como protocolo de red ni como de transporte. Por un lado, tiene que ver con la red, pero por otro, se ejecuta por encima de IP. Se utiliza para intercambiar información de la capa de red, como errores, pings, etc.

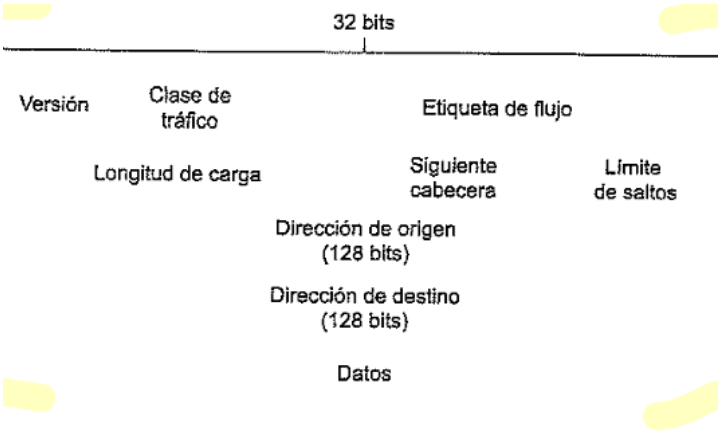
Los mensajes ICMP contienen un campo tipo, otro código, y la cabecera y 8 primeros bytes del paquete al que hacen respuesta (en caso de haberlo).

Un protocolo de aplicación que utiliza por debajo **ICMP** es **traceroute** (el que te muestra los routers por los que pasa una consulta IP). Funciona mediante el envío de muchos paquetes al destino, con un puerto raro, y TTL de 1, 2, 3, … Cuando los paquetes están en nodos intermedios, a algunos se les agotará el TTL, y enviarán un error ICMP, que son los que usamos para ir mostrando los nodos intermedios. Finalmente, al llegar al destino, como hemos puesto un puerto raro, nos dará otro error y así sabremos que hemos acabado.

4.4 IPV6

Surge como respuesta principalmente al agotamiento del espacio de IPs de 32bit a 128bit.

Tiene el siguiente formato:



Las principales diferencias son, primero, que la cabecera es de **longitud fija** (40B), segundo, que no se permite **la fragmentación** (si un paquete no cabe en un enlace, se manda error al origen y que reenvíe), tercero, que se quita la **suma de comprobación** y cuarto, que las **opciones** van ahora dentro del campo de datos.

- La **clase de tráfico** es como el tipo de servicio de IPv4, más o menos.
- La etiqueta de **flujo** se utiliza para identificar flujos continuos de datagramas
- **Longitud de carga** es, en bytes, la longitud de los datos.
- **Siguiente cabecera** es equivalente a 'protocolo' en IPv4

Transición desde V4

En lugar de, un día de golpe, cambiar de tecnología, la transición se está haciendo gradualmente. Hay dos métodos:

- **Pila dual:** dos routers consecutivos se enviarán IPv6 si son compatibles, y si no, pues convertirán el datagrama en uno IPv4 (perdiendo algo de información exclusiva de IPv6)
- **Tunelización:** cuando un router IPv6 detecta que el siguiente es IPv4, coge y mete tooodo el paquete IPv6 dentro de un IPv4 (en lugar de convertirlo, como hacíamos antes) y lo manda. Cuando un router IPv6 vea el pseudopaquete este, extraerá el IPv6 y lo enviará así. En este no hay pérdida de datos.

5. Algoritmos de enrutamiento

Existen tres maneras de clasificar los algoritmos de enrutamiento:

- O **globales** (también conocidos como *de estado de enlaces*, calculan las rutas mínimas entre destinos utilizando toda la información de la red) o **descentralizados** (también conocidos como *de vector de distancias*, y realizan los cálculos entre todos los nodos de la red.
- O **estáticos** (las rutas apenas cambian, normalmente solo por acción humana) o **dinámicos** (las rutas se modifican al cambiar el tráfico o topología de la red).
- **Sensibles a la carga** (los costes de los enlaces varían para representar el nivel actual de congestión) o no.

5.1 Algoritmo de enrutamiento de estado de enlaces (LS)

Como ya hemos dicho, en este tipo de algoritmos son conocidos todos los costes de los enlaces. Esto se consigue haciendo que cada nodo envíe los costes de llegar a sus vecinos. Para calcular la ruta de coste mínimo entre dos nodos, se utiliza el algoritmo de Dijkstra.

Breve explicación de Dijkstra

Vamos rellenando una tabla asín:

Paso	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)
0	u	2,u	5,u	1,u	inf
1	ux	2,u	4,x		8,x

En este caso, estamos calculando las rutas de coste mínimo desde **u** hasta el resto de nodos. Para ello, vamos apuntando en entradas de la tabla:

- N': nodos ya comprobados
- D(i): distancia mínima desde **u** hasta **i**
- p(i): predecesor a **i** en la ruta de coste mínimo desde **u**

Y lo que hacemos básicamente en cada iteración es, para cada columna:

1. Mirar si está en N'. Si está, no tocar.
2. Apuntar el coste D(***i***) para llegar al nodo de la iteración (último metido en N')
3. Mirar si desde ***i*** podemos llegar al nodo de la actual columna (**j**). Si es así, miramos cual es el coste del enlace que los une, y le sumamos D(***i***). Si esta suma es menor que el anterior D(**j**), ponemos ahora este nuevo valor.

Y una vez recorridas las columnas, buscamos aquella en la que D(**j**) sea menor y no esté ya en N', y la metemos como siguiente variable a comprobar.

Finalmente, debemos asegurarnos de que los routers de la red no ejecutan a la vez este algoritmo, ya que en ese caso y si utilizásemos información de congestión para los costes de enlaces, podríamos tener oscilaciones constantes de tráfico (porque todos los routers envíen por la misma ruta los datos, colapsándola, luego elijan otra, la colapsen, etc).

5.2 Algoritmo de enrutamiento por vector de distancias (DV)

En este algoritmo, cada nodo recibe información de sus vecinos, realiza un cálculo, y les devuelve el resultado a estos (por eso decimos que es **distribuido**). Es **iterativo** también porque continuamos iterando hasta que no hay más info que intercambiar, y es **asíncrono** porque carece de sincronización. Utiliza la ecuación de **Bellman-Ford**

Breve explicación del algoritmo

Cada nodo contiene la siguiente información:

- Su vector de **distancias** (con estimaciones actualizadas de su coste para llegar a todos los nodos de destino)
- El **coste** para llegar a todos sus vecinos
- Los **vectores de distancias** de los vecinos.

Un nodo **actuará** cuando reciba un anuncio de un nuevo vector de distancias de un vecino, o cuando un coste de uno de sus enlaces varíe. Lo que hará es recalcular su

vector de distancias usando Bellman-Ford: $D_x(y) = \min_v\{c(x,v) + D_v(y)\}$ para cada nodo de la red y, y donde v son los vecinos del nodo actual, x.

Si dicho vector de distancias varía, entonces el nodo enviará este nuevo vector a todos los vecinos.

Finalmente, para no encontrarnos en mitad de **bucles de enrutamiento** (en los que X enruta por Y, e Y por X) cuando el coste de un enlace aumente y se dé un problema de *cuenta hasta infinito*, lo que hacemos es utilizar el método de la **inversa envenenada**: si X enruta por Y para ir hacia Z, le dirá a Y que su coste hacia Z es infinito. Sin embargo, este método no evita bucles de enrutamiento de tres o más nodos.

5.3 Enrutamiento jerárquico

Los anteriores algoritmos solo se pueden aplicar en subredes más o menos pequeñas, debido a la cantidad de información que habría que manejar sino, y debido a que los dueños de empresas y demás quieren cierta autonomía administrativa en sus redes. Por ello, la red se divide en los llamados **sistemas autónomos** (AS).

Los routers de un mismo AS ejecutan el mismo algoritmo de enrutamiento, pero puede ser distinto al de otros. Los AS se conectan entre sí designando a unos routers, llamados **de pasarela** (Gateway) como los encargados. Si un router interno del AS quiere enrutar un paquete a otro AS, pues lo manda a un router de pasarela, y que él se encargue. ¿Cómo elige a qué router de pasarela mandarlo (si hubiera más de uno)? Pues primero determina qué hosts son alcanzables por cada router Gateway de su AS, y si solo es por uno, lo manda por ese, si es por varios, lo manda al que más cerca le pille, y que ese se encargue (**método de la patata caliente**).

6. Enrutamiento en internet

Los protocolos de enrutamiento interno de sistemas autónomos en internet se conocen como **protocolos de pasarela interior.** Los más importantes son RIP (protocolo de información de enrutamiento), OSPF (primero la ruta abierta más corta) y su mejora IS-IS.

De los protocolos de enrutamiento **entre sistemas autónomos,** el más común es el BGP.

6.1 Enrutamiento interno de un AS en internet: RIP

Es un protocolo de vector de distancias entre subredes y con alguna restricción:

- Utiliza como métrica de coste, sí o sí, el número de saltos (es decir, cada enlace cuenta como 1)
- El coste máximo de una ruta es de 15
- Los anuncios **RIP** se realizan cada 30 seg
- La tabla de **enrutamiento** contiene el vector de distancias del router propio, así como el siguiente router de salto calculado para cada entrada.
- Si no recibe noticias de un vecino en 180 segundos, lo da por muerto

Además, es un protocolo de capa de aplicación sobre UDP.

6.2 Enrutamiento interno de un AS en internet: OSPF

Se suele usar en ISP de nivel superior, y es un algoritmo de estado de enlaces, también con alguna peculiaridad:

- El administrador es quien configura los costes de los enlaces
- Un router no solo difunde la información sobre sus enlaces al detectar un cambio en un coste, sino también periódicamente (cada 30 min o así).

Además, es importante mencionar que es un protocolo de **red** (en realidad va sobre IP, como ICMP), y que tiene funcionalidades avanzadas, como: **seguridad**, permitir **varias rutas del mismo coste**, soporte para **unidifusión y multidifusión**, y soporte para establecer jerarquías dentro del sistema autónomo.

7. Enrutamiento por difusión y multidifusión

El **enrutamiento por difusión** consiste en enviar un paquete al resto de nodos de la red; el de **multidifusión** enviar un paquete a grupos de varios nodos a la vez.

7.1 Algoritmos de enrutamiento por difusión

La forma más directa de hacer difusión sería enviar, por unidifusión, el paquete a todos. Pero esto es ineficiente e ineficaz, ya que muchas veces se quiere usar la difusión para poder conocer las rutas de unidifusión.

Inundación no controlada

La siguiente forma más obvia sería que cuando un nodo reciba un mensaje de difusión lo reenvíe a todos sus enlaces, pero esto puede causar, si hay ciclos en la red, repeticiones infinitas de paquetes (**tormenta de difusión**).

Inundación controlada

Podemos controlar la inundación anterior controlando que los routers, cuando reciban un paquete de difusión:

- Apunten su **número de secuencia de difusión** y no vuelvan a enviar otro con el mismo
- Solo transmitan paquetes que les hayan llegado por la ruta de unidifusión más corta desde el origen (protocolo llamado **Reverse Path Forwarding**).

Difusión por árbol de recubrimiento

En las inundaciones controladas, aunque no sufrimos tormentas de difusión, sí hay cierto reenvío de paquetes redundantes. La solución es crear un llamado **árbol de recubrimiento (*spanning tree*)**, que contiene una vez a todos y cada uno de los nodos del grafo. Cuando un nodo quiera hacer difusión, solo la hará a través de los enlaces que formen parte del árbol, y el resto de nodos reenviarán también solo por estos mismos enlaces (y no reenviándoselo al que se lo ha mandado a ellos).

Para montar el árbol de recubrimiento, un método sencillo es el **basado en un nodo central (*rendezvous*)**: se define un centro del árbol, y cada nodo de la red envía mensajes de unidifusión hacia él para unirse. Una vez dichos mensajes llegan al centro, o a otro nodo que ya forme parte del árbol, la ruta que han seguido se convertirá en parte del árbol.

Tema 5: La capa de enlace y redes de área local

1. Capa de enlace: introducción y servicios

1.1 Servicios proporcionados por la capa de enlace

Los posibles servicios que podría ofrecer un protocolo de capa de enlace son:

- **Entramado**: encapsular un datagrama en una trama de enlace con cabecera
- **Acceso al enlace**
- **Entrega fiable**
- **Control de flujo**
- **Detección de errores**; **corrección de errores**
- **Semiduplex:** la comunicación en el enlace solo puede realizarse a la vez en un sentido; **full-duplex:** o en varios.

1.2 ¿Dónde se implementa la capa de enlace?

En tarjetas de red

2. Técnicas de detección y corrección de errores

2.1 Comprobaciones de paridad

En un esquema simple de **paridad** par, añadimos un bit para que el número de 1s de los datos enviados sea par. En uno impar, pues al revés.

En un esquema **de paridad bidimensional**, primero colocamos los datos por filas en una matriz cuadrada o rectangular, y luego añadimos una fila y una columna extra con bits de paridad para cada columna y fila de datos. En este esquema podremos detectar muchas veces dónde está el fallo, y corregirlo por tanto.

2.2 Métodos basados en suma de comprobación

Igual que en los protocolos de transporte y red

2.3 Comprobación de redundancia cíclica (CRC)

Interpretamos las cadenas de bits como coeficientes binarios de un polinomio. Sea D el campo de datos a enviar, R el CRC que debemos calcular, de longitud r previamente acordada, debemos:

1. Acordar un generador, G, de número de bits significativos $r + 1$
2. Calcular $R = resto\,\frac{D \cdot 2^r}{G}$ y añadirlo a la trama. Enviarla
3. Comprobar que ha sido recibida bien, dividiendo la trama D_R entre G. Si da 0 de resto, no hay errores.

3. Protocolos de acceso múltiple

Existen dos tipos de enlace: **punto a punto** (un único emisor y receptor) o de **difusión.** Un protocolo de difusión ideal cumplirá 4 condiciones:

- Cuando solo emita un nodo, lo hará aprovechando la totalidad de la v transmisión R.
- Cuando emitan M nodos, de media lo harán a R/M

- El protocolo será descentralizado para evitar puntos únicos de fallo
- El protocolo será simple

3.1 Protocolos de particionamiento del canal

Están **FDM** y **TDM,** que funcionan igual que los ya explicados en el tema 1, y **CDMA** (multiplexación por código), que permite que varios nodos emitan simultáneamente codificando sus mensajes con un código único cada uno.

3.2 Protocolos de acceso aleatorio

En estos protocolos, los nodos intentan emitir a máxima velocidad R, y cuando detectan que colisionan, se paran, esperan un rato aleatorio (para no volver a chocar) y vuelven a emitir.

Aloha con particiones

En este protocolo, todas las tramas son de longitud fija L, el tiempo se particiona en trozos para poder emitir justo una trama, los nodos transmiten tramas justo al principio de las particiones (para lo cual están sincronizados), y suponemos que si dos o más tramas colisionan, todo el mundo se dará cuenta antes de que acabe el *slot* de tiempo.

Todos los nodos intentan enviar datos –si los tienen- al comienzo de un slot. Si colisionan, cada uno de ellos retransmitirá en el *slot* siguiente con una probabilidad ***p***.

Si calculamos la **eficiencia** de un protocolo como el porcentaje de tiempo que se están enviando datos útiles, suponiendo que todos los nodos tengan datos que enviar, nos da $\frac{1}{e} = 0{,}37$

Aloha

Es el precursor del anterior, que no necesita **sincronización**, pero a cambio no hay particiones ni por tanto los nodos transmiten solo al comienzo de una. Su eficiencia es de la mitad que el anterior.

CSMA

Incorpora una mejora con respecto a ALOHA: **sondeo de portadora** (escucha el canal antes de transmitir, y si está ocupado, espera un tiempo aleatorio y luego vuelve a intentarlo)

CSMA/CD

Incorpora una mejora más con respecto a CSMA: **detección de colisiones** (mientras transmite, es capaz de detectar colisiones y responder ante ellas).

3.3 Protocolos de toma de turnos

Sondeo (*polling*)

Se designa un **nodo maestro** que va dando turnos de transmitir, uno a uno, a todos los nodos de la red. Elimina las colisiones y particiones, pero a cambio tiene un **retardo** por sondeo, y un **único punto de fallo**.

Paso de testigo (*token*)

Los nodos solo pueden transmitir si están en posesión de un **testigo** único que posee uno de ellos y pasa al siguiente cuando acabe de transmitir. Es **descentralizado** y **eficiente**, pero puede fallar si algún nodo se cae, o si el testigo se ‘pierde’.

3.4 Redes de área local (LAN)

En los 90, las LAN mayoritarias eran o bien Ethernet (con mecanismo de acceso aleatorio) o de paso de testigo (*token ring* o FDDI son dos ejemplos).

Token ring

Los N nodos de la red están en un anillo, y siempre que se envía una trama, se espera a que esta vuelva por el lado contrario al emisor para confirmarla. Esto es poco eficiente para redes grandes.

FDDI

Se usa en redes más grandes, y es como *token ring*, pero la trama se envía solo hasta el destino.

4. Direccionamiento en la capa de enlace

4.1 Direcciones MAC

Las direcciones **MAC** son las que se asignan a las interfaces de cada nodo, también llamadas direcciones físicas. Las utilizadas de manera estándar son 2^{48} direcciones de 6 bytes, permanentes (en teoría, no deberían poder cambiarse nunca en un hardware), y son gestionadas por el IEEE.

4.2 Protocolo de resolución de direcciones (ARP)

Para poder conocer la dirección MAC correspondiente a una IP, dentro de una subred, se usa este protocolo, que está a caballo entre las capas de red y enlace.

En él, se define una tabla que mantendrá cada nodo con las correspondencias que conoce entre MACs e IPs, con un TTL para cada entrada. Para ir rellenándolas, cuando un nodo necesite una IP, cogerá y preguntará mediante un broadcast de enlace (MAC: FF:FF:FF…) y el que sea le contestará ya luego en privado a él con su MAC.

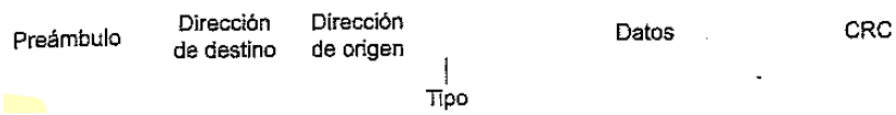
5. Ethernet

Es un protocolo **no fiable** (aunque tiene CRC, si peta no retransmite) y **sin conexión**.

A finales de los 90, las redes Ethernet LAN en bus se fueron sustituyendo por topologías en estrella con **concentradores** (dispositivos físicos con varios enlaces y que actúan a nivel de bits retransmitiendo todos los que llegan por el resto de enlaces).

En la década de los 2000, los **concentradores** dieron pasos a los **conmutadores** (*switches*) que más tarde veremos.

5.1 Estructura de la trama Ethernet



- **Preámbulo:** 8 bytes, 7 de ellos 10101010 y el último 10101011. Sirven para despertar a los adaptadores de red y sincronizar los relojes.
- **Tipo:** protocolo de capa de red al que pasar la trama
- **Datos:** de 46 a 1500B. Si tiene menos de 46B, se rellena hasta llegar!
- **CRC:** de 4 bytes.

Utiliza codificación **Machester** generalmente (un 1 es transición de la señal eléctrica de nivel alto a bajo y 0 viceversa) para mejorar la sincronización.

5.2 CSMA/CD: Protocolo de acceso múltiple Ethernet

Salvo cuando usamos switches, Ethernet es un protocolo de difusión, y por ello hay que controlarlo de alguna manera. En concreto se usa CSMA/CD que tiene una eficiencia de cerca del 100%. Específicamente, opera de la siguiente forma:

1. Se prepara y coloca la trama Ethernet en un buffer del adaptador de salida
2. Si detecta que el canal, durante 96 periodos de bit consecutivos, está vacío, manda la trama. Si no, espera a que se vacíe más 96 periodos de bit más.
3. Mientras transmite, escucha por si alguien más transmite.
4. Si detecta otra señal, deja de transmitir y manda una *jam signal* (interferenc.) de 48bits
5. Después de abortar, entra en fase de *exponential backoff*: siendo n el número de veces que lleva colisionando la trama, se calcula $m = \min(n, 10)$ y luego se selecciona aleatoriamente un valor $K \in \{0,1,2, \dots, 2^m - 1\}$. El adaptador espera $K \cdot 512$ periodos de bit y vuelve al paso 2.

La **eficiencia de Ethernet** es igual a $\frac{1}{1+5 \cdot d_{prop}/d_{trans}}$

5.3 Tecnologías Ethernet

Hay muchas tecnologías de Ethernet, como 1000BASE-LX, 10GBASE-T. El primer numerillo de los nombres es la velocidad: 1000Megabits/seg o 10Gigabits, en estos casos. El BASE es el tipo de multiplexación física utilizada (en este caso, banda BASE), y no hay que saber mucho de ella. La parte final es el tipo de medio físico.

6. Conmutadores de la capa de enlace

Los **conmutadores** o **switches** son dispositivos de capa de enlace que evitan las colisiones en dicha capa mediante un reenvío y filtrado inteligentes. Son transparentes para los nodos.

6.1 Reenvío y filtrado

Llamamos **filtrado** a determinar si una trama debe ser reenviada o descartada, y **reenvío** a determinar a qué interfaz o interfaces se ha de enviar dicha trama y enviarla.

Estas funciones se realizan mediante una tabla que contiene entradas para algunos nodos de la LAN, con su MAC, la interfaz que lleva a ellos, y un TTL.

Siguiendo esta tabla, si llega una trama de la que tenemos la dirección, la mandaremos por donde toque, y si no tenemos la dirección, pues por todos los enlaces (salvo el que la haya mandado, claro).

6.2 Auto-aprendizaje

Las tablas se van rellenando automáticamente cuando pasan tramas por ellas: el switch mira de dónde vienen y qué MAC de origen traen, y se la apunta. Cuando pasa un rato, las borra.

6.3 Propiedades de la conmutación de la capa de enlace

Elimina las colisiones, puede unir enlaces heterogéneos y mejora la seguridad y administración.

6.4 Redes de área local virtuales (VLAN)

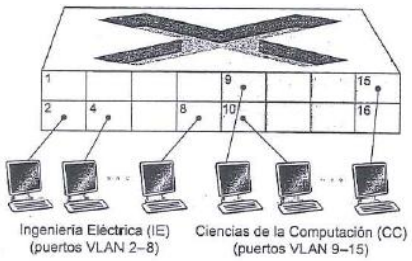
En el mundo de la empresa, las redes vistas hasta ahora tienen algunos aspectos mejorables:

- **Falta de aislamiento del tráfico:** aunque las jerarquías de routers y switches aíslan algo el tráfico, las difusiones pueden atravesar toda la red institucional

- **Uso ineficiente de los conmutadores**: si queremos tener por ejemplo 10 grupos cada uno de 3 personas separados, resulta que necesitamos 10 switches (son muchos)
- **Gestión de los usuarios:** si un empleado cambia de 'grupo', resulta que tenemos que recablear todo.

Para cubrir estas necesidades, surgen las redes **virtuales**, que sobre una única infraestructura de red como las que hemos visto, crean como varios grupos virtuales independientes de hosts que solo pueden comunicarse entre ellos.

Una **VLAN simple, basada en puertos**, simplemente reserva unas interfaces de cada switch para cada grupo, y listo. Para comunicarse dos grupos entre sí, deben hacerlo mediante un router.



Pero, ¿y si quisiéramos conectar en un mismo grupo hosts que están a cierta distancia, y tuviéramos que usar más de un switch? Podríamos conectar de forma chapucera los dos switches entre sí para cada grupo que compartamos entre ellos, pero la solución más eficiente es la técnica de **troncalización VLAN**. Consiste en reservar en cada switch un puerto para conectar a una red troncal común de toda la empresa. Cada vez que queramos comunicarnos con un host de nuestro grupo pero que no esté cerca, hacemos la conexión a través de la red troncal.

Para diferenciar los paquetes que van dirigidos a uno u otro grupo por la red troncal, que es común a todos, se usan unas **Etiquetas Vlan** de 4B que se añaden a la cabecera de Ehternet en el estándar 802.1Q

7. Virtualización de enlaces. La red como capa de enlace

Conmutación de etiquetas multiprotocolo (MPLS)

Este protocolo es complementario a IP, y se creó para mejorar su velocidad en algunos casos. Funciona de forma a las redes de conmutación de circuitos virtuales, usando unas etiquetas para enrutar de forma rápida por rutas ya creadas.

Se implementa mediante una cabecera colocada entre la de capa de red y de enlace y solo funciona entre routers **MPLS**. Cuando se envían paquetes por dos routers MPLS, no se mira la dirección IP, sino solo la etiqueta MPLS, y la IP se guarda por si fuera necesario para otra cosa.

La verdadera **ventaja** actual de MPLS no es la velocidad, sino la **ingeniería de tráfico**: al actuar sobre IP, un ISP por ejemplo puede coger a todos los paquetes que pasen por su red y etiquetarlos al entrar (sin olvidarse de desetiquetarlos al salir), para posteriormente poder controlarlos y mandarlos por las rutas que quieran de forma manual.

Otro ejemplo de uso de **MPLS** es para las **VLAN**.

Mirar punto 5.9 (pág 477) como repaso de todo el temario.