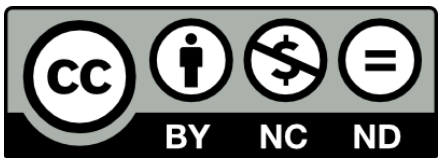
Fundamentos de Redes

Los Del DGIIM, losdeldgiim.github.io

Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada



Fundamentos de Redes

Los Del DGIIM, losdeldgiim.github.io

Irina Kuzyshyn Basarab
José Juan Urrutia Milán
Arturo Olivares Martos

Granada, 2023-2024

Índice general

1. Introducción a los fundamentos de redes

Objetivos

- Conocer y comprender los principios básicos de las comunicaciones.
- Entender el diseño funcional en capas de las redes y los conceptos y terminología fundamentales involucrados.
- Comprender desde un punto de vista teórico-conceptual el modelo de referencia OSI y su correspondencia con el modelo de capas usado en Internet.

Introducción

La arquitectura lógica de Internet está diseñada por capas. Veremos el modelo TCP/IP (aunque mencionaremos el modelo OSI):

Aplicación que hace uso de la red
Transporte **(TCP/UDP)**
Red **(IP)**
Enlace
Física

Tabla 1.1: Modelo de capas del protocolo TCP/IP.

Las dos últimas capas están implementadas en Hardware y las tres primeras en Software, también llamado Network Operating System (NOS). En la asignatura veremos las capas altas, las implementadas en software.

1.1. Sistemas de comunicación y redes

Definición 1.1 (Sistema de comunicación)**.** Es una infraestructura (hw + sw) que permite el intercambio de información. Un sistema típico es el siguiente:

- Tenemos una fuente y un transmisor en un mismo equipo (que es el que va a mandar la información). La fuente genera la información y el transmisor adapta la información al medio.
- Después tenemos el canal de comunicación, el cual produce errores: ruidos, interferencias, diafonías (cuando hay muchos cables en paralelo juntos, puede suceder que la información de un cable se meta en otro)...

- Al final tenemos un receptor y el destino (en un mismo equipo). El receptor adapta la información para el destino y éste espera los datos a recibir.

1.1.1. Motivación para usar redes

Para entender su uso hablaremos de la primera red de comunicaciones, que era una red de telefonía móvil. Cada usuario contaba con su línea de teléfono, que conectaba con una central de conmutación local, luego regional y luego nacional, la cual debía conectarse con la central local del destino. Se usaba conmutación de circuitos.

- Inicialmente se creaba un camino físico juntando cables, llamado circuito.
- Era ineficiente porque no se está hablando todo el tiempo, y los tiempos de silencio el circuito se desaprovecha.
- Era un problema de seguridad el mal funcionamiento de una central, pues dejaba a miles de teléfonos sin servicio.

Si ahora pensamos en ordenadores (o equipos más generales, móviles, PCs, portátiles, móviles...) en vez de móviles, y cambiamos las centrales de conmutación por routers, contamos con muchísimos caminos para conectar dos ordenadores, haciendo más segura la red.

En la actualidad ya no tenemos un camino físico, sino que son los routers quienes deciden por dónde enviar los paquetes y en qué momento. Estos tienen colas, lo que supone algo de retardo, pero tienen la ventaja de que se usa mucho mejor el canal y hay más seguridad, pues hay más de un camino.

Definición 1.2 (Red)**.** Sistema de comunicación con sistemas finales o terminales autónomos (con capacidad para procesar información) que facilita el intercambio eficaz y transparente de información. Concretamente tenemos:

- **Hosts:** sistemas finales o terminales autónomos. Son los que transmiten y reciben datos.
- **Subred:** infraestructura para el transporte de información, formada por líneas de transmisión y nodos o elementos de conmutación: routers y switches.

De una red esperamos:

- Autonomía.
- Interconexión.
- Intercambio de información con eficacia y transparencia.

En cuanto a medios de transmisión, originalmente se usaban cables de pares (pensados para transmitir 4 kHz, la media de la voz humana), luego cables coaxiales, que mejoraron mucho; y fibra óptica que puede transmitir sin interferencias, por lo que es el mejor medio guiado existente. Los cables trenzados son para distancias más cortas, Ethernet por ejemplo.

1.1.2. Topologías de redes

Dada una subred, su topología es el patrón de interconexión entre sus nodos. Las más relevantes las veremos a continuación.

En bus: Todos los nodos tienen acceso a un mismo medio, conocido como bus. Es la más sencilla, pero como el medio es común, todos intentan acceder y se producen colisiones.

En anillo: un círculo en el que tenemos los distintos nodos. Es similar al bus pues el medio es compartido. Una versión habitual es **token ring**, testigo de anillo, en el que se usa un testigo que se van pasando, de forma que así se evitan colisiones.

En estrella: todos están conectados a un centro principal, típicamente un switch.

A diferencia del bus, en este caso cada cable es independiente del resto. Si un PC pone algo en una toma el resto no lo escuchan. Cada línea tienen una cola para guardar a dónde enviar los paquetes y el switch tiene un procesador que coge los paquetes de dichas colas y los envían a las salidas. Es una topología mucho más segura por el hecho de no compartir el medio.

En árbol: típica en redes empresariales. Se suele estructurar en tres niveles:

- Primer nivel: red troncal.
- Segundo nivel: red de división.
- Tercer nivel: red de acceso.

Los equipos de primer y segundo nivel suelen ser switches.

Como potencial riesgo, pueden aparecer ciclos en el árbol. Ethernet no tiene ningún mecanismo para evitar que un paquete se mueva en círculo, lo que echaría la red abajo. El protocolo Spanning Tree Protocol (STP) elimina en cualquier topología los enlaces redundantes que formen bucles.

Mallada: todos los nodos están conectados entre sí por medios independientes.

Es muy fiable, ya que si se cae un enlace tienes más caminos para llegar a tu destino. No obstante, no es escalable, ya que si metemos un $n-$ésimo nodo hay que meter $n-1$ enlaces. Para redes pequeñas es de gran utilidad.

Dentro de una empresa, la red troncal puede seguir esta topología para evitar caídas importantes.

Híbrida: se usa una mezcla de todas. Es la más utilizada.

En cuanto a las topologías que comparten el medio, para evitar el ya mencionado problema de las colisiones, se usan los dos protocolos que veremos a continuación.

Definición 1.3 (CSMA/CD)**.** El algoritmo Carrier Sense Multiple Access / Collision Detection (CSMA/CD) se encarga de detectar colisiones en topologías que comparten el medio, y dar error en caso de que se produzcan. Estas se detectan comprobando si lo que hay en el bus es lo que se acaba de poner.

Es usado por Ethernet.

Definición 1.4 (CSMA/CA)**.** El algoritmo Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA) se encarga de evitar colisiones en topologías que comparten el medio. Para ello, primero escucha el medio y, si no hay ningún mensaje, envía el mensaje. Si no recibe confirmación, hay colisión.

Es usado por Wi-Fi.

1.1.3. Clasificación de redes

Según tamaño y extensión:

- Personal Area Network (PAN): Red de área personal. Incluye todo lo que puede tener una persona: relojes, portátiles, cascos...
- Local Area Network (LAN): Red de área local. Abarca unas decenas de metros, suele ser un mismo edificio.
- Metropolitan Area Network (MAN): Red de área metropolitana. Se usa para conectar un campus o una ciudad.
- Wide Area Network (WAN): Red de área extensa. Son redes desponibles en todo el país, como las redes telefónicas.

Según tecnología de transmisión:

- Difusión: lo que pone un nodo en el medio le llega a todos. Ejemplo de esto es un HUB.
- Punto a punto: Cada nodo solo está unido a otro. Ejemplo de esto es un switch.

Según el tipo de tranferencia de datos:

- Simple: solo transmite o recibe. Por ejemplo los TDT (para que una televisión analógica reciba señal digital).
- Half-duplex: transmite y recibe, pero no simultáneamente. Por ejemplo el Wi-Fi, aunque como cambia muy rápido no nos damos cuenta.
- Full-duplex: transmite y recibe simultáneamente. Por ejemplo, Ethernet.

1.1.4. Nomenclatura típica en figuras (Iconos)

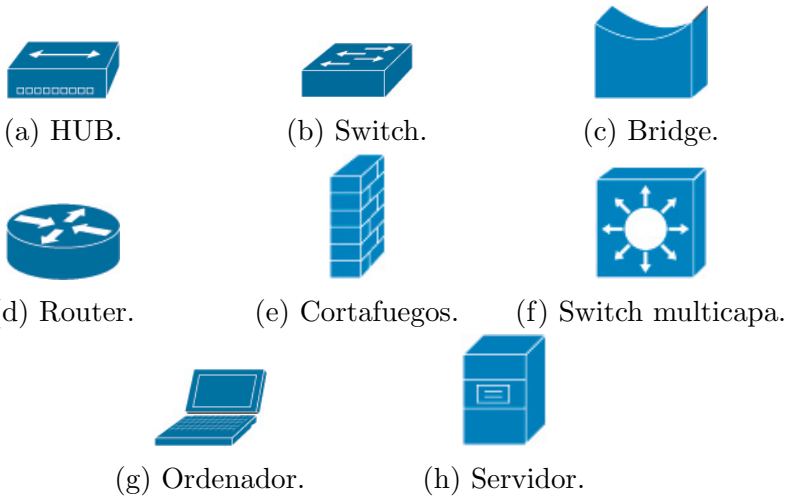
HUB: es un concentrador: permite centralizar los nodos de una red de computadoras. Se implementa mediante un bus.

Switch: tiene muchas bocas y conecta dispositivos dentro de la misma red LAN. Funciona en el nivel de enlace (nivel 2).

Bridge: funciona como un switch, pero uniendo tecnologías distintas. También funciona en el nivel de enlace.

Router: tiene pocas bocas, y se usa para conectar distintas redes.

Cortafuegos: bloquea el acceso no autorizado a una red, permitiendo tan solo el autorizado.



(a) HUB. (b) Switch. (c) Bridge.

(d) Router. (e) Cortafuegos. (f) Switch multicapa.

(g) Ordenador. (h) Servidor.

Figura 1.1: Iconos de los distintos elementos de una red.

NAT: dispotivo en el que se ejecuta el mecanismo Network Address Translation (NAT), que permite que una red privada pueda acceder a Internet. Se explicará más adelante.

Switch multicapa: todas las bocas de un switch pertenecen a la misma LAN, pero esto a veces no nos interesa. Podemos hacer distintas redes virtuales (VLAN) dividiendo un mismo switch en varias redes, permitiéndonos esto conectar dos switches distintos dentro de la misma VLAN. Esto no nos permite movernos por distintas redes en el mismo switch, ya habría que pasar por un router al necesitar movernos a nivel de red para cambiar de red[1].

Esta es la funcionalidad que sí se puede hacer con un switch multicapa, no necesitar pasar por un router para cambiar de red.

Todos estos elementos los veremos representados en distintos esquemas para mostrarnos las topologías de las redes. Estos vendrán identificados por los símbolos de la Figura 1.1.

1.2. Diseño y estandarización de redes

La idea principal que se sigue al diseñar redes es solucionar los problemas en capas. Se estandarizan Modelos de Referencia (no son implementaciones, solo una referencia), en los que se definen las distintas capas y las funcionalidades de cada una. Los principios que se siguen son que las funcionalidades distintas tienen que estar en distintas capas y minimizar el flujo de información entre las capas.

A continuación detallamos las capas de los modelos de referencia más importantes, junto a las funcionalidades de cada una y los problemas que han de solventar.

Capa física: se encarga de transmitir los datos. Hay distintos tipos de codificaciones para enviar bits de información.

[1]No se verá en la asignatura.

Capa de enlace: se encarga de los mecanismos de acceso al medio. Si hay un medio común, antes de transmitir datos tiene que asegurarse de que ningún equipo está transmitiendo. Suele seguir dos protocolos:

- Media Access Control (MAC), control de acceso al medio.
- Logical Link Control (LLC), control de acceso lógico para las primeras retransmisiones. Si algún paquete llega mal, retransmite varias veces.

Capa de red: una vez llegado a este punto, se asume que no han habido colisiones en la comunicación. Esta capa se encarga principalmente de:

- El direccionamiento: saber dar una dirección y tener un identificador dentro de la red.
- El encaminamiento: saber cómo llegar al destino.

Capa de transporte: se encarga de recuperar los paquetes que en la capa de enlace no se ha podido. Es la capa encargada de la fiabilidad.

- Corrige errores.
- Gestiona la congestión de la red.
- Control de flujos: si hay un receptor más lento que el emisor, debe decirle al emisor que disminuya la velocidad de emisión, para adecuarse a la del receptor.

Además se encarga de la multiplexación de datos: mediante puertos (los veremos más adelante) le indica al SO a qué aplicación corresponde cada paquete.

Capa de aplicación: los clientes y los servidores deben buscar alguna forma de comunicarse.

Los dos modelos de referencia más importantes son el OSI y el TCP/IP, que describiremos a continuación.

1.2.1. Modelo OSI vs TCP/IP

Aplicación
Presentación
Sesión
Transporte
Red
Enlace
Física

Tabla 1.2: Modelo OSI.

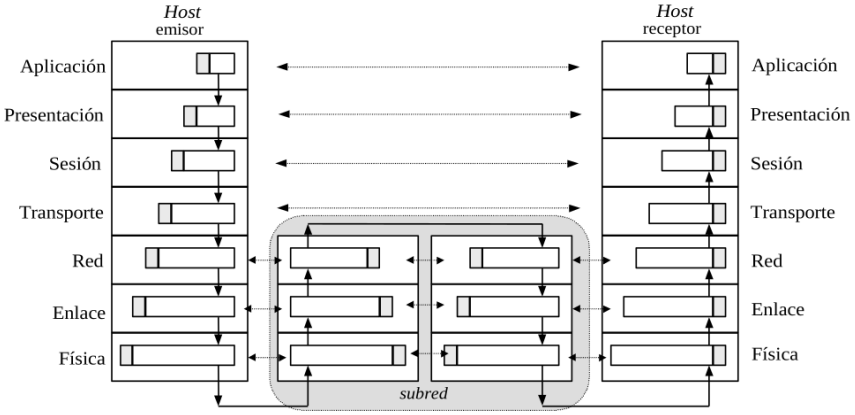


Figura 1.2: Comunicación real frente a comunicación virtual.

Aplicación
Transporte
Red
Red subyacente

Tabla 1.3: Modelo TCP/IP.

El modelo Open System Interconnection (OSI) fue propuesto por la ISO y el TCP/IP por el IETF. Las tres primeras capas del modelo OSI se corresponden con la capa de aplicación, y las dos última con la red subyacente, que es la parte física. Esta última en el modelo TCP/IP dependiendo un poco de la tecnología está implementada de una forma u otra, pero la comunicación con la capa de red no puede variar, pues la capa de red sí está estandarizada.

- Las capas físicas solo se encargan de hacer la primera conexión.
- La capa de red, salto a salto se encarga de llegar al destino, usando routers y sus tablas de encaminamiento.
- Una vez hecho el encaminamiento, las capas superiores son solo de los extremos, son los computadores de los extremos los que se comunican.
- Por tanto, los computadores tienen las 5 capas (en el caso de TCP/IP) y los routers solo las 3 más bajas.

1.3. Terminología, conceptos y servicios

En la Figura 1.2 vemos el camino que siguen los datos que le manda un emisor a un receptor. A la hora de emitir, cada capa recibe los datos de la capa superior, conocidos como Unidad de Datos de Servicio (Service Data Unit (SDU)), y le añade una cabecera, formando la Unidad de Datos de Protocolo (Protocol Data Unit (PDU)). Decimos que los datos de la capa superior se han *encapsulado* en la capa inferior. Por tanto, cada capa envía el PDU a la capa inferior convirtiéndose en el SDU de la

capa inferior. La capa física, al no tener capa inferior, manda señales eléctricas en vez de bits.

Por otra parte, en la recepción, cada capa recibe de la inferior el PDU, al cual le quita la cabecera quedándose con el SDU. En función de la cabecera, se estudia qué ha de hacerse en cada capa y, posteriormente, se envía el SDU a la capa superior.

En cada capa, el PDU se puede también denominar $n-$PDU, o con la inicial del nombre de la capa (en inglés). Por ejemplo, el PDU de la capa de transprte se puede abreviar por TPDU, y el de la capa de aplicación por APDU. Estos PDU también se pueden denominar como sigue:

- Capa de enlace: trama.
- Capa de red: datagrama.
- Capa de transporte: depende del protocolo:
  - TCP: segmento.
  - UDP: datagrama.
- Capa de aplicación: mensaje.

La información que se envía desde un host a otro ha de pasar por todas las capas para llegar al destino, lo que se conoce como *comunicación real* o vertical, y viene representada en la Figura 1.2 por las flechas continuas. No obstante, y tan solo en sentido abstracto, decimos que nos capas del mismo tipo en distintos equipos hablan entre sí, lo que se conoce como *comunicación virtual* u horizontal, y viene representada en la Figura 1.2 por las flechas discontinuas. Esta comunicación virtual, como ya hemos mencionado, no es directa, sino que se realiza a través de recursos que nos proporcionan otras capas adyacentes.

Además de las definiciones que acabamos de dar, otros términos relevantes son los siguientes:

Entidad de nivel n: entidad que se encuentra en la capa $n-$ésima.

Entidades pares: entidades de la misma capa, que se comunican horizontalmente entre sí.

Protocolo: reglas que describen cómo han de comunicarse las entidades pares. En ellos, se especifican los paquetes que se mandan, etc.

Interfaz: a diferencia del protocolo, que se refiere a cómo se comunican las entidades pares, la interfaz se refiere a cómo se comunican las entidades de capas adyacentes.

Service Access Point (SAP): punto de acceso al servicio. Es un punto específico en la interfaz entre dos capas donde se proporciona un servicio.

Servicio: conjunto de funciones que una capa proporciona a la capa superior.

Capa proveedora/usuaria del servicio: en la comunicación vertical, la capa que proporciona el servicio es la proveedora, y la que lo usa es la usuaria.

Pila de protocolos: conjunto de protocolos que se usan en cada capa.

Arquitectura de red: modelo de referencia, junto a la pila de protocolos.

Otro concepto importante es el de *retardo*, que describireos a continuación.

1.3.1. Retardos en la comunicación

Supongamos que queremos transmitir un paquete entre dos equipos (terminales), por medio de otro, un router. Veamos los retardos que tenemos en la comunicación.

1. En primer lugar, hemos de considerar el **tiempo de transmisión**, que es el tiempo que se tarda en poner el paquete en el medio.

   Si el tamaño del paquete es de L bytes y la velocidad de transmisión es de v_t bps, el tiempo de transmisión T_t es de:

   $$T_t = \frac{L \cdot 8}{v_t} \text{ s}$$

   La velocidad de transmisión depende exclusivamente de la tarjeta de red.

2. En segundo lugar, tenemos el **tiempo de propagación**, que es el tiempo desde que se escribe el último bit en el medio hasta que llega este último bit al siguiente equipo.

   Este tiempo depende de la distancia entre los equipos y de la velocidad de transmisión, que depende del medio. En el caso de una transmisión inalámbrica, la velocidad de transmisión es la velocidad de la luz, mientras que en una transmisión por cable suele ser de $^2/_3$ de la velocidad de la luz.

   Si la distancia entre los equipos es de d m y la velocidad de transmisión es de v_p $^{\text{m}}/_{\text{s}}$, el tiempo de propagación T_p es de:

   $$T_p = \frac{d}{v_p} \text{ s}$$

3. En tercer lugar, tenemos el tiempo que se encuentra en el equipo intermedio, en nuestro caso el router. Cuando el paquete llega al equipo intermedio, éste lo mete en una cola hasta que pueda procesarlo. El tiempo que el paquete está en la cola se conoce como **tiempo de cola**, y depende de la situación del equipo. Además, el equipo ha de procesar el paquete, lo que le lleva un tiempo conocido como **tiempo de procesamiento**, que suele ser del orden de milisegundos. Por último, y para poder continuar con la comunicación, el equipo ha de obtener acceso al medio, lo que le lleva un tiempo conocido como **tiempo de acceso al medio**.

   Por tanto, el tiempo que el paquete está en el equipo intermedio es la suma de estos tres tiempos.

4. Por último, la comunicación deberá continuar, por lo que se obtienen nuevos retardos de transmisión, propagación, etc. No obstante, en estos casos, estos no serán los mismos, pues la distancia entre equipos, tarjetas de red, etc., serán distintos.

1.3.2. Tipos de servicios

Relacionado con el nivel de transporte, hay dos clasificaciones importantes:

Según la conexión: En función de si, antes de enviar un paquete, se comprueba que el otro equipo esté encendido o no, tenemos dos tipos de servicios:

- Service Oriented to Connection (SOC): sí se comprueba.
- Service Not Oriented to Connection (SNOC): no se comprueba.

Según la fiabilidad: En función de si se asegura que todo funcione bien o no, (por ejemplo, que todos los bits de un archivo estén bien), tenemos dos tipos de servicios:

- Fiable: se asegura de que todo funcione bien. Si algo falla, la conexión se termina.
- No fiable: no comprueba que todo funcione bien. La finalidad de estos protocolos es ser rápidos.

Para tener un protocolo fiable, contamos con los siguientes mecanismos:

- Control de conexión. Ser fiable implica ser orientado a conexión.
- Control de errores.
- Control de congestión. Hablamos de congestión cuando las colas de los routers se llenan y empiezan a descartar paquetes.
- Control de flujo.
- Entrega ordenada. Si se envían muchos paquetes, estos deben llegar en orden.

Algunos protocolos que estudiaremos más adelante:

- TCP es un servicio fiable, y por tanto orientado a conexión.
- UDP es un servicio no orientado a conexión, y por tanto no fiable.

1.4. Internet: topología y direccionamiento

Internet tiene dos aspectos importantes:

- Los protocolos de comunicación.
- Cómo se organiza Internet, el direccionamiento.

Todo esto se describe en las conocidas Request for Comments (RFC), y son de gran importancia pues en ellas se describen estos protocolos con detalle. Cada protocolo está descrito en una (o normalmente varias) RFC, aunque en este documento no las mencionaremos de manera explícita.

1.4.1. Organización topológica

Los operadores se establecen según la siguiente jerarquía:

- **Tier 3**: son los más cercanos a los usuarios. Ofrecen servicios de conectividad a empresas y particulares, y se conocen como Internet Service Provider (ISP). Algunos conocidos en España son Movistar, Vodafone, Orange...
- **Tier 2**: son de ámbito más regional. Necesitan pasar por una Tier 1 para llegar a toda Internet y ofrecen servicios de conectividad a operadores de Tier 3.
- **Tier 1**: son los que componen la estructura troncal de Internet. Están todos comunicados entre sí y están como mínimo en dos continentes.

Hay dos tipos de relaciones entre operadores:

- **Tránsito:** conexiones entre distintos tier. Por ejemplo un tier 3 paga a un tier superior para enviar datos.
- **Peering:** conexiones entre el mismo tier.

Antiguamente, para que un ISP de un país hablase con otro del mismo país había que ir hasta EEUU por falta de recursos. Más tarde se pusieron puntos neutros en cada país para comunicar operadores dentro de un mismo país.

1.4.2. Red Iris

La Red Iris es la red española para investigación. Todas las universidades públicas y centros de investigación están conectados a ella.

La red se divide según autonomía (en Andalucía, se denomina Red RICA) y por otro lado tiene conexiones externas con la red científica europea.

1.4.3. Direccionamiento por capas

Según la capa en la que nos encontremos, hemos de realizar el direccionamiento de una forma u otra.

- Capa de Enlace: La dirección depende de la tarjeta de red. Las direcciones MAC son de la forma AA:BB:CC:DD:EE:FF, y son teóricamente únicas en todo el mundo. No obstante, en la realidad se ponen aleatorias para evitar seguimientos (puesto que si sabemos la dirección MAC de una tarjeta podemos hacer seguimiento de paquetes).
- Red: Se usan direcciones IP de la forma A.B.C.D. Las públicas son únicas en todo el mundo, las privadas no.
- Transporte: Direccionamiento a través de puertos, que identifican a qué proceso va un determinado paquete.
- Aplicación: Nombres de dominio mediante DNS.

2. Capa de red

En el presente tema, estudiaremos a fondo la capa de red. Recordemos que seguimos el Modelo TCP/IP descrito en la Tabla 1.1. Esta capa es la capa de más bajo nivel que pertenece al NOS, y será la primera estudiada en la asignatura.

Objetivos

- Comprender las funcionalidades y servicios de la capa de red.
  - Concepto de conmutación de paquetes y datagramas.
  - Direccionamiento en Internet.
  - Encaminamiento salto a salto.
  - Asociación con la capa de enlace a través del protocolo ARP.
  - Señalización de errores mediante el protocolo ICMP.

2.1. Funcionalidades

Funcionalidades y servicios TCP/IP:

- **Direccionamiento:** identificación de equipos dentro de la red.
- **Encaminamiento:** llegar salto a salto desde el origen al destino. Especifica el camino que deben seguir los paquetes.
- **Fragmentación:** las tarjetas suelen tener un tamaño máximo de paquete, y si queremos enviar un paquete más grande tenemos que fragmentarlo y, en el destino, ensamblarlo.
- **Conmutación.**
- **Interconexión de redes.**
- En OSI: **control de gestión.**

El protocolo que desarrollaremos en este tema es IP por ser el que en la actualidad se ha impuesto, aunque existen otros como ATM, x25...

2.2. Conmutación

Definición 2.1 (Conmutación)**.** Acción de establecer o determinar caminos de extremo a extremo que permitan transmitir información.

Uno de los primeros ejemplos claros de conmutación que se vio en la tecnología de las comunicaciones fue la conmutación de circuitos para la telefonía, que desarrollamos a continuación.

2.2.1. Conmutación de circuitos

Antiguamente existía una conmutación física de circuitos, muy usada en telefonía. De esta forma, hay muchos cables entre los usuarios y las centrales (uno por cada usuario), y menos cables entre cada par de centrales, ya que no todos los usuarios hablan al mismo tiempo.

La comunicación por conmutación de circuitos implica tres fases:

1. El establecimiento del circuito. Cada central une los cables que correspondan y se genera el camino.

2. La trasferencia de datos a través del circuito dedicado.

3. La desconexión del circuito, se libera el circuito para su reutilización.

Beneficios

- Recursos dedicados (tenemos un cable solo para nosotros), lo que facilita las comunicaciones a tiempo real y sin retardos.

- El recurso se mantiene dedicado toda la sesión.

- No hay competición por conseguir el medio.

- El circuito es fijo, no hay decisiones de encaminamiento una vez establecido.

- Simplicidad en la gestión de los nodos intermedios.

Desventajas

- Cuando un usuario no usa su cable no lo usa nadie más. Uso ineficiente de recursos.

- Hay establecimiento de llamada (para que todos los cables se toquen).

- Es poco tolerable a fallos, si algo no funciona, todo deja de funcionar.

2.2.2. Conmutación de paquetes

En la actualidad, no se envía una señal analógica; sino que, como sabemos, se envía el SDU junto con la cabecera. El SDU se ha de fragmentar en distintos bloques, a los que denominaremos *paquetes* o *fragmentos*. Por tanto, los paquetes son cada uno de los bloques del SDU de la capa de red y, tras añadirle a cada uno su correspondiente cabecera, es lo que se envía como tal por la red.

Observación. En general, cuando se quiera hacer referencia a un conjunto de datos que se envía por la red, sin especificar en qué capa nos encontramos, o sin ser más precisos, también usaremos el término de *paquete*.

A la hora de realizar la conmutación, hay dos formas de hacerlo, la conmutación mediante datagramas y la conmutación mediante circuitos virtuales.

Conmutación de datagramas

Las características de la conmutación de datagramas son:

- No hay establecimiento de conexión: enviamos un paquete y no sabemos si el otro extremo está encendido.
- El envío de los distintos paquetes se hace independientemente. El encaminamiento se hace paquete a paquete, por lo que se pueden seguir caminos distintos. Por este motivo, los paquetes pueden llegar desordenados, algo que controlarán otras capas.

  Además, si se produce fragmentación, no se ensamblarán los paquetes hasta que lleguen al destino, ya que distintos paquetes de un mismo datagrama pueden seguir distintos caminos.
- En cada nodo intermedio los paquetes que llegan se almacenan en una cola, y cuando sea posible se envían al próximo nodo.
- Como el encaminamiento se hace salto a salto, todos los paquetes han de tener la dirección de origen (para las respuestas o encaminamientos específicos, aunque esto no lo veremos en la asignatura) y de destino. A veces, para hacer difusiones de datos, nos puede interesar taner varias direcciones de destino.

Como el medio es común, los nodo de interconexión necesitan colas para poder gestionar los paquetes que le llegan. A la hora de esta conmutación, se hace el mejor esfuerzo, pero si algo falla la capa de red no se encarga de gestionar el fallo.

Un protocolo que lleve a cabo esta conmutación es IP, que desarrollaremos más adelante. Este es el tipo de conmutación que se usa mayoritariamente en la actualidad, y es en la que nos centraremos en la asignatura.

2.2.3. Conmutación con circuitos virtuales

En este caso, la conmutación difiere ligeramente de la conmutación por datagramas vista, siendo una mezcla entre la conmutación de circuitos y la de paquetes.

En este caso, para enviar un paquete de un origen a un destino, aunque haya distintos caminos posibles, se establece el camino desde el principio denominado

circuito (siguiendo la idea de la conmutación de circuitos). Este circuito no obstante es virtual, ya que los recursos no son dedicados completamente, sino que se reservan temporalmente pero se pueden reutilizar.

Cada router decide el camino que seguirá cada paquete, y los paquetes del mismo datagrama seguirán el mismo camino. Por tanto, el primer paquete en llegar al router reservará los recursos para los próximos paquetes.

- Hay que establecer conexión para averiguar la ruta a seguir.
- Si un router se cae, se cambia el camino.

Un protocolo que lleva a cabo esta conmutación es Asyncronous Transfer Mode (ATM), que estaba presente en el inicio de la telefonía digital.

2.3. El protocolo IP

El Internet Protocol (IP) es un protocolo para la interconexión de redes. Existen dos versiones:

- IPv4: Es la que se diseñó inicialmente, aunque tiene una limitación en la cantidad de direcciones.
- IPv6: Pasó de 32 a 128 bits, lo que supone una cantidad en la práctica ilimitada de direcciones.

En la actualidad la limitación de direcciones se empieza a notar, por lo que hay una transición gradual hacia IPv6, aunque sigue predominando IPv4. Desorrallaremos IPv4 en este tema, aunque mencionaremos algunas diferencias con IPv6.

Características de IPv4

- Resuelve el direccionamiento en Internet en la capa de red, ya que cada tarjeta de red tiene una dirección IP.
- Realiza el encaminamiento (o retransmisión) salto a salto entre equipos y routers.
- Ofrece un servicio no orientado a conexión y no fiable, ya que:
  - No hay establecimiento de conexión lógica entre las entidades.
  - No hay control de errores ni de flujos. Los errores que se produzcan tienen que arreglarlos una capa superior si se precisa.
- Gestiona la fragmentación para adaptarse al MTU de cada tarjeta de red, como veremos. A la unidad de datos completa se le llama datagrama y a los fragmentos paquetes.
- Es un protocolo de máximo esfuerzo. Los datagramas se pueden perder, duplicar, retrasar, llegar desordenados...

2.3.1. Direccionamiento

Para identificar cada equipo en la red, se usan direcciones IP. El lector posiblemente esté más familiarizado con las direcciones red, como `www.google.com`, pero estas en realidad son nombres de dominio que se traducen a direcciones IP, como veremos en el Capítulo dedicado a la capa de aplicación. Mientras tanto, hemos de saber que todo equipo en la red tiene una dirección IP asociada. Además, esta (a priori) es única y no se puede repetir, lo que supone una limitación. Como más adelante veremos, para solventar este problema se usan también direcciones privadas, algo que no contemplaremos por el momento.

Una dirección IP consta de 32 bits y la nomenclatura usada es: `A.B.C.D` donde cada letra es un número decimal en el rango `0-255` (ya que codificará 8 bits). El rango por tanto que tenemos es `0.0.0.0-255.255.255.255.` Una dirección tiene dos partes bien diferenciadas, la que identifica la red y la que identifica el equipo en cuestión (en realidad, identifica la tarjeta de red).

Para saber qué parte de la dirección IP identifica el equipo y cuál la red, se emplea la máscara de red.

Definición 2.2 (Máscara de red)**.** Es un conjunto de 32 bits (al igual que una dirección IP) que se usa para identificar qué parte de la dirección IP identifica la red y cuál el equipo. Contiene los primeros n bits consecutivos a 1, y el resto a 0.

¿Cómo se usa la máscara?
Para saber cuál es la dirección de la red, se hace un `AND` lógico entre la dirección IP y la máscara (por lo que nos quedaremos con los primeros n bits de la dirección IP). El resto de bits identificará al equipo dentro de dicha red.

Notemos por tanto que, dentro de las posibles direcciones IP de una misma red, la dirección con todos los bits de equipo a 0 está *reservada* para la dirección de la red, y no podrá asignarse a ningún equipo.

Notación. Es común querer dar una dirección IP junto a su máscara de red. Para esto, se podrá usar la notación `A.B.C.D/n`, donde `A.B.C.D` es la dirección IP en sí y `n` es el número de bits a 1 de la máscara de red. Como ya hemos mencionado que estos bits han de estar al inicio y consecutivos, sabiendo el valor de `n` sabremos cuál es la máscara de red.

Ejemplo. Supongamos que tenemos una dirección IP `192.168.1.27/24`, y queremos saber cuál es la dirección de la red. Pasando a binario y haciendo un `AND`, tenemos:

	1100 0000 . 1010 1000 . 0000 0001 . 0001 1011	(dirección IP)
AND	1111 1111 . 1111 1111 . 1111 1111 . 0000 0000	(máscara de red)
	1100 0000 . 1010 1000 . 0000 0001 . 0000 0000	(dirección de la red)

Por tanto, pasando de nuevo a decimal, la dirección de la red es `192.168.1.0`.

Direccionamiento jerárquico

Internet usa direccionamiento jerárquico basado en clases. Cada clase contiene las direcciones IP de un rango determinado:

- Clase A → `0xx...x/8` ⟹ `0.0.0.0 - 127.255.255.255`. Tenemos $2^7 = 128$ redes con $2^{24} \approx 16 \cdot 10^6$ equipos en cada una.
- Clase B → `10xx...x/16` ⟹ `128.0.0.0 - 191.255.255.255`. Tenemos $2^{14} = 16384$ redes con $2^{16} = 65536$ equipos en cada una.
- Clase C → `110xx...x/24` ⟹ `192.0.0.0 - 223.255.255.255`. Tenemos $2^{21} \approx 2 \cdot 10^6$ de redes con $2^8 = 256$ equipos en cada una.
- Clase D → `1110xx...x` ⟹ `224.0.0.0 - 239.255.255.255`. No se usa para identificar equipos ni redes sino para multidifusión (*multicast*). Cada dirección identifica a todo un grupo de equipos. Para gestionar esto existe el protocolo IGMP para suscribirse a grupos.
- Clase E → `1111xx...x` ⟹ `240.0.0.0 - 255.255.255.255`. Es el rango experimental; es decir, las direcciones que se dejan para hacer pruebas.

Observación. Como utilidad, en los sistemas operativos Linux se ofrece el comando `ipcalc`, que podrá ser de utilidad en los ejercicios prácticos. Dada una dirección IP y una máscara de red, nos devolverá la dirección de la red, la dirección de difusión, el rango de direcciones posibles, etc.

Observación. En Linux, podemos emplear el comando `ifconfig` para ver la configuración de red de nuestro equipo. En la salida, veremos para cada una de las interfaces de red la dirección IP, la máscara de red, la dirección de difusión, etc.

Direcciones reservadas

Además de las restricciones de cada clase, hay determinadas direcciones que están reservadas y no se pueden asignar a ningún equipo. Algunas de estas direcciones son:

- Dirección de red: Cualquier dirección IP con todos los bits de equipo a 0. Está dedicada para identificar la red en sí.
- Dirección de difusión (*broadcast*): Cualquier dirección IP con todos los bits de equipo a 1. Se usa para enviar un paquete a todos los equipos de la red.

  Cuando se tiene que encontrar un equipo y no se sabe cuál, se manda por la dirección de difusión y lo escuchará quien tenga que escucharlo.
- `127.a.b.c`: Denominada dirección de *loopback*, *localhost* o *localloop*. Se usa para hacer pruebas, y es una conexión que hacemos a nuestra propia máquina. Originalmente (y la más común) era `127.0.0.1`, pero en la actualidad se ha aumentado el rango. Estas redes no requieren de una tarjeta de red específica, y su interfaz de red se denomina `lo`.

Llegados a este punto, podemos dar una definición más correcta de router, que ya habíamos mencionado anteriormente.

Definición 2.3 (Router)**.** Es un dispositivo de la capa de red cuya funcionalidad principal es conectar distintas redes y encaminar los paquetes a través de ellas.

Cuenta con varias tarjetas de red (también llamadas interfaces), una por cada red a la que se conecta, y cada una cuenta con una dirección IP asociada en cada red.

Como curiosidad, es posible crear routers en un ordenador con varias tarjetas de red con `Linux`. Con el comando `sysctl -a` podemos consultar el valor de la variable `net.ipv4.ip_forward`, que nos informa sobre si redirigimos paquetes o no. Si está con el valor 1, dicho equipo es un router.

Observación. Como un switch funciona a nivel de enlace, todo lo conectado a dicho switch está en la misma red. Por tanto, tampoco tiene dirección IP asignada.

Direccionamiento sin clases

Si usamos solo el direccionamiento con clases estaríamos desperdiciando muchísimas direcciones IP. Por ejemplo, si tenemos 1000 equipos ($2^8 < 1000 < 2^{16}$) tendríamos que usar una red de clase B, con la que desperdiciaríamos más de 60.000 direcciones. La solución a este problema es usar el direccionamiento sin clase, que nos permite usar la máscara de red deseada.

- **Subredes**

  Si, por ejemplo, queremos una red de menos de 256 equipos, podemos aumentar el número de bits de la máscara a 1, para conseguir más bits dedicados a identificar la red y menos para identicar equipos. Cada vez que añadimos un bit a la máscara, estamos dividiendo una red en dos mitades.

  Ejemplo. Supongamos que queremos identificar 100 equipos dentro de una misma red. Contando además con la dirección de red y la de difusión, necesitamos 102 direcciones. Como $2^6 < 102 < 2^7$, necesitamos 7 bits para identificar a los equipos. Por tanto, la máscara a usar será `/25`.

- **Superredes**

  Si hacemos el procedimiento inverso, quitarle un bit a la máscara, duplicamos la cantidad de equipos que podemos direccionar. Por ejemplo, en `/23` estamos juntando dos redes de clase C.

  Ejemplo. Supongamos que queremos una red de 1000 equipos. Contando con la dirección de red y la de difusión, necesitamos 1002 direcciones. Como $2^9 < 1002 < 2^{10}$, necesitamos 10 bits para identificar a los equipos. Por tanto, la máscara a usar será `/22`.

Como vemos el funcionamiento es igual que en el direccionamiento con clase, pero reduce significativamente (aunque no elimina) el desperdicio de direcciones. A nivel práctico red, subred y superred no se diferencian, y nos referimos a todas ellas como redes.

Direcciones privadas

Como hemos venido mencionando en distintas ocasiones, la escasez de direcciones es un gran problema presente en IPv4, ya que tan solo hay 2^{32} direcciones posibles, las cuales ya se agotaron en Noviembre de 2019. Aunque se vayan recopilando direcciones de sitios obsoletos, empresas desaparecidas, etc. el problema sigue existiendo.

Hay varias soluciones posibles para solventarlo.

- Direccionamiento sin clase: es una solucion que reduce el desperdicio de direcciones, pero aun así tiene la limitación de 2^{32} direcciones.

- IPv6, el cual usa 128 bits para las direcciones. La notación utilizada es

  `FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF`, en el que cada dígito es un número hexadecimal.

  En total hay 2^{128} direcciones posibles (más de 10^{37}), lo que en la práctica las hace ilimitadas. Aunque sea compatible con IPv4, la transición está siendo lenta.

- Direcciones privadas: esta es la principal solución que se usa en la actualidad, ya que hace el número de direcciones prácticamente ilimitado. Desarrollaremos este concepto a continuación.

Direcciones públicas: Cada dirección se asigna a un único dispositivo en todo Internet. Se asignan centralizadamente[1], y como son limitadas, hay que pagar por cada una.

Direcciones privadas: Solo se pueden usar en redes privadas o *intranets*, sin acceso directo al resto de Internet. Por tanto, al no ser accesibles desde fuera, se pueden repetir en distintas redes privadas, lo que aumenta el número de direcciones disponibles.

Para poder comunicarse con el resto de Internet (ya que si no tendrían poca utilidad), será necesario una dirección pública por la cual se haga la comunicación. Para esto se usa el NAT, que veremos más adelante.

Respecto al direccionamiento jerárquico con clases, dentro de cada clase se definen algunos rangos de direcciones a usar como IP privadas. Estos rangos son:

- Clase A → `10.x.y.z/8`

- Clase B → `172.16-31.y.z/16`

- Clase C → `192.168.y.z/24`

[1] Inicialmente por IANA, actualmente por ICANN.

2.3.2. Network Address Translation (NAT)

Como hemos mencionado anteriormente, para que una red privada pueda comunicarse con el resto de Internet, es necesario que haya una dirección pública que haga de intermediario. Para esto se usa la técnica de NAT, que posibilita la traducción de direcciones. Al encontramos en la capa de red, el PDU contiene la cabecera IP[2] que contiene, entre otros datos:

- Dirección IP origen (IPsrc) junto con el puerto origen (sport).
- Dirección IP destino (IPdest) junto con el puerto destino (dport).

El concepto de puerto lo veremos más adelante y desarrollaremos a fondo en la Capa de Transporte. Por el momento, tan solo es necesario saber que es un número que se asigna a cada proceso que se comunica en la red, y que se usa para saber a qué proceso enviar la respuesta.

Para posibilitar la traducción, se usa una tabla de traducciones a modo de "diccionario", tal y como introducimos a continuación.

Definición 2.4 (Tabla de traducciones)**.** La Tabla de Traducciones es una tabla que se guarda en la memoria de todo router que haga NAT. Por cada traducción que deba hacerse, se guarda una entrada en la tabla que relaciona la dirección IP y puerto originales con la dirección IP y puerto traducidos.

La tabla se va actualizando con cada nueva traducción, y cuenta con un temporizador (normalmente de 5 minutos) que borra las entradas que lleven un tiempo sin usarse. De esta forma, se evita que la tabla se sature y se libera memoria.

En el caso de que llegue una petición que ya esté en la tabla, se reutiliza la información de la tabla, sin crearse una nueva entrada.

Definición 2.5 (*Masquerading*)**.** Proceso de enmascaramiento que hace el router al traducir la dirección privada del equipo en su dirección pública. Se "enmascara" la dirección privada, de forma que el servidor no sabe a qué equipo de la red privada está respondiendo.

Observación. El uso de NAT plantea un problema de seguridad. Un atacante, conociendo la IP pública del router, puede hacer un barrido de puertos y puede conseguir que algún paquete entre. En tal caso, el router le responderá, y el atacante sabrá que hay un equipo detrás de esa IP pública y puerto, por lo que podrá intentar atacar a ese equipo.

Para evitar esto, para cada traducción puede guardarse tanto las IP y puerto de origen y destino sin traducir, como las traducidas. De esta forma, si llega una petición que coincide con la IP y puerto origen, pero no con la IP y puerto destino, se descarta directamente. Esta técnica se denomina *NAT estricto*.

Hay dos tipos de NAT, en función de dónde y cuándo se haga la traducción.

Source NAT (SNAT): el origen de los datos está en una red privada. Por tanto, al enviarse se cambia la dirección IP de origen, y la traducción a la correcta (en la respuesta) se hará tras el encaminamiento (*postrouting*).

[2]En realidad, los puertos se encuentran en la cabecera de la capa de transporte; pero tras encapsularlo se puede acceder desde la capa de red.

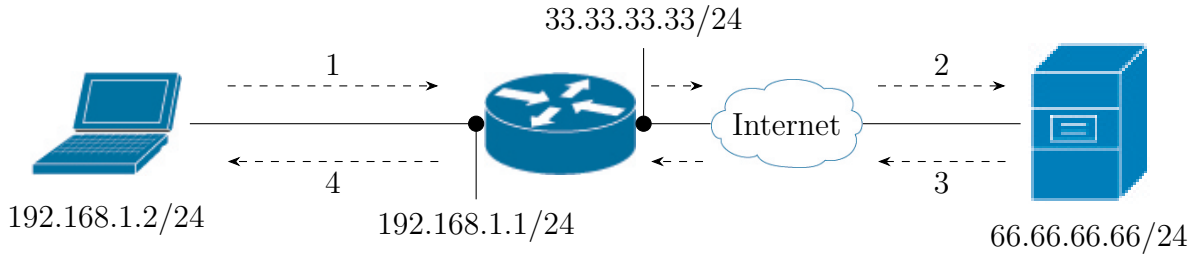


Figura 2.1: Ejemplo de red con SNAT.

Destination NAT (DNAT): el origen de los datos está en la red pública. Por tanto, al recibir los datos se cambia la dirección IP de destino, y la traducción a la correcta (en la respuesta) se hará antes del encaminamiento (*prerouting*).

En este caso la tabla de traducciones del router que realiza DNAT ha de ser estática (la inserción debe ser a mano), ya que en otro caso el router no sabrá a donde redirigir las peticiones entrantes. Este proceso se denomina *port forwarding*.

Planteemos un primer ejemplo de SNAT, que nos ayudará a comprender cómo funciona esta técnica.

Ejemplo. Supongamos la situación de la Figura 2.1, en la que un portátil dentro de una red privada quiere acceder a un servidor HTTP en Internet.

El equipo envía una petición al router (1), que este reenvía al servidor (2). El servidor responde al router (3), que a su vez reenvía la respuesta al equipo (4). Se trata de SNAT, ya que la petición parte de una red privada. Veamos qué ocurre en cada uno de los pasos:

(1) El ordenador envía una petición HTTP al router.

El puerto de origen, el cual asignará alteatoriamente el SO (ya se verá), pongamos que es el 1075. El puerto de destino, en el caso de HTTP, es el 80. Por tanto, la cabecera IP del paquete que envía el portátil al router contendrá:

- IPsrc:sport: `192.168.1.2:1075`.
- IPdest:dport: `66.66.66.66:80`.

(2) El router ha de realizar la traducción de direcciones, ya que la IP del portátil es privada y no puede ser usada en Internet. Para esto, modifica la cabecera IP poniendo como IP origen su propia IP pública, y como puerto origen un puerto que aún no haya sido usado (por ejemplo, 12345). La cabecera IP así:

- IPsrc:sport: `33.33.33.33:12345`.
- IPdest:dport: `66.66.66.66:80`.

La tabla de traducciones del router quedaría (donde notamos con "/" la traducida):

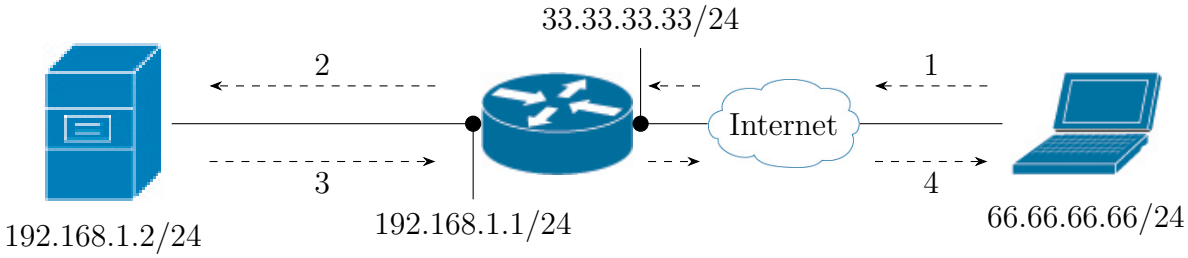


Figura 2.2: Ejemplo de red con DNAT.

IPsrc	sport	IPsrc′	sport′
`192.168.1.2`	`1075`	`33.33.33.33`	`12345`

Tabla 2.1: Tabla de traducciones del router con SNAT.

Tras esta traducción, el router envía el paquete al servidor.

(3) Tras el procesamiento del paquete en el servidor, este envía la respuesta al router. La cabecera IP del paquete de respuesta contendrá:

- IPsrc:sport: `66.66.66.66:80`.
- IPdest:dport: `33.33.33.33:12345`.

(4) El paquete llega sin problema al router, ya que la IP de destino es pública. El router debe realizar de nuevo la traducción para saber a qué equipo de la red privada debe enviar la respuesta. Para ello, consulta la tabla de traducciones (Tabla 2.1) y, tras modificar de nuevo la cabecera IP (*postrouting*), esta queda:

- IPsrc:sport: `66.66.66.66:80`.
- IPdest:dport: `192.168.1.2:1075`.

Planteamos ahora el siguiente ejemplo de DNAT, que nos ayudará ahora a comprender cómo funciona esta técnica.

Ejemplo. Supongamos la situación de la Figura 2.2, en la que un portátil dentro de una red privada quiere acceder a un servidor HTTP en Internet.

El equipo envía una petición al router (1), que este reenvía al servidor (2). El servidor responde al router (3), que a su vez reenvía la respuesta al equipo (4). Se trata de DNAT, ya que la petición parte de la red pública. Veamos qué ocurre en cada uno de los pasos:

(1) El ordenador envía una petición HTTP al router.

El puerto de origen pongamos que es el 1050. El puerto de destino, tras la traducción efectivamente ha de ser el 80 (ya que es un servidor HTTP). No obstante, antes de la traducción este puerto ha de ser el correspondiente al puerto que hayamos asignado al servidor HTTP al que queremos acceder. Por ejemplo, sea la tabla de traducciones del router la de la Tabla 2.2 (que hemos de haber configurado previamente en el *port forwarding*).

IPdest	dport	IPdest′	dport′
33.33.33.33	23456	192.168.1.2	80

Tabla 2.2: Tabla de traducciones del router con DNAT.

En tal caso, el puerto de destino será el 23456. Por tanto, la cabecera IP del paquete que envía el portátil al router contendrá:

- IPsrc:sport: `66.66.66.66:1050`.
- IPdest:dport: `33.33.33.33:23456`.

Notemos que la IP de destino no es el servidor como tal, sino el router (ya que es es al que tiene acceso el portátil), y el puerto de destino es el que hemos asignado en la tabla de traducciones para el servidor HTTP al que queremos acceder.

(2) Tras llegar al router, este ha de realizar la traducción de direcciones (*prerouting*), ya que la IP de destino era el router mismo. Consultando la tabla de traducciones (Tabla 2.2), la cabecera IP del paquete que envía el router al servidor quedará:

- IPsrc:sport: `66.66.66.66:1050`.
- IPdest:dport: `192.168.1.2:80`.

Tras esta traducción, el router envía el paquete al servidor (ya en la red privada).

(3) Tras el procesamiento del paquete en el servidor, este envía la respuesta al router. La cabecera IP del paquete de respuesta contendrá:

- IPsrc:sport: `192.168.1.2:80`.
- IPdest:dport: `66.66.66.66:1050`.

(4) El paquete llega sin problema al router, ya que la IP de destino es pública. El router debe realizar de nuevo la traducción para saber ahora de qué equipo de la red privada proviene la petición. Para ello, consulta de nuevo la tabla de traducciones (Tabla 2.1) y, tras modificar de nuevo la cabecera IP, esta queda:

- IPsrc:sport: `33.33.33.33:23456`.
- IPdest:dport: `66.66.66.66:1050`.

Notemos que, en el SNAT, la tabla de traducciones se va actualizando con cada nueva traducción, mientras que en el DNAT la tabla de traducciones ha de ser estática, ya que en otro caso el router no sabrá a donde redirigir las peticiones entrantes.

2.3.3. Encaminamiento

Se dice del proceso de encontrar el mejor camino para llevar la información (paquetes) de un origen a un destino dado. Como se vió en la Sección 2.2.2, este se realiza salto a salto y paquete a paquete en función de la dirección IP destino del paquete y de las tablas de encaminamiento que hay en cada uno de los routers.

Tablas de encaminamiento

Las tablas de encaminamiento son tablas que se guardan en la memoria de todo equipo conectado a la red (tanto hosts como routers), que informan sobre las redes a las que se puede llegar y la mejor forma de llegar a ellas.

Veamos los campos que tienen estas tablas, donde notaremos entre paréntesis aquellos que tienen menor relevancia y que incluso no son siempre necesarios.

- Red destino: Red a la que pertenecerá la dirección IP de destino, y a la cual queremos llegar.
- Máscara de red: Máscara de red correspondiente a dicha red de destino.
- Siguiente salto: Nodo al que debemos reenviar el paquete para que llegue a la red de destino.
- (Interfaz de salida del equipo), dato que puede ser redundante.
- (Protocolo).
- (Flags).
- (Coste): Coste esperado para llegar a dicha dirección IP de destino. Este se puede medir, por ejemplo, mediante el número de saltos que se han de realizar.

Hay distintos tipos de rutas que se pueden almacenar en una tabla de encaminamiento:

Rutas directas: (marcadas con $*$ en el campo de "Siguiente salto"). Estas son las redes a las que tenemos conexión directa, sin realizar ningún salto. Podemos enviar directamente el paquete al destinatario sin necesidad de pasar por nodos intermedios.

Un router tiene acceso directo a las redes que interconecta (por lo que tendrá una entrada de este tipo por cada red), mientras que un host suele estar en una única red (por lo que tan solo tendrá una entrada de este tipo).

En la mayoría de los casos, cuando se asigna una dirección IP a determinada tarjeta de red de un equipo, se almacena la entrada de esta ruta directa de forma automática.

Rutas indirectas: Estas son las redes a las que no tenemos conexión directa, pero sí a través de un intermediario. Por tanto, es necesario dar mínimo un salto para llegar al destino.

Entrada por defecto: (notado por `0.0.0.0` o `default` en red destino y `/0` en máscara). Hace referencia a cualquier red que no haya sido aceptada por el resto de entradas. El equipo que se encuentra en el campo de "Siguiente salto" será el que nos conexta con el exterior, y lo denominaremos *pasarela* o *gateway*.

Esta entrada no siempre es necesaria, aunque permite que no se produzcan errores (puesto que siempre habrá, al menos, una entrada válida para cada dirección IP, como más adelante veremos).

Entrada de `localhost`: En el caso de que queramos usar esta técnica, también debe haber una entrada en la tabla de encaminamiento con este fin. Su red de destino será `localhost`, y su interfaz, como mencionamos anteriormente, será `lo`.

Tenemos dos tipos de encaminamientos:

Estático: La tabla de encaminamiento se rellena a mano.

Dinámico: La tabla de encaminamiento se rellena de forma automática, ya que hay un protocolo (RIP, OSPF...) que se encarga de actualizarla. Tiene como ventaja que es dinámica (puede cambiar), ya que si se cae cierto router se puede buscar otro camino para llegar al destino.

Observación. En casos muy específicos (menos del 0,1 %), se puede encaminar en función de la dirección IP origen, pero este caso no se desarrollará en la asignatura.

Observación. Como curiosidad, para consultar la tabla de encaminamiento de un equipo con `Linux` se puede emplear el comando `route -n`.

Uso de la tabla de encaminamiento

En esta sección entenderemos cómo funcionan estas tablas. Dada una dirección IP de destino, buscamos saber cuál es la dirección IP del nodo al que debemos enviarle el paquete para que este, finalmente, llegue al destino.

Para esto, buscamos las redes de destino de la tabla de encaminamiento que admitan a la dirección IP de destino. Para ello, se hace la operación lógica `AND` entre la dirección IP de destino y la máscara de cada entrada, y si el resultado coincide con la red de destino entonces dicha entrada es válida para dicha IP.

- Si no hay ninguna entrada válida, se envía un mensaje de error ICMP, pero no se intenta solventar dicho error.
- Si hay más de una entrada válida, se escogerá aquella con la máscara de red más restrictiva, ya que la red de destino será más pequeña, teniendo así (a priori) una conexión más directa. Esto lo veremos en detalle en el próximo ejemplo.

Por tanto, una vez que tenemos la entrada asignada a la dirección IP de destino, se envía el paquete a la dirección IP del siguiente salto, continuando así el encaminamiento hasta llegar al destino.

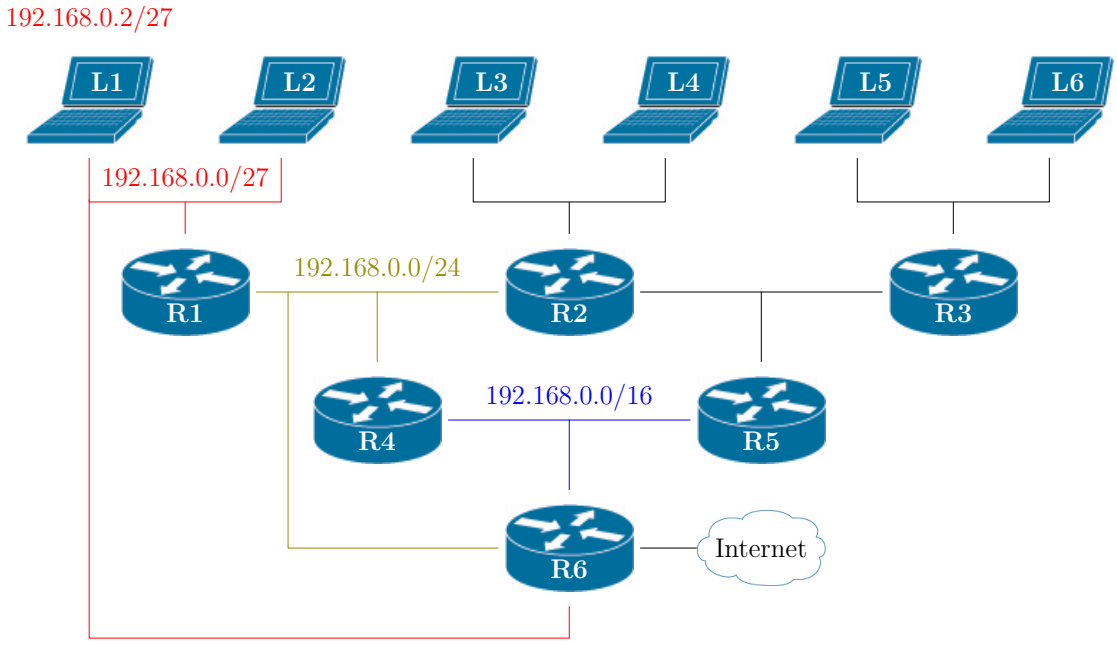


Figura 2.3: Situación para el ejemplo de la página 35.

Red de Destino	Máscara	Siguiente Salto	Interfaz
192.168.0.0	/27	*	-
192.168.0.0	/24	*	-
192.168.0.0	/16	*	-
`default`	/0	`IP_Exterior`	-

Tabla 2.3: Tabla de encaminamiento de R6 para la Figura 2.3.

Ejemplo. Veamos un ejemplo de encaminamiento. Supongamos que estamos en la situación de la Figura 2.3, y que la tabla de encaminamiento de R6 es la que se muestra en la Tabla 2.3 (donde hemos notado por `IP_Exterior` a la IP del siguiente router que nos conecta con Internet).

Supongamos que a R6 le llega un paquete con destino L1 (dirección IP de destino `192.168.0.1`). Tras hacer el `AND` con cada una de las máscaras, vemos que las 4 entradas son válidas para dicha dirección IP. No obstante, la máscara más restrictiva es `/27`, por lo que se elegirá dicha entrada y, por tanto, reenviará el paquete por la interfaz de red de R6 que pertenece a dicha red de destino. Esto permite que el camino se haga directo, con menos saltos.

En la mayoría de los casos, se buscará minimizar las tablas de encaminamiento agrupando las redes con las que se trabaja, permitiendo así que el encaminamiento sea más eficiente. Como mínimo tendremos una entrada por cada interfaz del dispositivo. Además, a menudo tenemos que compartir las tablas de encaminamiento (como veremos en algunos de los siguientes protocolos), y para ello lo mejor es que sean lo más compactas posible.

2.3.4. Protocolos de intercambio de información de encaminamiento

Para facilitar la administración y aumentar la escalabilidad, Internet se jerarquiza en Autonomous System (AS), que son redes muy grandes (en la mayoría de los casos, abarcan todo un país) gestionadas por una única autoridad.

Observación. Cada AS tiene un número único de 32 bits que lo identifica. Por ejemplo, La Red Iris tiene el número `AS766`.

De esta forma, cada AS informará al resto de los demás AS sobre las redes que tienen, de forma que compartirán su información de encaminamiento. Hay dos niveles de intercambio de tablas de encaminamiento:

- Algoritmos Interior Gateway Protocol (IGP): protocolos de intercambio de información de encaminamiento dentro de un mismo AS. Cada autoridad tiene libertad de elección, y los más comunes son RIP, OSPF....
- Algoritmos Exterior Gateway Protocol (EGP): protocolo de intercambio de información de encaminamiento entre distintos AS. Al ser estos distintas, se usa un único protocolo, BGP.

Routing Information Protocol (RIP)

Aunque es una funcionalidad de la capa de red, se implementa sobre la capa de aplicación (opera sobre UDP en el puerto 520), ya que la funcionalidad y la implementación son independientes.

Es un protocolo que adopta un algoritmo vector-distancia; es decir, se basa exclusivamente en el número de saltos, ignorando la velocidad de cada una de las conexiones. Una vez que un router aprende un camino para llegar a cierta red, no aprende otro a no ser que el número de saltos sea menor.

Cuando un router RIP se enciende y es configurado, envía cada cierto tiempo (por defecto 30 s) a todos los routers de sus respectivas redes un mensaje en el que informa de las redes a las que sabe llegar, junto con el coste que le supone para cada una de ellas. Además, recibirá de forma periódica la información correspondiente de los demás routers. Cuando un router recibe información de un vecino, si encuentra una ruta que no conocía, la añade a su tabla de encaminamiento, con el coste que le ha anunciado dicho vecino más 1 (el salto al correspondiente vecino). En el resto de casos, tan solo si el coste es menor que el que ya tenía, se actualiza la entrada.

Toda esta información se comparte por la dirección multicast `224.0.0.9`, en la que escuchan todos los routers que soportan RIP.

Problema de la cuenta al infinito

Un problema que puede surgir al emplear RIP es la convergencia lenta, ya que las malas noticias tardan en propagarse. Puede ocurrir que algún camino se rompa y, dada la naturaleza del protocolo, esta información tarda en notificarse.

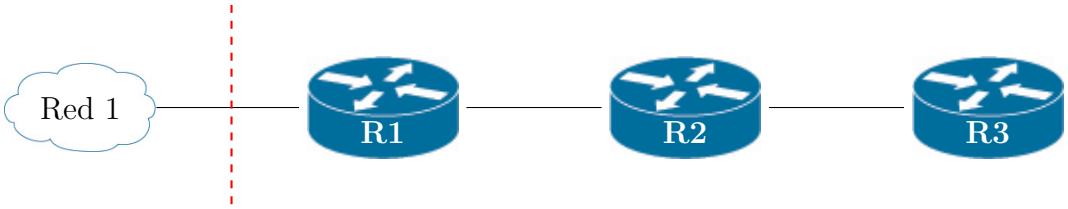


Figura 2.4: Problema de la cuenta al infinito en RIP.

Este se maximiza en el conocido "problema de la cuenta al infinito", que se muestra en la Figura 2.4. En dicha figura podemos ver que, inicialmente, R1 tenía acceso a la red 1, aunque posteriormente dicho camino se rompe y R1 es notificado de que ha perdido acceso a dicha red. No obstante, R2, que aún no ha sido notificado de que ya no puede llegar a dicha red, al compartir a R1 su tabla de encaminamiento le informa de que él sí sabe llegar a la red 1, por lo que R1 lo aprende (aumentando en una unidad el coste), y así sucesivamente. Cuando R2 reciba la información del corte, aprenderá el camino por R1, y así sucesivamente. Esto podría llegar a repetirse indefinidamente (de ahí del nombre) hasta 16 (ya que ese es el límite para RIP en el que se asume que no se sabe llegar a dicha red), sin que ninguno en realidad supiese llegar a dicha red. Veamos algunas posibles soluciones:

Split horizon: Se basa en que a un router se le prohíbe compartir una ruta por la misma interfaz por la que la aprendió en primer lugar.

De esta forma, R2 no podría enseñarle a R1 cómo llegar a la red 1, por lo que el problema no empieza a sucederse.

Hold down: Retrasa los mensajes que nos llegan de una dirección que ya conocemos 180 segundos, esperando a que nos respondan los anteriores, si siguen activos.

Poison reverse: Si no sabemos llegar a un destino, informamos de que nuestro coste es infinito (coste 16).

Open Shortest Path First (OSPF)

A diferencia de RIP (que siempre consideraba el coste como el número de saltos), OSPF es un protocolo que permite al administrador definir el coste en función de distintos aspectos (velocidad, latencia, etc.). Como criterio por defecto, el coste de un enlace es el inverso del ancho de banda (la velocidad) de dicho enlace. Este protocolo busca el camino global que minimiza la suma de todos los costes, usando para ello el algoritmo de Dijkstra[3].

Permite definir áreas, de forma que la difusión se hace en unas áreas concretas. Esto hace que sea mucho más escalable, al contrario que RIP.

Los mensajes que se envían entre los routers que usan OSPF son:

[3]Trata en la asignatura de Algorítmica

0-3	4-7	8-15	16-18	19-31
V	LC	TS	Longitud Total	
Identificador			I	Desplazamiento
TTL		Protocolo	Checksum	
Dirección IP de Origen				
Dirección IP de Destino				
Opciones				Relleno

Tabla 2.4: Cabecera IP.

- `Hello`: mensaje empleado para establecer la conexión, en el que se avisa a los routers que se van a comunicar.
- `Database description`: mensaje empleado para informar sobre la topología de las redes que conocemos.
- `Link status request/update/ack`: mensajes enviados para consultar, actualizar, o confirmar cambios.

2.3.5. Cabecera IP

En la presente sección profundizaremos en la cabecera IP, junto con sus campos. Esta se puede ver en la Tabla 2.4. Como vemos, está organizada en palabras de 32 bits (4 Bytes), y como mínimo ocupa 20 Bytes (ya que el campo *Opciones* es opcional, y el de *relleno* se emplea para que sean múltiplos de 32 bits).

Veamos los campos, en orden, que la componen:

Versión: (4 bits) Indica la versión de IP que se está utilizando. Contiene `0100` si es IPv4 y `0110` si es IPv6.

Tamaño de cabecera: (4 bits) Indica el tamaño de la cabecera en palabras de 32 bits. Su valor mínimo es de 5 palabras (20 Bytes) y su máximo de 15 palabras (60 Bytes).

Tipo de Servicio: (8 bits) Indica la calidad de servicio deseada durante el tránsito del paquete por una red. Algunas redes ofrecen prioridades de servicios, considerando determinados tipos de paquetes más prioritarios que otros (especialmente cuando la carga de la red es alta).

Longitud total: (16 bits) Indica el tamaño total, en bytes (también llamados octetos), del datagrama, incluyendo el tamaño de la cabecera y el de los datos.

Identificador: (16 bits) Identificador único del datagrama. Se utiliza en caso de que el datagrama deba ser fragmentado, para poder distinguir los fragmentos de un datagrama de los de otro. Resaltamos por tanto que todos los fragmentos de un mismo datagrama tienen el mismo identificador.

Indicadores: (3 bits) En la actualidad se utiliza para especificar valores relativos a la fragmentación. Los tres bits (por orden de mayor a menor peso) son:

0	DF	MF

donde:

- Bit 0: Reservado, debe ser 0.
- Bit 1 (Don't Fragment (DF)): indica si el datagrama puede ser fragmentado (0) o no (1). Si un paquete necesita ser fragmentado para enviarse y este bit es 1, se descartará.
- Bit 2 (More Fragments (MF)): indica si el fragmento es el último (0) o si le siguen más fragmentos (1).

Desplazamiento: (13 bits) En paquetes fragmentados, indica la posición, en unidades de 64 b = 8 B, que ocupa dentro del datagrama original.

Time To Live (TTL): (8 bits) Indica el número de saltos máximo de un paquete en una red para evitar que los paquetes naveguen en la red indefinidamente. En cada salto, el campo se reduce en una unidad; y si este campo llega a 0, el paquete se descarta.

Protocolo: (8 bits) Valor numérico que indica el protocolo de las capas superiores al que debe entregarse el paquete[4].

Checksum: (16 bits) Es una comprobación de la corrección del datagrama. Se recalcula cada vez que algún nodo cambia alguno de sus campos (como el TTL).

El método de cálculo consiste en sumar en complemento a 1 cada palabra de 16 bits de la cabecera (considerando como 0 el campo del *checksum*) y hacer el complemento a 1 del valor resultante. Así, cuando llega al destino, se hace esta misma operación y se comprueba si es correcta la cabecera.

Dirección IP de origen: (32 bits) Dirección IP del emisor del paquete.

Dirección IP de destino: (32 bits) Dirección IP del destino del paquete.

Opciones: (opcional) Campo que puede contener información adicional, como la ruta que ha seguido el paquete.

Relleno: Este campo tiene tantos bits como sean necesarios para que la cabecera tenga un tamaño múltiplo de 32 b = 4 B.

2.3.6. Fragmentación

Como hemos visto en el aprtado anterior, el tamaño máximo de un paquete que se envíe usando el protocolo IP es de $2^{16} - 1$ Bytes, aunque este es un valor teórico que ninguna red suele aceptar. Dentro de una red, cada tarjeta de red tiene un Maximum Transfer Unit (MTU), un valor numérico que indica el tamaño máximo de un paquete que se pasar a la capa de enlace para ser enviado por la red.

Como hemos mencionado, el MTU depende del estándar de cada tarjeta de red. Algunos ejemplos son:

[4]Estos valores se pueden consultar aquí.

Figura 2.5: Red para el ejemplo de la Página 40.

- Ethernet: 1500 Bytes.
- Wifi: Aunque el valor es mayor, normalmente el SAP lo restringe a 1500 Bytes.

Notemos que, en la cabecera IP vista en la sección anterior, los campos de *identificación*, *desplazamiento* e *Indicadores* son los que se usan para controlar la fragmentación.

Algunas observaciones importantes son:

- Si hay algún error y no llegan todos los fragmentos de un datagrama se descarta todo y debe ser una capa superior la que se encargue de arreglar el problema.
- Un datagrama solo se fragmentará cuando vaya a pasar por una tarjeta de red con un MTU menor que el tamaño del paquete. Esta fragmentación se hará cuando sea necesaria, y puede ser en cualquier nodo del encaminamiento.
- Los datagramas tan solo se podrán ensamblar en el destino, ya que distintos fragmentos podrán seguir caminos distintos, dependiendo del encaminamiento.

Observación. Es común hablar del MTU de una red. Esto se dirá cuando las tarjetas de red de dicha red tengan el mismo valor MTU.

Ejemplo. Supongamos que queremos enviar un datagrama con 4180 B de datos desde la red A a la red B, según el diagrama de la Figura 2.5. Supongamos que su identificador es X. Veamos si se produce fragmentación, y en qué paquetes se fragmenta.

Observación. Recordemos que el campo de *Fragmentación* mide en unidades de 8 B, por lo que si un paquete se fragmenta, el tamaño de los fragmentos ha de ser múltiplo de 8 B. No obstante, por simplificación en el ejemplo, supondremos que se mide en unidades de 1 B (aunque, reiteramos, no es lo que ocurriría en la realidad).

Como la cabecera ocupa 20 Bytes, el tamaño a enviar es de 4200 B. Este no se podrá enviar por la red que une R1 y R2, pues su MTU es de 1500 B. Por tanto, se fragmentará en R1. Veamos cada uno de los paquetes, teniendo en cuenta que de los 1500 B de límite, 20 han de ser para la cabecera:

1. Paquete 1. Faltan por enviar 4180 B de datos, por lo que se envían:

20	1480

La cabecera IP de este paquete tendrá los campos:

$$\text{Identificador} = X, \qquad \text{MF} = 1, \qquad \text{Desplazamiento} = 0$$

2. Paquete 2. Faltan por enviar 4180 B − 1480 B = 2700 B de datos, por lo que se envían:

20	1480

La cabecera IP de este paquete tendrá los campos:

$$\text{Identificador} = X, \qquad \text{MF} = 1, \qquad \text{Desplazamiento} = 1480$$

3. Paquete 3. Faltan por enviar 2700 B − 1480 B = 1220 B de datos, por lo que se envían:

20	1220

La cabecera IP de este paquete tendrá los campos:

$$\text{Identificador} = X, \qquad \text{MF} = 0, \qquad \text{Desplazamiento} = 2960$$

Cuando cada uno de estos paquetes llegue a R2, como el MTU de la red que lo conecta con R3 es de 1000 B y todos los paquetes tienen un tamaño mayor, se fragmentarán en R2. Veamos cada uno de los paquetes, teniendo en cuenta que de los 1000 B de límite, 20 han de ser para la cabecera:

1. Paquete 1.1. Faltan por enviar 1480 B de datos, por lo que se envían:

20	980

La cabecera IP de este paquete tendrá los campos:

$$\text{Identificador} = X, \qquad \text{MF} = 1, \qquad \text{Desplazamiento} = 0$$

2. Paquete 1.2. Faltan por enviar 1480 B − 980 B = 500 B de datos, por lo que se envían:

20	500

La cabecera IP de este paquete tendrá los campos:

$$\text{Identificador} = X, \qquad \text{MF} = 1, \qquad \text{Desplazamiento} = 980$$

3. Paquete 2.1. Faltan por enviar 1480 B de datos, por lo que se envían:

20	980

La cabecera IP de este paquete tendrá los campos:

$$\text{Identificador} = X, \qquad \text{MF} = 1, \qquad \text{Desplazamiento} = 1480$$

4. Paquete 2.2. Faltan por enviar 1480 B − 980 B = 500 B de datos, por lo que se envían:

20	500

La cabecera IP de este paquete tendrá los campos:

$$\text{Identificador} = X, \qquad \text{MF} = 1, \qquad \text{Desplazamiento} = 2460$$

Figura 2.6: Red para el funcionamiento de ARP.

5. Paquete 3.1. Faltan por enviar 1220 B de datos, por lo que se envían:

20	980

La cabecera IP de este paquete tendrá los campos:

$$\text{Identificador} = X, \qquad \text{MF} = 1, \qquad \text{Desplazamiento} = 2960$$

6. Paquete 3.2. Faltan por enviar $1220\,\text{B} - 980\,\text{B} = 240\,\text{B}$ de datos, por lo que se envían:

20	240

La cabecera IP de este paquete tendrá los campos:

$$\text{Identificador} = X, \qquad \text{MF} = 0, \qquad \text{Desplazamiento} = 3940$$

Cuando cada uno de estos paquetes llegue a R3, este los reenviará a la IP de destino, dentro de la red B. Allí, se ensamblarán de vuelta los paquetes, obteniendo así el datagrama original.

2.4. Asociación con la capa de enlace: Address Resolution Protocol (ARP)

Cuando queremos enviar un datagrama desde un origen a un destino, usando la tabla de encaminamiento del nodo en cuestión podemos saber la dirección IP del siguiente salto. No obstante, al bajar a la capa de enlace, ya no contemplamos direcciones IP, ya que el direccionamiento en esta capa se hace mediante direcciones MAC. Además, debido a que el encaminamiento en la capa de enlace se hace punto a punto, las direcciones MAC de origen y de destino cambian en cada salto.

Como hemos visto, sabemos las dirección IP de origen y la del siguiente salto, pero no la dirección MAC del siguiente salto. Esto nos lo proporcionará el protocolo ARP, que es un protocolo de la capa de enlace que se encarga de resolver direcciones MAC a partir de direcciones IP.

Funcionamiento

Supongamos la situación de la Figura 2.6, en la que PC1 quiere mandar un datagrama a PC2 (notemos que no consideramos intermedirarios, puesto que en el nivel de enlace el encaminamiento se hace punto a punto). El nodo PC1 conoce de sí mismo tanto su dirección IP como su dirección MAC, mientras que del siguiente salto

0-7	8-15	16-23	24-31
Htipo		Ptipo	
Hlen	Plen	Operacion	
Hemisor (Bytes 0-3)			
Hemisor (Bytes 4-5)		Pemisor (Bytes 0-1)	
Pemisor (Bytes 2-3)		Hsol (Bytes 0-1)	
Hsol (Bytes 2-5)			
Psol (Bytes 0-3)			

Tabla 2.5: Cabecera ARP.

(PC2) tan solo conoce la dirección IP tras haber consultado la tabla de encaminamiento. Para poder enviarle la trama[5] a PC2, necesita saber la dirección MAC de PC2. Esto nos lo proporciona el protocolo ARP, que funciona de la siguiente forma:

- PC1 manda una petición `ARP Request` a nivel de enlace por la dirección `FF:FF:FF:FF:FF:FF` (la dirección de difusión a nivel de enlace), preguntando por la dirección MAC de la dirección IP de PC2.
- PC2, que habrá recibido dicha petición, identifica que la dirección IP de la petición es la suya, por lo que contesta con un mensaje `ARP Reply` con su dirección MAC en unicast a PC1 (cuya dirección MAC ya conoce).

Este proceso no se hace (pues introduciría mucho tráfico) cada vez que se quiera mandar una trama, sino que las direcciones MAC que recibimos se van guardando en una caché y, tras cierto tiempo, expiran.

Observación. Como curiosidad, para consultar dicha caché en un equipo con `Linux` se puede emplear el comando `arp -a`.

2.4.1. Cabecera ARP

La cabecera de una trama ARP se muestra en la Tabla 2.5, donde notemos que "H" indica "Hardware" (capa de enlace) y "P" indica "Protocol" (capa de red). Los campos de dicha cabecera son:

Htipo: (2 Bytes) Número que indica el protocolo que se usa en el nivel de enlace (por ejemplo, Ethernet es `1`).

Ptipo: (2 Bytes) Número que indica el protocolo que se usa en el nivel de red (por ejemplo, IP es `0x0800`).

Hlen: (1 Byte) Número que indica la longitud de la dirección hardware (en Bytes). Para direcciones MAC es 6.

Plen: (1 Byte) Número que indica la longitud de la dirección del protocolo de red (en Bytes). Para direcciones IPv4 es 4.

[5]El datagrama ya se ha encapsulado en una trama, ya que en el nivel de enlace el PDU se denomina trama.

0-7	8-15	16-31
Tipo	Código	Comprobación

Tabla 2.6: Cabecera ICMP.

Operación: (2 Bytes) Número que indica si es una petición o una respuesta. El valor `1` indica `Request` y el valor `2` indica `Reply`.

Hemisor: (6 Bytes) Dirección hardware (normalmente MAC) del emisor.

Pemisor: (4 Bytes) Dirección de red (normalmente IP) del emisor.

Hsol: (6 Bytes) Dirección hardware (normalmente MAC) del receptor.

Psol: (4 Bytes) Dirección de red (normalmente IP) del receptor.

Notemos por tanto que, en una petición, el campo de Hsol contendrá la direción MAC de difusión, como hemos mencionado anteriormente.

Por último, destacar que el protocolo ARP tiene su homólogo Reverse ARP (RARP) que hace lo contrario, es decir, dado una dirección MAC nos devuelve su dirección IP. Cuando Internet comenzó, había equipos sencilos con pocas características, incluso sin disco duro. Al no tener disco duro, estos no podían almacenar su dirección IP, aunque las tarjetas de red si guardaban su dirección MAC. Era por tanto necesario un protocolo que nos diera la dirección IP a partir de la dirección MAC, y así nació el protocolo RARP. En la actualidad se encuentra en desuso, pero fue el precursor de BOOTP, el cual más tarde fue sustituido por DHCP.

2.5. El protocolo ICMP

El Internet Control Message Protocol (ICMP) es un protocolo que, aunque no es imprescindible, es de gran ayuda. En general, sirve para informar al origen de que ha habido un error. Este protocolo es útil pues, aunque IP no arregla ningún tipo de problema, este protocolo informa para que las capas superiores decidan qué hacer. Este es un protocolo de nivel de red que se encapsula también en el nivel de red, en un datagrama IP.

2.5.1. Paquete ICMP

La cabecera, que se muestra en la Tabla 2.6, se compone de 32 bits. Veamos cada uno de los campos que la componen:

Tipo: (1 Byte) Indica el tipo de mensaje que se está enviando. Los tipos más comunes se muestran en la Tabla 2.7.

Código: (1 Byte) Para cada tipo de mensaje, indica el subtipo que se está enviando, para detallarlo aún más.

Tipo	Descripción
8/0	Solicitud/Respuesta "echo" (`ping`)
3	Destino inalcanzable
4	Ralentización del origen
5	Redireccionamiento
11	TTL excedido
12	Problema de parámetros
13/14	Solicitud/Respuesta de sello de tiempo
17/18	Solicitud/Respuesta de máscara de red

Tabla 2.7: Tipos de mensajes ICMP.

Comprobación: (2 Bytes) Es un campo de checksum de la cabecera.

Respecto a la parte de datos, este paquete contiene los primeros 64 bytes del paquete que provocó el error. Es decir (suponiendo que la cabecera IP no tiene campo de "Opciones"), contiene los 20 Bytes de la cabecera IP y los 44 Bytes de datos del paquete IP que provocó el error. De esta forma, cuando el origen reciba este paquete podrá encontrar información sobre el paquete que provocó el error.

No debemos olvidar que ICMP se encapsula sobre el protocolo IP. Por tanto, el datagrama que se envía por la red (que es correcto) contiene una cabecera IP y en el SDU contiene el paquete ICMP, que contiene su respectiva cabecera y, en la parte de datos, la cabecera IP y los primeros 44 Bytes de datos del paquete que provocó el error. Por tanto, en el mismo datagrama se enviará tanto la cabecera del paquete IP que provocó el error como la cabecera IP que se envía encapsulando el mensaje ICMP que informa del error.

Como aspectos relevantes, además de informar de errores nos permite conocer la situación de la red usando el comando `ping`, que envía un paquete ICMP de tipo "echo" (tipo 8) y espera una respuesta de tipo "echo" (tipo 0). Además, nos permite conocer la ruta que sigue un paquete usando el comando `traceroute`, que envía paquetes ICMP de tipo "echo" (tipo 8) con un TTL creciente (1,2,...). De esta forma, en cada salto, un paquete excederá su TTL, por lo que se enviará un mensaje ICMP de tipo "TTL excedido" (tipo 11) que informará de que el paquete no ha podido llegar a su destino. Esto nos permitirá saber todos los nodos que se encuentran en la ruta que sigue el paquete.

2.6. Autoconfiguración de la capa de red (DHCP)

El Dynamic Host Configuration Protocol (DHCP) es un protocolo para configurar de forma automática la capa de red. Este protocolo se encarga de asignar direcciones IP, máscaras, pasarelas por defecto e IP del servidor DNS. Su funcionalidad es a nivel de red, aunque se implementa en capa de aplicación y se encapsula en UDP (puerto 67 para el servidor, 68 para el cliente).

Contamos con un cliente, que inicialmente no tiene dirección IP asignada (emplearemos la `0.0.0.0`), y un servidor DHCP, el cual se encargará de asignársela. Se trata de un protocolo de *leasing* (alquiler), ya que la dirección IP que se asigna al cliente es válida durante un tiempo.

Para conseguir una dirección IP, se intercambian los siguientes mensajes entre el cliente y el servidor DHCP (donde cada par pregunta-respuesta se etiqueta con un identificador de transacción para que el cliente sepa que el mensaje va para él):

- `DHCP Discover`: El cliente envía un mensaje para que saber si hay algún servidor DHCP en la red. Lo envía por tanto a la dirección de difusión.

$$\begin{aligned}\text{Dirección IP origen} &= \texttt{0.0.0.0}\\ \text{Dirección IP destino} &= \texttt{255.255.255.255}\\ \text{ID transacción} &= X\end{aligned}$$

- `DHCP Offer`: El servidor responde identificándose y proponiéndole una dirección IP al cliente, que será válida durante cierto tiempo (*lease time*), el cual se configura en el servidor DHCP. Notemos que esto es solo una oferta, no una imperativa.

  Como el cliente aún no tiene dirección IP asignada, se envía de nuevo a la dirección de difusión.

$$\begin{aligned}\text{Dirección IP origen} &= \text{Dirección IP del servidor}\\ \text{Dirección IP destino} &= \texttt{255.255.255.255}\\ \text{ID transacción} &= X\\ \text{Dirección IP ofrecida} &= Y\\ \text{Lease time} &= Z\,\text{s}\end{aligned}$$

- `DHCP Request`: El cliente le solicita al servidor la dirección IP que le ha ofrecido el servidor (o la misma que ya estaba usando, en el caso de que se trate de una renovación).

  Aunque ya no es necesario que se envíe a la dirección de difusión (ya que el cliente conoce la dirección IP del servidor DHCP que le ha hecho la oferta), el estándar establece que se envíe aun así a la dirección de difusión (aunque permite ambas formas).

$$\begin{aligned}\text{Dirección IP origen} &= \texttt{0.0.0.0}\\ \text{Dirección IP destino} &= \texttt{255.255.255.255}\\ \text{ID transacción} &= X'\\ \text{Dirección IP solicitada} &= Y\\ \text{Lease time} &= Z\,\text{s}\end{aligned}$$

- `DHCP ACK`: El servidor responde con la dirección IP del cliente definitiva, y esta sí es imperativa.

De nuevo, en este caso el servidor se ve obligado a enviar a la dirección de difusión.

$$\begin{aligned} \text{Dirección IP origen} &= \text{Dirección IP del servidor} \\ \text{Dirección IP destino} &= \texttt{255.255.255.255} \\ \text{ID transacción} &= X' \\ \text{Dirección IP asignada} &= Y \\ \text{Lease time} &= Z\,\text{s} \end{aligned}$$

Como hemos mencionado, la IP de destino en toda la transacción es la de difusión, y la de origen es la `0.0.0.0` en caso de ser el cliente el origen, o la del servidor DHCP en caso de ser él el origen.

Por último, para liberar la direcciín IP de forma correcta (sin que surja ningún error), el cliente debe enviar el mensaje `DHCP Release` al servidor DHCP antes de que termine el tiempo de alquiler. En caso contrario, el servidor liberará la IP igualmente, ya que es posible que el cliente ya no la esté usando y sea necesario liberarla para no quedarnos sin direcciones IPs disponibles. En el caso de que el cliente busque renovar su alquiler, volverá a enviará un mensaje `DHCP Request` al servidor DHCP.

Observación. Es posible configurar un servidor para que algunas IPs fijas se asignen a ciertos dispositivos.

3. Capa de transporte

En el presente tema, estudiaremos a fondo la capa de transporte. Recordemos que seguimos el Modelo TCP/IP descrito en la Tabla 1.1.

Objetivos

- Comprender las funiconalidades y servicios de la capa de transporte.
  - Servicio de **multiplexación/demultiplexación**.
  - Servicio **orientado a conexión** frente a **no orientado a conexión**.
  - Cómo conseguir una transferencia de datos **fiable**.
  - Cómo proporcionar **control de flujo**.
  - Cómo proporcionar **control de congestión**.
  - Cómo se han implementado estas funcionalidades en Internet.

3.1. Introducción

Tanto la capa de red como la de enlace realizan encaminamiento punto a punto, pero la capa de transporte en cambio se encarga de la comunicación extremo a extremo. Por tanto, solo los dispositivos extremos (normalmente los hosts) cuentan con la capacidad de procesamiento a nivel de transporte.

Como vimos en el primer tema, el PDU de la capa de transporte se denomina "datagrama" si se usa el protocolo UDP y "segmento" si se trata de TCP. El SDU de esta capa, como es lógico pensar, es el PDU de la capa de aplicación.

Como principales funcionalidades comunes a cualquier protocolo de transporte se encuentrn la comunicación extremo a extremo y la multiplexación/demultiplexación de aplicaciones. Esta segunda consiste en permitir la entrada desde el origen de muchos paquetes (multiplexación), y que al llegar al destino se distribuyan adecuadamente a los procesos indicados (demultiplexación). Esto se logra mediante los puertos, concepto que introducimos a continuación.

Definición 3.1 (Puertos)**.** Un puerto es un número natural que es usado por el Sistema Operativo para saber a qué proceso está destinado cada paquete que llegue por la red. Es decir, cuando un paquete llegue por la red a un equipo, el sistema operativo consultará el puerto para saber a qué proceso corresponde.

Los puertos que están por debajo de 1024 están reservados, y el Sistema Operativo los ofrece a los usuarios con privilegios de administrador. Por encima de este número, los puertos están a disposición del desarrollador y no se necesitan permisos de administrador para utilizarlos.

Por último, cabe mencionar que los puertos de distintos protocolos son independientes. Esto permite que dos paquetes que se han recibido usando distintos protocolos pero que tengan el mismo número de puerto correspondan, sin problema alguno, a procesos distintos.

Observación. Como curiosidad, en sistemas Linux el archivo `/etc/services` muestra los puertos activos, donde además se muestra el protocolo al que pertenecen.

Adicionalmente, cada protocolo de transporte puede implementar funcionalidades adicionales. Los principales protocolos presentes en la actualidad son UDP y TCP, aunque existen otros como SCTP, que trata de ofrecer una mezcla entre los dos anteriores. En la asignatura veremos los dos primeros, cuyas descripciones breves se presentan a continuación.

UDP
: Es un servicio no orientado a conexión, y por tanto no fiable. Su intención no es esta, sino la de ofrecer una comunicación rápida.

TCP
: Es un servicio fiable, y por tanto orientado a conexión. Como funcionalidades adicionales, ofrece control de conexión, de errores, de flujo y de congestión. Respecto a los errores, si la red es cableada se asume que son por congestión (pues la tasa de fallo es de 1 entre 1 millón si el medio es cableado), mientras que si es inalámbrica se asume que puede ser por otros motivos (pues la tasa de fallo es del 10 %).

3.2. User Datagram Protocol (UDP)

El protocolo UDP es un protocolo de la capa de transporte encapsulado sobre IP. Este, al igual que IP, es un protocolo de máximo esfuerzo, ya que intenta hacerlo lo mejor posible pero no se encarga de corregir errores. Esto se debe a que algunas de sus características son las siguientes:

- Servicio no orientado a conexión.

  No hay *hand-shaking*, lo que permite que no haya retardo de establecimiento (aumentando la velocidad de la comunicación). Además, cada TPDU es totalmente independiente.

- Servicio no fiable.

  Puede haber pérdidas, por ejemplo debido a que nunca se produzca la conexión. No obstante, esto no significa que no puedan haber aplicaciones fiables sobre UDP, sino que es la capa de aplicación la que debe encargarse de corregir los errores.

0-7	8-15	16-23	24-31
Puerto Origen		Puerto Destino	
Longitud UDP		Checksum	

Pseudocabecera		
IP Origen		
IP Destino		
0 . . . 0	Protocolo	Longitud UDP

Tabla 3.1: Cabecera UDP.

- No hay garantías de entrega ordenada.

  Como el lector conoce, los paquetes en la capa de red pueden fragmentarse y llegar desordenados. El protocolo UDP no hace nada para ordenarlo, por lo que debe ser la capa de aplicación la que se encargue de ello.

- No hay control de congestión.

  Debido a que se busca que la entrega de los datos a las capas adyacentes sean lo más rápida posible, no se implementa control de congestión.

Por tanto, las únicas funcionalidades de este protocolo son las comunes a cualquier protocolo de la capa de transporte, la conexión extremo a extremo y la multiplexación/demultiplexación.

Sus principales usos en la actualidad son las aplicaciones multimedia que son tolerantes a fallos y sensibles a retardos, como la voz sobre IP o el streaming.

3.2.1. Cabecera UDP

En la presente sección profundizaremos en la cabecera UDP, junto con sus campos. Esta se puede ver en la Tabla 3.1. Como vemos, está organizada en palabras de 32 bits (4 Bytes), y ocupa 8 Bytes.

Veamos los campos, en orden, que la componen:

Puerto origen: (16 bits) Puerto en el que escucha el emisor.

Puerto destino: (16 bits) Puerto al que está destinado el mensaje, donde se supone que escucha el receptor (aunque no se garantiza pues es SNOC).

Longitud UDP: (16 bits): en bytes, mide la lontidud total del PDU (cabecera, junto con datos).

Checksum: (16 bits) suma de comprobación calculada mediante el mismo algoritmo que en el caso de la cabecera IP. Comprueba la cabecera UDP y una pseudocabecera, que consiste en una parte relevante de la cabecera IP con datos relevantes para UDP. Estos son:

- Dirección IP de origen.
- Dirección IP de destino.

Puerto	Aplicación/Servicio	Descripción
7	"Echo"	"Echo"
13	"Daytime"	Fecha y Hora
37	"Time"	Fecha y Hora
42	"Nameserver"	Servidor de nombres
53	"Domain" o DNS	Servicio de nombres de dominio
69	TFTP	Transferencia simple de archivos
123	NTP	Protocolo de tiempo de red

Tabla 3.2: Puertos preasignados en UDP.

- Protocolo empleado (en la capa de red).
- Longitud UDP.

3.2.2. Multiplexación/demultiplexación

Como se mencionó anteriormente, los puertos con un número menor a 1024 están preasignados con servicios normalizados, y tan solo se podrán usar con privilegios de administrados. En el caso de UDP, algunos se muestran en la Tabla 3.2.

3.3. Transmission Control Protocol (TCP)

El protocolo TCP es un protocolo de la capa de transporte encapsulado sobre IP. Algunas de sus características son las siguientes:

- Servicio orientado a conexión:

  Emplea un protocolo[1] de establecimiento de conexión (*handshake*). Es por tanto un servicio punto a punto, y exige un estado común entre el emisor y el receptor.

- Es punto a punto (al contrario que UDP). Por tanto, si se quiere enviar un mensaje mediante la dirección de difusión, se deberá emplear UDP.

- La entrega de los datos a la capa de aplicación es ordenada; aunque no necesariamente lo es la llegada de los datos. Este protocolo se encarga de ordenarlos.

- Es transmisión full-duplex.

- Tiene un mecanismo de detección y recuperación de errores, retransmitiendo si es necesario. Emplea para ello confirmaciones (ACK).

- Es un servicio fiable, ya que tiene mecanismos de control de flujo y de congestión.

- Usa "piggybacking", concepto que consiste en que, al mandar una confirmación (ACK) se aprovecha y se envían más datos.

[1] Aunque se traduce como protocolo, es más bien un intercambio muy específico de mensajes.

0-3	4-9	10-15	16-31
Puerto Origen			Puerto Destino
Número de Secuencia			
Número de Acuse de Recibo			
Hlen	Reservado	Flags	Ventana del Receptor
Checksum			Puntero de Datos Urgentes
Opciones			

Pseudocabecera		
0-7	**8-15**	**16-31**
IP Origen		
IP Destino		
0000 0000	Protocolo	Longitud UDP

Tabla 3.3: Cabecera TCP.

Maximum Segment Size (MSS)

En el Capítulo anterior, vimos que en la capa de enlace existe un concepto denominado MTU que indica el valor máximo de su PDU. De igual forma, en la capa de transporte existe un valor denominado MSS que indica el tamaño máximo de los datos que se pueden enviar en un segmento. Este, viene determinado por el MTU, y se calcula como sigue:

$$\text{MSS} = \text{MTU} - \text{Longitud de cabecera IP} - \text{Longitud de cabecera TCP}$$

3.3.1. Cabecera TCP

En la presente sección profundizaremos en la cabecera TCP, junto con sus campos. Esta se puede ver en la Tabla 3.3. Como vemos, está organizada en palabras de 32 bits (4 Bytes), y como mínimo ocupa 20 Bytes (ya que el campo *Opciones* es opcional). Veamos los campos, en orden, que la componen:

Puerto origen: (16 bits) identifica el puerto del emisor.

Puerto destino: (16 bits) identifica el puerto del receptor.

Número de secuencia: (32 bits) identifica el byte del flujo de datos enviados por el emisor al receptor que representa el "offset" del segmento. Se desarrollará más adelante.

Número de acuse de recibo: (32 bits) el valor del siguiente número de secuencia que el receptor del segmento espera recibir. De esta forma se confirma todo lo anterior también. Se desarrollará más adelante.

Observación. Para simplificar diagramas para ilustrar ciertas situaciones, este se notará como ack = Número de acuse, pues como hemos indicado al actualizar el número de acuse de recibo se confirma todo lo anterior. No obstante, este campo no debe confundirse con el bit de ACK, que se desarrollará más adelante.

Longitud de cabecera: (4 bits) indica el tamaño de la cabecera en palabras de 32 bits (4 Bytes). Sin campos opcionales, es de 5, y como máximo es de 15, que equivale a 60 bytes.

Reservado: (6 bits) bits reservados para un posible uso futuro. Deben estar a 0.

Flags: (6 bits) Los bits (por orden de mayor a menor peso) son:

U	A	P	R	S	F

U: Urgente. De normal los datos se van introduciendo en el buffer del receptor por la derecha y sacando por la izquierda, tratándose de un buffer circular con ventana deslizante. No obstante, hay datos urgentes que precisan que este orden no se siga.

A: ACK. Se trata de un mensaje de confirmación. Si vale 0 el campo de acuse no es de utilidad.

P: Push. En TCP, el paquete no se transifiere a la aplicación hasta que no se alcance cierto tamaño. Esta forma es más eficiente, aunque a veces se necesita que los datos se envíen en un momento preciso, para lo cual se activará este flag.

R: Reset. Se resetea la conexión.

S: Sincronismo. Es bit se encuentra activado en el momento del establecimiento de conexión.

F: Fin. Cuando un equipo quiere terminar la conexión, activa este bit.

Ventana ofertada para el control de flujo: (16 bits) nos indica cuánto espacio libre le queda al buffer del receptor. Se desarrollará más adelante, en el apartado dedicado al control del flujo.

Checksum: (16 bits) Se calcula la suma de comprobación de igual forma a como se calculaba en el protocolo IP. Incluye la cabecera, los datos, y una pseudocabecera como la que se emplea en el protocolo UDP.

Puntero de datos urgentes: (16 bits) si el flag **P** está activo, este campo nos indica dónde terminan los datos urgentes, puesto que puede que no todo el segmento sea urgente.

Opciones: son opcionales. Por ejemplo, se emplean en distintas extensiones de este protocolo, que se denominan "sabores".

3.3.2. Multiplexación/demultiplexación

Una conexión TCP se identifica por:

- Puerto y dirección IP de origen.
- Puerto y dirección IP de destino.

Al igual que ocurría con el protocolo UDP, los puertos inferiores a 1024 están preasignados, y se necesitan permisos de administrador para usarlos. Algunos de ellos se muestran en la Tabla 3.4.

Puerto	Aplicación/Servicio	Descripción
20	FTP-Data	Transferencia de archivos: datos
21	FTP	Transferencia de archivos: control
22	SSH	Terminal seguro
23	Telnet	Acceso remoto
25	SMTP	Correo electrónico
53	DNS	Servicio de nombres de dominio
80	HTTP	Acceso hipertexto (web)
110	POP3	Descarga de correo electrónico
443	HTTPS	Acceso hipertexto seguro (web)

Tabla 3.4: Puertos preasignados en TCP.

3.3.3. Control de conexión

Como ya hemos comentado, el protocolo TCP ofrece un servicio orientado a conexión. El intercambio de mensajes que se dan en una conexión TCP tiene tres fases, que posteriormente desarrollaremos:

1. Establecimiento de conexión (sincroniza el número de secuencia y se reservan los recursos necesarios).

2. Intercambio de datos, donde la conexión sigue un esquema full-duplex.

3. Cierre de conexión, en el que se liberan los recursos empleados.

Notemos además que no es posible garantizar un establecimiento y cierre de conexión fiable de forma directa, pues los segmentos de TCP se encapsulan en datagramas de IP, que no es un protocolo fiable. Para garantizar la fiabilidad, se emplean temporizadores y reenvíos de paquetes que más adelante se desarrollarán.

Establecimiento de conexión

Este tipo de establecimiento se denomina "three-way handshake", pues se intercambian tres mensajes. Supongamos que A quiere establecer una conexión TCP con B para comunicarse. Los mensajes que se intercambian, siempre que no haya incidencias, son los siguientes:

1. A envía un primer mensaje, para solicitar el inicio de la conexión. Al tratarse de un mensaje de sincronismo, se activa el flag **SYN**, y además se envía un número de secuencia aleatorio X.

$$\text{SYN} = 1, \quad \text{Número de secuencia} = X$$

2. Cuando B recibe dicho segmento, activa el flag **ACK** (para confirmar el sincronismo) y, por convenio, en el número de acuse pone $X + 1$. Además, haciendo uso del "piggybacking", iniciamos el sincronismo en el sentido contrario activando el flag **SYN** enviando un número de secuencia aleatorio Y.

$$\text{SYN}, \text{ACK} = 1, \quad \text{Número de Acuse} = X + 1, \quad \text{Número de secuencia} = Y$$

3. Cuando A recibe este último mensaje, activa el flag **ACK** para confirmar la llegada, y en el número de acuse de recibo pone $Y + 1$. El flag de sincronismo no se activa, pues la sincronzación ya está terminada y, debido al "piggybacking", en la práctica se envían ya datos en este segmento. No obstante, por simplicidad, en la asignatura no realizaremos envío de datos en este mensaje.

$$\text{ACK} = 1, \quad \text{Número de Acuse} = Y + 1$$

Notemos que, la entidad que inicia la conexión (en este caso A, normalmente un cliente) realiza lo que se denomina una **apertura activa**, mientras que la otra entidad está escuchando (en este caso B, normalmente un servidor) y realiza una **apertura pasiva**.

Número de secuencia

En este apartado, desarrollaremos más a fondo el concepto de número de secuencia, ya introducido en la cabecera TCP. Es un campo de 32 bits cuya finalidad es contar qué byte del flujo de datos se ha enviado, para poder así ordenarlos en el destino. Cuando este llega al máximo $(2^{32} - 1)$ se reinicia a 0.

Como hemos mencionado en el apartado anterior, este campo de la cabecera no se inicializa a 0, sino que su valor inicial es un número aleatorio[2] denominado Initial Sequence Number (ISN) y elegido por el Sistema Operativo. El estándar TCP *sugiere* que se emplee un contador entero cíclico incrementado en una unidad cada $4\,\mu$s, por lo que el valor del ISN se repitirá tras casi 5 h.

Observación. Este mecanismo es suficiente para proteger de coincidencias (que dos conexiones distintas y simultáneas tengan el mismo valor en este campo), pero no protege frente a sabotajes, ya que es sencillo obtener el ISN de una conexión y suplantar a alguno de los participantes.

Por norma general, al enviar un segmento TCP, se aumenta el número de secuencia anterior según los bytes que ocupase el segmento anteriormente enviado. No obstante, existen las siguientes excepciones:

- Los flags SYN y FIN, cuando están activados, aumentan tan solo en una unidad este valor.
- El flag ACK, cuando está activado, no aumenta este valor.

Veamos algunos ejemplos de establecimientos de conexión particulares, aunque este protocolo contempla muchas otras opciones en las que no entraremos en detalle.

Establecimiento sin incidencias: Este es el caso deseado, en que no hay problema alguno. El intercambio de mensajes es el descrito previamente, y se ilustra en la Figura 3.1.

Caso de conexión simultánea: En este caso, ambas entidades buscan inicar la conexión a la vez, situación que se ilustra en la Figura 3.2. En este supuesto, el segundo y el tercer mensaje se combinan en un único mensaje, que envía cada una de las entidades. Por tanto, ambas se confirman a la vez.

[2]Como el lector sabrá, en la Informática no es posible generar números completamente aleatorios. Por tanto, en realidad es un número pseudoaleatorio, "teóricamente" aleatorio.

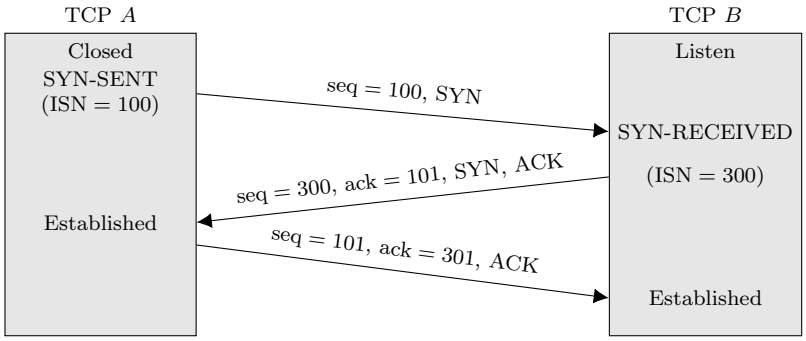


Figura 3.1: Establecimiento de conexión TCP sin incidencias.

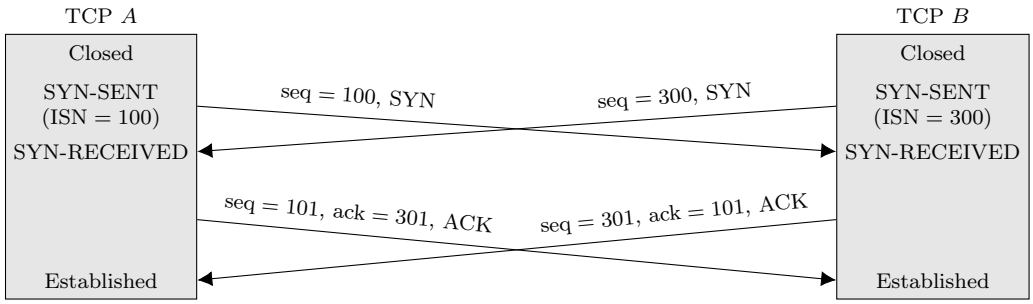


Figura 3.2: Establecimiento de conexión TCP simultánea.

Caso con SYN retrasados y duplicados: En este caso, el mensaje de apuertura de la primera entidad se retrasa, por lo que agota su timeout y se reenvía. Esta situación se ilustra en la Figura 3.3.

En este supuesto, cuando el emisor recibe el ACK de la petición que ya ha descartado, envía un segmento al receptor con el flag de RST activado, para que este se resetee y ambos puedan, de nuevo, sincronizarse.

Cierre de conexión

Una vez visto cómo se establece la conexión, veamos ahora cómo se cierra, procedimiento en el que, además, han de liberarse los recursos. Supongamos que A quiere cerrar la conexión TCP con B. Los mensajes que se intercambian, siempre que no haya incidencias, son los siguientes:

1. A envía un mensaje con el flag **FIN** activado, y en el número de secuencia pone el correspondiente, X.

$$\text{FIN} = 1, \quad \text{Número de secuencia} = X$$

2. La entidad B recibe el mensaje, y activa el flag **ACK** para confirmar la llegada, y en el número de acuse de recibo pone $X+1$. Además, el número de secuencia que envía es el que correspondía, Y, y activa también el flag **FIN**.

$$\text{FIN, ACK} = 1, \quad \text{Número de Acuse} = X + 1, \quad \text{Número de secuencia} = Y$$

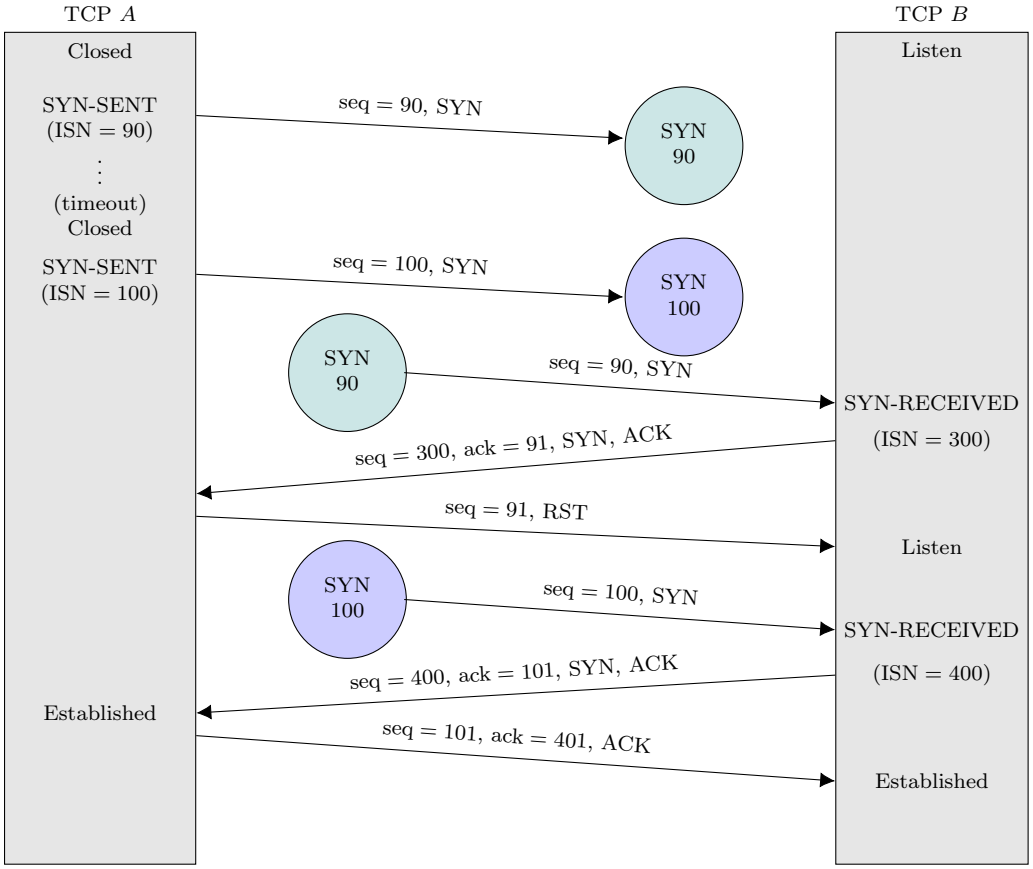


Figura 3.3: Establecimiento de conexión TCP con SYN retrasados y duplicados.

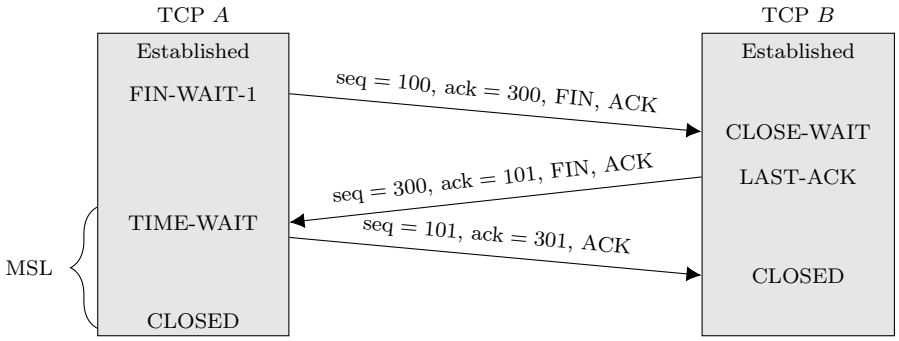


Figura 3.4: Cierre de conexión TCP sin incidencias.

3. La entidad A recibe el mensaje, y activa el flag **ACK** para confirmar la llegada, y en el número de acuse de recibo pone $Y + 1$.

$$\text{ACK} = 1, \quad \text{Número de Acuse} = Y + 1$$

En este caso, al igual que en el establecimiento de conexión, el que inicia el cierre realiza un **cierre activo**, mientras que la otra entidad realiza un **cierre pasivo**. Dependiendo de la aplicación, el cierre activo puede realizarlo tanto el cliente como el servidor.

Aunque la conexión se cierra, para liberar los recursos se deja un tiempo de margen por si tienen que llegar datos aún, denominado Maximum Segment Lifetime (MSL). Este normalmente es de 2 min, aunque puede variar. Toda esta situación se describe en la Figura 3.4.

Diagrama de estados de conexiones TCP

Un diagrama de estados es un autómata finito de estados TCP desde los cuales puedo hacer una serie de acciones, en las cuales puedo recibir y transmitir datos: a/b equivale a que recibo a y transmito b.

3.3.4. Control de errores

Una vez visto el control de la conexión, desarrollamos ahora cómo controla TCP los eventuales errores que se produzcan. Los campos de la cabecera TCP que tenemos involucrados en el control de errores son:

- Número de secuencia: Como se ha comentado, se utiliza para ordenar los datos en el receptor. Indica el offset en bytes dentro del flujo de datos enviado por el emisor.
- Número de acuse de recibo: Indica el siguiente byte que se espera recibir.
- Flag de ACK.
- Campo de checksum, que como se indicó incluye todo el segmento (cabecera TCP y SDU) y la pseudocabecera IP.

Este control de errores se realiza siguiendo un esquema denominado Automatic Repeat Request (ARQ), donde se hacen uso de confirmaciones positivas y acumulativas. En este esquema, si el emisor no ha recibido una confirmación de llegada mediante el ACK, reenvía automáticamente el segmento (de ahí lo de "automatic"). Para las confirmaciones se emplea el bit de ACK de la cabecera TCP junto con el número de acuse de recibo, y son positivas y acumulativas:

- Positivas: Tan solo se confirma lo que ha llegado bien. Si cierto segmento no llega, directamente no se confirma.
- Acumulativas: Cuando en el campo de acuse de recibo se indica un número X, se está confirmando que se han recibido correctamente todos los bytes hasta $X - 1$.

De esta forma, el emisor cuenta con temporizadores ("timeouts") para esperar el ACK de un segmento. Si este expira, se supone que ha habido un error y, por tanto, se reenvía el segmento. De esta forma, tan solo se reenvía un segmento si su correspondiente temporizador ha expirado. Es importante ajustar bien estos temporizadores, pues si son muy cortos no dará tiempo a que lleguen los ACK (provocando reenvíos innecesarios), y si son muy largos se tardará en detectar los errores, ralentizando la comunicación.

Respecto a la ordenación de los segmentos (recordamos que pueden llegar desordenados por la fragmentación de la capa de red), TCP emplea el número de secuencia para ordenarlos. El receptor cuenta con un buffer donde va almacenando los segmentos que llegan, y los va ordenando según el número de secuencia. De esta forma, cuando llega un segmento con un número de secuencia mayor al esperado, se dejará un hueco en el buffer pero se colocará donde le corresponde. Cuando la aplicación pida un dato a la capa de transporte, esta le devolverá los datos que lleven más tiempo en el buffer, gestionándose este como una cola FIFO circular.

Generación de ACKs

Como describíamos anteriormente, es vital la generación de mensajes de confirmación de llegada ACKs, de forma que el emisor sepa que los datos han llegado correctamente y, por tanto, no los reenvíe. Veamos cómo es esta generación:

1. Llegada ordenada de segmento, sin discontinuidad, y con todo lo anterior ya confirmado

   En este caso, y para no saturad la red, se retrasa el envío del ACK. Se establece un temporiador de 500 ms (tiempo fijado por el RFC correspondiente) y, si transcurrido este tiempo no ha llegado otro segmento, se envía el ACK.

2. Llegada ordenada de segmento, sin discontinuidad, y con ACK retrasado pendiente de enviar

   Esta es la situación que complementa a la anterior. En el caso de que todo vaya bien, se habrá recibido un segmento y no habrá sido confirmado porque se están esperando los $500ms$. Si, en ese tiempo, llega otro segmento, como

hay un ACK pendiente, se envía inmediatamente el ACK acumulativo, que confirma la llegada de ambos segmentos.

Vemos por tanto que, si todo va bien, se envía un ACK cada dos segmentos, lo que permite reducir el tráfico en la red.

3. Llegada desordenada de un segmento con número de secuencia mayor que el esperado

   En este caso, se ha detectado una discontinuidad, y TCP debe encargarse de mantener el orden en el buffer. Como se ha mencionado, se colocará donde le corresponde según su número de secuencia, y por tanto se dejará una discontinuidad (hueco). Se envía inmediatamente un ACK duplicado, con el mismo número de acuse de recibo que el último segmento confirmado. Por tanto, no se confirma la llegada del segmento desordenado.

4. Llegada de un segmento que completa una discontinuidad parcial o totalmente

   En este caso, el segmento recibido se encuentra al inicio de la discontinuidad. Si rellena el hueco entero, se dice que ha completado la discontinuidad totalmente, y si solo rellena una parte, se dice que la ha completado parcialmente. En ambos casos, se envía inmediatamente un ACK con el número de secuencia del siguiente byte que se espera.

Por último, es importante destacar que en el control de flujo veremos alguna situación más que requiere envío de estos segmentos de confirmación ACK, pero antes será necesario explicar cómo funciona este control.

Estimación de "timeouts"

Como mencionamos anteriormente, la estimación de estos "timeouts" es vital para, por un lado, no reenviar paquetes que llegasen bien pero, por otro lado, detectar rápidamente los errores. Estos temporizadores se basan en el cáclulo del RTT.

Definición 3.2 (RTT)**.** El Round Trip Time (RTT) es el tiempo que pasa desde que un emisor envía un segmento hasta que recibe el ACK de confirmación de dicho segmento.

Este valor no es fijo, ya que depende de muchos factores. Este tiempo incluye:

- Tiempo de transmisión T_t: Tiempo que tarda en enviarse un segmento, que depende de la tarjeta.
- Tiempo de propagación T_p: Tiempo que tarda desde que se empieza a enviarse el paquete hasta que se empieza a recibir, que depende de la distancia y de la red.
- Tiempo de procesado en cada uno de los routers T_r: Depende de la carga de la red.
- Además de esos tiempos, han de incluirse también los tiempos de vuelta, que son similares a los anteriores, aunque posiblemente algo menores pues el segmento de confirmación tiene un tamaño menor (solamente lleva la cabecera TCP).

Como vemos, calcularlo de forma precisa de forma teórica no es sencillo, pero en la práctica se puede medir de forma sencilla calculando la diferencia de tiempos mencionada. Este valor es el que denominamos $\text{RTT}_{\text{medido}}$. Para calcular el "timeout" de un paquete, hemos de predecir lo que va a valer su RTT. Esta predicción se basa en el valor anterior que habíamos supuesto y en el último valor medido. Para ello, se emplea un filtro que suaviza la media de los valores, y se calcula de la siguiente forma:

$$\text{RTT}_{\text{nuevo}} = \alpha \cdot \text{RTT}_{\text{anterior}} + (1-\alpha) \cdot \text{RTT}_{\text{medido}} \qquad \alpha \in [0,1]$$

Este valor, que de nuevo insistimos en que es una predicción, se utiliza para calcular el "timeout" de un segmento. Además, y debido a cálculos probabilísticos en los que no entraremos en detalle, se calcula también la desviación del RTT respecto a la media, que se utiliza para procurar cubrir todos los casos. Esta desviación se calcula de la siguiente forma:

$$\text{Desviación}_{\text{nueva}} = (1-\beta) \cdot \text{Desviación}_{\text{anterior}} + \beta \cdot |\text{RTT}_{\text{medido}} - \text{RTT}_{\text{nuevo}}| \qquad \beta \in [0,1]$$

Por tanto, usando estos valores, el "timeout" se calcula como:

$$\text{Timeout}_{\text{nuevo}} = \text{RTT}_{\text{nuevo}} + 4 \cdot \text{Desviación}_{\text{nueva}}$$

No obstante, este cálculo no es tan sencillo, ya que los ACKs repetidos generan ambigüedades, pues conocemos el instante de llegada del ACK pero desconocemos si se trata del primer envío del segmento o de envíos posteriores. Esto lo soluciona el Algoritmo de Karn, ya que propone actualizar tan solo el RTT para los ACKs no ambiguos.

No obstante, esto de nuevo supone un problema (algo también considerado por este algoritmo), ya que si todos los ACK se retrasan (por ejemplo, debido a una mayor congestión en la red), el RTT nunca se actualizaría y, por tanto, tampoco el correspondiente "timeout", por lo que siempre se produciría reenvío de paquetes. Para evitar esto, cuando expira un "timeout", no sólo se reenvía el segmento (como hemos comentado), sino que además el timeout se incrementa de forma proporcional al valor anterior (normalmente al doble). De esta forma, para los segmentos que han de reenviarse:

$$\text{Timeout}_{\text{nuevo}} = \gamma \cdot \text{Timeout}_{\text{anterior}}, \qquad \gamma \in \mathbb{R},\ \gamma > 1$$

En la práctica, generalmente se toma $\gamma = 2$. Este algoritmo ha probado ser extremadamente efectivo en el balance entre rendimiento y eficiencia en redes en la que hay alta pérdida de paquetes.

3.3.5. Control de flujo

El protocolo TCP también se encarga de controlar el flujo, que consiste en evitar que el emisor sature al receptor con el envío de demasiados datos o a una velocidad demasiado alta, de forma que al receptor no le dé tiempo a procesarlos. Cuando esto sucede y llegan segmentos estando el buffer del receptor lleno, estos segmentos se descartarán sin haber sido confirmados, lo que habrá supuesto un consumo inútil

de la red, sobrecargándola. De esta forma, el flujo tan solo depende del emisor y del receptor, no de la red; aunque si no se controla bien, puede tener consecuencias negativas en esta última.

Para controlar el flujo, el protocolo TCP emplea un mecanismo de atrás hacia delante: el receptor es quien le informa al emisor cuántos bytes puede enviar. Para ello, se emplea el campo de la cabecera *ventana* (16 bits), por lo que el máximo de bytes que se pueden enviar es $2^{16} - 1 = 65535$ bytes. Se trata por tanto de un sistema crediticio.

No obstante, el emisor ha de tener en cuenta que no tiene a su disponibilidad todos los bytes ofertados, puesto que hay bytes que ya ha enviado pero que no han sido confirmados. Por tanto, la ventana útil para el emisor (los bytes que puede enviar) se calcula como:

$$\text{Ventana útil emisor} = \text{Ventana ofertada receptor} - \text{Bytes en tránsito}$$

Cuando el buffer del receptor se llene, este informará al emisor de que su ventana ofrecida es nula y, por tanto, este se quedará bloqueado y no podrá enviar más datos. Una vez la capa de aplicación lea datos del buffer receptor, liberando así espacio en el buffer, el receptor ha de informar al emisor de que su ventana ofertada ha aumentado, de forma que este pueda desbloquearse y seguir enviando datos. Esto se consigue con una casuística más de los ACK que se envían, que introdujimos en el apartado correspondiente en su momento.

- Si el buffer se encontraba lleno y se ha liberado espacio

  Este es el caso descrito, en el que el buffer se encontraba lleno (por lo que la ventana ofertada era nula y, por tanto, el emisor estaba bloqueado), y en cierto momento la aplicación lee del buffer, liberando espacio. En este momento, el receptor envía un ACK con la nueva ventana ofertada, (que será mayor que cero), y el emisor se desbloqueará y podrá seguir enviando datos.

Este segmento ACK es de vital importancia, puesto que si se pierde, el emisor permanecería bloqueado. Para evitar esta situación, se emplea el denominado *temporizador de persistencia*, que consiste en un temporizador que se inicia tras quedar bloqueado y, cuando expira, envía un byte para forzar el posible reenvío del ACK.

Síndrome de la ventana tonta

Un posible problema es que la aplicación lea de forma lenta y progresivamente del buffer. Esto provocaría que, cada vez que la aplicación leyese, se enviase un ACK con la ventana ofertada, aunque esta sería muy pequeña por haber leído la aplicación pocos datos. Si esto sucede de forma continuada, la ventana estaría continuamente llenándose y liberando un poco de hueco (de ahí el nombre de "tonta") y, como principal problema, que se enviasen muchos segmentos distintos pero de un tamaño muy pequeño (la ventana ofertada en cada caso). Esto provocaría mayor tráfico en la red, algo que buscamos evitar.

Para solucionar esto, hay varias opciones que se pueden implementar:

- El receptor podría ignorar los ACK que ofrezcan una ventana muy pequeña, y esperar a que la ventana ofertada sea mayor.
- El emisor podría enviar estos ACK tan solo cuando la ventana ofertada sea mayor que un cierto umbral.
- Se puede usar también la denominada "ventana optimista", que consiste en que el emisor tome un valor ligeramente mayor que la ventana útil, pero no tanto como la ventana ofertada. De esta forma, se mitiga el problema de la ventana tonta.

Por último, y como aspectos adicionales sobre el control del flujo, es importante destacar que hay dos medios que nos permiten no realizar entregas ordenadas de los datos a la capa de aplicación:

- Emplear el flag de Urgente de la cabecera TCP, junto con el puntero de datos urgentes.
- Emplear el flag de Push, que indica que los datos han de ser entregados a la capa de aplicación inmediatamente.

3.3.6. Control de congestión

La última funcionalidad del protocolo TCP es el control de la congestión, que surge por la saturación de la red. A diferencia del flujo, que era un problema tan solo del emisor y del receptor, la congestión es un problema de la red que se debe a la insuficiencia de recursos, como a un bajo ancho de banda, descartado de datagramas en los nodos intermedios por no caber en sus buffers... Por tanto, y a diferencia del flujo, también involucra al resto de la red. Esta se manifiesta en las pérdidas y/o retrasos de los ACKs.

Controlar este problema es de vital importancia, pues la velocidad del protocolo TCP depende en gran medida de ello. Para ello, se emplea un mecanismo de naturaleza adelante-atrás, en el que el emisor es quien limita la cantidad de tráfico que envía adaptándose así a la situación de la red.

Para la explicación de este control, supondremos en primer lugar que el flujo no produce limitaciones ninguna, y finalmente veremos cómo se combinan ambos controles, dado que estos están directamente relacionados.

Funcionamiento del control de congestión

Como hemos descrito, es el emisor el que decidirá cuantos Bytes enviar. Estos vendrán limitados por la denominada *ventana de congestión* (Congestion Window (CW)), que es el número de Bytes máximo que el emisor puede enviar. El control de la congestión se basa por tanto en la variación de esta ventana para adaptarse a la situación de la red.

Observación. Aunque esta en realidad mide la cantidad de Bytes, en los ejercicios de la asignatura medirá la cantidad de segmentos que se pueden enviar, y supondremos que todos los segmentos tienen el mismo tamaño.

Para el cálculo de esta ventana se emplearán dos parámetros:

- Ventana Inicial: Es el valor que se le asigna a la ventana de congestión al inicio de la conexión. Este valor depende del sistema operativo, aunque ser suele ser el correspondiente para que se puedan enviar dos segmentos consecutivos sin tener que esperar los 500 ms para el envío del ACK.
- Umbral: Valor límite de la ventana de congestión que provocará que la congestión se trate de forma distinta (como se desorrallará más adelante). Normalmente, su valor al inicio de la conexión está fijado por cada versión del protocolo TCP, aunque una vez iniciada se va modificando de forma que se adapte a la situación de la red.

Una vez descrita la terminología necesaria, procedemos a explicar el funcionamiento del control de congestión. Tras el establecimiento de la conexión TCP, el valor de la ventana de congestión se establece en el valor de la ventana inicial:

$$\text{CW} = \text{Ventana Inicial}$$

Tras este momento, el control de congestión se divide en dos fases destinadas a evitar que ocurra un "timeout", junto con las acciones específicas a tomar en el momento en el que este se produce.

Inicio lento: $\text{CW} < \text{Umbral}$

En este caso, la ventana de congestión será pequeña, por lo que se busca ampliarla para poder realizar envíos mayores. Por ello, por cada segmento confirmado, aumentará en una unidad la ventana de congestión. Como si todo va bien cada ACK confirma dos segmentos, tenemos que la ventana de congestión se aumentará en dos unidades por cada ACK recibido. De hecho, si todo va bien, la ventana de congestión se duplicará cuando lleguen todas las confirmaciones de los segmentos enviados, pues al valor que tenía se le sumará el número de segmentos confirmados, que es el mismo valor que tenía.

$$\begin{aligned}\text{CW}_{\text{nueva}} &= \text{CW}_{\text{antigua}} + \text{nº segmentos confirmados} \\ &\overset{(*)}{=} \text{CW}_{\text{antigua}} + \text{CW}_{\text{antigua}} \\ &= 2 \cdot \text{CW}_{\text{antigua}}\end{aligned}$$

donde en $(*)$ hemos supuesto que todo va bien y no hay pérdidas de segmentos. De esta forma, el crecimiento de la ventana de congestión será exponencial, hasta que lleguemos al valor umbral y pasemos a la siguiente fase.

Prevención de congestión: $\text{CW} \geqslant \text{Umbral}$

En esta fase, buscamos que la ventana siga aumentando pero de una forma limitada, pues buscamos evitar la congestión. Para ello, por cada segmento confirmado, la ventana de congestión se modifica como sigue:

$$\text{CW}_{\text{nueva}} = \text{CW}_{\text{antigua}} + \left\lfloor \frac{1}{\text{CW}_{\text{antigua}}} \right\rfloor$$

De esta forma, cuando lleguen todas las confirmaciones de los segmentos enviados, la ventana de congestión aumentará en una unidad. Este crecimiento es por tanto lineal.

Expira un "timeout": En el momento en el que expire un "timeout", es un posible indicador de que la red está congestionada, por lo que hemos de enviar menos segmentos. Además, en ese caso hemos de reducir el umbral, para que se controle la congestión desde antes. Por tanto, las acciones a tomar son:

$$\text{umbral} = \frac{\text{CW}_{\text{antigua}}}{2}, \qquad \text{CW}_{\text{nueva}} = \text{Ventana Inicial}$$

De esta forma, volveremos a la fase de Inicio Lento, pero ahora con un umbral menor.

Observación. Como vemos, esta solución es muy restrictiva, pues el umbral siempre se reduce, llegando incluso al límite en el que el umbral sea 1. Esto no obstante lo resolverán otras versiones de TCP, que veremos más adelante.

Un aspecto importante a resaltar es el valor de la ventana de congestión para el cual el envío puede ser continuado, sin tener que esperar a que llegue el ACK de los segmentos enviados. Este valor es suficiente para que la transmisión de los CW segmentos al medio requiera de más tiempo que la llegada del primer ACK de los segmentos enviados, que permitirá así que se envíen más segmentos. Notando por T_t al tiempo de transmisión de un segmento, y por T_p al tiempo de propagación, tenemos que el valor de la ventana de congestión que no limita la transmisión cumple la siguiente relación:

$$\text{CW} \cdot T_t \geqslant 2T_t + 2T_p$$

Control simultáneo de Flujo y Congestión

Habiendo explicado ambos controles por separado, es directo razonar cuál será la cantidad total que el emisor podrá enviar. Ya que vamos a tener ambas limitaciones (flujo y congestión) de forma simultánea, esta será:

$$\text{Ventana útil} = \text{mín}\{\text{Ventana de congestión}, \text{Ventana Ofertada por el Receptor}\} - \text{Bytes en tránsito}$$

3.3.7. Extensiones TCP

EL protocolo TCP se define con múltiples "sabores" o *flavours* en inglés, que son distintas versiones o extensiones que no afectan a la interoperabilidad entre los extremos. Estas difieren principalmente en el control de la congestión, pues es uno de los aspectos más problemáticos del protocolo. Algunas de ellas son:

- TCP Tahoe: es la versión que se ha estudiado en la asignatura.
- TCP Reno: es la siguiente versión a TCP Tahoe. Para el control de la congestión, distingue entre los timeout (para los que opera igual que Tahoe), y los ACKs duplicados (para los que tan solo reduce a la mitad la ventana de congestión pero sigue en prevención de congestión).
- TCP NewReno: la versión anterior tenía un inconveniente: si se perdían muchos paquetes, en cada uno se reduce la ventana a la mitad, cuando realmente esto no es necesario porque probablemente reduciendo una vez hubiera se podría solucionar. Esta nueva versión intenta ponerle solución a esto empleando para ello ACKs parciales.

- TCP Vegas: Respecto a la congestión, si el RTT aumenta se disminuye la ventana de congestión, y si este disminuye se hace el proceso contrario.
- TCP Cubic: se usa en cualquier versión de Linux con kernel mayor que la 2.6.19. La ventana de congestión depende de los ACKs y del RTT.
- TCP Westwood: hay que tener en cuenta que, si bien en redes cableadas suponer que los errores siempre son por congestión es un buen enfoque; en redes inalámbricas no es ni de cerca el mejor, pues el 10 % de los errores son por el medio. Esta versión tiene en cuenta esto, y está pensada para redes inalámbricas.

4. Seguridad en redes

En el presente tema, dejaremos por un momento de lado la capa de aplicación para centrarnos en la seguridad en las comunicaciones, concepto esencial que iremos detallando a lo largo de las siguientes páginas.

Objetivos

- Comprender la importancia de la seguridad en las comunicaciones y aprender cómo desplegar mecanismos básicos de seguridad en redes de computadores e Internet.
- Conocer los aspectos de seguridad en redes: confidencialidad, autentificación, no repudio, integridad y disponibilidad.
- Entender los conceptos básicos de la seguridad en redes, como el uso de algoritmos de clave secreta, de clave pública, intercambio de claves...
- Comprender qué son los certificados digitales y las autoridades de certificación, y los diferentes mecanismos que se pueden implementar con certificados.
- Conocer algunos de los principales protocolos de comunicación seguros, como TLS e IPSec, y los mecanismos que utilizan.

4.1. Introducción

Una red de comunicaciones es **segura** cuando se garantizan todos los aspectos de seguridad, por lo que no hay protocolos ni redes 100 % seguras. No obstante, el objetivo de una red debe ser cubrir todos los aspectos de seguridad posibles. Definamos brevemente los aspectos de seguridad que vamos a estudiar, junto con los métodos que se utilizan para garantizarlos.

- **Confidencialidad / privacidad:** se garantiza que, cuando transmitimos algo a un receptor determinado, tan solo dicho receptor sea capaz de ver el mensaje. Se consigue con el cifrado.
- **Autenticación:** las entidades son quien dicen ser.
  Se consigue con algoritmos de Reto-Respuesta o doble cifrado.
- **No repudio o irrenunciabilidad:** no se permite la renuncia de la autoría de determinada acción, por lo que se convierte en una prueba legal en ante un juez en el caso de ser necesario. Por ejemplo, no podemos renunciar haber

participado en una transacción.
Se consigue con la firma digital o con el doble cifrado con certificado, pero ha de haber una entidad fiable.

- **Integridad:** se garantiza que los datos no sean manipulados por el camino (intencionadamente o no).
  Se consigue con funciones hash o compendios (resúmenes).

- **Disponibilidad:** el sistema mantiene las prestaciones de los servicios independientemente de la demanda[1].

Como hemos mencionado antes, una red es **segura** cuando se garantizan todos los aspectos de seguridad, y esta debe estar presente en todos los niveles de la red. El grado de seguridad lo *fija el punto más débil*, ya que este es el punto más vulnerable y por el que se podría producir un ataque de seguridad. Por tanto, es importante que haya seguridad en todos los niveles de la red.

Definición 4.1 (Ataque de seguridad)**.** Cualquier acción intencionada o no que menoscaba cualquiera de los aspectos de seguridad.

Veamos algunos ejemplos de ataque de seguridad:

- **Sniffing:** escuchar comunicaciones, por ejemplo mediante Wireshark. Se produce una vulneración de la confidencialidad.

- **Snooping (phishing):** suplantación de la identidad de alguna entidad. Se vulnera la autentificación.

- **Man-in-the-Middle:** un atacante se situa en medio de dos equipos que se comunican e intercepta todos los mensajes que se transmiten. Será un ataque que trataremos en numerosas ocasiones.

- **Distributed Denial of Service (DDoS):** ataque consistente en enviar muchas peticiones a un servidor para que este no pueda atender a todas, consiguiendo que el servicio deje de funcionar. Se denomina *distributed* si las peticiones provienen de distintos equipos, que suele ser lo más común.

- **Malware:** software malicioso, como troyanos, gusanos, *spyware*, *backdoors*, *rootkits*, *keyloggers*, etc. Un ejemplo es *ransomware*, en el que se encriptan todos o parte de los datos y se pide un rescate a cambio de estos.

Los mecanismos de seguridad que vamos a estudiar, como hemos mencionado antes, son:

- Para garantizar la confidencialidad:
  - Cifrado (simétrico y asimétrico).
- Para garantizar la autentificación:
  - Autentificación con clave secreta (reto-respuesta).

[1] Este aspecto no se tratará en la asignatura.

Figura 4.1: Proceso de cifrado y descifrado.

  - Intercambio de Diffie-Hellman (establecimiento de clave secreta).
  - Firma junto con Certificado Digital.
- Para garantizar la integridad:
  - Funciones Hash.
- Para garantizar el no repudio:
  - Firma digital y Certificados digitales.

4.2. Cifrado

Se trata de un procedimiento para garantizar la confidencialidad, y en muchos casos también la autenticación.

El proceso se ilustra en la Figura 4.1. Inicialmente, se dispone de un texto plano a transmitir (P), que buscamos que tan solo pueda ser leído por el receptor. Para ello, se emplea una función de cifrado E_k que dará lugar a un texto cifrado (C), el cual se mandará a través del canal de comunicaciones (no supone un problema, ya que este texto cifrado no será entendible). Llegará al otro extremo y será descifrado con una función $D_{k'}$, obteniendo así de nuevo el texto plano (P).

Los algoritmos de cifrado y descifrado (E_k y $D_{k'}$) normalmente son conocidos, pero estos dependen de claves k y k' que son secretas. La dificultad reside en hallar estas claves.

Veremos dos tipos de algoritmos de cifrado:

- Cifrado simétrico. La clave es secreta y única ($k = k'$), y se usan distintas funciones para cifrar y descifrar.
- Cifrado asimétrico. Hay dos claves (pública y privada), y se usa la misma función para cifrar y descifrar.

4.2.1. Cifrado simétrico

Este tipo de algoritmos de cifrado se denomina simétrico porque se usa la misma clave para cifrar y descifrar los datos. Por tanto, la clave es secreta y tan solo es conocida por el emisor de los datos y el receptor.

Data Encryption Standard (DES)

Se trata de algoritmo de cifrado simétrico que se basa en realizar permutaciones y funciones `XOR` encadenadas. Se cifran palabras de 64 bits usando una clave de 56 bits.

Como principal ventaja, como estas operaciones se pueden implementar de forma sencilla en hardware, es un algoritmo muy rápido, por lo que se puede usar en tiempo real (por ejemplo para codificar voz). No obstante, presenta una serie de problemas:

- La longitud de la clave es corta (2^{56} posibles claves), por lo que es vulnerable a ataques de fuerza bruta.
- Lo que se termina obteniendo es una sustitución, por lo que con la misma entrada el resultado siempre será el mismo. Usando estudios estadísticos dependiendo del idioma, se puede llegar a descifrar el mensaje.

Para mitigar este segundo aspecto, se utiliza un esquema de cifrado reentrante, donde la salida de aplicar una transformación se usa para el cifrado de la siguiente palabra a cifrar. De esta forma, quien recibe el mensaje codificado necesita conocer la última entrada usada para codificar y podrá así aplicar el proceso inverso.

DES doble y 3DES

Estas son distintas mejoras del algoritmo DES para aumentar su seguridad y robustez. Se toman dos claves k_1 y k_2 (en el caso de 3DES, podrían ser 3 distintas) y para cifrar se toma una función E y su inversa D y se concatenan E_{k_1}, D_{k_2}, E_{k_1} y para descifrar se concatenan D_{k_1}, E_{k_2}, D_{k_1}.

De esta forma, podemos simular una clave de 112 bits (en el caso de 2DES) o de 168 bits (en el caso de 3DES); aunque se reduce la velocidad de cifrado.

International Data Encryption Algorithm (IDEA)

Emplea la misma idea que DES (cifrado empleando permutaciones y funciones `XOR`), pero con claves de 128 bits en vez de 56. De esta forma, hay 2^{128} posibles claves, lo que reduce las posibilidades de un ataque por fuerza bruta. Los bloques que se encriptan siguen siendo de 64 bits.

4.2.2. Cifrado asimétrico

Cada usuario A tiene una clave pública K_{pub_A} y una clave privada K_{pri_A} distintas. Conociendo la pública no es posible conocer la privada, por lo que la pública la conocen todos pero la privada solo la conoce su propietario, A. Además, hay una correspondencia biunívoca entre las claves públicas y privadas.

Veamos ahora el funcionamiento de estos algoritmos. Supongamos que A quiere enviar un mensaje a B, por lo que cifraremos con la clave pública de B, de forma que sólo B podrá descifrarlo con su clave privada. De esta forma, se garantiza la confidencialidad del mensaje. Si P es el mensaje a enviar y C el mensaje cifrado, se tiene que:

$$C = K_{\text{pub}_B}(P) \longrightarrow P = K_{\text{pri}_B}(C)$$

De cara a la autentificación, si se cifra un documento con la clave privada de A, se garantiza que solo A ha podido cifrarlo, por lo que a priori[2] se garantiza la autentificación de A.

$$C = K_{\text{pri}_A}(P) \longrightarrow P = K_{\text{pub}_A}(C)$$

RSA

Este algoritmo, cuyo nombre se debe a sus creadores, es ampliamente utilizado en la actualidad. Es un algoritmo de cifrado asimétrico que se basa en la factorización de números enteros. Aunque no entraremos en el detalle de por qué es así el algoritmo[3], explicaremos tanto la generación de claves como el cifrado y descifrado de mensajes. Respecto a la generación de claves, se sigue el siguiente procedimiento:

1. Elegimos p, q primos grandes ($p, q > 10^{100}$).
2. $n = p \cdot q \qquad z = (p-1) \cdot (q-1)$.
3. Elegimos d primo relativo de z ($\text{mcd}(d, z) = 1$).
4. Calculamos e tal que $e \cdot d \mod z = 1$.

Las claves serán los siguientes pares:

$$K_{\text{pub}} = (e, n) \qquad K_{\text{pri}} = (d, n)$$

Para cifrar un número entero P en C, y descifrarlo de nuevo en P, se usan las siguientes funciones:

$$\begin{aligned} \text{Cifrado:} \quad & C = P^e \mod n \quad && \text{con } K_{\text{pub}} = (e, n) \\ \text{Descifrado:} \quad & P = C^d \mod n \quad && \text{con } K_{\text{pri}} = (d, n) \end{aligned}$$

Ejemplo. Veamos el siguiente ejemplo de aplicación del algoritmo.

1. Elegimos $p = 3$ y $q = 11$.
2. $n = 3 \cdot 11 = 33 \qquad z = (3-1) \cdot (11-1) = 20$.
3. Elegimos $d = 7$, ya que $\text{mcd}(7, 20) = 1$.
4. Tomamos $e = 3$, ya que $3 \cdot 7 \mod 20 = 1$.
5. Tenemos $K_{\text{pub}} = (3, 33)$ y $K_{\text{pri}} = (7, 33)$.

Suponemos que queremos codificar la palabra "SUZANNE". Para ello, asignamos un número a cada letra (el orden en el alfabeto sin la ñ), y el proceso se ilustra en la Tabla 4.1. Notemos que, por la red, tan solo se envía el número cifrado, C, y no se podría descifrar sin conocer la clave privada.

[2]En el apartado dedicado a la Firma digital se verá que no es suficiente.

[3]Se basa en factorización de números enteros, y se emplea para ello conocimientos mátemáticos, algunos vistos en la asignatura de Álgebra I, como la función de Euler.

Simbólico	Numérico (P)	$C = P^3$ mód 33	$P = C^7$ mód 33	Simbólico
S	19	28	19	S
U	21	21	21	U
Z	26	10	26	Z
A	1	1	1	A
N	14	5	14	N
N	14	5	14	N
E	5	26	5	E

Tabla 4.1: Cifrado y descifrado de la palabra "SUZANNE".

4.3. Autenticación

Pongámonos en el supuesto de que dos equipos, A y B, quieren autenticarse. Lo más sencillo es que cada uno tenga una base de datos con el usuario y la clave que comparten con el otro. Para autenticarse A, este le manda a B su usuario y su clave compartida, y B comprueba si son correctos. De forma análoga, B se autentica con A. Al estar enviándose la clave, este método es vulnerable, pero se usa en muchos servicios, como en el protocolo PAP. Se pueden hacer algunas mejoras, como enviar la información a través de túneles cifrados, pero la vulnerabilidad sigue presente.

Para evitar este problema, se pueden usar algoritmos de reto-respuesta.

4.3.1. Reto-respuesta

Este algoritmo se ilustra en la Figura 4.2. Al igual que antes, supongamos que dos equipos A y B quieren autenticarse. Ambos tienen una base de datos que contiene, para cada usuario, la clave que comparten. Para autenticarse A con B, este envía su identidad (usuario) A (que no es un dato sensible), y B le contesta con un número aleatorio (R_B) denominado reto. Usando la clave compartida K_{AB}, A cifra el reto ($K_{AB}(R_B)$) y se lo envía a B. El quipo B también cifra el reto con la clave compartida que posee en su base de datos, y si coincide con el que ha recibido, A queda autenticado. Además, A habrá enviado ya un reto R_A a B para que este se autentique de la misma forma. De esta forma, se garantiza la autenticación de ambos equipos.

Notemos que por la red no se envía ningún dado sensible, sino que solo se envían números aleatorios cifrados. Aunque parece seguro, este algoritmo tiene algunas vulnerabilidades:

- **Ataque por repetición:** el atacante escucha por mucho tiempo y va guardando las respuestas correctas para cada resto. De esta forma, cuando se repita un reto, ya sabe qué respuesta ha de enviar para identificarse él, suplantando así la identidad. **Solución:** que el reto no se pueda repetir (denominado *nonce*), como puede ser el instante de tiempo junto a un número aleatorio (por ejemplo).

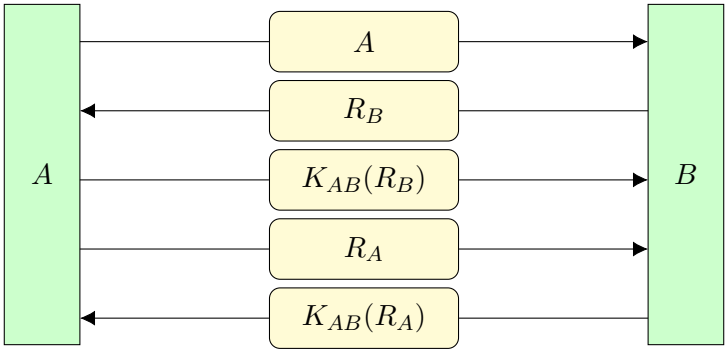


Figura 4.2: Algoritmo de reto-respuesta.

- **Ataque por reflexión:** el atacante, tras escuchar el reto de B, le envía a B su mismo reto, como si este fuese el reto de A. Cuando B le responda, le reenvía dicha respuesta a B como si hubiese sido él quien la ha cifrado. **Solución:** usar dominios de retos disjuntas, para que así no se pueda reenviar el reto de B a B.

4.3.2. Intercambio de Diffie-Hellman

Aunque en el algoritmo de reto-respuesta no se envían claves, estas se han tenido que establecer en algún momento. Además, también es posible que no sea posible almacenar las claves en las bases de datos mencionadas, y que tengamos que calcular una clave secreta y compartida en cada autenticación. Estos dos problemas se resuelven con el intercambio de Diffie-Hellman, ya que este algoritmo permite establecer una clave secreta entre dos entidades a través de un canal no seguro.

El algoritmo se muestra en la Figura 4.3. Aunque tampoco entraremos en detalle en las matemáticas que hay detrás, el algoritmo para crear una clave secreta compartida entre A y B es el siguiente:

1. A elige enteros x, n y g, y B elige el entero y.

2. A envía a B los valores g, n y $g^x \mod n$.

3. B envía a A el valor $g^y \mod n$, que calcula a partir de los valores recibidos.

4. La clave secreta que comparten A y B, que nunca se ha transmitido por la red y, como no se ha transmitido ni x ni y, tampoco la podrá calcular un atacante, es:

$$K_{AB} = (g^y \mod n)^x \mod n = g^{xy} \mod n$$
$$K_{BA} = (g^x \mod n)^y \mod n = g^{xy} \mod n$$

No obstante, este algoritmo también tiene sus vulnerabilidades. Por ejemplo, puede sufrir un ataque del tipo *man-in-the-middle*, en el que un atacante se sitúa entre A y B y se hace pasar por A ante B y por B ante A. De esta forma, hace de mensajero invisible, y las claves compartidas en realidad serán con el atacante.

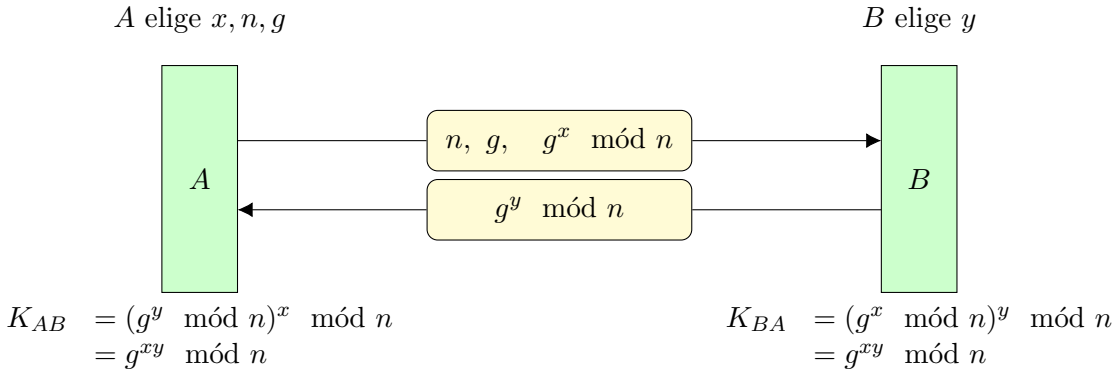


Figura 4.3: Algoritmo de intercambio de Diffie-Hellman.

4.4. Funciones Hash

Las funciones hash son funciones de forma que, dado un texto de entrada M de longitud variable, nos proporcionan un resumen (también llamado compendio) R de longitud fija. Estas funciones son unidireccionales e irreversibles; es decir, a partir de un resumen R no se puede obtener el mensaje original M. Estas deben además ser de cálculo sencillo, pues se usan en muchos protocolos de comunicación y han de poder calcularse rápidamente. Además, son invulnerables a ataques de colisión; es decir, dos mensajes distintos no pueden tener el mismo resumen.

Su principal utilidad es garantizar la **integridad** de un mensaje enviado. Un mensaje M se enviará junto al resumen R de este, y el receptor, tras recibir el mensaje, calculará el resumen del mensaje, R', y comprobará si coincide con el resumen recibido. En tal caso, se garantiza que el mensaje no ha sido modificado en el camino.

No obstante, podría ocurrir que un atacante interceptase el mensaje y lo modificase, cambiando M por M' y $h(M)$ por $h(M')$. Para evitar esto, hay varias alternativas:

- Cifrar el resumen usando la clave compartida, de forma que el atacante no podría cifrar $h(M')$ por no conocer la clave.
- El resumen puede incluir también a la clave compartida entre las entidades. A estos mensajes (M junto a $h(K \mid M)$) se les denomina Hash-based MAC (HMAC).

De ambas formas, también se garantiza la autenticación del emisor del mensaje. La confidencialidad no obstante no se garantiza, pues el mensaje no se cifra, sino que se envía en texto plano.

Message Digest Algorithm 5 (MD5)

Se trata de una función hash que, dado un mensaje, nos proporciona un resumen de 128 bits. Su funcionamiento se muestra en la Figura 4.4, y se desarrolla a continuación.

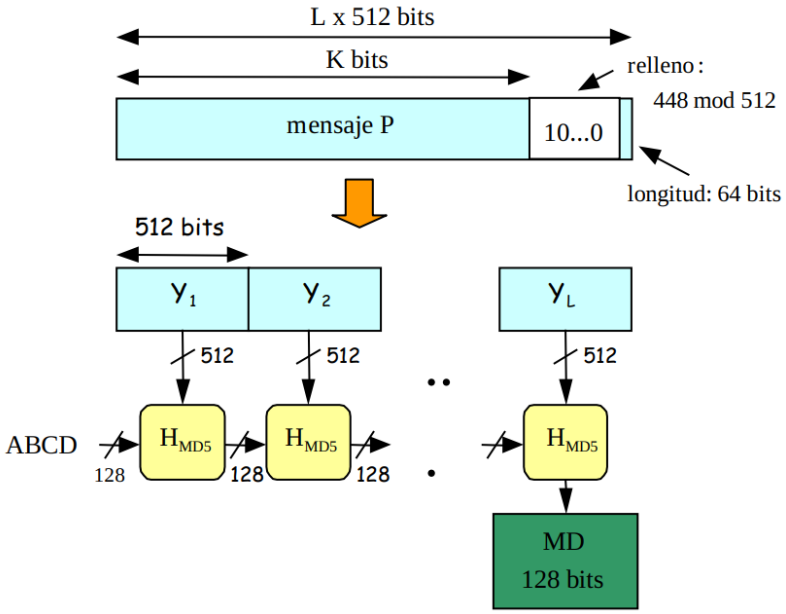


Figura 4.4: Funcionamiento de la función hash MD5.

Como el algoritmo trabaja sobre bloques de 512 bits, el mensaje será *siempre* extendido hasta que su longitud sea congruente con 448 módulo 512 (es decir, notando por K a su longitud, se extiende hasta que $K \equiv 448 \mod 512$, o lo que es lo mismo, hasta que $K - 448 \mod 512 = 0$). Esta extensión se hace añadiendo un 1 seguido de ceros hasta que se cumpla la congruencia, y se hace *siempre* (aunque la longitud del mensaje ya sea congruente con 448 módulo 512).

A continuación, se añade un campo de longitud de 64 bits, que indica la longitud del mensaje original. Si la longitud del mensaje original era mayor a 2^{64}, se toman los 64 bits de menor peso.

Tras añadir este campo de 64 bits, como $448 + 64 = 512$, el mensaje ya tendrá longitud múltiplo de 512, por lo que se divide en bloques de 512 bits, que son a los que les aplicaremos la función hash. Se hace un procesamiento secuencial por bloques, teniendo en cuenta que la salida tras procesar un bloque sirve como entrada para procesar el siguiente. El resumen se obtendrá tras procesar el último bloque.

Secure Hash Algorithm 1 (SHA-1)

El funcionamiento, desde el punto de vista de usuario[4], es análogo al de MD5, ya que también se procesan bloques de 512 bits; aunque en este caso los resúmenes son de 160 bits.

4.4.1. Ataque por Extensión

Supongamos que una entidad A quiere enviarle un mensaje M a otra entidad B. Para garantizar la integridad del mensaje, lo enviará junto a su resumen, y para

[4]No entraremos en detalle en cómo está implementada como tal la función hash, aunque evidentemente son distintas.

evitar que estos sean reemplazados por un atacante, el resumen también se hará de la clave compartida; es decir, se enviará M junto a $h(K_{AB} \mid M)$.

Supongamos no obstante que hay un atacante escuchando, y que intercepta el mensaje M junto al resumen calculado. Aunque no puede cambiar completamente M, sí podrá añadirle información (de ahí el nombre de extensión). Con el mensaje M, puede rellenarlo tal y como se describe en las funciones anteriores, y después de convertirlo a un mensaje con longitud múltiplo de 512, puede añadir al mensaje lo que quiera, obteniendo así un mensaje modificado M'. Para calcular $h(K_{AB} \mid M')$, como desconoce la clave compartida, no podrá hacerlo de forma directa. No obstante, como hemos visto que las funciones hash se basan en realimentación, podría usar el resumen $h(K_{AB} \mid M)$ que ha interceptado como entrada para el primer bloque que él haya añadida. De esta forma, cuando el mensaje llegue a B, este calculará el resumen de M', que coincidirá con el que ha calculado el atacante, creyendo por tanto que el mensaje no ha sido modificado y produciéndose una vulnerabilidad.

Para evitar este problema, el HMAC que en realidad se envía es:

$$h\left(K_{AB} \mid h(K_{AB} \mid M)\right)$$

De esta forma, el atacante podrá modificar el resumen calculado en la función exterior, pero nunca podrá calcular ni modificar el resumen que hay en el interior, ya que no conoce la clave compartida.

4.5. Firma digital y certificados digitales

Una **firma digital** intenta ser un sustituto de una firma escrita para poder garantizar el **no repudio** en nuestras acciones en Internet. Con ellas conseguimos:

- Autenticación del firmante frente al receptor.
- No repudio por parte del firmante.
- El firmante obtiene garantía de no falsificación, proporcionando integridad.
- Se garantiza la confidencialidad del mensaje entre el firmante y el receptor.

Hay dos tipos de firmas digitales, mediante clave secreta o mediante doble cifrado.

4.5.1. Firma digital con clave secreta. *Big Brother.*

Este método se basa el uso de una entidad fiable, denominada *Big Brother* o *BB*, en la que todos los usuarios confían (posiblemente sea del estado). Este comparte una clave con cada una de las entidades, de forma que todos los mensajes pasan por él. De esta forma, si surge algún problema, el Big Brother puede demostrar ante un juez quién ha hecho una transacción, en qué momento, etc.

Supongamos que A quiere enviarle un mensaje P a B firmado digitalmente. La forma de hacerlo se ilustra en la Figura 4.5. En primer lugar, A le envía al Big Brother un mensaje que contiene:

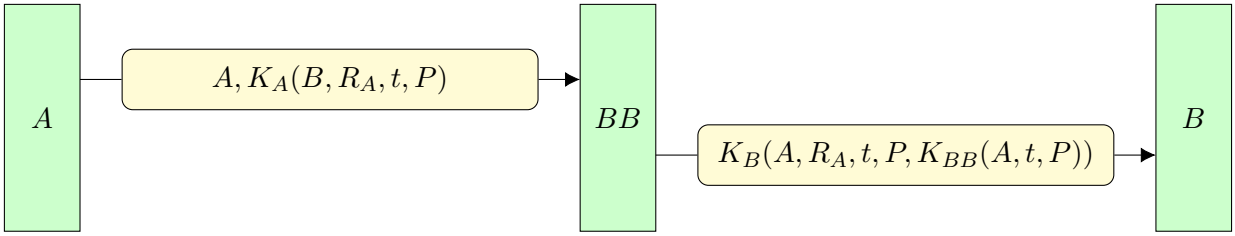


Figura 4.5: Firma digital con clave secreta. *Big Brother*.

- A: Identificador de A.
- B: Identificador de B, el destinatario.
- R_A: Resumen del mensaje para garantizar la integridad.
- t: Instante de tiempo.
- P: Mensaje a enviar, en texto plano.

Notemos que, exceptuando el identificador de A, el resto de campos van cifrados con la clave que comparte A con el Big Brother, proporcinando así confidencialidad con el BB y atenticándose así A. El mensaje es recibido por el BB, que lo descifra, identificando así que tiene que reenviarlo a B. Para ello, añade:

- $K_{BB}(A, t, P)$: Datos de la transacción, cifrados con la clave del BB, que solo él posee y por tanto se convierte en una prueba ante un juez. Se consigue así el no repudio.

Todo esto va cifrado con la clave que comparte B con el BB, garantizando así la confidencialidad del mensaje.

4.5.2. Firma digital con clave asimétrica. Doble cifrado

Supongamos que A le quiere mandar un mensaje a B. La idea se basa en lo siguiente:

- Cifrar con K_{pri_A} garantiza autenticación, ya que solo A ha podido cifrarlo.
- Cifrar con K_{pub_B} garantiza confidencialidad, ya que solo B podrá descifrarlo.

De esta manera, realizando ambos cifrados se garantiza la autenticación y la confidencialidad del mensaje. El orden de los cifrados no importa, pero ha de establecerse previamente para que el destinatario sepa cómo descifrarlo.

$$C = K_{\text{pub}_B}\left(K_{\text{pri}_A}(P)\right)$$
$$P = K_{\text{pub}_A}\left(K_{\text{pri}_B}(C)\right)$$

Sin embargo todo esto no garantiza el **no repudio**, puesto que nada nos garantiza que K_{pri_A} sea realmente de A. Para garantizarlo, necesitamos los **certificados digitales**, que son emitidos por autoridades de certificación.

Certificado digital

Definición 4.2 (Autoridades de certificación)**.** Entidad fiable (posiblemente del estado) que garantiza la asociación entre identidad y claves. En España, por ejemplo, la más extendida es la FNMT, aunque hay otras.

En primer lugar, el usuario obtiene sus claves pública y privada y envía una solicitud firmada digitalmente a la Autoridad de Certificación. Esta, tras comprobar la firma y que el solicitante es quien dice ser, emite el certificado digital solicitado, que contiene (entre otros campos):

- Identidad de la Autoridad de Certificación.
- Identidad del usuario.
- Clave pública del usuario.

Cualquier persona puede consultar este certificado digital, comprobando así que la clave pública que se le proporciona es realmente de la persona que dice ser. Para evitar falsificaciones, el certificado se cifra con la $K_{\text{priv}_{\text{AC}}}$ de la Autoridad de Certificación, de forma que solo esta ha podido ser la emisora del certificado.

El formato de certificados digitales más extendido es el estándar X.509, cuyos campos son:

- Versión del estándar X.509.
- Número de serie único, usado por la Autoridad de Certificación para identificar el certificado.
- Algoritmo empleado para calcular las claves.
- Identidad de la Autoridad de Certificación que ha emitido el certificado.
- Periodo de validez del certificado.
- Usuario al que se le ha emitido el certificado.
- Clave pública del usuario al que se le ha emitido el certificado.

4.6. Protocolos seguros

La seguridad se divide en dos tipos:

- **Perimetral:** consiste en garantizar la seguridad dentro de una misma red. Se usan firewalls, Intrusion Detection System (IDS) o Intrusion Response System (IRS).
- **Seguridad en protocolos:** si no podemos garantizar la seguridad en la red (o aun así, queremos mejorar la seguridad), se usan protocolos seguros para garantizar seguridad. Como vimos, estos han de establecerse en todas las capas, pues la seguridad la fija el punto más débil.

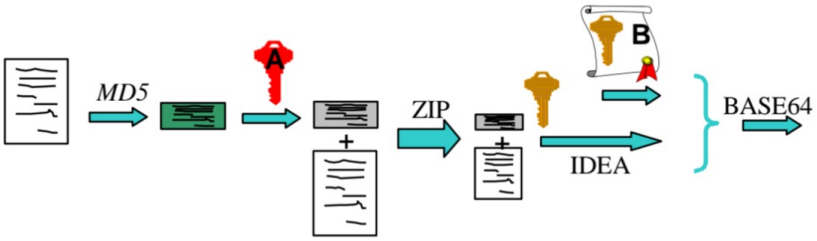


Figura 4.6: Protocolo PGP.

- Capa de aplicación: PGP o SSH.
- Capa de sesión[5]: TLS o SSL.
- Capa de red: IPSec.

4.6.1. Seguridad en la Capa de Aplicación. PGP.

En capa de aplicación deben usarse protocolos distintos para garantizar la seguridad en cada uno de los servicios prestados. Uno de ellos, el protocolo Pretty Good Privacy (PGP), tiene como objetivo garantizar la seguridad en el correo electrónico.

Supongamos que A quiere enviar un mensaje P a B mediante el correo electrónico. El protocolo se ilustra en la Figura 4.6. Al enviar A el correo, el proceso es el siguiente:

1. Se hace un resumen del mensaje mediante el algoritmo MD5, $R = \text{MD5}(P)$.
2. Este se firma digitalmente con la clave privada de A, $FD = K_{\text{pri}_A}(R)$.
3. Tanto el mensaje como resumen cifrado se comprimen (para enviar menos datos) usando para ello el el formato ZIP, $Z = \text{ZIP}(FD + P)$.
4. Se genera una clave privada específica para ese mensaje, K, y se cifra Z con esta clave usando para ello el algoritmo de IDEA, $\text{IDEA}_K(Z)$. A esta clave K se le denomina *clave de sesión*, pues solo se usa en esa sesión. Notemos que, si un atacante la consigue, tan solo podrá descifrar ese mensaje, y no otros.
5. Se cifra también la clave de sesión con la clave pública de B, $K_{\text{pub}_B}(K)$, para que solo B pueda obtener dicha clave y, por tanto, descifrar el mensaje. Por tanto, el mensaje cifrado es $C = K_{\text{pub}_B}(K) + \text{IDEA}_K(Z)$.
6. Este mensaje cifrado se codifica con el sistema Base64[6], $M = \text{Base64}(C)$, y será el que se envíe por internet.

El receptor, por su parte, cuando reciba el mensaje M, hará el proceso inverso:

1. Decodificará el mensaje usando Base64, $C = \text{Base64}^{-1}(M) = K_{\text{pub}_B}(K) + \text{IDEA}_K(Z)$.

[5]Entre aplicación y transporte. Ver Tabla 1.2 correspondiente al Modelo OSI.

[6]No tiene que ver con seguridad y no se verá en la asignatura. Es una forma de codificar mensajes para enviar por internet.

2. Descifrará la clave de sesión con su clave privada, $K = K_{\text{pri}_B}\left(K_{\text{pub}_B}(K)\right)$.

3. Descifrará el mensaje con la clave de sesión, $Z = \text{IDEA}_K^{-1}\left(\text{IDEA}_K(Z)\right)$.

4. Descomprimirá el mensaje, $FD + P = \text{ZIP}^{-1}(Z)$.

5. Se obtiene el resumen recibido, $R = K_{\text{pub}_A}(FD)$.

6. Se calcula el resumen del mensaje recibido, $R' = \text{MD5}(P)$.

7. Para comprobar que no se ha modificado, se comprueba si el resumen calculado es el mismo que el recibido, $R = R'$.

En resumen, tenemos lo siguiente:

Emisor (A)	Receptor (B)
$R = \text{MD5}(P)$	$C = \text{Base64}^{-1}(M)$
$FD = K_{\text{pri}_A}(R)$	$K = K_{\text{pri}_B}\left(K_{\text{pub}_B}(K)\right)$
$Z = \text{ZIP}(FD + P)$	$Z = \text{IDEA}_K^{-1}\left(\text{IDEA}_K(Z)\right)$
Generación K	$FD + P = \text{ZIP}^{-1}(Z)$
$C = K_{\text{pub}_B}(K) + \text{IDEA}_K(Z)$	$R = K_{\text{pub}_A}(FD)$
$M = \text{Base64}(C)$	$R' = \text{MD5}(P)$
	¿ $R = R'$?

Con este proceso conseguimos garantizar los siguientes aspectos de seguridad:

- Confidencialidad: el mensaje va cifrado con K, que va cifrada con la clave pública de B, luego solo B podrá obtener B y, por tanto, solo él descifrar el mensaje.

- Integridad: gracias al resumen, comprobamos que el mensaje no se ha modificado.

- Autenticación: gracias al cifrado con la clave privada de A, B sabe que el mensaje viene de A.

- No repudio: Tan solo si hay un certificado digital de A emitido por una Autoridad de Certificación, B podrá demostrar que el mensaje viene de A.

4.6.2. Seguridad en la Capa de Sesión. SSL y TLS.

Esta capa ofrece servicios de seguridad empleados por gran variedad de protocolos de aplicación, como HTTPS, IMAPS, SSL-POP o VPN. Los principales protocolos de esta capa son SSL y TLS, que en esencia crean túneles cifrados para el intercambio de información.

Secure Sockets Layer (SSL)

En realidad, no es un protocolo sino una familia de ellos que se usan de forma conjunta.

- SSL Handshake Protocol: se negocia el algoritmo de cifrado y la función hash; y el servidor se autentica con un certificado digital del tipo X.509. El cliente genera claves de sesión (de esta forma, si un atacante la consigue tan solo podrá obtener los mensajes de esa sesión), y hay dos opciones para ello:
  - Se generan aleatoriamente y se envían cifradas con la clave pública del servidor (este ya se ha autenticado).
  - Se generan mediante el intercambio de Diffie-Hellman. El problema del "man-in-the-middle" no se tiene en este caso, pues el servidor ya se ha autenticado.
- SSL Assert Protocol: informa sobre errores en la sesión.
- Change Cipher Spec Protocol: usado para notificar cambios en el cifrado.
- SSL Record Protocol: encapsula los protocolos anteriores, y ofrece un canal seguro.

  Para enviar datos, se hace lo siguiente:

  1. Se fragmenta el mensaje en fragmentos denominados *registros*.
  2. Cada registro es comprimido, y al comprimido se le calcula un resumen mediante la función hash acordada.
  3. Se cifra el mensaje comprimido junto con su resumen mediante la clave de sesión.
  4. Esto será lo que trasmite, encapsulado en un paquete TCP.

  Se garantiza *confidencialidad* (pues se cifra), *integridad* (pues se calcula el resumen) y *autenticación* por parte del servidor (pues se ha autenticado con su firma y certificado digital).

Transport Layer Security (TLS)

Inicialmente, el protocolo que había era SSL, desarrollado por la empresa Netscape. No obstante, y basándose en este, se desarrolló el protocolo TLS, al que se le cambió el nombre para indicar que ya no estaba asociado con dicha empresa. Este protocolo solventa distintas vulnerabilidades de SSL, aunque las diferencias inicialmente no eran muy significativas, **impedían la compatibilidad** entre ambos protocolos. En la actualidad el protocolo usado es TLS, aunque es común que, erróneamente y debido a la confusión histórica, la gente se refiera a él como SSL.

En la asignatura, y debido a que nos centramos tan solo en los contenidos básicos sin entrar en detalle en cada uno de los protocolos, hemos estudiado SSL por ser el primero, aunque en lo estudiado no difieren.

4.6.3. Seguridad en la Capa de Red. IPSec.

En la capa de red, se garantiza la seguridad mediante el protocolo IP Security (IPSec). Su objetivo es garantizar autenticación, integridad y (opcionalmente) confidencialidad. Se basa en la creación de túneles unidireccionales seguros[7], para lo cual encapsularemos cada paquete IP en otro paquete IP, con la diferencia de que el paquete encapsulado va cifrado. Se basa en tres procedimientos:

1. Establecimiento de una "Asociación de seguridad":

   Tiene el objetivo de establecer una clave secreta mediante el Intercambio de Diffie-Hellman, y para evitar el ataque de "man-in-the-middle", ha de haber autenticación previa mediante certificados.

   La asociación se identifica mediante la dirección IP de origen y un número de 32 bits, denominado *Security Parameter Index*. Se trata for tanto de una comunicación simplex (en un solo sentido).

   Como aspecto negativo, vulnera el carácter NO orientado a conexión del protocolo IP, y por tanto tu aceptación no es universal.

2. Garantizar la autenticación e integridad de los datos:

   Una vez se ha establecido una "Asociación de seguridad", ha de garantizarse la integridad de los datos y la autenticación de sus orígenes. Esto se consigue añadiendo cabeceras denominadas "Cabeceras de autenticación".

3. Garantizar la confidencialidad de los datos:

   Opcionalmente, este aspecto de seguridad se puede garantizar mediante el protocolo de "Encapsulado de seguridad de la carga".

Este protocolo tiene dos modos de funcionamiento, que diferen en qué túneles se establecen:

- Modo transporte: la asociación (túnel) se hace extremo a extremo entre el host origen y destino, por lo que se protege toda la comunicación entre ambos.
- Modo túnel: la asociación se hace entre dos routers intermediarios.

  Como ejemplo para ilustrar su utilidad, supongamos una empresa con varias sucursales. Si se establece una asociación entre los routers de las distintas sucursales, se garantiza la seguridad de la comunicación entre ellas. Dentro de una misma sucursal la red es segura, pues se habrán establecido medidas de seguridad perimetrales para evitar que entren intrusos desde Internet.

[7] Por tanto, para enviar de forma segura y bidireccional hemos de establecer dos túneles.

5. Relaciones de Problemas

5.1. Introducción

Ejercicio 5.1.1. Explique brevemente las funciones de cada una de las capas del modelo de comunicación de datos OSI.

El modelo de comunicación de datos OSI cuenta con 7 niveles o capas:

1. Capa física: Se encarga de la parte física de la transmisión de los datos. Encontramos distintas formas de codificar los bits para su envío.

   Realiza funciones adicionales, como la codificación del canal.

2. Capa de enlace: Se encarga de los mecanismos de acceso al medio. En caso de haber un medio compartido por varios dispositivos, debe encargarse de no transmitir datos cuando otro medio lo está haciendo y de hacerlo cuando el canal se encuentre libre.

   En esta capa nos encontramos con los protocolos MAC y LLC.

3. Capa de red: Se encarga principalmente del direccionamiento de equipos (saber dar una dirección y tener un identificador dentro de la red) y del encaminamiento de datos (saber cómo mandar los paquetes al destinatario).

4. Capa de transporte: Se encarga principalmente de la fiabilidad de las comunicaciones:

   - Corrección de errores.
   - Manejar el congestionamiento de la red.
   - Controlar el flujo de datos (reducir velocidades si el receptor no es capaz de adecuar la velocidad de recibo a la de envío).
   - Realizar la multiplexación de los datos (ya que un mismo equipo puede tener varias aplicaciones que estén recibiendo datos a la vez).

5. Capa de sesión.

6. Capa de presentación[1].

7. Capa de aplicación: Se encarga de decidir qué datos envía a qué equipo, así como de interpretar los datos recibidos por otros equipos.

[1] No se han mencionado en clase las funcionalidades de las capas de presentación ni de sesión.

Ejercicio 5.1.2. Si la unidad de datos de protocolo en la capa de enlace se llama trama y la unidad de datos de protocolo en la capa de red se llama paquete, ¿son las tramas las que encapsulan los paquetes o son los paquete los que encapsulan las tramas? Explicar la respuesta.

Son las tramas las que encapsulan a los paquetes, ya que son las capas inferiores (en este caso, la de enlace) las que encapsulan la información de las capas superiores (en este caso, la de red) para su envío.

De esta forma, los paquetes son el PDU de la capa de red, que se convierte en el SDU de la capa de enlace, la cual añade su cabecera al mismo convirtiéndolo en su PDU.

Ejercicio 5.1.3. Averigüe qué son los sistemas de representación de datos "*Little Endian*" y "*Big Endian*". ¿Puede un host que utilice representación *Little Endian* interpretar mensajes de datos numéricos provenientes de un host que utilice representación *Big Endian* y viceversa? Discuta la respuesta.

Sí que puede, para ello, debe haber un determinado protocolo que permita indicar qué codificación llevan los datos en binario. De esta forma, en alguna parte de la cabecera de los paquetes enviados, debe haber un bit que indique si los valores numéricos que se envían estén en *Big Endian* o en *Little Endian*.

Ejercicio 5.1.4. Cuando se intercambia un fichero entre dos hosts se pueden seguir dos estrategias de confirmación. En la primera, el fichero se divide en paquetes que se confirman individualmente por el receptor, pero el fichero en conjunto no se confirma. En la segunda, los paquetes individuales no se confirman individualmente, es el fichero entero el que se confirma cuando llega completo. Discutir las dos opciones.

Suponiendo que enviamos n paquetes de datos, la primera forma envía de vuelta al emisor n paquetes de confirmación, uno por cada paquete. De la segunda forma, el receptor espera a unir todos los paquetes en un fichero completo (y a verificar que no se ha perdido ningún paquete del mismo) para enviar el mensaje de verificación.

De la segunda forma se envían menos mensajes de verificación al emisor, por lo que la posibilidad de congestión de red por paquetes de verificación es menor. Sin embargo, en caso de que un paquete no consiga llegar o llegue en mal estado, no será hasta el final del envío de todos los paquetes que el receptor no genere el mensaje al emisor, por lo que en caso de errores en la comunicación, hay un mayor tiempo en la comunicación, al tener que esperar a que el receptor tenga todos los paquetes. Además, en el caso de error, el receptor no informará de qué paquete ha llegado mal, por lo que deberá pedir al emisor que reenvíe todos los paquetes de nuevo.

Resumiendo, ambas estrategias de confirmación tienen sus pros y sus contras. Dependiendo de la situación (si queremos mayor velocidad en la comunicación o si queremos menor saturación de red), puede interesarnos una u otra.

Ejercicio 5.1.5. ¿Para qué sirve el programa *ping*? ¿y el programa *traceroute*?

El programa *ping* usa el protocolo ICMP para enviar un paquete a un equipo, el cual tratará de responder con un paquete de confirmación de recepción del primer paquete. Sirve para comprobar la conexión y el buen funcionamiento de la red existente entre dos equipos. También sirve para calcular empíricamente la latencia de la conexión.

Por otra parte, el programa *traceroute* sirve para consultar todos los nodos intermedios por los que pasan los paquetes que salen de un emisor y llegan a un receptor, junto con la latencia de cada salto. Se usan varios paquetes ICMP con un valor creciente del campo TTL (1,2,...) para que cada salto intermedio devuelva un paquete de error ICMP por TTL excedido. De esta forma, el emisor puede saber cuántos saltos intermedios hay entre él y el receptor, así como la latencia de cada uno de ellos.

Ejercicio 5.1.6. ¿Qué protocolos de un paquete puede cambiar un router? ¿En qué circunstancias?

Un router puede cambiar los protocolos situados debajo de la capa de red, siempre que sea necesario debido a que las redes que interconecta tengan dichos protocolos diferentes.

Por ejemplo, una red doméstica típica es aquella basada en Wi-Fi y con acceso a Internet contratado con tecnología ADSL. En este caso, el router inalámbrico deberá modificar el protocolo de las capas físicas y de enlace convenientemente.

Ejercicio 5.1.7. Averigue qué ISP operan en España.

Algunos de los ISP que operan en España son:

- Movistar.
- Vodafone.
- Orange.
- Jazztel.
- MásMóvil.
- Yoigo.
- Digi.

Ejercicio 5.1.8. ¿Qué es una aplicación cliente-servidor? ¿y una aplicación *peer-to-peer*?

Una aplicación cliente-servidor es una aplicación que depende de otra que probablemente esté en un equipo remoto (llamado servidor) para su funcionamiento.

Un ejemplo de aplicación cliente-servidor es una página web: tenemos aplicaciones que se ejecutan en local en cada equipo que accede a una determinada url. Dicha aplicación solicita datos a una aplicación que se encuentra en un equipo remoto, la cual proporciona datos (por ejemplo, accediendo a una base de datos) como respuesta

a los datos solicitados por la aplicación que se ejecuta en cada equipo de forma local.

Por otra parte, una aplicación *peer-to-peer*[2] es una aplicación que se distribuye entre varios equipos (que pueden estar muy lejanos entre sí) de forma que todas las aplicaciones tienen la misma relevancia en el buen funcionamiento del sistema.

Ambos tipos de aplicaciones se estudiarán en el Capítulo dedicado a la Capa de Aplicación.

Ejercicio 5.1.9. Describa brevemente la diferencia entre un *switch*, *router* y un *hub*.

Para responder a la pregunta, usamos además información que hemos aprendido en el Tema 2:

- Un switch es un nodo en una red que permite conectar tantos equipos como deseemos (normalmente, estos tienen 48 bocas de entrada RJ45 en el caso de conectar los equipos por ethernet) a una red. Funcionan a nivel de enlace, luego no tienen una dirección IP asociada.

- Un router es un nodo en una red que permite conectar redes distintas entre sí. Para ello, disponen de distintas tarjetas de red, cada una asociada a una red que se encuentra conectada al router. Disponen por tanto de varias direcciones IP, una por cada red a la que se conecta. Funciona a nivel de red.

  Presenta en su interior la tabla de enrutamientos, que permite el encaminamiento en la capa de red. Cuenta además con el NAT, que permite traducir direcciones de IP privadas a públicas y viceversa.

- Un *hub* es un nodo en una red que permite implementar la difusión. Se trata de un conjunto de bocas ethernet que internamente funcionan como un bus. Cada vez que un paquete se envía por una de las bocas, este es reenviado a todas las demás bocas, por lo que todos los equipos conectados al *hub* reciben el paquete.

Ejercicio 5.1.10. ¿Qué diferencia, en el contexto de una red de computadores, existe entre la tecnología de difusión y la tecnología punto-a-punto?

La tecnología de difusión permite enviar un paquete desde un equipoo y hacer que este sea recibido por el resto de equipos que estén conectados a la misma red (o hacer llegar estos a equipos en distintas redes). Es cada dispositivo el que decide si el paquete es para él o no.

Por otra parte, la teconología punto-a-punto permite el envío de paquetes desde un equipo a otro usando un medio directo, por lo que el destino está implícito desde que se envía el paquete. Esta tecnología es más rápida y segura, pero su escalabilidad es mucho menor.

Ejercicio 5.1.11. Un sistema tiene una jerarquía de protocolos de n capas. Las aplicaciones generan mensajes de M bytes de longitud. En cada capa se añade una cabecera de h bytes. ¿Qué fracción del ancho de banda de la red se llena con cabeceras? Aplique el resultado a una conexión a 512 kbps con tamaño de datos de 1500

[2]En español, podemos pensar en "entre pares".

bytes y 4 capas, cada una de las cuales añade 64 bytes cabecera. ¿Qué velocidad real de envío de datos resulta?

Debemos sumar a los M bytes iniciales que proporcionan las aplicaciones n veces (la capa de aplicación también incluye una cabecera) h bytes, por lo que la longitud de los mensajes que de verdad se envían es de $n \cdot h + M$ bytes. Por tanto, por cada $n \cdot h + M$ bytes enviados, $n \cdot h$ de ellos son de cabeceras:

$$\frac{n \cdot h}{n \cdot h + M} \cdot 100 \ \% \text{ de ancho de banda que se llena de cabeceras}$$

Si ahora partimos de 1500 bytes iniciales y añadimos 4 veces (una por capa) 64 bytes, estamos en realidad enviando mensajes de longitud:

$$4 \cdot 64 + 1500 = 256 + 1500 = 1756 \text{ bytes}$$

Por tanto, el ($^{256}/_{1756} = 0{,}145786$) 14.58 % de la red se emplea para enviar cabeceras, luego se aprovecha el ($1 - 0{,}145786 = 0{,}854214\,\%$) 85.42 % de la red para el envío de datos.

Si enviamos paquetes a una velocidad de 512kbps, en realidad estaremos enviando datos a una velocidad real de:

$$0{,}854214 \cdot 512 = 437{,}357568 \text{ kbps}$$

Ejercicio 5.1.12. Clasifique como de *difusión* o *punto a punto* cada uno de los siguientes sistemas de transmisión:

1. Radio y TV: Difusión, ya que es un emisor (en este caso, una cadena de televisión o radio) que difunde paquetes a cualquiera que tenga sintonizado dicho canal.

2. Redes inalámbricas (WLAN): Difusión, ya que cualquier equipo puede conectarse a la red y recibir los paquetes que se envían de forma inalámbrica.

3. ADSL: Punto a punto, ya que la conexión se establece mediante un cable. Usa en medio único.

4. Redes de cable: Puede implementar ambas tecnologías, ya que puede ser punto a punto (si cada equipo tiene su propio cable) o de difusión (si todos los equipos comparten el mismo cable).

5. Comunicaciones móviles (por ejemplo, GSM, UMTS, ...): Difusión, ya que (de nuevo) cualquier equipo puede recibir los paquetes que se envían por la red.

Ejercicio 5.1.13. Clasifique los siguientes servicios como orientados a conexión/no orientados a conexión y confirmados/sin confirmación. Justifique la respuesta.

Recordamos que los medios orientados a conexión son aquellos que comprueban si el receptor está disponible antes de enviar la información, y que los confirmados son aquellos que confirman la recepción del mensaje.

1. Correo postal ordinario: No es orientado a conexión (ya que no comprobamos anteriormente que el destinatario esté disponible, simplemente enviamos la carta) y es sin confirmación, ya que tras enviar la carta nada nos garantiza que el destinatario nos responda, pese a pedirlo explícitamente.

2. Correo certificado: No es orientado a conexión por la misma razón que el correo normal. Sin embargo, es confirmado, ya que al recibir el destinatario la carta debe firmar para que el emisor sea consciente de que la carta ha sido recibida.

3. Envío y recepción de fax: Orientado a conexión y confirmado.

4. Conversación telefónica. Es orientado a conexión, puesto que no se puede establecer una llamada si la otra persona no coge el teléfono. Además, es confirmado, ya que la otra persona ha de responder para que se produzca una comunicación.

5. Domiciliación bancaria de recibos: No es orientado a conexión, ya que no se comprueba si el destinatario está disponible antes de enviar la información. Además, es confirmado, puesto que el banco envía un mensaje de confirmación al emisor del recibo.

6. Solicitud de certificado de empadronamiento. Es no orientado a conexión, ya que no se comprueba si el destinatario está disponible antes de enviar la solicitud. Además, es con confirmación, ya que el ayuntamiento envía un documento que certifica que el solicitante está empadronado.

Ejercicio 5.1.14. ¿Cuál es el tiempo necesario en enviar un paquete de 1000 Bytes, incluidos 50 Bytes de cabecera, por un enlace de 100 Mbps y 10Km? ¿cuál es el tiempo mínimo desde que se envía hasta que se recibe confirmación? ¿qué relación hay entre este tiempo y los temporizadores en, por ejemplo, las capas de enlace y transporte?

En primer lugar, hemos de calcular el retardo de transmisión T_t, que es el tiempo que se tarda en enviar el paquete por el enlace. Para ello, tenemos que:

$$T_t = 10^3\,\mathrm{B} \cdot \frac{8\,\mathrm{b}}{1\,\mathrm{B}} \cdot \frac{1\,\mathrm{s}}{100 \cdot 10^6\,\mathrm{b}} = 80 \cdot 10^{-6}\,\mathrm{s} = 80\,\mu\mathrm{s}$$

Por otra parte, el retardo de propagación T_p es el tiempo que se tarda en enviar el paquete por los 10Km de cable. Para ello, suponiendo que la velocidad de transmisión es $^2/_3c = 2 \cdot 10^8\,\mathrm{m/s}$, tenemos que:

$$T_p = 10 \cdot 10^3\,\mathrm{m} \cdot \frac{1\,\mathrm{s}}{2 \cdot 10^8\,\mathrm{m/s}} = 50 \cdot 10^{-6}\,\mathrm{s} = 50\,\mu\mathrm{s}$$

Por tanto, el tiempo total que se tarda en enviar el paquete es de $T_t + T_p = 130\,\mu\mathrm{s}$.

Veamos ahora el tiempo mínimo desde que se envía hasta que se recibe confirmación. Además de los tiempos anteriores, hemos de tener en cuenta el tiempo de procesamiento del paquete, el retardo de transmisión del paquete de confirmación y el retardo de propagación del paquete de confirmación. Tenemos que:

- No se proporciona información del tiempo de procesamiento del paquete. No obstante, en los dispositivos modernos, este retardo es de varios órdenes de magnitud menor que los otros, por lo que se puede considerar despreciable.
- El retardo de propagación del paquete de confirmación es el mismo, puesto que la distancia recorrida es la misma.
- El retardo de transmisión sí difiere, puesto que el tamaño del paquete de confirmación difiere. Normalmente, estos solo incluyen una cabecera, por lo que este tiempo, notado por T_{ACK}, es:

$$T_{\text{ACK}} = 50\,\text{B} \cdot \frac{8\,\text{b}}{1\,\text{B}} \cdot \frac{1\,\text{s}}{100 \cdot 10^6\,\text{b}} = 4 \cdot 10^{-6}\,\text{s} = 4\,\mu\text{s}$$

  Un temporizador de control de flujo en capa de enlace o de transporte debe ser suficientemente mayor a este tiempo mínimo para evitar un re-envío inmediato de paquetes ante cualquier eventualidad mínima en la red, como un retardo en las colas (mayor retardo de procesamiento) por un cierto nivel de congestión.

Por tanto, el tiempo total mínimo que se tarda en enviar el paquete y recibir confirmación es de:

$$T_{\text{total}} = T_t + T_p + T_{\text{ACK}} + T_p = 80\,\mu\text{s} + 50\,\mu\text{s} + 4\,\mu\text{s} + 50\,\mu\text{s} = 184\,\mu\text{s}$$

5.2. Capa de red

Ejercicio 5.2.1. Estime el tiempo involucrado en la transmisión de un mensaje de datos para la técnicas de conmutación de circuitos (CC) y de paquetes mediante datagramas (CDP) y mediante circuitos virtuales (CPCV) considerando los siguientes parámetros:

- M: longitud en bits del mensaje a enviar.
- V: velocidad de transmisión de las líneas en bps.
- P: longitud en bits de los paquetes, tanto en CPD como en CPCV.
- H_d: bits de cabecera de los paquetes en CPD.
- H_c: bits de cabecera de los paquetes en CPCV.
- T: longitud en bits de los mensajes intercambiados para el establecimiento y cierre de conexión, tanto en CC como en CPCV.
- N: número de nodos intermedios entre las estaciones finales.
- D: tiempo de procesamiento en segundos en cada nodo, tanto en CC como en CPD y en CPCV.
- R: retardo de propagación, en segundos, asociado a cada enlace, en CC, en CPD y en CPCV.

Ejercicio 5.2.2. Un mensaje de 64 kB se transmite a lo largo de dos saltos de una red. Ésta limita la longitud máxima de los paquetes a 2 kB y cada paquete tiene una cabecera de 32 B. Las líneas de transmisión de la red no presentan errores y tienen una capacidad de 50 Mbps. Cada salto corresponde a una distancia de 1000 km. ¿Qué tiempo se emplea en la transmisión del mensaje mediante datagramas?

Como las cabeceras son de 32 B y el MTU es de 2 kB, cada paquete podrá enviar, como máximo:

$$2\,\text{kB} - 32\,\text{B} = 2028\,\text{B} - 32\,\text{B} = 1996\,\text{B}$$

Además, y aunque no se especifica en el enunciado, se asume[3] que estos se envían empleando el protocolo IP y que la cabecera es mayor debido al uso de campos opcionales. Como el campo de desplazamiento de la cabecera se mide en múltiplos de 8 B, el tamaño de los datos ha de ser múltiplo de 8 B, por lo que como máximo se podrán enviar 1992 B de datos (que provocará que el desplazamiento aumente en 249 unidades en cada paquete). Por tanto, en cada paquete se enviarán 1992 B de datos y 32 B de cabecera.

Veamos cuántos paquetes se envían:

$$64\,\text{kB} = 32 \cdot 1992\,\text{B} + 1792\,\text{B}$$

[3]En el caso de no realizar esta asunción, sería similar solo que trabajando con el valor anteriormente calculado.

Figura 5.1: Red de conmutación de paquetes del ejercicio 5.2.4.

Por tanto, se envían 33 paquetes; el último de ellos con 1792 B de datos y 32 B de cabecera.

Busquemos por tanto ahora el tiempo. Para ello, primero calculamos el tiempo de transmisión de un paquete:

$$T_t = \frac{L\,\mathrm{B} + 32\,\mathrm{B}}{50\,\mathrm{Mbps}} = \begin{cases} 0{,}3238\,\mathrm{ms} & \text{para los primeros 32 paquetes} \\ 0{,}2918\,\mathrm{ms} & \text{para el último paquete} \end{cases}$$

Veamos ahora el tiempo de propagación entre dos nodos, donde suponemos que la red es cableada y, por tanto, la velocidad de propagación es de $^2/_3 \cdot c = 2 \cdot 10^5\ ^{\mathrm{km}}/_{\mathrm{s}}$:

$$T_p = \frac{1000\,\mathrm{km}}{2 \cdot 10^5\ ^{\mathrm{km}}/_{\mathrm{s}}} = 5\,\mathrm{ms}$$

Calculemos ahora el tiempo total que tarda en llegar tan sólo el primer paquete al segundo nodo (recorrido completo). Este será[4]:

$$T^1_{\mathrm{total}} = 2T_t + 2T_p$$

Una vez llegado este, cada paquete más tan sólo supondrá el retardo provocado por la transmisión, por lo que el tiempo total que tardan en llegar los 33 paquetes será, como mínimo:

$$\begin{aligned} T_{\mathrm{total}} &= T^1_{\mathrm{total}} + 31T_t + T_t = 33T_t + 2T_p + T_t = \\ &= 33 \cdot 0{,}3238\,\mathrm{ms} + 2 \cdot 5\,\mathrm{ms} + 0{,}2918\,\mathrm{ms} = 20{,}9772\,\mathrm{ms} \end{aligned}$$

Ejercicio 5.2.3. Suponga que una red de datagramas usa cabeceras de H bytes y que una red de paquetes de circuitos virtuales utiliza cabeceras de h bytes. Determine la longitud M de un mensaje que se consigue transmitir más rápido haciendo uso de la técnica de conmutación de circuitos virtuales que mediante la de datagramas. Suponga que los paquetes tienen la misma longitud en ambas redes y que los retardos de procesamiento son idénticos

Ejercicio 5.2.4. Una aplicación audiovisual en tiempo real hace uso de conmutación de paquetes para transmitir voz a 32 kbps y vídeo a 64 kbps a través de la conexión de red de la figura 5.1. Se consideran paquetes de voz e información de audio con dos longitudes distintas: 10 ms y 100 ms. Cada paquete tiene además una cabecera de 40 B.

1. Encuentre para ambos casos el porcentaje de bits suplementarios que supone la cabecera.

   Veamos en primer lugar cuántos datos hemos de enviar:

[4] Como mínimo, pues consideramos despreciable tiempos como el de procesamiento.

- Para la voz, se envían $32\,\text{kbps} \cdot 10\,\text{ms} = 320\,\text{b}$ de datos.
- Para el vídeo, se envían $64\,\text{kbps} \cdot 100\,\text{ms} = 6400\,\text{b}$ de datos.

Por tanto, el porcentaje de bits suplementarios que supone la cabecera es:

- Para la voz:
$$\frac{(40 \cdot 8)\,\text{b}}{(40 \cdot 8)\,\text{b} + 320\,\text{b}} = 0{,}5 \Longrightarrow 50\,\%$$
- Para el vídeo:
$$\frac{(40 \cdot 8)\,\text{b}}{(40 \cdot 8)\,\text{b} + 6400\,\text{b}} = 0{,}0476 \Longrightarrow 4{,}76\,\%$$

2. Dibuje un diagrama temporal e identifique todas las componentes del retardo extremo a extremo en la conexión anterior. Recuerde que un paquete no puede ser transmitido hasta que esté completo y que no se puede retransmitir hasta que no se haya recibido completamente. Suponga despreciables los errores a nivel de bit.

3. Evalúe todas las componentes del retardo de las que se dispone suficiente información. Considere las dos longitudes de paquete aceptadas. Suponga que la señal se propaga a una velocidad de 1 km/5 microsegundos y considere dos velocidades para la red troncal: 45 Mbps y 1,5 Mbps. Resuma el resultado para los cuatro posibles casos en una tabla con cuatro entradas.

4. ¿Cuál de las componentes anteriores implica la existencia de retardos de cola?

Ejercicio 5.2.5. Imagine la situación descrita en la figura 5.2, donde se muestra una red. El router 5 a Internet a través de la puerta de acceso especificada por el ISP. Especifique las tablas de encaminamiento en los routers. Asigne a voluntad las direcciones IP e interfaces necesarias.

En primer lugar, asignaremos las direcciones IP y las interfaces de cada router. Cada router presenta dos conexiones, por lo que sólo usaremos dos interfaces por cada router, luego asociaremos dos direcciones IP a cada router.

Por comodidad, asociaremos las conexiones más cercanas a Internet de cada router a la interfaz *ether0*, y a las más lejanas la interfaz *ether1*.

Una vez añadidas las interfaces, procederemos a asocciar direcciones IP a cada interfaz de cada router:

- R1:
  - Tendrá IP 192.168.2.1 en la red 192.168.2.0/24.
  - Tendrá IP 192.168.1.2 en la red 192.168.1.0/24.
- R2:
  - Tendrá IP 192.168.3.1 en la red 192.168.3.0/24.
  - Tendrá IP 192.168.1.3 en la red 192.168.1.0/24.
- R3:

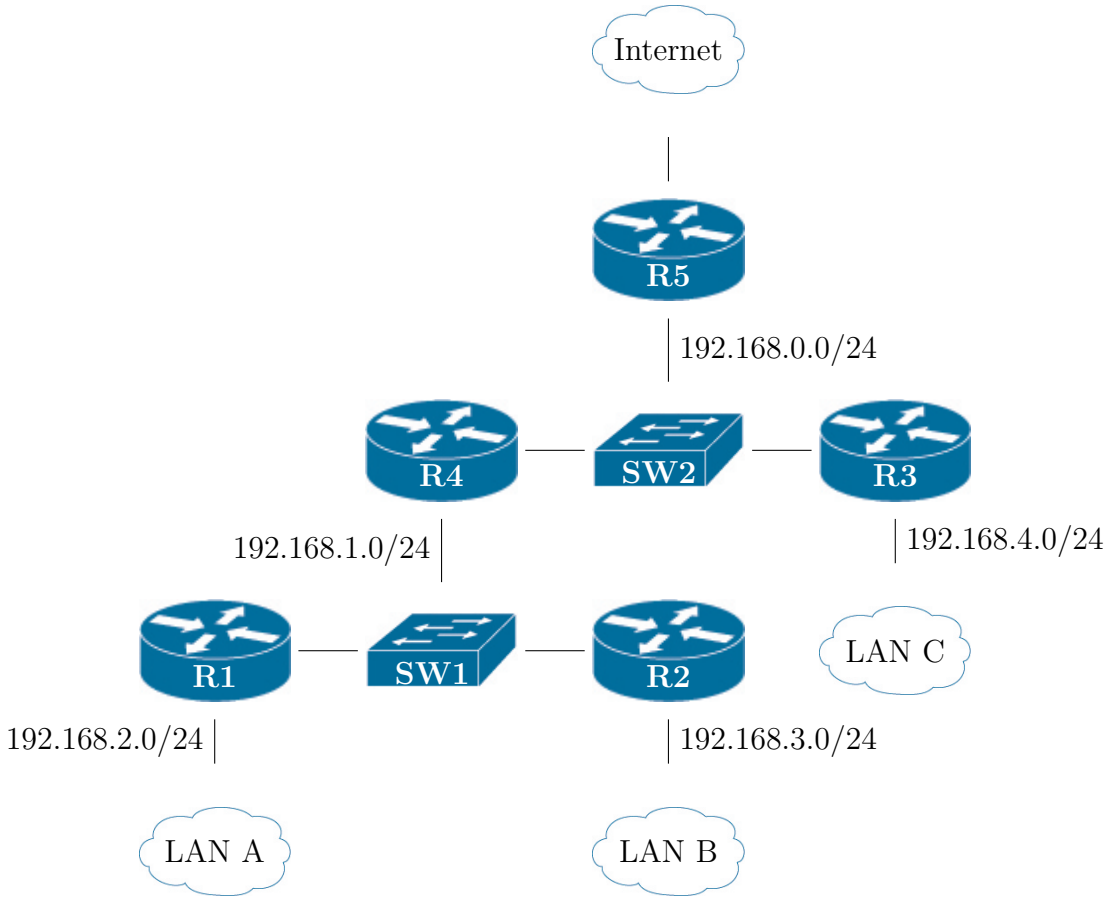


Figura 5.2: Situación del Ejercicio 5.2.5.

Red destino	Máscara	Siguiente salto	Interfaz
192.168.2.0	/24	*	ether1
192.168.1.0	/24	*	ether0
192.168.3.0	/24	192.168.1.3 (R2)	ether0
`default`	/0	192.168.1.1 (R4)	ether0

Tabla 5.1: Tabla de encaminamiento para R1.

Red destino	Máscara	Siguiente salto	Interfaz
192.168.3.0	/24	*	ether1
192.168.1.0	/24	*	ether0
192.168.2.0	/24	192.168.1.2 (R1)	ether0
`default`	/0	192.168.1.1 (R4)	ether0

Tabla 5.2: Tabla de encaminamiento para R2.

- Tendrá IP 192.168.1.1 en la red 192.168.1.0/24.
- Tendrá IP 192.168.0.2 en la red 192.168.0.0/24.

- R4:
  - Tendrá IP 192.168.4.1 en la red 192.168.4.0/24.
  - Tendrá IP 192.168.0.3 en la red 192.168.0.0/24.
- R5:
  - Tendrá IP 192.168.0.1 en la red 192.168.0.0/24.
  - Su IP en la red que le conecta con Internet la proveerá el ISP. Supongamos que es 33.33.0.0/16.

Procedemos ahora a rellenar las tablas de encaminamiento de cada router:

Donde hemos agrupado la Red A (192.168.2.0/24), B (192.168.3.0/24) y la red 192.168.1.0/24 en la superred 192.168.0.0/22. Notemos que dentro de las direcciones de la superred se encuentran las direcciones de la forma 192.168.0.x, que no se encuentran en dicha superred. Sin embargo, tenemos una entrada específica para dichas direcciones, con una máscara de mayor prioridad (más 1s), por lo que no tenemos problema[5].

[5]Si no tuviéramos dicha entrada, tendríamos un problema, ya que si mandamos un paquete a 192.168.0.26, por ejemplo, iría a la superred que hemos definido pero algún router se daría cuenta de que no sabe llegar a 192.168.0.0/24.

Red destino	Máscara	Siguiente salto	Interfaz
192.168.4.0	/24	*	ether1
192.168.0.0	/24	*	ether0
192.168.0.0	**/22**	192.168.0.2 (R4)	ether0
`default`	/0	192.168.0.1 (R5)	ether0

Tabla 5.3: Tabla de encaminamiento para R3.

Red destino	Máscara	Siguiente salto	Interfaz
192.168.1.0	/24	*	ether1
192.168.0.0	/24	*	ether0
192.168.2.0	/24	192.168.1.2 (R1)	ether1
192.168.3.0	/24	192.168.1.3 (R2)	ether1
192.168.4.0	/24	192.168.0.3 (R3)	ether0
`default`	/0	192.168.0.1 (R5)	ether0

Tabla 5.4: Tabla de encaminamiento para R4.

Red destino	Máscara	Siguiente salto	Interfaz
192.168.0.0	/24	*	ether1
33.33.0.0	/16	*	ether0
192.168.4.0	/24	192.168.0.3 (R3)	ether1
192.168.0.0	**/22**	192.168.0.2 (R4)	ether1
`default`	/0	Router_ISP_Gateway	ether0

Tabla 5.5: Tabla de encaminamiento para R5.

Donde hemos vuelto a usar la superred 192.168.0.0/22 que engloba a Red A, B y 192.168.1.0/24.

Ejercicio 5.2.6. Asigne las direcciones de subred en la siguiente topología a partir de 192.168.0.0 para minimizar el número de entradas en las tablas de encaminamiento, asumiendo que en las redes LAN puede haber hasta 50 PCs.

Ejercicio 5.2.7. Un datagrama de 4020 bytes pasa de una red Token Ring con THT 8 ms (MTU 4400) a una Ethernet (MTU 1500) y después pasa por un enlace PPP con bajo retardo (MTU 296). Si ese mismo datagrama pasara directamente de la red Token Ring al enlace PPP (sin pasa por la red Ethernet) ¿habría alguna diferencia en la forma como se produce la fragmentación? Especifique en ambos casos los fragmentos obtenidos.

Ejercicio 5.2.8. ¿Cómo podría utilizar ICMP para hacer una estimación de la latencia entre dos entidades finales? ¿Y para estimar la latencia de un enlace en particular entre dos routers?

Ejercicio 5.2.9. Considere la subred de la figura. Se utiliza el algoritmo de encaminamiento de vector distancia, habiéndose recibido en el encaminador C los siguientes vectores de encaminamiento: desde B (5, 0, 8, 12, 6, 2), desde D (16, 12, 6, 0, 9, 10) y desde E (7, 6, 3, 9, 0, 4). Los retardos medidos a B, D y E son, respectivamente, 6, 3 y 5. ¿Cuál es la nueva tabla de encaminamiento de C? Indique la línea de salida y el retardo esperado.

Ejercicio 5.2.10. Considere la red mostrada en la figura 5.4, en la que se representan 4 nodos unidos con enlaces. En cada enlace se indica el retardo sufrido por los mensajes al atravesarlo. Los nodos utilizan un protocolo de encaminamiento dinámico de tipo distribuido en el que la métrica está basada en el retardo. Se pide lo siguiente:

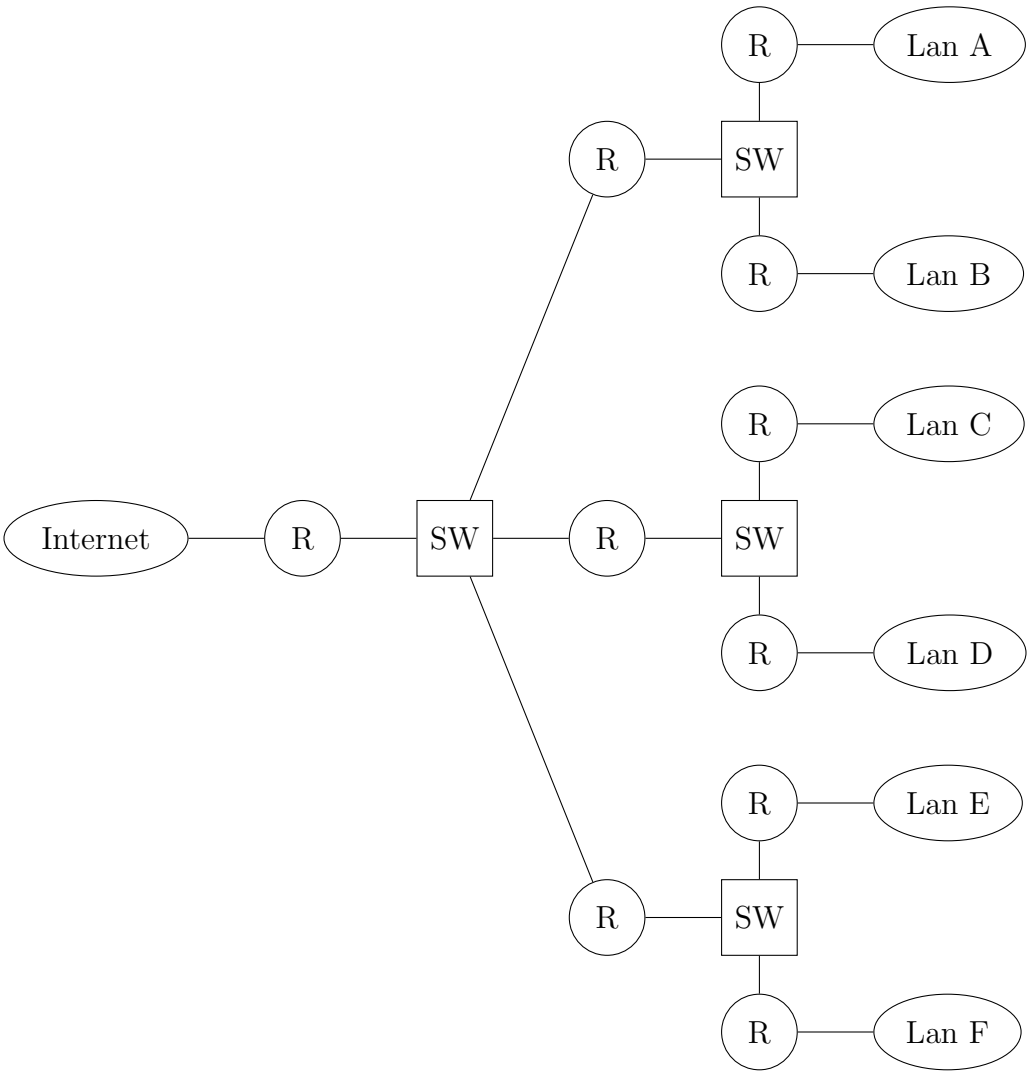


Figura 5.3: Situación del ejercicio 6

a. Escriba las tablas de encaminamiento para todos los nodos de la red una vez haya pasado el tiempo suficiente para que dichas tablas se construyan de forma estable.

b. Considere que los nodos envían y actualizan sus tablas cada 5 segundos, siendo la primera actualización en $t = 0$ s. Suponga que, en $t = 12$ s., el enlace BD pasa a tener un retardo de 3 s. ¿Cuál será el encaminamiento desde el nodo A hasta el nodo B cuando las tablas se estabilicen de nuevo?

c. ¿En qué instante comenzará dicho encaminamiento a funcionar? Justifique su respuesta explicando qué sucederá desde $t = 12$ s. hasta dicho instante y también después del mismo.

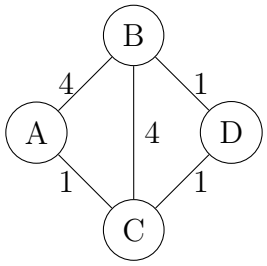


Figura 5.4: Grafo para el ejercicio 10.

Ejercicio 5.2.11. En la topología de red adjunta se indica la capacidad, en Kbps, de las líneas de transmisión entre los distintos nodos intermedios. Considérese al respecto que los enlaces son *full-duplex* y que la velocidad es la misma para cada uno de los sentidos. Por otra parte, la tabla anexa especifica el tráfico, en paquetes/segundo, entre cada par de nodos. Además, en cursiva se indica la ruta (secuencia de nodos) seguida en la transmisión. Teniendo en cuenta todo lo anterior, determine el retardo medio en el envío de un paquete sobre la red global.

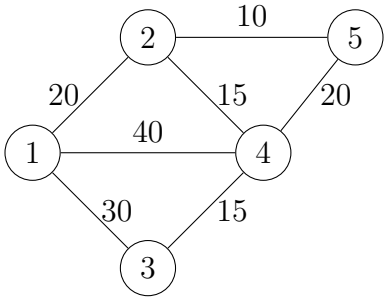


Figura 5.5: Grafo para el ejercicio 11.

		Nodo destino				
		1	2	3	4	5
Nodo origen	1		2-*12*	3-*13*	1-*14*	2-*145*
	2	2-*21*		4-*243*	2-*24*	2-*25*
	3	3-*31*	4-*342*		3-*34*	5-*345*
	4	1-*41*	2-*42*	3-*43*		1-*45*
	5	2-*541*	2-*52*	5-*543*	1-*54*	