

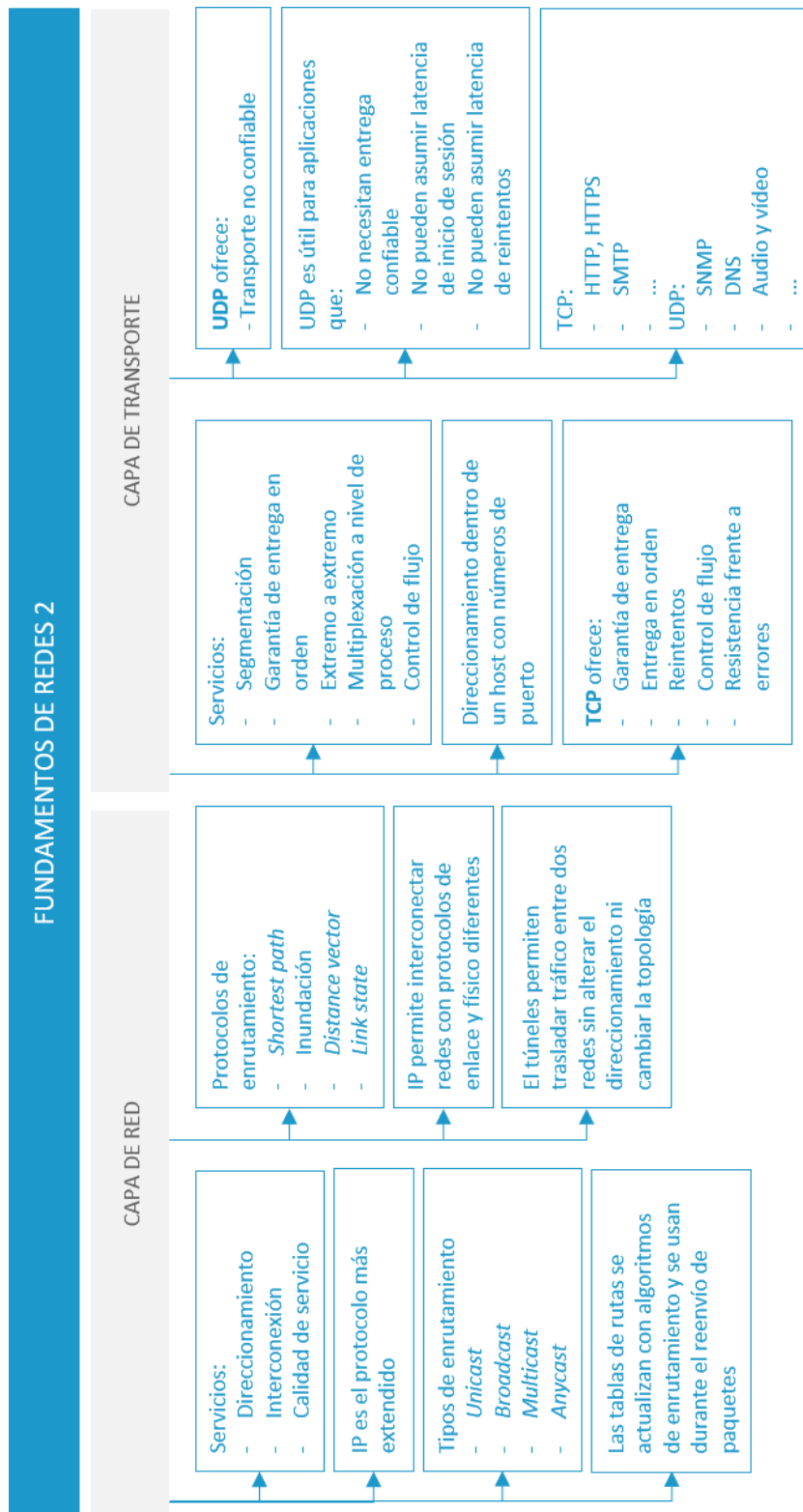
SecDevOps y Administración de Redes para Cloud

Fundamentos de redes 2

Índice

| | |
|---|----|
| Esquema | 3 |
| Ideas clave | 4 |
| 3.1. Introducción y objetivos | 4 |
| 3.2. Introducción a la capa de red | 5 |
| 3.3. Introducción a la capa de transporte | 29 |
| 3.4. Referencias bibliográficas | 42 |
| A fondo | 43 |
| Test | 44 |

Esquema



3.1. Introducción y objetivos

Este tema explica la **arquitectura de capas**. Se tratarán las **dos capas lógicas** que permiten que las aplicaciones se abstraigan de la mayoría de las complicaciones que se presentan en una red.

Los objetivos que se pretenden conseguir en este tema son:

- ▶ Presentar la capa de red y los servicios que ofrece.
- ▶ Introducir el concepto de dirección IP y subred.
- ▶ Presentar la capa de transporte y los servicios que ofrecen los dos protocolos principales.
- ▶ Poner de manifiesto las diferencias entre el protocolo de control de transmisión (TCP) y el protocolo de datagramas de usuario (UDP) y la necesidad de ambos.

En el siguiente vídeo, titulado **Capa de red y capa de transporte**, se resumen las características de las dos capas mencionadas.



Accede al vídeo

3.2. Introducción a la capa de red

La capa 3 del modelo de referencia es también llamada **capa de red**. Gestiona las opciones relacionadas con el *host* y el direccionamiento de red; la gestión de subredes e interconexión. También tiene la responsabilidad de **enrutar los paquetes** desde el origen hasta el destino tanto dentro como fuera de una **subred**.

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

Funciones

Los **dispositivos** que trabajan en la capa de red se centran en el **enrutamiento**. El enrutamiento incluye diversas tareas:

- ▶ Direccionar dispositivos y redes.
- ▶ Difundir tablas de enrutamiento o rutas estáticas.
- ▶ Mantener un sistema de colas para transmitirlos de acuerdo con la calidad de las restricciones del servicio establecidas para esos paquetes.
- ▶ Interconectar dos subredes diferentes.

Características de la capa de red

Con sus **funcionalidades estándar**, la capa 3 puede proporcionar varias características como:

- ▶ Gestión de la calidad de servicio.
- ▶ Balanceo de carga y gestión de enlaces.

- ▶ Interrelación de diferentes protocolos y subredes con diferentes esquemas.
- ▶ Diferentes diseños de red lógica sobre el diseño físico de la red.

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

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

Direccionamiento de red

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

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

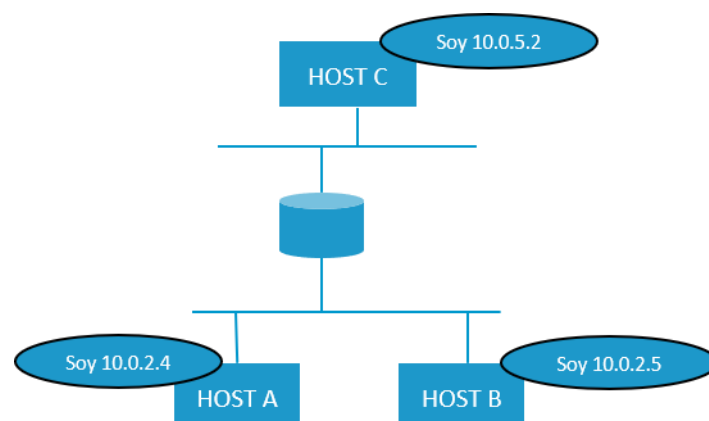


Figura 1. Direccionamiento de red. Fuente: elaboración propia.

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

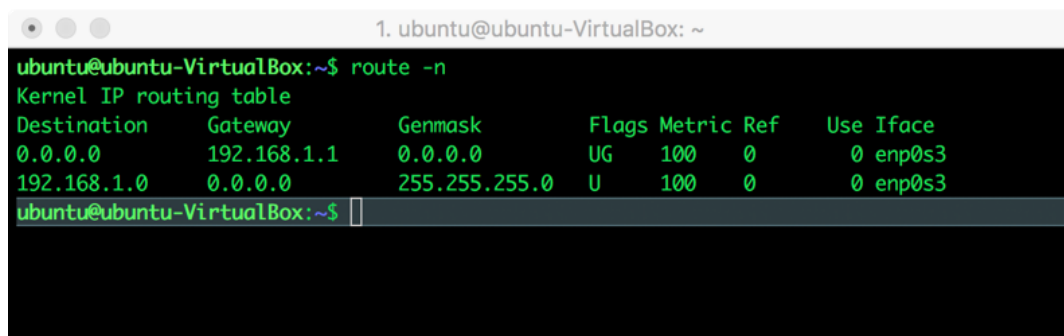
Los *hosts* en diferentes subredes necesitan un **mecanismo** para localizarse (el protocolo DNS ayudará en esa tarea, aunque no es estrictamente necesario para el funcionamiento de IP). Cuando un *host* adquiere la dirección IP del **host remoto**, envía los paquetes a su **gateway** o puerta de acceso.

El *gateway* será un **rúter** equipado con una **tabla de rutas** para poder decidir cómo enrutar el paquete al *host* de destino. Las tablas de rutas contienen la siguiente información:

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

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

La tabla de rutas puede ser tan sencilla como la de la Figura 2, extraída de un *host* con una única **interfaz de red**.



```
1. ubuntu@ubuntu-VirtualBox: ~  
ubuntu@ubuntu-VirtualBox:~$ route -n  
Kernel IP routing table  
Destination    Gateway         Genmask         Flags Metric Ref    Use Iface  
0.0.0.0        192.168.1.1    0.0.0.0         UG    100    0      0 enp0s3  
192.168.1.0    0.0.0.0        255.255.255.0   U     100    0      0 enp0s3  
ubuntu@ubuntu-VirtualBox:~$
```

Figura 2. Tabla de rutas de un *host* Linux con una interfaz. Fuente: elaboración propia.

La tabla de la Figura 2 se lee de la siguiente manera:

- Las direcciones que pertenecen a la subred 192.168.1.0, con la máscara de red 255.255.255.0, se encuentran accesible directamente (esto se representa con el valor 0.0.0.0 en la columna *gateway*).
- Para cualquier otra dirección, el siguiente salto es el *gateway* 192.168.1.1.

En un router con múltiples interfaces, algunas conectadas a redes locales y otras con enlaces a otros routers, la tabla puede ser mucho más completa. Por ejemplo, en un router [Juniper](#), la **tabla de rutas** podría ser parecida a lo siguiente:

```
0.0.0.0/0      *[Static/5] 00:51:57
               > to 172.16.5.254 via fxp0.0
1.1.1.30/32    [L-OSPF/10] 1d 00:01:01, metric 3
               > to 10.0.10.70 via lt-1/2/0.14, Push 800030
               to 10.0.6.60 via lt-1/2/0.12, Push 800030
1.1.1.40/32    [L-OSPF/10] 1d 00:01:01, metric 4
               > to 10.0.10.70 via lt-1/2/0.14, Push 800040
               to 10.0.6.60 via lt-1/2/0.12, Push 800040
1.1.1.50/32    [L-OSPF/10] 1d 00:01:01, metric 5
               > to 10.0.10.70 via lt-1/2/0.14, Push 800050
               to 10.0.6.60 via lt-1/2/0.12, Push 800050
10.0.0.1/32    *[Direct/0] 00:51:58
               > via at-5/3/0.0
10.0.0.2/32    *[Local/0] 00:51:58
               Local
10.13.13.13/32 *[Direct/0] 00:51:58
               > via t3-5/2/1.0
10.13.13.22/32 *[Direct/0] 00:33:59
               > via t3-5/2/0.0
127.0.0.1/32  [Direct/0] 00:51:58
               > via lo0.0
10.222.5.0/24 *[Direct/0] 00:51:58
               > via fxp0.0
10.222.5.81/32 *[Local/0] 00:51:58
               Local
```


Aunque con más entradas que la tabla de la Figura 2, la manera de leer esta tabla es similar. Hay **redes accesibles de manera local** sin necesidad de reenviar los paquetes a otros rúters (como 10.222.5.0/24), **redes remotas accesibles** mediante un salto (como 1.1.1.30/32) y una **ruta por defecto** para cualquier otra red no encontrada en la tabla (0.0.0.0/0).

Rutas

Cuando un dispositivo tiene **varias rutas** para llegar a un destino siempre selecciona una sobre las otras a partir de una cierta ruta. Este proceso de elección se denomina enrutamiento (o *routing*).

El **enrutamiento** no es una **funcionalidad exclusiva** de los rúters habituales: es posible configurar un equipo Linux o Windows como rúter, siempre y cuando este tenga **dos o más interfaces** (de lo contrario, el rúter no será muy útil) y se configure el software y la tabla de rutas correctamente.

Un rúter siempre está configurado con alguna **ruta predeterminada**. Esta le indica al enrutador **dónde enviar un paquete** si no hay ninguna ruta encontrada para un destino específico. En caso de que haya **múltiples rutas** existentes para llegar al mismo destino, el enrutador puede tomar una decisión basándose en la siguiente información:

- ▶ Número de saltos.
- ▶ Ancho de banda.
- ▶ Métrica estática.
- ▶ Prefijo-longitud.
- ▶ Retardo.

Las rutas pueden **configurarse estáticamente** o **generarse dinámicamente** mediante un protocolo como RIP o OSPF. Las métricas que dan prioridad a una ruta sobre otra pueden basarse en datos de rendimiento o a partir de un criterio del administrador.

La **dirección de red** puede ser una de las siguientes:

- ▶ **Unicast:** destinada a un *host* concreto.
- ▶ **Multicast:** destinado un grupo.
- ▶ **Broadcast:** destinada a todos los *hosts* de la red.
- ▶ **Anycast:** destinada al *host* más cercano de un grupo.

Un *rúter* nunca reenvía el tráfico de **broadcast** de forma predeterminada, ya que este suele estar confinado en una **red local**. Por ejemplo, el protocolo ARP usa *broadcast* en la dirección de red para mapear direcciones IP de nivel 3, a direcciones MAC de nivel 2. **Multicast** es utilizado, por ejemplo, para retransmisiones de audio y vídeo en tiempo real, de forma que, si N equipos quieren recibir un mismo flujo de datos, este puede enviarse una única vez desde el origen, y los *rúters* se encargarán de **duplicar** el flujo para que llegue a todos los destinos.

Una conexión **unicast** para este tipo de flujos implica que el origen va a emitir N flujos de salida, todos iguales, consumiendo más ancho de banda. **Anycast** es muy similar a **unicast**, a excepción de que los paquetes se entregan al destino más cercano, aun habiendo varios destinos disponibles.

Enrutamiento *unicast*

La mayor parte del tráfico en **Internet e intranets** se envía con un destino concreto y se conoce como tráfico **unicast**. El enrutamiento de datos *unicast*, a través de Internet, es la forma más simple de enrutamiento, porque el destino ya es conocido.

Por lo tanto, el *rúter* solo tiene que mirar la tabla de enrutamiento y enviar los datos hasta el **siguiente salto**. En la Figura 3, por ejemplo, el tráfico entre el *host* A y el *host* 1 solo involucra a ambos *hosts* y a los *rúters* que los conectan.

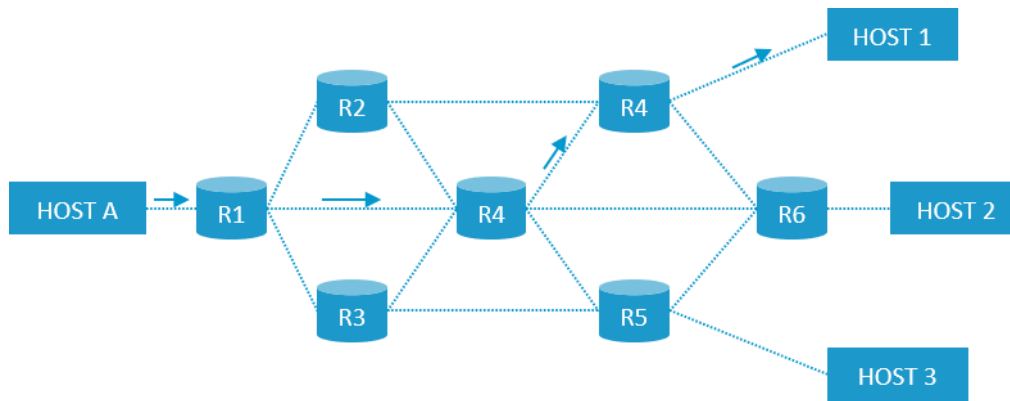


Figura 3. Enrutamiento *unicast*. Fuente: elaboración propia.

Enrutamiento *broadcast*

De forma predeterminada, los paquetes de *broadcast* **no son enrutados ni enviados** por los rúters en ninguna red. Los rúters crean dominios *broadcast*, pero pueden ser configurados para enviarlos en **casos especiales**. Un mensaje de *broadcast* está destinado a todos los **dispositivos de la red**.

Hay **dos algoritmos** principales para el enrutamiento *broadcast*:

- ▶ Un enrutador crea un **paquete de datos** y luego lo envía a cada *host* de uno en uno. En este caso, el enrutador crea **varias copias** de un único paquete de datos, cada una con una dirección de destino diferente. Todos los paquetes se envían como *unicast*, pero como se envían a todos los equipos, pareciera que el rúter estuviese transmitiendo en *broadcast*. Este método consume mucho **ancho de banda** y el rúter debe conocer la dirección de destino de cada nodo. Este método no exige demasiada CPU, pero puede causar problemas de **duplicados** si todos los rúters duplican los paquetes sin tener en cuenta los destinos que ya lo han recibido por otro camino.
- ▶ El reenvío de **ruta inversa**, o *reverse path forwarding*, es una técnica en la que el rúter conoce de antemano a su **predecesor** y sabe de dónde debería recibir el *broadcast*, de manera que es posible **detectar y descartar** duplicados.

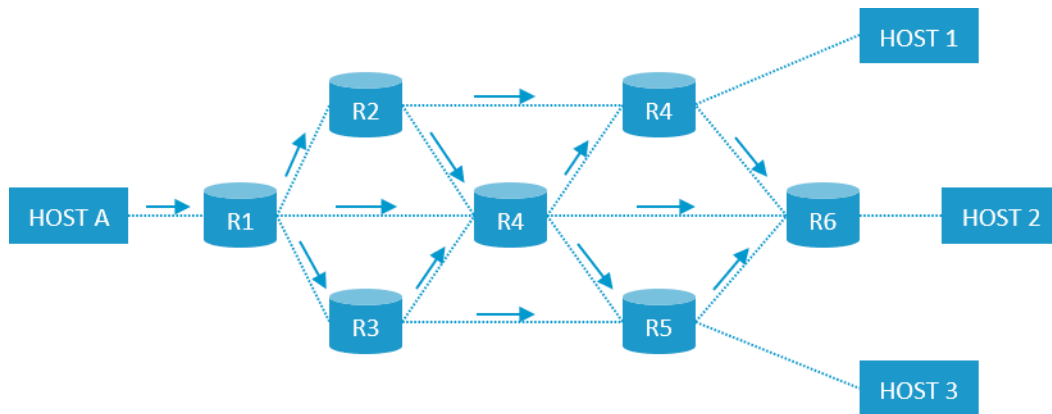


Figura 4. Enrutamiento *broadcast*. Fuente: elaboración propia.

Enrutamiento *multicast*

El enrutamiento *multicast*, también conocido como **multidifusión**, es un caso muy distinto a los anteriores y conlleva **nuevos desafíos**.

El enrutamiento *broadcast* podría usarse para hacer llegar un **flujo** a un subconjunto de equipos de la red, pero habría **tráfico no deseado**, ya que todos los nodos lo acabarían recibiendo.

El enrutamiento *multicast* busca enviar los datos solamente a los nodos que deseen recibirlos. En el diagrama de la Figura 5, por ejemplo, solo los **hosts 2 y 3** se han suscrito al flujo emitido por el **Host A**. Si la ruta a ambos nodos es **R1-R4-R6** y **R1-R2-R5**, respectivamente, los **rúters R2, R4 y R3** no tiene siquiera que recibir los paquetes del flujo.

Además, el **ancho de banda** usado en los enlaces indicados, con las flechas, es el necesario para un único flujo de paquetes del flujo. Si este tráfico fuera *unicast*, los enlaces **Host A-R1** y **R1-R4** estarían usando el doble de ancho de banda.

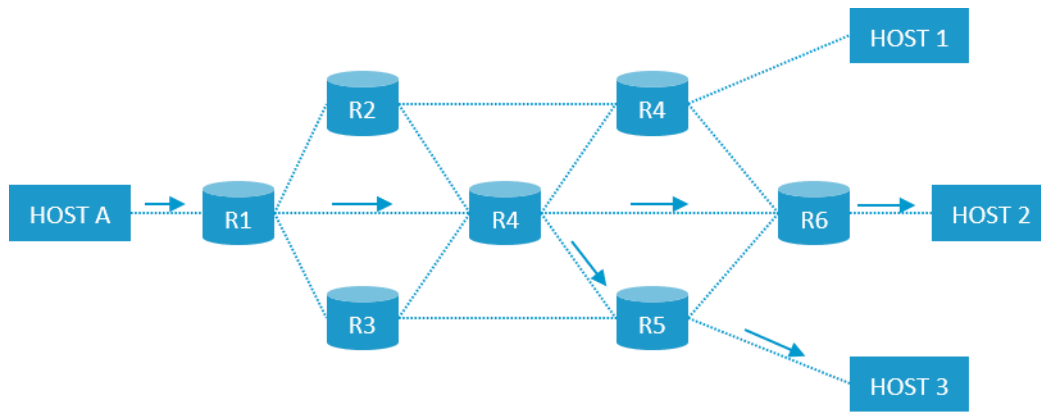


Figura 5. Enrutamiento *multicast*. Fuente: elaboración propia.

Los routers deben saber que hay nodos que desean recibir los **paquetes de multidifusión**, y entonces reenviar los **paquetes del flujo** solo a esos nodos. El ahorro de ancho de banda, especialmente en el enlace del **nodo emisor**, es considerable, ya que solo es necesario duplicar paquetes en aquellos routers involucrados en el camino de un nodo suscrito al flujo.

El **enrutamiento de multidifusión** también utiliza la técnica de reenvío de ruta inversa (*reverse path forwarding*) para detectar y descartar duplicados y bucles.

Enrutamiento *anycast*

El reenvío de paquetes *anycast* es un **mecanismo** en el que varios *hosts* de un grupo pueden tener la misma **dirección lógica**. Cuando se recibe un paquete, este se envía al *host* que esté más cerca en la **topología** de enrutamiento.

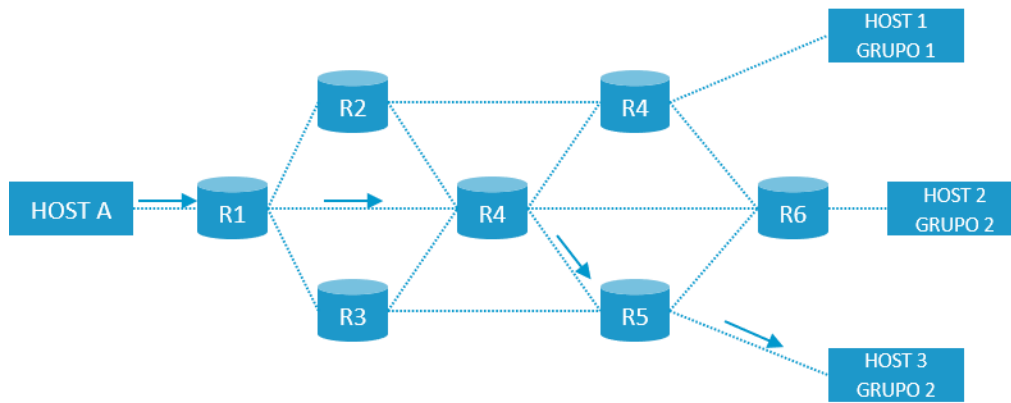


Figura 6. Enrutamiento *anycast*. Fuente: elaboración propia.

Este enrutamiento se realiza con la ayuda del **servidor DNS**. El servidor DNS responde a la solicitud con la **dirección IP más cercana** para el paquete *unicast*. Por ejemplo, en la Imagen 6, el nodo A envía un paquete *anycast* al grupo 2. El *Host 1* no recibirá el paquete por no pertenecer al grupo 2. Entre el *Host 2* y el *Host 3*, ambos del grupo 2, cualquiera podría recibir el paquete.

Asumiendo que el **enlace** entre R6 y el Host 2 tiene más latencia, por ejemplo, que el camino del *Host 1* al *Host 3*, el paquete sería entregado al **Host 3**.

Algoritmos de enrutamiento

Para cada **tipo de enrutamiento** hay **diversos algoritmos**, definidos como la **funcionalidad** de la red que decide a qué enlace redirigir un paquete. Se puede entender como el **proceso** que actualiza las tablas de rutas de un router; puede ser un proceso sencillo si se usan rutas estáticas, o muy complejo si es dinámico y hay **múltiples métricas** a considerar.

El proceso de **reenvío** hace uso de las tablas de rutas una vez actualizadas por el **algoritmo** de enrutamiento. Los siguientes apartados presentan algunos algoritmos de enrutamiento *unicast*.

Camino más corto (*shortest path*)

La **decisión de enrutamiento** en las redes se toma, principalmente, sobre la base del coste de **transmitir** un paquete entre la fuente y el destino. La **métrica** que decide la **ruta óptima** es el número de saltos. *Shortest path* es una técnica que utiliza varios algoritmos para decidir qué camino tiene el número mínimo de saltos. Los algoritmos más habituales son Dijkstra's, Bellman Ford y Floyd Warshall.

Inundación (*flooding*)

Este es el método más simple de **reenvío** de paquetes. Cuando se recibe un paquete, los enrutadores lo envían a todas las **interfaces**, excepto a aquella de la que se recibió. Por tanto, se puede considerar que este método es, simplemente, una **técnica de reenvío** más que un algoritmo de enrutamiento como tal.

Crea demasiada **carga en la red** y da lugar a muchos **paquetes duplicados**, consumiendo recursos por la red. Una manera de limitarlo es fijando un **tiempo de vida máximo** o TTL (*time to live*) en los paquetes, contabilizado como número de saltos (efectivamente, el TTL no marca un tiempo máximo, sino un número de saltos máximo). Al reenviar un paquete, el router reduce en 1 el TTL. Si un router recibe un paquete con TTL=0 sin haberlo entregado en el destino, simplemente, lo **descarta**.

Vectores de distancia (*distance vector*)

Es un **protocolo de enrutamiento** que decide sobre el número de saltos que debe haber entre origen y destino. La ruta que contenga la **menor cantidad de saltos** es considerada como la **mejor ruta**.

Cada rúter anuncia sus **mejores rutas** establecidas a otros rúters. En última instancia, todos los enrutadores construyen su **topología de red**, basada en la información que les proveen sus rúters vecinos. RIP, de *Routing Information Protocol*, es un ejemplo de protocolo que usa un algoritmo de enrutamiento basado en **vectores de distancia**.

Estado de enlace (*link state*)

El protocolo de **estado de enlace** es, ligeramente, más complicado que el **vector de distancia**. Toma en cuenta los estados de enlaces de todos los rúters en una red. Esto ayuda a las rutas a construir un **gráfico común** de toda la red. Los rúters calculan luego su mejor camino para fines de enrutamiento.

Algunos protocolos que usan este método son *Open Shortest Path First (OSPF)* e *Intermediate System to Intermediate System (ISIS)*.

Interconexión

En el mundo real, las **redes administradas** bajo la misma organización están, en general, **dispersas** geográficamente. Puede que exista el requisito de conectar dos redes diferentes del mismo tipo. El enrutamiento entre dos redes se llama **interconexión**.

En una topología de interconexión, los equipos tienen que estar preparados para permitir que un paquete **atraviere diferentes redes**. En el ejemplo de la Figura 7, un paquete enviado desde el destino tiene que atravesar una red **Wifi**, un **circuito virtual MPLS** y una red **Ethernet**, antes de llegar al destino.

El **protocolo IP** sirve de capa de abstracción en esta **topología**: el nivel de enlace añadirá cabeceras específicas en cada salto, pero serán las que permitan decidir el salto siguiente en cada dispositivo, identificando, unívocamente, el equipo de destino.

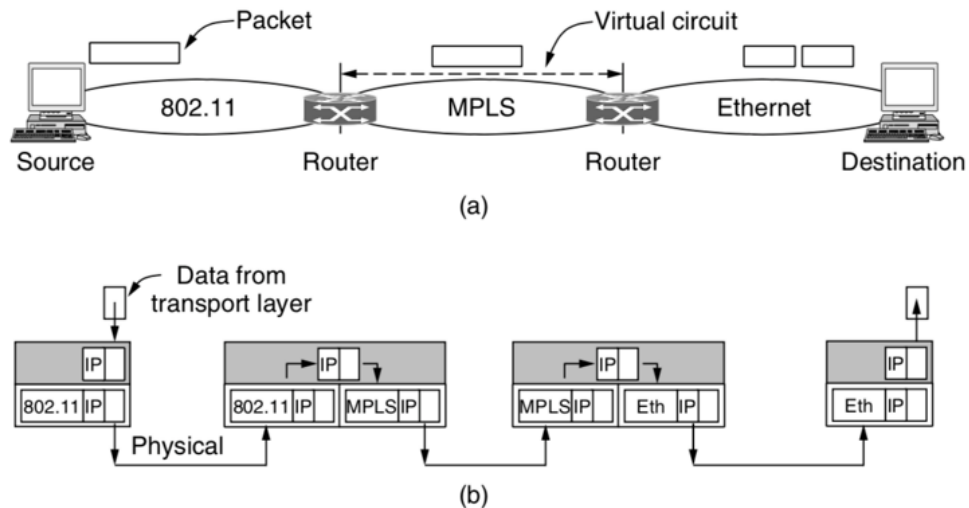


Figura 7. A) interconexión de redes; b) cabeceras de red y enlace en una interconexión. Fuente: Tanenbaum y Wetherall, 2011.

Túneles

Si hay **dos redes**, geográficamente separadas, que quieren comunicarse entre sí, hay **dos opciones** para conectarlas:

- ▶ Desplegar una línea específica entre ambas.
- ▶ Enviar sus datos a través de redes intermedias.

El **despliegue de túneles**, o *tunneling*, es una técnica que consiste en **encapsular un protocolo de red sobre otro** (protocolo de red encapsulador), creando un túnel de información dentro de una red de computadoras. Ambos extremos del túnel están configurados de manera específica.

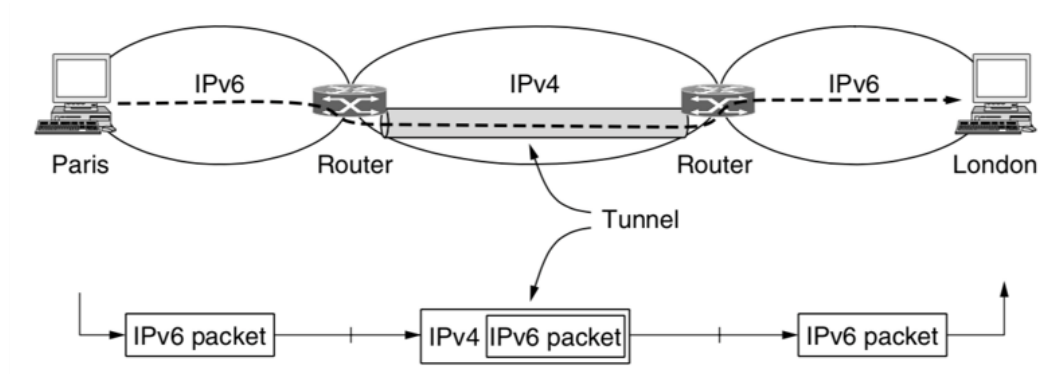


Figura 8. Túnel IPv4 entre dos redes IPv6. Fuente: Tanenbaum y Wetherall, 2011.

Cuando los datos entran desde un extremo del túnel, se etiquetan con unas **cabeceras adicionales**. Estos datos etiquetados son **enrutados** en el interior de una **red intermedia** o **de tránsito** hasta llegar al otro extremo del túnel. Cuando la información sale del túnel, se elimina su etiqueta y se entrega en otra parte de la red. Ambos extremos parecen estar conectados directamente y el etiquetado hace que los datos viajen a través de la red **sin modificaciones**.

Los túneles **alteran la arquitectura de capas**. En el ejemplo de la Figura 8, el contenido de los paquetes IPv4 no son tramas del nivel de transporte, como correspondería, sino **tramas de nivel de red** del tráfico que se desea transmitir por el túnel. La Figura 9 muestra la **estructura de capas** de este ejemplo. Como se puede ver, hay dos capas de nivel 3.

La arquitectura de capas no es más que un modelo que facilita el diseño y el análisis, así que, a efectos prácticos, esta alteración no supone ningún problema; es más, es muchas veces la forma más fácil de conectar dos redes sin apenas introducir modificaciones en los *hosts*.

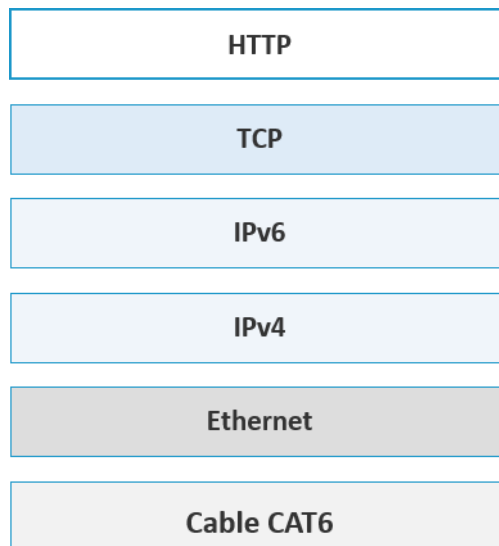


Figura 9. Arquitectura de capas en un túnel. Fuente: elaboración propia.

Fragmentación de paquetes

La mayoría de los segmentos *Ethernet* tienen su **unidad de transmisión máxima** o *máximum transmisión unit* (MTU) fijada en 1500 bytes. Un **paquete de datos** puede tener mayor o menor tamaño dependiendo de la aplicación. Los dispositivos en tránsito también tienen su **hardware y software** con sus respectivas **capacidades** que indican qué **cantidad de datos** maneja el dispositivo, así como el tamaño de los paquetes que pueden procesar.

Si el tamaño del paquete recibido de datos es **menor o igual** al tamaño máximo que se puede procesar, la red de tránsito puede manejarlo. En cambio, si el paquete es **más grande**, se divide en **varias piezas** de menor tamaño antes de enviarlo. Esto se llama **fragmentación de paquetes**. Cada fragmento contiene la misma dirección de destino y de origen y se encamina a través de la ruta de tránsito.

En el extremo receptor se monta de nuevo, **uniendo** cada uno de sus fragmentos para formar el **paquete original**. Opcionalmente, un paquete puede incluir un **modificador DF** (*do not fragment*) para evitar que los nodos intermedios lo fragmenten.

Si el paquete se fragmenta en demasiadas piezas, la **sobrecarga** se **incrementa**. Si el paquete no se fragmenta las veces necesarias, y las piezas son demasiado grandes, el rúter intermedio puede **no ser capaz de procesarlo** y podría **perderse**.

Protocolos de la capa de red

Cada ordenador en una red tiene una **dirección IP exclusiva** que permite **identificarla**. Una dirección IP es la dirección lógica de capa 3. Esta dirección puede **cambiar** cada vez que se reinicia el equipo: un ordenador puede tener una IP en determinado momento y otra IP en otro.

ARP o *Address Resolution Protocol*

Durante la comunicación, un *host* necesita la dirección de capa 2, conocida como **MAC**, del siguiente salto del paquete. Este salto puede ser el *host* de destino, si la comunicación no sale de la propia subred, o el *gateway* que el *host* ha decidido usar para salir de su subred, si el destino está en una red remota.

La dirección MAC está físicamente grabada en la **tarjeta de interfaz** de red de un equipo y nunca cambia. En un entorno virtualizado, una **NIC virtual** recibe una **MAC del hipervisor**. Esto puede provocar que haya MAC duplicadas, ya que la asignación es aleatoria, pero los hipervisores tienen mecanismos para evitar que las MAC se repitan en una misma **red virtual**. En cualquier caso, la NIC virtual de una máquina virtual (VM) no suele cambiar a lo largo de su vida.

Para conocer la dirección MAC del *host* remoto en una subred, el ordenador de origen envía un mensaje *broadcast* ARP preguntando qué *host* tiene dirección de IP. Debido a que es un mensaje *broadcast*, todos los *hosts* del segmento de red, que equivale a un dominio de *broadcast*, reciben este paquete y lo procesan.

El **paquete ARP** contiene la dirección IP del *host* de destino al que el *host* de origen quiere contactar. Cuando un *host* recibe un paquete ARP que se le ha destinado, responde con su propia **dirección MAC**. Una vez que el *host* recibe la dirección MAC de destino, puede comunicarse con el **host remoto** utilizando el protocolo de enlace.

Este **mapeo del MAC al IP** se guarda en una caché de ambos *hosts*. Si en el futuro necesitan comunicarse pueden, directamente, dirigirse a su **caché ARP**.

Internet Control Message Protocol (ICMP)

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

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

Los mensajes de **ICMP-echo** y **ICMP-echo-reply** son los mensajes ICMP más comúnmente utilizados para comprobar la **accesibilidad** de los *hosts* de extremo a extremo (son más conocidos como *ping*, y así se llama la **utilidad** en los sistemas operativos).

Cuando un *host* recibe una **solicitud ICMP-echo**, está obligado a enviar una respuesta **ICMP-echo**. No obstante, es habitual que muchos *firewalls* de red y de *host* descarten los mensajes de *ping* como medida de seguridad; por el hecho de que un equipo no conteste a un *ping*, no significa que no sea accesible.

IPv4

IPv4 sigue un **esquema de direccionamiento de 32 bits**, utilizado como mecanismo de direccionamiento en el modelo TCP/IP. Las direcciones IP son **jerárquicas**, a diferencia de las direcciones *Ethernet*. Cada dirección de 32 bits se compone de una porción de red de **longitud variable** en los bits más significativos y una porción de *host* en los bits menos significativos.

La porción de red tiene el mismo valor para todos los *hosts* en una red única, como una **LAN Ethernet**. Esto significa que una red corresponde a un **bloque contiguo** de espacio de direcciones IP. Este bloque se denomina **prefijo**.

| | | | | | | |
|-----------------|---|-----------------|---|-----------------|---|-----------------|
| 172 | . | 16 | . | 254 | . | 1 |
| 1 0 1 0 1 1 0 0 | . | 0 0 0 1 0 0 0 0 | . | 1 1 1 1 1 1 1 0 | . | 0 0 0 0 0 0 0 1 |
| 255 | . | 255 | . | 0 | . | 0 |
| 1 1 1 1 1 1 1 1 | . | 1 1 1 1 1 1 1 1 | . | 0 0 0 0 0 0 0 0 | . | 0 0 0 0 0 0 0 0 |
| 172 | . | 16 | . | 0 | . | 0 |

Figura 10. Dirección IP, su representación en octetos y bits, la máscara de red y la dirección de red. Fuente: elaboración propia.

Las direcciones IP se escriben en **notación decimal** con puntos. En este formato, cada uno de los 4 *bytes* se escribe en decimal, de 0 a 255. Por ejemplo, la dirección hexadecimal de 32 bits AC10FE01 se escribe como 172.16.254.1.

Los **prefijos** se escriben dando la **dirección IP más baja** en el bloque y el tamaño del bloque. Por convenio, el tamaño del prefijo en bits se escribe después de la dirección IP. Así, si la IP anterior pertenece a una red con un prefijo de 16 bits, el prefijo de red se indicaría como 172.16.0.0/16.

La **longitud del prefijo** corresponde a una **máscara binaria** de unos en la parte de la red, llamada máscara de red. Si se aplica un *AND* lógico entre la dirección IP y su máscara, se obtiene el **bloque de red**. En el ejemplo, con un prefijo de 16 bits, la máscara de subred es 255.255.0.0, que consisten en 2 *bytes* de unos seguido de 2 *bytes* a cero.

Dado que la longitud del prefijo no se puede **inferir** solo a partir de la dirección IP, los **protocolos de enrutamiento** incorporan los prefijos en la información que los rúters intercambian entre sí. La Figura 10 muestra la IP del ejemplo con su máscara y su dirección de red, tanto en **binario** como en **decimal**.

Las direcciones jerárquicas tienen ventajas y desventajas significativas. La ventaja clave de los prefijos es que los *rúters* pueden reenviar paquetes basados únicamente en la porción de red de la dirección, siempre que cada una de las redes tenga un bloque de dirección único.

La parte del *host* **no es relevante** para los rúters porque todos los *hosts* en la misma red se enviarán en la **misma dirección**. Solo cuando los paquetes llegan a la red a la que están destinados, se reenvían al *host* correcto. Esto hace que las tablas de enrutamiento sean mucho más pequeñas de lo que serían de otro modo. Mientras que el número de *hosts* en Internet se acerca a los **mil millones**, los rúters necesitan mantener rutas para alrededor de **300 000 prefijos**.

Si bien el uso de una jerarquía permite escalar el enrutamiento de Internet, tiene **dos desventajas**.

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

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

Subredes

Las direcciones IP están divididas en **tres categorías principales**:

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

| Clase | Bits de red | # redes | nodos/red | Primera dirección | Última dirección |
|-------|-------------|---------|------------|-------------------|------------------|
| A | 8 | 128 | 16.777.216 | 0.0.0.0 | 127.255.255.255 |
| B | 16 | 16384 | 65.536 | 128.0.0.0 | 191.255.255.255 |
| C | 24 | 2097152 | 256 | 192.0.0.0 | 223.255.255.255 |

Tabla 1. Clases de subredes IP. Fuente: elaboración propia.

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

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

Tabla 2. Rangos de redes privadas. Fuente: elaboración propia.

Gracias a que IP permite la **agregación jerárquica**, las direcciones IP pueden estar contenidas en prefijos de diferentes tamaños. La misma dirección IP que un router trata como parte de un **bloque /22** (que contiene 2^{10} direcciones) puede ser tratada por otro router como parte de un **bloque /20** más grande (que contiene 2^{12} direcciones).

Depende de cada router tener la información de prefijo correspondiente. Este diseño funciona a base de subredes y se llama *Classless Inter-Domain Routing* (CIDR) o **enrutamiento entre dominios sin clase**. CIDR permite el diseño de jerarquía de red, sin seguir el tamaño estricto de las clases A, B y C.

NAT

Las direcciones IP son **escasas**. Si un ISP dispone tener un **bloque /16**, dándole 65 534 direcciones, el número de clientes que puede tener está **limitado** a ese número. Un mecanismo para resolver este problema es **NAT**.

La idea básica detrás de NAT (*Network Address Translation*, traducción de direcciones de red) es que el ISP asignará a cada cliente una **única dirección IP** para el tráfico de Internet. Dentro de la red del cliente, cada equipo obtiene una dirección IP única, que se utiliza para **enrutar el tráfico local**, es decir, entre dos equipos del cliente.

Sin embargo, en el tráfico dirigido a Internet, justo antes de que un paquete salga de la red del cliente y vaya al ISP, se realiza una **traducción** de la **dirección IP interna única** a la **dirección IP pública compartida**. Esta traducción hace uso de tres rangos de direcciones IP que se han declarado como privadas (mostradas previamente en la Tabla 2). Los usuarios pueden usarlas internamente como lo deseen. La única regla es que ningún paquete que contenga estas direcciones puede aparecer en Internet.

El **funcionamiento** de NAT se muestra en la Figura 11. Dentro de las instalaciones del cliente, cada máquina tiene una dirección única en la red 10.x.y.z. Sin embargo, antes de que un paquete salga de las instalaciones del cliente, pasa a través de un **equipo con soporte de NAT** que convierte la dirección IP de origen interna, 10.0.0.1 en la Figura 11, a la dirección pública del cliente, 198.60.42.12. La caja NAT a menudo se **integra** en un solo dispositivo con un *firewall* o un *rúter*.

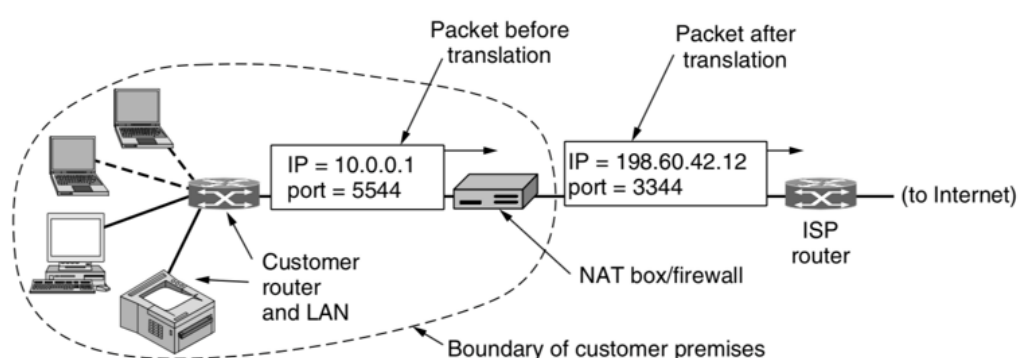


Figura 11. Funcionamiento de NAT. Fuente: Tanenbaum y Wetherall, 2011.

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

Cuando un **proceso** quiere establecer una **conexión TCP** con un **proceso remoto**, abre un **socket** en puerto TCP no utilizado en su propia máquina. Este puerto de origen identifica al proceso que ha iniciado la conexión en la **capa TCP**. El proceso también proporciona un **puerto de destino** para indicar a quién entregar los paquetes en el lado remoto, por ejemplo, el puerto 443 de un **servidor web seguro**. Estos puertos sirven para identificar los procesos, utilizando la conexión en ambos extremos.

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

Es necesario **reemplazar** el puerto de origen porque las conexiones de dos máquinas del cliente pueden utilizar mismo puerto de origen, sin por ello entrar en **conflicto**. Cuando un paquete llega al NAT desde el ISP, el puerto de origen en el encabezado TCP se utiliza como índice en la tabla de traducciones para localizar la dirección IP interna y el puerto de origen TCP original.

La dirección y el puerto se **insertan** en el paquete, se vuelven a calcular los *checksums* y el paquete se pasa al router (que puede ser el propio dispositivo NAT) para la entrega normal, utilizando la dirección 10.x.y.z.

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

- ▶ **Viola la arquitectura de IP** en la que cada dirección IP identifica, de forma única, una máquina en todo el mundo.
- ▶ **Rompe el modelo de conectividad** de extremo a extremo de Internet que dice que cualquier *host* puede enviar un paquete a cualquier otro *host* en cualquier momento. Dado que la tabla de traducciones de un equipo NAT se configura a partir de los paquetes salientes, los paquetes entrantes no pueden aceptarse hasta después de los salientes. Es decir, los equipos locales no son **enrutables** públicamente, a menos que hayan iniciado una **conexión** previamente.
- ▶ De manera efectiva, un **protocolo de red no orientado a conexión** se convierte en un tipo peculiar de red orientada a la conexión. Si el equipo NAT **falla**, y se pierde su **tabla de mapeo**, se destruyen todas sus conexiones TCP, aun cuando TCP está preparado para soportar la caída de equipos intermedios.
- ▶ NAT viola la regla más fundamental de la arquitectura de capas: la **capa N** puede no hacer suposiciones sobre lo que la **capa N+1** ha puesto en el campo de carga útil. Este principio básico está ahí para mantener las **capas independientes**. Además, los procesos que quieren comunicarse a través de Internet no están obligados a usar TCP o UDP. Sin embargo, NAT no soportará tráfico que use un protocolo diferente en la capa de transporte.
- ▶ Algunas aplicaciones usan **múltiples conexiones TCP o puertos UDP** en una misma conexión. Por ejemplo, FTP inserta direcciones IP y puertos TCP en el cuerpo del paquete, no en la cabecera para que el receptor las extraiga y use (los detalles de FTP se estudiarán en el siguiente tema). Dado que NAT no tiene por qué **analizar el tráfico de la aplicación**, no puede reescribir las direcciones IP, ni tenerlas en cuenta.

IPv6

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

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

IPv6 está todavía en **fase de transición** y se espera que reemplace completamente al IPv4 en los próximos años. En la actualidad, hay **pocas redes** que usen IPv6 nativamente sin depender de IPv4.

Hay algunos mecanismos de transición disponibles para que las redes IPv6 puedan hablar fácilmente en IPv4:

- ▶ Implantación dual.
- ▶ Túnel.
- ▶ NAT-PT.

3.3. Introducción a la capa de transporte

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

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

| | |
|--|---|
| FUNCIONES DE LA CAPA DE TRANSPORTE | <ul style="list-style-type: none"> ■ Es la primera que divide los datos suministrados por la capa de aplicación en unidades más pequeñas, llamadas segmentos. |
| | <ul style="list-style-type: none"> ■ En el caso de TCP, garantiza que los datos se reciban en la misma secuencia en la que fueron enviados. |
| | <ul style="list-style-type: none"> ■ Realiza una entrega de extremo a <i>extremo</i> (<i>end-to-end</i>) entre <i>hosts</i> que pueden, o no, pertenecer a la misma subred. |
| | <ul style="list-style-type: none"> ■ Todos los procesos del servidor que intentan comunicarse a través de la red están equipados con los puntos de acceso a los servicios de transporte (TSAP), también conocidos como números de puerto de red. |

Tabla 3. Funciones de la capa de transporte. Fuente: elaboración propia.

Comunicación de extremo a extremo

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

La **capa de transporte** ofrece dos protocolos principales:

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

TCP

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

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

Cabeceras

La **longitud del encabezado TCP** es de 20 *bytes* como mínimo y 60 *bytes* como máximo.

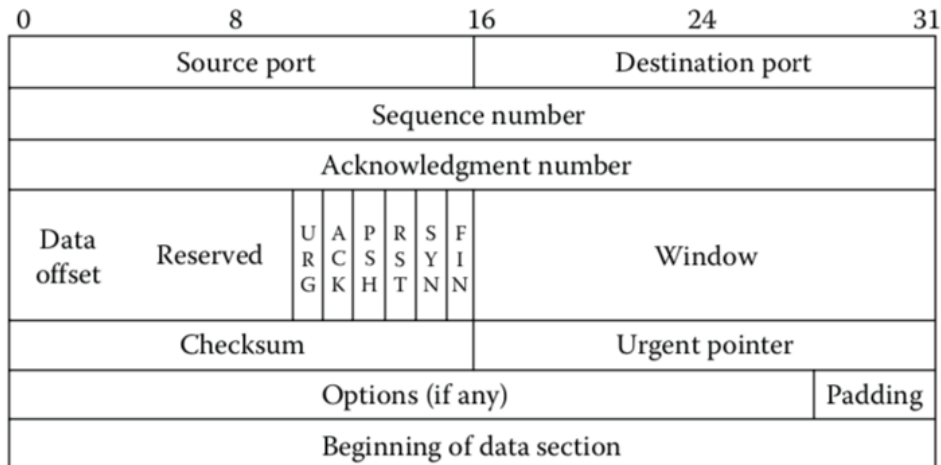


Figura 12. Cabecera TCP. Fuente: Chao, 2016.

- ▶ **Puerto de origen (16 bits):** identifica el puerto de origen del proceso de aplicación en el dispositivo emisor.
- ▶ **Puerto de destino (16 bits):** identifica el puerto de destino de la aplicación en proceso en el dispositivo receptor.
- ▶ **Número de Secuencia (32 bits):** número que identifica el segmento actual en una sesión TCP.
- ▶ **Número de acuse de recibo (*acknowledgement number*, 32 bits):** cuando está activado, este campo contiene el siguiente número de secuencia de *bytes* de datos esperado y funciona como confirmación de recepción de los datos anteriormente recibidos.

- ▶ **Desplazamiento de datos (*data offset*, 4 bits):** indica tanto el tamaño del encabezado TCP (en palabras de 32 bits) como el desplazamiento de los datos en el paquete actual en todo el segmento TCP.
- ▶ **Reservado (3 bits):** bits reservados para uso futuro y siempre a cero de forma predeterminada.
- ▶ **Flags (1 bit cada una):**
 - **URG:** indica que el campo urgente del puntero tiene datos significativos y debe ser procesado.
 - **ACK:** indica que el campo *acknowledgement number* tiene datos válidos. Si ACK está 0, el paquete no contiene ningún ACK.
 - **PSH:** cuando está activado, indica que es una petición a la estación receptora para que haga un PUSH datos tan pronto llegue a la aplicación receptora.
 - **RST:** *reset* se utiliza para rechazar una conexión entrante, para rechazar un segmento y para reiniciar una conexión.
 - **SYN:** este indicador se utiliza durante el arranque de la sesión.
 - **FIN:** este indicador se utiliza durante el arranque de la sesión.
- ▶ **Window:** este campo se utiliza para el control de flujo entre dos estaciones e indica el tamaño del búfer, en *bytes*, que el receptor ha asignado para un segmento. Por ejemplo, cuántos datos espera el receptor.
- ▶ **Checksum:** este campo contiene la suma de comprobación del encabezado y los datos.
- ▶ **Opciones:** facilita opciones adicionales que no están cubiertas por el protocolo, explícitamente. El campo de opción siempre se describe en palabras de 32 bits. Si este campo contiene datos de menos de 32 bits, se completan los bits con *padding* para llegar al tamaño de 32 bits.

Direccionamiento

La **comunicación TCP** entre dos *hosts* remotos se realiza mediante **números de puerto** (TSAP). Los números de los puertos se establecen entre 0 a 65 535, tal como permite el tamaño de 16 bits que tiene el campo en la **cabecera**. Se clasifican de la siguiente manera:

- ▶ Puertos del sistema: 0 – 1023.
- ▶ Puertos de usuario: 1024 – 49 151.
- ▶ Puertos privados/dinámicos: 49 152 – 65 535.

Gestión de conexiones

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

Establecimiento

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

El cliente, después de recibir el ACK de su segmento, envía un **acuse de recibo** de la respuesta del servidor.

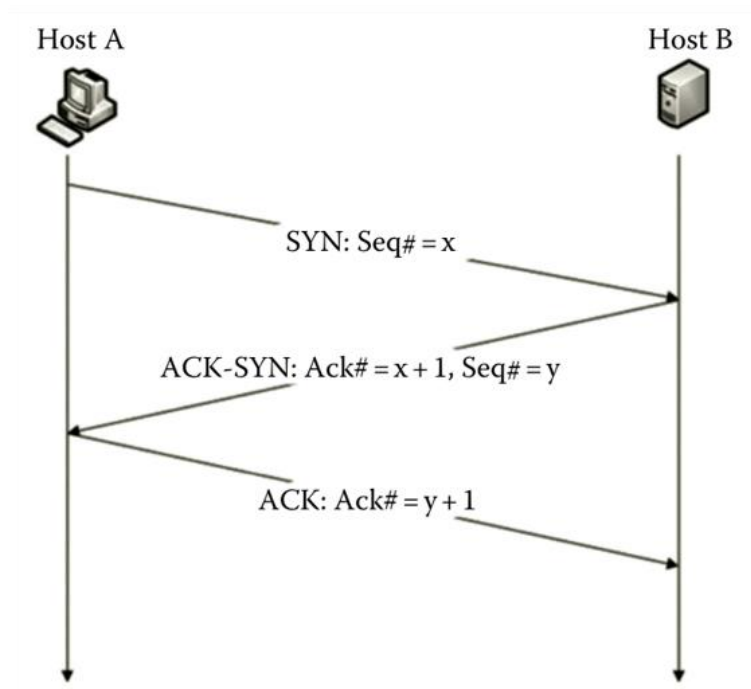


Figura 13. Establecimiento de sesión TCP: 3-way handshake. Fuente: Chao, 2016.

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

Gestión del ancho de banda

TCP utiliza el concepto de **tamaño de ventana** para administrar la necesidad de ancho de banda. El tamaño de la ventana indica al remitente, en el extremo remoto, el **número de datos** en *bytes* que el receptor en este extremo puede recibir.

TCP utiliza una **fase de inicio lento**, asignando al tamaño de la ventana el valor 1 y aumentando el tamaño de la ventana de forma exponencial después de cada transferencia exitosa.

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

Control de errores y control de flujo

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

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

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

Multiplexación

Cuando un cliente TCP inicializa una **conexión con un servidor**, siempre alude a un número de puerto definido que indica el proceso de aplicación. La sesión TCP también define un **puerto** en el *host* de origen, en este caso, escogido al azar de la lista de puertos privados que no están en uso.

El hecho de que varios **procesos** puedan usar números de puertos diferentes sirve en efecto de multiplexación a **nivel TCP**: un cliente puede comunicarse con varios procesos de aplicación de un único *host*, incluso aunque ese *host* disponga de una única **dirección de nivel 3** y de una **única interfaz**.

Timeouts

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

| | |
|--|---|
| Temporizador de validez (<i>keep alive</i>) | Este temporizador se utiliza para comprobar la integridad y la validez de una conexión. Cuando expira el tiempo, el <i>host</i> envía una sonda para comprobar si la conexión todavía existe. |
| Temporizador de retransmisión | Este temporizador mantiene una sesión con estado de los datos enviados. Si el acuse de recibo de los datos enviados no se recibe dentro del tiempo de retransmisión, el segmento de datos se vuelve a enviar. |
| Temporizador de persistencia | la sesión TCP puede ser pausada por cualquiera de los <i>hosts</i> , enviando un tamaño de ventana 0. Para reanudar la sesión, un <i>host</i> debe enviar un tamaño de ventana más grande. Cuando expira este temporizador, el <i>host</i> reenvía su tamaño de ventana para que el otro extremo lo sepa. Este temporizador ayuda a evitar los estancamientos en la comunicación. |
| Tiempo de espera | Después de establecer una conexión, ambos <i>hosts</i> esperan un tiempo determinado para finalizar la conexión completamente. Esto es para asegurarse de que el otro extremo ha recibido su solicitud de terminación de conexión. El tiempo de espera máximo puede ser de hasta 240 segundos. |

Tabla 4. Temporizadores de TCP. Fuente: elaboración propia.

Recuperación ante los fallos

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

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

Protocolo de datagrama de usuario

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

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

Requerimiento de UDP

¿Por qué es necesario un protocolo poco fiable para transportar datos? UDP se usa allí donde los **paquetes de confirmación, o ACK**, comparten una cantidad significativa de ancho de banda o añaden latencia.

Por ejemplo, en el caso de la transmisión de vídeo, se envían **miles de paquetes** a los usuarios. Si hubiera que pedir **acuse de recibo** de todos los paquetes, se usaría una gran cantidad de ancho de banda. Además, si se **pierde un paquete**, puede no ser necesario reenviarlo. En un canal de audio es preferible **no entregar** parte del contenido (es decir, no reproducir un pequeño intervalo de tiempo) que detener la reproducción completamente y reproducirlo más tarde. De cara a la interacción en tiempo real de usuarios, UDP ofrece una **mejor experiencia**.

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

Características

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

Encabezado UDP

El encabezado UDP es tan simple como sus funciones y contiene únicamente 4 campos.

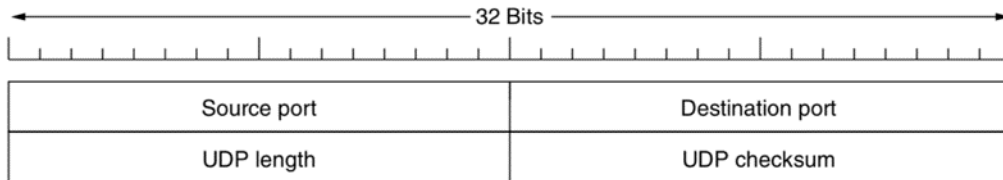


Figura 14. Encabezado UDP. Fuente: Tanenbaum y Wetherall, 2011.

Los **campos** son:

- ▶ **Puerto de origen** (16 bits): se utiliza para identificar el puerto de origen del paquete, igual que en TCP.
- ▶ **Puerto de destino** (16 bits): se utiliza para identificar el servicio del nivel de la aplicación en la máquina de destino.
- ▶ **Longitud** (16 bits): este campo especifica toda la longitud del paquete UDP, incluyendo la cabecera.
- ▶ **Checksum**: este campo almacena el valor de la suma de comprobación generada por el remitente antes de enviar. IPv4 tiene este campo como opcional, de manera que, si se transmiten datos sin *checksum*, este campo tiene su valor a cero.

Aplicaciones UDP

Los siguientes son **algunos** de los protocolos que usan UDP como protocolo de nivel 4:

- ▶ DNS.
- ▶ SNMP.
- ▶ TFTP.

- ▶ RIP.
- ▶ Kerberos.

Otros protocolos de nivel 4

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

▶ UDP-Lite.

UDP-Lite (*Lightweight User Datagram Protocol*), por ejemplo, es un protocolo, **no orientado a conexión**, que permite entregar una carga útil de datos potencialmente dañada a una aplicación. Esto es útil para sistemas en los que es preferible tomar las decisiones sobre la integridad de los datos en la propia **capa de aplicación**, donde se entiende la importancia de los bits, y no en la **capa de transporte**.

UDP-Lite se basa en **UDP**, pero, a diferencia de este, donde todos o ninguno de los bits del paquete están protegido por la **suma de verificación** (el *checksum*), UDP-Lite permite proteger con **checksums** parciales que solo cubren parte del paquete de datos y, por lo tanto, puede llegar a entregar paquetes que se hayan dañado parcialmente.

Está diseñado para **protocolos multimedia**, como voz sobre IP o flujos de vídeo, en los que recibir un paquete con una carga dañada es mejor que no recibir ningún paquete. Para el **UDP y TCP**, un solo bit con error invalidará la suma de verificación, lo que significa que se debe descartar todo el paquete.

De alguna manera, los errores de bit se convierten en **errores de paquete completo**, incluso cuando el daño a los datos es trivial. Para calcular la suma de comprobación, UDP-Lite utiliza el mismo algoritmo que UDP.

► Sctp.

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

Cuando no hay **soporte nativo de SCTP** en el sistema operativo, es posible hacer un túnel SCTP sobre UDP. También es posible asignar llamadas de sistema de TCP a llamadas SCTP para que las aplicaciones existentes puedan usar SCTP sin modificación. SCTP se ofrece como **alternativa** a TCP y UDP porque algunas aplicaciones necesitan una transferencia garantizada sin mantener el orden.

El **bloqueo de TCP**, hasta que los paquetes pueden ser entregados en orden, causa **retrasos innecesarios** a estas aplicaciones. Para otras aplicaciones, la naturaleza orientada a un flujo de *bytes* de TCP dificulta el envío de **datagramas individuales**, pero si necesitan garantía de entrega, no pueden depender de UDP.

3.4. Referencias bibliográficas

Chao, L. (2016). *Cloud Computing Networking*. CRC Press.

Juniper. (2021, marzo 26). *Show route table*.
<https://www.juniper.net/documentation/us/en/software/junos/bgp/topics/ref/commmand/show-route-table.html>

Tanenbaum, A. y Wetherall, D. (2011). *Computer Networks*. (5ª ed.). Pearson New International.

Computer Networks

Tanenbaum, A. y Wetherall, D. (2011). *Computer Networks*. (5ª ed.) Pearson New International.

La referencia básica de las redes de ordenadores entra en mucho más detalle en la arquitectura de capas y en los modelos de referencia, además de presentar muchos más ejemplos de protocolos reales. Te recomendamos sobre este tema los capítulos uno (apartados 1.3 y 1.4) y los capítulos dos, tres y cuatro.

Cloud Computing Networking

Chao, L. (2016). *Cloud Computing Networking*. CRC Press.

El capítulo dos trata muchos de los protocolos. El apartado 2.6, en particular, pone en contexto estos protocolos con la arquitectura de capas.

1. ¿Ofrece la capa de transporte un servicio orientado a conexión?
 - A. Sí, el protocolo TCP ofrece un servicio orientado a conexión.
 - B. No, en ningún caso.
 - C. Sí, el protocolo UDP ofrece un servicio orientado a conexión.
 - D. Sí, la capa de transporte siempre ofrece un sistema orientado a conexión.

2. ¿Qué protocolo ofrece garantía de entrega ordenada?
 - A. TPC.
 - B. IP.
 - C. UDP.
 - D. Todos los anteriores.

3. ¿Cómo asegura TCP el control de flujo?
 - A. Mediante un tamaño de ventana fijo.
 - B. TCP no ofrece control de flujo.
 - C. Mediante el 3-way handshake.
 - D. Mediante un tamaño de ventana variable.

4. ¿Qué columnas tiene, al menos, una tabla de rutas?
 - A. Red de destino, interfaz de red.
 - B. Red de destino, máscara, número de saltos, interfaz de red.
 - C. Red de destino, máscara, siguiente salto, interfaz de red.
 - D. Ninguna de las anteriores.

5. ¿Qué diferencia hay entre las direcciones 192.168.53.12 y 34.168.53.12?
- A. Ninguna, ambas son direcciones IP normales.
 - B. La primera dirección está reservada para direccionamiento privado, mientras que la segunda es enrutable públicamente.
 - C. El primer *byte*, nada más.
 - D. Pertenecen a subredes diferentes.
6. ¿Cuáles de los siguientes protocolos funcionan sobre TCP? Escoge todas las opciones correctas.
- A. FTP.
 - B. DNS.
 - C. HTTPS.
 - D. NTP.
7. ¿En qué tipo de enrutamiento se optimiza el ancho de banda enviando solo una copia del flujo en cada enlace, incluso si hay múltiples destinatarios?
- A. *Broadcast*.
 - B. *Multicast*.
 - C. *Unicast*.
 - D. *Anycast*.
8. ¿En qué puerto puede escuchar un servidor HTTP?
- A. 80 o 443 solamente.
 - B. En cualquiera entre 0 y 65 535.
 - C. En cualquiera menor de 1024.
 - D. En cualquiera entre 1024 y 49 151.
9. ¿Cuáles de los siguientes son algoritmos de enrutamiento *unicast*?
- A. Inundación.
 - B. Camino más corto.
 - C. Estado de enlace.
 - D. Todos los anteriores.

10. Relaciona cada protocolo con la funcionalidad que ofrece.

| | |
|------|---|
| IP | 1 |
| ICMP | 2 |
| TCP | 3 |
| UDP | 4 |

| | |
|---|-------------------------------------|
| A | Servicio de transporte no confiable |
| B | Direccionamiento de red |
| C | Servicio de transporte confiable |
| D | Diagnósticos |