Resumen Redes

Introducción

Dentro de un computador podemos querer permitir que un proceso se comunique con otro remoto (no en el mismo computador), por lo que es necesario emplear una red de computadores como infraestructura de conexión.

El propósito de las redes es traducir mensajes a señales que puedan ser transmitidas a través de un medio, pero deben encontrar al destinatario para rearmar el mensaje (a partir de distintos segmentos) y entregárselo a un proceso. Un ejemplo típico es el **internet**, éste es un conjunto descentralizado de redes de comunicación interconectadas que utilizan la familia de protocolos **TCP/IP**. Esto garantiza que las redes físicas (heterogéneas) que la componen formen una red lógica única de alcance mundial.

Protocolos

Un protocolo define:

1. Orden de mensajes
2. Formato de mensajes
3. Acciones ante cada mensaje

TCP/IP: es un modelo conceptual y un conjunto de protocolos de comunicación usado en el internet y redes computacionales similares.

Internet Protocol: Es un protocolo de comunicación de datos digitales clasificado funcionalmente en la capa de red, en donde su función principal es el uso bidireccional en origen o destino de comunicación para transmitir datos mediante un protocolo no orientado a conexión (una comunicación entre dos puntos finales de una red en que un mensaje puede ser enviado desde un punto a otro sin previo acuerdo) que transfiere **paquetes** (unidades de información) a través de distintas redes físicas previamente enlazadas.

Hardware de red

El internet posee dispositivos de origen/destino llamados **hosts** (nodos en la red) que pueden actuar como clientes y/o servidores que implementan las capas desde la física hasta la aplicación, por lo que en general son elementos que pueden ejecutar código arbitrario (computador, *Smartphone*, programas, etcétera):

- Clientes: En dispositivos livianos que solicitan servicios y suelen estar en control del usuario. Además, estos pueden conectarse a múltiples servidores ya que puede tener múltiples ***sockets*** (tupla: dirección IP, puerto)
- Servidores: En dispositivos poderosos que proveen servicios, estos suelen vivir en ***data centers***

Los *hosts* pueden enviar paquetes de datos que se transmiten mediante **enlaces (líneas)**.

Existen dispositivos de intercambio de información como los ***routers*** o ***switches*** y puntos de acceso a esta red como los ***modem***, *access point* y ***cell tower***. Dentro de la red, los proveedores de acceso son los ***Internet Service Provider*** **(ISP)**.

Un ***router*** es un *hardware* que permite interconectar paquetes de datos entre redes de computadores, estos paquetes típicamente son recibidos por otro *router* y así hasta llegar al nodo destino. Dado lo anterior un *host* puede actuar como *router*.

Los ***switches*** son los encargados de la interconexión de equipos dentro de una misma red, es decir, son los dispositivos que constituyen las redes de área local o LAN.

Un ***modem*** es un *hardware* que convierte los paquetes de datos en señales para transmitirlos de un computador a otro, y viceversa, por lo que no redirige paquetes.

Medio físico

Es el encargado de permitir que un proceso se comunique con otro para transmitir información codificada como bit. Estos se tienen que transmitir por medio de ondas:

1. Medios guiados: sólidos, par trenzado de cobre, cable coaxial, fibra óptica, entre otros.
2. Medios no guiados: inalámbricos, enlace satelital, entre otros.

Estos *hosts* suelen acceder a la red de distintas maneras, ya sea:

- DSL (Estándar es ADSL): Una misma línea para línea telefónica (distinta frecuencia), *upstream* y *downstream* (ADSL: distintas velocidades para los últimos dos) – Es de *host* a *host*
- Cable Coaxial: usa el cable *modem* con estándares asimétricos con un *broadcast* compartido – todos reciben lo mismo
- FTTH (*Fiber to the home*): Señal óptica a eléctrica por lo que se transmite rápidamente
- *Dial – Up, Satellite*: Enlace a través de línea telefónica y satelital para zonas rurales
- *Ethernet*: Estándar de conexión por pares trenzados de cobre, implementan LAN
- Wi – Fi: Estándar de conexión inalámbrico de corto alcance, implementan WLAN
- 3G/4G/LTE: Estándar de conexión inalámbrico de corto alcance, implementan WLAN.

Las tecnologías inalámbricas se rigen por el protocolo IEE 802.11X, donde X es alguna de sus variantes. A pesar de su ventaja de no presentar cables, se ve dificultada por los siguientes factores:

1. La señal es inherentemente de tipo *broadcast*
2. La señal puede llegar más de una vez debido al efecto de eco
3. La señal puede mezclarse con otras en el camino
4. La señal puede atenuarse al encontrar objetos sólidos en el camino

Funcionamiento de Red

Packet Switching

Es un método de transferencia de datos a una red en forma de paquetes compuestos por un *header* (datos para que el *hardware* de red redirija el paquete a su destino) y un *payload* (los datos per se), en donde un *router* almacena paquetes (*store*), los examina y determina el punto de reenvío (*forward*) gracias al uso de tablas de reenvío (*forwarding tables*).

La tasa de transmisión está determinada por el enlace más lento, por lo que el *router* puede sufrir congestión; si hay N enlaces iguales en la ruta, el tiempo de transmisión de origen a destino (*end – to – end delay*) es

$$d_{end-to-end} = N \times \frac{L}{R}$$

L bits sobre un enlace R bits/sec, por lo que la transmisión demora L/R.

Circuit Switching

Es un tipo de conexión, enfocado a la comunicación telefónica, que realizan los diferentes nodos de una red para lograr un camino apropiado para conectar a dos usuarios en una red, la diferencia con *packet switching* es que se establece un canal de comunicaciones dedicado entre dos estaciones con un ancho de banda garantizado. Es sensible a la saturación ante conexiones simultáneas.

Si hay N enlaces iguales en la ruta, el tiempo de transmisión de origen a destino (*end – to – end delay*) es

$$d_{end-to-end} = N \times \frac{L}{R}$$

L bits sobre un enlace R bits/sec, por lo que la transmisión demora L/R.

Conceptos de Software

El *hardware* solo soluciona la mitad de la tarea:

1. El mensaje debe llegar a un proceso
2. Redes se organizan en base a un modelo de capas
3. Capas se comunican usando protocolos

La separación de responsabilidades permite modelar e implementar sistemas complejos.

Protocolo

Es un contrato de comunicación entre distintas partes acerca de la manera en que se producirá la comunicación, éste puede ser cambiado sin afectar el servicio mediante el proceso de capas con el fin de separar las distintas preocupaciones en la infraestructura de redes.

De acuerdo al protocolo de comunicación confiable:

1. Cuando el *sender* recibe un *timeout*, debe reenviar un mensaje
2. El objetivo de los números de secuencia en el *receiver* es para que éste pueda detectar los mensajes duplicados

Modelo de capas

Una instanciación de este modelo define un *stack* de protocolos, en donde los mensajes pueden ser modificados en cada capa agregando *headers* o modificando todo el mensaje, es decir, las capas proveen servicios a otras capas (tienen que ser capas adyacentes).

No es posible utilizar protocolos distintos en una misma capa para el *sender* y *receiver*.

Un servicio es un conjunto de primitivas que provee una funcionalidad

Modelo OSI

Consiste en un modelo de referencia que define responsabilidades para cada capa, sin definir estándares, protocolos ni implementaciones.

Modelo Internet (stack TCP/IP)

Es una colección ordenada de protocolos organizados en capas que ponen unas encima de otras, y en donde cada protocolo implementa una abstracción encuadrada en la abstracción que proporciona la capa sobre la que está encuadrada. Los protocolos encuadrados en la capa inferior proporcionan sus servicios a los protocolos de la capa superior para que estos puedan realizar su propia funcionalidad.

Capas

Capa de aplicación:

Comunicación entre procesos (aplicaciones) a través del intercambio de mensajes (HTTP, FTP, DNS, SSH, DHCP, etcétera). Las aplicaciones se construyen a través de distintos nodos que se comunican (Web, email, Whatsapp, entre otros).

Arquitectura de software

Arquitectura cliente – servidor: Los clientes se comunican con el servidor, no entre ellos. Éste tiene una dirección fija y conocida donde puede recibir múltiples solicitudes, por lo que usualmente se ubican en *data centers* al necesitar mucha capacidad de atención y ancho de banda con altos requerimientos de infraestructura. Las tareas se reparten entre los proveedores de recursos o servicios, mientras que los demandantes de estos son los clientes.

Arquitectura peer – to – peer (P2P): Las aplicaciones se construyen en base a comunicación directa entre pares, por lo que no hay clientes ni servidores fijos, sino que una serie de nodos que se comportan como iguales entre sí, es decir, funcionan simultáneamente como clientes y servidores respecto a los demás nodos de la red. Es por lo anterior que permite mejor escalabilidad y abandono de miembros. Estas redes permiten el intercambio directo de información en cualquier formato.

Programación de sockets

Los **sockets** son elementos del *software* que interactúan con la red. Es una puerta de entrada/salida al sistema que utiliza servicios de transporte (capa inferior). Solo existe en *host* de origen y destino. También funciona como una interfaz entre el programador y la red, en donde el programador puede controlar los parámetros del *socket*, por ejemplo, el tipo de servicio de transporte (TCP/UDP).

Una vez que ambos extremos, cliente y servidor, poseen un *socket*, es posible mandar mensajes. Éste requiere definición del servicio de transporte y dirección del extremo opuesto (dirección IP remota o puerto remoto). Un ejemplo de aplicación de *sockets*:

1. Cliente lee una línea de entrada estándar (*stdin*)
2. Cliente envía línea al servidor
3. Servidor recibe línea
4. Servidor convierte caracteres a mayúsculas

5. Servidor envía línea modificada al cliente
6. Cliente recibe línea
7. Cliente escribe la línea en la salida estándar (*stdout*)

Tipos de conexiones

TCP *Handshake*: Es un establecimiento de conexión *full duplex* y *end – to – end* que requiere establecerse antes de poder enviar/recibir datos. El cliente inicia la solicitud de conexión en un puerto conocido del servidor, y éste a su vez debe tener un *socket* en ese puerto para aceptar la conexión. Al aceptarla, el servidor asigna un nuevo *socket* para la conexión.

El flag **PUSH** es un booleano que puede ser manipulado para enviar los datos aun cuando el *buffer* no está lleno.

El **TCP *transport entity*** es un módulo que ejecuta en *kernel space* que se encarga de dividir mensajes en paquetes para mandarlos como datagramas IP, además, re ensamblar mensajes fragmentados en el orden correcto, y hacer *timeout* seguido de retransmitir mensajes que no hayan sido recibidos.

UDP: Es un protocolo no orientado a conexión (*best effort*), es decir, cuando una máquina envía a otra, el flujo es unidireccional. La transferencia de datos es realizada sin haber realizado previamente una conexión con la máquina destino, y el destinatario recibirá los datos sin enviar una confirmación al emisor. Lo anterior es debido a que la encapsulación de datos enviada por el protocolo UDP no permite transmitir información relacionada al emisor. Luego, el destinatario no conocerá al emisor de los datos, sólo su dirección IP.

Debido a que es *conectionless*, el servidor para saber quién es el emisor debe implementar "recvfrom".

Protocolos de transferencia de datos

HTTP: Utiliza el modelo cliente – servidor, en donde el cliente HTTP solicita y recibe archivos (y opcionalmente procesarlos e interpretarlos localmente) mientras que el servidor HTTP recibe solicitudes y envía archivos. El formato de los mensajes es elegido por HTTP. Un ejemplo es el siguiente:

Ejemplo de request a http://iic2333.ing.puc.cl/slides/5-disk.html:

1. Cliente HTTP inicia conexion TCP a iic2333.ing.puc.cl/ en puerto 80. Socket en cliente y socket en server
2. Cliente HTTP envía *Request HTTP* incluyendo ruta y archivo: slides/5-disk.html
3. Servidor HTTP recibe *Request HTTP*, busca el archivo slides/5-disk.html, lo encapsula en *Response HTTP*, y envía al cliente a través de su *socket*
4. Servidor HTTP indica a *socket* que cierre la conexión TCP.
5. Cliente HTTP recibe *Response HTTP* y extrae el archivo. Examina el archivo HTML y encuentra referencias a otros archivos (por ejemplo, imágenes JPG, PNG)
6. Cliente puede re-ejecutar los pasos 1 al 4, para cada archivo que necesite.

Los servidores HTTP actuales usan conexiones persistentes y *pipelining*, que consiste en una técnica en la que se envían múltiples solicitudes HTTP en una única conexión TCP sin esperar las respuestas correspondientes. Las conexiones HTTP son *stateless*, sin embargo, puede existir el uso de *cookies*

para mantener un estado. Estas se almacenan en el cliente y son enviadas en las conexiones posteriores, así el servidor puede usar estas *cookies* para **recordar** el estado del cliente.

También, HTTP tiene lo que se denomina como ***web caching*:**

- **Proxy**: Un web proxy es solamente un representante que puede almacenar algunas páginas temporalmente en cache y que puede hacer solicitudes a nombre de otro cliente, pero no almacena contenido de manera estratégica y replicada como una CDN.
- **CDN (Content Delivery Network)**: Es más que solo un proxy, ya que también permite almacenar contenido para que esté replicado y cerca de sus potenciales consumidores, así se acelera el acceso a estos contenidos.

FTP (File Transfer Protocol): Es un protocolo de transferencia de archivos entre el sistema local y remoto entre sistemas conectados a una red TCP basado en la arquitectura cliente – servidor. Desde un equipo cliente se puede conectar a un servidor para descargar archivos desde él o para enviarle archivos, independientemente del SO usado en c/equipo.

Un problema típico de FTP es que está pensado para ofrecer la máxima velocidad en la conexión, pero no la de máxima seguridad ya que todo el intercambio de información, desde el *log – in* y *password* del usuario en el servidor hasta la transferencia de cualquier archivo se realiza en texto plano sin ningún tipo de cifrado, por lo que un atacante puede captar este tráfico y acceder al servidor, por lo que además puede apropiarse de los archivos transferidos.

Correo electrónico: Tiene 3 tipos de protocolos:

1. Simple Mail Transfer Protocol (SMTP): Es un protocolo de transferencia de correos que se da "en línea", de manera que opera en los servicios de correo electrónico por lo que cada vez que se quiera acceder a un correo es necesario que el servidor envíe los datos.
2. Post Office Protocol (POP): Es un protocolo de lectura de correos. No necesita una conexión permanente de internet. También, permite a un usuario descargar el correo electrónico a un cliente en un dispositivo y eliminar el mensaje original del servidor, pero puede causar problemas con la descarga en múltiples dispositivos y la sincronización de correo electrónico.
3. Internet Message Access Protocol (IMAP): Es un protocolo de lectura de correos. A diferencia de POP, necesita de una conexión permanente a internet. IMAP se puede usar para ver los mensajes en línea sin tener que descargarlos primero en un dispositivo.

Domain Name System (DNS): Permite controlar la configuración de correo electrónico y sitio web de tu nombre de dominio, por lo que cuando los visitantes van a tu nombre de dominio, la configuración de DNS controla a cuál servidor de la empresa se dirigen.

Miremos el mismo ejemplo de request a http://iic2333.ing.puc.cl/slides/5-disk.html Antes de iniciar la conexión TCP para el protocolo HTTP a URL:

1. Cliente ejecuta proceso cliente DNS
2. Browser extrae iic2333.ing.puc.cl y lo pasa al cliente DNS
3. Cliente DNS envía *query* por iic2333.ing.puc.cl a servidor DNS
4. Cliente DNS recibe respuesta de servidor DNS, que incluye IP de iic2333.ing.puc.cl
5. Cliente DNS pasa respuesta a browser

6. Cliente HTTP inicia conexión TCP a la IP de iic2333.ing.puc.cl

DNS utiliza una estructura jerárquica de dominios y servidores.

Los servidores almacenan ***mappings***, si no conoce un *mapping* lo consulta a un servidor superior en la jerarquía, por lo que eventualmente podría llegar al servidor raíz (***root server***). Los servidores que administran el *mapping* son **servidores autoritativos**, por lo que queda:

- Servidores autoritativos administran el *mapping* (a nivel de institución)
- Servidores no autoritativos mantienen temporalmente el *mapping*

Existen 13 *root servers* en el mundo con múltiples instancias de c/uno (4 de ellos replicados en Chile).

Por otra parte, existen consultas iterativas y recursivas, en donde las primeras envían una respuesta (completa o parcial) al origen mientras que las consultas recursivas reenvían la consulta. Al resolver un *mapping*, éste es almacenado en los servidores por lo que los *catching* de respuestas permiten reducir el tráfico y poseen un *timeout*. Los servidores TLD suelen estar en caché de servidores locales, por lo que los servidores raíz no suelen ser tan visitados.

Capas de Presentación y Sesión

Capa de Transporte

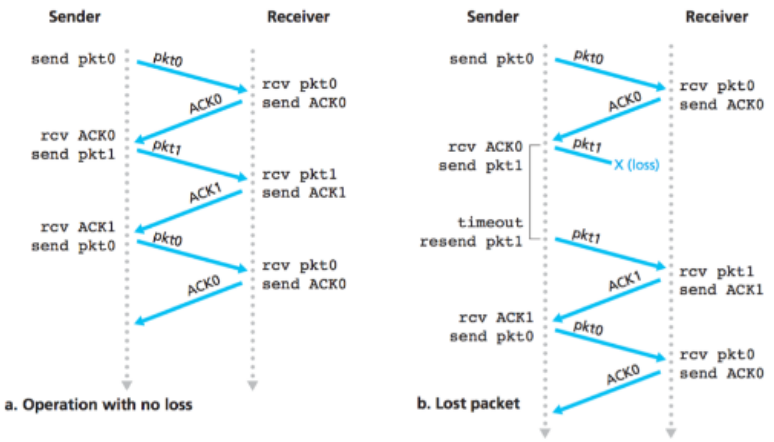
Esta capa tiende a trabajar con programas de usuario y tiene la función de ofrecer la alternativa al usuario de un servicio que puede ser confiable (TCP) o no confiable (UDP) para transferir segmentos hasta un destino final (conceptualmente es simple establecer una conexión entre dos puntos debido a la abstracción de capas). Luego, su objetivo es conseguir que un mensaje llegue desde el emisor (recibe un mensaje de capa de aplicación, lo divide en segmentos, y solicita a la capa de red que envíe los segmentos al receptor. Routers no examinan *headers* de transporte) al receptor (recibe segmentos, los ensambla para formar el mensaje, y lo entrega a la capa de aplicación).

Cada mensaje se divide en segmentos que se transmiten en **paquetes IP**. También, Los **puertos** permiten identificar procesos dentro del nodo emisor o receptor, por lo que pueden multiplexar mensajes y así enviarlos a donde es debido.

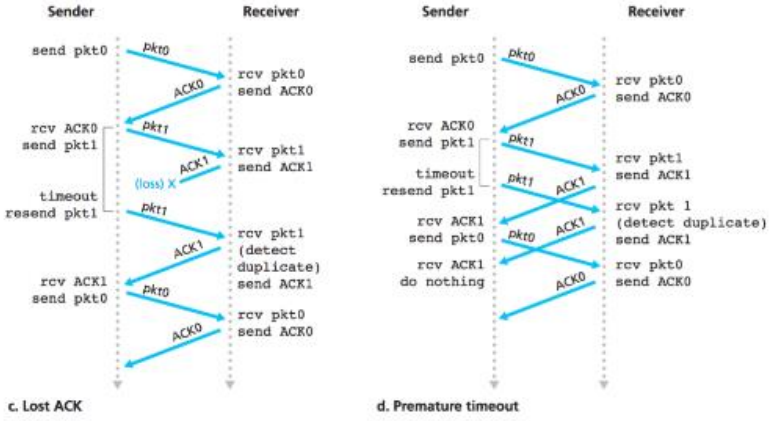
Los procesos envían mensajes a través de *sockets* del SO, en donde c/*socket* se crea con un número de puerto específico.

El desafío es proveer transferencia confiable sobre un medio no confiable ya que los paquetes pueden llegar con errores, por lo que surge el **protocolo ARQ** que detecta errores enviando *checksums*, paquetes de *feedback* ACK que pueden llegar con errores, y como el *sender* no sabe si el *receiver* recibió correctamente el paquete existe un *timeout* para retransmisiones.

Funcionamiento sin pérdida. (b) Con un paquete perdido



Funcionamiento con ACK perdido. (d) Con un *timeout* prematuro



Reliable Data Transfer:

- Se envían varios paquetes simultáneos en modo *pipelined*
- Números de secuencia deben ser incrementales
- Dos enfoques para manejar paquetes: ***Go – Back – N***, ***Selective Repeat***:
  - *Go – Back – N*: Emisor puede mantener hasta N paquetes sin ACK, receptor envía ACK por el último paquete recibido correctamente, y el emisor usa un *timer* para el paquete más antiguo sin ACK.
  - *Selective Repeat*: Emisor puede mantener hasta N paquetes sin ACK, receptor envía ACK para cada paquete individual, y el emisor usa *timer* para el paquete más antiguo sin ACK

Capa de Red

Su objetivo es transmitir paquetes/datagramas desde emisor a receptor mediante la búsqueda de la ruta que permita transferirlos (de forma no confiable). Hay dos problemas principales:

1. **Direccionamiento**: Determinar la ubicación del nodo destino, por lo que cada iembro (*host*) debe poseer una dirección única y la información del paquete debe ser suficiente para encontrar al destino.

2. **Enrutamiento**: Coordinar *routers* para que el paquete llegue al destino, esto se realiza mediante algoritmos para determinar rutas y deben ser eficientes.

Los *routers* son los dispositivos de *store – and – forward*, y deben determinar el próximo camino (salida) de un paquete mirando su dirección de destino. Por lo que el *routing* es una decisión distribuida entre *routers* acorde a un algoritmo que permite llenar las *forwarding tables*.

El ***virtual circuit network*** es un modo de conexión telefónica que garantiza el ancho de banda durante la conexión (desperdiciado cuando no se transmiten datos y poco flexible a congestiones).

La ***Datagram Network*** es un esquema de internet en donde cada paquete lleva la dirección de destino, los *routers* reenvían de acuerdo a su *forwarding table*, los paquetes podrían tener distintos caminos, y son *connectionless*. No garantiza ancho de banda por lo que es flexible ante congestión y ocupar mejor el ancho de banda.

SUBREDES Y DIRECCIONES ESPECIALES

Direcciones reservadas

- `0.0.0.0`. Dirección de host actual (sólo sirve como origen). "Todas las IPs locales".
- `127.0.0.1`. Dirección *loopback* (localhost)

Subredes reservadas

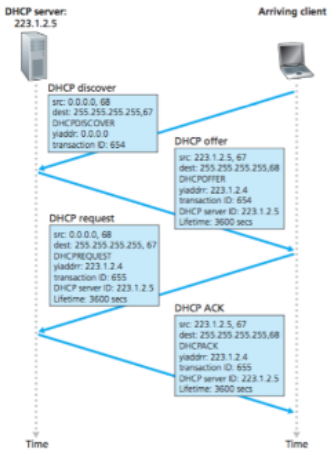
- Red local: `0.0.0.0/8`
- Red loopback: `127.0.0.0/8`
- Redes privadas (RFC1918):
  - `10.0.0.0/8`
  - `172.16.0.0/12`
  - `192.168.0.0/16`

ISPs asignan bloques de direcciones a organizaciones y publican sus bloques, a su vez, estas organizaciones pueden crear nuevas subredes dentro de su bloque.

Dynamic Host Configuration Protocol (DHCP): Se encarga de configurar nuestra IP automáticamente

Protocolo cliente-servidor de 4 etapas

- DHCP Server Discovery. Cliente envía **DHCP discover message**. Destino 255.255.255.255.
- DHCP Server Offer(s). Server envía **DHCP offer message**. Destino 255.255.255.255.
- DHCP Request. Cliente envía **DHCP request message**.
- DHCP ACK. Server envía **DHCP ACK message**.



NAT (Network Address Translation): Permite multiplexar direcciones IP en redes privadas ya que no hay infinitas direcciones IP, por lo que en diversos LANs es posible que dispositivos tengan la misma IP, ya que el *router* se encarga de acceder con su propio IP al internet aislando a los dispositivos LAN.

IPv6: Aumenta el tamaño de las direcciones, mientras que no considera fragmentación intermedia (solo los extremos), no usa *checksum* del *header* (40 byte) y agrega mensajes adicionales. Pueden ser *backwards compatible*, pero IPv4 no.

Algoritmos de *Routing*: El objetivo es conseguir que el paquete llegue desde un *router* de origen a un *router* de destino. El conjunto de *routers* puede verse como un grafo donde el costo entre conexiones es el ancho de banda, congestión, costo, distancia, etcétera.

Hay dos tipos de algoritmos:

1. **Algoritmos Centralizados**: nodos conocen la información completa de la red (algoritmos ***Link – State***)

   Link – state routing: Cada nodo envía su información de conectividad (vecinos y costos) a sus vecinos, la información se comunica a los demás vía *flooding*, cuando todos tienen información de la topología, cada uno calcula las rutas más cortas mediante Dijkstra (no permite costos negativos, pero es bastante rápido al ser un *greedy algorithm*)

2. **Algoritmos Descentralizados**: Cálculo iterativo y distribuido, sin conocer la topología completa (algoritmos de ***Distance – Vector***): Bellman – Ford (permite costos negativos, pero es más costoso)

Internet se compone de **sistemas autónomos (AS)**, los intra – AS resuelven ruteo en un subconjunto de la red, mientras que los inter – AS ayudan a determinar *forwarding tables* entre distintos AS. Los *routers* de salida son ***Gateway routers***.

Capa de Enlace

El objetivo es la transmisión de *frames* a través de un enlace. Lo que hace es:

1. Transformar paquetes en *frames* y transmitirlo a través de un enlace.
2. Determinar quién puede usar el medio compartidor: **Medium Access Control (MAC)**
3. Transferencia confiable en medios con alta tasa de errores.
4. Detección y corrección de errores
5. Qué hacer si un *host* llega a un punto de saturación y no puede recibir/emitir más *frames*: **control de flujo**.

Cada enlace funciona de manera independiente de los demás.

Para detectar errores se agregan bits de redundancia que ayudan al cometido (**paridad errores de un bit**) mediante *checksums* (no protege contra muchos, pero es fácil de implementar).

MAC: Acceso al medio, ocurre un error al acceder a este, generalmente se presenta en medios compartidos (*broadcast*):

Na pico GG

El *switch* es el elemento más representativo de esta capa, ya que permite extender un enlace a través de múltiples hosts y, asimismo, permite armar redes de *switches* para extender estos enlaces. No es el único dispositivo que cumple esta función.

Capa Física

GG

Encapsulamiento

GG xd