# Unidad 1: Introducción

## Requerimientos de las Redes

### Contectividad

- El servicio básico que se espera de una red es que conecte dos o más **nodos** a través de un medio denominado **enlace**.
- Los enlaces pueden ser de punto a punto o compartidos.
- No hace falta que haya un enlace concreto entre cada nodo.
- Puede soportar comunicación unicast (uno a uno), multicast (uno a algunos) o broadcast (uno a todos).
- Si es unicast, la comunicación se establece utilizando una **dirección** que se utiliza en la infraestructura interna de la red para hacer llegar los datos al nodo destino (**routing**)

Podemos definir a las redes recursivamente:

- Uno o más nodos conectados por un enlace físico
- Una o más redes conectadas por un nodo

#### Redes switcheadas

- Son redes en las que hay equipos especiales que se encargan de redirigir datos al nodo de destino.
- Pueden ser **circuit switched** o **packet switched**.
- Las de circuitos primero establecen un canal de comunicación y luego mantienen un flujo de bits entre ambos extremos.
- Las de paquetes transfieren paquetes/mensajes discretos que se recomponen en los extremos. Cada nodo intermedio hace **store and forward**: almacena los datos de un paquete hasta tenerlo entero y luego lo transfiere al siguiente.

### Eficiencia de recursos compartidos

- Para soportar la comunicación concurrente entre cualquier subconjunto de nodos en la red será necesario compartir parde de los recursos, puntualmente enlaces.
- Para eso se multiplexan los datos en el canal: los paquetes se mezclan y hay switches en los extremos de la comunicación que separan los paquetes y los envían al nodo que corresponde.

- A su vez, cada switch pued estar recibiendo datos de varios enlaces e insertándolos en un sólo enlace de salida.

- Para esto hace falta una estrategia otra de multiplexación, que puede o no garantizar tiempo máximo
- El nodo puede tener que almacenar paquetes si los recibe más rápido de lo que los puede enviar

#### Multiplexado estadístico

- Tanto el multiplexado *TDM* como *FDM* desperdician recursos cuando un flujo no hace uso de espacio asignado
- Ambos requieren conocer la cantidad total de flujos.

- El multiplexado estadístico es simlar a *TDM* en tanto asigna el canal de forma exclusiva por cuotas de tiempo acotadas

- A diferencia de *TDM*, lo hace por demanda: si sólo un nodo quiere enviar datos, éste dispondrá del canal de forma exclusiva
- Para evitar inanición de flujos, se define un tamaño máximo en que se puede usar el canal de forma ininterrumpida y así se determinan los **paquetes**
- Si nadie más necesita el enlace, un nodo puede enviar varios paquetes de forma consecutiva
- Puede hacer falta partir un mensaje en varios paquetes y que sean ensamblados en los extremos

### Soporte de servicios comunes

- El objetivo final de la red será proveer comunicación útil entre aplicaciones de los extremos.
- No tiene sentido que cada aplicación implemente completamente el mecanismo de comunicación
- Hay servicios de red comunes sobre los cuáles se construyen los usos específicos.
- Estos servicios se organizan en distintas capas de abtracción para permitir utilizarlos sin limitar la flexibilidad del usuario.
- Cada aplicación puede requerir distintas garantías sobe el canal de comunicación (tiempos, privacidad, orden, etc.)

#### Confiabilidad

- La confiabilidad (garantía de que llegarán los mensajes bajo ciertas condiciones) es uno de los requisitos fundamentales que se suele pedir
- Hay muchos tipos de errores distintos que pueden ocurrir en distintos niveles
- La red se debe encargar de esconderlos

Clases de errores principales:

- Bits corruptos: suelen poder detectarse y no son muy frecuentes
- Paquetes corruptos/desaparecidos: complejo, es difícil determinar cuándo se perdió y cuándo está tardando
- Nodos o enlaces caídos: pueden sobrellevarse routeando alrededor del problema

# Caracterización

Si se quieren caracterizar las redes con respecto a su tamaño, se suele hablar de las siguientes categorías:

- LAN (local área network)
- WAN (wide area network)
- MAN (metropolitan area network)
- SAN (system area network)

Conocer el tamaño que tendrá una red es importante para evaluar las tecnologías sobre las cuales construir el soporte de la misma.

# Arquitectura

## Capas de abstracción y protocolos

- Para atacar la complejidad se suelen encapsular en objetos (**protocolos**) las detalles de implementación
- Estos objetos se ubican en capas de distintos niveles de abstracción
- Se busca que cada protocolo sea fácilmente implementable pero útil en varios escensarios para las capas superiores

- Los protocolos definen una interfaz de servicio a las capas superiores

- También una interfaz de comunicación a su contraparte del mismo nivel de abstracción en el otro nodo
- Excepto en el nivel de hardware, la comunicación en todas las capas es indirecta (pasando por el protocolo de menor nivel
- Decimos que una **arquitectura** es un conjunto de reglas que rigen un grafo de protocolos

### Encapsulamiento

- Los protocolos se abstraen del uso que se les da: no son capaces de interpretar los datos suministrados por las capas superiores.
- Para comunicarse con su par, los protocolos suelen adjuntar al principio del mensaje cierta información de control que llamamos *header* (o *trailer* en caso de que vaya al final).
- En cada nivel, los protocolos encapsulan el mensaje superior adjuntando sus *headers*, que luego son removidos por su par.

### Multiplexación y demultiplexación

- Dado que puede haber muchos usuarios de un protocolo, ¿cómo se sabe (al recibir datos) a qué usuario se debe entregar un paquete?
- En cada paso se introduce en el header una *clave de demultiplexación*, que será utilizada en el otro extremo para ubicar el destinatario del mensaje.
- El formato de la clave de demultiplexación depende de la especificación del protocolo.

## Arquitectura OSI

Es un modelo de referencia para grafos de protocolos propuesto por ISO, que separa en siete capas la funcionalidad que debe ofrecer la red, en donde cada capa puede ser ocupada por distintos protocolos. Las capas son:

1. . Física: transmisión de bits sobre los canales de comunicación
2. . Enlace: manipulación de frames (interfaz de drivers típicamente)
3. . Red: routeo de paquetes en red switcheada
4. . Transporte: comunicación interp-roceso (mensajes), presente en nodos extremos
5. . Sesión (?)
6. . Presentación (?)
7. . Aplicación: donde se da significado a los datos que se envian

## Arquitectura de Internet

Es el grafo de protocolos utilizado para Internet y su predecesor, ARPANET, e influyó fuertemente al modelo OSI. Define los siguientes niveles:

1. . Nivel básico de servicios de red del cuál no se asume nada
2. . IP: soporta la interconección de redes de distintos tipos de forma unificada en una red lógica unificada
3. . TCP/UDP: protocolos de transporte orientados a conexión o datagramas, respectivamente
4. . Aplicaciones (FTP, Telnet, SMTP, etc.)

El foco del diseño está en la capa de *IP*. Este protocolo puede utilizarse sobre distintos soportes y sobre éste se pueden implementar protcolos de transporte y aplicaciones arbitrarias. Esta característica es una de las razones fundamentales de que Internet se haya podido adaptar a los cambios tecnológicos.

# Unidad 2: Teoría de la Información y Codificación

## Información

### Definición

Sea $E$ un suceso que puede presentarse con probabilidad $P(E)$. Cuando $E$ tiene lugar, decimos que hemos recibido

$$I(E) = \log \frac{1}{P(E)}$$

unidades de información.

- La unidad de medición depende de la base del logaritmo. Cuando usamos base 2 la unidad es el **bit**.
- Si $P = 1/2$, tendremos $I(E) = 1\ bit$. Es decir 1 bit es la información obtenida al especificar una de dos alternativas igualmente probables.

### Fuente de memoria nula

- Una fuente de información es cualquier proceso que genera eventos.
- Una fuente de información es de **memoria nula** si la probabilidad de cada evento es independiente de los eventos anteriores.
- Es decir, los eventos son variables aleatorias independientes e identicamente distribuídas.
- Una fuente de memoria nula puede describirse completamente con el alfabeto que representa los eventos ($S$) y la probabilidad de cada evento ($P_i$)).

#### Información en una fuente de memoria nula

La presencia de un símbolo da una información igual a

$$I(s_i) = \log \frac{1}{P(s_i)}\ bits$$

Dado que no son equiprobables, la cantidad media de información por símbolo de la fuente es

$$\sum_S P(s_i) I(s_i)\ bits$$

Esta magnitud recibe el nombre de **entropía de la fuente de información**, o $H(S)$.

$$H(S) = \sum_S P(s_i) I(s_i) = -\sum_S P(s_i) \log_2 P(s_i)$$

Propiedades:

- $H(S)$ es una medida de la incertidumbre media de una fuente de información.
- Se puede interpretar también como una idea de la información media que probee cada símbolo.
- Una fuente que produce siempre el mismo símbolo no provee información, y por ende tendrá $H(S) = 0$

La entropía es máxima cuando todos los valores son equiprobables. En este caso:

- $P(s_i) = 1/n$
- ...
- $H(S) = \log_2 n$

### Extensión de una fuente de memoria nula

Consideramoss $S^n$, la extensión de orden $n$ de la fuente de memoria nula $S$. Tenemos:

$$H(S^n) = n\, H(S)$$

## Fuente de Markov

- Una **fuente de Markov de orden m** es una fuente de información en donde la probabilidad de un símbolo viene determinada por los $m$ símbolos que lo preceden.
- En un determinado momento queda definido el estado de la fuente (la probabilidad asociada a cada símbolo) por los $m$ estados anteriores. Puesto que existen $q$ símbolos distintos, habrá $q^m$ estados posibles, y cada nuevo evento generado modificará el estado.

# Codificación

Codificar es establecer una correspondencia entre los símbolos de una fuente y los símbolos de un alfabeto de representación (*código*).

- Sólo cuando todos los símbolos de la fuente son equiprobables cada símbolo provee $\log_2 n$ bits.
- En otros casos, se puede utilizar el conocimiento de la fuente para lograr representaciones más eficientes eliminando redundancia. (ver *Eficiencia de códigos* más adelante)

*Definición:* Un código se dice **no singular** si todas las palabras del mismo son distintas.

*Definición:* Un código se dice **unívocamente decodificable** si, para cualquier $n$, la extensión de orden $n$ es no singular. Es decir, una sucesión de palabras del código sólo puede representar una única sucesión de símbolos del alfabeto original.

*Definición:* Un código se dice **instantáneo** si es libre de prefijos. Es decir, puedo decodificar cualquier secuencia sin conocer los símbolos que la siguen.

## Inecuación de Kraft

Consideremos

- $S = s_1, s_2, \ldots, s_q$ (alfabeto fuente)
- $X = x_1, x_2, \ldots, x_r$ (alfabeto código)
- $X_1, X_2, \ldots, X_q$ las palabras del código (una por cada símbolo de $S$)

La **inecuación de Kraft** es una condición necesaria y suficiente para que exista un código instantáneo de longitudes $l_1, l_2, \ldots, l_q$:

$$\sum_{i=1}^{q} r^{-l_i} \leq 1$$

En el caso de un alfabeto de código binario, considerando un bloque de $b$ símbolos a codificar:

$$\sum_{i=1}^{b} 2^{-l_i} \leq 1$$

## Eficiencia de códigos

Se buscará asignar palabras más cortas a los símbolos más probables. Es decir, símbolos que brindan poca información deberían tener representaciones que ocupen poco espacio (y viceversa).

Consideremos:

- $l_i$ longitud de la palabra que codifica el mensaje $m_i$
- $r$ número de símbolos del alfabeto de código

Entonces, el número medio de símbolos utilizado para codificar S será:

$$L = \sum_{i}^{q} p_i \, l_i$$

Utilizando la inecuación de Kraft se puede deducir:

$$\frac{H(S)}{\log r} \leq L$$

Si expresamos la entropía en unidades $r$-arias, tenemos:

$$H_r(S) \leq L$$

Esto significa que el tamaño mínimo que podrá utilizar un código para un símbolo $s_i$ será la cantidad de información que provee la aparición de ese símbolo. Un **codificador óptimo** es aquel que para codificar un mensaje utiliza la menor cantidad de bits posible.

### Ejemplo: codificación de Huffman

La codificación de Huffman es una forma de definir códigos óptimos asumiendo que se conoce la probabilidad de ocurrencia de los símbolos, que la codificación es símbolo por símbolo y la probabilidad de ocurrencia de cada símbolo es independiente.

- Se utiliza tomando un texto a partir del cuál se extrae la frecuencia de cada símbolo
- Se organizan los símbolos en un árbol dependiendo de la frecuencia de cada uno.
- Este árbol deja más cerca de la raíz a los símbolos más frecuentes.
- El código de un símbolo será entonces el caminio de la raíz al nodo donde está ubicado (utilizando ceros cuando se toma la rama izquierda y 1 cuando se toma la rama derecha).
- Así, los símbolos más frecuentes tendrán códigos más cortos.

# Medios de transmisión y perturbaciones

Consideramos un modelo donde se envía un mensaje desde una fuente a un destino mediante un transmisor y un receptor. En el canal de transmisión puede haber distorciones que hagan diferir la señal enviada de la recibida.

Posibles causas de las distorciones:

- ruido: señales extrañas a la transmisión
- atenuación: pérdida de potencia de la señal por causa de la distancia
- distorsión de retardo: en medios guiados, los distintos armónicos no viajan a la misma velocidad (necesidad de ecualizar)

## Ancho de banda de Nyquist

Es la máxima teórica velocidad de transmisión $C$ (bits por segundo) de un canal. Depende de la velocidad de modulación $B$ del canal expresada en baudios (símbolos por segundo) y la cantidad de niveles $M$:

$$C = B \log_2 M$$

## Capacidad de Shannon

Es la máxima teórica velocidad de transferencia libre de errores de un canal, en la presencia de ruido.

$$C = B \log_2(1 + \frac{S}{N})$$

donde

- $C$ es la capacidad (bits por segundo)
- $B$ es el ancho de banda del canal (hertz)
- $S$ es la potencia de la señal (watts)
- $N$ es la potencia media del ruido (watts)

Inicialmente se pensaría que aumentando el ancho de banda arbitrariamente aumentará la capacidad de transmisión. Sin embargo, cuando aumenta el ancho de banda también aumenta el ruido, que introduce errores y por lo tanto baja la capacidad de transmisión.

# Unidad 3: Nivel físico

## Conceptos básicos

Clasificación de medios de transmisión:

- Guiados: la señal electromagnética es guiada a través de un canal físico (cable coaxil, fibra óptica, etc.).
- No guiados: son medios inalámbricos (como por ejemplo, el vacío, el aire, agua)

Modos de operación:

- Simplex: siempre en el mismo sentido, hay un extremo emisor y uno receptor
- Half duplex: puede ser en ambos sentidos, pero no al mismo tiempo
- Full duplex: ambos lados pueden transmitir simultaneamente

La información viaja en el medio en forma de onda electromagnética. Estas ondas pueden entenderse como funciones del tiempo, pero también como funciones de la frencuencia. Pensar la frecuencia como dominio será más útil para entender la transmisión de datos.

### Señales como funciones del tiempo

Pensadas como función del tiempo, las señales pueden ser analógicas o digitales.

- Analógicas: la señal tiene variaciones suaves
- Digitales: la señal se mantiene estable durante un tiempo hasta que cambia de valor y vuelve a mantenerse

Las señales más sencillas son las **señales periódicas**. Una señal es periódica si tiene un patrón que se repite a lo largo del tiempo. Es decir, existe un $T$ mínimo que satisface para todo $t$:

$$s(t) = s(t + T)$$

La forma fundamental de señal periódica es la **onda sinusoidal**, que puede representarse por:

- amplitud: valor máximo de la señal
- frecuencia: cantidad de veces por segundo que la señal se repite
- fase: desplazamiento

De la frecuencia se deriva el período (T = 1/f), que es cuánto tarda la onda en llegar al mismo lugar.

También se puede interpretar esta onda en función de la distancia espacial de la fuente, y la relación entre ambas interpretaciones (tiempo y espacio) está dada por la longitud de onda.

### Señales como funciones de la frecuencia

Una onda electromagnética puede estar compuesta de varias frecuencias. Por ejemplo:

$$s_{ej}(t) = (4/\pi) \times [\sin(2\pi ft) + (1/3)\sin(2\pi 3ft)]$$

puede ser descompuesta en dos señales más simples, una con frecuencia $f$ y otra con frecuencia $3f$.

Utilizando análisis de Fourier, puede mostrarse que ***cualquier onda*** puede expresarse como una suma de sinusoides de distintas frecuencias (cada una con su amplitud, frecuencia y fase).

Así como definimos la señal como una función del tiempo, se puede derivar una función $S(f)$, que indica para cada frecuencia que compone la señal cuál es el pico de amplitud.

- El **espectro** de una señal es el rango de frecuencias que contiene. Por ejemplo, el espectro de $s_{ej}$ se extiende de $f$ a $3f$.
- El **ancho de banda absoluto** es el ancho del espectro. En el ejemplo, $2f$.
- Algunas señales tienen ancho de banda absoluto infinito pero la mayoría de la energía de la señal está contenida en un conjunto acotado de señales. Esta banda es el **ancho de banda efectivo** (o simplemente **ancho de banda**).

### Frecuencia fundamental

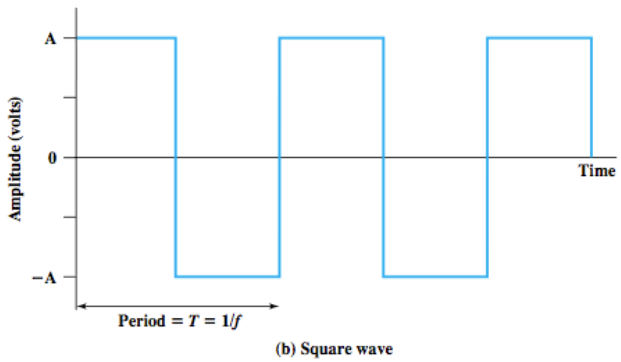
- Cuando en una señal existe una frecuencia $f$ tal que todas las otras señales son múltiplos enteros de la misma, llamamos a $f$ la **frecuencia fundamental**.
- El período de la onda será igual al período de la componente de frecuencia $f$: $T = 1/f$
- Esto se explica viendo que dentro de cada repetición de frecuencia $f$ habrá una cantidad entera de repeticiones de las mayores frecuencias. La onda total, entonces, se repetirá con cada período de la frecuencia menor.

## Relación entre capacidad y ancho de banda

Hay tres factores que entran en juego para entender la velocida de transmisión de un medio:

- Bitrate (cuántos bits por segundo se desean enviar)
- Ancho de banda utilizado
- Capacidad del receptor de distinguir impulsos positivos y negativos

Para ver cómo juegan entre sí, supongamos que queremos utilizar la onda cuadrada para codificar un flujo de unos y ceros alternados:

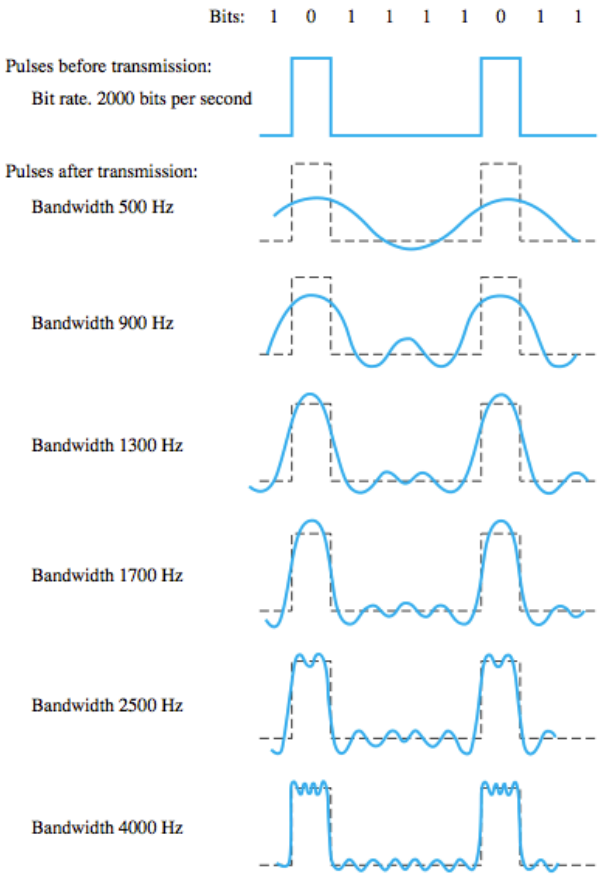


(b) Square wave

Un análisis arrojará que esta señal consiste de infinitas frecuencias:

$$s(t) = A \times \frac{4}{\pi} \times \sum_{k\ impar,\ k=1}^{\inf} \frac{\sin(2\pi kft)}{k}$$

Sin embargo, si incorporamos uno a uno los componentes (es decir, aumentamos el $k$) vemos que tras agregar algunas pocas frecuencias la señal se comienza a parecer mucho a la que queremos obtener. A medida que se aumenta el ancho de banda disminuye la distorción, y será más fácil reconocer pulsos positivos de los negativos. Depende del receptor de la señal qué grado de distorción es tolerable para distinguir los impulsos.




Supongamos, en el caso de la imagen anterior, que al receptor le alcanza un ancho de banda de 900 Hz para distinguir los impulsos. ¿Qué sucede si se quiere duplicar el bitrate? Suponiendo fija la capacidad del receptor, hará falta aumentar también el ancho de banda poder transmitir mayor cantidad de bits por segundo.

Conclusiones:

- Por motivos prácticos se limita el ancho de banda
- Limitar el ancho de banda crea distorciones, que dificultan interpretar los datos
- A mayor bitrate, hará falta mayor ancho de banda efectivo para poder transmitir

# Medios de transmisión

## Medios guiados

### Par trenzado:

- Es el medio más barato y muy utilizado.
- Consiste de dos alambres de cobre aislados trenzados que funcionan como un único canal.
- A veces se juntan varios pares en un cable.
- Sirve para transmisión analógica y digital.
- Limitado en distancia, ancho de banda y capacidad.

### Cable coaxil

- Consiste de dos conductores, un cilindro exterior que contiene a otro.
- Soporta un mayor rango de frecuencias (y por ende multiplexación FDM).
- Menos suceptible a distorciones.
- Sirve para transmisión analógica y digital.

### Fibra óptica

- Consiste de un cable muy fino compuesto de fibras por el que se guia un rayo óptico
- La luz se propaga por el núcleo y se va reflejando en el revestimiento
- Mucha mayor capacidad que los otros medios
- Más resistente a distorciones

## Medios no guiados

Los tipos de comunicación inalámbrica se dividen según el rango de frecuencia que utilicen las ondas

### Radio

- Entre 30 MHz y 1 GHz
- Transmisión omnidireccional
- Se transmite y recibe con antenas

### Microondas

- Entre 1 GHz y 40GHz.
- Útil para transmisiones punto a punto direccionales.
- Utilizado en transmisiones de satélite

### Infrarojo

- Entre $3 \times 10^{11}$ y $2 \times 10^{14}$ Hz.
- Transmisión local punto a punto y multipunto en espacios reducidos.

# Formatos de transmisión

Tanto información digital como analógica puede ser codificada por medios digitales o analógicos:

| Información | Señal | Características / Usos |
|---|---|---|
| Digital | Digital | Equipamiento sencillos |
| Digital | Analógico | Internet sobre medios que sólo soportan señales analógicas (teléfono) |
| Analógico | Digital | Enviar audio o video que se samplean para enviarse por un medio digital |
| Analógico | Analógico | Modular una señal para enviarla en una banda de frecuencia distinta. |

- Un **modem** se utiliza para convertir datos digitales y enviarlos en una señal analógica, y hacer la conversión inversa en el lado del receptor.
- Un **codec** se utiliza para convertir datos analógicos y enviarlos en una señal digital, y hacer la conversión inversa en el lado del receptor.
- Si bien ambos dispositivos saben revertir la conversión, no se puede reemplazar un modem con un codec (o viceversa). El codec reconstruya una señal analógica a partir de las muestras que otro codec generó, pero no puede convertir a analógico cualquier dato digital.

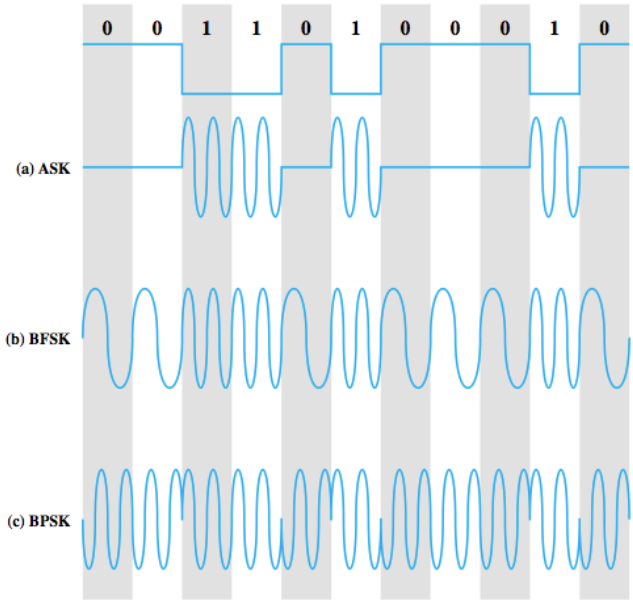
## Datos analógico en señales digitales

Proceso llevado a cabo por el codec:

- Se "separa" la señal por frecuencias (series de Fourier)
- Se muestrea al doble de frecuencia para no perder datos (Teorema de Muestreo, Nyquist)
- Se cuantifica cada muestra con un número entero de $n$ bits

## Datos digitales en señales analógicas

Para este tipo de conversión se modula alguno de los tres parámetros de la señal (amplitud, frecuencia, fase) para poder encodear los valores de la señal digital.




### ASK (amplitud)

Se codifica usando dos valores de amplitud (típicamente uno es cero).

### FSK (frecuencia)

Se codifica usando distintos valores de frecuencia cercanos a la frecuencia del carrier. El caso más sencillo es BFSK (*binary frecuency shift keying*), donde un valor de frecuencia representa un 1 y el otro un 0. Hay codificaciones que utilizan más valores.

### PSK (fase)

Se codifica usando distintos valores de fase. El caso más sencillo es BPSK (*binary phase shift keying*), donde un valor de fase representa un 1 y el otro un 0. Por ejemplo, usando un desplazamiento igual a $\pi$ resultará en que para un 1 se tendrá una sinusoide espejada 180º con respecto a la del 0. Hay codificaciones más complejas.

### Datos digitales en señales digitales

Se utilizan distintos tipos de encodings para representar los bits de información en la señal digital.

### NRZ

- *Non Return to Zero*.
- 0 se codifica con alta tensión.
- 1 se codifica con baja tensión.
- No provee sincronización.

### NRZI

- *Non Return to Zero, Inverted*.

- 0 se codifica con ausencia de transiciones al inicio del intervalo de cada bit.
- 1 se codifica con transición al inicio del intervalo de cada bit.
- Puede ser más fácil detectar cambios que el valor.
- Evita que una inversión de los valores afecte el valor decodificado.
- No provee sincronización.

### Códigos de alta densidad

- Atacan los problemas de sincronización reemplazando cadenas de tensión constante por otra que introduzca transiciones
- Ejemplos: B8ZS y HDB3

### Manchester

- Transiciones a mitad del intervalo de cada bit.
- 0 se codifica con transición de alto a bajo.
- 1 se codifica con transición de bajo a alto.

### Manchester diferencial

- Siempre hay transición a mitad del intervalo
- 0 se codifica si con transición al inicio del intervalo
- 1 se codifica con ausencia de transición al inicio del intervalo

# Unidad 4: Nivel de enlace (punto a punto)

## Medidas de eficiencia

### Ancho de banda y latencia

- El **ancho de banda** (o **throughput**) es la cantidad de bits que se pueden transmitir en una red en determinado período de tiempo. Ejemplo: 10Mbps
- La **latencia** es cuánto demora un mensaje en llegar de un extremo al otro. Se mide en unidades de tiempo.
- El **roundtrip time** (*RTT*) es cuánto tarda un mensaje en ir y volver.

- Latency = Propagation + Transmit + Queue

- Propagation = Distance / SpeedOfLight
- Transmit = Size / Bandwith

### Producto delay $\times$ ancho de banda

Se puede pensar el canal de comunicación como un tubo, donde la longitud es el delay y el diámetro el ancho de banda. En este caso el producto de ambas métricas nos da el volumen, que se corresponde con la cantidad de bits presentes en el canal en un determinado momento.

Esta métrica es útil porque da la idea de cuántos bits se deben transmitir hasta que el receptor comienza a recibir información. A su vez, el doble de esta magnitud es lo que se puede llegar a enviar hasta que se tiene una primera respuesta del receptor.

## Framing

En la capa física se resuelve el problema de transmitir secuencias de bits a través de un medio. Si queremos implementar una red de switcheo de paquetes, el siguiente problema que nos encontramos es cómo darnos cuenta cuándo empiezan y terminan los frames.

### Protocolos orientados a byte

Se entiende el flujo como una secuencia de bytes, y se interpretan utilizando algún charset predefinido (ASCII, por ejemplo).

#### Sentinel

- utilizar marcas de inicio y fin de la sección de datos del frame
- apariciones del marcador de fin de datos se escapa cuando aparece dentro del contenido
- se incluye un CRC (ver métodos de detección de errores) para el frame

- se incluyen campos extra para lograr confiabilidad

#### Byte count

- en vez de utilizar marcas de incicio y fin, se puede declarar el tamaño del paquete al inicio del mismo
- se introduce *framing error* si se corrompe el campo de tamaño.

### Protocolos orientados a bit

- Utilizan una secuencia especial que indica el inicio de un frame.
- A su vez, esta secuencia se envía constantemente cuando no se usa el canal para mantener emisor y receptor sincronizados.
- Se utilizan técnicas de *bit sttuffing* para escapar apariciones dentro de los datos de la secuencia especial.

## Detección de errores

El mecanismo fundamental para detectar errores es agregar un pequeño número de bits redundantes a cada paquete, que se calculan a través de un algoritmo definido y permiten que el receptor de un paquete pueda verificar con un alto nivel de certeza la ausencia de errores de bits (tras recalcular utilizando el mismo algoritmo).

### Bit de paridad bidimensional

- Agregar un bit al final indicando si la cantidad de 1s en el resto del mensaje es par o impar.
- Agregar un byte extra de paridad, donde cada posición indica paridad entre todos los bits de esa posición.
- Protege contra todos los errores de 1, 2 y 3 bits.
- Protege contra la mayoria de los errores de 4 bits.

### Internet checksum

- Sumar el contenido de todas las words de un paquete y usar la suma como código de verificación
- No se usa directamente en nivel de enlace
- Se suele usar por encima de otros mecanismos más efectivos

### CRC (cyclic redundancy check)

- Interpreta frames como polinomios con coeficientes 0 y 1
- Utiliza propiedades aritméticas de divisibilidad de polinomios
- Se pueden elegir distintos polinomios divisores para detectar distintos tipos de errores
- Fácil de implementar

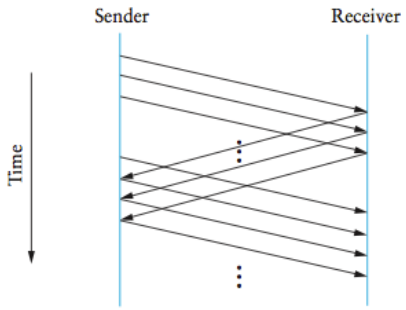
## Transmisión confiable

Al detectar errores no corregibles en frames se querrá forzar el reenvío de dichos frames. A su vez, se debería contar con un mecanismo para reenviar mensajes que nunca llegaron.

## Stop and wait

- Emisor espera el ACK de cada frame antes de enviar el siguiente
- Si el ACK no llega (o se demora) retransmite
- En caso de que un ACK se pierda o llegue tarde, se retransmite. Para evitar que el receptor confunda el frame retransmitido con el siguiente, se utiliza un número de secuencia de 1 bit.
- Problema: no se utiliza al máximo el enlace.

### Sliding window




Emisor:

- Genera números de secuencia consecutivos para los mensajes
- *SWS* (send window size): cantidad máxima de mensajes sin ACK
- *LAR* (last acknowledgement received)
- *LFS* (last frame sent)
- Invariante: $LFS - LAR \leq SWS$

Receptor:

- *RWS* (receiver window size): cantidad máxima de mensajes fuera de orden que se aceptan
- *LAF* (largest acceptabe frame)
- *LFR* (largest frame received)
- Invariante: $LAF - LFR \leq RWS$

Propiedades:

- En ausencia de errores aprovecha al máximo la capacidad.
- El emisor tiene un mecanismo de timeout para retransmitir.
- *SWS* se ajusta dependiendo de la capacidad del enlace.
- *RWS* indica cuánto está dispuesto a guardar en buffer el receptor.

Utilidad:

- Problema de pérdida de frames
- Problema de orden incorrecto de frames
- Problema de control de flujo: el receptor puede limitar el ritmo del emisor

#### Extensión: selective ACKs

- En caso de que el primer frame de un bloque grande no llegue, el receptor no puede enviar ACK del resto

- Esto fuerza a retransmitir innecesariamente muchos frames que el receptor ya tiene
- Si se utilizan **ACK selectivos**, el receptor puede enviar confirmación de bloques de frames ya recibidos para evitar este problema.

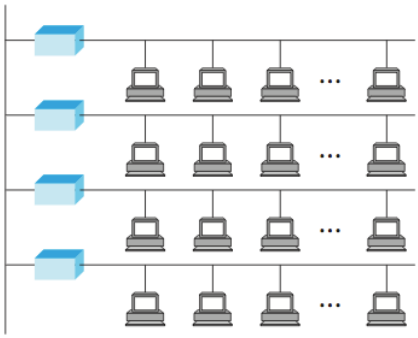
# Unidad 5: Nivel de enlace (acceso compartido)

## Ethernet (802.3)

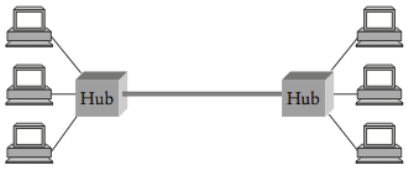
Se trata de una tecnología para crear redes de múltiple acceso sobre un canal compartido. Todos los nodos pueden sensar el estado del medio (*idle*/*busy*) y detectar colisones. El acceso compartido introduce dos nuevos desafíos: control de acceso y esquema de direccionamiento.

### Configuración física

- **Las señales se distribuyen a lo largo de todo el cableado**
- Segmento de cable de hasta 500m
- Hosts conectados al segmento
- Posibilidad de juntar segmentos con repetidores:



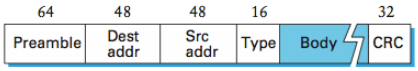
- Repetidores multidireccionales (hubs):




### Protocolo de acceso

El protocolo de acceso suele implementarse en hardware en el adaptador de red.

#### Formato de frame:




- Preámbulo (sincronización)
- Destination address
- Source address
- Type (demultiplexación)
- Body, con longitud mínima (para poder detectar colisiones)
- CRC

### Direccionamiento

- aka. MAC address
- Cada adaptador tiene una dirección única asignada en ROM
- Formato legible: seis números de 1 byte (ej: d9:b2:5a:8c:21:c7)
- Cada adaptador recibe todos los paquetes y acepta:
  - los dirigidos a ese adaptador
  - los dirigidos a la dirección de broadcast
  - los dirigidos a una dirección de multicast a la que está atento
  - todos, si está en modo promiscuo

### Algoritmo de transmisión

- Si el canal está *idle* se envía (máximo 1500 bytes)
- Si no, espera y transmite inmediatamente cuando se libera
- Cuando dos emiten al mismo tiempo se produce una **colisión**
- Todos los adaptadores pueden detectar la colisión
- Mientras más lejos están los hosts, más se tarde en detectar la colisión
- El mínimo de longitud de frame está para garantizar que siempre se detecten (con longitud máxima del cableado acotada)
- Cuando un emisor detecta colisión, realiza **exponential backoff** con un componente random antes de reintentar
- Tras cierta cantidad de reintentos, el adaptador reporta error al host

# Token Ring (802.5, FDDI)

- Otra forma de compartir el medio
- Topología de anillo
- Token (secuencia de bits específica) circulando todo el tiempo
- Cuando un host quiere enviar, espera a recibir el token.
- En vez de reenviarlo, envía los datos.
- Todos los hosts reenvían los datos alrededor del anillo
- El destinatario se guarda una copia
- Cuando el dato llega de vuelva al emisor, deja de retransmitir y pone en circulación el token nuevamente

# Wireless (802.11)

## Propiedades físicas

La transmisión puede realizarse a través de señales infrarojas o de radio. Las señales infrarojas ya no se utilizan por su poco alcance. En las de radio, la interferencia con otras señales es un problema. Para evitar limitarse a un rango de frecuencias limitado (y posiblemente saturado) se utilizan técnicas de **espectro disperso**.

### Espectro disperso

Consiste en expandir el espectro de la señal en un ancho de banda mayor evitando concentrar la potencia sobre una única y estrecha banda de frecuencias. Hay tres técnicas principales:

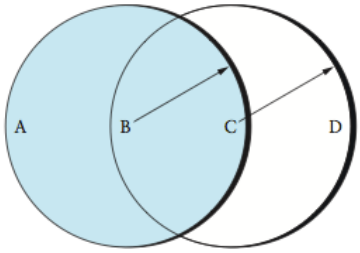
- Frecuency hopping: el emisor va cambiando la frecuencia y el receptor lo sigue, utilizando una secuencia pseudoaliatoria con semilla compartida
- Direct sequence: se agrega ruido pseudo-aleatorio a la señal para que ocupe un ancho de banda mayor
- Multiplexación por división de frecuencia ortogonal

## Manejo de colisiones (CSMA-CA)

El método de Ethernet (*CSMA-CD*, *Carrier sense/multiple access with collission detection*) busca detectar las colisiones cuando se sensa el medio. Esto no se puede implementar para el medio inalámbrico:

- No todos los hosts están al alcance de otros, lo cuál dificulta el sensado del medio y la detección de colisiones
- Se necesitaría un medio de radio full duplex que sería muy costoso

Los problemas principales que pueden ocurrir son:




- **Hidden nodes:** dos hosts que no se ven (*A* y *C*) intentan enviar datos a uno que está al alcance de ambos (*B*). Al no verse entre sí, no pueden detectar la colisión.
- **Exposed node:** ocurre cuando un nodo interpreta que no está en condiciones de transmitir por detectar una transmisión que en realidad no lo perjudicará. Ejemplo: *B* transmite a *A*, y *C* quiere transmitir a *D* pero cree que no puede por la transmisión de *B*. En realidad, la transmisión de *B* sólo introducirá interferencia en el segmento entre *B* y *C*.

Lo importante para mediar acceso será detectar actividad **en las cercanías del receptor**, y no alrededor del transmisor. Notar que en el caso de Ethernet no hacía falta diferenciar estos casos.

El método **CSMA-CA**, **Carrier sense/multiple access with collision avoidance** soluciona estos problemas evitando la generación de colisiones (sabiendo que no se pueden detectar tan fácilmente como en *CSMA-CD*).

**MACA (Multiple Access with Collision Avoidance)**

La idea fundamental es que emisor y receptor intercambian frames de control que son vistos por todos los hosts en el alcance de cada uno y les permiten decidir cuándo se puede transmitir.

- Cuando se quiere transmitir, se envía un *RTS* (request to send) indicando el receptor y por cuánto tiempo se quiere disponer del canal.
- El receptor responde con un *CTS* (clear to send), que también contiene el campo *length*.
- Si un nodo ve el *CTS*, sabe que está cerca del receptor y se abstiene de transmitir por el tiempo que indique *length*.
- Si un nodo ve *RTS* pero no *CTS*, sabe que está lejos del receptor y entonces puede transmitir.
- Si dos nodos alejados envian *RTS* al mismo host, sólo uno recibirá el *CTS*. El otro esperará un tiempo random y reintentará más tarde.

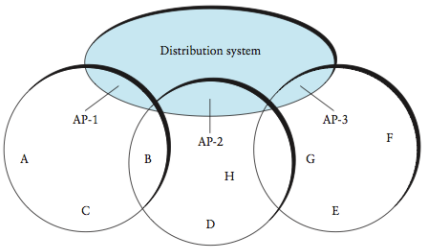
**MACAW (MACA for Wireless LANs)**

Agregados sobre **MACA**:

- ACK tras cada frame
- Intercambio de información sobre congestionamiento entre hosts
- Cambios en algoritmos de backoff en caso de no poder enviar

## Distribución

El estándar define una estructura para las redes, de modo de permitir que nodos que no sean visibles entre sí se puedan comunicar. Además, ataca el problema de la movilidad de los nodos.




- Algunos nodos distinguidos serán **access points** y están conectados a la infraestructura cableada
- Cada access point hace de estación a los hosts que están dentro de su cobertura
- La distribución se realiza a nivel 2 de la arquitectura ISO (no requiere protocolos superiores)
- Cuando un hosts quiere transmitir a otro fuera de su alcance, envía los datos a su access point, que forwardea los mismos al acces point del destinatario a través del sistema de distribución

Para asociarse o cambiar de access point, un host envía un request al cuál responden todos los access points dentro del alcance. Luego el host elige el que prefiere y le confirma la asociación. En caso de que ya haya estado asociado a otro, el nuevo acess point lo da de baja del anterior a través del sistema de distribución.

# Packet switching

Las redes punto a punto, ethernet o wireless tienen limitaciones en cuanto a cantidad de hosts y área de cobertura. Para lograr redes con mayor cobertura, se utilizan *switches* que permiten comunicarse a hosts que

no están directamente conectados. Cuando llega un paquete a un puerto de input, el trabajo del switch es decidir por qué puerto de output forwardearlo. Esto permite una nueva topología de estrella:

- Los switches se conectan entre sí y con hosts mediante enlaces punto a punto, permitiendo armar redes más grandes.
- Escalabilidad: con un bus compartido (Ethernet) la capacidad del enlace limita cuánto pueden transmitir los hosts. En esta nueva topología, cada host puede tener su propio enlace al switch y aprovecharlo al máximo.

A grandes rasgos, hay dos grandes tipos de estrategias de forwarding: las de datagramas y las orientadas a conexión.

## Datagramas

- Cada paquete tiene la información suficiente para ser dirigido correctamente.
- El switch tiene una tabla que indica por qué puerto enviar los paquetes para cada dirección.
- Es *stateless*, en cualquier momento se puede enviar paquetes.
- Al momento de enviar un paquete, el host no tiene garantías de que la red lo puede manejar.
- Una falla en un enlace puede no traer grandes problemas si la red es capaz de encontrar una via alternativa.

## Circuitos virtuales

Es un modelo distinto, donde primero se establece una conexión virtual y luego se transfieren los datos.

En la fase de establecimiento de conexión, los switches tienen estado: una entrada en una tabla de circuitos virtuales con:

- identificador del circuito interno del switch
- interfaz de entrada
- interfaz de salida
- identificador del circuito externo

Cuando entra un paquete por una interfaz con determinado identificador, se selecciona la interfaz de salida y se setea el identificador externo. Estas tablas de circuitos virtuales se pueden configurar estáticamente por el administrador (creando conexiones "permanentes"), pero también se puede utilizar *signalling* para que los hosts los creen y borren dinámicamente.

Una vez establecida la conexión, el origen sabe que el destino es alcanzable y está dispuesto a recibir datos. Además, se pueden reservar recursos para esta conexión (ejemplo: buffers en switches intermedios).

### Signalling

- Para iniciar una conexión el host origien envía un mensaje de *setup*
- El mensaje recorre el camino desde el origen al destino (utilizando algoritmo de routing, ver más adelante).
- Cuando llega a destino, los switches desde el final envían a su predecesor el identificador interno que eligieron para esa conexión.
- Cada switch recibe este número de su sucesor y lo setea como el identificador externo.
- Llegado este punto, se tiene la conexión establecida y se pueden enviar datos.
- Cuando se quiere terminar, de envía un mensaje de *teardown*.

## LAN Switching

- Llamamos **LAN switching** a la conmutación de paquetes entre LANs de acceso compartido (como Ethernets).
- A los switches en estos casos de los suele llamar LAN switch o **bridges**.
- Decimos que hay una *extended LAN* cuando tenemos una colección de LANs unidas por bridges.

### Learning bridges

En el caso más simple, el switch puede forwardear los paquetes por todos los puertos de salida. Una forma de optimizar esto sería forwardear sólo al puerto que corresponda. Esto se puede hacer del siguiente modo:

- En principio se puede enviar todos los paquetes por todos los puertos de salida
- Se inspecciona la dirección de destino de cada paquete, de modo de conocer en qué puerto está el host origgen.
- Se asocia un timeout a cada entrada.
- No hace falta que la tabla esté completa para el funcionamiento de la red, es sólo una optimización.

### Spanning tree algorithm

La topología de la LAN extendida puede tener ciclos, ya sea por error o porque se desea proveer caminos alternativos en caso de falla de bridges. Por esto, hace falta un mecanimos para evitar que los paquetes circulen constantemente en la presencia de ciclos.

Esto se logra implementando un algoritmo distribuído de *spanning tree*, que limita la topología a un árbol que cubra todos los puntos pero evite ciclos. Esto es, para cada bridge decide por qué puertos está dispuesto o no a retransmitir paquetes.

Descripción de alto nivel:

- Se asigna un id numérico a cada bridge (prioridad + MAC address)
- El bridge con menor id es elegido root, y siempre forwardea por todos los puertos
- Cada bridge computa el camino mínimo hacia el root, y marca el puerto que pertenece a este camino (*preferred port*).
- Por cada LAN, se elige entre los bridge el más cercano al root para que sea el encargado de forwardear paquetes a esa LAN (*designated bridge*). Se marca el puerto que conecta el bridge con esa LAN (*designated port*).
- Cada bridge desactiva todos sus puertos que no sean *preferred* o *designated*.

Este algoritmo se implementa con un protocolo de intercambio de mensajes de configuración. Los mensajes

contienen:

- Id del bridge
- Id del bridge que se cree que es root
- Hops hasta el supuesto root

Al inicio, todos los bridges se creen el root y envían un mensaje indicando eso. Cada bridge también recuerda cuál es el mejor mensaje que llegó. Un mensaje es mejor que otro si:

- identifica un root con menor id
- identifica un root con igual id pero menor distancia
- identifica un root con igual id a igual distancia, pero el id del emisor es menor

El sistema se estabiliza de la siguiente forma:

- Al momento que un bridge detecta que no es root, deja de enviar sus mensajes y sólo forwardea los que llegan (incrementando en uno la distancia).
- Al momento que un bridge detecta que no es el bridge designado para un puerto, deja de enviar mensajes a ese puerto. Esto ocurre cuando llega por ese puerto un mensaje de un bridge mejor ubicado o con menor id.
- Al final, sólo el root estará generando mensajes, y el resto de los bridges sólo forwardearán por los puertos en donde son bridge designado.

A su vez, el bridge continua enviando mensajes de configuración periódicamente. En caso de que falle un bridge, los que queden desconectados volverán a considerarse root, y el algoritmo vuelve a generar otro árbol. Este mecanismo soluciona fallas de bridges pero no provee routeo alternativo por congestión.

# Unidad 6: Nivel de Red

## Internetworking (IP)

Al querer conectar redes de distinto tipo nos encontramos con dos problemas fundamentales:

- *heterogeneidad:* ambas redes pueden ser de distinto tipo, así como también las redes que se tiene que atravesar para llegar de una a otra.
- *escala:* cómo encontrar eficientemente un camino desde un punto a otro, y qué convención de direcciones usar

Al hacer esto se construye una red "lógica" entre redes. A diferencia de las redes switcheadas, que tenían limitaciones en cuando a cobertura y cantidad de hosts, se querrá que estas *internets* escalen arbitrariamente. Los nodos que conectan cada una de las subredes se llaman **routers**.

De acá en adelante se describe el funcionamiento de IP, la tecnología de internetworking detrás de Internet.

### Modelo de servicio

El conjunto de servicios ofrecidos por la internetwork debe ser lo suficientemente laxo para que pueda implementarse sobre todos los tipos de redes que la componen y para un amplio abanico de casos de uso. Los protocolos montados sobre IP deben ser concientes de las garantías faltantes y compensarlas cuando lo requieran.

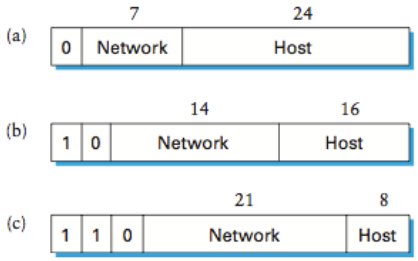
IP define tanto el **esquema de direccionamiento** como la forma de **enviar mensajes**, con las siguientes características:

- **datagramas**: no es orientado a conexión
- **best effort**: los paquetes pueden perderse, reordenarse, repetirse o llegar tarde

### Direccionamiento global

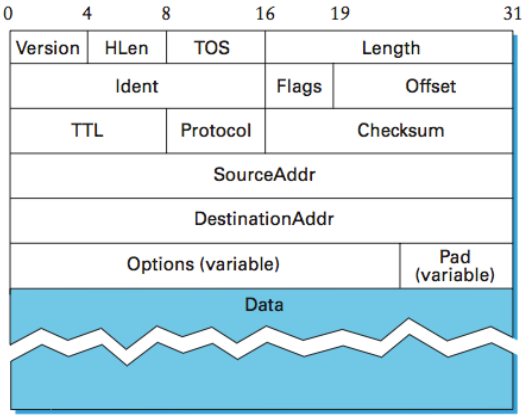
Dado que se quieren conectar redes que no tienen por qué conocerse inicialmente, es necesario asignar direcciones únicas a cada host. A su vez, es deseable que las direcciones tengan una estructura jerárquica que facilite el routeo (cosa que no tienen las direcciones Ethernet).

- Las direcciones IP tienen dos partes: una indica la red y otra identifica un host dentro de esa red.
- El espacio asignado a cada parte depende de la clase de dirección (ver figura).




## Envío de datagramas

### Formato de paquete




- Version: versión del protocolo utilizado
- HLen: longitud del header, variable cuando se incluyen opciones
- TOS: flags para modificar el envío
- Length: cantidad de bytes del datagrama completo
- Ident, Flags, Offset: datos sobre fragmentación (ver más adelante)
- TTL: cantidad de hops antes de ser descartado
- Protocol: demultiplexación de protocolos superiores (ejemplo: TCP y UDP)
- Checksum: control de integridad
- SourceAddr: utilizado para que el destinatario pueda o no aceptar el datagrama
- DestinationAddr: para permitir routeo por cada datagrama

### Fragmentación y rearmado de paquetes

IP puede funcionar sobre redes con distintas limitaciones en tanto al tamaño máximo de paquetes. Definir un tamaño tan pequeño como para viajar por cualquier soporte es poco conveniente, sin contar que es imposible conocer ese tamaño a priori. Por esto, IP tiene un mecanismo para desarmar paquetes cuando no pueden viajar por una red, y rearmarlos cuando es necesario.

- Se llama **MTU** al máximo tamaño de datagrama IP que puede transmitirse en una red
- Cuando un router debe enviar un datagrama por una red que tiene un MTU menor al del datagrama, lo fragmenta
- El campo *ident* del header indica que dos fragmentos forman parte del mismo datagrama original
- El campo *offset* indica el número de fragmento

- El flag *M* indica que hay al menos un fragmento con mayor offset
- El rearmado se realiza en el destinatario del mensaje, no en cada router
- Si en el rearmado falta un fragmento, se descartan todos los de ese ident

### Forwarding

- Los routers tienen varias interfaces, conectadas a distintas redes.
- Cada interfaz tiene una dirección distinta.
- Cuando un host o router debe enviar un datagrama se fija si pertenece a la misma red física (o una de ellas, en caso de routers).
- Si pertenece a la misma red, lo envía según corresponda para esa red.
- Si no, el datagrama debe reenviarse a un (otro) router (*next hop router*).
- El next hop se determina con una **tabla de routeo** (ver figura), y con un default en caso que no esté presente.

| NetworkNum | NextHop |
|---|---|
| 1 | R3 |
| 2 | R1 |
| 3 | Interface 1 |
| 4 | Interface 0 |

Observar cómo se utiliza agrupamiento jerárquico para reducir la cantidad de información requerida en cada router y mejorar la escalabilidad.

### ARP (Address Resolution Protocol)

Caundo se tiene un datagrama que pertenece a la misma red, se envía utilizando el enlace de esa red. Para hacer esto, se debe conocer la dirección del host dentro del esquema de direccionamiento utilizado por la misma. Hace falta, entonces, un mecanismo para traducir direcciones IP a direcciones físicas dentro de una red.

**ARP** es un protocolo que permite construir dinámicamente las tablas utilizadas para estos mapeos. Funciona de la siguiente forma:

- El host que desea enviar el datagrama chequea si tiene un mapeo para la dirección IP destino
- Si no es así, realiza un broadcast con una *ARP query* que contiene la dirección IP a resolver, aparte de la dirección IP y física del emisor
- Si un host recibe la query y ya posee una entrada en su tabla para el emisor, la actualiza aunque no sea el destino de la query
- Si algún host tiene asignada la dirección IP de la query, responde informando su dirección física y aparte guarda un mapeo para conocer la dirección física del emisor (dado que es probable que pronto tenga que responder algún mensaje de nivel de aplicación).
- En caso de recibir respuesta, el emisor se agrega una nueva entrada a la tabla ARP.

### DHCP (Dynamic Host Configuration Protocol)

A diferencia de las direcciones Ethernet (que están grabadas en ROM), las direcciones IP no pueden asignarse de antemano, ya que contienen información variable sobre la estructura de la red.

Aparte de su dirección, un host debe también conocer la dirección del router al cual debe delegar datagramas hacia afuera de la red.

**DHCP** es un protocolo para automatizar la configuración de los parámetros necesarios para operar en una red IP.

- Se basa en al menos un servidor DHCP
- El servidor puede utilizarse como un directorio centralizado configurado manualmente, pero también se puede encargar de asignar direcciones de un pool dinámicamente.
- Para descubrir el servidor, un host envía un mensaje *DHCPDISCOVER* a la ip 255.255.255.255 (broadcast)
- En caso de haber un servidor, responde al host con la información de configuración necesaria.

Otras consideraciones:

- Se podría querer tener un servidor DHCP que mantenga la información consistente para varias redes. En este caso se utiliza un **relay** en cada red, un host que reenvía los mensajes *DHCPDISCOVER* via unicast al servidor.
- Dado que no se puede depender de que los hosts liberen las direcciones, estas se asignan por un tiempo determinado a partir del cual el servidor puede volver a asignarlas (en caso de que no se hayan renovado).

### ICMP (Internet Control Message Protocol)

Si bien IP no brinda grandes garantías sobre el envío de datagramas, viene acompañado del protocolo **ICMP**, que define mensajes para reportar los errores que puedan llegar a surgir (por ejemplo, en caso de que no se pueda alcanzar un host o haya fallado el ensamblado).

### VPNs (Virtual Private Networks)

Uno podría querer brindar conectividad entre dos redes, manteniéndolas aisladas de otras. La forma más sencilla de lograr esto es utilizar cableados especiales. Sin embargo, también sería deseable crear circuitos virtuales aislados sobre una red existente. Esto permitiría, por ejemplo, reutilizar la gran infraestructura de Internet para crear redes privadas virtuales de escala global.

Para lograr esto sobre IP se crean **túneles IP**, que funcionan como un enlace punto a punto aunque encapsulen una cantidad arbitraria de redes intermedias.

- Un host en la red 1 envía normalmente un mensaje a una dirección IP interna de la red 2.
- El router de la red 1 (R1) sabe que debe enviar mensajes a direcciones de la red 2 a través del enlace virtual.
- Para enviar un mensaje a través del túnel, R1 encapsula el paquete en un nuevo datagrama IP con la dirección del router destino (R2) y lo envía.
- El datagrama viaja normalmente a través de la internet.
- Cuando R2 recibe el datagrama, ve que contiene su misma dirección, por lo cual inspecciona el contenido y encuentra otro datagrama IP con una dirección interna de su red.

Motivaciones para hacer eso:

- Enlaces seguros (complementado con encripción)
- Utilizar funcionalidad no presente a lo largo de toda la internet pero sí en los extremos (ejemplo: multicast)

- Enviar paquetes de protocolos no-IP sobre una red IP

## Intradomain Routing

- **Routing** es el proceso que se lleva a cabo de forma distribuída a lo largo de una serie de redes en los routers para dotarlos de la información que requieren para dirigir paquetes.
- El objetivo final es contruir tablas como la siguiente:

| Network Number | NextHop |
|---|---|
| 10 | 171.69.245.10 |

- En caso de circuitos virtuales, debe realizarse sólamente en el inicio de la conexión.
- Para comunicación sin conexión (datagramas) el ruteo debe realizarse por cada mensaje.
- El problema consite en encontrar el camino mínimo en un grafo, donde puede cambiar la topología los costos de los ejes.
- Se busca un protocolo mediante el cuál los nodos puedan ir descubriendo dinámicamente estos caminos.

Se habla de **ruteo interno** cuando es dentro del mismo domino/sistema autónomo (dimensiones reducidas), y de **ruteo externo** cuando ocurre entre dominios (necesidad de escala). Hay dos clases principales de protocolos de ruteo interno: **Distance Vector** y **Link State**.

### Distance Vector

Idea principal:

- En cada nodo se mantiene un vector con la distancia y siguiente paso hacia el resto de los nodos
- Inicialmente sólo se sabe el costo de llegar a los vecinos
- Los nodos vecinos se intercambian vectores repetidamente y actualizan los suyos en caso de encontrar algún camino mejor
- Ningún nodo tiene la información de toda la red, pero todos tienen información consistente sobre cuál es el mejor camino a tomar para cada caso.

Los mensajes se producen por dos razones:

- Períodicamente (sirve también como keep-alive)
- Si se detecta algún cambio en la topología o error en enlace

#### Count to infinity

Este algoritmo puede padecer del problema **count to infinity**: si un nodo -*A*- indica a otro -*B*- que conoce un camino, *B* no tiene forma de saber si él mismo forma parte de ese camino. Esto puede causar que, en determinadas condiciones, se tome mucho tiempo hasta darse cuenta que un nodo es inalcanzable. Ejemplo:

- *A*-*B*-*C*, todos los enlaces con el mismo peso
- Cae el enlace entre *A* y *B*
- *B* detecta que su enlace a *A* se cayó (por lo cuál ya no puede tomar el camino de longitud 1)
- Sin embargo, también recibe un camino de parte de *C* de distancia 2, y lo adopta sin saber que ese camino dependía de su enlace directo con *A*.
- En la siguiente actualización, *B* informará a *C* que su camino hacia *A* ahora tiene longitud 2, por lo cuál *C* ahora cree que está a distancia 3 de *A*.

Posibles soluciones:

- Limitar la cantidad máxima de hops para acotar la cuenta hasta infinito (podría traer problemas si la red crece)
- **Split horizon**: No se informa a al vecino las rutas que se aprendieron de él mismo (funciona sólo para loops entre dos nodos)
- Demorar las actualizaciones para evitar las condiciones de carrera (demora la convergencia)

**Ejemplo: RIP**

- **RIP** es un protocolo que implementa el algoritmo de distance vector de forma bastante directa.
- Los vectores indican direcciones desde el nodo hacia redes conocidas (no hacia routers)
- Asume que todos los enlaces tienen el mismo peso.
- Se utiliza un máximo de hops para evitar conteo al infinito. Esto limita la aplicación de RIP a internets chicas.

## Link State

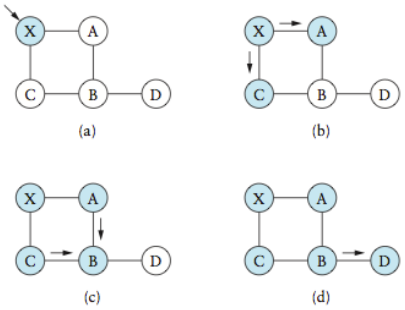
Los algoritmos de **link state** se basan en los iguiente:

- Cada nodo sabe llegar a sus vecinos, y disemina este conocimiento (*link state*) en toda la red
- Acumulando el *link state* del resto, todos los nodos pueden calcular rutas a los otros.

Observar que cada nodo transmite sólamente su *link state* y la reconstrucción de la topología se basa en acumular estos *link states*. En los algoritmos distance vector, por otra parte, cada nodo resumía todo el conocimiento que tenía y se iban propagando estos "resúmenes".

El mecanismo por el cual todos los nodos conocen el link state del resto se llama **Reliable Flooding**:

- Períodicamente o en caso de cambios de la topología, cada nodo envía a sus vecinos un paquete con su *link state*, número de secuencia y TTL.
- Cuando el vecino lo recibe, verifica mediante el número de secuencia si es más reciente de la que ya tiene.
- Si es así, reenvía a todas las interfaces (excepto aquella por la cual llegó el paquete) la nueva información.
- Para asegurarse de que eventualmente los paquetes viejos se descartan, se utiliza un TTL
- En caso de caída de un nodo, éste arranca con número de secuencia 0. En caso de que haya información vieja todavía sin caducar, eventualmente llegará al nodo y éste podrá aumentar su número de secuencia como sea necesario.




Aquí el término *reliable* significa que se tiene la última información y se descartan las versiones contradictorias. Una vez se tiene distrbuída la información de *link state*, cada nodo puede reconstruir toda la

topología por sí mismo y calcular los caminos mínimos. Esto se hace mediante una variación del algoritmo de Dijkstra (*forward search algorithm*).

- La idea principal es inicar con el nodo actual e ir explorando utilizando BFS el grafo actualizando la tabla de resultado cuando se encuentra un camino con menor costo que el ya conocido.
- A nivel implementativo, se utiliza una lista de *caminos confirmados* y otra de *caminos tentativos*.
- La nueva información se actualiza siempre en la lista de tentantivo.
- Al final de cada paso, se promueve a confirmado el dato de menor costo de los tentativos.

Estos algoritmos se estabilizan rápido y no generan demasiado tráfico, además de responder rápidamente a cambios en la topología o falla de nodos. Sin embargo, tienen la desventaja de que se debe almacenar una gran cantidad de informción en cada nodo (lo cuál atenta contra la escalabilidad de la red).

### Áreas

- Para permitir escalar la dimensión dentro de un dominio, se puede subdibidir un dominio en **áreas**.
- La información de floodeo se mantiene dentro del área
- Hay siempre un área especial denominada *backbone* (dentro del dominio, distinto a SAs backbones de Internet)
- Cada área tiene un router frontera expuesto al área backbone.
- Los routers frontera se intercambian información sumarizada de cómo alcanzar redes de su área.
- Todo el tráfico pasa entonces por el backbone.

#### Ejemplo: OSPF

**OSPF** (Open Shortest Path First) es un protocolo de tipo link state que agrega lo siguiente:

- autenticación: para evitar que hosts mal configurados anuncien distancia cero a todos lados y se conviertan en un "agujero negro"
- jerarquización: permite partir un dominio en áreas, de modo que entre áreas la cantidad de información que viaja es menor. un router sólo debe saber cómo dirigir un paquete al área que corresponde.
- balanceo de carga: permite distribuir carga cuando se cuenta con varias rutas del mismo costo.

Además, permite priorizar ciertas rutas asignando pesos a los enlaces para determinado tipo de tráfico (utilizando un campo de type of service en los paquetes de configuración).

Tipos de mensajes:

- *Hello* (discovery / heartbeat)
- *Link state update*
- *Link state ack*
- *Database description*
- *Link state request*

# Internet

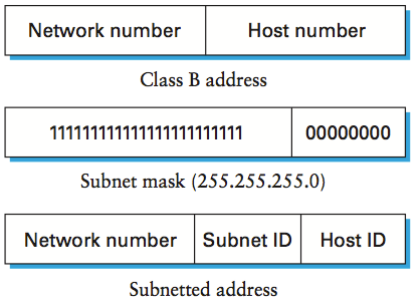
La implementación de una internet global trajo dos grandes problemas de escala: **routeo** (minimizar la cantidad de información que hace falta distribuir para poder routear) y **direccionamiento** (asegurarse de que no se consuma por completo el espacio de direcciones).

## Subnetting

El esquema de tres clases de direcciones IP le asigna un número a cada red. Esto provoca un uso ineficiente del espacio de direcciones: se puede tener una red de 2 hosts que use una dirección clase C (~255 direcciones) o una red de poco más de 255 host que use una dirección de clase B (~64K direcciones). A su vez, asignar una gran cantidad de números de red complica el routeo, ya que implica almacenar y transmitir más información entre routers.

Una solución a esto es realizar **subnetting**, asignar todas las IP dentro de un número de red a varias redes físicas. Asumiendo que las redes están cerca geográficamente (para no perjudicar el routeo) este esquema permite aprovechar más el espacio de direcciones.

Para poder compartir una dirección de red entre varias redes físicas, todos los hosts se configuran con una **máscara de subred**. Esto agrega un nuevo nivel de jerarquía a la dirección IP. La porción de la dirección que escape a la máscara será la que identifique al host:




Cuando un host quiere enviar un paquete a otro, utiliza la máscara para ver si ambos pertenecen a la misma subred (es decir, si el "and" binario de ambas contra la máscara arroja el mismo resultado). Si es así, puede enviarse dentro de la subred. En caso contrario, se envía al router.

Notar que utilizando subnetting cambia el mecanismo de forwarding de los routers: ahora deben conocer cada subred y su máscara en vez de números individuales de red. Cuando llega un paquete, se busca la subred que concuerde con la aplicación de la máscara a la dirección de destino.

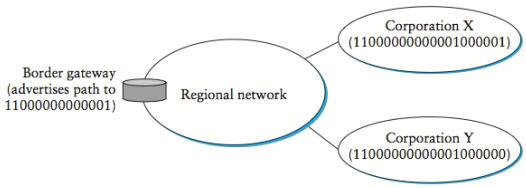
| SubnetNumber | SubnetMask | NextHop |
|---|---|---|
| 128.96.34.0 | 255.255.255.128 | Interface 0 |
| 128.96.34.128 | 255.255.255.128 | Interface 1 |
| 128.96.33.0 | 255.255.255.0 | R2 |

## Classless routing (CIDR - supernetting)

En vez de asignar una dirección de clase B para redes no tan grandes, se pueden asignar varias de clase C para aprovechar más eficientemente es espacio de direcciones. Sin embargo, implica mantener información de routeo para cada una de esas redes. Una alternativa para solucionar este problema sería otorgar varias redes tipo C con direcciones **consecutivas**. De esta forma, se puede direccionar la cantidad exacta deseada de equipos con una sóla entrada en cada tabla de routeo.

A este enfoque se lo denomina **supernetting**:

- En subnetting se divide un único número de red para que sea utilizado en varias redes físicas.
- En supernetting se agrupan varios números de red para direccionarlo de forma única.




De forma similar a como ocurría con subnetting, será necesario tener un sistema de routeo que utilize direcciones con prefijo de red variable.

## Interdomain Routing

Un **sistema autónomo** (SA) es una unidad administrada independientemente del resto de Internet. Organizar la red en SAs permite:

- desacoplar la administración técnica de casa SA (permitiendo que dentro de cada uno se use un mecanismo de intradomain routing)
- reducir la cantidad de información necesaria para routear, a través de la organización jerárquica y el uso de rutas default.

En Internet se implementaron dos protocolos de interdomain routing:

- EGP (Exterior Gateway Protocol): no utilizado actualmente, limitado a topologías arbóreas con un único backbone.
- BGP (Border Gateway Protocol):

### BGP

- Internet está orgizada de forma compleja con varios backbones desde los cuales se organizan ISP de distintos niveles
- BGP entiende a Internet como un grafo de SAs para no imponer una topología específica
- Se llama **tráfico local** al que se origina y termina dentro de un mismo SA
- Se llama **trafico en tránsito** al que sól pasa a través de un SA

Clasificación de SAs:

- *Stub AS*: tiene conexión a un sólo SA. Sólamente se encargará de tráfico local.
- *Multihomed AS*: tiene conexión a más de un SA, pero no lleva tráfico en tránsito.
- *Transit AS*: tiene conexión a más de un SA, y está diseñado para llevar tráfico loca y en tránsito (ejemplo: backbones)

Características:

- Debido a la complejidad del problema, no se busca necesariamente una ruta óptima, sino que alcanza con encontrar alguna.
- Cada SA tiene al menos un *speaker*.
- Los speakers de distintas SAs se intercambian información sobre qué puntos pueden alcanzar y cómo.
- A diferencia de los algoritmos de distance vector y link state, los SAs se intercambian rutas enteras (secuencias de SAs para llegar a una red).
- Se configuran políticas para elegir qué rutas conviene elegir y cuáles publicar.
- Tener rutas enteras también permite evitar ciclos.

### Combinando BGP con intradomain routing

- BGP indica cómo hacen los speakers de los SAs para intercambiarse información de routeo
- Los speakers deben tener un mecanismo para que estos datos sean conocidos por los routers internos.
- En general se inyectan los datos utilizando el mecanismo de routing intradomain que corresponda, y configurando adecuadamente las rutas default.
- En casos como los backbones, se utiliza una variante "interna" de BGP (IBGP) para que cada router sepa cuál es el mejor punto de salida para una ruta, y routing intranetwork regular para que sepan cómo llegar a esos routers.

# Routeo Multicast

- Multicast implementado en hardware para redes locales (ethernets, token rings)
- Protocolos para permitir multicast en redes globales
- Motivación: enviar un paquete a dirección multicast (IPs clase D) y que la red lo entregue a todos los host suscriptos

## Link state multicast

- Se extiende link state normal agregando información sobre qué grupos tienen miembros
- Un host anuncia periódicamente a qué grupos pertenece
- Con esta información se computan los **multicast trees** (árboles con los caminos mínimos a cada host del grupo desde un nodo fuente)

## Distance vector multicast

Mecanismo de **Reverse Path Broadcast**:

- Cada vez que llega un paquete desde una fuente S, forwardearlo a todos los puertos si el paquete llegó del shortest-path a S (esto evita ciclos y floodea la red)
- Para cada source, se designa un router parent para cada LAN (el que está más cercano), y sólo éste mete el paquete en la red (para evitar que entre por muchos lados)

Sobre el broadcast descrito anteriormente, se prunean las redes donde no hay miembros del grupo multicast. Esto se conoce como **Reverse Path Multicast**

- Se identifican los nodos hoja (redes con un único router)
- El router conectado a una hoja puede saber si no hay miembros (mediante los anuncios de los hosts), y evitar forwardearlo
- Cada router propaga hacia la internet cuáles son los grupos en los que está interesada la red
- Al final de la propagación, cada router sabe qué grupos hay con suscriptores en cada link

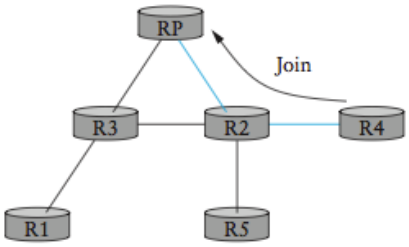
Dado el overhead de agregar esta información en los routers (los grupos activos) en la práctica, se utiliza broadcast hasta que se activa una dirección multicast y ahí se hace el pruneo. De todos modos, esto tiene problemas de escalabilidad.

## Prototol Independent Multicast (PIM)

Los algoritmos anteriores no se comportan bien cuando son muy pocos los routers interesados en recibir información de ciertos grupos. **PIM** es un protocolo que tiene dos modos de funcionamiento, *sparse mode* y *dense mode*.

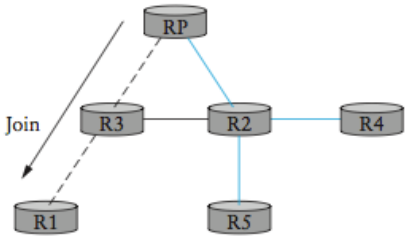
Características del sparse mode:

- Los nodos acuerdan entre sí designar a uno como **rendezvous point** (RP) de cada grupo.
- Se envían mensajes de *join* y de *prune* al RP utilizando unicast.
- A medida que los mensajes de join viajan hacia el RP, cada router anota en su tabla que cuando llegue un paquete para el grupo se debe forwardear por el puerto de donde provino el join (estado **<*, G>**).
- De esta forma, cada join agrega una rama al **shared tree**.
- Para enviar mensajes al grupo, se envía el mensaje mediante un túnel al RP (encapsulándolo en un datagrama unicast), y el mismo lo distribuye en el árbol.

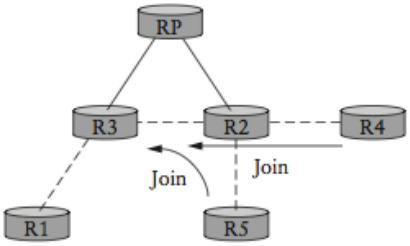



Optimizaciones:

- Si se observa gran cantidad de tráfico en un tunel hacia el RP, el RP puede enviar al sender un sender-specific join (agregando estado **<S, G>** a lo largo del camino), de modo que *para ese sender* el árbol comartido tendrá distinta raíz, y se evitará el overhead de encapsulamiento.




- Dado que el camino no es óptimo, cuando se observa mucho tráfico en una ruta mejorable se puede reemplazar todo el árbol compartido por un **source specific tree**. Para esto, cada router destino envía un source specific join al origien (agregando estado **<S,G>** en los routers del camino) de modo que para ese sender no hará falta pasar por la raíz.




Propiedades:

- se dice que es *protocol independant* porque puede utilizarse sobre cualquiera sea el mecanismo de ruteo unicast.
- el uso de shared trees aumenta la escalabilidad reduciendo la cantidad de estado que hay que tener en cada router (proporcional a la cantidad de grupos, y no al producto de cantidad de grupos por senders)
- cuando se requiere mayor eficiencia, se utilizan source specific trees (trade off entre escalabilidad y optimalidad)

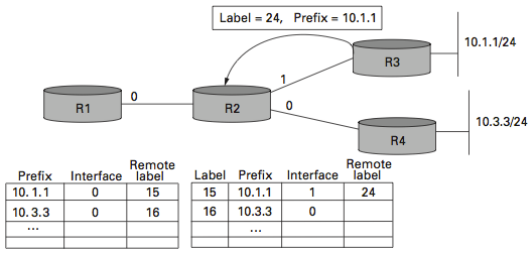
# MPLS (Multiprotocol Label Switching)


**MPLS** es una tecnología que combina algunas características de los circuitos virtuales con la robustez y flexibilidad de los datagramas. A grandes rasgos, consiste en attachear *antes del header IP* una serie de tags a los datagramas IP, que pueden ser utilizados para tomar decisiones a lo largo del routeo. Hoy se usa para:

- Utilizar funcionalidades de IP en dispositivos que no soportan el forwardeo de datagramas IP tradicional.
- Para forzar el routeo a través de rutas explícitas que pueden no ser las normales definidas por routeo IP.
- Para proveer algunas funcionalidades de VPN

### Uso: destination based routing

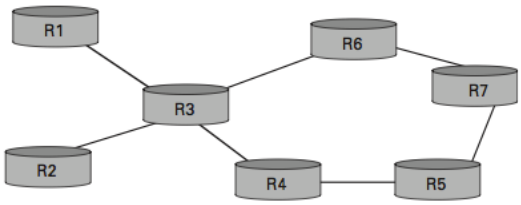
MPL define el protocolo **LDP** (Label Distribution Protocol) con con cuál un router comunica a sus adyacentes que desea recibir una etiqueta específica para paquetes de determinada red. De esta forma, se puede tener indexada la tabla de routeo por etiqueta. Así, al recibir un paquete no deberá hacer una búsqueda del mayor prefijo para conocer la ruta, sino que alcanzará por acceder a la entrada que matchee (de forma exacta) la etiqueta.




Este mecanismo permite implementer el mismo algoritmo de routeo (cualquiera sea) con algoritmos de forwarding más sencillos. Esto posibiltó hacer compatibles con IP switches de otros tipos de redes (ATM) sólo con actualizaciones de software.

### Uso: routing explícito

Supongamos que en la imagen siguiente se quiere que el tráfico proveniente de R1 a R7 vaya por la rama superior, y en proveniente de R2 por la rama inferior. Usando routeo tradicional sería imposible, ya que el mismo no tiene en cuenta el origen de los paquetes.




Suponiendo que se tienen las tablas de routeo MPLS configuradas adecuadamente, se podría lograr haciendo que R1 y R2 agreguen distintas etiquetas. Sin embargo, la distribución de etiquetas descrita anteriormente (*LDP*) no permite configurarlas de este modo, ya que sirve para emular el routeo tradicional de IP.

MPLS describe otro protocolo, ***RSVP*** (Resource Reservation Protocol) para lograr este objetivo. Consiste en enviar mensajes de configuración a lo largo del camino deseado que crean las entradas requeridas para el forwarding.

Esto es útil para **traffic engineering**, donde se busca asegurar que habrá suficientes recursos a lo largo de la red. Aparte de definir los caminos a tomar para partes del tráfico, se pueden precalcular rutas que eviten ciertos nodos, y utilizarlas (agregando la etiqueta que corresponda) para evadir nodos caídos sin tener que esperar a que ejecuten nuevamente los algoritmos distribuídos de routeo (**fast rerouting**).

## USO: VPNs y túneles

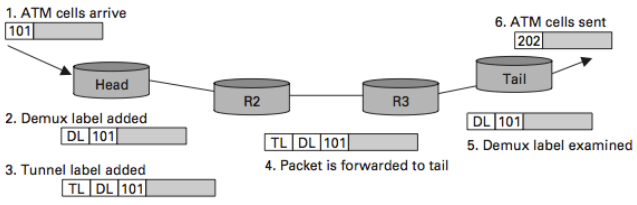
MPLS puede ser utilizado para construir túneles, que a su vez son un mecanismo útil par armar VPNs.

#### Layer 2 VPN

Una de las formas es armando una **layer 2 VPN**, que se basa en túneles que transmiten datos de capa 2 (frames ethernet o células ATM, por ejemplo) a través de una serie de routers. Notar que si el routeo se basa solamente en los labels, no hace falta que se transfiera contenido IP.

Ejemplo:

- Se quiere enviar tráfico no IP entre R1 y R2 (separados)
- R1 agrega un label de demultiplexación (para que R2 sepa cómo tratar el contenido)
- R1 agrega un label de routeo (como se explicó antes en destination based routing)
- La red transfiere el contenido utilizando los labels, sin necesidad de inspeccionar el contenido (¡que no es un datagrama IP!).
- R2 desencapsula el contenido: remueve el label de routeo y en base al label de demultiplexación sabe introducir el contenido en su red.




Observar que esta funcionalidad también puede lograrse usando túneles IP tradicionales, aunque MPLS tiene la ventaja de que ahorra ancho de banda debido a su header pequeño.

**Layer 3 VPN**

Una **layer 3 VPN** es otro tipo de VPN creada utilizando túneles MPLS. Es decir, routean utilizando labels que encapsulan los mensajes. La diferencia es que en este caso los mensajes encapsulados sí son datagramas IP.

La implementación es compleja, pero se utiliza para definir redes IP "vitualmente privadas". Es decir, redes IP basadas sobre internet donde los equipos están aislados y manejan su propio espacio de direcciones.

# Unidades 7 y 8: Nivel de Transporte y Congestión

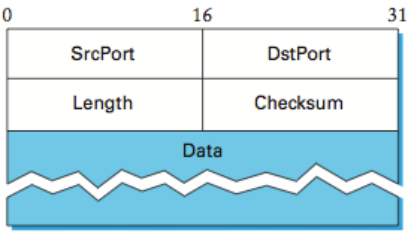
Hasta ahora la capa de red provee la funcionalidad de conectar dos host a través de una red. Debido a la dificultad del problema, provee un servicio best effort.

El objetivo final es comunicar procesos que requieren ciertas garantías sobre el canal de comunicación. El nivel de transporte se encargará de proveer la **demultiplexación a nivel proceso** y un conjunto de **garantías** útiles para las aplicaciones.

En Internet se utilizan básicamente dos protocolos de nivel de transporte: **UDP** (sin conexión y no confiable) y **TCP** (con conexión y confiable).

## UDP

**UDP** (User Datagram Protocol) es un protocolo de transporte que ***sólo provee demultiplexación entre procesos*** y ***chequeo de integridad***, sin agregar garantías sobre la trasmisión de paquetes.




- Para identificar procesos podría usarse el *pid* asignado por el sistema operativo, haría el protocolo muy dependiente de cómo el SO asigna números de proceso.
- En vez de esto, se utilizan direcciones más abstractas, **puertos** que funcionan a modo de casillas.
- Un proceso envía datos a un puerto en el host de destino, y el proceso receptor recibirá la información de ese puerto.
- La clave de demultiplexación es, entonces, el par <host, port>

Alternativas para conocer el puerto destino:

- Convención de números de puertos (ejemplo: 80 para web)
- Tener un puerto conocido para negociar otro puerto
- Port Mapper: servicio escuchando en un único puerto conocido, con protocolo para averiguar el puerto donde reside el servicio buscado

El chequeo de integridad es un checksum sobre:

- Header UDP
- Contenido del mensaje
- Pseudoheader (campos *protocol version*, *source address* y *destination address* del header IP).

Observar cómo aquí se rompe la capa de abstracción, acoplando UDP a IP. El chequeo de los campos de IP es para descartar mensajes dirigidos a otro destinatario que puedan haber llegado erróneamente.

# TCP

**TCP** (Transmission Control Protocol) es un protocolo de nivel de transporte con las siguientes características:

- Garantiza la transmisión ordenada de un stream de bytes
- Provee demultiplexación (como UDP)
- Orientado a conexión full duplex
- Mecanismo de **flow control** (para que no se transmita más de lo que puede procesar el receptor)
- Mecanismo de **congestion control** (para que no se sature la red)

## Características de la comunicación entre procesos

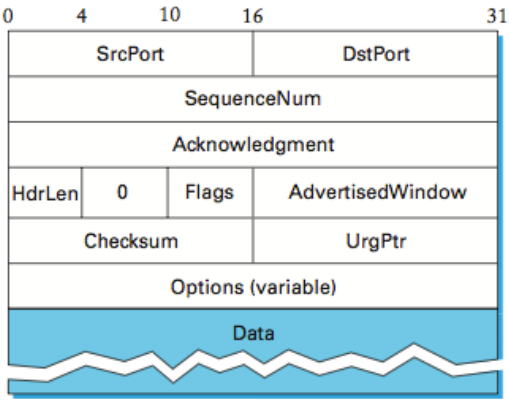
TCP debe lidiar con algunos de los problemas que también se atacaron en la capa física (control de errores, secuenciamiento, control de flujo). Sin embargo, en esta capa se deben tener en cuenta ciertas consideraciones:

- **Establecimiento de conexiones**: Una conexión TCP puede realizarse con cualquier otro host de Internet, a diferencia de contar con un canal fijo (como puede ser un cable).
- **Timeout adaptativo**: El delay de la transmisión es desconocido y variable, a diferencia de los físicos. Es más complicado saber cuándo declarar timeout para retransmitir un paquete.
- **Almacenamiento de la red**: Un paquete puede quedar almacenado en la red y llegar a destino varios segundos más tarde.
- **Necesidad de flow control**: No se tiene garantías de la cantidad de recursos que dispone el receptor para la conexión, por lo cuál el protocolo se debe encargar de averiguar cuánto se puede enviar en un determinado momento.
- **Necesidad de congestion control**: No se sabe conoce la capacidad de la red, por lo que el protocolo debe saber cómo prevenir que la red se inunde de mensajes que no llega a transmitir.

## Formato de segmento

Observaciones:

- TCP ofrece una interfaz de byte stream, pero lo que efectivamente se envían son segmentos.
- El host emisor guarda en un buffer una cantidad suficiente de bytes que luego son enviados juntos.
- El receptor, a su vez, guarda en buffers los segmentos recibidos que se van vaciando a medida que el proceso lee.



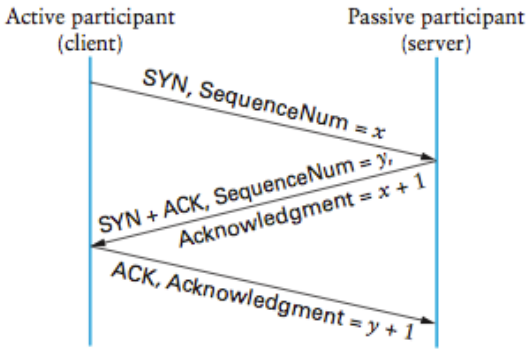

- SrcPort, DstPort: demultiplexación
- SequenceNum, Acknowledgement, AdvertisedWindow: usados para sliding window
- HdrLen: longitud del header (variable)
- Flags: información de control
- UrgPtr: indica dónde empiezan los datos no urgentes (si está habilitado el flag URG)
- Checksum: control de integridad

## Conexión

El setup de la conexión se inicia cuando el caller hace un **active open** a un callee que haya hecho un **pasive open** (osea, que esté escuchando en ese puerto). Una vez establecida la conexión, puede ocurrir que sólo un extremo la cierre y que el otro siga mandando datos.

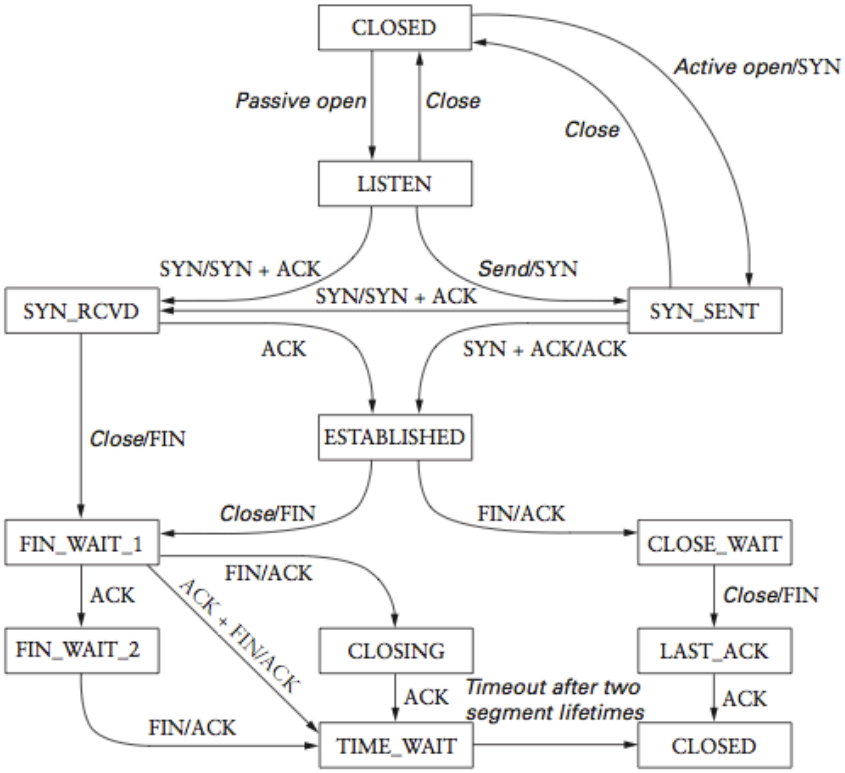
### Three way handshake

TCP requiere que se utilicen números de secuencia aleatorios para minimizar el riesgo de que interfieran segmentos de otra encarnación de la conexión (una conexión previa con mismo par de puertos entre los dos hosts).




Al iniciar la conexión, ambos lados realizan un intercambio conocido como **Three way handshake** para ponerse de acuerdo en estos parámetros. Cada mensaje del intercambio tiene un timer asociado, y es retransmitido en caso de que no llegue un acknowledge a tiempo.

### Estados de la conexión



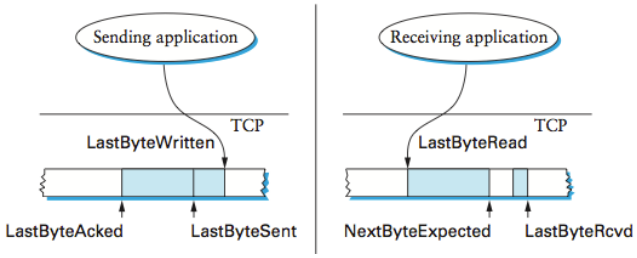

Notar:

- Si se puerde el último ACK del three way handshake no pasa nada. El servidor quedará en estado SYN_RCVD y el cliente en ESTABLISHED (osea que ya puede enviar). En el primer segmento de datos que envíe el cliente habrá también un ACK, lo que hará que el servidor pase a ESTABLISHED.
- La mayoría de los mensajes tiene un timeout y se retransmite si no llega a tiempo el ACK. Si tras varios intentos no llega, se corta la conexión.
- Ambos extremos deben cerrar su lado de la conexión (en caso contrario pueden seguir enviando).

## Sliding window en TCP

### Garantía de entrega y orden de paquetes

Se logra de forma similar a como ocurre en la capa física, teniendo encuenta también que se bufferean los datos generados y no enviados (en el lado del emisor) y los recibidos en orden pero no aún leídos (en el lado del receptor.)



## Control de flujo

Sliding window también se utiliza para evitar que el sender envíe más datos de lo que el receiver puede guardar en buffer. Sin embargo, el tamaño de la ventana no depende solamente del tamaño del buffer:

- Si la aplicación no consume, se debe seguir guardando en buffer.
- Si llega un segmento pero no están todos los anteriores, no se envia ACK y se guarda en buffer.

Esto hace que el temaño de la ventana (es decir, la cantidad de bytes que puede haber enviados sin haber recibido ACK) no sea fijo sino que dependa del estado del buffer, el orden de los datos que llegan y el ritmo al cual la aplicación consume.

Lo que se hace es que, con cada ACK, el receiver comunica al sender el tamaño de ventana actual. En caso de publicarse una ventana de tamaño 0, el sender envía períodicamente segmentos de 1 byte esperando que alguno sea aceptado.

## Invariantes

El tamaño del buffer del receptor limita la cantidad de bytes no leídos almacenados:

$$LastByteRcvd - LastByteRead \leq MaxRcvBuffer$$

No puede haber más bytes sin ACK de lo que indica la ventana publicada:

$$LastByteSent - LastByteAcked \leq AdvertisedWindow$$

La aplicación que envía no puede escribir más de lo que soporta su buffer (en caso de querer escribir más, se bloquea):

$$LastByteWritten - LastByteAcked \leq MaxSendBuffer$$

## Posibles problemas

#### Wraparound del SequenceNumber

Dado que el campo SequenceNumber sólo tiene 32 bits, con altas tasas de transferencia podría ocurrir un wraparound de este campo en un tiempo menor al máximo permitido para que un paquete TCP expire. Sería posible, entonces, que llegue un paquete viejo con un sequence number de una "vuelta anterior" y se confunda con los transmitidos actualmente.

Para solucionar esto, se envía un timestamp en la sección opcional del header y efectivamente se chequean con los 32 bits del sequence number y los 32 del timestamp. A fines de ordenar paquetes, sólo se utiliza el número de secuencia.

#### AdvertisedWindow muy pequeña

Dado que el campo AdvertisedWindow tiene sólo 16 bits, el máximo tamaño de ventana publicable es 64KB. Esto puede resultar muy poco para mantener lleno un enlace de alta capacidad, provocando subutilización de la infraestructura.

Para solucionar esto, se utiliza otro campo opcional en el header que funciona a modo de factor de escala del AdvertisedWindow. El mismo indica que el AdvertisedWindow no contará cantidad de bytes sino porciones

más grandes.

## Retransmisión adaptativa

Para garantizar el envío de datos, TCP reenvía segmentos cuando considera que pueden no haber llegado a destino. La variabilidad de los RTT entre las redes y a través del tiempo hace que no se pueda definir un timeout fijo a partir del cuál reenviar. Por esto, TCP necesita algún criterio adaptativo que defina en un determinado instante cuál es el timeout a asignarle a un segmento transmitido.

### Algoritmo original

El algoritmo original para calcular el timeout funcionaba así:

- Se lee un timestamp del reloj del sistema operativo y se envía dentro de la sección opcional del header.
- Al enviar el ACK, el receptor copia el mismo timestamp
- Al llegar el ACK del mismo, se calculaba un SampleRTT con el timestamp de emisión y un nuevo timestamp.
- Se calcula el nuevo EstimatedRTT como un promedio ponderado con el estimado anterior:

$$EstimatedRTT = \alpha \times EstimatedRTT + (1 - \alpha) \times SampleRTT$$

$$Timeout = 2 \times EstimatedRTT$$

### Algoritmo de Karn/Patridge

Dado un ACK, es imposible saber a qué transmisión de ese dato corresponde. Esto puede hacer que tomemos SampleRTT no representativos. Para solucionar esto, el algoritmo de Karn/Patridge propone tomar samples sólamente de paquetes que no requirieron retransmisión.

Además de esto se utiliza exponential backoff para retransmisiones, con el objetivo de evitar sobrecongestionar la red.

### Algoritmo de Jacobson/Karels

El algoritmo anterior no soluciona por completo el problema de congestión. Entre otras mejoras para reducir la congestión, Jacobson y Karels propusieron un algoritmo para calcular el timeout.

Este algoritmo tiene en cuenta la variación de RTT.

$$Difference = SampleRTT - EstimatedRTT$$

$$EstimatedRTT = EstimatedRTT + (\delta \times Difference)$$

$$Deviation = Deviation + \delta(|Difference| - Deviation)$$

$$TimeOut = \mu \times EstimatedRTT + \phi \times Deviation$$

Con valores cercanos a $\mu = 1, \phi = 4$. Con varianza pequeña, el timeout es cercano al EstimatedRTT. Varianzas mayores hacen que el término de desvío domine el timeout.

## Delimitación de segmentos

Queda el problema de decidir en qué momento dejar de bufferear y enviar un segmento., Asumiendo una ventana arbitrariamente grande, tres eventos pueden hacer que esto ocurra:

- Se alcanza el **MSS** (Maximum Segment Size, ~ MTU inmediato).
- Push por parte de la aplicación.
- Timer (ver adelante)

### Silly window sindrome

El **silly window sindrome** es un problema que ocurre cuando la capacidad de recepción es muy pequeña y el sender es muy agresivo. Es decir, intenta enviar constantemente apenas se abre una pequeña ventana de transmisión.

Cuando esto ocurre, cualquier espacio de ventana pequeño que se utilice será ocupado enseguida. Luego, el receptor volverá a publicar un pequeño espacio de ventana, que nunca será combinado para lograr un tamaño de ventana mayor.

Hay dos soluciones parciales a este problema:

- Haciendo que el receptor aguarde a que haya una capacidad considerable (~MSS) para abrir la ventana luego de que se cierre completamente.
- En caso de que el emisor genere muchos segmentos pequeños (por ejemplo, por uso de *push*) se pueden retrasar los ACK. Es decir, esperar varios segmentos y enviar un ACK por todos ellos al final.

Sin embargo, no se puede retrasar indefinidamente el envío del emisor sólo para evitar este problema. Hace falta que el mismo emisor se limite a no enviar paquetes pequeños cuando esto es posible.

### Algoritmo de Nagle

Si bien enviar paquetes muy pequeños puede hacernos caer en el silly window sindrome, esperar mucho para enviar puede perjudicar aplicaciones interactivas. Esto se podría solucionar con un *timer*.

El algoritmo de Nagle describe un mecanismo por el cual se utilizan ACKs que llegan como timers que disparan el envío de segmentos con todos los datos que se hayan acumulado hasta el momento.

Idea:

- Se puede enviar si hay un segmento completo y la ventana lo permite.
- Si no hay datos en tránsito, se puede enviar segmentos pequeños.
- Si hay datos en tránsito, se debe esperar a que lleguen los ACK para enviar.

Esto hace que la misma red actue como reloj: el buffer se vaciaría en intervalos de tiempo 2*RTT (con lo que se haya acumulado hasta el momento).

# Congestión

## El problema

El tráfico que fluye en una red requiere que se destinen recursos en los dispositivos a lo largo de toda la misma. En cada paso de transmisión, cada paquete compite con otros (posiblemente de otros hosts) por ancho de banda en el enlace siguiente o por espacio de buffer en caso de ser encolado. Cuando un

router/switch no está en condiciones de seguir almacenando paquetes debe descartarlos y en este caso se dice que está **congestionado**.

A diferencia del control de flujo, en donde se busca evitar que el emisor transmita más de lo que el receptor puede leer/almacenar, en control de congestión el recurso limitante es la capacidad de la red.

## Taxonomía de los métodos

Los métodos para lidiar con congestión se pueden clasificar de la siguiente forma:

- Lazo abierto: no utilizan feedback de la red
- Lazo cerrado: utilizan feedback de la red.
    - Feedback explícito: la red envía señales a los host (implementación más costosa)
    - Feedback implícito: cambia el comportamiento de la red y el host infiere qué sucede

## Flujos

En redes con conexión se puede optar reservar recursos en la etapa de setup. Esto lleva a subutilización de la infraestructura.

Tal como ocurre en IP, se considera que a nivel de red no se tienen conexiones. Sin embargo, si bien los datagramas pueden ser ruteados por canales distintos, en la práctica ocurre que muchas veces hay **flujos** de paquetes que atraviesan el mismo conjunto de routers. Distinguir estos flujos permite mantener cierto **soft state** para tomar decisiones en los mecanismos de control.

## Encolamiento

El algoritmo de encolamiento elegido para los switches de la red es fundamental ya que afecta directamente el delay. Puede hacer que un paquete sea descartado, o determinar cuánto tarda en transmitirse por el enlace.

**FIFO** es la estrategia más utilizada, lo que implica que se delega toda la responsabilidad del control de congestión a los extremos: esta estrategia no tiene en cuenta el flujo al que pertenece un paquete ni ningún parámetro sobre la congestión de la red más que la capacidad actual de su buffer.

Delegar toda la responsabilidad a los extremos tiene un problema grande: por más de que haya una estrategia que supera el problema de congestión (por ejemplo, la de TCP) no nos podemos asegurar que los hosts efectivamente la utilicen.

**FQ** (fair queueing) es una alternativa que básicamente mantiene colas distintas por cada flujo y las atiende utilizando *round robin*. En la implementación, para mantener el *fairness* requiere un mecanismo para aproximar el round robin bit a bit, ya que el tamaño de paquetes no es fijo.

## TCP Congestion Control

Características generales:

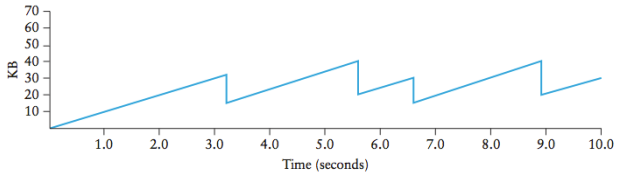
- Se realiza desde los hosts, enviando paquetes sin reserva y reaccionando al comportamiento de la red.
- No se asume nada especial sobre la política de encolamiento de routers.
- Esencialmente, un host evalúa cuántos paquetes puede enviar, e interpreta cada ACK como un

permiso para enviar otro en reemplazo del que acaba de salir de la red.

Hay tres mecanismos básicos que definen el control de congestión de TCP:

- Additive Increase / Multiplicative Decrease
- Slow Start
- Fast Retransmit - Fast Recovery

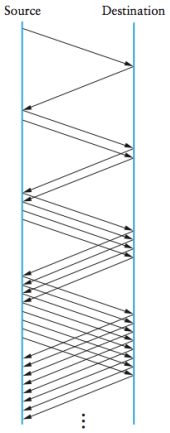
## Additive Increase / Multiplicative Decrease




- Se define nueva variable **CongestionWindow** (contraparte de AdvertisedWindow) y se modifica funcionamiento de TCP para que tome en cuenta el mínimo entre CongestionWindow y AdvertisedWindow.
- El valor de CongestionWindow se reduce cuando se detecta una suba en la congestión, y se aumenta cuando baja.
- Se toman los timeouts como indicadores de congestión (despreciando los errores de transmisión).
- Cada vez que ocure un timeout, se baja CongestionWindow a la mitad (con un mínimo en el MSS).
- Cada vez que llega un ACK, la ventana se incrementa por una fración del MSS.
- La idea es que tener una CongestionWindow demasiado grande cuando hay congestión produce más congestión, por lo cual es necesario salir rápido de esa situación.

## Slow Start

El comportamiento de AIMD sirve cuando se está operando cerca de los límites permitidos por la red, pero es muy conservador arrancar todas las conexiones transmitiendo con ventana mínima y subiendo linealmente. **Slow Start** hace este incremento exponencial.




- Se incremente en uno el CongestionWindow por cada ACK. Esto tiene el impacto de incrementar exponencialmente el tamaño de ventana cada RTT.
- El nombre se debe a que, a pesar de subir exponencialmente, se arranca desde un valor pequeño, tanteando la capacidad de la red para no enviar un paquete del tamaño máximo permitido por

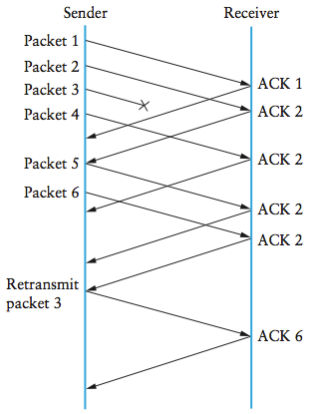
AdvertisedWindow de entrada (como ocurría anteriormente)

Además de en el inicio de la conexión, el mecanismo se utiliza cuando se comienza a enviar de nuevo tras la pérdida de paquetes. Escenario:

- Se pierden un segmento, y *source* llegó al punto que ya agotó todo el AdvertisedWindow por lo que no puede seguir enviando.
- Dado que el ACK nunca llega, eventualmente ocurre un timeout, y *source* considera que ya puede comenzar a enviar nuevamente un segmento del tamaño de AdvertisedWindow.
- En vez de hacer esto realiza slow start, comenzando del mínimo y aumentando hacia el último valor conocido de CongestionWindow (y luego siguiendo con additive increase).
- Para implementar esto, se introduce una nueva variable **CongestionThreshold**, que se setea igual a CongestionWindow tras cada multiplicative decrease.
- Slow start no se utiliza con timeouts ocurridos por fast retransmit (ver siguiente)

### Fast Retransmit - Fast Recovery

- El mecanismo tradicional de timeouts de TCP (self-clocked por RTT) daba lugar a varios períodos de inactividad en los cuales no ocurría nada hasta que se disparara un timeout.
- **Fast retransmit** es un mecanismo adicional que puede disparar retransmisión antes que el timeout ocurra.
- Se implementa haciendo que el receptor envíe un ACK por cada paquete que llega (aunque sea fuera de orden y tenga que mandar un ACK duplicado de un paquete anterior).
- Cuando el emisor detecta cierta cantidad de ACK duplicados, retransmite el segmento con número siguiente al ACK (sin necesidad de esperar el timeout).




Observar que cuando se dispara un timeout por fast retransmit todavía pueden quedar ACKs en camino. **Fast recovery** utiliza estos ACK para proceder con la dinámica común de AIMD, evitando la fase de slow start. En definitiva, slow start sólo se utilizará al principio de la conexión y cuando ocurra un timeout regular.

## Congestion Avoidance

TCP está diseñado para reducir la congestión una vez que ocurre. Incluso, para detectar el estado de la red aumenta el tráfico hasta que se detectan pérdidas de paquetes.

No incluye, sin embargo, mecanismos para sensar la capacidad de la red y controlar el tráfico emitido antes de que la misma se congestione. Los siguientes son mecanismos con ideas que podrían usarse también en TCP para evitar la congestión.

### DECbit

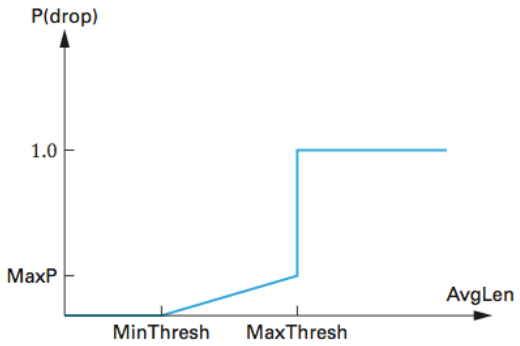
- Cada router monitorea su estado y notifica a los hosts cuando está congestionado.
- La notificacións se implementa seteando un bit en todos los paquetes que pasan por el router. El router destino copia el valor del bit en los ACK que envía.
- Los hosts deciden, en función del porcentaje de paquetes que activaron el bit, cuándo achicar su ventana de congestión con un mecanismo similar a AIMD.

### RED (Random Early Detection)

- Los routers también monitorean su estado y notifican a los hosts.
- En este caso, la notificación es **implícita**: se dropean paquetes y el host reacciona ante este evento.
- Se produce antes de llegar a la situación de congestión: preferible dropear unos pocos paquetes en un momento para evitar dropear muchos más después.
- "Colabora" con el mecanismo de TCP: lo que efectivamente reduce la ventana es el mecanismo de control de TCP, disparado por el timeout de RED.
- Cuando se excede un threshold de longitud en las colas, el router dropea paquetes aleatoriamente.
- Los flujos que ocupen mayor ancho de banda tendrán más paquetes dropeados.

Detalles:

- Se calculan longitud promedio de las colas para decidir en base a congestión "estable" y no por ráfagas de tráfico.
- Si la longitud promedio supera cierto intervalo, se descartarán paquetes aleatoriamente (con probabilidad creciente a medida que crecen las colas)
- Cuando se supera el threshold superior del intervalo, se dropean todos los paquetes.




### FRED (Flow Random Early Detection)

- Si un flujo que consume mucho no baja su taza de emisión, RED empieza a droppear y termina castigando a otros flujos más pequeños.
- FRED mantiene estadísticas similares a las de RED pero por flujo. Por ende, mantiene estado y requiere más cómputo por parte de los routers.
- El objetivo es ser más agresivo con los flujos que ocupan más lugar de buffer, manteniendo distintas probabilidades.
- Por ejemplo, permite que se dropeen todos los paquetes de un flujo "bandido" (habiendo superado el threshold) pero sin afectar a otros flujos.

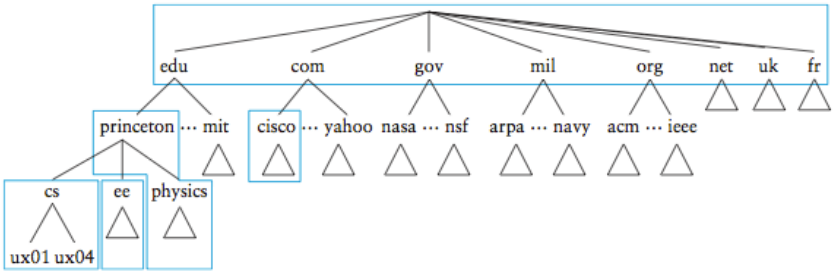
# Unidad 9: Capa de Aplicación

## DNS

El servidor de nombres es uno de los primeros servicios implementados para permitir que el resto se desentienda de las direcciones de red. En el caso de internet se usa un sistema distribuído llamado **DNS** (Domain Name System).

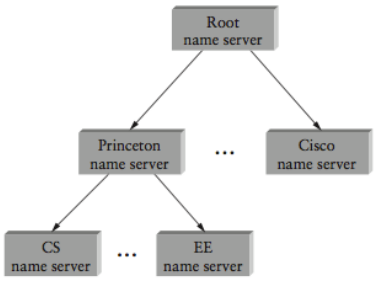
Características del **namespace**:

- Es jerárquico, procesado de derecha a izquierda
- Se puede pensar como un árbol, donde cada nodo es un **dominio**
- Un dominio es un contexto en el cuál se pueden definir más nombres




La estructura del namespace se mantiene de forma distribuída del siguiente modo:

- Jerarquía particionada en **zonas**
- Cada zona puede pertenecer a unidades administrativas independientes
- Las zonas son las unidades básicas de implementación: cada una tiene asignada dos o más **name servers** encargados de resolver los nombres de la misma.
- Cuando un host realiza un pedido a DNS, el sevidor puede dar una respuesta o derivar en otro servidor que tenga más información de la zona.
- Hay siempre más de un name server por zona para lograr redundancia.
- Un name server puede estar asignado a más de una zona.




Cada servidor contiene entradas de tipo **<Name, Value, Type, Class, TTL>**:

- **Name**: clave de la query

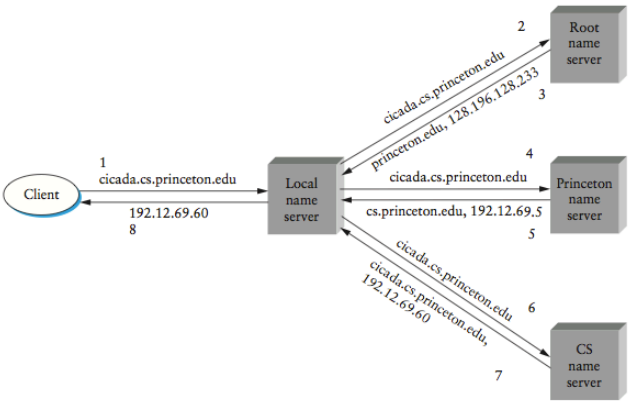
- **Value**: valor asociado
- **Type**: qué es el valor
- **Class**: punto de extensión
- **TTL**: cuánto falta hasta que expire el registro en servidores que lo tienen cacheado

El campo **type** indica cómo se debe interpretar el valor. Ejemplos:

- **A**: el valor es la dirección IP buscada
- **NS**: el valor es el domain name de un name server que puede conocer la respuesta
- **CNAME**: el valor es un alias del dominio buscado. se utiliza para agregar un nivel de indirección.
- **MX**: el valor es el domain name de un servidor que acepta mails a ese dominio

Configuración:

- Cada host debe conocer la dirección de un name server a utilizar.
- En la práctica, se conoce la dirección de un name server local que a su vez conoce algún root name server.
- Esto permite que el servidor cachee respuestas y evita configurar manualmente el root server en todos los hosts.




# Email

## Formato

RFC 822:

- Header (*From:*, *To:*, *Subject:*, etc.)
- Body
- Todo ASCII

MIME:

- Extensión para poder enviar datos
- Header lines adicionales (*MIME-Version:*, *Content-Description*, *Content-Type:*, *Content-Transfer-Encoding:*)
- Definición de content types estándar (ejemplo: *image/jpeg*, *text/plain*, *application/msword*)
- Definición de formas de encodear esos contenidos en ASCII (base64)
- Definición de content type **multitype**, que define estructura de mensajes que llevan varios content

types.

Por ejemplo, para enviar texto con archivos adjuntos, se utilizaría el content type *multipart/mixed* definido por MIME, y cada una de estas partes tendría sus propios headers que definen en tipo y encoding.

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="-------417CA6E2DE4ABCAFBC5"
From: Alice Smith <Alice@cisco.com>
To: Bob@cs.Princeton.edu
Subject: promised material
Date: Mon, 07 Sep 1998 19:45:19 -0400

---------417CA6E2DE4ABCAFBC5
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Bob,

Here's the jpeg image and draft report I promised.

--Alice

---------417CA6E2DE4ABCAFBC5
Content-Type: image/jpeg
Content-Transfer-Encoding: base64

. . . unreadable encoding of a jpeg figure

---------417CA6E2DE4ABCAFBC5
Content-Type: application/postscript; name="draft.ps"
Content-Transfer-Encoding: 7bit

. . . readable encoding of a PostScript document
```

### Transferencia

SMTP:

- Protocolo utilizado entre los **mail daemons** de cada host.
- Establece una sesión en donde se intercambian mensajes.
- Los mensajes de control y contenido de los correos son en ASCII (los correos respetando el formato de mensajes de RFC 822)
- El servidor avisa si recibe correo para cada uno de los destinatarios.

- Los mensajes atravesan un camino de **mail gateways**, que hacen store-and-forward de los mensajes.

- A diferencia de los gateways IP (routers), los mail gateways guardan en disco los mensajes y pueden llegar a reintentar el envío durante días.
- Los gateways sirven para esconder el host final de donde se lee el mail (enviando a un gateway "institucional") y para almacenar los mensajes en caso de que el host final no esté online.

### Lectura

Podrían leerse los correos con un programa corriendo localmente en el mismo equipo de la casilla de mail, pero generalmente se accede remotamente desde otro equipo. Para esto se utilizan típicamente dos protocolos:

- POP: Descarga los mensajes y se maneja una copia de los datos de la casilla.
- IMAP: Se maneja la casilla real localmente.

# Web

## HTTP

- Protocolo de comunicación entre clientes y servidores
- Stateless
- El cliente envía requests y el servidor responde

Los requests incluyen:

- Start line: definen la operación a realizar, el recurso solicitado y versión del protocolo
- Headers
- Body (suele ser vacío)

Las respuestas incluyen:

- Start line: versión del protocolo y status code
- Headers
- Body

| Operation | Description |
|---|---|
| OPTIONS | request information about available options |
| GET | retrieve document identified in URL |
| HEAD | retrieve metainformation about document identified in URL |
| POST | give information (e.g., annotation) to server |
| PUT | store document under specified URL |
| DELETE | delete specified URL |
| TRACE | loopback request message |
| CONNECT | for use by proxies |

### Mejoras de HTTP 1.1

A partir de HTTP 1.1 se soportan **conexiones persistentes**, que permiten utilizar una conexión para varios ciclos de request/response (y además mejoran el funcionamiento de los mecanismos de control de congestión de TCP).

Además de las conexiones persistentes, se puede utilizar **request pipelining** para enviar varios request antes de que llegue el primer response.

### Caching

En la web se suelen utilizar caches a distintos niveles (host, ISP, servers) para mejorar la performance y reducir la carga. HTTP define headers especiales para ayudar a controlar caches, evitando transmitir recursos si el cliente ya posee la última versión.

Por ejemplo: el header *Expires:* indica al cliente por cuánto tiempo puede guardar un recurso sin pedirlo nuevamente, y también se pueden realizar "gets condicionales" utilizando el header *If-Modified-Since:*